

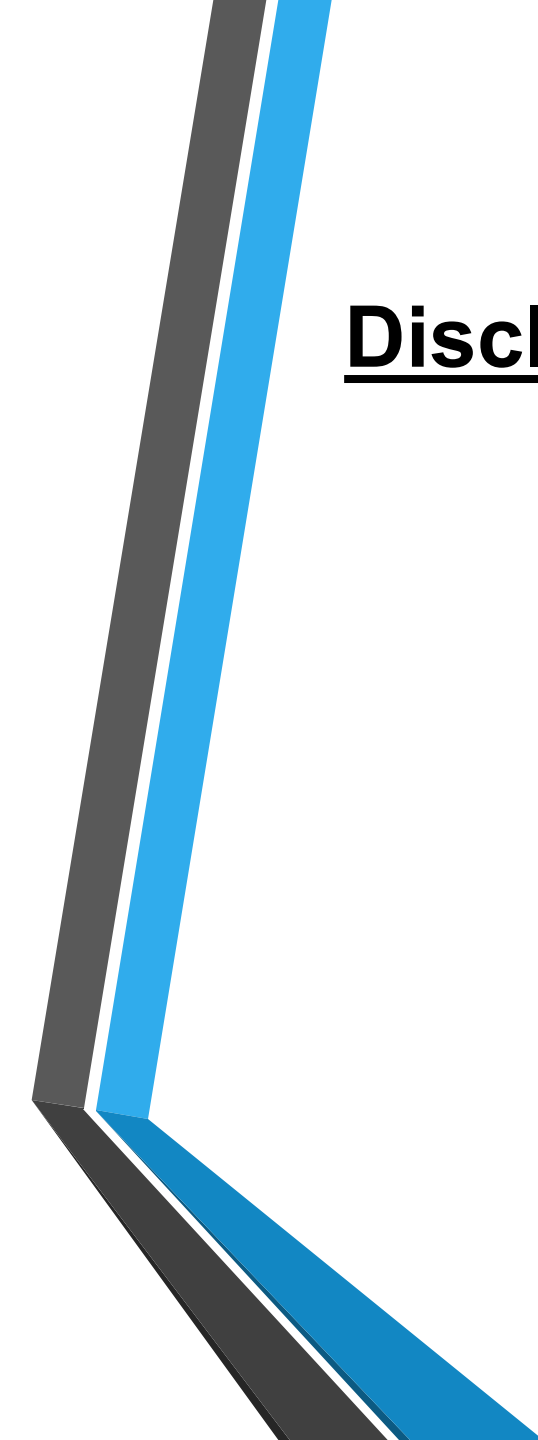


DH01

Data Anonymisation and Risk Assessment Automation — PHUSE WG Deliverable

PHUSE EU Connect 2020

Lukasz Kniola, Biogen, Maidenhead, UK



Disclaimer

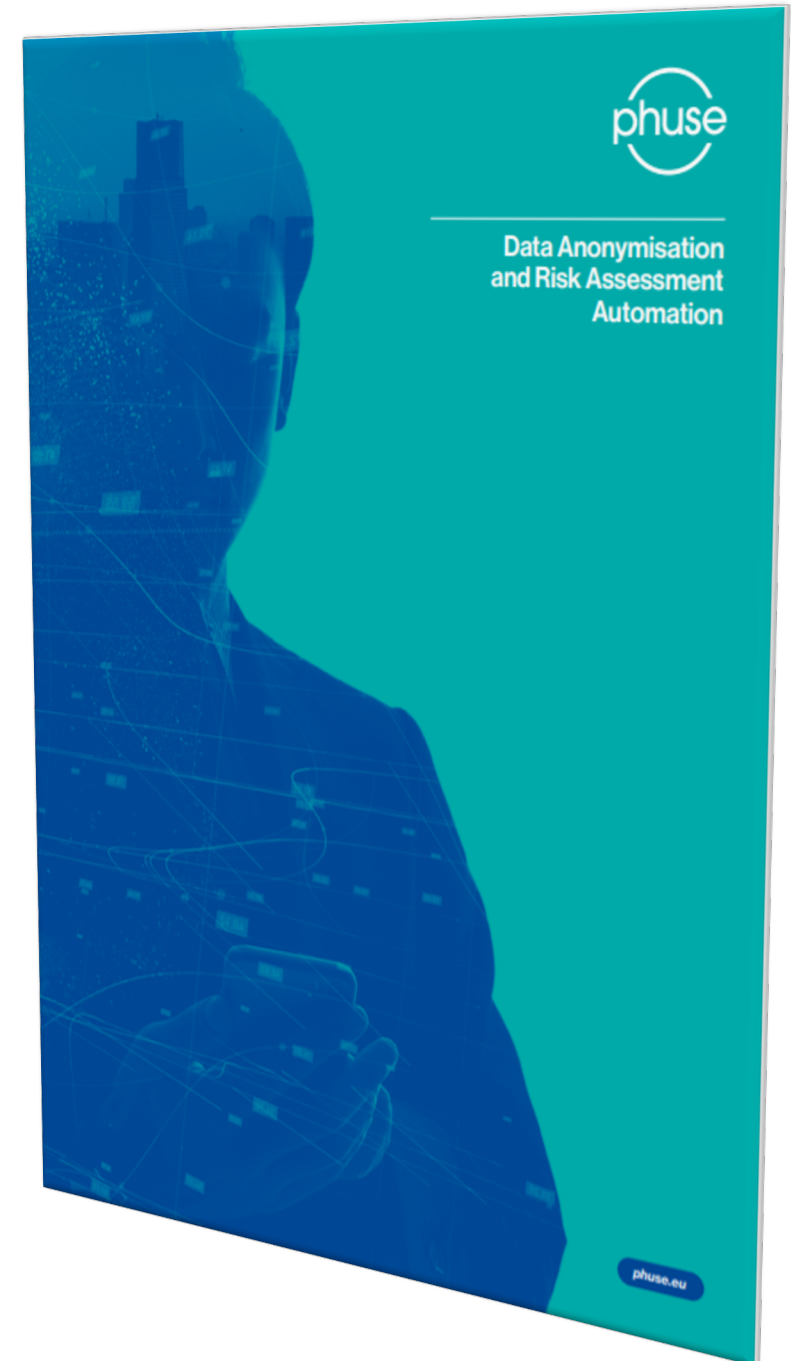
The scope of this presentation is to share the opinions and suggestions of the author. The interpretations of standards and procedures contained in this presentation are those of the author and are **not necessarily correct**.

