

### III. 데이터 시각화를 위한 `plotly`

---

R 을 다루는 많은 교육코스나 서적에서 데이터의 시각화는 대부분 R base 에서 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.

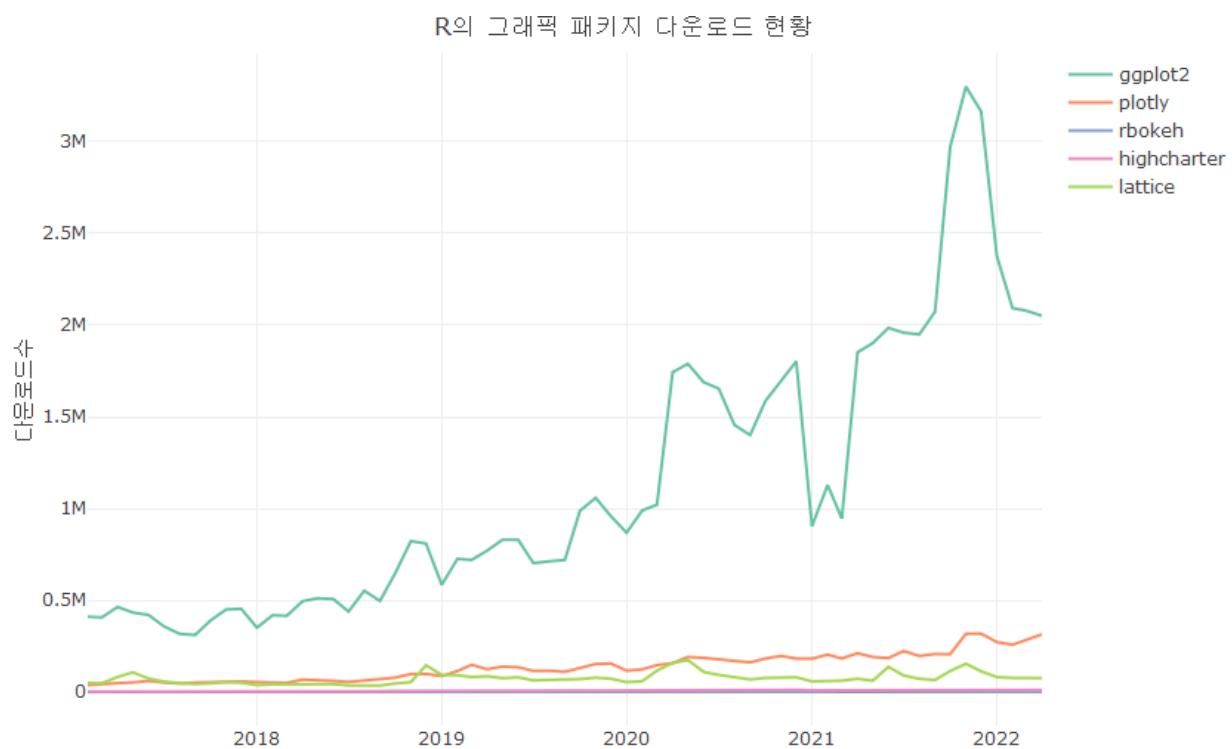
이러한 제한점을 극복하기 위해 사용되는 데이터 시각화 방법이 동적(Dynamic) 시각화 혹은 인터랙티브(Interactive) 시각화라고 하는 방법이다. 이 동적 시각화는 시각화를 사용하는 사용자의 의도에 따라 데이터를 다각적 관점에서 살펴볼 수 있다는 점이 동적 시각화와 가장 크게 차이나는 점이다. 사용자의 의도에 따라 데이터가 동적으로 변동되어야 하기 때문에 데이터 시각화에 사용되는 매체는 인쇄물 형태 매체가 불가능하고 웹을 통해 제공한다. 따라서 일반적으로 동적 시각화는 웹 사이트에서 제공하는 대시보드(DashBoard)의 형태로 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. R 에서 동적 시각화를 위해 제공되는 패키지는 `plotly`, `rbokeh`, `highcharter` 등이 제공된다. 다음의 R 그래픽 패키지 다운로드 현황에서 보듯이 여전히 R 에서 가장 많이 사용되고 있는 그래픽 패키지는 `ggplot2` 이지만 최근 들어 그 사용량이 줄고 있고 R 에서 많이 사용되던 `lattice` 패키지가 점차 줄고 있다. 반면 `plotly` 는 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'))

x1 |> plot_ly() |>
  add_lines(x = ~end, y = ~downloads, color = ~package) |>
```

```
layout(title = 'R 의 그래픽 패키지 다운로드 현황', xaxis = list(title = list(visible = F)), yaxis = list(title = '다운로드수'))
```



### 실행결과 3- 1. R 의 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.

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

코로나바이러스 데이터 셋은 Github 에서 매일 업데이트된 csv 파일을 다운로드 받았다.<sup>1</sup> 이를 **plotly** 실습에 사용하기 위해 다음과 같이 전처리 하였다.

<sup>1</sup> <https://github.com/owid/covid-19-data/blob/master/public/data/owid-covid-data.csv>

```

## 데이터 전처리를 위한 패키지 설치 및 로딩
if(!require(readr)) {
  install.packages('readr')
  library(readr)
}

if(!require(lubridate)) {
  install.packages('lubridate')
  library(lubridate)
}

## 코로나 관련 데이터 로딩
covid19_df <- read_csv(file = "D:/R/data/owid-covid-data.csv",
                        col_types = cols(Date = col_date(format = "%Y-%m-%d"
"))
)
)

## 코로나 관련 온라인 데이터 로딩
covid19_df <- read_csv(file = "https://covid.ourworldindata.org/data/owid-covid-
data.csv",
                        col_types = cols(Date = col_date(format = "%Y-%m-%d"
"))
)
)

covid19_df_100 <- covid19_df |>
  ## 한국 데이터와 각 대륙별 데이터만을 필터링
  filter((iso_code %in% c('KOR', 'OWID_ASIA', 'OWID_EUR', 'OWID_OCE', 'OWID_NAM',
  'OWID_SAM', 'OWID_AFR'))) |>
  ## 읽은 데이터의 마지막 데이터에서 100 일전 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')) |>
  ## 날짜로 정렬
  arrange(date)

```

```

## 국가 이름의 순서를 설정
covid19_df_100$location <- fct_relevel(covid19_df_100$location, '한국', '아시아',
                                         '유럽', '북미', '남미', '아프리카', '오세아니아')

## 한국과 각 대륙별 100 일간 데이터에서
covid19_df_100_wide <- covid19_df_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)

## 코로나19 데이터 중 통계치를 가지는 데이터 추출
covid19_stat <- covid19_df |> 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),
            십만명당사망자수 = round(total_deaths / population *100000, 5),
            십만명당중환자실 = last(icu_patients_per_million),
            재생산지수 = last(reproduction_rate),
            전체검사자수 = max(total_tests, na.rm = T), new_tests = 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),
            백신접종완료률 = people_fully_vaccinated / population,
            인구백명당백신접종완료률 = max(people_fully_vaccinated_per_hundred, na.rm = T),
            인구백명당부스터접종자수 = max(total_boosters_per_hundred, na.rm = T)
  ) |> ungroup()

## 여백 설정을 위한 리스트 설정
margins <- list(t = 50, b = 25, l = 25, r = 25)

```

## 1. **plotly** 란?

**plotly** 는 오픈 소스인 JavaScript 로 구현된 plotly.js 를 기반으로 R 에서 생성한 데이터 시각화 객체를 Javascript 로 생성해주는 패키지이다<sup>2</sup>. **plotly** 는 JavaScript 로 구현되기 때문에 **plotly** 객체는 결국 HTML 코드로 구현되고 이 코드는 웹브라우저 상에서 작동함으로써 사용자의 반응에 따른 데이터의 표현이 가능하다.

**plotly** 를 통해 생성된 데이터 시각화의 HTML 은 R 에서 JavaScript 를 사용할 수 있게하는 **htmlwidgets** 프레임워크에서 동작하기 할 수 있어 HTML 자체로 사용할 수도 있고 R Markdown 이나 Shiny App, R-Studio, Jupiter Notebook 등에서 자유롭게 사용이 가능하다.

R 의 **plotly** 패키지는 동적 시각화를 제공하면서 출판이 가능한 품질의 시각화를 만든다. **plotly** 로 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 **ggplot2** 보다 많다. **plotly** 는 R 에서 무료로 사용 가능한 패키지로 python, julia, F#, MATLAB 등에서도 지원되는 그래픽 라이브러리를 제공하고 있다.

### 1.1. **plotly** 의 내부 표현 구조

R 내부에서 **plotly** 객체는 plotly.js 에서 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 20 으로 설정하게 된다. 이 속성값은 **plotly** 의 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 = 20` 으로 설정할 수 있다. **plotly** 객체를 생성하기 위해서는 많은 노드의 속성 값이 필요하지만 사용자가 직접 설정하지 않은 속성 값은 기본값으로 설정되거나 **plotly** 가 내부적으로 계산하여 속성 값들이 설정되게 된다.

R 에서 **plotly** 객체를 표현하는데 사용되는 트리의 첫번째 레벨 노드는 'data', 'layout', 'frame'의 세 가지가 있다.

---

<sup>2</sup> <https://plotly.com>

## 1.2. ‘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.

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

각각의 `trace` 는 그 `trace` 도형 타입과 관련된 하나의 `subplot` 에 표현되고, 파이 `trace` 를 제외한 모든 `trace` 는 단일 범례 요소(Single Legend Entry)를 가진다. 하지만 `trace` 의 종류에 따라 설정 가능한 속성이 달라지기 때문에 각각의 `trace` 마다 설정할 수 있는 속성들을 잘 파악할 필요가 있다.

## 1.3. ‘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.

## 1.4. ‘frame’ 노드

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

# 2. `plotly` 시각화 만들기

R 에서 `plotly` 를 사용한 인터랙티브 데이터 시각화를 만드는 방법은 두 가지이다. 첫 번째는 R에서 데이터 시각화에 가장 많이 사용하는 `ggplot2` 를 사용하여 생성한 객체를 `plotly` 객체로 전환하는 방법이고 두 번째는 `plotly` 패키지에서 제공하는 함수들을 사용하여 `plotly` 객체를 직접 생성하는 방법이다.

## 2.1. ggplot 객체의 전환

기존의 R 사용자가 가장 쉽게 `plotly` 를 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` : `plotly` 로 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.

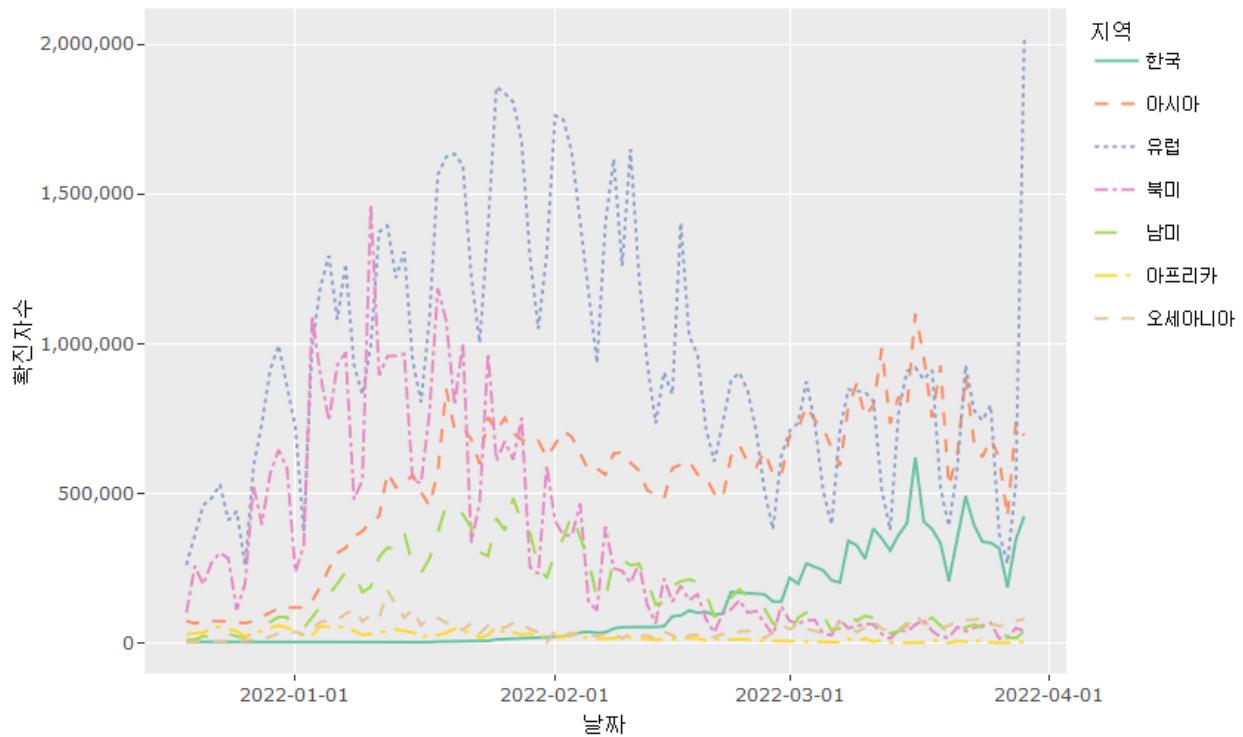
```
## plotly 패키지 설치 및 로딩  
if(!require('plotly')) {  
  install.packages('plotly')  
  library(plotly)  
}  
  
ggplotly <- covid19_df_100 |>  
  ## x 축이 date, y 축이 new_cases, color 가 location 으로 매핑된 ggplot 객체 생성  
  ggplot(aes(x = date, y = new_cases, color = location )) +  
  ## group 과 linetype 이 location 으로 매핑된 geom_line 레이어 생성  
  geom_line(aes(group = location, linetype = location)) +  
  ## x 축 스케일을 날짜형이고 1 개월 단위로 눈금 설정  
  scale_x_date(breaks = '1 months') +  
  ## y 축의 라벨에 천 단위 구분자 설정  
  scale_y_continuous(labels = scales::comma) +
```

```

## 색 설정을 RColorBrewer 패키지의 'Set2'로 설정
scale_color_brewer(palette = 'Set2') +
## x 축, y 축, linetype, color 이름 설정
labs(x = '날짜', y = '확진자수', linetype = '지역', color = '지역')

## ggplot 객체를 plotly 객체로 변환
ggplotly(ggplotly)

```



### 실행결과 3- 2. ggplot2 객체의 전환

앞의 예와 같이 `ggplotly()`는 `ggplot2` 패키지를 사용해서 생성한 대부분의 `ggplot` 객체를 전환할 수 있다. 게다가 `ggplot2` 패키지를 확장해서 사용하게 해주는 `ggforce`, `GGally` 와 같은 확장 패키지로 생성된 객체도 변환이 가능하다는 장점이 있다. 따라서 `plotly` 객체를 다루는 방법을 잘 익히면 기존에 생성했던 `ggplot` 객체의 시각화를 재활용 할 수 있게 된다. 그리고 `plotly` 를 사용하다보면 느끼겠지만 `ggplot2` 의 통계 요소와 분할 요소의 몇몇 요소들은 `plotly` 보다는 `ggplot2` 가 훨씬 편리할 때가 있다. 이러한 경우 `ggplot2` 와 `plotly` 를 적절히 혼용하면 매우 좋은 결과를 얻을 수 있다.

## 2.2. plotly 시각화 생성

plotly 객체는 plotly.js 를 지원하는 스키마로 표현된다. 사실 ggplot2 이던 plotly 이던 각각의 시각화 객체는 R 에서 특별하게 정의된 데이터 구조로 표현된다. 이 데이터 구조가 R 의 그래픽 엔진을 통해 이미지로 표현되는 것이다. 따라서 plotly 객체도 우리가 눈으로 보기에는 이미지로 보이지만 R 에서는 plotly.js 에서 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.

### 2.2.1. plotly 객체 초기화 : plot\_ly()

설명한 바와 같이 plotly 는 보여지기에 그래픽으로 보여지지만 내부적으로는 plotly.js 에서 정의한 데이터 구조 형태로 저장된다. 따라서 plotly 객체를 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 : plotly 로 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.org 의 팔레트 이름이나 16 진수의 '#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.org 의 팔레트 이름이나 16 진수의
```

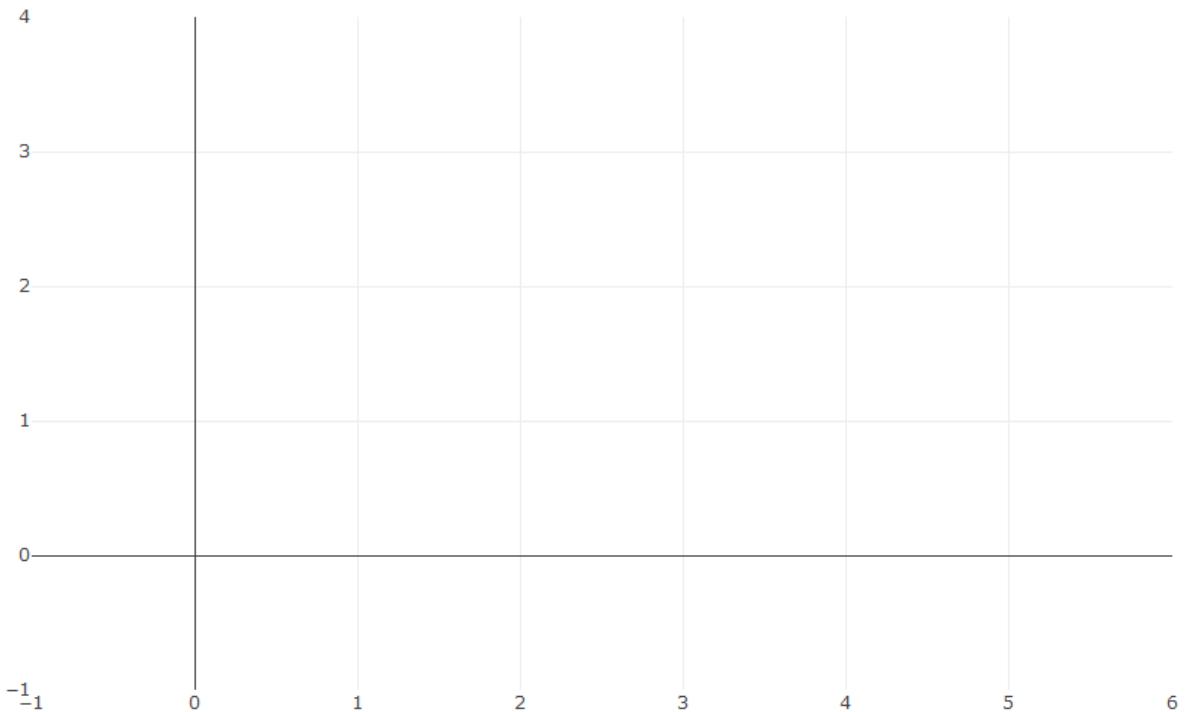
'#RRGGBB' 형태로 표현된 색 벡터(vector) 설정 / - alpha-stroke : stroke(외곽선)에 적용될 alpha 값(value) 설정

- size : 'fill-size'에 매핑될 크기값(value) 설정
- sizes : size 에 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 traces 를 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, trace 와 속성을 통한 시각화 생성, 하위 trace 에 공통적으로 사용될 속성 값의 설정의 세가지 기능을 수행한다.

첫 번째 기능은 `plot_ly()`의 호출을 통해 `plotly.js`에서 정의된 `plotly` 객체 스키마를 생성한다. 사용자가 직접 `plotly.js` 형태의 스키마 객체를 타이핑하여 생성하는 것은 어려움이 따르기 때문에 `plotly` 객체 스키마 생성, 기본 스키마 속성 설정 등을 지원하는 함수이다. 이 방법은 `ggplot` 객체를 생성하기 위해 `ggplot()`를 사용하여 `ggplot` 객체를 생성하는 것과 동일한 방법이고 이 방법에서 영감을 받았다고 한다.

```
## 긴 형태의 100 일간 코로나19 데이터를 사용한 plotly 객체 생성
covid19_df_100 |>
  plot_ly()
```



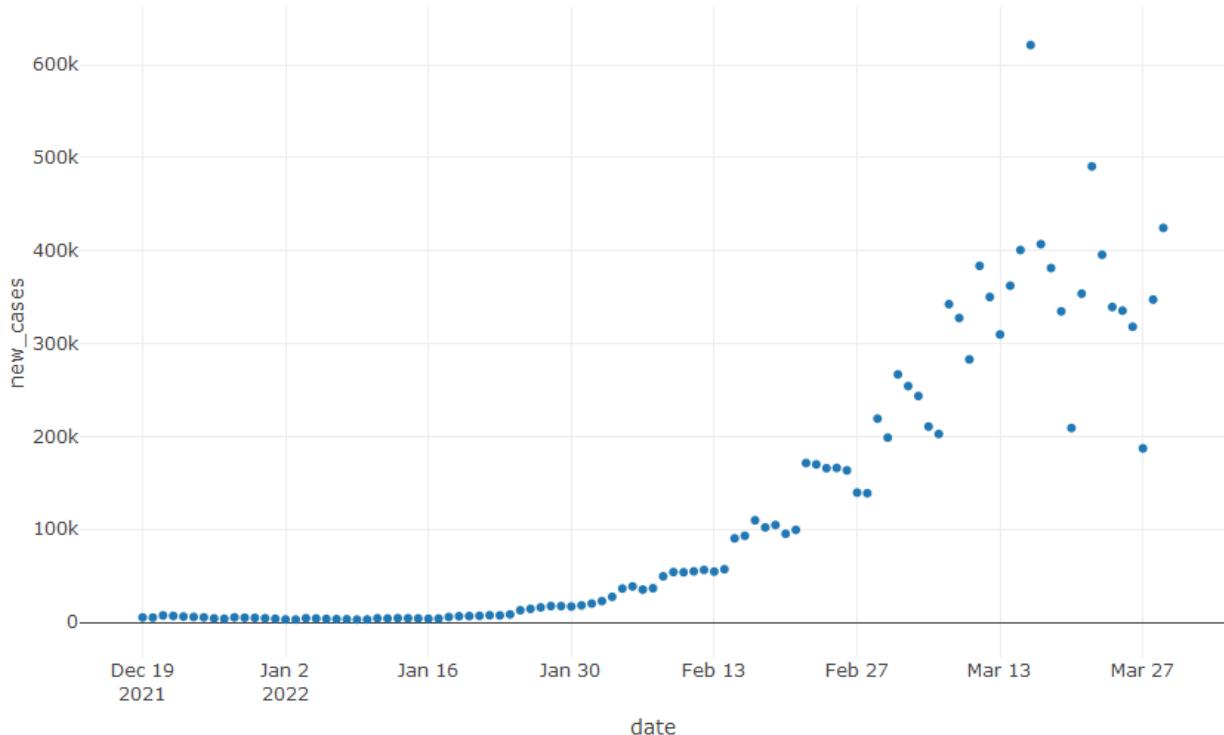
### 실행결과 3- 3. `plot_ly()`를 사용한 `plotly` 초기화

두 번째 기능은 `plotly` 객체 생성을 통한 시각화 생성이다. `ggplot2`에서 `ggplot()`을 사용하여 `ggplot` 객체를 초기화하고 `geom_*`()를 사용하여 기하 요소 레이어를 하나 하나 설정하면서 전체 시각화를 완성한다. `plotly` 도 이와 유사하게 초기 `plotly` 객체에 여러개의 trace 를 추가함으로써 전체 시각화를 완성해 나갈 수 있다. 하지만 `plotly()`에서는 데이터, trace, trace 에 따른 속성까지 설정이 가능하므로 `plot_ly()`만을 사용해도 시각화를 완성할 수 있다. 하지만 시각화에서는 하나 이상의 trace 가 사용될 때가 많기 때문에 뒤에서 설명할 `add_trace()`나 `add_*`()를 사용하여 trace 를 추가하여 시각화를 완성하는 것이 일반적이다.

`plotly` 를 사용한 시각화에서 속성에 변수나 값을 할당할 때 꼭 알아 두어야 하는 것은 변수를 매핑하는 방법과 값을 설정하는 방법이 다르다는 것이다. `ggplot2`에서는 변수를 매핑하기 위해서 `aes()`를 사용함으로써 변수를 매핑하였고 값을 설정하기 위해서는 `aes()` 밖에 선언함으로써 설정이 가능하였다. `plotly` 에서는 `aes()` 대신 `~`를 사용하여 변수를 매핑한다.

```
## 긴 형태의 100 일 코로나19 데이터에서
covid19_df_100 |>
  ## 한국 데이터만을 필터링
  filter(iso_code == 'KOR') |>
```

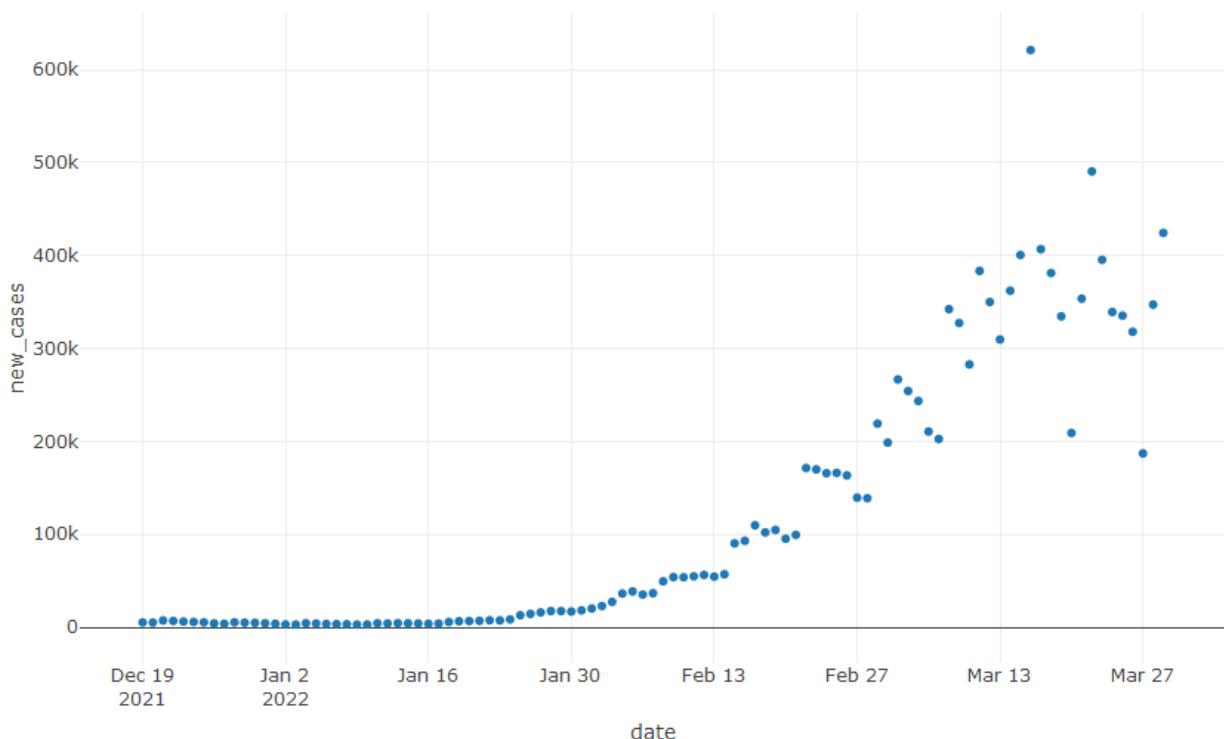
```
## X 축에 date, Y 축에 new_cases 를 매핑하여 plot_ly()로 시각화 생성  
plot_ly(x = ~date, y = ~new_cases)
```



#### 실행결과3- 4. plot\_ly 를 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 원치 않는 속성의 상속을 방지하기 위해 `plot_ly()`에 매개변수를 넣지 않고 단순히 `plotly` 객체의 초기화 명령으로 사용도 가능하다.

```
covid19_df_100 |>  
filter(iso_code == 'KOR') |>  
## plot_ly()로 시각화를 초기화  
plot_ly() |>  
## 스캐터 trace 타입으로 X 축에 date, Y 축에 new_cases 를 매핑한 시각화 생성  
add_trace(type = 'scatter', x = ~date, y = ~new_cases)
```



### 실행결과 3- 5. `add_trace()`를 사용한 시각화

`plotly` 객체에서 가장 중요한 것은 trace의 종류이다. `plot_ly()`만을 사용하여 그래프를 완성할 때 trace의 종류를 생략하면 `plotly`에서 데이터를 파악하여 가장 좋은 trace를 설정해준다.

하지만 `plotly` 사용하여 시각화를 만들 때는 trace type부터 시작하여 해당 trace의 세부 속성을 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.

`plotly`의 trace 설정에 공통적으로 사용되는 주요 속성은 다음과 같다.

#### 2.2.1.1. `x, y, z`

trace 설정에서 가장 기본적으로 설정하는 속성은 `x, y, z`이다. 이 세 개의 속성은 X 축, Y 축, Z 축에 매핑하는 변수를 설정한다. 앞서 설명한 바와 같이 `x, y, z`에 매핑하는 변수는 `~`를 사용하여 매핑하는데 변수를 매핑하지 않고 벡터를 설정하는 것도 가능하다.

```

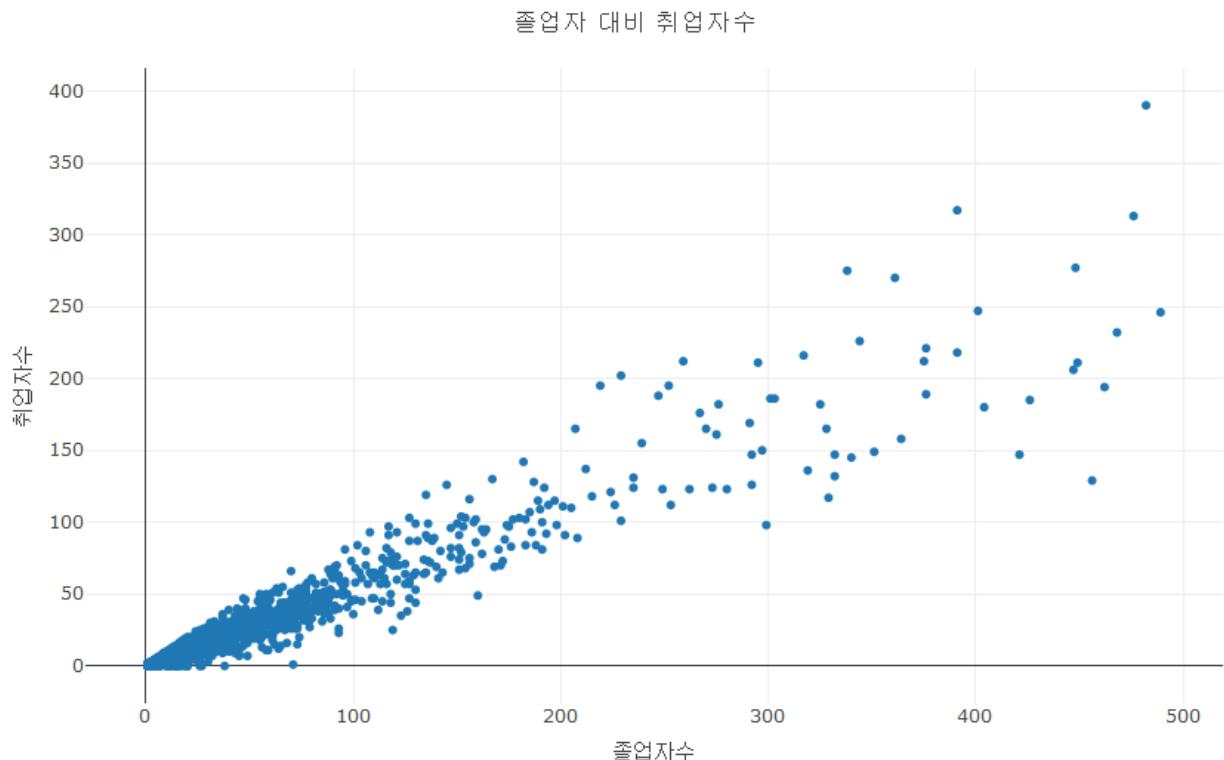
## 랜덤 샘플을 위한 시드 설정
set.seed(123)

## df_취업통계에서 졸업자가 500 명 이하인 학과 2000 개 샘플링
df_취업통계_2000 <- df_취업통계 |>
  filter(졸업자_계 < 500) |>
  sample_n(2000)

## 열 이름을 적절히 설정
names(df_취업통계_2000)[10:12] <- c('졸업자수', '취업률', '취업자수')

## df_취업통계_2000 에서
df_취업통계_2000 |>
  ## x 축은 졸업자수, y 축은 취업자수로 매팅한 plotly 객체 생성
  plot_ly(x = ~졸업자수, y = ~취업자수) |>
  ## 제목과 여백 설정
  layout(title = '졸업자 대비 취업자수', margin = margins)

```

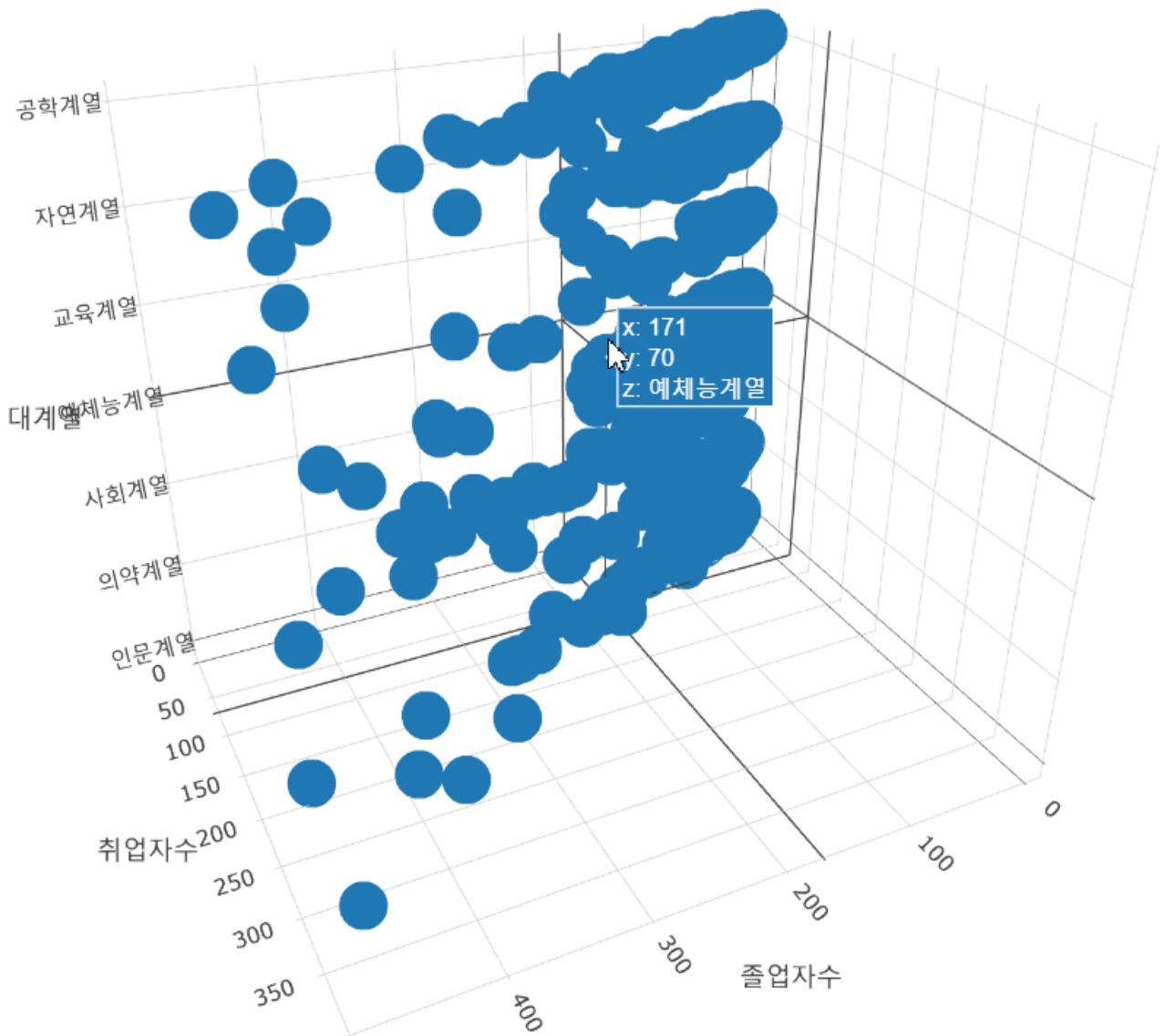


실행결과 3- 6. plotly 기본 산점도 생성

`plotly` 는 3 차원 그래픽을 지원하기 때문에 Z 축을 설정할 수 있다. 동적 시각화는 사용자가 시각화를 자신이 원하는 방향으로 설정하여 관찰할 수 있기 때문에 3 차원 효과가 효율적일 수 있다. 하지만 일반적으로 데이터 시각화에서는 3 차원을 사용하는 것은 크게 효과적이지 않다고 알려져 있기 때문에 3 차원의 활용은 주의할 필요가 있다.

```
df_취업통계_2000 |>
```

```
## x 축은 졸업자수, y 축은 취업자수, z 축은 대계열로 맵핑된 plotly 객체 생성  
plot_ly(x = ~졸업자수, y = ~취업자수, z = ~대계열)
```



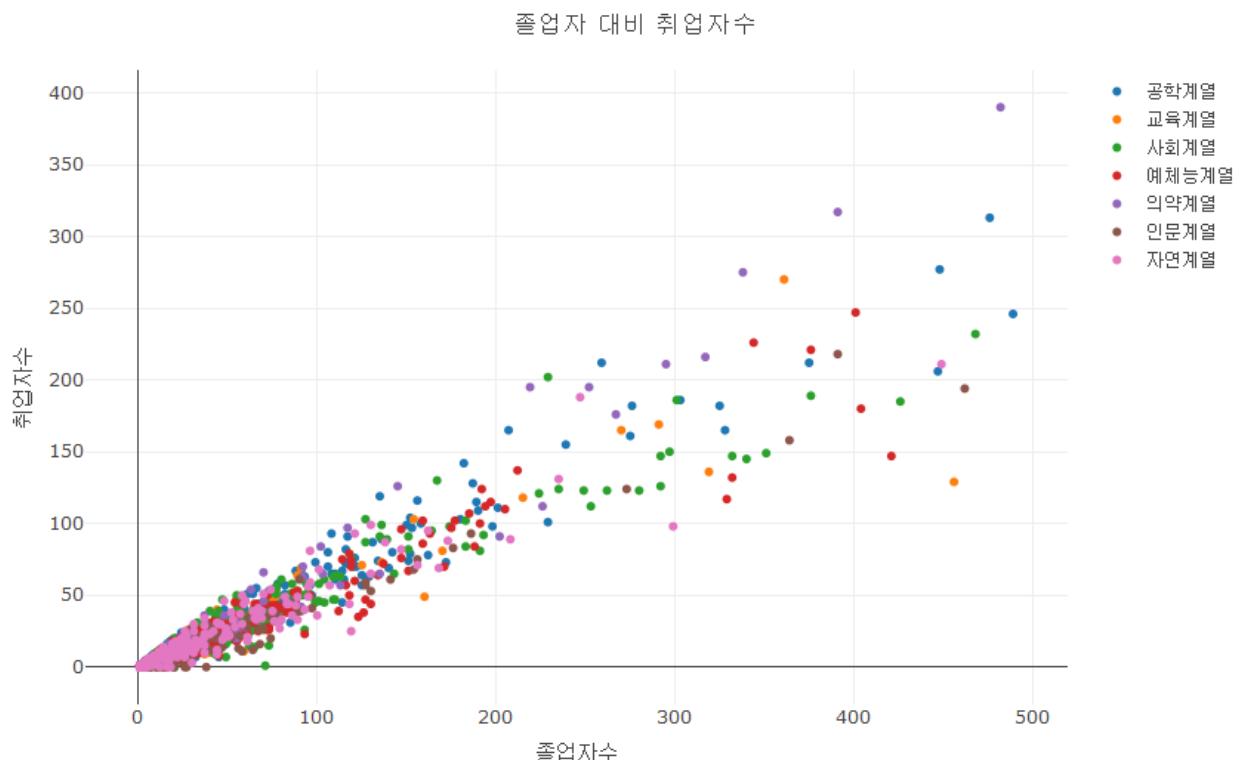
실행결과 3- 7.3 차원 `plotly`

### 2.2.1.2. name

`name` 은 `plotly` 에서 추가되는 각각의 trace 에 대한 이름을 설정한다. 이 이름은 범례 아이템과 마우스 포인터가 데이터 점에 위치할때 나타나는 호버(Hover)에 표기되는 이름이 된다.

다음은 `name` 에 범례에 표기되어야 할 변수를 매핑하여 범례 아이템 이름을 설정하는 코드이다.

```
df_취업통계_2000 |>  
## x 축은 졸업자수, y 축은 취업자수, name 은 대계열로 매핑한 plotly 객체 생성  
plot_ly(x = ~졸업자수, y = ~취업자수, name = ~대계열) |>  
layout(title = '졸업자 대비 취업자수', margin = margins)
```



실행결과 3- 8. trace 이름 매핑

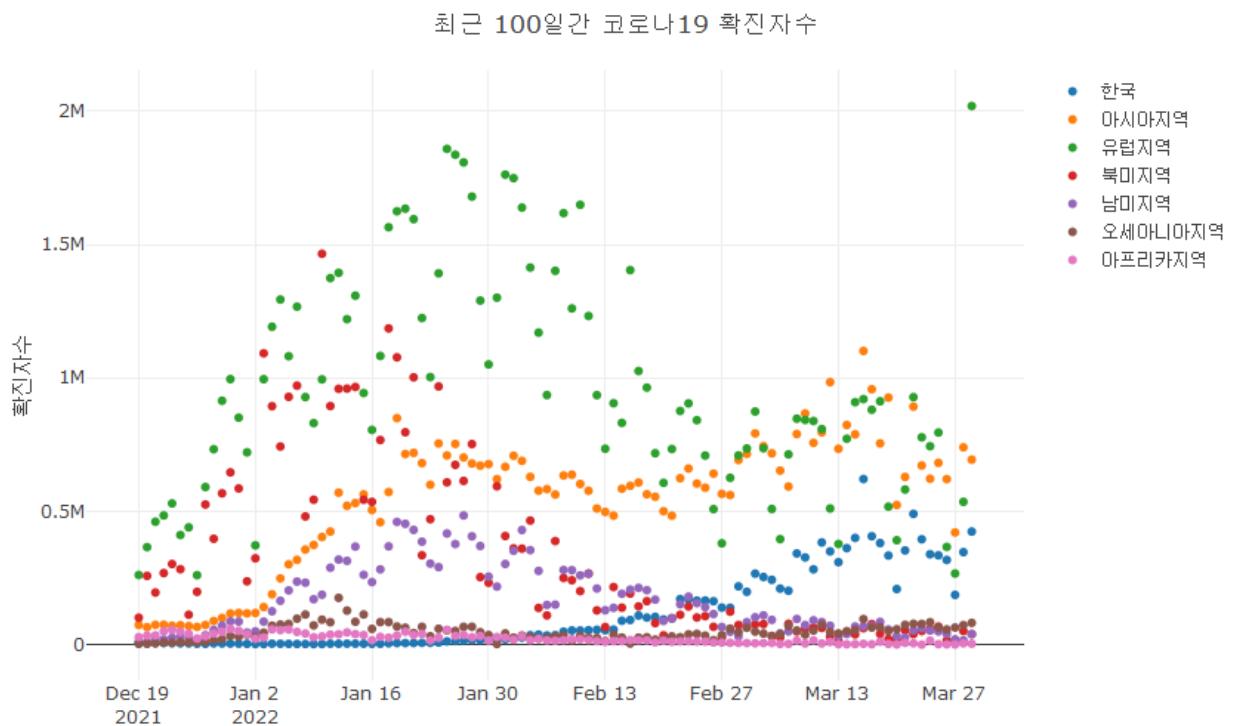
만약 범례를 사용자가 원하는 형태로 바꾸려면 범례 아이템을 바꾸는 방법으로 할 수 없고 각각의 개별 데이터에 매핑될 벡터나 열을 만들어 주어야 한다. `ggplot2` 와 같은 정적 시각화는 시각화가 만들어지면 사용자의 활동에 따라 반응이 없지만 동적 시각화는 마우스 포인터를 데이터에 가져가면 해당 데이터에 대한 내용이 표기되어야 한다. 이 값이 범례값과 달라지면

안되기 때문에 범례만 수정할 수 없고 모든 데이터에 1:1로 매핑되는 범례 아이템 이름 벡터가 필요하다.

```
## 범례로 사용할 문자열 벡터 생성
legend_items <- covid19_df_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')

covid19_df_100 |>
  ## X 축을 date, Y 축을 new_cases, 범례 이름을 앞서 생성한 문자열 벡터로 매핑
  plot_ly(x = ~date, y = ~new_cases, name = ~legend_items) |>
  layout(title = '최근 100 일간 코로나 19 확진자수',
    ## X 축 이름을 제거
    xaxis = list(title = list(text = '')),
    ## Y 축 이름을 설정
    yaxis = list(title = list(text = '확진자수')),
    ## 여백 설정
    margin = margins)
```



### 실행결과 3- 9. 범례 이름 변경

#### 2.2.1.3. `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. `plotly`의 많은 trace에서 데이터 값을 표현하기 위해서는 `text` 속성을 사용하면 간단히 표시할 수 있다. 표시되기를 원하는 변수를 `text`에 매핑함으로써 X, Y 축의 위치나 trace의 type에 적합한 위치에 데이터 값을 표시할 수 있으며 `textposition`과 `texttemplate`을 사용하여 표시되는 값의 세부 설정을 변경할 수 있다.

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

`textposition`은 `text`의 위치를 설정하는 속성이다. `textposition`에는 '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 `plotly`에서 자동적으로 계산된 형태로 텍스트가 표시된다. 'none'은 텍스트가 표시되지 않는다.

```
## 긴 형태의 100 일간 코로나19 데이터 중에
covid19_df_100 |>
  ## 국가명으로 그룹화
  group_by(location) |>
  ## 확진자수의 합계를 new_cases로 산출
  summarise(new_cases = sum(new_cases)) |>
  ## X 축을 location, Y 축과 text 를 new_case로 맵핑
  plot_ly(x = ~location, y = ~new_cases, text = ~new_cases,
          ## textposition 을 'inside'로 설정
          textposition = 'inside') |>
  layout(title = list(text = '지역별 코로나 19 확진자수'),
         xaxis = list(title = '지역'),
         yaxis = list(title = '확진자수'),
         margin = margins)

covid19_df_100 |>
  group_by(location) |>
  summarise(new_cases = sum(new_cases)) |>
  plot_ly(x = ~location, y = ~new_cases, text = ~new_cases,
          ## textposition 을 'outside'로 설정
          textposition = 'outside') |>
  layout(title = list(text = '지역별 코로나 19 확진자수'),
         xaxis = list(title = '지역'),
         yaxis = list(title = '확진자수'),
         margin = margins)

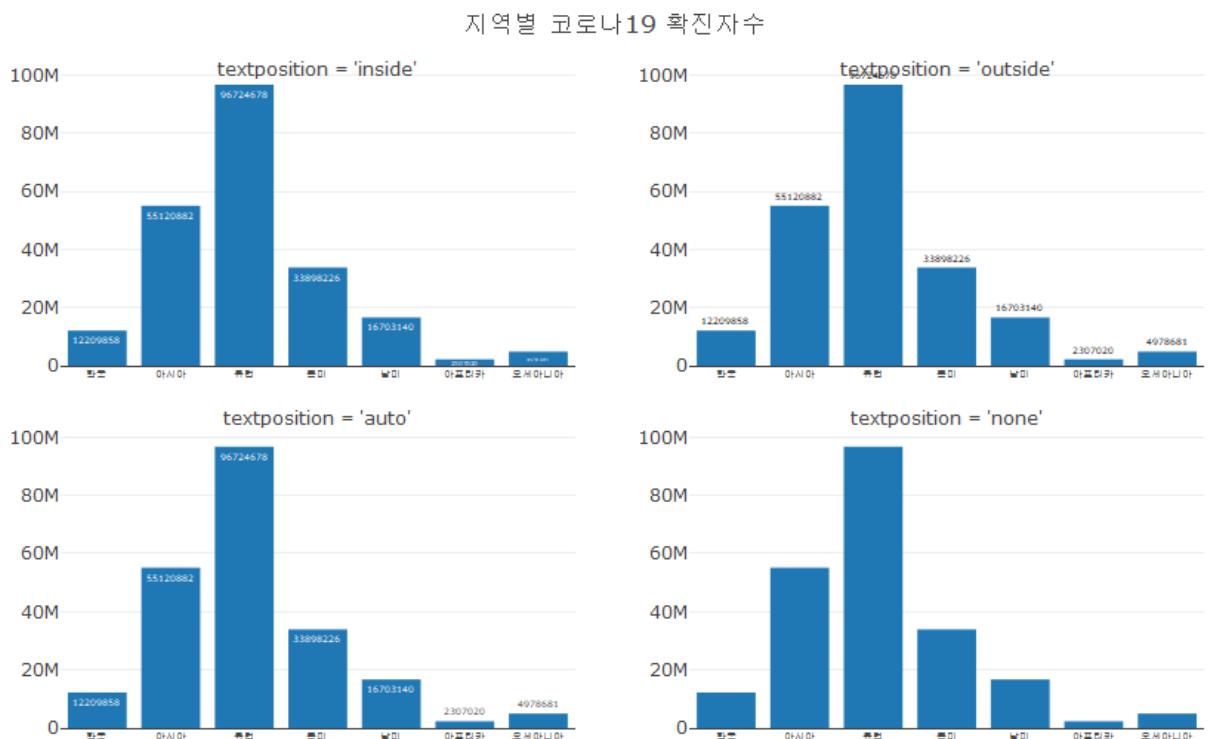
covid19_df_100 |>
  group_by(location) |>
  summarise(new_cases = sum(new_cases)) |>
  plot_ly(x = ~location, y = ~new_cases, text = ~new_cases,
          ## textposition 을 'auto'로 설정
          textposition = 'auto') |>
  layout(title = list(text = '지역별 코로나 19 확진자수'),
         xaxis = list(title = '지역'),
         yaxis = list(title = '확진자수'),
         margin = margins)

covid19_df_100 |>
  group_by(location) |>
```

```

summarise(new_cases = sum(new_cases)) |>
plot_ly(x = ~location, y = ~new_cases, text = ~new_cases,
## textposition 을 'none'으로 설정
textposition = 'none') |>
layout(title = list(text = '지역별 코로나 19 확진자수'),
xaxis = list(title = '지역'),
yaxis = list(title = '확진자수'),
margin = margins)

```



### 실행결과 3- 10. textposition 에 설정 결과

`texttemplate` 은 텍스트가 표시되는 형태를 설정하는 type 이다. `texttemplate` 는 `hovertemplate` 의 정의와 같이 사용되는데 '%{변수:변수포맷}'의 형태로 사용된다. 변수 포맷에서 사용하는 포맷은 자바 스크립트의 d3 format<sup>3</sup>을 사용한다. 앞의 예에서 일반대학의 입학생수의 포맷에 천단위 콤마를 넣는다면'%{text:}'로 설정하고 소수점 아래 두째 자리까지 표기한다면'%{text:2f}'로 설정한다.

```

covid19_df_100 |>
group_by(location) |>

```

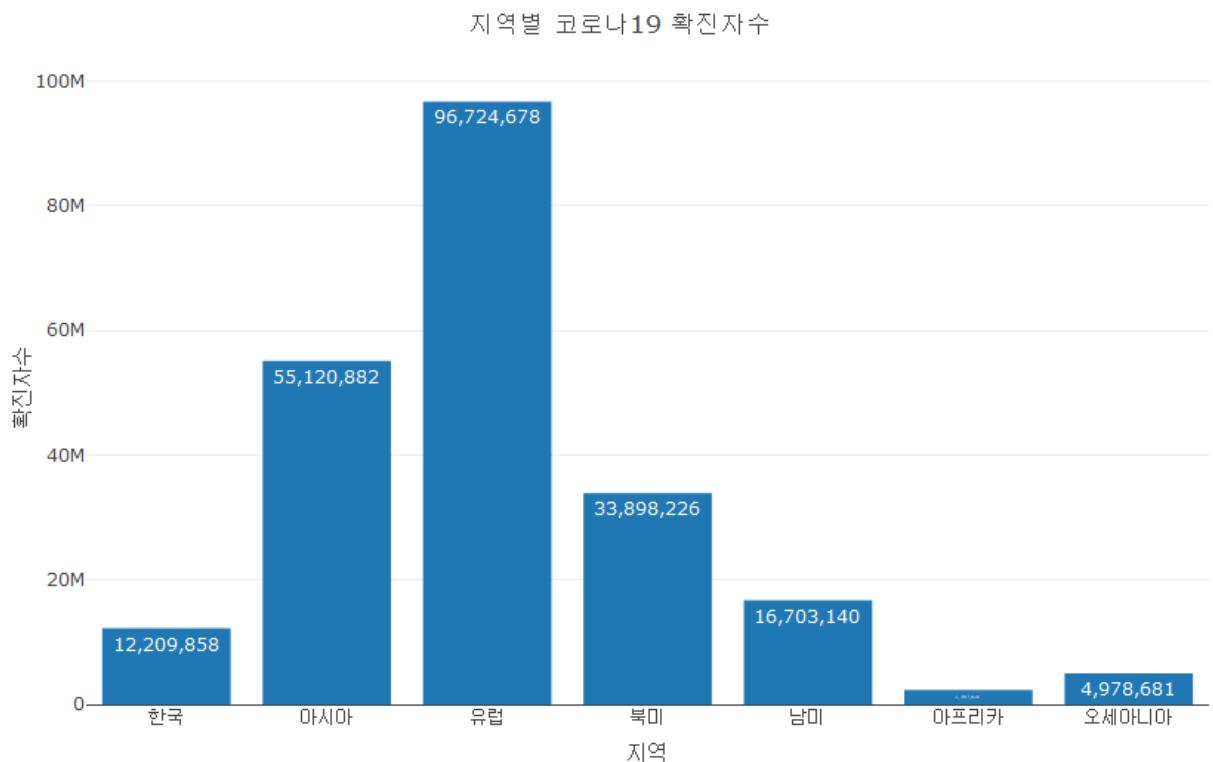
---

<sup>3</sup> <https://github.com/d3/d3-format/tree/v1.4.5#d3-format>

```

summarise(new_cases = sum(new_cases)) |>
plot_ly(x = ~location, y = ~new_cases, text = ~new_cases,
        textposition = 'inside',
        ## texttemplate 를 설정
        texttemplate = '%{text:,}')
) |>
## 제목, 축제목, 여백 설정
layout(title = list(text = '지역별 코로나19 확진자수'),
       xaxis = list(title = '지역'),
       yaxis = list(title = '확진자수'),
       margin = margins)

```



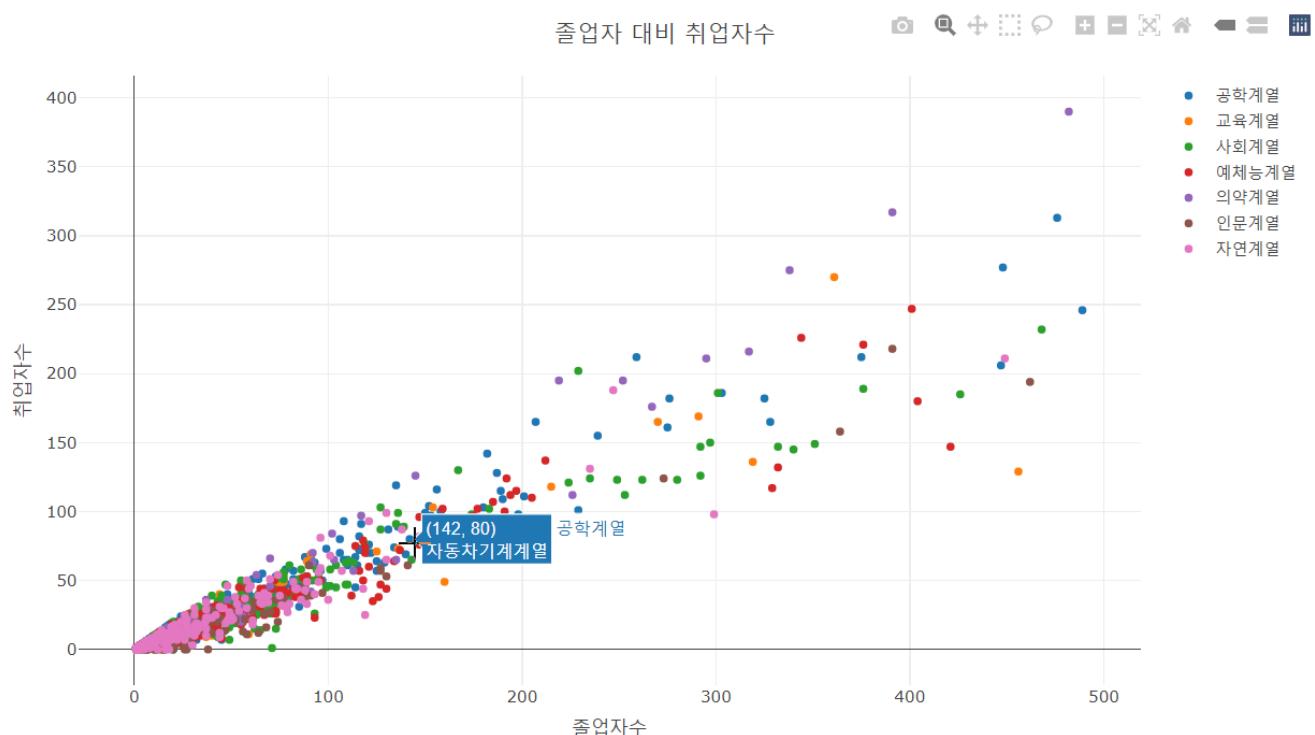
### 실행결과 3- 11. texttemplate 설정 결과

#### 2.2.1.4. hovertext, hoverinfo, hovermode, hovertemplate

`plotly` 와 같은 동적 시각화에서는 대부분 마우스 포인터를 데이터가 표시된 점이나 선에 위치하면 해당 위치의 데이터가 표시된다. `plotly` 에서는 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.

