Package 'sharpshootR'

January 31, 2020

Type Package

Title A Soil Survey Toolkit

Description Miscellaneous soil data management, summary, visualization, and conversion utilities to support soil survey.

Version 1.6

Date 2020-01-30

Author Dylan Beaudette [cre],
Jay Skovlin [aut],
Stephen Roecker [aut],
USDA-NRCS Soil Survey Staff [ctb]

Maintainer Dylan Beaudette <dylan.beaudette@usda.gov>

LazyLoad yes

LazyData true

License GPL (>= 3)

Repository CRAN

URL https://github.com/ncss-tech/aqp

BugReports https://github.com/ncss-tech/aqp/issues

Suggests MASS, rgdal, spdep, circlize, rvest, xml2, rgeos, raster, httr, jsonlite, dendextend, testthat, hydromad, latticeExtra

Depends R (>= 3.0.0)

Imports grDevices, graphics, methods, stats, utils, aqp, ape, soilDB, igraph, cluster, lattice, vegan, sp, reshape2, Hmisc, scales, circular, RColorBrewer, plyr, digest, e1071, stringi, parallel, curl

Additional_repositories http://hydromad.catchment.org

RoxygenNote 7.0.2

NeedsCompilation no

Date/Publication 2020-01-30 23:00:03 UTC

R topics documented:

sharpshootR-package . . . 3
aggregateColorPlot . . . 3
amador . . . 4
aspect.plot . . . 5
CDEC.snow.courses . . . 6
CDECquery . . . 7
CDECsnowQuery . . . 10
CDEC_StationInfo . . . 12
component.adj.matrix . . . 13
constantDensitySampling . . . 14
diagnosticPropertyPlot . . . 15
dist.along.grad . . . 16
dueling.dendrograms . . . 18
FFD . . . 19
formatPLSS . . . 20
generateLineHash . . . 21
geomorphBySoilSeries-SSURGO . . . 22
HHM . . . 23
joinAdjacency . . . 24
LL2PLSS . . . 25
monthlyWB . . . 26
multinominal2logical . . . 27
PCP_plot . . . 28
percentileDemo . . . 30
plotAvailWater . . . 31
plotProfileDendrogram . . . 32
plotSoilRelationChordGraph . . . 33
plotSoilRelationGraph . . . 34
plotTransect . . . 37
PLSS2LL . . . 39
polygonAdjacency . . . 41
sample.by.poly . . . 42
sampleRasterStackByMU . . . 43
site_photos_kml . . . 44
SoilTaxonomyDendrogram . . . 45
vizAnnualClimate . . . 47
vizHillslopePosition . . . 49

Index **51**

sharpshootR-package *A collection of functions to support soil survey*

Description

This package contains mish-mash of functionality and sample data related to the daily business of soil survey operations with the USDA-NRCS. Many of the functions are highly specialized and inherit default arguments from the names used by the various NCSS (National Cooperative Soil Survey) databases. A detailed description of this package with links to associated tutorials can be found at the project website.

aggregateColorPlot *Plot aggregate soil color data*

Description

Generate a plot from summaries generated by aqp::aggregateColor().

Usage

```
aggregateColorPlot(x, print.label=TRUE, label.font = 1,
label.cex = 0.65, buffer.pct = 0.02, print.n.hz=FALSE,
rect.border='black', horizontal.borders=FALSE,
horizontal.border.lwd=2, x.axis=TRUE, y.axis=TRUE,
...)
```

Arguments

x	a list, results from aqp::aggregateColor()
print.label	print Munsell color labels inside of rectangles, when they fit
label.font	font specification for color labels
label.cex	font size for color lables
buffer.pct	extra space between labels and color rectangles
print.n.hz	optionally print the number of horizons
rect.border	color for rectangle border
horizontal.borders	optionally add horizontal borders between bands of color
horizontal.border.lwd	line width for horizontal borders
x.axis	logical: add a scale and label to x-axis?
y.axis	logical: add group labels to y-axis?
...	additional arguments passed to plot

Details

Tutorial at http://ncss-tech.github.io/AQP/sharpshootR/aggregate-soil-color.html.

Author(s)

D.E. Beaudette

Examples

```
if(requireNamespace("curl") &
   curl::has_internet() &
   require(aqp) &
   require(soilDB)) {

  data(loafercreek, package = 'soilDB')

# generalize horizon names using REGEX rules
n <- c('Oi', 'A', 'BA','Bt1','Bt2','Bt3','Cr','R')
p <- c('O', '^A$|Ad|Ap|AB','BA$|Bw',
'Bt1$|^B$','^Bt$|^Bt2$','^Bt3|^Bt4|CBt$|BCt$|2Bt|2CB$|^C$','Cr','R')
loafercreek$genhz <- generalize.hz(loafercreek$hzname, n, p)

# remove non-matching generalized horizon names
loafercreek$genhz[loafercreek$genhz == 'not-used'] <- NA
loafercreek$genhz <- factor(loafercreek$genhz)

# aggregate color data, this function is from the `aqp` package
a <- aggregateColor(loafercreek, 'genhz')

# plot
par(mar=c(1,4,4,1))
aggregateColorPlot(a, print.n.hz = TRUE)

}
```

---

amador *SSURGO Data Associated with the Amador Soil Series*

Description

SSURGO Data Associated with the Amador Soil Series

Usage

```
data(amador)
```

Format

A subset of data taken from the "component" table of SSURGO

`mukey` map unit key

`compname` component name

`comppct_r` component percentage

Source

USDA-NRCS SSURGO Database

`aspect.plot` *Plot Aspect Data*

Description

Plot a graphical summary of multiple aspect measurements on a circular diagram.

Usage

```
aspect.plot(p, q=c(0.05, 0.5, 0.95), p.bins = 60, p.bw = 30, stack=TRUE,
p.axis = seq(0, 350, by = 10), plot.title = NULL,
line.col='RoyalBlue', line.lwd=1, line.lty=2,
arrow.col=line.col, arrow.lwd=1, arrow.lty=1,
arrow.length=0.15,
...)
```

Arguments

`p`	a vector of aspect angles in degrees, measured clock-wise from North
`q`	a vector of desired quantiles
`p.bins`	number of bins to use for circular histogram
`p.bw`	bandwidth used for circular density estimation
`stack`	TRUE/FALSE, should the individual points be stacked into `p.bins` number of bins and plotted
`p.axis`	a sequence of integers (degrees) describing the circular axis
`plot.title`	an informative title
`line.col`	density line color
`line.lwd`	density line width
`line.lty`	density line line style

arrow.col	arrow color
arrow.lwd	arrow line width
arrow.lty	arrow line style
arrow.length	arrow head length
...	further arguments passed to plot.circular

Details

Spread and central tendency are depicted with a combination of circular histogram and kernel density estimate. The circular mean, and relative confidence in that mean are depicted with an arrow: longer arrow lengths correspond to greater confidence in the mean.

Note

Manual adjustment of p.bw may be required in order to get an optimal circular density plot. This function requires the package circular, version 0.4-7 or later.

Author(s)

D.E. Beaudette

Examples

```
# simulate some data
p.narrow <- runif(n=25, min=215, max=280)
p.wide <- runif(n=25, min=0, max=270)

# set figure margins to 0, 2-column plot
par(mar=c(0,0,0,0), mfcol=c(1,2))

# plot
aspect.plot(p.narrow, p.bw=10, plot.title='Soil A', pch=21, col='black', bg='RoyalBlue')
aspect.plot(p.wide, p.bw=10, plot.title='Soil B', pch=21, col='black', bg='RoyalBlue')
```

CDEC.snow.courses *CDEC Snow Course List*

Description

The CDEC snow course list, updated September 2019

Usage

```
data(CDEC.snow.courses)
```

Format

A data frame with 259 observations on the following 9 variables.

`course_number` course number

`name` connotative course label

`id` course ID

`elev_feet` course elevation in feet

`latitude` latitude

`longitude` longitude

`april.1.Avg.inches` average inches of snow as of April 1st

`agency` responsible agency

`watershed` watershed label

Source

Data were scraped from http://cdec.water.ca.gov/misc/SnowCourses.html, 2019.

Examples

```
data(CDEC.snow.courses)
head(CDEC.snow.courses)
```

`CDECquery` *Get water-related data (California only) from the CDEC website.*

Description

Get water-related data (California only) from the CDEC website.

