R Shiny Tips and Tricks: An App Store Project Demo

Author: Bohdan Metenko

Thursday, August 15th, 2019 < > 6:00pm

PhillyR Meetup

Hosted by Slalom Consulting,

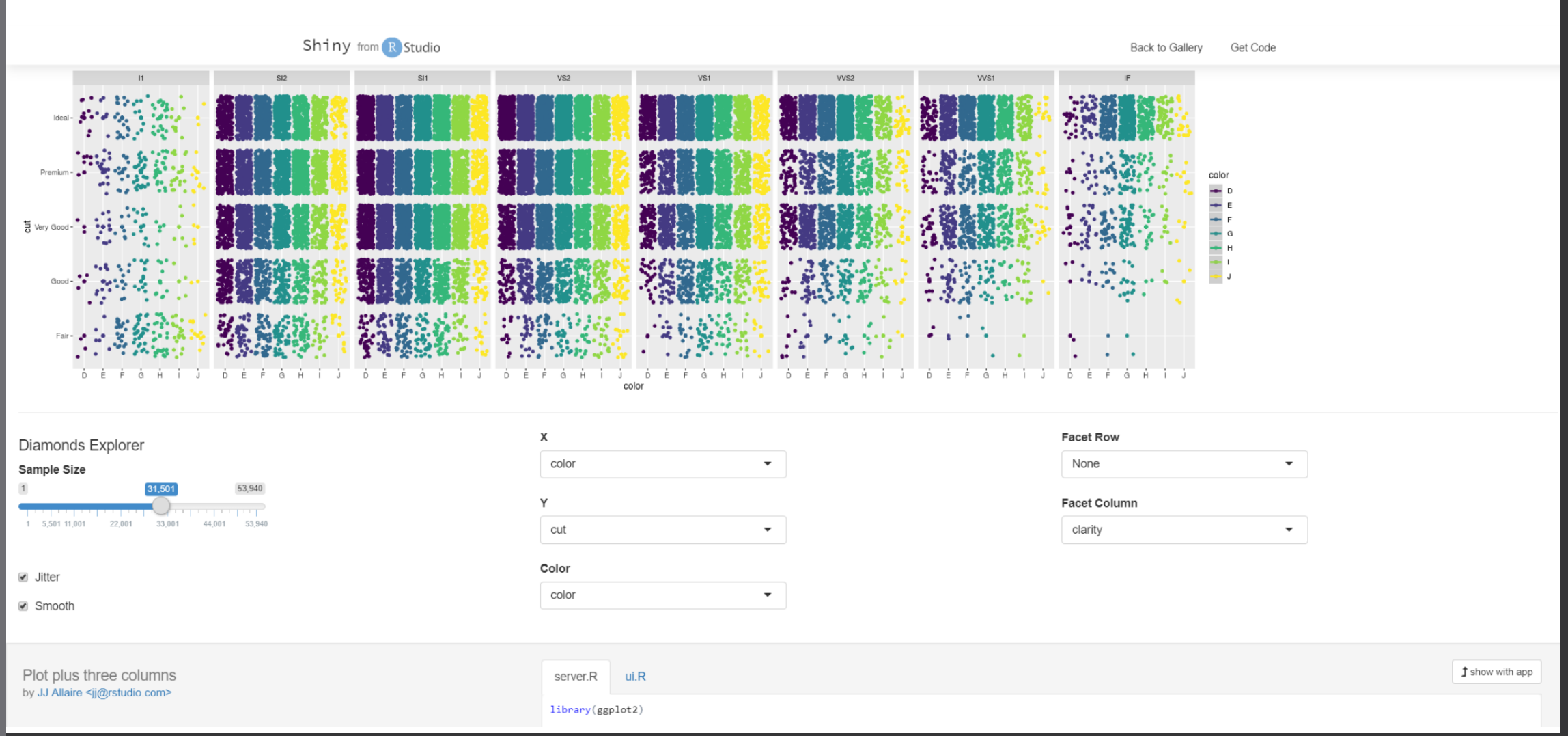
100 N 18th St #2000, Philadelphia, PA

Talk Outline

0. Intro to Shiny
1. App Overview
2. CSS styling and options.
3. Mobile considerations and splitting up plots.
4. Smaller Tips
5. Travis CI and requirements. (Continuous Deployment and Integration.)

Introduction to Shiny

- Shiny is an R package that helps build web-based applet-like functionality without the need for JavaScript, CSS, or HTML knowledge. (It can help though.)
- Enables interactivity for both you and the end-user.
- Easily web-publishable.
- Maintained by RStudio.
- Is the basis for packages like ggthemeassist and esquisse.
- Tends to be split into ui.R and server.R files.
- Example: Next slide.