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

```
df_취업통계_2000 |>
  plot_ly(x = ~졸업자수, y = ~취업자수, name = ~대계열,
          ## hovertext 를 학과명으로 매핑
          hovertext = ~학과명) |>
  ## 제목과 여백의 설정
  layout(title = '졸업자 대비 취업자수', margin = margins)
```



### 실행결과 3- 12. hovertext 매핑 결과

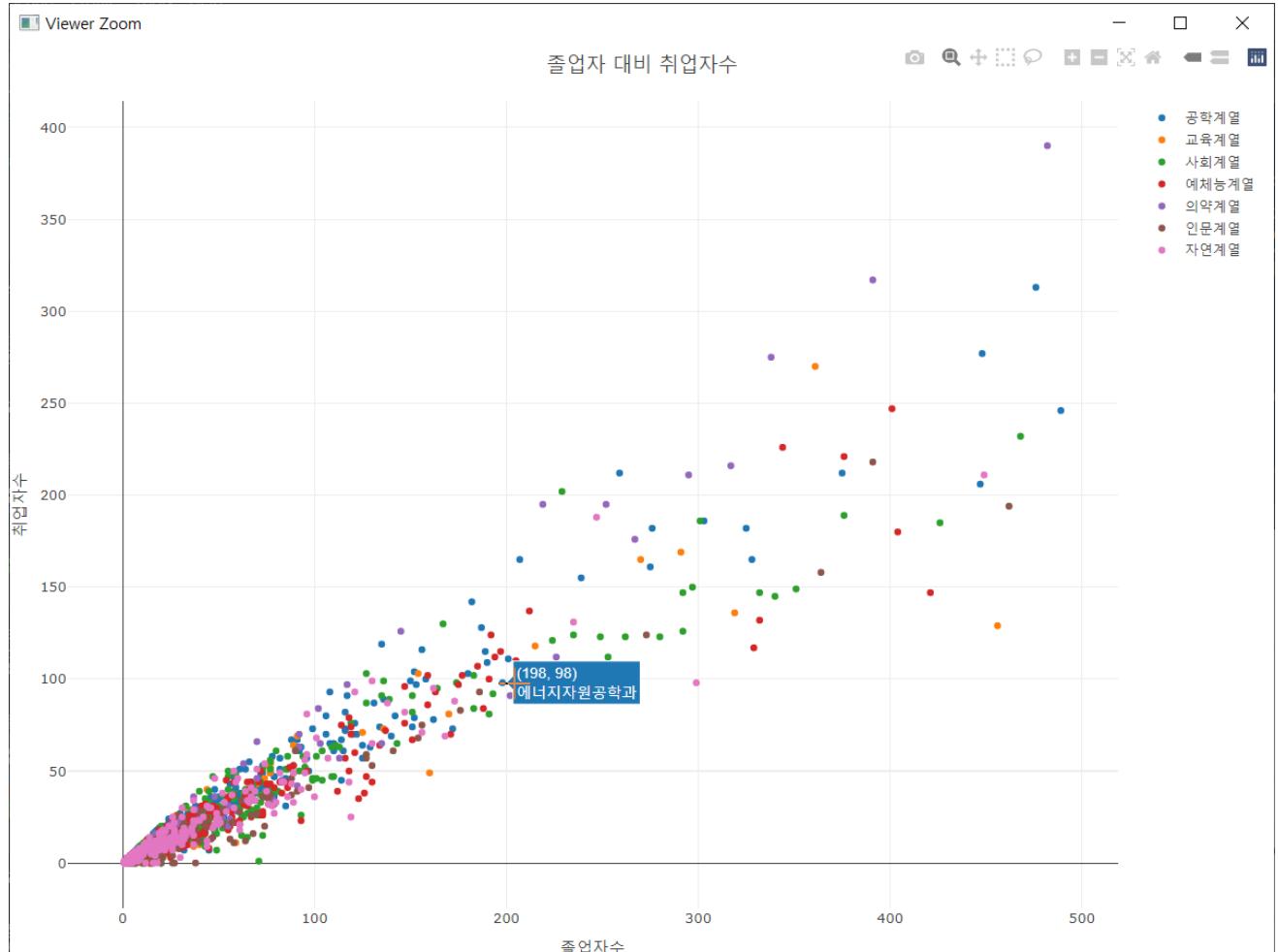
`hoverinfo` 는 호버에 표시되는 데이터 정보를 설정하는 속성이다. 호버에 표시되는 데이터 정보는 각각의 trace에 따라 달라지지만 각 trace에서 많이 사용되는 `hoverinfo` 속성은 x, y, z 축의 좌표를 표기하는 것으로 `x`(X 축 좌표), `y`(Y 축 좌표), `z`(Z 축 좌표) 속성을 사용한다. 이 외에도, `text`(특정 문자열), `name`(trace name), `none`(제거), `skip`(생략)이 사용될 수 있고 각각은 `+`를 사용하여 조합할 수 있다.

```
df_취업통계_2000 |>
  plot_ly(x = ~졸업자수, y = ~취업자수, name = ~대계열,
```

```

## hovertext, hoverinfo 의 설정
hovertext = ~학과명, hoverinfo = 'x+y+text') |>
layout(title = '졸업자 대비 취업자수', margin = margins)

```



### 실행결과 3- 13. hoverinfo 설정 결과

`hovertext` 는 X, Y 좌표에 표시되는 문자열을 설정하는 속성이다. `hovertext` 는 앞서 설명한 `hoverinfo` 를 `text` 로 설정하고 `hovertext` 를 설정할 때 호버 상자 안에 `text` 설정값이 표시되지만, `hoverinfo` 를 생략하고 `hovertext` 만 설정하면 호버 상자 밖에 `name` 속성 값(범례 값)이 표시된다는 차이가 있다. 또 `hovertext` 에 단순 문자열을 설정하면 모든 데이터 호버에 동일한 문자열이 표기되지만 벡터를 설정하면 각각의 호버에 매핑된 결과가 표시된다.

```

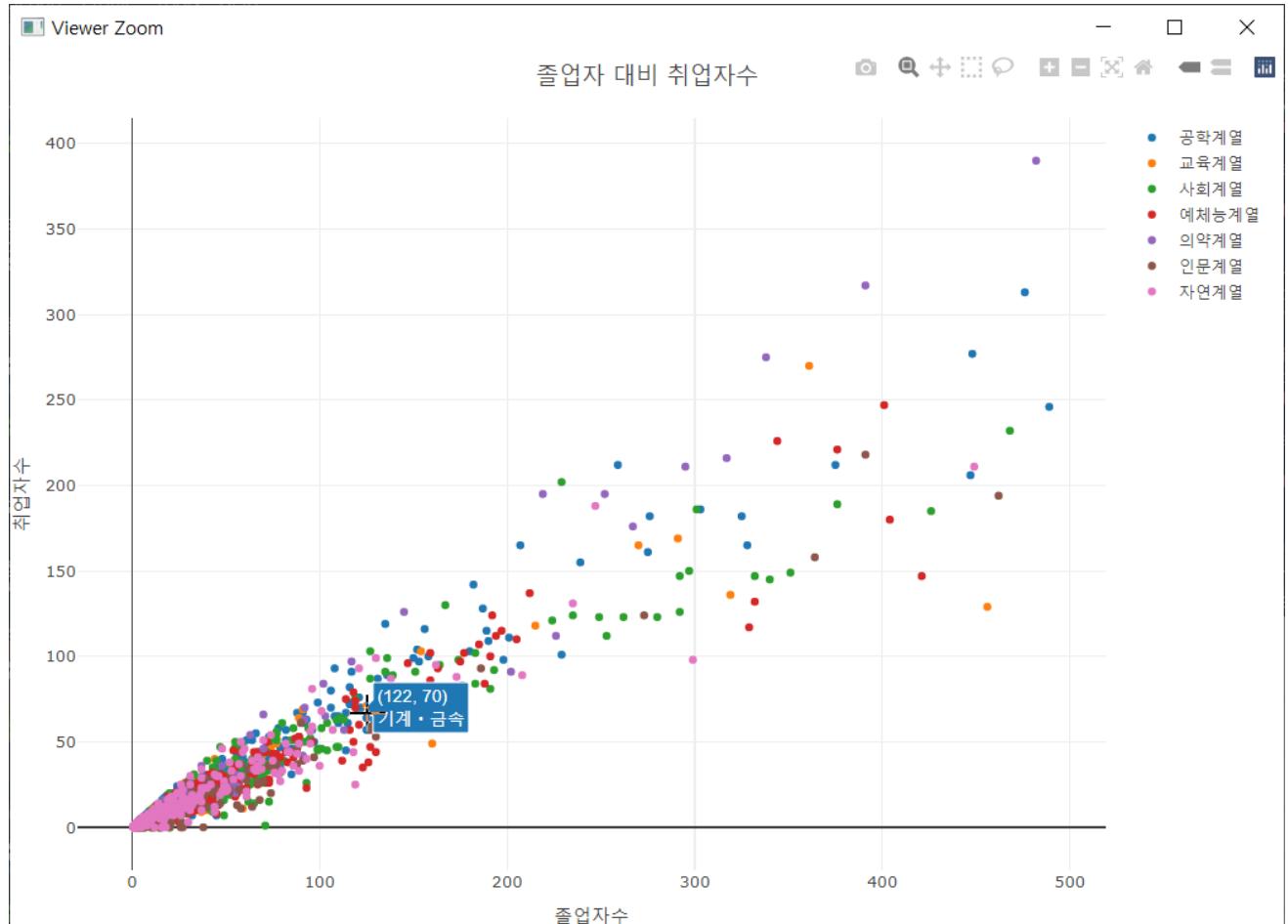
df_취업통계_2000 |>
plot_ly(x = ~졸업자수, y = ~취업자수, name = ~대계열,
## hoverinfo 를 설정하고 hovertext 를 중계열로 매핑

```

```

    hoverinfo = 'x+y+text', hovertext = ~중계열) |>
layout(title = '졸업자 대비 취업자수', margin = margins)

```

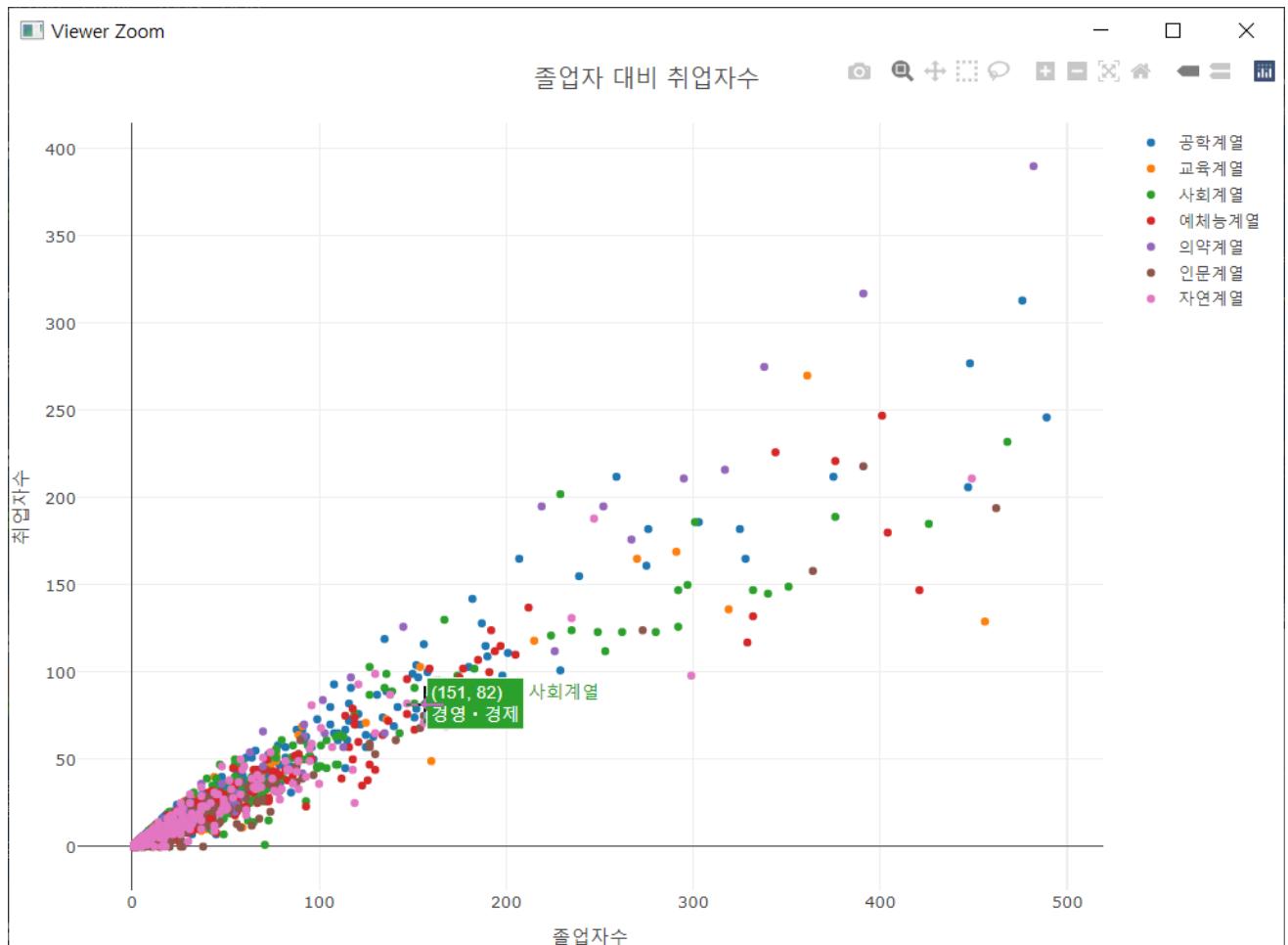


실행결과3- 14. hoverinfo 와 hovertext 동시 매핑 결과

```

df_취업통계_2000 |>
plot_ly(x = ~졸업자수, y = ~취업자수, name = ~대계열,
## hoverinfo 를 설정없이 hovertext 를 중계열로 매핑
hovertext = ~중계열) |>
layout(title = '졸업자 대비 취업자수', margin = margins)

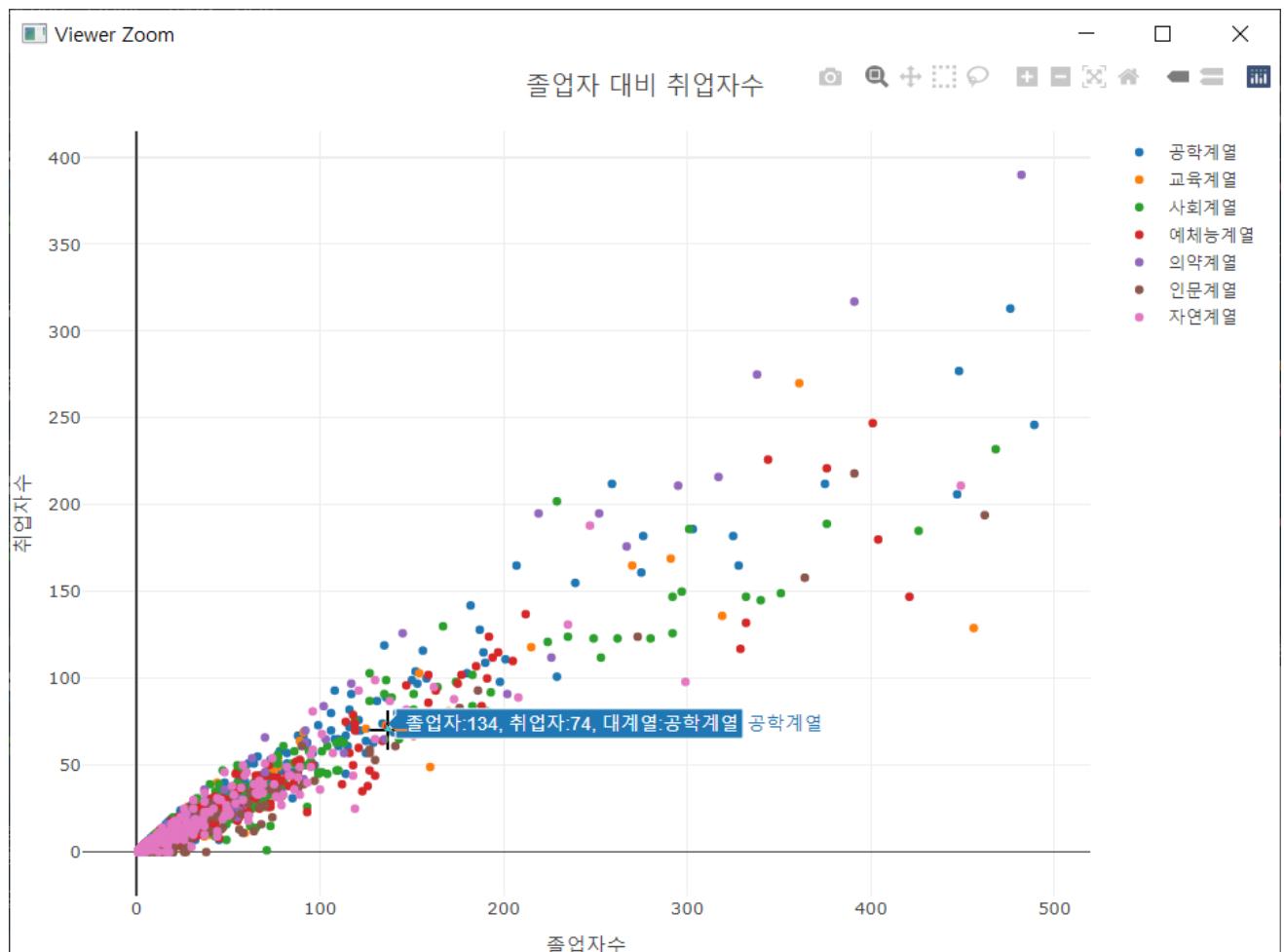
```



### 실행결과 3- 15. hoverinfo 없이 hovertext 매핑 결과

`hovertemplate` 는 호버 상자와 호버 상자에 표시되는 정보의 포맷을 설정하는 속성이다. 이 속성은 앞서 설명한 `hoverinfo` 에 설정된 사용되는 속성의 변수를 `%{변수}`의 형태로 사용할 수 있다. 예를 들어 호버 상자에 Y 축 값을 'Y 값 : '을 붙여 표시하기 위해서는 `Y 값 : %{y}`로 설정한다. 앞서 사용된 예에서 X 축의 값에 '졸업자:', Y 축의 값에 '취업자:', 대계열에 '대계열:'를 표시하고 값을 표시하는 코드는 다음과 같다.

```
df_취업통계_2000 |>
  plot_ly(x = ~졸업자수, y = ~취업자수, name = ~대계열,
           hoverinfo = 'x+y+text', text = ~대계열,
           ## hovertemplate 의 설정
           hovertemplate = '졸업자:{x}, 취업자:{y}, 대계열:{text}') |>
  layout(title = '졸업자 대비 취업자수', margin = margins)
```

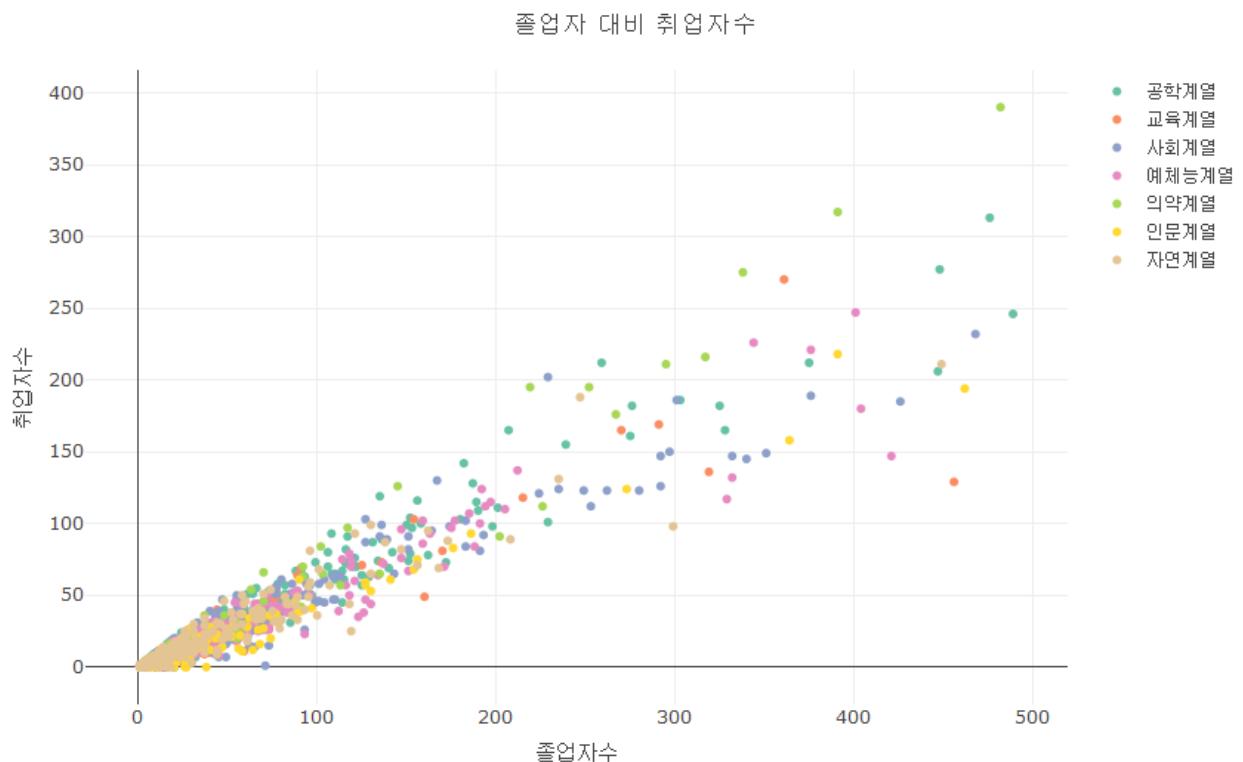


실행결과3- 16. hovertemplate 설정 결과

#### 2.2.1.5. color, colors

`color` 는 스캐터 trace에 표현되는 점, 선, 문자의 내부 색을 설정하는 속성이다. 내부 색을 설정할 때는 먼저 색을 변수에 매핑할지, 특정 색상으로 설정할지를 결정해야 한다. 이를 설정하는 매개변수가 `color` 이다. `color`에 변수를 ~를 사용하여 매핑하면 해당 변수의 값에 따라 색이 매핑되어 표현된다.

```
df_취업통계_2000 |>
## X, Y 축의 매핑과 color 를 대계열로 매핑
plot_ly(x = ~졸업자수, y = ~취업자수, color = ~대계열) |>
layout(title = '졸업자 대비 취업자수', margin = margins)
```



### 실행결과 3- 17. color 매핑 결과

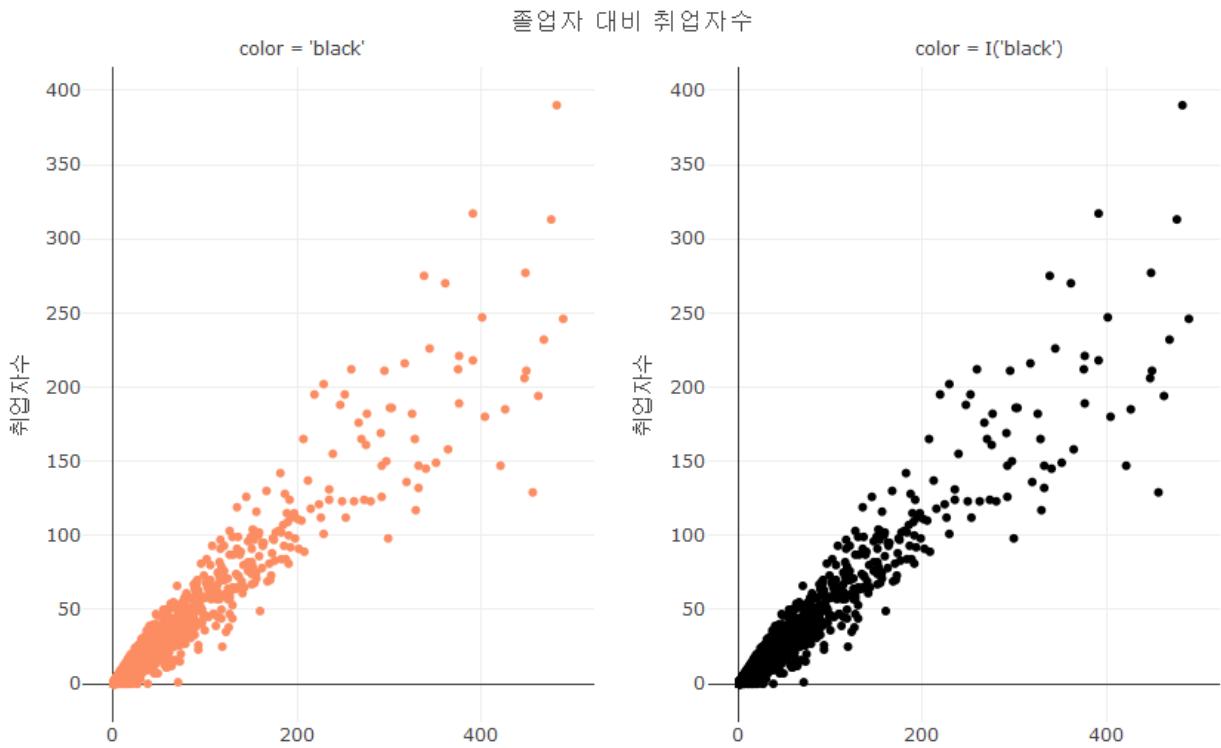
반면 특정한 색으로 설정할 때는 `color`에 특정 색 이름을 설정하면 모든 marker 가 동일한 색으로 나타난다. 이 과정에서 하나 주의해야 할 것이 설정의 방법이다. 일반적인 변수 설정과 같이 색상명을 설정하면 다음과 같이 정확한 색상이 나타나지 않는다.

```
df_취업통계_2000 |>
## plot_ly()에 color 를 색 이름 설정
plot_ly(x = ~졸업자수, y = ~취업자수, color = 'black') |>
layout(title = '졸업자 대비 취업자수', margin = margins)
```

`plotly`에서 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 `plotly`의 디폴트 색 팔레트를 사용하여 색이 설정된다. 자신이 원하는 색을 직접 설정하기 위해서는 '`I()`'를 의미하는 `I()`를 사용하여 색을 설정하여야 한다.

```
df_취업통계_2000 |>
## plot_ly()에 color 를 I() 사용하여 색 이름 설정
```

```
plot_ly(x = ~졸업자수, y = ~취업자수, color = I('black')) |>
  layout(title = '졸업자 대비 취업자수', margin = margins)
```



### 실행결과 3- 18. `I()`를 사용한 `color` 설정 결과

`colors` 는 `color` 에서 매핑된 변수에 따른 색의 스케일을 설정하는 속성이다. `plotly` 에서 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 `ggplot2`에서도 많이 사용된다. 이 패키지에서 제공하는 팔레트의 이름을 `colors`에 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 1 까지의 값 범위내에 해당하는 색을 반환해준다.

```

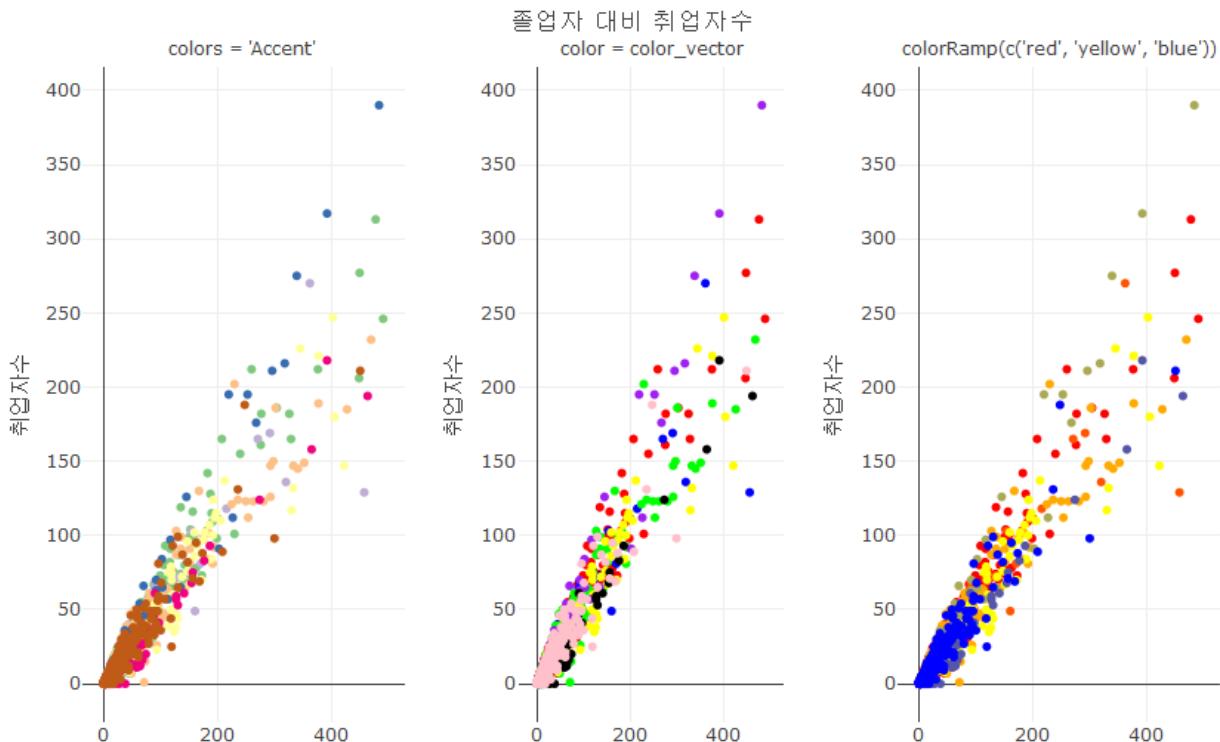
df_취업통계_2000 |>
  plot_ly(x = ~졸업자수, y = ~취업자수,
    ## color 를 대계열로 매핑하고 colors 를 'Accent' 팔레트로 설정
    color = ~대계열, colors = 'Accent') |>
  layout(title = '졸업자 대비 취업자수', margin = margins)

color_vector <- c('red', 'blue', 'green', 'yellow', 'purple', 'black', 'pink')

df_취업통계_2000 |>
  plot_ly(x = ~졸업자수, y = ~취업자수,
    ## color 를 대계열로 매핑하고 colors 를 색 이름 벡터로 설정
    color = ~대계열, colors = color_vector) |>
  layout(title = '졸업자 대비 취업자수', margin = margins)

df_취업통계_2000 |>
  plot_ly(x = ~졸업자수, y = ~취업자수,
    ## color 를 대계열로 매핑하고 colors 를 colorRamp()로 설정
    color = ~대계열, colors = colorRamp(c('red', 'yellow', 'blue'))) |>
  layout(title = '졸업자 대비 취업자수', margin = margins)

```



### 실행결과 3- 19. 색 팔레트 설정 결과

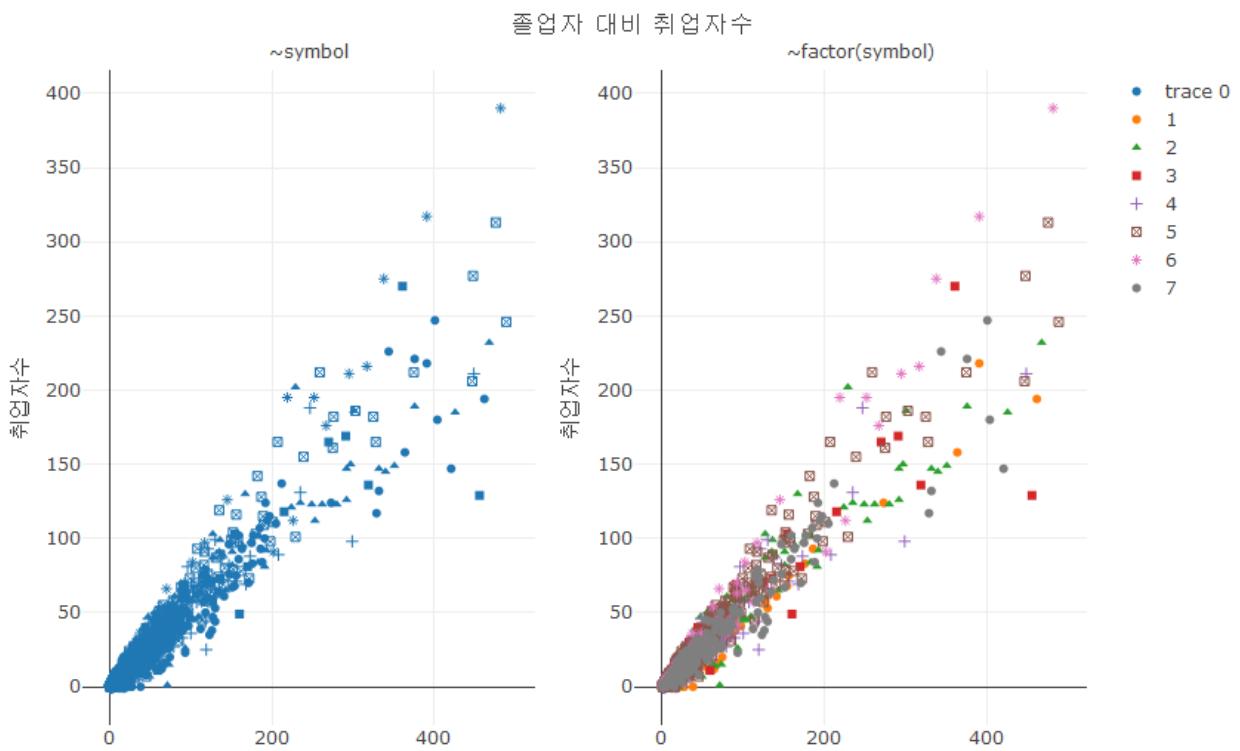
#### 2.2.1.6. symbol, symbols

`symbol` 은 점의 형태를 설정하는 속성이다. 앞서 `color` 와 같이 `symbol` 도 매핑할 변수를 설정하는 속성이고 `symbols` 는 변수 카테고리에 따라 설정하는 점의 형태를 설정하는 속성이다. `color` 와 다른 점은 연속형 변수에 매핑되는 `symbol` 은 단 하나의 카테고리로 분류된다는 점이다. 그렇기 때문에 좌표에 표시되는 점의 형태는 다르게 보이지만 범례에 구분되어 표현되지는 않는다는 점이다. 다음의 예를 살펴보자.

```
df_취업통계_symbol <- df_취업통계_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_취업통계_symbol |>
  plot_ly(x = ~졸업자수, y = ~취업자수,
          ## symbol 을 연속형 수치 변수에 매핑
          symbol = ~symbol) |>
  layout(title = '졸업자 대비 취업자수', margin = margins)

df_취업통계_symbol |>
  plot_ly(x = ~졸업자수, y = ~취업자수,
          ## symbol 을 팩터형 변수에 매핑
          symbol = ~factor(symbol)) |>
  layout(title = '졸업자 대비 취업자수', margin = margins)
```

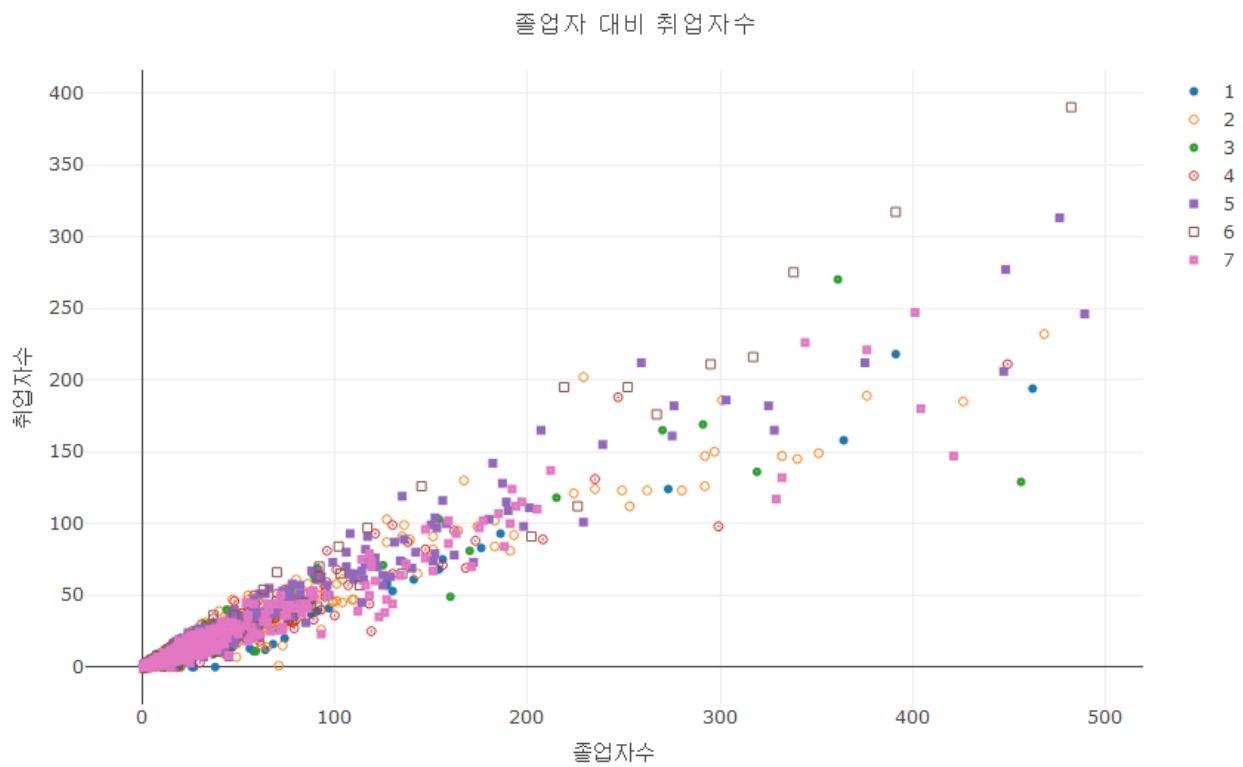


실행결과3- 20. symbol 연속형, 팩터형 변수 매핑 결과

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.

