Interactive graphics with Shiny

Winston Chang
RStudio

February 24, 2016

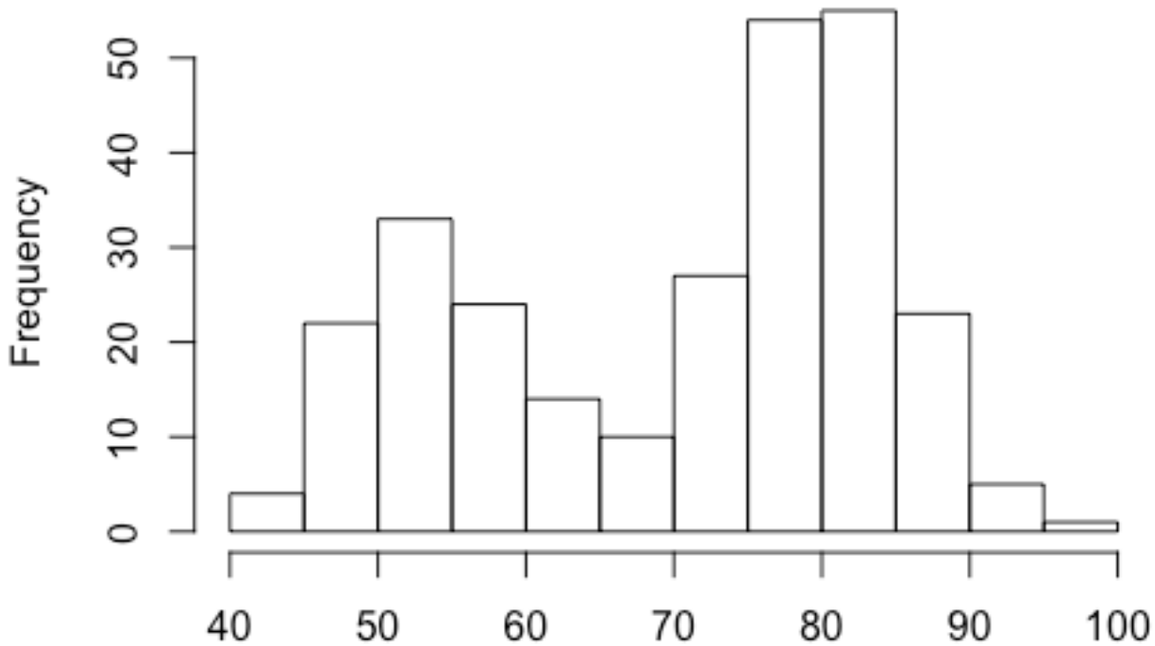
https://github.com/rstudio/webinars

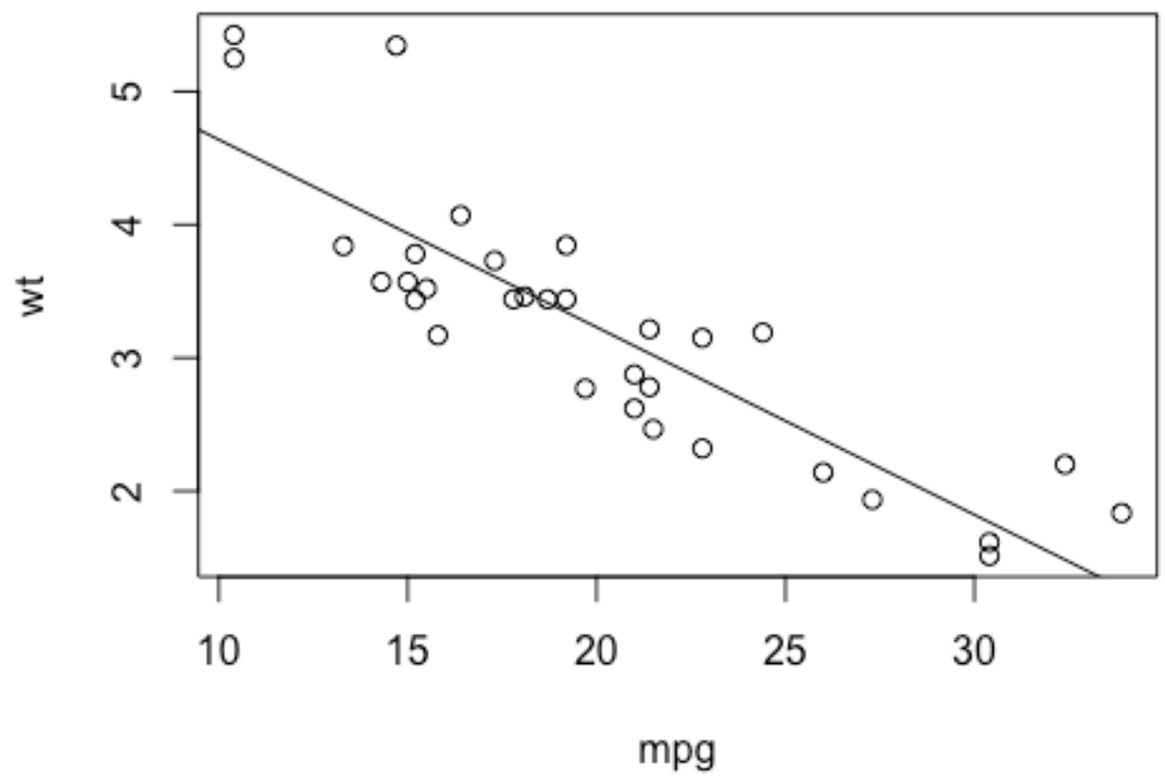
Overview

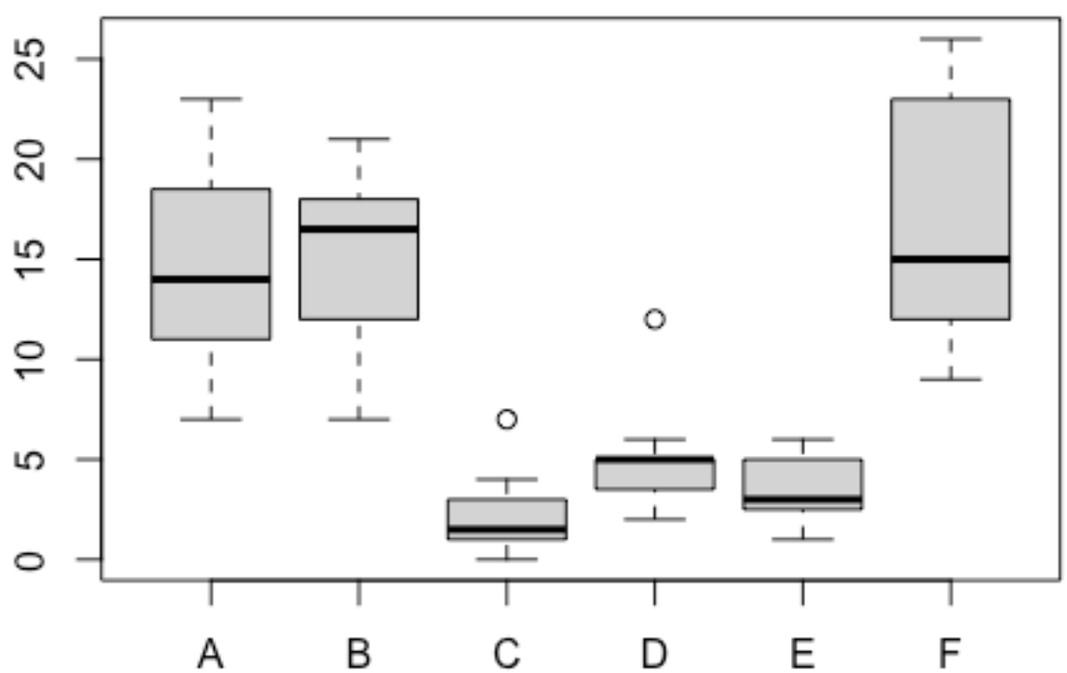
- Static vs. interactive graphics
- Graphics for presentation vs. exploration
- Nuts and bolts of interactive graphics with Shiny
- Shiny Gadgets

Is the graphic static or interactive?

