Package ‘deident’

November 19, 2024

Type Package

Title Persistent Data Anonymization Pipeline

Version 1.0.0

Description A framework for the replicable removal of personally identifiable data (PID) in data sets. The package implements a suite of methods to suit different data types based on the suggestions of Garfinkel (2015) <doi:10.6028/NIST.IR.8053> and the ICO ``Guidelines on Anonymization'' (2012) <https://ico.org.uk/media/1061/anonymisation-code.pdf>.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.2.1

Suggests testthat (>= 3.0.0), checkmate, knitr, rmarkdown, roxygen2

Imports R6, dplyr, openssl, tidyselect, rlang (>= 0.4.11), glue, purrr, stringr, yaml, readr, openxlsx, lemon, withr, fs

Depends R (>= 2.10)

VignetteBuilder knitr

Language en-GB

NeedsCompilation no

Author Robert Cook [aut, cre] (<https://orcid.org/0000-0003-3343-8271>), Md Assaduzaman [aut] (<https://orcid.org/0000-0002-8885-6721>), Sarahjane Jones [aut] (<https://orcid.org/0000-0003-4729-4029>)

Maintainer Robert Cook <robert.cook@staffs.ac.uk>

Repository CRAN

Date/Publication 2024-11-19 09:50:06 UTC

Contents

adaptive_noise 2
add_blur 3

add_encrypt . . . 4
add_group . . . 5
add_numeric_blur . . . 6
add_perturb . . . 6
add_pseudonymize . . . 7
add_shuffle . . . 8
apply_deident . . . 9
apply_to_data_frame . . . 10
BaseDeident . . . 10
Blurer . . . 11
category_blur . . . 12
create_deident . . . 12
deident . . . 13
deident_job_from_folder . . . 14
Drop . . . 14
Encrypter . . . 14
from_yaml . . . 15
GroupedShuffler . . . 16
lognorm_noise . . . 16
NumericBlurer . . . 17
Perturber . . . 17
Pseudonymizer . . . 18
ShiftsWorked . . . 18
Shuffler . . . 19
starwars . . . 20
white_noise . . . 21

Index **22**

`adaptive_noise` *Function factory to apply white noise to a vector proportional to the spread of the data*

Description

Function factory to apply white noise to a vector proportional to the spread of the data

Usage

```
adaptive_noise(sd.ratio = 1/10)
```

Arguments

`sd.ratio`	the level of noise to apply relative to the vectors standard deviation.

Value

a function

Examples

```
f <- adaptive_noise(0.2)
f(1:10)
```

`add_blur` *De-identification via categorical aggregation*

Description

`add_blur()` adds an bluring step to a transformation pipeline (NB: intended for categorical data). When ran as a transformation, values are recoded to a lower cardinality as defined by `blur`. #'

Usage

```
add_blur(object, ..., blur = c())
```

Arguments

`object`	Either a `data.frame`, `tibble`, or existing `DeidentList` pipeline.
`...`	variables to be transformed.
`blur`	a key-value pair such that 'key' is replaced by 'value' on transformation.

Value

A 'DeidentList' representing the untrained transformation pipeline. The object contains fields:

- `deident_methods` a list of each step in the pipeline (consisting of `variables` and `method`)

and methods:

- `mutate` apply the pipeline to a new data set
- `to_yaml` serialize the pipeline to a '.yml' file

See Also

`category_blur()` is provided to aid in defining the `blur`

Examples

```
.blur <- category_blur(ShiftsWorked$Shift, `Working` = "Day|Night")
pipe.blur <- add_blur(ShiftsWorked, `Shift`, blur = .blur)
pipe.blur$mutate(ShiftsWorked)
```

`add_encrypt` *De-identification via hash encryption*

Description

`add_encrypt()` adds an encryption step to a transformation pipeline. When ran as a transformation, each specified variable undergoes replacement via an encryption hashing function depending on the `hash_key` and `seed` set.

