Package 'ptools'

July 21, 2022

Title Tools for Poisson Data

Version 1.0.1

Maintainer Andrew Wheeler <apwheele@gmail.com>

Description Functions used for analyzing count data, mostly crime counts. Includes checking difference in two Poisson counts (e-test), checking the fit for a Poisson distribution, small sample tests for counts in bins, Weighted Displacement Difference test (Wheeler and Ratcliffe, 2018) <doi:10.1186/s40163-018-0085-5>, to evaluate crime changes over time in treated/control areas. Additionally includes functions for aggregating spatial data and spatial feature engineering.

License MIT + file LICENSE

URL https://github.com/apwheele/ptools

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

Imports partitions, sp, raster, igraph, RANN, spatstat.geom, spatstat.utils, rgeos, stats, methods

Depends R (>= 3.0.0)

VignetteBuilder knitr

NeedsCompilation no

Author Andrew Wheeler [aut, cre] (<https://orcid.org/0000-0003-2255-1316>)

Repository CRAN

Date/Publication 2022-07-21 14:40:02 UTC

R topics documented:

`bisq_xy` *Bisquare weighted sum*

Description

Given a base X/Y dataset, calculates bisquare weighted sums of points from feature dataset

Usage

```
bisq_xy(base, feat, bandwidth, weight = 1)
```

Arguments

`base`	base dataset (eg gridcells), needs to be SpatialPolygonsDataFrame
`feat`	feature dataset (eg another crime generator), needs to be SpatialPointsDataFrame
`bandwidth`	distances above this value do not contribute to the bi-square weight
`weight`	if 1 (default), does not use attribute weights, else pass in string that is the variable name for weights in `feat`

Details

This generates bi-square distance weighted sums of features within specified distance of the base centroid. Bisquare weights are calculated as:

$$w_{ij} = [1 - (d_{ij}/b)^2]^2$$

where d_ij is the Euclidean distance between the base point and and the feature point. If d < b, then w_ij equals 0. These are then multiplied and summed so each base point gets a cumulative weighted sum. See the GWR book for a reference. Uses loops and calculates all pairwise distances, so can be slow for large base and feature datasets. Consider aggregating/weighting feature dataset if it is too slow. Useful for quantifying features nearby (Groff, 2014), or for egohoods (e.g. spatial smoothing of demographic info, Hipp & Boessen, 2013).

Value

A vector of bi-square weighted sums

References

Fotheringham, A. S., Brunsdon, C., & Charlton, M. (2003). G*eographically weighted regression: the analysis of spatially varying relationships*. John Wiley & Sons.

Groff, E. R. (2014). Quantifying the exposure of street segments to drinking places nearby. *Journal of Quantitative Criminology*, 30(3), 527-548.

Hipp, J. R., & Boessen, A. (2013). Egohoods as waves washing across the city: A new measure of "neighborhoods". Criminology, 51(2), 287-327.

See Also

`dist_xy()` for calculating distance to nearest

`count_xy()` for counting points inside polygon

`kern_xy()` for estimating gaussian density of points for features at base polygon xy coords

`bisq_xy()` to estimate bi-square kernel weights of points for features at base polygon xy coords

`idw_xy()` to estimate inverse distance weights of points for features at base polygon xy coords

Examples

```
data(nyc_cafe); data(nyc_bor)
gr_nyc <- prep_grid(nyc_bor,10000)
gr_nyc$bscafe <- bisq_xy(gr_nyc,nyc_cafe,12000)
```

check_pois *Checks the fit of a Poisson Distribution*

Description

Provides a frequency table to check the fit of a Poisson distribution to empirical data.

Usage

```
check_pois(counts, min_val, max_val, pred, silent = FALSE)
```

Arguments

counts	vector of counts, e.g. c(0,5,1,3,4,6)
min_val	scaler minimum value to generate the grid of results, e.g. 0
max_val	scaler maximum value to generate the grid of results, e.g. max(counts)
pred	can either be a scaler, e.g. mean(counts), or a vector (e.g. predicted values from a Poisson regression)
silent	boolean, do not print mean/var stat messages, only applies when passing scaler for pred (default FALSE)

Details

Given either a scaler mean to test the fit, or a set of predictions (e.g. varying means predicted from a model), checks whether the data fits a given Poisson distribution over a specified set of integers. That is it builds a table of integer counts, and calculates the observed vs the expected distribution according to Poisson. Useful for checking any obvious deviations.

Value

A dataframe with columns

- Int, the integer value
- Freq, the total observed counts within that Integer value
- PoisF, the expected counts according to a Poisson distribution with mean/pred specified
- ResidF, the residual from Freq - PoisF
- Prop, the observed proportion of that integer (0-100 scale)
- PoisD, the expected proportion of that integer (0-100 scale)
- ResidD, the residual from Prop - PoisD

Examples

```
# Example use for constant over the whole sample
set.seed(10)
lambda <- 0.2
x <- rpois(10000,lambda)
pfit <- check_pois(x,0,max(x),mean(x))
print(pfit)
# 82% zeroes is not zero inflated -- expected according to Poisson!

# Example use if you have varying predictions, eg after Poisson regression
n <- 10000
ru <- runif(n,0,10)
x <- rpois(n,lambda=ru)
check_pois(x, 0, 23, ru)

# If you really want to do a statistical test of fit
chi_stat <- sum((pfit$Freq - pfit$PoisF)^2/pfit$PoisF)
df <- length(pfit$Freq) - 2
stats::dchisq(chi_stat, df) #p-value
# I prefer evaluating specific integers though (e.g. zero-inflated, longer-tails, etc.)


# If you want an example with real data, see the WaPo fatal officer involved shootings
w1 <- 'https://raw.githubusercontent.com/washingtonpost/' #too long url!
w2 <- 'data-police-shootings/master/fatal-police-shootings-data.csv'
wapo_url <- paste0(w1,w2)
oid <- read.csv(wapo_url, stringsAsFactors = FALSE)
# Now aggregating to count per day
oid$date_val <- as.Date(oid$date)
#may be biased low if several recent days with 0
date_range <- paste0(seq(as.Date('2015-01-01'),max(oid$date_val),by='days'))
day_counts <- as.data.frame(table(factor(oid$date,levels=date_range)))
check_pois(day_counts$Freq, 0, max(day_counts$Freq)+1, mean(day_counts$Freq))

# Example with varying predictions from a model
day_counts$wd <- weekdays(as.Date(day_counts$Var1))
mod <- stats::glm(Freq ~ as.factor(wd) - 1, family='poisson', data=day_counts)
lin_pred <- exp(predict(mod))
pfit_wd <- check_pois(day_counts$Freq, 0, 11, lin_pred)
print(pfit_wd)
```

---

count_xy *Count of points in polygon*

Description

Given a base X/Y dataset, calculates number of feature points that fall inside

Usage

```
count_xy(base, feat, weight = 1)
```

Arguments

base	base dataset (eg gridcells), needs to be SpatialPolygonsDataFrame
feat	feature dataset (eg another crime generator), needs to be SpatialPointsDataFrame
weight	if 1 (default), does not use weights, else pass in string that is the variable name for weights in feat

Details

This generates a count (or weighted count) of features inside of the base areas. Both should be projected in the same units. Uses sp::over() methods in the function.