Static graphics



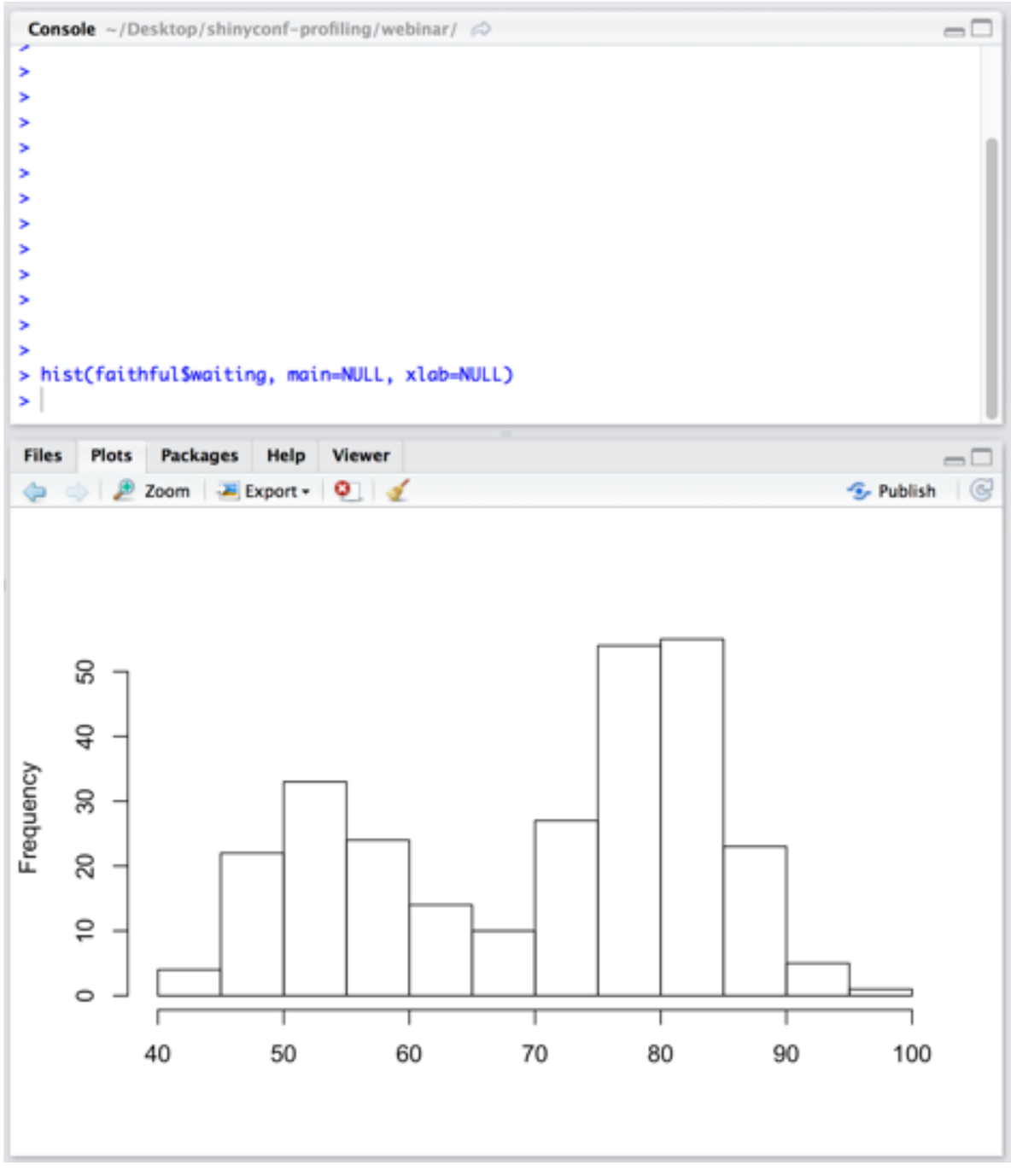




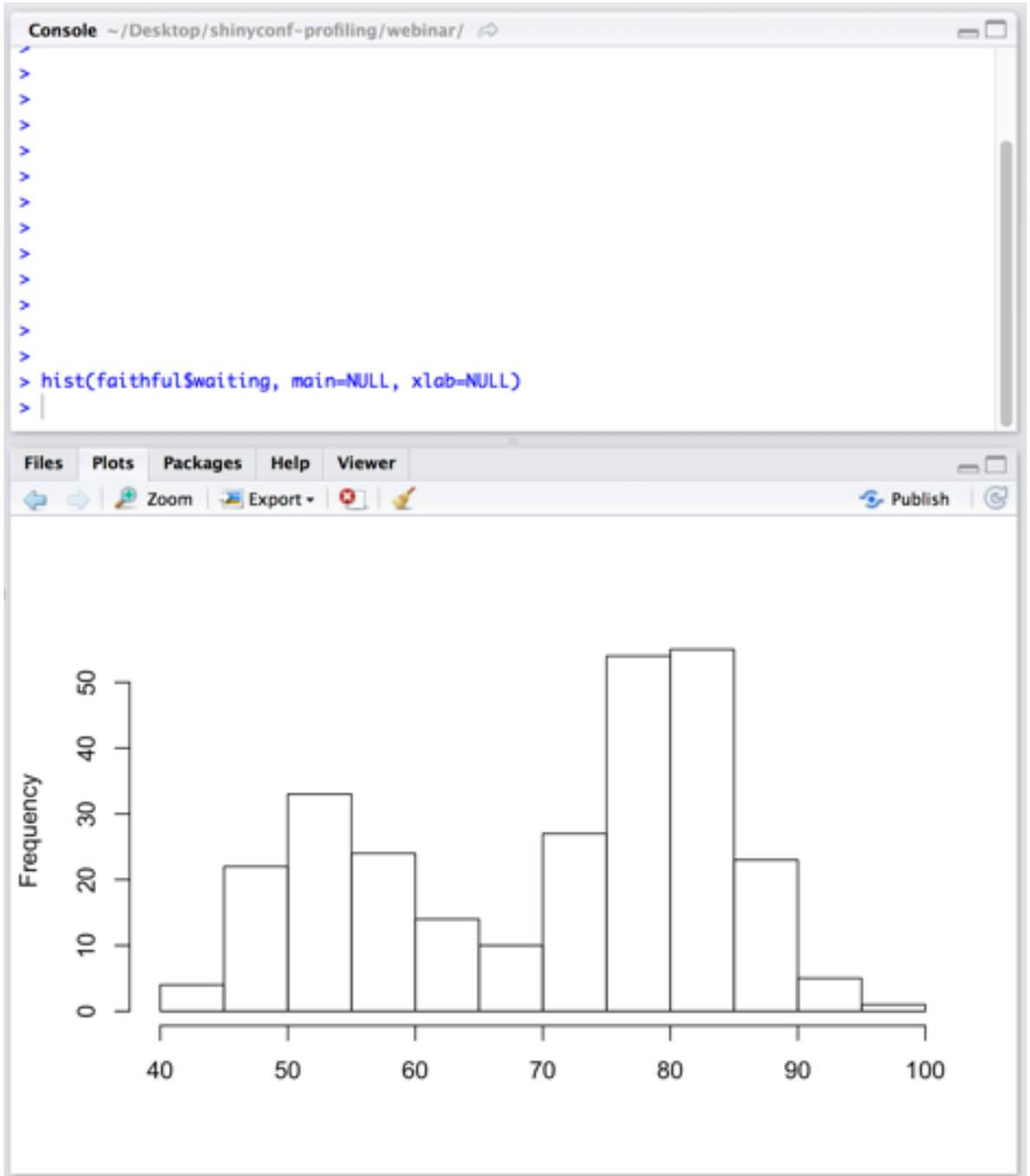
Interactive graphics

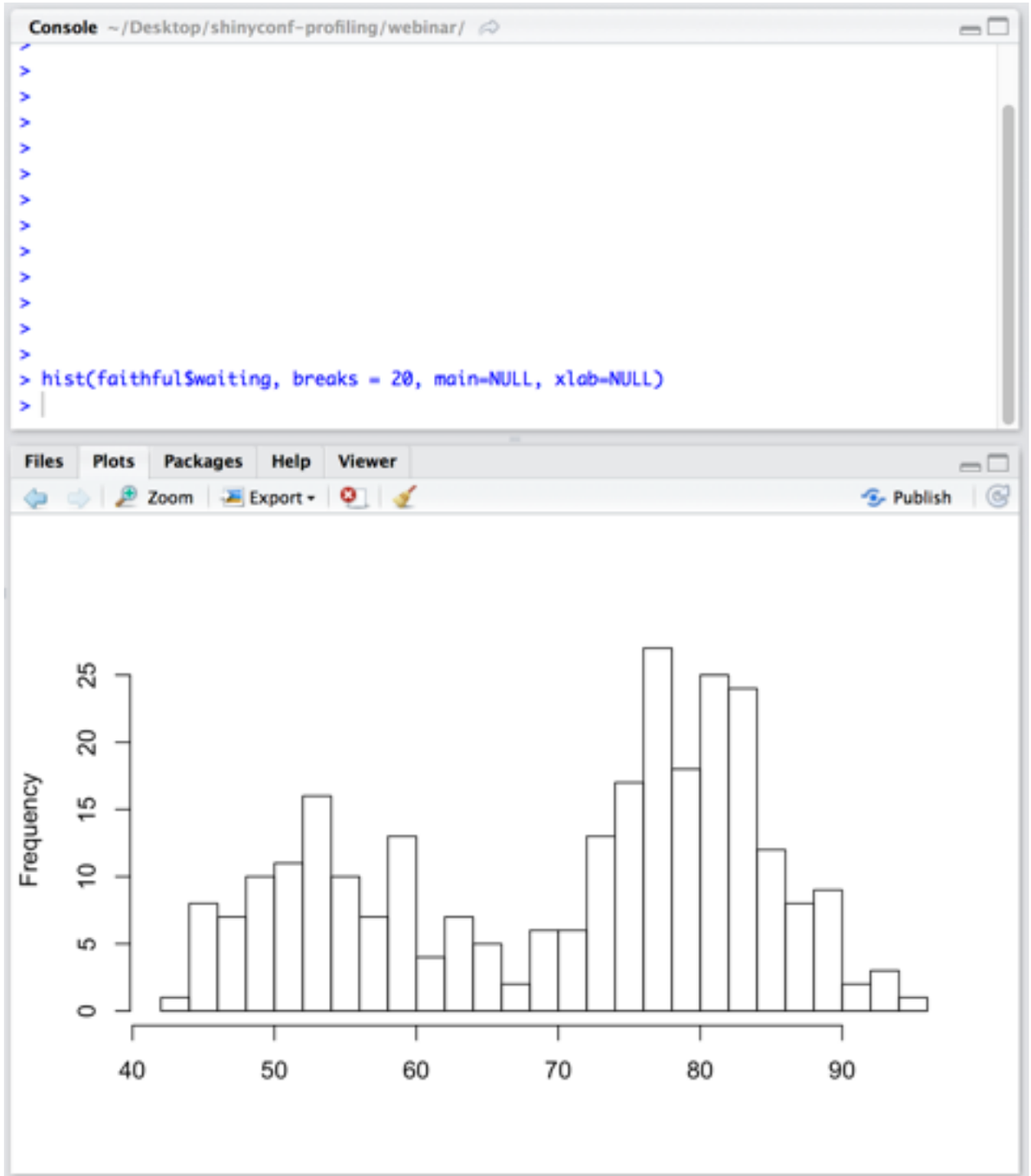
What is the purpose of the graphic?

Graphics for exploration



Graphics for exploration





Graphics for presentation

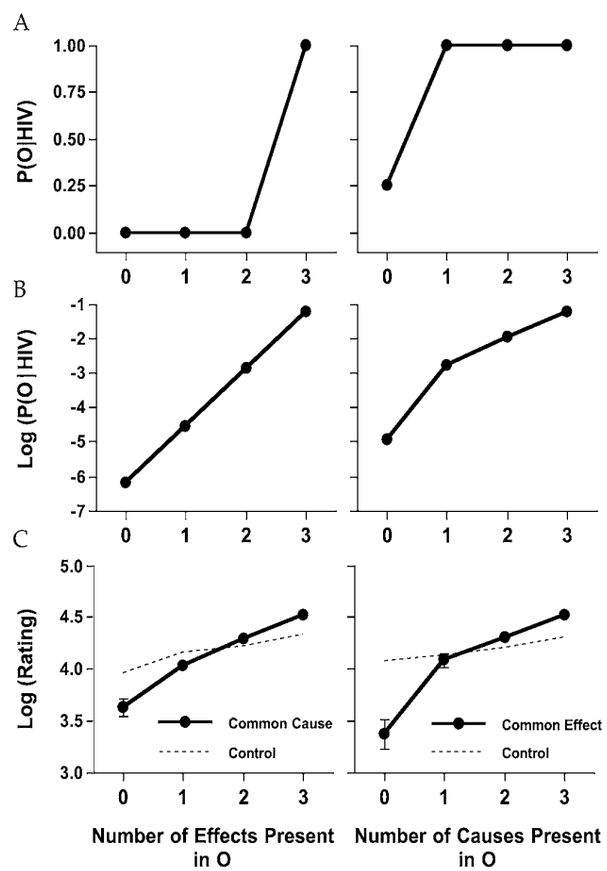


FIGURE 12-7 (A) Theoretical predictions for Experiment 3 assuming deterministic causal relations. (B) Theoretical predictions assuming probabilistic causal relations. (C) Results from Experiment 3.

Figure 12-7B, the logarithm of the categorization ratings have been taken. The left panel of Figure 12-7C indicates that, as predicted, in the common-cause condition the logarithm of the ratings were a linear function of the number of effect features present. In contrast, in the common effect condition those ratings exhibited a nonlinearity in which the presence of one potential cause of D produced a larger increase in the ratings compared to adding a second or third cause (right panel of Figure 12-7C).

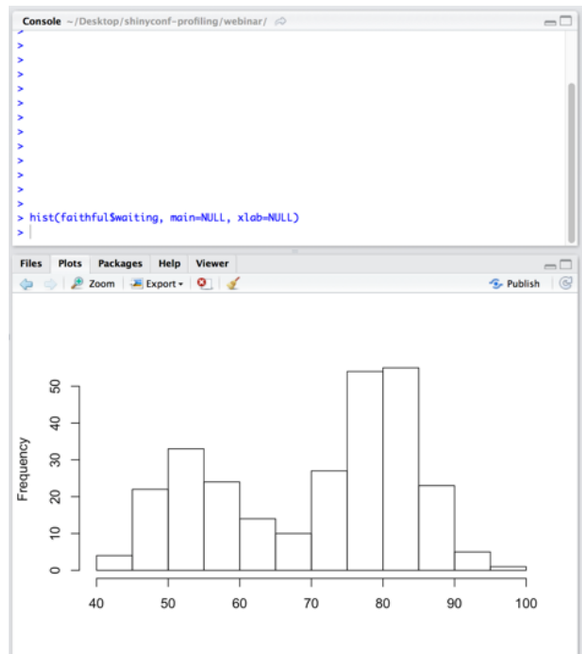
Besides being interesting in their own right, these results have important theoretical implications for models of categorization. For example, Rehder (2003a) has shown not only that standard categorization models like prototype and exemplar models cannot account for asymmetries between common-cause and common-effect networks (like those shown in Figure 7C), but also that those models cannot account for such results even when augmented with certain rudimentary representations of causal relations (e.g., adding to a prototype representation second-order features that encode interfeature causal relations). Of special theoretical importance are the results from the common effect condition that involves higher-order interactions among features—a cause producing a large increase in ratings only when *none* of the other causes are already present. (See Danks, chapter 11, this volume, for an extended discussion of different classes of categorization models and the constraints they place on possible patterns of classification ratings.)

