1. 데이터 시각화를 위한 plotly

R을 다루는 많은 교육코스나 서적에서 데이터의 시각화는 대부분 R base에서 제공하는 함수를 사용하거나 ggplot2 패키지를 사용하여는 방법을 위주로 설명한다. 이 두가지 방법은 데이터 시각화 결과가 우수한 편이기 때문에 많이 사용되고 있지만 정적(Static) 시각화이다. 정적 시각화는 최근 인포그래픽(Infographic)이라고 불리며 일반적으로 문서나 인쇄물에 많이 사용되고 웹에 게시되는 이미지로 사용된다. 그렇기 때문에 대부분 png, jpg, pdf 등의 벡터 혹은 픽셀 이미지 파일로 제공된다. 정적 데이터 시각화는 데이터 분석가의 의도에 맞춰 작성되기 때문에 데이터 분석가의 분석에 의존적일 수 밖에 없으며 독자의 의도에 따른 해석은 매우 제한될 수 밖에 없다.

이러한 제한점을 극복하기 위해 사용되는 데이터 시각화 방법이 동적(Dynamic) 시각화 혹은 인터랙티브(Interactive) 시각화라고 하는 방법이다. 이 동적 시각화는 시각화를 사용하는 사용자의 의도에 따라 데이터를 다각적 관점에서 살펴볼 수 있다는 점이 동적 시각화와 가장 크게 차이나는 점이다. 사용자의 의도에 따라 데이터가 동적으로 변동되어야 하기 때문에 데이터 시각화에 사용되는 매체는 인쇄물 형태 매체가 불가능하고 웹을 통해 제공한다. 따라서 일반적으로 동적 시각화는 웹 사이트에서 제공하는 대시보드(DashBoard)의 형태로 제공되는 것이 일반적이기 때문에 동적 시각화를 위해서는 동적 시각화 전용 패키지를 사용해 시각화 객체를 만드는 방법이외에 shiny 등의 패키지를 사용하여 대시보드를 만드는 것도 같이 익혀야 한다는 어려움이 따른다. R에서 동적 시각화를 위해 제공되는 패키지는 plotly, rbokeh, highcharter 등이 있다.

따라서 정적 시각화와 동적 시각화의 어느것이 더 효용성이 있는지를 단언할 수 없다. 데이터 시각화가 사용되는 매체, 데이터 시각화를 보는 대상, 데이터 시각화에서 보여주고자 하는 스토리에 따라서 정적 시각화를 사용해야 할 때와 동적 시각화를 사용해야 할 때를 적절히 선택해야 한다.

이번 장에서는 R에서 동적 시각화로 많이 사용되는 plotly 패키지를 사용하여 데이터를 시각화하는 방법을 알아본다. plotly를 사용한 데이터 시각화를 실습하기 위한 데이터는 ggplot2 실습에서 사용하던 취업통계 데이터(df\_취업통계)와 코로나바이러스 데이터 셋을 사용하도록 하겠다.

코로나바이러스 데이터 셋은 R CRAN에서 공식적으로 제공하는 coronavirus 패키지를 통해 제공받을 수 있다.[^1]

# if(!require(coronavirus)) {  
# install.packages('coronavirus')  
# library(coronavirus)  
# }  
  
if(!require(readr)) {  
 install.packages('readr')  
 library(readr)  
}  
  
if(!require(lubridate)) {  
 install.packages('lubridate')  
 library(lubridate)  
}  
#   
#   
# coronavirus\_df <- read\_csv("C:/R/git/datavisualization/chap3/coronavirus.csv",   
# col\_types = cols(date = col\_date(format = "%Y-%m-%d"),  
# cases = col\_number()  
# )  
# )  
  
##coronavirus\_df <- read\_csv("https://raw.githubusercontent.com/RamiKrispin/coronavirus/master/csv/coronavirus.csv",   
## col\_types = cols(date = col\_date(format = "%Y-%m-%d"),   
## cases = col\_number()))  
  
# covid\_stat\_by\_nation <- coronavirus\_df %>%  
# filter(type != "recovery", !is.na(code3)) %>%  
# group\_by(type, continent\_name, country, uid) %>%  
# summarise(cases = sum(cases), population = max(population), .groups = "drop") %>%  
# mutate(type = factor(type, levels = c("confirmed", "death"))) %>%  
# pivot\_wider(names\_from = type, values\_from = cases) %>%  
# mutate(death\_rate = death / confirmed, confirmed\_rate = confirmed / population) %>%  
# filter(!is.na(country))  
  
# covid19\_vaccine\_df <- read\_csv(file = "C:/R/git/datavisualization/chap3/covid19\_vaccine.csv",  
# col\_types = cols(Date = col\_date(format = "%Y-%m-%d"),  
# Report\_Date\_String = col\_date(format = "%Y-%m-%d"),  
# UID = col\_number(),  
# Province\_State =col\_character()  
# )  
# )  
# covid\_joined <- inner\_join(covid\_stat\_by\_nation, vaccine\_stat\_by\_nation, by = c('country' = 'country\_region', 'uid' = 'uid'))  
  
#https://github.com/owid/covid-19-data/tree/master/public/data  
  
  
covid19\_df <- read\_csv(file = "D:/R/data/owid-covid-data.csv",  
 col\_types = cols(Date = col\_date(format = "%Y-%m-%d")  
 )  
 )  
  
covid19\_df\_100 <- covid19\_df |>   
 filter((iso\_code %in% c('KOR', 'OWID\_ASI', 'OWID\_EUR', 'OWID\_OCE', 'OWID\_NAM', 'OWID\_SAM'))) |>  
 filter(date >= today() - 100) |>   
 arrange(date)  
  
covid19\_df\_100\_wide <- covid19\_df\_100 |> select(date, location, new\_cases) |>   
 pivot\_wider(id\_cols = date, names\_from = location, values\_from = new\_cases) |>   
 arrange(date)  
  
names(covid19\_df\_100\_wide) <- c('date', 'Asia', 'Europe', 'North\_America', 'Oceania', 'South\_America', 'South\_Korea')  
  
# covid19\_df <- covid19\_df |> select(iso\_code, continent, location, date, population, gdp\_per\_capita, total\_cases, new\_cases, total\_deaths, new\_deaths, icu\_patients\_per\_million, hosp\_patients\_per\_million, reproduction\_rate, total\_tests, new\_tests, total\_vaccinations, people\_fully\_vaccinated, total\_boosters, people\_fully\_vaccinated\_per\_hundred, total\_boosters\_per\_hundred)  
#   
#   
covid19\_stat <- covid19\_df |> group\_by(iso\_code, continent, location) |>  
 summarise(population = max(population, na.rm = T), gdp\_per\_capita = max(gdp\_per\_capita, na.rm = T),  
 total\_cases = max(total\_cases, na.rm = T), new\_cases = sum(new\_cases, na.rm = T),  
 total\_deaths = max(total\_deaths, na.rm = T), new\_deaths = sum(new\_deaths, na.rm = T), death\_per\_million = round(total\_deaths / population \*100000, 5),  
 icu\_patients\_per\_million = last(icu\_patients\_per\_million),  
 hosp\_patients\_per\_million = last(hosp\_patients\_per\_million),  
 reproduction\_rate = last(reproduction\_rate),  
 total\_tests = max(total\_tests, na.rm = T), new\_tests = sum(new\_tests, na.rm = T),  
 total\_vaccinations = max(total\_vaccinations, na.rm = T),  
 people\_fully\_vaccinated = max(people\_fully\_vaccinated, na.rm = T),  
 total\_boosters = max(total\_boosters, na.rm = T),  
 fully\_vaccinated\_rate = people\_fully\_vaccinated / population,  
 people\_fully\_vaccinated\_per\_hundred = max(people\_fully\_vaccinated\_per\_hundred, na.rm = T),  
 total\_boosters\_per\_hundred = max(total\_boosters\_per\_hundred, na.rm = T)  
 ) |> ungroup()

[^1] : coronavirus 패키지에서 제공하는 데이터 셋의 자세한 내용은 해당 패키지의 Github에서 확인할 수 있다.(<https://ramikrispin.github.io/coronavirus/articles/intro_coronavirus_dataset.html>)

# plotly란?

plotly는 오픈 소스인 JavaScript로 구현된 plotly.js를 기반으로 R에서 생성한 데이터 시각화 객체를 Javascript로 생성해주는 패키지이다. 따라서 plotly로 생성된 시각화는 결국 HTML 코드로 구현되고 이 코드는 웹브라우저 상에서 작동함으로써 사용자의 반응에 따른 데이터의 표현이 가능하다. plotly를 통해 생성된 데이터 시각화의 HTML은 R에서 JavaScript를 사용할 수 있게하는 htmlwidgets 프레임워크에서 동작하기 할 수 있어 HTML자체로 사용할 수도 있고 R Markdown이나 Shiny App, R-Studio, Jupiter Notebook 등에서 자유롭게 사용이 가능하다.

R에서 plotly를 사용한 인터랙티브 데이터 시각화를 만드는 방법은 두 가지이다. 첫 번째는 R에서 데이터 시각화에 가장 많이 사용하는 ggplot2를 사용하여 생성한 객체를 plotly 객체로 전환하는 방법이고 두 번쨰는 plotly 패키지에서 제공하는 함수들을 사용하여 plotly 객체를 직접 생성하는 방법이다.

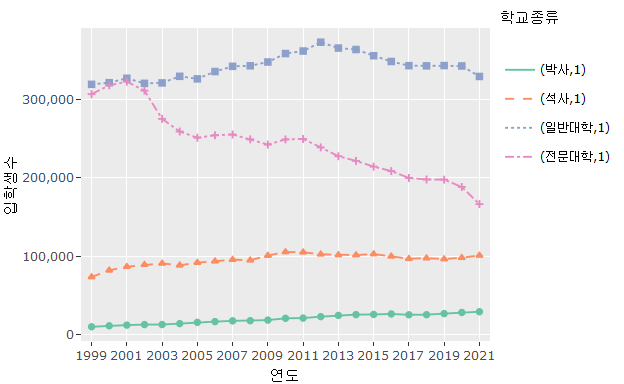
# ggplot 객체의 전환

기존의 R 사용자가 가장 쉽게 plotly를 사용한 인터랙티브 데이터 시각화를 생성하는 방법은 그동안 사용했던 ggplot2 패키지를 사용하여 생성했던 ggplot 객체를 plotly 객체로 전환하는 것이다. 이 방법은 plotly 패키지에서 제공하는 ggplotly()를 사용하면 간단히 전환된다.

ggplotly(p = ggplot2::last\_plot(), width = NULL, height = NULL, tooltip = “all”, dynamicTicks = FALSE, layerData = 1, originalData = TRUE, source = “A”, …)  
- p : plotly로 전환할 ggplot 객체  
- width : plotly 객체의 너비 설정  
- height : plotly 객체의 높이 설정  
- tooltip : plotly 객체에서 마우스의 위치에 따라 표시되는 툴팁의 문자열 설정  
- dynamicTicks : plotly 객체가 Zooming 될 때 눈금자(Tick)을 동적으로 재설정할 것인지를 설정하는 논리값  
- layerData : 레이어의 데이터를 리턴할지를 설정  
- originalData : 원천 데이터(original)를 리턴할지 스케일(scale)된 데이터를 리턴할지를 설정하는 논리값

앞 장에서 생성했던 ggplot 객체를 plotly 객체로 전환하는 코드는 다음과 같다.

if(!require('plotly')) {  
 install.packages('plotly')  
 library(plotly)  
}  
  
ggplotly <-df\_입학자\_long |>   
 ## 지역이 전체, 학교종류가 '전문대학', '일반대학', '석사', '박사'인 데이터 필터링  
 filter(지역 == '전체', 학교종류 %in% c('전문대학', '일반대학', '석사', '박사')) |>  
 ## x축이 연도로 매핑된 ggplot 객체 생성  
 ggplot(aes(x = 연도, y = 입학생수, color = 학교종류 )) +  
 ## y축이 입학생수, group, color, linetype이 학교종류로 매핑된 geom\_line 레이어 생성  
 geom\_line(aes(group = 학교종류, linetype = 학교종류)) +  
 ## y축이 입학생수, group, color이 학교종류로 매핑된 geom\_point 레이어 생성  
 geom\_point(aes(shape = 학교종류), show.legend = FALSE) +  
 ## x축의 눈금을 1999부터 2021까지 2씩 증가한 수치로 설정  
 scale\_x\_discrete(breaks = c(seq(from = 1999, to = 2021, by = 2))) +  
 ## y축에 표현된 라벨을 scales 패키지의 comma함수를 적용(scales 패키지 설치 필요)  
 scale\_y\_continuous(labels = scales::comma) +   
 ## 색 설정을 RColorBrewer 패키지의 'Accent'로 설정  
 scale\_color\_brewer(palette = 'Set2', labels = c('박사', '석사', '일반대학', '전문대학'))  
  
ggplotly(ggplotly)



앞의 plotly 객체에서 보면 ggplot 객체를 정확히 전환하지는 못한다.[[1]](#footnote-22) 그러나 ggplotly()는 ggplot2패키지를 사용해서 생성한 대부분의 ggplot 객체를 전환할 수 있다. 게다가 ggplot2 패키지를 확장해서 사용하게 해주는 ggforce, GGally와 같은 확장 패키지로 생성된 객체도 변환이 가능하다는 장점이 있다. 따라서 plotly 객체를 다루는 방법을 잘 익히면 기존에 생성했던 ggplot 객체의 시각화를 재활용 할 수있게 된다. 그리고 plotly를 사용하다보면 느끼겠지만 ggplot2의 통계 요소와 분할 요소의 몇몇 요소들은 plotly보다는 ggplot2가 훨씬 편리할 떄가 있다. 이러한 경우 ggplot2와 plotly를 적절히 혼용하면 매우 좋은 결과를 얻을 수 있다.

# plotly 구성

plotly 객체는 pltoly.js를 지원하는 스키마로 표현된다. 사실 ggplot2이던 plotly이던 각각의 시각화 객체는 R에서 특별하게 정의된 데이터 구조로 표현된다. 이 데이터 구조가 R의 그래픽 엔진을 통해 이미지로 표현되는 것이다. 따라서 plotly 객체도 우리가 눈으로 보기에는 이미지로 보이지만 R에서는 plotly.js에서 지원하는 데이터 구조로 표현된 데이터 객체인 것이다.

plotly 객체를 생성하기 위한 코드는 plot\_ly()를 사용한 plotly 객체 생성, add\_trace()를 사용한 trace 추가, layout()을 사용한 레이아웃 설정의 세 부분으로 구성된다.

## plotly 객체 생성 : plot\_ly()

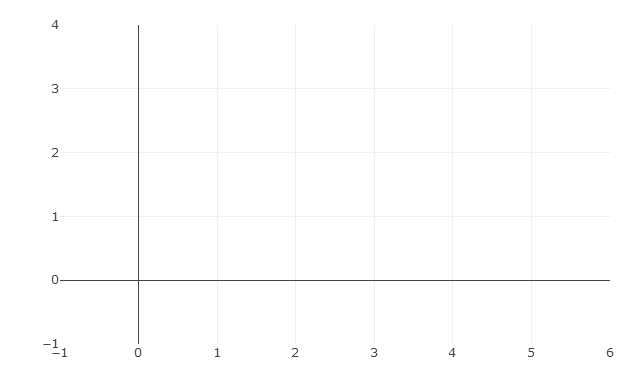
설명한바와 같이 plotly는 보여지기에 그래픽으로 보여지지만 내부적으로는 데이터 구조형태로 표현된다. 따라서 plotly객체를 표현하는 데이터 구조(스키마)를 생성하기 위해서는 제일 먼저 plotly 객체를 생성하는 기본 스키마를 정의하기 위한 초기화 함수가 필요하다. plotly 객체를 시작하기 위한 초기화 함수는 plot\_ly()이다.

plot\_ly()는 plotly.js에서 정의된 plotly 객체 스키마를 생성하는 함수이다. 사용자가 직접 plotly.js형태의 스키마 객체를 타이핑하여 생성하는 것은 어려움이 따르기 때문에 plotly 객체 스키마 생성, 기본 스키마 속성 설정 등을 지원하는 함수이다. 이 방법은 ggplot객체를 생성하기 위해 ggplot()를 사용하여 ggplot 객체를 생성하는 것과 동일한 방법이고 이 방법에서 영감을 받았다고 한다.

plot\_ly(data = data.frame(), …, type = NULL, name, color, colors = NULL, alpha = NULL, stroke, strokes = NULL, alpha\_stroke = 1, size, sizes = c(10, 100), span, spans = c(1, 20), symbol, symbols = NULL, linetype, linetypes = NULL, split, frame, width = NULL, height = NULL, source = “A”)  
- p : plotly로 시각화할 데이터프레임  
- … : type에서 설정하는 trace의 종류에 따라 설정할 수 있는 속성 설정  
- type : trace 타입 설정  
- name : plotly 객체의 trace name 속성 설정  
- color : ‘fill-color’ 속성으로 매핑될 색 값(value) 설정  
- colors : ‘fill-color’에 매핑될 colorbrewer2.org의 팔레트 이름이나 16진수의’#RRGGBB’형태로 표현된 색의 벡터(vector) 설정  
- alpha : color에서 설정된 색의 투명도 값(value) 설정  
- stroke : ‘stroke-color’(외곽선 색) 속성으로 매핑될 색 값(value) 설정  
- strokes : ‘stroke-color’(외곽선 색)에 매핑될 colorbrewer2.org의 팔레트 이름이나 16진수의 ‘#RRGGBB’형태로 표현된 색 벡터(vector) 설정/ - alpha-stroke : stroke(외곽선)에 적용될 alpha 값(value) 설정  
- size : ’fill-size’에 매핑될 크기값(value) 설정  
- sizes : size에 매핑될 수치 벡터(vector) 설정  
- span : ’stroke-size’(외곽선 두께)에 매핑될 두께 값(value) 설정  
- spans : ‘stroke-size’(외곽선 두께)에 매핑될 두께 벡터(vector) 설정  
- symbol : 점 표현에 사용되는 도형 번호(pch)나 도형 이름 값(value) 설정  
- symbols : 점 표현에 사용되는 도형 번호(pch)나 도형 이름 벡터(vector) 설정  
- linetype : 라인 타입의 설정에 사용되는 번호나 라인 타입 값(value) 설정  
- linetypes : 라인 타입의 설정에 사용되는 번호나 라인 타입 벡터(vector) 설정  
- split : 다중 traces를 생성시 사용하는 값 설정  
- frame : 애니메이션 프레임 생성시 사용할는 값 설정  
- width : 플롯의 너비(픽셀) 설정  
- height : 플롯의 높이(픽셀) 설정

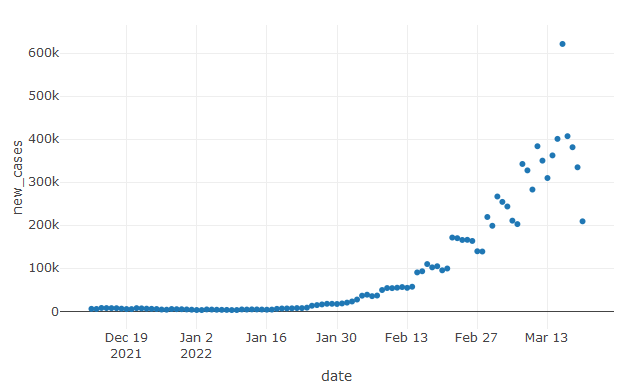
이처럼 plot\_ly()의 함수에서는 많은 매개변수가 사용된다. 특히 ’…’으로 표기된 부분은 type 매개변수에서 설정하는 trace 타입에 따라 설정 내용이 매우 달라진다. ggplot2에서 geom\_\*()를 사용하여 기하 요소 레이어를 하나 하나 설정하면서 전체 시각화를 완성하는 것과 유사하게 plotly에서는 여러개의 trace를 추가함으로써 전체 시각화를 완성해나갈수 있다. 다만 ggplot2의 ggplot()에서는 데이터 요소와 미적요소만을 설정할 수 있었지만 plotly에서는 데이터, trace, trace에 따른 속성까지 설정이 가능하므로 plot\_ly()만가지고도 시각화를 완성할 수 있다. 반면 동시에 여러개의 trace가 포함되는 시각화에서는 뒤에서 설명할 add\_trace()나 add\_\*()를 사용하여 trace를 추가하게 되는데 plot\_ly()에서 설정한 속성들을 상속받게 된다. 이 과정에서 원치않는 속성의 상속을 방지하기 위해 plot\_ly()에 매개변수를 넣지않고 단순히 plotly 객체의 초기화 명령으로 사용도 가능하다.

covid19\_df\_100 |> plot\_ly()



plot\_ly()의 매개변수를 설정할 때 꼭 알아 두어야 하는 것은 변수를 매핑하는 방법과 값을 설정하는 방법이 다르다는 것이다. ggplot2에서는 변수를 매핑하기 위해서 aes()를 사용함으로써 변수를 매핑하였고 값을 실정하기 위해서는 aes() 밖에 선언함으로써 설정이 가능하였다. plotly에서는 aes() 대신 ~를 사용하여 변수를 매핑한다.

covid19\_df\_100 |> filter(iso\_code == 'KOR') |>   
 plot\_ly(x = ~date, y = ~new\_cases)



앞선 plot\_ly()코드에서는 X축과 Y축을 지정하는 매개변수인 x, y에 ~를 사용하여 각각의 변수를 매핑하였고 색을 설정하는 매개변수인 color에는 I()를 사용하여 ’red’를 설정하였다. 사실 plotly 객체에서 가장 중요한 것은 trace의 종류이다. plot\_ly()만을 사용하여 그래프를 완성할 때 trace의 종류를 생략하면 plotly에서 데이터를 파악하여 가장 좋은 trace를 설정해준다.

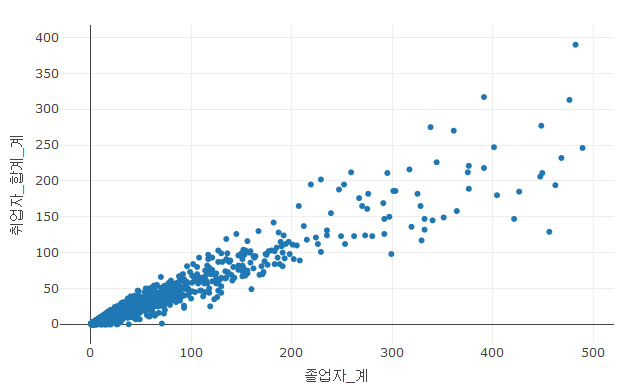
앞의 plot\_ly()에서 매개변수로 매핑하거나 설정되는 사용되는 type은 plot\_ly() 선언 단계에서 사용도 가능하고 다음번 단계인 trace 추가 단계에서도 사용 가능하다. 이 두 방법의 차이는 plot\_ly에서 사용된 type은 추가되는 trace에서 기본적으로 상속받아 사용되지만 trace 추가 단계에서 사용되는 type은 해당 trace에서만 사용되는 type이 된다는 점이 다르다.

plot\_ly에서 사용이 가능한 대표적인 type은 다음과 같다.

