The traitr package

John Verzani

CUNY/The College of Staten Island

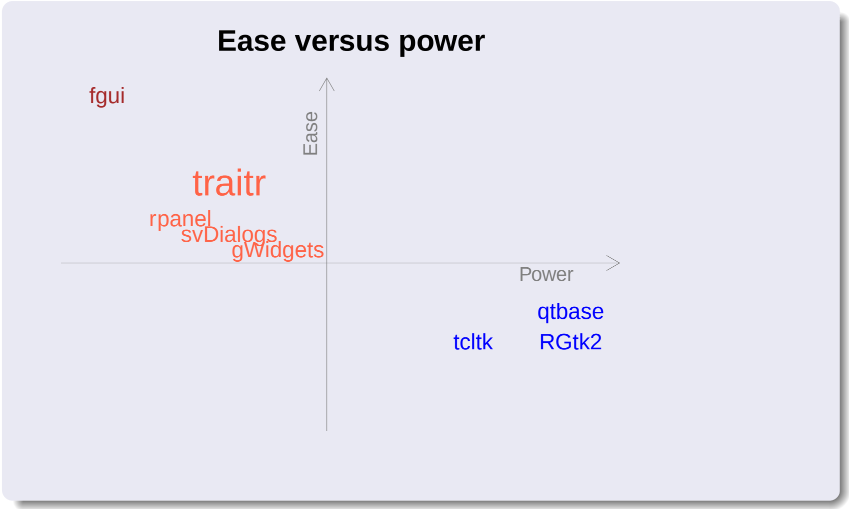
useR!2010

What is traitr? – why the funny name?

The `traitr` package facilitates the making of dialogs for GUIs

- Need not know GUI programming at all – focus is on type of data, *not* the type of widget
- Adding interactivity is straightforward
- Package uses `gWidgets` for the GUI parts - can use `RGtk2` (best), `tcltk` or `qtbase`.
- inspired by the traitsUI module for python

Where traitr sits



Comparison: native toolkit to traitr

We begin with a simple comparison of how one might build a GUI for a function which performs a t-test for summarized data.

Signature of our ttest function

```
function(xbar,    # numeric
         s,       # positive numeric
         n,       # integer
         mu,      # numeric
                  # a choice:
         alternative=c("two.sided", "less", "greater")
         )
```

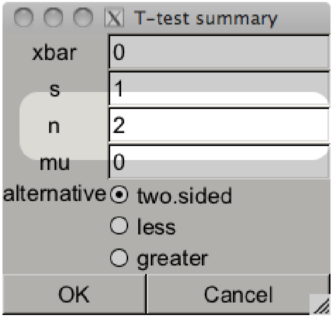
Our GUI will

- Gather the values from the user
- **Transport** the values back to R, **coerce** to the right type, then **call** the `ttest` function

A basic dialog in RGtk2

RGtk2 snippet

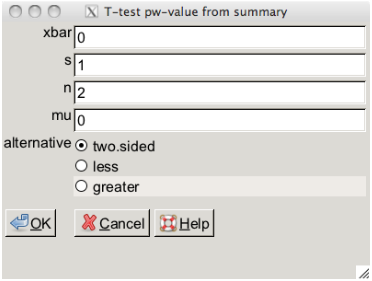
```
## Construction
n <- gtkEntryNew()
n$setText(2)
## Layout
tbl$attach(gtkLabel("n"),
                0, 1, i-1, i)
tbl$attach( n,  1, 2, i-1, i)

## Transport (GUI -> R); coerce
val <- n$getText()
val <- as.integer(val)
```


T-test summary

xbar	0
s	1
n	2
mu	0
alternative	[✓ two.sided] less greater

The entire GUI took over 40 lines of code and was a bit tedious to write (glade, strawman, ...).

