Sales Data Analysis

Applications of the Shiny App Environment

Kyle Stone

Aug 6th, 2024

Hello, and welcome to this build document showing process and code used in the creation of a Shiny App I built, which can be found here: https://kylestone225.shinyapps.io/Sales_Data_Analysis/. This app and document are the makeup of part two of a three part learning module I have created titled, "Practical Applications of the R Language for Sales Management". You can view all documentation related to this project at my GitHub: https://github.com/kylestone225.

If you are curious about the functionality of the Shiny App, I encourage you to visit the above shinyapps.io link and use it! The welcome page of the app should provide sufficient detail of the app's purpose and how to use it.

This document serves to provide the code I produced to create the app, and commentary on the methodology used to read and analyze the data set. Unlike other sections of the overall learning module; this does not intend to be a tutorial. If you have some familiarity with the R language, and would like to learn to build within the environment, visit https://mastering-shiny.org/index.html. Good luck!

The first portion of the code shows data collection and cleaning methods necessary to make joins and ultimately produce our primary data set used in the app.

```
# Mapping data for USA
usa <- st_as_sf(maps::map("state", fill = TRUE, plot = FALSE))
# Cleaning state data
usa <- usa |>
  rename(State = ID)

usa$State <- str_to_title(usa$State)

# Store our original data set found at
# 'https://www.kaggle.com/datasets/bhanupratapbiswas/superstore-sales/data'
store_data <- as.data.frame(read.csv("superstore_final_dataset (1).csv"))
# Cleaning store data
store_data$Postal_Code <- as.character(store_data$Postal_Code)

store_data$Order_Date <- format(as.Date(store_data$Order_Date,
                                        format = '%d/%m/%Y'),'%Y-%m-%d')
# Zip code data from the 'tigris' library
uszips <- zctas(cb = TRUE, year = 2020)
# Cleaning zip data
uszips <- uszips |>
  rename(zip_geometry = geometry)
```

OK, at this stage we have three data tables:

1. Our main set of superstore sales data.
2. US State mapping data.
3. US zip code mapping data.

Below we will do some calculations, and then join everything together. The code demonstrates general data munging techniques to extract all of the variable iterations of the sales dollar data you can find given the available vectors.

If you have some familiarity with R, you will notice in the server function of the app I use very few of the vectors created below. The app produces these figures on its own depending on user interaction. I have included this coding section to demonstrate best practices of initial calculations of the data. IE if you handed me the superstore sales data set with instructions to conduct general analysis, I would begin by performing these calculations.

```
# Sales by state
store_by_state <- store_data |>
  group_by(State) |>
  summarize(total_state_sales = sum(Sales))
# Sales by zip
store_by_zip <- store_data |>
  group_by(Postal_Code) |>
  summarize(total_zip_sales = sum(Sales))
# Sales by category
store_data_cat <- store_data |>
  group_by(Category) |>
  summarise(cat_sales = sum(Sales))
# Sales by subcategory
store_data_subcat <- store_data |>
  group_by(Sub_Category) |>
  summarise(subcat_sales = sum(Sales))
# Sales by region
store_data_region <- store_data |>
  group_by(Region) |>
  summarise(region_sales = sum(Sales))
# Sales by order date and cumulative sales by orderdate
store_data_order_date <- store_data |>
  group_by(Order_Date) |>
  summarize(sales_by_OD = sum(Sales)) |>
  mutate(sum_sales_OD = cumsum(sales_by_OD))
# Category sales by region
reg_cat_sales <- store_data |>
  group_by(Region, Category) |>
  summarize(region_cat_sales = sum(Sales))
# Subcategory sales by region
reg_subcat_sales <- store_data |>
  group_by(Region, Category, Sub_Category) |>
  summarize(regional_subcat_sales = sum(Sales))
# Category sales by order date
od_cat <- store_data |>
  group_by(Order_Date, Category) |>
  summarize(cat_date_sales = sum(Sales))

# Join everything to your orginal store data
store_data <- left_join(store_data, store_by_state, by = "State")
```

```
store_data <- left_join(store_data, store_by_zip, by = "Postal_Code")
store_data <- left_join(store_data, store_data_cat, by = "Category")
store_data <- left_join(store_data, store_data_subcat, by = "Sub_Category")
store_data <- left_join(store_data, store_data_region, by = "Region")
store_data <- left_join(store_data, store_data_order_date, by = "Order_Date")
store_data <- left_join(store_data, reg_cat_sales, by = c("Region",
                                                          "Category"))
store_data <- left_join(store_data, reg_subcat_sales, by = c("Region",
                                                             "Category",
                                                             "Sub_Category"))
store_data <- left_join(store_data, od_cat, by = c("Order_Date",
                                                   "Category"))
store_data <- left_join(store_data, usa, by = "State")
store_data <- left_join(store_data, uszips, by = c("Postal_Code" = "ZCTA5CE20"))
```

