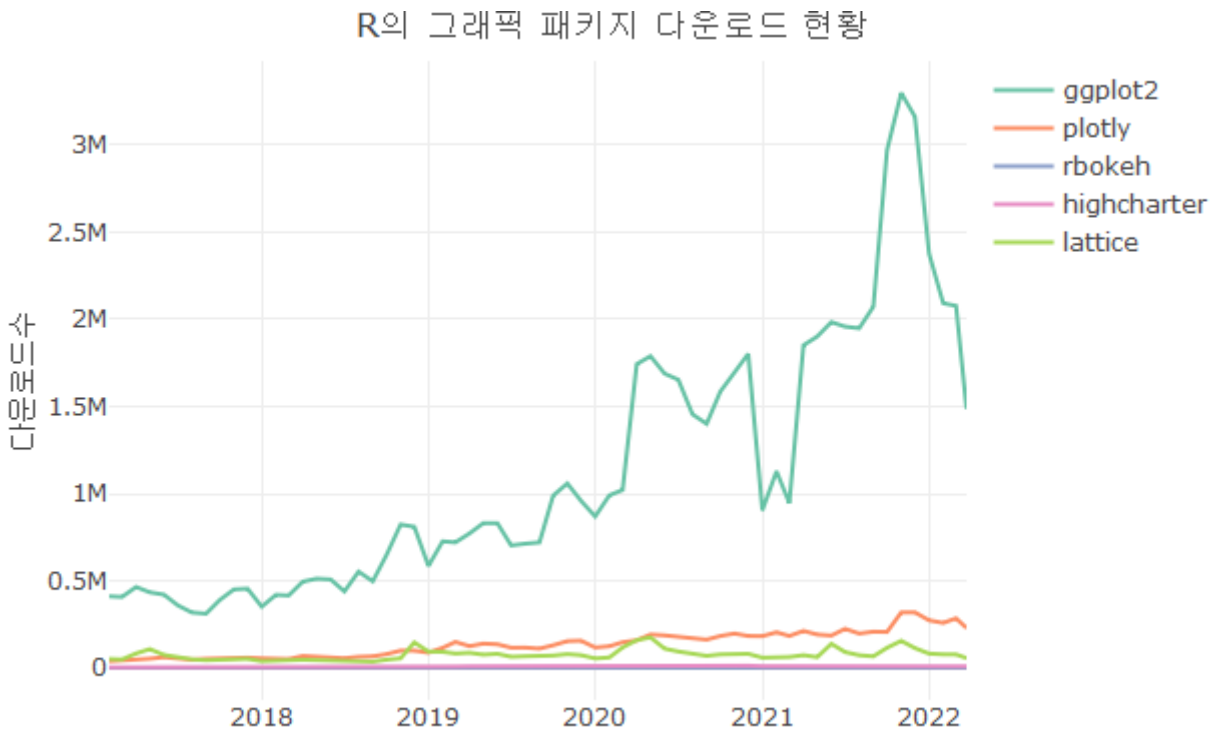
III. 데이터 시각화를 위한 plotly

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 2021 년 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(visi
ble = F)), yaxis = list(title = '다운로드수'))
```



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.[1] 이를 `plotly` 실습에 사용하기 위해 다음과 같이 전처리 하였다.

```
if(!require(readr)) {
  install.packages('readr')
```

[1] https://github.com/owid/covid-19-data/blob/master/public/data/owid-covid-data.csv

```
  library(readr)
}

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

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

covid19_df_100 <- covid19_df |>
  filter((iso_code %in% c('KOR', 'OWID_ASI', 'OWID_EUR', 'OWID_OCE', 'OWID_NAM',
 'OWID_SAM'))) |>
  filter(date >= today() - 100) |>
  arrange(date)

covid19_df_100_wide <- covid19_df_100 |> select(date, location, new_cases) |>
  pivot_wider(id_cols = date, names_from = location, values_from = new_cases) |>

  arrange(date)

names(covid19_df_100_wide) <- c('date', 'Asia', 'Europe', 'North_America', 'Ocea
nia', 'South_America', 'South_Korea')

covid19_stat <- covid19_df |> group_by(iso_code, continent, location) |>
  summarise(인구수 = max(population, na.rm = T), 인당GDP = max(gdp_per_capita, n
a.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()
```

1. plotly 란?

plotly 는 오픈 소스인 JavaScript 로 구현된 plotly.js 를 기반으로 R 에서 생성한 데이터 시각화 객체를 Javascript 로 생성해주는 패키지이다[2]. 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.

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

2. 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.

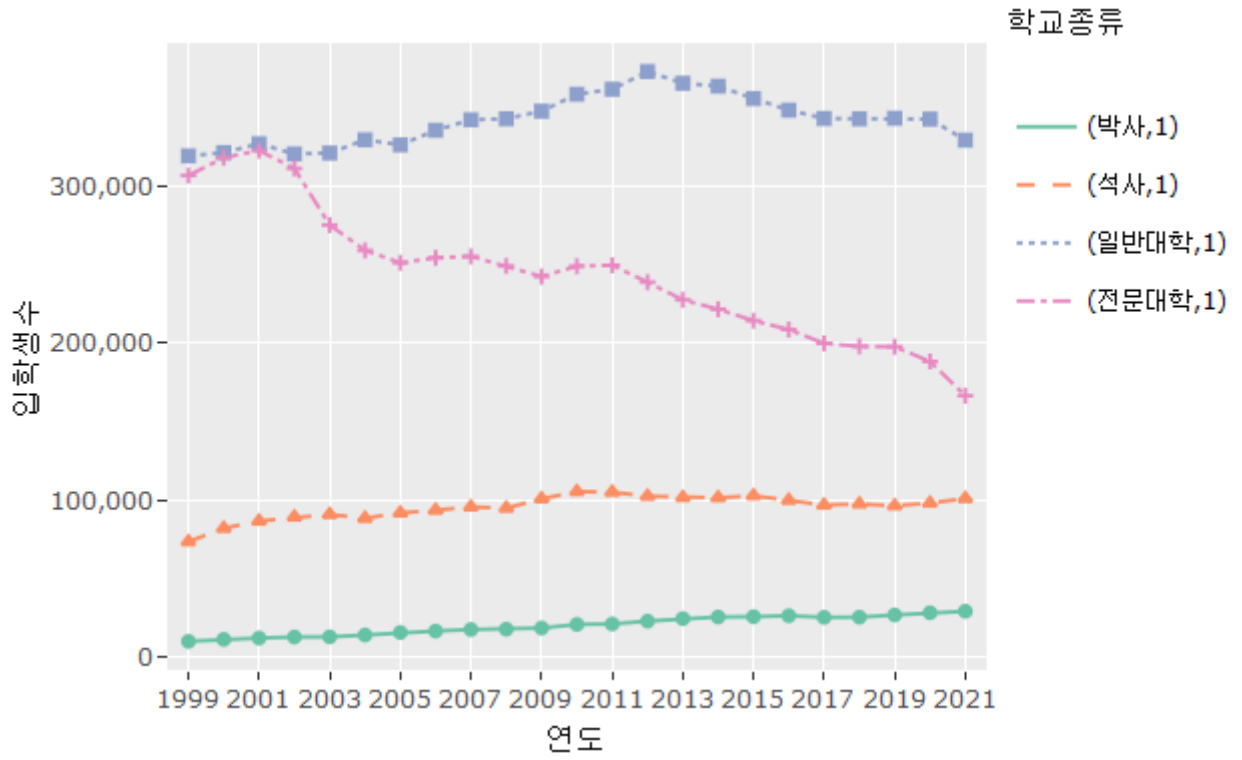
```
ggplotly(p = ggplot2::last_plot(), width = NULL, height = NULL, tooltip = "all",
dynamicTicks = FALSE, layerData = 1, originalData = TRUE, source = "A", …)
```

[2] https://plotly.com

- 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.

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

ggplotly <-df_입학자_long |>
  ## 지역이 전체, 학교종류가 '전문대학', '일반대학', '석사', '박사'인 데이터 필터링
  filter(지역 == '전체', 학교종류 %in% c('전문대학', '일반대학', '석사', '박사')) |>
  ## x 축이 연도로 매핑된 ggplot 객체 생성
  ggplot(aes(x = 연도, y = 입학생수, color = 학교종류 )) +
  ## y 축이 입학생수, group, color, linetype 이 학교종류로 매핑된 geom_line 레이어 생성
  geom_line(aes(group = 학교종류, linetype = 학교종류)) +
  ## y 축이 입학생수, group, color 이 학교종류로 매핑된 geom_point 레이어 생성
  geom_point(aes(shape = 학교종류), show.legend = FALSE) +
  ## x 축의 눈금을 1999 부터 2021 까지 2 씩 증가한 수치로 설정
  scale_x_discrete(breaks = c(seq(from = 1999, to = 2021, by = 2))) +
  ## y 축에 표현된 라벨을 scales 패키지의 comma 함수를 적용(scales 패키지 설치 필요)
  scale_y_continuous(labels = scales::comma) +
  ## 색 설정을 RColorBrewer 패키지의 'Accent'로 설정
  scale_color_brewer(palette = 'Set2', labels = c('박사', '석사', '일반대학', '전문대학'))

ggplotly(ggplotly)
```



앞의 `plotly` 객체에서 보면 `ggplot` 객체를 정확히 전환하지는 못한다.[3] 그러나 `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 수있게 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.

[3] 앞의 예에서 범례에 나타난 오류는 ggplotly()에서 다중 범례(Multiple Legend)의 변환 과정에서 발생하는 문제로 ggplot 의 범례를 제거하고 plotly 의 범례를 생성함으로써 해결가능하다.

3. plotly 시각화 생성

`plotly` 객체는 pltoly.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 위한 코드는 `plot_ly()`를 사용한 `plotly` 객체 생성, `add_trace()`를 사용한 trace 추가, `layout()`을 사용한 레이아웃 설정의 세 부분으로 구성된다.

3.1. plotly 객체 생성 : plot_ly()

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

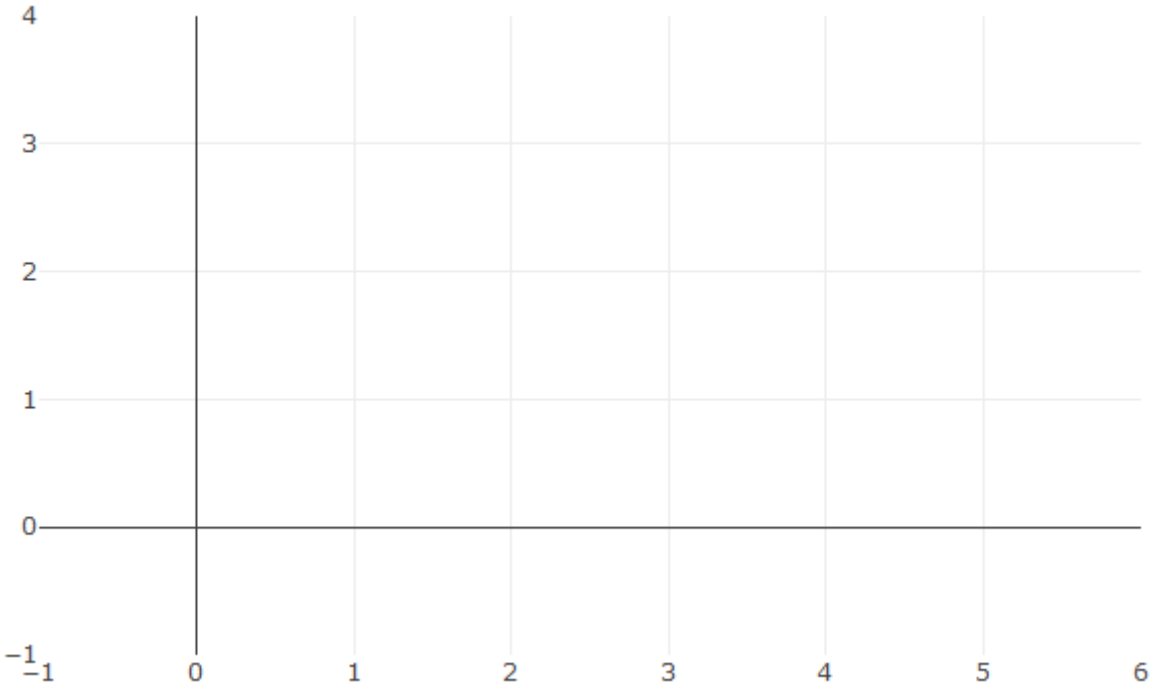
`plot_ly()`는 plotly.js 에서 정의된 `plotly` 객체 스키마를 생성하는 함수이다. 사용자가 직접 plotly.js 형태의 스키마 객체를 타이핑하여 생성하는 것은 어려움이 따르기 때문에 `plotly` 객체 스키마 생성, 기본 스키마 속성 설정 등을 지원하는 함수이다. 이 방법은 `ggplot` 객체를 생성하기 위해 `ggplot()`를 사용하여 `ggplot` 객체를 생성하는 것과 동일한 방법이고 이 방법에서 영감을 받았다고 한다.

```
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
- … : type 에서 설정하는 trace 의 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 : color 에서 설정된 색의 투명도 값(value) 설정
```

```
- stroke : ‘stroke-color’(외곽선 색) 속성으로 매핑될 색 값(value) 설정
- strokes : ‘stroke-color’(외곽선 색)에 매핑될 colorbrewer2.org의 팔레트 이름이나
16진수의 ‘#RRGGBB’형태로 표현된 색 벡터(vector) 설정/ - alpha-stroke :
stroke(외곽선)에 적용될 alpha 값(value) 설정
- size : ’fill-size’에 매핑될 크기값(value) 설정
- sizes : size에 매핑될 수치 벡터(vector) 설정
- span : ’stroke-size’(외곽선 두께)에 매핑될 두께 값(value) 설정
- spans : ‘stroke-size’(외곽선 두께)에 매핑될 두께 벡터(vector) 설정
- symbol : 점 표현에 사용되는 도형 번호(pch)나 도형 이름 값(value) 설정
- symbols : 점 표현에 사용되는 도형 번호(pch)나 도형 이름 벡터(vector) 설정
- linetype : 라인 타입의 설정에 사용되는 번호나 라인 타입 값(value) 설정
- linetypes : 라인 타입의 설정에 사용되는 번호나 라인 타입 벡터(vector) 설정
- split : 다중 traces를 생성시 사용하는 값 설정
- frame : 애니메이션 프레임 생성시 사용할는 값 설정
- width : 플롯의 너비(픽셀) 설정
- height : 플롯의 높이(픽셀) 설정
```

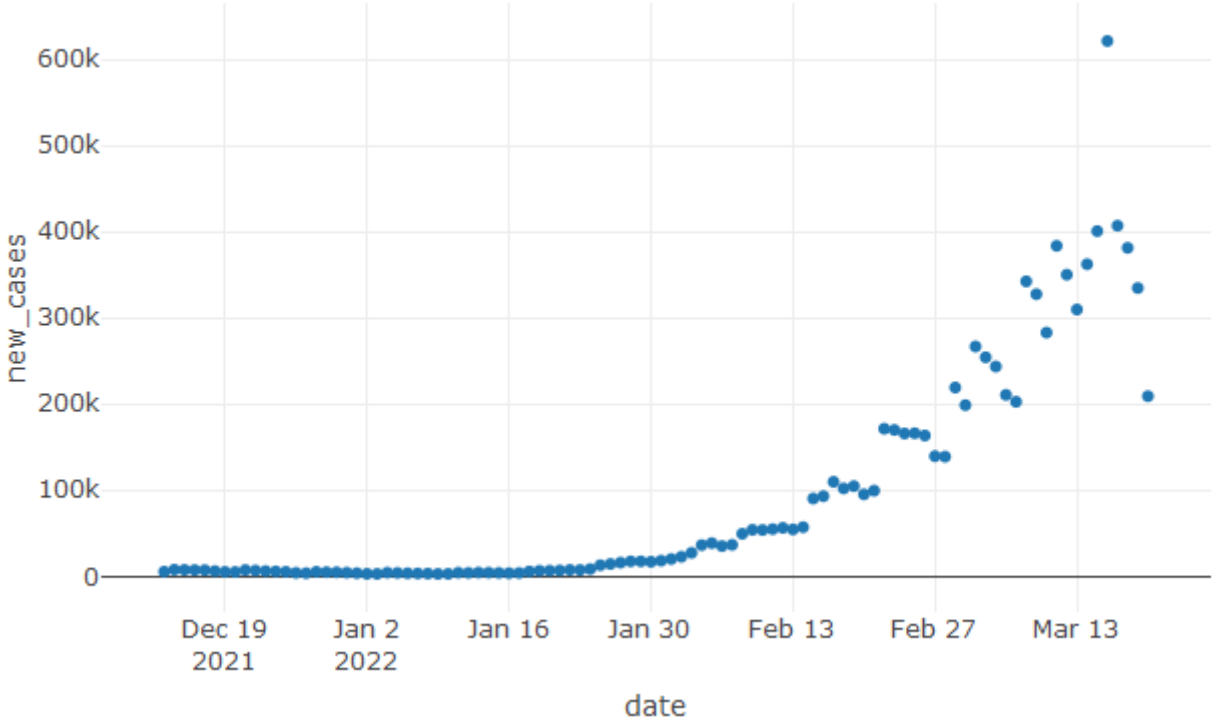
이처럼 `plot_ly()`의 함수에서는 많은 매개변수가 사용된다. 특히 '...'으로 표기된 부분은 `type` 매개변수에서 설정하는 trace 타입에 따라 설정 내용이 매우 달라진다. `ggplot2` 에서 `geom_*()`를 사용하여 기하 요소 레이어를 하나 하나 설정하면서 전체 시각화를 완성하는 것과 유사하게 `plotly` 에서는 여러개의 trace 를 추가함으로써 전체 시각화를 완성해나갈수 있다. 다만 `ggplot2` 의 `ggplot()`에서는 데이터 요소와 미적요소만을 설정할 수 있었지만 `plotly` 에서는 데이터, trace, trace 에 따른 속성까지 설정이 가능하므로 `plot_ly()`만가지고도 시각화를 완성할 수 있다. 반면 동시에 여러개의 trace 가 포함되는 시각화에서는 뒤에서 설명할 `add_trace()`나 `add_*()`를 사용하여 trace 를 추가하게 되는데 `plot_ly()`에서 설정한 속성들을 상속받게 된다. 이 과정에서 원치않는 속성의 상속을 방지하기 위해 `plot_ly()`에 매개변수를 넣지않고 단순히 `plotly` 객체의 초기화 명령으로 사용도 가능하다.

```
covid19_df_100 |> plot_ly()
```



`plot_ly()`의 매개변수를 설정할 때 꼭 알아 두어야 하는 것은 변수를 매핑하는 방법과 값을 설정하는 방법이 다르다는 것이다. `ggplot2`에서는 변수를 매핑하기 위해서 `aes()`를 사용함으로써 변수를 매핑하였고 값을 실정하기 위해서는 `aes()` 밖에 선언함으로써 설정이 가능하였다. `plotly`에서는 `aes()` 대신 `~`를 사용하여 변수를 매핑한다.

```
covid19_df_100 |> filter(iso_code == 'KOR') |>
  plot_ly(x = ~date, y = ~new_cases)
```



앞선 `plot_ly()`코드에서는 X 축과 Y 축을 지정하는 매개변수인 `x`, `y` 에 `~`를 사용하여 각각의 변수를 매핑하였고 색을 설정하는 매개변수인 `color` 에는 `I()`를 사용하여 'red'를 설정하였다. 사실 `plotly` 객체에서 가장 중요한 것은 trace 의 종류이다. `plot_ly()`만을 사용하여 그래프를 완성할 때 trace 의 종류를 생략하면 `plotly` 에서 데이터를 파악하여 가장 좋은 trace 를 설정해준다.

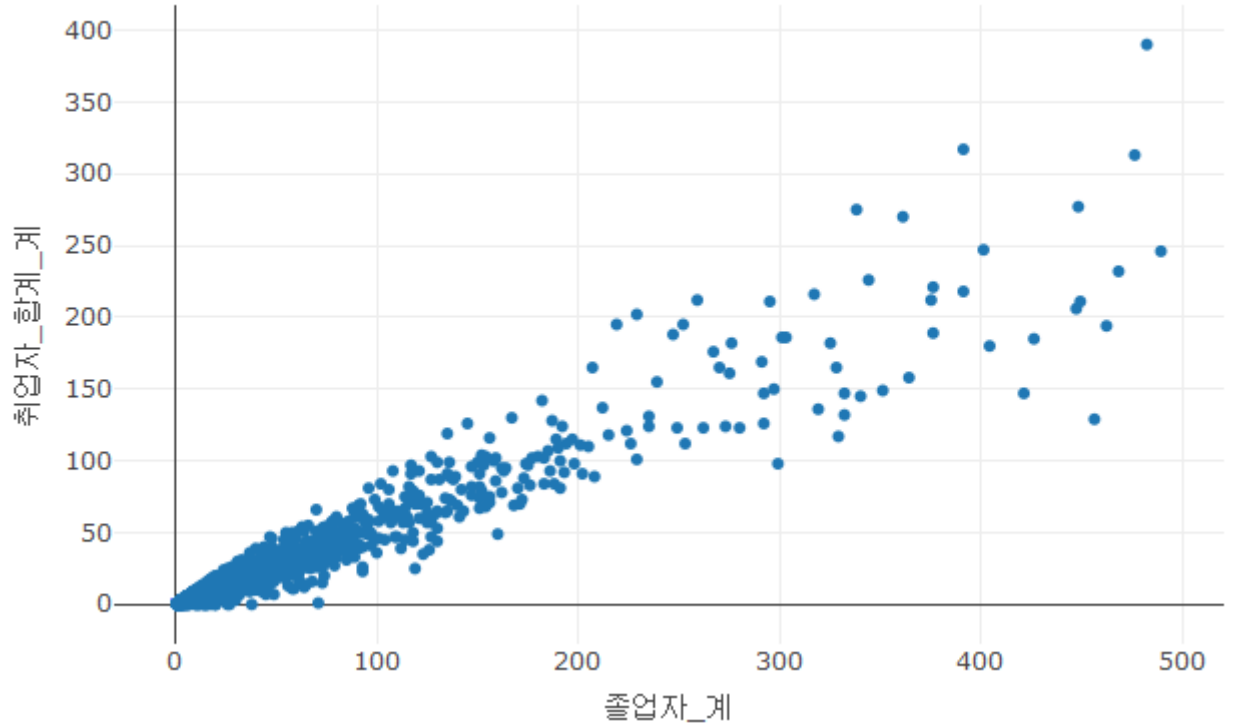
앞의 `plot_ly()`에서 매개변수로 매핑하거나 설정되는 사용되는 type 은 `plot_ly()` 선언 단계에서 사용도 가능하고 다음번 단계인 trace 추가 단계에서도 사용 가능하다. 이 두 방법의 차이는 `plot_ly` 에서 사용된 type 은 추가되는 trace 에서 기본적으로 상속받아 사용되지만 trace 추가 단계에서 사용되는 type 은 해당 trace 에서만 사용되는 type 이 된다는 점이 다르다.

`plot_ly` 에서 사용이 가능한 대표적인 type 은 다음과 같다.

3.1.1. x, y, z

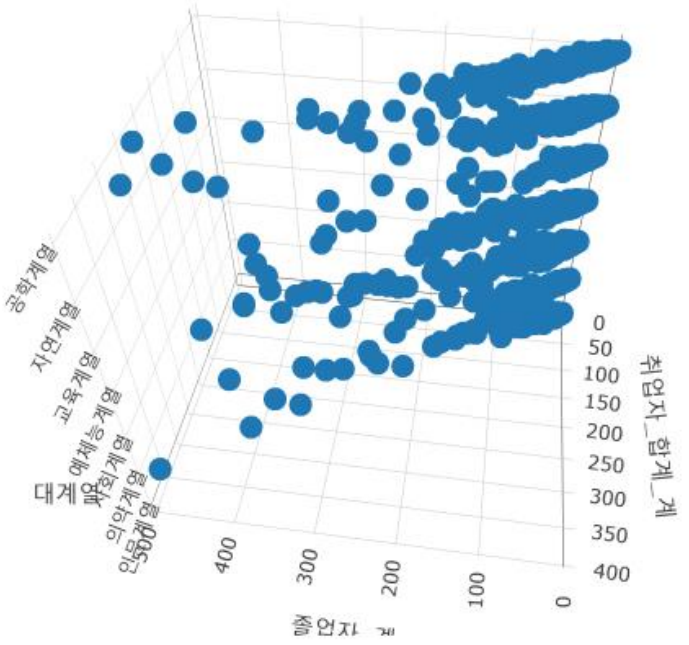
`add_trace()`에서 가장 기본적으로 설정하는 type 은 `x`, `y`, `z` 이다. 이 세 개의 type 은 X 축, Y 축, Z 축에 매핑하는 변수를 설정한다. 앞서 설명한 바와 같이 `x`, `y`, `z` 에 매핑하는 변수는 `~`를 사용하여 매핑하는데 변수를 매핑하지 않고 벡터를 설정하는 것도 가능하다.

```
set.seed(123)
df_취업통계_2000 <- df_취업통계 |> filter(졸업자_계 < 500) |> sample_n(2000)

df_취업통계_2000 |> plot_ly(x = ~졸업자_계, y = ~취업자_합계_계)
```



`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 차원 을 사용하는 것은 크게 효과적이지 않다고 알려져 있기 때문에 3 차원의 활용은 주의할 필요가 있다.

```
df_취업통계_2000 |>
  plot_ly(x = ~졸업자_계, y = ~취업자_합계_계, z = ~대계열)
```

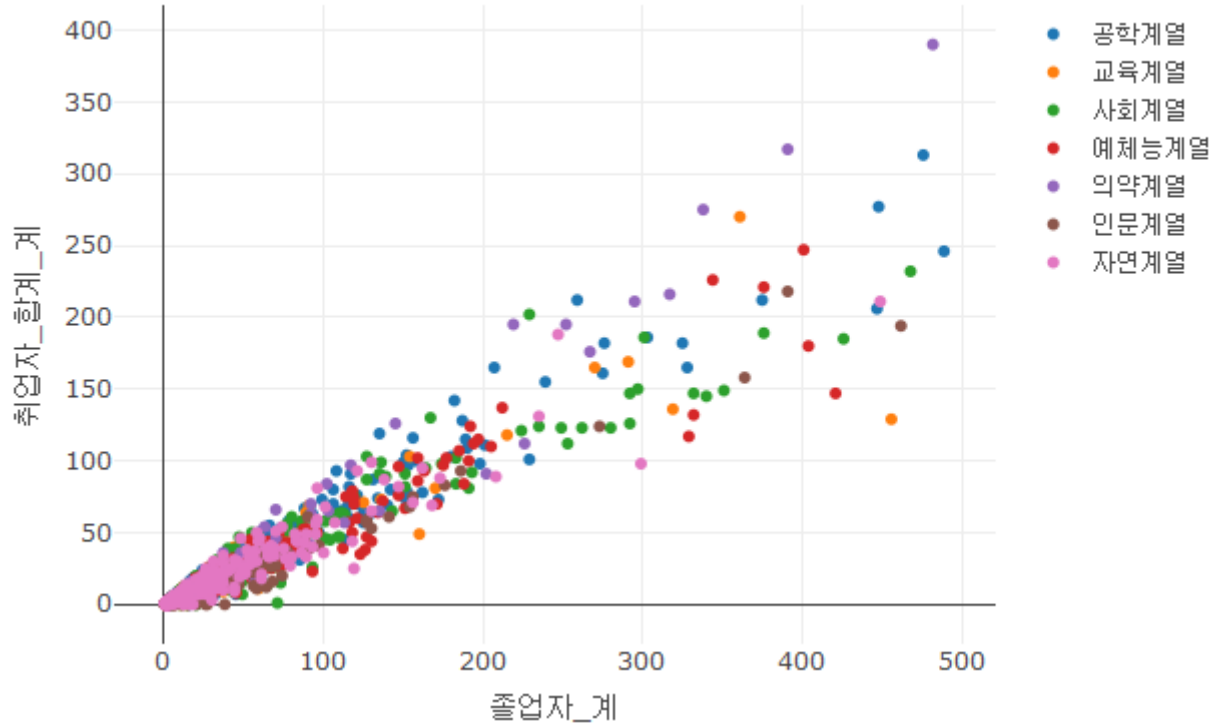


3.1.2. name

`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.

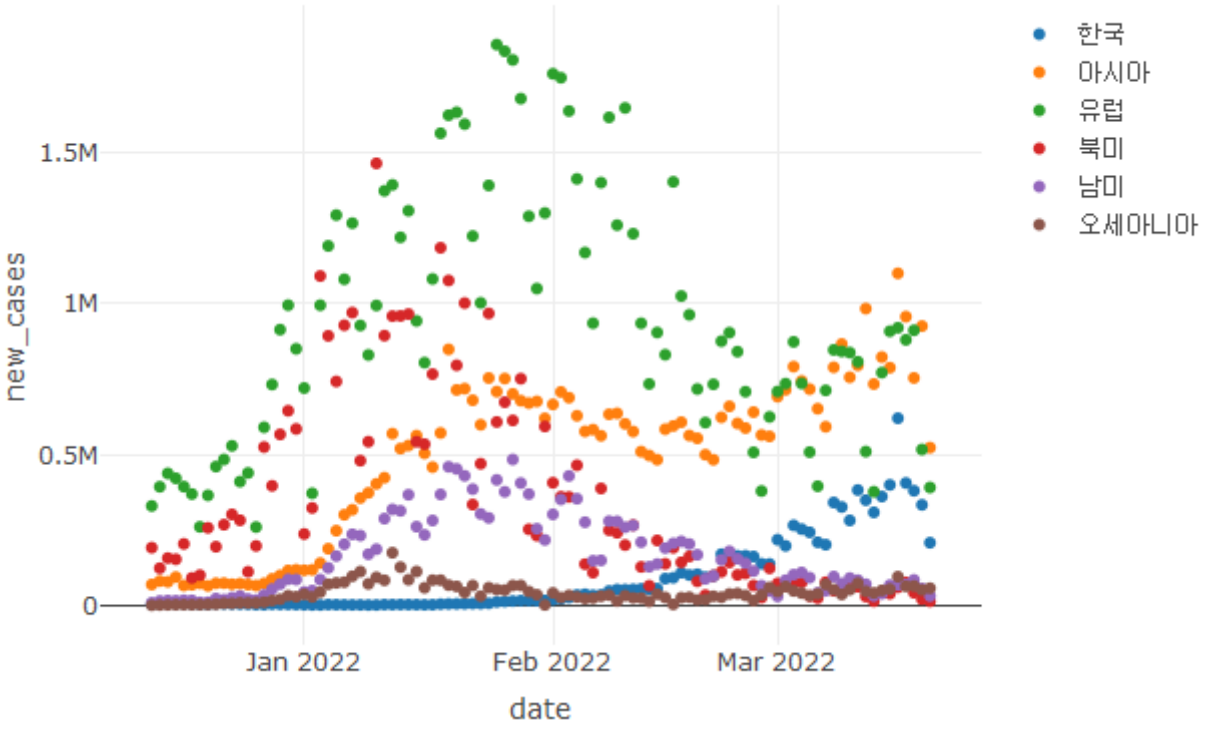
```
df_취업통계_2000 |>
  plot_ly(x = ~졸업자_계, y = ~취업자_합계_계, name = ~대계열)
```



만약 범례를 변수 열로 설정된 값을 사용하지 않고 바꾸려면 범례 아이템을 바꾸는 것이 아니고 각각의 데이터에 매핑될 벡터를 만들어 주어야 한다. `ggplot2` 와 같은 정적 시각화는 시각화가 만들어지면 변경이 되지 않기 때문에 데이터를 시각화하는 부분과는 별도로 범례만을 따로 설정할 수 있지만 동적 시각화는 마우스 포인터를 데이터에 가져가면 해당 데이터에 대한 내용이 표기되어야 하는데 이 값이 범례값과 달라지면 안되기 때문에 범례만 수정할 수 없고 모든 데이터에 1:1 로 매핑되는 범례 아이템 이름 벡터가 필요하다.

```
legend_items <- covid19_df_100 |>
  mutate(legend_name = case_when(
    iso_code == 'KOR' ~ '한국',
    iso_code == 'OWID_ASI' ~ '아시아',
    iso_code == 'OWID_EUR' ~ '유럽',
    iso_code == 'OWID_NAM' ~ '북미',
    iso_code == 'OWID_OCE' ~ '오세아니아',
    iso_code == 'OWID_SAM' ~ '남미')) |>
  select(legend_name) |>
  pull()

legend_items <- fct_relevel(legend_items, '한국', '아시아', '유럽', '북미', '남미
', '오세아니아')
```

```
covid19_df_100 |>
  plot_ly(x = ~date, y = ~new_cases, name = ~legend_items)
```

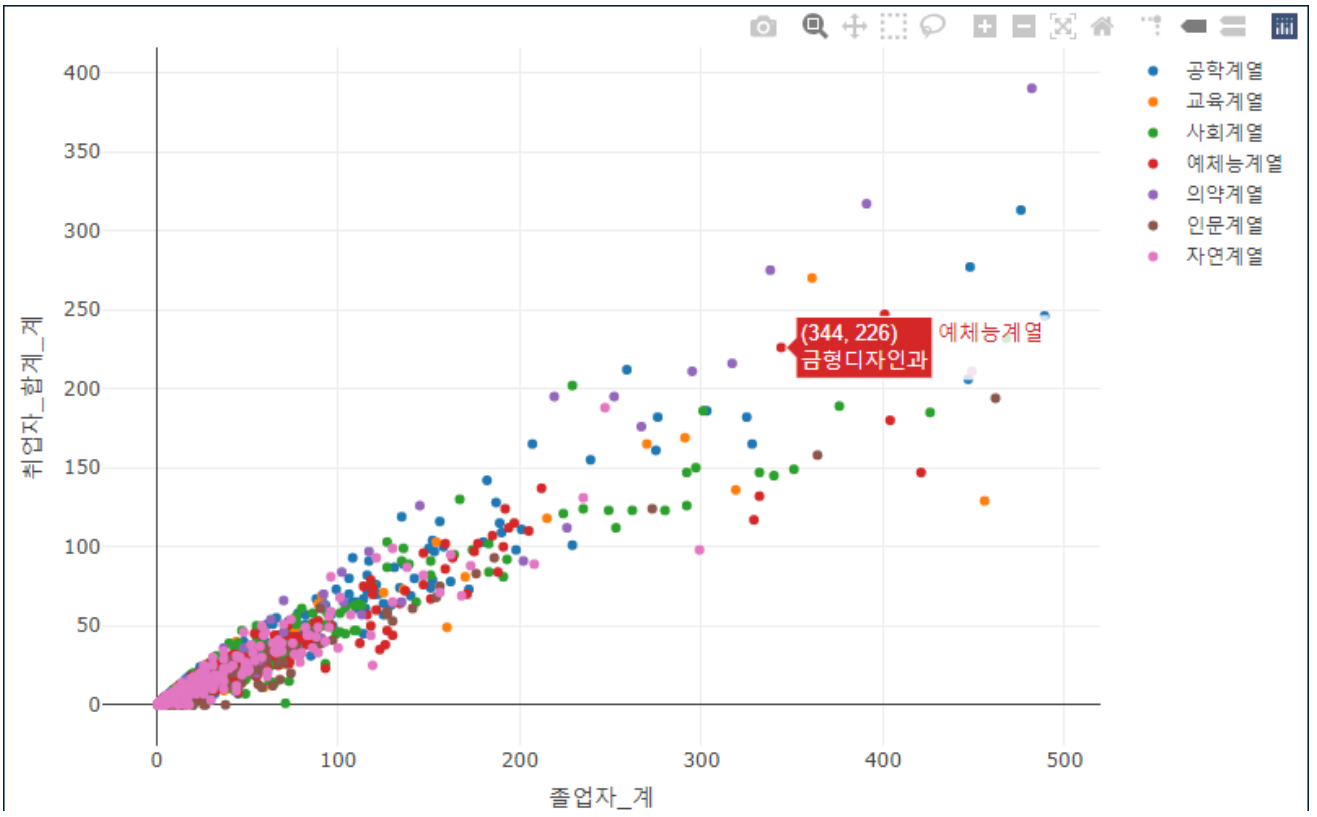


3.1.3. hovertext, hoverinfo, hovermode, hovertemplete

`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 type 이 `hover*`이다.

