Oct 2024

R Shiny Masterclass Series Introduction

Interactive charts with Plotly

EPI-interactive

Agenda

- **Session 1** | 30 September | Getting started with Posit Cloud and your first R Shiny app
- **Session 2** | 01 October | R Shiny core concepts and mobile ready layout
- **Session 3** | 03 October | R Shiny user interface components, reactivity and debugging
- **Session 4** | 07 October | Data sources and data processing in R Shiny
- **Session 5** | 08 October | Maps and spatial visualisation with Leaflet: adding map layers, annotations, pins, filters and legend
- **Session 6** | 10 October | Interactive charts with Plotly: chart types, customising hover boxes and chart styling
- **Session 7** | 14 October | Publishing R Shiny apps, design considerations and case study
- **Session 8** | 15 October | Case study, top 10 tips for data visualisation with R Shiny and wrap-up

Today

Recap: Session 5 challenge

Goals:

- Plot graphs in an interactive context

Steps:

- Graph data using Plotly
- Adding custom features to the graph

Exporting data

Getting data back out of the application

Some common functions:

- For CSV files: write.csv(data, “filepath”)
- For RDS files: saveRDS(data, “filepath”)

To link this up to R Shiny:

- downloadLink(outputId = “dl”, label = “download”)
  [*in ui.R*]
- downloadHandler(filename = function() {}, content = function() {}) [in server.R]

Exporting data

In your /stage2, add:

To *ui.R*: downloadLink(outputId = “download”, label = “Download”)

To *server.R*:

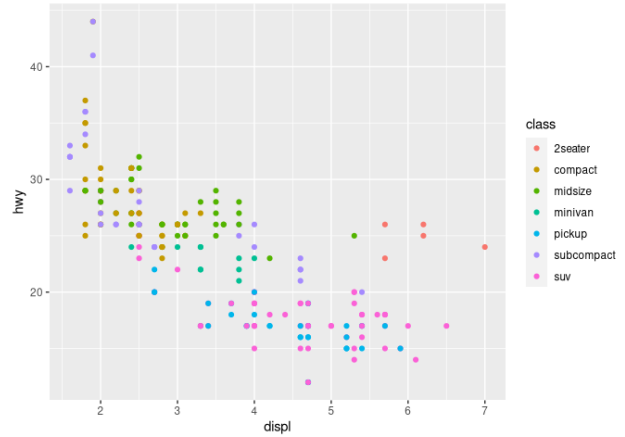
```
output$download <- downloadHandler(
        filename <- function() {
                ...
        },
        content = function(file) {
                write.csv(data, file)
        })
```

Getting started with Plotly

ggplot2 vs Plotly

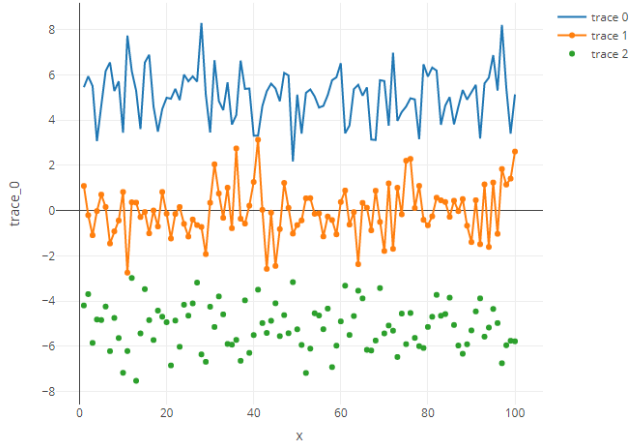
ggplot2

- Static
- PNG output
- Example: https://www.r-graph-gallery.com/line-chart-ggplot2.html

