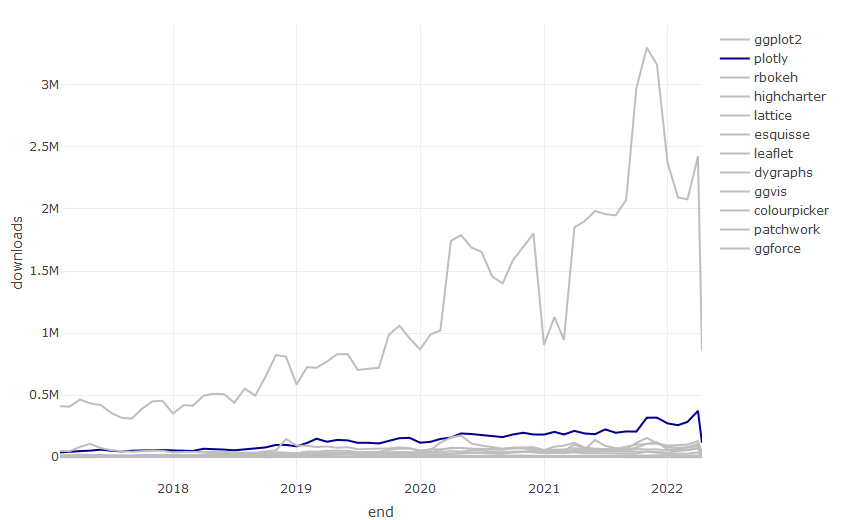
1. plotly 시각화 만들기

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 제공된다. 다음의 R 그래픽 패키지 다운로드 현황에서 보듯이 여전히 R에서 가장 많이 사용되고 있는 그래픽 패키지는 ggplot2이지만 최근 들어 그 사용량이 줄고 있고 R에서 많이 사용되던 lattice패키지가 점차 줄고 있다. 반면 plotly는 2021년 하반기부터 다운로드가 늘고 있고 다른 동적 그래픽 패키지에 비해서는 압도적인 다운로드 수를 보인다.

library(dlstats)  
  
#x <- cran\_stats(c("ggplot2", "plotly", "rbokeh", "highcharter"))  
x1 <- cran\_stats(c("ggplot2", "plotly", "rbokeh", "highcharter", 'lattice', 'esquisse', 'leaflet', 'dygraphs', 'ggvis', 'colourpicker', 'patchwork', 'ggforce'))  
  
x1 |> plot\_ly() |> #filter(package != 'ggplot2') |>   
 add\_lines(x = ~end, y = ~downloads, color = ~package, colors = ~ifelse(package == 'plotly', 'darkblue', 'gray'))



실행결과 2- . R의 주요 그래픽 패키지 다운로드 현황

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.

여기서는 R에서 동적 시각화로 많이 사용되는 plotly 패키지를 사용하여 데이터를 시각화하는 방법을 알아본다. plotly를 사용하여 데이터 시각화를 하기 위해서 R에서 가장 많이 사용되는 데이터 시각화 패키지인 ggplot2에 대한 사용법을 미리 익혀두면 좋겠지만 꼭 사용하지 못해도 plotly를 사용하여 시각화를 만드는데 큰 문제는 없다.

# 예제 데이터 Import와 전처리

먼저 plotly를 사용하여 시각화를 실습하는데 필요한 데이터 셋 두 가지를 전처리 하겠다.

## Covid19 데이터 셋

첫번째 데이터 셋은 2020년 1월부터 기록된 전세계 국가의 코로나19 발병 관련 데이터이다. 이 데이터는 Github에서 다양한 전세계 데이터를 배포하는 ’Open World in Data’에서 제공하는 ’COVID-19 Dataset by Our World in Data’를 사용한다.[[1]](#footnote-20) 이 데이터는 온라인으로 매일 업데이트되기 때문에 다운로드 시점에 따라 시각화 결과가 책과 다소 달라질 수 있다.[[2]](#footnote-22)

OWID에서 제공하는 데이터를 활용하여 4개의 데이터 셋을 만든다. 첫 번째 데이터 셋은 OWID에서 제공하는 원본 데이터를 가져와서 R에 로딩하는 원본 데이터 셋으로 ‘df\_covid19’ 데이터프레임에 저장한다. ‘df\_covid19’ 데이터 프레임은 2020년 1월 1일부터 기록되어 있기 때문에 데이터가 다소 많다. 따라서 이 데이터 중에 최근 100일간의 데이터와 한국과 각 대륙 데이터만을 필터링한 데이터 셋을 두 번째 데이터 셋인 ‘df\_covid19\_100’ 데이터프레임으로 저장한다. 세 번째 데이터 셋은 100일간의 데이터 셋을 넓은 형태의 데이터 셋으로 변환한 ‘df\_covid19\_100\_wide’로 저장한 데이터프레임이다. 네 번쨰는 2년 넘게 기록된 Covid19 데이터 셋의 각종 데이터를 국가별 요약 통계치를 산출하여 저장한 ’df\_covid19\_stat’ 데이터프레임이다.

## 데이터 전처리를 위한 패키지 설치 및 로딩  
if(!require(readr)) {  
 install.packages('readr')  
 library(readr)  
}  
  
if(!require(lubridate)) {  
 install.packages('lubridate')  
 library(lubridate)  
}  
  
if(!require(tidyverse)) {  
 install.packages('tidyverse')  
 library(tidyverse)  
}  
  
## 1. covid19 원본 데이터 셋 로딩  
## covid19 데이터 로딩(파일을 다운로드 받은 경우)  
# df\_covid19 <- read\_csv(file = "데이터저장경로/owid-covid-data.csv",  
# col\_types = cols(Date = col\_date(format = "%Y-%m-%d")  
# )  
# )  
## covid19 데이터 로딩(온라인에서 바로 로딩할 경우)  
df\_covid19 <- read\_csv(file = "https://covid.ourworldindata.org/data/owid-covid-data.csv",  
 col\_types = cols(Date = col\_date(format = "%Y-%m-%d")  
 )  
 )  
## 2. 전체 데이터셋 중 최근 100일간의 데이터를 필터링한 df\_covid19\_100 생성  
df\_covid19\_100 <- df\_covid19 |>   
 ## 한국 데이터와 각 대륙별 데이터만을 필터링  
 filter(iso\_code %in% c('KOR', 'OWID\_ASI', 'OWID\_EUR', 'OWID\_OCE', 'OWID\_NAM', 'OWID\_SAM', 'OWID\_AFR')) |>  
 ## 읽은 데이터의 마지막 데이터에서 100일전 데이터까지 필터링  
 filter(date >= max(date) - 100) |>  
 ## 국가명을 한글로 변환  
 mutate(location = case\_when(  
 location == 'South Korea' ~ '한국',   
 location == 'Asia' ~ '아시아',   
 location == 'Europe' ~ '유럽',   
 location == 'Oceania' ~ '오세아니아',   
 location == 'North America' ~ '북미',   
 location == 'South America' ~ '남미',   
 location == 'Africa' ~ '아프리카')) |>  
 ## 국가 이름의 순서를 설정   
 mutate(location = fct\_relevel(location, '한국', '아시아', '유럽', '북미', '남미', '아프리카', '오세아니아')) |>  
 ## 날짜로 정렬  
 arrange(date)  
  
  
## 3. df\_covid19\_100을 한국과 각 대륙별열로 배치한 넓은 형태의 데이터프레임으로 변환  
df\_covid19\_100\_wide <- df\_covid19\_100 |>  
 ## 날짜, 국가명, 확진자와, 백신접종완료자 데이터만 선택  
 select(date, location, new\_cases, people\_fully\_vaccinated\_per\_hundred) |>  
 ## 열 이름을 적절히 변경  
 rename('date' = 'date', '확진자' = 'new\_cases', '백신접종완료자' = 'people\_fully\_vaccinated\_per\_hundred') |>  
 ## 넓은 형태의 데이터로 변환  
 pivot\_wider(id\_cols = date, names\_from = location,   
 values\_from = c('확진자', '백신접종완료자')) |>  
 ## 날짜로 정렬  
 arrange(date)  
  
## 4. covid19 데이터를 국가별로 요약한 df\_covid19\_stat 생성  
df\_covid19\_stat <- df\_covid19 |>   
 group\_by(iso\_code, continent, location) |>  
 summarise(인구수 = max(population, na.rm = T),   
 인당GDP = max(gdp\_per\_capita, na.rm = T),  
 전체확진자수 = sum(new\_cases, na.rm = T),  
 전체사망자수 = sum(new\_deaths, na.rm = T),   
 십만명당중환자실 = last(icu\_patients\_per\_million),  
 재생산지수 = last(reproduction\_rate),  
 봉쇄지수 = max(stringency\_index),   
 전체검사자수 = max(total\_tests, na.rm = T),   
 신규검사자수 = sum(new\_tests, na.rm = T),  
 전체백신접종자수 = max(total\_vaccinations, na.rm = T),  
 백신접종자완료자수 = max(people\_fully\_vaccinated, na.rm = T),  
 부스터접종자수 = max(total\_boosters, na.rm = T),  
 인구백명당백신접종완료률 = max(people\_fully\_vaccinated\_per\_hundred, na.rm = T),  
 인구백명당부스터접종자수 = max(total\_boosters\_per\_hundred, na.rm = T)  
 ) |>   
 ungroup() |>  
 mutate(십만명당사망자수 = round(전체사망자수 / 인구수 \*100000, 5),  
 백신접종완료률 = 백신접종자완료자수 / 인구수)  
  
## 여백 설정을 위한 리스트 설정  
margins <- list(t = 50, b = 25, l = 25, r = 25)

## 대학 학과 취업률 데이터 셋

최근 청년층 실업 문제가 사회적 문제로 대두됨에 따라 대학 졸업생의 취업률이 매우 중요하게 활용되고 있는 데이터이다. 이 데이터는 대학 입학을 앞둔 수험생이나 학부모에게 대학 진학을 위한 학과 선택에 중요한 데이터이고 대학 입장에서는 학생들의 진로 지도를 위해 중요하게 사용되는 데이터이다. 이 데이터는 교육통계서비스 홈페이지에서 제공한다.[[3]](#footnote-24)

취업률 데이터 셋은 다음과 같이 데이터를 로딩하고 전처리한다.

df\_취업률 <- read\_excel('d:/R/data/2020년 학과별 고등교육기관 취업통계.xlsx',   
 ## '학과별' 시트의 데이터를 불러오는데,  
 sheet = '학과별',  
 ## 앞의 13행을 제외하고  
 skip = 13,   
 ## 첫번째 행은 열 이름으로 설정  
 col\_names = TRUE,   
 ## 열의 타입을 설정, 처음 9개는 문자형으로 다음 79개는 수치형으로 설정  
 col\_types = c(rep('text', 9), rep('numeric', 79)))  
  
## df\_취업률에서 첫번째부터 9번째까지의 열과 '계'로 끝나는 열을 선택하여 다시 df\_취업률에 저장  
df\_취업률 <- df\_취업률 |>   
 select(1:9, ends\_with('계'), '입대자')  
  
## 랜덤 샘플을 위한 시드 설정  
set.seed(123)  
  
## df\_취업률에서 졸업자가 500명 이하인 학과 2000개 샘플링  
df\_취업률\_2000 <- df\_취업률 |>   
 filter(졸업자\_계 < 500) |>   
 sample\_n(2000)  
  
## 열 이름을 적절히 설정  
names(df\_취업률\_2000)[10:12] <- c('졸업자수', '취업률', '취업자수')

# plotly란?

plotly는 오픈 소스인 JavaScript로 구현된 plotly.js를 기반으로 R에서 생성한 데이터 시각화 객체를 Javascript로 생성해주는 패키지이다[[4]](#footnote-28). plotly는 JavaScript로 구현되기 때문에 plotly 객체는 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.

R의 plotly패키지는 동적 시각화를 제공하면서 출판이 가능한 품질의 시각화를 만든다. plotly로 선 그래프, 산점도, 막대 차트, 박스 플롯, 히스토그램, 히트맵, 서브플롯, 다중 축 및 3D(WebGL 기반) 차트 등의 다양한 시각화를 생성할 수 있는데 생성할 수 있는 시각화의 종류는 ggplot2보다 많다. plotly는 R에서 무료로 사용 가능한 패키지로 python, julia, F#, MATLAB 등에서도 지원되는 그래픽 라이브러리를 제공하고 있다.

## plotly의 내부 표현 구조

R 내부에서 plotly 객체는 plotly.js에서 정의된 JSON 스키마로 저장된다. 이 스키마는 트리 형태로 구성되어 있는데 각 노드는 속성(attribute)로 불리는 값을 가지게 되고 이들 속성들이 모여서 전체 그림(Figure)를 구성한다. 예를 들어 fig |> layout(width = 20)이라는 코드는 ‘layout’ 루트 노드에 하위 노드인 ‘width’ 속성을 20으로 설정하게 된다. 이 속성값은 plotly의 함수(add\_trace(), layout() 등)를 사용하여 노드의 값을 설정하거나 직접 노드의 속성 값을 설정할 수 있다. 예를 들어 앞의 fig |> layout(width = 20)의 경우 fig.layout.width = 20으로 설정할 수 있다. plotly 객체를 생성하기 위해서는 많은 노드의 속성 값이 필요하지만 사용자가 직접 설정하지 않은 속성 값은 기본값으로 설정되거나 plotly가 내부적으로 계산하여 속성 값들이 설정되게 된다.

R에서 plotly 객체를 표현하는데 사용되는 트리의 첫번째 레벨 노드는 ‘data’, ‘layout’, ’frame’의 세 가지가 있다.

## ‘data’ 노드

plotly 스키마에서 사용하는 트리 중 첫 번째 레벨 노드인 ‘data’ 노드는 뒤에서 설명할 ’trace’에 관련한 속성 노드들이 하위 노드로 구성되는데 ’trace’로 설정되는 문자열로 설정되고 해당 ’trace’에 관련된 속성의 이름과 값들로 구성된 리스트로 표현된다.

plotly에서의 ’trace’는 스캐터, 선, 박스 등 약 40개 이상의 type이 제공되고 각각의 ’trace’는 데이터를 표현하는 도형적 요소들을 결정한다. 이 ’trace’는 type 속성으로 설정할 수 있고 trace의 종류에 따라 속성 값을 표현하는 노드들을 list()를 사용하여 리스트로 만들어 설정한다.

각각의 trace는 그 trace 도형 타입과 관련된 하나의 subplot에 표현되고, 파이 trace를 제외한 모든 trace는 단일 범례 요소(Single Legend Entry)를 가진다. 하지만 trace의 종류에 따라 설정 가능한 속성이 달라지기 때문에 각각의 trace마다 설정할 수 있는 속성들을 잘 파악할 필요가 있다.

## ‘layout’ 노드

‘layout’ 노드는 데이터와 관련되지 않는 속성들을 정의하는 노드로써 list()를 사용하여 하위 속성들을 리스트로 만들어 설정해야 한다.

데이터와 관련되지 않는 ‘layout’ 노드에서 설정할 수 있는 속성 값은 플롯 여백, 플롯 제목, 플롯 축 제목, 주석, 도형 등과 관련된 값이다.

## ‘frame’ 노드

마지막 첫 레벨 노드는 ‘frame’ 노드이다. 이 ‘frame’ 노드는 plotly의 애니메이션 기능과 관련된 속성 값을 설정하는 노드이다.

# plotly 시각화 만들기

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.

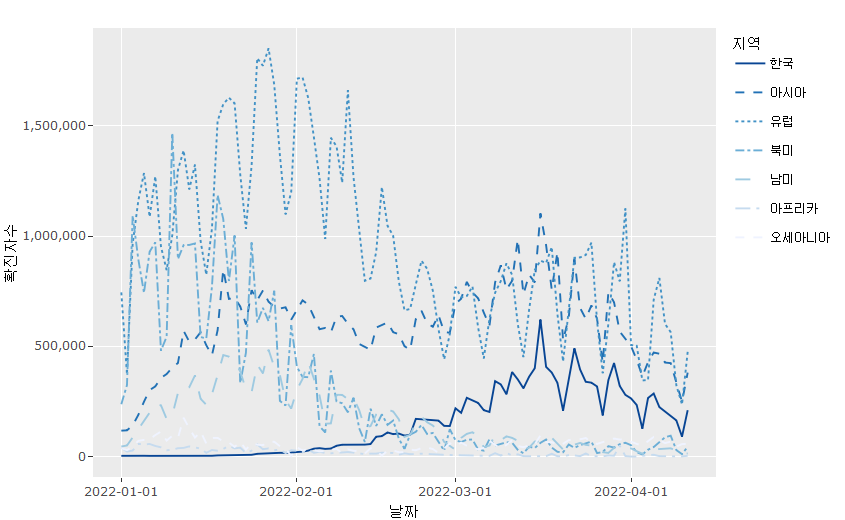
## ggplot 객체의 전환

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.

## plotly 패키지 설치 및 로딩  
if(!require('plotly')) {  
 install.packages('plotly')  
 library(plotly)  
}  
  
library(viridis)  
  
ggplotly <- df\_covid19\_100 |>   
 ## x축이 date, y축이 new\_cases, color가 location으로 매핑된 ggplot 객체 생성  
 ggplot(aes(x = date, y = new\_cases, color = location )) +  
 ## group과 linetype이 location으로 매핑된 geom\_line 레이어 생성  
 geom\_line(aes(group = location, linetype = location)) +  
 ## x축 스케일을 날짜형이고 1개월 단위로 눈금 설정  
 scale\_x\_date(breaks = '1 months') +  
 ## y축의 라벨에 천 단위 구분자 설정  
 scale\_y\_continuous(labels = scales::comma) +   
 ## 색 설정을 RColorBrewer 패키지의 'Set2'로 설정  
 scale\_color\_brewer(palette = 'Blues', direction = -1) +   
 ## x축, y축, linetype, color 이름 설정  
 labs(x = '날짜', y = '확진자수', linetype = '지역', color = '지역')  
  
## ggplot객체를 plotly 객체로 변환  
ggplotly(ggplotly)



실행결과 2- . ggplot2 객체의 전환

앞의 예와 같이 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 시각화 생성

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

### plotly 객체 초기화 : plot\_ly()

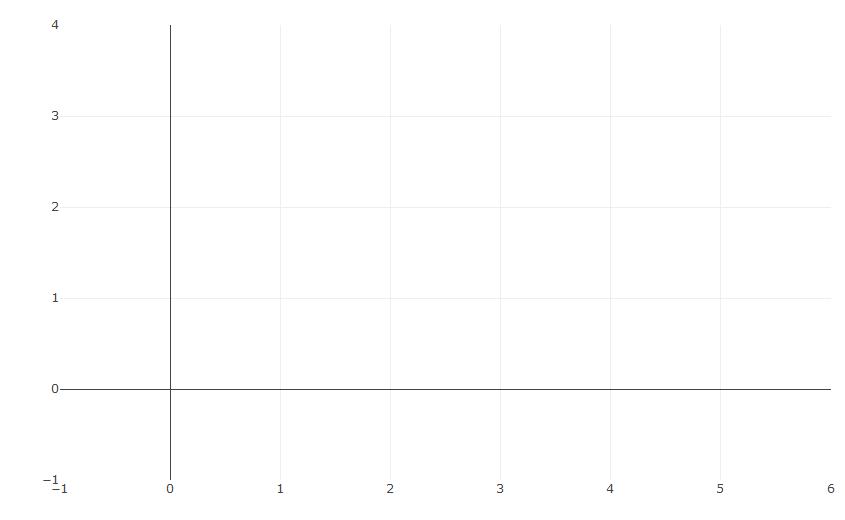
설명한 바와 같이 plotly는 보여지기에 그래픽으로 보여지지만 내부적으로는 plotly.js에서 정의한 데이터 구조 형태로 저장된다. 따라서 plotly객체를 표현하는 데이터 구조(스키마)를 생성하기 위해서는 제일 먼저 plotly 객체의 초기 스키마를 생성해주는 초기화 함수가 필요하다. plotly 객체를 시작하기 위한 초기화 함수는 plot\_ly()이다. plot\_ly()의 주요 용법은 다음과 같다.

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

plot\_ly()는 plotly 객체 생성을 위한 초기 데이터 생성, trace와 속성을 통한 시각화 생성, 하위 trace에 공통적으로 사용될 속성 값의 설정의 세가지 기능을 수행한다.

첫 번쨰 기능은 plot\_ly()의 호출을 통해 plotly.js에서 정의된 plotly 객체 스키마를 생성한다. 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.

## 긴 형태의 100일간 코로나19 데이터를 사용한 plotly 객체 생성  
df\_covid19\_100 |>   
 plot\_ly()

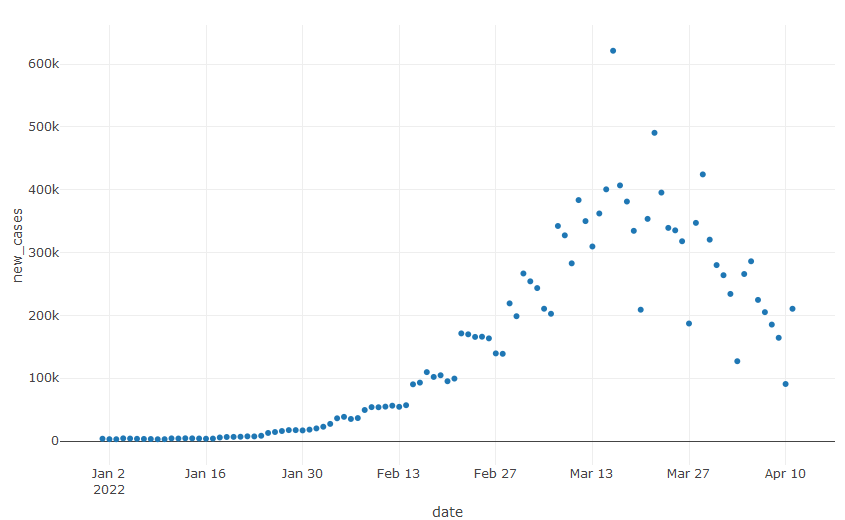


실행결과 2- . plot\_ly()를 사용한 plotly 초기화

두 번째 기능은 plotly 객체 생성을 통한 시각화 생성이다. ggplot2에서 ggplot()을 사용하여 ggplot 객체를 초기화하고 geom\_\*()를 사용하여 기하 요소 레이어를 하나 하나 설정하면서 전체 시각화를 완성한다. plotly도 이와 유사하게 초기 plotly 객체에 여러개의 trace를 추가함으로써 전체 시각화를 완성해 나갈 수 있다. 하지만 plotly()에서는 데이터, trace, trace에 따른 속성까지 설정이 가능하므로 plot\_ly()만을 사용해도 시각화를 완성할 수 있다. 하지만 시각화에서는 하나 이상의 trace가 사용될때가 많기 때문에 뒤에서 설명할 add\_trace()나 add\_\*()를 사용하여 trace를 추가하여 시각화를 완성하는 것이 일반적이다.

plotly를 사용한 시각화에서 속성에 변수나 값을 할당할 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.

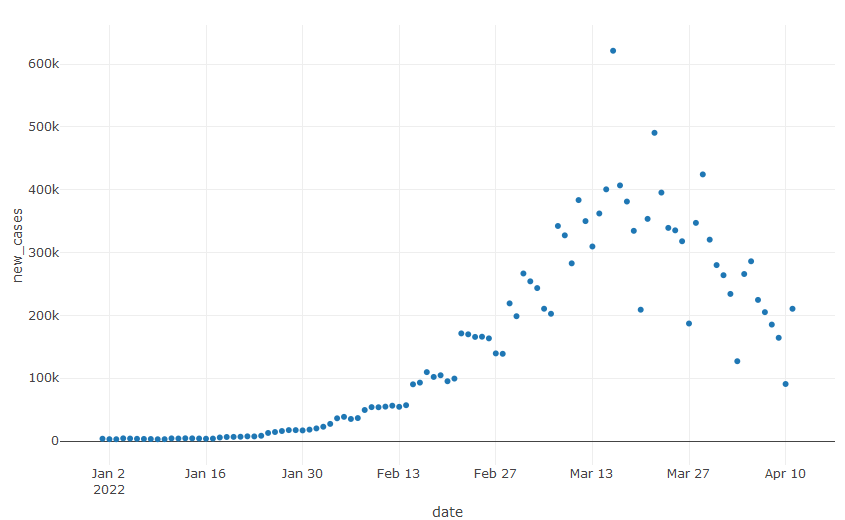
## 긴 형태의 100일 코로나19 데이터에서  
df\_covid19\_100 |>   
 ## 한국 데이터만을 필터링   
 filter(iso\_code == 'KOR') |>  
 ## X축에 data, Y축에 new\_cases를 매핑하여 plot\_ly()로 시각화 생성  
 plot\_ly(x = ~date, y = ~new\_cases)



실행결과 2- . plot\_ly를 사용한 plotly 시각화

세 번째 기능은 plot\_ly 객체의 전반적으로 사용될 속성 값의 설정 기능이다. plot\_ly()를 사용하여 설정된 속성 값은 이후 추가되는 trace에 상속되기 때문에 이 속성 값을 다시 설정하지 않으면 plot\_ly()에서 설정된 속성 값이 사용된다. 이 과정에서 원치않는 속성의 상속을 방지하기 위해 plot\_ly()에 매개변수를 넣지않고 단순히 plotly 객체의 초기화 명령으로 사용도 가능하다.

df\_covid19\_100 |>   
 filter(iso\_code == 'KOR') |>   
 ## plot\_ly()로 시각화를 초기화  
 plot\_ly() |>  
 ## 스캐터 trace 타입으로 X축에 date, Y축에 new\_cases를 매핑한 시각화 생성  
 add\_trace(type = 'scatter', x = ~date, y = ~new\_cases)



실행결과 2- . add\_trace()를 사용한 시각화

plotly 객체에서 가장 중요한 것은 trace의 종류이다. plot\_ly()만을 사용하여 그래프를 완성할 때 trace의 종류를 생략하면 plotly에서 데이터를 파악하여 가장 좋은 trace를 설정해준다.

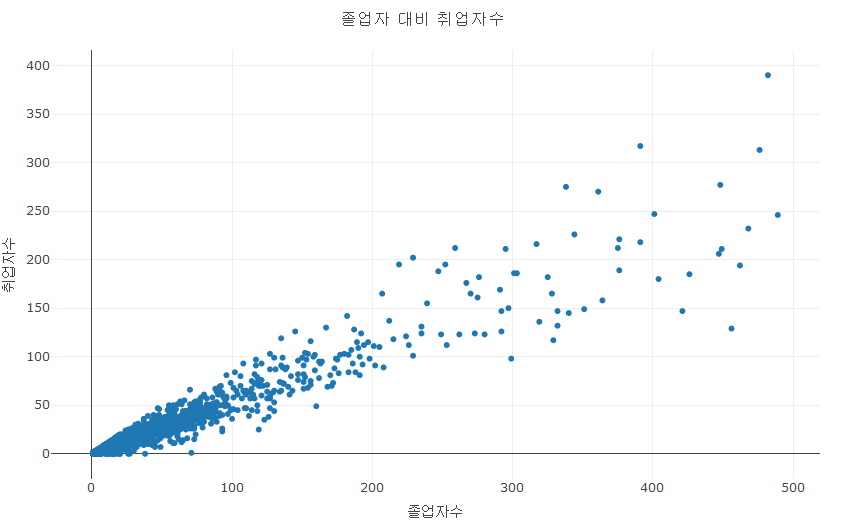
하지만 plotly 사용하여 시각화를 만들때는 trace type부터 시작하여 해당 trace의 세부 속성들을 설정해 가면서 만든다. 앞에서 본 것과 같이 plot\_ly()를 사용하여도 시각화를 완성할 수 있지만 보다 상세하고 섬세한 시각화를 만들기 위해서는 plot\_ly()로 plotly 객체를 위한 스키마를 초기화하고 이 객체에 add\_trace()나 add\_\*()를 사용하여 trace를 추가함으로써 만든다.

plotly의 trace 설정에 공통적으로 사용되는 주요 속성은 다음과 같다.