A basic dialog with traitr

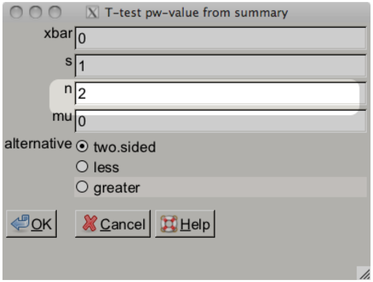


T-test pw-value from summary

xbar	0
s	1
n	2
mu	0
alternative	[✓ two.sided] less greater

```
dlg <- aDialog(items=list(
    xbar=numericItem(0),
       s=numericItem(1),
       n=integerItem(2),
      mu=numericItem(0),
    alternative=choiceItem(
       value="two.sided",
       values=c("two.sided", "less", "greater"))
    ),
  title="T-test p-value from summary",
  help_string="Adjust values then click 'OK'",
  OK_handler=function(.) print(do.call(ttest, .$to_R()))
  )
dlg$make_gui()
```

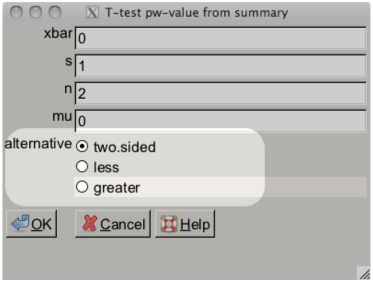
A basic dialog with traitr



xbar: 0
s: 1
n: 2
mu: 0
alternative: [✓ two.sided] less greater

```
dlg <- aDialog(items=list(
    xbar=numericItem(0),
    s=numericItem(1),
    n=integerItem(2),
    mu=numericItem(0),
    alternative=choiceItem(
       value="two.sided",
       values=c("two.sided", "less", "greater"))
    ),
  title="T-test p-value from summary",
  help_string="Adjust values then click 'OK'",
  OK_handler=function(.) print(do.call(ttest, .$to_R()))
  )
dlg$make_gui()
```

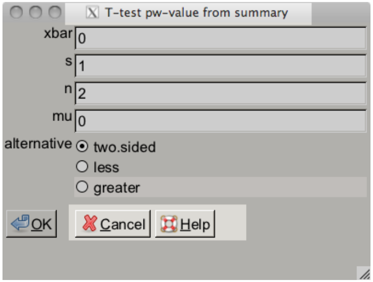
A basic dialog with traitr



xbar 0
s 1
n 2
mu 0
alternative [✓ two.sided] less greater

```
dlg <- aDialog(items=list(
    xbar=numericItem(0),
    s=numericItem(1),
    n=integerItem(2),
    mu=numericItem(0),
    alternative=choiceItem(
       value="two.sided",
       values=c("two.sided", "less", "greater"))
    ),
  title="T-test p-value from summary",
  help_string="Adjust values then click 'OK'",
  OK_handler=function(.) print(do.call(ttest, .$to_R()))
  )
dlg$make_gui()
```

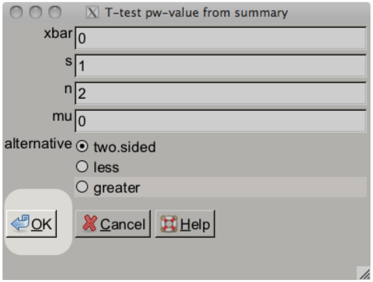
A basic dialog with traitr



xbar 0
s 1
n 2
mu 0
alternative [✓ two.sided] less greater

```
dlg <- aDialog(items=list(
    xbar=numericItem(0),
    s=numericItem(1),
    n=integerItem(2),
    mu=numericItem(0),
    alternative=choiceItem(
       value="two.sided",
       values=c("two.sided", "less", "greater"))
    ),
  title="T-test p-value from summary",
  help_string="Adjust values then click 'OK'",
  OK_handler=function(.) print(do.call(ttest, .$to_R()))
  )
dlg$make_gui()
```

A basic dialog with traitr


T-test pw-value from summary

xbar 0

s 1

n 2

mu 0

alternative [✓ two.sided] less greater

```
dlg <- aDialog(items=list(
    xbar=numericItem(0),
    s=numericItem(1),
    n=integerItem(2),
    mu=numericItem(0),
    alternative=choiceItem(
       value="two.sided",
       values=c("two.sided", "less", "greater"))
    ),
    title="T-test p-value from summary",
    help_string="Adjust values then click 'OK'",
    OK_handler=function(.) print(do.call(ttest, .$to_R()))
  )
  dlg$make_gui()
```

