Project Objectives

This is a partner project that involves creating functions to query an API and summarize and display data.

You **should not use generative AI** for any part of this project other than for debugging programs. This will be strictly enforced.

Setting Things Up

Creating the Repo

The first step is for one group member (you two pick) to create a github repo and link it with an RStudio project. Push up those changes so the .Rproj file exists. By using an R project you can stick to using relative file paths for everything. Next, go to the repo **settings** on github and add the *other* group member as a collaborator (you'll need their github account). That group member then needs to accept the membership. This gives everyone access to push changes up to the repository. **All project work should be done within this repo.**

Repo Settings

On your github project repo you should go into the settings and enable github pages. You'll output your files to a docs folder, similar to how we've done our homework assignments.

Collaboration Workflow

You and your partner should collaborate on the project. You should set up a list of tasks and timelines/milestones with flex time.

- I know some people tend to just try to do all the work for the project themselves.
- Sometimes people like to split things up so that one person does all the coding and one person does all the writing/narrative.

If either of these are done, both people in the group will lose credit. This should be a collaboration. We will be checking the commits on the repo to make sure both people are contributing substantially to the coding aspects of the project.

With this in mind, commit often! Don't work a ton locally without pushing things ups. We want to see the work develop on github. Again, if we can't see the contributions via github, both members may lose credit.

Please reach out early if the agreed upon milestones/work are not being completed by your partners.

For collaborating, you have two options:

- Both work on the same (main) branch (the easiest solution but least like real life). If this is the case, your workflow may look like the following:
  - Each time you go to work on the project, you pull down any of the latest changes (`git pull`)
  - When you are satisfied with some contributions, add, commit, and push your work up to the main branch. (Remember version control is part of the system so you can always roll back changes.)
  - There may occasionally be merge conflicts that have to be dealt with. This can be done with the `Git` tab in RStudio. Let us know if you are having issues with conflicts that you can't resolve!

- You each work on your own branch (or one on the main branch and one on their own branch). Notes about that process:
  - A branch can be created on github.com or via the command line with `git checkout -b new_branch_name`. Any work you then do will be on that branch.
  - In order to push up the new branch and any changes to github, you can do your usual add and commit steps. The first time you do a push step, you have to add some extra code: `git push --set-upstream origin branch_name`
  - You can switch between branches with `git checkout branch_name`
  - You can merge in your branch's changes to the `main` branch by
    1. Checking out the main branch: `git checkout main`
    2. Merging your changes in: `git merge branch_name`
  - As with the other workflow, you may need to resolve merge conflicts. We're happy to help out if you get stuck on that!

Overall Goal

Our goal is to write functions that will manipulate and process data sets that come from a census API. We'll create *generic* functions to automatically summarize and plot certain returned data. Lastly, we'll write up a document via quarto (just like we did with the homework) to describe our thought process and give examples of using the functions. This document (web site) is what you will ultimately turn in.

You must have a narrative throughout the document. This is often a place where people lose points on their first project. A narrative means you should have a story or thread that flows throughout the document. Things like an introduction about what you are doing, discussion of what you are about to do and why/how, etc. Don't lose these easy points!

Data

We are going to query the Public Use Microdata Sample (PUMS) Census API. This API gives pseudo *person-level* data. There is no need to obtain an API key but you can if you'd like.

Now there is an overwhelming amount of information about how to deal with census data and the different APIs there are. It is incredibly easy to go down a rabbit hole with older information or alternative ways to do things. **I hope to give you all the resources/links you need below!** Feel free to Google other things but you might get lost.

- Information about the PUMS API
  - We aren't going to worry about Puerto Rico data so focus on the basic PUMS information
  - Pay special attention to the example function calls for each year. For instance, Example Call: `api.census.gov/data/2022/acs/acs1/pums?get=SEX,PWGTP,MAR&SCHL=24&key=YOUR_KEY`
- The list of variables for the 2022 version is really useful
  - This page gives the values the variables can take on if you click on their name.
  - You can change the year in the URL to get a data dictionary for other years
  - Generally, ignore 'allocation flag' variables. This is something about imputing data or not.
- We'll want to allow the user to choose certain variables to return and choose certain conditions on what is returned (specific subsets)
- More examples of API calls are here
- A menu based explorer where you can see API calls (go to the 'download' tab and it is at the bottom)

Data Processing

Alright, so we'll want to write a function to query this API and return a nicely formatted tibble. When we query the API via our usual process (see notes), the data comes out pretty decently!

