OkCupid Demographic Analysis: GUI for filtering demographic of OkCupid dataset based on personality scales and profile information to observe corresponding probabilities of answers to personal questions

Harry Durnberger

Introduction to dataset

This project attempts to handle a very large dataset (N=68371, 2620 variables) from the online dating site OkCupid. The dataset consists of a total of 83 features, including basic profile information (gender, age, sexuality, etc.) and 50 continuous personality scales (confidence, creativity, honesty, etc.) that OkCupid calculates automatically for its users. In addition, there are categorical answers given to the top 2541 multiple-choice questions on the site. The questions are extremely varied, and many are personal or opinionated. The user is incentivised to answer the questions so that the OkCupid algorithm can match them to someone with similar answers.

Purpose of code

The purpose of the final code of this project is to provide an easy-to-use GUI to help the user filter the demographic of the OkCupid dataset and observe this demographic's probabilities of giving particular answers to a selected question, in comparison to that of the full population. This may be used as a tool for researchers or businesses to gain useful insight into a particular demographic's opinions on divisive topics. A researcher may use this information in preparation for their own investigations. A business may use this information to better understand their target market.

The program written to achieve this takes the form of a python script that is runnable in the browser as a streamlit application. The user may interact with the dataset by selecting from a range of filtering options via drop-down menus and tick boxes in the sidebar of the GUI. Relevant graphs and statistics are displayed for the user on the main page, which refresh instantly if the user changes their selections.

Initial cleaning of the dataset

The raw dataset is both huge and messy. Some features are irrelevant/uninformative, and others contain data of the wrong type (strings) for processing. For these reasons, it is important to clean the dataset before loading it to the streamlit application for filtering. 'clean_dataset.py' is a program intended for this purpose.

The main functions of this program is to remove irrelevant and useless features; binarise categorical features; and merge minority features.

The dataset is initially read into a pandas dataframe. Many features are obviously irrelevant and may be dropped immediately, e.g. 'd_astrology_sign' (using, pandas.DataFrame.drop).

However, other features are of interest but are categorical, and contain row values of the 'string' data type, e.g., 'd_religion_type', with categories: Christianity, Buddhism, Atheism, etc. For these features, 'clean_dataset.py' creates a new column for each category. This new column contains either a 1 or a 0 for each row, and is therefore binary. E.g., if someone is Christian, they will have a 1 in the 'Christianity' column, and a 0 in all other religion columns. This is achieved by looping over each categorical feature and creating a list of the unique categories of each, then looping over each unique category in turn. A column is created for each unique category and the 'numpy.where()' function is used to define the condition for which 1s or 0s are added to the correct rows.

Some features are Boolean-like, meaning their essential information can be boiled down to a 'yes/no' question. An example of this is the feature, 'd_drinks', which has five unique categories: 'Not at all', 'Rarely', 'Socially', 'Often', 'Very often'. Five new columns would be created if the previously described method was used to deal with this categorical feature. However, it would be better to condense this information into one binary column, 'Drinks often', where a 1 means the individual drinks 'Often' or 'Very often' and a 0 means they do not (or have selected one of the other three categories). This is again achieved using the 'numpy.where' function, here using multiple conditionals, depending on the number of categories chosen to condense into one.

Other features contain few majority categories and many minority categories, e.g., 'd_gender'. This is a feature that contains 107 unique categories. Just two of these categories, 'Man' and 'Woman', are attributed to ~97% of the samples. Of the other categories, there are less than 20 individual samples for each. Therefore, it is reasonable to cluster the remaining 105 minority genders into a single binary column, 'Other'. Now, there are simply three new features representing gender information; 'Man'; 'Woman'; and 'Other'.

Using the described methods, 'clean_dataset.py' drops 22 irrelevant features, and processes 10 categorical features into 31 new binary features. The cleaned dataset is written to 'ok.csv'. A list of all features of the cleaned dataset is written to 'features.txt'. Out of the newly created features, it also writes the names of groups of related features into lists and writes each of these into a list of lists and saves these to 'new_features.txt'. These files are then loaded by 'okapp.py', the demographic analysis streamlit application.

