REDD+ Specification Curve Repository Design Document

SPIRES Lab @ Yale School of the Environment

December 2023

SPIRES

Study of Politics and Institutions with Remote Environmental Sensing

1. Overview

The goal of the REDD+ Specification Curve project is to perform a more thorough analysis on the recent West 2022 and West 2023 papers. To this end, we aim to run matching and synthetic controls causal inference techniques (the two used in the West papers) using all plausible combinations of input parameters and analyze any differences this may have on the reported treatment (REDD+ project) effects. We will use a specification plot to visualize all these combinations and their impacts. The bulk of the project involves setting up infrastructure to not only allow for adding more causal inference methods to test, but also adding more project sites to increase the sample size.

2. Design Philosophy

When building the code base, we stressed the importance of having a convenient framework for adding on methods, from any contributors—not just the ones who are familiar with the code base. Thus, we split up the pipelines for adding methods and adding project sites. That way, contributors only need to work in one folder to perform their specific task (i.e. adding a project or new method).

3. Project Layout

Within the repository, the first main split is between code and data, where the data folder contains all the processed and raw data that the code folder uses and outputs to.

Data

1. **Raw**
   In general, any data downloaded directly from a source goes here. There are folders for each project, where the respective data goes.

   Example raw data includes KML project boundary files downloaded straight from Verra registry and CSVs from Google Earth Engine exports.

2. **Processed**

Processed data includes anything data that has been modified. It can, in effect, be thought of as the “save” destination folder for data after you perform some operations on it.

Examples of processed data include SHP files (converted from the KMLs) and .Rdata files storing the cleaned dataframes suited to run each causal inference method.

File structure:

```
data/
        raw/
                project1/
                        KML
                        exports/
                                CSV
                                CSV
                                …
                project2/
                        KML
                        exports/
                                CSV
                                CSV
                                …
        processed/
                project1/
                        SHP file
                        .Rdata
                project2/
                        SHP file
                        .Rdata
```

Code

The code folder is separated into 3 separate folders (plus the general non-folder area), each with their autonomous purpose, to facilitate the philosophy of ease of contributing. That means, adding to a certain part of the project should only ever take place in **one** of the folders at a time.

1. **Projects**
   This covers all aspects of the projects on which the methods and their numerous inputs combinations are applied.

   It contains three core files.
   1. `universal_list_of_projects.R`

      As the name suggests, this stores a list of all the project sites available to use for analysis. To add a new project, insert a new entry into the list.

      Its structure is a "getter" function that returns a list of the project names and their respective start dates.

   2. `convert_to_shp_function.R`

      Contains the function that converts a KML file into a SHP file that is able to be uploaded to Google Earth Engine.

      Takes KML inputs form the `/data/raw/<project_name>/` folder.

      Outputs SHP files into the `/data/processed/<project_name>/` folder.

      If the SHP file already exists, it will not create another duplicate. This allows for efficient and convenient usage by just calling the convert function on all the projects when you add a new one.

   3. `converting_all_projects.R`

      Script to run the `convert_to_shp` on all the available projects.

      Call this script every time you add a new project to the universal project list. This also serves as a check to make sure all the proper dependencies exist / aren't broken after you modify the projects section.

2. **Methods**

Contains the folders of all the causal inference methods used throughout the project. Within each folder, there is the following file layout:

1. `<method_name>_processing.R`

   This file contains the processing logic to convert the Google Earth Engine CSV exports into a usable dataframe for the respective methods.

   In general, it should be in the form of a function that takes in one input—the name of the project. That way, it can be applied elsewhere with ease.

   It's important to note that this depends on the naming of the variables and export CSV names from Google Earth Engine, so please check in with Luke or whoever does the Google Earth Engine work to get an up-do-date list of all the variables.

   Also, remember to save the post-processing dataframe as a Rdata file to `data/processed/project_name/` directory with the name `dat_<method_name>.Rdata`

2. `<method_name>_logic.R`

   Contains the function that performs the causal inference method on a given project.

   When adding a new method, it is essential to pass all the different ways to configure a method as function parameters. That way, there is flexibility in how you call your function and more importantly, you can cycle through these different ways when constructing a specification curve.

   It'd probably also be helpful to include a short comment above the function to detail what each parameter you can alter does.

   In general, we want to design these functions to return a dataframe of the results at the very least, rather than just creating graphs. Having the dataframe will allow for more customization

3. **Spec_curve**

   Contains two main files:

   1. Boilerplate schart code from Cornell
   2. Custom code written by Albert that applies the boilerplate code (1) to our project.

   In general, contributors to the specification curve project will not have to modify this code.