### x, y, z

add\_trace()에서 가장 기본적으로 설정하는 type은 x, y, z이다. 이 세 개의 type은 X축, Y축, Z축에 매핑하는 변수를 설정한다. 앞서 설명한 바와 같이 x, y, z에 매핑하는 변수는 ~를 사용하여 매핑하는데 변수를 매핑하지 않고 벡터를 설정하는 것도 가능하다.

set.seed(123)  
df\_취업통계\_2000 <- df\_취업통계 |> filter(졸업자\_계 < 500) |> sample\_n(2000)  
  
df\_취업통계\_2000 |> plot\_ly(x = ~졸업자\_계, y = ~취업자\_합계\_계)



plotly는 3차원 그래픽을 지원하기 때문에 Z축을 설정할 수 있다. 동적 시각화는 사용자가 시각화를 자신이 원하는 방향으로 설정하여 관찰할 수 있기 때문에 3차원 효과가 효율적일 수 있다. 하지만 일반적으로 데이터 시각화에서는 3차원 을 사용하는 것은 크게 효과적이지 않다고 알려져 있기 때문에 3차원의 활용은 주의할 필요가 있다.

df\_취업통계\_2000 |>  
 plot\_ly(x = ~졸업자\_계, y = ~취업자\_합계\_계, z = ~대계열)

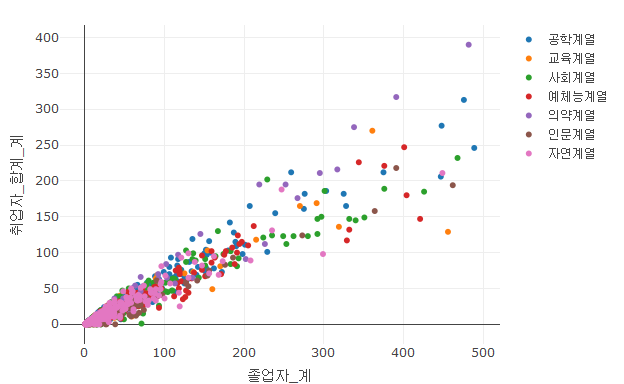


### name

name은 plotly에서 추가되는 각각의 trace에 대한 이름을 설정한다. 이 이름은 범례 아이템과 마우스 포인터가 데이터 점에 위치할때 나타나는 호버(Hover)에 표기되는 이름이 된다.

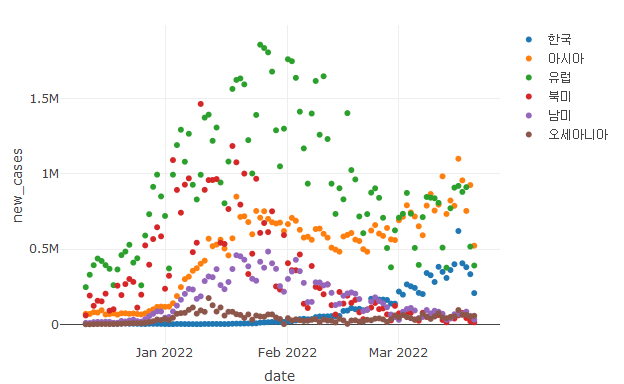
다음은 name에 범례에 표기되어야 할 변수를 매핑하여 범례 아이템 이름을 설정하는 코드이다.

df\_취업통계\_2000 |>  
 plot\_ly(x = ~졸업자\_계, y = ~취업자\_합계\_계, name = ~대계열)



만약 범례를 변수 열로 설정된 값을 사용하지 않고 바꾸려면 범례 아이템을 바꾸는 것이 아니고 각각의 데이터에 매핑될 벡터를 만들어 주어야 한다. ggplot2와 같은 정적 시각화는 시각화가 만들어지면 변경이 되지 않기 때문에 데이터를 시각화하는 부분과는 별도로 범례만을 따로 설정할 수 있지만 동적 시각화는 마우스 포인터를 데이터에 가져가면 해당 데이터에 대한 내용이 표기되어야 하는데 이 값이 범례값과 달라지면 안되기 때문에 범례만 수정할 수 없고 모든 데이터에 1:1로 매핑되는 범례 아이템 이름 벡터가 필요하다.

legend\_items <- covid19\_df\_100 |>   
 mutate(legend\_name = case\_when(  
 iso\_code == 'KOR' ~ '한국',   
 iso\_code == 'OWID\_ASI' ~ '아시아',  
 iso\_code == 'OWID\_EUR' ~ '유럽',  
 iso\_code == 'OWID\_NAM' ~ '북미',  
 iso\_code == 'OWID\_OCE' ~ '오세아니아',  
 iso\_code == 'OWID\_SAM' ~ '남미')) |>  
 select(legend\_name) |>  
 pull()  
  
legend\_items <- fct\_relevel(legend\_items, '한국', '아시아', '유럽', '북미', '남미', '오세아니아')  
  
covid19\_df\_100 |>  
 plot\_ly(x = ~date, y = ~new\_cases, name = ~legend\_items)

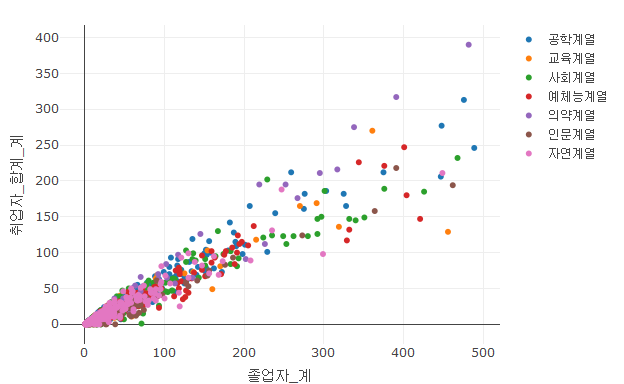


### hovertext, hoverinfo, hovermode, hovertemplete

plotly와 같은 동적 시각화에서는 대부분 마우스 포인터를 데이터가 표시된 점이나 선에 위치하면 해당 위치의 데이터가 표시된다. plotly에서는 이렇게 데이터의 정보를 표시하는 말풍선을 ’hover’라고 한다. ’hover’는 시각화를 설계하는 사용자에 따라 표시할 정보를 설정할 수 있는데 이 정보를 설정하는 type이 hover\*이다.

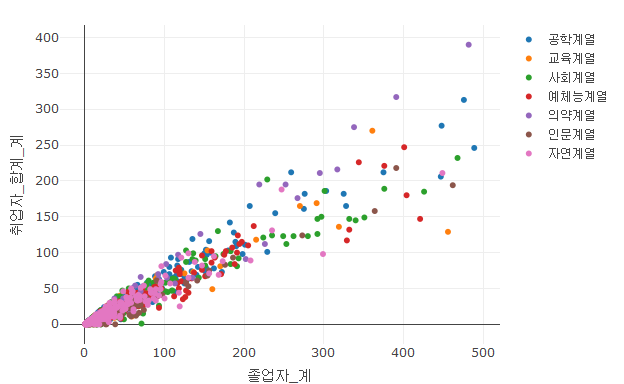
hovertext는 X, Y 좌표에 표시되는 문자열을 설정하는 type이다. 단순 문자열을 설정하면 모든 데이터 호버에 동일한 문자열이 표기되지만 벡터를 설정하면 각각의 호버에 매핑된 결과가 표시된다.

df\_취업통계\_2000 |>  
 plot\_ly(x = ~졸업자\_계, y = ~취업자\_합계\_계, name = ~대계열, hovertext = ~학과명)



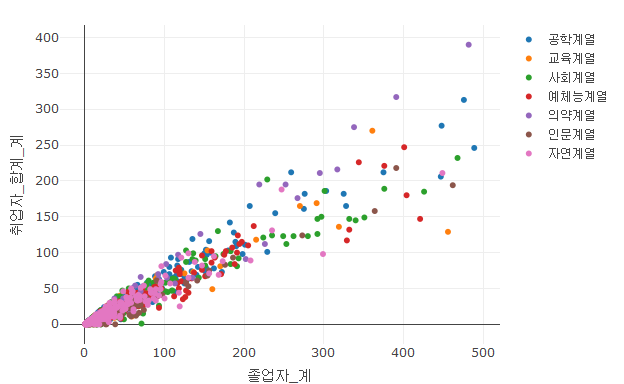
hoverinfo는 호버에 표시되는 데이터 정보를 설정하는 type이다. 호버에 표시되는 데이터 정보는 각각의 trace에 따라 다르지만 스캐터 trace에서는 기본적으로 X, Y, Z축의 좌표를 표기하는데 기본값은 ’all’이 설정된다. 이 외에도 x(X축 좌표), y(Y축 좌표), z(Z축 좌표), text(특정 문자열), name(trace name), none(제거), skip(생략)이 사용될 수 있고 각각은 +를 사용하여 조합할 수 있다.

df\_취업통계\_2000 |>  
 plot\_ly(x = ~졸업자\_계, y = ~취업자\_합계\_계, name = ~대계열, hovertext = ~학과명, hoverinfo = 'x+y+text')



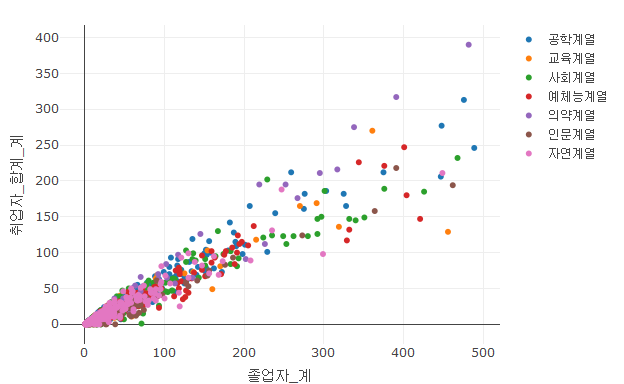
hovertext는 X, Y 좌표에 표시되는 문자열을 설정하는 type이다. hovertext는 앞서 설명한 hoverinfo를 text로 설정하고 hovertext를 설정할 때는 호버 상자 안에 text설정값이 표시되지만 hoverinfo를 생략하고 hovertext만 설정하면 호버 상자밖에 표시된다는 차이가 있다. hovertext에 단순 문자열을 설정하면 모든 데이터 호버에 동일한 문자열이 표기되지만 벡터를 설정하면 각각의 호버에 매핑된 결과가 표시된다.

df\_취업통계\_2000 |>   
 plot\_ly(x = ~졸업자\_계, y = ~취업자\_합계\_계, name = ~대계열,  
 hoverinfo = 'x+y+text', hovertext = ~중계열)



hovertemplete는 호버 상자와 호버 상자에 표시되는 정보의 포맷을 설정하는 type이다. 이 type은 앞서 설명한 hoverinfo에 설정된 사용되는 type에 사용된 변수를 %{변수}의 형태로 사용할 수 있다. 예를 들어 Y축 값을 표현하기 위해서는 %{y}로 설정한다. 앞서 사용된 예에서 X축의 값에 졸업자, Y축의 값에 취업자를 표기하고 값을 표시하는 코드는 다음과 같다.

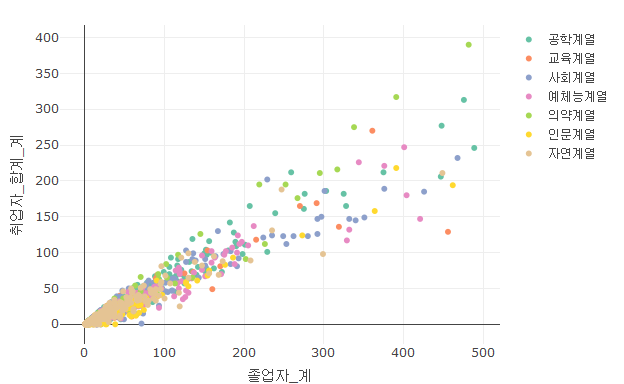
df\_취업통계\_2000 |>   
 plot\_ly(x = ~졸업자\_계, y = ~취업자\_합계\_계, name = ~대계열,   
 hoverinfo = 'x+y+text', hovertemplate = ' 졸업자:%{x}, 취업자:%{y}')



### color, colors

color는 스캐터 trace에 표현되는 점, 선, 문자의 내부 색을 설정하는 속성이다. 내부 색을 설정할 때는 먼저 색을 변수에 매핑할지, 특정 색상으로 설정할 지를 결정해야 한다. 이를 설정하는 매개변수가 color이다. color에 변수를 ~를 사용하여 매핑하면 해당 변수의 값에 따라 색이 매핑되어 표현된다.

df\_취업통계\_2000 |>   
 plot\_ly(x = ~졸업자\_계, y = ~취업자\_합계\_계, color = ~대계열)

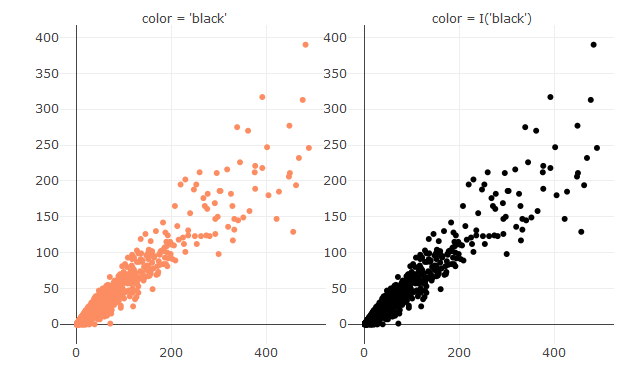


반면 특정한 색으로 설정할 때는 color에 특정 색 이름을 설정하면 모든 marker가 동일한 색으로 나타난다. 이 과정에서 하나 주의해야 할 것이 설정의 방법이다. 일반적인 변수 설정과 같이 색상명을 설정하면 다음과 같이 정확한 색상이 나타나지 않는다.

df\_취업통계\_2000 |>   
 plot\_ly(x = ~졸업자\_계, y = ~취업자\_합계\_계, color = 'black')

plotly에서 색의 사용은 기본적으로 매핑을 전제로 사용된다. 따라서 앞의 코드에서 color = 'black'으로 설정하는 것은 색을 검정색으로 지정하는 것이 아니고 ‘black’이라는 이름으로 매핑된 색을 불러온다는 의미이다. 하지만 미리 매핑되어 정의된 ’black’ 색 배열이 없기 때문에 plotly의 디폴트 색 팔레트를 사용하여 색이 설정된다. 자신이 원하는 색을 직접 설정하기 위해서는 ’asis’를 의미하는 I()를 사용하여 색을 설정하여야 한다.

df\_취업통계\_2000 |>   
 plot\_ly(x = ~졸업자\_계, y = ~취업자\_합계\_계, color = I('black'))



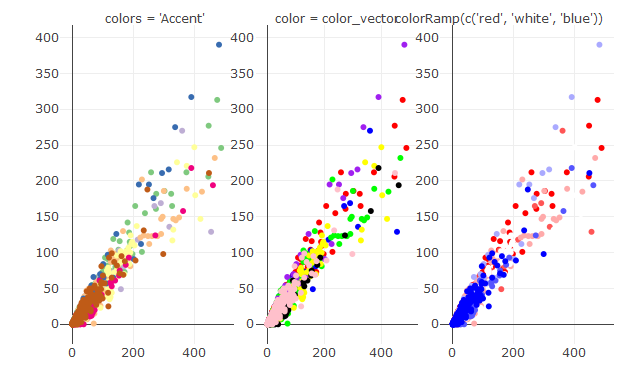
colors는 color에서 매핑된 변수에 따른 색의 스케일을 설정하는 매개변수이다. plotly에서는 색의 스케일을 설정하는 팔레트를 선정할 때 다음의 세 가지 방법을 사용한다.

첫 번째 방법은 RColorBrewer 패키지에서 제공하는 팔레트의 이름을 설정하는 방법이다. RColorBrewer 패키지는 R에서 가장 대중적으로 사용되는 색 팔레트를 제공하는 패키지로 ggplot2에서도 많이 사용된다. 이 패키지에서 제공하는 팔레트의 이름을 colors에 지정함으로써 해당 팔레트를 사용할 수 있다.

두 번째 방법은 사용할 색을 직접 지정하는 방법이다. 사용할 색의 이름을 가지는 문자열 벡터를 사용하여 직접 색을 지정한다.

세 번째는 colorRamp()나 scales::colour\_ramp()와 같은 색 보간 함수를 사용하는 방법이다. colorRamp()는 매개변수로 전달되는 색 벡터의 사이 색을 반환하는 함수를 만들어주는데 0부터 1까지의 값 범위내에 해당하는 색을 반환해준다.

df\_취업통계\_2000 |>   
 plot\_ly(x = ~졸업자\_계, y = ~취업자\_합계\_계, color = ~대계열, colors = 'Accent')  
  
color\_vector <- c('red', 'blue', 'green', 'yellow', 'purple', 'black', 'pink')  
  
df\_취업통계\_2000 |>   
 plot\_ly(x = ~졸업자\_계, y = ~취업자\_합계\_계,   
 color = ~대계열, colors = color\_vector)  
  
df\_취업통계\_2000 |>   
 plot\_ly(x = ~졸업자\_계, y = ~취업자\_합계\_계,   
 color = ~대계열, colors = colorRamp(c('red', 'white', 'blue')))

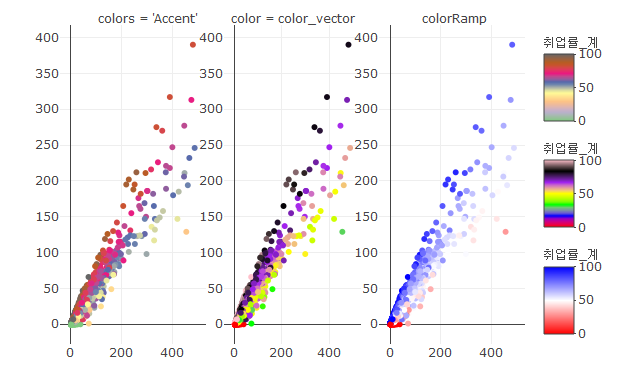


색 설정에서 하나 중요한 type은 colorbar()이다. color type에 매핑되는 변수가 이산형일 경우는 각각의 카테고리에 따라 설정된 색 팔레트에 정의된 색상이 설정되지만 연속형일 경우는 연속된 색상이 설정된다. 연속된 색상에서 변수에 따른 색상이 선택되어 표시된다. 이렇게 연속된 색상의 설정에 사용되는 색 스케일이 colorbar()이다.

colorbar(p, …, limits = NULL, which = 1)  
- p : plot\_ly()로 생성한 plotly 객체  
- … : 컬러바 설정을 위한 세부 속성  
- limits : 컬러바 범위 설정을 위한 수치 벡터  
- which : 다중 컬러바가 생성된 경우 컬러바 선택

컬러바 설정에 사용되는 세부 속성은 현재(22.03) 43개가 제공된다. 이에 대한 세부 속성과 설명은 plotly 매뉴얼을 참조하라.[[2]](#footnote-27)

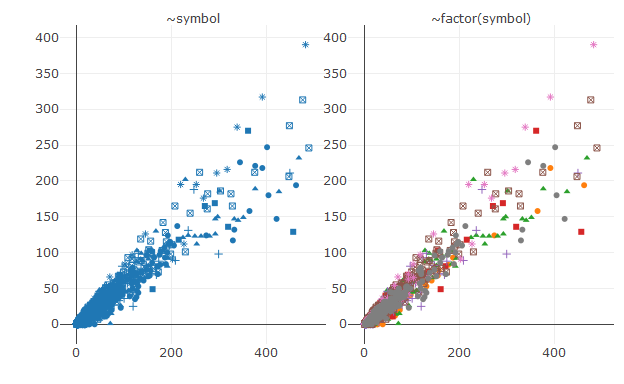
df\_취업통계\_2000 |>   
 plot\_ly(x = ~졸업자\_계, y = ~취업자\_합계\_계,   
 color = ~취업률\_계, colors = 'Accent')  
  
color\_vector <- c('red', 'blue', 'green', 'yellow', 'purple', 'black', 'pink')  
  
df\_취업통계\_2000 |>   
 plot\_ly(x = ~졸업자\_계, y = ~취업자\_합계\_계,   
 color = ~취업률\_계, colors = color\_vector)  
  
df\_취업통계\_2000 |>   
 plot\_ly(x = ~졸업자\_계, y = ~취업자\_합계\_계,   
 color = ~취업률\_계, colors = colorRamp(c('red', 'white', 'blue')))



### symbol, symbols

symbol은 점의 형태를 설정하는 속성이다. 앞서 color와 같이 symbol도 매핑할 변수를 설정하는 매개변수이고 symbols는 변수 카테고리에 따라 설정하는 점의 형태를 설정하는 매개변수이다. color와 다른 점은 연속형 변수에 매핑되는 symbol은 단 하나의 카테고리로 분류된다는 점이다. 그렇기 때문에 좌표에 표시되는 점의 형태는 다르게 보이지만 범례에 표현되지는 않는다는 점이다. 다음의 예를 살펴보자.

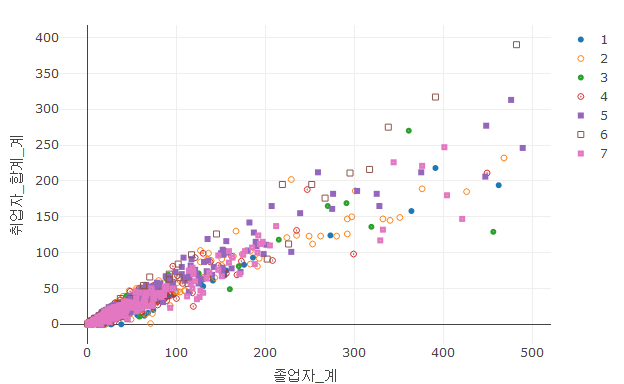
df\_취업통계\_symbol <- df\_취업통계\_2000 |>   
 mutate(symbol = case\_when(  
 대계열 == '인문계열' ~ 1,   
 대계열 == '사회계열' ~ 2,  
 대계열 == '교육계열' ~ 3,  
 대계열 == '자연계열' ~ 4,  
 대계열 == '공학계열' ~ 5,  
 대계열 == '의약계열' ~ 6,  
 대계열 == '예체능계열' ~ 7)  
 )   
  
df\_취업통계\_symbol |>   
 plot\_ly(x = ~졸업자\_계, y = ~취업자\_합계\_계, symbol = ~symbol)  
  
df\_취업통계\_symbol |>   
 plot\_ly(x = ~졸업자\_계, y = ~취업자\_합계\_계, symbol = ~factor(symbol))



앞의 예에서 ‘symbol’ 열은 대계열을 수치로 표현한 열로 수치형 열이다. 수치형 열이기 때문에 연속형 변수로 취급되고 이 열을 변수로 바로 사용하면 왼쪽과 같이 점의 모형은 달리지지만 하나의 trace로 표현된다. 반면 오른쪽과 같이 ‘symbol’ 열을 factor()를 사용하여 팩터로 전환하면 이산형 변수가 되기 때문에 각각의 카테고리를 trace로 구분하여 추가되기 때문에 색까지 구분해주며 범례에서 각각의 trace로 표현된다.

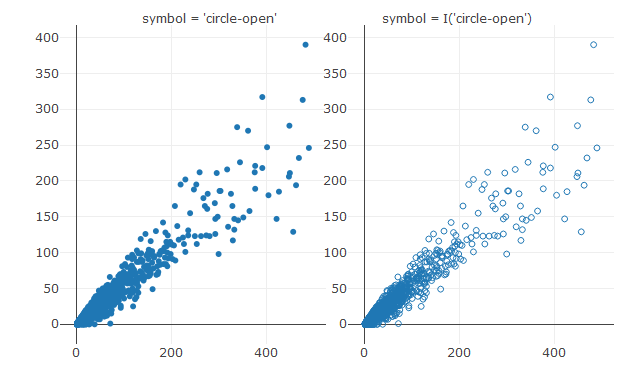
plotly에서는 0번부터 52번까지 총 53개의 symbol을 제공한다. 이 기본 53개의 도형 번호에 100을 더하면 내부가 빈 ’open’형 심볼, 200을 더하면 점이 찍힌 ’dot’형 심볼, 300을 더하면 내부가 비고 점이 찍힌 ’open-dot’형 심볼을 의미한다.

df\_취업통계\_symbol |>   
 plot\_ly(x = ~졸업자\_계, y = ~취업자\_합계\_계,   
 symbol = ~factor(symbol), symbols = c('circle', 'circle-open', 'circle-dot', "circle-open-dot", "square", "square-open", "square-dot"))



symbol에 변수를 매핑하지 않고 특정 심볼을 설정할 경우는 색의 설정과 같이 I()를 사용한다.

