Package ‘WVPlots’

February 19, 2020

Type Package
Title Common Plots for Analysis
Version 1.2.4
Date 2020-02-19
URL https://github.com/WinVector/WVPlots,
     https://winvector.github.io/WVPlots/
Maintainer John Mount <jmount@win-vector.com>
BugReports https://github.com/WinVector/WVPlots/issues
Description Select data analysis plots, under a standardized calling interface implemented on top of 'ggplot2' and 'plotly'.
               Plots of interest include: 'ROC', gain curve, scatter plot with marginal distributions, conditioned scatter plot with marginal densities,
               box and stem with matching theoretical distribution, and density with matching theoretical distribution.
License GPL-2 | GPL-3
VignetteBuilder knitr
Depends R (>= 3.4.0)
Imports ggplot2 (>= 2.2.0), wrapr (>= 1.9.6), sigr (>= 1.0.6), cdata
       (>= 1.1.6), rqdatatable (>= 1.2.7), rquery (>= 1.4.4), utils,
       grid, gridExtra, graphics, mgcv, stats
Suggests knitr, rmarkdown, RUnit, plotly, hexbin
RoxygenNote 7.0.2
ByteCompile true
NeedsCompilation no
Author John Mount [aut, cre],
       Nina Zumel [aut],
       Win-Vector LLC [cph]
Repository CRAN
Date/Publication 2020-02-19 18:40:03 UTC
R topics documented:

BinaryYScatterPlot .................................................. 3
ClevelandDotPlot ..................................................... 4
ConditionalSmoothedScatterPlot .................................... 5
DiscreteDistribution .................................................. 7
DoubleDensityPlot ................................................... 7
DoubleHistogramPlot .................................................. 9
GainCurvePlot ........................................................ 10
GainCurvePlotC ......................................................... 11
GainCurvePlotWithNotation .......................................... 13
HexBinPlot ............................................................ 14
LiftCurvePlot .......................................................... 16
LogLogPlot .............................................................. 17
PairPlot ................................................................. 18
PlotDistCountBinomial ............................................... 20
PlotDistCountNormal .................................................. 21
PlotDistDensityBeta .................................................... 22
PlotDistDensityNormal ............................................... 24
PlotDistHistBeta ....................................................... 25
plotlyROC .............................................................. 26
plot_fit_trajectory ..................................................... 27
plot_Keras_fit_trajectory ............................................. 29
PRPlot ................................................................. 31
PRTPlot ................................................................. 32
ROCPPlot ............................................................... 34
ROCPPlotPair .......................................................... 35
ROCPPlotPair2 .......................................................... 37
run_WVPlots_tests ..................................................... 38
ScatterBoxPlot ........................................................ 40
ScatterBoxPlotH ......................................................... 41
ScatterHist ............................................................. 42
ScatterHistC ............................................................ 44
ScatterHistN ............................................................ 46
ShadedDensity .......................................................... 47
ShadedDensityCenter ................................................... 48
ShadowHist ............................................................. 49
ShadowPlot ............................................................. 50
WVPLOTS ........................................................................ 52
**BinaryYScatterPlot**

Plot a scatter plot of a binary variable with smoothing curve.

**Description**

Plot the scatter plot of a binary variable with a smoothing curve.

**Usage**

```
BinaryYScatterPlot(
  frame,
  xvar,
  yvar,
  title,
  ..., 
  se = FALSE,
  use_glm = TRUE,
  point_color = "black",
  smooth_color = "blue"
)
```

**Arguments**

- `frame`: data frame to get values from
- `xvar`: name of the independent column in frame
- `yvar`: name of the dependent (output or result to be modeled) column in frame
- `title`: title to place on plot
- `...`: no unnamed argument, added to force named binding of later arguments.
- `se`: if TRUE, add error bars (defaults to FALSE). Ignored if useGlm is TRUE
- `use_glm`: if TRUE, "smoothes" with a one-variable logistic regression (defaults to TRUE)
- `point_color`: color for points
- `smooth_color`: color for smoothing line

**Details**

The points are jittered for legibility. By default, a logistic regression fit is used, so that the smoothing curve represents the probability of \( y = 1 \) (as fit by the logistic regression). If use_glm is set to FALSE, a standard smoothing curve (either loess or a spline fit) is used.

**Examples**

```
set.seed(34903490)
x = rnorm(50)
y = 0.5*x^2 + 2*x + rnorm(length(x))
```
ClevelandDotPlot

Plot a Cleveland dot plot.

Description

Plot counts of a categorical variable.

Usage

ClevelandDotPlot(
  frm, xvar, title, ...
  sort = -1, limit_n = NULL, stem = TRUE, color = "black"
)

Arguments

  frm  data frame to get values from
  xvar  name of the independent (input or model) column in frame
  title  title to place on plot
  ...  no unnamed argument, added to force named binding of later arguments.
  sort  if TRUE sort data
  limit_n  if not NULL number of items to plot
  stem  if TRUE add stems/whiskers to plot
  color  color for points and stems

Details

Assumes that xvar is a factor or can be coerced to one (character or integral).

- sort < 0 sorts the factor levels in decreasing order (most frequent level first)
- sort > 0 sorts the factor levels in increasing order (good when used in conjunction with co-ord_flip())

```
frm = data.frame(x=x, y=y, yC=y>=as.numeric(quantile(y, probs=0.8)))
frm$absY <- abs(frm$y)
frm$posY = frm$y > 0
frm$costX = 1
WVPlots::BinaryYScatterPlot(frm, "x", "posY",
  title="Example "Probability of Y" Plot")
```
### ConditionalSmoothedScatterPlot

Plot a scatter plot with smoothing line.

#### Description

Plot a scatter plot with a smoothing line; the smoothing window is aligned either left, center or right.

#### Usage

```r
ConditionalSmoothedScatterPlot(
  frame,
  xvar,
  yvar,
  groupvar,
  title,
  ...,
  k = 3,
  align = "center",
)```

- sort = 0 leaves the factor levels in "natural order" – usually alphabetical
- stem = FALSE will plot only the dots, without the stem to the y=0 line.
- limit_n = NULL plots all the levels, N an integer limits to the top N most populous levels

### Examples

```r
set.seed(34903490)
# discrete variable: letters of the alphabet
# frequencies of letters in English
# source: http://en.algoritmy.net/article/40379/Letter-frequency-English
letterFreqs = c(8.167, 1.492, 2.782, 4.253, 12.702, 2.228,
                2.015, 6.094, 6.966, 0.153, 0.772, 4.025, 2.406, 6.749, 7.507, 1.929,
                0.095, 5.987, 6.327, 9.056, 2.758, 0.978, 2.306, 0.150, 1.974, 0.074)
letterFreqs = letterFreqs/100
letterFrame = data.frame(letter = letters, freq=letterFreqs)
# now let's generate letters according to their letter frequencies
N = 1000
randomDraws = data.frame(draw=1:N,
                          letter=sample(letterFrame$letter, size=N, replace=TRUE, prob=letterFrame$freq))
WVPlots::ClevelandDotPlot(randomDraws, "letter",
                         title = "Example Cleveland-style dot plot")

# Note the use of sort = 0, and that the graph correctly includes counts
# with no occurrences (5, and 7)
WVPlots::ClevelandDotPlot(mtcars, "carb", sort = 0, "Example of counting integer values")
```
point_color = "black",
smooth_color = "black",
palette = "Dark2"
)

Arguments

frame data frame to get values from
xvar name of the independent column in frame. Assumed to be regularly spaced
yvar name of the dependent (output or result to be modeled) column in frame
groupvar name of the grouping column in frame. Can be NULL for an unconditional plot
title title for plot
... no unnamed argument, added to force named binding of later arguments.
k width of smoothing window. Must be odd for a center-aligned plot. Defaults to 3
align smoothing window alignment: 'center', 'left', or 'right'. Defaults to 'center'
point_color color of points, when groupvar is NULL
smooth_color color of smoothing line, when groupvar is NULL
palette name of Brewer palette, when groupvar is non-NULL (can be NULL)

Details

xvar is the continuous independent variable and yvar is the dependent binary variable. Smoothing is by a square window of width k.

If palette is NULL, and groupvar is non-NULL, plot colors will be chosen from the default ggplot2 palette. Setting palette to NULL allows the user to choose a non-Brewer palette, for example with scale_fill_manual.

Examples

y = c(1,2,3,4,5,10,15,18,20,25)
x = seq_len(length(y))
df = data.frame(x=x, y=y, group=x>5)
WVPlots::ConditionalSmoothedScatterPlot(df, "x", "y", NULL,
title="left smooth, one groups", align="left")
WVPlots::ConditionalSmoothedScatterPlot(df, "x", "y", "group",
title="left smooth, two groups", align="left")
DiscreteDistribution  
Plot distribution of a single discrete numerical variable.

Description

Similar to calling ClevelandDotPlot with sort = 0 on a numerical x variable that takes on a discrete set of values.

Usage

DiscreteDistribution(frm, xvar, title, ..., stem = TRUE, color = "black")

Arguments

- **frm**: data frame to get values from
- **xvar**: numeric: name of the variable whose distribution is to be plotted
- **title**: title to place on plot
- **...**: no unnamed argument, added to force named binding of later arguments.
- **stem**: if TRUE add whisker/stems to plot
- **color**: color of points and stems

Examples

