MET CS688

WEB ANALYTICS AND MINING

ZLATKO VASILKOSKI

USING SHINY

Using package shiny

- Shiny brings R to the Web and combines the computational power of R with the interactivity of the modern web.
- Shiny is an R package that makes it easy to build interactive web applications (apps) and visualizations straight from R.

  ```
  install.packages("shiny")
  ```
- To use it, not that much web development skills are needed.
- Shiny apps have two components:
  - user-interface script (ui.R) - controls the layout and appearance of the app.
  - server script (server.R) - contains the instructions that the computer needs to build the app.
- Shiny app are created simply by making a new directory and saving a *ui.R* and *server.R* file inside it.
- Every Shiny app has the same structure: two R scripts saved together in a directory.
- The user-interface (ui) script controls the layout and appearance of the app. It is defined in a source script named *ui.R*.
- The *server.R* script contains the instructions that the computer needs to build the app.
- Note that each app will need its own unique directory.

Using package shiny

- The Shiny apps collect a value from the user through widgets (web elements)
  - Widget a way for users to send messages to the Shiny app.
  - The widgets are added to a web page in the same way that another types of HTML content

- Another thing to notice is that the Shiny apps automatically responds to user changes in the widgets. Here is the Shiny Widgets Gallery

  http://shiny.rstudio.com/gallery/widget-gallery.html

- The reactive output is achieved by adding an R object to the user-interface with *ui.R* and calling the widget value of the R object in *server.R*
- In addition to the two regular scripts *ui.*R and *server.R* you can include other scripts too.
- You would reference them in *server.R* as

  source("MyRscript.R")

Using Shiny R package

- Shiny is an R package that makes it easy to build interactive web applications (apps) straight from R.
- Shiny apps have two components (scripts):
  - user-interface script (ui.R) - controls the layout and appearance of the app.
  - server script (server.R) - contains the instructions that the computer needs to build the app.
- Shiny app are created by making a **new directory** and saving a *ui.R* and *server.R* file inside it.

- The Shiny apps collect a value from the user through widgets (web elements)
  - Widget a way for users to send messages to the Shiny app.
  - The widgets are added to a web page in the same way that another types of HTML content

Example 5: Shiny Quotes Search App

Shiny app that searches for stock quote query specified by the user.

Note:

- The default stock symbol is “AAPL”
- Another query can be typed in.

This app uses the single line of code to retrieve a quote

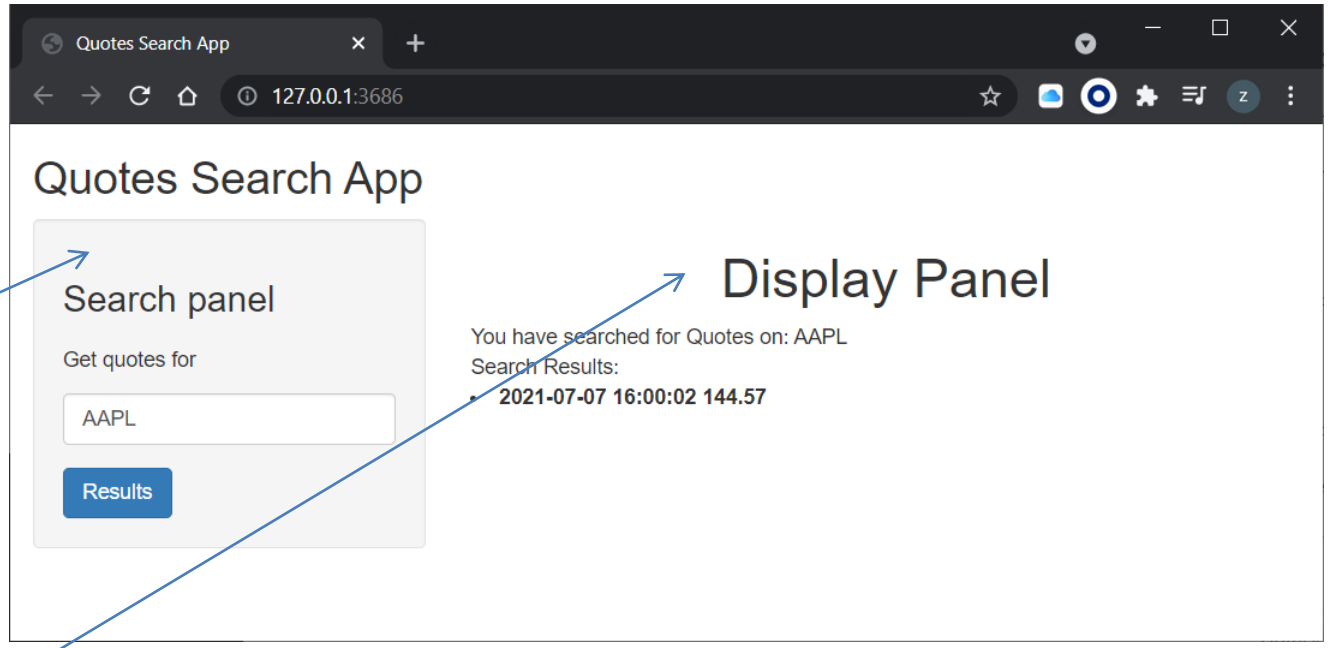
```
result <- getQuote(input$text.Search, what=yahooQF("Last Trade (Price Only)"))
```

The search panel (on the left) contains two fields:

- One to enter the stock symbol
- A button to submit the stock symbol.

The search results are displayed in the display panel (on the right).

