Package: rCISSVAE (via r-universe)

November 10, 2026

Title Clustering-Informed Shared-Structure VAE for Imputation

Version 1.0.1

Maintainer Danielle Vaithilingam <vaithid1@mskcc.org>

Description Implements the Clustering-Informed Shared-Structure Variational Autoencoder ('CISS-VAE'), a deep learning framework for missing data imputation introduced in Khadem Charvadeh et al. (2025) <doi:10.1002/sim.70335>. The model accommodates all three types of missing data mechanisms: Missing Completely At Random (MCAR), Missing At Random (MAR), and Missing Not At Random (MNAR). While it is particularly well-suited to MNAR scenarios, where missingness patterns carry informative signals, 'CISS-VAE' also functions effectively under MAR assumptions.

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 4.2.0)

Imports reticulate, purrr, gtsummary, rlang, ComplexHeatmap, stats

Suggests testthat (>= 3.0.0), dplyr, knitr, rmarkdown, tidyverse, kableExtra, MASS, fastDummies, palmerpenguins, glue, withr, ggplot2

URL https://ciss-vae.github.io/rCISS-VAE/

BugReports https://github.com/CISS-VAE/rCISS-VAE/issues

Config/testthat/edition 3

VignetteBuilder knitr

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

LazyData true

Config/pak/sysreqs cmake make libicu-dev libpng-dev libuv1-dev libxml2-dev libssl-dev perl python3 libnode-dev

Repository https://ciss-vae.r-universe.dev

Date/Publication 2026-05-14 13:12:39 UTC

RemoteUrl https://github.com/ciss-vae/rciss-vae

RemoteRef HEAD

RemoteSha b13cacd85446e69e417d5ab674b11bd161675845

Contents

autotune_cissvae . . . 2
check_devices . . . 7
cluster_heatmap . . . 7
cluster_on_missing . . . 9
cluster_on_missing_prop . . . 10
cluster_summary . . . 11
clusters . . . 13
create_cissvae_env . . . 13
create_missingness_prop_matrix . . . 14
df_missing . . . 16
dni . . . 16
impute_result . . . 17
impute_with_cissvae . . . 18
load_cissvae_model . . . 19
load_impute_result . . . 21
mock_surv . . . 21
performance_by_cluster . . . 22
plot_vae_architecture . . . 24
print.impute_result . . . 26
run_cissvae . . . 27
save_cissvae_model . . . 32
save_impute_result . . . 33
update_cissvae_env . . . 35

Index **37**

`autotune_cissvae` *Autotune CISS-VAE hyperparameters with Optuna*

Description

Performs hyperparameter optimization for CISS-VAE using Optuna with support for both tunable and fixed parameters.

Usage

```
autotune_cissvae(
  data,
  index_col = NULL,
  val_proportion = 0.1,
  replacement_value = 0,
  cols_ignore = NULL,
  imputable_matrix = NULL,
  binary_feature_mask = NULL,
  categorical_column_map = NULL,
  clusters,
  save_model_path = NULL,
  save_search_space_path = NULL,
  n_trials = 20,
  study_name = "vae_autotune",
  device_preference = "cuda",
  show_progress = FALSE,
  optuna_dashboard_db = NULL,
  load_if_exists = TRUE,
  seed = 42,
  verbose = FALSE,
  constant_layer_size = FALSE,
  evaluate_all_orders = FALSE,
  max_exhaustive_orders = 100,
  num_hidden_layers = c(1, 4),
  hidden_dims = c(64, 512),
  latent_dim = c(10, 100),
  latent_shared = c(TRUE, FALSE),
  output_shared = c(TRUE, FALSE),
  lr = c(1e-04, 0.001),
  decay_factor = c(0.9, 0.999),
  weight_decay = 0.001,
  beta = 0.01,
  num_epochs = 500,
  batch_size = 4000,
  num_shared_encode = c(0, 1, 3),
  num_shared_decode = c(0, 1, 3),
  encoder_shared_placement = c("at_end", "at_start", "alternating", "random"),
  decoder_shared_placement = c("at_start", "at_end", "alternating", "random"),
  refit_patience = 2,
  refit_loops = 100,
  epochs_per_loop = 500,
  reset_lr_refit = c(TRUE, FALSE),
  debug = FALSE,
  columns_ignore = NULL
)
```

Arguments

`data`	Data frame or matrix containing the input data
`index_col`	String name of index column to preserve (optional)
`val_proportion`	Proportion of non-missing data to hold out for validation.
`replacement_value`	Numeric value used to replace missing entries before model input.
`cols_ignore`	Character vector of column names to exclude from imputation scoring.
`imputable_matrix`	Logical matrix indicating entries allowed to be imputed.
`binary_feature_mask`	Logical vector marking which columns are binary.
`categorical_column_map`	Named list indicating what categorical column dummy variables belong to.
`clusters`	Integer vector specifying cluster assignments for each row.
`save_model_path`	Optional path to save the best model's state_dict
`save_search_space_path`	Optional path to save search space configuration
`n_trials`	Number of Optuna trials to run
`study_name`	Name identifier for the Optuna study
`device_preference`	Preferred device ("cuda", "mps", "cpu")
`show_progress`	Whether to display Rich progress bars during training
`optuna_dashboard_db`	RDB storage URL/file for Optuna dashboard
`load_if_exists`	Whether to load existing study from storage
`seed`	Base random seed for reproducible results
`verbose`	Whether to print detailed diagnostic information
`constant_layer_size`	Whether all hidden layers use same dimension
`evaluate_all_orders`	Whether to test all possible layer arrangements
`max_exhaustive_orders`	Max arrangements to test when evaluate_all_orders = TRUE
`num_hidden_layers`	Numeric(2) vector: (min, max) for number of hidden layers
`hidden_dims`	Numeric vector: hidden layer dimensions to test
`latent_dim`	Numeric(2) vector: (min, max) for latent dimension
`latent_shared`	Logical vector: whether latent space is shared across clusters
`output_shared`	Logical vector: whether output layer is shared across clusters
`lr`	Numeric(2) vector: (min, max) learning rate range

`decay_factor`	Numeric(2) vector: (min, max) LR decay factor range
`weight_decay`	Weight decay (L2 penalty) used in Adam optimizer.
`beta`	Numeric: KL divergence weight (fixed or range)
`num_epochs`	Integer: number of initial training epochs (fixed or range)
`batch_size`	Integer: mini-batch size (fixed or range)
`num_shared_encode`	Numeric vector: numbers of shared encoder layers to test
`num_shared_decode`	Numeric vector: numbers of shared decoder layers to test
`encoder_shared_placement`	Character vector: placement strategies for encoder shared layers
`decoder_shared_placement`	Character vector: placement strategies for decoder shared layers
`refit_patience`	Integer: early stopping patience for refit loops
`refit_loops`	Integer: maximum number of refit loops
`epochs_per_loop`	Integer: epochs per refit loop
`reset_lr_refit`	Logical vector: whether to reset LR before refit
`debug`	Logical; if TRUE, additional metadata is returned for debugging.
`columns_ignore`	Alias of cols_ignore. Kept for continuity.

