Addition

```
1 + 1
```

Multiplication

```
10 * 10
```

helloworld

```
h<-"Helloworld" print(h)
```

Objects

```
a<-6 a
```

Print nos from 1:6

```
die<-1:6 die
```

mean

```
mean(1:6)
```

round

```
round(mean(1:6))
```

sample function sample takes two arguments:

a vector named x and a number named size.

sample will return size elements from the vector:

```
sample(x = 1:4, size =3)
```

```
dice<-sample(die, size = 2) sum(dice)
```

```
x <- 2 #assigns the value '2' to the variable x
```

```
y = 3 #assigns the value '3' to the variable y
```

uses the sum() function to add variables x and y,

saving the results into variable sumofxy

```
sumofxy <- sum(x, y)
```

displays value of sumofxy

```
sumofxy
```

help on a function

```
?sum
```

List objects in current working space

ls()

rm(x,y)

sqrt(16)

seq(1,9,by=2)

generate the sequence

seq(8,20,length=6)

rep(1:3,6)

vector

d<- c(-1.0,-0.7,0.5,1.5,2) d length(d)

modules

x <- (17 %% 13) x

integer division

x<- 17%/%13 x

multiply a matrix with its transpose.

```
M = matrix( c(2,6,5,1,10,4), nrow = 2,ncol = 3,byrow = TRUE) print(t(M)) t = M %*% t(M) print(t)
```

looping repeat statement

```
v <- c("Hello","loop") cnt <- 2

repeat { print(cnt) print(v) cnt <- cnt+1

if(cnt > 5) { break } }
```

while loop

```
v <- c("Hello","while loop") cnt <- 2

while (cnt < 7) { print(v) cnt = cnt + 1 }
```

for loop

```
v <- LETTERS[1:4] for ( i in v) { print(i) }
```

Create a function to print squares of numbers in sequence.

```
new.function <- function(a) { for(i in 1:a) { b <- i^2 print(b) } }
```

Call the function new.function supplying 6 as an argument.

```
new.function(6)
```

Get the max salary from data frame.

```
sal <- max(data$salary) print(sal)
```

Get the person detail having max salary.

```
retval <- subset(data, salary == max(salary)) print(retval)

retval <- subset( data, dept == "IT") print(retval)

info <- subset(data, salary > 600 & dept == "IT") print(info)

retval <- subset(data, as.Date(start_date) > as.Date("2014-01-01")) print(retval)

write.csv(retval,"D:/R Workshop/output.csv") newdata <- read.csv("D:/R Workshop/output.csv", nrows=2) print(newdata)

df=data.frame(X=5,id=5,name='Rayon',salary=12000,start_date= '2014-11-15',dept='IT') df

write.table(df, "D:/R Workshop/output.csv", append = TRUE,sep = ",", col.names = FALSE, row.names = FALSE, quote = FALSE)

install.packages("xlsx") library("xlsx") mydata <- read.xlsx("D:/R Workshop/mydata.xls", sheetIndex=1) mydata
```

Example-1

library(plotly) df <- read.csv("Dataset/2014_world_gdp_with_codes.csv") head(df)

fig <- plot_ly(df, type='choropleth', locations=df$CODE, z=df$GDP..BILLIONS., text=df$COUNTRY, colorscale="Blues")

fig

Example-2

library(plotly) install.packages(listviewer) library(listviewer)

df <- read.csv("Dataset/worldhappiness.csv") head(df) s <- schema() agg <- s$transforms$aggregate$attributes$aggregations$items$aggregation$func$values

l = list() for (i in 1:length(agg)) { ll = list(method = "restyle", args = list('transforms[0].aggregations[0].func', agg[i]), label = agg[i]) l[[i]] = ll }

fig <- df %>% plot_ly(type = 'choropleth', locationmode = 'country names', locations = ~Country, z = ~HappinessScore, autocolorscale = F, reversescale = T, colorscale = 'Portland', transforms = list(list(type = 'aggregate', groups = ~Country, aggregations = list(list(target = 'z', func = 'sum', enabled = T))))) fig <- fig %>% layout(title = "**World Happiness**", geo = list(showframe = F, showcoastlines = F), updatemenus = list(list(x = 0.25, y = 1.04, xref = 'paper', yref = 'paper', yanchor = 'top', buttons = l)))

fig

Brief introduction to R Plotly package

The plotly package in R is an interface to open source Javascript charting library plotly.js

Plots can be created online in the plotly web GUI or offline using the plotly package

Interactive plots that can be easily shared online using the plotly API/account

Plotly plots can embed into R Markdown, R Shiny

Tool tip feature and other controls

ggplot2 graphs can be easily converted to interactive plotly objects using ggplotly() function

Package documentation

https://cran.r-project.org/web/packages/plotly/plotly.pdf

Install the 'plotly' package

#

install.packages("plotly")

install.packages("ggplot2")

load the 'plotly' package

library(plotly)

General Syntax plot_ly()

Plot_ly functions creates a plotly object and produces interactive visualization based on data and arguments supplied

plot_ly (data, x, y , type, mode, color, size,....)

?plot_ly()

data = a dataframe

x and y axis values

type = specifies the type of plot or trace such as 'histogram', 'scatter',

'bar', 'box', 'heatmap', 'histogram', 'histogram2d', 'histogram2dcontour', 'pie', 'contour', 'scatter3d',

'surface', 'mesh3d', 'scattergeo', 'choropleth', 'scattergl', 'scatterternary', 'scattermapbox', 'area'

mode = specifies the mode, such as 'line', 'points', 'markers'

color = specifies the color of data points usually a function of I() to avoid scaling

colors = colourbrewer pallete, vector of color in hexa format

size = name or expression that defines the size of the data points

We can use other functions such as layout() for axis formating

and other functions that comes with plotly package

"mtcars A data frame with 32 observations on 11 (numeric) variables. [, 1] mpg Miles/(US) gallon [, 2] cyl Number of cylinders [, 3] disp Displacement (cu.in.) [, 4] hp Gross horsepower [, 5] drat Rear axle ratio [, 6] wt Weight (1000 lbs) [, 7] qsec 1/4 mile time [, 8] vs Engine (0 = V-shaped, 1 = straight) [, 9] am Transmission (0 = automatic, 1 = manual) [,10] gear Number of forward gears "

str(mtcars)

Pipe operator %>% will forward a value, or the

result of an expression,

into the next function call/expression.

mtcars %>% filter(carb > 1) %>% group_by(cyl) %>% summarise(Avg_mpg = mean(mpg)) %>% arrange(desc(Avg_mpg))

Tilde operator is used to define the relationship between

dependent variable and independent variables in a statistical model

formula. The variable on the left-hand side of tilde operator

is the dependent variable and the variable(s)

on the right-hand side of

tilde operator is/are called the independent variable(s).