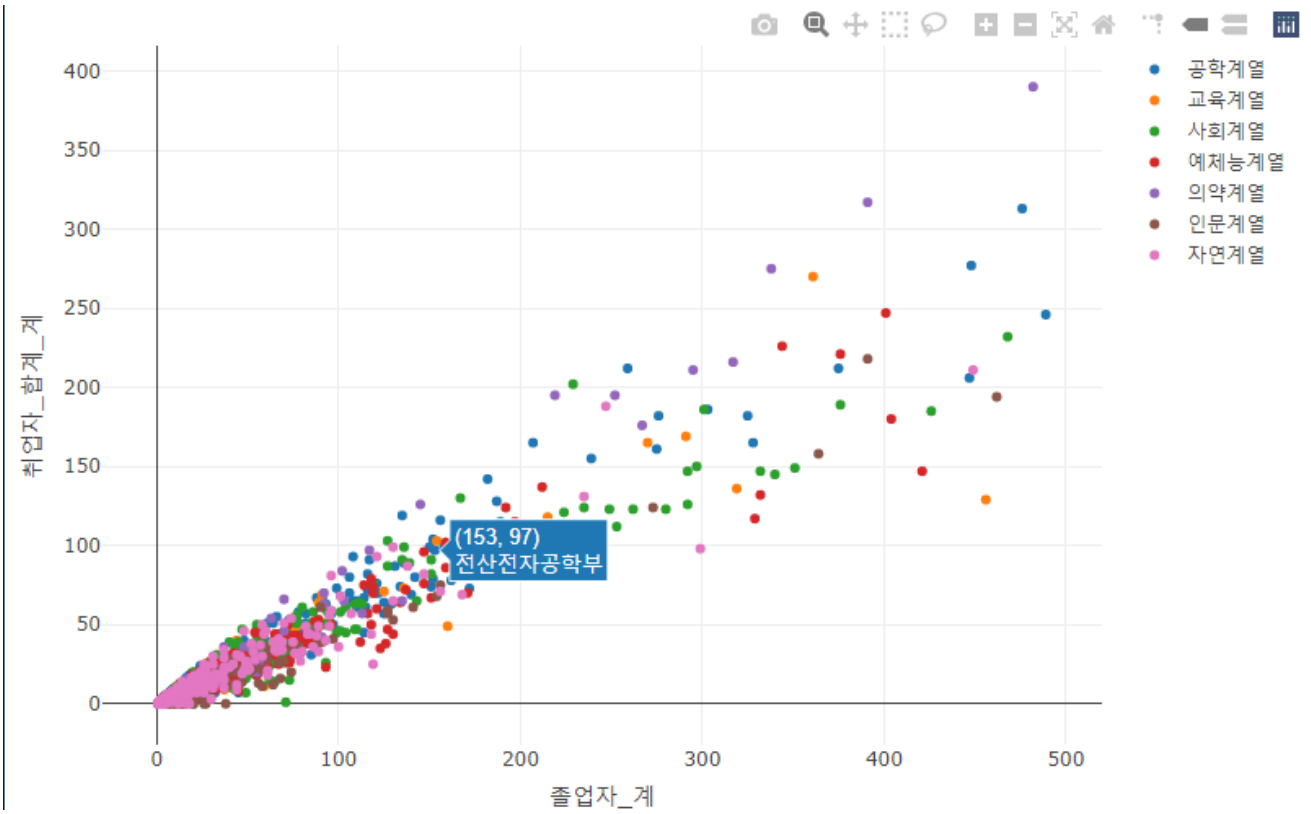
`hovertext` 는 X, Y 좌표에 표시되는 문자열을 설정하는 type 이다. 단순 문자열을 설정하면 모든 데이터 호버에 동일한 문자열이 표기되지만 벡터를 설정하면 각각의 호버에 매핑된 결과가 표시된다.

```
df_취업통계_2000 |>
  plot_ly(x = ~졸업자_계, y = ~취업자_합계_계, name = ~대계열, hovertext = ~학과명)
```



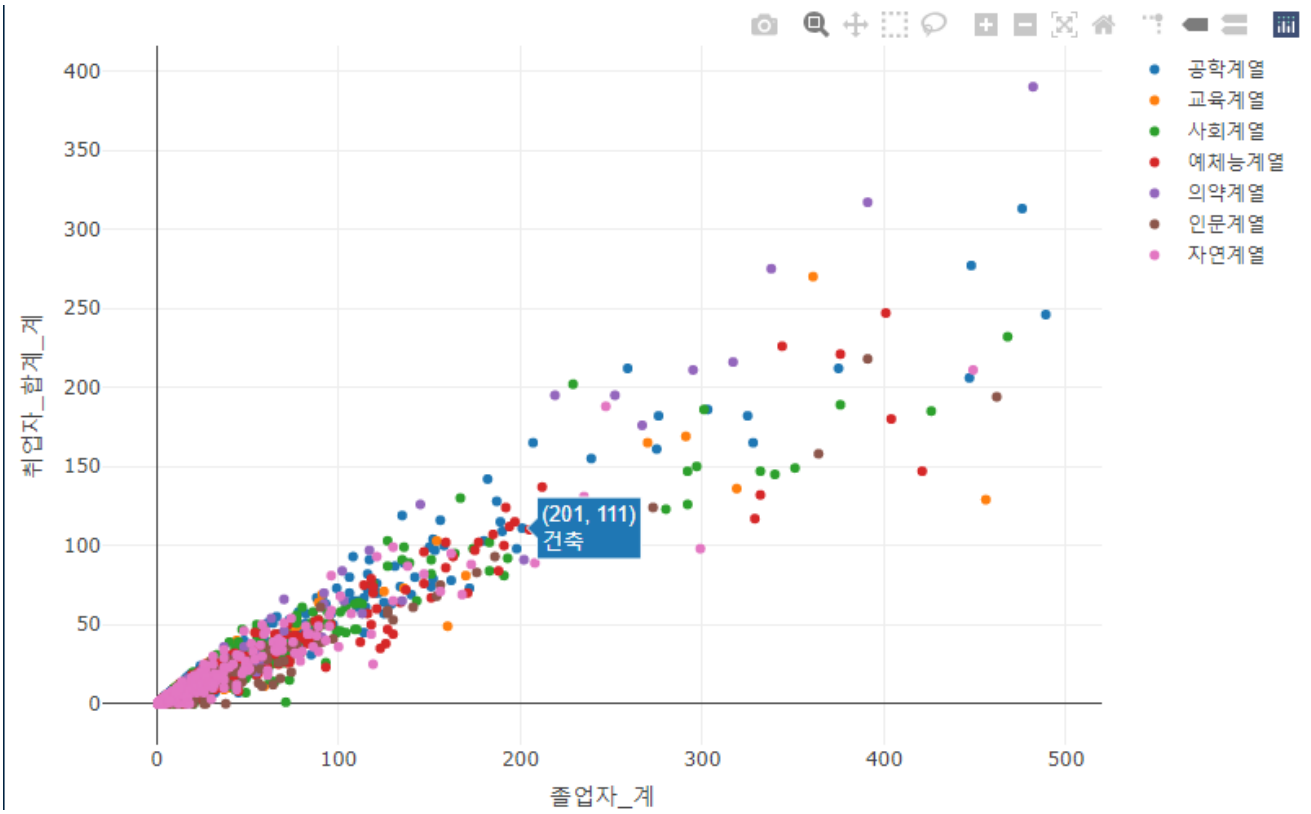
`hoverinfo` 는 호버에 표시되는 데이터 정보를 설정하는 type 이다. 호버에 표시되는 데이터 정보는 각각의 trace 에 따라 다르지만 스캐터 trace 에서는 기본적으로 X, Y, Z 축의 좌표를 표기하는데 기본값은 'all'이 설정된다. 이 외에도 `x`(X 축 좌표), `y`(Y 축 좌표), `z`(Z 축 좌표), `text`(특정 문자열), `name`(trace name), `none`(제거), `skip`(생략)이 사용될 수 있고 각각은 `+`를 사용하여 조합할 수 있다.

```
df_취업통계_2000 |>
  plot_ly(x = ~졸업자_계, y = ~취업자_합계_계, name = ~대계열, hovertext = ~학과명,
 hoverinfo = 'x+y+text')
```



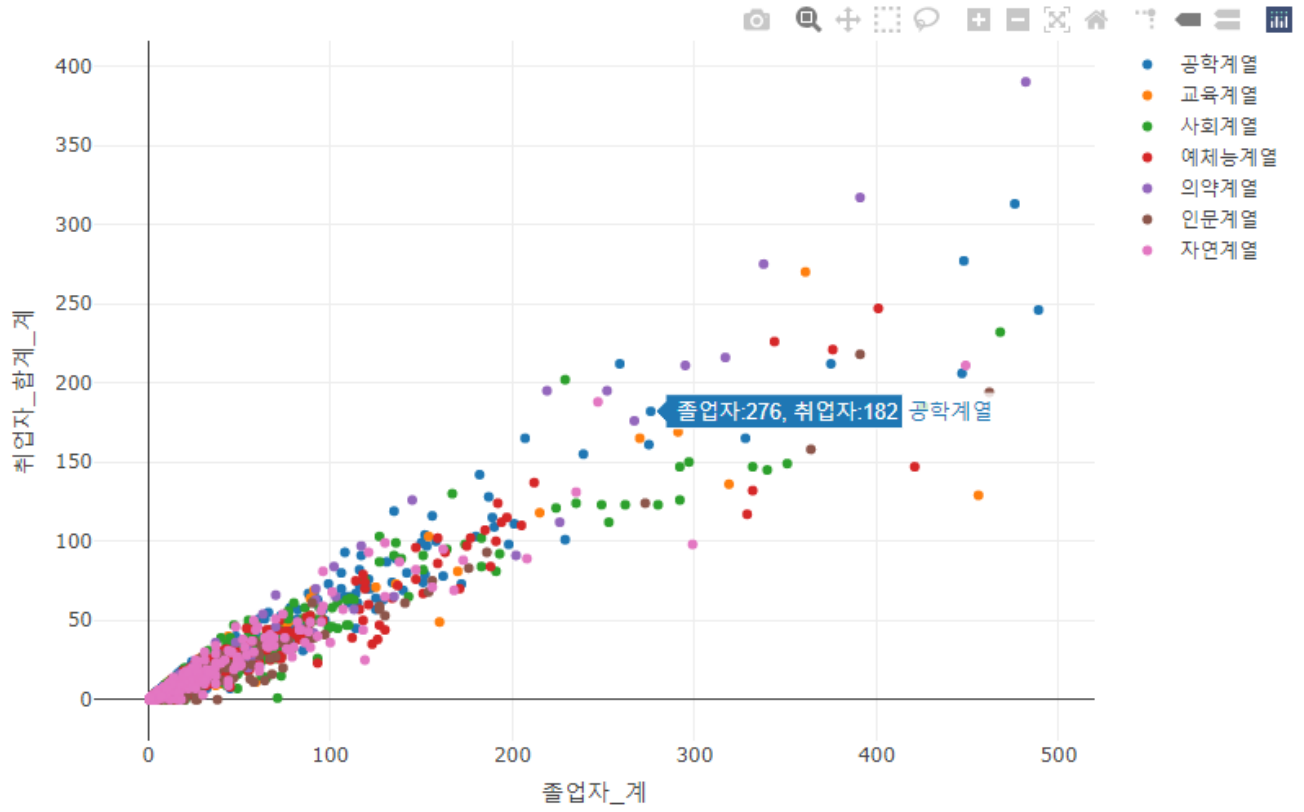
`hovertext` 는 X, Y 좌표에 표시되는 문자열을 설정하는 type 이다. `hovertext` 는 앞서 설명한 `hoverinfo` 를 `text` 로 설정하고 `hovertext` 를 설정할 때는 호버 상자 안에 `text` 설정값이 표시되지만 `hoverinfo` 를 생략하고 `hovertext` 만 설정하면 호버 상자밖에 표시된다는 차이가 있다. `hovertext` 에 단순 문자열을 설정하면 모든 데이터 호버에 동일한 문자열이 표기되지만 벡터를 설정하면 각각의 호버에 매핑된 결과가 표시된다.

```
df_취업통계_2000 |>
  plot_ly(x = ~졸업자_계, y = ~취업자_합계_계, name = ~대계열,
          hoverinfo = 'x+y+text', hovertext = ~중계열)
```



`hovertemplete` 는 호버 상자와 호버 상자에 표시되는 정보의 포맷을 설정하는 type 이다. 이 type 은 앞서 설명한 `hoverinfo` 에 설정된 사용되는 type 에 사용된 변수를 `%{변수}`의 형태로 사용할 수 있다. 예를 들어 Y 축 값을 표현하기 위해서는 `%{y}`로 설정한다. 앞서 사용된 예에서 X 축의 값에 졸업자, Y 축의 값에 취업자를 표기하고 값을 표시하는 코드는 다음과 같다.

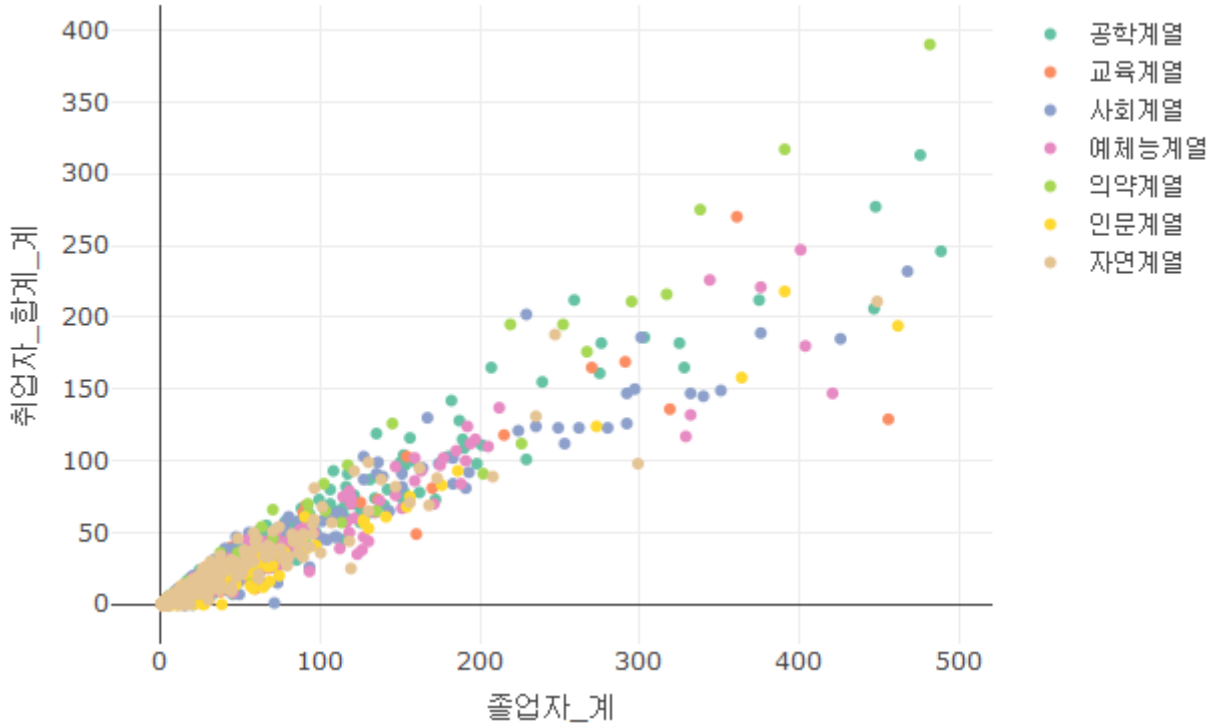
```
df_취업통계_2000 |>
  plot_ly(x = ~졸업자_계, y = ~취업자_합계_계, name = ~대계열,
          hoverinfo = 'x+y+text', hovertemplate = ' 졸업자:%{x}, 취업자:%{y}')
```



3.1.4. color, colors

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

```
df_취업통계_2000 |>
  plot_ly(x = ~졸업자_계, y = ~취업자_합계_계, color = ~대계열)
```

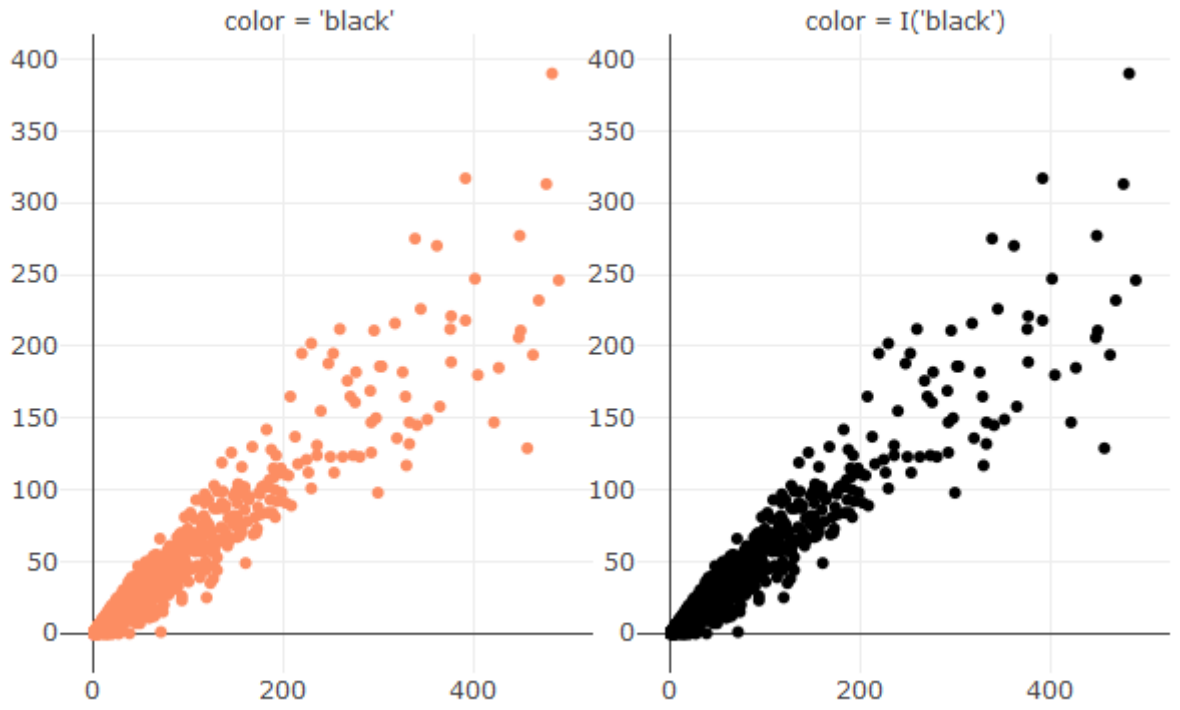


반면 특정한 색으로 설정할 때는 `color` 에 특정 색 이름을 설정하면 모든 marker 가 동일한 색으로 나타난다. 이 과정에서 하나 주의해야 할 것이 설정의 방법이다. 일반적인 변수 설정과 같이 색상명을 설정하면 다음과 같이 정확한 색상이 나타나지 않는다.

```
df_취업통계_2000 |>
  plot_ly(x = ~졸업자_계, y = ~취업자_합계_계, color = 'black')
```

`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 색을 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 'asis'를 의미하는 `I()`를 사용하여 색을 설정하여야 한다.

```
df_취업통계_2000 |>
  plot_ly(x = ~졸업자_계, y = ~취업자_합계_계, color = I('black'))
```



`colors` 는 `color` 에서 매핑된 변수에 따른 색의 스케일을 설정하는 매개변수이다. `plotly` 에서는 색의 스케일을 설정하는 팔레트를 선정할 때 다음의 세 가지 방법을 사용한다.

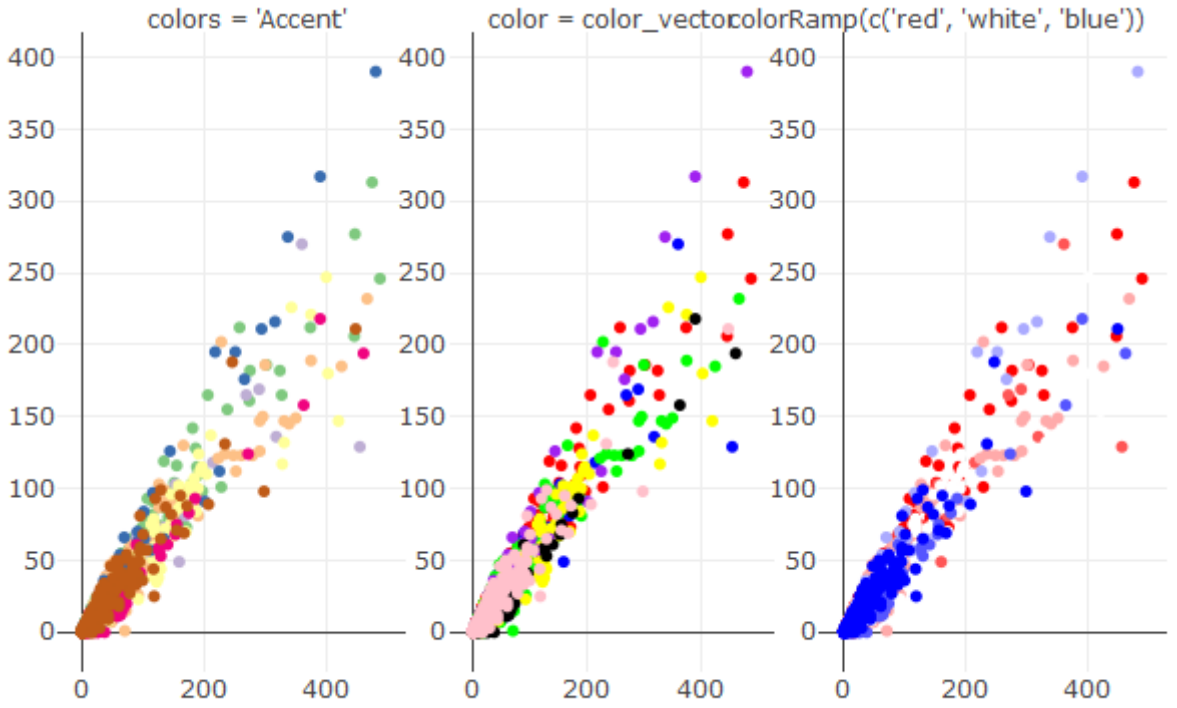
첫 번째 방법은 `RColorBrewer` 패키지에서 제공하는 팔레트의 이름을 설정하는 방법이다. `RColorBrewer` 패키지는 R 에서 가장 대중적으로 사용되는 색 팔레트를 제공하는 패키지로 `ggplot2` 에서도 많이 사용된다. 이 패키지에서 제공하는 팔레트의 이름을 `colors` 에 지정함으로써 해당 팔레트를 사용할 수 있다.

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 0 부터 1 까지의 값 범위내에 해당하는 색을 반환해준다.

```
df_취업통계_2000 |>
  plot_ly(x = ~졸업자_계, y = ~취업자_합계_계, color = ~대계열, colors = 'Accent')

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

df_취업통계_2000 |>
```

```
  plot_ly(x = ~졸업자_계, y = ~취업자_합계_계,
          color = ~대계열, colors = color_vector)

df_취업통계_2000 |>
  plot_ly(x = ~졸업자_계, y = ~취업자_합계_계,
          color = ~대계열, colors = colorRamp(c('red', 'white', 'blue')))
```

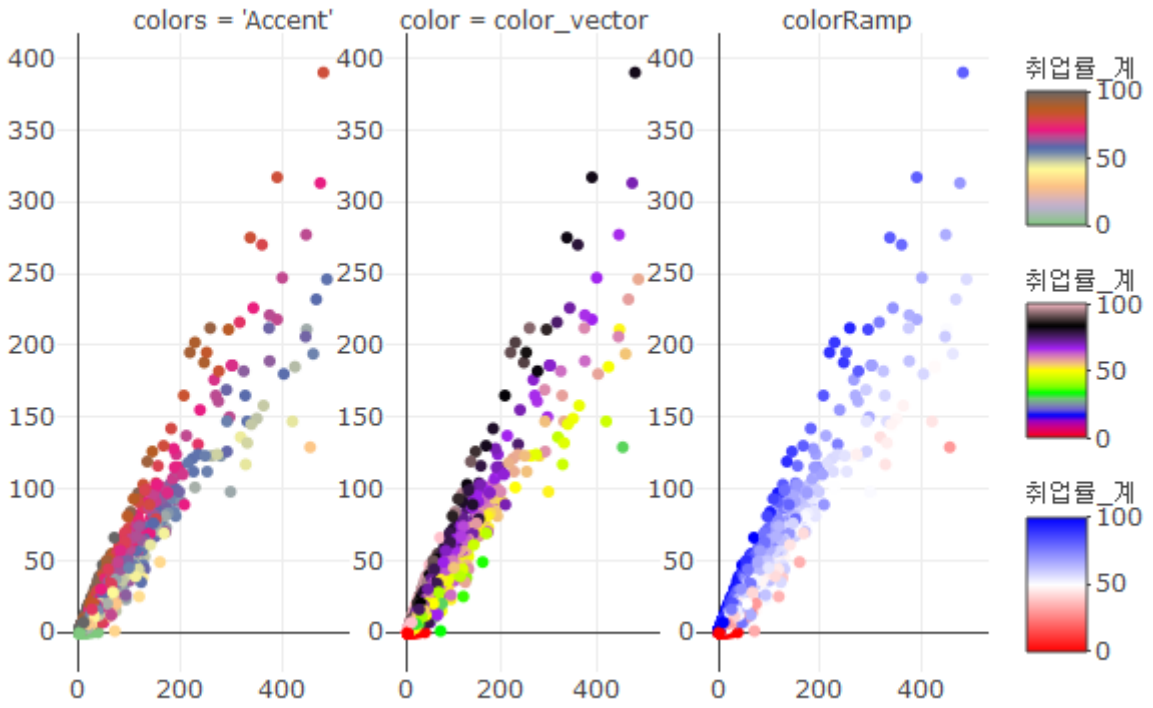


색 설정에서 하나 중요한 type 은 `colorbar()`이다. `color` type 에 매핑되는 변수가 이산형일 경우는 각각의 카테고리에 따라 설정된 색 팔레트에 정의된 색상이 설정되지만 연속형일 경우는 연속된 색상이 설정된다. 연속된 색상에서 변수에 따른 색상이 선택되어 표시된다. 이렇게 연속된 색상의 설정에 사용되는 색 스케일이 `colorbar()`이다.

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

컬러바 설정에 사용되는 세부 속성은 현재(22.03) 43 개가 제공된다. 이에 대한 세부 속성과 설명은 `plotly` 매뉴얼을 참조하라.[4]

```
df_취업통계_2000 |>
  plot_ly(x = ~졸업자_계, y = ~취업자_합계_계,
          color = ~취업률_계, colors = 'Accent')

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

df_취업통계_2000 |>
  plot_ly(x = ~졸업자_계, y = ~취업자_합계_계,
          color = ~취업률_계, colors = color_vector)

df_취업통계_2000 |>
  plot_ly(x = ~졸업자_계, y = ~취업자_합계_계,
          color = ~취업률_계, colors = colorRamp(c('red', 'white', 'blue')))
```

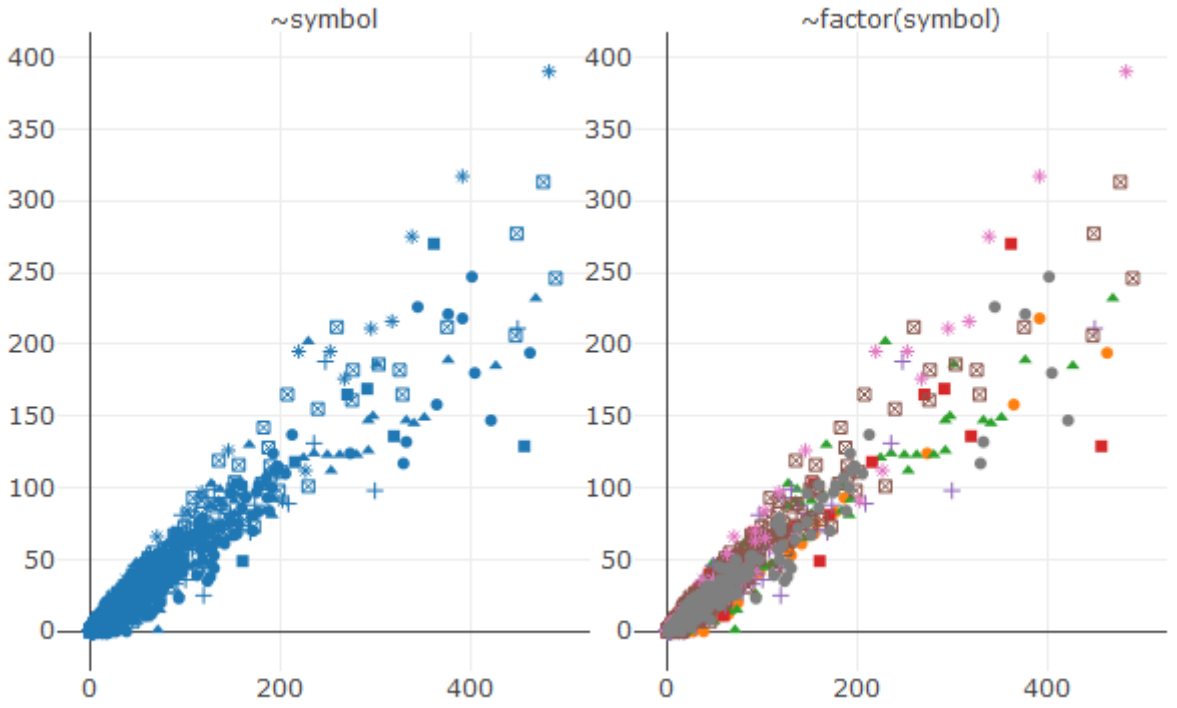


[4] https://plotly.com/r/reference/#scatter-marker-colorbar

3.1.5. symbol, symbols

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

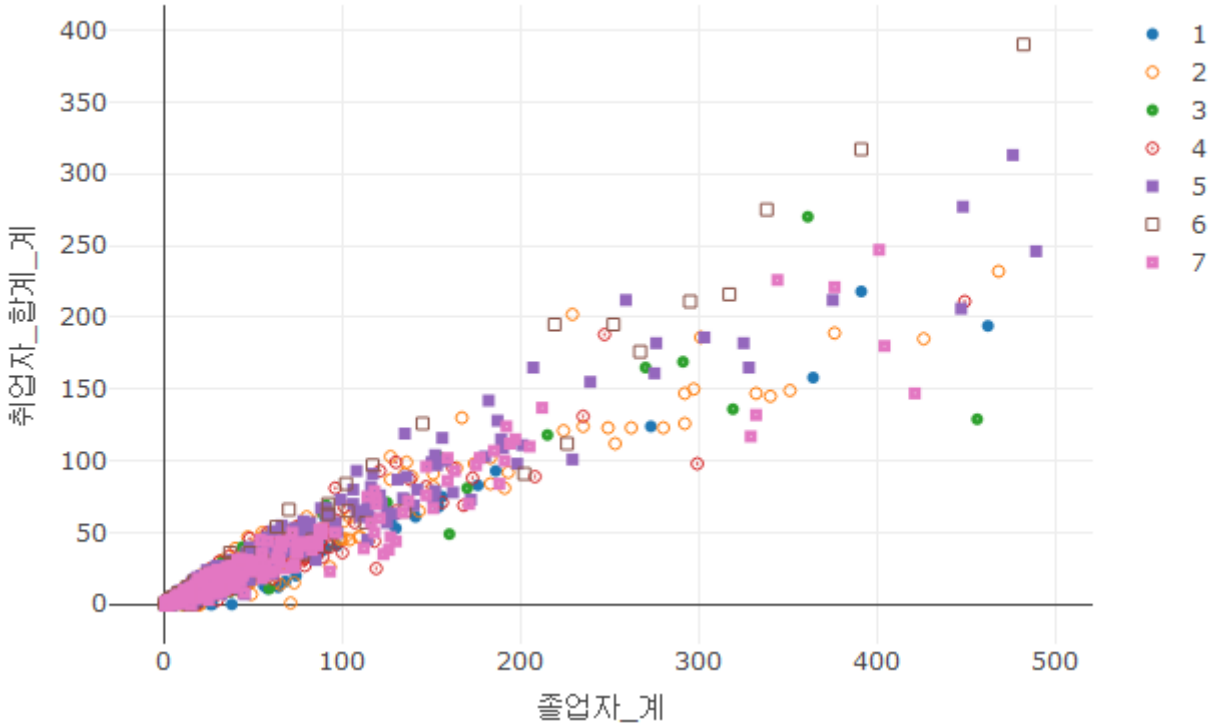
```
df_취업통계_symbol <- df_취업통계_2000 |>
  mutate(symbol = case_when(
    대계열 == '인문계열' ~ 1,
    대계열 == '사회계열' ~ 2,
    대계열 == '교육계열' ~ 3,
    대계열 == '자연계열' ~ 4,
    대계열 == '공학계열' ~ 5,
    대계열 == '의약계열' ~ 6,
    대계열 == '예체능계열' ~ 7)
    )

df_취업통계_symbol |>
  plot_ly(x = ~졸업자_계, y = ~취업자_합계_계, symbol = ~symbol)

df_취업통계_symbol |>
  plot_ly(x = ~졸업자_계, y = ~취업자_합계_계, symbol = ~factor(symbol))
```



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

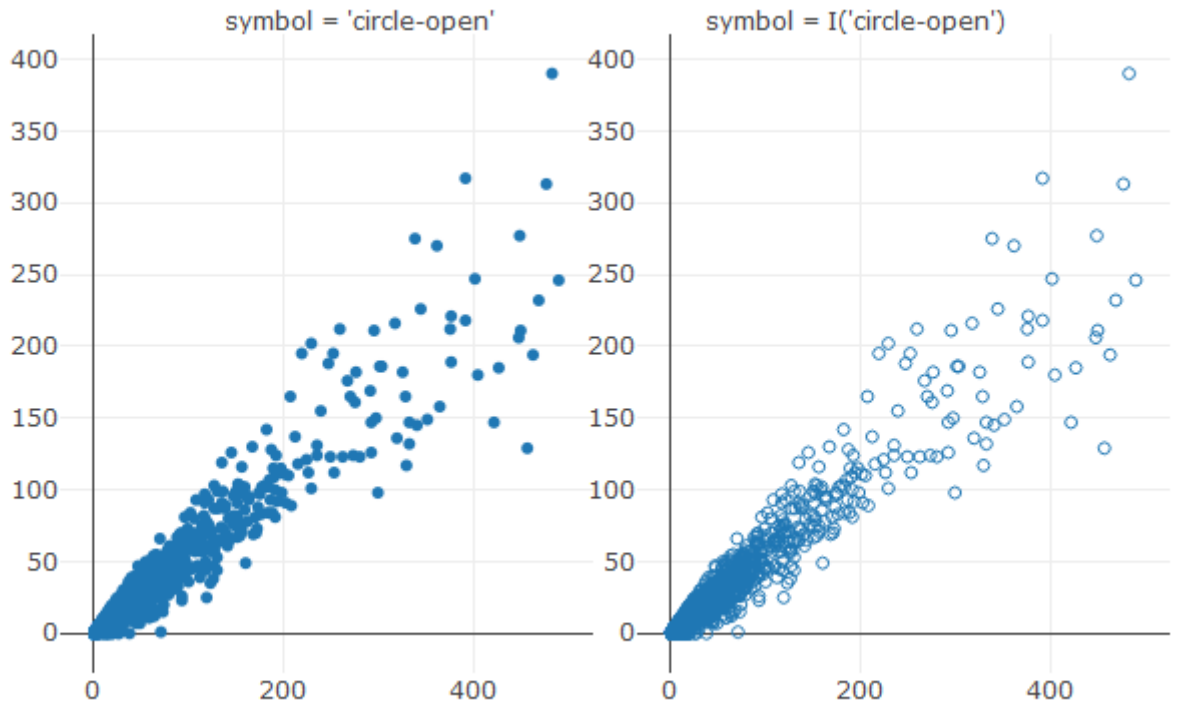
`plotly` 에서는 0 번부터 52 번까지 총 53 개의 `symbol` 을 제공한다. 이 기본 53 개의 도형 번호에 100 을 더하면 내부가 빈 'open'형 심볼, 200 을 더하면 점이 찍힌 'dot'형 심볼, 300 을 더하면 내부가 비고 점이 찍힌 'open-dot'형 심볼을 의미한다.

```
df_취업통계_symbol |>
  plot_ly(x = ~졸업자_계, y = ~취업자_합계_계,
          symbol = ~factor(symbol), symbols = c('circle', 'circle-open', 'circle
-dot', "circle-open-dot", "square", "square-open", "square-dot"))
```



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

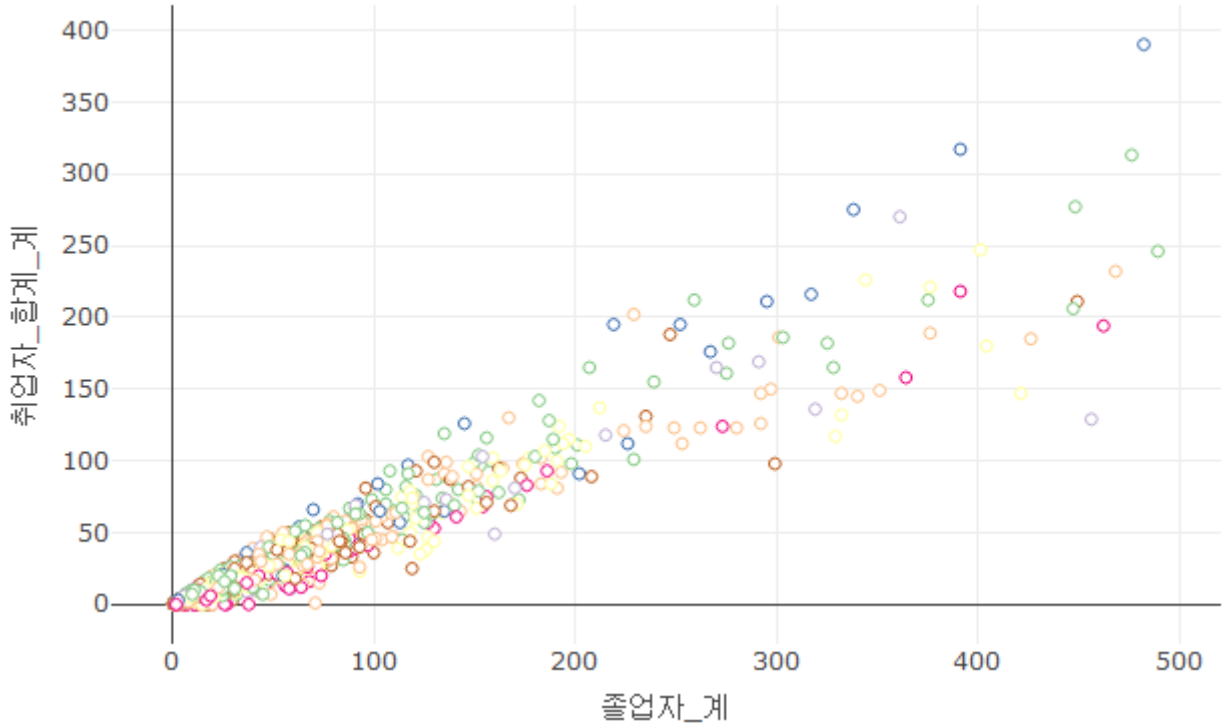
```
df_취업통계_symbol |>
  plot_ly(x = ~졸업자_계, y = ~취업자_합계_계, symbol = 'circle-open')

df_취업통계_symbol |>
  plot_ly(x = ~졸업자_계, y = ~취업자_합계_계, symbol = I('circle-open'))
```



3.1.6. stroke, strokes

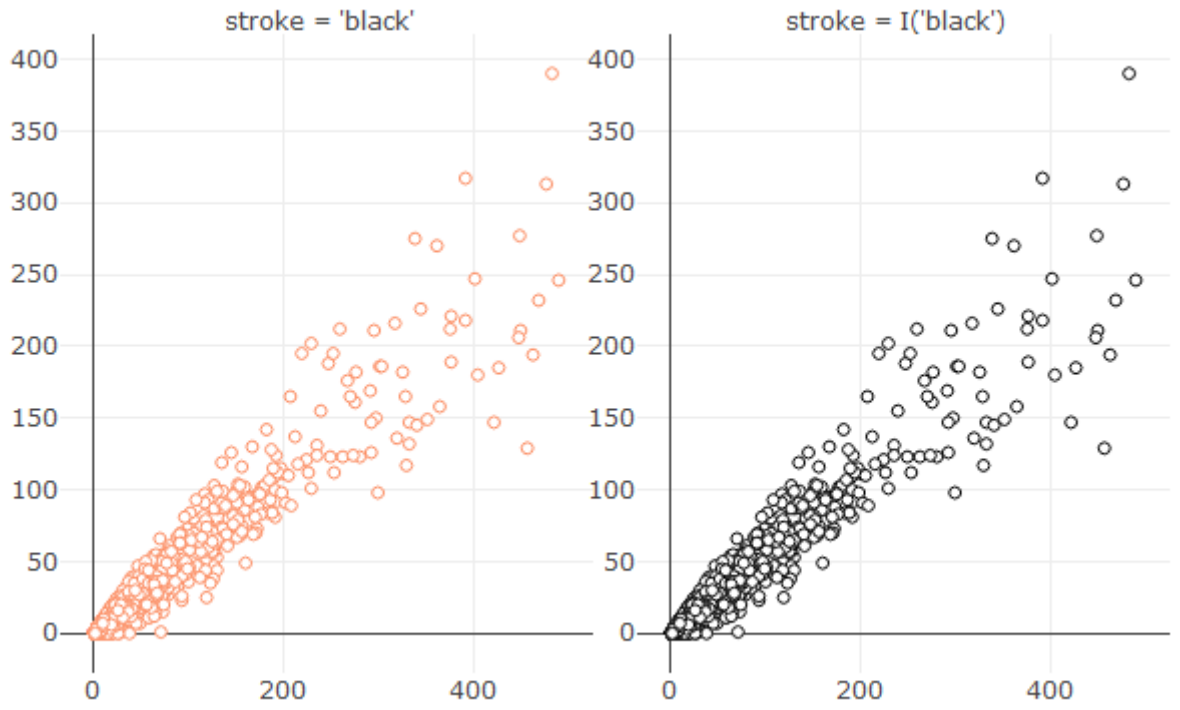
`stroke` 는 점의 외곽선 색을 설정하는 속성이다. `color` 의 설정은 점의 내부색과 외곽선 색을 동시에 설정하지만 외곽선 색을 따로 매핑해야할 경우에 `stroke` 를 사용하여 매핑하거나 설정할 수 있다. `stroke` 와 `strokes` 는 `color` 나 `symbol` 과 같이 변수 매핑으로 사용되는 매개변수는 `stroke` 이고 매핑된 변수의 카테고리에 해당하는 외곽선 색을 설정하는 매개변수는 `strokes` 이다. `strokes` 에 색을 설정하는 방법은 `colors` 에서 사용한 세 가지 방법을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

```
df_취업통계_2000 |>
  plot_ly(x = ~졸업자_계, y = ~취업자_합계_계,
          stroke = ~대계열, strokes = 'Accent', color = I('white'))
```



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

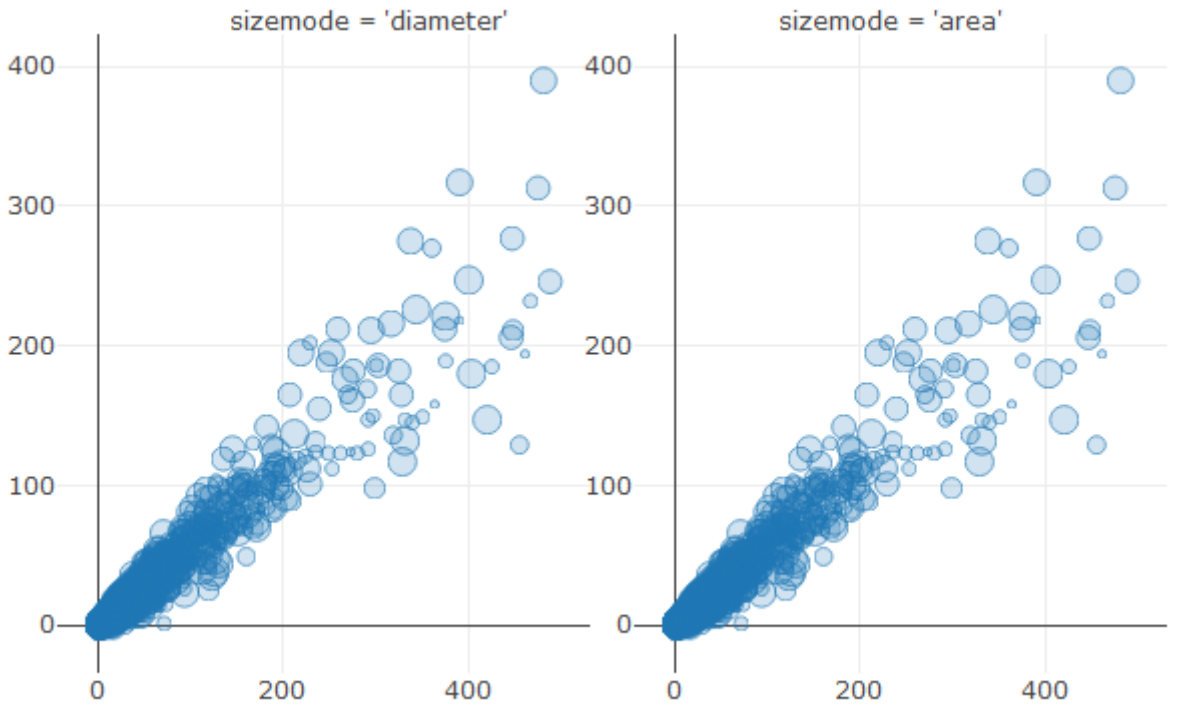
```
df_취업통계_2000 |>
  plot_ly(x = ~졸업자_계, y = ~취업자_합계_계,
          stroke = 'black', color = I('white'))

df_취업통계_2000 |>
  plot_ly(x = ~졸업자_계, y = ~취업자_합계_계,
          stroke = I('black'), color = I('white'))
```



3.1.7. size, sizes

`size`는 점의 크기를 설정하는 속성이다. `size`는 변수 매핑을 위한 매개변수이고 `sizes`는 변수에 매핑된 카테고리별로 크기를 설정하는 매개변수이다. 앞선 속성들과 달리 `size`는 `sizemode` 매개변수를 추가적으로 사용할 수 있다. `sizemode`는 점의 크기를 결정하는 기준을 설정하는 매개변수로 점의 크기를 지름으로 설정하는 'diameter'와 면적으로 설정하는 'area'의 두 가지가 있다.