Design and function of demographic analysis application

When the 'okapp.py' app is launched, the cleaned dataset, 'ok.csv' is loaded into a pandas dataframe via the function 'load_dataset'. This single process takes ~1 minute to complete. The nature of a streamlit application is that each time an input is changed by the user, the app is refreshed, and the code is run through again. To avoid loading the dataset each time the user changes their selections, this function must be cached. This is achieved by adding the '@experimental_singleton' operator above the definition of the function. This means the

function is run once the first time it's called, and its output is cached; and if called again, even after the app refreshes, it will not run again. This operator is added to other functions that are only called once, to further save loading times.

After the dataset has loaded, the sidebar of the app will intialise with the drop-down menus, 'Select keywords:' and 'Select question number:'. The keywords drop-down is used to filter the questions via one or many user-selected keywords from a list of 11. This an important assist to the user due to the great number of questions to choose from (2541) and is achieved using the 'streamlit.multiselect' function.

The question data is stored in 'question_data.csv' and is loaded into a pandas dataframe. Upon choosing a keyword (or multiple), the associated questions and their options are displayed for the user on the main page. Each question is numbered, and the total number of questions associated with the keyword(s) is given. The questions are displayed by calling the function, 'display_questions' which loops over each row in the questions dataframe and prints the question and options via the 'streamlit.text' function.

The user may browse through the questions until they find one they wish to select. The user may select a question by choosing the corresponding question number from the dop-down menu in the sidebar. This is created using the 'streamlit.selectbox' function.

Upon selection of a question, the ID of the question from the question dataframe is used to select the corresponding question column of the cleaned OkCupid dataset. The function, 'filter_chosen_question', is then called to filter the dataset. This function filters out all question data in the dataset except for that of the chosen question. It also removes all rows that contain no data corresponding to the chosen question. This process lightens the dataset and makes future processes faster. The columns of the dataset are now simply the features plus the chosen question.

Immediately after question selection, the app refreshes and the list of questions disappears. Instead, just the chosen question and its options are displayed. Underneath, a countplot appears displaying the population counts of each question option. The total number from the population who selected each option is explicitly displayed on each corresponding bar. This is achieved by calling the function, 'plot_histogram', which, for a particular demographic (in this case the full population), defines a countplot using 'plotly.express.histogram' and plots it using 'streamlit.plotly_chart'. The options are displayed in alphabetical order, so that they line up with a second plot displayed later for the chosen demographic, to allow for easier comparison.

Underneath the countplot, a set of probabilities appear, corresponding to the likelihood of an individual of the population selecting each particular option to the question. This is achieved by calling the 'display_probabilities' function which calculates and displays the probability of each option by taking the number of individuals who selected that option and dividing it by the total number who answered the question. The option the population is most likely to select is highlighted, along with the least likely option.

Another multiple-selection drop-down menu now appears in the sidebar labelled, 'Select categories to remove'. This refers to the options of the question. Many questions in the dataset consist of two options (yes/no), but some consist of multiple. An example is a question with the options; 'Yes', 'No' and 'I'm not sure'. The user may desire to remove samples from the dataset that answered, 'I'm not sure', to focus on direct comparison between those that answered, 'Yes' or 'No'. This is achieved by looping over the user's selections and using conditional slicing to keep only rows with values not equal to the given option to remove.

Also, text appears in the sidebar prompting the user to filter the demographic. Underneath this, six drop-down menus appear, corresponding to each group of similar categorical features. One of them, for example, is labelled, 'Gender:', with options: 'Male', 'Female' and 'Other gender'. The user may select a maximum of one category from each group. This is due to the orthogonality between almost all categories within a particular group, e.g., there is no data for individuals who are in both of the categories; 'Christianity' and 'Atheism'. With each selection made, the dataset is filtered accordingly, so that only data of the chosen demographic remains. It should be noted that the user may change their mind and remove filters and the dataset will refill with the previously removed data. This is due to the processes of the program being carried out on a copy of the loaded cleaned dataset, created using 'pandas.DataFrame.copy'. This means the dataset does not need to be reloaded using 'load_dataset' each time the user chooses to "unfilter" the filtered dataset.

