Problem Set 9

Overview:

In this problem set, you will be working on a shared repository with your homework group. We encourage you all to communicate with your group by creating issues on your shared repository.

We will not be asking for your git commands and output – you can simply use your command line interface. You are required to do your work in branches. You will only need to submit (via pushing to your Github repository) your R script and any data files you create.

The purpose of this problem set will be to give you more practice on the skills you learned in the previous weeks on writing loops and to combine these skills with writing conditional statements and functions. This problem set is based off of Ben Skinner's script to batch download Integrated Postsecondary Education Data System (IPEDS) files and script to add variable and value labels to ipeds data. The goal is to improve our code from the previous problem set. We will create a function to check whether a directory exists in our repository and create a directory if it does not exist. We will also create a function that downloads three types of files (data, dictionary, stata program) into their respective directories for each year. The last section is a **bonus question** and it is entirely **optional**. In the bonus question we are asking you to create multiple smaller functions to assign variable and value labels for each data file.

Part I: Setting up

Like the previous week, create a new private GitHub repository here for your group called `<team_name>_ps9` and initialize it with a `.gitignore` file.

1. Download and save the ipeds_file_list.txt file in your repository. Make sure it is in the main repository folder. Only one person in your group needs to add this file to the repository using the **master branch**. Once this file is added, everyone on the team should clone the repository to their local machines.
2. Create a new RStudio Project for this repository and complete all your work on a branch called `dev_<last_name>`.
3. Download and save the `lastname_script.R` file under "Weekly readings and assignments" >> Week 10.
    - Rename this file to your lastname and save it in your repository `<team_name>_ps9`.
4. Open the R script in your R Studio and add your header information on lines 1-8.
    - On line 19 change the name of `data_lastname` to your lastname (e.g., `data_martin`). This will be the directory where you will save all your data files.
    - Run lines 19-23 to create this directory
    - Then, run lines 26-33 to create the file path objects that will be the sub-directories within your `data_lastname` folder where you store specific file types (you will create these sub-directories later)
5. Run the lines given under `Part I` of `<lastname>_script.R` to create objects for later use (these are the same code from PS8).

Part II: Writing function to create directory

1. Run the lines given under `Part II` of `<lastname>_script.R` that uses `dir.exists()` to check if the file paths created in the previous section exist. Remember that only `data_lastname_dir` exists so far. Note that `dir.exists()` returns a logical, which can be used as a condition.
2. You'll be writing a function to create a new directory only if it does not already exist. But before doing that, try performing the task outside of a function.
   - Write an if-else statement that checks if `data_lastname_dir` exists
   - If `data_lastname_dir` exists, use `writeLines()` to print `"Already have directory: <data_lastname_dir>"`
   - Else, use `writeLines()` to print `"Creating new directory: <data_lastname_dir>"` and `dir.create()` to create the directory

   If you try running your if-else statement, the `if` block should execute because you've created the `data_lastname_dir` directory in the previous section.
3. Now, try refactoring your code from the previous step into a function.
   - **Function name**: `make_dir()`
   - **Function argument**: `dir_name`
   - **Function body**: Contains an if-else statement that checks if `dir_name` exists and create the directory if not (hint: base your code off the previous step)
4. Call your function with `data_lastname_dir`. You should get a similar output as Step 2 where it prints that the directory already exists.
5. Call your function with `data_dir`, `dict_dir`, and `stata_prog_dir`. You should see that these sub-directories will get created inside `data_lastname_dir`.

