Package 'openairmaps'

November 3, 2023

Type Package

Title Create Maps of Air Pollution Data

Version 0.8.1

Description Combine the air quality data analysis methods of 'openair' with the JavaScript 'Leaflet' (<https://leafletjs.com/>) library. Functionality includes plotting site maps, ``directional analysis'' figures such as polar plots, and air mass trajectories.

License GPL (>= 3)

URL https://davidcarslaw.github.io/openairmaps/, https://github.com/davidcarslaw/openairmaps

BugReports https://github.com/davidcarslaw/openairmaps/issues

Depends R (>= 3.2.0)

Imports cli, dplyr, forcats, ggmap, ggplot2, ggtext, leaflet, lifecycle, lubridate, magrittr, mgcv, openair (>= 2.13), purrr (>= 1.0.0), rlang, sf, stringr, tibble, tidyr

Suggests httr, jsonlite, worldmet

Encoding UTF-8

Language en-GB

LazyData true

RoxygenNote 7.2.3

NeedsCompilation no

Author Jack Davison [cre, aut], David Carslaw [aut]

Maintainer Jack Davison <jack.davison@ricardo.com>

Repository CRAN

Date/Publication 2023-11-03 10:30:02 UTC

R topics documented:

`addPolarMarkers` *Add polar markers to leaflet map*

Description

This function is similar (but not identical to) the `leaflet::addMarkers()` and `leaflet::addCircleMarkers()` functions in `leaflet`, which allows users to add `openair` directional analysis plots to any leaflet map and have more control over groups and layerIds than in "all-in-one" functions like `polarMap()`.

Usage

```
addPolarMarkers(
  map,
  pollutant,
  fun = openair::polarPlot,
  lng = NULL,
```

```
  lat = NULL,
  layerId = NULL,
  group = NULL,
  popup = NULL,
  popupOptions = NULL,
  label = NULL,
  labelOptions = NULL,
  options = leaflet::markerOptions(),
  clusterOptions = NULL,
  clusterId = NULL,
  key = FALSE,
  d.icon = 200,
  d.fig = 3.5,
  data = leaflet::getMapData(map),
  ...
)

addPolarDiffMarkers(
  map,
  pollutant,
  before = leaflet::getMapData(map),
  after = leaflet::getMapData(map),
  lng = NULL,
  lat = NULL,
  layerId = NULL,
  group = NULL,
  popup = NULL,
  popupOptions = NULL,
  label = NULL,
  labelOptions = NULL,
  options = leaflet::markerOptions(),
  clusterOptions = NULL,
  clusterId = NULL,
  key = FALSE,
  d.icon = 200,
  d.fig = 3.5,
  ...
)
```

Arguments

map	a map widget object created from leaflet()
pollutant	The name of the pollutant to be plot. Note that, if fun = openair::windRose, you must set pollutant = "ws".
fun	An openair directional analysis plotting function. Supported functions include openair::polarPlot() (the default), openair::polarAnnulus(), openair::polarFreq(), openair::percentileRose(), openair::pollutionRose() and openair::windRose(). For openair::polarDiff(), use addPolarDiffMarkers().

lng	The decimal longitude.
lat	The decimal latitude.
layerId	the layer id
group	the name of the group the newly created layers should belong to (for clearGroup and addLayersControl purposes). Human-friendly group names are permitted–they need not be short, identifier-style names. Any number of layers and even different types of layers (e.g. markers and polygons) can share the same group name.
popup	A column of data to be used as a popup.
popupOptions	A Vector of popupOptions to provide popups
label	A column of data to be used as a label.
labelOptions	A Vector of labelOptions to provide label options for each label. Default NULL
options	a list of extra options for tile layers, popups, paths (circles, rectangles, polygons, ...), or other map elements
clusterOptions	if not NULL, markers will be clustered using Leaflet.markercluster; you can use markerClusterOptions() to specify marker cluster options
clusterId	the id for the marker cluster layer
key	Should a key for each marker be drawn? Default is FALSE.
d.icon	The diameter of the plot on the map in pixels. This will affect the size of the individual polar markers. Alternatively, a vector in the form c(width, height) can be provided if a non-circular marker is desired.
d.fig	The diameter of the plots to be produced using openair in inches. This will affect the resolution of the markers on the map. Alternatively, a vector in the form c(width, height) can be provided if a non-circular marker is desired.
data	A data frame. The data frame must contain the data to plot your choice of openair directional analysis plot, which includes wind speed (ws), wind direction (wd), and the column representing the concentration of a pollutant. In addition, data must include a decimal latitude and longitude. By default, it is the data object provided to leaflet::leaflet() initially, but can be overridden.
...	Other arguments for the plotting function (e.g. period for openair::polarAnnulus()).
before, after	A data frame that represents the before/after case. See openair::polarPlot() for details of different input requirements. By default, both before and after are the data object provided to leaflet::leaflet() initially, but at least one should be overridden.

Value

A leaflet object.

Functions

- addPolarMarkers(): Add any one-table polar marker (e.g., openair::polarPlot())
- addPolarDiffMarkers(): Add the two-table openair::polarDiff() marker.

Examples

```
## Not run:
library(leaflet)
library(openair)

# different types of polar plot on one map
leaflet(data = polar_data) %>%
  addTiles() %>%
  addPolarMarkers("ws",
    fun = openair::windRose,
    group = "Wind Rose"
  ) %>%
  addPolarMarkers("nox",
    fun = openair::polarPlot,
    group = "Polar Plot"
  ) %>%
  addLayersControl(
    baseGroups = c("Wind Rose", "Polar Plot")
  )

# use of polar diff (NB: both 'before' and 'after' inherit from `leaflet()`,
# so at least one should be overridden - in this case 'after')
leaflet(data = polar_data) %>%
  addTiles() %>%
  addPolarDiffMarkers("nox",
    after = dplyr::mutate(polar_data, nox = jitter(nox, 5))
  )

## End(Not run)
```

`addTrajPaths` *Add trajectory paths to leaflet map*

Description

This function is similar (but not identical to) the `leaflet::addMarkers()` function in `leaflet`, which allows users to add trajectory paths to any leaflet map and have more control over groups and layerIds than in "all-in-one" functions like `trajMap()`.

Usage

```
addTrajPaths(
  map,
  lng = "lon",
  lat = "lat",
  layerId = NULL,
  group = NULL,
  data = leaflet::getMapData(map),
  npoints = 12,
```

```
  ...
)
```

Arguments

`map`	a map widget object created from `leaflet::leaflet()`.
`lng`	The decimal longitude.
`lat`	The decimal latitude.
`layerId`	The layer id.
`group`	the name of the group the newly created layers should belong to (for `leaflet::clearGroup()` and `leaflet::addLayersControl()` purposes). Human-friendly group names are permitted–they need not be short, identifier-style names. Any number of layers and even different types of layers (e.g. markers and polygons) can share the same group name.
`data`	Data frame, the result of importing a trajectory file using `openair::importTraj()`. By default, it is the data object provided to `leaflet::leaflet()` initially, but can be overridden.
`npoints`	A dot is placed every `npoints` along each full trajectory. For hourly back trajectories points are plotted every `npoints` hours. This helps to understand where the air masses were at particular times and get a feel for the speed of the air (points closer together correspond to slower moving air masses). Defaults to 12.
`...`	Other arguments to pass to both `leaflet::addCircleMarkers()` and `leaflet::addPolylines()`. If you use the `color` argument, it is important to ensure the vector you supply is of length one to avoid issues with `leaflet::addPolylines()` (i.e., use `color = ~ pal(nox)[1]`). Note that `opacity` controls the opacity of the lines and `fillOpacity` the opacity of the markers.

Details

`addTrajPaths()` can be a powerful way of quickly plotting trajectories on a leaflet map, but users should take some care due to any additional arguments being passed to both `leaflet::addCircleMarkers()` and `leaflet::addPolylines()`. In particular, users should be weary of the use of the `color` argument. Specifically, if `color` is passed a vector of length greater than one, multiple polylines will be drawn on top of one another. At best this will affect opacity, but at worst this will significantly impact the performance of R and the final leaflet map.

To mitigate this, please ensure that any vector passed to `color` is of length one. This is simple if you want the whole path to be the same colour, but more difficult if you want to colour by a pollutant, for example. The easiest way to achieve this is to write a for loop or use another iterative approach (e.g. the `purrr` package) to add one path per arrival date. An example of this is provided in the Examples.

Value

A leaflet object.

Examples

```
## Not run:
library(leaflet)
library(openairmaps)

pal <- colorNumeric(palette = "viridis", domain = traj_data$nox)

map <- leaflet() %>%
  addTiles()

for (i in seq(length(unique(traj_data$date)))) {
  data <- dplyr::filter(traj_data, date == unique(traj_data$date)[i])

  map <- map %>%
    addTrajPaths(
      data = data,
      color = pal(data$nox)[1]
    )
}

map

## End(Not run)
```

---

annulusMap *Polar annulus plots on interactive leaflet maps*

Description

annulusMap() creates a leaflet map using polar annulus plots as markers. Any number of pollutants can be specified using the pollutant argument, and multiple layers of markers can be added and toggled between using control.

Usage

```
annulusMap(
  data,
  pollutant = NULL,
  period = "hour",
  limits = "free",
  latitude = NULL,
  longitude = NULL,
  control = NULL,
  popup = NULL,
  label = NULL,
  provider = "OpenStreetMap",
  cols = "turbo",
  alpha = 1,
```

```
  key = FALSE,
  draw.legend = TRUE,
  collapse.control = FALSE,
  d.icon = 200,
  d.fig = 3.5,
  type = deprecated(),
  ...
)
```

Arguments

`data`	A data frame. The data frame must contain the data to plot the directional analysis marker, which includes wind speed (`ws`), wind direction (`wd`), and the column representing the concentration of a pollutant. In addition, `data` must include a decimal latitude and longitude.
`pollutant`	The column name(s) of the pollutant(s) to plot. If multiple pollutants are specified, they can be toggled between using a "layer control" interface.
`period`	This determines the temporal period to consider. Options are "hour" (the default, to plot diurnal variations), "season" to plot variation throughout the year, "weekday" to plot day of the week variation and "trend" to plot the trend by wind direction.
`limits`	One of: • `"fixed"` which ensures all of the markers use the same colour scale. • `"free"` (the default) which allows all of the markers to use different colour scales. • A numeric vector in the form `c(lower, upper)` used to define the colour scale. For example, `limits = c(0, 100)` would force the plot limits to span 0-100.
`latitude, longitude`	The decimal latitude/longitude. If not provided, will be automatically inferred from data by looking for a column named "lat"/"latitude" or "lon"/"lng"/"long"/"longitude" (case-insensitively).
`control`	Used for splitting the input data into different groups which can be selected between using a "layer control" interface, passed to the `type` argument of `openair::cutData()`. `control` cannot be used if multiple `pollutant` columns have been provided.
`popup`	Columns to be used as the HTML content for marker popups. Popups may be useful to show information about the individual sites (e.g., site names, codes, types, etc.). If a vector of column names are provided they are passed to `buildPopup()` using its default values.
`label`	Column to be used as the HTML content for hover-over labels. Labels are useful for the same reasons as popups, though are typically shorter.
`provider`	The base map(s) to be used. See `http://leaflet-extras.github.io/leaflet-providers/preview/` for a list of all base maps that can be used. If multiple base maps are provided, they can be toggled between using a "layer control" interface. By default, the interface will use the provider names as labels, but users can define their own using a named vector (e.g., `c("Default" = "OpenStreetMap", "Satellite" = "Esri.WorldImagery")`)

`cols` The colours used for plotting. See `openair::openColours()` for more information.

`alpha` The alpha transparency to use for the plotting surface (a value between 0 and 1 with zero being fully transparent and 1 fully opaque).

`key` Should a key for each marker be drawn? Default is `FALSE`.

`draw.legend` When `limits` are specified, should a shared legend be created at the side of the map? Default is `TRUE`.

`collapse.control`
Should the "layer control" interface be collapsed? Defaults to `FALSE`.

`d.icon` The diameter of the plot on the map in pixels. This will affect the size of the individual polar markers. Alternatively, a vector in the form `c(width, height)` can be provided if a non-circular marker is desired.

`d.fig` The diameter of the plots to be produced using `openair` in inches. This will affect the resolution of the markers on the map. Alternatively, a vector in the form `c(width, height)` can be provided if a non-circular marker is desired.

`type` **[Deprecated]**. Different sites are now automatically detected based on latitude and longitude. Please use `label` and/or `popup` to label different sites.

`...` Arguments passed on to `openair::polarAnnulus`

`resolution` Two plot resolutions can be set: "normal" and "fine" (the default).

`local.tz` Should the results be calculated in local time that includes a treatment of daylight savings time (DST)? The default is not to consider DST issues, provided the data were imported without a DST offset. Emissions activity tends to occur at local time e.g. rush hour is at 8 am every day. When the clocks go forward in spring, the emissions are effectively released into the atmosphere typically 1 hour earlier during the summertime i.e. when DST applies. When plotting diurnal profiles, this has the effect of "smearing-out" the concentrations. Sometimes, a useful approach is to express time as local time. This correction tends to produce better-defined diurnal profiles of concentration (or other variables) and allows a better comparison to be made with emissions/activity data. If set to `FALSE` then GMT is used. Examples of usage include `local.tz = "Europe/London"`, `local.tz = "America/New_York"`. See `cutData` and `import` for more details.

`statistic` The statistic that should be applied to each wind speed/direction bin. Can be "mean" (default), "median", "max" (maximum), "frequency". "stdev" (standard deviation), "weighted.mean" or "cpf" (Conditional Probability Function). Because of the smoothing involved, the colour scale for some of these statistics is only to provide an indication of overall pattern and should not be interpreted in concentration units e.g. for `statistic = "weighted.mean"` where the bin mean is multiplied by the bin frequency and divided by the total frequency. In many cases using `polarFreq` will be better. Setting `statistic = "weighted.mean"` can be useful because it provides an indication of the concentration * frequency of occurrence and will highlight the wind speed/direction conditions that dominate the overall mean.

percentile If statistic = "percentile" or statistic = "cpf" then percentile is used, expressed from 0 to 100. Note that the percentile value is calculated in the wind speed, wind direction 'bins'. For this reason it can also be useful to set min.bin to ensure there are a sufficient number of points available to estimate a percentile. See quantile for more details of how percentiles are calculated.

width The width of the annulus; can be "normal" (the default), "thin" or "fat".

min.bin The minimum number of points allowed in a wind speed/wind direction bin. The default is 1. A value of two requires at least 2 valid records in each bin an so on; bins with less than 2 valid records are set to NA. Care should be taken when using a value > 1 because of the risk of removing real data points. It is recommended to consider your data with care. Also, the polarFreq function can be of use in such circumstances.

exclude.missing Setting this option to TRUE (the default) removes points from the plot that are too far from the original data. The smoothing routines will produce predictions at points where no data exist i.e. they predict. By removing the points too far from the original data produces a plot where it is clear where the original data lie. If set to FALSE missing data will be interpolated.

date.pad For type = "trend" (default), date.pad = TRUE will pad-out missing data to the beginning of the first year and the end of the last year. The purpose is to ensure that the trend plot begins and ends at the beginning or end of year.

force.positive The default is TRUE. Sometimes if smoothing data with steep gradients it is possible for predicted values to be negative. force.positive = TRUE ensures that predictions remain positive. This is useful for several reasons. First, with lots of missing data more interpolation is needed and this can result in artefacts because the predictions are too far from the original data. Second, if it is known beforehand that the data are all positive, then this option carries that assumption through to the prediction. The only likely time where setting force.positive = FALSE would be if background concentrations were first subtracted resulting in data that is legitimately negative. For the vast majority of situations it is expected that the user will not need to alter the default option.

k The smoothing value supplied to gam for the temporal and wind direction components, respectively. In some cases e.g. a trend plot with less than 1-year of data the smoothing with the default values may become too noisy and affected more by outliers. Choosing a lower value of k (say 10) may help produce a better plot.

normalise If TRUE concentrations are normalised by dividing by their mean value. This is done *after* fitting the smooth surface. This option is particularly useful if one is interested in the patterns of concentrations for several pollutants on different scales e.g. NOx and CO. Often useful if more than one pollutant is chosen.

key.header Adds additional text/labels to the scale key. For example, passing the options key.header = "header", key.footer = "footer1" adds addition text above and below the scale key. These arguments are passed to

drawOpenKey via quickText, applying the auto.text argument, to handle formatting.

key.footer see key.footer.

key.position Location where the scale key is to plotted. Allowed arguments currently include "top", "right", "bottom" and "left".

auto.text Either TRUE (default) or FALSE. If TRUE titles and axis labels will automatically try and format pollutant names and units properly e.g. by subscripting the '2' in NO2.

Value

A leaflet object.

See Also

the original openair::polarAnnulus()

annulusMapStatic() for the static ggmap equivalent of annulusMap()

Other interactive directional analysis maps: diffMap(), freqMap(), percentileMap(), polarMap(), pollroseMap(), windroseMap()

Examples

```
## Not run:
annulusMap(polar_data,
  pollutant = "nox",
  period = "hour",
  provider = "Stamen.Toner"
)

## End(Not run)
```

---

annulusMapStatic *Bivariate polar plots on a static ggmap*

Description

annulusMapStatic() creates a ggplot2 map using polar annulus plots as markers. As this function returns a ggplot2 object, further customisation can be achieved using functions like ggplot2::theme() and ggplot2::guides().

Usage

```
annulusMapStatic(
  data,
  pollutant = NULL,
  ggmap,
  period = "hour",
```

```
  facet = NULL,
  limits = "free",
  latitude = NULL,
  longitude = NULL,
  cols = "turbo",
  alpha = 1,
  key = FALSE,
  facet.nrow = NULL,
  d.icon = 150,
  d.fig = 3,
  ...
)
```

Arguments

`data`	A data frame. The data frame must contain the data to plot the directional analysis marker, which includes wind speed (`ws`), wind direction (`wd`), and the column representing the concentration of a pollutant. In addition, `data` must include a decimal latitude and longitude.
`pollutant`	The column name(s) of the pollutant(s) to plot. If multiple pollutants are specified, they will each form part of a separate panel.
`ggmap`	A ggmap object obtained using `ggmap::get_map()` or a similar function to use as the basemap.
`period`	This determines the temporal period to consider. Options are "hour" (the default, to plot diurnal variations), "season" to plot variation throughout the year, "weekday" to plot day of the week variation and "trend" to plot the trend by wind direction.
`facet`	Used for splitting the input data into different panels, passed to the `type` argument of `openair::cutData()`. `facet` cannot be used if multiple `pollutant` columns have been provided.
`limits`	One of: • `"fixed"` which ensures all of the markers use the same colour scale. • `"free"` (the default) which allows all of the markers to use different colour scales. • A numeric vector in the form `c(lower, upper)` used to define the colour scale. For example, `limits = c(0, 100)` would force the plot limits to span 0-100.
`latitude, longitude`	The decimal latitude/longitude. If not provided, will be automatically inferred from data by looking for a column named "lat"/"latitude" or "lon"/"lng"/"long"/"longitude" (case-insensitively).
`cols`	The colours used for plotting. See `openair::openColours()` for more information.
`alpha`	The alpha transparency to use for the plotting surface (a value between 0 and 1 with zero being fully transparent and 1 fully opaque).
`key`	Should a key for each marker be drawn? Default is `FALSE`.

`facet.nrow` Passed to the `nrow` argument of `ggplot2::facet_wrap()`.

`d.icon` The diameter of the plot on the map in pixels. This will affect the size of the individual polar markers. Alternatively, a vector in the form `c(width, height)` can be provided if a non-circular marker is desired.

`d.fig` The diameter of the plots to be produced using `openair` in inches. This will affect the resolution of the markers on the map. Alternatively, a vector in the form `c(width, height)` can be provided if a non-circular marker is desired.

