Hands-on Exercise 3: Programming Interactive Data Visualisation with R

Dr. Kam Tin Seong
Assoc. Professor of Information Systems

School of Computing and Information Systems,
Singapore Management University

2020-2-15 (updated: 2022-04-26)

Interactive Data Visualisation with R

In this hands-on exercise, you will learn how to create:

- interactive data visualisation by using ggiraph and plotlyr packages,
- animated data visualisation by using gganimate and plotlyr packages.

Getting Started

First, write a code chunk to check, install and launch the following R packages:

- **ggiraph** for making 'ggplot' graphics interactive.
- **plotly**, R library for plotting interactive statistical graphs.
- **gganimate**, an ggplot extension for creating animated statistical graphs.
- **patchwork**, an ggplot extension for combining multiple ggplot objects into a single figure.
- **DT** provides an R interface to the JavaScript library DataTables that create interactive table on html page.
- **tidyverse**, a family of modern R packages specially designed to support data science, analysis and communication task including creating static statistical graphs.

The solution:

```
packages = c('ggiraph', 'plotly',
             'DT', 'patchwork',
             'gganimate', 'tidyverse',
             'readxl', 'gifski', 'gapminder')
for (p in packages){
  if(!require(p, character.only = T)){
    install.packages(p)
  }
  library(p,character.only = T)
}
```

Importing Data

In this section, Exam_data.csv provided will be used. Using *read_csv()* of **readr** package, import *Exam_data.csv* into R.

The solution:

```
exam_data <- read_csv("data/Exam_data.csv")
```

Interactive Data Visualisation - ggiraph methods

- An htmlwidget and a ggplot2 extension. It allows ggplot graphics to be interactive.
- Interactive is made with ggplot geometries that can understand three arguments:
  - **Tooltip**: a column of data-sets that contain tooltips to be displayed when the mouse is over elements.
  - **Onclick**: a column of data-sets that contain a JavaScript function to be executed when elements are clicked.
  - **Data_id**: a column of data-sets that contain an id to be associated with elements.
- If it used within a shiny application, elements associated with an id (data_id) can be selected and manipulated on client and server sides.

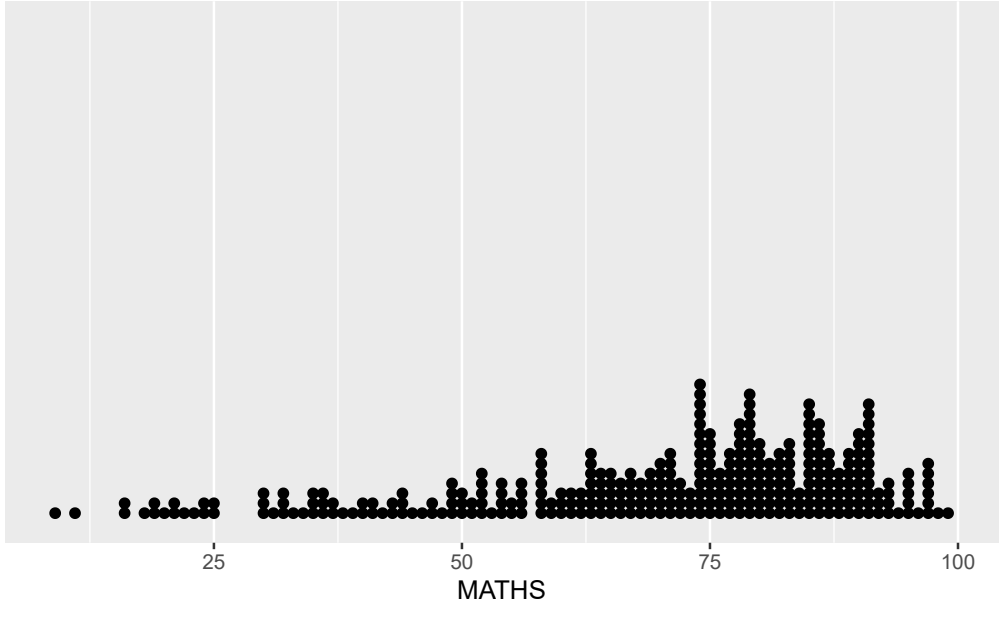
Reference: ggiraph package

Tooltip effect with *tooltip* aesthetic

Below shows a typical code chunk to plot an interactive statistical graph by using **ggiraph** package. Notice that the code chunk consists of two parts. First, an ggplot object will be created. Next, `girafe()` of **ggiraph** will be used to create an interactive svg object.

```
p <- ggplot(data=exam_data,
       aes(x = MATHS)) +
  geom_dotplot_interactive(
    aes(tooltip = ID),
    stackgroups = TRUE,
    binwidth = 1,
    method = "histodot") +
  scale_y_continuous(NULL,
                     breaks = NULL)
girafe(
  ggobj = p,
  width_svg = 6,
  height_svg = 6*0.618
)
```

Interactivity: By hovering the mouse pointer on an data point of interest, the student's ID will be displayed.



Comparing ggplot2 and ggiraph codes

The original ggplot2 code chunk.

```
ggplot(data=exam_data,
       aes(x = MATHS)) +
  geom_dotplot(binwidth=2.5,
               dotsize = 0.5) +
  scale_y_continuous(NULL,
                     breaks = NULL)
```

The ggiraph code chunk.

```
p <- ggplot(data=exam_data,
       aes(x = MATHS)) +
  geom_dotplot_interactive(
    aes(tooltip = ID),
    stackgroups = TRUE,
    binwidth = 1,
    method = "histodot") +
  scale_y_continuous(NULL,
                     breaks = NULL)
girafe(
  ggobj = p,
  width_svg = 6,
  height_svg = 6*0.618
)
```

A complete list of geometries supported by ggiraph and their corresponding command syntax can be found here.

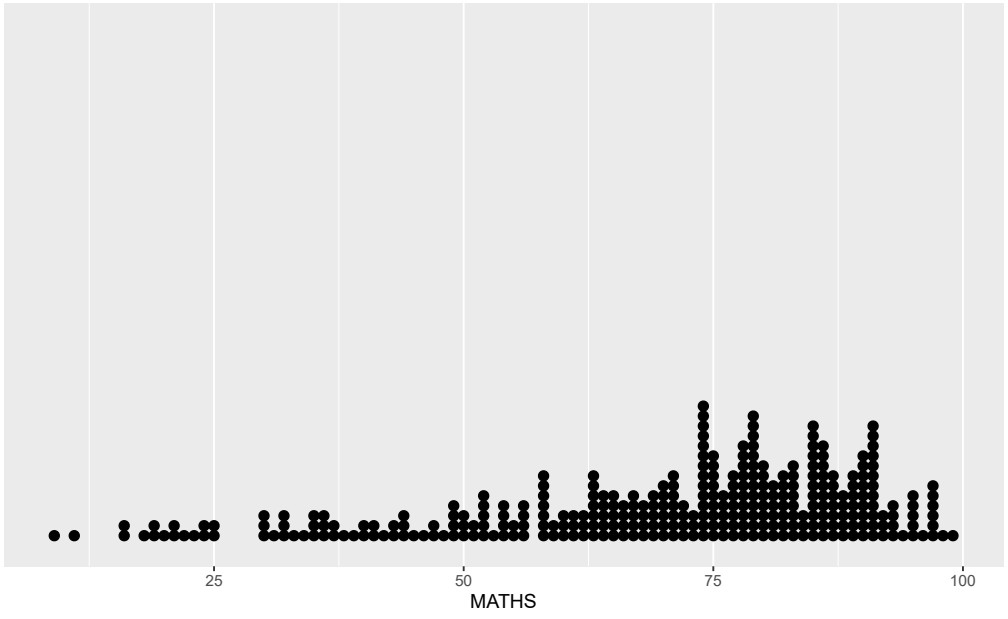
Displaying multiple information on tooltip

The content of the tooltip can be customised by including a list object as shown in the code chunk below.

```
exam_data$tooltip <- c(paste0(
  "Name = ", exam_data$ID,
  "\n Class = ", exam_data$CLASS))

p <- ggplot(data=exam_data,
       aes(x = MATHS)) +
  geom_dotplot_interactive(
    aes(tooltip = exam_data$tooltip),
    stackgroups = TRUE,
    binwidth = 1,
    method = "histodot") +
  scale_y_continuous(NULL,
                     breaks = NULL)
girafe(
  ggobj = p,
  width_svg = 8,
  height_svg = 8*0.618
)
```

Interactivity: By hovering the mouse pointer on an data point of interest, the student's ID and Class will be displayed.



