Package 'pivottabler'

July 23, 2025

Type Package

Title Create Pivot Tables

Version 1.5.6

Description Create regular pivot tables with just a few lines of R. More complex pivot tables can also be created, e.g. pivot tables with irregular layouts, multiple calculations and/or derived calculations based on multiple data frames. Pivot tables are constructed using R only and can be written to a range of output formats (plain text, 'HTML', 'Latex' and 'Excel'), including with styling/formatting.

Depends R (>= 3.3.0)

Imports R6 (>= 2.2.0), dplyr (>= 0.5.0), data.table (>= 1.10.0), htmltools(>= 0.3.5), htmlwidgets (>= 0.8)

Suggests ggplot2 (>= 2.2.0), jsonlite (>= 1.1), lubridate (>= 1.5.0), listviewer (>= 1.4.0), openxlsx (>= 4.0.17), basictabler (>= 1.0.2), shiny, knitr, rmarkdown, testthat

License GPL-3

Encoding UTF-8

LazyData true

URL http://www.pivottabler.org.uk/, https://github.com/cbailiss/pivottabler

BugReports https://github.com/cbailiss/pivottabler/issues

VignetteBuilder knitr

RoxygenNote 7.3.2

NeedsCompilation no

Author Christopher Bailiss [aut, cre]

Maintainer Christopher Bailiss <cbailiss@gmail.com>

Repository CRAN

Date/Publication 2025-04-22 09:00:02 UTC

Contents

`bhmtraindisruption` *Birmingham Train Disruptions, Dec 2016-Feb 2017.*

Description

A dataset containing details of the trains in the bhmtrains dataset that were partially/wholly cancelled and/or reinstated.

Usage

```
bhmtraindisruption
```

Format

A data frame with 2982 rows and 10 variables:

ServiceId Unique train identifier

LastCancellationAt Time of the last cancellation

LastCancellationLocation 3-letter code denoting the location of the last cancellation

LastCancellationReasonCategory The broad reason why the train was cancelled

LastCancellationReasonDesc A more specific reason why the train was cancelled

LastReinstatedAt Time of the last reinstatement

LastChangeOfOriginAt Time of the last change of origin

LastChangeOfOriginLocation 3-letter code denoting the location of the last change of origin

LastChangeOfOriginReasonCategory The broad reason why the origin was changed

LastChangeOfOriginReasonDesc A more specific reason why the origin was changed

Source

https://www.recenttraintimes.co.uk/

---

bhmtrains *Birmingham Trains, Dec 2016-Feb 2017.*

Description

A dataset containing all of the trains that either originated at, passed through or terminated at Birmingham New Street railway station in the UK between 1st December 2016 and 28th February 2017 inclusive.

Usage

```
bhmtrains
```

Format

A data frame with 83710 rows and 16 variables:

ServiceId Unique train identifier

Status Train status: A=Active, C=Cancelled, R=Reinstated

TOC Train operating company

TrainCategory Express Passenger or Ordinary Passenger

PowerType DMU=Diesel Multiple Unit, EMU=Electrical Multiple Unit, HST=High Speed Train

SchedSpeedMPH Scheduled maximum speed (in miles per hour)

Origin 3-letter code denoting the scheduled origin of the train

OriginGbttDeparture Scheduled departure time in the Great Britain Train Timetable (GBTT) from the origin station

OriginActualDeparture Actual departure time from the origin station

GbttArrival Scheduled arrival time in Birmingham in the GBTT

ActualArrival Actual arrival time in Birmingham in the GBTT

GbttDeparture Scheduled departure time from Birmingham in the GBTT

ActualDeparture Actual departure time from Birmingham in the GBTT

Destination 3-letter code denoting the scheduled destination of the train

DestinationGbttArrival Scheduled arrival time in the GBTT at the destination station

DestinationActualArrival Actual arrival time at the destination station

Source

https://www.recenttraintimes.co.uk/

`checkArgument` *Perform basic checks on a function argument.*

Description

`checkArgument` is a utility function that provides basic assurances about function argument values and generates standardised error messages when invalid values are encountered.

Usage

```
checkArgument(
  argumentCheckMode,
  checkDataTypes,
  className,
  methodName,
  argumentValue,
  isMissing,
  allowMissing = FALSE,
  allowNull = FALSE,
  allowedClasses = NULL,
  mustBeAtomic = FALSE,
  allowedListElementClasses = NULL,
  listElementsMustBeAtomic = FALSE,
  allowedValues = NULL,
  minValue = NULL,
  maxValue = NULL,
  maxLength = NULL
)
```

Arguments

argumentCheckMode	A number between 0 and 4 specifying the checks to perform.
checkDataTypes	A logical value specifying whether the data types should be checked when argumentCheckMode=3
className	The name of the calling class, for inclusion in error messages.
methodName	The name of the calling method, for inclusion in error messages.
argumentValue	The value to check.
isMissing	Whether the argument is missing in the calling function.
allowMissing	Whether missing values are permitted.
allowNull	Whether null values are permitted.
allowedClasses	The names of the allowed classes for argumentValue.
mustBeAtomic	Whether the argument value must be atomic.
allowedListElementClasses	For argument values that are lists(), the names of the allowed classes for the elements in the list.
listElementsMustBeAtomic	For argument values that are lists(), whether the list elements must be atomic.
allowedValues	For argument values that must be one value from a set list, the list of allowed values.
minValue	For numerical values, the lowest allowed value.
maxValue	For numerical values, the highest allowed value.
maxLength	For character values, the maximum allowed length (in characters) of the value.

Value

No return value. If invalid values are encountered, the `stop()` function is used to interrupt execution.

---

cleanCssValue *Cleans up a CSS attribute value.*

Description

`cleanCssValue` is a utility function that performs some basic cleanup on CSS attribute values. Leading and trailing whitespace is removed. The CSS values "initial" and "inherit" are blocked. The function is vectorised so can be used with arrays.

Usage

```
cleanCssValue(cssValue)
```

Arguments

cssValue	The value to cleanup.

Value

The cleaned value.

---

cleanOutlineArg *Clean the arguments specified for an outline group*

Description

cleanOutlineArg checks values and provides defaults.

Usage

```
cleanOutlineArg(
  pvt,
  outline = NULL,
  defaultCaption = "{value}",
  defaultIsEmpty = TRUE
)
```

Arguments

pvt	The pivot table.
outline	Either a logical value (TRUE to use the default outline settings) or a list specifying outline settings.
defaultCaption	The default caption of the outline group.
defaultIsEmpty	Specify whether the outline group is empty or contains a value (typically a subtotal)

Value

A listed containing checked/cleaned outline group settings.

containsText *Check whether a text value is present in another text value.*

Description

containsText is a utility function returns TRUE if one text value is present in another. Case sensitive. If textToSearch is a vector, returns TRUE if any element contains textToFind.

Usage

```
containsText(textToSearch, textToFind)
```

Arguments

textToSearch	The value to be searched.
textToFind	The value to find.

Value

TRUE if the textToFind value is found.

convertPvtStyleToBasicStyle *Convert a pivot table style to a basictabler style.*

Description

convertPvtStyleToBasicStyle is a utility function that converts a pivot table style to a basictabler style from the basictabler package.

Usage

```
convertPvtStyleToBasicStyle(btbl = NULL, pvtStyle = NULL)
```

Arguments

btbl	The basic table that will own the new style.
pvtStyle	The pivot style to convert.

Value

a basictabler style.

convertPvtTblToBasicTbl *Convert a pivot table to a basic table.*

Description

convertPvtTblToBasicTbl is a utility function that converts a pivot table to a basic table from the basictabler package.

Usage

```
convertPvtTblToBasicTbl(
  pvt = NULL,
  exportOptions = NULL,
  compatibility = NULL,
  showRowGroupHeaders = FALSE
)
```

Arguments

pvt	The pivot table to convert.
exportOptions	Options specifying how values are exported.
compatibility	Compatibility options specified when creating the basictabler table.
showRowGroupHeaders	Show captions at the top of the columns that comprise the row groups (i.e. in the top left root of then pivot table).

Value

a basictabler table.

exportValueAs *Replace the current value with a placeholder during export.*

Description

exportValueAs is a utility function that returns either the original value or a replacement placeholder value for export.

Usage

```
exportValueAs(
  rawValue,
  formattedValue,
  exportOptions,
  blankValue = character(0)
)
```

Arguments

rawValue	The raw value to check.
formattedValue	The formatted value to be exported.
exportOptions	A list of options controlling export behaviour.
blankValue	The 'placeholder' value to be exported when skipping the value.

Value

Either the original value or a placeholder value.

---

getBlankTheme *Get an empty theme for applying no styling to a table.*

Description

Get an empty theme for applying no styling to a table.

Usage

```
getBlankTheme(parentPivot, themeName = "blank")
```

Arguments

parentPivot	Owning pivot table.
themeName	The name to use as the new theme name.

Value

A TableStyles object.

---

getCompactTheme *Get the compact theme for styling a pivot table.*

Description

Get the compact theme for styling a pivot table.

Usage

```
getCompactTheme(parentPivot, themeName = "compact")
```

Arguments

parentPivot	Owning pivot table.
themeName	The name to use as the new theme name.

Value

A PivotStyles object.

`getDefaultTheme` *Get the default theme for styling a pivot table.*

Description

Get the default theme for styling a pivot table.

Usage

```
getDefaultTheme(parentPivot, themeName = "default")
```

Arguments

`parentPivot`	Owning pivot table.
`themeName`	The name to use as the new theme name.

Value

A PivotStyles object.

`getLargePlainTheme` *Get the large plain theme for styling a pivot table.*

Description

Get the large plain theme for styling a pivot table.

Usage

```
getLargePlainTheme(parentPivot, themeName = "largeplain")
```

Arguments

`parentPivot`	Owning pivot table.
`themeName`	The name to use as the new theme name.

Value

A PivotStyles object.

`getNextPosition` *Find the first value in an array that is larger than the specified value.*

Description

`getNextPosition` is a utility function that helps when parsing strings that contain delimiters.

Usage

```
getNextPosition(positions, afterPosition)
```

Arguments

`positions`	An ordered numeric vector.
`afterPosition`	The value to start searching after.

Value

The first value in the array larger than afterPosition.

`getPvtStyleDeclarations`
Get pivot table style declarations from a pivot table style.

Description

`getPvtStyleDeclarations` is a utility function that reads the styles from a pivot table style.

Usage

```
getPvtStyleDeclarations(pvtStyle = NULL)
```

Arguments

`pvtStyle`	The pivot table style to read.

Value

a list of style declarations.

`getSimpleColoredTheme` *Get a simple coloured theme.*

Description

Get a simple coloured theme that can be used to style a pivot table into a custom colour scheme.

Usage

```
getSimpleColoredTheme(
  parentPivot,
  themeName = "coloredTheme",
  colors = NULL,
  fontName = NULL,
  theme = NULL
)
```

Arguments

`parentPivot`	Owning pivot table.
`themeName`	The name to use as the new theme name.
`colors`	The set of colours to use when generating the theme (see the Styling vignette for details). This parameter exists for backward compatibility.
`fontName`	The name of the font to use, or a comma separated list (for font-fall-back). This parameter exists for backward compatibility.
`theme`	A simple theme specified in the form of a list. See example for supported list elements (all other elements will be ignored).

Value

A 'PivotStyles' object.

Examples

```
pt <- PivotTable$new()
# ...
simpleBlueTheme <- list(
  fontName="Verdana, Arial",
  fontSize="0.75em",
  headerBackgroundColor = "rgb(68, 114, 196)",
  headerColor = "rgb(255, 255, 255)",
  cellBackgroundColor = "rgb(255, 255, 255)",
  cellColor = "rgb(0, 0, 0)",
  outlineCellBackgroundColor = "rgb(186, 202, 233)",
  outlineCellColor = "rgb(0, 0, 0)",
  totalBackgroundColor = "rgb(186, 202, 233)",
  totalColor = "rgb(0, 0, 0)",
```

```
  borderColor = "rgb(48, 84, 150)"
)
pt$theme <- simpleBlueTheme
# or
theme <- getSimpleColoredTheme(pt, theme=simpleBlueTheme)
# make further changes to the theme
pt$theme <- theme
```

`getStandardTableTheme` *Get the a theme for styling to a pivot table that looks more like a standard table (i.e. no row column headings).*

Description

Get the a theme for styling to a pivot table that looks more like a standard table (i.e. no row column headings).

Usage

```
getStandardTableTheme(parentPivot, themeName = "standardtable")
```

Arguments

`parentPivot`	Owning pivot table.
`themeName`	The name to use as the new theme name.

Value

A PivotStyles object.

`getTheme` *Get a built-in theme for styling a pivot table.*

Description

`getTheme` returns the specified theme.

Usage

```
getTheme(parentPivot, themeName = NULL)
```

Arguments

`parentPivot`	Owning pivot table.
`themeName`	The name of the theme to retrieve.

Value

A PivotStyles object.

getXlBorderFromCssBorder

Convert CSS border values to those used by the openxlsx package.

Description

getXlBorderFromCssBorder parses the CSS combined border declarations (i.e. border, border-left, border-right, border-top, border-bottom) and returns a list containing an openxlsx border style and color as separate elements.

Usage

```
getXlBorderFromCssBorder(text)
```

Arguments

text	The border declaration to parse.

Value

A list containing two elements: style and color.

getXlBorderStyleFromCssBorder

Convert CSS border values to those used by the openxlsx package.

Description

getXlBorderStyleFromCssBorder takes border parameters expressed as a list (containing elements: width and style) and returns a border style that is compatible with the openxlsx package.

Usage

```
getXlBorderStyleFromCssBorder(border)
```

Arguments

border	A list containing elements width and style.

Value

An openxlsx border style.

isNumericValue *Check whether a numeric value is present.*

Description

isNumericValue is a utility function returns TRUE only when a numeric value is present. NULL, NA, numeric(0) and integer(0) all return FALSE.

Usage

```
isNumericValue(value)
```

Arguments

value	The value to check.

Value

TRUE if a numeric value is present.

isTextValue *Check whether a text value is present.*

Description

isTextValue is a utility function returns TRUE only when a text value is present. NULL, NA, character(0) and "" all return FALSE.

Usage

```
isTextValue(value)
```

Arguments

value	The value to check.

Value

TRUE if a non-blank text value is present.

oneToNULL *Convert a value of 1 to a NULL value.*

Description

oneToNull is a utility function that returns NULL when a value of 0 or 1 is passed to it, otherwise it returns the original value.

Usage

```
oneToNULL(value, convertOneToNULL)
```

Arguments

value	The value to check.
convertOneToNULL	TRUE to convert 1 to NULL.

Value

NULL if value is 0 or 1, otherwise value.

parseColor *Convert a CSS colour into a hex based colour code.*

Description

parseColor converts a colour value specified in CSS to a hex based colour code. Example supported input values/formats/named colours are: #0080FF, rgb(0, 128, 255), rgba(0, 128, 255, 0.5) and red, green, etc.

Usage

```
parseColor(color)
```

Arguments

color	The colour to convert.

Value

The colour as a hex code, e.g. #FF00A0.

parseCssBorder *Parse a CSS border value.*

Description

parseCssBorder parses the CSS combined border declarations (i.e. border, border-left, border-right, border-top, border-bottom) and returns a list containing the width, style and color as separate elements.

Usage

```
parseCssBorder(text)
```

Arguments

text	The border declaration to parse.

Value

A list containing three elements: width, style and color.

parseCssSizeToPt *Convert a CSS size value into points.*

Description

parseCssSizeToPt will take a CSS style and convert it to points. Supported input size units are in, cm, mm, pt, pc, px, em, are converted exactly: in, cm, mm, pt, pc: using 1in = 2.54cm = 25.4mm = 72pt = 6pc. The following are converted approximately: px, em, approx 1em=16px=12pt and 100

Usage

```
parseCssSizeToPt(size)
```

Arguments

size	A size specified in common CSS units.

Value

The size converted to points.

parseCssSizeToPx *Convert a CSS size value into pixels*

Description

parseCssSizeToPx will take a CSS style and convert it to pixels Supported input size units are in, cm, mm, pt, pc, px, em, are converted exactly: in, cm, mm, pt, pc: using 1in = 2.54cm = 25.4mm = 72pt = 6pc. The following are converted approximately: px, em, approx 1em=16px=12pt and 100

Usage

```
parseCssSizeToPx(size)
```

Arguments

size	A size specified in common CSS units.

Value

The size converted to pixels.

parseCssString *Split a CSS attribute value into a vector/array.*

Description

parseCssString is a utility function that splits a string into a vector/array. The function pays attention to text qualifiers (single and double quotes) so won't split if the delimiter occurs inside a value.

Usage

```
parseCssString(text, separator = ",", removeEmptyString = TRUE)
```

Arguments

text	The text to split.
separator	The field separator, default comma.
removeEmptyString	TRUE to not return empty string / whitespace values.

Value

An R vector containing the values from text split up.

parseXlBorder *Parse an xl-border value.*

Description

parseXlBorder parses the combined xl border declarations (i.e. xl-border, xl-border-left, xl-border-right, xl-border-top, xl-border-bottom) and returns a list containing style and color as separate elements.

Usage

```
parseXlBorder(text)
```

Arguments

text	The border declaration to parse.

Value

A list containing two elements: style and color.

PivotBatch *R6 class the represents a Calculation Batch*

Description

The 'PivotBatch' class represents one combination of data, variables and calculations that are needed when calculating the values of cells in a pivot table.

Format

R6Class object.

Details

The combination of data name and variable names defines a batch. When the batch is calculated, the calculations specified in the batch are evaluated against the specified data, with the data being grouped by the variables specified in the batch. Individual result values can then be retrieved from the batch. See the "Performance" vignette for details.

Active bindings

`batchId` The unique identifier for the batch.

`batchName` The unique name of the batch.

`compatibleCount` The number of pivot cell calculations that this batch supports.

`evaluated` TRUE if this batch has been evaluated.

`results` The results (a data frame) of the evaluation of the batch

`asString` A text description of the batch.

Methods

Public methods:

- `PivotBatch$new()`
- `PivotBatch$isCompatible()`
- `PivotBatch$addCompatible()`
- `PivotBatch$getCalculationInternalName()`
- `PivotBatch$evaluateBatch()`
- `PivotBatch$getSummaryValueFromBatch()`
- `PivotBatch$clone()`

Method `new()`**:** Create a new 'PivotBatch' object.

Usage:

```
PivotBatch$new(
  parentPivot = NULL,
  batchId = 0,
  dataName = NULL,
  variableNames = NULL,
  values = NULL,
  calculationName = NULL,
  calculationGroupName = NULL
)
```

Arguments:

`parentPivot` The pivot table that this 'PivotBatch' instance belongs to.

`batchId` The unique identifier for the batch.

`dataName` The name of the data frame (as specified in 'pt$addData()') that this batch relates to.

`variableNames` Specifies the combination of variable names (i.e. dimensionality) of the batch.

`values` A list specifying the distinct list of values for each variable, i.e. 'list(varName1=values1, varName2=values2, ...)'. 'values' is not currently used and does not affect the batch compatibility logic.

`calculationName` The first calculation added to this batch. Does not affect the batch compatibility logic.

`calculationGroupName` The calculation group of the first calculation added to this batch. Does not affect the batch compatibility logic.

Returns: A new 'PivotBatch' object.

Method `isCompatible()`**:** Determine whether a combination of data and variables is compatible with this batch.

Usage:

```
PivotBatch$isCompatible(dataName = NULL, variableNames = NULL)
```

Arguments:

`dataName` The name of the data frame (as specified in 'pt$addData()').

`variableNames` Specifies the combination of variable names (i.e. dimensionality)..

Returns: 'TRUE' or 'FALSE'.

Method `addCompatible()`**:** Add a new set of values or a new calculation to the batch. with this batch.

Usage:

```
PivotBatch$addCompatible(
  values = NULL,
  calculationName = NULL,
  calculationGroupName = NULL
)
```

Arguments:

`values` A list specifying the distinct list of values for each variable, i.e. 'list(varName1=values1, varName2=values2, ...)'. 'values' is not currently used and does not affect the batch compatibility logic.

`calculationName` The calculation to add to the batch. Does not affect the batch compatibility logic.

`calculationGroupName` The calculation group of the calculation to add to the batch. Does not affect the batch compatibility logic.

Returns: No return value.

Method `getCalculationInternalName()`**:** Find the internal name of a calculation in the batch.

Usage:

```
PivotBatch$getCalculationInternalName(
  calculationName = NULL,
  calculationGroupName = NULL
)
```

Arguments:

`calculationName` The name of the calculation to find.

`calculationGroupName` The calculation group of the calculation to find.

Returns: The internal name of the calculation in the batch.

Method `evaluateBatch()`**:** Carry out grouping and calculations to evaluate the batch.

Usage:

```
PivotBatch$evaluateBatch()
```

Returns: No return value.

Method `getSummaryValueFromBatch()`**:** Retrieve one calculation value from the batch, typically for the value of one cell in a pivot table.

Usage:

```
PivotBatch$getSummaryValueFromBatch(
  filters = NULL,
  calculationName = NULL,
  calculationGroupName = NULL
)
```

Arguments:

`filters` A 'PivotFilters' instance that specifies which value to retrieve. This filters object is a combination of the row, column and calculation filters.

`calculationName` The name of the calculation value to retrieve.

`calculationGroupName` The calculation group of the calculation to retrieve.

Returns: A single calculation value.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PivotBatch$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# This class should only be created by the pivot table.
# It is not intended to be created outside of the pivot table.
```

`PivotBatchCalculator` *R6 class that calculates the values for multiple cells in batches.*

Description

The 'PivotBatchCalculator' class calculates the values for multiple cells in the pivot table in one evaluation step (per batch), instead of evaluating every calculation once per pivot table cell.

Format

`R6Class` object.

Details

Evaluating a set of filters and calculations repetitively for each cell is inefficient and slow. The Batch Calculator executes a much small number of calculations which greatly reduces the CPU time and elapsed time required. See the "Performance" vignette for details.

Active bindings

`batchCount` The number of batches generated for the pivot table.

`calculationSummary` A summary of the batch compatibility for each calculation.

`batchSummary` A summary of the batches in the pivot table.

Methods

Public methods:

- `PivotBatchCalculator$new()`
- `PivotBatchCalculator$reset()`
- `PivotBatchCalculator$checkValidWorkingData()`
- `PivotBatchCalculator$isFiltersBatchCompatible()`
- `PivotBatchCalculator$generateBatchesForNamedCalculationEvaluation1()`
- `PivotBatchCalculator$generateBatchesForNamedCalculationEvaluation2()`
- `PivotBatchCalculator$generateBatchesForCellEvaluation()`
- `PivotBatchCalculator$evaluateBatches()`
- `PivotBatchCalculator$getSummaryValueFromBatch()`
- `PivotBatchCalculator$clone()`

Method `new()`**:** Create a new 'PivotBatchCalculator' object.

Usage:

```
PivotBatchCalculator$new(parentPivot = NULL)
```

Arguments:

`parentPivot` The pivot table that this 'PivotBatchCalculator' instance belongs to.

Returns: A new 'PivotBatchCalculator' object.

Method `reset()`**:** Reset the batch calculator, clearing all existing batches.

Usage:

```
PivotBatchCalculator$reset()
```

Returns: No return value.

Method `checkValidWorkingData()`**:** Run some additional checks to see whether the working data is valid. Typically only used in development builds of the package.

Usage:

```
PivotBatchCalculator$checkValidWorkingData(workingData = NULL)
```

Arguments:

`workingData` The working data to check.

Returns: No return value.

Method `isFiltersBatchCompatible()`**:** Examines a set of filters to see whether they are compatible with batch evaluation mode. Only filters that specify zero or one value for each variable are compatible with batch evaluation.

Usage:

```
PivotBatchCalculator$isFiltersBatchCompatible(filters = NULL)
```

Arguments:

`filters` A 'PivotFilters' object that represents a set of filters to examine.

Details: It is not practical to make batch evaluation work where a filter matches more than one value for a variable. One approach might be to add a derived column where a single value represents the multiple values, however the combination of values could partially overlap with combinations of values in other data groups. Also the value that represents the "combined" value could collide with other existing values in the column. In summary: Sequential mode is slower and more flexible. Batch is faster but stricter. Batch mode works for regular pivot tables (i.e. most cases).

Returns: 'TRUE' if the filters are batch compatible, 'FALSE' otherwise.

Method `generateBatchesForNamedCalculationEvaluation1()`**:** Generates a new batch or finds a relevant existing batch for a named calculation and single working filters object.

Usage:

```
PivotBatchCalculator$generateBatchesForNamedCalculationEvaluation1(
  dataName = NULL,
  calculationName = NULL,
  calculationGroupName = NULL,
  workingFilters = NULL
)
```

Arguments:

`dataName` The name of the data frame (as specified in 'pt$addData()').

`calculationName` The name of the calculation.

`calculationGroupName` The calculation group of the calculation.

`workingFilters` A 'PivotFilters' object that represents the working filters to generate the batch for.

Returns: The name of either the batch that was created or the relevant existing batch.

Method `generateBatchesForNamedCalculationEvaluation2()`**:** Generates one or more batches for the named calculations and set of working working data associated with a cell.

Usage:

```
PivotBatchCalculator$generateBatchesForNamedCalculationEvaluation2(
  calculationName = NULL,
  calculationGroupName = NULL,
  workingData = NULL
)
```

Arguments:

`calculationName` The name of the calculation.

`calculationGroupName` The calculation group of the calculation.

`workingData` A list containing filter and/or filter overrides.

Details: A wrapper around 'generateBatchesForNamedCalculationEvaluation1()', which invokes this function as appropriate, depending on whether a calculation if either of type "summary" or type "calculation".

Returns: One or more batch names of either the batches that were created or the relevant existing batches.

Method `generateBatchesForCellEvaluation()`**:** Generates the batches for batch evaluation mode.

Usage:

```
PivotBatchCalculator$generateBatchesForCellEvaluation()
```

Returns: One or more batch names of either the batches that were created or the relevant existing batches.

Method `evaluateBatches()`**:** Evaluate each of the batches defined in the batch calculator.

Usage:

```
PivotBatchCalculator$evaluateBatches()
```

Returns: The number of batches that were evaluated.

Method `getSummaryValueFromBatch()`**:** Retrieve one calculation value from one batch, typically for the value of one cell in a pivot table.

Usage:

```
PivotBatchCalculator$getSummaryValueFromBatch(
  batchName = NULL,
  calculationName = NULL,
  calculationGroupName = NULL,
  workingFilters = NULL
)
```

Arguments:

`batchName` The name of the batch containing the calculation result.

`calculationName` The name of the calculation.

`calculationGroupName` The calculation group of the calculation.

`workingFilters` A 'PivotFilters' object that represents the working filters to retrieve the value for.

Returns: A single calculation value.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PivotBatchCalculator$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# This class should only be created by the pivot table.
# It is not intended to be created outside of the pivot table.
```

PivotBatchStatistics *R6 class that provides summary statistics for batch calculations.*

Description

The 'PivotBatchStatistics' class contains a set of summary statistics that track how many calculations are batch compatible/incompatible.

Format

R6Class object.

Active bindings

asString A text description of the batch statistics.

Methods

Public methods:

- PivotBatchStatistics$new()
- PivotBatchStatistics$reset()
- PivotBatchStatistics$incrementNoData()
- PivotBatchStatistics$incrementCompatible()
- PivotBatchStatistics$incrementIncompatible()
- PivotBatchStatistics$clone()

Method new(): Create a new 'PivotBatchStatistics' object.

Usage:

```
PivotBatchStatistics$new(parentPivot = NULL)
```

Arguments:

parentPivot The pivot table that this 'PivotBatchStatistics' instance belongs to.

Returns: A new 'PivotBatchStatistics' object.

Method reset(): Clear the batch statistics.

Usage:

```
PivotBatchStatistics$reset()
```

Returns: No return value.

Method incrementNoData(): Increment the noData count for a batch.

Usage:

```
PivotBatchStatistics$incrementNoData(
  calculationName = NULL,
  calculationGroupName = NULL
)
```

Arguments:

`calculationName` The name of the calculation to increment the count for.

`calculationGroupName` The name of the calculation group for the calculation.

Returns: No return value.

Method `incrementCompatible()`**:** Increment the compatible count for a batch.

Usage:

```
PivotBatchStatistics$incrementCompatible(
  calculationName = NULL,
  calculationGroupName = NULL
)
```

Arguments:

`calculationName` The name of the calculation to increment the count for.

`calculationGroupName` The name of the calculation group for the calculation.

Returns: No return value.

Method `incrementIncompatible()`**:** Increment the incompatible count for a batch.

Usage:

```
PivotBatchStatistics$incrementIncompatible(
  calculationName = NULL,
  calculationGroupName = NULL
)
```

Arguments:

`calculationName` The name of the calculation to increment the count for.

`calculationGroupName` The name of the calculation group for the calculation.

Returns: No return value.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PivotBatchStatistics$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# This class should only be created by the pivot table.
# It is not intended to be created outside of the pivot table.
```

`PivotCalculation` *R6 class that defines a calculation.*

Description

The 'PivotCalculation' class defines one calculation in a pivot table.

Format

`R6Class` object.

Active bindings

`calculationName` Calculation unique name.

`caption` Calculation display name

`visible` 'TRUE' to show the calculation in the pivot table or 'FALSE' to hide it. Hidden calculations are typically used as base values for other calculations.

`displayOrder` The order the calculations are displayed in the pivot table.

`filters` Any additional data filters specific to this calculation. This can be a 'PivotFilters' object that further restricts the data for the calculation or a list of individual 'PivotFilter' objects that provide more flexibility (and/or/replace). See the Calculations vignette for details.

`format` A character, list or custom function to format the calculation result.

`fmtFuncArgs` A list that specifies any additional arguments to pass to a custom format function.

`dataName` Specifies which data frame in the pivot table is used for this calculation (as specified in 'pt$addData()').

`type` The calculation type: "summary", "calculation", "function" or "value".

`valueName` For type="value", the name of the column containing the value to display in the pivot table.

`summariseExpression` For type="summary", either the dplyr expression to use with dplyr::summarise() or a data.table calculation expression.

`calculationExpression` For type="calculation", an expression to combine aggregate values.

`calculationFunction` For type="function", a reference to a custom R function that will carry out the calculation.

`calcFuncArgs` For type="function", a list that specifies additional arguments to pass to calculationFunction.

`basedOn` A character vector specifying the names of one or more calculations that this calculation depends on.

`noDataValue` An integer or numeric value specifying the value to use if no data exists for a particular cell.

`noDataCaption` A character value that will be displayed by the pivot table if no data exists for a particular cell.

`headingBaseStyleName` The name of a style defined in the pivot table to use as the base styling for the data group heading.

`headingStyleDeclarations` A list of CSS style declarations (e.g. 'list("font-weight"="bold")') to override the base style.

`cellBaseStyleName` The name of a style defined in the pivot table to use as the base styling for the cells related to this calculation.

`cellStyleDeclarations` A list of CSS style declarations (e.g. 'list("font-weight"="bold")') to override the base style.

Methods

Public methods:

- `PivotCalculation$new()`
- `PivotCalculation$asList()`
- `PivotCalculation$asJSON()`
- `PivotCalculation$asString()`
- `PivotCalculation$clone()`

Method `new()`**:** Create a new 'PivotCalculation' object.

Usage:

```
PivotCalculation$new(
  parentPivot,
  calculationName = NULL,
  caption = NULL,
  visible = TRUE,
  displayOrder = NULL,
  filters = NULL,
  format = NULL,
  fmtFuncArgs = NULL,
  dataName = NULL,
  type = "summary",
  valueName = NULL,
  summariseExpression = NULL,
  calculationExpression = NULL,
  calculationFunction = NULL,
  calcFuncArgs = NULL,
  basedOn = NULL,
  noDataValue = NULL,
  noDataCaption = NULL,
  headingBaseStyleName = NULL,
  headingStyleDeclarations = NULL,
  cellBaseStyleName = NULL,
  cellStyleDeclarations = NULL
)
```

Arguments:

`parentPivot` The pivot table that this 'PivotCalculation' instance belongs to.

`calculationName` Calculation unique name.

`caption` Calculation display name

`visible` 'TRUE' to show the calculation in the pivot table or 'FALSE' to hide it. Hidden calculations are typically used as base values for other calculations.

`displayOrder` The order the calculations are displayed in the pivot table.

`filters` Any additional data filters specific to this calculation. This can be a 'PivotFilters' object that further restricts the data for the calculation or a list of individual 'PivotFilter' objects that provide more flexibility (and/or/replace). See the Calculations vignette for details.

`format` A character, list or custom function to format the calculation result.

`fmtFuncArgs` A list that specifies any additional arguments to pass to a custom format function.

`dataName` Specifies which data frame in the pivot table is used for this calculation (as specified in 'pt$addData()').

`type` The calculation type: "summary", "calculation", "function" or "value".

`valueName` For type="value", the name of the column containing the value to display in the pivot table.

`summariseExpression` For type="summary", either the dplyr expression to use with dplyr::summarise() or a data.table calculation expression.

`calculationExpression` For type="calculation", an expression to combine aggregate values.

`calculationFunction` For type="function", a reference to a custom R function that will carry out the calculation.

`calcFuncArgs` For type="function", a list that specifies additional arguments to pass to calculationFunction.

`basedOn` A character vector specifying the names of one or more calculations that this calculation depends on.

`noDataValue` An integer or numeric value specifying the value to use if no data exists for a particular cell.

`noDataCaption` A character value that will be displayed by the pivot table if no data exists for a particular cell.

`headingBaseStyleName` The name of a style defined in the pivot table to use as the base styling for the data group heading.

`headingStyleDeclarations` A list of CSS style declarations (e.g. 'list("font-weight"="bold")') to override the base style.

`cellBaseStyleName` The name of a style defined in the pivot table to use as the base styling for the cells related to this calculation.

`cellStyleDeclarations` A list of CSS style declarations (e.g. 'list("font-weight"="bold")') to override the base style.

Returns: A new 'PivotCalculation' object.

Method `asList()`**:** Return the contents of this object as a list for debugging.

Usage:

```
PivotCalculation$asList()
```

Returns: A list of various object properties.

Method `asJSON()`**:** Return the contents of this object as JSON for debugging.

Usage:

```
PivotCalculation$asJSON()
```

Returns: A JSON representation of various object properties.

Method `asString()`**:** Return a representation of this object as a character value.

Usage:

```
PivotCalculation$asString()
```

Returns: A character summary of various object properties.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PivotCalculation$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# This class should only be created by the pivot table.
# It is not intended to be created outside of the pivot table.
```

`PivotCalculationGroup` *R6 class that defines a group of calculations.*

Description

The 'PivotCalculationGroup' class is a container for multiple 'PivotCalculation' objects. Every pivot table has at least one pivot calculation group and this is sufficient for all regular pivot tables. Additional calculation groups are typically only created for irregular/custom pivot tables. See the "Irregular Layout" vignette for an example.