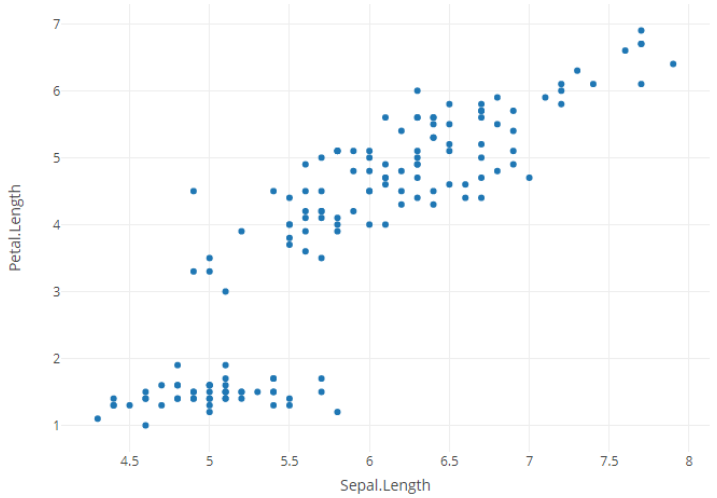


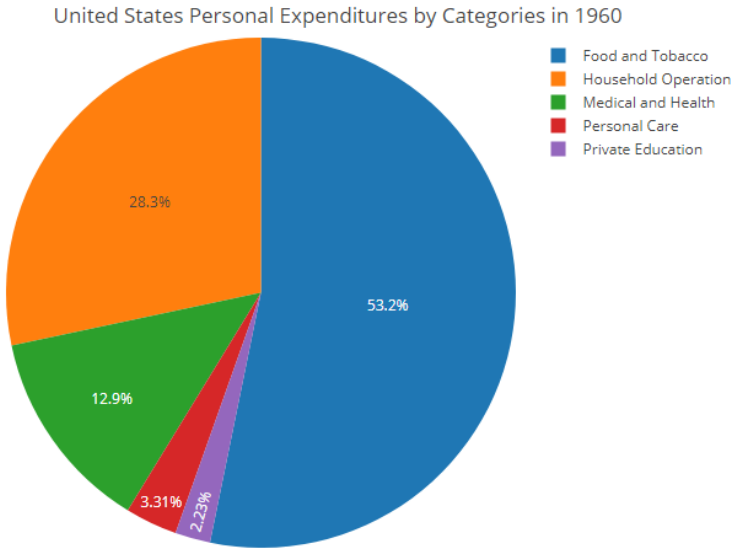
Plotly

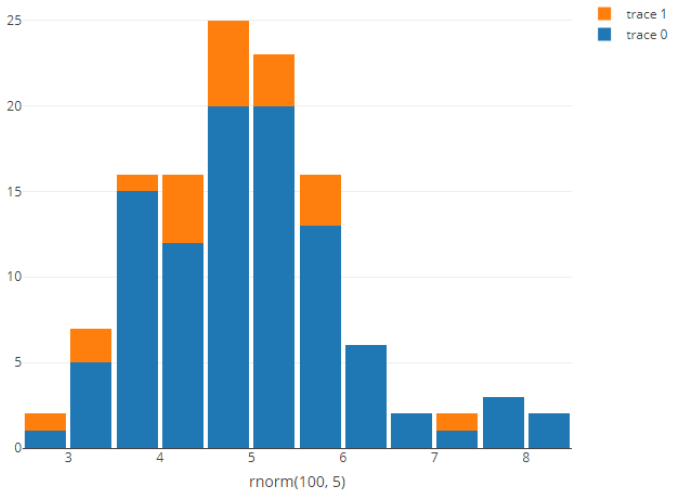
- Interactive
- SVG output
- Example: https://plotly.com/r/line-charts/
- Interaction of ggplot2: https://plotly.com/ggplot2/

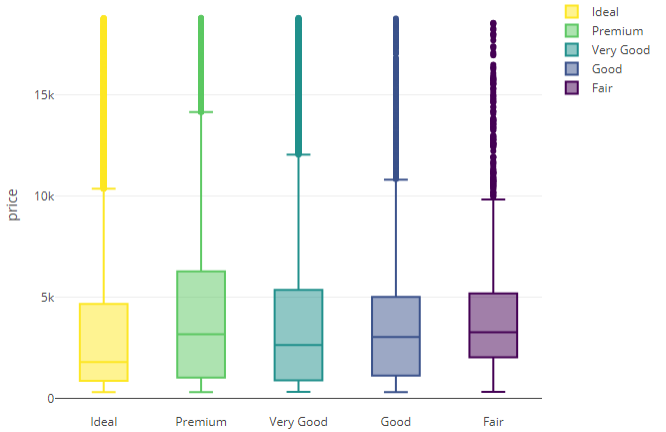


EPI-interactive









EPI-interactive

File Structure

- UI:
  ```
  plotlyOutput(“plot_id”)
  ```

- Server:
  ```
  output$plot_id <- renderPlotly({
        plot_ly(…)
  })
  ```

Common server functions

- **plot_ly():** Base of the graph; establishes type of graph, data, height, width. Can also be left blank if need be.
- **add_trace():** Individual traces, allows for easy customization between different traces – eg colour, hovers, type of trace
- **layout():** The look of the overall graph; sets axis, margins, title, legend etc
- **config():** Allows for configuration of the mode bar buttons and language

Common server functions: Example

```
plot_ly() %>%
add_trace(x = data$x,
          y = data$y) %>%
layout(yaxis= yaxis,
       xaxis= xaxis,
       showlegend = FALSE ) %>%
config(displayModeBar = F)
```

Note: plot_ly function must come first, the order of the other functions does not matter

Pipe (%>%) operation in Plotly

- Cleaner syntax
- Chaining functions instead of nesting

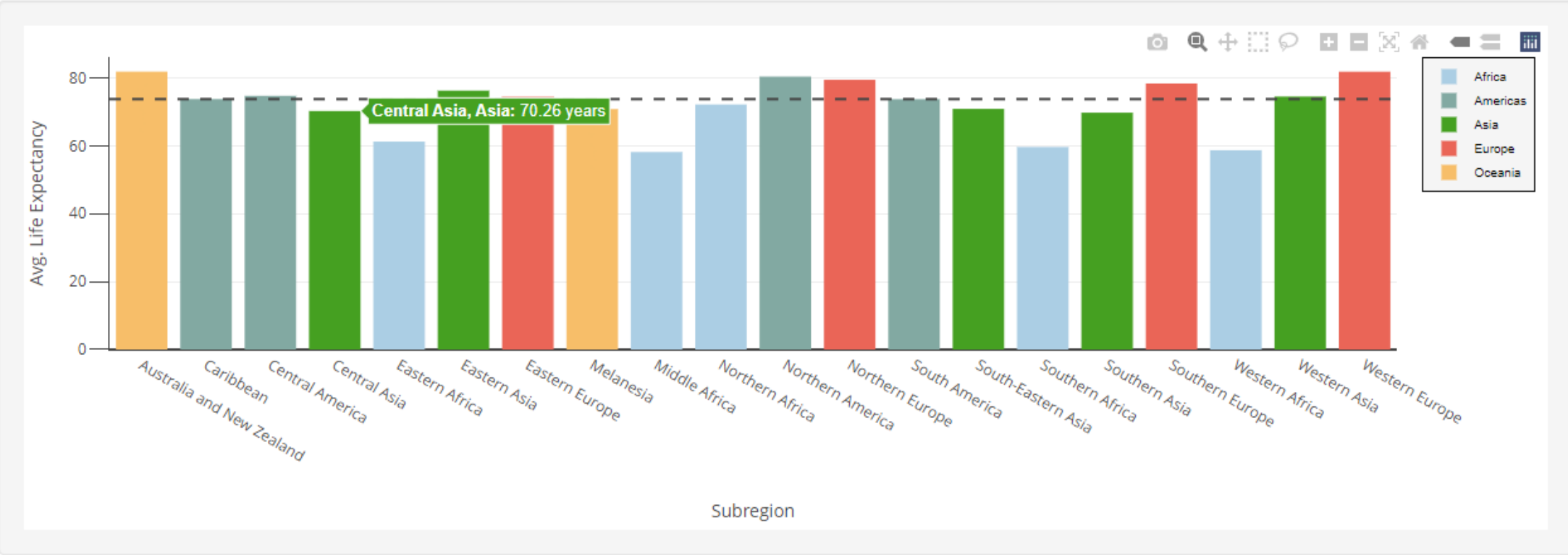
```
x %>%
  func1(y) %>%
  func2 %>%
  func3
```