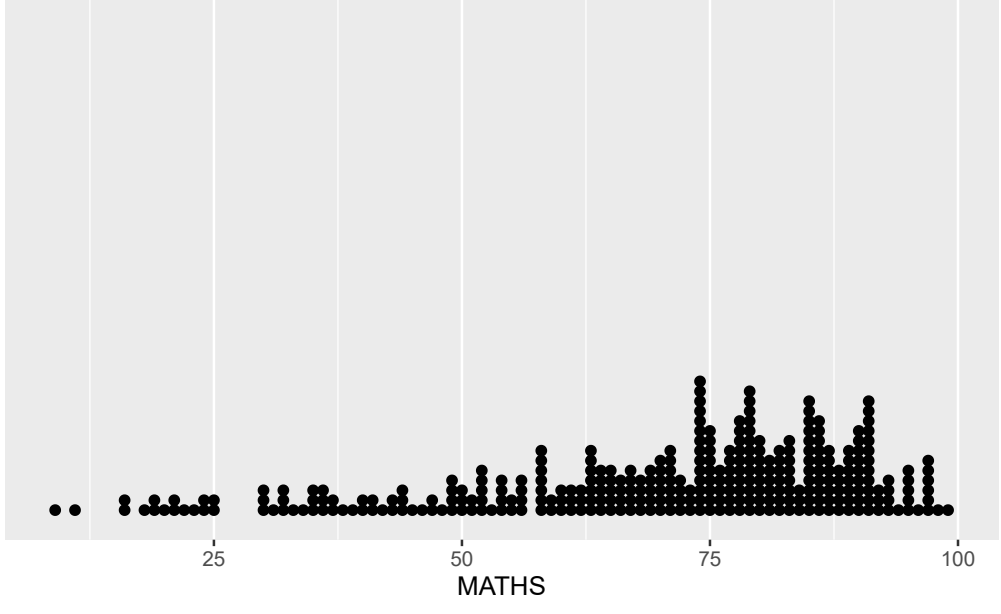
Customising Tooltip style

Code chunk below uses `opts_tooltip()` of **ggiraph** to customize tooltip rendering by add css declarations.

```
tooltip_css <- "background-color:white;
font-style:bold; color:black;"

p <- ggplot(data=exam_data,
       aes(x = MATHS)) +
  geom_dotplot_interactive(
    aes(tooltip = ID),
    stackgroups = TRUE,
    binwidth = 1,
    method = "histodot") +
  scale_y_continuous(NULL,
                     breaks = NULL)
girafe(
  ggobj = p,
  width_svg = 6,
  height_svg = 6*0.618,
  options = list(
    opts_tooltip(
      css = tooltip_css))
)
```

Notice that the background colour of the tooltip is black and the font colour is white and bold.



- Refer to Customizing girafe objects to learn more about how to customise ggiraph objects.

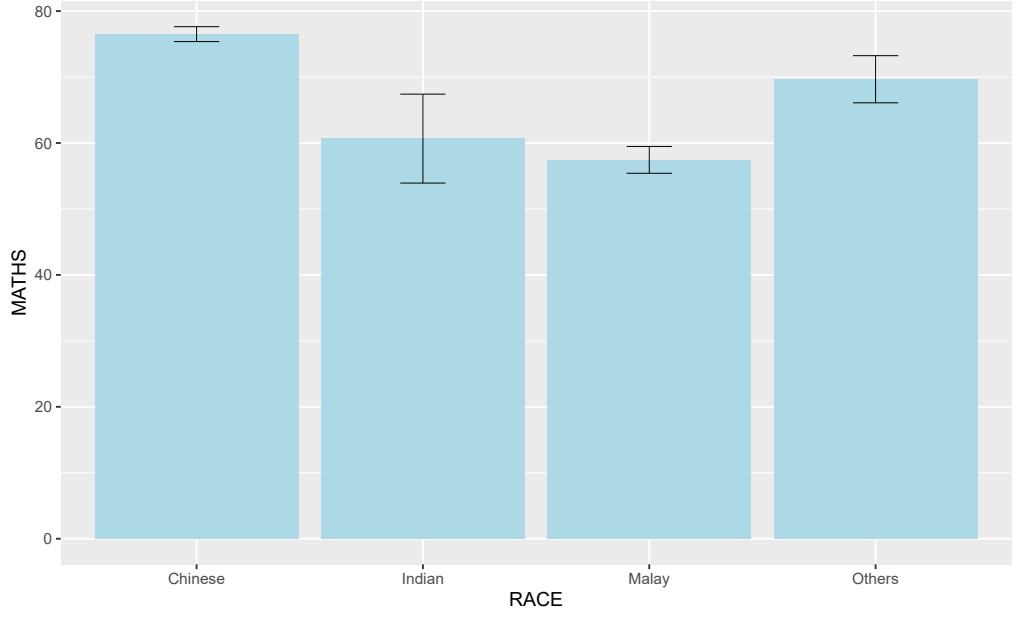
Displaying statistics on tooltip

```
tooltip <- function(y, ymax, accuracy = .01) {
  mean <- scales::number(y, accuracy = accuracy
  sem <- scales::number(ymax - y, accuracy = ac
  paste("Mean maths scores:", mean, "+/-", sem)
}

gg_point <- ggplot(data=exam_data,
                   aes(x = RACE),
) +
  stat_summary(aes(y = MATHS,
                   tooltip = after_stat(
                     tooltip(y, ymax))),
    fun.data = "mean_se",
    geom = GeomInteractiveCol,
    fill = "light blue"
  ) +
  stat_summary(aes(y = MATHS),
    fun.data = mean_se,
    geom = "errorbar", width = 0.2, size = 0.2
  )

girafe(ggobj = gg_point,
       width_svg = 8,
       height_svg = 8*0.618)
```

Code chunk on the left shows an advanced way to customise tooltip. In this example, a function is used to compute 90% confident interval of the mean. The derived statistics are then displayed in the tooltip.

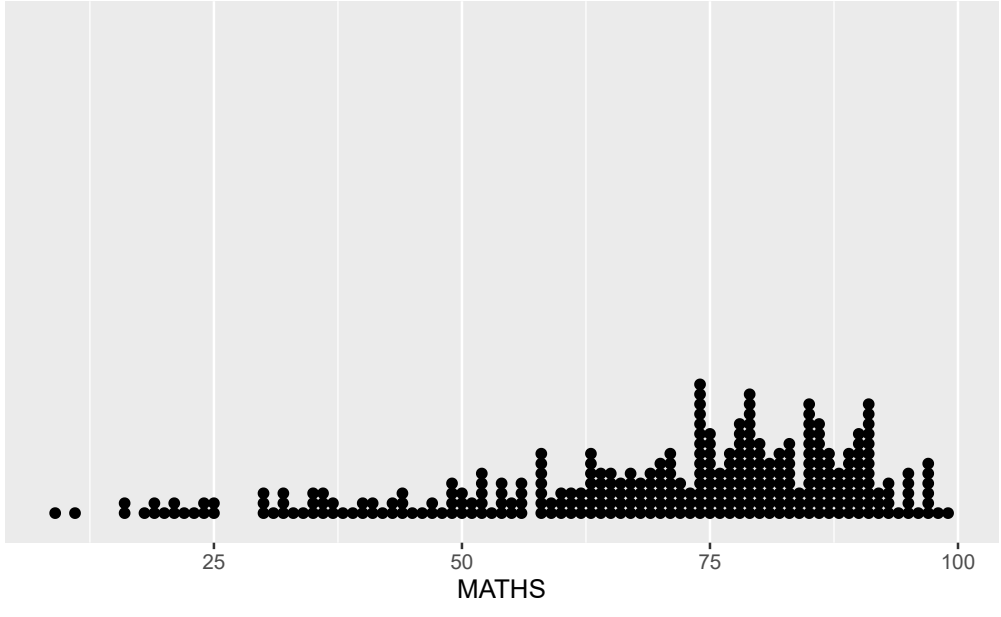


Hover effect with *data_id* aesthetic

Code chunk below show the second interactive feature of ggiraph, namely `data_id`.

```
p <- ggplot(data=exam_data,
       aes(x = MATHS)) +
  geom_dotplot_interactive(
    aes(data_id = CLASS),
    stackgroups = TRUE,
    binwidth = 1,
    method = "histodot") +
  scale_y_continuous(NULL,
                     breaks = NULL)
girafe(
  ggobj = p,
  width_svg = 6,
  height_svg = 6*0.618
)
```

Interactivity: Elements associated with a *data_id* (i.e CLASS) will be highlighted upon mouse over.



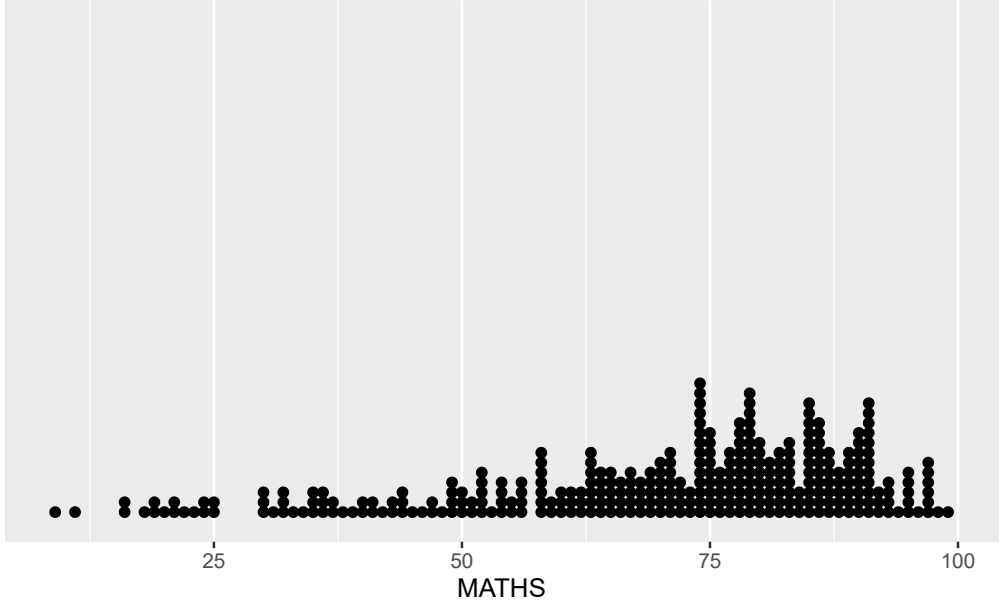
Note that the default value of the hover css is *hover_css = "fill:orange;"*.

