Problem Set #10

03/20/2023

Grade: /30

Overview:

In this problem set, you will be using regular expressions to parse data on off-campus recruiting events. You'll get some practice using a variety of functions from the `stringr` package to work with strings. There will also be a bonus problem (Part IV) at the end of the problem set, where you get a chance to apply regex to add variable and value labels to the HD datasets you downloaded from problem set 7. This part is **completely optional** (extra credit), but we encourage you all to try it out if you have time!

As usual, the requirements for this problem set are:

- You must create a separate branch (e.g., `dev_ozan_jaquette`, `dev_oj`) to do your work on
- Your `dev` branch must be pushed to the remote so we can see it
- You will turn in your files on the `main` branch (see final section), meaning you will need to merge in your work from the `dev` branch

Part I: Setting up repository

/1

1. Have one member of your group create a new private GitHub repository here called `ps10_team_name` (fill in your team name). Make sure to initialize the repo with a `README.md` as well as a `.gitignore` file with the R template.

   The member who created the private repository should also give the other group members access to the repository by going on Settings > Manage Access > Invite teams or people. Then, all members of your group should clone this repository to your computers. Don't forget to change directories to where you want to save the repo before cloning it.

/1

2. Create a new **RStudio Project** for this repository. Add this `.Rmd` file to your project directory and rename it as `ps10_lastname_firstname.Rmd` (fill in your name). In your **RStudio Terminal**, create a new branch with your name or initials in the branch name (e.g., `dev_ozan_jaquette`, `dev_oj`) and use that branch to do all your work on.

/1

3. Download the **Problem set R script template** available under the **Syllabus & Resources** section of the class website (or click here). Rename the downloaded `ps_template.R` to `ps10_lastname_firstname.R` (fill in your name) and save it inside your project directory.

   Open the R script and fill out your information in the header on top. Then, load the `tidyverse` library.

/1

4. Create a folder called `data` in your project root directory, then create a file path object called `data_dir` in your R script that points to this folder.

Part II: Parsing event locations

/1

1. First, use `readRDS()` and `url()` to load in the recruiting events data from the URL: `https://github.com/anyone-can-c` Save the data to an object called `events`.

/1

2. You'll start by parsing the event location information from this dataset. Use `print()` and `writeLines()` to print out the `address` variable of `events` to see how it looks. You should see that each event has a street address, city, state, and zip code.

/2

3. Use `str_view()` to help you write a regular expression that contains 4 capturing groups for matching the following items from the `address` variable:

   - 1st capturing group: Street address
   - 2nd capturing group: City
   - 3rd capturing group: 2-letter state code
   - 4th capturing group: 5-digit zip code (do not include the 4-digit extension if there is one)

/1

4. Now, plug the regular expression you wrote in the previous step into `str_match()` to obtain a character matrix containing the matches. Save this object as `event`. The 1st column of the matrix should contain the full match, and the next 4 columns contain matches for each capturing group.

/2

5. You will now add the following 4 columns to the `events` dataframe, which should contain the respective information from `loc_matches`:

   - `event_address`: Street address
   - `event_city`: City
   - `event_state`: 2-letter state code
   - `event_zip`: 5-digit zip code

   Hint: Assign the respective column of the `loc_matches` matrix for each new column in `events`.

Part III: Parsing event date and time

/2

1. In this section, you will be parsing the date and time information for the events. First, write a regular expression for the `date` column that contains 3 capturing groups for matching the following items:

   - 1st capturing group: Month
   - 2nd capturing group: Day
   - 3rd capturing group: Year

   Use both `str_view()` and `str_match()` to help you write your regex and visualize your matches.

/2

2. Using `str_replace()` and your regular expression from the previous step, add a new column to `events` called `event_date` where each element contains the date in the format: `YYYY-MM-DD`

   Hint: Use the `replacement` argument in `str_replace()` and backreferences to format the date and assign it to the `event_date` column in `events`.

/2

3. Now, write a regular expression for the `time` column that contains the following 3 capturing groups:

   - 1st capturing group: Hour
   - 2nd capturing group: Minute
   - 3rd capturing group: AM/PM

   Use `str_match()` to obtain a character matrix of your matches and save it to `time_matches`.

/1

4. Add the following 3 columns to the `events` dataframe, which should contain the respective information from the `time_matches` character matrix:

   - `hour`: Hour
     - Use `as.double()` to convert the hour to double type before assigning it as the `hour` column in `events`. This will allow us to perform arithmetic operations on it in the next step.
   - `minute`: Minute
   - `ampm`: AM/PM

/3

5. Let's turn the time into 24-hour clock instead of 12-hour clock using AM/PM. Create a new variable called `hour24` to the `events` dataframe that will be the 24-hour clock version of `hour`.

   Hint: If the hour is a PM hour and it is not 12 PM (noon), add 12 to the hour (e.g., 2 PM should become 2 + 12 = 14). This step does not involve regex. You could use an if_else statement inside your mutate() function.

