

II. plotly 시각화 만들기

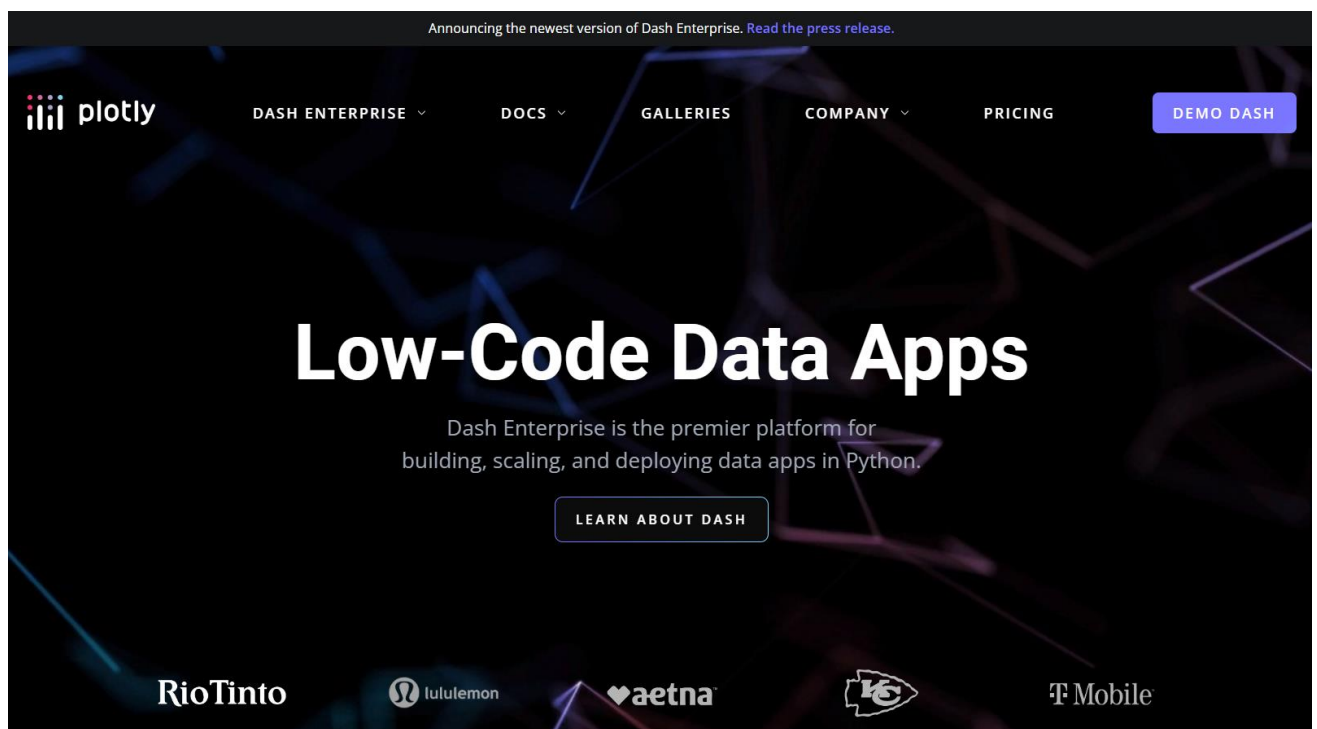
데이터 분석을 다루는 많은 교육코스나 서적에서 데이터의 시각화는 R의 경우 R base 나 `ggplot2` 패키지를 사용하여는 방법, 파이썬의 경우 `matplotlib` 이나 `seaborn` 패키지를 사용하는 방법을 위주로 설명한다. 이러한 방법들은 데이터 시각화 방법이 간단하고 그 품질이 좋은 편이기 때문에 많이 사용되고 있지만 정적(Static) 시각화라는 한계를 가진다. 정적 시각화는 대부분 이미지로 저장되며 인포그래픽(Infographic)이라고도 불리며, 일반적으로 문서나 인쇄물에 많이 사용되고 웹에 게시되는 이미지로도 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 데이터 분석 관점에 의존적일 수 밖에 없으며 시각화를 사용하는 사용자의 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 특징이다. 사용자의 의도에 따라 데이터가 동적으로 변동되어야 하기 때문에 데이터 시각화가 많이 사용되는 인쇄물 형태 매체에서 사용이 어렵고 웹을 통해 사용되어야 그 장점을 충실히 사용할 수 있다. 따라서 동적 시각화는 웹 사이트에서 제공하는 대시보드(DashBoard)의 형태로 제공되는 것이 일반적이기 때문에 동적 시각화를 위해서는 R 이나 파이썬에서 주로 사용되는 시각화 패키지가 아닌 동적 시각화 전용 패키지를 사용해야 한다. R에서는 동적 시각화를 위해 `rbokeh`, `highcharter` 등이 사용되었고 파이썬에서는 `bokeh`, `hvplot` 등을 사용되었다. 하지만 R 과 파이썬 모두에서 사용되는 `plotly` 패키지가 등장함으로써 R 과 파이썬의 동적 시각화 패키지 시장에서 매우 빠르게 사용자층을 넓혀가고 있다.

하지만 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.

1. plotly 란?

'plotly'는 캐나다 몬트리올에 본사를 두고 있는 데이터 시각화 전문 회사의 이름이다. 이 회사는 2012 년 처음 설립되었는데 데이터 전문 분석 도구와 데이터 시각화 전문 도구를 개발하여 보급한다. 'plotly'는 이 책에서 설명할 R 과 파이썬의 **plotly** 패키지 외에도 R, 파이썬, 줄리아 등의 개발도구에서 사용가능한 데이터 대시보드 플랫폼인 **dash** 를 비롯해 **dash** 플랫폼으로 개발된 웹페이지를 배포하기 위한 'Dash Enterprise', **plotly** 를 기반으로 온라인 동적 시각화를 만드는 'Chart Studio Cloud' 등의 서비스를 제공하고 있다. 이 중 가장 유명한 제품이 회사의 이름에서도 나타나듯이 **plotly** 시각화 패키지이다.



















plotly 홈페이지 화면

plotly 패키지는 R, 파이썬, Julia, Java Script, F#, MATLAB 등의 다양한 언어에서 사용이 가능하도록 각각의 언어에 바인딩되는 패키지를 개발하여 제공하고 있다. **plotly** 에서 제공하는 데이터 시각화는 산점도, 선 그래프와 같은 기본 차트(Basic Chart), 박스 플롯, 히스토그램과 같은 통계 차트(Statistical Chart), 히트맵, 삼각플롯(Ternary Plot)과 같은 과학 차트(Scientific Chart), 시계열 차트, 캔들 차트와 같은 재정 차트(Financial Chart) 등의 다양한 차트와 플롯을 제공한다.

Plotly Open Source Graphing Libraries

Interactive charts and maps for Python, R, Julia, Javascript, ggplot2, F#, MATLAB®, and Dash.

 <p>Plotly Python Open Source Graphing Library</p> <p> Star 12,301</p>	 <p>Plotly R Open Source Graphing Library</p> <p> Star 2,273</p>	 <p>Plotly Julia Open Source Graphing Library</p> <p> Star 350</p>	 <p>Plotly Javascript Open Source Graphing Library</p> <p> Star 15,074</p>
 <p>Plotly ggplot2 Open Source Graphing Library</p> <p> Star 2,273</p>	 <p>Plotly F# Open Source Graphing Library</p> <p> Star 367</p>	 <p>Plotly MATLAB® Open Source Graphing Library</p> <p> Star 321</p>	 <p>Plotly Dash Open Source Analytical App Framework</p> <p> Star 17,518</p>

Products

Dash
Consulting and Training

Pricing

Enterprise Pricing

About Us

Careers
Resources
Blog

Support

Community Support
Documentation

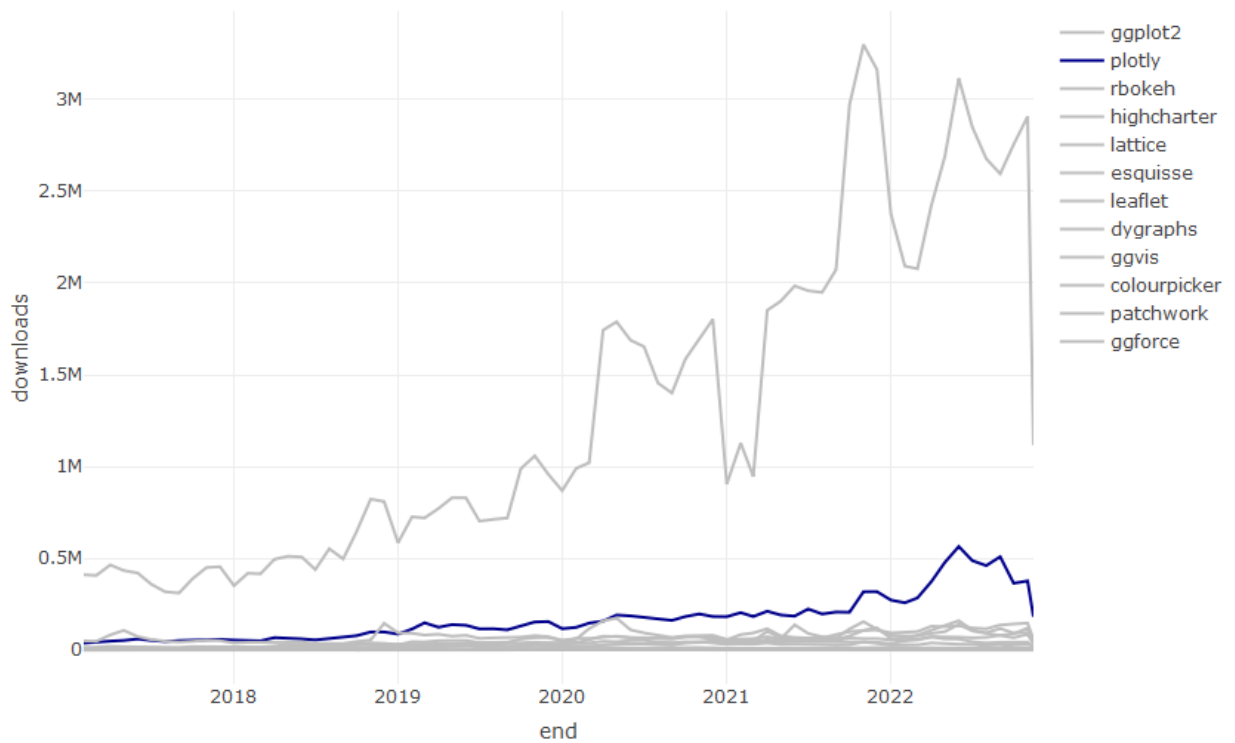
JOIN OUR MAILING LIST

Sign up to stay in the loop with all things Plotly — from Dash Club to product updates, webinars, and more!

[SUBSCRIBE](#)

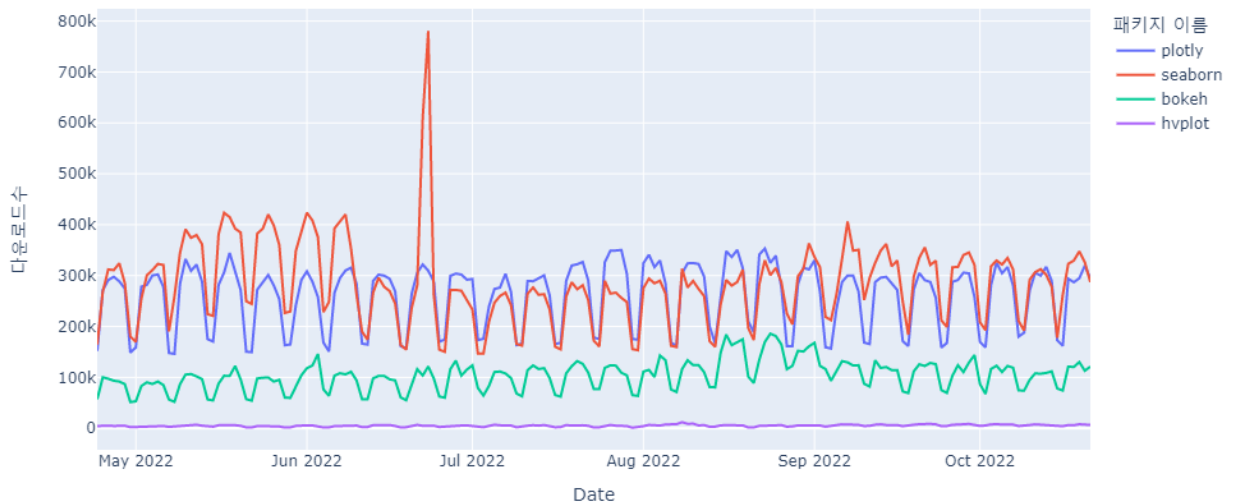
plotly 패키지 홈페이지

다음의 R 그래픽 패키지 다운로드 현황에서 보듯이 여전히 R 에서 가장 많이 사용되고 있는 그래픽 패키지는 **ggplot2** 이지만 **plotly** 는 2021 년 하반기부터 다운로드가 늘고 있고 다른 그래픽 패키지에 비해서는 압도적인 다운로드 수를 보인다.



plotly의 다운로드 수 증가는 파이썬에서도 유사한 흐름을 보인다. 다음의 그림에서도 보이듯이 파이썬에서 많이 사용되는 시각화 패키지 중 가장 다운로드수가 많은 것은 역시나 matplotlib이다. 하지만 matplotlib을 제외하면 2022년 중순까지만 해도 seaborn 패키지의 다운로드 수가 가장 많았으나 이후 plotly가 seaborn과 대등하거나 오히려 더 다운로드가 많은 날이 상당히 눈에 띈다. 하지만 파이썬에서 동적 시각화를 지원하는 bokeh나 hvplot 보다는 월등히 많은 다운로드를 보인다. 결국 파이썬에서도 동적 시각화에서는 plotly가 가장 많이 사용되는 패키지인 것이다.

최근 6개월간 패키지 다운로드수



파이썬 plotly 다운로드

2. 예제 데이터 Import 와 전처리

먼저 **plotly** 를 사용하여 시각화를 실습하는데 필요한 데이터 셋 두 가지를 전처리 하겠다.

2.1. Covid19 데이터 셋

첫번째 데이터 셋은 2020 년 1 월부터 기록된 전세계 국가의 코로나 19 발병 관련 데이터이다. 이 데이터는 Github 에서 다양한 전세계 데이터를 배포하는 'Open World in Data'에서 제공하는 'COVID-19 Dataset by Our World in Data'를 사용한다.¹ 이 데이터는 온라인으로 매일 업데이트되기 때문에 다운로드 시점에 따라 시각화 결과가 책과 다소 달라질 수 있다.²

OWID 에서 제공하는 데이터를 활용하여 4 개의 데이터 셋을 만든다. 첫 번째 데이터 셋은 OWID 에서 제공하는 원본 데이터를 가져와서 R 에 로딩하는 원본 데이터 셋으로 'df_covid19' 데이터프레임에 저장한다. 'df_covid19' 데이터 프레임은 2020 년 1 월 1 일부터 기록되어 있기

¹ https://github.com/plotly/plotly.js/blob/master/src/plot_api/plot_config.js

² 필자의 블로그에 업로드된 데이터를 활용하면 책과 동일한 결과를 얻을 수 있다.

때문에 데이터가 다소 많다. 따라서 이 데이터 중에 최근 100 일간의 데이터와 한국과 각 대륙 데이터만을 필터링한 데이터 셋을 두 번째 데이터 셋인 'df_covid19_100' 데이터프레임으로 저장한다. 세 번째 데이터 셋은 100 일간의 데이터 셋을 넓은 형태의 데이터 셋으로 변환한 'df_covid19_100_wide'로 저장한 데이터프레임이다. 네 번째는 2 년 넘게 기록된 Covid19 데이터 셋의 각종 데이터를 국가별 요약 통계치를 산출하여 저장한 'df_covid19_stat' 데이터프레임이다.

- R

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

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

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

## 1. covid19 원본 데이터 셋 로딩
## covid19 데이터 로딩(파일을 다운로드 받은 경우)
# df_covid19 <- read_csv(file = "데이터저장경로/owid-covid-data.csv",
#                         col_types = cols(Date = col_date(format = "%Y-%m-%d"))
#                         )
## covid19 데이터 로딩(온라인에서 바로 로딩할 경우)
df_covid19 <- read_csv(file = "https://covid.ourworldindata.org/data/owid-covid-data.csv",
                      col_types = cols(Date = col_date(format = "%Y-%m-%d"))
                      )

## 2. 전체 데이터셋 중 최근 100 일간의 데이터를 필터링한 df_covid19_100 생성
df_covid19_100 <- df_covid19 |>
  ## 한국 데이터와 각 대륙별 데이터만을 필터링
  filter(iso_code %in% c('KOR', 'OWID_ASI', 'OWID_EUR', 'OWID_OCE', 'OWID_NAM', 'OWID_SAM', 'OWID_AFR')) |>
  ## 읽은 데이터의 마지막 데이터에서 100 일전 데이터까지 필터링
  filter(date >= max(date) - 100) |>
  ## 국가명을 한글로 변환
  mutate(location = case_when(
    location == 'South Korea' ~ '한국',
    location == 'Asia' ~ '아시아',
    location == 'Europe' ~ '유럽',
```

```

location == 'Oceania' ~ '오세아니아',
location == 'North America' ~ '북미',
location == 'South America' ~ '남미',
location == 'Africa' ~ '아프리카')) |>
## 국가 이름의 순서를 설정
mutate(location = fct_relevel(location, '한국', '아시아', '유럽', '북미', '남미', '아프리카', '오세아니아')) |>
## 날짜로 정렬
arrange(date)

## 3. df_covid19_100 을 한국과 각 대륙별로 배치한 넓은 형태의 데이터프레임으로 변환
df_covid19_100_wide <- df_covid19_100 |>
## 날짜, 국가명, 확진자와, 백신접종완료자 데이터만 선택
select(date, location, new_cases, people_fully_vaccinated_per_hundred) |>
## 열 이름을 적절히 변경
rename('date' = 'date', '확진자' = 'new_cases', '백신접종완료자' = 'people_fully_vaccinated_per_hundred') |>
## 넓은 형태의 데이터로 변환
pivot_wider(id_cols = date, names_from = location,
            values_from = c('확진자', '백신접종완료자')) |>
## 날짜로 정렬
arrange(date)

## 4. covid19 데이터를 국가별로 요약한 df_covid19_stat 생성
df_covid19_stat <- df_covid19 |>
group_by(iso_code, continent, location) |>
summarise(인구수 = max(population, na.rm = T),
          인당 GDP = max(gdp_per_capita, na.rm = T),
          전체확진자수 = sum(new_cases, na.rm = T),
          전체사망자수 = sum(new_deaths, na.rm = T),
          십만명당중환자실 = last(icu_patients_per_million),
          재생산지수 = last(reproduction_rate),
          봉쇄지수 = max(stringency_index),
          전체검사자수 = max(total_tests, na.rm = T),
          신규검사자수 = sum(new_tests, na.rm = T),
          전체백신접종자수 = max(total_vaccinations, na.rm = T),
          백신접종자완료자수 = max(people_fully_vaccinated, na.rm = T),
          부스터접종자수 = max(total_boosters, na.rm = T),
          인구백명당백신접종완료률 = max(people_fully_vaccinated_per_hundred, na.rm = T),
          인구백명당부스터접종자수 = max(total_boosters_per_hundred, na.rm = T)
          ) |>
ungroup() |>
mutate(십만명당사망자수 = round(전체사망자수 / 인구수 * 100000, 5),
       백신접종완료률 = 백신접종자완료자수 / 인구수)

```

```
## [1] "2022-08-04"
```

- python

```
#####
## python 코드
## Covid19 데이터 셋
import pandas as pd
from datetime import datetime, timedelta
from pandas.api.types import CategoricalDtype
from matplotlib import pyplot as plt
import plotly.graph_objects as go

df_covid19 = pd.read_csv("https://covid.ourworldindata.org/data/owid-covid-data.csv")

df_covid19['date'] = pd.to_datetime(df_covid19['date'], format="%Y-%m-%d")

df_covid19_100 = df_covid19[(df_covid19['iso_code'].isin(['KOR', 'OWID_ASI', 'OWID_EUR',
'OWID_OCE', 'OWID_NAM', 'OWID_SAM', 'OWID_AFR'])) & (df_covid19['date'] >=
(max(df_covid19['date']) - timedelta(days = 100)))]

df_covid19_100.loc[df_covid19_100['location'] == 'South Korea', "location"] = '한국'

df_covid19_100.loc[df_covid19_100['location'] == 'Asia', "location"] = '아시아'
df_covid19_100.loc[df_covid19_100['location'] == 'Europe', "location"] = '유럽'
df_covid19_100.loc[df_covid19_100['location'] == 'Oceania', "location"] = '오세아니아'
df_covid19_100.loc[df_covid19_100['location'] == 'North America', "location"] = '북미'
df_covid19_100.loc[df_covid19_100['location'] == 'South America', "location"] = '남미'
df_covid19_100.loc[df_covid19_100['location'] == 'Africa', "location"] = '아프리카'

ord = CategoricalDtype(categories = ['한국', '아시아', '유럽', '북미', '남미', '아프리카',
'오세아니아'], ordered = True)

df_covid19_100['location'] = df_covid19_100['location'].astype(ord)

df_covid19_100 = df_covid19_100.sort_values(by = 'date')

df_covid19_100_wide = df_covid19_100.loc[:,['date', 'location', 'new_cases',
'people_fully_vaccinated_per_hundred']].rename(columns={'new_cases': '확진자',
'people_fully_vaccinated_per_hundred': '백신접종완료자'})

df_covid19_100_wide = df_covid19_100_wide.pivot(index='date', columns='location',
values=['확진자', '백신접종완료자']).sort_values(by = 'date')

df_covid19_100_wide.columns = ['확진자_한국', '확진자_아시아', '확진자_유럽', '확진자_북미',
'확진자_남미', '확진자_아프리카', '확진자_오세아니아',
'백신접종완료자_한국', '백신접종완료자_아시아',
```



```

'백신접종완료자_유럽', '백신접종완료자_북미', '백신접종완료자_남미',
'백신접종완료자_아프리카', '백신접종완료자_오세아니아']

df_covid19_stat = df_covid19.groupby(['iso_code', 'continent', 'location']).agg(
    인구수 = ('population', 'max'),
    인당 GDP = ('gdp_per_capita', 'max'),
    전체확진자수 = ('new_cases', 'sum'),
    전체사망자수 = ('new_deaths', 'sum'),
    십만명당중환자실 = ('icu_patients_per_million', 'last'),
    재생산지수 = ('reproduction_rate', 'last'),
    봉쇄지수 = ('stringency_index', 'max'),
    전체검사자수 = ('total_tests', 'max'),
    신규검사자수 = ('new_tests', 'sum'),
    전체백신접종자수 = ('total_vaccinations', 'max'),
    백신접종자완료자수 = ('people_fully_vaccinated', 'max'),
    부스터접종자수 = ('total_boosters', 'max'),
    인구백명당백신접종완료률 = ('people_fully_vaccinated_per_hundred', 'max'),
    인구백명당부스터접종자수 = ('total_boosters_per_hundred', 'max')
)

df_covid19_stat['십만명당사망자수'] = round(df_covid19_stat['전체사망자수'] /
df_covid19_stat['인구수'] * 100000, 5)

df_covid19_stat['백신접종완료률'] = df_covid19_stat['백신접종자완료자수'] /
df_covid19_stat['인구수']

```

2.2. 대학 학과 취업률 데이터 셋

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

³ 해당 데이터는 교육통계 서비스

홈페이지 https://kess.kedi.re.kr/contents/dataset?itemCode=04&menuId=m_02_04_03_02&tabId=m3 에서 다운로드를 받거나 필자의 블로그(2stndard.tistory.com)에서 다운로드 받을 수 있다.

취업률 데이터 셋은 다음과 같이 데이터를 로딩하고 전처리한다.

- R

```
## R 코드

df_취업률 <- read_excel('d:/R/data/2020 년 학과별 고등교육기관 취업통계.xlsx',
                        ## '학과별' 시트의 데이터를 불러오는데,
                        sheet = '학과별',
                        ## 앞의 13 행을 제외하고
                        skip = 13,
                        ## 첫번째 행은 열 이름으로 설정
                        col_names = TRUE,
                        ## 열의 타입을 설정, 처음 9 개는 문자형으로 다음 79 개는 수치형으로 설정
                        col_types = c(rep('text', 9), rep('numeric', 79)))

## df_취업률에서 첫번째부터 9 번째까지의 열과 '계'로 끝나는 열을 선택하여 다시 df_취업률에 저장
df_취업률 <- df_취업률 |>
  select(1:9, ends_with('계'), '입대자')

## df_취업률에서 졸업자가 500 명 이하인 학과 2000 개 샘플링
df_취업률_2000 <- df_취업률 |>
  filter(졸업자_계 < 500) |>
  mutate(id = row_number()) |>
  filter(row_number() %in% seq(from = 1, to = nrow(df_취업률), by = 4))

## 열 이름을 적절히 설정
names(df_취업률_2000)[10:12] <- c('졸업자수', '취업률', '취업자수')
```

- python

```
#####
## python 코드
## 대학 학과 취업률 데이터 셋

df_취업률 = pd.read_excel("d:/R/data/2020 년 학과별 고등교육기관 취업통계.xlsx",
                          sheet_name = '학과별',
                          skiprows=(13),
                          header = 0)

df_취업률 = pd.concat([df_취업률.iloc[:, 0:8],
                      df_취업률.loc[:, df_취업률.columns.str.endswith('계')],
                      df_취업률.loc[:, '입대자']],
                      axis = 1
                      )

df_취업률_2000 = df_취업률.loc[(df_취업률['졸업자_계'] < 500)]
```

```
df_취업률_2000 = df_취업률_2000.iloc[range(0, len(df_취업률_2000.index) , 4)]

df_취업률_2000 = df_취업률_2000.rename(columns = {'졸업자_계':'졸업자수', '취업률_계':'취업률',
'취업자_합계_계':'취업자수'})
```

3. plotly 의 구조

plotly 객체는 plotly.js 에서 정의된 JSON 스키마로 저장된다. 이 스키마는 트리 형태로 구성되어 있는데 각 노드는 속성(attribute)로 불리는 값을 가지게 되고 이들 속성들이 모여서 전체 그림(Figure)를 구성한다.

plotly 객체 트리의 루트 노드에 바로 아래인 최고 레벨 속성은 'data', 'layout', 'frame'의 세 가지 속성이다. 이 세 가지 속성의 세부 속성들이 설정되어 시각화를 구성한다. 이들 속성들은 다시 부모 속성과 자식 속성으로 구성하는데 이 구성 방법은 R 과 python 이 다소 다르다.

R 의 경우 속성이름에 '='을 사용해 속성 값을 할당하고 같은 부모를 가지는 자식 속성들은 R 의 list 데이터 구조로 구성하여 표현한다. 예를 들어 `layout(title = list(text = '타이틀 제목'))`이라는 코드는 'layout' 루트 노드에 하위 노드인 'title' 속성의 하위 노드인 text 속성을 20 으로 설정하는 코드이다.

python 의 경우는 하위 속성값을 python 의 딕셔너리 데이터 구조를 사용해 구성한다.

딕셔너리는 {}를 사용할 수도 있고 dict()를 사용할 수도 있다. {}를 사용하면 속성명과 속성값을 :을 사용하여 할당하고 dict()를 사용하면 속성명과 속성값은 =을 사용하여 설정해 준다.

R 에서 list 로 python 에서 딕셔너리로 설정한 값들을 직접 접근하기 위해서는 '.'을 사용해 속성값의 구조적 전체 경로(Path)를 통해 접근할 수 있다. 예를 들어 시각화 전체 제목의 색상에 접근하려면 `layout.title.font.color` 를 사용하여 접근할 수 있다.

plotly 로 시각화를 만들기 위해서는 먼저 `plot_ly()`나 `plotly.graph_object.Figure()`를 사용하여 초기화하고 'data', 'layout', 'frame'의 세 가지 루트 속성으로부터 파생된 하위 속성들을 list 나 dict 형태로 구성함으로써 시각화를 생성한다.

3.1. plotly 초기화

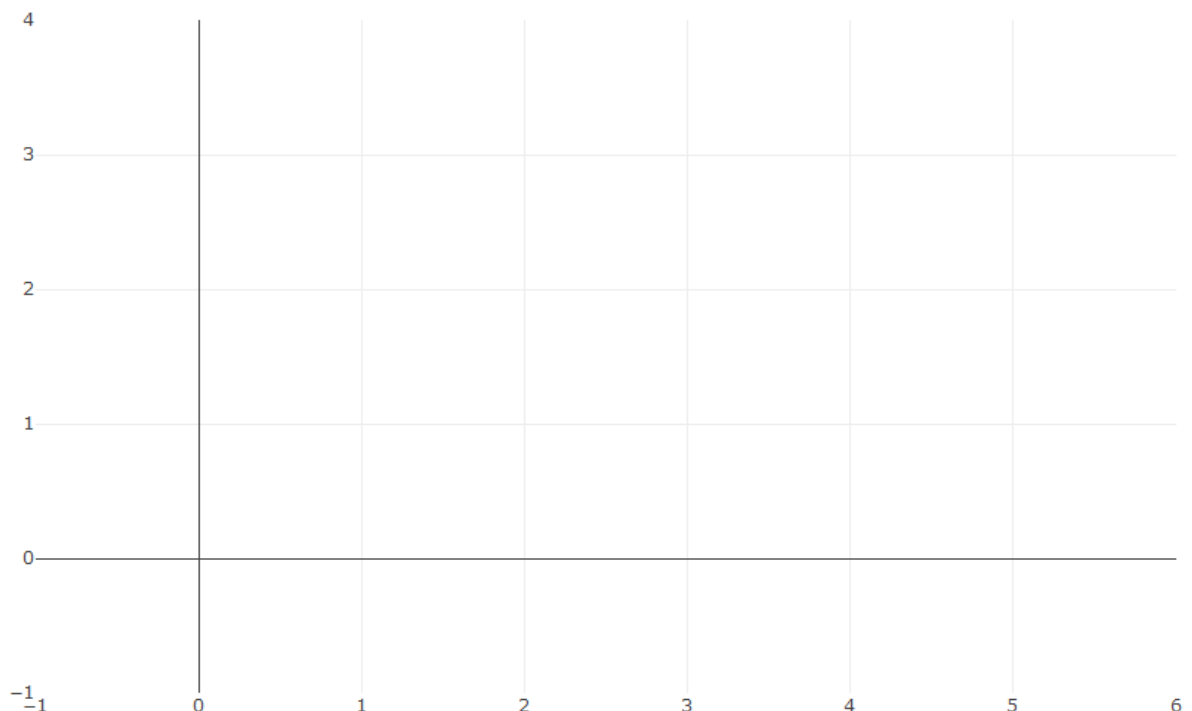
`plotly` 를 사용하기 위해서는 가장 먼저 해야하는 것이 `plotly` 객체의 초기화이다. 이는 R 과 python 에서 모두 수행해야 하는 과정이다. `plotly` 객체를 초기화하면 `plotly.js` 에서 사용될 수 있는 JSON 형태의 객체가 생성된다. 이 객체에 다양한 시각화 속성들을 추가함으로써 전체 시각화를 완성한다.

- R

R 에서 `plotly` 를 초기화하기 위해 `plot_ly()`를 사용한다. `plot_ly()`는 특별한 매개변수 없이 사용이 가능하지만 일반적으로 사용할 데이터프레임을 바인딩하는 경우가 많다.

`tidyverse` 에서 제공하는 `%>%`나 R base 에서 제공하는 `|>`를 사용하여 `plot_ly()`에 사용할 데이터프레임을 설정하거나 `plot_ly()`의 매개변수로 데이터프레임을 전달하면 초기화된 `plotly` 객체가 생성된다.