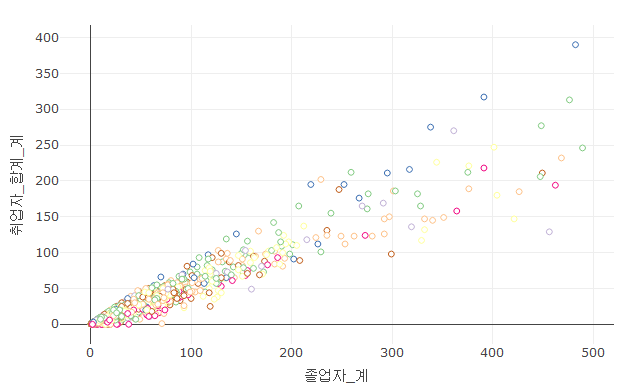
df\_취업통계\_symbol |>   
 plot\_ly(x = ~졸업자\_계, y = ~취업자\_합계\_계, symbol = 'circle-open')  
  
df\_취업통계\_symbol |>   
 plot\_ly(x = ~졸업자\_계, y = ~취업자\_합계\_계, symbol = I('circle-open'))



### stroke, strokes

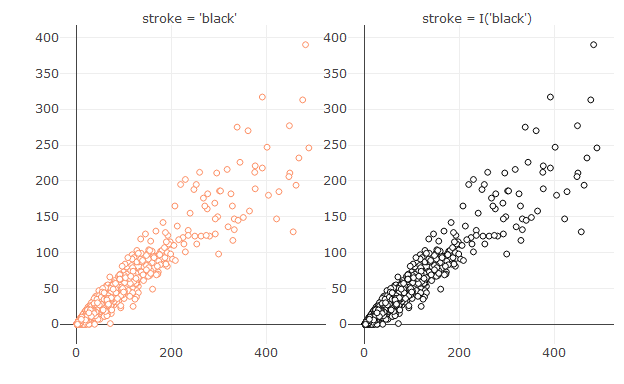
stroke는 점의 외곽선 색을 설정하는 속성이다. color의 설정은 점의 내부색과 외곽선 색을 동시에 설정하지만 외곽선 색을 따로 매핑해야할 경우에 stroke를 사용하여 매핑하거나 설정할 수 있다. stroke와 strokes는 color나 symbol과 같이 변수 매핑으로 사용되는 매개변수는 stroke이고 매핑된 변수의 카테고리에 해당하는 외곽선 색을 설정하는 매개변수는 strokes이다. strokes에 색을 설정하는 방법은 colors에서 사용한 세 가지 방법을 같이 사용할 수 있다 .

df\_취업통계\_2000 |>   
 plot\_ly(x = ~졸업자\_계, y = ~취업자\_합계\_계,   
 stroke = ~대계열, strokes = 'Accent', color = I('white'))



stroke를 특정 값으로 설정하기 위해서는 I()를 사용한다.

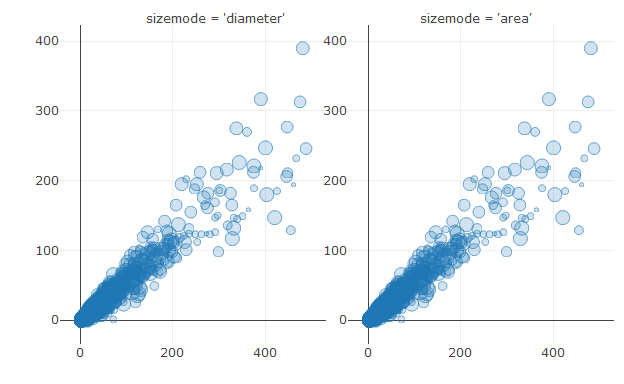
df\_취업통계\_2000 |>   
 plot\_ly(x = ~졸업자\_계, y = ~취업자\_합계\_계,   
 stroke = 'black', color = I('white'))  
  
df\_취업통계\_2000 |>   
 plot\_ly(x = ~졸업자\_계, y = ~취업자\_합계\_계,   
 stroke = I('black'), color = I('white'))



### size, sizes

size는 점의 크기를 설정하는 속성이다. size는 변수 매핑을 위한 매개변수이고 sizes는 변수에 매핑된 카테고리별로 크기를 설정하는 매개변수이다. 앞선 속성들과 달리 size는 sizemode 매개변수를 추가적으로 사용할 수 있다. sizemode는 점의 크기를 결정하는 기준을 설정하는 매개변수로 점의 크기를 지름으로 설정하는 ’diameter’와 면적으로 설정하는 ’area’의 두 가지가 있다.

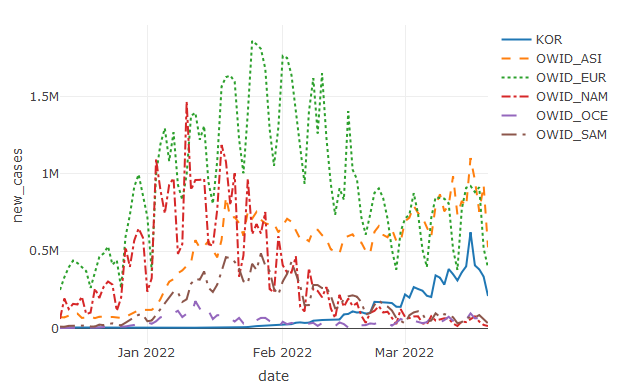
df\_취업통계\_symbol |>   
 plot\_ly(x = ~졸업자\_계, y = ~취업자\_합계\_계,   
 size = ~symbol, sizemode = 'diameter', alpha = 0.3)  
  
df\_취업통계\_symbol |>   
 plot\_ly(x = ~졸업자\_계, y = ~취업자\_합계\_계,   
 size = ~symbol, sizemode = 'area', alpha = 0.3)



### linetype, linetypes

linetype은 선의 형태를 설정하는 type이다. plotly에서 선의 형태를 설정할 때 변수 매핑을 통해 설정하기 위해서는 linetype을 사용하고 linetype에 매핑된 변수에 따라 선의 형태를 설정하기 위해서는 linetypes를 사용한다.

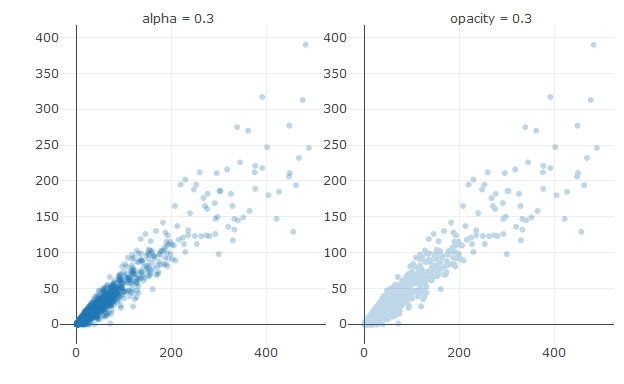
covid19\_df\_100 |>  
 plot\_ly(type = 'scatter', mode = 'lines', x = ~date, y = ~new\_cases, linetype = ~iso\_code)



### alpha, opacity

alpha와 opacity는 점의 투명도를 설정하는 속성이다. alpha와 opacity는 0부터 1까지의 수치로 설정하며 0은 완전 투명으로 보이지 않고 1은 불투명한 점을 표현하는데 alpha는 외곽선에 투명도가 적용되지 ㅇ낳고 서로 겹쳐진 부분의 투명도가 양쪽의 투명도가 더해져서 점점 불투명해지는데 반해 opacity는 외곽선에도 투명도가 적용되고 겹쳐진 부분도 처음 설정된 투명도가 유지된다는 차이가 있다. 이 두 type은 산점도의 표현시 오버플로팅을 방지하고 색으로 구분되는 데이터 그룹간의 표현에 효과적으로 사용된다.

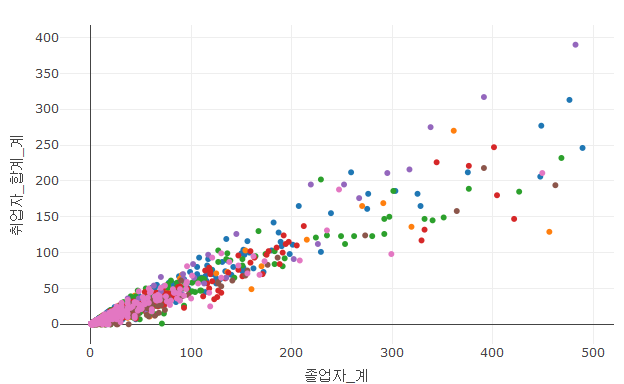
df\_취업통계\_2000 |>   
 plot\_ly(x = ~졸업자\_계, y = ~취업자\_합계\_계, alpha = 0.3)  
  
df\_취업통계\_2000 |>   
 plot\_ly(x = ~졸업자\_계, y = ~취업자\_합계\_계, opacity = 0.3)



### showlegend

showlegend는 범례를 표기할지 여부를 설정하는 논리값(TRUE/FALSE)이다. FALSE로 설정할 경우 범례가 표기되지 않는다.

df\_취업통계\_2000 |>   
 plot\_ly(x = ~졸업자\_계, y = ~취업자\_합계\_계, name = ~대계열, showlegend = FALSE)



## trace 추가 : add\_trace()

trace는 plotly 객체에서 표현되는 단일 데이터 아이템을 의미한다. 이를 간단히 이야기하자면 범례에 표시되는 데이터 아이템 하나 하나가 각각의 개별 trace라고 생각하면 편할 것이다. ggplot2에서는 geom\_\*()을 사용하여 점, 선, 막대 등의 기하 요소로 표현하였지만 plotly 객체에서는 trace라는 이름으로 각각의 데이터 표현을 추가한다. 하지만 ggplot2에 익숙한 사용자는 이 trace의 개념과 ggplot2의 기하 요소와의 차이를 잘 이해하여야 한다. ggplot2에서 데이터를 표현하는 기하 요소는 변수 매핑을 통해 하나의 geom\_\*()으로 생성하는 것이 일반적이다. 예를 들어 ggplot2를 사용하여 막대 그래프나 박스 플롯을 생성할 때 여러 개의 geom\_bar()나 geom\_boxplot()를 사용하지 않고 aes()에 변수 매핑을 통해 한 번의 함수 호출로 생성한다. 이러한 형태에 사용하기 위해 데이터프레임을 긴 형태의 데이터로 전환하여 사용하였다. 여기에 예외로 사용되는 것이 geom\_line()인데 넓은 형태의 데이터프레임에 각 열을 geom\_line()을 사용하여 각각의 선 레이어로 생성하고 이들을 계속 겹쳐 그림으로써 전체적인 선 그래프를 완성할 수 있다. 하지만 plotly에서는 이 두가지 방법을 모든 시각화 방법에 동일하게 지원한다. 따라서 plotly를 사용할 때 가장 큰 장점이 넓은 형태의 데이터프레임과 긴 형태의 데이터프레임을 서로 변환하지 않고 사용할 수 있다는 점이다. 긴 형태의 데이터프레임이라면 데이터를 그룹화할 변수를 매핑함으로서 시각화가 생성된다. 예를 들어 전문대학, 일반대학, 석사, 박사의 네 가지 데이터에 대한 막대 그래프를 생성한다고 가정할 때, trace를 추가하는 함수인 add\_trace()를 한번 호출하여 네개의 막대가 생성된 막대 그래프가 하나가 생성되더라도 trace는 네개가 존재한다는 것이다. 반면 넓은 형태의 데이터프레임이라면 시각화 해야할 각각의 열을 각각 trace로 추가하여 줌으로써 시각화를 완성한다. ggplot2의 선 그래프에 가능한 방법을 모든 시각화에서 사용할 수 있다는 것이다.

plotly에서는 trace를 표현하는데 40가지 이상의 시각화 type을 제공한다. 또 각각의 trace의 시각화 type에는 그 에는 해당 trace의 시각화 세부 속성을 설정을 위한 수많은 세부 type을 가지고 있다. 앞서 plot\_ly()에서 매개변수로 설정했던 type들은 모두 개별 trace의 설정으로도 사용이 가능하다. 하지만 plot\_ly()에서 trace의 type을 설정하지 않았는데 시각화의 결과는 데이터들이 trace로 표현되어 나타났다. 이는 plotly에서 데이터를 분석하여 가장 효과적인 trace를 매칭해주었다. 하지만 trace는 사용자가 지정하는 것이 일반적이다.

이 trace와 type을 설정하는 방법은 add\_trace()를 사용하여 trace와 trace에 속한 type 속성을 설정하는 방법과 add\_markers(), add\_lines()등과 같이 add\_trace()에서 파생된 래핑 함수를 사용하는 방법이 있다.

add\_scatter()의 주요 사용법은 다음과 같다.

add\_trace(p, …, data = NULL, inherit = TRUE)  
- p : plot\_ly()로 생성한 plotly 객체  
- … : 스캐터 trace에 설정할 수 있는 속성 설정  
- data : 시각화할 데이터프레임  
- inherit : plot\_ly()에 설정된 속성 type을 상속할지를 결정하는 논리값

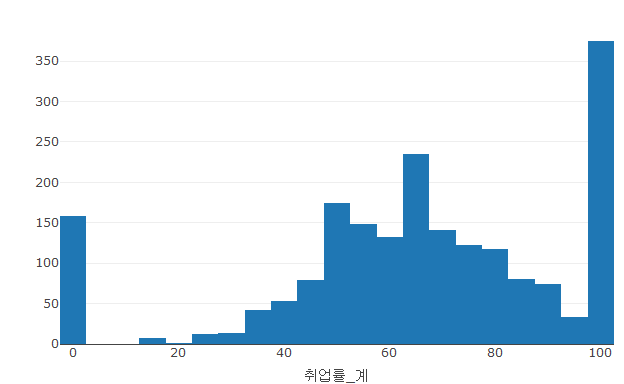
trace의 추가를 설명하기 전에 먼저 공통적으로

### 히스토그램(histogram) trace

plotly에서 히스토그램 시각화를 위한 trace type은 ’histogram’이다. 히스토그램은 하나의 변수에 따른 그 사례 빈도를 산출하는 시각화이고 X축에 매핑되는 변수외에 간격(bin)을 설정함으로써 히스토그램을 생성할 수 있다. 히스토그램 trace는 add\_trace()에 type을 ’histogram’으로 설정하거나 add\_histogram()을 사용하여 추가할 수 있다.

add\_trace(p, type = ‘histogram’, …, data = NULL, inherit = TRUE)  
add\_histogram(p, x = NULL, y = NULL, …, data = NULL, inherit = TRUE)  
- p : plot\_ly()로 생성한 plotly 객체  
- … : 히스토그램 trace에 설정할 수 있는 속성 설정  
- data : 시각화할 데이터프레임  
- inherit : plot\_ly()에 설정된 속성 type을 상속할지를 결정하는 논리값  
- x : X축에 매핑할 변수를 ~로 설정  
- y : Y축에 매핑할 변수를 ~로 설정

p\_histogram <- df\_취업통계\_2000 |> plot\_ly()  
  
p\_histogram |> add\_trace(type = 'histogram', x = ~취업률\_계, name = '취업률')



plotly의 히스토그램 trace에서 사용되는 주요 type들은 plot\_ly()에서 사용되는 대부분의 type을 사용할 수 있다. 그 외에 히스토그램에서 고유하게 사용되는 type은 다음과 같다.

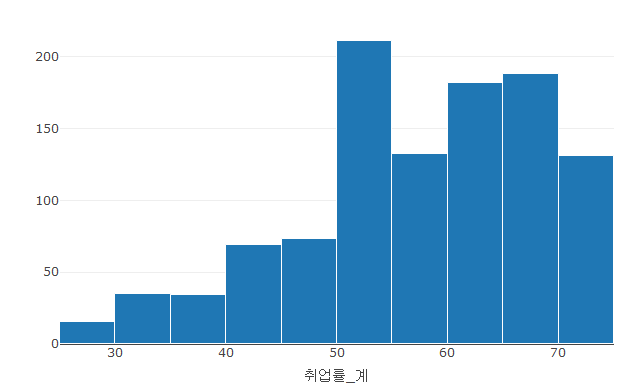
#### xbins

xbins는 X축에 매핑된 변수가 연속형 수치 변수일 경우 빈도를 표현할 구간의 범위를 설정하는 type이다. xbins는 바로 수치나 벡터를 설정하는 것이 아니고 xbins에 속하는 하위 세부 type의 설정을 list()로 묶어 정의한다. xbins에서 정의하는 세부 type의 종류는 size, start, end이다.

size는 히스토그램의 막대의 X축 범위를 지정하는 type이다. 예를 들어 X축의 수치형 변수 범위가 0부터 100까지라고 가정할때 size를 5으로 설정하면 막대는 총 20개가 생성된다.

start와 end는 히스토그램에서 표현하는 전체 X축의 범위를 지정하는 type이다. 만약 X축의 수치형 변수 범위가 0부터 100까지이라고 하더라도 25에서 75까지의 히스토그램을 생성하기를 원한다면 start와 end로 범위를 설정할 수 있다.

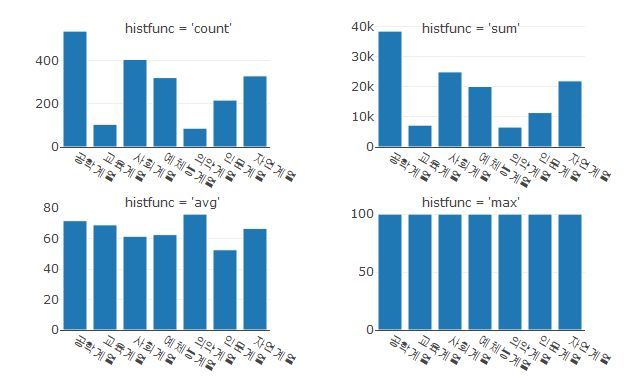
p\_histogram |> add\_trace(type = 'histogram', x = ~취업률\_계,   
 xbins = list(start = 25, end = 75, size = 5),   
 stroke = I('white'))



#### histfunc

histfunc은 히스토그램의 막대로 표현하는 값의 종류를 설정하는 type이다. 일반적으로 히스토그램은 사례수(‘count’)를 사용하지만 plotly에서는 사례수를 포함하여 합계(‘sum’), 평균(‘avg’), 최소(‘min’), 최대(‘max’)를 제공한다. 이를 histfunc으로 설정하면 자동적으로 통계 처리되어 히스토그램이 생성된다. 하나 주의해야 할 것은 histfunc에 사용되는 ‘sum’, ‘avg’, ‘max’, ’min’을 사용할 경우 해당 통계 변환에 사용할 변수열을 y type에 설정해야하고 이 변수열은 수치형 데이터가 아닌 문자형 데이터로 설정해야 한다.

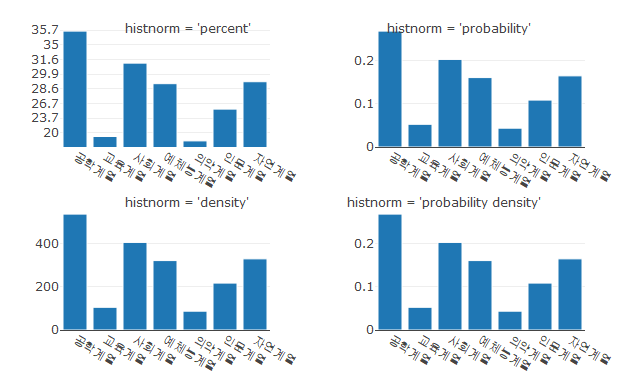
p\_histogram |> add\_trace(type = 'histogram', x = ~대계열,   
 stroke = I('white'), histfunc = 'count')  
  
p\_histogram |> add\_trace(type = 'histogram', x = ~대계열, y = ~as.character(취업률\_계),   
 histfunc = 'sum', stroke = I('white')) |>  
 layout(yaxis=list(type='linear'))  
  
p\_histogram |> add\_trace(type = 'histogram', x = ~대계열, y = ~as.character(취업률\_계),   
 histfunc = 'avg', stroke = I('white')) |>  
 layout(yaxis=list(type='linear'))  
  
p\_histogram |> add\_trace(type = 'histogram', x = ~대계열, y = ~as.character(취업률\_계),   
 histfunc = 'max', stroke = I('white')) |>  
 layout(yaxis=list(type='linear'))



#### histnorm

histnorm은 히스토그램 trace로 표시되는 데이터의 정규화(normalization)에 대한 type이다. histnorm으로 설정이 가능한 정규화는 ’‘, ’percent’, ‘probability’, ‘density’, ‘probability density’의 다섯 가지이다.’’은 특별한 정규화없이 사례수를 사용하고 ’percent’는 전체 중의 백분율, ’probability’는 전체의 비율을 나타닌다. ’density’는 빈의 발생 빈도에 따른 밀도를 표현하고, ’probability density’은 각 막대 구간에 데이터가 존재할 밀도 확률을 면적으로 변환하여 나타내기 때문에 모든 막대 면적의 합은 1이 된다.

p\_histogram |> add\_trace(type = 'histogram', x = ~대계열, y = ~as.character(취업률\_계),   
 stroke = I('white'), histnorm = 'percent')  
  
p\_histogram |> add\_trace(type = 'histogram', x = ~대계열, y = ~as.character(취업률\_계),   
 histnorm = 'probability', stroke = I('white')) |>  
 layout(yaxis=list(type='linear'))  
  
p\_histogram |> add\_trace(type = 'histogram', x = ~대계열, y = ~as.character(취업률\_계),   
 histnorm = 'density', stroke = I('white')) |>  
 layout(yaxis=list(type='linear'))  
  
p\_histogram |> add\_trace(type = 'histogram', x = ~대계열, y = ~as.character(취업률\_계),   
 histnorm = 'probability density', stroke = I('white')) |>  
 layout(yaxis=list(type='linear'))



### 스캐터(scatter) trace : 산점도, 선 그래프

스캐터 trace는 점, 선, 문자를 X, Y 축 좌표의 위치를 사용해 시각화를 하는 형태의 trace를 모두 말한다. 따라서 스캐터 trace를 통해 생성 가능한 시각화는 산점도, 선 그래프, 텍스트 차트, 풍선 차트 등이다. 이 스캐터 trace는 add\_trace()의 type 매개변수에 ’scatter’를 설정함으로써 사용할 수 있고 add\_\*() 함수(add\_markers(), add\_lines(), add\_paths(), add\_segments(), add\_ribbons())를 사용할 수 있다.

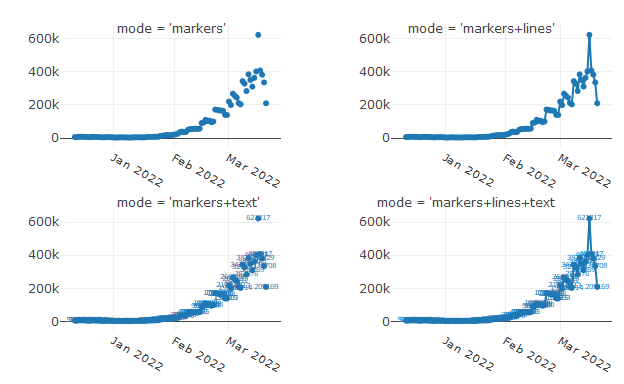
스캐터 trace에서 약 50여개가 넘는 type을 사용하여 세부 설정이 가능하다. 이 50여개의 type 중에는 다시 세부 type을 설정하는 type이 있고 총 4단계까지 내려가는 type도 존재한다. 세부 하위 type은 list()를 사용하여 세부 타입을 설정할 수 있다. 이 모든 type을 다 설명할 수는 없기 때문에 주요한 type을 위주로 설명한다. 전체 type은 plotly.com에 수록되어 있다.[[3]](#footnote-41)

#### mode

mode는 스캐터 trace에서 실제 데이터를 표현하는 방법을 지정하는 type이다. 사실 add\_trace()에서 type과 mode는 시각화의 종류를 지정하는 가장 중요한 type 설정이다. 스캐터 trace에서의 mode는 다음의 네 가지가 있다.