/2

6. Use `r` to pad the `hour24` variable you created in the previous step so that it is always 2 digits long (e.g., `8` would become `08`, etc.)

/3

7. Now, add the following 2 variables to the `events` dataframe:

   - `event_time`: 24-hour clock time in the format: `HH:MM:SS` (second can be `00`)
   - `event_datetime`: Date and time in the format: `YYYY-MM-DD HH:MM:SS`

/1

8. Finally, select only the following variables from `events` and save it to a new object called `results`: `event_datetime, event_date, event_time, event_location, event_address, event_city, event_state, event_zip`

/1

9. Write your `results` object to a CSV file named `events_<your_name>.csv` (fill in your name or initials). Save the file inside your `data_dir`.

/15

This part is **optional** and will be extra credit. If you choose to complete this, you will be doing this part in your problem set 7 repository using the HD data files you've downloaded. Click here to download the script you will be using, which contains some pre-written code. Create a new branch in your PS7 repo to do your work on and push your script there (no need to submit to `main`). If you do complete this bonus section, paste a link to your script below.

Link to script: https://github.com/anyone-can-cook/ps7_huang_hui/blob/dev_hh_bonus/ps10_bonus.R

1. First take some time to look through the provided code and get a general sense of what it is doing. Here are the main components:

   - **The `csv_to_df()` function**: Reads in the HD CSV file and returns the dataframe
   - **The `get_stata_labels()` function**: Reads in the HD Stata `.do` file, extracts the variable and value labels, and returns the labels
     - **Note**: You will implement this function, as described below
   - **The `add_var_labels()` function**: Uses the variable labels you extracted from `get_stata_labels()` to label the HD dataframe
   - **The `add_val_labels()` function**: Uses the value labels you extracted from `get_stata_labels()` to label the HD dataframe
   - There is a **for loop** at the very end of the script that calls these functions to read in and label your HD data. It won't run properly until you've implemented the `get_stata_labels()` function.

2. Paste your `files` object from Part II, Q1 of problem set 7 where indicated in the provided script.

3. The main task is to implement the `get_stata_labels()` function where indicated in the script. The goal is to read in and extract the variable and value labels from the Stata `.do` files you've downloaded. If you try opening any of the `.do` files, you can see following 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`)

Your `get_stata_labels()` function will be reading in the `.do` files and parsing out these labels, as described below. We encourage you to try completing this task first using a single `.do` file outside of the function. Then, you can incorporate your code into the body of this function.

- **Function name**: `get_stata_labels()`
- **Function argument**: `dir_name`, `file_name`
- **Function body**:
  - Use `readLines()` to read in the `.do` file called `file_name` in `dir_name` (these are your function inputs) and assign to an object called `stata`
    * *Hint*: The syntax is: `readLines(<path/to/do_file>, encoding = 'latin1')`
  - 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 result in `var_labels`
      · 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 result in `val_labels`
      · 1st capturing group: `*` (*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)

When you finish writing the `get_stata_labels()` function, you can run the final loop provided in the script. The line calling your function is `stata_labels <- get_stata_labels(stata_dir, file_name)`. As seen, it is passing in the `stata_dir` and the `file_name` of the HD file it is looping over. The returned labels from your function is stored in `stata_labels`, which are then passed to `add_var_labels()` and `add_val_labels()` to label the dataframe.

4. Nice job! To check that your dataframes are properly labelled, you can use the following functions:

```
var_label(hd2019$stabbr)
val_labels(hd2019$stabbr)
```

Part V: Create a GitHub issue

/2

- Go to the class repository and create a new issue.
- Please refer to rclass2 student issues readme for instructions on how to post questions or things you've learned.
- You can either:
  - Ask a question that you have about this problem set or the course in general. Make sure to assign the instructors (@ozanj, @xochilthlopez, @joycehnguy, @augias) and mention your team (e.g., @anyone-can-cook/your_team_name).
  - Share something you learned from this problem set or the course. Please mention your team (e.g., @anyone-can-cook/your_team_name).
- You are also required to respond to at least one issue posted by another student.
- Paste the url to your issue here: https://github.com/anyone-can-cook/rclass2_student_issues_w23/issues/443
- Paste the url to the issue you responded to here: https://github.com/anyone-can-cook/rclass2_student_issues_w23/issues/438

Knit to pdf and submit problem set

Knit to pdf by clicking the "Knit" button near the top of your RStudio window (icon with blue yarn ball) or drop down and select "Knit to PDF"

You will need to submit both the `.Rmd` and `.pdf` files, as well as your `ps10_lastname_firstname.R` and anything inside your `data/` folder. You should commit these items on your `dev` branch as you work through this problem set. In the end, merge your `dev` branch into `main` and then push to the remote `main` branch to submit.