Interactive Visualization of Population Aging with Plotly

The Plotly R package creats interactive web-based graphs via plotly.js. It can run locally in the web browser or in the R studio viewer. This tutorial will provide an introduction to the features of Plotly package, and will use R studio to reproduce three interactive charts originally produced in ggplot2.

First of all, we need to install and import the plotly and tidyverse (Alternatively, install just ggplot2) packages.

Installation

```
knitr::opts_chunk$set(message = F,warning = F,error = F)
```

```
# Install packages if not on computer:
# install.packages("plotly")
# install.packages("ggplot2")

# Load packages
library(plotly)
library(tidyverse)
```

Dataset 1: The Change in Population Age Structure

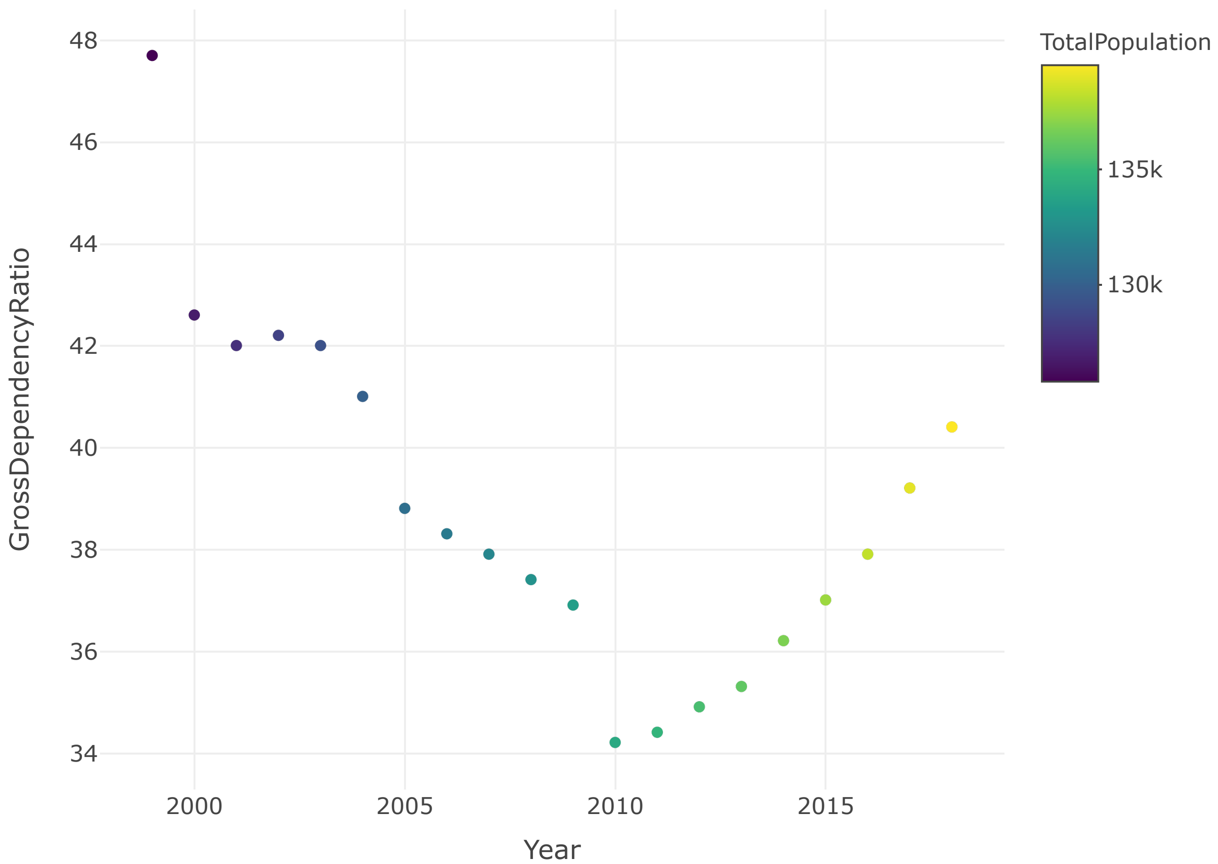
We will use the plot_ly() function to explore the “annual” dataset and create a stacked bar chart. This dataset deals with the age structure in China in the last two decades and gives insight into the number of people of nonworking age compared to the number of those of working age based on child and old dependency ratio.

```
# Load the dataset
annual <- read_csv("Annual.csv")

head(annual)
```

```
## # A tibble: 6 x 8
##    Year TotalPopulation Aged15_64 GrossDependency~ Children   Old
##   <dbl>           <dbl>     <dbl>            <dbl>    <dbl> <dbl>
## 1  2018          139538     99357             40.4     23.7  16.8
## 2  2017          139008     99829             39.2     23.4  15.9
## 3  2016          138271    100260             37.9     22.9  15
## 4  2015          137462    100361             37       22.6  14.3
## 5  2014          136782    100469             36.2     22.5  13.7
## 6  2013          136072    100582             35.3     22.2  13.1
## # ... with 2 more variables: NonWorkingProp <dbl>, workingprop <dbl>
```

If we assign variable names (Year and GrossDependencyRatio) to visual properties (e.g., x, y, color, etc) within plot_ly(), as done below, it tries to find a sensible geometric representation of that information for us. The plot_ly() function has numerous arguments that are unique to the R package (e.g., color, stroke, span, symbol, linetype, etc) and make it easier to encode data variables as visual properties (e.g., color). The default type of a plotly plot is “scatter”.

