# SACRO: Semi-Automated Checking Of Research Outputs: Technical Deliverables

Jim Smith*[1], Richard J. Preen[1], Maha Albashir[1], Alba Crespi-Boixader[2], Chris Cole[2], James Liley[3], Simon Rogers[4], Yola Jones[4], Benjamin Butler-Cole[5], and Simon Davy[5]

[1]School of Computer Science, University of the West of England
[2]School of Medicine, University of Dundee
[3]Department of Mathematical Sciences, University of Durham
[4]NHS National Services Scotland
[5]Bennett Institute, University of Oxford

October 31, 2023

**Abstract**

We report on the software deliverables produced by the DARE UK Driver project Semi Automated Checking of Research Outputs (SACRO). This project brought together Trusted Research Environments (TREs) with software developers and Machine Learning researchers and practitioners to produce a suite of tools to address (i) a current bottleneck in checking research outputs for privacy leakage, and (ii) an impending issue that TREs not not have the resources or skills to risk-assess trained machine learning models.

The tools produced broadly split into three parts. The first is a library of 'drop-in' like-for-like replacements for researchers, that replace commonly used query commands in Python/R and Stata with *acro* versions that conduct and report on disclosure risk assessments at the same time as reporting the results of the original queries.

The second is a viewer for TRE output checkers to use that ingests the requested outputs alongside the description of their disclosure risk to facilitate the checking and release process.

The third component is a range of tools for (i) encouraging researchers to consider disclosure risk throughout their Machine Learning workflow, (ii) running a range of different types of 'attacks' on trained models and (iii) co-ordinate running attacks on a model a researcher has requested to egress, and produce a report and recommendation for TRE staff.

*To whom correspondence should be addressed. email: james.smith@uwe.ac.uk

## 1 Introduction

Statistical agencies and other custodians of secure facilities such as Trusted Research Environments (TREs) [5] provide researchers with access to confidential data under the 'Five-Safes' framework [11]. This enforces five orthogonal layers of safety procedures, and the last requires explicit checking of research outputs for disclosure risk. This can be a time-consuming and costly task, requiring skilled staff. This paper discusses the development of an open source tool for automating the statistical disclosure control (SDC) of routine research outputs. The goal is to make the clearance process more efficient and timely, and to allow the skilled checkers to focus their attention on the less straightforward cases.

The purpose of the tool (SACRO, for Semi-Automated Checking of Research Outputs) is to assist researchers and output checkers by distinguishing between research output that is safe to publish, output that requires further analysis, and output that cannot be published because of substantial disclosure risk.

This work builds two prior streams of work:

1. A previous Eurostat-funded project [3, 4] in which Green, Ritchie and Smith developed a proof-of-concept prototype for the proprietary Stata software.The SACRO project enacted a step change in functionality, involving TRE staff in the co-design and evaluation of:
    - The implementation of a Python toolkit.
    - An extensible multi-language platform with interfaces familiar to users of popular statistical tools.
    - Package 'Skins' in Stata and the Rlanguage, demonstrating cross-language support (**see section 3**)
    - A platform-independent viewer tool for TRE output checkers to view requested outputs alongside the automatically generated risk report (**see section 4**).
    - An open source repository with examples, help, documentation, etc.
2. The DARE project *Guidelines and Resources for AI Model Access from TrusTEd Research environments* (GRAIMatter) [6]. The SACRO project builds on the software produced in that project to include:
    - A greater number of 'attacks' on trained machine learning models, including new ones that specifically link to concepts from 'traditional statistical disclosure control'.
    - A formalisation of different ways that a user may present a model and their data sets, with simply configurable scripts to support testing and decision making for different 'user stoies' (**see section 5**).

## 2 Background

The Five Safes framework [11] is a set of principles that enable services to provide safe research access to their data and has been adopted by a range of TREs, including the Office for National Statistics (ONS), Health Data Research-UK (HDR-UK), and the National Institute for Health Research Design Service (NIHR), as well as many others worldwide.

Essentially, these comprise separate considerations of safety in *data*, *projects*, *people*, *settings* and *outputs*. Ensuring the last of these, 'safe outputs' is a complex and often costly human labour-intensive process. Automated output checking aims to improve the rigour and consistency of the output disclosure control process and reduce human workload by automatically identifying, reporting, and (optionally) suppressing disclosive outputs where possible and categorising outputs as 'safe' or 'unsafe'. 'Safe' outputs requiring no or minimal further changes can be expedited through the clearing process whereas 'unsafe' outputs can be prioritised for human review [10].

A small number of SDC tools have been produced to assist in the process of achieving 'safe outputs', such as tauArgus and sdcTable[1], however these are primarily designed for users such as National Statistic Institutes as they require expert knowledge of SDC to use effectively. Moreover, they are designed for tabular outputs, and do not cover the range of statistics produced by researchers

With the aim of improving the efficiency of the process, and (where applicable) reducing the amount of user training required, a recent Eurostat project [3] developed a proof-of-concept prototype in Stata where primary disclosure is regulated by a set of simple rules. For example, requiring that summary statistics are computed on at least a minimum number of observations guarantees a degree of uncertainty with respect to any individual respondent. 'Dominance' rules obscure the reconstructibility of large respondent values where the contribution to a statistic is dominated by only a few individuals. For example, the $p$%-rule sorts the $N$ observations by magnitude and checks whether the sum of the smallest $N-3$ observations is at least $p$% of the largest observation. The $NK$ rule checks that the largest $N$ observations contribute less than $K$% of the total. Also, not all aggregation statistics are permitted: reporting minima or maxima values of a subgroup are prohibited, and regressions are protected by checking that the Residual degrees-of-freedom exceeds a minimum threshold.