Format

`R6Class` object.

Active bindings

`calculationGroupName` The name of the calculation group.

`defaultCalculationName` The name of the default calculation in this calculation group.

`count` The number of calculations in this calculation group.

`calculations` A list containing the calculations in this group.

`visibleCount` The number of visible calculations in this calculation group.

`visibleCalculations` A list containing the visible calculations in this group.

Methods

Public methods:

- PivotCalculationGroup$new()
- PivotCalculationGroup$isExistingCalculation()
- PivotCalculationGroup$item()
- PivotCalculationGroup$getCalculation()
- PivotCalculationGroup$defineCalculation()
- PivotCalculationGroup$asList()
- PivotCalculationGroup$asJSON()
- PivotCalculationGroup$asString()
- PivotCalculationGroup$clone()

Method `new()`**:** Create a new 'PivotCalculationGroup' object.

Usage:

```
PivotCalculationGroup$new(parentPivot, calculationGroupName = NULL)
```

Arguments:

`parentPivot` The pivot table that this 'PivotCalculationGroup' instance belongs to.

`calculationGroupName` Calculation group unique name. Recommendation: Do not have spaces in this name.

Returns: A new 'PivotCalculationGroup' object.

Method `isExistingCalculation()`**:** Check whether a calculation already exists in this calculation group.

Usage:

```
PivotCalculationGroup$isExistingCalculation(calculationName = NULL)
```

Arguments:

`calculationName` group unique name.

Returns: 'TRUE' if a calculation with the specified name exists in this calculation group object, 'FALSE' otherwise.

Method `item()`**:** Retrieve a calculation by index.

Usage:

```
PivotCalculationGroup$item(index)
```

Arguments:

`index` An integer specifying the calculation to retrieve.

Returns: The calculation that exists at the specified index.

Method `getCalculation()`**:** Retrieve a calculation by name.

Usage:

```
PivotCalculationGroup$getCalculation(calculationName = NULL)
```

Arguments:

`calculationName` The name of the calculation to retrieve.

Returns: The calculation with the specified name.

Method `defineCalculation()`**:** Create a new 'PivotCalculation' object.

Usage:

```
PivotCalculationGroup$defineCalculation(
  calculationName = NULL,
  caption = NULL,
  visible = TRUE,
  displayOrder = NULL,
  filters = NULL,
  format = NULL,
  fmtFuncArgs = NULL,
  dataName = NULL,
  type = "summary",
  valueName = NULL,
  summariseExpression = NULL,
  calculationExpression = NULL,
  calculationFunction = NULL,
  calcFuncArgs = NULL,
  basedOn = NULL,
  noDataValue = NULL,
  noDataCaption = NULL,
  headingBaseStyleName = NULL,
  headingStyleDeclarations = NULL,
  cellBaseStyleName = NULL,
  cellStyleDeclarations = NULL
)
```

Arguments:

`calculationName` Calculation unique name.

`caption` Calculation display name

`visible` 'TRUE' to show the calculation in the pivot table or 'FALSE' to hide it. Hidden calculations are typically used as base values for other calculations.

`displayOrder` The order the calculations are displayed in the pivot table.

`filters` Any additional data filters specific to this calculation. This can be a 'PivotFilters' object that further restricts the data for the calculation or a list of individual 'PivotFilter' objects that provide more flexibility (and/or/replace). See the Calculations vignette for details.

`format` A character, list or custom function to format the calculation result.

`fmtFuncArgs` A list that specifies any additional arguments to pass to a custom format function.

`dataName` Specifies which data frame in the pivot table is used for this calculation (as specified in 'pt$addData()').

`type` The calculation type: "summary", "calculation", "function" or "value".

`valueName` For type="value", the name of the column containing the value to display in the pivot table.

`summariseExpression` For type="summary", either the dplyr expression to use with dplyr::summarise() or a data.table calculation expression.

`calculationExpression` For type="calculation", an expression to combine aggregate values.

`calculationFunction` For type="function", a reference to a custom R function that will carry out the calculation.

`calcFuncArgs` For type="function", a list that specifies additional arguments to pass to calculationFunction.

`basedOn` A character vector specifying the names of one or more calculations that this calculation depends on.

`noDataValue` An integer or numeric value specifying the value to use if no data exists for a particular cell.

`noDataCaption` A character value that will be displayed by the pivot table if no data exists for a particular cell.

`headingBaseStyleName` The name of a style defined in the pivot table to use as the base styling for the data group heading.

`headingStyleDeclarations` A list of CSS style declarations (e.g. 'list("font-weight"="bold")') to override the base style.

`cellBaseStyleName` The name of a style defined in the pivot table to use as the base styling for the cells related to this calculation.

`cellStyleDeclarations` A list of CSS style declarations (e.g. 'list("font-weight"="bold")') to override the base style.

Returns: A new 'PivotCalculation' object.

Method `asList()`**:** Return the contents of this object as a list for debugging.

Usage:

```
PivotCalculationGroup$asList()
```

Returns: A list of various object properties.

Method `asJSON()`**:** Return the contents of this object as JSON for debugging.

Usage:

```
PivotCalculationGroup$asJSON()
```

Returns: A JSON representation of various object properties.

Method `asString()`**:** Return a representation of this object as a character value.

Usage:

```
PivotCalculationGroup$asString(seperator = ", ")
```

Arguments:

`seperator` A character value used when concatenating the text representations of different calculations.

Returns: A character summary of various object properties.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PivotCalculationGroup$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# This class should only be created by the pivot table.
# It is not intended to be created outside of the pivot table.
```

`PivotCalculationGroups` *R6 class that contains multiple calculation groups.*

Description

The 'PivotCalculationGroups' class stores all of the calculation groups for a pivot table. Every pivot table has at least one pivot calculation group and this is sufficient for all regular pivot tables. Additional calculation groups are typically only created for irregular/custom pivot tables. See the "Irregular Layout" vignette for an example.

Format

`R6Class` object.

Active bindings

`count` The number of calculation groups in the pivot table.

`groups` A list containing the calculation groups in the pivot table.

`defaultGroup` The default calculation group in the pivot table.

Methods

Public methods:

- `PivotCalculationGroups$new()`
- `PivotCalculationGroups$isExistingCalculationGroup()`
- `PivotCalculationGroups$item()`
- `PivotCalculationGroups$getCalculationGroup()`
- `PivotCalculationGroups$addCalculationGroup()`
- `PivotCalculationGroups$asList()`
- `PivotCalculationGroups$asJSON()`
- `PivotCalculationGroups$asString()`
- `PivotCalculationGroups$clone()`

Method `new()`**:** Create a new 'PivotCalculationGroups' object.

Usage:

```
PivotCalculationGroups$new(parentPivot)
```

Arguments:

`parentPivot` The pivot table that this 'PivotCalculationGroups' instance belongs to.

Returns: A new 'PivotCalculationGroups' object.

Method `isExistingCalculationGroup()`: Check if a calculation group exists with the specified name.

Usage:

```
PivotCalculationGroups$isExistingCalculationGroup(calculationGroupName = NULL)
```

Arguments:

`calculationGroupName` The name of the calculation group.

Returns: 'TRUE' if the calculation group already exists, 'FALSE' otherwise.

Method `item()`: Retrieve a calculation group by index.

Usage:

```
PivotCalculationGroups$item(index)
```

Arguments:

`index` An integer specifying the calculation group to retrieve.

Returns: The calculation group that exists at the specified index.

Method `getCalculationGroup()`: Retrieve a calculation group by name.

Usage:

```
PivotCalculationGroups$getCalculationGroup(calculationGroupName = NULL)
```

Arguments:

`calculationGroupName` The name of the calculation group to retrieve.

Returns: The calculation group with the specified name.

Method `addCalculationGroup()`: Create a new calculation group.

Usage:

```
PivotCalculationGroups$addCalculationGroup(calculationGroupName = NULL)
```

Arguments:

`calculationGroupName` The name of the calculation group to create

Returns: The new calculation group.

Method `asList()`: Return the contents of this object as a list for debugging.

Usage:

```
PivotCalculationGroups$asList()
```

Returns: A list of various object properties.

Method `asJSON()`: Return the contents of this object as JSON for debugging.

Usage:

```
PivotCalculationGroups$asJSON()
```

Returns: A JSON representation of various object properties.

Method `asString()`: Return a representation of this object as a character value.

Usage:

```
PivotCalculationGroups$asString(seperator = ", ")
```

Arguments:

`seperator` A character value used when concatenating the text representations of different calculation groups.

Returns: A character summary of various object properties.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PivotCalculationGroups$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# This class should only be created by the pivot table.
# It is not intended to be created outside of the pivot table.
```

`PivotCalculator` *R6 class that computes the value of a cell or cells.*

Description

The 'PivotCalculator' class has various functions and methods that assist with calculating the value of a cell or cells in a pivot table.

Format

`R6Class` object.

Details

This class contains all of the logic necessary for evaluating calculations. For batch mode calculations, it makes use of the 'PivotBatchCalculator' class to carry out the calculation batches, then retrieves the results from the relevant batch for each calculation. For sequential mode calculations, this class carries out the calculations. Where a pivot table contains some cells that can be evaluated in batch mode and some that cannot, this class contains the appropriate logic to use the relevant calculation mode in each case, preferring to use batch mode where possible, unless this has been disabled in the pivot table settings. There are many utility methods in this class that are thin wrappers around methods in other classes. This simplifies calling these other methods as well as providing a more unified way to change in the future how these common operations are performed. Custom calculation functions are passed an instance of the 'PivotCalculator' class, thereby also providing the authors of custom calculation functions an easy way for custom calculation functions to carry out common operations.

Active bindings

`batchInfo` A summary of the batches used in evaluating the pivot table.

Methods

Public methods:

- PivotCalculator$new()
- PivotCalculator$getDataFrame()
- PivotCalculator$countTotalData()
- PivotCalculator$getTotalDataFrame()
- PivotCalculator$getCalculationGroup()
- PivotCalculator$getCalculation()
- PivotCalculator$generateBatchesForCellEvaluation()
- PivotCalculator$evaluateBatches()
- PivotCalculator$newFilter()
- PivotCalculator$newFilters()
- PivotCalculator$setFilters()
- PivotCalculator$setFilter()
- PivotCalculator$setFilterValues()
- PivotCalculator$getFilteredDataFrame()
- PivotCalculator$getDistinctValues()
- PivotCalculator$formatValue()
- PivotCalculator$getCombinedFilters()
- PivotCalculator$getFiltersForNamedCalculation()
- PivotCalculator$setWorkingData()
- PivotCalculator$evaluateSingleValue()
- PivotCalculator$evaluateSummariseExpression()
- PivotCalculator$evaluateCalculationExpression()
- PivotCalculator$evaluateCalculateFunction()
- PivotCalculator$evaluateNamedCalculationWD()
- PivotCalculator$evaluateNamedCalculation()
- PivotCalculator$evaluateCell()
- PivotCalculator$clone()

Method `new()`**:** Create a new 'PivotCalculator' object.

Usage:

```
PivotCalculator$new(parentPivot = NULL)
```

Arguments:

`parentPivot` The pivot table that this 'PivotCalculator' instance belongs to.

Returns: A new 'PivotCalculator' object.

Method `getDataFrame()`**:** Retrieve a data frame that was added to the pivot table.

Usage:

```
PivotCalculator$getDataFrame(dataName = NULL)
```

Arguments:

`dataName` The name of the data frame (as specified in 'pt$addData()') to retrieve.

Returns: The data frame with the specified name.

Method `countTotalData()`**:** Count the number of "totals" data frames that have been added to the pivot table.

Usage:

```
PivotCalculator$countTotalData(dataName = NULL)
```

Arguments:

`dataName` The name of the data frame (as specified in 'pt$addData()') that the "totals" data frames are associated with.

Returns: The number of "totals" data frames associated with the specified name.

Method `getTotalDataFrame()`**:** Retrieve a "totals" data frame that was added to the pivot table.

Usage:

```
PivotCalculator$getTotalDataFrame(dataName = NULL, variableNames = NULL)
```

Arguments:

`dataName` The name of the data frame (as specified in 'pt$addData()') that the "totals" data frame is associated with.

`variableNames` The names of the variables that the totals are grouped by in the "totals" data frame (i.e. the dimensionality).

Returns: The "totals" data frame.

Method `getCalculationGroup()`**:** Retrieve a calculation group in the pivot table.

Usage:

```
PivotCalculator$getCalculationGroup(calculationGroupName = NULL)
```

Arguments:

`calculationGroupName` The name of the calculation group to retrieve.

Returns: The calculation group with the specified name.

Method `getCalculation()`**:** Retrieve a calculation in the pivot table.

Usage:

```
PivotCalculator$getCalculation(
  calculationGroupName = NULL,
  calculationName = NULL
)
```

Arguments:

`calculationGroupName` The name of the calculation group to retrieve.

`calculationName` The name of the calculation to retrieve.

Returns: The calculation with the specified name in the specified group.

Method `generateBatchesForCellEvaluation()`**:** Examine the data groups and cells in a pivot table to generate the structure of the batches in preparation for evaluating the pivot table.

Usage:

```
PivotCalculator$generateBatchesForCellEvaluation()
```

Returns: The batches that exist in the pivot table.

Method `evaluateBatches()`**:** Execute the batch calculations as part of evaluating the pivot table.

Usage:

```
PivotCalculator$evaluateBatches()
```

Returns: The number of batches that were evaluated.

Method `newFilter()`**:** Create a new 'PivotFilter' object associated with the specified data frame column name and column values. The new filter is conceptually of the form 'variableName

Usage:

```
PivotCalculator$newFilter(variableName = NULL, values = NULL)
```

Arguments:

`variableName` The data frame column name the filter is associated with.

`values` The filter values for the filter.

Returns: The new 'PivotFilter' object.

Method `newFilters()`**:** Create a new 'PivotFilters' object associated with the specified data frame column name and column values. The new filter is conceptually of the form 'variableName

Usage:

```
PivotCalculator$newFilters(variableName = NULL, values = NULL)
```

Arguments:

`variableName` The data frame column name the filter is associated with.

`values` The filter values for the filter.

Details: A 'PivotFilters' object is a collection of 'PivotFilter' objects, therefore the return value of this method is suitable for use where other filters will subsequently be needed/applied.

Returns: The new 'PivotFilter' object.

Method `setFilters()`**:** Combines two 'PivotFilters' objects, e.g. to intersect the filters coming from the row and column headings for a particular cell.

Usage:

```
PivotCalculator$setFilters(
  filters1 = NULL,
  filters2 = NULL,
  action = "replace"
)
```

Arguments:

`filters1` A 'PivotFilters' object.

`filters2` A 'PivotFilters' object.

`action` A character value specifying how to combine the two filters. Must be one of "intersect", "replace", "union".

Returns: A new 'PivotFilters' object.

Method `setFilter()`**:** Combines a 'PivotFilters' object with a 'PivotFilter' object.

Usage:

```
PivotCalculator$setFilter(filters = NULL, filter = NULL, action = "replace")
```

Arguments:

`filters` A 'PivotFilters' object.

`filter` A 'PivotFilters' object.

`action` A character value specifying how to combine the two filters. Must be one of "intersect", "replace", "union".

Returns: A new 'PivotFilters' object.

Method `setFilterValues()`**:** Combines a 'PivotFilters' object with additional filter criteria.

Usage:

```
PivotCalculator$setFilterValues(
  filters = NULL,
  variableName = NULL,
  values = NULL,
  action = "replace"
)
```

Arguments:

`filters` A 'PivotFilters' object.

`variableName` The name of the variable (i.e. column) in the data frame that the criteria relates to.

`values` The values that the specified variable will be filtered to.

`action` A character value specifying how to combine the existing filters and new filter criteria. Must be one of "intersect", "replace", "union".

Returns: A new 'PivotFilters' object.

Method `getFilteredDataFrame()`**:** Apply a set of filters to a data frame and return the filtered results.

Usage:

```
PivotCalculator$getFilteredDataFrame(dataFrame = NULL, filters = NULL)
```

Arguments:

`dataFrame` The data frame to filter.

`filters` A 'PivotFilters' object containing the filter criteria.

`dataName` The name of the data frame (as specified in 'pt$addData()') to be filtered.

Returns: A filtered data frame.

Method `getDistinctValues()`**:** Get the distinct values from a specified column in a data frame.

Usage:

```
PivotCalculator$getDistinctValues(dataFrame = NULL, variableName = NULL)
```

Arguments:

`dataFrame` The data frame.

`variableName` The name of the variable to get the distinct values for.

Returns: A vector containing the distinct values.

Method `formatValue()`**:** Format a value using a variety of different methods.

Usage:

```
PivotCalculator$formatValue(value = NULL, format = NULL, fmtFuncArgs = NULL)
```

Arguments:

`value` The value to format.

`format` Either a character format string to be used with 'sprintf()', a list of arguments to be used with 'base::format()' or a custom R function which will be invoked once per value to be formatted.

`fmtFuncArgs` If 'format' is a custom R function, then 'fmtFuncArgs' specifies any additional arguments (in the form of a list) that will be passed to the custom function.

Returns: The formatted value if 'format' is specified, otherwise the 'value' converted to a character value.

Method `getCombinedFilters()`**:** Get the working filters for a calculation by combining row-column filters and calculation filters.

Usage:

```
PivotCalculator$getCombinedFilters(
  rowColFilters = NULL,
  calcFilters = NULL,
  cell = NULL
)
```

Arguments:

`rowColFilters` A 'PivotFilters' object containing the combined filters from the row data groups and column data groups.

`calcFilters` Either 'PivotFilters' object or a 'PivotFilterOverrides' object containing filers defined as part of the calculation.

`cell` A 'PivotCell' object that is the cell for which the working data filters are being calculated.

Returns: A list of filters, element names: calculationFilters and workingFilters. The working filters are the row-column filters combined with the calculation filters.

Method `getFiltersForNamedCalculation()`**:** Get the working filters for a named calculation by calling 'getCombinedFilters()' as needed, depending on the calculation type.

Usage:

```
PivotCalculator$getFiltersForNamedCalculation(
  calculationName = NULL,
  calculationGroupName = NULL,
  rowColFilters = NULL,
  cell = NULL
)
```

Arguments:

`calculationName` The name of the calculation.

`calculationGroupName` The name of the calculation group.

`rowColFilters` A 'PivotFilters' object containing the combined filters from the row data groups and column data groups.

`cell` A 'PivotCell' object that is the cell for which the working data filters are being calculated.

Returns: A list of filters, where the element names are calculation names. Reminder: Evaluating a named calculation, if 'calc$type="calculation"', can involve computing multiple named calculations, which is why this return value is a list.

Method `setWorkingData()`**:** Set the working data filters for a cell in the pivot table.

Usage:

```
PivotCalculator$setWorkingData(cell = NULL)
```

Arguments:

`cell` The cell to generate the working data for.

Details: The working data for a cell is a list of 'PivotFilters' objects - one per named calculation. Most cells only relate to one calculation, but calculations of type 'calc$type="calculation"' can relate to multiple calculations, hence the working data is a list where the element name is the calculation name. This method calls 'getFiltersForNamedCalculation()' internally to generate the filters for the working data.

Returns: No return value.

Method `evaluateSingleValue()`**:** Get a single value from a data frame, as part of evaluating a calculation where the calculation is of type 'calc$type="value"'.

Usage:

```
PivotCalculator$evaluateSingleValue(
  dataFrame = NULL,
  workingFilters = NULL,
  valueName = NULL,
  format = NULL,
  fmtFuncArgs = NULL,
  noDataValue = NULL,
  noDataCaption = NULL
)
```

Arguments:

`dataFrame` The data frame to retrieve the value from.

`workingFilters` The relevant working data for the calculation.

`valueName` The name of the variable to retrieve from the data frame.

`format` The formatting to apply to the value. See 'formatValue()' for details.

`fmtFuncArgs` Additional arguments for a custom format function. See 'formatValue()' for details.

`noDataValue` A replacement raw value to use if the value is NULL.

`noDataCaption` A replacement formatted value to use if the value is NULL.

Returns: A list containing two elements: rawValue (typically numeric) and formattedValue (typically a character value).

Method `evaluateSummariseExpression()`**:** Get a summary value from a data frame, as part of evaluating a calculation where the calculation is of type 'calc$type="summary"'.

Usage:

```
PivotCalculator$evaluateSummariseExpression(
  dataName = NULL,
  dataFrame = NULL,
  workingFilters = NULL,
  batchName = NULL,
  calculationName = NULL,
  calculationGroupName = NULL,
  summaryName = NULL,
  summariseExpression = NULL,
  format = NULL,
  fmtFuncArgs = NULL,
  noDataValue = NULL,
  noDataCaption = NULL
)
```

Arguments:

`dataName` The name of the data frame (as specified in 'pt$addData()') containing the data.

`dataFrame` The data frame to retrieve the value from.

`workingFilters` The relevant working data for the calculation.

`batchName` The name of the batch that contains the results of the calculation (if batch evaluation is in use and possible for this cell and calculation).

`calculationName` The name of the calculation.

`calculationGroupName` The name of the calculation group.

`summaryName` The name of the summary (typically also the calculation name).

`summariseExpression` The dplyr or data.table expression to aggregate and summarise the data.

`format` The formatting to apply to the value. See 'formatValue()' for details.

`fmtFuncArgs` Additional arguments for a custom format function. See 'formatValue()' for details.

`noDataValue` A replacement raw value to use if the value is NULL.

`noDataCaption` A replacement formatted value to use if the value is NULL.

Details: Where batch evaluation is used, the value is retrieved from the pre-calculated batch, otherwise dplyr/data.table is used to calculate the value (i.e. reverting to sequential evaluation mode which performs calculations cell-by-cell, one cell at a time).

Returns: A list containing two elements: rawValue (typically numeric) and formattedValue (typically a character value).

Method `evaluateCalculationExpression()`**:** Evaluates an R expression in order to combine the results of other calculations, as part of evaluating a calculation where the calculation is of type 'calc$type="calculation"'.

Usage:

```
PivotCalculator$evaluateCalculationExpression(
  values = NULL,
  calculationExpression = NULL,
  format = NULL,
  fmtFuncArgs = NULL,
  noDataValue = NULL,
  noDataCaption = NULL
)
```

Arguments:

`values` The results of other calculations, passed in the form of a list where the element names are the names of those other calculations.

`calculationExpression` A character expression to be evaluated, e.g. "values$TotalIncome/values$SaleCount".

`format` The formatting to apply to the value. See 'formatValue()' for details.

`fmtFuncArgs` Additional arguments for a custom format function. See 'formatValue()' for details.

`noDataValue` A replacement raw value to use if the value is NULL.

`noDataCaption` A replacement formatted value to use if the value is NULL.

Details: A calculation, where 'calc$type="calculation"', combines the results of other calculations using a simple R expression.

Returns: A list containing two elements: rawValue (typically numeric) and formattedValue (typically a character value).

Method `evaluateCalculateFunction()`**:** Invokes a user-provided custom R function to aggregate data and perform calculations, as part of evaluating a calculation where the calculation is of type 'calc$type="function"'.

Usage:

```
PivotCalculator$evaluateCalculateFunction(
  workingFilters = NULL,
  calculationFunction = NULL,
  calcFuncArgs = NULL,
  format = NULL,
  fmtFuncArgs = NULL,
  baseValues = NULL,
  cell = NULL
)
```

Arguments:

`workingFilters` The relevant working data for the calculation.

`calculationFunction` The custom R function to invoke.

`calcFuncArgs` Specifies any additional arguments (in the form of a list) that will be passed to the custom calculation function.

`format` The formatting to apply to the value. See 'formatValue()' for details.

`fmtFuncArgs` Additional arguments for a custom format function. See 'formatValue()' for details.

`baseValues` The results of other calculations, passed in the form of a list where the element names are the names of those other calculations.

`cell` A 'PivotCell' object representing the cell being calculated.

Details: A calculation, where 'calc$type="function"', invokes a user provided R function on a cell-by-cell basis.

Returns: A list containing two elements: rawValue (typically numeric) and formattedValue (typically a character value).

Method `evaluateNamedCalculationWD()`**:** Invokes the relevant calculation function based upon the calculation type.

Usage:

```
PivotCalculator$evaluateNamedCalculationWD(
  calculationName = NULL,
  calculationGroupName = NULL,
  workingData = NULL,
  cell = NULL
)
```

Arguments:

`calculationName` The name of the calculation to execute.

`calculationGroupName` The calculation group that the calculation belongs to.

`workingData` The relevant working data for the calculation.

`cell` A 'PivotCell' object representing the cell being calculated.

Details: This function examines the 'calc$type' property then invokes either 'evaluateSingleValue()', 'evaluateSummariseExpression()', 'evaluateCalculationExpression()' or 'evaluateCalculateFunction()'. Sometimes, more than one of the these functions is invoked, since calculation type "calculation" and "function" can/do make use of values from other calculations, which must be evaluated first.

Returns: A list containing two elements: rawValue (typically numeric) and formattedValue (typically a character value).

Method `evaluateNamedCalculation()`**:** Invokes the relevant calculation function based upon the calculation type.

Usage:

```
PivotCalculator$evaluateNamedCalculation(
  calculationName = NULL,
  calculationGroupName = NULL,
  rowColFilters = NULL
)
```

Arguments:

`calculationName` The name of the calculation to execute.

`calculationGroupName` The calculation group that the calculation belongs to.

`rowColFilters` The filters arising from the row and column groups.

Details: This function is a higher-level wrapper around 'evaluateNamedCalculationWD()'. This version incorporates logic to convert the filters from the row and column data groups into the working data filters, then calls 'evaluateNamedCalculationWD()'. This version has no suffix in the name, since this is the version users are more likely to invoke, e.g. from within a custom calculation function.

Returns: A list containing two elements: rawValue (typically numeric) and formattedValue (typically a character value).

Method `evaluateCell()`**:** Evaluate calculations to compute the value of a cell in a pivot table.

Usage:

```
PivotCalculator$evaluateCell(cell = NULL)
```

Arguments:

`cell` A 'PivotCell' object representing the cell to calculate.

Returns: A list containing two elements: rawValue (typically numeric) and formattedValue (typically a character value).

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PivotCalculator$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# This class should only be created by the pivot table.
# It is not intended to be created outside of the pivot table.
```

`PivotCell` *R6 class that represents a cell in a pivot table.*

Description

The 'PivotCell' class represents a cell in the body of a pivot table (i.e. not a row/column heading, rather a cell typically containing a numerical value).

Format

`R6Class` object.

Active bindings

`instanceId` An integer value that uniquely identifies this cell. NB: This number is guaranteed to be unique within the pivot table, but the method of generation of the values may change in future, so you are advised not to base any logic on specific values.

`rowNumber` The row number of the cell. 1 = the first (i.e. top) data row.

`columnNumber` The column number of the cell. 1 = the first (i.e. leftmost) data column.

`calculationName` The name of the calculation that is displayed in the cell.

`calculationGroupName` The name of the calculation group that owns the calculation.

`isEmpty` 'TRUE' if this cell contains no data (e.g. if it is part of a header / outline row), 'FALSE' otherwise.

`rowFilters` A 'PivotFilters' object containing the filters applied to this cell from the row data groups (i.e. row headings).

`columnFilters` A 'PivotFilters' object containing the filters applied to this cell from the column data groups (i.e. column headings).

`rowColFilters` A 'PivotFilters' object containing the combined filters applied to this cell from both the row and column data groups.

`calculationFilters` The set of filters that apply to this cell to support calculation logic. Either a 'PivotFilters' object or a 'PivotFilterOverrides' object. See the "Appendix: Calculations" vignette for details.

`workingData` A list of filter objects that results when the 'rowColFilters' and 'calculationFilters' are combined prior to calculating the cell value. This is a list since some cells involve multiple calculations - where 'calc$type' is "calculation" or "function", the calculation can be based on the values of other calculations.

`evaluationFilters` The same as 'workingData' generally, except when custom calculation functions modify the filters whilst executing.

`rowLeafGroup` The row data group linked to this row.

`columnLeafGroup` The column data group linked to this column.

`isTotal` 'TRUE' is this cell is a total, 'FALSE' otherwise-

`rawValue` The raw cell value - i.e. unformatted, typically a numeric value.

`formattedValue` The formatted value - typically a character value.

`baseStyleName` The name of the style that defines the visual appearance of the cell.

`style` A 'PivotStyle' object that assists in managing the CSS style declarations that override the base style.

Methods

Public methods:

- `PivotCell$new()`
- `PivotCell$setStyling()`
- `PivotCell$getCopy()`
- `PivotCell$asList()`
- `PivotCell$asJSON()`
- `PivotCell$clone()`

Method `new()`**:** Create a new 'PivotCell' object.

Usage:

```
PivotCell$new(
  parentPivot,
  rowNumber = NULL,
  columnNumber = NULL,
  calculationName = NULL,
```

```
  calculationGroupName = NULL,
  isEmpty = FALSE,
  rowFilters = NULL,
  columnFilters = NULL,
  rowColFilters = NULL,
  rowLeafGroup = NULL,
  columnLeafGroup = NULL
)
```

Arguments:

`parentPivot` The pivot table that this 'PivotCell' instance belongs to.

`rowNumber` The row number of the cell. 1 = the first (i.e. top) data row.

`columnNumber` The column number of the cell. 1 = the first (i.e. leftmost) data column.

`calculationName` The name of the calculation that is displayed in the cell.

`calculationGroupName` The name of the calculation group that owns the calculation.

`isEmpty` 'TRUE' if this cell contains no data (e.g. if it is part of a header / outline row), 'FALSE' otherwise.

`rowFilters` A 'PivotFilters' object containing the filters applied to this cell from the row data groups (i.e. row headings).

`columnFilters` A 'PivotFilters' object containing the filters applied to this cell from the column data groups (i.e. column headings).

`rowColFilters` A 'PivotFilters' object containing the combined filters applied to this cell from both the row and column data groups.

`rowLeafGroup` The row data group linked to this row.

`columnLeafGroup` The column data group linked to this column.

Returns: A new 'PivotCell' object.

Method `setStyling()`**:** An internal method used to set style declarations on the cell. Using 'pt$setStyling(cells=x)' is preferred for users.

Usage:

```
PivotCell$setStyling(styleDeclarations = NULL)
```

Arguments:

`styleDeclarations` A list containing CSS style declarations.

Returns: No return value.

Method `getCopy()`**:** Non-functional legacy method soon to be removed.

Usage:

```
PivotCell$getCopy()
```

Returns: Returns an empty list.

Method `asList()`**:** Return the contents of this object as a list for debugging.

Usage:

```
PivotCell$asList()
```

Returns: A list of various object properties.

Method `asJSON()`**:** Return the contents of this object as JSON for debugging.

Usage:

```
PivotCell$asJSON()
```

Returns: A JSON representation of various object properties.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PivotCell$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# This class should only be created by the pivot table.
# It is not intended to be created outside of the pivot table.
```

`PivotCells` *R6 class that contains the cells in a pivot table.*

Description

The 'PivotCells' class contains all of the 'PivotCell' instances that comprise the body of a pivot table.

Format

`R6Class` object.

Active bindings

`rowCount` The number of rows in the pivot table (excluding column headings).

`columnCount` The number of columns in the pivot table (excluding column headings).

`rowGroups` A list of the leaf-level data groups on the rows axis.

`columnGroups` A list of the leaf-level data groups on the columns axis.

`rows` A list of the rows in the pivot table. Each element in this list is a list of 'PivotCell' objects comprising the row.

`all` A list of the cells in the pivot table. Each element in this list is a 'PivotCell' object.