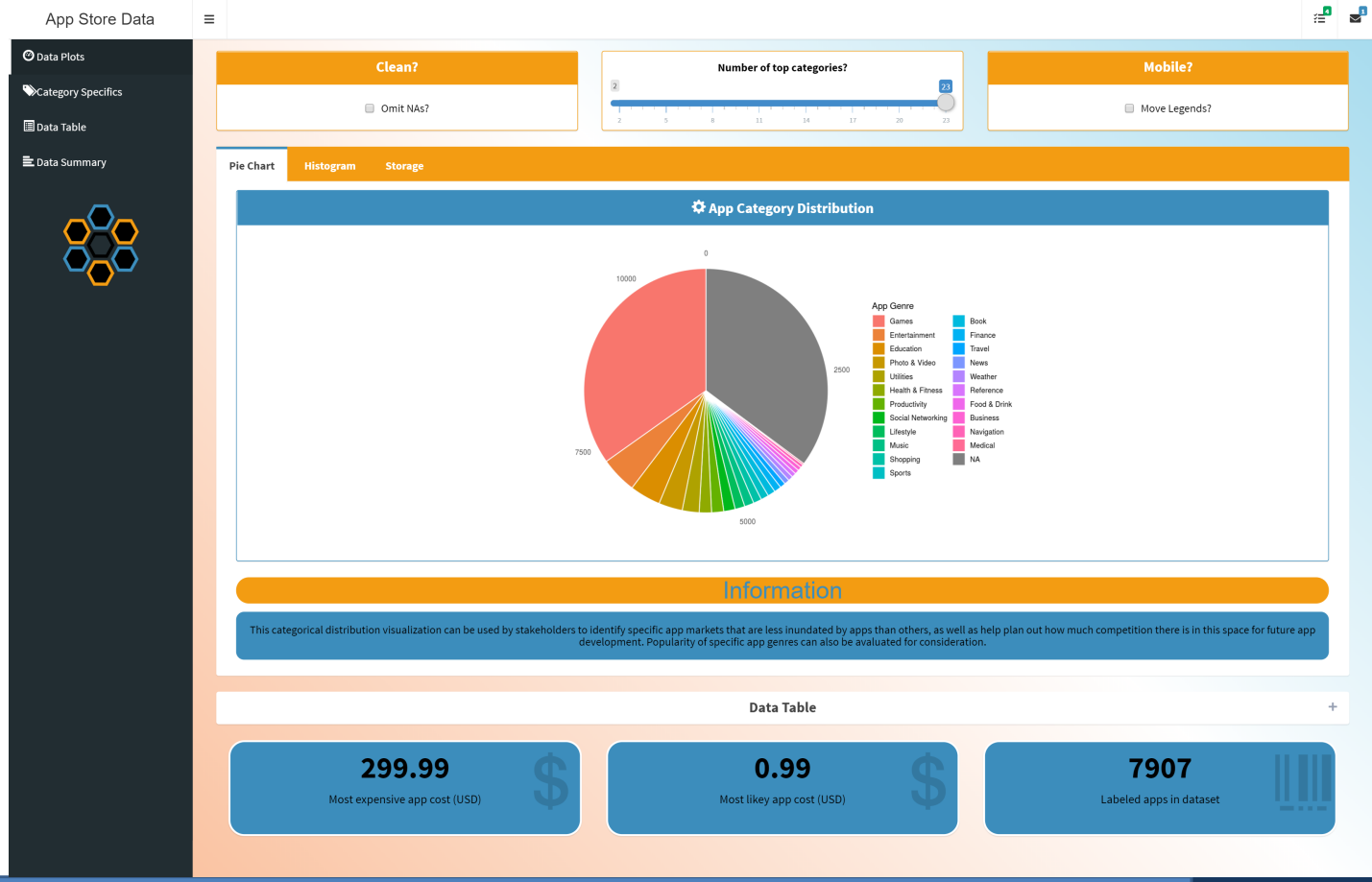
Example: https://shiny.rstudio.com/gallery/plot-plus-three-columns.html

My Current App State

Code available at
https://www.github.com/bmetenko/RShinyAppStore

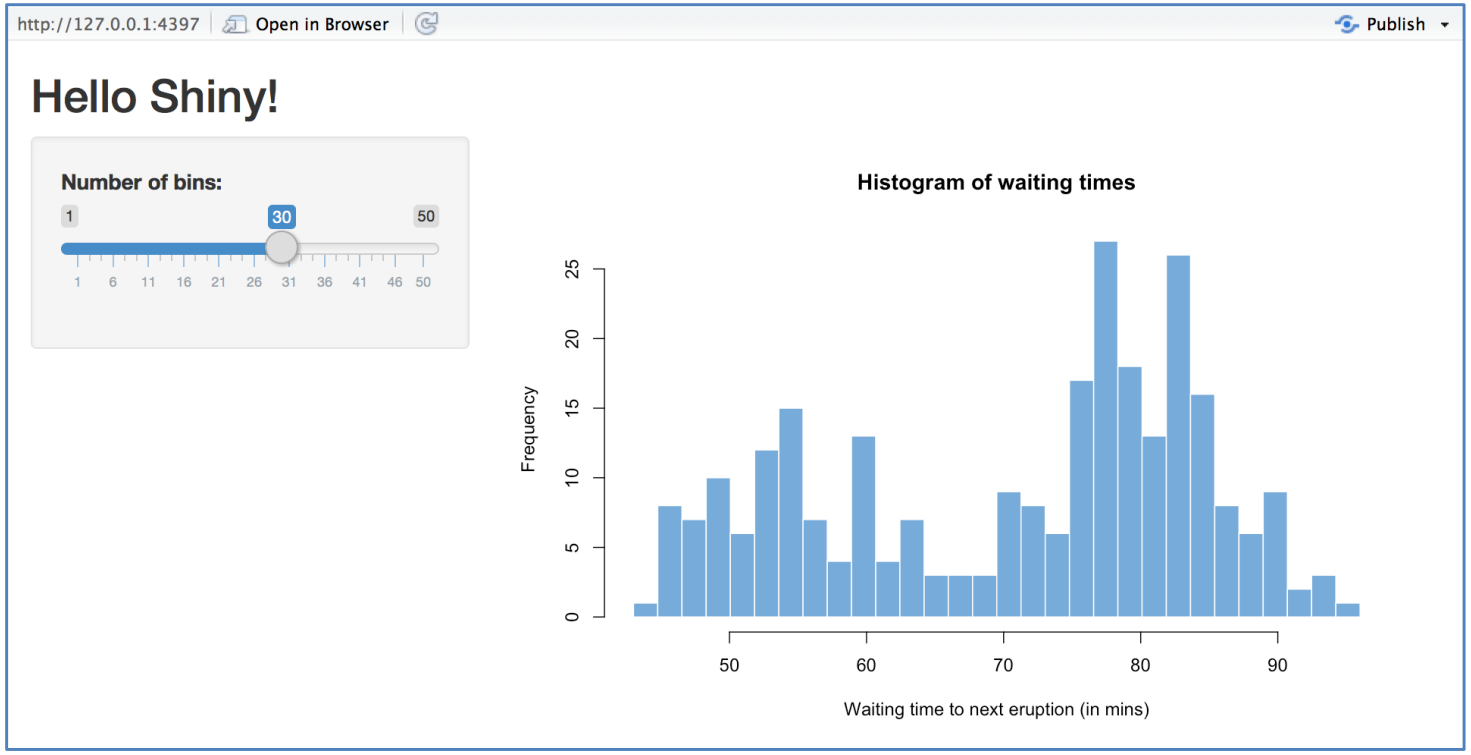
(Note: please run package_check.R before running the app from RStudio on your machine.)

A Live version is published on
https://bmetenko.shinyapps.io/RShinyAppStore/
(Limited to the Free tier, so please everyone don't jump on there at once.)