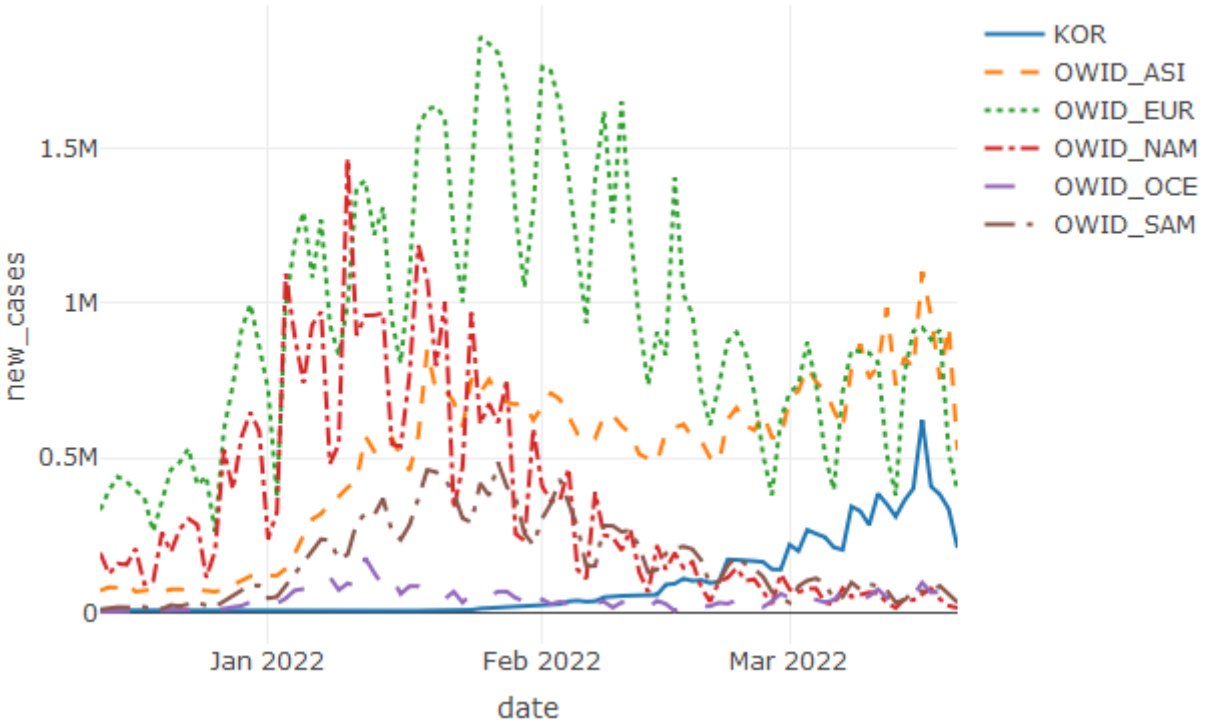
```
df_취업통계_symbol |>
  plot_ly(x = ~졸업자_계, y = ~취업자_합계_계,
          size = ~symbol, sizemode = 'diameter', alpha = 0.3)

df_취업통계_symbol |>
  plot_ly(x = ~졸업자_계, y = ~취업자_합계_계,
          size = ~symbol, sizemode = 'area', alpha = 0.3)
```



3.1.8. linetype, linetypes

`linetype` 은 선의 형태를 설정하는 type 이다. `plotly` 에서 선의 형태를 설정할 때 변수 매핑을 통해 설정하기 위해서는 `linetype` 을 사용하고 `linetype` 에 매핑된 변수에 따라 선의 형태를 설정하기 위해서는 `linetypes` 를 사용한다.

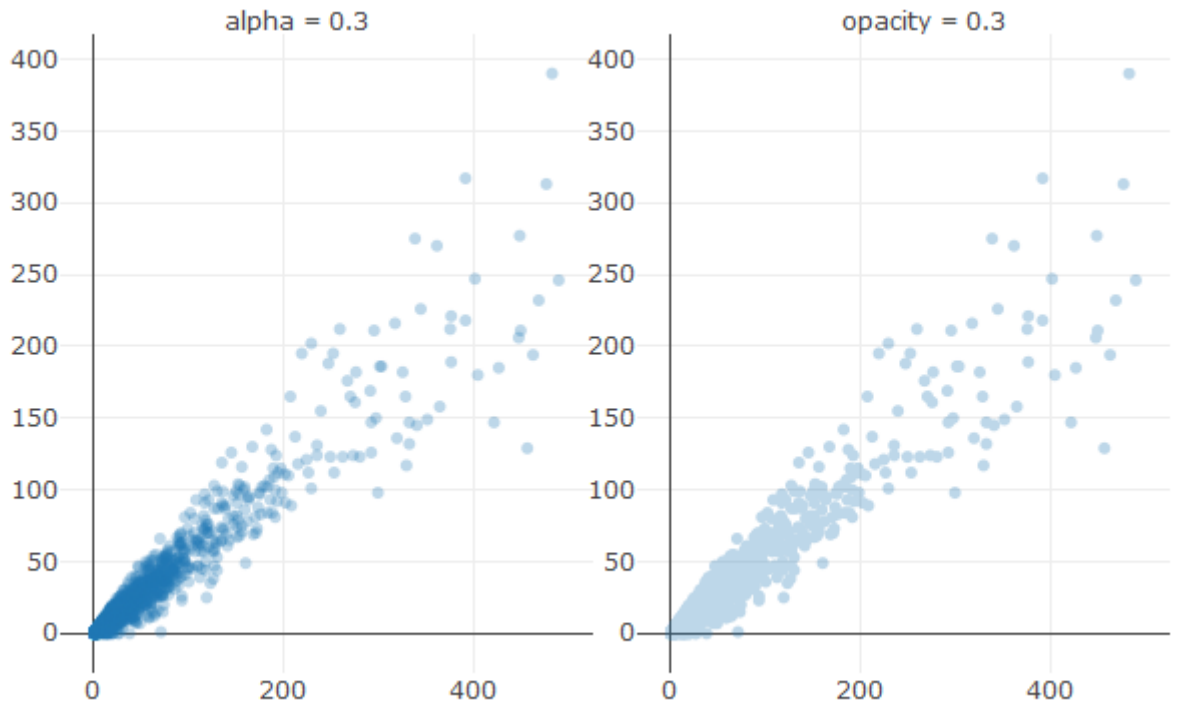
```
covid19_df_100 |>
  plot_ly(type = 'scatter', mode = 'lines', x = ~date, y = ~new_cases, linetype
= ~iso_code)
```



3.1.9. 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 ㅇ낳고 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.

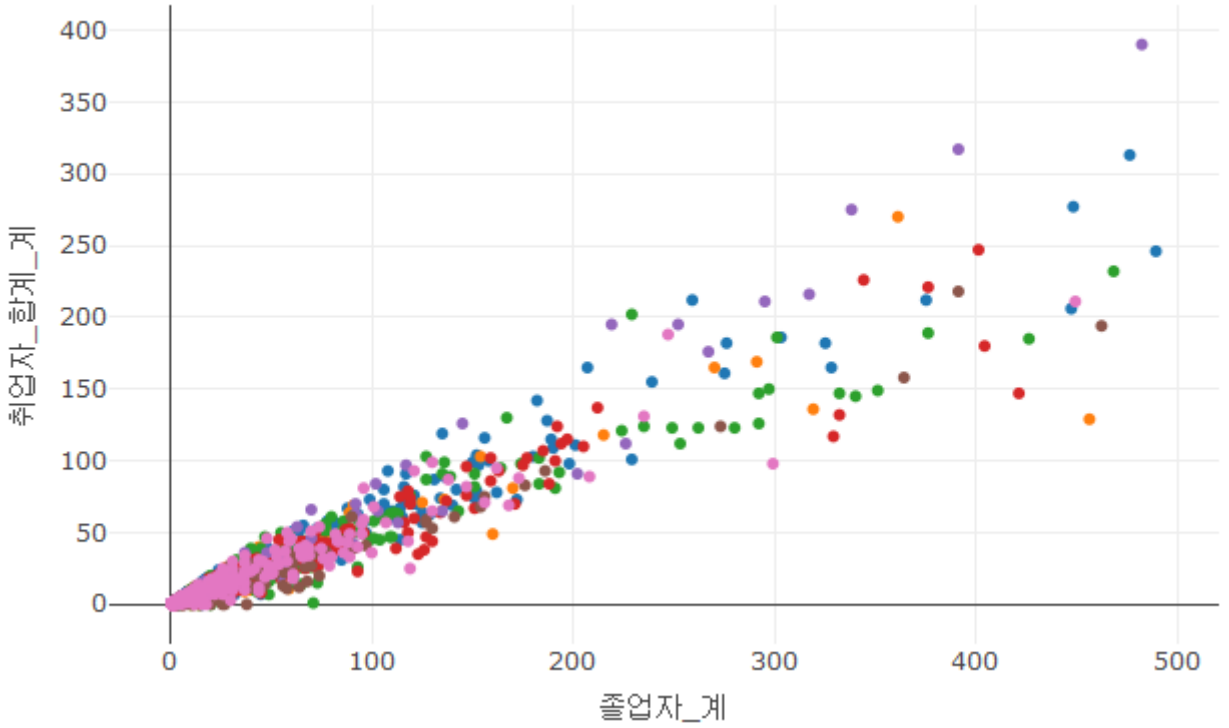
```
df_취업통계_2000 |>
  plot_ly(x = ~졸업자_계, y = ~취업자_합계_계, alpha = 0.3)

df_취업통계_2000 |>
  plot_ly(x = ~졸업자_계, y = ~취업자_합계_계, opacity = 0.3)
```



3.1.10. showlegend

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

```
df_취업통계_2000 |>
  plot_ly(x = ~졸업자_계, y = ~취업자_합계_계, name = ~대계열, showlegend = FALSE)
```



3.2. trace 추가 : add_trace()

trace 는 `plotly` 객체에서 표현되는 단일 데이터 아이템을 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. 하지만 `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 데이터로 전환하여 사용하였다. 여기에 예외로 사용되는 것이 `geom_line()`인데 넓은 형태의 데이터프레임에 각 열을 `geom_line()`을 사용하여 각각의 선 레이어로 생성하고 이들을 계속 겹쳐 그림으로써 전체적인 선 그래프를 완성할 수 있다. 하지만 `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 해야할 각각의 열을 각각 trace 로 추가하여 줌으로써 시각화를 완성한다. `ggplot2` 의 선 그래프에 가능한 방법을 모든 시각화에서 사용할 수 있다는 것이다.

`plotly` 에서는 trace 를 표현하는데 40 가지 이상의 시각화 type 을 제공한다. 또 각각의 trace 의 시각화 type 에는 그 에는 해당 trace 의 시각화 세부 속성을 설정을 위한 수많은 세부 type 을 가지고 있다. 앞서 `plot_ly()`에서 매개변수로 설정했던 type 들은 모두 개별 trace 의 설정으로도 사용이 가능하다. 하지만 `plot_ly()`에서 trace 의 type 을 설정하지 않았는데 시각화의 결과는 데이터들이 trace 로 표현되어 나타났다. 이는 `plotly` 에서 데이터를 분석하여 가장 효과적인 trace 를 매칭해주었다. 하지만 trace 는 사용자가 지정하는 것이 일반적이다.

이 trace 와 type 을 설정하는 방법은 `add_trace()`를 사용하여 trace 와 trace 에 속한 type 속성을 설정하는 방법과 `add_markers()`, `add_lines()`등과 같이 `add_trace()`에서 파생된 래핑 함수를 사용하는 방법이 있다.

`add_scatter()`의 주요 사용법은 다음과 같다.

```
add_trace(p, …, data = NULL, inherit = TRUE)
- p : plot_ly()로 생성한 plotly 객체
- … : 스캐터 trace 에 설정할 수 있는 속성 설정
- data : 시각화할 데이터프레임
- inherit : plot_ly()에 설정된 속성 type 을 상속할지를 결정하는 논리값
```

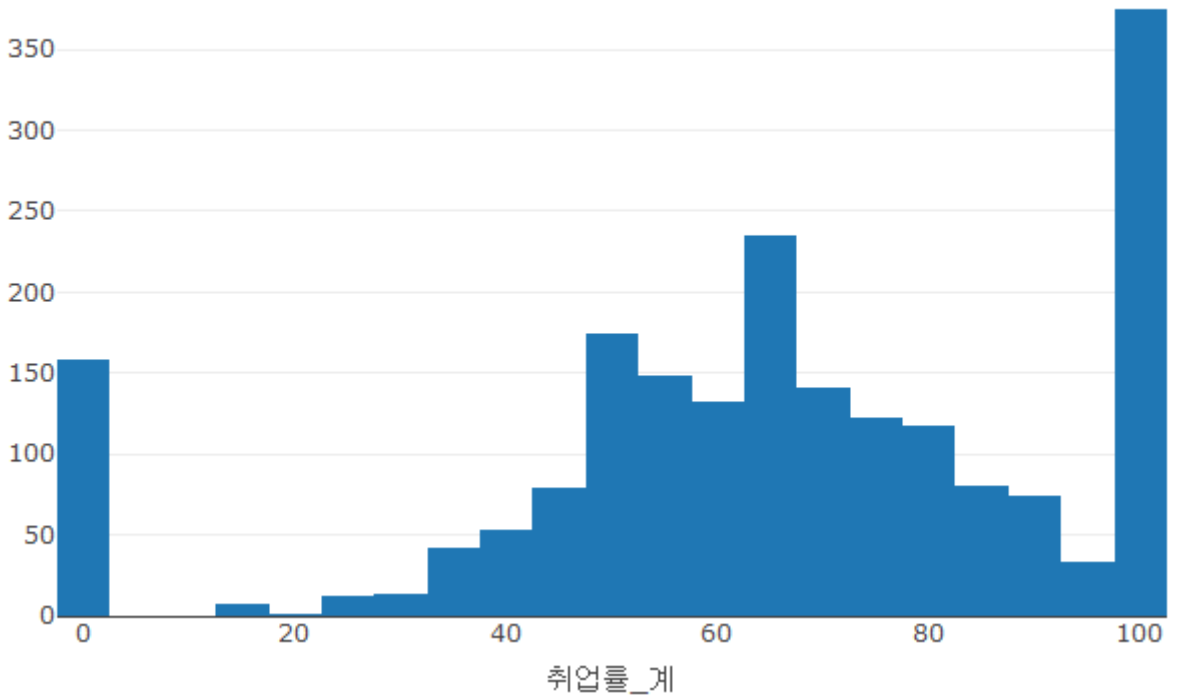
trace 의 추가를 설명하기 전에 먼저 공통적으로

3.2.1. 히스토그램(histogram) trace

`plotly` 에서 히스토그램 시각화를 위한 trace type 은 'histogram'이다. 히스토그램은 하나의 변수에 따른 그 사례 빈도를 산출하는 시각화이고 X 축에 매핑되는 변수외에 간격(bin)을 설정함으로써 히스토그램을 생성할 수 있다. 히스토그램 trace 는 `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 trace 에 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_histogram <- df_취업통계_2000 |> plot_ly()

p_histogram |> add_trace(type = 'histogram', x = ~취업률_계, name = '취업률')
```



`plotly` 의 히스토그램 trace 에서 사용되는 주요 type 들은 `plot_ly()`에서 사용되는 대부분의 type 을 사용할 수 있다. 그 외에 히스토그램에서 고유하게 사용되는 type 은 다음과 같다.

3.2.1.1. xbins

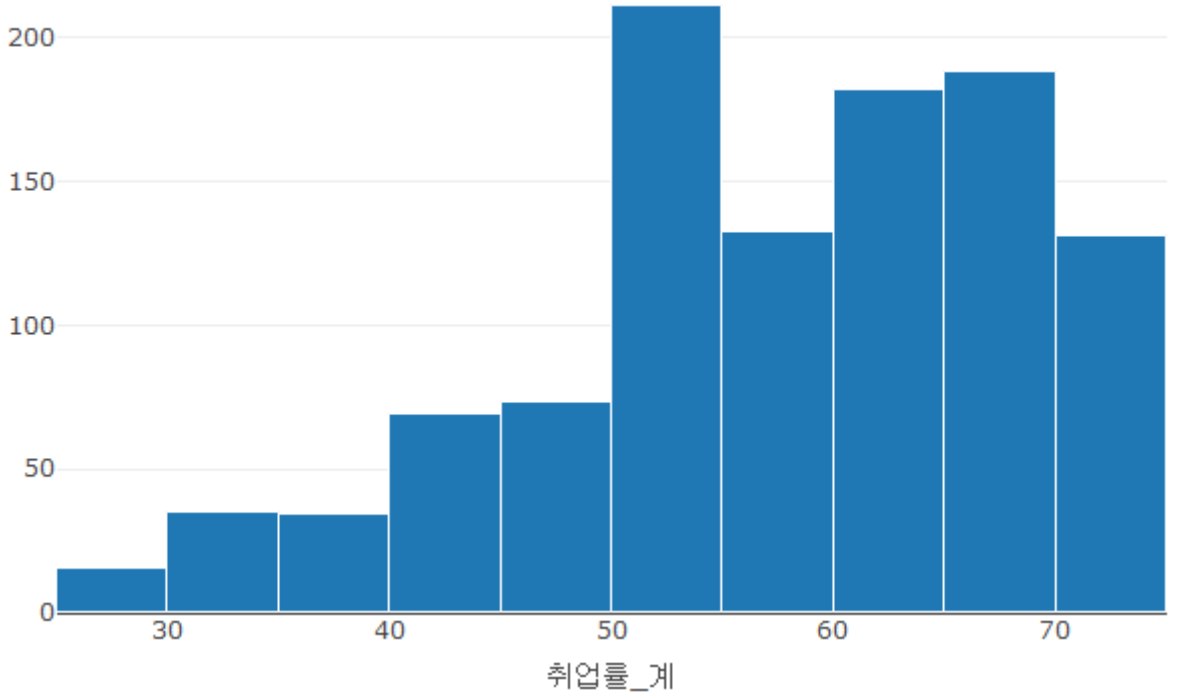
`xbins` 는 X 축에 매핑된 변수가 연속형 수치 변수일 경우 빈도를 표현할 구간의 범위를 설정하는 type 이다. `xbins` 는 바로 수치나 벡터를 설정하는 것이 아니고 `xbins` 에 속하는 하위

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_계,
                         xbins = list(start = 25, end = 75, size = 5),
                         stroke = I('white'))
```

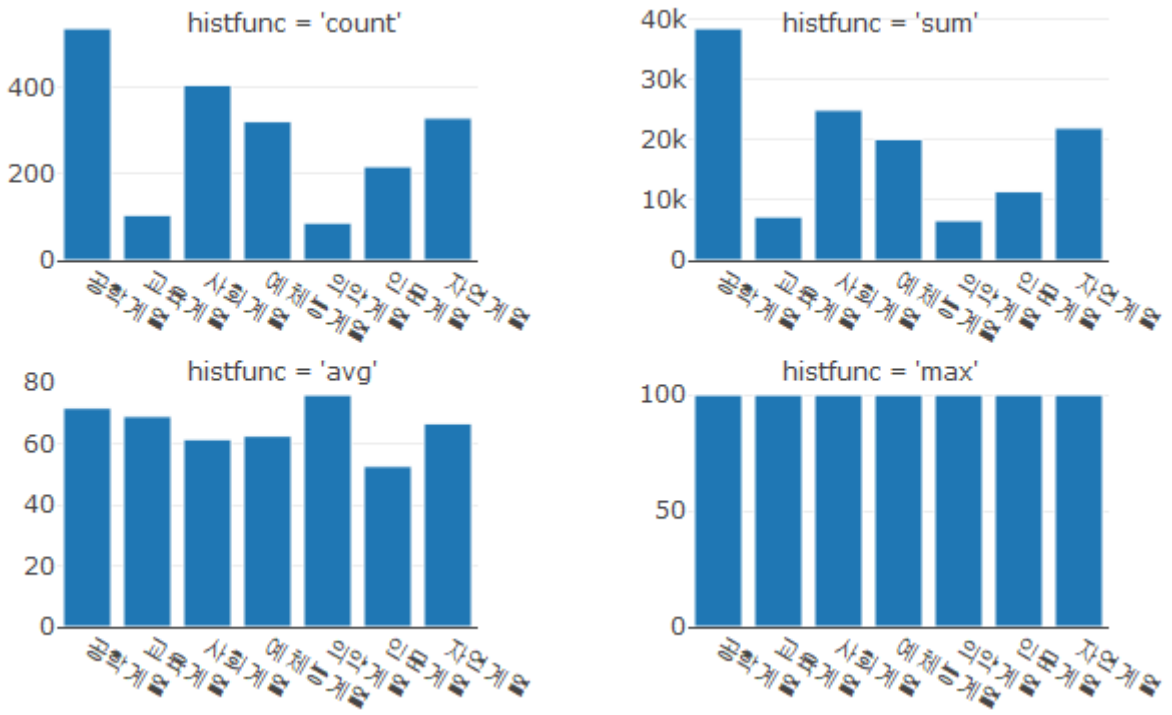


3.2.1.2. histfunc

`histfunc` 은 히스토그램의 막대로 표현하는 값의 종류를 설정하는 type 이다. 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.

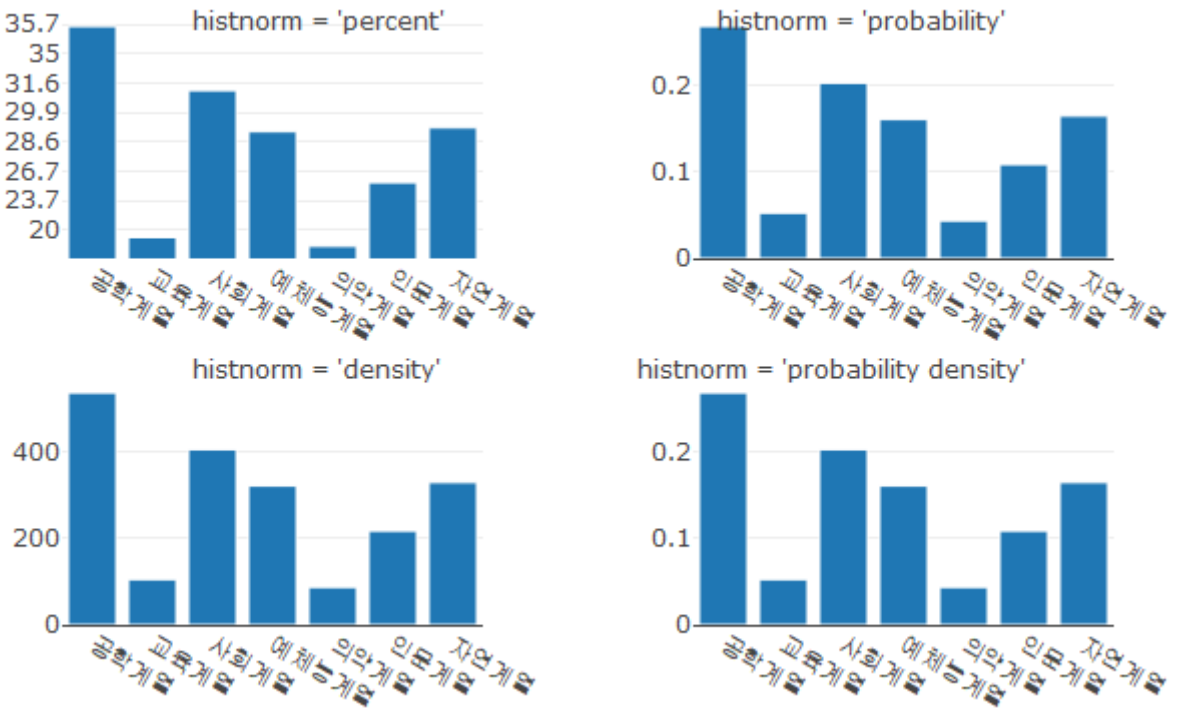
```
p_histogram |> add_trace(type = 'histogram', x = ~대계열,
                         stroke = I('white'), histfunc = 'count')

p_histogram |> add_trace(type = 'histogram', x = ~대계열, y = ~as.character(취업률_계),
                         histfunc = 'sum', stroke = I('white')) |>
  layout(yaxis=list(type='linear'))

p_histogram |> add_trace(type = 'histogram', x = ~대계열, y = ~as.character(취업률_계),
                         histfunc = 'avg', stroke = I('white')) |>
  layout(yaxis=list(type='linear'))

p_histogram |> add_trace(type = 'histogram', x = ~대계열, y = ~as.character(취업률_계),
                         histfunc = 'max', stroke = I('white')) |>
  layout(yaxis=list(type='linear'))
```



3.2.1.3. histnorm

`histnorm`은 히스토그램 trace 로 표시되는 데이터의 정규화(normalization)에 대한 type 이다. `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 나타닌다. 'density'는 빈의 발생 빈도에 따른 밀도를 표현하고, 'probability density'은 각 막대 구간에 데이터가 존재할 밀도 확률을 면적으로 변환하여 나타내기 때문에 모든 막대 면적의 합은 1 이 된다.

```
p_histogram |> add_trace(type = 'histogram', x = ~대계열, y = ~as.character(취업률_계),
                          stroke = I('white'), histnorm = 'percent')

p_histogram |> add_trace(type = 'histogram', x = ~대계열, y = ~as.character(취업률_계),
                          histnorm = 'probability', stroke = I('white')) |>
  layout(yaxis=list(type='linear'))

p_histogram |> add_trace(type = 'histogram', x = ~대계열, y = ~as.character(취업률_계),
                          histnorm = 'density', stroke = I('white')) |>
  layout(yaxis=list(type='linear'))

p_histogram |> add_trace(type = 'histogram', x = ~대계열, y = ~as.character(취업률_계),
                          histnorm = 'probability density', stroke = I('white'))
|>
  layout(yaxis=list(type='linear'))
```



3.2.2. 스캐터(scatter) trace : 산점도, 선 그래프

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

스캐터 trace 에서 약 50 여개가 넘는 type 을 사용하여 세부 설정이 가능하다. 이 50 여개의 type 중에는 다시 세부 type 을 설정하는 type 이 있고 총 4 단계까지 내려가는 type 도 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 에 수록되어 있다.[5]

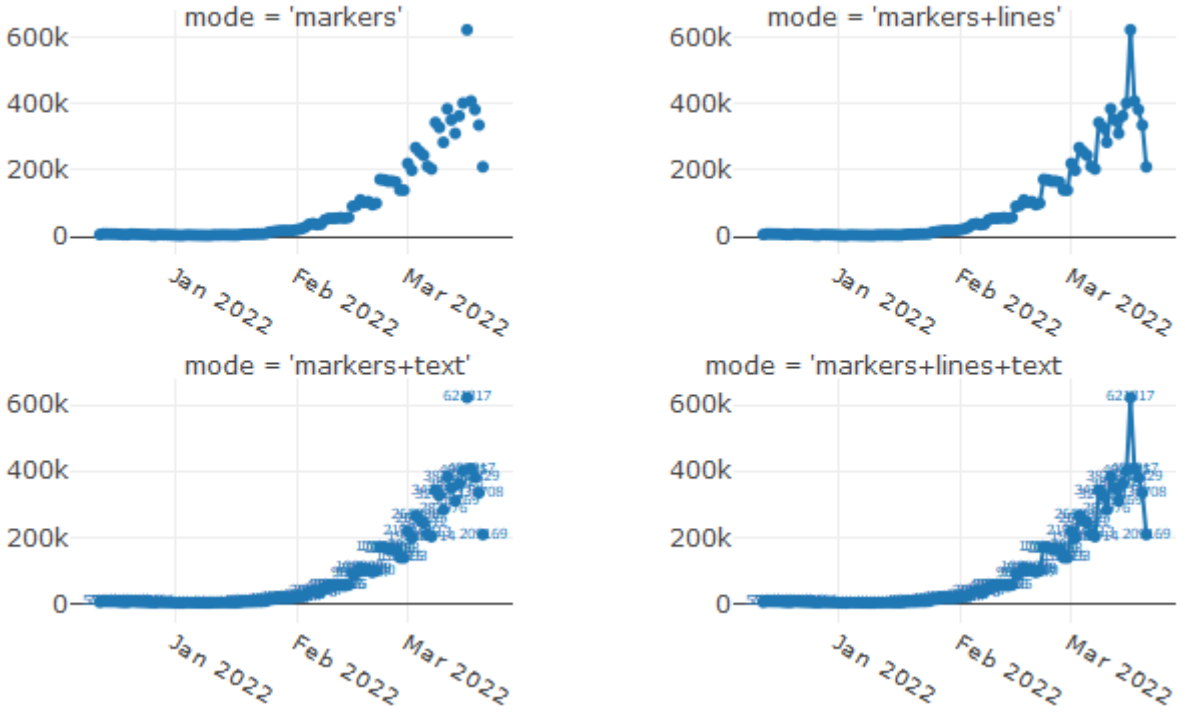
[5] https://plotly.com/r/reference/scatter/

3.2.2.1. 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.

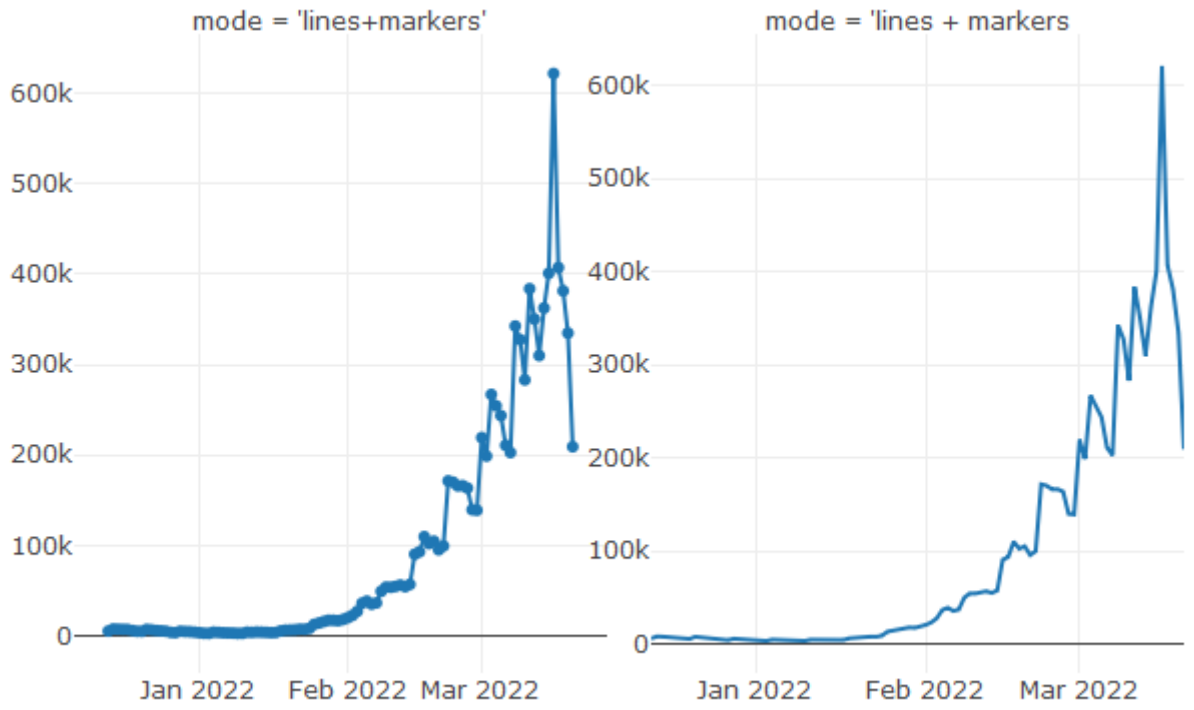
mode	설명
markers	데이터를 점으로 표시
lines	데이터를 서로 이어주는 선을 표시
text	데이터를 문자열로 표시
none	데이터를 표시하지 않음

```
p_line_wide <- covid19_df_100_wide |> plot_ly()

p_line_wide |> add_trace(type = 'scatter', mode = 'markers', x = ~date, y = ~Sou
th_Korea)

p_line_wide  |> add_trace(type = 'scatter', mode = 'markers+lines', x = ~date, y
 = ~South_Korea)

p_line_wide  |> add_trace(type = 'scatter', mode = 'markers+text', x = ~date, y
= ~South_Korea, text = ~South_Korea)

p_line_wide  |> add_trace(type = 'scatter', mode = 'markers+lines+text', x = ~da
te, y = ~South_Korea, text = ~South_Korea)
```



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.

```
p_line_wide  |> add_trace(type = 'scatter', mode = 'markers+lines', x = ~date, y
 = ~South_Korea)

p_line_wide  |> add_trace(type = 'scatter', mode = 'markers + lines', x = ~date,
 y = ~South_Korea)
```

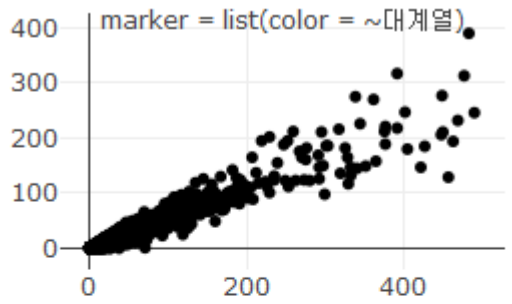


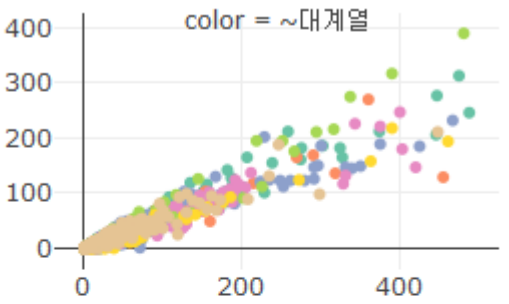
3.2.2.2. marker, add_markers()

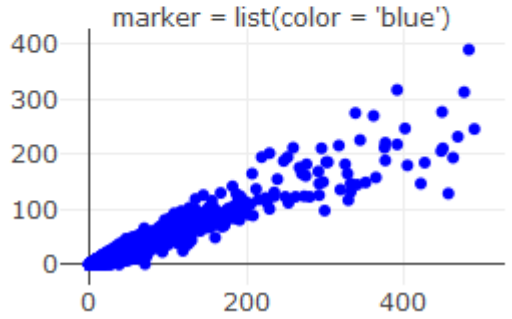
스캐터 trace 의 `marker` 타입은 스캐터 trace 에서 데이터를 표현하는 점의 속성을 설정하는 type 이다. `marker` 는 `add_trace()`에 `list()`를 사용하여 설정한다. 주의할 것은 `marker` 의 list 에 설정하는 type 들은 값을 설정할 수는 있지만 `~`를 사용해 변수를 매핑할 수 없다는 점이다. `marker` list 에 `~`를 사용한 변수 매핑을 사용한다고 해도 에러를 내지는 않지만 원하는 결과를 얻을 수는 없다. 또 하나 주의해야 하는 점은 다른 type 의 설정시 사용했던 `I()`의 사용이다. 앞서 색 type 과 같은 일부 type 의 설정시에 `I()`를 사용한다고 설명했는데 `marker` 로 설정하는 list 안에서는 `I()`를 사용하지 않아도 색의 설정이 가능하다. `marker` list 는 변수의 매핑에 사용하는 것이 아니고 값을 설정에 사용하는 것이기 때문에 구지 `I()`를 사용하여 매핑과 설정을 구분해 줄 필요가 없다는 것이다.