In addition to illustrating the predicted asymmetry between the common-cause and common-effect networks, the results in Figure 12-7C also demonstrate

State of data graphics ~5 years ago

	Exploration	Presentation
Static		
Interactive		?

State of data graphics ~5 years ago

	Exploration	Presentation
Static		
Interactive		?

State of data graphics ~5 years ago

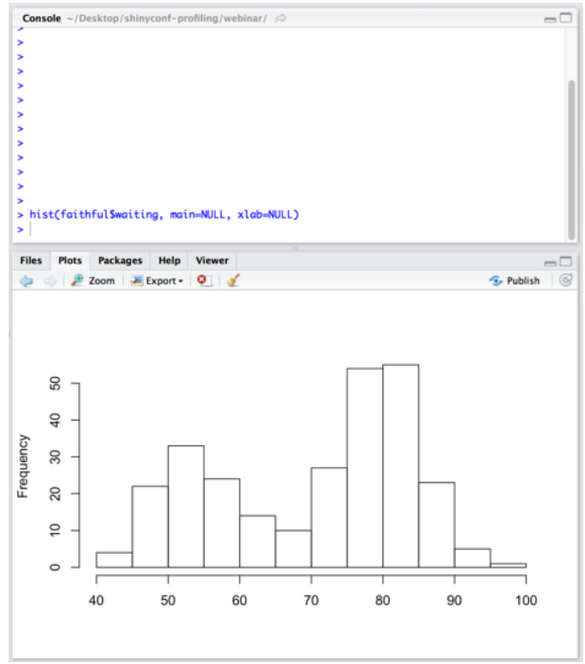
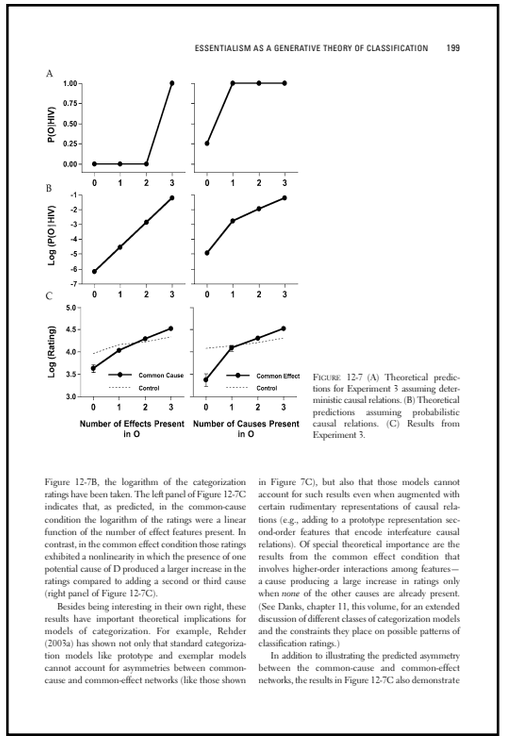
	Exploration	Presentation
Static		
Interactive		?

FIGURE 12-7 (A) Theoretical predictions for Experiment 3 assuming deterministic causal relations. (B) Theoretical predictions assuming probabilistic causal relations. (C) Results from Experiment 3.

Figure 12-7B, the logarithm of the categorization ratings have been taken. The left panel of Figure 12-7C indicates that, as predicted, in the common-cause condition the logarithm of the ratings were a linear function of the number of effect features present. In contrast, in the common effect condition those ratings exhibited a nonlinearity in which the presence of one potential cause of D produced a larger increase in the ratings compared to adding a second or third cause (right panel of Figure 12-7C).

Besides being interesting in their own right, these results have important theoretical implications for models of categorization. For example, Rehder (2003a) has shown not only that standard categorization models like prototype and exemplar models cannot account for asymmetries between common-cause and common-effect networks (like those shown in Figure 7C), but also that those models cannot account for such results even when augmented with certain rudimentary representations of causal relations (e.g., adding to a prototype representation second-order features that encode interfeature causal relations). Of special theoretical importance are the results from the common effect condition that involves higher-order interactions among features—a cause producing a large increase in ratings only when *none* of the other causes are already present. (See Danks, chapter 11, this volume, for an extended discussion of different classes of categorization models and the constraints they place on possible patterns of classification ratings.)

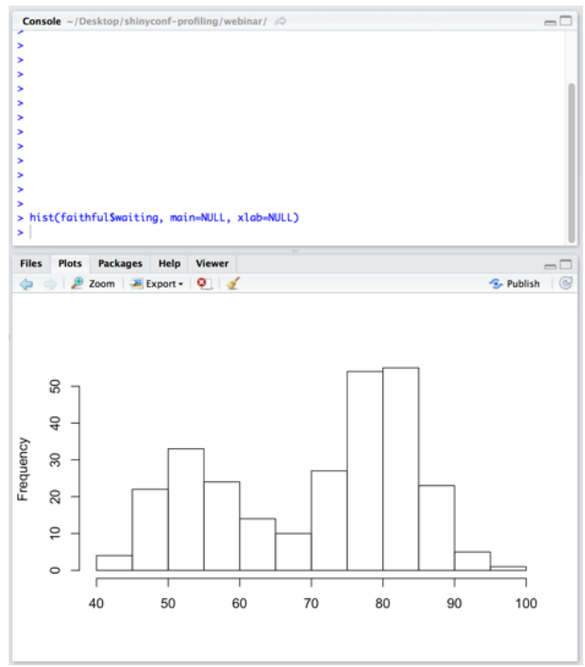
In addition to illustrating the predicted asymmetry between the common-cause and common-effect networks, the results in Figure 12-7C also demonstrate

