Package: sewage (via r-universe)

September 18, 2024

Type Package

Title A Light-Weight Data Pipelining Tool

Version 0.2.5.9000

Description Provides a simple interface to developing complex data pipelines which can be executed in a single call. 'sewage' makes it easy to test, debug, and share data pipelines through it's interface and visualizations.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.2.1

URL https://github.com/mwhalen18/sewage, https://mwhalen18.github.io/sewage/

BugReports https://github.com/mwhalen18/sewage/issues

Suggests testthat (>= 3.0.0), dplyr, knitr, rmarkdown

Config/testthat/edition 3

Imports DiagrammeR, glue, cli

VignetteBuilder knitr

Repository https://mwhalen18.r-universe.dev

RemoteUrl https://github.com/mwhalen18/sewage

RemoteRef HEAD

RemoteSha 8b9d8087cc75a9702f1aa48d83af9cd409c6cdd2

Contents

---

add_node *add node to a sewage pipeline*

Description

`add_node()` will place a new node in the specified pipeline. This will be executed sequentially when the pipeline is executed using `run()`

Usage

```
add_node(pipeline, component, name, input, ...)
```

Arguments

`pipeline`	an initialized sewage pipeline
`component`	a function to be executed. Must be a valid function specification or exported sewage object including `Joiner` and `Splitter`
`name`	a name to give to the given component. This will be used as the 'input' parameter for downstream nodes
`input`	the node to use as input into 'component'. Inputs should be either (1) the name of an existing node in the pipeline, or (2) the name(s) of any argument(s) in the first ndoe of the pipeline. These names can be whatever you want, but should match the arguments you pass to `run()`
`...`	additional arguments to be passed to the 'component' argument

Value

a `sewage_pipeline` object

Examples

```
my_func = function(df) {
    df %>%
        head(15)
}
pipeline = Pipeline()
pipeline = pipeline |>
    add_node(name = 'processor', component = my_func, input = 'file')
```

draw.sewage_pipeline *Visualize a pipeline*

Description

This function draws a DAG of the existing pipeline flow. For additional information see `igraph::spec_viz`

Usage

```
## S3 method for class 'sewage_pipeline'
draw(pipeline, ...)

draw(pipeline, ...)
```

Arguments

`pipeline`	an instantiated `pipeline` object
`...`	reserved for future use

Value

an `htmlwdget` object

Joiner *Initialize a Joiner object*

Description

The `Joiner` takes in objects and joins them according to a defined method into a single node.

Usage

```
Joiner(method)
```

Arguments

`method`	function to join incoming objects together

Value

a `sewage_joiner` object

Note

additional arguments to be passed to `method` should be passed in the `...` of [add_node()]

Examples

```
pipeline = Pipeline() |>
    add_node(Joiner(method = rbind), name = "Joiner", input = c("file1", "file2"))
```

Pipeline *Initialize a sewage Pipeline*

Description

Initialize a sewage Pipeline

Usage

```
Pipeline()
```

Value

A sewage pipeline object

---

print.sewage_pipeline *Printing Pipelines*

Description

print a sewage pipeline

this will print all nodes and theis inputs in the pipeline. Once the pipeline has been executed, print will show the outputs available through [pull_output()]

Usage

```
## S3 method for class 'sewage_pipeline'
print(x, ...)
```

Arguments

x	a [Pipeline()] object
...	not used

Value

formatted sewage pipeline output

Examples

```
pipeline = Pipeline() |>
    add_node(component = head, name = "Head", input = "file")
print(pipeline)
```

pull_output *Extract output components from a pipeline*

Description

Extract output components from a pipeline

Usage

```
pull_output(x, component, ...)

## S3 method for class 'sewage_pipeline'
pull_output(x, component, ...)
```

Arguments

x	an executed pipeline object
component	a character string specifying which output component to pull
...	reserved for future use

Value

output from a terminating node of an executed sewage pipeline

Examples

```
pipeline = Pipeline() |>
    add_node(component = head, name = "Head", input = 'file')
result = run(pipeline, file = iris)
pull_output(result, "Head")
```

run *Run a pipeline*

Description

This function is the extry point for executing a pipeline object

Usage

```
run(pipeline, start = NULL, halt = NULL, ...)
```

Arguments

pipeline	an initialized pipeline object
start	node at which to start execution. If NULL then execution will start at the first node
halt	halt execution at a specified node. Adding this parameter will halt execution of the remainder of the pipeline. Note that because pipelines are executed sequentially in the order you add them to the pipeline, in the case of a branching pipeline, any nodes from a different branch that were specified earlier in the pipeline will still be executed.
...	parameter(s) to pass to starting node of the pipeline. This should match the 'input' parameter of 'add_node' of the starting node. In the case that you have multiple inputs or are starting at a later point in the pipeline, each argument should match the name of a starting node in your pipeline.

Value

an executed sewage_pipeline object

Examples

```
func1 = function(x) {
    x
}
pipeline = Pipeline() |>
    add_node(component = func1, name = "Func1", input = "file") |>
    add_node(component = func1, name = "Func2", input = "Func1") |>
    add_node(component = func1, name = "Func3", input = "Func2")
run(pipeline, file = mtcars)
run(pipeline, start = "Func2", Func1 = iris)
run(pipeline, halt = "Func2", file = mtcars)
```

Splitter *Initialize a splitter object*

Description

`Splitter` takes in exactly one input node and propogates the input to *n* output nodes.

Usage

```
Splitter(edges = 2)
```

Arguments

edges	number out outputs. Must be greater than 1

Details

After executing a `Splitter` object, the pipeline will contains *n* outputs and will be named as `SplitterName_output{i}`.

Value

a `sewage_splitter` object

Note

The ouputs of a `Splitter` object are accessed through the naming convention `{name}.output_{i}` where `name` is the specified name of the Splitter object. This allows you to pass split objects to downstream nodes or access them through the pipeline results.

Examples

```
pipeline = Pipeline()
pipeline = pipeline |>
    add_node(name = 'Splitter', component = Splitter(), input = 'file')
result = run(pipeline, file = mtcars)
pull_output(result, 'Splitter.output_1')
pull_output(result, 'Splitter.output_2')
```

Index