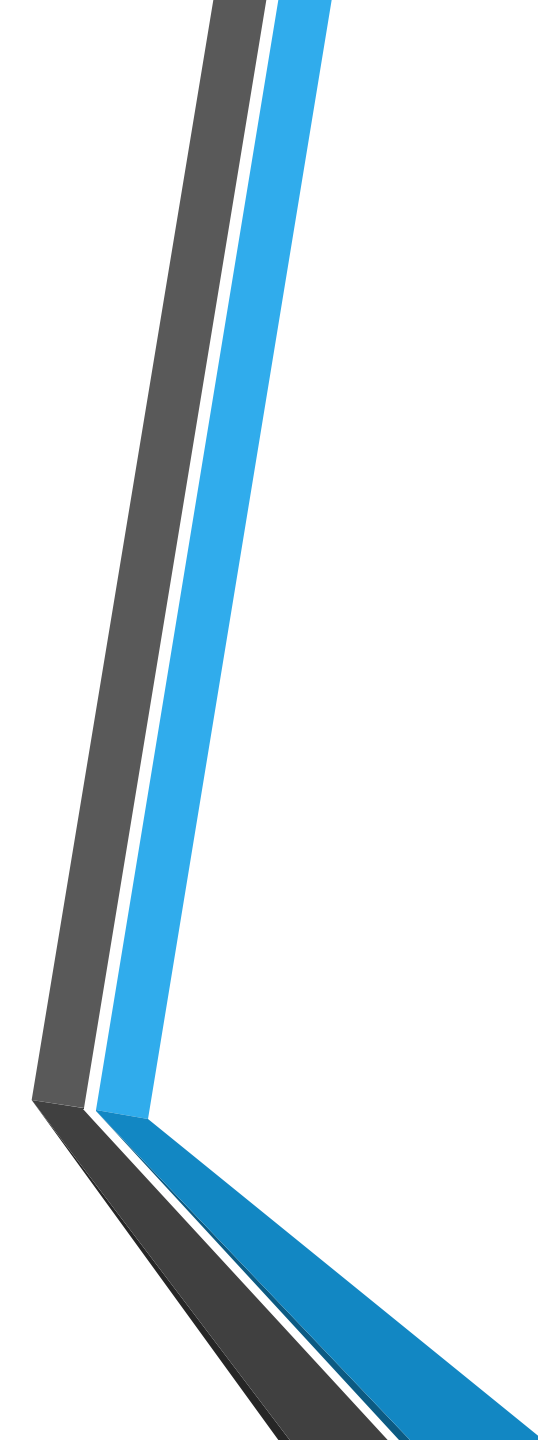
Any views, thoughts, and opinions stated within this paper are those of the author alone, and they **do not represent** the positions of their employer or any other organization or group they may be affiliated with.

www.phuse.eu/white-papers

Data Anonymisation and Risk Assessment Automation

Version 1.0, 01-Jul-2020





Acknowledgements

Authors:

Lukasz Kniola (Lead), Alex Hughes, Anna Paczewska-Sosnowska, Arlene Coleman, Gregory Campbell, Janine Hodgson, Jasper Enderman, Mahumood Hameed, Jonathan Hartzel, Liam Sheehan, Sarah Hewitt, Terry Walsh

Oversight:

