🧰 Starter Kit – Agentic models for post-processing

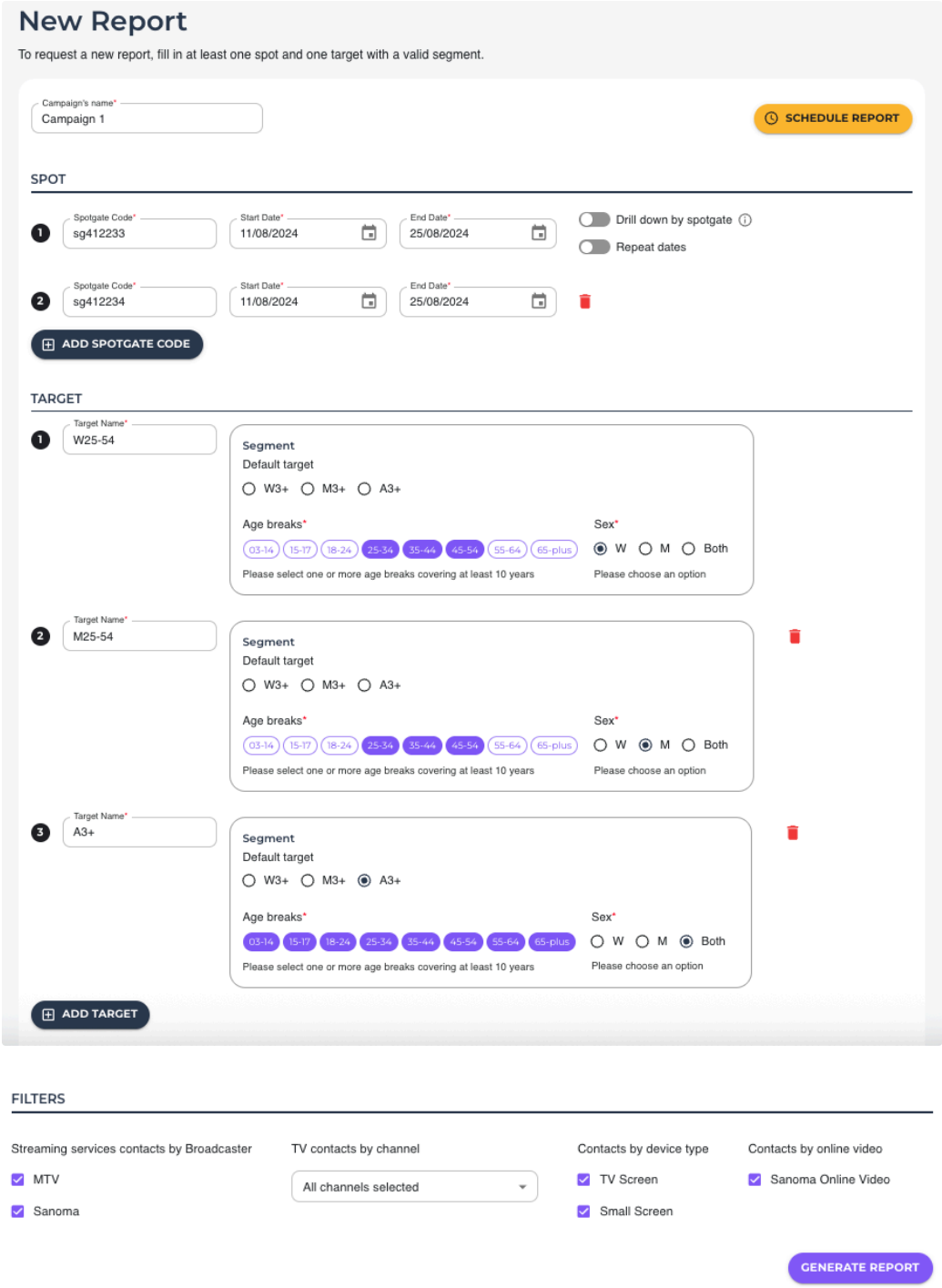
🧰 Starter Kit

1. 📁 input_from_api

These are the JSON files resulting from the application of statistical models combining data from multiple sources. They contain the essential information that is processed and loaded into a proprietary dataBreeders web app for visualization.

- **json_requests.json:** This file contains information about the user request.
  - name_campaign → Campaign name
  - target → The selected demographic targets
  - sg_code → The identifier of a spot, and the period for which the analysis is requested
  - filter → By broadcaster, channel, device type, and/or online video. Except for online_video (exclude this field for now), whenever fields are associated with an empty list, all options are selected

This is a mockup of a compiled report request in the web app.