Value

A named list with the following components:

imputed_dataset A data frame containing the imputed values.

model The fitted CISS-VAE model object

cluster_dataset The ClusterDataset object used

clusters The vector of cluster assignments

study An optuna study object containing the trial results

results A data frame of trial results

val_data Validation dataset used

val_imputed Imputed values of validation dataset

Tips

- Use `cluster_on_missing()` or `cluster_on_missing_prop()` for cluster assignments.
- Use GPU computation when available; call `check_devices()` to see available devices.
- Adjust `batch_size` based on memory (larger is faster but uses more memory).
- Set `verbose = TRUE` or `show_progress = TRUE` to monitor training.
- Explore the `optuna-dashboard` (see vignette `optunadb`) for hyperparameter importance.
- For binary features, set `names(binary_feature_mask) <- colnames(data)`.

Examples

```
## Requires a working Python environment via reticulate
## Examples are wrapped in try() to avoid failures on CRAN check systems
try({
reticulate::use_virtualenv("cissvae_environment", required = TRUE)


data(df_missing)
data(clusters)

## Run autotuning
aut <- autotune_cissvae(
  data = df_missing,
  index_col = "index",
  clusters = clusters$clusters,
  n_trials = 3,
  study_name = "comprehensive_vae_autotune",
  device_preference = "cpu",
  seed = 42,

  ## Hyperparameter search space
  num_hidden_layers = c(2, 5),
  hidden_dims = c(64, 512),
  latent_dim = c(10, 100),
  latent_shared = c(TRUE, FALSE),
  output_shared = c(TRUE, FALSE),
  lr = c(0.01, 0.1),
  decay_factor = c(0.99, 1.0),
  beta = c(0.01, 0.1),
  num_epochs = c(5, 20),
  batch_size = c(1000, 4000),
  num_shared_encode = c(0, 1, 2),
  num_shared_decode = c(0, 1, 2),

  ## Placement strategies
  encoder_shared_placement = c(
    "at_end", "at_start",
    "alternating", "random"
  ),
  decoder_shared_placement = c(
    "at_start", "at_end",
    "alternating", "random"
  ),

  refit_patience = 2,
  refit_loops = 10,
  epochs_per_loop = 5,
  reset_lr_refit = c(TRUE, FALSE)
)

## Visualize architecture
plot_vae_architecture(
```

```
  aut$model,
  title = "Optimized CISSVAE Architecture"
)
})
```

`check_devices` *Check PyTorch device availability*

Description

This function prints the available devices (cpu, cuda, mps) detected by PyTorch. If your mps/cuda device is not shown, check your PyTorch installation.

Usage

```
check_devices(env_path = NULL)
```

Arguments

`env_path`	Path to virtual environment containing PyTorch and ciss-vae. Defaults to NULL.

Value

Vector of strings for available devices.

Examples

```
try(
check_devices()
)
```

`cluster_heatmap` *Cluster-wise Heatmap of Missing Data Patterns*

Description

Visualize the pattern of missing values in a dataset, arranged by cluster. Each column in the heatmap represents one observation and each row a feature. Tiles indicate whether a value is missing (black) or present (white). Cluster labels are shown as a column annotation bar above the heatmap. The package **ComplexHeatmap** must be installed for this function to work.

Usage

```
cluster_heatmap(
  data,
  clusters,
  cols_ignore = NULL,
  show_row_names = TRUE,
  missing_color = "yellow",
  observed_color = "purple4",
  title = "Missingness Heatmap by Cluster"
)
```

Arguments

data	A data.frame or tibble containing the dataset with possible missing values. Rows represent observations and columns represent features.
clusters	A vector of cluster labels for each observation (row) in data. Must have the same length as nrow(data).
cols_ignore	Optional character vector of column names in data to exclude from the heatmap (e.g., identifiers or non-feature columns).
show_row_names	Logical. If TRUE, displays feature names on plot
missing_color	Display color of missing values. Default black.
observed_color	Display color of observed values. Default white.
title	Optional plot title. Defaults to "Missingness Heatmap by Cluster"

Details

This function constructs a binary missingness matrix where 1 indicates a missing value and 0 a present value. Columns (observations) are ordered by their cluster labels, and the function displays a heatmap of missingness patterns using **ComplexHeatmap**. Cluster membership is displayed as an annotation above the heatmap.

Value

A list of class "ComplexHeatmap" containing the heatmap object. This can be used for further inspection or manual redraw.

Examples

```
if(requireNamespace("ComplexHeatmap")){
# Simple example with small dataset
df <- data.frame(
  x1 = c(1, NA, 3),
  x2 = c(NA, 2, 3),
  x3 = c(1, 2, NA)
)
cl <- c("A", "B", "A")
cluster_heatmap(df, cl)
```

```
# Example excluding a column prior to plotting
cluster_heatmap(df, cl, cols_ignore = "x2")

# Adding a 'Cluster' label and changing colors
cluster_heatmap(df, clusters = paste0("Cluster ", cl), cols_ignore = "x2",
missing_color = "red", observed_color = "blue")
}
```

`cluster_on_missing` *Cluster on Missingness Patterns*

Description

Given an R data.frame or matrix with missing values, clusters on the pattern of missingness and returns cluster labels plus silhouette score.

Usage

```
cluster_on_missing(
  data,
  cols_ignore = NULL,
  n_clusters = NULL,
  seed = 42,
  k_neighbors = NULL,
  leiden_resolution = 0.25,
  leiden_objective = "CPM",
  use_snn = TRUE,
  columns_ignore = NULL
)
```

Arguments

`data`	A data.frame or matrix (samples × features), may contain `NA`.
`cols_ignore`	Character vector of column names to ignore when clustering.
`n_clusters`	Integer; if provided, will run KMeans with this many clusters. If `NULL`, will use Leiden.
`seed`	Integer; random seed for KMeans (or reproducibility in Leiden).
`k_neighbors`	Integer; minimum cluster size for Leiden. If `NULL`, defaults to `nrow(data) %/% 25`.
`leiden_resolution`	Resolution for Leiden Clustering.
`leiden_objective`	objective
`use_snn`	use snn
`columns_ignore`	Alias for cols_ignore. Kept for continuity.