Jean-Marc Ferran

PHUSE Data Transparency Working Group

Bringing the document to life

D25		Selecting quasi-identifiers for the calculation	
A	B	C	D
DE-ID AND RISK ANALYSIS AUTOMATION			
I	GATHER ORIGINAL	1.1	Identify source data, specifications, formats
II	PRE-PROCESS DATA	2.1	Pre-process study datasets to create standard input data (special care when dealing with non-CDISC datasets): • ensure each variable name has unique data type, • remove SAS formats, • convert formatted numeric variables to character • convert numeric dates to character ISO variables • replace non-standard values with controlled terminology values.
III	FIND POTENTIAL DIRECT AND QUASI	3.1	Use Phuse De-ID SDTM standard (spreadsheet) to cross-check all variables
		3.2	Find date variables
		3.3	Find demog variables based on standard values
		3.4	Find free-entry (comments) variables
		3.5	Find values (sex, age, race, dates) within result ("STRESC"/"STRESN"), free-entry fields,
		3.6	Find mentions of sensitive info (pregnancy, abortions, mental health, family, etc.)
		3.7	Identify gender specific laboratory analytes
		3.8	Identify rare diseases reported in Adverse event & Medical History data / medications for rare
IV	FIND RULES TO APPLY	4.1	Best age aggregation (optimal age bands)
		4.2	AGE>89: redact AGE (and YOB)
		4.3	Countries to regions aggregation
		4.4	Pool low freq values / change any values with <n% frequency to "OTHER"
		4.5	Date of birth: date offset then YOB transformation
		4.6	*REFID*/SPID*/LNKID variables: common deid transformation (linking records across datasets)
		4.7	Date of death
		4.8	Low frequency events / outliers (unusual LBVS/etc. Results)
V	CALCULATE RISK	5.1	Selecting quasi-identifiers for the calculation
		5.2	Calculate risk of successful re-identification. Test all combinations of QIs.
		5.3	Select reference population (within dataset or using bigger dataset)
		5.4	Assess risk of attempt being made (based on relationship with requestor)
		5.5	Calculate risk of inadvertent re-id
		5.6	Calculate risk of data breach
		5.7	Quantify overall risk of de-id, combining all risk ingredients
		5.8	Assess k-anonymity and uniqueness
		5.9	Assess km-anonymity and uniqueness

DE-ID AND RISK ANALYSIS AUTOMATION	
Row:	7.5
Task Name:	Rule: OFFSET (dates)
Description:	Offset date values by a certain number of days. This must be applied consistently, so that dates within any given dataset are shifted by the same number of days, to preserve relationships.
Variations of the problem:	Dates can be offset by a random number (e.g. -30 / +30 days) or they can be shifted to one starting point (as if all subjects had the same start date).
Pitfalls	Partial dates need special consideration Dates imputed from partial dates need special consideration
Other comments:	SDTM Date Offset specification provides further guidance on how to handle dates.
Resources:	(link)

1. Import packages

```

import sys
import os
import random
import math
import time
import datetime
import logging
import argparse
import itertools
import collections
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import networkx as nx
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torch.nn.parallel
import torch.distributed as dist
import torch.backends.cudnn as cudnn
import torch.multiprocessing as mp
import torch.utils.data
import torch.utils.data.distributed
import torch.nn.init
import torch.nn.modules.pooling
import torch.nn.modules.conv
import torch.nn.modules.linear
import torch.nn.modules.dropout
import torch.nn.modules.activation
import torch.nn.modules.normalization
import torch.nn.modules.batch_norm
import torch.nn.modules.pooling
import torch.nn.modules.conv
import torch.nn.modules.linear
import torch.nn.modules.dropout
import torch.nn.modules.activation
import torch.nn.modules.normalization
import torch.nn.modules.batch_norm

```

2. Define a graph

Graphs are a type of data structure that can be used to represent a network of nodes and edges. In this example, we will create a simple graph with 10 nodes and 20 edges.

```

graph = nx.Graph()
graph.add_nodes(range(10))
graph.add_edges([
    (0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (0, 7), (0, 8), (0, 9),
    (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 7), (1, 8), (1, 9),
    (2, 3), (2, 4), (2, 5), (2, 6), (2, 7), (2, 8), (2, 9),
    (3, 4), (3, 5), (3, 6), (3, 7), (3, 8), (3, 9),
    (4, 5), (4, 6), (4, 7), (4, 8), (4, 9),
    (5, 6), (5, 7), (5, 8), (5, 9),
    (6, 7), (6, 8), (6, 9),
    (7, 8), (7, 9),
    (8, 9)
])

```

3. Define a model

We will define a simple neural network model that takes a graph as input and outputs a scalar value. The model consists of three layers: an input layer, a hidden layer, and an output layer.

```

class GraphModel(nn.Module):
    def __init__(self):
        super(GraphModel, self).__init__()
        self.input_layer = nn.Linear(10, 10)
        self.hidden_layer = nn.Linear(10, 10)
        self.output_layer = nn.Linear(10, 1)
    def forward(self, x):
        x = self.input_layer(x)
        x = F.relu(x)
        x = self.hidden_layer(x)
        x = F.relu(x)
        x = self.output_layer(x)
        return x

```

4. Training and evaluation

We will train the model using a simple training loop. The training loop consists of the following steps:

- Load the training data.
- Define a loss function.
- Define an optimizer.
- Train the model for a fixed number of epochs.
- Evaluate the model on a validation set.