#### x, y, z

trace 설정에서 가장 기본적으로 설정하는 속성은 x, y, z이다. 이 세 개의 속성은 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.

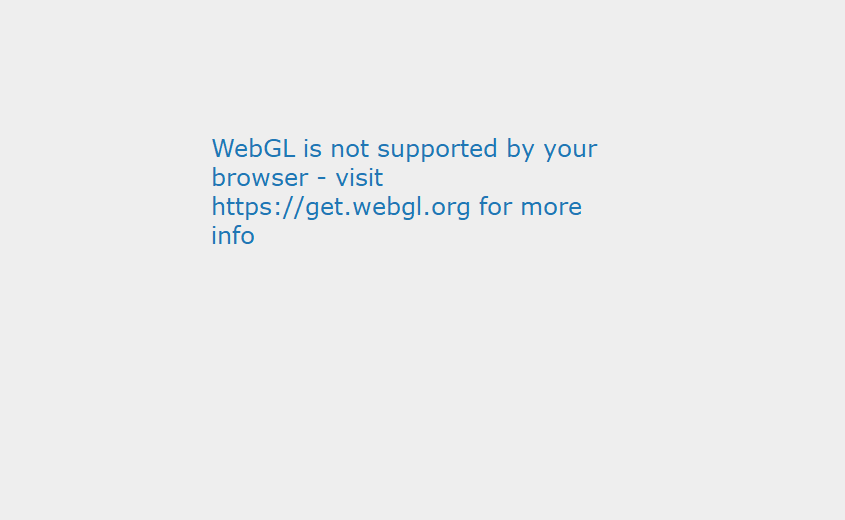
## df\_취업률\_2000에서   
df\_취업률\_2000 |>   
 ## X축은 졸업자수, Y축은 취업자수로 매핑한 plotly 객체 생성  
 plot\_ly(x = ~졸업자수, y = ~취업자수) |>  
 ## 제목과 여백 설정  
 layout(title = '졸업자 대비 취업자수', margin = margins)



실행결과 2- . plotly 기본 산점도 생성

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\_취업률\_2000 |>  
 ## X축은 졸업자수, Y축은 취업자수, Z축은 대계열로 매핑된 plotly 객체 생성  
 plot\_ly(x = ~졸업자수, y = ~취업자수, z = ~대계열)



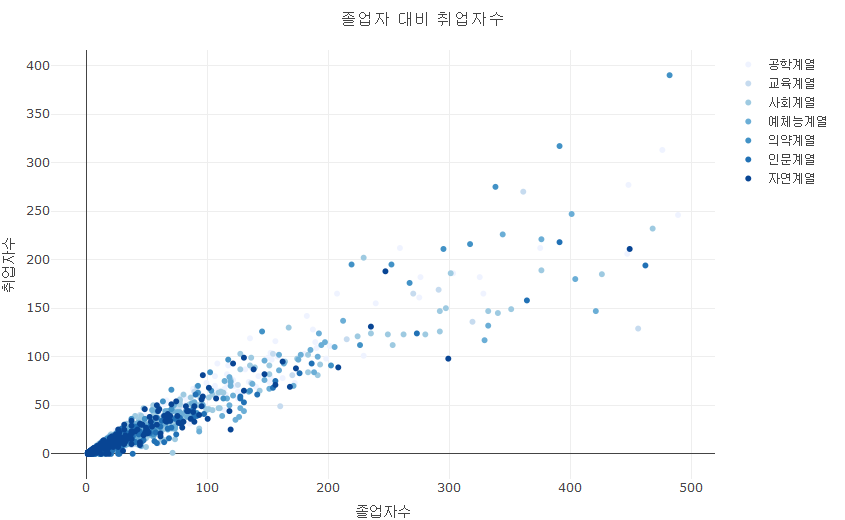
실행결과 2- . 3차원 plotly

#### 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.

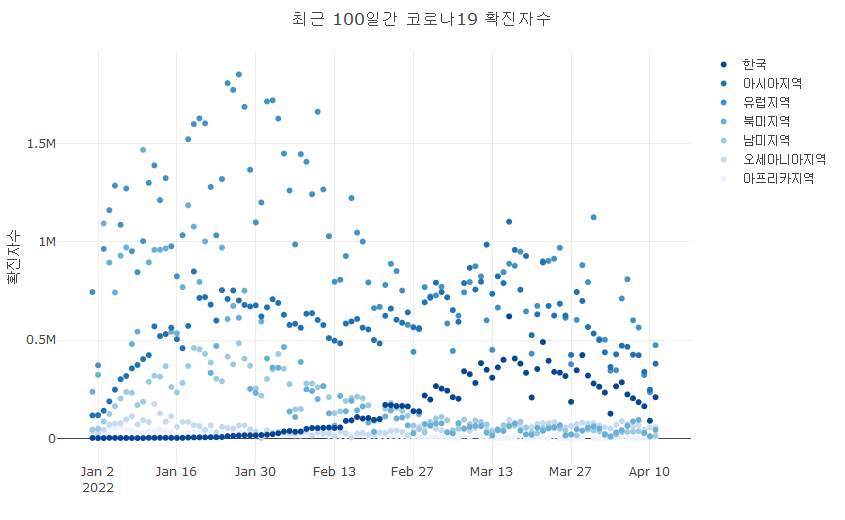
df\_취업률\_2000 |>  
 ## X축은 졸업자수, Y축은 취업자수, name은 대계열로 매핑한 plotly 객체 생성  
 plot\_ly(x = ~졸업자수, y = ~취업자수, name = ~대계열) |>  
 layout(title = '졸업자 대비 취업자수', margin = margins)



실행결과 2- . trace 이름 매핑

만약 범례를 사용자가 원하는 형태로 바꾸려면 범례 아이템을 바꾸는 방법으로 할 수 없고 각각의 개별 데이터에 매핑될 벡터나 열을 만들어 주어야 한다. ggplot2와 같은 정적 시각화는 시각화가 만들어지면 사용자의 활동에 따라 반응이 없지만 동적 시각화는 마우스 포인터를 데이터에 가져가면 해당 데이터에 대한 내용이 표기되어야 한다. 이 값이 범례값과 달라지면 안되기 때문에 범례만 수정할 수 없고 모든 데이터에 1:1로 매핑되는 범례 아이템 이름 벡터가 필요하다.

## 범례로 사용할 문자열 벡터 생성  
legend\_items <- df\_covid19\_100 |>   
 mutate(legend\_name = case\_when(  
 iso\_code == 'KOR' ~ '한국',   
 iso\_code == 'OWID\_ASI' ~ '아시아지역',  
 iso\_code == 'OWID\_EUR' ~ '유럽지역',  
 iso\_code == 'OWID\_NAM' ~ '북미지역',  
 iso\_code == 'OWID\_OCE' ~ '오세아니아지역',  
 iso\_code == 'OWID\_SAM' ~ '남미지역',   
 iso\_code == 'OWID\_AFR' ~ '아프리카지역')) |>  
 select(legend\_name) |>  
 pull()  
  
## 범례 순서 설정을 위한 팩터 설정  
legend\_items <- fct\_relevel(legend\_items, '한국', '아시아지역', '유럽지역', '북미지역', '남미지역', '오세아니아지역', '아프리카지역')  
  
df\_covid19\_100 |>  
 ## X축을 date, Y축을 new\_cases, 범례 이름을 앞서 생성한 문자열 벡터로 매핑   
 plot\_ly(x = ~date, y = ~new\_cases, name = ~legend\_items) |>  
 layout(title = '최근 100일간 코로나19 확진자수',   
 ## X축 이름을 제거  
 xaxis = list(title = list(text = '')),  
 ## Y축 이름을 설정  
 yaxis = list(title = list(text = '확진자수')),  
 ## 여백 설정  
 margin = margins)



실행결과 2- . 범례 이름 변경

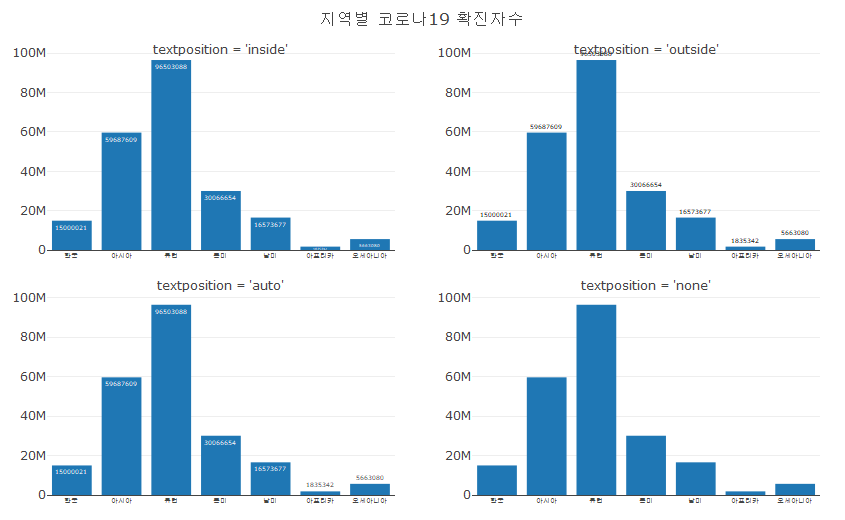
#### text, textposition, texttemplate, textfont

데이터 시각화에서 데이터는 다양한 도형으로 표현되기 때문에 데이터의 정확한 값을 알아보기는 어렵다. 이를 보완하기 위해 데이터 시각화에 데이터 값을 표기하는 경우가 많다. plotly의 많은 trace에서 데이터 값을 표현하기 위해서는 text 속성을 사용하면 간단히 표시할 수 있다. 표시되기를 원하는 변수를 text에 매핑함으로써 X, Y축의 위치나 trace의 type에 적합한 위치에 데이터 값을 표시할 수 있으며 textposition과 texttemplate를 사용하여 표시되는 값의 세부 설정을 변경할 수 있다.

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

textposition는 text의 위치를 설정하는 속성이다. 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.

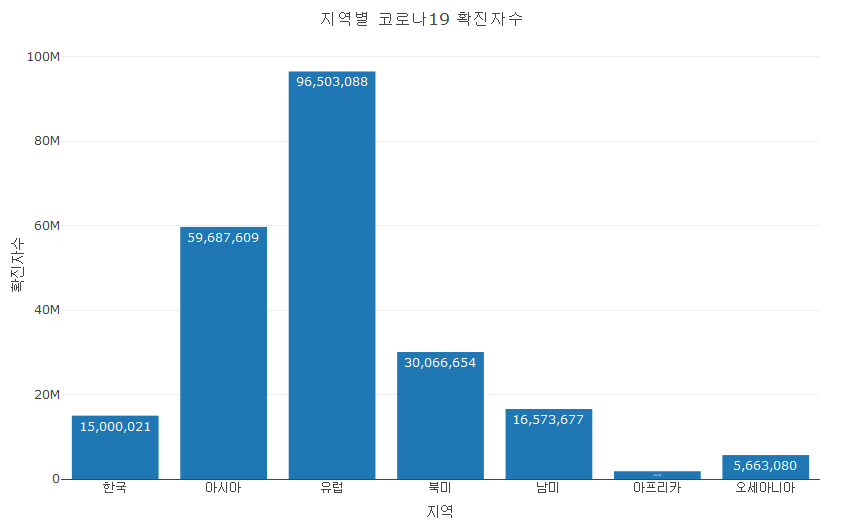
## 긴 형태의 100일간 코로나19 데이터 중에  
df\_covid19\_100 |>  
 ## 국가명으로 그룹화  
 group\_by(location) |>  
 ## 확진자수의 합계를 new\_cases로 산출  
 summarise(new\_cases = sum(new\_cases)) |>  
 ## X축을 location, Y축과 text를 new\_case로 매핑  
 plot\_ly(x = ~location, y = ~new\_cases, text = ~new\_cases,  
 ## textposition을 'inside'로 설정  
 textposition = 'inside') |>   
 layout(title = list(text = '지역별 코로나19 확진자수'),  
 xaxis = list(title = '지역'),  
 yaxis = list(title = '확진자수'),   
 margin = margins)  
  
df\_covid19\_100 |>  
 group\_by(location) |>  
 summarise(new\_cases = sum(new\_cases)) |>  
 plot\_ly(x = ~location, y = ~new\_cases, text = ~new\_cases,   
 ## textposition을 'outside'로 설정  
 textposition = 'outside') |>   
 layout(title = list(text = '지역별 코로나19 확진자수'),  
 xaxis = list(title = '지역'),  
 yaxis = list(title = '확진자수'),   
 margin = margins)  
  
df\_covid19\_100 |>  
 group\_by(location) |>  
 summarise(new\_cases = sum(new\_cases)) |>  
 plot\_ly(x = ~location, y = ~new\_cases, text = ~new\_cases,   
 ## textposition을 'auto'로 설정  
 textposition = 'auto') |>   
 layout(title = list(text = '지역별 코로나19 확진자수'),  
 xaxis = list(title = '지역'),  
 yaxis = list(title = '확진자수'),   
 margin = margins)  
  
df\_covid19\_100 |>  
 group\_by(location) |>  
 summarise(new\_cases = sum(new\_cases)) |>  
 plot\_ly(x = ~location, y = ~new\_cases, text = ~new\_cases,   
 ## textposition을 'none'으로 설정  
 textposition = 'none') |>   
 layout(title = list(text = '지역별 코로나19 확진자수'),  
 xaxis = list(title = '지역'),  
 yaxis = list(title = '확진자수'),   
 margin = margins)



실행결과 2- . textposition에 설정 결과

texttemplate은 텍스트가 표시되는 형태를 설정하는 type이다. texttemplate는 hovertemplate의 정의와 같이 사용되는데 ‘%{변수:변수포맷}’의 형태로 사용된다. 변수 포맷에서 사용하는 포맷은 자바 스크립트의 d3 format[[5]](#footnote-38)을 사용한다. 앞의 예에서 일반대학의 입학생수의 포맷에 천단위 콤마를 넣는다면’%{text:,}‘로 설정하고 소수점 아래 두째 자리까지 표기한다면’%{text:2f}’로 설정한다.

df\_covid19\_100 |>  
 group\_by(location) |>  
 summarise(new\_cases = sum(new\_cases)) |>  
 plot\_ly(x = ~location, y = ~new\_cases, text = ~new\_cases,   
 textposition = 'inside',  
 ## texttemplate를 설정  
 texttemplate = '%{text:,}'  
 ) |>   
 ## 제목, 축제목, 여백 설정  
 layout(title = list(text = '지역별 코로나19 확진자수'),  
 xaxis = list(title = '지역'),  
 yaxis = list(title = '확진자수'),   
 margin = margins)



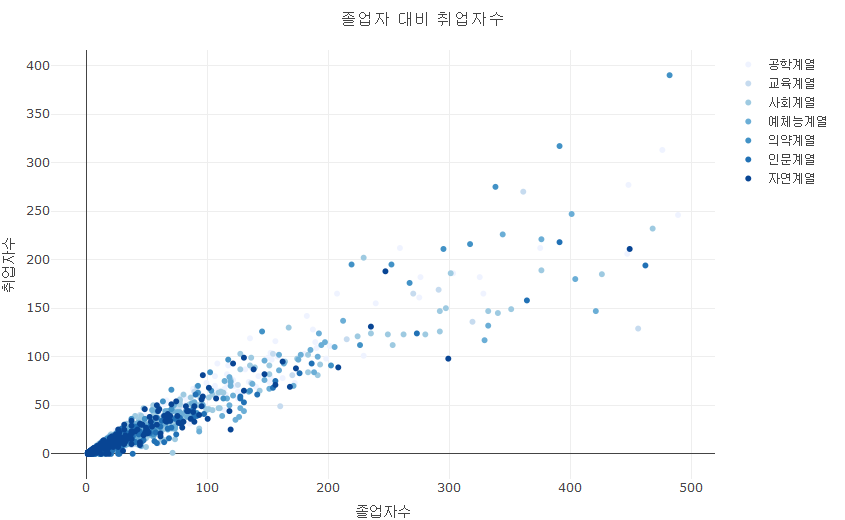
실행결과 2- . texttemplate 설정 결과

#### 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 속성이 hover\*이다.

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

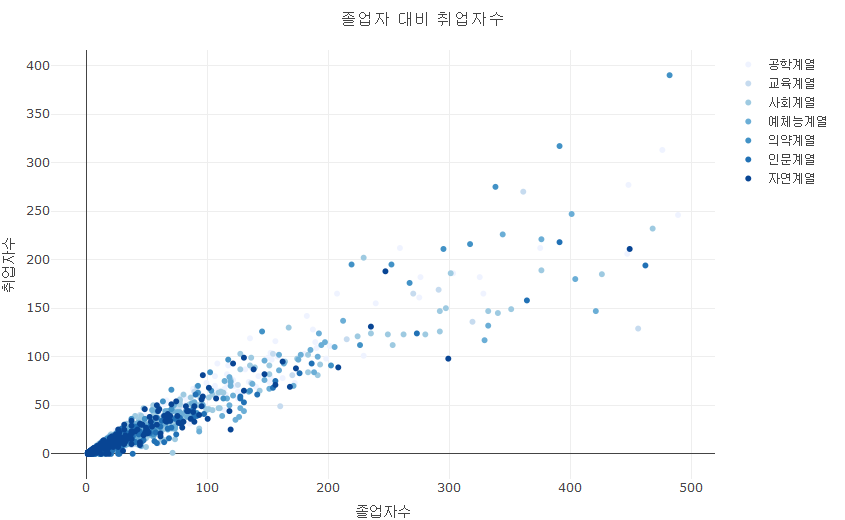
df\_취업률\_2000 |>  
 plot\_ly(x = ~졸업자수, y = ~취업자수, name = ~대계열,  
 ## hovertext를 학과명으로 매핑  
 hovertext = ~학과명) |>  
 ## 제목과 여백의 설정  
 layout(title = '졸업자 대비 취업자수', margin = margins)



실행결과 2- . hovertext 매핑 결과

hoverinfo는 호버에 표시되는 데이터 정보를 설정하는 속성이다. 호버에 표시되는 데이터 정보는 각각의 trace에 따라 달라지지만 각 trace에서 많이 사용되는 hoverinfo 속성은 X, Y, Z축의 좌표를 표기하는 것으로 x(X축 좌표), y(Y축 좌표), z(Z축 좌표) 속성을 사용한다. 이 외에도 , text(특정 문자열), name(trace name), none(제거), skip(생략)이 사용될 수 있고 각각은 +를 사용하여 조합할 수 있다.

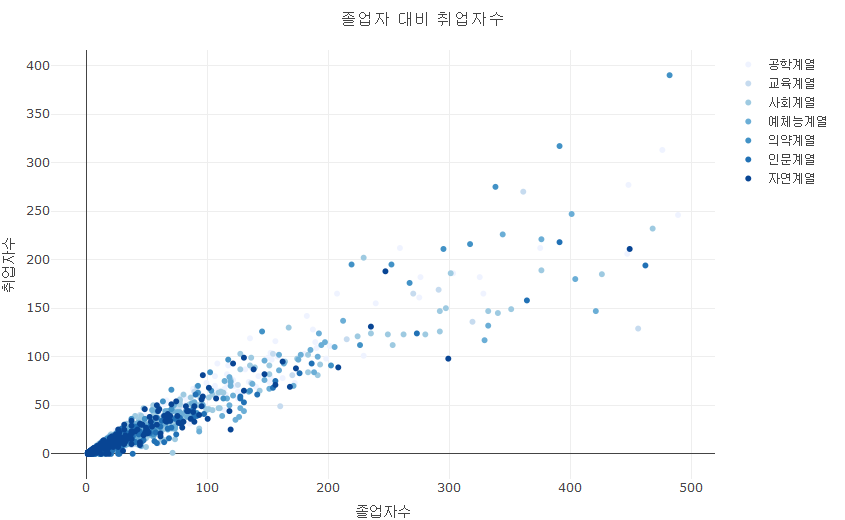
df\_취업률\_2000 |>  
 plot\_ly(x = ~졸업자수, y = ~취업자수, name = ~대계열,  
 ## hovertext, hoverinfo의 설정  
 hovertext = ~학과명, hoverinfo = 'x+y+text') |>  
 layout(title = '졸업자 대비 취업자수', margin = margins)



실행결과 2- . hoverinfo 설정 결과

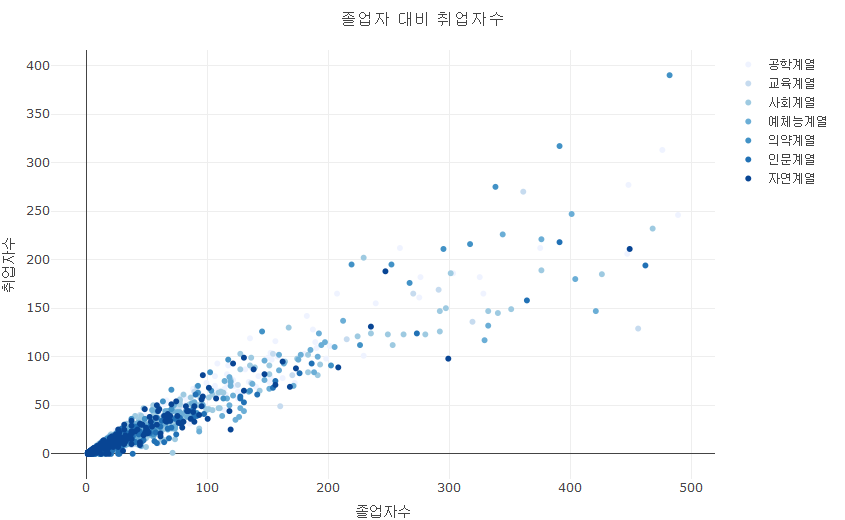
hovertext는 X, Y 좌표에 표시되는 문자열을 설정하는 속성이다. hovertext는 앞서 설명한 hoverinfo를 text로 설정하고 hovertext를 설정할 때 호버 상자 안에 text설정값이 표시되지만, hoverinfo를 생략하고 hovertext만 설정하면 호버 상자 밖에 name 속성 값(범례 값)이 표시된다는 차이가 있다. 또 hovertext에 단순 문자열을 설정하면 모든 데이터 호버에 동일한 문자열이 표기되지만 벡터를 설정하면 각각의 호버에 매핑된 결과가 표시된다.

df\_취업률\_2000 |>   
 plot\_ly(x = ~졸업자수, y = ~취업자수, name = ~대계열,  
 ## hoverinfo를 설정하고 hovertext를 중계열로 매핑  
 hoverinfo = 'x+y+text', hovertext = ~중계열) |>  
 layout(title = '졸업자 대비 취업자수', margin = margins)



실행결과 2- . hoverinfo와 hovertext 동시 매핑 결과

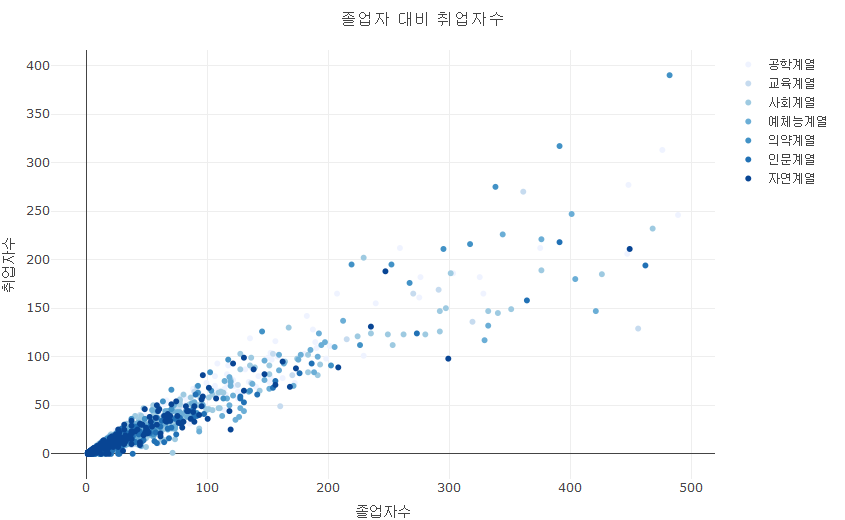
df\_취업률\_2000 |>   
 plot\_ly(x = ~졸업자수, y = ~취업자수, name = ~대계열,  
 ## hoverinfo를 설정없이 hovertext를 중계열로 매핑  
 hovertext = ~중계열) |>  
 layout(title = '졸업자 대비 취업자수', margin = margins)



실행결과 2- . hoverinfo 없이 hovertext 매핑 결과

hovertemplete는 호버 상자와 호버 상자에 표시되는 정보의 포맷을 설정하는 속성이다. 이 속성은 앞서 설명한 hoverinfo에 설정된 사용되는 속성의 변수를 %{변수}의 형태로 사용할 수 있다. 예를 들어 호버 상자에 Y축 값을 ‘Y값 : ’을 붙여 표시하기 위해서는 Y값 : %{y}로 설정한다. 앞서 사용된 예에서 X축의 값에 ’졸업자:’, Y축의 값에 ‘취업자:’, 대계열에 ’대계열:’를 표시하고 값을 표시하는 코드는 다음과 같다.

df\_취업률\_2000 |>   
 plot\_ly(x = ~졸업자수, y = ~취업자수, name = ~대계열,   
 hoverinfo = 'x+y+text', text = ~대계열,   
 ## hovertamplate의 설정  
 hovertemplate = ' 졸업자:%{x}, 취업자:%{y}, 대계열:%{text}') |>  
 layout(title = '졸업자 대비 취업자수', margin = margins)

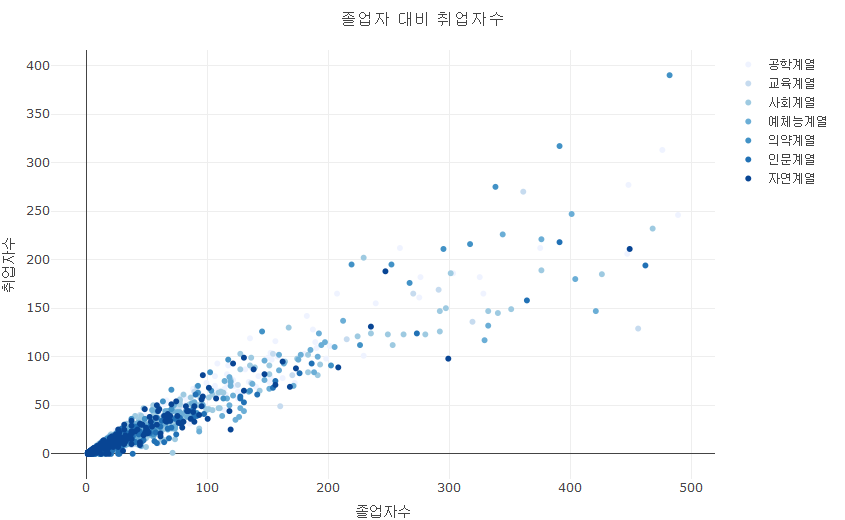


실행결과 2- . hovertemplate 설정 결과

#### 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\_취업률\_2000 |>   
 ## X, Y축의 매핑과 color를 대계열로 매핑  
 plot\_ly(x = ~졸업자수, y = ~취업자수, color = ~대계열) |>  
 layout(title = '졸업자 대비 취업자수', margin = margins)