Proto Methods

The package uses `proto` to provide a lightweight, object-oriented style with mutatable objects. Proto methods have an odd signature

proto method definition template

```
function(., x, y) { ... }
```

The “.” is a reference to the proto object (`self` in javascript)

"." passed by the $ calling mechanism

```
obj$meth_name(x, y)
```

Some key traitr methods for dialogs

`make_gui`	draws the GUI for a dialog
`OK_handler`	Called when the OK button is clicked.
`to_R`	Return a list each items value
`get_NAME`, `set_NAME`	getters/setters for NAME property

Refinements: validation

There are a handful of ways to refine our GUI that are not hard to implement.

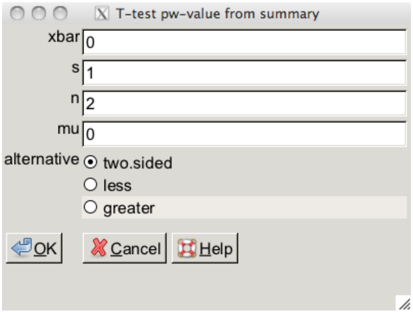
Validation: a positive standard deviation

```
tmp <- dlg1$get_item_by_name("s")        # item look up
tmp$validate <- function(., rawvalue)
  if(as.numeric(rawvalue) > 0) {
    return(rawvalue)
  } else {
    stop("s must be positive")           # throw error
  }
```

Throws an error

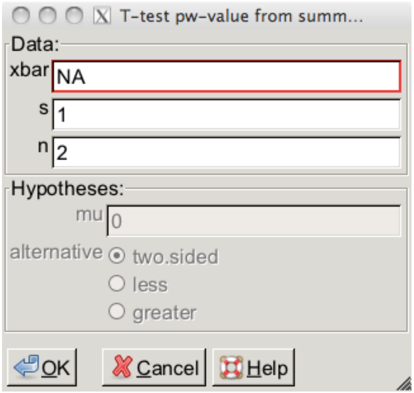
```
> tmp$set_s(0)
```

Refinements: Adjusting the layout with a view



T-test pw-value from summary

xbar	0
s	1
n	2
mu	0
alternative	[✓ two.sided] less greater



T-test pw-value from summ...

Data:

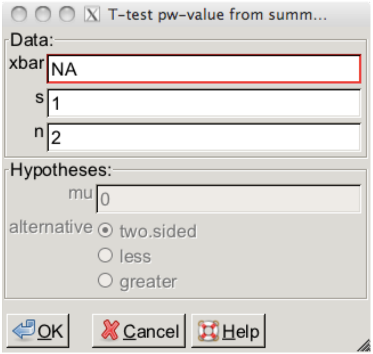
xbar	NA
s	1
n	2

Hypotheses:

mu	0
alternative	[✓ two.sided] less greater

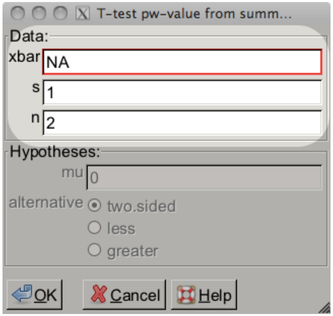
Refinements: Adjusting the layout with a view

```
dlg2 <- dlg$instance()
dlg2$set_xbar(NA)# no default

view <- aContainer(
  aFrame(
    label="Data:",
    aContainer("xbar","s","n")),
  aFrame(
    label="Hypotheses:",
    enabled_when=function(.)
      !is.na(.$get_xbar()),
    aContainer("mu","alernative"))
  )
dlg2$make_gui(gui_layout=view)
```



