Data Visualisation

Chapter 9: Building Apps

Dr James Baglin

How to use these slides

Viewing slides…

- Press ‘f’ enable fullscreen mode
- Press ‘o’ or ‘Esc’ to enable overview mode
- Pressing ‘Esc’ exits all of these modes.
- Hold down ‘alt’ and click on any element to zoom in. ‘Alt’ + click anywhere to zoom back out.
- Use the Search box (top right) to search keywords in presentation

Printing slides…

- Click here to open a printable version of these slides.
- Right click and print from browser or save as PDF (e.g. Chrome)

Shiny Apps

Histograms - Exploring the Effect of Sample Size and Bin Number

Select a sample size to randomly sample. Click sample to draw. If you draw multiple samples of the same sample size, you can see how histograms can vary from sample to sample.

Sample size:

10 10,000

10 1,009 2,008 4,006 6,004 8,002 10,000

Sample

Adjust the number of bins to see how it impacts the histogram's appearance.

Number of bins:

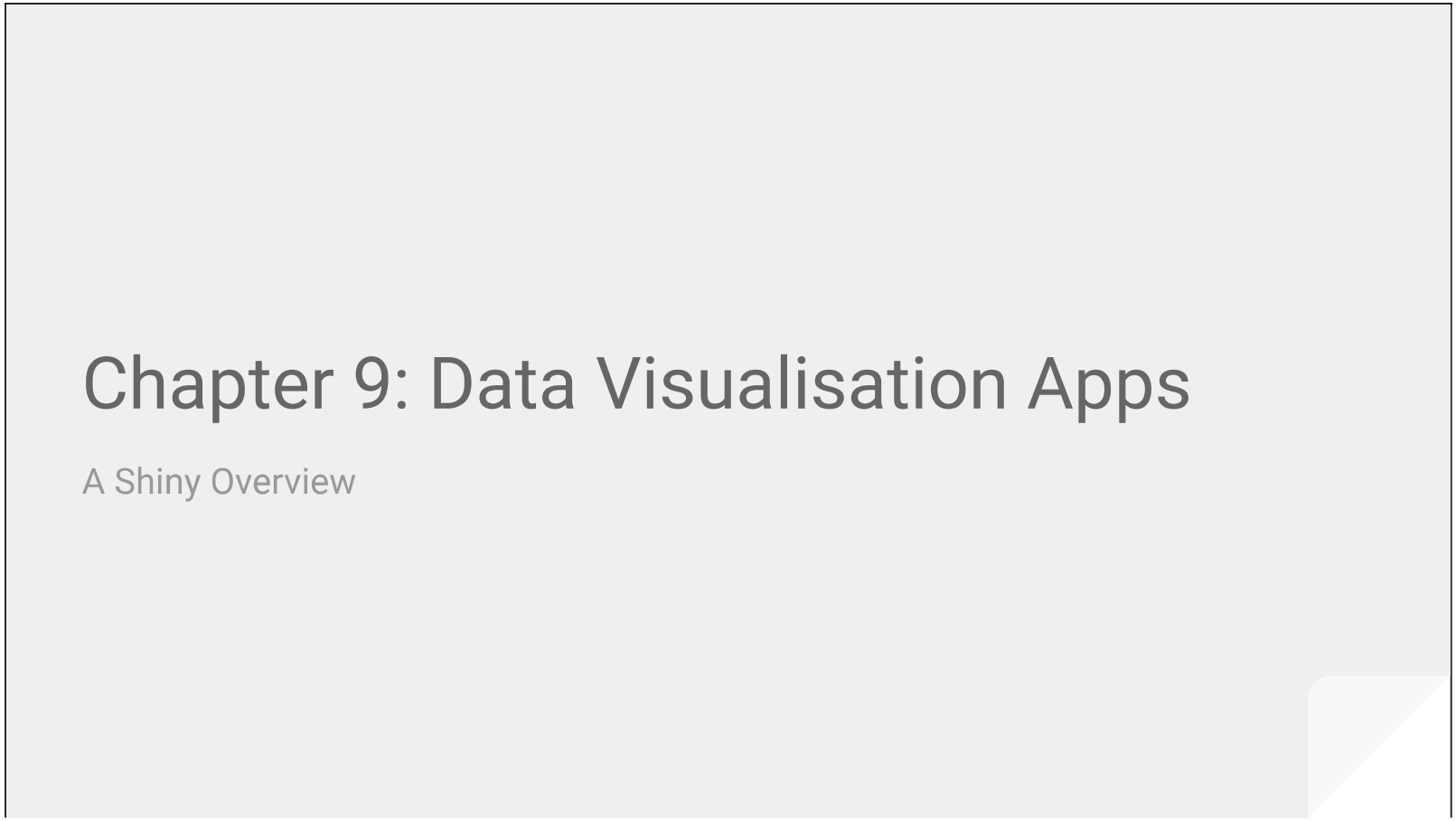
1 30 100

1 11 21 31 41 51 61 71 81 91 100

The red line is the target population distribution and the blue line is the sample density estimate.

Histograms perform best for large samples that can support many bins.