Value

A vector of counts (or weighted sums)

References

Wheeler, A. P. (2019). Quantifying the local and spatial effects of alcohol outlets on crime. *Crime & Delinquency*, 65(6), 845-871.

See Also

dist_xy() for calculating distance to nearest

dcount_xy() for counting points within distance of base polygon

kern_xy() for estimating gaussian density of points for features at base polygon xy coords

bisq_xy() to estimate bi-square kernel weights of points for features at base polygon xy coords

idw_xy() to estimate inverse distance weights of points for features at base polygon xy coords

Examples

```
data(nyc_liq); data(nyc_bor)
gr_nyc <- prep_grid(nyc_bor,5000)
gr_nyc$liq_cnt <- count_xy(gr_nyc,nyc_liq)

gr_nyc$table_cnt <- count_xy(gr_nyc,nyc_cafe,'SWC_TABLES')
head(gr_nyc@data)
sp::spplot(gr_nyc,zcol='liq_cnt')
```

dcount_xy *Count of points within distance of polygon*

Description

Given a base X/Y dataset, calculates number of feature points that are within particular distance

Usage

```
dcount_xy(base, feat, d, weight = 1)
```

Arguments

base	base dataset (eg gridcells), needs to be SpatialPolygonsDataFrame
feat	feature dataset (eg another crime generator), needs to be SpatialPointsDataFrame
d	scaler distance to count (based on polygon boundary for base, not centroid)
weight	if 1 (default), does not use weights, else pass in string that is the variable name for weights in feat

Details

This generates a count (or weighted count) of features within specified distance of the base *polygon* border. Both should be projected in the same units. Uses raster::buffer() on feat dataset (which calls rgeos) and sp::over functions.

Value

A vector of counts (or weighted sums)

References

Groff, E. R. (2014). Quantifying the exposure of street segments to drinking places nearby. *Journal of Quantitative Criminology*, 30(3), 527-548.

See Also

dist_xy() for calculating distance to nearest

count_xy() for counting points inside polygon

kern_xy() for estimating gaussian density of points for features at base polygon xy coords

bisq_xy() to estimate bi-square kernel weights of points for features at base polygon xy coords

idw_xy() to estimate inverse distance weights of points for features at base polygon xy coords

Examples

```
data(nyc_cafe); data(nyc_bor)
gr_nyc <- prep_grid(nyc_bor,6000)
gr_nyc$dcafe_8k <- dcount_xy(gr_nyc,nyc_cafe,8000)

gr_nyc$dtabl_5k <- dcount_xy(gr_nyc,nyc_cafe,5000,'SWC_TABLES')
head(gr_nyc@data)
sp::spplot(gr_nyc,zcol='dcafe_8k') #total tables within 8k feet of grid cell
```

`dist_xy` *Distance to nearest based on centroid*

Description

Given a base X/Y dataset, calculates distance to nearest for another feature X/Y dataset

Usage

```
dist_xy(base, feat, bxy = c("x", "y"), fxy = c("x", "y"))
```

Arguments

`base`	base dataset (eg gridcells)
`feat`	feature dataset (eg another crime generator)
`bxy`	vector of strings that define what the base xy fields are defined as, defaults `c('x','y')`
`fxy`	vector of strings that define what the base xy fields are defined as, defaults `c('x','y')`

Details

This generates a distance to nearest, based on the provided x/y coordinates (so if using polygons pass the centroid). This uses kd-trees from RANN, so should be reasonably fast. But I do no projection checking, that is on you. You should not use this with spherical coordinates. Useful for feature engineering for crime generators.

Value

A vector of distances from base dataset xy to the nearest feature xy

References

Caplan, J. M., Kennedy, L. W., & Miller, J. (2011). Risk terrain modeling: Brokering criminological theory and GIS methods for crime forecasting. *Justice Quarterly*, 28(2), 360-381.

Wheeler, A. P., & Steenbeek, W. (2021). Mapping the risk terrain for crime using machine learning. *Journal of Quantitative Criminology*, 37(2), 445-480.

See Also

count_xy() for counting points inside of base polygon

dcount_xy() for counting points within distance of base polygon

kern_xy() for estimating gaussian density of points for features at base polygon xy coords

bisq_xy() for estimate bi-square kernel of points for features at base polygon xy coords

idw_xy() for estimate inverese distance weighted of points for features at base polygon xy coords

Examples

```
data(nyc_bor); data(nyc_cafe)
gr_nyc <- prep_grid(nyc_bor,5000,clip_level=0.3)
gr_nyc$dist_cafe <- dist_xy(gr_nyc,nyc_cafe)
head(gr_nyc@data)

sp::spplot(gr_nyc,zcol='dist_cafe')
```

---

e_test *Poisson E-test*

Description

Tests differences in two Poisson means or rates.

Usage

```
e_test(k1, k2, n1 = 1, n2 = 1, d = 0, eps = 1e-20, silent = FALSE)
```

Arguments

k1	scaler Poisson count
k2	scaler Poisson count
n1	scaler divisor for k1 (e.g. rate per unit time or per area), default 1
n2	scaler divisor for k2 (e.g. rate per unit time or per area), default 1
d	scaler amends the null test by a constant amount, default 0
eps	scaler where to terminate sum in testing larger deviations, default 1e-20
silent	boolean if TRUE, does not print error messages

Details

This e-test tests the differences in two Poisson counts or rates. The null is more formally:

$$k_1/n_1 = k_2/n_2 + d$$

Note, I would be wary using the test for Poisson counts over 100 (the tail approximation in the sums will have issues, as the PMF is so spread out). (It is also the case with *very* large k's, e.g. `e_test(4000,4000)` my function could run out of memory.) In that case may use the n arguments to make it a rate per some unit time (which can change the p-value, although for smaller counts/rates should be very close).