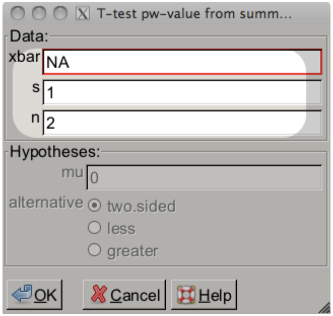
Refinements: Layouts

```
view <- aContainer(
  aFrame(
    label="Data:",
    aContainer("xbar","s","n")),
  aFrame(
    label="Hypotheses:",
    enabled_when=function(.)
      !is.na(.$get_xbar()),
    aContainer("mu","alernative"))
  )
```



Refinements: Layouts

```
view <- aContainer(
  aFrame(
    label="Data:",
    aContainer("xbar","s","n")),
  aFrame(
    label="Hypotheses:",
    enabled_when=function(.)
      !is.na(.$get_xbar()),
    aContainer("mu","alernative"))
  )
```


T-test pw-value from summ...

Data:

xbar NA

s 1

n 2

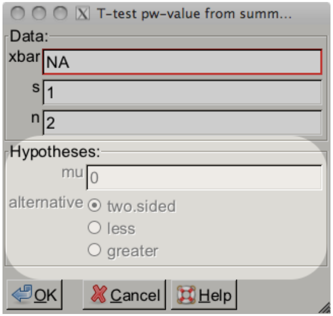
Hypotheses:

mu 0

alternative [✓ two.sided] less greater

Refinements: Layouts

```
view <- aContainer(
  aFrame(
    label="Data:",
    aContainer("xbar","s","n")),
  aFrame(
    label="Hypotheses:",
    enabled_when=function(.)
      !is.na(.$get_xbar()),
    aContainer("mu","alernative"))
  )
```


T-test pw-value from summ...

Data:

xbar: NA

s: 1

n: 2

Hypotheses:

mu: 0

alternative: (●) two.sided ( ) less ( ) greater

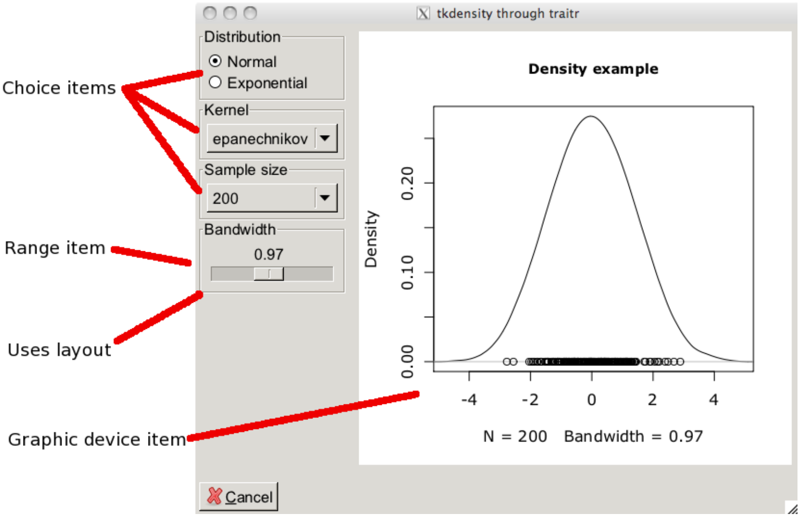
observers

Traitr has a simple implementation of the Model-View-Controller paradigm, where different components can observe changes to the other.
Dialogs observe themselves, so one need only define appropriately named methods to make changes in one item of a dialog propagate to other items.

Special method names for dialogs

`model_value_changed`	Called when any value is modified
`property_NAME_value_changed`	Called when property `NAME` has been modified.

tkdensity


Choice items
Range item
Uses layout
Graphic device item
tkdensity through traitr
Distribution
Normal
Exponential
Kernel
epanechnikov
Sample size
200
Bandwidth
0.97
Density example
Density
0.00
0.10
0.20
-4
-2
0
2
4
N = 200 Bandwidth = 0.97
Cancel

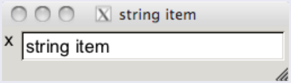
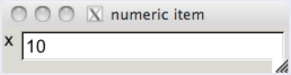
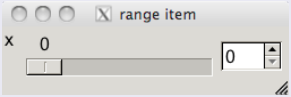
tcltk density continued

```
## modelItems a list of items already defined
modelItems$out <- graphicDeviceItem()   # New item type
dlg <- aDialog(
  items= modelItems,           # also dist, kernel, n, bw
  help_string="Adjust a parameter to update graphic",
  title="tkdensity through traitr",
  buttons="Cancel",
  model_value_changed=function(.) {
    do.call(makePlot, .$to_R())
  })
#
dlg$make_gui(gui_layout=view)
dlg$model_value_changed()                  # initial plot
```

Items – the basic unit

Items implement the model-view-controller pattern too.

