Package 'AutoCopula'

January 3, 2025

Type Package

Title Automated Copula Utilization

Version 1.0.0

Author Adrian Antico [aut, cre, cph]

Maintainer Adrian Antico <adrianantico@gmail.com>

Description AutoCopula is a comprehensive package for copula modeling, evaluation, and visualization.

Imports R6,
data.table,
dplyr,
echarts4r,
copula,
VineCopula,
future,
future.apply

Suggests testthat,
here,
shiny,
DT,
bs4Dash,
readxl

License AGPL (>= 3) + file LICENSE

Encoding UTF-8

LazyData true

Depends R (>= 4.1.0)

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

URL https://github.com/AdrianAntico/AutoCopula

BugReports https://github.com/AdrianAntico/AutoCopula/issues

Language en-US

NeedsCompilation no

R topics documented:

EDA 2
hello 4
ModelEvaluation 4
ModelFitter 6
ModelScorer 7

Index **11**

EDA *EDA (Exploratory Data Analysis) Class for AutoCopula*

Description

Provides tools for exploratory data analysis tailored to copula modeling.

Public fields

`data` A `data.table` containing the dataset for analysis.

`plots` A list of `echarts4r` plots generated during the analysis.

`correlation_matrix` A `data.table` of correlations Initialize the EDA class

Methods

Public methods:

- `EDA$new()`
- `EDA$summarize()`
- `EDA$correlate()`
- `EDA$visualize_distributions()`
- `EDA$visualize_scatterplots()`
- `EDA$generate_3d_scatter_plot()`
- `EDA$clone()`

Method `new()`**:**

Usage:

```
EDA$new(data)
```

Arguments:

`data` A `data.table` containing the dataset for analysis.

Method `summarize()`**:** Calculates mean, median, sd, and the count of missing values for each column.

Usage:

```
EDA$summarize()
```

Returns: A `data.table` containing the summary statistics.

Method `correlate()`**:** Calculates Pearson, Spearman, and Kendall's Tau correlations between all numeric columns in the dataset.

Usage:

```
EDA$correlate(input_cols = NULL)
```

Arguments:

`input_cols` Names of numeric variables to correlate. If NULL then all numeric columns in the data.table will be utilized

Returns: A `data.table` with the pairwise Pearson, Spearman, and Kendall's Tau correlation values for all numeric columns.

Method `visualize_distributions()`**:** Generates histograms for numeric columns and optionally overlays density lines.

Usage:

```
EDA$visualize_distributions(
  input_cols = NULL,
  title_prefix = "Distribution of",
  bins = 20,
  add_density = TRUE,
  tooltip_trigger = "axis",
  theme = "westeros",
  density_opacity = 0.4
)
```

Arguments:

`input_cols` Names of numeric variables to plot

`title_prefix` Character. Prefix for the plot title.

`bins` Integer. Number of bins for the histogram. Defaults to Sturges' formula.

`add_density` Logical. Whether to add a density line. Defaults to `TRUE`.

`tooltip_trigger` "axis"

`theme` Character. Theme for the plot

`density_opacity` numeric. default 0.4

Returns: A list of `echarts4r` histogram plots.

Method `visualize_scatterplots()`**:** Generates scatterplots for the percentile ranks of all numeric column pairs.

Usage:

```
EDA$visualize_scatterplots(
  title_prefix = "Empirical Copula View of",
  theme = "westeros"
)
```

Arguments:

`title_prefix` Character. Prefix for the plot title.

`theme` Character. Theme for the plot. Defaults to "westeros".

Returns: A list of `echarts4r` scatter plots for the percentile ranks.

Method `generate_3d_scatter_plot()`**:** Generates a 3D scatter plot for three numeric variables.

Usage:

```
EDA$generate_3d_scatter_plot(
  col_x,
  col_y,
  col_z,
```

```
  rank_values = TRUE,
  theme = "westeros"
)
```

Arguments:

`col_x` The name of the first numeric column.

`col_y` The name of the second numeric column.

`col_z` The name of the third numeric column.

`rank_values` Logical. Whether to transform variables to their percentile ranks. Defaults to TRUE.

`theme` Name of theme for `echarts4r` plot

Returns: An `echarts4r` 3D scatter plot.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
EDA$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`hello` *Hello, World!*

Description

Prints 'Hello, world!'.

Usage

```
hello()
```

Examples

```
hello()
```

`ModelEvaluation` *ModelEvaluation*

Description

ModelEvaluation

ModelEvaluation

Details

An R6 class to evaluate copula models. Includes tools to generate tables of evaluation metrics and visualizations to assess model performance and dependence structure.

