Demographic Visualisation App in Python

Project Scope Document Jon Minton

Project Title

Demographic visualisation App

Description

The aim of the project is to create a demographic data visualisation app that, at a minimum, implements the same core functionality as an R Shiny app I developed a number of years ago, but using Python as the primary programming language. The R Shiny app is available here, and its (very messy) codebase here. The core functionality that the Python app needs to replicate is that of allowing the user to interact with population data through a series of linked graphical displays, comprising a main display showing values in the form of a demographic landscape (Lexis surface) whose other two axes are age and year, and three subplots showing the age, period and cohort schedules through the same data (equivalent to 'slicing' the data at 0, 90 and 45 degrees). For the main display, clicking on a point on the landscape should generate or regenerate the three subplots with the specific age, period and cohort schedule selected. For the main display and subdisplays, hovering over either a voxel (main display) or position along a line (subplots) should bring up tooltips providing further information about that part of the plot.

Why

The existing R Shiny app was written along with two similar apps in a two and a half week break between jobs. (Cut from six weeks due to being called to Jury Duty :( ) The code base was written simply to see if I could implement data visualisation functionality I had been hoping to develop for some years previously, and without awareness of effective best practice for developing maintainable code, such as unit-testing, meaningful documentation, and modularisation. In developing the app in Python I hope to apply good coding practice and discipline I have learned during the Codeclan course, as well well as good design practice for improving the user experience for those using the app. (Such as more effective use of CSS for styling the app.) I see Python as an adjacent programming language to R, being both more effective and better designed as a general purpose programming language than R, having many data science packages (such as pandas, numpy, seaborn, etc) which are analogues of those in R, and which tends to have higher commercial demand than R. I am thus keen to know how to combine data science and general software development in Python in order to be able to do more in a single environment. Additionally, the interactive visualisations that are the core functionality of the app are created in Plotly, a Javascript library. Both R and Python have packages for interfacing with Plotly,

and I am keen to be able to compare between these two interfaces. Finally, the recommended front end library/framework for producing interactive data visualisation apps using Plotly is Dash, which is built on React but featuring annotations/callbacks specialised for scientific apps.

MVP Acceptance Criteria

- Must allow user to select from a range of countries/populations using real data from the Human Mortality Database
- Must display a Lexis surface using plotly based on user selection
- Must have subplots which render on on-click events ot main plot
- All plots must have meaningful procedurally generated tooltips which react to hover-over events.

Potential Extensions

- Data for Lexis surfaces are calculated/generated on-the-fly in response to user selections (e.g. when comparing two populations)
- Data are requested on-the-fly from the Human Mortality Database, either directly througn an API, or by calling R to use the HMDHFDplus package
  - Such data are cached as required.
- All plot types in the Shiny app are recreated (i.e. including comparisions between populations)
- The main plot allows Lexis surfaces to be displayed in multiple ways, including as 3D plots, contour lines and shade plots
- There are options to choose from multiple colour palettes for the display.
- The complete app is deployed, e.g. as a docker image
- A series of more conventional 2D (rather than 3D) graph options, such as showing life expectancy (conditional and unconditional) are also available.
- There are comprehensive instructions and guidance to the data and visualisation types, and how to use the app.
- The user can download both the specific data selected and the visualisations produced
- The user can resize plots (e.g. make main display larger and subplots smaller or vice versa)
- There is comprehensive documentation for each component/module within the app

Risks

- Without being able to make database request reliably, a lot of data may need to be attached with the app, which may make it slower to run and more expensive to store.
- Sending click-over events between displays may be less straightforward than in Shiny.

- Dash may turn out to be more difficult to use than initially expected (Though I have tried the brief Dash 20 minute tutorial, which did take me about 20 minutes)
- If using the HMD API, there is a risk that the database becomes unavailable at the time requests are being made. Conversely if downloading all data initially the app may be sluggish to run or expensive to host.

Exclusions

- I will not focus on demographic outcomes other than mortality and population size, such as fertility rates.
- I will only make minimal use of R where essential, and not introduce another programming language (e.g. Java) into the tech stack.

Prerequisites

- Familiarity with relevant databases and means of visualising data they contain.
- Good understanding of Python for both data science and software development.
- Some understanding of React (which powers Dash interface) and Javascript (which powers Plotly).
- Good understanding of modularisation of code, so that app modules can be developed and tested in isolation from each other.
- Some understanding of effective UX and design principles, including how to most effectively engage users of complex data visualisation apps.

Instructor Sign Off

Sign Off Date