Value

A scaler p-value. Will return -1 if inputs don't make sense and print an error message, e.g. `e_test(0,0)` is undefined and will return a -1.

References

Krishnamoorthy, K., & Thomson, J. (2004). A more powerful test for comparing two Poisson means. *Journal of Statistical Planning and Inference*, 119(1), 23-35.

See Also

`wdd()`, can use that function for a normal based approximation to the difference in Poisson means as well as pre/post designs

Examples

```
# For small N, changes in rates should result in same p-value minus floating point differences
e_test(3,0)
e_test(3,0,2,2)

# Not defined
e_test(0,0) #returns -1 and prints warning

# The same rates
e_test(20,10,4,2)
e_test(10,5,2,1) #not quite the same

# Order of counts/rates should not matter
e_test(6,2) #second example from Krishnamoorthy article
e_test(2,6) #when d=0, can switch arguments and get the same p-value

# These are not the same however, due to how the variance estimates work
e_test(3,2)
e_test(3,1,d=1)
```

hex_area *Get area of hexagon given length of side*

Description

The length of the side is half of the length from vertex to vertex (so height in geom_hex).

Usage

```
hex_area(side)
```

Arguments

side	scaler

Details

For use with ggplot and geom_hex binwidth arguments, which expects arguments in width/height. I want hexagons in maps to be a specific area. See this blog post for a specific use case with ggplot.

Value

A scaler for the width

See Also

hex_wd() for estimating the width given the height hex_dim() for estimating width/height given area

Examples

```
area_check <- 1000
wh <- hex_dim(area_check^2)   #e.g. a square kilometer if spatial units are in meters
area <- hex_area(wh[1]/2)       #inverse operation
all.equal(area_check,sqrt(area))
wi <- hex_wd(wh[1])
all.equal(wh[2],wi)
```

hex_dim *Get dimensions of hexagon given area*

Description

Get dimensions of hexagon given area

Usage

```
hex_dim(area)
```

Arguments

area	scaler

Details

For use with ggplot and geom_hex binwidth arguments, which expects arguments in width/height. I want hexagons in maps to be a specific area. See this blog post for a specific use case with ggplot.

Value

a vector with two elements, first element is the height (vertex to vertex), the second element is the width (side to side)

See Also

hex_wd() for estimating the width given the height hex_area() for estimating the area given side length

Examples

```
area_check <- 1000
wh <- hex_dim(area_check^2)   #e.g. a square kilometer if spatial units are in meters
area <- hex_area(wh[1]/2)       #inverse operation
all.equal(area_check,sqrt(area))
wi <- hex_wd(wh[1])
all.equal(wh[2],wi)
```

hex_wd *Get width of hexagon given height*

Description

Get width of hexagon given height

Usage

```
hex_wd(height)
```

Arguments

height	scaler

Details

For use with ggplot and geom_hex binwidth arguments, which expects arguments in width/height. I want hexagons in maps to be a specific area. See this blog post for a specific use case with ggplot.

Value

A scaler for the width

See Also

hex_area() for estimating the area given side length hex_dim() for estimating width/height given area

Examples

```
area_check <- 1000
wh <- hex_dim(area_check^2)   #e.g. a square kilometer if spatial units are in meters
area <- hex_area(wh[1]/2)       #inverse operation
all.equal(area_check,sqrt(area))
wi <- hex_wd(wh[1])
all.equal(wh[2],wi)
```

idw_xy *Inverse distance weighted sums*

Description

Given a base X/Y dataset, calculates clipped inverse distance weighted sums of points from feature dataset

Usage

```
idw_xy(base, feat, clip = 1, weight = 1)
```

Arguments

base	base dataset (eg gridcells), needs to be SpatialPolygonsDataFrame
feat	feature dataset (eg another crime generator), needs to be SpatialPointsDataFrame
clip	scaler minimum value for weight, default 1 (so weights cannot be below 0)
weight	if 1 (default), does not use weights, else pass in string that is the variable name for weights in feat

Details

This generates a inverse distance weighted sum of features within specified distance of the base centroid. Weights are clipped to never be below clip value, which prevents division by 0 (or division by a very small distance number) Uses loops and calculates all pairwise distances, so can be slow for large base and feature datasets. Consider aggregating/weighting feature dataset if it is too slow. Useful for quantifying features nearby (Groff, 2014), or for egohoods (e.g. spatial smoothing of demographic info, Hipp & Boessen, 2013).

Value

A vector of IDW weighted sums

References

Groff, E. R. (2014). Quantifying the exposure of street segments to drinking places nearby. *Journal of Quantitative Criminology*, 30(3), 527-548.

Hipp, J. R., & Boessen, A. (2013). Egohoods as waves washing across the city: A new measure of "neighborhoods". Criminology, 51(2), 287-327.

See Also

dist_xy() for calculating distance to nearest

count_xy() for counting points inside polygon

kern_xy() for estimating gaussian density of points for features at base polygon xy coords

bisq_xy() to estimate bi-square kernel weights of points for features at base polygon xy coords

idw_xy() to estimate inverse distance weights of points for features at base polygon xy coords

Examples

```
data(nyc_cafe); data(nyc_bor)
gr_nyc <- prep_grid(nyc_bor,6000)
gr_nyc$idwcafe <- idw_xy(gr_nyc,nyc_cafe)

gr_nyc$idwtabl <- idw_xy(gr_nyc,nyc_cafe,weight='SWC_TABLES')
head(gr_nyc@data)
sp::spplot(gr_nyc,zcol='idwtabl') #inverse distance weighted tables
```

---

kern_xy *Kernel density of nearby areas*

Description

Given a base X/Y dataset, calculates guassian kernel density for nearby points in feat dataset

Usage

```
kern_xy(base, feat, bandwidth, weight = 1)
```

Arguments

base	base dataset (eg gridcells), needs to be SpatialPolygonsDataFrame or SpatialPointsDataFrame
feat	feature dataset (eg another crime generator), needs to be SpatialPointsDataFrame
bandwidth	scaler bandwidth for the normal KDE
weight	if 1 (default), does not use weights, else pass in string that is the variable name for weights in feat

Details

This generates a density of nearby features at particular control points (specified by base). Useful for risk terrain style feature engineering given nearby crime generators. Loops through all pairwise distances (and uses dnorm()). So will be slow for large base + feature datasets (although should be OK memory wise). Consider aggregating/weighting data if feat is very large.