Apply Configuration to the Nodes

1. **Identify the nodes**
 List the nodes that are in the configuration management tool's inventory. You can use the `ansible-inventory` command to list the nodes in the inventory. The output of the command shows the nodes that are in the inventory and the groups that they belong to. You can use the `ansible` command to run a command on all the nodes in the inventory. For example, you can use the `ansible -i inventory -m shell -c ssh -u root -s -e 'cat /etc/passwd'` command to run the `cat /etc/passwd` command on all the nodes in the inventory.
2. **Apply the configuration to the nodes**
 You can use the `ansible` command to apply the configuration to the nodes. For example, you can use the `ansible -i inventory -m shell -c ssh -u root -s -e 'cat /etc/passwd'` command to run the `cat /etc/passwd` command on all the nodes in the inventory.
3. **Verify the configuration on the nodes**
 You can use the `ansible` command to verify the configuration on the nodes. For example, you can use the `ansible -i inventory -m shell -c ssh -u root -s -e 'cat /etc/passwd'` command to run the `cat /etc/passwd` command on all the nodes in the inventory.

Ansible Playbook

Ansible playbooks are YAML files that describe the configuration of a system. They are used to define the configuration of a system and to apply the configuration to the nodes. A playbook is a list of tasks that are executed on the nodes. The tasks are defined in the playbook and are executed in the order that they are defined. The tasks can be used to configure the system, to install software, to manage files, and to manage services. The tasks can also be used to verify the configuration of the system. The tasks are executed on the nodes using the `ansible` command. The output of the `ansible` command shows the results of the tasks. The output can be used to verify the configuration of the system.

```

# Example 1: A simple playbook that configures the system
---
- hosts: all
  tasks:
    - name: Install the system
      command: apt-get install -y system
    - name: Configure the system
      command: systemctl enable system
    - name: Verify the configuration
      command: systemctl status system

```

The above playbook is a simple playbook that configures the system. It consists of three tasks: `Install the system`, `Configure the system`, and `Verify the configuration`. The tasks are executed in the order that they are defined. The tasks are executed on all the nodes in the inventory. The output of the `ansible` command shows the results of the tasks. The output can be used to verify the configuration of the system.

[illegible][illegible]

3. Ergebnisinterpretation

Interpretation des Ergebnisses: Interpretation des Ergebnisses ist ein zentraler Bestandteil der Analyse. Es geht darum, die Ergebnisse in den Kontext der Forschungsfrage zu setzen und zu erklären. Dazu gehören:

- **Statistische Signifikanz:** Prüfung, ob die Ergebnisse statistisch signifikant sind (z.B. durch p-Werte).
- **Praktische Signifikanz:** Bewertung, ob die Ergebnisse in der Praxis relevant sind.
- **Limitationen:** Identifizierung der Grenzen der Studie (z.B. Stichprobengröße, Methodik).
- **Implikationen:** Ableitung von Schlussfolgerungen und Empfehlungen basierend auf den Ergebnissen.

Die Interpretation sollte klar, prägnant und verständlich sein. Sie sollte die Ergebnisse in den Kontext der Forschungsfrage setzen und die Bedeutung der Ergebnisse verdeutlichen.

Length of the process

Data Prep

Thresholds

Risk of Attempt

Direct & Quasi-identifiers

Rulesets

Variable Rules

Risk of Success

k-anonymity

Optimal scenario

Packaging

Anonymisation Report

Data Anonymisation and Risk Assessment Automation

Process in a Nutshell

In its most condensed form, the proposed anonymisation process looks as follows.

- Prepare input datasets (2.1)
- Set Thresholds (5.8)
- Assess risk of re-identification attempt – deliberate, inadvertent, breach (5.1)
- Find identifiers in the data, assign classification (3.1)
- Create BASE DATASET by extracting quasi-identifier values for all subjects (5.3)
- Create RULE DATASET by assigning possible rules to each quasi-identifier in BASE DATASET (6.3)
- For each scenario in RULE DATASET:
 - Create TRANSFORMED DATASET by applying de-identification rules to BASE DATASET (6.2)
 - Calculate risk of successful re-identification (5.5)
 - Combine risk of success and risk of attempt to find overall risk of re-identification (5.6)
 - Check k-anonymity condition (5.7)
 - Store results
- Select optimal (final) rule set based on risk calculations and request requirements (6.5)
- Apply final de-identification rules to all datasets (6.7) and convert to desired formats (6.8)
- Compile anonymised package (6.9)
- Create anonymisation report and add to the package (7)

Code examples

Example SAS® code

```
%macro AGE_BANDS(ds, var, size, start);  
  data &ds(drop=&var _lo _hi rename=( _&var=&var));  
    set &ds;  
    _lo = floor(((var - start) / size) * size + start);  
    _hi = _lo + size - 1;  
    _&var = compress(_&var, _lo _hi);  
  run;  
%mend;  
%AGE_BANDS(out, var, size, start);
```

AGE AGE
47 → 46-50

Example R code

```
age_bands <- function(var, size, start) {  
  return(paste((((var - start) %/% size  
    ) * size) + start, "-", (((var - start) %/% size  
    + 1) * size) + start))  
}
```

Example Python code

```
import math  
  
def age_bands(age, size, start):  
    lo = math.floor((age - start) / size) * size + start  
    hi = lo + size - 1  
    return "{} - {}".format(lo, hi)
```