`...` Arguments passed on to `openair::polarAnnulus`

`resolution` Two plot resolutions can be set: "normal" and "fine" (the default).

`local.tz` Should the results be calculated in local time that includes a treatment of daylight savings time (DST)? The default is not to consider DST issues, provided the data were imported without a DST offset. Emissions activity tends to occur at local time e.g. rush hour is at 8 am every day. When the clocks go forward in spring, the emissions are effectively released into the atmosphere typically 1 hour earlier during the summertime i.e. when DST applies. When plotting diurnal profiles, this has the effect of "smearing-out" the concentrations. Sometimes, a useful approach is to express time as local time. This correction tends to produce better-defined diurnal profiles of concentration (or other variables) and allows a better comparison to be made with emissions/activity data. If set to `FALSE` then GMT is used. Examples of usage include `local.tz = "Europe/London"`, `local.tz = "America/New_York"`. See `cutData` and `import` for more details.

`type` `type` determines how the data are split i.e. conditioned, and then plotted. The default is will produce a single plot using the entire data. Type can be one of the built-in types as detailed in `cutData` e.g. "season", "year", "weekday" and so on. For example, `type = "season"` will produce four plots — one for each season.

It is also possible to choose `type` as another variable in the data frame. If that variable is numeric, then the data will be split into four quantiles (if possible) and labelled accordingly. If type is an existing character or factor variable, then those categories/levels will be used directly. This offers great flexibility for understanding the variation of different variables and how they depend on one another.

Type can be up length two e.g. `type = c("season", "site")` will produce a 2x2 plot split by season and site. The use of two types is mostly meant for situations where there are several sites. Note, when two types are provided the first forms the columns and the second the rows.

Also note that for the `polarAnnulus` function some type/period combinations are forbidden or make little sense. For example, `type = "season"` and `period = "trend"` (which would result in a plot with too many gaps in it for sensible smoothing), or `type = "weekday"` and `period = "weekday"`.

`statistic` The statistic that should be applied to each wind speed/direction bin. Can be "mean" (default), "median", "max" (maximum), "frequency". "stdev" (standard deviation), "weighted.mean" or "cpf" (Conditional Probability Function). Because of the smoothing involved, the colour scale for some of these statistics is only to provide an indication of overall pattern

and should not be interpreted in concentration units e.g. for `statistic = "weighted.mean"` where the bin mean is multiplied by the bin frequency and divided by the total frequency. In many cases using `polarFreq` will be better. Setting `statistic = "weighted.mean"` can be useful because it provides an indication of the concentration * frequency of occurrence and will highlight the wind speed/direction conditions that dominate the overall mean.

`percentile` If `statistic = "percentile"` or `statistic = "cpf"` then `percentile` is used, expressed from 0 to 100. Note that the percentile value is calculated in the wind speed, wind direction 'bins'. For this reason it can also be useful to set `min.bin` to ensure there are a sufficient number of points available to estimate a percentile. See `quantile` for more details of how percentiles are calculated.

`width` The width of the annulus; can be "normal" (the default), "thin" or "fat".

`min.bin` The minimum number of points allowed in a wind speed/wind direction bin. The default is 1. A value of two requires at least 2 valid records in each bin an so on; bins with less than 2 valid records are set to NA. Care should be taken when using a value > 1 because of the risk of removing real data points. It is recommended to consider your data with care. Also, the `polarFreq` function can be of use in such circumstances.

`exclude.missing` Setting this option to `TRUE` (the default) removes points from the plot that are too far from the original data. The smoothing routines will produce predictions at points where no data exist i.e. they predict. By removing the points too far from the original data produces a plot where it is clear where the original data lie. If set to `FALSE` missing data will be interpolated.

`date.pad` For `type = "trend"` (default), `date.pad = TRUE` will pad-out missing data to the beginning of the first year and the end of the last year. The purpose is to ensure that the trend plot begins and ends at the beginning or end of year.

`force.positive` The default is `TRUE`. Sometimes if smoothing data with steep gradients it is possible for predicted values to be negative. `force.positive = TRUE` ensures that predictions remain positive. This is useful for several reasons. First, with lots of missing data more interpolation is needed and this can result in artefacts because the predictions are too far from the original data. Second, if it is known beforehand that the data are all positive, then this option carries that assumption through to the prediction. The only likely time where setting `force.positive = FALSE` would be if background concentrations were first subtracted resulting in data that is legitimately negative. For the vast majority of situations it is expected that the user will not need to alter the default option.

`k` The smoothing value supplied to `gam` for the temporal and wind direction components, respectively. In some cases e.g. a trend plot with less than 1-year of data the smoothing with the default values may become too noisy and affected more by outliers. Choosing a lower value of `k` (say 10) may help produce a better plot.

`normalise` If `TRUE` concentrations are normalised by dividing by their mean value. This is done *after* fitting the smooth surface. This option is particu-

larly useful if one is interested in the patterns of concentrations for several pollutants on different scales e.g. NOx and CO. Often useful if more than one `pollutant` is chosen.

`key.header` Adds additional text/labels to the scale key. For example, passing the options `key.header = "header", key.footer = "footer1"` adds addition text above and below the scale key. These arguments are passed to `drawOpenKey` via `quickText`, applying the `auto.text` argument, to handle formatting.

`key.footer` see `key.footer`.

`key.position` Location where the scale key is to plotted. Allowed arguments currently include `"top"`, `"right"`, `"bottom"` and `"left"`.

`auto.text` Either `TRUE` (default) or `FALSE`. If `TRUE` titles and axis labels will automatically try and format pollutant names and units properly e.g. by subscripting the '2' in NO2.

Value

a `ggplot2` plot with a `ggmap` basemap

Further customisation using ggplot2

As the outputs of the static directional analysis functions are `ggplot2` figures, further customisation is possible using functions such as `ggplot2::theme()`, `ggplot2::guides()` and `ggplot2::labs()`.

If multiple pollutants are specified, subscripting (e.g., the "x" in "NOx") is achieved using the ggtext package. Therefore if you choose to override the plot theme, it is recommended to use `[ggplot2::theme()]` and `[ggtext::element_markdown()]` to define the `strip.text` parameter.

When arguments like `limits`, `percentile` or `breaks` are defined, a legend is automatically added to the figure. Legends can be removed using `ggplot2::theme(legend.position = "none")`, or further customised using `ggplot2::guides()` and either `color = ggplot2::guide_colourbar()` for continuous legends or `fill = ggplot2::guide_legend()` for discrete legends.

See Also

the original `openair::polarAnnulus()`

`annulusMap()` for the interactive `leaflet` equivalent of `annulusMapStatic()`

Other static directional analysis maps: `diffMapStatic()`, `freqMapStatic()`, `percentileMapStatic()`, `polarMapStatic()`, `pollroseMapStatic()`, `windroseMapStatic()`

`buildPopup` *Build a Complex Popup for a Leaflet Map*

Description

Group a dataframe together by latitude/longitude columns and create a HTML popup with user-defined columns. By default, the unique values of character columns are collapsed into comma-separated lists, numeric columns are averaged, and date columns are presented as a range. This function returns the input dataframe appended with a "popup" column, which can then be used in the popup argument of a function like `polarMap()`.

Usage

```
buildPopup(
  data,
  cols,
  latitude = NULL,
  longitude = NULL,
  names = NULL,
  control = NULL,
  fun.character = function(x) paste(unique(x), collapse = ", "),
  fun.numeric = function(x) signif(mean(x, na.rm = TRUE), 3),
 fun.dttm = function(x) paste(lubridate::floor_date(range(x, na.rm = TRUE), "day"),
    collapse = " to ")
)
```

Arguments

`data`	A data frame containing latitude and longitude information that will go on to be used in a function such as `polarMap()`.
`cols`	A character vector of column names, the data from which will appear in the popup.
`latitude, longitude`	The decimal latitude/longitude. If not provided, will be automatically inferred from data by looking for a column named "lat"/"latitude" or "lon"/"lng"/"long"\"longitude". (case-insensitively).
`names`	Optional. A named vector used to rename certain columns in the popups. See the Example for more information.
`control`	Optional. Column which will be used for the `control` argument of other mapping functions. This only needs to be used if `control` is going to be used in `polarMap()` or another similar function, and you'd expect different values for the different map layers (for example, if you are calculating a mean pollutant concentration).
`fun.character`	A function to summarise character and factor columns. Defaults to collapsing unique values into a comma-separated list.
`fun.numeric`	A function to summarise numeric columns. Defaults to taking the mean to three significant figures.
`fun.dttm`	A function to summarise date columns. Defaults to presenting the date as a range.

Value

a tibble::tibble()

Examples

```
## Not run:
buildPopup(
  data = openairmaps::polar_data,
  cols = c("site", "site_type", "date", "nox"),
  names = c("Site" = "site", "Site Type" = "site_type", "Date Range" = "date")
) %>%
  polarMap("nox", popup = "popup")

## End(Not run)
```

---

convertPostcode *Convert a UK postcode to a latitude/longitude pair*

Description

This is a much simpler implementation of the tools found in the PostcodesioR R package, intended for use with the searchNetwork() function.

Usage

```
convertPostcode(postcode)
```

Arguments

postcode	A valid UK postcode, e.g., "SW1A 1AA".

Value

A list containing the latitude, longitude, and input postcode.

Source

https://postcodes.io/

See Also

The PostcodesioR package at https://github.com/ropensci/PostcodesioR/

Examples

```
# convert a UK postcode
convertPostcode("SW1A1AA")

## Not run:
# use with `searchNetwork()`
palace <- convertPostcode("SW1A1AA")
searchNetwork(lat = palace$lat, lng = palace$lng, max_dist = 10)

## End(Not run)
```

---

diffMap *Bivariate polar plots on interactive leaflet maps*

Description

diffMap() creates a leaflet map using bivariate polar "difference" plots as markers. Any number of pollutants can be specified using the pollutant argument, and multiple layers of markers can be added and toggled between using control.

Usage

```
diffMap(
  before,
  after,
  pollutant = NULL,
  x = "ws",
  limits = "free",
  latitude = NULL,
  longitude = NULL,
  control = NULL,
  popup = NULL,
  label = NULL,
  provider = "OpenStreetMap",
 cols = c("#002F70", "#3167BB", "#879FDB", "#C8D2F1", "#F6F6F6", "#F4C8C8", "#DA8A8B",
    "#AE4647", "#5F1415"),
  alpha = 1,
  key = FALSE,
  draw.legend = TRUE,
  collapse.control = FALSE,
  d.icon = 200,
  d.fig = 3.5,
  type = deprecated(),
  ...
)
```

Arguments

before	A data frame that represents the "before" case. See polarPlot() for details of different input requirements.
after	A data frame that represents the "after" case. See polarPlot() for details of different input requirements.
pollutant	Mandatory. A pollutant name corresponding to a variable in a data frame should be supplied e.g. pollutant = "nox". There can also be more than one pollutant specified e.g. pollutant = c("nox", "no2"). The main use of using two or more pollutants is for model evaluation where two species would be expected to have similar concentrations. This saves the user stacking the data and it is possible to work with columns of data directly. A typical use would be pollutant = c("obs", "mod") to compare two columns "obs" (the observations) and "mod" (modelled values). When pair-wise statistics such as Pearson correlation and regression techniques are to be plotted, pollutant takes two elements too. For example, pollutant = c("bc", "pm25") where "bc" is a function of "pm25".
x	Name of variable to plot against wind direction in polar coordinates, the default is wind speed, "ws".
limits	By default, each individual polar marker has its own colour scale. The limits argument will force all markers to use the same colour scale. The limits are set in the form c(lower, upper), so limits = c(-5, 5) would force the plot limits to span -5 to 5. It is recommended to use a symmetrical limit scale (along with a "diverging" colour palette) for effective visualisation.
latitude, longitude	The decimal latitude/longitude. If not provided, will be automatically inferred from data by looking for a column named "lat"/"latitude" or "lon"/"lng"/"long"/"longitude" (case-insensitively).
control	Used for splitting the input data into different groups which can be selected between using a "layer control" interface, passed to the type argument of openair::cutData(). control cannot be used if multiple pollutant columns have been provided.
popup	Columns to be used as the HTML content for marker popups. Popups may be useful to show information about the individual sites (e.g., site names, codes, types, etc.). If a vector of column names are provided they are passed to buildPopup() using its default values.
label	Column to be used as the HTML content for hover-over labels. Labels are useful for the same reasons as popups, though are typically shorter.
provider	The base map(s) to be used. See http://leaflet-extras.github.io/leaflet-providers/preview/ for a list of all base maps that can be used. If multiple base maps are provided, they can be toggled between using a "layer control" interface. By default, the interface will use the provider names as labels, but users can define their own using a named vector (e.g., c("Default" = "OpenStreetMap", "Satellite" = "Esri.WorldImagery"))
cols	The colours used for plotting. It is recommended to use a "diverging" colour palette (along with a symmetrical limit scale) for effective visualisation.
alpha	The alpha transparency to use for the plotting surface (a value between 0 and 1 with zero being fully transparent and 1 fully opaque).

`key` Should a key for each marker be drawn? Default is `FALSE`.

`draw.legend` When `limits` are specified, should a shared legend be created at the side of the map? Default is `TRUE`.

`collapse.control`
Should the "layer control" interface be collapsed? Defaults to `FALSE`.

`d.icon` The diameter of the plot on the map in pixels. This will affect the size of the individual polar markers. Alternatively, a vector in the form `c(width, height)` can be provided if a non-circular marker is desired.

`d.fig` The diameter of the plots to be produced using `openair` in inches. This will affect the resolution of the markers on the map. Alternatively, a vector in the form `c(width, height)` can be provided if a non-circular marker is desired.

`type` **[Deprecated]**. Different sites are now automatically detected based on latitude and longitude. Please use `label` and/or `popup` to label different sites.

`...` Arguments passed on to `openair::polarPlot`

`wd` Name of wind direction field.

`statistic` The statistic that should be applied to each wind speed/direction bin. Because of the smoothing involved, the colour scale for some of these statistics is only to provide an indication of overall pattern and should not be interpreted in concentration units e.g. for `statistic = "weighted.mean"` where the bin mean is multiplied by the bin frequency and divided by the total frequency. In many cases using `polarFreq` will be better. Setting `statistic = "weighted.mean"` can be useful because it provides an indication of the concentration * frequency of occurrence and will highlight the wind speed/direction conditions that dominate the overall mean.Can be:

- "mean" (default), "median", "max" (maximum), "frequency". "stdev" (standard deviation), "weighted.mean".
- `statistic = "nwr"` Implements the Non-parametric Wind Regression approach of Henry et al. (2009) that uses kernel smoothers. The `openair` implementation is not identical because Gaussian kernels are used for both wind direction and speed. The smoothing is controlled by `ws_spread` and `wd_spread`.
- `statistic = "cpf"` the conditional probability function (CPF) is plotted and a single (usually high) percentile level is supplied. The CPF is defined as CPF = my/ny, where my is the number of samples in the y bin (by default a wind direction, wind speed interval) with mixing ratios greater than the *overall* percentile concentration, and ny is the total number of samples in the same wind sector (see Ashbaugh et al., 1985). Note that percentile intervals can also be considered; see `percentile` for details.
- When `statistic = "r"` or `statistic = "Pearson"`, the Pearson correlation coefficient is calculated for *two* pollutants. The calculation involves a weighted Pearson correlation coefficient, which is weighted by Gaussian kernels for wind direction an the radial variable (by default wind speed). More weight is assigned to values close to a wind speed-direction interval. Kernel weighting is used to ensure that all data are used rather than relying on the potentially small number of values in a wind speed-direction interval.

- When `statistic = "Spearman"`, the Spearman correlation coefficient is calculated for *two* pollutants. The calculation involves a weighted Spearman correlation coefficient, which is weighted by Gaussian kernels for wind direction an the radial variable (by default wind speed). More weight is assigned to values close to a wind speed-direction interval. Kernel weighting is used to ensure that all data are used rather than relying on the potentially small number of values in a wind speed-direction interval.
- `"robust_slope"` is another option for pair-wise statistics and `"quantile.slope"`, which uses quantile regression to estimate the slope for a particular quantile level (see also `tau` for setting the quantile level).
- `"york_slope"` is another option for pair-wise statistics which uses the *York regression method* to estimate the slope. In this method the uncertainties in `x` and `y` are used in the determination of the slope. The uncertainties are provided by `x_error` and `y_error` — see below.

`exclude.missing` Setting this option to `TRUE` (the default) removes points from the plot that are too far from the original data. The smoothing routines will produce predictions at points where no data exist i.e. they predict. By removing the points too far from the original data produces a plot where it is clear where the original data lie. If set to `FALSE` missing data will be interpolated.

`uncertainty` Should the uncertainty in the calculated surface be shown? If `TRUE` three plots are produced on the same scale showing the predicted surface together with the estimated lower and upper uncertainties at the 95% confidence interval. Calculating the uncertainties is useful to understand whether features are real or not. For example, at high wind speeds where there are few data there is greater uncertainty over the predicted values. The uncertainties are calculated using the GAM and weighting is done by the frequency of measurements in each wind speed-direction bin. Note that if uncertainties are calculated then the type is set to "default".

`percentile` If `statistic = "percentile"` then `percentile` is used, expressed from 0 to 100. Note that the percentile value is calculated in the wind speed, wind direction 'bins'. For this reason it can also be useful to set `min.bin` to ensure there are a sufficient number of points available to estimate a percentile. See `quantile` for more details of how percentiles are calculated.
`percentile` is also used for the Conditional Probability Function (CPF) plots. `percentile` can be of length two, in which case the percentile *interval* is considered for use with CPF. For example, `percentile = c(90, 100)` will plot the CPF for concentrations between the 90 and 100th percentiles. Percentile intervals can be useful for identifying specific sources. In addition, `percentile` can also be of length 3. The third value is the 'trim' value to be applied. When calculating percentile intervals many can cover very low values where there is no useful information. The trim value ensures that values greater than or equal to the trim * mean value are considered *before* the percentile intervals are calculated. The effect is to extract more detail from many source signatures. See the manual for examples. Finally, if the trim value is less than zero the percentile range is interpreted as absolute concentration values and subsetting is carried out directly.

`weights` At the edges of the plot there may only be a few data points in each wind speed-direction interval, which could in some situations distort the plot if the concentrations are high. `weights` applies a weighting to reduce their influence. For example and by default if only a single data point exists then the weighting factor is 0.25 and for two points 0.5. To not apply any weighting and use the data as is, use `weights = c(1, 1, 1)`.
An alternative to down-weighting these points they can be removed altogether using `min.bin`.

`min.bin` The minimum number of points allowed in a wind speed/wind direction bin. The default is 1. A value of two requires at least 2 valid records in each bin an so on; bins with less than 2 valid records are set to NA. Care should be taken when using a value > 1 because of the risk of removing real data points. It is recommended to consider your data with care. Also, the `polarFreq` function can be of use in such circumstances.

`mis.col` When `min.bin` is > 1 it can be useful to show where data are removed on the plots. This is done by shading the missing data in `mis.col`. To not highlight missing data when `min.bin` > 1 choose `mis.col = "transparent"`.

`upper` This sets the upper limit wind speed to be used. Often there are only a relatively few data points at very high wind speeds and plotting all of them can reduce the useful information in the plot.

`force.positive` The default is `TRUE`. Sometimes if smoothing data with steep gradients it is possible for predicted values to be negative. `force.positive = TRUE` ensures that predictions remain positive. This is useful for several reasons. First, with lots of missing data more interpolation is needed and this can result in artefacts because the predictions are too far from the original data. Second, if it is known beforehand that the data are all positive, then this option carries that assumption through to the prediction. The only likely time where setting `force.positive = FALSE` would be if background concentrations were first subtracted resulting in data that is legitimately negative. For the vast majority of situations it is expected that the user will not need to alter the default option.