```
## R 에서 plotly 객체 초기화
## plot_ly(df_covid19_100)와 동일
df_covid19_100 |>
  plot_ly()
```



실행결과 2- 1. R 의 `plot_ly()` 를 사용한 `plotly` 초기화

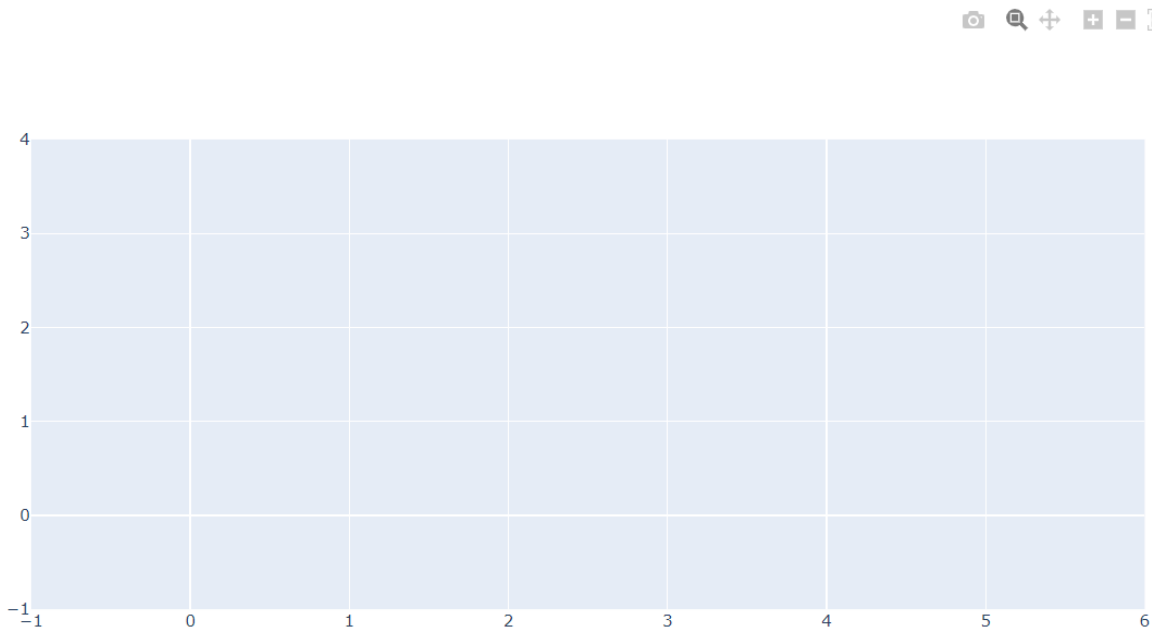
- python

python 에서 **plotly** 를 초기화하는 방법은 `plotly.graph_object` 을 사용하는 것과 `plotly.express` 를 사용하는 두 가지 방법을 제공한다. **plotly** 에서 제공하는 모든 기능을 사용하기 위해서는 `plotly.graph_object` 를 사용해야 하지만 사용의 편의성이 떨어진다. 반면 `plotly.express` 는 사용이 매우 간편하지만 **plotly** 에서 제공하는 모든 기능을 사용할 수 없다. 따라서 상황에 따라 잘 구분해서 사용해야 한다. 이번 장에서는 `plotly.graph_object` 를 사용하는 방법을 사용하고 다음 장부터는 `plotly.express` 위주로 사용하겠다.

python 에서 `plotly.graph_object` 를 사용하여 **plotly** 를 초기화하기 위해서는 `plotly.graph_objects.Figure()`를 사용한다. 앞서 R 에서는 **plotly** 를 초기화할때 사용하는 데이터프레임을 바인딩했지만 python 에서는 초기화때 데이터프레임을 바인딩하지 않는다.

```
## python 에서 plotly 객체 초기화
import plotly.graph_objects as go

go.Figure()
```



python 의 Figure() 를 사용한 plotly 초기화

3.2. 'data' 속성

초기화된 **plotly** 구조에서 사용하는 첫 번째 레벨 속성인 'data' 속성은 그 하위 속성들이 많은데, 이 중 가장 중요한 하위 속성이 'trace' 속성이다. 'trace' 속성은 **plotly** 로 시각화할 수

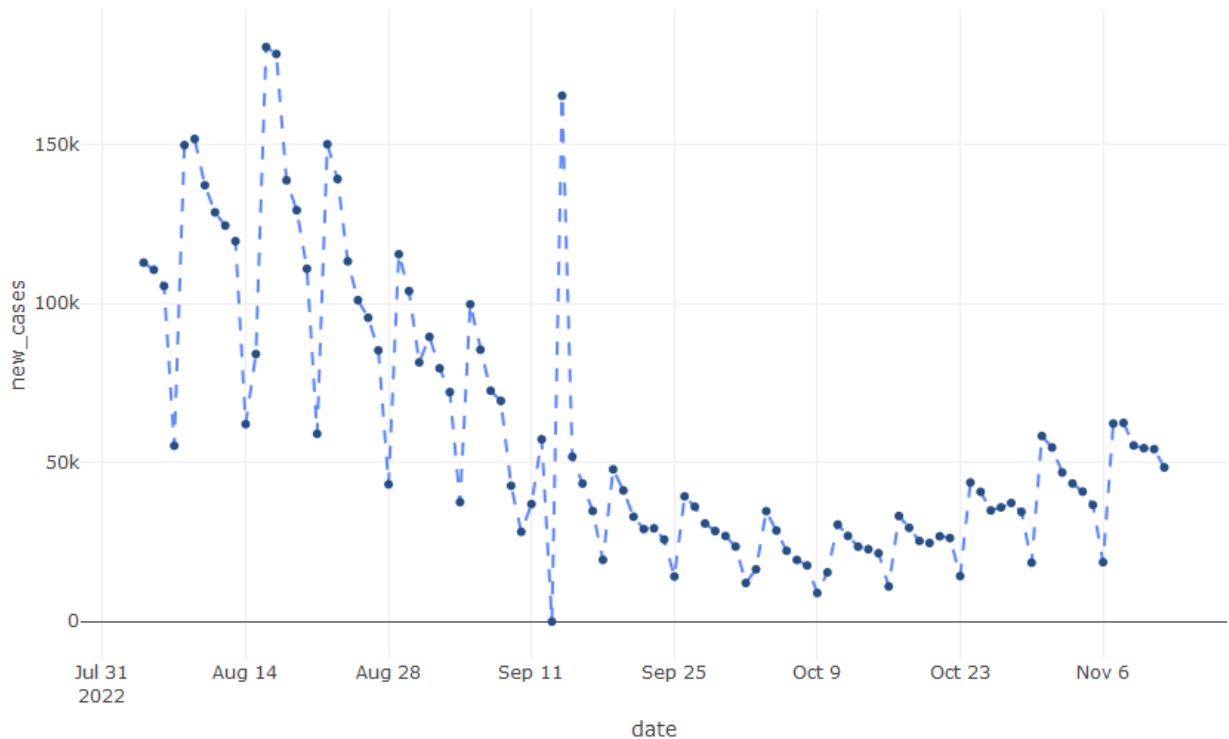
있는 그래픽적 데이터 표현 방법을 지칭한다. `plotly`에서는 'scatter', 'pie', 'bar' 등의 40 개가 넘는 트레이스를 제공한다.

40 여가지 트레이스 중에 사용하고자 하는 트레이스를 설정하기 위해서 'type' 속성을 사용한다. 예를 들어 R 과 python 모두 `add_trace()`를 사용하여 트레이스를 계속 추가할 수 있는데, `add_trace()`를 사용하기 위해서는 반드시 'type' 속성으로 트레이스 종류를 설정해야 한다. 하지만 'type'을 설정하지 않더라도 X 축과 Y 축에 바인딩된 변수들을 계산하여 자동적으로 설정하기도 하고, 각각의 트레이스에 특화된 개별 함수를 사용한다면(예를 들어 `add_lines()`나 `px.line()` 등) 'type' 속성을 사용할 필요는 없다.

- R

R에서는 'data' 속성을 바로 설정할 수 있는 방법을 제공하지 않는다. 하지만 `plotly` 패키지의 `plot_ly()`를 사용하면 이와 동일한 효과를 낼 수 있다. `plot_ly()`의 매개 변수로 'data'의 세부 속성들을 설정할 수 있는데 세부 속성에 다시 하위 세부 속성들이 포함될 경우 R의 기본 데이터 타입인 'list'를 사용하여 묶어서 설정한다. 다음의 코드는 `plot_ly()`를 사용하여 'data' 속성의 하부 속성인 'trace'를 'scatter', 'mode'를 'markers+lines'(산점도와 선그래프), 'marker'(점) 속성을 list로 묶어 'color' 속성을 설정하고 'line'(선) 속성을 list로 묶어 'color'와 'dash' 속성을 설정한 시각화를 보이고 있다.

```
## 긴 형태의 100 일 코로나 19 데이터에서
df_covid19_100 |>
## 한국 데이터만을 필터링
filter(iso_code == 'KOR') |>
## X 축에 date, Y 축에 new_cases 를 매핑하여 plot_ly()로 시각화 생성
plot_ly(type = 'scatter', x = ~date, y = ~new_cases,
        mode = 'markers+lines',
        marker = list(color = '#264E86'),
        line = list(color = '#5E88FC',
                    dash = 'dash'
        )
    )
```



실행결과 2- 2. R 의 `plot_ly` 를 사용한 `plotly` 시각화

- python

반면, python 에서는 `plotly.graph_object` 의 `Figure()` 를 사용한다. 위의 R 과 같은 시각화를 만드는 코드는 다음과 같다. `go.Figure()` 에 'data' 속성을 설정하기 위해서는 각각의 'trace'의 속성으로 구성된 딕셔너리의 리스트로 설정한다. 하지만 만약 단 하나의 'trace' 딕셔너리만 있다면 리스트(`[]`)로 묶어주지 않아도 무방하다. 다음의 코드는 위의 R 로 만든 시각화와 동일한 시각화를 만든다. `go.Figure()` 의 'data' 속성에 'type', 'mode', 'x', 'y', 'marker', 'line' 속성을 가지는 트레이스 딕셔너리를 설정하고 이 중 'marker'와 'line' 속성은 다시 세부속성을 가지기 때문에 이 세부 속성을 설정하기 위한 딕셔너리를 설정한다. 여기에는 하나의 'trace' 뿐이기 때문에 `[]` 를 사용하여 리스트로 묶지 않아도 되지만 묶어주는게 코드의 일관성을 위해 좋다.

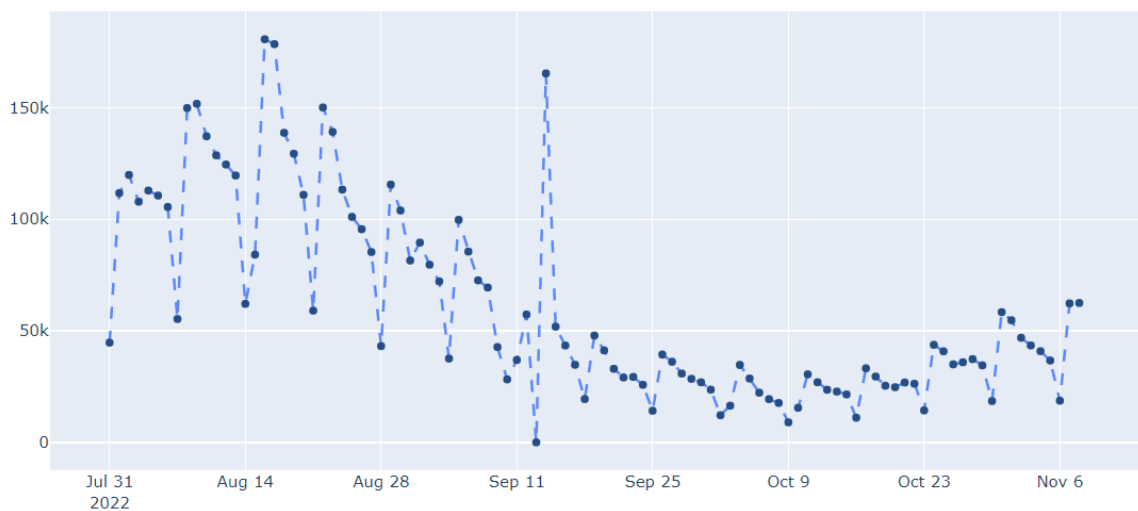
```
import plotly.graph_objects as go
import plotly.offline as py

fig = go.Figure(
    data = [
        {
            'type' : 'scatter',
```

```

    'mode' : 'markers+lines',
    'x' : df_covid19_100.loc[df_covid19_100['iso_code'] == 'KOR', 'date'],
    'y' : df_covid19_100.loc[df_covid19_100['iso_code'] == 'KOR', 'new_cases'],
    'marker' : {
        'color' : '#264E86'
    },
    'line' : {
        'color' : '#5E88FC',
        'dash' : 'dash'
    }
}
]
)

```



python 의 data 속성을 사용한 plotly 시각화

3.3. 'layout' 속성

'layout' 속성은 데이터를 표현하는 'trace'와 관련되지 않는 시각화의 나머지 속성들을 정의하는 설정들을 표현한다. 'layout' 에서 설정할 수 있는 속성은 시각화의 차원 축(2 차원, 3 차원), 여백, 제목, 축, 범례, 컬러바(색 범례), 주석 등 이다.

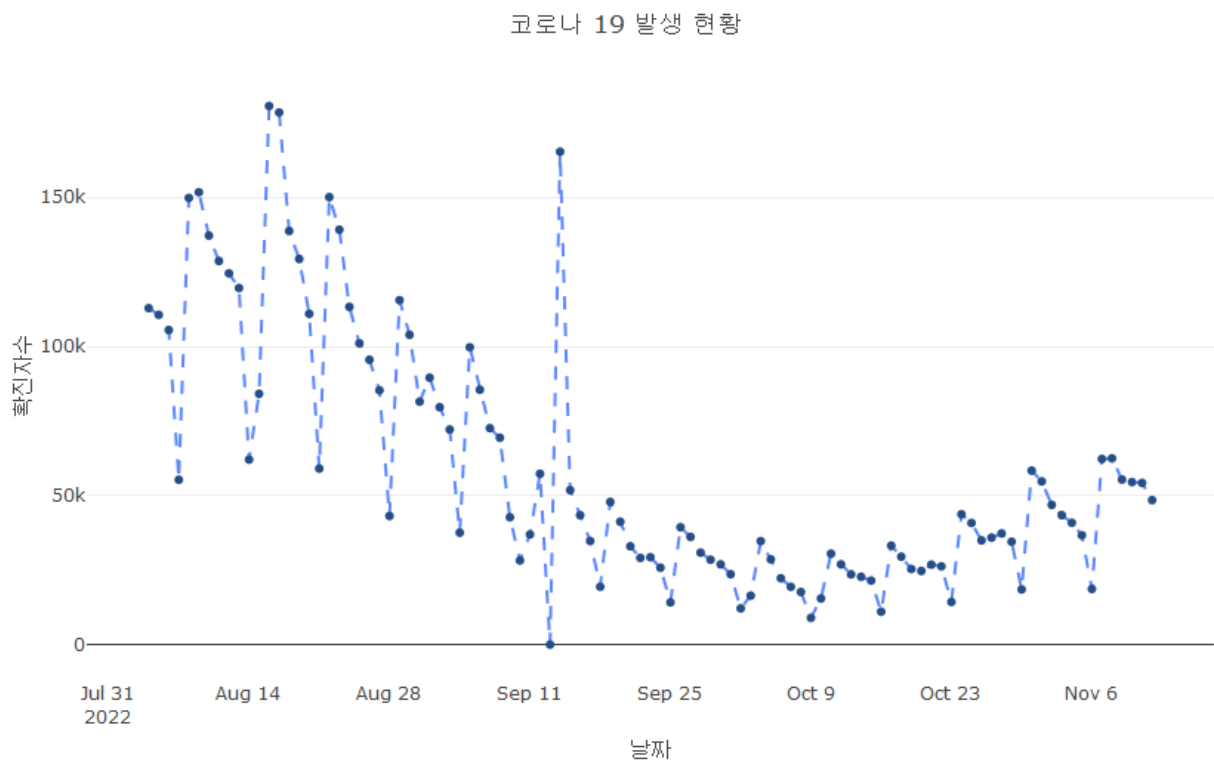
- R

R 에서 'layout' 속성을 설정하기 위해서는 `layout()`을 사용한다. 다만 `layout()`을 사용하기 위해서는 최소한 하나 이상의 'trace'가 설정되어야 한다. 다음의 코드는 앞서 그린

시각화에 'title' 속성으로 시각화 제목, 'xaxis' 속성으로 X 축 설정, 'yaxis' 속성으로 Y 축 설정, 'margin' 속성으로 여백 설정을 하고 있다. 이 중 'xaxis'와 'yaxis'속성은 세부 속성이 있기 때문에 다시 list 로 묶어서 설정하였다.

```
margins_R <- list(t = 50, b = 25, l = 25, r = 25)

## 긴 형태의 100 일 코로나 19 데이터에서
df_covid19_100 |>
## 한국 데이터만을 필터링
filter(iso_code == 'KOR') |>
## X 축에 date, Y 축에 new_cases 를 매핑하여 plot_ly()로 시각화 생성
plot_ly(type = 'scatter', x = ~date, y = ~new_cases,
        mode = 'markers+lines',
        marker = list(color = '#264E86'),
        line = list(color = '#5E88FC',
                    dash = 'dash'
                  )
        ) |>
layout(
  title = "코로나 19 발생 현황",
  xaxis = list(
    title = "날짜",
    showgrid = F),
  yaxis = list(title = "확진자수",
              margin = margins_R
              )
)
```



실행결과 2-3. R 의 layout 속성을 사용한 plotly 시각화

- python

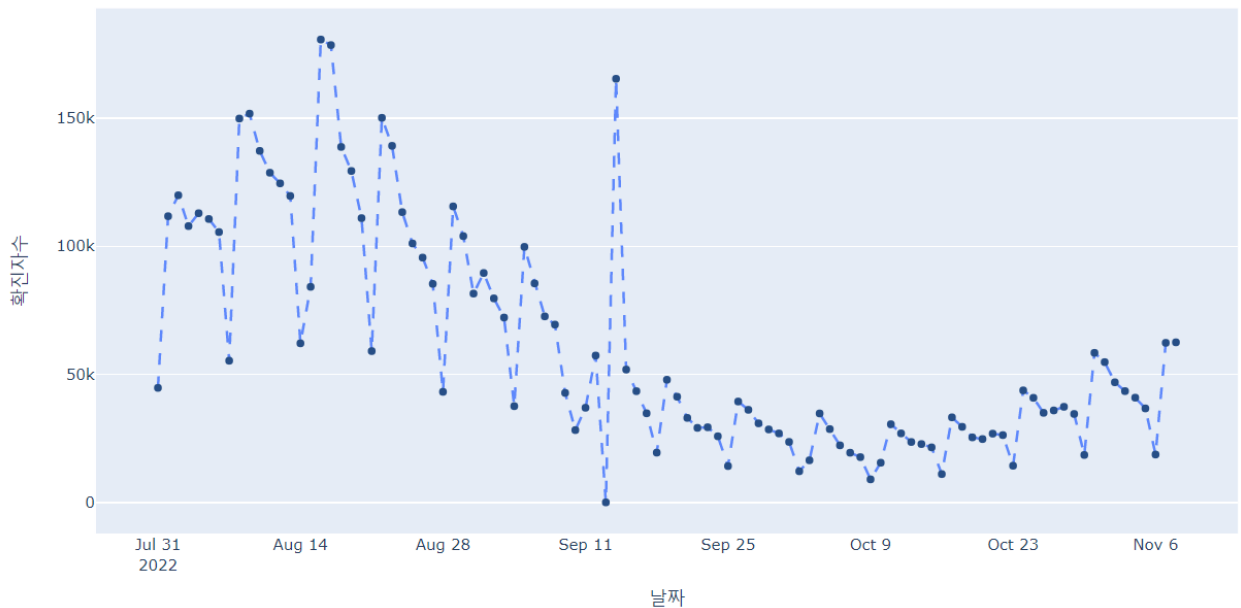
python 에서 'layout' 속성의 설정은 'data' 속성과 같이 `Figure()`에 속성들로 구성된 딕셔너리를 'layout'에 설정한다.

```
margins_python = {'t' : 50, 'b' : 25, 'l' : 25, 'r' : 25}

fig = go.Figure(
    data = [
        {
            'type' : 'scatter',
            'mode' : 'markers+lines',
            'x' : df_covid19_100.loc[df_covid19_100['iso_code'] == 'KOR', 'date'],
            'y' : df_covid19_100.loc[df_covid19_100['iso_code'] == 'KOR', 'new_cases'],
            'marker' : {
                'color' : '#264E86'
            },
            'line' : {
                'color' : '#5E88FC',
                'dash' : 'dash'
            }
        }
    ],
    layout = {
        'title' : "코로나 19 발생 현황",
        'xaxis' : {
            'title' : "날짜",
            'showgrid' : False},
        'yaxis' : {'title' : "확진자수"},
        'margin' : margins_python
    }
)

fig.show()
```

코로나 19 발생 현황



python 의 layout 속성을 사용한 plotly 시각화

3.4. 'frame' 속성

'frame' 속성은 `plotly`의 애니메이션 기능과 관련된 속성 값을 설정하는 노드이다. 이 책에서는 다루지 않겠다.

3.5. 'plotly' 구조 확인

앞서 언급했다시피 `plotly`는 JSON 형태의 데이터 타입으로 구성된다. 따라서 `plotly`로 만들어진 데이터는 그래프의 형태로 시각화 할수도 있지만 `plotly` 데이터의 구조를 직접 볼 수도 있다. 이 구조를 확인하면 `plotly`의 속성 설정을 이해하는데 도움을 받을 수 있다.

- R

R에서 `plotly`의 구조를 확인하기 위해서는 `plotly_json()`을 사용한다.

```
df_covid19_100 |>
  ## 한국 데이터만을 필터링
  filter(iso_code == 'KOR') |>
  ## X축에 date, Y축에 new_cases를 매핑하여 plot_ly()로 시각화 생성
  plot_ly(type = 'scatter', x = ~date, y = ~new_cases,
    mode = 'markers+lines',
    marker = list(color = '#264E86'),
    line = list(color = '#5E88FC',
      dash = 'dash')
```

```

    )
  ) |>
  plotly_json()

## {
## "visdat": {
## "211474e250d": ["function () ", "plotlyVisDat"]
## },
## "cur_data": "211474e250d",
## "attrs": {
## "211474e250d": {
## "x": {},
## "y": {},
## "mode": "markers+lines",
## "marker": {
## "color": "#264E86"
## },
## "line": {
## "color": "#5E88FC",
## "dash": "dash"
## },
## "alpha_stroke": 1,
## "sizes": [10, 100],
## "spans": [1, 20],
## "type": "scatter"
## }
## },
## "layout": {
## "margin": {
## "b": 40,
## "l": 60,
## "t": 25,
## "r": 10
## },
## "xaxis": {
## "domain": [0, 1],
## "automargin": true,
## "title": "date"
## },
## "yaxis": {
## "domain": [0, 1],
## "automargin": true,
## "title": "new_cases"
## },
## "hovermode": "closest",
## "showlegend": false
## },
## "source": "A",
## "config": {
## "modeBarButtonsToAdd": ["hoverclosest", "hovercompare"],
## "showSendToCloud": false
## },
## "data": [
## {
## "x": ["2022-08-04", "2022-08-05", "2022-08-06", "2022-08-07", "2022-08-08", "2022-08-09", "2022-08-10",
"2022-08-11", "2022-08-12", "2022-08-13", "2022-08-14", "2022-08-15", "2022-08-16", "2022-08-17", "2022-08-18", "2022-08-19", "2022-08-20", "2022-08-21", "2022-08-22", "2022-08-23", "2022-08-24", "2022-08-25", "2022-08-26", "2022-08-27", "2022-08-28", "2022-08-29", "2022-08-30", "2022-08-31", "2022-09-01", "2022-09-02", "2022-09-03", "2022-09-04", "2022-09-05", "2022-09-06", "2022-09-07", "2022-09-08", "2022-09-09", "2022-09-10", "2022-09-11", "2022-09-12", "2022-09-13", "2022-09-14", "2022-09-15", "2022-09-16", "2022-09-17", "2022-09-18", "2022-09-19", "2022-09-20", "2022-09-21", "2022-09-22", "2022-09-23", "2022-09-24", "2022-09-25", "2022-09-26", "2022-09-27", "2022-09-28", "2022-09-29", "2022-09-30", "2022-10-01", "2022-10-02", "2022-10-03", "2022-10-04", "2022-10-05", "2022-10-06", "2022-10-07", "2022-10-08", "2022-10-09", "2022-10-10", "2022-10-11", "2022-10-12", "2022-10-13", "2022-10-14", "2022-10-15", "2022-10-16", "2022-10-17", "2022-10-18", "2022-10-19", "2022-10-20", "2022-10-21", "2022-10-22", "2022-10-23", "2022-10-24", "2022-10-25", "2022-10-26", "2022-10-27", "2022-10-28", "2022-10-29", "2022-10-30", "2022-10-31", "2022-11-01", "2022-11-02", "2022-11-03", "2022-11-04", "2022-11-05", "2022-11-06", "2022-11-07", "2022-11-08", "2022-11-09", "2022-11-10", "2022-11-11", "2022-11-12", "2022-11-13", "2022-11-14", "2022-11-15", "2022-11-16", "2022-11-17", "2022-11-18", "2022-11-19", "2022-11-20", "2022-11-21", "2022-11-22", "2022-11-23", "2022-11-24", "2022-11-25", "2022-11-26", "2022-11-27", "2022-11-28", "2022-11-29", "2022-11-30", "2022-12-01", "2022-12-02", "2022-12-03", "2022-12-04", "2022-12-05", "2022-12-06", "2022-12-07", "2022-12-08", "2022-12-09", "2022-12-10", "2022-12-11", "2022-12-12", "2022-12-13", "2022-12-14", "2022-12-15", "2022-12-16", "2022-12-17", "2022-12-18", "2022-12-19", "2022-12-20", "2022-12-21", "2022-12-22", "2022-12-23", "2022-12-24", "2022-12-25", "2022-12-26", "2022-12-27", "2022-12-28", "2022-12-29", "2022-12-30", "2022-12-31"],

```

```

18", "2022-08-19", "2022-08-20", "2022-08-21", "2022-08-22", "2022-08-23", "2022-08-24", "2022-08-25",
"2022-08-26", "2022-08-27", "2022-08-28", "2022-08-29", "2022-08-30", "2022-08-31", "2022-09-01", "2022-09-
02", "2022-09-03", "2022-09-04", "2022-09-05", "2022-09-06", "2022-09-07", "2022-09-08", "2022-09-09",
"2022-09-10", "2022-09-11", "2022-09-12", "2022-09-13", "2022-09-14", "2022-09-15", "2022-09-16", "2022-09-
17", "2022-09-18", "2022-09-19", "2022-09-20", "2022-09-21", "2022-09-22", "2022-09-23", "2022-09-24",
"2022-09-25", "2022-09-26", "2022-09-27", "2022-09-28", "2022-09-29", "2022-09-30", "2022-10-01", "2022-10-
02", "2022-10-03", "2022-10-04", "2022-10-05", "2022-10-06", "2022-10-07", "2022-10-08", "2022-10-09",
"2022-10-10", "2022-10-11", "2022-10-12", "2022-10-13", "2022-10-14", "2022-10-15", "2022-10-16", "2022-10-
17", "2022-10-18", "2022-10-19", "2022-10-20", "2022-10-21", "2022-10-22", "2022-10-23", "2022-10-24",
"2022-10-25", "2022-10-26", "2022-10-27", "2022-10-28", "2022-10-29", "2022-10-30", "2022-10-31", "2022-11-
01", "2022-11-02", "2022-11-03", "2022-11-04", "2022-11-05", "2022-11-06", "2022-11-07", "2022-11-08",
"2022-11-09", "2022-11-10", "2022-11-11", "2022-11-12"],
##   "y": [112857, 110610, 105507, 55292, 149819, 151734, 137196, 128671, 124515, 119603, 62078, 84128,
180652, 178480, 138741, 129350, 110944, 59046, 150098, 139165, 113281, 101064, 95538, 85295, 43142,
115519, 103919, 81499, 89528, 79623, 72144, 37548, 99737, 85484, 72599, 69389, 42724, 28214, 36938, 57309,
0, 165336, 51832, 43400, 34764, 19407, 47864, 41231, 32972, 29081, 29315, 25792, 14168, 39367, 36126,
30846, 28466, 26913, 23597, 12150, 16423, 34710, 28603, 22259, 19379, 17654, 8981, 15476, 30503, 26928,
23562, 22757, 21469, 11040, 33190, 29482, 25369, 24709, 26823, 26256, 14302, 43714, 40805, 34950, 35887,
37296, 34511, 18510, 58358, 54740, 46870, 43424, 40863, 36675, 18671, 62273, 62472, 55365, 54519, 54225,
48465],
##   "mode": "markers+lines",
##   "marker": {
##     "color": "#264E86",
##     "line": {
##       "color": "rgba(31,119,180,1)"
##     }
##   },
##   "line": {
##     "color": "#5E88FC",
##     "dash": "dash"
##   },
##   "type": "scatter",
##   "error_y": {
##     "color": "rgba(31,119,180,1)"
##   },
##   "error_x": {
##     "color": "rgba(31,119,180,1)"
##   },
##   "xaxis": "x",
##   "yaxis": "y",
##   "frame": null
## }
## ],
## "highlight": {
##   "on": "plotly_click",
##   "persistent": false,
##   "dynamic": false,
##   "selectize": false,
##   "opacityDim": 0.2,
##   "selected": {
##     "opacity": 1
##   },
##   "debounce": 0
## },
## "shinyEvents": ["plotly_hover", "plotly_click", "plotly_selected", "plotly_relayout", "plotly_brushed",
"plotly_brushing", "plotly_clickannotation", "plotly_doubleclick", "plotly_deselect", "plotly_afterplot",

```

```
"plotly_sunburstclick"],  
## "base_url": "https://plot.ly"  
## }
```

- python

python 에서는 `print()`를 사용하여 데이터 구조를 확인할 수 있다.