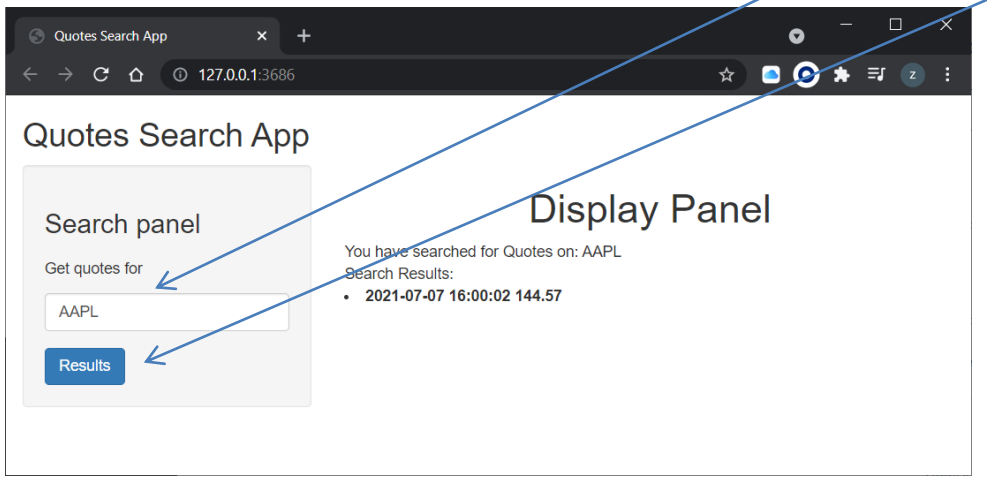
The app responds automatically to user’s changes in the widgets.



Example 5: Shiny Quotes Search App

Note:

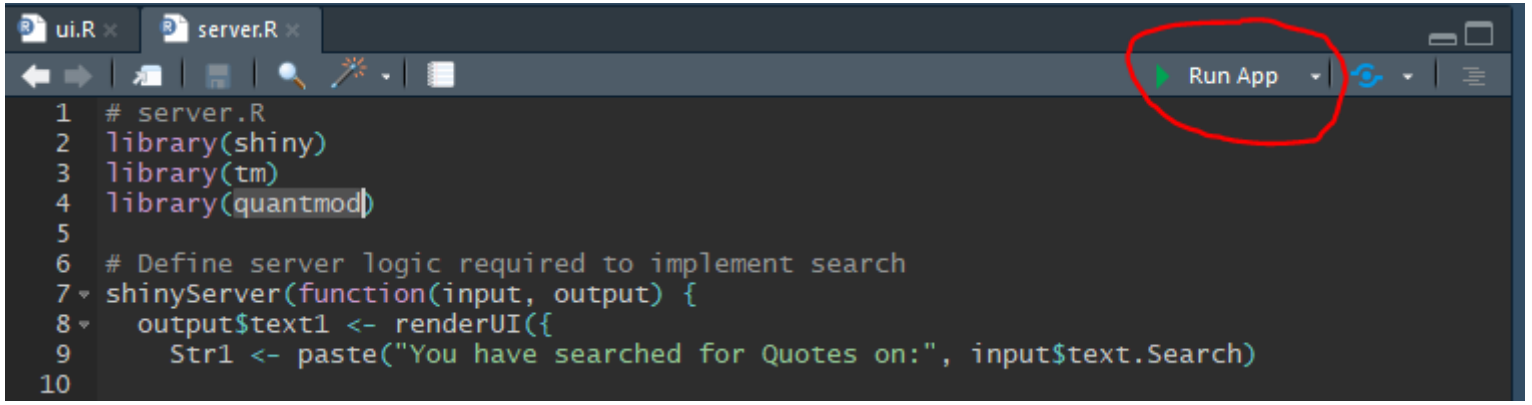
- Widget **textInput()** creates a field to enter text.
- Widget **submitButton()** gets the quotes.




```
# Example: Shiny Quotes Search App
# ui.R
library(shiny)

# Define UI for application
shinyUI(fluidPage(
 titlePanel(" Quotes Search App "),   # Application title (Panel 1)

 sidebarLayout(                                    # Widget (Panel 2)
  sidebarPanel(h3("Search panel"),
          # Search for
          textInput("text.Search", label = h5(" Get quotes for"),
                  value = "AAPL"),
          # Start Search
          submitButton("Results")
 ),
 # Display Panel (Panel 3)
  mainPanel(
   h1("Display Panel",align = "center"),
   htmlOutput("text1")
  )
 )
))
```

Example 5: Shiny Quotes Search App

- Note: You run a Shiny app with **runApp(**app's folder**)**
  - The "Shiny Quotes Search App" example line is in bold.
  - Note the WebSource is referenced (**input$text.Search**), which is specified in the user-interface script (*ui.R*)
- The rest of the code just displays the first heading of the result in an HTML format.

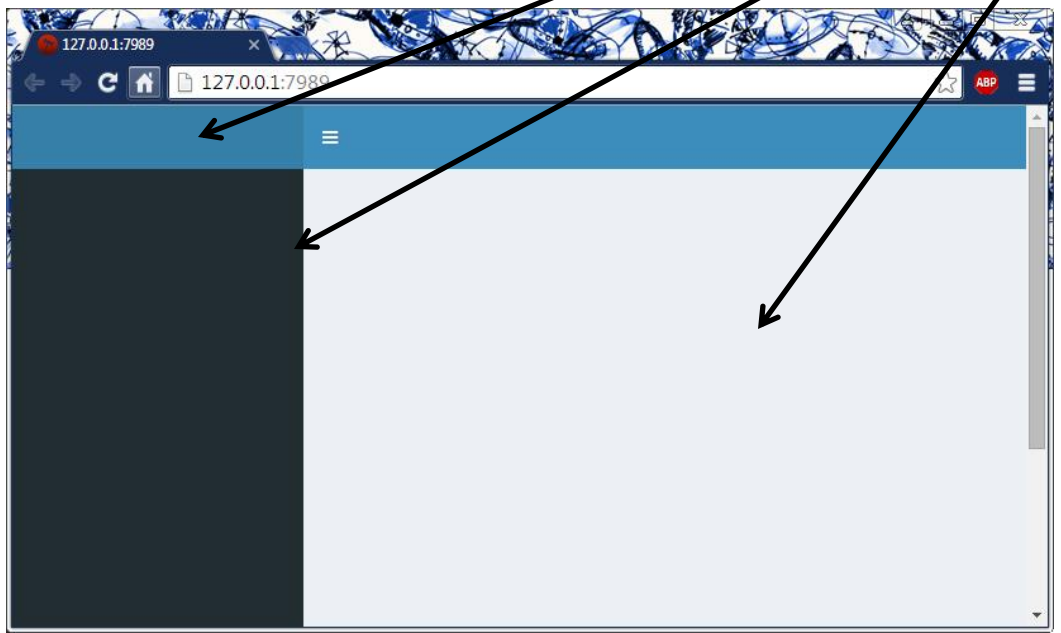



```
# Example: Shiny Quotes Search App
# server.R
library(shiny)
library(tm)
library(quantmod)
# Define server logic required to implement search
shinyServer(function(input, output) {
  output$text1 <- renderUI({
    Str1 <- paste(" You have searched for Quotes on:", input$text.Search)
    result <- getQuote(input$text.Search, what=yahooQF("Last Trade (Price Only)"))
    dataOutput <- paste("<li>",strong( result),"</li>") # Get the first result
    Str2 <- "Search Results:"
    HTML(paste(Str1, Str2, dataOutput, sep = '<br/>'))
  })
})
```