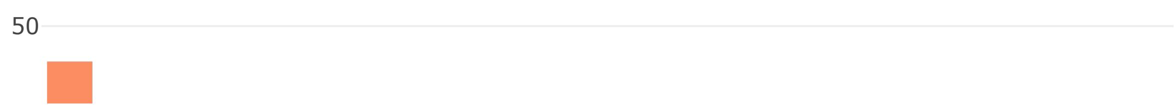
```
# create visualizations of the dataset - Gross Dependency Ratio over years
plot_ly(annual, x = ~Year, y = ~GrossDependencyRatio, color = ~TotalPopulation)
```

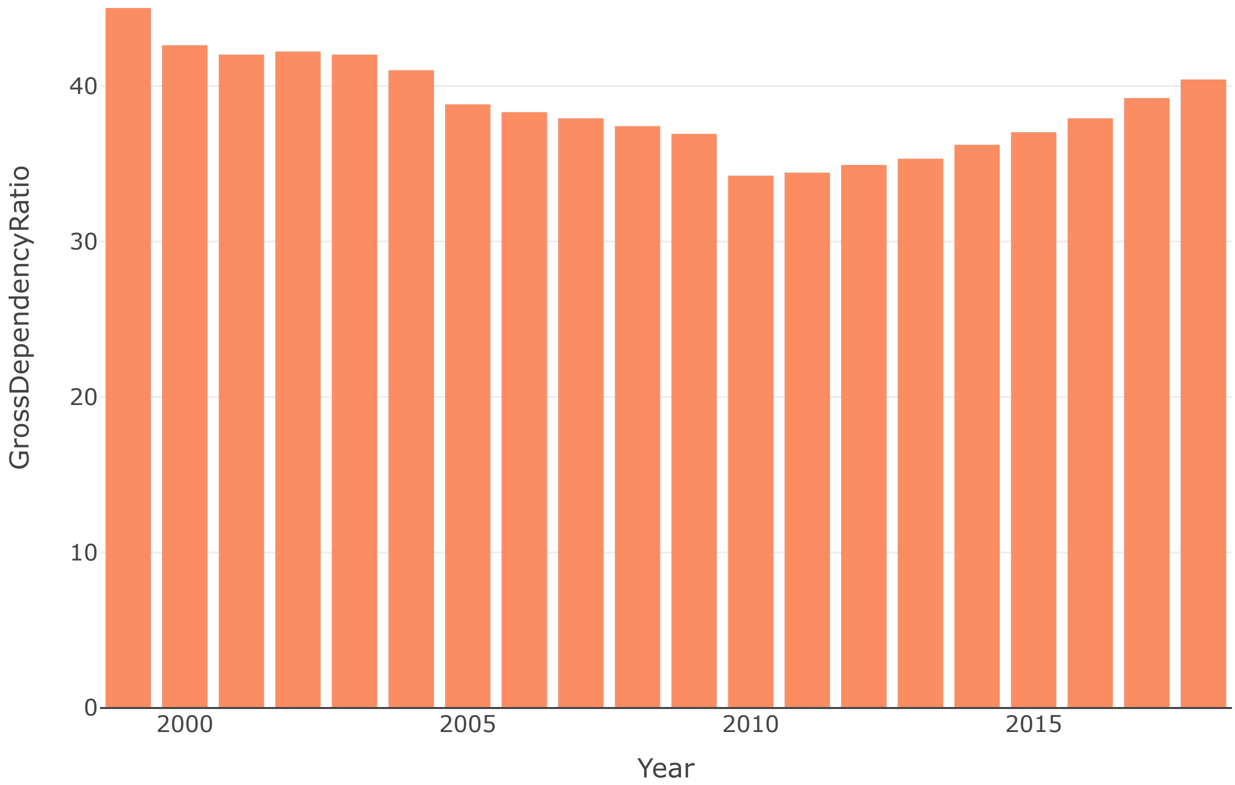


Basic interactions such as zooming, hovering and panning and can be accomplished in the plotly graph. Hovering over a point will bring up a tooltip with the year and gross dependency ratio expressed between 0 and 100, and clicking/dragging/scrolling will pan and zoom on the plot.