`k` This is the smoothing parameter used by the `gam` function in package `mgcv`. Typically, value of around 100 (the default) seems to be suitable and will resolve important features in the plot. The most appropriate choice of `k` is problem-dependent; but extensive testing of polar plots for many different problems suggests a value of `k` of about 100 is suitable. Setting `k` to higher values will not tend to affect the surface predictions by much but will add to the computation time. Lower values of `k` will increase smoothing. Sometimes with few data to plot `polarPlot` will fail. Under these circumstances it can be worth lowering the value of `k`.

`normalise` If `TRUE` concentrations are normalised by dividing by their mean value. This is done *after* fitting the smooth surface. This option is particularly useful if one is interested in the patterns of concentrations for several pollutants on different scales e.g. NOx and CO. Often useful if more than one `pollutant` is chosen.

`auto.text` Either `TRUE` (default) or `FALSE`. If `TRUE` titles and axis labels will automatically try and format pollutant names and units properly e.g. by subscripting the '2' in NO2.

`ws_spread` The value of sigma used for Gaussian kernel weighting of wind speed when `statistic = "nwr"` or when correlation and regression statistics are used such as *r*. Default is `0.5`.

`wd_spread` The value of sigma used for Gaussian kernel weighting of wind direction when `statistic = "nwr"` or when correlation and regression statistics are used such as *r*. Default is `4`.

`x_error` The `x` error / uncertainty used when `statistic = "york_slope"`.

`y_error` The `y` error / uncertainty used when `statistic = "york_slope"`.

`kernel` Type of kernel used for the weighting procedure for when correlation or regression techniques are used. Only `"gaussian"` is supported but this may be enhanced in the future.

`formula.label` When pair-wise statistics such as regression slopes are calculated and plotted, should a formula label be displayed?

`tau` The quantile to be estimated when `statistic` is set to `"quantile.slope"`. Default is `0.5` which is equal to the median and will be ignored if `"quantile.slope"` is not used.

`plot` Should a plot be produced? `FALSE` can be useful when analysing data to extract plot components and plotting them in other ways.

Value

A leaflet object.

See Also

the original `openair::polarDiff()`

`diffMapStatic()` for the static ggmap equivalent of `diffMap()`

Other interactive directional analysis maps: `annulusMap()`, `freqMap()`, `percentileMap()`, `polarMap()`, `pollroseMap()`, `windroseMap()`

Examples

```
## Not run:
# NB: "after" is some dummy data to demonstrate functionality
diffMap(
  before = polar_data,
  after = dplyr::mutate(polar_data, nox = jitter(nox, factor = 5)),
  pollutant = "nox",
  provider = "Stamen.Toner"
)

## End(Not run)
```

`diffMapStatic` *Bivariate polar plots on a static ggmap*

Description

`diffMapStatic()` creates a ggplot2 map using bivariate "difference" polar plots as markers. As this function returns a ggplot2 object, further customisation can be achieved using functions like `ggplot2::theme()` and `ggplot2::guides()`.

Usage

```
diffMapStatic(
  before,
  after,
  pollutant = NULL,
  ggmap,
  limits = "free",
  x = "ws",
  latitude = NULL,
  longitude = NULL,
  facet = NULL,
 cols = c("#002F70", "#3167BB", "#879FDB", "#C8D2F1", "#F6F6F6", "#F4C8C8", "#DA8A8B",
    "#AE4647", "#5F1415"),
  alpha = 1,
  key = FALSE,
  facet.nrow = NULL,
  d.icon = 150,
  d.fig = 3,
  ...
)
```

Arguments

`before`	A data frame that represents the "before" case. See `polarPlot()` for details of different input requirements.
`after`	A data frame that represents the "after" case. See `polarPlot()` for details of different input requirements.
`pollutant`	The column name(s) of the pollutant(s) to plot. If multiple pollutants are specified, they will each form part of a separate panel.
`ggmap`	A ggmap object obtained using `ggmap::get_map()` or a similar function to use as the basemap.
`limits`	One of: • `"fixed"` which ensures all of the markers use the same colour scale. • `"free"` (the default) which allows all of the markers to use different colour scales.

- A numeric vector in the form `c(lower, upper)` used to define the colour scale. For example, `limits = c(0, 100)` would force the plot limits to span 0-100.

`x` The radial axis variable to plot.

`latitude, longitude`
The decimal latitude/longitude. If not provided, will be automatically inferred from data by looking for a column named "lat"/"latitude" or "lon"/"lng"/"long"/"longitude" (case-insensitively).

`facet` Used for splitting the input data into different panels, passed to the `type` argument of `openair::cutData()`. `facet` cannot be used if multiple `pollutant` columns have been provided.

`cols` The colours used for plotting. See `openair::openColours()` for more information.

`alpha` The alpha transparency to use for the plotting surface (a value between 0 and 1 with zero being fully transparent and 1 fully opaque).

`key` Should a key for each marker be drawn? Default is `FALSE`.

`facet.nrow` Passed to the `nrow` argument of `ggplot2::facet_wrap()`.

`d.icon` The diameter of the plot on the map in pixels. This will affect the size of the individual polar markers. Alternatively, a vector in the form `c(width, height)` can be provided if a non-circular marker is desired.

`d.fig` The diameter of the plots to be produced using `openair` in inches. This will affect the resolution of the markers on the map. Alternatively, a vector in the form `c(width, height)` can be provided if a non-circular marker is desired.

`...` Arguments passed on to `openair::polarPlot`

`wd` Name of wind direction field.

`statistic` The statistic that should be applied to each wind speed/direction bin. Because of the smoothing involved, the colour scale for some of these statistics is only to provide an indication of overall pattern and should not be interpreted in concentration units e.g. for `statistic = "weighted.mean"` where the bin mean is multiplied by the bin frequency and divided by the total frequency. In many cases using `polarFreq` will be better. Setting `statistic = "weighted.mean"` can be useful because it provides an indication of the concentration * frequency of occurrence and will highlight the wind speed/direction conditions that dominate the overall mean.Can be:

- “mean” (default), “median”, “max” (maximum), “frequency”. “stdev” (standard deviation), “weighted.mean”.
- `statistic = "nwr"` Implements the Non-parametric Wind Regression approach of Henry et al. (2009) that uses kernel smoothers. The `openair` implementation is not identical because Gaussian kernels are used for both wind direction and speed. The smoothing is controlled by `ws_spread` and `wd_spread`.
- `statistic = "cpf"` the conditional probability function (CPF) is plotted and a single (usually high) percentile level is supplied. The CPF is defined as CPF = my/ny, where my is the number of samples in the y

bin (by default a wind direction, wind speed interval) with mixing ratios greater than the *overall* percentile concentration, and ny is the total number of samples in the same wind sector (see Ashbaugh et al., 1985). Note that percentile intervals can also be considered; see `percentile` for details.

- When `statistic = "r"` or `statistic = "Pearson"`, the Pearson correlation coefficient is calculated for *two* pollutants. The calculation involves a weighted Pearson correlation coefficient, which is weighted by Gaussian kernels for wind direction an the radial variable (by default wind speed). More weight is assigned to values close to a wind speed-direction interval. Kernel weighting is used to ensure that all data are used rather than relying on the potentially small number of values in a wind speed-direction interval.
- When `statistic = "Spearman"`, the Spearman correlation coefficient is calculated for *two* pollutants. The calculation involves a weighted Spearman correlation coefficient, which is weighted by Gaussian kernels for wind direction an the radial variable (by default wind speed). More weight is assigned to values close to a wind speed-direction interval. Kernel weighting is used to ensure that all data are used rather than relying on the potentially small number of values in a wind speed-direction interval.
- `"robust_slope"` is another option for pair-wise statistics and `"quantile.slope"`, which uses quantile regression to estimate the slope for a particular quantile level (see also `tau` for setting the quantile level).
- `"york_slope"` is another option for pair-wise statistics which uses the *York regression method* to estimate the slope. In this method the uncertainties in `x` and `y` are used in the determination of the slope. The uncertainties are provided by `x_error` and `y_error` — see below.

`exclude.missing` Setting this option to `TRUE` (the default) removes points from the plot that are too far from the original data. The smoothing routines will produce predictions at points where no data exist i.e. they predict. By removing the points too far from the original data produces a plot where it is clear where the original data lie. If set to `FALSE` missing data will be interpolated.

`uncertainty` Should the uncertainty in the calculated surface be shown? If `TRUE` three plots are produced on the same scale showing the predicted surface together with the estimated lower and upper uncertainties at the 95% confidence interval. Calculating the uncertainties is useful to understand whether features are real or not. For example, at high wind speeds where there are few data there is greater uncertainty over the predicted values. The uncertainties are calculated using the GAM and weighting is done by the frequency of measurements in each wind speed-direction bin. Note that if uncertainties are calculated then the type is set to "default".

`percentile` If `statistic = "percentile"` then `percentile` is used, expressed from 0 to 100. Note that the percentile value is calculated in the wind speed, wind direction 'bins'. For this reason it can also be useful to set `min.bin` to ensure there are a sufficient number of points available to estimate a percentile. See `quantile` for more details of how percentiles are calculated.

percentile is also used for the Conditional Probability Function (CPF) plots. percentile can be of length two, in which case the percentile *interval* is considered for use with CPF. For example, percentile = c(90, 100) will plot the CPF for concentrations between the 90 and 100th percentiles. Percentile intervals can be useful for identifying specific sources. In addition, percentile can also be of length 3. The third value is the 'trim' value to be applied. When calculating percentile intervals many can cover very low values where there is no useful information. The trim value ensures that values greater than or equal to the trim * mean value are considered *before* the percentile intervals are calculated. The effect is to extract more detail from many source signatures. See the manual for examples. Finally, if the trim value is less than zero the percentile range is interpreted as absolute concentration values and subsetting is carried out directly.

weights At the edges of the plot there may only be a few data points in each wind speed-direction interval, which could in some situations distort the plot if the concentrations are high. weights applies a weighting to reduce their influence. For example and by default if only a single data point exists then the weighting factor is 0.25 and for two points 0.5. To not apply any weighting and use the data as is, use weights = c(1, 1, 1).
An alternative to down-weighting these points they can be removed altogether using min.bin.

min.bin The minimum number of points allowed in a wind speed/wind direction bin. The default is 1. A value of two requires at least 2 valid records in each bin an so on; bins with less than 2 valid records are set to NA. Care should be taken when using a value > 1 because of the risk of removing real data points. It is recommended to consider your data with care. Also, the polarFreq function can be of use in such circumstances.

mis.col When min.bin is > 1 it can be useful to show where data are removed on the plots. This is done by shading the missing data in mis.col. To not highlight missing data when min.bin > 1 choose mis.col = "transparent".

upper This sets the upper limit wind speed to be used. Often there are only a relatively few data points at very high wind speeds and plotting all of them can reduce the useful information in the plot.

force.positive The default is TRUE. Sometimes if smoothing data with steep gradients it is possible for predicted values to be negative. force.positive = TRUE ensures that predictions remain positive. This is useful for several reasons. First, with lots of missing data more interpolation is needed and this can result in artefacts because the predictions are too far from the original data. Second, if it is known beforehand that the data are all positive, then this option carries that assumption through to the prediction. The only likely time where setting force.positive = FALSE would be if background concentrations were first subtracted resulting in data that is legitimately negative. For the vast majority of situations it is expected that the user will not need to alter the default option.

k This is the smoothing parameter used by the gam function in package mgcv. Typically, value of around 100 (the default) seems to be suitable and will resolve important features in the plot. The most appropriate choice of k is problem-dependent; but extensive testing of polar plots for many different

problems suggests a value of k of about 100 is suitable. Setting k to higher values will not tend to affect the surface predictions by much but will add to the computation time. Lower values of k will increase smoothing. Sometimes with few data to plot polarPlot will fail. Under these circumstances it can be worth lowering the value of k.

normalise If TRUE concentrations are normalised by dividing by their mean value. This is done *after* fitting the smooth surface. This option is particularly useful if one is interested in the patterns of concentrations for several pollutants on different scales e.g. NOx and CO. Often useful if more than one pollutant is chosen.

auto.text Either TRUE (default) or FALSE. If TRUE titles and axis labels will automatically try and format pollutant names and units properly e.g. by subscripting the '2' in NO2.

ws_spread The value of sigma used for Gaussian kernel weighting of wind speed when statistic = "nwr" or when correlation and regression statistics are used such as *r*. Default is 0.5.

wd_spread The value of sigma used for Gaussian kernel weighting of wind direction when statistic = "nwr" or when correlation and regression statistics are used such as *r*. Default is 4.

x_error The x error / uncertainty used when statistic = "york_slope".

y_error The y error / uncertainty used when statistic = "york_slope".

kernel Type of kernel used for the weighting procedure for when correlation or regression techniques are used. Only "gaussian" is supported but this may be enhanced in the future.

formula.label When pair-wise statistics such as regression slopes are calculated and plotted, should a formula label be displayed?

tau The quantile to be estimated when statistic is set to "quantile.slope". Default is 0.5 which is equal to the median and will be ignored if "quantile.slope" is not used.

plot Should a plot be produced? FALSE can be useful when analysing data to extract plot components and plotting them in other ways.

Value

a ggplot2 plot with a ggmap basemap

Further customisation using ggplot2

As the outputs of the static directional analysis functions are ggplot2 figures, further customisation is possible using functions such as ggplot2::theme(), ggplot2::guides() and ggplot2::labs().

If multiple pollutants are specified, subscripting (e.g., the "x" in "NOx") is achieved using the ggtext package. Therefore if you choose to override the plot theme, it is recommended to use [ggplot2::theme()] and [ggtext::element_markdown()] to define the strip.text parameter.

When arguments like limits, percentile or breaks are defined, a legend is automatically added to the figure. Legends can be removed using ggplot2::theme(legend.position = "none"), or further customised using ggplot2::guides() and either color = ggplot2::guide_colourbar() for continuous legends or fill = ggplot2::guide_legend() for discrete legends.

See Also

the original openair::polarDiff()

diffMap() for the interactive leaflet equivalent of diffMapStatic()

Other static directional analysis maps: annulusMapStatic(), freqMapStatic(), percentileMapStatic(), polarMapStatic(), pollroseMapStatic(), windroseMapStatic()

---

freqMap *Polar frequency plots on interactive leaflet maps*

Description

freqMap() creates a leaflet map using binned polar plots as markers. Any number of pollutants can be specified using the pollutant argument, and multiple layers of markers can be added and toggled between using control.

Usage

```
freqMap(
  data,
  pollutant = NULL,
  statistic = "mean",
  breaks = "free",
  latitude = NULL,
  longitude = NULL,
  control = NULL,
  popup = NULL,
  label = NULL,
  provider = "OpenStreetMap",
  cols = "turbo",
  alpha = 1,
  key = FALSE,
  draw.legend = TRUE,
  collapse.control = FALSE,
  d.icon = 200,
  d.fig = 3.5,
  type = deprecated(),
  ...
)
```

Arguments

data	A data frame. The data frame must contain the data to plot the directional analysis marker, which includes wind speed (ws), wind direction (wd), and the column representing the concentration of a pollutant. In addition, data must include a decimal latitude and longitude.

`pollutant` The column name(s) of the pollutant(s) to plot. If multiple pollutants are specified, they can be toggled between using a "layer control" interface.

`statistic` The statistic that should be applied to each wind speed/direction bin. Can be "frequency", "mean", "median", "max" (maximum), "stdev" (standard deviation) or "weighted.mean". The option "frequency" is the simplest and plots the frequency of wind speed/direction in different bins. The scale therefore shows the counts in each bin. The option "mean" (the default) will plot the mean concentration of a pollutant (see next point) in wind speed/direction bins, and so on. Finally, "weighted.mean" will plot the concentration of a pollutant weighted by wind speed/direction. Each segment therefore provides the percentage overall contribution to the total concentration. Note that for options other than "frequency", it is necessary to also provide the name of a pollutant. See function `openair::cutData()` for further details.

`breaks` One of:

- `"fixed"` which ensures all of the markers use the same colour scale.
- `"free"` (the default) which allows all of the markers to use different colour scales.
- A numeric vector defining a sequence of numbers to use as the breaks. The sequence could represent one with equal spacing, e.g., `breaks = seq(0, 100, 10)` - a scale from 0-10 in intervals of 10, or a more flexible sequence, e.g., `breaks = c(0, 1, 5, 7, 10)`, which may be useful for some situations.

`latitude, longitude`
The decimal latitude/longitude. If not provided, will be automatically inferred from data by looking for a column named "lat"/"latitude" or "lon"/"lng"/"long"/"longitude" (case-insensitively).

`control` Used for splitting the input data into different groups which can be selected between using a "layer control" interface, passed to the `type` argument of `openair::cutData()`. `control` cannot be used if multiple `pollutant` columns have been provided.

`popup` Columns to be used as the HTML content for marker popups. Popups may be useful to show information about the individual sites (e.g., site names, codes, types, etc.). If a vector of column names are provided they are passed to `buildPopup()` using its default values.

`label` Column to be used as the HTML content for hover-over labels. Labels are useful for the same reasons as popups, though are typically shorter.

`provider` The base map(s) to be used. See http://leaflet-extras.github.io/leaflet-providers/preview/ for a list of all base maps that can be used. If multiple base maps are provided, they can be toggled between using a "layer control" interface. By default, the interface will use the provider names as labels, but users can define their own using a named vector (e.g., `c("Default" = "OpenStreetMap", "Satellite" = "Esri.WorldImagery")`)

`cols` The colours used for plotting. See `openair::openColours()` for more information.

`alpha` The alpha transparency to use for the plotting surface (a value between 0 and 1 with zero being fully transparent and 1 fully opaque).

`key` Should a key for each marker be drawn? Default is `FALSE`.

`draw.legend` When `breaks` are specified, should a shared legend be created at the side of the map? Default is `TRUE`.

`collapse.control`
Should the "layer control" interface be collapsed? Defaults to `FALSE`.

`d.icon` The diameter of the plot on the map in pixels. This will affect the size of the individual polar markers. Alternatively, a vector in the form `c(width, height)` can be provided if a non-circular marker is desired.

`d.fig` The diameter of the plots to be produced using `openair` in inches. This will affect the resolution of the markers on the map. Alternatively, a vector in the form `c(width, height)` can be provided if a non-circular marker is desired.

`type` **[Deprecated]**. Different sites are now automatically detected based on latitude and longitude. Please use `label` and/or popup to label different sites.

`...` Arguments passed on to `openair::polarFreq`

`ws.int` Wind speed interval assumed. In some cases e.g. a low met mast, an interval of 0.5 may be more appropriate.

`wd.nint` Number of intervals of wind direction.

`grid.line` Radial spacing of grid lines.

`trans` Should a transformation be applied? Sometimes when producing plots of this kind they can be dominated by a few high points. The default therefore is `TRUE` and a square-root transform is applied. This results in a non-linear scale and (usually) a better representation of the distribution. If set to `FALSE` a linear scale is used.

`min.bin` The minimum number of points allowed in a wind speed/wind direction bin. The default is 1. A value of two requires at least 2 valid records in each bin an so on; bins with less than 2 valid records are set to NA. Care should be taken when using a value > 1 because of the risk of removing real data points. It is recommended to consider your data with care. Also, the `polarFreq` function can be of use in such circumstances.

`ws.upper` A user-defined upper wind speed to use. This is useful for ensuring a consistent scale between different plots. For example, to always ensure that wind speeds are displayed between 1-10, set `ws.int = 10`.

`offset` `offset` controls the size of the 'hole' in the middle and is expressed as a percentage of the maximum wind speed. Setting a higher `offset` e.g. 50 is useful for `statistic = "weighted.mean"` when `ws.int` is greater than the maximum wind speed. See example below.

`border.col` The colour of the boundary of each wind speed/direction bin. The default is transparent. Another useful choice sometimes is "white".

`key.header` Adds additional text/labels to the scale key. For example, passing the options `key.header = "header", key.footer = "footer1"` adds addition text above and below the scale key. These arguments are passed to `drawOpenKey` via `quickText`, applying the `auto.text` argument, to handle formatting.

`key.footer` see `key.footer`.