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

```
df_취업통계_symbol |>
  plot_ly(x = ~졸업자수, y = ~취업자수,
          symbol = ~factor(symbol),
          ## symbols 를 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)
```

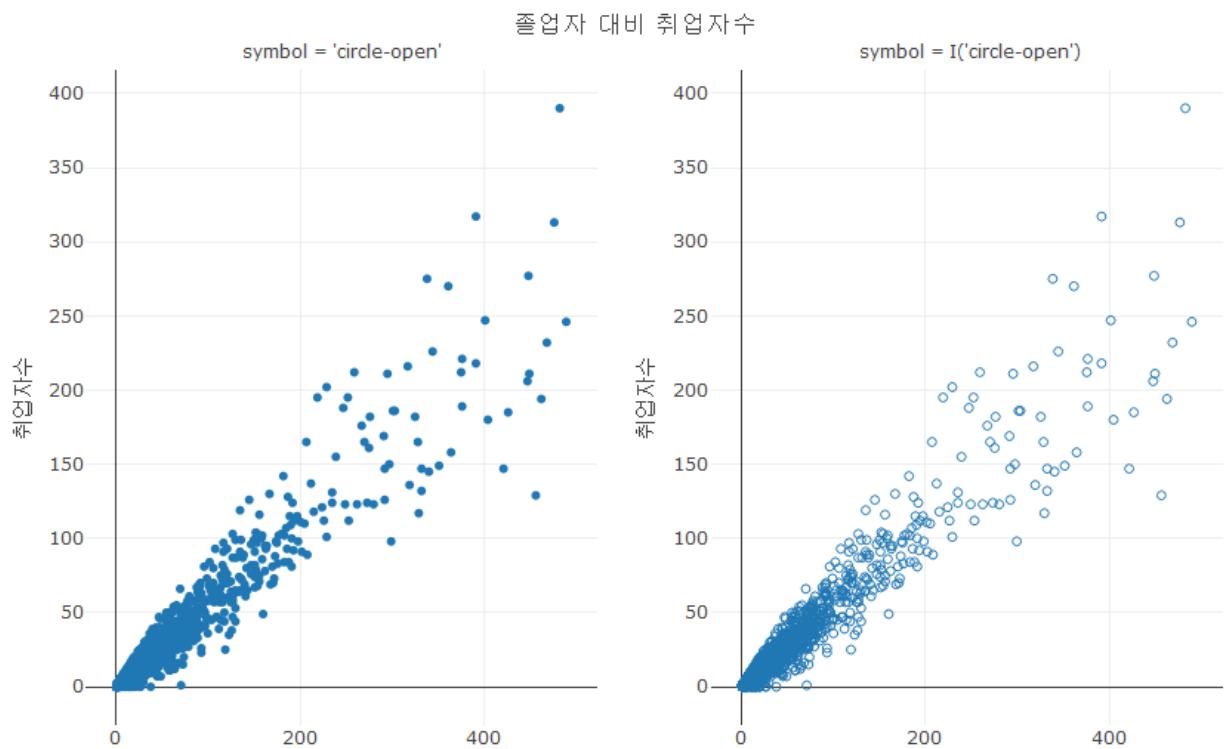


### 실행결과 3- 21. symbols 설정 결과

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

```
df_취업통계_symbol |>
## symbol 을 심볼 이름으로 설정
plot_ly(x = ~졸업자수, y = ~취업자수, symbol = 'circle-open') |>
layout(title = '졸업자 대비 취업자수', margin = margins)
```

```
df_취업통계_symbol |>
## symbol 을 I()를 사용하여 심볼 이름으로 설정
plot_ly(x = ~졸업자수, y = ~취업자수, symbol = I('circle-open')) |>
layout(title = '졸업자 대비 취업자수', margin = margins)
```

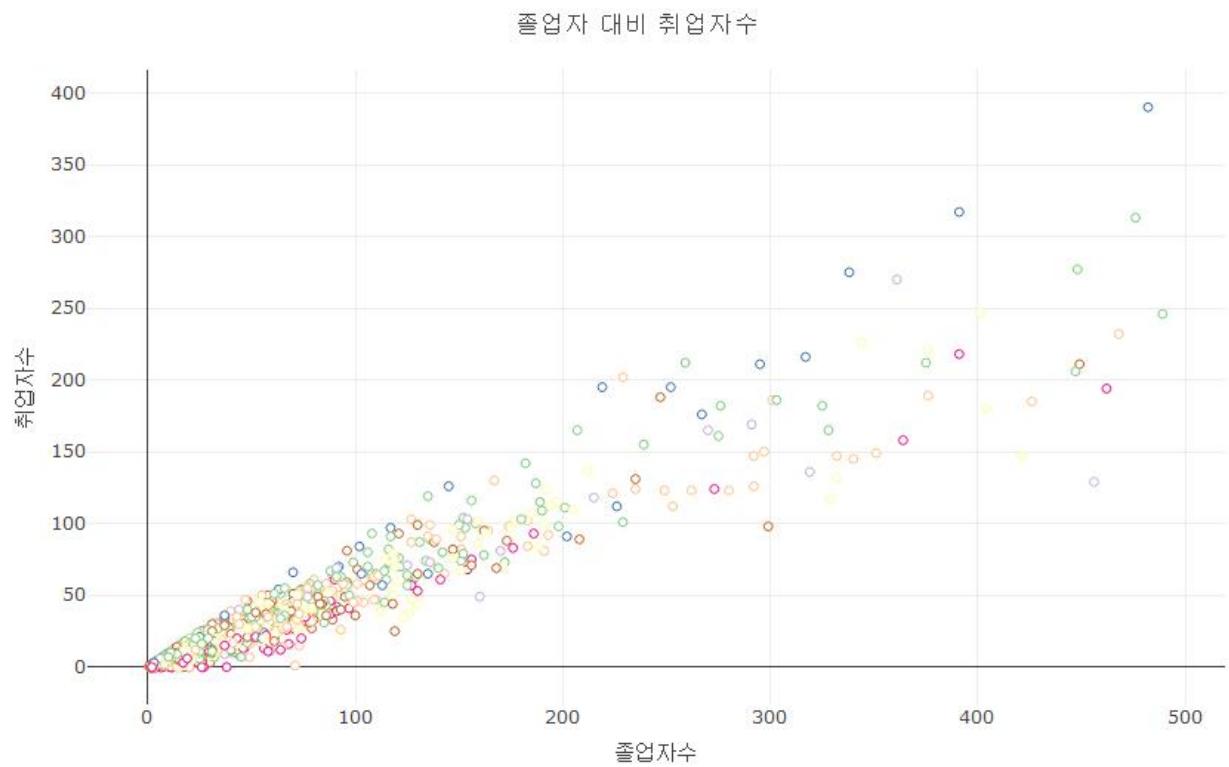


실행결과3- 22. `I()`를 사용한 `symbol` 설정 결과

### 2.2.1.7. `stroke`, `strokes`

`stroke` 는 점의 외곽선 색을 설정하는 속성이다. `color` 의 설정은 점의 내부색과 외곽선 색을 동시에 설정하지만 외곽선 색을 따로 매핑해야 할 경우에 `stroke` 를 사용하여 매핑하거나 설정할 수 있다. `stroke` 와 `strokes` 는 `color` 나 `symbol` 과 같이 변수 매핑으로 사용되는 매개변수는 `stroke` 이고 매핑된 변수의 카테고리에 해당하는 외곽선 색을 설정하는 매개변수는 `strokes` 이다. `strokes` 에 색을 설정하는 방법은 `colors` 에서 사용한 세 가지 방법을 같이 사용할 수 있다.

```
df_취업통계_2000 |>
  plot_ly(x = ~졸업자수, y = ~취업자수, color = I('white'),
    ## stroke 를 대계열로 매핑, strokes 를 Accent 팔레트로 설정
    stroke = ~대계열, strokes = 'Accent') |>
  layout(title = '졸업자 대비 취업자수', margin = margins)
```

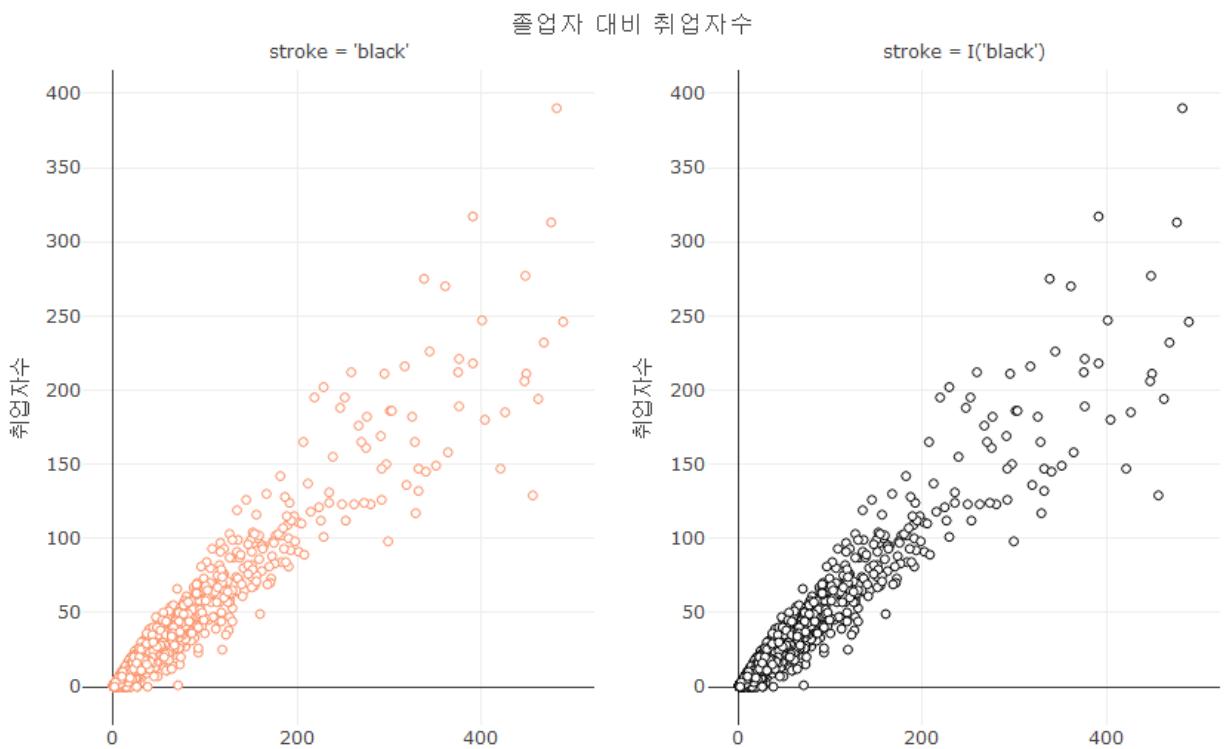


### 실행결과 3- 23. stroke 매핑과 strokes 의 설정 결과

`stroke` 를 특정 값으로 설정하기 위해서는 `I()`를 사용한다.

```
df_취업통계_2000 |>
  plot_ly(x = ~졸업자수, y = ~취업자수, color = I('white'),
           ## stroke에 색 이름 설정
           stroke = 'black') |>
  layout(title = '졸업자 대비 취업자수', margin = margins)
```

```
df_취업통계_2000 |>
  plot_ly(x = ~졸업자수, y = ~취업자수, color = I('white'),
           ## stroke에 I()를 사용한 색 이름 설정
           stroke = I('black')) |>
  layout(title = '졸업자 대비 취업자수', margin = margins)
```

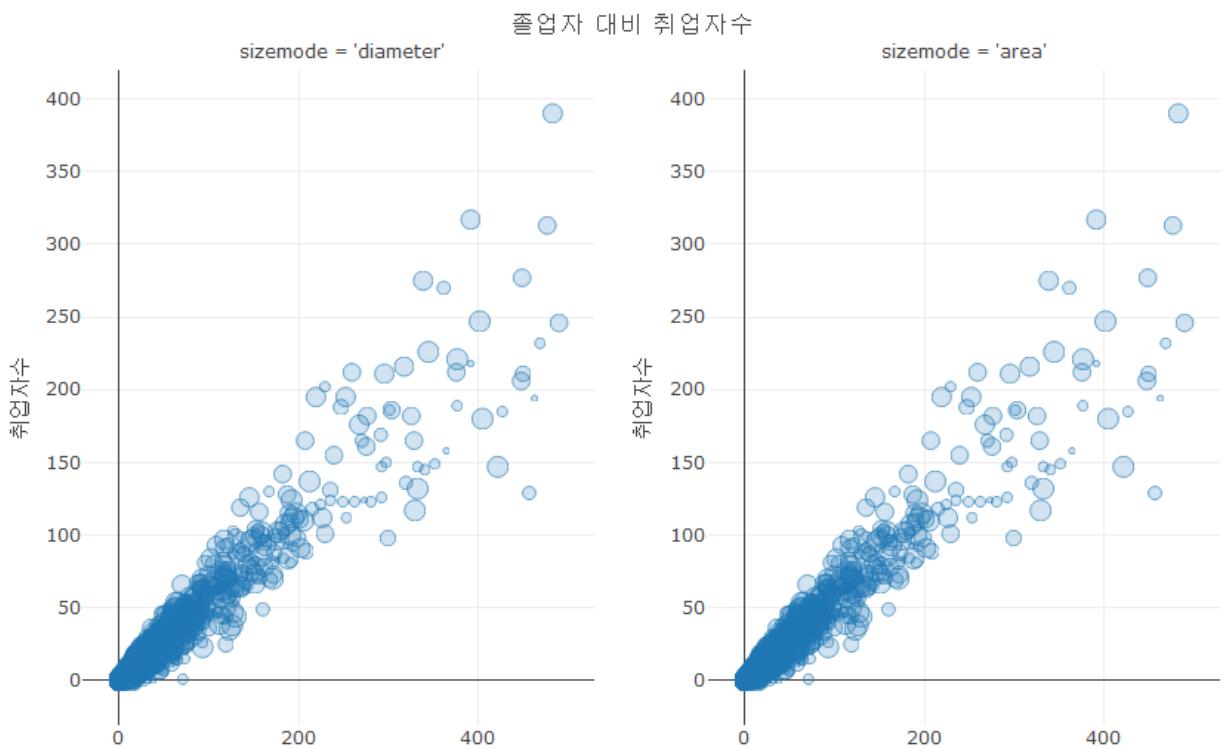


### 2.2.1.8. size, sizes

`size`는 점의 크기를 설정하는 속성이다. `size`는 변수 매핑을 위한 속성이고 `sizes`는 변수에 매핑된 카테고리 별로 크기를 설정하는 속성이다. 앞선 속성들과 달리 `size`는 `sizemode` 속성을 추가적으로 사용할 수 있다. `sizemode`는 점의 크기를 결정하는 기준을 설정하는 속성으로 점의 크기를 지름으로 설정하는 'diameter'와 면적으로 설정하는 'area'의 두 가지가 있다.

```
df_취업통계_symbol |>
  plot_ly(x = ~출업자수, y = ~취업자수, alpha = 0.3,
          size = ~symbol, sizemode = 'diameter') |>
  layout(title = '출업자 대비 취업자수', margin = margins)
```

```
df_취업통계_symbol |>
  plot_ly(x = ~출업자수, y = ~취업자수, alpha = 0.3,
          size = ~symbol, sizemode = 'area') |>
  layout(title = '출업자 대비 취업자수', margin = margins)
```

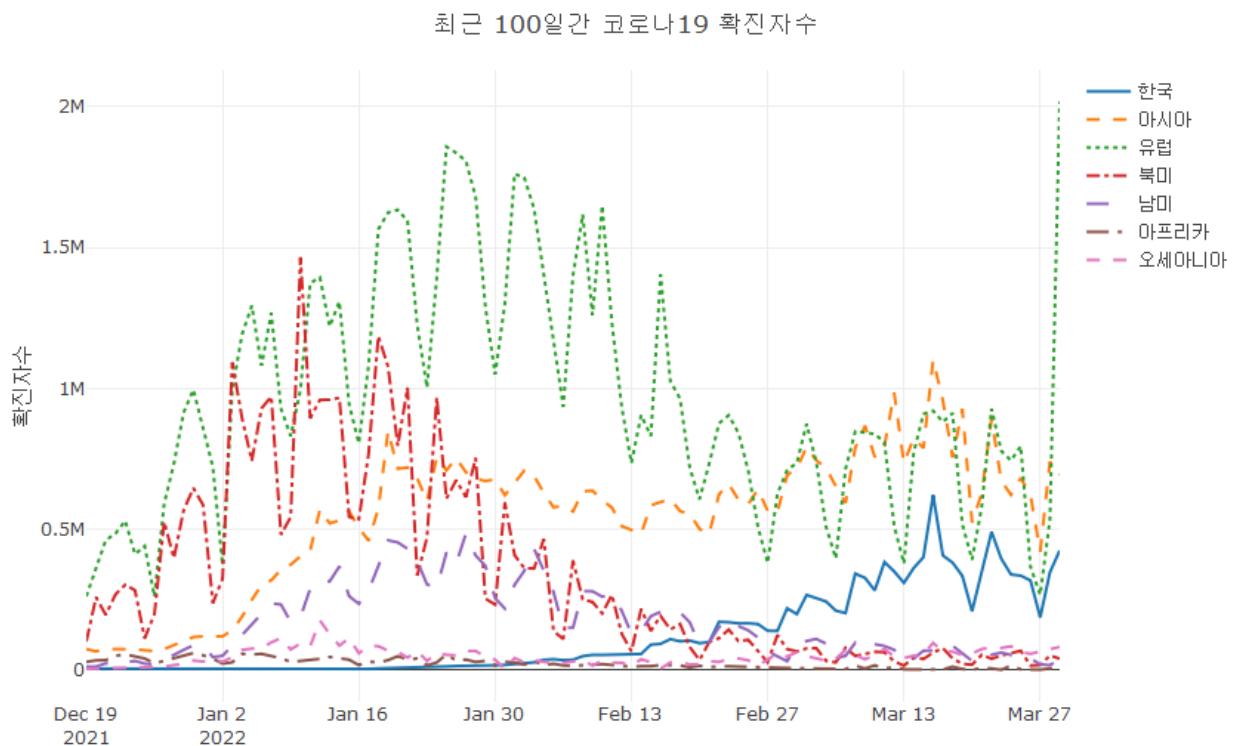


실행결과 3-24. size 매핑과 sizemode 의 설정 결과

### 2.2.1.9. linetype, linetypes

`linetype` 은 선의 형태를 설정하는 속성이다. `plotly` 에서 선의 형태를 설정할 때 변수 매핑 위해서는 `linetype` 을 사용하고 `linetype` 에 매핑된 변수에 따라 선의 형태를 설정하기 위해서는 `linetypes` 를 사용한다.

```
covid19_df_100 |>
  plot_ly(type = 'scatter', mode = 'lines', x = ~date, y = ~new_cases,
    ## Linetype と Location 0// 매핑
    linetype = ~location) |>
  layout(title = '최근 100 일간 코로나 19 확진자수',
    xaxis = list(title = list(text = NULL)),
    yaxis = list(title = list(text = '확진자수')),
    margin = margins)
```



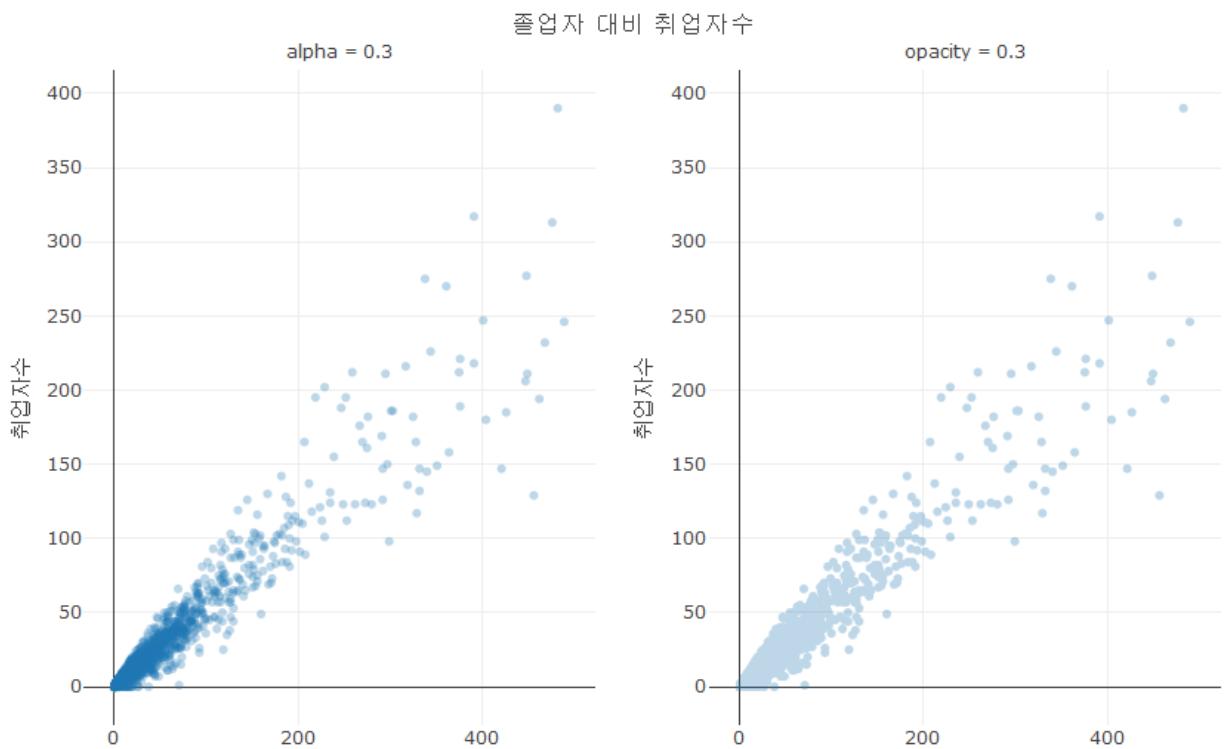
### 실행결과 3- 25. linetype 의 매팅 결과

#### 2.2.1.10. alpha, opacity

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

```
df_취업통계_2000 |>
## alpha 를 0.3 으로 설정
plot_ly(x = ~졸업자수, y = ~취업자수, alpha = 0.3) |>
layout(title = '졸업자 대비 취업자수', margin = margins)
```

```
df_취업통계_2000 |>
## opacity 를 0.3 으로 설정
plot_ly(x = ~졸업자수, y = ~취업자수, opacity = 0.3) |>
layout(title = '졸업자 대비 취업자수', margin = margins)
```

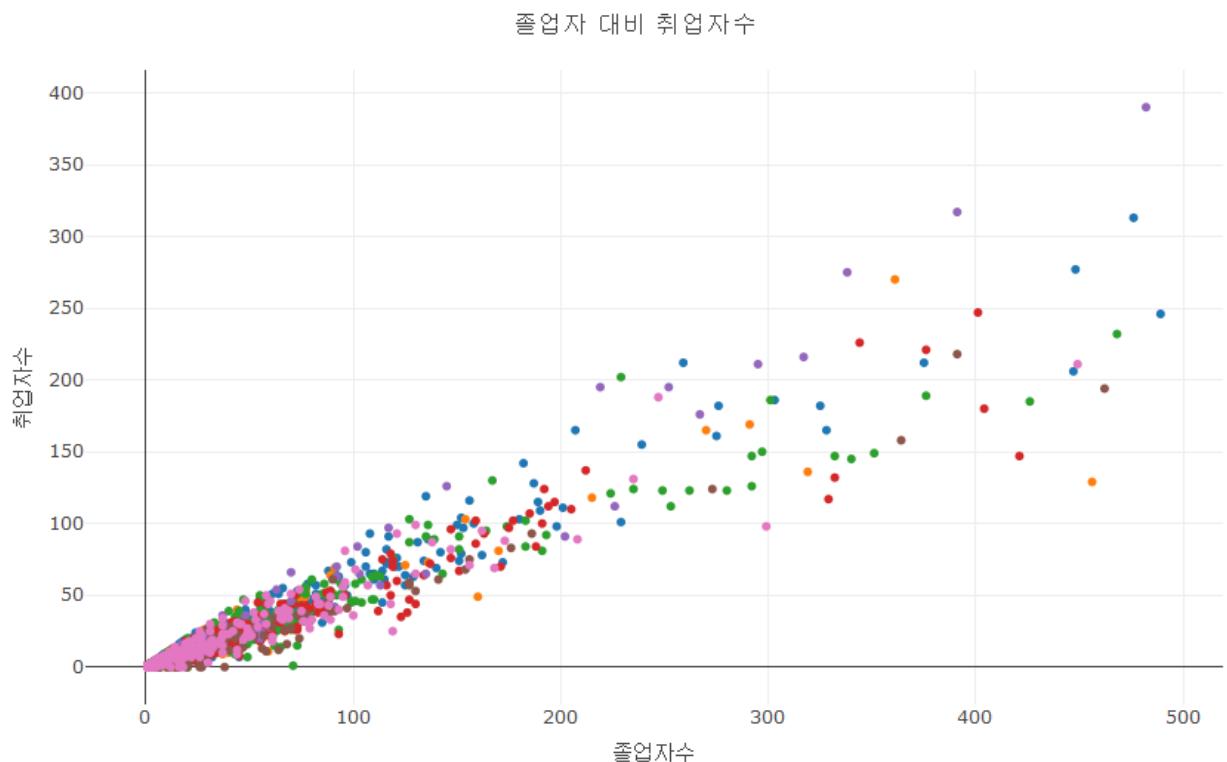


실행결과 3- 26. alpha 와 opacity 의 설정 결과

#### 2.2.1.11. showlegend

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

```
df_취업통계_2000 |>
  plot_ly(x = ~졸업자수, y = ~취업자수, name = ~대계열,
          ## showLegend 을 FALSE 로 설정
          showlegend = FALSE) |>
  layout(title = '졸업자 대비 취업자수', margin = margins)
```



실행결과 3- 27. *showlegend* 설정 결과

### 2.2.2. trace 추가 : add\_trace()

trace 는 `plotly` 객체에서 표현되는 단일 데이터의 그래픽적 표현을 의미한다. 이를 간단히 이야기하자면 범례에 표시되는 데이터 아이템 하나 하나가 각각의 개별 trace 라고 생각하면 편할 것이다. `ggplot2` 에서는 `geom_*`()을 사용하여 점, 선, 막대 등의 기하 요소로 표현하였지만 `plotly` 객체에서는 trace 라는 이름으로 각각의 데이터 표현을 추가한다.

`ggplot2` 에 익숙한 사용자는 이 trace 의 개념과 `ggplot2` 의 기하 요소와의 차이를 잘 이해하여야 한다. `ggplot2` 에서 데이터를 표현하는 기하 요소는 변수 매핑을 통해 하나의 `geom_*`()으로 생성하는 것이 일반적이다. 예를 들어 `ggplot2` 를 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 `plotly` 에서는 이 두가지 방법을 모든 시각화 방법에 동일하게 지원한다. 따라서 `plotly` 를 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, `trace` 를 추가하는 함수인 `add_trace()` 를 한번 호출하여 네개의 막대가 생성된 막대 그래프가 하나가 생성되더라도 `trace` 는 네개가 존재한다는 것이다. 반면 넓은 형태의 데이터프레임이라면 시각화 해야 할 각각의 열을 각각 `trace` 로 추가하여 줌으로써 시각화를 완성한다. `ggplot2` 의 선 그래프에 가능한 방법을 막대 그래프, 박스 플롯 등 모든 `plotly` 시각화에서 사용할 수 있다는 것이다.

`plotly` 에서는 `trace` 를 표현하는데 40 가지 이상의 시각화 type 을 제공한다. 또 각각의 `trace` 의 시각화 type 에는 그에는 해당 `trace` 의 시각화 세부 속성을 설정을 위한 수많은 세부 속성을 가지고 있다. 앞서 `plot_ly()` 에서 속성으로 설정했던 많은 속성들은 모두 개별 `trace` 의 설정으로도 사용이 가능하다. 하지만 `plot_ly()` 에서 `trace` 의 `type` 을 설정하지 않았는데 시각화의 결과는 데이터들의 `trace` 가 나타났다. 이는 `plotly` 에서 데이터를 분석하여 가장 효과적인 `trace` 를 매칭해주기 때문이다. 하지만 `trace` 는 사용자가 지정하는 것이 일반적이다.

이렇게 `trace` 의 `type` 을 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 trace 에 설정할 수 있는 속성 설정
- data : 시각화할 데이터프레임
- inherit : plot_ly()에 설정된 속성 type 을 상속할지를 결정하는 논리값
```

`trace` 의 종류에 따른 각각의 특성과 속성은 다음과 같다.

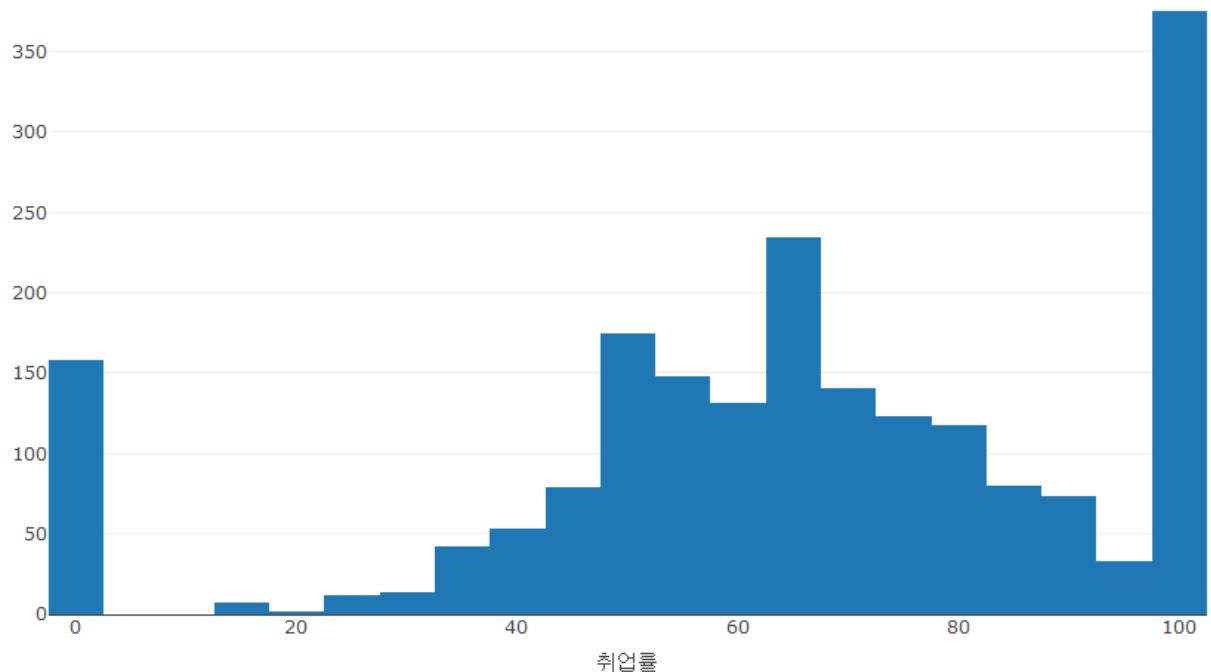
### 2.2.2.1. 히스토그램(histogram) trace

`plotly`에서 히스토그램 시각화를 위한 trace의 `type`은 'histogram'이다. 히스토그램은 하나의 변수에 따른 그 사례 빈도를 산출하는 시각화이고 X 축에 매핑되는 변수 외에 간격(bin)을 설정함으로써 히스토그램을 생성할 수 있다. 히스토그램 trace는 `add_trace()`에 `type`을 '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 : X 축에 매핑할 변수를 ~로 설정  
- y : Y 축에 매핑할 변수를 ~로 설정
```

```
## 취업통계 데이터를 사용해 plotly 객체 생성  
p_histogram <- df_취업통계_2000 |> plot_ly()  
  
p_histogram |>  
  ## histogram trace로 x 축을 취업률로 매핑, name을 취업률로 설정  
  add_trace(type = 'histogram', x = ~취업률, name = '취업률') |>  
  ## 제목과 여백 설정  
  layout(title = '취업률 histogram', margin = margins)
```

취업률 histogram



실행결과 3-28. 히스토그램 trace 시각화

plotly 의 히스토그램 trace 에서 사용되는 주요 속성들은 다음과 같다.

속성	설명	속성값	세부 속성
type	trace type 설정	'histogram'	
visible	trace 의 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 bin 의 최대값 설정	0 이상의 정수	
xbins, ybins	X, Y 축의 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

#### 2.2.2.1.1. *xbins*

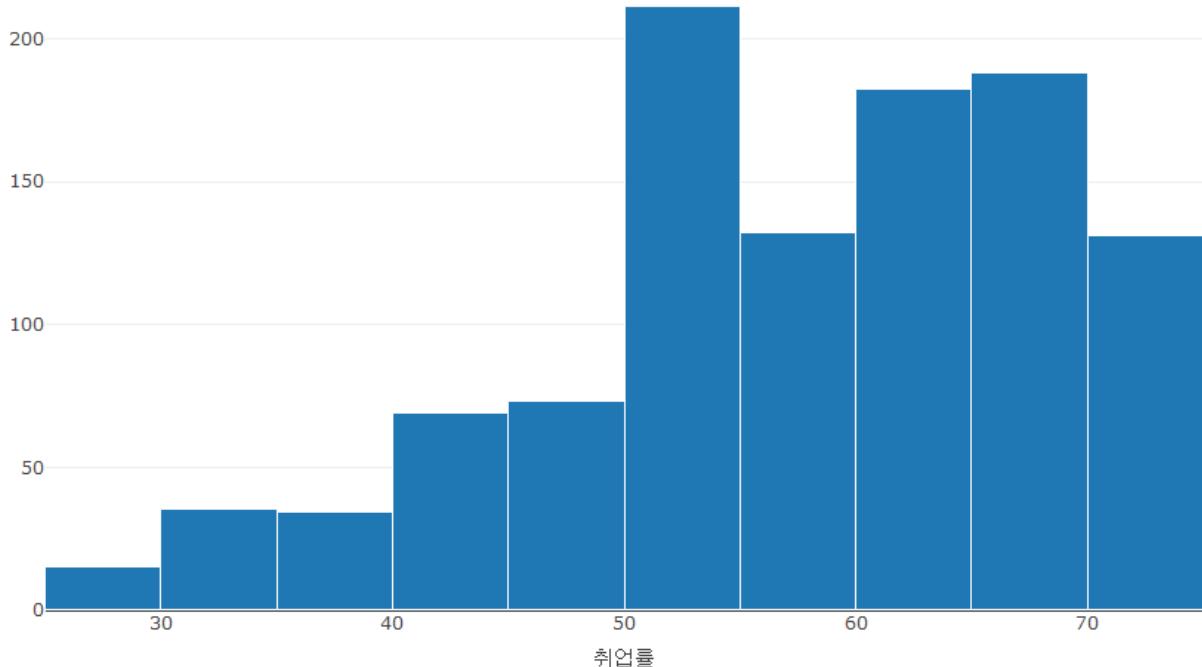
**xbins** 는 X 축에 매핑된 변수가 연속형 수치 변수일 경우 빈도를 표현할 구간의 범위를 설정하는 속성이다. **xbins** 는 바로 수치나 벡터를 설정하는 것이 아니고 **xbins** 에 속하는 하위 세부 type 의 설정을 `list()`로 묶어 정의한다. **xbins** 에서 정의하는 세부 type 의 종류는 `size`, `start`, `end` 이다.

`size` 는 히스토그램의 막대의 X 축 범위를 지정하는 type 이다. 예를 들어 X 축의 수치형 변수 범위가 0 부터 100 까지라고 가정할때 `size` 를 5 으로 설정하면 막대는 총 20 개가 생성된다.

`start` 와 `end` 는 히스토그램에서 표현하는 전체 X 축의 범위를 지정하는 type 이다. 만약 X 축의 수치형 변수 범위가 0 부터 100 까지이라고 하더라도 25 에서 75 까지의 히스토그램을 생성하기를 원한다면 `start` 와 `end` 로 범위를 설정할 수 있다.

```
p_histogram |>
  add_trace(type = 'histogram', x = ~취업률,
            ## xbins 의 start, end, size 설정
            xbins = list(start = 25, end = 75, size = 5),
            ## stroke 를 흰색으로 설정
            stroke = I('white')) |>
  layout(title = '취업률 histogram', margin = margins)
```

취업률 histogram



실행결과 3- 29. `xbins` 를 설정한 히스토그램

#### 2.2.2.1.2. `histfunc`

`histfunc` 은 히스토그램의 막대로 표현하는 값의 종류를 설정하는 속성이다. 일반적으로 히스토그램은 사례수('count')를 사용하지만 `plotly` 에서는 사례수를 포함하여 합계('sum'), 평균('avg'), 최소('min'), 최대('max')를 제공한다. 이를 `histfunc` 으로 설정하면 자동적으로

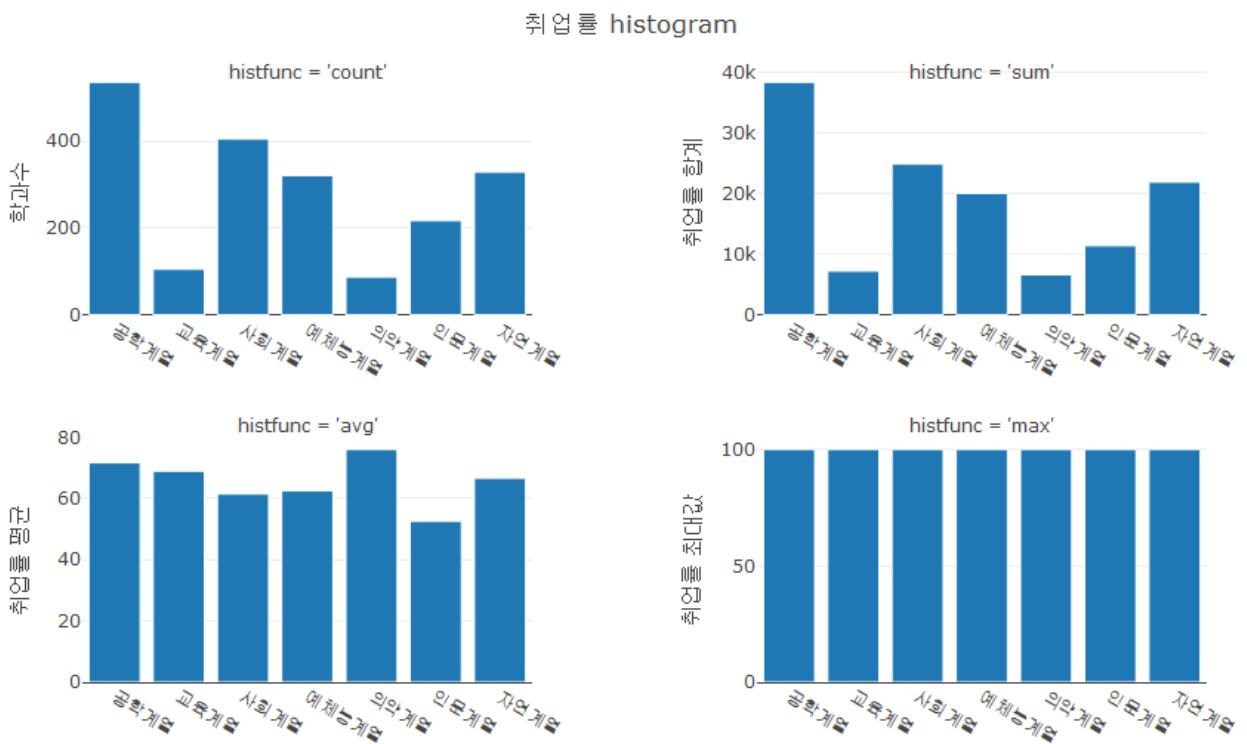
통계 처리되어 히스토그램이 생성된다. 하나 주의해야 할 것은 `histfunc`에 사용되는 'sum', 'avg', 'max', 'min'을 사용할 경우 해당 통계 변환에 사용할 변수열을 `y type`에 설정해야하고 이 변수열은 수치형 데이터가 아닌 문자형 데이터로 설정해야 한다.

```
p_histogram |>
  add_trace(type = 'histogram', x = ~대계열,
             ## stroke 를 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),
             ## stroke 를 흰색으로, 히그토그램 막대 값을 'sum'으로 설정
             histfunc = 'sum', stroke = I('white')) |>
  ## Y 축을 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),
             ## stroke 를 흰색으로, 히그토그램 막대 값을 'average'로 설정
             histfunc = 'avg', stroke = I('white')) |>
  ## Y 축을 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),
             ## stroke 를 흰색으로, 히그토그램 막대 값을 'max'로 설정
             histfunc = 'max', stroke = I('white')) |>
  ## Y 축을 선형으로 설정
  layout(yaxis=list(type='linear', title = list(text = '취업률 최대값')),
         title = '취업률 histogram', margin = margins)
```



실행결과3- 30. *histfunc*에 따른 히스토그램 결과

#### 2.2.2.1.3. *histnorm*

**histnorm**은 히스토그램 trace로 표시되는 데이터의 정규화(normalization)에 대한 속성이다.

**histnorm**으로 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 나타낸다. '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 1 이 된다.

```
p_histogram |>
  add_trace(type = 'histogram', x = ~대계열, y = ~as.character(취업률),
             ## histnorm 을 '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),
             ## histnorm 을 'probability'로 설정
             histnorm = 'probability', stroke = I('white')) |>
```

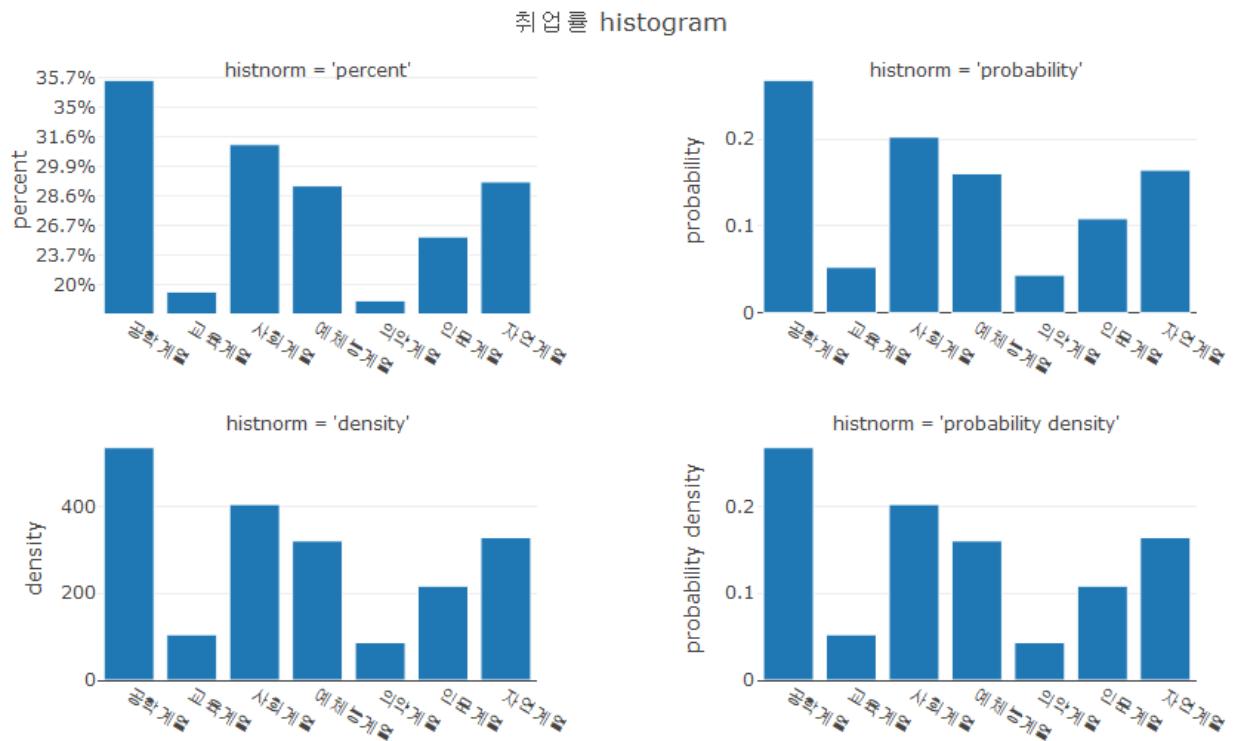
```

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),
          ## histnorm 을 '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),
          ## histnorm 을 '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 3-31. histnorm에 따른 히스토그램 결과

### 2.2.2.2. 스캐터(scatter) trace : 산점도, 선 그래프

스캐터 trace 는 점, 선, 문자를 X, Y 축 좌표의 위치를 사용해 시각화를 하는 형태의 trace 를 모두 통칭한다. 따라서 스캐터 trace 를 통해 생성 가능한 시각화는 산점도, 선 그래프, 텍스트 차트, 풍선 차트 등이다. 이 스캐터 trace 는 `add_trace()`의 `type` 속성에 'scatter'를 설정함으로써 사용할 수 있고 `add_*`() 함수(`add_markers()`, `add_lines()`, `add_paths()`, `add_segments()`, `add_rubbons()`)를 사용할 수 있다.

스캐터 trace 에서 약 50 여개가 넘는 속성을 사용하여 설정이 가능하다. 이 50 여개의 속성 중에는 다시 세부 속성을 설정하는 속성들이 있고 총 4 단계까지 내려가는 속성도 존재한다. 세부 하위 type 은 `list()`를 사용하여 세부 타입을 설정할 수 있다. 이 모든 type 을 다 설명할 수는 없기 때문에 주요한 type 을 위주로 설명한다. 전체 type 은 [plotly.com](https://plotly.com) 에 수록되어 있다.<sup>4</sup> 스캐터 trace 에서 사용하는 속성 중 주요 속성은 다음과 같다.

---

<sup>4</sup> <https://plotly.com/r/reference/scatter/>

속성	설명	속성값	세부 속성
type	trace type 설정	'scatter'	
visible	trace 의 표시 설정	TRUE, FALSE, 'legendonly'	
mode	스캐터 trace 의 모드 설정	'lines', 'markers', 'text'의 조합('+')	
x0, y0	x, y 의 대체로 X, Y 축의 시작점(x0, y0)	수치나 X 축에 매핑될 이산형 카테고리 변수	
dx, dy	x, y 의 대체로 X, Y 축의 변화량(dx, dy)	수치	
xperiod, yperiod	축이 'date' 타입인 경우 기간 위치를 설정	수치나 X 축에 매핑될 이산형 카테고리 변수	
marker	mode 가 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	mode 가 선(lines)인 경우 선의 세부 설정	세부 속성들의 리스트	color, dash, shape, simplify, smoothing, width
textfont	mode 가 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	

#### 2.2.2.2.1. mode

`mode` 는 스캐터 trace 에서 실제 데이터를 표현하는 방법을 지정하는 type 이다. 사실 `add_trace()` 에서 `type` 과 `mode` 는 시각화의 종류를 지정하는 가장 중요한 type 설정이다. 스캐터 trace 에서의 `mode` 는 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 100 일간 코로나19 데이터로 plotly 객체 생성
p_line_wide <- covid19_df_100_wide |> plot_ly()

p_line_wide |>
  ## type 을 scatter, mode 를 markers 로 설정
  add_trace(type = 'scatter', mode = 'markers',
             ## X 축은 data, Y 축은 확진자_한국으로 매팅
             x = ~date, y = ~확진자_한국) |>

  ## 제목, Y 축 제목, 여백 설정
  layout(title = '최근 100 일간 코로나 19 확진자수',
         yaxis = list(title = list(text = '확진자수')),
         margin = margins)

p_line_wide |>
  ## type 을 scatter, mode 를 markers+lines 로 설정
  add_trace(type = 'scatter', mode = 'markers+lines',
             x = ~date, y = ~확진자_한국) |>

  layout(title = '최근 100 일간 코로나 19 확진자수',
         yaxis = list(title = list(text = '확진자수')),
         margin = margins)

p_line_wide |>
  ## type 을 scatter, mode 를 markers+text 로 설정
  add_trace(type = 'scatter', mode = 'markers+text',
             x = ~date, y = ~확진자_한국, text = ~확진자_한국) |>

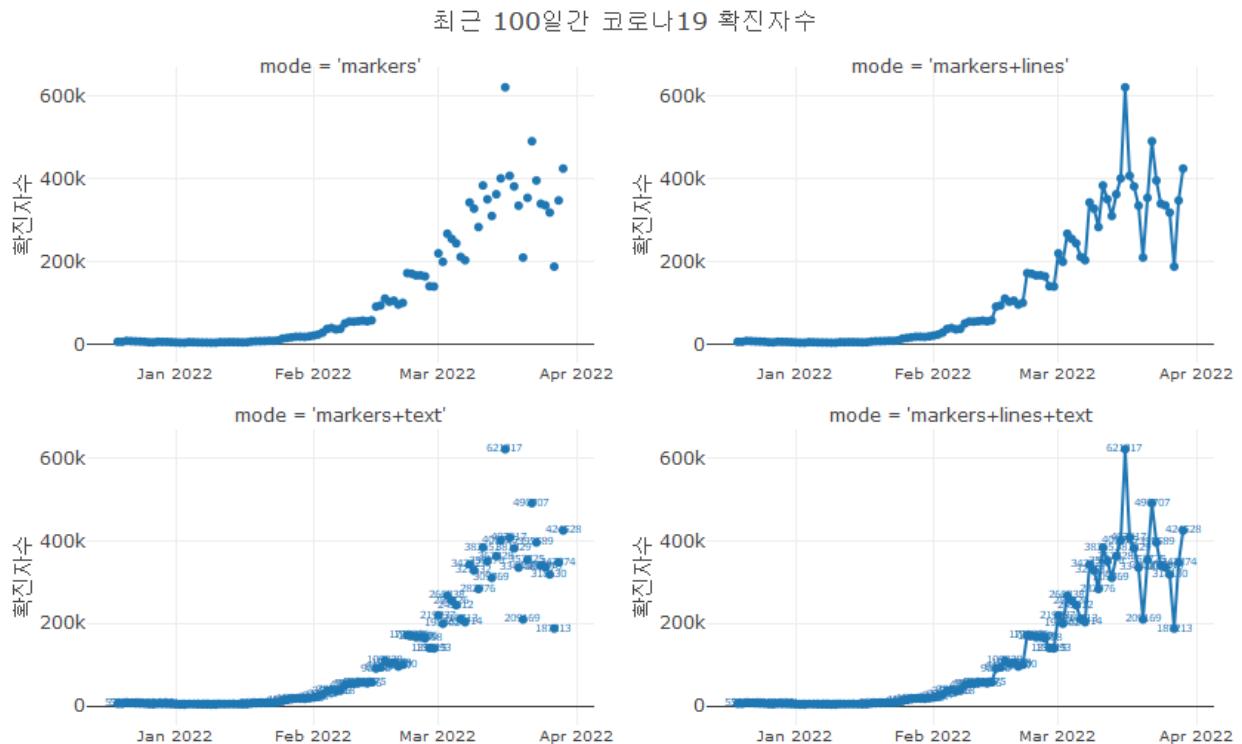
  layout(title = '최근 100 일간 코로나 19 확진자수',
         yaxis = list(title = list(text = '확진자수')),
```

```

margin = margins)

p_line_wide |>
## type 을 scatter, mode 를 markers+lines+text 로 설정
add_trace(type = 'scatter', mode = 'markers+lines+text',
          x = ~date, y = ~확진자_한국, text = ~확진자_한국) |>
layout(title = '최근 100 일간 코로나19 확진자수',
       yaxis = list(title = list(text = '확진자수')),
       margin = margins)

```



실행결과3- 32. 스캐터 trace 의 mode 설정 결과

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

```

## 넓은 형태의 데이터로 초기화한 plotly 객체로
p_line_wide |>
## type 을 scatter, mode 를 markers+lines 로 설정
add_trace(type = 'scatter', mode = 'markers+lines',
          ## X 축은 date, Y 축은 확진자_한국으로 맵핑
          x = ~date, y = ~확진자_한국) |>

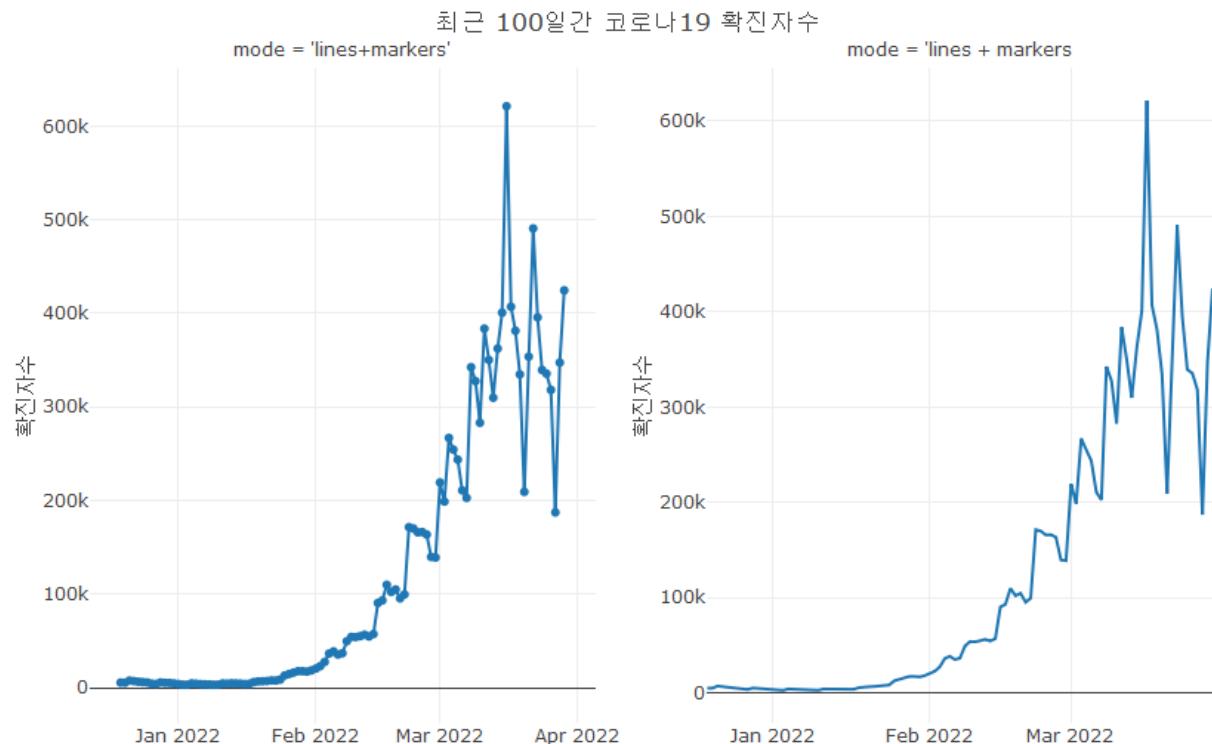
```

```

## 제목, X, Y 축 제목, 여백 설정
layout(title = '최근 100 일간 코로나 19 확진자수',
       xaxis = list(title = list(text = NULL)),
       yaxis = list(title = list(text = '확진자수'))),
margin = margins)

p_line_wide |>
## type 을 scatter, mode 를 markers + lines 로 설정(공백포함)
add_trace(type = 'scatter', mode = 'markers + lines',
          x = ~date, y = ~확진자_한국) |>
layout(title = '최근 100 일간 코로나 19 확진자수',
       xaxis = list(title = list(text = NULL)),
       yaxis = list(title = list(text = '확진자수'))),
margin = margins)

```



실행결과3-33. 공백이 포함된 mode 설정 결과

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

스캐터 trace 의 **marker** 타입은 스캐터 trace 에서 데이터를 표현하는 점의 속성을 설정하는 속성이다. 이 속성은 `add_trace(type = 'scatter', mode = 'marker')`를 사용하거나 `add_markers()`를 사용한다.

```

add_trace(p, type = 'scatter', 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 : marker 의 type 설정을 위한 매개변수 list 설정
- ... : 스캐터 trace 의 markers 모드에 설정할 수 있는 속성 설정
- data : 시각화할 데이터프레임
- inherit : plot_ly()에 설정된 속성 type 을 상속할지를 결정하는 논리값
- x : X 축에 매핑할 변수를 ~로 설정
- y : Y 축에 매핑할 변수를 ~로 설정
- z : Z 축에 매핑할 변수를 ~로 설정

```

`add_trace()`의 `marker` 는 `list()`를 사용하여 세부 설정 속성 리스트를 만들어서 설정한다. 주의할 것은 `marker` 의 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 주의해야 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 `marker`로 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.

```

## 취업통계 데이터를 사용해 plotly 객체 생성
p_marker <- df_취업통계_2000 |>
  plot_ly()

p_marker |>
  ## type 을 scatter, mode 를 markers 로 설정
  add_trace(type = 'scatter', mode = 'markers',
            ## X 축은 졸업자수, Y 축은 취업자수로 매핑
            x = ~졸업자수, y = ~취업자수,
            ## marker 의 color 를 대체열로 매핑하지만 설정되지 않음

```

```

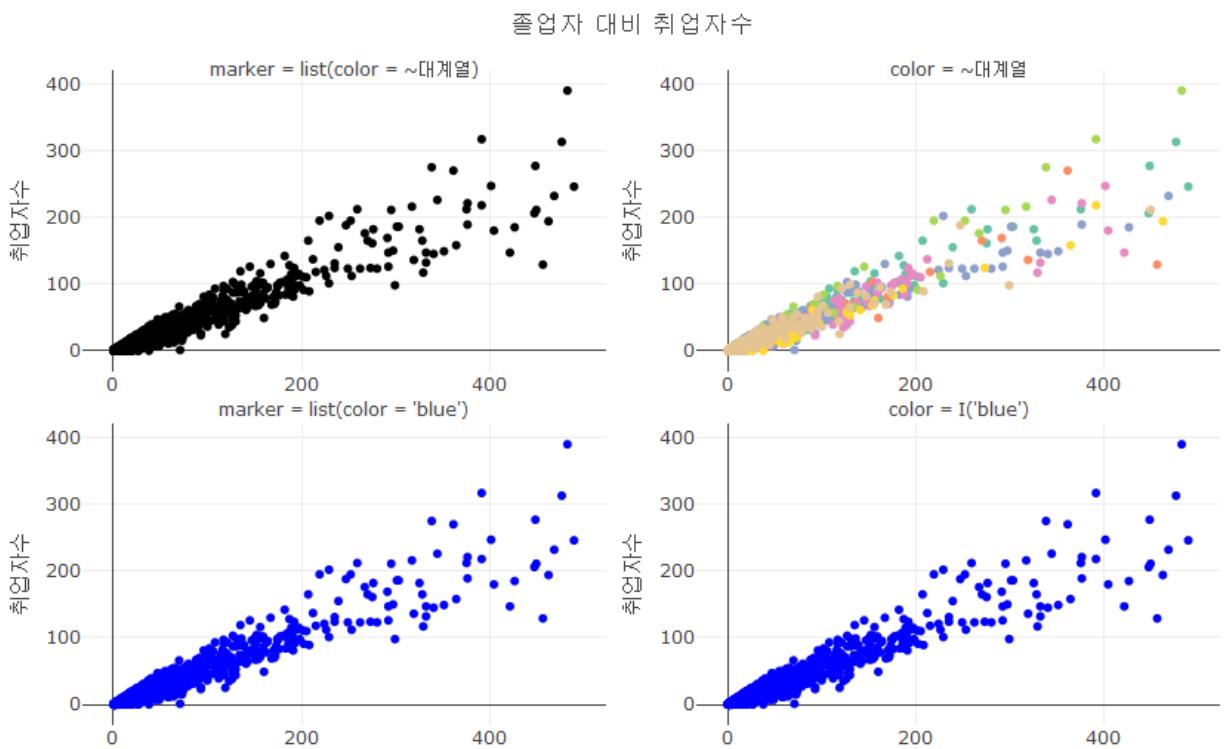
marker = list(color = ~대계열)) |>
layout(title = '졸업자 대비 취업자수', margin = margins)

p_marker |>
add_trace(type = 'scatter', mode = 'markers',
x = ~졸업자수, y = ~취업자수,
## trace 의 color 를 대계열로 매팅
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
## marker 의 color 를 '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
## marker 의 color 를 I()를 사용하여 'darkblue'로 설정
color = I('darkblue')) |>
layout(title = '졸업자 대비 취업자수', margin = margins)

```



실행결과 3-34. 스캐터 trace의 여러가지 color 설정 결과

`marker`에서 설정 가능한 세부 속성은 20 여개가 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.