Simple Scatter Plot

Type of plot is not exclusively defined.

Plotly will guess the best type based on the data type.

x~y

x~.

z<-x~.

help("~")

y=x*2

y~x*2

p = plot_ly(data=mtcars, x=~wt, y=~mpg) p

Define the plot type exclusively as "scatter"

& mode as "markers"

p = plot_ly(data=mtcars, x=~wt, y=~mpg, type = "scatter", mode = "markers") p

Change color of scatter data points

p = plot_ly(data=mtcars, x=~wt, y=~mpg, type = "scatter", mode = "markers", color = I("black")) p

Change color of data points using marker argument

p = plot_ly(data=mtcars, x=~wt, y=~mpg, type = "scatter", mode = "markers", marker = list(color="green", size=10)) p

Styled Marker

p = plot_ly(data=mtcars, x=~wt, y=~mpg, type = "scatter", mode = "markers", marker = list(size = 7, color = 'rgba(255, 182, 193, .9)', line = list(color = 'rgba(152, 0, 0, .8)', width = 3))) p

str(mtcars)

Set the color scale based on a factor variable

p = plot_ly(data=mtcars, x=~wt, y=~mpg, type = "scatter", mode = "markers", color = ~as.factor(cyl))

p

Use color Brewer palletes for data point colors

p = plot_ly(data=mtcars, x=~wt, y=~mpg, type = "scatter", mode = "markers", color = ~as.factor(cyl), colors = "Set1")

p

Use customized pallete for data point colors

pal <- c("red", "blue", "orange") p = plot_ly(data=mtcars, x=~wt, y=~mpg, type = "scatter", mode = "markers", color = ~as.factor(cyl), colors = pal)

p

Change data point shape based on factor variable

p = plot_ly(data=mtcars, x=~wt, y=~mpg, type = "scatter", mode = "markers", symbol = ~as.factor(cyl), symbols = c('circle','x','o'), marker = list(size = 5))

p

Set the color scale based on a continous variable

p= plot_ly(data=mtcars, x=~wt, y=~mpg, mode="markers", color = ~gear) p

str(mtcars) p= plot_ly(data=mtcars, x=~wt, y=~mpg, mode="markers", color = ~as.factor(disp)) p

str(mtcars)

Scatter plot with diffent data point size based on a continous variable

p = plot_ly(data=mtcars, x=~wt, y=~mpg, type="scatter", mode="markers", color = ~as.factor(cyl), size = ~hp)

p

Hide the legend

p <- plot_ly(data=mtcars, x=~wt, y=~mpg, type = "scatter", mode = "markers", symbol = ~as.factor(cyl), symbols = c('circle','x','o'), marker = list(size = 5)) %>% layout(showlegend = TRUE)

p

Change the orientation of the legend (below the plot)

p <- plot_ly(data=mtcars, x=~wt, y=~mpg, type = "scatter", mode = "markers", symbol = ~as.factor(cyl), symbols = c('circle','x','o'), marker = list(size = 5)) %>% layout(legend = list(orientation = 'h'))

p

Change the orientation of the legend (inside the plot)

p <- plot_ly(data=mtcars, x=~wt, y=~mpg, type = "scatter", mode = "markers", symbol = ~as.factor(cyl), symbols = c('circle','x','o'), marker = list(size = 5)) %>% layout(legend = list(x = 0.8, y = 0.9))

p

Change the orientation of the legend (outside the plot)

p <- plot_ly(data=mtcars, x=~wt, y=~mpg, type = "scatter", mode = "markers", symbol = ~as.factor(cyl), symbols = c('circle','x','o'), marker = list(size = 5)) %>% layout(legend = list(x = 1, y = 0.5))

p

Add title to the legen

p <- plot_ly(data=mtcars, x=~wt, y=~mpg, type = "scatter", mode = "markers", symbol = ~as.factor(cyl), symbols = c('circle','x','o'), marker = list(size = 5)) %>% layout(legend = list(x = 1, y = 0.5))

p

Adding chart title and axis labels

p <- plot_ly(data=mtcars, x=~wt, y=~mpg, type = "scatter", mode = "markers", symbol = ~as.factor(cyl), symbols = c('circle','x','o'), marker = list(size = 5)) %>%

layout(title="Scatter plot using R Plotly", xaxis=list(title="Weight", showgrid = F), yaxis=list(title="MPG", showgrid = F))

p

Customizing mouse hover text

p = plot_ly(data = mtcars, x=~mpg, y=~wt, type = "scatter", mode="markers", hoverinfo = "text", text = paste("Miles per gallon: ", mtcars$mpg, "
", "Weight: ", mtcars$wt)) p

Add annotations to the plot - add_annotations() function syntax

add_annotations() function is used to add annotations to the plot along with the following arguments

x = set annotations x coordinate position (x axis value based on the dataset)

y = set annotations y coordinate position (y axis value based on the dataset)

text = text associated with annotation

showarrow = 0 or 1 depending whether to show arrow or not

xref = "paper" telling plotly to set x reference to the plot (in which case x=0 means the left most, x=1 means right most)

yref = "paper" telling plotly to set y reference to the plot (in which case y=0 means the bottom, y=1 means top)

xref = "x" (non paper x coordinates)

yref = "y (non paper y coordinates)

arrowhead, arrowsize, font, ax, ay

Example of annotations on a single data points

display annotations for good mileage

```
plot_ly(data = mtcars, x=~mpg, y=~wt, type = "scatter", mode="markers") %>% add_annotations( x=mtcars$mpg[which.max(mtcars$mpg)],
y=mtcars$wt[which.max(mtcars$mpg)], text="Good mileage", showarrow=T )
```

Example of annotations - placing text at a desired location on the plot

Display Data Source

Demo of x/y ref as "paper"

```
plot_ly(data = mtcars, x=~mpg, y=~wt, type = "scatter", mode="markers") %>% add_annotations( xref="paper", yref="paper", x=1, y=-0.2, text="Data Source :
mtcars",

)
```