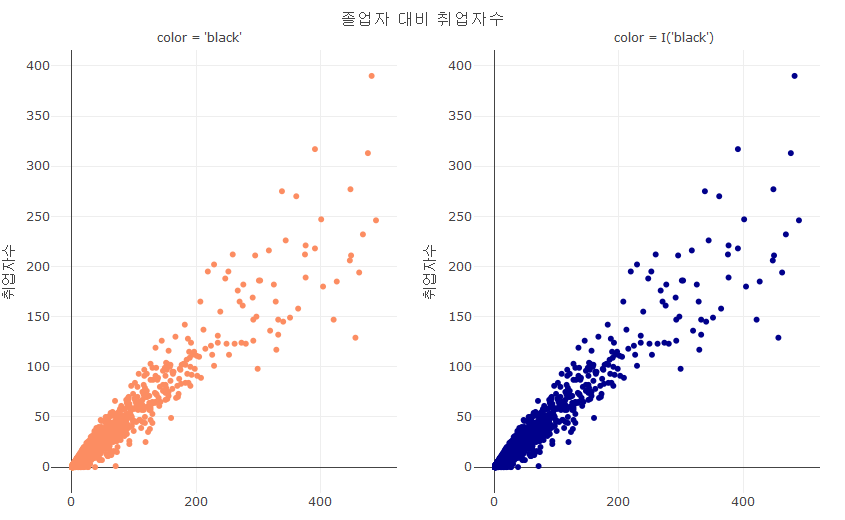
실행결과 2- . color 매핑 결과

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\_취업률\_2000 |>   
 ## plot\_ly()에 color를 색 이름 설정  
 plot\_ly(x = ~졸업자수, y = ~취업자수, color = 'darkblue') |>  
 layout(title = '졸업자 대비 취업자수', margin = margins)

plotly에서 색의 사용은 기본적으로 매핑을 전제로 사용된다. 따라서 앞의 코드에서 color = 'black'으로 설정하는 것은 색을 검정색으로 지정하는 것이 아니고 ‘black’이라는 이름으로 매핑된 색 배열을 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\_취업률\_2000 |>   
 ## plot\_ly()에 color를 I()사용하여 색 이름 설정  
 plot\_ly(x = ~졸업자수, y = ~취업자수, color = I('darkblue')) |>  
 layout(title = '졸업자 대비 취업자수', margin = margins)



실행결과 2- . I()를 사용한 color 설정 결과

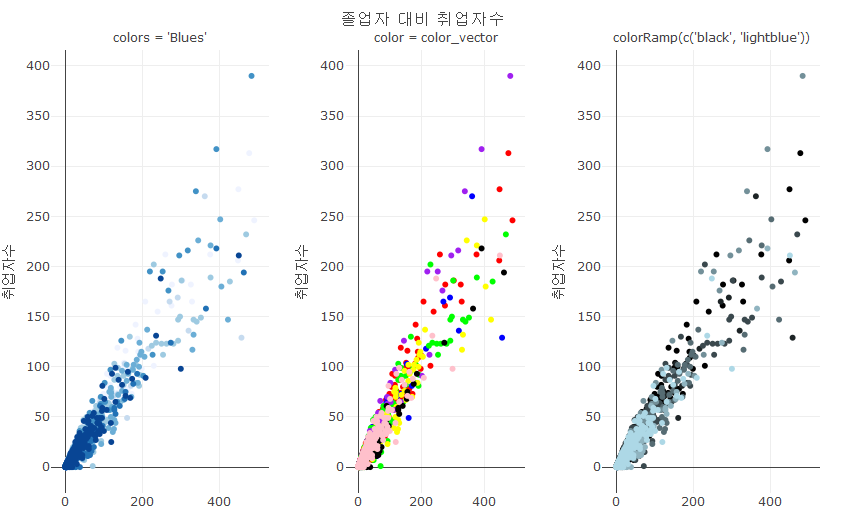
colors는 color에서 매핑된 변수에 따른 색의 스케일을 설정하는 속성이다. plotly에서 색의 스케일을 설정할 때 다음의 세 가지 방법 중 하나를 사용한다.

첫 번째 방법은 RColorBrewer 패키지에서 제공하는 팔레트의 이름을 설정하는 방법이다. RColorBrewer 패키지는 R에서 색 팔레트를 제공하는 패키지 중 가장 대중적으로 사용되는 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\_취업률\_2000 |>   
 plot\_ly(x = ~졸업자수, y = ~취업자수,  
 ## color를 대계열로 매핑하고 colors를 'Accent' 팔레트로 설정  
 color = ~대계열, colors = 'Blues') |>  
 layout(title = '졸업자 대비 취업자수', margin = margins)  
  
color\_vector <- c('red', 'blue', 'green', 'yellow', 'purple', 'black', 'pink')  
  
df\_취업률\_2000 |>   
 plot\_ly(x = ~졸업자수, y = ~취업자수,   
 ## color를 대계열로 매핑하고 colors를 색 이름 벡터로 설정  
 color = ~대계열, colors = color\_vector) |>  
 layout(title = '졸업자 대비 취업자수', margin = margins)  
  
df\_취업률\_2000 |>   
 plot\_ly(x = ~졸업자수, y = ~취업자수,   
 ## color를 대계열로 매핑하고 colors를 colorRamp()로 설정  
 color = ~대계열, colors = colorRamp(c('red', 'yellow', 'blue'))) |>  
 layout(title = '졸업자 대비 취업자수', margin = margins)

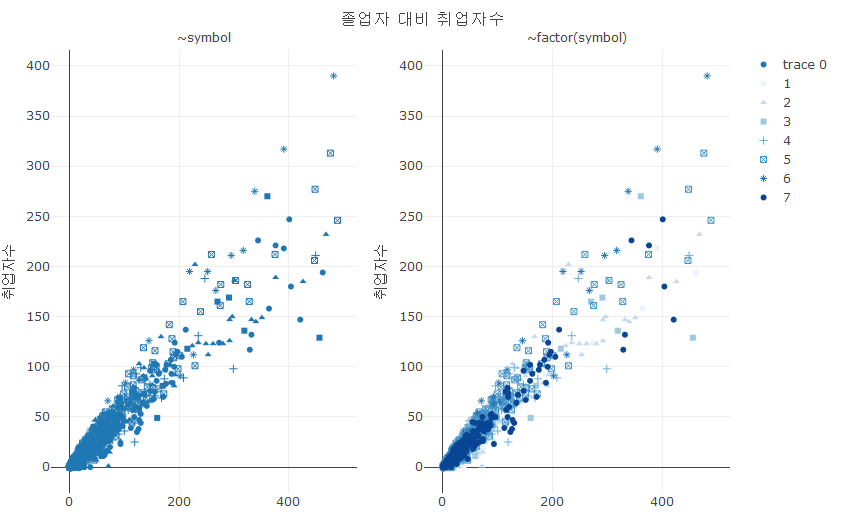


실행결과 2- . 색 팔레트 설정 결과

#### symbol, symbols

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

df\_취업률\_symbol <- df\_취업률\_2000 |>   
 ## 심볼 설정을 위한 열 생성  
 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)  
 )   
  
## 심볼 설정을 위한 열이 추가된 데이터프레임에서  
df\_취업률\_symbol |>   
 plot\_ly(x = ~졸업자수, y = ~취업자수,   
 ## symbol을 연속형 수치 변수에 매핑  
 symbol = ~symbol) |>  
 layout(title = '졸업자 대비 취업자수', margin = margins)  
  
df\_취업률\_symbol |>   
 plot\_ly(x = ~졸업자수, y = ~취업자수,   
 ## symbol을 팩터형 변수에 매핑  
 symbol = ~factor(symbol)) |>  
 layout(title = '졸업자 대비 취업자수', margin = margins)

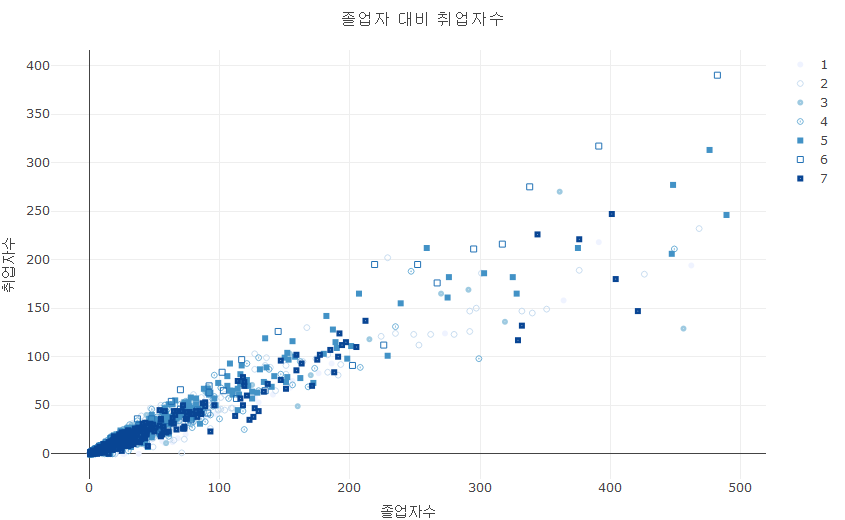


실행결과 2- . symbol 연속형, 팩터형 변수 매핑 결과

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 trace(trace 0)로 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.

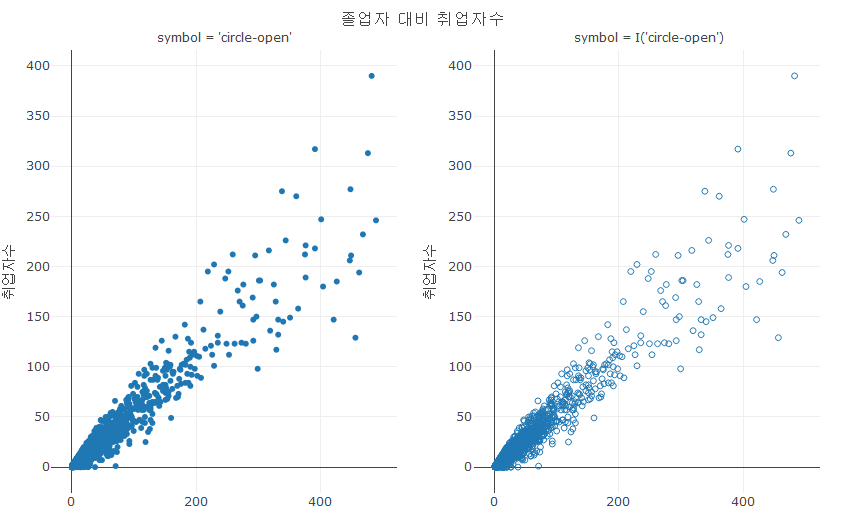
df\_취업률\_symbol |>   
 plot\_ly(x = ~졸업자수, y = ~취업자수,   
 symbol = ~factor(symbol),   
 ## symbols를 심볼 이름으로 설정  
 symbols = c('circle', 'circle-open', 'circle-dot', "circle-open-dot",   
 "square", "square-open", "square-dot")) |>  
 layout(title = '졸업자 대비 취업자수', margin = margins)



실행결과 2- . symbols 설정 결과

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

df\_취업률\_symbol |>  
 ## symbol을 심볼 이름으로 설정  
 plot\_ly(x = ~졸업자수, y = ~취업자수, symbol = 'circle-open') |>  
 layout(title = '졸업자 대비 취업자수', margin = margins)  
  
df\_취업률\_symbol |>   
 ## symbol을 I()를 사용하여 심볼 이름으로 설정  
 plot\_ly(x = ~졸업자수, y = ~취업자수, symbol = I('circle-open')) |>  
 layout(title = '졸업자 대비 취업자수', margin = margins)

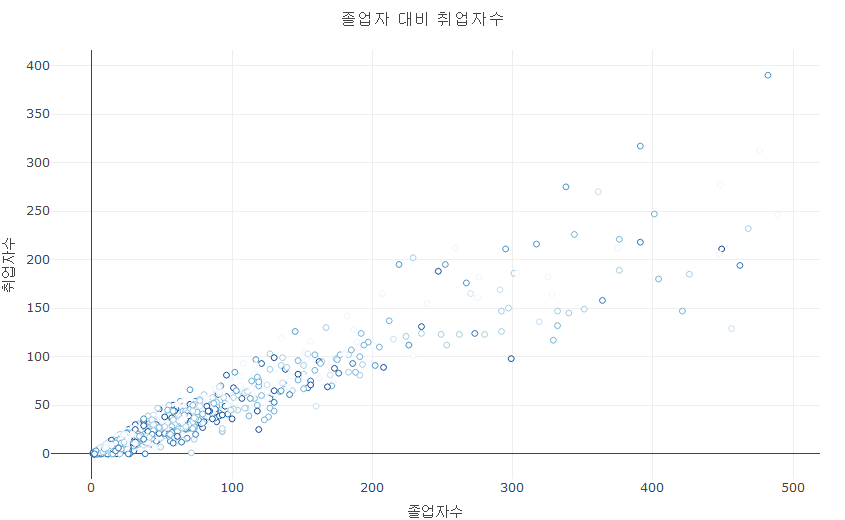


실행결과 2- . I()를 사용한 symbol 설정 결과

#### 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.

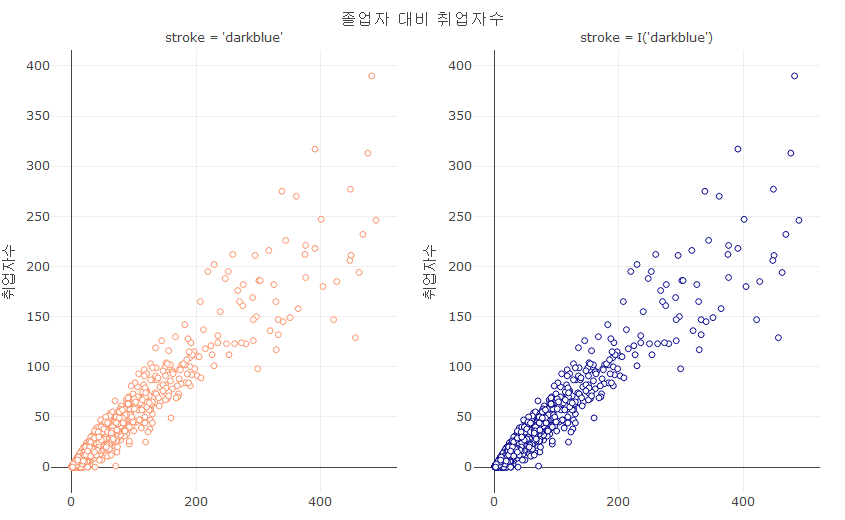
df\_취업률\_2000 |>   
 plot\_ly(x = ~졸업자수, y = ~취업자수, color = I('white'),  
 ## strock를 대계열로 매핑, strokes를 Accent 팔레트로 설정  
 stroke = ~대계열, strokes = 'Blues') |>  
 layout(title = '졸업자 대비 취업자수', margin = margins)



실행결과 2- . stroke 매핑과 strokes의 설정 결과

stroke를 특정 값으로 설정하기 위해서는 I()를 사용한다.

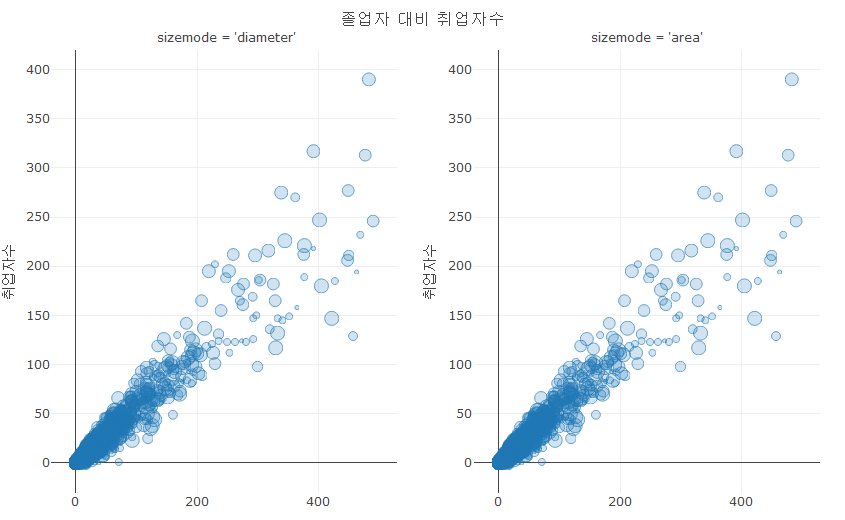
df\_취업률\_2000 |>   
 plot\_ly(x = ~졸업자수, y = ~취업자수, color = I('white'),  
 ## stroke에 색 이름 설정  
 stroke = 'darkblue') |>  
 layout(title = '졸업자 대비 취업자수', margin = margins)  
  
df\_취업률\_2000 |>   
 plot\_ly(x = ~졸업자수, y = ~취업자수, color = I('white'),   
 ## stroke에 I()를 사용한 색 이름 설정  
 stroke = I('darkblue')) |>  
 layout(title = '졸업자 대비 취업자수', margin = margins)



#### size, sizes

size는 점의 크기를 설정하는 속성이다. size는 변수 매핑을 위한 속성이고 sizes는 변수에 매핑된 카테고리 별로 크기를 설정하는 속성이다. 앞선 속성들과 달리 size는 sizemode 속성을 추가적으로 사용할 수 있다. sizemode는 점의 크기를 결정하는 기준을 설정하는 속성으로 점의 크기를 지름으로 설정하는 ’diameter’와 면적으로 설정하는 ’area’의 두 가지가 있다.

df\_취업률\_symbol |>   
 plot\_ly(x = ~졸업자수, y = ~취업자수, alpha = 0.3,   
 size = ~symbol, sizemode = 'diameter') |>  
 layout(title = '졸업자 대비 취업자수', margin = margins)  
  
df\_취업률\_symbol |>   
 plot\_ly(x = ~졸업자수, y = ~취업자수, alpha = 0.3,   
 size = ~symbol, sizemode = 'area') |>  
 layout(title = '졸업자 대비 취업자수', margin = margins)

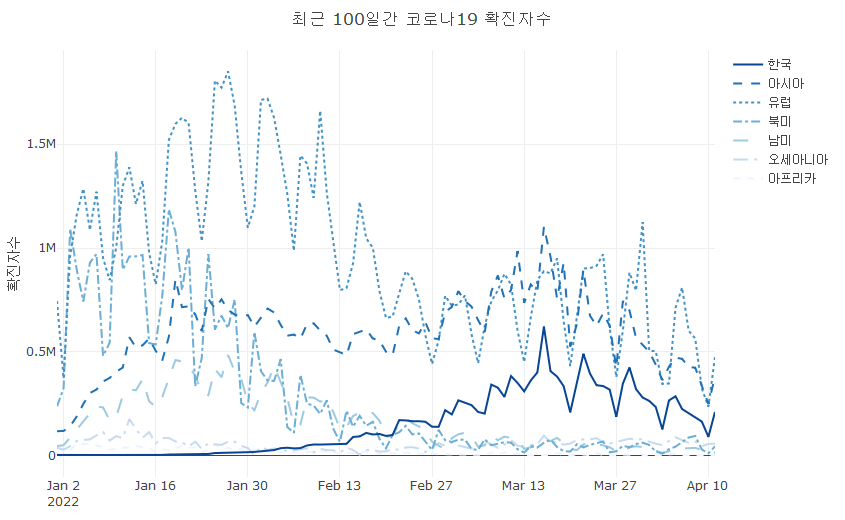


실행결과 2- . size 매핑과 sizemode의 설정 결과

#### linetype, linetypes

linetype은 선의 형태를 설정하는 속성이다. plotly에서 선의 형태를 설정할 때 변수 매핑 위해서는 linetype을 사용하고 linetype에 매핑된 변수에 따라 선의 형태를 설정하기 위해서는 linetypes를 사용한다.

df\_covid19\_100 |>  
 plot\_ly(type = 'scatter', mode = 'lines', x = ~date, y = ~new\_cases,  
 ## linetype을 location에 매핑  
 linetype = ~location) |>  
 layout(title = '최근 100일간 코로나19 확진자수',   
 xaxis = list(title = list(text = NULL)),   
 yaxis = list(title = list(text = '확진자수')),   
 margin = margins)

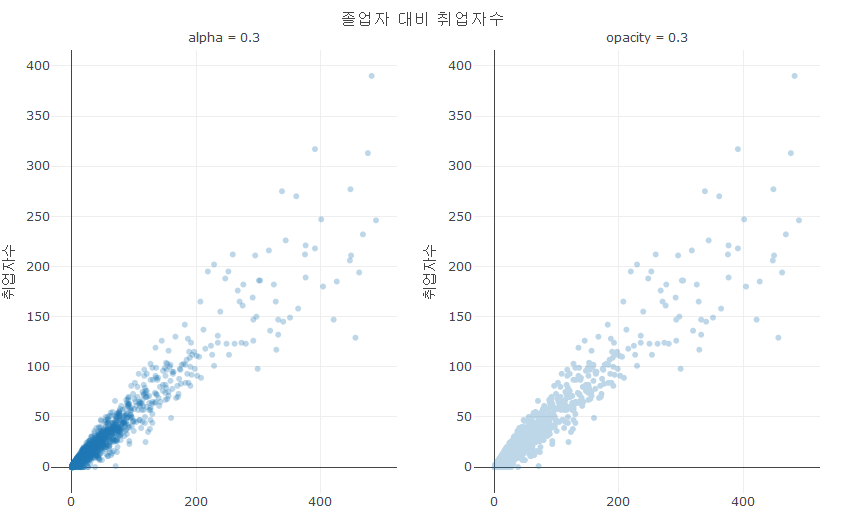


실행결과 2- . linetype의 매핑 결과

#### 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 않고 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\_취업률\_2000 |>   
 ## alpha를 0.3으로 설정  
 plot\_ly(x = ~졸업자수, y = ~취업자수, alpha = 0.3) |>  
 layout(title = '졸업자 대비 취업자수', margin = margins)  
  
df\_취업률\_2000 |>   
 ## opacity를 0.3으로 설정  
 plot\_ly(x = ~졸업자수, y = ~취업자수, opacity = 0.3) |>  
 layout(title = '졸업자 대비 취업자수', margin = margins)

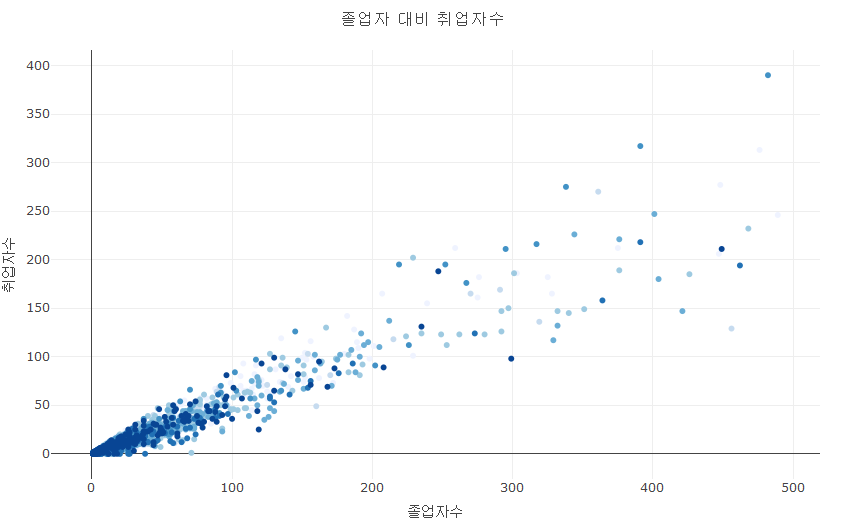


실행결과 2- . alpha와 opacity의 설정 결과

#### showlegend

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

df\_취업률\_2000 |>   
 plot\_ly(x = ~졸업자수, y = ~취업자수, name = ~대계열,  
 ## showlegend을 FALSE로 설정  
 showlegend = FALSE) |>  
 layout(title = '졸업자 대비 취업자수', margin = margins)



실행결과 2- . showlegend 설정 결과

### trace 추가 : add\_trace()

trace는 plotly 객체에서 표현되는 단일 데이터의 그래픽적 표현을 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.

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 만들어서 매핑할 변수를 만들어 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 막대 그래프, 박스 플롯 등 모든 plotly 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 속성을 가지고 있다. 앞서 plot\_ly()에서 속성으로 설정했던 많은 속성들은 모두 개별 trace의 설정으로도 사용이 가능하다. 하지만 plot\_ly()에서 trace의 type을 설정하지 않았는데 시각화의 결과는 데이터들의 trace가 나타났다. 이는 plotly에서 데이터를 분석하여 가장 효과적인 trace를 매칭해주기 때문이다. 하지만 trace는 사용자가 지정하는 것이 일반적이다.

이렇게 trace의 type을 설정하는 방법은 add\_trace()의 속성인 type 속성을 설정하는 방법과 add\_markers(), add\_lines()등과 같이 add\_trace()에서 파생된 래핑 함수를 사용하는 방법이 있다.

add\_trace()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

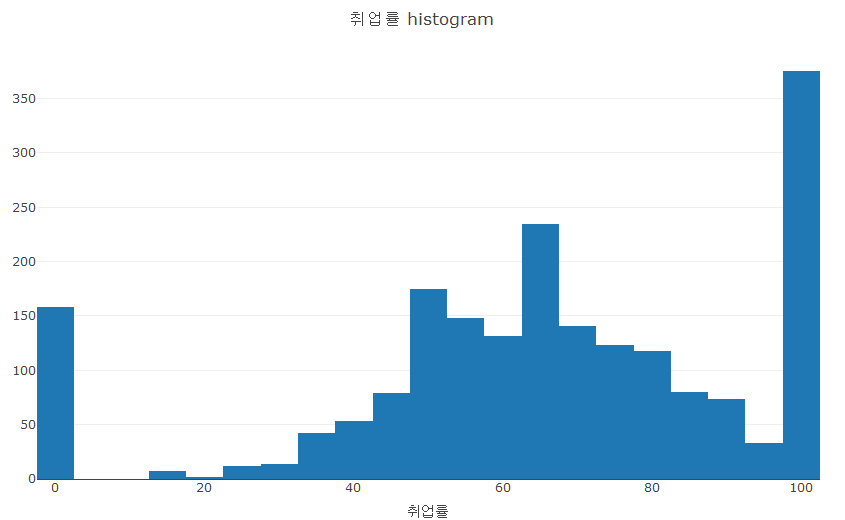
trace의 종류에 따른 각각의 특성과 속성은 다음과 같다.