```
import plotly.graph_objects as go  
  
fig = go.Figure(  
    data = [  
        {  
            'type' : 'scatter',  
            'mode' : 'markers+lines',  
            'x' : df_covid19_100.loc[df_covid19_100['iso_code'] == 'KOR', 'date'],  
            'y' : df_covid19_100.loc[df_covid19_100['iso_code'] == 'KOR', 'new_cases'],  
            'marker' : {  
                'color' : '#264E86'  
            },  
            'line' : {  
                'color' : '#5E88FC',  
                'dash' : 'dash'  
            }  
        }  
    ]  
)  
  
print(fig)  
  
## Figure({  
##     'data': [{ 'line': { 'color': '#5E88FC', 'dash': 'dash'},  
##               'marker': { 'color': '#264E86'},  
##               'mode': 'markers+lines',  
##               'type': 'scatter',  
##               'x': array([datetime.datetime(2022, 8, 4, 0, 0),  
##                          datetime.datetime(2022, 8, 5, 0, 0),  
##                          datetime.datetime(2022, 8, 6, 0, 0),  
##                          중략  
##                          datetime.datetime(2022, 11, 11, 0, 0),  
##                          datetime.datetime(2022, 11, 12, 0, 0)], dtype=object),  
##               'y': array([112857., 110610., 105507., 55292., 149819., 151734., 137196.,  
##                          128671.,
```

```
##          124515., 119603., 62078., 84128., 180652., 178480., 138741.,
129350.,
##          110944., 59046., 150098., 139165., 113281., 101064., 95538.,
85295.,
##          43142., 115519., 103919., 81499., 89528., 79623., 72144.,
37548.,
##
##          중략
##
##          58358., 54740., 46870., 43424., 40863., 36675., 18671.,
62273.,
##          62472., 55365., 54519., 54225., 48465.]]]],
##      'layout': {'template': '...'}
##  })
```

4. plotly 사용하기

앞에서 **plotly** 객체의 구조에 대해 살펴보았다. **plotly** 는 plotly.js 에서 사용할 수 있는 JSON 형태의 객체이기 때문에 **plotly** 는 이 JSON 형태의 결과물을 시각화로 보여줄 뿐 결국은 데이터 구조일 뿐이다. 따라서 **plotly** 를 사용하여 시각화 한다는 것은 ‘data’, ‘layout’, ‘frame’ 속성 값들을 설정하여 JSON 구조를 만드는 것이다.

4.1. data : trace 의 설정

앞서 설명한 바와 같이 **plotly** 객체의 구조에서 시각화해야하는 데이터를 정의하는 속성은 ‘data’ 속성이다. 하지만 이 속성은 점이든 막대이든 원이든 특정한 도형으로 표현하여야 하는데 이렇게 데이터를 도형으로 표현하는 각각의 레이어를 트레이스라고 한다. 따라서 ‘data’ 속성은 ‘trace’들을 모아 놓은 것이다.

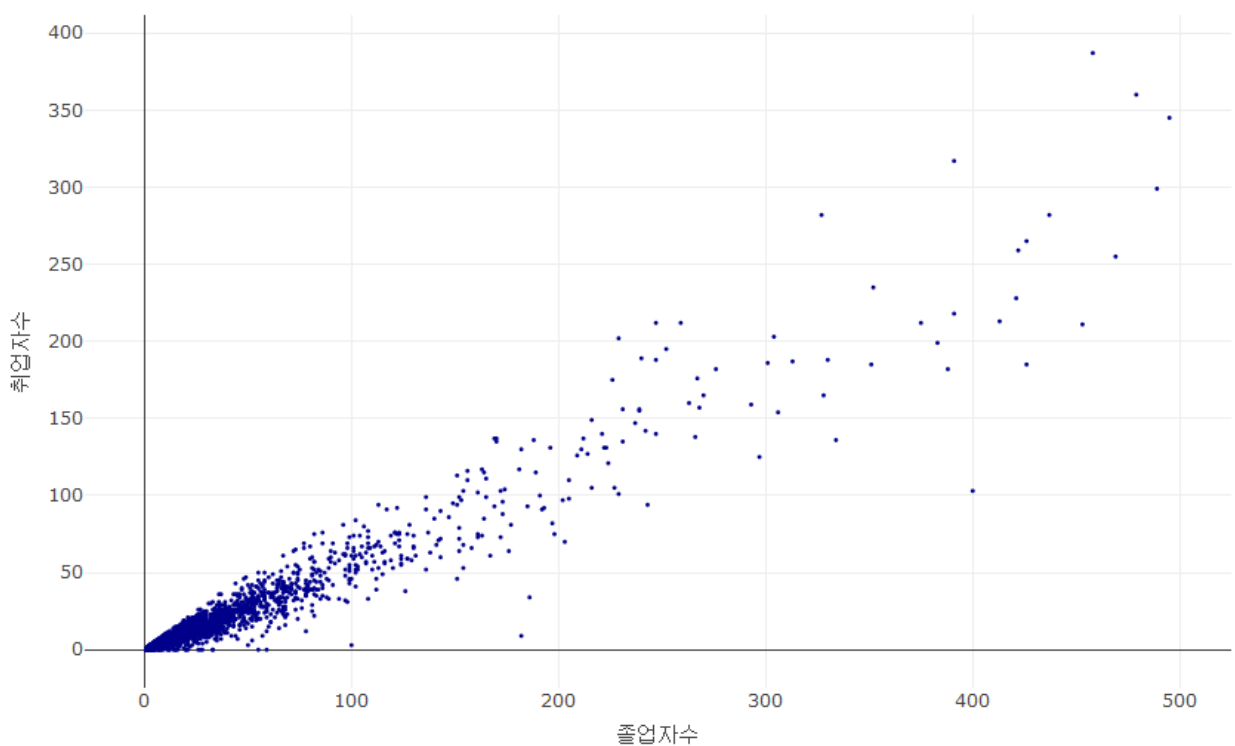
plotly 를 시작하려면 먼저 **plotly** 를 초기화해야 한다. 앞에서 본 것과 같이 초기화하는 **plot_ly()** 나 **Figure()** 에서 ‘data’ 속성의 하위 속성을 설정하여 **plotly** 시각화를 그릴수 있다. 하지만 이 방법을 사용하면 코드의 길이가 길어지고 리스트와 딕셔너리 등 **plotly** 구조에 맞게 괄호들을 설정해야 하기 때문에 매우 복잡해진다.

따라서 이 방법 보다는 **add_trace()** 를 사용하여 초기화된 **plotly** 객체에 트레이스를 추가하는 방법이 많이 사용된다.

- R

R 에서 트레이스를 설정하기 위해서는 `add_trace()`를 사용하거나 `add_markers()`, `add_bars()`와 같은 트레이스에 특화된 함수를 사용하는 방법이 있다. 이들 함수를 사용하여 바로 트레이스와 관련된 속성들을 설정한다. 여기서는 `add_trace()`를 사용하는 방법을 위주로 설명하고 다음장에서는 트레이스에 특화된 함수를 설명하겠다. `add_trace()`에서 설정하는 트레이스 속성 중 루트 노드에 해당하는 첫 번째 레벨의 속성은 매개변수처럼 `=`을 사용하여 속성명에 속성값을 설정한다. 만약 이 첫 번째 레벨의 속성이 하위 속성들로 구성되어야 한다면 `list()`를 사용하여 리스트 데이터 타입으로 하위 속성들을 설정한다.

```
df_취업률_2000 |>
  filter(졸업자수 < 500) |>
  plot_ly() |>
  add_trace(type = 'scatter', mode = 'markers',
    x = ~졸업자수, y = ~취업자수,
    marker = list(size = 3, color = 'darkblue'))
```



- python

python 에서 트레이스를 설정하기 위해서는 여러 가지 방법이 있다. `plotly` 를 그리기 위해서는 먼저 `'plotly.graph_object'`를 사용할 것인지, `'plotly.express'`를 사용할 것인지에 따라 나뉜다. `'plotly.express'`는 다음 장에서 설명하고 여기서는 `add_trace()`를 사용하여 `'plotly.graph_object'`를 사용해서 `plotly` 를 그리는 두 가지 방법을 설명한다.

'plotly.graph_object'를 사용하여 `plotly` 를 그리는 첫 번째 방법은 `add_trace()`의 매개변수로 딕셔너리로 구성된 해당 트레이스의 하위 속성들을 설정한다.

```
fig = go.Figure()

fig.add_trace({
    'type' : 'scatter',
    'mode' : 'markers',
    'x' : df_취업률_2000['졸업자수'],
    'y' : df_취업률_2000['취업자수'],
    'marker' : {
        'color' : 'darkblue'
    }
})
```

두 번째 방법은 `add_trace()`의 매개변수로 'plotly.graph_object'에서 제공하는 각각의 트레이스 함수를 사용하는 것이다. 'plotly.graph_object'에서 제공하는 각각의 트레이스 함수는 약 90 여개이다. 이들을 다 외울 수는 없으니 plotly 홈페이지에서 확인하여 사용하여야 한다. 하지만 이 트레이스 함수를 사용하는 것도 두 가지 방법이 있다. 첫 번째 방법은 앞의 `add_trace()`에 딕셔너리를 설정한 것과 같이 트레이스 함수에 트레이스 속성들로 구성된 딕셔너리를 설정하는 방법이다.

```
fig = go.Figure()

fig.add_trace({
    'type' : 'scatter',
    'mode' : 'markers',
    'x' : df_취업률_2000['졸업자수'],
    'y' : df_취업률_2000['취업자수'],
    'marker' : {
        'color' : 'darkblue'
    }
})
```

두 번째는 `=`을 사용하여 속성에 속성값을 직접 할당하는 방법이다. 이 방법은 각각의 트레이스의 루트노드인 첫 번째 레벨 속성값에 한해 설정할 수 있고 세부 속성으로 구성된 속성은 또 다시 딕셔너리로 설정하여야 한다. 그런데 이 딕셔너리를 설정하는데에도 두 가지 방법이 있다. 첫 번째 방법은 `{}`를 사용하여 딕셔너리를 구성하는 방법으로 이 방법을 사용할 때는 속성명을 `' '`나 `" "`로 묶어주어야 하고 할당 기호로 `:`을 사용한다.

```
fig = go.Figure()

fig.add_trace(go.Scatter({
    'type' : 'scatter',
    'mode' : 'markers',
    'x' : df_취업률_2000['졸업자수'],
    'y' : df_취업률_2000['취업자수'],
    'marker' : {'color' : 'darkblue'}}
))
```

두 번째 방법은 `dict()`를 사용하여 딕셔너리를 구성하는데 이 방법을 사용할 때는 속성명에 따옴표 없이 사용하고 할당 기호로 `=`을 사용한다.

```
fig = go.Figure()

fig.add_trace(go.Scatter(
    type = 'scatter',
    mode = 'markers',
    x = df_취업률_2000['졸업자수'],
    y = df_취업률_2000['취업자수'],
    marker = dict(color = 'darkblue')
))
```

4.2. 트레이스 공통 속성

앞서 설명한 바와 같이 `plotly`에서는 40 여 개가 넘는 트레이스를 제공한다. 그 트레이스마다 사용되는 속성들이 다르지만 공통적으로 사용되는 속성들이 있다. 각각의 트레이스에서 공통적으로 사용되는 대표적인 속성들은 다음과 같다.

속성	속성 설명	python 속성값	R 속성값
name	trace 이름을 설정, 설정된 이름은 범례와 호버에 표시	문자열	
visible	trace 의 표시 여부를 결정, 'legendonly'는 trace 는 표시하지 않고 범례만 표시	True False "legend only"	
showlegend	범례를 표현할지를 결정	boolean	
opacity	trace 의 투명도 설정	0 부터 1 까지의 수치	
x	x 축의 설정	list, numpy array, or Pandas series of	dataframe column, list, vector

속성	속성 설명	python 속성값	R 속성값
		numbers, strings, or datetimes	
y	Y 축의 설정	list, numpy array, or Pandas series of numbers, strings, or datetimes	dataframe column, list, vector
text	각각의 (x, y) 좌표에 해당하는 문자열 설정, 만약 단일 문자열이 설정되면 모든 데이터에 같은 문자열이 표시되고 문자열 배열이 온다면 각각의 데이터에 해당하는 문자열이 표시됨. 만약 'hoverinfo'에 text 가 포함되고 'hovertext'가 설정되지 않는다면 text 속성은 hover 라벨로 표시됨	문자열 또는 문자열 배열	
textposition	text 속성값이 (x, y) 위치에서 표시되는 위치를 설정	"top left" "top center" "top right" "middle left" "middle center" "middle right" "bottom left" "bottom center" "bottom right"	
texttemplate	각각의 포인트에 나타나는 정보를 표시하기 위한 템플릿, textinfo 속성에 오버라이드 됨. 변수는 %{변수명}의 형태로 설정이 가능하고 숫자 포맷은 d3-format 문법을 사용하여 설정함.	문자열 또는 문자열 배열	
hovertext	각각의 (x, y)에 관련된 호버의 text 개체를 설정, 만약 단일 문자열이 설정되면 모든 포인트에 대해 같은 문자가 표시되고, 문자 배열이 설정되면 각각의 (x, y)에 해당하는 문자열이 표시됨, hovertext 가	문자열 또는 문자열 배열	

속성		속성 설명	python 속성값	R 속성값
		보이려면 'hoverinfo' 속성에 'text'가 포함되어야 함		
hoverinfo		호버에 어떤 trace 정보가 표시되는지를 결정, 만약 none 이나 skip 이 설정되면 호버에 아무것도 표시되지 않지만 none 이 설정되면 호버 이벤트는 계속 발생된다.	Any combination of "x", "y", "z", "text", "name" joined with a "+" OR "all" or "none" or "skip".	
hovertemplate		호버박스에 나타나는 정보에 사용되는 템플릿 문자열	문자열 또는 문자열 배열	
xaxis		trace 의 X 좌표와 2 차원 X 축간의 참조 설정. 만약 'x'가 설정되면 X 축은 'layout.xaxis'를 참조하고 'x2'가 설정되면 'layout.xaxis2'를 참조하게 됨	subplotid	
yaxis		trace 의 Y 좌표와 2 차원 Y 축간의 참조 설정. 만약 'y'가 설정되면 Y 축은 'layout.yaxis'를 참조하고 'y2'가 설정되면 'layout.yaxis2'를 참조하게 됨	subplotid	
orientation			("v" "h")	
textfont	color	문자의 색 설정	색상이나 색상 배열	
	family	문자의 HTML 폰트 설정	문자열 또는 문자열 배열	
	size	문자의 크기 설정	0 부터 1 까지의 수치나 수치배열	
hoverlabel	align	호버 문자 박스안의 문자 콘텐츠의 수평 정렬 설정	("left" "right" "auto")	
	bgcolor	호버 라벨의 배경색 설정	색상이나 색상 배열	
	bordercolor	호버 라벨의 경계선 색 설정	색상이나 색상 배열	
	font	color	색상이나 색상 배열	
		family	문자열 또는 문자열 배열	
		size	0 부터 1 까지의 수치나 수치배열	

4.2.1. type

'trace' 설정에 가장 중요한 속성이 'type' 속성이다. 'type' 속성은 데이터를 표현할 방식을 설정하기 때문에 시각화의 가장 중요한 형태를 결정하는 속성이다. **plotly** 에서 지원하는 주요 'trace'는 다음과 같다.

- scatter 타입 : 'scatter', 'scattergl'
- bar 타입 : 'bar', 'funnel', 'waterfall'
- 집계된 bar 타입 : 'histogram'
- 1 차원 분포 타입 : 'box', 'violin'
- 2 차원 밀도 분포 타입 : 'histogram2d', 'histogram2dcontour'
- 매트릭스 타입 : 'image', 'heatmap', 'contour'
- 주가 타입 : 'ohlcv', 'candlestick'

4.2.2. mode

'mode'는 'type'이 'scatter'인 스캐터 타입의 트레이스에서 사용하는 속성이다. 스캐터 타입의 트레이스는 점, 선, 문자를 사용하여 데이터를 표현한다. 따라서 스캐터 차트를 그릴 때는 세 가지 타입의 스캐터 타입중 어떤 것을 사용할지를 설정해야하는데, 이것을 설정하는 속성이 'mode'이다. 'mode'는 다음의 네 가지가 있는데 이 'mode'들은 '+' 기호를 사용하여 여러개의 'mode'를 동시에 사용할 수 있다.

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

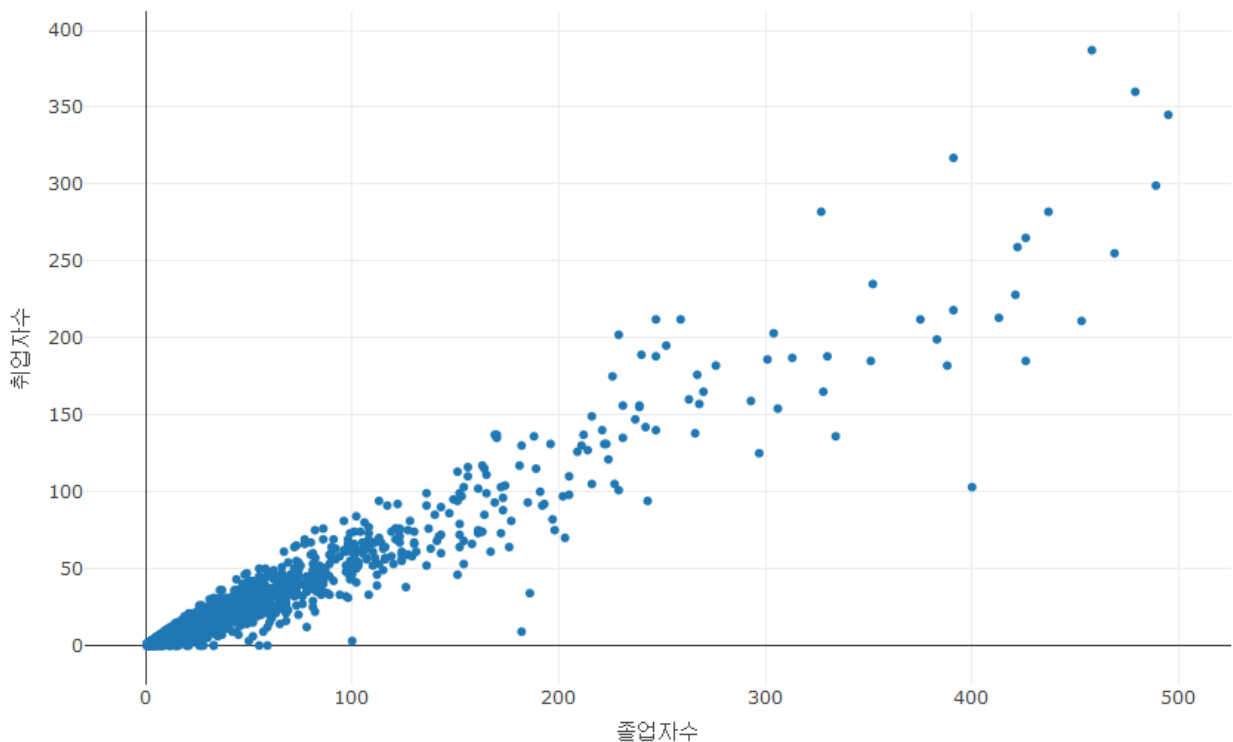
4.2.3. x, y

'trace' 설정에서 2 차원 데카르트 좌표축에 데이터를 매핑하는 속성이 **x, y** 이다. 데카르트 좌표계를 사용하는 'trace'에서 가장 기본적으로 사용되는 속성이며 하위 속성은 없다.

- R

R에서는 'x', 'y'에 변수를 할당 가능한 변수는 데이터프레임 열, 리스트, 벡터 등이다. 할당할 때 주의해야 할 것은 반드시 `~`를 변수명 앞에 붙여줘야 한다.

```
## df_취업률_2000 에서
df_취업률_2000 |>
  ## X 축은 졸업자수, Y 축은 취업자수로 매핑한 plotly 객체 생성
  plot_ly() |>
  add_trace(type = 'scatter', mode = 'markers',
            x = ~졸업자수, y = ~취업자수)
```



실행결과 2- 4. plotly 기본 산점도 생성

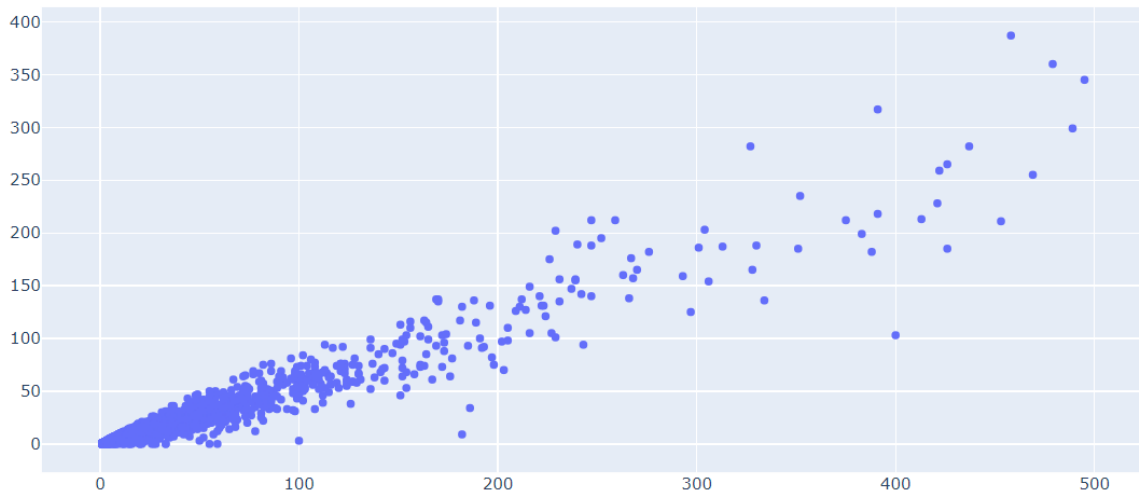
- python

python에서는 'x', 'y' 속성에 리스트(list), 넘파이 배열(numpy array), 수치형 판다스 시리즈(pandas series), 수치형 시리즈(series), 문자열(strings) 날짜와 시간형(datetimes) 등을 설정할 수 있다.

```
fig = go.Figure()

fig.add_trace(
  {'type' : 'scatter',
   'mode' : 'markers',
   'x': df_취업률_2000['졸업자수'],
```

```
'y': df_취업률_2000['취업자수']
    }
)
fig.show()
```



python 의 Figure() 를 사용한 plotly 초기화

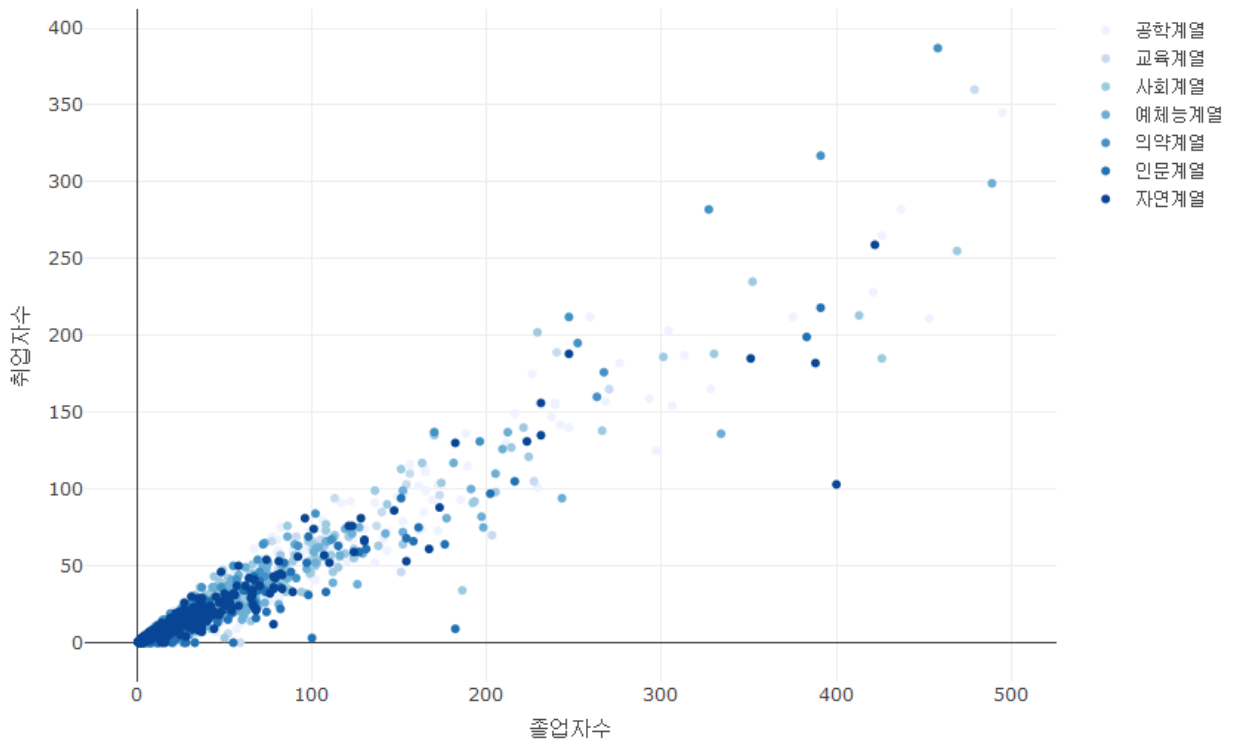
4.2.4. name

'name'은 **plotly** 에서 사용되는 각각의 트레이스의 이름을 설정하는 속성이다. 'name'은 범례와 호버에 해당 데이터를 인식하기 위해 나타난다.

- R

R에서는 'name' 속성에 단일 문자열을 할당할 수도 있고, 'x', 'y'과 같이 **plotly** 에 바인딩된 데이터 프레임의 문자형 열, 문자형 리스트, 문자형 벡터를 사용할 수도 있다. 다만 데이터프레임 열, 리스트, 벡터를 사용한다면 **plotly** 포 표시되는 데이터에 1:1 매핑되어야 한다.

```
df_취업률_2000 |>
## X 축은 졸업자수, Y 축은 취업자수, name 은 대계열로 매핑한 plotly 객체 생성
plot_ly() |>
add_trace(type = 'scatter', mode = 'markers',
          x = ~졸업자수, y = ~취업자수, name = ~대계열)
```



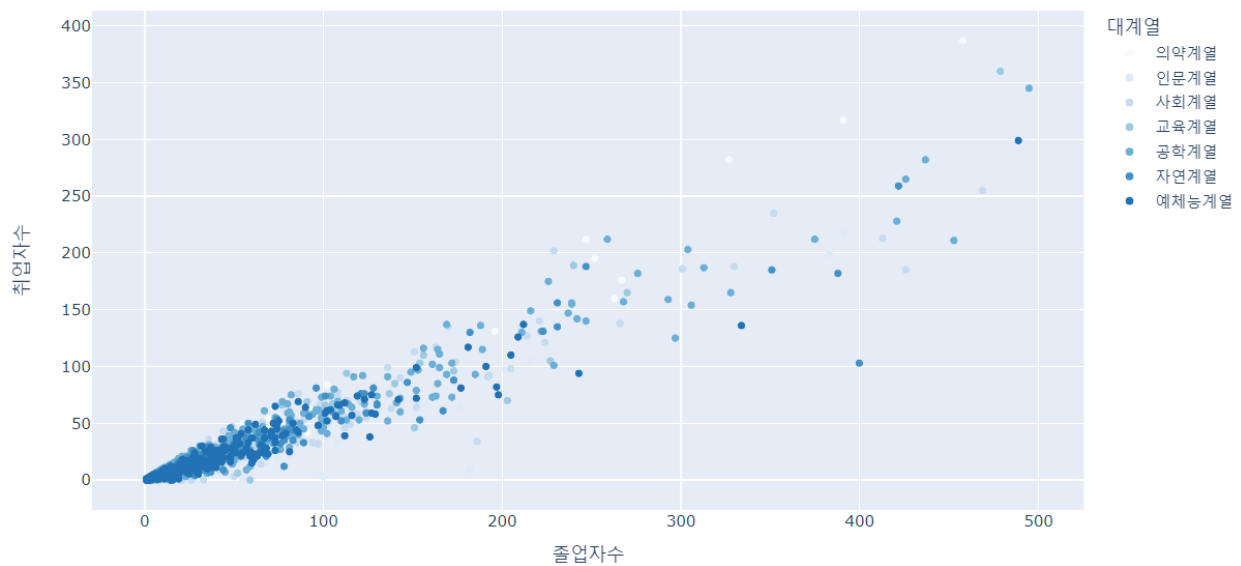
실행결과 2- 5. trace 이름 매핑

- python

python 에서 'name' 속성을 사용하는데는 R 보다는 다소 번거롭다. python 에서 'name'에 설정이 가능한 데이터 타입은 단일 문자열(string)만을 설정할 수 있다. 따라서 여러 개의 트레이스를 사용하려면 루프를 사용하여 각각의 트레이스의 'name'을 각각 설정해야 한다.