vs

```
func3(
  func2(
    func1(x, y)
  )
)
```

- Cannot be at the start of a new line
  -
```
x
%>% func1(y)
%>% func2
%>% func3
```
  - This will **not** work

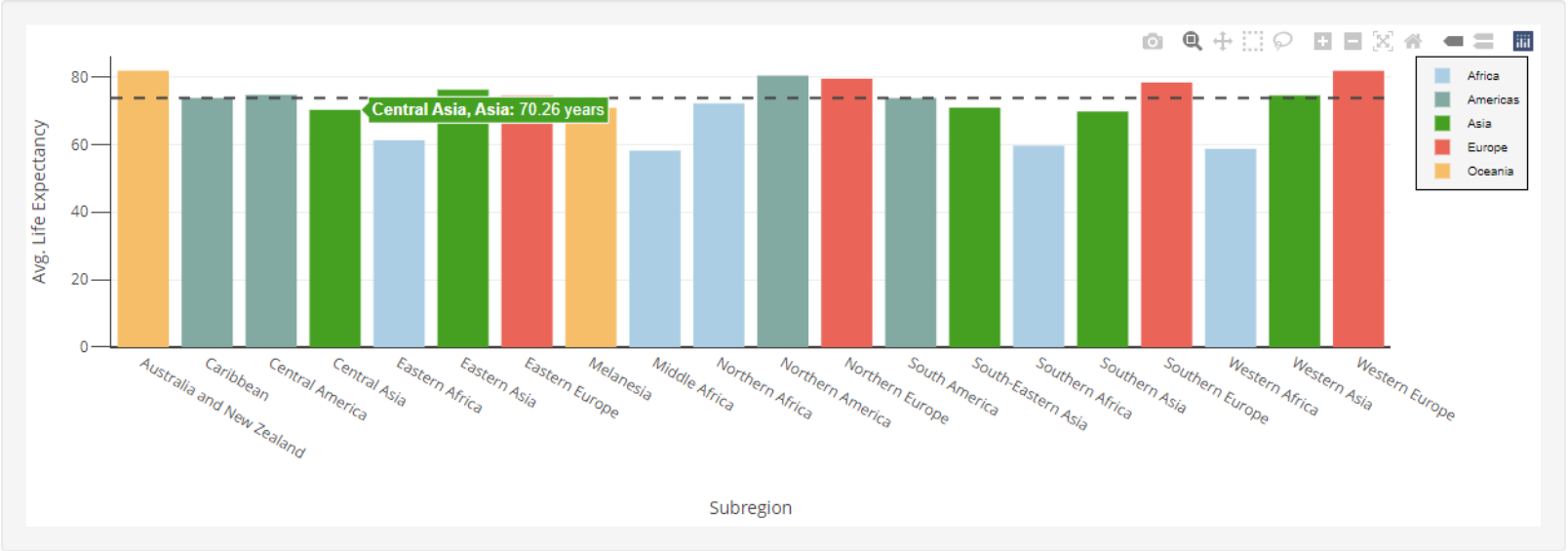
The mission



EPI-interactive

The mission

1. Basic plot
2. Multiple Layers
3. Hovers
4. Shapes
5. Axes



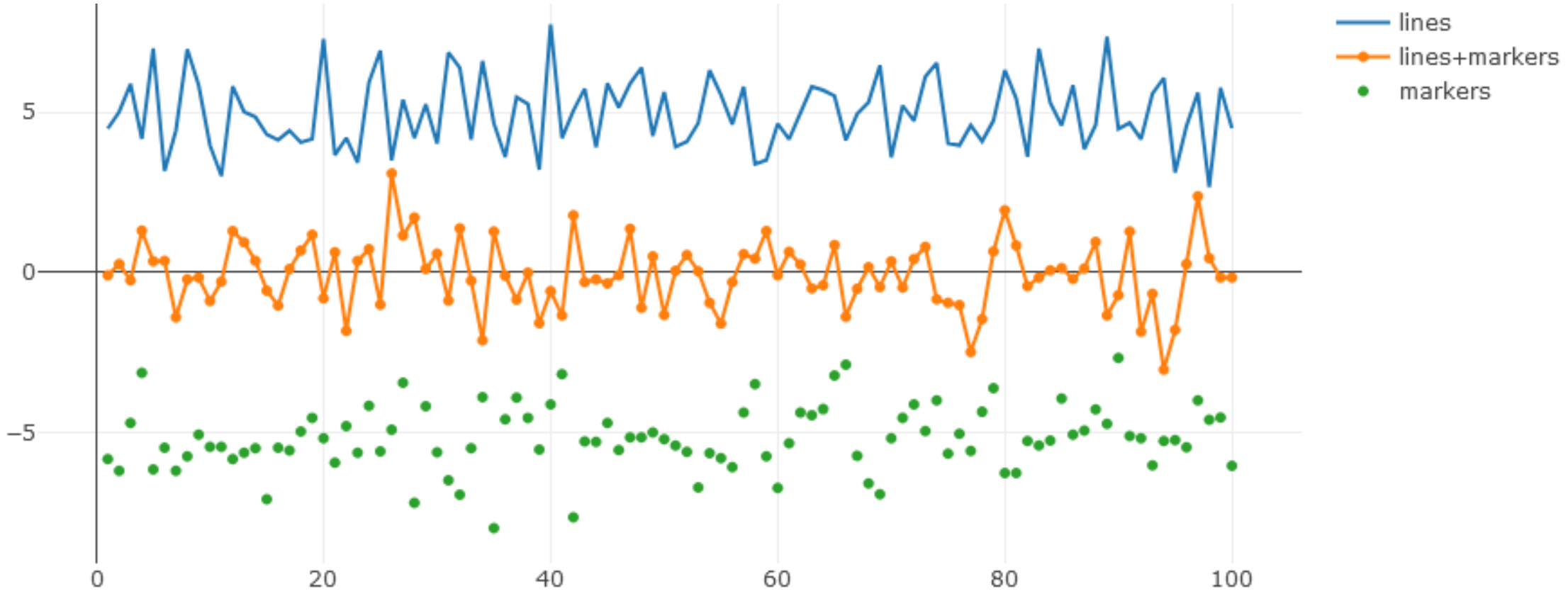
1. Basic plot

Basic plot

- Contains one ‘trace’ of data
- ‘**data**’: which data frame will be used in the plot
- ‘**type**’: what type of plot will be created
  - E.g., scatter, bar, pie
- ‘**mode**’: variations on the type of a plot
  - E.g. lines / markers / lines+markers

```
plot_ly(
    data = world_data,
    y = ~avg_lifeExp,
    x = ~subregion,
    type = “bar"
)
```

Basic plot – scatter plot modes



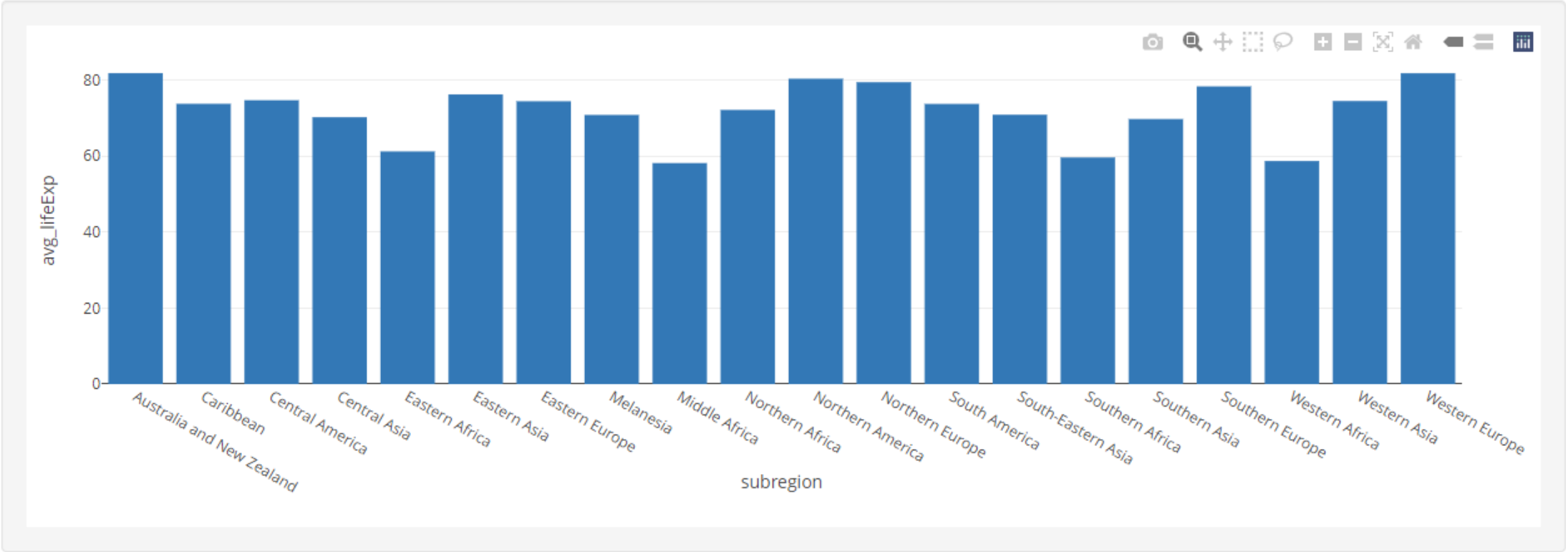
Basic plot – tilde (~)

- The tilde (~) is used to indicate that the variable names are to be interpreted as column names from the data frame specified in data
- The two statements below will produce the same plot

```
plot_ly(
    data = world_data,
    y = ~avg_lifeExp,
    x = ~subregion,
    type = “bar"
)
```

```
plot_ly(
    y = world_data$avg_lifeExp,
    x = world_data$subregion,
    type = “bar"
)
```

Basic plot


avg_lifeExp
0
20
40
60
80
Australia and New Zealand
Caribbean
Central America
Central Asia
Eastern Africa
Eastern Asia
Eastern Europe
Melanesia
Middle Africa
Northern Africa
Northern America
Northern Europe
South America
South-Eastern Asia
Southern Africa
Southern Asia
Southern Europe
Western Africa
Western Asia
Western Europe
subregion

Basic plot - Excercise

ui.R

```
plotlyOutput("world_data_chart")
```

server.R

```
output$world_data_chart <- renderPlotly({
        plot_ly(
            data = world_data_mod(),
            y = ~avg_lifeExp,
            x = ~subregion,
            type = “bar"
          )
})
```

In *stage1/server.R* add parameters to the `plot_ly` function to create the basic plot.