Usage

```
add_encrypt(object, ..., hash_key = "", seed = NA)
```

Arguments

`object`	Either a `data.frame`, `tibble`, or existing `DeidentList` pipeline.
`...`	variables to be transformed.
`hash_key`	a random alphanumeric key to control encryption
`seed`	a random alphanumeric to concat to the value being encrypted

Value

A 'DeidentList' representing the untrained transformation pipeline. The object contains fields:

- `deident_methods` a list of each step in the pipeline (consisting of `variables` and `method`)

and methods:

- `mutate` apply the pipeline to a new data set
- `to_yaml` serialize the pipeline to a '.yml' file

Examples

```
# Basic usage; without setting a `hash_key` or `seed` encryption is poor.
pipe.encrypt <- add_encrypt(ShiftsWorked, Employee)
pipe.encrypt$mutate(ShiftsWorked)

# Once set the encryption is more secure assuming `hash_key` and `seed` are
# not exposed.
pipe.encrypt.secure <- add_encrypt(ShiftsWorked, Employee, hash_key="hash1", seed="Seed2")
pipe.encrypt.secure$mutate(ShiftsWorked)
```

`add_group` *Add aggregation to pipelines*

Description

`add_group()` allows for the injection of aggregation into the transformation pipeline. Should you need to apply a transformation under aggregation (e.g. `add_shuffle`) this helper creates a grouped `data.frame` as would be done with `dplyr::group_by()`. The function `add_ungroup()` is supplied to perform the inverse operation.

Usage

```
add_group(object, ...)

add_ungroup(object, ...)
```

Arguments

`object`	Either a `data.frame`, `tibble`, or existing `DeidentList` pipeline.
`...`	Variables on which data is to be grouped.

Value

A 'DeidentList' representing the untrained transformation pipeline. The object contains fields:

- `deident_methods` a list of each step in the pipeline (consisting of `variables` and `method`)

and methods:

- `mutate` apply the pipeline to a new data set
- `to_yaml` serialize the pipeline to a '.yml' file

Examples

```
pipe.grouped <- add_group(ShiftsWorked, Date, Shift)
pipe.grouped_shuffle <- add_shuffle(pipe.grouped, `Daily Pay`)
add_ungroup(pipe.grouped_shuffle, `Daily Pay`)
```

`add_numeric_blur` *De-identification via numeric aggregation*

Description

`add_numeric_blur()` adds an bluring step to a transformation pipeline (NB: intended for numeric data). When ran as a transformation, the data is split into intervals depending on the `cuts` supplied of the series [-Inf, cut.1), [cut.1, cut.2), ..., [cut.n, Inf] where `cuts` = c(cut.1, cut.2, ..., cut.n).

Usage

```
add_numeric_blur(object, ..., cuts = 0)
```

Arguments

`object`	Either a `data.frame`, `tibble`, or existing `DeidentList` pipeline.
`...`	variables to be transformed.
`cuts`	The position in which data is to be divided.

Value

A 'DeidentList' representing the untrained transformation pipeline. The object contains fields:

- `deident_methods` a list of each step in the pipeline (consisting of `variables` and `method`)

and methods:

- `mutate` apply the pipeline to a new data set
- `to_yaml` serialize the pipeline to a '.yml' file

`add_perturb` *De-identification via random noise*

Description

`add_perturb()` adds an perturbation step to a transformation pipeline (NB: intended for numeric data). When ran as a transformation, each specified variable is transformed by the `noise` function.

Usage

```
add_perturb(object, ..., noise = adaptive_noise(0.1))
```

Arguments

`object`	Either a `data.frame`, `tibble`, or existing `DeidentList` pipeline.
`...`	variables to be transformed.
`noise`	a single-argument function that applies randomness.