`key.position` Location where the scale key is to plotted. Allowed arguments currently include `"top"`, `"right"`, `"bottom"` and `"left"`.

`auto.text` Either `TRUE` (default) or `FALSE`. If `TRUE` titles and axis labels will automatically try and format pollutant names and units properly e.g. by subscripting the '2' in NO2.

Value

A leaflet object.

See Also

the original `openair::polarFreq()`

`freqMapStatic()` for the static ggmap equivalent of `freqMap()`

Other interactive directional analysis maps: `annulusMap()`, `diffMap()`, `percentileMap()`, `polarMap()`, `pollroseMap()`, `windroseMap()`

Examples

```
## Not run:
freqMap(polar_data,
  pollutant = "nox",
  statistic = "mean",
  provider = "Stamen.Toner"
)

## End(Not run)
```

---

freqMapStatic *Polar frequency plots on a static ggmap*

Description

`freqMapStatic()` creates a ggplot2 map using polar frequency plots as markers. As this function returns a ggplot2 object, further customisation can be achieved using functions like `ggplot2::theme()` and `ggplot2::guides()`.

Usage

```
freqMapStatic(
  data,
  pollutant = NULL,
  ggmap,
  statistic = "mean",
  breaks = "free",
  latitude = NULL,
  longitude = NULL,
  facet = NULL,
  cols = "turbo",
  alpha = 1,
```

```
  key = FALSE,
  facet.nrow = NULL,
  d.icon = 150,
  d.fig = 3,
  ...
)
```

Arguments

`data`	A data frame. The data frame must contain the data to plot the directional analysis marker, which includes wind speed (`ws`), wind direction (`wd`), and the column representing the concentration of a pollutant. In addition, `data` must include a decimal latitude and longitude.
`pollutant`	The column name(s) of the pollutant(s) to plot. If multiple pollutants are specified, they will each form part of a separate panel.
`ggmap`	A ggmap object obtained using `ggmap::get_map()` or a similar function to use as the basemap.
`statistic`	The statistic that should be applied to each wind speed/direction bin. Can be "frequency", "mean", "median", "max" (maximum), "stdev" (standard deviation) or "weighted.mean". The option "frequency" is the simplest and plots the frequency of wind speed/direction in different bins. The scale therefore shows the counts in each bin. The option "mean" (the default) will plot the mean concentration of a pollutant (see next point) in wind speed/direction bins, and so on. Finally, "weighted.mean" will plot the concentration of a pollutant weighted by wind speed/direction. Each segment therefore provides the percentage overall contribution to the total concentration. Note that for options other than "frequency", it is necessary to also provide the name of a pollutant. See function `openair::cutData()` for further details.
`breaks`	One of: • `"fixed"` which ensures all of the markers use the same colour scale. • `"free"` (the default) which allows all of the markers to use different colour scales. • A numeric vector defining a sequence of numbers to use as the breaks. The sequence could represent one with equal spacing, e.g., `breaks = seq(0, 100, 10)` - a scale from 0-10 in intervals of 10, or a more flexible sequence, e.g., `breaks = c(0, 1, 5, 7, 10)`, which may be useful for some situations.
`latitude, longitude`	The decimal latitude/longitude. If not provided, will be automatically inferred from data by looking for a column named "lat"/"latitude" or "lon"/"lng"/"long"/"longitude" (case-insensitively).
`facet`	Used for splitting the input data into different panels, passed to the `type` argument of `openair::cutData()`. `facet` cannot be used if multiple `pollutant` columns have been provided.
`cols`	The colours used for plotting. See `openair::openColours()` for more information.

`alpha` The alpha transparency to use for the plotting surface (a value between 0 and 1 with zero being fully transparent and 1 fully opaque).

`key` Should a key for each marker be drawn? Default is `FALSE`.

`facet.nrow` Passed to the `nrow` argument of `ggplot2::facet_wrap()`.

`d.icon` The diameter of the plot on the map in pixels. This will affect the size of the individual polar markers. Alternatively, a vector in the form `c(width, height)` can be provided if a non-circular marker is desired.

`d.fig` The diameter of the plots to be produced using `openair` in inches. This will affect the resolution of the markers on the map. Alternatively, a vector in the form `c(width, height)` can be provided if a non-circular marker is desired.

`...` Arguments passed on to `openair::polarFreq`

`ws.int` Wind speed interval assumed. In some cases e.g. a low met mast, an interval of 0.5 may be more appropriate.

`wd.nint` Number of intervals of wind direction.

`grid.line` Radial spacing of grid lines.

`trans` Should a transformation be applied? Sometimes when producing plots of this kind they can be dominated by a few high points. The default therefore is `TRUE` and a square-root transform is applied. This results in a non-linear scale and (usually) a better representation of the distribution. If set to `FALSE` a linear scale is used.

`type` `type` determines how the data are split i.e. conditioned, and then plotted. The default is will produce a single plot using the entire data. Type can be one of the built-in types as detailed in `cutData` e.g. "season", "year", "weekday" and so on. For example, `type = "season"` will produce four plots — one for each season.

It is also possible to choose `type` as another variable in the data frame. If that variable is numeric, then the data will be split into four quantiles (if possible) and labelled accordingly. If type is an existing character or factor variable, then those categories/levels will be used directly. This offers great flexibility for understanding the variation of different variables and how they depend on one another.

Type can be up length two e.g. `type = c("season", "weekday")` will produce a 2x2 plot split by season and day of the week. Note, when two types are provided the first forms the columns and the second the rows.

`min.bin` The minimum number of points allowed in a wind speed/wind direction bin. The default is 1. A value of two requires at least 2 valid records in each bin an so on; bins with less than 2 valid records are set to NA. Care should be taken when using a value > 1 because of the risk of removing real data points. It is recommended to consider your data with care. Also, the `polarFreq` function can be of use in such circumstances.

`ws.upper` A user-defined upper wind speed to use. This is useful for ensuring a consistent scale between different plots. For example, to always ensure that wind speeds are displayed between 1-10, set `ws.int = 10`.

`offset` `offset` controls the size of the 'hole' in the middle and is expressed as a percentage of the maximum wind speed. Setting a higher `offset` e.g. 50 is useful for `statistic = "weighted.mean"` when `ws.int` is greater than the maximum wind speed. See example below.

`border.col` The colour of the boundary of each wind speed/direction bin. The default is transparent. Another useful choice sometimes is "white".

`key.header` Adds additional text/labels to the scale key. For example, passing the options `key.header = "header"`, `key.footer = "footer1"` adds addition text above and below the scale key. These arguments are passed to `drawOpenKey` via `quickText`, applying the `auto.text` argument, to handle formatting.

`key.footer` see `key.footer`.

`key.position` Location where the scale key is to plotted. Allowed arguments currently include `"top"`, `"right"`, `"bottom"` and `"left"`.

`auto.text` Either `TRUE` (default) or `FALSE`. If `TRUE` titles and axis labels will automatically try and format pollutant names and units properly e.g. by subscripting the '2' in NO2.

Value

a `ggplot2` plot with a `ggmap` basemap

Further customisation using ggplot2

As the outputs of the static directional analysis functions are `ggplot2` figures, further customisation is possible using functions such as `ggplot2::theme()`, `ggplot2::guides()` and `ggplot2::labs()`.

If multiple pollutants are specified, subscripting (e.g., the "x" in "NOx") is achieved using the ggtext package. Therefore if you choose to override the plot theme, it is recommended to use `[ggplot2::theme()]` and `[ggtext::element_markdown()]` to define the `strip.text` parameter.

When arguments like `limits`, `percentile` or `breaks` are defined, a legend is automatically added to the figure. Legends can be removed using `ggplot2::theme(legend.position = "none")`, or further customised using `ggplot2::guides()` and either `color = ggplot2::guide_colourbar()` for continuous legends or `fill = ggplot2::guide_legend()` for discrete legends.

See Also

the original `openair::polarFreq()`

`freqMap()` for the interactive `leaflet` equivalent of `freqMapStatic()`

Other static directional analysis maps: `annulusMapStatic()`, `diffMapStatic()`, `percentileMapStatic()`, `polarMapStatic()`, `pollroseMapStatic()`, `windroseMapStatic()`

---

networkMap *Create a leaflet map of air quality measurement network sites*

Description

This function uses `openair::importMeta()` to obtain metadata for measurement sites and uses it to create an attractive `leaflet` map. By default a map will be created in which readers may toggle between a vector base map and a satellite/aerial image, although users can further customise the control menu using the `provider` and `control` parameters.

Usage

```
networkMap(
  source = "aurn",
  control = NULL,
  year = NULL,
  cluster = TRUE,
  provider = c(Default = "OpenStreetMap", Satellite = "Esri.WorldImagery"),
  draw.legend = TRUE,
  collapse.control = FALSE
)
```

Arguments

source	One or more air quality networks for which data is available through openair. Available networks include: • "aurn", The UK Automatic Urban and Rural Network. • "aqe", The Air Quality England Network. • "saqn", The Scottish Air Quality Network. • "waqn", The Welsh Air Quality Network. • "ni", The Northern Ireland Air Quality Network. • "local", Locally managed air quality networks in England. • "kcl", King's College London networks. • "europe", European AirBase/e-reporting data. There are two additional options provided for convenience: • "ukaq" will return metadata for all networks for which data is imported by importUKAQ() (i.e., AURN, AQE, SAQN, WAQN, NI, and the local networks). • "all" will import all available metadata (i.e., "ukaq" plus "kcl" and "europe").
control	Option to add a "layer control" menu to allow readers to select between different site types. Can choose between effectively any column in the openair::importMeta() output, such as "variable", "site_type", or "agglomeration", as well as "network" when more than one source was specified.
year	By default, networkMap() visualises sites which are currently operational. year allows users to show sites open in a specific year, or over a range of years. See openair::importMeta() for more information.
cluster	When cluster = TRUE, markers are clustered together. This may be useful for sources like "kcl" where there are many markers very close together. Defaults to TRUE, and is forced to be TRUE when source = "europe" due to the large number of sites.
provider	The base map(s) to be used. See http://leaflet-extras.github.io/leaflet-providers/preview/ for a list of all base maps that can be used. If multiple base maps are provided, they can be toggled between using a "layer control" interface.
draw.legend	When multiple sources are specified, should a legend be created at the side of the map? Default is TRUE.
collapse.control	Should the "layer control" interface be collapsed? Defaults to FALSE.

Details

When selecting multiple data sources using `source`, please be mindful that there can be overlap between the different networks. For example, an air quality site in Scotland may be part of the AURN *and* the SAQN. `networkMap()` will only show one marker for such sites, and uses the order in which `source` arguments are provided as the hierarchy by which to assign sites to networks. The aforementioned AURN & SAQN site will therefore have its SAQN code displayed if `source = c("saqn", "aurn")`, and its AURN code displayed if `source = c("aurn", "saqn")`.

This hierarchy is also reflected when `control = "network"` is used. As `leaflet` markers cannot be part of multiple groups, the AURN & SAQN site will be part of the "SAQN" layer control group when `source = c("saqn", "aurn")` and the "AURN" layer control group when `source = c("aurn", "saqn")`.

Value

A leaflet object.

See Also

Other uk air quality network mapping functions: `searchNetwork()`

Examples

```
## Not run:
# view one network, grouped by site type
networkMap(source = "aurn", control = "site_type")

# view multiple networks, grouped by network
networkMap(source = c("aurn", "waqn", "saqn"), control = "network")

## End(Not run)
```

---

percentileMap *Percentile roses on interactive leaflet maps*

Description

`percentileMap()` creates a `leaflet` map using percentile roses as markers. Any number of pollutants can be specified using the `pollutant` argument, and multiple layers of markers can be added and toggled between using `control`.

Usage

```
percentileMap(
  data,
  pollutant = NULL,
  percentile = c(25, 50, 75, 90, 95),
```

```
  intervals = "fixed",
  latitude = NULL,
  longitude = NULL,
  control = NULL,
  popup = NULL,
  label = NULL,
  provider = "OpenStreetMap",
  cols = "turbo",
  alpha = 1,
  key = FALSE,
  draw.legend = TRUE,
  collapse.control = FALSE,
  d.icon = 200,
  d.fig = 3.5,
  type = deprecated(),
  ...
)
```

Arguments

`data`	A data frame. The data frame must contain the data to plot the directional analysis marker, which includes wind speed (`ws`), wind direction (`wd`), and the column representing the concentration of a pollutant. In addition, `data` must include a decimal latitude and longitude.
`pollutant`	The column name(s) of the pollutant(s) to plot. If multiple pollutants are specified, they can be toggled between using a "layer control" interface.
`percentile`	The percentile value(s) to plot. Must be between 0–100. If `percentile = NA` then only a mean line will be shown.
`intervals`	One of: • `"fixed"` (the default) which ensures all of the markers use the same radial axis scale. • `"free"` which allows all of the markers to use different radial axis scales. • A numeric vector defining a sequence of numbers to use as the intervals, e.g., `intervals = c(0, 10, 30, 50)`.
`latitude, longitude`	The decimal latitude/longitude. If not provided, will be automatically inferred from data by looking for a column named "lat"/"latitude" or "lon"/"lng"/"long"/"longitude" (case-insensitively).
`control`	Used for splitting the input data into different groups which can be selected between using a "layer control" interface, passed to the `type` argument of `openair::cutData()`. `control` cannot be used if multiple `pollutant` columns have been provided.
`popup`	Columns to be used as the HTML content for marker popups. Popups may be useful to show information about the individual sites (e.g., site names, codes, types, etc.). If a vector of column names are provided they are passed to `buildPopup()` using its default values.
`label`	Column to be used as the HTML content for hover-over labels. Labels are useful for the same reasons as popups, though are typically shorter.

provider The base map(s) to be used. See http://leaflet-extras.github.io/leaflet-providers/preview/ for a list of all base maps that can be used. If multiple base maps are provided, they can be toggled between using a "layer control" interface. By default, the interface will use the provider names as labels, but users can define their own using a named vector (e.g., `c("Default" = "OpenStreetMap", "Satellite" = "Esri.WorldImagery")`)

cols The colours used for plotting. See `openair::openColours()` for more information.

alpha The alpha transparency to use for the plotting surface (a value between 0 and 1 with zero being fully transparent and 1 fully opaque).

key Should a key for each marker be drawn? Default is `FALSE`.

draw.legend Should a shared legend be created at the side of the map? Default is `TRUE`.

collapse.control
Should the "layer control" interface be collapsed? Defaults to `FALSE`.

d.icon The diameter of the plot on the map in pixels. This will affect the size of the individual polar markers. Alternatively, a vector in the form `c(width, height)` can be provided if a non-circular marker is desired.

d.fig The diameter of the plots to be produced using `openair` in inches. This will affect the resolution of the markers on the map. Alternatively, a vector in the form `c(width, height)` can be provided if a non-circular marker is desired.

type **[Deprecated]**. Different sites are now automatically detected based on latitude and longitude. Please use `label` and/or popup to label different sites.

... Arguments passed on to `openair::percentileRose`

`wd` Name of wind direction field.

`smooth` Should the wind direction data be smoothed using a cyclic spline?

`method` When `method = "default"` the supplied percentiles by wind direction are calculated. When `method = "cpf"` the conditional probability function (CPF) is plotted and a single (usually high) percentile level is supplied. The CPF is defined as CPF = my/ny, where my is the number of samples in the wind sector y with mixing ratios greater than the *overall* percentile concentration, and ny is the total number of samples in the same wind sector (see Ashbaugh et al., 1985).

`angle` Default angle of "spokes" is when `smooth = FALSE`.

`mean` Show the mean by wind direction as a line?

`mean.lty` Line type for mean line.

`mean.lwd` Line width for mean line.

`mean.col` Line colour for mean line.

`fill` Should the percentile intervals be filled (default) or should lines be drawn (`fill = FALSE`).

`angle.scale` Sometimes the placement of the scale may interfere with an interesting feature. The user can therefore set `angle.scale` to any value between 0 and 360 degrees to mitigate such problems. For example `angle.scale = 45` will draw the scale heading in a NE direction.

`auto.text` Either `TRUE` (default) or `FALSE`. If `TRUE` titles and axis labels will automatically try and format pollutant names and units properly e.g. by subscripting the '2' in NO2.

`key.header` Adds additional text/labels to the scale key. For example, passing the options `key.header = "header"`, `key.footer = "footer1"` adds addition text above and below the scale key. These arguments are passed to `drawOpenKey` via `quickText`, applying the `auto.text` argument, to handle formatting.

`key.footer` see `key.footer`.

`key.position` Location where the scale key is to plotted. Allowed arguments currently include `"top"`, `"right"`, `"bottom"` and `"left"`.

Value

A leaflet object.

See Also

the original `openair::percentileRose()`

`percentileMapStatic()` for the static ggmap equivalent of `percentileMap()`

Other interactive directional analysis maps: `annulusMap()`, `diffMap()`, `freqMap()`, `polarMap()`, `pollroseMap()`, `windroseMap()`

Examples

```
## Not run:
percentileMap(polar_data,
  pollutant = "nox",
  provider = "Stamen.Toner"
)

## End(Not run)
```

`percentileMapStatic` *Percentile roses on a static ggmap*

Description

`percentileMapStatic()` creates a `ggplot2` map using percentile roses as markers. As this function returns a `ggplot2` object, further customisation can be achieved using functions like `ggplot2::theme()` and `ggplot2::guides()`.

Usage

```
percentileMapStatic(
  data,
  pollutant = NULL,
  ggmap,
  percentile = c(25, 50, 75, 90, 95),
  intervals = "fixed",
  latitude = NULL,
  longitude = NULL,
  facet = NULL,
  cols = "turbo",
  alpha = 1,
  key = FALSE,
  facet.nrow = NULL,
  d.icon = 150,
  d.fig = 3,
  ...
)
```

Arguments

`data`	A data frame. The data frame must contain the data to plot the directional analysis marker, which includes wind speed (`ws`), wind direction (`wd`), and the column representing the concentration of a pollutant. In addition, `data` must include a decimal latitude and longitude.
`pollutant`	The column name(s) of the pollutant(s) to plot. If multiple pollutants are specified, they will each form part of a separate panel.
`ggmap`	A ggmap object obtained using `ggmap::get_map()` or a similar function to use as the basemap.
`percentile`	The percentile value(s) to plot. Must be between 0–100. If `percentile = NA` then only a mean line will be shown.
`intervals`	One of: • `"fixed"` (the default) which ensures all of the markers use the same radial axis scale. • `"free"` which allows all of the markers to use different radial axis scales. • A numeric vector defining a sequence of numbers to use as the intervals, e.g., `intervals = c(0, 10, 30, 50)`.
`latitude, longitude`	The decimal latitude/longitude. If not provided, will be automatically inferred from data by looking for a column named "lat"/"latitude" or "lon"/"lng"/"long"/"longitude" (case-insensitively).
`facet`	Used for splitting the input data into different panels, passed to the `type` argument of `openair::cutData()`. `facet` cannot be used if multiple `pollutant` columns have been provided.
`cols`	The colours used for plotting. See `openair::openColours()` for more information.

alpha — The alpha transparency to use for the plotting surface (a value between 0 and 1 with zero being fully transparent and 1 fully opaque).

key — Should a key for each marker be drawn? Default is FALSE.

facet.nrow — Passed to the nrow argument of ggplot2::facet_wrap().

d.icon — The diameter of the plot on the map in pixels. This will affect the size of the individual polar markers. Alternatively, a vector in the form c(width, height) can be provided if a non-circular marker is desired.

d.fig — The diameter of the plots to be produced using openair in inches. This will affect the resolution of the markers on the map. Alternatively, a vector in the form c(width, height) can be provided if a non-circular marker is desired.