Methods

Public methods:

- PivotCells$new()
- PivotCells$reset()
- PivotCells$getColumnGroup()
- PivotCells$getRowGroup()
- PivotCells$setGroups()
- PivotCells$getCell()
- PivotCells$setCell()
- PivotCells$getCells()
- PivotCells$findCells()
- PivotCells$findGroupColumnNumbers()
- PivotCells$findGroupRowNumbers()
- PivotCells$getColumnWidths()
- PivotCells$removeColumn()
- PivotCells$removeColumns()
- PivotCells$removeRow()
- PivotCells$removeRows()
- PivotCells$asMatrix()
- PivotCells$asList()
- PivotCells$asJSON()
- PivotCells$clone()

Method `new()`**:** Create a new 'PivotCells' object.

Usage:

```
PivotCells$new(parentPivot = NULL)
```

Arguments:

`parentPivot` The pivot table that this 'PivotCells' instance belongs to.

Returns: A new 'PivotCells' object.

Method `reset()`**:** Remove all cells from the pivot table and reset row and column counts back to zero.

Usage:

```
PivotCells$reset()
```

Returns: No return value.

Method `getColumnGroup()`**:** Get the leaf-level data group that is associated with a specific column or columns in the pivot table.

Usage:

```
PivotCells$getColumnGroup(c = NULL)
```

Arguments:

`c` The column number or numbers. The first column is column 1, excluding the column(s) associated with row-headings.

Returns: A 'PivotDataGroup' that is associated with the specified column.

Method `getRowGroup()`**:** Get the leaf-level data group that is associated with a specific row or rows in the pivot table.

Usage:

```
PivotCells$getRowGroup(r = NULL)
```

Arguments:

`r` The row number or numbers. The first row is row 1, excluding the row(s) associated with column-headings.

Returns: A 'PivotDataGroup' that is associated with the specified row

Method `setGroups()`**:** An internal method used when building the cell structure of the pivot table.

Usage:

```
PivotCells$setGroups(rowGroups = NULL, columnGroups = NULL)
```

Arguments:

`rowGroups` A list of 'PivotDataGroup' objects to be set as the leaf-level row groups in the pivot table.

`columnGroups` A list of 'PivotDataGroup' objects to be set as the leaf-level column groups in the pivot table.

Returns: No return value.

Method `getCell()`**:** Get the cell at the specified row and column coordinates in the pivot table.

Usage:

```
PivotCells$getCell(r = NULL, c = NULL)
```

Arguments:

`r` Row number of the cell to retrieve.

`c` Column number of the cell to retrieve.

Details: The row and column numbers refer only to the cells in the body of the pivot table, i.e. row and column headings are excluded, e.g. row 1 is the first row of cells underneath the column headings.

Returns: A 'PivotCell' object representing the cell.

Method `setCell()`**:** Set the cell at the specified row and column coordinates in the pivot table.

Usage:

```
PivotCells$setCell(r, c, cell)
```

Arguments:

`r` Row number of the cell to retrieve.

`c` Column number of the cell to retrieve.

`cell` A 'PivotCell' object to set into the pivot table cells.

Details: This method is intended for internal package use only, used when building # the cell structure. The row and column numbers refer only to the cells in the body of the pivot table, i.e. row and column headings are excluded, e.g. row 1 is the first row of cells underneath the column headings.

Returns: No return value.

Method `getCells()`**:** Retrieve cells by a combination of row and/or column numbers. See the "Finding and Formatting" vignette for graphical examples.

Usage:

```
PivotCells$getCells(
  specifyCellsAsList = TRUE,
  rowNumbers = NULL,
  columnNumbers = NULL,
  cellCoordinates = NULL,
  excludeEmptyCells = FALSE,
  groups = NULL,
  rowGroups = NULL,
  columnGroups = NULL,
  matchMode = "simple"
)
```

Arguments:

`specifyCellsAsList` Specify how cells are retrieved. Default 'TRUE'. More information is provided in the details section.

`rowNumbers` A vector of row numbers that specify the rows or cells to retrieve.

`columnNumbers` A vector of row numbers that specify the columns or cells to retrieve.

`cellCoordinates` A list of two-element vectors that specify the coordinates of cells to retrieve. Ignored when 'specifyCellsAsList=FALSE'.

`excludeEmptyCells` Default 'FALSE'. Specify 'TRUE' to exclude empty cells.

`groups` A 'PivotDataGroup' object or a list of 'PivotDataGroup' objects on either the rows or columns axes. The cells to be retrieved must be related to at least one of these groups.

`rowGroups` A 'PivotDataGroup' object or a list of 'PivotDataGroup' objects on the rows axis. The cells to be retrieved must be related to at least one of these row groups. If both 'rowGroups' and 'columnGroups' are specified, then the cells to be retrieved must be related to at least one of the specified row groups and one of the specified column groups.

`columnGroups` A 'PivotDataGroup' object or a list of 'PivotDataGroup' objects on the columns axis. The cells to be retrieved must be related to at least one of these column groups. If both 'rowGroups' and 'columnGroups' are specified, then the cells to be retrieved must be related to at least one of the specified row groups and one of the specified column groups.

`matchMode` Either "simple" (default) or "combinations"

"simple" specifies that row and column arguments are considered separately (logical OR), e.g. rowNumbers=1 and columnNumbers=2 will match all cells in row 1 and all cells in column 2.

"combinations" specifies that row and column arguments are considered together (logical AND), e.g. rowNumbers=1 and columnNumbers=2 will match only the cell single at location (1, 2).

Arguments 'rowNumbers', 'columnNumbers', 'rowGroups' and 'columnGroups' are affected by the match mode. All other arguments are not.

Details: When ‘specifyCellsAsList=TRUE‘ (the default):
Get one or more rows by specifying the row numbers as a vector as the rowNumbers argument and leaving the columnNumbers argument set to the default value of ‘NULL‘, or
Get one or more columns by specifying the column numbers as a vector as the columnNumbers argument and leaving the rowNumbers argument set to the default value of ‘NULL‘, or
Get one or more individual cells by specifying the cellCoordinates argument as a list of vectors of length 2, where each element in the list is the row and column number of one cell,
e.g. ‘list(c(1, 2), c(3, 4))‘ specifies two cells, the first located at row 1, column 2 and the second located at row 3, column 4.
When ‘specifyCellsAsList=FALSE‘:
Get one or more rows by specifying the row numbers as a vector as the rowNumbers argument and leaving the columnNumbers argument set to the default value of ‘NULL‘, or
Get one or more columns by specifying the column numbers as a vector as the columnNumbers argument and leaving the rowNumbers argument set to the default value of ‘NULL‘, or
Get one or more cells by specifying the row and column numbers as vectors for the rowNumbers and columnNumbers arguments, or
a mixture of the above, where for entire rows/columns the element in the other vector is set to ‘NA‘, e.g. to retrieve whole rows, specify the row numbers as the rowNumbers but set the corresponding elements in the columnNumbers vector to ‘NA‘.

Returns: A list of ‘PivotCell‘ objects.

Method `findCells()`**:** Find cells matching specified criteria. See the "Finding and Formatting" vignette for graphical examples.

Usage:

```
PivotCells$findCells(
  variableNames = NULL,
  variableValues = NULL,
  totals = "include",
  calculationNames = NULL,
  minValue = NULL,
  maxValue = NULL,
  exactValues = NULL,
  valueRanges = NULL,
  includeNull = TRUE,
  includeNA = TRUE,
  emptyCells = "include",
  outlineCells = "exclude",
  rowNumbers = NULL,
  columnNumbers = NULL,
  cellCoordinates = NULL,
  groups = NULL,
  rowGroups = NULL,
  columnGroups = NULL,
  rowColumnMatchMode = "simple",
  cells = NULL,
  lowN = NULL,
  highN = NULL
)
```

Arguments:

`variableNames` A character vector specifying the name/names of the variables to find. This is useful generally only in pivot tables with irregular layouts, since in regular pivot tables every cell is related to every variable.

`variableValues` A list specifying the variable names and values to find, e.g. 'variableValues=list("PowerType"=c("DMU", "HST"))'.
Specify "**" as the variable value to match totals for the specified variable.
Specify "!*" as the variable value to match non-totals for the specified variable.
NB: The totals/non-totals criteria above won't work when visual totals are used.

`totals` A word that specifies how totals are matched (overrides the finer settings above) - must be one of "include" (default), "exclude" or "only".

`calculationNames` A character vector specifying the name/names of the calculations to find.

`minValue` A numerical value specifying a minimum value threshold.

`maxValue` A numerical value specifying a maximum value threshold.

`exactValues` A vector or list specifying a set of allowed values.

`valueRanges` A vector specifying one or more value range expressions which the cell values must match. If multiple value range expressions are specified, then the cell value must match any of one the specified expressions.

`includeNull` Specify TRUE to include 'NULL' in the matched cells, FALSE to exclude 'NULL' values.

`includeNA` Specify TRUE to include 'NA' in the matched cells, FALSE to exclude 'NA' values.

`emptyCells` A word that specifies how empty cells are matched - must be one of "include" (default), "exclude" or "only".

`outlineCells` A word that specifies how outline cells are matched - must be one of "include", "exclude" (default) or "only".

`rowNumbers` A vector of row numbers that specify the rows or cells to constrain the search.

`columnNumbers` A vector of column numbers that specify the columns or cells to constrain the search.

`cellCoordinates` A list of two-element vectors that specify the coordinates of cells to constrain the search.

`groups` A 'PivotDataGroup' object or a list of 'PivotDataGroup' objects on either the rows or columns axes. The cells to be searched must be related to at least one of these groups.

`rowGroups` A 'PivotDataGroup' object or a list of 'PivotDataGroup' objects on the rows axis. The cells to be searched must be related to at least one of these row groups. If both 'rowGroups' and 'columnGroups' are specified, then the cells to be searched must be related to at least one of the specified row groups and one of the specified column groups.

`columnGroups` A 'PivotDataGroup' object or a list of 'PivotDataGroup' objects on the columns axis. The cells to be searched must be related to at least one of these column groups. If both 'rowGroups' and 'columnGroups' are specified, then the cells to be searched must be related to at least one of the specified row groups and one of the specified column groups.

`rowColumnMatchMode` Either "simple" (default) or "combinations":
"simple" specifies that row and column arguments are considered separately (logical OR), e.g. rowNumbers=1 and columnNumbers=2 will match all cells in row 1 and all cells in column 2.
"combinations" specifies that row and column arguments are considered together (logical

AND), e.g. rowNumbers=1 and columnNumbers=2 will match only the cell single at location (1, 2).
Arguments 'rowNumbers', 'columnNumbers', 'rowGroups' and 'columnGroups' are affected by the match mode. All other arguments are not.

`cells` A 'PivotCell' object or a list of 'PivotCell' objects to constrain the scope of the search.

`lowN` Find the first N cells (ascending order, lowest values first).

`highN` Find the last N cells (descending order, highest values first).

Returns: A list of 'PivotCell' objects.

Method `findGroupColumnNumbers()`**:** Find the column numbers associated with a specific data group or groups.

Usage:

```
PivotCells$findGroupColumnNumbers(group = NULL, collapse = FALSE)
```

Arguments:

`group` A 'PivotDataGroup' in the column data groups (i.e. a column heading) or a list of column data groups..

`collapse` A logical value specifying whether the return value should be simplified. See details.

Details: If 'group' is a list: If 'collapse' is 'FALSE', then a list of vectors is returned, if 'collapse' is 'TRUE', then a single combined vector is returned.

Returns: Either a vector of column numbers related to the single specified group or a list of vectors containing column numbers related to the specified groups.

Method `findGroupRowNumbers()`**:** Find the row numbers associated with a specific data group or groups.

Usage:

```
PivotCells$findGroupRowNumbers(group = NULL, collapse = FALSE)
```

Arguments:

`group` A 'PivotDataGroup' in the row data groups (i.e. a row heading) or a list of row data groups.

`collapse` A logical value specifying whether the return value should be simplified. See details.

Details: If 'group' is a list: If 'collapse' is 'FALSE', then a list of vectors is returned, if 'collapse' is 'TRUE', then a single combined vector is returned.

Returns: Either a vector of row numbers related to the single specified group or a list of vectors containing row numbers related to the specified groups.

Method `getColumnWidths()`**:** Retrieve the width (in characters) of the longest value in each column.

Usage:

```
PivotCells$getColumnWidths()
```

Returns: A vector containing the length of the longest value in each column.

Method `removeColumn()`**:** Remove a column from the pivot table.

Usage:

```
PivotCells$removeColumn(c = NULL, renumberGroups = TRUE)
```

Arguments:

`c` The column number. The first column is column 1, excluding the column(s) associated with row-headings.

`renumberGroups` 'TRUE' (default) to renumber the 'rowColumnNumber' property of the data groups after removing the row.

Details: This method removes both the related column group and cells.

Returns: No return value.

Method `removeColumns()`**:** Remove multiple column from the pivot table.

Usage:

```
PivotCells$removeColumns(columnNumbers = NULL, renumberGroups = TRUE)
```

Arguments:

`columnNumbers` The column numbers. The first column is column 1, excluding the column(s) associated with row-headings.

`renumberGroups` 'TRUE' (default) to renumber the 'rowColumnNumber' property of the data groups after removing the row.

Details: This method removes both the related column groups and cells.

Returns: No return value.

Method `removeRow()`**:** Remove a row from the pivot table.

Usage:

```
PivotCells$removeRow(r = NULL, renumberGroups = TRUE)
```

Arguments:

`r` The row number. The first row is row 1, excluding the row(s) associated with column-headings.

`renumberGroups` 'TRUE' (default) to renumber the 'rowColumnNumber' property of the data groups after removing the row.

Details: This method removes both the related row group and cells.

Returns: No return value.

Method `removeRows()`**:** Remove multiple rows from the pivot table.

Usage:

```
PivotCells$removeRows(rowNumbers = NULL, renumberGroups = TRUE)
```

Arguments:

`rowNumbers` The row numbers. The first row is row 1, excluding the rows(s) associated with column-headings.

`renumberGroups` 'TRUE' (default) to renumber the 'rowColumnNumber' property of the data groups after removing the row.

Details: This method removes both the related row groups and cells.

Returns: No return value.

Method `asMatrix()`**:** Get a matrix containing all of the values from the body of the pivot table.

Usage:

```
PivotCells$asMatrix(rawValue = TRUE)
```

Arguments:

`rawValue` 'TRUE' (default) to populate the matrix with the numerical raw values, 'FALSE' to populate the matrix with the character formatted values.

Returns: A 'matrix' containing the values from the body of the pivot table.

Method `asList()`**:** Return the contents of this object as a list for debugging.

Usage:

```
PivotCells$asList()
```

Returns: A list of various object properties.

Method `asJSON()`**:** Return the contents of this object as JSON for debugging.

Usage:

```
PivotCells$asJSON()
```

Returns: A JSON representation of various object properties.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PivotCells$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# This class should only be created by the pivot table.
# It is not intended to be created outside of the pivot table.
```

`PivotData` *R6 class that contains named data frames and associated totals.*

Description

The PivotData class stores all of the data frames associated with a pivot table. Each data frame can have a set of associated "totals" data frames, which are used to enable the "value" calculation type.

Format

`R6Class` object.

Active bindings

`count` The number of named data frames in the pivot table (excluding totals/aggregate data frames).

`defaultData` The default data frame in the pivot table.

`defaultName` The name of the default data frame in the pivot table.

Methods

Public methods:

- `PivotData$new()`
- `PivotData$addData()`
- `PivotData$getData()`
- `PivotData$isKnownData()`
- `PivotData$addTotalData()`
- `PivotData$countTotalData()`
- `PivotData$getTotalData()`
- `PivotData$asList()`
- `PivotData$asJSON()`
- `PivotData$clone()`

Method `new()`**:** Create a new 'PivotData' object.

Usage:

```
PivotData$new(parentPivot = NULL)
```

Arguments:

`parentPivot` The pivot table that this 'PivotData' instance belongs to.

Returns: A new 'PivotData' object.

Method `addData()`**:** Add a data frame to the pivot table, specifying a name that can be used later to easily retrieve it or refer to it.

Usage:

```
PivotData$addData(dataFrame = NULL, dataName = NULL)
```

Arguments:

`dataFrame` The data frame to add to the pivot table.

`dataName` The name to assign to this data frame in the pivot table. If no name is specified, then the name of the data frame variable will be used.

Returns: No return value.

Method `getData()`**:** Retrieve the data frame with the specified name.

Usage:

```
PivotData$getData(dataName = NULL)
```

Arguments:

`dataName` The name that was assigned to the data frame when it was added to the pivot table.

Returns: A data frame.

Method `isKnownData()`**:** Check if a data frame exists with the specified name.

Usage:

```
PivotData$isKnownData(dataName = NULL)
```

Arguments:

`dataName` The name that was assigned to the data frame when it was added to the pivot table.

Returns: 'TRUE' if a data frame exists with the specified name, 'FALSE' otherwise.

Method `addTotalData()`**:** Add pre-calculated totals/aggregate data to the pivot table.

Usage:

```
PivotData$addTotalData(dataFrame = NULL, dataName = NULL, variableNames = NULL)
```

Arguments:

`dataFrame` The data frame to add to the pivot table.

`dataName` The name of the associated data frame in the pivot table which these totals relate to.

`variableNames` A character vector specifying the names of the variables which these totals are grouped by.

Returns: No return value.

Method `countTotalData()`**:** Count the number of data frames containing total/aggregate data that exist in the pivot table associated with a specific named data frame.

Usage:

```
PivotData$countTotalData(dataName = NULL)
```

Arguments:

`dataName` The name of the associated data frame in the pivot table which these totals relate to.

Returns: The number of total/aggregate data frames that exist in the pivot table associated with the specified data frame name.

Method `getTotalData()`**:** Retrieve pre-calculated totals/aggregate data from the pivot table.

Usage:

```
PivotData$getTotalData(dataName = NULL, variableNames = NULL)
```

Arguments:

`dataName` The name of the associated data frame in the pivot table which these totals relate to.

`variableNames` A character vector specifying the names of the variables which the totals are grouped by.

Returns: A data frame.

Method `asList()`**:** Return the contents of this object as a list for debugging.

Usage:

```
PivotData$asList()
```

Returns: A list of various object properties.

Method `asJSON()`**:** Return the contents of this object as JSON for debugging.

Usage:

```
PivotData$asJSON()
```

Returns: A JSON representation of various object properties.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PivotData$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# This class should only be created by the pivot table.
# It is not intended to be created outside of the pivot table.
```

`PivotDataGroup` *R6 class that defines a row or column data group.*

Description

The 'PivotDataGroup' class represents one row or column heading in a pivot table. Data groups exist in a hierarchy and have a parent-child relationship, i.e. each 'PivotDataGroup' instance can have one or more child data groups.

Format

`R6Class` object.

Active bindings

`instanceId` An integer value that uniquely identifies this group. NB: This number is guaranteed to be unique within the pivot table, but the method of generation of the values may change in future, so you are advised not to base any logic on specific values.

`rowOrColumn` Either "row" or "column"

`parentGroup` The parent 'PivotDataGroup' instance that this 'PivotDataGroup' instance belongs to.

`childGroups` A list of 'PivotDataGroup' objects that are the children of this data group.

`childGroupCount` A count of 'PivotDataGroup' objects that are the children of this data group.

`leafGroups` A list of 'PivotDataGroup' objects that are leaf-level descendants of this data group.

`levelNumber` An integer value specifying the level number where this data group exists in the hierarchy.

`filters` A 'PivotFilters' object containing the filters associated with this data group.

`variableName` A character value that specifies the name of the variable in the data frame that this group relates to and will filter.

`values` A vector that specifies the filter values applied to ‘variableName‘ to select the data to match this row/column in the pivot table.

`calculationGroupName` For calculation groups, this character value specifies the calculation group that ‘calculationName‘ belongs to.

`calculationName` For calculation groups, this character value specifies the name of the calculation.

`doNotExpand` ‘TRUE‘ if this data group prevent the high-level methods such as ‘addDataGroups()‘ from adding child groups.

`isEmpty` ‘TRUE‘ if this group contains no data (e.g. if it is part of a header or outline row)

`isOutline` ‘TRUE‘ if this data group is an outline group.

`styleAsOutline` ‘TRUE‘ if this data group is to be styled as an outline group. Only applicable when ‘isOutline‘ is ‘TRUE‘.

`outlineLinkedGroupId` The instance id of the child group related to this group, if this group is an outline group.

`outlineLinkedGroupExists` ‘TRUE‘ if the group specified by ‘outlineLinkedGroupId‘ still exists.

`captionTemplate` A character value that specifies the template for the data group caption, default "{values}".

`caption` The data group caption. Assigning a caption effectively overrides the built-in logic for generating a caption.

`sortValue` The raw (i.e. unformatted, typically numerical) value that represents this data group in sort operations.

`isTotal` ‘TRUE‘ if this data group is a total.

`isLevelSubTotal` ‘TRUE‘ if this data group is a sub-total.

`isLevelTotal` ‘TRUE‘ if this data group is a level-total.

`rowColumnNumber` The row or column number that this data group relates to. This property only has a value for leaf-level data groups.

`baseStyleName` The style name for the data group.

`style` A ‘PivotStyle‘ object that contains additional CSS style declarations that override the base style.

`mergeEmptySpace` A logical value that specifies whether empty space should be merged.

`cellBaseStyleName` The style name for cells related to this data group.

`netCellBaseStyleName` The style name for cells related to this data group - either from this group or the first ancestor that specifies a cellBaseStyleName if cellBaseStyleName is not specified on this group.

`cellStyle` A ‘PivotStyle‘ object that contains additional CSS style declarations that override the base style for cells related to this data group. If setting this property, a list can also be specified.

`netCellStyle` A ‘PivotStyle‘ object that contains additional CSS style declarations that override the base style for cells related to this data group - both from this group and all ancestors.

`fixedWidthSize` The width (in characters) needed for this data group when rendering to plain text.

`isMatch` An internal property used when finding data groups.

`isRendered` An internal property used when rendering data groups.

`isWithinVisibleRange` An internal property used when rendering data groups.

`visibleChildGroupCount` An internal property used when rendering data groups.

`visibleDescendantGroupCount` An internal property used when rendering data groups.

`visibleLeafGroupCount` An internal property used when rendering data groups.

`sortAnchor` Used to specify sort behaviour for outline groups, must be one of "fixed", "next" or "previous".

`sortGroupsBefore` An internal property used when sorting data groups.

`sortGroupsAfter` An internal property used when sorting data groups.

Methods

Public methods:

- `PivotDataGroup$new()`
- `PivotDataGroup$getLevelNumber()`
- `PivotDataGroup$getAncestorGroups()`
- `PivotDataGroup$getDescendantGroups()`
- `PivotDataGroup$getLeafGroups()`
- `PivotDataGroup$getLevelCount()`
- `PivotDataGroup$getLevelGroups()`
- `PivotDataGroup$getRelatedOutlineGroups()`
- `PivotDataGroup$getChildIndex()`
- `PivotDataGroup$findChildIndex()`
- `PivotDataGroup$addChildGroup()`
- `PivotDataGroup$removeChildGroup()`
- `PivotDataGroup$removeGroup()`
- `PivotDataGroup$addDataGroups()`
- `PivotDataGroup$sortDataGroups()`
- `PivotDataGroup$addCalculationGroups()`
- `PivotDataGroup$normaliseDataGroup()`
- `PivotDataGroup$getNetFilters()`
- `PivotDataGroup$getNetCalculationName()`
- `PivotDataGroup$isFindMatch()`
- `PivotDataGroup$findDataGroups()`
- `PivotDataGroup$setStyling()`
- `PivotDataGroup$clearSortGroups()`
- `PivotDataGroup$addSortGroupBefore()`
- `PivotDataGroup$addSortGroupAfter()`
- `PivotDataGroup$asList()`
- `PivotDataGroup$asJSON()`
- `PivotDataGroup$clone()`

Method `new()`**:** Create a new ‘PivotDataGroup‘ object.

Usage:

```
PivotDataGroup$new(
  parentGroup = NULL,
  parentPivot = NULL,
  rowOrColumn = NULL,
  doNotExpand = FALSE,
  isEmpty = FALSE,
  isOutline = FALSE,
  styleAsOutline = FALSE,
  captionTemplate = "{value}",
  caption = NULL,
  isTotal = FALSE,
  isLevelSubTotal = FALSE,
  isLevelTotal = FALSE,
  variableName = NULL,
  filterType = "ALL",
  values = NULL,
  calculationGroupName = NULL,
  calculationName = NULL,
  baseStyleName = NULL,
  styleDeclarations = NULL,
  mergeEmptySpace = NULL,
  cellBaseStyleName = NULL,
  cellStyleDeclarations = NULL,
  sortAnchor = NULL,
  outlineLinkedGroupId = NULL
)
```

Arguments:

`parentGroup` The parent ‘PivotDataGroup‘ instance that this ‘PivotDataGroup‘ instance belongs to.

`parentPivot` The pivot table that this ‘PivotDataGroup‘ instance belongs to.

`rowOrColumn` Either "row" or "column" indicating which axis this data group exists on.

`doNotExpand` Default value ‘FALSE‘ - specify ‘TRUE‘ to prevent the high-level methods such as ‘addDataGroups()‘ from adding child groups.

`isEmpty` Default value ‘FALSE‘, specify ‘TRUE‘ to mark that this group contains no data (e.g. if it is part of a header or outline row)

`isOutline` Default value ‘FALSE‘ - specify ‘TRUE‘ to mark that this data group is an outline group.

`styleAsOutline` Default value ‘FALSE‘ - specify ‘TRUE‘ to style this data group as an outline group. Only applicable when ‘isOutline‘ is ‘TRUE‘.

`captionTemplate` A character value that specifies the template for the data group caption, default "{values}".

`caption` Effectively a hard-coded caption that overrides the built-in logic for generating a caption.

`isTotal` Default ‘FALSE‘ - specify ‘TRUE‘ to mark that this data group is a total.

`isLevelSubTotal` Default ‘FALSE‘ - specify ‘TRUE‘ to mark that this data group is a sub-total within a level.

`isLevelTotal` Default 'FALSE' - specify 'TRUE' to mark that this data group is level total.

`variableName` A character value that specifies the name of the variable in the data frame that the group relates to and will filter.

`filterType` Must be one of "ALL", "VALUES", or "NONE" to specify the filter type: ALL means no filtering is applied. VALUEs is the typical value used to specify that 'variableName' is filtered to only 'values'. NONE means no data will match this data group.

`values` A vector that specifies the filter values applied to 'variableName' to select the data to match this row/column in the pivot table.

`calculationGroupName` For calculation groups, this character value specifies the calculation group that 'calculationName' belongs to.

`calculationName` For calculation groups, this character value specifies the name of the calculation.

`baseStyleName` The style name for the data group.

`styleDeclarations` A list of CSS style declarations to overlay on top of the base style.

`mergeEmptySpace` A character value that specifies how empty space should be merged. This is typically only used with outline groups (so applies to row groups only, not column groups). Must be one of "doNotMerge", "dataGroupsOnly", "cellsOnly", "dataGroupsAndCellsAs1" or "dataGroupsAndCellsAs2". See the "Regular Layout" vignette for more information.

`cellBaseStyleName` The style name for cells related to this data group.

`cellStyleDeclarations` A list of CSS style declarations to overlay on top of the base style for cells related to this data group

`sortAnchor` Used to specify sort behaviour for outline groups, must be one of "fixed", "next" or "previous".

`outlineLinkedGroupId` Used to link an outline group to the value data group which has the child data groups.

Returns: A new 'PivotDataGroup' object.

Method `getLevelNumber()`**:** Retrieve the level number in the hierarchy that the current data group exists at.

Usage:

```
PivotDataGroup$getLevelNumber()
```

Returns: An integer value specifying the level number where the data group exists.

Method `getAncestorGroups()`**:** Get all of the data groups above the current data group in the parent-child data group hierarchy.

Usage:

```
PivotDataGroup$getAncestorGroups(ancestors = NULL, includeCurrentGroup = FALSE)
```

Arguments:

`ancestors` A list containing ancestors closer to the current data group - to enable recursive execution of this function, or 'NULL' to begin with.

`includeCurrentGroup` Specify 'TRUE' to include the current group in the return value.

Returns: A list of data groups, where element 1 is the parent of the current group, element 2 is the grandparent of the current group, etc.

Method `getDescendantGroups()`**:** Get all of the data groups below the current data group in the parent-child data group hierarchy.

Usage:

```
PivotDataGroup$getDescendantGroups(
  descendants = NULL,
  includeCurrentGroup = FALSE
)
```

Arguments:

`descendants` A list containing descendants closer to the current data group - to enable recursive execution of this function, or 'NULL' to begin with.

`includeCurrentGroup` Specify 'TRUE' to include the current group in the return value.

Returns: A list of descendant data groups.

Method `getLeafGroups()`**:** Get all of the data groups below the current data group in the parent-child data group hierarchy.

Usage:

```
PivotDataGroup$getLeafGroups(leafGroups = NULL)
```

Arguments:

`leafGroups` A list containing other leaf-level groups - to enable recursive execution of this function, or 'NULL' to begin with.

Returns: A list of leaf-level data groups.

Method `getLevelCount()`**:** Count the number of levels in the data group hierarchy.

Usage:

```
PivotDataGroup$getLevelCount(includeCurrentLevel = FALSE)
```

Arguments:

`includeCurrentLevel` Default 'FALSE' to exclude the current level from the level count (since this method is most often called on the hidden root group).

Returns: The maximum number of levels in the hierarchy.

Method `getLevelGroups()`**:** Retrieve all of the data groups at a specific level in the data group hierarchy.

Usage:

```
PivotDataGroup$getLevelGroups(level = NULL, levelGroups = NULL)
```

Arguments:

`level` An integer specifying the level number. Level 0 represents the current data group.

`levelGroups` A list containing groups accumulated so far - to enable recursive execution of this function, or 'NULL' to begin with.

Returns: A list of data groups at the specified level in the hierarchy.

Method `getRelatedOutlineGroups()`**:** Retrieve the a list of the typically two or three related data groups that were created as one outlined group.

Usage:

```
PivotDataGroup$getRelatedOutlineGroups(group = NULL)
```

Arguments:

`group` The group to find the related outline groups.

Returns: A list of related outline data groups.

Method `getChildIndex()`**:** Get the index of a child group (or groups) in the current groups list of child groups.

Usage:

```
PivotDataGroup$getChildIndex(childGroup = NULL)
```

Arguments:

`childGroup` A single data group or a list of data groups that are children of the current group.

Returns: An integer vector.

Method `findChildIndex()`**:** Find the index of a child group (or groups) corresponding to the specified instance id(s) in the current groups list of child groups.

Usage:

```
PivotDataGroup$findChildIndex(childGroupInstanceId = NULL)
```

Arguments:

`childGroupInstanceId` An integer vector containing the instance ids of child groups of the current group.

Returns: An integer vector.

Method `addChildGroup()`**:** Add a new data group as a child of the current data group. The new group is added as the last child unless an index is specified.

Usage:

```
PivotDataGroup$addChildGroup(
  variableName = NULL,
  filterType = "ALL",
  values = NULL,
  doNotExpand = FALSE,
  isEmpty = FALSE,
  isOutline = FALSE,
  styleAsOutline = FALSE,
  captionTemplate = "{value}",
  caption = NULL,
  isTotal = FALSE,
  isLevelSubTotal = FALSE,
  isLevelTotal = FALSE,
  calculationGroupName = NULL,
  calculationName = NULL,
  baseStyleName = NULL,
  styleDeclarations = NULL,
  insertAtIndex = NULL,
  insertBeforeGroup = NULL,
  insertAfterGroup = NULL,
```

```
  mergeEmptySpace = NULL,
  cellBaseStyleName = NULL,
  cellStyleDeclarations = NULL,
  sortAnchor = NULL,
  outlineLinkedGroupId = NULL,
  resetCells = TRUE
)
```

Arguments:

`variableName` A character value that specifies the name of the variable in the data frame that the group relates to and will filter.

`filterType` Must be one of "ALL", "VALUES", or "NONE" to specify the filter type:
ALL means no filtering is applied.
VALUEs is the typical value used to specify that ‘variableName‘ is filtered to only ‘values‘.
NONE means no data will match this data group.

`values` A vector that specifies the filter values applied to ‘variableName‘ to select the data to match this row/column in the pivot table.

`doNotExpand` Default value ‘FALSE‘ - specify ‘TRUE‘ to prevent the high-level methods such as ‘addDataGroups()‘ from adding child groups.

`isEmpty` Default value ‘FALSE‘, specify ‘TRUE‘ to mark that this group contains no data (e.g. if it is part of a header or outline row)

`isOutline` Default value ‘FALSE‘ - specify ‘TRUE‘ to mark that this data group is an outline group.

`styleAsOutline` Default value ‘FALSE‘ - specify ‘TRUE‘ to style this data group as an outline group. Only applicable when ‘isOutline‘ is ‘TRUE‘.

`captionTemplate` A character value that specifies the template for the data group caption, default "{values}".

`caption` Effectively a hard-coded caption that overrides the built-in logic for generating a caption.

`isTotal` Default ‘FALSE‘ - specify ‘TRUE‘ to mark that this data group is a total.

`isLevelSubTotal` Default ‘FALSE‘ - specify ‘TRUE‘ to mark that this data group is a sub-total within a level.