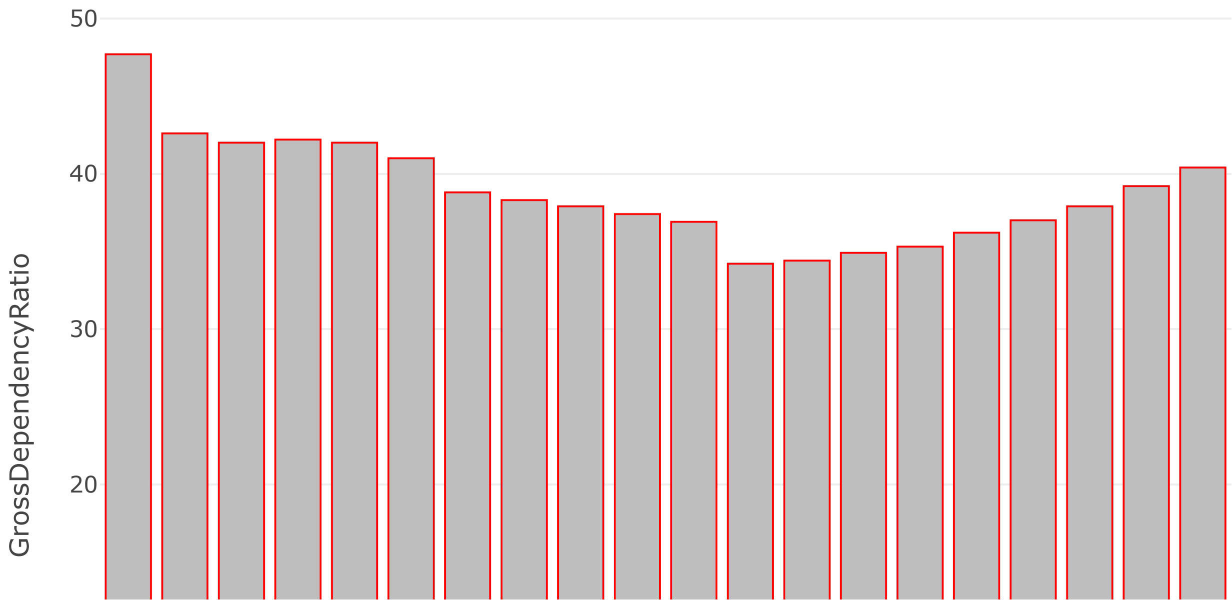
By default, these arguments map values of a data variable to a visual range. To specify the visual range directly, use the I() function to declare this value.

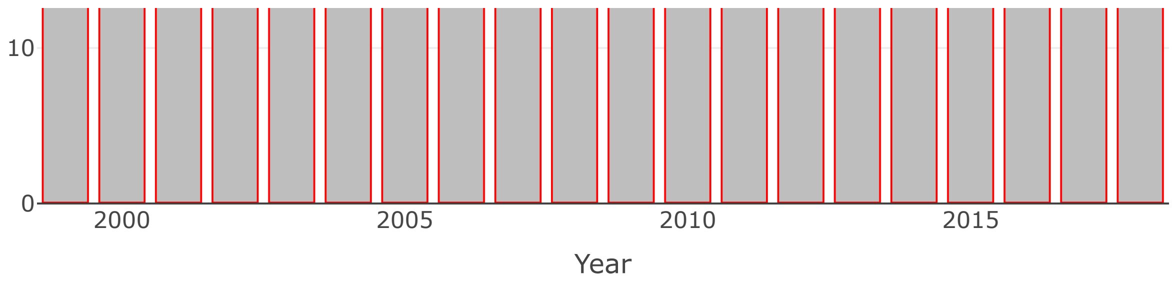
```
# doesn't produce black bars

plot_ly(annual, x = ~Year, y = ~GrossDependencyRatio, color = "black", type = "bar")
```






```r
# produces grey bars with red outline
plot_ly(
  annual,
  x = ~Year,
  y = ~GrossDependencyRatio,
  type = "bar",
  # fill color
  color = I("grey"),
  # line color
  stroke = I("red"),
  # stroke size
  span = I(1)
)
```





Plotly’s graph description places attributes into two categories: traces (describing a single series of data in a graph) and layout attributes (applying to the rest of the chart, like the title, xaxis, or annotations).

- Add_trace()

A plotly visualization is composed of one or more traces, and every trace has a type. The arguments that a trace will respect depend on it’s type. In this case, we will change the type into “bar” and set the opacity with “alpha”.

We can manually add a trace to an existing plot with add_trace(). In that case, we can choose to either name the traeces, or hide the legend by setting showlegend = FALSE.

- Layout()