Value

A list with components:

- `clusters` — integer vector of cluster labels
- `silhouette` — numeric silhouette score, or `NA` if not computable

`cluster_on_missing_prop` *Cluster Samples Based on Missingness Proportions*

Description

Groups **samples** with similar patterns of missingness across features using either K-means clustering (when `n_clusters` is specified) or Leiden (when `n_clusters` is `NULL`). This is useful for detecting cohorts with shared missing-data behavior (e.g., site/batch effects).

Usage

```
cluster_on_missing_prop(
  prop_matrix,
  n_clusters = NULL,
  seed = NULL,
  k_neighbors = NULL,
  leiden_resolution = 0.25,
  use_snn = TRUE,
  leiden_objective = "CPM",
  metric = "euclidean",
  scale_features = FALSE
)
```

Arguments

`prop_matrix`	Matrix or data frame where **rows are samples** and **columns are features**, entries are missingness proportions in `[0,1]`. Can be created with `create_missingness_prop_matrix()`.
`n_clusters`	Integer; number of clusters for KMeans. If `NULL`, uses Leiden (default: `NULL`).
`seed`	Integer; random seed for KMeans reproducibility (default: `NULL`).
`k_neighbors`	Integer; Leiden minimum cluster size. If `NULL`, Python default is used (default: `NULL`).
`leiden_resolution`	Numeric; Leiden cluster selection threshold (default: `0.25`).
`use_snn`	Logical; whether to use shared nearest neighbors (optional).
`leiden_objective`	Character; Leiden optimization objective (optional).
`metric`	Character; distance metric. Options include: `"euclidean"`, `"cosine"` (default: `"euclidean"`).
`scale_features`	Logical; whether to standardize **feature columns** before clustering samples (default: `FALSE`).

Value

A list with:

- `clusters`: Integer vector of cluster assignments per **sample**.
- `silhouette_score`: Numeric silhouette score, or `NULL` if not computable.

Examples

```
set.seed(123)

dat <- data.frame(
  sample_id = paste0("s", 1:12),
  # Two features measured at 3 timepoints each -> proportions by feature
  A_1 = c(NA, rnorm(11)),
  A_2 = c(NA, rnorm(11)),
  A_3 = rnorm(12),
  B_1 = rnorm(12),
  B_2 = c(rnorm(10), NA, NA),
  B_3 = rnorm(12)
)

pm <- create_missingness_prop_matrix(
  dat,
  index_col = "sample_id",
  repeat_feature_names = c("A", "B")
)

## cluster_on_missing_prop requires a working Python environment via reticulate
## Examples are wrapped in try() to avoid failures on CRAN check systems
try({
res <- cluster_on_missing_prop(
  pm,
  n_clusters = 2,
  metric = "cosine",
  scale_features = TRUE
)

table(res$clusters)
res$silhouette_score
})
```

`cluster_summary` *Cluster-wise summary table using a separate cluster vector (gtsummary + gt)*

Description

Produce a cluster-stratified summary table using **gtsummary**, where the cluster assignments are supplied as a separate vector. All additional arguments (`...`) are passed directly to `gtsummary::tbl_summary()`, so users can specify `all_continuous()` / `all_categorical()` selectors and custom statistics.

Usage

```
cluster_summary(
  data,
  clusters,
  add_options = list(add_overall = FALSE, add_n = TRUE, add_p = FALSE),
  return_as = c("gtsummary", "gt"),
  include = NULL,
  ...
)
```

Arguments

data	A data.frame or tibble of features to summarize.
clusters	A vector (factor, character, or numeric) of cluster labels with length equal to nrow(data).
add_options	List of post-processing options: • add_overall (default FALSE): add overall column • add_n (default TRUE) : add group Ns • add_p (default FALSE): add p-values
return_as	"gtsummary" (default) or "gt". When "gt", the function calls gtsummary::as_gt() for rendering.
include	Optional character vector of variables to include. Defaults to all columns in data.
...	Passed to gtsummary::tbl_summary() (e.g., statistic=, type=, digits=, missing=, label=, etc.).

Value

A gtsummary::tbl_summary (default) or gt::gt_tbl if return_as="gt".

Examples

```
if(requireNamespace("gtsummary")){
df <- data.frame(
  age = rnorm(100, 60, 10),
  bmi = rnorm(100, 28, 5),
  sex = sample(c("F","M"), 100, TRUE)
)
cl <- sample(1:3, 100, TRUE)

cluster_summary(
  data = df,
  clusters = cl,
  statistic = list(
    gtsummary::all_continuous()  ~ "{mean} ({sd})",
    gtsummary::all_categorical() ~ "{n} / {N} ({p}%)"
  ),
  missing = "always"
```

```
)
}
```

`clusters` *Cluster assignments based on missingness patterns*

Description

A tibble assigning each observation in `df_missing` to a cluster determined by its missingness pattern.

Usage

```
clusters
```

Format

A tibble with *8000* rows and 2 variables:

index Integer. Row identifier imported from `data_raw/clusters.csv`.

cluster Factor (or integer) giving the missingness-based cluster for each row.

Source

Imported from `data_raw/clusters.csv`, then renamed `...1` → `index`.

Examples

```
data(clusters)
table(clusters$cluster)
```

`create_cissvae_env` *Create or reuse a CISSVAE Python virtual environment*

Description

This function will either find an existing virtualenv by name (in the default location) or at a custom filesystem path, or create it (and install CISSVAE into it).

Usage

```
create_cissvae_env(
  envname = "cissvae_environment",
  path = NULL,
  install_python = FALSE,
  python_version = "3.10"
)
```

Arguments

envname	Name of the virtual environment (when using the default env location).
path	Character; optional path to the directory in which to create/use the virtualenv.
install_python	Logical; if TRUE, install Python if none of at least the requested version is found on the system.
python_version	Python version string (major.minor), used when installing Python.

Value

NULL. Called for side effects.