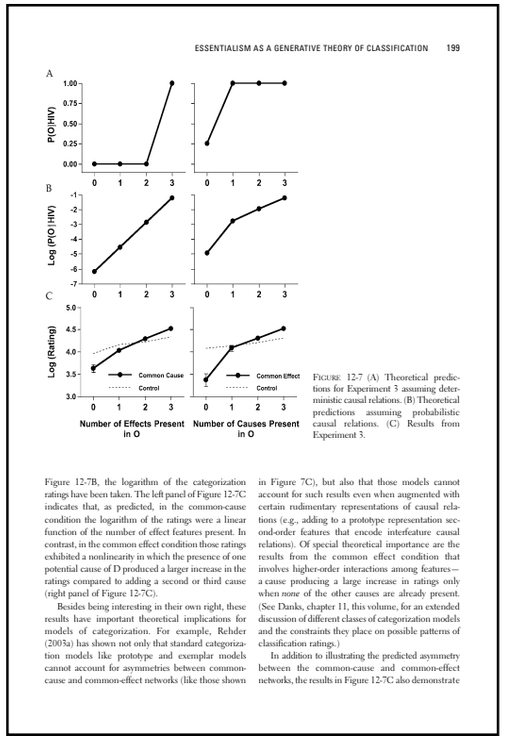
State of data graphics ~5 years ago

Exploration

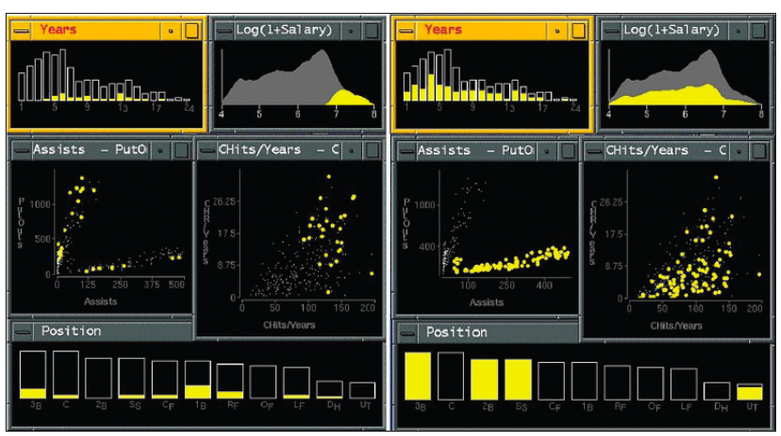
Presentation

Static



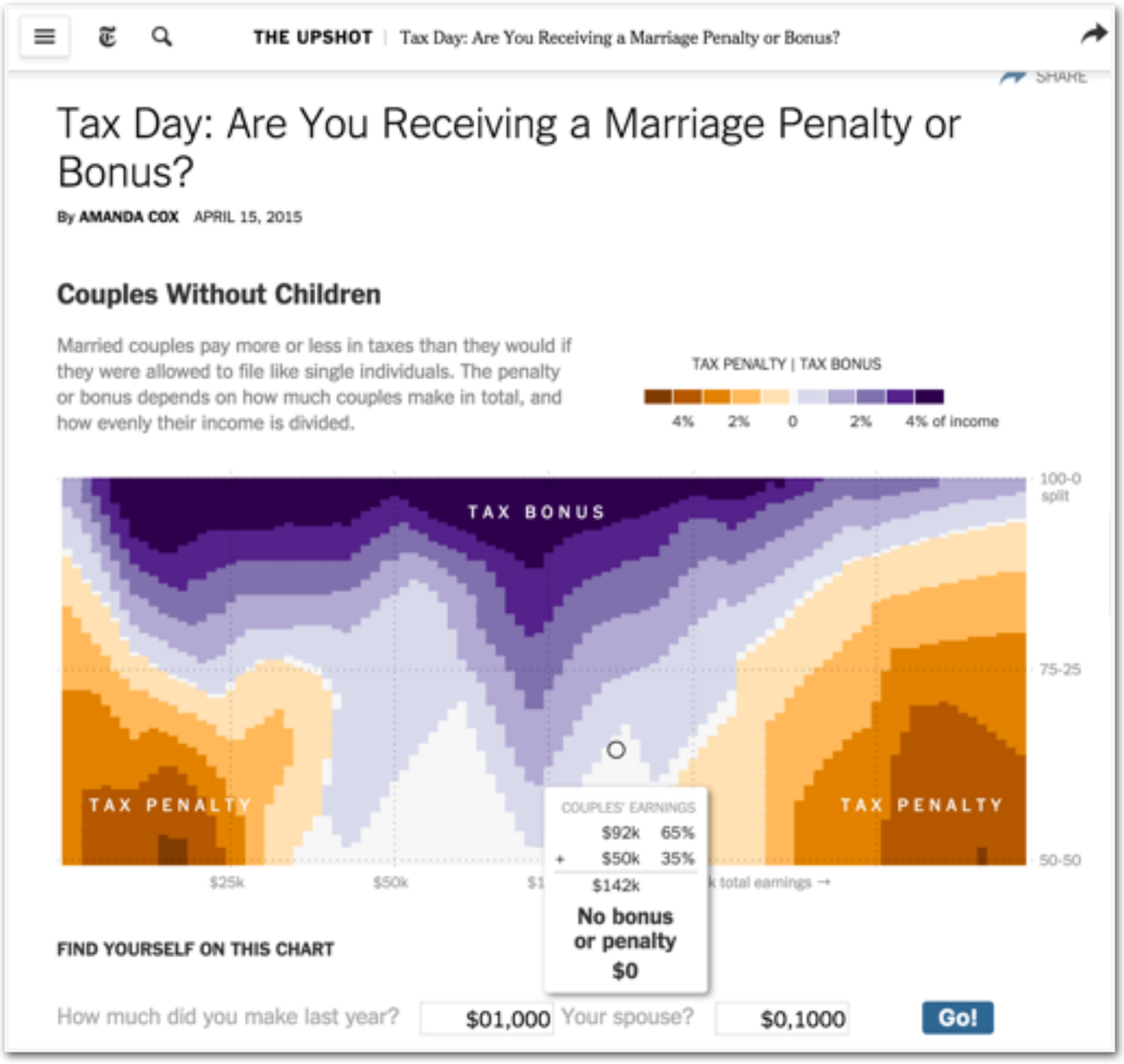


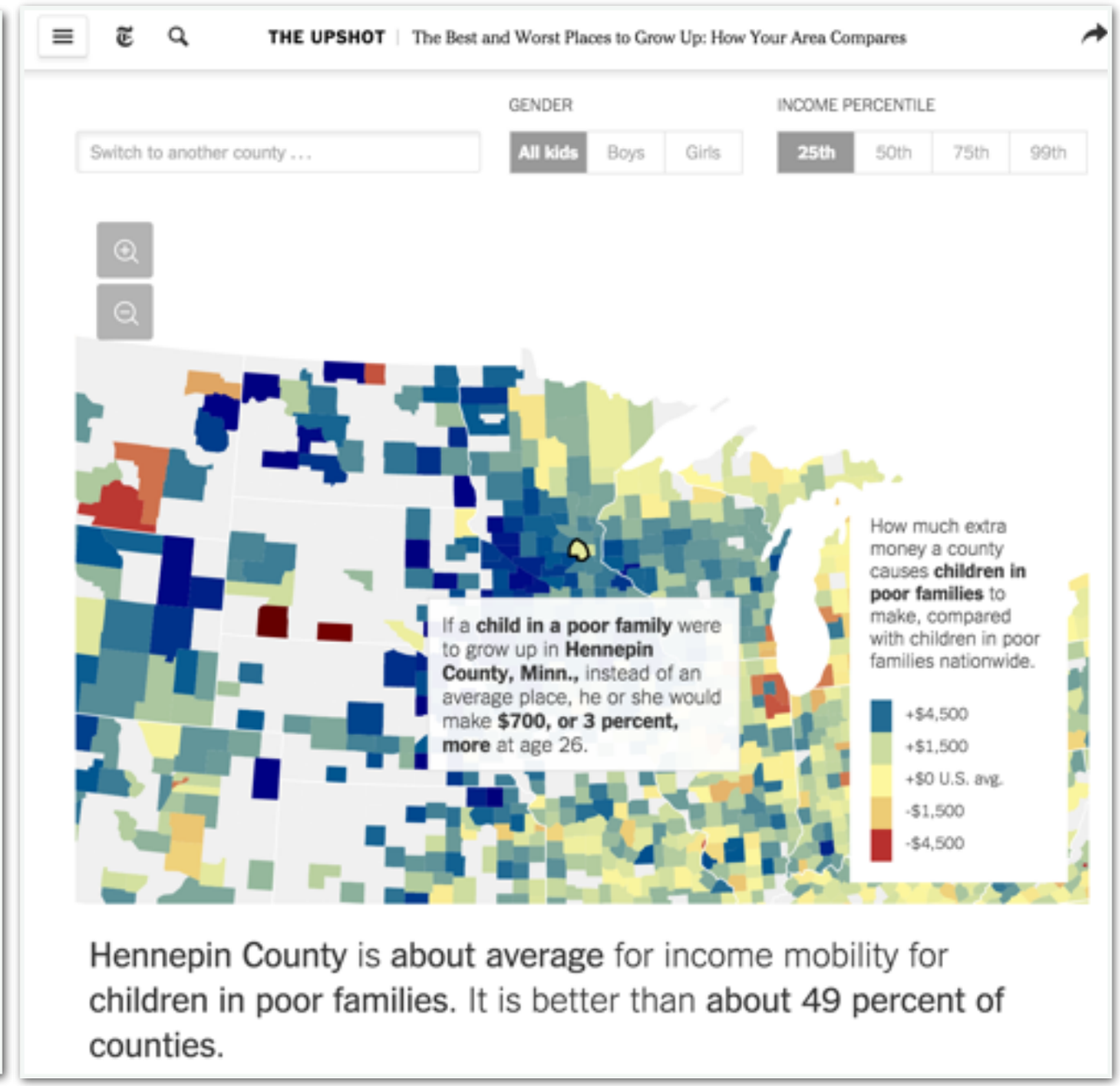
Interactive

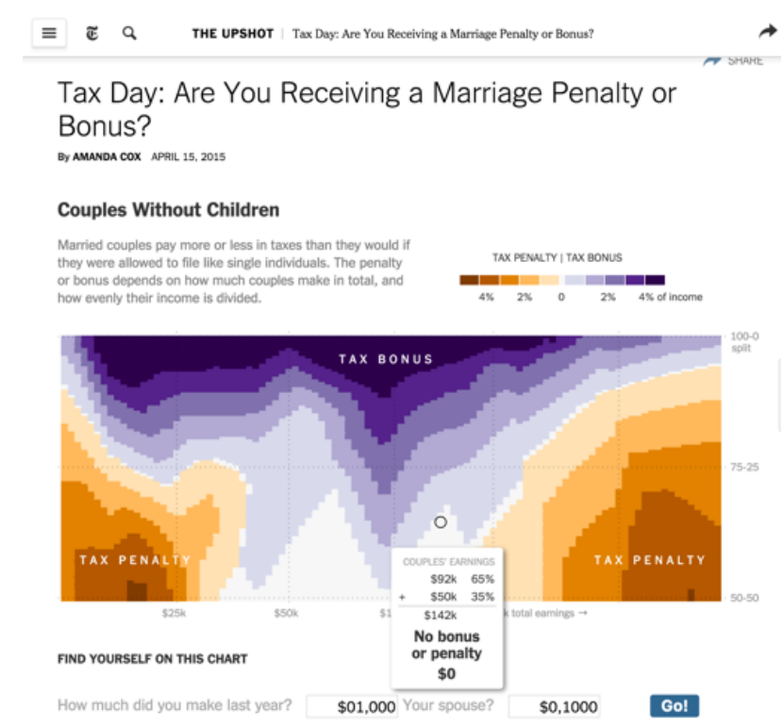


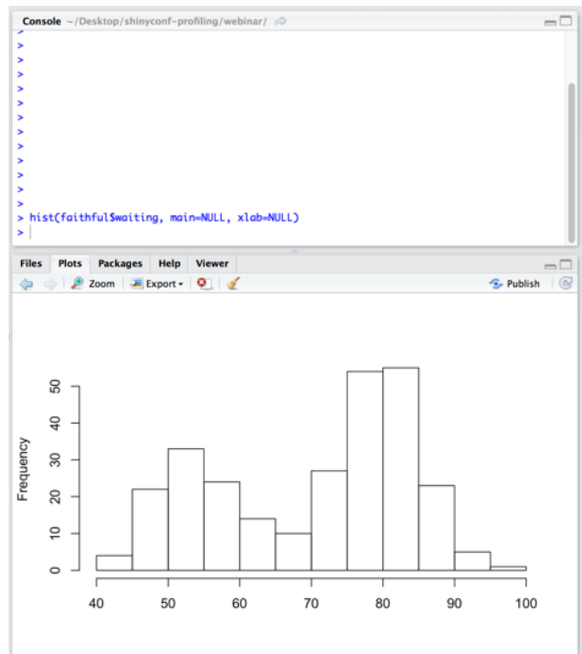
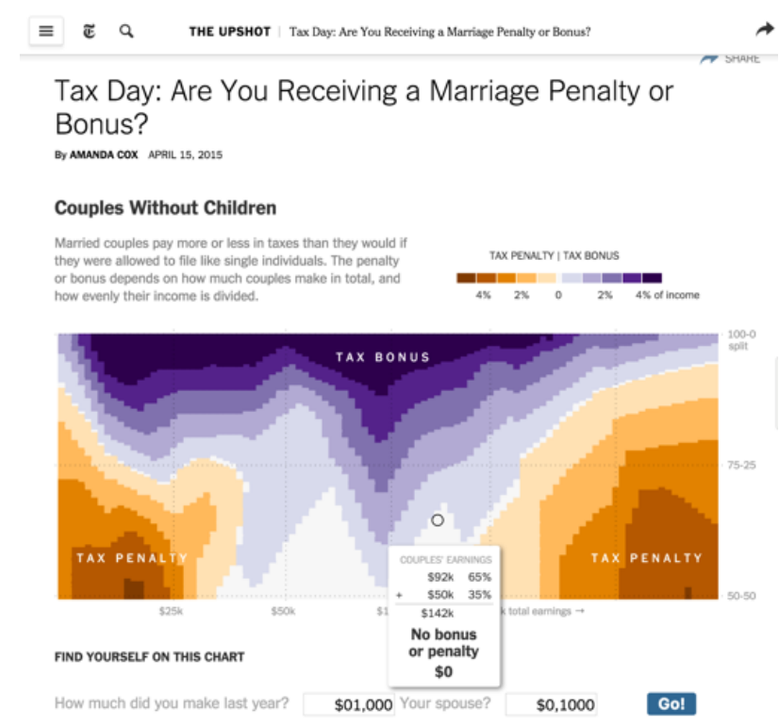
?

Interactive graphics for presentation





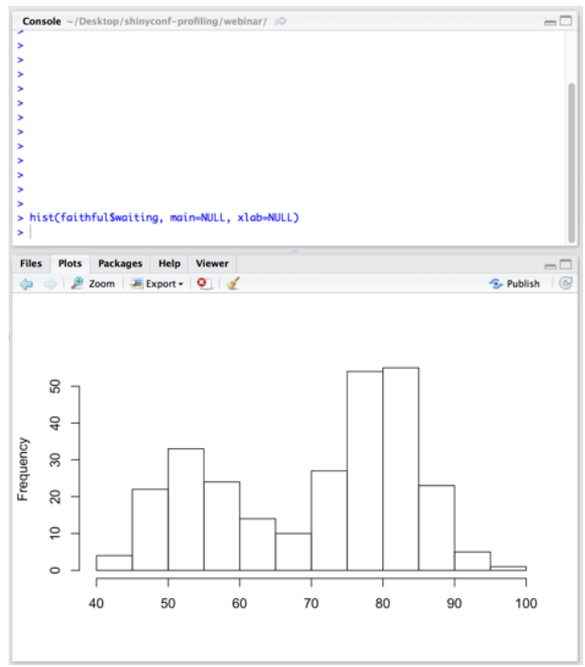
	Exploration	Presentation
Static		
Interactive		

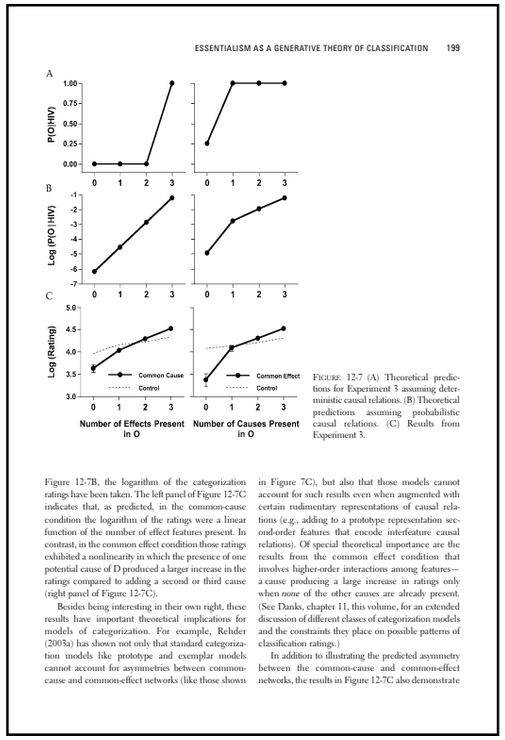
	Exploration	Presentation
Static		
Interactive		