Usage

```
CDECquery(id, sensor, interval = "D", start, end)
```

Arguments

`id`	station ID (e.g. 'spw'), single value or vector of station IDs, see details
`sensor`	the sensor ID, single value or vector of sensor numbers, see details
`interval`	character, 'D' for daily, 'H' for hourly, 'M' for monthly, 'E' for event: see Details.
`start`	starting date, in the format 'YYYY-MM-DD'
`end`	ending date, in the format 'YYYY-MM-DD'

Details

1. Station IDs can be found here: http://cdec.water.ca.gov/staInfo.html

2a. Sensor IDs can be found using this URL: http://cdec.water.ca.gov/dynamicapp/staMeta?station_id=, followed by the station ID.

2b. Sensor details can be accessed using CDEC_StationInfo with the station ID.

3. Resevoir capacities can be found here: http://cdec.water.ca.gov/misc/resinfo.html

4. A new interactive map of CDEC stations can be found here: http://cdec.water.ca.gov

Sensors that report data on an interval other than monthly ('M'), daily ('D'), or hourly ('H') can be queried with an 'event' interval ('E'). Soil moisture and temperature sensors are an example of this type of reporting. See examples below.

Value

a data.frame object with the following fields: 'datetime', 'year', 'month', 'value'.

Author(s)

D.E. Beaudette

References

http://cdec.water.ca.gov/queryCSV.html

See Also

CDECsnowQuery, CDEC_StationInfo

Examples

```
if(requireNamespace("curl") &
   curl::has_internet() &
   require(latticeExtra) &
   require(plyr) &
   require(e1071)) {

  # get daily resevoir storage (ac. ft) from Pinecrest, New Melones and Lyons resevoirs
 pinecrest <- CDECquery(id='swb', sensor=15, interval='D', start='2012-09-01', end='2015-01-01')
 new.melones <- CDECquery(id='nml', sensor=15, interval='D', start='2012-09-01', end='2015-01-01')
 lyons <- CDECquery(id='lys', sensor=15, interval='D', start='2012-09-01', end='2015-01-01')

  # compute storage capacity
 pinecrest$capacity <- pinecrest$value / 18312 * 100
 new.melones$capacity <- new.melones$value / 2400000 * 100
 lyons$capacity <- lyons$value / 6228 * 100

  # combine
 g <- make.groups(new.melones, lyons, pinecrest)
```

```
# resonable date scale
r <- range(g$datetime)
s.r <- seq(from=r[1], to=r[2], by='1 month')

# better colors
tps <- list(superpose.line=list(lwd=2, col=brewer.pal(n=3, name='Set1')))

# plot
xyplot(capacity ~ datetime, groups=which, data=g, type='l',
       xlab='', ylab='Capacity (%)', ylim=c(-5, 105),
       scales=list(x=list(at=s.r, labels=format(s.r, "%b\n%Y"))),
       auto.key=list(columns=3, lines=TRUE, points=FALSE),
       par.settings=tps,
       panel=function(...) {
         panel.abline(h=seq(0, 100, by=10), col='grey')
         panel.abline(v=s.r, col='grey')
         panel.xyplot(...)
       })


##
# New Melones monthly data, retrieve as far back in time as possible
new.melones.monthly <- CDECquery(id='nml', sensor=15, interval='M',
                                 start='1900-01-01', end='2015-01-01')

# convert to pct. capacity
new.melones.monthly$capacity <- new.melones.monthly$value / 2400000 * 100


# make a nice color ramp function
cols <- colorRampPalette(brewer.pal(9, 'Spectral'),
                         space='Lab', interpolate='spline')

# plot, each pixel is colored by the total precip by year/month
levelplot(capacity ~ year * month, data=new.melones.monthly, col.regions=cols, xlab='',
        ylab='', scales=list(x=list(tick.number=20)), main='New Melones Capacity (%)')


##
# get daily precip totals from Stan Powerhouse
x <- CDECquery(id='spw', sensor=45, interval='D', start='1900-01-01', end='2019-01-01')

# compute total precip by year/month
a <- ddply(x, c('year', 'month'), summarize, s=sum(value, na.rm=TRUE))

# convert monthly precipitation values into Z-scores by month
a.scaled <- ddply(a, 'month', summarize, year=year, scaled.ppt=scale(s))

# make a nice color ramp function, scaled by the skewness of the underlying distribution
cols <- colorRampPalette(brewer.pal(9, 'Spectral'),
space='Lab', interpolate='spline', bias=skewness(a.scaled$scaled.ppt, na.rm=TRUE))
```

```
  # plot, each pixel is colored by the total precip by year/month
  levelplot(scaled.ppt ~ year * month, data=a.scaled, col.regions=cols, xlab='',
            ylab='', scales=list(x=list(tick.number=10)),
            main='Monthly Total Precipitation (as z-score) SPW')


  ##
  # get pre-aggregated monthly data from Sonora RS
  x <- CDECquery(id='sor', sensor=2, interval='M', start='1900-01-01', end='2019-01-01')

  # make a nice color ramp function, scaled by the skewness of the underlying distribution
  cols <- colorRampPalette(brewer.pal(9, 'Spectral'), space='Lab',
  interpolate='spline', bias=skewness(x$value, na.rm=TRUE))

  # plot
  levelplot(value ~ year * month, data=x, col.regions=cols, xlab='',
  ylab='', scales=list(x=list(tick.number=20)),
  main='Monthly Total Precipitation (inches) SOR')


  ### query an 'event' type sensor
  # Bryte test site (BYT)
  # single request: air temperature and soil temperature at depth 1 (25cm)
  # measurement interval is 20 minutes
  x <- CDECquery('BYT', c(4, 194), 'E', '2016-01-01', '2017-01-01')

  # data are in long format, check number of records for each sensor
  table(x$sensor_type)

  # plot grouped data
  xyplot(value ~ datetime, groups=sensor_type, data=g, type=c('g', 'l'),
         auto.key=list(columns=2, points=FALSE, lines=TRUE))

}
```

CDECsnowQuery *Get snow survey data (California only) from the CDEC website.*

Description

Get snow survey data (California only) from the CDEC website.

Usage

```
CDECsnowQuery(course, start_yr, end_yr)
```

Arguments

course	integer, course number (e.g. 129)
start_yr	integer, the starting year (e.g. 2010)
end_yr	integer, the ending year (e.g. 2013)

Details

This function downloads data from the CDEC website, therefore an internet connection is required. The 'SWE' column contains adjusted SWE if available ('Adjusted' column), otherwise the reported SWE is used ('Water' column).

Value

a data.frame object, see examples

Note

Snow course locations, ID numbers, and other information can be found here: http://cdec.water.ca.gov/misc/SnowCourses.html

Author(s)

D.E. Beaudette

References

http://cdec.water.ca.gov/cgi-progs/snowQuery

Examples

```
if(requireNamespace("curl") &
   curl::has_internet()
   ) {

  # get data for course numbe 129
  x <- CDECsnowQuery(course=129, start_yr=2010, end_yr=2011)

}
```

CDEC_StationInfo *CDEC Sensor Details (by Station)*

Description

Query CDEC Website for Sensor Details

Usage

```
CDEC_StationInfo(s)
```

Arguments

s	a CDEC station ID (e.g. 'HHM')

Details

This function requires the 'rvest' package

Value

a 'list' object containing site metadata, sensor metadata, and possibly comments about the site

Author(s)

D.E. Beaudette

See Also

CDECquery

Examples

```
if(requireNamespace("curl") &
   curl::has_internet()
   ) {

CDEC_StationInfo('HHM')

}
```

`component.adj.matrix` *Create an adjacency matrix from a data.frame of component data*

Description

Create an adjacency matrix from SSURGO component data

Usage

```
component.adj.matrix(d, mu='mukey', co='compname', wt='comppct_r',
method='community.matrix', standardization='max', metric='jaccard',
rm.orphans=TRUE, similarity=TRUE, return.comm.matrix=FALSE)
```

Arguments

`d`	a `data.frame`, typically of SSURGO data
`mu`	name of the column containing the map unit ID (typically 'mukey')
`co`	name of the column containing the component ID (typically 'compname')
`wt`	name of the column containing the component weight percent (typically 'comppct_r')
`method`	one of either: 'community.matrix', or 'occurrence'; see details
`standardization`	community matrix standardization method, passed to `decostand`
`metric`	community matrix dissimilarity metric, passed to `vegdist`
`rm.orphans`	logical, should map units with a single component be ommitted? (typically yes)
`similarity`	logical, return a similarity matrix? (if FALSE, a distance matrix is returned)
`return.comm.matrix`	logical, return pseudo-community matrix? (if TRUE no adjacency matrix is created)

Details

Pending...