Examples

```
## Requires a working Python environment via reticulate
## Examples are wrapped in try() to avoid failures on CRAN check systems
try({
create_cissvae_env(
envname = "cissvae_environment",
install_python = FALSE,
python_version = "3.10")})
```

---

create_missingness_prop_matrix

Create Missingness Proportion Matrix

Description

Creates a matrix where each entry represents the proportion of missing values for each sample–feature combination across multiple timepoints. Each sample will have one proportion value per feature. Features may have repeated time points (columns named like `feature_1`, `feature_2`, ...). This matrix can be used with `cluster_on_missing_prop()` to group samples with similar missingness patterns.

Usage

```
create_missingness_prop_matrix(
  data,
  index_col = NULL,
  cols_ignore = NULL,
  na_values = c(NA, NaN, Inf, -Inf),
  repeat_feature_names = character(0),
  loose = FALSE
)
```

Arguments

data	Data frame or matrix containing the input data with potential missing values.
index_col	Character scalar. Name of an index column to exclude from analysis (optional). If supplied and present, it will be removed from analysis; row names are preserved as-is.
cols_ignore	Character vector of column names to exclude from the proportion matrix (optional).
na_values	Vector of values to treat as missing in addition to standard missing values. Defaults to c(NA, NaN, Inf, -Inf).
repeat_feature_names	Character vector of "base" feature names that have repeated timepoints. Repeat measurements must be in the form <feature>_<timepoint> where <feature> is alphanumeric (and may include dots) and <timepoint> is an integer (e.g., "CRP_1").
loose	Logical. If True, will match any column starting with feature from repeat_feature_names

Value

A numeric matrix of dimension nrow(data) by n_features, where rows are samples and columns are features (base names). Entries are per-sample missingness proportions in [0, 1]. The returned matrix has an attribute "feature_columns_map": a named list mapping each output feature to the source columns used to compute its proportion.

Examples

```
df <- data.frame(
  id = paste0("s", 1:4),
  CRP_1 = c(1.2, NA, 2.1, NaN),
  CRP_2 = c(NA, NA, 2.0, 1.9),
  IL6_1 = c(0.5, 0.7, Inf, 0.4),
  IL6_2 = c(0.6, -Inf, 0.8, 0.5),
  Albumin = c(3.9, 4.1, 4.0, NA)
)

m <- create_missingness_prop_matrix(
  data = df,
  index_col = "id",
  cols_ignore = NULL,
  repeat_feature_names = c("CRP", "IL6")
)

dim(m)         # 4 x 3 (CRP, IL6, Albumin)
# per-sample proportion missing across CRP_1 and CRP_2
m[ , "CRP"]
attr(m, "feature_columns_map")
```

df_missing *Sample dataset with missing values*

Description

A tibble of simulated biomarker measurements with missing entries. Each row corresponds to one observation (indexed by `index`), and the remaining columns are the measured biomarker values, some of which are set to NA to demonstrate imputation workflows.

Usage

```
df_missing
```

Format

A tibble with *8,000* rows and *30* variables:

index Integer. Row identifier imported from `data_raw/df_missing.csv`.

Age, Salary, ZipCode10001-ZipCode30003 Demographic columns. Omit from selection of validation set. No missingness

Y11, ..., Y55 Simulated Biomarker columns, have missingness

Source

Imported from `data_raw/df_missing.csv`, then renamed `...1` → `index`.

Examples

```
data(df_missing)
str(df_missing)
summary(df_missing)
```

dni *Example dni matrix for demo of imputable_matrix*

Description

A sample imputable_matrix (dataframe).

Usage

```
dni
```

Format

A dataframe:

imputable_matrix A mock imputable_matrix dataframe

Source

Imported from `data_raw/dni.csv`

Examples

```
data(dni)
```

`impute_result` *CISS-VAE imputation result object*

Description

An S3 class returned by the CISS-VAE imputation function. Inherits from `list`, so all standard list operations (e.g., `$`, `[[`) work as expected. The class exists primarily to enable type-safe save/load methods that correctly handle the embedded PyTorch model.

Fields

The following fields may be present depending on which `return_*` arguments were set when the model was run:

`imputed_dataset` Data frame. The full imputed dataset.

`raw_data` Data frame. The original data before imputation.

`model` Python object (`CISSVAE`). The trained PyTorch model. Present if `return_model = TRUE`.

`cluster_dataset` Python object (`ClusterDataset`). Present if `return_dataset = TRUE`.

`clusters` Integer vector. Cluster assignment per row. Present if `return_clusters = TRUE`.

`silhouette_width` Numeric vector. Per-sample silhouette widths. Present if `return_silhouettes = TRUE`.

`training_history` Data frame. Loss values recorded during training. Present if `return_history = TRUE`.

`val_data` Data frame. Validation data with observed entries replaced by `NA`. Present if `return_validation_dataset = TRUE`.

`val_imputed` Data frame. Model reconstructions of the validation entries. Present if `return_validation_dataset = TRUE`.

See Also

`save_impute_result`, `load_impute_result`, `print.impute_result`, `performance_by_cluster`

`impute_with_cissvae` *Impute new data with a loaded Python CISS-VAE model*

Description

Given a loaded model, an R data frame, and a vector of cluster labels, this builds the Python ClusterDataset and DataLoader, runs inference, and returns an imputed data frame in R.

Usage

```
impute_with_cissvae(
  model,
  data,
  index_col = NULL,
  cols_ignore = NULL,
  clusters,
  imputable_matrix = NULL,
  binary_feature_mask = NULL,
  categorical_column_map = NULL,
  replacement_value = 0,
  batch_size = NULL,
  seed = 42
)
```

Arguments

`model`	Python model object loaded via load_cissvae_model()
`data`	R data.frame with missing values
`index_col`	String name of index column to preserve (optional)
`cols_ignore`	Character vector of column names to exclude from imputation scoring.
`clusters`	Integer vector of cluster labels for rows of data
`imputable_matrix`	Logical matrix indicating entries allowed to be imputed.
`binary_feature_mask`	Logical vector marking which columns are binary.
`categorical_column_map`	Named list. Maps generated dummy variable column names to the original categorical column name.
`replacement_value`	Numeric value used to replace missing entries before model input.
`batch_size`	Batch size passed to Python DataLoader. If NULL, batch_size = nrow(data)
`seed`	Base random seed for reproducible results

Value

Imputed R data.frame

Tips

- Use same ClusterDataset parameters as for initial model training.
- Clusters must have same labels as clusters used for model training
- 'binary_feature_mask' is required for correct imputation of binary columns.