Exploration

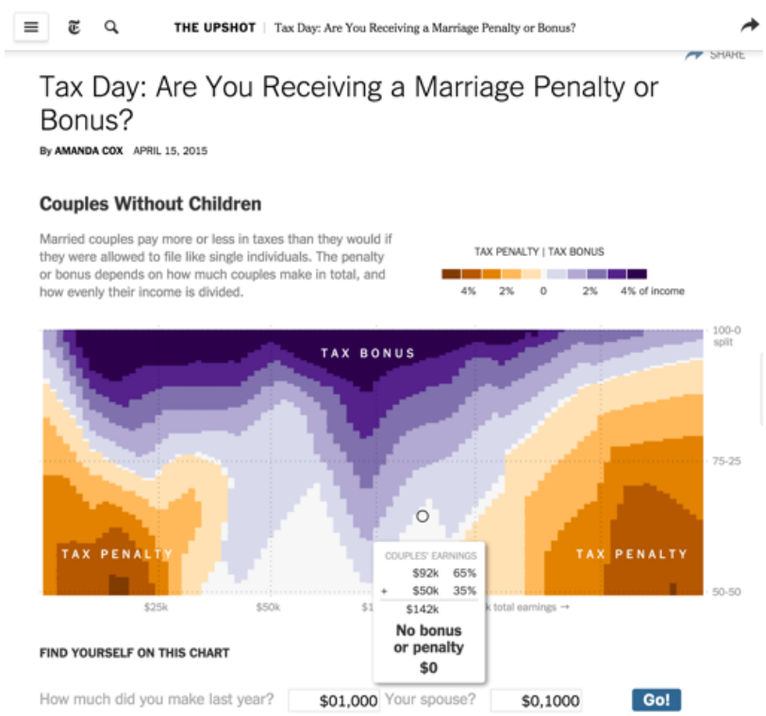
Presentation

Static





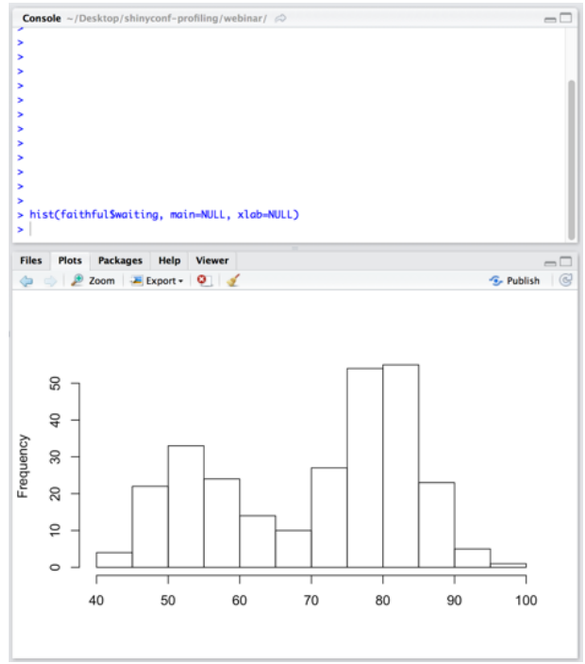
Interactive

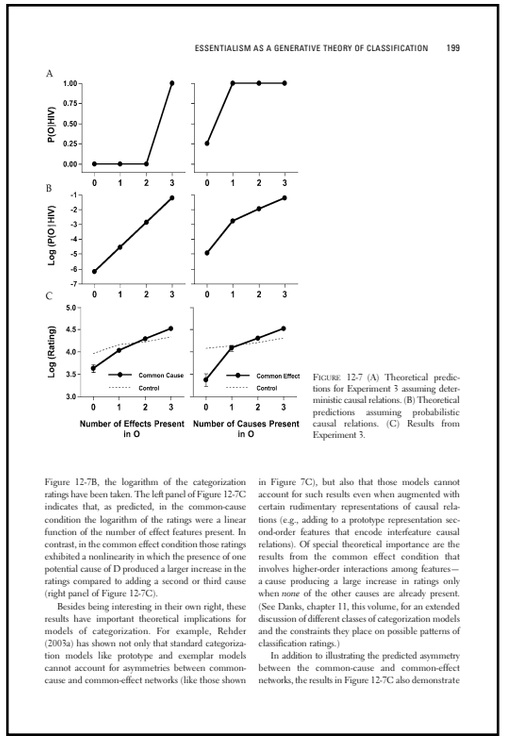


Exploration

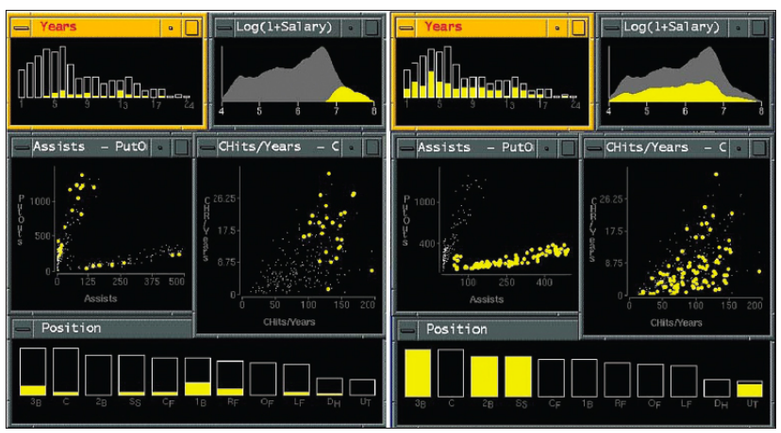
Presentation

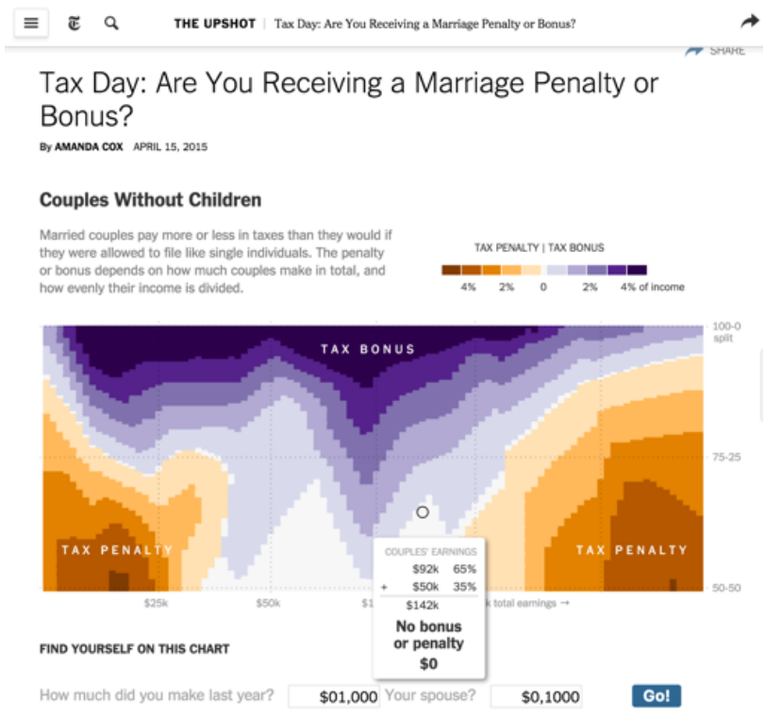
Static





Interactive





Current state of affairs for interactive graphics

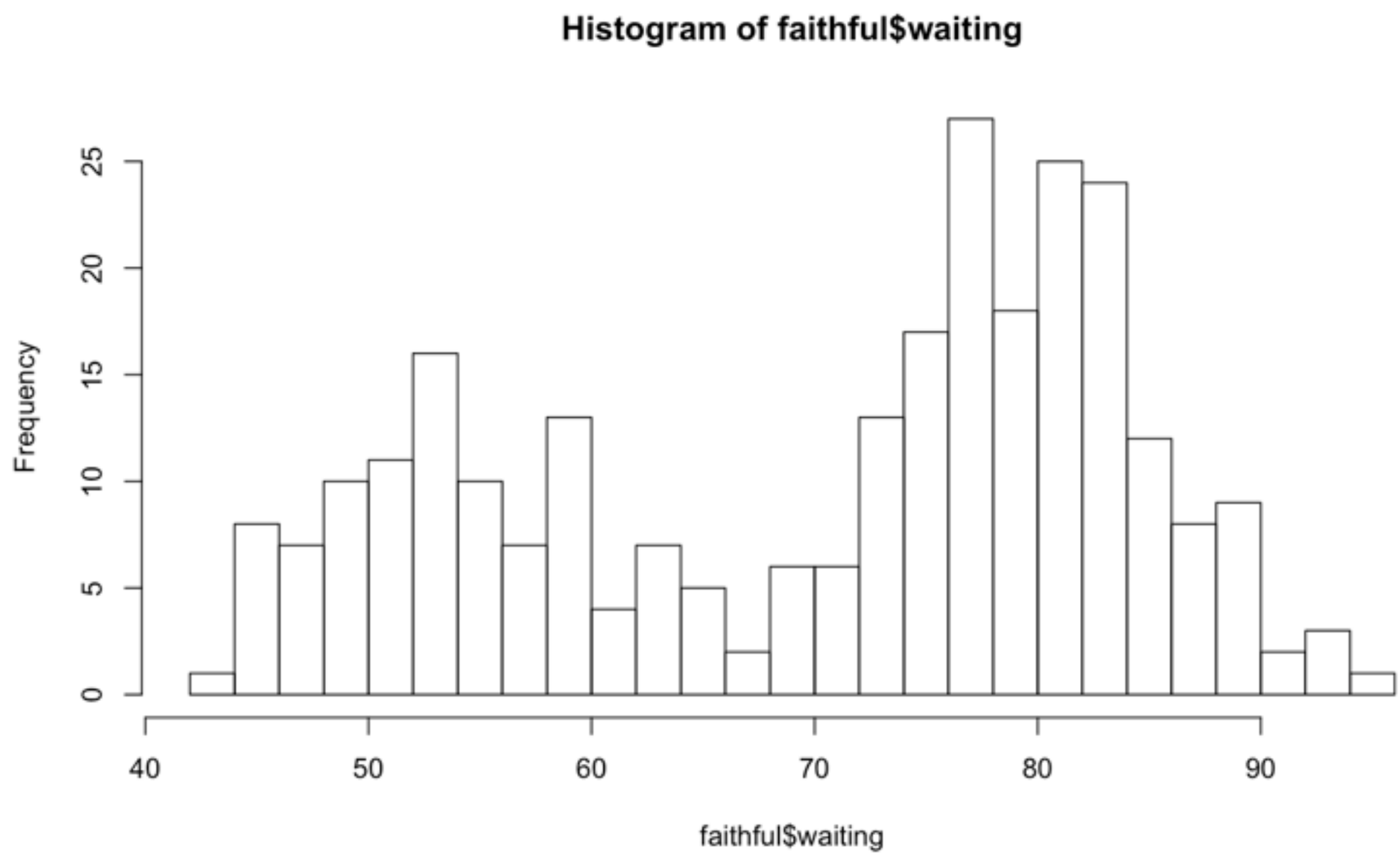
- Fast networks for sharing visualizations.
  (The Internet)

- Widespread, standardized interaction technologies.
  (Web browsers)

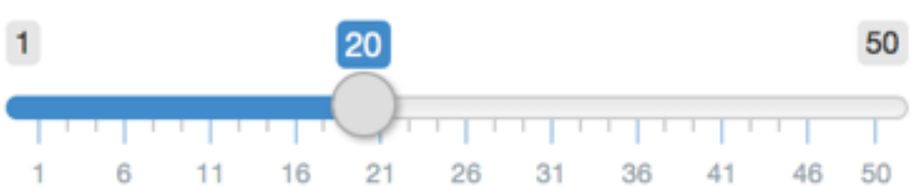
	Exploration	Presentation
Static	R*	R*
Interactive	R* + Shiny	R* + Shiny