Styling hover effect

In the code chunk below, css codes are used to change the highlighting effect.

```
p <- ggplot(data=exam_data,
       aes(x = MATHS)) +
  geom_dotplot_interactive(
    aes(data_id = CLASS),
    stackgroups = TRUE,
    binwidth = 1,
    method = "histodot") +
  scale_y_continuous(NULL,
                     breaks = NULL)
girafe(
  ggobj = p,
  width_svg = 6,
  height_svg = 6*0.618,
  options = list(
    opts_hover(css = "fill: #202020;"),
    opts_hover_inv(css = "opacity:0.2;")
  )
)
```

Interactivity: Elements associated with a *data_id* (i.e CLASS) will be highlighted upon mouse over.



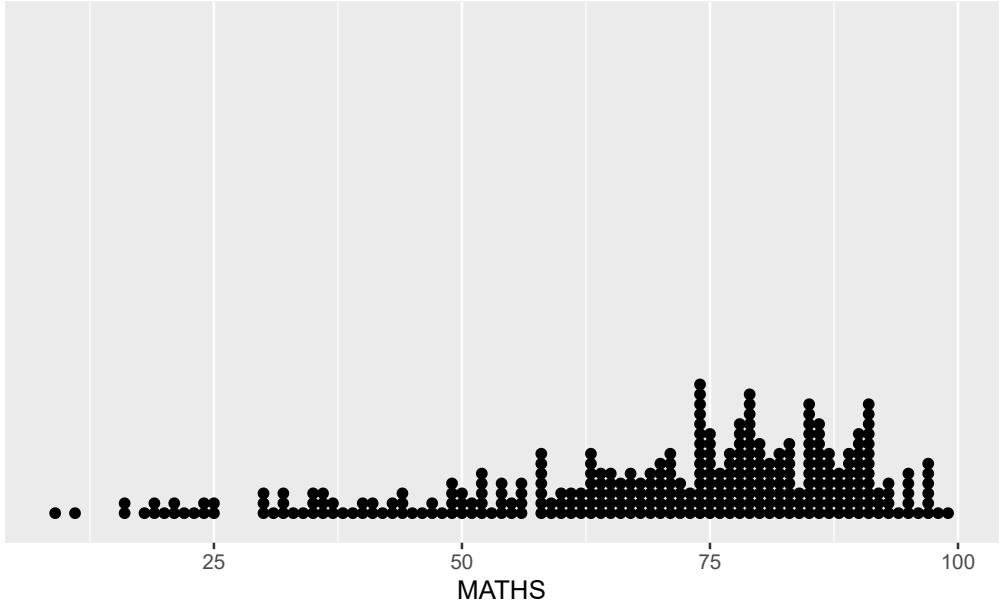
Note: Different from Slide 9, in this example the ccs customisation request are encoded directly.

Combining tooltip and hover effect

There are time that we want to combine tooltip and hover effect on the interactive statistical graph as shown in the code chunk below.

```
p <- ggplot(data=exam_data,
       aes(x = MATHS)) +
  geom_dotplot_interactive(
    aes(tooltip = CLASS,
        data_id = CLASS),
    stackgroups = TRUE,
    binwidth = 1,
    method = "histodot") +
  scale_y_continuous(NULL,
                     breaks = NULL)
girafe(
  ggobj = p,
  width_svg = 6,
  height_svg = 6*0.618,
  options = list(
    opts_hover(css = "fill: #202020;"),
    opts_hover_inv(css = "opacity:0.2;")
  )
)
```

Interactivity: Elements associated with a *data_id* (i.e CLASS) will be highlighted upon mouse over. At the same time, the tooltip will show the CLASS.



Click effect with onclick

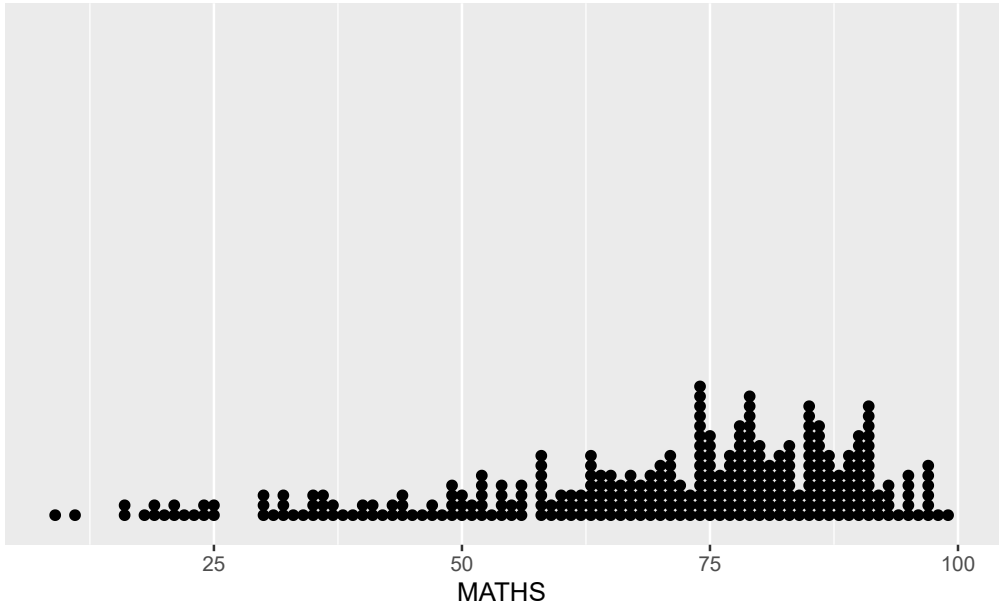
`onclick` argument of ggiraph provides hotlink interactivity on ther web.

The code chunk below shown an example of `onclick`.

```
exam_data$onclick <- sprintf("window.open(\"%s
"https://www.moe.gov.sg/schoolfinder?journey=P

p <- ggplot(data=exam_data,
       aes(x = MATHS)) +
  geom_dotplot_interactive(
    aes(onclick = onclick),
    stackgroups = TRUE,
    binwidth = 1,
    method = "histodot") +
  scale_y_continuous(NULL,
                     breaks = NULL)
girafe(
  ggobj = p,
  width_svg = 6,
  height_svg = 6*0.618)
```

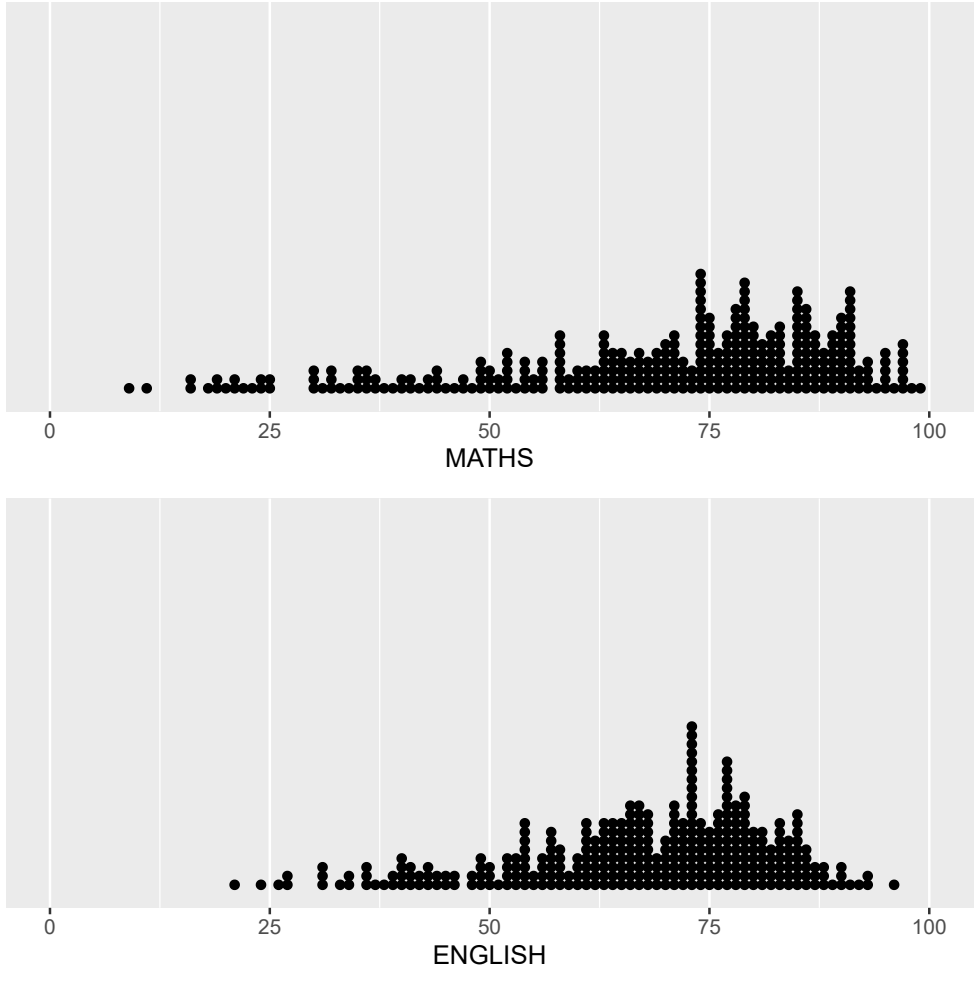
Interactivity: Web document link with a data object will be displayed on the web browser upon mouse click.