`isLevelTotal` Default ‘FALSE‘ - specify ‘TRUE‘ to mark that this data group is level total.

`calculationGroupName` For calculation groups, this character value specifies the calculation group that ‘calculationName‘ belongs to.

`calculationName` For calculation groups, this character value specifies the name of the calculation.

`baseStyleName` The style name for the data group.

`styleDeclarations` A list of CSS style declarations to overlay on top of the base style.

`insertAtIndex` An integer that specifies the index in the list of child groups where the new group should be inserted.

`insertBeforeGroup` Specifies an existing group that the new group should be inserted before.

`insertAfterGroup` Specifies an existing group that the new group should be inserted after

`mergeEmptySpace` A character value that specifies how empty space should be merged. This is typically only used with outline groups (so applies to row groups only, not column groups). Must be one of "doNotMerge", "dataGroupsOnly", "cellsOnly", "dataGroupsAndCellsAs1" or "dataGroupsAndCellsAs2". See the "Regular Layout" vignette for more information.

`cellBaseStyleName` The style name for cells related to this data group.

`cellStyleDeclarations` A list of CSS style declarations to overlay on top of the base style for cells related to this data group

`sortAnchor` Used to specify sort behaviour for outline groups, must be one of "fixed", "next" or "previous".

`outlineLinkedGroupId` Used to link an outline group to the value data group which has the child data groups.

`resetCells` Default 'TRUE' to reset any cells that currently exist in the pivot table and trigger a recalculation of the pivot table when it is next rendered.

Returns: The new 'PivotDataGroup' object.

Method `removeChildGroup()`**:** Remove a data group that is a child of the current data group.

Usage:

```
PivotDataGroup$removeChildGroup(index = NULL, group = NULL, resetCells = TRUE)
```

Arguments:

`index` An index that specifies the location of the group to remove in the list of child groups.

`group` A 'PivotDataGroup' object to be removed. Only one of 'index' or 'group' needs to be specified.

`resetCells` Default 'TRUE' to reset any cells that currently exist in the pivot table and trigger a recalculation of the pivot table when it is next rendered.

Returns: No return value.

Method `removeGroup()`**:** Remove the current data group.

Usage:

```
PivotDataGroup$removeGroup(
  removeAncestorsIfNoRemainingChildren = FALSE,
  removedRelatedOutlineGroups = FALSE,
  resetCells = TRUE
)
```

Arguments:

`removeAncestorsIfNoRemainingChildren` Default 'FALSE' - specify 'TRUE' to recursively remove ancestor groups if they have no remaining child groups.

`removedRelatedOutlineGroups` Default 'FALSE' - specify 'TRUE' to remove related outline groups.

`resetCells` Default 'TRUE' to reset any cells that currently exist in the pivot table and trigger a recalculation of the pivot table when it is next rendered.

Returns: No return value.

Method `addDataGroups()`**:** Add multiple new data groups based on the distinct values in a data frame column or using explicitly specified data values. See the "Irregular Layout" vignette for example usage.

Usage:

```
PivotDataGroup$addDataGroups(
  variableName = NULL,
  atLevel = NULL,
  fromData = TRUE,
  dataName = NULL,
  dataSortOrder = "asc",
  customSortOrder = NULL,
  caption = "{value}",
  dataFormat = NULL,
  dataFmtFuncArgs = NULL,
  onlyCombinationsThatExist = TRUE,
  explicitListOfValues = NULL,
  calculationGroupName = NULL,
  expandExistingTotals = FALSE,
  addTotal = TRUE,
  visualTotals = FALSE,
  totalPosition = "after",
  totalCaption = "Total",
  onlyAddGroupIf = NULL,
  preGroupData = TRUE,
  baseStyleName = NULL,
  styleDeclarations = NULL,
  outlineBefore = NULL,
  outlineAfter = NULL,
  outlineTotal = FALSE,
  onlyAddOutlineChildGroupIf = NULL
)
```

Arguments:

`variableName` The name of the related column in the data frame(s) of the pivot table.

`atLevel` The number of levels below the current group to add the new groups. 0 = at the current level, 1 = one level below the current group, etc. 'NULL' = create a new level at the bottom of the hierarchy for the new groups.

`fromData` Default 'TRUE' to generate the new data groups based on the data values that exist in the 'variableName' column in the named data frame. If 'FALSE', then 'explicitListOfValues' must be specified.

`dataName` The name of the data frame (as specified in 'pt$addData()') to read the data group values from.

`dataSortOrder` Must be one of "asc", "desc", "custom" or "none".

`customSortOrder` A vector values sorted into the desired order.

`caption` The template of data group captions to generate, default "{value}".

`dataFormat` A character, list or custom function to format the data value.

`dataFmtFuncArgs` A list that specifies any additional arguments to pass to a custom format function.

`onlyCombinationsThatExist` Default 'TRUE' to generate only combinations of data groups that exist in the data frame.

`explicitListOfValues` A list of explicit values to create data groups from. A data group is created for each element of the list. If a list element is vector of values (with length greater

than 1), then a data group is created for multiple values instead of just a single value.

`calculationGroupName` The calculation group that the new data groups are related to.

`expandExistingTotals` Default 'FALSE', which means totals are not broken down in multi-level hierarchies.

`addTotal` Default 'TRUE', which means sub-total and total data groups are automatically added.

`visualTotals` Default 'FALSE', which means visual totals are disabled. See the "Data Groups" vignette for more details about visual totals.

`totalPosition` Either "before" or "after" to specify where total groups are created, default "after".

`totalCaption` The caption to display on total groups, default "Total".

`onlyAddGroupIf` A filter expression that can be used to more finely control whether data groups are created at different locations in the hierarchy. There must be at least one row that matches this filter and the filters from the ancestor groups in order that the child group is created. E.g. 'MaxDisplayLevel>5'.

`preGroupData` Default 'TRUE', which means that the pivot table pre-calculates the distinct combinations of variable values to reduce the CPU time and elapsed time required to generate data groups. Cannot be used in conjunction with the

`baseStyleName` The name of the style applied to this data group (i.e. this row/column heading). The style must exist in the 'PivotStyles' object associated with the PivotTable.

`styleDeclarations` CSS style declarations that can override the base style, expressed as a list, e.g. 'list("font-weight"=bold)'.

`outlineBefore` Default 'FALSE' to disable the creation of outline header groups. Specify either 'TRUE' or a list of outline group settings to create outline header groups. See the "Regular Layout" vignette for details.

`outlineAfter` Default 'FALSE' to disable the creation of outline footer groups. Specify either 'TRUE' or a list of outline group settings to create outline footer groups. See the "Regular Layout" vignette for details.

`outlineTotal` Default 'FALSE' to disable the creation of outline totals. Specify either 'TRUE' or a list of outline group settings to create outline totals. See the "Regular Layout" vignette for details.

`onlyAddOutlineChildGroupIf` A filter expression that can be used to more finely control whether outline child groups are created at different locations in the hierarchy. There must be at least one row that matches this filter and the filters from the ancestor groups in order that the outline child group is created. E.g. 'MaxDisplayLevel>5'. See the "Regular Layout" vignette for an example.

Details: There are broadly three different ways to call 'addDataGroups()':
(1) dataName=name, fromData=TRUE, onlyCombinationsThatExist=TRUE - which considers the ancestors of each existing data group to generate only those combinations of values that exist in the data frame.
(2) dataName=name, fromData=TRUE, onlyCombinationsThatExist=FALSE - which ignores the ancestors of each existing data group and simply adds every distinct value of the specified variable under every existing data group, which can result in combinations of values in the pivot table that don't exist in the data frame (i.e. blank rows/columns in the pivot table).
(3) fromData=FALSE, explicitListOfValues=list(...) - simply adds every value from the specified list under every existing data group.

Returns: A list of new 'PivotDataGroup' objects that have been added.

Method `sortDataGroups()`**:** Sort data groups either by the data group data value, caption, a custom order or based on calculation result values.

Usage:

```
PivotDataGroup$sortDataGroups(
  levelNumber = 1,
  orderBy = "calculation",
  customOrder = NULL,
  sortOrder = "desc",
  calculationGroupName = "default",
  calculationName = NULL,
  fromIndex = NULL,
  toIndex = NULL,
  resetCells = TRUE
)
```

Arguments:

`levelNumber` The number of levels below the current group to sort the data groups. 0 = at the current level, 1 = one level below the current group, etc.

`orderBy` Must be either "value", "caption", "calculation", "customByValue" or "customByCaption".
"value" sorts by the raw (i.e. unformatted) group value.
"caption" sorts by the formatted character group caption.
"calculation" sorts using one of the calculations defined in the pivot table. "customValue" sorts by the raw (i.e. unformatted) group value according to the specified custom sort order. "customCaption" sorts by the formatted character group caption according to the specified custom sort order.

`customOrder` A vector values sorted into the desired order.

`sortOrder` Must be either "asc" or "desc".

`calculationGroupName` If sorting using a calculation, the name of the calculation group containing the specified calculation.

`calculationName` If sorting using a calculation, the name of the calculation.

`fromIndex` A boundary to limit the sort operation.

`toIndex` A boundary to limit the sort operation.

`resetCells` Default 'TRUE' to reset any cells that currently exist in the pivot table and trigger a recalculation of the pivot table when it is next rendered.

Returns: No return value.

Method `addCalculationGroups()`**:** Add multiple new groups to the data group hierarchy to represent calculations.

Usage:

```
PivotDataGroup$addCalculationGroups(
  calculationGroupName = NULL,
  atLevel = NULL,
  outlineBefore = NULL,
  outlineAfter = NULL,
```

```
  resetCells = TRUE
)
```

Arguments:

`calculationGroupName` The name of the calculation group to add into the data group hierarchy.

`atLevel` The number of levels below the current group to add the new groups. 0 = at the current level, 1 = one level below the current group, etc. 'NULL' = create a new level at the bottom of the hierarchy for the new groups.

`outlineBefore` Default 'FALSE' to disable the creation of outline header groups. Specify either 'TRUE' or a list of outline group settings to create outline header groups. See the "Regular Layout" vignette for details.

`outlineAfter` Default 'FALSE' to disable the creation of outline footer groups. Specify either 'TRUE' or a list of outline group settings to create outline footer groups. See the "Regular Layout" vignette for details.

`resetCells` Default 'TRUE' to reset any cells that currently exist in the pivot table and trigger a recalculation of the pivot table when it is next rendered.

Returns: A list of new 'PivotDataGroup' objects that have been added.

Method `normaliseDataGroup()`**:** Normalise the data group hierarchy so that all branches have the same number of levels - accomplished by adding empty child data groups where needed.

Usage:

```
PivotDataGroup$normaliseDataGroup(resetCells = TRUE)
```

Arguments:

`resetCells` Default 'TRUE' to reset any cells that currently exist in the pivot table and trigger a recalculation of the pivot table when it is next rendered.

Returns: A list of new 'PivotDataGroup' objects that have been added.

Method `getNetFilters()`**:** Get a 'PivotFilters' object that contains the filters applied in this data group and all of its ancestors in the data group hierarchy.

Usage:

```
PivotDataGroup$getNetFilters()
```

Returns: A 'PivotFilters' object.

Method `getNetCalculationName()`**:** Get the calculation name set in this data group or its nearest ancestor.

Usage:

```
PivotDataGroup$getNetCalculationName()
```

Returns: The name of a calculation.

Method `isFindMatch()`**:** Test whether this data group matches specified criteria.

Usage:

```
PivotDataGroup$isFindMatch(
  matchMode = "simple",
  variableNames = NULL,
  variableValues = NULL,
  totals = "include",
  calculationNames = NULL,
  atLevels = NULL,
  minChildCount = NULL,
  maxChildCount = NULL,
  emptyGroups = "exclude",
  outlineGroups = "exclude",
  outlineLinkedGroupExists = NULL
)
```

Arguments:

`matchMode` Either "simple" (default) or "combinations".
"simple" is used when matching only one variable-value, multiple variable-value combinations are effectively logical "OR".
"combinations" is used when matching for combinations of variable values, multiple variable-value combinations are effectively logical "AND". A child group is viewed as having the variable-value filters of itself and it's parent/ancestors, e.g. 'list("TrainCategory"="Express Passenger", "PowerType"="DMU")', would return the "DMU" data group underneath "Express Passenger". See the "Finding and Formatting" vignette for graphical examples.

`variableNames` A character vector specifying the name/names of the variables to find. This is useful generally only in pivot tables with irregular layouts, since in regular pivot tables every cell is related to every variable.

`variableValues` A list specifying the variable names and values to find, e.g. 'variableValues=list("PowerType"=c("DMU", "HST"))'.
Specify "**" as the variable value to match totals for the specified variable.
Specify "!*" as the variable value to match non-totals for the specified variable.
NB: The totals/non-totals criteria above won't work when visual totals are used.

`totals` A word that specifies how totals are matched (overrides the finer settings above) - must be one of "include" (default), "exclude" or "only".

`calculationNames` A character vector specifying the name/names of the calculations to find.

`atLevels` An integer vector constraining the levels in the hierarchy to search.

`minChildCount` Match only data groups with this minimum number of children.

`maxChildCount` Match only data groups with this maximum number of children.

`emptyGroups` A word that specifies how empty groups are matched - must be one of "include", "exclude" (default) or "only".

`outlineGroups` A word that specifies how outline cells are matched - must be one of "include", "exclude" (default) or "only".

`outlineLinkedGroupExists` 'TRUE' to match only groups where the related outline child group still exists. 'FALSE' to match only groups where the related outline child group no longer exists.

Returns: 'TRUE' if this group matches the specified criteria, 'FALSE' otherwise.

Method `findDataGroups()`**:** Find data groups that match specified criteria.

Usage:

```
PivotDataGroup$findDataGroups(
  matchMode = "simple",
  variableNames = NULL,
  variableValues = NULL,
  totals = "include",
  calculationNames = NULL,
  atLevels = NULL,
  minChildCount = NULL,
  maxChildCount = NULL,
  emptyGroups = "exclude",
  outlineGroups = "exclude",
  outlineLinkedGroupExists = NULL,
  includeDescendantGroups = FALSE,
  includeCurrentGroup = TRUE
)
```

Arguments:

`matchMode` Either "simple" (default) or "combinations".
"simple" is used when matching only one variable-value, multiple variable-value combinations are effectively logical "OR".
"combinations" is used when matching for combinations of variable values, multiple variable-value combinations are effectively logical "AND". A child group is viewed as having the variable-value filters of itself and it's parent/ancestors, e.g.
'list("TrainCategory"="Express Passenger", "PowerType"="DMU")', would return the "DMU" data group underneath "Express Passenger". See the "Finding and Formatting" vignette for graphical examples.

`variableNames` A character vector specifying the name/names of the variables to find. This is useful generally only in pivot tables with irregular layouts, since in regular pivot tables every cell is related to every variable.

`variableValues` A list specifying the variable names and values to find, e.g. 'variableValues=list("PowerType"=c("DMU", "HST"))'.
Specify "**" as the variable value to match totals for the specified variable.
Specify "!*" as the variable value to match non-totals for the specified variable.
NB: The totals/non-totals criteria above won't work when visual totals are used.

`totals` A word that specifies how totals are matched (overrides the finer settings above) - must be one of "include" (default), "exclude" or "only".

`calculationNames` A character vector specifying the name/names of the calculations to find.

`atLevels` An integer vector constraining the levels in the hierarchy to search.

`minChildCount` Match only data groups with this minimum number of children.

`maxChildCount` Match only data groups with this maximum number of children.

`emptyGroups` A word that specifies how empty groups are matched - must be one of "include", "exclude" (default) or "only".

`outlineGroups` A word that specifies how outline cells are matched - must be one of "include", "exclude" (default) or "only".

`outlineLinkedGroupExists` 'TRUE' to match only groups where the related outline child group still exists. 'FALSE' to match only groups where the related outline child group no longer exists.

`includeDescendantGroups` Default 'FALSE'. Specify true to also return all descendants of data groups that match the specified criteria.

`includeCurrentGroup` Default 'TRUE'. Specify 'FALSE' to prevent the current group being included in the returned results.

Returns: A list of data groups matching the specified criteria.

Method `setStyling()`**:** An internal method used to set style declarations on the data group. Using 'pt$setStyling(cells=x)' is preferred for users.

Usage:

```
PivotDataGroup$setStyling(styleDeclarations = NULL)
```

Arguments:

`styleDeclarations` A list containing CSS style declarations.

Returns: No return value.

Method `clearSortGroups()`**:** An internal method that clears state data used during sorting operations.

Usage:

```
PivotDataGroup$clearSortGroups()
```

Returns: No return value.

Method `addSortGroupBefore()`**:** An internal method used during sorting operations.

Usage:

```
PivotDataGroup$addSortGroupBefore(grp)
```

Arguments:

`grp` The group to insert as part of sorting.

Returns: No return value.

Method `addSortGroupAfter()`**:** An internal method used during sorting operations.

Usage:

```
PivotDataGroup$addSortGroupAfter(grp)
```

Arguments:

`grp` The group to insert as part of sorting.

Returns: No return value.

Method `asList()`**:** Return the contents of this object as a list for debugging.

Usage:

```
PivotDataGroup$asList()
```

Returns: A list of various object properties.

Method `asJSON()`**:** Return the contents of this object as JSON for debugging.

Usage:

```
PivotDataGroup$asJSON()
```

Returns: A JSON representation of various object properties.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PivotDataGroup$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# This class should only be created by the pivot table.
# It is not intended to be created outside of the pivot table.
```

`PivotFilter` *R6 class that defines a filter condition.*

Description

The 'PivotFilter' class represents a single filter condition.

Format

`R6Class` object.

Details

The filter condition represented by a 'PivotFilter' instance relates to one data frame variable/column and is of the form [ColumnName] IN c(Value1, Value2, Value3, ...). Often in a pivot table, each filter specifies only one data value, as typically each distinct data value exists in a separate row or column. The 'PivotFilter' class contains methods to perform set based operations on filter values when combining filters.

Active bindings

`variableName` The name of the column in the data frame that this filter applies to.

`safeVariableName` The name of the column in the data frame that this filter applies to, surrounded by back-ticks if the name is not legal.

`type` Either "ALL", "VALUES" or "NONE". "VALUES" is the most common type and means the data is filtered to a subset of values. "ALL" means there is no filtering, i.e. all values match. "NONE" means there can be no matching values/data.

`values` The subset of values that this filter matches.

Methods

Public methods:

- PivotFilter$new()
- PivotFilter$intersect()
- PivotFilter$union()
- PivotFilter$replace()
- PivotFilter$getCopy()
- PivotFilter$asList()
- PivotFilter$asJSON()
- PivotFilter$asString()
- PivotFilter$clone()

Method `new()`**:** Create a new 'PivotFilter' object.

Usage:

```
PivotFilter$new(parentPivot, variableName = NULL, type = "ALL", values = NULL)
```

Arguments:

`parentPivot` The pivot table that this 'PivotFilter' instance belongs to.

`variableName` The name of the column in the data frame that this filter applies to.

`type` Must be either "ALL", "VALUES" or "NONE". "VALUES" is the most common type and means the data is filtered to a subset of values. "ALL" means there is no filtering, i.e. all values match. "NONE" means there can be no matching values/data.

`values` A single data value or a vector of multiple data values that this filter will match on.

Returns: A new 'PivotFilter' object.

Method `intersect()`**:** Updates this filter by intersecting the values in this filter with the values from another 'PivotFilter' object.

Usage:

```
PivotFilter$intersect(filter)
```

Arguments:

`filter` A 'PivotFilter' object.

Returns: No return value.

Method `union()`**:** Updates this filter by unioning the values in this filter with the values from another 'PivotFilter' object.

Usage:

```
PivotFilter$union(filter)
```

Arguments:

`filter` A 'PivotFilter' object.

Returns: No return value.

Method `replace()`**:** Updates this filter by replacing the values in this filter with the values from another 'PivotFilter' object.

Usage:

```
PivotFilter$replace(filter)
```

Arguments:

`filter` A 'PivotFilter' object.

Returns: No return value.

Method `getCopy()`**:** Create a copy of this 'PivotFilter' object.

Usage:

```
PivotFilter$getCopy()
```

Returns: A copy of this 'PivotFilter' object.

Method `asList()`**:** Return the contents of this object as a list for debugging.

Usage:

```
PivotFilter$asList()
```

Returns: A list of various object properties.

Method `asJSON()`**:** Return the contents of this object as JSON for debugging.

Usage:

```
PivotFilter$asJSON()
```

Returns: A JSON representation of various object properties.

Method `asString()`**:** Return a representation of this object as a character value.

Usage:

```
PivotFilter$asString(includeVariableName = TRUE, seperator = " ")
```

Arguments:

`includeVariableName` 'TRUE' (default) to include the variable name in the string.

`seperator` A character value used when concatenating multiple filter values.

Returns: A character summary of various object properties.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PivotFilter$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
pt <- PivotTable$new()
# ...
PivotFilter$new(pt, variableName="Country", values="England")
```

PivotFilterOverrides *R6 class that defines a set of filter overrides.*

Description

The 'PivotFilterOverrides' class contains multiple PivotFilter objects that can be used later to override a set of filters, e.g. in a pivot table calculation.

Format

R6Class object.

Details

Each cell in a pivot table has context (i.e. filters) coming from the row and column groups that are applicable to the cell. The 'PivotFilterOverrides' class contains several different ways of changing this filter criteria as part of a calculation. In most use cases, only one of the available approaches will be used.

Active bindings

`removeAllFilters` TRUE to remove all existing filters before applying any other and/replace/or filters.

`keepOnlyFiltersFor` Specify the names of existing variables to retain the filters for. All other filters will be removed.

`removeFiltersFor` Specify the names of variables to remove filters for.

`overrideFunction` A custom R function to amend the filters in each cell.

`countAnd` The number of 'PivotFilters' that will be combined with other pivot filters by intersecting their lists of allowed values.

`countReplace` The number of 'PivotFilters' that will be combined with other pivot filters by entirely replacing existing PivotFilter objects.

`countOr` The number of 'PivotFilters' that will be combined with other pivot filters by unioning their lists of allowed values.

`countTotal` The total number of 'PivotFilters' that will be combined with other pivot filters.

`andFilters` The 'PivotFilters' that will be combined with other pivot filters by intersecting their lists of allowed values.

`replaceFilters` The 'PivotFilters' that will be combined with other pivot filters by entirely replacing existing PivotFilter objects.

`orFilters` The 'PivotFilters' that will be combined with other pivot filters by unioning their lists of allowed values.

`allFilters` The complete set of 'PivotFilters' that will be combined with other pivot filters.

Methods

Public methods:

- PivotFilterOverrides$new()
- PivotFilterOverrides$add()
- PivotFilterOverrides$apply()
- PivotFilterOverrides$asList()
- PivotFilterOverrides$asJSON()
- PivotFilterOverrides$asString()
- PivotFilterOverrides$clone()

Method `new()`**:** Create a new 'PivotFilterOverrides' object.

Usage:

```
PivotFilterOverrides$new(
  parentPivot = NULL,
  removeAllFilters = FALSE,
  keepOnlyFiltersFor = NULL,
  removeFiltersFor = NULL,
  overrideFunction = NULL,
  filter = NULL,
  variableName = NULL,
  type = "ALL",
  values = NULL,
  action = "replace"
)
```

Arguments:

`parentPivot` The pivot table that this 'PivotFilterOverrides' instance belongs to.

`removeAllFilters` Specifies whether to clear all existing filters, before applying the filter overrides. Default value 'FALSE'

`keepOnlyFiltersFor` A character vector specifying the variable names to retain the filter criteria for. Filter criteria for all other variables will be cleared.

`removeFiltersFor` A character vector specifying the variable names for which the filter criteria will be cleared. Filter criteria for all other variables will be retained.

`overrideFunction` A custom R function which will be invoked for each cell to modify the filters before the calculation is carried out.

`filter` A PivotFilter object containing filter criteria which will be combined with the current set of filters using the specified combine method.

`variableName` The variable name for a new filter to apply to. Specified in conjunction with the 'type' and 'values' parameters.

`type` The type of a new filter to apply, must be either "ALL", "VALUES" or "NONE".

`values` A single data value or a vector of multiple data values that a new filter will match on.

`action` Specifies how the new filter defined in 'filter' (or 'variableName', 'type' and 'values') should be combined with the existing filter criteria for the cell. Must be one of "intersect", "replace" or "union".

Returns: A new 'PivotFilterOverrides' object.

Method `add()`**:** Add additional filter criteria into this 'PivotFilterOverrides' object. Either 'filter' is specified, or 'variableName', 'type' and 'values' are specified.

Usage:

```
PivotFilterOverrides$add(
  filter = NULL,
  variableName = NULL,
  type = "ALL",
  values = NULL,
  action = "replace"
)
```

Arguments:

`filter` A 'PivotFilter' to take criteria from.

`variableName` The variable name the additional criteria applies to.

`type` The type of the additional filter criteria, must be either "ALL", "VALUES" or "NONE".

`values` A single data value or a vector of multiple data values that compromise the additional filter criteria.

`action` Specifies how the additional filter should be combined with the existing filter criteria for the cell. Must be one of "intersect", "replace" or "union".

Returns: No return value.

Method `apply()`**:** Apply the filter overrides to an existing 'PivotFilters' object.

Usage:

```
PivotFilterOverrides$apply(filters = NULL, cell = NULL)
```

Arguments:

`filters` A 'PivotFilters' object to apply the filter overrides to.

`cell` A 'PivotCell' object representing the cell that the 'filters' relate to.

Returns: No return value.

Method `asList()`**:** Return the contents of this object as a list for debugging.

Usage:

```
PivotFilterOverrides$asList()
```

Returns: A list of various object properties.

Method `asJSON()`**:** Return the contents of this object as JSON for debugging.

Usage:

```
PivotFilterOverrides$asJSON()
```

Returns: A JSON representation of various object properties.

Method `asString()`**:** Return a representation of this object as a character value.

Usage:

```
PivotFilterOverrides$asString(includeVariableName = TRUE, seperator = ", ")
```

Arguments:

`includeVariableName` 'TRUE' (default) to include the variable name in the string.

`seperator` A character value used when concatenating multiple filter overrides.

Returns: A character summary of various object properties.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PivotFilterOverrides$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
pt <- PivotTable$new()
# ...
# PivotFilterOverrides constructor allows a filter to be defined
# e.g. to enable %of row or column type calculations
filterOverrides <- PivotFilterOverrides$new(pt, keepOnlyFiltersFor="Volume")
# Alternatively/in addition, create a new filter
filter <- PivotFilter$new(pt, variableName="Country", values="England")
# Add the filter to the set of overrides
filterOverrides$add(filter=filter, action="replace")
```

`PivotFilters` *R6 class that defines a set of filter conditions.*

Description

The 'PivotFilters' class allows multiple filter conditions relating to different data frame columns to be combined, i.e. a 'PivotFilters' object typically contains multiple `PivotFilter` objects.

Format

`R6Class` object.

Details

As well as acting as a container for multiple filter conditions, the 'PivotFilters' class also contains logic for combining filter. The 'action' parameter in many of the methods controls how two filters are combined.
Most common cases:
(1) When working out the rowColFilters for each pivot table cell, the filters from the row and column leaf groups are combined using 'action="intersect"'.
(2) When combining the rowColFilters with calculation filters the action could be any of (in order of most typical) "intersect", "replace" or "union".
"intersect" would apply additional restrictions, e.g. see the example in the Calculations vignette that has a measure for weekend trains only.
"replace" would apply when doing things like percentage of row total calculations - again, see example in the calculations vignette

"union" is probably much less likely (hard to envisage many situations when that would be needed). (3) In custom calculation functions, the action could be any of "intersect", "replace" or "union". NOTE: 'pivottabler' does not allow complex conditions to be built up, such as ((A=X) or (B=Y)) and (C=2) since there is complex precedence involved and conditions like this are not typical of pivot tables. If they were really needed, a workaround would be to use a custom calculation function and include this logic in that function.
See Appendix 2 vignette for many more complex calculation details.

Active bindings

`count` The number of 'PivotFilter' objects in this 'PivotFilters' object.

`filters` A list of 'PivotFilter' objects in this 'PivotFilters' object.

`isALL` If TRUE, this 'PivotFilters' object matches all data.

`isNONE` If TRUE, this 'PivotFilters' object matches no data.

`filteredVariables` The names of the variables that are filtered by this 'PivotFilters' object.

`filteredValues` A list of the criteria values for each of the variables filtered by this 'PivotFilters' object, where the list element names are the variable names.

Methods

Public methods:

- `PivotFilters$new()`
- `PivotFilters$clearFilters()`
- `PivotFilters$keepOnlyFiltersFor()`
- `PivotFilters$removeFiltersFor()`
- `PivotFilters$getFilter()`
- `PivotFilters$isFilterMatch()`
- `PivotFilters$setFilters()`
- `PivotFilters$setFilter()`
- `PivotFilters$setFilterValues()`
- `PivotFilters$addFilter()`
- `PivotFilters$getFilteredDataFrame()`
- `PivotFilters$getCopy()`
- `PivotFilters$asList()`
- `PivotFilters$asJSON()`
- `PivotFilters$asString()`
- `PivotFilters$clone()`

Method `new()`**:** Create a new 'PivotFilters' object, optionally adding a filter.

Usage:

```
PivotFilters$new(
  parentPivot = NULL,
  variableName = NULL,
  type = "ALL",
  values = NULL
)
```

Arguments:

`parentPivot` The pivot table that this 'PivotFilters' instance belongs to.

`variableName` The name of the column in the data frame that this filter applies to. Specify 'NULL' to skip adding a filter.

`type` Must be either "ALL", "VALUES" or "NONE". "VALUES" is the most common type and means the data is filtered to a subset of values. "ALL" means there is no filtering, i.e. all values match. "NONE" means there can be no matching values/data.

`values` A single data value or a vector of multiple data values that the filter will match on.

Returns: A new 'PivotFilters' object.

Method `clearFilters()`**:** Remove all filters from this 'PivotFilters' object.

Usage:

```
PivotFilters$clearFilters()
```

Returns: No return value.

Method `keepOnlyFiltersFor()`**:** Remove the filters for all variables except those specified.

Usage:

```
PivotFilters$keepOnlyFiltersFor(variableNames = NULL)
```

Arguments:

`variableNames` A character vector specifying the variable names to retain the filter criteria for. Filter criteria for all other variables will be cleared.

Returns: No return value.

Method `removeFiltersFor()`**:** Remove the filters for the specified variables.

Usage:

```
PivotFilters$removeFiltersFor(variableNames = NULL)
```

Arguments:

`variableNames` A character vector specifying the variable names for which the filter criteria will be cleared. Filter criteria for all other variables will be retained.

Returns: No return value.

Method `getFilter()`**:** Find a filter with the specified variable name.

Usage:

```
PivotFilters$getFilter(variableName = NULL)
```

Arguments:

`variableName` The variable name to find a filter for.

Returns: A 'PivotFilter' object that filters on the specified variable.

Method `isFilterMatch()`**:** Tests whether this 'PivotFilters' object matches specified criteria.

Usage:

```
PivotFilters$isFilterMatch(
  matchMode = "simple",
  variableNames = NULL,
  variableValues = NULL
)
```

Arguments:

`matchMode` Either "simple" (default) or "combinations".
"simple" is used when matching only one variable-value, multiple variable-value combinations are effectively logical "OR", i.e. any one single 'PivotFilter' match means the 'PivotFilters' object is a match.
"combinations" is used when matching for combinations of variable values, multiple variable-value combinations are effectively logical "AND", i.e. there must be a matching 'PivotFilter' for every variable name / variable values criteria specified.
See the "Finding and Formatting" vignette for graphical examples.

`variableNames` The variable name(s) to find a filter for. This can be a vector containing more than one variable name.

`variableValues` A list specifying the variable names and values to find, e.g. 'variableValues=list("PowerType"=c("DMU", "HST"))'.

Returns: 'TRUE' if this filters object matches the specified criteria, 'FALSE' otherwise.

Method `setFilters()`**:** Update the value of this 'PivotFilters' object with the filters from the specified 'PivotFilters' object, either intersecting, replacing or unioning the filter criteria.

Usage:

```
PivotFilters$setFilters(filters = NULL, action = "replace")
```

Arguments:

`filters` A 'PivotFilters' object.

`action` Specifies how the criteria defined in 'filters' should be combined with the existing filter criteria. Must be one of "intersect", "replace" or "union".

Returns: No return value.

Method `setFilter()`**:** Update the value of this 'PivotFilters' object with the filters from the specified 'PivotFilter' object, either intersecting, replacing or unioning the filter criteria.

Usage:

```
PivotFilters$setFilter(filter = NULL, action = "replace")
```

Arguments:

`filter` A 'PivotFilter' object.

`action` Specifies how the criteria defined in 'filter' should be combined with the existing filter criteria. Must be one of "intersect", "replace" or "union".

Returns: No return value.

Method `setFilterValues()`**:** Update the value of this 'PivotFilters' object with additional filter criteria, either intersecting, replacing or unioning the filter criteria.

Usage:

```
PivotFilters$setFilterValues(
  variableName = NULL,
  type = "ALL",
  values = NULL,
  action = "replace"
)
```

Arguments:

`variableName` The name of the column in the data frame that this criteria applies to.

`type` Must be either "ALL", "VALUES" or "NONE".

`values` A single data value or a vector of multiple data values that comprise the additional filter criteria.

`action` Specifies how the criteria defined in 'filter' should be combined with the existing filter criteria. Must be one of "intersect", "replace" or "union".

Returns: No return value.

Method `addFilter()`**:** Add a new 'PivotFilter' object to the filter list in this 'PivotFilters' object.

Usage:

```
PivotFilters$addFilter(filter = NULL)
```

Arguments:

`filter` A 'PivotFilter' object.

Returns: No return value.

Method `getFilteredDataFrame()`**:** Filters the specified data frame using the filters defined in this 'PivotFilters' object and returns the results as another data frame.

Usage:

```
PivotFilters$getFilteredDataFrame(dataFrame = NULL)
```

Arguments:

`dataFrame` A data frame to filter.

Returns: A data frame filtered according to the criteria in this 'PivotFilters' object.

Method `getCopy()`**:** Create a copy of this 'PivotFilters' object.

Usage:

```
PivotFilters$getCopy()
```

Returns: A copy of this 'PivotFilters' object.

Method `asList()`**:** Return the contents of this object as a list for debugging.

Usage:

```
PivotFilters$asList()
```

Returns: A list of various object properties.

Method `asJSON()`**:** Return the contents of this object as JSON for debugging.

Usage:

```
PivotFilters$asJSON()
```

Returns: A JSON representation of various object properties.

Method `asString()`**:** Return a representation of this object as a character value.

Usage:

```
PivotFilters$asString(includeVariableName = TRUE, seperator = ", ")
```

Arguments:

`includeVariableName` 'TRUE' (default) to include the variable name in the string.

`seperator` A character value used when concatenating multiple filters.

Returns: A character summary of various object properties.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PivotFilters$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
pt <- PivotTable$new()
# ...
# PivotFilters constructor allows a filter to be defined
filters <- PivotFilters$new(pt, variableName="Year", values=2017)
# Create a new filter
filter <- PivotFilter$new(pt, variableName="Country", values="England")
# Combine the filters
filters$setFilter(filter)
# filters now contains criteria for both Year and Country
# Now add another filter, this time via an alternative method
filters$setFilterValues(variableName="Product", values="Cadbury Dairy Milk
Chocolate 100g")
# filters now contains criteria for Year, Country and Product
```

`PivotHtmlRenderer` *R6 class that renders a pivot table in HTML.*

Description

The 'PivotHtmlRenderer' class creates a HTML representation of a pivot table.

