Spatial analysis with R and `Rtargets_prep_tool`: An application example

Ivan Hanigan & Anh Han

20250722

Session 1: Project setup and data loading

Introduction

Welcome to this recording on the 7th of July, where Anh ans I are working through the process of starting a small project using the [Rtargets_prep_tool]{https://github.com/cardat/DatSciTrain_Rtargets_prep_tool}.

Objective

The core task is to take air pollution monitoring locations from a request and perform a GIS operation to identify postcodes within a certain distance of these stations. We will design this study using the targets prep tool.

`Rtargets_prep_tool` setup

We begin by downloading the targets prep toll as a zip file rather than cloning it, to serve as a framework or template for a new project.

This zip file is then transferred to the `projects` folder. The project folder's name is renamed to reflect a specific service: **DatSci_ap_monitors_and_postcodes**.

The `R` project file (*.Rproj) within this folder is also updated with the new name.

Designing the pipeline

The accompanying Excel spreadsheet prep tool is opened, which contains the template for designing the pipeline. the core design work is done by copying the code snippets from column D of the Excel prep tool into the `_targets.R` file.

The `_targets.R` file (between lines 15 and 21/22) is where the pipeline steps will be inserted. Initially, a demonstration set of code (from D2 down to D18) is copied into `_targets.R`.

The `config.R` file is also opened, intended later for storing variables like data directories.

The `run.R` file is opened and this file is used for running the pipeline and for interactive code development and checking. An initial `targets::tar_make()` command is attempted in the console to test the setup, which is expected to fail as the required files haven't been created yet.

Understanding GIS request and data acquisition

The requirements for the service are sourced from an email, which outlined a GIS request from Christine. The request involves **elevated air pollution concentration due to prescribed burning**. The core tasks are to:

- Calculate 5km and 10km buffers around air quality monitoring stations.
- Determine the **postcodes that intersect** each of these buffers.
- Calculate the **area in square metres and proportion** of the postcode within the buffer.

The original email and attached file containing **air pollution monitoring longitude/latitude data** (51 stations) are saved into a newly created `data_provided` folder within the project.

A critical principle is established: **never modify the raw data files** as they should be kept as a record, potentially set to read-only, and original emails converted to `pdf` for archival purposes.

The request details are copied from the email (now a `pdf`) into the top of the `run.R` script as comments. to serve as a reminder of the job at hand.

Step 1: Loading air pollution monitor data

The first step documented in the Excel prep tool is `dat_load_ap_monitor`. The naming convention suggested is `verb_noun` (e.g., `ap_monitor` for the noun, and `load` for the verb).

The input file is initially an Excel spreadsheet. It is strongly recommended to open it in Excel and export it to a `csv` file for better compatibility with R. the Excel data file is opened, checked for filters (none applied), and the latitude/longitude columns (E & F) are confirmed to be good quality decimal degrees. The Excel file is saved as a `csv` in the same `data_provided` folder.

The `csv` file name is then copied and added to the Excel prep tool in column C, row 5, formatted as `filename = "data_provided/file_name.csv"`.

The newly generated code for this target (from D2 down to D6) is copied into `_targets.R`, replacing the previous test code.

Step 2: Developing the `dat_load_ap_monitors` function

Running `tar_make()` confirms an error: no code for the data loading step.

The utility in column E of the Excel prep tool is used to generate the skeleton code for the `do_load_ap_monitors` function. This code is pasted into the `R/do_load_ap_monitors.R` file.

The function is completed to **read the csv file and return the data**: `dat <- read.csv(filename)` and `return(dat)`.

Running `tar_make()` again, with some warnings, confirms the pipeline works and creates the data object.

To inspect the created data, `tar_load(dat_load_ap_monitors)` is used in `run.R`.

The structure of the loaded data `dat` is inspected using `str(dat)`, confirming 51 objects and 32 variables, and that latitude and longitude are numerical.

A quick plot of `longitude` versus `latitude` is performed to visually confirm the spatial distribution, which resembles NSW.