Coordinated Multiple Views with ggiraph

Coordinated multiple views methods has been implemented in the data visualisation on the right.

- when a data point of one of the dotplot is selected, the corresponding data point ID on the second data visualisation will be highlighted too.



Coordinated Multiple Views with ggiraph

In order to build a coordinated multiple views, the following programming strategy will be used:

1. Appropriate interactive functions of **ggiraph** will be used to create the multiple views.
2. *patchwork* function of patchwork package will be used inside girafe function to create the interactive coordinated multiple views.

```
p1 <- ggplot(data=exam_data,
       aes(x = MATHS)) +
  geom_dotplot_interactive(
    aes(data_id = ID),
    stackgroups = TRUE,
    binwidth = 1,
    method = "histodot") +
  coord_cartesian(xlim=c(0,100)) +
  scale_y_continuous(NULL,
                     breaks = NULL)
```

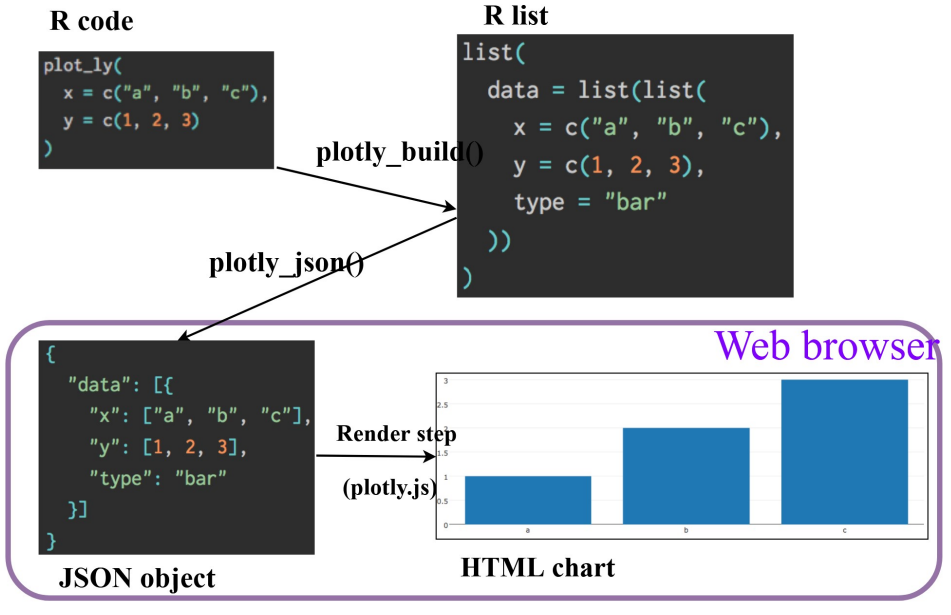
```
p2 <- ggplot(data=exam_data,
       aes(x = ENGLISH)) +
  geom_dotplot_interactive(
    aes(data_id = ID),
    stackgroups = TRUE,
    binwidth = 1,
    method = "histodot") +
  coord_cartesian(xlim=c(0,100)) +
  scale_y_continuous(NULL,
                     breaks = NULL)

girafe(code = print(p1 / p2),
       width_svg = 6,
       height_svg = 6,
       options = list(
         opts_hover(css = "fill: #202020;"),
         opts_hover_inv(css = "opacity:0.2;")
         )
       )
```

The data_id aesthetic is critical to link observations between plots and the tooltip aesthetic is optional but nice to have when mouse over a point.

Interactive Data Visualisation - plotly methods!

- Plotly's R graphing library create interactive web graphics from **ggplot2** graphs and/or a custom interface to the (MIT-licensed) JavaScript library **plotly.js** inspired by the grammar of graphics.
- Different from other plotly platform, plot.R is free and open source.



There are two ways to create interactive graph by using plotly, they are:

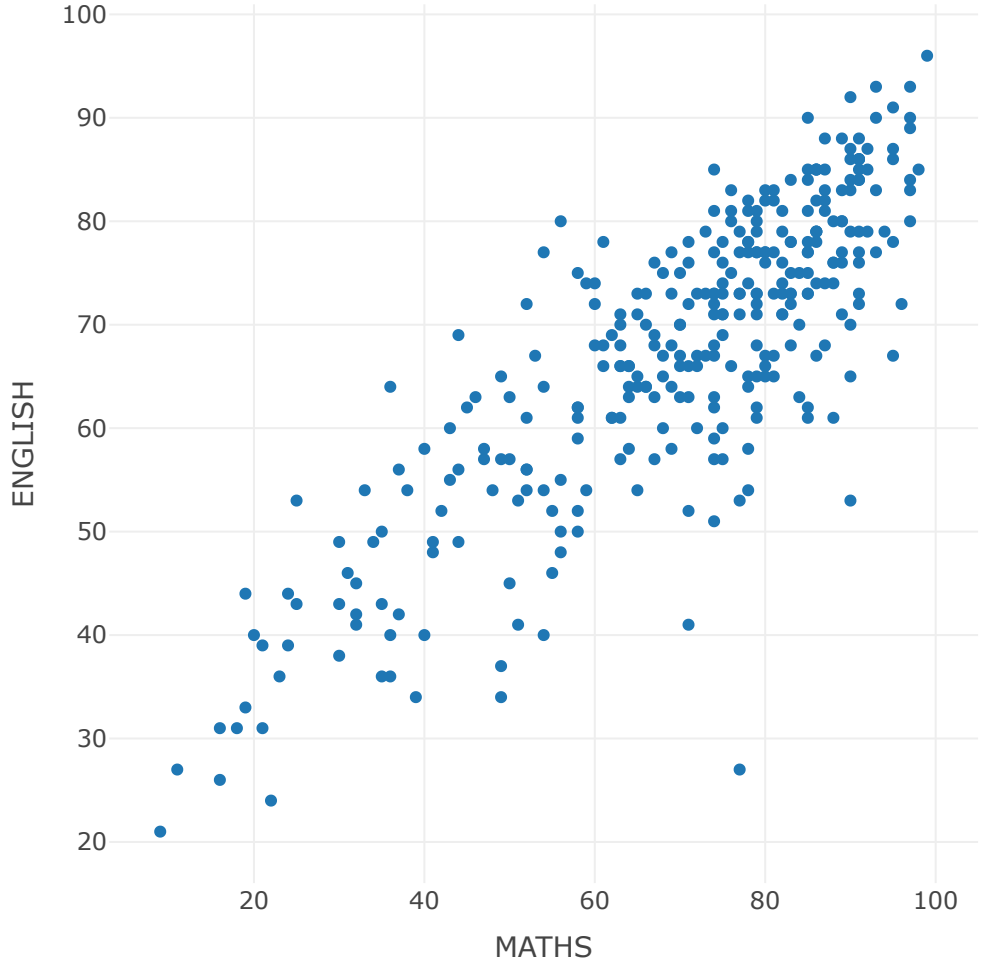
- by using *plot_ly()*, and
- by using *ggplotly()*

Creating an interactive scatter plot: plot_ly() method

The code chunk below plots an interactive scatter plot by using *plot_ly()*.

```
plot_ly(data = exam_data,
             x = ~MATHS,
             y = ~ENGLISH)
```

The output:



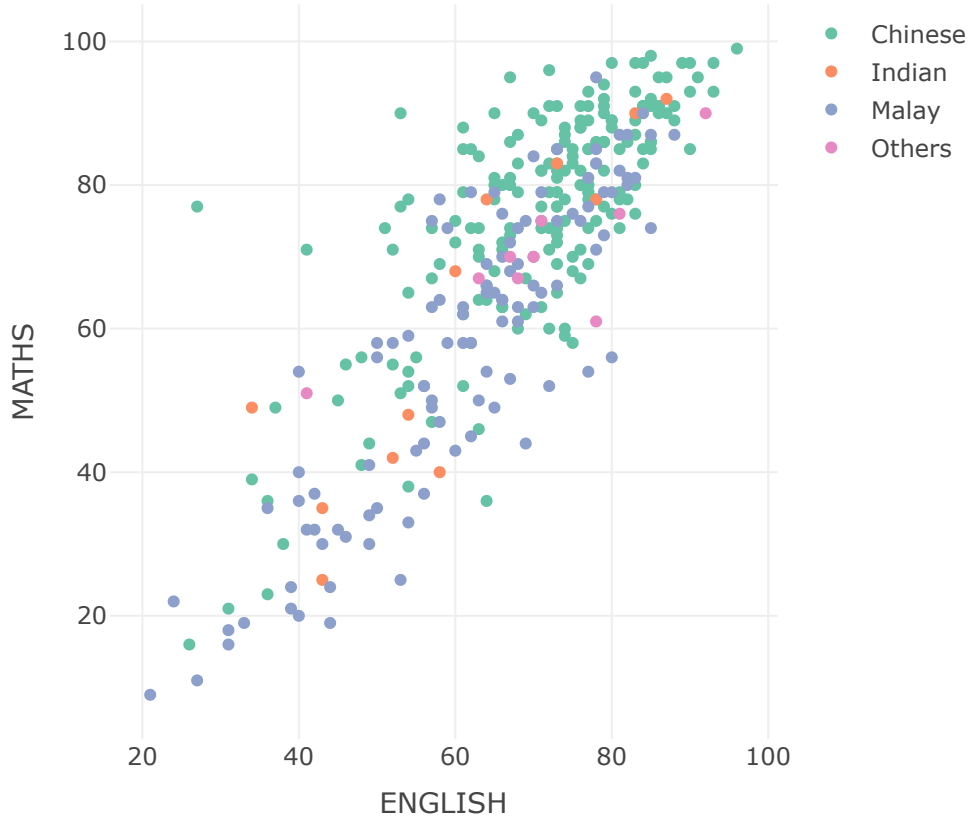
Working with visual variable: plot_ly() method

In the code chunk below, *color* argument is mapped to a qualitative visual variable (i.e. RACE).

```
plot_ly(data = exam_data,
        x = ~ENGLISH,
        y = ~MATHS,
        color = ~RACE)
```

Interactive:

- Click on the colour symbol at the legend.



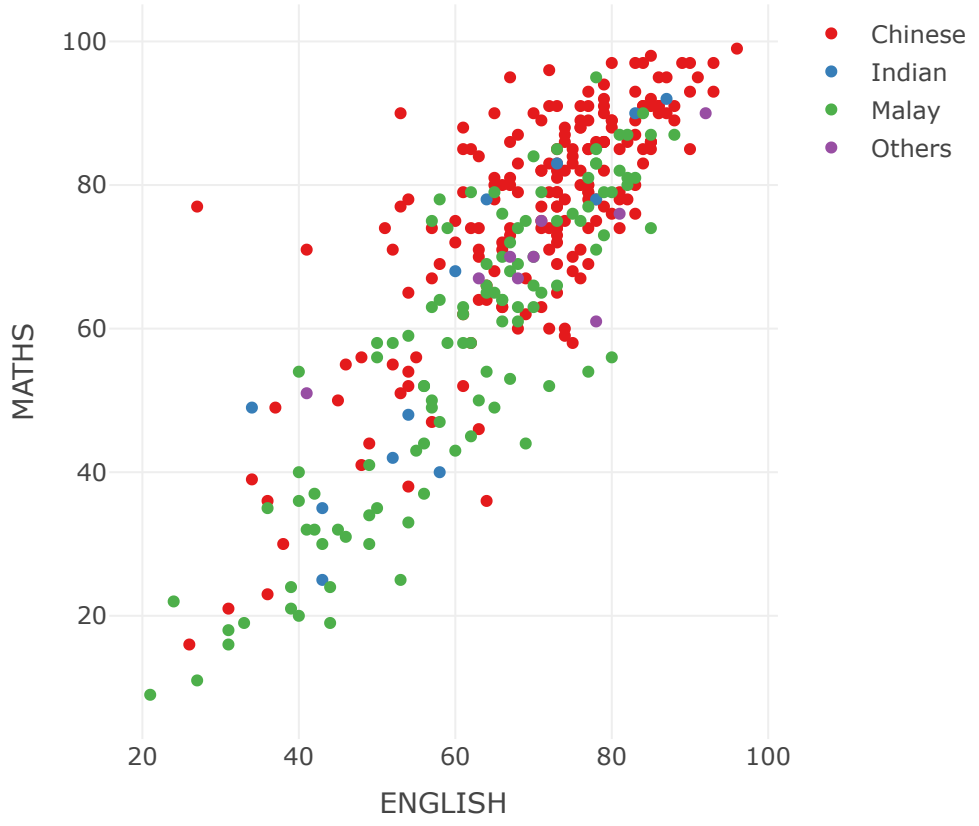
Changing colour pallete: plot_ly() method

In the code chunk below, *colors* argument is used to change the default colour palette to ColorBrewel colour palette.

```
plot_ly(data = exam_data,
        x = ~ENGLISH,
        y = ~MATHS,
        color = ~RACE,
        colors = "Set1")
```

Interactive:

- Click on the colour symbol at the legend.



Customising colour scheme: plot_ly() method

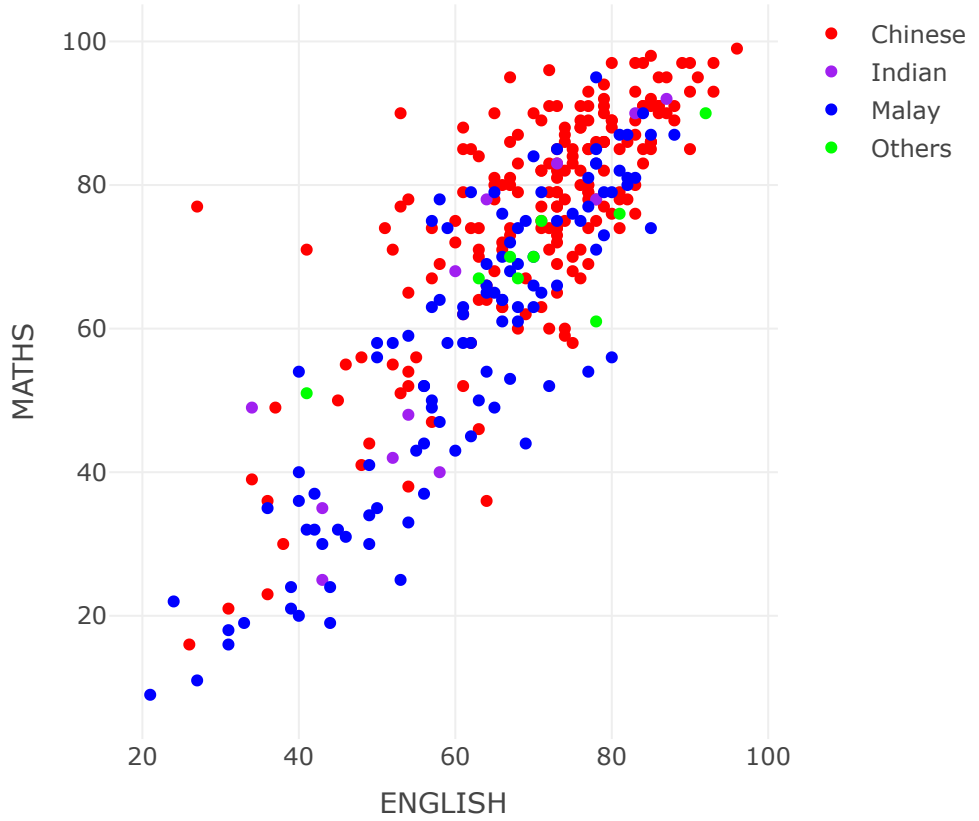
In the code chunk below, a customised colour scheme is created. Then, *colors* argument is used to change the default colour palette to the customised colour scheme.

```
pal <- c("red", "purple", "blue", "green")

plot_ly(data = exam_data,
        x = ~ENGLISH,
        y = ~MATHS,
        color = ~RACE,
        colors = pal)
```

Interactive:

- Click on the colour symbol at the legend.