Building on the experience of the initial proof-of concept, funding was secured from the UK Research Council's DARE initiative[2] for the project: Semi Automated Checking of Researcher Outputs (SACRO) which involves:

- Computer scientists with backgrounds ranging from AI research to commercial software development.
- A range of TREs as co-designers of a toolset.

[1]Respectively, `https://github.com/sdcTools/tauargus` and `https://github.com/sdcTools/sdcTable`

[2]`https://dareuk.org.uk/`

- SDC theorists and statisticians to provide a conceptual framework for handling different types of output and providing guidance to researchers and output checkers.
- Public Involvement and Engagement specialists and groups to develop a consensus statement around the use of (semi)-automation in disclosure control
- Researchers from a previous DARE project examining the output checking of machine learning models trained on sensitive data within a TRE [6].

In this paper we report on the principal tools developed within the SACRO project, specifically:

1. A toolkit for researchers to use within TREs that produces automated reports on disclosure risk with minimal changes to their practice - simply prefixing common commands with the word ‘acro’.
2. Explicit support for researchers to reduce the number of disclosive outputs they request.
3. Cross-language support: with exemplar interfaces provided for Stata and R.
4. Support for the output types that our TRE partners tell us form the majority of requested releases.
5. A stand-alone viewer for TRE output staff to facilitate rapid, informed, and audited, decision making.
6. A revised guide incorporating theoretical developments, directly linked to its implementation in SACRO.

## 3 The SACRO toolkit

SACRO is composed of three parts which may be deployed independently: the main ‘ACRO-engine’, a stand-alone viewer, and ‘AI-SDC’ - support for disclosure control of machine learning models (described elsewhere).

### 3.1 Design Philosophy

The operational design philosophy is extensively documented in [4], who studied the characteristics that an automated solution needs to have to be feasible, effective, and a positive choice for users. Essential criteria are that it should be:

- Acceptable to users, output checkers and TRE managers;
- Able to implement an organisation’s business rules for primary and secondary disclosure, which may vary across datasets or users;

- Comprehensive, even if the automated tool's response is "I don't know so this needs manual checking";
- Consistent, providing the same results across different studies within a TRE, and across TREs;
- Able to support exceptions under principles-based regimes;
- Scalable over users and outputs.

Key operational requirements were for the tool to work in different technical environments, and to be easily updated through well understood mechanisms. This meant separating the software itself (distributed through a recognised channel[3], from the specification of a given TRE's risk appetite (held in a human and machine readable and editable file).

Acceptability to users was identified as the most crucial element. If researchers and output checkers see the tool as something that makes their life better and easier, then they are more likely to use it effectively. Hence, designing the user interface was identified as a separate workstream in SACRO, and given the same resources as the design and implementation of the output-checking component. This is also one reason why SACRO set up a large network of potential users and tests (see Sec. 7 below).

The 'proof-of-concept' version of ACRO did not address secondary disclosure (such as checking for differencing across tables), for two reasons. First, business rules for secondary checking are often not clear or comprehensive. Second, ACRO/SACRO works by intercepting commands and assessing disclosure risk at the time the output is being produced. Analysing results post-hoc is a considerably harder problem, requiring the researcher to produce a lot more information and also locate the other outputs to be compared. Although SACRO does not currently (as of July 2023) carry out secondary disclosure review, we are investigating how to at least flag potential differencing risks across the set of outputs from a research 'session', and in future, create a library of outputs which might allow secondary disclosure to be assessed, even if only partially.

## 3.2 Workflow

ACRO [8] is an open source toolkit (MIT License) that provides a light-weight 'skin' that sits over well-known analysis tools, in a variety of languages researchers might use. The process is illustrated in Fig. 1. This adds functionality to identify potentially disclosive outputs against a range of commonly used disclosure tests and report to researchers and TREs reasons why outputs should not be released 'as-is'. It creates simple summary documents TRE staff can use to streamline their workflow.