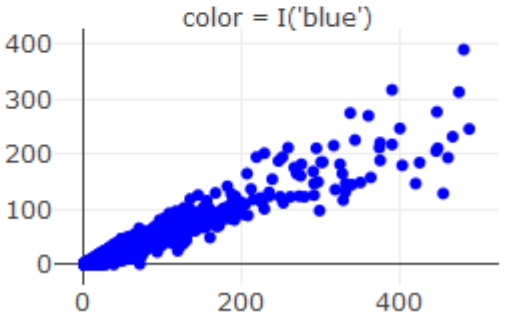
```
p_marker <- df_취업통계_2000 |> plot_ly()

p_marker |>
  add_trace(type = 'scatter', mode = 'markers', x = ~졸업자_계, y = ~취업자_합계_
계,
            marker = list(color = ~대계열))

p_marker |>
```

```
  add_trace(type = 'scatter', mode = 'markers', x = ~졸업자_계, y = ~취업자_합계_
계,
            color = ~대계열)

p_marker |>
  add_trace(type = 'scatter', mode = 'markers', x = ~졸업자_계, y = ~취업자_합계_
계,
            marker = list(color = 'blue'))

p_marker |>
  add_trace(type = 'scatter', mode = 'markers', x = ~졸업자_계, y = ~취업자_합계_
계,
            color = I('blue'))
```









add_trace(p, type = ‘ sactter ’ , mode = ‘ markers ’ , marker = NULL, …, data = NULL, inherit = TRUE)
add_markers(p, x = NULL, y = NULL, z = NULL, …, data = NULL, inherit = TRUE)
- p : plot_ly()로 생성한 plotly 객체
- type : trace 타입을 설정, add_markers()는 ‘ scatter ’ 로 설정
- mode : ’ scatter ’ trace 중 어떤 도형을 사용할지 설정, add_markers()는 ’ markers ’ 를 설정

```
- marker : marker의 type 설정을 위한 매개변수 list 설정
- … : 스캐터 trace의 markers 모드에 설정할 수 있는 속성 설정
- data : 시각화할 데이터프레임
- inherit : plot_ly()에 설정된 속성 type을 상속할지를 결정하는 논리값
- x : X축에 매핑할 변수를 ~로 설정
- y : Y축에 매핑할 변수를 ~로 설정
- z : Z축에 매핑할 변수를 ~로 설정
```

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

3.2.2.3. line, add_lines()

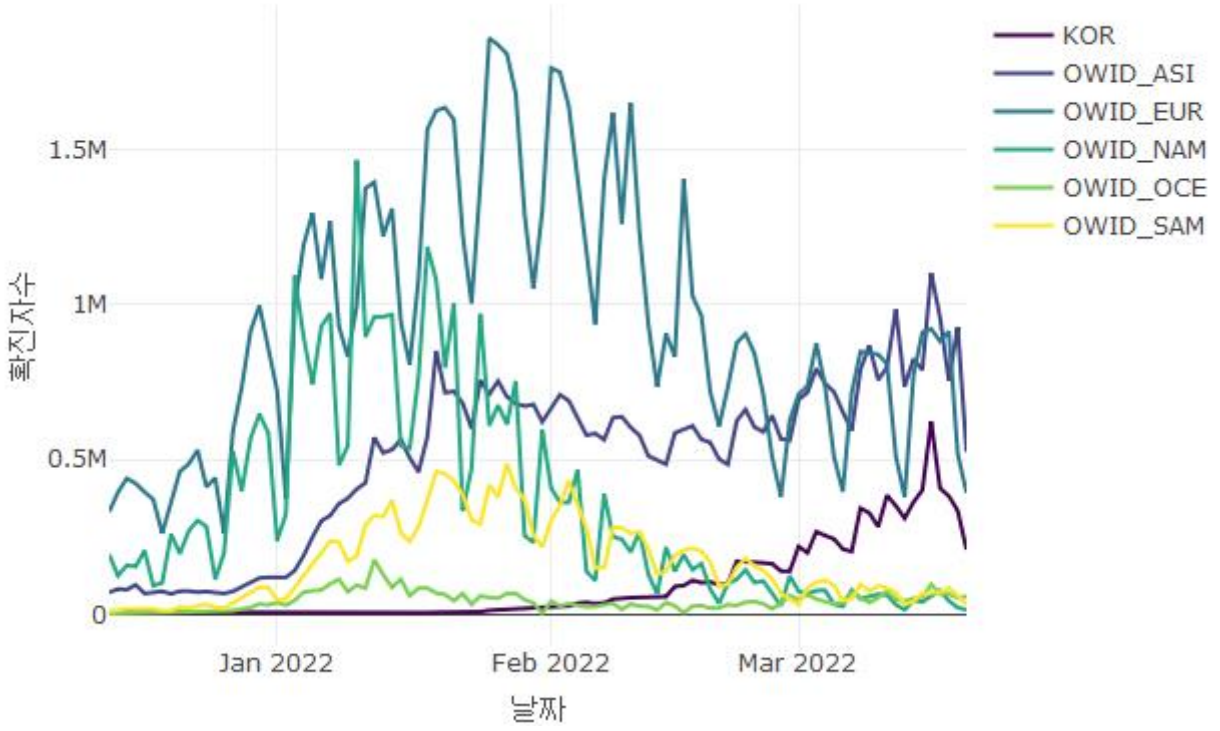
스캐터 trace의 `line` 타입은 스캐터 trace에서 데이터를을 이어주는 선의 속성을 설정하는 type이다. `line`은 `marker`와 같이 `add_trace()`에 `list()`를 사용하여 설정한다. 마찬가지로 주의할 것은 `line`의 list에 설정하는 type들은 값을 설정할 수는 있지만 `~`를 사용해 변수를 매핑할 수 없다는 점이고 `I()`를 사용하지 않고 값의 설정이 가능하다는 것이다.

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

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

trace 의 `type` 과 `mode` 를 선그래프에 맞에 설정하거나 `add_lines()`로 선그래프를 만들려면 먼저 X, Y, Z 축으로 매핑할 변수를 설정한다.선 그래프는 앞서 그려본 산점도와는 달리 데이터들을 선으로 연결해 줄 그룹 변수가 필요하다. 이는 일반적으로 `color`, `linetype` 으로 매핑된 변수를 기준으로 그룹화하여 선을 연결한다.

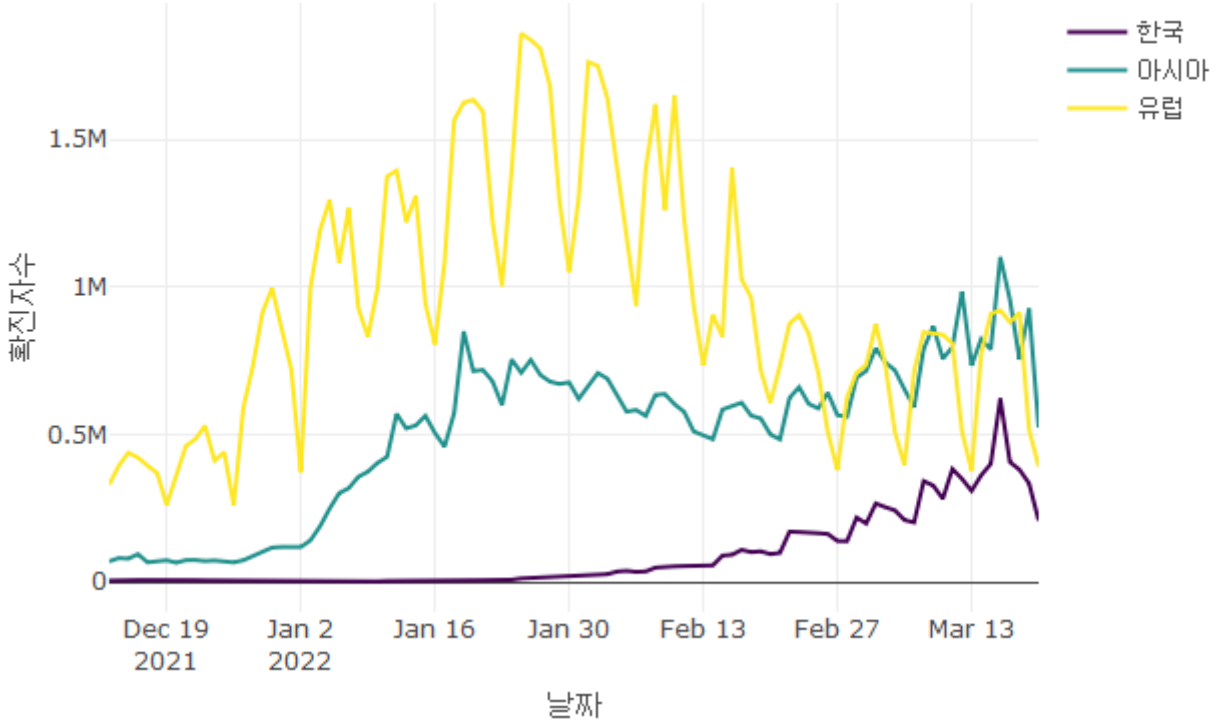
```
p_line <- covid19_df_100 |> plot_ly()

p_line |>
  add_trace(type = 'scatter', mode = 'lines', x = ~date, y = ~new_cases, color =
 ~ iso_code, colors = 'viridis') |>
  layout(xaxis = list(title = '날짜'), yaxis = list(title = '확진자수'))
```



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 3~4 개의 단일 선 그래프를 겹쳐 그리는 방법도 많이 사용된다. 이렇게 여러 개의 trace 를 겹쳐 그린다면 `plotly` 객체에 trace 를 순차적으로 추가하여 생성할 수 있다.

```
fig <- add_trace(p_line_wide, type = 'scatter', mode = 'lines', x = ~date,
                 y = ~South_Korea, name = '한국')
```

```
fig <- fig |> add_trace(type = 'scatter', mode = 'lines', x = ~date, y = ~Asia,
                        name = '아시아')

fig |> add_trace(type = 'scatter', mode = 'lines', x = ~date, y = ~Europe,
                 name = '유럽') |>
  layout(xaxis = list(title = '날짜'), yaxis = list(title = '확진자수'), colorway
= viridisLite::viridis(3))
```



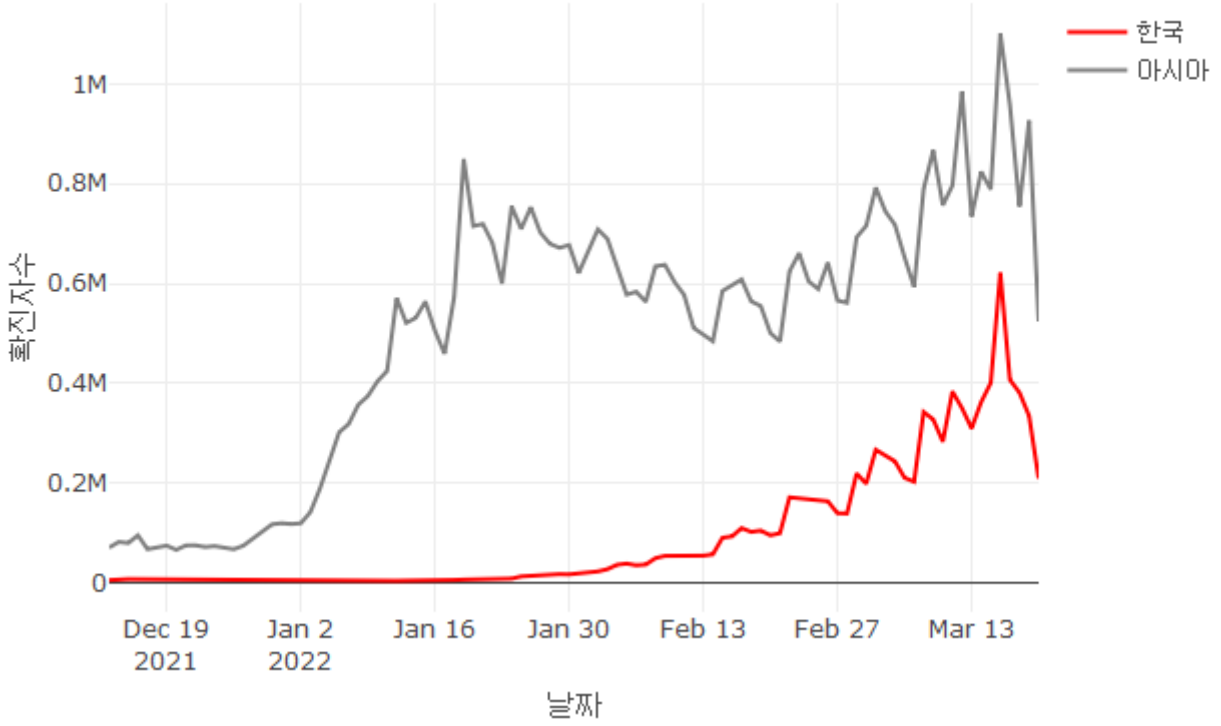
이렇게 추가적인 방법으로 시각화를 완성하면 몇 가지 장점이 있다.

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 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 설정해준다. 특히 이 작업은 여러 선 중에서 강조하고 싶은 선을 표시할 때 매우 효과적으로 사용이 가능하다.

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

```
fig <- add_trace(p_line_wide, type = 'scatter', mode = 'lines',
                 x = ~date, y = ~South_Korea, name = '한국',
                 line = list(color = 'red'))

fig |> add_trace(type = 'scatter', mode = 'lines',
                       x = ~date, y = ~Asia, name = '아시아',
                       line = list(color = 'gray')) |>
  layout(xaxis = list(title = '날짜'), yaxis = list(title = '확진자수'))
```



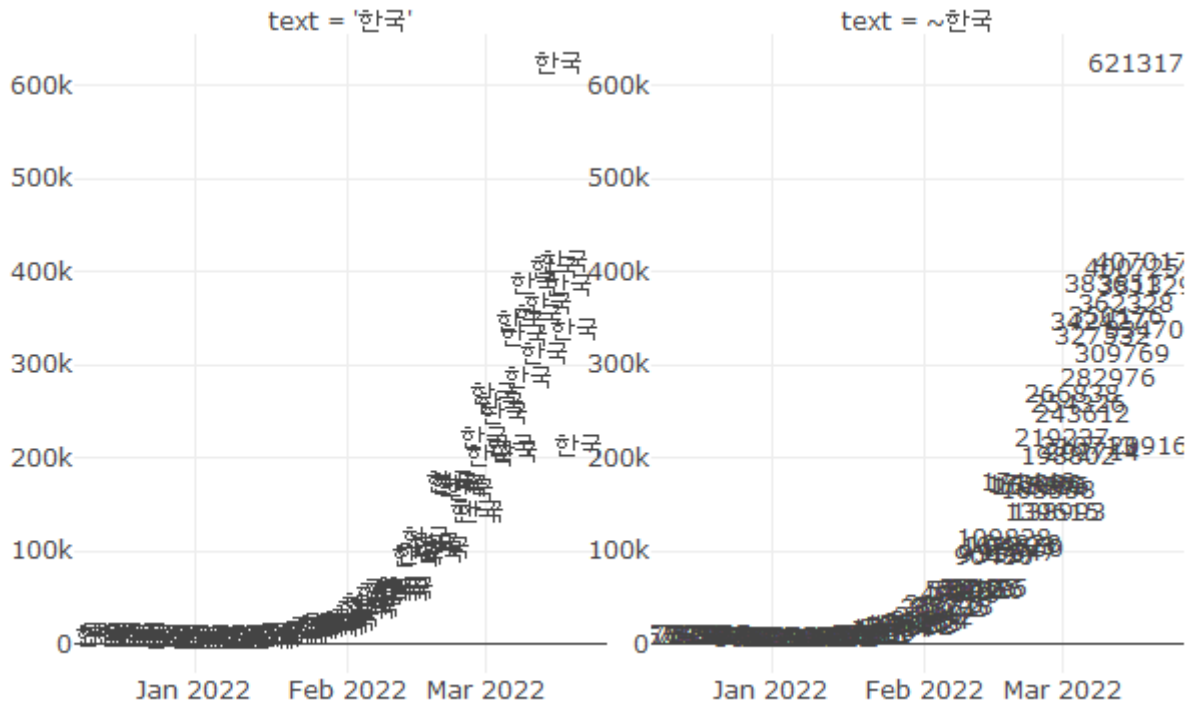
3.2.2.4. text, add_text()

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

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

`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 trace 의 `hoverinfo`에 ’text’가 설정되고 `hovertext`가 설정되지 않은 경우에는 `text`에 설정된 문자열이 호버 레이블로 사용된다.

```
add_trace(p_line_wide, type = 'scatter', mode = 'text',
          x = ~date, y = ~South_Korea, text = '한국', showlegend = FALSE)

add_trace(p_line_wide, type = 'scatter', mode = 'text',
          x = ~date, y = ~South_Korea, text = ~South_Korea, showlegend = FALSE)
```



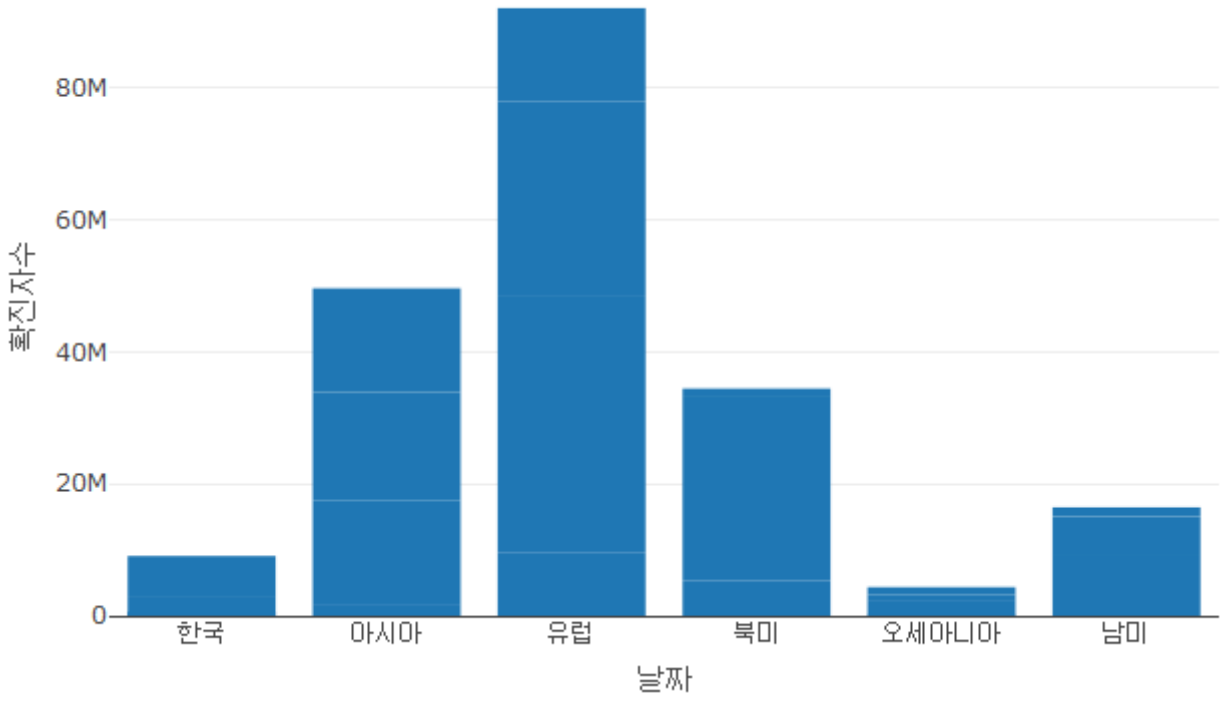
3.2.3. 막대(bar) trace : 막대 그래프

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` 매개변수에 'bar'를 설정함으로써 사용할 수 있고 `add_bar()` 함수를 사용할 수 있다.

```
add_trace(p, type = ‘bar’, …, data = NULL, inherit = TRUE)
add_bars(p, x = NULL, y = NULL, …, data = NULL, inherit = TRUE)
- p : plot_ly()로 생성한 plotly 객체
- type : trace 타입을 설정, add_markers()는 ‘scatter’로 설정 - mode : ’scatter’ trace 중 어떤 도형을 사용할지 설정, add_markers()는 ’markers’를 설정
- text : text 로 매핑할 변수를 ~로 설정
- … : 막대 trace 에 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
```

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.

```
p_bar <- covid19_df_100 |> mutate(yearmonth = tsibble::yearmonth(date)) |> group
_by(iso_code, yearmonth) |>
  summarise(new_cases = sum(new_cases)) |> plot_ly()

p_bar |>
  add_trace(type = 'bar', x = ~iso_code, y = ~new_cases) |>
  layout(xaxis = list(title = '날짜',
                      ticktext = c('한국', '아시아', '유럽', '북미', '오세아니아',
'남미'),
                      tickvals = c('KOR', 'OWID_ASI', 'OWID_EUR', 'OWID_NAM', 'O
WID_OCE', 'OWID_SAM')),
         yaxis = list(title = '확진자수'))
```

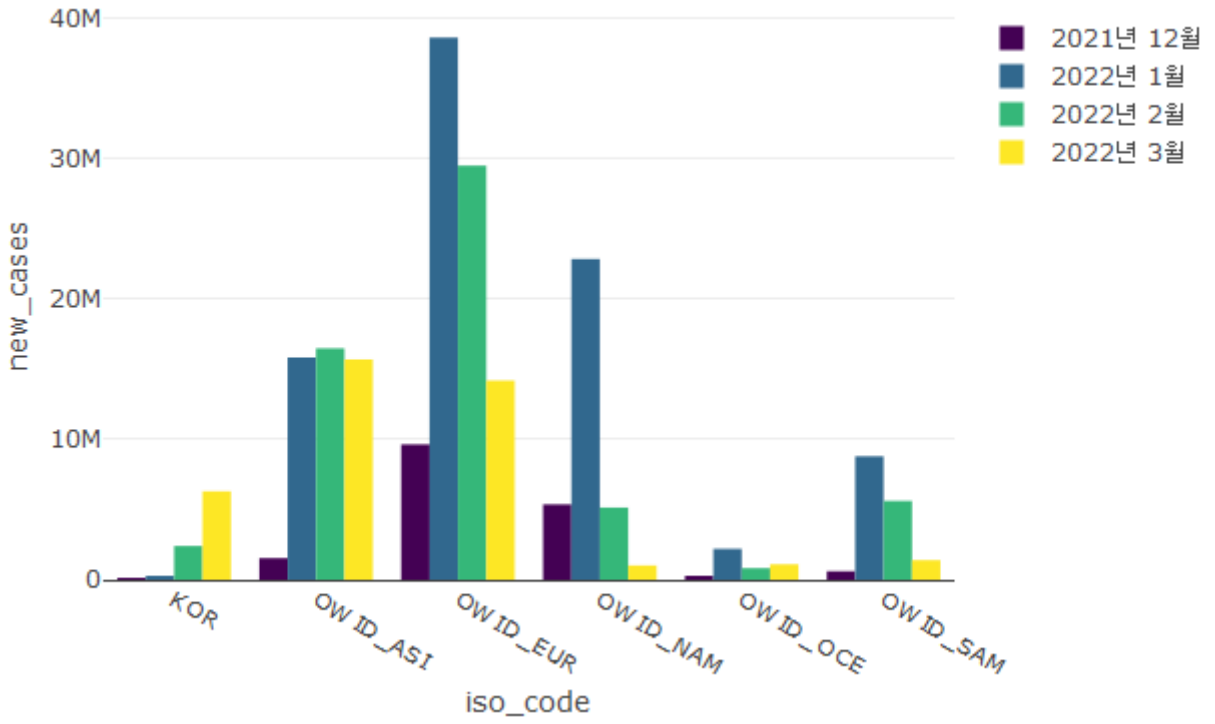


3.2.3.1. barmode

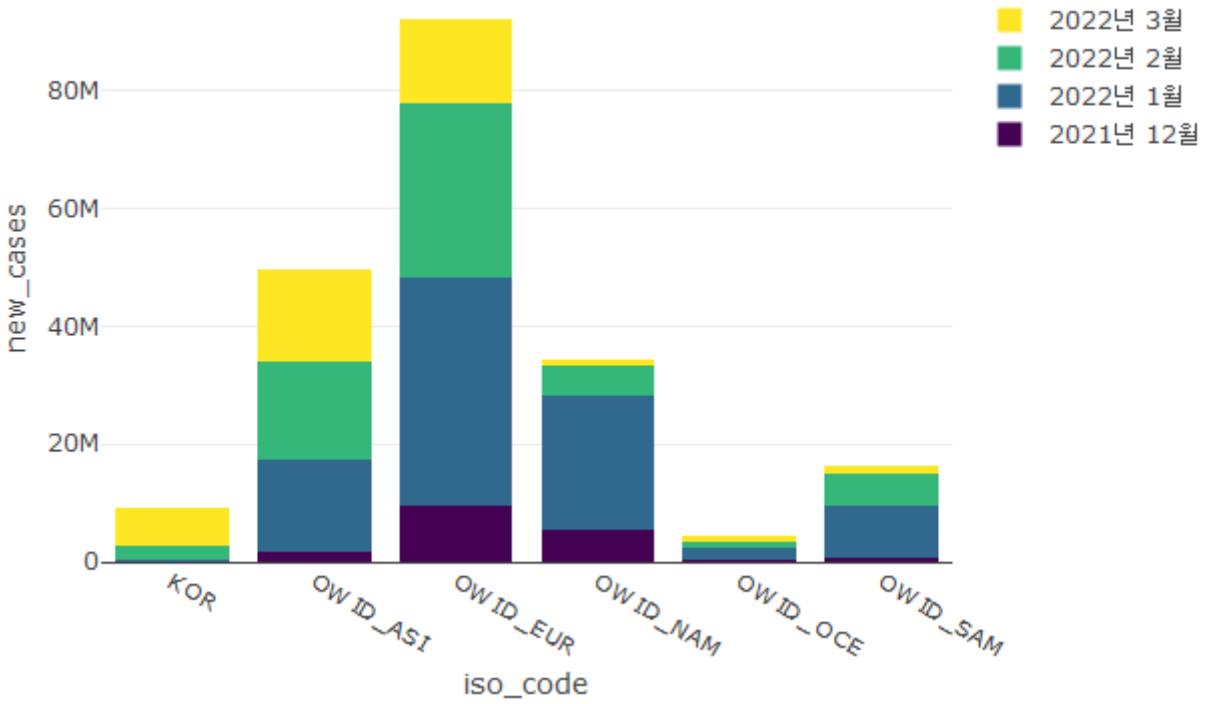
막대 그래프는 일반적으로 세 가지 형태로 생성된다. `ggplot2` 에서는 dodge, stack, fill 로 불리며 position 을 셋중에 하나로 설정함으로써 자동적으로 그려지지만 `plotly` 에서 막대 그래프를 생성하는 `add_trace()`나 `add_bars()`로는 이 세 가지 막대 그래프를 구분하여 생성할

수 없다. 이를 구분하여 생성하기 위한 방법은 차후 설명할 `layout()`의 `barmode` type 을 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 방식이으로 `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.

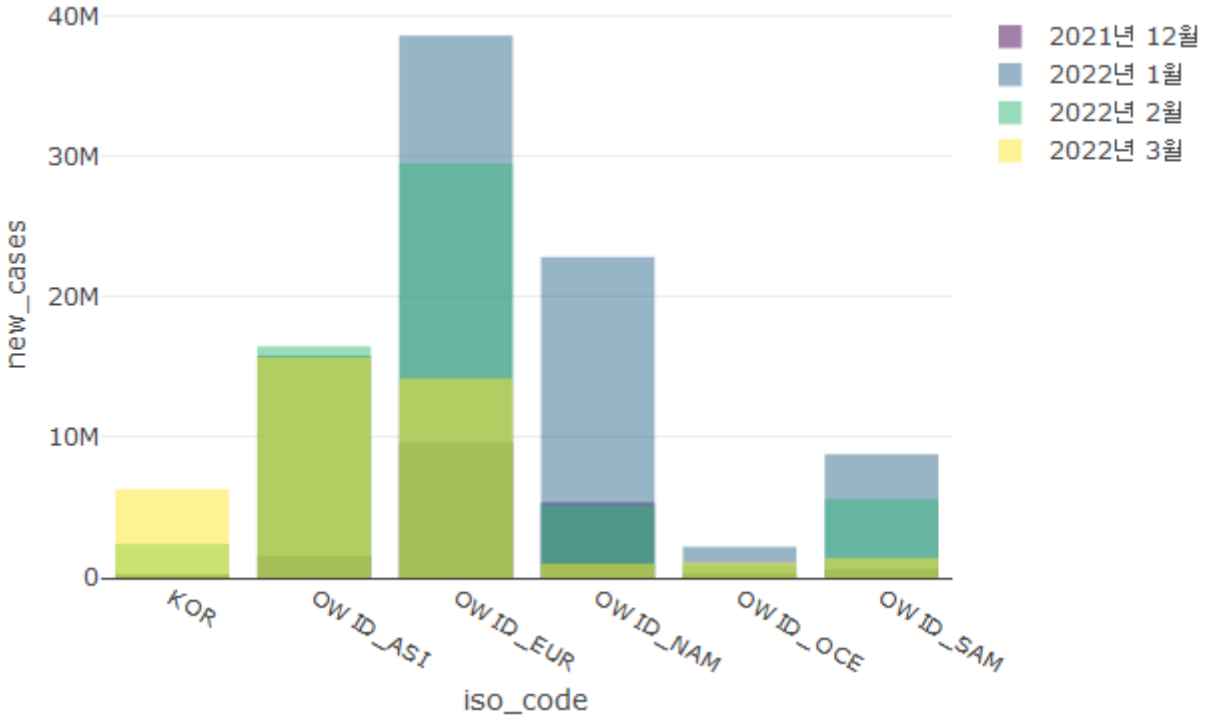
```
p_bar_mode <- covid19_df_100 |> mutate(yearmonth = lubridate::floor_date(date, "
month")) |> group_by(iso_code, yearmonth) |>
  summarise(new_cases = sum(new_cases)) |> plot_ly()

p_bar_mode |>
  add_trace(type = 'bar', x = ~iso_code, y = ~new_cases,
            color = ~as.factor(yearmonth), colors = 'viridis',
            name = ~paste0(lubridate::year(yearmonth), '년 ', lubridate::month(ye
armonth, label = TRUE, abbr = FALSE))) |>
  layout(barmode = 'group')
```



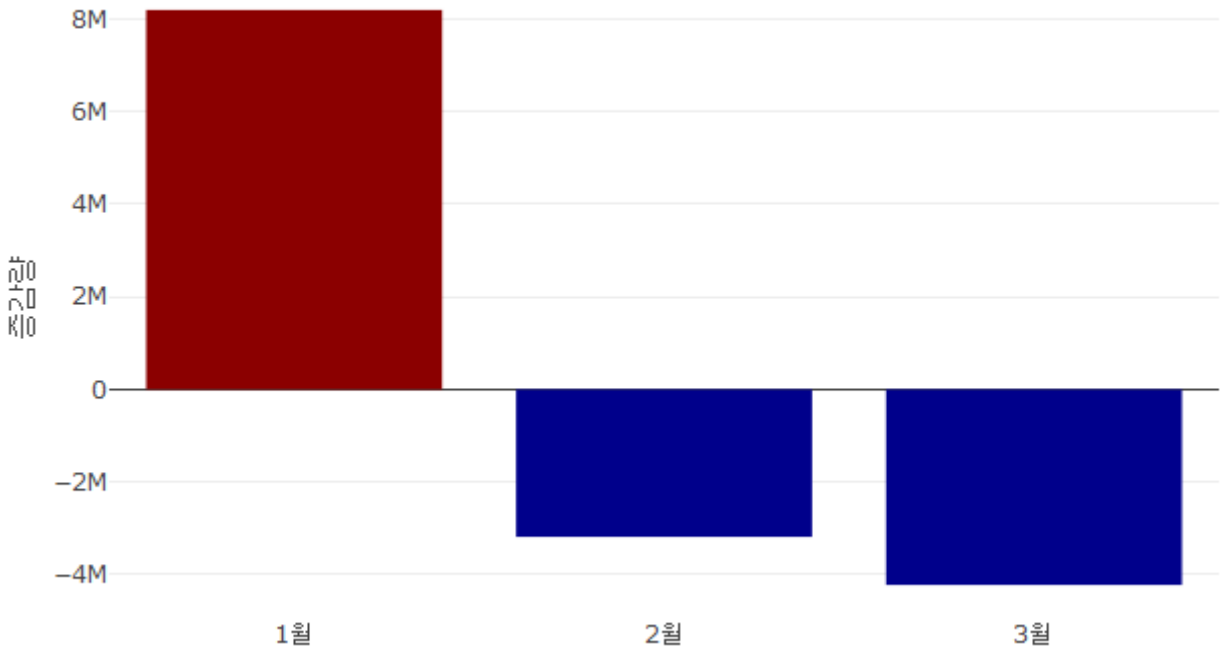
```
p_bar_mode |>
  add_trace(type = 'bar', x = ~iso_code, y = ~new_cases,
            color = ~as.factor(yearmonth), colors = 'viridis',
            name = ~paste0(lubridate::year(yearmonth), '년 ', lubridate::month(ye
armonth, label = TRUE, abbr = FALSE))) |>
  layout(barmode = 'stack')
```




```
p_bar_mode |>
  add_trace(type = 'bar', x = ~iso_code, y = ~new_cases,
            color = ~as.factor(yearmonth), colors = 'viridis', opacity = 0.5,
            name = ~paste0(lubridate::year(yearmonth), '년 ', lubridate::month(ye
armonth, label = TRUE, abbr = FALSE))) |>
  layout(barmode = 'overlay')
```




```
diff_sam <- covid19_df_100 |> mutate(yearmonth = lubridate::floor_date(date, "mo
nth")) |> group_by(iso_code, yearmonth) |>
  summarise(new_cases = sum(new_cases)) |>
  pivot_wider(names_from = iso_code, values_from = new_cases) |>
  select(OWID_SAM) |>
  pull()

plot_ly() |> add_trace(type = 'bar', x = 1:3, y = diff(diff_sam),
                       color = I(ifelse(diff(diff_sam) > 0, "darkred", "darkblue
"))
                       ) |>
  layout(barmode = 'relative',
         xaxis = list(ticktext = c('1 월', '2 월', '3 월'),
                      tickvals = c(1, 2, 3)),
         yaxis = list(title = '증감량'))
```



앞서 스캐터 trace 의 선 그래프를 그릴때 넓은 형태의 데이터프레임을 사용하여 trace 를 하나 하나 추가하는 형태의 시각화를 그려보았다. 막대 trace 도 동일하게 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.

```
p_bar_mode1 <- covid19_df_100 |>
  mutate(yearmonth = lubridate::floor_date(date, "month")) |>
  group_by(iso_code, yearmonth) |>
  summarise(new_cases = sum(new_cases)) |>
  pivot_wider(names_from = yearmonth, values_from = new_cases) |>
  plot_ly()

fig <- p_bar_mode1 |>
  add_trace(type = 'bar', x = ~iso_code, y = ~`2021-12-01`, color = I('gray30'),

            name = '2021 월 12 월')

fig <- fig |>
  add_trace(type = 'bar', x = ~iso_code, y = ~`2022-01-01`, color = I('gray40'),

            name = '2022 월 1 월')
```

```
fig <- fig |>
  add_trace(type = 'bar', x = ~iso_code, y = ~`2022-02-01`, color = I('gray50'),

            name = '2022 월 2 월')

fig <- fig |>
  add_trace(type = 'bar', x = ~iso_code, y = ~`2022-03-01`, color = I('darkred
'),
            name = '2022 월 3 월')

fig |> layout(barmode = 'stack',
              xaxis = list(title = '지역',
                           ticktext = c('한국', '아시아', '유럽', '북미', '오세아니
아', '남미'),
                           tickvals = c('KOR', 'OWID_ASI', 'OWID_EUR', 'OWID_NAM
', 'OWID_OCE', 'OWID_SAM')),
              yaxis = list(title = '확진자수')
              )
```

3.2.3.2. text, textposition, texttemplate, textfont

막대 그래프는 일반적으로 데이터 간의 비교를 위해 사용된다. 따라서 막대 그래프에 표시된 막대로는 데이터의 정확한 값을 알아보기는 어렵다. 이를 위해 막대그래프에 데이터 값을 표기하는 경우가 많다. `plotly` 의 막대 trace 에 데이터 값을 표현하기 위해서는 `text` type 을 사용하면 간단히 표시할 수 있다. 표시되기를 원하는 변수를 `text` 에 매핑함으로써 각각의 막대에 데이터 값을 표시할 수 있으며 `textposition` 과 `texttemplate` 를 사용하여 표시되는 값의 세부 설정을 변경할 수 있다.