... — Arguments passed on to openair::percentileRose

wd Name of wind direction field.

type type determines how the data are split i.e. conditioned, and then plotted. The default is will produce a single plot using the entire data. Type can be one of the built-in types as detailed in cutData e.g. "season", "year", "weekday" and so on. For example, type = "season" will produce four plots — one for each season.
It is also possible to choose type as another variable in the data frame. If that variable is numeric, then the data will be split into four quantiles (if possible) and labelled accordingly. If type is an existing character or factor variable, then those categories/levels will be used directly. This offers great flexibility for understanding the variation of different variables and how they depend on one another.
Type can be up length two e.g. type = c("season", "weekday") will produce a 2x2 plot split by season and day of the week. Note, when two types are provided the first forms the columns and the second the rows.

smooth Should the wind direction data be smoothed using a cyclic spline?

method When method = "default" the supplied percentiles by wind direction are calculated. When method = "cpf" the conditional probability function (CPF) is plotted and a single (usually high) percentile level is supplied. The CPF is defined as CPF = my/ny, where my is the number of samples in the wind sector y with mixing ratios greater than the *overall* percentile concentration, and ny is the total number of samples in the same wind sector (see Ashbaugh et al., 1985).

angle Default angle of "spokes" is when smooth = FALSE.

mean Show the mean by wind direction as a line?

mean.lty Line type for mean line.

mean.lwd Line width for mean line.

mean.col Line colour for mean line.

fill Should the percentile intervals be filled (default) or should lines be drawn (fill = FALSE).

angle.scale Sometimes the placement of the scale may interfere with an interesting feature. The user can therefore set angle.scale to any value between 0 and 360 degrees to mitigate such problems. For example angle.scale = 45 will draw the scale heading in a NE direction.

auto.text Either TRUE (default) or FALSE. If TRUE titles and axis labels will automatically try and format pollutant names and units properly e.g. by subscripting the '2' in NO2.

key.header Adds additional text/labels to the scale key. For example, passing the options key.header = "header", key.footer = "footer1" adds addition text above and below the scale key. These arguments are passed to drawOpenKey via quickText, applying the auto.text argument, to handle formatting.

key.footer see key.footer.

key.position Location where the scale key is to plotted. Allowed arguments currently include "top", "right", "bottom" and "left".

Value

a ggplot2 plot with a ggmap basemap

Further customisation using ggplot2

As the outputs of the static directional analysis functions are ggplot2 figures, further customisation is possible using functions such as ggplot2::theme(), ggplot2::guides() and ggplot2::labs().

If multiple pollutants are specified, subscripting (e.g., the "x" in "NOx") is achieved using the ggtext package. Therefore if you choose to override the plot theme, it is recommended to use [ggplot2::theme()] and [ggtext::element_markdown()] to define the strip.text parameter.

When arguments like limits, percentile or breaks are defined, a legend is automatically added to the figure. Legends can be removed using ggplot2::theme(legend.position = "none"), or further customised using ggplot2::guides() and either color = ggplot2::guide_colourbar() for continuous legends or fill = ggplot2::guide_legend() for discrete legends.

See Also

the original openair::percentileRose()

percentileMap() for the interactive leaflet equivalent of percentileMapStatic()

Other static directional analysis maps: annulusMapStatic(), diffMapStatic(), freqMapStatic(), polarMapStatic(), pollroseMapStatic(), windroseMapStatic()

---

polarMap *Bivariate polar plots on interactive leaflet maps*

Description

polarMap() creates a leaflet map using bivariate polar plots as markers. Any number of pollutants can be specified using the pollutant argument, and multiple layers of markers can be added and toggled between using control.

Usage

```
polarMap(
  data,
  pollutant = NULL,
  x = "ws",
  limits = "free",
  upper = "fixed",
  latitude = NULL,
  longitude = NULL,
  control = NULL,
  popup = NULL,
  label = NULL,
  provider = "OpenStreetMap",
  cols = "turbo",
  alpha = 1,
  key = FALSE,
  draw.legend = TRUE,
  collapse.control = FALSE,
  d.icon = 200,
  d.fig = 3.5,
  type = deprecated(),
  ...
)
```

Arguments

`data`	A data frame. The data frame must contain the data to plot the directional analysis marker, which includes wind speed (`ws`), wind direction (`wd`), and the column representing the concentration of a pollutant. In addition, `data` must include a decimal latitude and longitude.
`pollutant`	The column name(s) of the pollutant(s) to plot. If multiple pollutants are specified, they can be toggled between using a "layer control" interface.
`x`	The radial axis variable to plot.
`limits`	One of: • `"fixed"` which ensures all of the markers use the same colour scale. • `"free"` (the default) which allows all of the markers to use different colour scales. • A numeric vector in the form `c(lower, upper)` used to define the colour scale. For example, `limits = c(0, 100)` would force the plot limits to span 0-100.
`upper`	One of: • `"fixed"` (the default) which ensures all of the markers use the same radial axis scale. • `"free"` which allows all of the markers to use different radial axis scales. • A numeric value, used as the upper limit for the radial axis scale.

`latitude, longitude`
: The decimal latitude/longitude. If not provided, will be automatically inferred from data by looking for a column named "lat"/"latitude" or "lon"/"lng"/"long"/"longitude" (case-insensitively).

`control`
: Used for splitting the input data into different groups which can be selected between using a "layer control" interface, passed to the `type` argument of `openair::cutData()`. `control` cannot be used if multiple `pollutant` columns have been provided.

`popup`
: Columns to be used as the HTML content for marker popups. Popups may be useful to show information about the individual sites (e.g., site names, codes, types, etc.). If a vector of column names are provided they are passed to `buildPopup()` using its default values.

`label`
: Column to be used as the HTML content for hover-over labels. Labels are useful for the same reasons as popups, though are typically shorter.

`provider`
: The base map(s) to be used. See http://leaflet-extras.github.io/leaflet-providers/preview/ for a list of all base maps that can be used. If multiple base maps are provided, they can be toggled between using a "layer control" interface. By default, the interface will use the provider names as labels, but users can define their own using a named vector (e.g., `c("Default" = "OpenStreetMap", "Satellite" = "Esri.WorldImagery")`)

`cols`
: The colours used for plotting. See `openair::openColours()` for more information.

`alpha`
: The alpha transparency to use for the plotting surface (a value between 0 and 1 with zero being fully transparent and 1 fully opaque).

`key`
: Should a key for each marker be drawn? Default is `FALSE`.

`draw.legend`
: When `limits` are specified, should a shared legend be created at the side of the map? Default is `TRUE`.

`collapse.control`
: Should the "layer control" interface be collapsed? Defaults to `FALSE`.

`d.icon`
: The diameter of the plot on the map in pixels. This will affect the size of the individual polar markers. Alternatively, a vector in the form `c(width, height)` can be provided if a non-circular marker is desired.

`d.fig`
: The diameter of the plots to be produced using `openair` in inches. This will affect the resolution of the markers on the map. Alternatively, a vector in the form `c(width, height)` can be provided if a non-circular marker is desired.

`type`
: **[Deprecated]**. Different sites are now automatically detected based on latitude and longitude. Please use `label` and/or `popup` to label different sites.

`...`
: Arguments passed on to `openair::polarPlot`

  `wd` Name of wind direction field.

  `statistic` The statistic that should be applied to each wind speed/direction bin. Because of the smoothing involved, the colour scale for some of these statistics is only to provide an indication of overall pattern and should not be interpreted in concentration units e.g. for `statistic = "weighted.mean"` where the bin mean is multiplied by the bin frequency and divided by the total frequency. In many cases using `polarFreq` will be better. Setting

statistic = "weighted.mean" can be useful because it provides an indication of the concentration * frequency of occurrence and will highlight the wind speed/direction conditions that dominate the overall mean.Can be:

- "mean" (default), "median", "max" (maximum), "frequency". "stdev" (standard deviation), "weighted.mean".
- statistic = "nwr" Implements the Non-parametric Wind Regression approach of Henry et al. (2009) that uses kernel smoothers. The openair implementation is not identical because Gaussian kernels are used for both wind direction and speed. The smoothing is controlled by ws_spread and wd_spread.
- statistic = "cpf" the conditional probability function (CPF) is plotted and a single (usually high) percentile level is supplied. The CPF is defined as CPF = my/ny, where my is the number of samples in the y bin (by default a wind direction, wind speed interval) with mixing ratios greater than the *overall* percentile concentration, and ny is the total number of samples in the same wind sector (see Ashbaugh et al., 1985). Note that percentile intervals can also be considered; see percentile for details.
- When statistic = "r" or statistic = "Pearson", the Pearson correlation coefficient is calculated for *two* pollutants. The calculation involves a weighted Pearson correlation coefficient, which is weighted by Gaussian kernels for wind direction an the radial variable (by default wind speed). More weight is assigned to values close to a wind speed-direction interval. Kernel weighting is used to ensure that all data are used rather than relying on the potentially small number of values in a wind speed-direction interval.
- When statistic = "Spearman", the Spearman correlation coefficient is calculated for *two* pollutants. The calculation involves a weighted Spearman correlation coefficient, which is weighted by Gaussian kernels for wind direction an the radial variable (by default wind speed). More weight is assigned to values close to a wind speed-direction interval. Kernel weighting is used to ensure that all data are used rather than relying on the potentially small number of values in a wind speed-direction interval.
- "robust_slope" is another option for pair-wise statistics and "quantile.slope", which uses quantile regression to estimate the slope for a particular quantile level (see also tau for setting the quantile level).
- "york_slope" is another option for pair-wise statistics which uses the *York regression method* to estimate the slope. In this method the uncertainties in x and y are used in the determination of the slope. The uncertainties are provided by x_error and y_error — see below.

exclude.missing Setting this option to TRUE (the default) removes points from the plot that are too far from the original data. The smoothing routines will produce predictions at points where no data exist i.e. they predict. By removing the points too far from the original data produces a plot where it is clear where the original data lie. If set to FALSE missing data will be interpolated.

`uncertainty` Should the uncertainty in the calculated surface be shown? If `TRUE` three plots are produced on the same scale showing the predicted surface together with the estimated lower and upper uncertainties at the 95% confidence interval. Calculating the uncertainties is useful to understand whether features are real or not. For example, at high wind speeds where there are few data there is greater uncertainty over the predicted values. The uncertainties are calculated using the GAM and weighting is done by the frequency of measurements in each wind speed-direction bin. Note that if uncertainties are calculated then the type is set to "default".

`percentile` If `statistic = "percentile"` then `percentile` is used, expressed from 0 to 100. Note that the percentile value is calculated in the wind speed, wind direction 'bins'. For this reason it can also be useful to set `min.bin` to ensure there are a sufficient number of points available to estimate a percentile. See `quantile` for more details of how percentiles are calculated.
`percentile` is also used for the Conditional Probability Function (CPF) plots. `percentile` can be of length two, in which case the percentile *interval* is considered for use with CPF. For example, `percentile = c(90, 100)` will plot the CPF for concentrations between the 90 and 100th percentiles. Percentile intervals can be useful for identifying specific sources. In addition, `percentile` can also be of length 3. The third value is the 'trim' value to be applied. When calculating percentile intervals many can cover very low values where there is no useful information. The trim value ensures that values greater than or equal to the trim * mean value are considered *before* the percentile intervals are calculated. The effect is to extract more detail from many source signatures. See the manual for examples. Finally, if the trim value is less than zero the percentile range is interpreted as absolute concentration values and subsetting is carried out directly.

`weights` At the edges of the plot there may only be a few data points in each wind speed-direction interval, which could in some situations distort the plot if the concentrations are high. `weights` applies a weighting to reduce their influence. For example and by default if only a single data point exists then the weighting factor is 0.25 and for two points 0.5. To not apply any weighting and use the data as is, use `weights = c(1, 1, 1)`.
An alternative to down-weighting these points they can be removed altogether using `min.bin`.

`min.bin` The minimum number of points allowed in a wind speed/wind direction bin. The default is 1. A value of two requires at least 2 valid records in each bin an so on; bins with less than 2 valid records are set to NA. Care should be taken when using a value > 1 because of the risk of removing real data points. It is recommended to consider your data with care. Also, the `polarFreq` function can be of use in such circumstances.

`mis.col` When `min.bin` is > 1 it can be useful to show where data are removed on the plots. This is done by shading the missing data in `mis.col`. To not highlight missing data when `min.bin` > 1 choose `mis.col = "transparent"`.

`angle.scale` Sometimes the placement of the scale may interfere with an interesting feature. The user can therefore set `angle.scale` to any value between 0 and 360 degrees to mitigate such problems. For example `angle.scale = 45` will draw the scale heading in a NE direction.

`units` The units shown on the polar axis scale.

`force.positive` The default is `TRUE`. Sometimes if smoothing data with steep gradients it is possible for predicted values to be negative. `force.positive = TRUE` ensures that predictions remain positive. This is useful for several reasons. First, with lots of missing data more interpolation is needed and this can result in artefacts because the predictions are too far from the original data. Second, if it is known beforehand that the data are all positive, then this option carries that assumption through to the prediction. The only likely time where setting `force.positive = FALSE` would be if background concentrations were first subtracted resulting in data that is legitimately negative. For the vast majority of situations it is expected that the user will not need to alter the default option.

`k` This is the smoothing parameter used by the `gam` function in package `mgcv`. Typically, value of around 100 (the default) seems to be suitable and will resolve important features in the plot. The most appropriate choice of `k` is problem-dependent; but extensive testing of polar plots for many different problems suggests a value of `k` of about 100 is suitable. Setting `k` to higher values will not tend to affect the surface predictions by much but will add to the computation time. Lower values of `k` will increase smoothing. Sometimes with few data to plot `polarPlot` will fail. Under these circumstances it can be worth lowering the value of `k`.

`normalise` If `TRUE` concentrations are normalised by dividing by their mean value. This is done *after* fitting the smooth surface. This option is particularly useful if one is interested in the patterns of concentrations for several pollutants on different scales e.g. NOx and CO. Often useful if more than one `pollutant` is chosen.

`key.header` Adds additional text/labels to the scale key. For example, passing the options `key.header = "header", key.footer = "footer1"` adds addition text above and below the scale key. These arguments are passed to `drawOpenKey` via `quickText`, applying the `auto.text` argument, to handle formatting.

`key.footer` see `key.footer`.

`key.position` Location where the scale key is to plotted. Allowed arguments currently include `"top"`, `"right"`, `"bottom"` and `"left"`.

`auto.text` Either `TRUE` (default) or `FALSE`. If `TRUE` titles and axis labels will automatically try and format pollutant names and units properly e.g. by subscripting the '2' in NO2.

`ws_spread` The value of sigma used for Gaussian kernel weighting of wind speed when `statistic = "nwr"` or when correlation and regression statistics are used such as *r*. Default is `0.5`.

`wd_spread` The value of sigma used for Gaussian kernel weighting of wind direction when `statistic = "nwr"` or when correlation and regression statistics are used such as *r*. Default is `4`.

`x_error` The `x` error / uncertainty used when `statistic = "york_slope"`.

`y_error` The `y` error / uncertainty used when `statistic = "york_slope"`.

`kernel` Type of kernel used for the weighting procedure for when correlation or regression techniques are used. Only `"gaussian"` is supported but this

may be enhanced in the future.

`formula.label` When pair-wise statistics such as regression slopes are calculated and plotted, should a formula label be displayed?

`tau` The quantile to be estimated when `statistic` is set to `"quantile.slope"`. Default is `0.5` which is equal to the median and will be ignored if `"quantile.slope"` is not used.

Value

A leaflet object.

See Also

the original `openair::polarPlot()`

`polarMapStatic()` for the static ggmap equivalent of `polarMap()`

Other interactive directional analysis maps: `annulusMap()`, `diffMap()`, `freqMap()`, `percentileMap()`, `pollroseMap()`, `windroseMap()`

Examples

```
## Not run:
polarMap(polar_data,
  pollutant = "nox",
  x = "ws",
  provider = "Stamen.Toner"
)

## End(Not run)
```

---

polarMapStatic *Bivariate polar plots on a static ggmap*

Description

`polarMapStatic()` creates a ggplot2 map using bivariate polar plots as markers. As this function returns a ggplot2 object, further customisation can be achieved using functions like `ggplot2::theme()` and `ggplot2::guides()`.

Usage

```
polarMapStatic(
  data,
  pollutant = NULL,
  ggmap,
  x = "ws",
  limits = "free",
  upper = "fixed",
```

```
  latitude = NULL,
  longitude = NULL,
  facet = NULL,
  cols = "turbo",
  alpha = 1,
  key = FALSE,
  facet.nrow = NULL,
  d.icon = 150,
  d.fig = 3,
  ...
)
```

Arguments

`data`	A data frame. The data frame must contain the data to plot the directional analysis marker, which includes wind speed (`ws`), wind direction (`wd`), and the column representing the concentration of a pollutant. In addition, `data` must include a decimal latitude and longitude.
`pollutant`	The column name(s) of the pollutant(s) to plot. If multiple pollutants are specified, they will each form part of a separate panel.
`ggmap`	A ggmap object obtained using `ggmap::get_map()` or a similar function to use as the basemap.
`x`	The radial axis variable to plot.
`limits`	One of: • `"fixed"` which ensures all of the markers use the same colour scale. • `"free"` (the default) which allows all of the markers to use different colour scales. • A numeric vector in the form `c(lower, upper)` used to define the colour scale. For example, `limits = c(0, 100)` would force the plot limits to span 0-100.
`upper`	One of: • `"fixed"` (the default) which ensures all of the markers use the same radial axis scale. • `"free"` which allows all of the markers to use different radial axis scales. • A numeric value, used as the upper limit for the radial axis scale.
`latitude, longitude`	The decimal latitude/longitude. If not provided, will be automatically inferred from data by looking for a column named "lat"/"latitude" or "lon"/"lng"/"long"/"longitude" (case-insensitively).
`facet`	Used for splitting the input data into different panels, passed to the `type` argument of `openair::cutData()`. `facet` cannot be used if multiple `pollutant` columns have been provided.
`cols`	The colours used for plotting. See `openair::openColours()` for more information.

`alpha` The alpha transparency to use for the plotting surface (a value between 0 and 1 with zero being fully transparent and 1 fully opaque).

`key` Should a key for each marker be drawn? Default is `FALSE`.

`facet.nrow` Passed to the `nrow` argument of `ggplot2::facet_wrap()`.

`d.icon` The diameter of the plot on the map in pixels. This will affect the size of the individual polar markers. Alternatively, a vector in the form `c(width, height)` can be provided if a non-circular marker is desired.

`d.fig` The diameter of the plots to be produced using `openair` in inches. This will affect the resolution of the markers on the map. Alternatively, a vector in the form `c(width, height)` can be provided if a non-circular marker is desired.

`...` Arguments passed on to `openair::polarPlot`

`wd` Name of wind direction field.

`statistic` The statistic that should be applied to each wind speed/direction bin. Because of the smoothing involved, the colour scale for some of these statistics is only to provide an indication of overall pattern and should not be interpreted in concentration units e.g. for `statistic = "weighted.mean"` where the bin mean is multiplied by the bin frequency and divided by the total frequency. In many cases using `polarFreq` will be better. Setting `statistic = "weighted.mean"` can be useful because it provides an indication of the concentration * frequency of occurrence and will highlight the wind speed/direction conditions that dominate the overall mean.Can be:

- "mean" (default), "median", "max" (maximum), "frequency". "stdev" (standard deviation), "weighted.mean".
- `statistic = "nwr"` Implements the Non-parametric Wind Regression approach of Henry et al. (2009) that uses kernel smoothers. The `openair` implementation is not identical because Gaussian kernels are used for both wind direction and speed. The smoothing is controlled by `ws_spread` and `wd_spread`.
- `statistic = "cpf"` the conditional probability function (CPF) is plotted and a single (usually high) percentile level is supplied. The CPF is defined as CPF = my/ny, where my is the number of samples in the y bin (by default a wind direction, wind speed interval) with mixing ratios greater than the *overall* percentile concentration, and ny is the total number of samples in the same wind sector (see Ashbaugh et al., 1985). Note that percentile intervals can also be considered; see `percentile` for details.
- When `statistic = "r"` or `statistic = "Pearson"`, the Pearson correlation coefficient is calculated for *two* pollutants. The calculation involves a weighted Pearson correlation coefficient, which is weighted by Gaussian kernels for wind direction an the radial variable (by default wind speed). More weight is assigned to values close to a wind speed-direction interval. Kernel weighting is used to ensure that all data are used rather than relying on the potentially small number of values in a wind speed-direction interval.
- When `statistic = "Spearman"`, the Spearman correlation coefficient is calculated for *two* pollutants. The calculation involves a weighted

Spearman correlation coefficient, which is weighted by Gaussian kernels for wind direction an the radial variable (by default wind speed). More weight is assigned to values close to a wind speed-direction interval. Kernel weighting is used to ensure that all data are used rather than relying on the potentially small number of values in a wind speed-direction interval.