```
fig = go.Figure()

for 대계열, group in df_취업률_2000.groupby('대계열'):
    fig.add_trace({
        'type' : 'scatter',
        'mode' : 'markers',
        'x': group['졸업자수'],
        'y': group['취업자수'],
        'name' : 대계열
    })
fig.show()
```

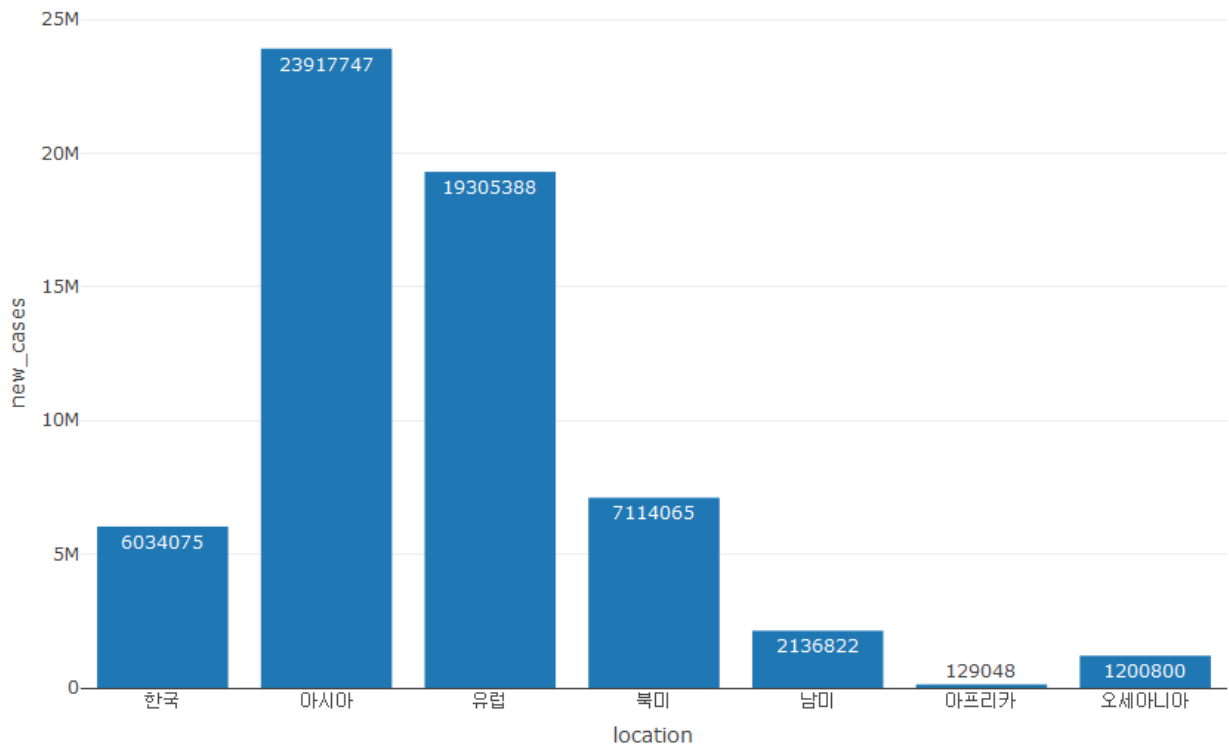
4.2.5. text

데이터 시각화에서 데이터는 다양한 그래픽적 기하 도형으로 표현되기 때문에 데이터의 정확한 값을 측정하는데 어려움이 있다. 이를 보완하기 위해 데이터 시각화에 데이터 값을 표기하는 경우가 많은데, **plotly** 에서 지원하는 대부분의 트레이스에서는 'text' 속성을 사용해서 시각화의 데이터 값을 표현할 수 있다. 표시되기를 원하는 변수를 'text' 속성에 할당함으로써 데이터 값을 표시할 수 있으며 'textposition'과 'texttemplate'를 사용하여 표시되는 값의 위치나 표현 형태에 대한 세부 속성을 설정할 수 있다.

- R

R 에서 'text' 속성에 할당할 수 있는 변수는 단일 문자열이나 데이터 프레임의 문자형 열, 문자형 리스트, 문자형 벡터를 사용할 수 있다. 'text'에 단일 문자열을 할당하면 모든 데이터에 설정된 문자열이 표시되고 데이터 프레임의 문자형 열, 문자형 리스트, 문자형 벡터가 설정되면 문자열 벡터와 표현되는 데이터가 1:1 로 매핑되어 해당 데이터에 매핑된 문자열이 표시된다.

```
## 긴 형태의 100 일간 코로나 19 데이터 중에
df_covid19_100 |>
## 국가명으로 그룹화
group_by(location) |>
## 확진자수의 합계를 new_cases 로 산출
summarise(new_cases = sum(new_cases)) |>
## X 축을 location, Y 축과 text 를 new_case 로 매핑
plot_ly() |>
add_trace(type = 'bar', x = ~location, y = ~new_cases, text = ~new_cases)
```



- python

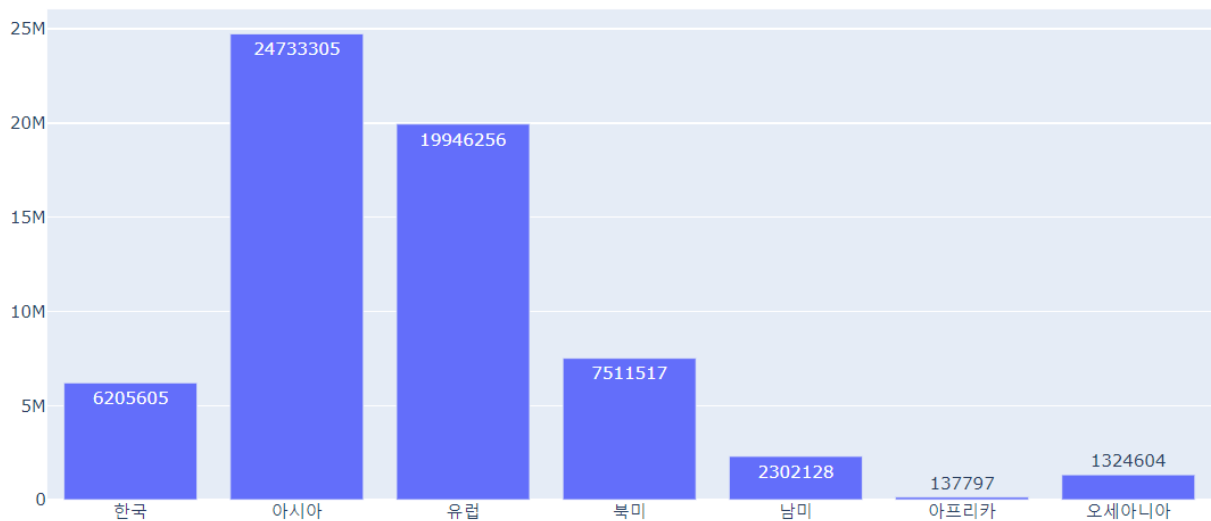
python 에서 'text' 속성에는 문자열 또는 문자열 배열을 할당할 수 있다. 앞서 'name' 속성때는 문자열 배열을 설정할 수 없었기 때문에 루프를 사용했지만 'text' 속성은 문자열 배열을 사용할 수 있기 때문에 표현되는 데이터에 1:1 로 매핑되는 문자를 표시할 수 있다.

```
fig = go.Figure()

temp = df_covid19_100.groupby('location').agg(new_cases = ('new_cases', 'sum'))

fig.add_trace({
    'type' : 'bar',
    'x': temp.index,
    'y': temp['new_cases'],
    'text' : temp['new_cases']
})

fig.show()
```



4.2.6. textposition

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

- R

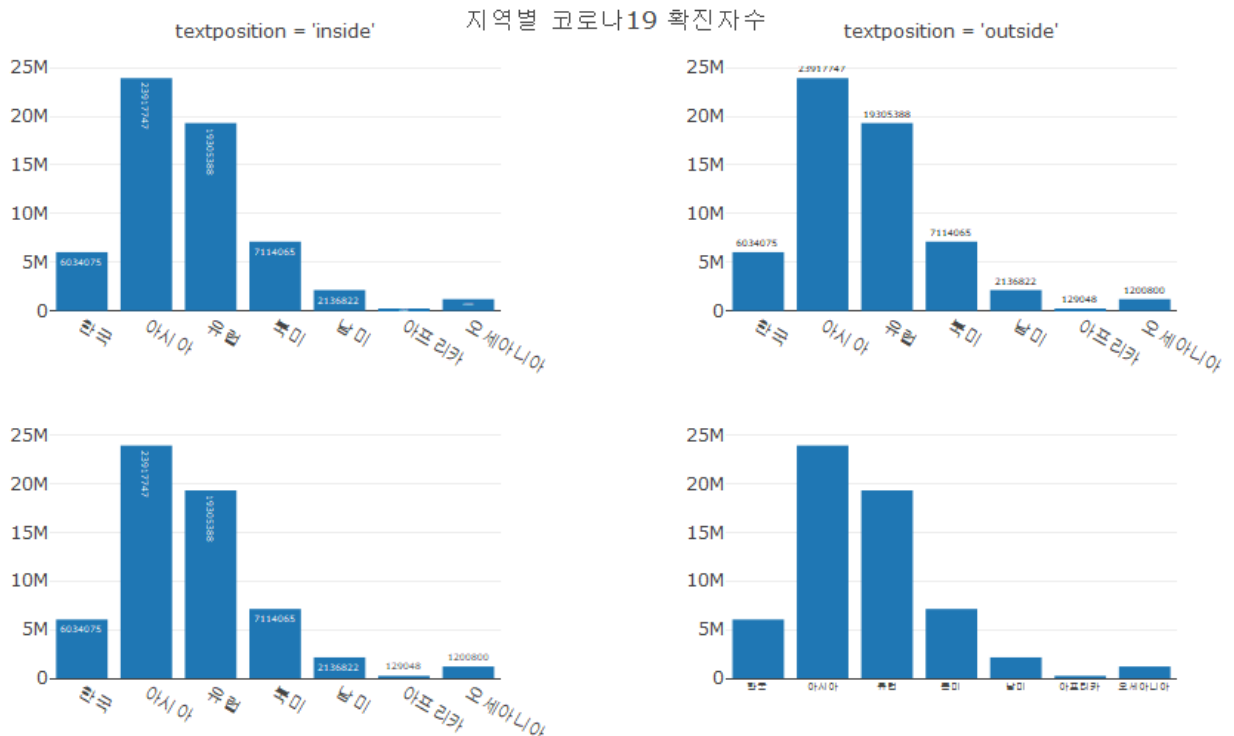
```
## 긴 형태의 100 일간 코로나 19 데이터 중에
df_covid19_100 |>
  ## 국가명으로 그룹화
  group_by(location) |>
  ## 확진자수의 합계를 new_cases 로 산출
  summarise(new_cases = sum(new_cases)) |>
  ## X 축을 location, Y 축과 text 를 new_case 로 매핑
  plot_ly() |>
  add_trace(type = 'bar', x = ~location, y = ~new_cases, text = ~new_cases,
            ## textposition 을 'inside'로 설정
            textposition = 'inside')
)

df_covid19_100 |>
  group_by(location) |>
  summarise(new_cases = sum(new_cases)) |>
  plot_ly() |>
  add_trace(type = 'bar', x = ~location, y = ~new_cases, text = ~new_cases,
```

```
## textposition 을 'outside'로 설정
textposition = 'outside')
```

```
df_covid19_100 |>
group_by(location) |>
summarise(new_cases = sum(new_cases)) |>
plot_ly() |>
add_trace(type = 'bar', x = ~location, y = ~new_cases, text = ~new_cases,
  ## textposition 을 'auto'로 설정
  textposition = 'auto')
```

```
df_covid19_100 |>
group_by(location) |>
summarise(new_cases = sum(new_cases)) |>
plot_ly() |>
add_trace(type = 'bar', x = ~location, y = ~new_cases, text = ~new_cases,
  ## textposition 을 'none'으로 설정
  textposition = 'none')
```



실행결과2- 6. textposition 에 설정 결과

- python

```
temp = df_covid19_100.groupby('location').agg(new_cases = ('new_cases', 'sum'))
```

```
## textposition 을 'inside'로 설정
fig = go.Figure()
```

```

fig.add_trace({
    'type' : 'bar',
    'x': temp.index,
    'y': temp['new_cases'],
    'text' : temp['new_cases'],
    'textposition' : 'inside'
})

fig.show()

## textposition 을 'outside'로 설정
fig = go.Figure()

fig.add_trace({
    'type' : 'bar',
    'x': temp.index,
    'y': temp['new_cases'],
    'text' : temp['new_cases'],
    'textposition' : 'outside'
})

fig.show()

## textposition 을 'auto'로 설정
fig = go.Figure()

fig.add_trace({
    'type' : 'bar',
    'x': temp.index,
    'y': temp['new_cases'],
    'text' : temp['new_cases'],
    'textposition' : 'auto'
})

fig.show()

## textposition 을 'none'로 설정
fig = go.Figure()

fig.add_trace({

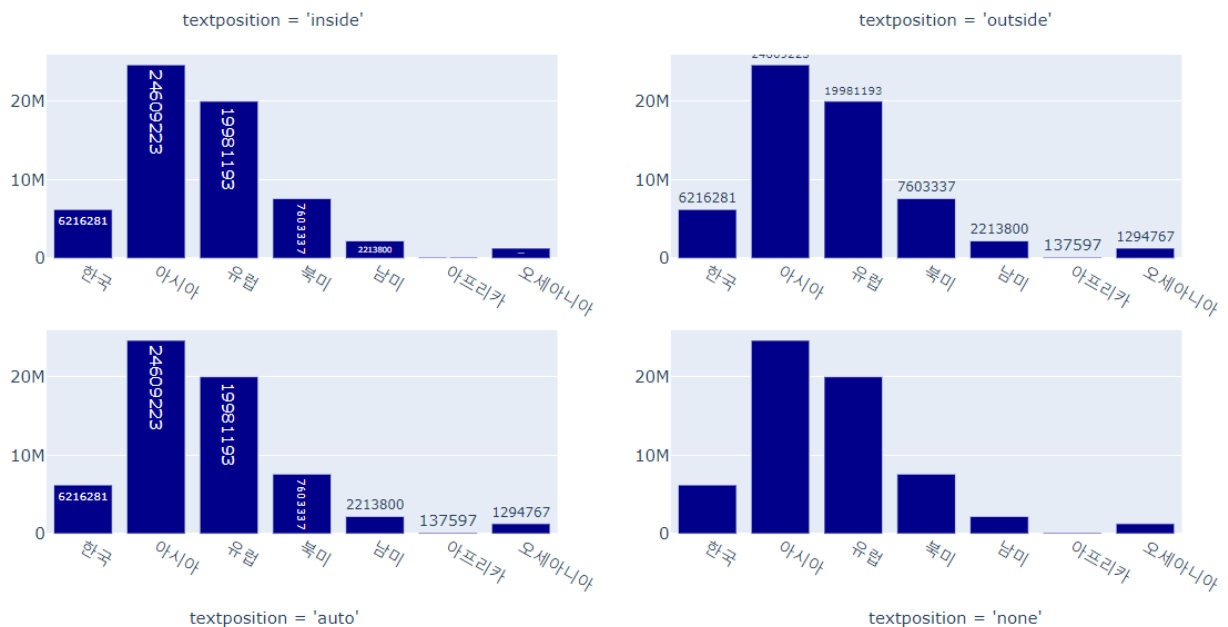
```

```

    'type' : 'bar',
    'x': temp.index,
    'y': temp['new_cases'],
    'text' : temp['new_cases'],
    'textposition' : 'none'
  }
)

```

fig.show()



4.2.7. texttemplate

texttemplate은 텍스트가 표시되는 형태를 설정하는 type 이다. **texttemplate**은 **hovertemplate**의 정의와 같이 사용되는데 ‘%{변수:변수포맷}’의 형태로 사용된다. 변수 포맷에서 사용하는 포맷은 자바 스크립트의 d3 format⁴을 사용한다. 앞의 예에서 일반대학의 입학생수의 포맷에 천단위 콤마를 넣는다면 ‘%{text:,}’로 설정하고 소수점 아래 두째 자리까지 표기한다면 ‘%{text:2f}’로 설정한다.

- R

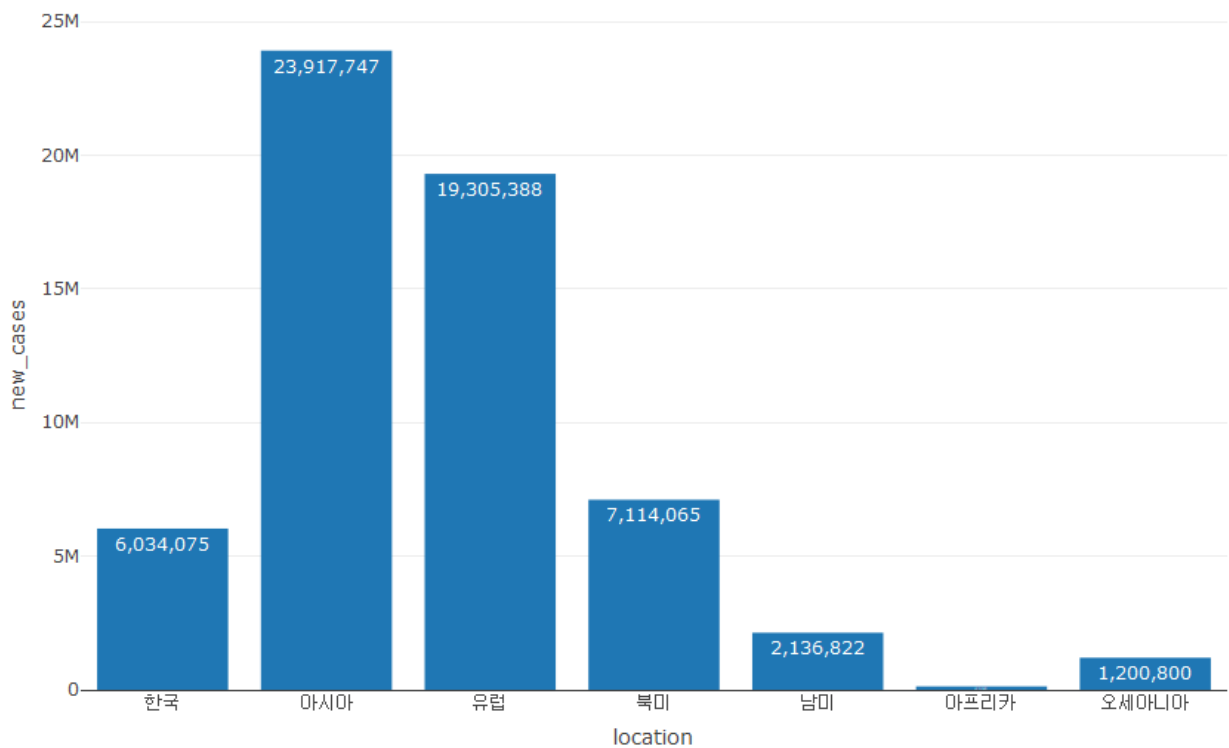
```

## 긴 형태의 100 일간 코로나 19 데이터 중에
df_covid19_100 |>
## 국가명으로 그룹화
group_by(location) |>

```

⁴ <https://github.com/d3/d3-format/tree/v1.4.5#d3-format>

```
## 확진자수의 합계를 new_cases 로 산출
summarise(new_cases = sum(new_cases)) |>
## X 축을 location, Y 축과 text 를 new_case 로 매핑
plot_ly() |>
add_trace(type = 'bar', x = ~location, y = ~new_cases, text = ~new_cases,
  ## textposition 을 'inside'로 설정
  textposition = 'inside',
  ## texttemplate 를 설정
  texttemplate = '%{text:,}'
)
```



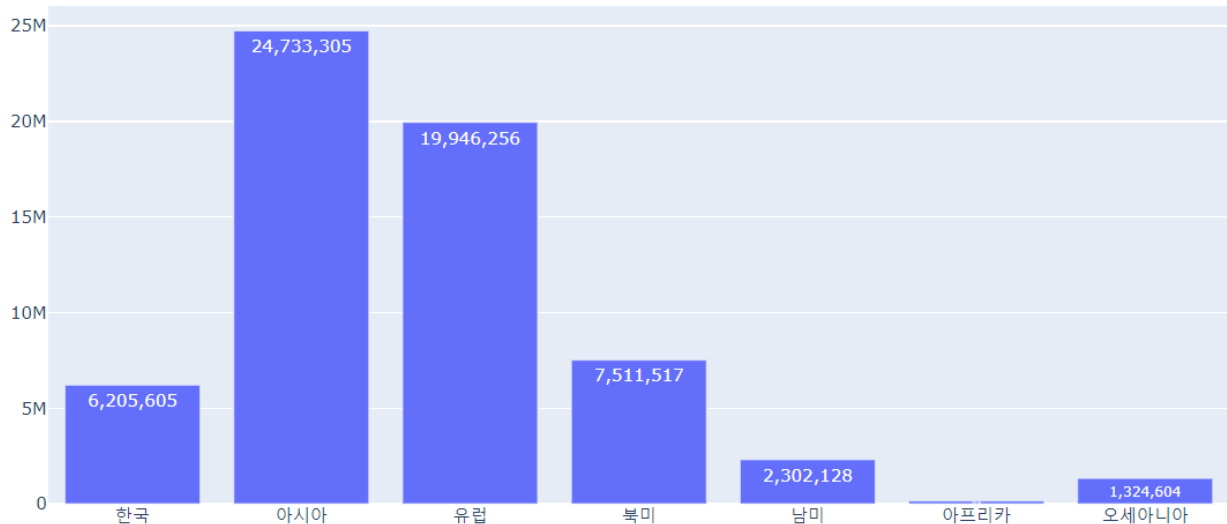
실행결과 2- 7. texttemplate 설정 결과

- python

```
fig = go.Figure()

fig.add_trace({
  'type' : 'bar',
  'x': temp.index,
  'y': temp['new_cases'],
  'text' : temp['new_cases'],
  'textposition' : 'inside',
  ## texttemplate 를 설정
  'texttemplate' : '%{text:,}'
})
```

```
)  
fig.show()
```



4.2.8. hover

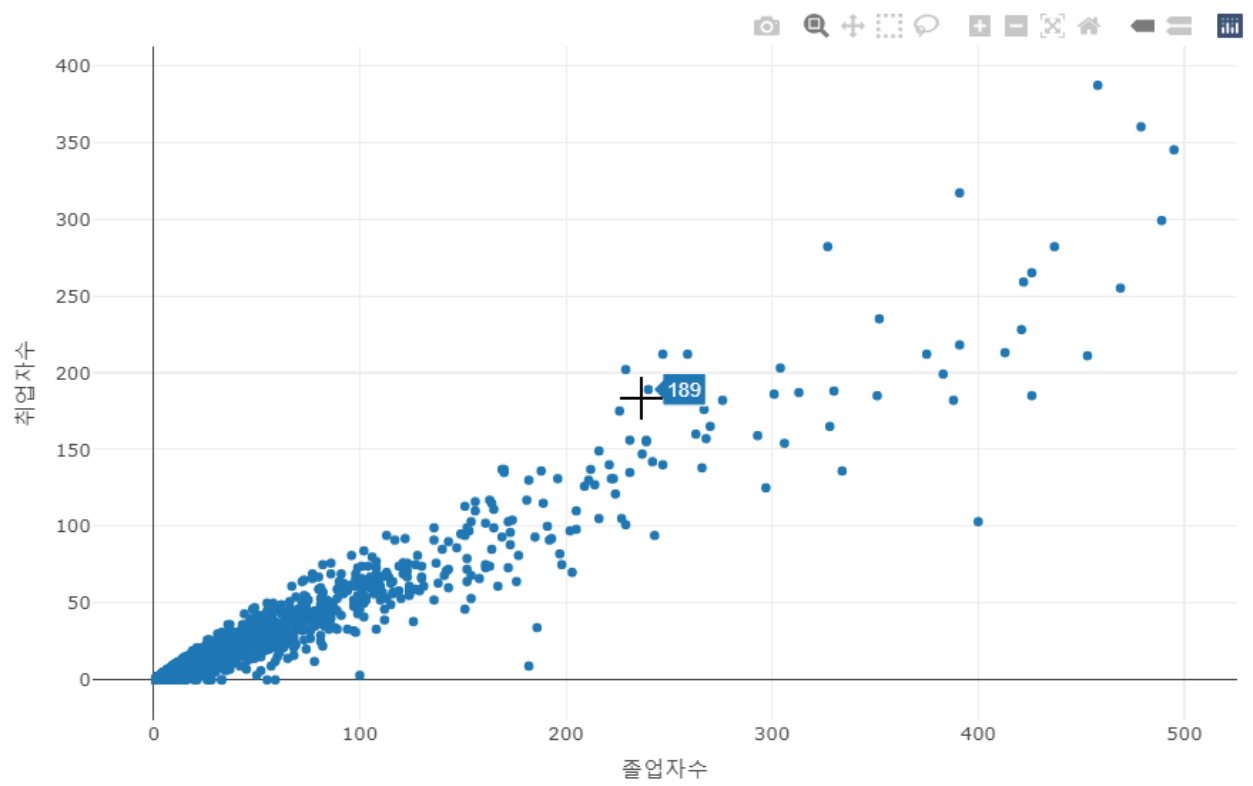
plotly 와 같은 동적 시각화에서는 대부분 마우스 포인터를 데이터가 표시된 점이나 선에 위치하면 해당 위치의 데이터가 표시된다. **plotly** 에서는 이렇게 데이터의 정보를 표시하는 말풍선을 'hover'라고 한다. 'hover'는 시각화를 설계하는 사용자에게 따라 표시할 정보를 설정할 수 있는데 이 정보를 설정하는 속성이 **hover***이다. 호버는 'trace'마다 다르기 때문에 각각의 'trace'마다 설정하는 항목이 다르다.

4.2.8.1. hoverinfo

'hoverinfo'는 호버에 표시되는 데이터 정보를 설정하는 속성으로 'x'(X 축 좌표값), 'y'(Y 축 좌표값), 'z'(Z 축 좌표값), 'text'('text' 속성값), 'name'(trace 이름), 'none'(호버 제거), 'skip'(생략)이 사용될 수 있고 각각은 **+**를 사용하여 조합할 수 있다.

- R

```
df_취업률_2000 |>  
plot_ly() |>  
add_trace(type = 'scatter', mode = 'markers',  
          x = ~졸업자수, y = ~취업자수,  
          hoverinfo = 'y')
```

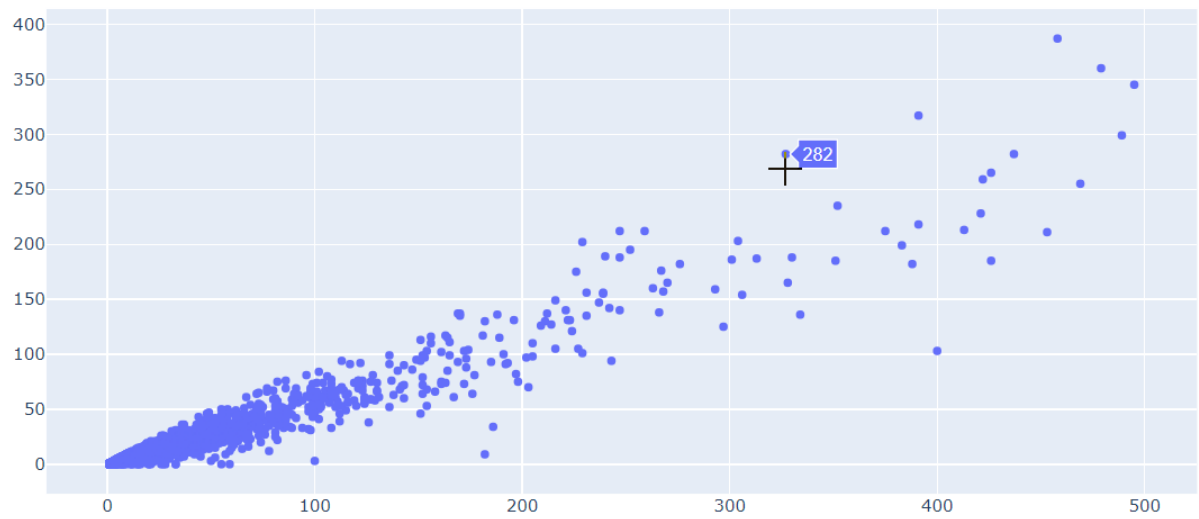
hoverinfo

- python