The 'filter_categoricals' function is used to filter the dataset in this way. This function loops over each of the chosen categories and creates a new subset of the dataset for each. These subsets only contain rows for which the particular chosen category contains a 1. All subsets are then concatenated, and duplicate rows are removed. This produces a dataframe which contains 1s in at least one of the chosen categories. The categories are looped over once more and only the rows containing 1s in *all* categories are kept. The resulting dataframe is the dataset successfully filtered to only contain samples of the desired demographic, according to the user's categorical feature selections.

Below the categorical drop-down menus in the sidebar, there are two tick boxes; 'Have kids' and 'Don't have kids'. These are created using 'streamlit.checkbox' and, when ticked, filter the dataset in a similar way to the other categorical selections.

Underneath the tick boxes in the sidebar, there are drop-down menus to select from the continuous features. These include the 50 personality scales ("traits") and other continuous features such as 'Age'. The drop-down menus are created using 'streamlit.multiselect', allowing the user to select multiple of these to filter by.

Upon selection of at least one of the continuous features, the 'percentile_range' function is called. This function provides percentile range sliders to allow the user to choose the percentile range over which to filter the traits. For a chosen continuous feature, sliders are displayed using 'streamlit.slider'; one for the lower bound; one for the upper bound. The lower and upper bound sliders initialise at value of 0 and 100, respectively. This selects the entire range over the chosen continuous feature, and so does not change the dataset. However, the user may adjust either of the sliders to choose a different percentile range. The function displays the chosen range to the user and informs them if they have chosen a "top x%" or "bottom x%" of the demographic, in terms of the chosen continuous feature. The function

outputs a matrix with each row containing the chosen continuous feature ID and corresponding upper and lower percentile values, for each chosen continuous feature.

This information is then immediately used by the 'filter_traits' function which filters the dataset according by the selected continuous features according to their associated selected percentile range. Firstly, it immediately drops all columns from the dataset corresponding to continuous features the user has not selected. It loops over all chosen continuous features. For each, it finds the maximum and minimum values of the corresponding column. Using this information, the lower and upper percentile values are calculated. These values act as boundaries. Conditional slicing is used to keep only rows with values between these boundaries. This is done for each chosen continuous feature. The final output is the successfully filtered dataset. Each time the user adjusts the sliders, these functions will be called again and the dataset will be updated accordingly.

