

Data Visualization & R Shiny

CEE 412/ CET 522

Transportation Data Management and Visualization

WINTER 2020

Outline

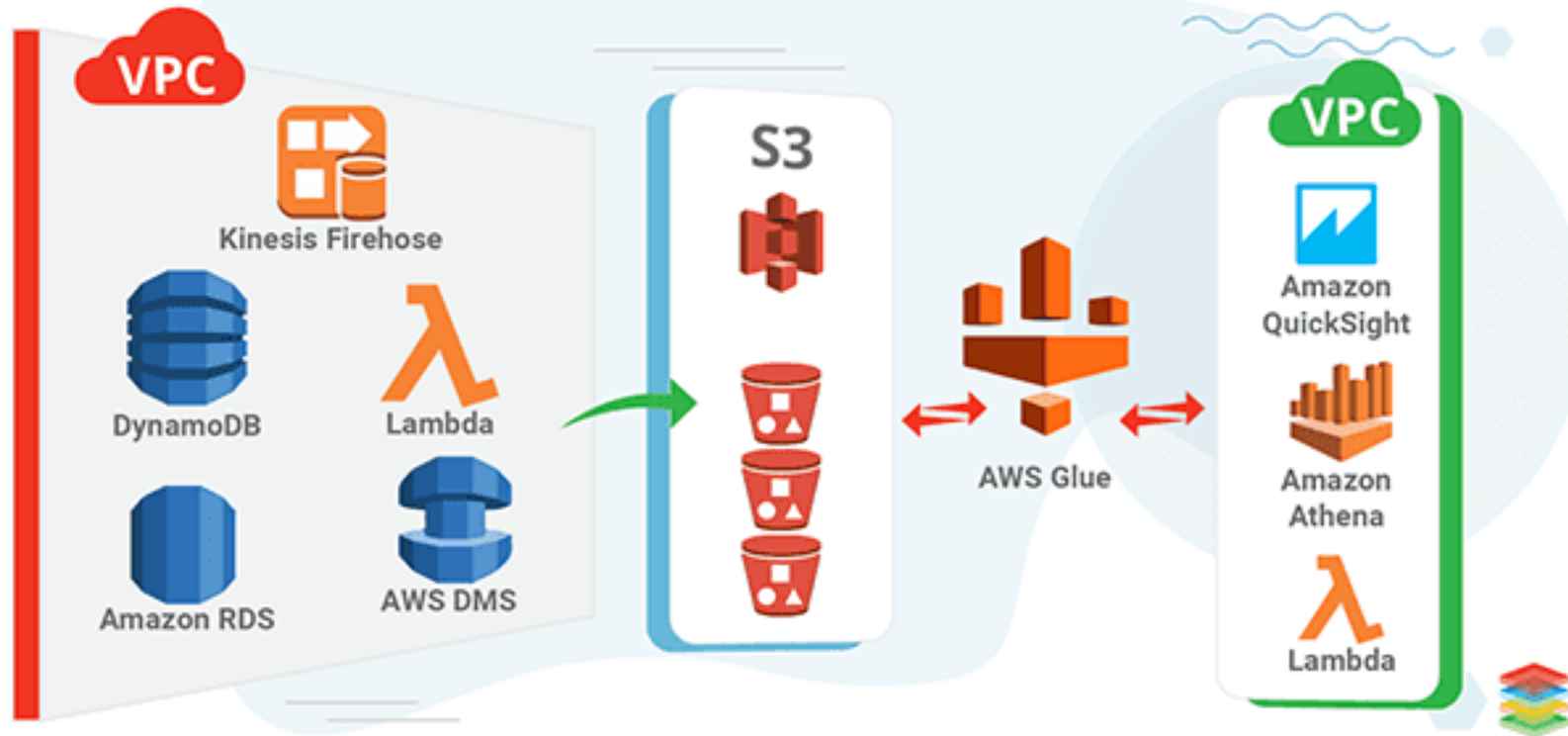
Data Pipeline

Visualization in R

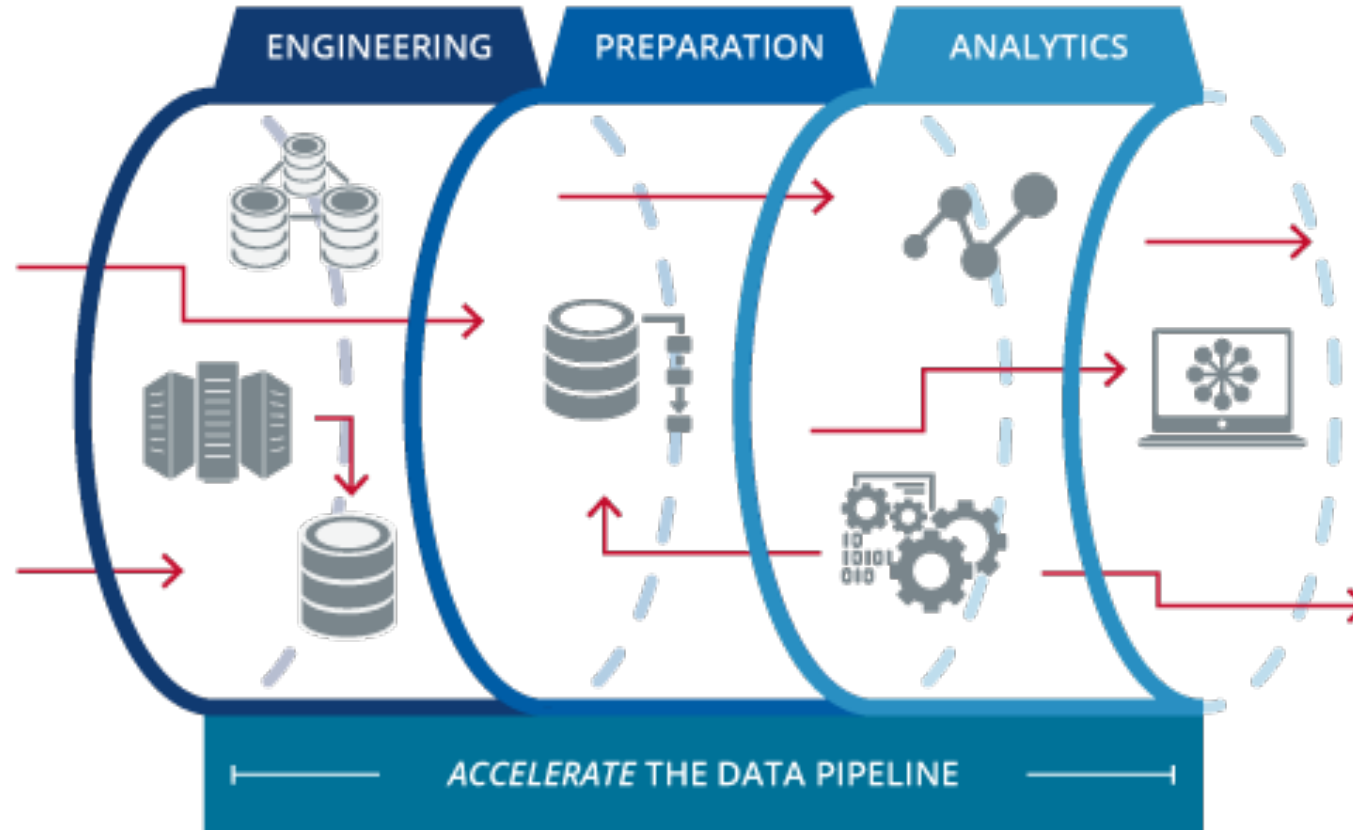
Introduction to R Shiny

Data Pipeline on AWS

Big Data Pipeline on AWS



What is a data pipeline?