- `"robust_slope"` is another option for pair-wise statistics and `"quantile.slope"`, which uses quantile regression to estimate the slope for a particular quantile level (see also `tau` for setting the quantile level).
- `"york_slope"` is another option for pair-wise statistics which uses the *York regression method* to estimate the slope. In this method the uncertainties in `x` and `y` are used in the determination of the slope. The uncertainties are provided by `x_error` and `y_error` — see below.

`exclude.missing` Setting this option to `TRUE` (the default) removes points from the plot that are too far from the original data. The smoothing routines will produce predictions at points where no data exist i.e. they predict. By removing the points too far from the original data produces a plot where it is clear where the original data lie. If set to `FALSE` missing data will be interpolated.

`uncertainty` Should the uncertainty in the calculated surface be shown? If `TRUE` three plots are produced on the same scale showing the predicted surface together with the estimated lower and upper uncertainties at the 95% confidence interval. Calculating the uncertainties is useful to understand whether features are real or not. For example, at high wind speeds where there are few data there is greater uncertainty over the predicted values. The uncertainties are calculated using the GAM and weighting is done by the frequency of measurements in each wind speed-direction bin. Note that if uncertainties are calculated then the type is set to "default".

`percentile` If `statistic = "percentile"` then `percentile` is used, expressed from 0 to 100. Note that the percentile value is calculated in the wind speed, wind direction 'bins'. For this reason it can also be useful to set `min.bin` to ensure there are a sufficient number of points available to estimate a percentile. See `quantile` for more details of how percentiles are calculated. `percentile` is also used for the Conditional Probability Function (CPF) plots. `percentile` can be of length two, in which case the percentile *interval* is considered for use with CPF. For example, `percentile = c(90, 100)` will plot the CPF for concentrations between the 90 and 100th percentiles. Percentile intervals can be useful for identifying specific sources. In addition, `percentile` can also be of length 3. The third value is the 'trim' value to be applied. When calculating percentile intervals many can cover very low values where there is no useful information. The trim value ensures that values greater than or equal to the trim * mean value are considered *before* the percentile intervals are calculated. The effect is to extract more detail from many source signatures. See the manual for examples. Finally, if the trim value is less than zero the percentile range is interpreted as absolute concentration values and subsetting is carried out directly.

`weights` At the edges of the plot there may only be a few data points in each wind speed-direction interval, which could in some situations distort the

plot if the concentrations are high. `weights` applies a weighting to reduce their influence. For example and by default if only a single data point exists then the weighting factor is 0.25 and for two points 0.5. To not apply any weighting and use the data as is, use `weights = c(1, 1, 1)`.
An alternative to down-weighting these points they can be removed altogether using `min.bin`.

`min.bin` The minimum number of points allowed in a wind speed/wind direction bin. The default is 1. A value of two requires at least 2 valid records in each bin an so on; bins with less than 2 valid records are set to NA. Care should be taken when using a value > 1 because of the risk of removing real data points. It is recommended to consider your data with care. Also, the `polarFreq` function can be of use in such circumstances.

`mis.col` When `min.bin` is > 1 it can be useful to show where data are removed on the plots. This is done by shading the missing data in `mis.col`. To not highlight missing data when `min.bin` > 1 choose `mis.col = "transparent"`.

`angle.scale` Sometimes the placement of the scale may interfere with an interesting feature. The user can therefore set `angle.scale` to any value between 0 and 360 degrees to mitigate such problems. For example `angle.scale = 45` will draw the scale heading in a NE direction.

`units` The units shown on the polar axis scale.

`force.positive` The default is `TRUE`. Sometimes if smoothing data with steep gradients it is possible for predicted values to be negative. `force.positive = TRUE` ensures that predictions remain positive. This is useful for several reasons. First, with lots of missing data more interpolation is needed and this can result in artefacts because the predictions are too far from the original data. Second, if it is known beforehand that the data are all positive, then this option carries that assumption through to the prediction. The only likely time where setting `force.positive = FALSE` would be if background concentrations were first subtracted resulting in data that is legitimately negative. For the vast majority of situations it is expected that the user will not need to alter the default option.

`k` This is the smoothing parameter used by the `gam` function in package `mgcv`. Typically, value of around 100 (the default) seems to be suitable and will resolve important features in the plot. The most appropriate choice of `k` is problem-dependent; but extensive testing of polar plots for many different problems suggests a value of `k` of about 100 is suitable. Setting `k` to higher values will not tend to affect the surface predictions by much but will add to the computation time. Lower values of `k` will increase smoothing. Sometimes with few data to plot `polarPlot` will fail. Under these circumstances it can be worth lowering the value of `k`.

`normalise` If `TRUE` concentrations are normalised by dividing by their mean value. This is done *after* fitting the smooth surface. This option is particularly useful if one is interested in the patterns of concentrations for several pollutants on different scales e.g. NOx and CO. Often useful if more than one `pollutant` is chosen.

`key.header` Adds additional text/labels to the scale key. For example, passing the options `key.header = "header"`, `key.footer = "footer1"` adds addition text above and below the scale key. These arguments are passed to

drawOpenKey via quickText, applying the auto.text argument, to handle formatting.

key.footer see key.footer.

key.position Location where the scale key is to plotted. Allowed arguments currently include "top", "right", "bottom" and "left".

auto.text Either TRUE (default) or FALSE. If TRUE titles and axis labels will automatically try and format pollutant names and units properly e.g. by subscripting the '2' in NO2.

ws_spread The value of sigma used for Gaussian kernel weighting of wind speed when statistic = "nwr" or when correlation and regression statistics are used such as *r*. Default is 0.5.

wd_spread The value of sigma used for Gaussian kernel weighting of wind direction when statistic = "nwr" or when correlation and regression statistics are used such as *r*. Default is 4.

x_error The x error / uncertainty used when statistic = "york_slope".

y_error The y error / uncertainty used when statistic = "york_slope".

kernel Type of kernel used for the weighting procedure for when correlation or regression techniques are used. Only "gaussian" is supported but this may be enhanced in the future.

formula.label When pair-wise statistics such as regression slopes are calculated and plotted, should a formula label be displayed?

tau The quantile to be estimated when statistic is set to "quantile.slope". Default is 0.5 which is equal to the median and will be ignored if "quantile.slope" is not used.

Value

a ggplot2 plot with a ggmap basemap

Further customisation using ggplot2

As the outputs of the static directional analysis functions are ggplot2 figures, further customisation is possible using functions such as ggplot2::theme(), ggplot2::guides() and ggplot2::labs().

If multiple pollutants are specified, subscripting (e.g., the "x" in "NOx") is achieved using the ggtext package. Therefore if you choose to override the plot theme, it is recommended to use [ggplot2::theme()] and [ggtext::element_markdown()] to define the strip.text parameter.

When arguments like limits, percentile or breaks are defined, a legend is automatically added to the figure. Legends can be removed using ggplot2::theme(legend.position = "none"), or further customised using ggplot2::guides() and either color = ggplot2::guide_colourbar() for continuous legends or fill = ggplot2::guide_legend() for discrete legends.

See Also

the original openair::polarPlot()

polarMap() for the interactive leaflet equivalent of polarMapStatic()

Other static directional analysis maps: annulusMapStatic(), diffMapStatic(), freqMapStatic(), percentileMapStatic(), pollroseMapStatic(), windroseMapStatic()

polar_data *Example data for polar mapping functions*

Description

The polar_data dataset is provided as an example dataset as part of the openairmaps package. The dataset contains hourly measurements of air pollutant concentrations, location and meteorological data.

Format

Data frame with example data from four sites in London in 2009.

date The date and time of the measurement

nox, no2, pm2.5, pm10 Pollutant concentrations

site The site name. Useful for use with the popup and label arguments in openairmaps functions.

latitude, longitude Decimal latitude and longitude of the sites.

site.type Site type of the site (either "Urban Traffic" or "Urban Background").

wd Wind direction, in degrees from North, as a numeric vector.

ws Wind speed, in m/s, as numeric vector.

visibility The visibility in metres.

air_temp Air temperature in degrees Celcius.

Details

polar_data is supplied with the openairmaps package as an example dataset for use with documented examples.

Source

polar_data was compiled from data using the openair::importAURN() function from the openair package with meteorological data from the worldmet package.

Examples

```
# basic structure
head(polar_data)
```

pollroseMap *Pollution rose plots on interactive leaflet maps*

Description

pollroseMap() creates a leaflet map using "pollution roses" as markers. Any number of pollutants can be specified using the pollutant argument, and multiple layers of markers can be added and toggled between using control.

Usage

```
pollroseMap(
  data,
  pollutant = NULL,
  statistic = "prop.count",
  breaks = NULL,
  latitude = NULL,
  longitude = NULL,
  control = NULL,
  popup = NULL,
  label = NULL,
  provider = "OpenStreetMap",
  cols = "turbo",
  alpha = 1,
  key = FALSE,
  draw.legend = TRUE,
  collapse.control = FALSE,
  d.icon = 200,
  d.fig = 3.5,
  type = deprecated(),
  ...
)
```

Arguments

data	A data frame. The data frame must contain the data to plot the directional analysis marker, which includes wind speed (ws), wind direction (wd), and the column representing the concentration of a pollutant. In addition, data must include a decimal latitude and longitude.
pollutant	The column name(s) of the pollutant(s) to plot. If multiple pollutants are specified, they can be toggled between using a "layer control" interface.
statistic	The statistic to be applied to each data bin in the plot. Options currently include "prop.count", "prop.mean" and "abs.count". The default "prop.count" sizes bins according to the proportion of the frequency of measurements. Similarly, "prop.mean" sizes bins according to their relative contribution to the mean. "abs.count" provides the absolute count of measurements in each bin.

`breaks` Most commonly, the number of break points. If not specified, each marker will independently break its supplied data at approximately 6 sensible break points. When `breaks` are specified, all markers will use the same break points. Breaks can also be used to set specific break points. For example, the argument `breaks = c(0, 1, 10, 100)` breaks the data into segments <1, 1-10, 10-100, >100.

`latitude, longitude`
The decimal latitude/longitude. If not provided, will be automatically inferred from data by looking for a column named "lat"/"latitude" or "lon"/"lng"/"long"/"longitude" (case-insensitively).

`control` Used for splitting the input data into different groups which can be selected between using a "layer control" interface, passed to the `type` argument of `openair::cutData()`. `control` cannot be used if multiple `pollutant` columns have been provided.

`popup` Columns to be used as the HTML content for marker popups. Popups may be useful to show information about the individual sites (e.g., site names, codes, types, etc.). If a vector of column names are provided they are passed to `buildPopup()` using its default values.

`label` Column to be used as the HTML content for hover-over labels. Labels are useful for the same reasons as popups, though are typically shorter.

`provider` The base map(s) to be used. See http://leaflet-extras.github.io/leaflet-providers/preview/ for a list of all base maps that can be used. If multiple base maps are provided, they can be toggled between using a "layer control" interface. By default, the interface will use the provider names as labels, but users can define their own using a named vector (e.g., `c("Default" = "OpenStreetMap", "Satellite" = "Esri.WorldImagery")`)

`cols` The colours used for plotting. See `openair::openColours()` for more information.

`alpha` The alpha transparency to use for the plotting surface (a value between 0 and 1 with zero being fully transparent and 1 fully opaque).

`key` Should a key for each marker be drawn? Default is `FALSE`.

`draw.legend` When `breaks` are specified, should a shared legend be created at the side of the map? Default is `TRUE`.

`collapse.control`
Should the "layer control" interface be collapsed? Defaults to `FALSE`.

`d.icon` The diameter of the plot on the map in pixels. This will affect the size of the individual polar markers. Alternatively, a vector in the form `c(width, height)` can be provided if a non-circular marker is desired.

`d.fig` The diameter of the plots to be produced using `openair` in inches. This will affect the resolution of the markers on the map. Alternatively, a vector in the form `c(width, height)` can be provided if a non-circular marker is desired.

`type` **[Deprecated]**. Different sites are now automatically detected based on latitude and longitude. Please use `label` and/or `popup` to label different sites.

`...` Arguments passed on to `openair::pollutionRose`

`key.footer` Adds additional text/labels below the scale key. See `key.header` for further information.

`key.position` Location where the scale key is to plotted. Allowed arguments currently include "top", "right", "bottom" and "left".

`paddle` Either `TRUE` or `FALSE`. If `TRUE` plots rose using 'paddle' style spokes. If `FALSE` plots rose using 'wedge' style spokes.

`seg` When `paddle = TRUE`, `seg` determines with width of the segments. For example, `seg = 0.5` will produce segments 0.5 * `angle`.

`normalise` If `TRUE` each wind direction segment is normalised to equal one. This is useful for showing how the concentrations (or other parameters) contribute to each wind sector when the proportion of time the wind is from that direction is low. A line showing the probability that the wind directions is from a particular wind sector is also shown.

Value

A leaflet object.

See Also

the original `openair::pollutionRose()`

`pollroseMapStatic()` for the static ggmap equivalent of `pollroseMap()`

Other interactive directional analysis maps: `annulusMap()`, `diffMap()`, `freqMap()`, `percentileMap()`, `polarMap()`, `windroseMap()`

Examples

```
## Not run:
pollroseMap(polar_data,
  pollutant = "nox",
  statistic = "prop.count",
  provider = "Stamen.Toner"
)

## End(Not run)
```

---

pollroseMapStatic *Percentile roses on a static ggmap*

Description

`pollroseMapStatic()` creates a `ggplot2` map using percentile roses as markers. As this function returns a `ggplot2` object, further customisation can be achieved using functions like `ggplot2::theme()` and `ggplot2::guides()`.

Usage

```
pollroseMapStatic(
  data,
  pollutant = NULL,
  ggmap,
  statistic = "prop.count",
  breaks = NULL,
  facet = NULL,
  latitude = NULL,
  longitude = NULL,
  cols = "turbo",
  alpha = 1,
  key = FALSE,
  facet.nrow = NULL,
  d.icon = 150,
  d.fig = 3,
  ...
)
```

Arguments

`data`	A data frame. The data frame must contain the data to plot the directional analysis marker, which includes wind speed (`ws`), wind direction (`wd`), and the column representing the concentration of a pollutant. In addition, `data` must include a decimal latitude and longitude.
`pollutant`	The column name(s) of the pollutant(s) to plot. If multiple pollutants are specified, they will each form part of a separate panel.
`ggmap`	A ggmap object obtained using `ggmap::get_map()` or a similar function to use as the basemap.
`statistic`	The `statistic` to be applied to each data bin in the plot. Options currently include "prop.count", "prop.mean" and "abs.count". The default "prop.count" sizes bins according to the proportion of the frequency of measurements. Similarly, "prop.mean" sizes bins according to their relative contribution to the mean. "abs.count" provides the absolute count of measurements in each bin.
`breaks`	Most commonly, the number of break points. If not specified, each marker will independently break its supplied data at approximately 6 sensible break points. When `breaks` are specified, all markers will use the same break points. Breaks can also be used to set specific break points. For example, the argument `breaks = c(0, 1, 10, 100)` breaks the data into segments <1, 1-10, 10-100, >100.
`facet`	Used for splitting the input data into different panels, passed to the `type` argument of `openair::cutData()`. `facet` cannot be used if multiple `pollutant` columns have been provided.
`latitude, longitude`	The decimal latitude/longitude. If not provided, will be automatically inferred from data by looking for a column named "lat"/"latitude" or "lon"/"lng"/"long"/"longitude" (case-insensitively).

cols — The colours used for plotting. See openair::openColours() for more information.

alpha — The alpha transparency to use for the plotting surface (a value between 0 and 1 with zero being fully transparent and 1 fully opaque).

key — Should a key for each marker be drawn? Default is FALSE.

facet.nrow — Passed to the nrow argument of ggplot2::facet_wrap().

d.icon — The diameter of the plot on the map in pixels. This will affect the size of the individual polar markers. Alternatively, a vector in the form c(width, height) can be provided if a non-circular marker is desired.

d.fig — The diameter of the plots to be produced using openair in inches. This will affect the resolution of the markers on the map. Alternatively, a vector in the form c(width, height) can be provided if a non-circular marker is desired.

... — Arguments passed on to openair::pollutionRose

key.footer Adds additional text/labels below the scale key. See key.header for further information.

key.position Location where the scale key is to plotted. Allowed arguments currently include "top", "right", "bottom" and "left".

paddle Either TRUE or FALSE. If TRUE plots rose using 'paddle' style spokes. If FALSE plots rose using 'wedge' style spokes.

seg When paddle = TRUE, seg determines with width of the segments. For example, seg = 0.5 will produce segments 0.5 * angle.

normalise If TRUE each wind direction segment is normalised to equal one. This is useful for showing how the concentrations (or other parameters) contribute to each wind sector when the proportion of time the wind is from that direction is low. A line showing the probability that the wind directions is from a particular wind sector is also shown.

Value

a ggplot2 plot with a ggmap basemap

Further customisation using ggplot2

As the outputs of the static directional analysis functions are ggplot2 figures, further customisation is possible using functions such as ggplot2::theme(), ggplot2::guides() and ggplot2::labs().

If multiple pollutants are specified, subscripting (e.g., the "x" in "NOx") is achieved using the ggtext package. Therefore if you choose to override the plot theme, it is recommended to use [ggplot2::theme()] and [ggtext::element_markdown()] to define the strip.text parameter.

When arguments like limits, percentile or breaks are defined, a legend is automatically added to the figure. Legends can be removed using ggplot2::theme(legend.position = "none"), or further customised using ggplot2::guides() and either color = ggplot2::guide_colourbar() for continuous legends or fill = ggplot2::guide_legend() for discrete legends.

See Also

the original openair::pollutionRose()

pollroseMap() for the interactive leaflet equivalent of pollroseMapStatic()

Other static directional analysis maps: annulusMapStatic(), diffMapStatic(), freqMapStatic(), percentileMapStatic(), polarMapStatic(), windroseMapStatic()

---

quickTextHTML *Automatic text formatting for openairmaps*

Description

Workhorse function that automatically applies routine text formatting to common pollutant names which may be used in the HTML widgets produced by openairmaps.

Usage

```
quickTextHTML(text)
```

Arguments

text	A character vector.

Details

quickTextHTML() is routine formatting lookup table. It screens the supplied character vector text and automatically applies formatting to any recognised character sub-series to properly render in HTML.

Value

The function returns a character vector for HTML evaluation.

Author(s)

Jack Davison.

See Also

The original openair::quickText(), useful for non-HTML/static maps and plots

Examples

```
labs <- c("no2", "o3", "so2")
quickTextHTML(labs)
```

searchNetwork *Geographically search the air quality networks made available by* `openair::importMeta()`

Description

While `networkMap()` visualises entire UK air quality networks, `searchNetwork()` can subset specific networks to find air quality sites near to a specific site of interest (for example, the location of known industrial activity, or the centroid of a specific urban area).