`marker`의 설정에서 하나 중요한 속성은 `colorbar`이다. `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 `colorbar`이다.

컬러바 설정에 사용되는 세부 속성은 현재(22.03) 43 개가 제공된다. 이에 대한 세부 속성과 설명은 `plotly` 매뉴얼을 참조하라.<sup>5</sup>

```
df_취업통계_2000 |>
  plot_ly(x = ~졸업자수, y = ~취업자수,
```

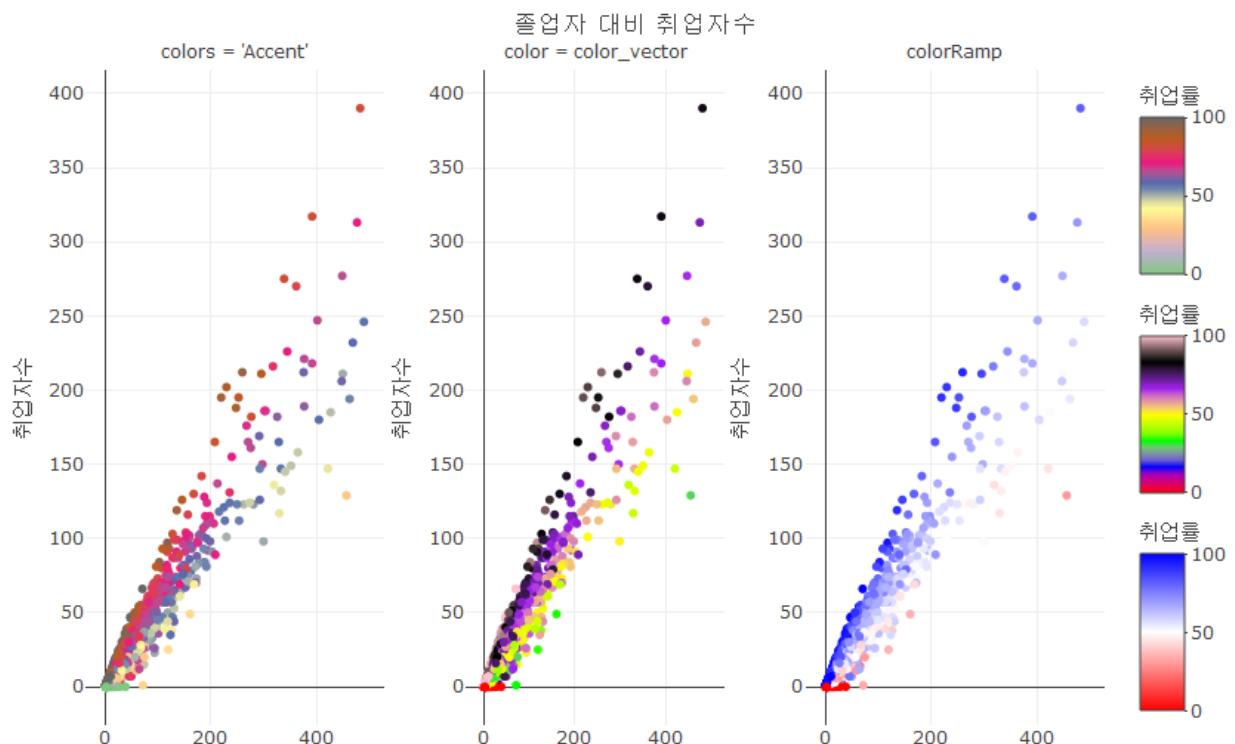
<sup>5</sup> <https://plotly.com/r/reference/#scatter-marker-colorbar>

```
## color 를 연속형 수치 변수에 맵핑
color = ~취업률,
## colors 를 Rcolorbrewer 의 'Accent' 팔레트로 설정
colors = 'Accent') |>
layout(title = '졸업자 대비 취업자수', margin = margins)

## 컬러바 설정을 위한 벡터 설정
color_vector <- c('red', 'blue', 'green', 'yellow', 'purple', 'black', 'pink')

df_취업통계_2000 |>
plot_ly(x = ~졸업자수, y = ~취업자수,
        color = ~취업률,
        ## colors 를 컬러 벡터로 설정
        colors = color_vector) |>
layout(title = '졸업자 대비 취업자수', margin = margins)

df_취업통계_2000 |>
plot_ly(x = ~졸업자수, y = ~취업자수,
        color = ~취업률,
        ## colors 를 colorRamp()를 사용하여 설정
        colors = colorRamp(c('red', 'white', 'blue'))) |>
layout(title = '졸업자 대비 취업자수', margin = margins)
```



실행결과 3-35. 스캐터 trace 의 colorbar 의 설정 결과

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

스캐터 trace 의 `line` type 은 스캐터 trace 에서 데이터를 이어주는 선 그래프를 생성하는 스캐터 trace type 이다. `line` type 은 `marker` 와 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 list 에 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 = 'scatter', 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 : `line` 의 type 설정을 위한 매개변수 list 설정

- ... : 스캐터 trace 의 `line` 모드에 설정할 수 있는 속성 설정

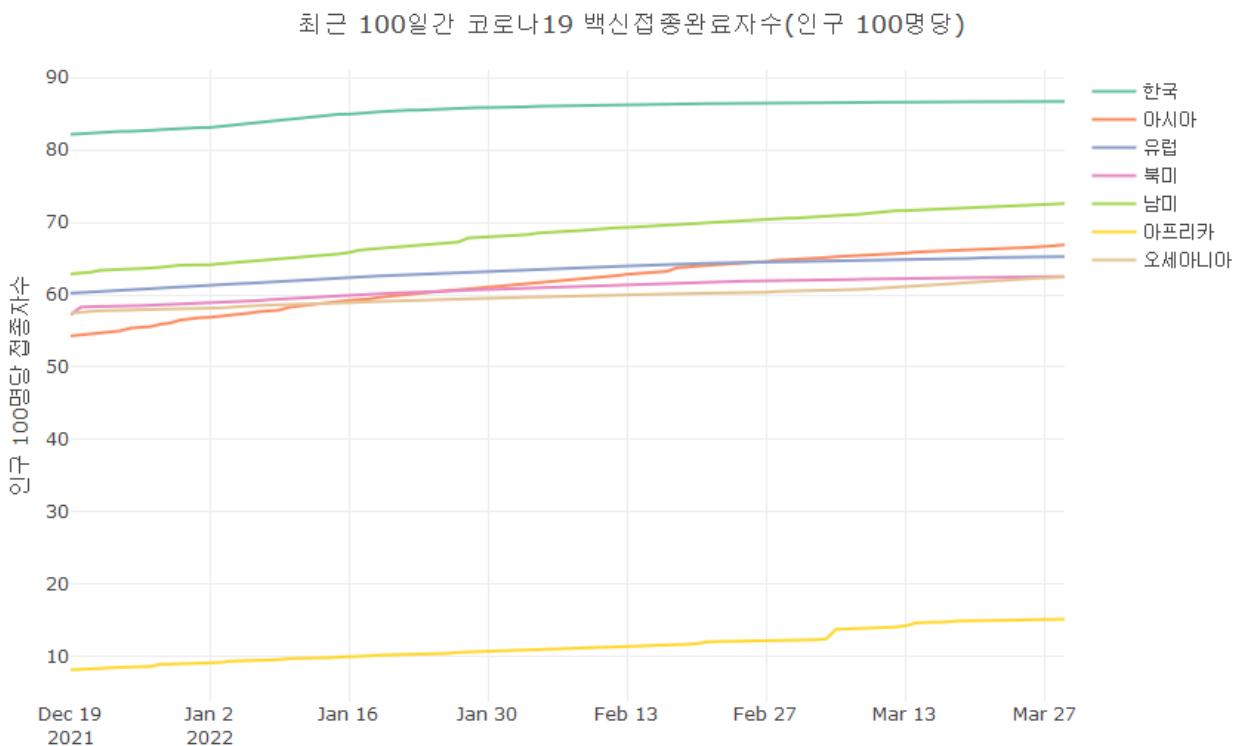
- data : 시각화할 데이터프레임
- inherit : plot\_ly()에 설정된 속성 type 을 상속할지를 결정하는 논리값
- x : X 축에 매핑할 변수를 ~로 설정
- y : Y 축에 매핑할 변수를 ~로 설정
- z : Z 축에 매핑할 변수를 ~로 설정

trace 의 `type` 과 `mode` 를 선 그래프에 맞게 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`, `linetype` 으로 매핑된 변수를 기준으로 그룹화하여 선을 연결한다.

```
## 긴 형태의 100 일 코로나19 데이터로 plotly 객체 생성
p_line <- covid19_df_100 |> plot_ly()

p_line |>
  ## type 을 scatter, mode 를 lines 로 설정한 trace 추가
  add_trace(type = 'scatter', mode = 'lines',
             ## X 축을 date,
             ## Y 축을 people_fully_vaccinated_per_hundred 로 매핑
             x = ~date, y = ~people_fully_vaccinated_per_hundred,
             ## color 를 location 으로 매핑
             color = ~ location) |>

  ## 제목, X, Y 축 제목, 여백 설정
  layout(title = '최근 100 일간 코로나 19 백신접종완료자수(인구 100 명당)',
         xaxis = list(title = list(text = NULL)),
         yaxis = list(title = '인구 100 명당 접종자수'), margin = margins)
```



### 실행결과 3-36. scatter trace line 모드의 color 설정 결과

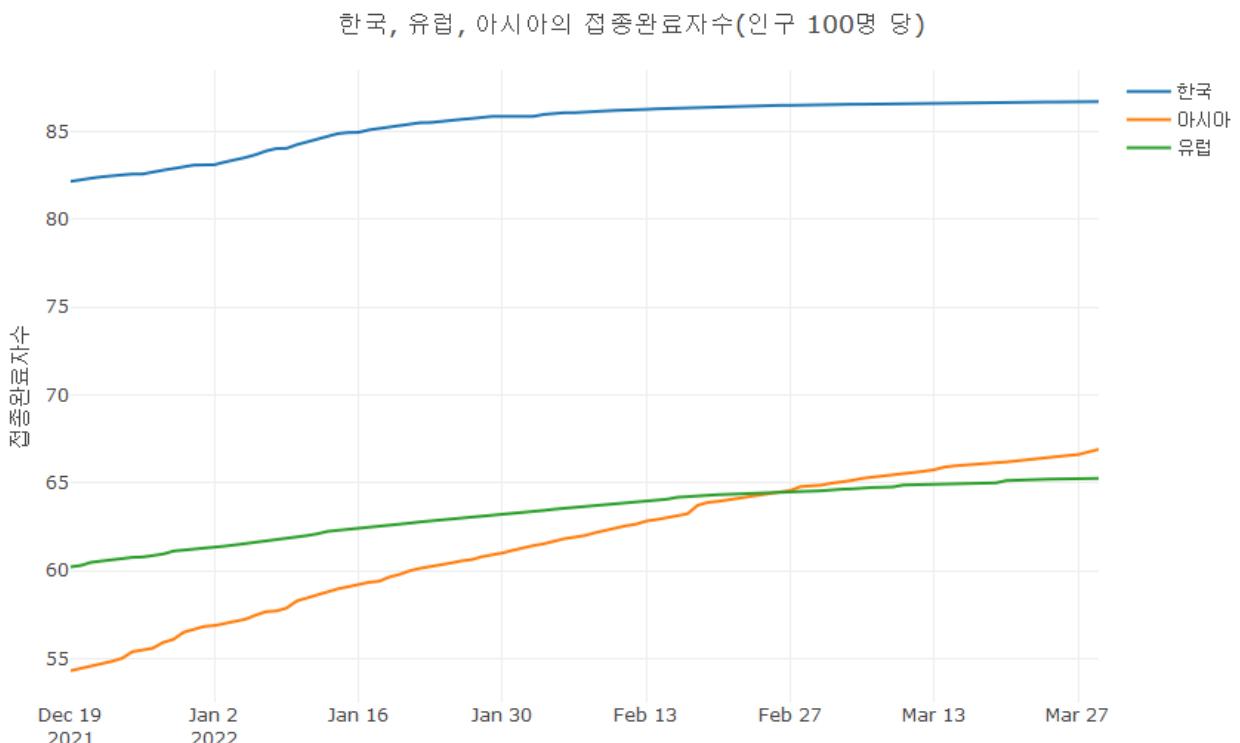
선 그래프의 경우는 선을 너무 많이 그려서 꼬이기 시작하면 오히려 데이터를 확인하는데 방해가 되는 시각화가 되는 경우가 있다. 이를 스파게티 선 그래프라고 하는데 하나의 선 그래프에는 보통 5~6 개 이상의 선을 넣지 않는 것이 일반적이다. 따라서 선 그래프의 경우는 변수 매핑에 의해 단번에 선 그래프를 생성하는 방법도 많이 사용하지만 긴 형태의 데이터프레임으로 변환하여야 하고 매핑된 변수의 카테고리가 너무 많으면 스파게티 선 그래프가 생성된다. 따라서 사용자가 그리기를 원하는 3~4 개의 단일 선 그래프를 겹쳐 그리는 방법도 많이 사용된다. 특히 이 방법은 사람이 이해하기 쉬운 넓은 형태의 데이터프레임을 직접 사용할 수 있다.

```
## 넓은 형태의 100 일 코로나19 데이터로 x 축을 date 로 매핑한 plotly 객체 생성
covid19_df_100_wide |> plot_ly(x = ~date) |>
  ## type 을 scatter, mode 를 lines 로 설정한 trace 추가
  add_trace(type = 'scatter', mode = 'lines',
            ## Y 축을 백신접종완료자_한국으로 매핑, name 을 한국으로 설정
            ## X 축은 초기 생성된 plotly 객체에 이미 매핑됨
            y = ~백신접종완료자_한국, name = '한국') |>
  ## type 을 scatter, mode 를 lines 로 설정한 trace 추가
  add_trace(type = 'scatter', mode = 'lines',
```

```

## Y 축을 백신접종완료자_아시아로 매팅, name 을 아시아로 설정
y = ~백신접종완료자_아시아, name = '아시아') |>
add_trace(type = 'scatter', mode = 'lines',
          ## Y 축을 백신접종완료자_유럽로 매팅, name 을 유럽로 설정
          y = ~백신접종완료자_유럽, name = '유럽') |>
## 제목, X, Y 축 제목, 여백 설정
layout(title = '한국, 유럽, 아시아의 접종완료자수(인구 100 명 당)',
       xaxis = list(title = list(text = NULL)),
       yaxis = list(title = list(text = '접종완료자수'))),
       margin = margins)

```



### 실행결과 3- 37. trace 를 추가하는 plotly 생성 방법

이렇게 trace 를 추가하는 방법으로 시각화를 완성하면 몇 가지 장점이 있다.

첫 번째 장점은 특별한 작업 없이 범례의 순서를 결정할 수 있다. 최종적으로 나타나는 선 그래프의 범례는 trace 가 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 trace 를 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() |>
  ## covid19_df 의 아시아 데이터만 필터링한 데이터를 사용하는 trace 추가
  add_trace(data = covid19_df |> filter(continent == 'Asia'),
             ## type 을 scatter, mode 를 lines 로 설정
             type = 'scatter', mode = 'lines',
             ## X, Y 축의 변수 매핑
             x = ~date, y = ~people_fully_vaccinated_per_hundred,
             ## name 의 변수 매핑(아래의 showLegend=F 로 의미없는 설정)
             name = ~iso_code,
             ## line 의 세부 속성 설정
             line = list(color = 'lightgray'),
             ## OI trace 의 범례는 제거
             showlegend = F,
             ## 결측치에 의한 갭은 연결
             connectgaps = TRUE)
## 최종 plotly 결과는 fig 로 저장

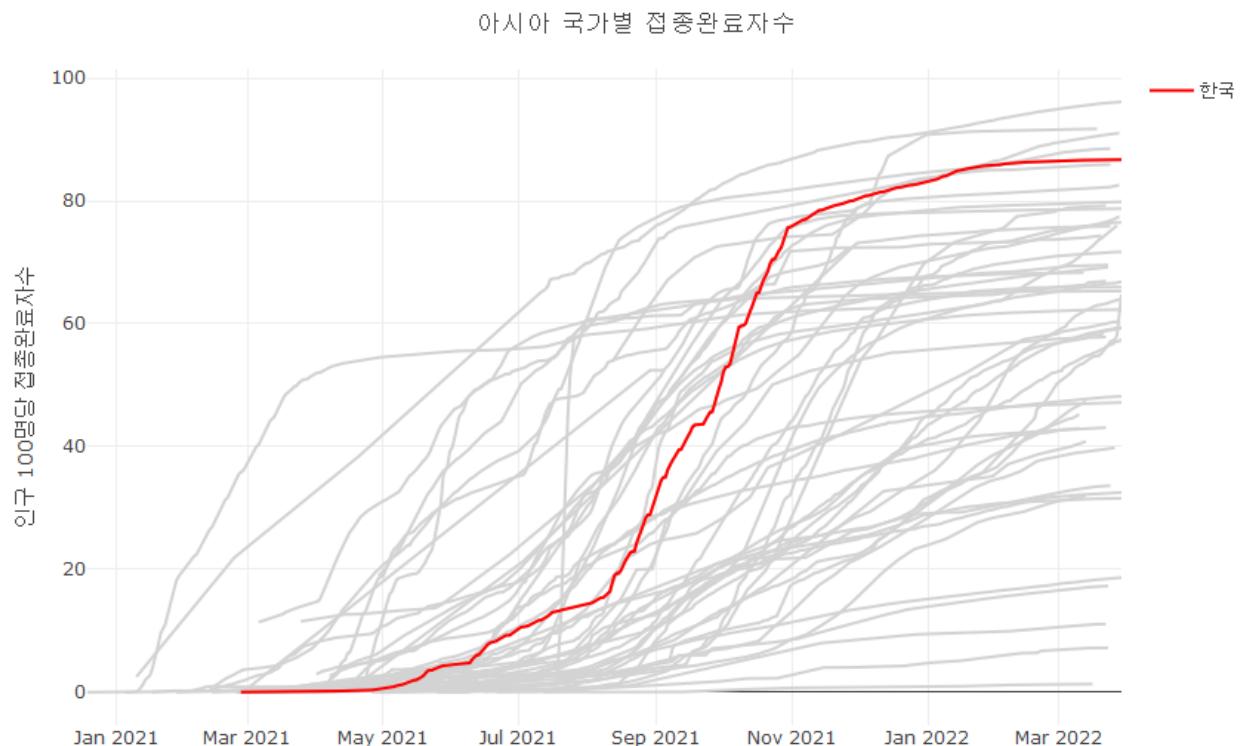
fig <- fig |>    ## 앞서 생성한 fig 를 사용하여
  ## covid19_df 의 한국 데이터만 필터링한 데이터를 사용하는 trace 추가
  add_trace(data = covid19_df |> filter(iso_code == 'KOR'),
             ## type 을 scatter, mode 를 lines 로 설정
             type = 'scatter', mode = 'lines',
             ## X, Y 축의 변수 매핑
             x = ~date, y = ~people_fully_vaccinated_per_hundred,
             ## name 의 설정
             name = '한국',
             ## line 의 세부 속성 설정
             line = list(color = 'red'),
```

```

## 0) trace 의 범례는 표시
showlegend = T)
## 최종 plotly 결과는 fig 로 저장

fig |> ## 앞서 생성한 fig 를 사용하여
## 제목, X, Y 축 제목, 범례 표시, 여백 설정
layout(title = list(text = '아시아 국가별 접종완료자수'),
       xaxis = list(title = list(text = NULL)),
       yaxis = list(title = '인구 100 명당 접종완료자수'),
       showlegend = T, margin = margins)

```



### 실행결과 3-38. 스파게티 선 그래프의 특정 trace 강조

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

스캐터 trace 의 **text** trace 은 각 X 축과 Y 축에 매핑된 좌표에 표시될 문자열을 설정하는 trace 이다. 0) trace 는 `add_trace(type = 'scatter', mode = 'text')`나 `add_text()`를 사용하여 추가할 수 있다.

```

add_trace(p, type = 'scatter', 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 : X 축에 매핑할 변수를 ~로 설정
- y : Y 축에 매핑할 변수를 ~로 설정
- z : Z 축에 매핑할 변수를 ~로 설정

`text`에 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 `hoverinfo`에 `text`가 설정되고 `hovertext`가 설정되지 않은 경우에는 `text`에 설정된 문자열이 호버레이블로 사용된다.

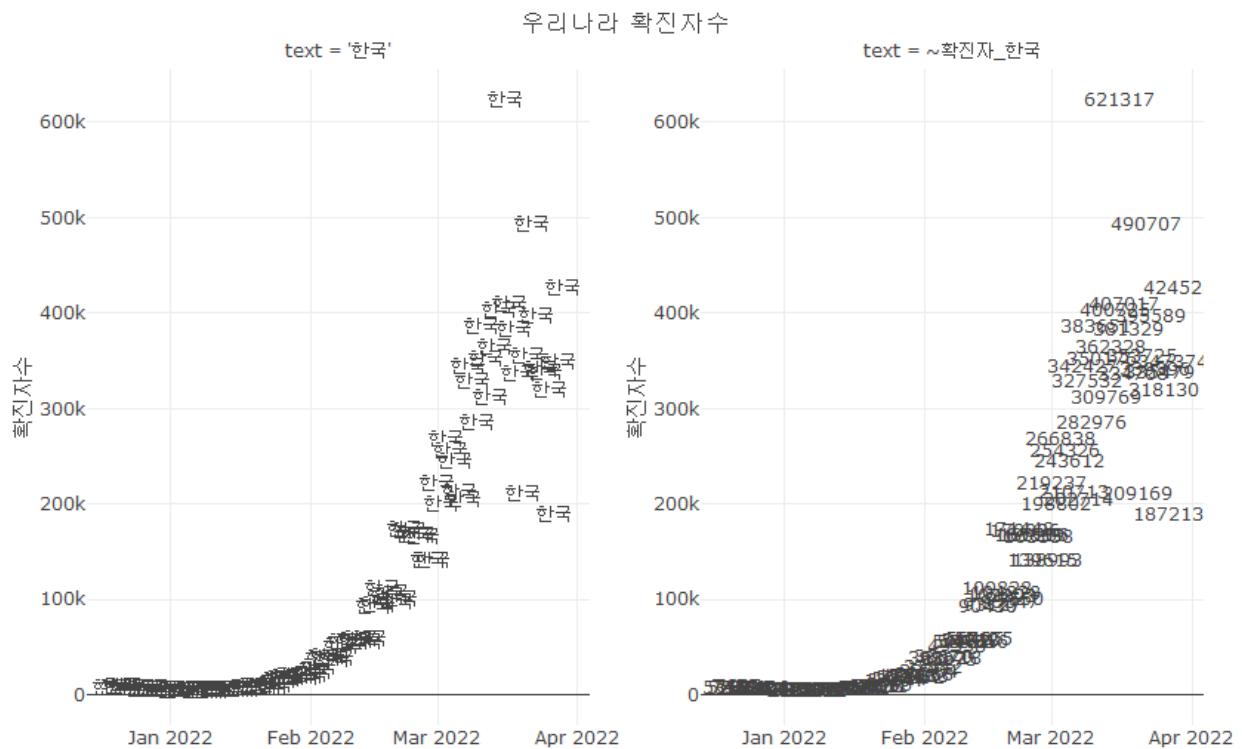
```
## 앞서 생성한 p_line_wide 객체에 trace 추가
add_trace(p_line_wide,
          ## type 을 scatter, mode 를 text로 설정
          type = 'scatter', mode = 'text',
          ## X, Y 축 변수 매핑
          x = ~date, y = ~확진자_한국,
          ## text 매핑
          text = '한국')|>
layout(title = list(text = '우리나라 확진자수'),
       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,
          ## type 을 scatter, mode 를 text로 설정
          type = 'scatter', mode = 'text',
```

```

## X, Y 축 변수 매팅
x = ~date, y = ~확진자_한국,
## text 매팅!
text = ~확진자_한국) |>
layout(title = list(text = '확진자수'),
      xaxis = list(title = list(text = NULL)),
      yaxis = list(title = '확진자수'),
      margin = margins)

```



### 실행결과 3- 39. text 매팅과 설정 결과

#### 2.2.2.3. 막대(bar) trace : 막대 그래프

막대 trace 는 데이터 변량의 크기를 막대를 사용하여 시각화를 하는 형태의 trace 를 말한다. 따라서 막대 trace 를 통해 생성 가능한 시각화는 막대 그래프이다. 이 막대 trace 는 `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 : X 축에 매핑할 변수를 ~로 설정
- y : Y 축에 매핑할 변수를 ~로 설정

막대 trace에서 설정이 가능한 주요 속성은 다음과 같다.

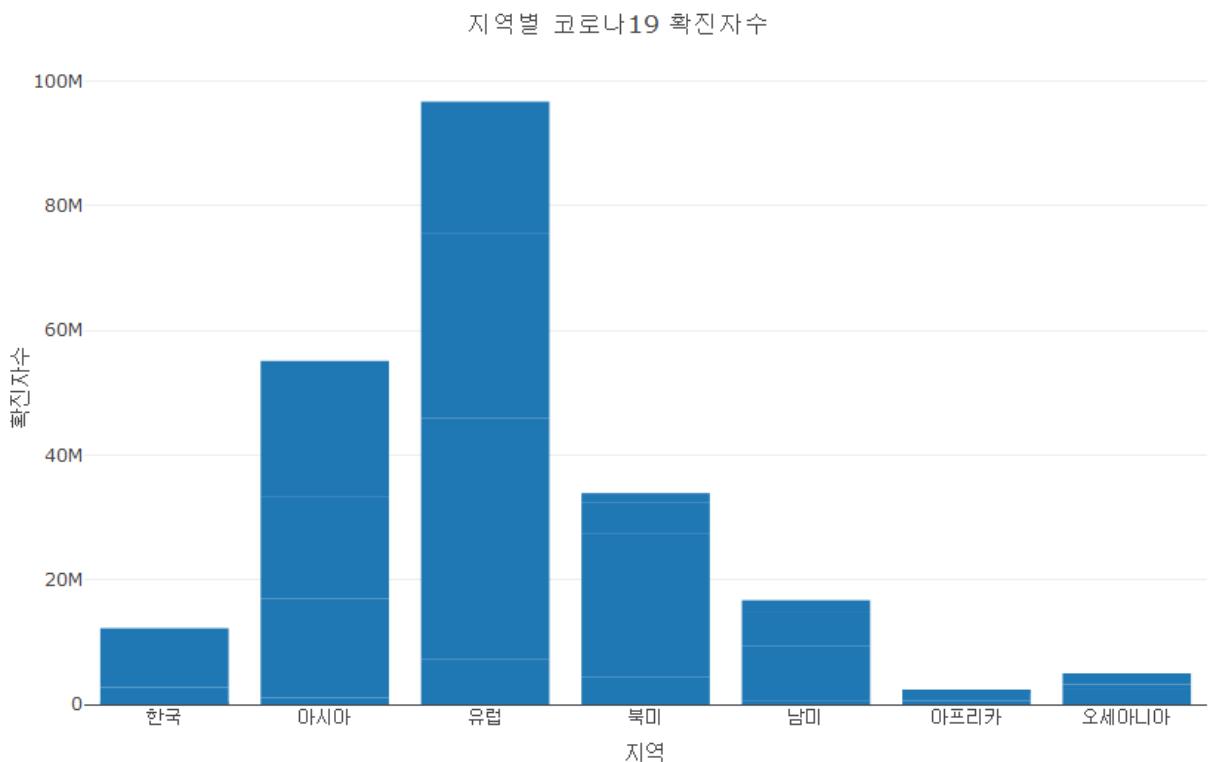
속성	설명	속성 값	세부속성
type	trace type 설정	'bar'	
visible	trace 의 표시 설정	TRUE, FALSE, 'legendonly'	
x0, y0	x, y 의 대체로 X, Y 축의 시작점(x0, y0)	수치나 X 축에 매핑될 이산형 카테고리 변수	
dx, dy	x, y 의 대체로 X, Y 축의 변화량(dx, dy)	수치	
base	막대 base 의 위치 설정	수치나 X 축에 매핑될 이산형 카테고리 변수	
width	막대의 너비 설정	0 이상의 수치값이나 수치 벡터	
offset	막대 위치의 이동값 설정	수치나 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 X 축에 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, ...

속성	설명	속성 값	세부속성
textfont	mode 가 텍스트(text)인 경우 문자의 세부 설정	세부 속성들의 리스트	color, family, size

앞서 설명한 바와 같이 막대 그래프(`add_bars()`)는 X 축과 Y 축의 이변량을 필요로 하는 시각화이고 히스토그램(`add_histogram()`)은 X 축에 따른 사례수를 구간(bin)에 따라 막대로 표시한 시각화이다.

```
## 최종 결과는 p_bar 에 저장
p_bar <-
  ## 긴 형태의 100 일 코로나19 데이터에서
  covid19_df_100 |>
  ## date 에서 연월을 추출한 yearmonth 열을 생성
  mutate(yearmonth = tsibble::yearmonth(date)) |>
  ## Location 과 yearmonth 로 그룹화
  group_by(location, yearmonth) |>
  ## new_cases 의 합계 산출
  summarise(new_cases = sum(new_cases)) |>
  ## plotly 객체 생성
  plot_ly()

## p_bar 를 사용
p_bar |>
  ## 막대 타입 trace 를 추가
  add_trace(type = 'bar',
            ## X, Y 축에 변수 매핑
            x = ~location, y = ~new_cases) |>
  ## 제목, X, Y 축 제목, 여백 설정
  layout(title = list(text = '지역별 코로나 19 확진자수'),
         xaxis = list(title = list(text = '지역')),
         yaxis = list(title = '확진자수'),
         margin = margins)
```



### 실행결과 3- 40. 막대 trace plotly 생성

#### 2.2.2.3.1. *barmode*

막대 그래프는 일반적으로 세 가지 형태로 생성된다. `ggplot2` 에서는 `dodge`, `stack`, `fill` 로 불리며 `position` 을 셋중에 하나로 설정함으로써 자동적으로 그려지지만 `plotly` 에서 막대 그래프를 생성하는 `add_trace()`나 `add_bars()`로는 이 세 가지 막대 그래프를 구분하여 생성할 수 없다.

이와 같이 여러개의 막대 trace 를 겹쳐 그릴때 그 구성 형태를 결정하기 위해서는 차후 설명할 `layout()`의 `barmode` 속성을 설정함으로써 가능하다. `barmode` 는 '`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 방식이므로 `ggplot2` 의 '`dodge`'에 해당한다. '`overlay`'는 막대들이 서로 겹쳐져서 표시되는데 정상적으로 생성하기 위해서는 `opacity` 로 투명도를 조절하여야 한다. '`relative`'는 Y 축을 중간에 두고 양의 값은 위로 음의 값은 아래로 막대가 향하게 하는 막대 그래프이다. `barmode` 의 기본값은 '`group`'이기 때문에 X, Y 축 설정외에 추가적인 그룹화 변수를 매핑하면 기본값인 '`group`'형 막대 그래프가 생성된다.

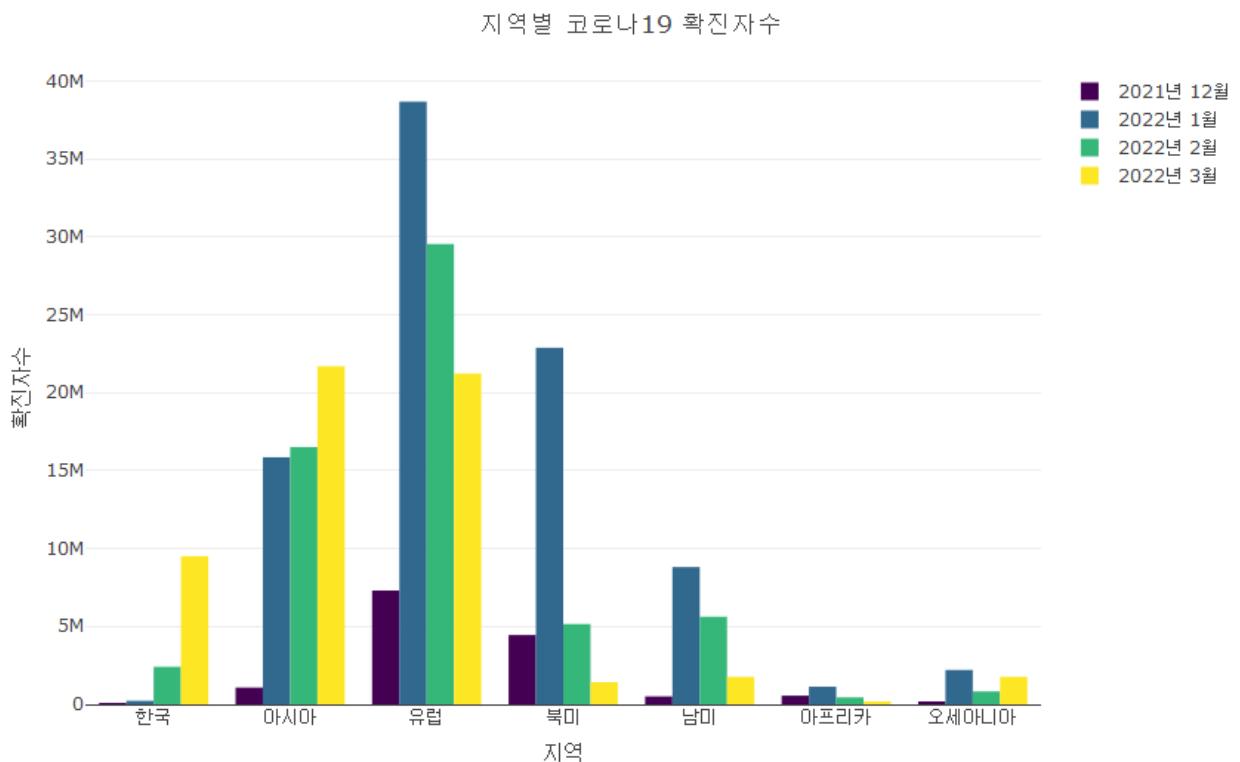
```

## 최종 결과는 p_bar_mode 에 저장
p_bar_mode <-
  ## 긴 형태의 100 일 코로나19 데이터에서
  covid19_df_100 |>
  ## date 에서 연월을 추출한 yearmonth 열을 생성
  mutate(yearmonth = tsibble::yearmonth(date)) |>
  ## Location 과 yearmonth 로 그룹화
  group_by(location, yearmonth) |>
  ## new_cases 의 합계 산출
  summarise(new_cases = sum(new_cases)) |>
  ## plotly 객체 생성
  plot_ly()

## p_bar_mode 를 사용하여
p_bar_mode |>
  ## 막대 trace 추가
  add_trace(type = 'bar',
            ## X, Y 축 변수 매핑
            x = ~location, y = ~new_cases,
            ## color 는 연월 팩터로 매핑
            color = ~as.factor(yearmonth),
            ## colors 는 viridis 팔레트로 설정
            colors = 'viridis',
            ## 범례 표시 이름의 설정
            name = ~paste0(lubridate::year(yearmonth), '년 ',
                           lubridate::month(yearmonth, label = TRUE,
                                         abbr = FALSE))) |>

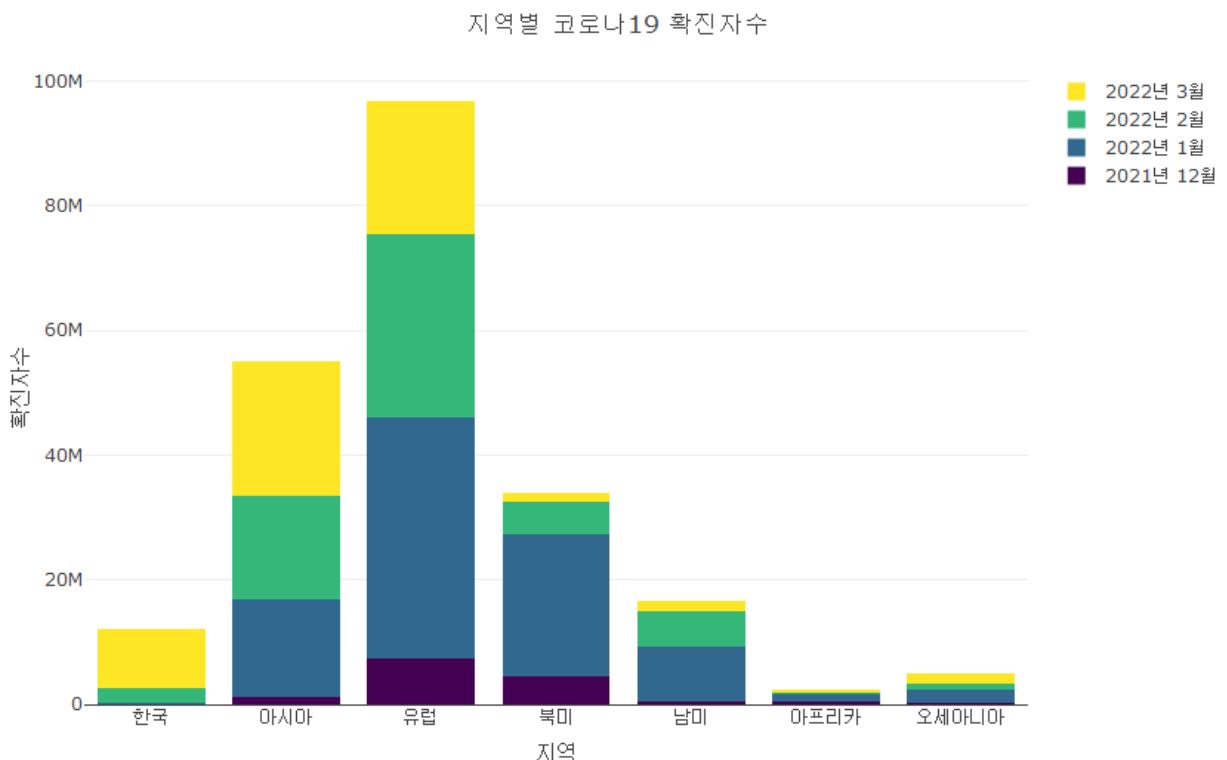
## barmode 를 group 으로 설정
layout(barmode = 'group',
       title = list(text = '지역별 코로나 19 확진자수'),
       xaxis = list(title = list(text = '지역')),
       yaxis = list(title = '확진자수'),
       margin = margins)

```



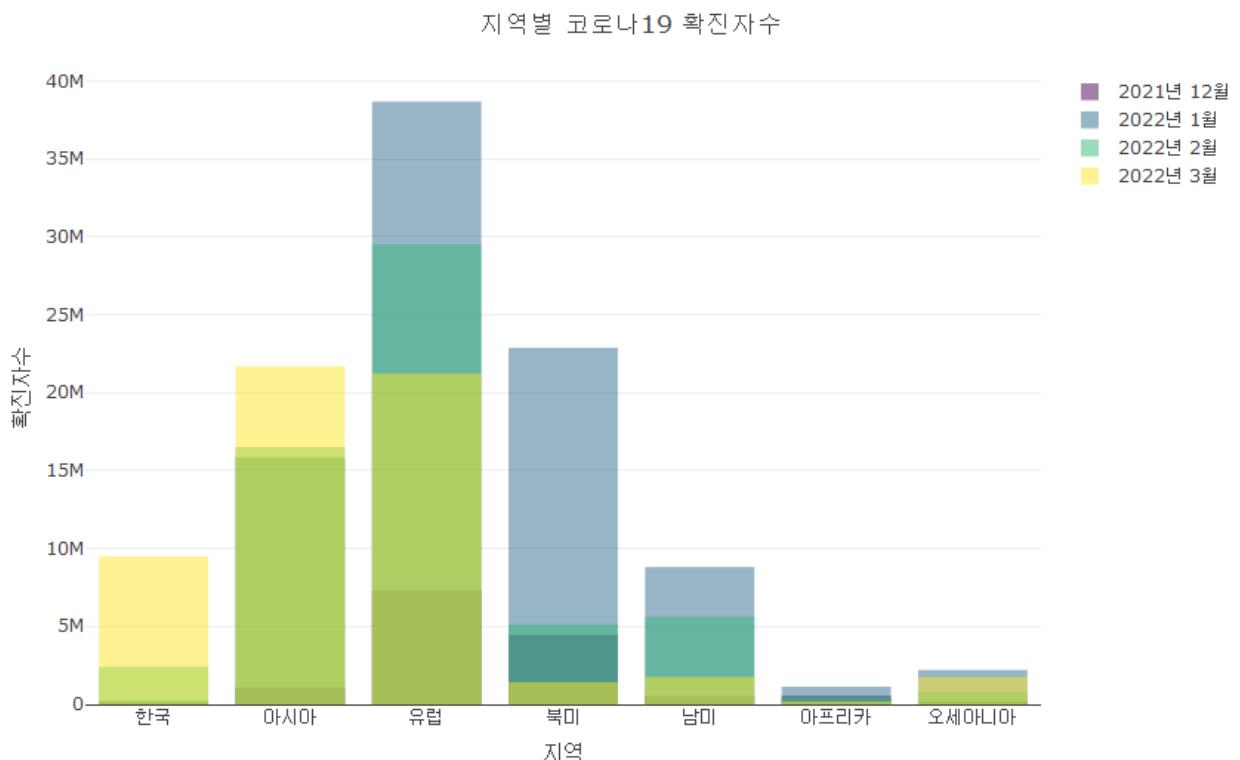
실행결과 3- 41. group 형 막대 trace

```
p_bar_mode |>
  add_trace(type = 'bar', x = ~location, y = ~new_cases,
            color = ~as.factor(yearmonth), colors = 'viridis',
            name = ~paste0(lubridate::year(yearmonth), '년 ',
                           lubridate::month(yearmonth, label = TRUE,
                                             abbr = FALSE))
  )
) |>
## barmode 를 stack 으로 설정
layout(barmode = 'stack',
       title = list(text = '지역별 코로나 19 확진자수'),
       xaxis = list(title = list(text = '지역')),
       yaxis = list(title = '확진자수'),
       margin = margins)
```



실행결과 3-42. stack 형 막대 trace

```
p_bar_mode |>
  add_trace(type = 'bar', x = ~location, y = ~new_cases,
             color = ~as.factor(yearmonth), colors = 'viridis', opacity = 0.5,
             name = ~paste0(lubridate::year(yearmonth), '년 ',
                           lubridate::month(yearmonth, label = TRUE,
                                             abbr = FALSE))
  )
## barmode 를 overlay 으로 설정
layout(barmode = 'overlay',
       title = list(text = '지역별 코로나 19 확진자수'),
       xaxis = list(title = list(text = '지역')),
       yaxis = list(title = '확진자수'),
       margin = margins)
```



실행결과 3- 43. overlay 형 막대 trace

```
## 최종 결과는 diff_sam에 저장
diff_sam <-
  ## 긴 형태의 100 일 코로나19 데이터에서
  covid19_df_100 |>
  ## date의 월 단위 열을 yearmonth에 저장
  mutate(yearmonth = lubridate::floor_date(date, "month")) |>
  ## iso_code, yearmonth로 그룹화
  group_by(iso_code, yearmonth) |>
  ## new_cases 합계 산출
  summarise(new_cases = sum(new_cases)) |>
  ## 지역이름을 열로 값을 new_cases로 설정한 넓은 데이터프레임 생성
  pivot_wider(names_from = iso_code, values_from = new_cases) |>
  ## 남미 열만 선택
  select(OWID_SAM) |>
  ## 선택된 열을 벡터로 변환
  pull()

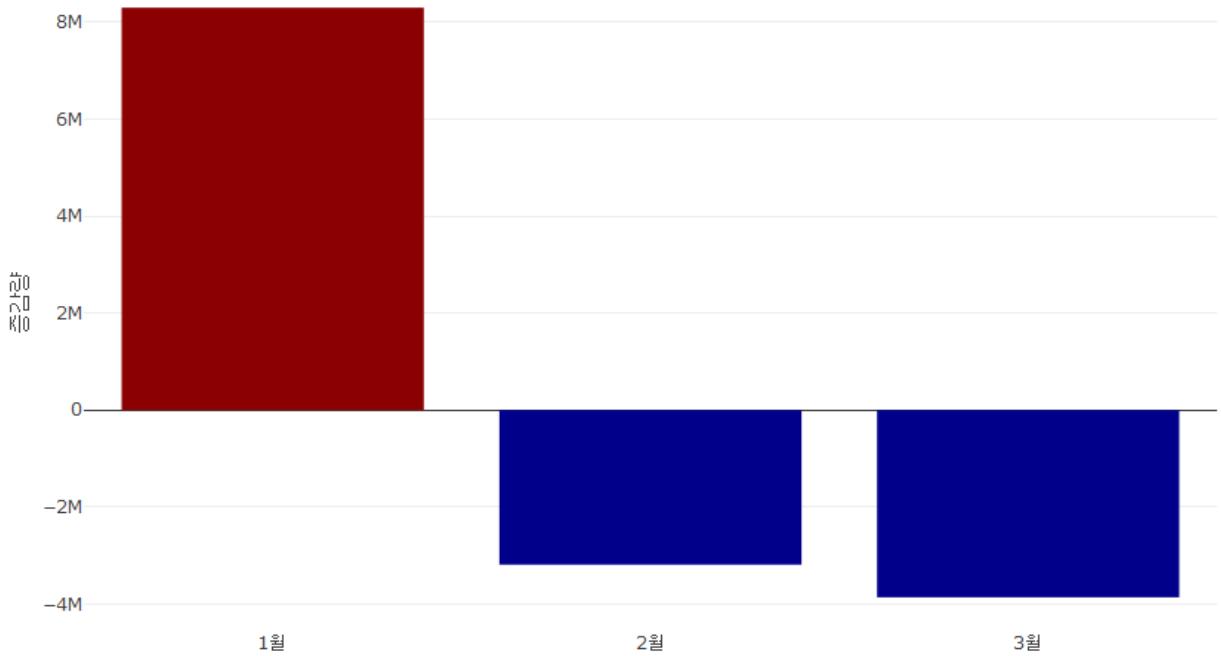
## 빈 plotly 객체 생성
plot_ly() |>
  ## 막대 trace 추가
  add_trace(type = 'bar',
```

```

## X 축은 1 부터 3 까지, Y 축은 전월 대비 차분 산출
x = 1:3, y = diff(diff_sam),
## 차분이 0 보다 크면 'darkred', 아니면 'darkblue'로 설정
color = I(ifelse(diff(diff_sam) > 0, "darkred", "darkblue"))
) |>
## barmode 를 relative 로 설정
layout(barmode = 'relative',
       xaxis = list(ticktext = c('1 월', '2 월', '3 월'),
                     tickvals = c(1, 2, 3)),
       yaxis = list(title = '증감량'),
       title = list(text = '남미 코로나19 확진자수 증감'),
       margin = margins)

```

남미 코로나19 확진자수 증감



### 실행결과 3- 44. relative 형 막대 trace

앞서 스캐터 trace 의 선 그래프를 그릴 때, 넓은 형태의 데이터프레임을 사용하여 추가하고자 하는 열을 각각의 trace 로 하나 하나 추가하여 전체 시각화를 완성하는 예를 보았다. 막대 trace 도 이와 동일한 방법으로 생성할 수 있다. `plotly` 객체에 계속적으로 막대 trace 를 추가한 후 `layout()`의 `barmode` 를 원하는 형태로 설정하면 된다. 이 과정에서 `name` 으로 각각의 범례에 표시될 아이템 이름을 설정할 수 있고 trace 의 추가 순서에 따라 범례의 순서를 설정할

수 있으며, 특정 막대의 색을 강조함으로써 특정 데이터를 강조하는데 유용하게 활용할 수 있다.

```
## 최종 결과는 p_bar_mode1 에 저장
p_bar_mode1 <-
  ## 긴 형태의 100 일 코로나19 데이터에서
  covid19_df_100 |>
  ## date 의 월 단위 열을 yearmonth 에 저장
  mutate(yearmonth = lubridate::floor_date(date, "month")) |>
  ## Location, yearmonth 로 그룹화
  group_by(location, yearmonth) |>
  ## new_cases 합계 산출
  summarise(new_cases = sum(new_cases)) |>
  ## 연월을 열로 값을 new_cases 로 설정한 넓은 데이터프레임 생성
  pivot_wider(names_from = yearmonth, values_from = new_cases) |>
  ## plotly 객체 생성
  plot_ly()

## 최종 결과는 fig 에 저장
fig <- p_bar_mode1 |>
  ## 막대 trace 를 추가
  add_trace(type = 'bar',
            ## X, Y 축을 변수에 매팅, color 를 설정
            x = ~location, y = ~`2021-12-01`, color = I('gray30'),
            ## 범례 이름 설정, text 는 변수(특수문자가 포함되어 ` 사용) 매팅
            name = '2021 월 12 월', text = ~`2021-12-01`,
            ## textposition 을 'inside' 로 설정
            textposition = 'inside',
            ## textfont 의 세부 속성 설정
            textfont = list(color = 'white', size = 10)
            )

## 최종 결과는 다시 fig 에 저장
fig <-
  ## 앞서 생성한 fig 에
  fig |>
  ## 막대 trace 를 추가
  add_trace(type = 'bar',
            ## X, Y 축을 변수에 매팅, color 를 설정
            x = ~location, y = ~`2022-01-01`, color = I('gray40'),
            ## 범례 이름 설정, text 는 변수(특수문자가 포함되어 ` 사용) 매팅
```

```

name = '2022 월 1 월', text = ~`2022-01-01`,
## textposition 을 'inside'로 설정
textposition = 'inside',
## textfont 의 세부 속성 설정
textfont = list(color = 'white', size = 10)
)

## 최종 결과는 다시 fig에 저장
fig <-
## 앞서 생성한 fig에
fig |>
## 막대 trace 를 추가
add_trace(type = 'bar',
          ## X, Y 축을 변수에 매팅, color 를 설정
          x = ~location, y = ~`2022-02-01`, color = I('gray50'),
          ## 범례 이름 설정, text 는 변수(특수문자가 포함되어 ` 사용) 매팅
          name = '2022 월 2 월', text = ~`2022-02-01`,
          ## textposition 을 'inside'로 설정
          textposition = 'inside',
          ## textfont 의 세부 속성 설정
          textfont = list(color = 'white', size = 10)
          )

## 최종 결과는 다시 fig에 저장
fig <-
## 앞서 생성한 fig에
fig |>
## 막대 trace 를 추가
add_trace(type = 'bar',
          ## X, Y 축을 변수에 매팅, color 를 설정
          x = ~location, y = ~`2022-03-01`, color = I('darkred'),
          ## 범례 이름 설정, text 는 변수(특수문자가 포함되어 ` 사용) 매팅
          name = '2022 월 3 월', text = ~`2022-03-01`,
          ## textposition 을 'inside'로 설정
          textposition = 'inside',
          ## textfont 의 세부 속성 설정
          textfont = list(color = 'white', size = 10)
          )

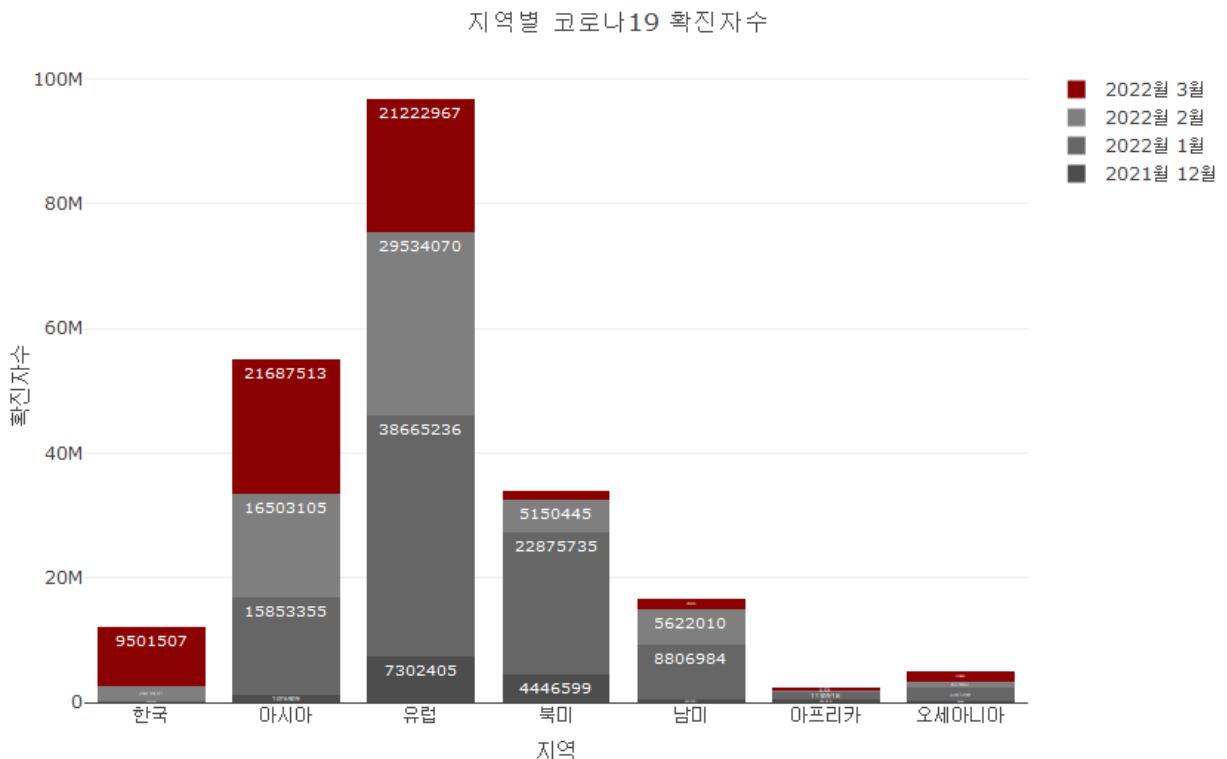
fig |>
## barmode 를 stack 으로 설정