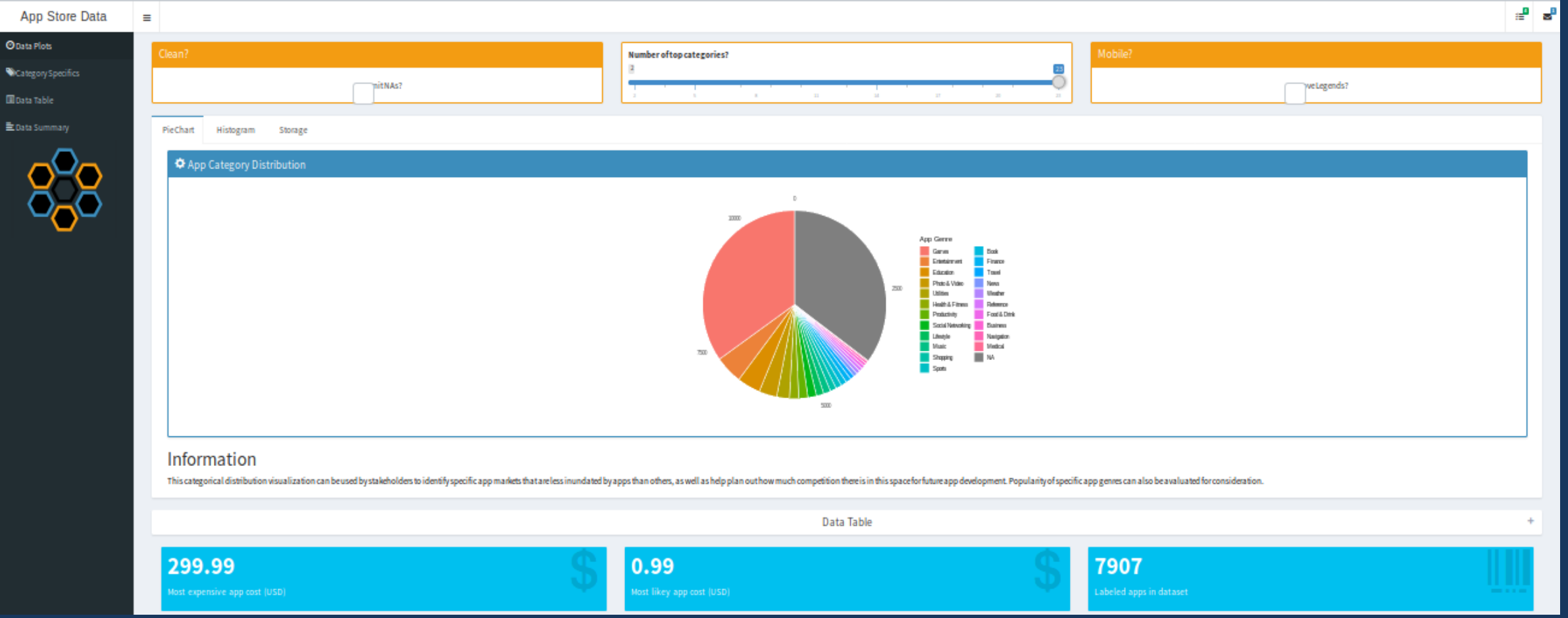
Purpose: Visualization and Presentation of Apple App Store Data from 2017

(App Sizes, App Ratings, App Genre)



Tips and Tricks #1: CSS styling

- Normal Shiny is plain.
- (Kind of like this PowerPoint)
- Overriding CSS can change that.
- It helps you change colors, add hover over effects, create background gradients and the like.
- A little bit of CSS knowhow is useful, but it is very easy for anyone to pick up.



Without CSS

Caveat: Also using shinydashboard library for page theme.

App Store Data

Data Plots
Category Specifics
Data Table
Data Summary

Clean?

NAs?

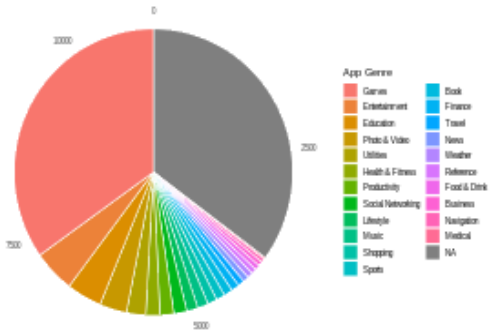
Number of top categories?

Mobile?

Legends?

Pie Chart | Histogram | Storage

App Category Distribution



Information

This categorical distribution visualization can be used by stakeholders to identify specific app markets that are less inundated by apps than others, as well as help plan out how much competition there is in this space for future app development. Popularity of specific app genres can also be evaluated for consideration.

Data Table

299.99
Most Expensive App Cost (USD)

0.99
Most likey app cost (USD)

7907
Labeled apps in dataset

With CSS

Tips and Tricks #1 | CSS styling

In Practice:

Multiple choices for CSS:

- Inside specific elements of the app

```
Box_Clean <- box(
  width = "50%",
  height = "100px",
  align = "center",
  uiOutput(outputId = "input1"),
  solidHeader = T,
  title = "Clean?",
  status = "warning",

  style = "background-color: red;"
)
```

Tips and Tricks #1 | CSS styling

In Practice:

Multiple choices for CSS:

- Inside specific elements of the app

- In the header element of your main app body.

```
#### SIDEBAR ####
Main_Sidebar <- dashboardSidebar(
  #### MAIN CSS ####
  HTML(
    '<style>
    /* Navigation Bar */
    ul.nav.nav-tabs { align: center; width: 100%;
    background-color: #f39c12;
    color: white;
    }

    ul.nav.nav-tabs a {
    color: white;
    font-weight: bold;
    }
```

Tips and Tricks #1 | CSS styling

In Practice:

Multiple choices for CSS:

- Inside specific elements of the app
- In the header element of your main app body.
- Personal and separate stylesheet and reference. [www/main.css]

```
#### SIDEBAR ####
Main_Sidebar <- dashboardSidebar(
  #### MAIN CSS ####
  tags$link(rel = "stylesheet",
            type = "text/css",
            href = "main.css"),
```

```
/* Navigation Bar */

ul.nav.nav-tabs {
    text-align: center;
    width: 100%;
    font-weight: bolder;
    font-size: 18px;
    background-color: #f39c12;
    color: white;
}
```

Tips and Tricks #1 | CSS styling

In Practice:

Multiple choices for CSS:

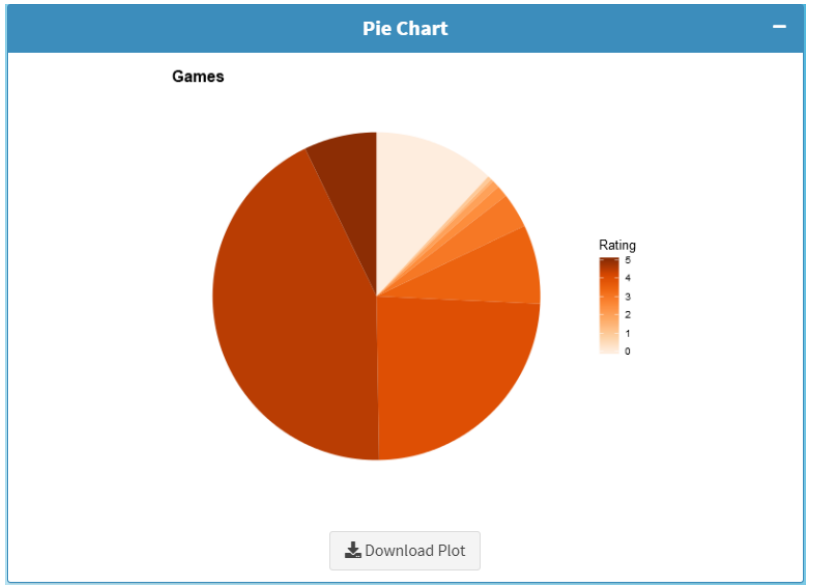
- External Specialized/Simplified Stylesheet
- Tailwind CSS using class assignment

```
#### SIDEBAR ####
Main_Sidebar <- dashboardSidebar(
  #### MAIN CSS ####
  HTML('<link
  href="https://unpkg.com/tailwindcss@^1.0/dist/tailwind.min.css"
  rel="stylesheet">
       '),
```

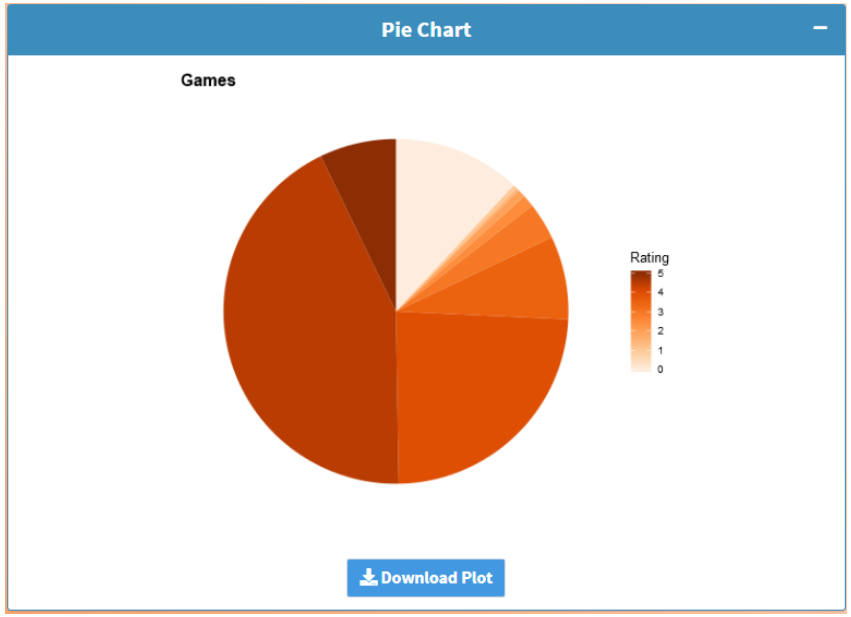
```
#### Data Download ####
Down_Cat_Pie <-
  downloadButton(label = "Download Plot",
                 outputId = "Pie_Download",
                 class="bg-blue-500
                 hover:bg-blue-600
                 text-white font-bold
                 py-2 px-4 rounded")
```

More info at https://tailwindcss.com/
And https://nerdcave.com/tailwind-cheat-sheet

```
#### SIDEBAR ####
Main_Sidebar <- dashboardSidebar(
  #### MAIN CSS ####
  HTML('<link
  href="https://unpkg.com/tailwindcss@^1.0/dist/tailwind.min.css"
  rel="stylesheet">
       '),
#### Data Download ####
Down_Cat_Pie <-
  downloadButton(label = "Download Plot",
                 outputId = "Pie_Download",
                 class="bg-blue-500
                 hover:bg-blue-600
                 text-white font-bold
                 py-2 px-4 rounded")
```

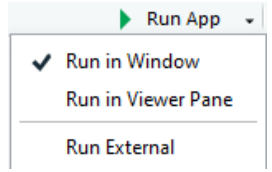


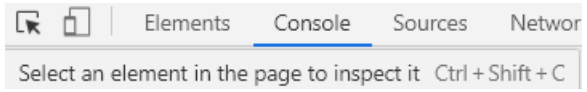
Caveats:
Loading this stylesheet will override any default shiny CSS properties for elements like a, h1 to h5 tags.
May also take longer to load.



Finding the class tag you want to assign CSS properties to.

- Run your shiny app externally
- Press F12
- Click
- and select the element
- Or
- Right click your element and click inspect





Add something new to your CSS file.

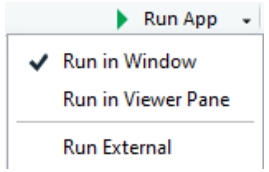
```
.box-header .box-title:hover {

color: black;
padding: 0.1em;

}
```

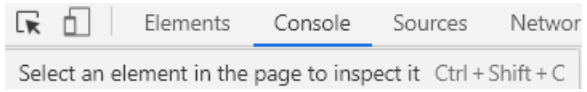

```
.box-header .box-title {
  font-weight: bold;
}
.box-header .box-title, .box-header>.fa, .box-header>.glyphicon, .box-header>.ion {
  display: inline-block;
  font-size: 18px;
  margin: 0;
  line-height: 1;
}
```

#1

Finding the class tag you want to assign CSS properties to.