```r
frmx = data.frame(x = rbinom(1000, 20, 0.5))
WVPlots::DiscreteDistribution(frmx, "x","Discrete example")
```

DoubleDensityPlot  
Plot two density plots conditioned on an outcome variable.

Description

Plot two density plots conditioned on a binary outcome variable.

Usage

```r
DoubleDensityPlot(
    frame, xvar, truthVar, title, ...
    truth_target = NULL,
    palette = "Dark2"
)
```
DoubleDensityPlot

Arguments

frame  data frame to get values from
xvar   name of the independent (input or model) column in frame
truthVar  name of the dependent (output or result to be modeled) column in frame
title  title to place on plot
...  no unnamed argument, added to force named binding of later arguments.
truth_target  if not NULL compare to this scalar value.
palette  name of Brewer palette (can be NULL)

Details

The use case for this visualization is to plot the distribution of a predictive model score (usually the predicted probability of a desired outcome) conditioned on the actual outcome. However, you can use it to compare the distribution of any numerical quantity conditioned on a binary feature. See the examples.

The plot will degrade gracefully in degenerate conditions, for example when only one category is present.

If palette is NULL, plot colors will be chosen from the default ggplot2 palette. Setting palette to NULL allows the user to choose a non-Brewer palette, for example with `scale_fill_manual`.

Examples

```r
mpg = ggplot2::mpg
mpg$trans = gsub("\(.*\)", ', mpg$trans)
WVPlots::DoubleDensityPlot(mpg, "cty", "trans", "City driving mpg by transmission type")

# redo the last plot with a custom palette
cmap = c("auto" = "#b2df8a", "manual" = "#1f78b4")
plt = WVPlots::DoubleDensityPlot(mpg, "cty", "trans",
palette = NULL,
title="City driving mpg by transmission type")
plt + ggplot2::scale_color_manual(values=cmap) +
ggplot2::scale_fill_manual(values=cmap)

set.seed(34903490)
x = rnorm(50)
y = 0.5*x^2 + 2*x + rnorm(length(x))
frm = data.frame(score=x,
    truth=(y>=as.numeric(quantile(y,probs=0.8))),
    stuck=TRUE,
    rare=FALSE)
frm[,rare] = TRUE
WVPlots::DoubleDensityPlot(frm, "score", "truth", title="Example double density plot")
```
DoubleHistogramPlot  

Plot two histograms conditioned on an outcome variable.

Description

Plot two histograms conditioned on a binary outcome variable.

Usage

DoubleHistogramPlot(
  frame,  
  xvar,  
  truthVar,  
  title,  
  ...,  
  palette = "Dark2",  
  breaks = 40
)

Arguments

frame   data frame to get values from
xvar    name of the independent (input or model) column in frame
truthVar name of the dependent (output or result to be modeled) column in frame
title   title to place on plot
...     no unnamed argument, added to force named binding of later arguments.
palette name of Brewer palette (can be NULL)
breaks   breaks to pass to histogram

Details

To distinguish the two conditions, one histogram is plotted upside-down.

The use case for this visualization is to plot a predictive model score (usually the predicted probability of a desired outcome) conditioned on the actual outcome. However, you can use it to compare any numerical quantity conditioned on a binary feature.

If palette is NULL, plot colors will be chosen from the default ggplot2 palette. Setting palette to NULL allows the user to choose a non-Brewer palette, for example with scale_fill_manual.

Examples

set.seed(34903490)
x = rnorm(50)
y = 0.5*x^2 + 2*x + rnorm(length(x))
frm = data.frame(x=x,y=y,yC=y>=as.numeric(quantile(y,probs=0.8)))
GainCurvePlot

Plot the cumulative gain curve of a sort-order.

Description
Plot the cumulative gain curve of a sort-order.

Usage
GainCurvePlot(
  frame,
  xvar,
  truthVar,
  title,
  ...,  
estimate_sig = FALSE,
  large_count = 1000,
  truth_target = NULL,
  model_color = "darkblue",
  wizard_color = "darkgreen",
  shadow_color = "darkgray"
)

Arguments
frame data frame to get values from
xvar name of the independent (input or model score) column in frame
truthVar name of the dependent (output or result to be modeled) column in frame
title title to place on plot
... no unnamed argument, added to force named binding of later arguments.
estimate_sig logical, if TRUE compute significance.
large_count numeric, upper bound target for number of plotting points.
truth_target if not NULL compare to this scalar value.
The use case for this visualization is to compare a predictive model score to an actual outcome (either binary (0/1) or continuous). In this case the gain curve plot measures how well the model score sorts the data compared to the true outcome value.

The x-axis represents the fraction of items seen when sorted by score, and the y-axis represents the cumulative summed true outcome represented by the items seen so far. See, for example, https://www.ibm.com/support/knowledgecenter/SSLVMB_24.0.0/spss/tutorials/mlp_bankloan_outputtype_02.html.

For comparison, GainCurvePlot also plots the "wizard curve": the gain curve when the data is sorted according to its true outcome.

To improve presentation quality, the plot is limited to approximately `large_count` points (default: 1000). For larger data sets, the data is appropriately randomly sampled down before plotting.

Examples

```r
set.seed(34903490)
y = abs(rnorm(20)) + 0.1
x = abs(y + 0.5*rnorm(20))
frm = data.frame(model=x, value=y)
WVPlots::GainCurvePlot(frm, "model", "value",
    title="Example Continuous Gain Curve")
```

---

**GainCurvePlotC**

Plot the cumulative gain curve of a sort-order with costs.

**Description**

Plot the cumulative gain curve of a sort-order with costs.

**Usage**

```r
GainCurvePlotC(
    frame,
    xvar,
    costVar,
    truthVar,
    title,
    ...
    estimate_sig = FALSE,
```
large_count = 1000,
model_color = "darkblue",
wizard_color = "darkgreen",
shadow_color = "darkgray"
)

Arguments

frame data frame to get values from
xvar name of the independent (input or model score) column in frame
costVar cost of each item (drives x-axis sum)
truthVar name of the dependent (output or result to be modeled) column in frame
title title to place on plot
... no unnamed argument, added to force named binding of later arguments.
estimate_sig logical, if TRUE compute significance
large_count numeric, upper bound target for number of plotting points
model_color color for the model curve
wizard_color color for the "wizard" (best possible) curve
shadow_color color for the shaded area under the curve

Details

GainCurvePlotC plots a cumulative gain curve for the case where items have an additional cost, in addition to an outcome value.

The x-axis represents the fraction of total cost experienced when items are sorted by score, and the y-axis represents the cumulative summed true outcome represented by the items seen so far.

For comparison, GainCurvePlotC also plots the "wizard curve": the gain curve when the data is sorted according to its true outcome/cost (the optimal sort order).

To improve presentation quality, the plot is limited to approximately large_count points (default: 1000). For larger data sets, the data is appropriately randomly sampled down before plotting.

See Also

GainCurvePlot

Examples

set.seed(34903490)
y = abs(rnorm(20)) + 0.1
x = abs(y + 0.5*rnorm(20))
frm = data.frame(model=x, value=y)
frm$costs=1
frm$costs[1]=5
WVPlots::GainCurvePlotC(frm, "model", "costs", "value", title="Example Continuous Gain CurveC")
GainCurvePlotWithNotation

Plot the cumulative gain curve of a sort-order with extra notation

Description

Plot the cumulative gain curve of a sort-order with extra notation.

Usage

GainCurvePlotWithNotation(
  frame,
  xvar,
  truthVar,
  title,
  gainx,
  labelfun,
  ..., estimate_sig = FALSE,
  large_count = 1000,
  model_color = "darkblue",
  wizard_color = "darkgreen",
  shadow_color = "darkgray",
  crosshair_color = "red",
  text_color = "black"
)

Arguments

frame data frame to get values from
xvar name of the independent (input or model score) column in frame
truthVar name of the dependent (output or result to be modeled) column in frame
title title to place on plot
gainx the point on the x axis corresponding to the desired label
labelfun a function to return a label for the marked point
... no unarmed argument, added to force named binding of later arguments.
estimate_sig logical, if TRUE compute significance
large_count numeric, upper bound target for number of plotting points
model_color color for the model curve
wizard_color color for the "wizard" (best possible) curve
shadow_color color for the shaded area under the curve
crosshair_color color for the annotation location lines
text_color color for the annotation text
Details

This is the standard gain curve plot (see `GainCurvePlot`) with a label attached to a particular value of x. The label is created by a function `labelfun`, which takes as inputs the x and y coordinates of a label and returns a string (the label).

See Also

`GainCurvePlot`

Examples