Examples

```
## Requires a working Python environment via reticulate
## Wrapped in try() and donttest to avoid CRAN check failures

try({
  # Activate your reticulate Python environment with ciss_vae installed
  reticulate::use_virtualenv("cissvae_environment", required = TRUE)

  # Load example data and clusters (replace with your own)
  data(df_missing)
  data(clusters)

  # Load a previously saved model
  model <- try(load_cissvae_model("model.pt", python_env = "cissvae_environment"))

  # Perform imputation on new data
  imputed_df <- try(
    impute_with_cissvae(
      model = model,
      data = df_missing,
      index_col = "index",
      cols_ignore = c("Age", "Salary"),
      clusters = clusters$clusters,
      imputable_matrix = NULL,
      binary_feature_mask = NULL,
      replacement_value = 0,
      batch_size = 4000L,
      seed = 42
    )
  )
})
```

`load_cissvae_model` *Load a saved CISS-VAE model*

Description

Loads a CISSVAE model previously saved by save_cissvae_model. When the model was saved with method = "state_dict" (the default), the architecture is automatically reconstructed from the paired .config.rds file and the weights are loaded via load_state_dict. When the model was saved with method = "full", the entire Python object is deserialised directly.

Usage

```
load_cissvae_model(
  file,
  method = c("state_dict", "full"),
  device = "cpu",
  python_env = NULL
)
```

Arguments

file	Character string. Path to the saved model file (.pt). For method = "state_dict", a paired <file>.config.rds must exist in the same location (written automatically by save_cissvae_model).
method	Character string. One of "state_dict" (default) or "full". Must match the method used when saving.
device	Character string. PyTorch device string passed to torch.load as map_location. Defaults to "cpu". Use "cuda" to load directly onto GPU.
python_env	Optional character string. Name of a virtualenv or conda environment to activate before loading. If NULL (default), the currently active Python session is used.

Details

The CISSVAE class is imported from the installed ciss_vae.classes using reticulate::import reticulate::import).

Value

A CISSVAE Python object in evaluation mode (model.eval() has been called).

See Also

save_cissvae_model, load_impute_result

Examples

```
try({
  reticulate::use_virtualenv("cissvae_environment", required = TRUE)

  pt_file <- tempfile(fileext = ".pt")

  # save first
  save_cissvae_model(dat$model, file = pt_file)
```

```
  # reload in same or new session
  model <- load_cissvae_model(file = pt_file, device = "cpu")
})
```

---

load_impute_result *Load an* impute_result *object from disk*

Description

Reads an impute_result previously saved by save_impute_result. R-native fields are restored from the RDS file. If a CISSVAE model was saved, the architecture is rebuilt from the stored configuration, the state_dict weights are loaded, and the model is placed in evaluation mode (model.eval()).

The CISSVAE class is imported from the installed ciss_vae.classes Python package using reticulate::import.

Usage

```
load_impute_result(dir, name = "result", device = "cpu")
```

Arguments

dir	Character string. Path to the directory written by save_impute_result.
name	Character string. Base name used when saving.
device	Character string. PyTorch device ("cpu" or "cuda").

Value

An impute_result object with model restored if present.

---

mock_surv *Example survival data for demo of imputable_matrix*

Description

A sample survival dataset

Usage

```
mock_surv
```

Format

A dataframe:

mock_surv A mock survival dataset

Source

Imported from `data_raw/mock_survival.csv`

Examples

```
data(mock_surv)
```

`performance_by_cluster` *Compute per-cluster and per-group performance metrics (MSE, BCE)*

Description

Calculates mean squared error (MSE) for continuous features and binary cross-entropy (BCE) for features explicitly marked as binary, comparing model-imputed validation values against ground-truth validation data.

Usage

```
performance_by_cluster(
  res,
  clusters = NULL,
  group_col = NULL,
  feature_cols = NULL,
  binary_features = character(0),
  categorical_features = list(),
  by_group = TRUE,
  by_cluster = TRUE,
  cols_ignore = NULL,
  eps = 1e-07
)
```

Arguments

`res`	A list containing CISS-VAE run outputs. Must include: • `res$val_data`: validation data frame (with `NA` for non-validation cells) • `res$val_imputed`: model-imputed validation predictions • `res$clusters`: cluster labels for each row
`clusters`	Optional vector (same length as rows in `val_data`) of cluster labels. If `NULL`, `res$clusters` will be used.
`group_col`	Optional character string naming a column in `val_data` used for grouping (e.g., sex, treatment group, etc.). If supplied, summaries can be computed per group and group-by-cluster.
`feature_cols`	Character vector specifying which feature columns to evaluate. Defaults to all numeric columns except `group_col` and those in `cols_ignore`.

`binary_features`
: Character vector naming those columns (subset of `feature_cols`) that should use BCE instead of MSE.

`categorical_features`
: List of categorical features. Key should be the category name and values should be the name of the dummy/one-hot-encoded variables.

`by_group`
: Logical; if `TRUE`, compute summaries by `group_col`. Ignored if `group_col` is `NULL`.

`by_cluster`
: Logical; if `TRUE`, compute summaries by cluster.

`cols_ignore`
: Character vector of column names to exclude from scoring (e.g., IDs).

`eps`
: Numeric. Small constant used for clipping probabilities in BCE calculation. Default is `1e-7`.

Details

Validation loss is computed at the cell level and then aggregated to produce overall, per-cluster, per-group, and group-by-cluster summaries.

For features listed in `binary_features`, performance is binary cross-entropy (BCE):

$$-[y \log(p) + (1 - y) \log(1 - p)]$$

where p is the predicted probability.

For other numeric features, performance is mean squared error (MSE):

$$(y - \hat{y})^2$$

.

Losses are computed at the individual cell level using only validation entries (non-NA in `val_data`), then aggregated.