Part III: Writing function to download data

1. Run the lines given under `Part III` of `<lastname>_script.R` to download `HD2018.zip` into your `data_dir` (these are the same code from PS8). Remember that `data_dir` is one of the 3 sub-directories inside `data_lastname_dir`.
2. Following the instructions in `<lastname>_script.R`, update `data_url` and `data_destfile` to download the following.
   - Download `HD2018_Dict.zip` inside `dict_dir`
   - Download `HD2018_Stata.zip` inside `stata_prog_dir`

   Note that we are downloading each of the 3 file types (data, dictionary, stata program) into their respective directories inside `data_lastname_dir`.
3. Next, you'll be writing a function to download a data file only if it does not already exist. But before doing that, try performing the task outside of a function.
   - Write an if-else statement that checks if `data_destfile` exists
   - If `data_lastname_dir` exists, use `writeLines()` to print `"Already have file: <data_destfile>"`
   - Else, use `writeLines()` to print `"Downloading file: <data_destfile>"` and `download.file()` to download the file at `data_url` into `data_destfile`

   At this point, your `data_url` and `data_destfile` should correspond to the stata program file since that's what was last assigned. If you try running your if-else statement, the `if` block should execute because you've already downloaded `HD2018_Stata.zip` into the `stata_prog_dir` directory in the previous step.
4. Now, try refactoring your code from the previous step into a function.

- **Function name**: `download_file()`
- **Function argument**: `dir_name`, `file_name`, `file_suffix`
- **Function body**: (hint: base your code off the previous steps)
  - Create `data_url` object based on `file_name` and `file_suffix` inputs
  - Create `data_destfile` object based on `dir_name`, `file_name`, and `file_suffix` inputs
  - Include an if-else statement that checks if `data_destfile` exists and download the file at `data_url` if not

5. Following the skeleton code given in `<lastname>_script.R`, call the `download_file()` function for the 2018 and 2017 HD files.

6. Write a for loop to loop over all years of data in the `hd` vector and download the three types of files (data, dictionary, stata program) into their respective directories for each year.

Part IV: Labelling dataframe variables and values [BONUS 10 pts]

In this bonus section, you'll work on labelling the variables and values in the IPEDS data. Start by running the line `library(labelled)` at the top of the `Part IV` in `<lastname>_script.R`, then follow along the comments and instructions below. Remember that for each step, you can try performing the task outside of a function (i.e., test your code using data for one specific year), then try adapting it to work inside a function.

1. First, write a for loop to unzip all the downloaded files from each of the 3 sub-directories. This is similar to Loop 3 from PS8.

   You are given the vectors `file_suffixes` and `file_dirs` to help with this step. You can access the corresponding elements from `file_suffixes` and `file_dirs` by their indices.

2. Now, recall Loop 4 from PS8. You'll see a similar loop at the very end of `Part IV` in `<lastname>_script.R` that loops through the `hd` vector. The first and last line in the loop body should look familiar. The part of Loop 4 where you read in the CSV data and changed all column names to lowercase will now be moved to the function `csv_to_df()` (*Don't worry about the other commented lines of code for now*).

   - **Function name**: `csv_to_df()`
   - **Function argument**: `file_name`
   - **Function body**: (hint: base your code off your Loop 4 from PS8)
     - Use `read_csv()` to read in the CSV file called `file_name` in `data_dir` and assign to `df`
     - Change the column names in `df` to lowercase
     - Return `df`

   When you finish writing the `csv_to_df()` function, try running the for loop and you should get the same results as Loop 4 from PS8 (i.e., dataframe objects should be created for each HD file).

3. The next step is to read in the variable and value labels from the Stata `.do` files located in `stata_prog_dir`. If you try opening the `.do` files, you can see Stata commands for labelling.

   - **Variable labelling syntax**: `label variable <col_name> "<var_label>"`
     - `<col_name>`: The variable, or dataframe column name (e.g., `stabbr`)
     - `<var_label>`: The variable label (e.g., `State abbreviation`)
   - **Value labelling syntax**: `(*)label define label_<col_name> <val_name> "<val_label>"`
     - `*`: There *may or may not* be an asterisk at the start of the command
       (a) An `*` indicates that the value is of type `character`
       (b) No `*` indicates that the value is of type `numeric`
     - `<col_name>`: The variable, or dataframe column name (e.g., `stabbr`)
     - `<val_name>`: A value in `<col_name>` (e.g., `AL`)
     - `<val_label>`: The label for `<val_name>` (e.g., `Alabama`)

   You will write a function `get_stata_labels()` that reads in the `.do` files and parses out these labels.