Shinydashboard

Similarly Shinydashboard also contains 3 panels

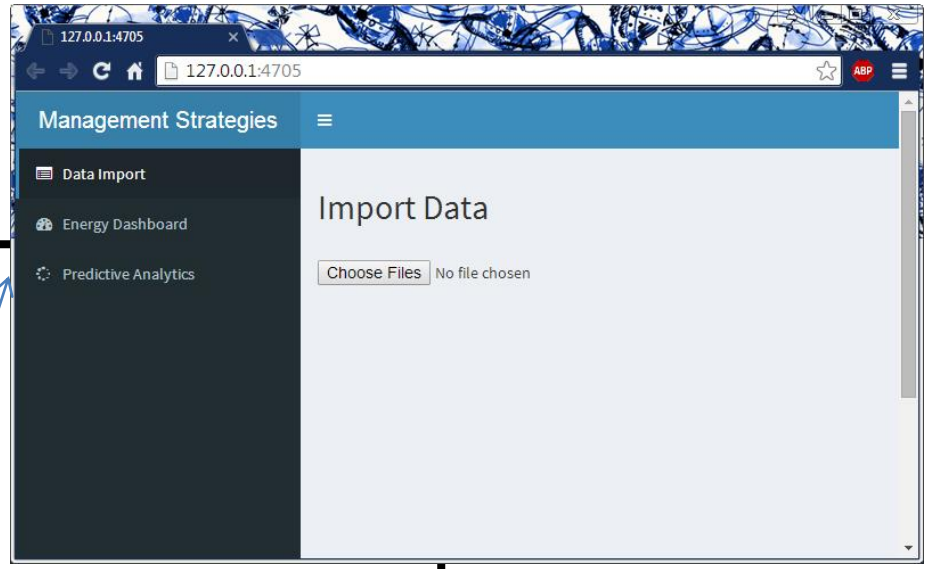
- dashboardHeader
- dashboardSidebar
- dashboardBody




```
# Single script app
# Shinydashboard Example
## app.R ##
library(shiny)
library(shinydashboard)

ui <- dashboardPage(
  dashboardHeader(),
  dashboardSidebar(),
  dashboardBody()
)

server <- function(input, output) { }


shinyApp(ui, server)
```

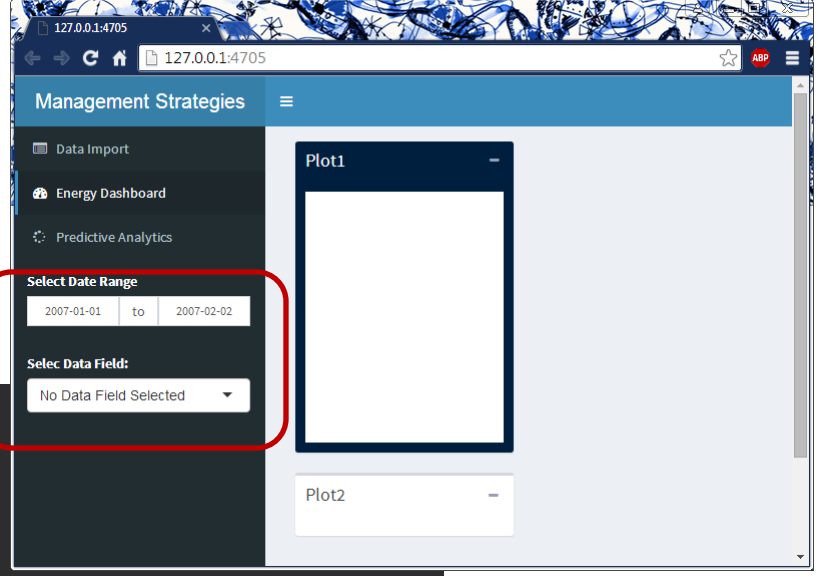
Building your first Shinydashboard

- In a similar fashion to Tabs, Menu items are added to dashboardSidebar().
- The header is added with




```
## app.R ##
library(shiny)
library(shinydashboard)
ui <- dashboardPage(
dashboardHeader(title = "Management Strategies", titleWidth = 250,
                dropdownMenuOutput("messageMenu")
 ),

 dashboardSidebar(
   width = 250,
 dashboardSidebar(width = 250,
                  sidebarMenu(id="menu",
                    menuItem("Data Import", tabName = "readdata", icon = icon("list-alt")),
                    menuItem("Energy Dashboard", tabName = "dashboard", icon = icon("dashboard")),
                    menuItem("Predictive Analytics", tabName = "analytics", icon = icon("spinner"))
                  )
 ),
 dashboardBody())
)
server <- function(input, output) { }
shinyApp(ui, server)
```

- A conditional panel can be created depending on the selected menu, in this case when menu with tabName “dashboard” is selected two other widgets are displayed




```
1  library(shiny)
2  library(shinydashboard)
3  library(googleVis)
4
5
6  ui <- dashboardPage(
7    dashboardHeader(title = "Management Strategies", titleWidth = 250,
8                    dropdownMenuOutput("messageMenu")
9    ),
10
11   dashboardSidebar(
12     width = 250,
13     sidebarMenu(id="menu",
14       menuItem("Data Import", tabName = "readdata", icon = icon("list-alt")),
15       menuItem("Energy Dashboard", tabName = "dashboard", icon = icon("dashboard")),
16       menuItem("Predictive Analytics", tabName = "analytics", icon = icon("spinner"))
17     ),
18     conditionalPanel(
19       condition = "input.menu == 'dashboard'",
20       dateRangeInput('dateRange',
21                      label = 'Select Date Range',
22                      start = "2007-01-01", end = "2007-02-02"),
23       selectizeInput("selectedData", ("Selec Data Field:"),
24                      options = list(dropdownParent = 'body'),
25                      choices = c("No Data Field Selected"))
26     )
27
28   ),
29   dashboardBody(
```