Format

`R6Class` object.

Methods

Public methods:

- PivotHtmlRenderer$new()
- PivotHtmlRenderer$clearIsRenderedFlags()
- PivotHtmlRenderer$getTableHtml()
- PivotHtmlRenderer$clone()

Method `new()`**:** Create a new 'PivotHtmlRenderer' object.

Usage:

```
PivotHtmlRenderer$new(parentPivot)
```

Arguments:

`parentPivot` The pivot table that this 'PivotHtmlRenderer' instance belongs to.

Returns: A new 'PivotHtmlRenderer' object.

Method `clearIsRenderedFlags()`**:** An internal method used when rendering a pivot table to HTML. Clear the IsRendered flags that exist on the 'PivotDataGroup' class.

Usage:

```
PivotHtmlRenderer$clearIsRenderedFlags()
```

Returns: No return value.

Method `getTableHtml()`**:** Generate a HTML representation of the pivot table, optionally including additional detail for debugging purposes.

Usage:

```
PivotHtmlRenderer$getTableHtml(
  styleNamePrefix = NULL,
  includeHeaderValues = FALSE,
  includeRCFilters = FALSE,
  includeCalculationFilters = FALSE,
  includeWorkingData = FALSE,
  includeEvaluationFilters = FALSE,
  includeCalculationNames = FALSE,
  includeRawValue = FALSE,
  includeTotalInfo = FALSE,
  exportOptions = NULL,
  showRowGroupHeaders = FALSE
)
```

Arguments:

`styleNamePrefix` A character variable specifying a prefix for all named CSS styles, to avoid style name collisions where multiple pivot tables exist.

`includeHeaderValues` Default 'FALSE', specify 'TRUE' to render this debug information.

`includeRCFilters` Default 'FALSE', specify 'TRUE' to render this debug information.

`includeCalculationFilters` Default 'FALSE', specify 'TRUE' to render this debug information.

`includeWorkingData` Default 'FALSE', specify 'TRUE' to render this debug information.

`includeEvaluationFilters` Default 'FALSE', specify 'TRUE' to render this debug information.

`includeCalculationNames` Default 'FALSE', specify 'TRUE' to render this debug information.

`includeRawValue` Default 'FALSE', specify 'TRUE' to render this debug information.

`includeTotalInfo` Default 'FALSE', specify 'TRUE' to render this debug information.

`exportOptions` A list of additional export options - see the "A1. Appendix" for details.

`showRowGroupHeaders` Default 'FALSE', specify 'TRUE' to render the row group headings. See the "Data Groups" vignette for details.

Returns: A list containing HTML tags from the 'htmltools' package. Convert this to a character variable using 'as.character()'.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PivotHtmlRenderer$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# This class should only be created by the pivot table.
# It is not intended to be created outside of the pivot table.
```

`PivotLatexRenderer` *R6 class that renders a pivot table in Latex.*

Description

The 'PivotLatexRenderer' class creates a Latex representation of a pivot table.

Format

`R6Class` object.

Methods

Public methods:

- `PivotLatexRenderer$new()`
- `PivotLatexRenderer$clearIsRenderedFlags()`
- `PivotLatexRenderer$resetVisibleRange()`
- `PivotLatexRenderer$setVisibleRange()`
- `PivotLatexRenderer$getTableLatex()`
- `PivotLatexRenderer$clone()`

Method `new()`**:** Create a new 'PivotLatexRenderer' object.

Usage:

```
PivotLatexRenderer$new(parentPivot = NULL)
```

Arguments:

`parentPivot` The pivot table that this 'PivotLatexRenderer' instance belongs to.

Returns: A new 'PivotLatexRenderer' object.

Method `clearIsRenderedFlags()`**:** An internal method used when rendering a pivot table to Latex Clear the IsRendered flags that exist on the 'PivotDataGroup' class.

Usage:

```
PivotLatexRenderer$clearIsRenderedFlags()
```

Returns: No return value.

Method `resetVisibleRange()`**:** Clears the visible range that has been set, so the next call to 'getTableLatex()' will render the whole table.

Usage:

```
PivotLatexRenderer$resetVisibleRange()
```

Returns: No return value.

Method `setVisibleRange()`**:** Specifies a subset of the pivot table to be rendered, e.g. for use when a pivot table will not fit into a single A4 page.

Usage:

```
PivotLatexRenderer$setVisibleRange(
  fromRow = NULL,
  toRow = NULL,
  fromColumn = NULL,
  toColumn = NULL
)
```

Arguments:

`fromRow` The row number to render from.

`toRow` The row number to render to.

`fromColumn` The column number to render from.

`toColumn` The column number to render to.

Returns: No return value.

Method `getTableLatex()`**:** Generate a Latex representation of the pivot table.

Usage:

```
PivotLatexRenderer$getTableLatex(
  caption = NULL,
  label = NULL,
  boldHeadings = FALSE,
  italicHeadings = FALSE,
  exportOptions = NULL
)
```

Arguments:

`caption` The caption to appear above the table.

`label` The label to use when referring to the table elsewhere in the document

`boldHeadings` Default 'FALSE', specify 'TRUE' to render headings in bold.

`italicHeadings` Default 'FALSE', specify 'TRUE' to render headings in italic.

`exportOptions` A list of additional export options - see the "A1. Appendix" for details.

Returns: A character variable containing the Latex representation of the pivot table.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PivotLatexRenderer$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# This class should only be created by the pivot table.
# It is not intended to be created outside of the pivot table.
```

`PivotOpenXlsxRenderer` *R6 class that renders a pivot table into an Excel worksheet.*

Description

The 'PivotOpenXlsxRenderer' class creates a representation of a pivot table in an Excel file using the 'openxlsx' package. See the "Excel Export" vignette for details and examples.

Format

`R6Class` object.

Methods

Public methods:

- `PivotOpenXlsxRenderer$new()`
- `PivotOpenXlsxRenderer$clearIsRenderedFlags()`
- `PivotOpenXlsxRenderer$writeToCell()`
- `PivotOpenXlsxRenderer$writeToWorksheet()`
- `PivotOpenXlsxRenderer$clone()`

Method `new()`**:** Create a new 'PivotOpenXlsxRenderer' object.

Usage:

```
PivotOpenXlsxRenderer$new(parentPivot)
```

Arguments:

`parentPivot` The pivot table that this 'PivotOpenXlsxRenderer' instance belongs to.

Returns: A new 'PivotOpenXlsxRenderer' object. An internal method used when rendering a pivot table to HTML. Clear the IsRendered flags that exist on the 'PivotDataGroup' class.

Method `clearIsRenderedFlags()`**:**

Usage:

```
PivotOpenXlsxRenderer$clearIsRenderedFlags()
```

Returns: No return value.

Method `writeToCell()`**:** Writes a value to a cell and applies styling as needed.

Usage:

```
PivotOpenXlsxRenderer$writeToCell(
  wb = NULL,
  wsName = NULL,
  rowNumber = NULL,
  columnNumber = NULL,
  value = NULL,
  applyStyles = TRUE,
  baseStyleName = NULL,
  style = NULL,
  mapFromCss = TRUE,
  mergeRows = NULL,
  mergeColumns = NULL
)
```

Arguments:

`wb` A 'Workbook' object representing the Excel file being written to.

`wsName` A character value specifying the name of the worksheet to write to.

`rowNumber` An integer value specifying the row number of the cell to write to.

`columnNumber` An integer value specifying the column number of the cell to write to.

`value` The value to write into the cell.

`applyStyles` Default 'TRUE' to write styling information to the cell.

`baseStyleName` A character value specifying a named style defined in the pivot table.

`style` A 'PivotStyle' object containing CSS style declarations to override the base style.

`mapFromCss` Default 'TRUE' to automatically convert CSS style declarations to their Excel equivalents.

`mergeRows` An integer vector specifying the row extent of a merged cell.

`mergeColumns` An integer vector specifying the column extent of a merged cell.

Returns: No return value.

Method `writeToWorksheet()`**:** Write the pivot table into the specified workbook and worksheet at the specified row-column location.

Usage:

```
PivotOpenXlsxRenderer$writeToWorksheet(
  wb = NULL,
  wsName = NULL,
  topRowNumber = NULL,
```

```
  leftMostColumnNumber = NULL,
  outputHeadingsAs = "formattedValueAsText",
  outputValuesAs = "rawValue",
  applyStyles = TRUE,
  mapStylesFromCSS = TRUE,
  exportOptions = NULL,
  showRowGroupHeaders = FALSE
)
```

Arguments:

`wb` A 'Workbook' object representing the Excel file being written to.

`wsName` A character value specifying the name of the worksheet to write to.

`topRowNumber` An integer value specifying the row number in the Excel worksheet to write the pivot table.

`leftMostColumnNumber` An integer value specifying the column number in the Excel worksheet to write the pivot table.

`outputHeadingsAs` Must be one of "rawValue", "formattedValueAsText" (default) or "formattedValueAsNumber" to specify how data groups are written into the Excel sheet.

`outputValuesAs` Must be one of "rawValue" (default), "formattedValueAsText" or "formattedValueAsNumber" to specify how cell values are written into the Excel sheet.

`applyStyles` Default 'TRUE' to write styling information to the cell.

`mapStylesFromCSS` Default 'TRUE' to automatically convert CSS style declarations to their Excel equivalents.

`exportOptions` A list of additional export options - see the "A1. Appendix" for details.

`showRowGroupHeaders` Default 'FALSE', specify 'TRUE' to write row group headers.

Returns: No return value.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PivotOpenXlsxRenderer$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# This class should only be created by the pivot table.
# It is not intended to be created outside of the pivot table.
```

`PivotOpenXlsxStyle` *R6 class that specifies Excel styling as used by the openxlsx package.*

Description

The 'PivotOpenXlsxStyle' class specifies the styling for cells in an Excel worksheet.

Format

R6Class object.

Active bindings

`baseStyleName` The name of the base style in the pivot table.

`isBaseStyle` 'TRUE' when this style is the equivalent of a named style in the pivot table, 'FALSE' if this style has additional settings over and above the base style of the same name.

`fontName` The name of the font (single font name, not a CSS style list).

`fontSize` The size of the font (units: point, 4-72).

`bold` 'TRUE' if text is bold.

`italic` 'TRUE' if text is italic.

`underline` 'TRUE' if text is underlined.

`strikethrough` 'TRUE' if text has a line through it.

`superscript` 'TRUE' if text is small and raised.

`subscript` 'TRUE' if text is small and lowered.

`fillColor` The background colour for the cell (as a hex value, e.g. #00FF00).

`textColor` The color of the text (as a hex value).

`hAlign` The horizontal alignment of the text: left, center or right.

`vAlign` The vertical alignment of the text: top, middle or bottom.

`wrapText` 'TRUE' if the text is allowed to wrap onto multiple lines.

`textRotation` The rotation angle of the text (0 to 359) or 255 for vertical.

`indent` Horizontal indentation of cell contents (0 to 250.).

`borderAll` A list (with elements style and color) specifying the border settings for all four sides of each cell at once.

`borderLeft` A list (with elements style and color) specifying the border settings for the left border of each cell.

`borderRight` A list (with elements style and color) specifying the border settings for the right border of each cell.

`borderTop` A list (with elements style and color) specifying the border settings for the top border of each cell.

`borderBottom` A list (with elements style and color) specifying the border settings for the bottom border of each cell.

`valueFormat` The Excel formatting applied to the field value. One of the following values: GENERAL, NUMBER, CURRENCY, ACCOUNTING, DATE, LONGDATE, TIME, PERCENTAGE, FRACTION, SCIENTIFIC, TEXT, COMMA. Or for dates/datetimes, a combination of d, m, y. Or for numeric values, use a numeric format code such as 0.00, #,###.00, etc

`minColumnWidth` The minimum width of this column (0 to 255).

`minRowHeight` The minimum height of this row (0 to 400).

`openxlsxStyle` The style object returned from 'openxlsx::createStyle()'.

Methods

Public methods:

- PivotOpenXlsxStyle$new()
- PivotOpenXlsxStyle$isBasicStyleNameMatch()
- PivotOpenXlsxStyle$isFullStyleDetailMatch()
- PivotOpenXlsxStyle$createOpenXslxStyle()
- PivotOpenXlsxStyle$asList()
- PivotOpenXlsxStyle$asJSON()
- PivotOpenXlsxStyle$asString()
- PivotOpenXlsxStyle$clone()

Method `new()`**:** Create a new 'PivotOpenXlsxStyle' object.

Usage:

```
PivotOpenXlsxStyle$new(
  parentPivot,
  baseStyleName = NULL,
  isBaseStyle = NULL,
  fontName = NULL,
  fontSize = NULL,
  bold = NULL,
  italic = NULL,
  underline = NULL,
  strikethrough = NULL,
  superscript = NULL,
  subscript = NULL,
  fillColor = NULL,
  textColor = NULL,
  hAlign = NULL,
  vAlign = NULL,
  wrapText = NULL,
  textRotation = NULL,
  indent = NULL,
  borderAll = NULL,
  borderLeft = NULL,
  borderRight = NULL,
  borderTop = NULL,
  borderBottom = NULL,
  valueFormat = NULL,
  minColumnWidth = NULL,
  minRowHeight = NULL
)
```

Arguments:

`parentPivot` The pivot table that this 'PivotOpenXlsxStyle' instance belongs to.

`baseStyleName` The name of the base style in the pivot table.

`isBaseStyle` 'TRUE' when this style is the equivalent of a named style in the pivot table, 'FALSE' if this style has additional settings over and above the base style of the same name.

`fontName` The name of the font (single font name, not a CSS style list).

`fontSize` The size of the font (units: point, 4-72).

`bold` 'TRUE' if text is bold.

`italic` 'TRUE' if text is italic.

`underline` 'TRUE' if text is underlined.

`strikethrough` 'TRUE' if text has a line through it.

`superscript` 'TRUE' if text is small and raised.

`subscript` 'TRUE' if text is small and lowered.

`fillColor` The background colour for the cell (as a hex value, e.g. #00FF00).

`textColor` The color of the text (as a hex value).

`hAlign` The horizontal alignment of the text: left, center or right.

`vAlign` The vertical alignment of the text: top, middle or bottom.

`wrapText` 'TRUE' if the text is allowed to wrap onto multiple lines.

`textRotation` The rotation angle of the text (0 to 359) or 255 for vertical.

`indent` Horizontal indentation of cell contents (0 to 250.).

`borderAll` A list (with elements style and color) specifying the border settings for all four sides of each cell at once.

`borderLeft` A list (with elements style and color) specifying the border settings for the left border of each cell.

`borderRight` A list (with elements style and color) specifying the border settings for the right border of each cell.

`borderTop` A list (with elements style and color) specifying the border settings for the top border of each cell.

`borderBottom` A list (with elements style and color) specifying the border settings for the bottom border of each cell.

`valueFormat` The Excel formatting applied to the field value. One of the following values: GENERAL, NUMBER, CURRENCY, ACCOUNTING, DATE, LONGDATE, TIME, PERCENTAGE, FRACTION, SCIENTIFIC, TEXT, COMMA. Or for dates/datetimes, a combination of d, m, y. Or for numeric values, use a numeric format code such as 0.00, #,###.00, etc

`minColumnWidth` The minimum width of this column (0 to 255).

`minRowHeight` The minimum height of this row (0 to 400).

Returns: A new 'PivotOpenXlsxStyle' object.

Method `isBasicStyleNameMatch()`**:** Test whether a style is a base style with the specified name.

Usage:

```
PivotOpenXlsxStyle$isBasicStyleNameMatch(baseStyleName = NULL)
```

Arguments:

`baseStyleName` The name of the style to match.

Returns: No return value.

Method `isFullStyleDetailMatch()`**:** Test whether a style has matching style attributes.

Usage:

```
PivotOpenXlsxStyle$isFullStyleDetailMatch(
  baseStyleName = NULL,
  isBaseStyle = NULL,
  fontName = NULL,
  fontSize = NULL,
  bold = NULL,
  italic = NULL,
  underline = NULL,
  strikethrough = NULL,
  superscript = NULL,
  subscript = NULL,
  fillColor = NULL,
  textColor = NULL,
  hAlign = NULL,
  vAlign = NULL,
  wrapText = NULL,
  textRotation = NULL,
  indent = NULL,
  borderAll = NULL,
  borderLeft = NULL,
  borderRight = NULL,
  borderTop = NULL,
  borderBottom = NULL,
  valueFormat = NULL,
  minColumnWidth = NULL,
  minRowHeight = NULL
)
```

Arguments:

`baseStyleName` The name of the base style in the pivot table.

`isBaseStyle` 'TRUE' when this style is the equivalent of a named style in the pivot table, 'FALSE' if this style has additional settings over and above the base style of the same name.

`fontName` The name of the font (single font name, not a CSS style list).

`fontSize` The size of the font (units: point, 4-72).

`bold` 'TRUE' if text is bold.

`italic` 'TRUE' if text is italic.

`underline` 'TRUE' if text is underlined.

`strikethrough` 'TRUE' if text has a line through it.

`superscript` 'TRUE' if text is small and raised.

`subscript` 'TRUE' if text is small and lowered.

`fillColor` The background colour for the cell (as a hex value, e.g. #00FF00).

`textColor` The color of the text (as a hex value).

`hAlign` The horizontal alignment of the text: left, center or right.

`vAlign` The vertical alignment of the text: top, middle or bottom.

`wrapText` 'TRUE' if the text is allowed to wrap onto multiple lines.

`textRotation` The rotation angle of the text (0 to 359) or 255 for vertical.

`indent` Horizontal indentation of cell contents (0 to 250.).

`borderAll` A list (with elements style and color) specifying the border settings for all four sides of each cell at once.

`borderLeft` A list (with elements style and color) specifying the border settings for the left border of each cell.

`borderRight` A list (with elements style and color) specifying the border settings for the right border of each cell.

`borderTop` A list (with elements style and color) specifying the border settings for the top border of each cell.

`borderBottom` A list (with elements style and color) specifying the border settings for the bottom border of each cell.

`valueFormat` The Excel formatting applied to the field value. One of the following values: GENERAL, NUMBER, CURRENCY, ACCOUNTING, DATE, LONGDATE, TIME, PERCENTAGE, FRACTION, SCIENTIFIC, TEXT, COMMA. Or for dates/datetimes, a combination of d, m, y. Or for numeric values, use a numeric format code such as 0.00, #,###.00, etc

`minColumnWidth` The minimum width of this column (0 to 255).

`minRowHeight` The minimum height of this row (0 to 400).

Details: Base styles are compared on name only, otherwise the style attributes are compared.

Returns: No return value.

Method `createOpenXslxStyle()`**:** Create an 'openxlsx' style from this style definition.

Usage:

```
PivotOpenXlsxStyle$createOpenXslxStyle()
```

Returns: No return value. Retrieve the style using the 'openxlsxStyle' property.

Method `asList()`**:** Return the contents of this object as a list for debugging.

Usage:

```
PivotOpenXlsxStyle$asList()
```

Returns: A list of various object properties.

Method `asJSON()`**:** Return the contents of this object as JSON for debugging.

Usage:

```
PivotOpenXlsxStyle$asJSON()
```

Returns: A JSON representation of various object properties.

Method `asString()`**:** Return a representation of this object as a character value.

Usage:

```
PivotOpenXlsxStyle$asString()
```

Returns: A character summary of various object properties.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PivotOpenXlsxStyle$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# PivotOpenXlsxStyle objects are created by the PivotOpenXlsxRenderer class.
# See that class for details.
```

`PivotOpenXlsxStyles` *R6 class that defines a collection of Excel styles as used by the openxlsx package.*

Description

The 'PivotOpenXlsxStyles' class stores a collection of 'PivotOpenXlsxStyle' style objects.

Format

`R6Class` object.

Active bindings

`count` The number of 'PivotOpenXlsxStyle' objects in this 'PivotOpenXlsxStyles' collection.

`styles` A list containing the 'PivotOpenXlsxStyle' objects in this 'PivotOpenXlsxStyles' collection.

Methods

Public methods:

- `PivotOpenXlsxStyles$new()`
- `PivotOpenXlsxStyles$clearStyles()`
- `PivotOpenXlsxStyles$findNamedStyle()`
- `PivotOpenXlsxStyles$findOrAddStyle()`
- `PivotOpenXlsxStyles$addNamedStyles()`
- `PivotOpenXlsxStyles$asList()`
- `PivotOpenXlsxStyles$asJSON()`
- `PivotOpenXlsxStyles$asString()`
- `PivotOpenXlsxStyles$clone()`

Method `new()`**:** Create a new 'PivotOpenXlsxStyles' object.

Usage:

```
PivotOpenXlsxStyles$new(parentPivot)
```

Arguments:

`parentPivot` The pivot table that this 'PivotOpenXlsxStyles' instance belongs to.

Returns: A new 'PivotOpenXlsxStyles' object.

Method `clearStyles()`**:** Clear the internal list of styles.

Usage:

```
PivotOpenXlsxStyles$clearStyles()
```

Returns: No return value.

Method `findNamedStyle()`**:** Find an existing openxlsx style matching the name of a base style.

Usage:

```
PivotOpenXlsxStyles$findNamedStyle(baseStyleName)
```

Arguments:

`baseStyleName` The name of the base style to find.

Returns: A 'PivotOpenXlsxStyle' object with the specified name.

Method `findOrAddStyle()`**:** Find an existing openxlsx style, add a new openxlsx style matching a base style and/or existing 'PivotStyle' object.

Usage:

```
PivotOpenXlsxStyles$findOrAddStyle(
  action = "findOrAdd",
  baseStyleName = NULL,
  isBaseStyle = NULL,
  style = NULL,
  mapFromCss = TRUE
)
```

Arguments:

`action` Must be one of "find" (to search for an existing style), "add" (to add a new style) or "findOrAdd" (default, to first search for an existing style, and if no match is found then add a new style)

`baseStyleName` The name of the base style to find.

`isBaseStyle` 'TRUE' if the style being sought is a base style.

`style` An existing 'PivotStyle' object.

`mapFromCss` Default 'TRUE', to create a new 'PivotOpenXlsxStyle' by mapping from CSS style declarations.

Details: This function is used in two different ways: (1) When adding base styles (i.e. named styles in the pivot table) to this 'PivotOpenXlsxStyles' collection: In this case, 'baseStyleName' is the name of the style and 'isBaseStyle=TRUE' (so matching is by name only) and 'style' is the 'PivotStyle' object for the base style. (2) When finding styles that have been applied to individual cells using the 'PivotStyle' object that is attached to each cell: In this case, 'baseStyleName' may or may not be present, 'isBaseStyle=FALSE' and 'style' is the 'PivotStyle' object from the cell.

Returns: A 'PivotOpenXlsxStyle' object that has been found or added.

Method `addNamedStyles()`**:** Populate the OpenXlsx styles based on the styles defined in the pivot table.

Usage:

```
PivotOpenXlsxStyles$addNamedStyles(mapFromCss = TRUE)
```

Arguments:

`mapFromCss` Default 'TRUE', to create a new 'PivotOpenXlsxStyle' by mapping from CSS style declarations.

Returns: No return value.

Method `asList()`**:** Return the contents of this object as a list for debugging.

Usage:

```
PivotOpenXlsxStyles$asList()
```

Returns: A list of various object properties.

Method `asJSON()`**:** Return the contents of this object as JSON for debugging.

Usage:

```
PivotOpenXlsxStyles$asJSON()
```

Returns: A JSON representation of various object properties.

Method `asString()`**:** Return a representation of this object as a character value.

Usage:

```
PivotOpenXlsxStyles$asString(seperator = ", ")
```

Arguments:

`seperator` A character value used when concatenating multiple styles.

Returns: A character summary of various object properties.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PivotOpenXlsxStyles$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# This class should only be created by the pivot table.
# It is not intended to be created outside of the pivot table.
```

`PivotStyle` *R6 class that specifies styling.*

Description

The 'PivotStyle' class specifies the styling for headers and cells in a pivot table. Styles are specified in the form of Cascading Style Sheet (CSS) name-value pairs.

Format

`R6Class` object.

Active bindings

`name` The name of the style (for a named style).

`declarations` A list containing the style declarations.

Methods

Public methods:

- `PivotStyle$new()`
- `PivotStyle$setPropertyValue()`
- `PivotStyle$setPropertyValues()`
- `PivotStyle$getPropertyValue()`
- `PivotStyle$asCSSRule()`
- `PivotStyle$asNamedCSSStyle()`
- `PivotStyle$getCopy()`
- `PivotStyle$asList()`
- `PivotStyle$asJSON()`
- `PivotStyle$clone()`

Method `new()`**:** Create a new 'PivotStyle' object.

Usage:

```
PivotStyle$new(parentPivot, styleName = NULL, declarations = NULL)
```

Arguments:

`parentPivot` The pivot table that this 'PivotStyle' instance belongs to.

`styleName` The name of the style (for a named style).

`declarations` CSS style declarations in the form of a list, e.g. 'list("font-weight"="bold", "color"="#0000FF")'

Returns: A new 'PivotStyle' object.

Method `setPropertyValue()`**:** Set a single style property.

Usage:

```
PivotStyle$setPropertyValue(property = NULL, value = NULL)
```

Arguments:

`property` The name of the style property to set, e.g. "font-weight".

`value` The value of the style property to set, e.g. "bold".

Returns: No return value.

Method `setPropertyValues()`**:** Set multiple style properties.

Usage:

```
PivotStyle$setPropertyValues(declarations = NULL)
```

Arguments:

`declarations` CSS style declarations in the form of a list, e.g. 'list("font-weight"="bold", "color"="#0000FF")'

Returns: No return value.

Method `getPropertyValue()`**:** Get the value of a single style property.

Usage:

```
PivotStyle$getPropertyValue(property = NULL)
```

Arguments:

`property` The name of the style property to set, e.g. "font-weight".

Returns: The value of the style property.

Method `asCSSRule()`**:** Get the style definition in the form of a CSS rule.

Usage:

```
PivotStyle$asCSSRule(selector = NULL)
```

Arguments:

`selector` A CSS selector, used to select the element(s) to be styled.

Returns: The style declarations in the form of a CSS rule, i.e. selector { property-name1: property-value1, property-name2: property-value2, ... } e.g. div { font-weight: bold, color: #0000FF }

Method `asNamedCSSStyle()`**:** Get the style definition in the form of a named CSS style.

Usage:

```
PivotStyle$asNamedCSSStyle(styleNamePrefix = NULL)
```

Arguments:

`styleNamePrefix` A prefix to prepend to the style name.

Returns: The style declarations in the form of named CSS style, i.e. .prefix-stylename { property-name1: property-value1, property-name2: property-value2, ... } e.g. .pvt1Cell { font-weight: bold, color: #0000FF }

Method `getCopy()`**:** Create a copy of this 'PivotStyle' object.

Usage:

```
PivotStyle$getCopy(newStyleName = NULL)
```

Arguments:

`newStyleName` The name of the new style.

Returns: A 'PivotStyle' object.

Method `asList()`**:** Return the contents of this object as a list for debugging.

Usage:

```
PivotStyle$asList()
```

Returns: A list of various object properties.

Method `asJSON()`**:** Return the contents of this object as JSON for debugging.

Usage:

```
PivotStyle$asJSON()
```

Returns: A JSON representation of various object properties.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PivotStyle$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# This class should only be created by the pivot table.
# It is not intended to be created outside of the pivot table.
```

`PivotStyles` *R6 class that defines a collection of styles.*

Description

The 'PivotStyles' class is a collection of 'PivotStyle' objects. It defines all of the base styles needed to style/theme a pivot table. It also defines the names of the styles that are used for styling the different parts of the pivot table.

Format

`R6Class` object.

Active bindings

`count` The number of 'PivotStyle' objects in this 'PivotStyles' collection.

`theme` A theme name to represent this collection of styles.

`styles` A list containing the 'PivotStyle' objects in this 'PivotStyles' collection.

`allowExternalStyles` Default 'FALSE', which means this 'PivotStyles' object checks that style names specified for styling the different parts of the pivot table must exist in the styles collection. If they do not an error will occur. Specify 'TRUE' to disable this check, e.g. if the style definitions are not managed by 'pivottabler' but instead in an external system.

`tableStyle` The name of the style for the HTML table element.

`rootStyle` The name of the style for the HTML cell at the top left of the pivot table.

`rowHeaderStyle` The name of the style for the row headers in the pivot table.

`colHeaderStyle` The name of the style for the column headers in the pivot table.

`outlineRowHeaderStyle` The name of the style for the outline row headers in the pivot table.

`outlineColHeaderStyle` The name of the style for the outline column headers in the pivot table.

`cellStyle` The name of the cell style for the non-total cells in the body of the pivot table.

`outlineCellStyle` The name of the cell style for the non-total outline cells in the body of the pivot table.

`totalStyle` The name of the cell style for the total cells in the pivot table.

Methods

Public methods:

Method `new()`**:** Create a new 'PivotStyles' object.

Usage:

```
PivotStyles$new(parentPivot, themeName = NULL, allowExternalStyles = FALSE)
```

Arguments:

`parentPivot` The pivot table that this 'PivotStyles' instance belongs to.

`themeName` A theme name to represent this collection of styles.

`allowExternalStyles` Default 'FALSE', which means this 'PivotStyles' object checks that style names specified for styling the different parts of the pivot table must exist in the styles collection. If they do not an error will occur. Specify 'TRUE' to disable this check, e.g. if the style definitions are not managed by 'pivottabler' but instead in an external system.

Returns: A new 'PivotStyles' object.

Method `isExistingStyle()`**:** Check whether a style with the specified style name exists in the collection.

Usage:

```
PivotStyles$isExistingStyle(styleName = NULL)
```

Arguments:

`styleName` The name of the style.

Returns: 'TRUE' if a style with the specified name exists, 'FALSE' otherwise.

Method `getStyle()`**:** Retrieve a style with the specified style name.

Usage:

```
PivotStyles$getStyle(styleName = NULL)
```

Arguments:

`styleName` The name of the style.

Returns: A 'PivotStyle' object.

Method `addStyle()`**:** Add a new style to the collection of styles.

Usage:

```
PivotStyles$addStyle(styleName = NULL, declarations = NULL)
```

Arguments:

`styleName` The name of the new style.

`declarations` CSS style declarations in the form of a list, e.g. 'list("font-weight"="bold", "color"="#0000FF")'

Returns: The newly created 'PivotStyle' object.

Method `copyStyle()`**:** Create a copy of a style with the specified name and store it in the collection.

Usage:

```
PivotStyles$copyStyle(styleName = NULL, newStyleName = NULL)
```

Arguments:

`styleName` The name of the style to copy.

`newStyleName` The name for the new style.

Returns: The newly created 'PivotStyle' object.

Method `asCSSRule()`**:** Get a style definition in the form of a CSS rule.

Usage:

```
PivotStyles$asCSSRule(styleName = NULL, selector = NULL)
```

Arguments:

`styleName` The name of the style.

`selector` A CSS selector, used to select the element(s) to be styled.