Shiny interactivity does not mesh well with 'list' objects which compose the State mapping data. Below creates a dueling data frame with the 'list' object (maps) data removed. It is used to build the interactive plots within the app

```
sd_plotting <- as.data.frame(store_data)
sd_plotting$geom <- NULL
sd_plotting$zip_geometry <- NULL
```

With the munging completed above, you could from here move on to more complex analytic exercises incorporating linear modeling to make predictive claims.
However, my purpose of working with this data is to make it easily presentable and digestible to a wide audience. The data is simple, but says important things very clearly: who bought what kinds of products, where, when, and for how much.

Shiny Applications provide an easy to use platform for broad audiences, internally within your organization, to access and understand data in a way that allows for more productive decision making.

Shiny Apps are not just for data presentation. Part 1 of this learning tutorial focuses on textual analysis. A Shiny App could be created to read transcripts, run analysis, and produce reports. Engineers on product development teams use Shiny Apps to make design adjustments, and simulate the effects of those adjustments in their product's environment.

We begin with the creation of our Shiny App below. The first section of code builds the user interface of the app: tabs, selection inputs, buttons, and placement.

```
ui <- fluidPage(
  tabsetPanel(
    tabPanel("Welcome",
             tags$h1("Analyze A Sales Dataset"),
             tags$p("Hello and thank you for taking interest in this Shiny App I
             have created to analyze some sample sales data!  The data used in
             this app can be found here:"),
             tags$a(href=
                    "https://www.kaggle.com/datasets/bhanupratapbiswas/superstore-sales/data",
                    "Superstore Sales Data"),
             tags$br(),
             tags$b(),
             tags$p("The main purpose of this app is to demonstrate general functionality
             of the Shiny App environment, and the capability of myself to build something
```

```
          useful within it.  The superstore sales data is a fairly simple dataset,
          however the functionality presented should be translatable to  more complex
          data from your organization."),
          tags$p("The data is broken down in two ways: location and product category.
          Each observation is a product sale which includes a state and zip code,
          as well as a product category, sub category, and the product's name and
          sale price. The app utilizes the following logic:  Categories determine
          sub categories -> Sub categories determine states -> States determine zip
          codes."),
          tags$p("Begin by selecting a category, clicking the 'Set Category' button
                  and then selecting a sub category. After making a category selection,
                  you set your inputs for the next selection (sub category, state,
                  zip code) by a simple click of the appropriate button.  For example,
                  selecting the category 'Furniture' and clicking the 'Set Category'
                  button, will yield only the Furniture sub categories to select from.
                  National data requires only a category and sub category selection.
                  State data requires a state selection.  Zip code data requires a state
                  and zip code selection. Inputs are reset by selecting new categories
                  and clicking the 'Set Category' button. You can select multiple inputs
                  of each selection. When you have made your input selections,
                 Visualizations are produced by clicking the 'Fetch Data' button."),
          tags$p("This app is part 1 of a three part module titled",
          tags$strong("'Practical Applications of the R language for
                 Sales Management'."), "You can view the entirety of the module here at:"),
          tags$a(href="https://github.com/kylestone225", "My GitHub Repository"),
          tags$b(),
          tags$p("I hope you enjoy the app and find it useful!  I really enjoy coding
          and analyzing data using R. Utilizing analytical models and programming
          concepts is a powerful way to protect sales strategy from subjectivity.
          I look forward to discussing how I can help apply these methods within
          your organization, and make your sales department achieve more productive
          successes.")
  ),
  tabPanel("National Data",
          fluidRow(
            column(6,
                   selectInput("natcat", "Category",
                               unique(store_data$Category),
                               multiple = TRUE)
            ),
            column(6,
                   dateRangeInput("natorder", "Select Date Range",
                                  start = min(store_data$Order_Date),
                                  end = max(store_data$Order_Date))
            )
          ),