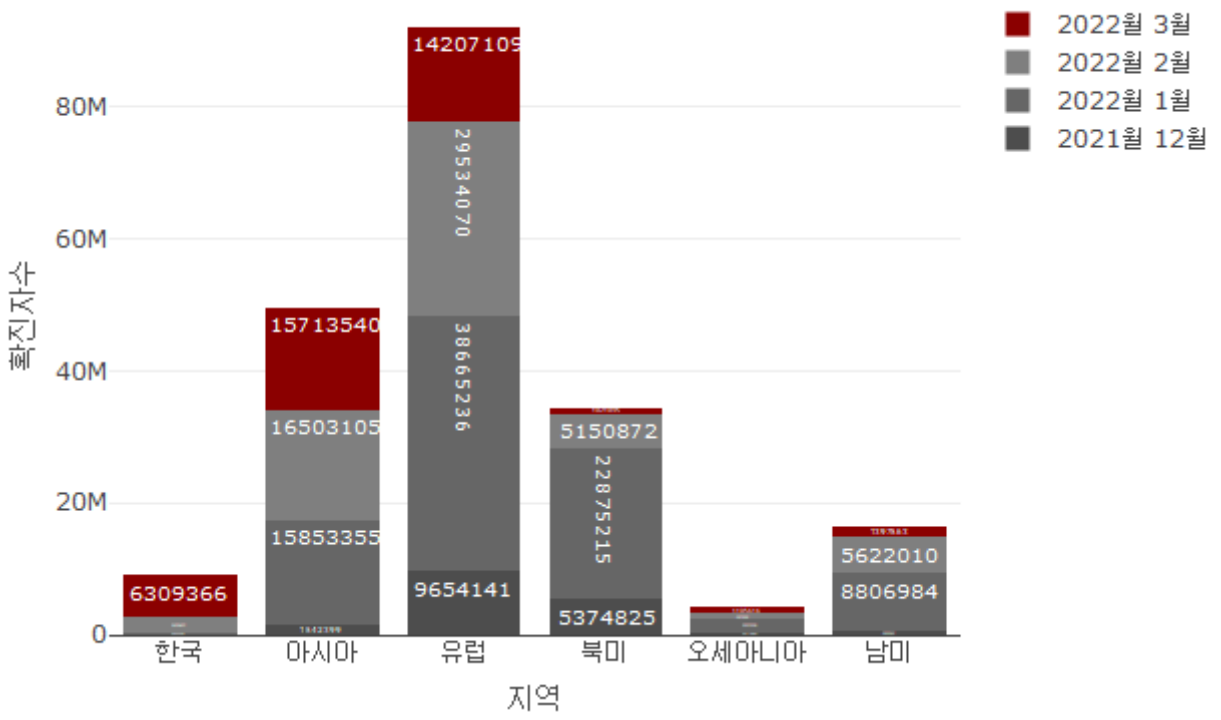
`text` type 은 막대에 표시되어야 하는 문자열을 매핑하거나 설정하는 type 이다. `text` 에 단일 문자열을 설정하면 모든 막대에 설정된 문자열이 표시되고 문자열 벡터가 설정되면 문자열 벡터의 순서에 따라 막대 위에 문자열이 표시된다.

`textposition` 는 `text` 의 위치를 설정하는 type 이다. `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.

```
p_bar_mode1 <- covid19_df_100 |>
  mutate(yearmonth = lubridate::floor_date(date, "month")) |>
  group_by(iso_code, yearmonth) |>
  summarise(new_cases = sum(new_cases)) |>
  pivot_wider(names_from = yearmonth, values_from = new_cases) |>
  plot_ly()

fig <- p_bar_mode1 |>
  add_trace(type = 'bar', x = ~iso_code, y = ~`2021-12-01`, color = I('gray30'),

            name = '2021 월 12 월', text = ~`2021-12-01`,
            textposition = 'inside', textfont = list(color = 'white', size = 10)
            )

fig <- fig |>
  add_trace(type = 'bar', x = ~iso_code, y = ~`2022-01-01`, color = I('gray40'),

            name = '2022 월 1 월', text = ~`2022-01-01`,
            textposition = 'inside', textfont = list(color = 'white', size = 10)
            )

fig <- fig |>
  add_trace(type = 'bar', x = ~iso_code, y = ~`2022-02-01`, color = I('gray50'),

            name = '2022 월 2 월', text = ~`2022-02-01`,
            textposition = 'inside', textfont = list(color = 'white', size = 10)
            )

fig <- fig |>
  add_trace(type = 'bar', x = ~iso_code, y = ~`2022-03-01`, color = I('darkred
'),
            name = '2022 월 3 월', text = ~`2022-03-01`,
            textposition = 'inside', textfont = list(color = 'white', size = 10)
            )

fig |> layout(barmode = 'stack',
              xaxis = list(title = '지역',
                           ticktext = c('한국', '아시아', '유럽', '북미', '오세아니
아', '남미'),
                           tickvals = c('KOR', 'OWID_ASI', 'OWID_EUR', 'OWID_NAM
', 'OWID_OCE', 'OWID_SAM')),
```

```
            yaxis = list(title = '확진자수')
            )
```

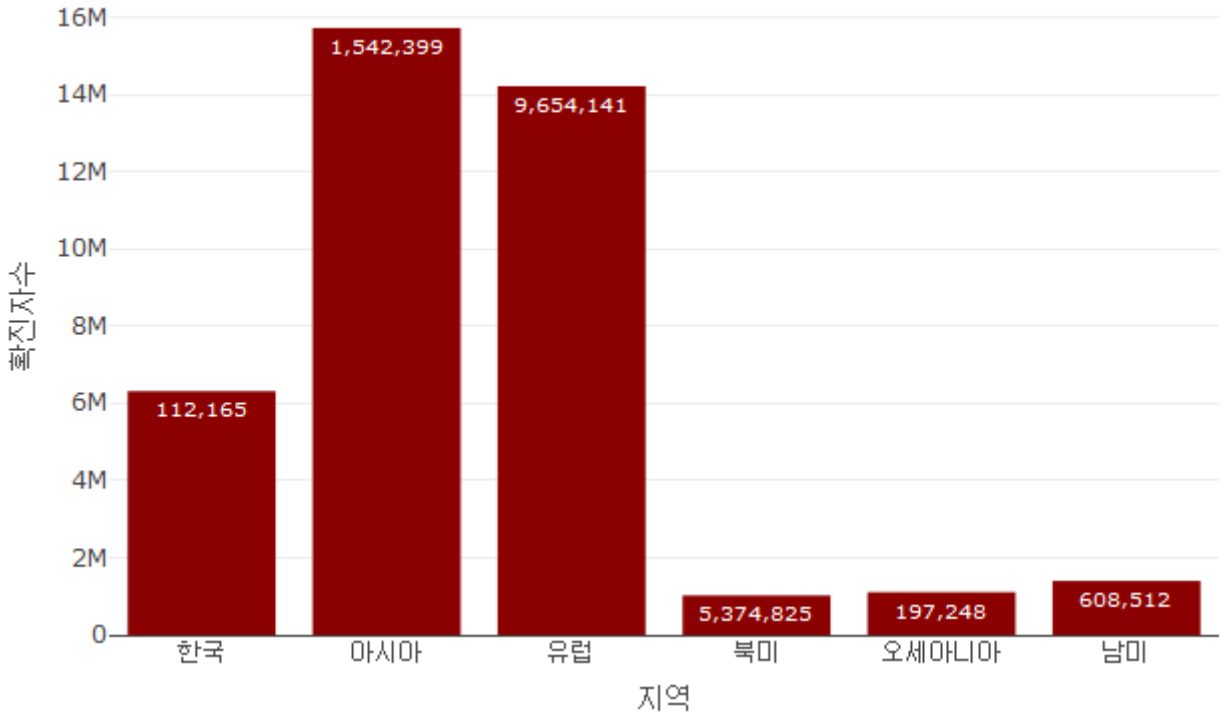


texttemplate 은 텍스트가 표시되는 형태를 설정하는 type 이다. texttemplate 는 hovertemplate 의 정의와 같이 사용되는데 '%{변수:변수포맷}'의 형태로 사용된다. 변수 포맷에서 사용하는 포맷은 자바 스크립트의 d3 format[6]을 사용한다. 앞의 예에서 일반대학의 입학생수의 포맷에 천단위 콤마를 넣는다면'%{text:,}'로 설정하고 소수점 아래 두째 자리까지 표기한다면'%{text:2f}'로 설정한다.

```
p_bar_mode1 |>
  add_trace(type = 'bar', x = ~iso_code, y = ~`2022-03-01`, color = I('darkred
'),
            name = '2021 월 12 월', text = ~`2021-12-01`,
            textposition = 'inside', textfont = list(color = 'white', size = 1
0),
            texttemplate = '%{text:,}'
            ) |>
  layout(xaxis = list(title = '지역',
                      ticktext = c('한국', '아시아', '유럽', '북미', '오세아니아',
```

[6] https://github.com/d3/d3-format/tree/v1.4.5#d3-format

```
'남미'),
                      tickvals = c('KOR', 'OWID_ASI', 'OWID_EUR', 'OWID_NAM', 'O
WID_OCE', 'OWID_SAM')
                      ),
         yaxis = list(title= '확진자수'))
```

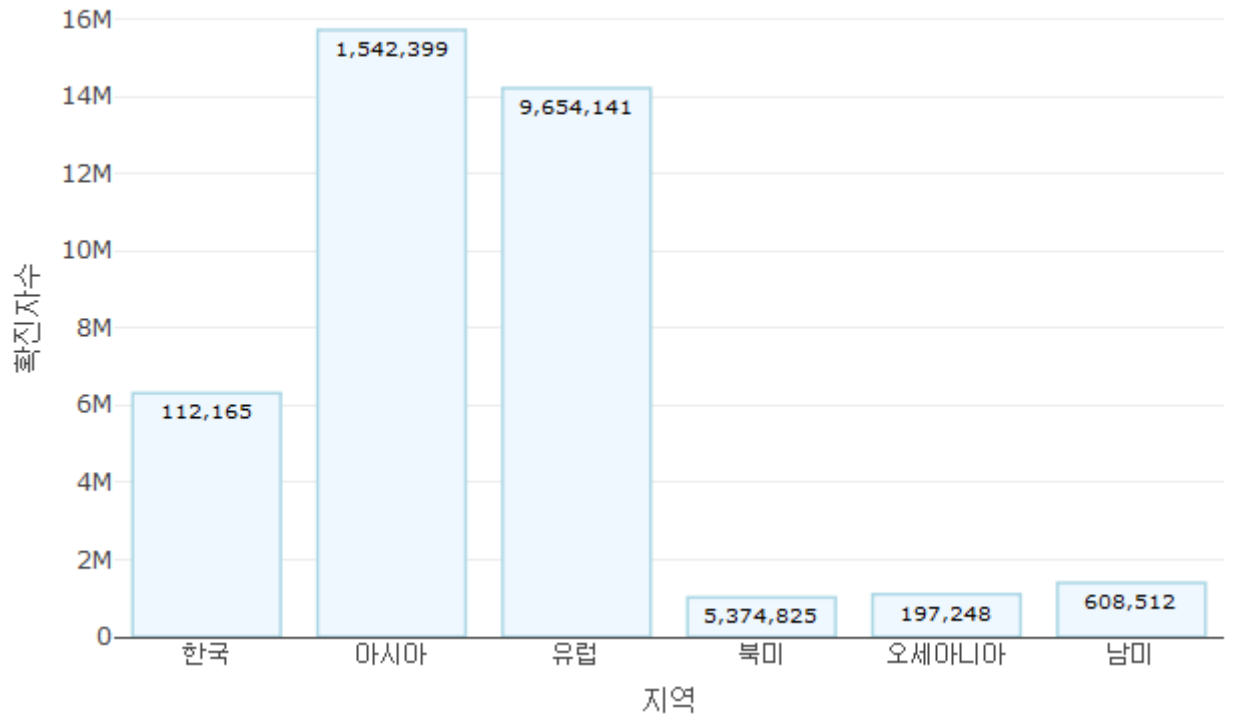


3.2.3.3. marker

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 설정 가능한 type 은 막대의 내부 색, 외부 외곽선 등의 세부 설정을 위한 type 을 지정할 수 있다.

```
p_bar_mode1 |>
  add_trace(type = 'bar', x = ~iso_code, y = ~`2022-03-01`, color = I('darkred
'),
            name = '2021 월 12 월', text = ~`2021-12-01`,
            textposition = 'inside', textfont = list(color = 'black', size = 1
0),
            texttemplate = '%{text:,}',
            marker = list(color = 'aliceblue',
                          line = list(color = 'lightblue', width = 1.5)
                          )
            ) |>
```

```
  layout(xaxis = list(title = '지역',
                      ticktext = c('한국', '아시아', '유럽', '북미', '오세아니아',
'남미'),
                      tickvals = c('KOR', 'OWID_ASI', 'OWID_EUR', 'OWID_NAM', 'O
WID_OCE', 'OWID_SAM')
                      ),
         yaxis = list(title= '확진자수'))
```



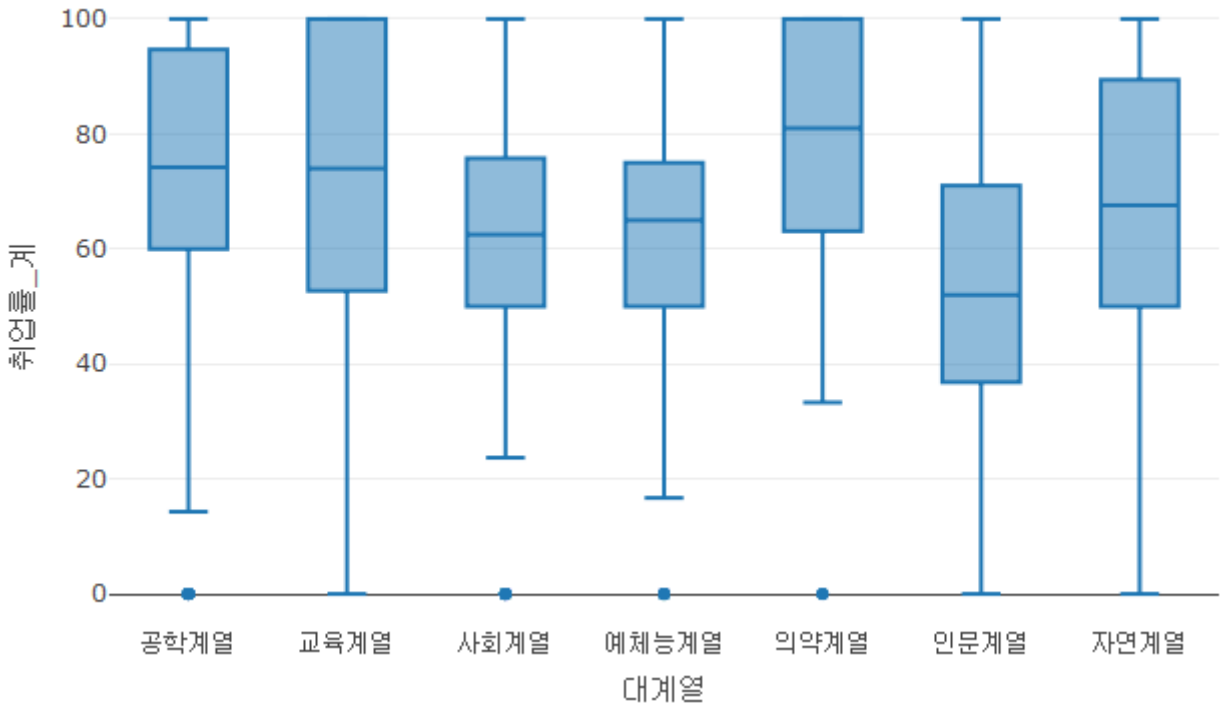
3.2.4. 박스(Box) trace : 박스 플롯

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
```

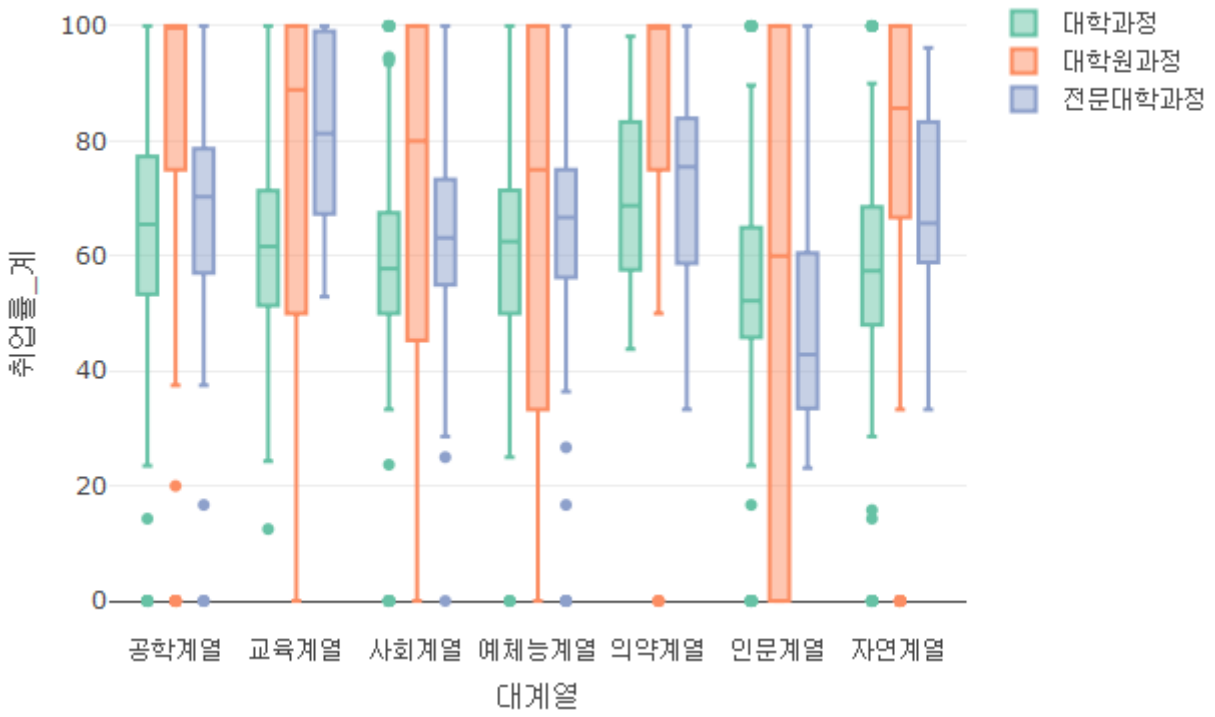
- data : 시각화할 데이터프레임
- inherit : plot_ly()에 설정된 속성 type 을 상속할지를 결정하는 논리값
- x : X 축에 매핑할 변수를 ~로 설정
- y : Y 축에 매핑할 변수를 ~로 설정

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

p_box |> add_trace(type = 'box', x = ~대계열, y = ~취업률_계)
```



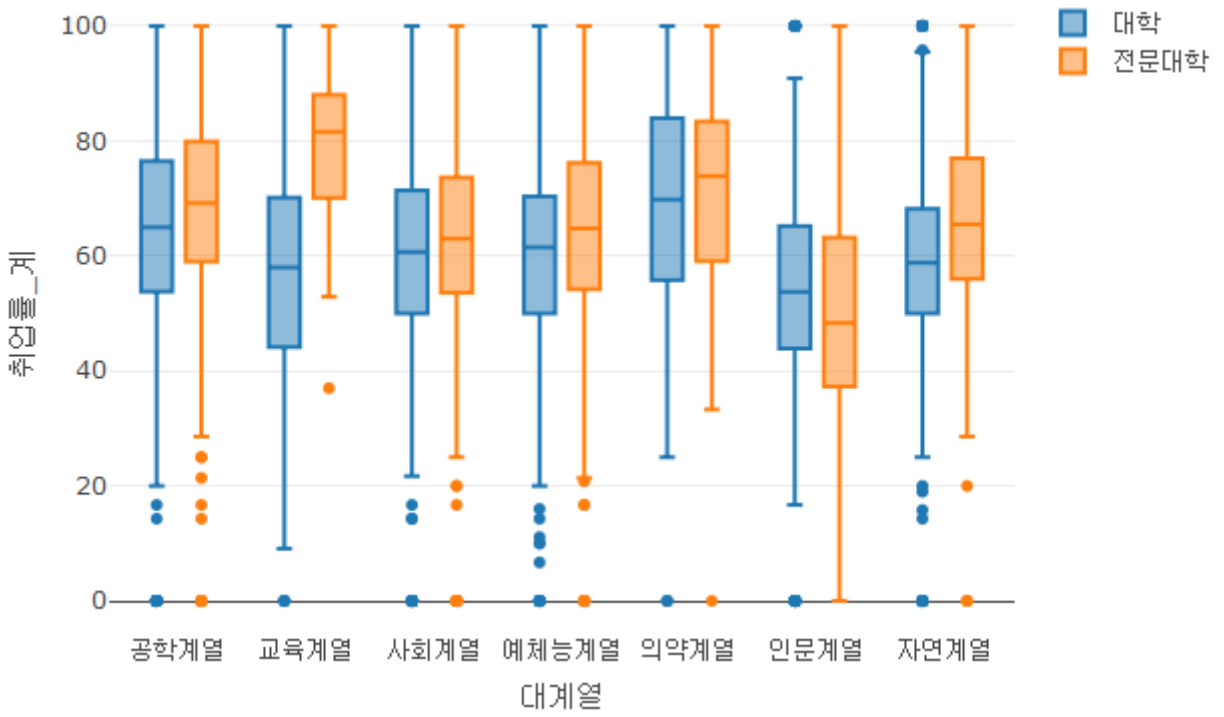
각 데이터별로 그룹화하여 분할된 바이올린 trace 를 생성하기 위해서는 그룹화할 변수를 `color` 나 `linetype` 등의 type 에 매핑 시키거나 각각의 바이올린 trace 를 추가하고 `layout()`의 `boxmode` 를 'group'으로 설정함으로써 생성할 수 있다. 다음의 코드는 `color` 로 과정구분을 매핑하여 각각의 과정별 박스 플롯을 생성하는 코드이다.

```
p_box_group <- p_box |> add_trace(type = 'box', x = ~대계열, y = ~취업률_계,
                                  color = ~과정구분)

p_box_group %>% layout(boxmode = "group")
```



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

```
p_box_add <- p_box |>
  add_trace(data = df_취업통계 |> filter(졸업자_계 < 500, 과정구분 == '대학과정'),
            type = 'box', x = ~대계열, y = ~취업률_계, name = '대학')

p_box_add <- p_box_add |>
  add_trace(data = df_취업통계 |> filter(졸업자_계 < 500, 과정구분 == '전문대학과정
'),
            type = 'box', x = ~대계열, y = ~취업률_계, name = '전문대학')

p_box_add %>% layout(boxmode = "group")
```

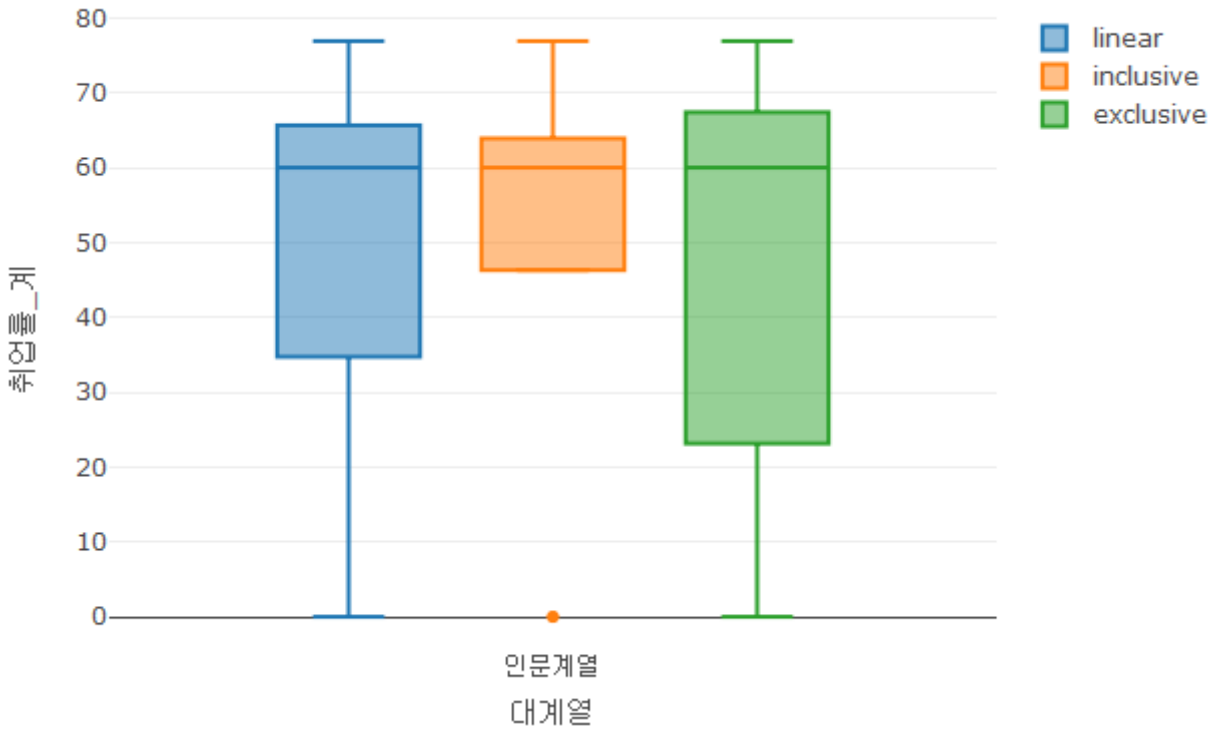


3.2.4.1. quartilemethod

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 두가지 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 |> filter(대계열 == '인문계열') |> head
(9) |> plot_ly()

p_box_quartilemethod <- p_box_quartilemethod |>
  add_trace(type = 'box', x = ~대계열, y = ~취업률_계,
            quartilemethod = 'linear', name = 'linear')

p_box_quartilemethod <- p_box_quartilemethod |>
  add_trace(type = 'box', x = ~대계열, y = ~취업률_계,
```

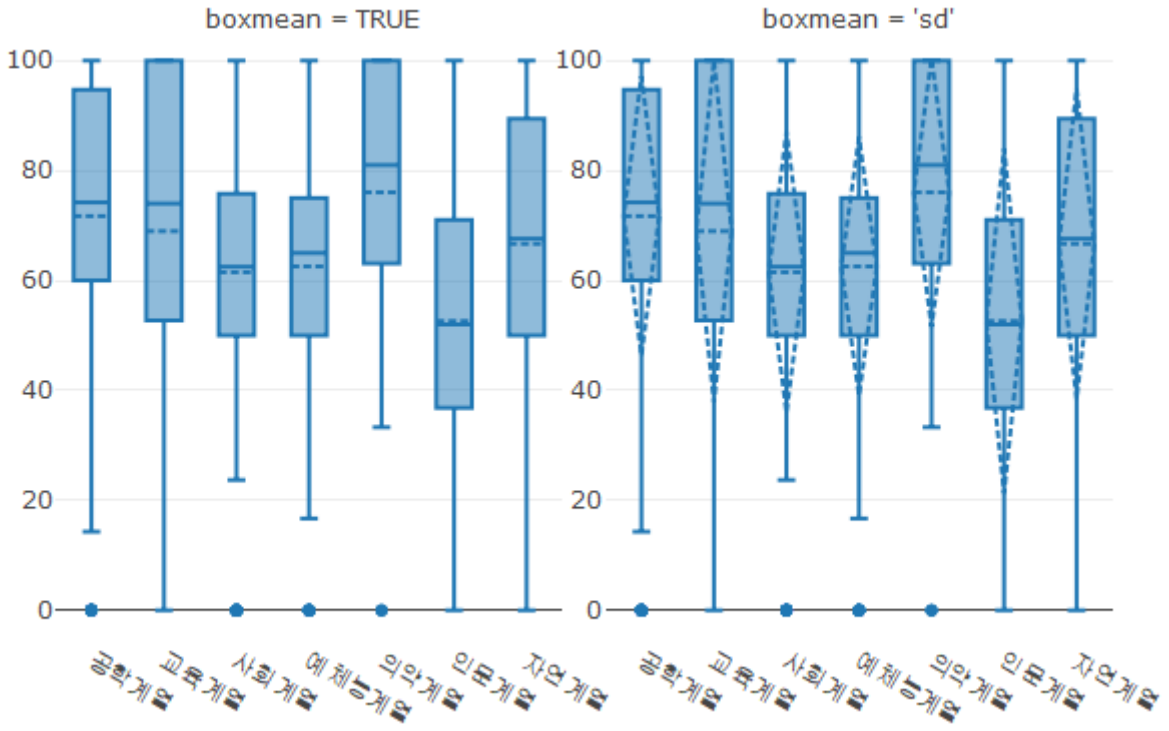
```
            quartilemethod = 'inclusive', name = 'inclusive')

p_box_quartilemethod <- p_box_quartilemethod |>
  add_trace(type = 'box', x = ~대계열, y = ~취업률_계,
            quartilemethod = 'exclusive', name = 'exclusive')

p_box_quartilemethod %>% layout(boxmode = "group")
```



3.2.4.2. boxmean, boxpoints

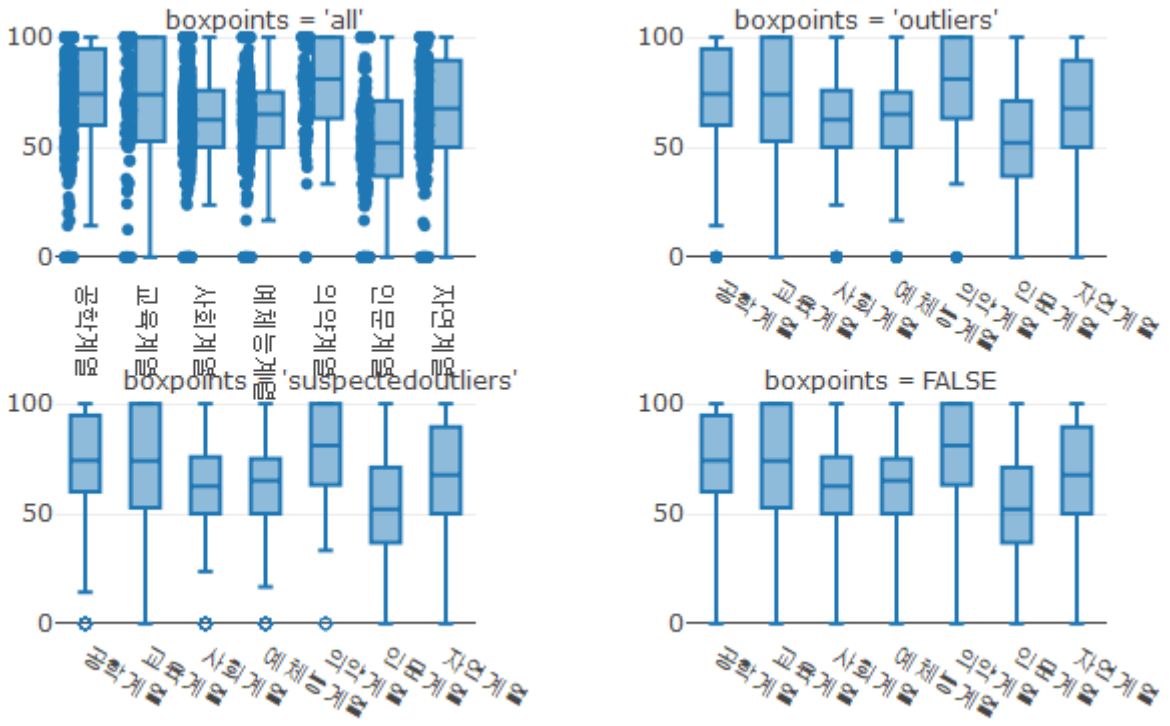
박스 trace 에서 제공하는 요약 통계중에 가장 많이 사용되지만 제공되지 않는 요약통계가 바로 평균(mean)이다. `ggplot2` 에서는 평균을 표현하기 위해 다소 어려운 과정을 거쳐야 했지만 `plotly` 에서는 `boxmean` type 의 설정만으로 간단하게 평균값을 표현할 수 있다. `boxmean` 은 TRUE/FALSE 의 논리값에 표준편차가 추가로 표시되는 'sd'를 제공한다.

```
p_box |> add_trace(type = 'box', x = ~대계열, y = ~취업률_계, boxmean = TRUE)

p_box |> add_trace(type = 'box', x = ~대계열, y = ~취업률_계, boxmean = 'sd')
```



또 박스 trace 에서 유용하게 사용되는 type 이 `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.

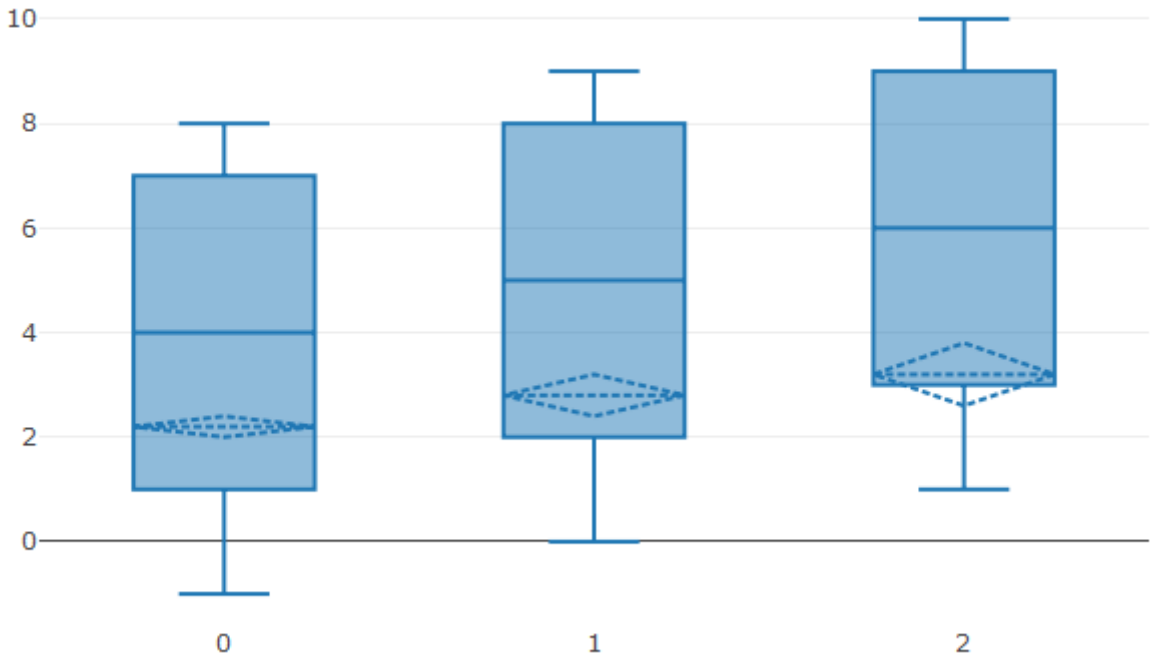
```
p_box |> add_trace(type = 'box', x = ~대계열, y = ~취업률_계,
                   boxpoints = 'all')

p_box |> add_trace(type = 'box', x = ~대계열, y = ~취업률_계,
                   boxpoints = 'outliers')

p_box |> add_trace(type = 'box', x = ~대계열, y = ~취업률_계,
                   boxpoints = 'suspectedoutliers')

p_box |> add_trace(type = 'box', x = ~대계열, y = ~취업률_계,
                   boxpoints = FALSE)
```



3.2.4.3. mean, median, q1, q3, upperfence, lowerfence

박스 trace 는 박스 플롯의 생성시 자동적으로 여러가지 요약 통계를 산출하지만 경우에 따라 이 값들을 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 type 은 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.

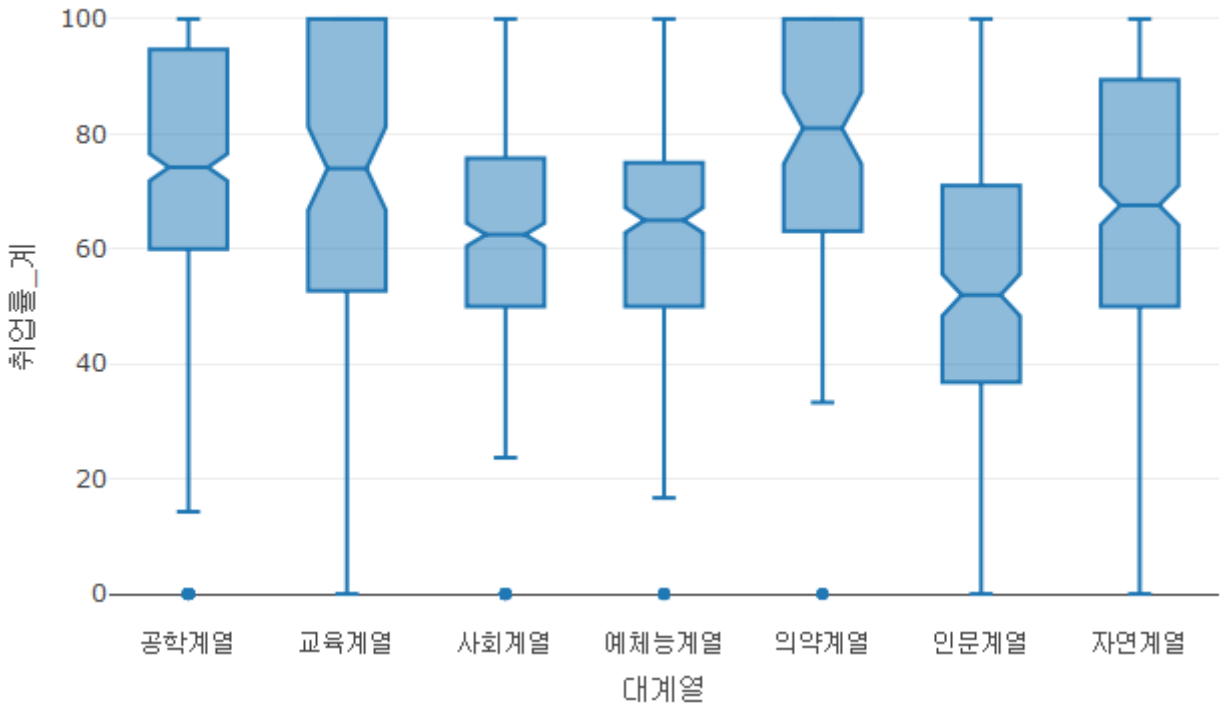
```
plot_ly() |>
  add_trace(type= 'box', q1=c(1, 2, 3), median=c(4, 5, 6),
            q3=c(7, 8, 9 ), lowerfence=c(-1, 0, 1),
            upperfence=c(8, 9, 10), mean=c(2.2, 2.8, 3.2 ),
            sd=c(0.2, 0.4, 0.6))
```



3.2.4.4. notched, notchwidth, notchspan, whiskerwidth

2017 년 iPhoneX 가 처음 소개되면서 유명세를 탄 디자인 스타일이 notch 스타일이다. 이 notch 스타일은 평면이나 직선의 일부에 삼각형이나 사각형의 홈을 내는 디자인 스타일을 말한다. 박스 trace 에도 이 notch 스타일을 적용할 수 있는데 중앙값이 표시되는 부분에 삼각형 홈을 내는 type 이 `notched` 이다. `notched` 는 TRUE/FALSE 의 두 가지 중에 하나의 값을 가지는데 TRUE 일 경우 notch 디자인으로 설정된다. `notched` 가 TRUE 로 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 0 부터 0.5 까지의 값을 가질수 있다. 또 `notchspan` 은 수직 방향으로의 홈의 크기를 설정하는 type 이다. 막대 외부에 표시되는 수염의 끝에 있는 가로선의 크기를 설정하는 type 이 `whiskerwidth` 이다. 이 type 도 `notchwidth` 와 같이 0 부터 0.5 사이의 값을 가진다.