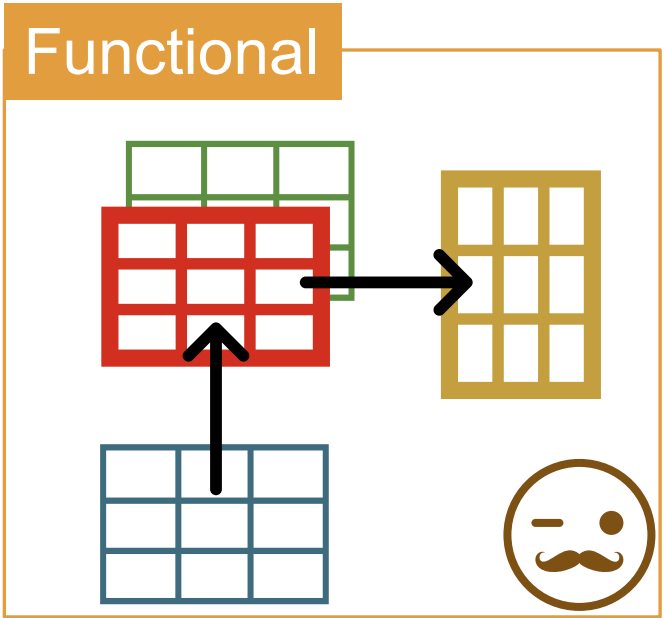
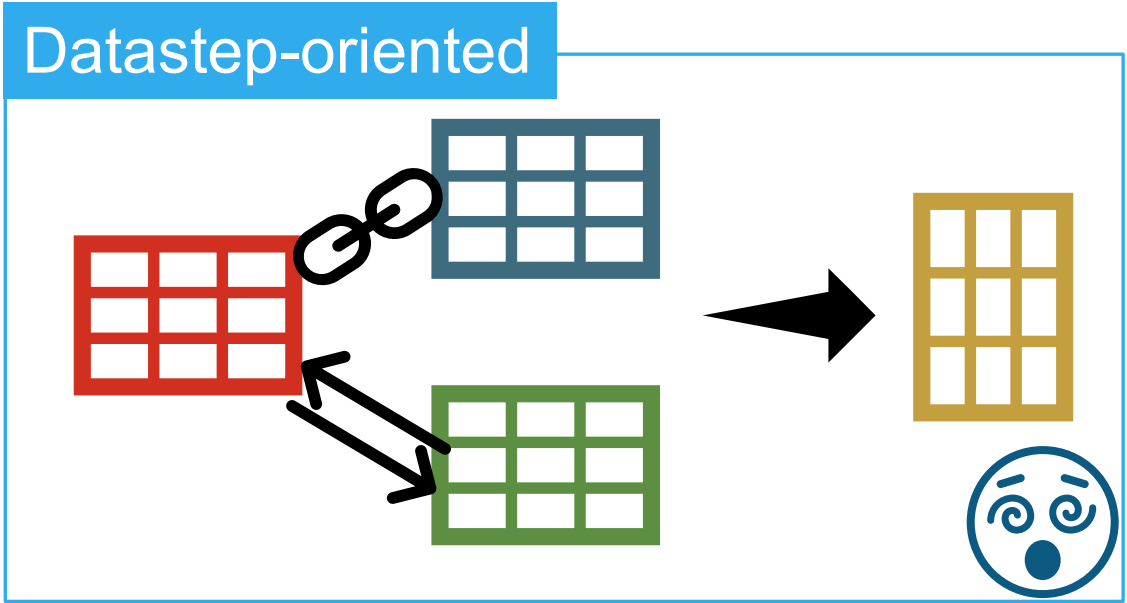
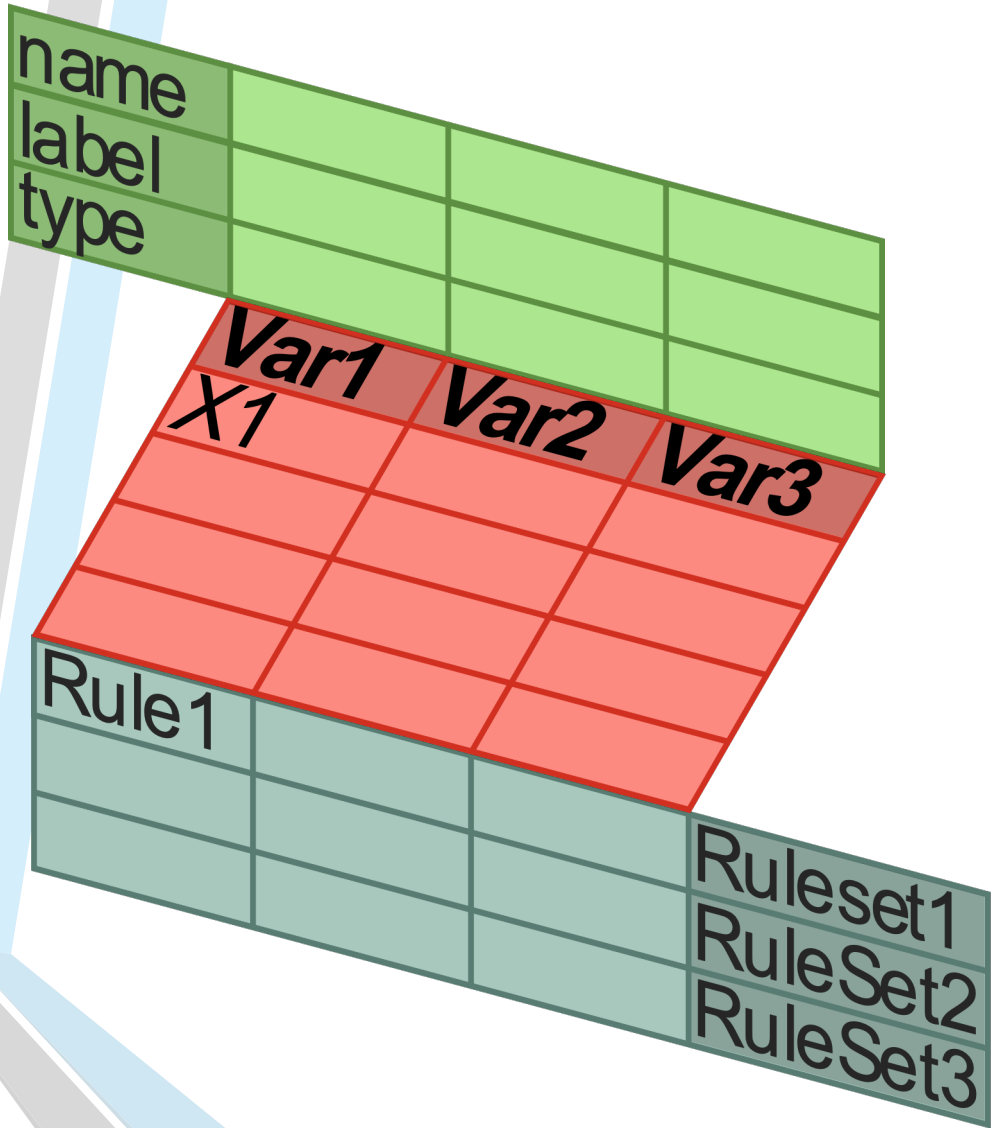