Example #1 of annotations on multiple data points

display annotation for low and high mileage

```
plot_ly(data = mtcars, x=~mpg, y=~wt, type = "scatter", mode="markers") %>% add_annotations(x=mtcars$mpg[which.max(mtcars$mpg)],
y=mtcars$wt[which.max(mtcars$mpg)], text="high mileage" ) %>% add_annotations(x=mtcars$mpg[which.min(mtcars$mpg)], y=mtcars$wt[which.min(mtcars$mpg)],
text="low mileage" )
```

Example #2 of annotations on multiple data points

display annotations for automatic transission cars

```
plot_ly(data = mtcars, x=~mpg, y=~wt, type = "scatter", mode="markers") %>% add_annotations( x = mtcars$mpg[mtcars$am==0], y= mtcars$wt[mtcars$am==0],
text="auto", )
```

Styling annotations

using font argument

```
plot_ly(data = mtcars, x=~mpg, y=~wt, type = "scatter", mode="markers") %>% add_annotations(x=mtcars$mpg[which.max(mtcars$mpg)],
y=mtcars$wt[which.max(mtcars$mpg)], text="Good mileage", font = list(color = "green", family="sans serif", size = 20))
```

mtcars

stacked-bar

mtcars$gear <- factor(mtcars$gear) # Create a categorical variable mtcars$cyl <- factor(mtcars$cyl) # Create categorical variable

Creates stacked bar chart

fig <- plot_ly(data = mtcars, type = "bar",x=mtcars$gear, y=mtcars$cyl) fig <- fig %>% add_trace(data = mtcars, x=mtcars$gear, y=mtcars$cyl, type = "bar") fig <- fig %>% layout(yaxis = list(title = 'Count'), barmode = 'stack')

fig

mtcars

mtcars$gear <- as.numeric(mtcars$gear) # Create a num variable

mtcars$cyl <- as.numeric(mtcars$cyl)

mtcars$am <- as.numeric(mtcars$am) # Create a num variable

mtcars$cyl <- factor(mtcars$cyl)

str(mtcars)

Grouped Bar Chart

plot_ly(mtcars, x = ~cyl, y=~mpg,type="bar",name="mpg") %>% add_trace(y = ~hp, name="Horse Power") %>% layout(yaxis=list(title='count'),barmode='group')

Stack bar chart

plot_ly(mtcars, x = ~cyl, y=~mpg,type="bar",name="mpg") %>% add_trace(y = ~hp, name="Horse Power") %>% layout(yaxis=list(title='mpg and hp by cylinders'),barmode='stack')

Stack bar chart

plot_ly(mtcars, x = ~cyl, y=~mpg,type="bar",name="mpg") %>% add_trace(y = ~hp, name="Horse Power") %>% add_trace(y = ~carb, name="Carb") %>% layout(yaxis=list(title='mpg and hp by cylinders'),barmode='stack')

a <- aggregate(mpg ~ cyl+hp, data=mtcars, sum) a

Adding Custom labels that will show when you hover your mouse over to bar

a$cyl_label <- ifelse(a$cyl == 4,'4 Cylinder',ifelse(a$cyl==6,'6 Cylinder','8 Cylinder')) plot_ly(a, x = ~cyl,y = ~mpg,name = "Plotly Bar Chart Example", type = "bar", text = ~cyl_label)

Customizing bar color and Changing the x labels angle

plot_ly(mtcars, x = ~cyl, y=~mpg,type="bar",color = I("blue") ,name="mpg and hp by cylinders") %>% add_trace(y = ~hp, name="Horse Power", color = I("black")) %>% layout(xaxis = list(title='Cylinders',tickanle=-45),yaxis=list(title='count'),barmode='group')

subplots

fig1 <- plot_ly(data = mtcars, x=~mpg, y=~wt) fig1 <- fig1 %>% add_lines(name = ~"cyl") fig2 <- plot_ly(data = mtcars, x=~mpg, y=~wt) fig2 <- fig2 %>% add_lines(name = ~"cyl") fig3<- subplot(fig1, fig2)

fig3

mtcars

3d

```
mtcars$am[which(mtcars$am == 0)] <- 'Automatic' mtcars$am[which(mtcars$am == 1)] <- 'Manual' mtcars$am <- as.factor(mtcars$am)

fig <- plot_ly(mtcars, x = ~wt, y = ~hp, z = ~qsec, color = ~am, colors = c('#BF382A', '#0C4B8E')) fig <- fig %>% add_markers() fig <- fig %>% layout(scene = list(xaxis =
list(title = 'Weight'), yaxis = list(title = 'Gross horsepower'), zaxis = list(title = '1/4 mile time')))

fig
```

Brief introduction to R Plotly package

The plotly package in R is an interface to open source Javascript charting library plotly.js

Plots can be created online in the plotly web GUI or offline using the plotly package

Interactive plots that can be easily shared online using the plotly API/account

Plotly plots can embed into R Markdown, R Shiny

Tool tip feature and other controls

ggplot2 graphs can be easily converted to interactive plotly objects using ggplotly() function

Package documentation

https://cran.r-project.org/web/packages/plotly/plotly.pdf

Install the 'plotly' package

#

install.packages("plotly")

install.packages("ggplot2")

load the 'plotly' package

library(plotly)

General Syntax plot_ly()

Plot_ly functions creates a plotly object and produces interactive visualization based on data and arguments supplied

plot_ly (data, x, y , type, mode, color, size,....)

?plot_ly()

data = a dataframe

x and y axis values

type = specifies the type of plot or trace such as 'histogram', 'scatter',

'bar', 'box', 'heatmap', 'histogram', 'histogram2d', 'histogram2dcontour', 'pie', 'contour', 'scatter3d',

'surface', 'mesh3d', 'scattergeo', 'choropleth', 'scattergl', 'scatterternary', 'scattermapbox', 'area'

mode = specifies the mode, such as 'line', 'points', 'markers'

color = specifies the color of data points usually a function of I() to avoid scaling

colors = colourbrewer pallete, vector of color in hexa format

size = name or expression that defines the size of the data points

We can use other functions such as layout() for axis formating

and other functions that comes with plotly package

"mtcars A data frame with 32 observations on 11 (numeric) variables. [, 1] mpg Miles/(US) gallon [, 2] cyl Number of cylinders [, 3] disp Displacement (cu.in.) [, 4] hp Gross horsepower [, 5] drat Rear axle ratio [, 6] wt Weight (1000 lbs) [, 7] qsec 1/4 mile time [, 8] vs Engine (0 = V-shaped, 1 = straight) [, 9] am Transmission (0 = automatic, 1 = manual) [,10] gear Number of forward gears "

str(mtcars)

Pipe operator %>% will forward a value, or the

result of an expression,

into the next function call/expression.

mtcars %>% filter(carb > 1) %>% group_by(cyl) %>% summarise(Avg_mpg = mean(mpg)) %>% arrange(desc(Avg_mpg))

Tilde operator is used to define the relationship between

dependent variable and independent variables in a statistical model

formula. The variable on the left-hand side of tilde operator

is the dependent variable and the variable(s)

on the right-hand side of

tilde operator is/are called the independent variable(s).