Step 3: Data cleaning within the function

Variable names are cleaned within the `do_load_ap_monitors` function:

- All names are converted to lowercase for standardisation: `names(dat) <- tolower(names(dat))`.
- Dots in names are replaced with underscores: `names(dat) <- gsub("\\.", "_", names(dat))`.
- Any double underscores are replaced with a single underscores: `names(dat) <- gsub("__", "_", names(dat))`.

The interactive testing process for these cleaning steps involves temporarily running the function's internal code within the R console, with a defined `filename` variable.

Step 4: Converting to a spatial object

The next step is to convert the data frame into a **terra `vect` spatial object**. AI chatbots (Copilot/Claude) are used for assistance in generating R code. The query to the chatbot is

> "In base R, convert data frame to terra vect with CRS (Coordinate Reference System) equals GDA94."

The suggested code `vect_obj <- vect(df, geom=c("lon","lat"), crs="GDA94")` is adapted to use `dat` as the data frame and the correct longitude/latitude column names (longitude_s, latitude_e).

The `terra` package (which provides the `vect()` function) and `sf` package are added to the `R/load_packages.R` file.

It is also identified that packages need to be explicitly listed at the top of the `_targets.R` file (e.g. `tar_option_set(packages = c("targets", "data.table", "terra", "sf"))`) for `tar_make()` to work correctly in its clean environment.

Session 2: Buffering and postcode data

Step 5: Refining the first target

A typo in the `do_load_ap_monitors` function is fixed, ensuring the function return the `vect_obj` (the spatial object) rather than the original `dat` (data frame).

`plot(vect_obj)` is used to visually confirm the object is a spatial representation.

Step 6: Buffering air pollution monitors

A new target `dat_buffers_around_ap_monitors` is added to the Excel prep tool.

The input for this target is the output of the previous target: `dat_load_ap_monitors`.

The skeleton code for the `do_buffers_around_ap_monitors()` function is generated using the Excel utility and saved in the R folder.

A critical issue for passing spatial objects between targets is identified:

> They must be **wrap()** when returned from a function and **unwrap()** when loaded by the next function.
>
> In `do_load_ap_monitors`, the return line becomes `return(wrap(vect_obj))`.
>
> In `do_buffers_around_ap_monitors`, the first line after loading the previous target becomes `vect_obj <- unwrap(dat_load_ap_monitors)`.

The buffering command `buff_5km <- buffer(vect_obj, width = 0.05)` is added to the function. The width is set as 0.05 because one decimal degree is approximately 100km, so 0.05 approximates 5km. The function returns the wrapped 5km buffer: `return(wrap(buff_5km))`.

Step 7: Loading postcode data

A new target `dat_load_postcodes` is added to the Excel prep tool with `filename_polygons` as its input. The postcode data file is located in `cloud-car-dat/Environment_General/ABS_data/ABS_POA/abs_poa_2016_data_provided`. The path to this shapefile is then defined as a variable `filename_polygons`, and stored in the `config.R` file for better management.

In the `do_load_postcodes()` function, the `filename_polygons` is used to load the data, with backslashes (\) in the path converted to forward slashes (/).

The final target planned is `dat_intersect_buffers_and_postcodes` with inputs from `dat_buffers_around_ap_monitors` and `dat_load_postcides`.

The `tar_visnetwork()` command in `run.R` file is used to visualise the building blocks and dependencies of the pipeline.

Session 3: Version control, output definition, and intersection

Step 8: Version control with Git

The project is setup with **Git version control** in RStudio. The setup process involves:

- Going to Tools > Project Options > Git/SVN and setting Version control to Git.
- Adding key project files (e.g., `run.R`, `config.R`, `_targets.R`, `Rtarget_prep_tool.xlsx`, and the `R/` folder contents) to the staging area.
- Making the initial commit to Git. This ensures all subsequent changes are tracked, which is good practice to establish early in a project.