```r
set.seed(34903490)
y = abs(rnorm(20)) + 0.1
x = abs(y + 0.5*rnorm(20))
frm = data.frame(model=x, value=y)
gainx = 0.10  # get the top 10% most valuable points as sorted by the model
# make a function to calculate the label for the annotated point
labelfun = function(gx, gy) {
  pctx = gx*100
  pcty = gy*100

  paste("The top ", pctx, ", % most valuable points by the model\n", "are ", pcty, " % of total actual value", sep='')
}
WVPlots::GainCurvePlotWithNotation(frm, "model", "value",
title="Example Gain Curve with annotation",
gainx=gainx,labelfun=labelfun)
```

---

### HexBinPlot

**Build a hex bin plot**

**Description**

Build a hex bin plot with rational color coding.

**Usage**

```r
HexBinPlot(
  d,
  xvar,
  yvar,
  title,
  ...
, lightcolor = "#deebf7",
  darkcolor = "#000000",
  bins = 30,
)```
Arguments

d: data frame
xvar: name of x variable column
yvar: name of y variable column
title: plot title
...: not used, forces later arguments to bind by name
lightcolor: light color for least dense areas
darkcolor: dark color for most dense areas
bins: passed to geom_hex
binwidth: passed to geom_hex
na.rm: passed to geom_hex

Details
Builds a standard ggplot2 hexbin plot, with a color scale such that dense areas are colored darker (the default ggplot2 fill scales will color dense areas lighter).
The user can choose an alternate color scale with endpoints lightcolor and darkcolor; it is up to the user to make sure that lightcolor is lighter than darkcolor.
Requires the hexbin package.

Value
a ggplot2 hexbin plot

See Also
geom_hex

Examples

if(requireNamespace("hexbin", quietly = TRUE)) {
  set.seed(634267)
dframe = data.frame(x = rnorm(1000), y = rnorm(1000))
print(HexBinPlot(dframe, "x", "y", "Example hexbin"))

diamonds = ggplot2::diamonds
print(HexBinPlot(diamonds, "carat", "price", "Diamonds example"))

  # change the colorscale
  print(HexBinPlot(diamonds, "carat", "price", "Diamonds example",
                   lightcolor="#fed98e",}
LiftCurvePlot

Plot the cumulative lift curve of a sort-order.

Description

Plot the cumulative lift curve of a sort-order.

Usage

LiftCurvePlot(
  frame,
  xvar,
  truthVar,
  title,
  ...
  large_count = 1000,
  include_wizard = TRUE,
  truth_target = NULL,
  model_color = "darkblue",
  wizard_color = "darkgreen"
)

Arguments

frame data frame to get values from
xvar name of the independent (input or model score) column in frame
truthVar name of the dependent (output or result to be modeled) column in frame
title title to place on plot
... no unnamed argument, added to force named binding of later arguments.
large_count numeric, upper bound target for number of plotting points
include_wizard logical, if TRUE plot the ideal or wizard plot.
truth_target if not NULL compare to this scalar value.
model_color color for the model curve
wizard_color color for the "wizard" (best possible) curve
The use case for this visualization is to compare a predictive model score to an actual outcome (either binary (0/1) or continuous). In this case the lift curve plot measures how well the model score sorts the data compared to the true outcome value.

The x-axis represents the fraction of items seen when sorted by score, and the y-axis represents the lift seen so far (cumulative value of model over cumulative value of random selection).

For comparison, LiftCurvePlot also plots the "wizard curve": the lift curve when the data is sorted according to its true outcome.

To improve presentation quality, the plot is limited to approximately large_count points (default: 1000). For larger data sets, the data is appropriately randomly sampled down before plotting.

Examples

```
set.seed(34903490)
y = abs(rnorm(20)) + 0.1
x = abs(y + 0.5*rnorm(20))
frm = data.frame(model=x, value=y)
WVPlots::LiftCurvePlot(frm, "model", "value",
                       title="Example Continuous Lift Curve")
```

---

**LogLogPlot**

**Log-log plot**

**Description**

Plot a trend on log-log paper.

**Usage**

```
LogLogPlot(  
  frame,  
  xvar,  
  yvar,  
  title,  
  ...,  
  use_coord_trans = FALSE,  
  point_color = "black",  
  linear_color = "#018571",  
  quadratic_color = "#a6611a",  
  smoothing_color = "blue"
)
```
Arguments

frame data frame to get values from
xvar name of the independent (input or model) column in frame
yvar name of the dependent (output or result to be modeled) column in frame
title title to place on plot
... no unnamed argument, added to force named binding of later arguments.
use_coord_trans logical if TRUE, use coord_trans instead of coord_trans(x = "log10", y = "log10") instead of scale_x_log10() + scale_y_log10() (useful when there is not enough range to show ticks).
point_color the color of the data points
linear_color the color of the linear growth lines
quadratic_color the color of the quadratic growth lines
smoothing_color the color of the smoothing line through the data

Details

This plot is intended for plotting functions that are observed costs or durations as a function of problem size. In this case we expect the ideal or expected cost function to be non-decreasing. Any negative trends are assumed to arise from the noise model. The graph is specialized to compare non-decreasing linear and non-decreasing quadratic growth.

Some care must be taken in drawing conclusions from log-log plots, as the transform is fairly violent. Please see: "(Mar’s Law) Everything is linear if plotted log-log with a fat magic marker" (from Akin’s Laws of Spacecraft Design http://spacecraft.ssl.umd.edu/akins_laws.html), and "So You Think You Have a Power Law" http://bactra.org/weblog/491.html.

Examples

set.seed(5326)
frm = data.frame(x = 1:20)
frm$y <- 5 + frm$x + 0.2 * frm$x * frm$x + 0.1*abs(rnorm(nrow(frm)))
WVPlots::LogLogPlot(frm, "x", "y", title="Example Trend")

Description

Build a pair plot

Creates a matrix of scatterplots, one for each possible pair of variables.
Usage

PairPlot(
    d,
    meas_vars,
    title,
    ..., 
    group_var = NULL, 
    alpha = 1, 
    palette = "Dark2", 
    point_color = "darkgray"
)

Arguments

d   data frame
meas_vars the variables to be plotted
title plot title
...    not used, forces later arguments to bind by name
group_var variable for grouping and colorcoding
alpha alpha for points on plot
palette name of a brewer palette (NULL for ggplot2 default coloring)
point_color point color for monochrome plots (no grouping)

Details

If palette is NULL, and group_var is non-NULL, plot colors will be chosen from the default ggplot2 palette. Setting palette to NULL allows the user to choose a non-Brewer palette, for example with scale_color_manual.

Value

a ggplot2 pair plot

Examples

PairPlot(iris, colnames(iris)[1:4], "Example plot", group_var = "Species")

# custom palette
colormap = c(’#a6611a’, ’#dfc27d’, ’#018571’)
PairPlot(iris, colnames(iris)[1:4], "Example plot", group_var = "Species", palette=NULL) + 
ggplot2::scale_color_manual(values=colormap)

# no color-coding
PairPlot(iris, colnames(iris)[1:4], "Example plot")
PlotDistCountBinomial  
*Plot count data with a theoretical binomial*

**Description**

Compares empirical count data to a binomial distribution.