#### 히스토그램(histogram) trace

plotly에서 히스토그램 시각화를 위한 trace의 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

## 취업률 데이터를 사용해 plotly 객체 생성  
p\_histogram <- df\_취업률\_2000 |> plot\_ly()  
  
p\_histogram |>   
 ## histogram trace로 X축을 취업률로 매핑, name을 취업률로 설정  
 add\_trace(type = 'histogram', x = ~취업률, name = '취업률') |>  
 ## 제목과 여백 설정  
 layout(title = '취업률 histogram', margin = margins)



실행결과 2- . 히스토그램 trace 시각화

plotly의 히스토그램 trace에서 사용되는 주요 속성들은 다음과 같다.

| 속성 | 설명 | 속성값 | 세부 속성 |
| --- | --- | --- | --- |
| type | trace type 설정 | ‘histogram’ |  |
| visible | trace의 표시 설정 | TRUE, FALSE, ‘legendonly’ |  |
| textposition | 히스토그램 텍스트의 위치 설정 | ‘inside’, ‘outside’, ‘auto’, ‘none’ |  |
| histfunc | 히스토그램 값 설정 | ‘count’, ‘sum’, ‘avg’, ‘min’, ‘max’ |  |
| histnorm | 히스토그램 정규화 설정 | ’‘, ’percent’, ‘probability’, ‘density’, ‘probability density’ |  |
| nbinsx, nbinsy | 히스토그램 bin의 최대값 설정 | 0 이상의 정수 |  |
| xbins, ybins | X, Y축의 bin 개수 설정 | 세부 속성들의 리스트 | end, size, start |
| marker | 히스토그램을 표현하는 막대 설정 | 세부 속성들의 리스트 | color, colorbar, colorscale, line(color, width, …), opacity, pattern(bgcolor, fgcolor, fillmode, shape, …), … |
| textfont | 히스토그램에 표시되는 문자열 설정 | 세부 속성들의 리스트 | color, family, size |
| cumulative | 누적형 히스토그램 설정 | 세부 속성들의 리스트 | currentbin, direction, enabled |

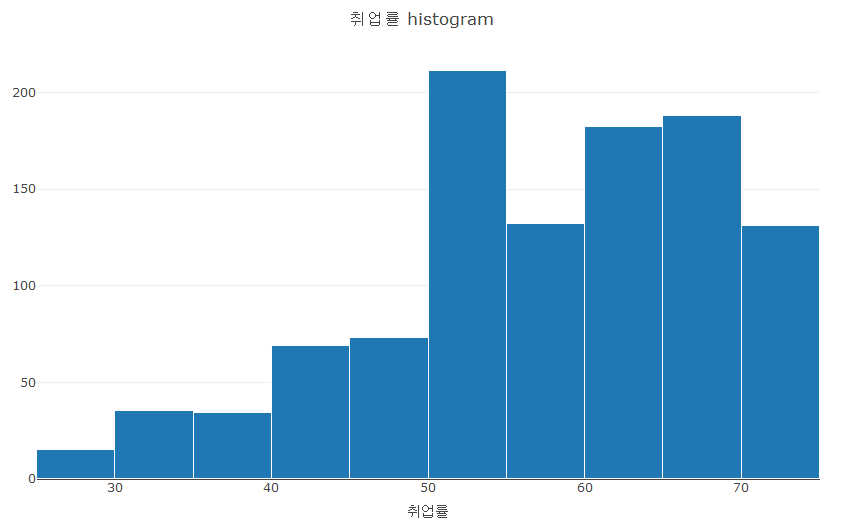
##### xbins

xbins는 X축에 매핑된 변수가 연속형 수치 변수일 경우 빈도를 표현할 구간의 범위를 설정하는 속성이다. 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,  
 ## xbins의 start, end, size 설정  
 xbins = list(start = 25, end = 75, size = 5),  
 ## stroke를 흰색으로 설정  
 stroke = I('white')) |>  
 layout(title = '취업률 histogram', margin = margins)

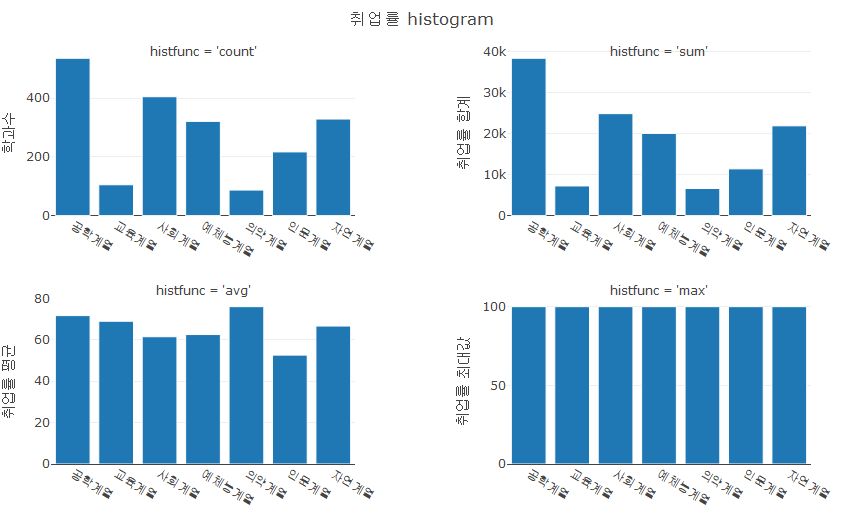


실행결과 2- . xbins를 설정한 히스토그램

##### histfunc

histfunc은 히스토그램의 막대로 표현하는 값의 종류를 설정하는 속성이다. 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,   
 ## stroke를 흰색으로, 히그토그램 막대 값을 'count'로 설정  
 stroke = I('white'), histfunc = 'count') |>  
 layout(title = '취업률 histogram', margin = margins,   
 yaxis = list(title = list(text = '학과수')))  
  
p\_histogram |>   
 add\_trace(type = 'histogram', x = ~대계열, y = ~as.character(취업률),   
 ## stroke를 흰색으로, 히그토그램 막대 값을 'sum'으로 설정  
 histfunc = 'sum', stroke = I('white')) |>  
 ## Y축을 선형으로 설정  
 layout(yaxis=list(type='linear',   
 title = list(text = '취업률 합계')),   
 title = '취업률 histogram', margin = margins)  
  
p\_histogram |>   
 add\_trace(type = 'histogram', x = ~대계열, y = ~as.character(취업률),   
 ## stroke를 흰색으로, 히그토그램 막대 값을 'average'로 설정  
 histfunc = 'avg', stroke = I('white')) |>  
 ## Y축을 선형으로 설정  
 layout(yaxis=list(type='linear',   
 title = list(text = '취업률 평균')),   
 title = '취업률 histogram', margin = margins)  
  
p\_histogram |>   
 add\_trace(type = 'histogram', x = ~대계열, y = ~as.character(취업률),  
 ## stroke를 흰색으로, 히그토그램 막대 값을 'max'로 설정  
 histfunc = 'max', stroke = I('white')) |>  
 ## Y축을 선형으로 설정  
 layout(yaxis=list(type='linear', title = list(text = '취업률 최대값')),   
 title = '취업률 histogram', margin = margins)

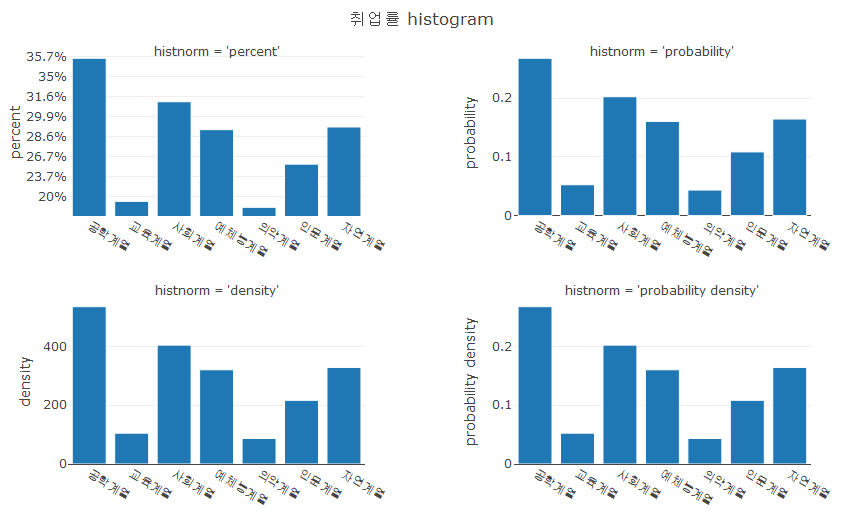


실행결과 2- . histfunc에 따른 히스토그램 결과

##### histnorm

histnorm은 히스토그램 trace로 표시되는 데이터의 정규화(normalization)에 대한 속성이다. 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),   
 ## histnorm을 'percent'로 설정  
 stroke = I('white'), histnorm = 'percent') |>  
 layout(title = '취업률 histogram', margin = margins,  
 yaxis = list(title = list(text = 'percent'), ticksuffix = "%"))  
  
p\_histogram |>   
 add\_trace(type = 'histogram', x = ~대계열, y = ~as.character(취업률),   
 ## histnorm을 'probability'로 설정  
 histnorm = 'probability', stroke = I('white')) |>  
 layout(yaxis=list(type='linear', title = list(text = 'probability'))) |>  
 layout(title = '취업률 histogram', margin = margins)  
  
p\_histogram |>   
 add\_trace(type = 'histogram', x = ~대계열, y = ~as.character(취업률),   
 ## histnorm을 'density'로 설정  
 histnorm = 'density', stroke = I('white')) |>  
 layout(yaxis=list(type='linear', title = list(text = 'density'))) |>  
 layout(title = '취업률 histogram', margin = margins)  
  
p\_histogram |>   
 add\_trace(type = 'histogram', x = ~대계열, y = ~as.character(취업률),   
 ## histnorm을 'probability density'로 설정  
 histnorm = 'probability density', stroke = I('white')) |>  
 layout(yaxis=list(type='linear',   
 title = list(text = 'probability density'))) |>  
 layout(title = '취업률 histogram', margin = margins)



실행결과 2- . histnorm에 따른 히스토그램 결과

#### 스캐터(scatter) trace : 산점도, 선 그래프

스캐터 trace는 점, 선, 문자를 X, Y 축 좌표의 위치를 사용해 시각화를 하는 형태의 trace를 모두 통칭한다. 따라서 스캐터 trace를 통해 생성 가능한 시각화는 산점도, 선 그래프, 텍스트 차트, 풍선 차트 등이다. 이 스캐터 trace는 add\_trace()의 type 속성에 ’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 속성을 사용하여 설정이 가능하다. 이 50여개의 속성 중에는 다시 세부 속성을 설정하는 속성들이 있고 총 4단계까지 내려가는 속성도 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.[[6]](#footnote-53) 스캐터 trace에서 사용하는 속성 중 주요 속성은 다음과 같다.

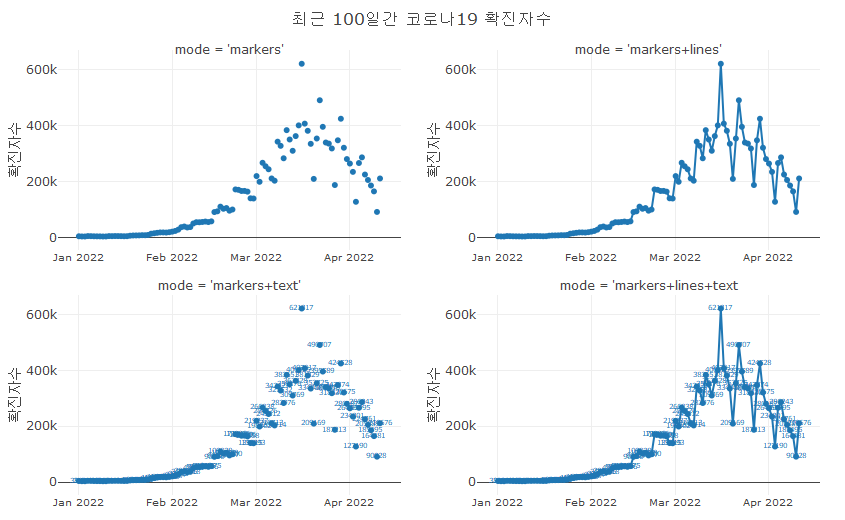
| 속성 | 설명 | 속성값 | 세부 속성 |
| --- | --- | --- | --- |
| type | trace type 설정 | ‘scatter’ |  |
| visible | trace의 표시 설정 | TRUE, FALSE, ‘legendonly’ |  |
| mode | 스캐터 trace의 모드 설정 | ‘lines’, ‘markers’, ‘text’의 조합(’+’) |  |
| x0, y0 | x, y의 대체로 X, Y축의 시작점(x0, y0) | 수치나 X축에 매핑될 이산형 카테고리 변수 |  |
| dx, dy | x, y의 대체로 X, Y축의 변화량(dx, dy) | 수치 |  |
| xperiod, yperiod | 축이 ‘date’ 타입인 경우 기간 위치를 설정 | 수치나 X축에 매핑될 이산형 카테고리 변수 |  |
| marker | mode가 점(markers)인 경우 점의 세부 설정 | 세부 속성들의 리스트 | color, colorbar, colorscale, line(color, width, …), opacity, pattern(bgcolor, fgcolor, fillmode, shape, …), gradient(color, type), symbol, size, sizemode, … |
| line | mode가 선(lines)인 경우 선의 세부 설정 | 세부 속성들의 리스트 | color, dash, shape, simplify, smoothing, width |
| textfont | mode가 텍스트(text)인 경우 문자의 세부 설정 | 세부 속성들의 리스트 | color, family, size |
| connectgaps | 결측치간을 이어줄 것인지 결정 | 논리값 |  |
| fillcolor | 점, 선등의 내부색 설정 | 색 이름, 색 값, 색 벡터 중 하나 |  |

##### 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 |

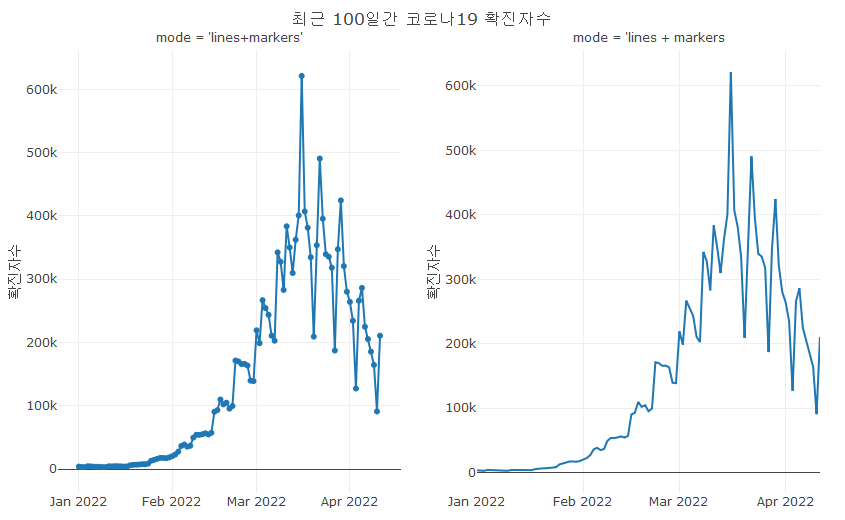
## 넓은 형태의 100일간 코로나19 데이터로 plotly 객체 생성  
p\_line\_wide <- df\_covid19\_100\_wide |> plot\_ly()  
  
p\_line\_wide |>   
 ## type을 scatter, mode를 markers로 설정  
 add\_trace(type = 'scatter', mode = 'markers',   
 ## X축은 data, Y축은 확진자\_한국으로 매핑  
 x = ~date, y = ~확진자\_한국) |>  
 ## 제목, Y축 제목, 여백 설정  
 layout(title = '최근 100일간 코로나19 확진자수',   
 yaxis = list(title = list(text = '확진자수')),   
 margin = margins)  
  
p\_line\_wide |>   
 ## type을 scatter, mode를 markers+lines로 설정  
 add\_trace(type = 'scatter', mode = 'markers+lines',   
 x = ~date, y = ~확진자\_한국) |>  
 layout(title = '최근 100일간 코로나19 확진자수',   
 yaxis = list(title = list(text = '확진자수')),   
 margin = margins)  
  
p\_line\_wide |>   
 ## type을 scatter, mode를 markers+text로 설정  
 add\_trace(type = 'scatter', mode = 'markers+text',   
 x = ~date, y = ~확진자\_한국, text = ~확진자\_한국) |>  
 layout(title = '최근 100일간 코로나19 확진자수',   
 yaxis = list(title = list(text = '확진자수')),   
 margin = margins)  
  
p\_line\_wide |>   
 ## type을 scatter, mode를 markers+lines+text로 설정  
 add\_trace(type = 'scatter', mode = 'markers+lines+text',  
 x = ~date, y = ~확진자\_한국, text = ~확진자\_한국) |>  
 layout(title = '최근 100일간 코로나19 확진자수',   
 yaxis = list(title = list(text = '확진자수')),   
 margin = margins)



실행결과 2- . 스캐터 trace의 mode 설정 결과

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.

## 넓은 형태의 데이터로 초기화한 plotly 객체로  
p\_line\_wide |>   
 ## type을 scatter, mode를 markers+lines로 설정  
 add\_trace(type = 'scatter', mode = 'markers+lines',   
 ## X축은 data, Y축은 확진자\_한국으로 매핑  
 x = ~date, y = ~확진자\_한국) |>  
 ## 제목, X, Y축 제목, 여백 설정  
 layout(title = '최근 100일간 코로나19 확진자수',   
 xaxis = list(title = list(text = NULL)),   
 yaxis = list(title = list(text = '확진자수')),   
 margin = margins)  
  
p\_line\_wide |>   
 ## type을 scatter, mode를 markers + lines로 설정(공백포함)  
 add\_trace(type = 'scatter', mode = 'markers + lines',   
 x = ~date, y = ~확진자\_한국) |>  
 layout(title = '최근 100일간 코로나19 확진자수',   
 xaxis = list(title = list(text = NULL)),   
 yaxis = list(title = list(text = '확진자수')),   
 margin = margins)



실행결과 2- . 공백이 포함된 mode 설정 결과

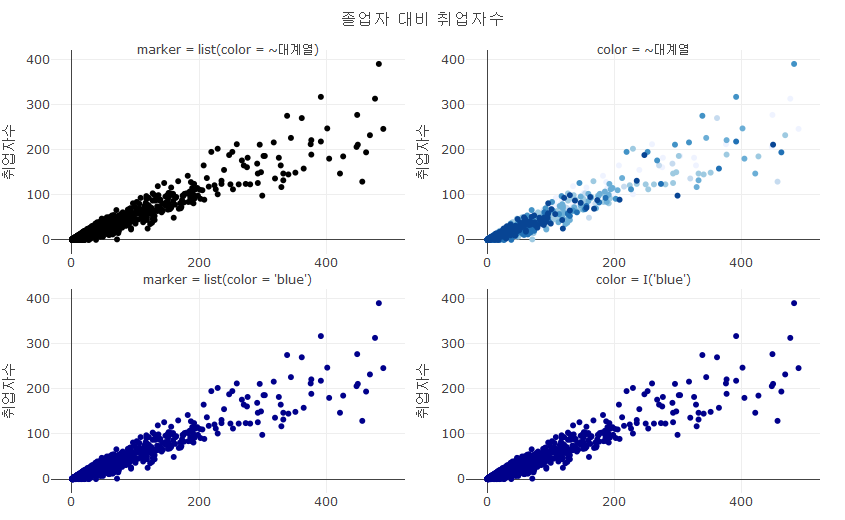
##### marker, add\_markers()

스캐터 trace의 marker 타입은 스캐터 trace에서 데이터를 표현하는 점의 속성을 설정하는 속성이다. 이 속성은 add\_trace(type = 'scatter', mode = 'marker')를 사용하거나 add\_markers()를 사용한다.

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

add\_trace()의 marker는 list()를 사용하여 세부 설정 속성 리스트를 만들어서 설정한다. 주의할 것은 marker의 리스트에 설정하는 속성들은 값을 설정할 수는 있지만 ~를 사용해 변수를 매핑할 수 없다는 점이다. marker 리스트에 ~를 사용한 변수 매핑을 사용한다고 해도 에러를 내지는 않지만 원하는 결과를 얻을 수는 없다. 또 하나 주의해아 하는 점은 다른 plot\_ly()의 속성 설정시 사용했던 I()의 사용이다. 앞서 plot\_ly()의 색 속성과 같은 일부 속성의 설정시에 I()를 사용한다고 설명했는데 marker로 설정하는 리스트안에서는 I()를 사용하지 않아도 색의 설정이 가능하다. marker 리스트는 변수의 매핑에 사용하는 것이 아니고 값을 설정에 사용하는 것이기 때문에 구지 I()를 사용하여 매핑과 설정을 구분해 줄 필요가 없다는 것이다.

## 취업률 데이터를 사용해 plotly 객체 생성  
p\_marker <- df\_취업률\_2000 |>   
 plot\_ly()  
  
p\_marker |>  
 ## type을 scatter, mode를 markers로 설정  
 add\_trace(type = 'scatter', mode = 'markers',   
 ## X축은 졸업자수, Y축은 취업자수로 매핑  
 x = ~졸업자수, y = ~취업자수,  
 ## marker의 color를 대계열로 매핑하지만 설정되지 않음  
 marker = list(color = ~대계열)) |>  
 layout(title = '졸업자 대비 취업자수', margin = margins)  
  
p\_marker |>  
 add\_trace(type = 'scatter', mode = 'markers',   
 x = ~졸업자수, y = ~취업자수,   
 ## trace의 color를 대계열로 매핑  
 color = ~대계열) |>  
 layout(title = '졸업자 대비 취업자수', margin = margins)  
  
p\_marker |>  
 add\_trace(type = 'scatter', mode = 'markers',   
 x = ~졸업자수, y = ~취업자수,  
 ## I()를 사용하는 설정과 동일한 결과  
 ## marker의 color를 'darkblue'로 설정  
 marker = list(color = 'darkblue')) |>  
 layout(title = '졸업자 대비 취업자수', margin = margins)  
  
p\_marker |>  
 add\_trace(type = 'scatter', mode = 'markers',   
 x = ~졸업자수, y = ~취업자수,  
 ## I()를 사용하지 않는 설정과 동일한 결과  
 ## marker의 color를 I()를 사용하여 'darkblue'로 설정  
 color = I('darkblue')) |>  
 layout(title = '졸업자 대비 취업자수', margin = margins)



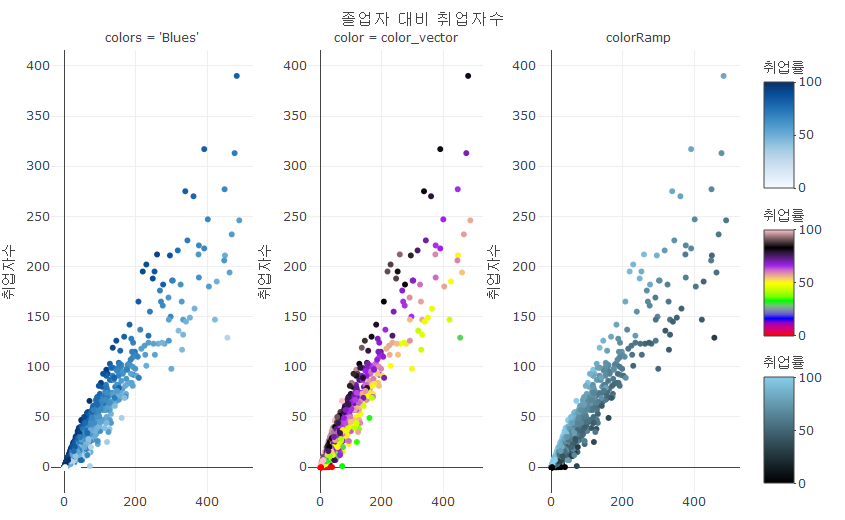
실행결과 2- . 스캐터 trace의 여러가지 color 설정 결과

marker 에서 설정 가능한 세부 속성은 20여개가 있지만 이 세부 속성의 세부 속성까지 포함하면 그 종류가 매우 많아진다. add\_markers()를 사용한다면 세부 속성을 바로 설정할 수 있지만 add\_trace()를 사용하면 marker 매개변수에 세부 속성을 list()를 사용하여 리스트로 만들어 사용할 수 있다.

marker의 설정에서 하나 중요한 속성은 colorbar이다. color 속성에 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 colorbar이다.

컬러바 설정에 사용되는 세부 속성은 현재(22.03) 43개가 제공된다. 이에 대한 세부 속성과 설명은 plotly 매뉴얼을 참조하라.[[7]](#footnote-56)

df\_취업률\_2000 |>   
 plot\_ly(x = ~졸업자수, y = ~취업자수,  
 ## color를 연속형 수치 변수에 매핑  
 color = ~취업률,   
 ## colors를 Rcolorbrewer의 'Accent' 팔레트로 설정  
 colors = 'viridis') |>  
 layout(title = '졸업자 대비 취업자수', margin = margins)  
  
## 컬러바 설정을 위한 벡터 설정  
color\_vector <- c('red', 'blue', 'green', 'yellow', 'purple', 'black', 'pink')  
  
df\_취업률\_2000 |>   
 plot\_ly(x = ~졸업자수, y = ~취업자수,   
 color = ~취업률,   
 ## colors를 컬러 벡터로 설정  
 colors = color\_vector) |>  
 layout(title = '졸업자 대비 취업자수', margin = margins)  
  
df\_취업률\_2000 |>   
 plot\_ly(x = ~졸업자수, y = ~취업자수,   
 color = ~취업률,   
 ## colors를 colorRamp()를 사용하여 설정  
 colors = colorRamp(c('red', 'yellow', 'blue'))) |>  
 layout(title = '졸업자 대비 취업자수', margin = margins)



실행결과 2- . 스캐터 trace의 colorbar의 설정 결과

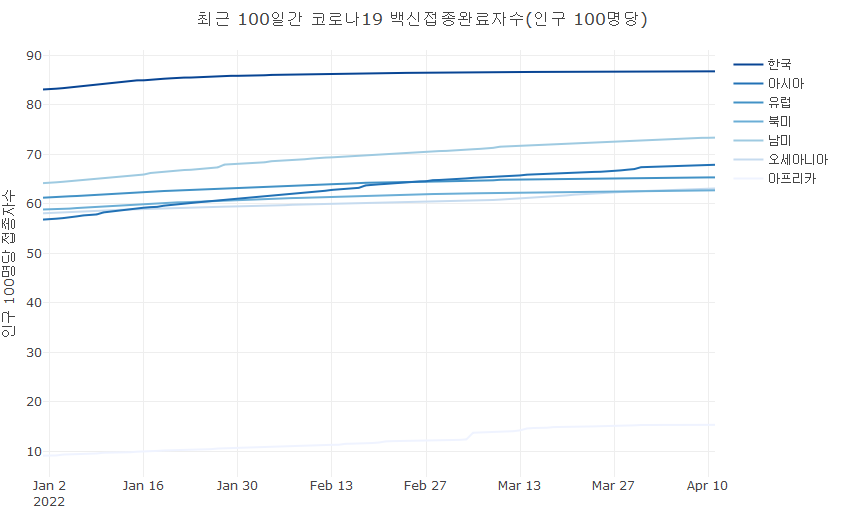
##### line, add\_lines()

스캐터 trace의 line type은 스캐터 trace에서 데이터를 이어주는 선 그래프를 생성하는 스캐터 trace type이다. line type은 marker와 같이 add\_trace()에 list()를 사용하여 리스트를 만들어 설정하거나 add\_lines()를 사용할 수 있다. ’marker’의 설정과 마찬가지로 주의할 것은 add\_trace(type = 'scatter', mode = 'lines')을 사용하여 line 속성의 list에 설정하는 속성 값들은 값을 설정할 수는 있지만 ~를 사용해 변수를 매핑할 수 없다는 점이고 I()를 사용하지 않고 값의 설정이 가능하다는 것이다.

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 선 그래프에 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.