```
fig = go.Figure()

fig.add_trace({
    'type' : 'scatter',
    'mode' : 'markers',
    'x': df_취업률_2000['졸업자수'],
    'y': df_취업률_2000['취업자수'],
    'hoverinfo' : 'y'
})

fig.show()
```



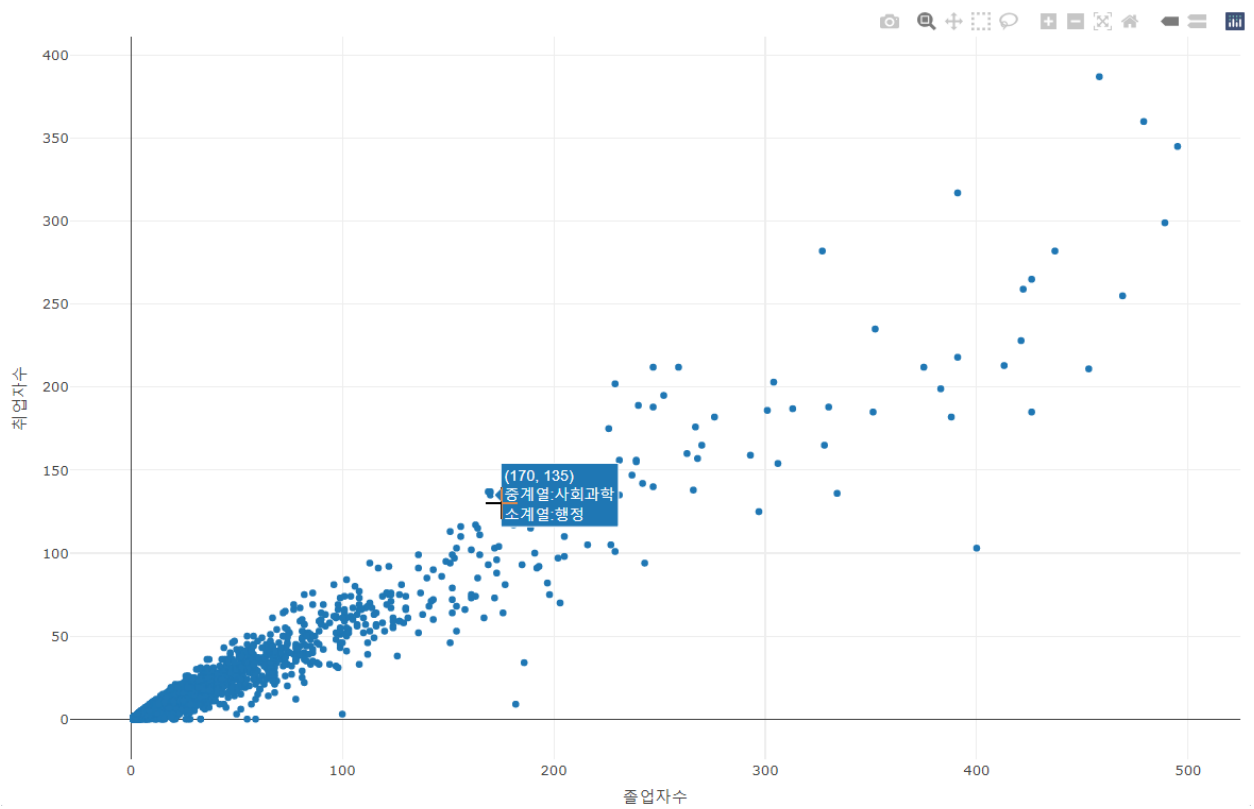
hoverinfo

4.2.8.2. hovertext

'hovertext'는 기본적으로 표시되는 호버의 정보에 추가적인 정보를 표시하는 속성이다. 이 속성에는 데이터프레임의 열을 매핑할 수도 있고 고정된 문자열을 설정할 수도 있고 문자열과 데이터프레임의 열을 포함한 문자열을 설정할 수 있다.

- R

```
df_취업률_2000 |>
  plot_ly() |>
  add_trace(type = 'scatter', mode = 'markers',
    x = ~졸업자수, y = ~취업자수,
    ## hovertext 를 학과명으로 매핑
    hovertext = ~paste0('중계열:', 중계열, '\n', '소계열:', 소계열))
```



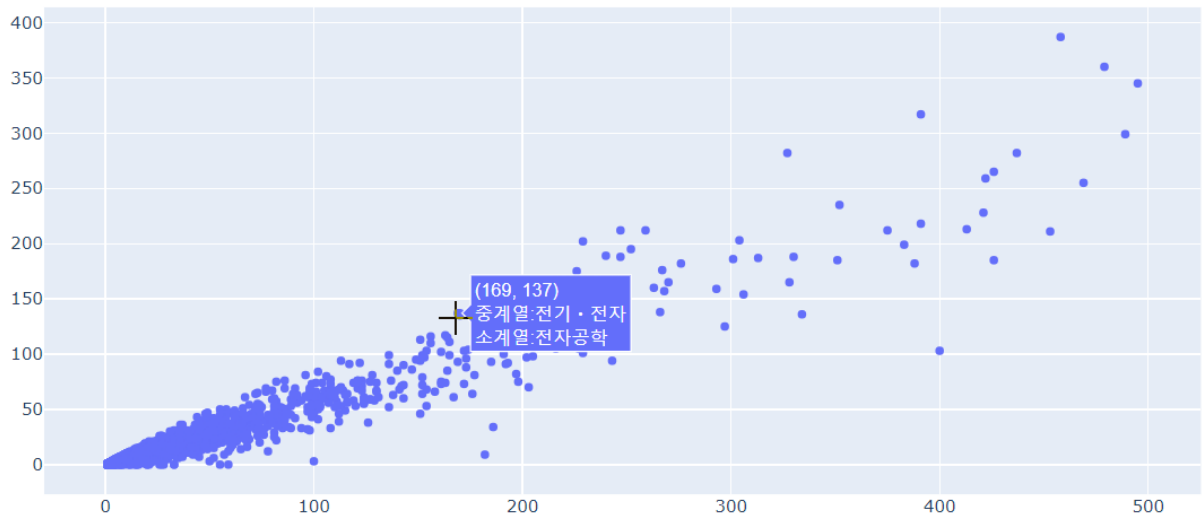
hovertext

- python

```
fig = go.Figure()

fig.add_trace({
    'type' : 'scatter',
    'mode' : 'markers',
    'x': df_취업률_2000['졸업자수'],
    'y': df_취업률_2000['취업자수'],
    'hoverinfo' : 'x+y+text',
    'hovertext' : '중계열:'+df_취업률_2000['중계열']+'<br>'+ '소계열:'+df_취업률_2000['소계열']
})

fig.show()
```



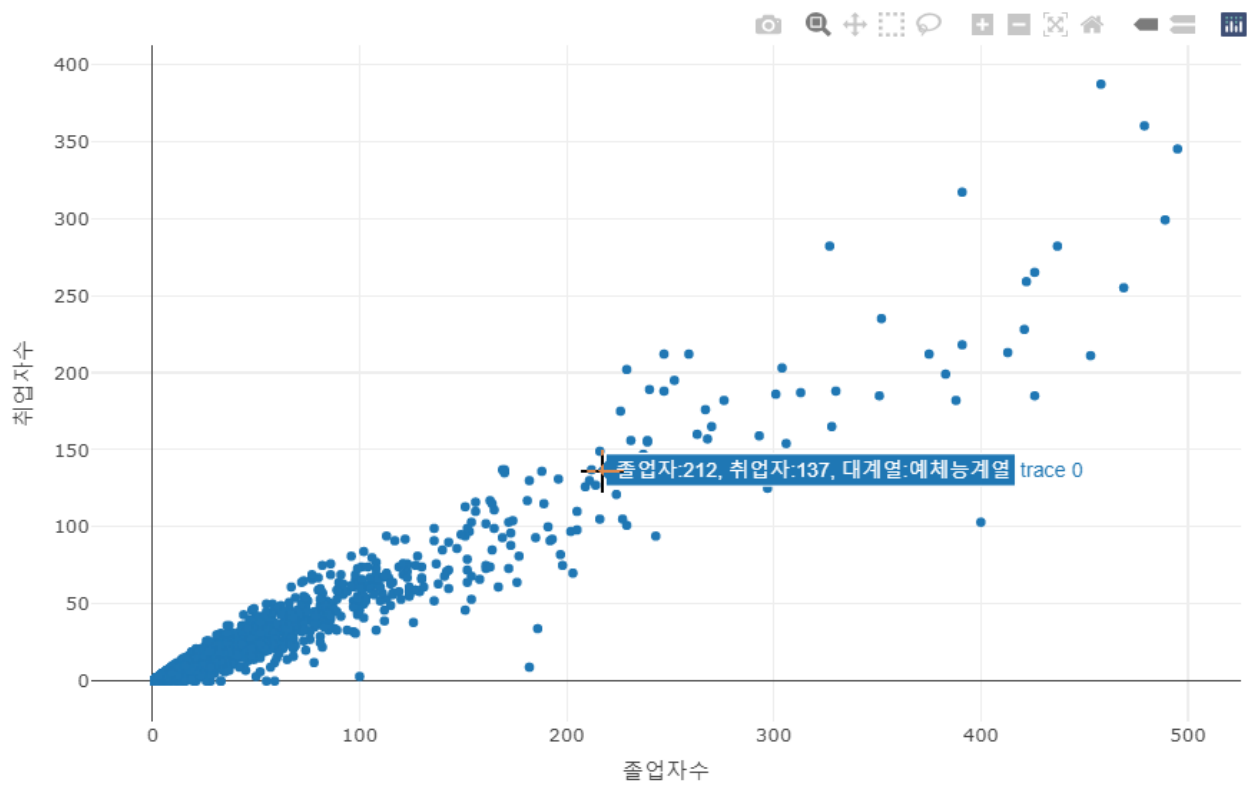
hovertext

4.2.8.3. hovertemplate

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

- R

```
df_취업률_2000 |>
  plot_ly() |>
  add_trace(type = 'scatter', mode = 'markers',
    x = ~졸업자수, y = ~취업자수, hovertext = ~대계열,
    ## hovertemplate 의 설정
    hovertemplate = ' 졸업자:%{x}, 취업자:%{y}, 대계열:%{hovertext}')
```



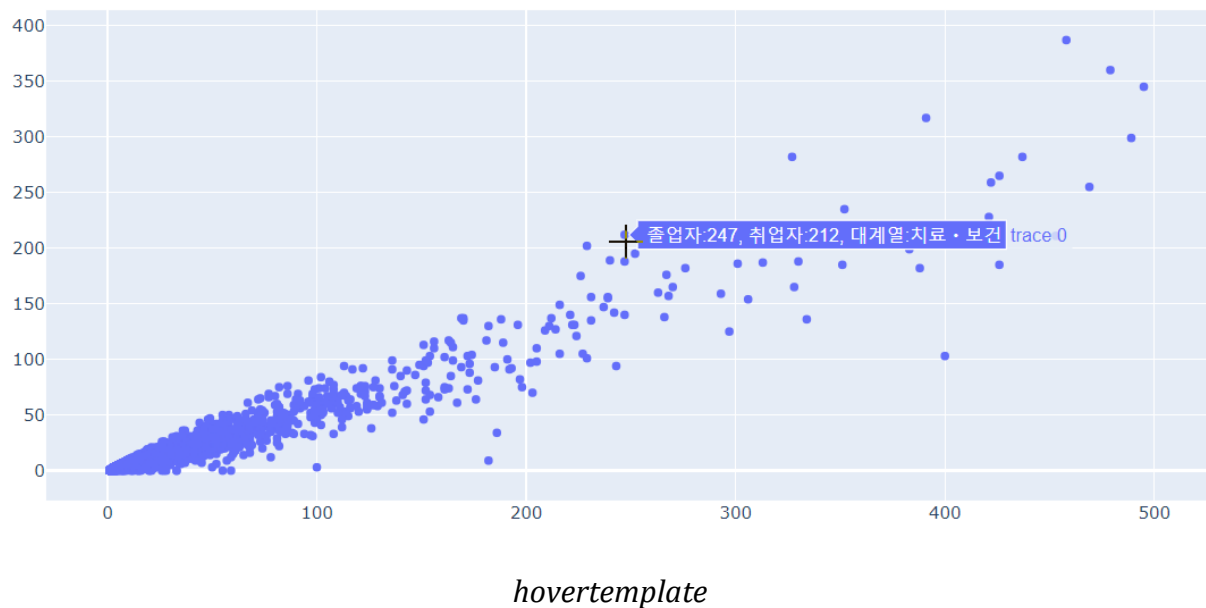
hovertemplate

- python

```
fig = go.Figure()

fig.add_trace({
    'type' : 'scatter',
    'mode' : 'markers',
    'x': df_취업률_2000['졸업자수'],
    'y': df_취업률_2000['취업자수'],
    'hovertext' : df_취업률_2000['중계열'],
    'hovertemplate' : ' 졸업자:%{x}, 취업자:%{y}, 대계열:%{hovertext}'
})

fig.show()
```



4.2.9. opacity

'opacity'는 투명도를 설정하는 속성이다. 투명도는 0 부터 1 사이의 값을 가지는데 0 은 투명하고 1 은 불투명하다. 여기서 하나 주의해야할 점은 가급적 'opacity'의 값은 0.5 이하로 설정하는 것이 바람직하다는 점이다. 'opacity'는 동일한 'trace' 안에서는 겹쳐진다고 해도 투명도가 변치않는다. 그러나 다른 'trace' 위에 겹쳐지는 경우는 아래의 투명도와 겹쳐져서 투명도가 올라가게 된다. 따라서 0.5 이상의 투명도가 2 개 이상 겹치게 되면 투명도가 1 이 넘기 때문에 불투명해진다.

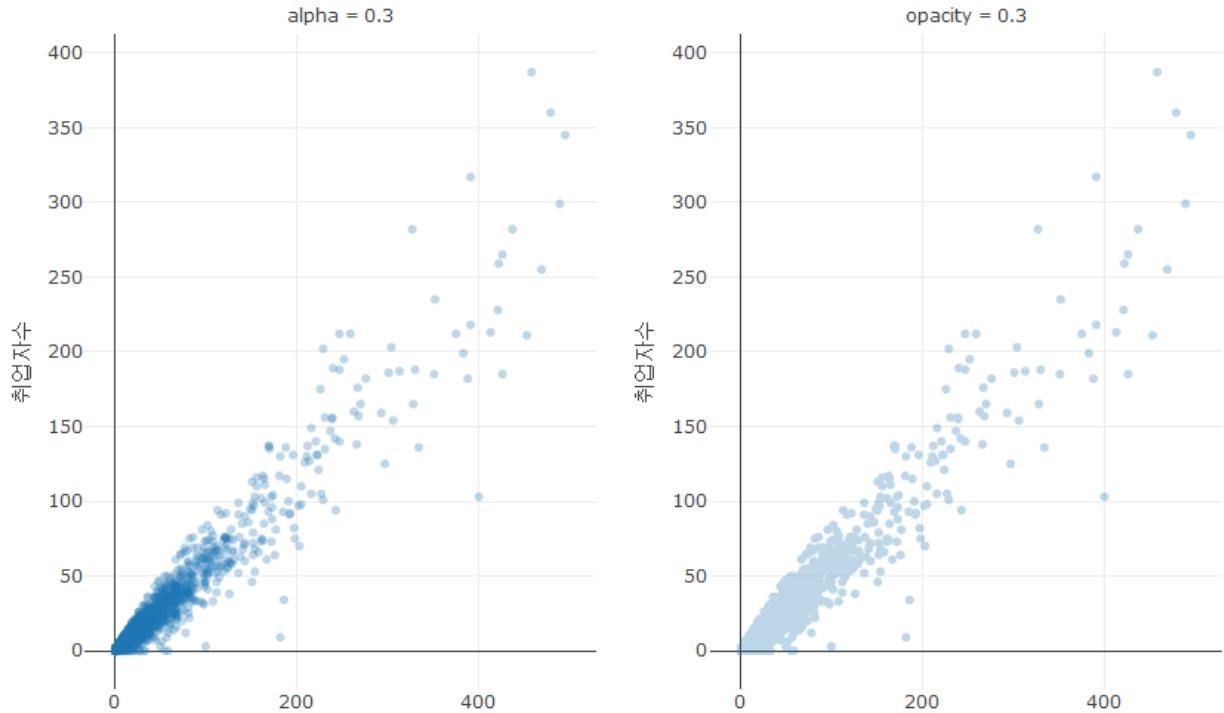
- R

R 의 경우에는 투명도 조절에 'opacity'와 'alpha'의 두가지 속성이 사용된다. 아래의 두 시각화는 R 에서 'opacity'와 'alpha'와의 차이를 보여준다. 'alpha'는 각각의 색상 채널에 투명도가 적용되기 때문에 백그라운드 색의 영향을 받지만 'opacity'는 해당 채널에는 모두 같은 투명도가 적용되기 때문에 백그라운드의 영향을 받지 않는다. 하지만 'opacity'도 각각의 채널에 설정이 되면 'alpha'와 동일한 효과가 나온다.

```
df_취업률_2000 |>
  ## alpha 를 0.3 으로 설정
  plot_ly() |>
  add_trace(type = 'scatter', mode = 'markers',
            x = ~졸업자수, y = ~취업자수, alpha = 0.3)

df_취업률_2000 |>
  ## opacity 를 0.3 으로 설정
```

```
plot_ly() |>
  add_trace(type = 'scatter', mode = 'markers',
    x = ~졸업자수, y = ~취업자수, opacity = 0.3)
```



실행결과 2- 8. alpha 와 opacity 의 설정 결과

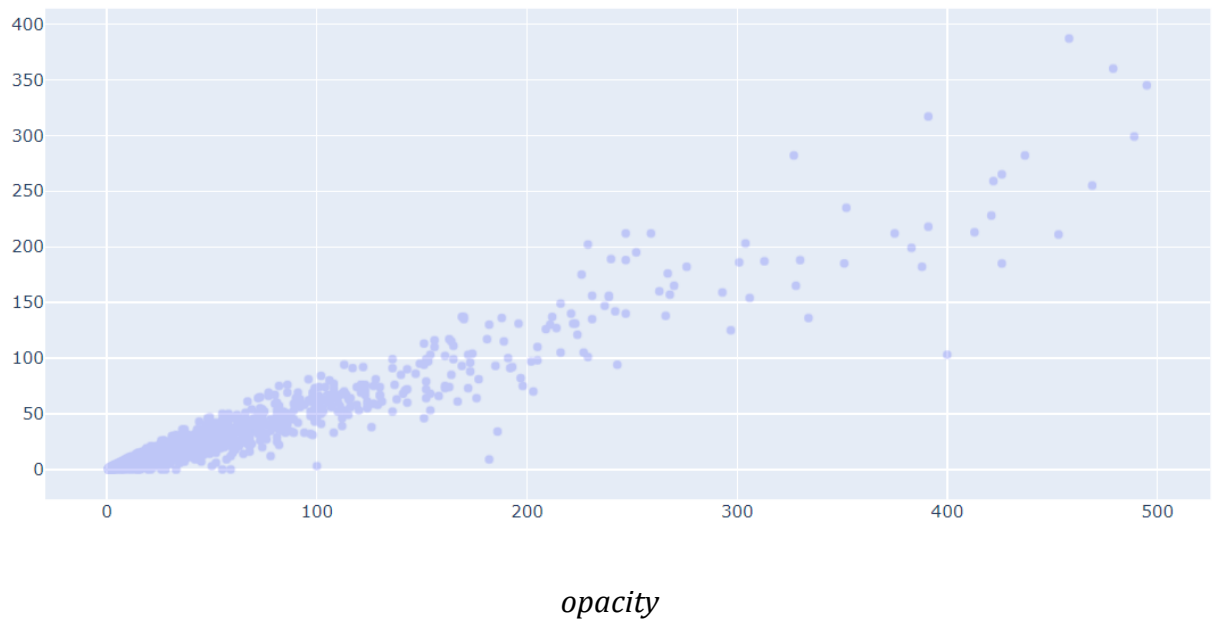
- python

python 의 경우는 투명도 설정에 'opacity' 속성을 사용한다. 하지만 R 과 같이 alpha 값을 사용할 수도 있는데 alpha 값은 색상의 RGB 값을 설정할때 RGBA 값을 사용하면 각각의 색상 채널에 따른 alpha 값을 설정할 수 있다.

```
fig = go.Figure()

fig.add_trace({
  'type' : 'scatter',
  'mode' : 'markers',
  'x': df_취업률_2000['졸업자수'],
  'y': df_취업률_2000['취업자수'],
  'opacity' : 0.3
})

fig.show()
```

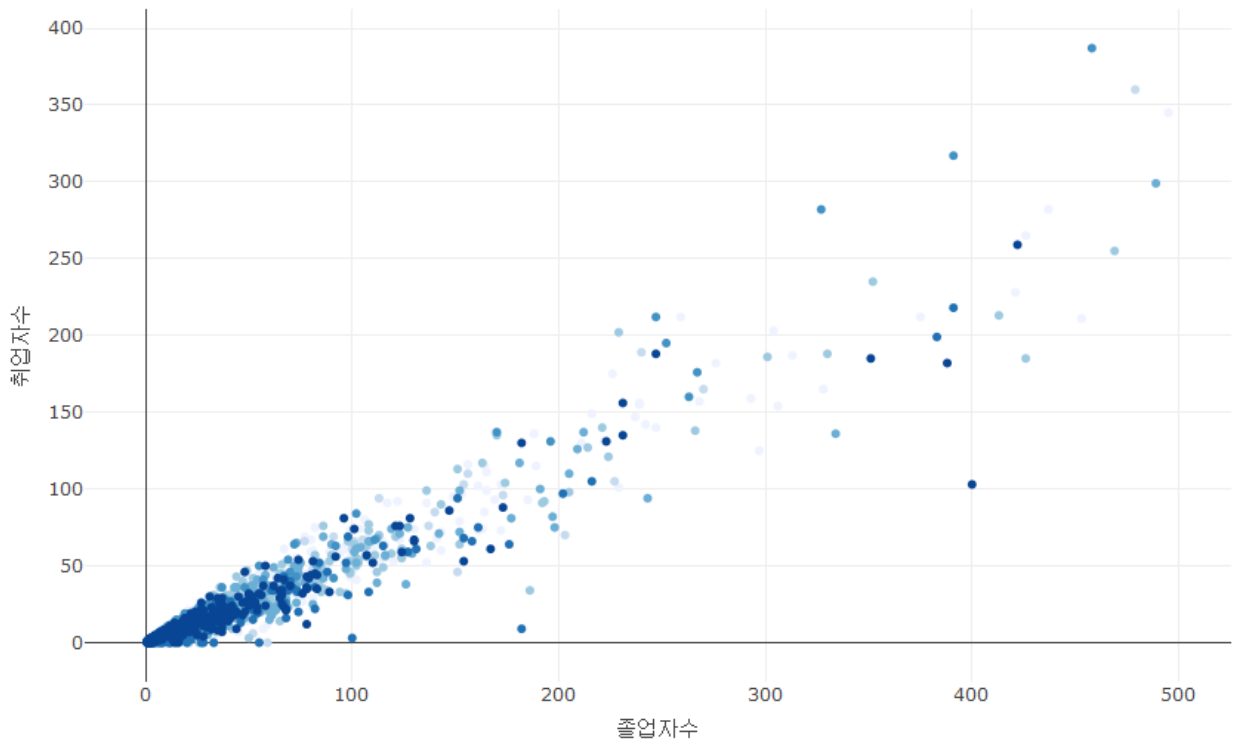


4.2.10. showlegend

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

- R

```
df_취업률_2000 |>
  plot_ly() |>
  add_trace(type = 'scatter', mode = 'markers',
            x = ~졸업자수, y = ~취업자수, name = ~대계열,
            ## showlegend 을 FALSE 로 설정
            showlegend = FALSE)
```

실행결과 2- 9. *showlegend* 설정 결과

- python

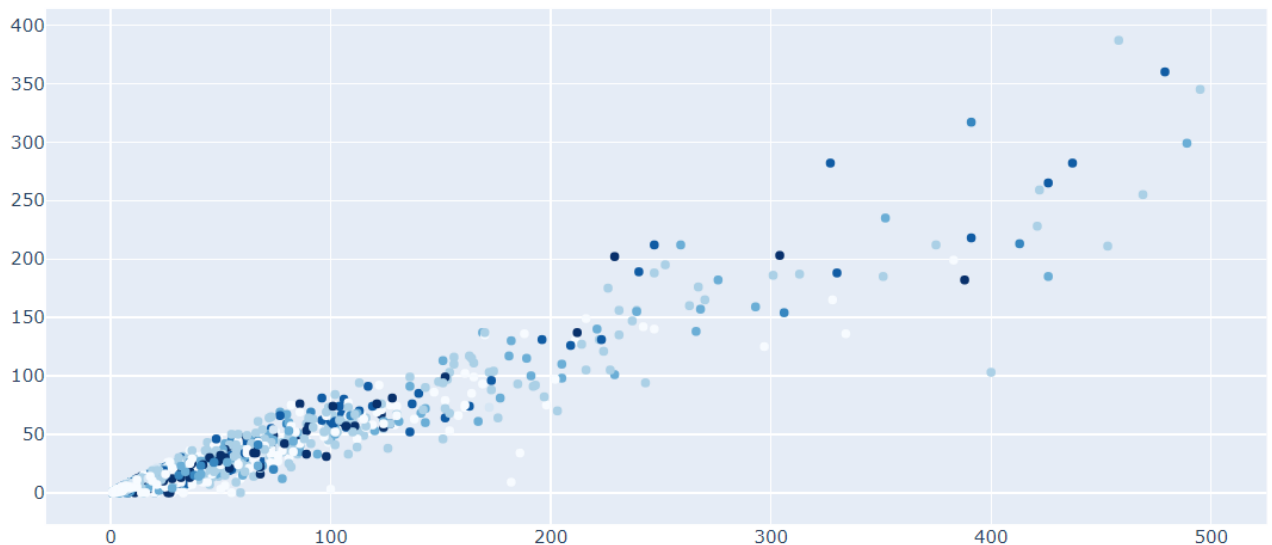
python 에서도 'showlegend' 속성을 'True' 또는 'False'로 설정함으로써 범례를 표시하거나 없앨 수 있다.