- The content of the dashboardBody() for each menu name is created here. Note that menu1 "readdata“ contains fileInput() widget while menu2 “dashboard” contains 2 plots.

```
29    dashboardBody(
30      tabItems(
31        # 1 First tab content
32        tabItem(tabName = "readdata",
33                h2("Import Data"),
34                fileInput('fileIn', '', multiple=T, accept=c('application/txt')) # Data Loading
35        ),
36
37        # 2 Second tab content
38        tabItem(tabName = "dashboard",
39                fluidRow(
40                  column(width = 12,
41                  box( title = "Plot1", background = "navy", collapsible = TRUE,
42                       plotOutput("plot1", height = 250)
43                  )),
44                  column(width = 12,
45                  box( title = "Plot2", collapsible = TRUE, #widht = 900,
46                    htmlOutput("plot2", height = 250)
47                  ))
48                )
49        ),
50
51        # 3 Tab content
52        tabItem(tabName = "analytics",
53                h2("Machine Learning Tools")
54        )
55      )
56    )
57  )
58
59  server <- function(input, output, session) {
```

The “server” function contains most of the code that does the work.

```
server <- function(input, output, session) {
  source("DataPreprocessing.R") # Call to script
  options(shiny.maxRequestSize=130*1024^2) # Increase Shiny upload maximum of 130Mb


  # Initialize reactiveValues
   values <- reactiveValues(EnergyData = data.frame(),
                            Selected.Field.Indx = 1,
                            Selected.Field.Data = 1,
                            File.Data = 1
                            ) # Set init Reactive Values

  observe({

    if (is.null(input$fileIn)) # Anything below does not execute until file is selected
      return(NULL)
    inFile <- input$fileIn
    fileNames <- inFile$name

    print(input$dateRange)


    load(inFile$datapath)
    power$Date <- as.Date(power$Date, format="%d/%m/%Y")
    power <- power[complete.cases(power),] # remove NA

    FROM <- 2; TO <- 1;
    updateSelectizeInput(session, "selectedData",  choices = head(tail(names(power),n=-FROM),-TO))

    temp <- Data_Preprocessing(power,c(power$Date[1],tail(power$Date,1)))
    isolate({ values$EnergyData <- temp })

  })

```

Run R script & Specify data upload size

Set global variables used in server.

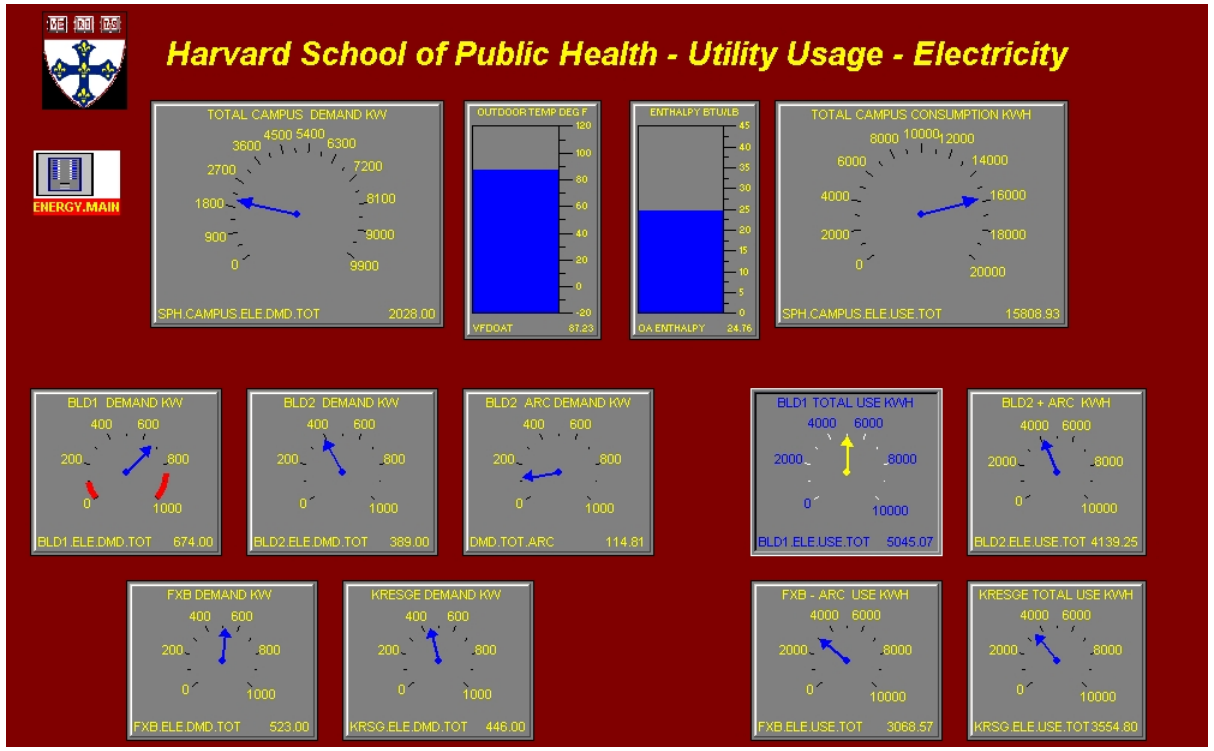
Used for Input/Output immediate execution on change such as file load.

Populates with data the “dashboard” menu widget ”selectedData”.