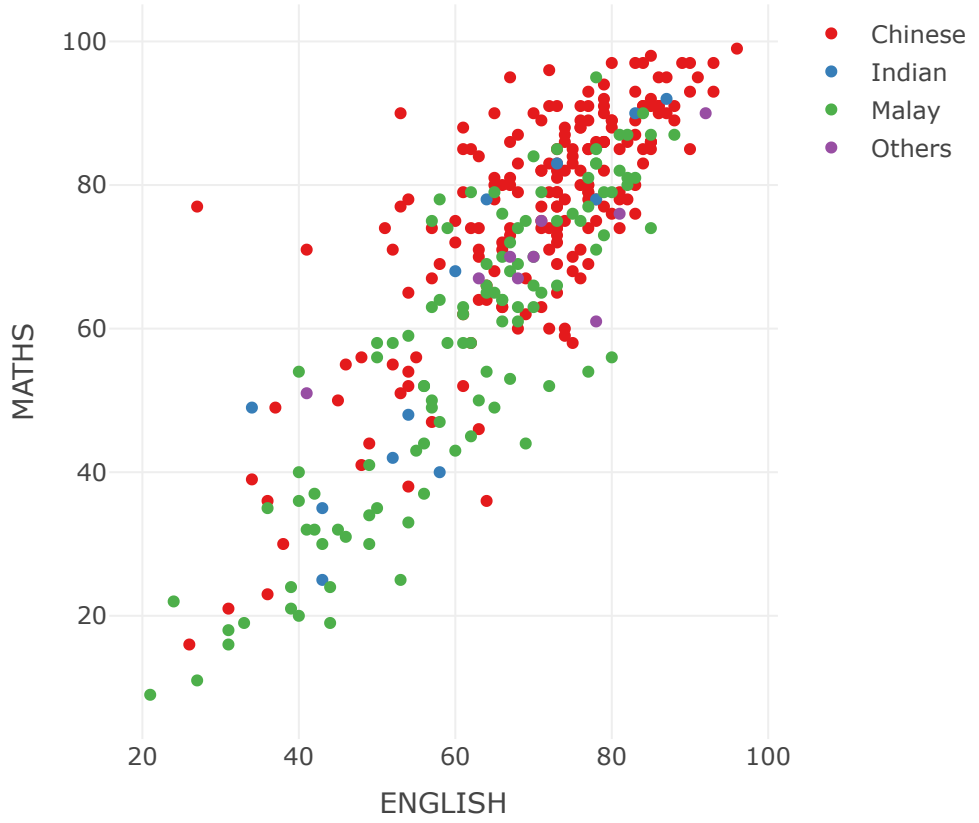
Customising tooltip: plot_ly() method

In the code chunk below, *text* argument is used to change the default tooltip.

```
plot_ly(data = exam_data,
        x = ~ENGLISH,
        y = ~MATHS,
        text = ~paste("Student ID:", ID,
                      "<br>Class:", CLASS),
        color = ~RACE,
        colors = "Set1")
```

Interactive:

- Click on the colour symbol at the legend.



Working with layout: plot_ly() method

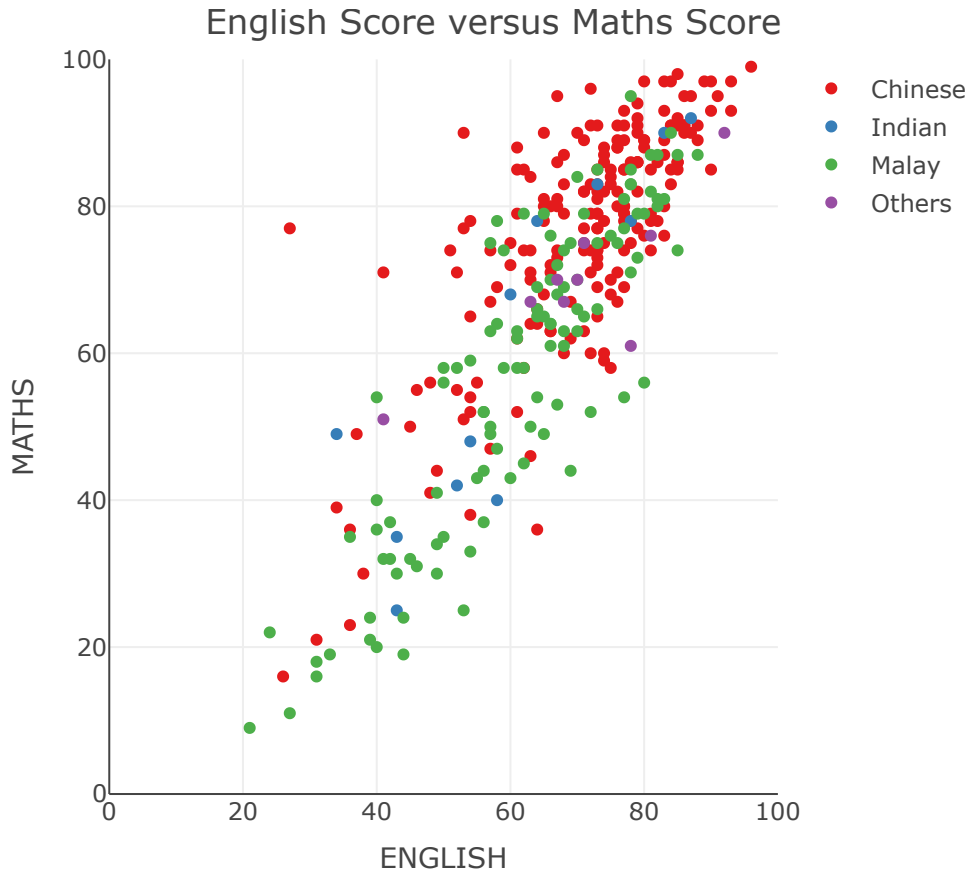
In the code chunk below, *layout* argument is used to change the default tooltip.

```
plot_ly(data = exam_data,
        x = ~ENGLISH,
        y = ~MATHS,
        text = ~paste("Student ID:", ID,
                      "<br>Class:", CLASS),
        color = ~RACE,
        colors = "Set1") %>%
  layout(title = 'English Score versus Maths Sc
         xaxis = list(range = c(0, 100)),
         yaxis = list(range = c(0, 100)))
```

To learn more about layout, visit this link.

Interactive:

- Click on the colour symbol at the legend.

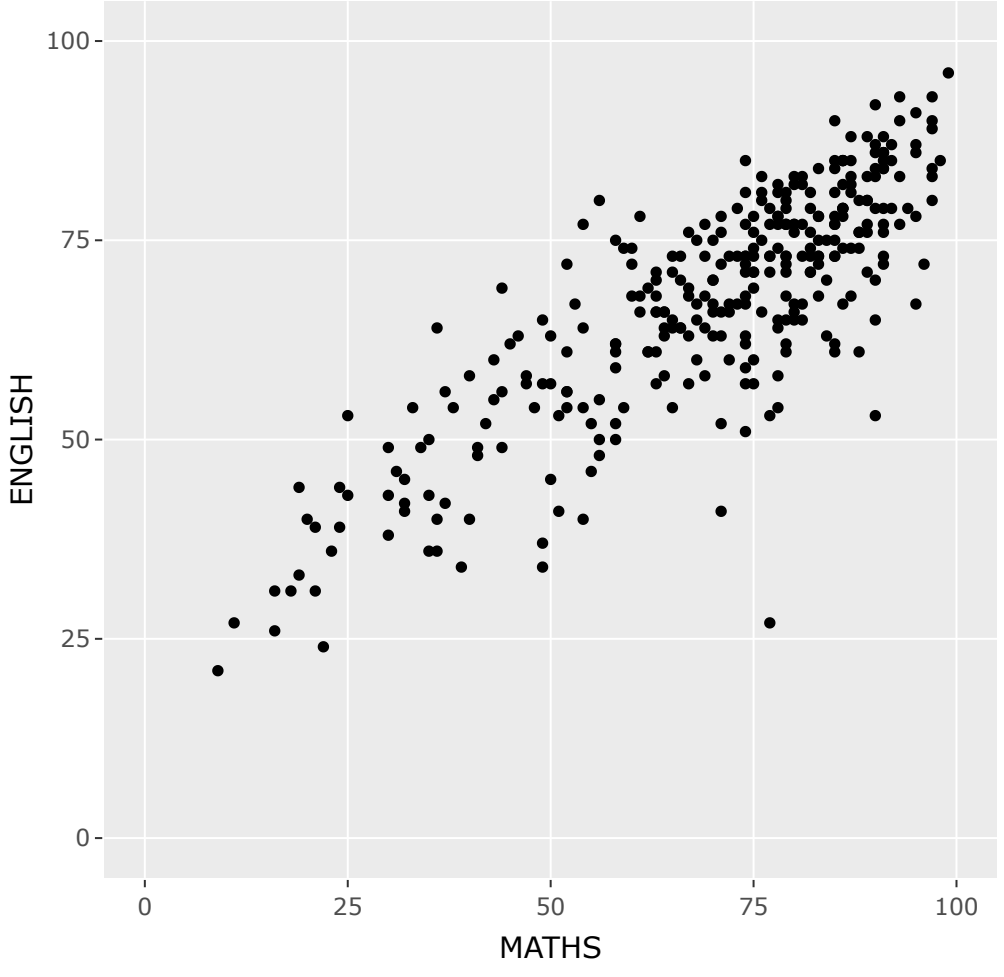


Creating an interactive scatter plot: ggplotly() method

The code chunk below plots an interactive scatter plot by using *ggplotly()*.

```
p <- ggplot(data=exam_data,
            aes(x = MATHS,
                y = ENGLISH)) +
  geom_point(dotsize = 1) +
  coord_cartesian(xlim=c(0,100),
                  ylim=c(0,100))
ggplotly(p)
```

Notice that the only extra line you need to include in the code chunk is *ggplotly()*.