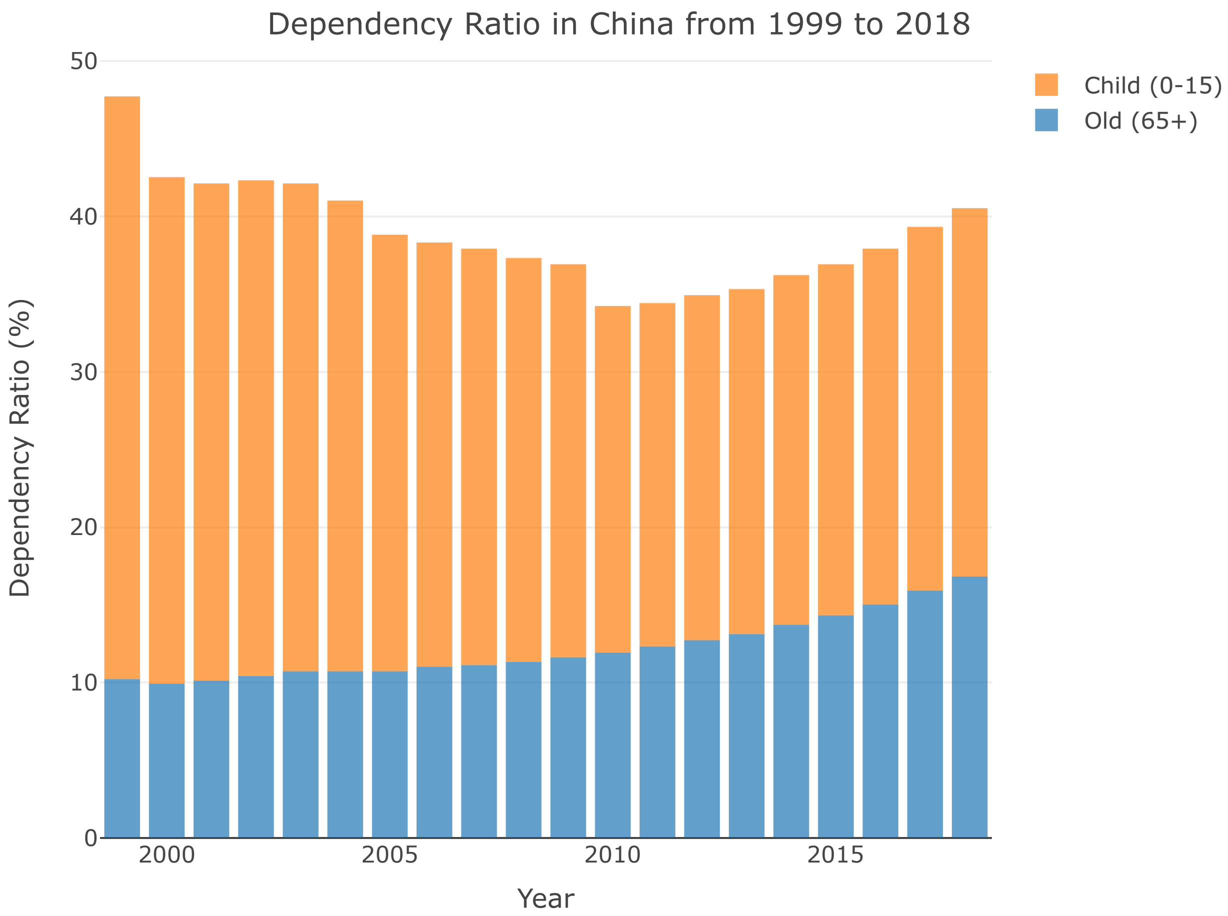
The layout() function is designed for adding or modifying parts of the graph’s layout. A layer can be thought of as a group of graphical elements that can be sufficiently described using only 5 components: data, aesthetic mappings (e.g., assigning year to x and old dependency ratio to y), a geometric representation (e.g. rectangles, circles, etc), statistical transformations (e.g., sum, mean, etc), and positional adjustments (e.g., dodge, stack, etc).

Furthermore, plot_ly(), add_trace(), and layout(), all accept a data frame as their first argument and output a data frame. As a result, we can integrate data manipulations and visual mappings in a single pipeline.

- Hoverinfo

The hoverinfo attribute controls what other plot attributes are shown into the tooltip text. The default value of hoverinfo is x+y+text+name, meaning that plotly.js will use the relevant values of x, y, text, and name to populate the tooltip text. As in the bar chart below shows, we can supply custom text by supplying a character string text and setting hoverinfo = “text”.

```
# Visualizing dependency ratio over time
  # Step 1: plot a single bar chart using old depdendency ratio
plot_ly(data = annual, x = ~Year, y = ~Old, type = 'bar', name = 'Old (65+)', alpha = 0.
7,
        hoverinfo = text, text = ~paste('Year: ', Year, '</br> DepRatio: ', Old) ) %>%

  # Step 2: add a trace (child dependency ratio)
  add_trace(y = ~Children, name = 'Child (0-15)', hoverinfo = text, text = ~paste('Year:
', Year, '</br> DepRatio: ', Children)) %>%

  # Step 3: add title and label
  layout(title = 'Dependency Ratio in China from 1999 to 2018',titlefont = list(size = 1
6),
         yaxis = list(title = 'Dependency Ratio (%)'),
  # specify bar mode
         barmode = 'stack')
```



A stacked bar chart is useful for comparing the dependecy ratio based on two aspects of the child and old. For instance, old dependency ratio has been increasing over the last two decades, but it is hard to see the trend for child dependency ratio in a static plot. The interactive figure above makes this comparison easier by allowing users to clicking on the upper right legend: the child dependency ratio shows a downward tredn in general.

