Package 'causact'

July 22, 2025

Type Package

Title Fast, Easy, and Visual Bayesian Inference

Version 0.5.7

Description Accelerate Bayesian analytics workflows in 'R' through interactive modelling, visualization, and inference. Define probabilistic graphical models using directed acyclic graphs (DAGs) as a unifying language for business stakeholders, statisticians, and programmers. This package relies on interfacing with the 'numpyro' python package.

License MIT + file LICENSE

URL https://github.com/flyaflya/causact, https://www.causact.com/

BugReports https://github.com/flyaflya/causact/issues

SystemRequirements Python and numpyro are needed for Bayesian inference computations; python (>= 3.8) with header files and shared library; numpyro (= v0.12.1; https://https://num.pyro.ai/en/latest/index.html); arviz (= v0.15.1; https://https://python.arviz.org/en/stable/)

Encoding UTF-8

LazyData true

Depends R (>= 4.1.0)

Imports DiagrammeR (>= 1.0.9), dplyr (>= 1.0.8), magrittr (>= 1.5), ggplot2 (>= 3.4.0), rlang (>= 1.0.2), purrr (>= 1.0.0), tidyr (>= 1.1.4), igraph (>= 1.2.7), stringr (>= 1.4.1), cowplot (>= 1.1.0), forcats (>= 0.5.0), rstudioapi (>= 0.11), lifecycle (>= 1.0.2), reticulate (>= 1.30)

RoxygenNote 7.3.2

Suggests knitr, covr, testthat (>= 3.0.0), rmarkdown, extraDistr, mvtnorm

Config/testthat/edition 3

VignetteBuilder knitr

Config/reticulate list(packages = list(list(package = ``jax[cpu]'', pip = TRUE), list(package = ``numpyro[cpu]'', pip = TRUE), list(package = ``arviz'', pip = TRUE)))

NeedsCompilation no

Author Adam Fleischhacker [aut, cre, cph],
    Daniela Dapena [ctb],
    Rose Nguyen [ctb],
    Jared Sharpe [ctb]

Maintainer Adam Fleischhacker `<ajf@udel.edu>`

Repository CRAN

Date/Publication 2025-01-15 21:20:01 UTC

Contents

---

addPriorGroups	*Group together latent parameters by prior distribution.*

Description

[Experimental]

Add a column to a tidy dataframe of draws that groups parameters by their prior distribution. All parameters with the same prior distribution receive the same index.

Usage

```
addPriorGroups(drawsDF)
```

Arguments

drawsDF the dataframe created by dag_numpyro() where each row represents one draw of MCMC output. Two columns are expected, param - the parameter name, value - the realized value, and a third column, priorGroup, is appended as an integer grouping parameters by their prior distributions. The data for this third column is stored in an environment called cacheEnv when the dag_numpyro() function is called. Any parameters with the same prior end up in the same prior group; used by dagp_plot() to group parameters when plotted.

Value

a tidy dataframe of posterior draws. Useful for passing to dagp_plot() or for creating plots using ggplot().

---

baseballData	*Dataframe of 12,145 observations of baseball games in 2010 - 2014*

Description

Dataframe of 12,145 observations of baseball games in 2010 - 2014

Usage

```
baseballData
```

Format

A data frame with 12145 rows and 5 variables:

Date date game was played

Home abbreviation for home team (i.e. stadium where game played)

Visitor abbreviation for visiting team

HomeScore Runs scored by the home team

VisitorScore Runs scored by the visiting team

---

beachLocDF	*Dataframe where each row represents data about one of the 26 mile markers (fake) from mile 0 to mile 2.5 along the Ocean City, MD beach/boardwalk.*

Description

Dataframe where each row represents data about one of the 26 mile markers (fake) from mile 0 to mile 2.5 along the Ocean City, MD beach/boardwalk.

Usage

    beachLocDF

Format

A data frame with 26 rows and 3 variables:

mileMarker a number representing a location on the Ocean City beach/boardwalk.

beachgoerProb The probability of any Ocean City, MD beachgoer (during the hot swimming days) exiting the beach at that mile marker.

expenseEst The estimated annual expenses of running a business at that location on the beach. It is assumed a large portion of the expense is based on commercial rental rates at that location. More populated locations tend to have higher expenses.

carModelDF	*Dataframe of 1000 (fake) observations of whether certain car buyers were willing to get information on a credit card speciailizing in rewards for adventure travellers.*

Description

Dataframe of 1000 (fake) observations of whether certain car buyers were willing to get information on a credit card speciailizing in rewards for adventure travellers.

Usage

```
carModelDF
```

Format

A data frame with 1000 rows and 3 variables:

customerID a unique id of a potential credit card customer. They just bought a car and are asked if they want information on the credit card.

carModel The model of car purchased.

getCard Whether the customer expressed interest in hearing more about the card.

check_r_causact_env	*Check if 'r-causact' Conda environment exists*

Description

Check if 'r-causact' Conda environment exists

Usage

```
check_r_causact_env()
```

---

chimpanzeesDF   *Data from behavior trials in a captive group of chimpanzees, housed*
                                 in Lousiana. From Silk et al. 2005. Nature 437:1357-1359 and further
                                 popularized in McElreath, Richard. Statistical rethinking: A Bayesian
                                 course with examples in R and Stan. CRC press, 2020. Experiment

---

Description

Data from behavior trials in a captive group of chimpanzees, housed in Lousiana. From Silk et al.
2005. Nature 437:1357-1359 and further popularized in McElreath, Richard. Statistical rethinking:
A Bayesian course with examples in R and Stan. CRC press, 2020. Experiment