Simple Scatter Plot

Type of plot is not exclusively defined.

Plotly will guess the best type based on the data type.

x~y

x~.

z<-x~.

help("~")

y=x*2

y~x*2

p = plot_ly(data=mtcars, x=wt, y=mpg) p

Define the plot type exclusively as "scatter"

& mode as "markers"

p = plot_ly(data=mtcars, x=~wt, y=~mpg, type = "scatter", mode = "markers") p

Change color of scatter data points

p = plot_ly(data=mtcars, x=~wt, y=~mpg, type = "scatter", mode = "markers", color = I("black")) p

Change color of data points using marker argument

p = plot_ly(data=mtcars, x=~wt, y=~mpg, type = "scatter", mode = "markers", marker = list(color="green", size=10)) p

Styled Marker

p = plot_ly(data=mtcars, x=~wt, y=~mpg, type = "scatter", mode = "markers", marker = list(size = 7, color = 'rgba(255, 182, 193, .9)', line = list(color = 'rgba(152, 0, 0, .8)', width = 3))) p

str(mtcars)

Set the color scale based on a factor variable

p = plot_ly(data=mtcars, x=~wt, y=~mpg, type = "scatter", mode = "markers", color = ~as.factor(cyl))

p

Use color Brewer palletes for data point colors

p = plot_ly(data=mtcars, x=~wt, y=~mpg, type = "scatter", mode = "markers", color = ~as.factor(cyl), colors = "Set1")

p

Use customized pallete for data point colors

pal <- c("red", "blue", "orange") p = plot_ly(data=mtcars, x=~wt, y=~mpg, type = "scatter", mode = "markers", color = ~as.factor(cyl), colors = pal)

p

Change data point shape based on factor variable

p = plot_ly(data=mtcars, x=~wt, y=~mpg, type = "scatter", mode = "markers", symbol = ~as.factor(cyl), symbols = c('circle','x','o'), marker = list(size = 5))

p

Set the color scale based on a continous variable

p= plot_ly(data=mtcars, x=~wt, y=~mpg, mode="markers", color = ~gear) p

str(mtcars) p= plot_ly(data=mtcars, x=~wt, y=~mpg, mode="markers", color = ~as.factor(disp)) p

str(mtcars)

Scatter plot with diffent data point size based on a continous variable

p = plot_ly(data=mtcars, x=~wt, y=~mpg, type="scatter", mode="markers", color = ~as.factor(cyl), size = ~hp)

p

Hide the legend

p <- plot_ly(data=mtcars, x=~wt, y=~mpg, type = "scatter", mode = "markers", symbol = ~as.factor(cyl), symbols = c('circle','x','o'), marker = list(size = 5)) %>% layout(showlegend = TRUE)

p

Change the orientation of the legend (below the plot)

p <- plot_ly(data=mtcars, x=~wt, y=~mpg, type = "scatter", mode = "markers", symbol = ~as.factor(cyl), symbols = c('circle','x','o'), marker = list(size = 5)) %>% layout(legend = list(orientation = 'h'))

p

Change the orientation of the legend (inside the plot)

p <- plot_ly(data=mtcars, x=~wt, y=~mpg, type = "scatter", mode = "markers", symbol = ~as.factor(cyl), symbols = c('circle','x','o'), marker = list(size = 5)) %>% layout(legend = list(x = 0.8, y = 0.9))

p

Change the orientation of the legend (outside the plot)

p <- plot_ly(data=mtcars, x=~wt, y=~mpg, type = "scatter", mode = "markers", symbol = ~as.factor(cyl), symbols = c('circle','x','o'), marker = list(size = 5)) %>% layout(legend = list(x = 1, y = 0.5))

p

Add title to the legen

p <- plot_ly(data=mtcars, x=~wt, y=~mpg, type = "scatter", mode = "markers", symbol = ~as.factor(cyl), symbols = c('circle','x','o'), marker = list(size = 5)) %>% layout(legend = list(x = 1, y = 0.5))

p

Adding chart title and axis labels

p <- plot_ly(data=mtcars, x=~wt, y=~mpg, type = "scatter", mode = "markers", symbol = ~as.factor(cyl), symbols = c('circle','x','o'), marker = list(size = 5)) %>%

layout(title="Scatter plot using R Plotly", xaxis=list(title="Weight", showgrid = F), yaxis=list(title="MPG", showgrid = F))

p

Customizing mouse hover text

p = plot_ly(data = mtcars, x=~mpg, y=~wt, type = "scatter", mode="markers", hoverinfo = "text", text = paste("Miles per gallon: ", mtcars$mpg, "
", "Weight: ", mtcars$wt)) p

Add annotations to the plot - add_annotations() function syntax

add_annotations() function is used to add annotations to the plot along with the following arguments

x = set annotations x coordinate position (x axis value based on the dataset)

y = set annotations y coordinate position (y axis value based on the dataset)

text = text associated with annotation

showarrow = 0 or 1 depending whether to show arrow or not

xref = "paper" telling plotly to set x reference to the plot (in which case x=0 means the left most, x=1 means right most)

yref = "paper" telling plotly to set y reference to the plot (in which case y=0 means the bottom, y=1 means top)

xref = "x" (non paper x coordinates)

yref = "y (non paper y coordinates)

arrowhead, arrowsize, font, ax, ay

Example of annotations on a single data points

display annotations for good mileage

plot_ly(data = mtcars, x=~mpg, y=~wt, type = "scatter", mode="markers") %>% add_annotations(x=mtcars$mpg[which.max(mtcars$mpg)], y=mtcars$wt[which.max(mtcars$mpg)], text="Good mileage", showarrow=T)

Example of annotations - placing text at a desired location on the plot

Display Data Source

Demo of x/y ref as "paper"

plot_ly(data = mtcars, x=~mpg, y=~wt, type = "scatter", mode="markers") %>% add_annotations(xref="paper", yref="paper", x=1, y=-0.2, text="Data Source : mtcars",