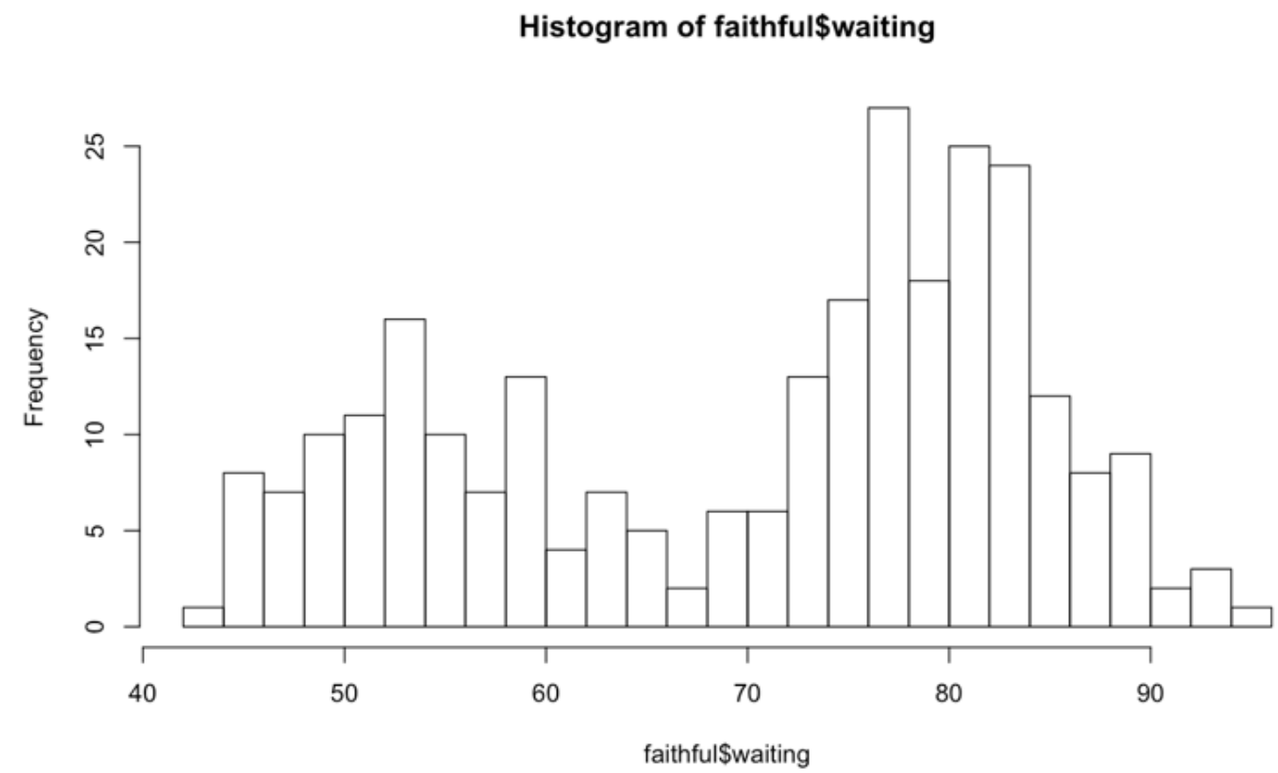
* R and packages like ggplot2 and lattice

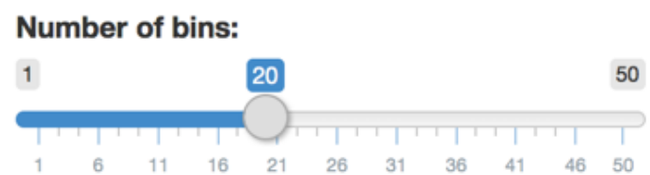
Interactive graphics with Shiny



Number of bins:








```
ui <- basicPage(
  plotOutput("plot"),
  sliderInput("bins", "Number of bins:", 1, 50, 20)
)

server <- function(input, output) {
  output$plot <- renderPlot({
    hist(faithful$waiting, breaks = input$bins)
  })
}

shinyApp(ui, server)
```

Direct interactions with plots

Clicking

```
ui <- basicPage(
  plotOutput("plot1", click = "plot_click", width = 400),
  verbatimTextOutput("info")
)

server <- function(input, output) {
  output$plot1 <- renderPlot({
    plot(mtcars$wt, mtcars$mpg)
  })

  output$info <- renderText({
    paste0("x=", input$plot_click$x, "\n",
           "y=", input$plot_click$y)
  })
}

shinyApp(ui, server)
```

Selecting nearest point(s)

```
## Server ##
output$info <- renderPrint({

 row <- nearPoints(mtcars, input$plot_click,
    xvar = "wt", yvar = "mpg",
    threshold = 5, maxpoints = 1)

  cat("Nearest point within 5 pixels:\n")
  print(row)
})
```

Selecting nearest point(s)

```
## Server ##
output$info <- renderPrint({

 row <- nearPoints(mtcars, input$plot_click,
    xvar = "wt", yvar = "mpg",
    threshold = 5, maxpoints = 1)

  cat("Nearest point within 5 pixels:\n")
  print(row)
})
```

With ggplot2, no need to pass xvar and yvar.

Adding a point

```
## Server ##
output$plot1 <- renderPlot({
  mtc <- mtcars[, c("wt", "mpg")]

  if (!is.null(input$plot_click)) {
    mtc <- rbind(
      mtc,
      data.frame(wt  = input$plot_click$x,
                 mpg = input$plot_click$y)
    )
  }

  plot(mtc$wt, mtc$mpg)
})
```

State accumulation

```
## Server ##
vals <- reactiveValues(
  mtc = mtcars[, c("wt", "mpg")]
)

observeEvent(input$plot_click, {
  vals$mtc <- rbind(
    vals$mtc,
    data.frame(wt  = input$plot_click$x,
               mpg = input$plot_click$y)
  )
})

output$plot1 <- renderPlot({
  plot(vals$mtc$wt, vals$mtc$mpg)
})
```

Returning values

```
## UI ##
actionButton("done", "Done")


## Server ##
observeEvent(input$done, {
  stopApp(vals$mtc)
})
```

Return value

```
## At console ##
value <- runApp(app)
```

Other interactions

```
## UI ##
plotOutput("plot1",
  click = "plot_click",

  dblclick = dblclickOpts(id = "plot_dblclick"),

  hover = hoverOpts(
    id = "plot_hover", delay = 500
  )
)
```

Other interactions

```
## UI ##
plotOutput("plot1",
  click = "plot_click",

  dblclick = dblclickOpts(id = "plot_dblclick"),

  hover = hoverOpts(
    id = "plot_hover", delay = 500
  )
)
```

Equivalent to:

`click = clickOpts(id = "plot_click")`

Brushing

```r
## UI ##
plotOutput("plot1",
  brush = "plot_brush"
)

## Server ##
output$info <- renderPrint({
  rows <- brushedPoints(mtcars, input$plot_brush)
  cat("Brushed points:\n")
  print(rows)
})
```

Faster responsiveness

```
## UI ##
plotOutput("plot1",
  brush = brushOpts(
    id = "plot_brush",
    delayType = "throttle",
    delay = 30
  )
)
```

Linking plots

Linked brushing

```
## UI ##
plotOutput("scatter1", brush = "brush"),
plotOutput("scatter2")


## Server ##
output$scatter1 <- renderPlot({
  ggplot(mtcars, aes(disp, hp)) +
  geom_point()
})

output$scatter2 <- renderPlot({
  brushed <- brushedPoints(mtcars, input$brush)

  ggplot(mtcars, aes(wt, mpg)) +
    geom_point() +
    geom_point(data = brushed, colour = "#4488ee")
})
```

Linked zooming

```
## UI ##
plotOutput("zoom", height = "350px"),
plotOutput("overall", height = "150px",
  brush =  brushOpts(id = "brush", direction = "x")
)

## Server ##
p <- ggplot(ss, aes(year, n)) + geom_line()

output$zoom <- renderPlot({
  if (!is.null(input$brush)) {
    p <- p + xlim(input$brush$xmin, input$brush$xmax)
  }
  p
})

output$overall <- renderPlot(p)
```

Shiny Gadgets

Shiny Gadgets

	Exploration	Presentation
Static		
Interactive		**Regular Shiny apps**

Shiny Gadgets

	Exploration	Presentation
Static		
Interactive	**Shiny Gadgets**	**Regular Shiny apps**

Shiny Gadgets

	Exploration	Presentation
Static		
Interactive	**Shiny Gadgets**	**Regular Shiny apps**

- New in Shiny 0.13.0 (released January)

Shiny Gadgets

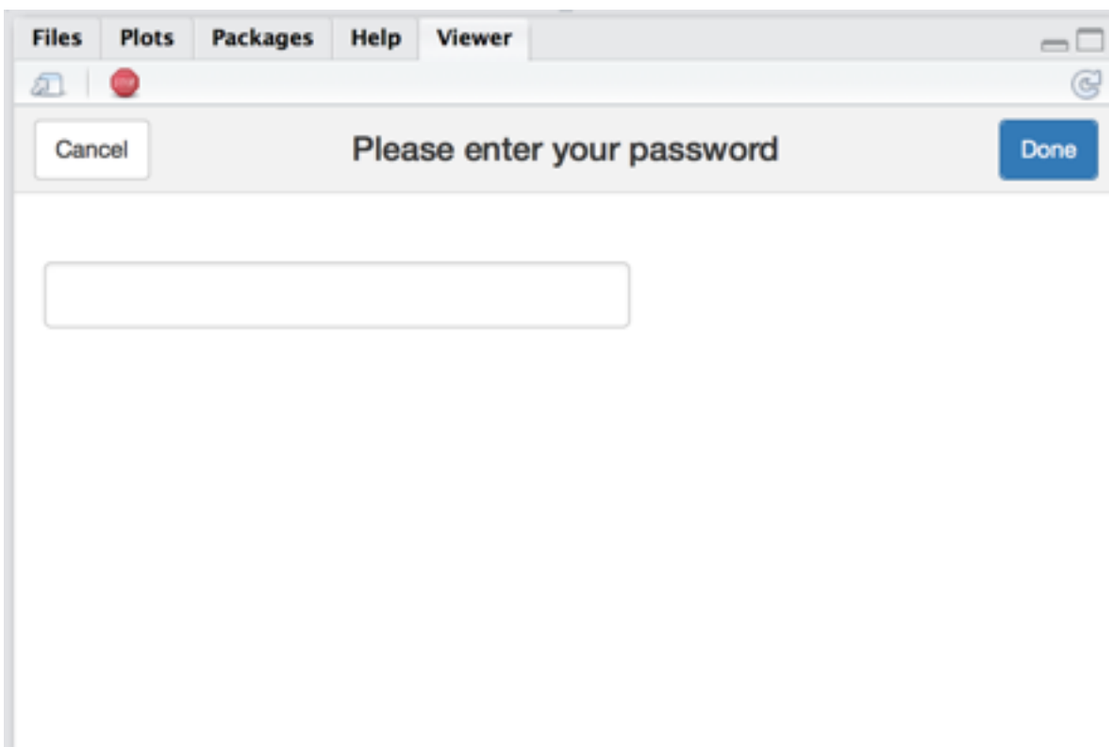
	Exploration	Presentation
Static		
Interactive	**Shiny Gadgets**	**Regular Shiny apps**

- New in Shiny 0.13.0 (released January)
- Gadgets make it easy to use interactive graphics in the exploration phase of data analysis.