File structure:

```
methods/
        matching/
                matching_logic.R
                matching_processing.R
        synthetic_controls/
                synthetic_controls_logic.R
                synthetic_controls_processing.R
        <your_method>/
                <your_method>_logic.R
                <your_method>_processing.R
        …
```

4. Sample Workflows

Adding a project

1. Go to Verra registry and find the project you are looking to add. Download the KML file.
2. Upload KML file to `data/raw/<new_project_name>/` with a reasonable name. Remember, this is the name that will be used to reference the project from here on after.
3. Add the project properties (name, start_year) to `universal_list_of_projects.R`
4. Run the `converting_all_projects.R` to automatically call the convert SHP function on the new project you've added (and also check that all the other projects in the file are up-to-date.

5. Upload the new SHP file found in `data/processed/<new_project_name>/` to Google Earth Engine.
6. Add project details to Google Sheets in the repository, so we can have a non-code way to keep track of project details!
7. For all existing causal inference methods, run the relevant Google Earth Engine scripts to get the export CSVs of those methods for your new project location. Save these into `data/raw/<new_project_name>_exp/`

    This is to ensure that your additional project won't impact the current pipelines in place.

Adding a causal inference method

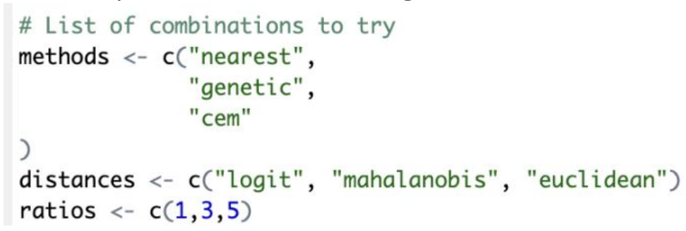
1. Research the causal inference method and figure out what type of data you need. Make a request for a script to get this data in Google Earth Engine
2. If not already done, obtain the Google Earth Engine exports (by either asking Luke or whoever is doing the Google Earth Engine code at the time) and save the CSVs to `data/raw/`. Usually, we put all the CSVs in a folder in the directory titled `<project_name>_exp/`
3. Create a new folder in `code/methods/` for your new method. Again, remember to have consistent naming, as the rest of the project will have to refer to this name as well.
4. Add the logic and processing R scripts. (refer to the ones for matching as an example)

Creating a spec chart

This involves combining all the functions we are previously created.

1. Import the universal list of projects and the processing and logic files for your desired method.
2. Loop through the projects and apply the processing function on each project.
3. Determine the different combinations of inputs to the causal inference method you are trying to test.

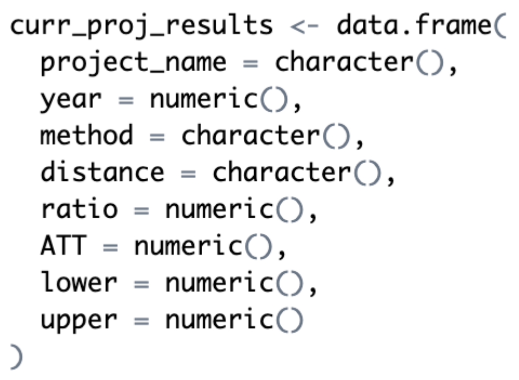
    An example of this for matching can be seen below



```r
# List of combinations to try
methods <- c("nearest",
             "genetic",
             "cem"
)
distances <- c("logit", "mahalanobis", "euclidean")
ratios <- c(1,3,5)
```

4. Create a list (of usually dataframes) to store the results of all the combinations on each project. The length of the list is the number of projects and each individual dataframe stores the relevant information (below) about the specific result from running all the combinations of the method with that specific project.

   Example of the information in each dataframe (in the list) using the matching combinations: (fields like distance, ratio, method will definitely be if you were using say synthetic controls instead)

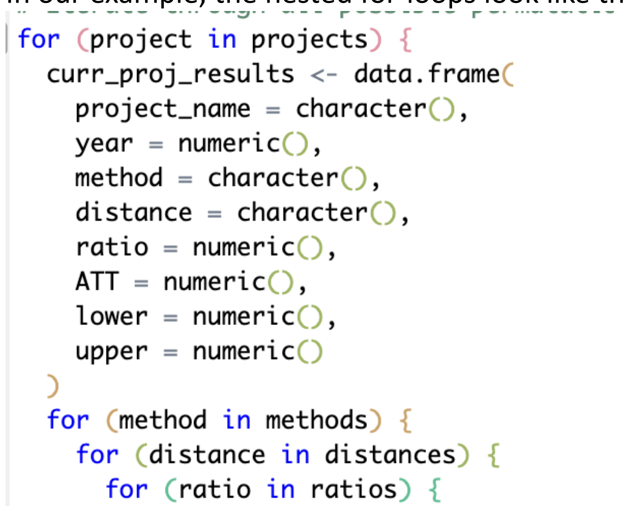


```r
curr_proj_results <- data.frame(
  project_name = character(),
  year = numeric(),
  method = character(),
  distance = character(),
  ratio = numeric(),
  ATT = numeric(),
  lower = numeric(),
  upper = numeric()
)
```

5. Create a nested for-loop, ensuring a layer for all the projects and also all the different ways to modify the method (in our case, there's just three: ratio, distance, method).