)

Example #1 of annotations on multiple data points

display annotation for low and high mileage

plot_ly(data = mtcars, x=~mpg, y=~wt, type = "scatter", mode="markers") %>% add_annotations(x=mtcars$mpg[which.max(mtcars$mpg)], y=mtcars$wt[which.max(mtcars$mpg)], text="high mileage") %>% add_annotations(x=mtcars$mpg[which.min(mtcars$mpg)], y=mtcars$wt[which.min(mtcars$mpg)], text="low mileage")

Example #2 of annotations on multiple data points

display annotations for automatic transission cars

plot_ly(data = mtcars, x=~mpg, y=~wt, type = "scatter", mode="markers") %>% add_annotations(x = mtcars$mpg[mtcars$am==0], y= mtcars$wt[mtcars$am==0], text="auto",)

Styling annotations

using font argument

plot_ly(data = mtcars, x=~mpg, y=~wt, type = "scatter", mode="markers") %>% add_annotations(x=mtcars$mpg[which.max(mtcars$mpg)], y=mtcars$wt[which.max(mtcars$mpg)], text="Good mileage", font = list(color = "green", family="sans serif", size = 20))

mtcars

```
library(plotly)
```

volcano is a numeric matrix

```
volcano
```

heatmap

```
fig <- plot_ly(z = volcano, type = "heatmap") fig
```

Contour

```
fig <- plot_ly(z = volcano, type = "contour", contours = list(showlabels = TRUE)) fig <- fig %>% colorbar(title = "Elevation \n in meters")

fig
```

Smoothing Contour Lines

```
fig1 <- plot_ly(z = volcano, type = "contour", contours = list(showlabels = TRUE), line = list(smoothing = 0.85)) fig1 <- fig1 %>% colorbar(title = "Elevation \n in meters")

fig2 <- subplot(fig,fig1)

fig2
```

greyscale

```
fig <- plot_ly(z = volcano,colors = "Greys", type = "heatmap") fig
```

single color

```
vals <- unique(scales::rescale(c(volcano))) o <- order(vals, decreasing = FALSE) cols <- scales::col_numeric("Oranges", domain = NULL)(vals) colz <- setNames(data.frame(vals[o], cols[o]), NULL) fig <- plot_ly(z = volcano, colorscale = colz, type = "heatmap") fig
```

multi

```
fig <- plot_ly(z = volcano, colors = colorRamp(c("Black","Red","orange")), type = "heatmap")

fig
```

density plot

```
dens <- with(diamonds, tapply(price, INDEX = cut, density)) df <- data.frame( x = unlist(lapply(dens, "[[", "x")), y = unlist(lapply(dens, "[[", "y")), cut = rep(names(dens), each = length(dens[[1]]$x)) )

fig <- plot_ly(df, x = ~x, y = ~y, color = ~cut) fig <- fig %>% add_lines() fig
```

3d surface

plot_ly(z = matrix(1:100, nrow = 10)) %>% add_surface()

3d surface plot

```
fig <- plot_ly(z = ~volcano,type = 'surface') fig
```

Surface Plot With Contours

```
fig <- plot_ly(z = ~volcano) %>% add_surface( contours = list( z = list( show=TRUE, usecolormap=TRUE, highlightcolor="#ff0000", project=list(z=TRUE) ) ) ) fig <- fig %>% layout( scene = list( camera=list( eye = list(x=1.87, y=0.88, z=-0.64) ) ) )

fig
```

Multiple Surfaces

```
z <- c( c(8.83,8.89,8.81,8.87,8.9,8.87), c(8.89,8.94,8.85,8.94,8.96,8.92), c(8.84,8.9,8.82,8.92,8.93,8.91), c(8.79,8.85,8.79,8.9,8.94,8.92), c(8.79,8.88,8.81,8.9,8.95,8.92), c(8.8,8.82,8.78,8.91,8.94,8.92), c(8.75,8.78,8.77,8.91,8.95,8.92), c(8.8,8.8,8.77,8.91,8.95,8.94), c(8.74,8.81,8.76,8.93,8.98,8.99), c(8.89,8.99,8.92,9.1,9.13,9.11), c(8.97,8.97,8.91,9.09,9.11,9.11), c(9.04,9.08,9.05,9.25,9.28,9.27), c(9,9.01,9,9.2,9.23,9.2), c(8.99,8.99,8.98,9.18,9.2,9.19), c(8.93,8.97,8.97,9.18,9.2,9.18) ) dim(z) <- c(15,6) z2 <- z + 1 z3 <- z - 1

fig <- plot_ly(showscale = FALSE) fig <- fig %>% add_surface(z = ~z) fig <- fig %>% add_surface(z = ~z2, opacity = 0.98) fig <- fig %>% add_surface(z = ~z3, opacity = 0.98)

fig
```

3d streamtube plot

df = read.csv('Dataset/streamtube-wind.csv')

df = read.csv('https://raw.githubusercontent.com/plotly/datasets/master/streamtube-wind.csv')

fig <- df %>% plot_ly(type = 'streamtube', x = ~x, y = ~y, z = ~z, u = ~u, v = ~v, w = ~w, sizeref = 0.5, cmin = 0, cmax = 3) fig <- fig %>% layout(scene = list(camera = list(eye = list(x = -0.7243612458865182, y = 1.9269804254717962, z = 0.6704828299861716)

```
    )
  )
```

)

fig

Starting Position and Segments

fig <- df %>% plot_ly(type = 'streamtube', x = ~x, y = ~y, z = ~z, u = ~u, v = ~v, w = ~w, starts = list(x = rep(80, 16), y = rep(c(20,30,40,50), 4), z = c(rep(0,4),rep(5,4),rep(10,4),rep(15,4))), sizeref = 0.3, showscale = F, maxdisplayed = 3000) fig <- fig %>% layout(scene = list(aspectratio = list(x = 2, y = 1, z = 0.3)), margin = list(t = 20, b = 20, l = 20, r = 20))

fig

Word Cloud provides an excellent option to visualize the text data

in the form of tags, or words,

where the importance of a word is identified by its frequency.

install.packages("wordcloud2")

library(wordcloud2) library(readr) library(dplyr) library(e1071) library(mlbench)

install.packages("mlbench")

Text mining packages

install.packages("NLP")

install.packages("tm")

install.packages("SnowballC")

install.packages("wordcloud")

library(tm) library(SnowballC) library("wordcloud") library("RColorBrewer")

loading the data

t1 <- read_csv("Dataset/Womens Clothing E-Commerce Reviews.csv") glimpse(t1)

t1$Recommended_IND[1]