What is a data pipeline?

“A arbitrarily complex chain of processes that manipulate data where the output data of one process becomes the input to the next.”

“A process to take raw data and transform in a way that usable by the entire organization.”

“Data Processing Pipeline is a collection of instructions to read, transform or write data that is designed to be executed by a data processing engine.”

What is a data pipeline

A data pipeline can have these characteristics:

- 1 or more data inputs.
- 1 or more data outputs.
- Optional filtering.
- Optional transformation, including schema changes (adding or removing fields) and transforming the format.
- Optional aggregation, including group by, joins, and statistics.
- Other robustness features.

Who needs a data pipeline?

Generate, rely on, store, or Maintain large amounts or multiple sources of data

Require real-time or highly sophisticated data analysis

Store data in the cloud

Most of the companies you interface with on a daily basis — and probably your own — would benefit from a data pipeline

ETL

The general procedure of copying data from one or more sources into a destination system which represents the data differently from the source(s)

The term comes from the three basic steps needed:

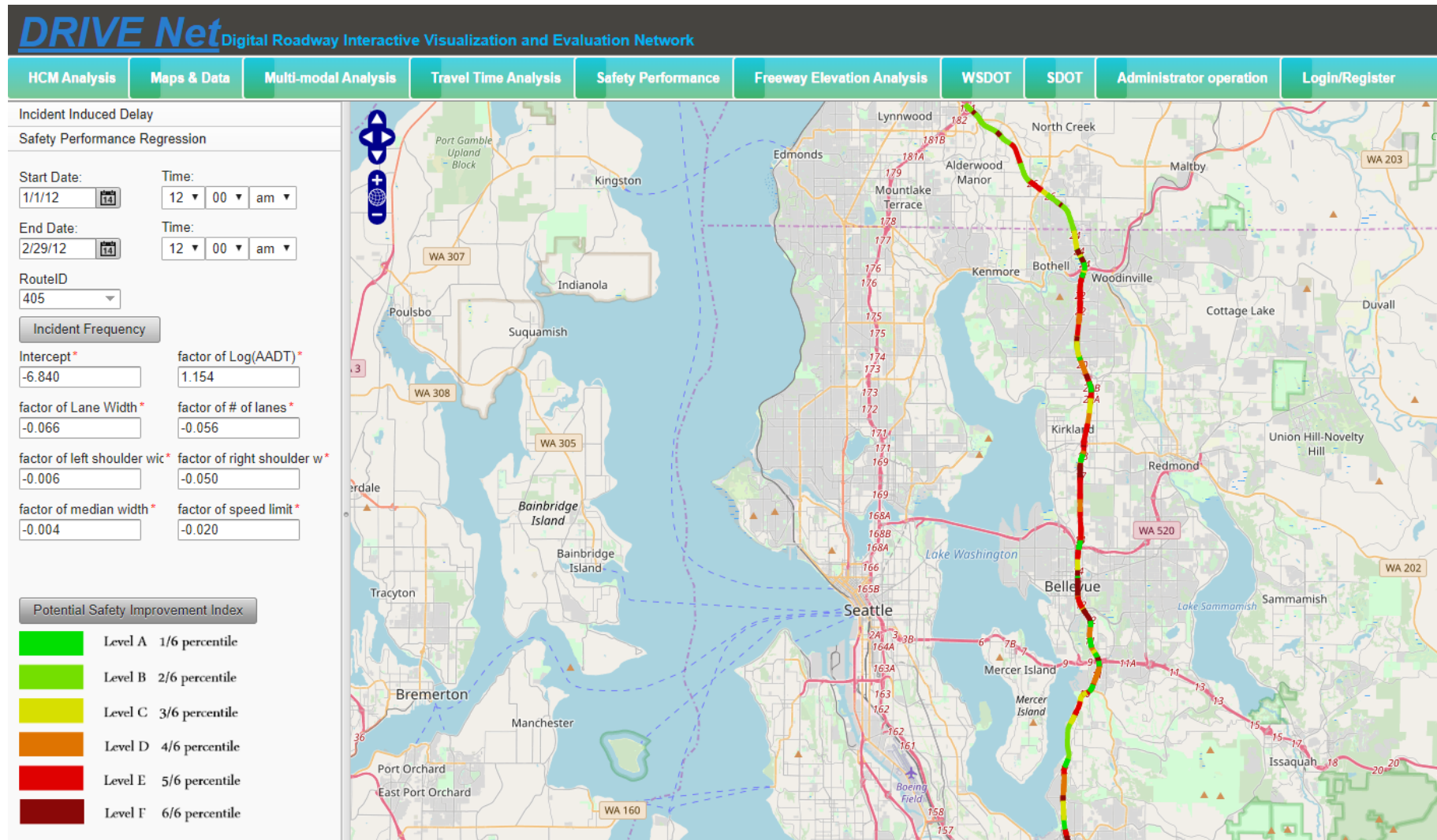
- ***Extracting***(selecting and exporting) data from the source
- ***Transforming*** the way the data is represented to the form expected by the destination
- ***Loading***(reading or importing) the transformed data into the destination system

Data pipeline

ETL pipeline refers to a set of processes extracting data from one system, transforming it, and loading into some database or data-warehouse.

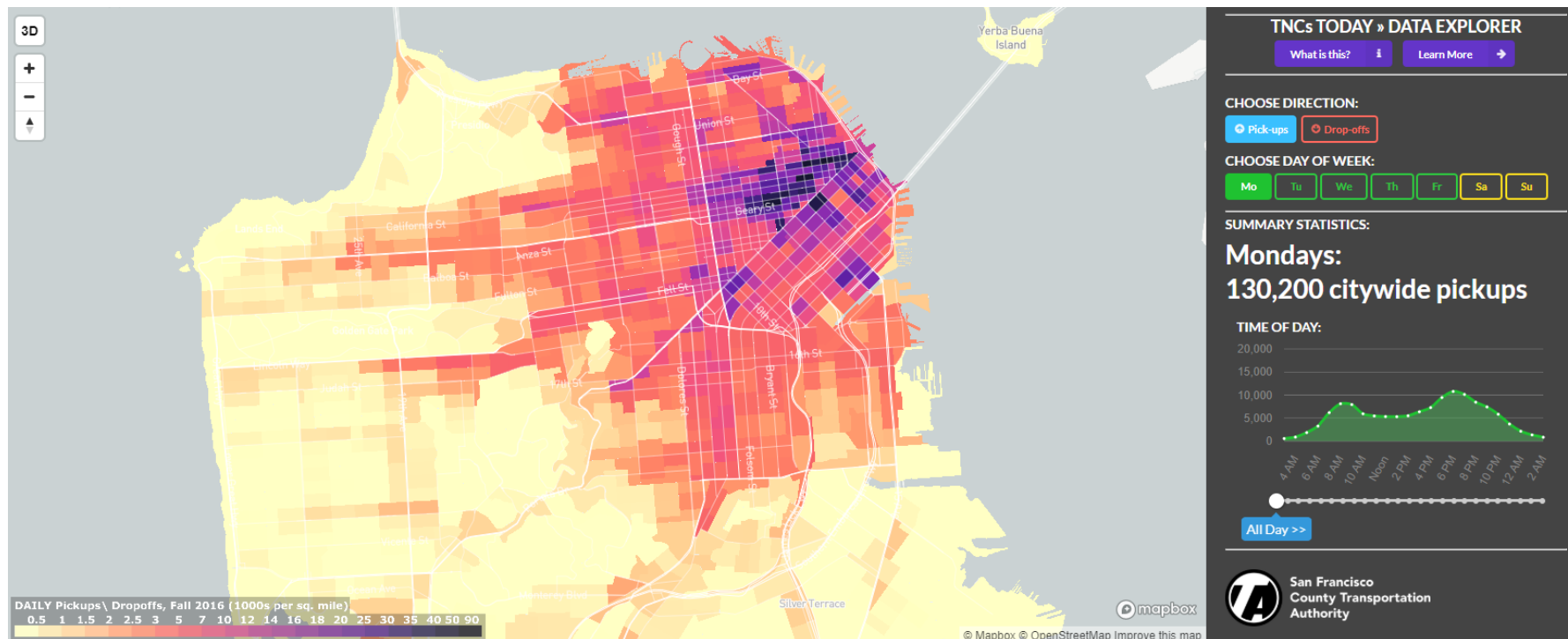
Data pipeline is a slightly more generic term. It refers to any set of processing elements that move data from one system to another, possibly transforming the data along the way.