```
fig = go.Figure()

for 대계열, group in df_취업률_2000.groupby('대계열'):
    fig.add_trace({
        'type' : 'scatter',
        'mode' : 'markers',
        'x': group['졸업자수'],
        'y': group['취업자수'],
        'name' : 대계열,
        'showlegend' : False
    })

fig.show()
```

Opacity



4.3. layout

지금까지는 시각화를 만들기 위해 필요한 데이터를 트레이스로 표현하는 방법에 대해 알아보았다. `plotly` 에서 'data' 속성을 사용하여 데이터를 직접 표현하는 트레이스 외에 데이터와 직접적 관련이 없는 다양한 정보를 표현하는 속성들을 'layout' 속성이라고 한다. 'layout' 속성에는 시각화의 제목, 범례, 여백, 크기, 폰트, 축 등을 설정할 수 있다.

R 에서 'layout' 속성의 세부 속성을 설정하기 위해서는 `layout()` 을 사용한다. python 에서는 `plotly.graph_object` 를 사용하여 'layout' 을 설정하는데 `plotly.graph_object.Figure()` 에 'layout' 속성에 세부 속성들을 딕셔너리로 구성하거나 `plotly.graph_object.add_trace()` 로 트레이스를 추가한 후 `plotly.graph_object.update_layout()` 에 세부 속성에 대한 속성값을 할당하는 방법이 있다.

4.3.1. layout 공통 속성

4.3.1.1. title : 제목 설정

`plotly` 의 제목을 설정하는 type 은 'title'이다. 'title'의 세부 type 은 다음과 같다. 앞서 설명한 바와 같이 'title'은 다른 속성과 조금 다른 부분이 있다. 원칙적으로 'title'은 상위 속성으로 세부 속성들의 list 나 dict 로 구성해야 한다. 하지만 'title'에 바로 문자열을 설정하면 그 자체가 속성으로 사용되어 제목을 설정하는 속성이 된다.

'title'에서 사용하는 주요 하위 속성은 다음과 같다.

속성			속성 설명	속성값
title	font	color	제목 글자 색상 설정	문자열
		family	제목 글자 HTML 폰트 설정	폰트명
		size	제목 글자 크기 설정	1 이상의 수치
	pad	b	제목의 아래 패딩 여백 설정	수치
		l	제목의 왼쪽 패딩 여백 설정	수치
		r	제목의 오른쪽 패딩 여백 설정	수치
		t	제목의 윗쪽 패딩 여백 설정	수치
	text		제목의 텍스트 설정, 제목의 텍스트는 'title' 자체 속성으로도 설정 가능	문자열
	x		xref로부터의 x 축 위치 설정	0 부터 1 사이의 수치
	xanchor		제목의 수평 정렬 설정	"auto" "left" "center" "right"
	xref		x 축의 위치 설정 기준. Container 는 plot 의 전체, Paper 는 플로팅 영역	"container" "paper"
	y		xref로부터의 y 축 위치 설정	0 부터 1 사이의 수치
	yanchor		제목의 수직 정렬 설정	"auto" "top" "middle" "bottom"
	yref		y 축의 위치 설정 기준. Container 는 plot 의 전체, Paper 는 플로팅 영역	"container" "paper"

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

HTML tag	설명
	볼드체 설정
<i></i>	이탤릭체 설정

	줄 바꿈 설정
<sup></sup>	윗 첨자 설정
<sub></sub>	아래 첨자 설정
	하이퍼링크 설정

- R

R에서는 **add_trace()**로 트레이스를 추가한 후 **layout()**을 사용하여 'layout'의 세부 속성을 설정할 수 있다. 하나 주의해야 할 것은 **add_trace()**로 트레이스를 추가해 만든 **plotly** 객체를 대상으로 **layout()**을 실행해야 한다는 것이다.

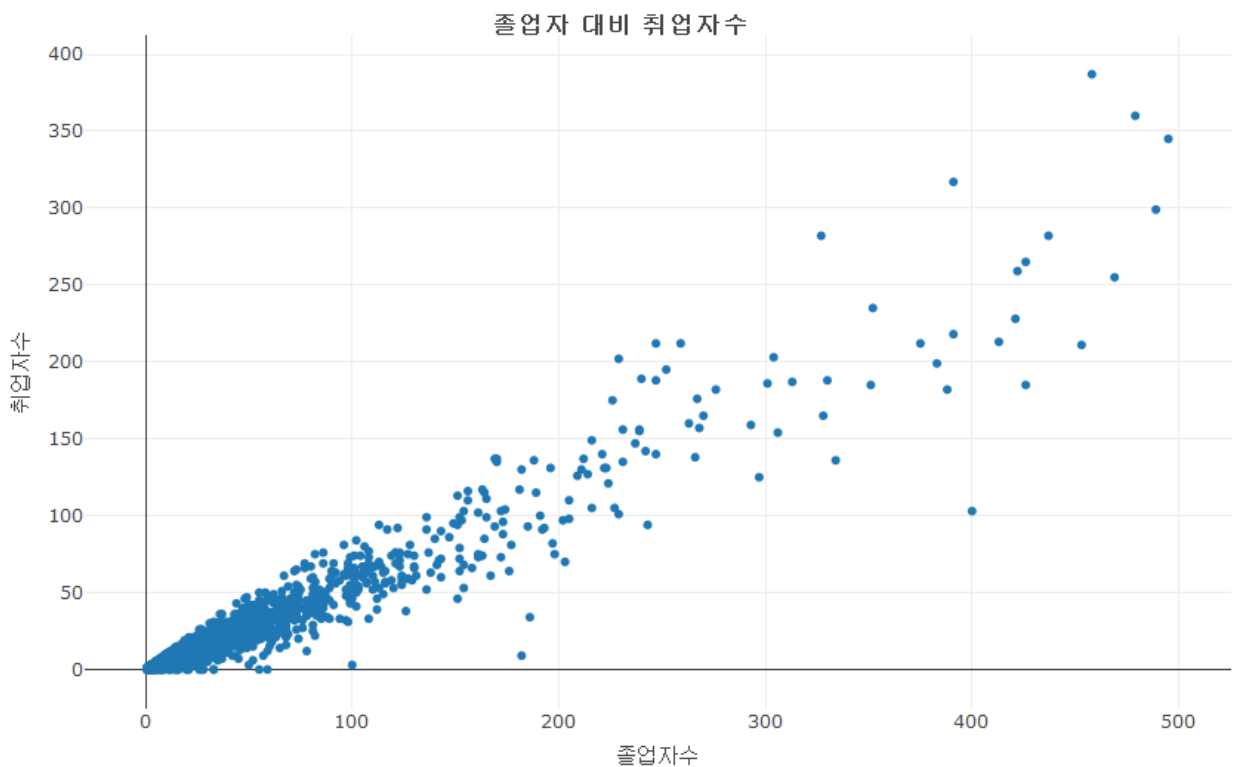
```

R_layout_scatter <- df_취업률_2000 |>
  filter(졸업자수 < 500) |>
  plot_ly() |>
  add_trace(type = 'scatter', mode = 'markers',
    x = ~졸업자수, y = ~취업자수)

R_layout_scatter <- R_layout_scatter |>
  layout(title = list(text = '<b>졸업자 대비 취업자수</b>',
    x = 0.5, xanchor = 'center', yanchor = 'top')
  )

R_layout_scatter

```

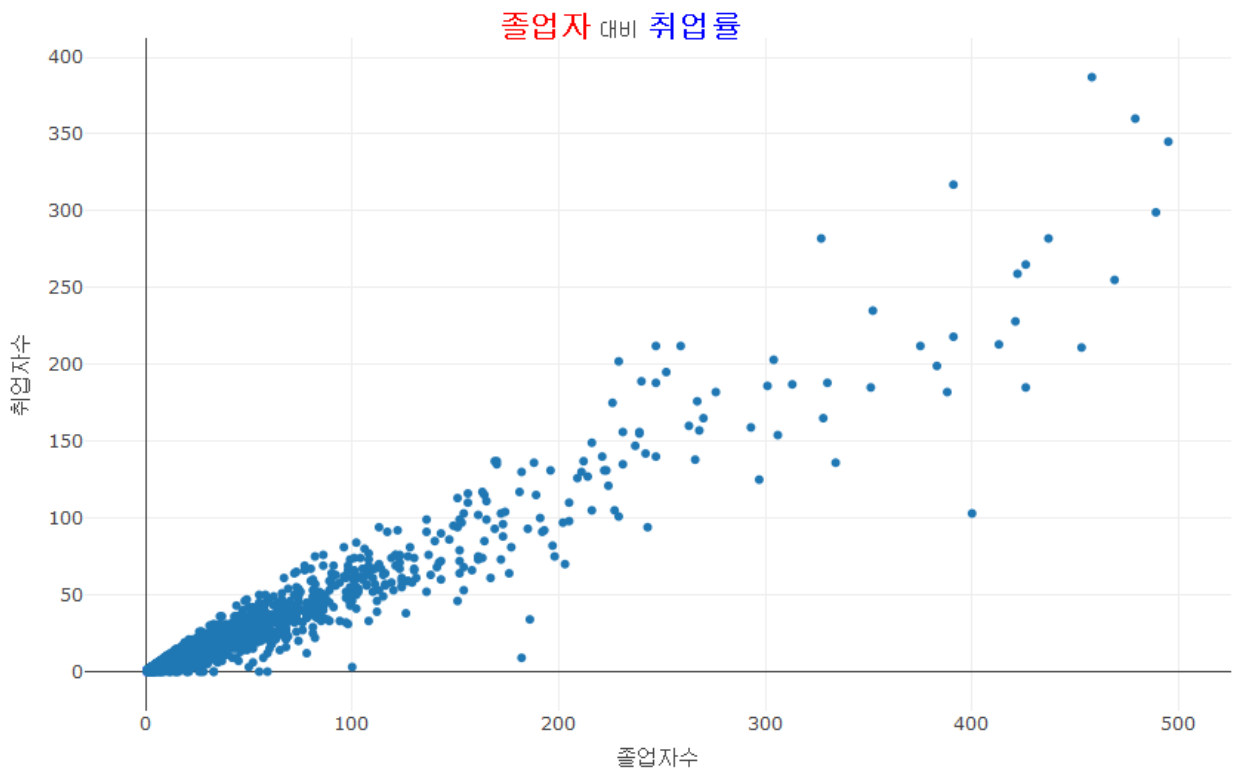


plotly 가 HTML 텍스트 tag 를 일부만 지원함으로써 문자열 스타일링에 한계가 있을 듯 하지만 ”을 사용하는 HTML inline 속성을 지원하기 때문에 CSS 의 스타일을 사용하여 문자열의 세부 설정이 가능하다. 다음은 HTML 의 inline 속성을 사용하여 제목을 설정하는 코드이다.

```

R_layout_scatter |> layout(title = list(text = "<span style = 'font-size:15pt'><span style = 'color:red;font-weight:bold;'> 졸업자</span><span style = 'font-size:10pt'> 대비</span> <span style = 'color:blue;font-weight:bold;'>취업률</span></span>",
  x = 0.5, xanchor = 'center', yanchor = 'top')
)

```



- python

앞서 설명했듯이 python 에서 plotly.graph_object 를 사용하여 'layout'을 설정하기 위해서는 plotly.graph_object 의 초기화 함수인 `Figure()`에서 'layout' 속성의 세부 속성들을 딕셔너리로 설정하거나 `add_trace()`로 트레이스를 설정하고 이 `plotly` 객체에 `update_layout()`을 사용한다.

`Figure()`를 사용하는 방법은 다음과 같다.

```
fig = go.Figure()

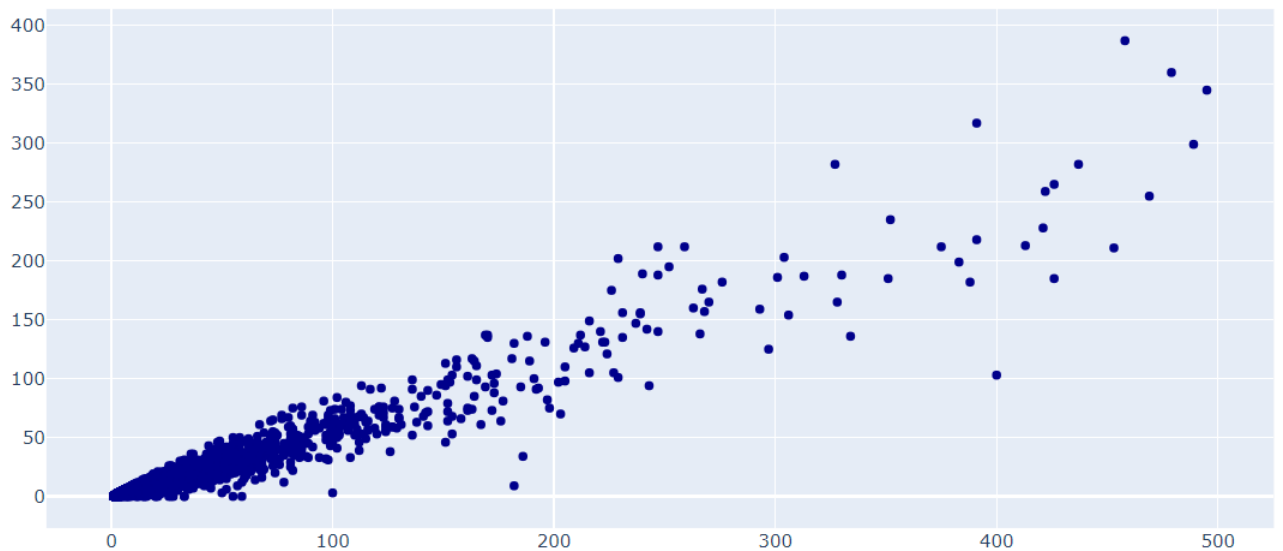
go.Figure(
    data = [
        {
            'type' : 'scatter',
            'mode' : 'markers',
            'x' : df_취업률_2000['졸업자수'],
            'y' : df_취업률_2000['취업자수'],
            'marker' : {
                'color' : 'darkblue'
            }
        }
    ],
    layout = {
        'title' : '졸업자 대비 취업률'
    }
)
```

```

    layout = {
        'title' : {'text' : "<b>졸업자 대비 취업자수</b>",
                    'x' : 0.5,
                    'xanchor' : 'center',
                    'yanchor' : 'top'
                  }
    }
)

```

졸업자 대비 취업자수



`update_layout()`을 사용하는 방법은 다음과 같다. `update_layout()`를 사용할 때 주의해야 하는 것은 'layout'에 바로 아래 하위 속성들은 딕셔너리로 설정하지 않고 `=`을 사용하여 할당하고 이들 하위 속성의 하위 속성이 존재할 때는 다시 딕셔너리로 구성하여야 한다..

```

fig_scatter = go.Figure()

fig_scatter.add_trace({
    'type' : 'scatter',
    'mode' : 'markers',
    'x' : df_취업률_2000['졸업자수'],
    'y' : df_취업률_2000['취업자수'],
    'marker' : {
        'color' : 'darkblue'
    }
})

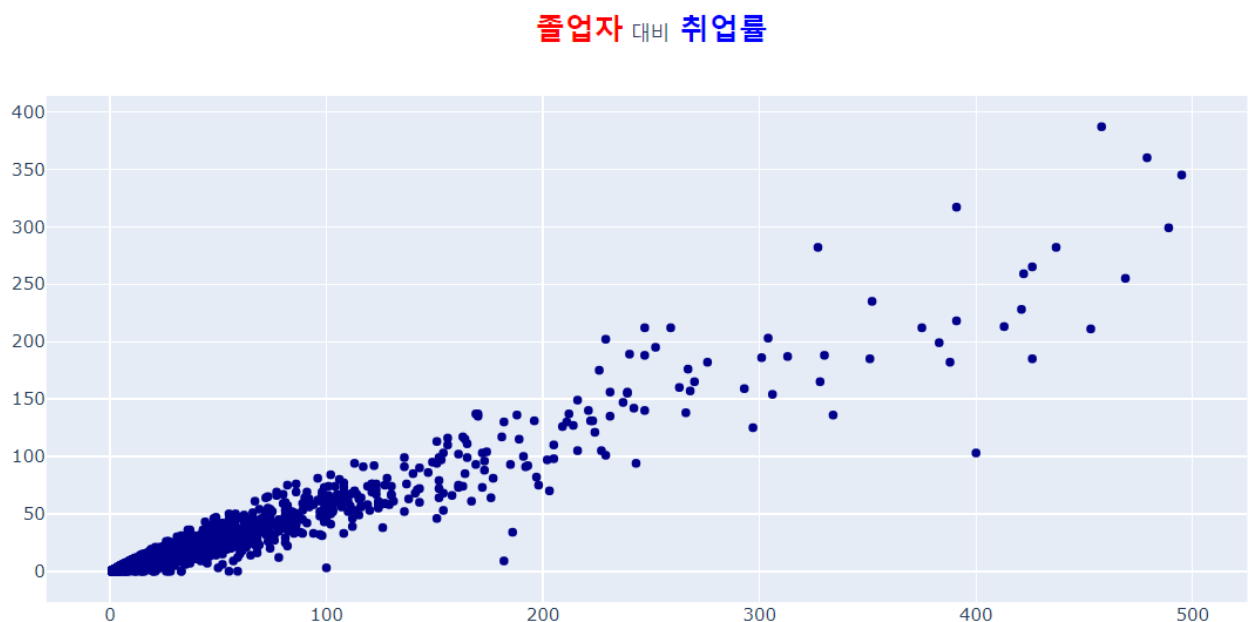
```

```
fig_scatter.update_layout(title = {'text' : "<b>졸업자 대비 취업자수</b>",
                                  'x' : 0.5, 'xanchor' : 'center',
                                  'yanchor' : 'top'})
```

python 도 ”을 사용하는 HTML inline 속성을 사용하여 CSS 의 스타일의 세부 설정이 가능하다. 다음의 코드는 앞의 코드와 동일하나 `add_trace()`에 속성 설정을 `go.Scatter()`를 사용하고 제목 문자열을 CSS inline 으로 설정한 코드이다.

```
fig_scatter_temp = go.Figure(fig_scatter)

fig_scatter_temp.update_layout(title = {'text' : "<span style = 'font-size:15pt'><span style = 'color:red;font-weight:bold;'> 졸업자</span><span style = 'font-size:10pt'> 대비</span> <span style = 'color:blue;font-weight:bold;'>취업률</span></span></span>",
                                  'x' : 0.5, 'xanchor' : 'center',
                                  'yanchor' : 'top'})
```



4.3.1.2. legend : 범례 설정

`layout()`에서 범례 설정과 관련된 속성은 'showlegend'와 'legend'이다.

'showlegend'는 범례를 표시하거나 삭제하는 속성인데 `add_trace()`에서도 설정이 가능하다. `add_trace()`에서 설정하면 해당 트레이스만 범례에서 삭제하고 `layout()`에서 설정하면 전체 범례를 삭제하게 된다.

'legend'는 범례의 세부 설정을 위한 세부 속성으로 구성된다. 'legend'로 설정이 가능한 주요 속성은 다음과 같다.

속성			속성 설명	속성값	
showlegend			범례를 표시할 것인지 설정	논리값	
legend	bgcolor		범례 배경색 설정	색상값	
	bordercolor		범례 경계선색 설정	색상값	
	borderwidth		범례 경계선 두께 설정	0 이상의 수치	
	entrywidth		범례의 두께 설정	0 이상의 수치	
	font	color	범례 문자색 설정	색상값	
		family	범례 문자 폰트 설정	폰트명	
		size	범례 문자 크기 설정	1 이상의 수치	
	orientation		범례의 표시 방향 설정	"v" "h"	
	title	font	color	범례 제목 폰트색 설정	색상값
			family	범례 제목 폰트 설정	폰트명
			size	범례 제목 크기 설정	1 이상의 수치
		side		범례 제목의 위치 설정	"top" "left" "top left"
		text		범례 제목 문자열 설정	문자열
	traceorder		범례 순서 설정	"reversed", "grouped", "reverse d+grouped", "normal"	
	valign		텍스트에 연관된 심볼의 수직 정렬 설정	"top" "middle" "bottom"	
x		범례의 X 축 위치	-2 에서 3 까지의 수치		
xanchor		범례의 수평 위치 앵커를 설정	"auto" "left" "center" "right"		
y		범례의 Y 축 위치	-2 에서 3 까지의 수치		
yanchor		범례의 수평 위치 앵커를 설정	"auto" "top" "middle" "bottom"		

- R

다음의 코드는 코로나 19 의 확진자 수를 넓은 형태의 데이터프레임에서 선형 차트로 그리고 있다. 각각의 대륙을 `add_trace()`로 추가하여 총 4 개의 트레이스를 추가하였고 이중 아시아 트레이스에는 'showlegend'를 FALSE 로 설정하여 아시아 트레이스의 범례를 제거하였다. 또 'layout'에서 제목과 범례의 세부 설정을 해주었다.

```
R_layout_line <- df_covid19_100_wide |>
plot_ly() |>
add_trace(type = 'scatter', mode = 'lines',
```



```

x = ~date, y = ~확진자_한국, name = '한국')

R_layout_line <- R_layout_line |>
  add_trace(type = 'scatter', mode = 'lines',
    x = ~date, y = ~확진자_아시아, name = '아시아', showlegend = FALSE)

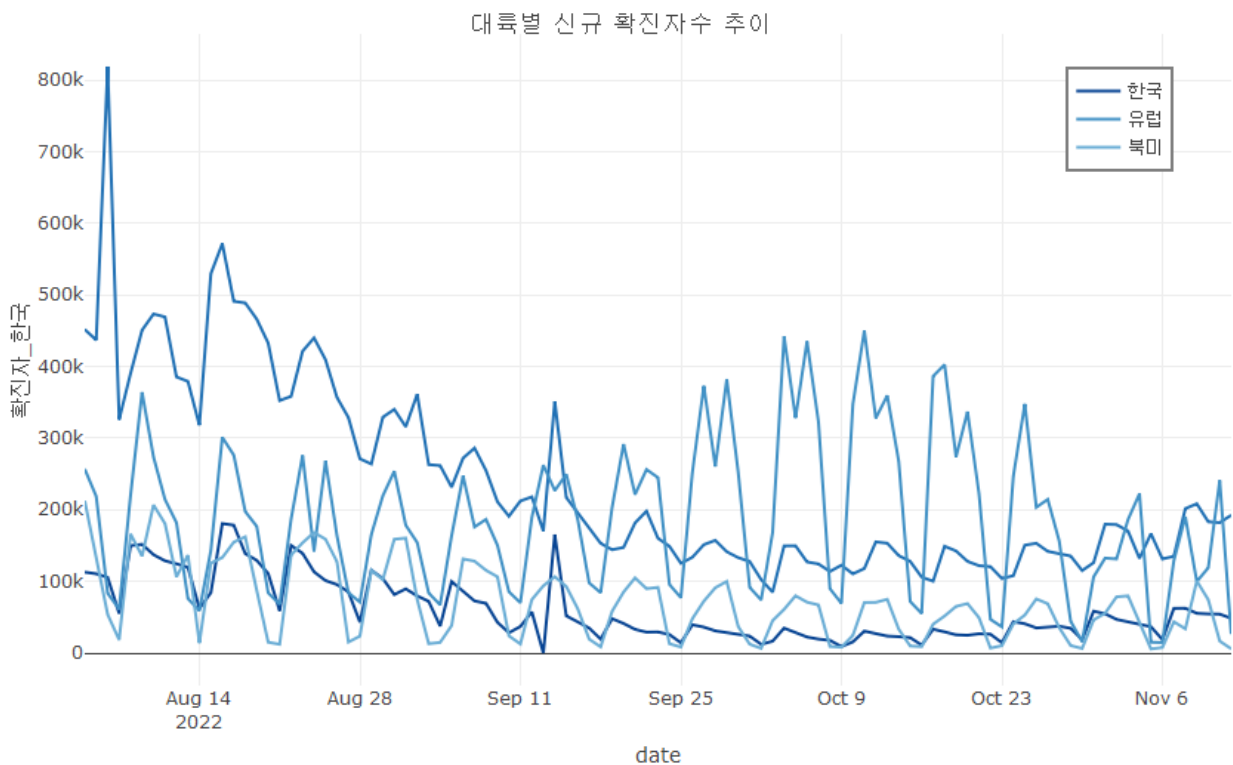
R_layout_line <- R_layout_line |>
  add_trace(type = 'scatter', mode = 'lines',
    x = ~date, y = ~확진자_유럽, name = '유럽')

R_layout_line <- R_layout_line |>
  add_trace(type = 'scatter', mode = 'lines',
    x = ~date, y = ~확진자_북미, name = '북미')

R_layout_line <- R_layout_line |>
  layout(title = list(text = '<b>대륙별 신규 확진자수 추이</b>',
    x = 0.5, xanchor = 'center', yanchor = 'top'),
    legend = list(orientation = 'v', bordercolor = 'gray', borderwidth = 2,
    x = 0.95, y = 0.95, xanchor = 'right')
  )

R_layout_line

```



- python

다음의 코드는 코로나 19 의 확진자 수를 넓은 형태의 데이터프레임에서 선형 차트로 그리고 있다. 각각의 대륙을 `add_trace()` 로 추가하여 총 4 개의 트레이스를 추가하였고 이중 아시아

트레이스에는 'showlegend'를 False 로 설정하여 아시아 트레이스의 범례를 제거하였다.
또 'layout'에서 제목과 범례의 세부 설정을 해주었다.

```
fig_line = go.Figure()

fig_line.add_trace(go.Scatter(
    type = 'scatter', mode = 'lines',
    x = df_covid19_100_wide.index,
    y = df_covid19_100_wide['확진자_한국'],
    name = '한국'
))

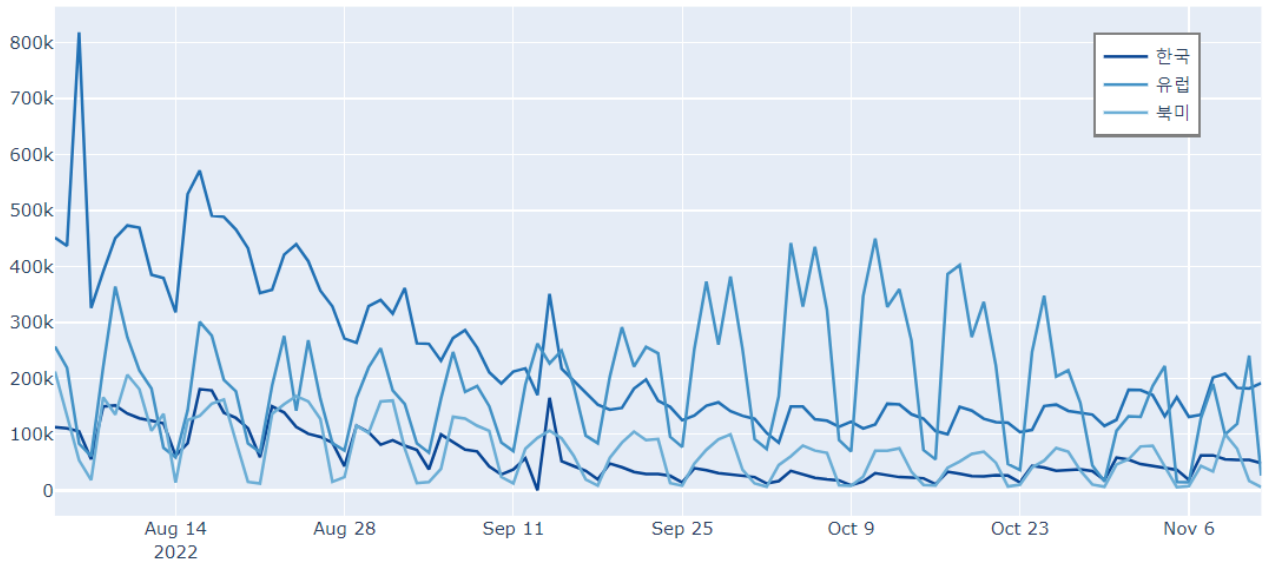
fig_line.add_trace(go.Scatter(
    type = 'scatter', mode = 'lines',
    x = df_covid19_100_wide.index,
    y = df_covid19_100_wide['확진자_아시아'],
    name = '아시아', showlegend = False
))

fig_line.add_trace(go.Scatter(
    type = 'scatter', mode = 'lines',
    x = df_covid19_100_wide.index,
    y = df_covid19_100_wide['확진자_유럽'],
    name = '유럽'
))

fig_line.add_trace(go.Scatter(
    type = 'scatter', mode = 'lines',
    x = df_covid19_100_wide.index,
    y = df_covid19_100_wide['확진자_북미'],
    name = '북미'
))

fig_line.update_layout(
    title = dict(text = '대륙별 신규 확진자수 추이', x = 0.5,
                 xanchor = 'center', yanchor = 'top'),
    legend = dict(orientation = 'v', bordercolor = 'gray', borderwidth = 2,
                  x = 0.95, y = 0.95, xanchor = 'right'),
    showlegend = True)
```

대륙별 신규 확진자수 추이



4.3.1.3. margin : 여백 설정

'layout'의 하위 속성 중 에서 여백의 설정과 관련된 속성은 'margin'이다. 'margin'으로 설정되는 여백은 플롯이 표시되는 면적과 전체 플롯 경계선과의 여백을 말한다. 여백 설정에 사용되는 세부 속성은 다음과 같다.

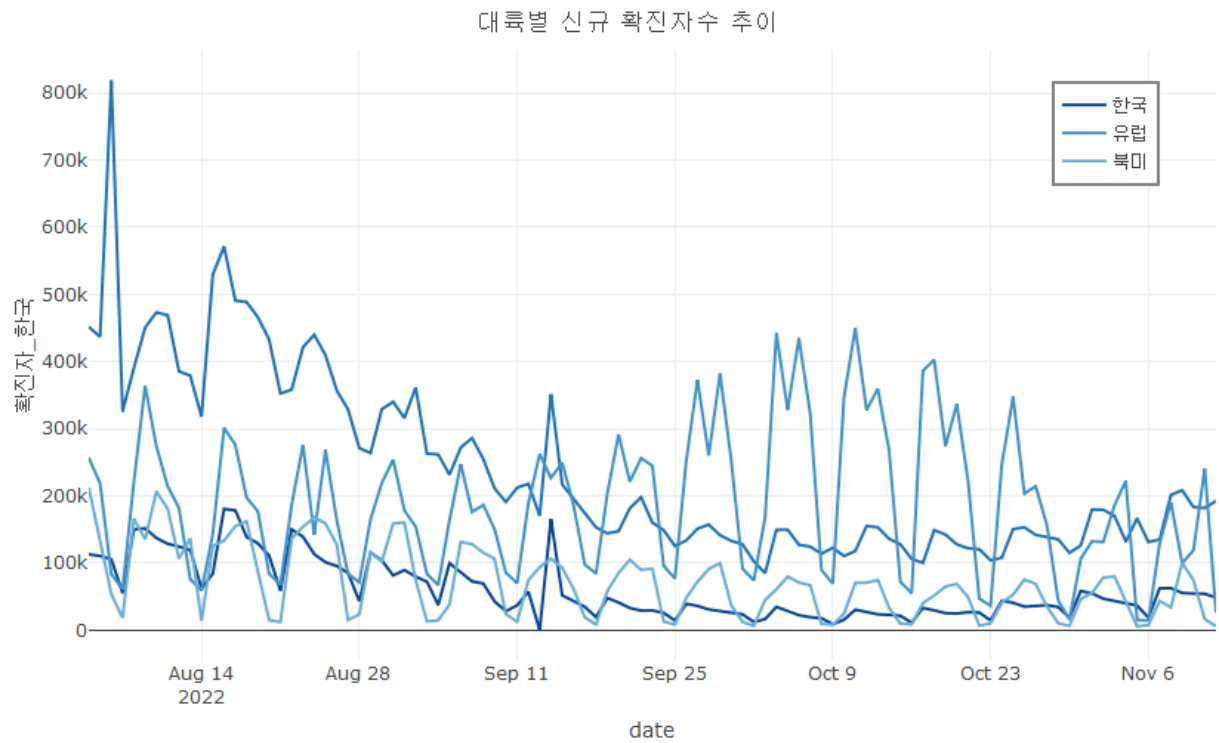
속성		속성 설명	속성값
margin	b	아래쪽 여백 설정	0 이상의 수치
	l	왼쪽 여백 설정	0 이상의 수치
	r	오른쪽 여백 설정	0 이상의 수치
	t	위쪽 여백 설정	0 이상의 수치

- R

R 에서 여백의 설정은 다음과 같다.

```
R_layout_line <- R_layout_line |>
  layout(margin = list(t = 50, b = 25, l = 25, r = 25))

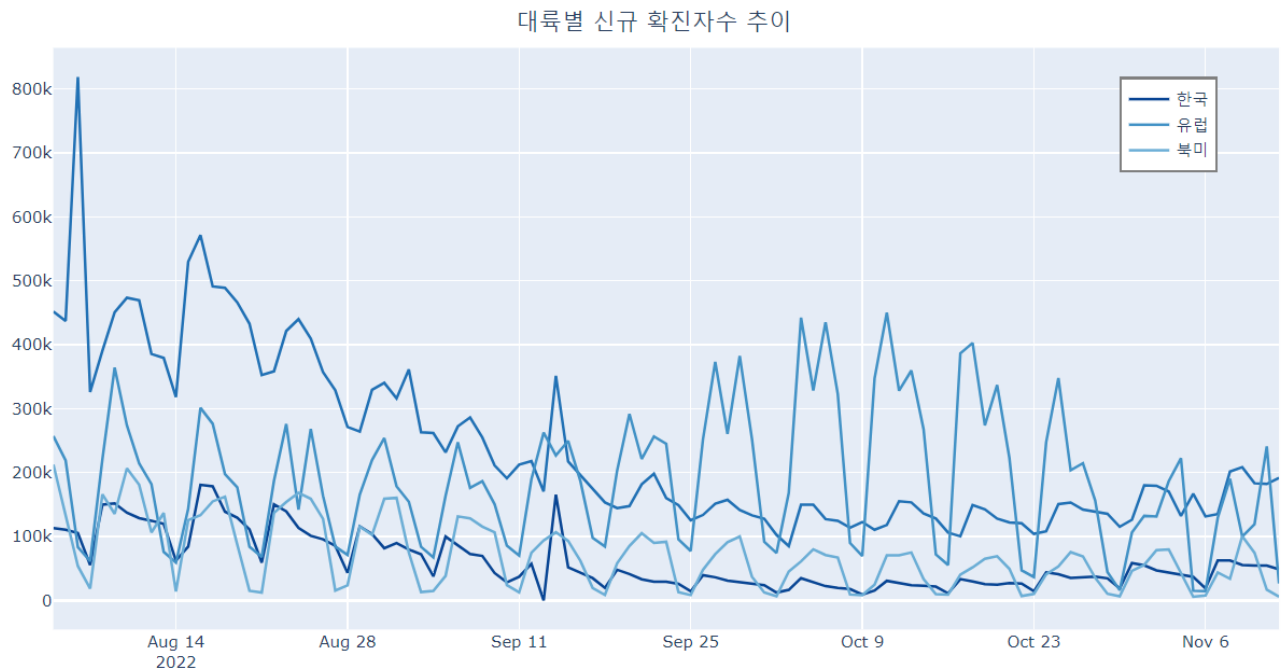
R_layout_line
```



- python

python 에서 여백의 설정은 다음과 같다.