Create corpus

corpus = Corpus(VectorSource(t1$Review_Text))

Look at corpus

corpus[[1]][1]

Conversion to Lowercase

corpus = tm_map(corpus, PlainTextDocument) corpus = tm_map(corpus, tolower)

Removing Punctuation

corpus = tm_map(corpus, removePunctuation)

corpus[[1]][1]

Remove stopwords

corpus = tm_map(corpus, removeWords, c("cloth", stopwords("english")))

corpus[[1]][1]

Stemming

corpus = tm_map(corpus, stemDocument)

corpus[[1]][1]

Eliminate white spaces

corpus = tm_map(corpus, stripWhitespace)

corpus[[1]][1]

Create Document Term Matrix

DTM <- TermDocumentMatrix(corpus) mat <- as.matrix(DTM) f <- sort(rowSums(mat),decreasing=TRUE) dat <- data.frame(word = names(f),freq=f) head(dat, 5)

" Word Cloud Generation

Word Cloud in R is generated using the wordcloud function. The major arguments of this function are given below:

```
 words: The words to be plotted.

freq: The frequencies of the words.

min.freq: An argument that ensures that words with a frequency below min.freq will not be plotted in the word cloud.

max.words: The maximum number of words to be plotted.

random.order: An argument that specifies plotting of words in random order. If false, the words are plotted in decreasing frequency.

rot.per: The proportion of words with 90 degree rotation (vertical text).

colors: An argument that specifies coloring of words from least to most frequent.
"
```

WordCloud 1

set.seed(100) wordcloud(words = dat$word, freq = dat$freq, random.order=TRUE)

WordCloud 2

set.seed(100) wordcloud(words = dat$word, freq = dat$freq, random.order=FALSE)

WordCloud 3

set.seed(100) wordcloud(words = dat$word, freq = dat$freq, min.freq = 15, max.words=250, random.order=FALSE, rot.per=0.30, colors=brewer.pal(8, "Dark2"))

example 2

text <- readLines(file.choose())

Load the data as a corpus

VectorSource() function creates a corpus of character vectors

docs <- Corpus(VectorSource(text))

Inspect the content of the document

inspect(docs)

Text transformation

is performed using tm_map() function to replace,

for example, special characters from the text.

Replacing "/", "@" and "|" with space:

toSpace <- content_transformer(function (x , pattern) gsub(pattern, " ", x)) docs <- tm_map(docs, toSpace, "/") docs <- tm_map(docs, toSpace, "@") docs <- tm_map(docs, toSpace, "\|")

Cleaning the text

the tm_map() function is used to remove unnecessary white space, to convert the text to lower case,

to remove common stopwords like 'the', "we"

Convert the text to lower case

docs <- tm_map(docs, content_transformer(tolower))

Remove numbers

docs <- tm_map(docs, removeNumbers)

Remove english common stopwords

docs <- tm_map(docs, removeWords, stopwords("english"))

Remove your own stop word

specify your stopwords as a character vector

docs <- tm_map(docs, removeWords, c("blabla1", "blabla2"))

Remove punctuations

docs <- tm_map(docs, removePunctuation)

Eliminate extra white spaces

docs <- tm_map(docs, stripWhitespace)

Text stemming

docs <- tm_map(docs, stemDocument)

Build a term-document matrix

```
dtm <- TermDocumentMatrix(docs) m <- as.matrix(dtm) v <- sort(rowSums(m),decreasing=TRUE) d <- data.frame(word = names(v),freq=v) head(d, 10)
```

Generate the Word cloud

```
set.seed(1234) wordcloud(words = d$word, freq = d$freq, min.freq = 1, max.words=200, random.order=FALSE, rot.per=0.35, colors=brewer.pal(8, "Dark2"))
wordcloud2(words = d$word, size = 0.7, shape = 'star')
```

dataframes in R tool

data()

mtcars

"mtcars A data frame with 32 observations on 11 (numeric) variables. [, 1] mpg Miles/(US) gallon [, 2] cyl Number of cylinders [, 3] disp Displacement (cu.in.) [, 4] hp Gross horsepower [, 5] drat Rear axle ratio [, 6] wt Weight (1000 lbs) [, 7] qsec 1/4 mile time [, 8] vs Engine (0 = V-shaped, 1 = straight) [, 9] am Transmission (0 = automatic, 1 = manual) [,10] gear Number of forward gears "

str(mtcars)

summary(mtcars)

plot(mtcars)

plot(mtcars$mpg)

Grid chart the margin of the grid(mar),

no of rows and columns(mfrow),

whether a border is to be included(bty) and position of the

labels(las: 1 for horizontal, las: 0 for vertical).

par(mfrow=c(2,2), mar=c(2,5,2,1), las=1, bty="n")

Simple Histogram

A histogram is very common plot.

It plots the frequencies that data appears

within certain ranges.

hist(mtcars$mpg, main="Histogram",xlab='MPG',col="blue",ylim=c(0,30))

Simple Scatterplot

A scatter plot provides a graphical view of

the relationship between two sets of numbers.

plot(mtcars$wt, mtcars$mpg, main="Scatterplot Example", xlab="Car Weight ", ylab="Miles Per Gallon ", pch=1, col=c("blue", "green"))

Simple Bar Plot

Barplots are useful for comparing the distribution of

a quantitative variable (numeric) between groups or categories.

counts <- table(mtcars$gear) barplot(counts, main="Car Distribution", xlab="Number of Gears",horiz=TRUE)

Stacked Bar Plot with Colors and Legend

counts <- table(mtcars$vs, mtcars$gear) barplot(counts, main="Car Distribution by Gears and VS", xlab="Number of Gears", col=c("darkblue","red"), legend = rownames(counts))

Grouped Bar Plot

counts <- table(mtcars$vs, mtcars$gear) barplot(counts, main="Car Distribution by Gears and VS", xlab="Number of Gears", col=c("darkblue","red"), legend = rownames(counts), beside=TRUE)

Simple Pie Chart

slices <- c(15,10, 12,4, 16, 8) lbls <- c("India","US", "UK", "Australia", "Germany", "France") pie(slices, labels = lbls, main="Pie Chart of Countries")

Boxplot of MPG by Car Cylinders

A boxplot provides a graphical view of the median, quartiles,

maximum, and minimum of a data set.