- **impacts_by_sex_age.json**: This file contains campaign results by date, broadcaster, device_type, ad_type, and sex and age_break.
  - Impacts: It is an impression-related metrics (one impression=one view; one impact=usually more than one view).
  - Do not care about metrics in *30sec_eq* (the equivalent for 30-second spots)
- **impacts_in_target.json**: This file contains campaign results by date, broadcaster, device_type, ad_type, and target.

- Impacts: It is an impression-related metrics (one impression=one view; one impact=usually more than one view).
- Do not care about metrics in *30sec_eq* (the equivalent for 30-second spots)

- **r1_plus_in_target_buildup.json**: This file contains campaign results by start_date, end_date, broadcaster, ad_type, and target.
  - Reach: The percentage of individuals reached during a given period.
  - Build-up: The purpose for this data is to illustrate the cumulative reach over a period.
- **rf_in_target_overall.json**: This file contains campaign results by start_date, end_date, broadcaster, ad_type, and target.
  - Frequency: A sequence, usually up to 20, indicating how many times a target watched the selected campaign over a period. It supports the reach data.
  - Reach: The percentage of individuals reached during a given period.
- **target_universe.json**: This file contains information on the universe (total target population). Even if multiple dates appear, the target universe is identical for each target, and refers to the start date of the user request.
- **tv_spot_schedule.json**: This file contains information on the spots schedule (not used much).
- **universe_by_sex_age.json**: This file contains information on the universe by sex and age (not used much)

2. 📁 postprocessing_functions

- **post_processing_functions.py**: This file contains all functions used to postprocess API input data to produce the desired output on our proprietary web app.
  You will only need to focus on the `main_postprocess_request` function which is based on a *methodcaller* (a callable object that, when called, invokes a specific method on its input). Ignore `main_generator_file` and `main_generator_zip_json_report_to_download` functions.
  Normally, functions within `main_postprocess_request` generate output files for tables and plots. For the agentic models development, we will focuses on tables only.
  Note that:
  - Input parameters: The `main_postprocess_request` function receives several inputs, including **excel_config.json**, which controls calls through *methodcaller*, and **label_objects.json**, which provides replacement and order criteria for postprocessing. The **label_attribute.json** file is minimal, defining details for plots. When testing `main_postprocess_request`, consider that the “path_tables” parameter points to the folder containing files like **excel_config.json** and **label_objects.json**.

- Naming convention: All the invoked functions follow this naming convention → `postprocessing_standard_{tab name}_{type of visualisation}_{kpi to show}_{aggregation level}_{first level metric}_{second level metric}`.

  Where:

  - tab name: name of the specific tab of the webapp
  - type of visualisation: graphical element to display {"plot", "table"}
  - kpi to show: kpi to display {"contact", "contactcum", "contactdaily", "reach"}
  - aggregation level: type of aggregation to report in the plot {"sexage", "target"}
  - first level metric: metric to display {"abs", "trp", "perc"}. TRP stands for Target Rating Point and is a metric generally requested by dataBreeders' clients.
  - second level metric: additional metric to display (only for contacts) {"raw", "30eq"}
- Example: To produce tables with contacts by sex and age figures (contacts refer to a metric similar to impacts, where one contact=usually more than one view), the `postprocessing_standard_tabcontacts_df_contact_sexage_abs_raw` is invoked four times, producing four JSON outputs:
  - Table “contacts by sex and age” - absolute values, raw
  - Table “contacts by sex and age” - trp values, raw
  - ~~Table “contacts by sex and age” - absolute values, 30eq (ignore it)~~
  - ~~Table “contacts by sex and age” - trp values, 30eq (ignore it)~~

Since 30eq is a metric which is not used in our produced reports, ignore any reference to it.

- **excel_config.json:** This file contains all the necessary information to be served to a *methodcaller*. For our scopes, it is important to focus on:
  - python_function → the function name invoked to generate JSON files for the web app (tables and plots – we will only work with tables)
  - file_name → the output file name placed inside the output_json folder

3. 📁 output_json

Once the `main_postprocess_request` finishes to run, JSON files containing postprocessed data are saved to a dedicated output folder.

These are the *in-scope* tables for our project (excluding TRPs, as it does not represent a straightforward metric, and summaries):

- contacts by sex and age in absolute values

- contacts by target in absolute values
- reach 1+ in absolute and percentage values
- reach and frequency by target in absolute and percentage values

We could use these files as a reference to cross-check results produced by the agentic model.

---

4. 📈 Applications

Possible applications include:

- Retrieve specific information from the output files based on user provided input data
- Provide a brief summary of the results, highlighting key findings. Probably, to do this, it would be necessary to gather more context information (to provide some benchmarks).