Returns: The style declarations in the form of a CSS rule, i.e. selector { property-name1: property-value1, property-name2: property-value2, ... } e.g. div { font-weight: bold, color: #0000FF }

Method `asNamedCSSStyle()`**:** Get a style definition in the form of a named CSS style.

Usage:

```
PivotStyles$asNamedCSSStyle(styleName = NULL, styleNamePrefix = NULL)
```

Arguments:

`styleName` The name of the style.

`styleNamePrefix` A prefix to prepend to the style name.

Returns: The style declarations in the form of named CSS style, i.e. .prefix-stylename { property-name1: property-value1, property-name2: property-value2, ... } e.g. .pvt1Cell { font-weight: bold, color: #0000FF }

Method `asList()`**:** Return the contents of this object as a list for debugging.

Usage:

```
PivotStyles$asList()
```

Returns: A list of various object properties.

Method `asJSON()`**:** Return the contents of this object as JSON for debugging.

Usage:

```
PivotStyles$asJSON()
```

Returns: A JSON representation of various object properties.

Method `asString()`**:** Return a representation of this object as a character value.

Usage:

```
PivotStyles$asString(seperator = ", ")
```

Arguments:

`seperator` A character value to use when concatenating multiple styles.

Returns: A character summary of various object properties.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PivotStyles$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
pt <- PivotTable$new()
# ...
pivotStyles <- PivotStyles$new(pt, themeName="compact")
pivotStyles$addStyle(styleName="MyNewStyle", list(
    font="0.75em arial",
    padding="2px",
    border="1px solid lightgray",
    "vertical-align"="middle",
    "text-align"="center",
    "font-weight"="bold",
    "background-color"="#F2F2F2"
  ))
```

`PivotTable` *R6 class that represents a pivot table.*

Description

The 'PivotTable' class is the primary class for constructing and interacting with a pivot table.

Format

`R6Class` object.

Active bindings

`argumentCheckMode` The level of argument checking to perform. One of "auto", "none", "minimal", "basic", "balanced" (default) or "full".

`compatibility` A list containing compatibility options to force legacy behaviours. See the NEWS file for details.

`traceEnabled` Default 'FALSE'. Specify 'TRUE' to generate a trace for debugging purposes.

`processingLibrary` The package to use when processing data. Must be one of "auto" (which today is dplyr), "dplyr" or "data.table".

`data` A 'PivotData' object containing the data frames added to the pivot table.

`rowGroup` The hidden root 'PivotDataGroup' at the top of the row data groups hierarchy. The children of this group form the first level of visible row data groups.

`columnGroup` The hidden root 'PivotDataGroup' at the top of the column data groups hierarchy. The children of this group form the first level of visible column data groups.

`rowGroupLevelCount` The number of visible levels in the row data group hierarchy.

`columnGroupLevelCount` The number of visible levels in the column data group hierarchy.

`topColumnGroups` A list containing the first level of column data groups.

`leafColumnGroups` A list containing the bottom level of column data groups.

`allColumnGroups` A list containing all of the column data groups.

`topRowGroups` A list containing the first level of row data groups.

`leafRowGroups` A list containing the bottom level of row data groups.

`allRowGroups` A list containing all of the row data groups.

`rowGrpHeaders` A list containing the row group headers.

`calculationGroups` A list containing the calculation groups in the pivot table.

`calculationsPosition` Either "row" or "column" describing which axis the calculations are rendered.

`evaluationMode` Either "batch" (default) or "sequential" (legacy).

`batchInfo` Diagnostic information describing the batches used in the last pivot table evaluation.

`cells` A 'PivotCells' object that contains all of the cells in the pivot table.

`allCells` A list of all of the cells in the pivot table, where each element in the list is a 'PivotCell' object.

`rowCount` The number of rows in the pivot table, excluding headings.

`columnCount` The number of columns in the pivot table, excluding headings.

`fixedWidthSized` The total width of the pivot table in characters if the pivot table were to be rendered as plain text, e.g. to the console.

`asCharacter` A plain text representation of the pivot table.

`theme` The name of the theme used to style the pivot table. If setting this property, either a theme name can be used, or a list can be used (which specifies a simple theme) or a 'PivotStyles' object can be used. See the "Styling" vignette for details and examples.

`styles` A 'PivotStyles' object that contains the styles applied to the pivot table.

`allowExternalStyles` Default 'FALSE', which means the 'PivotStyles' object checks that style names specified for styling the different parts of the pivot table must exist in the styles collection. If they do not an error will occur. Specify 'TRUE' to disable this check, e.g. if the style definitions are not managed by 'pivottabler' but instead in an external system.

`mergeEmptyRowSpace` A character value describing how empty space is merged. Allowed values: "doNotMerge", "dataGroupsOnly", "cellsOnly", "dataGroupsAndCellsAs1", "dataGroupsAndCellsAs2".

`mergeEmptyColumnSpace` A character value describing how empty space is merged. Allowed values: "doNotMerge", "dataGroupsOnly", "cellsOnly", "dataGroupsAndCellsAs1", "dataGroupsAndCellsAs2".

`mergeEmptySpaceDirection` A character value describing how empty space is merged. Allowed values: "row" or "column"

`allTimings` Get a data frame containing timing details of pivot table operations.

`significantTimings` Get a data frame containing timing details of significant pivot table operations (i.e. where elapsed>0.1).

Methods

Public methods:

- `PivotTable$new()`
- `PivotTable$setDefault()`
- `PivotTable$getDefault1()`
- `PivotTable$getDefault2()`
- `PivotTable$getDefault3()`
- `PivotTable$getNextInstanceId()`
- `PivotTable$addData()`
- `PivotTable$addTotalData()`
- `PivotTable$getColumnGroupsByLevel()`
- `PivotTable$getTopColumnGroups()`
- `PivotTable$getLeafColumnGroups()`
- `PivotTable$getLeafColumnGroup()`
- `PivotTable$addColumnGroup()`
- `PivotTable$addColumnDataGroups()`
- `PivotTable$normaliseColumnGroups()`
- `PivotTable$sortColumnDataGroups()`
- `PivotTable$getRowGroupsByLevel()`
- `PivotTable$getTopRowGroups()`
- `PivotTable$getLeafRowGroups()`
- `PivotTable$getLeafRowGroup()`
- `PivotTable$addRowGroup()`
- `PivotTable$addRowDataGroups()`
- `PivotTable$normaliseRowGroups()`
- `PivotTable$sortRowDataGroups()`

- PivotTable$setRowDataGroupHeader()
- PivotTable$addCalculationGroup()
- PivotTable$defineCalculation()
- PivotTable$addColumnCalculationGroups()
- PivotTable$addRowCalculationGroups()
- PivotTable$addStyle()
- PivotTable$createInlineStyle()
- PivotTable$setStyling()
- PivotTable$mapStyling()
- PivotTable$generateCellStructure()
- PivotTable$resetCells()
- PivotTable$evaluateCells()
- PivotTable$evaluatePivot()
- PivotTable$findRowDataGroups()
- PivotTable$findColumnDataGroups()
- PivotTable$getEmptyRows()
- PivotTable$getEmptyColumns()
- PivotTable$getCell()
- PivotTable$getCells()
- PivotTable$findCells()
- PivotTable$findGroupColumnNumbers()
- PivotTable$findGroupRowNumbers()
- PivotTable$removeColumn()
- PivotTable$removeColumns()
- PivotTable$removeEmptyColumns()
- PivotTable$removeRow()
- PivotTable$removeRows()
- PivotTable$removeEmptyRows()
- PivotTable$print()
- PivotTable$asMatrix()
- PivotTable$asDataMatrix()
- PivotTable$asDataFrame()
- PivotTable$asTidyDataFrame()
- PivotTable$getMerges()
- PivotTable$asBasicTable()
- PivotTable$getCss()
- PivotTable$getHtml()
- PivotTable$saveHtml()
- PivotTable$renderPivot()
- PivotTable$getLatex()
- PivotTable$writeToExcelWorksheet()
- PivotTable$trace()

- PivotTable$showBatchInfo()
- PivotTable$asList()
- PivotTable$asJSON()
- PivotTable$viewJSON()
- PivotTable$clone()

Method `new()`**:** Create a new 'PivotTable' object.

Usage:

```
PivotTable$new(
  processingLibrary = "auto",
  evaluationMode = "batch",
  argumentCheckMode = "auto",
  theme = NULL,
  replaceExistingStyles = FALSE,
  tableStyle = NULL,
  headingStyle = NULL,
  cellStyle = NULL,
  totalStyle = NULL,
  compatibility = NULL,
  traceEnabled = FALSE,
  traceFile = NULL
)
```

Arguments:

`processingLibrary` The package to use when processing data. Must be one of "auto" (which today is dplyr), "dplyr" or "data.table".

`evaluationMode` Either "batch" (default) or "sequential" (legacy).

`argumentCheckMode` The level of argument checking to perform. Must be one of "auto", "none", "minimal", "basic", "balanced" (default) or "full".

`theme` A theme to use to style the pivot table. Either:
(1) The name of a built in theme, or
(2) A list of simple style settings, or
(3) A 'PivotStyles' object containing a full set of styles.
See the "Styling" vignette for many examples.

`replaceExistingStyles` Default 'FALSE' to retain existing styles in the styles collection and add specified styles as new custom styles. Specify 'TRUE' to update the definitions of existing styles.

`tableStyle` Styling to apply to the table. Either:
(1) The name of a built in style, or
(2) A list of CSS style declarations, e.g.
'list("font-weight"="bold", "color"="#0000FF")', or
(3) A 'PivotStyle' object.

`headingStyle` Styling to apply to the headings. See the 'tableStyle' argument for details.

`cellStyle` Styling to apply to the normal cells. See the 'tableStyle' argument for details.

`totalStyle` Styling to apply to the total cells. See the 'tableStyle' argument for details.

`compatibility` A list containing compatibility options to force legacy behaviours. See the NEWS file for details.

`traceEnabled` Default 'FALSE'. Specify 'TRUE' to generate a trace for debugging purposes.

`traceFile` If tracing is enabled, the location to generate the trace file.

Returns: A new 'PivotTable' object.

Method `setDefault()`**:** Specify default values for some function arguments.

Usage:

```
PivotTable$setDefault(...)
```

Arguments:

`...` Default values to specify. See details.

Details: Defaults can be set for the following arguments of 'pt$addRowDataGroups()' and 'pt$addColumnDataGroups()': 'logical' values: 'addTotal', 'expandExistingTotals', 'visualTotals'. 'character' values: 'totalPosition', 'totalCaption'. 'list' or 'logical' values: 'outlineBefore', 'outlineAfter', 'outlineTotal'.
Errors are generated for default values that could not be set.
Warnings are generated for attempts to set defaults that aren't supported.
See the "A1. Appendix" vignette for more details.

Returns: No return value.

Method `getDefault1()`**:** Get the default value of an argument.

Usage:

```
PivotTable$getDefault1(argValue = NULL, useDefault = NULL)
```

Arguments:

`argValue` The name and value of the argument.

`useDefault` Specify 'TRUE' to use the default.

Details: Both the argument name and argument value are taken from the 'argValue' argument. The name is obtained from 'as.character(substitute(argValue))'. This function is designed to easily slot into existing code, e.g. 'getDefault1(addTotal, missing(addTotal))'.

Returns: The current value of the argument or the default value.

Method `getDefault2()`**:** Get the default value of an argument.

Usage:

```
PivotTable$getDefault2(argName = NULL, argValue = NULL, useDefault = NULL)
```

Arguments:

`argName` The name of the argument.

`argValue` The current value of the argument.

`useDefault` Specify 'TRUE' to use the default.

Returns: The current value of the argument or the default value.

Method `getDefault3()`**:** Get the default value of an argument.

Usage:

```
PivotTable$getDefault3(argName)
```

Arguments:

`argName` The name of the argument.

Returns: The default value.

Method `getNextInstanceId()`**:** Get the next unique object instance identifier.

Usage:

```
PivotTable$getNextInstanceId()
```

Details: R6 classes cannot be easily compared to check if two variables are both referring to the same object instance. Instance ids are a mechanism to work around this problem. Each data group and cell is assigned an instance id during object creation, which enables reliable reference comparisons.

Returns: An integer instance id.

Method `addData()`**:** Add a data frame with the specified name to the pivot table.

Usage:

```
PivotTable$addData(dataFrame = NULL, dataName = NULL)
```

Arguments:

`dataFrame` The data frame to add.

`dataName` The name to be used to refer to the data frame. If no name is specified, the data frame variable name from the calling code is used, retrieved via 'deparse(substitute(dataFrame))'.

Details: The name is used to refer to the data frame when generating data groups or defining calculations. The pivot table tracks the first data frame added as the default data frame, so if only a single data frame is used, it is typically not necessary to ever explicitly refer to the name. Pivot tables are typically based on a single data frame, however it is possible to build a pivot table that uses data from multiple data frames.

Returns: The 'PivotData' object managing the data frames for the pivot table.

Method `addTotalData()`**:** Add a data frame containing totals data with the specified name and variables to the pivot table.

Usage:

```
PivotTable$addTotalData(
  dataFrame = NULL,
  dataName = NULL,
  variableNames = NULL
)
```

Arguments:

`dataFrame` The data frame to add.

`dataName` The name of the data frame to associate these totals with.

`variableNames` A vector specifying how the aggregate data/totals in the data frame are grouped.

Details: When generating pivot tables, the package typically calculates cell values. However, the package can also use provided values (i.e. carry out no calculations). This presents a challenge in that the sub-totals and totals in a pivot table display values at a higher aggregation level than the normal cells in the body of the pivot table. This method allows further data frames to be specified that contain aggregated versions of the data. See the "Calculations" vignette for details and an example.

Returns: No return value.

Method `getColumnGroupsByLevel()`**:** Retrieve the data groups at the specified level or levels in the column groups hierarchy.

Usage:

```
PivotTable$getColumnGroupsByLevel(level = NULL, collapse = FALSE)
```

Arguments:

`level` An integer value or vector specifying one or more level numbers. Level 1 represents the first visible level of data groups.

`collapse` A logical value specifying whether the return value should be simplified. See details.

Details: If 'level' is a vector: If 'collapse' is 'FALSE', then a list of lists is returned, if 'collapse' is 'TRUE', then a single combined list is returned.

Returns: A list containing 'PivotDataGroup' objects.

Method `getTopColumnGroups()`**:** [Deprecated: Use topColumnGroups instead] Retrieve the first level of column data groups.

Usage:

```
PivotTable$getTopColumnGroups()
```

Returns: A list containing 'PivotDataGroup' objects.

Method `getLeafColumnGroups()`**:** [Deprecated: Use leafColumnGroups instead] Retrieve the bottom level of column data groups.

Usage:

```
PivotTable$getLeafColumnGroups()
```

Returns: A list containing 'PivotDataGroup' objects.

Method `getLeafColumnGroup()`**:** Retrieve the leaf-level data group associated with a specific column or columns.

Usage:

```
PivotTable$getLeafColumnGroup(c = NULL)
```

Arguments:

`c` An integer column number or an integer vector of column numbers.

Returns: A 'PivotDataGroup' object or a list of 'PivotDataGroup' objects.

Method `addColumnGroup()`**:** Add a new column data group at the top level of the column group hierarchy. The new group is added as the last child unless an index is specified.

Usage:

```
PivotTable$addColumnGroup(
  variableName = NULL,
  filterType = "ALL",
  values = NULL,
  doNotExpand = FALSE,
  isEmpty = FALSE,
  isOutline = FALSE,
```

```
  styleAsOutline = FALSE,
  captionTemplate = "{value}",
  caption = NULL,
  isTotal = FALSE,
  isLevelSubTotal = FALSE,
  isLevelTotal = FALSE,
  calculationGroupName = NULL,
  calculationName = NULL,
  baseStyleName = NULL,
  styleDeclarations = NULL,
  insertAtIndex = NULL,
  insertBeforeGroup = NULL,
  insertAfterGroup = NULL,
  mergeEmptySpace = NULL,
  cellBaseStyleName = NULL,
  cellStyleDeclarations = NULL,
  sortAnchor = NULL,
  resetCells = TRUE
)
```

Arguments:

`variableName` A character value that specifies the name of the variable in the data frame that the group relates to and will filter.

`filterType` Must be one of "ALL", "VALUES", or "NONE" to specify the filter type:
ALL means no filtering is applied.
VALUEs is the typical value used to specify that ‘variableName‘ is filtered to only ‘values‘.
NONE means no data will match this data group.

`values` A vector that specifies the filter values applied to ‘variableName‘ to select the data to match this row/column in the pivot table.

`doNotExpand` Default value ‘FALSE‘ - specify ‘TRUE‘ to prevent the high-level methods such as ‘addDataGroups()‘ from adding child groups.

`isEmpty` Default value ‘FALSE‘, specify ‘TRUE‘ to mark that this group contains no data (e.g. if it is part of a header or outline row)

`isOutline` Default value ‘FALSE‘ - specify ‘TRUE‘ to mark that this data group is an outline group.

`styleAsOutline` Default value ‘FALSE‘ - specify ‘TRUE‘ to style this data group as an outline group. Only applicable when ‘isOutline‘ is ‘TRUE‘.

`captionTemplate` A character value that specifies the template for the data group caption, default "{values}".

`caption` Effectively a hard-coded caption that overrides the built-in logic for generating a caption.

`isTotal` Default ‘FALSE‘ - specify ‘TRUE‘ to mark that this data group is a total.

`isLevelSubTotal` Default ‘FALSE‘ - specify ‘TRUE‘ to mark that this data group is a sub-total within a level.

`isLevelTotal` Default ‘FALSE‘ - specify ‘TRUE‘ to mark that this data group is level total.

`calculationGroupName` For calculation groups, this character value specifies the calculation group that ‘calculationName‘ belongs to.

`calculationName` For calculation groups, this character value specifies the name of the calculation.

`baseStyleName` The style name for the data group.

`styleDeclarations` A list of CSS style declarations to overlay on top of the base style.

`insertAtIndex` An integer that specifies the index in the list of child groups where the new group should be inserted.

`insertBeforeGroup` Specifies an existing group that the new group should be inserted before.

`insertAfterGroup` Specifies an existing group that the new group should be inserted after

`mergeEmptySpace` A character value that specifies how empty space should be merged. This is typically only used with outline groups (so applies to row groups only, not column groups). Must be one of "doNotMerge", "dataGroupsOnly", "cellsOnly", "dataGroupsAndCellsAs1" or "dataGroupsAndCellsAs2". See the "Regular Layout" vignette for more information.

`cellBaseStyleName` The style name for cells related to this data group.

`cellStyleDeclarations` A list of CSS style declarations to overlay on top of the base style for cells related to this data group

`sortAnchor` Used to specify sort behaviour for outline groups, must be one of "fixed", "next" or "previous".

`resetCells` Default 'TRUE' to reset any cells that currently exist in the pivot table and trigger a recalculation of the pivot table when it is next rendered.

Details: See the "Irregular Layout" vignette for details and examples.

Returns: The new 'PivotDataGroup' object.

Method `addColumnDataGroups()`**:** Add multiple new data groups to the column group hierarchy based on the distinct values in a data frame column or using explicitly specified data values. See the "Data Groups" vignette for example usage.

Usage:

```
PivotTable$addColumnDataGroups(
  variableName = NULL,
  atLevel = NULL,
  fromData = TRUE,
  dataName = NULL,
  dataSortOrder = "asc",
  customSortOrder = NULL,
  caption = "{value}",
  dataFormat = NULL,
  dataFmtFuncArgs = NULL,
  onlyCombinationsThatExist = TRUE,
  explicitListOfValues = NULL,
  calculationGroupName = NULL,
  expandExistingTotals = FALSE,
  addTotal = TRUE,
  visualTotals = FALSE,
  totalPosition = "after",
  totalCaption = "Total",
  onlyAddGroupIf = NULL,
  preGroupData = TRUE,
```

```
  baseStyleName = NULL,
  styleDeclarations = NULL
)
```

Arguments:

`variableName` The name of the related column in the data frame(s) of the pivot table.

`atLevel` The level number that specifies where to add the new groups. Level 1 = on the first visible level of the hierarchy. 'NULL' = create a new level at the bottom of the hierarchy for the new groups.

`fromData` Default 'TRUE' to generate the new data groups based on the data values that exist in the 'variableName' column in the named data frame. If 'FALSE', then 'explicitListOfValues' must be specified.

`dataName` The name of the data frame (as specified in 'pt$addData()') to read the data group values from.

`dataSortOrder` Must be one of "asc", "desc", "custom" or "none".

`customSortOrder` A vector values sorted into the desired order.

`caption` The template of data group captions to generate, default "{value}".

`dataFormat` A character, list or custom function to format the data value.

`dataFmtFuncArgs` A list that specifies any additional arguments to pass to a custom format function.

`onlyCombinationsThatExist` Default 'TRUE' to generate only combinations of data groups that exist in the data frame.

`explicitListOfValues` A list of explicit values to create data groups from. A data group is created for each element of the list. If a list element is vector of values (with length greater than 1), then a data group is created for multiple values instead of just a single value.

`calculationGroupName` The calculation group that the new data groups are related to.

`expandExistingTotals` Default 'FALSE', which means totals are not broken down in multi-level hierarchies.

`addTotal` Default 'TRUE', which means sub-total and total data groups are automatically added.

`visualTotals` Default 'FALSE', which means visual totals are disabled. See the "Data Groups" vignette for more details about visual totals.

`totalPosition` Either "before" or "after" to specify where total groups are created, default "after".

`totalCaption` The caption to display on total groups, default "Total".

`onlyAddGroupIf` A filter expression that can be used to more finely control whether data groups are created at different locations in the hierarchy. There must be at least one row that matches this filter and the filters from the ancestor groups in order that the child group is created. E.g. 'MaxDisplayLevel>5'.

`preGroupData` Default 'TRUE', which means that the pivot table pre-calculates the distinct combinations of variable values to reduce the CPU time and elapsed time required to generate data groups. Cannot be used in conjunction with the

`baseStyleName` The name of the style applied to this data group (i.e. this row/column heading). The style must exist in the 'PivotStyles' object associated with the PivotTable.

`styleDeclarations` CSS style declarations that can override the base style, expressed as a list, e.g. 'list("font-weight"=bold")'.

Details: There are broadly three different ways to call 'addColumnDataGroups()':
(1) dataName=name, fromData=TRUE, onlyCombinationsThatExist=TRUE - which considers the ancestors of each existing data group to generate only those combinations of values that exist in the data frame.
(2) dataName=name, fromData=TRUE, onlyCombinationsThatExist=FALSE - which ignores the ancestors of each existing data group and simply adds every distinct value of the specified variable under every existing data group, which can result in combinations of values in the pivot table that don't exist in the data frame (i.e. blank rows/columns in the pivot table).
(3) fromData=FALSE, explicitListOfValues=list(...) - simply adds every value from the specified list under every existing data group.

Returns: A list of new 'PivotDataGroup' objects that have been added.

Method `normaliseColumnGroups()`**:** Normalise the column data group hierarchy so that all branches have the same number of levels - accomplished by adding empty child data groups where needed.

Usage:

```
PivotTable$normaliseColumnGroups(resetCells = TRUE)
```

Arguments:

`resetCells` Default 'TRUE' to reset any cells that currently exist in the pivot table and trigger a recalculation of the pivot table when it is next rendered.

Returns: A list of new 'PivotDataGroup' objects that have been added.

Method `sortColumnDataGroups()`**:** Sort column data groups either by the data group data value, caption, a custom order or based on calculation result values.

Usage:

```
PivotTable$sortColumnDataGroups(
  levelNumber = 1,
  orderBy = "calculation",
  customOrder = NULL,
  sortOrder = "desc",
  calculationGroupName = "default",
  calculationName = NULL,
  fromIndex = NULL,
  toIndex = NULL,
  resetCells = TRUE
)
```

Arguments:

`levelNumber` The level number to sort the data groups, e.g. level 1 (default) sorts the data groups at level 1 of the hierarchy (which is the first visible level of data groups).

`orderBy` Must be either "value", "caption", "calculation", "customByValue" or "customByCaption".
"value" sorts by the raw (i.e. unformatted) group value.
"caption" sorts by the formatted character group caption.
"calculation" sorts using one of the calculations defined in the pivot table.
"customValue" sorts by the raw (i.e. unformatted) group value according to the specified custom sort order.

"customCaption" sorts by the formatted character group caption according to the specified custom sort order.

`customOrder` A vector values sorted into the desired order.

`sortOrder` Must be either "asc" or "desc".

`calculationGroupName` If sorting using a calculation, the name of the calculation group containing the specified calculation.

`calculationName` If sorting using a calculation, the name of the calculation.

`fromIndex` A boundary to limit the sort operation.

`toIndex` A boundary to limit the sort operation.

`resetCells` Default 'TRUE' to reset any cells that currently exist in the pivot table and trigger a recalculation of the pivot table when it is next rendered.

Returns: No return value.

Method `getRowGroupsByLevel()`**:** Retrieve the data groups at the specified level or levels in the row groups hierarchy.

Usage:

```
PivotTable$getRowGroupsByLevel(level = NULL, collapse = FALSE)
```

Arguments:

`level` An integer value or vector specifying one or more level numbers. Level 1 represents the first visible level of data groups.

`collapse` A logical value specifying whether the return value should be simplified. See details.

Details: If 'level' is a vector: If 'collapse' is 'FALSE', then a list of lists is returned, if 'collapse' is 'TRUE', then a single combined list is returned.

Returns: A list containing 'PivotDataGroup' objects.

Method `getTopRowGroups()`**:** [Deprecated: Use topRowGroups instead] Retrieve the first level of row data groups.

Usage:

```
PivotTable$getTopRowGroups()
```

Returns: A list containing 'PivotDataGroup' objects.

Method `getLeafRowGroups()`**:** [Deprecated: Use leafRowGroups instead] Retrieve the bottom level of row data groups.

Usage:

```
PivotTable$getLeafRowGroups()
```

Returns: A list containing 'PivotDataGroup' objects.

Method `getLeafRowGroup()`**:** Retrieve the leaf-level data group associated with a specific row or rows.

Usage:

```
PivotTable$getLeafRowGroup(r = NULL)
```

Arguments:

`r` An integer row number or an integer vector of row numbers.

Returns: A ‘PivotDataGroup‘ object or a list of ‘PivotDataGroup‘ objects.

Method `addRowGroup()`**:** Add a new column data group at the top level of the row group hierarchy. The new group is added as the last child unless an index is specified.

Usage:

```
PivotTable$addRowGroup(
  variableName = NULL,
  filterType = "ALL",
  values = NULL,
  doNotExpand = FALSE,
  isEmpty = FALSE,
  isOutline = FALSE,
  styleAsOutline = FALSE,
  captionTemplate = "{value}",
  caption = NULL,
  isTotal = FALSE,
  isLevelSubTotal = FALSE,
  isLevelTotal = FALSE,
  calculationGroupName = NULL,
  calculationName = NULL,
  baseStyleName = NULL,
  styleDeclarations = NULL,
  insertAtIndex = NULL,
  insertBeforeGroup = NULL,
  insertAfterGroup = NULL,
  mergeEmptySpace = NULL,
  cellBaseStyleName = NULL,
  cellStyleDeclarations = NULL,
  sortAnchor = NULL,
  resetCells = TRUE
)
```

Arguments:

`variableName` A character value that specifies the name of the variable in the data frame that the group relates to and will filter.

`filterType` Must be one of "ALL", "VALUES", or "NONE" to specify the filter type:
ALL means no filtering is applied.
VALUEs is the typical value used to specify that ‘variableName‘ is filtered to only ‘values‘.
NONE means no data will match this data group.

`values` A vector that specifies the filter values applied to ‘variableName‘ to select the data to match this row/column in the pivot table.

`doNotExpand` Default value ‘FALSE‘ - specify ‘TRUE‘ to prevent the high-level methods such as ‘addDataGroups()‘ from adding child groups.

`isEmpty` Default value ‘FALSE‘, specify ‘TRUE‘ to mark that this group contains no data (e.g. if it is part of a header or outline row)

`isOutline` Default value ‘FALSE‘ - specify ‘TRUE‘ to mark that this data group is an outline group.

`styleAsOutline` Default value 'FALSE' - specify 'TRUE' to style this data group as an outline group. Only applicable when 'isOutline' is 'TRUE'.

`captionTemplate` A character value that specifies the template for the data group caption, default "{values}".

`caption` Effectively a hard-coded caption that overrides the built-in logic for generating a caption.

`isTotal` Default 'FALSE' - specify 'TRUE' to mark that this data group is a total.

`isLevelSubTotal` Default 'FALSE' - specify 'TRUE' to mark that this data group is a sub-total within a level.

`isLevelTotal` Default 'FALSE' - specify 'TRUE' to mark that this data group is level total.

`calculationGroupName` For calculation groups, this character value specifies the calculation group that 'calculationName' belongs to.

`calculationName` For calculation groups, this character value specifies the name of the calculation.

`baseStyleName` The style name for the data group.

`styleDeclarations` A list of CSS style declarations to overlay on top of the base style.

`insertAtIndex` An integer that specifies the index in the list of child groups where the new group should be inserted.

`insertBeforeGroup` Specifies an existing group that the new group should be inserted before.

`insertAfterGroup` Specifies an existing group that the new group should be inserted after

`mergeEmptySpace` A character value that specifies how empty space should be merged. This is typically only used with outline groups (so applies to row groups only, not column groups). Must be one of "doNotMerge", "dataGroupsOnly", "cellsOnly", "dataGroupsAndCellsAs1" or "dataGroupsAndCellsAs2". See the "Regular Layout" vignette for more information.

`cellBaseStyleName` The style name for cells related to this data group.

`cellStyleDeclarations` A list of CSS style declarations to overlay on top of the base style for cells related to this data group

`sortAnchor` Used to specify sort behaviour for outline groups, must be one of "fixed", "next" or "previous".

`resetCells` Default 'TRUE' to reset any cells that currently exist in the pivot table and trigger a recalculation of the pivot table when it is next rendered.

`outlineLinkedGroupId` Used to link an outline group to the value data group which has the child data groups.

Details: See the "Irregular Layout" vignette for details and examples.

Returns: The new 'PivotDataGroup' object.

Method `addRowDataGroups()`**:** Add multiple new data groups to the row group hierarchy based on the distinct values in a data frame column or using explicitly specified data values. See the "Data Groups" vignette for example usage.

Usage:

```
PivotTable$addRowDataGroups(
  variableName = NULL,
  atLevel = NULL,
  fromData = TRUE,
  dataName = NULL,
```

```
  dataSortOrder = "asc",
  customSortOrder = NULL,
  caption = "{value}",
  dataFormat = NULL,
  dataFmtFuncArgs = NULL,
  onlyCombinationsThatExist = TRUE,
  explicitListOfValues = NULL,
  calculationGroupName = NULL,
  expandExistingTotals = FALSE,
  addTotal = TRUE,
  visualTotals = FALSE,
  totalPosition = "after",
  totalCaption = "Total",
  onlyAddGroupIf = NULL,
  preGroupData = TRUE,
  baseStyleName = NULL,
  styleDeclarations = NULL,
  header = NULL,
  outlineBefore = NULL,
  outlineAfter = NULL,
  outlineTotal = NULL,
  onlyAddOutlineChildGroupIf = NULL
)
```

Arguments:

`variableName` The name of the related column in the data frame(s) of the pivot table.

`atLevel` The level number that specifies where to add the new groups. Level 1 = on the first visible level of the hierarchy. 'NULL' = create a new level at the bottom of the hierarchy for the new groups.

`fromData` Default 'TRUE' to generate the new data groups based on the data values that exist in the 'variableName' column in the named data frame. If 'FALSE', then 'explicitListOfValues' must be specified.

`dataName` The name of the data frame (as specified in 'pt$addData()') to read the data group values from.

`dataSortOrder` Must be one of "asc", "desc", "custom" or "none".

`customSortOrder` A vector values sorted into the desired order.

`caption` The template of data group captions to generate, default "{value}".

`dataFormat` A character, list or custom function to format the data value.

`dataFmtFuncArgs` A list that specifies any additional arguments to pass to a custom format function.

`onlyCombinationsThatExist` Default 'TRUE' to generate only combinations of data groups that exist in the data frame.

`explicitListOfValues` A list of explicit values to create data groups from. A data group is created for each element of the list. If a list element is vector of values (with length greater than 1), then a data group is created for multiple values instead of just a single value.

`calculationGroupName` The calculation group that the new data groups are related to.

`expandExistingTotals` Default 'FALSE', which means totals are not broken down in multi-level hierarchies.

`addTotal` Default 'TRUE', which means sub-total and total data groups are automatically added.

`visualTotals` Default 'FALSE', which means visual totals are disabled. See the "Data Groups" vignette for more details about visual totals.

`totalPosition` Either "before" or "after" to specify where total groups are created, default "after".

`totalCaption` The caption to display on total groups, default "Total".

`onlyAddGroupIf` A filter expression that can be used to more finely control whether data groups are created at different locations in the hierarchy. There must be at least one row that matches this filter and the filters from the ancestor groups in order that the child group is created. E.g. 'MaxDisplayLevel>5'.

`preGroupData` Default 'TRUE', which means that the pivot table pre-calculates the distinct combinations of variable values to reduce the CPU time and elapsed time required to generate data groups. Cannot be used in conjunction with the

`baseStyleName` The name of the style applied to this data group (i.e. this row/column heading). The style must exist in the 'PivotStyles' object associated with the PivotTable.

`styleDeclarations` CSS style declarations that can override the base style, expressed as a list, e.g. 'list("font-weight"=bold)'.

`header` A character value used as the row-group column caption when row group headers are rendered.

`outlineBefore` Default 'FALSE' to disable the creation of outline header groups. Specify either 'TRUE' or a list of outline group settings to create outline header groups. See the "Regular Layout" vignette for details.

`outlineAfter` Default 'FALSE' to disable the creation of outline footer groups. Specify either 'TRUE' or a list of outline group settings to create outline footer groups. See the "Regular Layout" vignette for details.

`outlineTotal` Default 'FALSE' to disable the creation of outline totals. Specify either 'TRUE' or a list of outline group settings to create outline totals. See the "Regular Layout" vignette for details.

`onlyAddOutlineChildGroupIf` A filter expression that can be used to more finely control whether outline child groups are created at different locations in the hierarchy. There must be at least one row that matches this filter and the filters from the ancestor groups in order that the outline child group is created. E.g. 'MaxDisplayLevel>5'. See the "Regular Layout" vignette for an example.

Details: There are broadly three different ways to call 'addRowDataGroups()':
(1) dataName=name, fromData=TRUE, onlyCombinationsThatExist=TRUE - which considers the ancestors of each existing data group to generate only those combinations of values that exist in the data frame.
(2) dataName=name, fromData=TRUE, onlyCombinationsThatExist=FALSE - which ignores the ancestors of each existing data group and simply adds every distinct value of the specified variable under every existing data group, which can result in combinations of values in the pivot table that don't exist in the data frame (i.e. blank rows/columns in the pivot table).
(3) fromData=FALSE, explicitListOfValues=list(...) - simply adds every value from the specified list under every existing data group.

Returns: A list of new 'PivotDataGroup' objects that have been added.

Method `normaliseRowGroups()`**:** Normalise the row data group hierarchy so that all branches have the same number of levels - accomplished by adding empty child data groups where needed.

Usage:

```
PivotTable$normaliseRowGroups(resetCells = TRUE)
```

Arguments:

`resetCells` Default 'TRUE' to reset any cells that currently exist in the pivot table and trigger a recalculation of the pivot table when it is next rendered.

Returns: A list of new 'PivotDataGroup' objects that have been added.

Method `sortRowDataGroups()`**:** Sort row data groups either by the data group data value, caption, a custom order or based on calculation result values.

Usage:

```
PivotTable$sortRowDataGroups(
  levelNumber = 1,
  orderBy = "calculation",
  customOrder = NULL,
  sortOrder = "desc",
  calculationGroupName = "default",
  calculationName = NULL,
  fromIndex = NULL,
  toIndex = NULL,
  resetCells = TRUE
)
```

Arguments:

`levelNumber` The level number to sort the data groups, e.g. level 1 (default) sorts the data groups at level 1 of the hierarchy (which is the first visible level of data groups).

`orderBy` Must be either "value", "caption", "calculation", "customByValue" or "customByCaption".
"value" sorts by the raw (i.e. unformatted) group value.
"caption" sorts by the formatted character group caption.
"calculation" sorts using one of the calculations defined in the pivot table. "customValue" sorts by the raw (i.e. unformatted) group value according to the specified custom sort order.
"customCaption" sorts by the formatted character group caption according to the specified custom sort order.

`customOrder` A vector values sorted into the desired order.

`sortOrder` Must be either "asc" or "desc".

`calculationGroupName` If sorting using a calculation, the name of the calculation group containing the specified calculation.

`calculationName` If sorting using a calculation, the name of the calculation.

`fromIndex` A boundary to limit the sort operation.

`toIndex` A boundary to limit the sort operation.

`resetCells` Default 'TRUE' to reset any cells that currently exist in the pivot table and trigger a recalculation of the pivot table when it is next rendered.

Returns: No return value.

Method `setRowDataGroupHeader()`**:** Set the row group header associated with a level of the row data group hierarchy.

Usage:

```
PivotTable$setRowDataGroupHeader(levelNumber = NULL, header = NULL)
```

Arguments:

`levelNumber` An integer specifying the level number.

`header` A character value specifying the caption.

Details: By default, the row data groups (i.e. row headings) at the left of the pivot table have no column headings. This method can specify the headings, which can be rendered by specifying the 'showRowGroupHeaders=TRUE' in the render methods.

Returns: No return value.

Method `addCalculationGroup()`**:** Create a new calculation group. This is rarely needed since the default group is sufficient for all regular pivot tables.

Usage:

```
PivotTable$addCalculationGroup(calculationGroupName = NULL)
```

Arguments:

`calculationGroupName` The name of the new calculation group to create.

Returns: A 'PivotCalculationGroup' object.

Method `defineCalculation()`**:** Create a new 'PivotCalculation' object.

Usage:

```
PivotTable$defineCalculation(
  calculationGroupName = "default",
  calculationName = NULL,
  caption = NULL,
  visible = TRUE,
  displayOrder = NULL,
  filters = NULL,
  format = NULL,
  fmtFuncArgs = NULL,
  dataName = NULL,
  type = "summary",
  valueName = NULL,
  summariseExpression = NULL,
  calculationExpression = NULL,
  calculationFunction = NULL,
  calcFuncArgs = NULL,
  basedOn = NULL,
  noDataValue = NULL,
  noDataCaption = NULL,
  headingBaseStyleName = NULL,
  headingStyleDeclarations = NULL,
  cellBaseStyleName = NULL,
  cellStyleDeclarations = NULL,
  resetCells = TRUE
)
```

Arguments:

`calculationGroupName` The name of the calculation group this calculation will belong to. The default calculation group will be used if this parameter is not specified (this is sufficient for all regular pivot tables).

`calculationName` Calculation unique name.

`caption` Calculation display name

`visible` 'TRUE' to show the calculation in the pivot table or 'FALSE' to hide it. Hidden calculations are typically used as base values for other calculations.

`displayOrder` The order the calculations are displayed in the pivot table.

`filters` Any additional data filters specific to this calculation. This can be a 'PivotFilters' object that further restricts the data for the calculation or a list of individual 'PivotFilter' objects that provide more flexibility (and/or/replace). See the Calculations vignette for details.

`format` A character, list or custom function to format the calculation result.

`fmtFuncArgs` A list that specifies any additional arguments to pass to a custom format function.

`dataName` Specifies which data frame in the pivot table is used for this calculation (as specified in 'pt$addData()').

`type` The calculation type: "summary", "calculation", "function" or "value".

`valueName` For type="value", the name of the column containing the value to display in the pivot table.

`summariseExpression` For type="summary", either the dplyr expression to use with dplyr::summarise() or a data.table calculation expression.

`calculationExpression` For type="calculation", an expression to combine aggregate values.

`calculationFunction` For type="function", a reference to a custom R function that will carry out the calculation.

`calcFuncArgs` For type="function", a list that specifies additional arguments to pass to calculationFunction.

`basedOn` A character vector specifying the names of one or more calculations that this calculation depends on.

`noDataValue` An integer or numeric value specifying the value to use if no data exists for a particular cell.

`noDataCaption` A character value that will be displayed by the pivot table if no data exists for a particular cell.

`headingBaseStyleName` The name of a style defined in the pivot table to use as the base styling for the data group heading.

`headingStyleDeclarations` A list of CSS style declarations (e.g. 'list("font-weight"="bold")') to override the base style.

`cellBaseStyleName` The name of a style defined in the pivot table to use as the base styling for the cells related to this calculation.

`cellStyleDeclarations` A list of CSS style declarations (e.g. 'list("font-weight"="bold")') to override the base style.

`resetCells` Default 'TRUE' to reset any cells that currently exist in the pivot table and trigger a recalculation of the pivot table when it is next rendered.

Returns: A new 'PivotCalculation' object.

Method `addColumnCalculationGroups()`: Set calculations on existing data groups or add multiple new groups to the column data group hierarchy to represent calculations.

Usage:

```
PivotTable$addColumnCalculationGroups(
  calculationGroupName = "default",
  atLevel = NULL
)
```

Arguments:

`calculationGroupName` The name of the calculation group to add into the data group hierarchy.

`atLevel` The level number that specifies where to add the new groups. Level 1 = on the first visible level of the hierarchy. 'NULL' = create a new level at the bottom of the hierarchy for the new groups.

Details: If only one calculation is defined in the pivot table, then the calculation is set onto the existing column data groups (and no new groups are generated). If multiple calculations are defined, then a new level of data groups is added, e.g. if two calculations are defined, then two new data groups will be created under each existing leaf-level column data group.

Returns: A list of new 'PivotDataGroup' objects that have been added.

Method `addRowCalculationGroups()`: Set calculations on existing data groups or add multiple new groups to the row data group hierarchy to represent calculations.

Usage:

```
PivotTable$addRowCalculationGroups(
  calculationGroupName = "default",
  atLevel = NULL,
  outlineBefore = NULL,
  outlineAfter = NULL
)
```

Arguments:

`calculationGroupName` The name of the calculation group to add into the data group hierarchy.

`atLevel` The level number that specifies where to add the new groups. Level 1 = on the first visible level of the hierarchy. 'NULL' = create a new level at the bottom of the hierarchy for the new groups.

`outlineBefore` Default 'FALSE' to disable the creation of outline header groups. Specify either 'TRUE' or a list of outline group settings to create outline header groups. See the "Regular Layout" vignette for details.

`outlineAfter` Default 'FALSE' to disable the creation of outline footer groups. Specify either 'TRUE' or a list of outline group settings to create outline footer groups. See the "Regular Layout" vignette for details.

Details: If only one calculation is defined in the pivot table, then the calculation is set onto the existing row data groups (and no new groups are generated). If multiple calculations are defined, then a new level of data groups is added, e.g. if two calculations are defined, then two new data groups will be created under each existing leaf-level row data group.

Returns: A list of new 'PivotDataGroup' objects that have been added.

Method `addStyle()`**:** Add a new named style to the pivot table.

Usage:

```
PivotTable$addStyle(styleName = NULL, declarations = NULL)
```

Arguments:

`styleName` The name of the new style.

`declarations` CSS style declarations in the form of a list, e.g. 'list("font-weight"="bold", "color"="#0000FF")'

Returns: The newly created 'PivotStyle' object.

Method `createInlineStyle()`**:** Create an inline style that can be used to override a base style. For general use cases, the 'setStyling()' method provides a simpler and more direct way of styling specific parts of a pivot table.

Usage:

```
PivotTable$createInlineStyle(baseStyleName = NULL, declarations = NULL)
```

Arguments:

`baseStyleName` The name of an existing style to base the new style on.

`declarations` CSS style declarations in the form of a list, e.g. 'list("font-weight"="bold", "color"="#0000FF")'

Details: Inline styles are typically used to override the style of some specific cells in a pivot table. Inline styles have no name. In HTML, they are rendered as 'style' attributes on specific table cells, where as named styles are linked to cells using the 'class' attribute.

Returns: The newly created 'PivotStyle' object.

Method `setStyling()`**:** Apply styling to a set of data groups or cells in a pivot table.

Usage:

```
PivotTable$setStyling(
  rFrom = NULL,
  cFrom = NULL,
  rTo = NULL,
  cTo = NULL,
  rowNumbers = NULL,
  columnNumbers = NULL,
  groups = NULL,
  cells = NULL,
  baseStyleName = NULL,
  style = NULL,
  declarations = NULL,
  applyBorderToAdjacentCells = FALSE
)
```

Arguments:

`rFrom` An integer row number that specifies the start row for the styling changes.

`cFrom` An integer column number that specifies the start column for the styling changes.

`rTo` An integer row number that specifies the end row for the styling changes.

`cTo` An integer column number that specifies the end column for the styling changes.

`rowNumbers` An integer vector that specifies the row numbers for the styling changes.

`columnNumbers` An integer vector that specifies the column numbers for the styling changes.

`groups` A list containing 'PivotDataGroup' objects.

`cells` A list containing 'PivotCell' objects.

`baseStyleName` The name of a style to apply.

`style` A 'PivotStyle' object to apply.

`declarations` CSS style declarations to apply in the form of a list, e.g. 'list("font-weight"="bold", "color"="#0000FF")'

`applyBorderToAdjacentCells` TRUE to override the border in neighbouring cells, e.g. the left border of the current cell becomes the right border of the cell to the left. Does not apply to row/column groups.

Details: There are five ways to specify the part(s) of a pivot table to apply styling to:
(1) By specifying a list of data groups using the 'groups' argument.
(2) By specifying a list of cells using the 'cells' argument.
(3) By specifying a single cell using the 'rFrom' and 'cFrom' arguments.
(4) By specifying a rectangular cell range using the 'rFrom', 'cFrom', 'rTo' and 'cTo' arguments.
(5) By specifying a vector of rowNumbers and/or columnNumbers. If both rowNumbers and columnNumbers are specified, then the cells at the intersection of the specified row numbers and column numbers are styled.
If both rFrom/rTo and rowNumbers are specified, then rFrom/rTo constrain the row numbers specified in rowNumbers.
If both cFrom/cTo and columnNumbers are specified, then cFrom/cTo constrain the column numbers specified in columnNumbers.
See the "Styling" and "Finding and Formatting" vignettes for more information and many examples.

Returns: No return value.

Method `mapStyling()`**:** Apply styling to pivot table cells based on the value of each cell.

Usage:

```
PivotTable$mapStyling(
  styleProperty = NULL,
  cells = NULL,
  valueType = "text",
  mapType = "range",
  mappings = NULL,
  styleLowerValues = FALSE,
  styleHigherValues = TRUE
)
```

Arguments:

`styleProperty` The name of the style property to set on the specified cells, e.g. backgroundcolor.

`cells` A list containing 'PivotCell' objects.

`valueType` The type of style value to be set. Must be one of: "text", "character", "number", "numeric", "color" or "colour".
"text" and "character" are equivalent. "number" and "numeric" are equivalent. "color" and "colour" are equivalent.

`mapType` The type of mapping to be performed. The following mapping types are supported:
(1) "value" = a 1:1 mapping which maps each specified "from" value to the corresponding "to" value, e.g. 100 -> "green".
(2) "logic" = each from value is logical criteria. See details.
(3) "range" = values between each pair of "from" values are mapped to the corresponding "to" value, e.g. values in the range 80-100 -> "green" (more specifically values greater than or equal to 80 and less than 100).
(4) "continuous" = rescales values between each pair of "from" values into the range of the corresponding pair of "to" values, e.g. if the "from" range is 80-100 and the corresponding "to" range is 0.8-1, then 90 -> 0.9.
"continuous" cannot be used with valueType="text"/"character".

`mappings` The mappings to be applied, specified in one of the following three forms:
(1) a list containing pairs of values, e.g. 'list(0, "red", 0.4, "yellow", 0.8, "green")'.
(2) a list containing "from" and "to" vectors/lists, e.g. 'list(from=c(0, 0.4, 0.8), to=c("red", "yellow", "green"))'.
(3) a custom mapping function that will be invoked once per cell, e.g. 'function(v, cell) { if(isTRUE(v>0.8)) return("green") }'.
Mappings must be specified in ascending order when valueType="range" or valueType="continuous".
If a custom mapping function is specified, then the valueType and mapType parameters are ignored.

`styleLowerValues` A logical value, default 'FALSE', that specifies whether values less than the lowest specified "from" value should be styled using the style specified for the lowest "from" value. Only applies when valueType="range" or valueType="continuous".

`styleHigherValues` A logical value, default 'TRUE', that specifies whether values greater than the highest specified "from" value should be styled using the style specified for the highest "from" value. Only applies when valueType="range" or valueType="continuous".

Details: 'mapStyling()' is typically used to conditionally apply styling to cells based on the value of each individual cell, e.g. cells with values less than a specified number could be coloured red.
mapType="logic" maps values matching specified logical criteria to specific "to" values. The logical criteria can be any of the following forms (the first matching mapping is used):
(1) a specific value, e.g. 12.
(2) a specific value equality condition, e.g. "v==12", where v represents the cell value.
(3) a value range expression using the following abbreviated form: "value1<=v<value2", e.g. "10<=v<15". Only "<" or "<=" can be used in these value range expressions.
(4) a standard R logical expression, e.g. "10<=v && v<15".
Basic R functions that test the value can also be used, e.g. is.na(v).
See the "Styling" and Finding and Formatting" vignettes for more information and many examples.

Returns: No return value.

Method `generateCellStructure()`**:** Generate the cells that will form the body of the pivot table.

Usage:

```
PivotTable$generateCellStructure()
```

Details: This method rarely needs to be called explicitly, since other methods will invoke it if needed.

Returns: No return value.

Method `resetCells()`**:** Clear the cells of the pivot table.

Usage:

```
PivotTable$resetCells()
```

Details: The cells are reset automatically when structural changes are made to the pivot table, so this method rarely needs to be called explicitly.

Returns: No return value.

Method `evaluateCells()`**:** Calculate the cell values in the body of the pivot table.

Usage:

```
PivotTable$evaluateCells()
```

Details: This method rarely needs to be called explicitly, since other methods will invoke it if needed.

Returns: No return value.

Method `evaluatePivot()`**:** Calculate the cell values in the body of the pivot table.

Usage:

```
PivotTable$evaluatePivot()
```

Details: This generally only needs to be called explicitly if specific pivot cells need to be further processed (e.g. formatted) before the pivot table is rendered.
This method is a wrapper for calling 'normaliseColumnGroups()', 'normaliseRowGroups()', 'generateCellStructure()' and 'evaluateCells()' in sequence.

Returns: No return value.

Method `findRowDataGroups()`**:** Find row data groups that match specified criteria.

Usage:

```
PivotTable$findRowDataGroups(
  matchMode = "simple",
  variableNames = NULL,
  variableValues = NULL,
  totals = "include",
  calculationNames = NULL,
  atLevels = NULL,
  minChildCount = NULL,
  maxChildCount = NULL,
  emptyGroups = "exclude",
  outlineGroups = "exclude",
  outlineLinkedGroupExists = NULL,
  includeDescendantGroups = FALSE,
  rowNumbers = NULL,
  cells = NULL
)
```

Arguments:

`matchMode` Either "simple" (default) or "combinations".
"simple" is used when matching only one variable-value, multiple variable-value combinations are effectively logical "OR".
"combinations" is used when matching for combinations of variable values, multiple variable-value combinations are effectively logical "AND". A child group is viewed as having the variable-value filters of itself and it's parent/ancestors, e.g.
'list("TrainCategory"="Express Passenger", "PowerType"="DMU")', would return the "DMU" data group underneath "Express Passenger".
See the "Finding and Formatting" vignette for graphical examples.

`variableNames` A character vector specifying the name/names of the variables to find. This is useful generally only in pivot tables with irregular layouts, since in regular pivot tables every cell is related to every variable.

`variableValues` A list specifying the variable names and values to find, e.g. 'variableValues=list("PowerType"=c("DMU", "HST"))'.
Specify "**" as the variable value to match totals for the specified variable.
Specify "!*" as the variable value to match non-totals for the specified variable.
NB: The totals/non-totals criteria above won't work when visual totals are used.

`totals` A word that specifies how totals are matched (overrides the finer settings above) - must be one of "include" (default), "exclude" or "only".

`calculationNames` A character vector specifying the name/names of the calculations to find.

`atLevels` An integer vector constraining the levels in the hierarchy to search.

`minChildCount` Match only data groups with this minimum number of children.

`maxChildCount` Match only data groups with this maximum number of children.

`emptyGroups` A word that specifies how empty groups are matched - must be one of "include", "exclude" (default) or "only".

`outlineGroups` A word that specifies how outline cells are matched - must be one of "include", "exclude" (default) or "only".

`outlineLinkedGroupExists` 'TRUE' to match only groups where the related outline child group still exists. 'FALSE' to match only groups where the related outline child group no longer exists.

`includeDescendantGroups` Default 'FALSE'. Specify true to also return all descendants of data groups that match the specified criteria.

`rowNumbers` An integer vector specifying row numbers that constrains the data groups to be found.

`cells` A 'PivotCell' object or a list of 'PivotCell' objects to specify one or more cells that must intersect the data groups.

Returns: A list of data groups matching the specified criteria.

Method `findColumnDataGroups()`**:** Find column data groups that match specified criteria.

Usage:

```
PivotTable$findColumnDataGroups(
  matchMode = "simple",
  variableNames = NULL,
  variableValues = NULL,
  totals = "include",
```

```
  calculationNames = NULL,
  atLevels = NULL,
  minChildCount = NULL,
  maxChildCount = NULL,
  emptyGroups = "exclude",
  includeDescendantGroups = FALSE,
  columnNumbers = NULL,
  cells = NULL
)
```

Arguments:

`matchMode` Either "simple" (default) or "combinations".
"simple" is used when matching only one variable-value - multiple variable-value combinations are effectively logical "OR".
"combinations" is used when matching for combinations of variable values - multiple variable-value combinations are effectively logical "AND". A child group is viewed as having the variable-value filters of itself and it's parent/ancestors, e.g. 'list("TrainCategory"="Express Passenger", "PowerType"="DMU")', would return the "DMU" data group underneath "Express Passenger".
See the "Finding and Formatting" vignette for graphical examples.

`variableNames` A character vector specifying the name/names of the variables to find. This is useful generally only in pivot tables with irregular layouts, since in regular pivot tables every cell is related to every variable.

`variableValues` A list specifying the variable names and values to find, e.g. 'variableValues=list("PowerType"=c("DMU", "HST"))'.
Specify "**" as the variable value to match totals for the specified variable.
Specify "!*" as the variable value to match non-totals for the specified variable.
NB: The totals/non-totals criteria above won't work when visual totals are used.

`totals` A word that specifies how totals are matched (overrides the finer settings above) - must be one of "include" (default), "exclude" or "only".

`calculationNames` A character vector specifying the name/names of the calculations to find.

`atLevels` An integer vector constraining the levels in the hierarchy to search.

`minChildCount` Match only data groups with this minimum number of children.

`maxChildCount` Match only data groups with this maximum number of children.

`emptyGroups` A word that specifies how empty groups are matched - must be one of "include", "exclude" (default) or "only".

`includeDescendantGroups` Default 'FALSE'. Specify true to also return all descendants of data groups that match the specified criteria.

`columnNumbers` An integer vector specifying column numbers that constrains the data groups to be found.

`cells` A 'PivotCell' object or a list of 'PivotCell' objects to specify one or more cells that must intersect the data groups.

Returns: A list of data groups matching the specified criteria.

Method `getEmptyRows()`**:** Retrieve row numbers for rows where all cells are empty.

Usage:

```
PivotTable$getEmptyRows(
  NAasEmpty = TRUE,
  zeroAsEmpty = FALSE,
  zeroTolerance = 1e-06,
  includeOutlineRows = FALSE
)
```

Arguments:

`NAasEmpty` 'TRUE' (default) specifies that 'NA' is treated as empty.

`zeroAsEmpty` 'TRUE' specifies that zero is treated as empty, default 'FALSE'.

`zeroTolerance` The tolerance for zero comparisons, default 0.000001.

`includeOutlineRows` 'TRUE' to also examine outline rows, default 'FALSE'.

Details: 'NULL' cell values are always regarded as empty.

Returns: An integer vector of row numbers.

Method `getEmptyColumns()`**:** Retrieve column numbers for columns where all cells are empty.

Usage:

```
PivotTable$getEmptyColumns(
  NAasEmpty = TRUE,
  zeroAsEmpty = FALSE,
  zeroTolerance = 1e-06
)
```

Arguments:

`NAasEmpty` 'TRUE' (default) specifies that 'NA' is treated as empty.

`zeroAsEmpty` 'TRUE' specifies that zero is treated as empty, default 'FALSE'.

`zeroTolerance` The tolerance for zero comparisons, default 0.000001.

Details: 'NULL' cell values are always regarded as empty.

Returns: An integer vector of column numbers.

Method `getCell()`**:** Get the cell at the specified row and column coordinates in the pivot table.

Usage:

```
PivotTable$getCell(r = NULL, c = NULL)
```

Arguments:

`r` Row number of the cell to retrieve.

`c` Column number of the cell to retrieve.

Details: The row and column numbers refer only to the cells in the body of the pivot table, i.e. row and column headings are excluded, e.g. row 1 is the first row of cells underneath the column headings.

Returns: A 'PivotCell' object representing the cell.

Method `getCells()`**:** Retrieve cells by a combination of row and/or column numbers. See the "Finding and Formatting" vignette for graphical examples.

Usage:

```
PivotTable$getCells(
  specifyCellsAsList = TRUE,
  rowNumbers = NULL,
  columnNumbers = NULL,
  cellCoordinates = NULL,
  excludeEmptyCells = FALSE,
  groups = NULL,
  rowGroups = NULL,
  columnGroups = NULL,
  matchMode = "simple"
)
```

Arguments:

`specifyCellsAsList` Specify how cells are retrieved. Default 'TRUE'. More information is provided in the details section.

`rowNumbers` A vector of row numbers that specify the rows or cells to retrieve.

`columnNumbers` A vector of column numbers that specify the columns or cells to retrieve.

`cellCoordinates` A list of two-element vectors that specify the coordinates of cells to retrieve. Ignored when 'specifyCellsAsList=FALSE'.

`excludeEmptyCells` Default 'FALSE'. Specify 'TRUE' to exclude empty cells.

`groups` A 'PivotDataGroup' object or a list of 'PivotDataGroup' objects on either the rows or columns axes. The cells to be retrieved must be related to at least one of these groups.

`rowGroups` A 'PivotDataGroup' object or a list of 'PivotDataGroup' objects on the rows axis. The cells to be retrieved must be related to at least one of these row groups. If both 'rowGroups' and 'columnGroups' are specified, then the cells to be retrieved must be related to at least one of the specified row groups and one of the specified column groups.

`columnGroups` A 'PivotDataGroup' object or a list of 'PivotDataGroup' objects on the columns axis. The cells to be retrieved must be related to at least one of these column groups. If both 'rowGroups' and 'columnGroups' are specified, then the cells to be retrieved must be related to at least one of the specified row groups and one of the specified column groups.

`matchMode` Either "simple" (default) or "combinations":
"simple" specifies that row and column arguments are considered separately (logical OR), e.g. rowNumbers=1 and columnNumbers=2 will match all cells in row 1 and all cells in column 2.
"combinations" specifies that row and column arguments are considered together (logical AND), e.g. rowNumbers=1 and columnNumbers=2 will match only the cell single at location (1, 2).
Arguments 'rowNumbers', 'columnNumbers', 'rowGroups' and 'columnGroups' are affected by the match mode. All other arguments are not.

Details: When 'specifyCellsAsList=TRUE' (the default):
Get one or more rows by specifying the row numbers as a vector as the rowNumbers argument and leaving the columnNumbers argument set to the default value of 'NULL', or
Get one or more columns by specifying the column numbers as a vector as the columnNumbers argument and leaving the rowNumbers argument set to the default value of 'NULL', or
Get one or more individual cells by specifying the cellCoordinates argument as a list of vectors of length 2, where each element in the list is the row and column number of one cell,
e.g. 'list(c(1, 2), c(3, 4))' specifies two cells, the first located at row 1, column 2 and the second

located at row 3, column 4.
When ‘specifyCellsAsList=FALSE‘:
Get one or more rows by specifying the row numbers as a vector as the rowNumbers argument and leaving the columnNumbers argument set to the default value of ‘NULL‘, or
Get one or more columns by specifying the column numbers as a vector as the columnNumbers argument and leaving the rowNumbers argument set to the default value of ‘NULL‘, or
Get one or more cells by specifying the row and column numbers as vectors for the rowNumbers and columnNumbers arguments, or
a mixture of the above, where for entire rows/columns the element in the other vector is set to ‘NA‘, e.g. to retrieve whole rows, specify the row numbers as the rowNumbers but set the corresponding elements in the columnNumbers vector to ‘NA‘.

Returns: A list of ‘PivotCell‘ objects.

Method `findCells()`**:** Find cells matching specified criteria. See the "Finding and Formatting" vignette for graphical examples.

Usage:

```
PivotTable$findCells(
  variableNames = NULL,
  variableValues = NULL,
  totals = "include",
  calculationNames = NULL,
  minValue = NULL,
  maxValue = NULL,
  exactValues = NULL,
  valueRanges = NULL,
  includeNull = TRUE,
  includeNA = TRUE,
  emptyCells = "include",
  outlineCells = "exclude",
  rowNumbers = NULL,
  columnNumbers = NULL,
  cellCoordinates = NULL,
  groups = NULL,
  rowGroups = NULL,
  columnGroups = NULL,
  rowColumnMatchMode = "simple",
  cells = NULL,
  lowN = NULL,
  highN = NULL
)
```

Arguments:

`variableNames` A character vector specifying the name/names of the variables to find. This is useful generally only in pivot tables with irregular layouts, since in regular pivot tables every cell is related to every variable.

`variableValues` A list specifying the variable names and values to find, e.g. ‘variableValues=list("PowerType"=c("DMU", "HST"))‘.
Specify "**" as the variable value to match totals for the specified variable.

Specify "!*" as the variable value to match non-totals for the specified variable.
NB: The totals/non-totals criteria above won't work when visual totals are used.

`totals` A word that specifies how totals are matched (overrides the finer settings above) - must be one of "include" (default), "exclude" or "only".

`calculationNames` A character vector specifying the name/names of the calculations to find.

`minValue` A numerical value specifying a minimum value threshold.

`maxValue` A numerical value specifying a maximum value threshold.

`exactValues` A vector or list specifying a set of allowed values.

`valueRanges` A vector specifying one or more value range expressions which the cell values must match. If multiple value range expressions are specified, then the cell value must match any of one the specified expressions. See details.

`includeNull` specify TRUE to include 'NULL' in the matched cells, FALSE to exclude 'NULL' values.

`includeNA` specify TRUE to include 'NA' in the matched cells, FALSE to exclude 'NA' values.

`emptyCells` A word that specifies how empty cells are matched - must be one of "include" (default), "exclude" or "only".

`outlineCells` A word that specifies how outline cells are matched - must be one of "include", "exclude" (default) or "only".

`rowNumbers` A vector of row numbers that specify the rows or cells to constrain the search.

`columnNumbers` A vector of column numbers that specify the columns or cells to constrain the search.

`cellCoordinates` A list of two-element vectors that specify the coordinates of cells to constrain the search.

`groups` A 'PivotDataGroup' object or a list of 'PivotDataGroup' objects on either the rows or columns axes. The cells to be searched must be related to at least one of these groups.

`rowGroups` A 'PivotDataGroup' object or a list of 'PivotDataGroup' objects on the rows axis. The cells to be searched must be related to at least one of these row groups. If both 'rowGroups' and 'columnGroups' are specified, then the cells to be searched must be related to at least one of the specified row groups and one of the specified column groups.

`columnGroups` A 'PivotDataGroup' object or a list of 'PivotDataGroup' objects on the columns axis. The cells to be searched must be related to at least one of these column groups. If both 'rowGroups' and 'columnGroups' are specified, then the cells to be searched must be related to at least one of the specified row groups and one of the specified column groups.

`rowColumnMatchMode` Either "simple" (default) or "combinations":
"simple" specifies that row and column arguments are considered separately (logical OR), e.g. rowNumbers=1 and columnNumbers=2 will match all cells in row 1 and all cells in column 2.
"combinations" specifies that row and column arguments are considered together (logical AND), e.g. rowNumbers=1 and columnNumbers=2 will match only the cell single at location (1, 2).
Arguments 'rowNumbers', 'columnNumbers', 'rowGroups' and 'columnGroups' are affected by the match mode. All other arguments are not.

`cells` A 'PivotCell' object or a list of 'PivotCell' objects to constrain the scope of the search.

`lowN` Find the first N cells (ascending order, lowest values first).

`highN` Find the last N cells (descending order, highest values first).

Details: The valueRanges parameter can be any of the following forms:
(1) a specific value, e.g. 12.
(2) a specific value equality condition, e.g. "v==12", where v represents the cell value.
(3) a value range expression using the following abbreviated form: "value1<=v<value2", e.g. "10<=v<15". Only "<" or "<=" can be used in these value range expressions.
(4) a standard R logical expression, e.g. "10<=v && v<15".
Basic R functions that test the value can also be used, e.g. is.na(v).

Returns: A list of 'PivotCell' objects.

Method `findGroupColumnNumbers()`**:** Find the column numbers associated with a specific data group or groups.

Usage:

```
PivotTable$findGroupColumnNumbers(group = NULL, collapse = FALSE)
```

Arguments:

`group` A 'PivotDataGroup' in the column data groups (i.e. a column heading) or a list of column data groups.

`collapse` A logical value specifying whether the return value should be simplified. See details.

Details: If 'group' is a list: If 'collapse' is 'FALSE', then a list of vectors is returned, if 'collapse' is 'TRUE', then a single combined vector is returned.

Returns: Either a vector of column numbers related to the single specified group or a list of vectors containing column numbers related to the specified groups.

Method `findGroupRowNumbers()`**:** Find the row numbers associated with a specific data group or groups.

Usage:

```
PivotTable$findGroupRowNumbers(group = NULL, collapse = FALSE)
```

Arguments:

`group` A 'PivotDataGroup' in the row data groups (i.e. a row heading) or a list of row data groups.

`collapse` A logical value specifying whether the return value should be simplified. See details.

Details: If 'group' is a list: If 'collapse' is 'FALSE', then a list of vectors is returned, if 'collapse' is 'TRUE', then a single combined vector is returned.

Returns: Either a vector of row numbers related to the single specified group or a list of vectors containing row numbers related to the specified groups.

Method `removeColumn()`**:** Remove a column from the pivot table.

Usage:

```
PivotTable$removeColumn(c = NULL)
```

Arguments:

`c` The column number. The first column is column 1, excluding the column(s) associated with row-headings.

Details: This method removes both the related column group and cells.

Returns: No return value.

Method `removeColumns()`**:** Remove multiple column from the pivot table.

Usage:

```
PivotTable$removeColumns(columnNumbers = NULL)
```

Arguments:

`columnNumbers` The column numbers. The first column is column 1, excluding the column(s) associated with row-headings.

Details: This method removes both the related column groups and cells.

Returns: No return value.

Method `removeEmptyColumns()`**:** Remove columns where all cells are empty.

Usage:

```
PivotTable$removeEmptyColumns(
  NAasEmpty = TRUE,
  zeroAsEmpty = FALSE,
  zeroTolerance = 1e-06
)
```

Arguments:

`NAasEmpty` 'TRUE' (default) specifies that 'NA' is treated as empty.

`zeroAsEmpty` 'TRUE' specifies that zero is treated as empty, default 'FALSE'.

`zeroTolerance` The tolerance for zero comparisons, default 0.000001.

Details: 'NULL' cell values are always regarded as empty.

Returns: No return value.

Method `removeRow()`**:** Remove a row from the pivot table.

Usage:

```
PivotTable$removeRow(r = NULL)
```

Arguments:

`r` The row number. The first row is row 1, excluding the row(s) associated with column-headings.

Details: This method removes both the related row group and cells.

Returns: No return value.

Method `removeRows()`**:** Remove multiple rows from the pivot table.

Usage:

```
PivotTable$removeRows(rowNumbers = NULL)
```

Arguments:

`rowNumbers` The row numbers. The first row is row 1, excluding the rows(s) associated with column-headings.

Details: This method removes both the related row groups and cells.

Returns: No return value.

Method `removeEmptyRows()`**:** Remove rows where all cells are empty.

Usage:

```
PivotTable$removeEmptyRows(
  NAasEmpty = TRUE,
  zeroAsEmpty = FALSE,
  zeroTolerance = 1e-06,
  includeOutlineRows = FALSE
)
```

Arguments:

`NAasEmpty` 'TRUE' (default) specifies that 'NA' is treated as empty.

`zeroAsEmpty` 'TRUE' specifies that zero is treated as empty, default 'FALSE'.

`zeroTolerance` The tolerance for zero comparisons, default 0.000001.

`includeOutlineRows` 'TRUE' to also remove empty outline rows, default 'FALSE'.

Details: 'NULL' cell values are always regarded as empty.

Returns: No return value.

Method `print()`**:** Outputs a plain text representation of the pivot table to the console or returns a character representation of the pivot table.

Usage:

```
PivotTable$print(asCharacter = FALSE, showRowGroupHeaders = FALSE)
```

Arguments:

`asCharacter` 'FALSE'(default) outputs to the console, specify 'TRUE' to instead return a character value (does not output to console).

`showRowGroupHeaders` 'TRUE' to include the row group headers in the output, default 'FALSE'.

Returns: Plain text representation of the pivot table.

Method `asMatrix()`**:** Convert the pivot table to a matrix, where the data group headings are included in the body of the matrix. This method tends to produce a character matrix.

Usage:

```
PivotTable$asMatrix(
  includeHeaders = TRUE,
  repeatHeaders = FALSE,
  rawValue = FALSE,
  showRowGroupHeaders = FALSE
)
```

Arguments:

`includeHeaders` 'TRUE' (default) to include the headings in the body of the matrix. Specifying 'FALSE' omits the headings.

`repeatHeaders` 'FALSE' (default) only outputs the first occurrence of each header. Specify 'TRUE' to repeat the headings.

`rawValue` 'FALSE' (default) outputs the formatted (character) values. Specify 'TRUE' to output the raw cell values.

`showRowGroupHeaders` 'TRUE' to include the row group headers in the matrix, default 'FALSE'.

Details: The newer 'asDataMatrix()' tends to produce more a useful matrix. See the "Outputs" vignette for a comparison of outputs.

Returns: A matrix.

Method `asDataMatrix()`**:** Convert the pivot table to a matrix, where the data group headings are included as row/column headings in the matrix. This method tends to produce a numeric matrix.

Usage:

```
PivotTable$asDataMatrix(
  includeHeaders = TRUE,
  rawValue = TRUE,
  separator = " "
)
```

Arguments:

`includeHeaders` 'TRUE' (default) to include the headings in the matrix. Specifying 'FALSE' omits the headings.

`rawValue` 'TRUE' (default) outputs the raw cell values. Specify 'FALSE' to output the formatted (character) values.

`separator` Specifies the character value used to concatenate data group captions where multiple levels exist in the data group hierarchy.

Details: Where there are multiple levels in a data group hierarchy, the captions are concatenated to form the row/column headings in the matrix. See the "Outputs" vignette for a comparison of outputs.

Returns: A matrix.

Method `asDataFrame()`**:** Convert the pivot table to a data frame, combining multiple levels of headings with the specified separator and/or exporting the row groups as columns in the data frame.

Usage:

```
PivotTable$asDataFrame(
  separator = " ",
  stringsAsFactors = NULL,
  forceNumeric = FALSE,
  rowGroupsAsColumns = FALSE
)
```

Arguments:

`separator` Specifies the character value used to concatenate data group captions where multiple levels exist in the data group hierarchy.

`stringsAsFactors` Specify 'TRUE' to convert strings to factors, default is 'default.stringsAsFactors()' for R < 4.1.0 and 'FALSE' for R >= 4.1.0.

`forceNumeric` Specify 'TRUE' to force the conversion of cell values to a numeric value, default 'FALSE'.

`rowGroupsAsColumns` Specify 'TRUE' to include the row groups as additional columns in the data frame. Default 'FALSE'.

Details: See the "Outputs" vignette for more details and examples

Returns: A data frame.

Method `asTidyDataFrame()`**:** Convert the pivot table to tidy data frame, where each cell in the body of the pivot table becomes one row in the data frame.

Usage:

```
PivotTable$asTidyDataFrame(
  includeGroupCaptions = TRUE,
  includeGroupValues = TRUE,
  separator = " ",
  stringsAsFactors = NULL,
  excludeEmptyCells = TRUE
)
```

Arguments:

`includeGroupCaptions` 'TRUE' (default) to include the data group captions as columns in the data frame.

`includeGroupValues` 'TRUE' (default) to include the data group values as columns in the data frame.

`separator` Specifies the character value used to concatenate filter values where multiple values exist in a filter.

`stringsAsFactors` Specify 'TRUE' to convert strings to factors, default is 'default.stringsAsFactors()' for R < 4.1.0 and 'FALSE' for R >= 4.1.0.

`excludeEmptyCells` Specify 'FALSE' to also include rows for empty cells in the data frame, default 'TRUE'.

Details: See the "Outputs" vignette for more details and examples

Returns: A data frame.

Method `getMerges()`**:** Generate a list of the merged cell information arising from the data group hierarchies. This is an internal method used to support rendering the pivot table.

Usage:

```
PivotTable$getMerges(axis = NULL)
```

Arguments:

`axis` Either "row" or "column".

Returns: A list containing details of the merged cells.

Method `asBasicTable()`**:** Convert the pivot table to a 'basictabler' table (from the 'basictabler' R package) which allows further custom manipulation of the pivot table.

Usage:

```
PivotTable$asBasicTable(
  exportOptions = NULL,
  compatibility = NULL,
  showRowGroupHeaders = FALSE
)
```

Arguments:

`exportOptions` A list of additional export options - see the "A1. Appendix" for details.

`compatibility` A list containing compatibility options to force legacy behaviours in the resulting 'basictabler' table.

`showRowGroupHeaders` 'TRUE' to include the row group headers in the matrix, default 'FALSE'.

Details: See the "Outputs" vignette for more details and examples

Returns: A 'basictabler' table.

Method `getCss()`**:** Get the CSS declarations for the pivot table.

Usage:

```
PivotTable$getCss(styleNamePrefix = NULL)
```

Arguments:

`styleNamePrefix` A character variable specifying a prefix for all named CSS styles, to avoid style name collisions where multiple pivot tables exist.

Details: See the "Outputs" vignette for more details and examples.

Returns: A character value containing the CSS style declaration.

Method `getHtml()`**:** Generate a HTML representation of the pivot table, optionally including additional detail for debugging purposes.

Usage:

```
PivotTable$getHtml(
  styleNamePrefix = NULL,
  includeHeaderValues = FALSE,
  includeRCFilters = FALSE,
  includeCalculationFilters = FALSE,
  includeWorkingData = FALSE,
  includeEvaluationFilters = FALSE,
  includeCalculationNames = FALSE,
  includeRawValue = FALSE,
  includeTotalInfo = FALSE,
  exportOptions = NULL,
  showRowGroupHeaders = FALSE
)
```

Arguments:

`styleNamePrefix` A character variable specifying a prefix for all named CSS styles, to avoid style name collisions where multiple pivot tables exist.

`includeHeaderValues` Default 'FALSE', specify 'TRUE' to render this debug information.

`includeRCFilters` Default 'FALSE', specify 'TRUE' to render this debug information.

`includeCalculationFilters` Default 'FALSE', specify 'TRUE' to render this debug information.

`includeWorkingData` Default 'FALSE', specify 'TRUE' to render this debug information.

`includeEvaluationFilters` Default 'FALSE', specify 'TRUE' to render this debug information.

`includeCalculationNames` Default 'FALSE', specify 'TRUE' to render this debug information.

`includeRawValue` Default 'FALSE', specify 'TRUE' to render this debug information.

includeTotalInfo Default 'FALSE', specify 'TRUE' to render this debug information.

exportOptions A list of additional export options - see the "A1. Appendix" for details.

showRowGroupHeaders Default 'FALSE', specify 'TRUE' to render the row group headings. See the "Data Groups" vignette for details.

Details: See the "Outputs" vignette for more details and examples.

Returns: A list containing HTML tags from the 'htmltools' package. Convert this to a character variable using 'as.character()'.

Method `saveHtml()`**:** Save a HTML representation of the pivot table to file, optionally including additional detail for debugging purposes.

Usage:

```
PivotTable$saveHtml(
  filePath = NULL,
  fullPageHTML = TRUE,
  styleNamePrefix = NULL,
  includeHeaderValues = FALSE,
  includeRCFilters = FALSE,
  includeCalculationFilters = FALSE,
  includeWorkingData = FALSE,
  includeEvaluationFilters = FALSE,
  includeCalculationNames = FALSE,
  includeRawValue = FALSE,
  includeTotalInfo = FALSE,
  exportOptions = NULL,
  showRowGroupHeaders = FALSE
)
```

Arguments:

filePath The file to save the HTML to.

fullPageHTML 'TRUE' (default) includes basic HTML around the pivot table HTML so that the result file is a valid HTML file.

styleNamePrefix A character variable specifying a prefix for all named CSS styles, to avoid style name collisions where multiple pivot tables exist.

includeHeaderValues Default 'FALSE', specify 'TRUE' to render this debug information.

includeRCFilters Default 'FALSE', specify 'TRUE' to render this debug information.

includeCalculationFilters Default 'FALSE', specify 'TRUE' to render this debug information.

includeWorkingData Default 'FALSE', specify 'TRUE' to render this debug information.

includeEvaluationFilters Default 'FALSE', specify 'TRUE' to render this debug information.

includeCalculationNames Default 'FALSE', specify 'TRUE' to render this debug information.

includeRawValue Default 'FALSE', specify 'TRUE' to render this debug information.

includeTotalInfo Default 'FALSE', specify 'TRUE' to render this debug information.

exportOptions A list of additional export options - see the "A1. Appendix" for details.

showRowGroupHeaders Default 'FALSE', specify 'TRUE' to render the row group headings. See the "Data Groups" vignette for details.

Details: See the "Outputs" vignette for more details and examples.

Returns: No return value.

Method `renderPivot()`**:** Render a HTML representation of the pivot table as an HTML widget, optionally including additional detail for debugging purposes.

Usage:

```
PivotTable$renderPivot(
  width = NULL,
  height = NULL,
  styleNamePrefix = NULL,
  includeHeaderValues = FALSE,
  includeRCFilters = FALSE,
  includeCalculationFilters = FALSE,
  includeWorkingData = FALSE,
  includeEvaluationFilters = FALSE,
  includeCalculationNames = FALSE,
  includeRawValue = FALSE,
  includeTotalInfo = FALSE,
  exportOptions = NULL,
  showRowGroupHeaders = FALSE
)
```

Arguments:

width The width of the widget.

height The height of the widget.

styleNamePrefix A character variable specifying a prefix for all named CSS styles, to avoid style name collisions where multiple pivot tables exist.

includeHeaderValues Default 'FALSE', specify 'TRUE' to render this debug information.

includeRCFilters Default 'FALSE', specify 'TRUE' to render this debug information.

includeCalculationFilters Default 'FALSE', specify 'TRUE' to render this debug information.

includeWorkingData Default 'FALSE', specify 'TRUE' to render this debug information.

includeEvaluationFilters Default 'FALSE', specify 'TRUE' to render this debug information.

includeCalculationNames Default 'FALSE', specify 'TRUE' to render this debug information.

includeRawValue Default 'FALSE', specify 'TRUE' to render this debug information.

includeTotalInfo Default 'FALSE', specify 'TRUE' to render this debug information.

exportOptions A list of additional export options - see the "A1. Appendix" for details.

showRowGroupHeaders Default 'FALSE', specify 'TRUE' to render the row group headings. See the "Data Groups" vignette for details.

Details: See the "Outputs" vignette for more details and examples.

Returns: A HTML widget from the 'htmlwidgets' package.

Method `getLatex()`**:** Generate a Latex representation of the pivot table.

Usage:

```
PivotTable$getLatex(
  caption = NULL,
  label = NULL,
  fromRow = NULL,
  toRow = NULL,
  fromColumn = NULL,
  toColumn = NULL,
  boldHeadings = FALSE,
  italicHeadings = FALSE,
  exportOptions = NULL
)
```

Arguments:

`caption` The caption to appear above the table.

`label` The label to use when referring to the table elsewhere in the document

`fromRow` The row number to render from.

`toRow` The row number to render to.

`fromColumn` The column number to render from.

`toColumn` The column number to render to.

`boldHeadings` Default 'FALSE', specify 'TRUE' to render headings in bold.

`italicHeadings` Default 'FALSE', specify 'TRUE' to render headings in italic.

`exportOptions` A list of additional export options - see the "A1. Appendix" for details.

Returns: A character variable containing the Latex representation of the pivot table.

Method `writeToExcelWorksheet()`**:** Write the pivot table into the specified workbook and worksheet at the specified row-column location.

Usage:

```
PivotTable$writeToExcelWorksheet(
  wb = NULL,
  wsName = NULL,
  topRowNumber = NULL,
  leftMostColumnNumber = NULL,
  outputHeadingsAs = "formattedValueAsText",
  outputValuesAs = "rawValue",
  applyStyles = TRUE,
  mapStylesFromCSS = TRUE,
  exportOptions = NULL,
  showRowGroupHeaders = FALSE
)
```

Arguments:

`wb` A 'Workbook' object representing the Excel file being written to.

`wsName` A character value specifying the name of the worksheet to write to.

`topRowNumber` An integer value specifying the row number in the Excel worksheet to write the pivot table.

`leftMostColumnNumber` An integer value specifying the column number in the Excel worksheet to write the pivot table.

`outputHeadingsAs` Must be one of "rawValue", "formattedValueAsText" (default) or "formattedValueAsNumber" to specify how data groups are written into the Excel sheet.

`outputValuesAs` Must be one of "rawValue" (default), "formattedValueAsText" or "formattedValueAsNumber" to specify how cell values are written into the Excel sheet.

`applyStyles` Default 'TRUE' to write styling information to the cell.

`mapStylesFromCSS` Default 'TRUE' to automatically convert CSS style declarations to their Excel equivalents.

`exportOptions` A list of additional export options - see the "A1. Appendix" for details.

`showRowGroupHeaders` Default 'FALSE', specify 'TRUE' to write row group headers.

Returns: No return value.

Method `trace()`**:** Capture a call for tracing purposes. This is an internal method.

Usage:

```
PivotTable$trace(methodName, desc, detailList = NULL)
```

Arguments:

`methodName` The name of the method being invoked.

`desc` Short description of method call.

`detailList` A list containing detail such as parameter values.

Returns: No return value.

Method `showBatchInfo()`**:** Output batch information to the console.

Usage:

```
PivotTable$showBatchInfo()
```

Returns: No return value.

Method `asList()`**:** Return the contents of the pivot table as a list for debugging.

Usage:

```
PivotTable$asList()
```

Returns: A list of various object properties.

Method `asJSON()`**:** Return the contents of the pivot table as JSON for debugging.

Usage:

```
PivotTable$asJSON()
```

Returns: A JSON representation of various object properties.

Method `viewJSON()`**:** Use the 'listviewer' package to view the pivot table as JSON for debugging.

Usage:

```
PivotTable$viewJSON()
```

Returns: No return value.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PivotTable$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# The package vignettes include extensive examples of working with the
# PivotTable class.
library(pivottabler)
pt <- PivotTable$new()
pt$addData(bhmtrains)
pt$addColumnDataGroups("TrainCategory")
pt$addRowDataGroups("TOC")
pt$defineCalculation(calculationName="TotalTrains",
summariseExpression="n()")
pt$renderPivot()
```

`pivottabler` *Render a pivot table as a HTML widget.*

Description

The `pivottabler` function is primarily intended for use with Shiny web applications.

Usage

```
pivottabler(
  pt,
  width = NULL,
  height = NULL,
  styleNamePrefix = NULL,
  includeRCFilters = FALSE,
  includeCalculationFilters = FALSE,
  includeWorkingData = FALSE,
  includeEvaluationFilters = FALSE,
  includeCalculationNames = FALSE,
  includeRawValue = FALSE
)
```

Arguments

`pt`	The pivot table to render.
`width`	The target width.
`height`	The target height.

styleNamePrefix
: A text prefix to be prepended to the CSS declarations (to ensure uniqueness).

includeRCFilters
: Show/hide filter detail for debugging.

includeCalculationFilters
: Show/hide filter detail for debugging.

includeWorkingData
: Show/hide working data detail for debugging.

includeEvaluationFilters
: Show/hide filter detail for debugging.

includeCalculationNames
: Show/hide filter detail for debugging.

includeRawValue
: Show/hide filter detail for debugging.

Value

A HTML widget.

Examples

```
# See the Shiny vignette in this package for examples.
```

---

pivottablerOutput *Standard function for Shiny scaffolding.*

Description

Standard function for Shiny scaffolding.

Usage

```
pivottablerOutput(outputId, width = "100%", height = "100%")
```

Arguments

outputId	The id of the html element that will contain the htmlwidget.
width	The target width of the htmlwidget.
height	The target height of the htmlwidget.

processIdentifier *Handle an identifier that may be illegal (e.g. containing spaces).*

Description

`processIdentifier` is a utility function that wraps an illegal identifier in backticks.

Usage

```
processIdentifier(identifier)
```

Arguments

`identifier`	The identifier that may be illegal.

Value

The identifier wrapped in backticks (if illegal) or unchanged.

processIdentifiers *Handle identifiers that may be illegal (e.g. containing spaces).*

Description

`processIdentifiers` is a utility function that wraps illegal identifiers in backticks.

Usage

```
processIdentifiers(identifiers)
```

Arguments

`identifiers`	The identifiers that may be illegal.

Value

The identifiers wrapped in backticks (if illegal) or unchanged.

pvtperfresults *Performance Comparison Results*

Description

A reference dataset containing the full results of an example performance comparison for different pivot table test cases.

Usage

```
pvtperfresults
```

Format

A data frame with 216 rows and 11 variables:

id A unique identifier for this test case.

evaluationMode The pivot table evaluation mode used for this test case.

rowCount The number of rows in the data frame used for this test case.

cellCount The number of cells in the pivot table used for this test case.

argumentCheckMode The pivot table argument check mode used this test case.

processingLibrary The pivot table processing library used this test case.

description A description of this test case.

completed A logical value indicating whether this test case completed.

userTime The user time for this test case.

systemTime The system time for this test case.

elapsedTime The elapsed time for this test case.

pvtperfsummary *Performance Comparison Summary*

Description

A reference dataset containing summary results of an example performance comparison for different pivot table test cases.

Usage

```
pvtperfsummary
```

Format

A data frame with 36 rows and 18 variables:

id A unique identifier for this test case.
evaluationMode The pivot table evaluation mode used for this test case.
rowCount The number of rows in the data frame used for this test case.
cellCount The number of cells in the pivot table used for this test case.
argumentCheckMode The pivot table argument check mode used this test case.
processingLibrary The pivot table processing library used this test case.
description A description of this test case.
userTimeAvg The average user time for this test case.
systemTimeAvg The average system time for this test case.
elapsedTimeAvg The average elapsed time for this test case.
userTimeMin The minimum user time for this test case.
userTimeMax The maximum user time for this test case.
systemTimeMin The minimum system time for this test case.
systemTimeMax The maximum system time for this test case.
elapsedTimeMin The minimum elapsed time for this test case.
elapsedTimeMax The maximum elapsed time for this test case.
testName A short name for this test case.
testIndex An index for this test case.

---

qhpvt *Quickly render a basic pivot table in HTML.*

Description

The `qhpvt` function renders a basic pivot table as a HTML widget with one line of R.

Usage

```
qhpvt(
  dataFrame,
  rows = NULL,
  columns = NULL,
  calculations = NULL,
  theme = NULL,
  replaceExistingStyles = FALSE,
  tableStyle = NULL,
  headingStyle = NULL,
  cellStyle = NULL,
  totalStyle = NULL,
  ...
)
```

Arguments

dataFrame	The data frame containing the data to be summarised in the pivot table.
rows	A character vector of variable names to be plotted on the rows of the pivot table, or "=" to specify the position of the calculations.
columns	A character vector of variable names to be plotted on the columns of the pivot table, or "=" to specify the position of the calculations.
calculations	One or more summary calculations to use to calculate the values of the cells in the pivot table.
theme	Either the name of a built-in theme (default, largeplain, compact or blank/none) or a list which specifies the default formatting for the table.
replaceExistingStyles	TRUE to completely replace the default styling with the specified tableStyle, headingStyle, cellStyle and/or totalStyle
tableStyle	A list of CSS style declarations that apply to the table.
headingStyle	A list of CSS style declarations that apply to the heading cells in the table.
cellStyle	A list of CSS style declarations that apply to the normal cells in the table.
totalStyle	A list of CSS style declarations that apply to the total cells in the table.
...	Additional arguments, currently format, formats, totals, styleNamePrefix, compatibility and/or argumentCheckMode.

Value

A HTML widget.

Examples

```
qhpvt(bhmtrains, "TOC", "TrainCategory", "n()")
qhpvt(bhmtrains, "TOC", "TrainCategory",
     c("Mean Speed"="mean(SchedSpeedMPH, na.rm=TRUE)",
       "Std Dev Speed"="sd(SchedSpeedMPH, na.rm=TRUE)"),
     formats=list("%.0f", "%.1f"),
     totals=list("TOC"="All TOCs",
       "TrainCategory"="All Categories"))
```

---

qlpvt *Quickly get a Latex representation of a basic pivot table.*

Description

The qlpvt function returns the Latex for a basic pivot table with one line of R.

Usage

```
qlpvt(dataFrame, rows = NULL, columns = NULL, calculations = NULL, ...)
```

Arguments

dataFrame	The data frame containing the data to be summarised in the pivot table.
rows	A character vector of variable names to be plotted on the rows of the pivot table, or "=" to specify the position of the calculations.
columns	A character vector of variable names to be plotted on the columns of the pivot table, or "=" to specify the position of the calculations.
calculations	One or more summary calculations to use to calculate the values of the cells in the pivot table.
...	Additional arguments, currently format, formats, totals, argumentCheckMode, compatibility, caption and/or label. See the Latex output vignette for a description of caption and label.

Value

Latex.

Examples

```
qlpvt(bhmtrains, "TOC", "TrainCategory", "n()")
qlpvt(bhmtrains, "TOC", "TrainCategory", "n()",
      caption="my caption", label="mylabel")
```

---

qpvt *Quickly build a basic pivot table.*

Description

The qpvt function builds a basic pivot table with one line of R.

Usage

```
qpvt(
  dataFrame,
  rows = NULL,
  columns = NULL,
  calculations = NULL,
  theme = NULL,
  replaceExistingStyles = FALSE,
  tableStyle = NULL,
  headingStyle = NULL,
  cellStyle = NULL,
  totalStyle = NULL,
  ...
)
```

Arguments

dataFrame	The data frame containing the data to be summarised in the pivot table.
rows	A character vector of variable names to be plotted on the rows of the pivot table, or "=" to specify the position of the calculations.
columns	A character vector of variable names to be plotted on the columns of the pivot table, or "=" to specify the position of the calculations.
calculations	One or more summary calculations to use to calculate the values of the cells in the pivot table.
theme	Either the name of a built-in theme (default, largeplain, compact or blank/none) or a list which specifies the default formatting for the table.
replaceExistingStyles	TRUE to completely replace the default styling with the specified tableStyle, headingStyle, cellStyle and/or totalStyle
tableStyle	A list of CSS style declarations that apply to the table.
headingStyle	A list of CSS style declarations that apply to the heading cells in the table.
cellStyle	A list of CSS style declarations that apply to the normal cells in the table.
totalStyle	A list of CSS style declarations that apply to the total cells in the table.
...	Additional arguments, currently format, formats, totals, compatibility and/or argumentCheckMode.

Value

A pivot table.

Examples

```
qpvt(bhmtrains, "TOC", "TrainCategory", "n()")
qpvt(bhmtrains, c("=", "TOC"), c("TrainCategory", "PowerType"),
     c("Number of Trains"="n()",
       "Maximum Speed"="max(SchedSpeedMPH, na.rm=TRUE)"))
```

---

renderBasicTable *Output a table into a package vignette.*

Description

renderBasicTable is utility function that renders a basic table into a package vignette. This function is primarily intended for internal use by the pivottabler package.

Usage

```
renderBasicTable(
  matrix = NULL,
  stylePrefix = NULL,
  columnNamesAsHeader = FALSE,
  rowNamesAsHeader = FALSE,
  columnAlignment = "right"
)
```

Arguments

matrix	Tabular data to render.
stylePrefix	Text prefix for CSS style declarations.
columnNamesAsHeader	Include column names in output (if FALSE, the first row from the matrix is used as the column headings).
rowNamesAsHeader	Include row names in output.
columnAlignment	A character vector specifying the horizontal alignment of each column.

Value

A basic table rendered as a HTML widget.

Examples

```
renderBasicTable(matrix(c(1:12), nrow=3))
```

---

renderPivottabler *Standard function for Shiny scaffolding.*

Description

Standard function for Shiny scaffolding.

Usage

```
renderPivottabler(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

expr	The R expression to execute and render in the Shiny web application.
env	Standard shiny argument for a render function.
quoted	Standard shiny argument for a render function.

skipExportingValue *Should the current value be skipped during export?*

Description

skipExportingValue is a utility function that returns true if the current value should be skipped when exporting.

Usage

```
skipExportingValue(rawValue, exportOptions)
```

Arguments

rawValue	The value to check.
exportOptions	A list of options controlling export behaviour.

Value

TRUE or FALSE indicating whether the current value should be skipped.

trainstations *Train Stations*

Description

A reference dataset listing the codes, names and locations of trains stations in Great Britain.

Usage

```
trainstations
```

Format

A data frame with 2568 rows and 7 variables:

CrsCode 3-letter code for the station

StationName The name of the station

OsEasting The UK Ordnance Survey Easting coordinate for the station

OsNorthing The UK Ordnance Survey Northing coordinate for the station

GridReference Grid reference for the station

Latitude Latitude of the station location

Longitude Longitude of the station location

Source

https://www.recenttraintimes.co.uk/

typeSafeIntersect *Intersect two vectors without changing their data types.*

Description

typeSafeIntersect preserves data types in a way that the base::intersect function does not, e.g. for Date values.

Usage

```
typeSafeIntersect(x, y, dedupe = FALSE)
```

Arguments

x	First set of values.
y	Second set of values.
dedupe	Specify TRUE to remove duplicate values.

Value

A vector containing the intersection of x and y.

typeSafeUnion *Union two vectors without changing their data types.*

Description

typeSafeUnion preserves data types in a way that the base::union function does not, e.g. for Date values.

Usage

```
typeSafeUnion(x, y, dedupe = FALSE)
```

Arguments

x	First set of values.
y	Second set of values.
dedupe	Specify TRUE to remove duplicate values.

Value

A vector containing the union of x and y

`typeSafeUnlist` *Unlist a list into a vector in a type-safe way where possible.*

Description

`typeSafeUnlist` tries to preserve data types in a way that the `base::unlist` function does not for Date, POSIXct and POSIXlt values.

Usage

```
typeSafeUnlist(x)
```

Arguments

`x`	A list to convert to a vector.

Details

If a list containing mixed types is specified, then `typeSafeUnlist` falls back to using `base::unlist`.

Value

A vector containing the values from x.

`vreConvertSimpleNumericRange`
Convert a simple range expression to a standard R logical expression.

Description

`vreConvertSimpleNumericRange` is a utility function that converts a simple range expression of the form "value1<=v<value2" to a standard R logical expression of the form "value1<=v && v<value2".

Usage

```
vreConvertSimpleNumericRange(vre)
```

Arguments

`vre`	The value range expression to examine.

Value

A standard R logical expression.

vreGetSingleValue *Read the value from a single-valued value range expression.*

Description

vreGetSingleValue is a utility function reads the single value from a value range expression (it assumes the specified is either numeric, a number expressed as text or an expression of the form "v=" or "v==").

Usage

```
vreGetSingleValue(vre)
```

Arguments

vre	The value range expression to examine.

Value

The value read from the expression.

vreHexToClr *Convert a colour in hex format (#RRGGBB) into a list.*

Description

vreHexToClr converts a colour in hex format (#RRGGBB) into a list of three element (r, g and b).

Usage

```
vreHexToClr(hexclr)
```

Arguments

hexclr	The colour to convert.

Value

The converted colour.

vreIsEqual — *Test if two numeric values are equal within tolerance.*

Description

vreIsEqual tests whether two values are equal within sqrt(.Machine$double.eps).

Usage

```
vreIsEqual(value1, value2)
```

Arguments

value1	The first value to compare.
value2	The second value to compare.

Value

'TRUE' if the two numbers are equal, 'FALSE' otherwise.

vreIsMatch — *Test whether a value matches a value range expression.*

Description

vreIsMatch tests a value (e.g. from a cell) matches the criteria specified in a value range expression.

Usage

```
vreIsMatch(vre, v, testOnly = FALSE)
```

Arguments

vre	The value range expression.
v	The value.
testOnly	'TRUE' if this comparison is just a test.

Value

'TRUE' if v matches the criteria specified in the value range expression, 'FALSE' otherwise.

`vreIsSimpleNumericRange` *Determine if a value range expression is a simple range expression.*

Description

`vreIsSingleValue` is a utility function that returns 'TRUE' if the specified value range expression is a simple range expression of the form "value1<=v<value2", where the logical comparisons can be < or <= only and the values must be numbers.

Usage

```
vreIsSimpleNumericRange(vre)
```

Arguments

`vre`	The value range expression to examine.

Value

'TRUE' if vre is a simple range expression, 'FALSE' otherwise.

`vreIsSingleValue` *Determine if a value range expression is a single value.*

Description

`vreIsSingleValue` is a utility function that returns 'TRUE' if the specified value range expression is either numeric, a number expressed as text or an expression of the form "v=" or "v==".

Usage

```
vreIsSingleValue(vre)
```

Arguments

`vre`	The value range expression to examine.

Value

'TRUE' if vre is a single value, 'FALSE' otherwise.

vreScale2Colours *Scale a number from a range into a colour gradient.*

Description

vreScale2Colours takes a value from a range and scales it proportionally into a colour from a colour gradient.

Usage

```
vreScale2Colours(clr1, clr2, vMin, vMax, value)
```

Arguments

clr1	The colour representing the lower value of the target range.
clr2	The colour representing the upper value of the target range.
vMin	The lower value of the source range.
vMax	The upper value of the source range.
value	The source value to rescale into the target range.

Value

The value scaled into the target colour gradient.

vreScaleNumber *Rescale a number from one range into another range.*

Description

vreScaleNumber takes a value from one range and scales it proportionally into another range.

Usage

```
vreScaleNumber(n1, n2, vMin, vMax, value, decimalPlaces = 3)
```

Arguments

n1	The lower value of the target range.
n2	The upper value of the target range.
vMin	The lower value of the source range.
vMax	The upper value of the source range.
value	The source value to rescale into the target range.
decimalPlaces	The number of decimal places to round the result to.

Value

The value rescaled into the target range.

Index