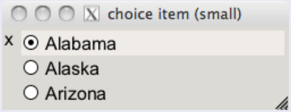
Simple mappings

`stringItem`	 string item x string item	A basic item for holding a string value
`numericItem`	 numeric item x 10	A basic item for holding a numeric value
`rangeItem`	 range item x 0 0	For selecting from a range of values (seq)

Different Editors

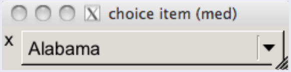
Editor types

`choiceItem`



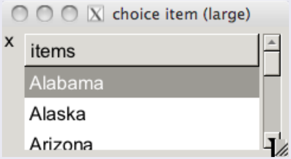
First few states

`choiceItem`



First 10 states

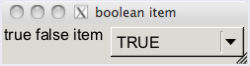
`choiceItem`

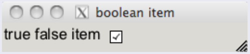


All states

Different styles: compact

`trueFalseItem`





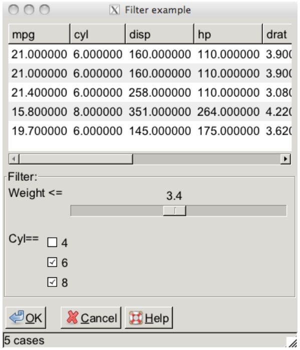
Types of items

In addition to the item types illustrated so far, at this point there are also:

Other item types

`expressionItem`	expression is `eval`-parsed
`dateItem`	for selecting a date
`fileItem`	for file selection
`buttonItem`	to implement an action
`labelItem`	for text labels
`separatorItem`	for layout
`variableSelectorItem`	to select a data frame's variable
`imageItem`	to place an image
`tableItem`	to select from a table
`itemList`	an interface to a list of similar items

Example: Filtering



Filter example

mpg	cyl	disp	hp	drat
21.000000	6.000000	160.000000	110.000000	3.90(
21.000000	6.000000	160.000000	110.000000	3.90(
21.400000	6.000000	258.000000	110.000000	3.08(
15.800000	8.000000	351.000000	264.000000	4.22(
19.700000	6.000000	145.000000	175.000000	3.62(

Filter:

Weight <= 3.4

Cyl== ☐ 4 ☑ 6 ☑ 8

5 cases

The table item allows one to display tabular data. Filtering such data is a common desire. For our next example we have this method to update a data set:

```
do_find_ind <- function(., value, old_value) {
  ind <-  mtcars$wt <= .$get_wt() &
          mtcars$cyl %in% .$get_cyl()
  .$set_tbl(.$data[ind,])
}
```

Filtering continued

```
dlg <- aDialog(items=list(
    tbl=tableItem(mtcars, attr=list(size=c(300,200))),
    wt=rangeItem(max(wt), from=min(wt), to=max(wt), by=.1,
      label="Weight <=",
      tooltip="Slide to adjust maximum weight for data"),
    cyl=choiceItem(cyls, values=cyls, multiple=TRUE,
      label="Cyl==",
      tooltip="Restrict number of cylinders in data set")),
  data=mtcars,                    # add property
  status_text=sprintf("%s cases",nrow(mtcars)),
  #
  property_wt_value_changed=do_find_ind,
  property_cyl_value_changed=do_find_ind,
  property_tbl_value_changed=function(., value, old_value)
    .$set_status_text(sprintf("%s cases", nrow(value)))
  },
```

Conclusion

Future plans

- Add some more items (formula item, data editor, ...)
- Optimize for speed
- Better documentation
- More useRs!