Gadgets: three main differences

```
get_password()
```

```
library(miniUI)
```



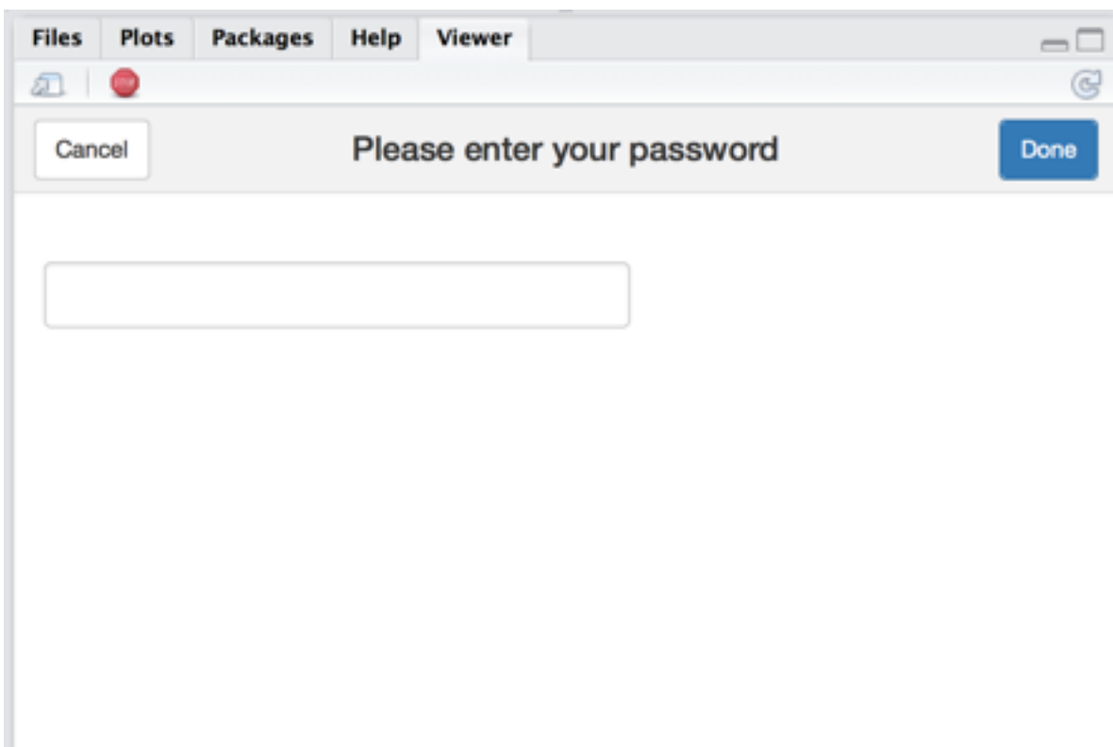

```
stopApp()
```

Gadgets: three main differences

Invoked from a function

```
get_password()
```

```
library(miniUI)
```




```
stopApp()
```

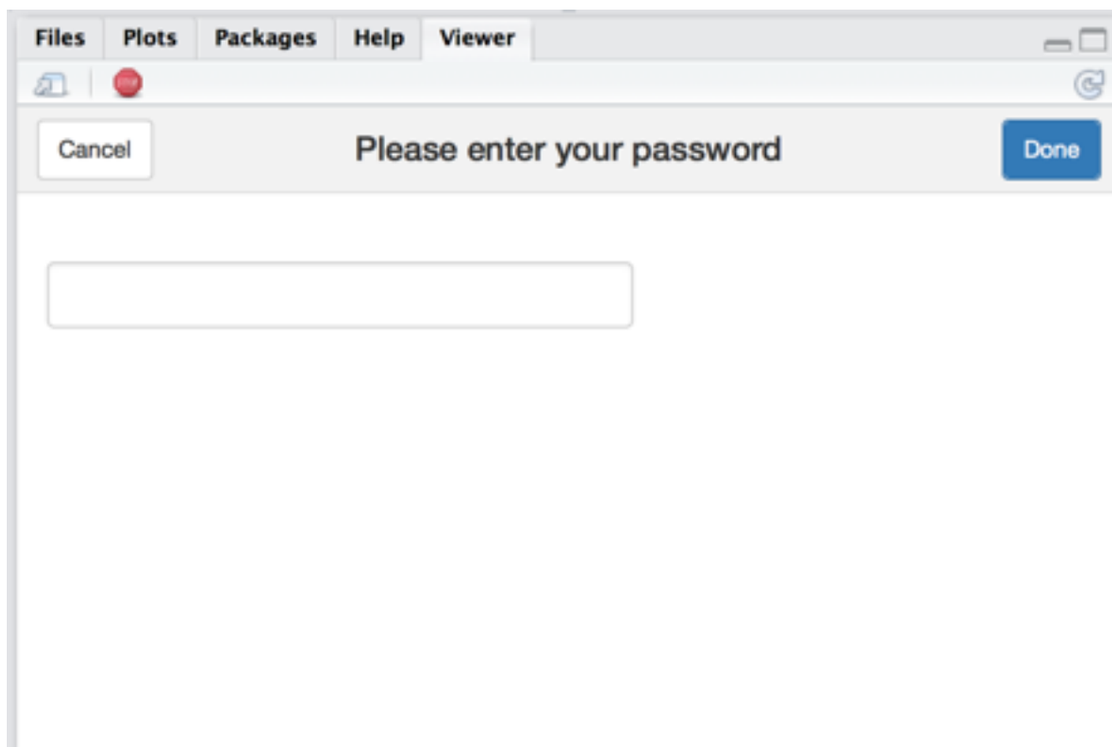
Gadgets: three main differences

Invoked from a function

```
get_password()
```

Uses miniUI for layout

```
library(miniUI)
```




```
stopApp()
```

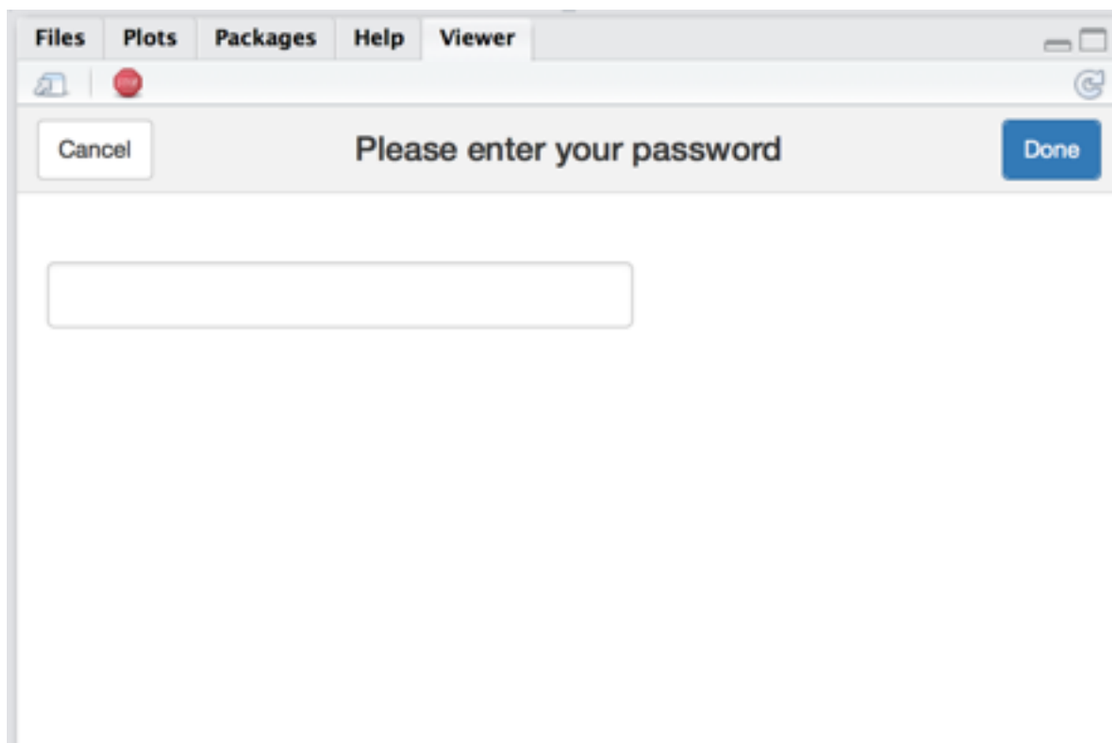
Gadgets: three main differences

Invoked from a function

```
get_password()
```

Uses miniUI for layout

```
library(miniUI)
```



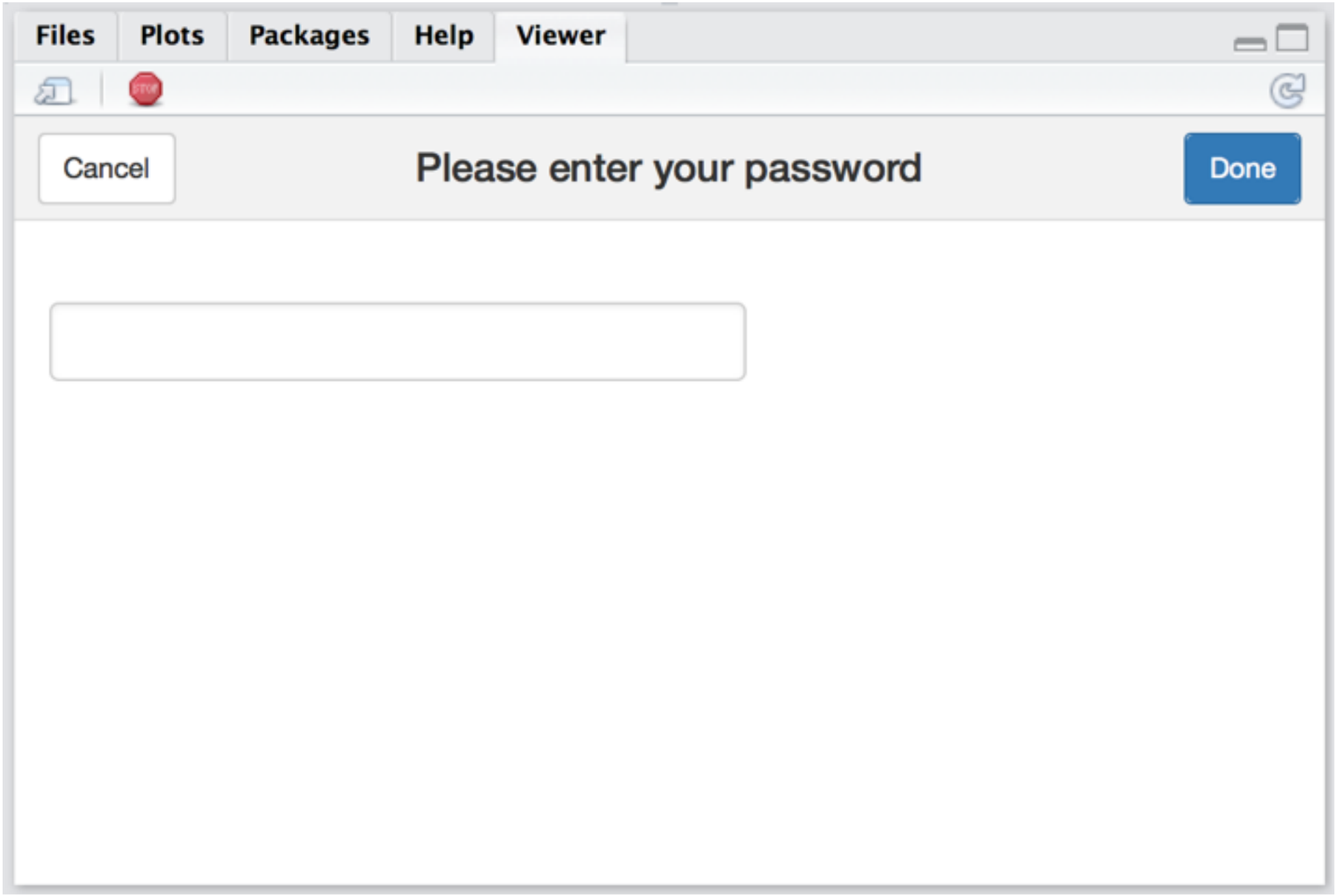
Stops and returns a value

```
stopApp()
```