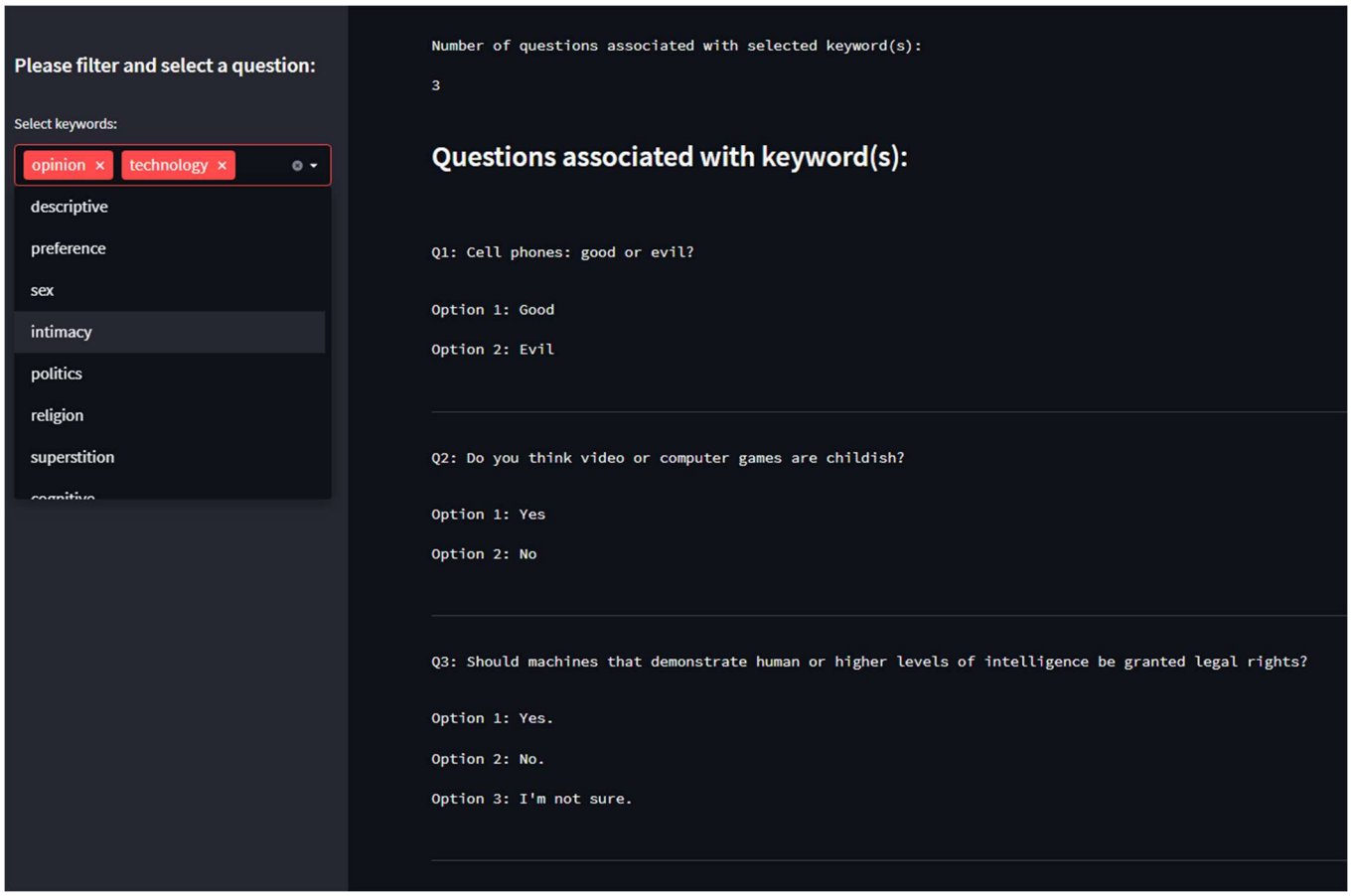
Once the demographic has been filtered in some way, calls to the 'plot_histogram' and 'display_probabilities' functions are made again. Now, a countplot and set of probabilities are instead displayed using the data from the filtered dataset, corresponding to the chosen demographic selected by the user. The user is able to compare these to those of the population, which remain above.

There is also an option below to tick a check box to reveal the raw dataframe corresponding to the chosen demographic.