- Start by getting the usual process to work with a given URL
- Next, write a *helper* function to take what is returned by `GET()` and turn it into a nice `tibble`.
- Write a function to query the API that allows the user to change the following items:
  - Year of survey (2022 as default). Only a single value here
    - Check that a valid value was given (number between 2010 and 2022)

  - Specify the numeric variables to be returned (AGEP and PWGTP as default). PWGTP should always be returned.
    * Options for the user should be AGEP, GASP, GRPIP, JWAP (time), JWDP (time), and JWMNP
    * Your function should turn variables into numeric values or time values (use the middle of the time period) where appropriate.
    * One numeric variable other than PWGTP must be returned
  - Specify the categorical variables to be returned (SEX as default).
    * Options for the user should be FER, HHL, HISPEED, JWTRNS, SCH, SCHL, and SEX
    * Your function should turn variables into factors with appropriate levels, where appropriate
    * One categorical variable must be returned
  - (Check that the variables asked for are in this set of variables)
  - Specify the geography level: All, Region, Division, or State (with the default of All)
    * Check that the value specified by the user is one of the above values
  - An optional argument to subset the data (this subsetting should be on the API call itself, not on what is returned - see the examples in the links above)
    * User should be able to specify specific Regions, Divisions, or States for this part (and only those specified geography levels would be returned)
    * How you allow them to specify this is up to you (you'll have to parse it appropriately)

- Lastly, write a function that allows the user to specify multiple years of survey data (and all the other options above)
  - This function should call your single year function as many times as needed and then combine the data into one final tibble (a year variable should be included in this final tibble)

Obtaining Person Level Records

Lastly, we need to process the data in our tibble appropriately. The `PWGTP` variable actually represents the number of (people) observations there should be for a particular row.

We won't do this, but we could create a new tibble with each row replicated `PWGTP` times. (This would create very large data frames in some instances and processing would be slow!)

Writing a Generic Function for Summarizing

First steps We briefly discussed the ideas around object oriented programming and method dispatch. That is, we discussed why `plot(iris)` and `plot(exp)` give different types of plots. They look at the `class` of `iris` and the `class` of `exp` and the appropriate plotting function is called.

```
#Run these in your console
plot.function #what is used for a class = function
getS3method("plot","data.frame")  #what is used for a class = data frame
```

The generic function is `plot()`. Notice that it calls `UseMethod("plot")` and that is how the appropriate plotting function is determined.

```
plot
```

```
## function (x, y, ...)
## UseMethod("plot")
## <bytecode: 0x00000000131a0430>
## <environment: namespace:base>
```

Great, well we can add our own class to our census data for summarizing purposes.

- **Go back to your function to create the data and make your tibble have an additional class of 'census'.**
  - Something like:

```
class(your_tibble) <- c("census", class(your_tibble))
```

Now we can write our own custom `summary` function for these! We simply write our function as follows:

```
summary.census <- function(...){...}
```

For the `census` summary method, let's write a function that produces means and standard deviations for our numeric variable(s) and counts for our categorical variable(s).

- This function should take three arguments: the tibble with class census, the numeric variable(s) to summarize, the categorical variable(s) to summarize.
- By default, it should summarize all numeric variables (other than `PWGTP`) and all categorical variables in the tibble. However, the user should be able to specify the variables they'd like to summarize if they'd like.
  - To find a sample mean for a numeric variable, we would do something like this:

```
sum(numeric_vector*weight_vector)/sum(weight_vector)
```

  - To find a sample standard deviation:

```
sqrt(sum(numeric_vector^2*weight_vector)/sum(weight_vector)-sample_mean^2)
```

  - Return the values as a named list

Test out this function by running `summary(_your_census_tibble_)` on something you've returned from your census API function.

Similarly, let's create a generic `plot()` function for a `census` class tibble. Require the user to specify one categorical variable and one numeric variable for plotting purposes.

We haven't covered plotting yet so I'll provide the generic code for it below:

```
library(ggplot2) # put this in your setup code chunk
ggplot(_your_census_tibble_or_modification_,
       aes(x = get(cat_var), y = get(num_var), weight = PWGTP)) +
  geom_boxplot()
```

- The `weight = PWGTP` argument accounts for the weights when making the plots.
- I have `get(cat_var)` and `get(num_var)` within the function as I'm assuming you'll pass the column names as strings.

Web Page

Your web page should have a narrative going through your process of creating the above functions and testing them.

You should have a section at the end where you investigate something interesting from the data using your API function and plotting/summarizing functions.

Submission

For the submission link, you should give the link to your nicely rendered webpage!

Rubric for Grading (total = 100 points)

Item	Points	Notes
Data Processing Functions	35	Worth either 0, 2.5, 5, ... 35
Generic Functions	35	Worth either 0, 2.5, 5, ... 35
Narrative/Introduction/End Example/Good Coding/etc.	30	Worth either 0, 2.5, 5, ... 30

Notes on grading:

- For each item in the rubric, your grade will be lowered one level for each each error (syntax, logical, or other) in the code and for each required item that is missing or lacking a description.
- You should use Good Programming Practices when coding (see wolfware). If you do not follow GPP you can lose up to 50 points on the project.
- If the webpage doesn't work correctly you can lose up to 30 points.