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: 2022-05-07)

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.
- Visualising univariate data with large number of categories by using rPackedBar package.

At the same time, you will also learn how to:

- reshape data by using tidyr package, and
- process, wrangle and transform data by using dplyr package.

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',
             'treemap', 'treemapify',
             'rPackedBar')
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:

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

Interactive Data Visualisation - ggiraph methods

- ggiraph is 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. Refer to this article for more detail explanation.

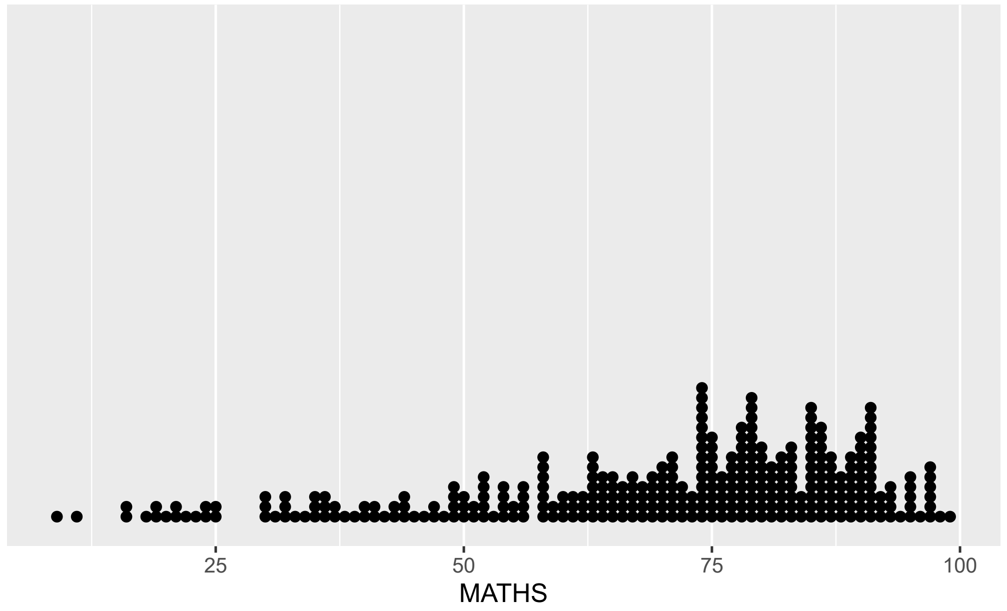
Reference: ggiraph package

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.

```r
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.

```r
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
)
```

Notice that two steps are involved. First, an interactive version of ggplot2 geom (i.e. `geom_dotplot_interactive()`) will be used to create the basic graph. Then, `girafe()` will be used to generate an svg object to be displayed on an html page.

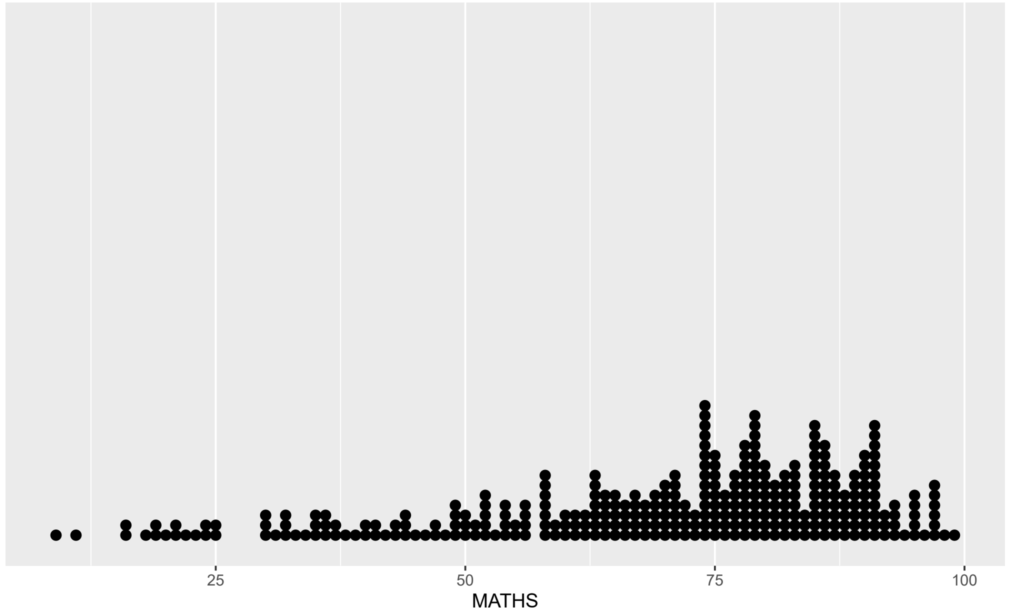
Displaying multiple information on tooltip

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

```r
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.



The first three lines of codes in the code chunk create a new field called *tooltip*. At the same time, it populates text in ID and CLASS fields into the newly created field. Next, this newly created field is used as tooltip field as shown in the code of line 7.

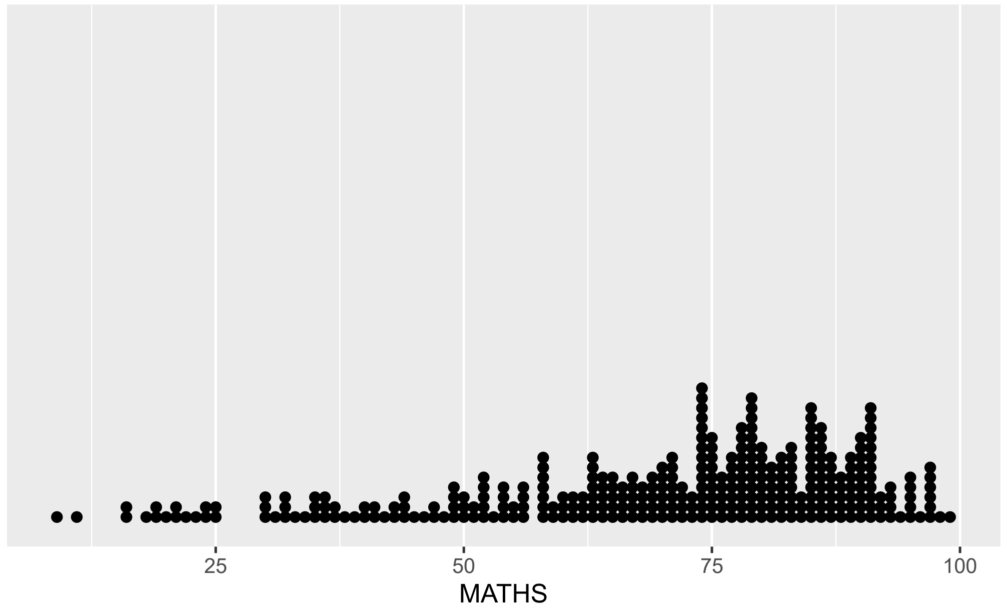
Customising Tooltip style

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.

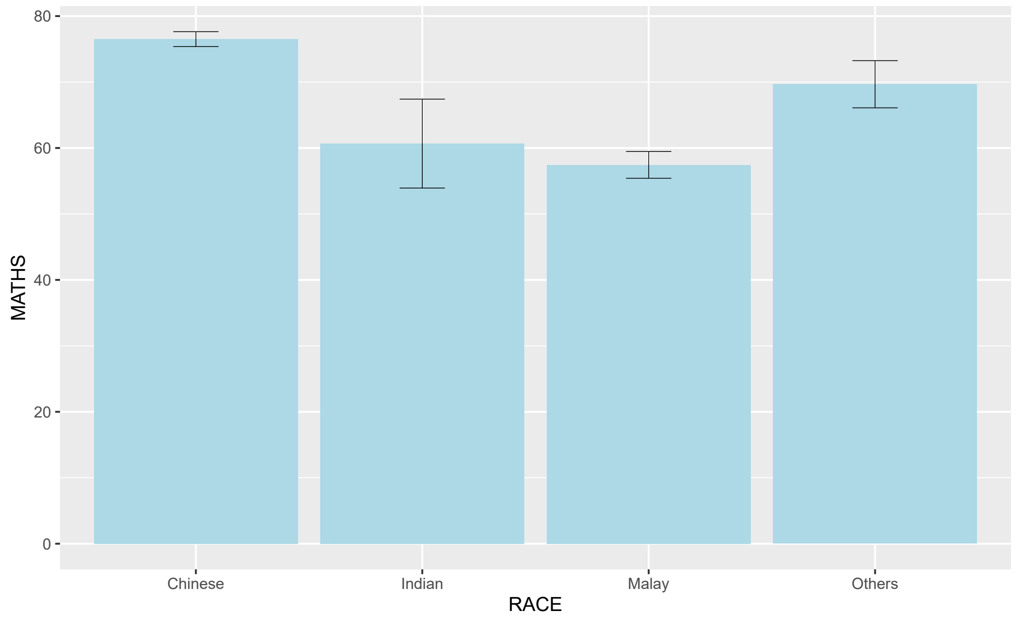
Displaying statistics on tooltip

```
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.

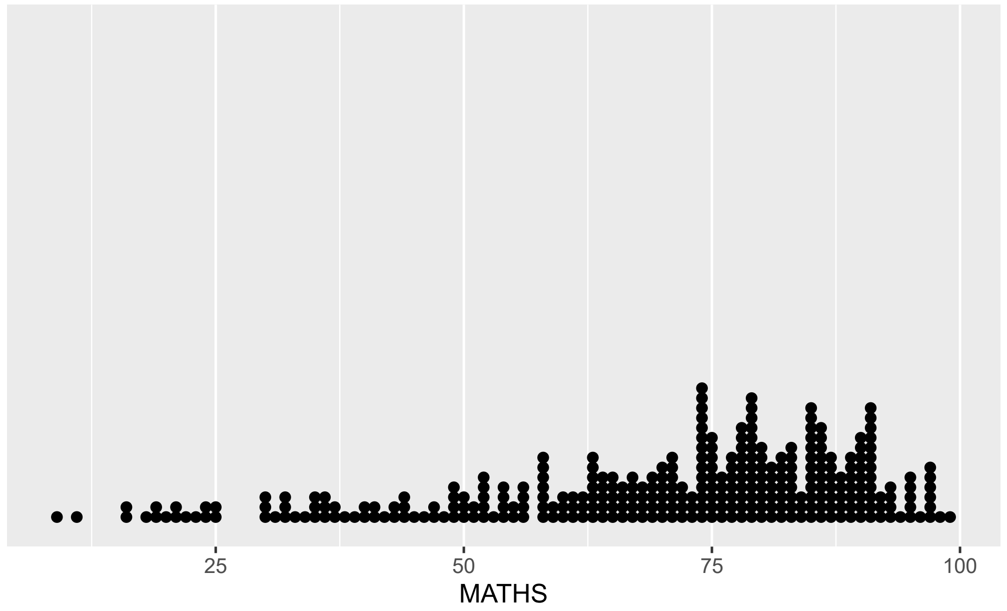


Hover effect with *data_id* aesthetic

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

```r
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.