          actionButton("natcatset", "Set Category",
                       class = "btn-sm btn-primary"),
          fluidRow(
            column(6,
                   selectInput("natsubcat", "Sub Category",
                               unique(store_data$Sub_Category),
                               multiple = TRUE)
```

```
            ),
            column(6,
                   selectInput("streg", "Select Map View",
                               c("State", "Region", "Zip Code"),
                               selected = "State")
            )
          ),
          actionButton("natfetch", tags$strong("Fetch Data"),
                       class = "btn-sm btn-success"),
          plotOutput("natplot"),
          fluidRow(
            column(12,
                   plotlyOutput("natcol")
            )
          ),
          fluidRow(
            column(12,
                   plotOutput("natscat", brush = "natscat_brush")
            )
          ),
          column(12,
                 tableOutput("nattable")
          )

  ),
  tabPanel("State Data",
          fluidRow(
            column(6,
                   selectInput("stcat", "Category", unique(store_data$Category),
                               multiple = TRUE)
                   ),
            column(6,
                   dateRangeInput("storder", "Select Date Range",
                                  start = min(store_data$Order_Date),
                                  end = max(store_data$Order_Date))
            )
          ),
          actionButton("stcatset", "Set Category", class = "btn-sm btn-primary"),
          fluidRow(
            column(6,
                   selectInput("stsubcat", "Sub Category",
                               unique(store_data$Sub_Category),
                               multiple = TRUE)
                   ),
            column(6,
                   selectInput("state", "Select State",
                               unique(store_data$State),
                               multiple = TRUE)
                   )
          ),
          fluidRow(
            column(6,
                   actionButton("stset", "Set States",
```

```
                                class = "btn-sm btn-primary")
                    ),
             column(6,
                    actionButton("stfetch", tags$strong("Fetch State Data"),
                                 class = "btn-sm btn-success")
                    )
           ),
           plotOutput("stplot"),
           fluidRow(
             column(12,
                    plotlyOutput("sttcol")
             )
           ),
           fluidRow(
             column(12,
                    plotOutput("stscat", brush = "stscat_brush")
             )
           ),
           column(12,
                  tableOutput("sttable")
                  )

  ),
  tabPanel("Zip Code Data",
           fluidRow(
             column(6,
                    selectInput("zipcat", "Category",
                                unique(store_data$Category),
                                multiple = TRUE)
                    ),
             column(6,
                    dateRangeInput("ziporder", "Select Date Range",
                                   start = min(store_data$Order_Date),
                                   end = max(store_data$Order_Date))
                    )
             ),
             actionButton("zipcatset", "Set Category",
                          class = "btn-sm btn-primary"),
           fluidRow(column(6,
                           selectInput("zipsubcat", "Sub Category",
                                       unique(store_data$Sub_Category),
                                       multiple = TRUE)
                           ),
                    column(6,
                           selectInput("statezip", "Select State",
                                       unique(store_data$State),
                                       multiple = TRUE)
                    )
           ),
           fluidRow(
             column(6,
                    actionButton("stzipset", "Set States",
                                 class = "btn-sm btn-primary")
```

```
                    ),
             column(6,
                    actionButton("zipset", "Set Zip Codes",
                                 class = "btn-sm btn-primary"))
           ),
           fluidRow(
             column(6,
                    selectInput("zips", "Select Zip Codes",
                                unique(store_data$Postal_Code),
                                multiple = TRUE),
                    actionButton("zipfetch", tags$strong("Fetch Zip Code Data"),
                                 class = "btn-sm btn-success")
                    )
           ),
           plotOutput("zipplot"),
           fluidRow(
             column(12,
                    plotlyOutput("zipcol")
             )
           ),
           fluidRow(
             column(12,
                    plotOutput("zipscat", brush = "zipscat_brush")
             )
           ),
           column(12,
                  tableOutput("ziptable")
           )
    )
  )
)
```