Dataset 2: Growth Rate of Population

The plot_ly() function supports key frame animations through the frame argument. Next we will recreate the animation of the evolution of population growth rates evolved over time.

```
eap <- read_csv("eap.csv")
head(eap)
```

```
## # A tibble: 6 x 3
##    Year Type       GrowthRate
##   <dbl> <chr>           <dbl>
## 1  2017 WorkingAge      -0.01
## 2  2016 WorkingAge       0.75
## 3  2015 WorkingAge       0.5
## 4  2014 WorkingAge       0.49
## 5  2013 WorkingAge       0.51
## 6  2012 WorkingAge       0.4
```

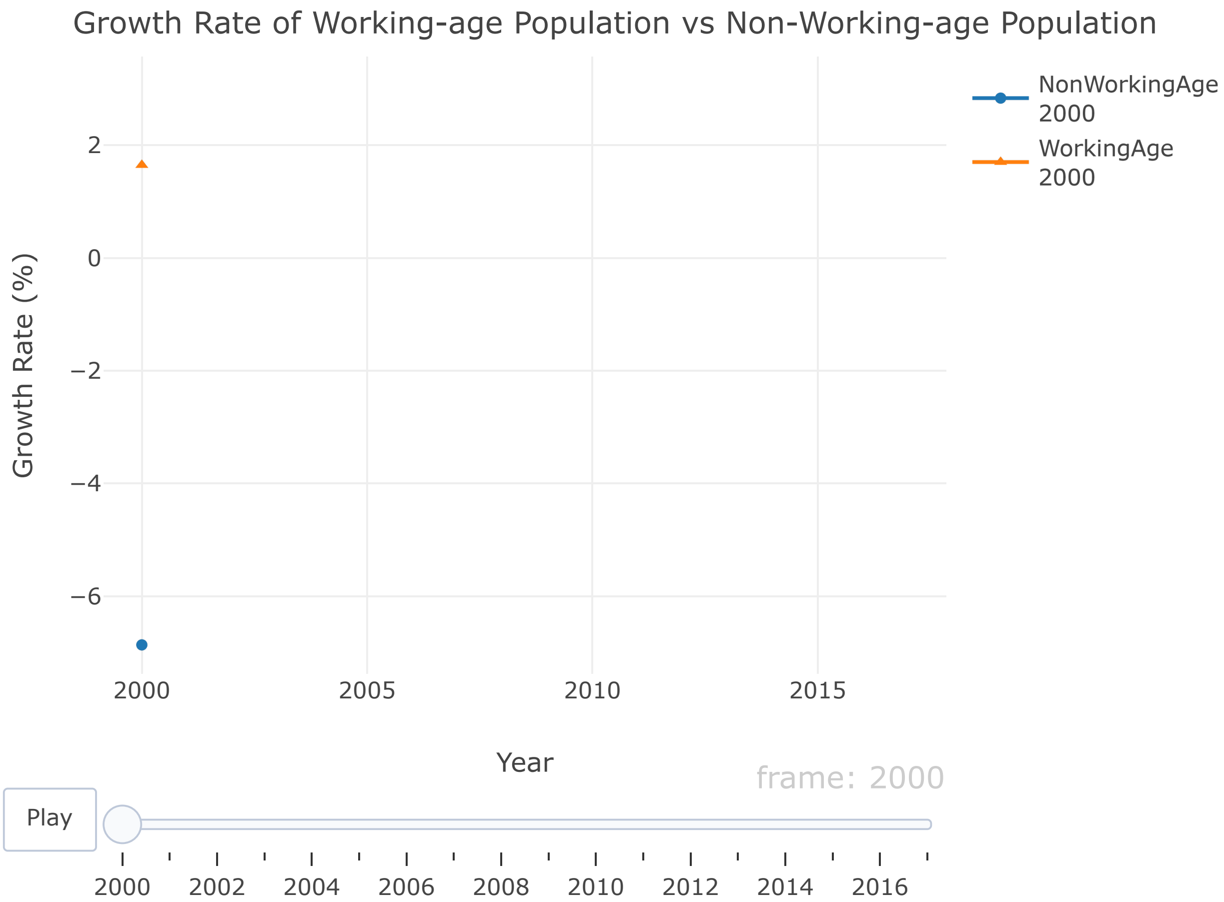
The frames key points to a list of figures, each of which will be cycled through upon instantiation of the plot. In order to create two time series line graphs on the same plot, we will first build a variable to be used in the frame parameter by year.

```
# source: https://plot.ly/r/cumulative-animations/

# Step 1: function definition for cumulative animation
accumulate_by <- function(dat, var) {
  # delaying evaluation for thr specified variable in the dataset
  var <- lazyeval::f_eval(var, dat)
  lvls <- plotly:::getLevels(var)
  dats <- lapply(seq_along(lvls), function(x) {
    cbind(dat[var %in% lvls[seq(1, x)], ], frame = lvls[[x]])
  })
  dplyr::bind_rows(dats)
}
```

```
# Step 2: creation of to-be-used for framing variable
gr <- eap %>%
  accumulate_by(~Year)
```

The data is recorded on a yearly basis, so the year is assigned to frame, and each point in the plot represents one population (WorkingAge/Non-Working-Age), so the type is assigned to “split”, ensuring a smooth transition from year to year for a given group of population.

