

plotly 시각화 만들기

데이터 분석을 다루는 많은 교육 코스나 서적에서 데이터의 시각화는 R의 경우 R base 나 `ggplot2` 패키지를 사용하여는 방법, python의 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 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 수밖에 없으며, 시각화를 사용하는 사용자의 의도에 따른 해석은 매우 제한될 수밖에 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 인쇄물 형태 매체에서 사용이 어렵고 웹을 통해 사용되어야 그 장점을 충실히 사용할 수 있다. R에서는 동적 시각화를 위해 `rbokeh`, `highcharter` 등이 사용되었고 python에서는 `bokeh`, `hvplot` 등을 사용되었다. 하지만 R과 python 모두에서 사용되는 plotly 패키지가 등장함으로써 R과 python의 동적 시각화 패키지 시장에서 매우 빠르게 사용자층을 넓혀가고 있다.

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 2012년 처음 설립되었는데 데이터 전문 분석 도구와 데이터 시각화 전문 도구를 개발하여 보급한다. ‘plotly’는 이 책에서 설명할 R과 python의 plotly 패키지 외에도 R, python, Julia 등의 개발도구에서 사용가능한 데이터 대시보드 플랫폼인 `dash`를 비롯해 `dash` 플랫폼으로 개발된 웹페이지를 배포하기 위한 ‘Dash Enterprise’, plotly를 기반으로 온라인 동적 시각화를

만드는 ‘Chart Studio Cloud’ 등의 서비스를 제공하고 있다. 이 중 가장 유명한 제품이 회사의 이름에서도 나타나듯이 plotly 시각화 패키지이다.

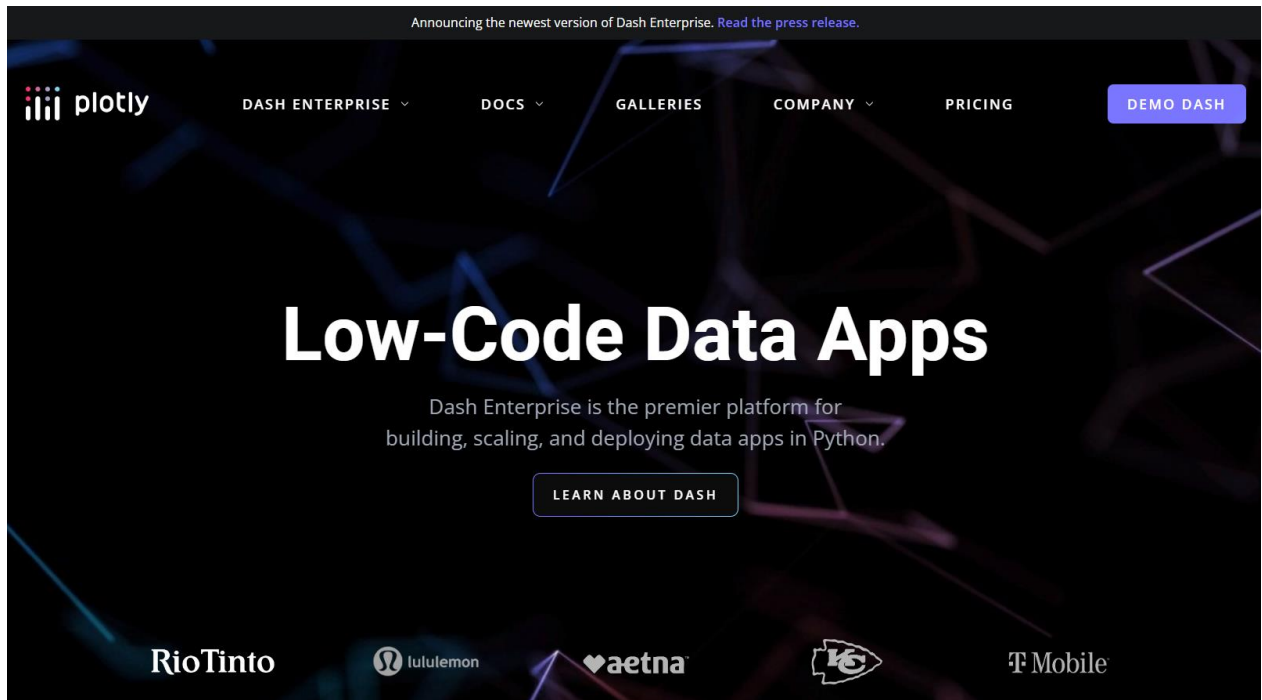


















그림 II-1. plotly 홈페이지 화면

plotly 패키지는 R, python, 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

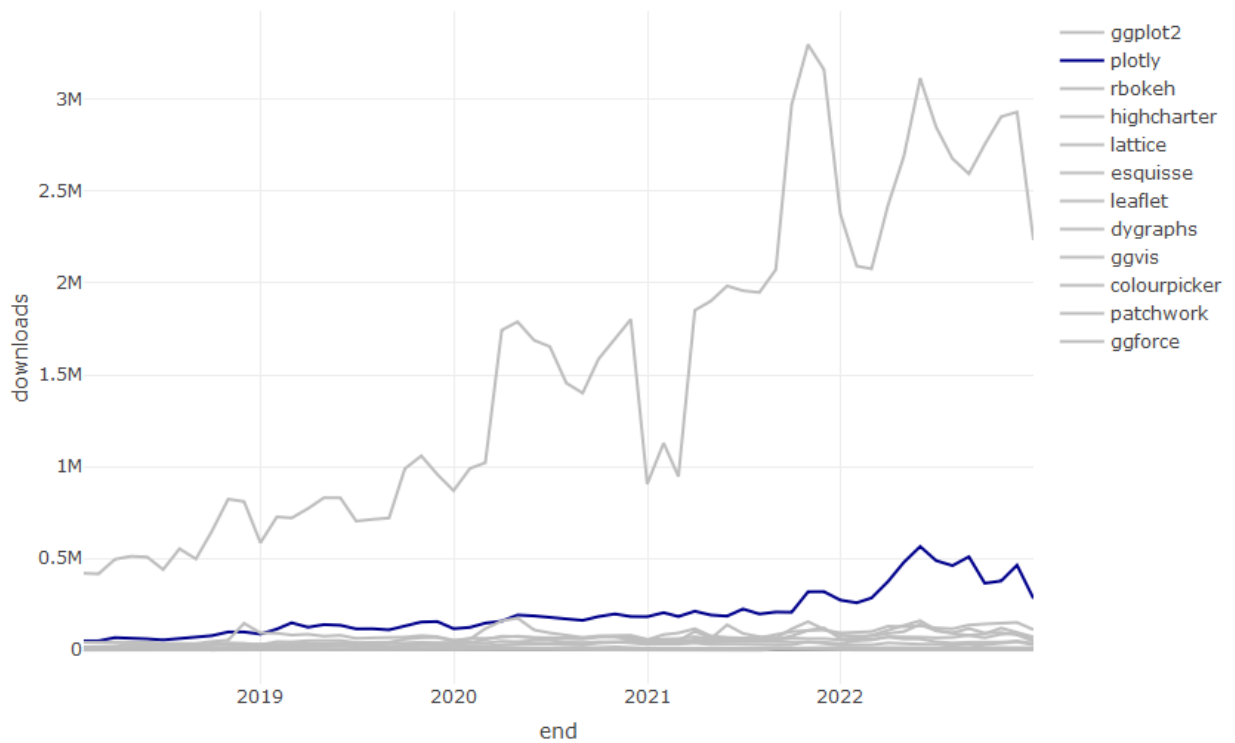
JOIN OUR MAILING LIST

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

[SUBSCRIBE](#)

그림 11-2. plotly 패키지 홈페이지

다음의 R 그래픽 패키지 다운로드 현황에서 보이듯이, 여전히 R에서 가장 많이 사용되고 있는 그래픽 패키지는 **ggplot2**이다. 하지만 plotly는 2021년 하반기부터 다운로드가 늘고 있고, **ggplot2**를 제외한 다른 그래픽 패키지에 비해서는 압도적인 다운로드 수를 보인다.



plotly 의 다운로드 수 증가는 python 에서도 유사한 흐름을 보인다. 다음의 그림에서도 보이듯이 python 에서 많이 사용되는 시각화 패키지 중 가장 다운로드수가 많은 것은 역시나 **matplotlib** 이다. 하지만 **matplotlib** 을 제외하면 2022 년 중순까지만 해도 **seaborn** 패키지의 다운로드 수가 가장 많았으나 이 후 plotly 가 **seaborn** 과 대등하거나 오히려 더 다운로드가 많은 날이 상당히 눈에 띈다. 하지만 python 에서 동적 시각화를 지원하는 **bokeh** 나 **hvplot** 보다는 월등히 많은 다운로드를 보인다. 결국 python 에서도 동적 시각화에서는 plotly 가 가장 많이 사용되는 패키지인 것이다.

최근 6개월간 패키지 다운로드수

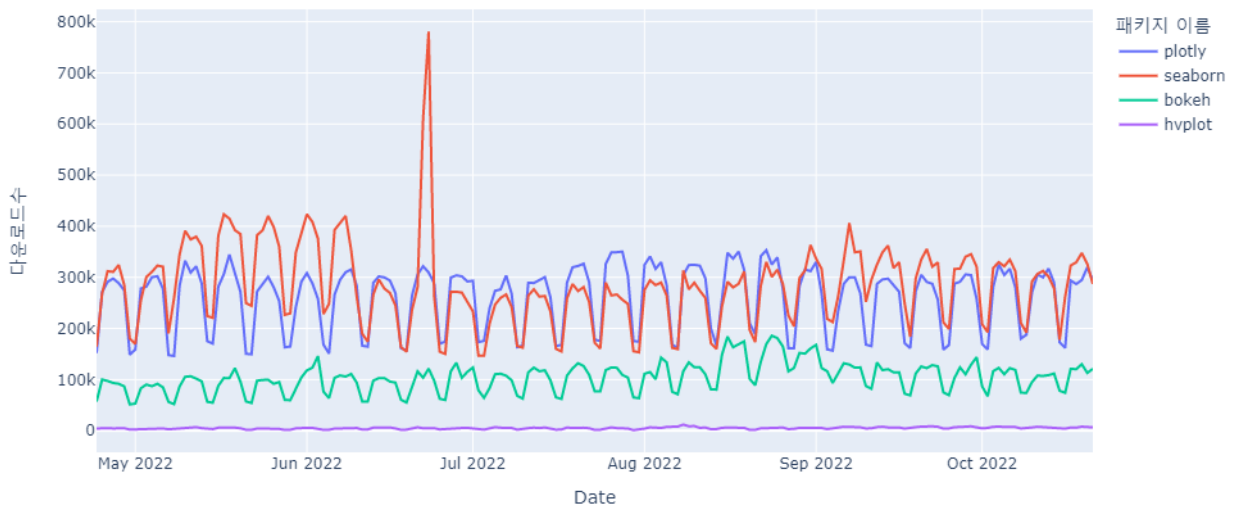


그림 II-3. python plotly 다운로드

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(pacman)) {  
  install.packages('pacman')  
  library(pacman)  
}  
  
p_load("tidyverse", "readxl", "readr", "lubridate")
```

1. covid19 원본 데이터 셋 로딩

covid19 데이터 로딩(파일을 다운로드 받은 경우)

```
df_covid19 <- read_csv(file = "데이터파일저장경로/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),

```

```

전체사망자수 = sum(new_deaths, na.rm = T),
백신접종자완료자수 = max(people_fully_vaccinated, na.rm = T),
인구백명당백신접종완료율 = 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),
       백신접종완료율 = 백신접종자완료자수 / 인구수)

```

여백 설정을 위한 변수 설정

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

- python

필요한 라이브러리 로딩

```

import pandas as pd
from datetime import datetime, timedelta
from pandas.api.types import CategoricalDtype
import plotly.graph_objects as go

```

1. covid19 원본 데이터 셋 로딩

covid19 데이터 로딩(파일을 다운로드 받은 경우)

```
df_covid19 = pd.read_csv("데이터파일저장경로/owid-covid-data_221203.csv")
```

covid19 데이터 로딩(온라인에서 바로 로딩할 경우)

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

2. 전체 데이터셋 중 최근 100 일간의 데이터를 필터링한 df_covid19_100 생성

##df_covid19['date']를 datetime 으로 변환

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

대륙 데이터와 최종 데이터로부터 100 일전 데이터 필터링

```
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)))]
```


대륙명을 한글로 변환

```
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'
```

이산형 변수 설정

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

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

date 로 정렬

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

3. df_covid19_100 을 한국과 각 대륙별로 배치한 넓은 형태의 데이터프레임으로 변환

```
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']
```

4. covid19 데이터를 국가별로 요약한 df_covid19_stat 생성

```
df_covid19_stat = df_covid19.groupby(['iso_code', 'continent', 'location'], dropna=False).agg(
    인구수 = ('population', 'max'),
```

```

전체사망자수 = ('new_deaths', 'sum'),
백신접종자완료자수 = ('people_fully_vaccinated', 'max'),
인구백명당백신접종완료율 = ('people_fully_vaccinated_per_hundred', 'max'),
인구백명당부스터접종자수 = ('total_boosters_per_hundred', 'max')

).reset_index()

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

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

## 여백 설정을 위한 변수 설정
margins_P = {'t' : 50, 'b' : 25, 'l' : 25, 'r' : 25}

## 필요한 라이브러리 로딩
import pandas as pd
from datetime import datetime, timedelta
from pandas.api.types import CategoricalDtype
import plotly.graph_objects as go

## 1. covid19 원본 데이터 셋 로딩
## covid19 데이터 로딩(파일을 다운로드 받은 경우)
df_covid19 = pd.read_csv("D:/R/git/datavisualization/plotly/RnPy/owid-covid-data_221203.csv")

## covid19 데이터 로딩(온라인에서 바로 로딩할 경우)
##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'
```

```
## C:\W\ANACON~1\lib\site-packages\pandas\core\indexing.py:1817: SettingWithCopyWarning:
```

```
## A value is trying to be set on a copy of a slice from a DataFrame.
```

```
## Try using .loc[row_indexer,col_indexer] = value instead
```

```
##
```

```
## See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy
```

```
## self._setitem_single_column(loc, value, pi)
```

```
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)
```

```
## <string>:1: SettingWithCopyWarning:
```

```
## A value is trying to be set on a copy of a slice from a DataFrame.
```

```
## Try using .loc[row_indexer,col_indexer] = value instead
```

```
##
```

```
## See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy
```

```
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_h'
```

```

undred':'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', '확
진자_남미', '확진자_아프리카', '확진자_오세아니아',
                               '백신접종완료자_한국', '백신접종완료자_아시아', '백신접종완료자_유럽', '백신접종
완료자_북미', '백신접종완료자_남미', '백신접종완료자_아프리카', '백신접종완료자_오세아니아']

df_covid19_stat = df_covid19.groupby(['iso_code', 'continent', 'location'], dropna=False).agg(
    인구수 = ('population', 'max'),
    전체사망자수 = ('new_deaths', 'sum'),
    백신접종자완료자수 = ('people_fully_vaccinated', 'max'),
    인구백명당백신접종완료율 = ('people_fully_vaccinated_per_hundred', 'max'),
    인구백명당부스터접종자수 = ('total_boosters_per_hundred', 'max')

).reset_index()

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

df_covid19_stat['백신접종완료율'] = df_covid19_stat['백신접종자완료자수'] / df_covid19_stat['인
구수']

## 여백 설정을 위한 변수 설정
margins_P = {'t' : 50, 'b' : 25, 'l' : 25, 'r' : 25}

```

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

취업률 데이터 셋은 다음과 같이 데이터를 로딩하고 전처리한다.

- R

대학 학과 취업률 데이터 로딩

```
df_취업률 <- read_excel('데이터파일저장경로/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 중 25 % 샘플링

```
df_취업률_500 <- df_취업률 |>  
  filter(졸업자_계 < 500) |>  
  mutate(id = row_number()) |>  
  filter(row_number() %in% seq(from = 1, to = nrow(df_취업률), by = 4))
```

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.