Data pipeline



Data Visualization

- An imperative part of creating a reliable data pipeline.
- Final step in conversion of “data into information”.
 - This is an estimate of the number of TNC (Uber and Lyft) pickups and dropoffs in San Francisco—by location and by time of day.



Visualization in R

Basic plots and tables are necessary to visually check the analysis results.

ggplot2 is a system for declaratively creating graphics

Leaflet for R: interactive maps

- Leaflet.js is one of the most popular open-source JavaScript libraries for interactive maps.

Shiny: interactive visualization → dynamic visualization

- Connected to database

ggplot2

You provide the data, tell ggplot2 how to map variables to aesthetics, what graphical primitives to use, and it takes care of the details.

Installation:

```
# The easiest way to get ggplot2 is to install the whole tidyverse:  
install.packages("tidyverse")  
  
# Alternatively, install just ggplot2:  
install.packages("ggplot2")
```

Take the demographic information of midwest counties as an example

- A data frame with 437 rows and 28 variables

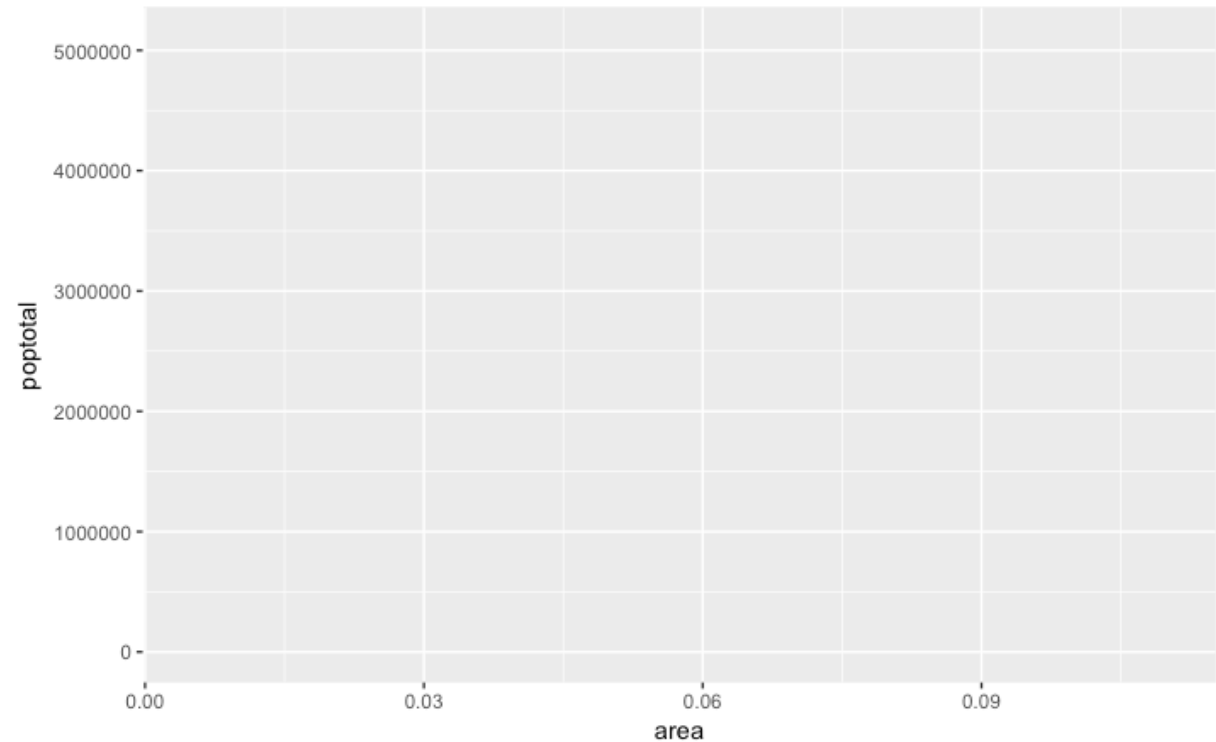
```
library(ggplot2)  
data("midwest", package = "ggplot2") # load the data  
# midwest <- read.csv("http://goo.gl/G1K41K") # alt source
```

ggplot2

A basic ggplot

```
ggplot(midwest, aes(x=area, y=poptotal)) # area and poptotal are columns in 'midwest'
```

- `aes()`: specify the X and Y axes

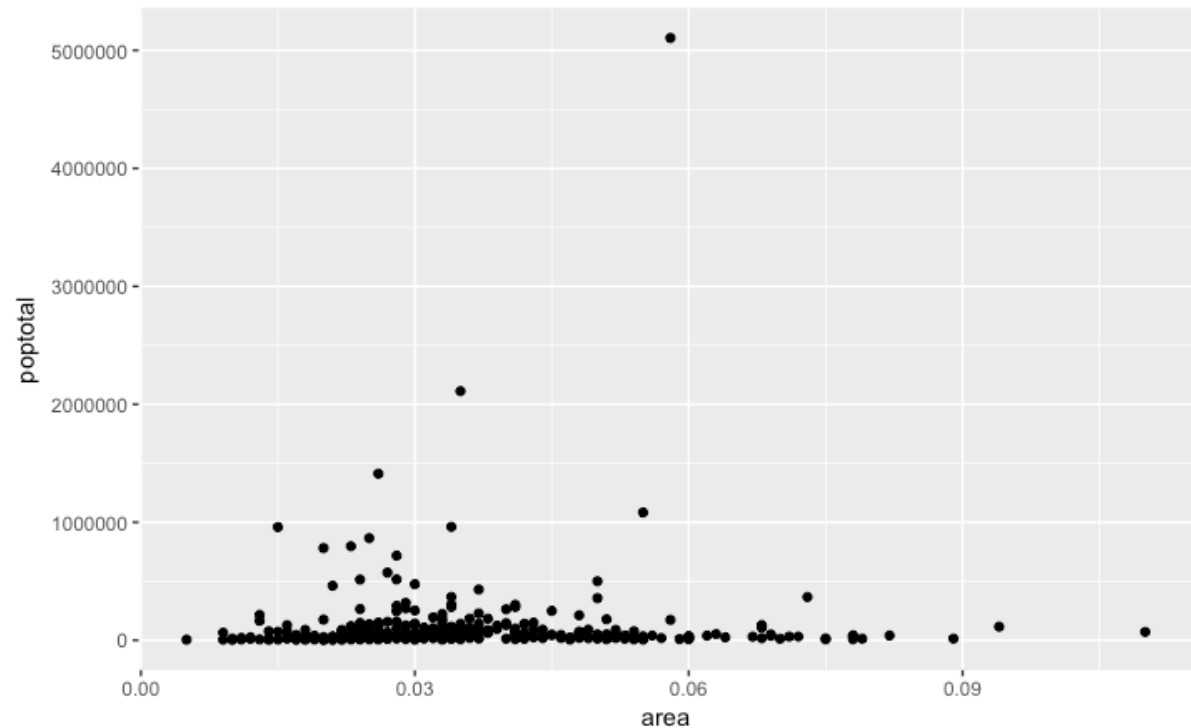


ggplot2

Make a scatterplot on top of the blank ggplot by adding points using a geom layer called `geom_point()`

```
ggplot(midwest, aes(x=area, y=poptotal)) + geom_point()
```