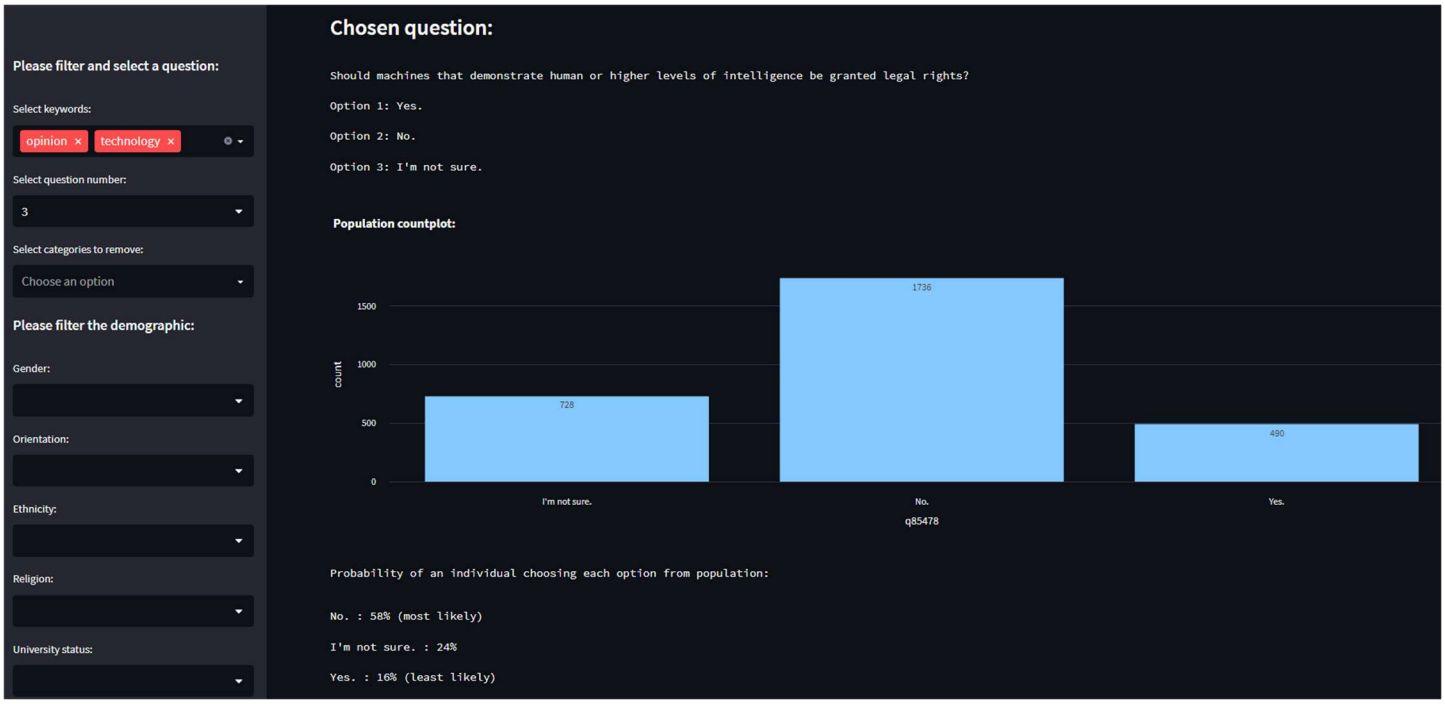
Throughout the program, error messages are added for certain scenarios for which the user must correct their selections. For example, if the user selects too many options or too niche of a demographic and the filtered dataset contains no data; the error message, "No data for chosen demographic." is displayed.

Example use of demographic analysis application

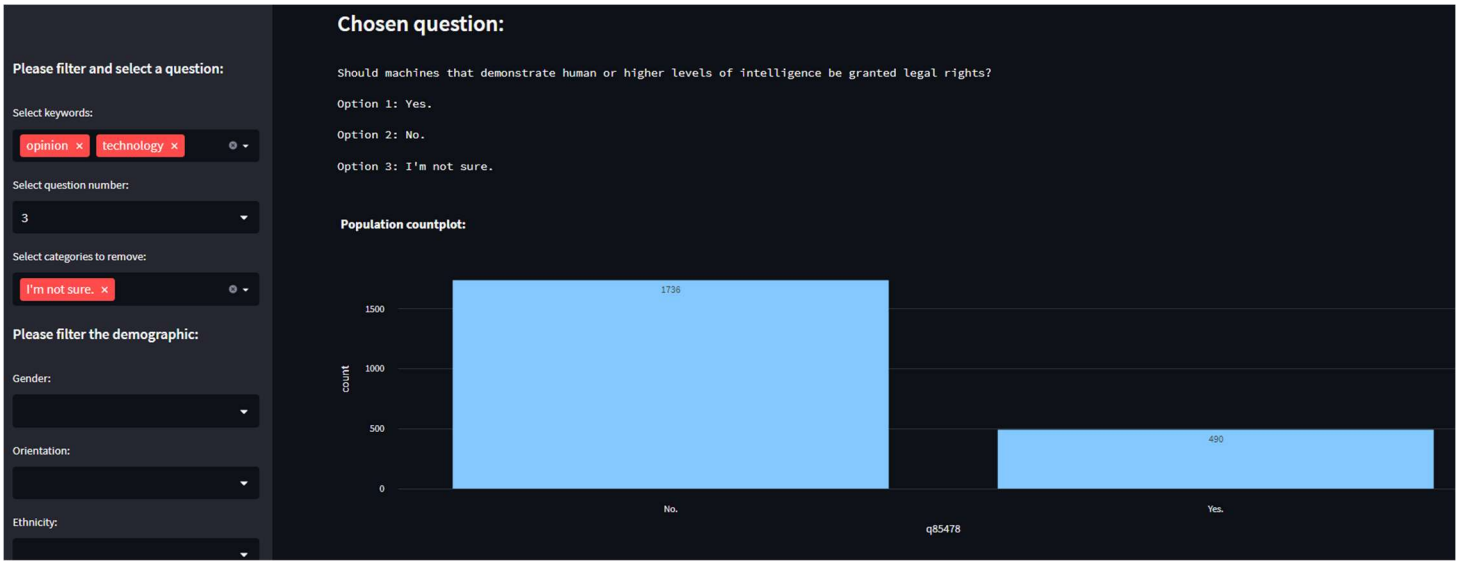
The user selecting from the list of keywords:



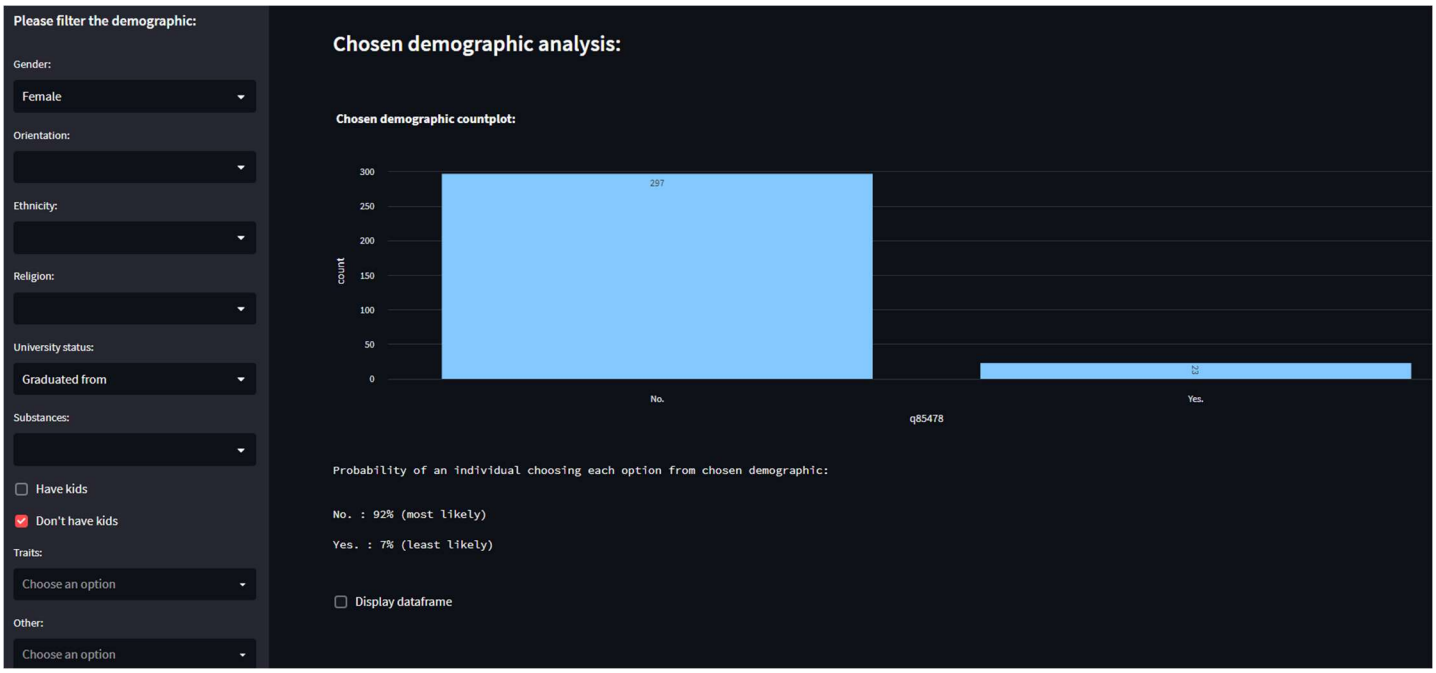
After selection of a question:



After removal of the ‘I’m not sure.’ option:

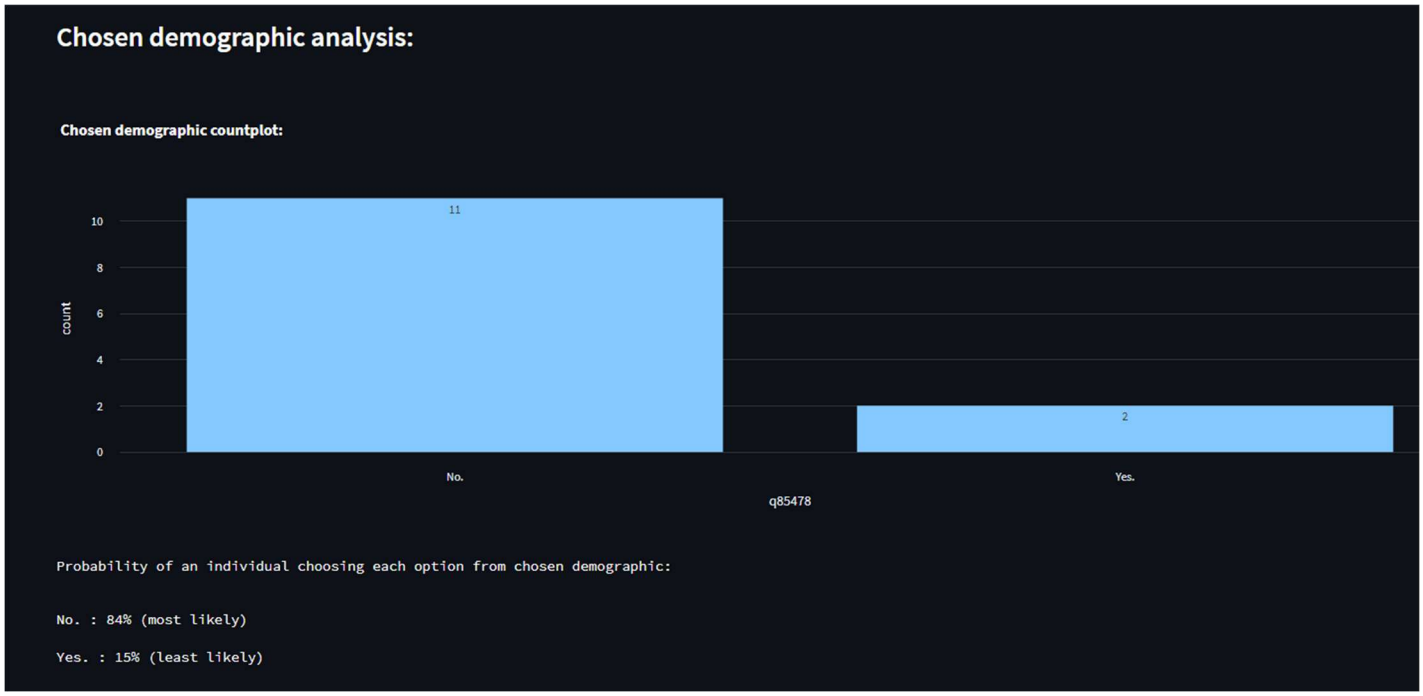


After selecting from a few of the categorical features:



After selecting some continuous features and adjusting their percentile sliders:

Final filtered demographic analysis:



Scope for improvements

One bug not addressed in the code is the fact that the probabilities displayed after removal of an option do not reflect their true values. This is due to the calculation being carried out on the filtered dataset, after all rows containing the options to remove have been removed. This bug could be addressed by calculating the probabilities prior to removing undesired options or saving the information to a variable that is then used later to calculate the probabilities.

There is also potential for extra features to be added. Rather than selecting a particular question to analyse, an option could be added to consider how likely individuals in a demographic are to also be members of a selected category. For example, one could observe how likely members of a demographic are to have graduated from university/college. This could be expanded to observing how likely members of a demographic are to be situated in a particular percentile range of a personality trait, e.g., in the top 10% of the population in confidence.