열 이름을 적절히 설정

```
names(df_취업률_500)[10:12] <- c('졸업자수', '취업률', '취업자수')
```

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_500 <- 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_500)[10:12] <- c('졸업자수', '취업률', '취업자수')
```

- python

python 코드

대학 학과 취업률 데이터 로딩

```
df_취업률 = pd.read_excel("데이터파일저장경로/2020 년 학과별 고등교육기관 취업통계.xlsx",
```

```
  ## '학과별' 시트의 데이터를 불러오는데,
```

```
  sheet_name = '학과별',
```

```

    ## 앞의 13 행을 제외하고
    skiprows=(13),
    ## 첫번째 행은 열 이름으로 설정
    header = 0)

## df_취업률에서 첫번째부터 9 번째까지의 열과 '계'로 끝나는 열을 선택하여 다시 df_취업률에 저장
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_취업률에서 졸업자가 500 명 이하인 학과 중 25% 샘플링
df_취업률_500 = df_취업률.loc[(df_취업률['졸업자_계'] < 500)]

df_취업률_500 = df_취업률_500.iloc[range(0, len(df_취업률_500.index) , 4)]

df_취업률_500 = df_취업률_500.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 구성하는데 이들 속성들은 부모 속성과 자식 속성으로 구성되는 트리 형태의 JSON 구조로 구성된다. 이 구성은 R 과 python 이 동일하지만 그 구성 방법은 다소 다르다.

- R

R에서는 plotly 구조를 구성하기 위해서 먼저 `plot_ly()`를 사용하여 plotly 객체를 초기화하여야 한다. 이후 세 가지 방법을 사용할 수 있다.

첫 번째 방법은 `add_*()`와 `layout()`을 사용하는 방법이다. 이 두가지 함수를 사용하기 위해서는 먼저 `plot_ly()`로 이 초기화된 plotly 객체에 `add_trace()`를 사용하여 'data'의 하위 속성들을 구성하고 `layout()`을 사용하여 'layout'의 하위 속성들을 구성한다.

두 번째 방법은 이 함수들을 사용하지 않고 `plot_ly()`에 직접 'data'와 'layout' 속성들을 설정할 수 있다. 이 방법은 `plotly` 구조를 하나 하나 설정해야하기 때문에 구현하는데 어려움이 있다.

세 번째 방법은 `add_trace()` 대신 트레이스 종류에 따라 제공되는 `add_*()`(`add_markers()`, `add_lines()`, `add_bars()` 등)을 사용하여 'data' 속성들을 구성하고 `layout()`을 사용하여 'layout' 속성들을 설정한다.

- python

python 에서 `plotly`를 설명하기 위해서는 먼저 plotly 에서 제공하는 라이브러리의 종류를 이해하여야 한다. plotly 에서는 python 을 위한 라이브러리로 `plotly.graph_objects` 와 `plotly.express` 의 두 가지 라이브러리 모듈을 사용한다. 기본적으로는 `plotly.graph_objects` 모듈이 plotly 에서 제공하는 모든 기능을 사용하는 정규 방법이다. 하지만 `plotly.graph_objects` 에서 제공하는 수많은 속성중에 자주 사용되는 속성과 이들 속성을 설정하는 다소 쉬운 인터페이스로 설계한 함수들로 구성된 `plotly.express` 모듈도 있다.

`plotly.express` 에서 제공하는 함수들은 `plotly.graphic_object` 보다 직관적이고 사용하기 쉽다. plotly 는 'data' 속성과 'layout' 속성을 루트 노드로하는 속성들의 트리 구조로 구현하기 때문에 이들 구조를 파악하는 것이 매우 중요하지만 그 양도 많고 구조도 복잡하다.

`plotly.express` 에서 제공하는 함수들은 이 속성들을 트리 형태로 구성하지 않고 모두 매개 변수의 형태로 설정하는 방법을 사용하기 때문에 일반적 함수의 사용과 비슷해서 사용하기 쉽다는 장점이 있다. 또 `plotly.graph_objects` 의 함수들보다 코드의 길이가 매우 짧아진다는 장점도 있다.

하지만 `plotly.express` 는 결정적인 몇가지 단점이 있다. 첫 번째 단점은 'mesh'나 'isosurface'와 같은 3 차원 시각화는 아직 `plotly.express` 는 지원하지 않는다. 두 번째는 여러개의 trace 를 가지는 서브플롯, 다중 축의 사용, 여러개의 trace 를 가지는 패싯(facet)과 같은 시각화는 `plotly.express` 로 생성하는데 다소 어려움이 있다. 따라서 `plotly.express` 로는 다소 복잡한 plotly 객체를 생성하는데에는 한계가 있어 `plotly.graphic_object` 의 함수들을 사용하여 보완해주어야 한다. 또 `plotly.graphic_object` 에서 제공하는 함수와 `plotly.express` 에서 제공하는 함수의 속성도

다소 차이가 있기 때문에 `plotly.express` 는 사용이 간편하긴 하지만 사용할 때는 사용법을 잘 확인하고 사용해야 한다.⁴

`plotly.graph_objects` 모듈은 `plotly` 객체를 만들기 위해서는 먼저 `plotly` 객체를 초기화해야 한다. 이를 위해서는 `plotly.graph_objects` 의 `Figure()`를 사용하여 초기화된 `plotly` 객체를 생성한다.

이 후 초기화된 `plotly` 객체에 'data', 'layout' 속성을 할당하기 위해

`plotly.graph_objects.add_trace()`와 `plotly.graph_objects.update_layout()`을 사용한다.

3.1. plotly 속성의 설정

`plotly` 는 시각화 생성을 위한 다양한 속성 데이터를 트리 형태로 구조화한 JSON 객체라고 소개하였다. 따라서 `plotly` 를 구성하는데 사용하는 함수들은 대부분 이 속성 데이터를 트리 구조의 JSON 객체로 구조화 하는데 사용한다. 이렇게 속성 데이터를 트리 형태로 구조화하는 방법은 R 과 python 이 유사하지만 약간의 차이가 있다.

- R

R 에서 `plotly` 의 속성 트리를 만들기 위해서는 리스트 데이터 타입을 사용한다. 리스트 데이터 타입은 벡터와는 다소 다른데 벡터는 벡터를 구성하는 원소들의 데이터 타입이 모두 같아야하지만 리스트는 리스트를 구성하는 원소들의 데이터 타입이 서로 다를 수 있다. 특히 `plotly` 에서 사용하는 리스트는 원소의 이름이 붙은 네임드 리스트(named list)를 사용한다.

R 에서 네임드 리스트를 만들기 위해서는 `list()`를 사용하는데 속성명과 속성값은 '=' 기호를 사용하여 서로 할당해 준다. 속성 값은 또 다시 벡터나 리스트의 설정이 가능한데, 이 기능을 사용해 `plotly` 의 전체 트리 구조를 만드는데 사용한다. `plotly` 에서 특정 속성의 하위 속성들이 존재하면 해당 하위 속성들을 다시 `list()`를 사용하여 네임드 리스트로 구성한다.

예를 들어 `layout = list(title = list(text = '타이틀 제목'))`이라는 코드는 'layout' 에 리스트를 할당하는데 이 리스트는 속성명이 'title'이고 속성값은 다시 리스트로 구성된 속성값을 가진다. 결국 하위 속성인 'title' 속성, 이 'title' 속성의 하위 속성인 'text' 속성값을 '타이틀 제목'으로 설정한다.

⁴ 본 책에서는 `plotly.graph_objects` 위주로 설명한다. `plotly.express` 의 사용법은 <https://plotly.com/python-api-reference/> 을 참조하라.

이렇게 구성된 plotly 객체의 속성과 속성값은 '.'을 사용해 속성값의 구조적 경로(Path)를 통해 접근할 수 있다. 앞서 예를 들었던 `layout = list(title = list(text = '타이틀 제목'))`에 바로 접근하려면 `layout.title.text`를 사용하여 접근할 수 있다. 이 방법을 사용하여 plotly 객체의 속성값을 추출하거나 해당 속성값을 바로 바꾸어 줄 수 있다.

- python

python에서 `plotly`의 속성 트리를 만드는데는 R보다 다소 복잡하다. python에서 속성 트리를 만드는데 사용하는 기본적 데이터 타입은 딕셔너리이다. 하지만 하나의 속성에 여러개의 딕셔너리가 구성되어야 하는 경우는 다시 python의 리스트로 구성한다. 결국 속성값들의 집합을 만드는 것은 딕셔너리를 사용하지만 딕셔너리의 집합은 리스트를 사용한다는 것이다.

python에서 딕셔너리를 구성하는 방법도 두 가지이고, plotly에서는 이 두 가지 방법을 모두 지원한다.

첫 번째 방법은 `{}` 기호를 사용하는 방법이다. 이 방법은 속성명을 지정할 때 반드시 인용부호(작은 따옴표 혹은 큰 따옴표)를 사용해야하고 할당 기호로 '='를 사용한다.

두 번째 방법은 `dict()`를 사용한다. `dict()`는 `{}`와는 달리 속성명을 지정할 때 인용부호를 사용하지 않아도 되고 할당 기호로 '='를 사용한다.

이렇게 속성명과 속성값으로 구성된 딕셔너리 여러개를 하나의 속성에 할당할 때는 리스트 데이터 타입을 사용한다. 리스트 데이터 타입은 대괄호(`[]`)로 묶어준다.

3.2. plotly 설치와 로딩

plotly를 사용하기 위해서는 먼저 R과 python에 `plotly` 패키지와 라이브러리를 설치하여야 한다. plotly 패키지는 자체적으로 실행가능한 HTML 파일과 시각화를 작성하는 데 필요한 많은 것이 포함되어 있다.

- R

R에서 plotly는 이미 공식 CRAN에 등록되어 있기 때문에 `install.packages()`를 사용하여 설치할 수 있다. 설치한 이후 R에서 'plotly'를 사용하기 위해서는 먼저 `library()`나 `require()`로 plotly 패키지를 로딩하고 사용할 수 있다. 다음과 같이 패키지를 설치하고 로딩할 수 있다.

```
if(!require(plotly)) {    ## plotly 로딩이 안되면
  install.packages('plotly') ## plotly 패키지 설치
}
```

```
library(plotly)      ## plotly 패키지 로딩
}
```

- python

python 에서 plotly 를 설치하기 위해서는 pip 를 다음과 같이 실행하여 설치한다.

```
$ pip install plotly  ## pip 를 사용해 plotly 설치
```

또는 콘다에서 다음과 같이 실행하여 설치한다.

```
$ conda install -c plotly plotly  ## conda 를 사용해 plotly 설치
```

3.3. 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.

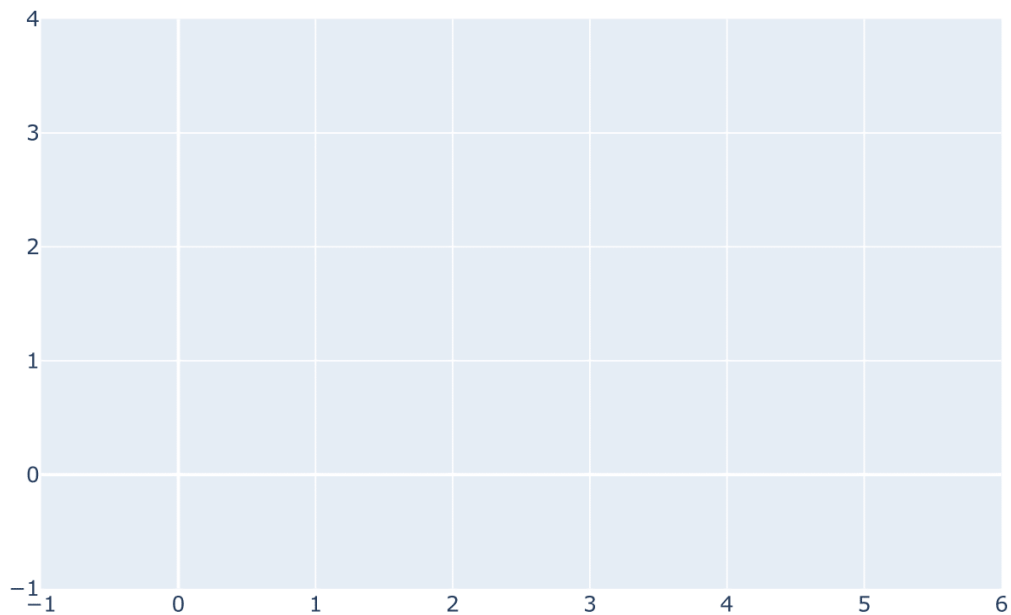
```
## R 에서 plotly 객체 초기화
df_covid19_100 |>
  plot_ly()
```

- python

python 에서 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. 하지만 plotly.express 모듈 함수에는 첫 번째 매개변수로 해당 함수에서 사용하는 데이터프레임을 첫번째 매개변수로 사용하여 해당 plotly 객체에 사용할 데이터를 바인딩해줄수 있다.