   It's important to note that the connection between these loops and the list mentioned in step 4 is that for each new project, a "current dataframe" will be created and will store the information from running all of the combinations of a method on a given project. Then, each of these "current dataframes" are appended together to form the list from step 4.

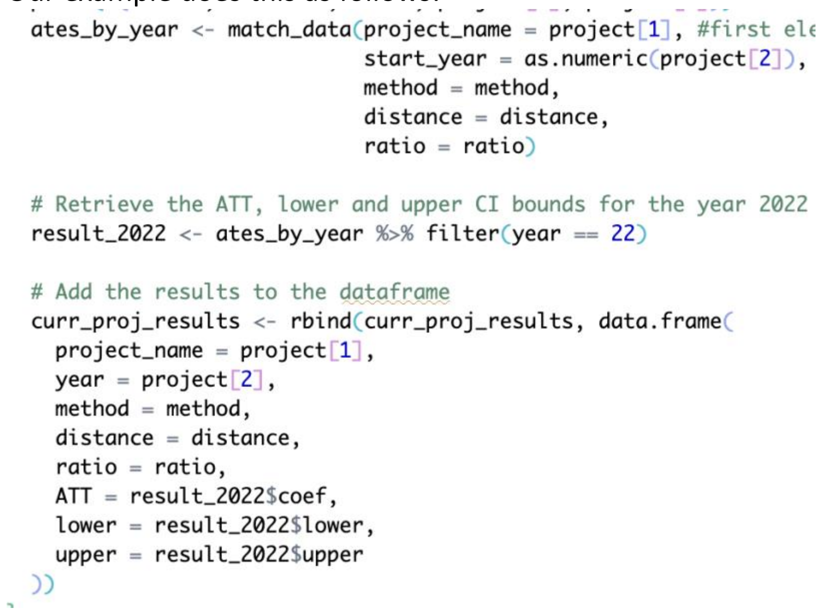
   In our example, the nested for-loops look like this



```r
for (project in projects) {
  curr_proj_results <- data.frame(
    project_name = character(),
    year = numeric(),
    method = character(),
    distance = character(),
    ratio = numeric(),
    ATT = numeric(),
    lower = numeric(),
    upper = numeric()
  )
  for (method in methods) {
    for (distance in distances) {
      for (ratio in ratios) {
```

6. Within the for-loops, you usually want to run that specific combination of inputs for the method and save the results of the final year (in our case 2022). Append this to the "current dataframe" mentioned in step 5 and continue from there.

   Our example does this as follows:



```
ates_by_year <- match_data(project_name = project[1], #first ele
                           start_year = as.numeric(project[2]),
                           method = method,
                           distance = distance,
                           ratio = ratio)

# Retrieve the ATT, lower and upper CI bounds for the year 2022
result_2022 <- ates_by_year %>% filter(year == 22)

# Add the results to the dataframe
curr_proj_results <- rbind(curr_proj_results, data.frame(
  project_name = project[1],
  year = project[2],
  method = method,
  distance = distance,
  ratio = ratio,
  ATT = result_2022$coef,
  lower = result_2022$lower,
  upper = result_2022$upper
))
```

7. Now to actually plot the data, loop through all the projects and call the function in `code/spec_curve/create_spec_chart_function.R`

   Pass in the inputs using the list of dataframes you created earlier. This will store the results of running all combinations of the method for each project.

Please refer to `code/matching_example.R` for a more material walkthrough of these steps.

`code/matching_test.R` is similar but only performs these steps for one project (adpml).

The steps to replicate this for any other method (other than matching) will be very similar, other than slight changes with calling the correct method and also the number of nested for loop levels.

5. Best Practices / Conventions

- Naming conventions follow the snake_case format. Only lowercase letters are used.

- The `here` R package is also used for sourcing and referencing other files.
    - Note: I'm not too sure how different this is than just using the R project files default of setting the "source" of each file to be the project root? But maybe this is future-proofing it in case, it is switched off R project format?

- At the top of each new file, please label it a comment of the following format:

```
# Author: <your name>
# Date: <current date>
# Purpose: <brief statement of file's purpose>
```

- Try to label R scripts with only functions in them with a "_function" at the end of the name.

6. To-do / Current State of Project

December 2023:

Bugs:

- There's an error you run the synthetic controls logic file on some projects (works fine for just adpml, though).

```
Error in gsynth.default(formula = NULL, data = data, Y = Yname, D = Dname, :
 All treated units have been removed.
```

  Some quick research indicates this means that there aren't enough data points with a long enough pretreatment period for this project?

Todos:

- add more projects and methods
- better labels for synthetic controls graphs
- In the processing files, add a check to make sure the relevant CSVs exist. That way, you won't accidentally crash the process method functions (and prevents less user-error).

7. Contact

For any questions regarding this document, please reach out to henry.chen@yale.edu

For any questions, comments, or concerns on the overall project please reach out to either Luke, Albert, Nick, or Henry.