We have now completed the build of our user interface. Below we build out our server, or back end functions of the app. A good way to think about what is happening with the code is that I am providing the app with recipes, the user provides the ingredients, and the app cooks us a meal.

The comments provide some insight into methodology. To gain a deeper understanding I am happy to provide further context, or read through the "Mastering Shiny" textbook linked above.

```
server <- function(input, output, session){
  #create reactive data from main data
  store_values <- reactiveValues(
    df = data.frame()
  )

  store_values$df <- as.data.frame(store_data)

  #Filter the category selection
  nat_cat <- eventReactive(input$natcatset,{
    filter(store_values$df, Category %in% input$natcat)
  })
  #Update the Subcategory selection with available sub categories from above
  observeEvent(input$natcatset,{
    updateSelectInput(session, "natsubcat", "Sub Category",
```

```
                    unique(nat_cat()$Sub_Category))
})
#Final filter collecting subcat and date range info and filtering.
nat_sub_cat <- eventReactive(input$natfetch,{
  filter(nat_cat(), Sub_Category %in% input$natsubcat
         & Order_Date > input$natorder[1]
         & Order_Date < input$natorder[2])
})
#toggle your map view
streg <- reactive(input$streg)
#map plots
output$natplot <- renderPlot({
  if(streg() == "State"){
    nat_sub_cat() |>
      group_by(State, geom) |>
      summarize(tot_st_sales = sum(Sales)) |>
      ggplot() +
      geom_sf(aes(fill = tot_st_sales, color = "grey", geometry = geom),
              show.legend = FALSE) +
      geom_sf_text(aes(label = dollar(tot_st_sales), geometry = geom, size = 4),
                   show.legend = FALSE) +
      labs(title = "Total Sales By State Based On Your Selection") +
      scale_fill_distiller("tot_st_sales", palette="Spectral") +
      scale_color_identity(guide = "legend") +
      scale_size_identity(guide = "legend") +
      theme_classic()
  }else{
    if(streg() == "Region"){
      nat_sub_cat() |>
        ggplot() +
        geom_sf(aes(fill = region_sales, color = "grey", geometry = geom)) +
        labs(title = "Total Sales By Region (unfiltered)") +
        scale_fill_distiller("region_sales", palette="Spectral",
                             label = label_comma()) +
        scale_color_identity(guide = "legend") +
        theme_classic()
    }else{
      if(streg() == "Zip Code"){
        nat_sub_cat() |>
          group_by(Postal_Code, zip_geometry) |>
          summarize(tot_z_sales = sum(Sales)) |>
          ggplot() +
          geom_sf(aes(fill = tot_z_sales, geometry = zip_geometry),
                  show.legend = FALSE) +
          labs(title = "Zip Codes Containing Sales of Your Selections") +
          scale_color_identity(guide = "legend") +
          theme_minimal()
      }
    }
  }
})
#SF geoms are lists and not conducive to the interactive features of shiny. Below an identical
#set of data without the SF geoms is filtered for plotting the blow charts and the brush table.
```

```
plot_values <- reactiveValues(
  df = data.frame()
)

plot_values$df <- as.data.frame(sd_plotting)

plot_cat <- eventReactive(input$natcatset,{
  filter(plot_values$df, Category %in% input$natcat)
})

plot_sub_cat <- eventReactive(input$natfetch,{
  filter(plot_cat(), Sub_Category %in% input$natsubcat
         & Order_Date > input$natorder[1]
         & Order_Date < input$natorder[2])
})

#Scatter chart used for your brushed points
output$natscat <- renderPlot({
  plot_sub_cat() |>
    ggplot(aes(x = Order_Date, y = Sales, colour = Category)) +
    geom_point() +
    labs(title = "Quickly Visualize Your Largest Sales",