Value

A named list containing:

- `overall`: overall validation metrics (MSE, BCE, total)
- `per_cluster`: metrics summarized by cluster (if `by_cluster = TRUE`)
- `per_group`: metrics summarized by group (if `by_group = TRUE`)
- `group_by_cluster`: metrics summarized by group and cluster (if both `by_group` and `by_cluster` are `TRUE`)

Each summary contains:

- `mse`: mean squared error across continuous validation cells
- `bce`: mean binary cross-entropy across binary validation cells
- `imputation_error`: `mse + bce`

Examples

```
data_complete <- data.frame(
  id = 1:10,
  group = sample(c("A", "B"), 10, replace = TRUE),
  x1 = rnorm(10),
  x2 = rnorm(10)
)

missing_mask <- matrix(
  sample(c(TRUE, FALSE), 20, replace = TRUE),
  nrow = 10
)

val_data <- data_complete
val_data[which(missing_mask, arr.ind = TRUE)] <- NA

val_imputed <- data.frame(
  id = data_complete$id,
  group = data_complete$group,
  x1 = mean(data_complete$x1),
  x2 = mean(data_complete$x2)
)

val_imputed[which(missing_mask, arr.ind = TRUE)] <- NA

result <- list(
  val_data = val_data,
  val_imputed = val_imputed,
  clusters = sample(c(0, 1), 10, replace = TRUE)
)

performance_by_cluster(
  res = result,
  group_col = "group",
  binary_features = character(0),
  by_group = TRUE,
  by_cluster = TRUE,
  cols_ignore = "id"
)
```

---

plot_vae_architecture *Plot VAE Architecture Diagram*

Description

Creates a horizontal schematic diagram of the CISS-VAE architecture, showing shared and cluster-specific layers. This function wraps the Python `plot_vae_architecture` function from the ciss_vae package.

Usage

```
plot_vae_architecture(
  model,
  title = NULL,
  color_shared = "skyblue",
  color_unshared = "lightcoral",
  color_latent = "gold",
  color_input = "lightgreen",
  color_output = "lightgreen",
  figsize = c(16, 8),
  save_path = NULL,
  dpi = 300,
  return_plot = FALSE,
  display_plot = TRUE
)
```

Arguments

model	A trained CISSVAE model object (Python object)
title	Title of the plot. If NULL, no title is displayed. Default NULL.
color_shared	Color for shared hidden layers. Default "skyblue".
color_unshared	Color for unshared (cluster-specific) hidden layers. Default "lightcoral".
color_latent	Color for latent layer. Default "gold".
color_input	Color for input layer. Default "lightgreen".
color_output	Color for output layer. Default "lightgreen".
figsize	Size of the matplotlib figure as c(width, height). Default c(16, 8).
save_path	Optional path to save the plot as PNG. If NULL, plot is displayed. Default NULL.
dpi	Resolution for saved PNG file. Default 300.
return_plot	Logical; if TRUE, returns the plot as an R object using reticulate. Default FALSE.
display_plot	Logical; if TRUE, displays the plot. Set to FALSE when only saving. Default TRUE.

Value

If return_plot is TRUE, returns a Python matplotlib figure object that can be further manipulated. Otherwise returns NULL invisibly.

Tips

- If you get a TCL or TK error, run: `reticulate::py_run_string("import matplotlib; matplotlib.use('Agg')")` to change the matplotlib backend to use 'Agg' instead.

Examples

```
## Requires a working Python environment via reticulate
## Examples are wrapped in try() to avoid failures on CRAN check systems

try({
  # Train a model first
  result <- run_cissvae(my_data, return_model = TRUE)

  # Basic plot
  plot_vae_architecture(result$model)

  # Save plot to file
  plot_vae_architecture(
    model = result$model,
    title = "CISS-VAE Architecture",
    save_path = "vae_architecture.png",
    dpi = 300
  )

  # Return plot object for further manipulation
  fig <- plot_vae_architecture(
    model = result$model,
    return_plot = TRUE,
    display_plot = FALSE
  )
})
```

---

print.impute_result *Print an* impute_result *object*

Description

Prints a structured summary of an `impute_result` object, including the dimensions of the imputed dataset, a preview of the imputed data, cluster composition, and which optional components are present.

Usage

```
## S3 method for class 'impute_result'
print(x, n = 6, ...)
```

Arguments

x	An object of class `impute_result`.
n	Integer. Number of rows of the imputed dataset to preview. Defaults to `6`.
...	Further arguments passed to or from other methods (currently unused).

Value

x, invisibly.

Examples

```
try({
  res <- ciss_vae(data = my_data, ...)
  print(res)      # default: 6 row preview
  print(res, n = 10)
})
```

---

run_cissvae *Run the CISS-VAE pipeline for missing data imputation*

Description

This function wraps the Python run_cissvae function from the ciss_vae package, providing a complete pipeline for missing data imputation using a Cluster-Informed Shared and Specific Variational Autoencoder (CISS-VAE). The function handles data preprocessing, model training, and returns imputed data along with optional model artifacts.

The CISS-VAE architecture uses cluster information to learn both shared and cluster-specific representations, enabling more accurate imputation by leveraging patterns within and across different data subgroups.

Usage

```
run_cissvae(
  data,
  index_col = NULL,
  val_proportion = 0.1,
  replacement_value = 0,
  cols_ignore = NULL,
  imputable_matrix = NULL,
  binary_feature_mask = NULL,
  categorical_column_map = NULL,
  print_dataset = TRUE,
  clusters = NULL,
  n_clusters = NULL,
  seed = 42,
  missingness_proportion_matrix = NULL,
  scale_features = FALSE,
  k_neighbors = 15L,
  leiden_resolution = 0.5,
  leiden_objective = "CPM",
  hidden_dims = c(150, 120, 60),
  latent_dim = 15,
```

```
  layer_order_enc = c("unshared", "unshared", "unshared"),
  layer_order_dec = c("shared", "shared", "shared"),
  latent_shared = FALSE,
  output_shared = FALSE,
  batch_size = 4000,
  epochs = 500,
  initial_lr = 0.01,
  decay_factor = 0.999,
  weight_decay = 0.001,
  beta = 0.001,
  device = NULL,
  max_loops = 100,
  patience = 2,
  epochs_per_loop = NULL,
  initial_lr_refit = NULL,
  decay_factor_refit = NULL,
  beta_refit = NULL,
  verbose = FALSE,
  return_model = TRUE,
  return_clusters = TRUE,
  return_silhouettes = FALSE,
  return_history = FALSE,
  return_dataset = FALSE,
  return_validation_dataset = TRUE,
  debug = FALSE,
  columns_ignore = NULL
)
```

Arguments

`data`	A data.frame or matrix (samples × features) containing the data to impute. May contain `NA` values which will be imputed.
`index_col`	Character. Name of column in `data` to treat as sample identifier. This column will be removed before training and re-attached to results. Default `NULL`.
`val_proportion`	Numeric. Fraction of non-missing entries to hold out for validation during training. Must be between 0 and 1. Default `0.1`.
`replacement_value`	Numeric. Fill value for masked entries during training. Default `0.0`.
`cols_ignore`	Character or integer vector. Columns to exclude from validation set. Can specify by name or index. Default `NULL`.
`imputable_matrix`	Logical matrix indicating entries allowed to be imputed.
`binary_feature_mask`	Logical vector marking which columns are binary.
`categorical_column_map`	Named list. Maps generated dummy variable column names to the original categorical column name.