```

```

layout(barmode = 'stack',
       title = list(text = '지역별 코로나 19 확진자수'),
       xaxis = list(title = '지역'),
       yaxis = list(title = '확진자수'),
       margin = margins
)

```



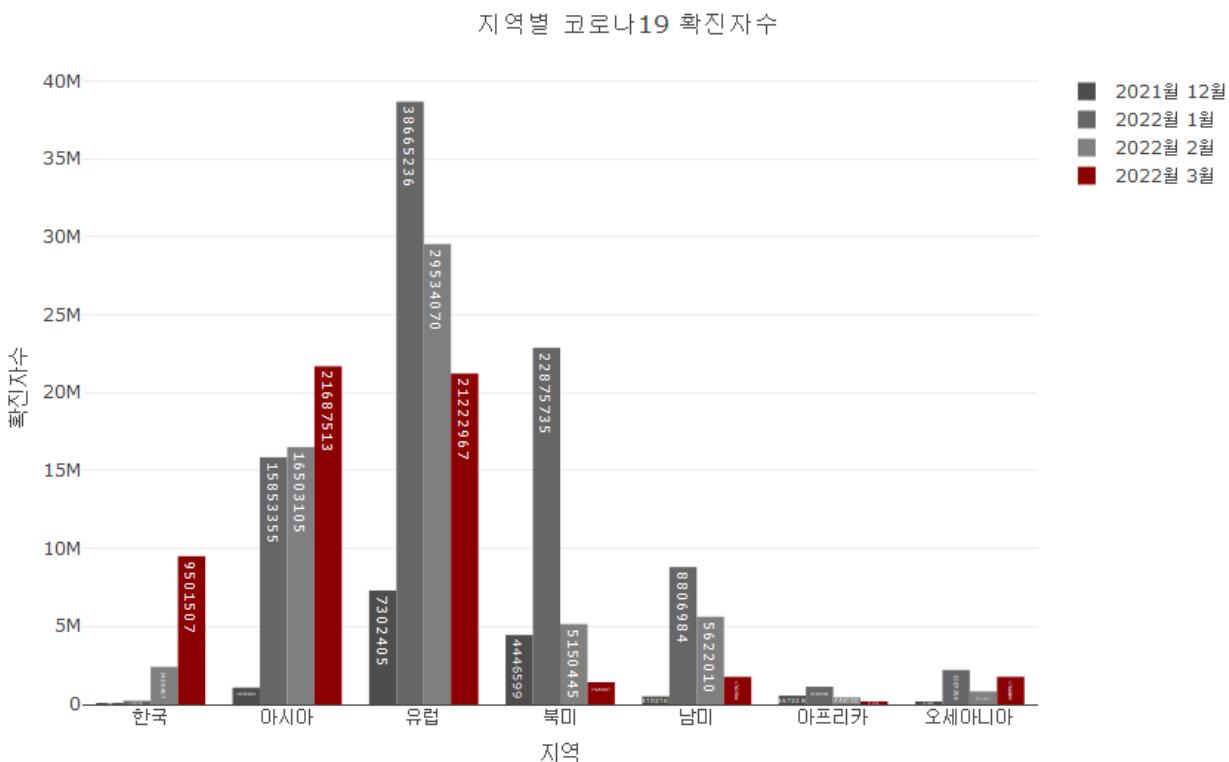
실행결과 3-45. 막대 trace의 stack 모드의 text 설정 결과

앞서 만들어진 `plotly` 객체인 `fig` 를 다음과 같이 'group'형 막대 trace 로 바꾸어 표시할 수 있다.

```

## 앞서 생성한 fig 에
fig |>
  ## barmode 를 group 으로 설정
  layout(barmode = 'group',
         title = list(text = '지역별 코로나 19 확진자수'),
         xaxis = list(title = '지역'),
         yaxis = list(title = '확진자수'),
         margin = margins
)

```



실행결과 3-46. 막대 trace 의 group 모드의 text 설정 결과

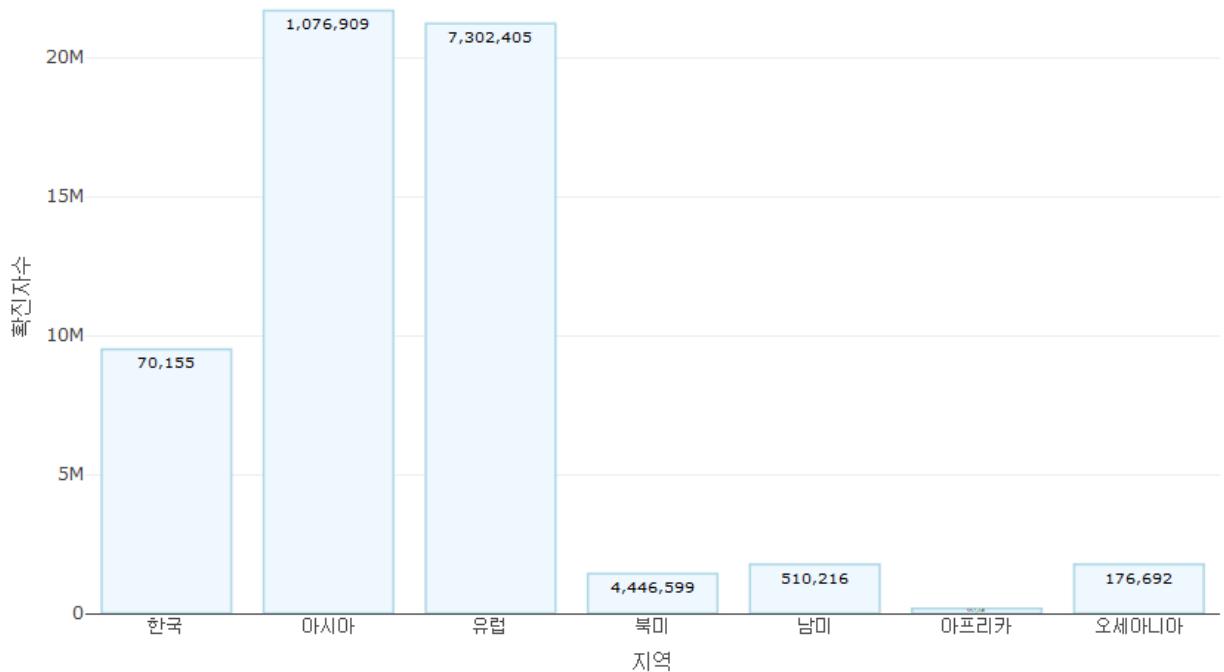
#### 2.2.2.3.2. marker

앞서 스캐터 trace 에서 산점도, 선 그래프를 생성할 때 `marker` 를 사용하여 점과 선의 형태를 설정하였다. 그러나 막대 trace 에서의 `makrker` 는 막대 자체를 의미한다. 따라서 `marker` 로 막대의 내부 색, 외부 외곽선 등의 세부 설정을 위한 속성을 지정할 수 있다.

```
## p_bar_mode1 0##
p_bar_mode1 |>
  ## 막대 trace 추가
  add_trace(type = 'bar',
             x = ~location, y = ~`2022-03-01`, color = I('darkred'),
             name = '2021 월 12 월', text = ~`2021-12-01`,
             textposition = 'inside',
             textfont = list(color = 'black', size = 10),
  ## texttemplate 를 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)
```

지역별 코로나19 확진자수



실행결과3- 47. marker 를 사용한 막대 설정 결과

#### 2.2.2.4. 박스(Box) trace : 박스 플롯

박스 trace 는 박스 플롯을 생성하기 위해 사용되는 trace 이다. 앞 장에서 설명했듯이 박스 플롯은 데이터의 전체적 분포를 4 분위수(quantile)과 IQR(Inter Quartile Range)를 사용하여 표시하는 시각화로 연속형 변수와 이산형 변수의 시각화에 사용되는 방법이다. 박스 trace 를 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
```

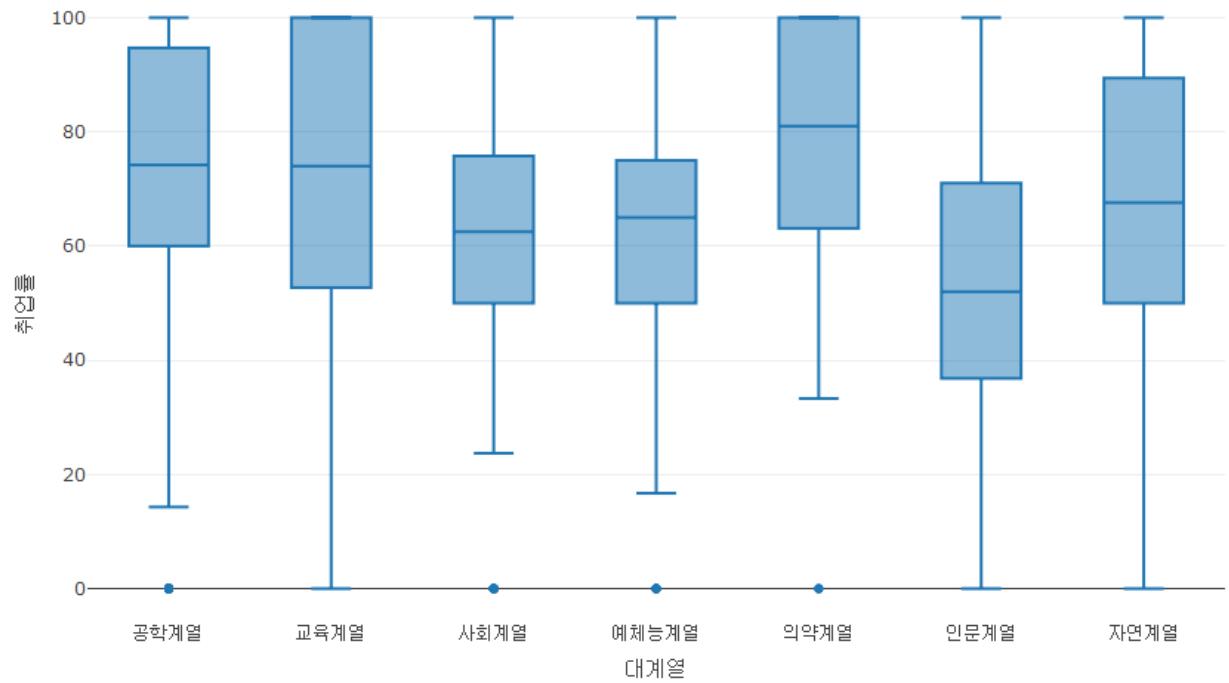
- ... : 박스 trace 의 line 모드에 설정할 수 있는 속성 설정
- data : 시각화할 데이터프레임
- inherit : plot\_ly()에 설정된 속성 type 을 상속할지를 결정하는 논리값
- x : X 축에 매핑할 변수를 ~로 설정
- y : Y 축에 매핑할 변수를 ~로 설정

```
p_box <-
  ## 취업통계 데이터를 사용하여 plotly 객체 생성
  df_취업통계_2000 |>
    plot_ly()

p_box |>
  ## 박스 trace 추가
  add_trace(type = 'box',
            ## X, Y 축 변수 매핑
            x = ~대계열, y = ~취업률) |>

  ## 제목, 여백 설정
  layout(title = list(text = '대학 계열별 취업률 분포'),
         margin = margins)
```

대학 계열별 취업률 분포



실행결과 3-48. 박스 trace 생성

박스 trace 에서의 사용되는 주요 속성은 다음과 같다.

속성	설명	속성 값	세부 속성
type	trace type 속성	'box'	
visible	trace 의 표시 설정	TRUE, FALSE, 'legendonly'	
width	박스의 너비 설정	0 이상의 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	
notched	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	Q1 과 Q3 를 계산하는 방식 설정	'linear', 'inclusive', 'exclusive'	

속성	설명	속성 값	세부 속성
pointpos	샘플 점이 그려지는 상대적 위치 설정	-2~2 까지의 수치	
jitter	샘플 점이 그려지는 너비 설정	0 ~ 1 사이 값	

#### 2.2.2.4.1. *boxmode*

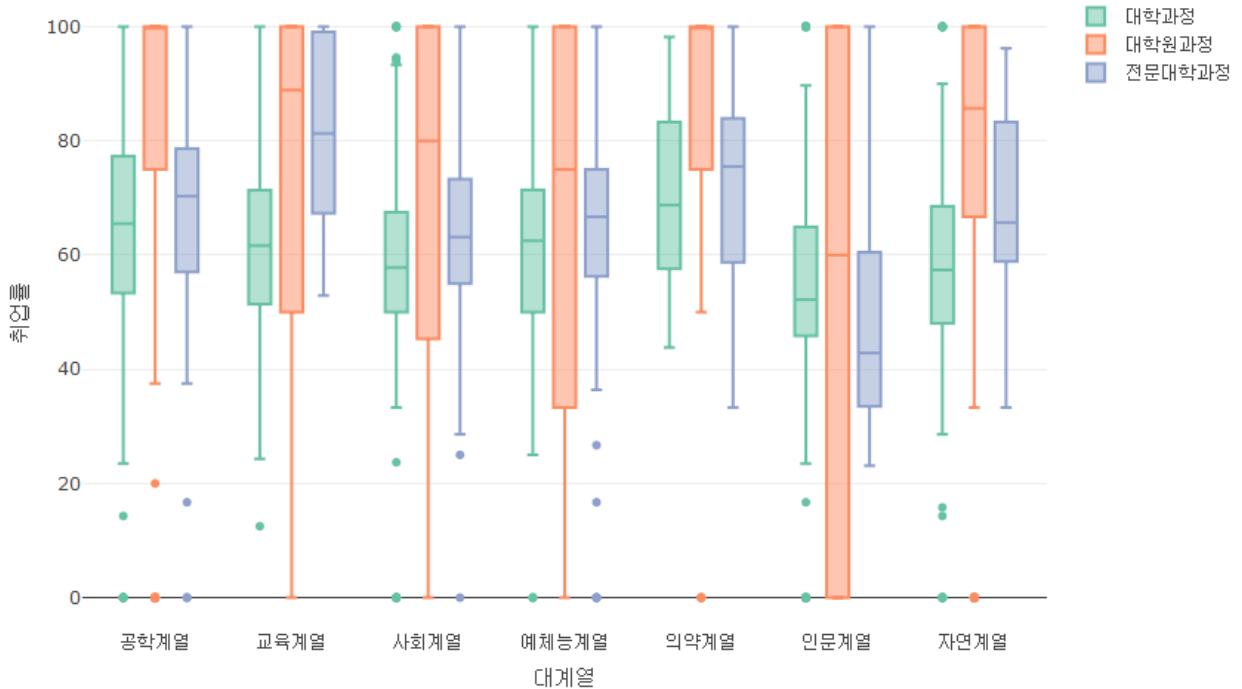
박스 trace 도 막대 trace 와 같이 그룹화한 변수를 **color** 나 **linetype** 등의 속성에 매핑시켜서  
다수의 박스 trace 를 한번에 만들거나 각각의 박스 trace 를 추가하여 그릴 수 있다. 이렇게  
여러개의 박스 trace 를 그릴 때 그 구성 형태를 결정하기 위해서는 차후 설명할 **layout()**의  
**boxmode** 속성을 설정함으로써 가능하다.

**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 **color** 로 과정구분을 매핑하여 'group'형 박스 플롯을  
생성하는 코드이다.

```
p_box_group <- p_box |>
  add_trace(type = 'box',
            x = ~대계열, y = ~취업률,
            ## color 를 과정구분으로 매핑
            color = ~과정구분)

p_box_group |>
  ## boxmode 를 group 으로 설정
  layout(boxmode = "group") |>
  layout(title = list(text = '대학 계열별 취업률 분포'),
         margin = margins)
```

대학 계열별 취업률 분포



### 실행결과 3-49. 박스 trace 의 group mode 실행 결과

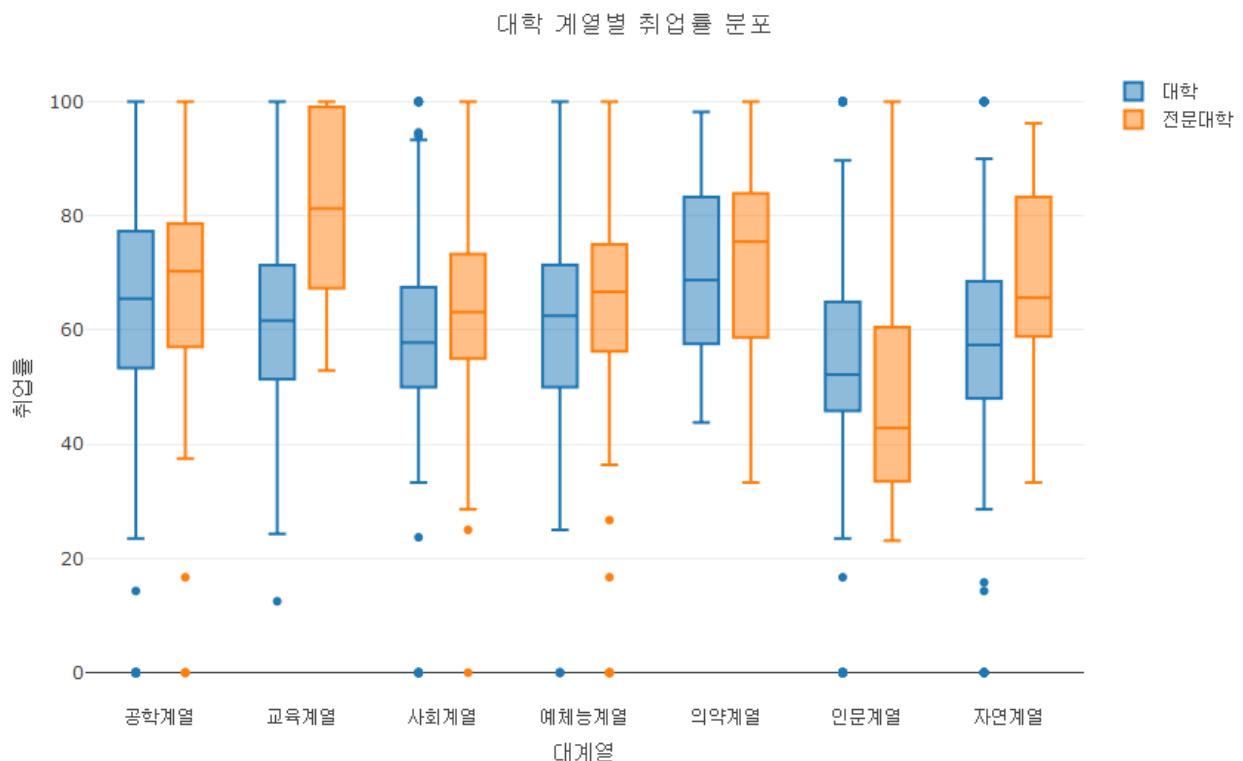
박스 trace 도 앞선 막대 trace 와 같이 넓은 형태의 데이터프레임에 각 열을 하나의 박스 trace 로 추가함으로써 전체 막대 trace 를 만들어 줄 수 있다.

```
p_box_add <- p_box |>
  ## df_취업통계_2000 에서 대학과정 데이터를 필터링한 데이터를 사용한 박스 trace 추가
  add_trace(type = 'box', data = df_취업통계_2000 |> filter(과정구분 == '대학과정'),
  ## X, Y 축을 변수에 맵핑, 범례이름을 대학으로 설정
  x = ~대계열, y = ~취업률, name = '대학')

p_box_add <- p_box_add |>
  ## df_취업통계_2000 에서 전문대학과정 데이터를 필터링한 데이터를 사용한 박스 trace 추가
  add_trace(type = 'box', data = df_취업통계_2000 |> filter(과정구분 == '전문대학과정'),
  ## X, Y 축을 변수에 맵핑, 범례이름을 전문대학으로 설정
  x = ~대계열, y = ~취업률, name = '전문대학')

p_box_add |>
  ## boxmode 를 group 으로 설정
  layout(boxmode = "group",
```

```
title = list(text = '대학 계열별 취업률 분포'),
margin = margins)
```



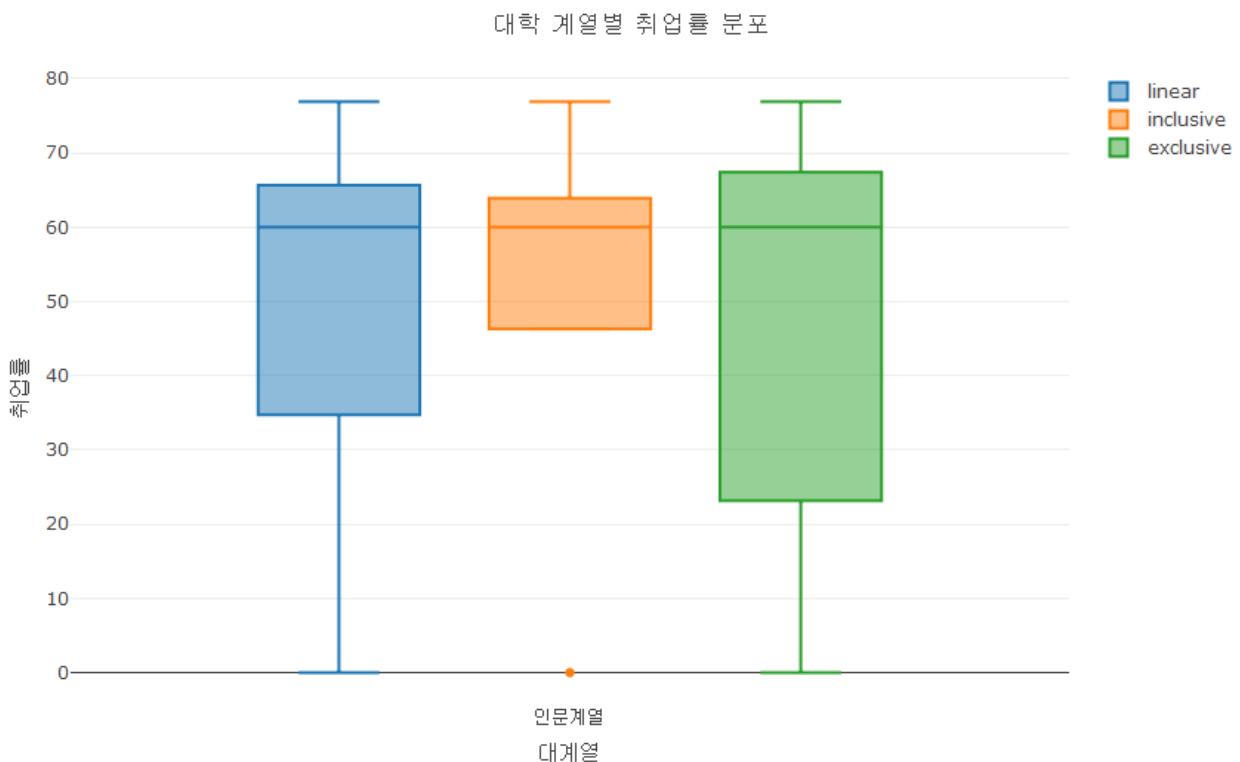
### 실행결과 3- 50. 여러 박스 trace 를 추가

#### 2.2.2.4.2. quartilemethod

박스 플롯에는 여러가지 요약 통계치가 사용되는데 그중 하나가 전체 데이터의 25% 위치인 Q1 과 75% 위치인 Q3 이다. 이 위치는 전체 데이터를 Y 값에 따라 정렬하고 25%, 75%의 값을 찾는 'linear' 방법이 기본적으로 사용된다. 하지만 `plotly` 는 'inclusive'와 'exclusive'의 두 가지 방법을 추가적으로 제공한다. 'exclusive' 방법은 전체 데이터 사례수가 홀수 인경우 중앙값을 사용하여 두 개의 그룹으로 분리하고 다시 이 두개의 그룹의 중앙값을 사용해 Q1 과 Q3 를 구하는 방법이다. 'inclusive' 방법도 중앙값을 사용하여 두 그룹으로 분리하지만 중앙값을 포함하여 다시 두 그룹의 중앙값을 구하여 Q1 과 Q3 를 구하는 방법이다. 이들 방법은 사례수가 적은 경우 그 차이가 확실히 드러나지만 사례수가 많으면 그 차이가 잘 드러나지는 않는다.

```
p_box_quartilemethod <-
## df_취업통계_2000 에서
df_취업통계_2000 |>
## 인문계열의 상위 9 개 데이터 추출
```

```
filter(대계열 == '인문계열') |> head(9) |>  
## plotly 객체 생성  
plot_ly()  
  
p_box_quartilemethod <- p_box_quartilemethod |>  
## quartilemethod 가 Linear 인 박스 trace 추가  
add_trace(type = 'box', x = ~대계열, y = ~취업률,  
quartilemethod = 'linear', name = 'linear')  
  
p_box_quartilemethod <- p_box_quartilemethod |>  
## quartilemethod 가 inclusive 인 박스 trace 추가  
add_trace(type = 'box', x = ~대계열, y = ~취업률,  
quartilemethod = 'inclusive', name = 'inclusive')  
  
p_box_quartilemethod <- p_box_quartilemethod |>  
## quartilemethod 가 exclusive 인 박스 trace 추가  
add_trace(type = 'box', x = ~대계열, y = ~취업률,  
quartilemethod = 'exclusive', name = 'exclusive')  
  
p_box_quartilemethod |>  
## boxmode 를 group 으로 설정  
layout(boxmode = "group",  
       title = list(text = '대학 계열별 취업률 분포'),  
       margin = margins)
```



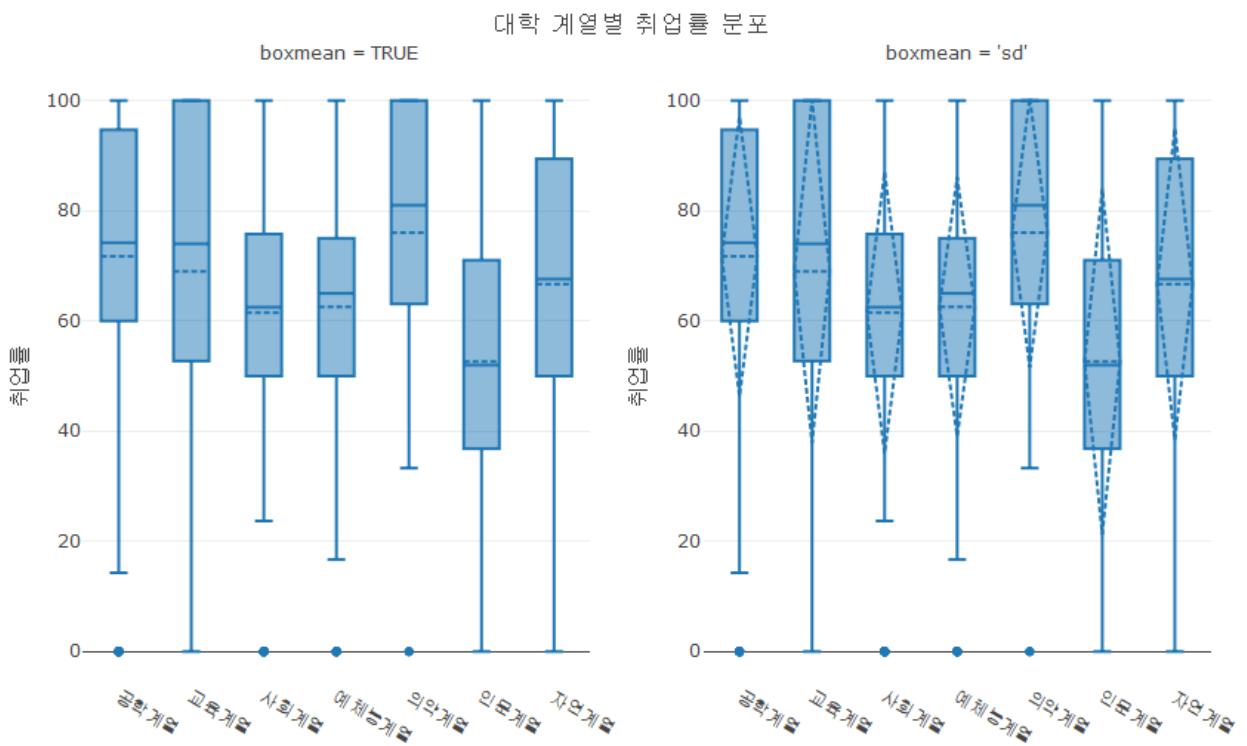
### 실행결과3- 51. quartilemethod 의 설정에 따른 결과

#### 2.2.2.4.3. boxmean, boxpoints

박스 trace 에서 제공하는 요약 통계중에 가장 많이 사용되지만 제공되지 않는 요약통계가 바로 평균(mean)이다. `ggplot2` 에서는 평균을 표현하기 위해 다소 어려운 과정을 거쳐야 했지만 `plotly` 에서는 `boxmean` 속성의 설정만으로 간단하게 평균값을 표현할 수 있다. `boxmean` 은 TRUE/FALSE 의 논리값에 표준편차가 추가로 표시되는 'sd'를 제공한다.

```
p_box |>
  ## boxmean 을 TRUE 로 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 |>
  ## boxmean 을 sd 로 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 3- 52. boxmean 설정에 따른 결과

또 박스 trace 에서 유용하게 사용되는 속성이 `boxpoints` 이다. `boxpoints` 는 이상치(outlier)로 표현되는 점의 표현을 제어할 수 있다. `boxpoints` 로 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 IQR 의 4 배가 넘어가는 이상치는 다시 강조되는 방법이다. 'FALSE'는 이상치를 표시하지 않는다.

```
p_box |>
  ## boxpoint 을 all 로 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 |>
  ## boxpoint 을 outliers 로 설정한 박스 trace 추가
  add_trace(type = 'box', boxpoints = 'outliers'
            , x = ~대계열, y = ~취업률) |>
  layout(title = list(text = '대학 계열별 취업률 분포'),
```

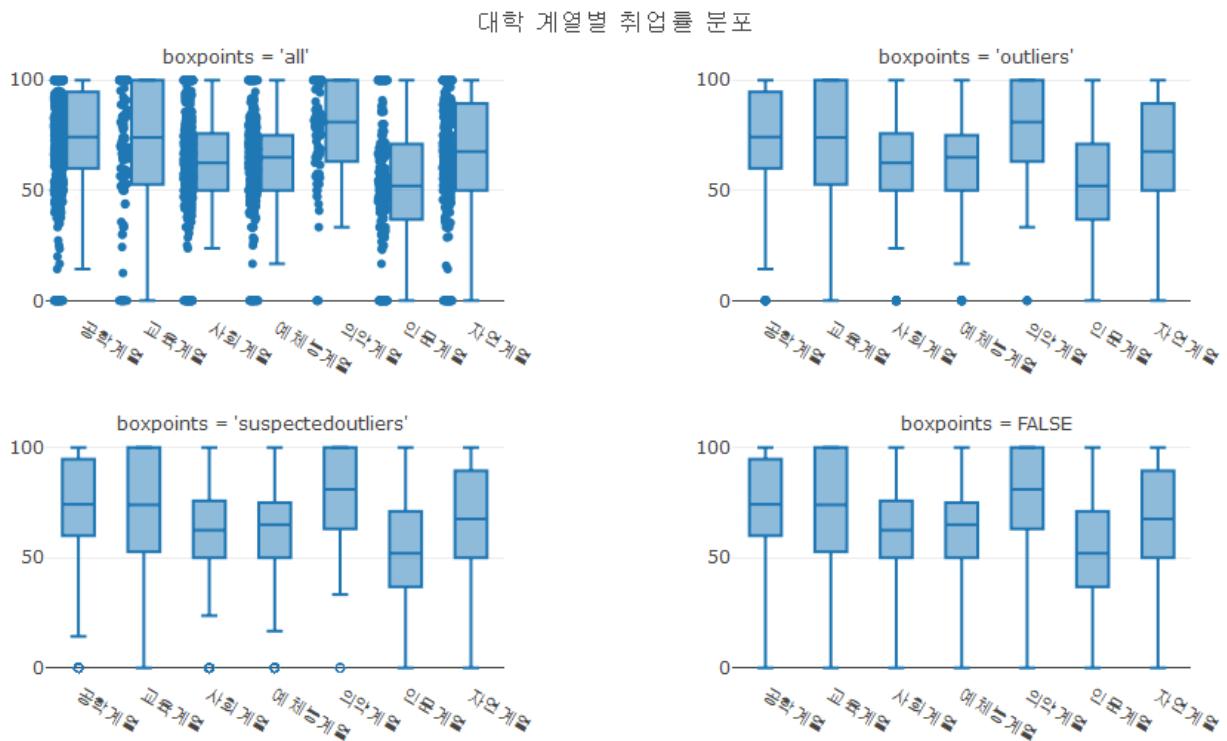
```

margin = margins)

p_box |>
## boxpoint 을 suspectedoutliers 로 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 |>
## boxpoint 을 FALSE 로 설정한 박스 trace 추가
add_trace(type = 'box', boxpoints = FALSE,
          x = ~대계열, y = ~취업률) |>
layout(title = list(text = '대학 계열별 취업률 분포'),
       margin = margins)

```



실행결과3- 53. boxpoints 설정에 따른 결과

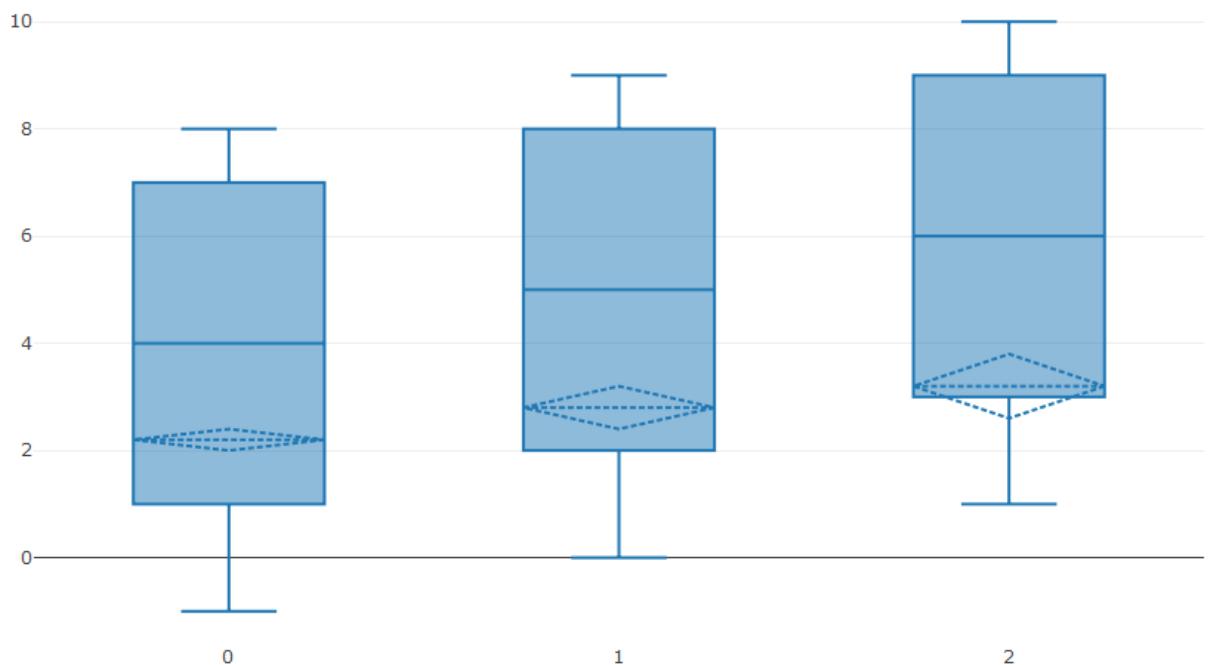
#### 2.2.2.4.4. mean, median, q1, q3, upperfence, lowerfence

박스 trace 는 박스 플롯의 생성시 자동적으로 여러가지 요약 통계를 산출하지만 경우에 따라 이 값을 미리 정해진 값으로 사용할 수도 있다. 이를 위해 사용되는 type 들이 **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(X 축 아이템의 수)와 동일해야 한다. 따라서 이 type 들을 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
            ## q1 과 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)
```

수동으로 박스 플롯 만들기



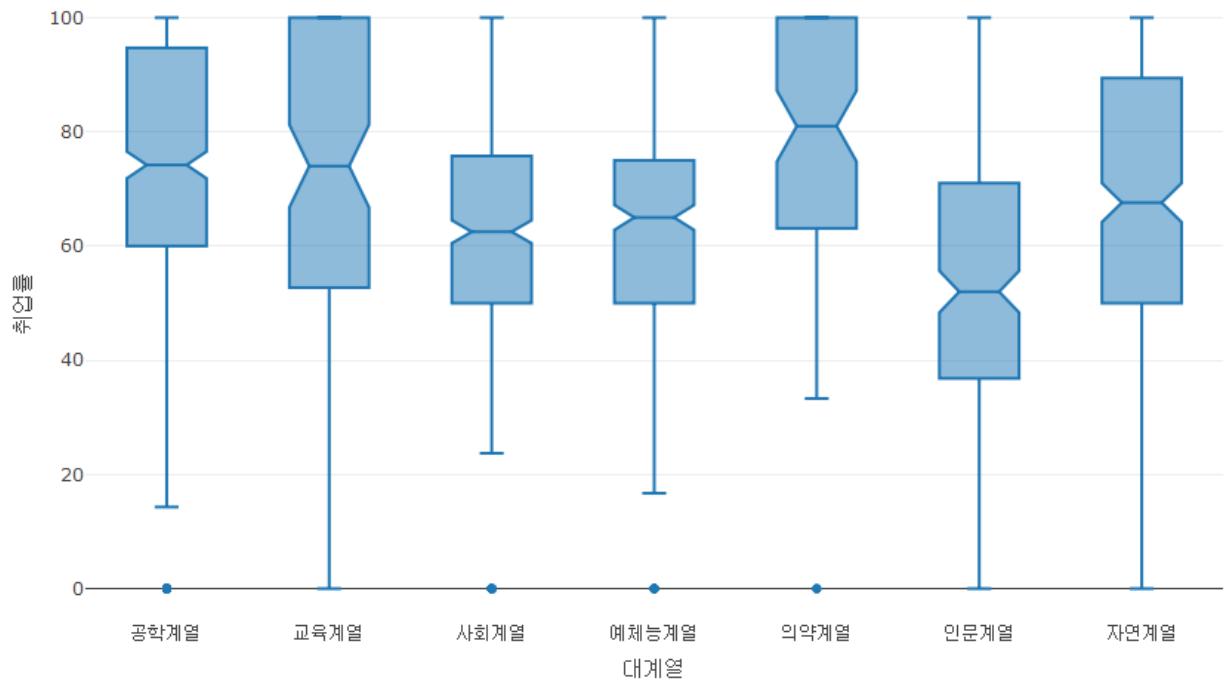
실행결과 3- 54. 수동으로 만든 박스 trace

#### 2.2.2.4.5. *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 `notched`이다. `notched`는 TRUE/FALSE의 두 가지 중에 하나의 값을 가지는데 TRUE일 경우 notch 디자인으로 설정된다. `notched`가 TRUE로 설정된 박스 플롯은 `notchwidth`, `notchspan`을 사용하여 홈의 크기를 설정할 수 있는데 `notchwidth`는 홈이 파고들어간 너비를 설정하는데 전체 막대의 너비에 대한 비율로 설정하기 때문에 0부터 0.5까지의 값을 가질 수 있다. 또 `notchspan`은 수직 방향으로의 홈의 크기를 설정하는 속성이다. 막대 외부에 표시되는 수염의 끝에 있는 가로선의 크기를 설정하는 속성이 `whiskerwidth`이다. 이 속성도 `notchwidth`와 같이 0부터 0.5 사이의 값을 가진다.

```
p_box |>
  ## 박스 trace를 추가
  add_trace(type = 'box', x = ~대계열, y = ~취업률,
            ## notch 스타일의 설정
            notched = TRUE, notchwidth = 0.2,
            notchspan = 0.5,
            ## 수염 길이 설정
            whiskerwidth = 0.3) |>
  layout(title = list(text = '대학 계열별 취업률 분포'),
         margin = margins)
```

대학 계열별 취업률 분포



실행결과3- 55. norch 가 적용된 박스 trace

#### 2.2.2.4.6. marker, jitter

막대 trace 를 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 trace 에서 데이터의 실제 분포를 표시하는 type 이 **marker** 와 **jitter** 이다.

**marker** 는 박스 trace 에서 사용되는 점들의 세부 설정을 위한 속성이다. 이 세부 설정은 스캐너 trace 에서 사용되는 **marker** 와 거의 동일하기 때문에 앞의 스캐터 trace 의 **marker** 를 참조하면 된다.

**jitter** 에는 0 부터 1 까지의 값을 가지는데 샘플 데이터 점들이 표시되는 구간의 너비가 설정된다. 0 은 한 줄에 표기되고 1 은 박스의 너비와 동일한 너비로 점들이 분산되어 표기된다. **jitter** 와 함께 쓰여 샘플 데이터의 표시를 제어하는 속성으로 **boxpoints** 와 **pointpos** 가 있다. **boxpoints** 는 박스에 표시되는 점의 범위를 설정한다. 기본값은 'outlier'로 설정되어 있는데 이 경우 표시되는 점은 이상치에 속하는 점만 표현되기 때문에 **jitter** 의 효과를 보지 못한다. 이 값을 'all'로 설정하여 전체 데이터로 범위를 설정하여야 전체적인 분포를 볼 수 있다. 또

`pointpos` 는 점이 표기되는 위치를 설정하는데 -2 부터 2 까지의 값을 가진다. 이 값은 0(박스의 중심)부터 박스 너비의 2 배까지의 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 3- 56. 점이 표시된 박스 trace

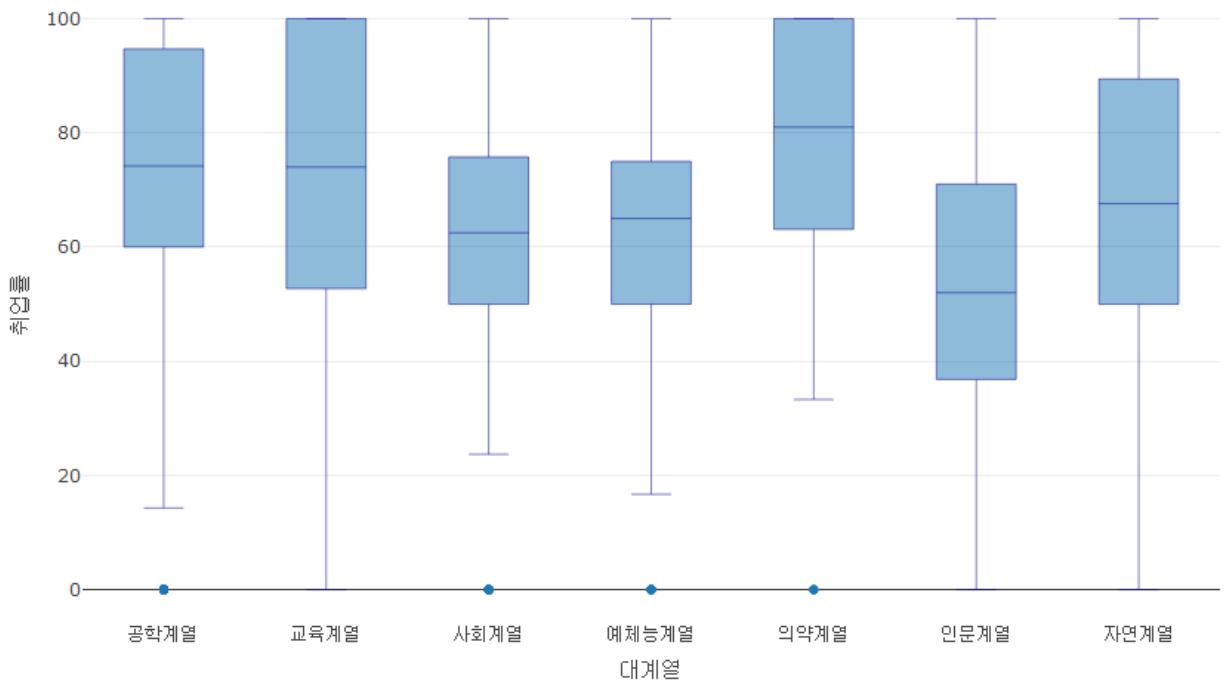
#### 2.2.2.4.7. line

`line` 은 막대 trace 에서 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 trace 의 선 세부 속성 설정
             line = list(width = 0.5, color = 'darkblue')) |>
```

```
layout(title = list(text = '대학 계열별 취업률 분포'),  
       margin = margins)
```

대학 계열별 취업률 분포



실행결과 3- 57. 막대 trace 의 선 설정 결과

#### 2.2.2.5. 바이올린(Violin) trace : 바이올린 플롯

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

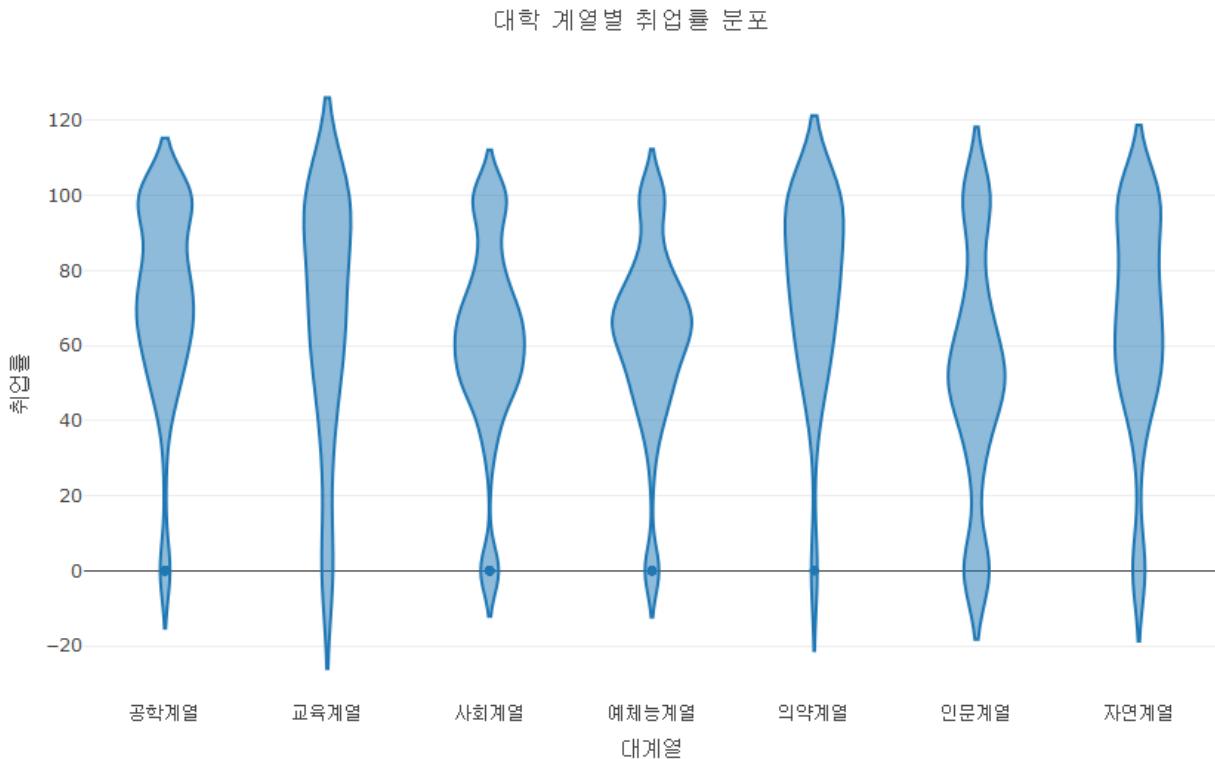
바이올린 trace 를 사용해 바이올린 플롯을 생성하기 위해서는 `add_trace(type = 'violin')`를 사용하여야하고 래핑함수를 제공하지 않는다.

```
add_trace(p, type = 'violin', ..., data = NULL, inherit = TRUE)
```

- p : `plot_ly()`로 생성한 `plotly` 객체
- type : trace 타입을 'violin'로 설정
- ... : 바이올린 trace 의 line 모드에 설정할 수 있는 속성 설정
- data : 시각화할 데이터프레임

- inherit : plot\_ly()에 설정된 속성 type 을 상속할지를 결정하는 논리값

```
p_violin <-  
df_취업통계_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)
```



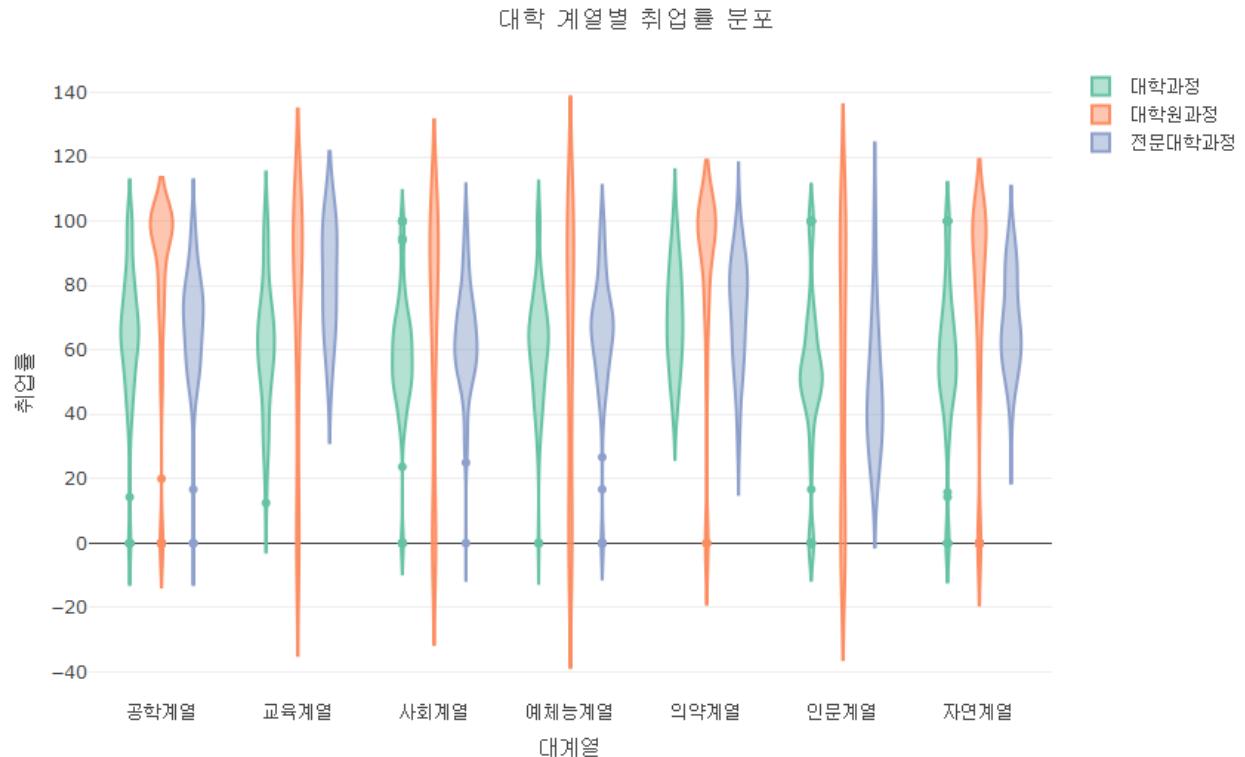
실행결과 3- 58. 바이올린 trace 생성

#### 2.2.2.5.1. violinemode

바이올린 trace 도 막대 trace 와 같이 그룹화한 변수를 color 나 linetype 등의 속성에 매핑시켜서 다수의 바이올린 grace 를 한번에 만들거나 각각의 바이올린 trace 를 추가하여 그릴 수 있다. 이렇게 여러개의 바이올린 trace 를 그릴 때 그 구성 형태는 막대 trace 와 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 `color`로 과정구분을 매핑하여 'group'형 바이올린 플롯을 생성하는 코드이다.

```
p_violin |>
  add_trace(type = 'violin', x = ~대계열, y = ~취업률,
             ## color 를 변수에 매핑
             color = ~과정구분) |>
  ## violinmode 를 group 으로 설정
  layout(violinmode = 'group',
         title = list(text = '대학 계열별 취업률 분포'),
         margin = margins)
```



실행결과3-59. `color` 가 매핑된 바이올린 trace

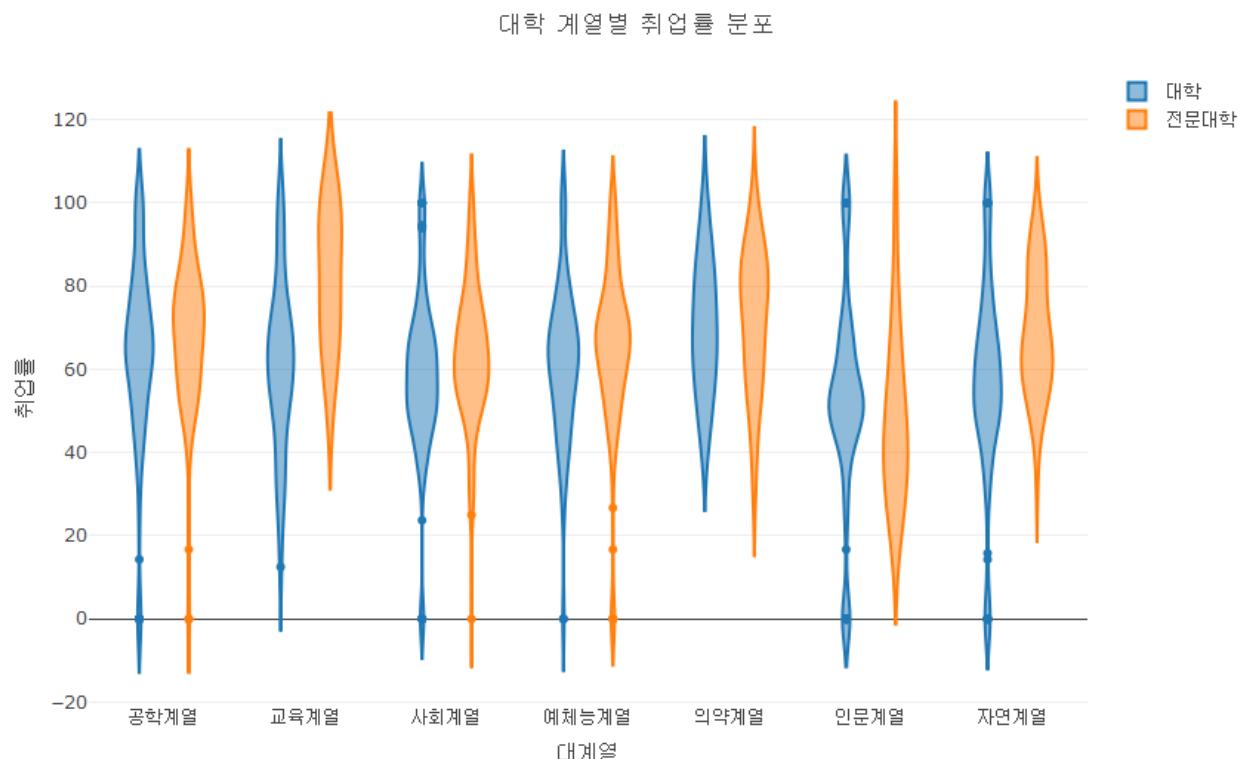
다음의 코드는 과정구분이 '대학'인 바이올린 trace 와 '전문대학'인 바이올린 trace 를 추가함으로서 그룹화된 바이올린 플롯을 생성하는 코드이다.

```
p_violin |>
  ## 대학과정만 필터링한 데이터를 사용한 바이올린 trace 추가
```

```

add_trace(data = df_취업통계_2000 |> filter(과정구분 == '대학과정'),
          type = 'violin', x = ~대계열, y = ~취업률, name = '대학') |>
## 전문대학과정만 필터링한 데이터를 사용한 바이올린 trace 추가
add_trace(data = df_취업통계_2000 |> filter(과정구분 == '전문대학과정'),
          type = 'violin', x = ~대계열, y = ~취업률, name = '전문대학') |>
## violinmode 를 group 으로 설정
layout(violinmode = 'group',
       title = list(text = '대학 계열별 취업률 분포'),
       margin = margins)

```



### 실행결과 3- 60. 여러 바이올린 trace 의 추가

바이올린 trace 에서 사용되는 주요 속성은 다음과 같다.