Introduction to Shiny



Population Pyramid App

- Let's use Shiny to replicate the Australian Bureau of Statistics (ABS) Population Pyramid app.
- This app will show you how to handle a dataset.
- The app shows how the age and gender distribution of the Australian population has grown and changed over time.
- The `aus_age_gender_hist` dataset was extracted from the ABS catalogue number 3101.0 - Australian Demographic Statistics, Sep 2015 data cubes.
- The dataset contains the frequency distribution of Australia by age category and gender from 1921 to 2011.

Population Pyramid App Cont.

Here is a preview of the dataset.

Show 5 entries Search:

	Gender	Age_Cat	X1901	X1911	X1921	X1922	X1923	X1924
1	Males	1	219987	267411	307300	311900	317800	322800
2	Males	2	231132	229586	302200	305200	305200	301700
3	Males	3	218946	215804	268600	276200	283900	293000
4	Males	4	189434	226831	237800	244500	252600	260900
5	Males	5	174644	228179	220100	225800	230600	235700

Showing 1 to 5 of 36 entries

Previous 1 2 3 4 5 … 8

Next

Population Pyramid App - ui.R

```
# Australian Population Pyramid

# Load required packages
library(shiny)

shinyUI(fluidPage(

  # Application title
  titlePanel("Population Pyramid - Australia 1921 - 2011"),

  # Sidebar with a slider input for year
  sidebarPanel(
    sliderInput("year", label = "Year", min = 1921, sep="",
                max = 2011, value = 1921,
                animate = animationOptions(interval = 500, loop =
  TRUE))
```

Population Pyramid App - server.R

```r
# Australian Population Pyramid

# Load required packages

library(shiny)
library(ggplot2)
library(dplyr)
library(tidyr)
library(stringr)

# Thanks to https://rpubs.com/walkerke/pyramids_ggplot2!
# Import data
ABS <- read.csv("aus_age_gender_hist.csv",stringsAsFactors = TRUE)

# Label Age_cat factor
ABS$Age_Cat<- factor(ABS$Age_Cat,
```

Population Pyramid App - server.R Cont.

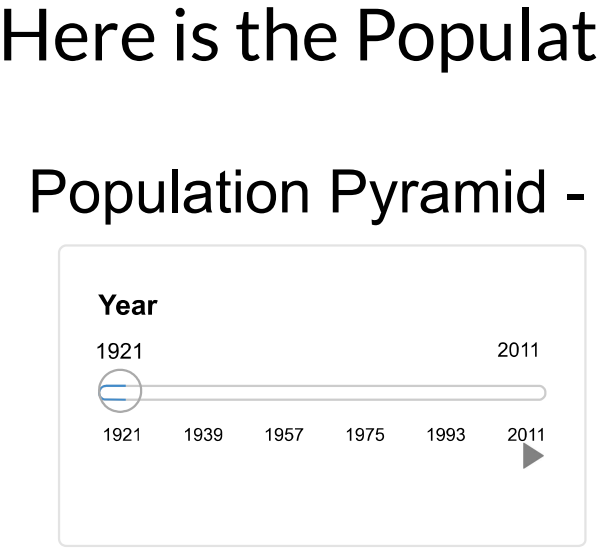
```
# Now we define the shinyServer function

shinyServer(function(input, output) {

  output$pyramid <- renderPlot({
    p1 <- ggplot(ABS_long, aes(x = Age_Cat, y = Population,
                               fill = Gender))
    p1 + geom_bar(data = filter(ABS_long,Gender == "Females"
                                & Year == input$year),
                  stat = "identity") +
      geom_bar(data = filter(ABS_long,Gender == "Males" &
                               Year == input$year),
               aes(y=Population*(-1)), stat = "identity") +
      scale_y_continuous(breaks = seq(-1000000, 1000000, 100000),
                         limits = c(-1000000, 1000000),
                         labels = paste0(as.character(abs(seq(-1,
```

Population Pyramid App Cont. 2

Here is the Population Pyramid app:



Shiny Apps II

- In the following slides, we will look at improving coding efficiency, improving appearance and adding advanced features to Shiny apps. Specifically:
  - Single file apps
  - Building a `ui` Quickly
  - Tabsets and Navbars
  - Plotly and Shiny
  - Reactivity
  - Upload Data and Preview Data tables
  - Shiny Themes
  - Avoiding Common Errors

Single File Apps

- Shiny apps can be created as a single file named `app.R`.
- For small apps, this can make it easier to code and share.
- However, with larger apps, the separation of the `ui.R` and `server.R` files may be more efficient.

```
## Single file Shiny app.R

# Load packages and prepare data

library(ggplot2)
library(shiny)
library(dplyr)
library(lubridate)

texas <- ggplot2::txhousing

texas_sum <- txhousing %>% group_by(year,month) %>%
  summarise(sales = sum(sales,na.rm = TRUE),
            volume = sum(volume,na.rm = TRUE),
            median = sum(median,na.rm = TRUE),
            listings = sum(listings,na.rm = TRUE),
```

Building a `ui` Quickly

- Building a `ui` can be tedious. However, there are shortcuts.
- We can use a `data.frame`'s `colnames` to create a list that can be fed into to the `ui`
- For example, if we wanted a list of factors included in the `MGP` dataset, we could write:

```
MPG <- ggplot2::mpg
names(MPG)[ sapply(MPG, is.character) | sapply(MPG, is.factor) ]
```

```
## [1] "manufacturer" "model"        "trans"        "drv"          "fl"
## [6] "class"
```

- Using this list, we can automatically label widgets in the `ui`.