print_dataset Logical. If TRUE, prints dataset summary information during processing. Default TRUE.

clusters Optional vector or single-column data.frame of precomputed cluster labels for samples. If NULL, clustering will be performed automatically. Default NULL.

n_clusters Integer. Number of clusters for KMeans clustering when clusters is NULL. Number of clusters for KMeans clustering when 'clusters' is NULL. If NULL, will use Leiden for clustering. Default NULL.

seed Integer. Random seed for reproducible results. Default 42.

missingness_proportion_matrix
Optional pre-computed missingness proportion matrix for biomarker-based clustering. If provided, clustering will be based on these proportions. Default NULL.

scale_features Logical. Whether to scale features when using missingness proportion matrix clustering. Default FALSE.

k_neighbors Integer. Number of nearest neighbors for Leiden clustering. Defaults to 15.

leiden_resolution
Float. Resolution parameter for Leiden clustering. Defaults to 0.5.

leiden_objective
Character. Objective function for Leiden clustering. One of ("CPM", "RB", "Modularity")

hidden_dims Integer vector. Sizes of hidden layers in encoder/decoder. Length determines number of hidden layers. Default c(150, 120, 60).

latent_dim Integer. Dimension of latent space representation. Default 15.

layer_order_enc
Character vector. Sharing pattern for encoder layers. Each element should be "shared" or "unshared". Length must match length(hidden_dims). Default c("unshared", "unshared", "unshared").

layer_order_dec
Character vector. Sharing pattern for decoder layers. Each element should be "shared" or "unshared". Length must match length(hidden_dims). Default c("shared", "shared", "shared").

latent_shared Logical. Whether latent space weights are shared across clusters. Default FALSE.

output_shared Logical. Whether output layer weights are shared across clusters. Default FALSE.

batch_size Integer. Mini-batch size for training. Larger values may improve training stability but require more memory. Default 4000.

epochs Integer. Number of epochs for initial training phase. Default 500.

initial_lr Numeric. Initial learning rate for optimizer. Default 0.01.

decay_factor Numeric. Exponential decay factor for learning rate scheduling. Must be between 0 and 1. Default 0.999.

weight_decay Weight decay (L2 penalty) used in Adam optimizer.

beta Numeric. Weight for KL divergence term in VAE loss function. Controls regularization strength. Default 0.001.

device	Character. Device specification for computation ("cpu" or "cuda"). If NULL, automatically selects best available device. Default NULL.
max_loops	Integer. Maximum number of impute-refit loops to perform. Default 100.
patience	Integer. Early stopping patience for refit loops. Training stops if validation loss doesn't improve for this many consecutive loops. Default 2.
epochs_per_loop	Integer. Number of epochs per refit loop. If NULL, uses same value as epochs. Default NULL.
initial_lr_refit	Numeric. Learning rate for refit loops. If NULL, uses same value as initial_lr. Default NULL.
decay_factor_refit	Numeric. Decay factor for refit loops. If NULL, uses same value as decay_factor. Default NULL.
beta_refit	Numeric. KL weight for refit loops. If NULL, uses same value as beta. Default NULL.
verbose	Logical. If TRUE, prints detailed progress information during training. Default FALSE.
return_model	Logical. If TRUE, returns the trained Python VAE model object. Default TRUE.
return_clusters	Logical. If TRUE returns cluster vector
return_silhouettes	Logical. If TRUE, returns silhouette scores for cluster quality assessment. Default FALSE.
return_history	Logical. If TRUE, returns training history as a data.frame containing loss values and metrics over epochs. Default FALSE.
return_dataset	Logical. If TRUE, returns the ClusterDataset object used during training (contains validation data, masks, etc.). Default FALSE.
return_validation_dataset	Logical. If TRUE returns validation dataset
debug	Logical; if TRUE, additional metadata is returned for debugging.
columns_ignore	Alias of cols_ignore. Kept for continuity

Details

The CISS-VAE method works in two main phases:

1. **Initial Training**: The model is trained on the original data with validation holdout to learn initial representations and imputation patterns.
2. **Impute-Refit Loops**: The model iteratively imputes missing values and retrains on the updated dataset until convergence or maximum loops reached.

The architecture uses both shared and cluster-specific layers to capture:

- **Shared patterns**: Common relationships across all clusters
- **Specific patterns**: Unique relationships within each cluster

Value

A list containing imputed data and optional additional outputs:

imputed_dataset data.frame of imputed data with same dimensions as input. Missing values are filled with model predictions. If `index_col` was provided, it is re-attached as the first column.

model (if `return_model=TRUE`) Python CISSVAE model object. Can be used for further analysis or predictions.

cluster_dataset (if `return_dataset=TRUE`) Python ClusterDataset object containing validation data, masks, normalization parameters, and cluster labels. Can be used with performance_by_cluster() and other analysis functions.

clusters (if `return_clusters=TRUE`) Returns vector of cluster assignments

silhouettes (if `return_silhouettes=TRUE`) Numeric silhouette score measuring cluster separation quality.

training_history (if `return_history=TRUE`) data.frame containing training history with columns for epoch, losses, and validation metrics.

val_data (if `return_validation_dataset=TRUE`) data.frame containing values held aside for validation.

val_imputed (if `return_validation_dataset=TRUE`) data.frame containing imputed values of set held aside for validation.

Requirements

This function requires the Python `ciss_vae` package to be installed and accessible via `reticulate`.

Performance tips

- If Leiden clustering yields too many clusters, consider increasing `k_neighbors` or reducing `leiden_resolution`.
- Use GPU computation when available for faster training on large datasets. Use `check_devices()` to see what devices are available.
- Adjust `batch_size` based on available memory (larger is faster but uses more memory).
- Set `verbose = TRUE` to monitor training progress.