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

go.Figure()
```



실행결과 II-1. python 의 Figure()를 사용한 초기화

3.4. 'data' 속성

plotly 구조의 첫 번째 레벨 속성인 'data' 속성은 시각화를 통해 표현해야 할 데이터와 그 표현 방식을 설정하는 속성이다. 'data' 속성은 데이터를 표현하는 트레이스를 구성하는 세부 속성들을 말하고, 트레이스는 plotly 로 시각화할 수 있는 그래픽적 데이터 표현 방법이다. plotly 에서는 'scatter', 'pie', 'bar' 등의 40 개가 넘는 트레이스를 제공한다.

40 여가지 트레이스 중에 사용하고자 하는 트레이스를 설정하기 위해서는 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 R 의 `add_lines()` 나 python 의 `plotly.express.line()` 등) 'type' 속성을 사용할 필요는 없다.

- R

R 에서 'data' 속성은 `plot_ly()` 나 `add_trace()` 를 사용하여 설정한다. 'data' 속성의 첫 번째 레벨 속성들은 바로 '=' 을 사용하여 속성값을 할당하지만 해당 속성이 하위 속성으로 구성되는 컨테이너 속성의 경우 R 의 기본 데이터 타입인 'list' 를 사용하여 묶어서 설정한다.

다음의 코드는 `plot_ly()` 를 사용하여 'data' 속성의 첫 레벨 속성인 'type', 'x', 'y' 는 바로 속성값을 설정하지만 컨테이너 속성인 'marker' 와 'line' 은 `list()` 를 사용하여 리스트로 구성된 속성값이 설정되었다.

```
## 긴 형태의 100 일 코로나 19 데이터에서
df_covid19_100 |>
  ## 한국 데이터만을 필터링
  filter(iso_code == 'KOR') |>
  ## scatter 트레이스의 markers 와 lines 모드의 plotly 시각화 생성
  plot_ly(type = 'scatter', mode = 'markers+lines',
    ## X, Y 축에 변수 매핑
    x = ~date, y = ~new_cases,
    ## 마커 색상 설정
    marker = list(color = '#264E86'),
    ## 라인 색상과 대시 설정
    line = list(color = '#5E88FC', dash = 'dash')
  )
```

앞 코드의 'data' 속성 트리 구조는 다음의 그림과 같다.

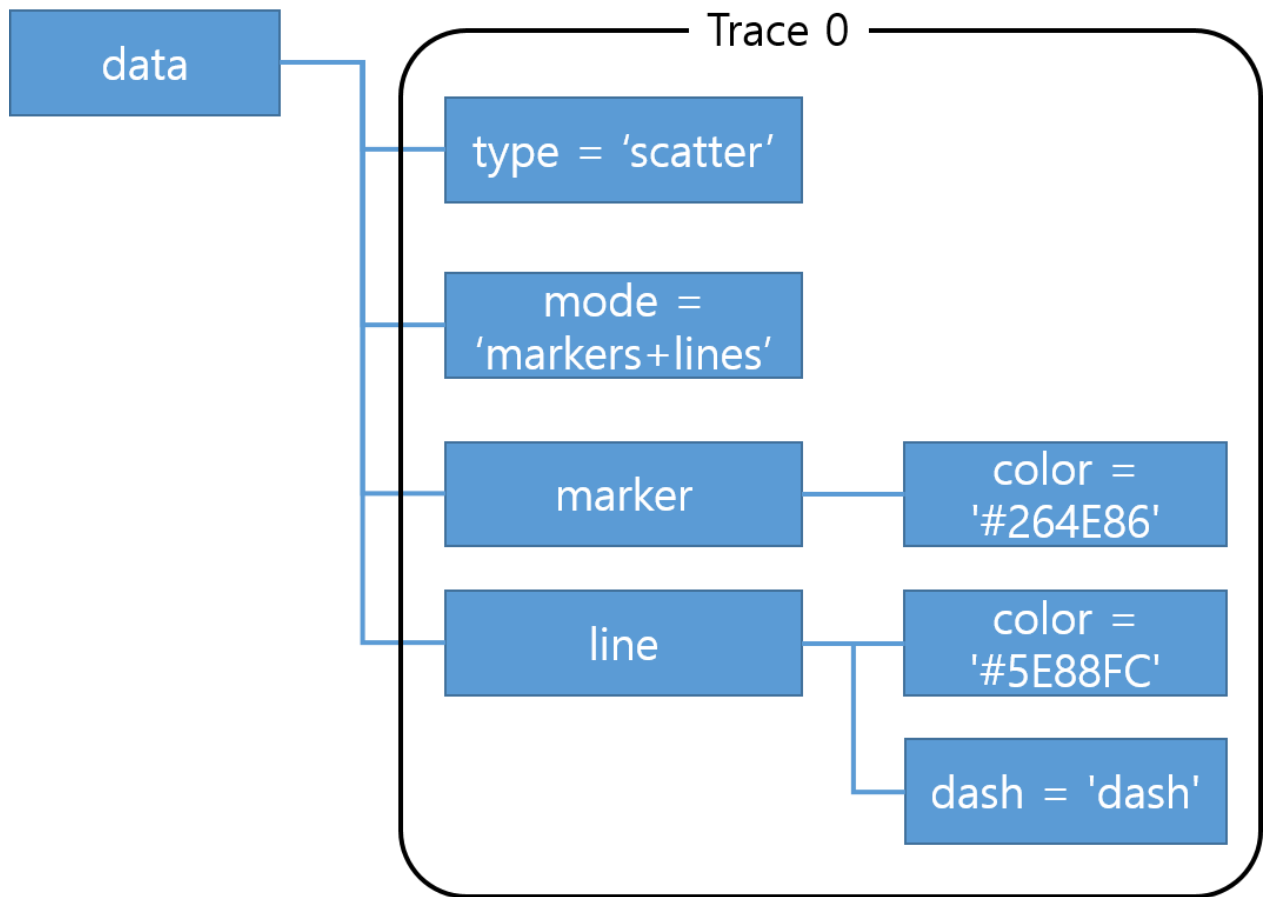


그림 II-4. R 샘플 코드의 data 속성 트리 구조

- python

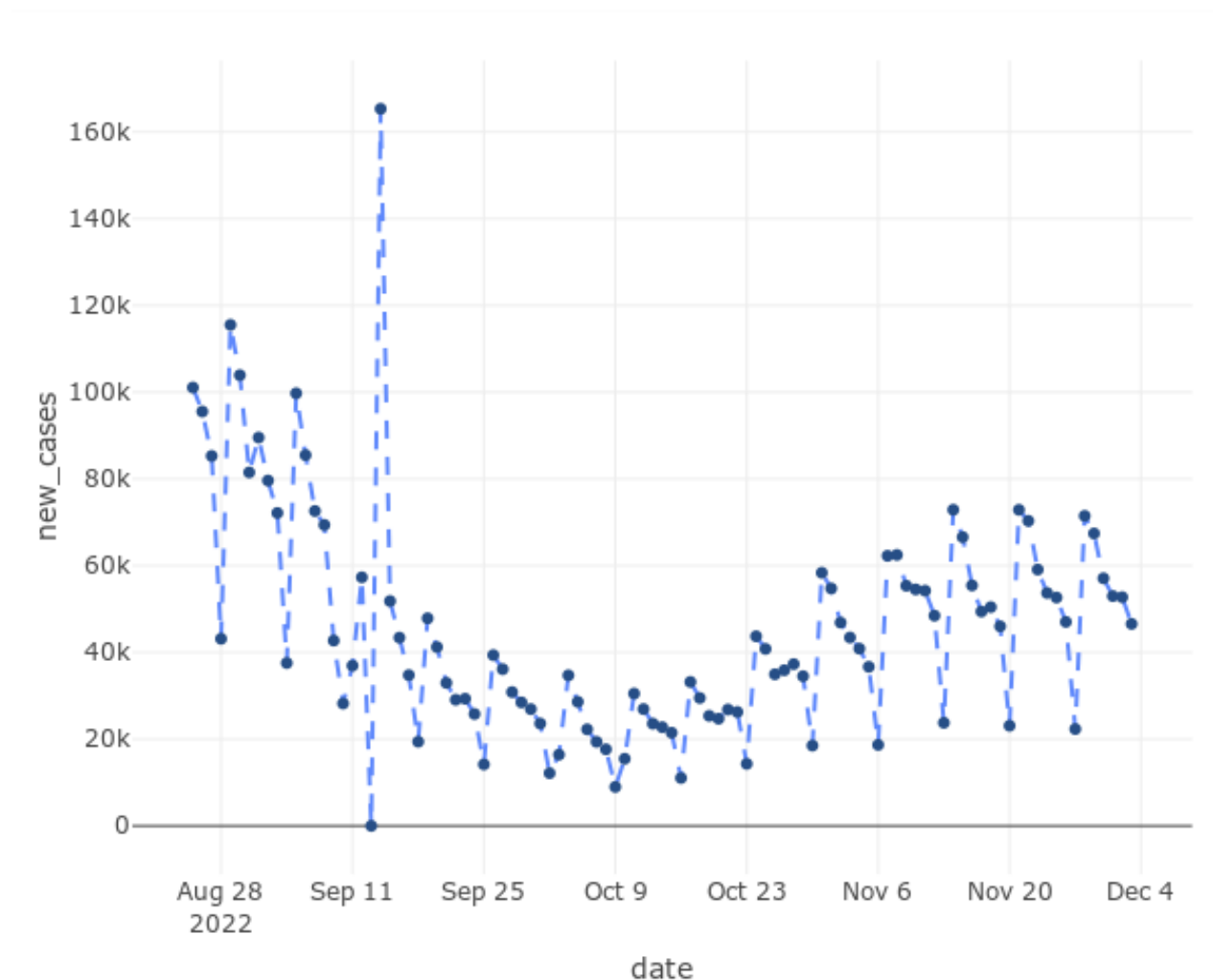
python 에서도 'data' 속성 설정을 위해 plotly 초기화 함수인 `plotly.graph_objects.Figure()`와 `add_trace()`를 사용하는 두 가지 방법이 있다.

`plotly.graph_objects.Figure()`로 'data' 속성을 설정하기 위해서는 먼저 `plotly.graph_objects` 라이브러리를 'go'로 임포트하고 `go.Figure()`의 'data' 매개 변수로 'data' 속성들로 구성된 딕셔너리를 할당 한다.

다음의 코드는 `Figure()`를 사용하여 R 로 만든 시각화와 동일한 시각화를 만드는 python 코드이다. 하지만 앞의 R 코드에서는 scatter 트레이스의 "markers+lines"로 'mode'를 설정해서 하나의 트레이스로 설정했지만 python 의 코드에서는 markers 스캐터 트레이스와 lines 스캐터 트레이스의 두 개의 트레이스로 구성하였다.

```
## plotly 초기화 함수로 data 속성 값 설정
go.Figure(
```

```
## data 키워드로 data 속성의 설정임을 명시
data = [{
    ## scatter 트레이스의 markers 속성 설정
    'type' : 'scatter', 'mode' : 'markers',
    ## X, Y 축에 변수 매핑
    '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'],
    ## 마커 색상 설정
    'marker' : {'color' : '#264E86'}
},
    ## scatter 트레이스의 line 속성 설정
    {'type' : 'scatter', 'mode' : 'lines',
    ## X, Y 축에 변수 매핑
    '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'],
    ## 라인 색상과 대시 설정
    'line' : {'color' : '#5E88FC', 'dash' : 'dash'}
}
]
```



실행결과 II-2. R의 `Figure()`를 사용한 `data` 속성 설정

앞 코드의 `plotly` 'data' 속성의 속성 트리 구조는 다음의 그림과 같다.

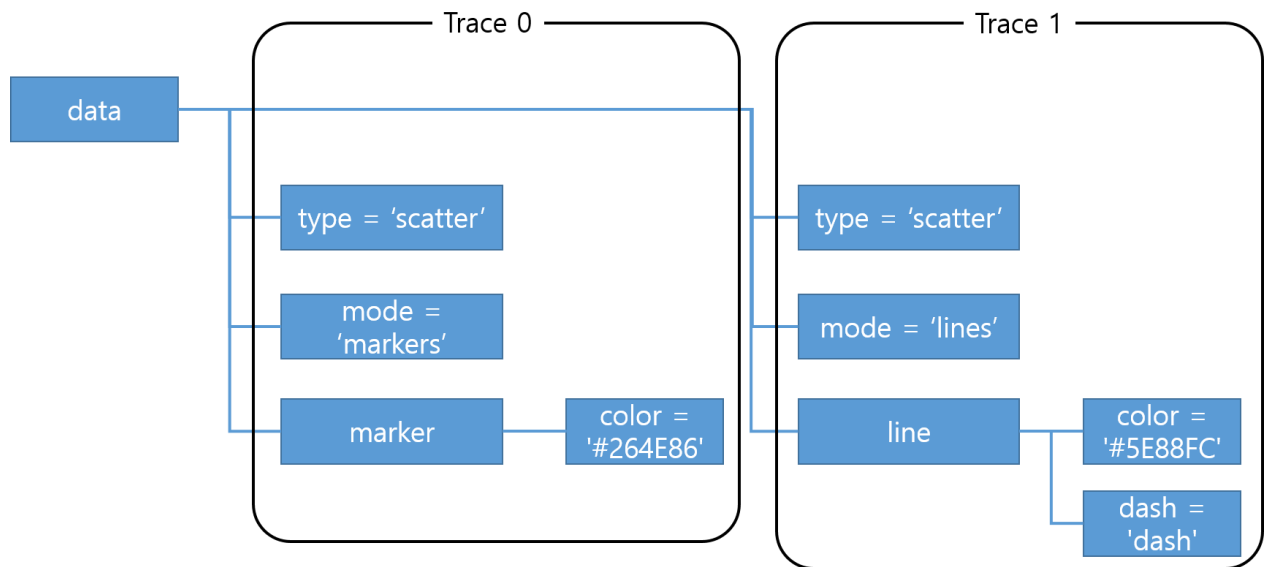


그림 II-5. python 샘플 코드의 data 속성 트리 구조

3.5. 'layout' 속성

'layout' 속성은 데이터를 표현하는 트레이스와 관련되지 않는 시각화의 나머지 속성들을 정의하는 설정들의 최상위 속성이다. 'layout' 에서 설정할 수 있는 속성은 시각화의 차원 축(2 차원, 3 차원), 여백, 제목, 축, 범례, 컬러바(색 범례), 주석 등이다. 'layout' 속성들도 하위 속성들로 구성되어 있고 이들 하위 속성들은 리프 노드로 구성된 속성도 있지만 다시 하위 속성으로 구성된 컨테이너 속성도 있다.

- R

R 에서 'layout' 속성을 설정하기 위해서는 `layout()`을 사용한다. 다만 `layout()`을 사용하기 위해서는 plotly 객체가 있어야 한다. 다음의 코드는 앞서 그린 시각화에 'layout' 속성인 '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 각각의 하위 속성이 있기 때문에 다시 list 로 묶어서 설정하였다.