속성	설명	속성 값	세부 속성
type	trace type 속성	'violin'	
visible	trace 의 표시 설정	TRUE, FALSE, 'legendonly'	
width	바이올린 trace 너비 설정	0 이상의 수치값	
orientation	바이올린 trace 의 방향 설정	'v', 'h'	
marker	바이올린 trace 의 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	-2~2 까지의 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	

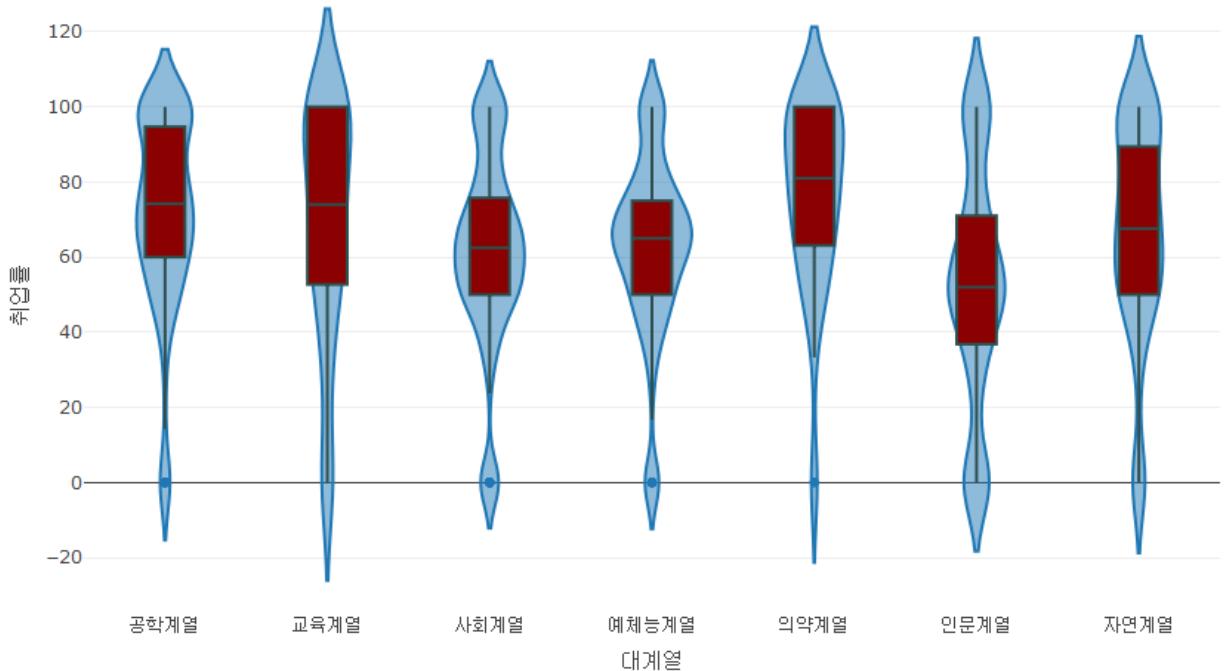
#### 2.2.2.5.2. *box, meanline*

바이올린 플롯은 박스 플롯과 유사한 정보를 제공하는 시각화이지만 박스 플롯의 정보와 같이 사용되면 더욱 데이터 시각화의 효과가 높아진다. 바이올린 trace 에 박스 trace 를 추가하는

속성은 `box` 이고 이 `box` 속성은 세부 타입을 리스트로 설정하는 속성이다. `box`의 세부 속성은 `fillcolor`, `line`(세부 속성으로 `color`, `width`), `visible`, `width`의 네 가지이다.

```
p_violin |>
  add_trace(type = 'violin', x = ~대계열, y = ~취업률,
             ## 박스의 세부 속성 설정
             box = list(visible = TRUE, fillcolor = 'darkred',
                        line = list(color = 'darkslategray'),
                        width = 0.5)) |>
  layout(violinmode = 'group',
         title = list(text = '대학 계열별 취업률 분포'),
         margin = margins)
```

대학 계열별 취업률 분포

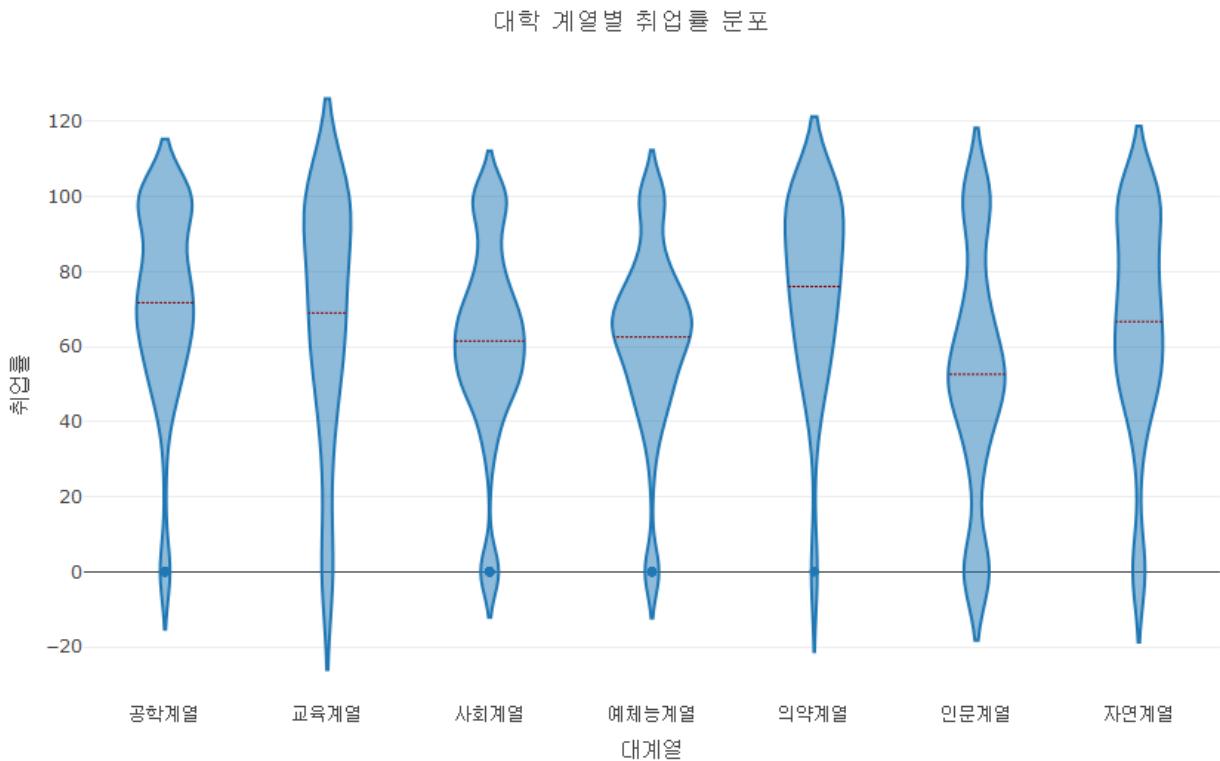


실행결과 3- 61. 바이올린 trace 과 박스 trace 의 병합

`meanline` 은 바이올린 trace 에 평균 선을 설정하는 속성으로 `color`, `visible`, `width`의 세부 속성을 리스트로 설정하여 사용한다.

```
p_violin |>
  add_trace(type = 'violin', x = ~대계열, y = ~취업률,
             ## meanline 세부 속성 설정
             meanline = list(visible = TRUE, color = 'darkred',
                           width = 1)) |>
```

```
layout(violinmode = 'group',
       title = list(text = '대학 계열별 취업률 분포'),
       margin = margins)
```



### 실행결과 3- 62. meanline 0/ 추가된 바이올린 trace

#### 2.2.2.5.3. side

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

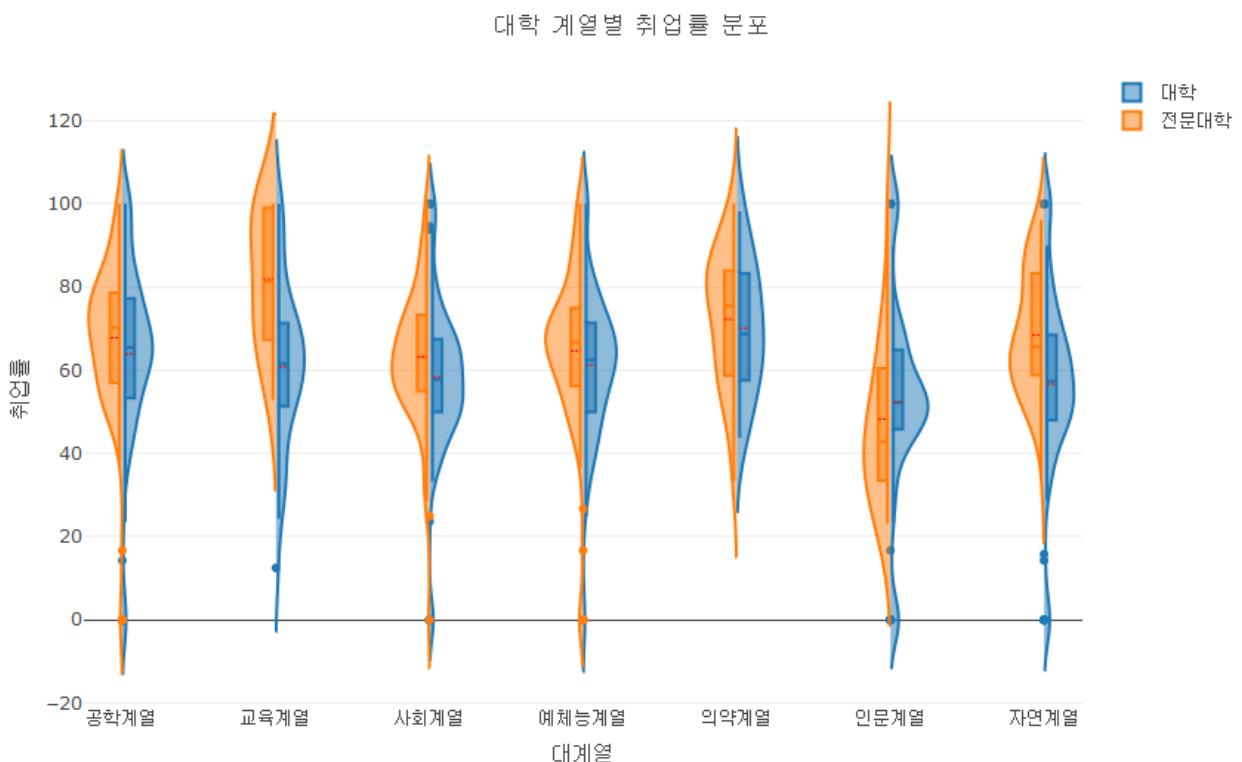
이렇게 두개의 바이올린 플롯을 반씩 잘라 붙이는 속성이 `side` 이다. `side` 는 바이올린의 양쪽을 다 사용하는 'both', 왼쪽 부분을 사용하는 'negative', 오른쪽 부분을 사용하는 'positive'를 설정할 수 있다. 그리고 이 두 바이올린을 붙이기 위해 `layout()`의 `violinmode` 를 `overlay` 로 설정한다. 여기에 앞서 설정한 `box` 와 `meanline` 설정하면 박스 trace 와 평균선도 반으로 그려서 붙여줄 수 있다. 앞서 그렸던 대학과 전문대학의 계열별 바이올린 플롯을 붙이는 코드는 다음과 같다.

```
p_violin |>
## 대학과정을 필터링한 데이터 설정
```

```

add_trace(data = df_취업통계_2000 |> filter(과정구분 == '대학과정'),
          ## 바이올린 trace 로 추가
          type = 'violin', x = ~대계열, y = ~취업률, name = '대학',
          ## side, box 의 설정
          side = 'positive', box = list(visible = TRUE, width = 0.5),
          ## meanline 의 속성 설정
          meanline = list(visible = TRUE, color = 'red', width = 1)) |>
## 전문대학과정을 필터링한 데이터 설정
add_trace(data = df_취업통계_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)

```

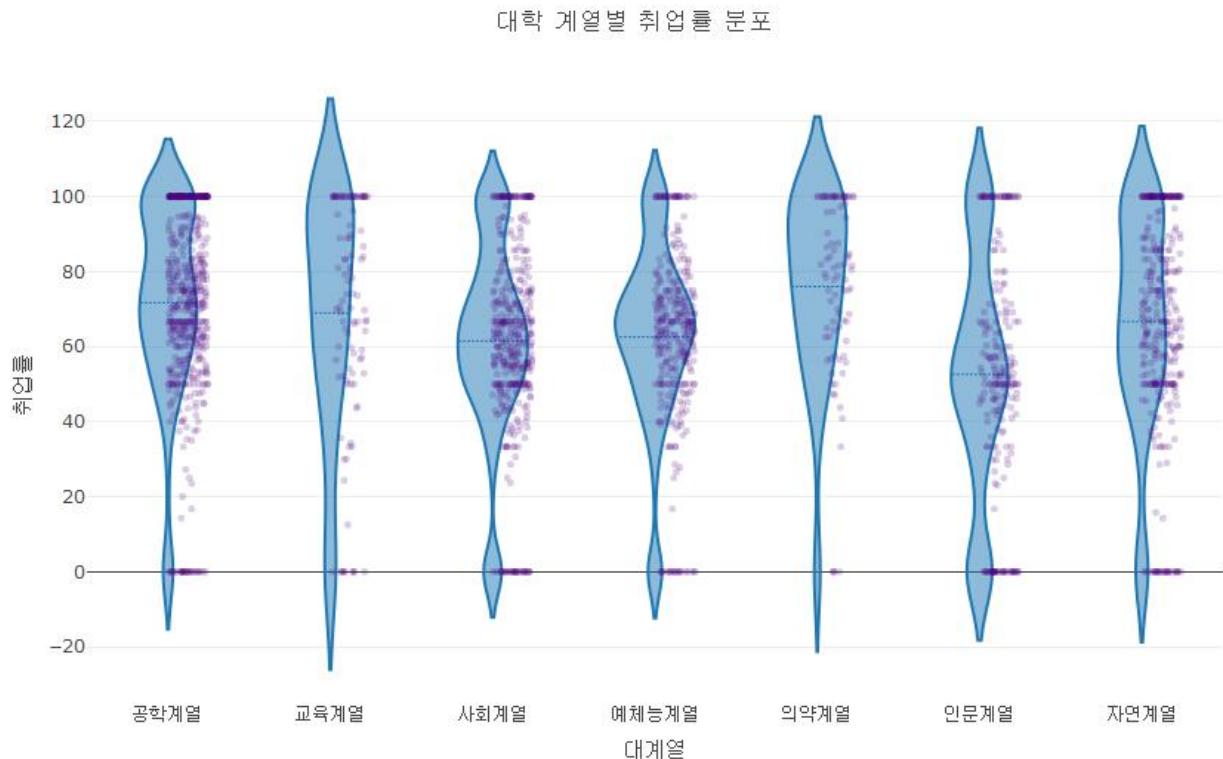


실행결과 3- 63. 바이올린trace 와 박스 trace 의 side 병합

#### 2.2.2.5.4. marker, jitter, line

바이올린 trace 에서도 박스 trace 와 같이 데이터를 점으로 표현할 수 있는데 이를 제어하기 위한 속성이 **marker** 와 **jitter** 이고 바이올린의 외곽선을 설정하는 속성이 **line** 이다. 이들의 설정은 박스 trace 와 동일하나 **boxpoints** 가 **points** 로 바뀐다는 점이 다르다.

```
p_violin |>
  add_trace(type = 'violin', x = ~대계열, y = ~취업률,
             ## meanline 의 세부 속성 설정
             meanline = list(visible = TRUE, color = 'darkred',
                             width = 1),
             ## marker 의 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 3- 64. 점이 표시된 바이올린 trace

### 2.2.2.6. 파이(pie) trace : 원 그래프

`ggplot2`에서 지원하지 않는 그래프 스타일이 원 그래프이다. 그래서 `ggplot2`에서 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 `plotly`에서는 trace로 지원한다.

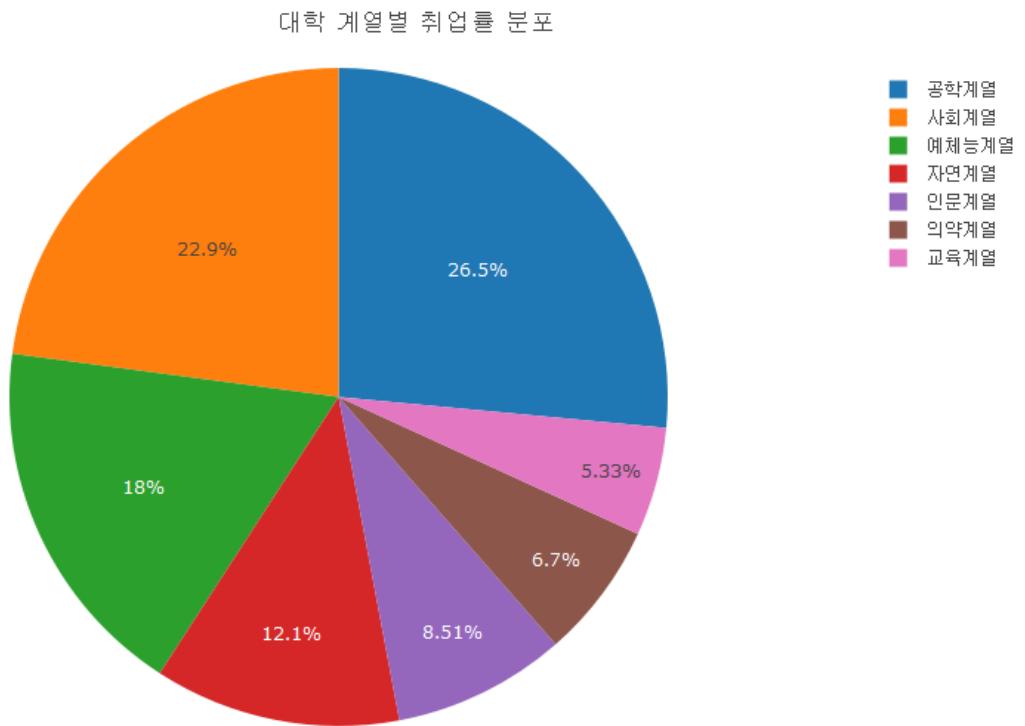
```
add_trace(p, type = 'pie', ..., data = NULL, inherit = TRUE)  
add_pie(p, values = NULL, labels = NULL, ..., data = NULL, inherit = TRUE)  
- p : plot_ly()로 생성한 plotly 객체  
- type : trace 타입을 'violin'로 설정  
- ... : 바이올린 trace의 line 모드에 설정할 수 있는 속성 설정  
- data : 시각화할 데이터프레임  
- inherit : plot_ly()에 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, Y 축에 매핑되는 `x`, `y` 속성의 설정이 없다. 대신 필수적 속성으로 `values`와 `labels`를 설정해야 한다.

`values`에 매핑되거나 설정된 수치 벡터에 따라 원을 각각의 슬라이스로 구분하게 되고 `labels`에 매핑하거나 설정한 벡터가 각각의 trace 이름으로 설정된다. 또 원 그래프에 표시되는 데이터 값은 기본적으로 백분율 값이 표시된다.

```
## 대계열별 졸업자수 합계 데이터로 설정된 plotly 객체 생성  
p_pie <- df_취업통계_2000 |> group_by(대계열) |>  
  summarise(졸업자수 = sum(졸업자수)) |>  
  plot_ly()  
  
p_pie |>  
  ## value 와 labels 를 매핑한 파이 trace 추가  
  add_trace(type = 'pie', values = ~졸업자수, labels = ~대계열) |>
```

```
layout(title = list(text = '대학 계열별 취업률 분포'),  
       margin = margins)
```



파이 trace에서 사용되는 주요 속성은 다음과 같다.

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

속성	설명	속성 값	세부 속성
sort	파이 trace 의 섹터 정렬 설정	논리값	

#### 2.2.2.6.1. *textinfo*

앞의 원 그래프에서 원 안에 표현된 데이터는 `values` 에 매핑된 데이터의 백분율이 표시되었다. 이 데이터의 표시를 설정하는 속성이 `textinfo` 이다. `textinfo` 에서 설정 가능한 값은 'label', 'text', 'value', 'percent'의 네 가지이고 이 네 가지를 `+/-`를 사용하여 동시에 사용할 수 있다. 'label'은 `labels` 로 설정된 값, `text` 는 `text` 로 설정된 값, 'value'는 `values` 에 설정된 값, 'percent'는 백분률 값이 표시된다.

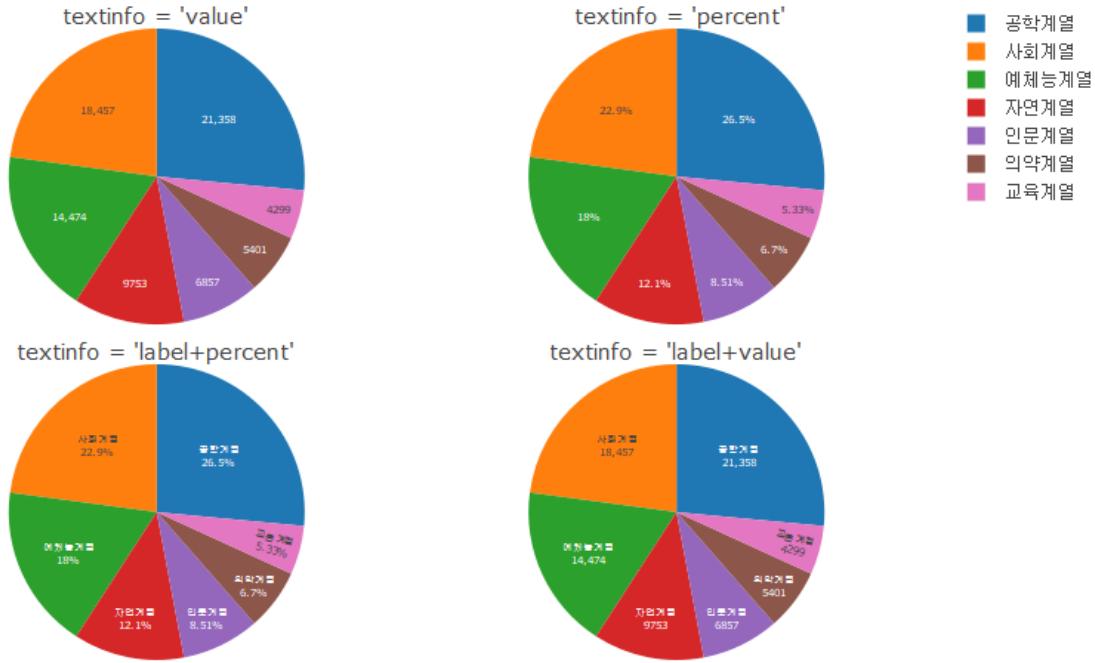
```
p_pie |>
  add_trace(type = 'pie', values = ~졸업자수, labels = ~대계열,
             ## textinfo 를 value 로 설정
             textinfo = 'value') |>
  layout(title = list(text = '대학 계열별 취업률 분포'),
         margin = margins)

p_pie |>
  add_trace(type = 'pie', values = ~졸업자수, labels = ~대계열,
             ## textinfo 를 percent 로 설정
             textinfo = 'percent') |>
  layout(title = list(text = '대학 계열별 취업률 분포'),
         margin = margins)

p_pie |>
  add_trace(type = 'pie', values = ~졸업자수, labels = ~대계열,
             ## textinfo 를 label+percent 로 설정
             textinfo = 'label+percent') |>
  layout(title = list(text = '대학 계열별 취업률 분포'),
         margin = margins)

p_pie |>
  add_trace(type = 'pie', values = ~졸업자수, labels = ~대계열,
             ## textinfo 를 label+value 로 설정
             textinfo = 'label+value') |>
  layout(title = list(text = '대학 계열별 취업률 분포'),
         margin = margins)
```

대학 계열별 취업률 분포

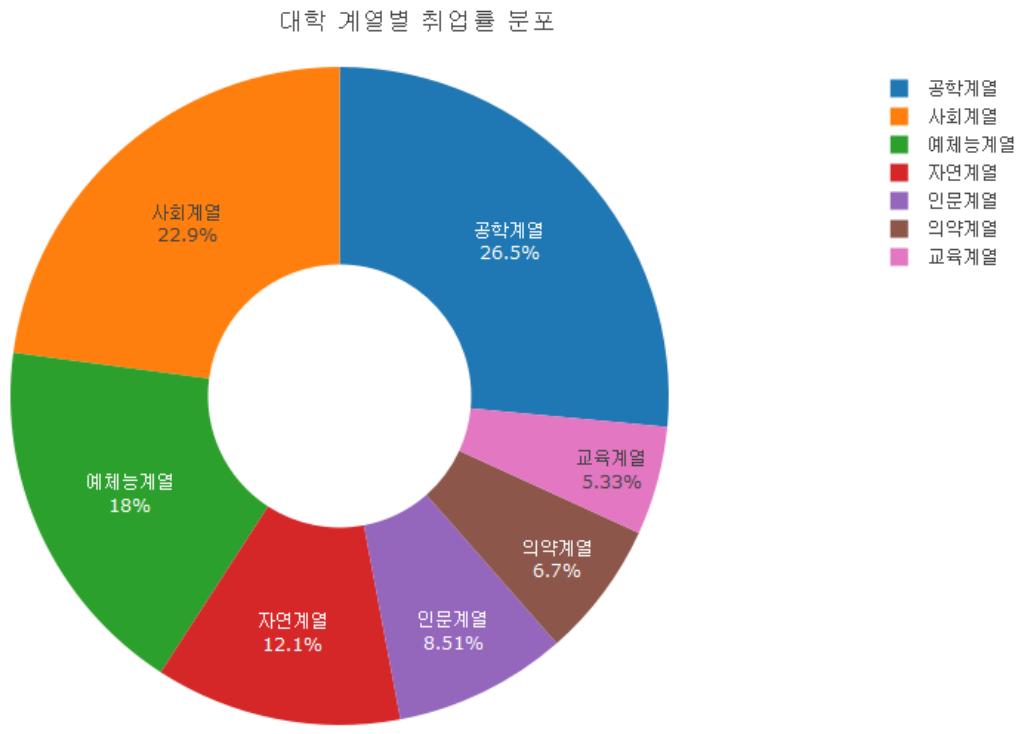


### 실행결과 3- 65. `textinfo` 설정에 따른 결과

#### 2.2.2.6.2. `hole`

`hole` 은 원 그래프의 내부에 표시되는 구멍의 크기를 설정하는 속성이다. `hole` 이 설정되지 않거나 0 으로 설정되면 완전한 원 그래프로 표시되지만 0 부터 1 사이의 값이 설정되면 그 크기만큼 빈 원이 생성되면서 도넛 그래프로 전환된다.

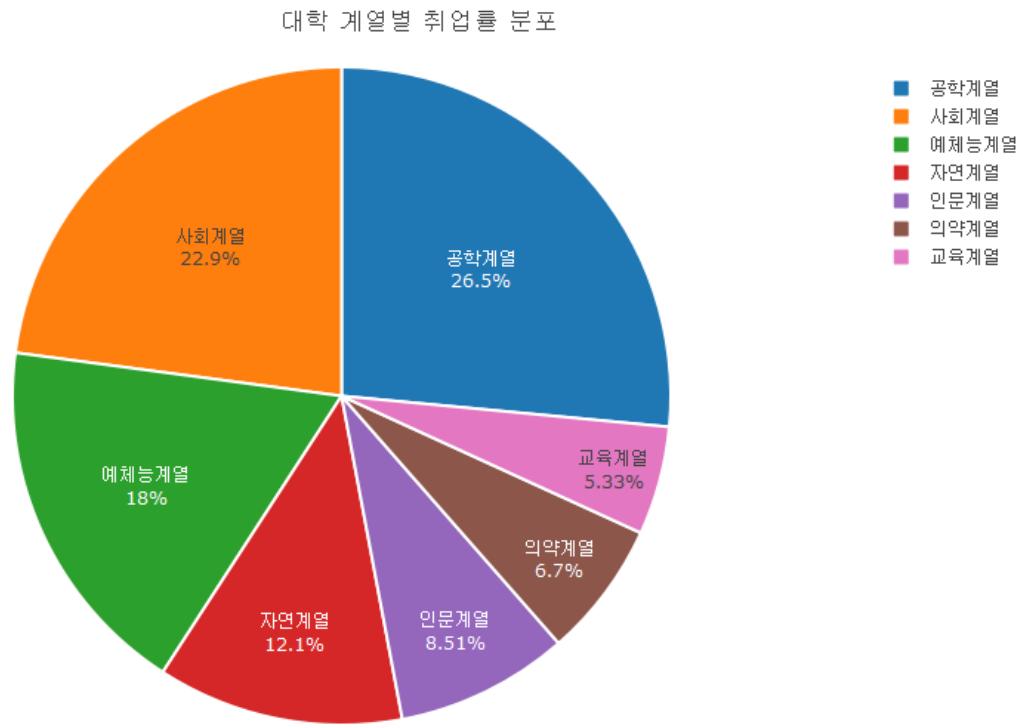
```
p_pie |>
  add_trace(type = 'pie', values = ~졸업자수, labels = ~대계열,
            textinfo = 'label+percent',
            ## hole 을 0.4 로 설정
            hole = 0.4) |>
  layout(title = list(text = '대학 계열별 취업률 분포'),
         margin = margins)
```



#### 2.2.2.6.3. marker

**marker** 는 파이 trace 로 표현되는 원의 형태를 설정하는 속성이다. 이 속성은 색(color)과 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 = 'viridis',
                          line = list(color = 'white',
                                      width = 2))
  ) |>
layout(title = list(text = '대학 계열별 취업률 분포'),
       margin = margins)
```



실행결과3- 66. marker 가 설정된 파이 trace

#### 2.2.2.6.4. pull, rotation, sort

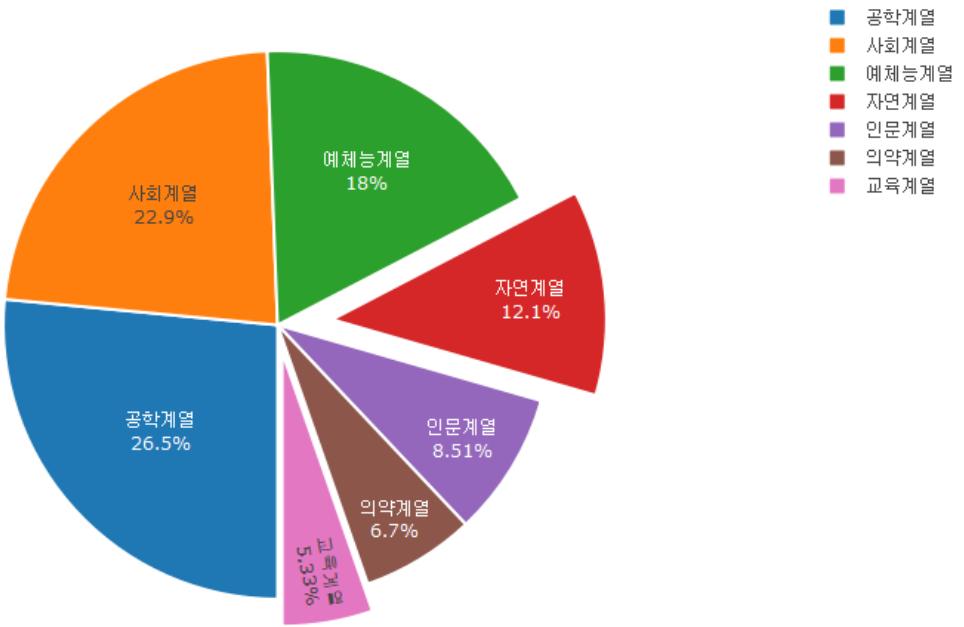
원 그래프를 그릴 때 특정 섹터를 강조하기 위해 살짝 밖으로 빼서 그리는 경우가 있다. 이러한 효과를 위해 사용하는 속성이 `pull` 이다. `pull` 은 0 부터 1 사이의 수치값을 설정하거나 수치값 벡터를 설정하는데 빠져나올 섹터의 반지름을 설정한다.

또 원 그래프를 그릴 때 처음 섹터를 시작하는 각도 지정을 위한 속성이 `rotation` 이고 섹터의 위치를 설정할 때 값의 정렬 순서대로 표시하는 속성이 `sort` 이다. `sort` 는 기본값이 TRUE 이기 때문에 특별한 설정이 없다면 데이터 값의 순서대로 표시되며 반시계 방향으로 표시된다. 이를 시계방향으로 설정하는 속성이 `direction` 이다.

```
p_pie |>
  add_trace(type = 'pie', values = ~졸업자수, labels = ~대계열,
             textinfo = 'label+percent',
             marker = list(line = list(color = 'white', width = 2)),
             ## pull 의 설정
             pull = c(0,0.1,0,0,0,0,0.2),
             ## rotation, sort, direction 의 설정
             rotation = 180, sort = TRUE, direction = 'clockwise') |>
```

```
layout(title = list(text = '대학 계열별 취업률 분포'),
      margin = margins)
```

대학 계열별 취업률 분포



### 실행결과 3- 67. 섹터가 강조된 파이 trace

#### 2.2.2.6.5. *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 `plotly`에서 자동적으로 계산하여 표시하는데 내부에 표시되는 문자열의 형태를 설정하는 속성이 `insidetextfont`이고 밖에 표시되는 문자열의 형태를 설정하는 속성이 `outsidetextfont`이다. 이들 속성은 모두 `color`, `family`, `size`의 세가지 세부 속성을 사용하여 설정한다. 또 이들 문자열이 표시되는 방향을 설정하는 속성이 `insidetextorientation`과 `outsidetextorientation`이다. 이 속성 값은 문자열이 표시되는 방향을 가로 형태로 설정하는 'horizontal', 섹터의 반경을 따라 표시되는 'radial', 섹터 반경의 수직으로 표시되는 'tangential', `plotly`가 자동으로 설정하는 'auto'의 네 가지이다.

```
p_pie |>
  add_trace(type = 'pie', values = ~졸업자수, labels = ~대계열,
            textinfo = 'label+percent',
```

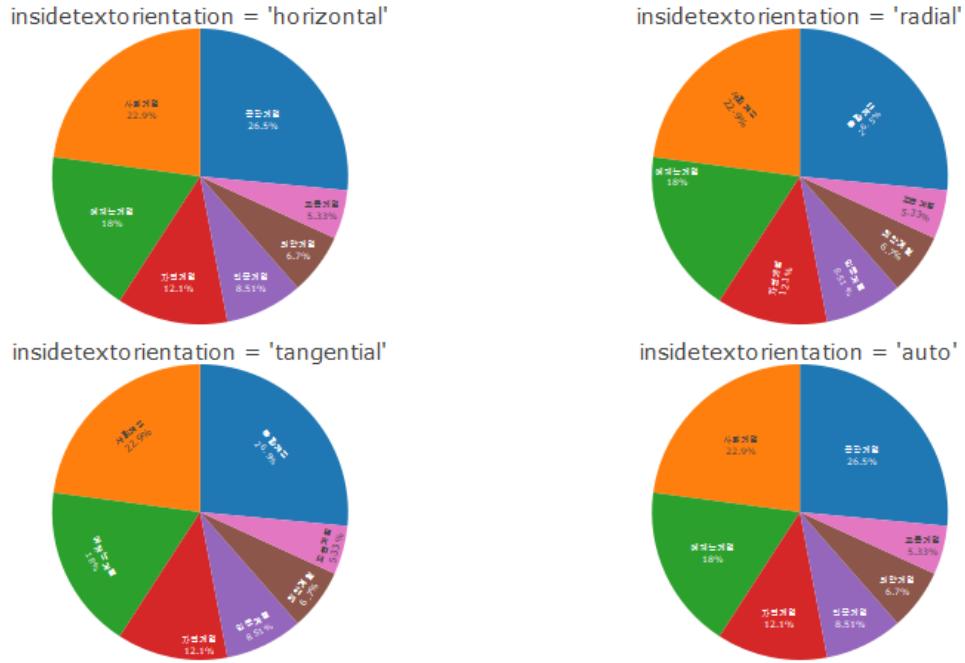
```
## insidetextorientation 을 horizontal 로 설정
insidetextorientation = 'horizontal') |>
layout(title = list(text = '대학 계열별 취업률 분포'),
       margin = margins)

p_pie |>
add_trace(type = 'pie', values = ~졸업자수, labels = ~대계열,
          textinfo = 'label+percent',
          ## insidetextorientation 의 radial 로 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',
          ## insidetextorientation 의 tangential 로 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',
          ## insidetextorientation 의 auto 로 설정
          insidetextorientation = 'auto') |>
layout(title = list(text = '대학 계열별 취업률 분포'),
       margin = margins)
```

### 대학 계열별 취업률 분포



실행결과3- 68. `insidetextorientation` 설정 결과

#### 2.2.2.7. 인디케이터(indicator) trace : 지수 차트

인디케이터는 단일 수치에 대한 시각화 방법이다. `plotly`에서 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 trace 를 사용하기 위해서는 `add_trace()`의 속성을 'indicator'로 설정함으로써 사용할 수 있다.

인디케이터 trace 에서 사용하는 주요 속성은 다음과 같다.

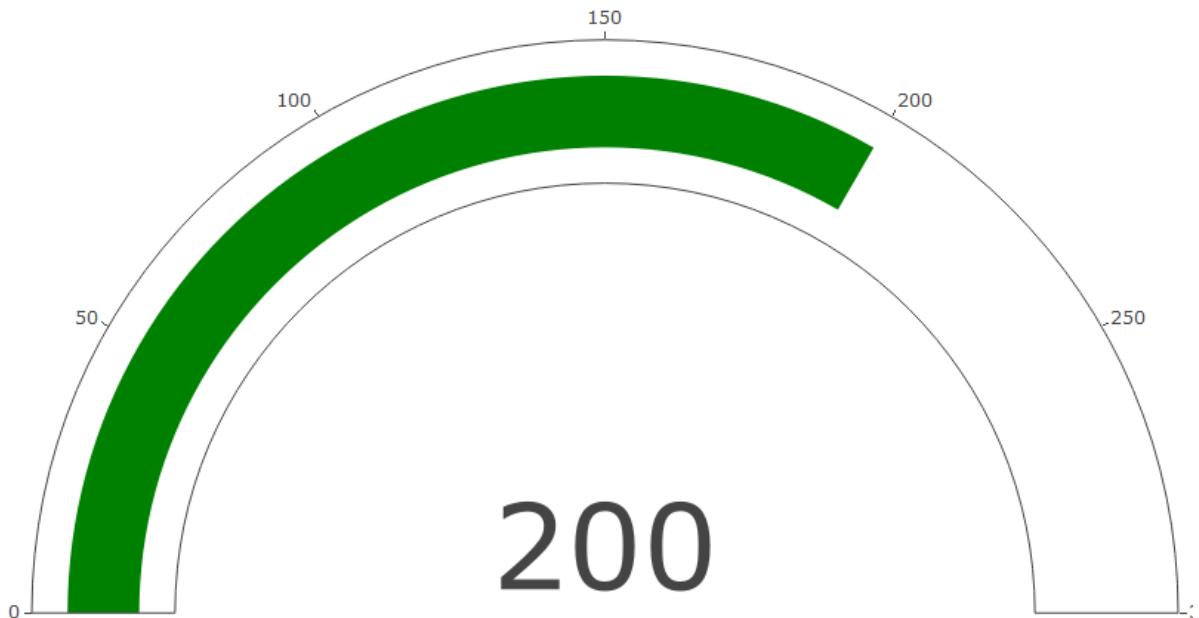
속성	설명	속성 값	세부 속성
type	trace type 속성	'indicator'	
visible	trace 의 표시 설정	TRUE, FALSE, 'legendonly'	
title	인디케이터 trace 의 제목 설정	세부 속성들의 리스트	font(color, family, size), position, text
align	인디케이터 박스 안의 문자열 정렬 설정	'left', 'center', 'right'	
delta	증감량을 표시하는 인디케이터 설정	세부 속성들의 리스트	decreasing(color, symbol), font(color, family, size), increasing(color, symbol), position, reference, relative, valueformat
number	수치를 표시하는 인디케이터 설정	세부 속성들의 리스트	font(color, family, size), prefix, suffix, 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, ...

#### 2.2.2.7.1. mode, value

인디케이터 trace 에서 제공하는 표시 방법은 수치 표시, 증감량 표시, 게이지 표시의 세 가지 방법을 제공한다. 이들을 설정하는 속성이 `mode` 이다. 이 `mode` 는 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 `value` 로 설정하고 단일 수치를 설정한다.

```
plot_ly() |>
  add_trace(type = 'indicator', mode = "gauge+number", value = 200)
```



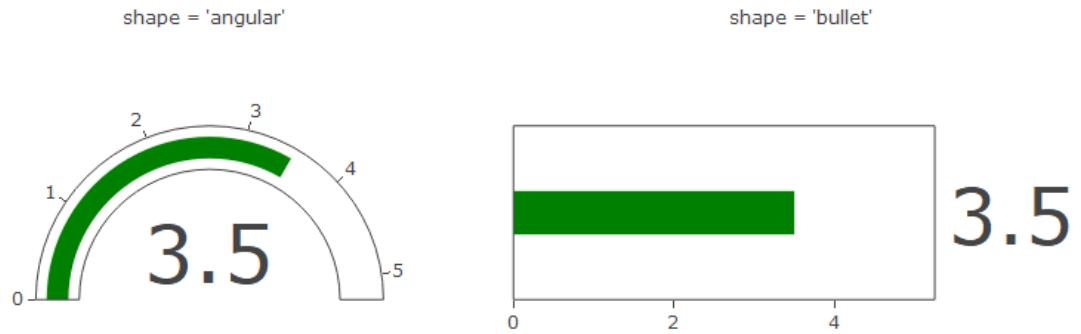
실행결과 3-69. 인디케이터 trace 생성

#### 2.2.2.7.2. *gauge*

**gauge** 는 게이지 형태의 시각화를 통해서 단일 수치를 표현한다. 게이지의 형태는 둥근 반원 형태의 게이지를 사용하는 'angular'와 막대 형태의 게이지를 사용하는 'bullet'의 두 가지가 제공된다. 이는 **shape** 속성을 사용하여 설정이 가능하다.