- `aes()`: specify the X and Y axes
- `geom_point()`: scatterplot

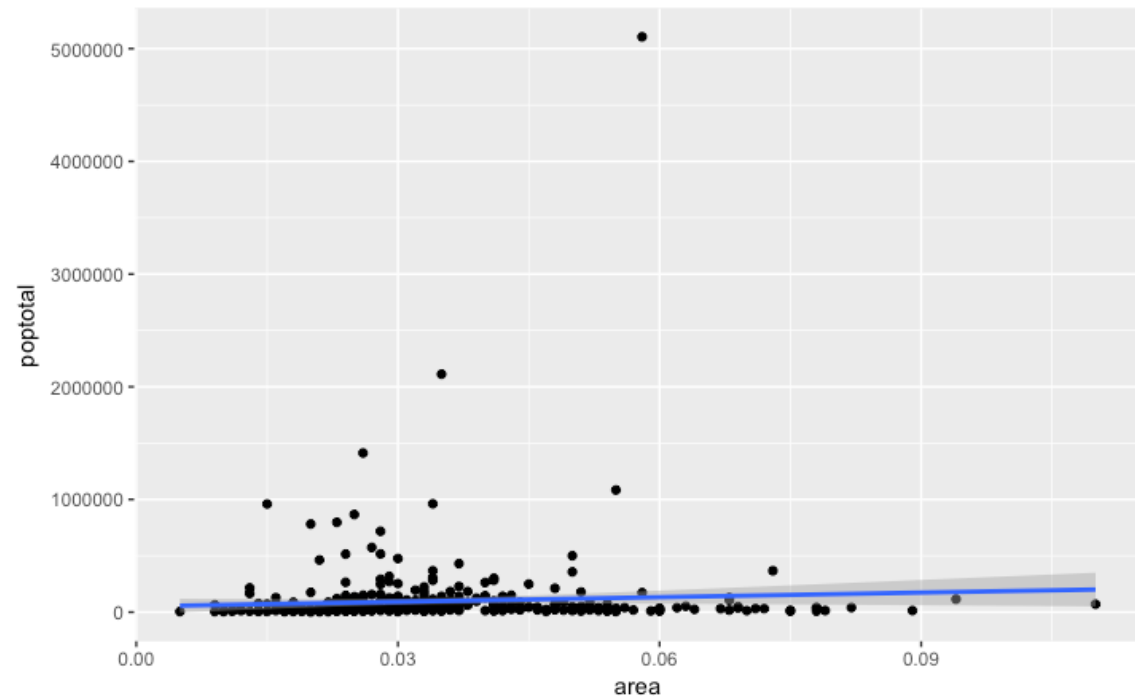


ggplot2

Add a smoothing layer

```
g <- ggplot(midwest, aes(x=area, y=poptotal)) + geom_point() + geom_smooth(method="lm") #  
set se=FALSE to turnoff confidence bands  
plot(g)
```

- `aes()`: specify the X and Y axes
- `geom_point()`: scatterplot
- `geom_smooth(method='lm')`:
smoothing layer
 - `lm`: short for linear model

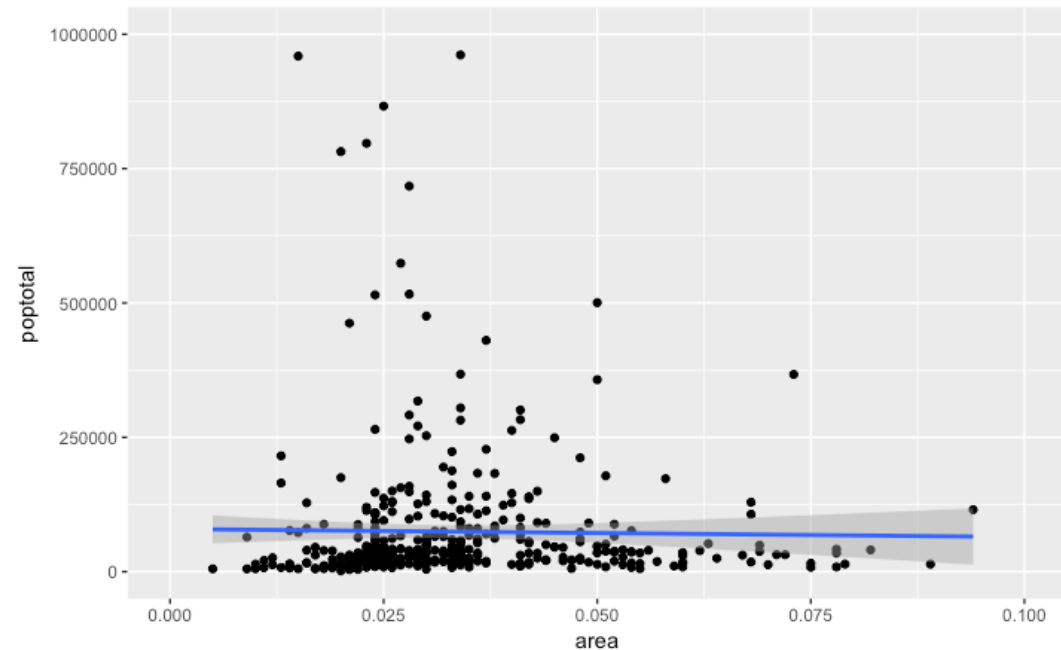


ggplot2

Adjusting the X and Y axis limits

```
g <- ggplot(midwest, aes(x=area, y=poptotal)) + geom_point() + geom_smooth(method="lm") #  
set se=FALSE to turnoff confidence bands  
  
# Delete the points outside the limits  
g + xlim(c(0, 0.1)) + ylim(c(0, 1000000)) # deletes points
```

- `aes()`: specify the X and Y axes
- `geom_point()`: scatterplot
- `geom_smooth(method='lm')`:
smoothing layer
 - `lm`: short for linear model
- `xlim()`: X axis limit
- `ylim()`: Y axis limit

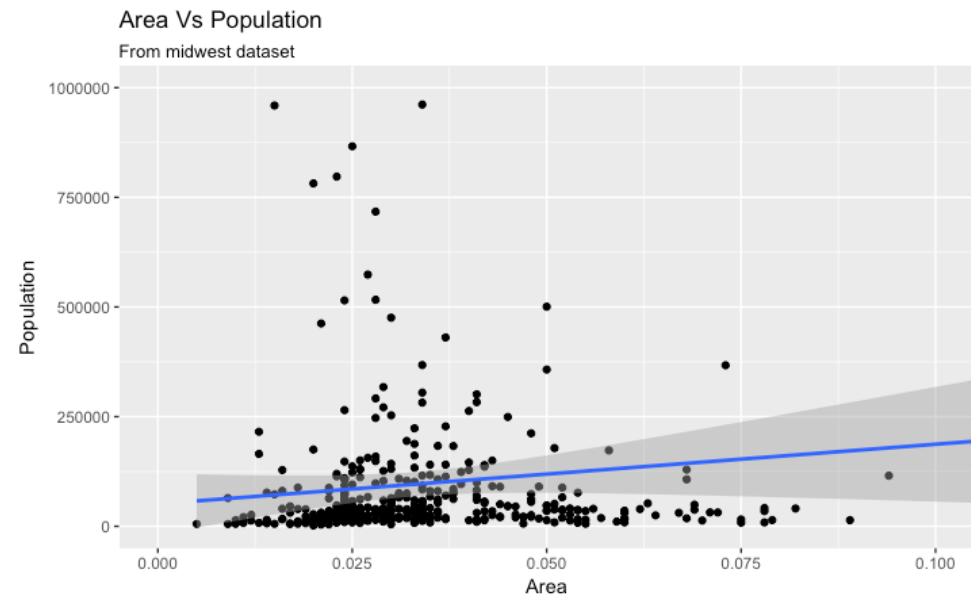


ggplot2

Change the Title and Axis Labels

```
g <- ggplot(midwest, aes(x=area, y=poptotal)) + geom_point() + geom_smooth(method="lm") #  
set se=FALSE to turnoff confidence bands  
  
g1 <- g + coord_cartesian(xlim=c(0,0.1), ylim=c(0, 1000000)) # zooms in  
  
# Add Title and Labels  
g1 + labs(title="Area Vs Population", subtitle="From midwest dataset", y="Population", x="Area",  
caption="Midwest Demographics")
```