Usage

```
searchNetwork(
  lat,
  lng,
  source = "aurn",
  year = NULL,
  site_type = NULL,
  variable = NULL,
  max_dist = NULL,
  n = NULL,
  map = TRUE
)
```

Arguments

`lat, lng`	The latitude and longitude for the location of interest.
`source`	One or more air quality networks for which data is available through openair. Available networks include: • `"aurn"`, The UK Automatic Urban and Rural Network. • `"aqe"`, The Air Quality England Network. • `"saqn"`, The Scottish Air Quality Network. • `"waqn"`, The Welsh Air Quality Network. • `"ni"`, The Northern Ireland Air Quality Network. • `"local"`, Locally managed air quality networks in England. • `"kcl"`, King's College London networks. • `"europe"`, European AirBase/e-reporting data. There are two additional options provided for convenience: • `"ukaq"` will return metadata for all networks for which data is imported by `importUKAQ()` (i.e., AURN, AQE, SAQN, WAQN, NI, and the local networks). • `"all"` will import all available metadata (i.e., `"ukaq"` plus `"kcl"` and `"europe"`).
`year`	If a single year is selected, only sites that were open at some point in that year are returned. If `all = TRUE` only sites that measured a particular pollutant in that year are returned. Year can also be a sequence e.g. `year = 2010:2020` or of

	length 2 e.g. `year = c(2010, 2020)`, which will return only sites that were open over the duration. Note that `year` is ignored when the `source` is either `"kcl"` or `"europe"`.
`site_type`	Optional. One or more site types with which to subset the site metadata. For example, `site_type = "urban background"` will only search urban background sites.
`variable`	Optional. One or more variables of interest with which to subset the site metadata. For example, `variable = c("pm10", "co")` will search sites that measure PM10 and/or CO.
`max_dist`	Optional. A maximum distance from the location of interest in kilometres.
`n`	Optional. The maximum number of sites to return.
`map`	If `TRUE`, the default, `searchNetwork()` will return a `leaflet` map. If `FALSE`, it will instead return a tibble.

Details

Data subsetting progresses in the order in which the arguments are given; first `source` and `year`, then `site_type` and `variable`, then `max_dist`, and finally `n`.

Value

Either a tibble or `leaflet` map.

See Also

Other uk air quality network mapping functions: `networkMap()`

Examples

```
## Not run:
# get all AURN sites open in 2020 within 20 km of Buckingham Palace
palace <- convertPostcode("SW1A1AA")
searchNetwork(lat = palace$lat, lng = palace$lng, max_dist = 20, year = 2020)

## End(Not run)
```

`trajLevelMap` *Trajectory level plots in* `leaflet`

Description

This function plots back trajectories on a `leaflet` map. This function requires that data are imported using the `openair::importTraj()` function.

Usage

```
trajLevelMap(
  data,
  longitude = "lon",
  latitude = "lat",
  pollutant,
  control = "default",
  smooth = FALSE,
  statistic = "frequency",
  percentile = 90,
  lon.inc = 1,
  lat.inc = 1,
  min.bin = 1,
  .combine = NA,
  sigma = 1.5,
  cols = "default",
  alpha = 0.5,
  tile.border = NA,
  provider = "OpenStreetMap"
)
```

Arguments

`data`	Data frame, the result of importing a trajectory file using `openair::importTraj()`.
`latitude, longitude`	The decimal latitude/longitude.
`pollutant`	Pollutant to be plotted. By default the trajectory height is used.
`control`	Used for splitting the trajectories into different groups which can be selected between using a "layer control" menu. Passed to `openair::cutData()`.
`smooth`	Should the trajectory surface be smoothed? Defaults to `FALSE`. Note that, when `smooth = TRUE`, no popup information will be available.
`statistic`	Statistic to use for `trajLevel()`. By default, the function will plot the trajectory frequencies (`statistic = "frequency"`). As an alternative way of viewing trajectory frequencies, the argument `method = "hexbin"` can be used. In this case hexagonal binning of the trajectory *points* (i.e., a point every three hours along each back trajectory). The plot then shows the trajectory frequencies uses hexagonal binning. There are also various ways of plotting concentrations. It is possible to set `statistic = "difference"`. In this case trajectories where the associated concentration is greater than `percentile` are compared with the the full set of trajectories to understand the differences in frequencies of the origin of air masses. The comparison is made by comparing the percentage change in gridded frequencies. For example, such a plot could show that the top 10\ tend to originate from air-mass origins to the east. If `statistic = "pscf"` then a Potential Source Contribution Function map is produced. This statistic method interacts with `percentile`.

If `statistic = "cwt"` then concentration weighted trajectories are plotted.

If `statistic = "sqtba"` then Simplified Quantitative Transport Bias Analysis is undertaken. This statistic method interacts with `.combine` and `sigma`.

`percentile` The percentile concentration of `pollutant` against which the all trajectories are compared.

`lon.inc, lat.inc`
The longitude and latitude intervals to be used for binning data.

`min.bin` The minimum number of unique points in a grid cell. Counts below `min.bin` are set as missing.

`.combine` When statistic is "SQTBA" it is possible to combine lots of receptor locations to derive a single map. `.combine` identifies the column that differentiates different sites (commonly a column named `"site"`). Note that individual site maps are normalised first by dividing by their mean value.

`sigma` For the SQTBA approach `sigma` determines the amount of back trajectory spread based on the Gaussian plume equation. Values in the literature suggest 5.4 km after one hour. However, testing suggests lower values reveal source regions more effectively while not introducing too much noise.

`cols` Colours to be used for plotting. Options include "default", "increment", "heat", "turbo" and `RColorBrewer` colours — see the `openair::openColours()` function for more details. For user defined the user can supply a list of colour names recognised by R (type `grDevices::colours()` to see the full list). An example would be `cols = c("yellow", "green", "blue")`.

`alpha` Opacity of the tiles. Must be between `0` and `1`.

`tile.border` Colour to use for the border of binned tiles. Defaults to `NA`, which draws no border.

`provider` The base map to be used. See `http://leaflet-extras.github.io/leaflet-providers/preview/` for a list of all base maps that can be used.

Value

A leaflet object.

See Also

the original `openair::trajLevel()`

`trajLevelMapStatic()` for the static `ggplot2` equivalent of `trajLevelMap()`

Other interactive trajectory maps: `trajMap()`

Examples

```
## Not run:
trajLevelMap(traj_data, pollutant = "pm2.5", statistic = "pscf", min.bin = 10)

## End(Not run)
```

`trajLevelMapStatic` *Trajectory level plots in* `ggplot2`

Description

[Experimental]

This function plots back trajectories on a `ggplot2` map. This function requires that data are imported using the `openair::importTraj()` function. It is a `ggplot2` implementation of `openair::trajLevel()` with many of the same arguments, which should be more flexible for post-hoc changes.

Usage

```
trajLevelMapStatic(
  data,
  longitude = "lon",
  latitude = "lat",
  pollutant,
  facet = "default",
  smooth = FALSE,
  statistic = "frequency",
  percentile = 90,
  lon.inc = 1,
  lat.inc = 1,
  min.bin = 1,
  .combine = NA,
  sigma = 1.5,
  alpha = 0.5,
  tile.border = NA,
  xlim = NULL,
  ylim = NULL,
  crs = sf::st_crs(4326),
  map = TRUE,
  map.fill = "grey85",
  map.colour = "grey75",
  map.alpha = 0.8,
  map.lwd = 0.5,
  map.lty = 1,
  ...
)
```

Arguments

`data`	Data frame, the result of importing a trajectory file using `openair::importTraj()`.
`latitude, longitude`	The decimal latitude/longitude.
`pollutant`	Pollutant to be plotted. By default the trajectory height is used.

`facet` Used for splitting the trajectories into different panels. Passed to `openair::cutData()`.

`smooth` Should the trajectory surface be smoothed? Defaults to `FALSE`. Note that smoothing may cause the plot to render slower, so consider setting `crs` to `sf::st_crs(4326)` or `NULL`.

`statistic` Statistic to use for `trajLevel()`. By default, the function will plot the trajectory frequencies (`statistic = "frequency"`). As an alternative way of viewing trajectory frequencies, the argument `method = "hexbin"` can be used. In this case hexagonal binning of the trajectory *points* (i.e., a point every three hours along each back trajectory). The plot then shows the trajectory frequencies uses hexagonal binning.

There are also various ways of plotting concentrations.

It is possible to set `statistic = "difference"`. In this case trajectories where the associated concentration is greater than `percentile` are compared with the the full set of trajectories to understand the differences in frequencies of the origin of air masses. The comparison is made by comparing the percentage change in gridded frequencies. For example, such a plot could show that the top 10\ tend to originate from air-mass origins to the east.

If `statistic = "pscf"` then a Potential Source Contribution Function map is produced. This statistic method interacts with `percentile`.

If `statistic = "cwt"` then concentration weighted trajectories are plotted.

If `statistic = "sqtba"` then Simplified Quantitative Transport Bias Analysis is undertaken. This statistic method interacts with `.combine` and `sigma`.

`percentile` The percentile concentration of `pollutant` against which the all trajectories are compared.

`lon.inc, lat.inc`
The longitude and latitude intervals to be used for binning data.

`min.bin` The minimum number of unique points in a grid cell. Counts below `min.bin` are set as missing.

`.combine` When statistic is "SQTBA" it is possible to combine lots of receptor locations to derive a single map. `.combine` identifies the column that differentiates different sites (commonly a column named `"site"`). Note that individual site maps are normalised first by dividing by their mean value.

`sigma` For the SQTBA approach `sigma` determines the amount of back trajectory spread based on the Gaussian plume equation. Values in the literature suggest 5.4 km after one hour. However, testing suggests lower values reveal source regions more effectively while not introducing too much noise.

`alpha` Opacity of the tiles. Must be between `0` and `1`.

`tile.border` Colour to use for the border of binned tiles. Defaults to `NA`, which draws no border.

`xlim, ylim` The x- and y-limits of the plot. If `NULL`, limits will be estimated based on the lat/lon ranges of the input data.

`crs` The coordinate reference system (CRS) into which all data should be projected before plotting. Defaults to latitude/longitude (`sf::st_crs(4326)`).

`map` Should a base map be drawn? Defaults to `TRUE`.

map.fill Colour to use to fill the polygons of the base map (see colors()).

map.colour Colour to use for the polygon borders of the base map (see colors()).

map.alpha Transparency of the base map polygons. Must be between 0 (fully transparent) and 1 (fully opaque).

map.lwd Line width of the base map polygon borders.

map.lty Line type of the base map polygon borders. See ggplot2::scale_linetype() for common examples.

... Arguments passed on to ggplot2::coord_sf

expand If TRUE, the default, adds a small expansion factor to the limits to ensure that data and axes don't overlap. If FALSE, limits are taken exactly from the data or xlim/ylim.

datum CRS that provides datum to use when generating graticules.

label_graticule Character vector indicating which graticule lines should be labeled where. Meridians run north-south, and the letters "N" and "S" indicate that they should be labeled on their north or south end points, respectively. Parallels run east-west, and the letters "E" and "W" indicate that they should be labeled on their east or west end points, respectively. Thus, label_graticule = "SW" would label meridians at their south end and parallels at their west end, whereas label_graticule = "EW" would label parallels at both ends and meridians not at all. Because meridians and parallels can in general intersect with any side of the plot panel, for any choice of label_graticule labels are not guaranteed to reside on only one particular side of the plot panel. Also, label_graticule can cause labeling artifacts, in particular if a graticule line coincides with the edge of the plot panel. In such circumstances, label_axes will generally yield better results and should be used instead.

This parameter can be used alone or in combination with label_axes.

label_axes Character vector or named list of character values specifying which graticule lines (meridians or parallels) should be labeled on which side of the plot. Meridians are indicated by "E" (for East) and parallels by "N" (for North). Default is "--EN", which specifies (clockwise from the top) no labels on the top, none on the right, meridians on the bottom, and parallels on the left. Alternatively, this setting could have been specified with list(bottom = "E", left = "N").

This parameter can be used alone or in combination with label_graticule.

lims_method Method specifying how scale limits are converted into limits on the plot region. Has no effect when default_crs = NULL. For a very nonlinear CRS (e.g., a perspective centered around the North pole), the available methods yield widely differing results, and you may want to try various options. Methods currently implemented include "cross" (the default), "box", "orthogonal", and "geometry_bbox". For method "cross", limits along one direction (e.g., longitude) are applied at the midpoint of the other direction (e.g., latitude). This method avoids excessively large limits for rotated coordinate systems but means that sometimes limits need to be expanded a little further if extreme data points are to be included in the final plot region. By contrast, for method "box", a box is generated out

of the limits along both directions, and then limits in projected coordinates are chosen such that the entire box is visible. This method can yield plot regions that are too large. Finally, method "orthogonal" applies limits separately along each axis, and method "geometry_bbox" ignores all limit information except the bounding boxes of any objects in the geometry aesthetic.

ndiscr Number of segments to use for discretising graticule lines; try increasing this number when graticules look incorrect.

default Is this the default coordinate system? If FALSE (the default), then replacing this coordinate system with another one creates a message alerting the user that the coordinate system is being replaced. If TRUE, that warning is suppressed.

clip Should drawing be clipped to the extent of the plot panel? A setting of "on" (the default) means yes, and a setting of "off" means no. In most cases, the default of "on" should not be changed, as setting clip = "off" can cause unexpected results. It allows drawing of data points anywhere on the plot, including in the plot margins. If limits are set via xlim and ylim and some data points fall outside those limits, then those data points may show up in places such as the axes, the legend, the plot title, or the plot margins.

Value

A ggplot2 plot

See Also

the original openair::trajLevel()

trajLevelMap() for the interactive leaflet equivalent of trajLevelMapStatic()

Other static trajectory maps: trajMapStatic()

---

trajMap *Trajectory line plots in* leaflet

Description

This function plots back trajectories on a leaflet map. This function requires that data are imported using the openair::importTraj() function. Options are provided to colour the individual trajectories (e.g., by pollutant concentrations) or create "layer control" menus to show/hide different layers.

Usage

```
trajMap(
  data,
  longitude = "lon",
```

```
  latitude = "lat",
  colour,
  control = "default",
  cols = "default",
  alpha = 0.5,
  npoints = 12,
  provider = "OpenStreetMap",
  collapse.control = FALSE
)
```

Arguments

`data`	Data frame, the result of importing a trajectory file using `openair::importTraj()`.
`latitude, longitude`	The decimal latitude/longitude.
`colour`	Column to be used for colouring each trajectory. This column may be numeric, character or factor. This will commonly be a pollutant concentration which has been joined (e.g., by `dplyr::left_join()`) to the trajectory data by "date".
`control`	Used for splitting the trajectories into different groups which can be selected between using a "layer control" menu. Passed to `openair::cutData()`.
`cols`	Colours to be used for plotting. Options include "default", "increment", "heat", "turbo" and `RColorBrewer` colours — see the `openair::openColours()` function for more details. For user defined the user can supply a list of colour names recognised by R (type `grDevices::colours()` to see the full list). An example would be `cols = c("yellow", "green", "blue")`. If the `"colour"` argument was not used, a single colour can be named which will be used consistently for all lines/points (e.g., `cols = "red"`).
`alpha`	Opacity of lines/points. Must be between `0` and `1`.
`npoints`	A dot is placed every `npoints` along each full trajectory. For hourly back trajectories points are plotted every `npoints` hours. This helps to understand where the air masses were at particular times and get a feel for the speed of the air (points closer together correspond to slower moving air masses). Defaults to `12`.
`provider`	The base map to be used. See `http://leaflet-extras.github.io/leaflet-providers/preview/` for a list of all base maps that can be used.
`collapse.control`	Should the "layer control" interface be collapsed? Defaults to `FALSE`.

Value

A leaflet object.

See Also

the original `openair::trajPlot()`

`trajMapStatic()` for the static `ggplot2` equivalent of `trajMap()`

Other interactive trajectory maps: `trajLevelMap()`

Examples

```
## Not run:
trajMap(traj_data, colour = "nox")

## End(Not run)
```

`trajMapStatic` *Trajectory line plots in* `ggplot2`

Description

[Experimental]

This function plots back trajectories using `ggplot2`. The function requires that data are imported using `openair::importTraj()`. It is a `ggplot2` implementation of `openair::trajPlot()` with many of the same arguments, which should be more flexible for post-hoc changes.

Usage

```
trajMapStatic(
  data,
  colour = "height",
  facet = NULL,
  group = NULL,
  longitude = "lon",
  latitude = "lat",
  npoints = 12,
  xlim = NULL,
  ylim = NULL,
  crs = sf::st_crs(3812),
  origin = TRUE,
  map = TRUE,
  map.fill = "grey85",
  map.colour = "grey75",
  map.alpha = 0.8,
  map.lwd = 0.5,
  map.lty = 1,
  ...
)
```

Arguments

`data`	Data frame, the result of importing a trajectory file using `openair::importTraj()`.
`colour`	Column to be used for colouring each trajectory. This column may be numeric, character or factor. This will commonly be a pollutant concentration which has been joined (e.g., by `dplyr::left_join()`) to the trajectory data by "date".

facet Used for splitting the trajectories into different panels. Passed to openair::cutData().

group By default, trajectory paths are distinguished using the arrival date. group allows for additional columns to be used (e.g., "receptor").

latitude, longitude
The decimal latitude/longitude.

npoints A dot is placed every npoints along each full trajectory. For hourly back trajectories points are plotted every npoints hours. This helps to understand where the air masses were at particular times and get a feel for the speed of the air (points closer together correspond to slower moving air masses). Defaults to 12.

xlim, ylim The x- and y-limits of the plot. If NULL, limits will be estimated based on the lat/lon ranges of the input data.

crs The coordinate reference system (CRS) into which all data should be projected before plotting. Defaults to the Lambert projection (sf::st_crs(3812)). Alternatively, can be set to NULL, which will typically render the map quicker but may cause countries far from the equator or large areas to appear distorted.

origin Should the receptor point be marked with a circle? Defaults to TRUE.

map Should a base map be drawn? Defaults to TRUE.

map.fill Colour to use to fill the polygons of the base map (see colors()).

map.colour Colour to use for the polygon borders of the base map (see colors()).

map.alpha Transparency of the base map polygons. Must be between 0 (fully transparent) and 1 (fully opaque).

map.lwd Line width of the base map polygon borders.

map.lty Line type of the base map polygon borders. See ggplot2::scale_linetype() for common examples.

... Arguments passed on to ggplot2::coord_sf

expand If TRUE, the default, adds a small expansion factor to the limits to ensure that data and axes don't overlap. If FALSE, limits are taken exactly from the data or xlim/ylim.

datum CRS that provides datum to use when generating graticules.

label_graticule Character vector indicating which graticule lines should be labeled where. Meridians run north-south, and the letters "N" and "S" indicate that they should be labeled on their north or south end points, respectively. Parallels run east-west, and the letters "E" and "W" indicate that they should be labeled on their east or west end points, respectively. Thus, label_graticule = "SW" would label meridians at their south end and parallels at their west end, whereas label_graticule = "EW" would label parallels at both ends and meridians not at all. Because meridians and parallels can in general intersect with any side of the plot panel, for any choice of label_graticule labels are not guaranteed to reside on only one particular side of the plot panel. Also, label_graticule can cause labeling artifacts, in particular if a graticule line coincides with the edge of the plot panel. In such circumstances, label_axes will generally yield better results and should be used instead.
This parameter can be used alone or in combination with label_axes.

`label_axes` Character vector or named list of character values specifying which graticule lines (meridians or parallels) should be labeled on which side of the plot. Meridians are indicated by `"E"` (for East) and parallels by `"N"` (for North). Default is `"--EN"`, which specifies (clockwise from the top) no labels on the top, none on the right, meridians on the bottom, and parallels on the left. Alternatively, this setting could have been specified with `list(bottom = "E", left = "N")`.
This parameter can be used alone or in combination with `label_graticule`.