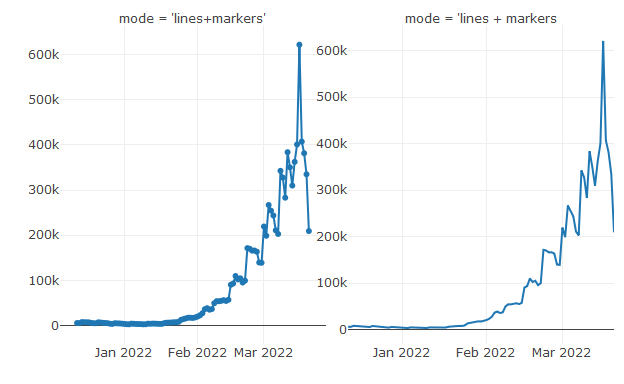
| mode | 설명 |
| --- | --- |
| markers | 데이터를 점으로 표시 |
| lines | 데이터를 서로 이어주는 선을 표시 |
| text | 데이터를 문자열로 표시 |
| none | 데이터를 표시하지 않음 |

p\_line\_wide <- covid19\_df\_100\_wide |> plot\_ly()  
  
p\_line\_wide |> add\_trace(type = 'scatter', mode = 'markers', x = ~date, y = ~South\_Korea)  
  
p\_line\_wide |> add\_trace(type = 'scatter', mode = 'markers+lines', x = ~date, y = ~South\_Korea)  
  
p\_line\_wide |> add\_trace(type = 'scatter', mode = 'markers+text', x = ~date, y = ~South\_Korea, text = ~South\_Korea)  
  
p\_line\_wide |> add\_trace(type = 'scatter', mode = 'markers+lines+text', x = ~date, y = ~South\_Korea, text = ~South\_Korea)



이 네가지 mode는 단일 모드로 사용이 가능하지만 ’none’을 제외하고 +기호를 사용하여 연결하여 사용하는데 최대 3개까지 연결하여 사용할 수 있다. 여기서 하나 주의해야하는 것은 + 기호 앞뒤에 공백이 있어서는 안된다.

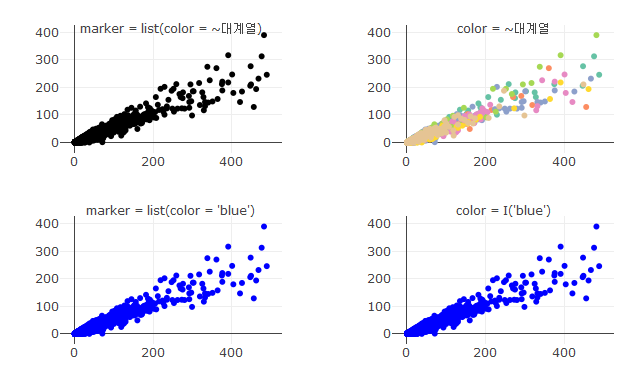
p\_line\_wide |> add\_trace(type = 'scatter', mode = 'markers+lines', x = ~date, y = ~South\_Korea)  
  
p\_line\_wide |> add\_trace(type = 'scatter', mode = 'markers + lines', x = ~date, y = ~South\_Korea)



#### marker, add\_markers()

스캐터 trace의 marker 타입은 스캐터 trace에서 데이터를 표현하는 점의 속성을 설정하는 type이다. marker는 add\_trace()에 list()를 사용하여 설정한다. 주의할 것은 marker의 list에 설정하는 type들은 값을 설정할 수는 있지만 ~를 사용해 변수를 매핑할 수 없다는 점이다. marker list에 ~를 사용한 변수 매핑을 사용한다고 해도 에러를 내지는 않지만 원하는 결과를 얻을 수는 없다. 또 하나 주의해아 하는 점은 다른 type의 설정시 사용했던 I()의 사용이다. 앞서 색 type과 같은 일부 type의 설정시에 I()를 사용한다고 설명했는데 marker로 설정하는 list안에서는 I()를 사용하지 않아도 색의 설정이 가능하다. marker list는 변수의 매핑에 사용하는 것이 아니고 값을 설정에 사용하는 것이기 때문에 구지 I()를 사용하여 매핑과 설정을 구분해 줄 필요가 없다는 것이다.

p\_marker <- df\_취업통계\_2000 |> plot\_ly()  
  
p\_marker |>  
 add\_trace(type = 'scatter', mode = 'markers', x = ~졸업자\_계, y = ~취업자\_합계\_계,  
 marker = list(color = ~대계열))  
  
p\_marker |>  
 add\_trace(type = 'scatter', mode = 'markers', x = ~졸업자\_계, y = ~취업자\_합계\_계,  
 color = ~대계열)  
  
p\_marker |>  
 add\_trace(type = 'scatter', mode = 'markers', x = ~졸업자\_계, y = ~취업자\_합계\_계,  
 marker = list(color = 'blue'))  
  
p\_marker |>  
 add\_trace(type = 'scatter', mode = 'markers', x = ~졸업자\_계, y = ~취업자\_합계\_계,  
 color = I('blue'))



add\_trace(p, type = ‘sactter’, mode = ‘markers’, marker = NULL, …, data = NULL, inherit = TRUE)  
add\_markers(p, x = NULL, y = NULL, z = NULL, …, data = NULL, inherit = TRUE)  
- p : plot\_ly()로 생성한 plotly 객체  
- type : trace 타입을 설정, add\_markers()는 ‘scatter’로 설정  
- mode : ’scatter’ trace 중 어떤 도형을 사용할지 설정, add\_markers()는 ’markers’를 설정  
- marker : marker의 type 설정을 위한 매개변수 list 설정  
- … : 스캐터 trace의 markers 모드에 설정할 수 있는 속성 설정  
- data : 시각화할 데이터프레임  
- inherit : plot\_ly()에 설정된 속성 type을 상속할지를 결정하는 논리값  
- x : X축에 매핑할 변수를 ~로 설정  
- y : Y축에 매핑할 변수를 ~로 설정  
- z : Z축에 매핑할 변수를 ~로 설정

marker 에서 설정 가능한 세부 type은 20여개가 있지만 이 세부 type의 세부 type까지 포함하면 그 종류가 매우 많아진다. add\_markers()를 사용한다면 세부 type 설정을 바로 사용할 수 있지만 add\_trace()를 사용하면 marker 매개변수에 세부 type을 list()로 설정하여 지정할 수 있다.

#### line, add\_lines()

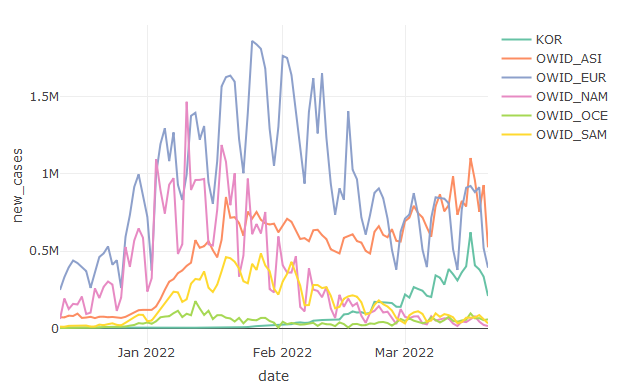
스캐터 trace의 line 타입은 스캐터 trace에서 데이터를을 이어주는 선의 속성을 설정하는 type이다. line은 marker와 같이 add\_trace()에 list()를 사용하여 설정한다. 마찬가지로 주의할 것은 line의 list에 설정하는 type들은 값을 설정할 수는 있지만 ~를 사용해 변수를 매핑할 수 없다는 점이고 I()를 사용하지 않고 값의 설정이 가능하다는 것이다.

add\_lines()는 add\_trace(type = 'scatter', mode = 'line')의 래핑 함수로 line으로 설정이 가능한 type과 스캐터 type을 모두 정의할 수 있는데 add\_trace()와 같이 line list를 사용하지 않아도 되고 일부 type의 설정에 I()를 사용한다는 점에 주의하여야 한다.

add\_trace(p, type = ‘sactter’, mode = ‘lines’, line = NULL, …, data = NULL, inherit = TRUE)  
add\_lines(p, x = NULL, y = NULL, z = NULL, …, data = NULL, inherit = TRUE)  
- p : plot\_ly()로 생성한 plotly 객체  
- type : trace 타입을 설정, add\_markers()는 ‘scatter’로 설정 - mode : ’scatter’ trace 중 어떤 도형을 사용할지 설정, add\_markers()는 ’markers’를 설정  
- line : line의 type 설정을 위한 매개변수 list 설정  
- … : 스캐터 trace의 line 모드에 설정할 수 있는 속성 설정  
- data : 시각화할 데이터프레임  
- inherit : plot\_ly()에 설정된 속성 type을 상속할지를 결정하는 논리값  
- x : X축에 매핑할 변수를 ~로 설정  
- y : Y축에 매핑할 변수를 ~로 설정  
- z : Z축에 매핑할 변수를 ~로 설정

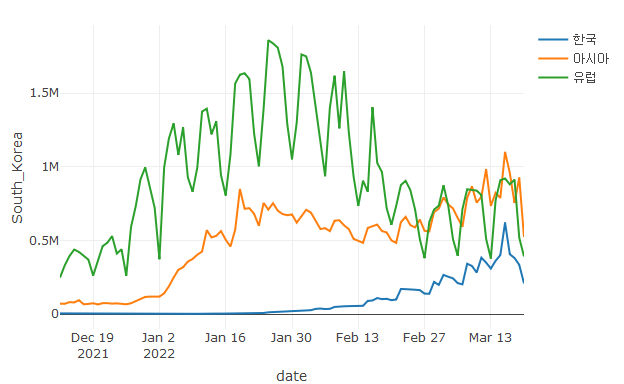
trace의 type과 mode를 선그래프에 맞에 설정하거나 add\_lines()로 선그래프를 만들려면 먼저 X, Y, Z 축으로 매핑할 변수를 설정한다.선 그래프는 앞서 그려본 산점도와는 달리 데이터들을 선으로 연결해 줄 그룹 변수가 필요하다. 이는 일반적으로 color, linetype으로 매핑된 변수를 기준으로 그룹화하여 선을 연결한다.

p\_line <- covid19\_df\_100 |> plot\_ly()  
  
p\_line |>   
 add\_trace(type = 'scatter', mode = 'lines', x = ~date, y = ~new\_cases, color = ~ iso\_code)



선 그래프의 경우는 선을 너무 많이 그려서 꼬이기 시작하면 오히려 데이터를 확인하는데 방해가 되는 시각화가 되는 경우가 있다. 이를 스파게티 선 그래프라고 하는데 하나의 선 그래프에는 보통 5~6개 이상의 선을 넣지 않는 것이 일반적이다. 따라서 선 그래프의 경우는 변수 매핑에 의해 단번에 선 그래프를 생성하는 방법도 많이 사용하지만 3~4개의 단일 선 그래프를 겹쳐 그리는 방법도 많이 사용된다. 이렇게 여러 개의 trace를 겹쳐 그린다면 plotly 객체에 trace를 순차적으로 추가하여 생성할 수 있다.

fig <- add\_trace(p\_line\_wide, type = 'scatter', mode = 'lines', x = ~date, y = ~South\_Korea, name = '한국')  
  
fig <- fig |> add\_trace(type = 'scatter', mode = 'lines', x = ~date, y = ~Asia, name = '아시아')  
  
fig <- fig |> add\_trace(type = 'scatter', mode = 'lines', x = ~date, y = ~Europe, name = '유럽')  
  
fig



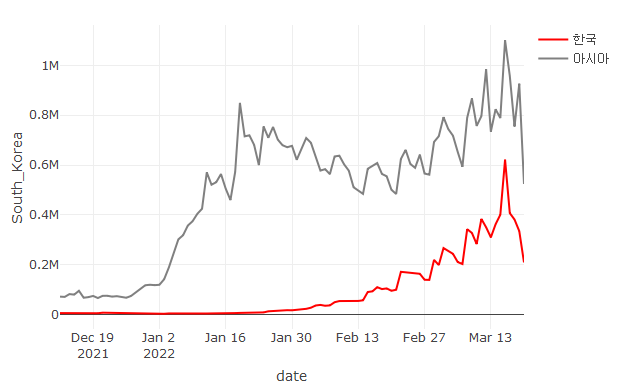
이렇게 추가적인 방법으로 시각화를 완성하면 몇 가지 장점이 있다.

첫 번쨰 장점은 특별한 작업 없이 범례의 순서를 결정할 수 있다. 최종적으로 나타나는 선 그래프의 범례는 trace가 추가된 순서대로 표시되기 때문에 범례 열을 팩터화하고 레벨을 설정하는 등의 작업이 필요없다.

두 번째는 각각의 데이터의 특성에 맞는 선 색과 선 타입 등을 설정할 수 있다. 선 타입의 경우는 특별히 설정하지 않으면 모두 실선으로 표시되지만 선 색의 경우는 설정된 색 팔레트에 의해 자동적으로 설정된다. 이를 바꾸기 위해서는 해당 trace를 추가할 때 설정해준다. 특히 이 작업은 여러 선 중에서 강조하고 싶은 선을 표시할 때 매우 효과적으로 사용이 가능하다.

세 번째는 사람이 인식하기 쉬운 넓은 형태의 데이터프레임을 그대로 사용할 수 있다는 점이다. 선 그래프의 선을 변수에 매핑하여 단번에 선 그래프를 완성할 때는 반드시 긴 형태의 데이터프레임으로 피봇하여 사용해야 하지만 긴 형태의 데이터프레임은 사람이 인식하기가 매우 어렵다. 하지만 사람이 인식하기 쉬운 넓은 형태의 데이터프레임을 사용하면 생성된 선 그래프와 데이터를 비교해 볼 수 있다는 장점이 있다.

fig <- add\_trace(p\_line\_wide, type = 'scatter', mode = 'lines',   
 x = ~date, y = ~South\_Korea, name = '한국',   
 line = list(color = 'red'))  
  
fig <- fig |> add\_trace(type = 'scatter', mode = 'lines',   
 x = ~date, y = ~Asia, name = '아시아',   
 line = list(color = 'gray'))  
  
fig |> layout(showlegend = TRUE)



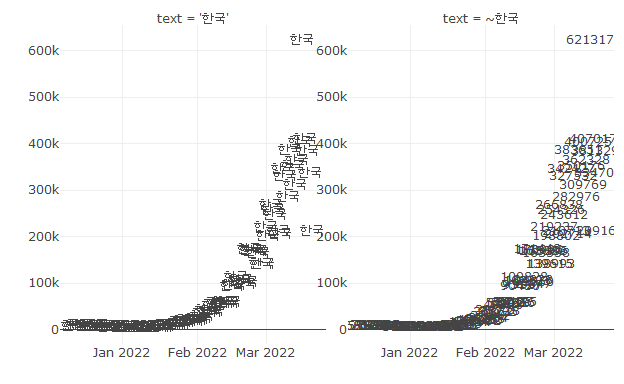
#### text, add\_text()

스캐터 trace의 text type은 각 X축과 Y축에 매핑된 좌표에 표시될 문자열을 설정하는 trace type이다. 이 trace는 add\_trace(type = 'scatter', mode = 'text'나 add\_text()를 사용하여 추가할 수 있다.

add\_trace(p, type = ‘sactter’, mode = ‘text’, text = NULL, …, data = NULL, inherit = TRUE)  
add\_lines(p, x = NULL, y = NULL, z = NULL, …, data = NULL, inherit = TRUE)  
- p : plot\_ly()로 생성한 plotly 객체  
- type : trace 타입을 설정, add\_markers()는 ‘scatter’로 설정 - mode : ’scatter’ trace 중 어떤 도형을 사용할지 설정, add\_markers()는 ’markers’를 설정  
- text : text로 매핑할 변수를 ~로 설정  
- … : 스캐터 trace의 line 모드에 설정할 수 있는 속성 설정  
- data : 시각화할 데이터프레임  
- inherit : plot\_ly()에 설정된 속성 type을 상속할지를 결정하는 논리값  
- x : X축에 매핑할 변수를 ~로 설정  
- y : Y축에 매핑할 변수를 ~로 설정  
- z : Z축에 매핑할 변수를 ~로 설정

text에 단일 문자열을 설정한 경우는 x, y 좌표위의 모든 데이터에 동일한 문자열이 표시된다. 반면 문자열 벡터를 설정하거나 문자열 변수를 매핑한 경우는 각각의 X, Y 좌표에 따라 해당 문자열이 표시된다. 여기에 하나 추가적으로 살펴야 하는 것은 해당 trace의 hoverinfo에 ’text’가 설정되고 hovertext가 설정되지 않은 경우에는 text에 설정된 문자열이 호버 레이블로 사용된다.

add\_trace(p\_line\_wide, type = 'scatter', mode = 'text',   
 x = ~date, y = ~South\_Korea, text = '한국', showlegend = FALSE)  
  
add\_trace(p\_line\_wide, type = 'scatter', mode = 'text',   
 x = ~date, y = ~South\_Korea, text = ~South\_Korea, showlegend = FALSE)



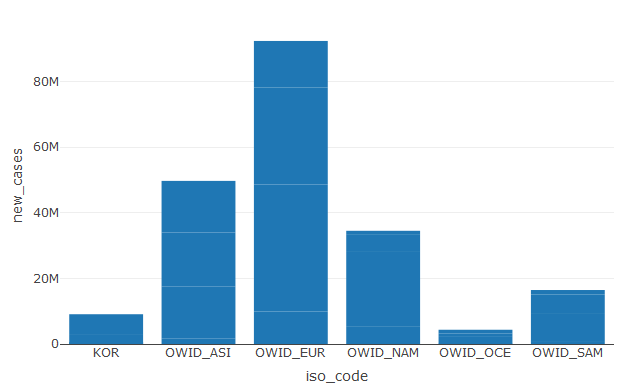
### 막대(bar) trace : 막대 그래프

막대 trace는 데이터 변량의 크기를 막대를 사용하여 시각화를 하는 형태의 trace를 말한다. 따라서 막대 trace를 통해 생성 가능한 시각화는 막대 그래프이다. 이 막대 trace는 add\_trace()의 type 매개변수에 ’bar’를 설정함으로써 사용할 수 있고 add\_bar() 함수를 사용할 수 있다.

add\_trace(p, type = ‘bar’, …, data = NULL, inherit = TRUE)  
add\_bars(p, x = NULL, y = NULL, …, data = NULL, inherit = TRUE)  
- p : plot\_ly()로 생성한 plotly 객체  
- type : trace 타입을 설정, add\_markers()는 ‘scatter’로 설정 - mode : ’scatter’ trace 중 어떤 도형을 사용할지 설정, add\_markers()는 ’markers’를 설정  
- text : text로 매핑할 변수를 ~로 설정  
- … : 막대 trace에 설정할 수 있는 속성 설정  
- data : 시각화할 데이터프레임  
- inherit : plot\_ly()에 설정된 속성 type을 상속할지를 결정하는 논리값  
- x : X축에 매핑할 변수를 ~로 설정  
- y : Y축에 매핑할 변수를 ~로 설정

앞서 설명한 바와 같이 막대 그래프(add\_bars())는 X축과 Y축의 이변량을 필요로 하는 시각화이고 히스토그램(add\_histogram())은 X축에 따른 사례수를 구간(bin)에 따라 막대로 표시한 시각화이다.

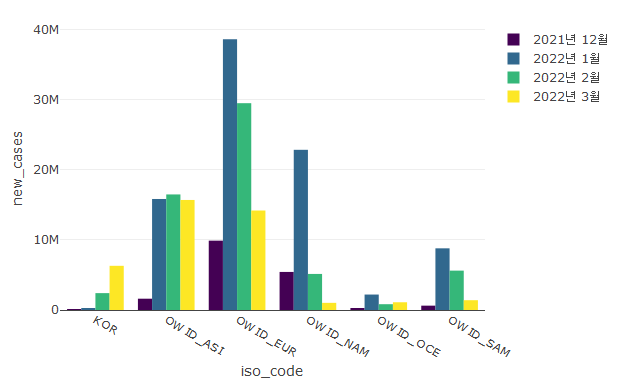
p\_bar <- covid19\_df\_100 |> mutate(yearmonth = tsibble::yearmonth(date)) |> group\_by(iso\_code, yearmonth) |>  
 summarise(new\_cases = sum(new\_cases)) |> plot\_ly()  
  
p\_bar |>  
 add\_trace(type = 'bar', x = ~iso\_code, y = ~new\_cases)



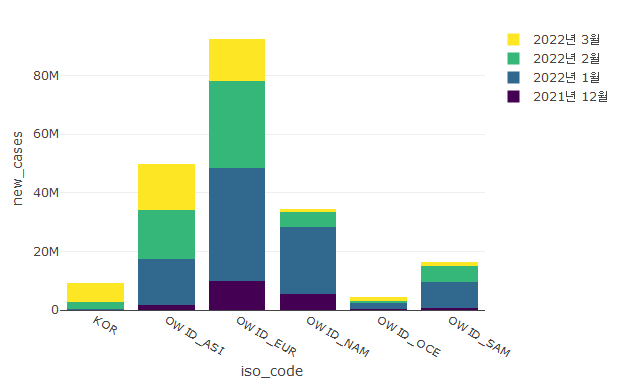
#### barmode

막대 그래프는 일반적으로 세 가지 형태로 생성된다. ggplot2에서는 dodge, stack, fill로 불리며 position을 셋중에 하나로 설정함으로써 자동적으로 그려지지만 plotly에서 막대 그래프를 생성하는 add\_trace()나 add\_bars()로는 이 세 가지 막대 그래프를 구분하여 생성할 수 없다. 이를 구분하여 생성하기 위한 방법은 차후 설명할 layout()의 barmode type을 설정함으로써 가능하다. barmode는 ‘stack’, ‘group’, ‘overlay’, ’relative’의 네 가지 모드에 대해 설정이 가능하다. ’stack’은 세부 그룹 카테고리를 위에 쌓아 올리면서 막대를 구성하고 ’group’은 세부 그룹 카테고리를 옆에 배열하여 막대를 그룹화하는 방식이으로 ggplot2의 ’dodge’에 해당한다. ’overlay’는 막대들이 서로 겹쳐져서 표시되는데 정상적으로 생성하기 위해서는 opacity로 투명도를 조절하여야 한다. ’relative’는 Y축을 중간에 두고 양의 값은 위로 음의 값은 아래로 막대가 향하게 하는 막대 그래프이다. barmode의 기본값은 ’group’이기 때문에 X, Y 축 설정외에 추가적인 그룹화 변수를 매핑하면 기본값인 ’group’형 막대 그래프가 생성된다.