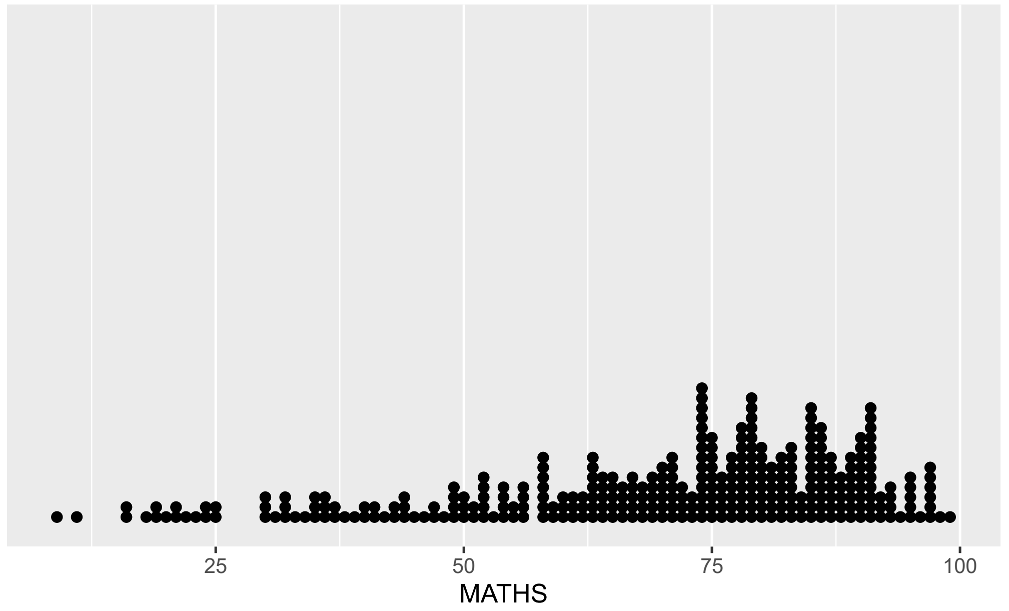
Note that the default value of the hover css is *hover_css = "fill:orange;"*.

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.



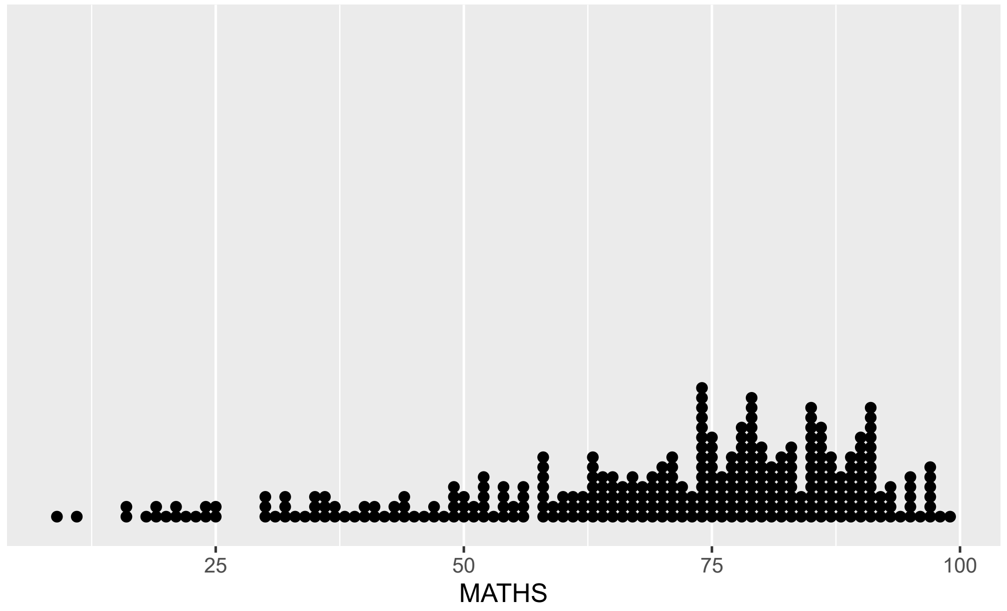
Note: Different from Slide 9, in this example the ccs customisation request are encoded directly.

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

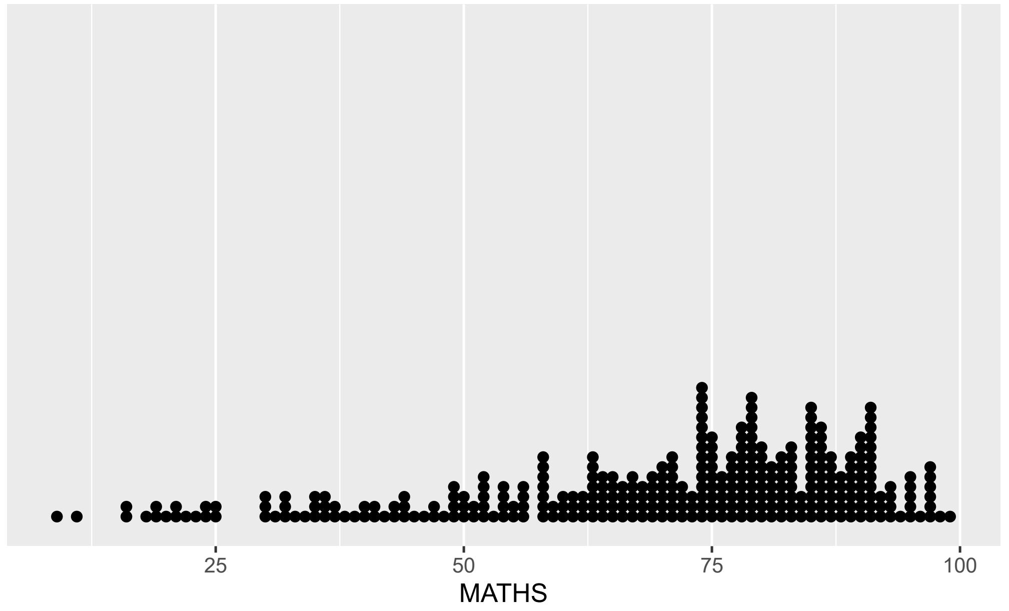
`onclick` argument of ggiraph provides hotlink interactivity on ther web.

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

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

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.

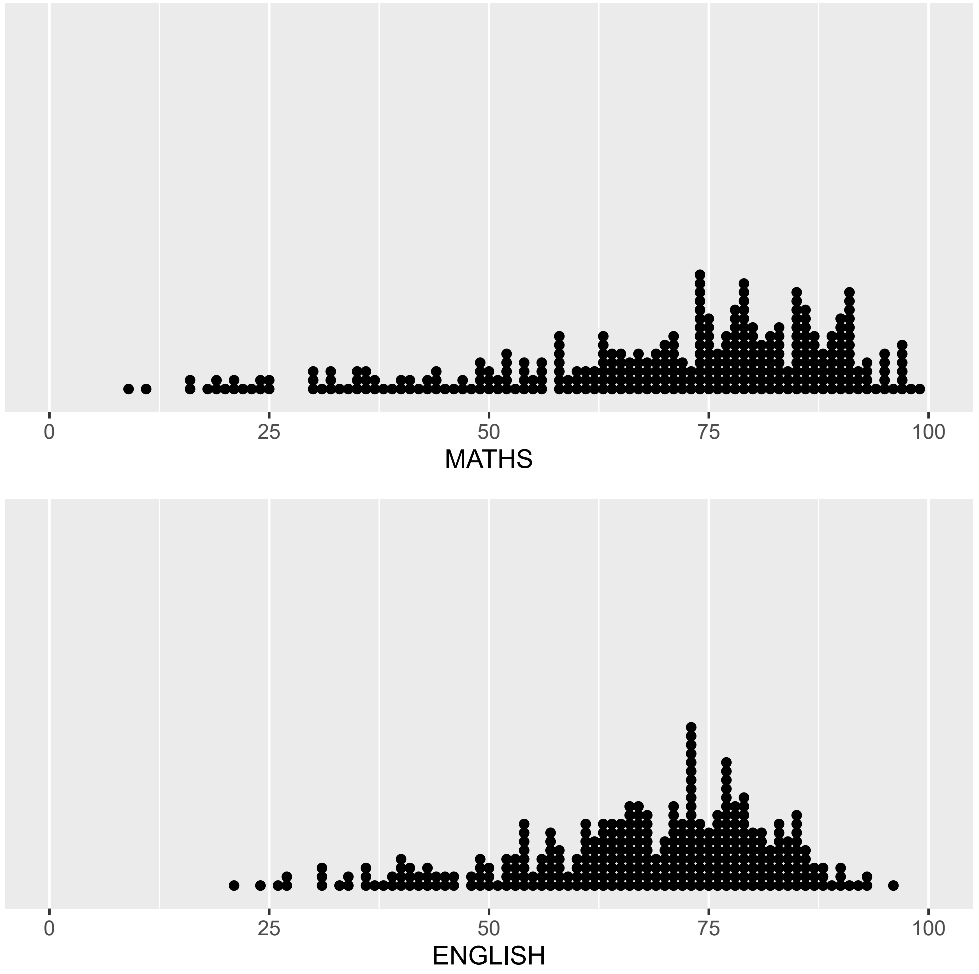


Note that click actions must be a string column in the dataset containing valid javascript instructions.