Value

a similatiy matrix / ajacency matrix suitable for use with `igraph` functions or anything else that can accomodate a _similarity_ matrix.

Author(s)

D.E. Beaudette

Examples

```
# load sample data set
data(amador)

# convert into adjacency matrix
m <- component.adj.matrix(amador)

# plot network diagram, with Amador soil highlighted
plotSoilRelationGraph(m, s='amador')
```

`constantDensitySampling` *Constant Density Sampling*

Description

Perform sampling at a constant density over all polygons within a SpatialPolygonsDataFrame object.

Usage

```
constantDensitySampling(x, polygon.id='pID', parallel=FALSE, cores=NULL,
n.pts.per.ac=1, min.samples=5, sampling.type='regular', iterations=10)
```

Arguments

`x`	a `SpatialPolygonsDataFrame` object in a projected CRS with units of meters
`polygon.id`	name of attribute in `x` that contains a unique ID for each polygon
`parallel`	invoke parallel back-end
`cores`	number of CPU cores to use for parallel operation
`n.pts.per.ac`	requested sampling density in points per acre (results will be close)
`min.samples`	minimum requested number of samples per polygon
`sampling.type`	sampling type, see `spsample`
`iterations`	number of tries that `spsample` will attempt

Value

a `SpatialPointsDataFrame` object

Note

This function expects that `x` has coordinates associated with a projected CRS and units of meters.

Author(s)

D.E. Beaudette

See Also

sample.by.poly

---

diagnosticPropertyPlot

Diagnostic Property Plot

Description

Generate a graphical description of the presence/absence of soil diagnostic properties.

Usage

```
diagnosticPropertyPlot(f, v, k, grid.label='pedon_id',
dend.label='pedon_id', sort.vars=TRUE)
diagnosticPropertyPlot2(f, v, k, grid.label='pedon_id', sort.vars=TRUE)
```

Arguments

f	a SoilProfileCollection object
v	a character vector of site-level attribute names that are boolean (e.g. TRUE/FALSE) data
k	an integer, number of groups to highlight
grid.label	the name of a site-level attribute (usually unique) annotating the y-axis of the grid
dend.label	the name of a site-level attribute (usually unique) annotating dendrogram terminal leaves
sort.vars	sort variables according to natural clustering (TRUE), or use supplied ordering in v (FALSE)

Details

This function attempts to display several pieces of information within a single figure. First, soil profiles are sorted according to the presence/absence of diagnostic features named in v. Second, these diagnostic features are sorted according to their distribution among soil profiles. Third, a binary grid is established with row-ordering of profiles based on step 1 and column-ordering based on step 2. Blue cells represent the presence of a diagnostic feature. Soils with similar diagnostic features should 'clump' together. See examples below.

Value

a list is silently returned by this function, containing:

rd a data.frame containing IDs and grouping code

profile.order a vector containing the order of soil profiles (row-order in figure), according to diagnostic property values

var.order a vector containing the order of variables (column-order in figure), according to their distribution among profiles

Author(s)

D.E. Beaudette and J.M. Skovlin

See Also

multinominal2logical

Examples

```
if(require(aqp) &
  require(soilDB) &
  require(latticeExtra)
  ) {

  # sample data, an SPC
data(gopheridge, package='soilDB')

# get depth class
sdc <- getSoilDepthClass(gopheridge)
site(gopheridge) <- sdc

# diagnostic properties to consider, no need to convert to factors
v <- c('lithic.contact', 'paralithic.contact', 'argillic.horizon',
'cambic.horizon', 'ochric.epipedon', 'mollic.epipedon', 'very.shallow',
'shallow', 'mod.deep', 'deep', 'very.deep')

# base graphics
x <- diagnosticPropertyPlot(gopheridge, v, k=5)

# lattice graphics
x <- diagnosticPropertyPlot2(gopheridge, v, k=3)

# check output
str(x)

}
```

---

dist.along.grad *Compute Euclidean distance along a gradient.*

Description

This function computes Euclidean distance along points aligned to a given gradient (e.g. elevation).

Usage

```
dist.along.grad(coords, var, grad.order, grad.scaled.min, grad.scaled.max)
```

Arguments

`coords`	a matrix of x and y coordinates in some projected coordinate system
`var`	a vector of the same length as `coords`, describing the gradient of interest
`grad.order`	vector of integers that define ordering of coordinates along gradient
`grad.scaled.min`	min value of rescaled gradient values
`grad.scaled.max`	max value of rescaled gradient values

Details

This function is primarily intended for use within `plotTransect`.

Value

A `data.frame` object:

scaled.grad scaled gradient values

scaled.distance cumulative distance, scaled to the interval of `0.5,nrow(coords) + 0.5`

distance cumulative distance computed along gradient, e.g. transect distance

variable sorted gradient values

x x coordinates, ordered by gradient values

y y coordinate, ordered by gradient values

grad.order a vector index describing the sort order defined by gradient values

Note

This function is very much a work in progress, ideas welcome.

Author(s)

D.E. Beaudette

See Also

`plotTransect`

`dueling.dendrograms` *Dueling Dendrograms*

Description

Graphically compare two related dendrograms

Usage

```
dueling.dendrograms(p.1, p.2, lab.1 = "D1",
lab.2 = "D2", cex.nodelabels=0.75, arrow.length=0.05)
```

Arguments

`p.1`	left-hand phylo-class dendrogram
`p.2`	right-hand phylo-class dendrogram
`lab.1`	left-hand title
`lab.2`	right-hand title
`cex.nodelabels`	character expansion size for node labels
`arrow.length`	arrow head size

Details

Connector arrows are used to link nodes from the left-hand dendrogram to the right-hand dendrogram.

Author(s)

D. E. Beaudette

Examples

```
if(require(aqp) &
  require(cluster) &
  require(latticeExtra) &
  require(ape)
) {

  # load sample dataset from aqp package
data(sp3)

# promote to SoilProfileCollection
depths(sp3) <- id ~ top + bottom

# compute dissimilarity using different sets of variables
# note that these are rescaled to the interval [0,1]
d.1 <- profile_compare(sp3, vars=c('clay', 'cec'), k=0, max_d=100, rescale.result=TRUE)
```

```
d.2 <- profile_compare(sp3, vars=c('clay', 'L'), k=0, max_d=100, rescale.result=TRUE)

# cluster via divisive hierarchical algorithm
# convert to 'phylo' class
p.1 <- as.phylo(as.hclust(diana(d.1)))
p.2 <- as.phylo(as.hclust(diana(d.2)))

# graphically compare two dendrograms
dueling.dendrograms(p.1, p.2, lab.1='clay and CEC', lab.2='clay and L')

# graphically check the results of ladderize() from ape package
dueling.dendrograms(p.1, ladderize(p.1), lab.1='standard', lab.2='ladderized')

# sanity-check: compare something to itself
dueling.dendrograms(p.1, p.1, lab.1='same', lab.2='same')

# graphically compare diana() to agnes() using d.2
dueling.dendrograms(as.phylo(as.hclust(diana(d.2))),
as.phylo(as.hclust(agnes(d.2))), lab.1='diana', lab.2='agnes')
}
```

FFD *Frost-Free Day Evaluation*

Description

Evaluation frost-free days and related metrics from daily climate records.

Usage

```
FFD(d, returnDailyPr = TRUE, minDays = 165, ...)
FFDplot(s, sub.title = NULL)
```

Arguments

d	data.frame with columns 'datetime' 'year', and 'value'
returnDailyPr	optionally return list with daily summaries
minDays	min number of days / springlfall required for a reasonable estimate of FFD
...	further arguments passed to frostFreePeriod
s	object returned by FFD
sub.title	override default subtitle

Details

FFD tutorial

Value

a `list` with the following elements:

`summary`	FFD summary statistics as a `data.frame`
`fm`	frost matrix
`Pr.frost`	Pr(frost|day): daily probability of frost

Note

This is a work in progress.

Author(s)

D.E. Beaudette

`formatPLSS` *formatPLSS*

Description

Format PLSS information into a coded format that can be digested by PLSS web service.

Usage

```
formatPLSS(p, type='SN')
```

Arguments

`p`	data.frame with chunks of PLSS coordinates
`type`	an option to format protracted blocks 'PB', unprotracted blocks 'UP', or standard section number 'SN' (default).