Public fields

`fit_results` A list of fitted copula model objects.

`evaluation_metrics` A data.table containing model evaluation metrics.

`plots` A list of visualizations for evaluating copula models.

`data` The original dataset used for fitting copulas. Initialize the ModelEvaluation class

Methods

Public methods:

- `ModelEvaluation$new()`
- `ModelEvaluation$generate_metrics()`
- `ModelEvaluation$generate_overlay_plot()`
- `ModelEvaluation$clone()`

Method `new()`**:**

Usage:

```
ModelEvaluation$new(fit_results, data)
```

Arguments:

`fit_results` A list of fitted copula model objects.

`data` The dataset used for fitting models, in uniform margins.

Returns: A new instance of the ModelEvaluation class.

Method `generate_metrics()`**:** Compute and summarize evaluation metrics for fitted copulas. Metrics include log-likelihood, AIC, BIC, and dependence measures.

Usage:

```
ModelEvaluation$generate_metrics()
```

Returns: A data.table of evaluation metrics.

Method `generate_overlay_plot()`**:** Generate overlay plot for observed vs simulated pseudo-observations.

Usage:

```
ModelEvaluation$generate_overlay_plot(model_name, theme = "westeros")
```

Arguments:

`model_name` The name of the model for which to generate the plot.

`theme` name of theme for `echarts4r`

Returns: An echarts4r plot object.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ModelEvaluation$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

ModelFitter *ModelFitter*

Description

ModelFitter

ModelFitter

Details

An R6 class for fitting and evaluating copula models. Provides a library of pre-defined copula models and tools for custom models.

Public fields

`data` A data.table containing the dataset for modeling.

`data_matrix` A matrix of post-transformed data for copula fitting

`copula_library` A pre-defined library of copula models.

`fit_results` A list to store results of copula fits. Initialize the ModelFitter class

Methods

Public methods:

- `ModelFitter$new()`
- `ModelFitter$fit_models()`
- `ModelFitter$list_models()`
- `ModelFitter$clone()`

Method `new()`**:**

Usage:

```
ModelFitter$new(data)
```

Arguments:

`data` A data.table containing the dataset for modeling.

Returns: A new instance of the ModelFitter class.

Method `fit_models()`**:** Fits multiple copula models in a loop.

Usage:

```
ModelFitter$fit_models(model_names)
```

Arguments:

`model_names` A character vector of model names to fit.

Returns: A data.table summarizing the results for each model.

Method `list_models()`**:** Lists available copula models in the library.

Usage:

```
ModelFitter$list_models()
```

Returns: A data.table summarizing the copula models.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ModelFitter$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`ModelScorer` *ModelScorer*

Description

ModelScorer

ModelScorer

Details

An R6 class to score copula models on new data, perform predictions, generate simulations, and visualize the results.

This method generates large-scale simulations by splitting the total number of samples into smaller batches. If `parallel` is set to TRUE, it uses the `future.apply` package to process batches in parallel. On sequential processing, batches are processed one at a time. The function ensures that all parallel sessions are closed upon completion or in the case of an error.

The `conditional_range_prediction()` function generates conditional samples from a fitted copula model for all combinations of specified ranges for multiple variables. This batch version is particularly useful for scenarios where multiple variables are partially observed or constrained within specific ranges, and predictions or simulations are required for the remaining variables across all combinations.

- **Input Parameters**:
  - `model_name`: The name of the fitted copula model from which predictions will be generated.
  - `known_ranges`: A named list where:
    - Each key represents a variable name.
    - The value is either:
      - **Single Value**: Represents an exact observation for that variable (e.g., `X = 0.5`).
      - **Range**: A numeric vector (e.g., `X = c(0.4, 0.6)`) representing the range within which the variable is constrained.
  - `n`: The number of conditional samples to generate for each combination.
  - `parallel`: Logical. If `TRUE`, computations are distributed across multiple threads for efficiency.
  - `threads`: Integer. Specifies the number of threads to use if `parallel = TRUE`. Defaults to one less than the number of available cores if not specified.
- **Process**:
  1. Validate that `model_name` corresponds to a fitted copula model and that all `known_ranges` variables match the dataset used for model fitting.
  2. Generate all combinations of values based on `known_ranges`:
     - For single values, treat them as exact observations.