Building a `ui` Quickly Cont.

```
# Load packages and prepare data

library(ggplot2)
library(shiny)

MPG <- ggplot2::mpg

MPG$year <- MPG$year %>% as.factor()
MPG$cyl <- MPG$cyl %>% as.factor()

# Generate variables lists for ui

factors <- names(MPG)[ sapply(MPG, is.character) |
                       sapply(MPG, is.factor) ]
quantitative <- names(MPG)[ sapply(MPG, is.numeric)]
```

Tabsets and Navbars

- Tabsets and navbars are an effective way to organise visualisations in an app.
- Tabsets share a common layout, but can have multiple panels.

```
# Load packages and prepare data

library(ggplot2)
library(shiny)

MPG <- ggplot2::mpg

MPG$year <- MPG$year %>% as.factor()
MPG$cyl <- MPG$cyl %>% as.factor()
MPG$displ <- MPG$displ %>% as.factor()

# Generate variables lists for ui

factors <- names(MPG)[ sapply(MPG, is.character) |
                       sapply(MPG, is.factor) ]
quantitative <- names(MPG)[ sapply(MPG, is.numeric)]
```

Tabsets and Navbars Cont.

- Navbars can be used alternatively or in addition to tabsets.
- Navbars create independent pages within an app

```
# Load packages and prepare data

library(ggplot2)
library(shiny)

MPG <- ggplot2::mpg

MPG$year <- MPG$year %>% as.factor()
MPG$cyl <- MPG$cyl %>% as.factor()
MPG$displ <- MPG$displ %>% as.factor()

# Generate variable lists for ui

factors <- names(MPG)[ sapply(MPG, is.character) |
                       sapply(MPG, is.factor) ]
quantitative <- names(MPG)[ sapply(MPG, is.numeric)]
```

Plotly and Shiny

- Plotly and Shiny work seamlessly.

```
# Load packages and prepare data

library(ggplot2)
library(shiny)
library(plotly)
library(gapminder)
library(RColorBrewer)

# remove outlier!

gapminder_filt <- gapminder::gapminder %>% filter(country != "Kuwait")

# Assign server function

server5 <- function(input, output) {
  output$gapminder <- renderPlotly({
```

Reactive Events

- Reactive output is a great way to speed up apps as it only execute changes on the server following user input.
- The following example shows the use of an `actionButton`.

```
library(shiny)
library(ecodist)

ui6 <- fluidPage(
  titlePanel("Guess the Correlation"),
   sidebarLayout(
    sidebarPanel(
      actionButton("go", "Generate random correlation"),

      actionButton("answer", "Show answer"),
      verbatimTextOutput("answer")),

    mainPanel(plotOutput("plot"))
)
)
```

Uploading and Viewing Data

- Shiny can also be used to upload user data and preview it.

```
library(shiny)
library(DT)

ui7 <- fluidPage(
  titlePanel("Uploading and Viewing Data"),
  sidebarLayout(
    sidebarPanel(

      fileInput("file", "Choose CSV Dataset",
                multiple = TRUE,
                accept = c("text/csv",
                           "text/comma-separated-values,text/plain",
                           ".csv"))
    ),

    mainPanel(
```

Shiny Themes

- You can change Shiny themes for a different look..
- First install and load `shinythemes`
- Themes include the following: cerulean, cosmo, cyborg, darkly, flatly, journal, lumen, paper, readable, sandstone, simplex, slate, spacelab, superhero, united, yeti

```
install.packages("shinythemes")
library(shinythemes)
```

Shiny Themes Cont.

- We can change the theme by adding a `theme =` option to the `ui`.

```
library(shiny)
library(DT)
library(shinythemes)

ui8 <- fluidPage(theme = shinytheme("cyborg"),
  titlePanel("Uploading, Viewing and Downloading Data"),
  sidebarLayout(
    sidebarPanel(

      fileInput("file", "Choose CSV Dataset",
                multiple = TRUE,
                accept = c("text/csv",
                           "text/comma-separated-values,text/plain",
                           ".csv"))
    ),
```

Avoiding Common Errors in Shiny

- Ensure all your packages and data are loaded in the app.
- Only load the packages required for the app to run.
- Use current versions of packages (GitHub is supported using devtools v. > 1.4).
- Place your dataset in your app directory and load using a relative path.

```
# Wrong
data <- read.csv("C:/folder/subfolder/data.csv")

# Right
data <- read.csv("data.csv")
```

Avoiding Common Errors in Shiny Cont.

- Build your app one feature at a time.
- Check for error clues using `rsconnect::showLogs()`
- Avoid Shiny killing functions such as `install.packages()`, `View()`, `setwd()` etc.
- No spaces or unusual characters in the app name when uploading to Shinyapps.io

References