Details

This function is typically accessed as a helper function to prepare data for use within `PLSS2LL` function.

Value

A vector of PLSS codes.

Note

This function expects that the `Polygon` object has coordinates associated with a projected CRS– e.g. units of meters.

Author(s)

D.E. Beaudette, Jay Skovlin

See Also

PLSS2LL

Examples

```
# create some data
d <- data.frame(
id=1:3,
qq=c('SW', 'SW', 'SE'),
q=c('NE', 'NW', 'SE'),
s=c(17, 32, 30),
t=c('T36N', 'T35N', 'T35N'),
r=c('R29W', 'R28W', 'R28W'),
type='SN',
m='MT20', stringsAsFactors = FALSE)

# add column names
names(d) <- c('id', 'qq', 'q', 's', 't', 'r', 'type', 'm')

# generate formatted PLSS codes
formatPLSS(d, type='SN')
```

---

generateLineHash *Generate a unique ID for line segments*

Description

Generate a unique ID for a line segment, based on the non-cryptographic murmur32 hash.

Usage

```
generateLineHash(x, precision=-1, algo='murmur32')
```

Arguments

x	a SpatialLinesDataFrame object, with 1 line segment per feature (e.g. simple features)
precision	digits are rounded to this many places to the right (negative) or left (positive) of the decimal place
algo	hash function algorithm

Details

The input `SpatialLinesDataFrame` object must NOT contain multi-part features. The precision specified should be tailored to the coordinate system in use and the snapping tolerance used to create join decision line segments. A `precision` of 4 is reasonable for geographic coordinates (snapping tolerance of 0.0001 degrees or ~ 10 meters). A `precision` of -1 (snapping tolerance of 10 meters) is reasonable for projected coordinate systems with units in meters.

Value

A vector of unique IDs created from the hash of line segment start and end vertex coordinates. Unique IDs are returned in the order of records of `x` and can therefore be saved into a new column of the associated attribute table.

Note

An error is issued if any non-unique IDs are generated. This could be casued by using coordiantes that do not contain enough precision for unique hashing.

Author(s)

D.E. Beaudette

---

geomorphBySoilSeries-SSURGO

Geomorphic Position Probability via SDA

Description

Hillslope position probability estimates from the SDA query service (SSURGO)

Usage

```
hillslopeProbability(s, replaceNA=TRUE)
surfaceShapeProbability(s, replaceNA=TRUE)
geomPosHillProbability(s, replaceNA=TRUE)
geomPosMountainProbability(s, replaceNA=TRUE)
```

Arguments

`s`	a character vector of soil series names, automatically normalized to upper case
`replaceNA`	boolean: should missing classes be converted to probabilities of 0?

Details

These functions send a query to the SDA webservice. Further information on the SDA webservice and query examples can be found at `http://sdmdataaccess.nrcs.usda.gov/QueryHelp.aspx`

Value

A `data.frame` object with rows representing soil series, and columns representing probability estimates of that series occuring at specificed geomorphic positions or associated with a surface shape.

Note

Probability values are computed from SSURGO data.

Author(s)

D.E. Beaudette

Examples

```
if(requireNamespace("curl") &
   curl::has_internet() &
   require(soilDB)) {

  # soil series of interest
s <- c('amador', 'peters', 'pentz', 'inks', 'auburn', 'dunstone', 'argonaut')

# generate hillslope probability table
hillslopeProbability(s)

# generate surface 2D shape probability table
surfaceShapeProbability(s)

}
```

HHM *Highland Meadows*

Description

11 years of climate data from the Highland Meadows weather station, as maintained by CA DWR.

Usage

```
data("HHM")
```

Format

A data frame with 3469 observations on the following 12 variables.

`station_id` a character vector

`dur_code` a character vector

`sensor_num` a numeric vector

`sensor_type` a character vector

`value` a numeric vector

`flag` a character vector

`units` a character vector

`datetime` a POSIXct

`year` a numeric vector

`month` a factor with levels `January` `February` `March` `April` `May` `June` `July` `August` `September` `October` `November` `December`

`water_year` a numeric vector

`water_day` a numeric vector

`joinAdjacency` *Join Document Adjacency*

Description

Convert a set of line segment "join decisions" into a weighted adjacency matrix describing which map unit symbols touch.

Usage

```
joinAdjacency(x, vars = c("l_musym", "r_musym"))
```

Arguments

`x`	a `SpatialLinesDataFrame` object, with 1 line segment per feature (e.g. simple features)
`vars`	a vector of two characters naming columns containing "left", and "right" map unit symbols

Value

A weighted adjacency matrix is returned, suitable for plotting directly with `plotSoilRelationGraph`.

Author(s)

D.E. Beaudette

See Also

`plotSoilRelationGraph`

LL2PLSS *PLSS2LL*

Description

Uses lattitude and longitude coordinates to return the PLSS section geometry from the BLM PLSS web service.

Usage

```
LL2PLSS(x, y)
```

Arguments

x	longitude coordinates
y	lattitude coordinates

Details

This function takes xy coordinates and returns the PLSS section geometry to the quarter-quarter section.

Value

`list` of of PLSS codes and coordinates.

Note

This function requires the following packages: `httr`, `jsonlite`, and `sp`.

Author(s)

D.E. Beaudette, Jay Skovlin

See Also

`PLSS2LL`, `formatPLSS`

Examples

```
if(requireNamespace("curl") &
   curl::has_internet() &
   require(sp)) {

  # create coordinates
x <- -115.3823
y <- 48.88228
```

```
# fetch PLSS geometry for these coordinates
p.plss <- LL2PLSS(x, y)

# plot geometry
plot(p.plss$geom)

}
```

`monthlyWB` *Monthly Water Balances*

Description

Monthly water balances and visualization by "leaky bucket" model, provided by the hydromad package.

Usage

```
monthlyWB(AWC, PPT, PET, S_init=AWC, starting_month=1, rep=1, keep_last=FALSE)

plotWB(AWC, WB, fig.title='', sw.col='#377EB8',
surplus.col='#4DAF4A', et.col='#E41A1C', deficit.col='#FF7F00')
```

Arguments

`AWC`	available water-holding capacity (mm)
`PPT`	time-series of monthly PPT (mm), calendar year ordering
`PET`	time-series of monthly PET (mm), calendar year ordering
`S_init`	intitial soil water storage (mm)
`starting_month`	starting month index, 1=January, 9=September
`rep`	number of cycles to run water balance
`keep_last`	keep only the last iteration of the water balance
`WB`	output from `monthlyWB`
`fig.title`	a title
`sw.col`	color for soil water
`surplus.col`	color for surplus water
`et.col`	color for ET
`deficit.col`	color for deficit

Details

This function depends on the hydromad package.

Value

a `data.frame` with the following elements:

`PPT`	monthly PPT values
`PPT`	monthly PET values
`U`	monthly U values
`S`	monthly S values
`ET`	monthly ET values
`D`	monthly D values
`month`	month number
`mo`	month label

Note

This is a work in progress.

Author(s)

D.E. Beaudette

`multinominal2logical` *Convert multinominal to logical matrix*

Description

Convert a single multinominal, site-level attribute from a SoilProfileCollection into a matrix of corresponding logical values. The result contains IDs from the SoilProfileCollection and can easily be joined to the original site-level data.

Usage

```
multinominal2logical(x, v)
```

Arguments

`x`	a `SoilProfileCollection` object
`v`	the name of a site-level attribute that is a factor with more than 2 levels

Value

A `data.frame` with IDs in the first column, and as many columns of logical vectors as there were levels in `v`. See examples.

Author(s)

D.E. Beaudette

See Also

diagnosticPropertyPlot

Examples

```
if(require(soilDB) &
  require(aqp) &
  require(latticeExtra)) {


# sample data, an SPC
data(loafercreek, package='soilDB')

# convert to logical matrix
hp <- multinominal2logical(loafercreek, 'hillslopeprof')

# join-in to site data
site(loafercreek) <- hp

# variable names
v <- c('lithic.contact', 'paralithic.contact',
'argillic.horizon', 'Toeslope', 'Footslope',
'Backslope', 'Shoulder', 'Summit')

# visualize with some other diagnostic features
x <- diagnosticPropertyPlot(loafercreek, v, k=5,
grid.label='bedrckkind', dend.label='pedon_id')

}
```

PCP_plot *Percentiles of Cumulative Precipitation*

Description

Generate a plot representing percentiles of cumulative precipitation, given a historic record, and criteria for selecting a year of data for comparison.