Value

A vector of densities (or weighted densities)

References

Caplan, J. M., Kennedy, L. W., & Miller, J. (2011). Risk terrain modeling: Brokering criminological theory and GIS methods for crime forecasting. *Justice Quarterly*, 28(2), 360-381.

Wheeler, A. P., & Steenbeek, W. (2021). Mapping the risk terrain for crime using machine learning. *Journal of Quantitative Criminology*, 37(2), 445-480.

See Also

dist_xy() for calculating distance to nearest

count_xy() for counting points inside polygon

kern_xy() for estimating gaussian density of points for features at base polygon xy coords

bisq_xy() to estimate bi-square kernel weights of points for features at base polygon xy coords

idw_xy() to estimate inverse distance weights of points for features at base polygon xy coords

Examples

```
data(nyc_cafe); data(nyc_bor)
gr_nyc <- prep_grid(nyc_bor,6000)
gr_nyc$kdecafe_5k <- kern_xy(gr_nyc,nyc_cafe,8000)
head(gr_nyc@data)

sp::spplot(gr_nyc,zcol='kdecafe_5k')
```

---

near_strings1 *Strings of Near Repeats*

Description

Identifies cases that are nearby each other in space/time

Usage

```
near_strings1(dat, id, x, y, tim, DistThresh, TimeThresh)
```

Arguments

dat	data frame
id	string for id variable in data frame (should be unique)
x	string for variable that has the x coordinates
y	string for variable that has the y coordinates
tim	string for variable that has the time stamp (should be numeric or datetime)
DistThresh	scaler for distance threshold (in whatever units x/y are in)
TimeThresh	scaler for time threshold (in whatever units tim is in)

Details

This function returns strings of cases nearby in space and time. Useful for near-repeat analysis, or to identify potentially duplicate cases. This particular function is memory safe, although uses loops and will be approximately $O(n^2)$ time (or more specifically choose(n,2)). Tests I have done on my machine 5k rows take only ~10 seconds, but ~100k rows takes around 12 minutes with this code.

Value

A data frame that contains the ids as row.names, and two columns:

- `CompId`, a unique identifier that lets you collapse original cases together
- `CompNum`, the number of linked cases inside of a component

References

Wheeler, A. P., Riddell, J. R., & Haberman, C. P. (2021). Breaking the chain: How arrests reduce the probability of near repeat crimes. *Criminal Justice Review*, 46(2), 236-258.

See Also

`near_strings2()`, which uses kdtrees, so should be faster with larger data frames, although still may run out of memory, and is not 100% guaranteed to return all nearby strings.

Examples

```
# Simplified example showing two clusters
s <- c(0,0,0,4,4)
ccheck <- c(1,1,1,2,2)
dat <- data.frame(x=1:5,y=0,
                  ti=s,
                  id=1:5)
res1 <- near_strings1(dat,'id','x','y','ti',2,1)
print(res1)

#Full nyc_shoot data with this function takes ~40 seconds
library(sp)
data(nyc_shoot)
nyc_shoot$id <- 1:nrow(nyc_shoot) #incident ID can have dups
mh <- nyc_shoot[nyc_shoot$BORO == 'MANHATTAN',]
print(Sys.time())
res <- near_strings1(mh@data,id='id',x='X_COORD_CD',y='Y_COORD_CD',
                     tim='OCCUR_DATE',DistThresh=1500,TimeThresh=3)
print(Sys.time()) #3k shootings takes only ~1 second on my machine
```

`near_strings2` *Strings of Near Repeats using KDtrees*

Description

Identifies cases that are nearby each other in space/time

Usage

```
near_strings2(dat, id, x, y, tim, DistThresh, TimeThresh, k = 300, eps = 1e-04)
```

Arguments

dat	data frame
id	string for id variable in data frame (should be unique)
x	string for variable that has the x coordinates
y	string for variable that has the y coordinates
tim	string for variable that has the time stamp (should be numeric or datetime)
DistThresh	scaler for distance threshold (in whatever units x/y are in)
TimeThresh	scaler for time threshold (in whatever units tim is in)
k,	the k for the max number of neighbors to grab in the nn2 function in RANN package
eps,	the nn2 function returns <=, so to return less (like near_strings1()), needs a small fudge factor

Details

This function returns strings of cases nearby in space and time. Useful for near-repeat analysis, or to identify potentially duplicate cases. This particular function uses kdtrees (from the RANN library). For very large data frames, this will run quite a bit faster than near_strings1 (although still may run out of memory). And it is not 100% guaranteed to grab all of the pairs. Tests I have done on my machine ~100k rows takes around 2 minutes with this code.

Value

A data frame that contains the ids as row.names, and two columns:

- CompId, a unique identifier that lets you collapse original cases together
- CompNum, the number of linked cases inside of a component

References

Wheeler, A. P., Riddell, J. R., & Haberman, C. P. (2021). Breaking the chain: How arrests reduce the probability of near repeat crimes. *Criminal Justice Review*, 46(2), 236-258.

See Also

near_strings1(), which uses loops but is guaranteed to get all pairs of cases and should be memory safe.

Examples

```
# Simplified example showing two clusters
s <- c(0,0,0,4,4)
ccheck <- c(1,1,1,2,2)
dat <- data.frame(x=1:5,y=0,
                  ti=s,
                  id=1:5)
res1 <- near_strings2(dat,'id','x','y','ti',2,1)
```

```
print(res1)


# This runs faster than near_strings1
library(sp)
nyc_shoot$id <- 1:nrow(nyc_shoot)  #incident ID can have dups
print(Sys.time())
res <- near_strings2(nyc_shoot@data,id='id',x='X_COORD_CD',y='Y_COORD_CD',
                     tim='OCCUR_DATE',DistThresh=1500,TimeThresh=3)
print(Sys.time()) #around 4 seconds on my machine
head(res)
```

---

nyc_bor *NYC Boroughs*

Description

Spatial file for New York City Borough outlines without water areas

Usage

```
nyc_bor
```

Format

A SpatialPolygonsDataFrame object of the NYC Boroughs. This is projected (same coordinates as shootings). See the Bytes of the Big Apple for any details on the file.

Source

- https://www1.nyc.gov/assets/planning/download/zip/data-maps/open-data/nybb_21c.zip

---

nyc_cafe *NYC Sidewalk Cafes*

Description

Point locations for sidewalk cafes in NYC

Usage

```
nyc_cafe
```

Format

A SpatialPointsDataFrame with point locations Sidwalk cafes in NYC. Note currently includes only active license locations. Current N around 400 and none in Staten Island.