A very simple gadget

```
get_password <- function() {
  library(miniUI)
  ui <- miniPage(
    gadgetTitleBar("Please enter your password"),
    miniContentPanel(
      passwordInput("password", "")
    )
  )

  server <- function(input, output) {
    observeEvent(input$done, {
      stopApp(input$password)
    })
    observeEvent(input$cancel, {
      stopApp(stop("No password.", call. = FALSE))
    })
  }

  runGadget(ui, server)
}

get_password()
```

Captures password without recording it in command history


Files
Plots
Packages
Help
Viewer
Cancel
Please enter your password
Done

A very simple gadget

```
get_password <- function() {
  library(miniUI)
  ui <- miniPage(
    gadgetTitleBar("Please enter your password"),
    miniContentPanel(
      passwordInput("password", "")
    )
  )

  server <- function(input, output) {
    observeEvent(input$done, {
      stopApp(input$password)
    })
    observeEvent(input$cancel, {
      stopApp(stop("No password.", call. = FALSE))
    })
  }

  runGadget(ui, server)
}

get_password()
```

A very simple gadget

```
get_password <- function() {
  library(miniUI)
  ui <- miniPage(
    gadgetTitleBar("Please enter your password"),
    miniContentPanel(
      passwordInput("password", "")
    )
  )

  server <- function(input, output) {
    observeEvent(input$done, {
      stopApp(input$password)
    })
    observeEvent(input$cancel, {
      stopApp(stop("No password.", call. = FALSE))
    })
  }

  runGadget(ui, server)
}

get_password()
```

Invoked from a function

A very simple gadget

```
get_password <- function() {
  library(miniUI)
  ui <- miniPage(
    gadgetTitleBar("Ple
    miniContentPanel(
      passwordInput("pa
    )
  )

  server <- function(input, output) {
    observeEvent(input$done, {
      stopApp(input$password)
    })
    observeEvent(input$cancel, {
      stopApp(stop("No password.", call. = FALSE))
    })
  }

  runGadget(ui, server)
}

get_password()
```

miniUI package has UI customized for small spaces

Invoked from a function

A very simple gadget

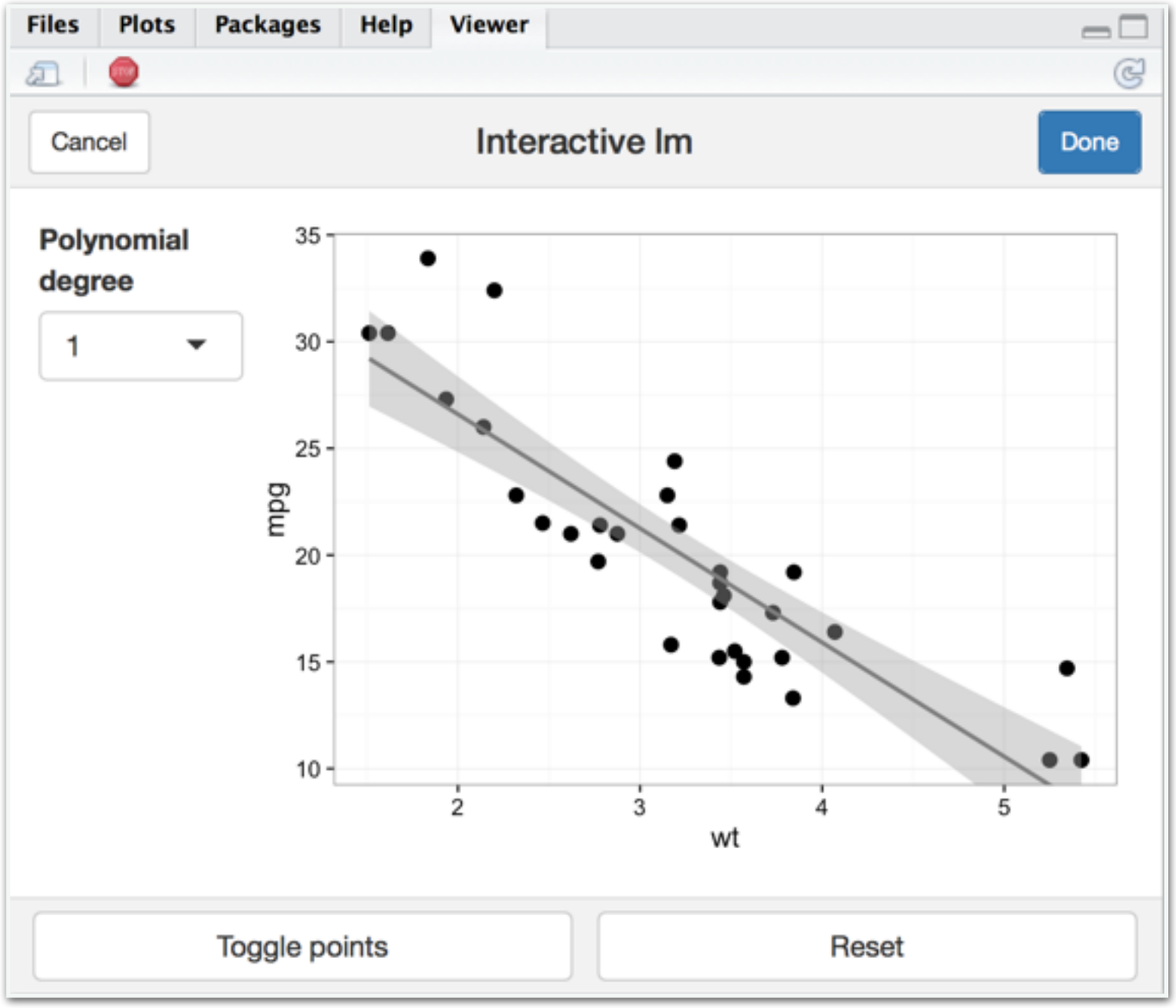
```
get_password <- function() {
  library(miniUI)
  ui <- miniPage(
    gadgetTitleBar("Ple
    miniContentPanel(
      passwordInput("pa
    )
  )

  server <- function(input, output) {
    observeEvent(input$done, {
      stopApp(input$password)
    })
    observeEvent(input$cancel, {
      stopApp(stop("No password.", call. = FALSE))
    })
  }

  runGadget(ui, server)
}

get_password()
```

miniUI package has UI customized for small spaces

Stops and returns a value

Invoked from a function

Example with interactive graphics

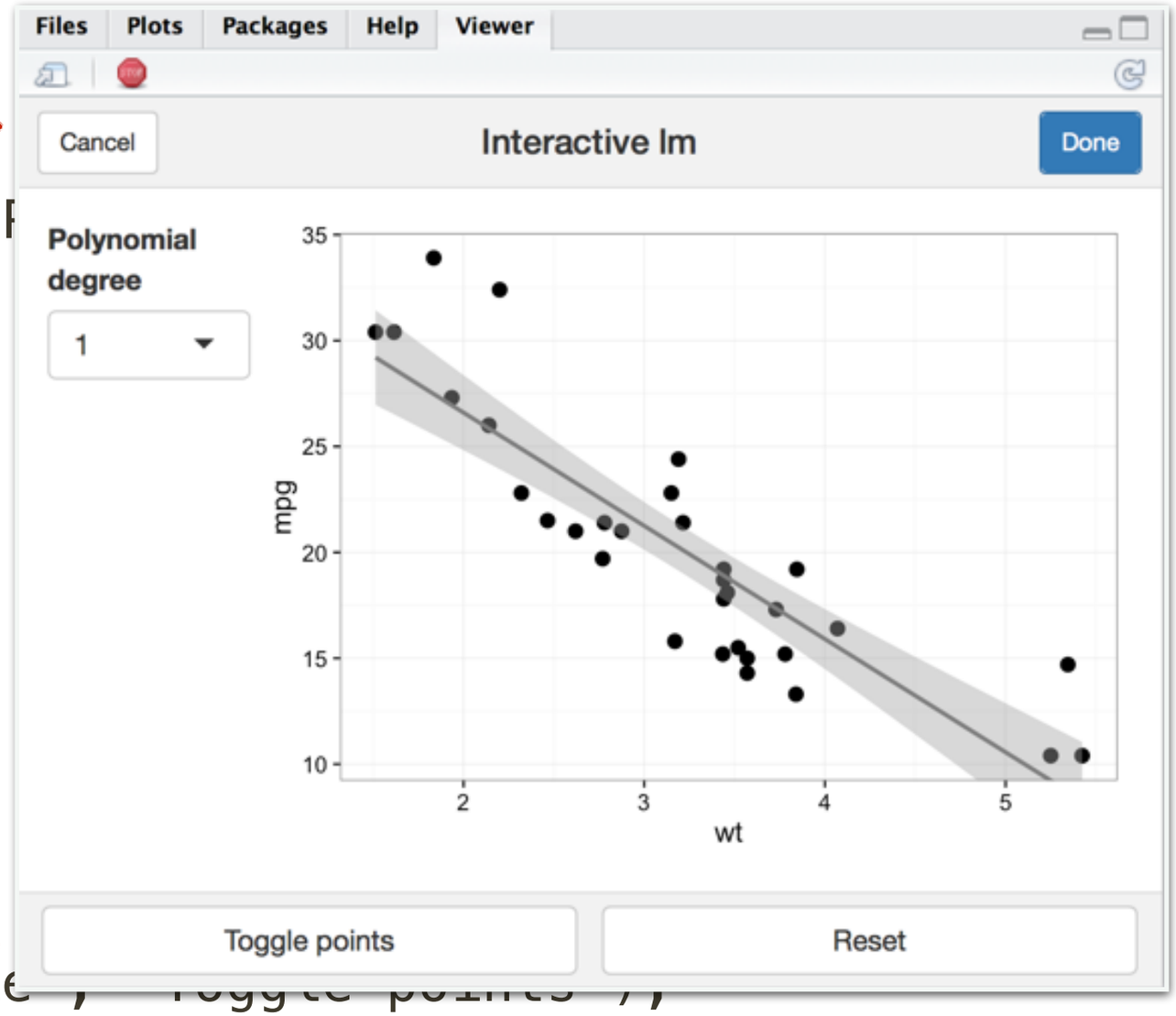

Files
Plots
Packages
Help
Viewer
Cancel
Interactive lm
Done
Polynomial degree
1
35
30
25
20
15
10
mpg
2
3
4
5
wt
Toggle points
Reset

miniUI for layout

```
ui <- miniPage(
  gadgetTitleBar("Interactive lm"),
  miniContentPanel(
    fillRow(flex = c(NA, 1),
      fillCol(width = "100px",
        selectInput("degree", "Polynomial degree", c(1, 2, 3, 4))
      ),
      plotOutput("plot1",
        height = "100%",
        click = "plot1_click",
        brush = brushOpts(
          id = "plot1_brush"
        )
      )
    )
  ),
  miniButtonBlock(
    actionButton("exclude_toggle", "Toggle points"),
    actionButton("exclude_reset", "Reset")
  )
)
```

miniUI for layout

```
ui <- miniPage(
  gadgetTitleBar("Interactive lm"),
  miniContentPanel(
    fillRow(flex = c(NA, 1),
      fillCol(width = "100px",
        selectInput("degree", "P
      ),
      plotOutput("plot1",
        height = "100%",
        click = "plot1_click",
        brush = brushOpts(
          id = "plot1_brush"
        )
      )
    )
  ),
  miniButtonBlock(
    actionButton("exclude_toggle", "Toggle points"),
    actionButton("exclude_reset", "Reset")
  )
)
```



Wrapping up

http://shiny.rstudio.com/articles/

=> Interactive plots

=> Shiny Gadgets

http://shiny.rstudio.com/gallery/

=> Interactive plots