Value

A 'DeidentList' representing the untrained transformation pipeline. The object contains fields:

- `deident_methods` a list of each step in the pipeline (consisting of `variables` and `method`)

and methods:

- `mutate` apply the pipeline to a new data set
- `to_yaml` serialize the pipeline to a '.yml' file

See Also

`adaptive_noise()`, `white_noise()`, and `lognorm_noise()`

Examples

```
pipe.perturb <- add_perturb(ShiftsWorked, `Daily Pay`)
pipe.perturb$mutate(ShiftsWorked)

pipe.perturb.white_noise <- add_perturb(ShiftsWorked, `Daily Pay`, noise=white_noise(0.1))
pipe.perturb.white_noise$mutate(ShiftsWorked)

pipe.perturb.noisy_adaptive <- add_perturb(ShiftsWorked, `Daily Pay`, noise=adaptive_noise(1))
pipe.perturb.noisy_adaptive$mutate(ShiftsWorked)
```

`add_pseudonymize` *De-identification via replacement*

Description

`add_pseudonymize()` adds a psuedonymization step to a transformation pipeline. When ran as a transformation, terms that have not been seen before are given a new random alpha-numeric string while terms that have been previously transformed reuse the same term.

Usage

```
add_pseudonymize(object, ..., lookup = list())
```

Arguments

`object`	Either a `data.frame`, `tibble`, or existing `DeidentList` pipeline.
`...`	variables to be transformed.
`lookup`	a pre-existing name-value pair to define intended psuedonymizations. Instances of 'name' will be replaced with 'value' on transformation.#'

Value

A 'DeidentList' representing the untrained transformation pipeline. The object contains fields:

- `deident_methods` a list of each step in the pipeline (consisting of `variables` and `method`)

and methods:

- `mutate` apply the pipeline to a new data set
- `to_yaml` serialize the pipeline to a '.yml' file

Examples

```
# Basic usage;
pipe.pseudonymize <- add_pseudonymize(ShiftsWorked, Employee)
pipe.pseudonymize$mutate(ShiftsWorked)

pipe.pseudonymize2 <- add_pseudonymize(ShiftsWorked, Employee,
                                      lookup=list("Kyle Wilson" = "Kyle"))
pipe.pseudonymize2$mutate(ShiftsWorked)
```

`add_shuffle` *De-identification via random sampling*

Description

`add_shuffle()` adds a shuffling step to a transformation pipeline. When ran as a transformation, each specified variable undergoes a random sample without replacement so that summary metrics on a single variable are unchanged, but inter-variable metrics are rendered spurious.

Usage

```
add_shuffle(object, ..., limit = 0)
```

Arguments

`object`	Either a `data.frame`, `tibble`, or existing `DeidentList` pipeline.
`...`	variables to be transformed.
`limit`	integer - the minimum number of observations a variable needs to have for shuffling to be performed. If the variable has length less than `limit` values are replaced with `NA`s.

Value

A ’DeidentList’ representing the untrained transformation pipeline. The object contains fields:

- `deident_methods` a list of each step in the pipeline (consisting of `variables` and `method`)

and methods:

- `mutate` apply the pipeline to a new data set
- `to_yaml` serialize the pipeline to a ’.yml’ file

See Also

`add_group()` for usage under aggregation

Examples

```
# Basic usage;
pipe.shuffle <- add_shuffle(ShiftsWorked, Employee)
pipe.shuffle$mutate(ShiftsWorked)

pipe.shuffle.limit <- add_shuffle(ShiftsWorked, Employee, limit=1)
pipe.shuffle.limit$mutate(ShiftsWorked)
```

`apply_deident` *Apply a ’deident’ pipeline*

Description

Applies a pipeline as defined by `deident` to a data frame. tibble, or file.