ACRO has been designed with the following aims:

[3]for example, PyPi (`https://pypi.org` or CRAN (`https://cran.r-project.org`)

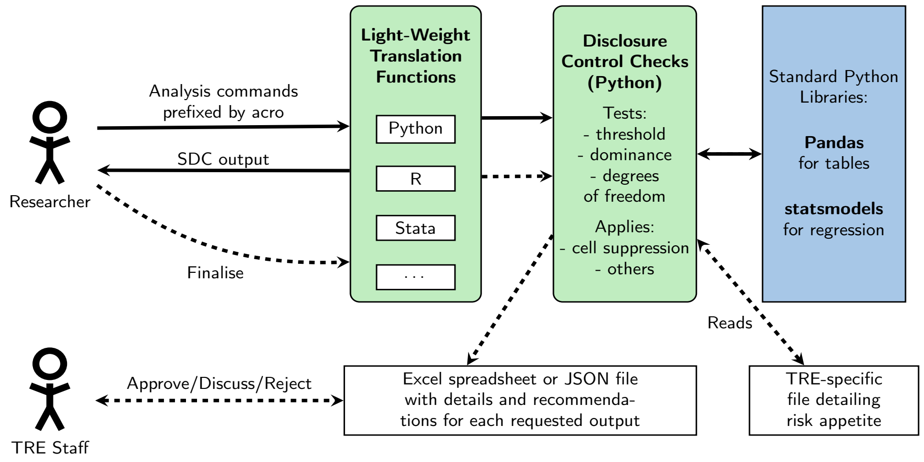

Light-Weight Translation Functions
Python
R
Stata
...
Disclosure Control Checks (Python)
Tests:
- threshold
- dominance
- degrees of freedom
Applies:
- cell suppression
- others
Standard Python Libraries:
Pandas for tables
statsmodels for regression
Analysis commands prefixed by acro
SDC output
Finalise
Researcher
Reads
TRE Staff
Approve/Discuss/Reject
Excel spreadsheet or JSON file with details and recommendations for each requested output
TRE-specific file detailing risk appetite

Figure 1: Schematic illustration of ACRO.

- Reducing barriers to adoption via a front-end application programming interface (API) that is similar to those already commonly used by researchers in their favoured language.
- Providing researchers with: immediate feedback on the results of disclosure checks (on-screen alongside their query results); facilities to add comments or exception requests, and control over what is submitted for review, e.g., removing disclosive outputs if they use feedback to design non-disclosive ones.
- Having a single back-end code base constituting a single source of truth for performing checks, with extensibility for different languages and ongoing support and consistency.
- Providing easy to understand help and documentation.

In practice, researchers prepare their data and statistical queries in the usual way, in their preferred language, using common commands prefixed by 'acro'. The lightweight ACRO translation functions then call the Python back-end, which executes the queries and performs the requisite output checks. The results of the checks, and the queries are immediately displayed to the researcher, and full details are stored in a list. When the user calls `acro.finalise()` to end their session, outputs and all SDC details are saved to file for review by a TRE output checker. A schematic illustration of the ACRO workflow is shown in Figure 1 and some notebooks demonstrating example code usage and output are available via the ACRO project wiki[4].

[4] `https://github.com/AI-SDC/ACRO/wiki`

## 3.3 Checks Implemented

For tabular data (e.g., cross tabulation and pivot tables), we prohibit the reporting of the maximum or minimum value in any cell that represents a sub-group of one or more contributors. Moreover, we suppress, and report the reason, the value of the aggregation statistic (mean, median, variance, etc.) for any cell deemed to be *sensitive*. ACRO currently supports the three most common tests for sensitivity: ensuring the number of contributors is above a frequency threshold, and testing for dominance via $p\%$ and $NK$ rules. ACRO builds a series of suppression masks, which indicate which cells are to be suppressed for each check. A summary outcome table indicating which suppression rule was applied to each cell is presented to the researcher (the grey box in Fig. 2, alongside the query results. For regressions, e.g., linear, probit and logit regression, the tests verify the number of degrees of freedom exceeds a threshold. Immediate feedback on all these checks is designed to support researchers to improve their practice and so reduce the SDC bottleneck by making fewer disclosive requests

As of October 2023 we support the checking of graphical plots in two forms: Survival plots (with accompanying frequency tables) - based on the statsmodels 'SurvfuncRight' method, and histograms based on the pandas 'DataFrame.hist()' method. Having establsihed the method supporting plots - in this case both display the same risks as frequencies- other graphical outputs would be straightforward to support where the results they demonstrate map onto well understood types of analyses

As noted above, all of these tests and checks are configurable according to the TRE's risk appetite. The data custodian, e.g., TRE staff member, specifies the parameter values used for the output checks in a YAML[5] configuration file, which is loaded upon ACRO initialisation. The default ACRO parameters are shown in Table 1. Future releases will offer the option to over-ride these on a dataset, or even attribute level.

Table 1
ACRO Default Parameters for sensitivity tests

| Description | Parameter | Value |
|---|---|---|
| Min frequency threshold for tabular data | `safe_threshold` | 10.0 |
| Min degrees-of-freedom for analytical stats | `safe_dof_threshold` | 10.0 |
| $N$ parameter in $NK$ test | `safe_nk_n` | 2.0 |
| $K$ parameter in $NK$ test | `safe_nk_k` | 0.9 |
| Min ratio for $p\%$ test | `safe_pratio_p` | 0.1 |

## 3.4 The SACRO Python 'Engine'

Python is a popular multi-platform language widely used for data analysis and machine learning. PyPI provides a simple package management system for dis-

[5] `https://yaml.org`

tributing open source Python libraries. Pandas and Statsmodels[6] are industry-standard, mature, popular, and well-supported python packages for data analysis, statistical testing, and statistical data exploration. Pandas is currently used by more than 55% of all Python users [9] and there are many web-sites and user groups providing help with formulating queries.