2. Multiple data layers

Multiple data layers

- We may want to show more than one sub-category of data
  - E.g. continent, region, type
- How do we distinguish this in Plotly? Two options
  - ***Use colours to categorise data***
  - Use multiple traces

```
plot_ly(
    data = world_data_mod(),
    y = ~avg_lifeExp,
    x = ~subregion,
    type = “bar",
    colors = c(…),
    color = ~region_un
)

Or…

plot_ly(…) %>%
add_trace(
    data = cat1Data,
    …
)
```

Getting a colour palette

- We can either do this manually

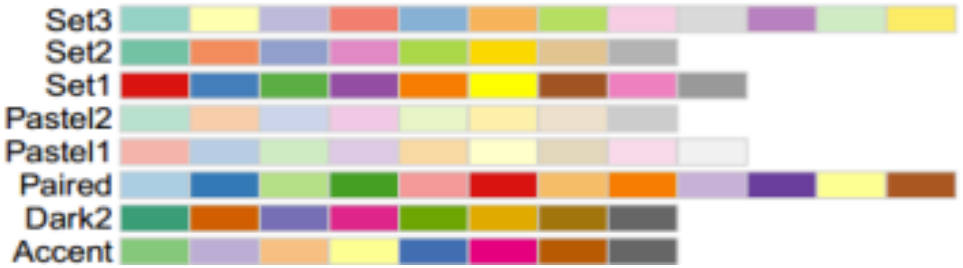
```
chart_colours <- c("red", "green", "blue", "yellow", "purple")
```

- Use grDevices::palette.colors()

```
chart_colours <- grDevices::palette.colors("Dark 2")
```

- Use RColorBrewer::brewer.pal()

```
chart_colours <- RColorBrewer::brewer.pal(n = ?, name = "Paired")
```



Legend functionality

- Positioning in comparison to the graph
- Orientation
- Order
- Styling
- Visibility

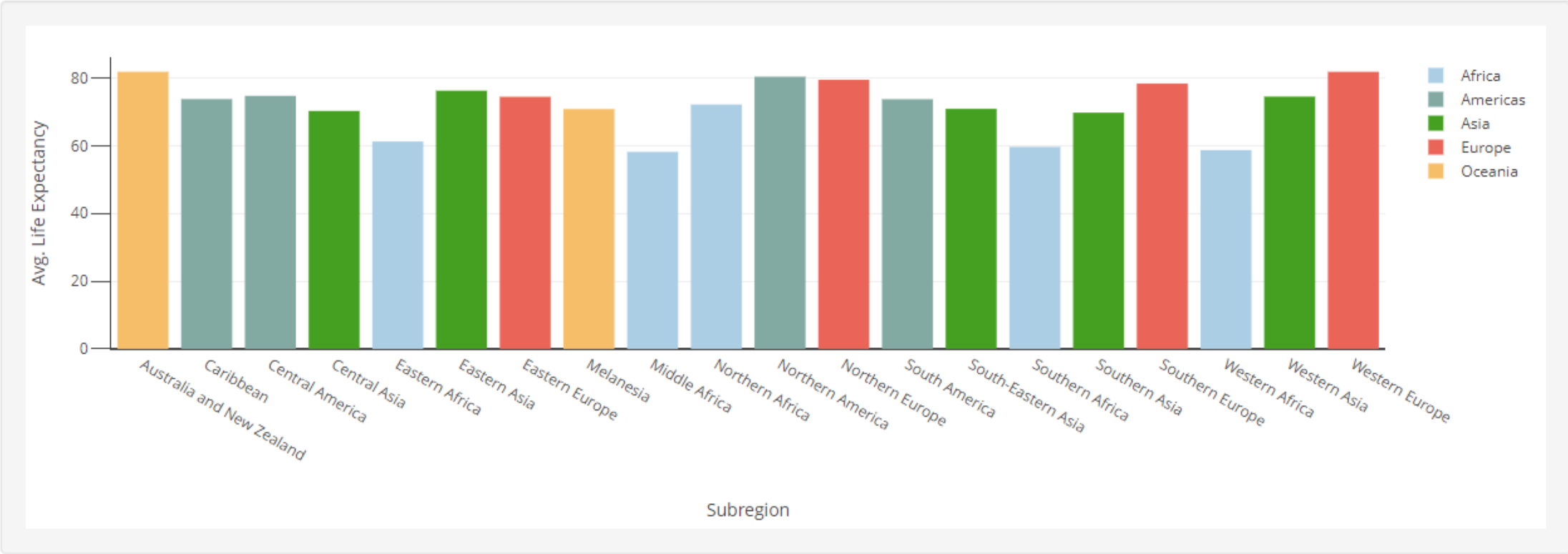
Documentation of these can be found:
https://plotly.com/r/reference/layout/#layout-legend

Legend customisation

```
l <- list(
    x = 0.5,
    y = 0,
    orientation = "h",
    font = list(
        family = "sans-serif",
        size = 12,
        color = "#000"),
    bgcolor = "#F5F5F5",
    bordercolor = "#000",
    borderwidth = 2
)
```

```
# apply legend
plot_ly(
    ...
) %>% layout (...,legend = l)
```