Source

- https://data.cityofnewyork.us/Business/Sidewalk-Caf-Licenses-and-Applications/qcdj-rwhu

---

nyc_liq *NYC Alcohol Licenses*

Description

Point locations for alcohol locations inside NYC boroughs

Usage

```
nyc_liq
```

Format

A SpatialPointsDataFrame with point locations for alcohol licenses inside of NYC. Note that some of these are not the actual sales place, but another address for the business. Currently over 18,000 addresses.

Source

- https://data.ny.gov/Economic-Development/Liquor-Authority-Current-List-of-Active-Licenses/hrvs-fxs2

---

nyc_shoot *NYPD Open Data on Shootings*

Description

Shootings recorded from the New York City Police Department from 2006 to current.

Usage

```
nyc_shoot
```

Format

A SpatialPointsDataFrame with currently over 20k rows and 21 fields, including date/time and address level geocoordinates for the event. Data from 2006 to currently. See the info on Socrata for the field name codebook.

Source

- https://data.cityofnewyork.us/Public-Safety/NYPD-Shooting-Incident-Data-Year-To-Date-/5ucz-vwe8 for current
- https://data.cityofnewyork.us/Public-Safety/NYPD-Shooting-Incident-Data-Historic-/833y-fsy8 for historical

---

`pai` *Predictive Accuracy Index*

Description

Given a set of predictions and observed counts, returns the PAI (predictive accuracy index), PEI (predictive efficiency index), and the RRI (recovery rate index)

Usage

```
pai(dat, count, pred, area, other = c())
```

Arguments

`dat`	data frame with the predictions, observed counts, and area sizes (can be a vector of ones)
`count`	character specifying the column name for the observed counts (e.g. the out of sample crime counts)
`pred`	character specifying the column name for the predicted counts (e.g. predictions based on a model)
`area`	character specifying the column name for the area sizes (could also be street segment distances, see Drawve & Wooditch, 2019)
`other`	vector of strings for any other column name you want to keep (e.g. an ID variable), defaults to empty `c()`

Details

Given predictions over an entire sample, this returns a dataframe with the sorted best PAI (sorted by density of predicted counts per area). PAI is defined as:

$$PAI = \frac{c_t/C}{a_t/A}$$

Where the numerator is the percent of crimes in cumulative t areas, and the denominator is the percent of the area encompassed. PEI is the observed PAI divided by the best possible PAI if you were a perfect oracle, so is scaled between 0 and 1. RRI is `predicted/observed`, so if you have very bad predictions can return Inf or undefined! See Wheeler & Steenbeek (2019) for the definitions of the different metrics. User note, PEI may behave funny with different sized areas.

Value

A dataframe with the columns:

- `Order`, The order of the resulting rankings
- `Count`, the counts for the original crimes you specified
- `Pred`, the original predictions
- `Area`, the area for the units of analysis
- `Cum*`, the cumulative totals for Count/Pred/Area
- `PCum*`, the proportion cumulative totals, e.g. `CumCount/sum(Count)`
- `PAI`, the PAI stat
- `PEI`, the PEI stat
- `RRI`, the RRI stat (probably should analyze/graph the `log(RRI)`)!

Plus any additional variables specified by `other` at the end of the dataframe.

References

Drawve, G., & Wooditch, A. (2019). A research note on the methodological and theoretical considerations for assessing crime forecasting accuracy with the predictive accuracy index. *Journal of Criminal Justice*, 64, 101625.

Wheeler, A. P., & Steenbeek, W. (2021). Mapping the risk terrain for crime using machine learning. *Journal of Quantitative Criminology*, 37(2), 445-480.

See Also

`pai_summary()` for a summary table of metrics for multiple pai tables given fixed N thresholds

Examples

```
# Making some very simple fake data
crime_dat <- data.frame(id=1:6,
                        obs=c(6,7,3,2,1,0),
                        pred=c(8,4,4,2,1,0))
crime_dat$const <- 1
p1 <- pai(crime_dat,'obs','pred','const')
print(p1)

# Combining multiple predictions, making
# A nice table
crime_dat$rand <- sample(crime_dat$obs,nrow(crime_dat),FALSE)
p2 <- pai(crime_dat,'obs','rand','const')
pai_summary(list(p1,p2),c(1,3,5),c('one','two'))
```

`pai_summary` *Summary Table for Multiple PAI Stats*

Description

Takes a list of multiple PAI summary tables (for different predictions) and returns summaries at fixed area thresholds

Usage

```
pai_summary(pai_list, thresh, labs, wide = TRUE)
```

Arguments

`pai_list,`	list of data frames that have the PAI stats from the `pai` function
`thresh,`	vector of area numbers to select, e.g. 10 would select the top 10 areas, c(10,100) would select the top 10 and the top 100 areas
`labs`	vector of characters that specifies the labels for each PAI dataframe, should be the same length as `pai_list`
`wide`	boolean, if TRUE (default), returns data frame in wide format. Else returns summaries in long format

Details

Given predictions over an entire sample, this returns a dataframe with the sorted best PAI (sorted by density of predicted counts per area). PAI is defined as:

$$PAI = \frac{c_t/C}{a_t/A}$$

Where the numerator is the percent of crimes in cumulative t areas, and the denominator is the percent of the area encompassed. PEI is the observed PAI divided by the best possible PAI if you were a perfect oracle, so is scaled between 0 and 1. RRI is `predicted/observed`, so if you have very bad predictions can return Inf or undefined! See Wheeler & Steenbeek (2019) for the definitions of the different metrics. User note, PEI may behave funny with different sized areas.

Value

A dataframe with the PAI/PEI/RRI, and cumulative crime/predicted counts, for each original table

References

Drawve, G., & Wooditch, A. (2019). A research note on the methodological and theoretical considerations for assessing crime forecasting accuracy with the predictive accuracy index. *Journal of Criminal Justice*, 64, 101625.

Wheeler, A. P., & Steenbeek, W. (2021). Mapping the risk terrain for crime using machine learning. *Journal of Quantitative Criminology*, 37(2), 445-480.

See Also

pai() for a summary table of metrics for multiple pai tables given fixed N thresholds

Examples

```
# Making some very simple fake data
crime_dat <- data.frame(id=1:6,
                        obs=c(6,7,3,2,1,0),
                        pred=c(8,4,4,2,1,0))
crime_dat$const <- 1
p1 <- pai(crime_dat,'obs','pred','const')
print(p1)

# Combining multiple predictions, making
# A nice table
crime_dat$rand <- sample(crime_dat$obs,nrow(crime_dat),FALSE)
p2 <- pai(crime_dat,'obs','rand','const')
pai_summary(list(p1,p2),c(1,3,5),c('one','two'))
```

---

pois_contour *Checks the fit of a Poisson Distribution*

Description

Provides contours (for use in graphs) to show changes in Poisson counts in a pre vs post period.