- `aes()`: specify the X and Y axes
- `geom_point()`: scatterplot
- `geom_smooth(method='lm')`:
smoothing layer
 - `lm`: short for linear model
- `xlim()`: X axis limit
- `ylim()`: Y axis limit
- `labs()`: labels
- `ggtitle()`: set title
- `xlab()`: x label
- `ylab()`: y label



ggplot2

Pros

- Consistent, concise syntax
- Intuitive (to many)
- Visually appealing by default
- Entirely customizable
- Documentation

Cons

- Different syntax from the rest of R
- Static visualization

Leaflet for R

This R package makes it easy to integrate and control Leaflet maps in R.

Features

- Interactive panning/zooming
- Compose maps using arbitrary combinations of:
 - Map tiles
 - Markers
 - Polygons
 - Lines
 - Popups
 - GeoJSON
- Create maps right from the R console or RStudio
- Embed maps in *R Markdown* documents and *Shiny* apps

Leaflet for R

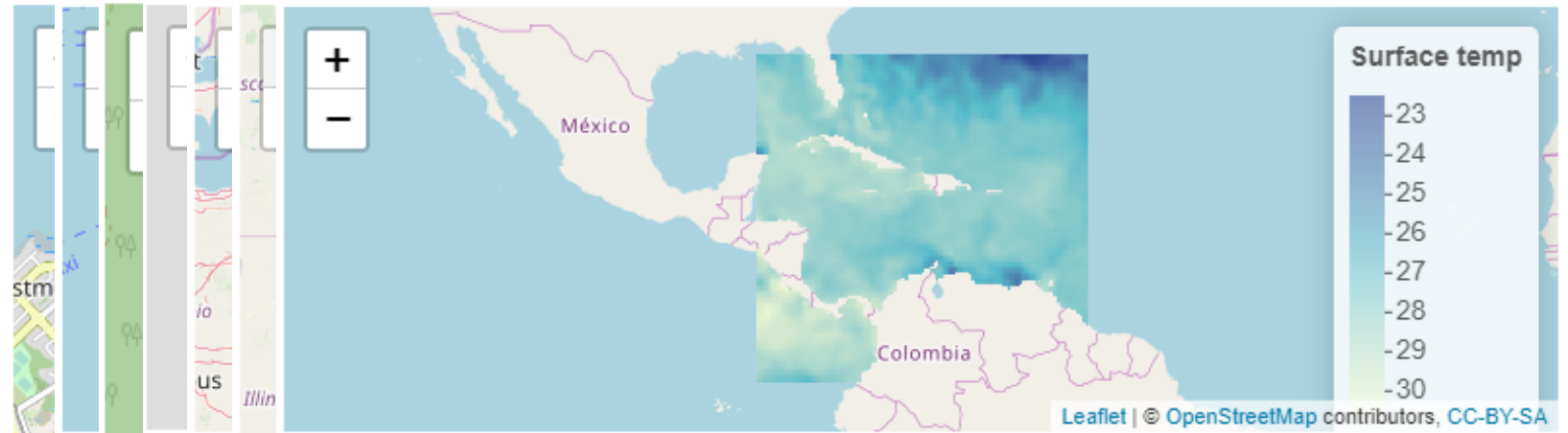
□ Installation

```
install.packages("leaflet")  
# to install the development version from Github, run  
# devtools::install_github("rstudio/Leaflet")
```

□ Basic Example

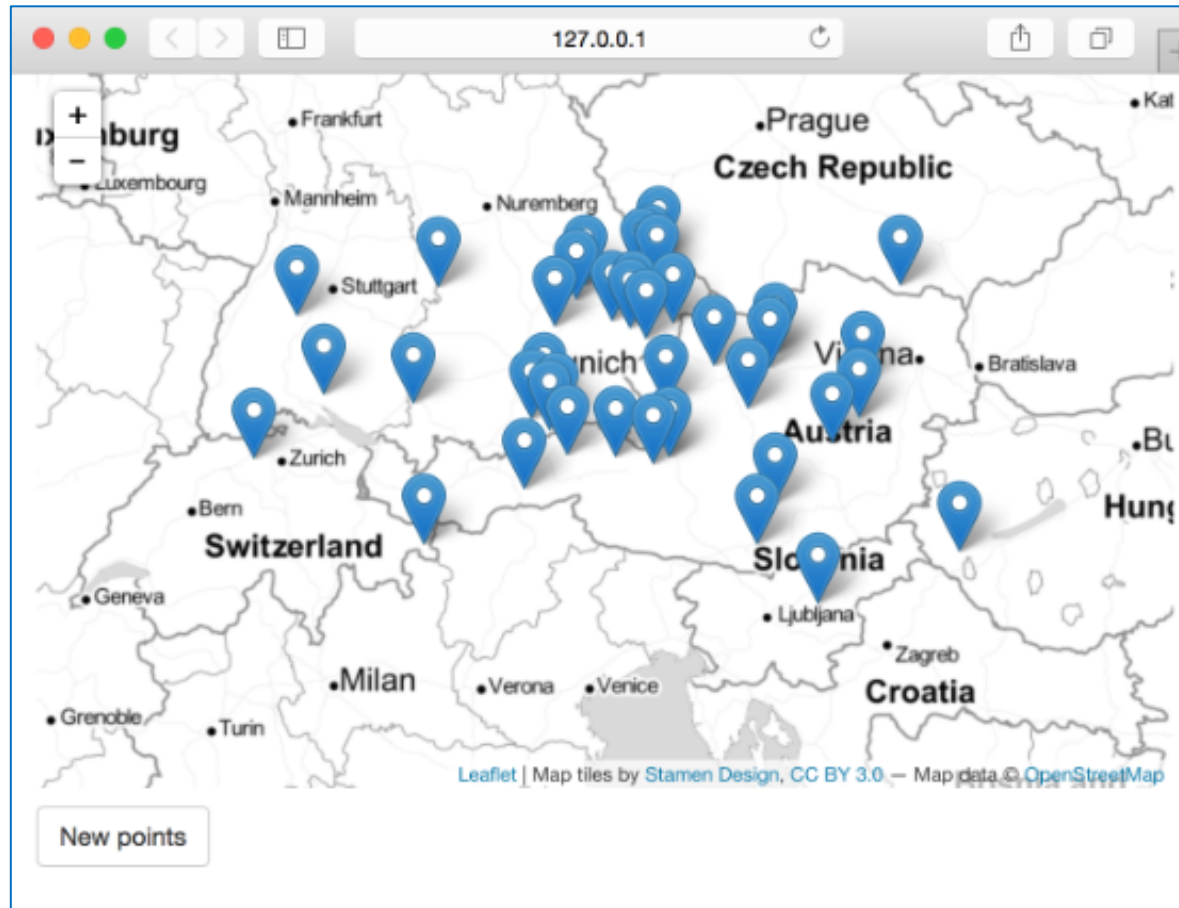
- Map → Add tile: OSM
- Markers
- Popups and Labels
- Lines and Shapes
- GeoJSON
- Raster Images

```
library(leaflet)  
  
m <- leaflet() %>%  
  addTiles() %>% # Add default OpenStreetMap map tiles  
  addMarkers(lng=174.768, lat=-36.852, popup="The birthplace of R")  
m # Print the map
```