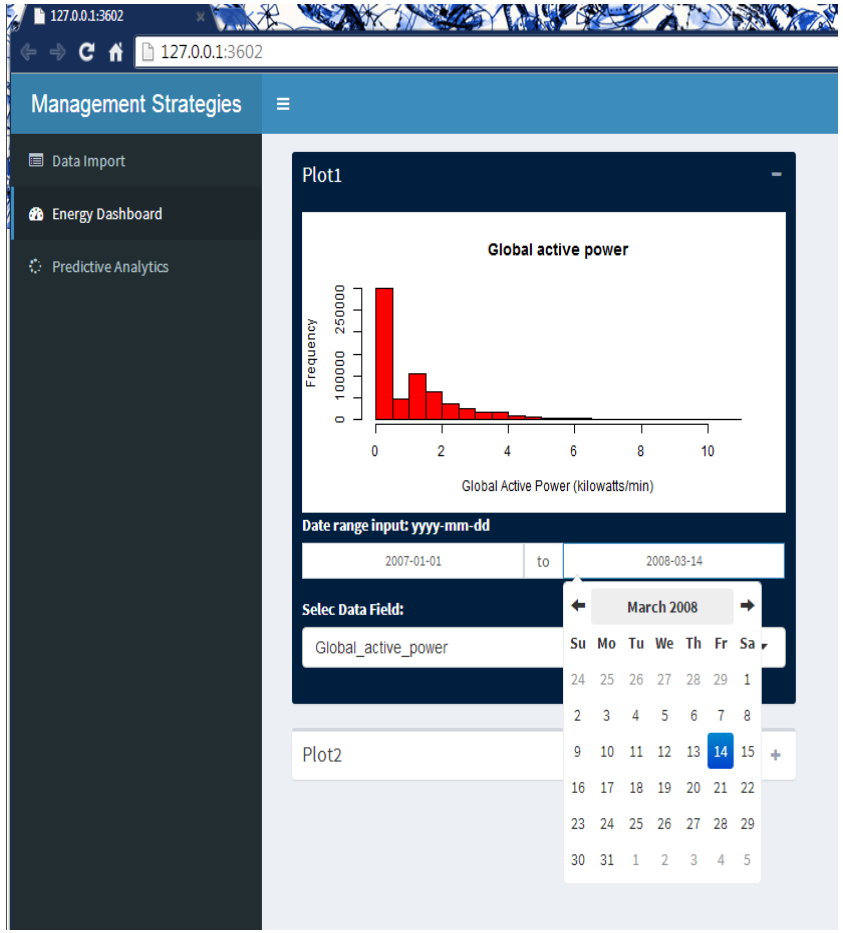
Populate with data the global server variable “EnergyData”. Note how “isolate({})” is used to change its value, otherwise the variable value is used as “values$EnergyData”.

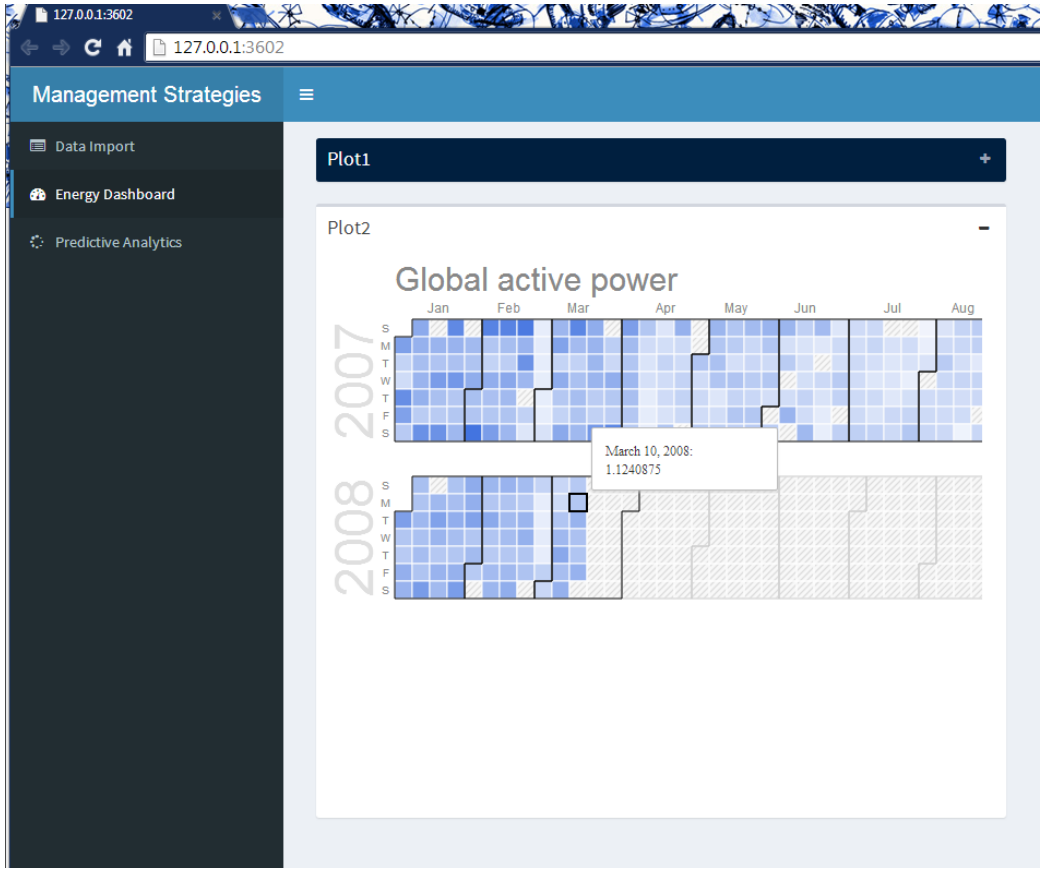
Case Study

- This is the application interface (as of 2015) that Harvard School of Public Health and Boston College are using for energy management.



More user friendly Energy Management Dashboard can be easily created and relevant data visualized with Shiny Dashboard and Google visualization package using all of the statistical and computational power of R.





WikiSearch – Exercise/Assignment

- Task: Rank how close are the several selected Wikipedia pages
  - Use the “Web Mining” code posted on Blackboard to create a Web app to
    - Download several Wikipedia pages, analyze the text content and
    - Cluster (find similarity) between them.