Automation

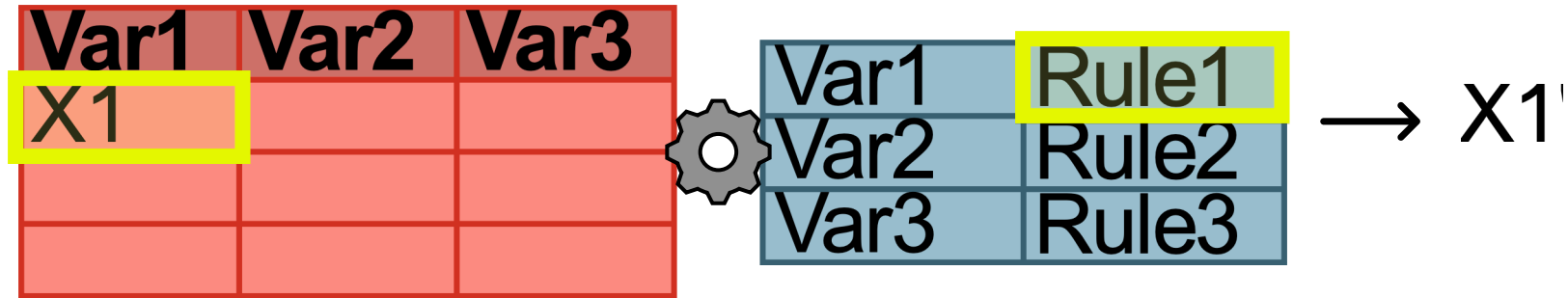
The image shows a 3D data cube with three dimensions: 'name', 'label', and 'type'. The 'name' dimension has three categories: 'Var1', 'Var2', and 'Var3'. The 'label' dimension has three categories: 'Rule1', 'RuleSet2', and 'RuleSet3'. The 'type' dimension has three categories: 'X1', 'X2', and 'X3'. The cube is colored with a gradient from light blue to light green.



Automation vs Languages



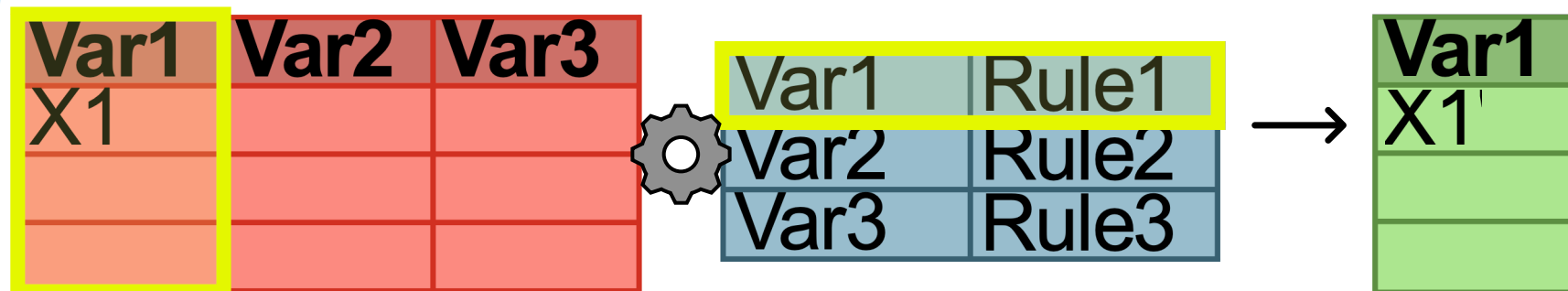
Use rule to transform value



```
function rule1(var):  
    return round(var,10)
```

```
%macro rule1(var);  
    &var = round(&var,10);  
%mend;
```