**Usage**

```r
PlotDistCountBinomial(
  frm, xvar, trial_size, title, ...
  p = NULL, limit_to_observed_range = FALSE,
  count_color = "black", binom_color = "blue"
)
```

**Arguments**

- `frm` : data frame to get values from
- `xvar` : column of `frm` that counts the number of successes for each trial
- `trial_size` : the number of "coin flips" in a trial
- `title` : title to place on plot
- `p` : mean of the binomial. If NULL, use empirical mean
- `limit_to_observed_range` : If TRUE, limit plot to observed counts
- `count_color` : color of empirical distribution
- `binom_color` : color of theoretical binomial

**Details**

This function is useful for comparing the number of successes that occur in a series of trials, all of the same size, to a binomial of a given success-probability.

Plots the empirical distribution of successes, and a theoretical matching binomial. If the mean of the binomial, `p`, is given, the binomial with success-probability `p` is plotted. Otherwise, `p` is taken to be the pooled success rate of the data: `sum(frm[[xvar]]) / (trial_size*nrow(frm))`. The mean of the binomial is reported in the subtitle of the plot (to three significant figures).

If `limit_to_observed_range` is TRUE, the range of the plot will only cover the range of the empirical data. Otherwise, the range of the plot will be `0:trial_size` (the default).
PlotDistCountNormal

Plot distribution details as a histogram plus matching normal

Description

Compares empirical data to a normal distribution with the same mean and standard deviation.

Usage

PlotDistCountNormal(
  frm,
  xvar,
  title,
  ..., 
  binWidth = c(),
  hist_color = "black",
  normal_color = "blue",
  mean_color = "blue",
  sd_color = "blue"
)
Arguments

- **frm**: data frame to get values from
- **xvar**: name of the independent (input or model) column in frame
- **title**: title to place on plot
- **...**: no unarmed argument, added to force named binding of later arguments.
- **binWidth**: width of histogram bins
- **hist_color**: color of empirical histogram
- **normal_color**: color of matching theoretical normal
- **mean_color**: color of mean line
- **sd_color**: color of 1-standard deviation lines (can be NULL)

Details

Plots the histograms of the empirical distribution and of the matching normal distribution. Also plots the mean and plus/minus one standard deviation.

Bin width for the histogram is calculated automatically to yield approximately 50 bins across the range of the data, unless the `binWidth` argument is explicitly passed in. `binWidth` is reported in the subtitle of the plot.

Examples

```r
set.seed(52523)
d <- data.frame(wt=100*rnorm(100))
PlotDistCountNormal(d, 'wt', 'example')

# no sd lines
PlotDistCountNormal(d, 'wt', 'example', sd_color=NULL)
```

Description

Plot empirical rate data as a density with the matching beta distribution.

Usage

```r
PlotDistDensityBeta(
  frm,
  xvar,
  title,
  ...,  
)```
curve_color = "lightgray",
beta_color = "blue",
mean_color = "blue",
sd_color = "darkgray"
)

Arguments

frm data frame to get values from
xvar name of the independent (input or model) column in frame
title title to place on plot
... force later arguments to bind by name
curve_color color for empirical density curve
beta_color color for matching theoretical beta
mean_color color for mean line
sd_color color for 1-standard deviation lines (can be NULL)

Details

Plots the empirical density, the theoretical matching beta, the mean value, and plus/minus one standard deviation from the mean.

Examples

set.seed(52523)
N = 100
pgray = 0.1 # rate of gray horses in the population
herd_size = round(runif(N, min=25, 50))
ngray = rbinom(N, herd_size, pgray)
hdata = data.frame(n_gray=ngray, herd_size=herd_size)

# observed rate of gray horses in each herd
hdata$rate_gray = with(hdata, ngray/herd_size)
title = "Observed prevalence of gray horses in population"
PlotDistDensityBeta(hdata, "rate_gray", title) +
  ggplot2::geom_vline(xintercept = pgray, linetype=4, color="maroon") +
  ggplot2::annotate("text", x=pgray+0.01, y=0.01, hjust="left",
                    label = paste("True prevalence =", pgray))

# no sd lines
PlotDistDensityBeta(hdata, "rate_gray", title,
                     sd_color=NULL)
PlotDistDensityNormal  
*Plot an empirical density with the matching normal distribution*

**Description**

Compares empirical data to a normal distribution with the same mean and standard deviation.

**Usage**

```r
PlotDistDensityNormal(
  frm,
  xvar,
  title,
  ...,  
  adjust = 0.5,
  curve_color = "lightgray",
  normal_color = "blue",
  mean_color = "blue",
  sd_color = "darkgray"
)
```

**Arguments**

- `frm`  
  data frame to get values from
- `xvar`  
  name of the independent (input or model) column in frame
- `title`  
  title to place on plot
- `...`  
  no unarmed argument, added to force named binding of later arguments.
- `adjust`  
  passed to `geom_density`; controls smoothness of density plot
- `curve_color`  
  color for empirical density curve
- `normal_color`  
  color for theoretical matching normal
- `mean_color`  
  color of mean line
- `sd_color`  
  color for 1-standard deviation lines (can be NULL)

**Details**

Plots the empirical density, the theoretical matching normal, the mean value, and plus/minus one standard deviation from the mean.

**See Also**

- `geom_density`
Examples

```r
set.seed(52523)
d <- data.frame(wt=100*rnorm(100))
PlotDistDensityNormal(d,'wt','example')

# no sd lines
PlotDistDensityNormal(d, 'wt', 'example', sd_color=NULL)
```

### Description

Compares empirical rate data to a beta distribution with the same mean and standard deviation.

### Usage

```r
PlotDistHistBeta(
  frm,
  xvar,
  title,
  ...,  # force later arguments to bind by name
  bins = 30,
  hist_color = "darkgray",
  beta_color = "blue",
  mean_color = "blue",
  sd_color = "darkgray"
)
```

### Arguments

- `frm` data frame to get values from
- `xvar` name of the independent (input or model) column in frame
- `title` title to place on plot
- `...` force later arguments to bind by name
- `bins` passed to `geom_histogram()`. Default: 30
- `hist_color` color of empirical histogram
- `beta_color` color of matching theoretical beta
- `mean_color` color of mean line
- `sd_color` color of 1-standard deviation lines (can be NULL)
Details

Plots the histogram of the empirical distribution and the density of the matching beta distribution. Also plots the mean and plus/minus one standard deviation.

The number of bins for the histogram defaults to 30. The binwidth can also be passed in instead of the number of bins.

Value

ggplot2 plot

Examples

```r
set.seed(52523)
N = 100
pgray = 0.1  # rate of gray horses in the population
herd_size = round(runif(N, min=25, 50))
ngray = rbinom(N, herd_size, pgray)
hdata = data.frame(n_gray=ngray, herd_size=herd_size)

# observed rate of gray horses in each herd
hdata$rate_gray = with(hdata, n_gray/herd_size)

title = "Observed prevalence of gray horses in population"

PlotDistHistBeta(hdata, "rate_gray", title) +
  ggplot2::geom_vline(xintercept = pgray, linetype=4, color="maroon") +
  ggplot2::annotate("text", x=pgray+0.01, y=0.01, hjust="left",
                   label = paste("True prevalence =", pgray))

# no sd lines
PlotDistHistBeta(hdata, "rate_gray", title,
                 sd_color=NULL)
```

---

**plotlyROC**

*Use plotly to produce a ROC plot.*

Description

Use plotly to produce a ROC plot.

Usage

```r
plotlyROC(
  d,
  predCol,
  outcomeCol,
  outcomeTarget,
```
plot_fit_trajectory

    title,
    ..., 
    estimate_sig = FALSE 
    }

Arguments

  d               dataframe
  predCol         name of column with numeric predictions
  outcomeCol      name of column with truth
  outcomeTarget   value considered true
  title           character title for plot
  ...             no unnamed argument, added to force named binding of later arguments.
  estimate_sig    logical, if TRUE estimate and display significance of difference from AUC 0.5.

Value

  plotly plot

See Also

  ROCPlot

Examples

if(requireNamespace("plotly", quietly = TRUE)) {
  set.seed(34903490)
  x = rnorm(50)
  y = 0.5*x^2 + 2*x + rnorm(length(x))
  frm = data.frame(x=x,yC=y>=as.numeric(quantile(y,probs=0.8)))
  plotlyROC(frm, 'x', 'yC', TRUE, 'example plot', estimate_sig = TRUE)
}

plot_fit_trajectory  Plot the trajectory of a model fit.

Description

Plot a history of model fit performance over the a trajectory of times.
Usage