Usage

```
apply_deident(object, deident, ...)
```

Arguments

`object`	The data to be deidentified
`deident`	A deidentification pipeline to be used.
`...`	Terms to be passed to other methods

`apply_to_data_frame` *Apply a 'deident' pipeline to a new data frame*

Description

Apply a 'deident' pipeline to a new data frame

Usage

```
apply_to_data_frame(data, transformer, ...)
```

Arguments

`data`	The data set to be converted
`transformer`	The pipeline to be used
`...`	To be passed on to other methods

`BaseDeident` *Base class for all De-identifier classes*

Description

Create new Deidentifier object

Setter for 'method' field

Save 'Deidentifier' to serialized object.

Apply 'method' to a vector of values

Apply 'method' to variables in a data frame

Apply 'mutate' method to an aggregated data frame.

Aggregate a data frame and apply 'mutate' to each.

Convert `self` to a list

String representation of `self`

Check if parameters are in allowed fields

Arguments

`method`	New function to be used as the method.
`location`	File path to save to.
`keys`	Vector of values to be processed
`force`	Perform transformation on all variables even if some given are not in the data.
`grouped_data`	a 'grouped_df' object

`data`	A data frame to be manipulated
`grp_cols`	Vector of variables in 'data' to group on.
`mutate_cols`	Vector of variables in 'data' to transform.
`type`	character vector describing the object. Defaults to class.
`...`	Options to check exist

Fields

`method` Function to call for data transform.

`Blurer` *Deidentifier class for applying 'blur' transform*

Description

Convert `self` to a list.

Arguments

`blur`	Look-up list to define aggregation.
`keys`	Vector of values to be processed
`...`	Values to be concatenated to keys

Details

'Bluring' refers to aggregation of data e.g. converting city to country, or post code to IMD. The level of blurring is defined by the list given at initialization which maps key to value e.g. list(London = "England", Paris = "France").

Value

`Blurer` Apply blur to a vector of values

Fields

`blur` List of aggregations to be applied. Create new Blurer object

`category_blur` *Utility for producing 'blur'*

Description

Utility for producing 'blur'

Usage

```
category_blur(vec, ...)
```

Arguments

`vec`	The vector of values to be used
`...`	`Replacement` = `RegexPattern` pairs of arguments

`create_deident` *Create a deident pipeline*

Description

Create a deident pipeline

Usage

```
create_deident(method, ...)
```

Arguments

`method`	A deidentifier to initialize.
`...`	list of variables to be deidentifier. NB: key word arguments will be passed to method at initialization.

`deident`	*Define a transformation pipeline*

Description

`deident()` creates a transformation pipeline of 'deidentifiers' for the repeated application of anonymization transformations.

Usage

```
deident(data, deidentifier, ...)
```

Arguments

`data`	A data frame, existing pipeline, or a 'deidentifier' (as either initialized object, class generator, or character string)
`deidentifier`	A deidentifier' (as either initialized object, class generator, or character string) to be appended to the current pipeline
`...`	Positional arguments are variables of 'data' to be transformed and key-word arguments are passed to 'deidentifier' at creation

Value

A 'DeidentList' representing the untrained transformation pipeline. The object contains fields:

- `deident_methods` a list of each step in the pipeline (consisting of `variables` and `method`)

and methods:

- `mutate` apply the pipeline to a new data set
- `to_yaml` serialize the pipeline to a '.yml' file

Examples

```
#
pipe <- deident(ShiftsWorked, Pseudonymizer, Employee)

print(pipe)

apply_deident(ShiftsWorked, pipe)
```

deident_job_from_folder — *Apply a pipeline to files on disk.*

Description

Apply a deident pipeline to a set of files and save them back to disk

Usage

```
deident_job_from_folder(
  deident_pipeline,
  data_dir,
  result_dir = "Deident_results"
)
```

Arguments

`deident_pipeline`	The deident list to be used.
`data_dir`	a path to the files to be transformed.
`result_dir`	a path to where files are to be saved.

Drop — *R6 class for the removal of variables from a pipeline*

Description

A `Deident` class dealing with the exclusion of variables.