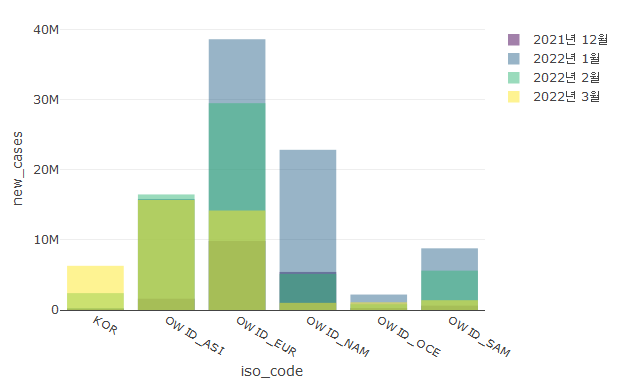
p\_bar\_mode <- covid19\_df\_100 |> mutate(yearmonth = lubridate::floor\_date(date, "month")) |> group\_by(iso\_code, yearmonth) |>  
 summarise(new\_cases = sum(new\_cases)) |> plot\_ly()  
  
p\_bar\_mode |>  
 add\_trace(type = 'bar', x = ~iso\_code, y = ~new\_cases,   
 color = ~as.factor(yearmonth), colors = 'viridis',   
 name = ~paste0(lubridate::year(yearmonth), '년 ', lubridate::month(yearmonth, label = TRUE, abbr = FALSE))) |>  
 layout(barmode = 'group')



p\_bar\_mode |>  
 add\_trace(type = 'bar', x = ~iso\_code, y = ~new\_cases,   
 color = ~as.factor(yearmonth), colors = 'viridis',   
 name = ~paste0(lubridate::year(yearmonth), '년 ', lubridate::month(yearmonth, label = TRUE, abbr = FALSE))) |>  
 layout(barmode = 'stack')



p\_bar\_mode |>  
 add\_trace(type = 'bar', x = ~iso\_code, y = ~new\_cases,   
 color = ~as.factor(yearmonth), colors = 'viridis', opacity = 0.5,   
 name = ~paste0(lubridate::year(yearmonth), '년 ', lubridate::month(yearmonth, label = TRUE, abbr = FALSE))) |>  
 layout(barmode = 'overlay')



# plot\_ly() |> add\_trace(type = 'bar', x = 2000:2021, y = diff(df\_입학자[df\_입학자$지역 == '전체', ]$일반대학),   
# color = I(ifelse(diff(df\_입학자[df\_입학자$지역 == '전체', ]$일반대학)> 0, "red","blue"))) |>   
# layout(barmode = 'relative')

앞서 스캐터 trace의 선 그래프를 그릴때 넓은 형태의 데이터프레임을 사용하여 trace를 하나 하나 추가하는 형태의 시각화를 그려보았다. 막대 trace도 동일하게 생성할 수 있다. plotly 객체에 계속적으로 막대 trace를 추가한 후 layout()의 barmode를 원하는 형태로 설정하면 된다. 이 과정에서 name으로 각각의 범례에 표시될 아이템 이름을 설정할 수 있고 trace의 추가 순서에 따라 범례의 순서를 설정할 수 있으며, 특정 막대의 색을 강조함으로써 특정 데이터를 강조하는데 유용하게 활용할 수 있다.

fig <- df\_취업통계\_2000 |> group\_by(과정구분, 대계열) |>  
 summarise(졸업자합계 = sum(졸업자\_계)) |>   
 pivot\_wider(names\_from = 과정구분, values\_from = 졸업자합계) |>   
 plot\_ly()  
  
fig <- fig |> add\_trace(type = 'bar', x = ~대계열, y = ~전문대학과정,   
 name = '전문대학과정', color = I('gray40'))  
  
fig <- fig |> add\_trace(type = 'bar', x = ~대계열, y = ~대학과정,   
 name = '대학과정', color = I('darkred'))  
  
fig <- fig |> add\_trace(type = 'bar', x = ~대계열, y = ~대학원과정,   
 name = '대학원과정', color = I('gray60'))  
  
fig |> layout(barmode = 'stack')

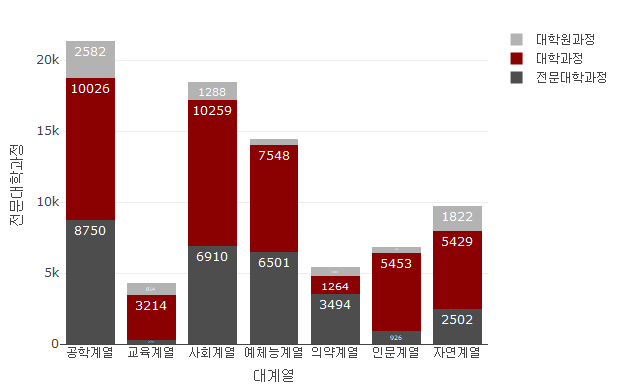
#### text, textposition, texttemplate, textfont

막대 그래프는 일반적으로 데이터 간의 비교를 위해 사용된다. 따라서 막대 그래프에 표시된 막대로는 데이터의 정확한 값을 알아보기는 어렵다. 이를 위해 막대그래프에 데이터 값을 표기하는 경우가 많다. plotly의 막대 trace에 데이터 값을 표현하기 위해서는 text type을 사용하면 간단히 표시할 수 있다. 표시되기를 원하는 변수를 text에 매핑함으로써 각각의 막대에 데이터 값을 표시할 수 있으며 textposition과 texttemplate를 사용하여 표시되는 값의 세부 설정을 변경할 수 있다.

text type은 막대에 표시되어야 하는 문자열을 매핑하거나 설정하는 type이다. text에 단일 문자열을 설정하면 모든 막대에 설정된 문자열이 표시되고 문자열 벡터가 설정되면 문자열 벡터의 순서에 따라 막대 위에 문자열이 표시된다.

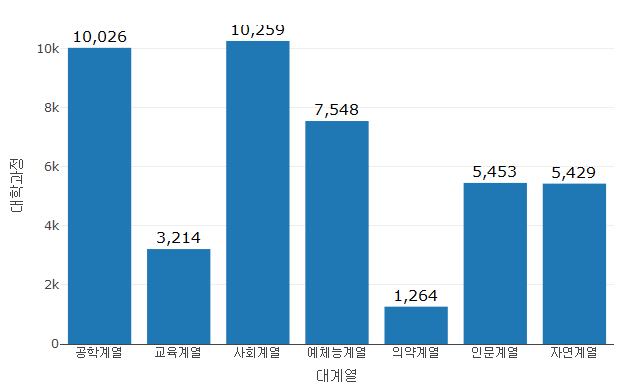
textposition는 text의 위치를 설정하는 type이다. textposition에는 ‘inside’, ‘outside’, ‘auto’, ‘none’의 네 가지를 설정할 수 있다. ’inside’는 막대의 안쪽에 텍스트를 위치시킨다. 이 경우 막대의 너비에 따라 가로로 표시될 수도 있고 세로로 표시될 수도 있다. ’outside’는 막대 끝의 바깥에 텍스트를 위치시키는데 마찬가지로 막대의 너비에 따라 가로 혹은 세로로 표기될 수 있다. 또 ’outside’는 막대가 쌓이는 ’stack’ 형의 막대 그래프에서는 ’inside’와 동일하게 표시된다. ’auto’는 plotly에서 자동적으로 계산된 형태로 텍스트가 표시된다. ’none’은 텍스트가 표시되지 않는다.

fig <- df\_취업통계\_2000 |> group\_by(과정구분, 대계열) |>  
 summarise(졸업자합계 = sum(졸업자\_계)) |>   
 pivot\_wider(names\_from = 과정구분, values\_from = 졸업자합계) |>   
 plot\_ly()  
  
fig <- fig |> add\_trace(type = 'bar', x = ~대계열, y = ~전문대학과정,   
 name = '전문대학과정', color = I('gray30'),   
 text = ~전문대학과정, textposition = 'inside',   
 textfont = list(color = 'white'))  
  
fig <- fig |> add\_trace(type = 'bar', x = ~대계열, y = ~대학과정,   
 name = '대학과정', color = I('darkred'),   
 text = ~대학과정, textposition = 'inside',   
 textfont = list(color = 'white'))  
  
fig <- fig |> add\_trace(type = 'bar', x = ~대계열, y = ~대학원과정,   
 name = '대학원과정', color = I('gray70'),   
 text = ~대학원과정, textposition = 'inside',   
 textfont = list(color = 'white'))  
  
fig |> layout(barmode = 'stack')



texttemplate은 텍스트가 표시되는 형태를 설정하는 type이다. texttemplate는 hovertemplate의 정의와 같이 사용되는데 ‘%{변수:변수포맷}’의 형태로 사용된다. 변수 포맷에서 사용하는 포맷은 자바 스크립트의 d3 format[[4]](#footnote-49)을 사용한다. 앞의 예에서 일반대학의 입학생수의 포맷에 천단위 콤마를 넣는다면’%{text:,}‘로 설정하고 소수점 아래 두째 자리까지 표기한다면’%{text:2f}’로 설정한다.

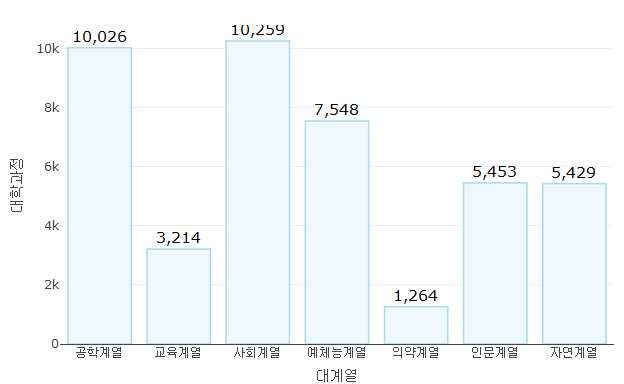
df\_취업통계\_2000 |> group\_by(과정구분, 대계열) |>  
 summarise(졸업자합계 = sum(졸업자\_계)) |>   
 pivot\_wider(names\_from = 과정구분, values\_from = 졸업자합계) |>   
 plot\_ly() |>  
 add\_trace(type = 'bar', x = ~대계열, y = ~대학과정, name = '대학과정',   
 text = ~대학과정, textposition = 'outside',   
 textfont = list(color = 'black', size = 15), texttemplate = '%{text:,}')



#### marker

앞서 스캐터 trace에서 산점도, 선 그래프를 생성할 때 marker를 사용하여 점과 선의 형태를 설정하였다. 그러나 막대 trace에서의 makrker는 막대 자체를 의미한다. 따라서 marker로 설정 가능한 type은 막대의 내부 색, 외부 외곽선 등의 세부 설정을 위한 type을 지정할 수 있다.

df\_취업통계\_2000 |> group\_by(과정구분, 대계열) |>  
 summarise(졸업자합계 = sum(졸업자\_계)) |>   
 pivot\_wider(names\_from = 과정구분, values\_from = 졸업자합계) |>   
 plot\_ly() |>  
 add\_trace(type = 'bar', x = ~대계열, y = ~대학과정, name = '대학과정',   
 text = ~대학과정, textposition = 'outside',   
 textfont = list(color = 'black', size = 15), texttemplate = '%{text:,}',   
 marker = list(color = 'aliceblue',  
 line = list(color = 'lightblue', width = 1.5))  
)

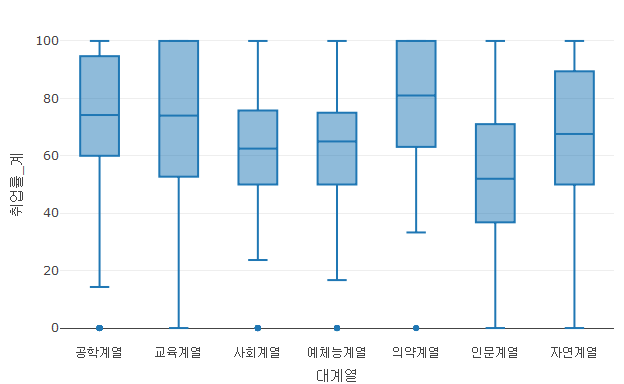


### 박스(Box) trace : 박스 플롯

박스 trace는 박스 플롯을 생성하기 위해 사용되는 trace이다. 앞 장에서 설명했듯이 박스 플롯은 데이터의 전체적 분포를 4분위수(quantile)과 IQR(Inter Quartile Range)를 사용하여 표시하는 시각화로 연속형 변수와 이산형 변수의 시각화에 사용되는 방법이다. 박스 trace를 사용해 박스 플롯을 생성하기 위해서는 add\_trace(type = 'box')를 사용하거나 add\_boxplot()을 사용한다.

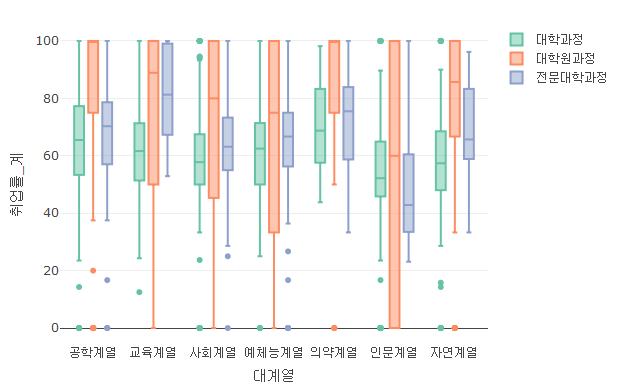
add\_trace(p, type = ‘box’, …, data = NULL, inherit = TRUE)  
add\_boxplot(p, x = NULL, y = NULL, …, data = NULL, inherit = TRUE)  
- p : plot\_ly()로 생성한 plotly 객체  
- type : trace 타입을 ’box’로 설정  
- … : 박스 trace의 line 모드에 설정할 수 있는 속성 설정  
- data : 시각화할 데이터프레임  
- inherit : plot\_ly()에 설정된 속성 type을 상속할지를 결정하는 논리값  
- x : X축에 매핑할 변수를 ~로 설정  
- y : Y축에 매핑할 변수를 ~로 설정

p\_box <- df\_취업통계\_2000 |> plot\_ly()  
  
p\_box |> add\_trace(type = 'box', x = ~대계열, y = ~취업률\_계)



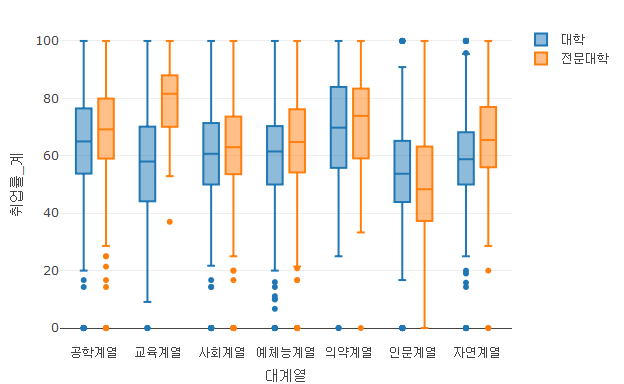
각 데이터별로 그룹화하여 분할된 바이올린 trace를 생성하기 위해서는 그룹화할 변수를 color나 linetype등의 type에 매핑 시키거나 각각의 바이올린 trace를 추가하고 layout()의 boxmode를 ’group’으로 설정함으로써 생성할 수 있다. 다음의 코드는 color로 과정구분을 매핑하여 각각의 과정별 박스 플롯을 생성하는 코드이다.

p\_box\_group <- p\_box |> add\_trace(type = 'box', x = ~대계열, y = ~취업률\_계,   
 color = ~과정구분)  
  
p\_box\_group %>% layout(boxmode = "group")



다음의 코드는 과정구분이 ’대학’인 박스 trace와 ’전문대학’인 박스 trace를 추가함으로서 그룹화된 박스 플롯을 생성하는 코드이다.

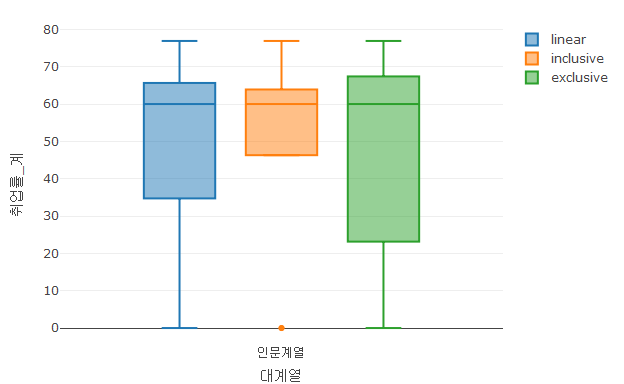
p\_box\_add <- p\_box |>   
 add\_trace(data = df\_취업통계 |> filter(졸업자\_계 < 500, 과정구분 == '대학과정'),  
 type = 'box', x = ~대계열, y = ~취업률\_계, name = '대학')  
  
p\_box\_add <- p\_box\_add |>   
 add\_trace(data = df\_취업통계 |> filter(졸업자\_계 < 500, 과정구분 == '전문대학과정'),   
 type = 'box', x = ~대계열, y = ~취업률\_계, name = '전문대학')  
  
p\_box\_add %>% layout(boxmode = "group")



#### quartilemethod

박스 플롯에는 여러가지 요약 통계치가 사용되는데 그중 하나가 전체 데이터의 25% 위치인 Q1과 75% 위치인 Q3이다. 이 위치는 전체 데이터를 Y값에 따라 정렬하고 25%, 75%의 값을 찾는 ‘linear’ 방법이 기본적으로 사용된다. 하지만 plotly 는 ‘inclusive’와 ’exclusive’의 두가지 방법을 추가적으로 제공한다. ’exclusive’ 방법은 전체 데이터 사례수가 홀수 인경우 중앙값을 사용하여 두 개의 그룹으로 분리하고 다시 이 두개의 그룹의 중앙값을 사용해 Q1과 Q3를 구하는 방법이다. ‘inclusive’ 방법도 중앙값을 사용하여 두 그룹으로 분리하지만 중앙값을 포함하여 다시 두 그룹의 중앙값을 구하여 Q1과 Q3를 구하는 방법이다. 이들 방법은 사례수가 적은 경우 그 차이가 확실히 드러나지만 사례수가 많으면 그 차이가 잘 드러나지는 않는다.

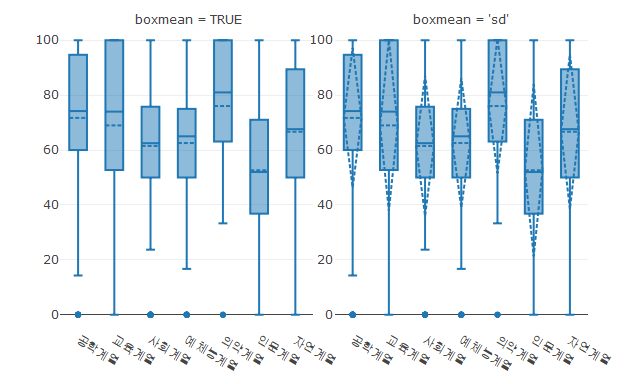
p\_box\_quartilemethod <- df\_취업통계\_2000 |> filter(대계열 == '인문계열') |> head(9) |> plot\_ly()  
  
p\_box\_quartilemethod <- p\_box\_quartilemethod |>   
 add\_trace(type = 'box', x = ~대계열, y = ~취업률\_계,   
 quartilemethod = 'linear', name = 'linear')  
  
p\_box\_quartilemethod <- p\_box\_quartilemethod |>   
 add\_trace(type = 'box', x = ~대계열, y = ~취업률\_계,   
 quartilemethod = 'inclusive', name = 'inclusive')  
  
p\_box\_quartilemethod <- p\_box\_quartilemethod |>   
 add\_trace(type = 'box', x = ~대계열, y = ~취업률\_계,   
 quartilemethod = 'exclusive', name = 'exclusive')  
  
p\_box\_quartilemethod %>% layout(boxmode = "group")



#### boxmean, boxpoints

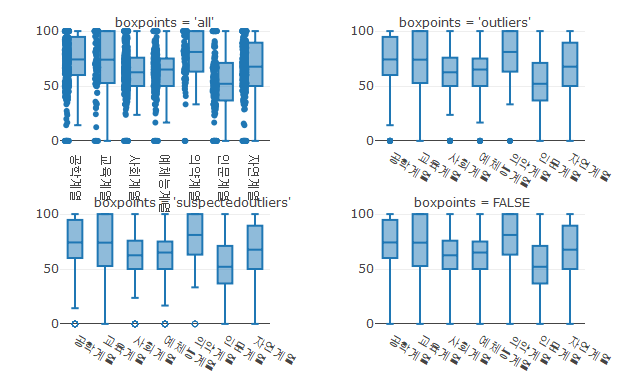
박스 trace에서 제공하는 요약 통계중에 가장 많이 사용되지만 제공되지 않는 요약통계가 바로 평균(mean)이다. ggplot2에서는 평균을 표현하기 위해 다소 어려운 과정을 거쳐야 했지만 plotly에서는 boxmean type의 설정만으로 간단하게 평균값을 표현할 수 있다. boxmean은 TRUE/FALSE의 논리값에 표준편차가 추가로 표시되는 ’sd’를 제공한다.

p\_box |> add\_trace(type = 'box', x = ~대계열, y = ~취업률\_계, boxmean = TRUE)  
  
p\_box |> add\_trace(type = 'box', x = ~대계열, y = ~취업률\_계, boxmean = 'sd')



또 박스 trace에서 유용하게 사용되는 type이 boxpoints이다. boxpoints는 이상치(outlier)로 표현되는 점의 표현을 제어할 수 있다. boxpoints로 설정 가능한 이상치 표시 설정은 ‘all’, ‘outliers’, ‘suspectedoutliers’, ‘FALSE’의 네 가지 방법이 제공된다. ’all’은 모든 이상치를 보여주지만 ’outliers’는 수염 외부에 있는 이상치만 표시하고 ’suspectedoutliers’ 전체 이상치가 표시되지만 값의 범위가 IQR의 4배가 넘어가는 이상치는 다시 강조되는 방법이다. ’FALSE’는 이상치를 표시하지 않는다.

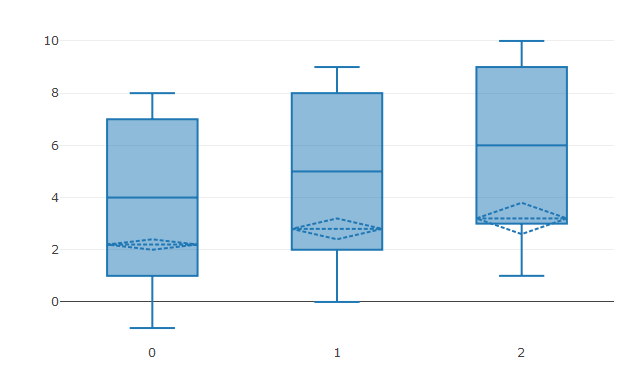
p\_box |> add\_trace(type = 'box', x = ~대계열, y = ~취업률\_계,   
 boxpoint = 'all')  
  
p\_box |> add\_trace(type = 'box', x = ~대계열, y = ~취업률\_계,   
 boxpoint = 'outliers')  
  
p\_box |> add\_trace(type = 'box', x = ~대계열, y = ~취업률\_계,   
 boxpoint = 'suspectedoutliers')  
  
p\_box |> add\_trace(type = 'box', x = ~대계열, y = ~취업률\_계,   
 boxpoint = FALSE)



#### mean, median, q1, q3, upperfence, lowerfence

박스 trace는 박스 플롯의 생성시 자동적으로 여러가지 요약 통계를 산출하지만 경우에 따라 이 값들을 미리 정해진 값으로 사용할 수도 있다. 이를 위해 사용되는 type들이 mean(평균), median(중앙값), q1(25%), q3(75%), upperfence(상위 펜스), lowerfence(하위 펜스) 등이다. 이 type은 각각의 값이 설정된 데이터프레임 열, 벡터, 리스트를 사용할 수 있는데 이들의 길이는 박스의 개수(X축 아이템의 수)와 동일해야 한다. 따라서 이 type들을 설정함으로써 박스 플롯을 사용자가 직접 생성할 수도 있다.