- Run your shiny app externally
- Press F12
- Click
- and select the element
- Or
- Right click your element and click inspect





Add something new to your CSS file.

```
.box-header .box-title:hover {

color: black;
padding: 0.1em;

}
```

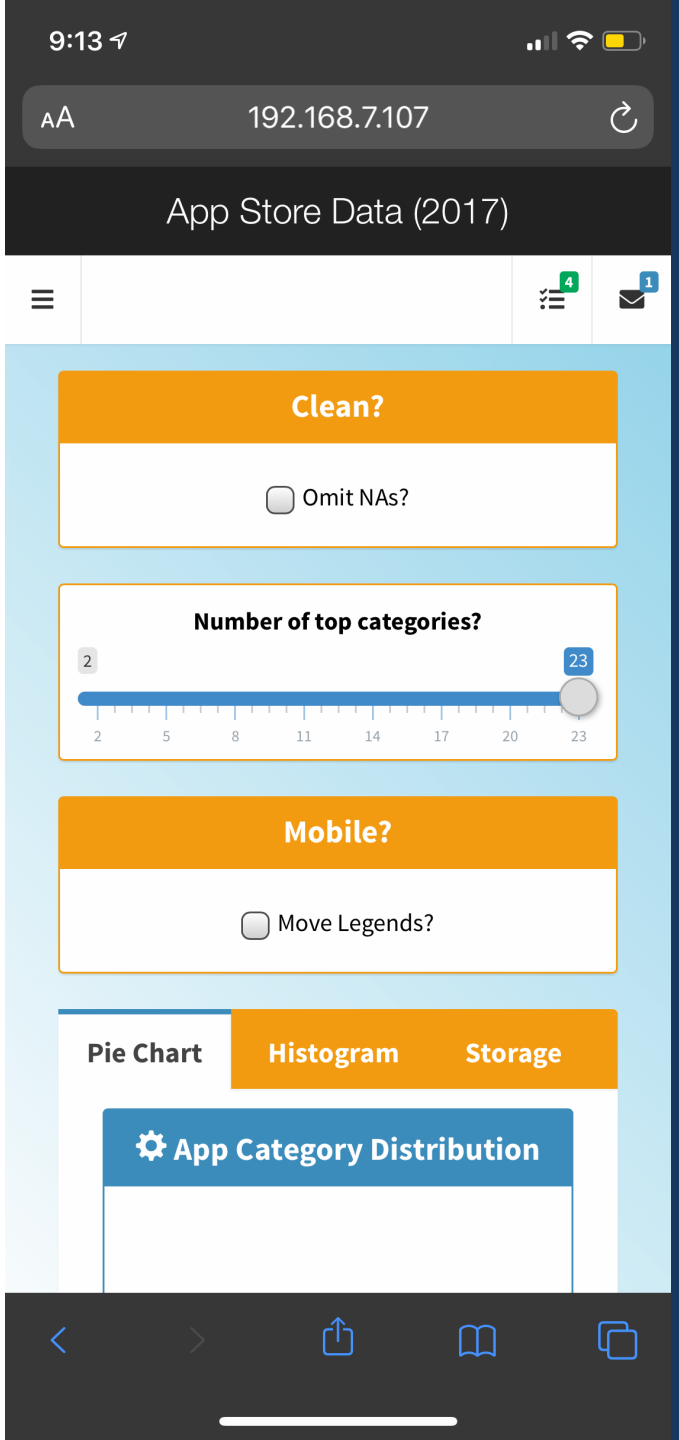

```
.box-header .box-title {
  font-weight: bold;
}
.box-header .box-title, .box-header>.fa, .box-header>.glyphicon, .box-header>.ion {
  display: inline-block;
  font-size: 18px;
  margin: 0;
  line-height: 1;
}
```

Important notes

- If you change some color or property that is locked in by the stylesheet provided by shiny, it may be useful to override it with:
- color: blue!important;
- Another important point is that once you change and apply styles to one class of objects, they will all have those styles.
- If this is not useful:
  - Make a new class for the element you care about and style it using .new-class {}
- Adding JavaScript, jQuery, and HTML follows some of the rules mentioned, but would deserve their own talk and are worth learning more about.
- CSS is just more manageable for most people.

Remember your cell phone?

- Your shiny applications can be published to online services like shinyapps.io where they can be accessed from most devices by anyone.
- To test how they will look on other machines, you can run your shiny app in an R session using
  - Prereq = { library(shiny) }
  - **runApp(host="0.0.0.0", port=5050)**
  - You'll have to check for your IP address using ipconfig or ifconfig then connect to the app via a webpage URL such as 192.168.1.107:5050
  - If the device you are using is connected to the same network, then there shouldn't be any issues.



This helps us find interesting problems…

- Such as incorrect plot scaling.
- But the fix leads us into two more shiny tips:
- renderUI (on the fly) based on conditions
  - Toggle a new plot to be rendered based on mobile checkbox state.