- Note the similarity depends not that much on the most common words to all documents but on the most common words in a particular **subset** of document.

- As in any ML task (classification, clustering, etc.) figuring out which data features are common to a particular subset is the most important aspect.

- Profiling the data (visually) is always necessary.

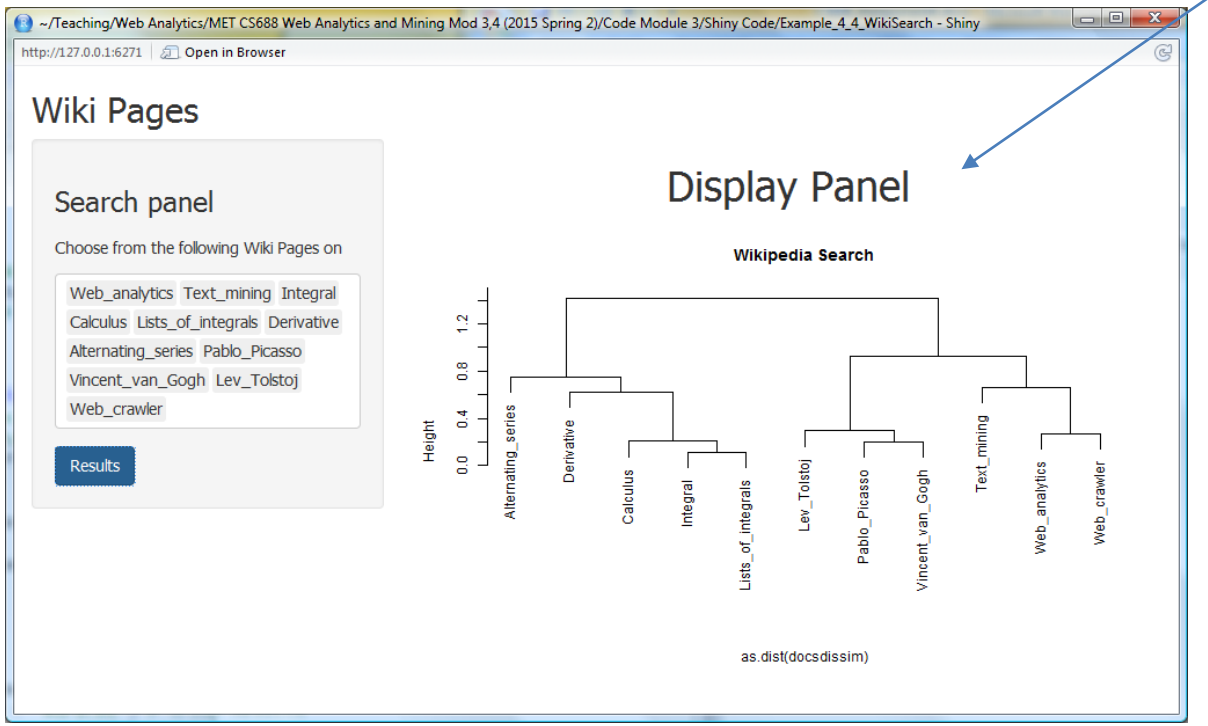
- Follow WikiSearch assignment instructions.

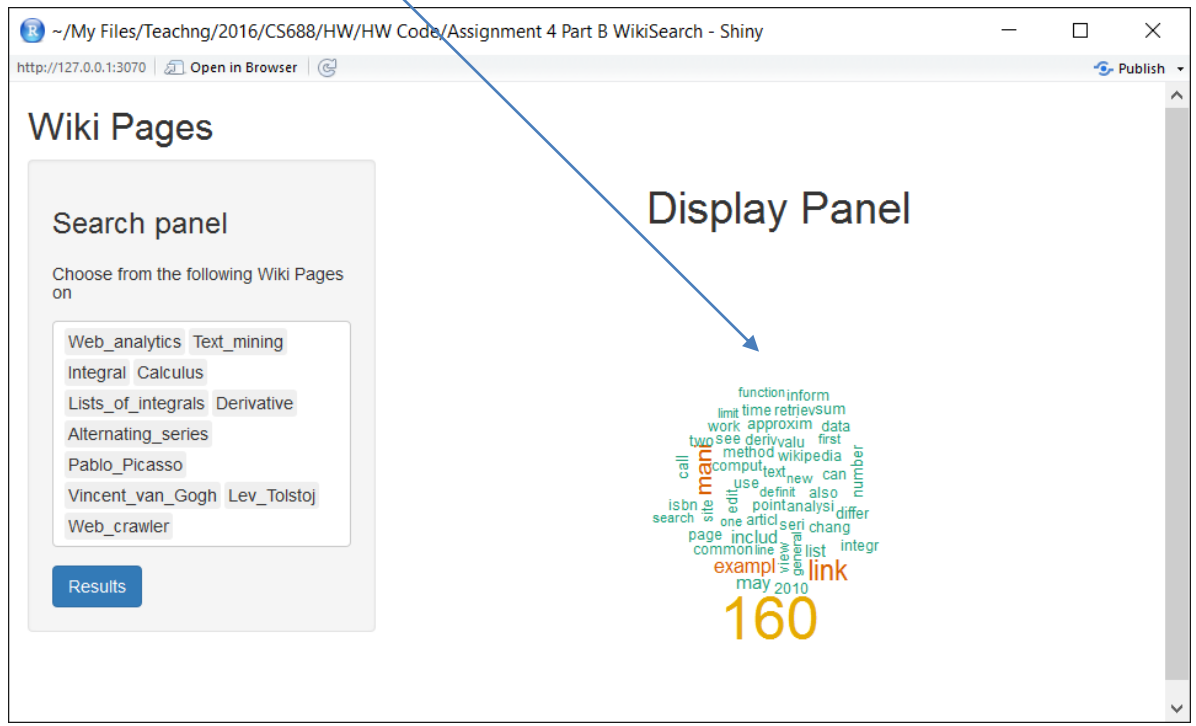
WikiSearch – Exercise/Assignment

- Submit your assignment as an R project named with your First, Last name!
- Create a Shiny App from the provided code in the Web Mining R script.
- Run the Search Wikipedia web pages app and get familiar with the Shiny code used to create it.
- Modify few of the lines in the code in server.R and WikiSearch.R that will replace the hierarchical clustering and the displayed dendrogram with a word cloud of the 50 most frequent terms in the document term matrix.