Multiple data layers

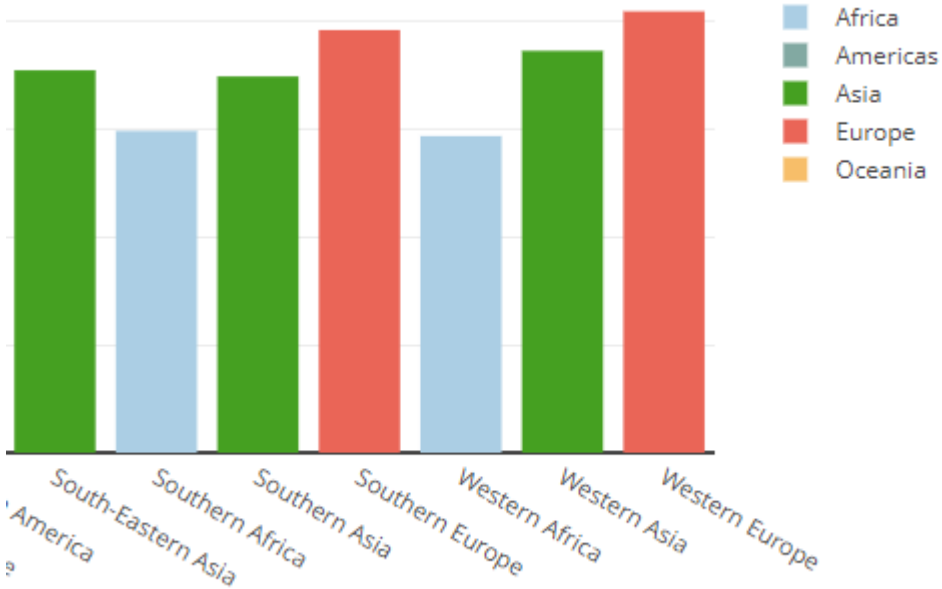


To do:

In *stage2/server.R* or continuing from that last exercise:

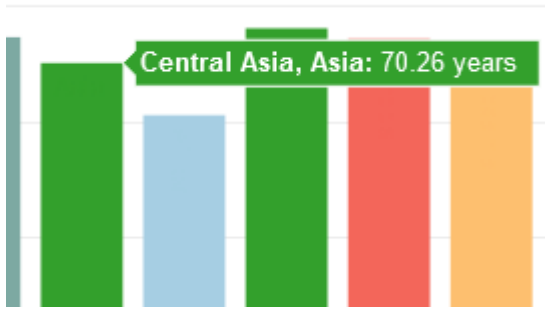
- Choose a variable to use as your category
- Set this variable as the ‘color’ argument
- Create a vector for your colours and assign this to the ‘colors’ argument
  - Hint: if using RColorBrewer to create the colour palette, think about how many colours should be created?

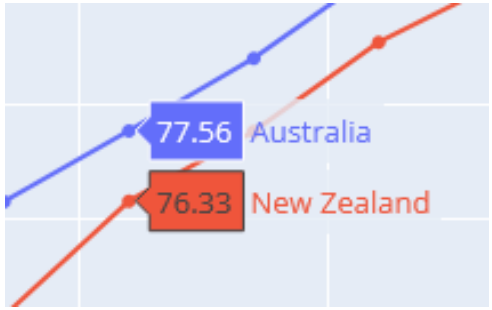
- (Optional) put a border on the legend



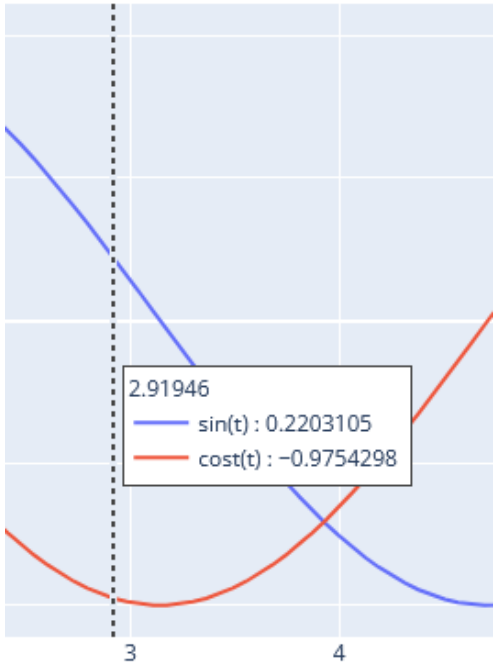
3. Hovers

Hover functionality





- Add custom text
- Format text
- Optional: use existing plotly attributes
- Display hovers: hoverinfo or hovertemplate



Documentation of these can be found: https://plotly.com/r/reference/scatter/#scatter-hovertext

Hover equivalents

Hoverinfo

```
text = sprintf('%s, %s: %s years', dat$subregion,
dat$continent, round(data$avg_lifeExp, 2))
hoverinfo = 'text'
```