```

df_covid19_100 |>
  filter(iso_code == 'KOR') |>
  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", ## 전체 제목 설정
    xaxis = list(title = "날짜", showgrid = F), ## X 축 layout 속성 설정
    yaxis = list(title = "확진자수"), ## y 축 layout 속성 설정
    margin = margins_R) ## 여백 설정

```

- python

python 에서 ‘layout’ 속성의 설정은 ‘data’ 속성과 같이 두 가지 방법이 제공되는데 `Figure()`를 사용하는 방법과 `update_layout()`을 사용하는 방법이 있다. `Figure()`에서의 설정은 ‘data’ 속성의 설정과 같이 ‘layout’ 속성들로 구성된 딕셔너리를 “layout” 매개변수에 할당한다.

다음의 코드는 앞서 그린 시각화에 ‘layout’ 속성을 설정하는 python 코드이다. ‘title’ 속성으로 시각화 제목을 설정하였고, 컨테이너 속성인 ‘xaxis’, ‘yaxis’ 속성에 그 하위 속성들로 구성된 딕셔너리를 할당하였다. 또 ‘margin’ 속성으로 미리 설정된 딕셔너리로 설정하였다.

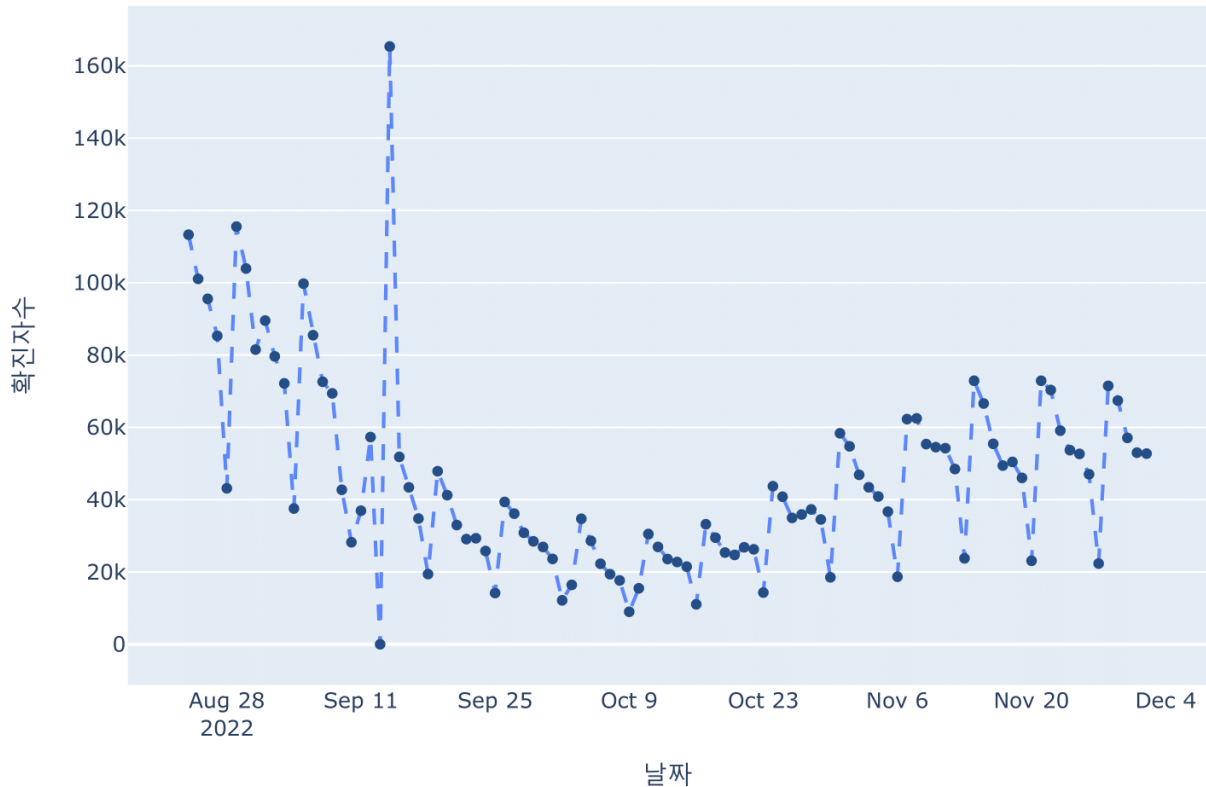
```

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", ## 전체 제목 설정
        'xaxis' : {'title' : "날짜", 'showgrid' : False}, ## X 축 layout 속성 설정
        'yaxis' : {'title' : "확진자수"}, ## y 축 layout 속성 설정
        'margin' : margins_P ## 여백 설정
    })

fig.show()

```

코로나 19 발생 현황



실행결과 II-3. python 의 layout 속성설정

3.6. 'frame' 속성

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

3.7. 'plotly' 구조 확인

앞 서 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

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

```
df_covid19_100 |>
  filter(iso_code == 'KOR') |>
  plot_ly(type = 'scatter', x = ~date, y = ~new_cases,
          mode = 'markers+lines',
          marker = list(color = '#264E86'),
          line = list(color = '#5E88FC', dash = 'dash')) |>
  plotly_json() ## plotly 구조 출력
```

```
## {
##   "visdat": {
##     "40d033e444a2": ["function () ", "plotlyVisDat"]
##   },
##   "cur_data": "40d033e444a2",
##   "attrs": {
##     "40d033e444a2": {
##       "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": ["2022-08-25", "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"],  
##      "y": [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, 23765, 72883, 66587, 55437, 49418, 50435, 46011, 23091, 72873, 70324, 59089, 53698, 52648, 47028, 22327, 71476, 67415, 57079, 52987, 52726, 46564],  
##      "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)    ## plotly 구조 출력
```

[illegible]

##	datetime.datetime(2022, 9, 21, 0, 0),
##	datetime.datetime(2022, 9, 22, 0, 0),
##	datetime.datetime(2022, 9, 23, 0, 0),
##	datetime.datetime(2022, 9, 24, 0, 0),
##	datetime.datetime(2022, 9, 25, 0, 0),
##	datetime.datetime(2022, 9, 26, 0, 0),
##	datetime.datetime(2022, 9, 27, 0, 0),
##	datetime.datetime(2022, 9, 28, 0, 0),
##	datetime.datetime(2022, 9, 29, 0, 0),
##	datetime.datetime(2022, 9, 30, 0, 0),
##	datetime.datetime(2022, 10, 1, 0, 0),
##	datetime.datetime(2022, 10, 2, 0, 0),
##	datetime.datetime(2022, 10, 3, 0, 0),
##	datetime.datetime(2022, 10, 4, 0, 0),
##	datetime.datetime(2022, 10, 5, 0, 0),
##	datetime.datetime(2022, 10, 6, 0, 0),
##	datetime.datetime(2022, 10, 7, 0, 0),
##	datetime.datetime(2022, 10, 8, 0, 0),
##	datetime.datetime(2022, 10, 9, 0, 0),
##	datetime.datetime(2022, 10, 10, 0, 0),
##	datetime.datetime(2022, 10, 11, 0, 0),
##	datetime.datetime(2022, 10, 12, 0, 0),
##	datetime.datetime(2022, 10, 13, 0, 0),
##	datetime.datetime(2022, 10, 14, 0, 0),
##	datetime.datetime(2022, 10, 15, 0, 0),
##	datetime.datetime(2022, 10, 16, 0, 0),
##	datetime.datetime(2022, 10, 17, 0, 0),
##	datetime.datetime(2022, 10, 18, 0, 0),
##	datetime.datetime(2022, 10, 19, 0, 0),
##	datetime.datetime(2022, 10, 20, 0, 0),
##	datetime.datetime(2022, 10, 21, 0, 0),
##	datetime.datetime(2022, 10, 22, 0, 0),
##	datetime.datetime(2022, 10, 23, 0, 0),
##	datetime.datetime(2022, 10, 24, 0, 0),
##	datetime.datetime(2022, 10, 25, 0, 0),

##	datetime.datetime(2022, 10, 26, 0, 0),
##	datetime.datetime(2022, 10, 27, 0, 0),
##	datetime.datetime(2022, 10, 28, 0, 0),
##	datetime.datetime(2022, 10, 29, 0, 0),
##	datetime.datetime(2022, 10, 30, 0, 0),
##	datetime.datetime(2022, 10, 31, 0, 0),
##	datetime.datetime(2022, 11, 1, 0, 0),
##	datetime.datetime(2022, 11, 2, 0, 0),
##	datetime.datetime(2022, 11, 3, 0, 0),
##	datetime.datetime(2022, 11, 4, 0, 0),
##	datetime.datetime(2022, 11, 5, 0, 0),
##	datetime.datetime(2022, 11, 6, 0, 0),
##	datetime.datetime(2022, 11, 7, 0, 0),
##	datetime.datetime(2022, 11, 8, 0, 0),
##	datetime.datetime(2022, 11, 9, 0, 0),
##	datetime.datetime(2022, 11, 10, 0, 0),
##	datetime.datetime(2022, 11, 11, 0, 0),
##	datetime.datetime(2022, 11, 12, 0, 0),
##	datetime.datetime(2022, 11, 13, 0, 0),
##	datetime.datetime(2022, 11, 14, 0, 0),
##	datetime.datetime(2022, 11, 15, 0, 0),
##	datetime.datetime(2022, 11, 16, 0, 0),
##	datetime.datetime(2022, 11, 17, 0, 0),
##	datetime.datetime(2022, 11, 18, 0, 0),
##	datetime.datetime(2022, 11, 19, 0, 0),
##	datetime.datetime(2022, 11, 20, 0, 0),
##	datetime.datetime(2022, 11, 21, 0, 0),
##	datetime.datetime(2022, 11, 22, 0, 0),
##	datetime.datetime(2022, 11, 23, 0, 0),
##	datetime.datetime(2022, 11, 24, 0, 0),
##	datetime.datetime(2022, 11, 25, 0, 0),
##	datetime.datetime(2022, 11, 26, 0, 0),
##	datetime.datetime(2022, 11, 27, 0, 0),
##	datetime.datetime(2022, 11, 28, 0, 0),
##	datetime.datetime(2022, 11, 29, 0, 0),

```
##          datetime.datetime(2022, 11, 30, 0, 0),
##          datetime.datetime(2022, 12, 1, 0, 0),
##          datetime.datetime(2022, 12, 2, 0, 0),
##          datetime.datetime(2022, 12, 3, 0, 0)], dtype=object),
##          'y': array([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.,
##          23765., 72883., 66587., 55437., 49418., 50435., 46011., 23091.,
##          72873., 70324., 59089., 53698., 52648., 47028., 22327., 71476.,
##          67415., 57079., 52987., 52726., 46564.])),
##    'layout': {'template': '...'}
## })
```

4. plotly 생성

앞에서 plotly 객체의 구조에 대해 살펴보았다. plotly 객체는 plotly.js 가 시각화해주는 JSON 형태의 데이터 구조체일 뿐이다. 따라서 plotly 를 사용하여 시각화 한다는 것은 ‘data’, ‘layout’, ‘frame’ 속성 값들을 설정하여 JSON 구조를 만드는 것이다.

4.1. data : 트레이스의 생성

데이터를 시각화하기 위해서는 점이든 막대이든 원이든 특정한 도형으로 데이터를 표현하여야 한다. 이렇게 데이터를 시각화한 도형 레이어를 트레이스라고 한다. 그렇다면 이 트레이스의 크기, 색상 등과 같은 세부적인 속성들을 설정할 수 있어야 하는데, 트레이스의 세부적인 특성을 설정하는 속성을 ‘data’ 속성이라고 한다. 따라서 ‘data’ 속성은 트레이스로 표현되고 이 트레이스들이 적절하게 구성함으로써 plotly 를 만들게 된다.

plotly 를 잘 이해하기 위해서는 이 트레이스를 잘 이해해야 한다. 앞서 설명한 바와 같이 트레이스는 데이터를 시각화하기 위한 그래픽적 표현방법이지만, 같은 그래픽적 표현방법이라 하더라도 여러개의 트레이스로 구분될 수 있다. 예를 들어 남성과 여성으로 구성된 데이터를 점 산점도로 표시한다고 하면, 그 점이 표현하는 데이터의 특성, 즉 남자와 여자의 특성에 따라 두 개의 트레이스로 구성할 수 있다. 이렇게 구성된 남자와 여자의 트레이스는 각각의 'data' 속성에 따라 크기나 색상들이 달리 표현되고 범례와 호버에 의해서 구분된다.

plotly 에서 plotly 의 초기화 함수를 사용하는 방법과 트레이스를 추가하는 함수를 사용하는 두 가지 방법이 있다.

R 의 `plot_ly()` 나 python 의 `Figure()` 와 같은 plotly 초기화 함수에서 'data' 속성을 구성하여 plotly 객체를 만들 수 있다. 하지만 이 방법은 하나의 함수 코드 길이가 길어지고 리스트와 딕셔너리의 괄호들이 많아져서 매우 복잡해진다.

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

- R

R 에서 트레이스를 만들기 위해서는 먼저 `plot_ly()` 로 초기화 된 plotly 객체에 `add_trace()` 나 `add_markers()`, `add_bars()` 와 같이 트레이스를 추가하는 함수를 사용하여 트레이스를 추가한다. `add_markers()`, `add_bars()` 와 같이 특정 트레이스 전용 함수를 사용할 때는 해당 함수명에 이미 트레이스의 종류가 설정되어 있기 때문에 바로 해당 트레이스에 관련된 'data' 속성을 설정한다. 하지만 `add_trace()` 의 경우는 모든 트레이스의 추가에 사용되는 함수이기 때문에 'type' 속성을 사용하여 먼저 어떤 형태의 트레이스인지 설정해야 한다.

`add_trace()` 이든 각각의 트레이스 전용 함수이든 하나 주의해야할 점은 `plot_ly()` 의 초기화 때 바인딩된 데이터프레임의 열을 속성값으로 할당한다면 반드시 열 이름 앞에 '~'를 붙여줘야 한다는 것이다. 만약 `c()` 를 사용하여 직접 벡터형 변수를 할당한다면 '~'없이 할당할 수 있다.