WikiSearch – Assignment

- Modify WikiSearch web app by just adding few lines of code to display the most frequent words in a word cloud.
- Web page selection - by removing or adding additional Wikipedia web pages.
- Modify WikiSearch web app and change the application from this to this
- Simple task, just add few lines of code in WikiSearch.R.
- These are lines of code you already have used before.





Checkout package wordcloud2 https://cran.r-project.org/web/packages/wordcloud2/vignettes/wordcloud.html

Example Search Wikipedia web pages: Script ui.R

- As before creates the Panels
  1. The object "titles" that contains the titles of the several Wikipedia web pages is passed to “choices” in selectInput().
  2. The widget selectInput() has an argument multiple set to TRUE so multiple choices can be selected/deselected.
  3. Before in mainPanel() we had htmlOutput() to display an HTML text.
  4. Now we use plotOutput() in mainPanel() to be able to display a plot in the Display Panel.
- **For the Assignment 4 it is good enough to just wordcloud-plot for the most frequent terms.**
- The rest of the code is standard compared to the previous ui.R script that you have used.

```
# Example: Shiny app that search Wikipedia web pages
# ui.R
library(shiny)
titles <- c("Web_analytics","Text_mining","Integral", "Calculus",
        "Lists_of_integrals", "Derivative","Alternating_series",
"Pablo_Picasso","Vincent_van_Gogh","Lev_Tolstoj","Web_crawler")
# Define UI for application
shinyUI(fluidPage(
  # Application title (Panel 1)
  titlePanel("Wiki Pages"),
  # Widget (Panel 2)
  sidebarLayout(
    sidebarPanel(h3("Search panel"),
          # Where to search
          selectInput("select",
                label = h5("Choose from the following Wiki Pages on"),
                choices = titles,
                selected = titles, multiple = TRUE),
          # Start Search
          submitButton("Results")
    ),
    # Display Panel (Panel 3)
    mainPanel(
      h1("Display Panel",align = "center"),
      plotOutput("distPlot")
    )
  )
))
```

Example Search Wikipedia web pages: Script server.R

- Note what you need to change in **server.R**
  - This code returns result from "WikiSearch.R"
  - This code plots the results as a dendrogram.
  - You need a wordcloud instead.

```
# Example: Shiny app that search Wikipedia web pages
# server.R
library(shiny)
library(tm)
library(stringi)
library(proxy)
source("WikiSearch.R")

shinyServer(function(input, output) {
  output$distPlot <- renderPlot({
 # Progress Bar while executing function
    withProgress({
      setProgress(message = "Mining Wikipedia ...")
      result <- SearchWiki(input$select)
    })
plot(result, labels = input$select, sub = "",main="Wikipedia Search")
  })
})
```

Example Search Wikipedia web pages: Script WikiSearch.R

- The reference to the used libraries and the script *WikiSearch.R.*
  1. The script *WikiSearch.R* uses *lapply()* to download all of the selected web pages and to create a corpus.
  2. Some items are removed (articles) to save space.
  3. The preprocessing as before is done with the content_transformer() function and then applied to the corpus with tm_map().
  4. The document term matrix is formed and it can be fairly large for a larger set of web pages.
  5. Note the reduction of the document term matrix by removing the sparse terms.
  6. In this particular case the euclidian distance measure is used for hierarchical clustering.
- Note that it may take some time to execute the *WikiSearch.R* script before it displays the result.

```
# Example: Shiny app that search Wikipedia web pages
# Wikipedia Search
library(tm)
library(stringi)
library(WikipediR)
SearchWiki <- function (titles) {
articles <- lapply(titles,function(i) page_content("en","wikipedia", page_name =
i,as_wikitext=TRUE)$parse$wikitext)
 docs <- Corpus(VectorSource(articles)) # Get Web Pages' Corpus
 remove(articles)
  # Text analysis - Preprocessing
 transform.words <- content_transformer(function(x, from, to) gsub(from, to, x))
 temp <- tm_map(docs, transform.words, "<.+?>", " ")
 temp <- tm_map(temp, transform.words, "\t", " ")
 temp <- tm_map(temp, content_transformer(tolower)) # Conversion to Lowercase
 temp <- tm_map(temp, stripWhitespace)
 temp <- tm_map(temp, removeWords, stopwords("english"))
 temp <- tm_map(temp, removePunctuation)
 temp <- tm_map(temp, stemDocument, language = "english") # Perform Stemming
 remove(docs)
# Create Dtm
 dtm <- DocumentTermMatrix(temp)
 dtm <- removeSparseTerms(dtm, 0.4)
 dtm$dimnames$Docs <- titles
 docsdissim <- dist(as.matrix(dtm), method = "euclidean") # Distance Measure
 h <- hclust(as.dist(docsdissim), method = "ward.D2") # Group Results
}
```