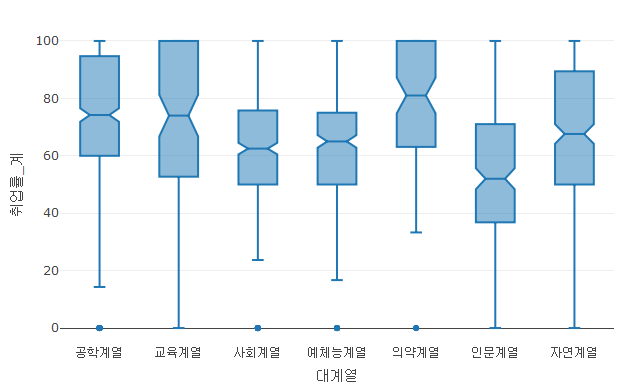
plot\_ly() |>   
 add\_trace(type= 'box', q1=c(1, 2, 3), median=c(4, 5, 6),  
 q3=c(7, 8, 9 ), lowerfence=c(-1, 0, 1),  
 upperfence=c(8, 9, 10), mean=c(2.2, 2.8, 3.2 ),  
 sd=c(0.2, 0.4, 0.6))



#### notched, notchwidth, notchspan, whiskerwidth

2017년 iPhoneX가 처음 소개되면서 유명세를 탄 디자인 스타일이 notch 스타일이다. 이 notch 스타일은 평면이나 직선의 일부에 삼각형이나 사각형의 홈을 내는 디자인 스타일을 말한다. 박스 trace에도 이 notch 스타일을 적용할 수 있는데 중앙값이 표시되는 부분에 삼각형 홈을 내는 type이 notched이다. notched는 TRUE/FALSE의 두 가지 중에 하나의 값을 가지는데 TRUE일 경우 notch 디자인으로 설정된다. notched가 TRUE로 설정된 박스 플롯은 notchwidth, notchspan을 사용하여 홈의 크기를 설정할 수 있는데 notchwidth는 홈이 파고들어간 너비를 설정하는데 전체 막대의 너비에 대한 비율로 설정하기 때문에 0부터 0.5까지의 값을 가질수 있다. 또 notchspan은 수직 방향으로의 홈의 크기를 설정하는 type이다. 막대 외부에 표시되는 수염의 끝에 있는 가로선의 크기를 설정하는 type이 whiskerwidth이다. 이 type도 notchwidth와 같이 0부터 0.5사이의 값을 가진다.

p\_box |> add\_trace(type = 'box', x = ~대계열, y = ~취업률\_계,   
 notched = TRUE, notchewidth = 0.2, notchespan = 0.5,   
 whiskerwidth = 0.3)



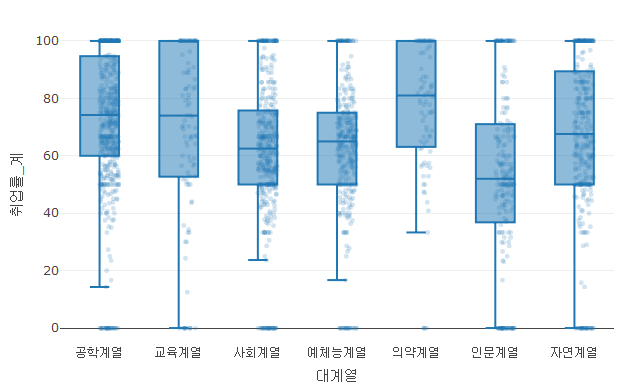
#### marker, jitter

막대 trace를 사용하면 전체적인 데이터의 분포를 막대의 선으로 살펴볼 수 있지만 이 구간에서 데이터가 집중된 위치나 데이터 발생 빈도가 낮은 위치를 알아볼 수는 없다. 이를 알아보기 위해서는 실제 데이터의 위치에 점을 찍어 데이터의 분포를 직접 보는 방법이 사용되는데 이렇게 막대 trace에서 데이터의 실제 분포를 표시하는 type이 marker와 jitter이다.

marker는 박스 trace에서 사용되는 점들의 세부 설정을 위한 type이다. 이 세부 설정은 스캐너 trace에서 사용되는 marker와 거의 동일하기 때문에 앞의 스캐터 trace의 marker를 참조하면 된다.

jitter에는 0부터 1까지의 값을 가지는데 샘플 데이터 점들이 표시되는 구간의 너비가 설정된다. 0은 한 줄에 표기되고 1은 박스의 너비와 동일한 너비로 점들이 분산되어 표기된다. jitter와 함께 쓰여 샘플 데이터의 표시를 제어하는 type으로 boxpoints와 pointpos가 있다. boxpoints는 박스에 표시되는 점의 범위를 설정한다. 기본값은 ’outlier’로 설정되어 있는데 이 경우 표시되는 점은 이상치에 속하는 점만 표현되기 때문에 jitter의 효과를 보지 못한다. 이 값을 ’all’로 설정하여 전체 데이터로 범위를 설정하여야 전체적인 분포를 볼 수 있다. 또 pointpos는 점이 표기되는 위치를 설정하는데 -2부터 2까지의 값을 가진다. 이 값은 0(박스의 중심)부터 박스 너비의 2배까지의 위치를 좌우로 설정할 수 있다.

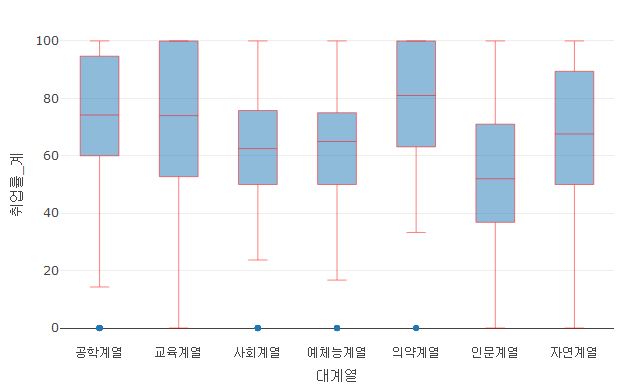
p\_box |>   
 add\_trace(type = 'box', x = ~대계열, y = ~취업률\_계,   
 marker = list(opacity = 0.2, size = 5, color = 'aquablue'),  
 boxpoints = "all", jitter = 0.5, pointpos = 0.5)



#### line

line은 막대 trace에서 외곽선을 설정하는 type이다. 이 type에서는 세부 선의 색과 두께를 설정할 수 있다.

p\_box |>   
 add\_trace(type = 'box', x = ~대계열, y = ~취업률\_계,   
 line = list(width = 0.5, color = 'red'))



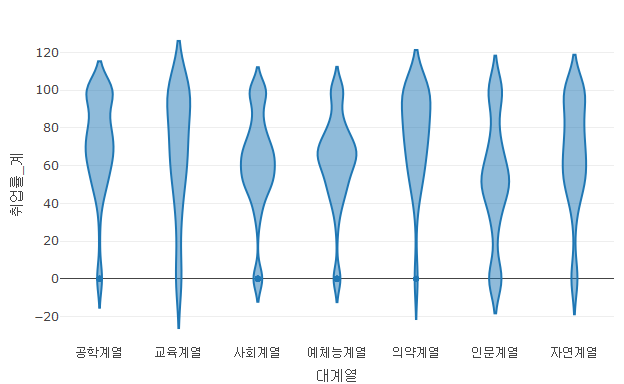
### 바이올린(Violin) trace : 바이올린 플롯

바이올린 trace는 바이올린 플롯을 생성하기 위해 사용되는 trace이다. 앞 장에서 설명했듯이 바이올린 플롯은 박스 플롯에서 확인하기 어려운 데이터의 분포를 확인할 수 있는 시각화 방법이다.

바이올린 trace를 사용해 바이올린 플롯을 생성하기 위해서는 add\_trace(type = 'violin')를 사용하여야하고 래핑함수를 제공하지 않는다.

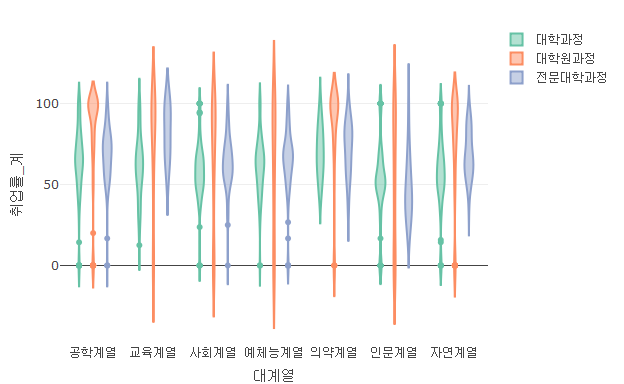
add\_trace(p, type = ‘violin’, …, data = NULL, inherit = TRUE)  
- p : plot\_ly()로 생성한 plotly 객체  
- type : trace 타입을 ’violin’로 설정  
- … : 바이올린 trace의 line 모드에 설정할 수 있는 속성 설정  
- data : 시각화할 데이터프레임  
- inherit : plot\_ly()에 설정된 속성 type을 상속할지를 결정하는 논리값

p\_violin <- df\_취업통계\_2000 |> plot\_ly()  
  
p\_violin |> add\_trace(type = 'violin', x = ~대계열, y = ~취업률\_계)



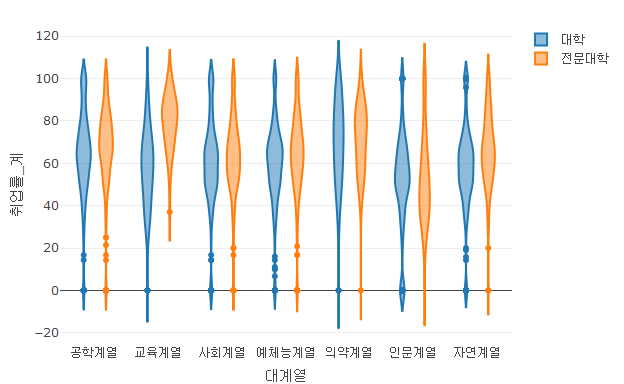
박스 trace와 마찬가지로 각 데이터별로 그룹화하여 분할된 바이올린 trace를 생성하기 위해서는 그룹화할 변수를 color나 linetype등의 type에 매핑 시키거나 각각의 바이올린 trace를 추가하고 layout()의 violinmode를 ’group’으로 설정함으로써 생성할 수 있다. 다음의 코드는 color로 과정구분을 매핑하여 각각의 과정별 박스 플롯을 생성하는 코드이다.

p\_violin |> add\_trace(type = 'violin', x = ~대계열, y = ~취업률\_계,   
 color = ~과정구분) |>   
 layout(violinmode = 'group')



다음의 코드는 과정구분이 ’대학’인 바이올린 trace와 ’전문대학’인 바이올린 trace를 추가함으로서 그룹화된 바이올린 플롯을 생성하는 코드이다.

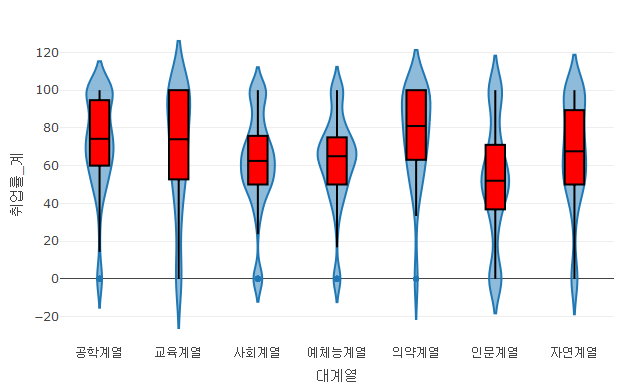
p\_violin |> add\_trace(data = df\_취업통계 |> filter(졸업자\_계 < 500, 과정구분 == '대학과정'),   
 type = 'violin', x = ~대계열, y = ~취업률\_계, name = '대학') |>   
 add\_trace(data = df\_취업통계 |> filter(졸업자\_계 < 500, 과정구분 == '전문대학과정'),   
 type = 'violin', x = ~대계열, y = ~취업률\_계, name = '전문대학') |>  
 layout(violinmode = "group")



#### box, meanline

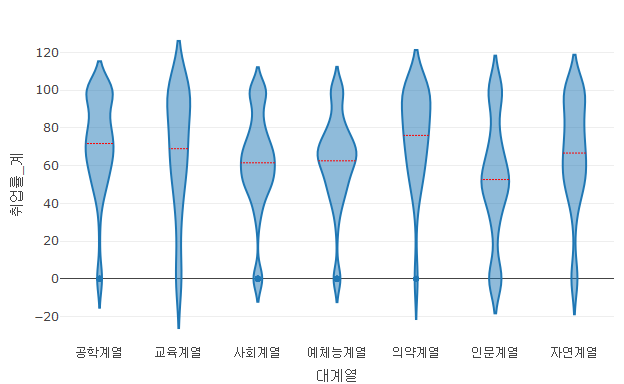
바이올린 플롯은 박스 플롯와 유사한 정보를 제공하는 시각화이지만 박스 플롯의 정보와 같이 사용되면 더욱 데이터 시각화의 효과가 높아진다. 바이올린 trace에 박스 trace를 추가하는 type은 box이고 이 box type은 세부 타입을 list로 설정하는 type이다. box의 세부 type은 fillcolor, line(세부 type으로 color, width), visible, width의 네 가지이다.

p\_violin |> add\_trace(type = 'violin', x = ~대계열, y = ~취업률\_계,   
 box = list(visible = TRUE, fillcolor = 'red',   
 line = list(color = 'black'), width = 0.5  
 )  
 )



meanline은 바이올린 trace에 평균 선을 설정하는 type으로 color, visible, width의 세부 type을 리스트로 설정한다

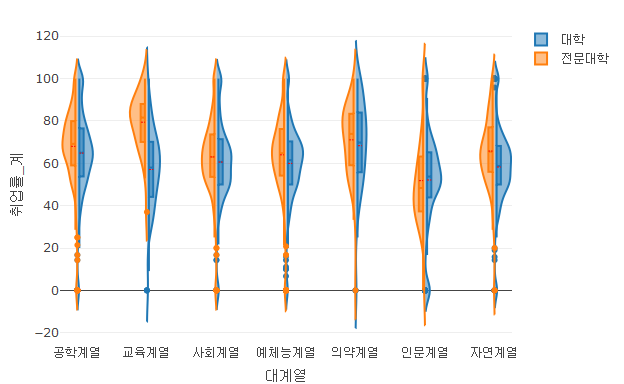
p\_violin |>  
 add\_trace(type = 'violin', x = ~대계열, y = ~취업률\_계,   
 meanline = list(visible = TRUE, color = 'red', width = 1)  
 )



#### side

앞의 예에서 대학과 전문대학의 바이올린 trace를 추가함으로써 그룹화된 바이올린 플롯을 그렸다 그런데 이 두 바이올린 플롯을 반씩 붙여서 그리면 각각의 바이올린 플롯이 넓어지기 때문에 데이터를 확인하기 쉬울 것이다. 이렇게 두개의 바이올린 플롯을 반씩 잘라 붙이는 type이 side이다. side는 바이올린의 양쪽을 다 사용하는 ‘both’, 왼쪽 부분을 사용하는 ‘negative’, 오른쪽 부분을 사용하는 ’positive’를 설정할 수 있다. 그리고 이 두 바이올린을 붙이기 위해 layout()의 violinmode를 overlay로 설정한다. 여기에 앞서 설정한 box와 meanline 설정하면 박스 trace와 평균선도 반으로 그려서 붙여주게 된다. 앞서 그렸던 대학과 전문대학의 계열별 바이올린 플롯을 붙이는 코드는 다음과 같다.

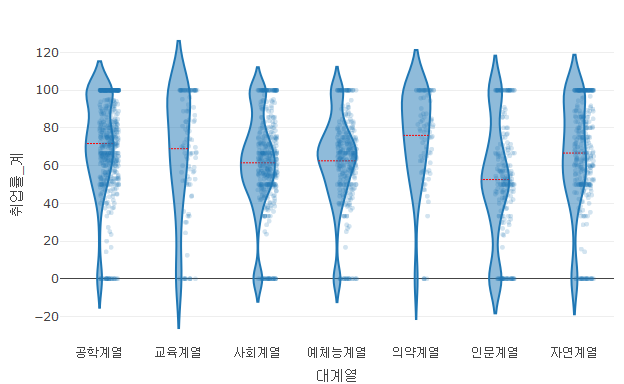
p\_violin |>   
 add\_trace(data = df\_취업통계 |> filter(졸업자\_계 < 500, 과정구분 == '대학과정'),  
 type = 'violin', x = ~대계열, y = ~취업률\_계, name = '대학',   
 side = 'positive', box = list(visible = TRUE, width = 0.5),   
 meanline = list(visible = TRUE, color = 'red', width = 1)  
 ) |>  
 add\_trace(data = df\_취업통계 |> filter(졸업자\_계 < 500, 과정구분 == '전문대학과정'),   
 type = 'violin', x = ~대계열, y = ~취업률\_계, name = '전문대학',   
 side = 'negative', box = list(visible = TRUE, width = 0.5),   
 meanline = list(visible = TRUE, color = 'red', width = 1)  
 ) |>   
 layout(violinmode = "overlay")



#### marker, jitter, line

바이올린 trace에서도 박스 trace와 같이 데이터를 점으로 표현할 수 있는데 이를 제어하기 위한 type이 marker와 jitter이고 바이올린의 외곽선을 설정하는 type이 line이다. 이들의 설정은 박스 trace와 동일하나 boxpoints가 points로 바뀐다는 점이 다르다.

p\_violin |>  
 add\_trace(type = 'violin', x = ~대계열, y = ~취업률\_계,   
 meanline = list(visible = TRUE, color = 'red', width = 1),   
 marker = list(opacity = 0.2, size = 5, color = 'aquablue'),  
 points = "all", jitter = 0.5, pointpos = 0.5)



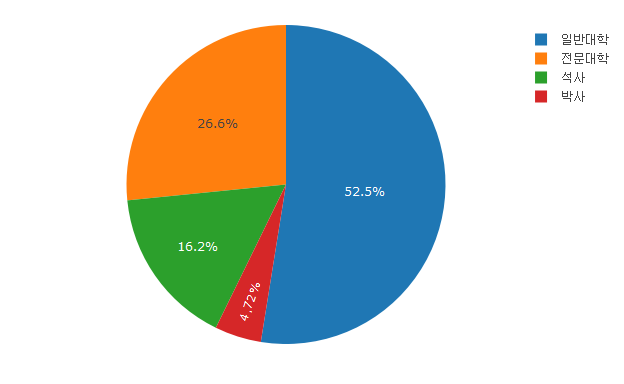
### 파이(pie) trace : 원 그래프

ggplot2에서 지원하지 않는 그래프 스타일이 원 그래프이다. 그래서 ggplot2에서 원 그래프를 그리기 위해서는 막대 그래프를 그리고 이 막대 그래프를 극 좌표계를 사용하여 돌리는 트릭을 사용한다. 원 그래프는 데이터를 정확히 분석하기 어렵다는 차원에서 데이터 분석가들이 잘 사용하지 않는 그래프 타입이지만 현실적으로 많이 사용되고 있기 때문에 plotly에서는 trace로 지원한다.

add\_trace(p, type = ‘pie’, …, data = NULL, inherit = TRUE)  
add\_pie(p, values = NULL, labels = NULL, …, data = NULL, inherit = TRUE)  
- p : plot\_ly()로 생성한 plotly 객체  
- type : trace 타입을 ’violin’로 설정  
- … : 바이올린 trace의 line 모드에 설정할 수 있는 속성 설정  
- data : 시각화할 데이터프레임  
- inherit : plot\_ly()에 설정된 속성 type을 상속할지를 결정하는 논리값  
- values : 원 그래프의 각각의 슬라이스에 매칭될 값 설정  
- labels : 원 그래프의 각각의 슬라이스에 표시될 라벨 설정

앞선 여타 trace와는 달리 파이 trace에는 X, Y축에 매핑되는 x, y type의 설정이 없다. 대신 필수적 type으로 values와 labels를 설정해야하는데 values에 매핑되거나 설정된 수치 벡터에 따라 원을 각각의 슬라이스로 구분하게 되고 labels에 매핑하거자 설정한 벡터가 각각의 trace 이름으로 설정된다. 또 원 그래프에 표시되는 데이터 값은 기본적으로 백분률 값이 표시된다.

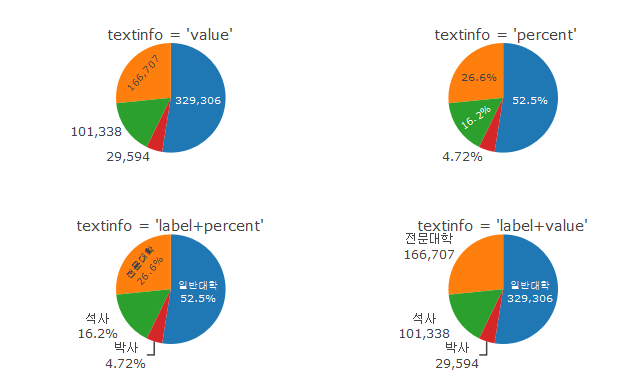
p\_pie <- df\_입학자\_long |> filter(연도 == '2021', 지역 == '전체', 학교종류 %in% c('전문대학', '일반대학', '석사', '박사')) |> plot\_ly()  
  
p\_pie |>  
 add\_trace(type = 'pie', values = ~입학생수, labels = ~학교종류)



#### textinfo

앞의 원 그래프에서 원 안에 표현된 데이터는 values에 매핑된 데이터의 백분율이 표시되었다. 이 데이터의 표시를 설정하는 type이 textinfo이다. textinfo에서 설정사 가능한 값은 ‘label’, ‘text’, ‘value’, ’percent’의 네 가지이고 이 네가지를 +를 사용하여 동시에 사용할 수 있다. ’label’은 labels로 설정된 값, text는 text로 설정된 값, ’value’는 values에 설정된 값, ’percent’는 백분률 값이 표시된다.

p\_pie |>  
 add\_trace(type = 'pie', values = ~입학생수, labels = ~학교종류, textinfo = 'value')  
  
p\_pie |>  
 add\_trace(type = 'pie', values = ~입학생수, labels = ~학교종류, textinfo = 'percent')  
  
p\_pie |>  
 add\_trace(type = 'pie', values = ~입학생수, labels = ~학교종류, textinfo = 'label+percent')  
  
p\_pie |>  
 add\_trace(type = 'pie', values = ~입학생수, labels = ~학교종류, textinfo = 'label+value')



#### hole

hole은 원 그래프의 내부에 표시되는 구멍의 크기를 설정하는 type이다. hole이 설정되지 않거나 0으로 설정되면 완전한 원 그래프로 표시되지만 0부터 1사이의 값이 설정되면 그 크기만큼 빈 원이 생성되면서 도넛 그래프로 전환된다.