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)
```

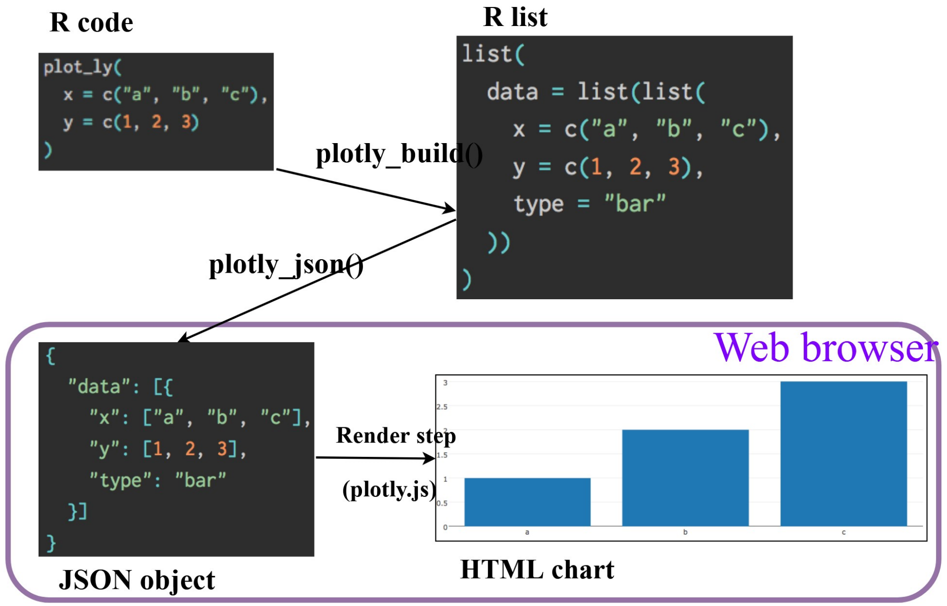
```
p2 <- ggplot(data=exam_data,
       aes(x = ENGLISH)) +
  geom_dotplot_interactive(
    aes(data_id = ID),
    stackgroups = TRUE,
    binwidth = 1,
    method = "histodot") +
  coord_cartesian(xlim=c(0,100)) +
  scale_y_continuous(NULL,
                     breaks = NULL)

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.



```
plot_ly(
  x = c("a", "b", "c"),
  y = c(1, 2, 3)
)
```

```
list(
  data = list(list(
    x = c("a", "b", "c"),
    y = c(1, 2, 3),
    type = "bar"
  ))
)
```

```
{
  "data": [{
    "x": ["a", "b", "c"],
    "y": [1, 2, 3],
    "type": "bar"
  }]
}
```

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

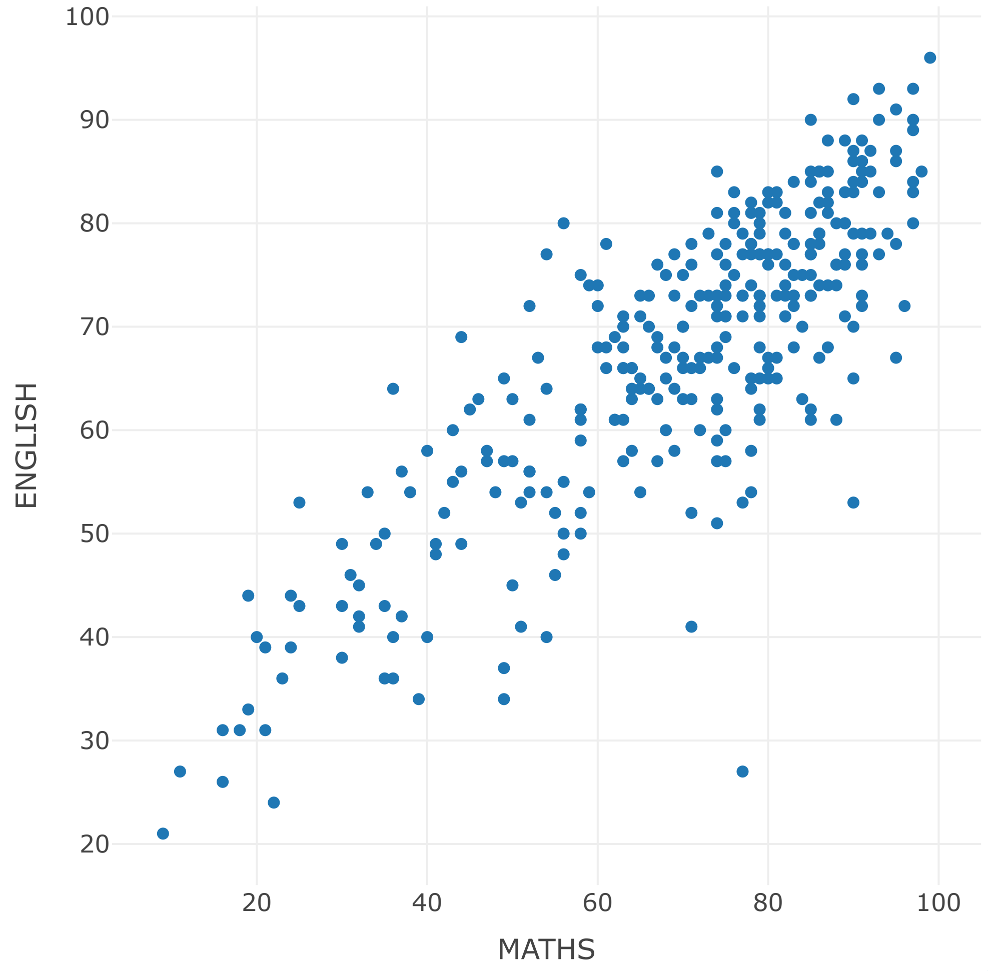
- by using *plot_ly()*, and
- by using *ggplotly()*

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:



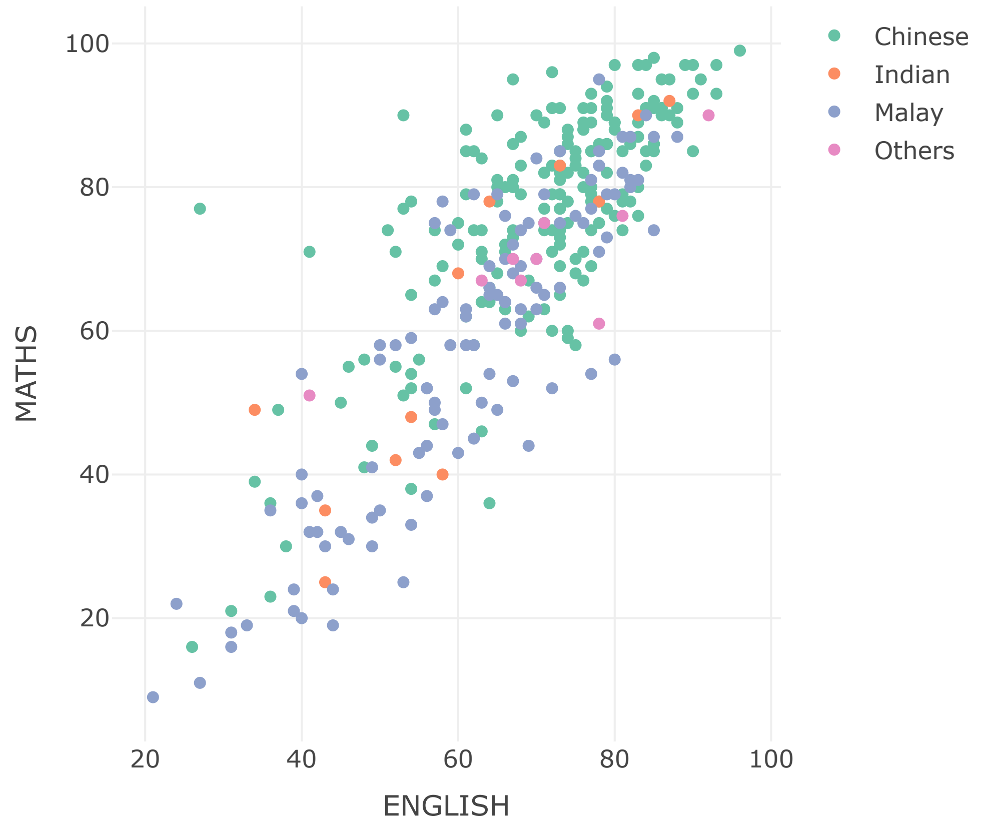
Working with visual variable: plot_ly() method

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.



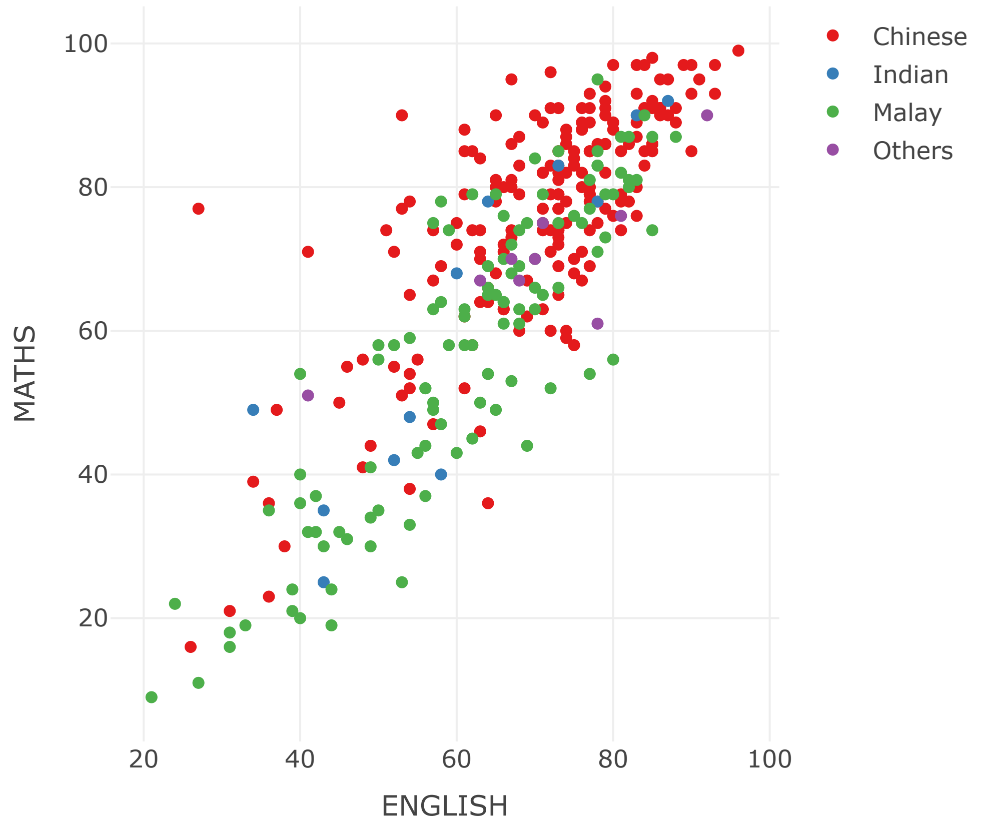
Changing colour pallete: plot_ly() method