Encrypter — *Deidentifier class for applying 'encryption' transform*

Description

Create new Encrypter object

Convert `self` to a list.

Arguments

`hash_key`	An alpha numeric key for use in encryption
`seed`	An alpha numeric key which is concatenated to minimize brute force attacks
`keys`	Vector of values to be processed
`...`	Values to be concatenated to keys

Details

'Encrypting' refers to the cryptographic hashing of data e.g. md5 checksum. Encryption is more powerful if a random hash and seed are supplied and kept secret.

Value

`Encrypter` Apply blur to a vector of values

Fields

`hash_key` Alpha-numeric secret key for encryption

`seed` String for concatenation to raw value

`from_yaml` *Restore a serialized deident from file*

Description

Restore a serialized deident from file

Usage

```
from_yaml(path)
```

Arguments

`path`	Path to serialized deident.

Examples

```
deident <- deident(ShiftsWorked, Pseudonymizer, Employee)
.tempfile <- tempfile(fileext = ".yml")
deident$to_yaml(.tempfile)

deident.yaml <- from_yaml(.tempfile)
deident.yaml$mutate(ShiftsWorked)
```

`GroupedShuffler`	*GroupedShuffler class for applying 'shuffling' transform with data aggregated*

Description

Convert `self` to a list.

Character representation of the class

Arguments

`limit`	Minimum number of rows required to shuffle data
`data`	A data frame to be manipulated
`...`	Vector of variables in 'data' to transform.

Details

'Shuffling' refers to the a random sampling of a variable without replacement e.g. [A, B, C] becoming [B, A, C] but not [A, A, B]. "Grouped shuffling" refers to aggregating the data by another feature before applying the shuffling process. Grouped shuffling will preserve aggregate level metrics (e.g. mean, median, mode) but removes ordinal properties i.e. correlations and auto-correlations

Fields

`group_on` Symbolic representation of grouping variables

`limit` Minimum number of rows required to shuffle data Create new GroupedShuffler object

`lognorm_noise`	*Function factory to apply log-normal noise to a vector*

Description

Function factory to apply log-normal noise to a vector

Usage

```
lognorm_noise(sd = 0.1)
```

Arguments

`sd`	the standard deviation of noise to apply.

Value

a function

Examples

```
f <- lognorm_noise(1)
f(1:10)
```

`NumericBlurer` *Group numeric data into baskets*

Description

Group numeric data into baskets

`Perturber` *R6 class for deidentification via random noise*

Description

A `Deident` class dealing with the addition of random noise to a numeric variable.

Create new Perturber object

Apply noise to a vector of values

Convert `self` to a list.

Character representation of the class

Arguments

`noise`	a single-argument function that applies randomness.
`keys`	Vector of values to be processed
`...`	Values to be concatenated to keys

Fields

`noise.str` character representation of `noise`

`method` random noise function

Examples

```
  pert <- Perturber$new()
  pert$transform(1:10)
```

`Pseudonymizer` *R6 class for deidentification via replacement*

Description

A `Deident` class dealing with the (repeatable) random replacement of string for deidentification.

Create new `Pseudonymizer` object

Check if a key exists in `lookup`

Check if a key exists in `lookup`

Retrieve a value from `lookup`

Returns `self$lookup` formatted as a tibble

Convert `self` to a list.

Apply the deidentifcation method to the supplied `keys`

Arguments

`lookup`	a pre-existing name-value pair to define intended psuedonymizations. Instances of 'name' will be replaced with 'value' on transformation.
`keys`	value to be checked
`...`	values to concatenate to `key` and check
`parse_numerics`	True: Force columns to characters. NB: only character vectors will be parsed.

Fields

`lookup` list of mapping from key-value on transform.

`ShiftsWorked` *Synthetic data set listing daily shift pattern for fictitious employees*

Description

A synthetic data set intended to demonstrate the design and application of a deidentification pipeline. Employee names are entirely fictitious and constructed from the `FiveThirtyEight Most Common Name Dataset`.