```r
plot_fit_trajectory(
  d,
  column_description,
  title,
  ...,  
  epoch_name = "epoch",
  needs_flip = c(),
  pick_metric = NULL,
  discount_rate = NULL,
  draw_ribbon = FALSE,
  draw_segments = FALSE,
  val_color = "#d95f02",
  train_color = "#1b9e77",
  pick_color = "#e6ab02"
)
```

Arguments

- `d` : data frame to get values from.
- `column_description` : description of column measures (data.frame with columns measure, validation, and training).
- `title` : character title for plot.
- `...` : force later arguments to be bound by name
- `epoch_name` : name for epoch or trajectory column.
- `needs_flip` : character array of measures that need to be flipped.
- `pick_metric` : character metric to maximize.
- `discount_rate` : numeric what fraction of over-fit to subtract from validation performance.
- `draw_ribbon` : present the difference in training and validation performance as a ribbon rather than two curves? (default FALSE)
- `draw_segments` : logical if TRUE draw over-fit/under-fit segments.
- `val_color` : color for validation performance curve
- `train_color` : color for training performance curve
- `pick_color` : color for indicating optimal stopping point

Details

This visualization can be applied to any staged machine learning algorithm. For example one could plot the performance of a gradient boosting machine as a function of the number of trees added. The fit history data should be in the form given in the example below.

Value

ggplot2 plot

See Also

plot_Keras_fit_trajectory

Examples

d <- data.frame(
  epoch = c(1, 2, 3, 4, 5),
  val_loss = c(0.3769818, 0.2996994, 0.2963943, 0.2779052, 0.2842501),
  val_acc = c(0.8722000, 0.8895000, 0.8822000, 0.8899000, 0.8861000),
  loss = c(0.5067290, 0.3002033, 0.2165675, 0.1738829, 0.1410933),
  acc = c(0.7852000, 0.9040000, 0.9303333, 0.9428000, 0.9545333) )

cT <- data.frame(
  measure = c("minus binary cross entropy", "accuracy"),
  training = c("loss", "acc"),
  validation = c("val_loss", "val_acc"),
  stringsAsFactors = FALSE)

plt <- plot_fit_trajectory(
  d, 
  column_description = cT, 
  needs_flip = "minus binary cross entropy", 
  title = "model performance by epoch, dataset, and measure", 
  epoch_name = "epoch", 
  pick_metric = "minus binary cross entropy", 
  discount_rate = 0.1)

suppressWarnings(print(plt)) # too few points for loess

plot_Keras_fit_trajectory

Plot the trajectory of a Keras model fit.

Description

Plot a history of model fit performance over the number of training epochs.

Usage

plot_Keras_fit_trajectory(
  d, 
  title, 
  ...
)
epoch_name = "epoch",
lossname = "loss",
loss_pretty_name = "minus binary cross entropy",
perfname = "acc",
perf_pretty_name = "accuracy",
pick_metric = loss_pretty_name,
fliploss = TRUE,
discount_rate = NULL,
draw_ribbon = FALSE,
val_color = "#d95f02",
train_color = "#1b9e77",
pick_color = "#e6ab02"
)

Arguments

d data frame to get values from.
title character title for plot.
... force later arguments to be bound by name
epoch_name name for epoch or trajectory column.
lossname name of training loss column (default 'loss')
loss_pretty_name name for loss on graph (default 'minus binary cross entropy')
perfname name of training performance column (default 'acc')
perf_pretty_name name for performance metric on graph (default 'accuracy')
pick_metric character: metric to maximize (NULL for no pick line - default loss_pretty_name)
fliploss flip the loss so that "larger is better"? (default TRUE)
discount_rate numeric: what fraction of over-fit to subtract from validation performance.
draw_ribbon present the difference in training and validation performance as a ribbon rather than two curves? (default FALSE)
val_color color for validation performance curve
train_color color for training performance curve
pick_color color for indicating optimal stopping point

Details

Assumes a performance matrix that carries information for both training and validation loss, and an additional training and validation performance metric, in the format that a Keras history object returns.

By default, flips the loss so that better performance is larger for both the loss and the performance metric, and then draws a vertical line at the minimum validation loss (maximum flipped validation loss). If you choose not to flip the loss, you should not use the loss as the pick_metric.

The example below gives a fit plot for a history report from Keras R package. Please see http://winvector.github.io/FluidData/PlotExample/KerasPerfPlot.html for some details.
PRPlot

Value

`ggplot2` plot

See Also

`plot_fit_trajectory`

Examples

```r
# example data (from Keras)
d <- data.frame(
  val_loss = c(0.3769818, 0.2996994, 0.2963943, 0.2779052, 0.2842501),
  val_acc  = c(0.8722000, 0.8895000, 0.8822000, 0.8899000, 0.8861000),
  loss     = c(0.5067290, 0.3002033, 0.2165675, 0.1738829, 0.1410933),
  acc      = c(0.7852000, 0.9040000, 0.9303333, 0.9428000, 0.9545333) )

plt <- plot_Keras_fit_trajectory(
  d,
  title = "model performance by epoch, dataset, and measure")

suppressWarnings(print(plt)) # too few points for loess
```

Description

Plot Precision-Recall plot.

Usage

```r
PRPlot(frame, xvar, truthVar, truthTarget, title, ..., estimate_sig = FALSE)
```

Arguments

- `frame`: data frame to get values from
- `xvar`: name of the independent (input or model) column in frame
- `truthVar`: name of the dependent (output or result to be modeled) column in frame
- `truthTarget`: value we consider to be positive
- `title`: title to place on plot
- `...`: no unnamed argument, added to force named binding of later arguments.
- `estimate_sig`: logical, if TRUE compute significance
Details

See http://www.nature.com/nmeth/journal/v13/n8/full/nmeth.3945.html for a discussion of precision and recall, and how the precision/recall plot relates to the ROC plot.

In addition to plotting precision versus recall, PRTPlot reports the best achieved F1 score, and plots an isoline corresponding to that F1 score.

See Also

ROCPlot

Examples

```r
set.seed(34903490)
x = rnorm(50)
y = 0.5*x^2 + 2*x + rnorm(length(x))
frm = data.frame(x=x,y=y,yC=y>=as.numeric(quantile(y,probs=0.8)))
frm$absY <- abs(frm$y)
frm$posY = frm$y > 0
frm$costX = 1
WVPlots::PRTPlot(frm, "x", "yC", TRUE, title="Example Precision-Recall plot")
```

---

PRTPlot  
Plot Precision-Recall or Enrichment-Recall as a function of threshold.

Description

Plot classifier performance metrics as a function of threshold.

Usage

```r
PRTPlot(
  frame,
  predVar,
  truthVar,
  truthTarget,
  title,
  ...
  plotvars = c("precision", "recall"),
  thresholdrange = c(-Inf, Inf),
  linecolor = "black"
)
```
Arguments

frame data frame to get values from
predVar name of the column of predicted scores
truthVar name of the column of actual outcomes in frame
truthTarget value we consider to be positive
title title to place on plot
... no unnamed argument, added to force named binding of later arguments.
plotvars variables to plot, must be at least one of the measures listed below. Defaults to c("precision", "recall")
thresholdrange range of thresholds to plot.
linecolor line color for the plot

Details

For a classifier, the precision is what fraction of predicted positives are true positives; the recall is what fraction of true positives the classifier finds, and the enrichment is the ratio of classifier precision to the average rate of positives. Plotting precision-recall or enrichment-recall as a function of classifier score helps identify a score threshold that achieves an acceptable tradeoff between precision and recall, or enrichment and recall.

In addition to precision/recall, PRTPlot can plot a number of other metrics:

- precision: fraction of predicted positives that are true positives
- recall: fraction of true positives that were predicted to be true
- enrichment: ratio of classifier precision to prevalence of positive class
- sensitivity: the same as recall (also known as the true positive rate)
- specificity: fraction of true negatives to all negatives (or 1 - false_positive_rate)
- false_positive_rate: fraction of negatives predicted to be true over all negatives

For example, plotting sensitivity/false_positive_rate as functions of threshold will "unroll" an ROC Plot.

Plots are in a single column, in the order specified by plotvars.

See Also

PRPlot, ROCPlot

Examples

df <- iris
df$isVersicolor <- with(df, Species=='versicolor')
model = glm(isVersicolor ~ Petal.Length + Petal.Width + Sepal.Length + Sepal.Width, data=df, family=binomial)
df$pred = predict(model, newdata=df, type="response")
WVPlots::PRTPlot(df, "pred", "isVersicolor", TRUE, title="Example Precision-Recall threshold plot")

WVPlots::PRTPlot(df, "pred", "isVersicolor", TRUE, plotvars = c("sensitivity", "specificity", "false_positive_rate"), title="Sensitivity/specificity/FPR as functions of threshold")

---

**ROCPlot**

Plot receiver operating characteristic plot.

### Description

Plot receiver operating characteristic plot.

### Usage