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

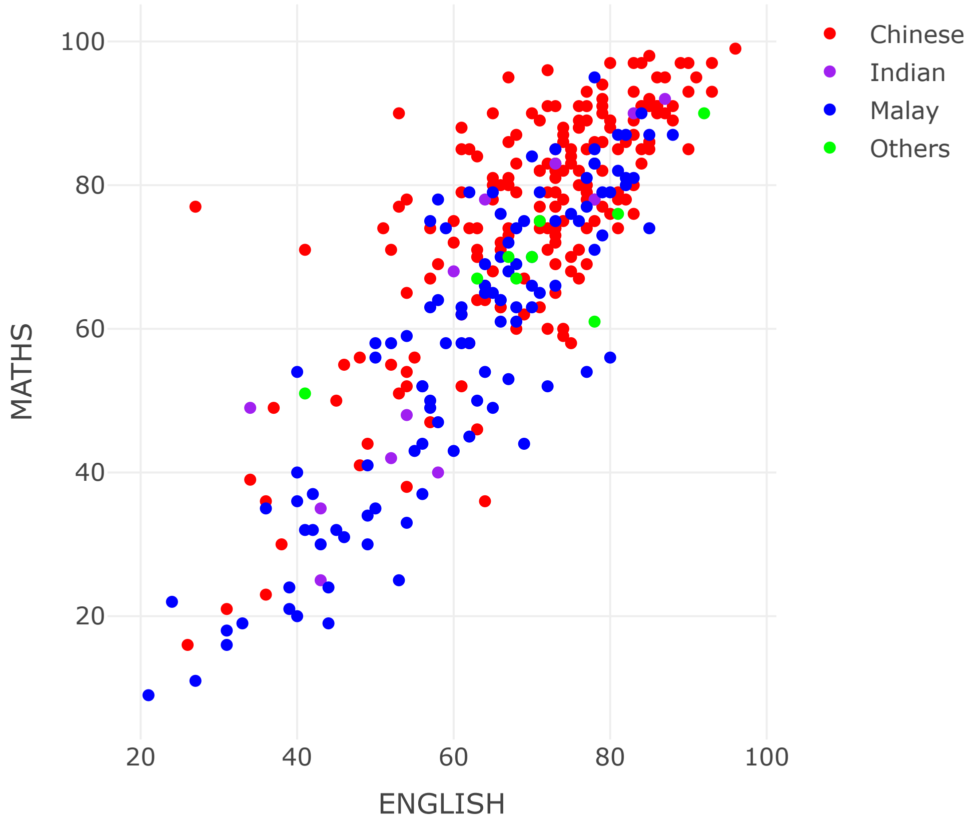
In the code chunk below, a customised colour scheme is created. Then, *colors* argument is used to change the default colour palette to the customised colour scheme.

```
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.



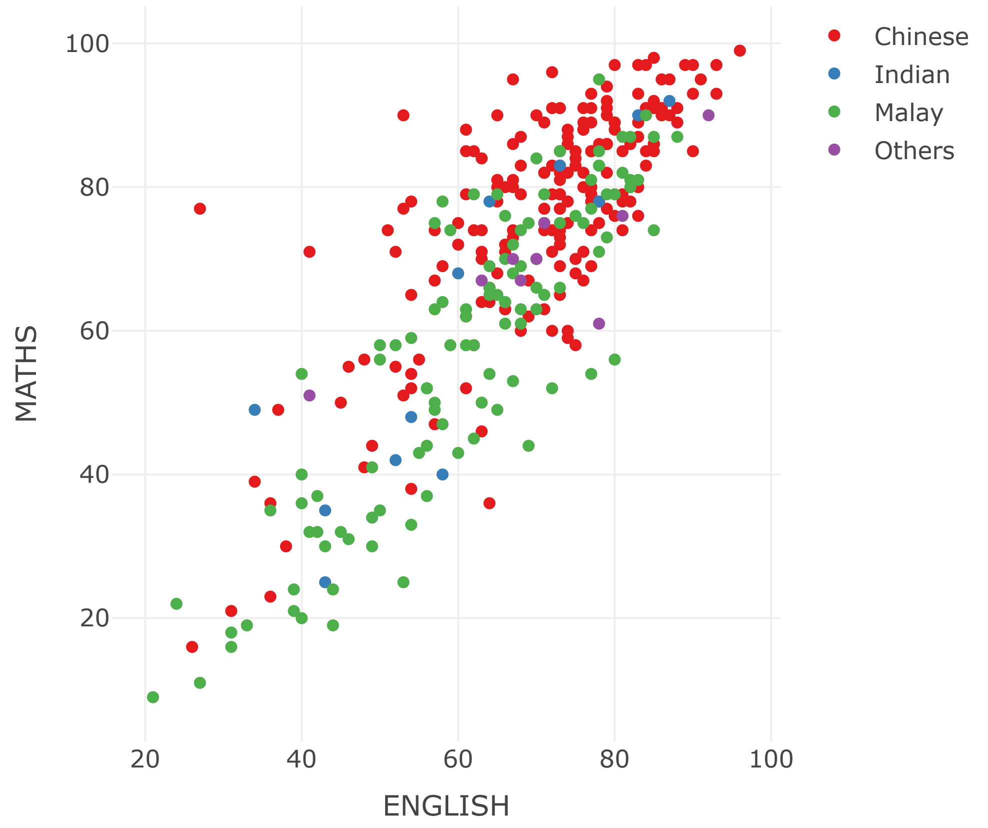
Customising tooltip: plot_ly() method

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

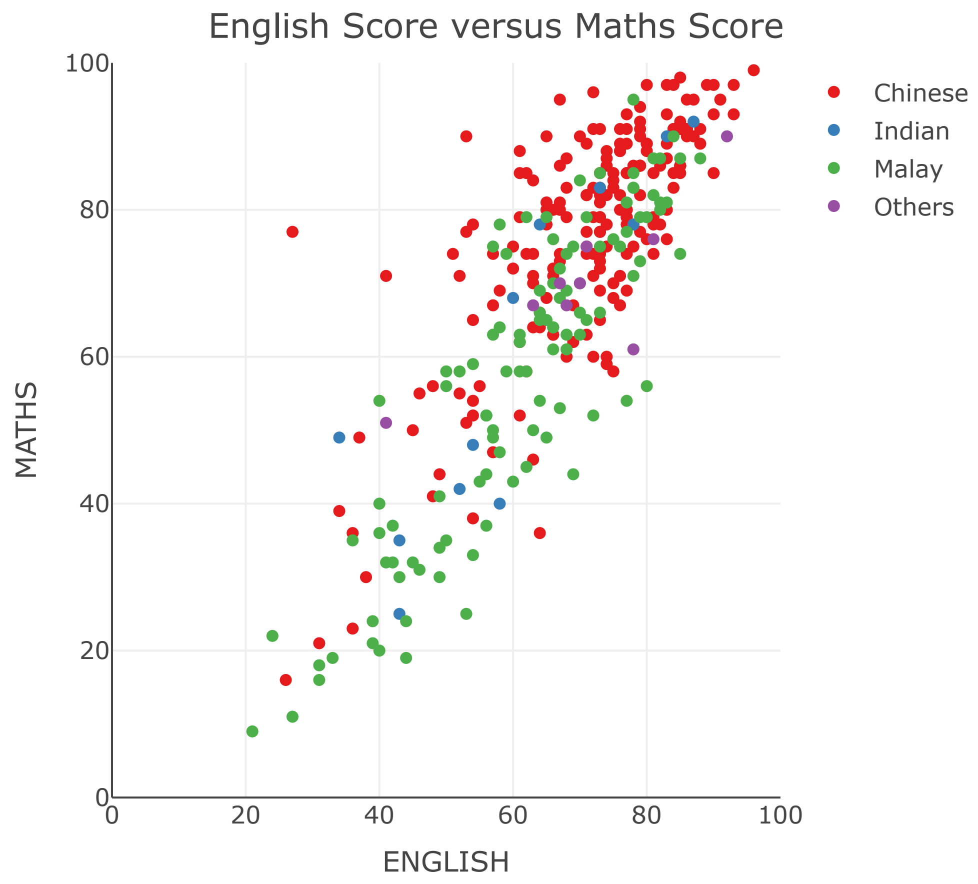
In the code chunk below, *layout* argument is used to change the default tooltip.

```
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.