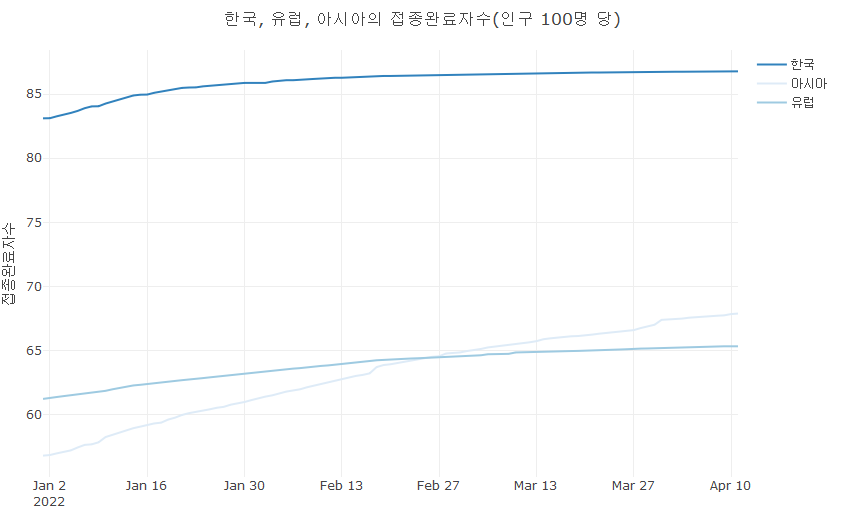
## 긴 형태의 100일 코로나19 데이터로 plotly 객체 생성  
p\_line <- df\_covid19\_100 |> plot\_ly()  
  
p\_line |>   
 ## type을 scatter, mode를 lines로 설정한 trace 추가  
 add\_trace(type = 'scatter', mode = 'lines',   
 ## X축을 date,   
 ## Y축을 people\_fully\_vaccinated\_per\_hundred로 매핑  
 x = ~date, y = ~people\_fully\_vaccinated\_per\_hundred,   
 ## color를 location으로 매핑  
 color = ~ location) |>  
 ## 제목, X, Y축 제목, 여백 설정  
 layout(title = '최근 100일간 코로나19 백신접종완료자수(인구 100명당)',   
 xaxis = list(title = list(text = NULL)),   
 yaxis = list(title = '인구 100명당 접종자수'), margin = margins)



실행결과 2- . 스캐터 trace line 모드의 color 설정 결과

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 긴 형태의 데이터프레임으로 변환하여야 하고 매핑된 변수의 카테고리가 너무 많으면 스파게티 선 그래프가 생성된다. 따라서 사용자가 그리기를 원하는 3~4개의 단일 선 그래프를 겹쳐 그리는 방법도 많이 사용된다. 특히 이 방법은 사람이 이해하기 쉬운 넓은 형태의 데이터프레임을 직접 사용할 수 있다.

## 넓은 형태의 100일 코로나19 데이터로 X축을 date로 매핑한 plotly 객체 생성  
df\_covid19\_100\_wide |> plot\_ly(x = ~date) |>  
 ## type을 scatter, mode를 lines로 설정한 trace 추가  
 add\_trace(type = 'scatter', mode = 'lines',   
 ## Y축을 백신접종완료자\_한국으로 매핑, name을 한국으로 설정  
 ## X축은 초기 생성된 plotly 객체에 이미 매핑됨  
 y = ~백신접종완료자\_한국, name = '한국') |>   
 ## type을 scatter, mode를 lines로 설정한 trace 추가  
 add\_trace(type = 'scatter', mode = 'lines',   
 ## Y축을 백신접종완료자\_아시아로 매핑, name을 아시아로 설정  
 y = ~백신접종완료자\_아시아, name = '아시아') |>   
 add\_trace(type = 'scatter', mode = 'lines',   
 ## Y축을 백신접종완료자\_유럽로 매핑, name을 유럽로 설정  
 y = ~백신접종완료자\_유럽, name = '유럽') |>  
 ## 제목, X, Y 축 제목, 여백 설정  
 layout(title = '한국, 유럽, 아시아의 접종완료자수(인구 100명 당)',   
 xaxis = list(title = list(text = NULL)),   
 yaxis = list(title = list(text = '접종완료자수')),   
 margin = margins)



실행결과 2- . trace를 추가하는 plotly 생성 방법

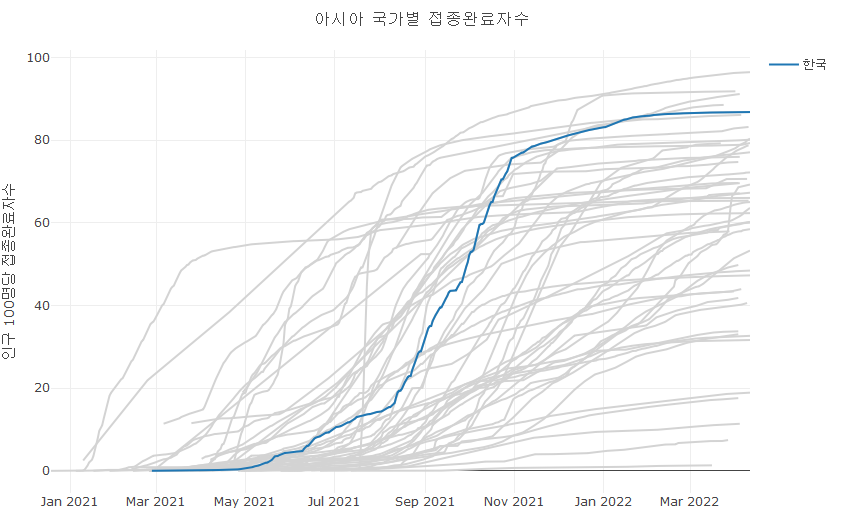
이렇게 trace를 추가하는 방법으로 시각화를 완성하면 몇 가지 장점이 있다.

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 사용자가 원하는 대로 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 개별적으로 설정한다. 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.

## 빈 plotly 객체 생성  
fig <- plot\_ly() |>   
 ## df\_covid19의 아시아 데이터만 필터링한 데이터를 사용하는 trace 추가  
 add\_trace(data = df\_covid19 |> filter(continent == 'Asia'),   
 ## type을 scatter, mode를 lines로 설정  
 type = 'scatter', mode = 'lines',   
 ## X, Y축의 변수 매핑  
 x = ~date, y = ~people\_fully\_vaccinated\_per\_hundred,   
 ## name의 변수 매핑(아래의 showlegen=F로 의미없는 설정)  
 name = ~iso\_code,   
 ## line의 세부 속성 설정  
 line = list(color = 'lightgray'),   
 ## 이 trace의 범례는 제거  
 showlegend = F,   
 ## 결측치에 의한 갭은 연결  
 connectgaps = TRUE)  
## 최종 plotly 결과는 fig로 저장  
  
fig <- fig |> ## 앞서 생성한 fig를 사용하여  
 ## df\_covid19의 한국 데이터만 필터링한 데이터를 사용하는 trace 추가  
 add\_trace(data = df\_covid19 |> filter(iso\_code == 'KOR'),   
 ## type을 scatter, mode를 lines로 설정  
 type = 'scatter', mode = 'lines',   
 ## X, Y축의 변수 매핑  
 x = ~date, y = ~people\_fully\_vaccinated\_per\_hundred,   
 ## name의 설정  
 name = '한국',   
 ## line의 세부 속성 설정  
 line = list(color = 'dardblue'),   
 ## 이 trace의 범례는 표시  
 showlegend = T)  
## 최종 plotly 결과는 fig로 저장  
  
fig |> ## 앞서 생성한 fig를 사용하여  
 ## 제목, X, Y축 제목, 범례 표시, 여백 설정  
 layout(title = list(text = '아시아 국가별 접종완료자수'),  
 xaxis = list(title = list(text = NULL)),   
 yaxis = list(title = '인구 100명당 접종완료자수'),   
 showlegend = T, margin = margins)



실행결과 2- . 스파게티 선 그래프의 특정 trace 강조

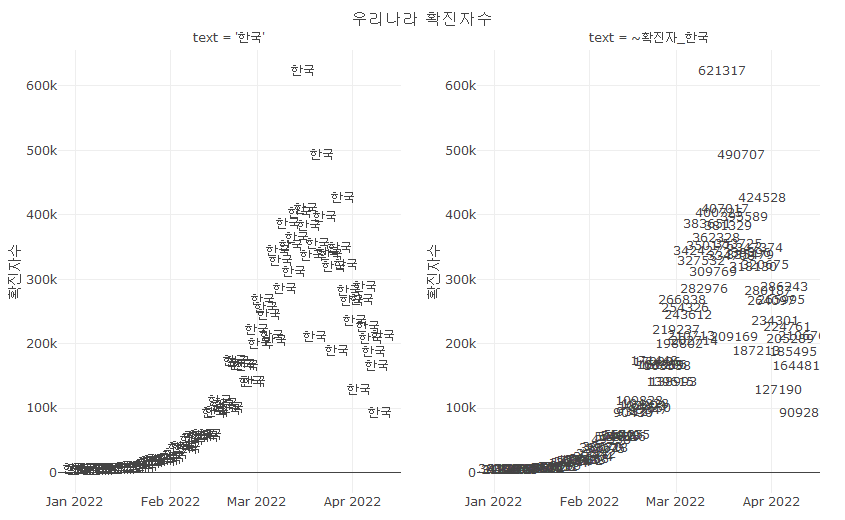
##### text, add\_text()

스캐터 trace의 text trace은 각 X축과 Y축에 매핑된 좌표에 표시될 문자열을 설정하는 trace이다. 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 text가 설정되고 hovertext가 설정되지 않은 경우에는 text에 설정된 문자열이 호버 레이블로 사용된다.

## 앞서 생성한 p\_line\_wide 객체에 trace 추가  
add\_trace(p\_line\_wide,   
 ## type을 scatter, mode를 text로 설정  
 type = 'scatter', mode = 'text',   
 ## X, Y축 변수 매핑  
 x = ~date, y = ~확진자\_한국,   
 ## text 매핑  
 text = '한국')|>  
 layout(title = list(text = '우리나라 확진자수'),  
 xaxis = list(title = list(text = NULL)),   
 yaxis = list(title = '확진자수'),   
 margin = margins)  
  
## 앞서 생성한 p\_line\_wide 객체에 trace 추가  
add\_trace(p\_line\_wide,   
 ## type을 scatter, mode를 text로 설정  
 type = 'scatter', mode = 'text',   
 ## X, Y축 변수 매핑  
 x = ~date, y = ~확진자\_한국,   
 ## text 매핑  
 text = ~확진자\_한국)|>  
 layout(title = list(text = '확진자수'),  
 xaxis = list(title = list(text = NULL)),   
 yaxis = list(title = '확진자수'),   
 margin = margins)



실행결과 2- . text 매핑과 설정 결과

#### 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 속성에 ’bar’를 설정함으로써 사용할 수 있고 add\_bar() 함수를 사용할 수 있다.

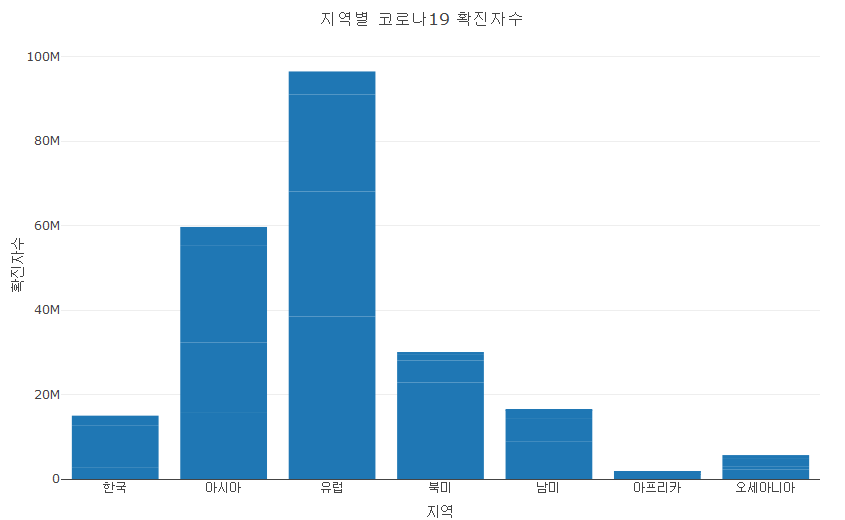
add\_trace(p, type = ‘bar’, …, data = NULL, inherit = TRUE)  
add\_bars(p, x = NULL, y = NULL, …, data = NULL, inherit = TRUE)  
- p : plot\_ly()로 생성한 plotly 객체  
- type : trace 타입을 설정, add\_markers()는 ‘scatter’로 설정  
- mode : ’scatter’ trace 중 어떤 도형을 사용할지 설정, add\_markers()는 ’markers’를 설정  
- text : 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

막대 trace에서 설정이 가능한 주요 속성은 다음과 같다.

| 속성 | 설명 | 속성 값 | 세부속성 |
| --- | --- | --- | --- |
| type | trace type 설정 | ‘bar’ |  |
| visible | trace의 표시 설정 | TRUE, FALSE, ‘legendonly’ |  |
| x0, y0 | x, y의 대체로 X, Y축의 시작점(x0, y0) | 수치나 X축에 매핑될 이산형 카테고리 변수 |  |
| dx, dy | x, y의 대체로 X, Y축의 변화량(dx, dy) | 수치 |  |
| base | 막대 base의 위치 설정 | 수치나 X축에 매핑될 이산형 카테고리 변수 |  |
| width | 막대의 너비 설정 | 0 이상의 수치값이나 수치 벡터 |  |
| offset | 막대 위치의 이동값 설정 | 수치나 수치 벡터 |  |
| textposition | 텍스트의 위치 설정 | ‘inside’, ‘outside’, ‘auto’, ‘none’ |  |
| orientation | 막대의 방향 설정 | ‘v’, ‘h’ |  |
| xperiod, yperiod | 축이 ‘date’ 타입인 경우 기간 위치를 설정 | 수치나 X축에 매핑될 이산형 카테고리 변수 |  |
| marker | 막대의 세부 설정 | 세부 속성들의 리스트 | color, colorbar, colorscale, line(color, width, …), opacity, pattern(bgcolor, fgcolor, fillmode, shape, …), gradient(color, type), symbol, size, sizemode, … |
| textfont | mode가 텍스트(text)인 경우 문자의 세부 설정 | 세부 속성들의 리스트 | color, family, size |

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.

## 최종 결과는 p\_bar에 저장  
p\_bar <-   
 ## 긴 형태의 100일 코로나19 데이터에서  
 df\_covid19\_100 |>   
 ## date에서 연월을 추출한 yearmonth 열을 생성  
 mutate(yearmonth = tsibble::yearmonth(date)) |>   
 ## location과 yearmonth로 그룹화  
 group\_by(location, yearmonth) |>  
 ## new\_cases의 합계 산출  
 summarise(new\_cases = sum(new\_cases)) |>   
 ## plotly 객체 생성  
 plot\_ly()  
  
## p\_bar를 사용  
p\_bar |>  
 ## 막대 타입 trace를 추가  
 add\_trace(type = 'bar',   
 ## X, Y 축에 변수 매핑  
 x = ~location, y = ~new\_cases) |>  
 ## 제목, X, Y축 제목, 여백 설정  
 layout(title = list(text = '지역별 코로나19 확진자수'),  
 xaxis = list(title = list(text = '지역')),   
 yaxis = list(title = '확진자수'),   
 margin = margins)



실행결과 2- . 막대 trace plotly 생성

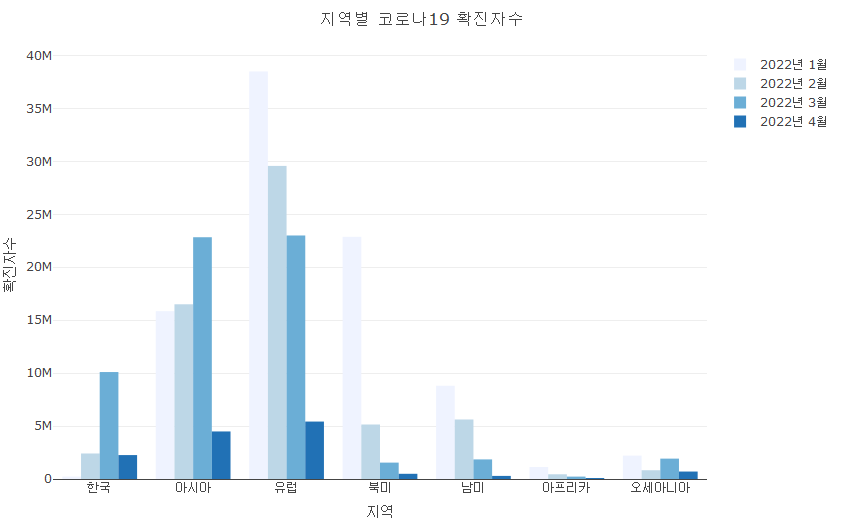
##### barmode

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.

이와 같이 여러개의 막대 trace를 겹쳐 그릴때 그 구성 형태를 결정하기 위해서는 차후 설명할 layout()의 barmode 속성을 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.

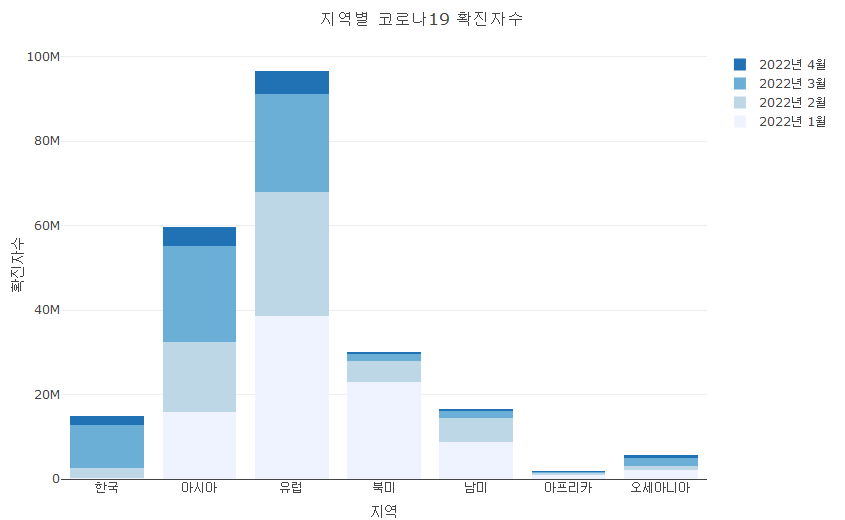
## 최종 결과는 p\_bar\_mode에 저장  
p\_bar\_mode <-   
 ## 긴 형태의 100일 코로나19 데이터에서  
 df\_covid19\_100 |>   
 ## date에서 연월을 추출한 yearmonth 열을 생성  
 mutate(yearmonth = tsibble::yearmonth(date)) |>   
 ## location과 yearmonth로 그룹화  
 group\_by(location, yearmonth) |>  
 ## new\_cases의 합계 산출  
 summarise(new\_cases = sum(new\_cases)) |>   
 ## plotly 객체 생성  
 plot\_ly()

## p\_bar\_mode를 사용하여  
p\_bar\_mode |>  
 ## 막대 trace 추가  
 add\_trace(type = 'bar',   
 ## X, Y축 변수 매핑  
 x = ~location, y = ~new\_cases,   
 ## color는 연월 팩터로 매핑  
 color = ~as.factor(yearmonth),   
 ## colors는 viridis 팔레트로 설정  
 colors = 'Blues',   
 ## 범례 표시 이름의 설정  
 name = ~paste0(lubridate::year(yearmonth), '년 ',   
 lubridate::month(yearmonth, label = TRUE,   
 abbr = FALSE))) |>  
 ## barmode를 group으로 설정  
 layout(barmode = 'group',   
 title = list(text = '지역별 코로나19 확진자수'),  
 xaxis = list(title = list(text = '지역')),   
 yaxis = list(title = '확진자수'),   
 margin = margins)



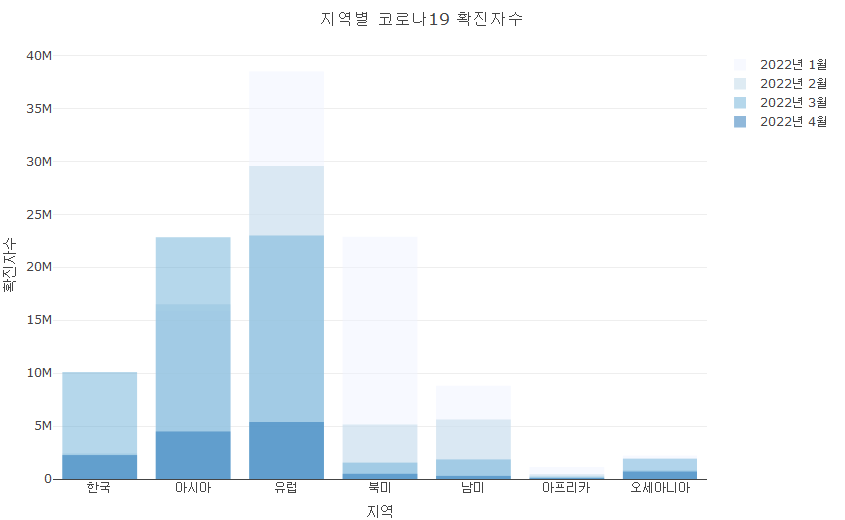
실행결과 2- . group형 막대 trace

p\_bar\_mode |>  
 add\_trace(type = 'bar', x = ~location, y = ~new\_cases,   
 color = ~as.factor(yearmonth), colors = 'Blues',   
 name = ~paste0(lubridate::year(yearmonth), '년 ',   
 lubridate::month(yearmonth, label = TRUE,   
 abbr = FALSE)  
 )  
 ) |>  
 ## barmode를 stack으로 설정  
 layout(barmode = 'stack',   
 title = list(text = '지역별 코로나19 확진자수'),  
 xaxis = list(title = list(text = '지역')),   
 yaxis = list(title = '확진자수'),   
 margin = margins)



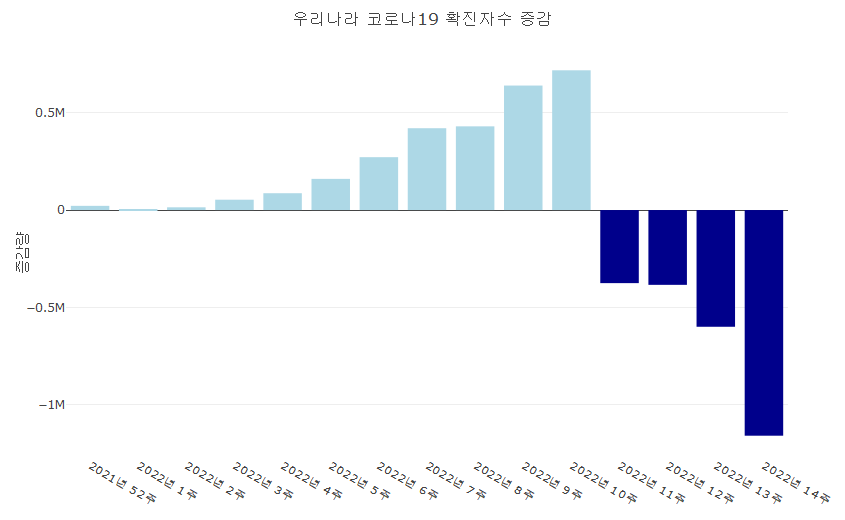
실행결과 2- . stack형 막대 trace

p\_bar\_mode |>  
 add\_trace(type = 'bar', x = ~location, y = ~new\_cases,   
 color = ~as.factor(yearmonth), colors = 'Blues', opacity = 0.5,   
 name = ~paste0(lubridate::year(yearmonth), '년 ',   
 lubridate::month(yearmonth, label = TRUE,   
 abbr = FALSE)  
 )  
 ) |>  
 ## barmode를 overlay으로 설정  
 layout(barmode = 'overlay',   
 title = list(text = '지역별 코로나19 확진자수'),  
 xaxis = list(title = list(text = '지역')),   
 yaxis = list(title = '확진자수'),   
 margin = margins)



실행결과 2- . overlay형 막대 trace

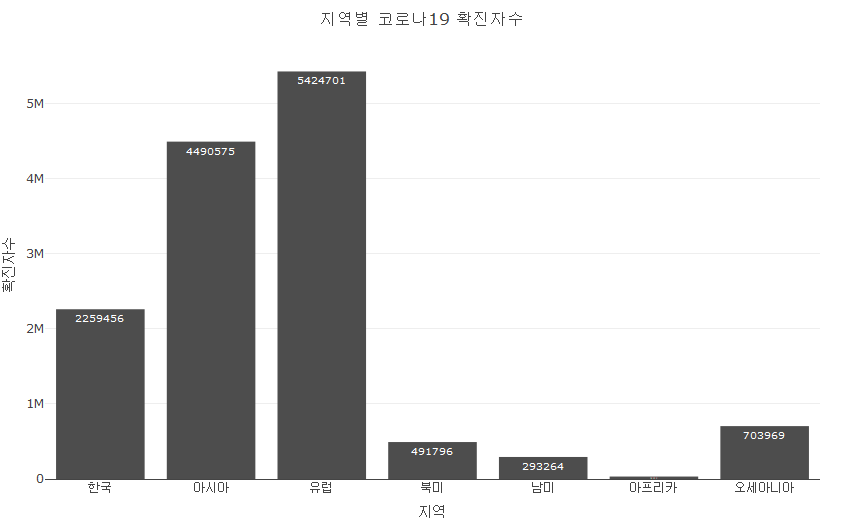
## 최종 결과는 diff\_sam에 저장  
diff\_kor <-   
 ## 긴 형태의 100일 코로나19 데이터에서  
 df\_covid19\_100 |>   
 ## date의 월 단위 열을 yearmonth에 저장  
 mutate(date\_by\_week = lubridate::floor\_date(date, "week"),   
 yearweekth = paste0(lubridate::year(date\_by\_week), '년 ',   
 lubridate::week(date\_by\_week), '주')) |>   
 ## iso\_code, yearmonth로 그룹화  
 group\_by(iso\_code, date\_by\_week, yearweekth) |>  
 ## new\_cases 합계 산출  
 summarise(new\_cases = sum(new\_cases))  
  
diff\_kor\_value <- diff\_kor |>   
 filter(iso\_code == 'KOR') |>   
 select(new\_cases) |> pull()  
  
diff\_kor\_ticktext <- diff\_kor |>   
 filter(iso\_code == 'KOR') |>   
 select(yearweekth) |> pull()  
  
## 빈 plotly객체 생성  
plot\_ly() |>   
 ## 막대 trace 추가  
 add\_trace(type = 'bar',   
 ## X축은 1부터 3까지, Y축은 전월 대비 차분 산출  
 x = 1:length(diff(diff\_kor\_value)), y = diff(diff\_kor\_value),   
 ## 차분이 0보다 크면 'darkred', 아니면 'darkblue'로 설정  
 color = I(ifelse(diff(diff\_kor\_value) > 0, "lightblue", "darkblue"))  
 ) |>   
 ## barmode를 relative로 설정  
 layout(barmode = 'relative',   
 xaxis = list(tickvals = 1:length(diff(diff\_kor\_value)),   
 ticktext = diff\_kor\_ticktext),  
 yaxis = list(title = '증감량'),   
 title = list(text = '우리나라 코로나19 확진자수 증감'),  
 margin = margins)



실행결과 2- . relative형 막대 trace

앞서 스캐터 trace의 선 그래프를 그릴 때, 넓은 형태의 데이터프레임을 사용하여 추가하고자 하는 열을 각각의 trace로 하나 하나 추가하여 전체 시각화를 완성하는 예를 보았다. 막대 trace도 이와 동일한 방법으로 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.