```
df_취업률_500 |>
  filter(졸업자수 < 500) |>
  plot_ly() |>      ## plotly 초기화
  ## scatter 트레이스에 makers 모드 설정
  add_trace(type = 'scatter', mode = 'markers',
            x = ~졸업자수, y = ~취업자수,
```

```
## marker 사이즈와 색상 설정
```

```
marker = list(size = 3, color = 'darkblue'))
```

- python

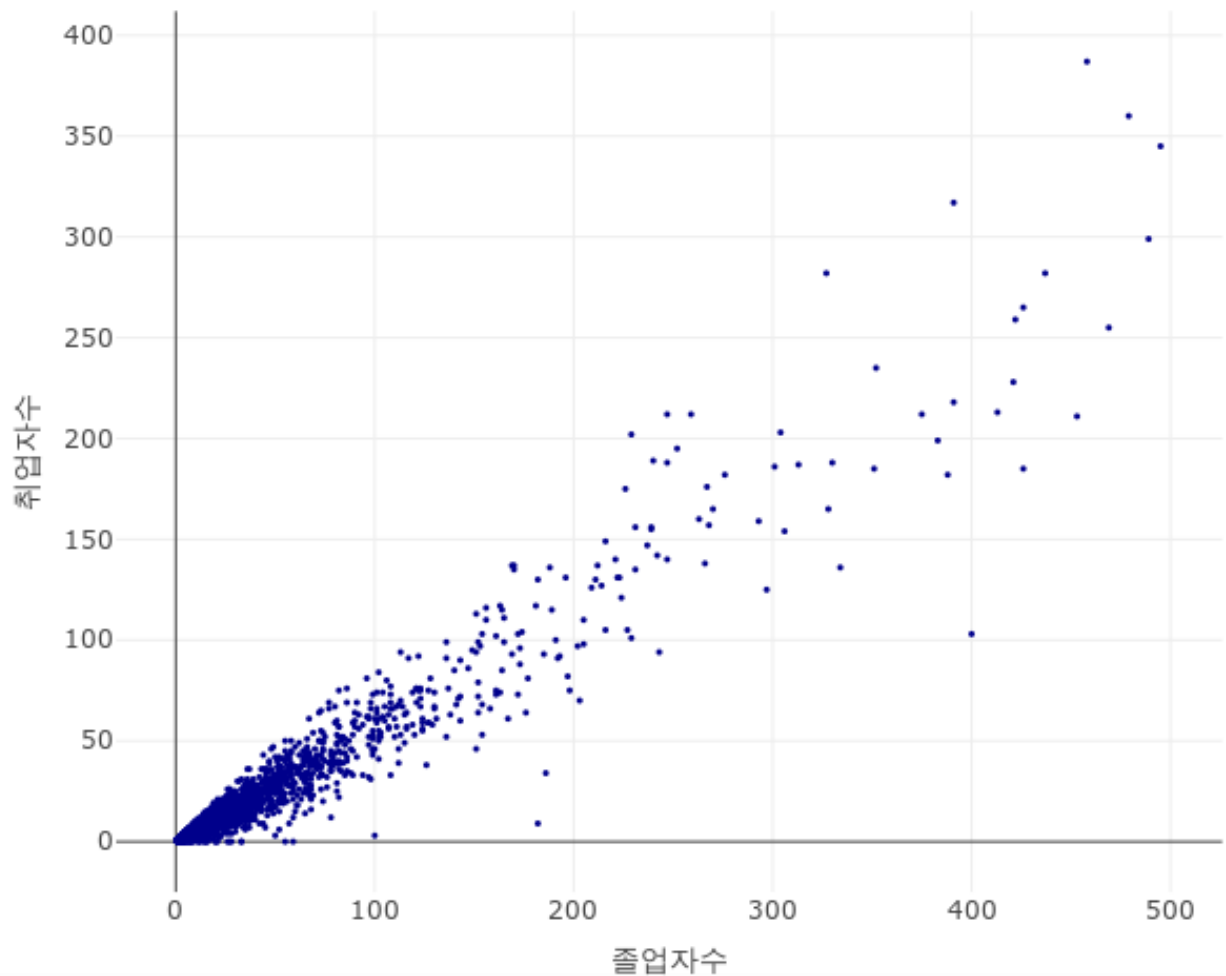
python 에서 트레이스를 설정하기 위해서는 먼저 `plotly.graph_objects` 를 사용할 것인지, `plotly.express` 를 사용할 것인지를 결정해야한다. 여기서는 `plotly.graph_objects` 를 사용해서 `plotly` 를 그리는 두 가지 방법을 설명한다.

`plotly.graph_objects` 를 사용하여 `plotly` 를 그리는 첫 번째 방법은 `add_trace()` 에 'data'의 하위 속성들을 딕셔너리로 구성하여 설정하는 방법이다. `add_trace()` 는 'data' 속성값들의 딕셔너리가 매개 변수로 할당되어야 하고, 이 딕셔너리가 여러개라면 리스트로 묶어 할당한다. 각각의 딕셔너리는 먼저 'type' 속성을 사용하여 어떤 타입의 트레이스인지 설정하고 해당 트레이스에 관련한 'data' 속성들을 설정한다.

```
## plotly 초기화
```

```
fig = go.Figure()
```

```
fig.add_trace({          ## 트레이스 추가
    'type' : 'scatter',   ## scatter 트레이스
    'mode' : 'markers',  ## markers 모드
    'x' : df_취업률_500['졸업자수'], 'y' : df_취업률_500['취업자수'],
    'marker' : { 'size' : 3, 'color' : 'darkblue' } ## marker 크기와 색상설정
})
```



실행결과 II-4. R의 트레이스 생성

두 번째 방법은 `add_trace()`의 매개변수로 `plotly.graph_objects`에서 제공하는 각각의 트레이스 함수를 사용하는 것이다. `plotly.graph_objects`에서 제공하는 각각의 트레이스 함수는 약 90여개이다. 이들을 다 외울 수는 없으니 plotly 홈페이지에서 확인하여 사용하여야 한다.

다음은 scatter 트레이스를 추가하는 `go.Scatter()`에 속성값을 설정하는데 `{}`를 사용하는 방법을 사용한 python 코드이다.

```
fig = go.Figure()

fig.add_trace(go.Scatter(
    {          ## data 속성들로 구성되 딕셔너리 정의
      'type': 'scatter', 'mode': 'markers',
      'x': df_취업률_500['졸업자수'], 'y': df_취업률_500['취업자수'],
```

```
'marker' : {'size' : 3, 'color' : 'darkblue'}
} ## {} 사용
))
```

다음은 속성값을 설정하는데 `dict()`를 사용하는 방법을 사용한 python 코드이다. 앞 서 `{}`를 사용할 때와 다른 것은 전체를 묶어주는 딕셔너리를 구성하지 않는다는 것이다. `{}`의 형태로 속성명과 속성값을 할당할 때는 반드시 전체 구조를 딕셔너리로 묶기 위해 `{}`를 사용하지만 `dict()` 형태의 속성명과 속성값을 할당할 때는 전체 구조를 딕셔너리를 묶어줄 필요가 없다는 것이다. 여러가지 면에서 `{}`보다는 `dict()` 형태의 설정이 편리하기 때문에 3 장 이후는 모두 `dict()` 형태의 설정법을 사용하도록 하겠다.

```
fig = go.Figure()

fig.add_trace(go.Scatter( ## dict() 형태의 속성 할당 방법 사용
    type = 'scatter', mode = 'markers',
    x = df_취업률_500['졸업자수'], y = df_취업률_500['취업자수'],
    marker = dict(size = 3, color = 'darkblue') ## dict() 사용
))
```

4.2. 트레이스 공통 속성

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	속성값(R 속성값)
name		trace 이름을 설정, 설정된 이름은 범례와 호버에 표시	문자열
visible		trace의 표시 여부를 결정, 'legendonly'는 trace는 표시하지 않고 범례만 표시	True False "legendonly"
showlegend		범례를 표현할지를 결정	boolean
opacity		trace의 투명도 설정	0부터 1까지의 수치
x		X축의 설정	list, numpy array, or Pandas series of numbers, strings, or datetimes(dataframe column, list, vector)
y		Y축의 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 text가 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 오버라이드됨, 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, hovertext가 보이려면 'hoverinfo' 속성에 'text'가 포함되어야 함	문자열 또는 문자열 배열
hoverinfo		호버에 어떤 trace 정보가 표시되는지를 결정, 만약 none이나 skip이 설정되면 호버에 아무것도 표시되지 않지만 none이 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		trace의 X좌표와 2차원 X축간의 참조 설정, 만약 'x'가 설정되면 X축은 'layout.xaxis'를 참조하고 'x2'가 설정되면 'layout.xaxis2'를 참조하게 됨	subplotid
yaxis		trace의 Y좌표와 2차원 Y축간의 참조 설정, 만약 'y'가 설정되면 Y축은 '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	0부터 1까지의 수치나 수치배열
hoverlabel	align	호버 문자 박스안의 문자 컨텐트의 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	0부터 1까지의 수치나 수치배열

4.2.1. type

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

- 산점도 타입 : 'scatter', 'scattergl'
- 막대 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. scatter 타입의 트레이스는 점, 선, 문자를 사용하여 데이터를 표현한다. 따라서 스캐터 차트를 그릴때는 세 가지 타입의 표현 방법 중 어떤 것을 사용할지를 설정해야한다. 이것을 설정하는 속성이 'mode'이다. 'mode'는 다음의 네 가지가 있는데 "none"을 제외한 세 가지 'mode'는 '+' 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

트레이스 중 2 차원 데카르트 좌표계를 사용하는 트레이스에서 X, Y 축에 데이터를 매핑하는 속성이 'x', 'y'이다. 데카르트 좌표계를 사용하는 트레이스에서 가장 기본적으로 사용되는 속성이며 하위 속성이 없는 리프노드 속성이다.

- R

R 에서 'x', 'y'에 할당 가능한 변수는 데이터프레임 열, 리스트, 벡터 등이다. 할당할 때 주의 해야 할 것은 `plot_ly()` 초기화시 바인딩된 데이터프레임의 열을 사용한다면 반드시 ~를 열 이름 앞에 붙여줘야 한다.

```
df_취업률_500 |>
  plot_ly() |>
  add_trace(type = 'scatter', mode = 'markers',
            x = ~졸업자수, y = ~취업자수) ## X, Y 축에 매핑되는 변수 설정
```

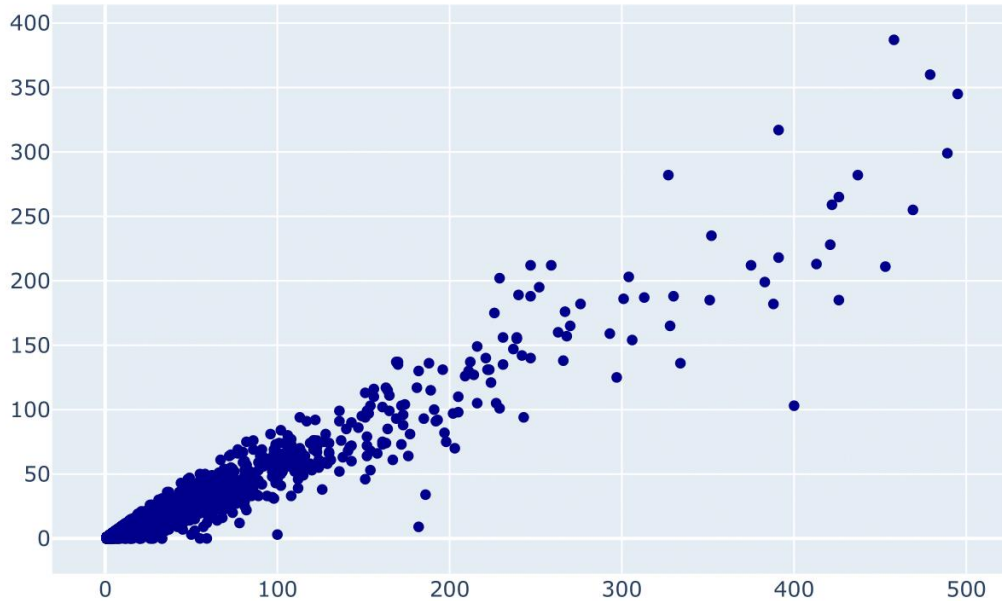
- python

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. `plotly.graph_objects`는 R과 같이 `go.Figure()`의 초기화에 사용할 데이터프레임을 바인딩하지 않기 때문에 'x', 'y'에 데이터프레임 열을 설정할 때는 데이터프레임 열을 인덱싱하여 리스트나 시리즈로 사용하여야 한다.