Usage

```
chimpanzeesDF
```

Format

A data frame with 504 rows and 9 variables:

actor name of actor

recipient name of recipient (NA for partner absent condition)

condition partner absent (0), partner present (1)

block block of trials (each actor x each recipient 1 time)

trial trial number (by chimp = ordinal sequence of trials for each chimp, ranges from 1-72; partner
  present trials were interspersed with partner absent trials)

prosoc_left prosocial_left : 1 if prosocial (1/1) option was on left

chose_prosoc choice chimp made (0 = 1/0 option, 1 = 1/1 option)

pulled_left which side did chimp pull (1 = left, 0 = right)

treatment narrative description combining condition and prosoc_left that describes the side the
  prosical food option was on and whether a partner was present

Source

Silk et al. 2005. Nature 437:1357-1359..

corruptDF	*Dataframe of 174 observations where information on the human developmet index (HDI) and the corruption perceptions index (CPI) both exist. Each observation is a country.*

Description

Dataframe of 174 observations where information on the human developmet index (HDI) and the corruption perceptions index (CPI) both exist. Each observation is a country.

Usage

```
corruptDF
```

Format

A data frame with 174 rows and 7 variables:

country country name

region region name as given with CPI rating

countryCode three letter abbreviation for country

regionCode four letter or less abbreviation for country

population 2017 country population

CPI2017 The Corruption Perceptions Index score for 2017: A country/territory's score indicates the perceived level of public sector corruption on a scale of 0-100, where 0 means that a country is perceived as highly corrupt and a 100 means that a country is perceived as very clean.

HDI2017 The human development index score for 2017: the Human Development Index (HDI) is a measure of achievement in the basic dimensions of human development across countries. It is an index made from a simple unweighted average of a nation's longevity, education and income and is widely accepted in development discourse.

Source

<https://www.transparency.org/en/cpi/2017> CPI data available from https://www.transparency.org/en/cpi/2017. Accessed Feb 24, 2024. Consumer Perception Index 2017 by Transparency International is licensed under CC-BY- ND 4.0.

<https://hdr.undp.org/data-center/human-development-index#/indicies/HDI> HDA data accessed on Oct 1, 2018.

<https://data.worldbank.org/> Population data accessed on Oct 1, 2018.

---

dagp_plot   *Plot posterior distribution from dataframe of posterior draws.*

Description

[Stable]

Plot the posterior distribution of all latent parameters using a dataframe of posterior draws from a
`causact_graph` model.

Usage

```
dagp_plot(drawsDF, densityPlot = FALSE, abbrevLabels = FALSE)
```

Arguments

drawsDF	the dataframe output of `dag_numpyro(mcmc=TRUE)` where each column is a parameter and each row a single draw from a representative sample.
densityPlot	If `TRUE`, each parameter gets its own density plot. If `FALSE` (recommended usage), parameters are grouped into facets based on whether they share the same prior or not. 10 and 90 percent credible intervals are displayed for the posterior distributions.
abbrevLabels	If `TRUE`, long labels on the plot are abbreviated to 10 characters. If `FALSE` the entire label is used.

Value

a credible interval plot of all latent posterior distribution parameters.

Examples

```
# A simple example
posteriorDF = data.frame(x = rnorm(100),
y = rexp(100),
z = runif(100))
posteriorDF %>%
dagp_plot(densityPlot = TRUE)

# More complicated example requiring 'numpyro'
## Not run:
# Create a 2 node graph
graph = dag_create() %>%
  dag_node("Get Card","y",
         rhs = bernoulli(theta),
         data = carModelDF$getCard) %>%
  dag_node(descr = "Card Probability by Car",label = "theta",
           rhs = beta(2,2),
           child = "y")
graph %>% dag_render()
```

```
# below requires Tensorflow installation
drawsDF = graph %>% dag_numpyro(mcmc=TRUE)
drawsDF %>% dagp_plot()

## End(Not run)

# A multiple plate example
library(dplyr)
poolTimeGymDF = gymDF %>%
mutate(stretchType = ifelse(yogaStretch == 1,
                            "Yoga Stretch",
                            "Traditional")) %>%
group_by(gymID,stretchType,yogaStretch) %>%
  summarize(nTrialCustomers = sum(nTrialCustomers),
            nSigned = sum(nSigned))
graph = dag_create() %>%
  dag_node("Cust Signed","k",
           rhs = binomial(n,p),
           data = poolTimeGymDF$nSigned) %>%
  dag_node("Probability of Signing","p",
           rhs = beta(2,2),
           child = "k") %>%
  dag_node("Trial Size","n",
           data = poolTimeGymDF$nTrialCustomers,
           child = "k") %>%
  dag_plate("Yoga Stretch","x",
            nodeLabels = c("p"),
            data = poolTimeGymDF$stretchType,
            addDataNode = TRUE) %>%
  dag_plate("Observation","i",
            nodeLabels = c("x","k","n")) %>%
  dag_plate("Gym","j",
            nodeLabels = "p",
            data = poolTimeGymDF$gymID,
            addDataNode = TRUE)
graph %>% dag_render()
## Not run:
# below requires Tensorflow installation
drawsDF = graph %>% dag_numpyro(mcmc=TRUE)
drawsDF %>% dagp_plot()

## End(Not run)
```

---

dag_create	*Create a graph object for drawing a DAG.*

Description

[Stable]

Generates a `causact_graph` graph object that is set-up for drawing DAG graphs.

Usage

```
dag_create()
```

Value

a list object of class `causact_graph` consisting of 6 dataframes. Each data frame is responsible for storing information about nodes, edges, plates, and the relationships among them.

Examples

```
# With `dag_create()` we can create an empty graph and
# add in nodes (`dag_node()`), add edges (`dag_edge`), and
# view the graph with `dag_render()`.
dag_create()
```

---

dag_diagrammer   *Convert graph to Diagrammer object for visualization*

Description

[Stable]

Convert a `causact_graph` to a `DiagrammeR` object for visualization.