The use of Python as the primary implementation therefore enables the leveraging of existing expertise and community support with these packages so that the ACRO front-end can be as similar to the API researchers already know and trust, and further facilitates the rapid development of disclosure checking functionality on the back-end. As the PyPI distribution system is simple and allows the use of semantic versioning, it supports a rapid and iterative develop-and-deploy strategy to provide continuing functionality and improvements.

For example, the current version of ACRO may be installed [or updated] as simply as:

```
pip install [--upgrade] acro
```

The currently implemented methods are listed below, split into analysis commands, and sessions management commands. For more details see the ACRO project documentation[7].

#### 3.4.1 Analysis commands for Researchers

These are implemented via the use of multiple inheritance from Pandas and Statsmodels. For making tables, the relevant methods are:

`crosstab`(index, columns[, values, rownames, . . . ])
  Compute a simple cross tabulation of two (or more) factors,
  with options for hierarchies in rows/columns and multiple aggreagation functions.
  Same API as pandas.crosstab.

`pivot_table`(data[, values, index, columns, . . . ])
  Create a spreadsheet-style pivot table as a DataFrame.
  Same API as pandas.pivot_table.

and for regression analysis:

`logit`(endog, exog[, missing, check_rank])
  Fits Logit model.
  Same API as statsmodels.discrete.discrete_model.Logit.

`logitr`(formula, data[, subset, drop_cols])
  Fits Logit model from an R-style formula and DataFrame.
  Same API as statsmodels.formula.api.logit.

[6] `https://github.com/pandas-dev/pandas` and `https://www.statsmodels.org/stable/index.html` respectively

[7] `https://ai-sdc.github.io/ACRO/`

`ols`(endog[, exog, missing, hasconst])
: Fits Ordinary Least Squares Regression.
  Same API as statsmodels.regression.linear_model.OLS.

`olsr`(formula, data[, subset, drop_cols])
: Fits Ordinary Least Squares Regression from an R-style formula and DataFrame.
  Same API as statsmodels.formula.api.ols.

`probit`(endog, exog[, missing, check_rank])
: Fits Probit model.
  Same API as statsmodels.discrete.discrete_model.Probit.

`probitr`(formula, data[, subset, drop_cols])
: Fits Probit model from an R-style formula and DataFrame.
  Same API as statsmodels.formula.api.probit.

We currently support two forms of graphical outputs:

`surv_func`(time,status,output,...)
: Fits Survival model e.g Kaplan-Meier wiht option for plots, frequency tables or both.
  Same API as statsmodels.Survfuncright.

`hist`(data,column,by,...)
: Produces a histogram. Same api as pandas.DataFrame.hist()

### 3.4.2 Session Management Commands

`ACRO()`(config,suppress)
: Creates an ACRO session object with optional parameters for a config (risk appetite) filename
  and whether disclosive tables should have suppression applied (default False).

`print_outputs`()
: Prints the current results dictionary - i.e., the outputs that would be sent for checking.

`remove_output`(key)
: Removes an output from the results dictionary.

`rename_output`(key, newname=)
: Assigns a new (ideally more self-explanatory) name to an output from the results dictionary.

`add_comments`(key,text)
: Allows researcher to add a description for an output

`add_exception`(key,text)
: Allows a user to request and justify an exception to strict rules-based checking.

```
» safe_table = acro.crosstab(
    df.recommend, df.parents, values=df.children, aggfunc="mean")
» print(safe_table)
```

```
  INFO:get_summary:fail;
  threshold:  4 cells may need suppressing

  INFO:outcome_df:
  parents          great_pret   pretentious   usual
  recommend
  not_recom        ok           ok            ok
  priority         ok           ok            ok
  recommend        threshold    threshold     threshold
  spec_prior       ok           ok            ok
  very_recommend   threshold    ok            ok

  INFO:acro:add():  output_1
```

```
grant_type   great_pret   pretentious   usual
recommend
not_recom    1440         1440          1440
priority     858          1484          1924
recommend    0            0             0
spec_prior   2022         1264          758
very_recom   0            132           196
```

Figure 2: Example ACRO query for the ‘nursery’ data(top), with immediate disclosure control reporting (middle, grey background - pink onscreen) followed output (bottom). This ’researcher-view’ corresponds to the top image in the viewer screenshots

`custom_output`(filename,description)
: Adds a file containing output from unsupported analysis to an ACRO session for inclusion in outputs shown in viewer.

`finalise`(directory_name, format)
: Creates a results file for checking in the desired format(*json* or *xlsx*).

An example ACRO query run on the nursery admission dataset[8] and its output is shown in Fig. 2. This is the ‘researchers-view’ of the output at run-time. The corresponding ‘TRE-view’ is shown in the top screenshot in Fig. 3. This example does not have an aggregation function so dominance rules are not applied, otherwise they would also show in the ‘INFO’ section of the report in any relevant cells. Note that if the user starts their session with `acro= ACRO(suppress=True)` then any disclosive cells would have their values set to `NaN`

[8]`https://www.openml.org/search?type=data&sort=runs&id=1568&status=active`

### 3.5 The R interface to ACRO

The R front-end is an example of cross-language support. It provides a set of wrapper functions that execute Python back-end checking via the reticulate[9] package, which provides automatic conversations for many types, e.g., R data frame to Pandas DataFrame.