Usage

```
PCP_plot(x, this.year, method = "exemplar",
q.color = "RoyalBlue", c.color = "firebrick", ...)
```

Arguments

x	result from CDECquery() for now, will need to generalize to other sources
this.year	a single water year, e.g. 2020
method	'exemplar' or 'daily', currently 'exemplar' is the only method available
q.color	color of percentiles cumulative precipitation
c.color	color of selected year
...	addtional arguments to plot()

Details

This is very much a work in progress. Further examples at http://ncss-tech.github.io/AQP/sharpshootR/CDEC.html

Value

Currently nothing is returned.

Author(s)

D.E. Beaudette

See Also

waterDayYear

Examples

```
if(requireNamespace("curl") &
   curl::has_internet()
   ) {

  s <- 'SPW'
# get metadata
s.info <- CDEC_StationInfo(s)
# format title for cumulative PPT
title.text <- sprintf("%s [%s]", s.info$site.meta$Name, s)

# get data
x <- CDECquery(id=s, sensor=45, interval='D', start='2000-01-01', end='2030-01-01')

## NOTE: requires sharpshootR >= 1.4.02
# plot
par(mar=c(4.5, 4.5, 2.5, 1.5))
PCP_plot(x[1:(nrow(x)-60), ], ylab='Cumulative PPT (inches)', main=title.text, this.year = 2020)

}
```

`percentileDemo` *Demonstration of Percentiles vs. Mean / SD*

Description

This function can be used to graphically demonstrate the relationship between distribution shape, an idealized normal distribution (based on sample mean and sd) shape, and measures of central tendency / spread.

Usage

```
percentileDemo(x, labels.signif = 3, pctile.color = "RoyalBlue",
mean.color = "Orange", range.color = "DarkRed",
hist.breaks = 30, boxp = FALSE, ...)
```

Arguments

`x`	vector of values to summarize
`labels.signif`	integer, number of significant digits to be used in figure annotation
`pctile.color`	color used to demonstrate range from 10th to 90th percentiles
`mean.color`	color used to specify mean +/- 2SD
`range.color`	color used to specify data range
`hist.breaks`	integer, number of suggested breaks to `hist`
`boxp`	logical, add a box and whisker plot?
`...`	further arguments to `plot`

Value

A 1-row matrix of summary stats is invisibly returned.

Note

This function is mainly for educational purposes.

Author(s)

D.E. Beaudette

References

https://ncss-tech.github.io/soil-range-in-characteristics/why-percentiles.html

Examples

```
x <- rnorm(100)
percentileDemo(x)

x <- rlnorm(100)
percentileDemo(x)
```

`plotAvailWater` *Visual Demontration of Available Soil Water*

Description

Generate a simplistic diagram of the various fractions of water held within soil pore-space.

Usage

```
plotAvailWater(x, width = 0.25, cols = c(grey(0.5),
"DarkGreen", "LightBlue", "RoyalBlue"), name.cex = 0.8,
annotate=TRUE)
```

Arguments

`x`	a `data.frame` containing sample names and water retention data, see examples below
`width`	vertical width of each bar graph
`cols`	a vector of colors used to symbolize 'solid phase', 'unavailable water', 'available water', and 'gravitational water'
`name.cex`	character scaling of horizon names, printed on left-hand side of figure
`annotate`	logical, annotate AWC

Author(s)

D.E. Beaudette

Examples

```
# demonstration
s <- data.frame(
name=c('loamy sand', 'sandy loam', 'silt loam', 'clay loam'),
pwp=c(0.05, 0.1, 0.18, 0.2),
fc=c(0.1, 0.2, 0.38, 0.35),
sat=c(0.25, 0.3, 0.45, 0.4))
s$solid <- with(s, 1-sat)

par(mar=c(5, 6, 0.5, 0.5))
plotAvailWater(s, name.cex=1.25)
```

```
# use some real data from SSURGO


if(requireNamespace("curl") &
   curl::has_internet() &
   require(soilDB)) {

  q <- "SELECT hzdept_r as hztop, hzdepb_r as hzbottom,
hzname as name, wsatiated_r/100.0 as sat,
wthirdbar_r/100.0 as fc, wfifteenbar_r/100.0 as pwp, awc_r as awc
FROM chorizon
WHERE cokey IN (SELECT cokey from component where compname = 'dunstone')
AND wsatiated_r IS NOT NULL
ORDER BY cokey, hzdept_r ASC;"

x <- SDA_query(q)
x <- unique(x)
x <- x[order(x$name), ]
x$solid <- with(x, 1-sat)

par(mar=c(5, 5, 0.5, 0.5))
plotAvailWater(x)

}
```

`plotProfileDendrogram` *Plot soil profiles below a dendrogram*

Description

Plot soil profiles below a dendrogram

Usage

```
plotProfileDendrogram(x, clust, scaling.factor=0.01, width=0.1, y.offset=0.1,
dend.y.scale= max(clust$height * 2, na.rm=TRUE) ,
dend.color=par('fg'), dend.width=1, debug=FALSE, ...)
```

Arguments

`x`	a `SoilProfileCollection` object
`clust`	a hierachical clustering object generated by `hclust()`, `cluster::agnes()`, or `cluster::diana()`
`scaling.factor`	vertical scaling of the profile heights (may have to tinker with this)
`width`	scaling of profile widths

y.offset	vertical offset for top of profiles
dend.y.scale	extent of y-axis (may have to tinker with this)
dend.color	dendrogram line color
dend.width	dendrogram line width
debug	optionally print debugging data
...	additional arguments to plotSPC

Details

This function places soil profile sketches below a dendrogram.

Note

You may have to tinker with some of the arguments to get optimal arrangement and scaling of soil profiles.

Author(s)

D.E. Beaudette

See Also

plotSPC

---

plotSoilRelationChordGraph

Vizualize Soil Relationships via Chord Diagram.

Description

Plot a soil relationship diagram using a chord diagram.

Usage

```
plotSoilRelationChordGraph(m, s, mult = 2, base.color = "grey",
highlight.colors = c("RoyalBlue", "DarkOrange", "DarkGreen"),
add.legend = TRUE, ...)
```

Arguments

m	an adjacency matrix, no NA allowed
s	soil of interest, must exist in the column or row names of m
mult	multiplier used to re-scale data in m associated with s
base.color	color for all soils other than s and 1st and 2nd most commonly co-occuring
highlight.colors	vector of 3 colors: soil of interest, 1st most common, 2nd most common
add.legend	logical, add a legend
...	additional arguments passed to circlize::chordDiagramFromMatrix

Details

This function is experimental. Documentation pending. See http://jokergoo.github.io/circlize/ for ideas.

Author(s)

D.E. Beaudette

References

https://github.com/jokergoo/circlize

See Also

plotSoilRelationGraph

Examples

```
data(amador)
m <- component.adj.matrix(amador)
plotSoilRelationChordGraph(m, 'amador')
```

---

plotSoilRelationGraph *Plot a component relation graph*

Description

Plot a component relation graph based on an adjacency or similarity matrix.

Usage

```
plotSoilRelationGraph(m, s='', plot.style='network', graph.mode='upper',
spanning.tree=NULL, del.edges=NULL, vertex.scaling.method='degree',
vertex.scaling.factor=2, edge.scaling.factor=1,
vertex.alpha=0.65, edge.transparency=1,
edge.col=grey(0.5), edge.highlight.col='royalblue', g.layout=layout_with_fr,
vertex.label.color='black', ...)
```

Arguments

m	adjacency matrix
s	central component; an empty character string is interpreted as no central component
plot.style	plot style ('network', or 'dendrogram'), or 'none' for no graphical output
graph.mode	interpretation of adjacency matrix: 'upper' or 'directed', see details

`spanning.tree` plot the minimim or maximum spaning tree ('min', 'max'), or, max spanning tree plus edges with weight greater than the n-th quantile specified in 'spanning.tree'. See details and examples.