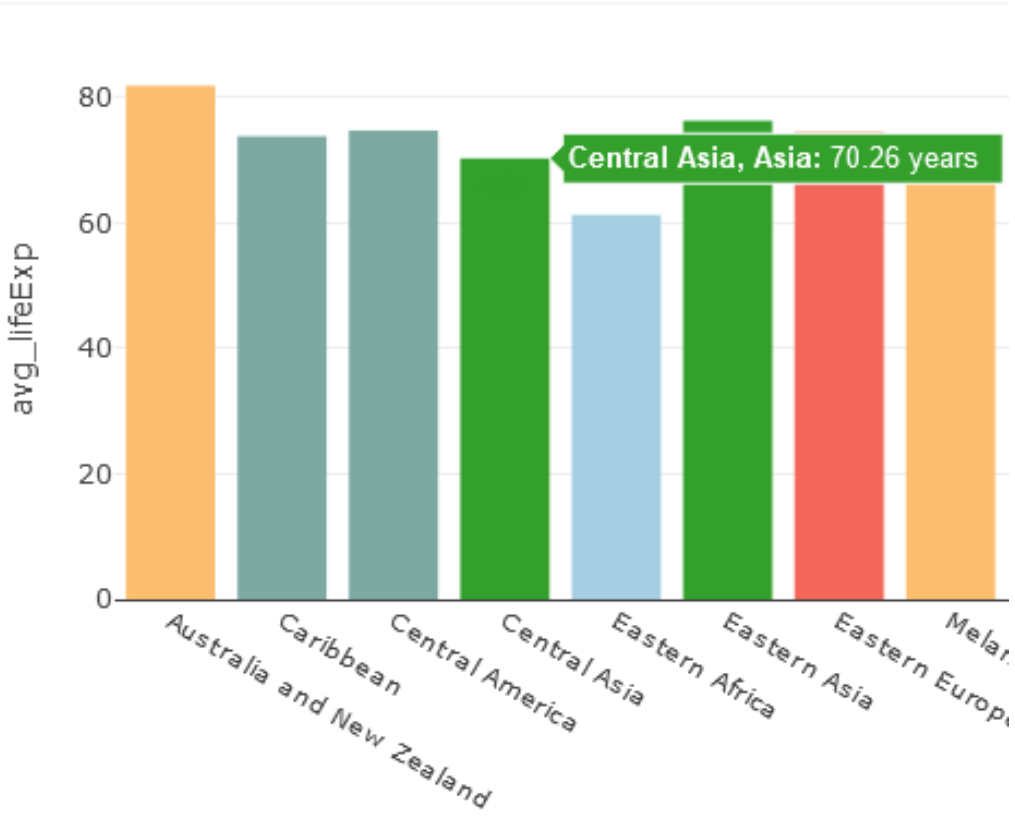
Hovertemplate

```
text = ~continent,
hovertemplate = '<b>%{x}, %{text}:</b> %{y:.2f}
years<extra></extra>'
```

To do:

In *stage3/server.R* or continuing from that last exercise:

- Add either a hovertemplate or hovertext to your line chart
- In this hover text, show the X and Y values of the hovered point on the chart
- Add formatting using HTML tags



4. Shapes

Shapes

- Use shapes to provide other visuals alongside data
  - E.g. median ***line***, highlight region with ***rectangles*** or ***circles***
- Create a list with named attributes
  - **xref**, x0, x1
  - **yref**, y0, y1
  - fillcolor, **line**, opacity

```
Line <- list(
    type = “line”,
    xref = “paper”, yref = “y”
    x0 = 0, x1 = 1,
    y0 = 0, y1 = 1
)
Rect <- list(
    type = “rect”,
    xref = “x”, yref = “y”,
    …
)
plot_ly(…) %>%
    layout(
        shapes = list(
            Line,
            Rect
        )
)
```

Shapes

- Some attributes may require more tuning

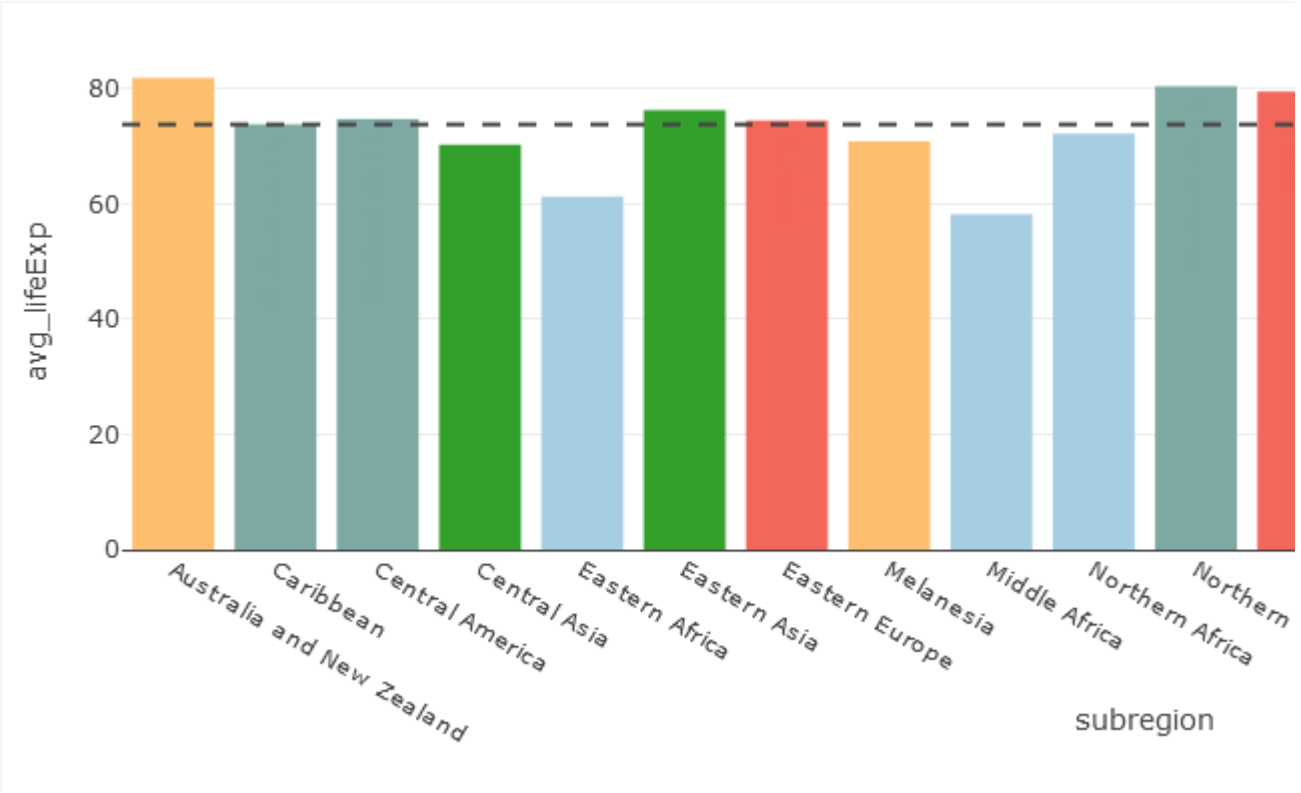
- ***xref, yref*** – what units are used for positioning the shape
  - Container – position from edge to edge of plotly container (0-1)
  - Paper – position from edge to edge of plotly inner chart (0-1)
  - x / y – position in relation to an x / y axis value (variable)