Usage

```
dag_diagrammer(
  graph,
  wrapWidth = 24,
  shortLabel = FALSE,
  fillColor = "aliceblue",
  fillColorObs = "cadetblue"
)
```

Arguments

graph	a graph object of class `causact_graph` created using dag_create().
wrapWidth	a required character label that describes the node.
shortLabel	a longer more descriptive character label for the node.
fillColor	a valid R color to be used as the default node fill color.
fillColorObs	a valid R color to be used as the fill color for observed nodes.

Value

a graph object of class `dgr_graph`. Useful for further customizing graph displays using the `DiagrammeR` package.

Examples

```
library("DiagrammeR")
dag_create() %>%
dag_node("Get Card","y",
          rhs = bernoulli(theta),
          data = carModelDF$getCard) %>%
  dag_diagrammer() %>%
  render_graph(title = "DiagrammeR Version of causact_graph")
```

---

dag_dim	*Add dimension information to* causact_graph

Description

Internal function that is used as part of rendering graph or running greta.

Usage

```
dag_dim(graph)
```

Arguments

graph   a graph object of class causact_graph created using dag_create().

Value

a graph object of class causact_graph with populated dimension information.

---

dag_edge	*Add edge (or edges) between nodes*

Description

[Stable]

With a graph object of class causact_graph created from dag_create, add an edge between nodes in the graph. Vector recycling is used for all arguments.

Usage

```
dag_edge(graph, from, to, type = as.character(NA))
```

Arguments

`graph`	a graph object of class `causact_graph`.
`from`	a character vector representing the parent nodes label or description from which the edge is connected.
`to`	the child node label or description from which the edge is connected.
`type`	character string used to represent the DiagrammeR line type (e.g. `"solid"`). Use `type = "extract"` to encourage causact to only pass indexed elements of the parent node to each instance of the child node. Specify `type = "solid"` to override any automated extract behavior.

Value

a graph object of class `dgr_graph` with additional edges created by this function.

Examples

```
# Create a graph with 2 connected nodes
dag_create() %>%
  dag_node("X") %>%
  dag_node("Y") %>%
  dag_edge(from = "X", to = "Y") %>%
  dag_render(shortLabel = TRUE)
```

---

dag_greta	*Generate a representative sample of the posterior distribution*

Description

[Defunct]

This function is currently defunct. It has been superseded by `dag_numpyro()` because of tricky and sometimes unresolvable installation issues related to the greta package's use of tensorflow. If the greta package resolves those issues, this function may return, but please use `dag_numpyro()` as a direct replacement.

Generate a representative sample of the posterior distribution. The input graph object should be of class `causact_graph` and created using `dag_create()`. The specification of a completely consistent joint distribution is left to the user. Helpful error messages are scheduled for future versions of the `causact` package.

Usage

```
dag_greta(graph, mcmc = TRUE, meaningfulLabels = TRUE, ...)
```

Arguments

graph
: a graph object of class `causact_graph` representing a complete and conistent specification of a joint distribution.

mcmc
: a logical value indicating whether to sample from the posterior distribution. When mcmc=FALSE, the greta code is printed to the console, but not executed. The user can cut and paste the code to another script for running line-by-line. This option is most useful for debugging purposes. When mcmc=TRUE, the code is executed and outputs a dataframe of posterior draws.

meaningfulLabels
: a logical value indicating whether to replace the indexed variable names in `draws` with abbreviated names representing the factor value corresponding to the index. This argument is treated as TRUE regardless of user input. The ability to retain numerical indexing will be in a subsequent release.

...
: additional arguments to be passed onto `greta::mcmc()`.

Value

If mcmc=TRUE, returns a dataframe of posterior distribution samples corresponding to the input `causact_graph`. Each column is a parameter and each row a draw from the posterior sample output. If mcmc=FALSE, running `dag_greta` returns a character string of code that would help the user create three objects representing the posterior distribution:

1. `draws`: An mcmc.list object containing raw output from the HMCMC sampler used by `greta`.

2. `drawsDF`: A wide data frame with all latent variables as columns and all draws as rows. This data frame is useful for calculations based on the posterior

3. `tidyDrawsDF`: A long data frame with each draw represented on one line. This data frame is useful for plotting posterior distributions.

Examples

```
## Not run:
library(greta)
graph = dag_create() %>%
  dag_node("Get Card","y",
           rhs = bernoulli(theta),
           data = carModelDF$getCard) %>%
  dag_node(descr = "Card Probability by Car",label = "theta",
           rhs = beta(2,2),
           child = "y") %>%
  dag_node("Car Model","x",
           data = carModelDF$carModel,
           child = "y") %>%
  dag_plate("Car Model","x",
            data = carModelDF$carModel,
            nodeLabels = "theta")

graph %>% dag_render()
gretaCode = graph %>% dag_greta(mcmc=FALSE)
## default functionality returns a data frame
```

```
# below requires Tensorflow installation
drawsDF = graph %>% dag_greta()
drawsDF %>% dagp_plot()

## End(Not run)
```

---

dag_merge   *Merge two non-intersecting* causact_graph *objects*

Description

[Experimental]

Generates a single causact_graph graph object that combines multiple graphs.

Usage

```
dag_merge(graph1, ...)
```

Arguments

graph1	A causact_graph objects to be merged with
...	As many causact_graph's as wish to be merged

Value

a merged graph object of class causact_graph. Useful for creating simple graphs and then merging them into a more complex structure.