`del.edges` optionally delete edges with weights less than the specified quantile (0-1)

`vertex.scaling.method` 'degree' (default) or 'distance', see details

`vertex.scaling.factor` scaling factor applied to vertex size

`edge.scaling.factor` optional scaling factor applied to edge width

`vertex.alpha` optional transparency setting for vertices (0-1)

`edge.transparency` optional transparency setting for edges (0-1)

`edge.col` edge color, applied to all edges

`edge.highlight.col` edge color applied to all edges connecting to component named in `s`

`g.layout` an igraph layout function, defaults to `layout_with_fr`

`vertex.label.color` vertex label color

`...` further arguments passed to plotting function

Details

Vertex size is based on a normalized index of connectivity: "degree" size = sqrt(degree(g)/max(degree(g))) * scaling.factor, or "distance" size = sqrt(distance(V->s)/max(distance(V->s))) * scaling.factor, where distance(V->s) is the distance from all nodes to the named series, `s`.

Edge width can be optionally scaled by edge weight by specifying an `edge.scaling.factor` value. The maximum spanning tree represents a sub-graph where the sum of edge weights are maximized. The mimimum spanning tree represents a sub-graph where the sum of edge weights are minimized. The maximum spanning tree is likely a more useful simplification of the full graph, in which only the strongest relationships (e.g. most common co-occurrences) are preserved.

The maximum spanning tree + edges with weights > n-th quantile is an experimental hybrid. The 'backbone' of the graph is created by the maximum spanning tree, and augmented by 'strong' auxillary edges– defined by a value between 0 and 1.

The `graph.mode` argument is passed to `igraph::graph_from_adjacency_matrix()` and determines how vertex relationships are coded in the adjacency matrix `m`. Typically, the default value of 'upper' (the upper triangle of `m` contains adjacency information) is the desired mode. If `m` contains directional information, set `graph.mode` to 'directed'. This has the side-effect of altering the default community detection algorithm from `igraph::cluster_fast_greedy` to `igraph::cluster_walktrap`.

Value

an igraph 'graph' object is invisibly returned

Note

This function is a work in progress, ideas welcome.

Author(s)

D.E. Beaudette

Examples

```
# load sample data set
data(amador)

# create weighted adjacency matrix (see ?component.adj.matrix for details)
m <- component.adj.matrix(amador)

# plot network diagram, with Amador soil highlighted
plotSoilRelationGraph(m, s='amador')

# dendrogram representation
plotSoilRelationGraph(m, s='amador', plot.style='dendrogram')

# compare methods
m.o <- component.adj.matrix(amador, method='occurrence')

par(mfcol=c(1,2))
plotSoilRelationGraph(m, s='amador', plot.style='dendrogram')
title('community matrix')
plotSoilRelationGraph(m.o, s='amador', plot.style='dendrogram')
title('occurence')

# investigate max spanning tree
plotSoilRelationGraph(m, spanning.tree='max')

# investigate max spanning tree + edges with weights > 75-th pctile
plotSoilRelationGraph(m, spanning.tree=0.75)



if(requireNamespace("curl") &
   curl::has_internet() &
   require(soilDB)) {

 # get similar data from soilweb, for the Pardee series
s <- 'pardee'
d <- siblings(s, component.data = TRUE)

# normalize component names
d$sib.data$compname <- tolower(d$sib.data$compname)

# keep only major components
d$sib.data <- subset(d$sib.data, subset=compkind == 'Series')
```

```
# build adj. matrix and plot
m <- component.adj.matrix(d$sib.data)
plotSoilRelationGraph(m, s=s, plot.style='dendrogram')

# alter plotting style, see ?plot.phylo
plotSoilRelationGraph(m, s=s, plot.style='dendrogram', type='fan')
plotSoilRelationGraph(m, s=s, plot.style='dendrogram', type='unrooted', use.edge.length=FALSE)

}
```

`plotTransect` *Plot a collection of Soil Profiles linked to their position along some gradient (e.g. transect).*

Description

Plot a collection of Soil Profiles linked to their position along some gradient (e.g. transect).

Usage

```
plotTransect(s, grad.var.name,
grad.var.order = order(site(s)[[grad.var.name]]),
transect.col = "RoyalBlue",
tick.number=7, y.offset = 100,
scaling.factor = 0.5,
distance.axis.title = "Distance Along Transect (km)",
crs = NULL, grad.axis.title = NULL, dist.scaling.factor=1000,
spacing='regular',
fix.relative.pos=list(thresh = 0.6, trace = TRUE, maxIter = 5000),
...)
```

Arguments

`s`	a `SoilProfileCollection` object
`grad.var.name`	the name of a site-level attribute containing gradient values
`grad.var.order`	optional vector of indices used to override sorting along `grad.var.name`
`transect.col`	color used to plot gradient (transect) values
`tick.number`	number of desired ticks and labels on the gradient axis
`y.offset`	vertical offset used to position profile sketches
`scaling.factor`	scaling factor used to adjust profile sketches
`distance.axis.title`	a tital for the along-transect distances
`crs`	an optional `CRS` object used to convert coordinates into a planar system

`grad.axis.title`
: a title for the gradient axis

`dist.scaling.factor`
: scaling factor applied to linear distance units, default is conversion from m to km

`spacing`
: regular (profiles aligned to an integer grid) or relative (relative distance along transect) spacing

`fix.relative.pos`
: FALSE to suppress, otherwise list of arguments to `aqp::fixOverlap`

`...`
: further arguments passed to `plotSPC`

Details

Depending on the nature of your SoilProfileCollection and associated gradient values, it may be necessary to tinker with figure margins, `y.offset` and `scaling.factor`.

Value

An invisibly-returned `data.frame` object:

scaled.grad scaled gradient values

scaled.distance cumulative distance, scaled to the interval of `0.5,nrow(coords) + 0.5`

distance cumulative distance computed along gradient, e.g. transect distance

variable sorted gradient values

x x coordinates, ordered by gradient values

y y coordinate, ordered by gradient values

grad.order a vector index describing the sort order defined by gradient values

Note

This function is very much a work in progress, ideas welcome!

Author(s)

D.E. Beaudette

Examples

```
if(require(aqp) &
require(sp) &
require(soilDB)
) {

# sample data
data("mineralKing", package = "soilDB")

# quick overview
```

```
par(mar=c(1,1,2,1))
groupedProfilePlot(mineralKing, groups='taxonname', print.id=FALSE)

# init coords and CRS
coordinates(mineralKing) <- ~ x_std + y_std
proj4string(mineralKing) <- '+proj=longlat +datum=NAD83'

# projected CRS, units of meters
crs.utm <- CRS('+proj=utm +zone=11 +datum=NAD83')

# adjust margins
par(mar=c(4.5,4,4,1))

# standard transect plot, profile sketches arranged along integer sequence
plotTransect(mineralKing, grad.var.name='elev_field', crs=crs.utm,
grad.axis.title='Elevation (m)', label='pedon_id', name='hzname')

# default behavior, attempt adjustments to prevent over-plot and preserve relative spacing
# use set.seed() to fix outcome
plotTransect(mineralKing, grad.var.name='elev_field', crs=crs.utm,
grad.axis.title='Elevation (m)', label='pedon_id', name='hzname', width=0.15, spacing = 'relative')

# attempt relative positioning based on scaled distances, no corrections for overlap
# profiles are clustered in space and therefore over-plot
plotTransect(mineralKing, grad.var.name='elev_field', crs=crs.utm,
grad.axis.title='Elevation (m)', label='pedon_id', name='hzname',
width=0.15, spacing = 'relative', fix.relative.pos = FALSE)

# customize arguments to aqp::fixOverlap()
plotTransect(mineralKing, grad.var.name='elev_field', crs=crs.utm,
grad.axis.title='Elevation (m)', label='pedon_id', name='hzname',
width=0.15, spacing = 'relative',
fix.relative.pos = list(maxIter=6000, trace=TRUE, adj=0.2, thresh=0.7))

plotTransect(mineralKing, grad.var.name='elev_field', crs=crs.utm,
grad.axis.title='Elevation (m)', label='pedon_id', name='hzname',
width=0.15, spacing = 'relative',
fix.relative.pos = list(maxIter=6000, trace=TRUE, adj=0.2, thresh=0.5))

}
```

PLSS2LL *PLSS2LL*

Description

Fetch lattitude and longitude centroid coordinates for coded PLSS information from the BLM PLSS web service.

Usage

```
PLSS2LL(p)
```

Arguments

p	dataframe with chunks of PLSS coordinates

Details

This function takes coded PLSS information and fetches the centroid lat/long coordinates down to the quarter-quarter section. The 'plssid' column is generated within a dataframe using the following `formatPLSS` function.

Value

A dataframe of PLSS codes and coordinates.