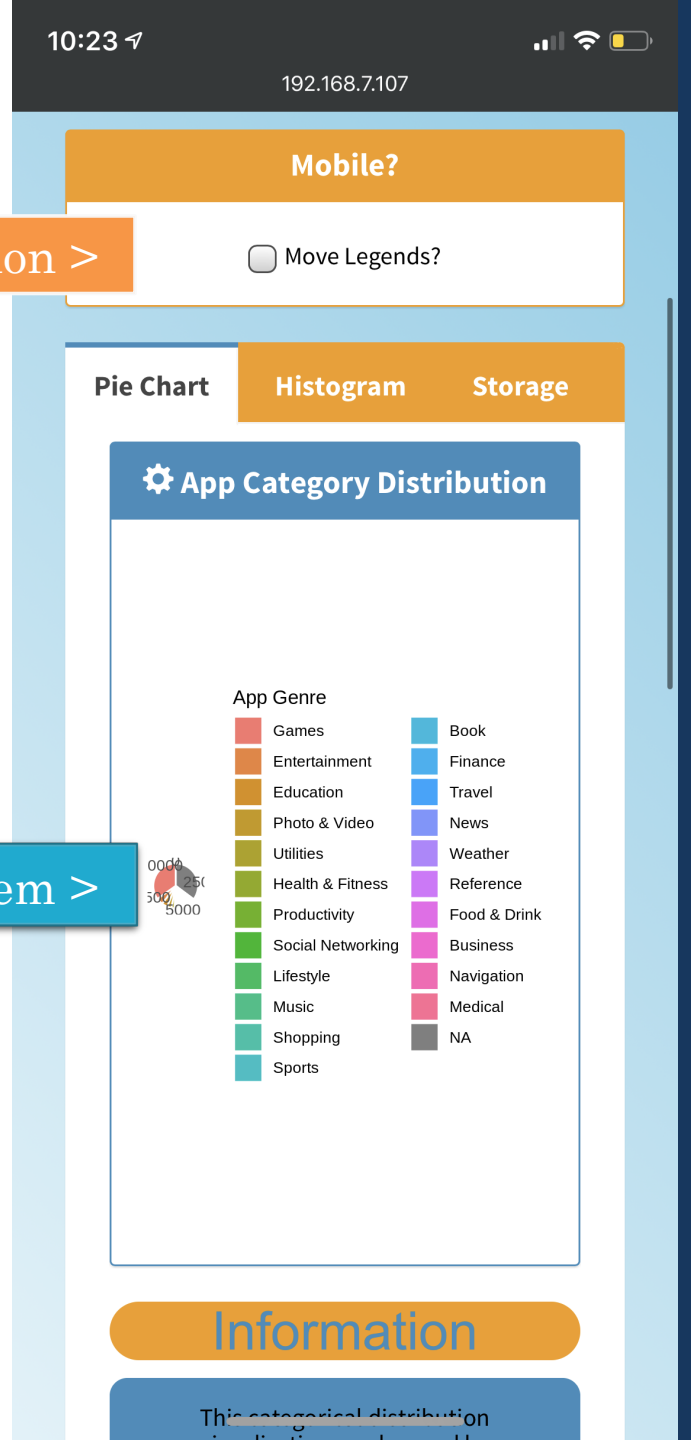
```
output$Legend_Pie_Mobile <- renderUI({
  if (!input$mobileCheck) {
    return()
  } else {
    box(plotOutput(
      outputId = "Pie_Legend",
      height = "250px"
    ),
    width = "100%")
  }
})
```

  - Said plot only contains the legend. (cowplot library required.)

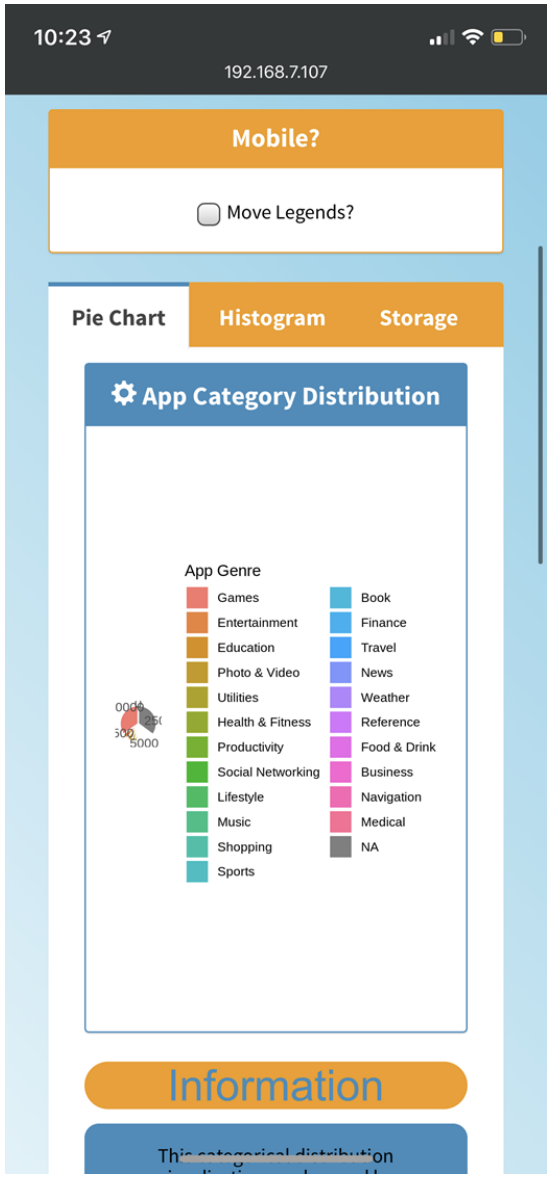
```
bp # ggplot object from first pie chart render

get_legend(bp) %>% plot_grid() # extracted and plotted legend grob
```

Solution >

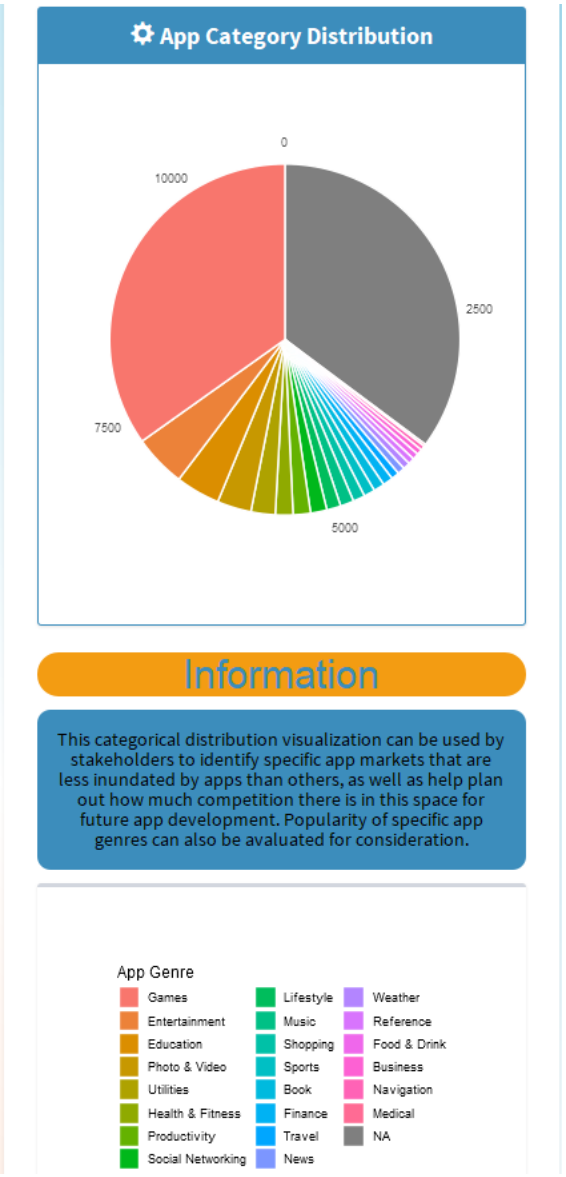
Problem >



#2


10:23
192.168.7.107
Mobile?
Move Legends?
Pie Chart
Histogram
Storage
App Category Distribution
App Genre
Games
Entertainment
Education
Photo & Video
Utilities
Health & Fitness
Productivity
Social Networking
Lifestyle
Music
Shopping
Sports
Book
Finance
Travel
News
Weather
Reference
Food & Drink
Business
Navigation
Medical
NA
5000
Information

VS.