```
fig_line.update_layout(
    margin = dict(t = 50, b = 25, l = 25, r = 25)
)
```



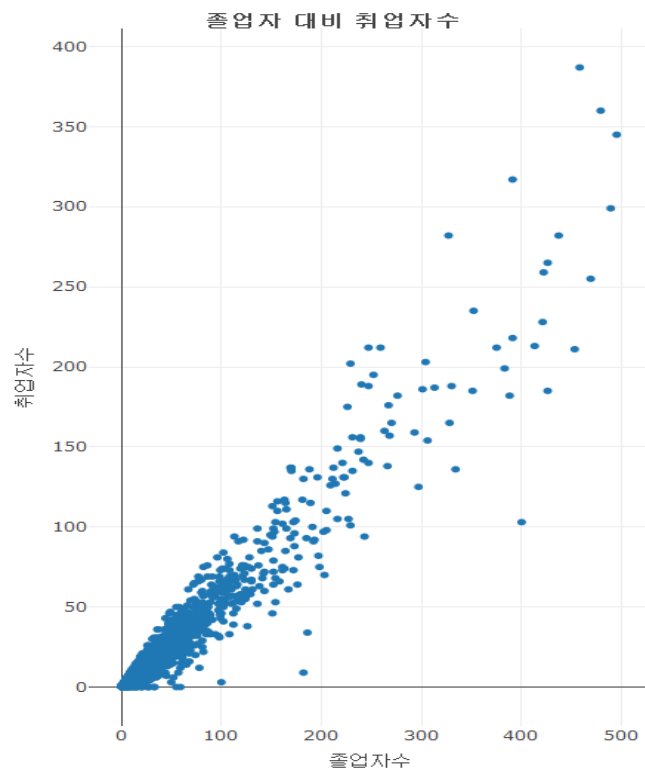
4.3.1.4. size : 플롯 크기 설정

'layout'에서 전체 플롯 크기의 설정을 위해서는 'autosize', 'height', 'width'의 세 가지 속성이 주로 사용된다. 'autosize'는 사용자가 정의하지 않은 레이아웃의 너비 또는 높이를 자동적으로 설정하는 논리값을 설정한다. 그리고 사용자가 크기를 결정하려면 'height'와 'width'는 전체 플롯 레이아웃의 높이와 너비를 설정하는 속성이다.

속성	속성 설명	속성값
width	플롯의 너비 설정	10 이상의 수치
height	플롯의 높이 설정	10 이상의 수치

- R

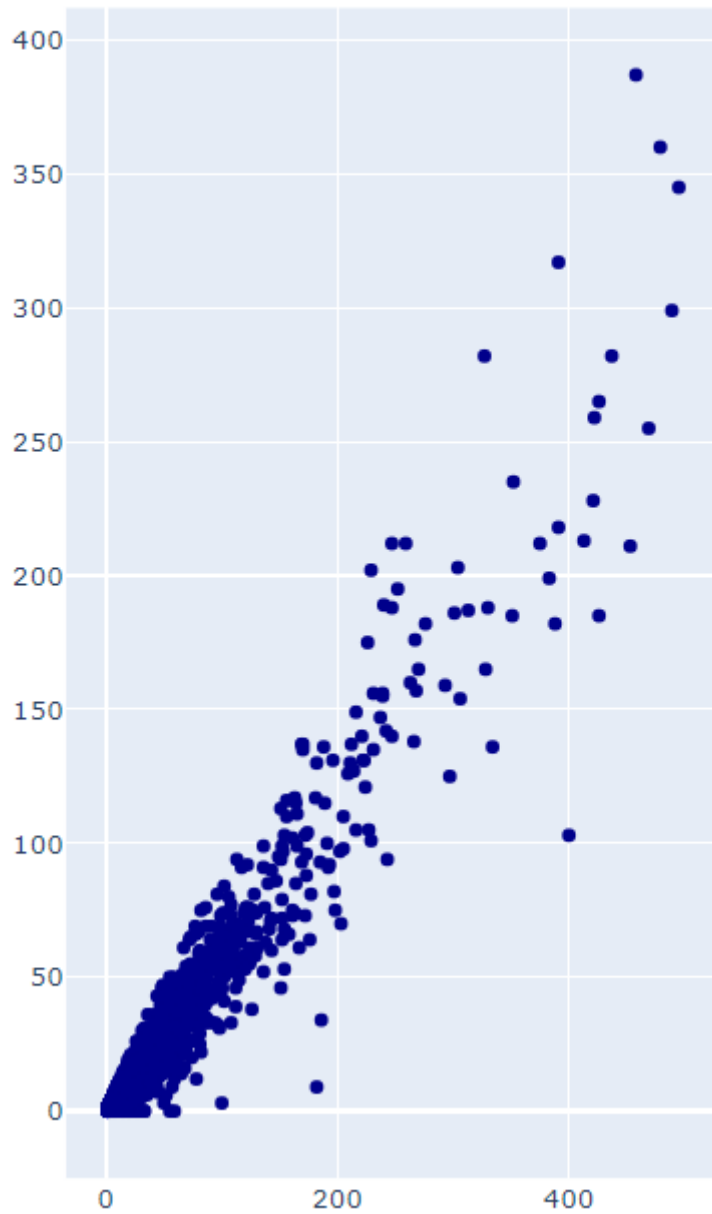
```
R_layout_scatter |>
  layout(width = 450, height = 700)
```



- python

```
fig_scatter_temp = go.Figure(fig_scatter)
fig_scatter_temp.update_layout(width = 450, height = 700)
```

졸업자 대비 취업자수



4.3.1.5. font : 폰트 설정

'layout'에서 플롯 전체적인 문자열의 설정과 관련된 속성은 'font'이다. 'font'는 다음과 같은 세 개의 세부 속성을 설정할 수 있다.

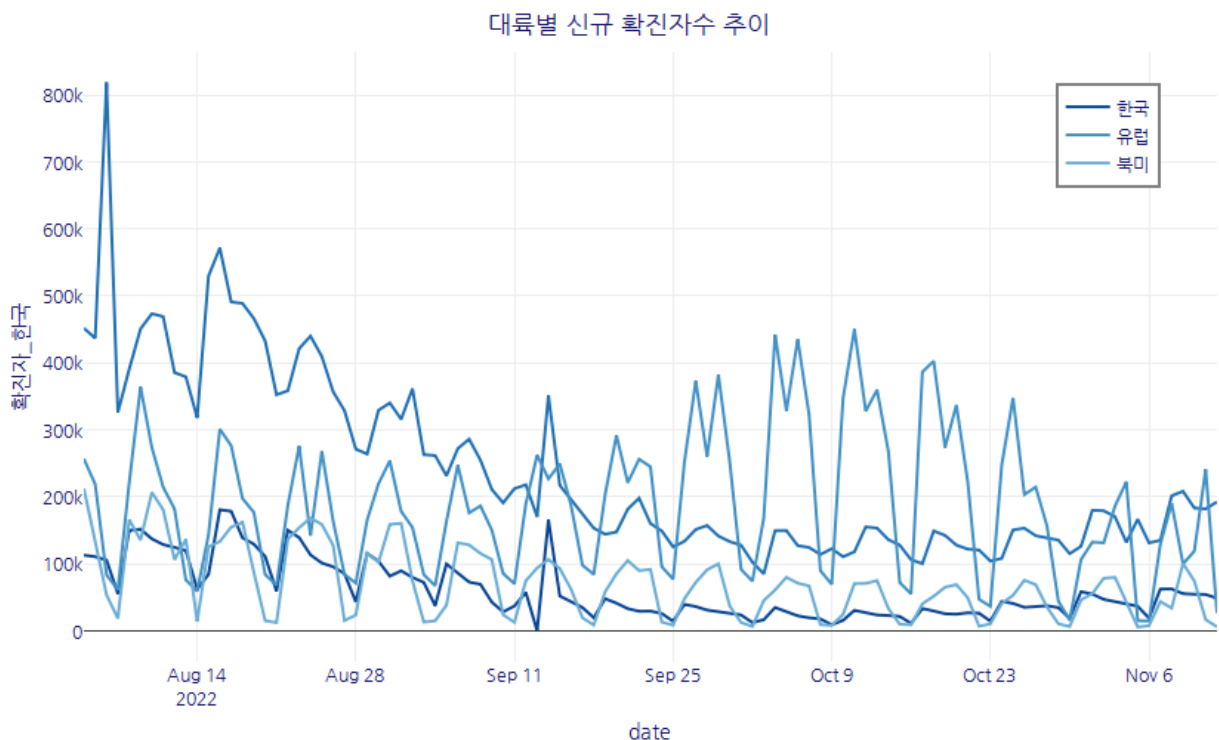
속성		속성 설명	속성값
font	color	플롯 전체 글자 색상 설정	문자열

속성		속성 설명	속성값
	family	플롯 전체 글자 HTML 폰트 설정	폰트명
	size	플롯 전체 글자 크기 설정	1 이상의 수치

- R

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.

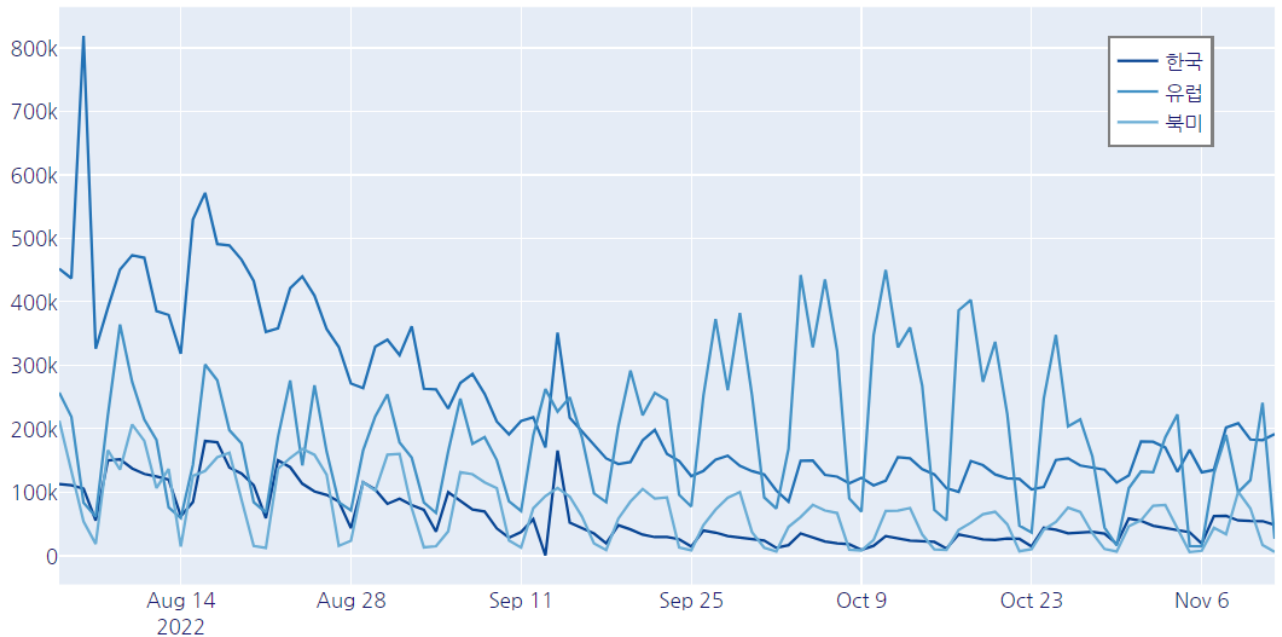
```
R_layout_line |>
  layout(font = list(family = "나눔고딕", color = 'MidnightBlue', size = 12))
```



- python

```
fig_line.update_layout(font = dict(family = "나눔고딕", color = 'MidnightBlue', size = 15))
```


대륙별 신규 확진자수 추이



4.3.1.6. color : 색 설정

'layout'에서 플롯의 배경색이나 플롯에서 사용되는 전반적인 색 스케일을 설정하는 속성은 다음과 같다.

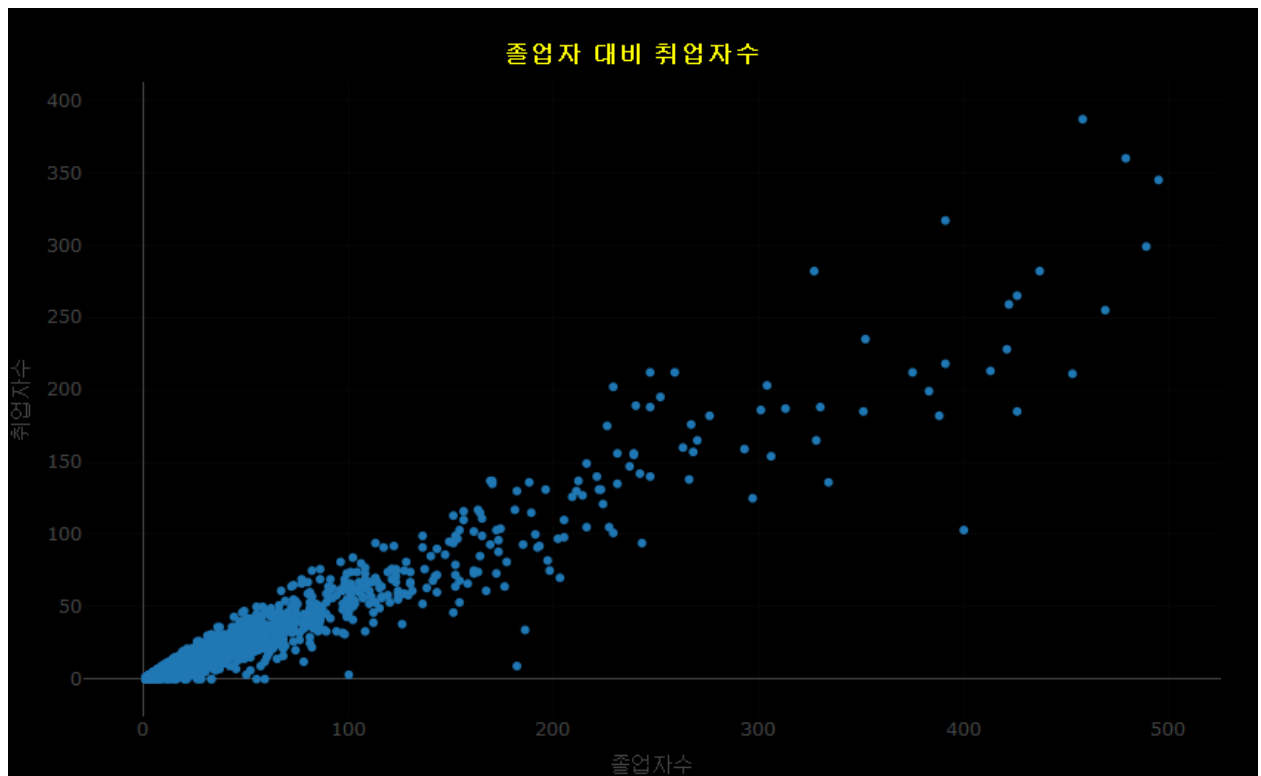
속성	속성 설명	속성값
paper_bgcolor	플롯이 그려지는 용지의 배경색을 설정	색상값
plot_bgcolor	x 축과 y 축 사이의 플로팅 영역의 배경색 설정	색상값

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.⁵

• R

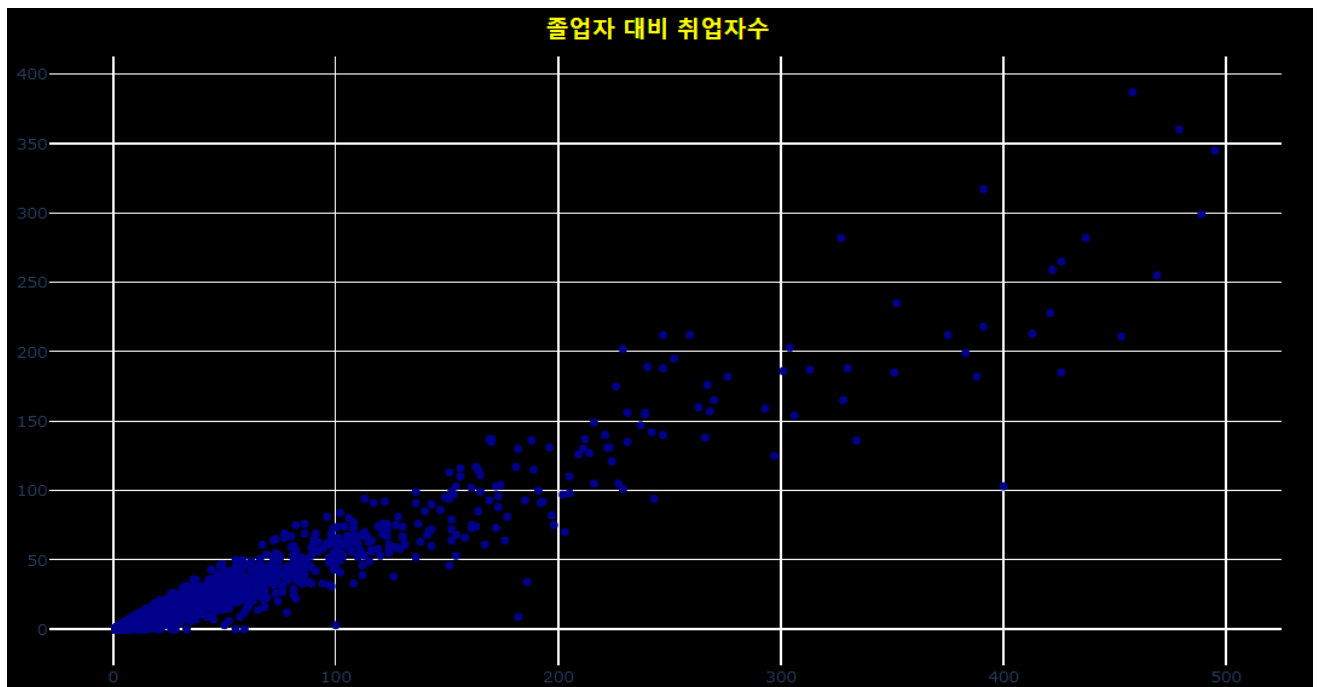
```
R_layout_scatter <- R_layout_scatter |>
  layout(paper_bgcolor = 'black', plot_bgcolor = 'black',
    title = list(font = list(color = 'yellow')),
    margin = list(t = 50, b = 25, l = 25, r = 25))
```

⁵ https://www.w3schools.com/cssref/css_colors.asp



- python

```
fig_scatter.update_layout(paper_bgcolor = 'black', plot_bgcolor = 'black',  
                           title = dict(font = dict(color = 'yellow')),  
                           margin = dict(t = 50, b = 25, l = 25, r = 25))
```



4.3.1.7. axis, yaxis : 축 설정

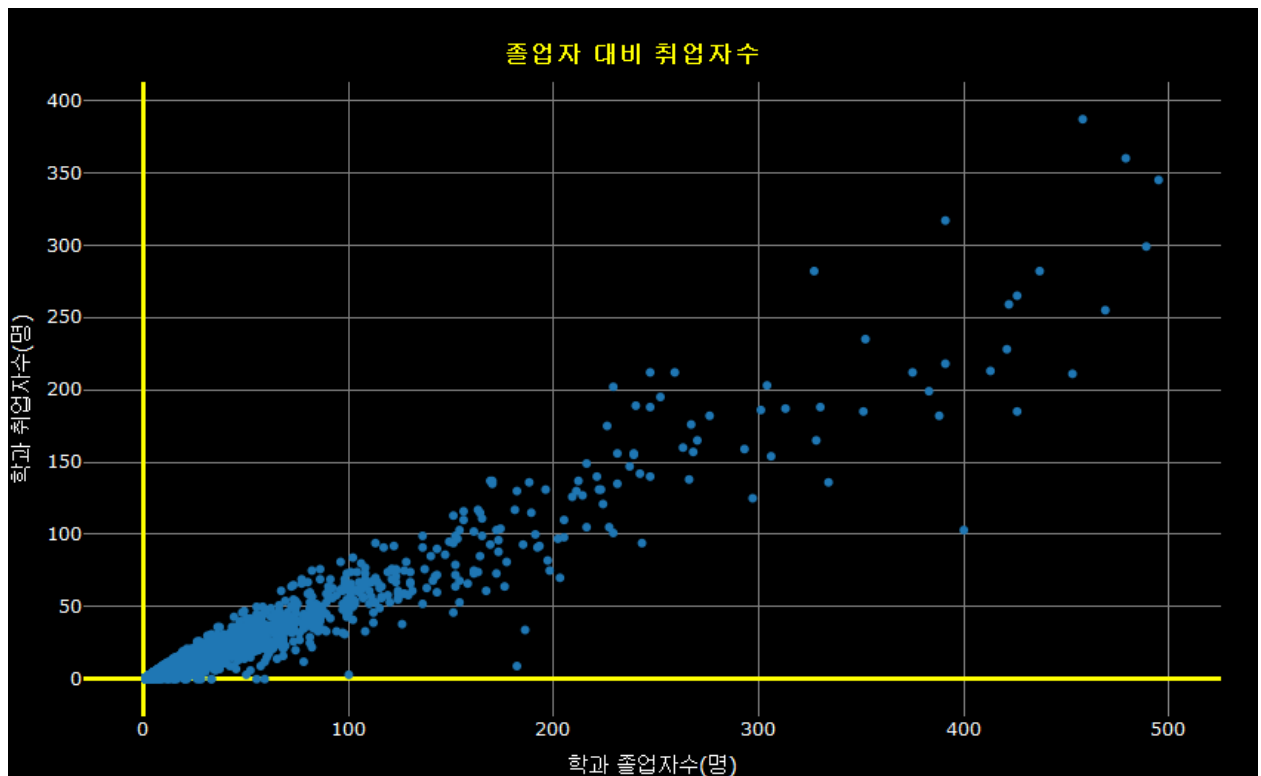
plotly 에서 축은 트레이스에 따라 매우 다른 이름으로 불린다. X, Y 축을 사용하는 2 차원의 데카르트 좌표계에서는 'axis', 3 차원 trace 에서는 'scene', 극 좌표계에서는 'polar', 삼각축 좌표계(ternary)에서는 'ternary', 지형(Geo) 좌표계에서는 'geo', Mapbox 좌표계에서는 'mapbox', 색 좌표계에서는 'coloraxis'로 사용된다. 이 중 2 차원 데카르트 좌표계에서 사용되는 'xaxis'와 'yaxis'의 주요 속성은 다음과 같다.

속성	속성 설명	속성값
color	축과 관련된 전체 색상 설정	색상값
dtick	눈금간의 간격 설정	수치나 카테고리 문자열
gridcolor,	그리드 라인 색상 설정	색상값
gridwidth	그리드 라인 두께 설정	0 이상의 수치
layer	축이 표시되는 레이어 설정	"above traces" "below traces"
linecolor	축 선 색상 설정	색상값
linewidth	축 선 두께 설정	0 이상의 수치
nticks	축에 표시되는 눈금의 최대치 설정	0 이상의 수치
range	축 범위 설정	list
rangemode	축 범위 모드 설정, tozero 는 0 까지 확장, nonnegative 는 양수 범위만 설정	"normal" "tozero" "nonnegative"

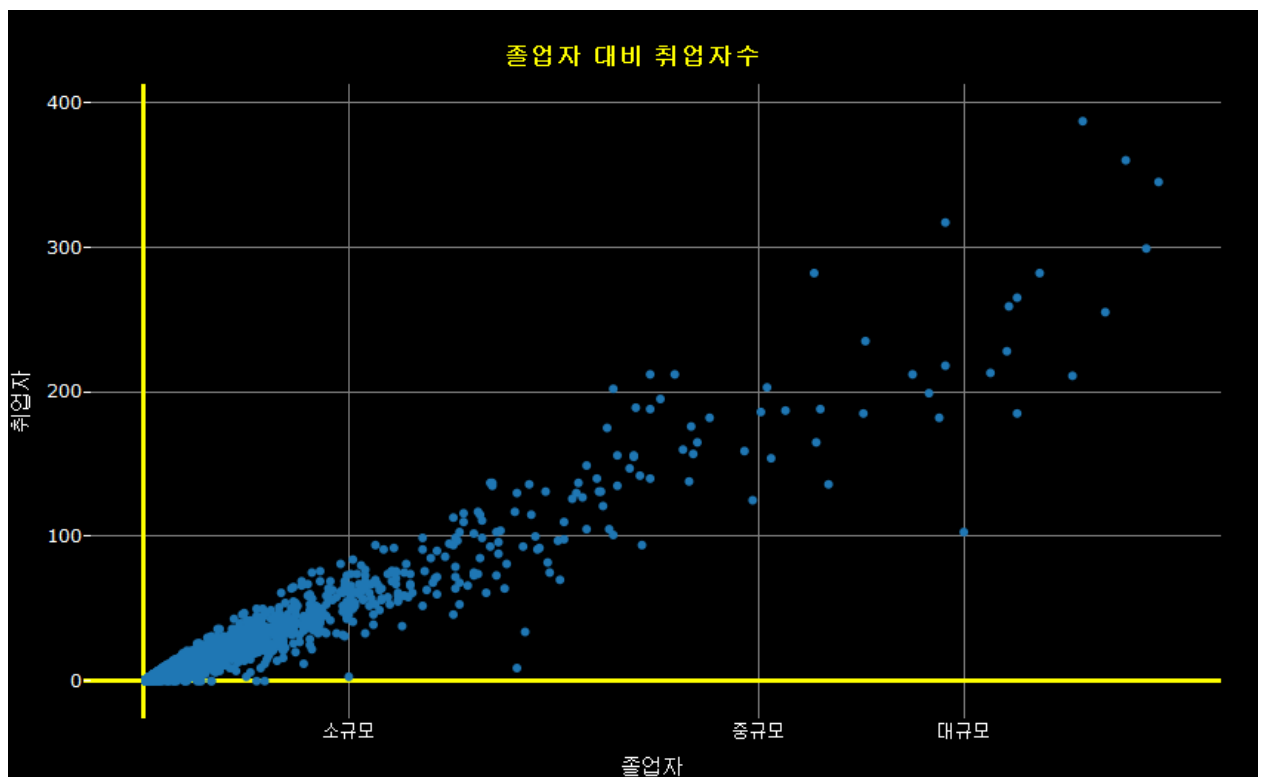
속성			속성 설명	속성값
showgrid			그리드를 표시할지 설정	논리값
tick0			축에서 첫번째 눈금이 표시될 값 설정	수치나 카테고리 문자열
tickangle			눈금 라벨의 각도 설정	각도값
tickcoor			눈금색 설정	색상값
tickfont	color			
	family			
	size			
tickformat			눈금 라벨의 d3 format 형태의 포맷 설정	문자열
ticklabelposition			눈금 라벨의 위치 설정	"outside" "inside" "outside top" "inside top" "outside left" "inside left" "outside right" "inside right" "outside bottom" "inside bottom"
ticklen			눈금 길이 설정	0 이상의 수치
tickmode			눈금 표시 모드 설정	"auto" "linear" "array"
ticks			눈금이 표시될지, 어디에 표시될지 설정	"outside" "inside" ""
ticktext			tickvals 에 매핑되어 표시될 눈금 라벨 설정	list, numpy array, or Pandas series of numbers, strings, or datetimes
tickvals			축에 표시될 눈금값 설정	list, numpy array, or Pandas series of numbers, strings, or datetimes
tickwidth			눈금 두께 설정	0 이상의 수치
title	font	color	축 제목 글자 색상 설정	문자열
		family	축 제목 글자 HTML 폰트 설정	폰트명
		size	축 제목 글자 크기 설정	1 이상의 수치
	text		축 제목 설정	문자열
type			축 타입 설정	"-" " "linear" "log" "date" "category" "multicategory"
zeroline			0 값이 표시되는 축선을 표시할지 설정	논리값
zerolinecolor			0 값이 표시되는 축선색 설정	색상값
zerolinewidth			0 값이 표시되는 축선두께 설정	수치

- R
-