Apply rule to variable

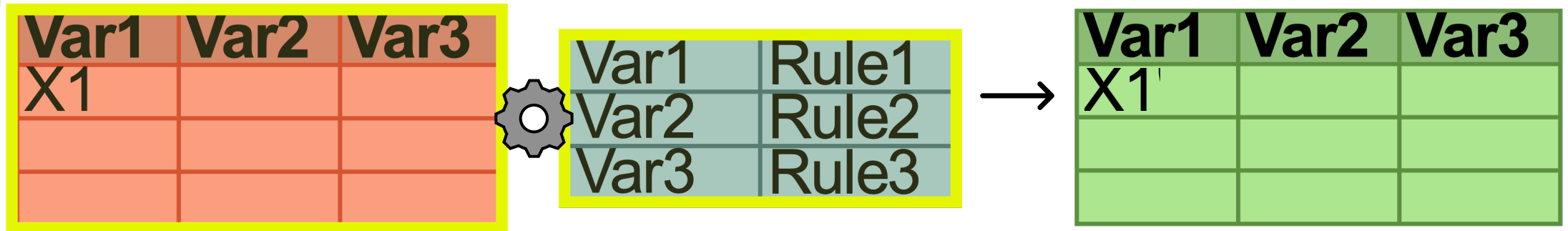


```
function apply_rule(var,*args) :  
  rule = find_rule(var)  
  return var.map(rule,*args)  
  
DS.Var1 = apply_rule(DS.Var1)
```

```
%macro apply_rule(var,rule);  
  %&rule(&var);  
%mend;  
  
data DS; set DS;  
  %apply_rule(Var1,Rule1);  
run;
```



Apply ruleset to dataset



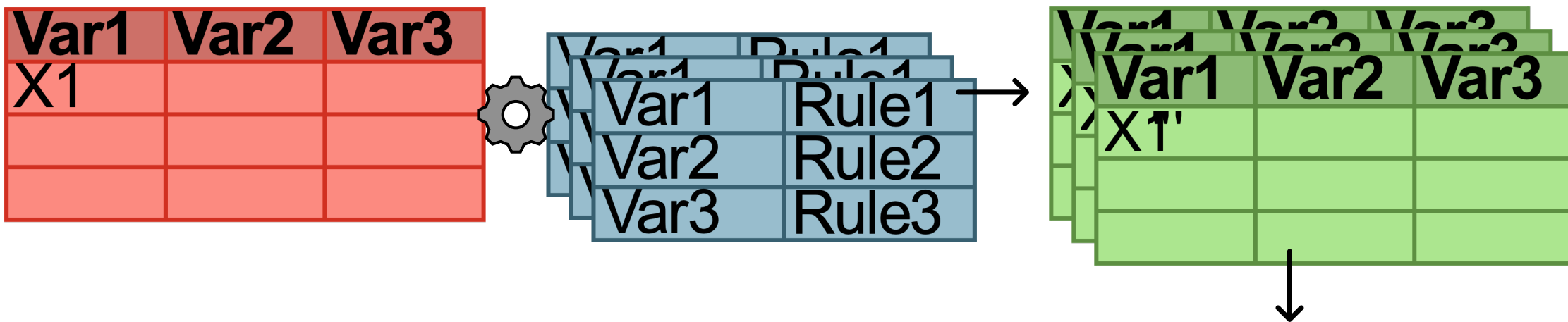
```
for var in ds:  
    var = apply_rule(var)
```

```
DS.Var1 = apply_rule(DS.Var1)  
DS.Var2 = apply_rule(DS.Var2)  
DS.Var3 = apply_rule(DS.Var3)
```

```
data DS;  
set DS;  
%apply_rule(Var1, Rule1);  
%apply_rule(Var2, Rule2);  
%apply_rule(Var3, Rule3);  
run;
```

```
data _null_;  
set RULESET end=last;  
if _N_ eq 1 then do;  
    call execute('data '||dataset||';');  
    call execute(' set '||dataset||';');  
end;  
call execute(  
    ' %apply_rule('||var||','||rule||');'  
);  
if last then call execute('run;');  
run;
```