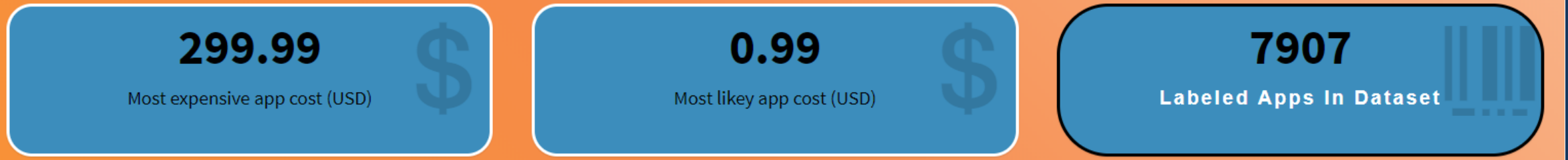

App Category Distribution
0
10000
2500
7500
5000
Information
This categorical distribution visualization can be used by stakeholders to identify specific app markets that are less inundated by apps than others, as well as help plan out how much competition there is in this space for future app development. Popularity of specific app genres can also be avaluated for consideration.
App Genre
Games
Entertainment
Education
Photo & Video
Utilities
Health & Fitness
Productivity
Social Networking
Lifestyle
Music
Shopping
Sports
Book
Finance
Travel
News
Weather
Reference
Food & Drink
Business
Navigation
Medical
NA

Smaller tips

```
output$plot3 <- renderCachedPlot({
  validate(need(input$catChoice != "", "Loading..."))
  # render plot here ::::

}, cacheKeyExpr = {input$catChoice})
```

- Plot caching.
  - Using renderCachedPlot(…) instead of renderPlot(…)
  - Specifying hashable elements for caching based on list of inputs
  - https://shiny.rstudio.com/articles/plot-caching.html
- Taglists to plot multiple UI elements.
- Highlighting information using KPI boxes.
- Pivot table and scrollability.
- [[ShinyWidgets]]?
- Maintaining a color theme.

```
#### KPI Container ####
KPI_Container <- tagList(
  valueBox(
    value = 299.99,
    width = 4,
    icon = icon("usd", lib = "glyphicon"),
    subtitle = "Most expensive app cost (USD)"
  ),
  valueBox(
    value = 0.99,
    width = 4,
    icon = icon("usd", lib = "glyphicon"),
    subtitle = "Most likey app cost (USD)"
  ),
  valueBox(
    value = 7907,
    width = 4,
    icon = icon("barcode", lib = "glyphicon"),
    "Labeled apps in dataset"
  )
)
```



Smaller tips

- Plot caching.
  - Using renderCachedPlot(…) instead of renderPlot(…)
  - Specifying hashable elements for caching based on list of inputs
  - https://shiny.rstudio.com/articles/plot-caching.html
- Taglists to plot multiple UI elements.
- Highlighting information using KPI boxes.
- Pivot table and scrollability. (more niche)
  - Without these modifications, interactions with the rPivotTable tables overflow beyond page borders, and text isn't visible in the main viewport.
- Maintaining a color theme. (less niche)

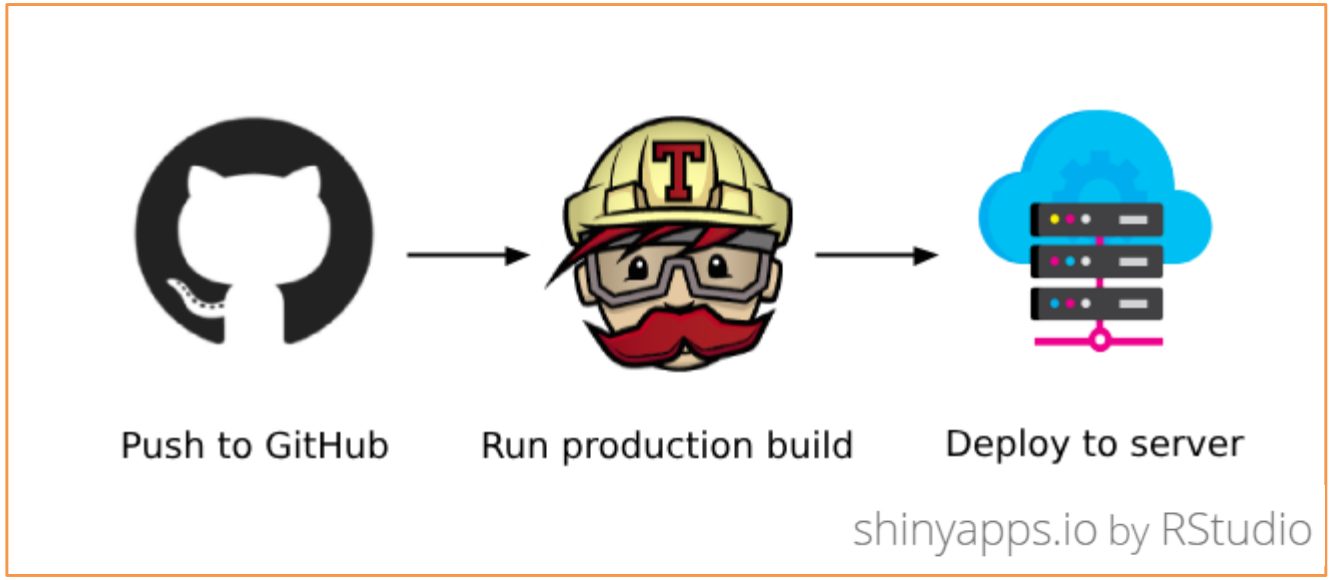
```
.rpivotTable {
    overflow-x: scroll;
    overflow-y: scroll;
    text-decoration: none;
}

.pvtAxisContainer, .pvtVals {
    text-align: left;
}
```

Travis CI
(Continuous Deployment and Integration)