```
plot_ly() |>
  add_trace(type = 'indicator', mode = "gauge+number",
            ## gauge 의 shape 를 angular 로 설정
            gauge = list(shape = 'angular'),
            value = 3.5)

plot_ly() |>
  add_trace(type = 'indicator', mode = "gauge+number",
            ## gauge 의 shape 를 bullet' 로 설정
            gauge = list(shape = 'bullet'),
            value = 3.5)
```



### 실행결과 3- 70. angular 와 bullet 인디케이터 trace

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

```
plot_ly() |>
  add_trace(type = 'indicator', mode = "gauge+number",
            ## 인디케이터 제목 설정
            title = list(text = '조사 만족도',
                         font = list(size = 15),
                         align = 'right'),
            ## gauge 의 세부 속성 설정
            gauge = list(shape = 'bullet',
                         ## 전체 범위는 0~5 로 설정
                         axis = list(range = c(0, 5)),
                         ## 각 단계별로 range, color, line 의 속성 설정
                         steps = list(
                           list(range = c(0,1), color = "lightgreen",
                                line = list(color = 'white', width = 5)),
                           list(range = c(1,2), color = "lime",
                                line = list(color = 'white', width = 5)),
                           list(range = c(2,3), color = "yellow",
                                line = list(color = 'black', width = 5)),
                           list(range = c(3,4), color = "orange",
                                line = list(color = 'black', width = 5)),
                           list(range = c(4,5), color = "red",
                                line = list(color = 'black', width = 5))))
```

```

        list(range = c(2,3), color = "yellow",
              line = list(color = 'white', width = 5)),
        list(range = c(3,4), color = 'orange',
              line = list(color = 'white', width = 5)),
        list(range = c(4,5), color = "darkred",
              line = list(color = 'white', width = 5))
      ),
## threshold(임계치) 세부 속성 설정
threshold = list(line = list(color = 'yellow', width =
3),
                  value = 4.5)
),
value = 3.5) |>
## 가로로 길게 시각화 크기 설정
layout(height = 100, width = 800, margin = list(l= 120, r= 10))

```



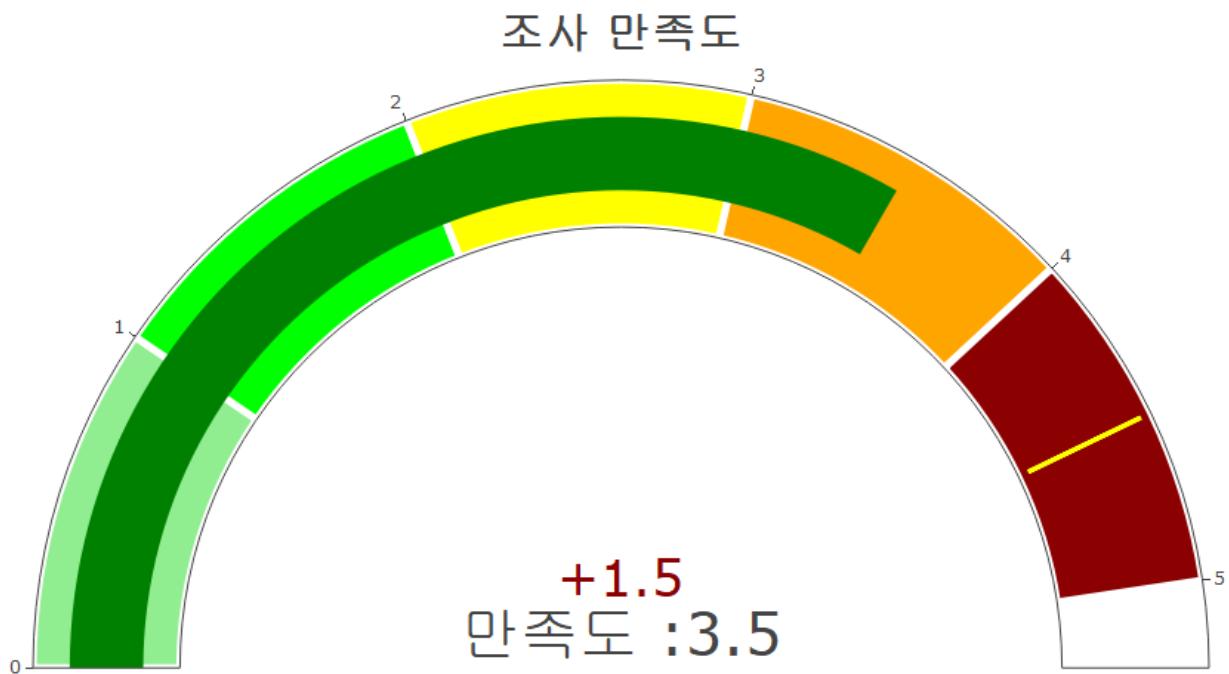
### 실행결과 3- 71. 인디케이터 trace 의 step 설정

#### 2.2.2.7.3. delta

**delta** 는 데이터의 증감량을 나타내는 인디케이터를 표시한다. 가장 흔히 보이는 곳이 주식관련 시각화인데 전일 대비 주가를 파란색, 빨간색 삼각형과 같이 나타내는 것이 **delta** 이다. 증감량을 표시해야 하기 때문에 증감의 기준이 되는 수치인 **reference** 가 설정되어야 하고 **increasing** 과 **decreasing** 을 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',
      ## steps 세부 속성 설정
      steps = list(
        list(range = c(0,1), color = "lightgreen",
          line = list(color = 'white', width = 5)),
        list(range = c(1,2), color = "lime",
          line = list(color = 'white', width = 5)),
        list(range = c(2,3), color = "yellow",
          line = list(color = 'white', width = 5)),
        list(range = c(3,4), color = 'orange',
          line = list(color = 'white', width = 5)),
        list(range = c(4,5), color = "darkred",
          line = list(color = 'white', width = 5))
      ),
      ## threshold 세부 속성 설정
      threshold = list(line = list(color = 'yellow',
        width = 3),
        value = 4.5)
      ),
      value = 3.5,
      ## delta 의 세부 속성 설정
      delta = list(reference = 2, 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 3- 72. steps 가 설정된 angular 형태의 인디케이터 trace

### 2.2.3. 레이아웃 설정 : layout()

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

#### 2.2.3.1. 제목 설정 : title

`plotly`의 제목을 설정하는 type은 `title`이다. `title`의 세부 type은 다음과 같다.

속성	설명	속성 값	세부 속성
font	제목 글꼴 설정	세부 속성들의 리스트	color, family, size
pad	제목 패딩(여백과 시각화 개체와의 거리) 설정	세부 속성들의 리스트	b, l, r, t
text	제목 문자열 설정	문자열	
x, y	제목의 위치	0 ~ 1 사이의 수치	
xanchor, yanchor	X, Y 방향의 정렬 방법	'auto', 'left', 'right', 'center'	
xref, yref	x, y 가 참조하는 범위 설정	'container', 'paper'	

## df\_취업통계\_2000 데이터를 사용하는 plotly 객체 생성

```
p_layout_1 <- df_취업통계_2000 |>
  plot_ly()
```

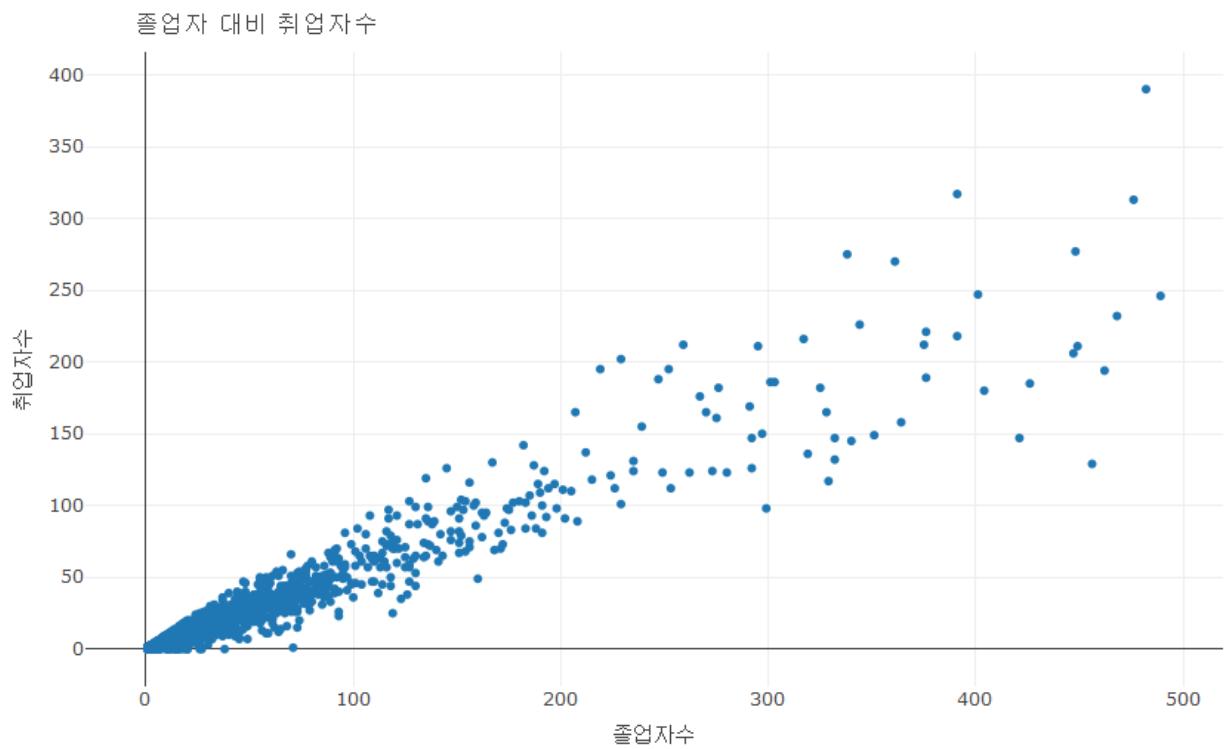
## markers 모드의 스캐터 trace 추가

```
p_layout_1 <- p_layout_1 |>
  add_trace(type = 'scatter', mode = 'markers',
            x = ~졸업자수, y = ~취업자수)
```

```
p_layout_1 |>
```

## Layout 의 title 세부 속성으로 시각화 제목 설정

```
layout(title = list(text = '졸업자 대비 취업자수', x = 0.1,
                    xanchor = 'left', yanchor = 'top'),
      margin= margins)
```



### 실행결과 3- 73. layout 을 이용한 시각화 제목 설정

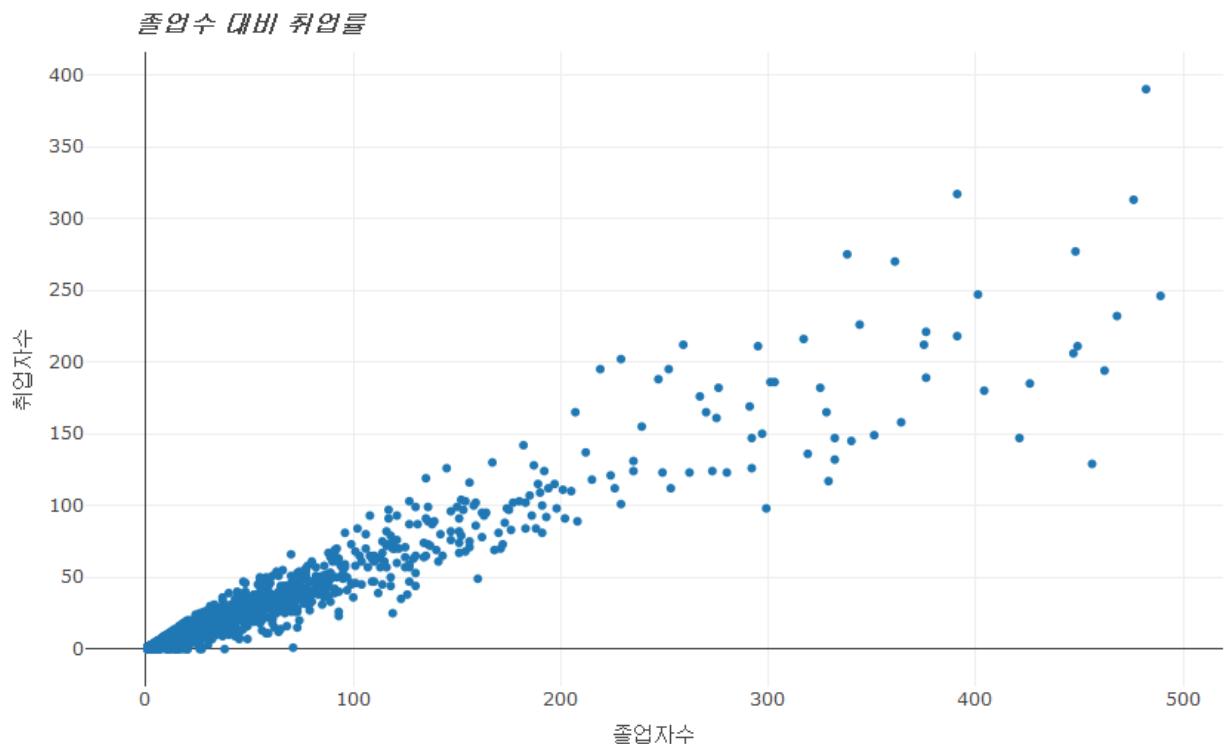
**plotly** 에서는 제목과 같은 문자열을 꾸미기 위해 HTML 텍스트 태그를 지원하는데 전체 HTML 중 5 가지만을 지원한다.

HTML tab	설명
<b></b>	볼드체 설정
<i></i>	이탤릭체 설정
 	줄 바꿈 설정
<sup></sup>	윗 첨자 설정
<sub></sub>	아래 첨자 설정
<a href='url'></a>	하이퍼링크 설정

p\_layout\_1 |>

## 문자열 강조를 위해 <b></b> 태그 사용

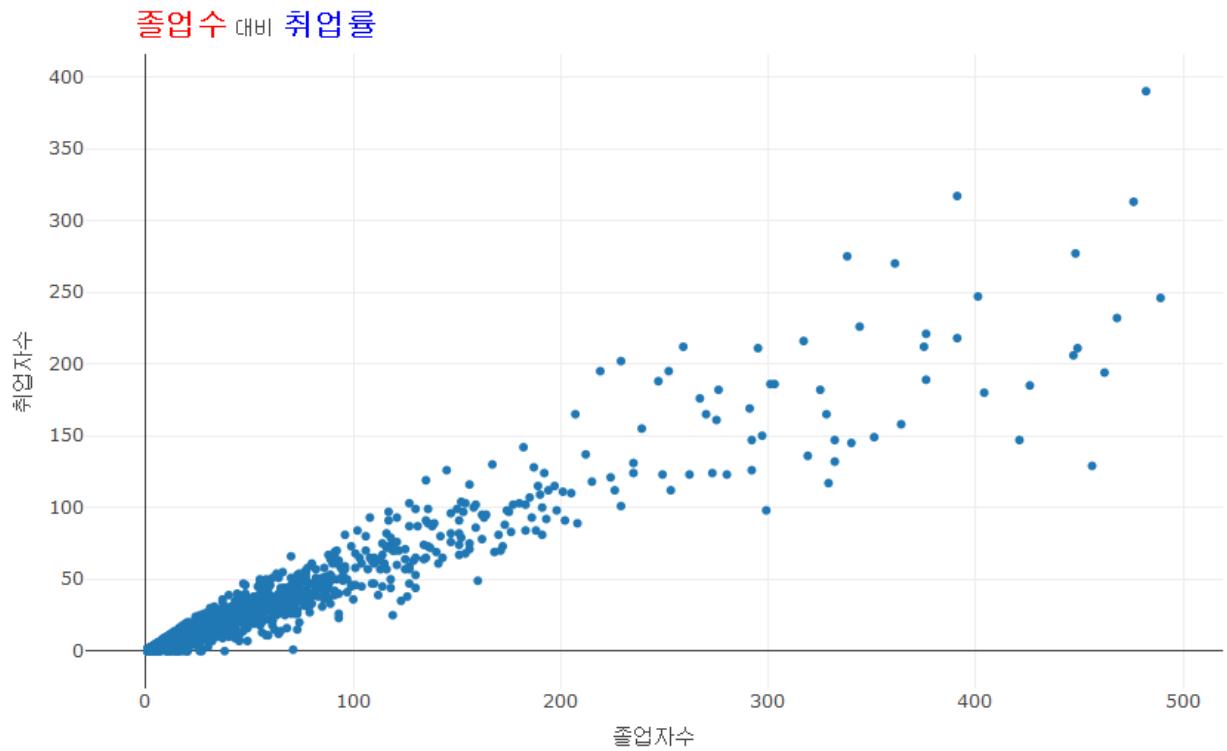
```
layout(title = list(text = "<b><i>졸업수 대비 취업률</i></b>",
                    x = 0.1, xanchor = 'left', yanchor = 'top'),
      margin = margins
    )
```



### 실행결과3- 74. html 태그를 사용한 제목 설정

`plotly` 가 HTML 텍스트 tag 를 일부만 지원함으로써 문자열 스타일링에 한계가 있을듯 하지만 ”을 사용하는 HTML inline 속성을 지원하기 때문에 CSS 의 스타일을 사용하여 문자열의 세부 설정이 가능하다. 다음은 HTML 의 inline 속성을 사용하여 제목을 설정하는 코드이다.

```
p_layout_1 |>
## <span></span>을 사용한 제목 설정
layout(title = list(text = "<span style = 'font-size:15pt'><span style = 'color:red;font-weight:bold;'> 졸업수</span><span style = 'font-size:10pt'> 대비</span> <span style = 'color:blue;font-weight:bold;'>취업률</span></span>",  
           x = 0.1, xanchor = 'left', yanchor = 'top'),  
       margin = margins  
)
```



실행결과3- 75. CSS 스타일의 제목 설정

### 2.2.3.2. 폰트 설정 : font

`layout()`에서 플롯 전체적인 문자열의 설정과 관련된 type 은 `font` 이다. `font` 는 다음과 같은 세 개의 세부 type 을 설정할 수 있다.

속성	설명	속성 값	세부 속성
color	플롯 전체 문자 색 설정	컬러 값	
family	플롯 전체 문자 폰트 설정	폰트명	
size	플롯 전체 문자 크기 설정	1 이상의 수치값	

전체적인 폰트의 설정은 다음과 같이 설정할 수 있다. `ggplot2` 에서는 한글 폰트의 설정에 다소 어려움이 있었지만 `plotly` 에서는 `family` type 에 바로 폰트 이름을 설정하면 쉽게 한글 폰트를 사용할 수 있다.

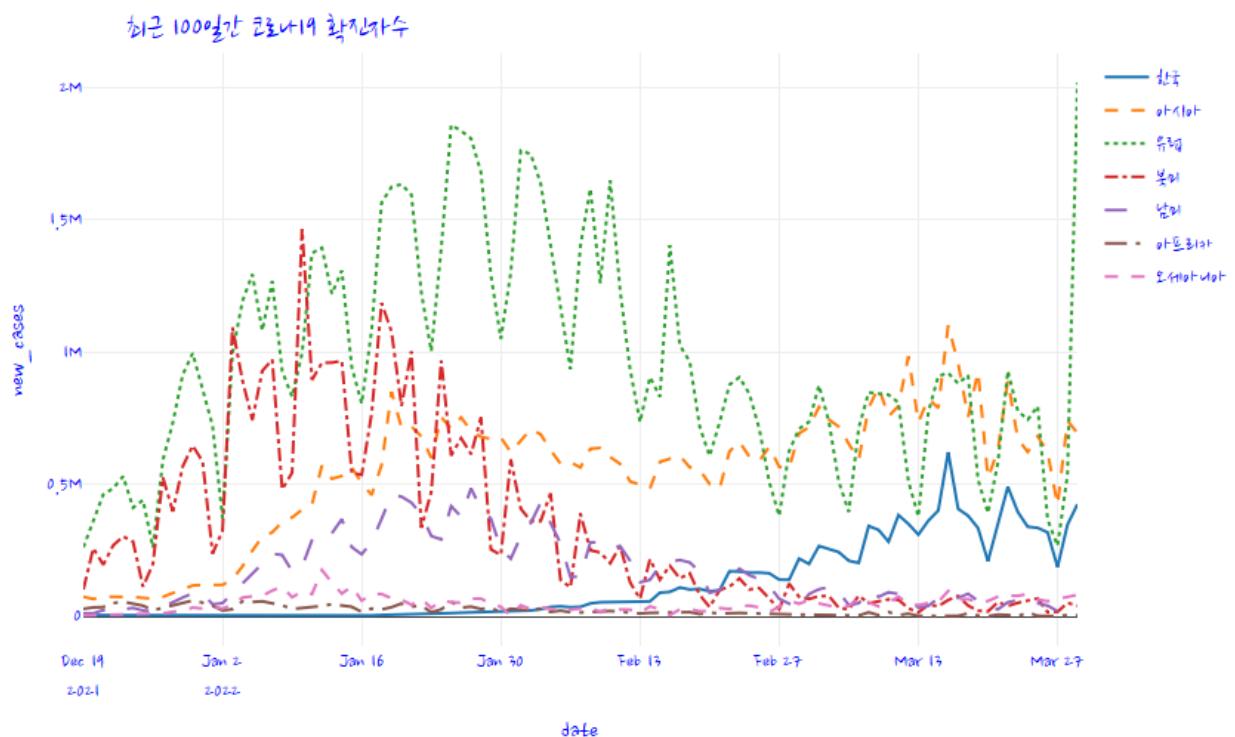
```
p_layout_2 <- covid19_df_100 |>
## mode 가 lines 인 스캐터 trace plotly 객체 생성
plot_ly(type = 'scatter', mode = 'lines', x = ~date,
```

```

y = ~new_cases, linetype = ~location)

p_layout_2 |>
  layout(title = list(text = '최근 100 일간 코로나19 확진자수', x = 0.1,
                      xanchor = 'left', yanchor = 'top'),
         ## font 의 세부 속성 설정
         font = list(family = "나눔손글씨 봄", color = 'blue',
                     size = 15),
         margin = margins
      )

```



실행결과3- 76. layout 을 사용한 폰트 설정

### 2.2.3.3. 축 설정 : xaxis, yaxis

**plotly** 에서 축은 trace type 에 따라 매우 다른 이름으로 불린다. X, Y 축을 사용하는 2 차원의 데카르트 좌표계에서는 ‘axis’, 3 차원 trace 에서는 ‘scene’, 극 좌표계에서는 ‘polar’, 삼각축 좌표계(ternary)에서는 ‘ternary’, 지형(Geo) 좌표계에서는 ‘geo’, Mapbox 좌표계에서는 ‘mapbox’, 색 좌표계에서는 ‘coloraxis’로 사용된다. 이 절에서는 이 중 2 차원 데카르트 좌표계에서 사용되는 **xaxis** 와 **yaxis** 를 위주로 설명하겠다.

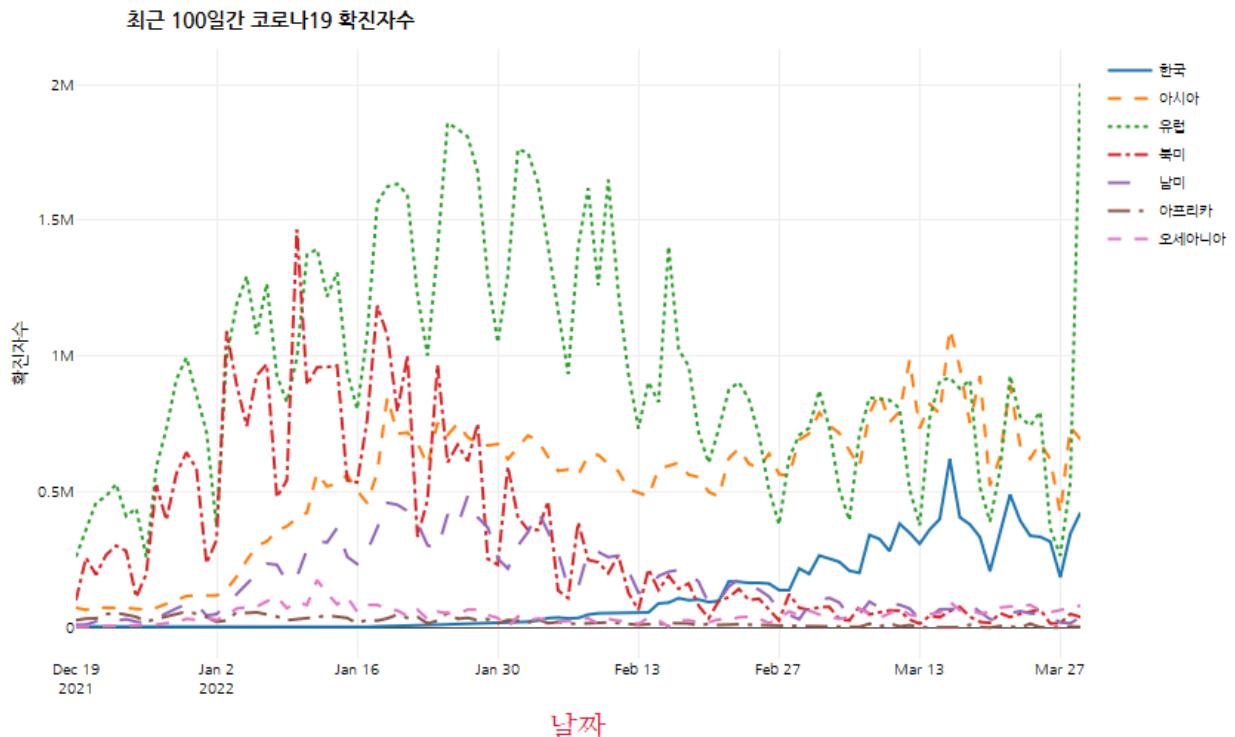
`layout()`에서 X 축의 설정과 관련된 type 은 `xaxis` 이다. `xaxis` 에서 제공하는 주요 type 은 다음과 같다.

속성	설명	속성 값	세부 속성
title	축 이름 설정	세부 속성들의 리스트	font(color, family, size), standoff, text
type	축 유형 설정	'linear', 'log', 'date', 'category', 'multicategory'	
range	축 범위 설정	리스트	
rangemode	축 범위의 특성 설정	'normal', 'tozero', 'non-negative'	
linecolor	축 선 색 설정	컬러 값	
linewidth	축 선 두께 설정	0 이상의 수치값	
zeroline	0 선 표시 설정	논리값	
zerolinecolor	0 선 색 설정	컬러 값	
zerolinewidth	0 선 두께 설정	0 이상의 수치값	
position	축 선 위치 설정	0 ~1 까지 수치값	
gridcolor	눈금선 색 설정	컬러 값	
gridwidth	눈금선 두께 설정	0 이상의 수치값	
autotypenumber	축에 설정되는 값이 문자형일때 숫자형으로 변환할지를 결정	'convert types', 'strict'	
ticks	눈금자를 어느쪽으로 그릴지 설정	'outside', 'inside'	
tickangle	눈금자의 각도 설정	각도 값	
ticklen	눈금자의 길이 설정	0 이상의 수치값	
tickcolor	눈금자의 색 설정	컬러 값	
tick0, dtick	눈금자의 시작점, 간격 설정	수치 또는 카테고리 좌표 문자열	
nticks	눈금자 개수 설정	0 이상의 수치값	

속성	설명	속성 값	세부 속성
tickvals, ticktext	눈금자의 위치, 표시문자 설정	데이터 프레임열, 리스트, 벡터	
showgrid, showline, showticklabel	눈금선, 축 선, 눈금자 라벨의 표시 여부 설정	논리값	
tickprefix, ticksuffix	눈금자 라벨의 접두어, 접미어 설정	문자열	

축 설정에 가장 많이 사용되는 속성중에 하나는 축 제목이다. 축 제목은 `title` 속성을 사용하여 설정이 가능한데 축 제목은 설정하는 두 가지 방법이 있다. 첫 번째는 `title`에 바로 문자열을 설정함으로써 해당 축의 제목을 설정하는 방법이다. 이 방법은 이전 버전의 `plotly`에서 사용되던 방법인데 이 방법을 사용하면 축 제목의 폰트 패밀리, 색, 크기를 설정할 수 없다. 만약 세부 설정을 해야한다면 `title`을 `list()`로 세부 속성 값을 리스트로 묶어 설정해야 한다. 첫 번째 방법은 쉽게 사용이 가능하지만 현재 버전의 `plotly`에서는 권장되지 않기 때문에 가급적 두번째 방법을 사용하는 것이 좋을 듯 하다.

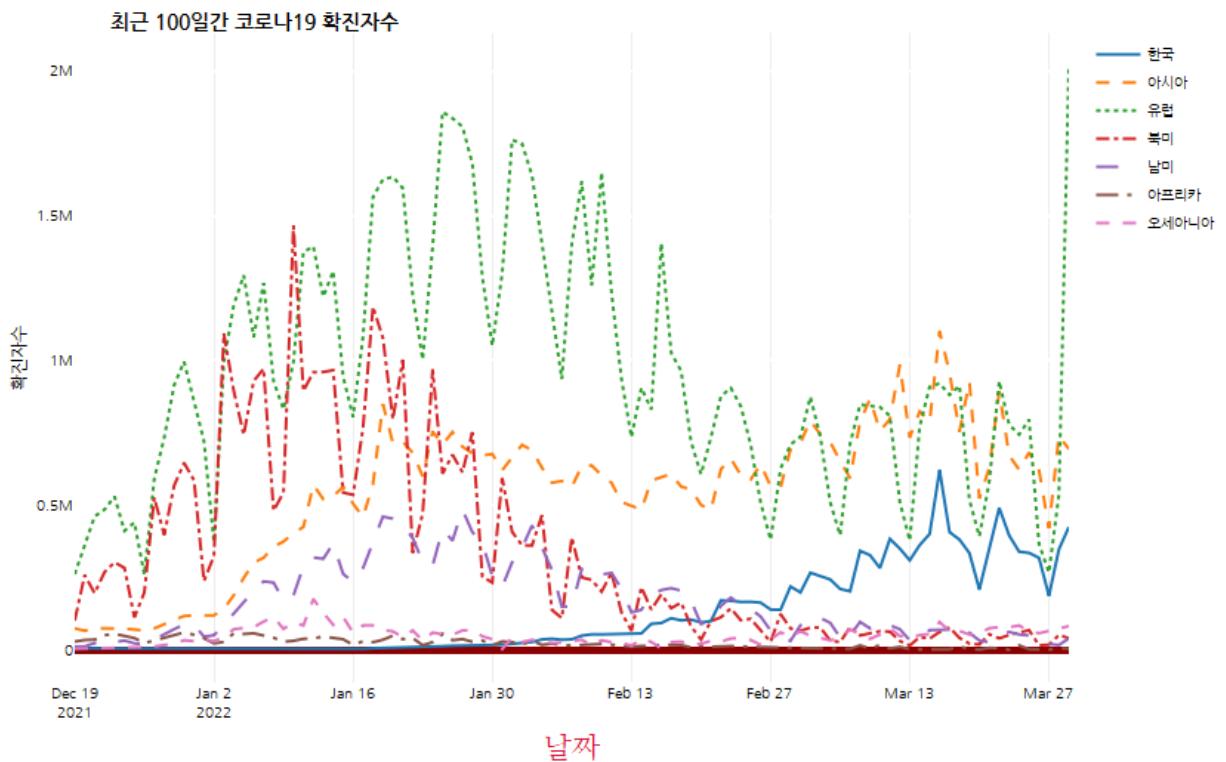
```
p_layout_2 |>
  layout(title = list(text = '<b>최근 100 일간 코로나 19 확진자수</b>', x = 0.1,
                    xanchor = 'left', yanchor = 'top', size = 15),
         font = list(family = "나눔고딕", color = 'black', size = 10),
         ## xaxis 제목 속성 설정 - 정상적 방법
         xaxis = list(title = list(text = '날짜',
                                    font = list(family = '나눔명조',
                                                size = 20, color = 'Crimson'))),
         ## yaxis 제목 속성 설정 - 약식 방법
         yaxis = list(title = '확진자수'),
         margin = margins
       )
```



### 실행결과3- 77. layout 을 사용한 축 제목 설정

다음의 예는 Y 축의 grid 와 zeroline 의 속성을 설정하는 예이다.

```
p_layout_2 |>
  layout(title = list(text = '<b>최근 100 일간 코로나 19 확진자수</b>',
                      x = 0.1, xanchor = 'left', yanchor = 'top', size = 15),
         font = list(family = "나눔고딕", color = 'black', size = 10),
         xaxis = list(title = list(text = '날짜',
                                   font = list(family = '나눔명조',
                                               size = 20, color = 'Crimson')
                                   )
                     ),
         yaxis = list(title = '확진자수',
                      ## grid 속성 설정
                      gridcolor = 'white', gridwidth = 2,
                      ## zeroline 속성 설정
                      zerolinewidth = 5, zerolinecolor = 'darkred')
         ))
```



### 실행결과 3- 78. layout 을 사용한 grid 와 zeroline 설정

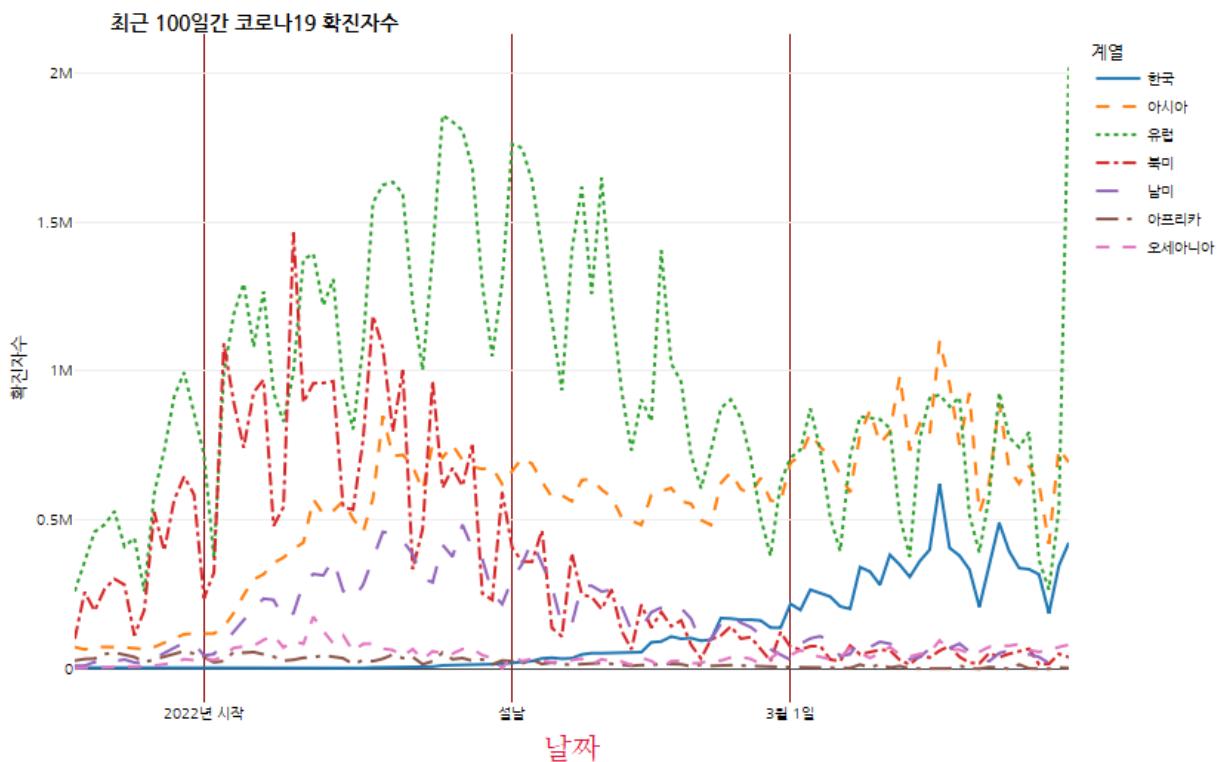
X 축과 Y 축의 눈금선은 `plotly` 에서 자동적으로 계산하여 가장 적절한 눈금선의 개수와 간격, 라벨을 설정한다. 그러나 사용자가 직접 원하는 눈금선을 설정하기 위해서는 `ticktext` 와 `tickvals` 를 사용하여 설정할 수 있다. 여기서 주의해야 할 점은 `ticktext` 와 `tickvals` 는 모두 벡터나 리스트로 여러 개의 눈금선을 설정할 수 있는데 두 벡터의 길이가 같아야 한다는 것이고 `tickvals` 에 설정되는 벡터는 X 축에 매핑되는 데이터 변수의 데이터 탑과 동일한 탑이어야 한다는 점이다. 물론 이 두 가지 조건이 충족되지 않는다고 해도 에러를 내지는 않지만 사용자가 원하는 대로 표시되지는 않을 것이다.

```
p_layout_2 |>
  layout(title = list(text = '<b>최근 100 일간 코로나 19 확진자수</b>',
                      x = 0.1, xanchor = 'left', yanchor = 'top', size = 15),
         font = list(family = "나눔고딕", color = 'black', size = 10),
         legend = list(title = list(text = '계열', side = 'top')),
         xaxis = list(title = list(text = '날짜',
                                    font = list(family = '나눔명조',
                                                size = 20, color = 'Crimson'))),
         ## tickmode 설정
         tickmode = 'array',
```

```

## ticktext 를 벡터로 설정
ticktext = c('2022년 시작', '설날', '3월 1일'),
## tickvals 를 벡터로 설정
tickvals = c('2022-01-01', '2022-02-01', '2022-03-01'),
## grid 의 설정
gridwidth = 1, gridcolor = 'darkred'
),
yaxis = list(title = '확진자수')
)

```



실행결과3- 79. `layout` 을 사용한 눈금라벨 설정

#### 2.2.3.4. 범례 설정 : `legend`

`layout()`에서 범례 설정과 관련된 type 은 앞서 설명한 `showlegend` 와 `legend` 이다.

`showlegend` 는 `add_trace()`에서도 설정이 가능한 type 인데 `add_trace()`에서 설정하면 해당 trace 만을 범례에서 제거하고 `layout()`에서 설정하면 전체 범례를 제거하게 된다.

`legend` 는 범례의 세부 설정을 위한 세부 type 의 list 로 설정된다. `legend` 로 설정이 가능한 주요 세부 type 은 다음과 같다.

속성	설명	속성 값	세부 속성
bgcolor, bordercolor	범례의 배경색, 외곽선 색 설정	컬러 값	
borderwidth	범례를 둘러싸는 외곽선 색, 두께 설정	컬러 값, 0 이상의 수치	
font	범례의 문자열에 적용되는 문자 속성 설정	0 이상의 수치	color, family, size
groupclick	범례 아이템을 클릭할 때 범례 그룹의 동작 설정	'toggleitem', 'toglegroup'	
grouptitlefont	범례 아이템 그룹의 제목과 제목 설정	세부 속성들의 리스트	color, family, size
itemclick	범례 아이템을 클릭 할 때의 동작 설정	'toggle', 'toggleothers', FALSE	
itemdoubleclick	범례 아이템을 더블 클릭 할 때의 동작 설정	'toggle', 'toggleothers', FALSE	
orientation	범례 표시 방향의 설정	'v', 'h'	
title	범례 제목 설정	문자열	font(color, family, size), side, text
traceorder	범례 순서 설정	'reversed', 'grouped', 'normal'의 조합('+')	

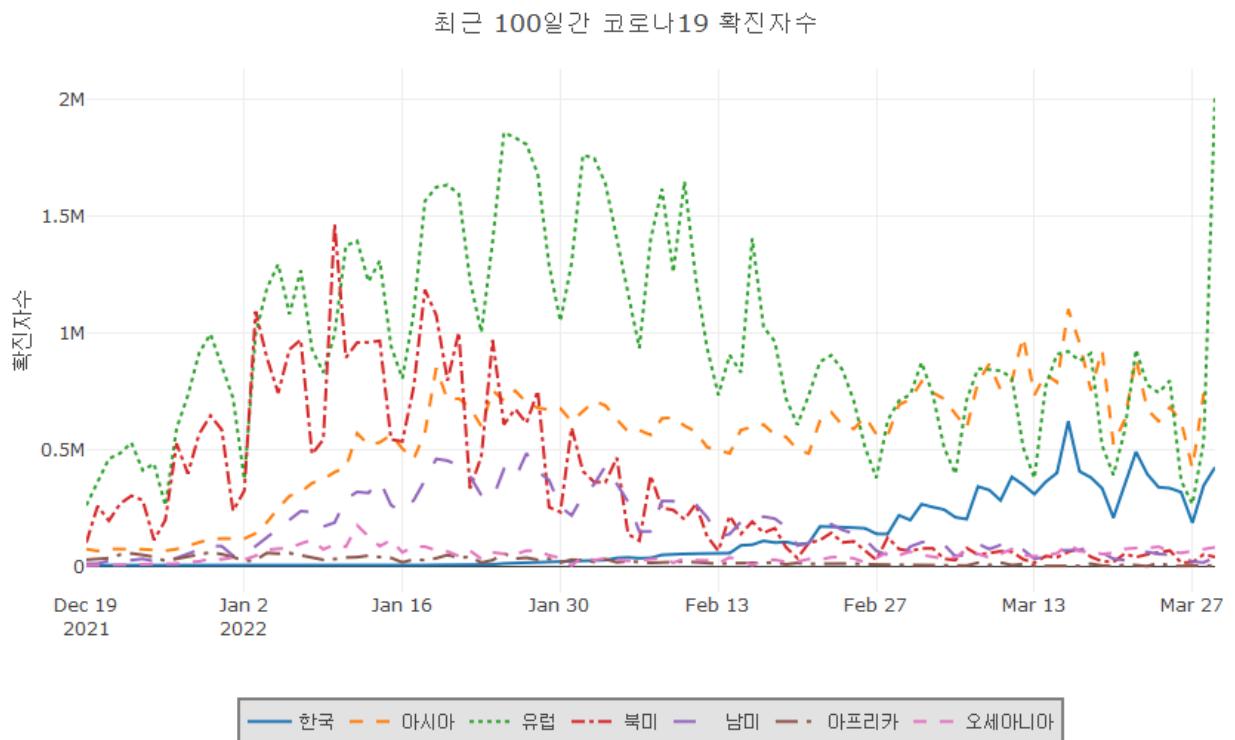
다음의 예는 범례를 아래 쪽으로 배치하고 범례 방향을 수평으로 범례 배경색, 범례 외곽선  
선과 색을 설정하는 예이다.

```
p_layout_2 |>
  layout(title = '최근 100 일간 코로나 19 확진자수',
         xaxis = list(title = list(text = NULL)),
         yaxis = list(title = list(text = '확진자수')),
         margin = margins,
         ## Legend 세부 속성을 사용한 범례 설정
         legend = list(
```

```

## 범례 배경색, 외곽선 색 설정
bgcolor = "#E2E2E2", bordercolor = 'gray',
## 범례 외곽선 두께, 표시 방향 설정
borderwidth = 2, orientation = 'h',
## X, Y 위치 설정과 위치 정렬 방법 설정
x = 0.5, y = -0.2, xanchor = 'center')
)

```



### 실행결과3- 80. layout 을 사용한 범례 설정

#### 2.2.3.5. 여백 설정 : margin

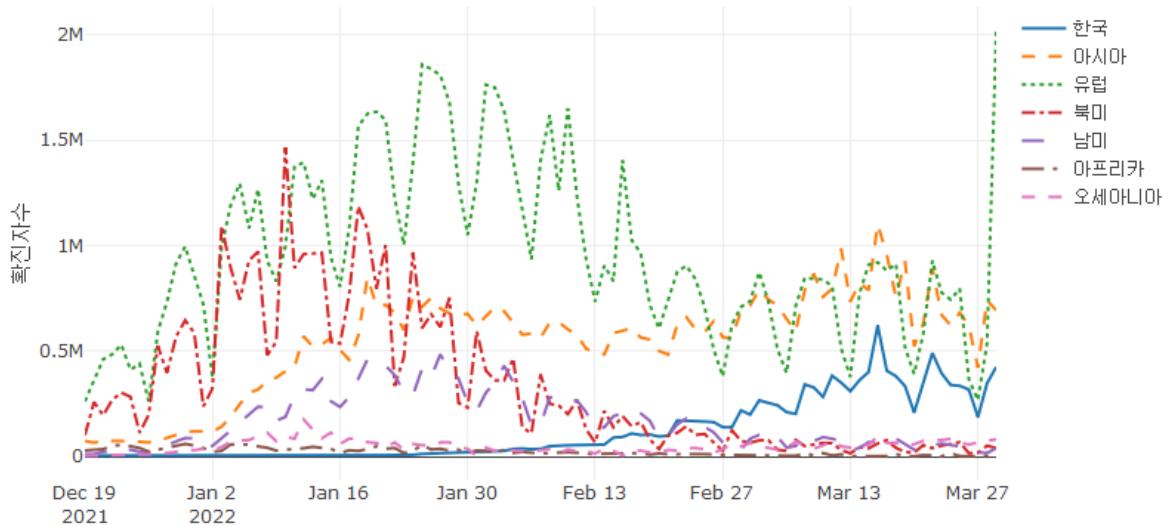
`layout()`에서 여백의 설정과 관련된 type 은 `margin` 이다. `margin` 으로 설정되는 여백은 플롯이 표시되는 면적과 전체 플롯 경계선과의 여백을 말한다. 여백 설정에 사용되는 세부 type 은 다음과 같다.

속성	설명	속성 값	세부 속성
autoexpand	여백 설정 값을 사용할지 설정	논리값	
b	아래쪽 여백 설정	0 이상의 수치값	
l	왼쪽 여백 설정	0 이상의 수치값	
r	오른쪽 여백 설정	0 이상의 수치값	
t	위쪽 여백 설정	0 이상의 수치값	
pad	플로팅 지역과 축과의 여백 설정	0 이상의 수치값	

앞선 `plot_ly()`와 `add_trace()`의 예를 보일 때 이미 `layout()`의 `margin` 속성이 사용되었다. 특히 제목이 설정된 `plotly` 객체에서는 이 여백을 적절히 설정해 주어야 전체 시각화가 보기 좋아진다.

```
p_layout_2 |>
  layout(title = '최근 100 일간 코로나 19 확진자수',
         xaxis = list(title = list(text = NULL)),
         yaxis = list(title = list(text = '확진자수')),
         ## margin 세부 속성을 사용한 여백 설정
         margin = list(t = 100, b = 100, l = 100, r = 100)
       )
```

최근 100일간 코로나19 확진자수

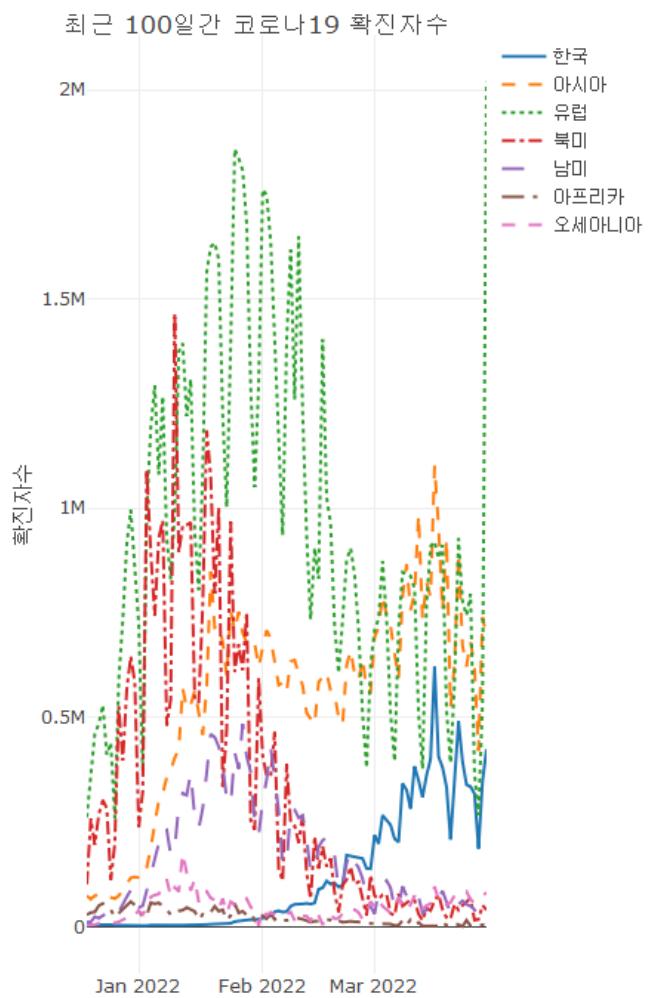


### 실행결과3- 81. layout 을 사용한 여백 설정

#### 2.2.3.6. 크기 설정 : size

`layout()`에서 전체 플롯 크기의 설정을 위해서는 `autosize`, `height`, `width` 의 세 가지 type 이 사용된다. `autosize` 는 사용자가 정의하지 않은 레이아웃의 너비 또는 높이를 자동적으로 설정하는 논리값을 설정한다. `height` 와 `width` 는 전체 플롯 레이아웃의 높이와 너비를 설정하는 type 으로 기본값이 450 과 700 이다.