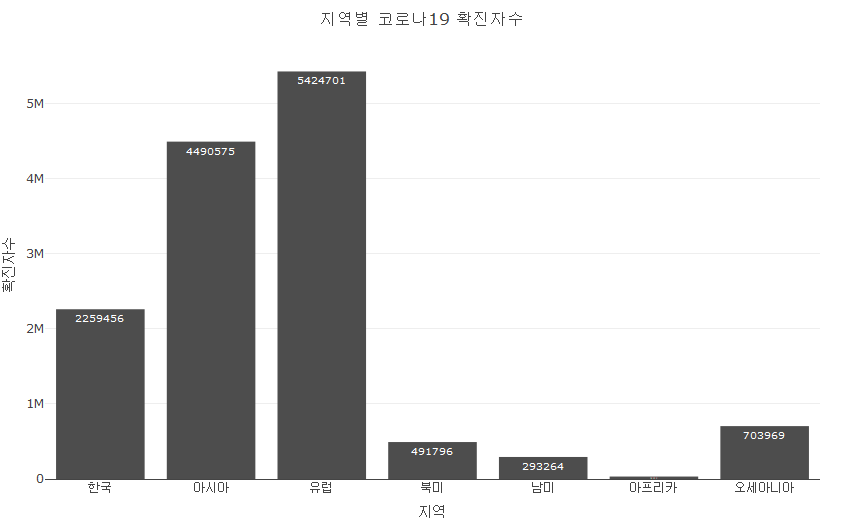
## 최종 결과는 p\_bar\_mode1에 저장  
p\_bar\_mode1 <-   
 ## 긴 형태의 100일 코로나19 데이터에서  
 df\_covid19\_100 |>   
 ## date의 월 단위 열을 yearmonth에 저장  
 mutate(yearmonth = lubridate::floor\_date(date, "month")) |>   
 ## location, yearmonth로 그룹화  
 group\_by(location, yearmonth) |>  
 ## new\_cases 합계 산출  
 summarise(new\_cases = sum(new\_cases)) |>   
 ## 연월을 열로 값을 new\_cases로 설정한 넓은 데이터프레임 생성  
 pivot\_wider(names\_from = yearmonth, values\_from = new\_cases) |>  
 ## plotlt 객체 생성  
 plot\_ly()  
  
## 최종 결과는 다시 fig에 저장  
fig <-   
 ## 앞서 생성한 fig에   
 fig |>  
 ## 막대 trace를 추가  
 add\_trace(type = 'bar',   
 ## X, Y축을 변수에 매핑, color를 설정  
 x = ~location, y = ~`2022-01-01`, color = I('gray40'),   
 ## 범례 이름 설정, text는 변수(특수문자가 포함되어 ` 사용) 매핑  
 name = '2022월 1월', text = ~`2022-01-01`,   
 ## textposition을 'inside'로 설정  
 textposition = 'inside',   
 ## textfont의 세부 속성 설정  
 textfont = list(color = 'white', size = 10)  
 )  
  
## 최종 결과는 다시 fig에 저장  
fig <-   
 ## 앞서 생성한 fig에   
 fig |>  
 ## 막대 trace를 추가  
 add\_trace(type = 'bar',   
 ## X, Y축을 변수에 매핑, color를 설정  
 x = ~location, y = ~`2022-02-01`, color = I('gray50'),   
 ## 범례 이름 설정, text는 변수(특수문자가 포함되어 ` 사용) 매핑  
 name = '2022월 2월', text = ~`2022-02-01`,   
 ## textposition을 'inside'로 설정  
 textposition = 'inside',   
 ## textfont의 세부 속성 설정  
 textfont = list(color = 'white', size = 10)  
 )  
   
## 최종 결과는 다시 fig에 저장  
fig <-   
 ## 앞서 생성한 fig에   
 fig |>  
 ## 막대 trace를 추가  
 add\_trace(type = 'bar',   
 ## X, Y축을 변수에 매핑, color를 설정  
 x = ~location, y = ~`2022-03-01`, color = I('darkblue'),   
 ## 범례 이름 설정, text는 변수(특수문자가 포함되어 ` 사용) 매핑  
 name = '2022월 3월', text = ~`2022-03-01`,   
 ## textposition을 'inside'로 설정  
 textposition = 'inside',   
 ## textfont의 세부 속성 설정  
 textfont = list(color = 'white', size = 10)  
 )  
  
## 최종 결과는 fig에 저장  
fig <- p\_bar\_mode1 |>  
 ## 막대 trace를 추가  
 add\_trace(type = 'bar',  
 ## X, Y축을 변수에 매핑, color를 설정  
 x = ~location, y = ~`2022-04-01`, color = I('gray30'),   
 ## 범례 이름 설정, text는 변수(특수문자가 포함되어 ` 사용) 매핑  
 name = '2022월 4월', text = ~`2022-04-01`,   
 ## textposition을 'inside'로 설정  
 textposition = 'inside',   
 ## textfont의 세부 속성 설정  
 textfont = list(color = 'white', size = 10)  
 )  
  
fig |>   
 ## barmode를 stack으로 설정  
 layout(barmode = 'stack',   
 title = list(text = '지역별 코로나19 확진자수'),  
 xaxis = list(title = '지역'),   
 yaxis = list(title = '확진자수'),   
 margin = margins  
 )



실행결과 2- . 막대 trace의 stack 모드의 text 설정 결과

앞서 만들어진 plotly 객체인 fig를 다음과 같이 ’group’형 막대 trace로 바꾸어 표시할 수 있다.

## 앞서 생성한 fig에   
fig |>   
 ## barmode를 group으로 설정  
 layout(barmode = 'group',   
 title = list(text = '지역별 코로나19 확진자수'),  
 xaxis = list(title = '지역'),   
 yaxis = list(title = '확진자수'),   
 margin = margins  
 )

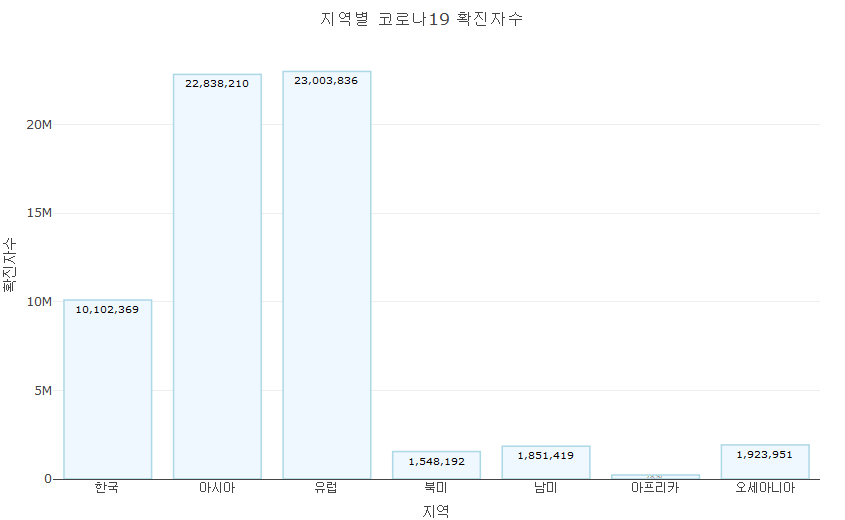


실행결과 2- . 막대 trace의 group 모드의 text 설정 결과

##### marker

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

## p\_bar\_mode1에  
p\_bar\_mode1 |>  
 ## 막대 trace 추가  
 add\_trace(type = 'bar',   
 x = ~location, y = ~`2022-03-01`, color = I('darkblue'),   
 text = ~`2022-03-01`,   
 textposition = 'inside',   
 textfont = list(color = 'black', size = 10),  
 ## texttemplate를 설정  
 texttemplate = '%{text:,}',   
 ## 막대 설정을 위해 marker 세부 속성 설정  
 marker = list(color = 'aliceblue',  
 line = list(color = 'lightblue',   
 width = 1.5)  
 )  
 ) |>  
 layout(title = list(text = '지역별 코로나19 확진자수'),  
 xaxis = list(title = '지역'),   
 yaxis = list(title= '확진자수'),  
 margin = margins)



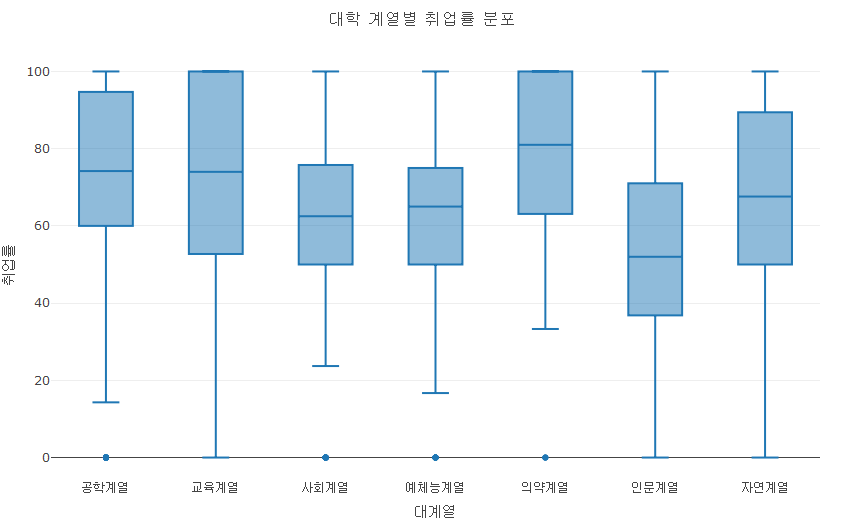
실행결과 2- . marker를 사용한 막대 설정 결과

#### 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 <-   
 ## 취업률 데이터를 사용하여 plotly 객체 생성  
 df\_취업률\_2000 |>   
 plot\_ly()  
  
p\_box |>   
 ## 박스 trace 추가  
 add\_trace(type = 'box',   
 ## X, Y축 변수 매핑  
 x = ~대계열, y = ~취업률) |>  
 ## 제목, 여백 설정  
 layout(title = list(text = '대학 계열별 취업률 분포'),   
 margin = margins)



실행결과 2- . 박스 trace 생성

박스 trace에서의 사용되는 주요 속성은 다음과 같다.

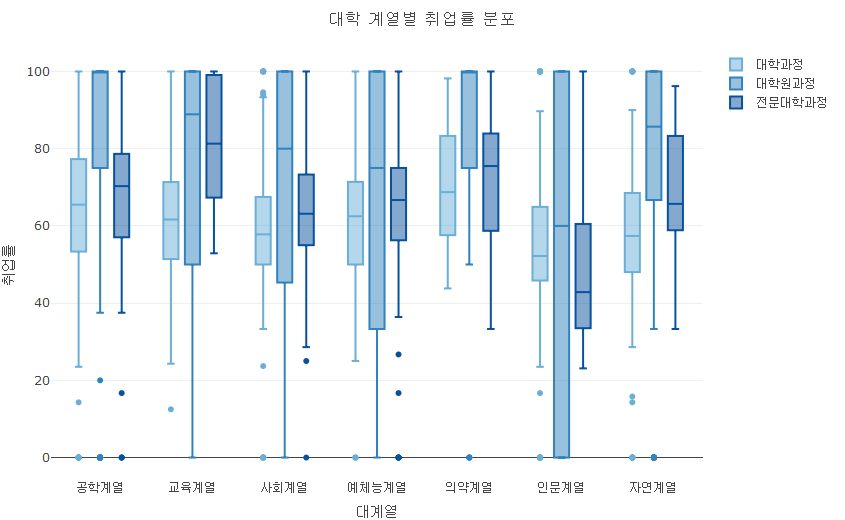
| 속성 | 설명 | 속성 값 | 세부 속성 |
| --- | --- | --- | --- |
| type | trace type 속성 | ‘box’ |  |
| visible | trace의 표시 설정 | TRUE, FALSE, ‘legendonly’ |  |
| width | 박스의 너비 설정 | 0이상의 수치값 |  |
| orientation | 박스의 방향 설정 | ‘v’, ‘h’ |  |
| marker | 박스의 세부 설정 | 세부 속성들의 리스트 | color, line(color, outliercolor, outlierwidth, width), opacity, outliercolor, size, symbol |
| line | 박스의 외곽선 설정 | 세부 속성들의 리스트 | color, width |
| boxmean | 박스의 평균값 표시 설정 | TRUE, FALSE, ‘sd’ |  |
| boxpoints | 데이터 값을 점으로 표시할 여부 설정 | ‘all’, ‘outlier’, ‘suspectedoutliers’, FALSE |  |
| norched | 노치 스타일 박스 설정 | 논리값 |  |
| notched width | 노치의 상대적 너비 설정 | 0~0.5 사이 값 |  |
| whisker width | 수염의 상대적 너비 설정 | 0~1 사이 값 |  |
| q1, median, q3, mean, sd | 박스를 구성하는 통계값 설정 | 데이터프레임 열, 리스트, 벡터 |  |
| lowerfence, upperfence | 하위펜스와 상위펜스 값의 설정 | 데이터프레임 열, 리스트, 벡터 |  |
| quantilemode | Q1과 Q3를 계산하는 방식 설정 | ‘linear’, ‘inclusive’, ‘exclusive’ |  |
| pointpos | 샘플 점이 그려지는 상대적 위치 설정 | -2~2까지의 수치 |  |
| jitter | 샘플 점이 그려지는 너비 설정 | 0 ~ 1 사이 값 |  |

##### boxmode

박스 trace도 막대 trace와 같이 그룹화한 변수를 color나 linetype등의 속성에 매핑시켜서 다수의 박스 trace를 한번에 만들거나 각각의 박스 trace를 추가하여 그릴 수 있다. 이렇게 여러개의 박스 trace를 그릴 때 그 구성 형태를 결정하기 위해서는 차후 설명할 layout()의 boxmode 속성을 설정함으로써 가능하다.[[8]](#footnote-65)

boxmode 속성은 ’group’과 ’overlay’의 두 가지를 설정할 수 있다. ’group’은 각각의 박스들이 옆으로 배치되면서 전체 박스 플롯이 완성되고 ’overlay’는 각각의 박스들이 겹쳐져 그려지면서 완성된다. 다음의 코드는 color로 과정구분을 매핑하여 ’group’형 박스 플롯을 생성하는 코드이다.

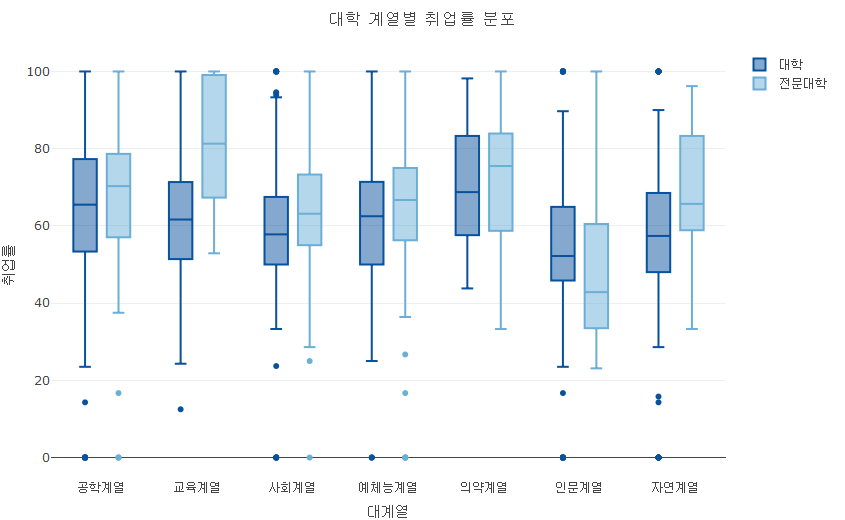
p\_box\_group <- p\_box |>   
 add\_trace(type = 'box',   
 x = ~대계열, y = ~취업률,   
 ## color를 과정구분으로 매핑  
 color = ~과정구분)  
  
p\_box\_group |>   
 ## boxmode를 group으로 설정  
 layout(boxmode = "group") |>  
 layout(title = list(text = '대학 계열별 취업률 분포'),   
 margin = margins)



실행결과 2- . 박스 trace의 group mode 실행 결과

박스 trace도 앞선 막대 trace와 같이 넓은 형태의 데이터프레임에 각 열을 하나의 박스 trace로 추가함으로써 전체 막대 trace를 만들어 줄 수 있다.

p\_box\_add <- p\_box |>   
 ## df\_취업률\_2000에서 대학과정 데이터를 필터링한 데이터를 사용한 박스 trace 추가  
 add\_trace(type = 'box', data = df\_취업률\_2000 |> filter(과정구분 == '대학과정'),  
 ## X, Y축을 변수에 매핑, 범례이름을 대학으로 설정  
 x = ~대계열, y = ~취업률, name = '대학')  
  
p\_box\_add <- p\_box\_add |>   
 ## df\_취업률\_2000에서 대학과정 데이터를 필터링한 데이터를 사용한 박스 trace 추가  
 add\_trace(type = 'box', data = df\_취업률\_2000 |> filter(과정구분 == '전문대학과정'),   
 ## X, Y축을 변수에 매핑, 범례이름을 대학으로 설정  
 x = ~대계열, y = ~취업률, name = '전문대학')  
  
p\_box\_add |>   
 ## boxmode를 group으로 설정  
 layout(boxmode = "group",   
 title = list(text = '대학 계열별 취업률 분포'),   
 margin = margins)

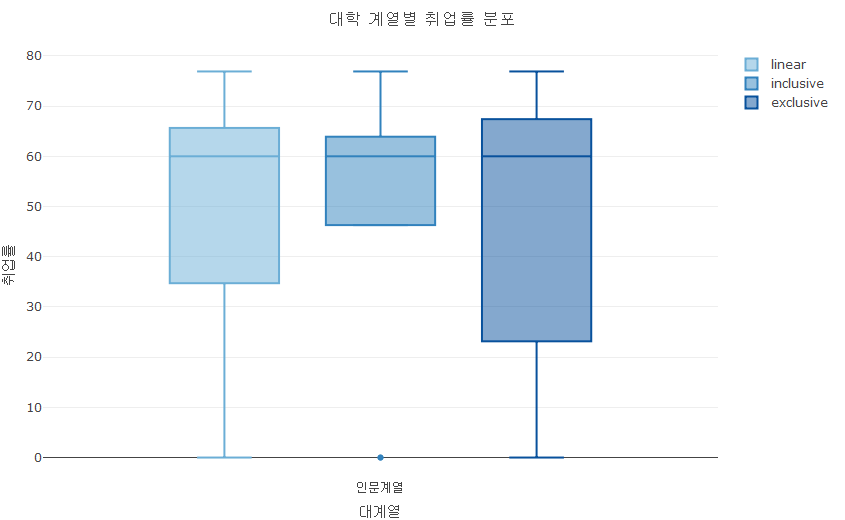


실행결과 2- . 여러 박스 trace를 추가

##### 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\_취업률\_2000에서   
 df\_취업률\_2000 |>   
 ## 인문계열의 상위 9개 데이터 추출  
 filter(대계열 == '인문계열') |> head(9) |>   
 ## plotly 객체 생성  
 plot\_ly()  
  
p\_box\_quartilemethod <- p\_box\_quartilemethod |>   
 ## quartilemethod가 linear인 박스 trace 추가  
 add\_trace(type = 'box', x = ~대계열, y = ~취업률,   
 quartilemethod = 'linear', name = 'linear')  
  
p\_box\_quartilemethod <- p\_box\_quartilemethod |>   
 ## quartilemethod가 inclusive인 박스 trace 추가  
 add\_trace(type = 'box', x = ~대계열, y = ~취업률,   
 quartilemethod = 'inclusive', name = 'inclusive')  
  
p\_box\_quartilemethod <- p\_box\_quartilemethod |>   
 ## quartilemethod가 exclusive인 박스 trace 추가  
 add\_trace(type = 'box', x = ~대계열, y = ~취업률,   
 quartilemethod = 'exclusive', name = 'exclusive')  
  
p\_box\_quartilemethod |>  
 ## boxmode를 group으로 설정  
 layout(boxmode = "group",   
 title = list(text = '대학 계열별 취업률 분포'),   
 margin = margins)

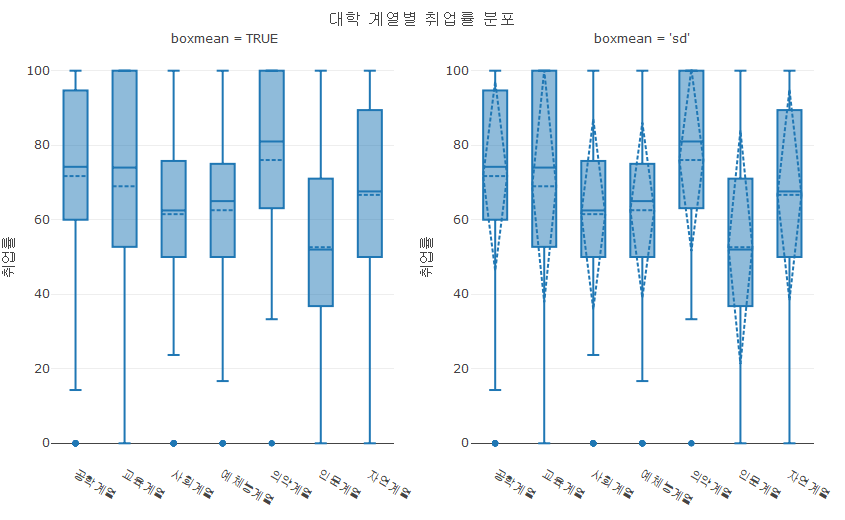


실행결과 2- . quartilemethod의 설정에 따른 결과

##### 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 속성의 설정만으로 간단하게 평균값을 표현할 수 있다. boxmean은 TRUE/FALSE의 논리값에 표준편차가 추가로 표시되는 ’sd’를 제공한다.

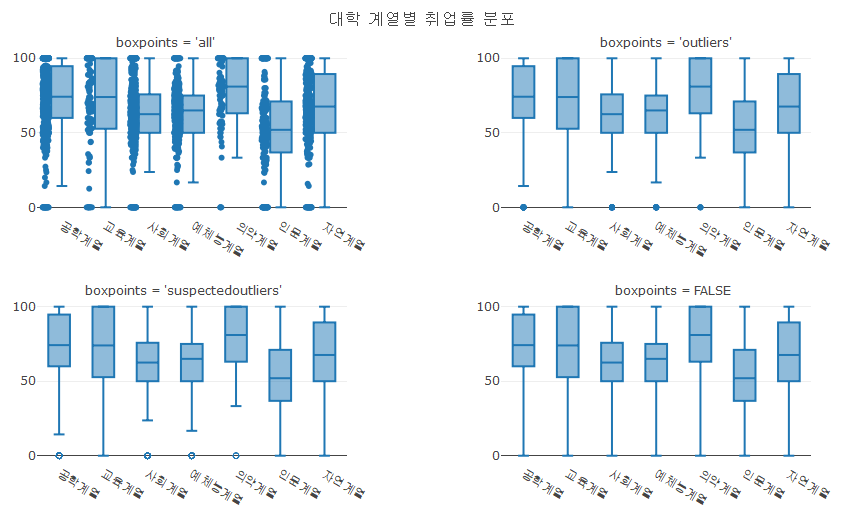
p\_box |>   
 ## boxmean을 TRUE로 설정한 박스 trace 추가  
 add\_trace(type = 'box', boxmean = TRUE,   
 x = ~대계열, y = ~취업률) |>  
 layout(title = list(text = '대학 계열별 취업률 분포'),   
 margin = margins)  
  
p\_box |>   
 ## boxmean을 sd로 설정한 박스 trace 추가  
 add\_trace(type = 'box', boxmean = 'sd',   
 x = ~대계열, y = ~취업률) |>  
 layout(title = list(text = '대학 계열별 취업률 분포'),   
 margin = margins)



실행결과 2- . boxmean 설정에 따른 결과

또 박스 trace에서 유용하게 사용되는 속성이 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 |>   
 ## boxpoint을 all로 설정한 박스 trace 추가  
 add\_trace(type = 'box', boxpoints = 'all',   
 x = ~대계열, y = ~취업률) |>  
 layout(title = list(text = '대학 계열별 취업률 분포'),   
 margin = margins)  
  
p\_box |>   
 ## boxpoint을 outliers로 설정한 박스 trace 추가  
 add\_trace(type = 'box', boxpoints = 'outliers'   
 , x = ~대계열, y = ~취업률) |>  
 layout(title = list(text = '대학 계열별 취업률 분포'),   
 margin = margins)  
  
p\_box |>   
 ## boxpoint을 suspectedoutliers로 설정한 박스 trace 추가  
 add\_trace(type = 'box', boxpoints = 'suspectedoutliers',   
 x = ~대계열, y = ~취업률) |>  
 layout(title = list(text = '대학 계열별 취업률 분포'),   
 margin = margins)  
  
p\_box |>   
 ## boxpoint을 FALSE로 설정한 박스 trace 추가  
 add\_trace(type = 'box', boxpoints = FALSE,   
 x = ~대계열, y = ~취업률) |>  
 layout(title = list(text = '대학 계열별 취업률 분포'),   
 margin = margins)

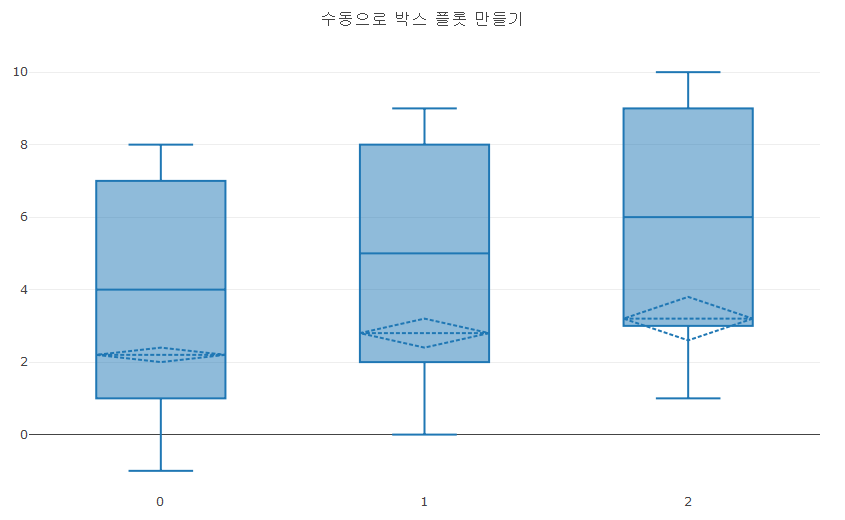


실행결과 2- . boxpoints 설정에 따른 결과

##### 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 속성은 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.

## 빈 plotly 객체 생성  
plot\_ly() |>   
 ## 박스 trace 추가  
 add\_trace(type= 'box',   
 ## 세개의 막대를 설정하기 위해   
 ## 통계값을 모두 길이가 3인 벡터로 설정  
 ## q1과 q3 설정  
 q1=c(1, 2, 3), q3=c(7, 8, 9 ),  
 ## median, mean, sd 설정  
 median=c(4, 5, 6), mean=c(2.2, 2.8, 3.2 ),   
 sd=c(0.2, 0.4, 0.6),  
 ## lowerfence, upperfence 설정  
 lowerfence=c(-1, 0, 1), upperfence=c(8, 9, 10)) |>  
 layout(title = list(text = '수동으로 박스 플롯 만들기'),   
 margin = margins)