Usage

```
pois_contour(
  pre_crime,
  post_crime,
  lev = c(-3, 0, 3),
  lr = 5,
  hr = max(pre_crime) * 1.05,
  steps = 1000
)
```

Arguments

pre_crime,	vector of crime counts in the pre period
post_crime,	vector of crime counts in the post period
lev,	vector of Poisson Z-scores to draw the contours at, defaults to c(-3,0,3)
lr,	scaler lower limit for where to draw the contour lines, defaults to 5
hr,	scaler upper limit for where to draw the contour lines, defaults to max(pre_crime)*1.05
steps,	scaler how dense to make the lines, defaults to 1000 steps

Details

Provides a set of contour lines to show whether increases/decreases in Poisson counts between two periods are outside of those expected by chance according to the Poisson distribution based on the normal approximation. Meant to be used in subsequent graphs. Note the approximation breaks down at smaller N values, so below 5 is not typically recommended.

Value

A dataframe with columns

- x, the integer value
- y, the y-value in the graph for expected changes (will not be below 0)
- levels, the associated Z-score level

References

Drake, G., Wheeler, A., Kim, D.-Y., Phillips, S. W., & Mendolera, K. (2021). *The Impact of COVID-19 on the Spatial Distribution of Shooting Violence in Buffalo, NY*. CrimRxiv. https://doi.org/10.21428/cb6ab371.e187aede

Examples

```
# Example use with NYC Shooting Data pre/post Covid lockdowns
# Prepping the NYC shooting data
data(nyc_shoot)
begin_date <- as.Date('03/01/2020', format="%m/%d/%Y")
nyc_shoot$Pre <- ifelse(nyc_shoot$OCCUR_DATE < begin_date,1,0)
nyc_shoot$Post <- nyc_shoot$Pre*-1 + 1
# Note being lazy, some of these PCTs have changed over time
pct_tot <- aggregate(cbind(Pre,Post) ~ PRECINCT, data=nyc_shoot@data, FUN=sum)
cont_lines <- pois_contour(pct_tot$Pre,pct_tot$Post)
# Now making an ugly graph
sp <- split(cont_lines,cont_lines$levels)
plot(pct_tot$Pre,pct_tot$Post)
for (s in sp){
  lines(s$x,s$y,lty=2)
}
# Can see it is slightly overdispersed, but pretty close!
# See https://andrewpwheeler.com/2021/02/02/the-spatial-dispersion-of-nyc-shootings-in-2020/
# For a nicer example using ggplot
```

---

powalt *Power for Small Sample Exact Test*

Description

A helper function to calculate power for different alternative distributions

Usage

```
powalt(SST, p_alt, a = 0.05)
```

Arguments

SST	a small_samptest object created with the small_samptest function
p_alt	vector of alternative probabilities to calculate power for
a	scaler, alpha level for power estimate, default 0.05

Details

This construct a null distribution for small sample statistics for N counts in M bins. Example use cases are to see if a repeat offender have a proclivity to commit crimes on a particular day of the week (see the referenced paper). It can also be used for Benford's analysis of leading/trailing digits for small samples.

Value

A PowerSmallSamp object with slots for:

- permutations, a dataframe that contains the exact probabilities and test statistic values for every possible permutation
- power, the estimated power of the scenario
- alternative, the alternative distribution of probabilities specified
- null, the null distribution (taken from the SST object)
- alpha, the specified alpha level

See Also

small_samptest() for generating the SST object needed to estimate the power

Examples

```
# Counts for different days of the week
d <- c(3,1,2,0,0,0,0) #format N observations in M bins
res <- small_samptest(d=d,type="G")
# Power if someone only commits crime on 4 days of the week
alt_p <- c(1/4,1/4,1/4,1/4,0,0,0)
rp <- powalt(res,alt_p) #need to use previously created SST object
print(rp)

# Example for Benfords analysis
f <- 1:9
p_fd <- log10(1 + (1/f)) #first digit probabilities
#check data from Nigrini page 84
checks <- c(1927.48,27902.31,86241.90,72117.46,81321.75,97473.96,
           93249.11,89658.17,87776.89,92105.83,79949.16,87602.93,
           96879.27,91806.47,84991.67,90831.83,93766.67,88338.72,
           94639.49,83709.28,96412.21,88432.86,71552.16)
```

```
# To make example run a bit faster
checks <- checks[1:10]
# extracting the first digits
fd <- substr(format(checks,trim=TRUE),1,1)
tot <- table(factor(fd, levels=paste(f)))
resG <- small_samptest(d=tot,p=p_fd,type="Chi")
# Lets look at alt under equal probabilities (very conservative)
alt_equal <- rep(1/length(p_fd),length(p_fd))
powalt(resG,alt_equal)
```

`prep_grid` *Creates vector grid cells over study area*

Description

Creates grid cells of given size over particular study area.

Usage

```
prep_grid(outline, size, clip_level = 0, point_over = NULL, point_n = 0)
```

Arguments

`outline`	SpatialPolygon or SpatialPolygonDataFrame that defines the area to draw grid cells over
`size`	scaler for the size of the grid cells (one side), in whatever units the outline is in
`clip_level`	, you can clip grid cells if they are not entirely inside the outlined area, defaults to `0` so any cells at least touching are included
`point_over`	default `NULL`, but can pass in SpatialPoints and will only include grid cells that have at least one point
`point_n`	default 0, only used if passing in `point_over`. Will return only grid cells with greater than `point_n` points

Details

This generates a vector grid over the study area of interest. Intentionally working with vector data for use with other feature engineering helper functions (that can pass in X/Y).

Value

A SpatialPolygonDataFrame object with columns

- `id`, integer id value (not the same as row.names!)
- `x`, x centroid of grid cell
- `y`, y centroid of grid cell
- `cover`, proportion that grid cell is covered by `outline`
- `count`, optional (only if you pass in `point_over`)

References

Wheeler, A. P. (2018). The effect of 311 calls for service on crime in DC at microplaces. *Crime & Delinquency*, 64(14), 1882-1903.

Wheeler, A. P., & Steenbeek, W. (2021). Mapping the risk terrain for crime using machine learning. *Journal of Quantitative Criminology*, 37(2), 445-480.