```
p_layout_2 |>  
  layout(title = list(text = '최근 100 일간 코로나 19 확진자수',  
                  x = 0.1, xanchor = 'left', yanchor = 'top'),  
         xaxis = list(title = list(text = NULL)),  
         yaxis = list(title = list(text = '확진자수'))),  
 ## width, height 를 사용한 시각화 크기 설정  
         width = 450, height = 700)
```



실행결과 3- 82. *layout* 을 사용한 시각화 크기 설정

#### 2.2.3.7. 색 설정 : colors

`layout()`에서 플롯의 배경색이나 플롯에서 사용되는 전반적인 색 스케일을 설정하는 type 은 다음과 같다.

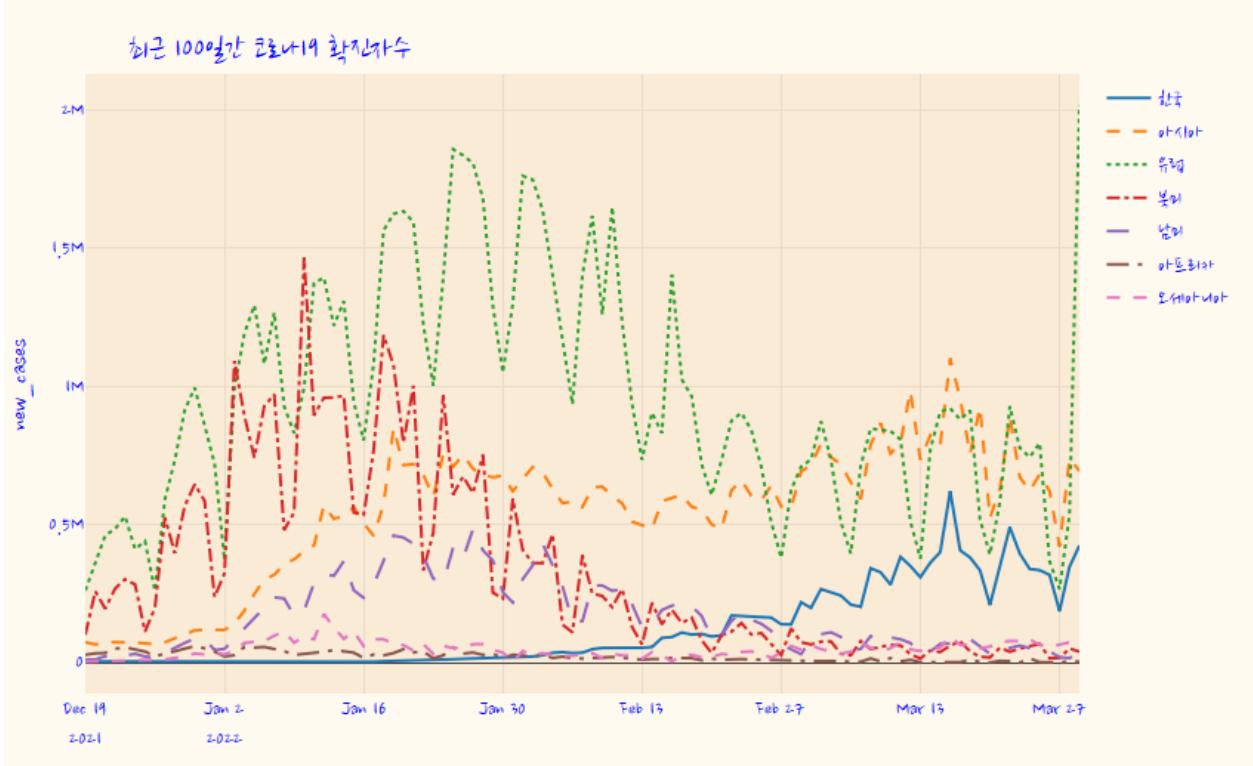
속성	설명	속성 값	세부 속성
paper_bgcolor	플롯이 그려지는 용지의 배경색을 설정	컬러 값	
plot_bgcolor	x 축과 y 축 사이의 플로팅 영역의 배경색 설정	컬러 값	
colorscale	컬러 스케일의 설정	세부 속성들의 리스트	diverging, sequential, sequentialminus
colorway	기본 컬러 벡터 설정	컬러 리스트	

`plotly`에서 사용하는 색의 설정은 색 이름 설정, 16 진수로 설정된 RGB 값 설정, `rgb()` 함수를 사용한 RGB 값의 설정 세 가지 방법이 주로 사용된다. 이외에도 색조, 채도, 명도(lightness)를 사용하는 `hsl()`을 사용하는 방법, 색조, 채도, 명도(value)를 사용하는 `hsv()`를 사용하는 방법도 있다. `plotly`에서 사용이 가능한 색 이름은 W3.org에서 제공하는 CSS 색 이름을 사용한다.<sup>6</sup>

```
p_layout_2 |>
  layout(title = list(text = '최근 100 일간 코로나 19 확진자수', x = 0.1,
                     xanchor = 'left', yanchor = 'top'),
         font = list(family = "나눔손글씨 봇", color = 'blue',
                     size = 15),
         xaxis = list(title = list(text = NULL)),
         ## paper_bgcolor, plot_bgcolor 의 속성 설정
         paper_bgcolor = 'floralwhite', plot_bgcolor = 'antiquewhite',
         margin = margins
       )
```

---

<sup>6</sup> [https://www.w3schools.com/cssref/css\\_colors.asp](https://www.w3schools.com/cssref/css_colors.asp)



실행결과3- 83. `layout` 을 사용한 배경색 설정

#### 2.2.3.8. 주석 설정 : annotations

시각화 그래프를 만들때 시각화를 만드는 목적에 따라 시각화 그래프에 데이터에 대한 추가적인 설명이 들어가는 경우가 많다. 이와 같이 시각화를 만드는 사용자가 강조하고자하거나 추가적으로 설명하고자 할때 사용하는 것이 주석(annotation)이다. `plotly`에서는 `layout()`에서 선과 화살표, 문자열을 사용하여 주석을 넣는 것이 가능하고 `add_annotation()`을 사용할 수도 있다. `layout()`을 사용할 때는 `annotations` type 을 사용하여 설정한다.

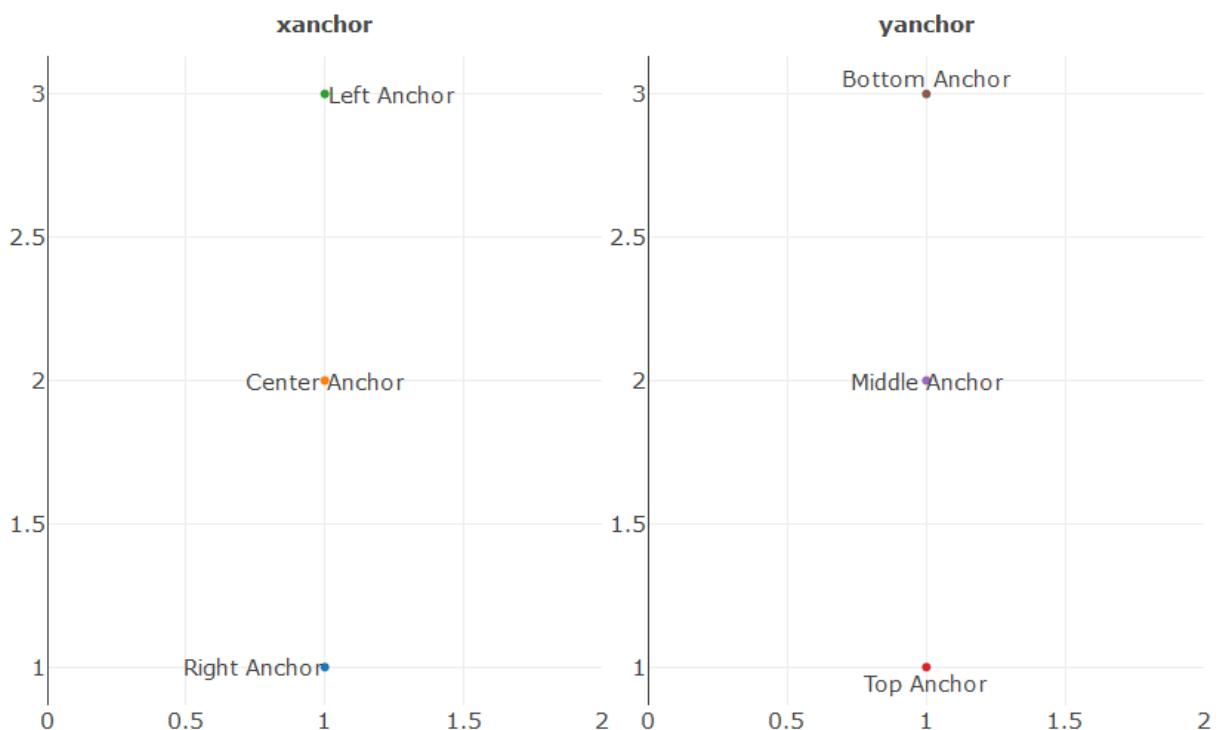
속성	설명	속성 값	세부 속성
text	주석으로 표시할 문자열을 설정	문자열	
x, y	주석을 달아야 할 X, Y 축 좌표 설정	수치 또는 카테고리 좌표 문자열	
ax, ay	주석의 화실표의 끝점 위치	수치 또는 카테고리 좌표 문자열	
xanchor, yanchor	X, Y 방향의 정렬 방법	'auto', 'left', 'center', 'right'	
xref, yref	x, y 가 참조하는 범위 설정	'x', 'y', 'paper'	
arrowcolor	화살촉의 색 설정	컬러 값	
arrowhead	화살촉의 모양 설정	0 ~ 8 까지의 수치값	
arrowside	화살촉 위치 설정	'end', 'start', 'none'의 조합('+)	
arrowsize	화살촉의 크기 설정	0.3 보다 큰 수치값	
arrowwidth	화살촉의 너비 설정	0.1 보다 큰 수치값	
bgcolor, bordercolor	주석의 배경색, 주석 외곽선 색 설정	컬러값	
borderpad, borderwidth	패딩(주석과 외곽선까지의 여백), 주석 외곽선 너비 설정	0 보다 큰 수치값	
font	주석의 문자 형태 설정	세부 속성들의 리스트	color, family, size

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

`x, y` 는 주석을 설정할 X 축과 Y 축의 좌표를 설정하는 type 이다. 주석은 `x` 와 `y` 에 변수를 매핑해서 사용하거나 하나하나 표시할 수 있다. 따라서 `x, y` 에 문자열이 표시될 좌표를 설정하거나 벡터나 변수를 매핑하여 주석을 표시할 수 있다.

`ax` 와 `ay` 는 `x`, `y` 에 설정된 좌표에 대한 주석이 실제로 위치할 좌표의 상대적 위치를 설정한다. 이 상대적 위치는 화살표로 표시되는데 `ax` 는 `x` 에 설정된 좌표에서의 거리를 설정하고 `ay` 는 `y` 에 설정된 좌표에서의 거리를 설정한다. `ay` 설정에 하나 주의할 점은 Y 축의 양의 방향은 아래에서 위로 향하지만 `ay` 의 양의 방향은 위에서 아래를 향하기 때문에 아래쪽에 위치한다.

`x`, `y`, `ax`, `ay` 에 의해 결정된 좌표에 `text` 로 설정한 문자열이 표시된다. 이 문자열의 위치 정렬을 설정하는 type 이 `xanchor` 와 `yanchor` 이다. `xanchor` 는 ‘auto’, ‘left’, ‘center’, ‘right’ 중에 하나의 값을 가지는데 `x` 에 설정된 좌표값이 전체 문자열의 왼쪽인지, 중앙인지, 오른쪽인지를 설정한다. 또 `yanchor` 는 ‘auto’, ‘top’, ‘middle’, ‘bottom’ 중에 하나의 값을 가지고 `y` 에 설정된 자표값이 전체 문자열의 위쪽인지, 중앙인지, 아래쪽인지를 설정한다.



실행결과 3- 84. anchor 설정에 따른 문자열 표시

`xref` 와 `yref` 는 `x`, `y`, `ax`, `ay` 의 절대 좌표와 상대 좌표의 참조 기준을 설정한다. `xref`, `yref` 가 ‘x’, ‘y’ 등의 축으로 설정되면 해당 축의 스케일에 맞게 `x`, `y`, `ax`, `ay` 의 좌표와 거리를 계산하고 ‘paper’로 설정되면 전체 플롯 영역을 0 부터 1 까지로 정규화한 상대적 거리로 계산된다.

2

xref = "paper", yref = "paper" 일 때 (0.5, 1)의 위치

1.5

xref = "x", yref = "y" 일 때 (0.5, 1)의 위치

1

0.5

-0.5

0

0.5

1

1.5

### 실행결과 3- 85. xref, yref 의 설정에 따른 위치 설정

다음의 예는 '졸업자 대비 취업자수' 산점도에서 '정보통신학과'의 위치를 확인하기 위한 예이다.

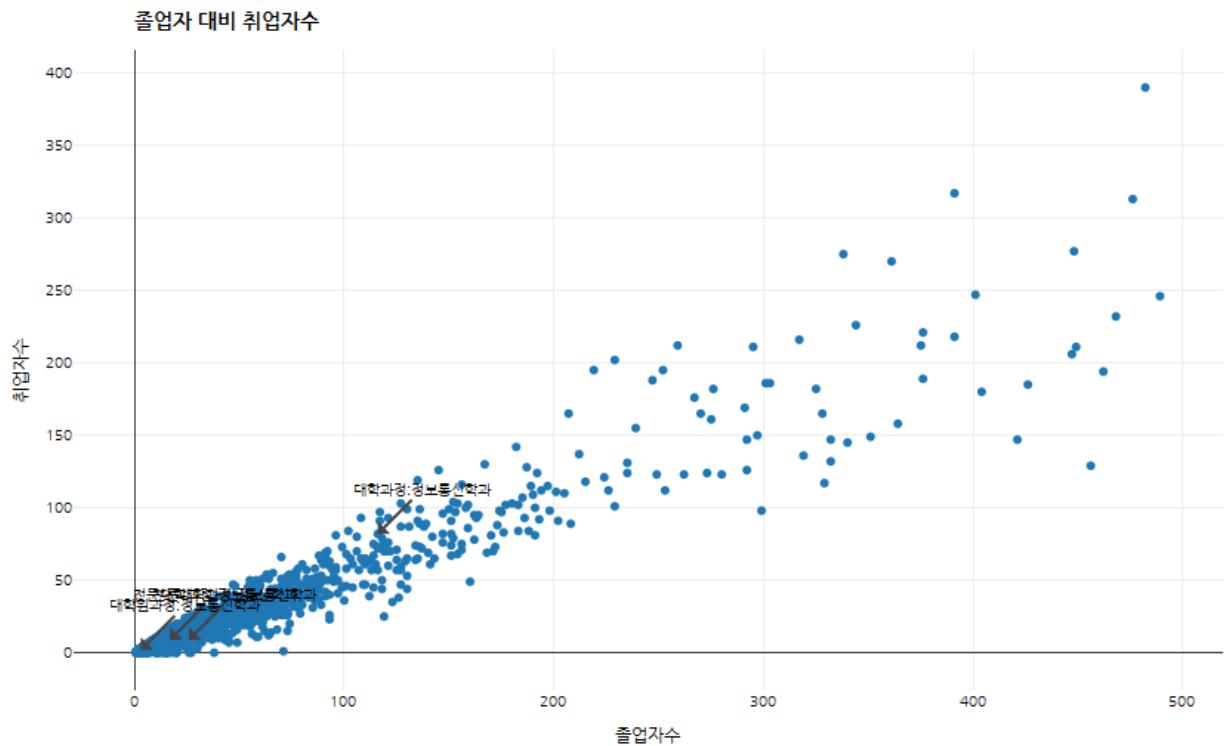
```
## 정보통신공학과 데이터만 필터링
df_annotation <- df_취업통계_2000 |> filter(학과명 == '정보통신학과') |>
  select(과정구분, 학과명, 졸업자수, 취업자수)

p_layout_1 |>
  layout(title = list(text = '<b>졸업자 대비 취업자수</b>',
                      x = 0.1, xanchor = 'left', yanchor = 'top', size = 15),
         font = list(family = "나눔고딕", color = 'black', size = 10),
         ## 정보통신공학과의 과정구분을 text로 맵핑
         annotations = list(text = ~paste0(df_annotation$과정구분, ':',
                                           df_annotation$학과명),
                           ## X, Y 축의 위치를 맵핑
                           x = df_annotation$졸업자수,
                           y = df_annotation$취업자수,
                           ## 화살표의 위치를 설정
```

```

    ax = 30, ay = -30),
margin = margins
)

```



### 실행결과 3-86. 특정 아이템 주석 달기

앞의 예에서 시각화 어디엔가에는 정보통신학과 데이터의 특성을 설명해 주면 효율적일 수 있다. 이를 위해 다음과 같이 제목 아래에 주석을 붙여주고 플로팅 영역을 적절히 조절해주었다.

```

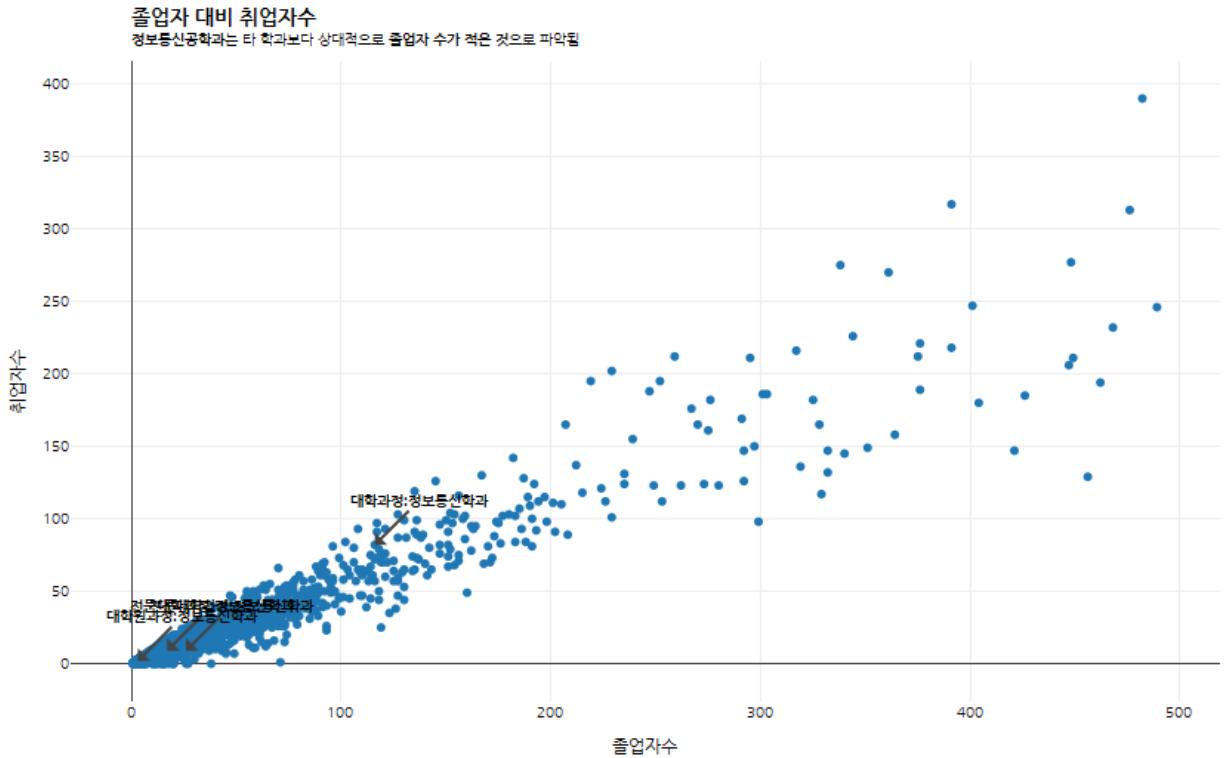
p_layout_1 |>
  layout(title = list(text = '졸업자 대비 취업자수', x = 0.1,
                     xanchor = 'left', yanchor = 'bottom', size = 15),
         font = list(family = "나눔고딕", color = 'black', size = 10),
         annotations = list(text = ~paste0('<b>', df_annotation$과정구분, ':',
                                           df_annotation$학과명, '</b>'),
                           x = df_annotation$졸업자수,
                           y = df_annotation$취업자수,
                           ax = 30, ay = -30),
         margin = margins) |>
  add_annotations(xref = 'paper', yref = 'paper',
                  text = '<b>정보통신공학과</b>는 타 학과보다 상대적으로 <b>졸업자')

```

```

수가 적은 것</b>으로 파악됨',
      x = 0.05, y = 1.05,
      showarrow = FALSE)

```



### 실행결과 3- 87. 시각화 세부 설명 문구 추가

앞의 예에서 보면 아래쪽의 정보통신학과는 위치의 차이가 크지 않아 주석들이 겹쳐진다. 이를 수정하기 위해서는 각각의 주석들을 설정하여야 한다. 다음의 예는 각각의 주석을 설정하면서 주석의 화살표 형태를 다르게 설정하였다.

```

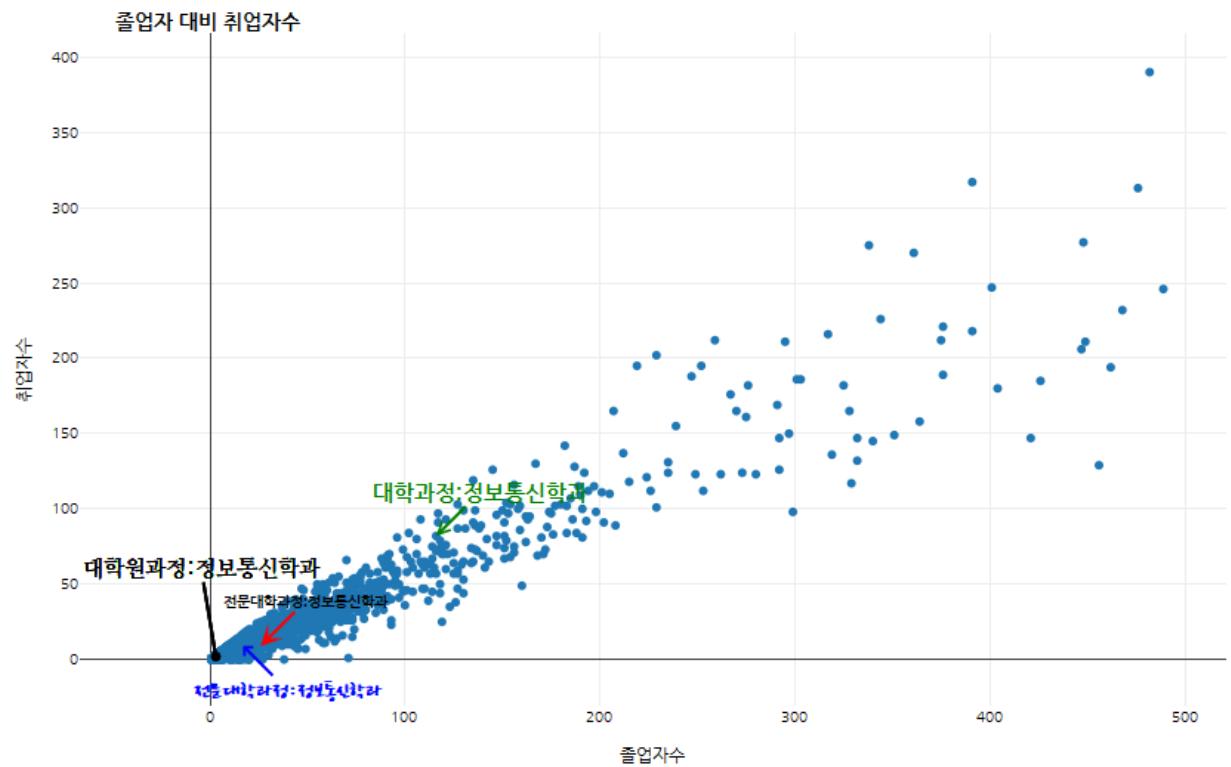
## 네개의 주석중에 세개는 layout 의 annotations 의 리스트로 설정
p_layout_1 |>
  layout(title = list(text = '<b>졸업자 대비 취업자수</b>', x = 0.1,
                    xanchor = 'left', yanchor = 'top', size = 15),
         font = list(family = "나눔고딕", color = 'black', size = 10),
         ## annotations 리스트 설정
         annotations = list(
           ## 첫 번째 주석의 annotations 세부 속성 설정
           list(text = ~paste0('<b>', df_annotation$과정구분[1], ':',
                             df_annotation$학과명[1], '</b>'),
                x = df_annotation$졸업자수[1],

```

```

y = df_annotation$취업자수[1],
ax = 30, ay = -30,
arrowhead = 3, arrowcolor = 'red',
arrowsize = 1.2
),
## 두 번째 주석의 annotations 세부 속성 설정
list(text = ~paste0('<b>', df_annotation$과정구분[2], ':', df_annotation$학과명[2], '</b>'),
      x = df_annotation$졸업자수[2],
      y = df_annotation$취업자수[2],
      ax = 30, ay = 30,
      arrowhead = 4, arrowcolor = 'blue',
      arrowsize = 0.8,
      font = list(family = '나눔손글씨 펜', size = 15, color = 'blue')
),
## 세 번째 주석의 annotations 세부 속성 설정
list(text = ~paste0('<b>', df_annotation$과정구분[3], ':', df_annotation$학과명[3], '</b>'),
      x = df_annotation$졸업자수[3],
      y = df_annotation$취업자수[3],
      ax = 30, ay = -30,
      arrowhead = 5, arrowcolor = 'green',
      arrowsize = 0.8,
      font = list(family = '나눔고딕', size = 15, color = 'green')
)
)
) |>
## 네 번째 주석의 annotations 을 add_annotations()로 설정
add_annotations(text = ~paste0('<b>', df_annotation$과정구분[4], ':', df_annotation$학과명[4], '</b>'),
      x = df_annotation$졸업자수[4],
      y = df_annotation$취업자수[4],
      ax = -10, ay = -60,
      arrowhead = 6, arrowcolor = 'black',
      arrowsize = 0.8,
      font = list(family = '나눔명조', size = 15, color = 'black'))

```



실행결과 3-88. 각각 주석의 세부 설정

#### 2.2.3.9. 도형 설정 : shapes

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

속성	설명	속성 값	세부 속성
type	그려질 도형의 형태 설정	'circle', 'rect', 'path', 'line'	
name	그려질 도형의 이름 설정	문자열	
x0, x1, y0, y1	도형의 X, Y 축 시작점(x0, y0)과 끝점(x1, y1) 설정	수치 또는 카테고리 좌표 문자열	
xref, yref	주석 도형이 그려지는 좌표의 참조 대상 설정	'paper', X 축 또는 Y 축 이름	
xsizemode, ysizemode	X, Y 축에 따른 크기 조절 모드 설정	'scaled', 'pixel'	
layer	그려지는 도형 레이어의 위치를 설정	'above', 'below'	
line	선 도형의 세부 설정	세부 속성들의 리스트	color, dash, width
filcolor	도형 내부 색 설정	컬러 값	

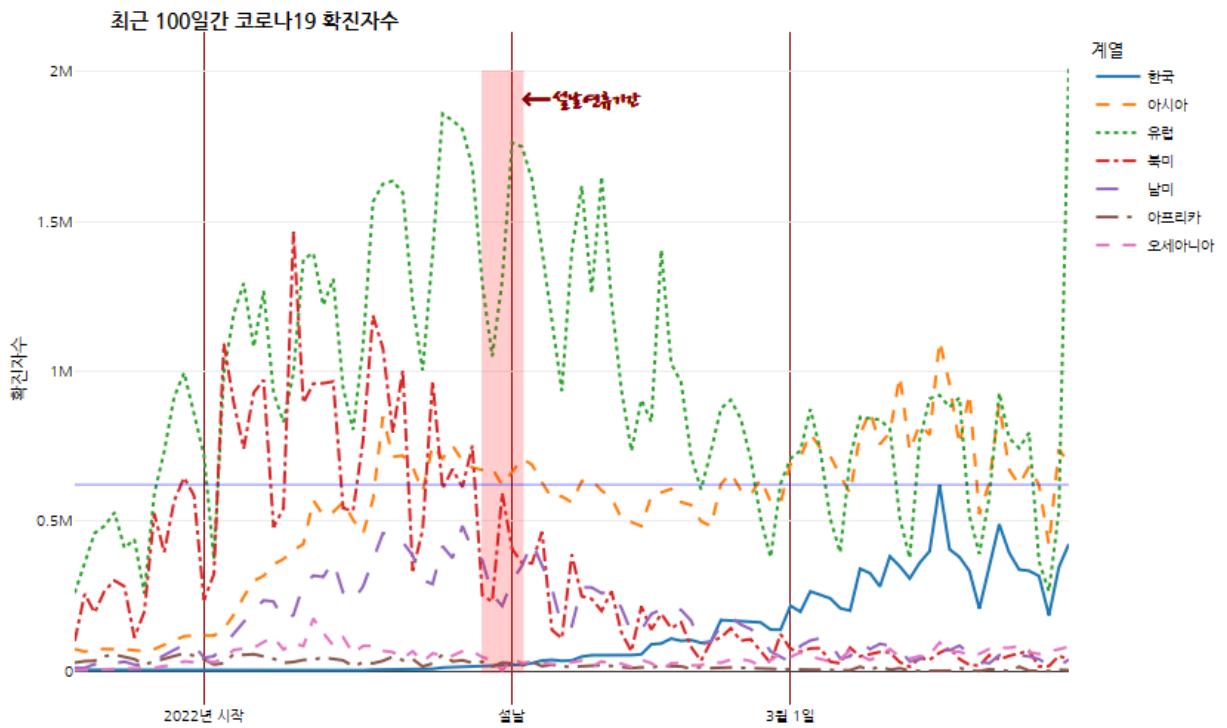
다음의 예는 최근 100 일간의 코로나 19 확진자수의 그래프에 주요한 시간적 이벤트를 표시한 그래프이다. 'rect' 타입의 **shape**로 설날 연휴기간을 표시하였고 가로선으로 한국 확진자의 최대치를 그려주었다.

```
p_layout_2 |>
  layout(title = list(text = '<b>최근 100 일간 코로나 19 확진자수</b>',
                      x = 0.1, xanchor = 'left', yanchor = 'top', size = 15),
         font = list(family = "나눔고딕", color = 'black', size = 10),
         legend = list(title = list(text = '계열', side = 'top')),
         xaxis = list(title = list(text = NULL),
                      tickmode = 'array',
                      ticktext = c('2022년 시작', '설날', '3월 1일'),
                      tickvals = c('2022-01-01', '2022-02-01', '2022-03-01'),
                      gridwidth = 1, gridcolor = 'darkred'
                     ),
         yaxis = list(title = '확진자수'),
         shapes = list(
           ## rect 타입을 위한 shapes 세부 속성 설정
           list(type = 'rect',
                 x0 = as.Date('2022-01-29'), x1 = as.Date('2022-02-02'),
                 y0 = 0, y1 = 2000000, fillcolor = "red",
                 line = list(color = 'black', width = 2)
               )
         )
       )
```

```

        line = list(color = "red"), opacity = 0.2),
## Line 타입을 위한 shapes 세부 속성 설정
    list(type = 'line', x0 = 0, x1= 1,
        y0 = max(covid19_df_100[covid19_df_100$iso_code == 'KOR', 6]),
        y1 = max(covid19_df_100[covid19_df_100$iso_code == 'KOR', 6]),
        xref = 'paper', yref = 'y',
        line = list(color = 'blue'), opacity = 0.3
    )
),
## 주석 설정
annotations = list(text = '<b>설날연휴기간</b>',
                    x = '2022-02-02', xref = 'x',
                    y = 0.9, yref = 'paper',
                    ax = 50, ay = 0,
                    arrowhead = 4, arrowcolor = 'darkred',
                    arrowsize = 0.8,
                    font = list(family = '나눔손글씨 펜',
                                size = 15, color = 'darkred')
)
)
)

```



실행결과3- 89. layout 을 사용한 도형 설정

## 2.2.4. 서브 플롯 : subplot()

데이터를 시각화할 때 하나의 시각화에 여러 개의 데이터를 표시하는 경우 각각의 데이터 trace 들이 너무 많아지거나 일부 구간에 중복되면 데이터의 정확한 확인이 어렵다. 이럴때는 각각의 데이터 trace 들을 따로 떼서 작은 시각화를 여러개 그려줌으로써 해결할 수 있다. 이렇게 여러개의 `plotly` 객체를 모아서 하나의 `plotly` 시각화로 만들어주는 함수가 `subplot()`이다. 앞서 `plotly` 의 설명 과정에서 여러개의 시각화가 모여서 그려진 사례는 모두 `subplot()`과 'annotation'을 사용해서 만들었다.

`subplot()`의 주요 사용법은 다음과 같다.

```
subplot(..., nrows = 1, widths = NULL, heights = NULL, margin = 0.02, shareX = FALSE, shareY =  
FALSE, titleX = shareX, titleY = shareY, which_layout = "merge")  
- ... : plotly 나 ggplot2 객체의 이름, 리스트, tibble  
- nrows : subplot 의 행의 개수  
- widths : 각각의 subplot 의 0 부터 1 사이의 상대적 열 너비  
- heights : 각각의 subplot 의 0 부터 1 사이의 상대적 열 높이  
- margin : 0 부터 1 사이의 단일 수치나 4 개의 수치(왼쪽, 오른쪽, 위, 아래)의 여백  
- shareX : X 축을 공유할지에 대한 논리값  
- shareY : Y 축을 공유할지에 대한 논리값  
- titleX : X 축의 제목을 남길지에 대한 논리값  
- titleY : Y 축의 제목을 남길지에 대한 논리값  
- which_layout : 어떤 플롯의 레이아웃에 적용할지 결정, 기본값은 "merge"로 뒤 플롯의 레이아웃  
옵션이 앞 옵션보다 우선함을 의미
```

`subplot()`은 subplot 으로 그릴 각각의 `plotly` 객체나 `ggplot2` 를 먼저 생성하고 이들 객체들을 `subplot()`을 사용하여 묶어 그려주는 형태로 사용된다. 이 과정에서 `shareX`, `shareY` 를 사용하여 X, Y 축의 공유를 설정하거나 `nrows` 를 사용하여 `subplot()`의 행 개수를 설정할 수 있다. 또 각각의 서브 플롯이 겹쳐지지 않도록 `margin` 을 설정할 수 있다.

```

## 서브플롯 생성을 위한 기본 plotly 객체 생성
p_line_wide <- covid19_df_100_wide |> plot_ly()

## 첫 번째 서브 플롯 생성
p1 <- p_line_wide |>
  add_trace(type = 'scatter', mode = 'lines', x = ~date,
            y = ~확진자_한국, color = I('#1f77b4')) |>
  layout(title = list(title = NULL),
         xaxis = list(tickfont = list(size = 10)),
         yaxis = list(title = list(text = '확진자수')))

## 두 번째 서브 플롯 생성
p2 <- p_line_wide |>
  add_trace(type = 'scatter', mode = 'lines', x = ~date,
            y = ~확진자_아시아, color = I('#1f77b4')) |>
  layout(title = list(title = NULL),
         xaxis = list(tickfont = list(size = 10)),
         yaxis = list(title = list(text = '확진자수')))

## 세 번째 서브 플롯 생성
p3 <- p_line_wide |>
  add_trace(type = 'scatter', mode = 'lines', x = ~date,
            y = ~확진자_유럽, color = I('#1f77b4')) |>
  layout(title = list(title = NULL),
         xaxis = list(tickfont = list(size = 10)),
         yaxis = list(title = list(text = '확진자수')))

## 네 번째 서브 플롯 생성
p4 <- p_line_wide |>
  add_trace(type = 'scatter', mode = 'lines', x = ~date,
            y = ~확진자_북미, color = I('#1f77b4')) |>
  layout(title = list(title = NULL),
         xaxis = list(tickfont = list(size = 10)),
         yaxis = list(title = list(text = '확진자수')))

## 다섯 번째 서브 플롯 생성
p5 <- p_line_wide |>
  add_trace(type = 'scatter', mode = 'lines', x = ~date,
            y = ~확진자_남미, color = I('#1f77b4')) |>
  layout(title = list(title = NULL),
         xaxis = list(tickfont = list(size = 10)),
         yaxis = list(title = list(text = '확진자수')))
```

```

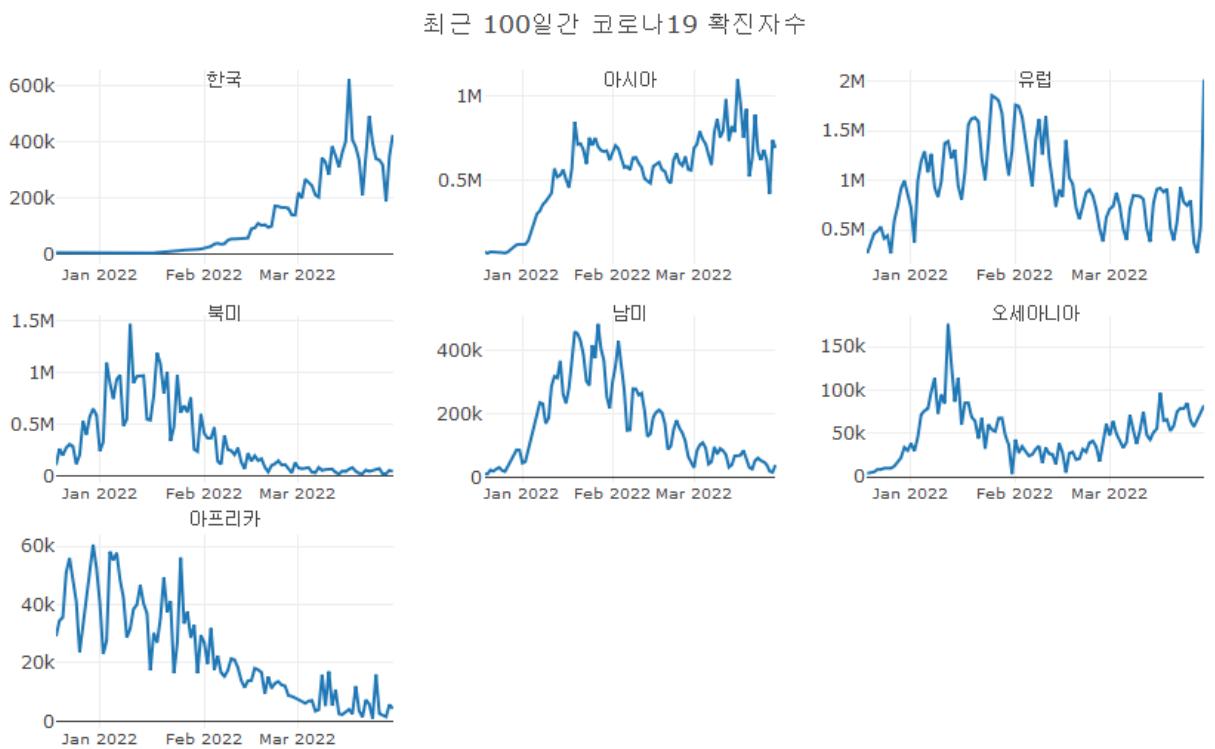
## 여섯 번째 서브 플롯 생성
p6 <- p_line_wide |>
  add_trace(type = 'scatter', mode = 'lines', x = ~date,
             y = ~확진자_오세아니아, color = I('#1f77b4')) |>
  layout(title = list(title = NULL),
         xaxis = list(tickfont = list(size = 10)),
         yaxis = list(title = list(text = '확진자수')))

## 일곱 번째 서브 플롯 생성
p7 <- p_line_wide |>
  add_trace(type = 'scatter', mode = 'lines', x = ~date,
             y = ~확진자_아프리카, color = I('#1f77b4')) |>
  layout(title = list(title = NULL),
         xaxis = list(tickfont = list(size = 10)),
         yaxis = list(title = list(text = '확진자수')))

## subplot 설정
subplot(
  p1 |>
    ## Layout 으로 첫 번째 서브 플롯의 주석 표시
    layout(annotations = list(
      ## 주석의 위치와 주석 text 설정
      x = 0.5 , y = 1.02, text = "한국",
      ## 화살표는 표시하지 않음
      showarrow = F,
      ## X, Y 의 참조 위치는 'paper'
      xref='paper', yref='paper',
      ## X 방향 정렬은 'center'
      xanchor = 'center'))),
  p2 |>
    ## Layout 으로 두 번째 서브 플롯의 주석 표시
    layout(annotations = list(
      x = 0.5 , y = 1.02, text = "아시아",
      showarrow = F, xref='paper', yref='paper',
      xanchor = 'center'))),
  p3 |>
    ## Layout 으로 세 번째 서브 플롯의 주석 표시
    layout(annotations = list(
      x = 0.5 , y = 1.02, text = "유럽",
      showarrow = F, xref='paper', yref='paper',
      xanchor = 'center'))),
  p4 |>

```

```
## Layout 으로 네 번째 서브 플롯의 주석 표시
layout(annotations = list(
  x = 0.5 , y = 1.02, text = "북미",
  showarrow = F, xref='paper', yref='paper',
  xanchor = 'center'))),
p5 |>
## Layout 으로 다섯 번째 서브 플롯의 주석 표시
layout(annotations = list(
  x = 0.5 , y = 1.02, text = "남미",
  showarrow = F, xref='paper', yref='paper',
  xanchor = 'center'))),
p6 |>
## Layout 으로 여섯 번째 서브 플롯의 주석 표시
layout(annotations = list(
  x = 0.5 , y = 1.02, text = "오세아니아",
  showarrow = F, xref='paper', yref='paper',
  xanchor = 'center'))),
p7 |>
## Layout 으로 일곱 번째 서브 플롯의 주석 표시
layout(annotations = list(
  x = 0.5 , y = 1.02, text = "아프리카",
  showarrow = F, xref='paper', yref='paper',
  xanchor = 'center'))),
## 서브플롯은 3 개의 열로 설정
nrows = 3,
## 서브플롯간의 여백 설정
margin = 0.04) |>
## 범례는 제거
layout(showlegend = FALSE,
      ## 전체 제목 설정
      title = '최근 100 일간 코로나 19 확진자수',
      ## 전체 여백 설정
      margin = margins)
```



### 실행결과3- 90. 서브 플롯 생성

`subplot()`으로 완성되는 시각화는 `layout()`으로 세부 속성의 설정이 가능하다. 하지만 `subplot()`에서 사용되는 속성값이 몇 개 있는데 그 중 꼭 알아두어야 하는 속성 값이 `shareX`와 `shareY`이다. 이 두 속성은 서브 플롯에 표시되는 각각의 플롯의 X, Y 축을 공유할지를 결정하는 속성이다. 이 속성값이 `shareX` 값이 TRUE로 설정되면 수직으로 같은 열에 표시된 서브플롯들의 X 축은 모두 스케일로 고정된다. 하지만 각각의 열들의 X 축은 모두 각각의 열에 적합한 축으로 표현되기 때문에 열 별로는 X 축의 스케일이 달라질 수 있다. 마찬가지로 `shareY`가 TRUE로 설정되면 전체 플롯의 같은 행에 표현된 서브 플롯은 같은 스케일의 Y 축을 공유한다.

```
subplot(
  p1 |>
    ## Layout 으로 첫 번째 서브 플롯의 주석 표시
    layout(annotations = list(
      x = 0.5 , y = 1.02, text = "한국",
      showarrow = F,
      xref='paper', yref='paper',
      xanchor = 'center'))),
p2 |>
  ## Layout 으로 두 번째 서브 플롯의 주석 표시
```

```

layout(annotations = list(
  x = 0.5 , y = 1.02, text = "아시아",
  showarrow = F, xref='paper', yref='paper',
  xanchor = 'center')),  

p3 |>
## Layout 으로 세 번째 서브 플롯의 주석 표시  

layout(annotations = list(
  x = 0.5 , y = 1.02, text = "유럽",
  showarrow = F, xref='paper', yref='paper',
  xanchor = 'center')),  

p4 |>
## Layout 으로 네 번째 서브 플롯의 주석 표시  

layout(annotations = list(
  x = 0.5 , y = 1.02, text = "북미",
  showarrow = F, xref='paper', yref='paper',
  xanchor = 'center')),  

p5 |>
## Layout 으로 다섯 번째 서브 플롯의 주석 표시  

layout(annotations = list(
  x = 0.5 , y = 1.02, text = "남미",
  showarrow = F, xref='paper', yref='paper',
  xanchor = 'center')),  

p6 |>
## Layout 으로 여섯 번째 서브 플롯의 주석 표시  

layout(annotations = list(
  x = 0.5 , y = 1.02, text = "오세아니아",
  showarrow = F, xref='paper', yref='paper',
  xanchor = 'center')),  

p7 |>
## Layout 으로 일곱 번째 서브 플롯의 주석 표시  

layout(annotations = list(
  x = 0.5 , y = 1.02, text = "아프리카",
  showarrow = F, xref='paper', yref='paper',
  xanchor = 'center')),  

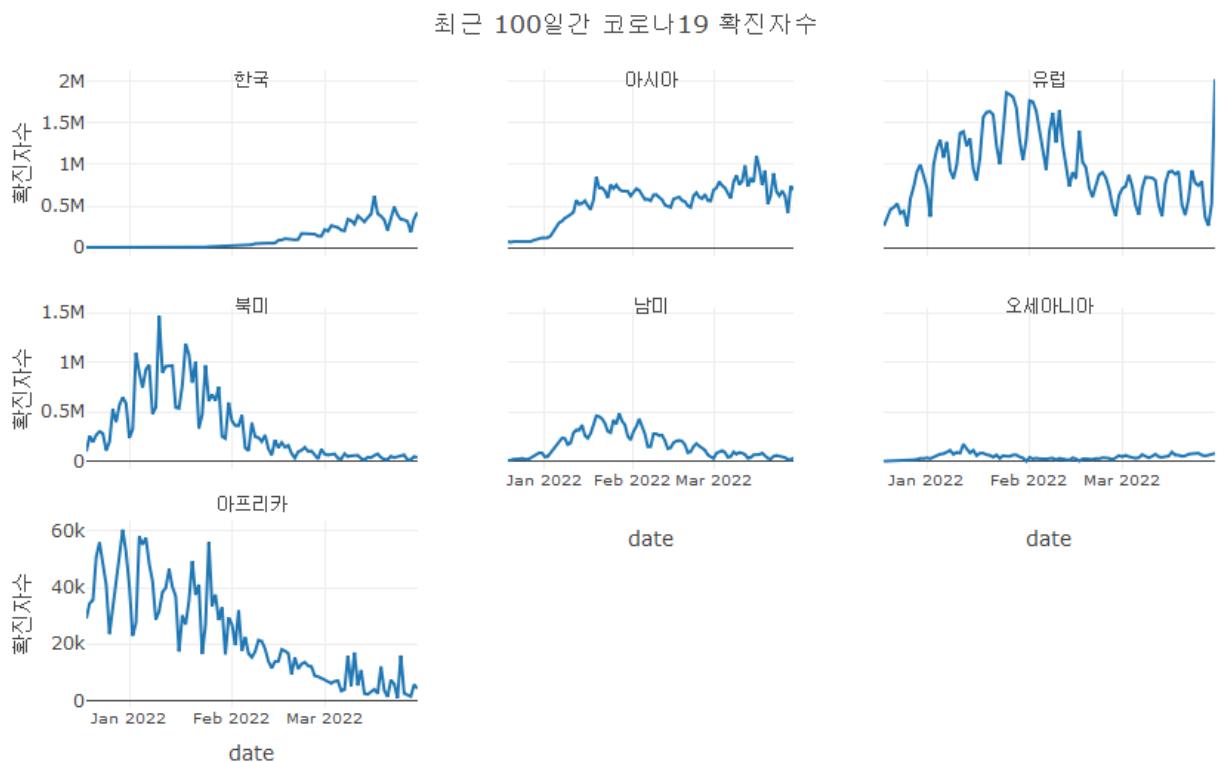
nrows = 3, margin = 0.04,  

## shareX, shareY 를 TRUE 로 설정하여 축을 공유  

shareX = TRUE, shareY = TRUE) |>  

layout(showlegend = FALSE,
      title = '최근 100 일간 코로나 19 확진자수',
      margin = margins)

```



### 실행결과 3- 91. shareX, shareY 가 설정된 subplot

만약 모든 서브 플롯의 X, Y 축의 스케일을 동일하게 표현하기 위해서는 각각의 서브 플롯의 X, Y 축의 **range** 속성을 설정함으로써 가능하다.

```
## 서브 플롯들의 Y 축 범위 설정
p1 <- p1 |> layout(yaxis = list(range = c(0, 2000000)))
p2 <- p2 |> layout(yaxis = list(range = c(0, 2000000)))
p3 <- p3 |> layout(yaxis = list(range = c(0, 2000000)))
p4 <- p4 |> layout(yaxis = list(range = c(0, 2000000)))
p5 <- p5 |> layout(yaxis = list(range = c(0, 2000000)))
p6 <- p6 |> layout(yaxis = list(range = c(0, 2000000)))
p7 <- p7 |> layout(yaxis = list(range = c(0, 2000000)))

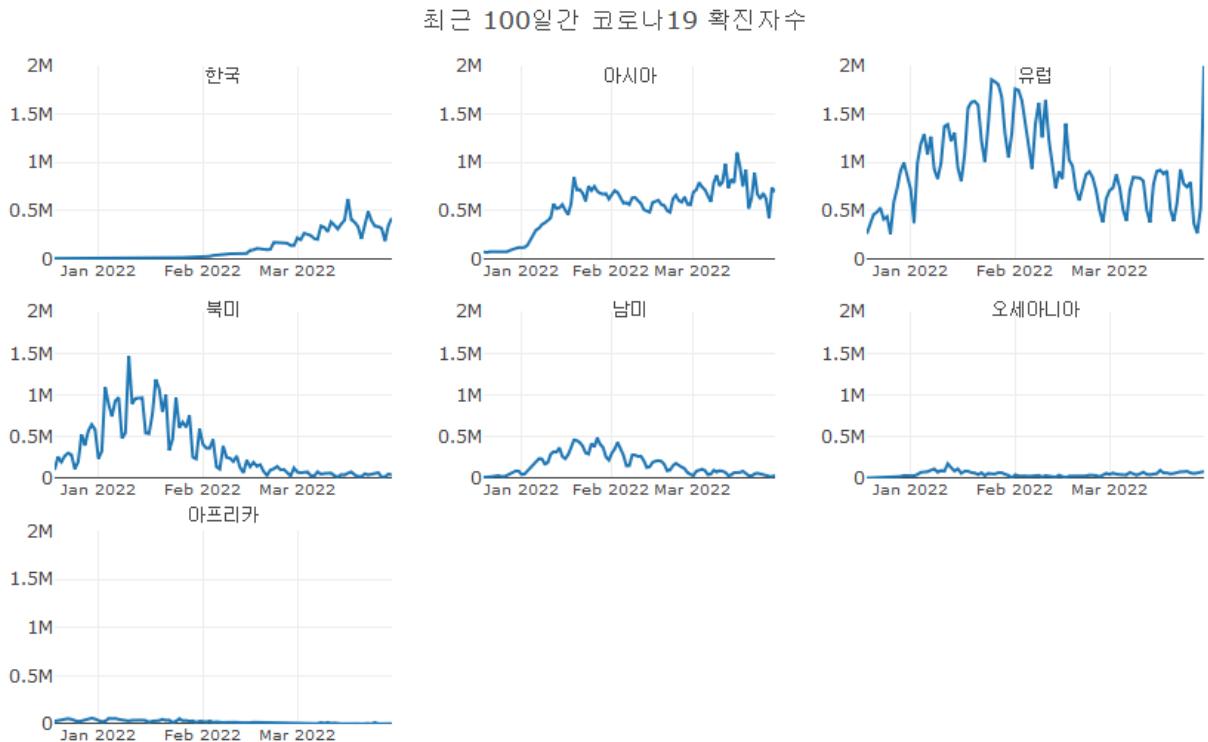
subplot(
  p1 |>
    ## Layout 으로 첫 번째 서브 플롯의 주석 표시
    layout(annotations = list(
```

```

x = 0.5 , y = 1.02, text = "한국",
showarrow = F,
xref='paper', yref='paper',
xanchor = 'center')),
p2 |>
## Layout 으로 두 번째 서브 플롯의 주석 표시
layout(annotations = list(
  x = 0.5 , y = 1.02, text = "아시아",
  showarrow = F, xref='paper', yref='paper',
  xanchor = 'center')),
p3 |>
## Layout 으로 세 번째 서브 플롯의 주석 표시
layout(annotations = list(
  x = 0.5 , y = 1.02, text = "유럽",
  showarrow = F, xref='paper', yref='paper',
  xanchor = 'center')),
p4 |>
## Layout 으로 네 번째 서브 플롯의 주석 표시
layout(annotations = list(
  x = 0.5 , y = 1.02, text = "북미",
  showarrow = F, xref='paper', yref='paper',
  xanchor = 'center')),
p5 |>
## Layout 으로 다섯 번째 서브 플롯의 주석 표시
layout(annotations = list(
  x = 0.5 , y = 1.02, text = "남미",
  showarrow = F, xref='paper', yref='paper',
  xanchor = 'center')),
p6 |>
## Layout 으로 여섯 번째 서브 플롯의 주석 표시
layout(annotations = list(
  x = 0.5 , y = 1.02, text = "오세아니아",
  showarrow = F, xref='paper', yref='paper',
  xanchor = 'center')),
p7 |>
## Layout 으로 일곱 번째 서브 플롯의 주석 표시
layout(annotations = list(
  x = 0.5 , y = 1.02, text = "아프리카",
  showarrow = F, xref='paper', yref='paper',
  xanchor = 'center')),
nrows = 3,
shareX = FALSE, shareY = FALSE, margin = 0.04) |>
layout(showlegend = FALSE,

```

```
title = '최근 100 일간 코로나 19 확진자수',
margin = margins)
```



### 실행결과3- 92. 축 범위가 설정된 subplot

앞의 `subplot()`에서의 예는 각각의 `plotly` 객체를 각각 만든 후에 이들 `plotly` 객체들을 `subplot()`으로 하나의 플롯으로 만들어 주었다. 서브 플롯이 몇개 되지 않을 때는 이렇게 각각 만든 `plotly` 객체를 묶어주는 것이 가능하겠지만 서브 플롯이 많을 때는 이러한 작업이 매우 어려워진다. 특히 긴 형태의 데이터프레임의 경우 앞서와 같이 서브 플롯을 만들어 주기 위해서는 각각의 서브 플롯마다 데이터를 필터링하여 만들어야 하기 때문에 매우 번거로운 작업이 수반된다. 이럴 경우는 `group_by()`로 데이터프레임을 그룹화하고 `.do()`를 사용하여 각각의 그룹화된 데이터 그룹별로 `plotly` 객체를 만드는 방식으로 간단히 서브 플롯을 만들어 줄 수 있다. 이 때 `do`를 사용하여 각각의 그룹화된 데이터 그룹별 `plotly` 객체를 만드는 코드를 적용하여야 한다.

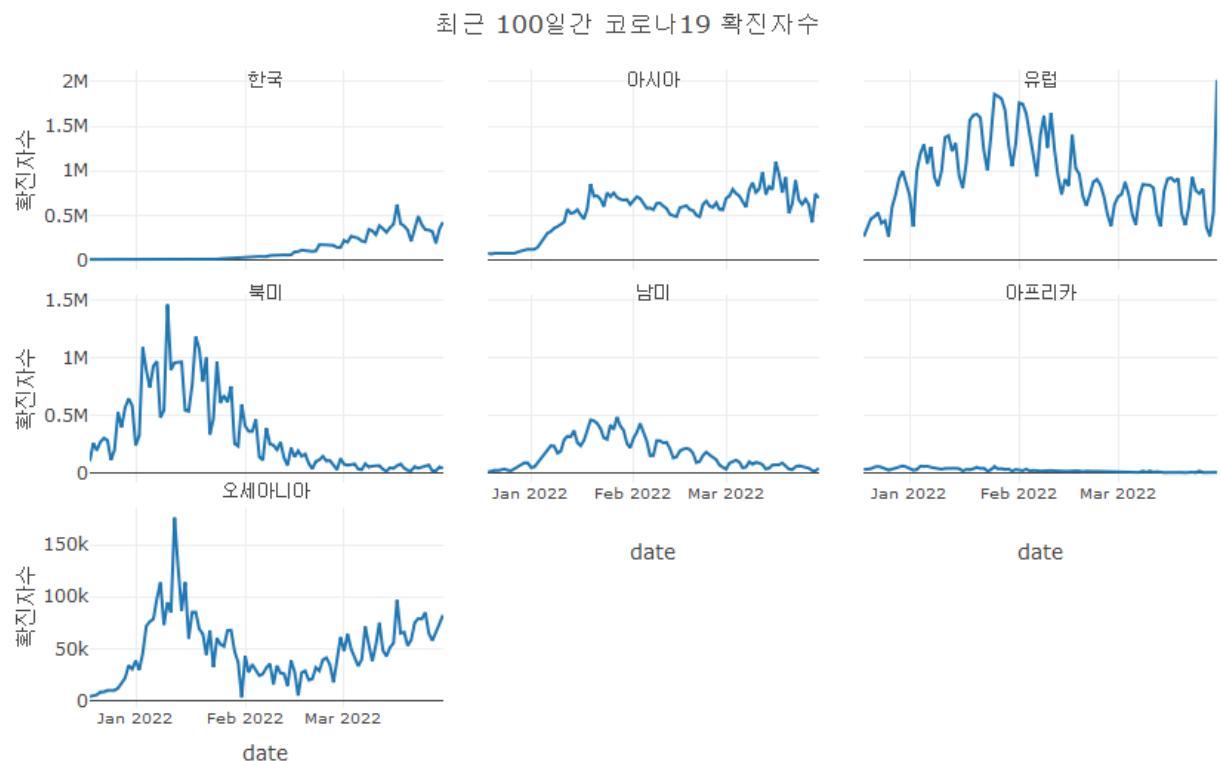
```
## 긴 형태의 100 일 코로나19 데이터에서
covid19_df_100 |>
## 국가명으로 그룹화
group_by(location) |>
## 그룹화한 각각의 데이터 그룹들에 적용할 코드 설정
do(
```

```

## 각 그룹화한 데이터를 사용해 plotly 객체 생성
p = plot_ly(.) |>
    ## Line 모드의 스캐터 trace 추가
    add_trace(type = 'scatter', mode = 'lines',
              ## X, Y 축에 변수 맵핑, color 를 설정
              x = ~date, y = ~new_cases, color = I('#1f77b4')) |>
    ## Layout 으로 X, Y 축을 설정
    layout(title = list(title = NULL),
           xaxis = list(tickfont = list(size = 10)),
           yaxis = list(title = list(text = '확진자수')),
           ## 국가명을 주석으로 표시
           annotations = list(x = 0.5, y = 1.02, text = ~location,
                               showarrow = F, xref='paper',
                               yref='paper', xanchor = 'center'))
) |>

## 생성된 plotly 객체들을 subplot 생성
subplot(nrows = 3, shareX = TRUE, shareY = TRUE) |>
## 생성된 subplot 의 layout 설정
layout(showlegend = FALSE,
       title = '최근 100일간 코로나19 확진자수',
       margin = margins)

```



실행결과3- 93. group\_by()를 사용한 subplot 생성