Examples

```
# With `dag_merge()` we
# reset the node ID's and all other item ID's,
# bind together the rows of all given graphs, and
# add in nodes and edges later
# with other functions
# to connect the graph.
#
# THE GRAPHS TO BE MERGED MUST BE DISJOINT
# THERE CAN BE NO IDENTICAL NODES OR PLATES
# IN EACH GRAPH TO BE MERGED, AT THIS TIME


g1 = dag_create() %>%
 dag_node("Demand for A","dA",
          rhs = normal(15,4)) %>%
  dag_node("Supply for A","sA",
          rhs = uniform(0,100)) %>%
  dag_node("Profit for A","pA",
          rhs = min(sA,dA)) %>%
```

```
    dag_edge(from = c("dA","sA"),to = c("pA"))


g2 <- dag_create() %>%
  dag_node("Demand for B","dB",
          rhs = normal(20,8)) %>%
  dag_node("Supply for B","sB",
          rhs = uniform(0,100)) %>%
  dag_node("Profit for B","pB",
          rhs = min(sB,dB)) %>%
  dag_edge(from = c("dB","sB"),to = c("pB"))

g1 %>% dag_merge(g2) %>%
  dag_node("Total Profit", "TP",
          rhs = sum(pA,pB)) %>%
  dag_edge(from=c("pA","pB"), to=c("TP")) %>%
  dag_render()
```

---

dag_node   *Add a node to an existing* causact_graph *object*

Description

[Stable]

Add a node to an existing `causact_graph` object. The graph object should be of class `causact_graph` and created using `dag_create()`.

Usage

```
dag_node(
  graph,
  descr = as.character(NA),
  label = as.character(NA),
  rhs = NA,
  child = as.character(NA),
  data = NULL,
  obs = FALSE,
  keepAsDF = FALSE,
  extract = as.logical(NA),
  dec = FALSE,
  det = FALSE
)
```

Arguments

graph	a graph object of class `causact_graph`. An initial object gets created using `dag_create()`.
descr	a longer more descriptive character label for the node.

label	a shorter character label for referencing the node (e.g. "X","beta"). Labels with . in the name will be replaced by _ to ensure inter-operability with Python. Additionally, Python reserved words, like lambda should not be used.
rhs	either a distribution such as uniform, normal, lognormal, bernoulli, etc. or an R expression. Valid values include normal(mu,sigma), normal, and normal(6,2). R computation/expression examples include alpha+beta*x,c(int, coefs), or 1 / exp(-(alpha + beta * x)). Concatenation using c() is NOT supported. If a distribution is given, this is a random/stochastic node, if a formula is given it is a deterministic node once given the values of its parents. Quotes should not be used as all function/computations should consist of R objects, functions, and constants. Common R arithmetic and geometric operators are supported, but less common R expressions may yield errors when running dag_numpyro().
child	an optional character vector of existing node labels. Directed edges from the newly created node to the supplied nodes will be created.
data	a vector or data frame (with observations in rows and variables in columns).
obs	a logical value indicating whether the node is observed. Assumed to be TRUE when data argument is given.
keepAsDF	a logical value indicating whether the data argument should be split into one random variable node per column or kept together as a random matrix for matrix computation. Defaults to creating one node per column of the data frame.
extract	a logical value. When TRUE, child nodes will try to extract an indexed value from this node. When FALSE, the entire random object (e.g. scalar, vector, matrix) is passed to children nodes. Only use this argument when overriding default behavior seen using dag_render().
dec	a logical value indicating whether the node is a decision node. Used to show nodes as rectangles instead of ovals when using dag_render().
det	a logical value indicating whether the node is a deterministic function of its parents Used to draw a double-line (i.e. peripheries = 2) around a shape when using dag_render(). Assumed to be TRUE when rhs is a formula.

Value

a graph object of class causact_graph with an additional node(s).

Examples

```
# Create an empty graph and add 2 nodes by using
# the `dag_node()` function twice
graph2 = dag_create() %>%
  dag_node("Get Card","y",
        rhs = bernoulli(theta),
        data = carModelDF$getCard) %>%
  dag_node(descr = "Card Probability by Car",label = "theta",
          rhs = beta(2,2),
          child = "y")
graph2 %>% dag_render()
```

```
# The Eight Schools Example from Gelman et al.:

schools_dat <- data.frame(y = c(28,  8, -3,  7, -1,  1, 18, 12),
sigma = c(15, 10, 16, 11,  9, 11, 10, 18), schoolName = paste0("School",1:8))

graph = dag_create() %>%
  dag_node("Treatment Effect","y",
           rhs = normal(theta, sigma),
           data = schools_dat$y) %>%
  dag_node("Std Error of Effect Estimates","sigma",
           data = schools_dat$sigma,
           child = "y") %>%
  dag_node("Exp. Treatment Effect","theta",
           child = "y",
           rhs = avgEffect + schoolEffect) %>%
  dag_node("Pop Treatment Effect","avgEffect",
           child = "theta",
           rhs = normal(0,30)) %>%
  dag_node("School Level Effects","schoolEffect",
           rhs = normal(0,30),
           child = "theta") %>%
  dag_plate("Observation","i",nodeLabels = c("sigma","y","theta")) %>%
  dag_plate("School Name","school",
             nodeLabels = "schoolEffect",
             data = schools_dat$schoolName,
             addDataNode = TRUE)

graph %>% dag_render()
## Not run:
# below requires Tensorflow installation
drawsDF = graph %>% dag_numpyro(mcmc=TRUE)
tidyDrawsDF %>% dagp_plot()

## End(Not run)
```

---

dag_numpyro	*Generate a representative sample of the posterior distribution*

Description

Generate a representative sample of the posterior distribution. The input graph object should be of
class `causact_graph` and created using `dag_create()`. The specification of a completely consis-
tent joint distribution is left to the user.