Note

This function expects that the dataframe will have a 'plssid' column generated by the `formatPLSS` function. Requires the following packages: `httr`, and `jsonlite`.

Author(s)

D.E. Beaudette, Jay Skovlin

See Also

`LL2PLSS`, `formatPLSS`

Examples

```
if(requireNamespace("curl") &
   curl::has_internet()
   ) {


# create some data
d <- data.frame(
id=1:3,
qq=c('SW', 'SW', 'SE'),
q=c('NE', 'NW', 'SE'),
s=c(17, 32, 30),
t=c('T36N', 'T35N', 'T35N'),
r=c('R29W', 'R28W', 'R28W'),
type='SN',
m='MT20', stringsAsFactors = FALSE)

# add column names
```

```
names(d) <- c('id', 'qq', 'q', 's', 't', 'r', 'type', 'm')

# generate formatted PLSS codes
d$plssid <- formatPLSS(d)

# fetch lat/long coordinates
PLSS2LL(d)

}
```

`polygonAdjacency` *Evaluate Spatial Adjacency of SpaitialPolygonsDataFrame Objects*

Description

This function utilizes the ‘spdep‘ and ‘igraph‘ packages to evaluate several measures of spatial connectivity.

Usage

```
polygonAdjacency(x, v='MUSYM', ...)
```

Arguments

`x`	a `SpatialPolygonsDataFrame` object
`v`	name of the field in the attribute table to use when searching for ‘common lines‘, see details
`...`	additional arguments passed to `spdep::poly2nb`

Details

Examples are presented in this tutorial.

Value

A `list` object containing:

commonLines An integer vector of feature IDs, that share a common boundary and attribute `v.commonLines`. Sometimes referred to as "common soil lines".

adjMat A weighted adjacency matrix

Author(s)

D.E. Beaudette

sample.by.poly *Sample a Polygon at Fixed Density*

Description

Generate sampling points within a SpatialPolygon object, according to a specified sampling density.

Usage

```
sample.by.poly(p, n.pts.per.ac=1, min.samples=5,
sampling.type='regular', iterations=10, p4s=NULL)
```

Arguments

p	a Polygon object, with coordinates in a projected CRS with units of meters
n.pts.per.ac	requested sampling density in points per acre (results will be close)
min.samples	minimum requested number of samples per polygon
sampling.type	sampling type, see spsample
iterations	number of tries that spsample will attempt
p4s	a qualified proj4string that will be assigned to sampling points

Details

This function is typically accessed via some kind of helper function such as constantDensitySampling.

Value

A SpatialPoints object.

Note

This function expects that the Polygon object has coordinates associated with a projected CRS– e.g. units of meters. Invalid geometries may cause errors or yield incorrect sample sizes.

Author(s)

D.E. Beaudette

See Also

spsample, constantDensitySampling

`sampleRasterStackByMU` *Sample a Raster Stack*

Description

Sample a raster stack by map unit polygons, at a constant density.

Usage

```
sampleRasterStackByMU(mu, mu.set, mu.col, raster.list, pts.per.acre,
p = c(0, 0.05, 0.25, 0.5, 0.75, 0.95, 1), progress = TRUE,
estimateEffectiveSampleSize=TRUE)
```

Arguments

`mu`	a `SpatialPolygonsDataFrame` object in a projected coordinate reference system (CRS)
`mu.set`	character vector of map unit labels to be sampled
`mu.col`	column name in attribute table containing map unit labels
`raster.list`	a `list` containing raster names and paths, see details below
`pts.per.acre`	target sampling density in 'points per acre'
`p`	percentiles for polygon area stats, e.g. (0.05, 0.25, 0.5, 0.75, 0.95)
`progress`	logical, print a progress bar while sampling?
`estimateEffectiveSampleSize`	estimate an effective sample size via Moran's I?

Details

This function is used by various NRCS reports that summarize or compare concepts defined by collections of polygons using raster data sampled from within each polygon, at a constant sampling density. Even though the function name includes "rasterSTack", this function doesn't actually operate on a 'stack' object as defined in the raster package. The collection of raster data defined in `raster.list` do not have to share a common coordinate reference system, grid spacing, or extent. Point samples generated from `mu` are automatically converted to the CRS of each raster before extracting values. The extent of each raster in `raster.list` must completely contain the extent of `mu`.

Value

A `list` containing:

'raster.samples' a `data.frame` containing samples from all rasters in the stack

'area.stats' a `data.frame` containing area statistics for all map units in the collection

'unsampled.ids' an index to rows in the original SPDF associated with polygons not sampled

'raster.summary' a `data.frame` containing information on sampled rasters

'Moran_I' a `data.frame` containing estimates Moran's I (index of spatial autocorrelation)

Author(s)

D.E. Beaudette

See Also

constantDensitySampling, sample.by.poly

---

site_photos_kml *site_photos_kml*

Description

Generates a KML file of site locations with associated site photos and a link to a pedon description report.

Usage

```
site_photos_kml(data,
filename='photos.kml', make.image.grid=FALSE,
file.source = c('local', 'relative')
)
```

Arguments

data	a dataframe
filename	full file path and name with .kml extension
make.image.grid	logical, include linked site images, default is FALSE
file.source	'local' sources the image files to a specific system path, 'relative' sources the image files to files folder that can be included and referenced within a .kmz file

Details

This function simplifies writing a kml file of site and/or sites with linked photos. Further documentation is provided in this tutorial.

Value

A KML file of of sites with embedded associated site photos.

Author(s)

Jay Skovlin, D.E. Beaudette

SoilTaxonomyDendrogram

Soil Taxonomy Dendrogram

Description

Plot a dendrogram based on the first 4 levels of Soil Taxonomy, with soil profiles hanging below. A dissimmilarity matrix is computed using Gower's distance metric for nominal-scale variables, based on order, sub order, great group, and subgroup level taxa. See the Details and Examples sections below for more information.

Usage

```
SoilTaxonomyDendrogram(spc, name = "hzname", rotationOrder=NULL,
max.depth = 150,
n.depth.ticks = 6, scaling.factor = 0.015, cex.names = 0.75,
cex.id = 0.75, axis.line.offset = -4, width = 0.1, y.offset = 0.5,
shrink=FALSE, font.id=2,
cex.taxon.labels = 0.66, dend.color = par("fg"), dend.width = 1)
```

Arguments

spc	a SoilProfileCollection object, see details
name	column name containing horizon names
rotationOrder	numeric vector with desired ordering of leaves in the dendrogram from left to right, see details
max.depth	depth at which profiles are truncated for plotting
n.depth.ticks	suggested number of ticks on the depth axis
scaling.factor	scaling factor used to convert depth units into plotting units
cex.names	character scaling for horizon names
cex.id	character scaling for profile IDs
axis.line.offset	horizontal offset for depth axis
width	width of profiles
y.offset	vertical offset between dendrogram and profiles
shrink	should long horizon names be shrunk by 80% ?
font.id	font style applied to profile id, default is 2 (bold)
cex.taxon.labels	character scaling for taxonomic information
dend.color	dendrogram line color
dend.width	dendrogram line width

Details

This function looks for specific site-level attributes named: `soilorder`, `suborder`, `greatgroup`, and `subgroup`.

The `rotationOrder` argument uses (i.e. requires) the `dendextend::rotate()` function to re-order leaves within the `hclust` representation of the ST hierarchy. Perfect sorting is not always possible.