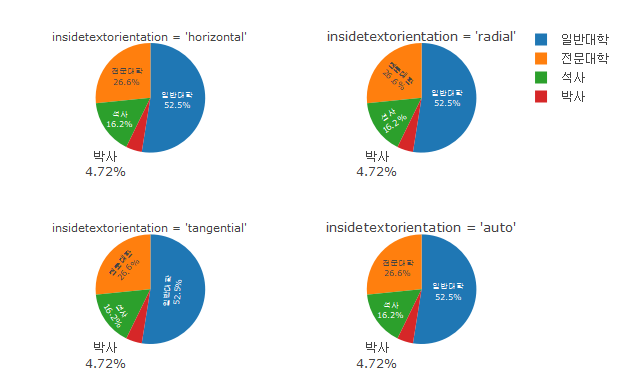
p\_pie |>  
 add\_trace(type = 'pie', values = ~입학생수, labels = ~학교종류, textinfo = 'label+percent', hole = 0.4)

#### marker

p\_pie |>  
 add\_trace(type = 'pie', values = ~입학생수, labels = ~학교종류, textinfo = 'label+percent', marker = list(colors = c('red', 'green', 'blue', 'black'), line = list(color = 'white', width = 1)))

#### insidetextorientation

p\_pie |>  
 add\_trace(type = 'pie', values = ~입학생수, labels = ~학교종류, textinfo = 'label+percent', insidetextorientation = 'horizontal')  
  
p\_pie |>  
 add\_trace(type = 'pie', values = ~입학생수, labels = ~학교종류, textinfo = 'label+percent', insidetextorientation = 'radial')  
  
p\_pie |>  
 add\_trace(type = 'pie', values = ~입학생수, labels = ~학교종류, textinfo = 'label+percent', insidetextorientation = 'tangential')  
  
p\_pie |>  
 add\_trace(type = 'pie', values = ~입학생수, labels = ~학교종류, textinfo = 'label+percent', insidetextorientation = 'auto')



### 인디케이터(indicator) trace : 지수 차트

인디케이터는 단일 수치에 대한 시각화 방법이다. plotly에서 단일 수치를 표현하는 방법으로 수치 표시, 증감치 표시, 게이지 사용의 세 가지 시각적 방법을 제공하고 이들을 서로 조합하여 사용할 수 있다. 이들을 사용하여 ‘단계’, ‘임계값’ 등의 컨텍스트 정보를 포함하는 수치를 시각화할 수 있다.

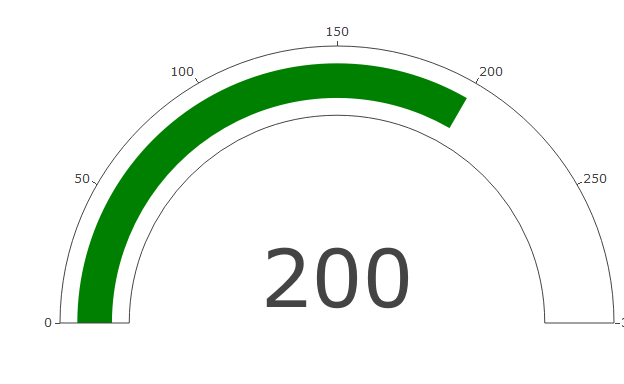
인디케이터 trace를 사용하기 위해서는 add\_trace()의 type을 ’indicator’로 설정함으로써 사용할 수 있다.

#### mode, value

인디케이터 trace에서 제공하는 표시 방법은 수치 표시, 증감량 표시, 게이지 표시의 세가지 방법을 제공한다. 이들을 설정하는 type이 mode이다. 이 mode는 수치를 표현하는 ‘number’, 증감을 표현하는 ‘delta’, 게이지를 사용하는 ’gauge’의 세 가지를 설정할 수 있고 각각은 +를 사용하여 조합할 수 있다.

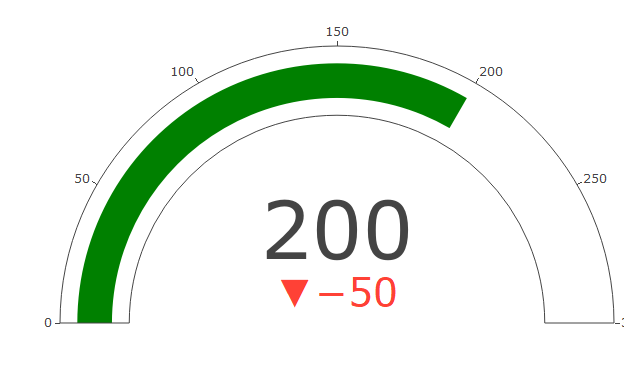
인디케이터에서 표시해야 할 수치는 value로 설정이 가능하고 단일 수치를 설정한다.

plot\_ly() |>  
 add\_trace(type = 'indicator', mode = "gauge+number", value = 200)



#### gauge

plot\_ly() |>  
 add\_trace(type = 'indicator', mode = "gauge+number+delta", gauge = list(shape = 'angular'), value = 200, delta = list(reference = 250))



## 레이아웃 설정 : layout()

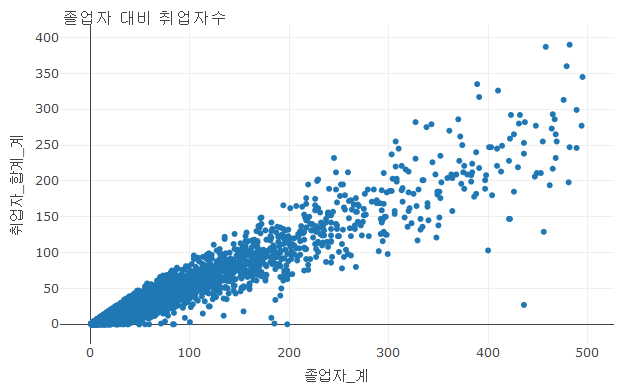
지금까지는 시각화를 만들기 위해 필요한 데이터를 trace로 표현하는 방법에 대해 알아보았다. 이렇게 추가된 trace들을 보다 효과적으로 시각화하기 위해서는 시각화의 제목, 범례, 여백, 크기, 폰트, 축 등에 대한 세부 설정을 해야 할 필요가 있다. plotly에서 이들을 설정하기 위해서는 layout()을 사용하여 설정할 수 있다. 이 설정들은 앞선 add\_trace()와 마찬가지로 list()를 사용하여 설정이 가능하다.

### 제목 설정 : title

plotly의 제목을 설정하는 type은 title이다. title의 세부 type은 다음과 같다.

| type | 설명 | type 값 및 설명 | 세부 type |
| --- | --- | --- | --- |
| font | 제목 글꼴 설정 |  | color, family, size |
| pad | 제목 패딩(여백과 시각화 개체와의 거리) 설정 |  | b, l, r, t |
| text | 제목 문자열 설정 |  |  |
| x | X축의 정렬 | 0부터 1까지의 상대적 위치 설정 |  |
| xanchor | X축의 정렬 | auto, left, right, center 중 하나 설정 |  |
| xref | x가 참조하는 범위 설정 | ’container’는 전체 플롯 width, ’paper’는 플롯이 표시되는 범위 |  |
| y | Y축의 정렬, | 0부터 1까지의 상대적 위치 설정 |  |
| yanchor | Y축의 정렬 | auto, top, middle, bottom 중 하나 설정 |  |
| yref | y가 참조하는 범위 설정 | ’container’는 전체 플롯 width, ’paper’는 플롯이 표시되는 범위 |  |

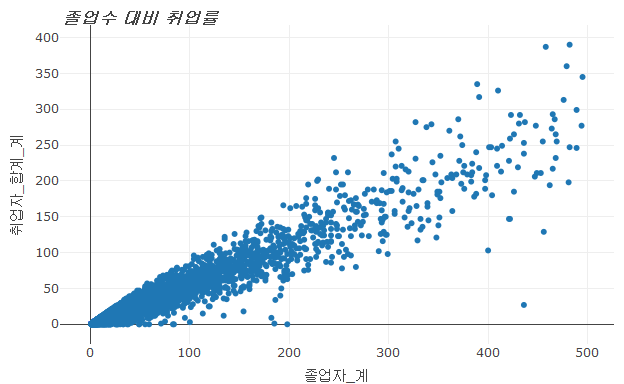
p\_title\_marker <- df\_취업통계 |> filter(졸업자\_계 < 500) |> plot\_ly()  
  
p\_title\_1 <- p\_title\_marker |>  
 add\_trace(type = 'scatter', mode = 'markers', x = ~졸업자\_계,   
 y = ~취업자\_합계\_계)  
  
p\_title\_1 |> layout(title = list(text = '졸업자 대비 취업자수', x = 0.1,   
 xanchor = 'left', yanchor = 'top'  
 )  
 )



plotly에서는 제목과 같은 문자열을 꾸미기 위해 HTML 텍스트 태그를 지원하는데 전체 HTML 중 5가지만을 지원한다.

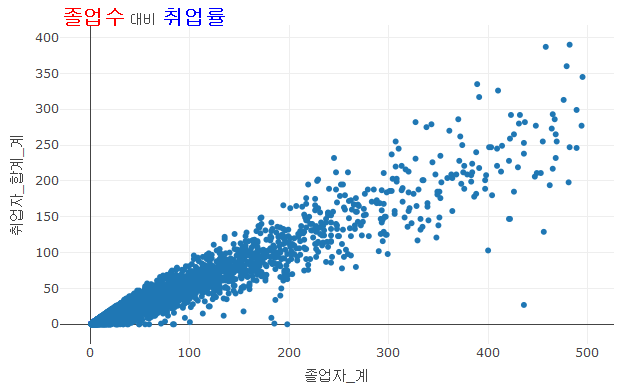
| HTML tab | 설명 |
| --- | --- |
|  | 볼드체 설정 |
|  | 이탤릭체 설정 |
|  | 줄 바꿈 설정 |
|  | 윗 첨자 설정 |
|  | 아래 첨자 설정 |
|  | 하이퍼링크 설정 |

p\_title\_1 |> layout(title = list(text = "<b><i>졸업수 대비 취업률</b></i>",   
 x = 0.1, xanchor = 'left', yanchor = 'top')  
 )



plotly가 HTML 텍스트 tag를 일부만 지원함으로써 문자열 스타일링에 한계가 있을듯 하지만 ’’을 사용하는 HTML inline 속성을 지원하기 때문에 CSS의 스타일을 사용하여 문자열의 세부 설정이 가능하다. 다음은 HTML의 inline 속성을 사용하여 제목을 설정하는 코드이다.

p\_title\_1 |> layout(title = list(text = "<span style = 'font-size:15pt'><span style = 'color:red;font-weight:bold;'> 졸업수</span><span style = 'font-size:10pt'> 대비</span> <span style = 'color:blue;font-weight:bold;'>취업률</span></span>",   
 x = 0.1, xanchor = 'left', yanchor = 'top')  
 )



### 범례 설정 : legend

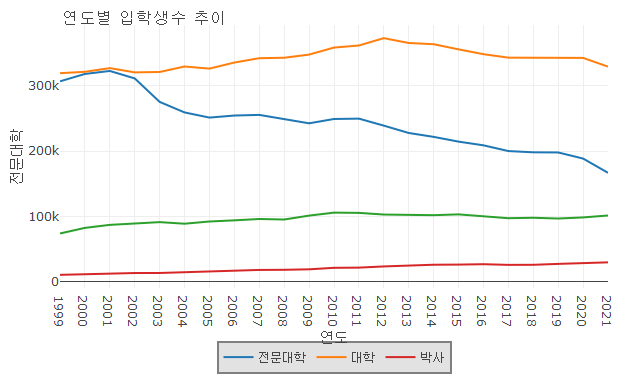
layout()에서 범례 설정과 관련된 type은 앞서 설명한 showlegend와 legend이다.

showlegend는 add\_trace()에서도 설정이 가능한 type인데 add\_trace()에서 설정하면 해당 trace만을 범례에서 제거하고 layout()에서 설정하면 전체 범례를 제가하게 된다.

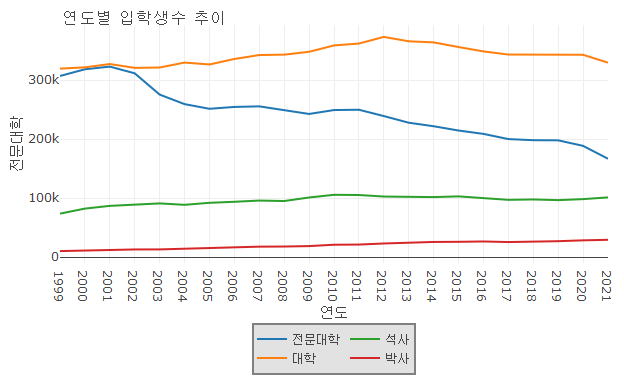
legend는 범례의 세부 설정을 위한 세부 type의 list로 설정된다. legend로 설정이 가능한 주요 세부 type은 다음과 같다.

| type | 설명 | type 값 및 설명 | 세부 type |
| --- | --- | --- | --- |
| bgcolor | 범례의 배경색 설정 |  |  |
| bordercolor, borderwidth | 범례를 둘러싸는 외곽선 색, 두께 설정 |  |  |
| font | 범례의 문자열에 적용되는 문자 속성 설정 |  | color, family, size |
| groupclick | 범례 아이템을 클릭할 때 범례 그룹의 동작 설정 | ’toggleitem’은 범례에서 클릭한 항목의 표시를 토들, ’togglegroup’은 범례에서 클릭한 항목 그룹의 표시를 토글 |  |
| grouptitlefont | 범례 아이템 그룹의 제목과 제목 설정 | 범례 그룹 제목과 그 형태 설정 | color, family, size |
| itemclick | 범례 아이템을 클릭 할 때의 동작 설정 | ’toggle’은 범례에서 클릭한 항목의 표시를 토글, ’toggleothers’는 클릭한 항목만을 보이게 설정, “FALSE”는 범례 항목 클릭 동작을 비활성화 |  |
| itemdoubleclick | 범례 아이템을 더블 클릭 할 때의 동작 설정 | ’toggle’은 범례에서 더블 클릭한 항목의 표시를 토글, ’toggleothers’는 더블 클릭한 항목만을 보이게 설정, “FALSE”는 범례 항목 더블 클릭 동작을 비활성화 |  |
| orientation | 범례 표시 방향의 설정 | ’v’는 세로형태, ’h’는 가로 형태로 표시 |  |
| title | 범례 제목 설정 |  | font(color, family, size), side, text |
| traceorder | 범례 순서 설정 | ‘reversed’, ‘grouped’, ‘normal’의 조합(’+’)  “normal”은 trace의 추가와 동일한 순서, “reversed”는 ’normal’과 반대 순서, ’grouped’는 ’legendgroup’이 설정된 경우 그룹의 순서로 표시 |  |
| x, y, xanchor, yanchor | 범례 위치 설정 |  |  |

p\_title\_line <- df\_입학자 |> filter(지역 == '전체') |> plot\_ly()  
  
p\_title\_2 <- add\_trace(p\_title\_line, type = 'scatter', mode = 'lines',   
 x = ~연도, y = ~전문대학, name = '전문대학')  
  
p\_title\_2 <- p\_title\_2 |> add\_trace(type = 'scatter', mode = 'lines',  
 x = ~연도, y = ~일반대학, name = '대학')  
  
p\_title\_2 <- p\_title\_2 |> add\_trace(type = 'scatter', mode = 'lines',   
 x = ~연도, y = ~석사, name = '석사', showlegend = FALSE)  
  
p\_title\_2 <- p\_title\_2 |> add\_trace(type = 'scatter', mode = 'lines',   
 x = ~연도, y = ~박사, name = '박사')  
  
p\_title\_2 |> layout(title = list(text = '연도별 입학생수 추이', x = 0.1,   
 xanchor = 'left', yanchor = 'top'),  
 legend = list(bgcolor = "#E2E2E2", bordercolor = 'gray',  
 borderwidth = 2, orientation = 'h',   
 x = 0.5, y = -0.2, xanchor = 'center')  
 )



p\_title\_line <- df\_입학자 |> filter(지역 == '전체') |> plot\_ly()  
  
p\_title\_2 <- add\_trace(p\_title\_line, type = 'scatter', mode = 'lines',   
 x = ~연도, y = ~전문대학, name = '전문대학', legendgroup = '학부')  
  
p\_title\_2 <- p\_title\_2 |>   
 add\_trace(type = 'scatter', mode = 'lines',   
 x = ~연도, y = ~일반대학, name = '대학', legendgroup = '학부')  
  
p\_title\_2 <- p\_title\_2 |>   
 add\_trace(type = 'scatter', mode = 'lines',   
 x = ~연도, y = ~석사, name = '석사', legendgroup = '대학원')  
  
p\_title\_2 <- p\_title\_2 |>   
 add\_trace(type = 'scatter', mode = 'lines',   
 x = ~연도, y = ~박사, name = '박사', legendgroup = '대학원')  
  
p\_title\_2 |> layout(title = list(text = '연도별 입학생수 추이', x = 0.1,   
 xanchor = 'left', yanchor = 'top'),  
 legend = list(bgcolor = "#E2E2E2", bordercolor = 'gray',  
 borderwidth = 2, orientation = 'h',   
 x = 0.5, y = -0.25, xanchor = 'center',   
 grouptitlefont = list(color = 'red')  
 )  
 )

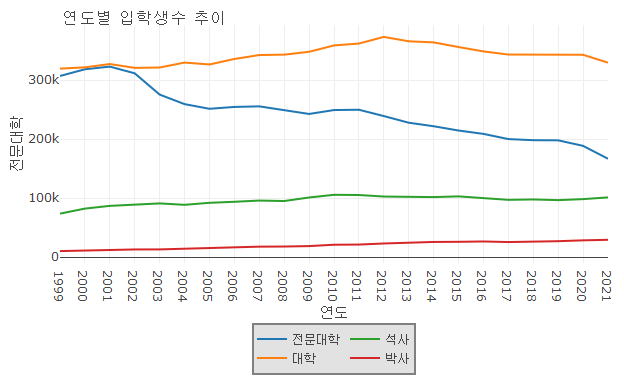


### 여백 설정 : margin

layout()에서 여백의 설정과 관련된 type은 margin이다. margin으로 설정되는 여백은 플롯이 표시되는 면적과 전체 플롯 경계선과의 여백을 말한다. 여백 설정에 사용되는 세부 type은 다음과 같다.

| type | 설명 | type 값 및 설명 | 세부 type |
| --- | --- | --- | --- |
| autoexpand | 여백 설정 값을 사용할지 설정 | TRUE/FALSE |  |
| b | 아래쪽 여백 설정 |  |  |
| l | 왼쪽 여백 설정 |  |  |
| r | 오른쪽 여백 설정 |  |  |
| t | 위쪽 여백 설정 |  |  |
| pad | 플로팅 지역과 축과의 여백 설정 |  |  |

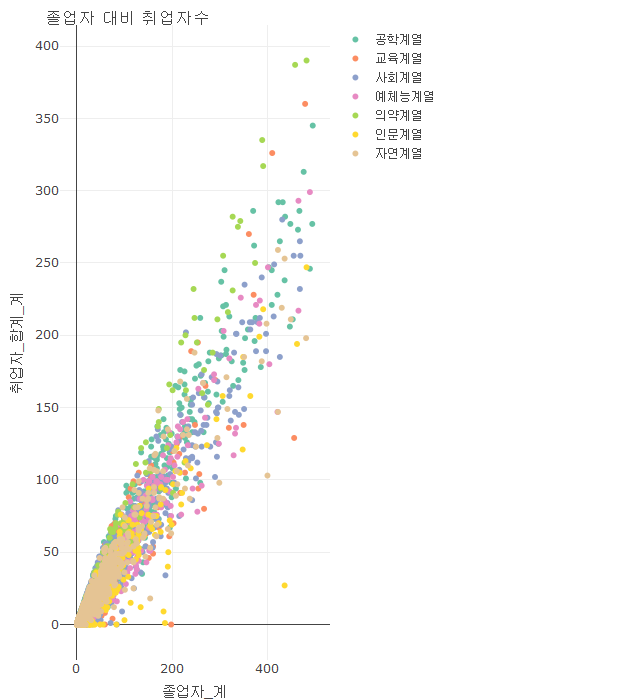
p\_title\_2 |> layout(title = list(text = '연도별 입학생수 추이', x = 0.1,   
 xanchor = 'left', yanchor = 'top'),  
 legend = list(bgcolor = "#E2E2E2", bordercolor = 'gray',  
 borderwidth = 2, orientation = 'h',   
 x = 0.5, y = -0.25, xanchor = 'center'),   
 margin = list(autoexpand = TRUE)  
 )



### 크기 설정 : size

layout()에서 전체 플롯 크기의 설정을 위해서는 autosize, height, width의 세 가지 type이 사용된다. autosize는 사용자가 정의하지 않은 레이아웃의 너비 또는 높이를 자동적으로 설정하는 논리값을 설정한다. height와 width는 전체 플롯 레이아웃의 높이와 너비를 설정하는 type으로 기본값이 450과 700이다.

p\_title\_1 <- p\_title\_marker |>  
 add\_trace(type = 'scatter', mode = 'markers', x = ~졸업자\_계,   
 y = ~취업자\_합계\_계, color = ~대계열)  
  
p\_title\_1 |> layout(title = list(text = '졸업자 대비 취업자수', x = 0.1,   
 xanchor = 'left', yanchor = 'top'),   
 width = 450, height = 700)



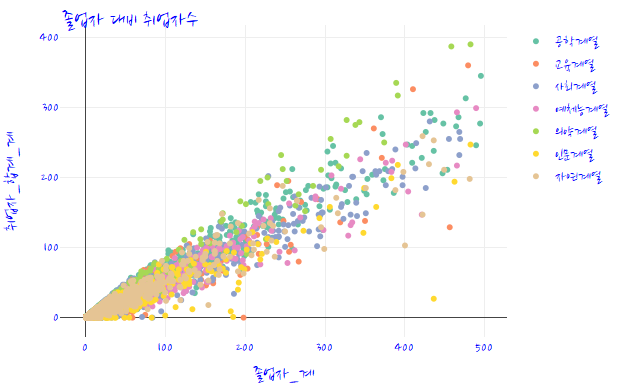
### 폰트 설정 : font

layout()에서 플롯 전체적인 문자열의 설정과 관련된 type은 font이다. font는 다음과 같은 세 개의 세부 type을 설정할 수 있다.

| type | 설명 | type 값 및 설명 | 세부 type |
| --- | --- | --- | --- |
| color | 플롯 전체 문자 색 설정 |  |  |
| family | 플롯 전체 문자 폰트 설정 |  |  |
| size | 플롯 전체 문자 크기 설정 |  |  |

전체적인 폰트의 설정은 다음과 같이 설정할 수 있다. ggplot2에서는 한글 폰트의 설정에 다소 어려움이 있었지만 plotly에서는 family type에 바로 폰트 이름을 설정하면 쉽게 한글 폰트를 사용할 수 있다.

p\_title\_1 |> layout(title = list(text = '졸업자 대비 취업자수', x = 0.1,   
 xanchor = 'left', yanchor = 'top'),   
 font = list(family = "나눔손글씨 붓", color = 'blue', size = 15)  
 )



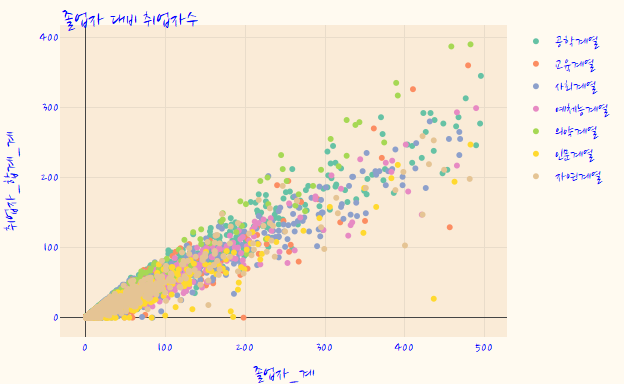
### 색 설정 : color

layout()에서 플롯의 배경색이나 플롯에서 사용되는 전반적인 색 스케일을 설정하는 type은 다음과 같다.

| type | 설명 | type 값 및 설명 | 세부 type |
| --- | --- | --- | --- |
| paper\_bgcolor | 플롯이 그려지는 용지의 배경색을 설정 |  |  |
| plot\_bgcolor | x축과 y축 사이의 플로팅 영역의 배경색 설정 |  |  |
| colorscale | 컬러 스케일의 설정 |  | diverging, sequential, sequentialminus |
| colorway | 기본 컬러 벡터 설정 |  |  |

plotly에서 사용하는 색의 설정은 색 이름 설정, 16진수로 설정된 RGB값 설정, rgb() 함수를 사용한 RGB값의 설정 세 가지 방법이 주로 사용된다. 이외에도 색조, 채도, 명도(lightness)를 사용하는 hsl()을 사용하는 방법, 색조, 채도, 명도(value)를 사용하는 hsv()를 사용하는 방법도 있다. plotly에서 사용이 가능한 색 이름은 W3.org에서 제공하는 CSS 색 이름을 사용한다.[[5]](#footnote-79)

p\_title\_1 |> layout(title = list(text = '졸업자 대비 취업자수', x = 0.1,   
 xanchor = 'left', yanchor = 'top'),   
 font = list(family = "나눔손글씨 붓", color = 'blue', size = 15),   
 paper\_bgcolor = 'floralwhite', plot\_bgcolor = 'antiquewhite'  
 )



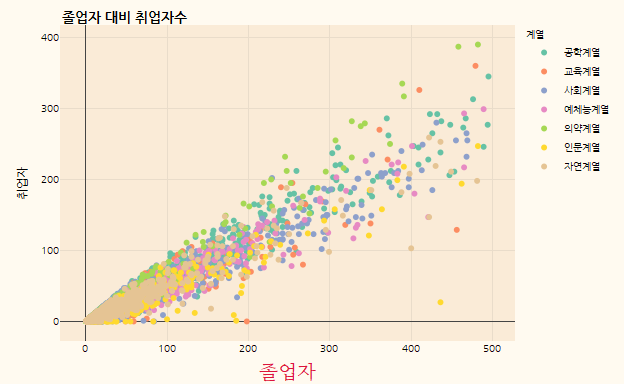
### 축 설정 : xaxis, yaxis

plotly에서 축은 trace type에 따라 매우 다른 이름으로 불린다. X, Y축을 사용하는 2차원의 데카르트 좌표계에서는 ‘axis’, 3차원 trace에서는 ‘scene’, 극 좌표계에서는 ‘polar’, 삼각축 좌표계(ternary)에서는 ‘ternary’, 지형(Geo) 좌표계에서는 ‘geo’, Mapbox 좌표계에서는 ‘mapbox’, 색 좌표계에서는 ’coloraxis’로 사용된다. 이 절에서는 이 중 2차원 데카르트 좌표계에서 사용되는 xaxis와 yaxis를 위주로 설명하겠다.