- **Function name**: `get_stata_labels()`
- **Function argument**: `file_name`
- **Function body**:
  - Use `readLines()` to read in the `.do` file called `file_name` in `stata_prog_dir` and assign to `stata` (hint: see code in `Part I` of `<lastname>_script.R` for similar example)
  - Create object called `var_labels` to store variable labels:
    - Use `str_subset()` to subset `stata` to only include lines with variable labelling commands
    - Use `str_match()` with the following capturing groups and save the resulting matrix in `var_labels` (hint: use parenthesis `()` in your `pattern` argument for capturing groups)
      - 1st capturing group: `<col_name>`
      - 2nd capturing group: `<var_label>`
  - Create object called `val_labels` to store value labels:
    - Use `str_subset()` to subset `stata` to only include lines with value labelling commands
    - Use `str_match()` with the following capturing groups and save the resulting matrix in `val_labels`
      - 1st capturing group: `(*)?` (hint: *This is to capture the `*` that may or may not be at the start of the Stata command. If there is no `*` matched, the capturing group returns `NA`.*)
      - 2nd capturing group: `<col_name>`
      - 3rd capturing group: `<val_name>`
      - 4th capturing group: `<val_label>`
  - Return the 2 matrices, `var_labels` and `val_labels`, as a named list (Since R cannot return multiple objects, we need to combine them in a list):
    - `list(var_labels = var_labels, val_labels = val_labels)`

When you finish writing the `get_stata_labels()` function, you can uncomment the line `stata_labels <- get_stata_labels(file_name)` inside the final for loop. `stata_labels` will be the list containing the matrices, `var_labels` and `val_labels`.

4. Write a function `add_var_labels()` that will add variable labels to the dataframe.

- **Function name**: `add_var_labels()`
- **Function argument**: `df`, `var_labels`
- **Function body**:
  - Loop through each row in the `var_labels` matrix by index (hint: use `nrow()` to create the sequence)
  - Recall that the 1st capturing group (`<col_name>`) is in the 2nd column of the matrix and the 2nd capturing group (`<var_label>`) is in the 3rd column (hint: use `[[<row>, <col>]]` to access specific cells in the matrix by index)
  - To label each variable (i.e., dataframe column): `var_label(<df_col>) <- <var_label>`
  - Return the labelled `df`

When you finish writing the `add_var_labels()` function, you can uncomment the line `df <- add_var_labels(df, stata_labels$var_labels)` inside the final for loop.

5. Write a function `add_val_labels()` that will add value labels to the dataframe.

- **Function name**: `add_val_labels()`
- **Function argument**: `df`, `val_labels`
- **Function body**:
  - Loop through each row in the `val_labels` matrix by index (hint: use `nrow()` to create the sequence)
  - Recall that the `<col_name>`, `<val_name>`, and `<val_label>` are in the 3rd, 4th, and 5th column of the matrix, respectively (hint: use `[[<row>, <col>]]` to access specific cells in the matrix by index)
  - The 2nd column of the matrix (i.e., 1st capturing group) contains either `*` or `NA` depending on if there was a match for an `*` before the Stata command. Recall that an `*` indicates that the value is of type `character` and no `*` means it is of type `numeric`.

    - When labelling values, the type of `<val_label>` must match the intended value type. Since `<val_label>` is always `character` because it is parsed from the `.do` file, you'll need to convert it to `numeric` if there is no `*`. (hint: use an `if` statement, `is.na()`, and `as.numeric()`)
  - To label each value: `val_label(<df_col>, <val_name>) <- <val_label>`
  - Return the labelled `df`

  When you finish writing the `add_val_labels()` function, you can uncomment the line `df <- add_val_labels(df, stata_labels$val_labels)` inside the final for loop.

6. Run the loop! You should end up with labelled dataframes for each year of HD data.

Part V: Wrapping up

1. How much time did you spend on this problem set? Write your response as a comment in `<last_name>_script.R`.
2. Add your `<last_name>_script.R` file to the `dev_<last_name>` branch and make a commit. Switch back to the `master` branch and merge in your `dev_<last_name>` branch. Push your work to the remote.