`lims_method` Method specifying how scale limits are converted into limits on the plot region. Has no effect when `default_crs = NULL`. For a very nonlinear CRS (e.g., a perspective centered around the North pole), the available methods yield widely differing results, and you may want to try various options. Methods currently implemented include `"cross"` (the default), `"box"`, `"orthogonal"`, and `"geometry_bbox"`. For method `"cross"`, limits along one direction (e.g., longitude) are applied at the midpoint of the other direction (e.g., latitude). This method avoids excessively large limits for rotated coordinate systems but means that sometimes limits need to be expanded a little further if extreme data points are to be included in the final plot region. By contrast, for method `"box"`, a box is generated out of the limits along both directions, and then limits in projected coordinates are chosen such that the entire box is visible. This method can yield plot regions that are too large. Finally, method `"orthogonal"` applies limits separately along each axis, and method `"geometry_bbox"` ignores all limit information except the bounding boxes of any objects in the `geometry` aesthetic.

`ndiscr` Number of segments to use for discretising graticule lines; try increasing this number when graticules look incorrect.

`default` Is this the default coordinate system? If `FALSE` (the default), then replacing this coordinate system with another one creates a message alerting the user that the coordinate system is being replaced. If `TRUE`, that warning is suppressed.

`clip` Should drawing be clipped to the extent of the plot panel? A setting of `"on"` (the default) means yes, and a setting of `"off"` means no. In most cases, the default of `"on"` should not be changed, as setting `clip = "off"` can cause unexpected results. It allows drawing of data points anywhere on the plot, including in the plot margins. If limits are set via `xlim` and `ylim` and some data points fall outside those limits, then those data points may show up in places such as the axes, the legend, the plot title, or the plot margins.

Value

a `ggplot2` plot

See Also

the original `openair::trajPlot()`

`trajMap()` for the interactive `leaflet` equivalent of `trajMapStatic()`

Other static trajectory maps: `trajLevelMapStatic()`

Examples

```
## Not run:
# colour by height
trajMapStatic(traj_data) +
  ggplot2::scale_color_gradientn(colors = openair::openColours())

# colour by PM10, log transform scale
trajMapStatic(traj_data, colour = "pm10") +
  ggplot2::scale_color_viridis_c(trans = "log10") +
  ggplot2::labs(color = openair::quickText("PM10"))

# color by PM2.5, lat/lon projection
trajMapStatic(traj_data, colour = "pm2.5", crs = sf::st_crs(4326)) +
  ggplot2::scale_color_viridis_c(option = "turbo") +
  ggplot2::labs(color = openair::quickText("PM2.5"))

## End(Not run)
```

---

traj_data *Example data for trajectory mapping functions*

Description

The `traj_data` dataset is provided as an example dataset as part of the `openairmaps` package. The dataset contains HYSPLIT back trajectory data for air mass parcels arriving in London in 2009. It has been joined with air quality pollutant concentrations from the "London N. Kensington" AURN urban background monitoring site.

Usage

```
traj_data
```

Format

A data frame with 53940 rows and 10 variables:

date The arrival time of the air-mass

receptor The receptor number

year Trajectory year

month Trajectory month

day Trajectory day

hour Trajectory hour

hour.inc Trajectory hour offset from the arrival date

lat Latitude

lon Longitude

height Height of trajectory in m

pressure Pressure of the trajectory in Pa

date2 Date of the trajectory

nox Concentration of oxides of nitrogen (NO + NO2)

no2 Concentration of nitrogen dioxide (NO2)

o3 Concentration of ozone (O3)

pm10 Concentration of particulates (PM10)

pm2.5 Concentration of fine particulates (PM2.5)

Details

`traj_data` is supplied with the `openairmaps` package as an example dataset for use with documented examples.

Source

`traj_data` was compiled from data using the `openair::importTraj()` function from the `openair` package with air quality data from `openair::importAURN()` function.

Examples

```
# basic structure
head(traj_data)
```

`windroseMap` *Wind rose plots on interactive leaflet maps*

Description

`windroseMap()` creates a `leaflet` map using wind roses as markers. Multiple layers of markers can be added and toggled between using `control`.

Usage

```
windroseMap(
  data,
  ws.int = 2,
  breaks = 4,
  latitude = NULL,
  longitude = NULL,
  control = NULL,
  popup = NULL,
  label = NULL,
```

```
  provider = "OpenStreetMap",
  cols = "turbo",
  alpha = 1,
  key = FALSE,
  draw.legend = TRUE,
  collapse.control = FALSE,
  d.icon = 200,
  d.fig = 3.5,
  type = deprecated(),
  ...
)
```

Arguments

`data`	A data frame. The data frame must contain the data to plot a `openair::windRose()`, which includes wind speed (`ws`), and wind direction (`wd`). In addition, `data` must include a decimal latitude and longitude.
`ws.int`	The wind speed interval. Default is 2 m/s but for low met masts with low mean wind speeds a value of 1 or 0.5 m/s may be better.
`breaks`	Most commonly, the number of break points for wind speed in windRose. For windRose and the ws.int default of 2 m/s, the default, 4, generates the break points 2, 4, 6, 8 m/s. Breaks can also be used to set specific break points. For example, the argument breaks = c(0, 1, 10, 100) breaks the data into segments <1, 1-10, 10-100, >100.
`latitude, longitude`	The decimal latitude/longitude. If not provided, will be automatically inferred from data by looking for a column named "lat"/"latitude" or "lon"/"lng"/"long"/"longitude" (case-insensitively).
`control`	Used for splitting the input data into different groups which can be selected between using a "layer control" interface, passed to the `type` argument of `openair::cutData()`. `control` cannot be used if multiple `pollutant` columns have been provided.
`popup`	Columns to be used as the HTML content for marker popups. Popups may be useful to show information about the individual sites (e.g., site names, codes, types, etc.). If a vector of column names are provided they are passed to `buildPopup()` using its default values.
`label`	Column to be used as the HTML content for hover-over labels. Labels are useful for the same reasons as popups, though are typically shorter.
`provider`	The base map(s) to be used. See `http://leaflet-extras.github.io/leaflet-providers/preview/` for a list of all base maps that can be used. If multiple base maps are provided, they can be toggled between using a "layer control" interface. By default, the interface will use the provider names as labels, but users can define their own using a named vector (e.g., `c("Default" = "OpenStreetMap", "Satellite" = "Esri.WorldImagery")`)
`cols`	The colours used for plotting. See `openair::openColours()` for more information.
`alpha`	The alpha transparency to use for the plotting surface (a value between 0 and 1 with zero being fully transparent and 1 fully opaque).

key Should a key for each marker be drawn? Default is FALSE.

draw.legend Should a shared legend be created at the side of the map? Default is TRUE.

collapse.control
Should the "layer control" interface be collapsed? Defaults to FALSE.

d.icon The diameter of the plot on the map in pixels. This will affect the size of the individual polar markers. Alternatively, a vector in the form c(width, height) can be provided if a non-circular marker is desired.

d.fig The diameter of the plots to be produced using openair in inches. This will affect the resolution of the markers on the map. Alternatively, a vector in the form c(width, height) can be provided if a non-circular marker is desired.

type **[Deprecated]**. Different sites are now automatically detected based on latitude and longitude. Please use label and/or popup to label different sites.

... Arguments passed on to openair::windRose

ws Name of the column representing wind speed.

wd Name of the column representing wind direction.

ws2,wd2 The user can supply a second set of wind speed and wind direction values with which the first can be compared. See pollutionRose() for more details.

angle Default angle of "spokes" is 30. Other potentially useful angles are 45 and 10. Note that the width of the wind speed interval may need adjusting using width.

bias.corr When angle does not divide exactly into 360 a bias is introduced in the frequencies when the wind direction is already supplied rounded to the nearest 10 degrees, as is often the case. For example, if angle = 22.5, N, E, S, W will include 3 wind sectors and all other angles will be two. A bias correction can made to correct for this problem. A simple method according to Applequist (2012) is used to adjust the frequencies.

grid.line Grid line interval to use. If NULL, as in default, this is assigned based on the available data range. However, it can also be forced to a specific value, e.g. grid.line = 10. grid.line can also be a list to control the interval, line type and colour. For example grid.line = list(value = 10, lty = 5, col = "purple").

width For paddle = TRUE, the adjustment factor for width of wind speed intervals. For example, width = 1.5 will make the paddle width 1.5 times wider.

seg When paddle = TRUE, seg determines with width of the segments. For example, seg = 0.5 will produce segments 0.5 * angle.

auto.text Either TRUE (default) or FALSE. If TRUE titles and axis labels will automatically try and format pollutant names and units properly, e.g., by subscripting the '2' in NO2.

offset The size of the 'hole' in the middle of the plot, expressed as a percentage of the polar axis scale, default 10.

normalise If TRUE each wind direction segment is normalised to equal one. This is useful for showing how the concentrations (or other parameters) contribute to each wind sector when the proportion of time the wind is from

that direction is low. A line showing the probability that the wind directions is from a particular wind sector is also shown.

`max.freq` Controls the scaling used by setting the maximum value for the radial limits. This is useful to ensure several plots use the same radial limits.

`paddle` Either `TRUE` or `FALSE`. If `TRUE` plots rose using 'paddle' style spokes. If `FALSE` plots rose using 'wedge' style spokes.

`key.header` Adds additional text/labels above the scale key. For example, passing `windRose(mydata, key.header = "ws")` adds the addition text as a scale header. Note: This argument is passed to `drawOpenKey()` via `quickText()`, applying the auto.text argument, to handle formatting.

`key.footer` Adds additional text/labels below the scale key. See `key.header` for further information.

`key.position` Location where the scale key is to plotted. Allowed arguments currently include "top", "right", "bottom" and "left".

`dig.lab` The number of significant figures at which scientific number formatting is used in break point and key labelling. Default 5.

`include.lowest` Logical. If `FALSE` (the default), the first interval will be left exclusive and right inclusive. If `TRUE`, the first interval will be left and right inclusive. Passed to the `include.lowest` argument of `cut()`.

`statistic` The `statistic` to be applied to each data bin in the plot. Options currently include "prop.count", "prop.mean" and "abs.count". The default "prop.count" sizes bins according to the proportion of the frequency of measurements. Similarly, "prop.mean" sizes bins according to their relative contribution to the mean. "abs.count" provides the absolute count of measurements in each bin.

`pollutant` Alternative data series to be sampled instead of wind speed. The `windRose()` default NULL is equivalent to `pollutant = "ws"`. Use in `pollutionRose()`.

`angle.scale` The scale is by default shown at a 315 degree angle. Sometimes the placement of the scale may interfere with an interesting feature. The user can therefore set `angle.scale` to another value (between 0 and 360 degrees) to mitigate such problems. For example `angle.scale = 45` will draw the scale heading in a NE direction.

`border` Border colour for shaded areas. Default is no border.

Value

A leaflet object.

See Also

the original `openair::windRose()`

`windroseMapStatic()` for the static ggmap equivalent of `windroseMap()`

Other interactive directional analysis maps: `annulusMap()`, `diffMap()`, `freqMap()`, `percentileMap()`, `polarMap()`, `pollroseMap()`

Examples

```
## Not run:
windroseMap(polar_data,
  provider = "Stamen.Toner"
)

## End(Not run)
```

`windroseMapStatic` *Wind rose plots on a static ggmap*

Description

`windroseMapStatic()` creates a ggplot2 map using wind roses as markers. As this function returns a ggplot2 object, further customisation can be achieved using functions like `ggplot2::theme()` and `ggplot2::guides()`. See `openair::polarPlot()` for more information.

Usage

```
windroseMapStatic(
  data,
  ggmap = NULL,
  ws.int = 2,
  breaks = 4,
  facet = NULL,
  latitude = NULL,
  longitude = NULL,
  cols = "turbo",
  alpha = 1,
  key = FALSE,
  facet.nrow = NULL,
  d.icon = 150,
  d.fig = 3,
  ...
)
```

Arguments

`data`	A data frame. The data frame must contain the data to plot the directional analysis marker, which includes wind speed (`ws`), wind direction (`wd`), and the column representing the concentration of a pollutant. In addition, `data` must include a decimal latitude and longitude.
`ggmap`	A ggmap object obtained using `ggmap::get_map()` or a similar function to use as the basemap.
`ws.int`	The wind speed interval. Default is 2 m/s but for low met masts with low mean wind speeds a value of 1 or 0.5 m/s may be better.

`breaks` Most commonly, the number of break points for wind speed in windRose. For windRose and the ws.int default of 2 m/s, the default, 4, generates the break points 2, 4, 6, 8 m/s. Breaks can also be used to set specific break points. For example, the argument breaks = c(0, 1, 10, 100) breaks the data into segments <1, 1-10, 10-100, >100.

`facet` Used for splitting the input data into different panels, passed to the `type` argument of `openair::cutData()`. `facet` cannot be used if multiple `pollutant` columns have been provided.

`latitude, longitude`
The decimal latitude/longitude. If not provided, will be automatically inferred from data by looking for a column named "lat"/"latitude" or "lon"/"lng"/"long"/"longitude" (case-insensitively).

`cols` The colours used for plotting. See `openair::openColours()` for more information.

`alpha` The alpha transparency to use for the plotting surface (a value between 0 and 1 with zero being fully transparent and 1 fully opaque).

`key` Should a key for each marker be drawn? Default is `FALSE`.

`facet.nrow` Passed to the `nrow` argument of `ggplot2::facet_wrap()`.

`d.icon` The diameter of the plot on the map in pixels. This will affect the size of the individual polar markers. Alternatively, a vector in the form `c(width, height)` can be provided if a non-circular marker is desired.

`d.fig` The diameter of the plots to be produced using `openair` in inches. This will affect the resolution of the markers on the map. Alternatively, a vector in the form `c(width, height)` can be provided if a non-circular marker is desired.

`...` Arguments passed on to `openair::polarAnnulus`

`resolution` Two plot resolutions can be set: "normal" and "fine" (the default).

`local.tz` Should the results be calculated in local time that includes a treatment of daylight savings time (DST)? The default is not to consider DST issues, provided the data were imported without a DST offset. Emissions activity tends to occur at local time e.g. rush hour is at 8 am every day. When the clocks go forward in spring, the emissions are effectively released into the atmosphere typically 1 hour earlier during the summertime i.e. when DST applies. When plotting diurnal profiles, this has the effect of "smearing-out" the concentrations. Sometimes, a useful approach is to express time as local time. This correction tends to produce better-defined diurnal profiles of concentration (or other variables) and allows a better comparison to be made with emissions/activity data. If set to `FALSE` then GMT is used. Examples of usage include `local.tz = "Europe/London"`, `local.tz = "America/New_York"`. See `cutData` and `import` for more details.

`type` `type` determines how the data are split i.e. conditioned, and then plotted. The default is will produce a single plot using the entire data. Type can be one of the built-in types as detailed in `cutData` e.g. "season", "year", "weekday" and so on. For example, `type = "season"` will produce four plots — one for each season.

It is also possible to choose type as another variable in the data frame. If that variable is numeric, then the data will be split into four quantiles (if possible) and labelled accordingly. If type is an existing character or factor variable, then those categories/levels will be used directly. This offers great flexibility for understanding the variation of different variables and how they depend on one another.

Type can be up length two e.g. type = c("season", "site") will produce a 2x2 plot split by season and site. The use of two types is mostly meant for situations where there are several sites. Note, when two types are provided the first forms the columns and the second the rows.

Also note that for the polarAnnulus function some type/period combinations are forbidden or make little sense. For example, type = "season" and period = "trend" (which would result in a plot with too many gaps in it for sensible smoothing), or type = "weekday" and period = "weekday".

statistic The statistic that should be applied to each wind speed/direction bin. Can be "mean" (default), "median", "max" (maximum), "frequency". "stdev" (standard deviation), "weighted.mean" or "cpf" (Conditional Probability Function). Because of the smoothing involved, the colour scale for some of these statistics is only to provide an indication of overall pattern and should not be interpreted in concentration units e.g. for statistic = "weighted.mean" where the bin mean is multiplied by the bin frequency and divided by the total frequency. In many cases using polarFreq will be better. Setting statistic = "weighted.mean" can be useful because it provides an indication of the concentration * frequency of occurrence and will highlight the wind speed/direction conditions that dominate the overall mean.

percentile If statistic = "percentile" or statistic = "cpf" then percentile is used, expressed from 0 to 100. Note that the percentile value is calculated in the wind speed, wind direction 'bins'. For this reason it can also be useful to set min.bin to ensure there are a sufficient number of points available to estimate a percentile. See quantile for more details of how percentiles are calculated.

width The width of the annulus; can be "normal" (the default), "thin" or "fat".

min.bin The minimum number of points allowed in a wind speed/wind direction bin. The default is 1. A value of two requires at least 2 valid records in each bin an so on; bins with less than 2 valid records are set to NA. Care should be taken when using a value > 1 because of the risk of removing real data points. It is recommended to consider your data with care. Also, the polarFreq function can be of use in such circumstances.

exclude.missing Setting this option to TRUE (the default) removes points from the plot that are too far from the original data. The smoothing routines will produce predictions at points where no data exist i.e. they predict. By removing the points too far from the original data produces a plot where it is clear where the original data lie. If set to FALSE missing data will be interpolated.

date.pad For type = "trend" (default), date.pad = TRUE will pad-out missing data to the beginning of the first year and the end of the last year. The purpose is to ensure that the trend plot begins and ends at the beginning or

end of year.

`force.positive` The default is `TRUE`. Sometimes if smoothing data with steep gradients it is possible for predicted values to be negative. `force.positive = TRUE` ensures that predictions remain positive. This is useful for several reasons. First, with lots of missing data more interpolation is needed and this can result in artefacts because the predictions are too far from the original data. Second, if it is known beforehand that the data are all positive, then this option carries that assumption through to the prediction. The only likely time where setting `force.positive = FALSE` would be if background concentrations were first subtracted resulting in data that is legitimately negative. For the vast majority of situations it is expected that the user will not need to alter the default option.

`k` The smoothing value supplied to `gam` for the temporal and wind direction components, respectively. In some cases e.g. a trend plot with less than 1-year of data the smoothing with the default values may become too noisy and affected more by outliers. Choosing a lower value of `k` (say 10) may help produce a better plot.

`normalise` If `TRUE` concentrations are normalised by dividing by their mean value. This is done *after* fitting the smooth surface. This option is particularly useful if one is interested in the patterns of concentrations for several pollutants on different scales e.g. NOx and CO. Often useful if more than one `pollutant` is chosen.

`key.header` Adds additional text/labels to the scale key. For example, passing the options `key.header = "header"`, `key.footer = "footer1"` adds addition text above and below the scale key. These arguments are passed to `drawOpenKey` via `quickText`, applying the `auto.text` argument, to handle formatting.

`key.footer` see `key.footer`.

`key.position` Location where the scale key is to plotted. Allowed arguments currently include `"top"`, `"right"`, `"bottom"` and `"left"`.

`auto.text` Either `TRUE` (default) or `FALSE`. If `TRUE` titles and axis labels will automatically try and format pollutant names and units properly e.g. by subscripting the '2' in NO2.

Value

a `ggplot2` plot with a `ggmap` basemap

Further customisation using ggplot2

As the outputs of the static directional analysis functions are `ggplot2` figures, further customisation is possible using functions such as `ggplot2::theme()`, `ggplot2::guides()` and `ggplot2::labs()`.

If multiple pollutants are specified, subscripting (e.g., the "x" in "NOx") is achieved using the ggtext package. Therefore if you choose to override the plot theme, it is recommended to use `[ggplot2::theme()]` and `[ggtext::element_markdown()]` to define the `strip.text` parameter.

When arguments like `limits`, `percentile` or `breaks` are defined, a legend is automatically added to the figure. Legends can be removed using `ggplot2::theme(legend.position = "none")`, or

further customised using ggplot2::guides() and either color = ggplot2::guide_colourbar() for continuous legends or fill = ggplot2::guide_legend() for discrete legends.

See Also

the original openair::windRose()

windroseMap() for the interactive leaflet equivalent of windroseMapStatic()

Other static directional analysis maps: annulusMapStatic(), diffMapStatic(), freqMapStatic(), percentileMapStatic(), polarMapStatic(), pollroseMapStatic()

Index