Usage

```
dag_numpyro(
  graph,
```

```
  mcmc = TRUE,
  num_warmup = 1000,
  num_samples = 4000,
  seed = 1234567
)
```

Arguments

graph	a graph object of class `causact_graph` representing a complete and conistent specification of a joint distribution.
mcmc	a logical value indicating whether to sample from the posterior distribution. When mcmc=FALSE, the numpyro code is printed to the console, but not executed. The user can cut and paste the code to another script for running line-by-line. This option is most useful for debugging purposes. When mcmc=TRUE, the code is executed and outputs a dataframe of posterior draws.
num_warmup	an integer value for the number of initial steps that will be discarded while the markov chain finds its way into the typical set.
num_samples	an integer value for the number of samples.
seed	an integer-valued random seed that serves as a starting point for a random number generator. By setting the seed to a specific value, you can ensure the reproducibility and consistency of your results.

Value

If mcmc=TRUE, returns a dataframe of posterior distribution samples corresponding to the input `causact_graph`. Each column is a parameter and each row a draw from the posterior sample output. If mcmc=FALSE, running `dag_numpyro` returns a character string of code that would help the user generate the posterior distribution; useful for debugging.

Examples

```
graph = dag_create() %>%
  dag_node("Get Card","y",
           rhs = bernoulli(theta),
           data = carModelDF$getCard) %>%
  dag_node(descr = "Card Probability by Car",label = "theta",
           rhs = beta(2,2),
           child = "y") %>%
  dag_node("Car Model","x",
           data = carModelDF$carModel,
           child = "y") %>%
  dag_plate("Car Model","x",
            data = carModelDF$carModel,
            nodeLabels = "theta")

graph %>% dag_render()
numpyroCode = graph %>% dag_numpyro(mcmc=FALSE)
## Not run:
## default functionality returns a data frame
# below requires numpyro installation
```

```
drawsDF = graph %>% dag_numpyro()
drawsDF %>% dagp_plot()

## End(Not run)
```

---

dag_plate   *Create a plate representation for repeated nodes.*

Description

[Experimental]

Given a graph object of class `causact_graph`, create collections of nodes that should be repeated i.e. represent multiple instances of a random variable, random vector, or random matrix. When nodes are on more than one plate, graph rendering will treat each unique combination of plates as separate plates.

Usage

```
dag_plate(
  graph,
  descr,
  label,
  nodeLabels,
  data = as.character(NA),
  addDataNode = FALSE,
  rhs = NA
)
```

Arguments

graph	a graph object of class `dgr_graph` created using `dag_create()`.
descr	a longer more descriptive label for the cluster/plate.
label	a short character string to use as an index. Any . in the names is automatically replaced by _ for interoperability with Python.
nodeLabels	a character vector of node labels or descriptions to include in the list of nodes.
data	a vector representing the categorical data whose unique values become the plate index. To use with `addDataNode = TRUE`, this vector should represent observations of a variable that can be coerced to a factor.
addDataNode	a logical value. When `addDataNode = TRUE`, the code attempts to add a node of observed data that is used as an index for extracting the correct parameter from parent nodes that are on the newly created plate. Verify the graphical model using `dag_render()` to ensure correct behavior.

rhs   Optional rhs expression for when addDataNode = TRUE. This can be either a
                       distribution such as uniform, normal, lognormal, bernoulli, etc. or
                       an R expression. Distribution arguments are optional. Valid values include
                       normal(mu,sigma), normal, and normal(6,2). R computation/expression ex-
                       amples include alpha+beta*x. If a distribution is given, this is a random/stochastic
                       node, if a formula is given it is a deterministic node once given the values of its
                       parents. Quotes should not be used as all function/computations should consist
                       of R objects, functions, and constants.

Value

an expansion of the input causact_graph object with an added plate representing the repetition of
nodeLabels for each unique value of data.

Examples

```
# single plate example
graph = dag_create() %>%
dag_node("Get Card","y",
         rhs = bernoulli(theta),
         data = carModelDF$getCard) %>%
  dag_node(descr = "Card Probability by Car",label = "theta",
           rhs = beta(2,2),
           child = "y") %>%
  dag_node("Car Model","x",
           data = carModelDF$carModel,
           child = "y") %>%
  dag_plate("Car Model","x",
            data = carModelDF$carModel,
            nodeLabels = "theta")
graph %>% dag_render()

# multiple plate example
library(dplyr)
poolTimeGymDF = gymDF %>%
mutate(stretchType = ifelse(yogaStretch == 1,
                            "Yoga Stretch",
                            "Traditional")) %>%
group_by(gymID,stretchType,yogaStretch) %>%
  summarize(nTrialCustomers = sum(nTrialCustomers),
            nSigned = sum(nSigned))
graph = dag_create() %>%
  dag_node("Cust Signed","k",
           rhs = binomial(n,p),
           data = poolTimeGymDF$nSigned) %>%
  dag_node("Probability of Signing","p",
           rhs = beta(2,2),
           child = "k") %>%
  dag_node("Trial Size","n",
           data = poolTimeGymDF$nTrialCustomers,
           child = "k") %>%
  dag_plate("Yoga Stretch","x",
```

```
            nodeLabels = c("p"),
            data = poolTimeGymDF$stretchType,
            addDataNode = TRUE) %>%
  dag_plate("Observation","i",
            nodeLabels = c("x","k","n")) %>%
  dag_plate("Gym","j",
            nodeLabels = "p",
            data = poolTimeGymDF$gymID,
            addDataNode = TRUE)
graph %>% dag_render()
```

---

dag_render   *Render the graph as an htmlwidget*

Description

[Stable]

Using a `causact_graph` object, render the graph in the RStudio Viewer.