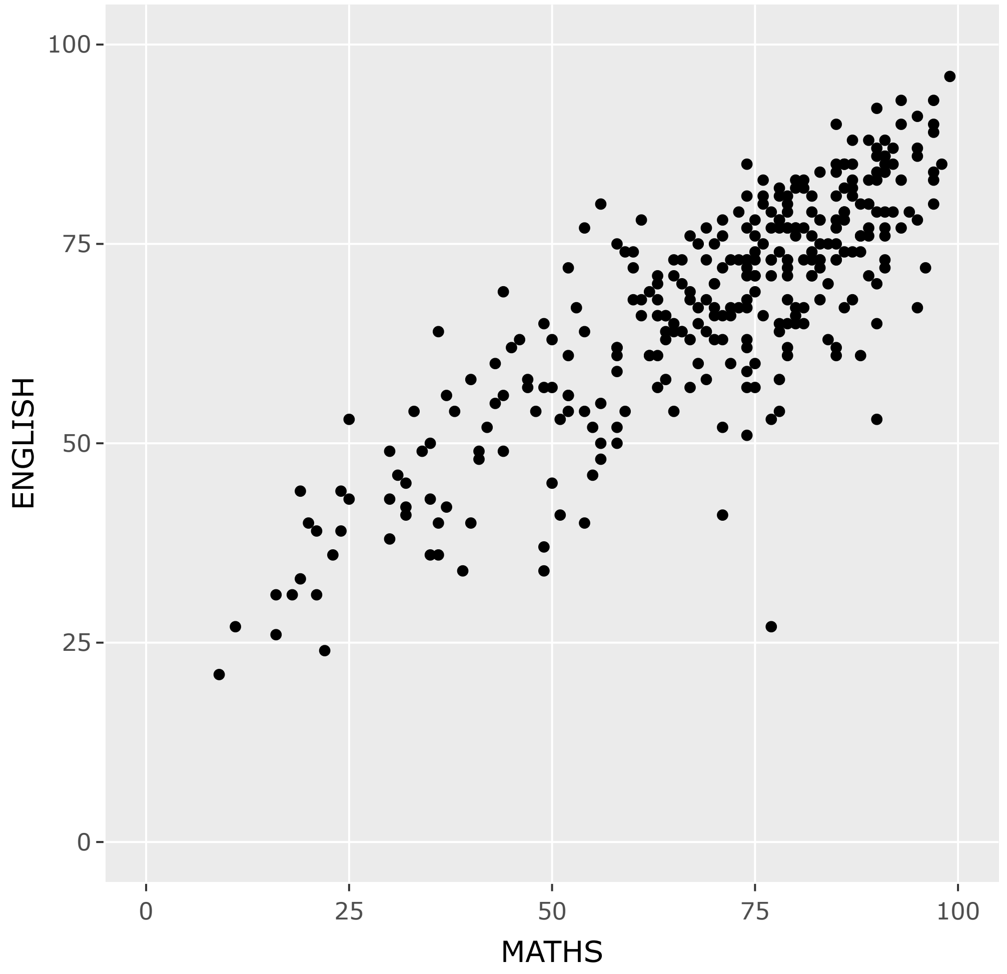


Creating an interactive scatter plot: ggplotly() method

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()*.



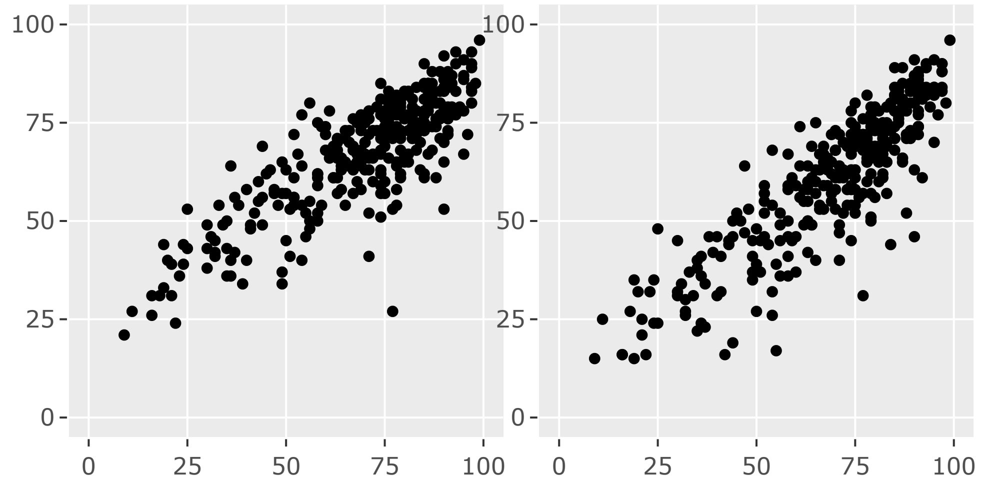
Coordinated Multiple Views with plotly

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.

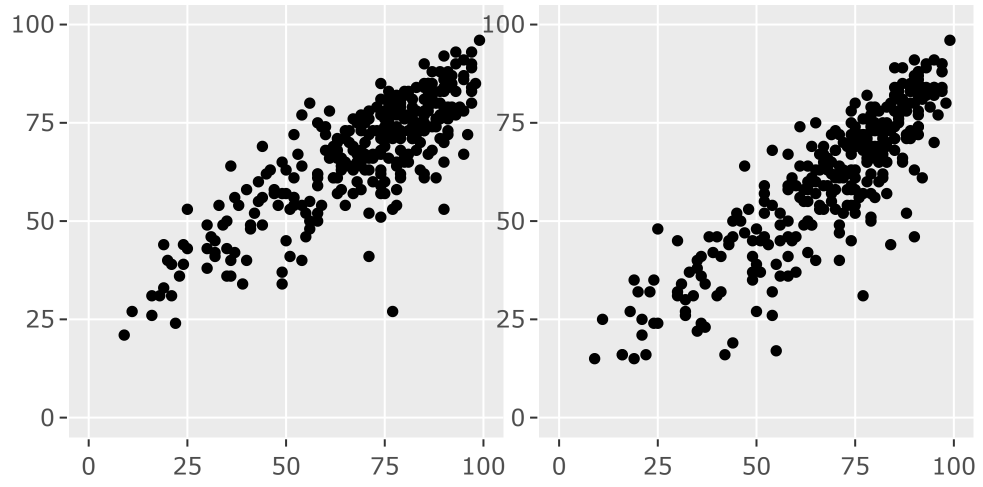
Coordinated Multiple Views with plotly

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

```r
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

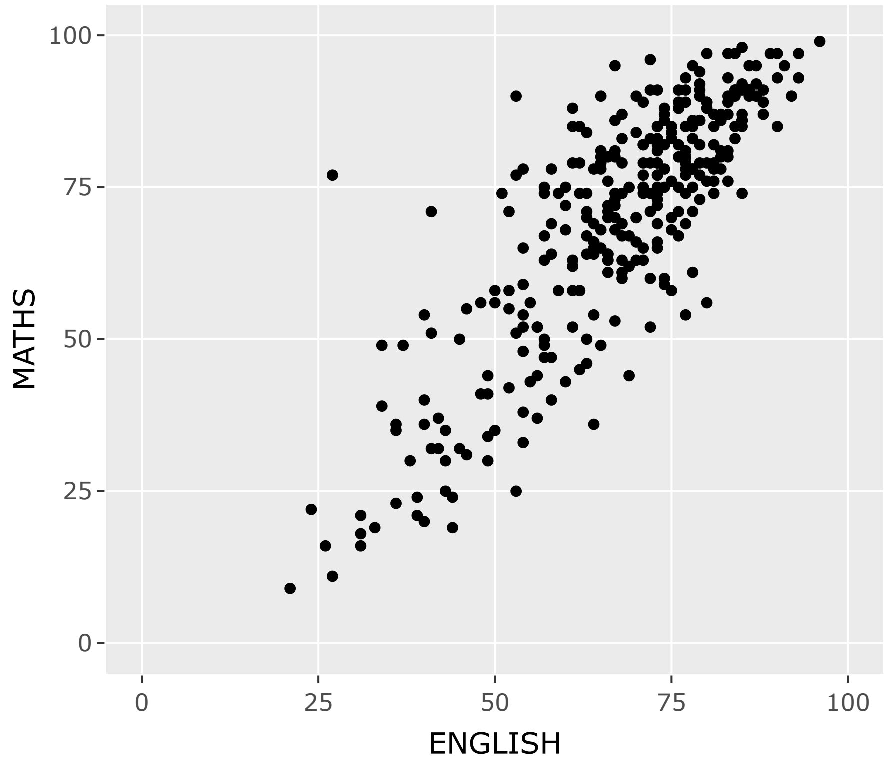
- Data objects in R can be rendered as HTML tables using the JavaScript library 'DataTables' (typically via R Markdown or Shiny).

```r
DT::datatable(exam_data, class= "compact")
```

Show 10 entries Search:

	ID	CLASS	GENDER	RACE	ENGLISH	MATHS	SCIENCE	tooltip	or
1	Student321	3I	Male	Malay	21	9	15	Name = Student321 Class = 3I	window.open("https://ww journey=Primary%20scho
2	Student305	3I	Female	Malay	24	22	16	Name = Student305 Class = 3I	window.open("https://ww journey=Primary%20scho
3	Student289	3H	Male	Chinese	26	16	16	Name = Student289 Class = 3H	window.open("https://ww journey=Primary%20scho

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.

```r
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.

```
col <- c("Country", "Continent")
globalPop <- read_xls("data/GlobalPopulation.x
                      sheet="Data") %>%
  mutate_each_(funs(factor(.)), col) %>%
  mutate(Year = as.integer(Year))
```

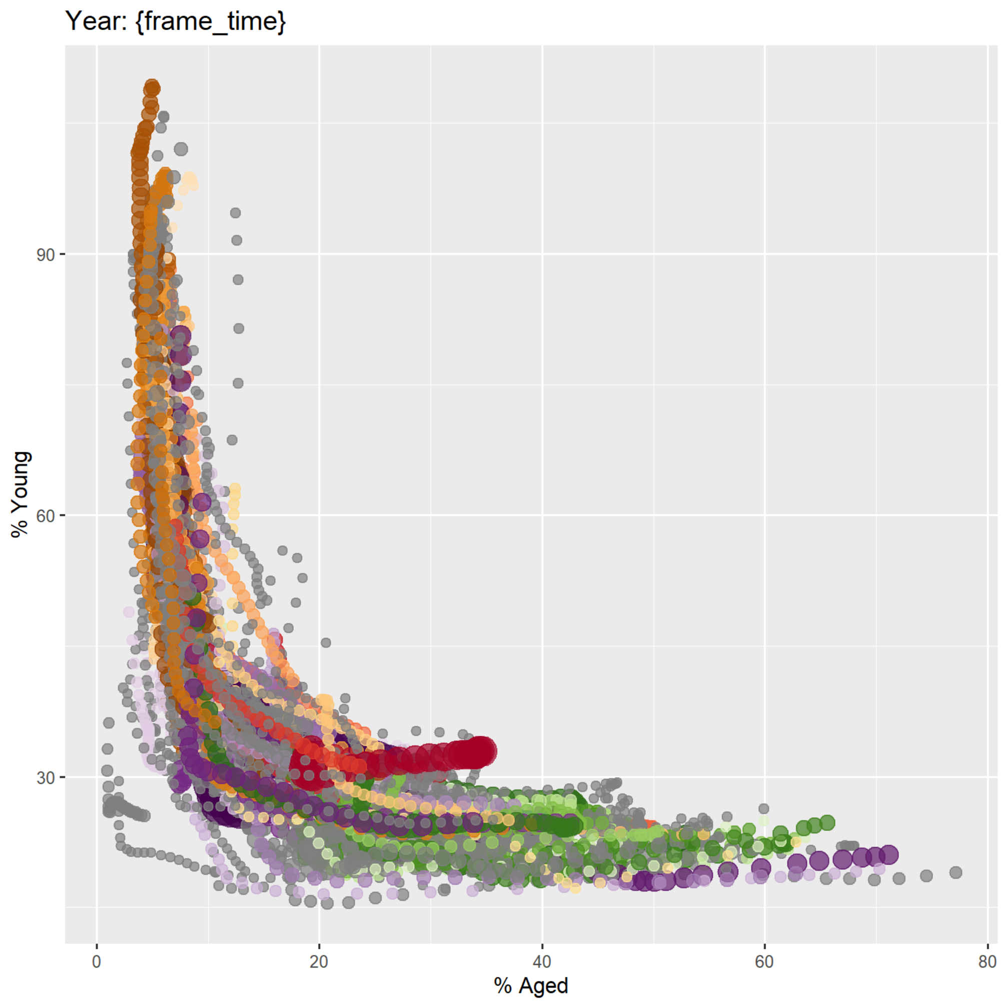
Things to learn from the code chunk above:

- `read_xls()` of **readxl** package is used to import the Excel worksheet.
- `mutate_each_()` of dplyr package is used to convert all character data type into factor.
- `mutate` of dplyr package is used to convert data values of Year field into integer.