         subtitle = "Click and Drag Your Cursor Around a Group of Points to View Data") +
    theme_classic() +
    theme(axis.text.x = element_blank())

})

#Plotly geom_col chart
output$natcol <- renderPlotly({
  plot_sub_cat() |>
    ggplot(aes(Category, Sales, fill = Sub_Category)) +
    geom_col(aes(label = Product_Name), show.legend = FALSE) +
    labs(title = "Total Sales By Category: Hover To View") +
    xlab("") + ylab("") +
    scale_y_continuous(labels = label_comma()) +
    theme_classic()
})
# Brushed points
output$nattable <- renderTable({
  brushedPoints(plot_sub_cat()[c(3,7,10,11,15:18)], brush = input$natscat_brush)
})

#State level data
#Filter the category selection
st_cat <- eventReactive(input$stcatset,{
  filter(store_values$df, Category %in% input$stcat)
})
#Update the Subcategory selection with available sub categorys from above
observeEvent(input$stcatset,{
  updateSelectInput(session, "stsubcat", "Sub Category",
                    unique(st_cat()$Sub_Category))
})
```

```
st_subcat <- eventReactive(input$stset,{
  filter(st_cat(), Sub_Category %in% input$stsubcat)
})

observeEvent(input$stset,{
  updateSelectInput(session, "state", "Select State",
                    unique(st_subcat()$State))
})
#Final filter collecting subcat and date range info and filtering.
st_sub_final <- eventReactive(input$stfetch,{
  filter(st_subcat(), State %in% input$state &
           Order_Date > input$storder[1] &
           Order_Date < input$storder[2])
})

# Plot of selected state
output$stplot <- renderPlot({
  st_sub_final() |>
    group_by(State, geom) |>
    summarize(tot_st_sales = sum(Sales)) |>
    ggplot() +
    geom_sf(aes(fill = tot_st_sales, color = "grey", geometry = geom),
            show.legend = FALSE) +
    geom_sf_text(aes(label = dollar(tot_st_sales), geometry = geom, size = 4),
                 show.legend = FALSE) +
    labs(title = "Total Sales By State Based On Your Selection") +
    scale_fill_distiller("tot_st_sales", palette="Spectral") +
    scale_color_identity(guide = "legend") +
    scale_size_identity(guide = "legend") +
    theme_classic()
})


st_plot_cat <- eventReactive(input$stcatset,{
  filter(plot_values$df, Category %in% input$stcat)
})

st_plot_sub_cat <- eventReactive(input$stfetch,{
  filter(st_plot_cat(), Sub_Category %in% input$stsubcat &
           Order_Date > input$storder[1] &
           Order_Date < input$storder[2] &
           State %in% input$state)
})


# Plotly geom_col
output$sttcol <- renderPlotly({
  st_plot_sub_cat() |>
    ggplot(aes(Category, Sales, fill = Sub_Category)) +
    geom_col(aes(label = Product_Name), show.legend = FALSE) +
    labs(title = "Total Sales By Category: Hover To View") +
    xlab("") + ylab("") +
    scale_y_continuous(labels = label_comma()) +
```

```
    theme_classic()
})

# brushed points
output$stscat <- renderPlot({
  st_plot_sub_cat() |>
    ggplot(aes(x = Order_Date, y = Sales, colour = Category)) +
    geom_point() +
    labs(title = "Quickly Visualize Your Largest Sales",
         subtitle = "Click and Drag Your Cursor Around a Group of Points to View Data") +
    theme_classic() +
    theme(axis.text.x = element_blank())
})

# brushed points table
output$sttable <- renderTable({
  brushedPoints(st_plot_sub_cat()[c(3,7,10,11,15:18)], brush = input$stscat_brush)
})

# OK lets do our final page for zip code data
#Filter the category selection
zip_cat <- eventReactive(input$zipcatset,{
  filter(store_values$df, Category %in% input$zipcat)
})
#Update the Subcategory selection with available sub categorys from above
observeEvent(input$zipcatset,{
  updateSelectInput(session, "zipsubcat", "Sub Category",