Usage

```
dag_render(
  graph,
  shortLabel = FALSE,
  wrapWidth = 24,
  width = NULL,
  height = NULL,
  fillColor = "aliceblue",
  fillColorObs = "cadetblue"
)
```

Arguments

graph	a graph object of class `dgr_graph`.
shortLabel	a logical value. If set to `TRUE`, distribution and formula information is suppressed. Meant for communication with non-statistical stakeholders.
wrapWidth	a numeric value. Used to restrict width of nodes. Default is wrap text after 24 characters.
width	a numeric value. an optional parameter for specifying the width of the resulting graphic in pixels.
height	a numeric value. an optional parameter for specifying the height of the resulting graphic in pixels.
fillColor	a valid R color to be used as the default node fill color during `dag_render()`.
fillColorObs	a valid R color to be used as the fill color for observed nodes during `dag_render()`.

Value

Returns an object of class `grViz` and `htmlwidget` that is also rendered in the RStudio viewer for interactive buidling of graphical models.

Examples

```
# Render a simple graph
dag_create() %>%
  dag_node("Demand","X") %>%
  dag_node("Price","Y", child = "X") %>%
  dag_render()

# Hide the mathematical details of a graph
dag_create() %>%
  dag_node("Demand","X") %>%
  dag_node("Price","Y", child = "X") %>%
  dag_render(shortLabel = TRUE)
```

---

delivDF   *117,790 line items associated with 23,339 shipments.*

Description

A dataset containing the line items, mostly parts, asssociated with 23,339 shipments from a US-based warehouse.

Usage

```
delivDF
```

Format

A data frame (tibble) with 117,790 rows and 5 variables:

shipID unique ID for each shipment

plannedShipDate shipment date promised to customer

actualShipDate date the shipment was actually shipped

partID unique part identifier

quantity quantity of partID in shipment

Source

Adam Fleischhacker

---

distributions   *probability distributions*

Description

These functions can be used to define random variables in a causact model.

Usage

```
uniform(min, max, dim = NULL)

normal(mean, sd, dim = NULL, truncation = c(-Inf, Inf))

lognormal(meanlog, sdlog, dim = NULL)

bernoulli(prob, dim = NULL)

binomial(size, prob, dim = NULL)

negative_binomial(size, prob, dim = NULL)

poisson(lambda, dim = NULL)

gamma(shape, rate, dim = NULL)

inverse_gamma(alpha, beta, dim = NULL, truncation = c(0, Inf))

weibull(shape, scale, dim = NULL)

exponential(rate, dim = NULL)

pareto(a, b, dim = NULL)

student(df, mu, sigma, dim = NULL, truncation = c(-Inf, Inf))

laplace(mu, sigma, dim = NULL, truncation = c(-Inf, Inf))

beta(shape1, shape2, dim = NULL)

cauchy(location, scale, dim = NULL, truncation = c(-Inf, Inf))

chi_squared(df, dim = NULL)

logistic(location, scale, dim = NULL, truncation = c(-Inf, Inf))

multivariate_normal(mean, Sigma, dimension = NULL)
```

```
lkj_correlation(eta, dimension = 2)

multinomial(size, prob, dimension = NULL)

categorical(prob, dimension = NULL)

dirichlet(alpha, dimension = NULL)
```

Arguments

min, max	scalar values giving optional limits to `uniform` variables. Like `lower` and `upper`, these must be specified as numerics, unlike `lower` and `upper`, they must be finite. `min` must always be less than `max`.
dim	Currently ignored. If `dag_greta` becomes functional again, this specifies the dimensions of the greta array to be returned, either a scalar or a vector of positive integers. See details.
mean, meanlog, location, mu	
	unconstrained parameters
sd, sdlog, sigma, lambda, shape, rate, df, scale, shape1, shape2, alpha, beta, a, b, eta, size	
	positive parameters, `alpha` must be a vector for `dirichlet`.
truncation	a length-two vector giving values between which to truncate the distribution.
prob	probability parameter ($0 < prob < 1$), must be a vector for `multinomial` and `categorical`
Sigma	positive definite variance-covariance matrix parameter
dimension	Currently ignored. If `dag_greta` becomes functional again, this specifies, the dimension of a multivariate distribution

Details

The discrete probability distributions (`bernoulli`, `binomial`, `negative_binomial`, `poisson`, `multinomial`, `categorical`) can be used when they have fixed values, but not as unknown variables.

For univariate distributions `dim` gives the dimensions of the array to create. Each element will be (independently) distributed according to the distribution. `dim` can also be left at its default of `NULL`, in which case the dimension will be detected from the dimensions of the parameters (provided they are compatible with one another).

For multivariate distributions (`multivariate_normal()`, `multinomial()`, `categorical()`, and `dirichlet()` each row of the output and parameters corresponds to an independent realisation. If a single realisation or parameter value is specified, it must therefore be a row vector (see example). `n_realisations` gives the number of rows/realisations, and `dimension` gives the dimension of the distribution. I.e. a bivariate normal distribution would be produced with `multivariate_normal(...,  dimension = 2)`. The dimension can usually be detected from the parameters.

`multinomial()` does not check that observed values sum to `size`, and `categorical()` does not check that only one of the observed entries is 1. It's the user's responsibility to check their data matches the distribution!