```
p_box |> add_trace(type = 'box', x = ~대계열, y = ~취업률_계,
                   notched = TRUE, notchewidth = 0.2, notchespan = 0.5,
                   whiskerwidth = 0.3)
```



3.2.4.5. marker, jitter

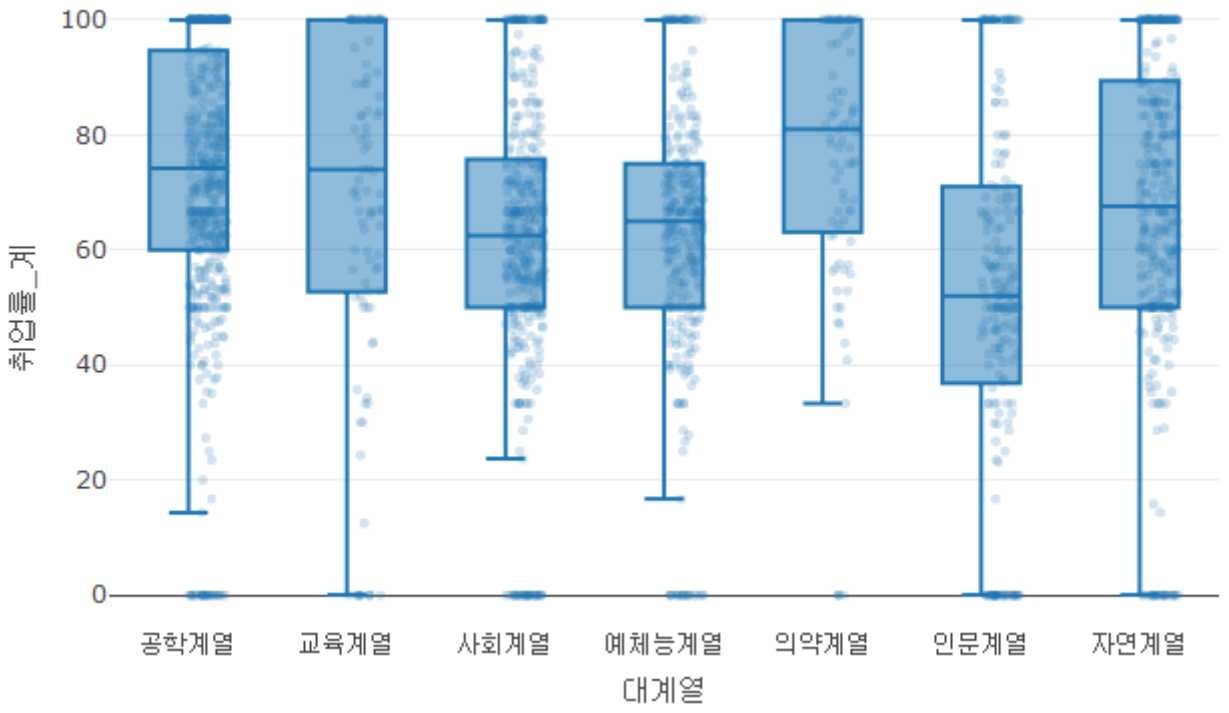
막대 trace 를 사용하면 전체적인 데이터의 분포를 막대의 선으로 살펴볼 수 있지만 이 구간에서 데이터가 집중된 위치나 데이터 발생 빈도가 낮은 위치를 알아볼 수는 없다. 이를 알아보기 위해서는 실제 데이터의 위치에 점을 찍어 데이터의 분포를 직접 보는 방법이 사용되는데 이렇게 막대 trace 에서 데이터의 실제 분포를 표시하는 type 이 `marker` 와 `jitter` 이다.

`marker` 는 박스 trace 에서 사용되는 점들의 세부 설정을 위한 type 이다. 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 type 으로 `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.

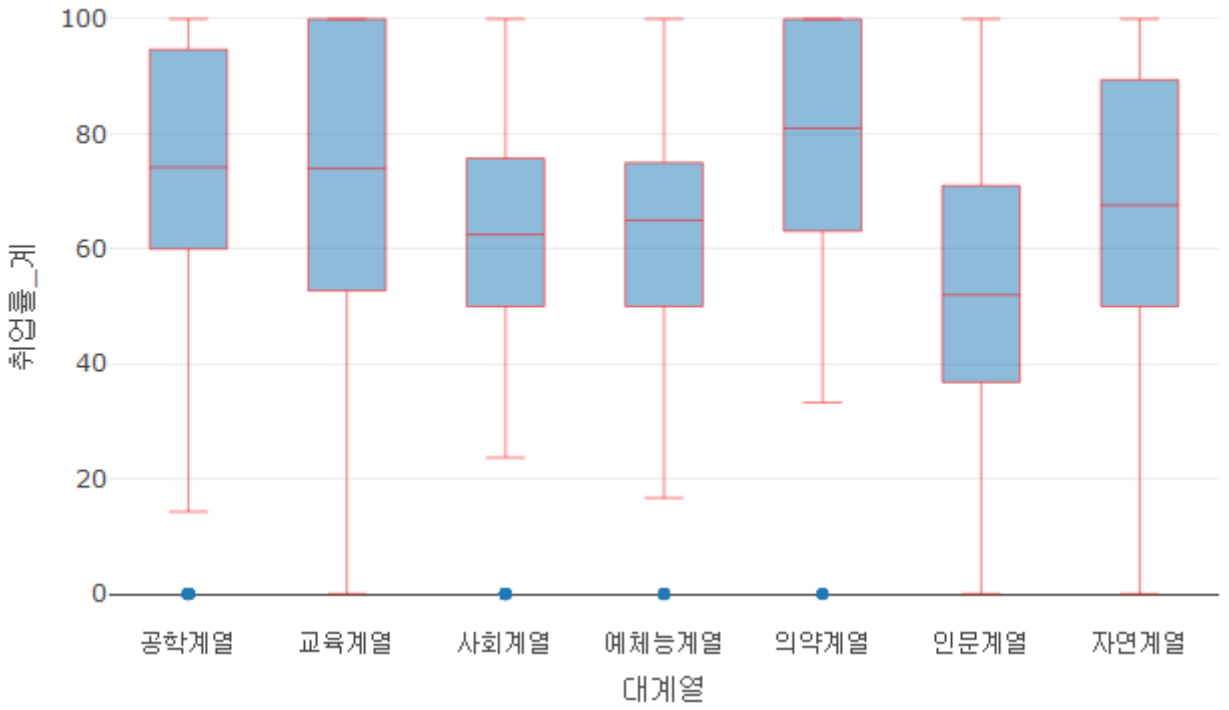
```
p_box |>
  add_trace(type = 'box', x = ~대계열, y = ~취업률_계,
            marker = list(opacity = 0.2, size = 5, color = 'aquablue'),
            boxpoints = "all", jitter = 0.5, pointpos = 0.5)
```



3.2.4.6. line

`line` 은 막대 trace 에서 외곽선을 설정하는 type 이다. 이 type 에서는 세부 선의 색과 두께를 설정할 수 있다.

```
p_box |>
  add_trace(type = 'box', x = ~대계열, y = ~취업률_계,
            line = list(width = 0.5, color = 'red'))
```



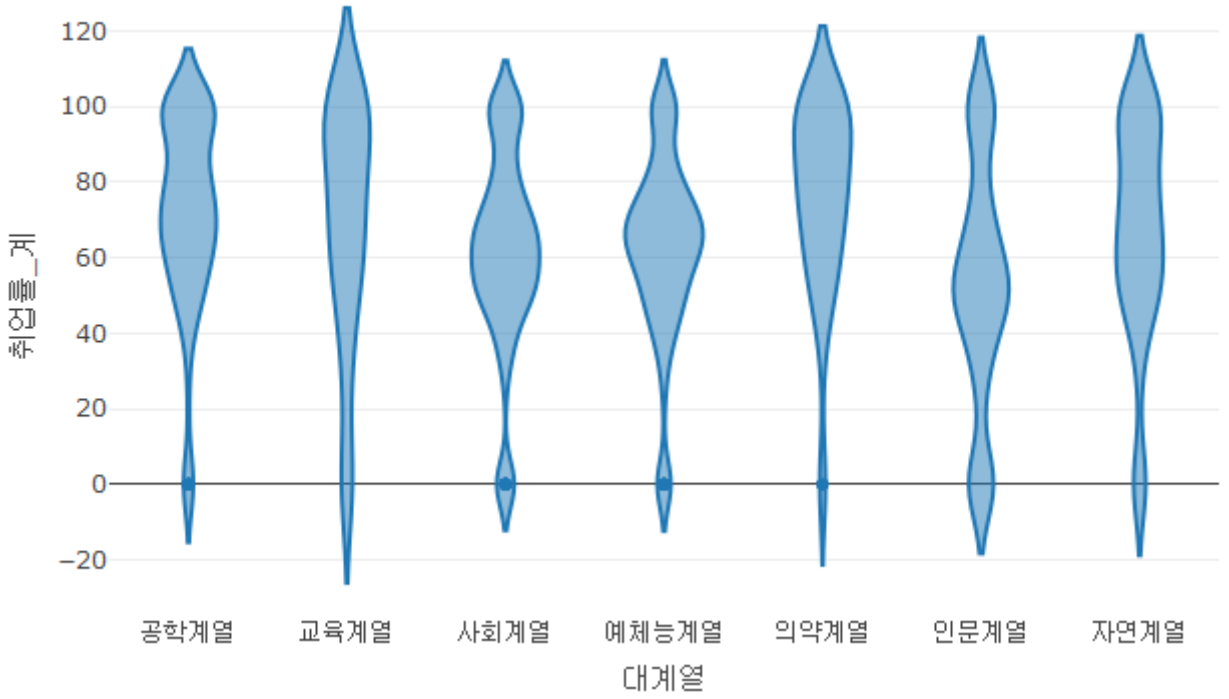
3.2.5. 바이올린(Violin) trace : 바이올린 플롯

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.

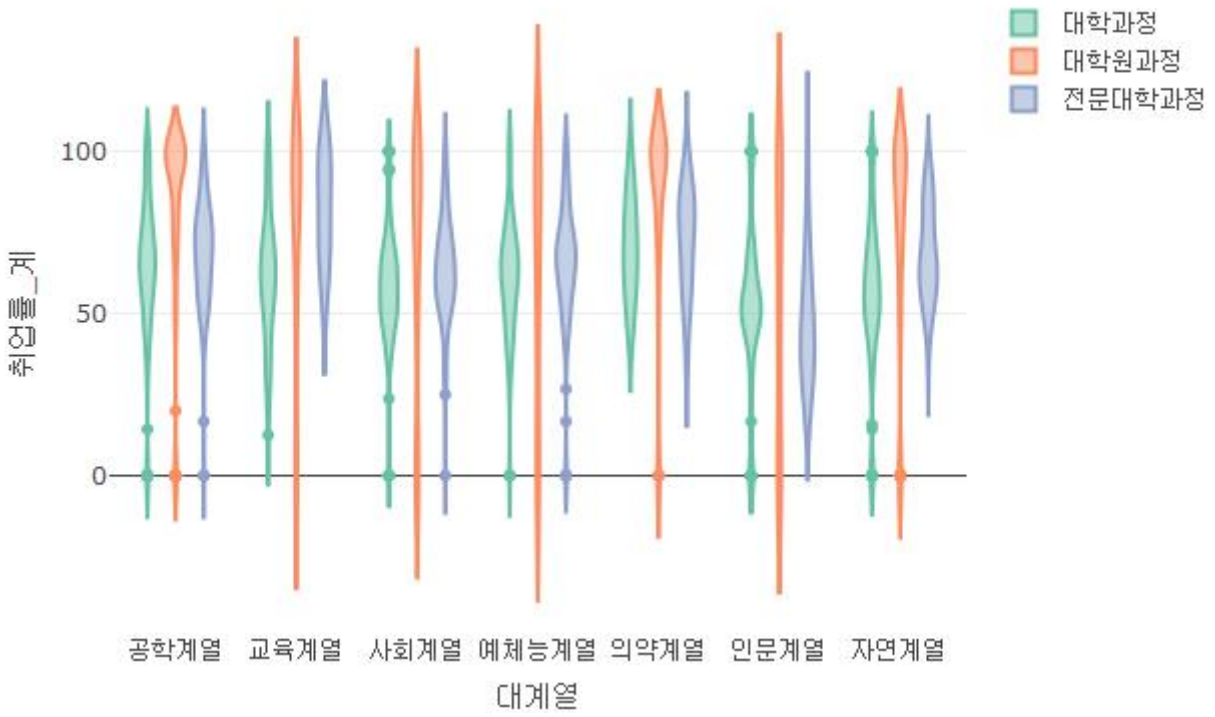
```
add_trace(p, type = ‘violin’, …, data = NULL, inherit = TRUE)
- p : plot_ly()로 생성한 plotly 객체
- type : trace 타입을 ’violin’로 설정
- … : 바이올린 trace 의 line 모드에 설정할 수 있는 속성 설정
- data : 시각화할 데이터프레임
- inherit : plot_ly()에 설정된 속성 type 을 상속할지를 결정하는 논리값
```

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

p_violin |> add_trace(type = 'violin', x = ~대계열, y = ~취업률_계)
```



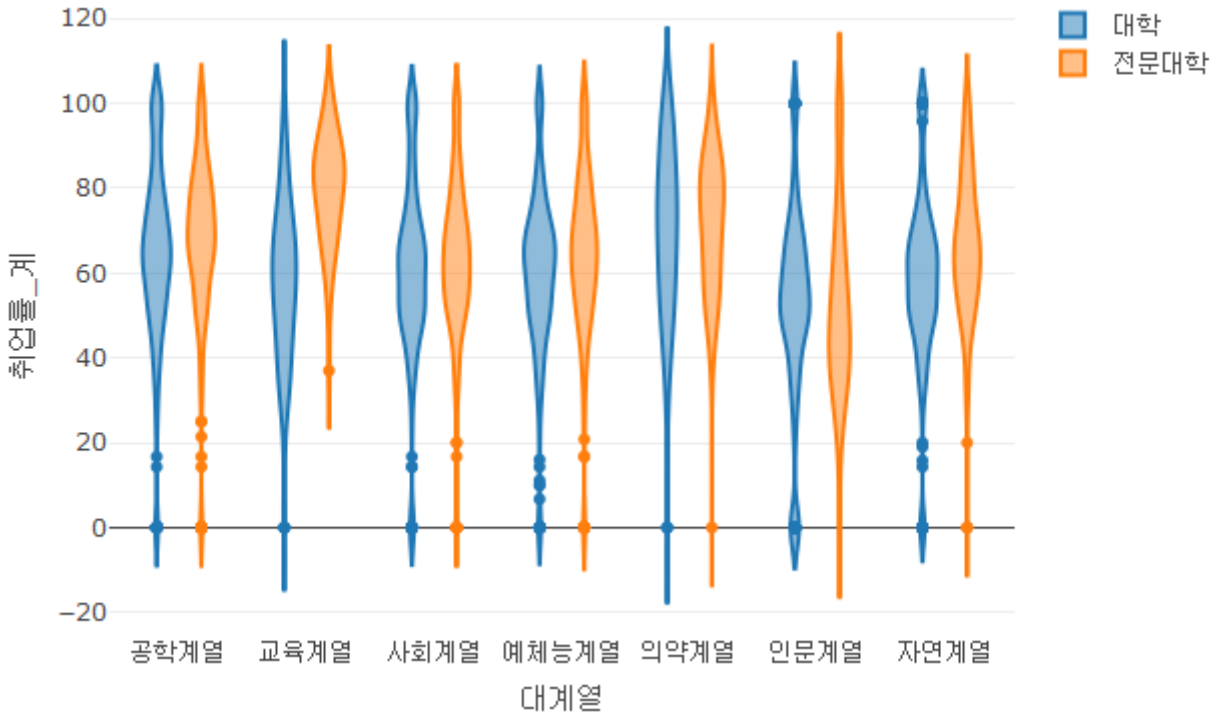
박스 trace 와 마찬가지로 각 데이터별로 그룹화하여 분할된 바이올린 trace 를 생성하기 위해서는 그룹화할 변수를 `color` 나 `linetype` 등의 type 에 매핑 시키거나 각각의 바이올린 trace 를 추가하고 `layout()`의 `violinmode` 를 'group'으로 설정함으로써 생성할 수 있다. 다음의 코드는 `color` 로 과정구분을 매핑하여 각각의 과정별 박스 플롯을 생성하는 코드이다.

```
p_violin |> add_trace(type = 'violin', x = ~대계열, y = ~취업률_계,
                      color = ~과정구분) |>
  layout(violinmode = 'group')
```



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

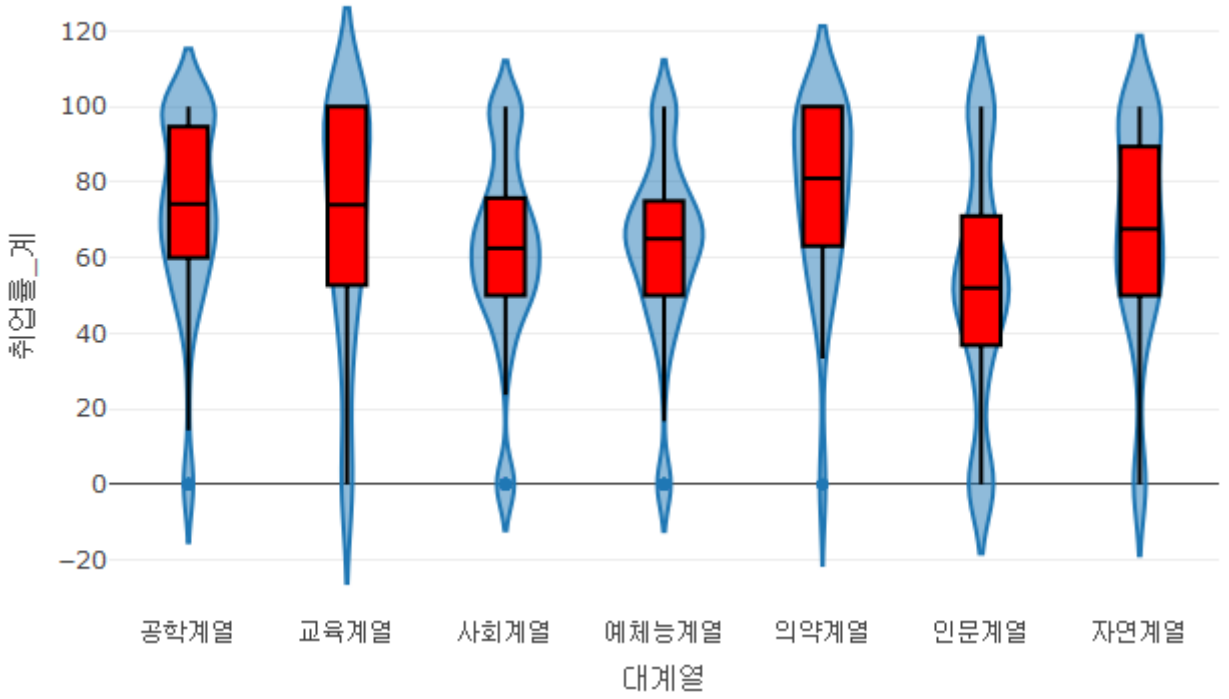
```
p_violin |> add_trace(data = df_취업통계 |> filter(졸업자_계 < 500, 과정구분 == '대학과정'),
                      type = 'violin', x = ~대계열, y = ~취업률_계, name = '대학')
 |>
  add_trace(data = df_취업통계 |> filter(졸업자_계 < 500, 과정구분 == '전문대학과정'),
            type = 'violin', x = ~대계열, y = ~취업률_계, name = '전문대학') |>
  layout(violinmode = "group")
```



3.2.5.1. box, meanline

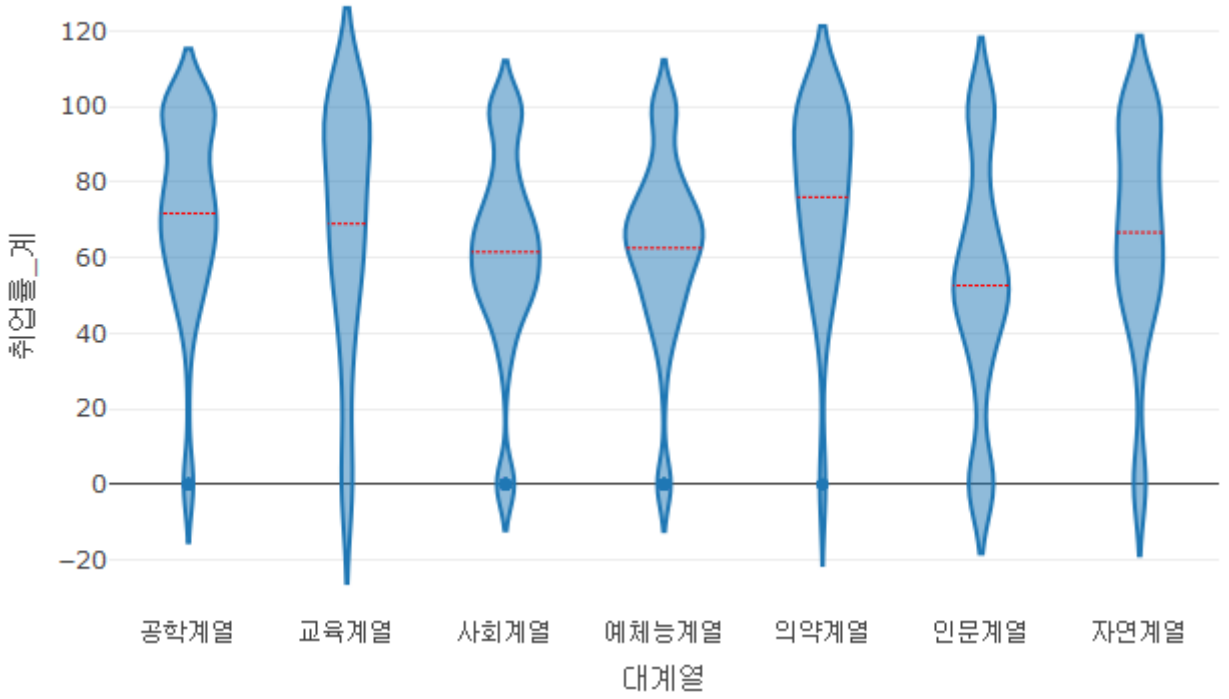
바이올린 플롯은 박스 플롯와 유사한 정보를 제공하는 시각화이지만 박스 플롯의 정보와 같이 사용되면 더욱 데이터 시각화의 효과가 높아진다. 바이올린 trace 에 박스 trace 를 추가하는 type 은 `box` 이고 이 `box` type 은 세부 타입을 list 로 설정하는 type 이다. `box` 의 세부 type 은 `fillcolor`, `line`(세부 type 으로 `color`, `width`), `visible`, `width` 의 네 가지이다.

```
p_violin |> add_trace(type = 'violin', x = ~대계열, y = ~취업률_계,
                      box = list(visible = TRUE, fillcolor = 'red',
                                 line = list(color = 'black'), width = 0.5
                                 )
                      )
```



`meanline` 은 바이올린 trace 에 평균 선을 설정하는 type 으로 `color`, `visible`, `width` 의 세부 type 을 리스트로 설정한다

```
p_violin |>
  add_trace(type = 'violin', x = ~대계열, y = ~취업률_계,
            meanline = list(visible = TRUE, color = 'red', width = 1)
            )
```

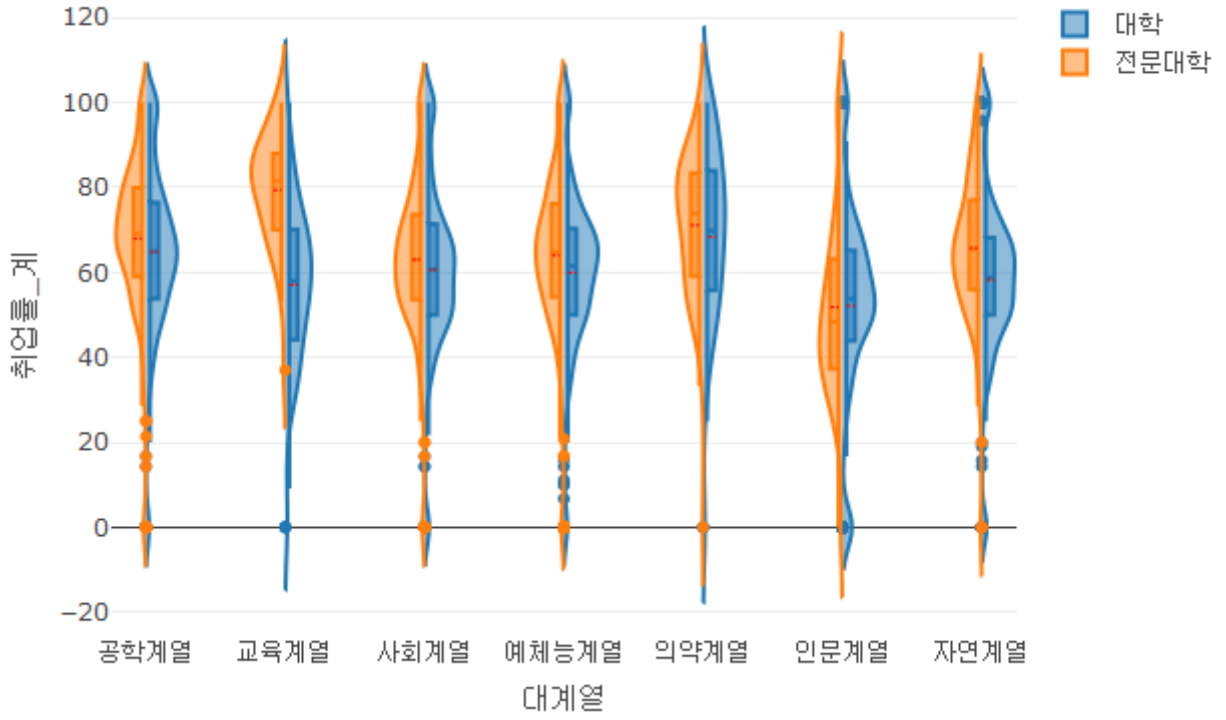


3.2.5.2. side

앞의 예에서 대학과 전문대학의 바이올린 trace 를 추가함으로써 그룹화된 바이올린 플롯을 그렸다 그런데 이 두 바이올린 플롯을 반씩 붙여서 그리면 각각의 바이올린 플롯이 넓어지기 때문에 데이터를 확인하기 쉬울 것이다. 이렇게 두개의 바이올린 플롯을 반씩 잘라 붙이는 type 이 `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 붙여주게 된다. 앞서 그렸던 대학과 전문대학의 계열별 바이올린 플롯을 붙이는 코드는 다음과 같다.

```
p_violin |>
  add_trace(data = df_취업통계 |> filter(졸업자_계 < 500, 과정구분 == '대학과정'),
            type = 'violin', x = ~대계열, y = ~취업률_계, name = '대학',
            side = 'positive', box = list(visible = TRUE, width = 0.5),
            meanline = list(visible = TRUE, color = 'red', width = 1)
            ) |>
  add_trace(data = df_취업통계 |> filter(졸업자_계 < 500, 과정구분 == '전문대학과정
'),
            type = 'violin', x = ~대계열, y = ~취업률_계, name = '전문대학',
            side = 'negative', box = list(visible = TRUE, width = 0.5),
```

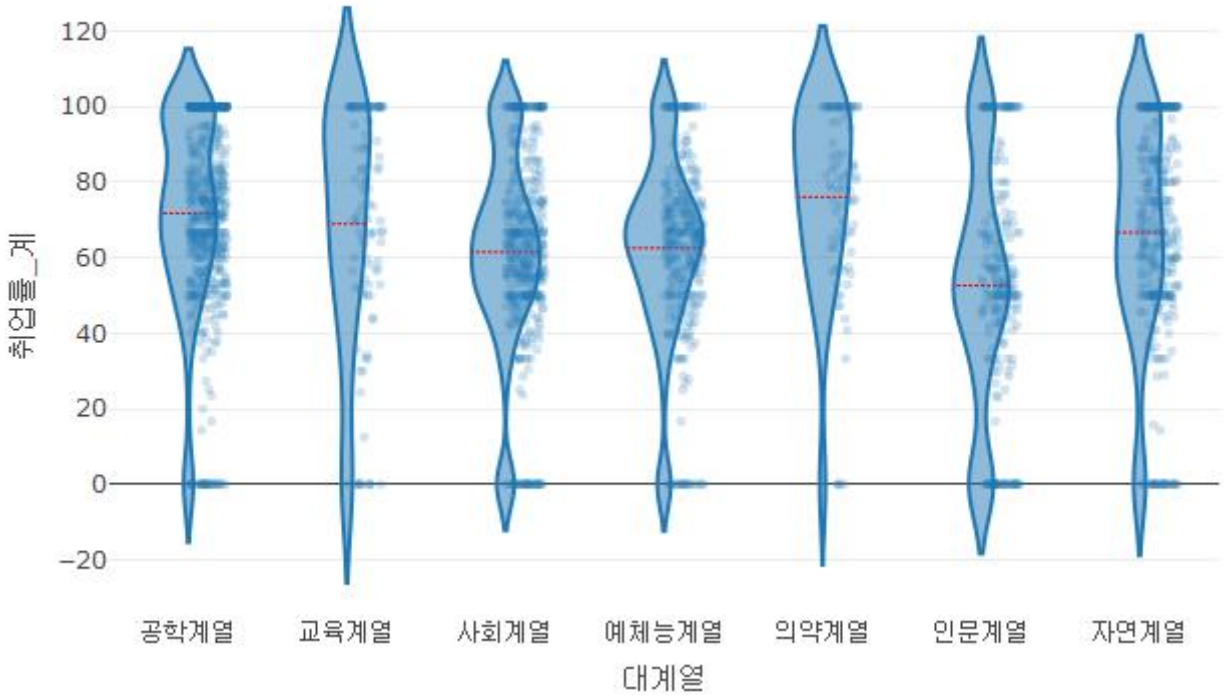
```
            meanline = list(visible = TRUE, color = 'red', width = 1)
            ) |>
  layout(violinmode = "overlay")
```



3.2.5.3. marker, jitter, line

바이올린 trace 에서도 박스 trace 와 같이 데이터를 점으로 표현할 수 있는데 이를 제어하기 위한 type 이 `marker` 와 `jitter` 이고 바이올린의 외곽선을 설정하는 type 이 `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_계,
            meanline = list(visible = TRUE, color = 'red', width = 1),
            marker = list(opacity = 0.2, size = 5, color = 'aquablue'),
            points = "all", jitter = 0.5, pointpos = 0.5)
```



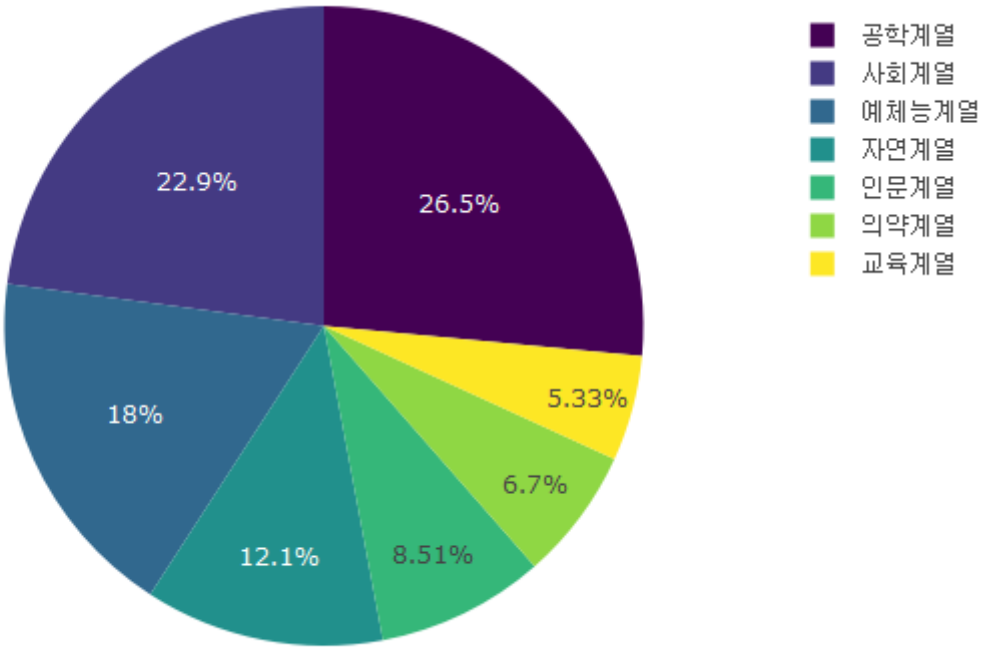
3.2.6. 파이(pie) trace : 원 그래프

`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 trace 로 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 trace 의 line 모드에 설정할 수 있는 속성 설정
- data : 시각화할 데이터프레임
- inherit : plot_ly()에 설정된 속성 type 을 상속할지를 결정하는 논리값
- values : 원 그래프의 각각의 슬라이스에 매칭될 값 설정
- labels : 원 그래프의 각각의 슬라이스에 표시될 라벨 설정
```

앞선 여타 trace 와는 달리 파이 trace 에는 X, Y 축에 매핑되는 `x`, `y` type 의 설정이 없다. 대신 필수적 type 으로 `values` 와 `labels` 를 설정해야하는데 `values` 에 매핑되거나 설정된 수치 벡터에 따라 원을 각각의 슬라이스로 구분하게 되고 `labels` 에 매핑하거자 설정한 벡터가 각각의 trace 이름으로 설정된다. 또 원 그래프에 표시되는 데이터 값은 기본적으로 백분률 값이 표시된다.

```
p_pie <- df_취업통계_2000 |> group_by(대계열) |>
  summarise(졸업자수 = sum(졸업자_계)) |> plot_ly()

cols <- viridisLite::inferno(7)

p_pie |>
  add_trace(type = 'pie', values = ~졸업자수, labels = ~대계열) |>
  layout(colorway = viridis::viridis(7))
```

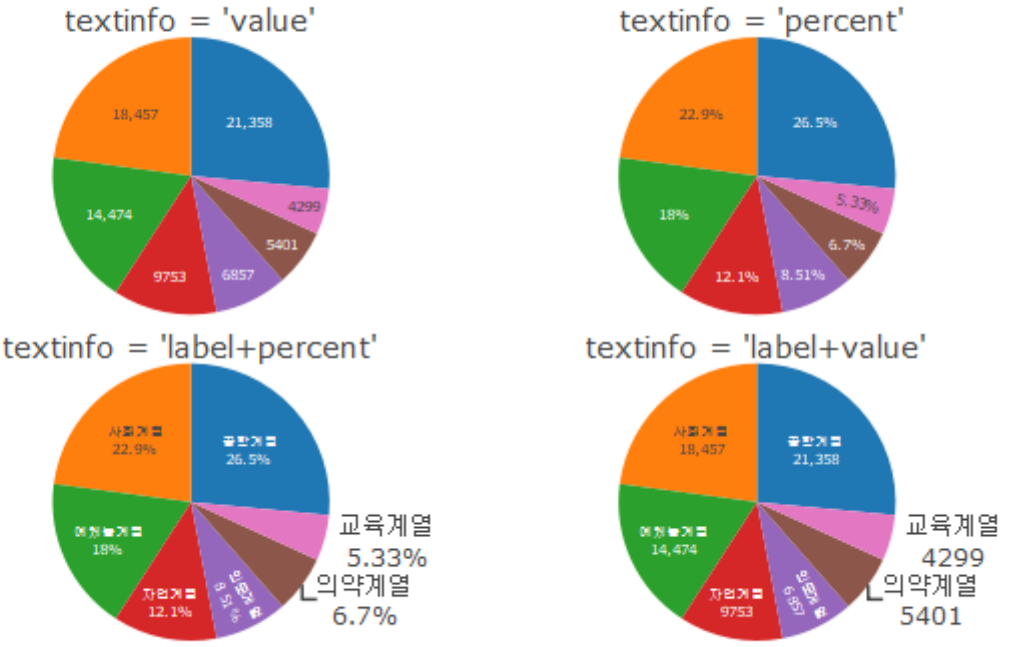


3.2.6.1. textinfo

앞의 원 그래프에서 원 안에 표현된 데이터는 `values` 에 매핑된 데이터의 백분율이 표시되었다. 이 데이터의 표시를 설정하는 type 이 `textinfo` 이다. `textinfo` 에서 설정사 가능한 값은 'label', 'text', 'value', 'percent'의 네 가지이고 이 네가지를 `+`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 = 'value
')

p_pie |>
  add_trace(type = 'pie', values = ~졸업자수, labels = ~대계열, textinfo = 'percen
t')

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

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



3.2.6.2. hole

`hole` 은 원 그래프의 내부에 표시되는 구멍의 크기를 설정하는 type 이다. `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.

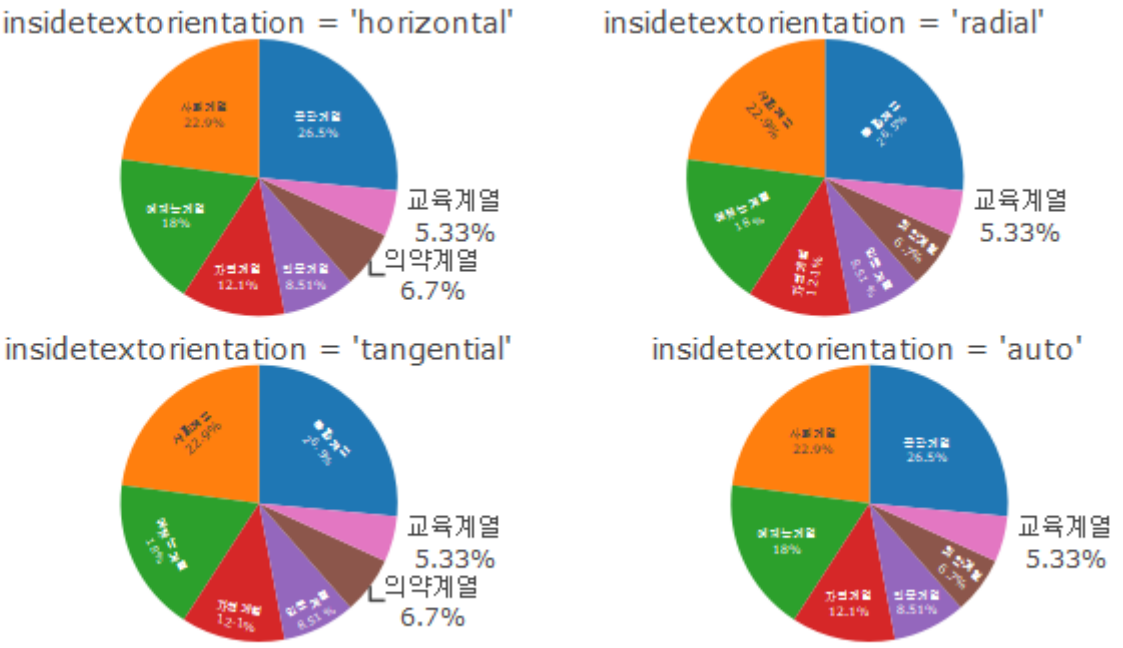
```
p_pie |>
  add_trace(type = 'pie', values = ~졸업자수, labels = ~대계열, textinfo = 'label+
percent', hole = 0.4)
```

3.2.6.3. marker

```
p_pie |>
  add_trace(type = 'pie', values = ~졸업자수, labels = ~대계열, textinfo = 'label+
percent', marker = list(colors = 'viridis', line = list(color = 'white', width =
 1)))
```

3.2.6.4. insidetextorientation

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

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

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

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



3.2.7. 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.