```r
ROCPlot(
  frame,  
  xvar,   
  truthVar,  
  truthTarget,  
  title,  
  ...,  
  estimate_sig = FALSE,  
  returnScores = FALSE,  
  nrep = 100,  
  parallelCluster = NULL,  
  curve_color = "darkblue",  
  fill_color = "black",  
  diag_color = "black"
)
```

### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>frame</td>
<td>data frame to get values from</td>
</tr>
<tr>
<td>xvar</td>
<td>name of the independent (input or model) column in frame</td>
</tr>
<tr>
<td>truthVar</td>
<td>name of the dependent (output or result to be modeled) column in frame</td>
</tr>
<tr>
<td>truthTarget</td>
<td>value we consider to be positive</td>
</tr>
<tr>
<td>title</td>
<td>title to place on plot</td>
</tr>
<tr>
<td>...</td>
<td>no unnamed argument, added to force named binding of later arguments.</td>
</tr>
<tr>
<td>estimate_sig</td>
<td>logical, if TRUE estimate and display significance of difference from AUC 0.5.</td>
</tr>
<tr>
<td>returnScores</td>
<td>logical if TRUE return detailed permutedScores</td>
</tr>
<tr>
<td>nrep</td>
<td>number of permutation repetitions to estimate p values.</td>
</tr>
<tr>
<td>parallelCluster</td>
<td>(optional) a cluster object created by package parallel or package snow.</td>
</tr>
</tbody>
</table>
ROCPlotPair

- curve_color: color of the ROC curve
- fill_color: shading color for the area under the curve
- diag_color: color for the AUC=0.5 line (x=y)

Details

See http://www.nature.com/nmeth/journal/v13/n8/full/nmeth.3945.html for a discussion of true positive and false positive rates, and how the ROC plot relates to the precision/recall plot.

See Also

PRTPlot

Examples

```r
set.seed(34903490)
x = rnorm(50)
y = 0.5*x^2 + 2*x + rnorm(length(x))
frm = data.frame(x=x, yC=y>=as.numeric(quantile(y,probs=0.8)))
WVPlots::ROCPlot(frm, "x", "yC", TRUE, title="Example ROC plot", estimate_sig = TRUE)
```

ROCPlotPair

Compare two ROC plots.

Description

Plot two receiver operating characteristic curves from the same data.frame.

Usage

```r
ROCPlotPair(
  frame,
  xvar1,
  xvar2,
  truthVar,
  truthTarget,
  title,
  ...
  estimate_sig = FALSE,
  returnScores = FALSE,
  nrep = 100,
  parallelCluster = NULL,
  palette = "Dark2"
)
```
Arguments

- `frame` data frame to get values from
- `xvar1` name of the first independent (input or model) column in frame
- `xvar2` name of the second independent (input or model) column in frame
- `truthVar` name of the dependent (output or result to be modeled) column in frame
- `truthTarget` value we consider to be positive
- `title` title to place on plot
- `estimate_sig` logical, if TRUE estimate and display significance of difference from AUC 0.5.
- `returnScores` logical if TRUE return detailed permutedScores
- `nrep` number of permutation repetitions to estimate p values.
- `parallelCluster` (optional) a cluster object created by package parallel or package snow.
- `palette` name of a brewer palette (NULL for ggplot2 default coloring)

Details

The use case for this function is to compare the performance of two models when applied to a data set, where the predictions from both models are columns of the same data frame.

If `palette` is NULL, plot colors will be chosen from the default ggplot2 palette. Setting `palette` to NULL allows the user to choose a non-Brewer palette, for example with `scale_color_manual`.

See Also

- ROCPlot

Examples

```r
set.seed(34903490)
x1 = rnorm(50)
x2 = rnorm(length(x1))
y = 0.2*x2^2 + 0.5*x2 + x1 + rnorm(length(x1))
frm = data.frame(x1=x1,x2=x2,yC=y>=as.numeric(quantile(y,probs=0.8)))
# WVPlots::ROCPlot(frm, "x1", "yC", TRUE, title="Example ROC plot")
# WVPlots::ROCPlot(frm, "x2", "yC", TRUE, title="Example ROC plot")
WVPlots::ROCPlotPair(frm, "x1", "x2", "yC", TRUE,
                      title="Example ROC pair plot", estimate_sig = TRUE)
```
ROCPlotPair2

Compare two ROC plots.

Description

Plot two receiver operating characteristic curves from different data frames.

Usage

ROCPlotPair2(
  nm1,  
  frame1,  
  xvar1,  
  truthVar1,  
  truthTarget1,  
  nm2,  
  frame2,  
  xvar2,  
  truthVar2,  
  truthTarget2,  
  title,  
  ...,  
  estimate_sig = TRUE,  
  returnScores = FALSE,  
  nrep = 100,  
  parallelCluster = NULL,  
  palette = "Dark2"  
)

Arguments

nm1 name of first model
frame1 data frame to get values from
xvar1 name of the first independent (input or model) column in frame
truthVar1 name of the dependent (output or result to be modeled) column in frame
truthTarget1 value we consider to be positive
nm2 name of second model
frame2 data frame to get values from
xvar2 name of the first independent (input or model) column in frame
truthVar2 name of the dependent (output or result to be modeled) column in frame
truthTarget2 value we consider to be positive
title title to place on plot
... no unnamed argument, added to force named binding of later arguments.
estimate_sig       logical, if TRUE estimate and display significance of difference from AUC 0.5.
returnScores      logical if TRUE return detailed permutedScores
nrep             number of permutation repetitions to estimate p values.
parallelCluster  (optional) a cluster object created by package parallel or package snow.
palette          name of Brewer palette to color curves (can be NULL)

Details

Use this curve to compare model predictions to true outcome from two data frames, each of which has its own model predictions and true outcome columns.

If palette is NULL, plot colors will be chosen from the default ggplot2 palette. Setting palette to NULL allows the user to choose a non-Brewer palette, for example with scale_color_manual.

See Also

ROCPlot

Examples

```r
set.seed(34903490)
x1 = rnorm(50)
x2 = rnorm(length(x1))
y = 0.2*x2^2 + 0.5*x2 + x1 + rnorm(length(x1))
frm = data.frame(x1=x1,x2=x2,yC=y>=as.numeric(quantile(y,probs=0.8)))
# WVPlots::ROCPlot(frm, "x1", "yC", TRUE, title="Example ROC plot")
# WVPlots::ROCPlot(frm, "x2", "yC", TRUE, title="Example ROC plot")
WVPlots::ROCPlotPair2('train',frm, "x1", "yC", TRUE, 'test', frm, "x2", "yC", TRUE, title="Example ROC pair plot", estimate_sig = TRUE)
```

---

**run_WVPlots_tests**

Run WVPlots package tests.

Description

For all files with names of the form "^test_.+\.R$" in the package directory unit_tests run all functions with names of the form "^test_.+$" as RUnit tests. Attaches RUnit and pkg, requires RUnit. Stops on error.
run_WVPlots_tests

Usage

run_WVPlots_tests(
  ..., 
  verbose = TRUE,
  package_test_dirs = "unit_tests",
  test_dirs = character(0),
  stop_on_issue = TRUE,
  stop_if_no_tests = TRUE,
  require_RUnit_attached = FALSE,
  require_pkg_attached = TRUE,
  rngKind = "Mersenne-Twister",
  rngNormalKind = "Inversion"
)

Arguments

... not used, force later arguments to bind by name.
verbose logical, if TRUE print more.
package_test_dirs directory names to look for in the installed package.
test_dirs paths to look for tests in.
stop_on_issue logical, if TRUE stop after errors or failures.
stop_if_no_tests logical, if TRUE stop if no tests were found.
require_RUnit_attached logical, if TRUE require RUnit be attached before testing.
require_pkg_attached logical, if TRUE require pkg be attached before testing.
rngKind pseudo-random number generator method name.
rngNormalKind pseudo-random normal generator method name.

Details
Based on Rcpp/doRUnit.R.

Value
RUnit test results (invisible).
ScatterBoxPlot

Plot a scatter box plot.

Description

Plot a boxplot with the data points superimposed.

Usage

ScatterBoxPlot(
  frm,
  xvar,
  yvar,
  title,
  ..., 
  pt_alpha = 0.3,
  pt_color = "black",
  box_color = "black",
  box_fill = "lightgray"
)

Arguments

  frm       data frame to get values from
  xvar      name of the independent column in frame; assumed discrete
  yvar      name of the continuous column in frame
  title     plot title
  ...       (doesn’t take additional arguments, used to force later arguments by name)
  pt_alpha  transparency of points in scatter plot
  pt_color  point color
  box_color boxplot line color
  box_fill  boxplot fill color (can be NA for no fill)

Details

  xvar is a discrete variable and yvar is a continuous variable.

See Also

  ScatterBoxPlotH
**Examples**