```
R_layout_scatter <- R_layout_scatter |>
  layout(xaxis = list(title = list(text = '학과 졸업자수(명)'), color = 'white',
    zerolinecolor = 'yellow', zerolinewidth = 3,
    gridcolor = 'grey', gridwidth = 1), ## 정상적 방법
    yaxis = list(title = list(text = '학과 취업자수(명)'), color = 'white',
    zerolinecolor = 'yellow', zerolinewidth = 3,
    gridcolor = 'grey', gridwidth = 1, griddash = 'dot') ## 약식 방법
  )
R_layout_scatter
```

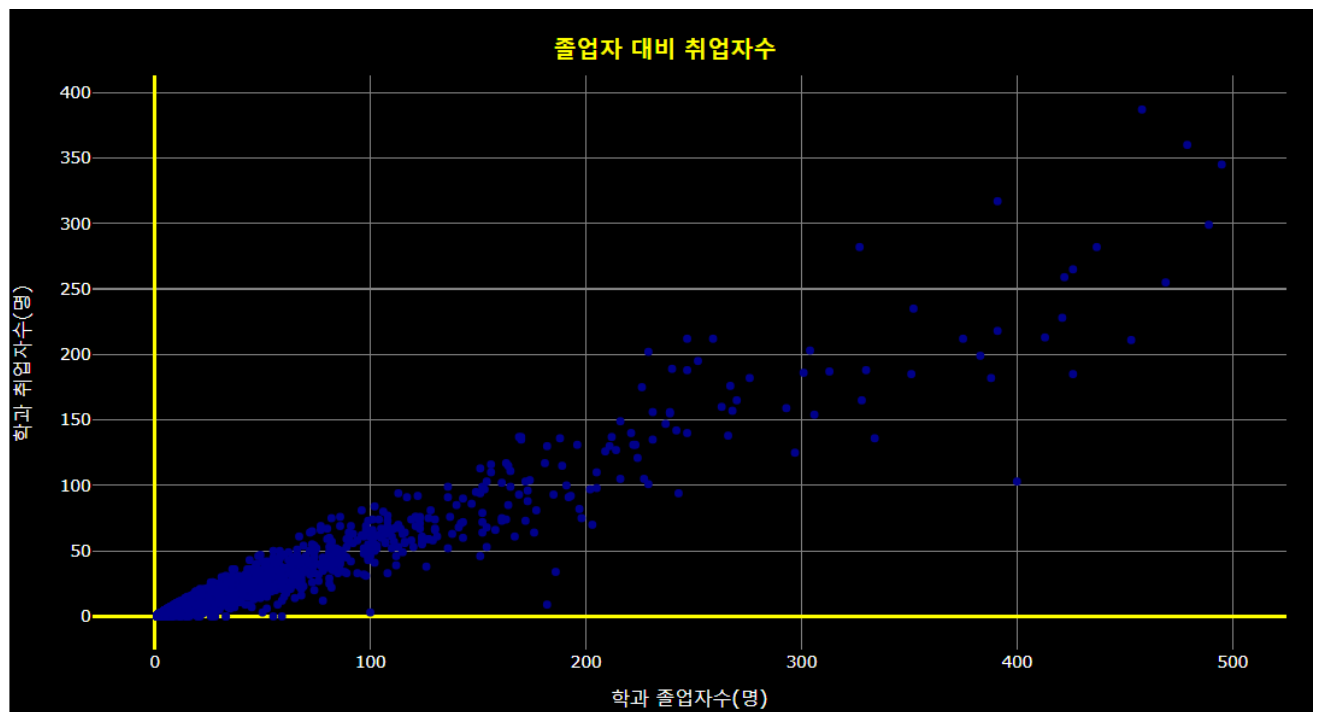


```
R_layout_scatter <- R_layout_scatter |>
  layout(xaxis = list(title = list(text = '졸업자'),
    tickmode = 'array',
    ticktext = c('소규모', '중규모', '대규모'),
    tickvals = c(100, 300, 400)
  ), ## 정상적 방법
    yaxis = list(title = '취업자', ticks="inside", tick0 = 100, dtick = 100) ## 약식 방법
  )
R_layout_scatter
```

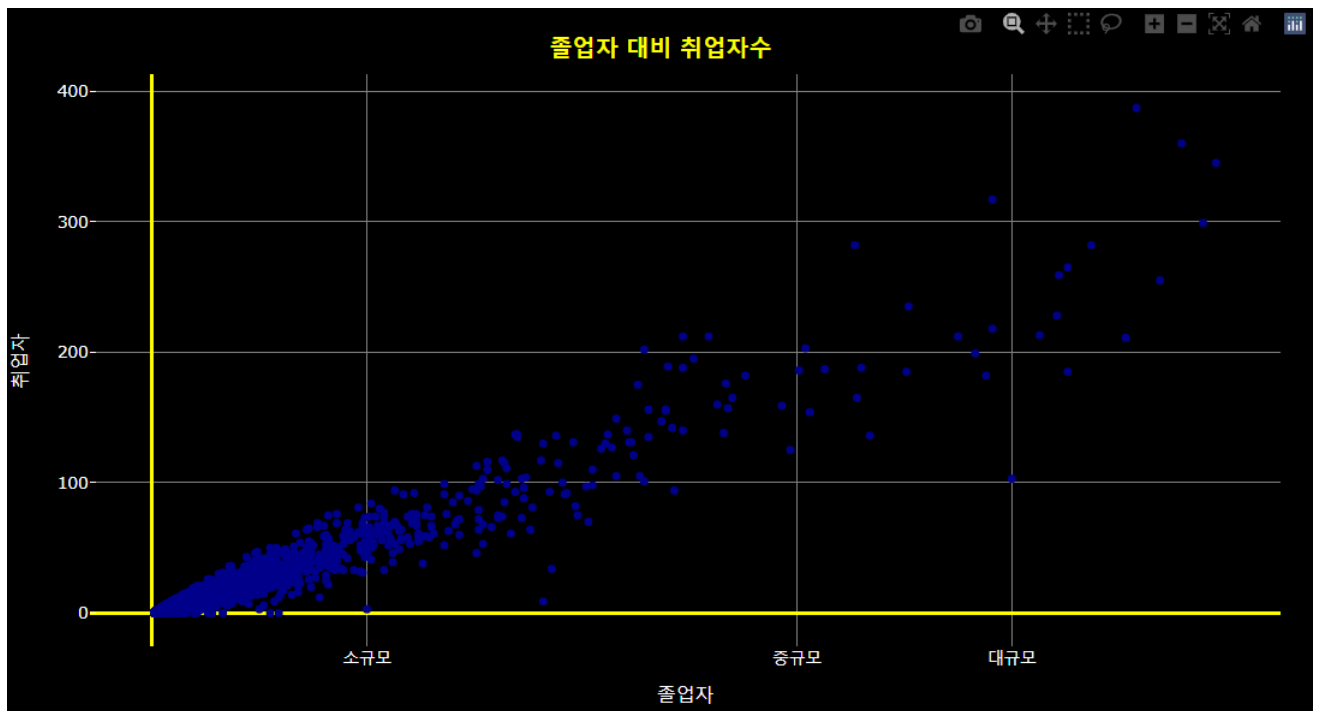


- Python

```
fig_scatter.update_layout(xaxis = dict(title = dict(text = '학과 졸업자수(명)'), color =
'white',
                                zerolinecolor = 'yellow', zerolinewidth = 3,
                                gridcolor = 'grey', gridwidth = 1), ## 정상적 방법
yaxis = dict(title = dict(text = '학과 취업자수(명)'), color = 'white',
                                zerolinecolor = 'yellow', zerolinewidth = 3,
                                gridcolor = 'grey', gridwidth = 1)
)
)
```



```
fig_scatter.update_layout(
    xaxis = dict(title = dict(text = '졸업자'),
        tickmode = 'array',
        ticktext = ('소규모', '중규모', '대규모'),
        tickvals = (100, 300, 400)
    ), ## 정상적 방법
    yaxis = dict(title = '취업자', ticks="inside", tick0 = 100, dtick = 100) ## 약식
)
```



5. 정적 시각화(ggplot, matplotlib)의 plotly 변환

기존의 데이터 분석에서 시각화를 위해 많이 사용하는 패키지가 R 은 `ggplot2` 일 것이고 python 사용자는 `matplotlib` 이나 `seaborn` 을 사용한다. `plotly` 는 이들 정적 시각화를 인터랙티브 데이터 시각화로 변환하는 기능도 제공하기 때문에 기존의 시각화도 재활용할 수 있다.

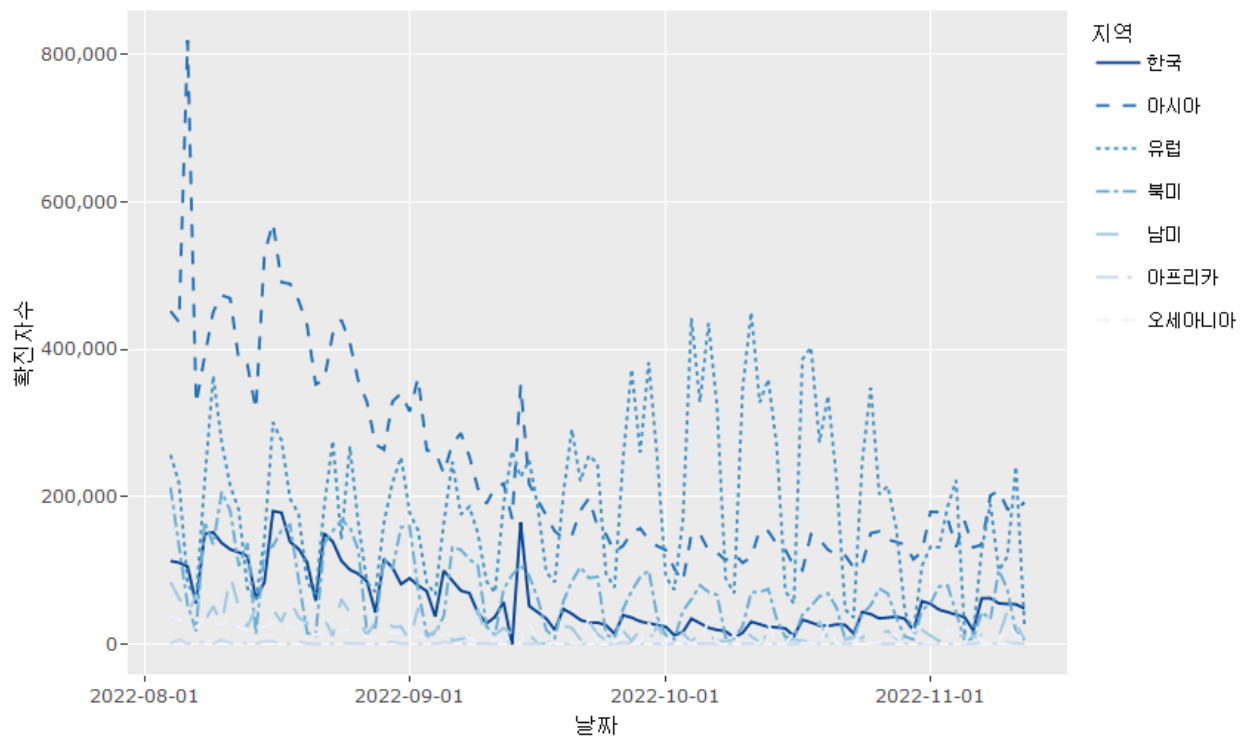
- R

R 의 경우는 `plotly` 패키지에서 제공하는 `ggplotly()` 를 사용하면 간단히 변환된다.

`ggplot2` 로 생성된 객체를 `ggplotly()` 에 매개변수로 전달해 주면 `plotly` 객체로 변환된다.

```
ggplotly <- df_covid19_100 |>
  ggplot(aes(x = date, y = new_cases, color = location )) +
  geom_line(aes(group = location, linetype = location)) +
  scale_x_date(breaks = '1 months') +
  scale_y_continuous(labels = scales::comma) +
  labs(x = '날짜', y = '확진자수', linetype = '지역', color = '지역')

## ggplot 객체를 plotly 객체로 변환
ggplotly(ggplotly)
```

실행결과 2- 10. ggplot2 객체의 전환

- python

python 의 경우는 `plotly.tools` 패키지의 `mpl_to_plotly()`를 사용하여 `matplotlib` 과 `seaborn` 패키지로 생성된 정적 시각화를 `plotly` 의 동적 시각화로 변환할 수 있다.

```
import seaborn as sns
import plotly.tools as tls

sns.lineplot(x="date", y="new_cases",
             hue="location",
             data=df_covid19_100)

mpl_fig = plt.gcf()
plotly_fig = tls.mpl_to_plotly(mpl_fig)
plotly_fig.show()
```



tls.mpl_to_plotly

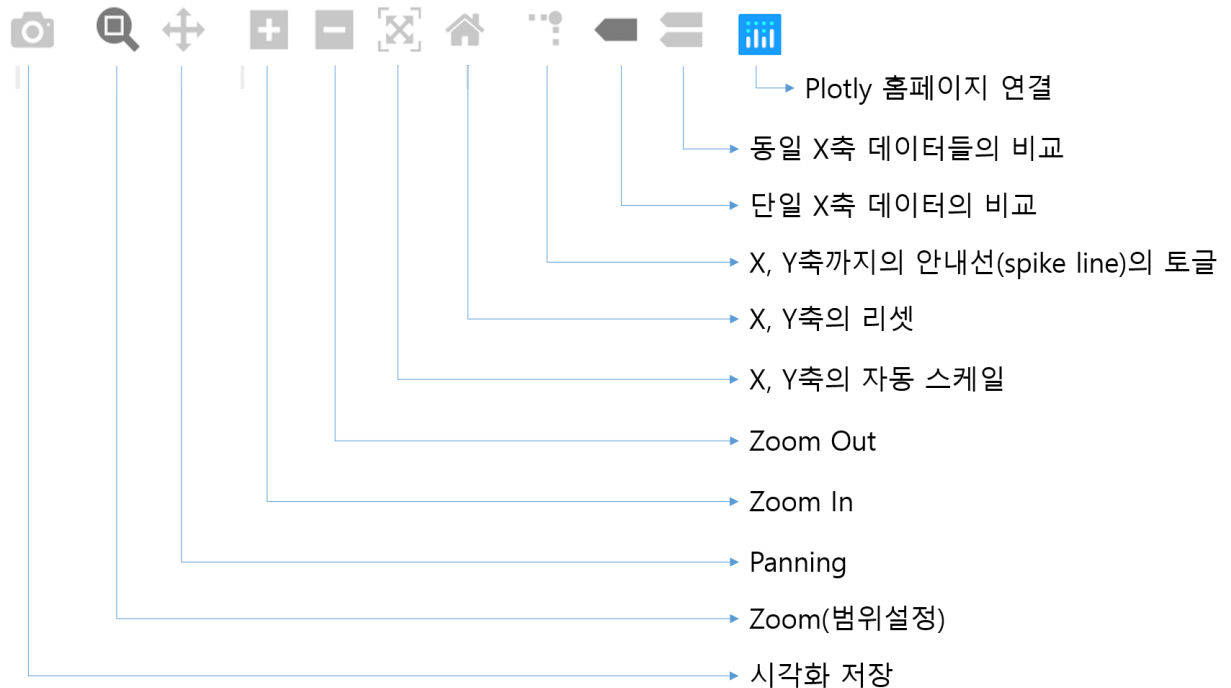
6. plotly 시각화 사용하기

R의 `ggplot2` 나 python의 `matplotlib`, `seaborn`로 만든 정적 시각화는 그래프를 만들 때 시각화한 데이터 외에 시각화 자체에서 추가적인 데이터를 얻기가 어렵다. 따라서 시각화에 추가적인 데이터를 제공하기 위해서는 다시 코딩해서 만들어야 하는 불편함이 따른다. 특히 특정 위치의 데이터 값을 확인하거나 특정 구간 데이터를 zoom하기 위해서도 다시 코딩해야 하는데 사용자의 사용을 예상하여 따라 수없이 많은 시각화를 만들어 놓을 수는 없다. 반면 `plotly`와 같은 동적 시각화에서는 특징적 데이터 값의 확인, zoom, zoom out, 특정 데이터만의 표기 등과 같은 탐색적 데이터 분석에 자유롭게 사용할 수 있는 다양한 기능을 제공한다.

6.1. modebar의 사용

`plotly`가 시각화 사용자와의 상호작용을 위한 주요 기능을 제공하는 메뉴가 'modebar'이다. 'modebar'는 `plotly`가 실행되는 R-Studio 나 웹 브라우저에서 오른쪽

상단에 나타나는 버튼 메뉴를 말한다. 기본적으로 설정되는 'modebar'는 다음의 그림과 같이 8 개의 기능을 버튼을 통해 제공한다.

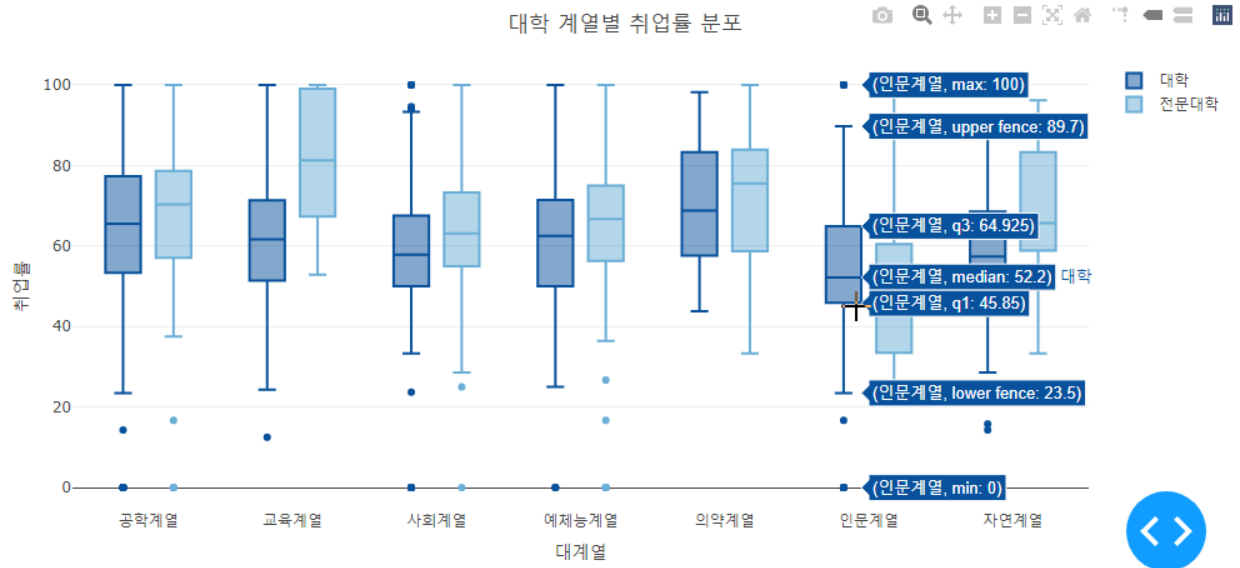


modebar 버튼과 기능

6.2. 마우스를 사용한 데이터 확인

plotly 시각화에서 가장 쉽게 사용하는 기능은 마우스를 사용하여 해당 위치의 데이터 정보를 확인하는 기능이다. **plotly** 객체로 생성된 시각화 위에 표현된 각 trace 들은 자체 데이터를 JSON 의 형태로 포함하고 있기 때문에 마우스 포인터를 trace 위에 위치시키면 해당 trace 의 정보가 표시된다.

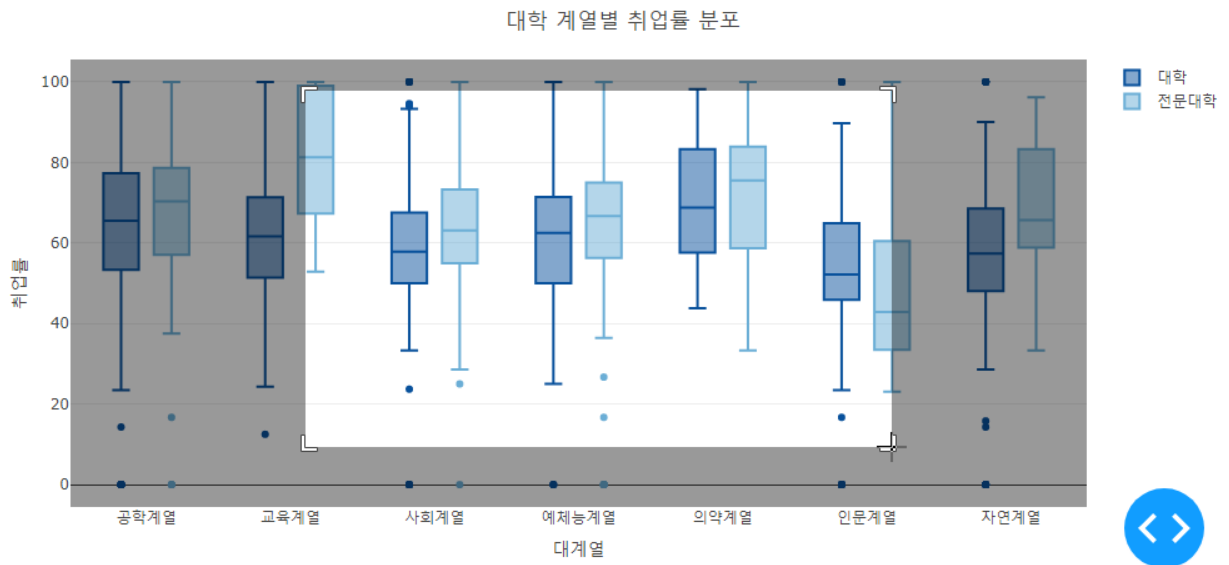
마우스 포인터를 사용하여 표시되는 기본 정보는 X, Y 축에 매핑된 정보와 trace 에 설정되거나 매핑된 **name** 속성이 표시된다. 이 정보를 변경하기 위해서는 두가지 방법이 있는데 앞선 **plotly** 시각화 생성에서 설명한 호버(hover)의 설정인 **hoverinfo** 를 사용하는 방법과 **hovertemplate** 를 사용하는 방법이다. **hoverinfo**(실행결과 2-*)를 사용하는 방법이 보다 간단하고 쉬운 반면 **hovertemplate**(실행결과 2-*)는 표시되는 정보를 세부적으로 설정할 수 있다는 장점이 있다.



박스 trace 의 정보 표시

6.3. 마우스 드래그를 통한 줌

plotly 로 완성된 시각화에서 trace 가 나타나는 plotting area 에서 마우스를 클릭한 상태로 드래그하면 다음 그림과 같이 줌 인 영역을 선택할 수 있다. 이렇게 영역을 선택한 후에 마우스 클릭을 놓으면 해당 부분이 줌인 되어 표시되게 된다. 만약 다시 처음의 상태로 돌아가려면 모드바의 집 아이콘인 'Reset Axes' 버튼을 사용한다.



마우스 드래그를 통한 줌

6.4. 마우스 휠을 사용한 줌

줌인과 줌아웃을 지원하는 많은 프로그램에서는 마우스 휠을 사용하는 프로그램이 많다. `plotly` 도 마우스 휠을 사용하여 줌인과 줌아웃을 사용할 수 있는데 이 기능은 기본적으로 지원하지 않고 `config()`를 사용하여 설정해야 한다. `config()`는 `plotly` 시각화가 사용자와의 상호작용을 지원하는 기능을 조절하고 설정하는 다양한 속성을 조절하는 함수이다. `config()`에서 설정할 수 있는 주요 속성은 다음과 같다.⁶

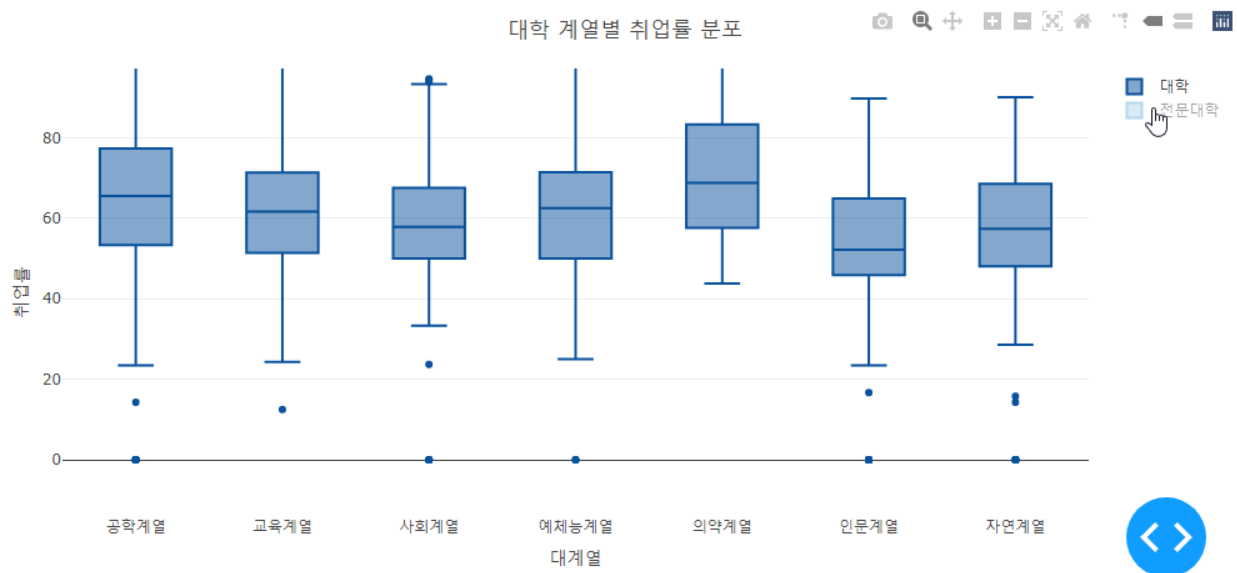
⁶ https://github.com/plotly/plotly.js/blob/master/src/plot_api/plot_config.js

속성	설명	속성값	세부 속성
staticPlot	정적 시각화로 설정	논리값	
editable	시각화를 편집할 수 있는지를 설정	논리값	
edit	editable 이 TRUE 인 경우 편집 가능한 세부 속성	세부 속성들의 리스트	annotationPosition, annotationText, axisTitleText, legendPosition, legendText, titleText
scrollZoom	마우스 휠을 줌으로 사용할지를 설정	'cartesian', 'gl3d', 'geo', 'mapbox', TRUE, FALSE	
doubleClick	마우스 더블 클릭을 어떤 기능으로 사용할지 설정	FALSE, 'reset', 'autosize', 'reset+autosize'	
showTips	팁(호버)를 사용할지 설정	논리값	
displayModeBar	모드바의 표시 방법 설정	'hover', TRUE, FALSE	
modeBarButtonsToRemove	모드바의 버튼 중 없앨 버튼 설정	버튼 ID 문자열	
modeBarButtonsToAdd	모드바에 포함시킬 버튼 설정	버튼 ID 문자열	

속성	설명	속성값	세부 속성
toImageButtonOptions	이미지 저장 버튼의 설정		
displaylogo	모드바의 plotly 로고를 표시할지 설정	논리값	

6.5. 축의 이동

plotly 시각화에서 마우스를 사용한 기능중에 또 하나가 축의 이동이다. **plotly**의 플롯 영역(plotting area)에서 마우스를 클릭하고 드래그를 하면 줌인의 기능을 위한 사각형이 만들어지지만 x, y 축의 위치에서 마우스를 클릭하고 드래그하면 축을 이동시킬 수 있다. 따라서 줌인한 후에 축을 이동시키면 사용자가 자세히 보기를 원하는 지역의 데이터를 확인할 수 있다.

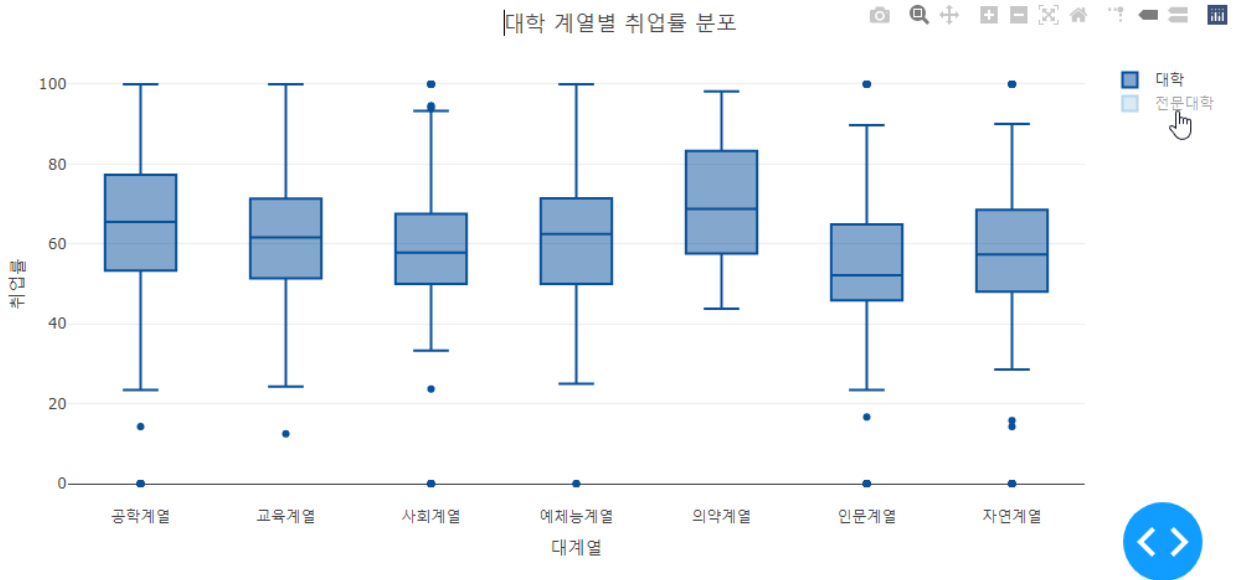


축의 이동

6.6. 범례의 사용

plotly에서 범례는 단순히 trace 들의 이름과 표현방식을 매핑해주는 역할을 넘어 trace 들의 표시를 조절할 수 있는 기능이 있다. 범례의 아이템을 클릭하면 해당 trace 가 표시를 토글하는

역할을 한다. 따라서 여러 데이터 trace 중에 특정한 trace 만을 확인하기 위해서 해당 trace 만 남기고 다른 trace 의 표시를 꺼둘수 있는 기능이 있다. 또 **legendgroup** 으로 그룹화된 범례는 해당 그룹의 아이템의 클릭만으로 그 그룹의 전체 trace 를 켜거나 꺼둘 수 있다.



범례의 토글