Usage

```
ShiftsWorked
```

Format

A data frame with 3,100 rows and 6 columns:

Record ID Table primary key (integer)

Employee Name of listed employee

Date The date being considered

Shift The shift-type done by `employee` on `date`. One of 'Day', 'Night' or 'Rest'.

Shift Start Shift start time (missing if on 'Rest' shift)

Shift End Shift end time (missing if on 'Rest' shift)

Daily Pay Shift end time (missing if on 'Rest' shift)

`Shuffler` *Shuffler class for applying 'shuffling' transform*

Description

Create new Shuffler object

Update minimum vector size for shuffling

Apply the deidentifcation method to the supplied `keys`

Convert `self` to a list.

Arguments

`method`	[optional] A function representing the method of re-sampling to be used. By default uses exhaustive sampling without replacement.
`keys`	Value(s) to be transformed.
`...`	Value(s) to concatenate to `keys` and transform @inheritParams Pseudonymizer
`limit`	integer - the minimum number of observations a variable needs to have for shuffling to be performed. If the variable has length less than `limit` values are replaced with `NA`s.

Details

'Shuffling' refers to the a random sampling of a variable without replacement e.g. [A, B, C] becoming [B, A, C] but not [A, A, B]. Shuffling will preserve top level metrics (e.g. mean, median, mode) but removes ordinal properties i.e. correlations and auto-correlations

Fields

`limit` minimum vector length to be shuffled. If vector to be transformed has length < limit, the data is replaced with NAs

`starwars` *Starwars characters*

Description

The original data, from SWAPI, the Star Wars API, https://swapi.py4e.com/, has been revised to reflect additional research into gender and sex determinations of characters. NB: taken from `dplyr`

Usage

```
starwars
```

Format

A tibble with 87 rows and 14 variables:

name Name of the character

height Height (cm)

mass Weight (kg)

hair_color,skin_color,eye_color Hair, skin, and eye colors

birth_year Year born (BBY = Before Battle of Yavin)

sex The biological sex of the character, namely male, female, hermaphroditic, or none (as in the case for Droids).

gender The gender role or gender identity of the character as determined by their personality or the way they were programmed (as in the case for Droids).

homeworld Name of homeworld

species Name of species

films List of films the character appeared in

vehicles List of vehicles the character has piloted

starships List of starships the character has piloted

Examples

```
starwars
```

`white_noise` *Function factory to apply white noise to a vector*

Description

Function factory to apply white noise to a vector

Usage

```
white_noise(sd = 0.1)
```

Arguments

`sd`	the standard deviation of noise to apply.

Value

a function

Examples

```
f <- white_noise(1)
f(1:10)
```

Index

∗ **API**
- add_blur, 3
- add_encrypt, 4
- add_group, 5
- add_numeric_blur, 6
- add_perturb, 6
- add_pseudonymize, 7
- add_shuffle, 8

∗ **datasets**
- ShiftsWorked, 18
- starwars, 20

adaptive_noise, 2
adaptive_noise(), *7*
add_blur, 3
add_encrypt, 4
add_group, 5
add_group(), *9*
add_numeric_blur, 6
add_perturb, 6
add_pseudonymize, 7
add_shuffle, 8
add_ungroup *(*add_group*)*, 5
apply_deident, 9
apply_to_data_frame, 10

BaseDeident, 10
Blurer, 11

category_blur, 12
category_blur(), *3*
create_deident, 12

deident, 13
deident_job_from_folder, 14
dplyr::group_by(), *5*
Drop, 14

Encrypter, 14

from_yaml, 15

GroupedShuffler, 16

lognorm_noise, 16
lognorm_noise(), *7*

NumericBlurer, 17

Perturber, 17
Pseudonymizer, 18

ShiftsWorked, 18
Shuffler, 19
starwars, 20

white_noise, 21
white_noise(), *7*