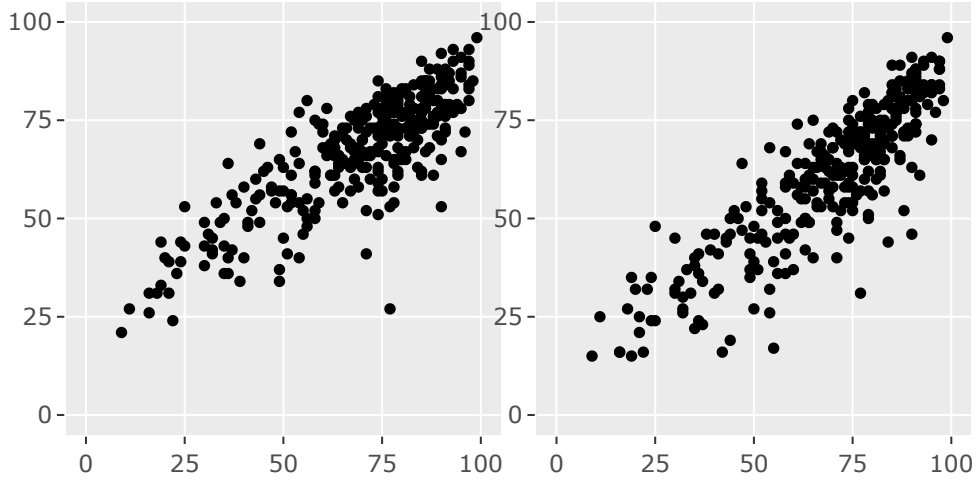
Coordinated Multiple Views with plotly

Code chunk below plots two scatterplots and places them next to each other side-by-side by using *subplot()* of **plotly** package.

```
p1 <- ggplot(data=exam_data,
               aes(x = MATHS,
                   y = ENGLISH)) +
  geom_point(size=1) +
  coord_cartesian(xlim=c(0,100),
                  ylim=c(0,100))

p2 <- ggplot(data=exam_data,
             aes(x = MATHS,
                 y = SCIENCE)) +
  geom_point(size=1) +
  coord_cartesian(xlim=c(0,100),
                  ylim=c(0,100))
subplot(ggplotly(p1),
        ggplotly(p2))
```

The side-by-side scatterplots.



Notice that these two scatter plots are not linked.

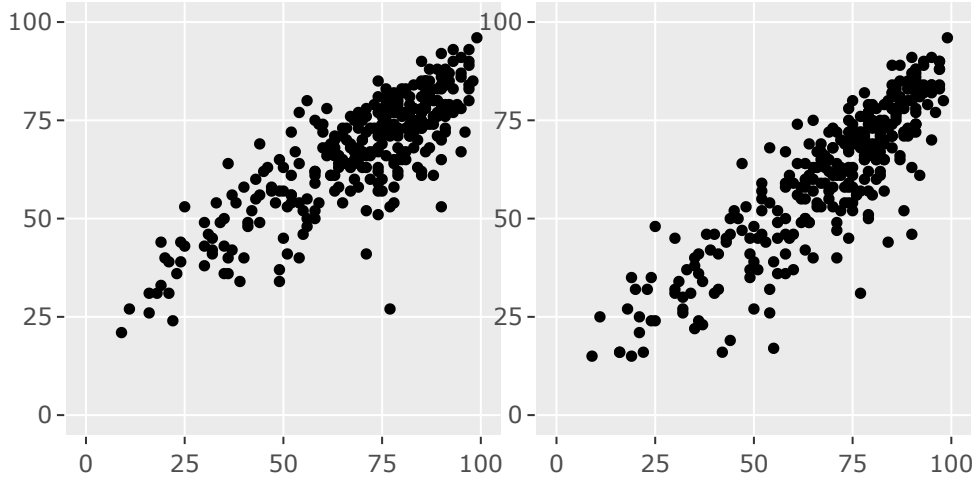
Coordinated Multiple Views with plotly

To create a coordinated scatterplots, *highlight_key()* of **plotly** package is used.

```
d <- highlight_key(exam_data)
p1 <- ggplot(data=d,
             aes(x = MATHS,
                 y = ENGLISH)) +
  geom_point(size=1) +
  coord_cartesian(xlim=c(0,100),
                  ylim=c(0,100))

p2 <- ggplot(data=d,
             aes(x = MATHS,
                 y = SCIENCE)) +
  geom_point(size=1) +
  coord_cartesian(xlim=c(0,100),
                  ylim=c(0,100))
subplot(ggplotly(p1),
        ggplotly(p2))
```

Click on a data point of one of the scatterplot and see how the corresponding point on the other scatterplot is selected.



Thing to learn from the code chunk:

- *highlight_key()* simply creates an object of class crosstalk::SharedData.
- Visit this link to learn more about crosstalk,

Interactive Data Table: DT package

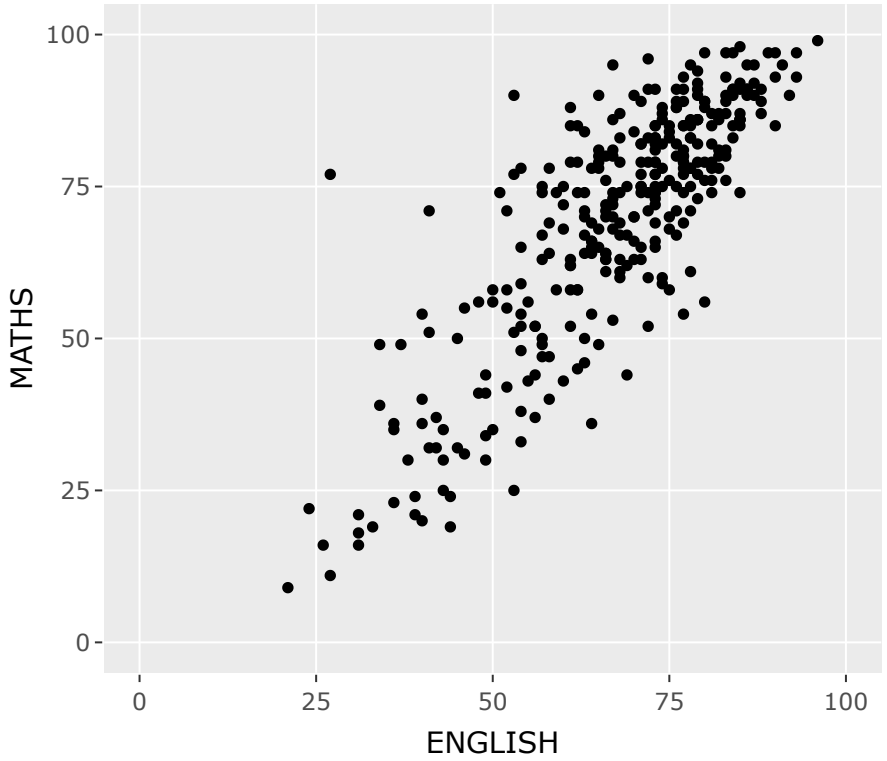
- A wrapper of the JavaScript Library DataTables
- Data objects in R can be rendered as HTML tables using the JavaScript library 'DataTables' (typically via R Markdown or Shiny).

```
DT::datatable(exam_data)
```

Show 10 entries Search:

	ID	CLASS	GENDER	RACE	ENGLISH	MATHS	SCIENCE	tooltip	
1	Student321	3I	Male	Malay	21	9	15	Name = Student321 Class = 3I	window.open("http journey=Primary%2
2	Student305	3I	Female	Malay	24	22	16	Name = Student305 Class = 3I	window.open("http journey=Primary%2
3	Student289	3H	Male	Chinese	26	16	16	Name = Student289 Class = 3H	window.open("http journey=Primary%2

Linked brushing: crosstalk method



Show 10 entries

	ID	CLASS	GENDER	RACE	ENGLISH	MATHS
1	Student321	3I	Male	Malay	21	
2	Student305	3I	Female	Malay	24	2
3	Student289	3H	Male	Chinese	26	1
4	Student227	3F	Male	Chinese	27	7
5	Student318	3I	Male	Malay	27	1

Linked brushing: crosstalk method

Code chunk below is used to implement the coordinated brushing shown on Slide 24.

```
d <- highlight_key(exam_data)
p <- ggplot(d,
            aes(ENGLISH,
                MATHS)) +
  geom_point(size=1) +
  coord_cartesian(xlim=c(0,100),
                  ylim=c(0,100))

gg <- highlight(ggplotly(p),
                "plotly_selected")

crosstalk::bscols(gg,
                  DT::datatable(d),
                  widths = 5)
```

Things to learn from the code chunk:

- *highlight()* is a function of **plotly** package. It sets a variety of options for brushing (i.e., highlighting) multiple plots. These options are primarily designed for linking multiple plotly graphs, and may not behave as expected when linking plotly to another htmlwidget package via crosstalk. In some cases, other htmlwidgets will respect these options, such as persistent selection in leaflet.

- *bscols()* is a helper function of **crosstalk** package. It makes it easy to put HTML elements side by side. It can be called directly from the console but is especially designed to work in an R Markdown document. **Warning:** This will bring in all of Bootstrap!.

Animated Data Visualisation: gganimate methods

gganimate extends the grammar of graphics as implemented by ggplot2 to include the description of animation. It does this by providing a range of new grammar classes that can be added to the plot object in order to customise how it should change with time.