```
fig = go.Figure()

fig.add_trace(
  {'type' : 'scatter', 'mode' : 'markers',
   'x': df_취업률_500['졸업자수'], ## X, Y 축에 매핑되는 변수 설정
   'y': df_취업률_500['취업자수']})

fig.show()
```



실행결과 II-5. python 의 x, y 속성 설정

4.2.4. name

plotly의 모든 트레이스는 각각의 고유한 이름을 가진다. 이 이름이 'name' 속성이다. 'name'은 plotly에서 범례와 호버에 해당 데이터를 표시하기 위해 사용된다. 따라서 범례에 의해 색상이나 크기로 데이터를 구분하기 위해서는 각각의 고유한 'name' 속성이 설정되어야 한다.

- R

R에서는 'name' 속성을 사용하여 하나의 `add_trace()` 함수로 여러개의 트레이스를 생성할 수 있다. R은 python과 달리 'name'에 단일 문자열을 할당할 수도 있고 데이터 프레임의 열, 문자열 리스트, 문자열 벡터를 설정할 수 있다. 만약 단일 문자열을 할당한다면 `add_trace()`로 생성된 트레이스는 하나의 이름을 가지는 단일 트레이스로 생성된다. 그러나 'name'을 데이터프레임의 열, 리스트, 벡터를 설정한다면 그 데이터의 그룹에 따라 여러개의 트레이스로 분리된다. 따라서 데이터프레임의 열, 리스트, 벡터로 'name'을 설정한다면 표시되는 모든 데이터에 'name' 값이 매칭되도록 설정되어야 한다. 다음의 코드의 예에서 보면 'name' 속성에 `df_취업률_500`

데이터프레임의 대계열 열을 할당하였다. 대계열 값은 총 7 개인데, 'x', 'y'에 매핑된 졸업자수, 취업자수에 의해 표시되는 데이터들은 대계열 7 개 중에 하나의 값을 가진다. 따라서 표시되는 모든 데이터는 'name' 속성에 따라 7 개의 트레이스로 분리되어 생성된다.

```
df_취업률_500 |>
  plot_ly() |>
  add_trace(type = 'scatter', mode = 'markers',
            x = ~졸업자수, y = ~취업자수,
            name = ~대계열) ## name 속성 설정
```

- 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 표시되는 데이터를 그 특성에 따라 구분하기 위해서는 데이터의 특성 값에 따른 그룹별로 `add_trace()`를 사용하여 트레이스를 추가해야 한다. 이 때 각각의 그룹의 특성을 잘 반영하는 트레이스 이름을 'name' 속성에 설정하여야 한다.

만약 데이터가 넓은 형태로 구성되어 있다면 각각의 열별로 `add_trace()`를 사용하여 트레이스를 추가하는 방식으로 사용할 수 있다. 만약 데이터가 긴 형태로 구성되어 있다면 `groupby()`를 사용하여 데이터를 그룹화하고, 그룹화된 세부 그룹 데이터프레임을 ~~for 루프를 사용하여~~ 각각의 그룹을 `add_trace()`로 추가해주는 방식으로 사용할 수 있다.

하지만 `plotly.express` 에서 제공하는 함수들은 R 과 같이 'name'에 데이터프레임 열을 설정할 수 있어 다소 간단히 설정할 수 있다.

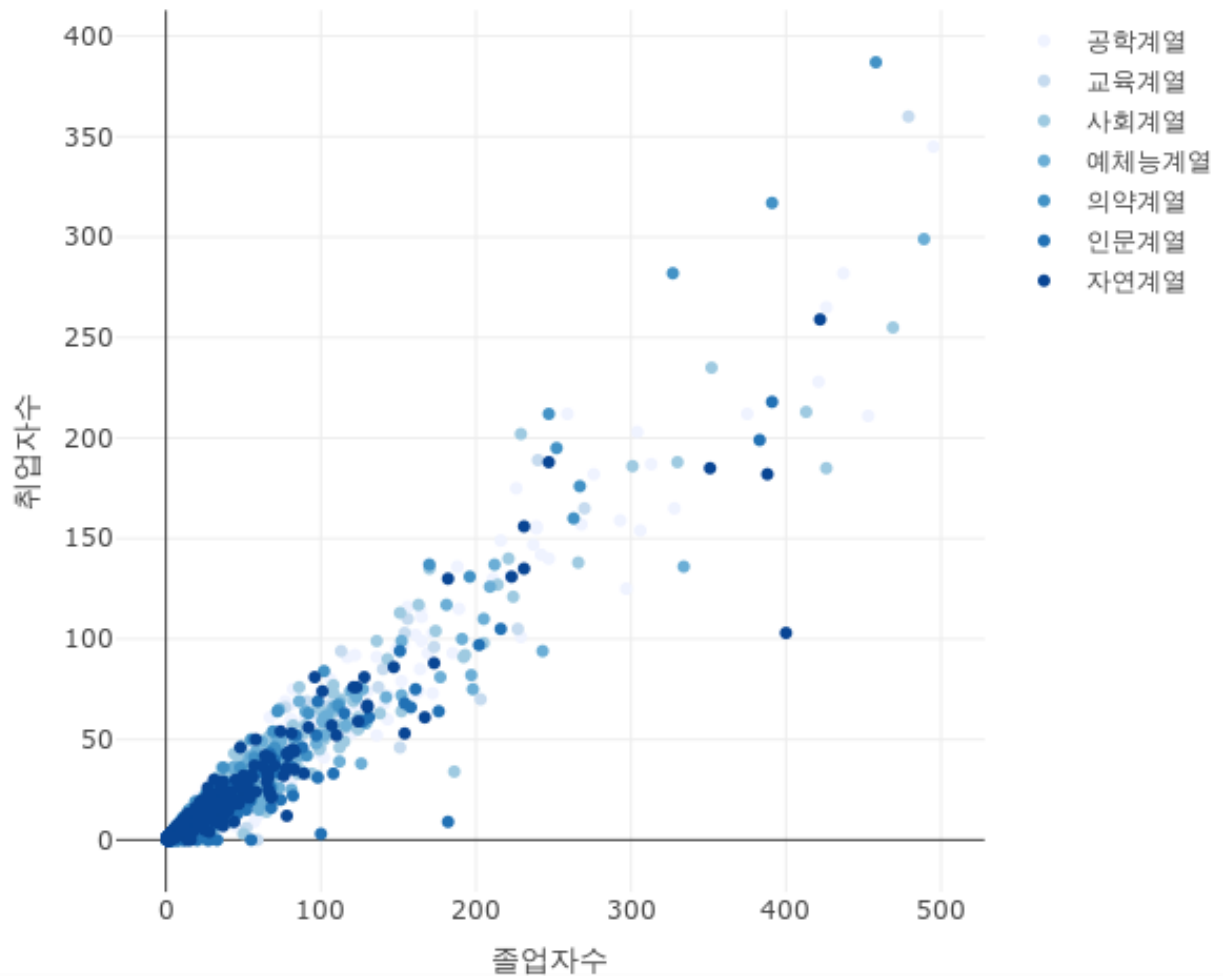
다음의 코드는 `df_취업률_500` 의 학과별 졸업자수와 취업자수를 시각화하는데 그 학과의 대계열 값에 따라 트레이스를 분리하여 추가하는 코드이다. `df_취업률_500` 데이터프레임을 대계열로 그룹화하고, `for` 루프를 사용하여 각각의 서브 그룹들에 대해 `add_trace()`로 트레이스를 추가한다. 이 때에 각각의 트레이스 이름을 'name' 속성으로 지정한다.

```
fig = go.Figure()

for 대계열, group in df_취업률_500.groupby('대계열'):
    fig.add_trace({
        'type' : 'scatter', 'mode' : 'markers',
```

```
'x': group['졸업자수'], 'y': group['취업자수'],
'name': 대계열)) ## name 속성 설정
```

```
fig.show()
```



실행결과 II-6. R의 name 속성 설정

4.2.5. 데이터 값의 표시

데이터 시각화에서 데이터는 다양한 그래픽적 기하 도형으로 표현되기 때문에 데이터의 정확한 값을 측정하는데 어려움이 있다. 이를 보완하기 위해 표시되는 각각의 데이터 트레이스에 데이터 값을 표기하는 경우가 많다. plotly에서 지원하는 대부분의 트레이스는 'text' 속성을 사용해서 트레이스에 데이터 값을 표시할 수 있다. 또 'textposition'과 'texttemplate'를 사용하여 표시되는 값의 위치나 표현 형태를 설정할 수 있다.

4.2.5.1. text

‘data’ 하위 속성 중 ‘text’ 속성은 트레이스에 표시되는 문자열을 지정하는 속성이다. 이 속성은 하위 속성값을 가지지 않는 리프 노드 속성이기 때문에 단일 문자열을 지정하여 모든 데이터에 동일한 문자열이 표시되게 하거나 각각의 데이터에 1:1 매칭되는 고유한 문자열 배열을 설정해 각 데이터에 따른 문자열을 표시해 줄수도 있다.

- R

R에서 ‘text’ 속성에 할당할 수 있는 데이터는 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 1:1로 매핑되어 해당 데이터에 매핑된 문자열이 표시된다. 만약 plotly 초기화때 바인딩된 데이터프레임의 열을 사용한다면 ‘~’를 붙여주어야 한다.

```
## 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', ## bar 트레이스 설정
            x = ~location, y = ~new_cases,
            text = ~new_cases) ## 텍스트 설정
```

- python

python에서 ‘text’ 속성에는 문자열 또는 문자열 배열을 할당할 수 있다. 앞서 ‘name’ 속성때는 문자열 배열을 설정할 수 없었기 때문에 ~~for를 사용한 루프를 사용했지만~~ ‘text’ 속성은 문자열 배열을 사용할 수 있기 때문에 표현되는 데이터에 1:1로 매핑되는 문자열을 표시할 수 있다.

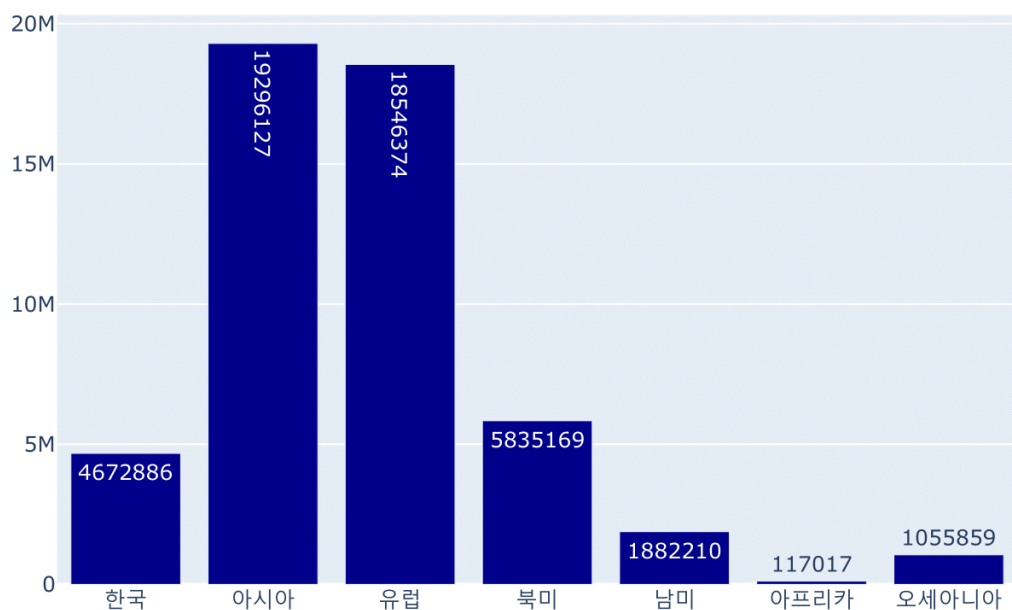
```
fig = go.Figure()

## 긴 형태의 100 일간 코로나 19 데이터 중에 국가명으로 그룹화, 확진자수의 합계를 new_cases 로 산출하여 temp 에 저장
```

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

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

fig.show()
```



실행결과 II-7. python 의 text 속성 설정

4.2.5.2. textposition

'textposition'는 'text'의 위치를 설정하는 속성이다. 'textposition'은 리프 노드 속성으로 "inside", "outside", "auto", "none"의 네 가지 플래그 문자열 중 하나를 설정한다. "inside"는 막대의 안쪽에 텍스트를 위치시킨다. 이 경우 막대의 너비와 표시되는 텍스트의 길이에 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

다음은 R 에서 'textposition'의 설정에 따른 결과를 보여준다. 설명한 바와 같이 막대의 너비에 따라 문자열이 가로 혹은 세로로 표시된다.

```
#####  
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 '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')
```

- python

다음은 python 에서 'textposition'의 설정에 따른 결과를 보여준다. 설명한 바와 같이 막대의 너비에 따라 문자열이 가로 혹은 세로로 표시된다.