A session is created when the acro package is called `source("../acro.R")` and thereafter the acro methods work as callable functions with the prefix `acro_` e.g., `acro_rename_output(output5,"xy-plot")` etc., and to end a session the user calls `acro_finalise(results_dir,“json")`

For regressions, the common R `lm()` and `glm()` functions were shadowed with equivalent versions implemented as `acro_lm()` and `acro_glm()`, respectively. For tabular data, the dplyr[10] package is commonly used within R, however no simple cross tabulation or pivot table functions are provided; instead various combinations of `groupby()` and `summarize()` etc. are used. Therefore, at this stage of development, the Python cross tabulation and pivot table functions were directly interfaced with `acro_crosstab()` and `acro_pivot_table()`.

We support the R`table()` command for simple tables of frequencies (contingencies) with that command's parameters being automatically mapped on to the equivalent `acro.crosstab()` parameters.

Finally , to provide more help we provide pass through access to the extensive help on acro commands available in Python via the function `acro_help(command)`.

**Update:** As of October 2023 an Rpackage is being finalised for submission to CRAN allowing one-click installation of the Rinterface and the underpinning python infrastructure.

### 3.6 Stata Interface

This makes extensive use of Stata's `SFIToolkit` library to manage a python session, transfer data in memory from stata to a Pandas dataframe in the python session, and results back to the Stata window. A simple *acro.ado* file defines a new function `acro` which takes as parameters either one of the ACRO session management methods (adding `init()` to start a session) or the name of a standard Stata function such as `table, regress` etc. Stata's inbuilt parsing functions are used to separate out the parts of command and pass them as lists to a python function `parse_and_run()` which handles the rest of the translation between the two languages.

## 4 SACRO Viewer for Output Checking

We have also created an open-source platform-independent stand-alone viewer for output checkers to use to: view outputs and their risks; make decisions with reasons (all recorded for auditing purposes); and produce zipped packages of files

[9]`https://github.com/rstudio/reticulate`
[10]`https://github.com/tidyverse/dplyr`

for release [7]. Figure 3 illustrates two screenshots from the version currently (July 2023) being evaluated by TREs.

The SACRO Outputs Viewer runs on Windows, Linux (Ubuntu/Debian), and macOS. It is packaged using standard tooling (MSI on Windows and .deb on Linux) and uses the Electron browser and is written in JavaScript and Python. The Windows/Linux installers bundle all dependencies, including JavaScript and Python runtimes.

The viewer supports and renders a range of different file types for results from unsupported queries. A separate script lets TRE staff create an ACRO session from a set of output files in a directory, and hence use the viewer for making and recording decisions, even if the researcher has not used ACRO during their analysis. Automated disclosure risk analysis is not provided in those cases.

## 5 Risk Assessment of Trained Machine Learning Models

Risk Assessment of Trained Machine Learning Models is supported via the GitHub repository `https://github.com/AI-SDC/AI-SDC`, with documentation at `https://ai-sdc.github.io/AI-SDC/`.

The package is hosted on PyPi and can be installed via the command `pip install aisdc`.

This repository contains collection of tools and resources for managing the statistical disclosure control of trained machine learning models. For a brief introduction to the underlying resources developed during the GRAIMatter project, see [12].

This code has been extensively refactored during SACRO, in particular informed by an increasing number of requests to support TREs where researchers has presented models of egress with varying amount of supporting information.