Leaflet for R

The Leaflet package includes powerful and convenient features for integrating with Shiny applications.



```
library(shiny)
library(leaflet)

r_colors <- rgb(t(col2rgb(colors()) / 255))
names(r_colors) <- colors()

ui <- fluidPage(
  leafletOutput("mymap"),
  p(),
  actionButton("recalc", "New points")
)

server <- function(input, output, session) {

  points <- eventReactive(input$recalc, {
    cbind(rnorm(40) * 2 + 13, rnorm(40) + 48)
  }, ignoreNULL = FALSE)

  output$mymap <- renderLeaflet({
    leaflet() %>%
      addProviderTiles(providers$Stamen.TonerLite,
        options = providerTileOptions(nowrap = TRUE)
      ) %>%
      addMarkers(data = points())
  })
}

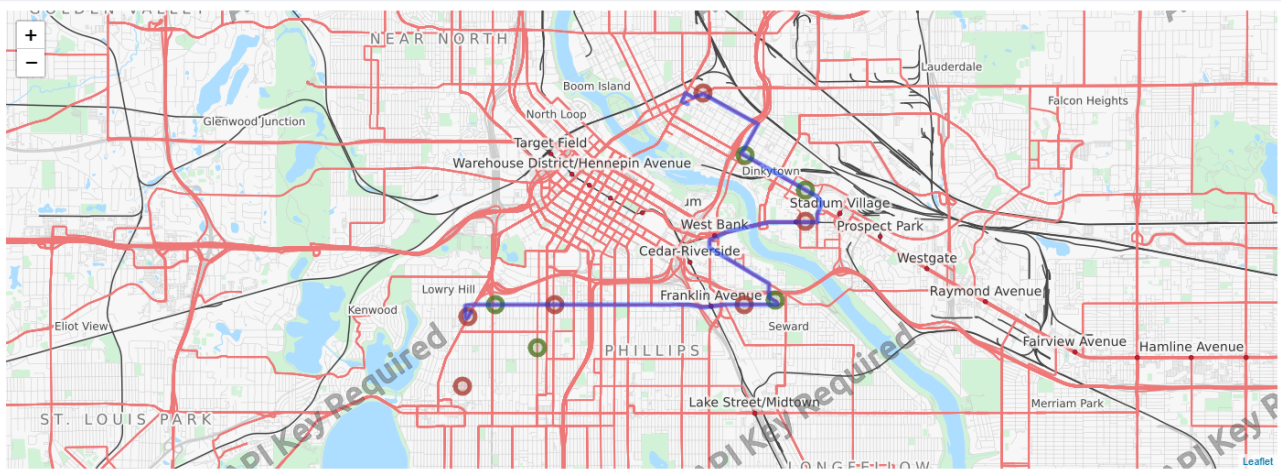
shinyApp(ui, server)
```

R Shiny

“Open source **R** package that provides an elegant and powerful web framework for building web applications”

Shiny from R Studio Back to Gallery Get Code

Twin Cities Buses



The map displays a network of bus routes in the Twin Cities area. Route 2 is highlighted in blue. Various bus stops are marked with colored dots: purple for northbound, dark blue for southbound, green for eastbound, and red for westbound. The map includes street names and neighborhood labels like NEAR NORTH, PHILLIPS, and ST. LOUIS PARK.

Route
2

Show

- Northbound
- Southbound
- Eastbound
- Westbound

Note: a route number can have several different trips, each with a different path. Only the most commonly-used path will be displayed on the map.

Zoom to fit buses

Refresh interval
1 minute

Data refreshed 5 seconds ago.

Refresh now

Source data updates every 30 seconds.

Color	Direction	Number of vehicles
■	Northbound	0
■	Southbound	0
■	Eastbound	5
■	Westbound	6
	Total	11

Features

- Build useful web applications with only a few lines of code.
- No JavaScript required.
- Shiny applications are automatically “live” in the same way that spreadsheets are live.
- Outputs change instantly as users modify inputs, without requiring a reload of the browser.
- Shiny user interfaces can be built entirely using R, or can be written directly in HTML, CSS, and JavaScript for more flexibility.

Features

- Works in any R environment (Console R, Rgui for Windows or Mac, ESS, StatET, RStudio, etc.).
- Attractive default UI theme based on Twitter Bootstrap.
- A highly customizable slider widget with built-in support for animation.



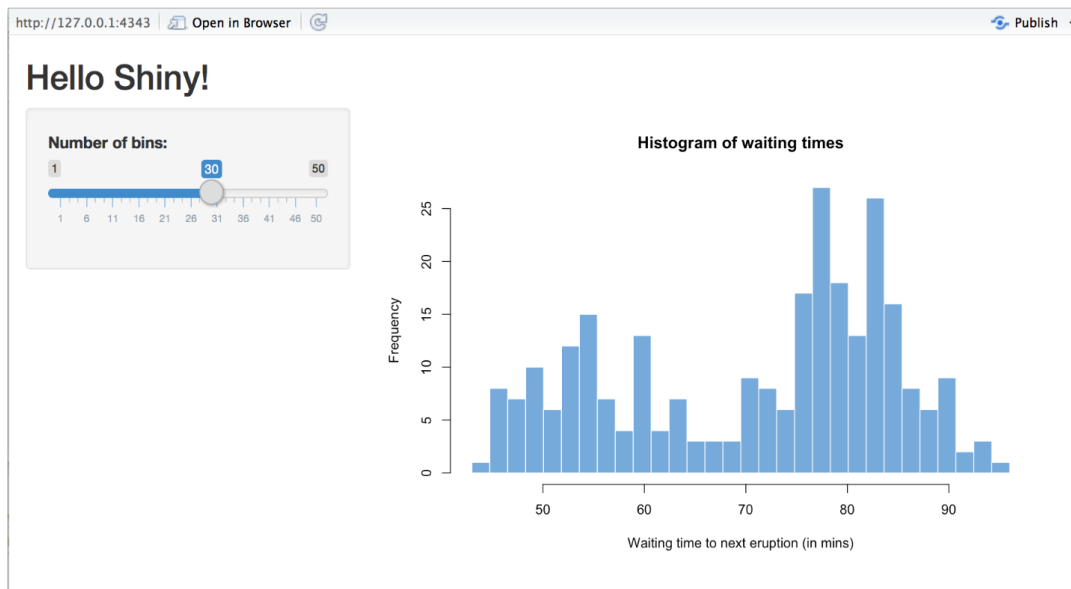
- Pre-built output widgets for displaying plots, tables, and printed output of R objects.

Features

- Fast bidirectional communication between the web browser and R using the websockets package.
- Uses a reactive programming model that eliminates messy event handling code, so you can focus on the code that really matters.