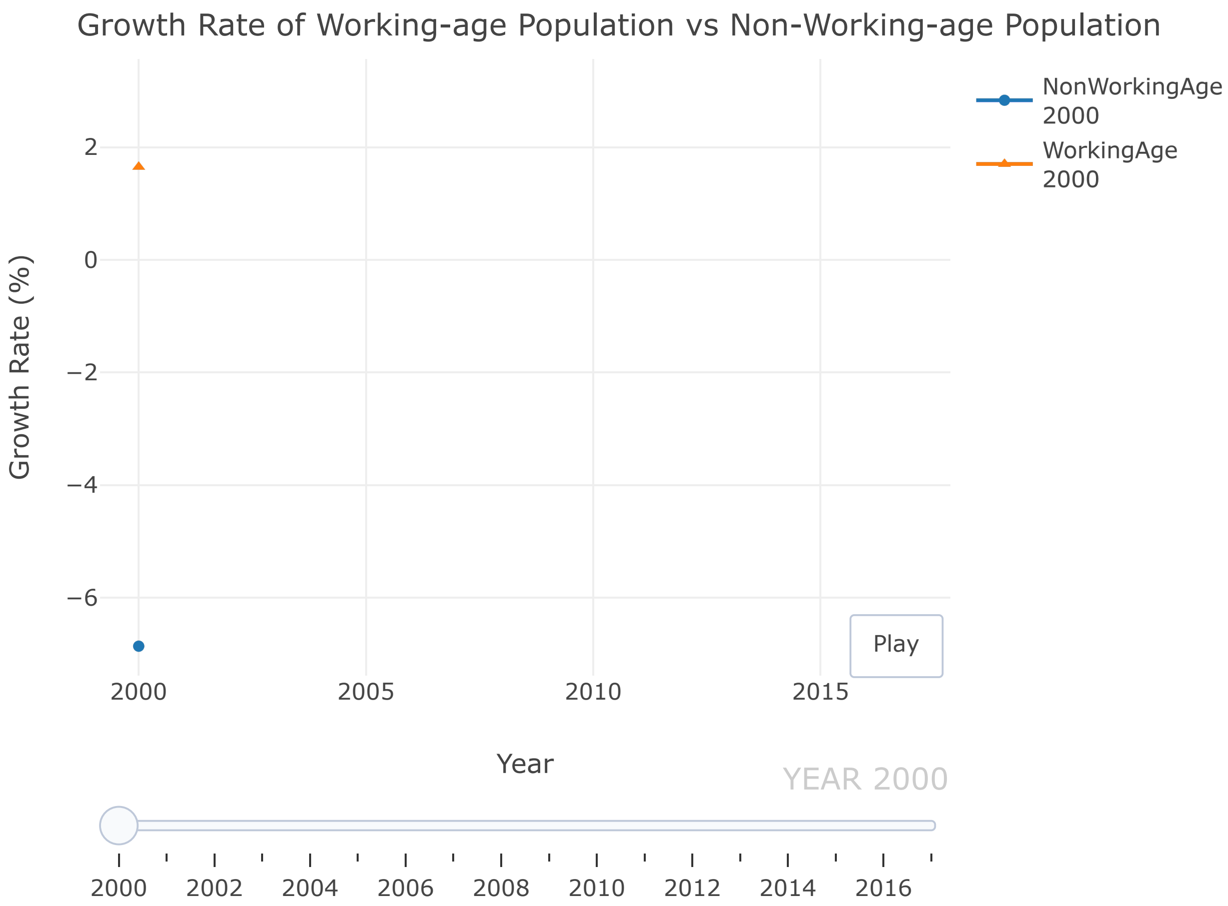
```
# Step 3 : graphing cumulative animation
base <- gr %>%
  plot_ly(
    x = ~Year,
    y = ~GrowthRate,
    split = ~Type,
    frame = ~frame,
    type = 'scatter',
    mode = 'lines + markers',
    symbol = ~Type,
    line = list(simplyfy = F),
    hoverinfo = text, text = ~paste('Year: ', Year, '</br> GrowthRate: ', GrowthRate)
  ) %>%
  layout(
    title = "Growth Rate of Working-age Population vs Non-Working-age Population",
    titlefont = list(size = 16),
    xaxis = list(
      title = "Year",
      zeroline = F
    ),
    yaxis = list(
      title = "Growth Rate (%)",
      zeroline = F
    )
  )

base
```



As shown in the figure above, the proportion of non-working-age population has been increasing faster than that of the working population since 2011, which indicated that the two-child policy is already having an effect on Chinese population.