Building a static population bubble plot

In the code chunk below, the basic ggplot2 functions are used to create a static 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')
```



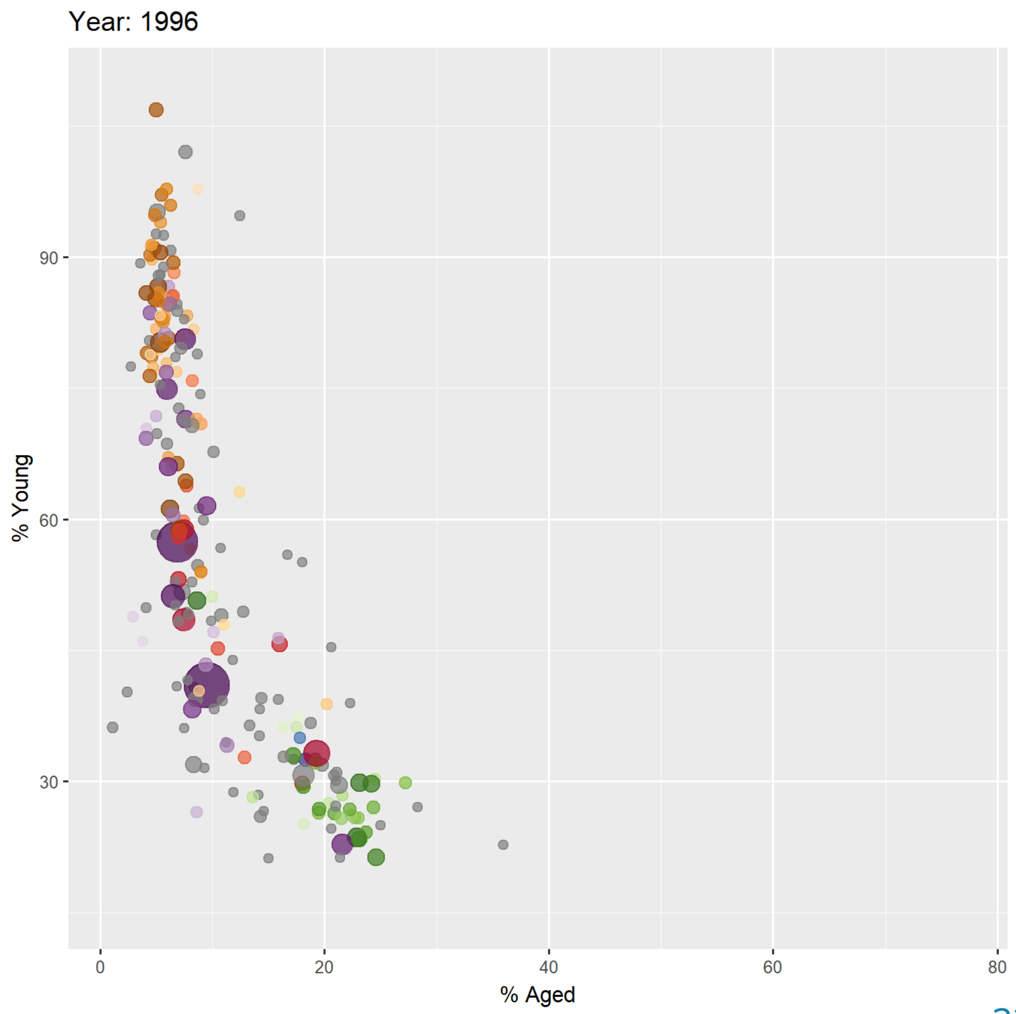
Building the animated bubble plot

In the code chunk below,

- `transition_time()` of **gganimate** is used to create transition through distinct states in time (i.e. Year).
- `ease_aes()` is used to control easing of aesthetics. The default is `linear`. Other methods are: quadratic, cubic, quartic, quintic, sine, circular, exponential, elastic, back, and bounce.

```
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



Visualising Large Data Interactively

In this hands-on exercise you will learn how to visualise large data by using treemap and packed bar methods. For the purpose of this hands-on exercise, two data sets will be used. They are:

- *GDP.csv* provides GDP, GDP per capita and GDP PPP data for world countries from 2000 to 2020. The data was extracted from World Development Indicators Database of World Bank.
- *WorldCountry.csv* provides a list of country names and the continent they belong to extracted from Statistics Times.

- Write a code chunk to import both data sets by using `read_csv()` of **readr** package.

The solution:

```
GDP <- read_csv("data/GDP.csv")
WorldCountry <- read_csv("data/WorldCountry.cs
```

Note: It is always a good practice to check the data structure and examine data values in RStudio.

Data preparetion

Before programming the data visualisation, it is important for us to reshape, wrangle and transform the raw data to meet the data visualisation need.

Code chunk below performs following tasks:

- `mutate()` of dplyr package is used to convert all values in the 202 field into numeric data type.
- `select()` of dplyr package is used to extract column 1 to 3 and Values field.
- `pivot_wider()` of tidyr package is used to split the values in Series Name field into columns.
- `left_join()` of dplyr package is used to perform a left-join by using Country Code of GDP_selected and ISO-alpha3 Code of WorldCountry tibble data tables as unique identifier.

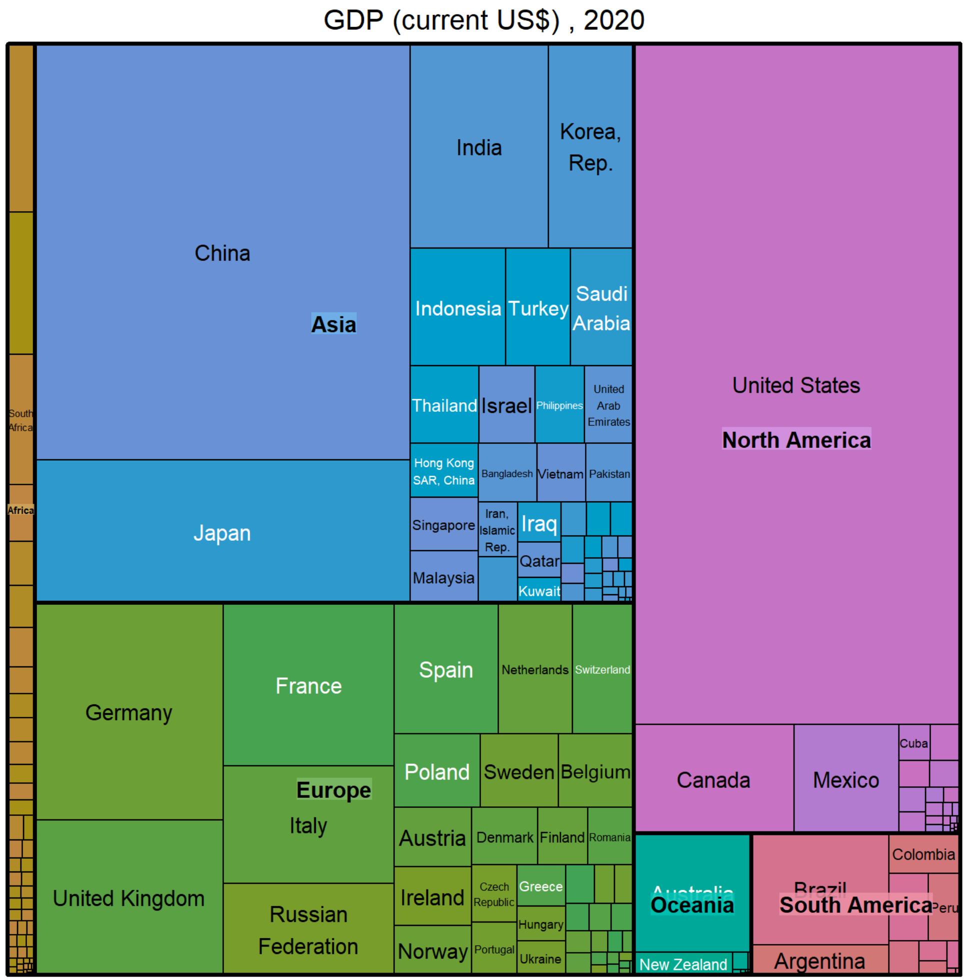
```r
GDP_selected <- GDP %>%
  mutate(Values = as.numeric(`2020`)) %>%
  select(1:3, Values) %>%
  pivot_wider(names_from = `Series Name`,
              values_from = `Values`) %>%
  left_join(y=WorldCountry, by = c("Country Code" = "ISO-alpha3 Code"))
```

Building static treemap using treemap package

treemap is an R package specially designed to plot treemap. It provides many functions to customise a treemap.

The code chunk below designed a static treemap by using three core arguments of `treemap()` of treemap package, namely: `index`, `vSize` and `vColor`.