```r
classes = c("a", "b", "c")
means = c(2, 4, 3)
names(means) = classes
label = sample(classes, size=1000, replace=TRUE)
meas = means[label] + rnorm(1000)
frm2 = data.frame(label=label,
                 meas = meas)
WVPlots::ScatterBoxPlot(frm2, "label", "meas", pt_alpha=0.2, title="Example Scatter/Box plot")
```

---

**ScatterBoxPlotH**  
*Plot a scatter box plot in horizontal mode.*

**Description**

Plot a boxplot with the data points superimposed. Box plots are aligned horizontally.

**Usage**

```r
ScatterBoxPlotH(
  frm,  
xvar,  
yvar,  
title,  
...,  
pt_alpha = 0.3,  
pt_color = "black",  
box_color = "black",  
box_fill = "lightgray"
)
```

**Arguments**

- `frm`  
  data frame to get values from
- `xvar`  
  name of the continuous column in frame
- `yvar`  
  name of the independent column in frame; assumed discrete
- `title`  
  plot title
- `pt_alpha`  
  transparency of points in scatter plot
- `pt_color`  
  point color
- `box_color`  
  boxplot line color
- `box_fill`  
  boxplot fill color (can be NA for no fill)
Details

`xvar` is a continuous variable and `yvar` is a discrete variable.

See Also

`ScatterBoxPlot`

Examples

classes = c("a", "b", "c")
means = c(2, 4, 3)
names(means) = classes
label = sample(classes, size=1000, replace=TRUE)
meas = means[label] + rnorm(1000)
frm2 = data.frame(label=label,
                  meas = meas)
WVPlots::ScatterBoxPlotH(frm2, "meas", "label", pt_alpha=0.2, title="Example Scatter/Box plot")

---

ScatterHist  

*Plot a scatter plot with marginals.*

Description


Usage

```r
ScatterHist(
  frame,
  xvar,
  yvar,
  title,
  ...
  smoothmethod = "lm",
  estimate_sig = FALSE,
  minimal_labels = TRUE,
  binwidth_x = NULL,
  binwidth_y = NULL,
  adjust_x = 1,
  adjust_y = 1,
  point_alpha = 0.5,
  contour = FALSE,
  point_color = "black",
)```
hist_color = "gray",
smoothing_color = "blue",
density_color = "blue",
contour_color = "blue"
)

Arguments

frame data frame to get values from
xvar name of the independent (input or model) column in frame
yvar name of the dependent (output or result to be modeled) column in frame
title title to place on plot
... no unnamed argument, added to force named binding of later arguments.
smoothmethod (optional) one of 'auto', 'loess', 'gam', 'lm', 'identity', or 'none'.
estimate_sig logical if TRUE and smoothmethod is 'identity' or 'lm', report goodness of fit and significance of relation.
minimal_labels logical drop some annotations
binwidth_x numeric binwidth for x histogram
binwidth_y numeric binwidth for y histogram
adjust_x numeric adjust x density plot
adjust_y numeric adjust y density plot
point_alpha numeric opaqueness of the plot points
contour logical if TRUE add a 2d contour plot
point_color color for scatter plots
hist_color fill color for marginal histograms
smoothing_color color for smoothing line
density_color color for marginal density plots
contour_color color for contour plots

Details

If smoothmethod is:

- 'auto', 'loess' or 'gam': the appropriate smoothing curve is added to the scatterplot.
- 'lm' (the default): the best fit line is added to the scatterplot.
- 'identity': the line x = y is added to the scatterplot. This is useful for comparing model predictions to true outcome.
- 'none': no smoothing line is added to the scatterplot.

If estimate_sig is TRUE and smoothmethod is:

- 'lm': the R-squared of the linear fit is reported.
• 'identity': the R-squared of the exact relation between `xvar` and `yvar` is reported.

Note that the identity R-squared is NOT the square of the correlation between `xvar` and `yvar` (which includes an implicit shift and scale). It is the coefficient of determination between `xvar` and `yvar`, and can be negative. See [https://en.wikipedia.org/wiki/Coefficient_of_determination](https://en.wikipedia.org/wiki/Coefficient_of_determination) for more details. If `xvar` is the output of a model to predict `yvar`, then the identity R-squared, not the lm R-squared, is the correct measure.

If `smoothmethod` is neither 'lm' or 'identity' then `estimate_sig` is ignored.

Value

plot grid

See Also

`ScatterHistC`

Examples

```r
set.seed(34903490)
x = rnorm(50)
y = 0.5*x^2 + 2*x + rnorm(length(x))
frm = data.frame(x=x,y=y)
WVPlots::ScatterHist(frm, "x", "y",
  title= "Example Fit",
  smoothmethod = "gam",
  contour = TRUE)

# Same plot with custom colors
WVPlots::ScatterHist(frm, "x", "y",
  title= "Example Fit",
  smoothmethod = "gam",
  contour = TRUE,
  point_color = "#006d2c", # dark green
  hist_color = "#6baed6", # medium blue
  smoothing_color = "#54278f", # dark purple
  density_color = "#08519c", # darker blue
  contour_color = "#9e9ac8") # lighter purple
```

---

**ScatterHistC**

*Plot a conditional scatter plot with marginals.*

**Description**

Plot a scatter plot conditioned on a discrete variable, with marginal conditional density plots.
Usage

ScatterHistC(
    frame,
    xvar,
    yvar,
    cvar,
    title,
    ...,
    annot_size = 3,
    colorPalette = "Dark2",
    adjust_x = 1,
    adjust_y = 1
)

Arguments

frame        data frame to get values from
xvar         name of the x variable
yvar         name of the y variable
cvar         name of condition variable
title        title to place on plot
...           no unnamed argument, added to force named binding of later arguments.
annot_size   numeric scale annotation text (if present)
colorPalette name of a Brewer palette (see http://colorbrewer2.org/)
adjust_x     numeric: adjust x density plot
adjust_y     numeric: adjust y density plot

Details

xvar and yvar are the coordinates of the points, and cvar is the discrete conditioning variable that indicates which category each point (x,y) belongs to.

Value

plot grid

See Also

ScatterHist

Examples

set.seed(34903490)
frm = data.frame(x=rnorm(50),y=rnorm(50))
frm$cat <- frm$x+frm$y>0
WVPlots::ScatterHistC(frm, "x", "y", "cat", 45
ScatterHistN

Plot a height scatter plot with marginals.

Description
Plot a scatter plot conditioned on a continuous variable, with marginal conditional density plots.

Usage
ScatterHistN(
  frame,
  xvar,
  yvar,
  zvar,
  title,
  ...,
  annot_size = 3,
  colorPalette = "RdYlBu",
  nclus = 3,
  adjust_x = 1,
  adjust_y = 1
)

Arguments
frame          data frame to get values from
xvar           name of the x variable
yvar           name of the y variable
zvar           name of height variable
title          title to place on plot
...            no unnamed argument, added to force named binding of later arguments.
annot_size     numeric: scale annotation text (if present)
colorPalette   name of a Brewer palette (see http://colorbrewer2.org/)
nclus          scalar: number of z-clusters to plot
adjust_x       numeric: adjust x density plot
adjust_y       numeric: adjust y density plot

Details
xvar and yvar are the coordinates of the points, and zvar is the continuous conditioning variable. zvar is partitioned into nclus disjoint ranges (by default, 3), which are then treated as discrete categories. The scatterplot and marginal density plots are color-coded by these categories.
ShadedDensity

See Also

ScatterHistC

Examples

```r
set.seed(34903490)
frm = data.frame(x=rnorm(50),y=rnorm(50))
frm$z <- frm$x+frm$y
WVPlots::ScatterHistN(frm, "x", "y", "z", title="Example Joint Distribution")
```

---

**ShadedDensity**

*Plot the distribution of a variable with a tail shaded*

**Description**

Plot the distribution of a variable with a tail shaded. Annotate with the area of the shaded region.

