

# Package ‘muir’

May 9, 2026

**Type** Package

**Title** Exploring Data with Tree Data Structures

**Version** 0.1.0

**Description** A simple tool allowing users to easily and dynamically explore or document a data set using a tree structure.

**URL** <https://github.com/alforj/muir>

**BugReports** <https://github.com/alforj/muir/issues>

**Depends** R (>= 3.1.2)

**License** GPL (>= 2)

**LazyData** true

**Imports** DiagrammeR (>= 0.6), dplyr (>= 0.4.1), stringr (>= 0.6.2)

**Suggests** htmlwidgets (>= 0.3.2)

**Author** Justin Alford [aut, cre]

**Maintainer** Justin Alford <[justin.alford@gmail.com](mailto:justin.alford@gmail.com)>

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2015-05-02 16:46:05

## Contents

build_tree . . . . .	2
muir . . . . .	2
<b>Index</b>	<b>7</b>

---

build_tree	<i>Build dynamic tree using DiagrammeR and muir-generated data frame</i>
------------	--

---

### Description

This function allows users to easily and dynamically explore or document a dataset using a tree structure.

### Usage

```
build_tree(data, tree.dir = "LR", tree.height = NULL, tree.width = NULL)
```

### Arguments

data	A muir-generated data frame to be processed into a tree using DiagrammeR
tree.dir	Direction of tree graph. Use either "LR" for left-to-right, "RL" for right-to left, "TB" for top-to-bottom, or "BT" for bottom-to-top.
tree.height	Control tree height to zoom in/out on nodes. Defaults to NULL
tree.width	Control tree width to zoom in/out on nodes. Defaults to NULL

### Value

An object of class `htmlwidget` (via `DiagrammeR`) that will intelligently print itself into HTML in a variety of contexts including the R console, within R Markdown documents, and within Shiny output bindings.

---

muir	<i>Explore Datasets with Trees</i>
------	------------------------------------

---

### Description

This function allows users to easily and dynamically explore or document a data.frame using a tree data structure. Columns of interest in the data.frame can be provided to the function, as well as criteria for how they should be represented in discrete nodes, to generate a data tree representing those columns and filters.

### Usage

```
muir(data, node.levels, node.limit = 3, level.criteria = NULL,
      label.vals = NULL, tree.dir = "LR", show.percent = TRUE,
      num.precision = 2, show.empty.child = FALSE, tree.height = -1,
      tree.width = -1)
```

## Arguments

<code>data</code>	A <code>data.frame</code> to be explored using trees
<code>node.levels</code>	<p>A character vector of columns from <code>data</code> that will be used to construct the tree that are provided in the order that they should appear in the tree levels.</p> <p>For each column, the user can add a suffix to the column name to indicate whether to generate nodes for all distinct values of the column in the <code>data.frame</code>, a specific number of values (i.e., the "Top (n)" values), and whether or not to aggregate remaining values into a separate "Other" node, or to use user-provided filter criteria for the column as provided in the <code>level.criteria</code> parameter. This does mean that the column names cannot have a ":" and must be replaced in the <code>data.frame</code> before being passed in to <code>muir</code> as the <code>data</code> param.</p> <p>Values can be provided as "colname", "colname:*", "colname:3", "colname:+", or "colname:*+". The separator character ":" and the special characters in the suffix that follow (as outlined below) indicate which approach to take for each column.</p> <ul style="list-style-type: none"> <li>• Providing just the column name itself (e.g., "hp") will return results based on the operators and values provided in the <code>level.criteria</code> parameter for that column name. See <code>level.criteria</code> for more details.</li> <li>• Providing the column name with an ":*" suffix (e.g., "hp:*") will return a node for all distinct values for that column up to the limit imposed by the <code>node.limit</code> value. If the number of distinct values is greater than the <code>node.limit</code>, only the top "n" values (based on number of occurrences) will be returned.</li> <li>• Providing the column name with an ":n" suffix (e.g., "hp:3"), where <code>n</code> = a positive integer, will return a node for all distinct values for that column up to the limit imposed by the integer provided in <code>n</code>. If the number of distinct values is greater than the value provided in <code>n</code>, only the top "n" values (based on number of occurrences) will be returned.</li> <li>• Providing the column name ending with an ":+ " suffix (e.g., "hp:+") will return all the values provided in the <code>level.criteria</code> parameter for that column plus an extra node titled "Other" for that column that aggregates all the remaining values not included in the filter criteria provided in <code>level.criteria</code> for that column.</li> <li>• Providing a column name ending with both symbols (e.g., "hp:*+", "hp:3+") in the suffix will return a node for all distinct values for that column up to the limit imposed by either the <code>node.limit</code> or the <code>n</code> value plus an additional "Other" node aggregating any remaining values beyond the <code>node.limit</code> or <code>n</code>, if applicable. If the number of distinct values is <math>\leq</math> the <code>node.limit</code> or <code>n</code> then the "Other" node will not be created.</li> </ul>
<code>node.limit</code>	<p>Numeric value. When providing a column in <code>node.levels</code> with an ":*" suffix, the <code>node.limit</code> will limit how many distinct values to actually process to prevent run-away queries and unreadable trees. The limit defaults to 3 (not including an additional 4th if requesting to provide an "Other" node as well with a ":+ " suffix). If the number of distinct values for the column is greater than the <code>node.limit</code>, the tree will include the Top "X" values based on count, where "X" = <code>node.limit</code>. If the <code>node.limit</code> is greater than the number of distinct values for the column, it will be ignored.</p>