As long as a frame variable is provided, an animation is produced with a play button and a slider component for controlling the animation. These components can be removed or customized via the animation_button() and animation_slider() functions. Moreover, various animation options, like the amount of time between frames, the smooth transition duration, and the type of transition easing may be altered via the animation_opts() function. The figure below shows the same data, but doubles the amount of time between frames, uses linear transition easing, places the animation buttons closer to the slider, and modifies the default currentvalue.prefix settings for the slider.

```
base %>%
  animation_opts(1000, redraw = FALSE) %>%
  animation_button(
    x = 1, xanchor = "right", y = 0, yanchor = "bottom"
  ) %>%
  animation_slider(
    currentvalue = list(prefix = "YEAR ")
  )
```



Dataset 3: GDP

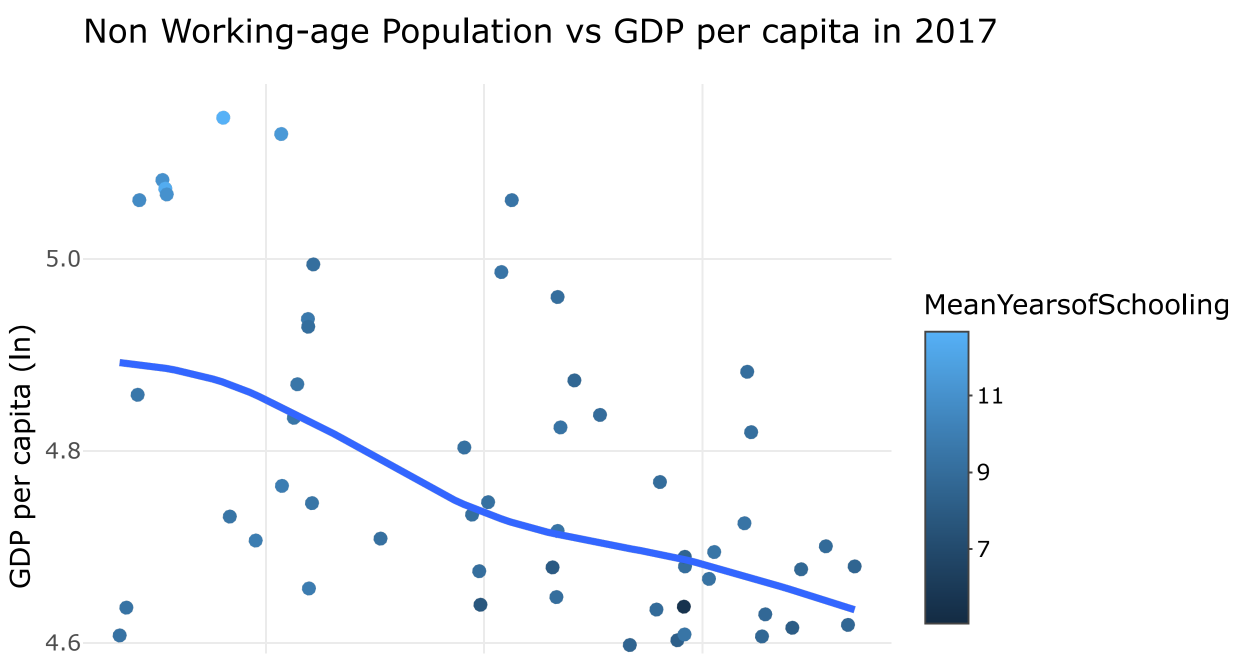
In addition to directly initializing a plotly object with plot_ly, the ggplotly() function from the plotly package has the ability to translate ggplot2 to plotly. This functionality can be really helpful for quickly adding interactivity to the existing ggplot2 workflow. To demonstrate, let's explore the relationship between GDP per capita and other variables from the "gdp" dataset.

```
# Loading the dataet
gdp <- read_csv("gdp.csv")
head(gdp)
```

```
## # A tibble: 6 x 4
##   Region          lnGDP nonworkingprop MeanYearsofSchooling
##   <chr>           <dbl>          <dbl>                <dbl>
## 1 Beijing          5.15           23.4                 12.7
## 2 Tianjin          5.08           22.6                 11.0
## 3 Hebei            4.68           29.8                  9.09
## 4 Shanxi           4.66           24.6                  9.86
## 5 Inner Mongolia  4.83           24.4                  9.52
## 6 Liaoning         4.76           24.2                  9.93
```