**Usage**

```r
ShadedDensity(
  frame,  # data frame to get values from
  xvar,   # name of the variable to be density plotted
  threshold,  # boundary value for the tail
  title,  # title to place on plot
  ...,  # no unnamed argument, added to force named binding of later arguments.
  tail = "left",  # which tail to shade, 'left' (default) or 'right'
  linecolor = "darkgray",  # color of density curve
  shading = "darkblue",  # color of shaded region and boundaries
  annotate_area = TRUE  # if TRUE (default), report the area of the shaded region
)
```

**Arguments**

- `frame`
- `xvar`
- `threshold`
- `title`
- `tail`
- `linecolor`
- `shading`
- `annotate_area`
ShadedDensityCenter

**See Also**

ShadedDensityCenter

**Examples**

```r
set.seed(52523)
d = data.frame(meas=rnorm(100))
threshold = -1.5
WVPlots::ShadedDensity(d, "meas", threshold,
   title="Example shaded density plot, left tail")
WVPlots::ShadedDensity(d, "meas", -threshold, tail="right",
   title="Example shaded density plot, right tail")
```

---

**ShadedDensityCenter  Plot the distribution of a variable with a center region shaded**

**Description**

Plot the distribution of a variable with a center region shaded. Annotate with the area of the shaded region.

**Usage**

```r
ShadedDensityCenter(
   frame,
   xvar,
   boundaries,
   title,
   ...
   linecolor = "darkgray",
   shading = "darkblue",
   annotate_area = TRUE
)
```

**Arguments**

- `frame`: data frame to get values from
- `xvar`: name of the variable to be density plotted
- `boundaries`: vector of the min and max boundaries of the shaded region
- `title`: title to place on plot
- `...`: no unnamed argument, added to force named binding of later arguments.
- `linecolor`: color of density curve
- `shading`: color of shaded region and boundaries
- `annotate_area`: if TRUE (default), report the area of the shaded region
ShadowHist

Plot a Shadow Histogram Plot

Description

Plot a histogram of a continuous variable xvar, faceted on a categorical conditioning variable, condvar. Each faceted plot also shows a "shadow plot" of the unconditioned histogram for comparison.

Usage

ShadowHist(
  frm,
  xvar,
  condvar,
  title,
  ...,  
  ncol = 1,
  monochrome = FALSE,
  palette = "Dark2",
  fillcolor = "darkblue",
  bins = 30,
  binwidth = NULL
)

Arguments

  frm       data frame to get values from.
  xvar      name of the primary continuous variable
  condvar   name of conditioning variable (categorical variable, controls faceting).
  title     title to place on plot.
  ...       no unnamed argument, added to force named binding of later arguments.
  ncol      numeric: number of columns in facet_wrap.

See Also

ShadedDensity

Examples

set.seed(52523)
d = data.frame(meas=rnorm(100))
boundaries = c(-1.5, 1.5)
WVPlots::ShadedDensityCenter(d, "meas", boundaries,
                              title="Example center-shaded density plot")
monochrome logical: if TRUE, all facets filled with same color
palette character: if monochrome==FALSE, name of brewer color palette (can be NULL)
fillcolor character: if monochrome==TRUE, name of fill color
bins number of bins. Defaults to thirty.
binwidth width of the bins. Overrides bins.

Details

Currently supports only the bins and binwidth arguments (see geom_histogram), but not the center, boundary, or breaks arguments.

By default, the facet plots are arranged in a single column. This can be changed with the optional ncol argument.

If palette is NULL, and monochrome is FALSE, plot colors will be chosen from the default ggplot2 palette. Setting palette to NULL allows the user to choose a non-Brewer palette, for example with scale_fill_manual. For consistency with previous releases, ShadowHist defaults to monochrome = FALSE, while ShadowPlot defaults to monochrome = TRUE.


Value

a ggplot2 histogram plot

Examples

ShadowHist(iris, "Petal.Length", "Species",
    title = "Petal Length distribution by Species")

# make all the facets the same color
ShadowHist(iris, "Petal.Length", "Species",
    monochrome=TRUE,
    title = "Petal Length distribution by Species")

---

ShadowPlot  Plot a Shadow Bar Plot

Description

Plot a bar chart of row counts conditioned on the categorical variable condvar, faceted on a second categorical variable, refinevar. Each faceted plot also shows a "shadow plot" of the totals conditioned on condvar alone.
**Usage**

```r
ShadowPlot(
  frm,
  condvar,
  refinevar,
  title,
  ..., 
  monochrome = TRUE,
  palette = "Dark2",
  fillcolor = "darkblue",
  ncol = 1
)
```

**Arguments**

- `frm` data frame to get values from.
- `condvar` name of the primary conditioning variable (a categorical variable, controls x-axis).
- `refinevar` name of the second or refining conditioning variable (also a categorical variable, controls faceting).
- `title` title to place on plot.
- `monochrome` logical: if TRUE, all facets filled with same color
- `palette` character: if monochrome==FALSE, name of brewer color palette (can be NULL)
- `fillcolor` character: if monochrome==TRUE, name of fill color for bars
- `ncol` numeric: number of columns in facet_wrap.

**Details**

This plot enables comparisons of subpopulation totals across both `condvar` and `refinevar` simultaneously.

By default, the facet plots are arranged in a single column. This can be changed with the optional `ncol` argument.

If `palette` is NULL, and `monochrome` is FALSE, plot colors will be chosen from the default ggplot2 palette. Setting `palette` to NULL allows the user to choose a non-Brewer palette, for example with `scale_fill_manual`. For consistency with previous releases, `ShadowPlot` defaults to `monochrome = TRUE`, while `ShadowHist` defaults to `monochrome = FALSE`.


**Value**

a ggplot2 bar chart counting examples grouped by `condvar`, faceted by `refinevar`. 
Examples

```r
ShadowPlot(mtcars, "carb", "cyl",
    title = "Number of example cars by carb and cyl counts")
```

# colorcode the facets
```r
ShadowPlot(mtcars, "carb", "cyl",
    monochrome = FALSE,
    title = "Number of example cars by carb and cyl counts")
```

---

**WVPlots**

**WVPlots: Common Plots for Analysis**

**Description**

Select data analysis plots, under a standardized calling interface implemented on top of ggplot2 and plotly. Plots of interest include: ROC, gain curve, scatter plot with marginal distributions, conditioned scatter plot with marginal densities, box and stem with matching theoretical distribution, density with matching theoretical distribution.

**Details**

For more information:

- vignette(package='WVPlots')
- RShowDoc('WVPlots_examples',package='WVPlots')
- Website: https://github.com/WinVector/WVPlots
Index

BinaryYScatterPlot, 3
ClevelandDotPlot, 4
ConditionalSmoothedScatterPlot, 5
DiscreteDistribution, 7
DoubleDensityPlot, 7
DoubleHistogramPlot, 9
GainCurvePlot, 10, 12, 14
GainCurvePlotC, 11
GainCurvePlotWithNotation, 13
gem_density, 24
gem_hex, 15
gem_histogram, 50
HexBinPlot, 14
LiftCurvePlot, 16
LogLogPlot, 17
PairPlot, 18
plot_fit_trajectory, 27, 31
plot_Keras_fit_trajectory, 29, 29
PlotDistCountBinomial, 20
PlotDistCountNormal, 21
PlotDistDensityBeta, 21, 22
PlotDistDensityNormal, 24
PlotDistHistBeta, 21, 25
plotlyROC, 26
PRPlot, 31, 33
PRTPlot, 32, 35
ROCPlot, 27, 32, 33, 34, 36, 38
ROCPlotPair, 35
ROCPlotPair2, 37
run_WVPlots_tests, 38
scale_color_manual, 19, 36, 38
scale_fill_manual, 6, 8, 9, 50, 51
ScatterBoxPlot, 40, 42
ScatterBoxPlotH, 40, 41
ScatterHist, 42, 45
ScatterHistC, 44, 44, 47
ScatterHistN, 46
ShadedDensity, 47, 49
ShadedDensityCenter, 48, 48
ShadowHist, 49, 51
ShadowPlot, 50, 50
WVPlots, 52