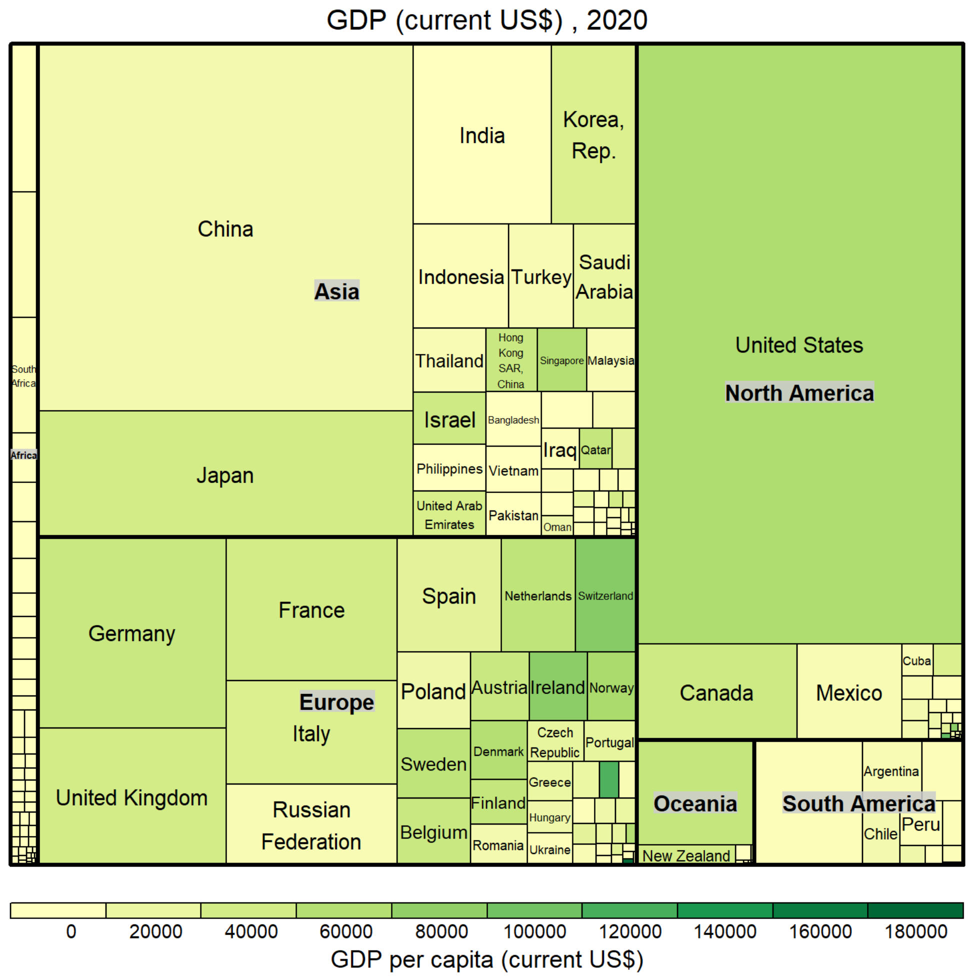
```
treemap(GDP_selected,
        index=c("Continent", "Country Name"),
        vSize="GDP (current US$)",
        vColor="GDP (current US$)",
        title="GDP (current US$) , 2020",
        title.legend = "GDP per capita (curren
        )
```



Working with vColor and type arguments

In the code chunk below, `type` argument is define as `value`.

```
treemap(GDP_selected,
        index=c("Continent", "Country Name"),
        vSize="GDP (current US$)",
        vColor="GDP per capita (current US$)",
        type = "value",
        title="GDP (current US$) , 2020",
        title.legend = "GDP per capita (curren
        )
```



Things to learn from the code chunk above:

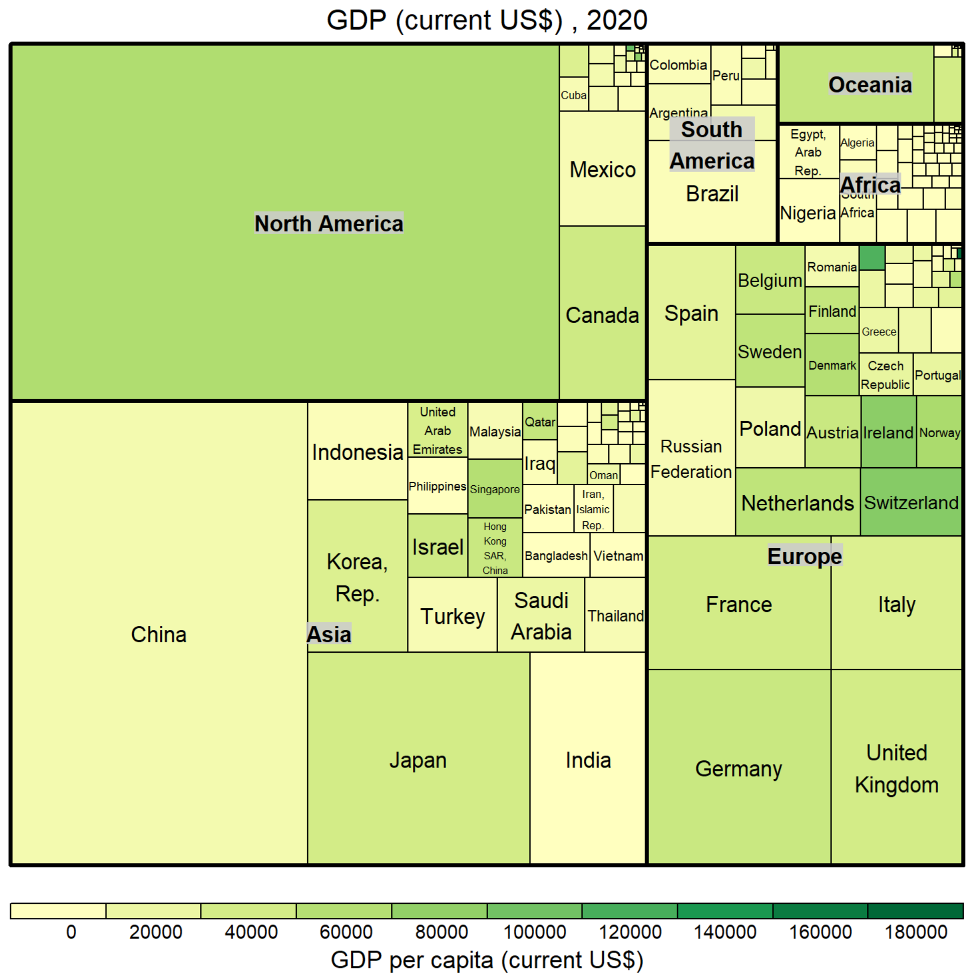
- the size of the rectangle is mapped to *GDP (current US$)* by using `vSize` argument.
- the colour of the rectangle is now mapped to *GDP per capita (current US$)* by using `vColor` argument.

Working with three layout

treemap() supports two popular treemap layouts, namely: “squarified” and “pivotSize”. The default is “pivotSize”.

The squarified treemap algorithm (Bruls et al., 2000) produces good aspect ratios, but ignores the sorting order of the rectangles (sortID). The ordered treemap, pivot-by-size, algorithm (Bederson et al., 2002) takes the sorting order (sortID) into account while aspect ratios are still acceptable.

```
treemap(GDP_selected,
        index=c("Continent", "Country Name"),
        vSize="GDP (current US$)",
        vColor="GDP per capita (current US$)",
        type = "value",
        algorithm = "squarified",
        title="GDP (current US$) , 2020",
        title.legend = "GDP per capita (curren
        )
```



Building an interactive treemap using d3treeR package

d3treeR is an R htmlwidget for d3.js treemaps. It is designed to integrate seamlessly with the R treemap package.

d3treeR package is not in RCran, you will load the devtools library and install the package found in github by using the codes below.

```r
library(devtools)
install_github("timelyportfolio/d3treeR")
```

Note: You should only run the code chunk above once.

Next, you will launch d3treeR package by using the `library()`

```r
library(d3treeR)
```

Building an interactive treemap using d3treeR package

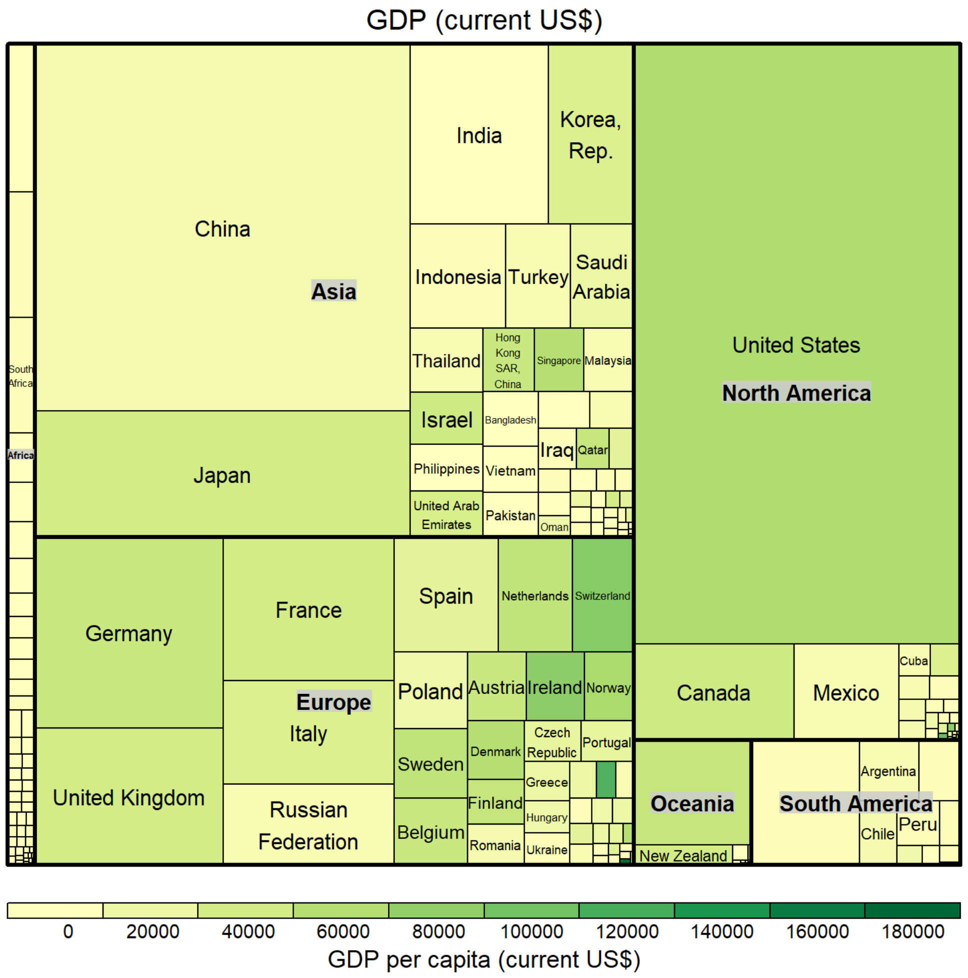
Step 1: Create a statistic treemap by using treemap().