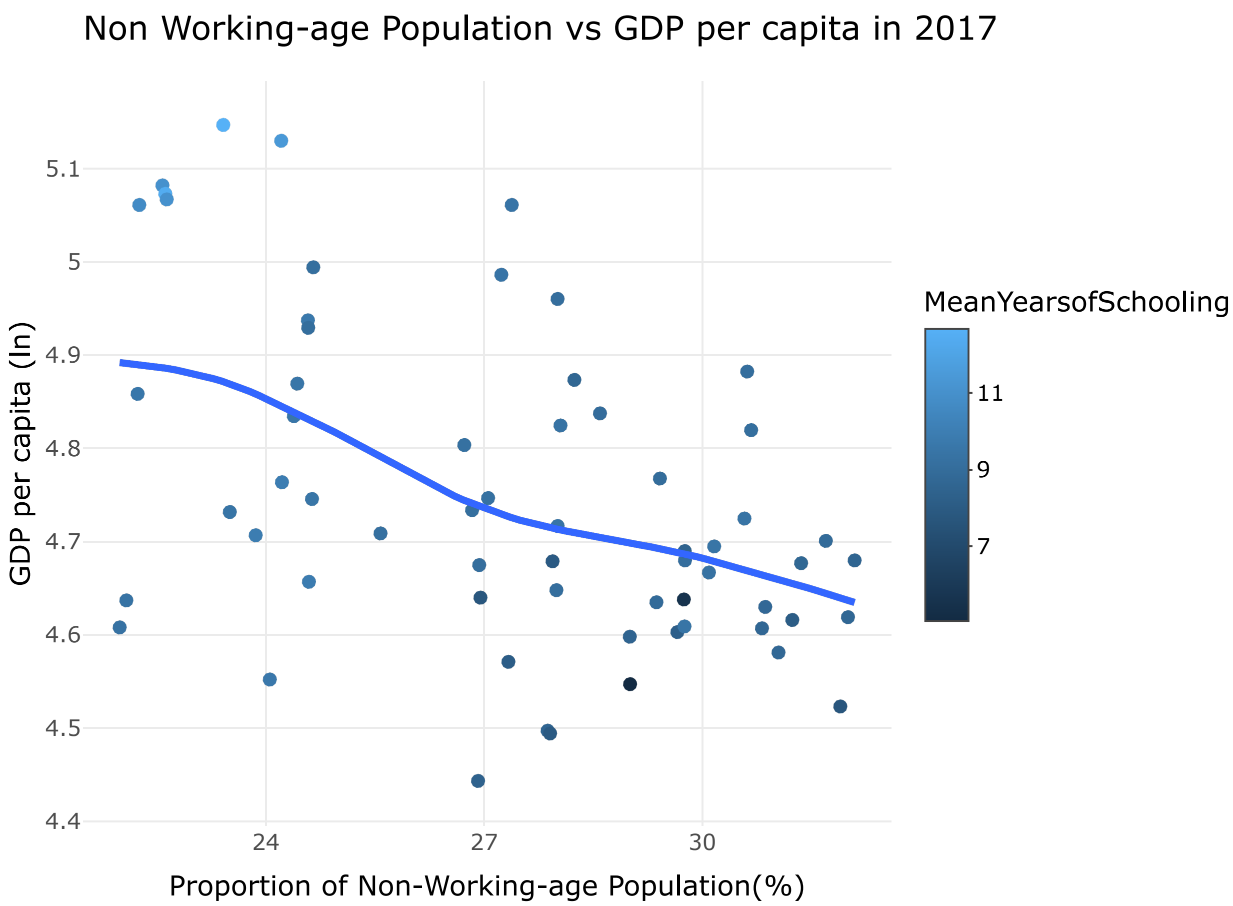
```
# Step 1: create a ggplot object
p1 <- ggplot(data = gdp, aes(x = nonworkingprop, y = lnGDP)) +
  geom_point(aes(colour = MeanYearsofSchooling)) +
  scale_size(range = c(.01, 9), name = "Mean Years of Schooling") +
  geom_smooth(se = F, lwd = 1) +
  theme_minimal() +
  labs(y = "GDP per capita (ln)",
       x = "Proportion of Non-Working-age Population(%)",
       title = "Non Working-age Population vs GDP per capita in 2017")
```

```
# Step 2: transform ggplot object into interactive visualization using ggplotly
ggplotly(p1)
```



By default, ggplotly() tries to replicate the static ggplot2 version exactly (before any interaction occurs), but sometimes we need greater control over the interactive behavior. The ggplotly() function itself has some convenient “high-level” arguments, such as dynamicTicks, which tells plotly.js to dynamically recompute axes, when appropriate. The style() function also comes in handy for modifying the underlying trace attributes (e.g. hoveron) used to generate the plot:

```
# Step 3: Modify axies
gg <- ggplotly(p1, dynamicTicks = "y")
style(gg, hoveron = "points", hoverinfo = "x+y+text", hoverlabel = list(bgcolor = "whit
e"))
```



From the figure above, we can see there is a negative relationship between proportion of Non-working-age poplulation and log of GDP per capita. It also shows that, provinces with higher level of educational attainment performs relatively high labor productivity. Making this plot interactive makes it easier to decode the shapes of the color into mean years of schooling that they represent.

In this introduction, we only touched relatively simple techniques and features of the plotly R library, but the true power of interactive graphics in the official documents and plotly graph libraries will be explored.

Works Cited

Carson Sievert, Interactive web-based data visualization with R, plotly, and shiny, https://plotly-r.com/introduction.html (https://plotly-r.com/introduction.html)

Plotly R Open Source Graphing Library, https://plot.ly/r/ (https://plot.ly/r/)

rOpenSci project/plotly, https://github.com/ropensci/plotly#readme (https://github.com/ropensci/plotly#readme)