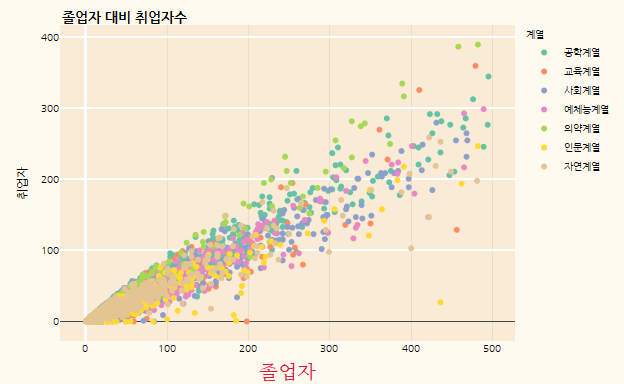
layout()에서 X축의 설정과 관련된 type은 xaxis이다. xaxis에서 제공하는 주요 type은 다음과 같다.

| type | 설명 | type 값 및 설명 | 세부 type |
| --- | --- | --- | --- |
| title | 축 이름 설정 | 예외적으로 title에 세부 list를 설정하지 않고 직접 문자열로 축이름 설정 가능 | font(color, family, size), standoff, text |
| type | 축 유형 설정 | ’linear’은 선형 축, ’log’은 로그 변환 축, ’date’은 날짜형 축, ’category’은 팩터형 축, ’multicategory’은 다중 팩터형 축 설정 |  |
| range | 축 범위 설정 | 축 범위가 지정된 list 설정 |  |
| rangemode | 축 범위의 특성 설정 | ’normal’이면 입력 데이터의 최대, 최소값으로 범위를 설정, ’tozero’이면 입력 데이터에 관계없이 범위가 0부터 시작, ’non-negative’이면 음수 범위는 제거됨 |  |
| linecolor, linewidth | 축 선 설정 | ’linecolor’는 축 선 색, ’linewidth’는 축 선 두께 설정 |  |
| zeroline, zerolinecolor, zerolinewidth | 0 선 설정 | ’zeroline’은 TRUE/FALSE, ’zerolinecolor’는 색, ’zerolinewidth’는 두께 설정 |  |
| position | 축 선 위치 설정 | 0부터 1까지 상대 위치 설정 |  |
| gridcolor, gridwidth | 눈금선 설정 | ’gridcolor’는 눈금선 색, ’gridwidth’는 눈금선 두께 설정 |  |
| autotypenumber | 축에 설정되는 값이 문자형일때 숫자형으로 변환할지를 결정 | ’convert types’은 문자형 수치를 숫자형으로 변환하고 ’strict’는 문자형 수치를 그대로 사용 |  |
| ticks | 눈금자를 어느쪽으로 그릴지 설정 | ’outside’는 플롯 바깥쪽, ’inside’는 플롯 안쪽으로 눈금자를 그림 |  |
| tickangle, ticklen, tickcolor | 눈금자의 각도, 길이, 색 설정 | ’tickangle’은 눈금자 각도, ’ticklen’은 눈금자 길이, ’tickcolor’는 눈금자 색 설정 |  |
| tick0, dtick, nticks | 눈금자의 시작점, 눈금자의 간격, 눈금자의 개수 설정 | ’tick0’는 눈금자가 시작하는 위치, ’dtick’은 눈금자의 간격, ’nticks’는 눈금자의 개수 설정 |  |
| tickvals, ticktext | 눈금자의 위치, 표시문자 설정 | ’tickvals’에 설정된 수치 벡터나 리스트의 위치에 눈금자 설정, ’ticktext’는 ’tickvalues’에 설정된 위치에 표기할 문자열 설정 |  |
| showgrid, showline, showticklabel | 눈금선, 축 선, 눈금자 라벨의 표시 여부 설정 | ’showgrid’는 눈금선, ’showline’은 축 선, ’showticklabel’은 눈금자 라벨의 표시 여부를 설정하는 TRUE/FALSE |  |
| tickprefix, ticksuffix | 눈금자 라벨의 접두어, 접미어 설정 | ’tickprefix’는 눈금자 라벨의 접두어, ’ticksuffix’는 접미어 설정 |  |

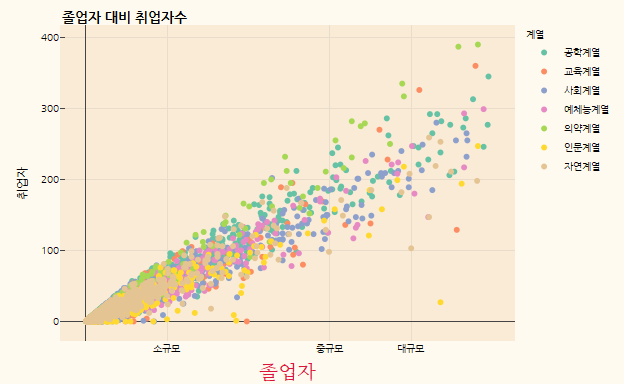
p\_title\_1 |> layout(title = list(text = '<b>졸업자 대비 취업자수</b>', x = 0.1,   
 xanchor = 'left', yanchor = 'top', size = 15),   
 font = list(family = "나눔고딕", color = 'black', size = 10),   
 legend = list(title = list(text = '계열', side = 'top')  
 ),   
 paper\_bgcolor = 'floralwhite', plot\_bgcolor = 'antiquewhite',   
 xaxis = list(title = list(text = '졸업자',   
 font = list(family = '나눔명조', size = 20, color = 'Crimson')  
 )  
 ), ## 정상적 방법   
 yaxis = list(title = '취업자') ## 약식 방법  
 )



p\_title\_1 |> layout(title = list(text = '<b>졸업자 대비 취업자수</b>', x = 0.1,   
 xanchor = 'left', yanchor = 'top', size = 15),   
 font = list(family = "나눔고딕", color = 'black', size = 10),   
 legend = list(title = list(text = '계열', side = 'top')  
 ),   
 paper\_bgcolor = 'floralwhite', plot\_bgcolor = 'antiquewhite',   
 xaxis = list(title = list(text = '졸업자', font = list(family = '나눔명조', size = 20, color = 'Crimson')  
 ), ## 정상적 방법  
 zerolinewidth = 3, zerolinecolor = 'white'),   
 yaxis = list(title = '취업자', gridcolor = 'white', gridwidth = 2) ## 약식 방법  
 )



p\_title\_1 |> layout(title = list(text = '<b>졸업자 대비 취업자수</b>', x = 0.1,   
 xanchor = 'left', yanchor = 'top', size = 15),   
 font = list(family = "나눔고딕", color = 'black', size = 10),   
 legend = list(title = list(text = '계열', side = 'top')),   
 paper\_bgcolor = 'floralwhite', plot\_bgcolor = 'antiquewhite',   
 xaxis = list(title = list(text = '졸업자',   
 font = list(family = '나눔명조', size = 20, color = 'Crimson')  
 ),   
 tickmode = 'array',   
 ticktext = c('소규모', '중규모', '대규모'),  
 tickvals = c(100, 300, 400)  
 ), ## 정상적 방법   
 yaxis = list(title = '취업자', ticks="inside", tick0 = 100, dtick = 100) ## 약식 방법  
 )



### 주석 설정 : annotations

시각화 그래프를 만들때 시각화를 만드는 목적에 따라 시각화 그래프에 데이터에 대한 추가적인 설명이 들어가는 경우가 많다. 이와 같이 시각화를 만드는 사용자가 강조하고자 하거나 추가적으로 설명하고자 할때 사용하는 것이 주석(annotation)이다. plotly에서는 layout()에서 선과 화살표, 문자열을 사용하여 주석을 넣는 것이 가능하고 add\_annotation()을 사용할 수도 있다. layout()을 사용할 때는 annotations type을 사용하여 설정한다.

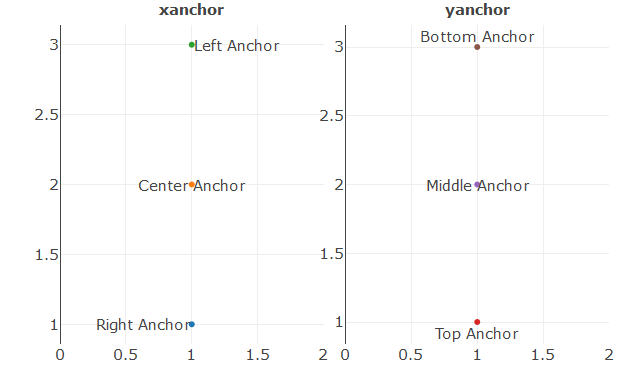
| type | 설명 | type 값 및 설명 | 세부 type |
| --- | --- | --- | --- |
| text | 주석으로 표시할 문자열을 설정 |  |  |
| x, y | 주석을 설정할 X축과 Y축의 좌표를 설정 |  |  |
| ax, ay | 주석이 실제로 위치할 좌표의 상대적 위치 |  |  |
| xanchor | 주석 문자열의 수평 위치 맟춤값 설정 | ’auto’는 자동설정, ’left’는 왼쪽 맞춤, ’center’는 중간 맞춤, ’right’는 오른쪽 맞춤 |  |
| yanchor | 주석 문자열의 수직 위치 맟춤값 설정 | ’auto’는 자동설정, ’top’은 상단 맞춤, ’middle’은 중간 맞춤, ’bottom’은 하단 맞춤 |  |
| xref, yref | x, y, ax, ay의 절대 좌표와 상대 좌표의 참조 기준을 설정 | ‘x’, ’y’는 축 스케일을 참조, ’paper’는 플로팅 영역을 0부터 1까지 정규화하여 참조 |  |
| arrowcolor | 화살촉의 색 설정 |  |  |
| arrowhead | 화살촉의 모양 설정 | 0부터 8까지의 수치 |  |
| arrowside | 화살촉 위치 설정 | ‘end’, ‘start’, ‘none’을’+’로 조합 가능  ’end’는 화살표 직선 끝에 화살촉, ’head’는 화살표 직선 앞에 화살촉을 설정 |  |
| arrowsize, arrowwidth | 화살촉의 크기, 너비 설정 |  |  |
| bgcolor, bordercolor | 주석의 배경색, 주석 외곽선 색 설정 |  |  |
| borderpad, borderwidth | 패딩(주석과 외곽선까지의 여백), 주석 외곽선 너비 설정 |  |  |
| font | 주석의 문자 형태 설정 |  | color, family, size |

text는 주석으로 표시할 문자열을 설정하는 type이다. text는 주석의 표시에 있어 가장 중요한 type이고 필수적인 type이다. text는 문자열을 꾸밀수 있는 HTML 태그 중 줄 바꿈(), 굵은 글씨(), 기울임 글씨(), 하이퍼링크()를 사용할수 있고 , , 도 사용이 가능하다. 특히 을 사용하면 CSS 스타일을 사용할 수 있어 문자열을 세부적으로 꾸밀 수 있다.

x, y는 주석을 설정할 X축과 Y축의 좌표를 설정하는 type이다. 주석은 x와 y에 변수를 매핑해서 사용하거나 하나하나 표시할 수 있다. 따라서 x, y에 문자열이 표시될 좌표를 설정하거나 벡터나 변수를 매핑하여 주석을 표시할 수 있다.

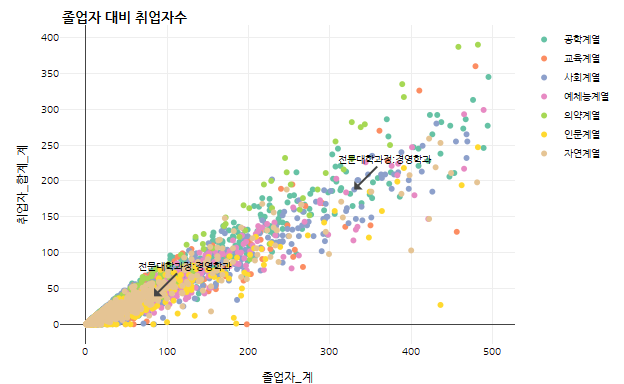
ax와 ay는 x, y에 설정된 좌표에 대한 주석이 실제로 위치할 좌표의 상대적 위치를 설정한다. 이 상대적 위치는 화살표로 표시되는데 ax는 x에 설정된 좌표에서의 거리를 설정하고 ay는 y에 설정된 좌표에서의 거리를 설정한다. ay 설정에 하나 주의할 점은 Y축의 양의 방향은 아래에서 위로 향하지만 ay의 양의 방향은 위에서 아래를 향하기 때문에 아래쪽에 위치한다.

x, y, ax, ay에 의해 결정된 좌표에 text로 설정한 문자열이 표시된다. 이 문자열의 위치 정렬을 설정하는 type이 xanchor와 yanchor이다. xanchor는 ‘auto’, ‘left’, ‘center’, ‘right’ 중에 하나의 값을 가지는데 x에 설정된 좌표값이 전체 문자열의 왼쪽인지, 중앙인지, 오른쪽인지를 설정한다. 또 yanchor는 ‘auto’, ‘top’, ‘middle’, ‘bottom’ 중에 하나의 값을 가지고 y에 설정된 자표값이 전체 문자열의 위쪽인지, 중앙인지, 아래쪽인지를 설정한다.

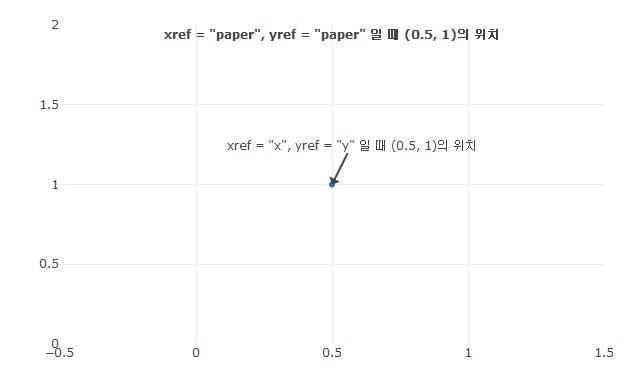


xref와 yref는 x, y, ax, ay의 절대 좌표와 상대 좌표의 참조 기준을 설정한다. xref, yref가 ‘x’, ’y’등의 축으로 설정되면 해당 축의 스케일에 맞게 x, y, ax, ay의 좌표와 거리를 계산하고 ’paper’로 설정되면 전체 플롯 영역을 0부터 1까지로 정규화한 상대적 거리로 계산된다.

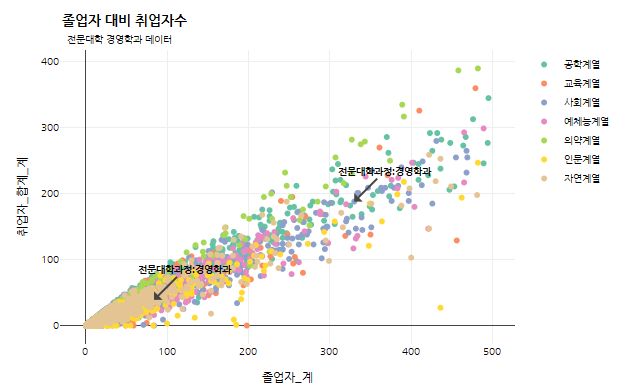
df\_annotation <- df\_취업통계 |> filter(졸업자\_계 < 500, 학과명 == '경영학과') |> select(과정구분, 학과명, 졸업자\_계, 취업자\_합계\_계)  
  
p\_title\_1 |>   
 layout(title = list(text = '<b>졸업자 대비 취업자수</b>', x = 0.1,   
 xanchor = 'left', yanchor = 'top', size = 15),   
 font = list(family = "나눔고딕", color = 'black', size = 10),   
 annotations = list(text = ~paste0(df\_annotation$과정구분, ':', df\_annotation$학과명), x = df\_annotation$졸업자\_계, y = df\_annotation$취업자\_합계\_계, ax = 30, ay = -30)  
 )



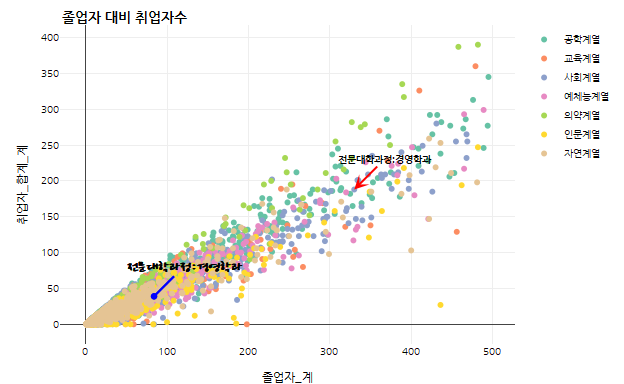
xref와 yref는 x, y, ax, ay의 절대 좌표와 상대 좌표의 참조 기준을 설정한다. xref, yref가 ‘x’, ’y’등의 축으로 설정되면 해당 축의 스케일에 맞게 x, y, ax, ay의 좌표와 거리를 계산하고 ’paper’로 설정되면 전체 플롯 영역을 0부터 1까지로 정규화한 상대적 거리로 계산된다.



df\_annotation <- df\_취업통계 |> filter(졸업자\_계 < 500, 학과명 == '경영학과') |> select(과정구분, 학과명, 졸업자\_계, 취업자\_합계\_계)  
  
p\_title\_1 |>   
 layout(title = list(text = '<b>졸업자 대비 취업자수</b>', x = 0.1,   
 xanchor = 'left', yanchor = 'bottom', size = 15),   
 font = list(family = "나눔고딕", color = 'black', size = 10),   
 annotations = list(text = ~paste0('<b>', df\_annotation$과정구분, ':', df\_annotation$학과명, '</b>'),   
 x = df\_annotation$졸업자\_계, y = df\_annotation$취업자\_합계\_계, ax = 30, ay = -30),   
 margin = list(t = 50)) |>  
 add\_annotations(xref = 'paper', yref = 'paper',   
 text = '전문대학 경영학과 데이터', x = 0.01, y = 1.06,  
 showarrow = FALSE)



p\_title\_1 |>   
 layout(title = list(text = '<b>졸업자 대비 취업자수</b>', x = 0.1,   
 xanchor = 'left', yanchor = 'top', size = 15),   
 font = list(family = "나눔고딕", color = 'black', size = 10),   
 annotations = list(text = ~paste0('<b>', df\_annotation$과정구분[1], ':', df\_annotation$학과명[1], '</b>'), x = df\_annotation$졸업자\_계[1], y = df\_annotation$취업자\_합계\_계[1], ax = 30, ay = -30,   
 arrowhead = 3, arrowcolor = 'red', arrowsize = 1.2)  
 ) |>  
 add\_annotations(text = ~paste0('<b>', df\_annotation$과정구분[2], ':', df\_annotation$학과명[2], '</b>'), x = df\_annotation$졸업자\_계[2], y = df\_annotation$취업자\_합계\_계[2], ax = 30, ay = -30,   
 arrowhead = 6, arrowcolor = 'blue', arrowsize = 0.8,   
 font = list(family = '나눔손글씨 펜', size = 15))



### 도형 설정 : shapes

데이터를 설명하기 위해 넣는 주석은 주로 문자열에 국한된다. 하지만 데이터를 설명하기 위해서는 선, 네모, 원 등의 다양한 도형을 사용할 필요가 있다. 이와 같이 데이터를 설명하기 위한 각종 도형을 사용자가 직접 그리기 위해서는 layout()의 shape type을 설정하여 생성할 수 있다. shape type에서 설정가능한 세부 type은 다음과 같다.

| type | 설명 | type 값 및 설명 | 세부 type |
| --- | --- | --- | --- |
| type | 그려질 도형의 형태 설정 | ’circle’은 원, ’rect’는 사각형, ’path’는 SVG path, ’line’은 선 설정 |  |
| name | 그려질 도형의 이름 설정 |  |  |
| x0, x1, y0, y1 | 도형의 X, Y축 시작점(x0, y0)과 끝점(x1, y1) 설정 |  |  |
| layer | 그려지는 도형 레이어의 위치를 설정 | ’above’는 다른 도형보다 위, ’below’는 다론 도형보다 아래에 위치 |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

add\_segments

## function (p, x = NULL, y = NULL, xend = NULL, yend = NULL, ...,   
## data = NULL, inherit = TRUE)   
## {  
## if (inherit) {  
## x <- x %||% p$x$attrs[[1]][["x"]]  
## xend <- xend %||% p$x$attrs[[1]][["xend"]]  
## y <- y %||% p$x$attrs[[1]][["y"]]  
## yend <- yend %||% p$x$attrs[[1]][["yend"]]  
## }  
## if (is.null(x) || is.null(y) || is.null(xend) || is.null(yend)) {  
## stop("Must supply `x`/`y`/`xend`/`yend` attributes",   
## call. = FALSE)  
## }  
## add\_trace\_classed(p, x = x, y = y, xend = xend, yend = yend,   
## class = "plotly\_segment", type = "scatter", mode = "lines",   
## ..., data = data, inherit = inherit)  
## }  
## <bytecode: 0x000000002bfd1f98>  
## <environment: namespace:plotly>

1. 앞의 예에서 범례에 나타난 오류는 ggplotly()에서 다중 범례(Multiple Legend)의 변환 과정에서 발생하는 문제로 ggplot의 범례를 제거하고 plotly의 범례를 생성함으로써 해결가능하다. [↑](#footnote-ref-22)
2. <https://plotly.com/r/reference/#scatter-marker-colorbar> [↑](#footnote-ref-27)
3. <https://plotly.com/r/reference/scatter/> [↑](#footnote-ref-41)
4. <https://github.com/d3/d3-format/tree/v1.4.5#d3-format> [↑](#footnote-ref-49)
5. <https://www.w3schools.com/cssref/css_colors.asp> [↑](#footnote-ref-79)