Assignment Hints

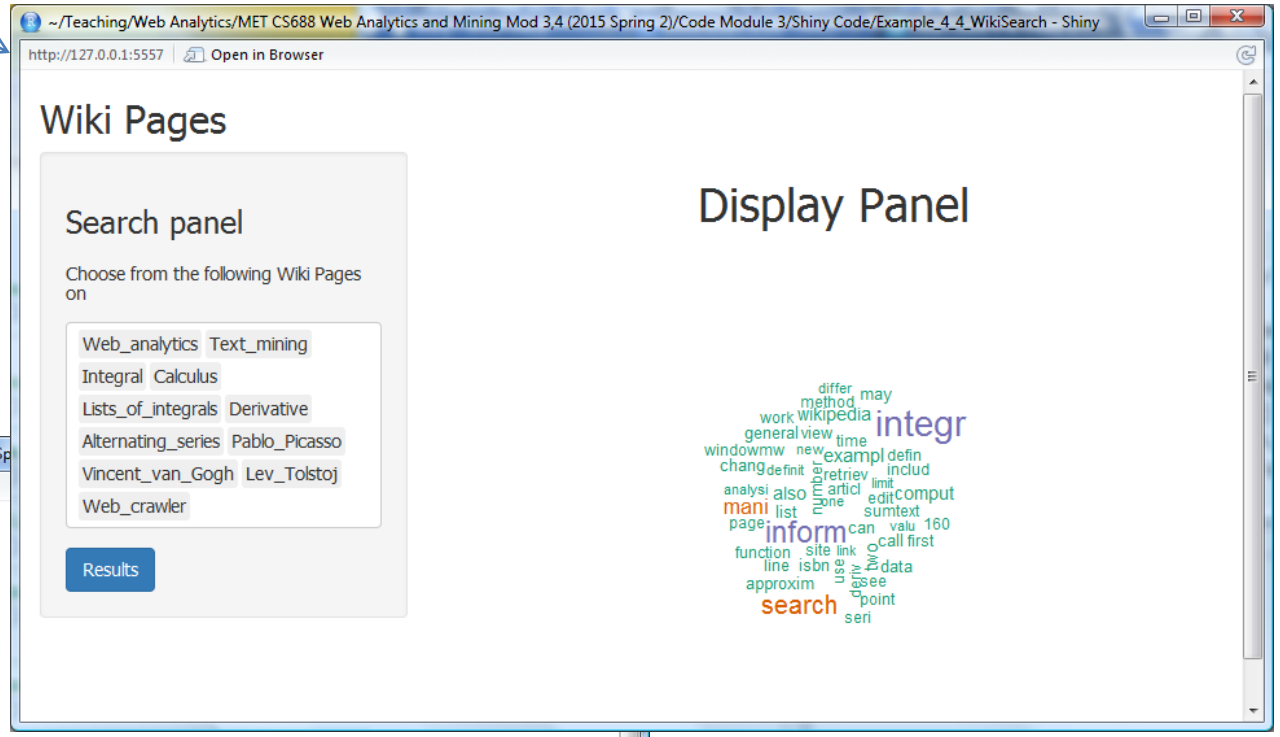
- Easiest maybe to create a script and test without Shiny, and then add it to Shiny.
- Changes are needed only at the plotting part.
- You would need to disable the code for hierarchical clustering.
- You already know how to find the most frequent terms from a **matrix** of dtm.
- Note it is easier to order the terms in decreasing order, you can find out how
  - > ?order
- Considering the decreasing order (most to list frequent) it is easier instead of *tail()* to use *head()*. Check how do you select the n first terms using *head()*.
  - > ?head
- At the end it is just a matter of replacing the hierarchical plot line in the Example code with the line for wordcloud. Something like:
  - wordcloud(names(freq[head(ord,n=50)]), freq[head(ord,n=50)], scale=c(4,0.9), colors=brewer.pal(6, "Dark2"))
- You can find more on the scale() and the other wordcloud options from the help files.

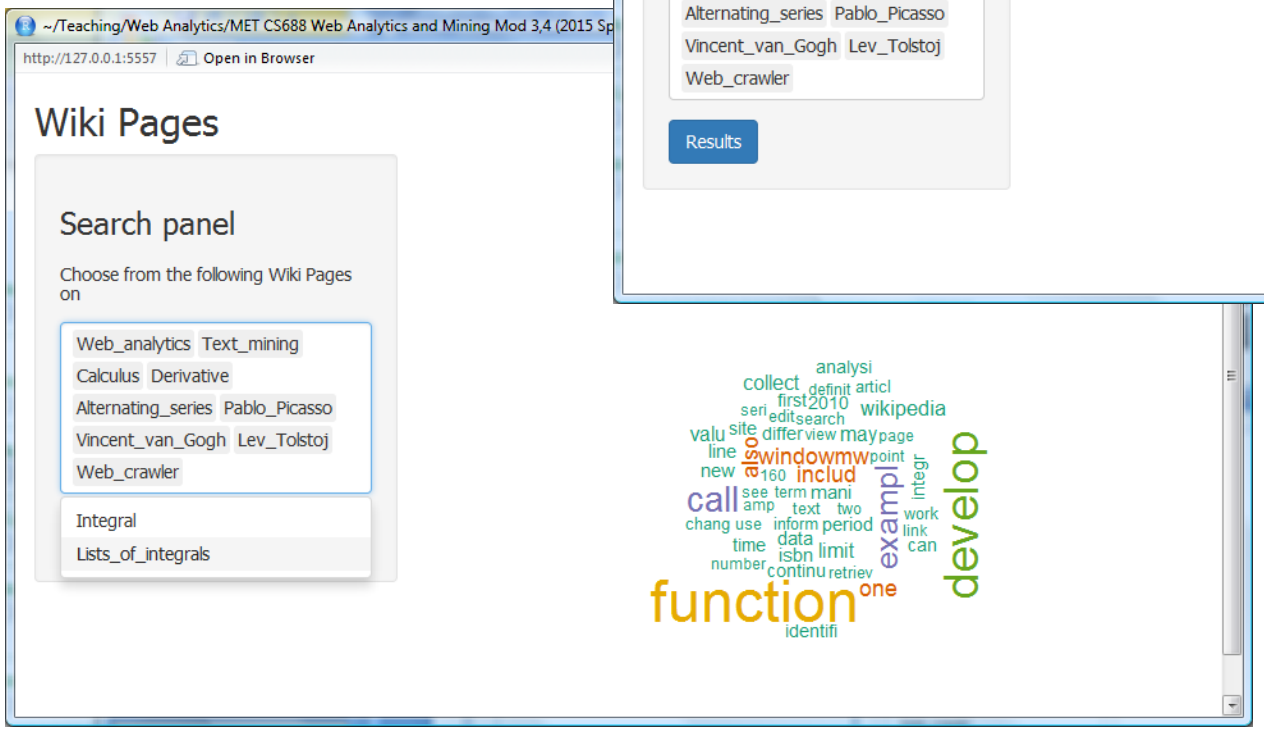
Word cloud for the 50 most common words for:

1) All the Wiki Pages

2) Removed pages:

- Integral,
- List of Integrals





Word Clouds in Shiny

If you are more enthusiastic about this for more in depth example you can see:

- http://shiny.rstudio.com/gallery/word-cloud.html