Wherever possible, the parameterizations and argument names of `causact` distributions match commonly used R functions for distributions, such as those in the `stats` or `extraDistr` packages. The following table states the distribution function to which causact's implementation corresponds (this code largely borrowed from the greta package):

causact	reference
uniform	stats::dunif
normal	stats::dnorm
lognormal	stats::dlnorm
bernoulli	extraDistr::dbern
binomial	stats::dbinom
beta_binomial	extraDistr::dbbinom
negative_binomial	stats::dnbinom
hypergeometric	stats::dhyper
poisson	stats::dpois
gamma	stats::dgamma
inverse_gamma	extraDistr::dinvgamma
weibull	stats::dweibull
exponential	stats::dexp
pareto	extraDistr::dpareto
student	extraDistr::dlst
laplace	extraDistr::dlaplace
beta	stats::dbeta
cauchy	stats::dcauchy
chi_squared	stats::dchisq
logistic	stats::dlogis
f	stats::df
multivariate_normal	mvtnorm::dmvnorm
multinomial	stats::dmultinom
categorical	stats::dmultinom (size = 1)
dirichlet	extraDistr::ddirichlet

Examples

```
## Not run:

# a uniform parameter constrained to be between 0 and 1
phi <- uniform(min = 0, max = 1)

# a length-three variable, with each element following a standard normal
# distribution
alpha <- normal(0, 1, dim = 3)

# a length-three variable of lognormals
sigma <- lognormal(0, 3, dim = 3)

# a hierarchical uniform, constrained between alpha and alpha + sigma,
eta <- alpha + uniform(0, 1, dim = 3) * sigma
```

```
# a hierarchical distribution
mu <- normal(0, 1)
sigma <- lognormal(0, 1)
theta <- normal(mu, sigma)

# a vector of 3 variables drawn from the same hierarchical distribution
thetas <- normal(mu, sigma, dim = 3)

# a matrix of 12 variables drawn from the same hierarchical distribution
thetas <- normal(mu, sigma, dim = c(3, 4))

# a multivariate normal variable, with correlation between two elements
# note that the parameter must be a row vector
Sig <- diag(4)
Sig[3, 4] <- Sig[4, 3] <- 0.6
theta <- multivariate_normal(t(rep(mu, 4)), Sig)

# 10 independent replicates of that
theta <- multivariate_normal(t(rep(mu, 4)), Sig, n_realisations = 10)

# 10 multivariate normal replicates, each with a different mean vector,
# but the same covariance matrix
means <- matrix(rnorm(40), 10, 4)
theta <- multivariate_normal(means, Sig, n_realisations = 10)
dim(theta)

# a Wishart variable with the same covariance parameter
theta <- wishart(df = 5, Sigma = Sig)

## End(Not run)
```

---

gymDF   *Dataframe of 44 observations of free crossfit classes data Each obser-*
                                   vation indicates how many students that participated in the free month
                                   of crossfit signed up for the monthly membership afterwards

---

Description

Dataframe of 44 observations of free crossfit classes data Each observation indicates how many
students that participated in the free month of crossfit signed up for the monthly membership after-
wards

Usage

    gymDF

Format

A data frame with 44 rows and 5 variables:

gymID unique gym identifier

nTrialCustomers number of unique customers taking free trial classes

nSigned number of customers from trial that sign up for membership

yogaStretch whether trial classes included a yoga type stretch

timePeriod month number, since inception of company, for which trial period was offered

---

houseDF *Dataframe of 1,460 observations of home sales in Ames, Iowa. Known as The Ames Housing dataset, it was compiled by Dean De Cock for use in data science education. Each observation is a home sale. See* houseDFDescr *for more info.*

Description

Dataframe of 1,460 observations of home sales in Ames, Iowa. Known as The Ames Housing dataset, it was compiled by Dean De Cock for use in data science education. Each observation is a home sale. See houseDFDescr for more info.

Usage

```
houseDF
```

Format

A data frame with 1,460 rows and 37 variables:

SalePrice the property's sale price in dollars. This is the target variable

MSSubClass The building class

MSZoning The general zoning classification

LotFrontage Linear feet of street connected to property

LotArea Lot size in square feet

Street Type of road access

LotShape General shape of property

Utilities Type of utilities available

LotConfig Lot configuration

Neighborhood Physical locations within Ames city limits

BldgType Type of dwelling

HouseStyle Style of dwelling

OverallQual Overall material and finish quality

OverallCond Overall condition rating

YearBuilt Original construction date

YearRemodAdd Remodel date

ExterQual Exterior material quality

ExterCond Present condition of the material on the exterior

BsmtQual Height of the basement

BsmtCond General condition of the basement

BsmtExposure Walkout or garden level basement walls

BsmtUnfSF Unfinished square feet of basement area

TotalBsmtSF Total square feet of basement area

1stFlrSF First Floor square feet

2ndFlrSF Second floor square feet

LowQualFinSF Low quality finished square feet (all floors)

GrLivArea Above grade (ground) living area square feet

FullBath Full bathrooms above grade

HalfBath Half baths above grade

BedroomAbvGr Number of bedrooms above basement level

TotRmsAbvGrd Total rooms above grade (does not include bathrooms)

Functional Home functionality rating

GarageCars Size of garage in car capacity

MoSold Month Sold

YrSold Year Sold

SaleType Type of sale

SaleCondition Condition of sale

Source

Accessed Jan 22, 2019. Kaggle dataset on "House Prices: Advanced Regression Techniques".

---

houseDFDescr | *Dataframe of 523 descriptions of data values from "The Ames Housing dataset", compiled by Dean De Cock for use in data science education. Each observation is a possible value from a variable in the* houseDF *dataset.*

Description

Dataframe of 523 descriptions of data values from "The Ames Housing dataset", compiled by Dean De Cock for use in data science education. Each observation is a possible value from a variable in the houseDF dataset.

Usage

houseDFDescr

Format

A data frame with 260 rows and 2 variables:

varName the name and description of a variable stored in the `houseDF` dataset

varValueDescr The value and accompanying interpretation for values in the `houseDF` dataset

Source

Accessed Jan 22, 2019. Kaggle dataset on "House Prices: Advanced Regression Techniques".