    - For ranges, sample uniformly within the range for each combination.
  3. For each combination of known values:
    - Dynamically select the appropriate private conditional sampling method for the copula family.
    - Compute the conditional distribution of the remaining variables given the known values.
  4. Combine the results into a unified `data.table` containing all conditional samples.
- **Output**:
  - A `data.table` containing:
    * The conditional samples for the remaining variables.
    * The fixed or sampled values for the known variables for each combination.
    * A `batch_id` column that uniquely identifies each combination of known variable values.
- **Error Handling**:
  - If `model_name` is not a supported copula type or `known_ranges` contains invalid inputs, the function returns `NULL` with an appropriate warning or error message.
- **Applications**:
  - Suitable for use cases in finance, insurance, and other domains requiring predictions under multiple partially observed or constrained conditions. The batch version allows for efficient exploration of the conditional distribution over a wide range of possible scenarios.

The `hybrid_range_simulation()` function extends `hybrid_simulation()` by enabling simulations over a range of values for one known variable. It generates both conditional and unconditional samples for each value in the specified range and combines the results.

- **Input Parameters**:
  - `model_name`: The name of the fitted copula model to use for simulations.
  - `known_ranges`: List containing name as key and variable values as the values
  - `n`: Number of instances to simulate for each known value.
  - `parallel`: Logical indicating whether to run the simulation in parallel. Default is `FALSE`.
  - `threads`: Number of parallel threads to use if `parallel = TRUE`. Defaults to available cores minus one.
- **Output**:
  - A `data.table` containing the hybrid simulation results for all values in the specified range.

Public fields

`fit_results` A list of fitted copula model objects.

`data` Scoring data

`score_plots` A list of plots visualizing scored data.

Methods

Public methods:

- `ModelScorer$new()`
- `ModelScorer$single_instance_prediction()`

- ModelScorer$batch_prediction()
- ModelScorer$large_scale_simulation()
- ModelScorer$conditional_range_prediction()
- ModelScorer$hybrid_range_simulation()
- ModelScorer$clone()

Method `new()`**:**

Usage:

```
ModelScorer$new(fit_results, data)
```

Arguments:

`fit_results` A list of fitted copula model objects.

`data` data for scoring a single instance of copula models

Returns: A new instance of the ModelScorer class.

Method `single_instance_prediction()`**:** Generate a single instance prediction using the entire model.

Usage:

```
ModelScorer$single_instance_prediction(model_name)
```

Arguments:

`model_name` Name of the model to use.

Returns: A single joint prediction as a named list.

Method `batch_prediction()`**:** Generate batch predictions using the entire model.

Usage:

```
ModelScorer$batch_prediction(model_name, n = 100)
```

Arguments:

`model_name` Name of the model to use.

`n` Number of instances to generate.

Returns: A data.table of batch predictions.

Method `large_scale_simulation()`**:** Perform large-scale simulations using fitted copula models. The simulation can be processed sequentially or in parallel, with user-defined batch sizes and thread counts.

Usage:

```
ModelScorer$large_scale_simulation(
  model_name,
  batches = 10,
  batch_size = 1000,
  parallel = FALSE,
  threads = NULL
)
```

Arguments:

`model_name` A string specifying the name of the fitted model to use.

`batches` An integer specifying the number of batches to process. Default is 10.

`batch_size` An integer specifying the number of samples per batch. Default is 1000.

`parallel` A logical value indicating whether to use parallel processing. Default is FALSE.

threads An integer specifying the number of threads to use for parallel processing. If NULL, defaults to the number of available cores minus one.

Returns: A data.table containing all simulated values, with a BatchID column to indicate the batch each row belongs to.

Method conditional_range_prediction(): Generate conditional predictions for a range of known values.

Usage:

```
ModelScorer$conditional_range_prediction(
  model_name,
  known_ranges,
  n = 100,
  parallel = FALSE,
  threads = NULL
)
```

Arguments:

model_name Name of the model to use.

known_ranges List with variable name as key and variable values as the list values

n Number of samples to generate for each value in the range.

parallel Logical, whether to process the range values in parallel. Default is FALSE.

threads Integer, the number of threads to use if parallel = TRUE. Defaults to all available cores minus one.

Returns: A data.table containing conditional predictions for the specified range.

Method hybrid_range_simulation(): Generate hybrid simulations for a range of known values.

Usage:

```
ModelScorer$hybrid_range_simulation(
  model_name,
  known_ranges,
  n = 100,
  parallel = FALSE,
  threads = NULL
)
```

Arguments:

model_name Name of the model to use.

known_ranges List with variable name as key and variable values as the list values

n Number of instances to simulate for each known value.

parallel Logical, whether to use parallel processing.

threads Number of threads to use for parallel processing. Defaults to available cores minus one.

Returns: A data.table of hybrid simulations for the specified range of values.

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
ModelScorer$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Index

EDA, 2

hello, 4

ModelEvaluation, 4
ModelFitter, 6
ModelScorer, 7