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

```
#####
```

```
fig = go.Figure()
```

```
fig.add_trace({
```

```
    'type' : 'bar', 'x': temp.index, 'y': temp['new_cases'],
```

```
    'text' : temp['new_cases'],
```

```
    'textposition' : 'inside'}) ## textposition 을 'inside'로 설정
```

```
fig.show()
```

```
#####
```

```
fig = go.Figure()
```

```
fig.add_trace({
```

```
    'type' : 'bar', 'x': temp.index, 'y': temp['new_cases'],
```

```
    'text' : temp['new_cases'],
```

```
'textposition' : 'outside' ## textposition 을 'outside'로 설정
})
```

```
fig.show()
```

```
#####
```

```
fig = go.Figure()
```

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

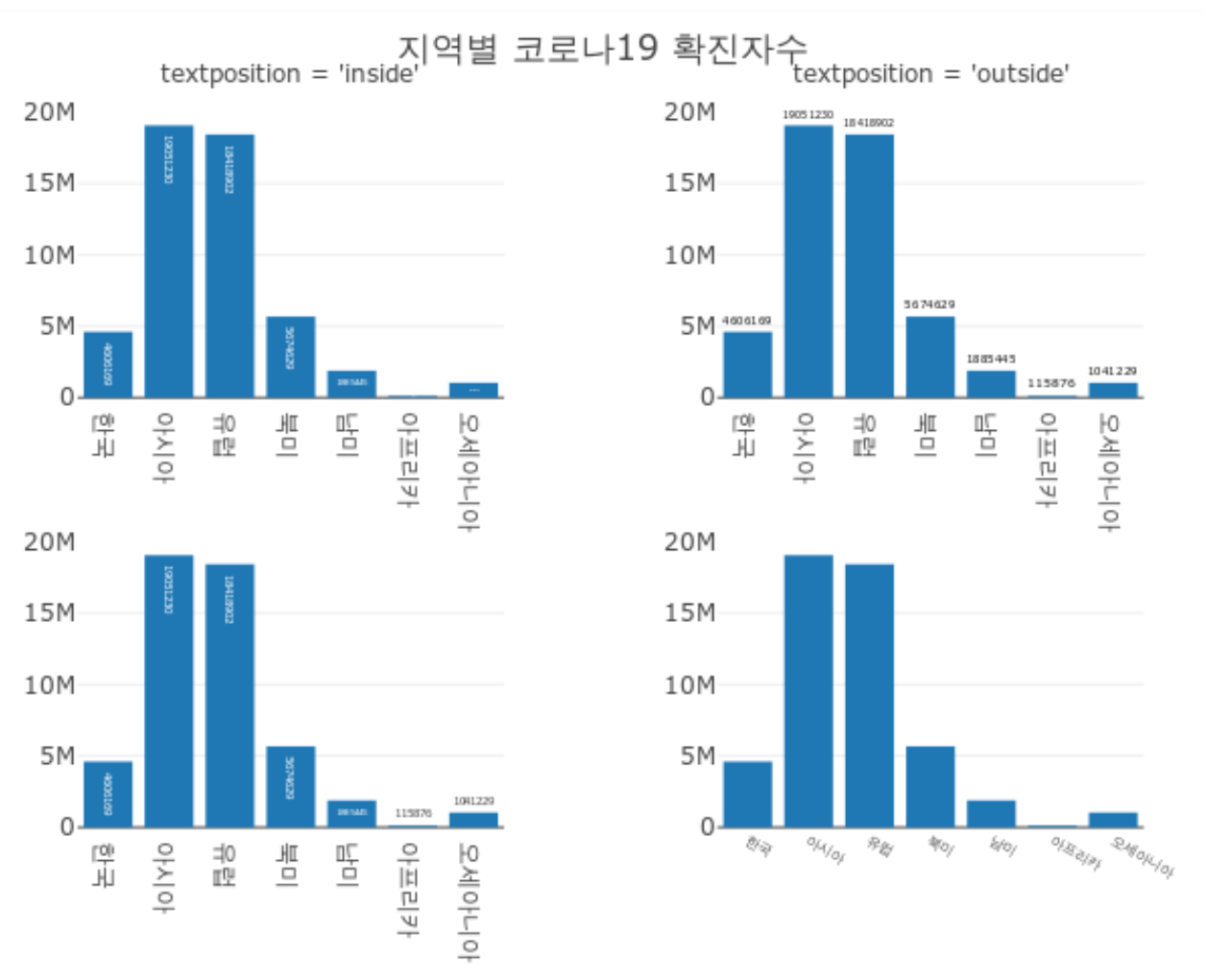
```
fig.show()
```

```
#####
```

```
fig = go.Figure()
```

```
fig.add_trace({
    'type' : 'bar', 'x': temp.index, 'y': temp['new_cases'],
    'text' : temp['new_cases'],
    'textposition' : 'none' ## textposition 을 'none'로 설정
})
```

```
fig.show()
```



실행결과 II-8. R의 textposition 속성 설정

4.2.5.3. texttemplate

‘texttemplate’ 속성은 텍스트가 표시되는 형태를 설정하는 속성이다. ‘texttemplate’ 속성을 사용하여 텍스트가 표시되는 전체 문자열을 설정할 수도 있고 데이터가 표시되는 포맷을 설정할 수도 있다.

텍스트에 트레이스의 속성값을 변수로 표시해야 한다면 ‘%{속성이름}’의 형태로 속성값을 변수로 포함시킬수 있다. ‘%{속성이름}’으로 설정된 부분은 해당 데이터의 속성값으로 대체되어 표시된다.

앞선 ‘textposition’의 코드에서 ‘x’, ‘y’, ‘text’의 세 가지 속성을 사용했다. 이 경우 ‘texttemplate’에서 사용할 수 있는 속성은 “%{x}”, “%{y}”, “%{text}”의 세 가지이다.

이 속성 값에 대한 표시형식을 설정하고자 한다면 “%{속성이름:포맷}”의 형태로 사용할 수 있다. 포맷의 지정 방식은 자바 스크립트의 d3 format⁵을 사용한다. 표시할 텍스트 수치를 천단위 콤마가 포함된 포맷으로 설정하고자 한다면 “%{text:,}”로 설정하고 소수점 아래 두째 자리까지 표기한다면 “%{text:2f}”로 설정할 수 있다.

다음은 ‘texttemplate’ 속성을 사용하여 “확진자수:”를 표기한 후 확진자수를 표시하는데 천 단위 콤마를 포함한 포맷으로 표시하는 R 과 python 코드이다.

- R

```
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 = 'inside',
            texttemplate = '확진자수:%{text:,}') ## texttemplate 를 설정
```

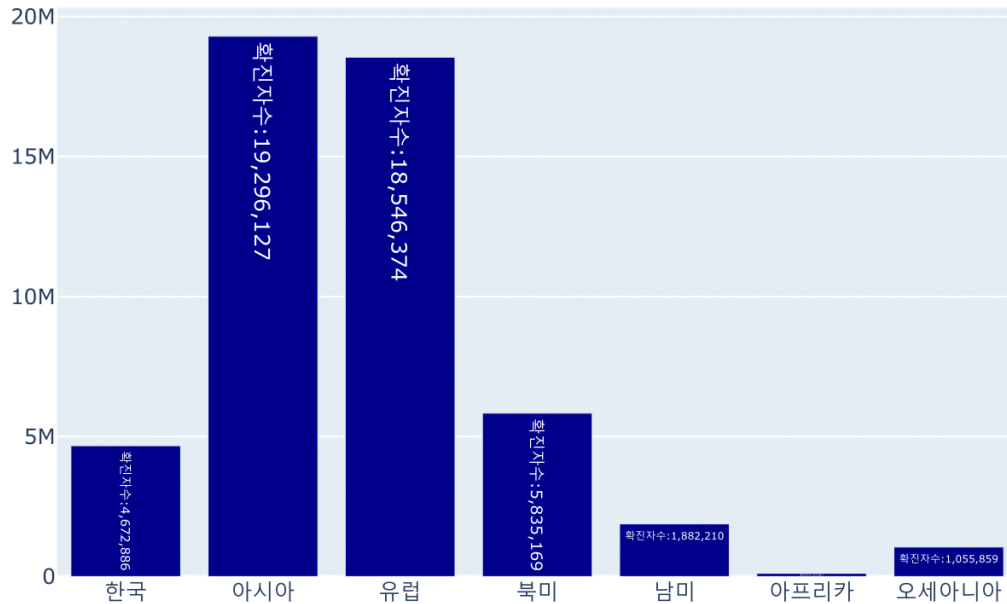
- python

```
fig = go.Figure()

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

fig.show()
```

⁵ <https://github.com/d3/d3-format/tree/v1.4.5#d3-format>



실행결과 II-9. python 의 texttemplate 속성 설정

4.2.6. 호버

plotly 와 같은 동적 시각화에서는 대부분 마우스 포인터를 데이터가 표시된 점이나 선에 위치하면 해당 위치의 데이터에 대한 정보가 표시된다. plotly 에서는 이렇게 데이터의 정보를 표시하는 말풍선을 호버(hover)라고 한다. 'hover'를 설정하는 속성들은 여러가지가 있지만 모두 'hover'로 시작한다. 호버는 트레이스의 종류마다 표시되는 호버가 다르기 때문에 각각의 트레이스마다 설정하는 항목이 다르지만, 대부분의 트레이스에서 공통으로 사용되는 호버 설정 속성들은 다음과 같다.

4.2.6.1. hoverinfo

'hoverinfo' 속성은 호버에 표시되는 데이터 정보를 설정하는 속성으로 "x"(X 축 좌표값), "y"(Y 축 좌표값), "z"(Z 축 좌표값), "text"('hovertext' 속성값), "name"(trace 이름), "none"(호버 제거), "skip"(생략)이 사용될 수 있고 각각은 +로 조합하여 여러개를 동시에 사용할 수 있다.

다음은 'hoverinfo' 속성을 'y'로 설정하여 호버값에 Y 축 값만 표시되도록 설정한 R 과 python 코드이다.

- R

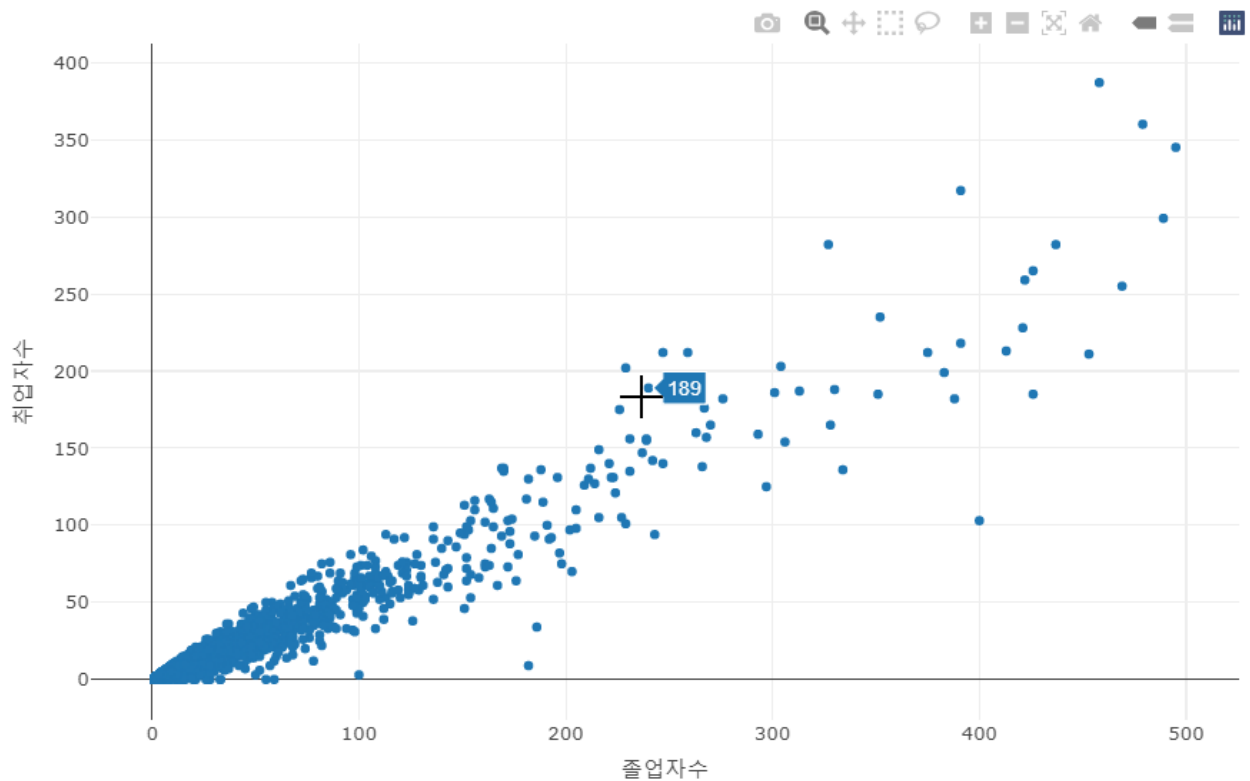
```
df_취업률_500 |>
  plot_ly() |>
  add_trace(type = 'scatter', mode = 'markers',
            x = ~졸업자수, y = ~취업자수,
            hoverinfo = 'y') ## hoverinfo 설정
```

- python

```
fig = go.Figure()

fig.add_trace({
    'type': 'scatter', 'mode': 'markers',
    'x': df_취업률_500['졸업자수'], 'y': df_취업률_500['취업자수'],
    'hoverinfo': 'y'}) ## hoverinfo 설정

fig.show()
```



실행결과 II-10. R의 `hoverinfo` 속성 설정

4.2.6.2. `hovertext`

'`hovertext`' 속성은 X, Y 좌표에 해당하는 데이터에 대한 호버 정보를 설정하는 속성이다. 이 속성에는 단일 문자열이나 문자열 배열을 설정할 수 있다. 단일 문자열을 설정하면 모든 X, Y 좌표에 해당하는 데이터에 모두 같은 문자열이 호버에 표시된다. 반면 문자열 벡터를 설정하면 X, Y 좌표에 해당하는 데이터에 연관된 문자열을 호버에 표시해 준다. '`hovertext`'에 설정된 문자열이 호버에 표시되기 위해서는 반드시 '`hoverinfo`' 속성에 "text"가 포함되어야 한다.

다음은 호버에 기본적으로 표시되는 졸업자수와 취업자수 외에 해당 학과의 중계열, 소계열 정보를 추가적으로 표시해주는 R과 python 코드이다.