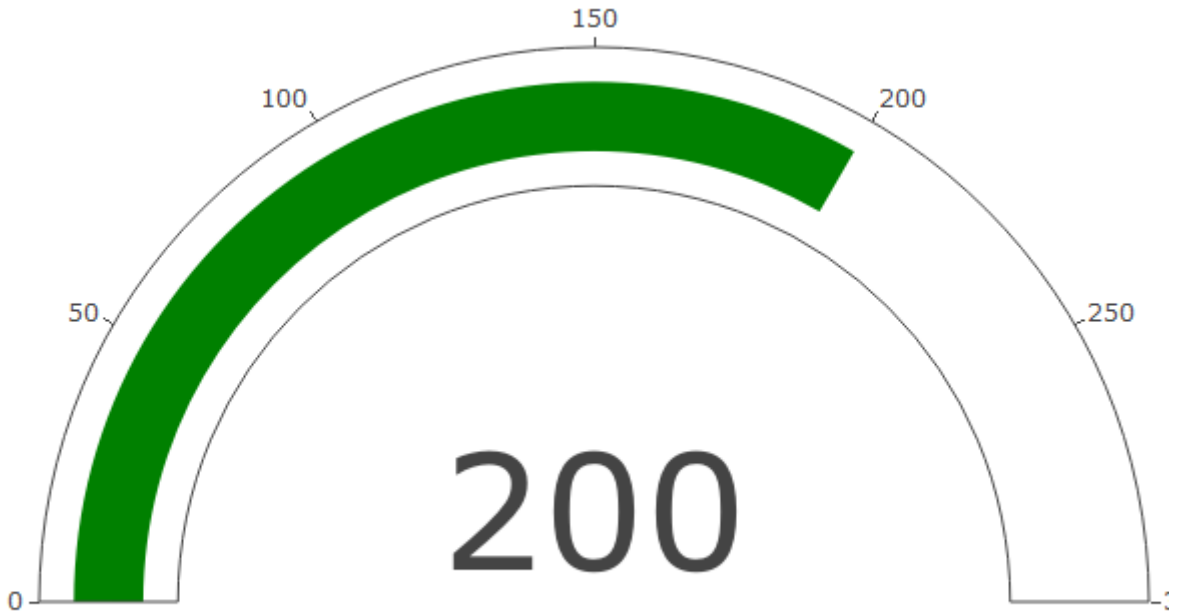
인디케이터 trace 를 사용하기 위해서는 `add_trace()`의 `type` 을 'indicator'로 설정함으로써 사용할 수 있다.

3.2.7.1. mode, value

인디케이터 trace 에서 제공하는 표시 방법은 수치 표시, 증감량 표시, 게이지 표시의 세가지 방법을 제공한다. 이들을 설정하는 type 이 `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 설정이 가능하고 단일 수치를 설정한다.

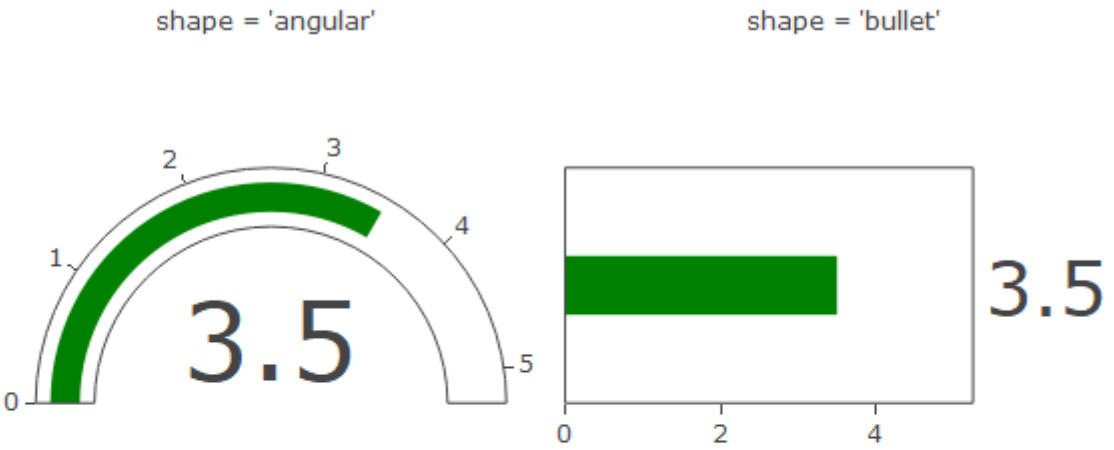
```
plot_ly() |>
  add_trace(type = 'indicator', mode = "gauge+number", value = 200)
```



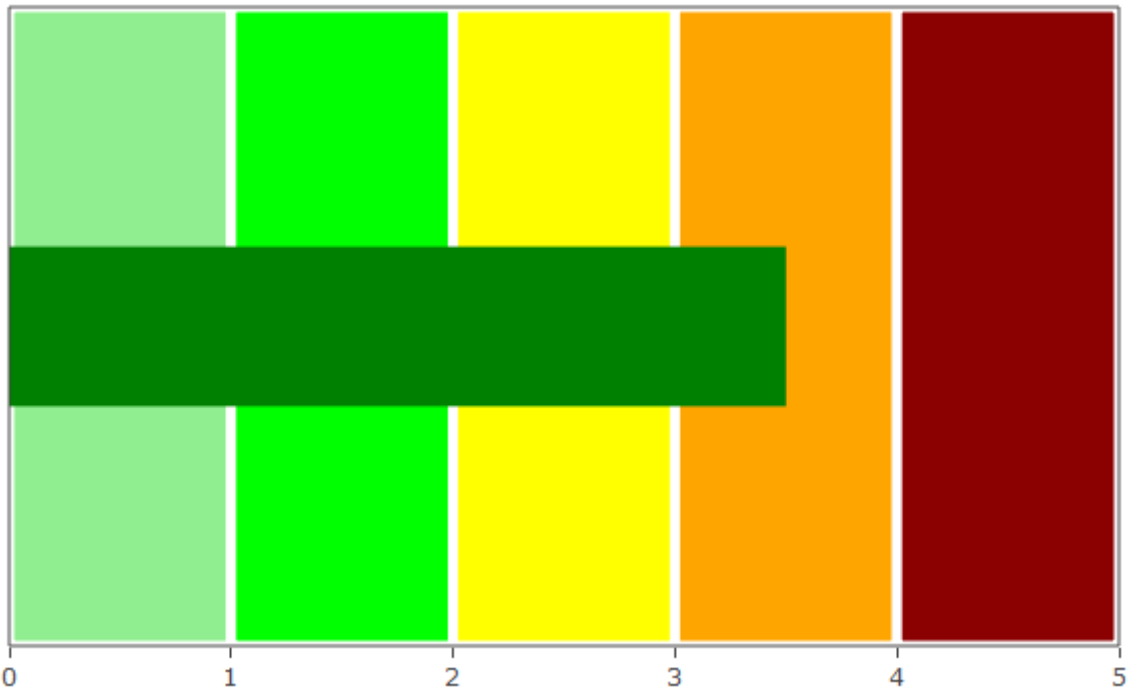
3.2.7.2. gauge

`gauge` 는 게이지 형태의 시각화를 통해서 단일 수치를 표현한다. 게이지의 형태는 둥근 반원 형태의 게이지를 사용하는 'angler'와 막대 형태의 게이지를 사용하는 'bullet'의 두가지가 제공된다. 이는 `shape` type 을 사용하여 설정이 가능하다.

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

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



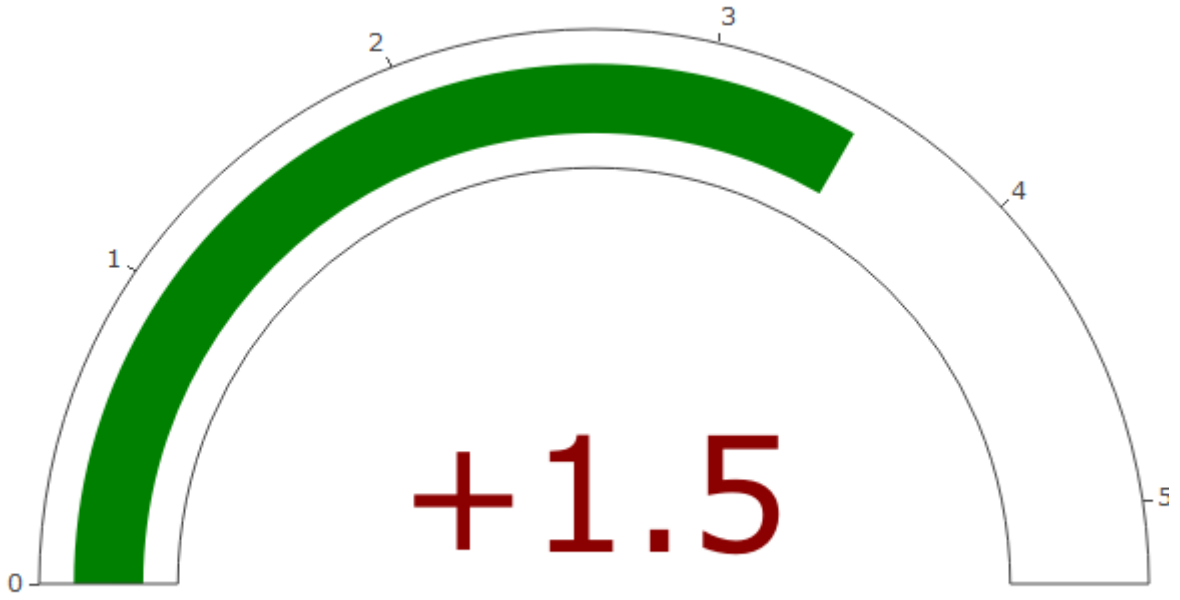

```
plot_ly() |>
  add_trace(type = 'indicator', mode = "gauge",
            gauge = list(shape = 'bullet',
                         axis = list(range = c(0, 5)),
                         steps = list(
                           list(range = c(0,1), color = "lightgreen", name = 'E
',
                                line = list(color = 'white', width = 5)),
                           list(range = c(1,2), color = "lime", name = 'D',
                                line = list(color = 'white', width = 5)),
                           list(range = c(2,3), color = "yellow", name = 'C',
                                line = list(color = 'white', width = 5)),
                           list(range = c(3,4), color = 'orange', name = 'B',
                                line = list(color = 'white', width = 5)),
                           list(range = c(4,5), color = "darkred", name = 'A',
                                line = list(color = 'white', width = 5))
                           ),
                         threshold = list(line = list(color = 'red'),
                                          value = 90)
                         ),
            value = 3.5)
```



3.2.7.3. 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",
            gauge = list(shape = 'angular'), value = 3.5,
            delta = list(reference = 2,
                         increasing = list(color = 'darkred', symbol = '+',
                                           font = list(family = '나눔손글씨 펜')),

                         decreasing = list(color = 'darkblue', symbol = '-')
                         )
  )
```



3.3. 레이아웃 설정 : layout()

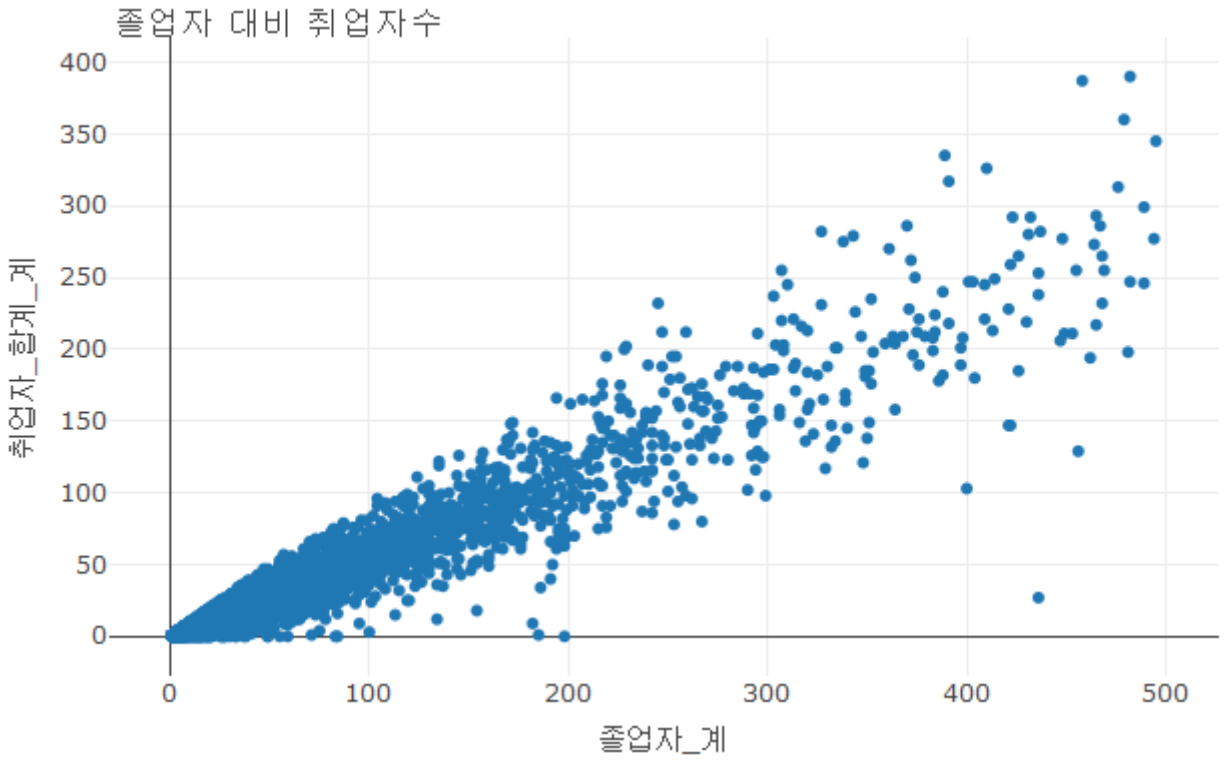
지금까지는 시각화를 만들기 위해 필요한 데이터를 trace 로 표현하는 방법에 대해 알아보았다. 이렇게 추가된 trace 들을 보다 효과적으로 시각화하기 위해서는 시각화의 제목, 범례, 여백, 크기, 폰트, 축 등에 대한 세부 설정을 해야 할 필요가 있다. `plotly` 에서 이들을 설정하기 위해서는 `layout()`을 사용하여 설정할 수 있다. 이 설정들은 앞선 `add_trace()`와 마찬가지로 `list()`를 사용하여 설정이 가능하다.

3.3.1. 제목 설정 : title

`plotly` 의 제목을 설정하는 type 은 `title` 이다. `title` 의 세부 type 은 다음과 같다.

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

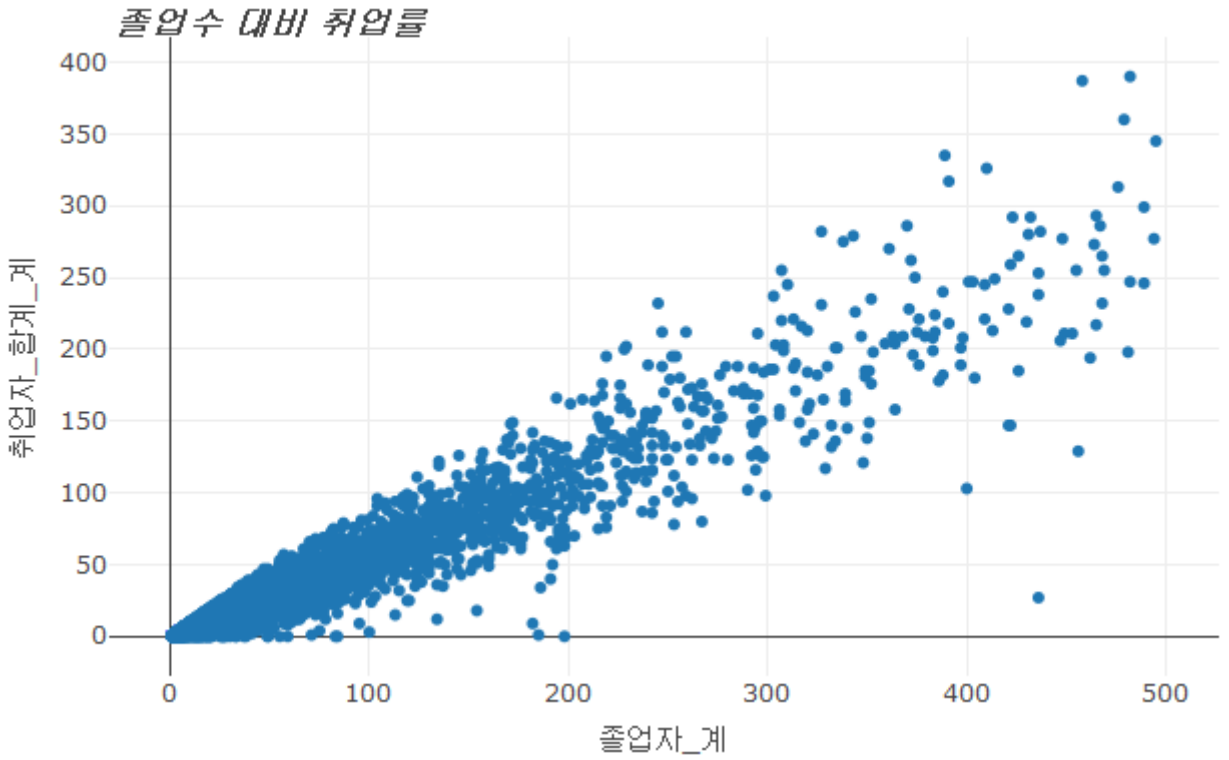
```
p_title_marker <- df_취업통계 |> filter(졸업자_계 < 500) |> plot_ly()

p_title_1 <- p_title_marker |>
  add_trace(type = 'scatter', mode = 'markers', x = ~졸업자_계,
            y = ~취업자_합계_계)

p_title_1 |> layout(title = list(text = '졸업자 대비 취업자수', x = 0.1,
                                 xanchor = 'left', yanchor = 'top'
                                 )
                    )
```



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

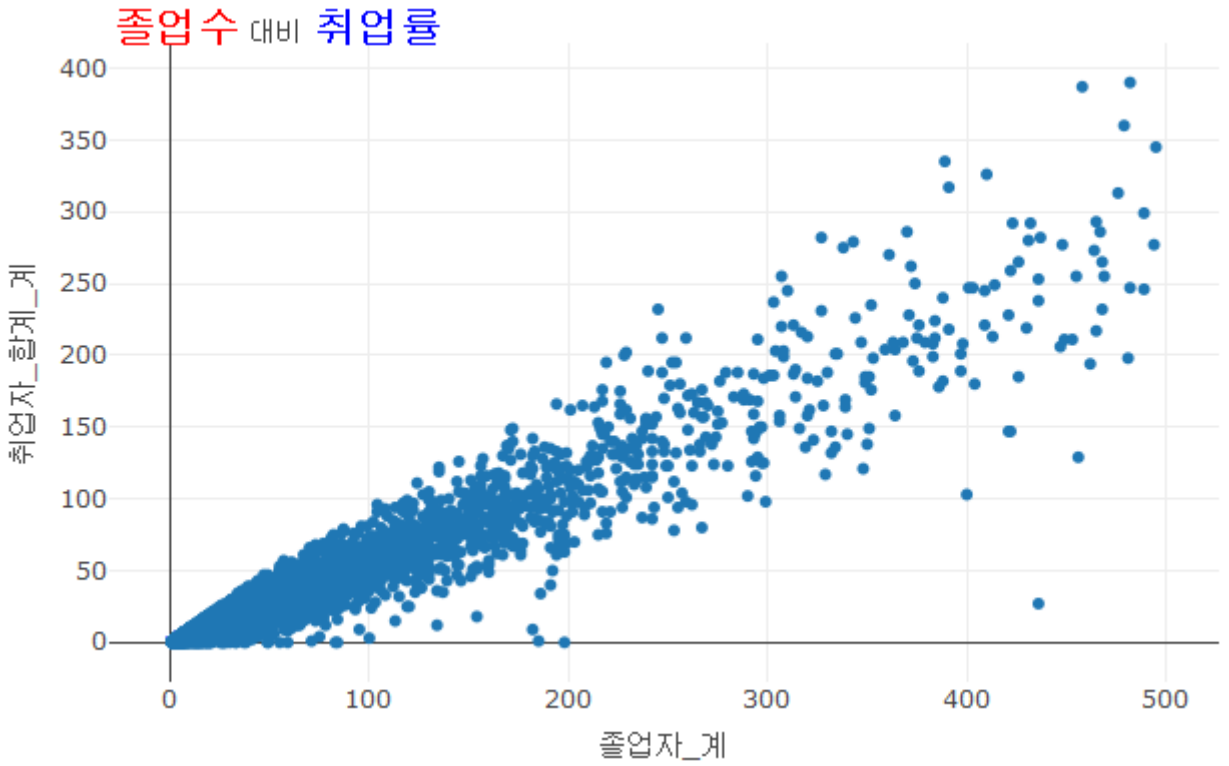
HTML tab	설명
	볼드체 설정
	이탤릭체 설정
 	줄 바꿈 설정
	윗 첨자 설정
	아래 첨자 설정
	하이퍼링크 설정

```
p_title_1 |> layout(title = list(text = "<b><i>졸업수 대비 취업률</b></i>",
                                 x = 0.1, xanchor = 'left', yanchor = 'top')
                     )
```



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_title_1 |> layout(title = list(text = "<span style = 'font-size:15pt'><span st
yle = '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')
                    )
```



3.3.2. 범례 설정 : legend

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

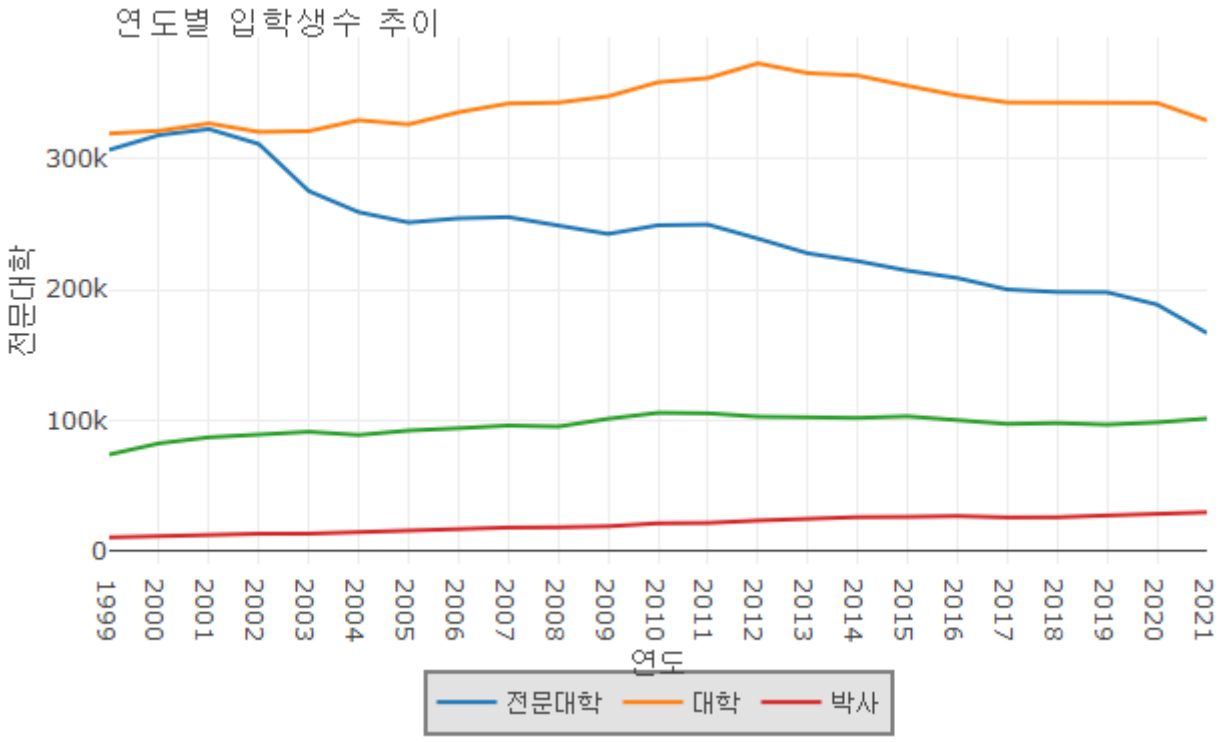
`showlegend`는 `add_trace()`에서도 설정이 가능한 type 인데 `add_trace()`에서 설정하면 해당 trace 만을 범례에서 제거하고 `layout()`에서 설정하면 전체 범례를 제가하게 된다.

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

type	설명	type 값 및 설명	세부 type
bgcolor	범례의 배경색 설정		
bordercolor, borderwidth	범례를 둘러싸는 외곽선 색, 두께 설정		
font	범례의 문자열에 적용되는 문자 속성 설정		color, family, size
groupclick	범례 아이템을 클릭할 때 범례 그룹의 동작 설정	'toggleitem'은 범례에서 클릭한 항목의 표시를 토들, 'togglegroup'은 범례에서 클릭한 항목 그룹의 표시를 토글	
grouptitlefont	범례 아이템 그룹의 제목과 제목 설정	범례 그룹 제목과 그 형태 설정	color, family, size
itemclick	범례 아이템을 클릭 할 때의 동작 설정	'toggle'은 범례에서 클릭한 항목의 표시를 토글, 'toggleothers'는 클릭한 항목만을 보이게 설정, "FALSE"는 범례 항목 클릭 동작을 비활성화	
itemdoubleclick	범례 아이템을 더블 클릭 할 때의 동작 설정	'toggle'은 범례에서 더블 클릭한 항목의 표시를 토글, 'toggleothers'는 더블 클릭한 항목만을 보이게 설정, "FALSE"는 범례 항목 더블 클릭 동작을 비활성화	

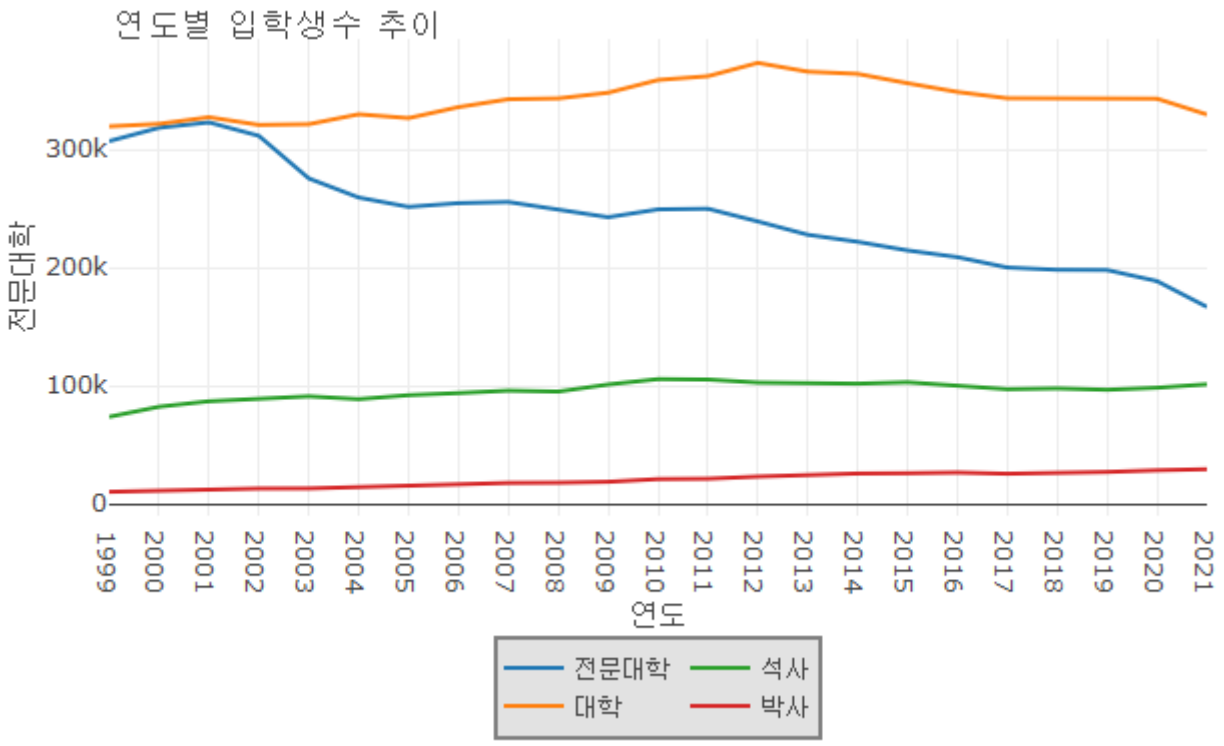
type	설명	type 값 및 설명	세부 type
orientation	범례 표시 방향의 설정	'v'는 세로형태, 'h'는 가로 형태로 표시	
title	범례 제목 설정		font(color, family, size), side, text
traceorder	범례 순서 설정	'reversed', 'grouped', 'normal'의 조합('+') "normal"은 trace 의 추가와 동일한 순서, "reversed"는 'normal'과 반대 순서, 'grouped'는 'legendgroup'이 설정된 경우 그룹의 순서로 표시	
x, y, xanchor, yanchor	범례 위치 설정		

```
p_title_line <- df_입학자 |> filter(지역 == '전체') |> plot_ly()

p_title_2 <- add_trace(p_title_line, type = 'scatter', mode = 'lines',
                       x = ~연도, y = ~전문대학, name = '전문대학')

p_title_2 <- p_title_2 |> add_trace(type = 'scatter', mode = 'lines',
                                    x = ~연도, y = ~일반대학, name = '대학')

p_title_2 <- p_title_2 |> add_trace(type = 'scatter', mode = 'lines',
                                    x = ~연도, y = ~석사, name = '석사', showlegen
d = FALSE)

p_title_2 <- p_title_2 |> add_trace(type = 'scatter', mode = 'lines',
                                    x = ~연도, y = ~박사, name = '박사')

p_title_2 |> layout(title = list(text = '연도별 입학생수 추이', x = 0.1,
                                 xanchor = 'left', yanchor = 'top'),
                    legend = list(bgcolor = "#E2E2E2", bordercolor = 'gray',
                                  borderwidth = 2, orientation = 'h',
                                  x = 0.5, y = -0.2, xanchor = 'center')
                    )
```




```
p_title_line <- df_입학자 |> filter(지역 == '전체') |> plot_ly()

p_title_2 <- add_trace(p_title_line, type = 'scatter', mode = 'lines',
                       x = ~연도, y = ~전문대학, name = '전문대학', legendgroup =
'학부')

p_title_2 <- p_title_2 |>
  add_trace(type = 'scatter', mode = 'lines',
            x = ~연도, y = ~일반대학, name = '대학', legendgroup = '학부')

p_title_2 <- p_title_2 |>
  add_trace(type = 'scatter', mode = 'lines',
            x = ~연도, y = ~석사, name = '석사', legendgroup = '대학원')

p_title_2 <- p_title_2 |>
  add_trace(type = 'scatter', mode = 'lines',
            x = ~연도, y = ~박사, name = '박사', legendgroup = '대학원')

p_title_2 |> layout(title = list(text = '연도별 입학생수 추이', x = 0.1,
                                 xanchor = 'left', yanchor = 'top'),
                    legend = list(bgcolor = "#E2E2E2", bordercolor = 'gray',
                                  borderwidth = 2, orientation = 'h',
                                  x = 0.5, y = -0.25, xanchor = 'center',
                                  grouptitlefont = list(color = 'red')
```

```
                                    )
                )
```



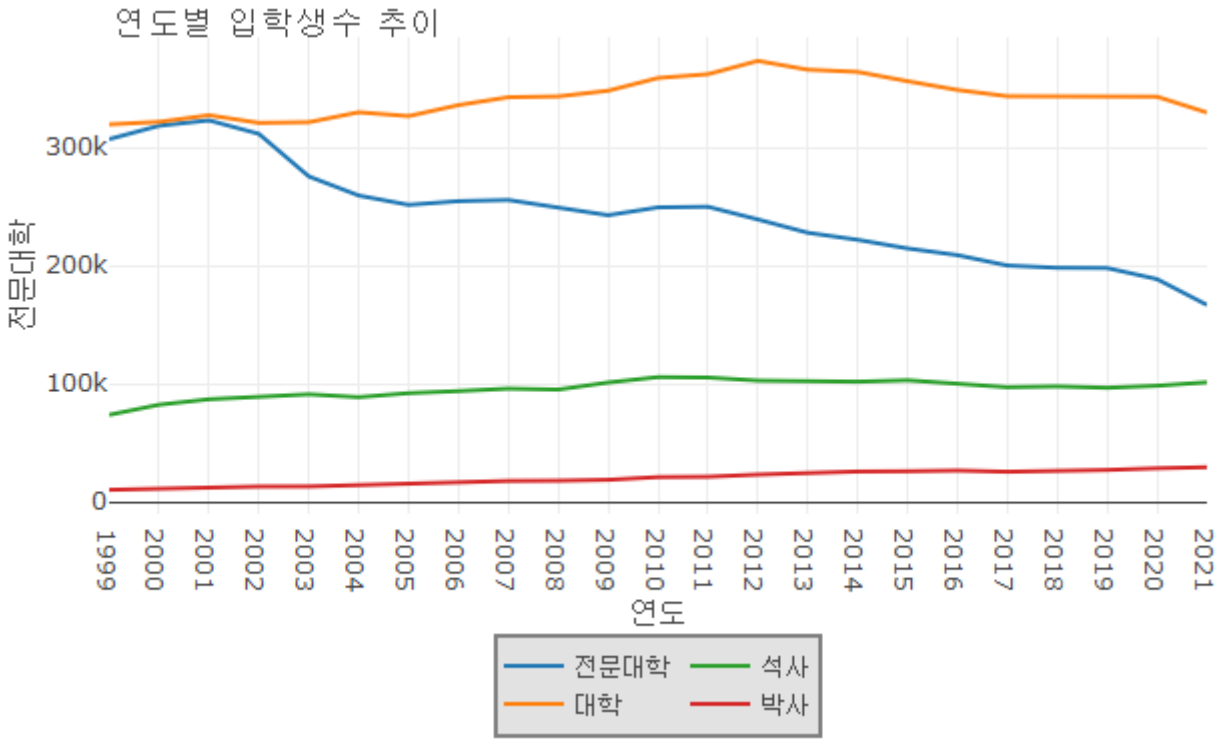
3.3.3. 여백 설정 : margin

`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.