- Basic requirements:
  - YAML configuration file
  - Deploy script
  - Environmental Variables

https://travis-ci.org

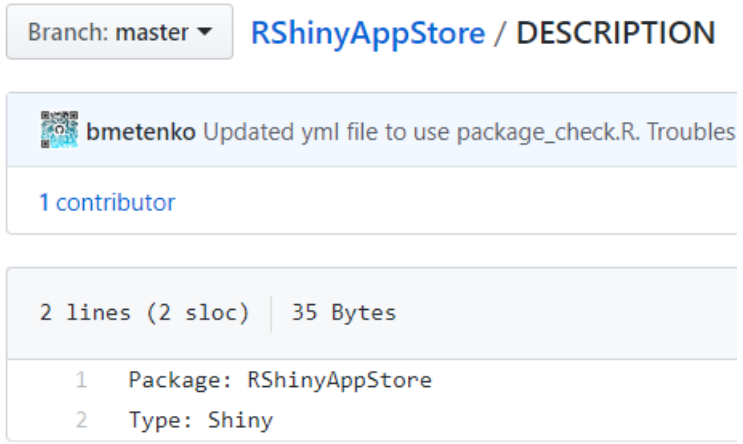
- Before starting, make sure to hook up your GitHub account with your Travis CI account and your shinyapps.io account.



```yaml
language: r
r: release
sudo: required
warnings_are_errors: false

cache:
  packages: true

git:
  depth: false

r_packages:
  - shiny
  - ggplot2
  - dplyr
  - jpeg
  - shinydashboard
  - httr
  - jsonlite
  - scales
  - forcats
  - cowplot
  - tidyverse
  - rsconnect
  - ggpubr
  - sqldf
  - RColorBrewer
  - rpivotTable

script:
  - R -f deploy.R
```

.travis.yml file

- Simple file specifying to Travis-CI what to install.
- R version and package versions can be specified.
- Last entry in the list is the shell deploy script to upload and re-deploy the shiny app.
- Another Requirement:
- A DESCRIPTION file:



Deploy script (deploy.R)

- Doesn't have to be much more than

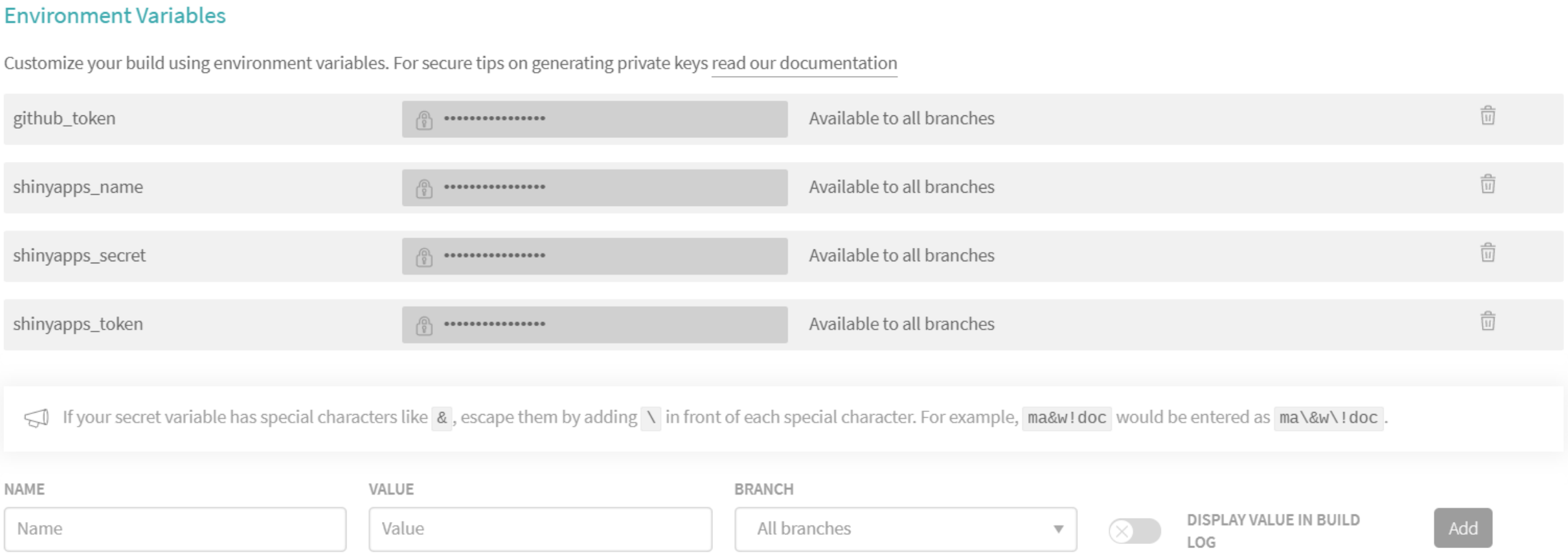
```
library(rsconnect)
# source("package_check.R")

setAccountInfo(name = Sys.getenv("shinyapps_name"),
               token = Sys.getenv("shinyapps_token"),
               secret = Sys.getenv("shinyapps_secret"))

deployApp()
```

- Note the commented-out package_check.R file.
  - These packages are installed based on the YAML config file.

- Travis can also be used to perform unit testing while/before deploying.

Environmental Variables

- You want to keep these private, otherwise anyone can publish anything to your account on shinyapps.io:
- Found and editable on Travis CI settings of your repo.



Environmental Variables

- You want to keep these private, otherwise anyone can publish anything to your account on shinyapps.io:
- Found and editable on Travis CI settings of your repo.
- Tokens to add in can are listed here:
- https://www.shinyapps.io/admin/#/tokens
- (Not shown here for privacy reasons but refer to the previous slide and the deploy script after clicking show and show secret on this webpage.)

```
library(rsconnect)
# source("package_check.R")

setAccountInfo(name = Sys.getenv("shinyapps_name"),
               token = Sys.getenv("shinyapps_token"),
               secret = Sys.getenv("shinyapps_secret"))

deployApp()
```

More information at: https://docs.travis-ci.com/user/languages/r/

Please remember to uncheck "display values in build log"

After everything is setup and saved, you can git commit and push your code and the Linux container that Travis runs should rebuild, test, and publish the changes you made.

Questions?

Ask now, or email me at
bohdanmetenko@outlook.com