Step 9: Defining the end product

Before proceeding, the structure of the final output data is determined. The output should be a **"long but narrow" data frame**. The proposed columns are `postcode`, `ap_monitor`, `buffer_km`, `area_overlap`, `percent_overlap`. This structure is preferred over a "wide" format that would have too many columns (e.g., 102 columns for all monitors and metrics).

This output structure definition is then used to prompt Claude for assistance. The prompt includes the original request, the project's setup using base R and terra package, and the desired output format.

A technique for prompting Claude is shown, using single quotes around the R comments to send the text directly. Claude is largely agrees with the proposed long format.

Step 10: Refining postcode data loading

The focus returns to refining the `do_load_postcodes()` function. The postcode shapefile is loaded into a vect object: `vect_obj <- vect(filename_polygons)`. The `vect_obj` is inspected, confirming its CRS is GDA94 and it contains 2670 polygons with columns such as postal_area_code, postal_area_name, and area_square_km_16.

Similar to the air pollution data, postcode variable names are converted to lowercase: `names(vect_obj) <- tolower(names(vect_obj))`.

To simplify the analysis, the data is subsetted to only include postcodes in NSW (starting with the number 2). This is achivied using `vect_obj <- vect_obj[substr(vect_obj$poa_code_16, 1, 1) == "2", ]`. A plot confirms the subset correctly displays NSW boundaries (including Lord Howe Island). The function concludes by returning the `wrap(vect_obj)`.

Step 11: Developing the intersection logic

The next target to work on is `load_intersect_buffer_and_postcodes`. The entire `_targets.R` script is shared with Claude, explaining the current pipeline status. Claude provides a function for calculating buffer intersections. It is identified that the input targets need to be `tar_load()` and `unwrap()` individually within the intersection function.

CRS compatibility between the buffer and postcode layers is confirmed.

The `expanse()` function is used to calculate the total postcode area in square metres , which is compared to the `area_square_km_16` variable (with appropriate conversion).

A plot to confirm linearity requires converting the spatial vector to a data frame first, which Claude assists with using `as.data.frame`. the approximation of decimal degrees for distance is acknowledged.

Step 12: Revisiting buffer generation

It is discovered that the `do_buffers_around_ap_monitors` functions currently only generates the 5km buffer.

The function is modified to create **both 5km and 10km buffers**: `buff_10km <- buffer(vect_obj, width = 0.1)` and return them as a list: `return(list(wrap(buff_5km), wrap(buff_10km)))`. This change makes `dat_buffers_around_ap_monitors` a list of two spatial objects. This intersection function is updated to unwrap both the 5km and 10km buffers from the list, along with the postcodes.

Initial interactive testing of the intersection logic reveals no intersections, which is unexpected.

Step 13: Troubleshooting - buffer scale issue

To diagnose the lack of intersections, the postcode polygons and the generated buffers are plotted together in **QGIS**.

The `buffer_5km` is exported from R to a geopackge `foo.gpkg` for inspection in QGIS.

In QGIS, upon adding a base map (OpenStreetMap) and zooming in, it becomes evident that the **buffers are extremely small**, appearing as mere dots. Using the scale bar in QGIS confirms the issue: the **5km buffer is actually only 5cm**. This indicates a problem with how the `width` parameter in the buffer cammand is being interpreted when using decimal degrees.

The session concludes with the plan to consult Claude specifically about this buffer scale issue using a prompt like:

> "In the buffer command, the result is only 5cm? Can you help me make it 5km because what we did with the decimal degrees didn't work as I expected."

Step 13.1: `width` parameter correction

To generate a **5km buffer**, the `width` should be set to **5000**, and for a **10km buffer**, the correct value is **10000**. These units reflect the actual radial distance in metres around each monitoring location.

With this correction, the `do_buffer_around_ap_monitors` function was upadted accordingly:

```
buff_5km <- buffer(vect_obj, width = 5000)

buff_10km <- buffer(vect_obj, width = 10000)
```