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

```
p_title_2 |> layout(title = list(text = '연도별 입학생수 추이', x = 0.1,
                                 xanchor = 'left', yanchor = 'top'),
                     legend = list(bgcolor = "#E2E2E2", bordercolor = 'gray',
```

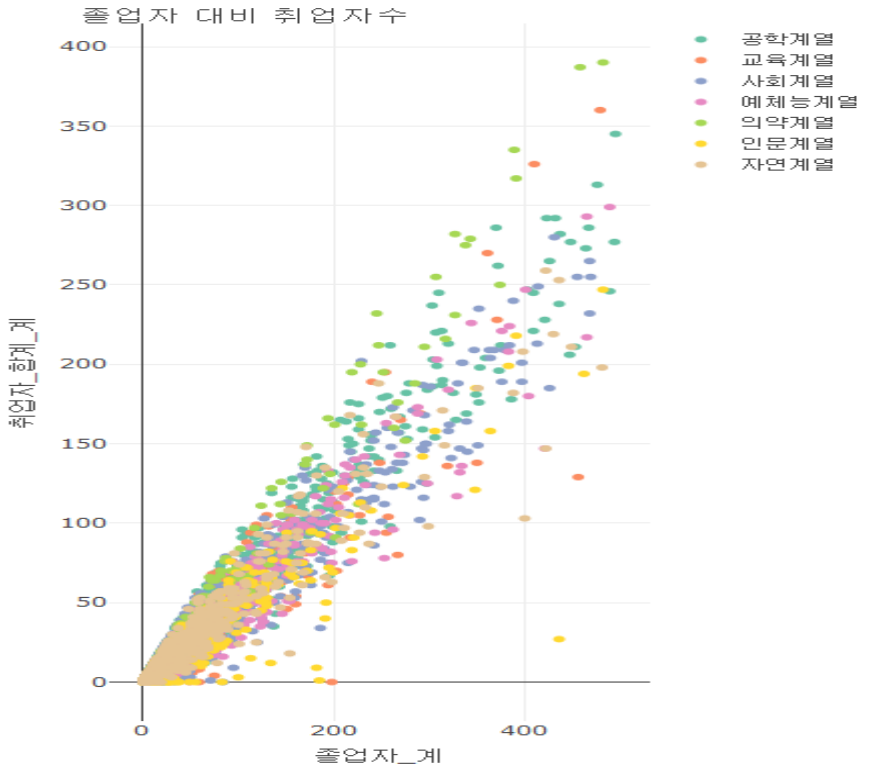
```
                                borderwidth = 2, orientation = 'h',
                                x = 0.5, y = -0.25, xanchor = 'center'),
                  margin = list(autoexpand = TRUE)
                  )
```



3.3.4. 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_title_1 <- p_title_marker |>
  add_trace(type = 'scatter', mode = 'markers', x = ~졸업자_계,
            y = ~취업자_합계_계, color = ~대계열)

p_title_1 |> layout(title = list(text = '졸업자 대비 취업자수', x = 0.1,
                                 xanchor = 'left', yanchor = 'top'),
                    width = 450, height = 700)
```



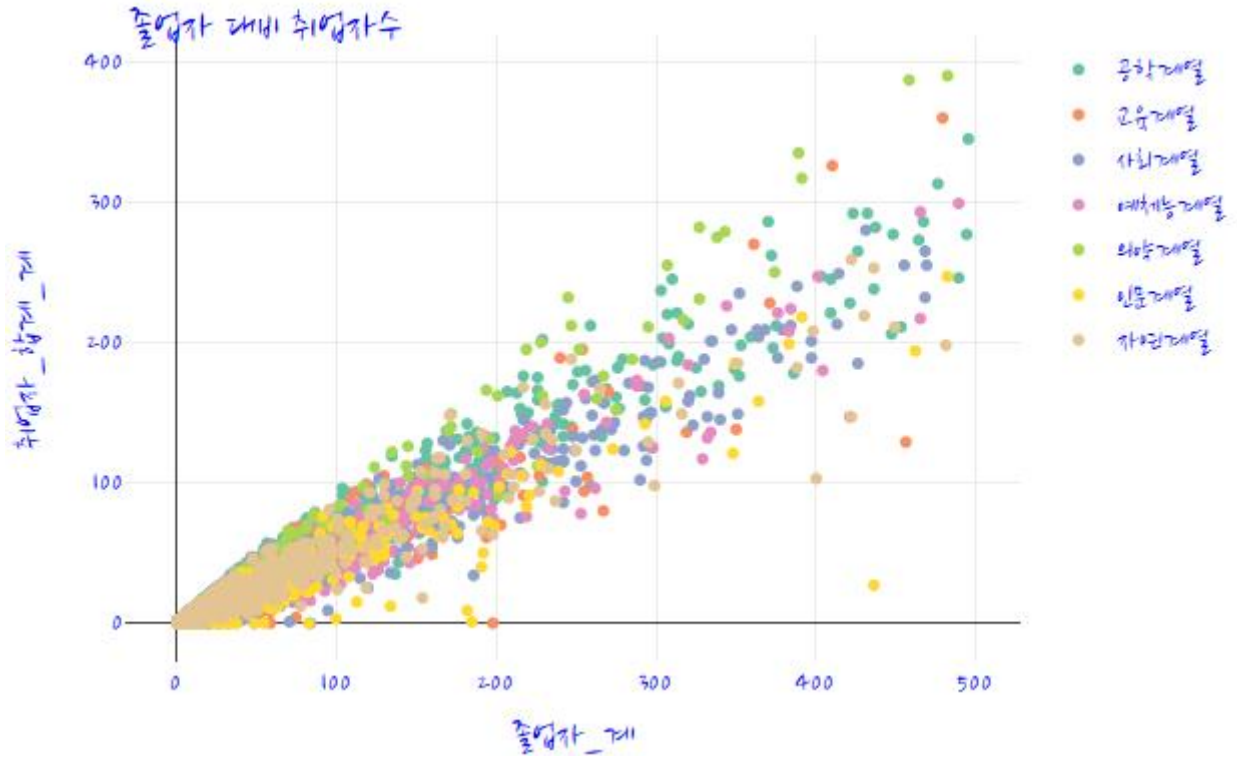
3.3.5. 폰트 설정 : font

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

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

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

```
p_title_1 |> layout(title = list(text = '졸업자 대비 취업자수', x = 0.1,
                                xanchor = 'left', yanchor = 'top'),
                   font = list(family = "나눔손글씨 붓", color = 'blue', size = 1
5)
                   )
```



3.3.6. 색 설정 : color

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

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

`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.org 에서 제공하는 CSS 색 이름을 사용한다.[7]

```
p_title_1 |> layout(title = list(text = '졸업자 대비 취업자수', x = 0.1,
                                 xanchor = 'left', yanchor = 'top'),
                   font = list(family = "나눔손글씨 붓", color = 'blue', size = 1
5),
                   paper_bgcolor = 'floralwhite', plot_bgcolor = 'antiquewhite'
                   )
```

3.3.7. 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.

---

[7] https://www.w3schools.com/cssref/css_colors.asp

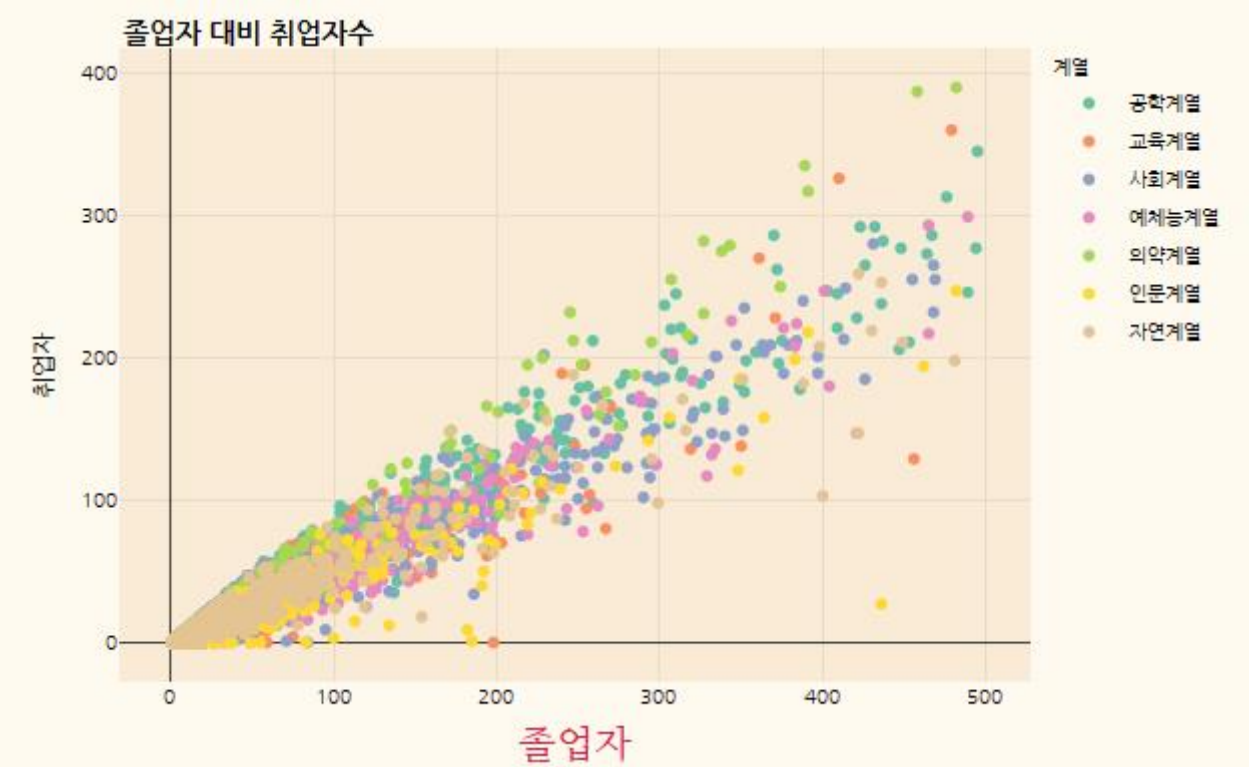
`layout()`에서 X 축의 설정과 관련된 type 은 `xaxis` 이다. `xaxis` 에서 제공하는 주요 type 은 다음과 같다.

type	설명	type 값 및 설명	세부 type
title	축 이름 설정	예외적으로 title 에 세부 list 를 설정하지 않고 직접 문자열로 축이름 설정 가능	font(color, family, size), standoff, text
type	축 유형 설정	'linear'은 선형 축, 'log'은 로그 변환 축, 'date'은 날짜형 축, 'category'은 팩터형 축, 'multicategory'은 다중 팩터형 축 설정	
range	축 범위 설정	축 범위가 지정된 list 설정	
rangemode	축 범위의 특성 설정	'normal'이면 입력 데이터의 최대, 최소값으로 범위를 설정, 'tozero'이면 입력 데이터에 관계없이 범위가 0 부터 시작, 'non-negative'이면 음수 범위는 제거됨	
linecolor, linewidth	축 선 설정	'linecolor'는 축 선 색, 'linewidth'는 축 선 두께 설정	
zeroline, zerolinecolor, zerolinewidth	0 선 설정	'zeroline'은 TRUE/FALSE, 'zerolinecolor'는 색, 'zerolinewidth'는 두께 설정	
position	축 선 위치 설정	0 부터 1 까지 상대 위치 설정	
gridcolor, gridwidth	눈금선 설정	'gridcolor'는 눈금선 색, 'gridwidth'는 눈금선 두께 설정	
autotypenumber	축에 설정되는 값이 문자형일때 숫자형으로 변환할지를 결정	'convert types'은 문자형 수치를 숫자형으로 변환하고 'strict'는 문자형 수치를 그대로 사용	

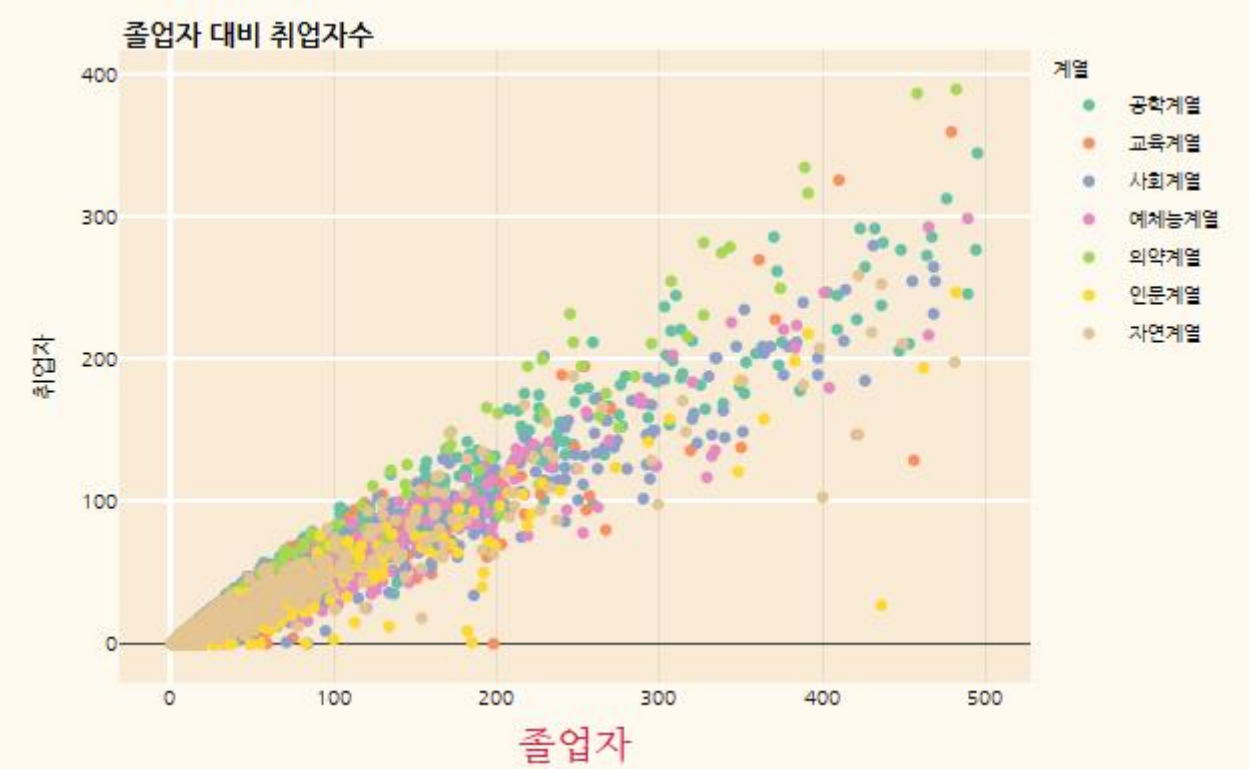
type	설명	type 값 및 설명	세부 type
ticks	눈금자를 어느쪽으로 그릴지 설정	'outside'는 플롯 바깥쪽, 'inside'는 플롯 안쪽으로 눈금자를 그림	
tickangle, ticklen, tickcolor	눈금자의 각도, 길이, 색 설정	'tickangle'은 눈금자 각도, 'ticklen'은 눈금자 길이, 'tickcolor'는 눈금자 색 설정	
tick0, dtick, nticks	눈금자의 시작점, 눈금자의 간격, 눈금자의 개수 설정	'tick0'는 눈금자가 시작하는 위치, 'dtick'은 눈금자의 간격, 'nticks'는 눈금자의 개수 설정	
tickvals, ticktext	눈금자의 위치, 표시문자 설정	'tickvals'에 설정된 수치 벡터나 리스트의 위치에 눈금자 설정, 'ticktext'는 'tickvalues'에 설정된 위치에 표기할 문자열 설정	
showgrid, showline, showticklabel	눈금선, 축 선, 눈금자 라벨의 표시 여부 설정	'showgrid'는 눈금선, 'showline'은 축 선, 'showticklabel'은 눈금자 라벨의 표시 여부를 설정하는 TRUE/FALSE	
tickprefix, ticksuffix	눈금자 라벨의 접두어, 접미어 설정	'tickprefix'는 눈금자 라벨의 접두어, 'ticksuffix'는 접미어 설정	

```
p_title_1 |> layout(title = list(text = '<b>졸업자 대비 취업자수</b>', x = 0.1,
                                 xanchor = 'left', yanchor = 'top', size = 15),
                     font = list(family = "나눔고딕", color = 'black', size = 10),

                     legend = list(title = list(text = '계열', side = 'top')
                                   ),
                     paper_bgcolor = 'floralwhite', plot_bgcolor = 'antiquewhite
',
                     xaxis = list(title = list(text = '졸업자',
                                               font = list(family = '나눔명조', si
ze = 20, color = 'Crimson')
```

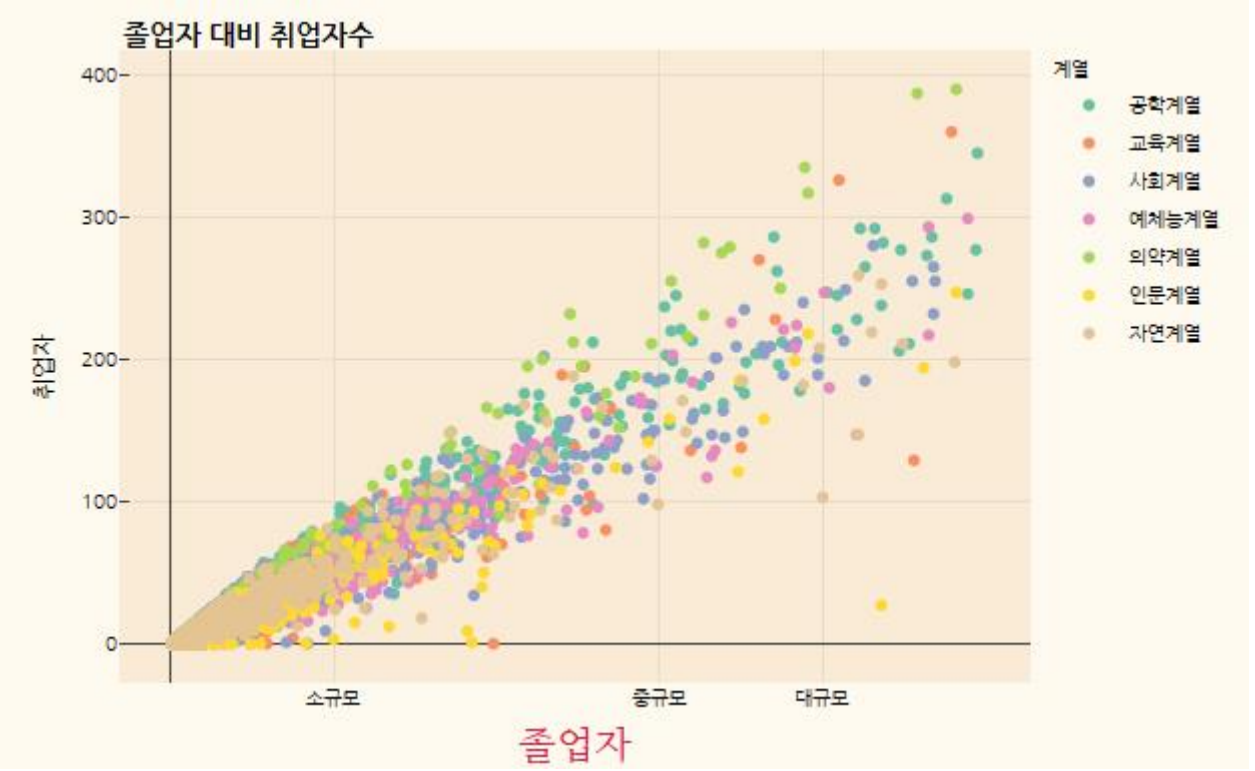
```
                                    )
                                ),  ## 정상적 방법
                    yaxis = list(title = '취업자') ## 약식 방법
                    )
```




```
p_title_1 |> layout(title = list(text = '<b>졸업자 대비 취업자수</b>', x = 0.1,
                                xanchor = 'left', yanchor = 'top', size = 15),
                    font = list(family = "나눔고딕", color = 'black', size = 10),

                    legend = list(title = list(text = '계열', side = 'top')
                                  ),
                    paper_bgcolor = 'floralwhite', plot_bgcolor = 'antiquewhite
',
                    xaxis = list(title = list(text = '졸업자', font = list(family
 = '나눔명조', size = 20, color = 'Crimson')
                                               ),  ## 정상적 방법
                                 zerolinewidth = 3, zerolinecolor = 'white'),
                    yaxis = list(title = '취업자', gridcolor = 'white', gridwidth
 = 2) ## 약식 방법
                    )
```




```
p_title_1 |> layout(title = list(text = '<b>졸업자 대비 취업자수</b>', x = 0.1,
                                 xanchor = 'left', yanchor = 'top', size = 15),
                   font = list(family = "나눔고딕", color = 'black', size = 10),

                   legend = list(title = list(text = '계열', side = 'top')),
                   paper_bgcolor = 'floralwhite', plot_bgcolor = 'antiquewhite
',
                   xaxis = list(title = list(text = '졸업자',
                                             font = list(family = '나눔명조', si
ze = 20, color = 'Crimson')
                                             ),
                                             tickmode = 'array',
                                             ticktext = c('소규모', '중규모', '대
규모'),

                                             tickvals = c(100, 300, 400)
                                ),  ## 정상적 방법
                   yaxis = list(title = '취업자', ticks="inside", tick0 = 100, d
tick = 100) ## 약식 방법
                   )
```



3.3.8. 주석 설정 : annotations

시각화 그래프를 만들때 시각화를 만드는 목적에 따라 시각화 그래프에 데이터에 대한 추가적인 설명이 들어가는 경우가 많다. 이와 같이 시각화를 만드는 사용자가 강조하고자 하거나 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.

type	설명	type 값 및 설명	세부 type
text	주석으로 표시할 문자열을 설정		
x, y	주석을 설정할 X 축과 Y 축의 좌표를 설정		
ax, ay	주석이 실제로 위치할 좌표의 상대적 위치		
xanchor	주석 문자열의 수평 위치 맞춤값 설정	'auto'는 자동설정, 'left'는 왼쪽 맞춤, 'center'는 중간 맞춤, 'right'는 오른쪽 맞춤	
yanchor	주석 문자열의 수직 위치 맞춤값 설정	'auto'는 자동설정, 'top'은 상단 맞춤, 'middle'은 중간 맞춤, 'bottom'은 하단 맞춤	
xref, yref	x, y, ax, ay 의 절대 좌표와 상대 좌표의 참조 기준을 설정	'x', 'y'는 축 스케일을 참조, 'paper'는 플로팅 영역을 0 부터 1 까지 정규화하여 참조	
arrowcolor	화살촉의 색 설정		
arrowhead	화살촉의 모양 설정	0 부터 8 까지의 수치	
arrowside	화살촉 위치 설정	'end', 'start', 'none'을'+'로 조합 가능 'end'는 화살표 직선 끝에 화살촉, 'head'는 화살표 직선 앞에 화살촉을 설정	
arrowsize, arrowwidth	화살촉의 크기, 너비 설정		
bgcolor, bordercolor	주석의 배경색, 주석 외곽선 색 설정		

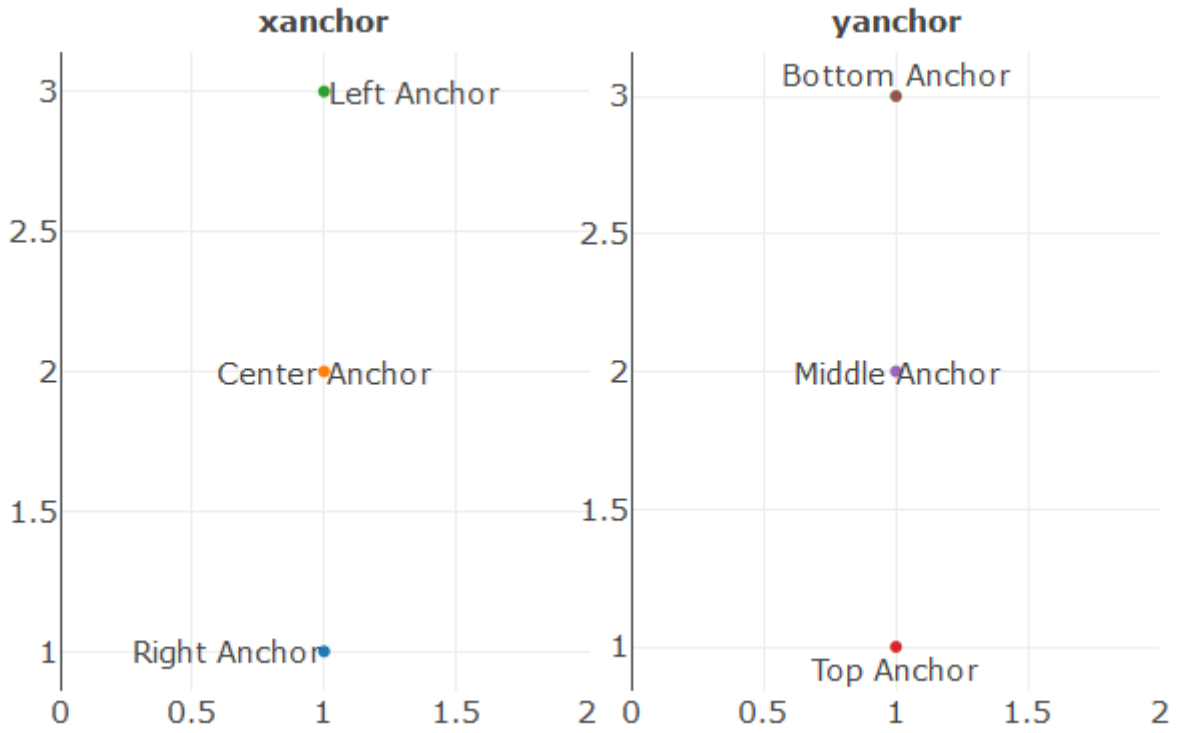
type	설명	type 값 및 설명	세부 type
borderpad, borderwidth	패딩(주석과 외곽선까지의 여백), 주석 외곽선 너비 설정		
font	주석의 문자 형태 설정		color, family, size

`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.

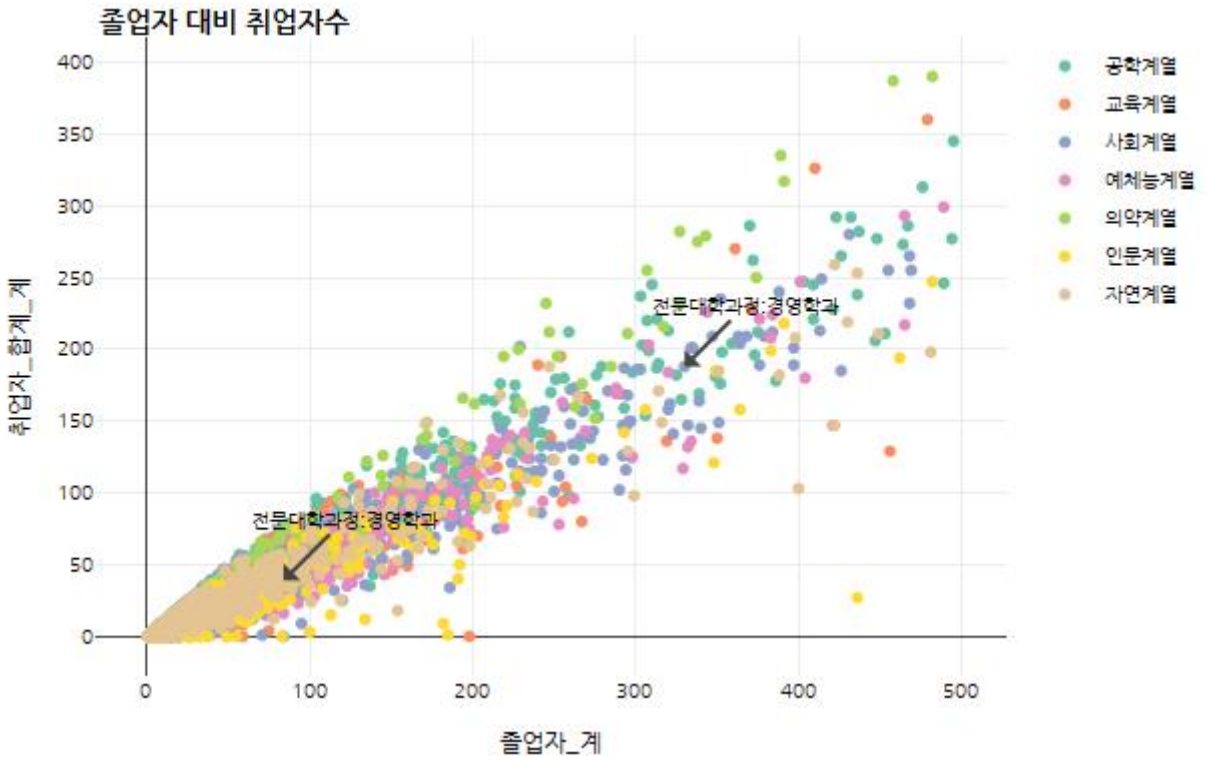
`ax` 와 `ay` 는 `x`, `y` 에 설정된 좌표에 대한 주석이 실제로 위치할 좌표의 상대적 위치를 설정한다. 이 상대적 위치는 화살표로 표시되는데 `ax` 는 `x` 에 설정된 좌표에서의 거리를 설정하고 `ay` 는 `y` 에 설정된 좌표에서의 거리를 설정한다. `ay` 설정에 하나 주의할 점은 Y 축의 양의 방향은 아래에서 위로 향하지만 `ay` 의 양의 방향은 위에서 아래를 향하기 때문에 아래쪽에 위치한다.

`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.

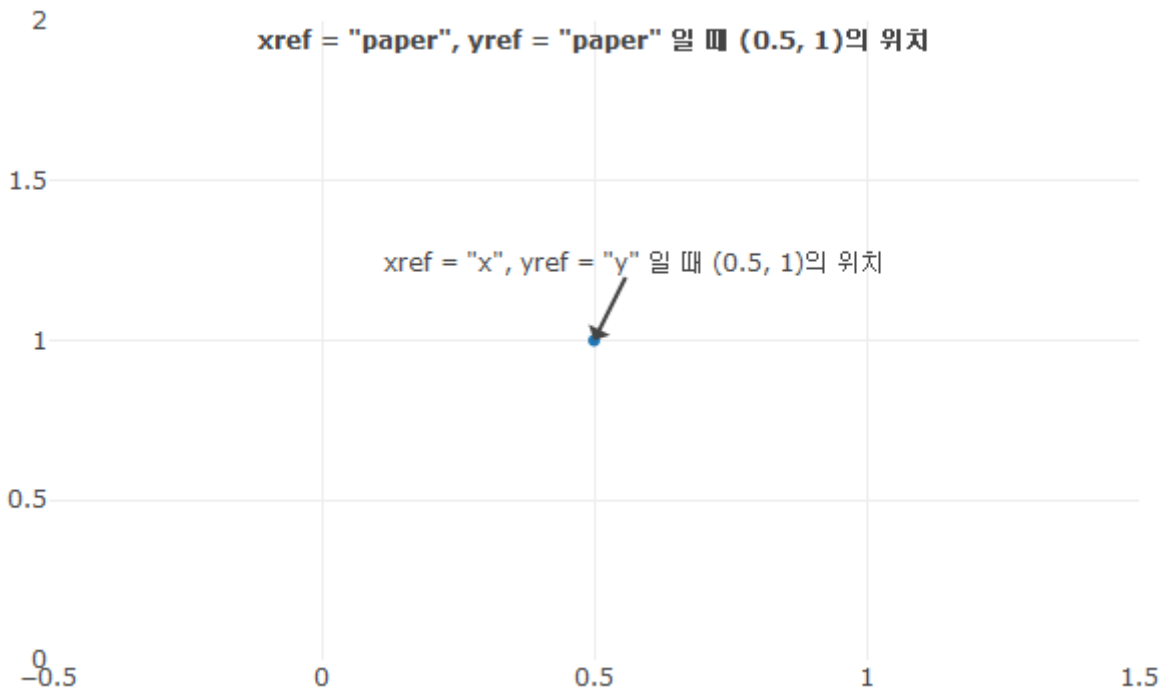


`xref` 와 `yref` 는 `x`, `y`, `ax`, `ay` 의 절대 좌표와 상대 좌표의 참조 기준을 설정한다. `xref`, `yref` 가 'x', 'y'등의 축으로 설정되면 해당 축의 스케일에 맞게 `x`, `y`, `ax`, `ay` 의 좌표와 거리를 계산하고 'paper'로 설정되면 전체 플롯 영역을 0 부터 1 까지로 정규화한 상대적 거리로 계산된다.

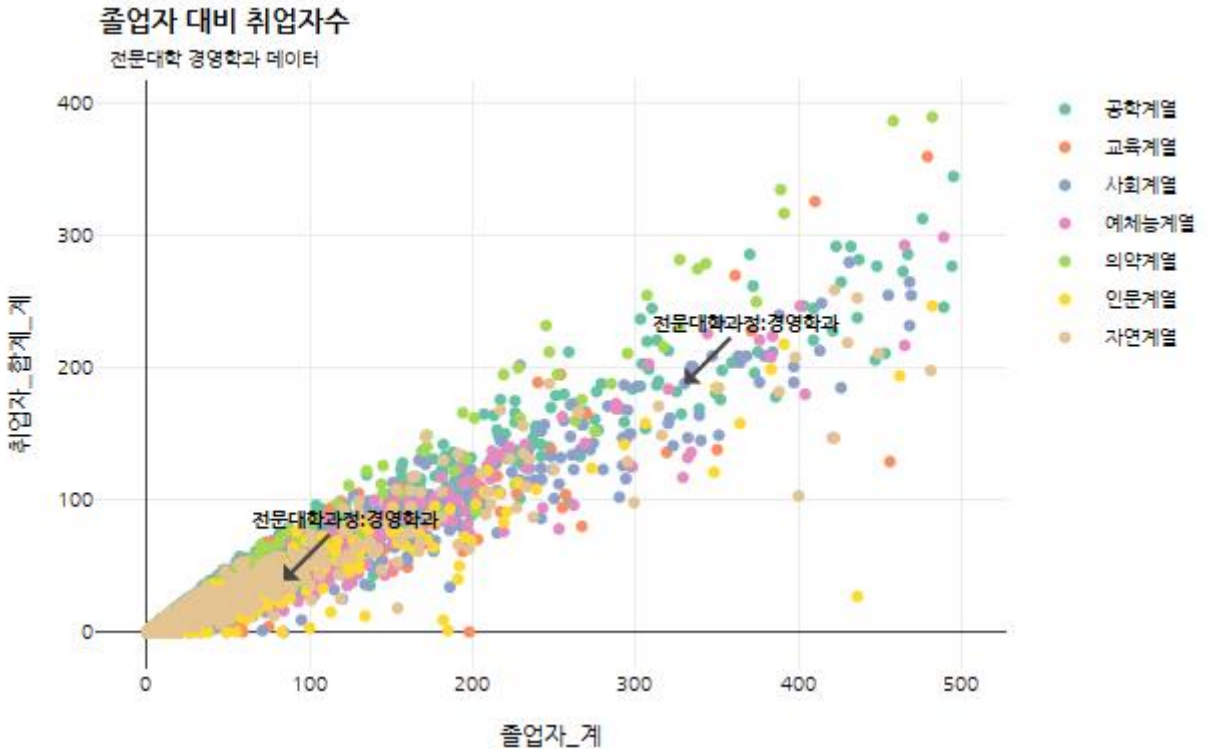
```
df_annotation <- df_취업통계 |> filter(졸업자_계 < 500, 학과명 == '경영학과') |> se
lect(과정구분, 학과명, 졸업자_계, 취업자_합계_계)

p_title_1 |>
  layout(title = list(text = '<b>졸업자 대비 취업자수</b>', x = 0.1,
                      xanchor = 'left', yanchor = 'top', size = 15),
         font = list(family = "나눔고딕", color = 'black', size = 10),
         annotations = list(text = ~paste0(df_annotation$과정구분, ':', df_annota
tion$학과명), x = df_annotation$졸업자_계, y = df_annotation$취업자_합계_계, ax = 3
0, ay = -30)
         )
```



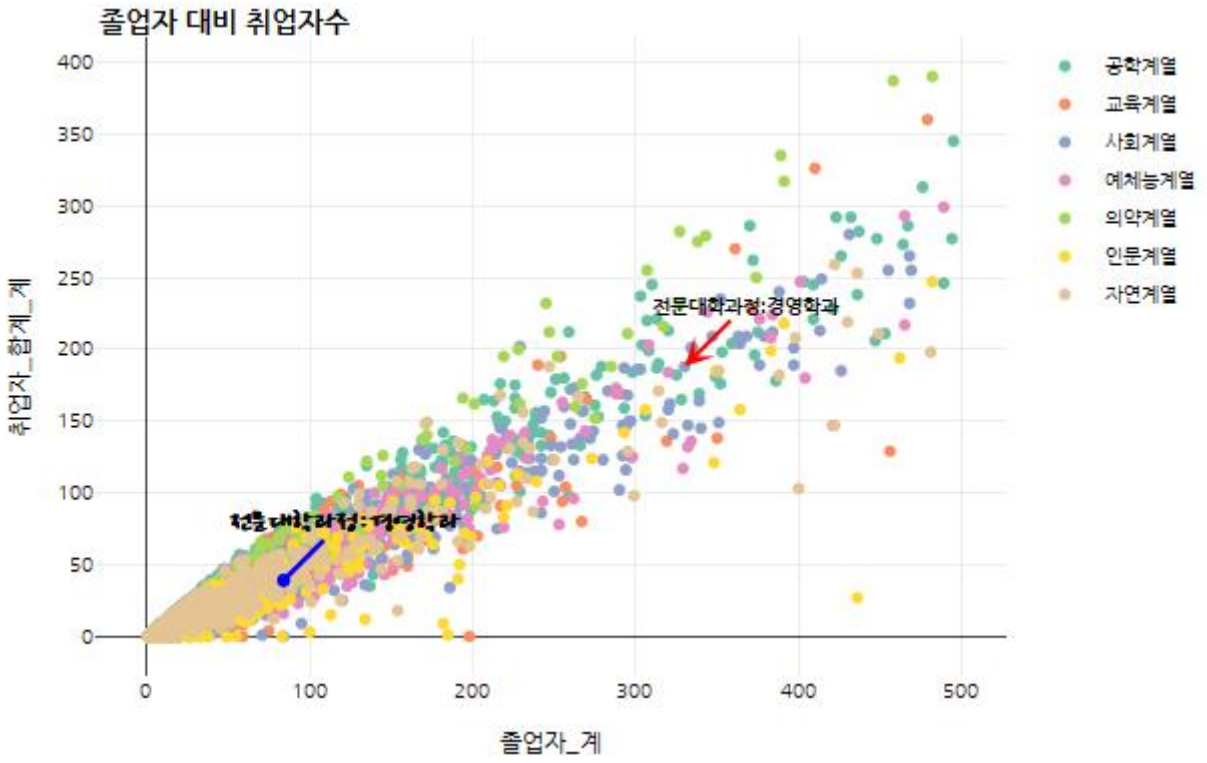
`xref` 와 `yref` 는 `x`, `y`, `ax`, `ay` 의 절대 좌표와 상대 좌표의 참조 기준을 설정한다. `xref`, `yref` 가 'x', 'y'등의 축으로 설정되면 해당 축의 스케일에 맞게 `x`, `y`, `ax`, `ay` 의 좌표와 거리를 계산하고 'paper'로 설정되면 전체 플롯 영역을 0 부터 1 까지로 정규화한 상대적 거리로 계산된다.




```
df_annotation <- df_취업통계 |> filter(졸업자_계 < 500, 학과명 == '경영학과') |> se
lect(과정구분, 학과명, 졸업자_계, 취업자_합계_계)

p_title_1 |>
  layout(title = list(text = '<b>졸업자 대비 취업자수</b>', x = 0.1,
                      xanchor = 'left', yanchor = 'bottom', size = 15),
         font = list(family = "나눔고딕", color = 'black', size = 10),
         annotations = list(text = ~paste0('<b>', df_annotation$과정구분, ':', df
_annotation$학과명, '</b>'),
                            x = df_annotation$졸업자_계, y = df_annotation$취업자_
합계_계, ax = 30, ay = -30),
         margin = list(t = 50)) |>
  add_annotations(xref = 'paper', yref = 'paper',
                  text = '전문대학 경영학과 데이터', x = 0.01, y = 1.06,
                  showarrow = FALSE)
```




```
p_title_1 |>
  layout(title = list(text = '<b>졸업자 대비 취업자수</b>', x = 0.1,
                      xanchor = 'left', yanchor = 'top', size = 15),
         font = list(family = "나눔고딕", color = 'black', size = 10),
         annotations = list(text = ~paste0('<b>', df_annotation$과정구분[1], ':',
 df_annotation$학과명[1], '</b>'), x = df_annotation$졸업자_계[1], y = df_annotati
on$취업자_합계_계[1], ax = 30, ay = -30,
                            arrowhead = 3, arrowcolor = 'red', arrowsize = 1.2)
         ) |>
  add_annotations(text = ~paste0('<b>', df_annotation$과정구분[2], ':', df_annota
tion$학과명[2], '</b>'), x = df_annotation$졸업자_계[2], y = df_annotation$취업자_
합계_계[2], ax = 30, ay = -30,
                            arrowhead = 6, arrowcolor = 'blue', arrowsize = 0.8,

                  font = list(family = '나눔손글씨 펜', size = 15))
```



3.3.9. 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.

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

```
add_segments
```

```
## function (p, x = NULL, y = NULL, xend = NULL, yend = NULL, ...,
##     data = NULL, inherit = TRUE)
## {
##     if (inherit) {
##         x <- x %||% p$x$attrs[[1]][["x"]]
##         xend <- xend %||% p$x$attrs[[1]][["xend"]]
##         y <- y %||% p$x$attrs[[1]][["y"]]
##         yend <- yend %||% p$x$attrs[[1]][["yend"]]
##     }
##     if (is.null(x) || is.null(y) || is.null(xend) || is.null(yend)) {
##         stop("Must supply `x`/`y`/`xend`/`yend` attributes",
##             call. = FALSE)
##     }
##     add_trace_classed(p, x = x, y = y, xend = xend, yend = yend,
##         class = "plotly_segment", type = "scatter", mode = "lines",
##         ..., data = data, inherit = inherit)
## }
## <bytecode: 0x000000002c92a698>
## <environment: namespace:plotly>
```