Value

An invisibly-returned list containing:

`dist` pair-wise dissimilarity matrix

`order` final ordering of hclust leaves

Author(s)

D.E. Beaudette

References

D.E. Beaudette, P. Roudier and A.T. O'Geen. 2012. Algorithms for Quantitative Pedology, a Toolkit for Soil Scientists. Computers & Geosciences: 52: 258–268. (doi: 10.1016/j.cageo.2012.10.020)

Examples

```
if(requireNamespace("curl") &
   curl::has_internet() &
   require(aqp) &
   require(soilDB)
) {



# soils of interest
s.list <- c('musick', 'cecil', 'drummer', 'amador', 'pentz', 'reiff',
'san joaquin','montpellier','grangeville','pollasky','ramona')

# fetch and convert data into an SPC
h <- fetchOSD(s.list)

# plot dendrogram + profiles
SoilTaxonomyDendrogram(h)

# again, this time save the pair-wise dissimilarity matrix
# note that there isn't a lot of discrimination between soils
(d <- SoilTaxonomyDendrogram(h))


# a different set
soils <- c('cecil', 'altavista', 'lloyd', 'wickham', 'wilkes',
```

```
'chewacla', 'congaree')

# get morphology + extended summaries for sorting of dendrogram
s <- fetchOSD(soils, extended = TRUE)

# get summary and ignore the figure
res <- vizHillslopePosition(s$hillpos)

# compare default sorting to soils sorting according to catenary, e.g.
# hillslope position
par(mar=c(0,0,1,1), mfrow=c(2,1))

SoilTaxonomyDendrogram(s$SPC, width=0.25)
mtext('default sorting', side = 2, line=-1, font=3, cex=1.25)

SoilTaxonomyDendrogram(s$SPC, rotationOrder = res$order, width=0.25)
mtext('approx. catenary sorting', side = 2, line=-1, font=3, cex=1.25)

}
```

`vizAnnualClimate` *Annual Climate Summaries for Soil Series Data*

Description

Annual climate summaries for soil series, based on `latticeExtra::segplot`, based on 5th, 25th, 50th, 75th, and 95th percentiles. Input data should be from `soilDB::fetchOSD`.

Usage

```
vizAnnualClimate(climate.data, IQR.cex=1, s=NULL, s.col='firebrick', ...)
```

Arguments

`climate.data`	Annual climate summaries, as returned from `soilDB::fetchOSD(...,extended=TRUE)`
`IQR.cex`	scaling factor for bar representing interquartile range
`s`	a soil series name, e.g. "AMADOR", to highlight
`s.col`	color for highlighted soil series
`...`	further arguments passed to `latticeExtra::segplot`

Details

This function was designed for use with `soilDB::fetchOSD`. It might be possible to use with other sources of data but your mileage may vary.

Value

a `list` with the following elements:

`fig`	lattice object (the figure)
`clust`	clustering object returned by `cluster::diana`

Author(s)

D.E. Beaudette

See Also

`vizHillslopePosition`

Examples

```
if(requireNamespace("curl") &
   curl::has_internet() &
   require(soilDB) &
   require(aqp) &
   require(latticeExtra)
) {

  # soil series of interest
soil <- 'ARBUCKLE'

# get competing series
sdata <- fetchOSD(soil, extended = TRUE)

# get competing series' data
sdata.competing <- fetchOSD(c(soil, sdata$competing$competing))

# only use established series
idx <- which(sdata.competing$series_status == 'established')

# subset as needed
if(length(idx) < length(sdata.competing)) {
  sdata.competing <- sdata.competing[idx, ]

}

# now get the extended data
sdata.competing.full <- fetchOSD(site(sdata.competing)$id, extended = TRUE)

# extract SPC
spc <- sdata.competing.full$SPC

# full set of series names
s.names <- unique(site(spc)$id)
```

```
# todo: probably better ways to do this...
# note: need to load lattice for this to work
trellis.par.set(plot.line=list(col='RoyalBlue'))

# control center symbol and size here
res <- vizAnnualClimate(sdata.competing.full$climate.annual, s=soil, IQR.cex = 1.1, cex=1.1, pch=18)

# plot figure
print(res$fig)

# check clustering
str(res$clust)

# do something with clustering
par(mar=c(0,0,1,1))
plotProfileDendrogram(spc, clust = res$clust, scaling.factor = 0.075, width = 0.2, y.offset = 0.5)
mtext('sorted by annual climate summaries', side = 3, at = 0.5, adj = 0, line = -1.5, font=3)

}
```

`vizHillslopePosition` *Hillslope / Geomorphic Component Vizualization*

Description

A unique display of hillslope or geomorhphic component probability.

Usage

```
vizHillslopePosition(x, s = NULL)
vizGeomorphicComponent(x, s = NULL)
```

Arguments

x	`data.frame` as created by `soilDB::fetchOSD(...,extended=TRUE)`, see details
s	an optional soil series name, highlighted in the figure

Details

Example usage

Value

a `list` with the following elements:

`fig`	lattice object (the figure)
`order`	ordering of soil series

Note

This is a work in progress.

Author(s)

D.E. Beaudette

References

http://ncss-tech.github.io/AQP/soilDB/soil-series-query-functions.html

Examples

```
if(requireNamespace("curl") &
    curl::has_internet() &
    require(aqp) &
    require(soilDB)) {

# soils of interest
s.list <- c('musick', 'cecil', 'drummer', 'amador', 'pentz', 'reiff',
'san joaquin','montpellier','grangeville','pollasky','ramona')

# fetch and convert data into an SPC
s <- fetchOSD(s.list, extended=TRUE)

res <- vizHillslopePosition(s$hillpos)
print(res$fig)

}
```

Index

∗Topic **datasets**
amador, 4
CDEC.snow.courses, 6
HHM, 23
∗Topic **hplots**
aggregateColorPlot, 3
aspect.plot, 5
diagnosticPropertyPlot, 15
dueling.dendrograms, 18
PCP_plot, 28
percentileDemo, 30
plotAvailWater, 31
plotProfileDendrogram, 32
plotSoilRelationChordGraph, 33
plotTransect, 37
SoilTaxonomyDendrogram, 45
∗Topic **hplot**
plotSoilRelationGraph, 34
vizAnnualClimate, 47
∗Topic **manip**
CDEC_StationInfo, 12
component.adj.matrix, 13
constantDensitySampling, 14
dist.along.grad, 16
formatPLSS, 20
generateLineHash, 21
geomorphBySoilSeries-SSURGO, 22
joinAdjacency, 24
LL2PLSS, 25
multinominal2logical, 27
PLSS2LL, 39
polygonAdjacency, 41
sample.by.poly, 42
sampleRasterStackByMU, 43
site_photos_kml, 44

aggregateColorPlot, 3
alignDOY *(*FFD*)*, 19
amador, 4
aspect.plot, 5

CDEC.snow.courses, 6
CDEC_StationInfo, *8*, 12
CDECquery, 7, *12*
CDECsnowQuery, *8*, 10
component.adj.matrix, 13
constantDensitySampling, 14, *42*, *44*

d4 *(*LL2PLSS*)*, 25
diagnosticPropertyPlot, 15, *28*
diagnosticPropertyPlot2
*(*diagnosticPropertyPlot*)*, 15
dist.along.grad, 16
dueling.dendrograms, 18

ESS_by_Moran_I *(*sampleRasterStackByMU*)*, 43

FFD, 19
FFDplot *(*FFD*)*, 19
findFirstLastFrostDOY *(*FFD*)*, 19
formatPLSS, 20, *25*, *40*
frostFreePeriod *(*FFD*)*, 19

generateLineHash, 21
geomorphBySoilSeries-SSURGO, 22
geomPosHillProbability
*(*geomorphBySoilSeries-SSURGO*)*, 22
geomPosMountainProbability
*(*geomorphBySoilSeries-SSURGO*)*, 22

HHM, 23
hillslope.probability
*(*geomorphBySoilSeries-SSURGO*)*, 22
hillslopeProbability
*(*geomorphBySoilSeries-SSURGO*)*, 22

joinAdjacency, 24

LL2PLSS, 25, *40*

makeFrostMatrix *(*FFD*)*, 19
monthlyWB, 26
Moran_I_ByRaster
*(*sampleRasterStackByMU*)*, 43
multinominal2logical, *16*, 27

PCP_plot, 28
percentileDemo, 30
plotAvailWater, 31
plotProfileDendrogram, 32
plotSoilRelationChordGraph, 33
plotSoilRelationGraph, *24*, *34*, 34
plotSPC, *33*
plotTransect, *17*, 37
plotWB *(*monthlyWB*)*, 26
PLSS2LL, *20*, *21*, *25*, 39
PLSS2LL_1 *(*PLSS2LL*)*, 39
plssMeridians *(*LL2PLSS*)*, 25
polygonAdjacency, 41

sample.by.poly, *15*, 42, *44*
sampleRasterStackByMU, 43
samplingStability
*(*sampleRasterStackByMU*)*, 43
sharpshootR *(*sharpshootR-package*)*, 3
sharpshootR-package, 3
site_photos_kml, 44
SoilTaxonomyDendrogram, 45
spsample, *42*
surfaceShapeProbability
*(*geomorphBySoilSeries-SSURGO*)*,
22

vizAnnualClimate, 47
vizGeomorphicComponent
*(*vizHillslopePosition*)*, 49
vizHillslopePosition, *48*, 49

waterDayYear, *29*