boxplot(mpg~cyl,data=mtcars, main="Car Milage Data", xlab="Number of Cylinders", ylab="Miles Per Gallon")

EuStockMarkets

library(dplyr)

library(ggplot2)

arrange(), filter(),select(), slice(),

summarise(), rename()

Center: mean(), median()

Spread: sd(), IQR(), mad()

Range: min(), max(), quantile()

Position: first(), last(), nth(),

Count: n(), n_distinct()

Logical: any(), all()

Other single table verbs: arrange, filter, mutate, select, slice

"playerID: Player ID code. Factor yearID: Year. Factor teamID: Team. factor lgID: League. Factor: AA AL FL NL PL UA AB: At bats. Numeric G: Games: number of games by a player. Numeric R: Runs. Numeric HR: Homeruns. Numeric SH: Sacrifice hits. Numeric "

rename(data, g=no.of games)

Step 1

data <- read.csv("Dataset/lahman-batting.csv") data

Step 2

data%>%select(c(playerID, yearID, AB, teamID, lgID, G, R, HR, SH))%>% arrange(yearID)

Step 2

data%>%select(c(playerID, yearID, AB, teamID, lgID, G, R, HR, SH))%>% arrange(desc(yearID))

data1 <- data %>% filter(teamID == "ATL" | teamID == "BOS" | teamID == "CHA") head(data1)

" data: Dataset used to construct the summary statistics group_by(lgID): Compute the summary by grouping the variable `lgID summarise(mean_run = mean(HR)): Compute the average homerun" data %>% group_by(lgID) %>% summarise(mean_run = mean(HR))

Mean

ex1 <- data %>% group_by(yearID) %>% summarise(mean_game_year = mean(G)) head(ex1)

Plot the graph

ggplot(ex1, aes(x = yearID, y = mean_game_year)) + geom_line() + theme_classic() + labs(x = "Year", y = "Average games played", title = paste("Average games played from 1871 to 2016"))

Sum

```
data %>% group_by(lgID) %>% summarise(sum_homerun_league = sum(HR))
```

Min and max

```
data %>% group_by(playerID) %>% summarise(min_G = min(G), max_G = max(G))
```

count observations–The number of observations in the current group

```
data %>% group_by(playerID) %>% summarise(number_year = n()) %>% arrange(desc(number_year))
```

first and last

```
data %>% group_by(playerID) %>% summarise(first_appearance = first(yearID), last_appearance = last(yearID))
```

nth

```
data %>% group_by(teamID) %>% summarise(second_game = nth(yearID, 3)) %>% arrange(second_game)
```

distinct values

```
data %>% group_by(teamID) %>% summarise(number_player = n_distinct(playerID)) %>% arrange(desc(number_player))
```

Multiple groups

```
data %>% group_by(yearID, teamID) %>% summarise(mean_games = mean(G)) %>% arrange(desc(teamID, yearID))
```

Filter

```
data %>% filter(yearID > 2002) %>% group_by(yearID) %>% summarise(mean_game_year = mean(G))
```

Ungroup the data

```
data %>% filter(HR > 0) %>% group_by(playerID) %>% summarise(average_HR_game = sum(HR) / sum(G)) %>% ungroup() %>% summarise(total_average_homerun =
mean(average_HR_game))
```

"A data frame is a table or a two-dimensional array-like structure in which each column contains values of one variable and each row contains one set of values from each column. Following are the characteristics of a data frame. ??? The column names should be non-empty. ??? The row names should be unique. ??? The data stored in a data frame can be of numeric, factor or character type. ??? Each column should contain same number of data items."

Create the data frame.

emp.data <- data.frame(emp_id = c (1:5), emp_name = c("Ram","Alex","Raj","Ryan","siva"), salary = c(623.3,515.2,611.0,729.0,843.25), start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", NA,"2015-03-27")), stringsAsFactors = FALSE)

Print the data frame.

print(emp.data)

Get the structure of the data frame.

str(emp.data)

Extract Specific columns.

result <- data.frame(emp.data$emp_name,emp.data$salary) print(result)

Extract first two rows.

result <- emp.data[1:2,] print(result)

Extract 3rd and 5th row with 2nd and 4th column.

result <- emp.data[c(3,5),c(2,4)] print(result)

Add the "dept" coulmn.

emp.data$dept <- c("IT","Operations","IT","HR","Finance") v <- emp.data print(v)

Bind the two data frames.

emp.finaldata <- rbind(emp.data,emp.newdata) print(emp.finaldata)

check dataframe

print(is.data.frame(emp.data)) print(ncol(emp.data)) print(nrow(emp.data))

null

sum(is.na(emp.data$start_date)) sum(is.na(emp.data))

sort

print(emp.data[order(emp.data$salary),])

Get the max salary from data frame.

sal <- max(emp.data$salary) print(sal)

Get the person detail having max salary.

retval <- subset(emp.data, salary == max(salary)) print(retval)

retval <- subset(emp.data, dept == "IT") print(retval)

```
info <- subset(emp.data, salary > 600 & dept == "IT") print(info)

retval <- subset(emp.data, as.Date(start_date) > as.Date("2014-01-01")) print(retval)

write.csv(retval,"E:/COURSE/Data-Visualization/R-Lan/Prog/output.csv") newdata <- read.csv("E:/COURSE/Data-Visualization/R-Lan/Prog/output.csv", nrows=2)
print(newdata)
```

library(tidyverse) library(ggplot2)

install.packages("gridExtra")

library(gridExtra) library(ggplot2) library(dplyr) library(plotly) library(hrbrthemes)

data()

head(diamonds) str(diamonds)

qplot(carat, price, data = diamonds)

qplot(log(carat), log(price), data = diamonds)

qplot(carat, price, data = diamonds[1:50,], colour = color)

qplot(carat, price, data = diamonds[1:50,], shape = cut)

qplot(carat, price, data = diamonds[1:50,], size = price)

library(scales) qplot(carat, price, data = diamonds, colour = I(alpha("black", 1/200)))

qplot(carat, price, data = diamonds, geom = c("point", "smooth"))

qplot(carat, data = diamonds, geom = "histogram") qplot(carat, data = diamonds, geom = "density") qplot(carat, data = diamonds, geom = "histogram", fill = color) qplot(carat, data = diamonds, geom = "density", colour = color)

str(diamonds)

points-scatterplot

p<-ggplot(diamonds, aes(x=carat, y=price, color=cut))+ geom_point() p ggsave("myggplot.png", plot=p, width = 10,height=20, dpi = 300) # save a stored ggplot

bar

p<- ggplot(diamonds, aes(cut)) + geom_bar() p

p<-ggplot(diamonds, aes(cut)) + geom_bar(aes(fill = clarity),position = "stack") p

ggplot(diamonds, aes(cut)) + geom_bar(aes(fill = clarity), position = "fill")

ggplot(diamonds, aes(cut)) + geom_bar(aes(fill = clarity), position = "dodge")