- ***Line*** – list of named attributes related to the outline of a shape
  - color – the color of the line
  - dash – the style of the line e.g., ‘solid’, ‘dot’, ‘dash’
  - width – the width of the line in pixels

To do:

In *stage4/server.R* or continuing from that last exercise:

- Create a variable for the median or mean of your y axis data
- Create a line shape, using this variable to set the positional attributes
- Add this shape to your Plotly chart



5. Axes

Common attributes

- Fonts: Size, family, colour
- Titles: Suffix, prefix
- Ticks: colour, type/format (date vs number vs string), ranges, angle, intervals
- Grid: colour, thickness, show lines or not
- Zeroline
- Visibility

Documentation of these can be found: https://plotly.com/r/reference/layout/yaxis/ or https://plotly.com/r/reference/layout/xaxis/

server.R

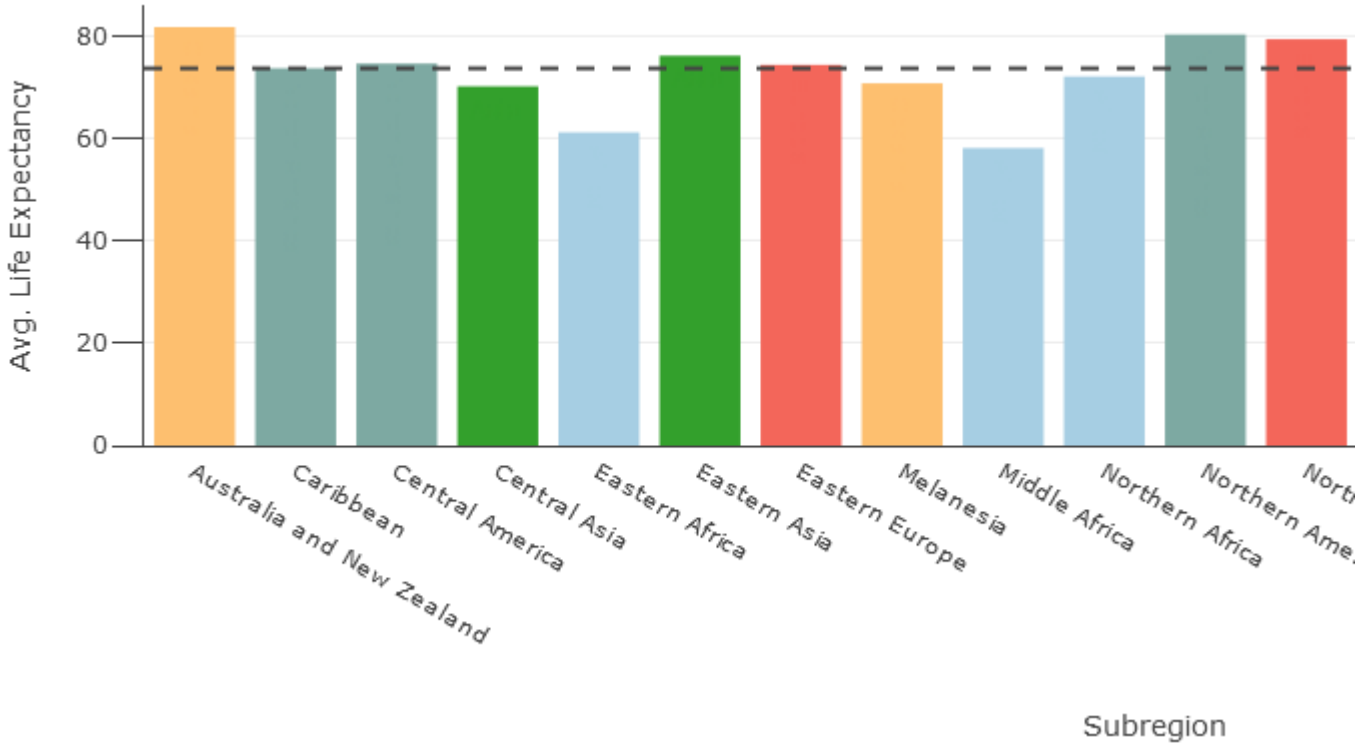
```
y <- list(…)
x <- list(
    …,
    title = list(text = “title“, …)
)

plot_ly(…) %>%
     layout(xaxis = x, yaxis = y)
```

To do

In *stage5/server.R* or continuing from that last exercise:

- Add appropriate titles to the x / y axes
  - Increase title standoff from ticks (I.e. distance between title and tick text)
- Show tick lines on the Y axis
- Hide the X axis grid lines
- Show the X / Y axis lines
- Apply layout to chart



Hint: *CTRL + F* to search through the attributes in the documentation to find how to use them.

Completed!

Next time

- Design considerations
- Publishing Shiny apps
- Practise / recap of concepts

Challenge:

- Add another trace to your Plotly chart to show a different summary variable (E.g. avg. GDP, total population, total area) on a second Y axis
- Make this secondary trace show as a line chart (use the ‘type’ and ‘mode’ arguments to change this)
- Apply appropriate labels, hovers and styling to this new axis and data
- Share your work on the session 5 forum