Examples

```
library(sp) #for sp plot methods
# large grid cells
data(nyc_bor)
res <- prep_grid(nyc_bor,5000)
plot(nyc_bor)
plot(res,border='BLUE',add=TRUE)


# clipping so majority of grid is inside outline
res <- prep_grid(nyc_bor,2000,clip_level=0.5)
plot(nyc_bor)
plot(res,border='BLUE',add=TRUE)

# only grid cells that have at least one shooting
data(nyc_shoot)
res <- prep_grid(nyc_bor,2000,clip_level=0,nyc_shoot)
plot(nyc_bor)
plot(res,border='RED',add=TRUE)
```

---

prep_hexgrid *Creates hexagon grid cells over study area*

Description

Creates hexagon grid cells of given area over particular study area.

Usage

```
prep_hexgrid(outline, area, clip_level = 0, point_over = NULL, point_n = 0)
```

Arguments

`outline`	SpatialPolygon or SpatialPolygonDataFrame that defines the area to draw hexgrid cells over
`area`	scaler for the area of the grid cells in whatever units the outline is in
`clip_level`	, you can clip grid cells if they are not entirely inside the outlined area, defaults to `0` so any cells at least touching are included. Specify as proportion (so should not be greater than 1!)

point_over	default NULL, but can pass in SpatialPoints and will only include grid cells that have at least one point
point_n	default 0, only used if passing in point_over. Will return only grid cells with greater than point_n points

Details

This generates a vector hex grid over the study area of interest. Hexgrids are sometimes preferred over square grid cells to prevent aliasing like artifacts in maps (runs of particular values).

Value

A SpatialPolygonDataFrame object with columns

- id, integer id value (not the same as row.names!)
- x, x centroid of grid cell
- y, y centroid of grid cell
- cover, optional (only if clip_level > 0) proportion that grid cell is covered by outline
- count, optional (only if you pass in point_over), total N of points over

References

Circo, G. M., & Wheeler, A. P. (2021). Trauma Center Drive Time Distances and Fatal Outcomes among Gunshot Wound Victims. *Applied Spatial Analysis and Policy*, 14(2), 379-393.

Examples

```
library(sp) #for sp plot methods
#Base example, some barely touch
hnyc <- prep_hexgrid(nyc_bor,area=20000^2)
plot(hnyc)
plot(nyc_bor,border='red',add=TRUE)

#Example clipping hexagons that have dongle hexagons
hex_clip <- prep_hexgrid(nyc_bor,area=20000^2,clip_level=0.3)
plot(hex_clip,border='blue',add=TRUE)
plot(nyc_bor,border='red',add=TRUE)
summary(hnyc)

#Example clipping hexagons with no overlap crimes
hnyc <- prep_hexgrid(nyc_bor,area=4000^2,point_over=nyc_shoot)
plot(hnyc)
plot(nyc_shoot,pch='.')
```

`small_samptest` *Small Sample Exact Test for Counts in Bins*

Description

Small sample test statistic for counts of N items in bins with particular probability.

Usage

```
small_samptest(d, p = rep(1/length(d), length(d)), type = "G", cdf = FALSE)
```

Arguments

`d`	vector of counts, e.g. c(0,2,1,3,1,4,0) for counts of crimes in days of the week
`p`	vector of baseline probabilities, defaults to equal probabilities in each bin
`type`	string specifying "G" for likelihhood ratio G stat (the default), "V" for Kuipers test (for circular data), "KS" for Komolgrov-Smirnov test, and "Chi" for Chi-square test
`cdf`	if `FALSE` (the default) generates a new permutation vector (using `exactProb`), else pass it a final probability dataset previously created

Details

This construct a null distribution for small sample statistics for N counts in M bins. Example use cases are to see if a repeat offender have a proclivity to commit crimes on a particular day of the week (see the referenced paper). It can also be used for Benford's analysis of leading/trailing digits for small samples. Referenced paper shows G test tends to have the most power, although with circular data may consider Kuiper's test.

Value

A small_sampletest object with slots for:

- `CDF`, a dataframe that contains the exact probabilities and test statistic values for every possible permutation
- `probabilities`, the null probabilities you specified
- `data`, the observed counts you specified
- `test`, the type of test conducted (e.g. G, KS, Chi, etc.)
- `test_stat`, the test statistic for the observed data
- `p_value`, the p-value for the observed stat based on the exact null distribution
- `AggregateStatistics`, here is a reduced form aggregate table for the CDF/p-value calculation

If you wish to save the object, you may want to get rid of the CDF part, it can be quite large. It will have a total of `choose(n+n-1,m-1)` total rows, where m is the number of bins and n is the total counts. So if you have 10 crimes in 7 days of the week, it will result in a dataframe with `choose(7 + 10 - 1,7-1)`, which is 8008 rows. Currently I keep the CDF part though to make it easier to calculate power for a particular test

References

Nigrini, M. J. (2012). *Benford's Law: Applications for forensic accounting, auditing, and fraud detection*. John Wiley & Sons.

Wheeler, A. P. (2016). Testing Serial Crime Events for Randomness in Day-of-Week Patterns with Small Samples. *Journal of Investigative Psychology and Offender Profiling*, 13(2), 148-165.

See Also

`powalt()` for calculating power of a test under alternative

Examples

```
# Counts for different days of the week
d <- c(3,1,1,0,0,1,1) #format N observations in M bins
res <- small_samptest(d=d,type="G")
print(res)

# Example for Benfords analysis
f <- 1:9
p_fd <- log10(1 + (1/f)) #first digit probabilities
#check data from Nigrini page 84
checks <- c(1927.48,27902.31,86241.90,72117.46,81321.75,97473.96,
           93249.11,89658.17,87776.89,92105.83,79949.16,87602.93,
           96879.27,91806.47,84991.67,90831.83,93766.67,88338.72,
           94639.49,83709.28,96412.21,88432.86,71552.16)
# To make example run a bit faster
c1 <- checks[1:10]
#extracting the first digits
fd <- substr(format(c1,trim=TRUE),1,1)
tot <- table(factor(fd, levels=paste(f)))
resG <- small_samptest(d=tot,p=p_fd,type="Chi")
resG

#Can reuse the cdf table if you have the same number of observations
c2 <- checks[11:20]
fd2 <- substr(format(c2,trim=TRUE),1,1)
t2 <- table(factor(fd2, levels=paste(f)))
resG2 <- small_samptest(d=t2,p=p_fd,type="Chi",cdf=resG$CDF)
```

---

vor_sp *Voronoi tesselation from input points*

Description

Given an outline and feature points, calculates Voronoi areas

Usage

```
vor_sp(outline, feat)
```

Arguments

outline	object that can be coerced to a spatstat window via as.owin (so SpatialPolygonDataFrame, SpatialPolygon, owin)
feat	A SpatialPointsDataFrame object (if duplicate X/Y coordinates will get errors)

Details

Outline should be a single polygon area. Uses spatstats dirichlet and window to compute the Voronoi tesselation. Will generate errors if feat has duplicate X/Y points. Useful to create areas for other functions, such as dcount_xy() or count_xy(). Common spatial unit of analysis used in crime research when using points (e.g. intersections and street midpoints).