"

marker

p<-ggplot(diamonds, aes(x=carat, y=price)) + geom_point(aes(size=carat, shape=clarity, alpha=price)) p

"

Layers

Overlay a smoothing line on top of the scatter plot using geom_smooth

Adding scatterplot geom (layer1) and smoothing geom (layer2).

p<-ggplot(diamonds) + geom_point(aes(x=carat, y=price, color=cut)) + geom_smooth(aes(x=carat, y=price, color=cut)) p

facet-faceting that allows the user to split one plot into

multiple plots based on a factor included in the dataset.

columns defined by 'cut'

facet_wrap(~ factor1 + factor2 + ... + factorn)

p<- p+facet_wrap(~ cut, nrow=3,ncol=2)
p

facet_grid(row_variable ~ column_variable)

p<-p + facet_grid(color ~ cut) p

p <- ggplotly(p) p

ggplot(diamonds, aes(x=carat, y=price, color=cut)) + geom_point() + facet_wrap(~ clarity)

plot <- ggplot(diamonds, aes(x=carat,y=price))+ geom_density(aes(fill=cut),alpha=0.5)

plot<-plot+facet_wrap(~cut) plot

the main title, x and y axis labels

ggplot(diamonds, aes(x=carat, y=price, color=cut)) + geom_point() + geom_smooth() + ggtitle("Scatter") + xlab("carat") + ylab("price")+ coord_cartesian(ylim=c(0, 10000))

Change the appearance of the main title &axis labels -- theme() & element_text()

main title -p + theme(plot.title = element_text(family, face, colour, size))

x/y axis title -p + theme(axis.title.x/y = element_text(family, face, colour, size))

"

family : font family face : font face. Possible values are plain, italic, bold and bold.italic colour : text color size : text size in pts hjust : horizontal justification (in [0, 1]) vjust : vertical justification (in [0, 1]) lineheight : line height. In multi-line text, the lineheight argument is used to change the spacing between lines. color : an alias for colour "

Legend - Deleting and Changing Position

remove legend

p1 <- ggplot(diamonds, aes(x=carat, y=price, color=cut)) + geom_point() + geom_smooth() +
theme(legend.position="none", axis.title.x = element_text(color="blue", size=14, face="bold"), axis.title.y = element_text(color="#993333", size=14, face="bold")) +
labs(title="legend.position='none'")

legend at top

p2 <- ggplot(diamonds, aes(x=carat, y=price, color=cut)) + geom_point() + geom_smooth() + theme(legend.position="top") + labs(title="legend.position='top'")

legend inside the plot.

p3 <- ggplot(diamonds, aes(x=carat, y=price, color=cut)) + geom_point() + geom_smooth() + labs(title="legend.position='coords inside plot'") +
theme(legend.justification=c(1,0), legend.position=c(1,0))

arrange

grid.arrange(p1, p2, p3,nrow=1,ncol=3)

library(ggplot2) library(dplyr) require(maps) require(viridis) theme_set(theme_void())

Create a simple map World map

world_map <- map_data("world") world_map <- map_data("usa") world_map <- map_data("nz") world_map <- map_data("county") world_map <- map_data("italy") world_map <- map_data("state") head(world_map)

ggplot(world_map, aes(x = long, y = lat, group = group)) + geom_polygon(fill="lightgray", colour = "white")

Map for specific regions Some EU Contries

some.eu.countries <- c("Portugal", "Spain", "France", "Switzerland", "Germany", "Austria", "Belgium", "UK", "Netherlands", "Denmark", "Poland", "Italy", "Croatia", "Slovenia", "Hungary", "Slovakia", "Czech republic")

Retrieve the map data

some.eu.maps <- map_data("world", region = some.eu.countries)

Compute the centroid as the mean longitude and lattitude

Used as label coordinate for country's names

region.lab.data <- some.eu.maps %>% group_by(region) %>% summarise(long = mean(long), lat = mean(lat))

map-plot

ggplot(some.eu.maps, aes(x = long, y = lat)) + geom_polygon(aes(group = group, fill = region))+ geom_text(aes(label = region), data = region.lab.data, size = 3, hjust = 0.5)+ scale_fill_viridis_d()+ theme_void()+ theme(legend.position = "none")

Example-2 Prepare the USArrests data

library(dplyr) arrests <- USArrests USArrests arrests$region <- tolower(rownames(USArrests)) head(arrests)

Retrieve the states map data and merge with crime data

states_map <- map_data("state") arrests_map <- left_join(states_map, arrests, by = "region")

Create the map

ggplot(arrests_map, aes(long, lat, group = group))+ geom_polygon(aes(fill = Assault), color = "white")+ scale_fill_viridis_c(option = "C")

Create the map

choro <- merge(states_map, arrests_map, sort = FALSE, by = "region") choro <- choro[order(choro$order),] ggplot(choro, aes(long, lat)) + geom_polygon(aes(group = group, fill = assault)) + coord_map("albers", at0 = 45.5, lat1 = 29.5)

Example-3 World map colored by life expectancy

library(ggplot2) install.packages("WHO") library("WHO") library("dplyr") life.exp <- get_data("WHOSIS_000001") # Retrieve the data life.exp <- life.exp %>% filter(year == 2015 & sex == "Both sexes") %>% # Keep data for 2015 and for both sex select(country, value) %>% # Select the two columns of interest rename(region = country, lifeExp = value) %>% # Rename columns # Replace "United States of America" by USA in the region column mutate(region = ifelse(region == "United States of America", "USA", region))

Merge map and life expectancy data:

world_map <- map_data("world") life.exp.map <- left_join(life.exp, world_map, by = "region")

geom_polygon

ggplot(life.exp.map, aes(long, lat, group = group))+ geom_polygon(aes(fill = lifeExp), color = "white")+ scale_fill_viridis_c(option = "C")

geom_map

ggplot(life.exp.map, aes(map_id = region, fill = lifeExp))+ geom_map(map = life.exp.map, color = "white")+ expand_limits(x = life.exp.map$long, y = life.exp.map$lat)+ scale_fill_viridis_c(option = "C")