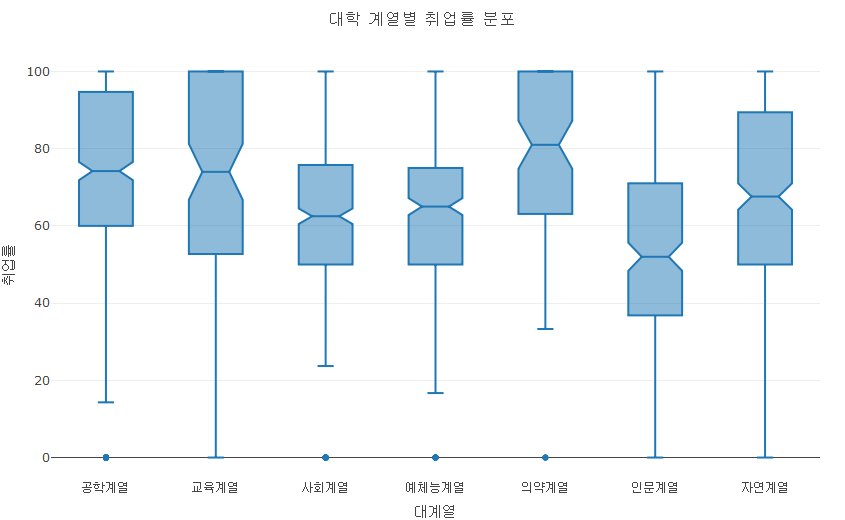


실행결과 2- . 수동으로 만든 박스 trace

##### 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 속성이 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 속성이다. 막대 외부에 표시되는 수염의 끝에 있는 가로선의 크기를 설정하는 속성이 whiskerwidth이다. 이 속성도 notchwidth와 같이 0부터 0.5사이의 값을 가진다.

p\_box |>   
 ## 박스 trace를 추가  
 add\_trace(type = 'box', x = ~대계열, y = ~취업률,   
 ## notch 스타일의 설정  
 notched = TRUE, notchewidth = 0.2,   
 notchespan = 0.5,   
 ## 수염 길이 설정  
 whiskerwidth = 0.3) |>  
 layout(title = list(text = '대학 계열별 취업률 분포'),   
 margin = margins)



실행결과 2- . norch가 적용된 박스 trace

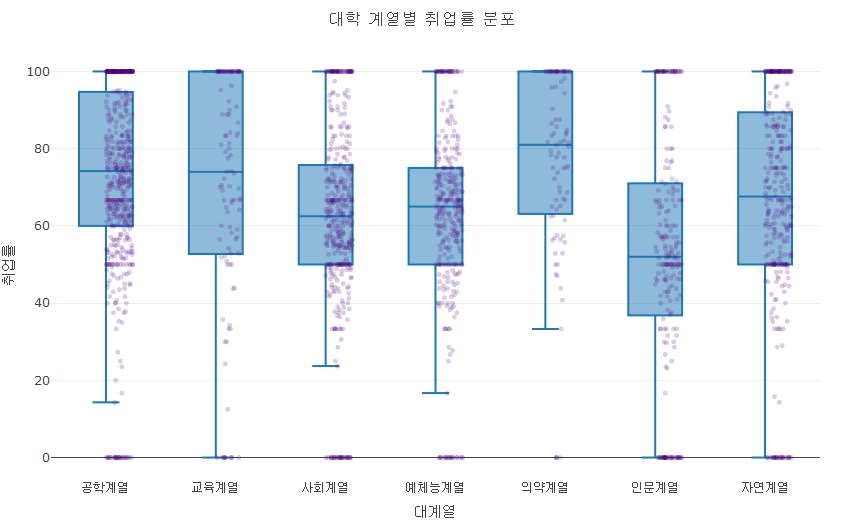
##### marker, jitter

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 속성이다. 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 속성으로 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,   
 ## 점 표시를 위한 marker 세부 속성 설정  
 marker = list(opacity = 0.2, size = 5, color = 'indigo'),  
 ## pointpos, jitter 설정  
 boxpoints = "all", jitter = 0.5, pointpos = 0.5) |>  
 layout(title = list(text = '대학 계열별 취업률 분포'),   
 margin = margins)

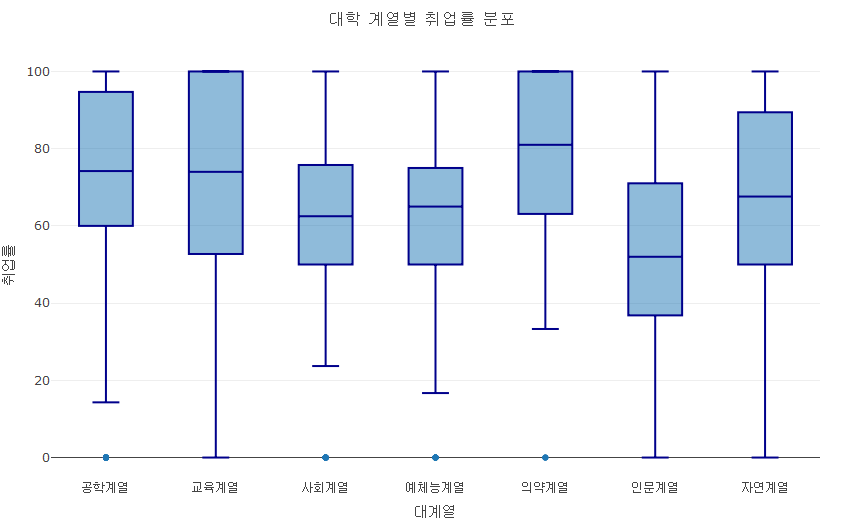


실행결과 2- . 점이 표시된 박스 trace

##### line

line은 막대 trace에서 외곽선을 설정하는 속성이다. 이 속성에서는 세부 선의 색(color)과 두께(width)를 설정할 수 있다.

p\_box |>   
 add\_trace(type = 'box', x = ~대계열, y = ~취업률,   
 ## 박스 trace의 선 세부 속성 설정  
 line = list(width = 2, color = 'darkblue')) |>  
 layout(title = list(text = '대학 계열별 취업률 분포'),   
 margin = margins)



실행결과 2- . 막대 trace의 선 설정 결과

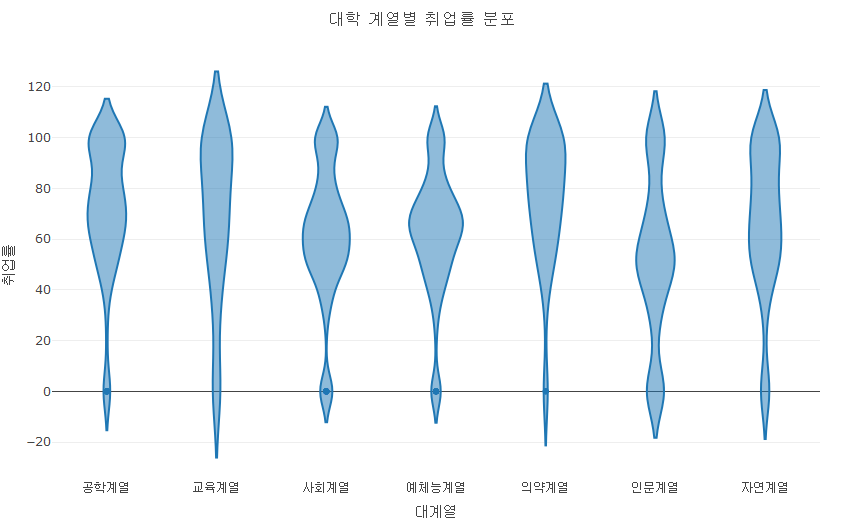
#### 바이올린(Violin) trace : 바이올린 플롯

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\_취업률\_2000 |> plot\_ly()  
  
p\_violin |>   
 ## 바이올린 trace 추가  
 add\_trace(type = 'violin', x = ~대계열, y = ~취업률) |>  
 layout(title = list(text = '대학 계열별 취업률 분포'),   
 margin = margins)



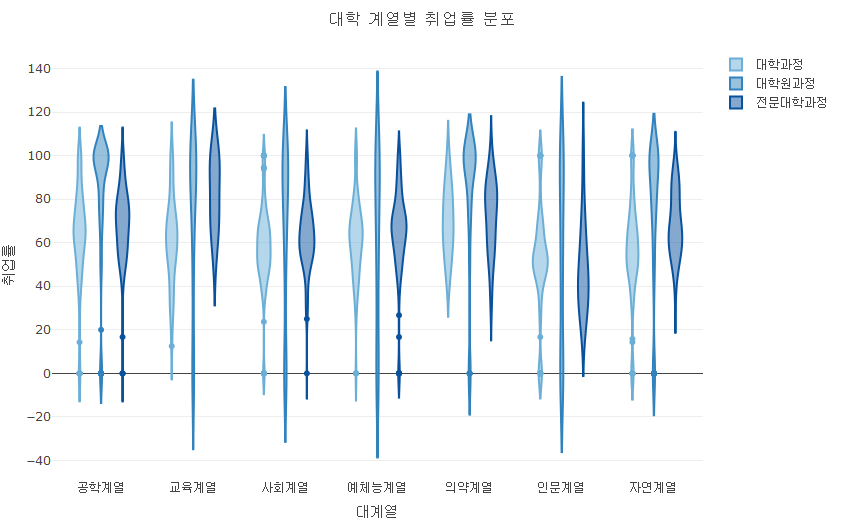
실행결과 2- . 바이올린 trace 생성

##### violinemode

바이올린 trace도 막대 trace와 같이 그룹화한 변수를 color나 linetype등의 속성에 매핑시켜서 다수의 바이올린 grace를 한번에 만들거나 각각의 바이올린 trace를 추가하여 그릴 수 있다. 이렇게 여러개의 바이올린 trace를 그릴 때 그 구성 형태는 막대 trace와 같이 layout()의 violinmode 속성을 설정함으로써 가능하다.

violinmode 속성은 ’group’과 ’overlay’의 두 가지를 설정할 수 있다. ’group’은 각각의 박스들이 옆으로 배치되면서 전체 바이올린 플롯이 완성되고 ’overlay’는 각각의 바이올린 박스들이 겹쳐져 그려지면서 완성된다. 다음의 코드는 color로 과정구분을 매핑하여 ’group’형 바이올린 플롯을 생성하는 코드이다.

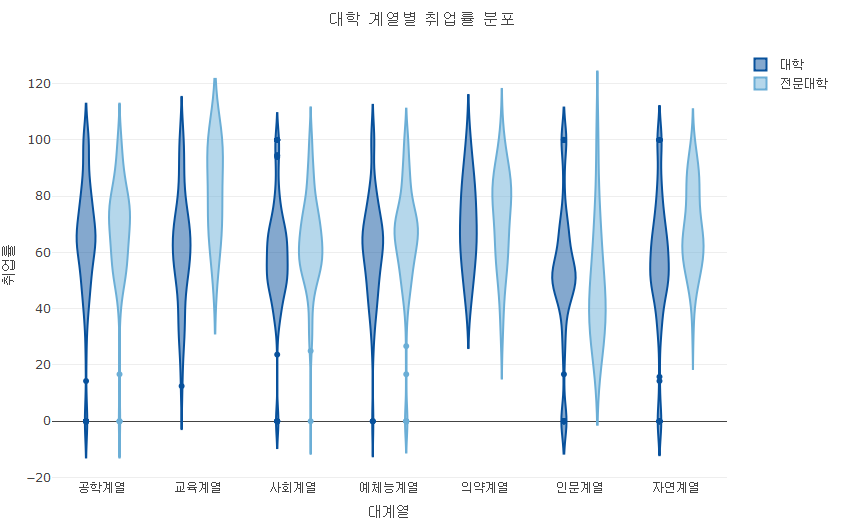
p\_violin |>   
 add\_trace(type = 'violin', x = ~대계열, y = ~취업률,   
 ## color를 변수에 매핑  
 color = ~과정구분) |>  
 ## violinmode를 group으로 설정  
 layout(violinmode = 'group',   
 title = list(text = '대학 계열별 취업률 분포'),   
 margin = margins)



실행결과 2- . color가 매핑된 바이올린 trace

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

p\_violin |>   
 ## 대학과정만 필터링한 데이터를 사용한 바이올린 trace 추가  
 add\_trace(data = df\_취업률\_2000 |> filter(과정구분 == '대학과정'),   
 type = 'violin', x = ~대계열, y = ~취업률, name = '대학') |>   
 ## 전문대학과정만 필터링한 데이터를 사용한 바이올린 trace 추가  
 add\_trace(data = df\_취업률\_2000 |> filter(과정구분 == '전문대학과정'),   
 type = 'violin', x = ~대계열, y = ~취업률, name = '전문대학') |>   
 ## violinmode를 group으로 설정  
 layout(violinmode = 'group',   
 title = list(text = '대학 계열별 취업률 분포'),   
 margin = margins)



실행결과 2- . 여러 바이올린 trace의 추가

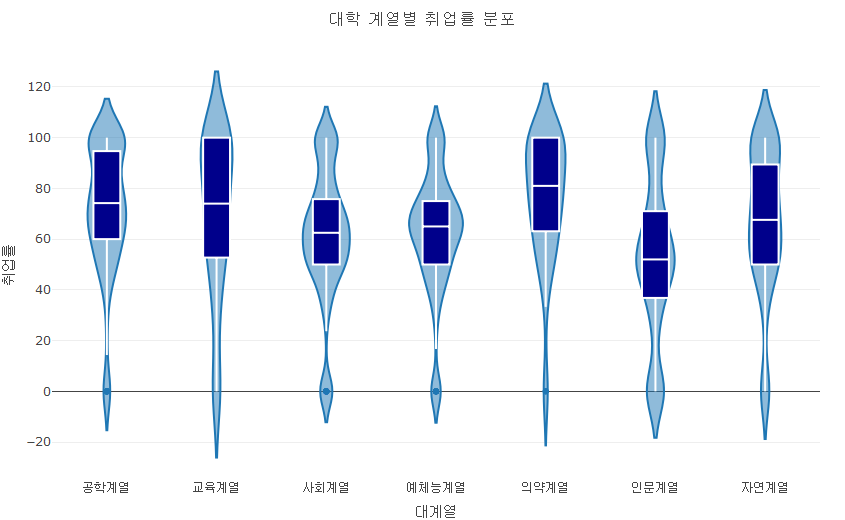
바이올린 trace에서 사용되는 주요 속성은 다음과 같다.

| 속성 | 설명 | 속성 값 | 세부 속성 |
| --- | --- | --- | --- |
| type | trace type 속성 | ‘violin’ |  |
| visible | trace의 표시 설정 | TRUE, FALSE, ‘legendonly’ |  |
| width | 바이올린 trace 너비 설정 | 0 이상의 수치값 |  |
| orientation | 바이올린 trace의 방향 설정 | ‘v’, ‘h’ |  |
| marker | 바이올린 trace의 세부 설정 | 세부 속성들의 리스트 | color, line(color, outliercolor, outlierwidth, width), opacity, outliercolor, size, symbol |
| line | 바이올린 trace 외곽선 설정 | 세부 속성들의 리스트 | color, width |
| box | 바이올린 trace 내부에 표현되는 박스 trace 설정 | 세부 속성들의 리스트 | fillcolor, line(color, width), visible, width |
| pointpos | 샘플 점이 그려지는 상대적 위치 설정 | -2~2까지의 수치 |  |
| jitter | 샘플 점이 그려지는 너비 설정 | 0 ~ 1 사이 값 |  |
| meanline | 평균 선의 설정 | 세부 속성들의 리스트 | color, visible, width |
| points | 데이터 값을 점으로 표시 여부 설정 | ‘all’, ‘outlier’, ‘suspectedoutliers’, FALSE |  |

##### 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 속성은 box이고 이 box 속성은 세부 타입을 리스트로 설정하는 속성이다. box의 세부 속성은 fillcolor, line(세부 속성으로 color, width), visible, width의 네 가지이다.

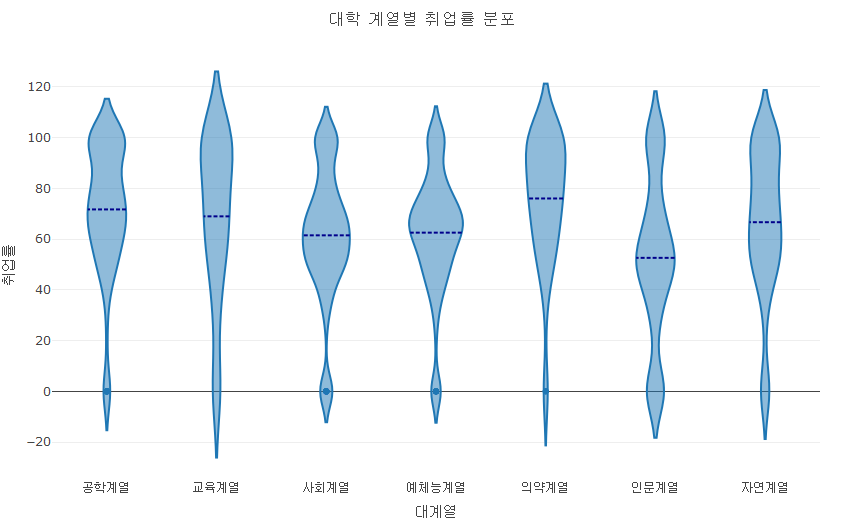
p\_violin |>   
 add\_trace(type = 'violin', x = ~대계열, y = ~취업률,   
 ## 박스의 세부 속성 설정  
 box = list(visible = TRUE, fillcolor = 'darkblue',   
 line = list(color = 'white'),   
 width = 0.5)) |>   
 layout(violinmode = 'group',   
 title = list(text = '대학 계열별 취업률 분포'),   
 margin = margins)



실행결과 2- . 바이올린 trace과 박스 trace의 병합

meanline은 바이올린 trace에 평균 선을 설정하는 속성으로 color, visible, width의 세부 속성을 리스트로 설정하여 사용한다.

p\_violin |>  
 add\_trace(type = 'violin', x = ~대계열, y = ~취업률,   
 ## meanline 세부 속성 설정  
 meanline = list(visible = TRUE, color = 'darkblue',   
 width = 2)) |>   
 layout(violinmode = 'group',   
 title = list(text = '대학 계열별 취업률 분포'),   
 margin = margins)



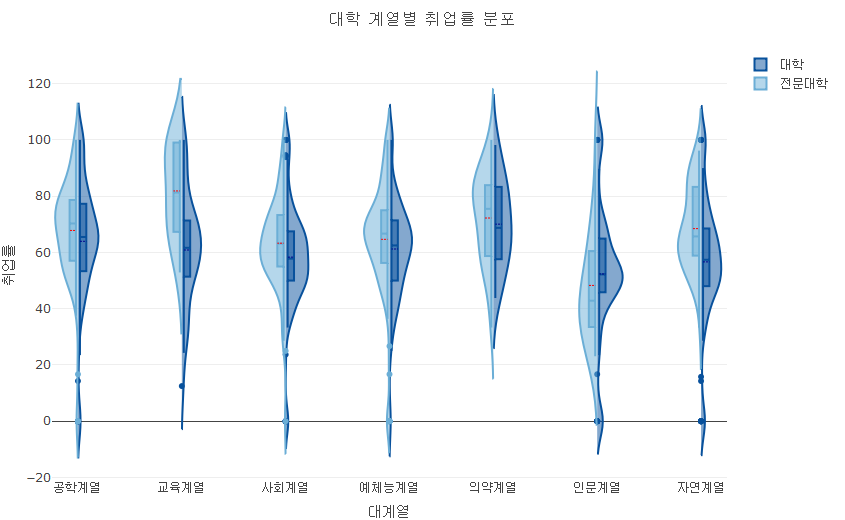
실행결과 2- . meanline이 추가된 바이올린 trace

##### side

앞의 예에서 대학과 전문대학의 바이올린 trace를 각각 추가함으로써 그룹화된 바이올린 플롯을 그렸다. 그런데 이 두 바이올린 플롯을 반씩 붙여서 그리면 바이올린 박스가 반으로 줄기 때문에 데이터를 확인하기 쉬울 것이다.

이렇게 두개의 바이올린 플롯을 반씩 잘라 붙이는 속성이 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 붙여줄 수 있다. 앞서 그렸던 대학과 전문대학의 계열별 바이올린 플롯을 붙이는 코드는 다음과 같다.

p\_violin |>   
 ## 대학과정을 필터링한 데이터 설정  
 add\_trace(data = df\_취업률\_2000 |> filter(과정구분 == '대학과정'),  
 ## 바이올린 trace로 추가  
 type = 'violin', x = ~대계열, y = ~취업률, name = '대학',   
 ## side, box의 설정  
 side = 'positive', box = list(visible = TRUE, width = 0.5),   
 ## meanline의 속성 설정  
 meanline = list(visible = TRUE, color = 'red', width = 1)) |>  
 ## 전문대학과정을 필터링한 데이터 설정  
 add\_trace(data = df\_취업률\_2000 |> filter(과정구분 == '전문대학과정'),   
 type = 'violin', x = ~대계열, y = ~취업률, name = '전문대학',   
 side = 'negative', box = list(visible = TRUE, width = 0.5),   
 meanline = list(visible = TRUE, color = 'red', width = 1)) |>   
 layout(violinmode = "overlay",   
 title = list(text = '대학 계열별 취업률 분포'),   
 margin = margins)

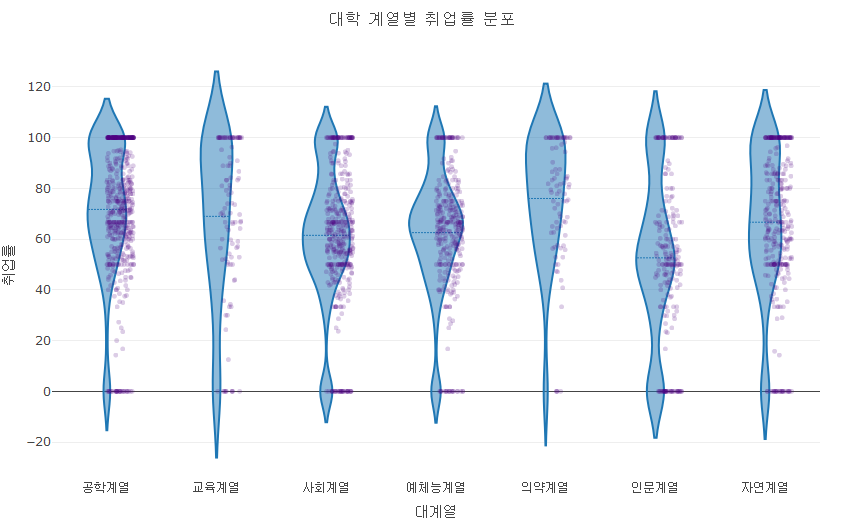


실행결과 2- . 바이올린 trace와 박스 trace의 side 병합

##### marker, jitter, line

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

p\_violin |>  
 add\_trace(type = 'violin', x = ~대계열, y = ~취업률,   
 ## meanline의 세부 속성 설정  
 meanline = list(visible = TRUE, color = 'dardred',   
 width = 1),   
 ## marker의 세부 속성 설정  
 marker = list(opacity = 0.2, size = 5, color = 'indigo'),  
 ## point, jitter, pointpos 설정  
 points = "all", jitter = 0.5, pointpos = 0.5) |>   
 layout(title = list(text = '대학 계열별 취업률 분포'),   
 margin = margins)



실행결과 2- . 점이 표시된 바이올린 trace

#### 파이(pie) trace : 원 그래프

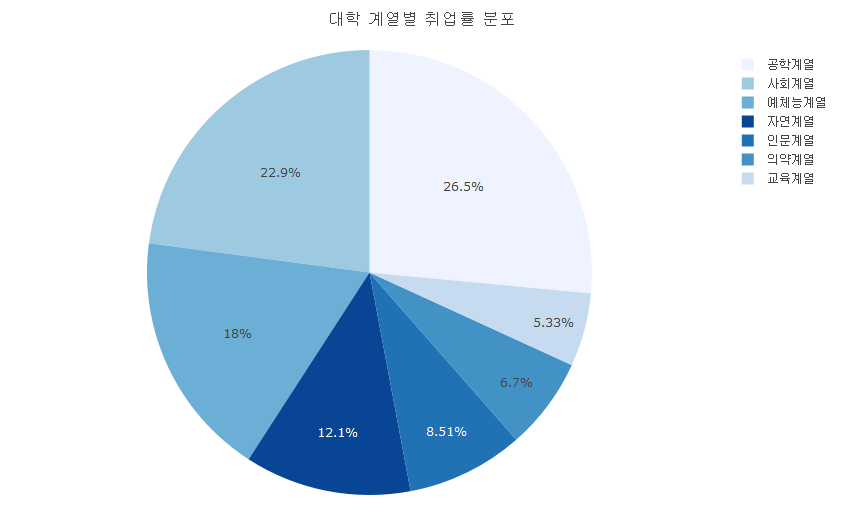
ggplot2에서 지원하지 않는 그래프 스타일이 원 그래프이다. 그래서 ggplot2에서 원 그래프를 그리기 위해서는 막대 그래프를 그리고 이 막대 그래프를 극 좌표계를 사용하여 돌리는 트릭을 사용한다. 원 그래프는 데이터를 정확히 분석하기 어렵다는 차원에서 데이터 분석가들이 잘 사용하지 않는 그래프 타입이지만 현실적으로 많이 사용되고 있기 때문에 plotly에서는 trace로 지원한다.

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 속성의 설정이 없다. 대신 필수적 속성으로 values와 labels를 설정해야한다.

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.