Before you can build an interactive treemap by using d3treeR package, you need to build a static treemap by using `treemap()` as shown in the code chunk below.

```
tm <- treemap(GDP_selected,
        index=c("Continent", "Country Name"),
        vSize="GDP (current US$)",
        vColor="GDP per capita (current US$)",
        type = "value")
```

Things to learn from the code chunk above:

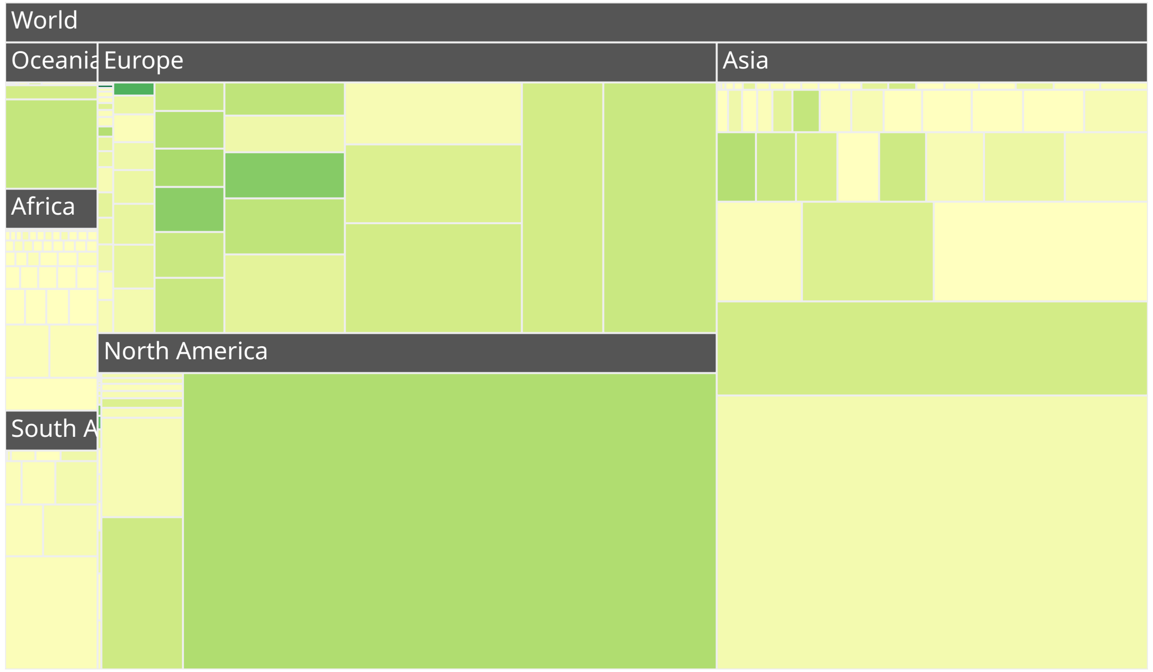
- Only index, vSize, vColor and type arguments are used because d3treeR only support these four arguments.
- The output treemap is save as an object called *tm*.



Building an interactive treemap using d3treeR package

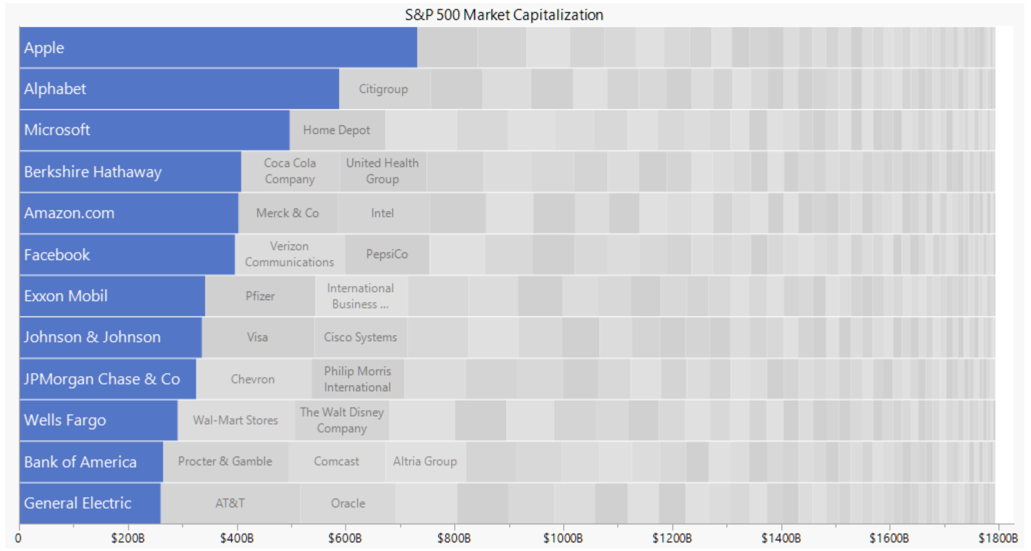
Step 2: Using d3tree() to create an interactive treemap.

```
d3tree(tm, rootname = "World" )
```



Visualising Large Data Interactively - packed bar method

- **packed bar** is a relatively new data visualisation method introduced by Xan Gregg from JMP.
- It aims to support the need of visualising skewed data over hundreds of categories.
- The idea is to support the Focus+Context data visualization principle.
- Visit this JMP Blog to learn more about the design principles of packed bar.



Data Preparation

Step 1: Prepare the data by using the code chunk below.

As usual, we need to prepare the data before building the packed bar.

```r
GDP_selected <- GDP %>%
  mutate(GDP = as.numeric(`2020`)) %>%
  filter(`Series Name` == "GDP (current US$)") %>%
  select(1:2, GDP) %>%
  na.omit()
```

Thing to learn from the code chunk above:

- `na.omit()` is used to exclude rows with missing values. This is because rPackedBar package does not support missing values.

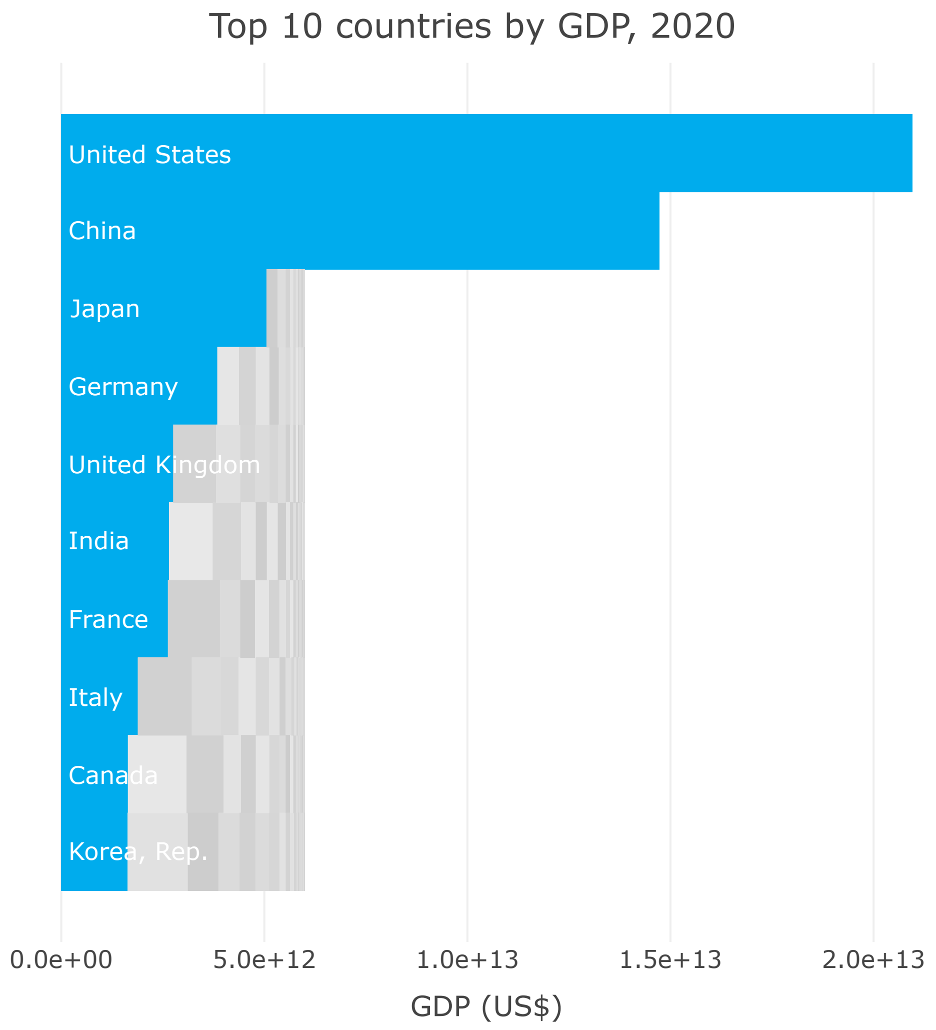
Building a packed bar by using rPackedBar package.

In the code chunk below, `plotly_packed_bar()` of rPackedBar package is used to create an interactive packed bar.

```
p = plotly_packed_bar(
  input_data = GDP_selected,
  label_column = "Country Name",
  value_column = "GDP",
  number_rows = 10,
  plot_title = "Top 10 countries by GDP, 2020"
  xaxis_label = "GDP (US$)",
  hover_label = "GDP",
  min_label_width = 0.018,
  color_bar_color = "#00aced",
  label_color = "white")
plotly::config(p, displayModeBar = FALSE)
```

- Read this Vignettes and the user guide to learn more about the package.

The output packed bar:



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.
- Building an animation step-by-step with gganimate.
- Creating a composite gif with multiple gganimate panels

Packed Bar

rPackedBar: Packed Bar Charts with 'plotly'

- Visualizing Twitter Data with a Packed Barchart