<code>level.criteria</code>	A data.frame consisting of 4 character columns containing column names (matching – without suffixes – the columns in <code>node.levels</code> that will use the criteria in <code>level.criteria</code> to determine the filters used for each node), an operator or boolean function (e.g., " <code>==</code> ", " <code>&gt;</code> ", " <code>is.na</code> ", " <code>is.null</code> "), a value, and a corresponding node title for the node displaying that criteria. E.g., " <code>wt</code> ", " <code>&gt;=</code> ", " <code>4000</code> ", " <code>Heavy Cars</code> "
<code>label.vals</code>	Character vector of additional values to include in the node provided as a character vector. The values must take the form of <code>dplyr summarise</code> functions (as characters) and include the columns the functions should be run against (e.g., " <code>min(hp)</code> ", " <code>mean(hp)</code> ", etc.). If no custom suffix is added, the summary function itself will be used as the label. Similar to <code>node.levels</code> a custom suffix can be added using ":" to print a more meaningful label (e.g., " <code>mean(hp):Avg HP</code> "). In this example, the label printed in the node will be " <code>Avg HP:</code> ", otherwise it would be <code>mean_hp</code> (note that the parens "(" and ")" are removed to be rendered in HTML without error). As with <code>node.levels</code> , the column name itself cannot have ":" and must be replaced in the data.frame before being passed in to <code>muir</code> as the <code>data</code> param.
<code>tree.dir</code>	Character. The direction the tree graph should be rendered. Defaults to " <code>LR</code> " <ol style="list-style-type: none"> <li>1. Use "<code>LR</code>" for left-to-right</li> <li>2. Use "<code>RL</code>" for right-to left</li> <li>3. Use "<code>TB</code>" for top-to-bottom</li> <li>4. User "<code>BT</code>" for bottom-to-top</li> </ol>
<code>show.percent</code>	Logical. Should nodes show the percent of records represented by that node compared to the total number of records in <code>data</code> . Defaults to <code>TRUE</code>
<code>num.precision</code>	Number of digits to print numeric label values out to
<code>show.empty.child</code>	Logical. Show a balanced tree with children nodes that are all empty or stop expanding the tree once there is a parent node that is empty. Defaults to <code>FALSE</code> – don't show empty children nodes
<code>tree.height</code>	Numeric. Control tree height to zoom in/out on nodes. Passed to <code>DiagrammeR</code> as <code>height</code> param. Defaults to <code>-1</code> , which appears to optimize the tree size for viewing (still researching why exactly that works! :-))
<code>tree.width</code>	Numeric. Control tree width to zoom in/out on nodes. Passed to <code>DiagrammeR</code> as <code>width</code> param. Defaults to <code>-1</code> , which appears to best optimize the tree size for viewing (still researching why exactly that works! :-))

### Value

An object of class `htmlwidget` (via `DiagrammeR`) that will intelligently print itself into HTML in a variety of contexts including the R console, within R Markdown documents, and within Shiny output bindings.

### Examples

```
## Not run:
# Load in the 'mtcars' dataset
```

```

data(mtcars)

# Basic exploration - show all values
mtTree <- muir(data = mtcars, node.levels = c("cyl:*", "carb:*"))
mtTree

# Basic exploration - show all values overriding default node.limit
mtTree <- muir(data = mtcars, node.levels = c("cyl:*", "carb:*"), node.limit = 5)
mtTree

# Show all values overriding default node.limit differently for each column
mtTree <- muir(data = mtcars, node.levels = c("cyl:2", "carb:5"))
mtTree

# Show all values overriding default node.limit for each column
# and aggregating all distinct values above the node.limit into a
# separate "Other" column to collect remaining values

# Top 2 occurring 'carb' values will be returned in their own nodes,
# remaining values/counts will be aggregated into a separate "Other" node
mtTree <- muir(data = mtcars, node.levels = c("cyl:2", "carb:2+"))
mtTree

# Add additional calculations to each node output (dplyr::summarise functions)
mtTree <- muir(data = mtcars, node.levels = c("cyl:2", "carb:2+"),
label.vals = c("min(wt)", "max(wt)"))
mtTree

# Make new label values more reader-friendly
mtTree <- muir(data = mtcars, node.levels = c("cyl:2", "carb:2+"),
label.vals = c("min(wt):Min Weight", "max(wt):Max Weight"))
mtTree

# Instead of just returning top counts for columns provided in \code{node.levels},
# provide custom filter criteria and custom node titles in \code{label.vals}
# (criteria could also be read in from a csv file as a data.frame)
criteria <- data.frame(col = c("cyl", "cyl", "carb"),
oper = c("<", ">=", "=="),
val = c(4, 4, 2),
title = c("Less Than 4 Cylinders", "4 or More Cylinders", "2 Carburetors"))

mtTree <- muir(data = mtcars, node.levels = c("cyl", "carb"),
level.criteria = criteria,
label.vals = c("min(wt):Min Weight", "max(wt):Max Weight"))
mtTree

# Use same criteria but show all other values for the column where NOT
# EQUAL to the combination of the filters provided for that column (e.g., for cyl
# where !(cyl < 4 | cyl >= 4) in an "Other" node
mtTree <- muir(data = mtcars, node.levels = c("cyl:+", "carb:+"),
level.criteria = criteria,
label.vals = c("min(wt):Min Weight", "max(wt):Max Weight"))
mtTree

```

```
# Show empty child nodes (balanced tree)
mtTree <- muir(data = mtcars, node.levels = c("cyl:+", "carb:+"),
  level.criteria = criteria,
  label.vals = c("min(wt):Min Weight", "max(wt):Max Weight"),
  show.empty.child = TRUE)
mtTree

# Save tree to HTML file with \code{htmlwidgets} package to working directory
mtTree <- muir(data = mtcars, node.levels = c("cyl:2", "carb:2+"))
htmlwidgets::saveWidget(mtTree, "mtTree.html")

## End(Not run)
```

# Index

`build_tree`, 2

`muir`, 2