Value

A SpatialPolygonsDataFrame object, including the dataframe for all the info in the orignal feat@data dataframe.

References

Wheeler, A. P. (2018). The effect of 311 calls for service on crime in DC at microplaces. *Crime & Delinquency*, 64(14), 1882-1903.

Wheeler, A. P. (2019). Quantifying the local and spatial effects of alcohol outlets on crime. *Crime & Delinquency*, 65(6), 845-871.

Examples

```
library(sp) # for sample/coordinates
data(nyc_bor)
nyc_buff <- raster::buffer(nyc_bor,50000)
po <- sp::spsample(nyc_buff,50,'hexagonal')
po$id <- 1:dim(coordinates(po))[1] # turns into SpatialDataFrame
vo <- vor_sp(nyc_buff,po)
plot(vo)
plot(nyc_buff,border='RED',lwd=3, add=TRUE)
```

---

wdd *Estimates the WDD Test*

Description

Estimates the weighted displacement difference test from Wheeler & Ratcliffe, *A simple weighted displacement difference test to evaluate place based crime interventions*, Crime Science

Usage

```
wdd(
  control,
  treated,
  disp_control = c(0, 0),
  disp_treated = c(0, 0),
  time_weights = c(1, 1),
  area_weights = c(1, 1, 1, 1),
  alpha = 0.1,
  silent = FALSE
)
```

Arguments

`control`	vector with counts in pre,post for control area
`treated`	vector with counts in pre,post for treated area
`disp_control`	vector with counts in pre,post for displacement control area (default `c(0,0)`)
`disp_treated`	vector with counts in pre,post for displacement treated area (default `c(0,0)`)
`time_weights`	vector with weights for time periods for pre/post (default `c(1,1)`)
`area_weights`	vector with weights for different sized areas, order is c(control,treated,disp_control,disp_treated) (default `c(1,1,1,1)`)
`alpha`	scaler alpha level for confidence interval (default `0.1`)
`silent`	boolean set to TRUE if you do not want anything printed out (default FALSE)

Details

The wdd (weighted displacement difference) test is an extensions to differences-in-differences when observed count data pre/post in treated control areas. The test statistic (ignoring displacement areas and weights) is:

$$WDD = \Delta T - \Delta Ct$$

where $\Delta T = T_1 - T_0$, the post time period count minus the pre time period count for the treated areas. And ditto for the control areas, Ct. The variance is calculated as:

$$T_1 + T_0 + Ct_1 + Ct_0$$

that is this test uses the normal approximation to the Poisson distribution to calculate the standard error for the WDD. So beware if using very tiny counts – this approximation is less likely to be applicable (or count data that is Poisson, e.g. very overdispersed).

This function also incorporates weights for different examples, such as differing pre/post time periods (e.g. 2 years in pre and 1 year in post), or different area sizes (e.g. a one square mile area vs a two square mile area). The subsequent test statistic can then be interpreted as changes per unit time or changes per unit area (e.g. density) or both per time and density.

Value

A length 9 vector with names:

- `Est_Local` and `SE_Local`, the WDD and its standard error for the local estimates
- `Est_Displace` and `SE_Displace`, the WDD and its standard error for the displacement areas
- `Est_Total` and `SE_Total`, the WDD and its standard error for the combined local/displacement areas
- `Z`, the Z-score for the total estimate
- and the lower and upper confidence intervals, `LowCI` and `HighCI`, for whatever alpha level you specified for the total estimate.

References

Wheeler, A. P., & Ratcliffe, J. H. (2018). A simple weighted displacement difference test to evaluate place based crime interventions. *Crime Science*, 7(1), 1-9.

See Also

`wdd_harm()` for aggregating multiple WDD tests into one metric (e.g. based on crime harm weights)
`e_test()` for checking the difference in two Poisson means

Examples

```
# No weights and no displacement
cont <- c(20,20); treat <- c(20,10)
wdd(cont,treat)

# With Displacement stats
disptreat <- c(30,20); dispcont <- c(30,30)
wdd(cont,treat,dispcont,disptreat)

# With different time periods for pre/post
wdd(cont,treat,time_weights=c(2,1))

# With different area sizes
wdd(cont,treat,dispcont,disptreat,area_weights=c(2,1.5,3,3.2))

# You can technically use this even without pre (so just normal based approximation)
# just put in 0's for the pre data (so does not factor into variance)
res_test <- wdd(c(0,20),c(0,10))
twotail_p <- pnorm(res_test['Z'])*2
print(twotail_p) #~0.068
# e-test is very similar
e_test(20,10) #~0.069
```

wdd_harm *Combines Multiple WDD Tests*

Description

Combines multiple weighted displacement difference tests into one final weighted harm metric.

Usage

```
wdd_harm(est, se, weight, alpha = 0.1, silent = FALSE)
```

Arguments

est	vector with WDD estimates (e.g. difference in crime counts for treated vs controls)
se	vector with standard errors for WDD estimates
weight	vector with weights to aggregate results
alpha	scaler alpha level for confidence interval (default 0.1)
silent	boolean, do not print stat messages (default FALSE)

Details

This test combines multiple wdd estimates with different weights. Created to combine tests for crime harm weights.

Value

A length 5 vector with names:

- HarmEst, the combined harm estimate
- SE_HarmEst its standard error
- Z, the Z-score
- and the lower and upper confidence intervals, LowCI and HighCI, for whatever alpha level you specified.

See Also

wdd() for estimating the individual wdd outcomes

Examples

```
# Creating wdd tests for three different crimes and combining
rob <- wdd(c(20,20),c(20,10))
burg <- wdd(c(30,30),c(25,20))
theft <- wdd(c(80,60),c(70,20))
dat = data.frame(rbind(rob,burg,theft))
# passing those columns now to the wdd_harm function
harm_weights <- c(10,5,1)
wdd_harm(dat$Est_Local,dat$SE_Local,harm_weights)
```

Index