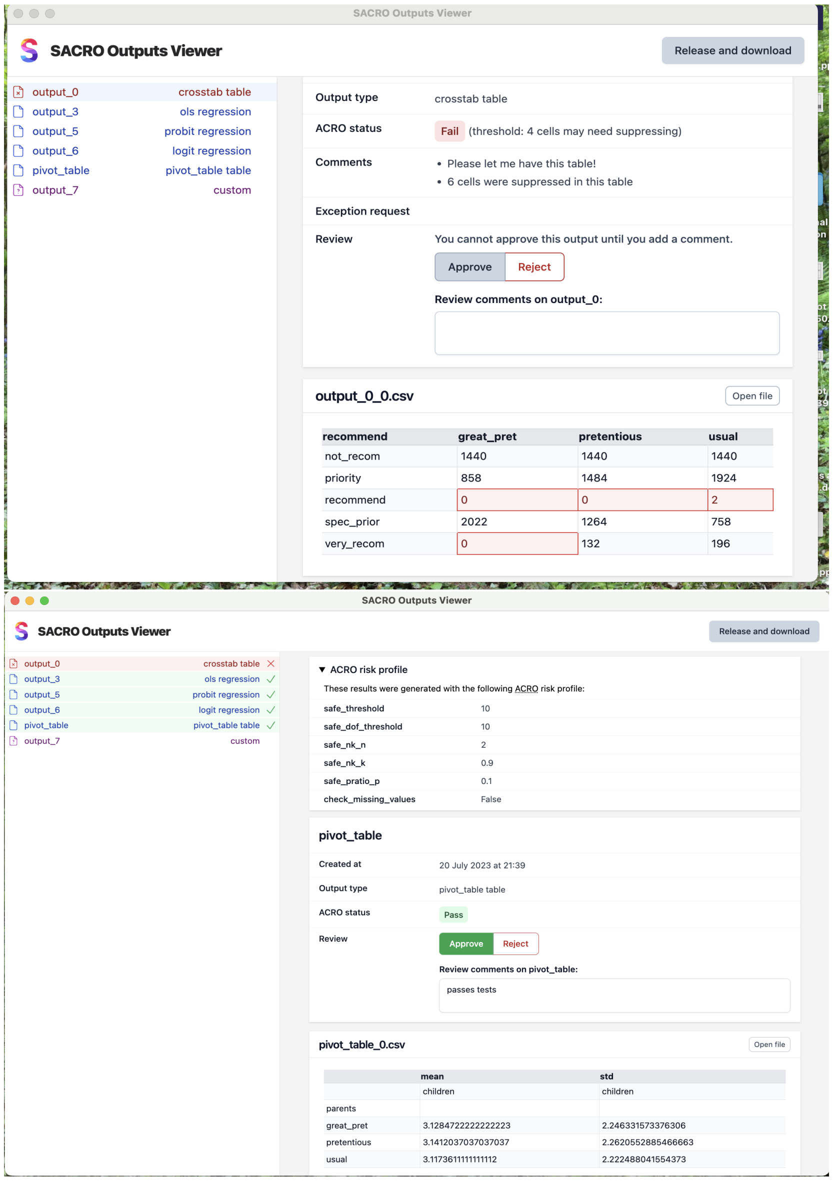
### 5.1 User Guides

In response to that practice 'on the ground', and to developments elsewhere in the Machine Learning field, we created a taxonomy of different 'user stories', illustrated in Fig. 4.

A collection of user guides can be found in the 'user_stories' folder of this repository. These guides include configurable examples from the perspective of both a researcher and a TRE, with separate scripts for each. Instructions on how to use each of these scripts and which scripts to use are included in the README of the `https://github.com/AI-SDC/AI-SDC/user_stories` folder.

The repository contains the following directories:

- 'aisdc': the main code for the package:
  - 'attacks' Contains a variety of privacy attacks on machine learning models, including membership and attribute inference.


SACRO Outputs Viewer

**SACRO Outputs Viewer**

| | |
|---|---|
| output_0 | crosstab table |
| output_3 | ols regression |
| output_5 | probit regression |
| output_6 | logit regression |
| pivot_table | pivot_table table |
| output_7 | custom |

| | |
|---|---|
| Output type | crosstab table |
| ACRO status | Fail (threshold: 4 cells may need suppressing) |
| Comments | • Please let me have this table!<br>• 6 cells were suppressed in this table |
| Exception request | |
| Review | You cannot approve this output until you add a comment. |

Review comments on output_0:

**output_0_0.csv**

| recommend | great_pret | pretentious | usual |
|---|---|---|---|
| not_recom | 1440 | 1440 | 1440 |
| priority | 858 | 1484 | 1924 |
| recommend | 0 | 0 | 2 |
| spec_prior | 2022 | 1264 | 758 |
| very_recom | 0 | 132 | 196 |

SACRO Outputs Viewer

**SACRO Outputs Viewer**

| | | |
|---|---|---|
| output_0 | crosstab table | ✗ |
| output_3 | ols regression | ✓ |
| output_5 | probit regression | ✓ |
| output_6 | logit regression | ✓ |
| pivot_table | pivot_table table | ✓ |
| output_7 | custom | |

▼ ACRO risk profile

These results were generated with the following ACRO risk profile:

| | |
|---|---|
| safe_threshold | 10 |
| safe_dof_threshold | 10 |
| safe_nk_n | 2 |
| safe_nk_k | 0.9 |
| safe_pratio_p | 0.1 |
| check_missing_values | False |

**pivot_table**

| | |
|---|---|
| Created at | 20 July 2023 at 21:39 |
| Output type | pivot_table table |
| ACRO status | Pass |
| Review | |

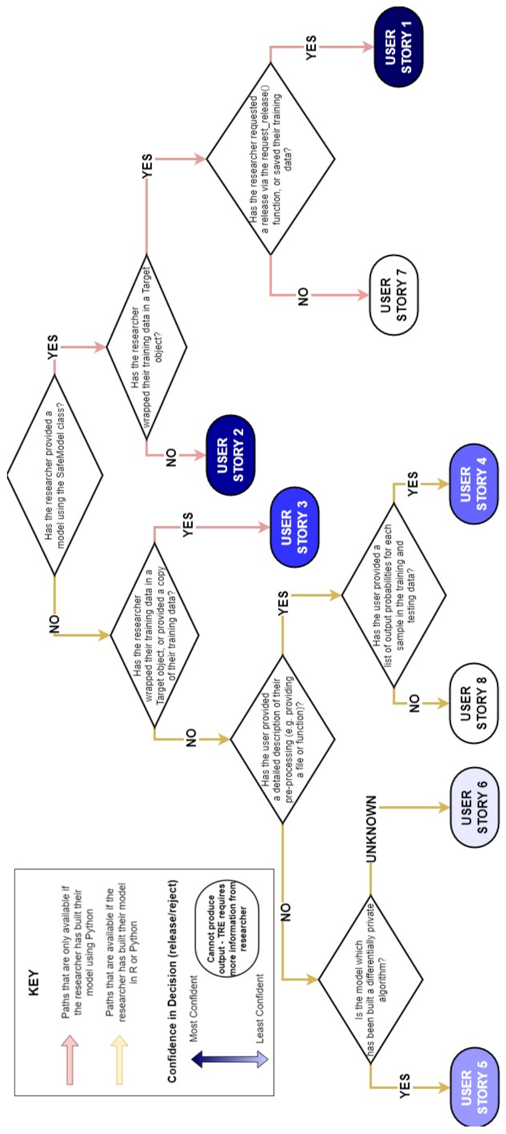
Review comments on pivot_table:

passes tests

**pivot_table_0.csv**

| | mean | std |
|---|---|---|
| | children | children |
| parents | | |
| great_pret | 3.1284722222222223 | 2.246331573376306 |
| pretentious | 3.1412037037037037 | 2.2620552885466663 |
| usual | 3.1173611111111112 | 2.222488041554373 |

Figure 3: Two screenshots of viewer. The left hand column shows list of files requested. In top image, colouring of file names suggests which files require special attention. In lower image background colour-coding and tick/cross symbols show decisions made by output checker. Top image shows checker viewing table that fails disclosure tests, with problematic cells highlighted in red. Bottom shows acceptable table. Also in this image the top right hand panel shows option to view TRE 'risk appetite' expanded.



Has the researcher provided a model using the SafeModel class?
YES
NO
Has the researcher wrapped their training data in a Target object?
YES
NO
USER STORY 2
Has the researcher requested a release via the request_release() function, or saved their training data?
YES
NO
USER STORY 1
USER STORY 7
Has the researcher wrapped their training data in a Target object, or provided a copy of their training data?
YES
NO
USER STORY 3
Has the user provided a detailed description of their pre-processing (e.g. providing a file or function)?
YES
NO
Has the user provided a list of output probabilities for each sample in the training and testing data?
YES
NO
USER STORY 4
USER STORY 8
Is the model which has been built a differentially private algorithm?
UNKNOWN
YES
USER STORY 6
USER STORY 5
KEY
Paths that are only available if the researcher has built their model using Python
Paths that are available if the researcher has built their model in R or Python
Confidence in Decision (release/reject)
Most Confident
Least Confident
Cannot produce output - TRE requires more information from researcher

Figure 4: Taxonomy of models and evidence to support assessment

  - ‘preprocessing’ Contains preprocessing modules for test datasets.
  - ‘safemodel’ The safemodel package is an open source wrapper for common machine learning models. It is designed for use by researchers in TREs where disclosure control methods must be implemented. Safemodel aims to give researchers greater confidence that their models are more compliant with disclosure control. Currently there are wrappers for:
    * The sklearn implementation of DecisionTreeClassifier().
    * The sklearn implementation of RandomForestClassifier().
    * The sklearn implementation of SupportVectorClassifier(), using a differential private implementation.
    * Artificial Neural Networks created using the Keras/Tensorflow ecosystem. In this case the SafeKerasModel() class enforces the use of a differentially private optimiser (from the tensorflow-privacy package), and removes the optimizer object prior from the model saved for egress.

    In addition the safemodel classes perform a range of checks to make sure that users have not inadvertently (or otherwise) changed their models since they were created by the `fit()` method.

- ‘docs’ Contains Sphinx documentation files.
- ‘example_notebooks’ Contains short tutorials on the basic concept of “safe_XX" versions of machine learning algorithms, and examples of some specific algorithms. These are in the form of Jupyter notebooks which may be viewed on GitHub or downloaded and run locally for a more interactive experience.
- ‘examples’ Contains examples of how to run the code contained in this repository including:
  - How to simulate attribute inference attacks ‘attribute_inference_example.py’.
  - How to simulate membership inference attacks:
    Worst case scenario attack ‘worst_case_attack_example.py’.
  - LIRA scenario attack ‘lira_attack_example.py’.
    An implementation of the Likelihood Ratio Attack from [1]
  - Integration of attacks into safemodel classes
    ‘safemodel_attack_integration_bothcalls.py’.
- ‘risk_examples’ Contains hypothetical examples of data leakage through machine learning models as described in the [6].
- ‘tests’ Contains unit tests.

## 6 Linking theory and implementation

As part of the project, the SACRO team committed to review and re-develop the theory and operational guidelines for output SDC. The aim was threefold; first, to bring together key points from the OSDC literature (and fill in some of the theoretical gaps) to provide an integrated guide to both theory and practice of output checking; second, to develop a new approach to OSDC based on classifications into groups (see [2], for details); third, to explicitly link theory to operational rules and their implementation in manual and automatic checking regimes. The third aim is essential to demonstrating that SACRO is not seen as a 'black box' implementing its own rules, but is fully integrated into core theory. It is also important for showing how manual and automatic output checking necessarily differs. For example, dominance checks are almost impossible for a human, but straightforward for computers; on the other hand, computers cannot easily identify whether zero cells in tables are structural or disclosive, but humans can. The purpose of the guide is to show precisely what checks have been made, where differences occur between humans and computers, and why they are necessary.