See Also

`create_missingness_prop_matrix` for creating missingness proportion matrices `performance_by_cluster` for analyzing model performance using the returned dataset

Examples

```
## Requires a working Python environment via reticulate
## Examples are wrapped in try() to avoid failures on CRAN check systems
library(rCISSVAE)

data(df_missing)
data(clusters)
```

```
try({
dat = run_cissvae(
 data = df_missing,
 index_col = "index",
 val_proportion = 0.1, ## pass a vector for different proportions by cluster
 cols_ignore = c("Age", "Salary", "ZipCode10001", "ZipCode20002", "ZipCode30003"),
 clusters = clusters$clusters, ## we have precomputed cluster labels so we pass them here
 epochs = 5,
 return_silhouettes = FALSE,
 return_history = TRUE,  # Get detailed training history
 verbose = FALSE,
 return_model = TRUE, ## Allows for plotting model schematic
 device = "cpu",  # Explicit device selection
 layer_order_enc = c("unshared", "shared", "unshared"),
 layer_order_dec = c("shared", "unshared", "shared")
)
})
```

`save_cissvae_model` *Save a trained CISS-VAE model to disk*

Description

Saves a trained `CISSVAE` PyTorch model to disk. The default method (`method = "state_dict"`) saves only the model weights alongside an automatically captured architecture config, which is the recommended approach for portability and long-term reproducibility. A full-object save (`method = "full"`) is also available but is less portable across Python / PyTorch versions.

Usage

```
save_cissvae_model(
  model,
  file,
  method = c("state_dict", "full"),
  overwrite = FALSE
)
```

Arguments

`model`	A `CISSVAE` Python object (reticulate proxy), e.g., `res$model` from `run_cissvae()`.
`file`	Character string. File path for the saved model. For `method = "state_dict"` two files are written: • `<file>` — the PyTorch `state_dict` checkpoint (`.pt`). • `<file>.config.rds` — the architecture config needed to reconstruct the model on load. For `method = "full"` only `<file>` is written.

method	Character string. One of "state_dict" (default) or "full".
	"state_dict" Saves weights and architecture config separately. Recommended. Requires load_cissvae_model() to reload.
	"full" Saves the entire Python object with torch.save. Less portable; may break across PyTorch versions.
overwrite	Logical. If FALSE (default), an error is raised when output files already exist.

Value

NULL, invisibly. Called for its side effects.

See Also

load_cissvae_model, save_impute_result

Examples

```
try({
  reticulate::use_virtualenv("cissvae_environment", required = TRUE)
  data(df_missing)
  data(clusters)

  dat <- run_cissvae(
    data             = df_missing,
    index_col        = "index",
    val_proportion   = 0.1,
    clusters         = clusters$clusters,
    epochs           = 5,
    return_model     = TRUE,
    device           = "cpu",
    layer_order_enc  = c("unshared", "shared", "unshared"),
    layer_order_dec  = c("shared",   "unshared", "shared")
  )

  # default: saves state_dict + config
  save_cissvae_model(dat$model, file = tempfile(fileext = ".pt"))

  # alternative: full object save
  save_cissvae_model(dat$model, file = tempfile(fileext = ".pt"),
                     method = "full")
})
```

---

save_impute_result *Save an* impute_result *object to disk*

Description

Saves an impute_result to a directory. R-native fields are written to an RDS file. The CISSVAE PyTorch model, if present, is saved separately as a state_dict checkpoint (.pt) so that the live Python object is never passed to saveRDS (which cannot serialise it). The model architecture is captured automatically from the live object and stored inside the RDS so that load_impute_result can reconstruct it without any additional input from the user.

Usage

```
save_impute_result(x, dir, name = "result", overwrite = FALSE)
```

Arguments

x	An object of class impute_result.
dir	Character string. Path to the directory in which to write files. Created recursively if it does not exist.
name	Character string. Base name used for output files. Defaults to "result". Two files are written: • <name>.rds — R data (all fields except the live model). • <name>_state_dict.pt — PyTorch model weights.
overwrite	Logical. If FALSE (default), an error is raised when output files already exist. Set to TRUE to overwrite silently.

Value

Invisibly, a named list with elements meta (path to the RDS) and model (path to the .pt file, or NULL if no model was present).

See Also

load_impute_result, impute_result

Examples

```
## Not run:
res <- ciss_vae(data = my_data, return_model = TRUE, ...)
save_impute_result(res, dir = "run_001", name = "imputation_result")

## End(Not run)
```

update_cissvae_env *Update (pip install –upgrade) CISSVAE inside the CISSVAE virtualenv*

Description

Uses the same env discovery rules as `create_cissvae_env()`.

Usage

```
update_cissvae_env(
  envname = "cissvae_environment",
  path = NULL,
  install_python = FALSE,
  python_version = "3.10",
  pkgs = "ciss-vae",
  upgrade = TRUE,
  quiet = FALSE
)
```

Arguments

envname	Name of the virtual environment (default location).
path	Optional directory containing the virtualenv.
install_python	Logical; if TRUE, install Python if none of at least the requested version is found on the system.
python_version	Python version string (major.minor), used when installing Python.
pkgs	Character vector of pip packages to upgrade in the env. Defaults to ciss-vae only. Other dependencies of ciss-vae ("numpy", "pandas", "torch", "rich", "matplotlib", "scikit-learn", "optuna", "typing") can also be upgraded using this function by adding them to the character vector.
upgrade	Logical; if TRUE pass –upgrade to pip.
quiet	Logical; reduce output.

Value

NULL, invisibly.

Examples

```
## Requires a working Python environment via reticulate
## Examples are wrapped in try() to avoid failures on CRAN check systems
try({
update_cissvae_env(
envname = "cissvae_environment",
install_python = FALSE,
python_version = "3.10",
```

```
    pkgs            = "ciss-vae")})
```

Index

∗ **datasets**
clusters, 13
df_missing, 16
dni, 16
mock_surv, 21

autotune_cissvae, 2

check_devices, 7
cluster_heatmap, 7
cluster_on_missing, 9
cluster_on_missing_prop, 10
cluster_summary, 11
clusters, 13
create_cissvae_env, 13
create_cissvae_env(), *35*
create_missingness_prop_matrix, 14, *31*

df_missing, 16
dni, 16

gtsummary::as_gt(), *12*
gtsummary::tbl_summary(), *11, 12*

impute_result, 17, *34*
impute_with_cissvae, 18

load_cissvae_model, 19, *33*
load_impute_result, *17*, *20*, 21, *34*

mock_surv, 21

performance_by_cluster, *17*, 22, *31*
plot_vae_architecture, 24
print.impute_result, *17*, 26

run_cissvae, 27

save_cissvae_model, *20*, 32
save_impute_result, *17*, *21*, 33, *33*

update_cissvae_env, 35