Test multiple rulesets



```
function test_ruleset(ds, ruleset):  
    for var in ds:  
        var = apply_rule(var, ruleset)  
    return calc_risk(ds)
```

```
Risk1 = test_ruleset(DS, RuleSet1)  
Risk2 = test_ruleset(DS, RuleSet2)  
Risk3 = test_ruleset(DS, RuleSet3)
```

```
data _null_;  
    set RULESET end=last;  
    if _N_ eq 1 then do;  
        data _null_;  
            set data _null_;  
            if _N_ eq 1 then do;  
                call execute('data '||dataset||';');  
                call execute('set '||dataset||';');  
            end;  
        end;  
        call execute(' '||var||', '||rule||');  
        %apply_rule  
    );  
    if last then call execute('run;');  
run;
```

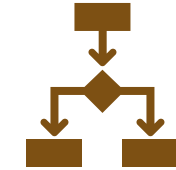
Languages:



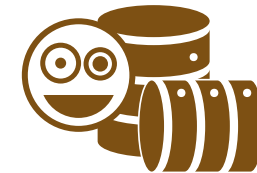
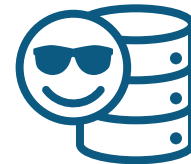
Availability



Logic



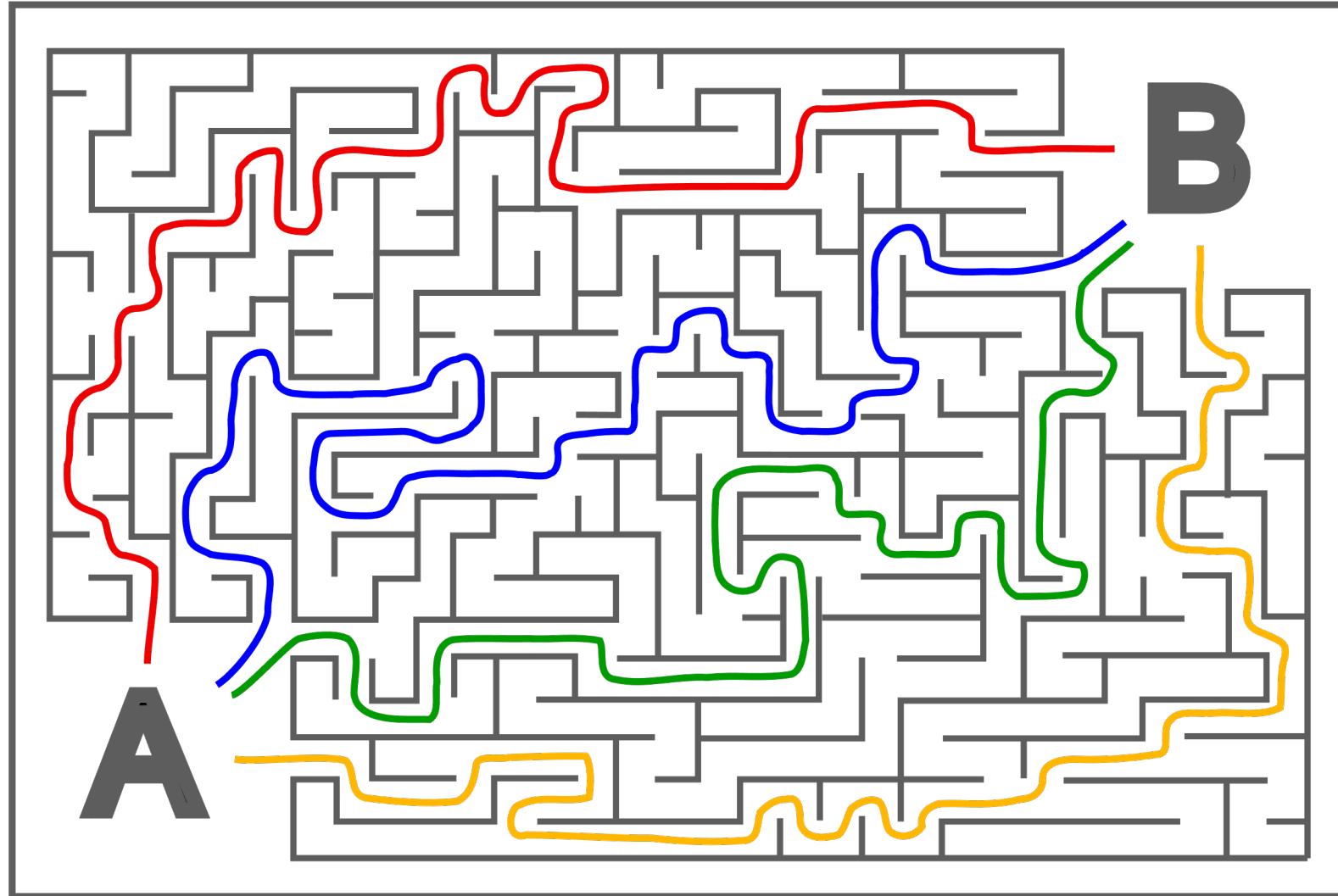
Native Data Files



Working with Other Files



Options, options, options...



Data Anonymisation and Risk Assessment Automation

Version 1.0, 01-Jul-2020

www.phuse.eu/white-papers

DH01

Lukasz Kniola

lukasz.kniola@biogen.com