- `transition_*()` defines how the data should be spread out and how it relates to itself across time.
- `view_*()` defines how the positional scales should change along the animation.
- `shadow_*()` defines how data from other points in time should be presented in the given point in time.
- `enter_*()/exit_*()` defines how new data should appear and how old data should disappear during the course of the animation.
- `ease_aes()` defines how different aesthetics should be eased during transitions.

Getting started

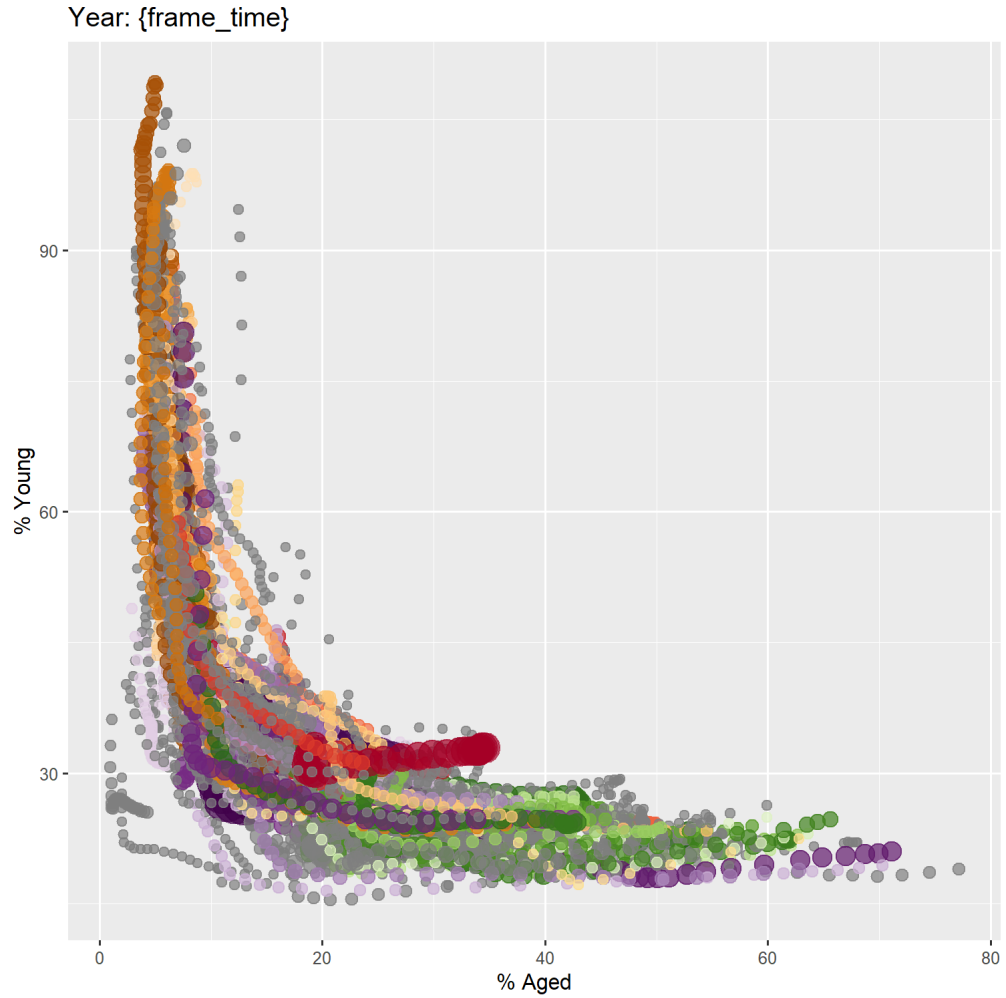
Add the following packages in the packages list:

- **gganimate**: An ggplot extension for creating animated statistical graphs.
- **gifski** converts video frames to GIF animations using pngquant's fancy features for efficient cross-frame palettes and temporal dithering. It produces animated GIFs that use thousands of colors per frame.
- **gapminder**: An excerpt of the data available at Gapminder.org. We just want to use its *country_colors* scheme.

- Import the *Data* worksheet from *GlobalPopulation* Excel workbook.

Building a static population bubble plot

```
ggplot(globalPop, aes(x = Old, y = Young,
                      size = Population,
                      colour = Country)) +
  geom_point(alpha = 0.7,
             show.legend = FALSE) +
  scale_colour_manual(values = country_colors)
  scale_size(range = c(2, 12)) +
  labs(title = 'Year: {frame_time}',
       x = '% Aged',
       y = '% Young')
```

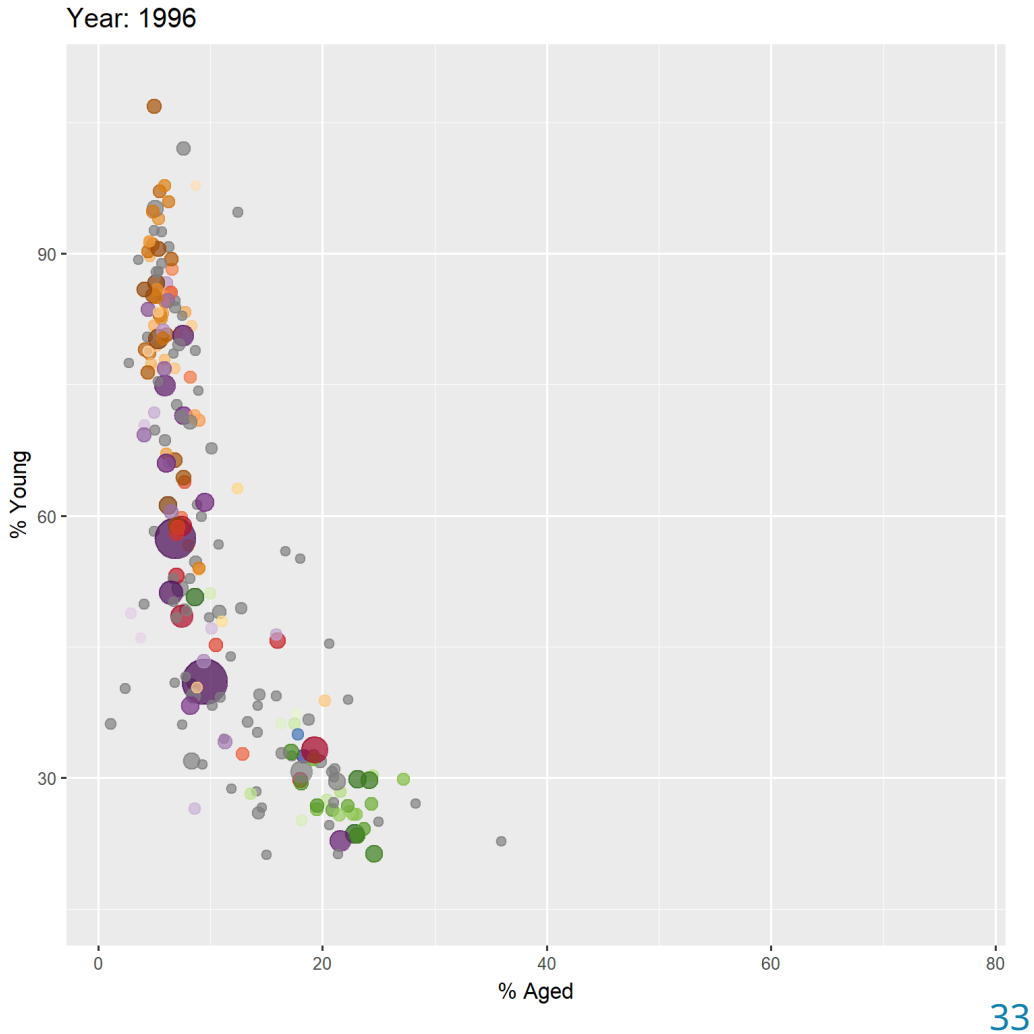


Building the animated bubble plot

The code chunk:

```
ggplot(globalPop, aes(x = Old, y = Young,
                      size = Population,
                      colour = Country)) +
  geom_point(alpha = 0.7,
             show.legend = FALSE) +
  scale_colour_manual(values = country_colors)
  scale_size(range = c(2, 12)) +
  labs(title = 'Year: {frame_time}',
       x = '% Aged',
       y = '% Young') +
  transition_time(Year) +
  ease_aes('linear')
```

The animated bubble chart



Reference

ggiraph

This link provides online version of the reference guide and several useful articles. Use this link to download the pdf version of the reference guide.

- How to Plot With Ggiraph
- Interactive map of France with ggiraph
- Custom interactive sunbursts with ggplot in R
- This link provides code example on how ggiraph is used to interactive graphs for Swiss Olympians - the solo specialists.

plotly for R

- Getting Started with Plotly in R
- - A collection of plotly R graphs are available via this link.
- Carson Sievert (2020) **Interactive web-based data visualization with R, plotly, and shiny**, Chapman and Hall/CRC is the best resource to learn plotly for R. The online version is available via this link
- Plotly R Figure Reference provides a comprehensive discussion of each visual representations.
- Plotly R Library Fundamentals is a good place to learn the fundamental features of Plotly's R API.

Reference

gganimate

- Getting Started
- Visit this link for a very interesting implementation of gganimate by your senior.