Features

- Eleven built-in examples.
- Each example demonstrates how Shiny works.
- Each example is a self-contained Shiny app.



```
runExample("01_hello")      # a histogram
runExample("02_text")      # tables and data frames
runExample("03_reactivity") # a reactive expression
runExample("04_mpg")       # global variables
runExample("05_sliders")   # slider bars
runExample("06_tabsets")   # tabbed panels
runExample("07_widgets")   # help text and submit buttons
runExample("08_html")      # Shiny app built from HTML
runExample("09_upload")    # file upload wizard
runExample("10_download")  # file download wizard
runExample("11_timer")     # an automated timer
```

Shiny Application Structure

- `install.packages("shiny")`
- `library("shiny")`
- `runExample("01_hello")`
- Contained in a single script called **app.R**
- The script **app.R** is in a directory (for example, `newdir/`)
- The app can be run with **`runApp("newdir")`**

Shiny Application Structure

app.R has three components:

- a user interface object

The user interface (ui) object controls the layout and appearance of your app. Also displays the output. For example, title, page layout, text input, radio buttons, drop down menu etc.

- a server function

The server function contains the instructions that your computer needs to build your app. In other words a set of instructions that uses the input provided by the user, process them and produce the required output.

- a call to the `shinyApp` function

The `shinyApp` function creates Shiny app objects from an explicit UI/server pair.

Running the Shiny Application

- Every Shiny app has the same structure: an `app.R` file that contains `ui` and `server`.
- You can create a Shiny app by making a new directory and saving an `app.R` file inside it. It is recommended that each app will live in its own unique directory.
- You can run a Shiny app by giving the name of its directory to the function `runApp`. For example if your Shiny app is in a directory called `my_app`, run it with the following code:

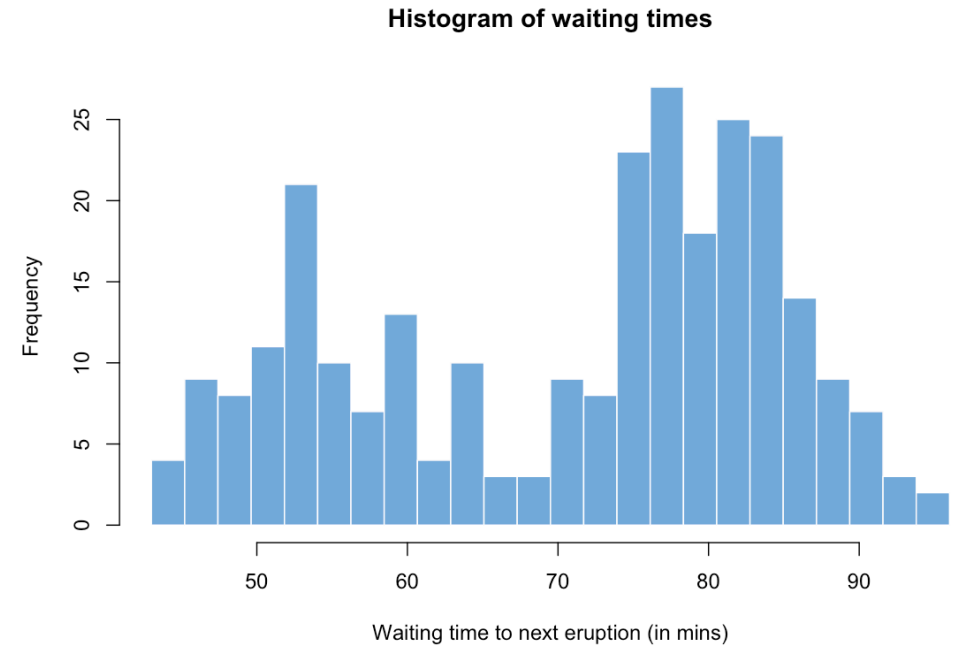
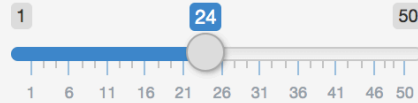
Shiny Examples

Easiest example:

```
library(shiny)
runExample("01_hello")
```

Hello Shiny!

Number of bins:



Structure of ui.R

```
library(shiny)

# See above for the definitions of ui and server
ui <- ...

server <- ...

shinyApp(ui = ui, server = server)
```

```
library(shiny)

# Define UI for app that draws a histogram ----
ui <- fluidPage(

  # App title ----
  titlePanel("Hello Shiny!"),

  # Sidebar layout with input and output definitions
  sidebarLayout(

    # Sidebar panel for inputs ----
    sidebarPanel(),

    # Main panel for displaying outputs ----
    mainPanel()

  )
)
```


Structure of the Server

- Input consists of all the logical functionalities/calculations.
- The output is displayed in the main panel.

```
# Define server logic required to draw a histogram ----  
server <- function(input, output) {  
  
  # 1. It is "reactive" and therefore should be automatically  
  #   re-executed when inputs change  
  # 2. Its output type is a plot  
  
}
```

- Shiny App:

```
shinyApp(ui = ui, server = server)
```

Example

Check out the real demo:

Connection between UI & Server

UI

```
# Sidebar panel for inputs ----
sidebarPanel(

  # Input: Slider for the number of bins ----
  sliderInput(inputId = "bins",
             label = "Number of bins:",
             min = 1,
             max = 50,
             value = 30)

),
```

```
# Main panel for displaying outputs -
mainPanel(

  # Output: Histogram ----
  plotOutput(outputId = "distPlot")

)
```

Server

```
# Define server logic required to draw a histogram ----
server <- function(input, output) {

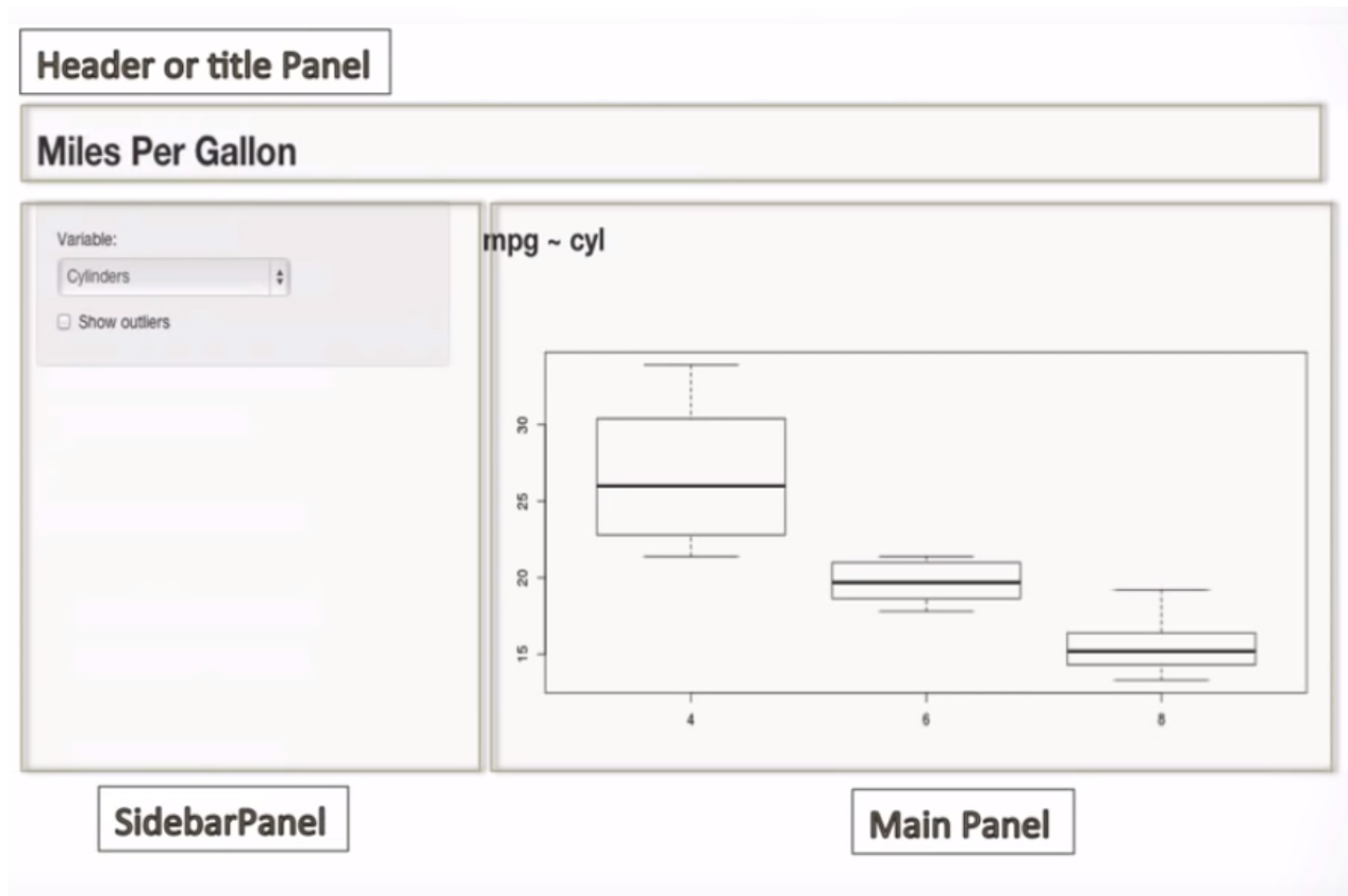
  output$distPlot <- renderPlot({

    x <- faithful$waiting
    bins <- seq(min(x), max(x), length.out = input$bins + 1)

    hist(x, breaks = bins, col = "#75AADB", border = "white",
         xlab = "Waiting time to next eruption (in mins)",
         main = "Histogram of waiting times")

  })
}
```