---

install_causact_deps *Install causact's python dependencies like numpyro, arviz, and xarray.*

Description

`install_causact_deps()` installs python, the numpyro and arviz packages, and their direct dependencies.

Usage

```
install_causact_deps()
```

Details

You may be prompted to download and install miniconda if reticulate did not find a non-system installation of python. Miniconda is the only supported installation method for users, as it ensures that the R python installation is isolated from other python installations. All python packages will by default be installed into a self-contained conda or venv environment named "r-causact". Note that "conda" is the only supported method for install.

If you initially declined the miniconda installation prompt, you can later manually install miniconda by running `reticulate::install_miniconda()`.

If you manually configure a python environment with the required dependencies, you can tell R to use it by pointing reticulate at it, commonly by setting an environment variable:

```
Sys.setenv("RETICULATE_PYTHON" = "~/path/to/python-env/bin/python")
```

---

meaningfulLabels   *Store meaningful parameter labels*

Description

[Experimental]

Store meaningful parameter labels as as part of running dag_numpyro(). When numpyro creates posterior distributions for multi-dimensional parameters, it creates an often meaningless number system for the parameter (e.g. beta[1,1], beta[2,1], etc.). Since parameter dimensionality is often determined by a factor, this function creates labels from the factors unqiue values. replaceLabels() applies the text labels stored using this function to the numpyro output. The meaningful parameter names are stored in an environment, cacheEnv.

Usage

```
meaningfulLabels(graph)
```

Arguments

graph   a causact_graph object.

Value

a data frame meaningfulLabels stored in an environment named cacheEnv that contains a lookup table between greta labels and meaningful labels.

---

prodLineDF   *Product line and product category assignments for 12,026 partID's.*

Description

A dataset containing partID attributes.

Usage

```
prodLineDF
```

Format

A data frame (tibble) with 117,790 rows and 5 variables:

partID unique part identifier

productLine a product line associated with the partID

prodCategory a product category associated with the partID

Source

Adam Fleischhacker

---

rbern	*The Bernoulli Distribution*

Description

#' **[Stable]**

Density, distribution function, quantile function and random generation for the benoulli distribution with parameter `prob`.

Usage

```
rbern(n, prob)
```

Arguments

n	number of observations. If `length(n) > 1`, the length is taken to be the number required.
prob	probability of success of each trial

Value

A vector of 0's and 1's representing failure and success.

Examples

```
#Return a random result of a Bernoulli trial given `prob`.
rbern(n =1, prob = 0.5)
```

---

schoolsDF	*This example, often referred to as 8-schools, was popularized by its inclusion in Bayesian Data Analysis (Gelman, Carlin, & Rubin 1997).*

Description

This example, often referred to as 8-schools, was popularized by its inclusion in Bayesian Data Analysis (Gelman, Carlin, & Rubin 1997).

Usage

```
schoolsDF
```

Format

A data frame with 8 rows and 3 variables:

y estimated treatment effect at a particular school

sigma standard error of the treamtment effect estimate

schoolName an identifier for the school represented by this row

---

setDirectedGraphTheme   *Set DiagrammeR defaults for graphical models*

Description

`setDirectedGraph` returns a graph with good defaults.

Usage

```
setDirectedGraphTheme(
  dgrGraph,
  fillColor = "aliceblue",
  fillColorObs = "cadetblue"
)
```

Arguments

`dgrGraph`	A DiagrammeR graph
`fillColor`	Default R color for filling nodes.
`fillColorObs`	R color for filling obeserved nodes.

Value

An updated version of `dgrGraph` with good defaults for graphical models.

return a `dgrGraph` object with the color and shape defaults used by the `causact` package.

Examples

```
library(DiagrammeR)
create_graph() %>% add_node() %>% render_graph()  # default DiagrammeR aesthetics
create_graph() %>% add_node() %>% setDirectedGraphTheme() %>% render_graph() ## causact aesthetics
```

---

ticketsDF   *Dataframe of 55,167 observations of the number of tickets written by NYC precincts each day Data modified from https://github.com/stan-dev/stancon_talks/tree/master/2018/Contributed-Talks/01_auerbach which originally sourced data from https://opendata.cityofnewyork.us/*

Description

Dataframe of 55,167 observations of the number of tickets written by NYC precincts each day Data modified from https://github.com/stan-dev/stancon_talks/tree/master/2018/Contributed-Talks/01_auerbach which originally sourced data from https://opendata.cityofnewyork.us/

Usage

```
ticketsDF
```

Format

A data frame with 55167 rows and 4 variables:

precinct unique precinct identifier representing precinct of issuing officer

date the date on which ticket violations occurred

month_year the month_year extracted from date column

daily_tickets Number of tickets issued out of precinct on this day

---

```
totalBeachgoersRepSample
```
A representative sample from a random variable that represents the annual number of beach goers to Ocean City, MD beaches on hot days. Think of this representative sample as coming from either a prior or posterior distribution. An example using this sample is can be found in The Business Analyst's Guide To Business Analytics at https://www.causact.com/.

Description

A representative sample from a random variable that represents the annual number of beach goers to Ocean City, MD beaches on hot days. Think of this representative sample as coming from either a prior or posterior distribution. An example using this sample is can be found in The Business Analyst's Guide To Business Analytics at https://www.causact.com/.

Usage

```
totalBeachgoersRepSample
```

Format

A 4,000 element vector.

totalBeachgoersRepSample a draw from a representative sample of total beachgoers to Ocean City, MD.

%>% *The magrittr pipe*

Description

causact uses the pipe function, \%>\% to turn function composition into a series of imperative statements.

Value

Pipe a value forward into a function- or call expression and return the function on the rhs with the lhs used as the first argument.

Index