## 7 Engagement with TREs

One of the lessons learned from the original Stata version of ACRO [3] was the importance of user buy-in. Although that version met its design goals (and has subsequently been adopted by Eurostat in its TRE), reaction to it was a mixture of *"this looks useful, I'll give a go"*, *"this looks useful, I'll wait to see it installed before I commit myself"*, and *"I've read the installation manual and have no idea what's going on, so it's a no"*. As a result,that version of ACRO has remained largely within the project remit: a demonstration of possibilities. The SACRO project was intended to involve co-design from the outset to take ACRO to the next stage, of general utility and application. This involved three tests:

1. Would a new tool be acceptable to users?
2. Would a new tool be acceptable to output checkers?
3. Could a new tool be installed in secure research environments?

The SACRO project took two approaches. First, six TREs (OpenSAFELY at Oxford University, and four Scottish Safe Havens) are co-investigators on the project to provide input from user and output checker perspectives (OpenSAFELY led the design of the user interface). This group also directly tested the feasibility of installing and allowing the Python code to run on their systems as TREs differ in their perceptions of python's 'riskiness'. Second, the SACRO team contacted a large number of TREs in the UK and abroad, and set up a network of interested parties potentially willing to be testers. Several engagement events with this group identified how they worked and what they would expect from an automatic solution. At the time of writing (July 2023), the first 'external' TRE's are starting to install and run the tool with genuine users. SACRO has a workpackage dedicated to helping TREs set up their systems, and then collecting evaluation feedback. This aims to make sure that the tool is tested in as wide a variety of environments as possible, given the time constrain. A secondary aim is to involve TREs in the development, to build a sense of ownership and lay the foundations for widespread adoption. This helps to address the concerns of 'wait-and-see' TREs.

## 8 Acknowledgements and Future Plans

This work is funded by UK Research and Innovation, (UKRI) [Grant Number MC_PC_23006], as part of Phase 1 of the DARE UK (Data and Analytics Research Environments UK) programme, delivered in partnership with Health Data Research UK (HDR UK) and Administrative Data Research UK (ADR UK).

Additional features and improved user experience will be facilitated by the involvement of end-users and output checkers. Beyond then, UWE has committed to web hosting various resources for the indefinite future, and partners have agreed to continue support and development of the toolkits. We are keen to engage with any interested parties to enrich and build an on-going community of support for SACRO.

## References

[1] Nicholas Carlini, Steve Chien, Milad Nasr, Shuang Song, Andreas Terzis, and Florian Tramer. Membership inference attacks from first principles. In Rakesh Bobba, editor, *IEEE Symposium on Security and Privacy*, pages 1897–1914, Piscataway, NJ, USA, 2022. IEEE Press.

[2] B. Derrick, E. Green, F. Ritchie, J. Smith, and P. White. Towards a comprehensive theory and practice of output SDC. In *UNECE/Eurostat Workshop on Statistical Data Confidentiality*, 2023.

[3] E. Green, F. Ritchie, and J. Smith. Automatic checking of research outputs (ACRO): A tool for dynamic disclosure checks. *ESS Statistical Working Papers*, 2021:1–27, October 2021. doi: 10.2785/75954.

[4] Elizabeth Green, Felix Ritchie, and James Smith. Understanding output checking. Technical report, European Commission (Eurostat - Methodology Directorate), 2020.

[5] T. Hubbard, G. Reilly, S. Varma, and D. Seymour. Trusted research environments (TRE) green paper. *ZENODO*, 2020:1–31, July 2020. doi: 10.5281/zenodo.4594704.

[6] Emily et al. Jefferson. GRAIMATTER Green Paper: Recommendations for disclosure control of trained Machine Learning (ML) models from Trusted Research Environments (TREs), September 2022.

[7] Open-Safely. Sacro:a tool for fast, secure and effective output checking, which can work in any TRE. `https://github.com/opensafely-core/sacro`, 2023.

[8] Richard John Preen, Jim Smith, Maha Albashir, and Simon Davy. ACRO. `https://github.com/AI-SDC/ACRO`, 2023.

[9] Python Software Foundation. Python developers survey 2021 results. `https://lp.jetbrains.com/python-developers-survey-2021/`, 2021. Accessed: 24/07/2023.

[10] F. Ritchie. Disclosure detection in research environments in practice. In *Joint UNECE/Eurostat work session on statistical data confidentiality*, volume WP. 73. United Nations Statistical Commission and Economic Commission for Europe Conference of Europe Statisticians, 2008.

[11] F. Ritchie. The 'five safes': A framework for planning, designing and evaluating data access solutions. *Zenodo*, 2017:1–5, September 2017. doi: 10.5281/zenodo.897821.

[12] Jim Smith, Richard J. Preen, Andrew McCarthy, Alba Crespi-Boixader, James Liley, and Simon Rogers. Safe machine learning model release from trusted research environments: The ai-sdc package, 2022.