Example



Running the Shiny Application

- Save both the files for the ui.r and server.r in the R working directory (or any other folder).
 - Or save ui and server in one file.
- Run the application in your local system using the runApp() command (specifying the folder name).
- You can also run the application from Rstudio.
- The application can be run in the browser too.

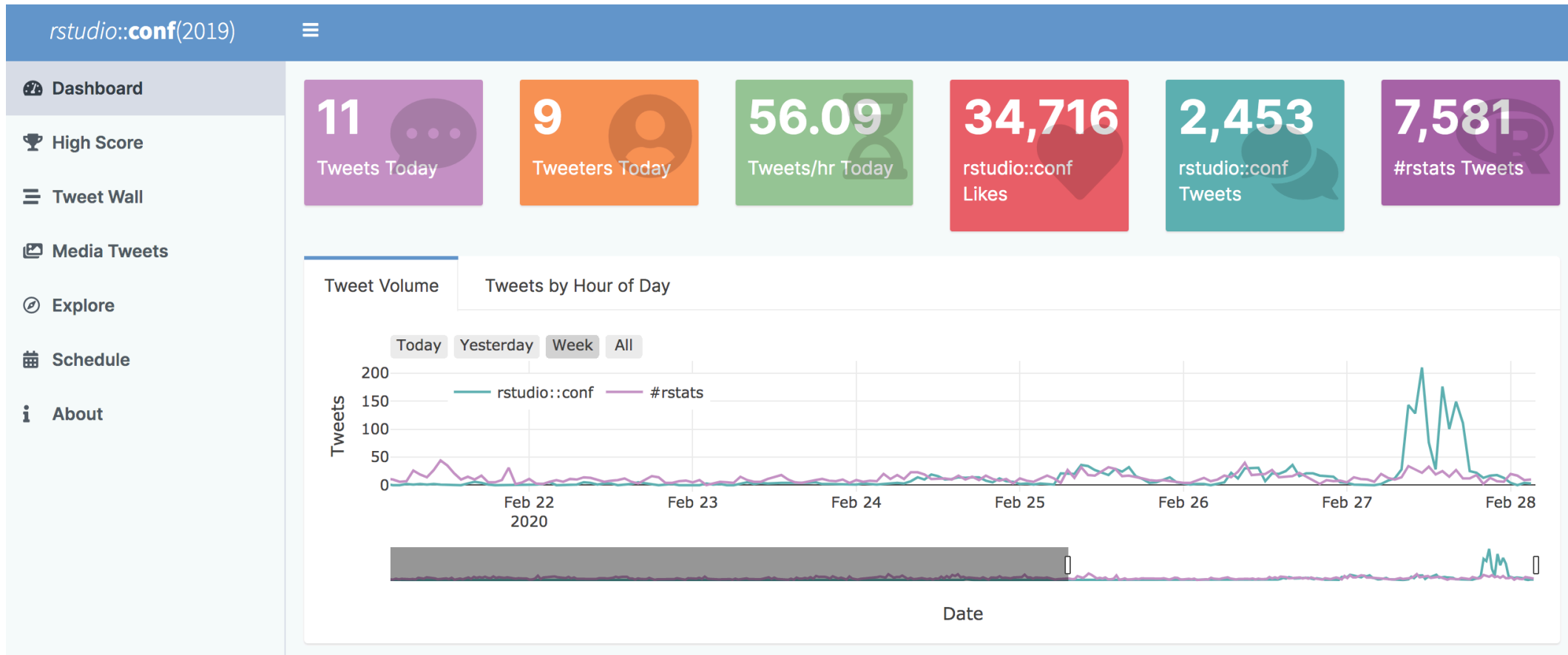
UI

Check out Shiny Gallery: <https://shiny.rstudio.com/gallery/>

A lot for widgets for your functional design

A bunch of interface design or user showcase, which can be adopted as your template.

Dashboard



<https://shiny.rstudio.com/gallery/conference-tweet-dashboard.html>

Data source for Shiny

Local files

- Read local files

Databases

- Read from and save into database
- Need database driver



Cloud

- AWS

Use package DBI, odbc:
Define all connection information in R code

```
# install.packages("DBI")
library(DBI)
# install.packages("odbc")
library(odbc)

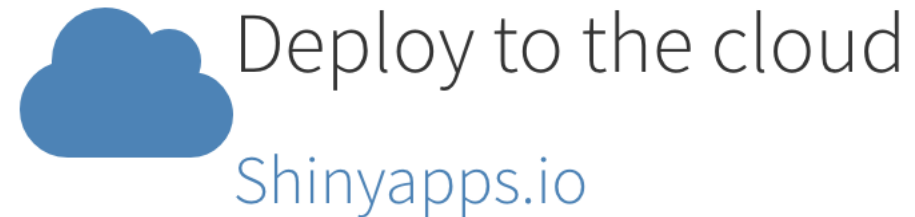
con <- DBI::dbConnect(odbc::odbc(),
                      Driver   = "SQL Server",
                      Server   = "128.95.29.72",
                      Database = "CEE412_CET522_w20",
                      UID      = "██████████",
                      PWD      = "██████████",
                      Port     = 1433)

dbGetQuery(con, "SELECT * FROM E2_CEOs")
```


Deploy

Check out Shiny Deploy page: <https://shiny.rstudio.com/deploy/>

Share your Shiny Application Online: <https://www.shinyapps.io/>



Check out my demo:

<https://zhiyongc.shinyapps.io/test/>

Final Project

Final Project has been posted.

- Build a Shiny App as a data pipeline or even a data platform
- Try more available Shiny widgets and functions!
- Use well-designed and decent UI, dashboard, or User Showcase as your template.
- Be creative!!!

Check out the requirement and the criteria in the project document.

Final Project

Groups that do well in the Final presentation will have opportunity to post your Shiny App **in the student project gallery** of CEE412/CET522 Winter 2020.

Great Opportunity: You are encouraged to participate the **Shiny Contest 2020**

- <https://blog.rstudio.com/2020/02/12/shiny-contest-2020-is-here/>
- Your final project can be quite related to this contest!