## 대계열별 졸업자수 합계 데이터로 설정된 plotly 객체 생성  
p\_pie <- df\_취업률\_2000 |> group\_by(대계열) |>  
 summarise(졸업자수 = sum(졸업자수)) |>   
 plot\_ly()  
  
p\_pie |>  
 ## value와 labels를 매핑한 파이 trace 추가  
 add\_trace(type = 'pie', values = ~졸업자수, labels = ~대계열) |>   
 layout(title = list(text = '대학 계열별 취업률 분포'),   
 margin = margins)



실행결과 2- . 파이 trace의 생성

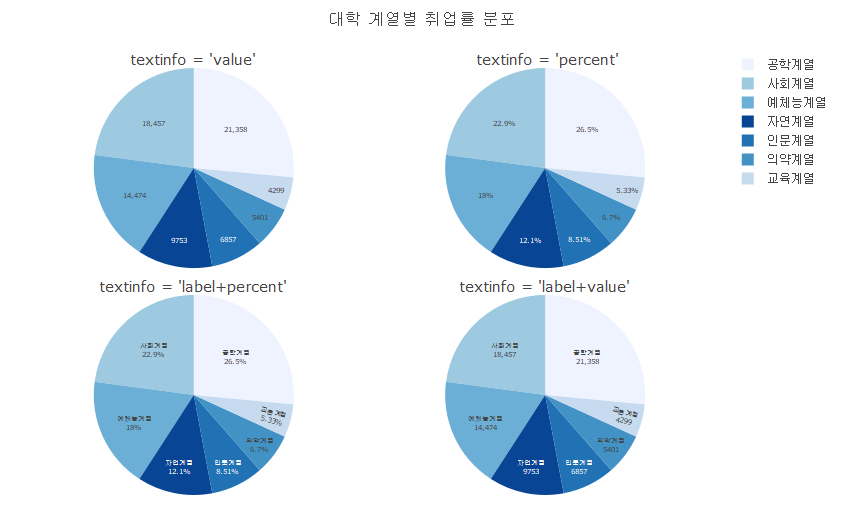
파이 trace에서 사용되는 주요 속성은 다음과 같다.

| 속성 | 설명 | 속성 값 | 세부 속성 |
| --- | --- | --- | --- |
| type | trace type 속성 | ‘pie’ |  |
| visible | trace의 표시 설정 | TRUE, FALSE, ‘legendonly’ |  |
| title | 파이 trace의 제목 설정 | 세부 속성들의 리스트 | font(color, family, size), position, text |
| values | 파이 trace의 각 섹터에 적용될 값 설정 | 데이터프레임 열, 리스트, 벡터 |  |
| labels | 파이 trace의 각 섹터에 적용될 라벨 설정 | 데이터프레임 열, 리스트, 벡터 |  |
| pull | 특정 섹터를 빼내기 위해 사용하는 반경 설정 | 0이상의 수치 나 수치 벡터 |  |
| marker | 파이의 세부 속성 설정 | 세부 속성들의 리스트 | color, line(color, width) |
| textfont | 파이 trace의 글꼴 설정 | 세부 속성들의 리스트 | color, family, size |
| direction | 파이 trace의 순서 방향 설정 | ‘clockwise’, ‘counterclockwise’ |  |
| hole | 파이 trace를 도넛형으로 만들기 위한 구멍 설정 | 0~1 사이의 수치 |  |
| insidetextfont, outsidetextfont | 파이 trace의 내부, 외부 글꼴 설정 | 세부 속성들의 리스트 | color, family, size |
| insidetextorientation, outsidetextorientation | 내부, 외부 문자의 표시 방향 설정 | ‘horizontal’, ‘radial’, ‘tangential’, ‘auto’ |  |
| rotation | 파이 trace의 시작 각도 설정 | -360 ~ 360 사이의 수치 |  |
| sort | 파이 trace의 섹터 정렬 설정 | 논리값 |  |

##### textinfo

앞의 원 그래프에서 원 안에 표현된 데이터는 values에 매핑된 데이터의 백분율이 표시되었다. 이 데이터의 표시를 설정하는 속성이 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 = ~졸업자수, labels = ~대계열,   
 ## textinfo를 value로 설정  
 textinfo = 'value') |>   
 layout(title = list(text = '대학 계열별 취업률 분포'),   
 margin = margins)  
  
p\_pie |>  
 add\_trace(type = 'pie', values = ~졸업자수, labels = ~대계열,   
 ## textinfo를 percent로 설정  
 textinfo = 'percent') |>   
 layout(title = list(text = '대학 계열별 취업률 분포'),   
 margin = margins)  
  
p\_pie |>  
 add\_trace(type = 'pie', values = ~졸업자수, labels = ~대계열,   
 ## textinfo를 label+percent로 설정  
 textinfo = 'label+percent') |>   
 layout(title = list(text = '대학 계열별 취업률 분포'),   
 margin = margins)  
  
p\_pie |>  
 add\_trace(type = 'pie', values = ~졸업자수, labels = ~대계열,   
 ## textinfo를 label+value로 설정  
 textinfo = 'label+value') |>   
 layout(title = list(text = '대학 계열별 취업률 분포'),   
 margin = margins)

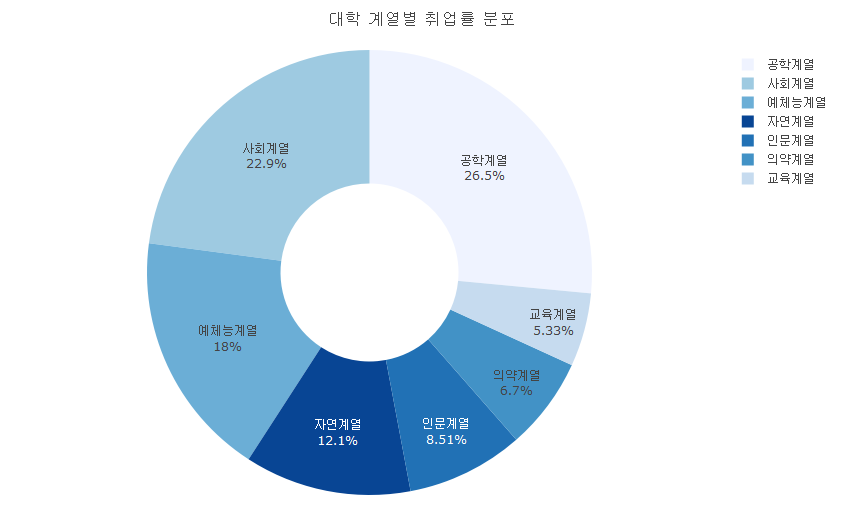


실행결과 2- . textinfo 설정에 따른 결과

##### hole

hole은 원 그래프의 내부에 표시되는 구멍의 크기를 설정하는 속성이다. 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 = ~졸업자수, labels = ~대계열,   
 textinfo = 'label+percent',   
 ## hole을 0.4로 설정  
 hole = 0.4) |>   
 layout(title = list(text = '대학 계열별 취업률 분포'),   
 margin = margins)

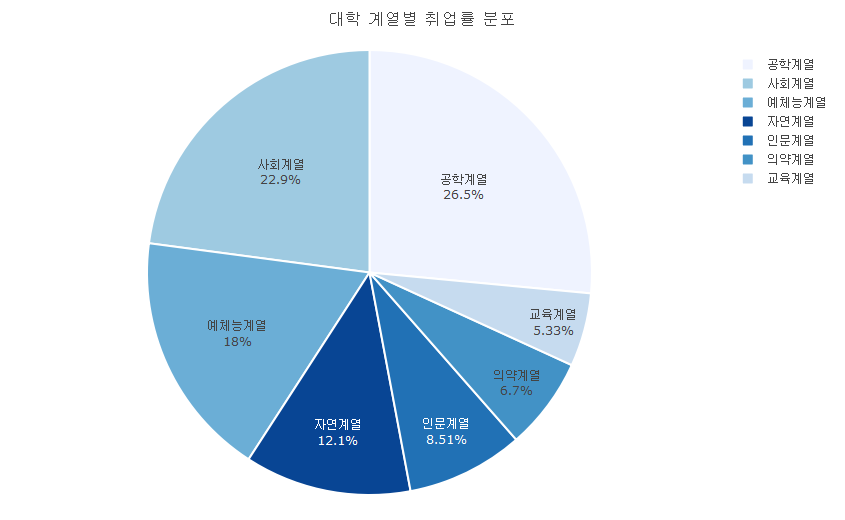


실행결과 2- . hole을 사용한 도넛 차트

##### marker

marker는 파이 trace로 표현되는 원의 형태를 설정하는 속성이다. 이 속성은 색(color)와 선(line)을 설정할 수 있고 선은 다시 세부 속성을 가지는 리스트로 구성한다.

p\_pie |>  
 add\_trace(type = 'pie', values = ~졸업자수, labels = ~대계열,   
 textinfo = 'label+percent',   
 ## marker 세부 속성 설정  
 marker = list(colors = RColorBrewer::brewer.pal(7, 'Blues'),   
 line = list(color = 'white',   
 width = 2)  
 )  
 ) |>   
 layout(title = list(text = '대학 계열별 취업률 분포'),   
 margin = margins)



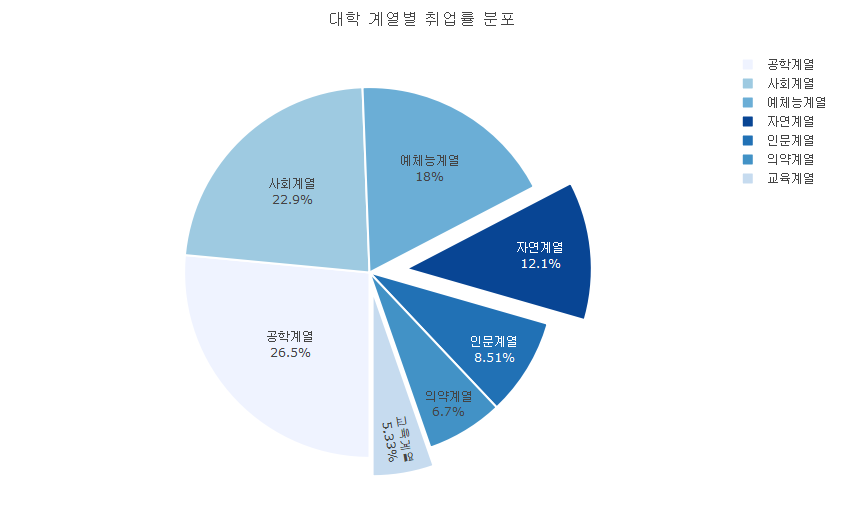
실행결과 2- . marker가 설정된 파이 trace

##### pull, rotation, sort

원 그래프를 그릴때 특정 섹터를 강조하기 위해 살짝 밖으로 빼서 그리는 경우가 있다. 이러한 효과를 위해 사용하는 속성이 pull이다. pull은 0부터 1사이의 수치값을 설정하거나 수치값 벡터를 설정하는데 빠져나올 섹터의 반지름을 설정한다.

또 원 그래프를 그릴때 처음 섹터를 시작하는 각도 지정을 위한 속성이 rotation이고 섹터의 위치를 설정할 때 값의 정렬 순서대로 표시하는 속성이 sort이다. sort는 기본값이 TRUE이기 때문에 특별한 설정이 없다면 데이터 값의 순서대로 표시되며 반시계 방향으로 표시된다. 이를 시계방향으로 설정하는 속성이 direction이다.

p\_pie |>  
 add\_trace(type = 'pie', values = ~졸업자수, labels = ~대계열,   
 textinfo = 'label+percent',   
 marker = list(line = list(color = 'white', width = 2)),   
 ## pull의 설정  
 pull = c(0,0.1,0,0,0,0,0.2),   
 ## rotation, sort, direction의 설정  
 rotation = 180, sort = TRUE, direction = 'clockwise') |>   
 layout(title = list(text = '대학 계열별 취업률 분포'),   
 margin = margins)

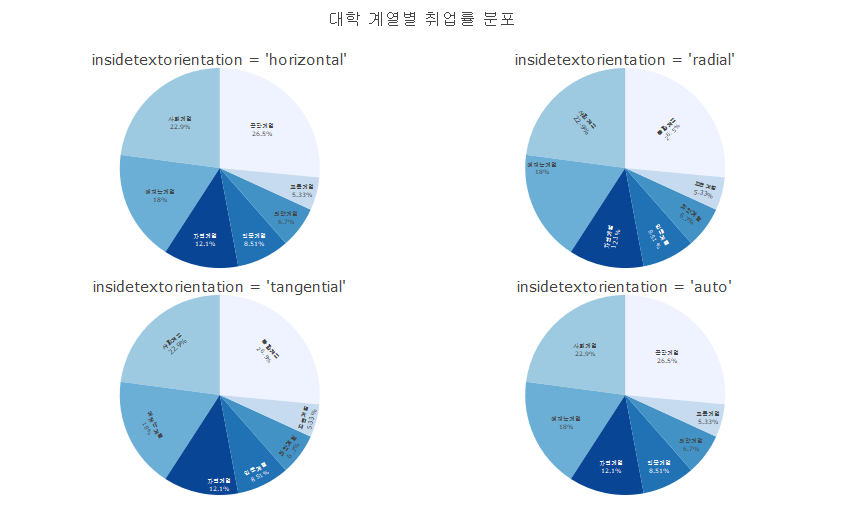


실행결과 2- . 섹터가 강조된 파이 trace

##### insidetextfont, outsidetextfont, insidetextorientation, outsidetextorientation

파이 trace로 그려지는 원 그래프는 원에 나누어지는 섹터의 크기에 따라서 표시되는 문자열이 섹터 안에 표시될 수도 있고 섹터 밖에 표시될 수도 있다. 이는 plotly에서 자동적으로 계산하여 표시하는데 내부에 표시되는 문자열의 형태를 설정하는 속성이 insidetextfont이고 밖에 표시되는 문자열의 형태를 설정하는 속성이 outsidetextfont이다. 이들 속성은 모두 color, family, size의 세가지 세부 속성을 사용하여 설정한다. 또 이들 문자열이 표시되는 방향을 설정하는 속성이 insidetextorientation과 outsidetextorientation이다. 이 속성 값은 문자열이 표시되는 방향을 가로 형태로 설정하는 ‘horizontal’, 섹터의 반경을 따라 표시되는 ‘radial’, 섹터 반경의 수직으로 표시되는 ‘tangential’, plotly가 자동으로 설정하는 ’auto’의 네 가지이다.

p\_pie |>  
 add\_trace(type = 'pie', values = ~졸업자수, labels = ~대계열,   
 textinfo = 'label+percent',   
 ## insidetextorientation을 horizontal로 설정  
 insidetextorientation = 'horizontal') |>   
 layout(title = list(text = '대학 계열별 취업률 분포'),   
 margin = margins)  
  
p\_pie |>  
 add\_trace(type = 'pie', values = ~졸업자수, labels = ~대계열,   
 textinfo = 'label+percent',   
 ## insidetextorientation의 radial로 설정  
 insidetextorientation = 'radial') |>   
 layout(title = list(text = '대학 계열별 취업률 분포'),   
 margin = margins)  
  
p\_pie |>  
 add\_trace(type = 'pie', values = ~졸업자수, labels = ~대계열,   
 textinfo = 'label+percent',   
 ## insidetextorientation의 tangential로 설정  
 insidetextorientation = 'tangential') |>   
 layout(title = list(text = '대학 계열별 취업률 분포'),   
 margin = margins)  
  
p\_pie |>  
 add\_trace(type = 'pie', values = ~졸업자수, labels = ~대계열,   
 textinfo = 'label+percent',   
 ## insidetextorientation의 auto로 설정  
 insidetextorientation = 'auto') |>   
 layout(title = list(text = '대학 계열별 취업률 분포'),   
 margin = margins)



실행결과 2- . insidetextorientation 설정 결과

#### 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 속성을 ’indicator’로 설정함으로써 사용할 수 있다.

인디케이터 trace에서 사용하는 주요 속성은 다음과 같다.

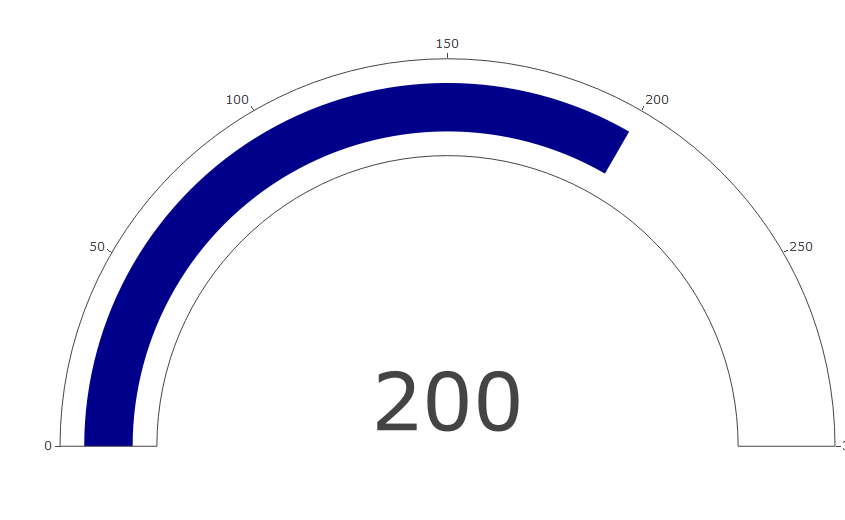
| 속성 | 설명 | 속성 값 | 세부 속성 |
| --- | --- | --- | --- |
| type | trace type 속성 | ‘indicator’ |  |
| visible | trace의 표시 설정 | TRUE, FALSE, ‘legendonly’ |  |
| title | 인디케이터 trace의 제목 설정 | 세부 속성들의 리스트 | font(color, family, size), position, text |
| align | 인디케이터 박스 안의 문자열 정렬 설정 | ‘left’, ‘center’, ‘right’ |  |
| delta | 증감량을 표시하는 인디케티터 설정 | 세부 속성들의 리스트 | decreasing(color, symbol), font(color, family, size), increasing(color, symbol), position, reference, relative, valueformat |
| number | 수치를 표시하는 인디케이터 설정 | 세부 속성들의 리스트 | font(color, family, size), prefix, surfix, valueformat |
| gauge | 게이지를 표시하는 인디케이터 설정 | 세부 속성들의 리스트 | axis(tickcolor, tickfont, ticks, …), bar(color, line(color, width), thickness), threshhold(line(color, width), thickness, value), bgcolor, shape, steps, … |

##### mode, value

인디케이터 trace에서 제공하는 표시 방법은 수치 표시, 증감량 표시, 게이지 표시의 세가지 방법을 제공한다. 이들을 설정하는 속성이 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 설정하고 단일 수치를 설정한다.

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

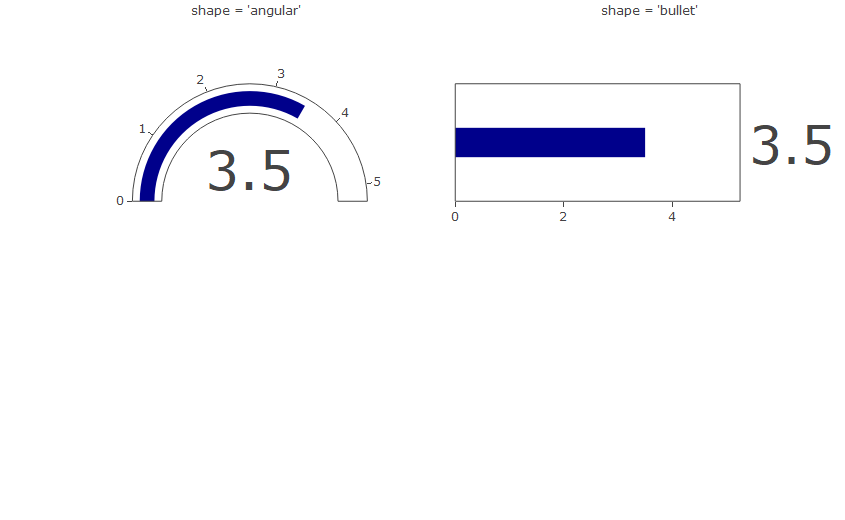


실행결과 2- . 인디케이터 trace 생성

##### gauge

gauge는 게이지 형태의 시각화를 통해서 단일 수치를 표현한다. 게이지의 형태는 둥근 반원 형태의 게이지를 사용하는 ’angler’와 막대 형태의 게이지를 사용하는 ’bullet’의 두가지가 제공된다. 이는 shape 속성을 사용하여 설정이 가능하다.

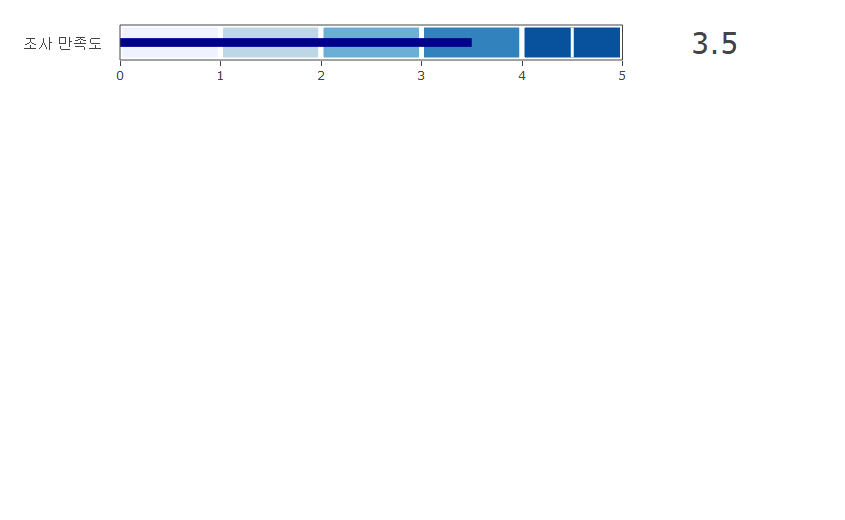
plot\_ly() |>  
 add\_trace(type = 'indicator', mode = "gauge+number",   
 ## gauge의 shape를 angular로 설정  
 gauge = list(shape = 'angular',   
 bar = list(color = 'darkblue')),   
 value = 3.5)  
  
plot\_ly() |>  
 add\_trace(type = 'indicator', mode = "gauge+number",   
 ## gauge의 shape를 angular로 설정  
 gauge = list(shape = 'bullet',   
 bar = list(color = 'darkblue')),   
 value = 3.5)



실행결과 2- . angula와 bullet 인디케이터 trace

인디케이터 trace의 게이지 모드에서 하나 중요하게 사용되는 속성이 steps 속성이다. steps 속성은 전체 게이지를 단계별로 구간을 설정하는 데 사용되는 속성이다. steps는 range, color, line 등의 세부 속성으로 구성된 리스트로 설정할 수 있다. 각 단계는 range로 시작점과 끝점이 포함된 벡터로 구분되며 이 구간들의 color, line 등의 속성을 설정할 수 있다.

step\_colors <- RColorBrewer::brewer.pal(5, 'Blues')  
  
plot\_ly() |>  
 add\_trace(type = 'indicator', mode = "gauge+number",  
 ## 인디케이터 제목 설정  
 title = list(text = '조사 만족도',   
 font = list(size = 15),   
 align = 'right'),   
 ## gauge의 세부 속성 설정  
 gauge = list(shape = 'bullet',   
 bar = list(color = 'darkblue'),   
 ## 전체 범위는 0~5로 설정  
 axis = list(range = c(0, 5)),  
 ## 각 단계별로 range, color, line의 속성 설정  
 steps = list(  
 list(range = c(0,1), color = step\_colors[1],   
 line = list(color = 'white', width = 5)),  
 list(range = c(1,2), color = step\_colors[2],   
 line = list(color = 'white', width = 5)),  
 list(range = c(2,3), color = step\_colors[3],   
 line = list(color = 'white', width = 5)),  
 list(range = c(3,4), color = step\_colors[4],   
 line = list(color = 'white', width = 5)),  
 list(range = c(4,5), color = step\_colors[5],   
 line = list(color = 'white', width = 5))  
 ),  
 ## threshhold(임계치) 세부 속성 설정  
 threshold = list(line = list(color = 'white', width = 3),  
 value = 4.5)  
 ),   
 value = 3.5) |>  
 ## 가로로 길게 시각화 크기 설정  
 layout(height = 100, width = 800, margin = list(l= 120, r= 10))

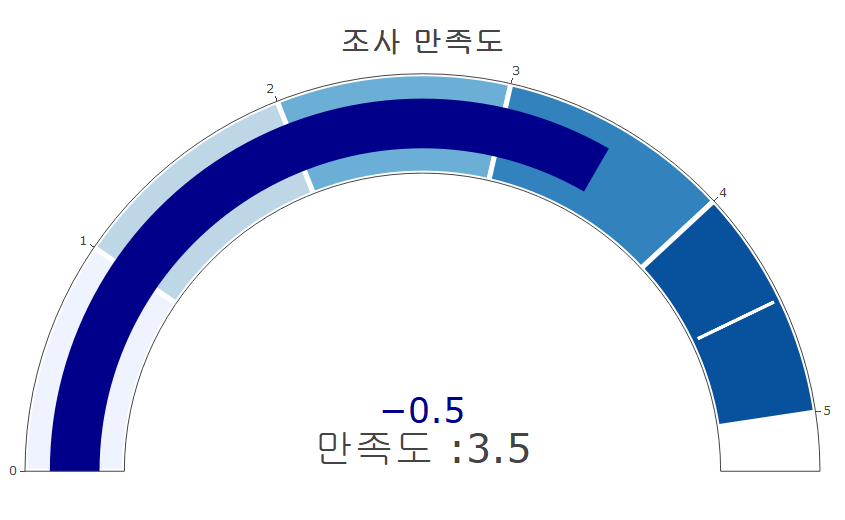


실행결과 2- . 인디케이터 trace의 step 설정

##### delta

delta는 데이터의 증감량을 나타내는 인디케이터를 표시한다. 가장 흔히 보이는 곳이 주식관련 시각화인데 전일 대비 주가를 파란색, 빨간색 삼각형과 같이 나타내는 것이 delta이다. 증감량을 표시해야 하기 때문에 증감의 기준이 되는 수치인 reference가 설정되어야 하고 increasing과 decreasing을 통해 증감의 표시를 설정할 수 있다.

plot\_ly() |>  
 add\_trace(type = 'indicator', mode = "gauge+delta+number",   
 ## 인디케이터의 제목 설정  
 title = list(text = '<b>조사 만족도</b>', font = list(size = 30)),   
 ## angular 형태의 gauge 세부 속성 설정   
 gauge = list(shape = 'angular',  
 bar = list(color = 'darkblue'),   
 ## steps 세부 속성 설정  
 steps = list(  
 list(range = c(0,1), color = step\_colors[1],   
 line = list(color = 'white', width = 5)),  
 list(range = c(1,2), color = step\_colors[2],   
 line = list(color = 'white', width = 5)),  
 list(range = c(2,3), color = step\_colors[3],   
 line = list(color = 'white', width = 5)),  
 list(range = c(3,4), color = step\_colors[4],   
 line = list(color = 'white', width = 5)),  
 list(range = c(4,5), color = step\_colors[5],   
 line = list(color = 'white', width = 5))  
 ),  
 ## threshold 세부 속성 설정  
 threshold = list(line = list(color = 'white',   
 width = 3),  
 value = 4.5)  
 ),   
 value = 3.5,   
 ## delta의 세부 속성 설정  
 delta = list(reference = 4, position = "top",   
 ## 증가치 표현을 위한 increasing 세부 속성 설정  
 increasing = list(color = 'darkred', symbol = '+'),   
 ## 감소치 표현을 위한 decreasing 세부 속성 설정  
 decreasing = list(color = 'darkblue', symbol = ''),   
 font = list(size = 35)  
 ),   
 ## 수치 표현을 위한 number 세부 속성 설정  
 number = list(prefix = '만족도 :', font = list(size = 40))  
 ) |> layout(margin = margins)



실행결과 2- . steps가 설정된 angular 형태의 인디케이터 trace

1. <https://github.com/owid/covid-19-data/blob/master/public/data/owid-covid-data.csv> [↑](#footnote-ref-20)
2. 필자의 블로그에 업로드된 데이터를 활용하면 책과 동일한 결과를 얻을 수 있다. [↑](#footnote-ref-22)
3. 해당 데이터는 교육통계 서비스 홈페이지<https://kess.kedi.re.kr/contents/dataset?itemCode=04&menuId=m_02_04_03_02&tabId=m3>에서 다운로드를 받거나 필자의 블로그(2stndard.tistory.com)에서 다운로드 받을 수 있다. [↑](#footnote-ref-24)
4. <https://github.com/d3/d3-format/tree/v1.4.5#d3-format> [↑](#footnote-ref-28)
5. <https://github.com/d3/d3-format/tree/v1.4.5#d3-format> [↑](#footnote-ref-38)
6. <https://plotly.com/r/reference/scatter/> [↑](#footnote-ref-53)
7. <https://plotly.com/r/reference/#scatter-marker-colorbar> [↑](#footnote-ref-56)
8. boxmode를 사용할 때 ’Warning message’를 내는 경우가 있는데 이는 Plotly Community Forum에서도 적절치 않은 경고 메세지로 지적되고 있는데 무시해도 되는 메세지이다. [↑](#footnote-ref-65)