                    unique(zip_cat()$Sub_Category))
})

zip_subcat <- eventReactive(input$stzipset,{
  filter(zip_cat(), Sub_Category %in% input$zipsubcat)
})

observeEvent(input$stzipset,{
  updateSelectInput(session, "statezip", "Select State",
                    unique(zip_subcat()$State))
})

zip_subcat_state <- eventReactive(input$zipset,{
  filter(zip_subcat(), State %in% input$statezip)
})

observeEvent(input$zipset,{
  updateSelectInput(session, "zips", "Select Zip Codes",
                    unique(zip_subcat_state()$Postal_Code))
})


#Final filter collecting subcat and date range info and filtering.
zip_sub_final <- eventReactive(input$zipfetch,{
  filter(zip_subcat_state(), Order_Date > input$storder[1] &
           Order_Date < input$storder[2] &
```

```
            Postal_Code %in% input$zips)
})

#plot zip codes
output$zipplot <- renderPlot({
  zip_sub_final() |>
    group_by(Postal_Code, zip_geometry) |>
    summarize(tot_z_sales = sum(Sales)) |>
    ggplot() +
    geom_sf(aes(fill = tot_z_sales, geometry = zip_geometry), show.legend = FALSE) +
    geom_sf_text(aes(label = dollar(tot_z_sales), geometry = zip_geometry, size = 4),
                 show.legend = FALSE) +
    labs(title = "Zip Codes Containing Sales of Your Selections") +
    scale_fill_distiller("tot_z_sales", palette="Spectral") +
    scale_color_identity(guide = "legend") +
    theme_minimal()
})

# zip code plotting data without the geom lists
zip_plot_cat <- eventReactive(input$zipcatset,{
  filter(plot_values$df, Category %in% input$zipcat)
})

zip_plot_sub_cat <- eventReactive(input$zipfetch,{
  filter(zip_plot_cat(), Sub_Category %in% input$zipsubcat &
           Order_Date > input$ziporder[1] &
           Order_Date < input$ziporder[2] &
           Postal_Code %in% input$zips)
})


# PLotly columns
output$zipcol <- renderPlotly({
  zip_plot_sub_cat() |>
    ggplot(aes(Category, Sales, fill = Sub_Category)) +
    geom_col(aes(label = Product_Name), show.legend = FALSE) +
    labs(title = "Total Sales By Category: Hover To View") +
    xlab("") + ylab("") +
    scale_y_continuous(labels = label_comma()) +
    theme_classic()
})

# Brushed zip points
output$zipscat <- renderPlot({
  zip_plot_sub_cat() |>
    ggplot(aes(x = Order_Date, y = Sales, colour = Category)) +
    geom_point() +
    labs(title = "Quickly Visualize Your Largest Sales",
         subtitle = "Click and Drag Your Cursor Around a Group of Points to View Data") +
    theme_classic() +
    theme(axis.text.x = element_blank())
})
```

```
  # Brushed table
  output$ziptable <- renderTable({
    brushedPoints(zip_plot_sub_cat()[c(3,7,10,11,15:18)], brush = input$zipscat_brush)
  })


}
```

So, this is what the guts of a complete and usable Shiny App look like!

Please use the app here: https://kylestone225.shinyapps.io/Sales_Data_Analysis/

Working with data and building apps like the one shown here is a fun and rewarding exercise for me. Having this functionality as a sales department manager allows me to make the most productive use of your organizations sales data. As department manager I will know exactly what the sales needs are, and build pragmatic and productive functionality for use within the department. Outsourcing this functionality is unnecessarily expensive. Even internal data/IT teams will create sub par applications in this context mostly due to asymmetry of experience.

Thank you for your interest in this work. I look forward to speaking with you further regarding implementing Shiny Apps, and other aspects of "Practical Applications of the R Language for Sales Management" modules within your sales function.

Kyle Stone

215.880.8774

kylestone225@gmail.com