- R

'`hovertext`' 속성에 문자열을 설정하기 위해서 `paste0()`나 `paste()`를 사용할 수 있다. 여기서 하나 주의할 것은 'text' 속성값을 설정할 때 바인딩된 데이터프레임의 열을 속성값을 사용할 때 '~'를 붙여주었지만 `paste0()`나 `paste()`에 데이터프레임 열을 사용한다면 '~'을 열 이름 앞이 아닌 `paste0()`나 `paste()` 앞에 붙여야 한다는 것이다. 만약 `paste0()`나 `paste()`를 사용하지 않고 바로 데이터프레임 열을 사용한다면 여전히 열 앞에 '~'를 사용하여야 한다.

```
df_취업률_500 |>
  plot_ly() |>
  add_trace(type = 'scatter', mode = 'markers',
            x = ~졸업자수, y = ~취업자수,
            ## hovertext 의 설정
            hovertext = ~paste0('중계열:', 중계열, '\n', '소계열:', 소계열))
```

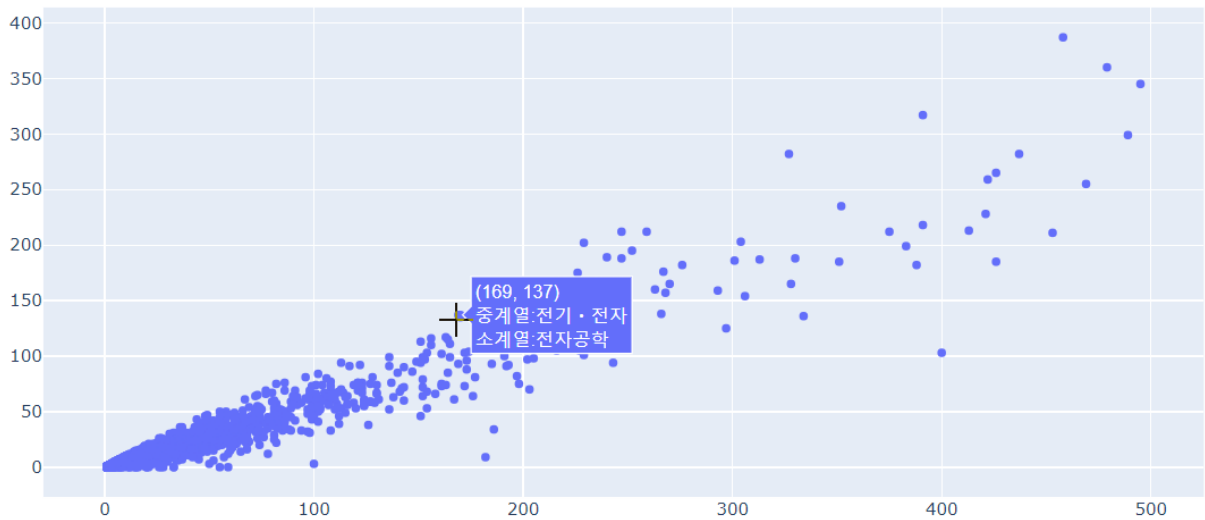
- python

python 에서 'hovertext' 속성 표시할 데이터와 문자열을 '+'로 연결하여 하나의 문자열로 만들고, 이 문자열을 설정해 준다. 만약 데이터가 수치일 경우 이를 문자열로 변환하여야 문자열로 연결이 가능하다.

```
fig = go.Figure()

fig.add_trace({
  'type' : 'scatter', 'mode' : 'markers',
  'x': df_취업률_500['졸업자수'], 'y': df_취업률_500['취업자수'],
  ## hovertext 의 설정
  'hovertext' : '중계열:'+df_취업률_500['중계열']+'<br>'+소계열:'+df_취업률_500['소계열']
})

fig.show()
```

실행결과 II-11. python 의 hovertext 속성 설정

4.2.6.3. hovertemplate

'hovertemplate'는 호버에 표시되는 정보와 표시 형식을 결정하는 템플릿 포맷을 설정하는 속성이다. 이 속성은 앞서 설명한 'texttemplate'와 유사하게 설정한다.

호버에 트레이스의 속성값을 변수로 표시해야 한다면 '%{속성이름}'의 형태로 속성값을 변수로 포함시킬수 있다. '%{속성이름}'으로 설정된 부분은 해당 데이터의 속성값으로 대체되어 표시된다.

이 속성 값에 대한 표시형식을 설정하고자 한다면 "%{속성이름:포맷}"의 형태로 사용할 수 있다. 포맷의 지정 방식은 자바 스크립트의 d3 format 을 사용한다. 표시할 텍스트로 X 축의 값 수치, 형식을 천단위 콤마가 포함된 포맷으로 설정하고자 한다면 "%{x:,}"로 설정하고 소수점 아래 두째 자리까지 표기한다면 "%{x:2f}"로 설정할 수 있다.

앞서 설명한 'hovertext'와 'hovertemplate'는 모두 호버에 표시되는 정보를 설정한다는 동일한 기능을 가지고 있다. 하지만 몇 가지 차이점이 있다.

첫 번째 차이점은 'hovertext'는 호버 우측에 트레이스 이름이 표시되지 않지만 'hovertemplate'는 트레이스 이름이 표기된다. 물론 'hovertemplate'에서도 " "를 붙여주면 트레이스 이름을 제거할 수 있지만 기본적으로 표시된다.

두 번째는 'hovertemplate'에서는 d3 format 으로 표시되는 데이터의 형태를 쉽게 표시할 수 있지만 'hovertext'에서는 데이터의 포맷 설정기능을 제공하지 않는다.

세번째는 'hovertext'에서는 표시하는 변수에 특별한 제한이 없지만 'hovertemplate'에서는 속성으로 설정된 속성값만 변수로 사용할 수 있다.

다음은 호버에 “졸업자:”, X 축 값, “취업자:”, Y 축의 값, “대계열:”, 대계열 값을 표시는 R 과 python 코드이다.

- R

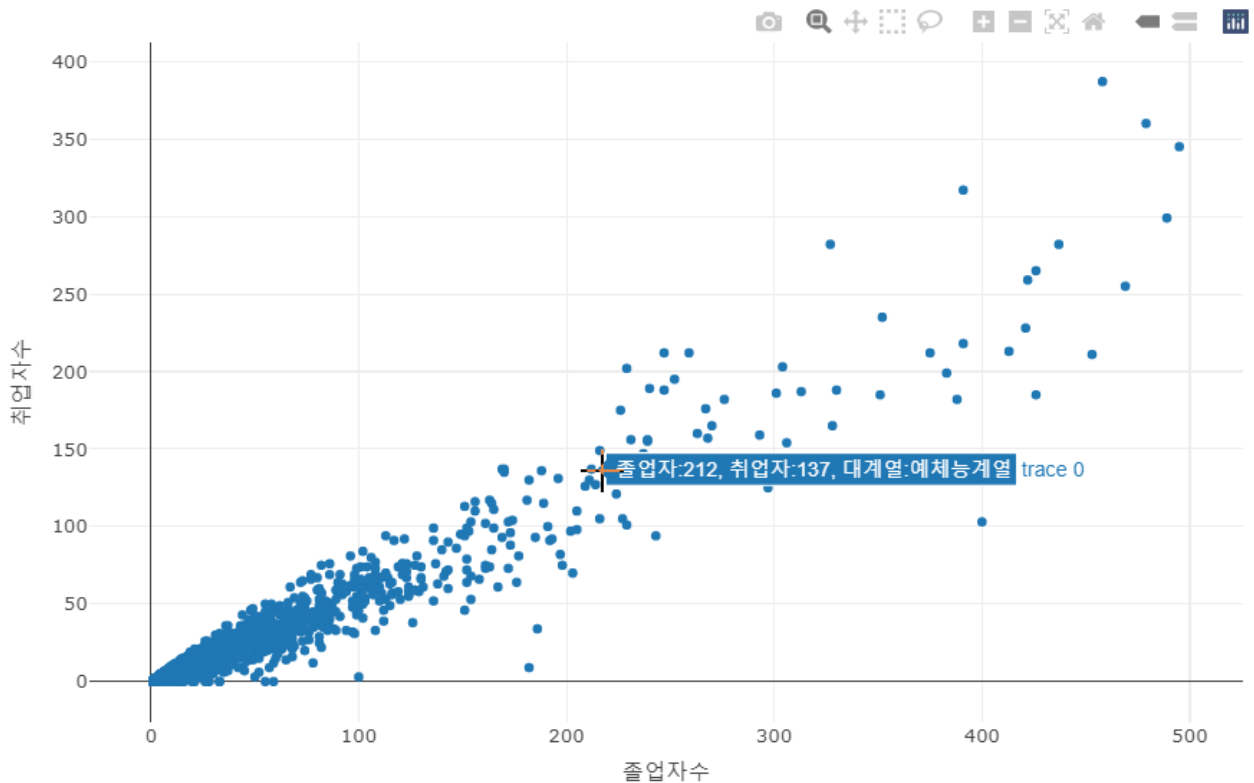
```
df_취업률_500 |>
plot_ly() |>
add_trace(type = 'scatter', mode = 'markers',
          x = ~졸업자수, y = ~취업자수, hovertext = ~대계열,
          ## hovertemplate 의 설정
          hovertemplate = ' 졸업자:%{x}, 취업자:%{y}, 대계열:%{hovertext}')
```

- python

```
fig = go.Figure()

fig.add_trace({
    'type' : 'scatter', 'mode' : 'markers',
    'x': df_취업률_500['졸업자수'], 'y': df_취업률_500['취업자수'],
    'hovertext' : df_취업률_500['중계열'],
    ## hovertemplate 의 설정
    'hovertemplate' : ' 졸업자:%{x}, 취업자:%{y}, 대계열:%{hovertext}'})

fig.show()
```



실행결과 II-12. R의 *hovertemplate* 속성 설정

4.2.7. opacity, alpha

'opacity'는 투명도를 설정하는 속성이다. 투명도는 0 부터 1 사이의 값을 가지는데 0 은 투명하고 1 은 불투명하다. 여기서 하나 주의해야할 점은 'opacity'가 어디에서 정의되는지이다. 'opacity'는 'marker' 속성 내에서 설정할 수도 있고 'marker' 설정 밖에서 설정할 수도 있다. 'marker'의 속성 내에 설정되면 서로 겹치는 부분의 투명도가 서로간의 영향을 받게 된다. 그러나 'marker'의 외부에 설정되면 해당 트레이스 전체에 대한 투명도가 설정되기 때문에 겹치는 부분에 대해서 서로간의 영향을 받지 않는다. 만약 'marker'의 투명도를 설정하게 된다면 가급적 'opacity'의 값은 0.5 이하로 설정하는 것이 바람직하다. 그래야 2 개 이상의 데이터가 겹칠 때 효과를 나타낼 수 있다.

반면 투명도는 'alpha'를 사용하여 설정할 수도 있다. 다만 'alpha'는 각각의 색상 채널에 투명도가 적용된다. 따라서 'marker'의 내부 색에 'alpha' 속성 값을 설정하면 내부색에만 투명도가 설정된다. 만약 외곽선이 존재한다면 이 선의 색상에는 'alpha' 속성이 영향받지 않기 때문에 선들은 투명도가 설정되지 않는다.

다음은 'marker' 내에서 설정한 'opacity'와 'marker' 밖에서 설정한 'opacity'의 결과를 보여주는 R 과 python 코드이다.

- R

```
df_취업률_500 |> plot_ly() |>
  add_trace(type = 'scatter', mode = 'markers',
            x = ~졸업자수, y = ~취업자수,
            ## marker 내부에서 opacity 설정
            marker = list(opacity = 0.3, color = 'darkblue'))

df_취업률_500 |> plot_ly() |>
  add_trace(type = 'scatter', mode = 'markers',
            x = ~졸업자수, y = ~취업자수,
            marker = list(color = 'darkblue'),
            ## marker 외부에서 opacity 설정
            opacity = 0.3)
```

- python

```
#####
fig = go.Figure()

fig.add_trace({
  'type' : 'scatter', 'mode' : 'markers',
  'x': df_취업률_500['졸업자수'], 'y': df_취업률_500['취업자수'],
  ## marker 외부에서 opacity 설정
  'opacity' : 0.3}) ## opacity 를 0.3 으로 설정

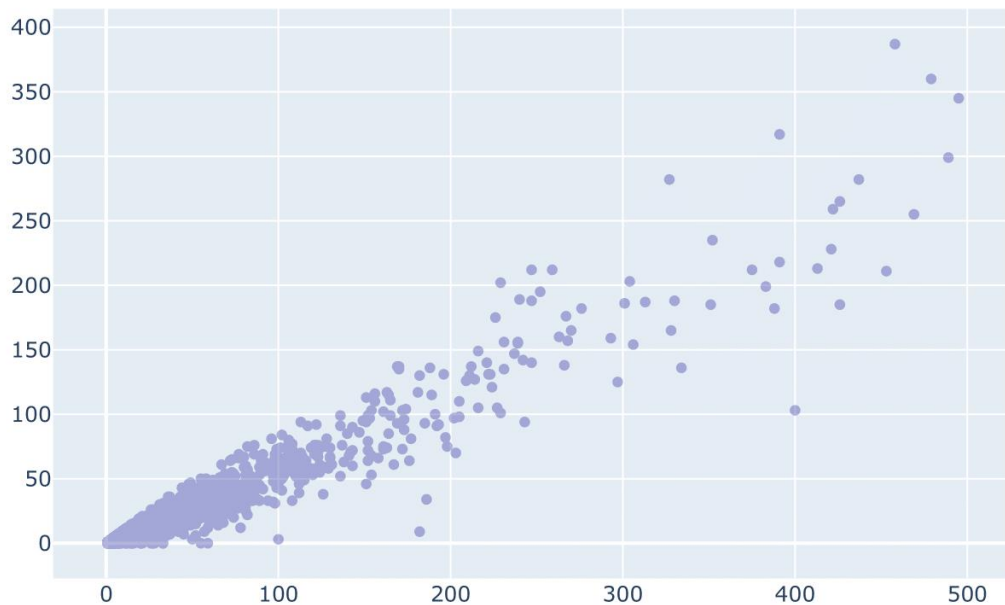
fig.show()

#####
fig = go.Figure()

fig.add_trace({
  'type' : 'scatter', 'mode' : 'markers',
  'x': df_취업률_500['졸업자수'], 'y': df_취업률_500['취업자수'],
```

```
## marker 내부에서 opacity 설정
'marker' : {'opacity' : 0.3}) ## opacity 를 0.3 으로 설정
```

```
fig.show()
```



실행결과 II-13. python 의 opacity 속성 설정

4.2.8. showlegend

‘showlegend’는 범례를 표기할지 여부를 설정하는 논리값(TRUE/FALSE) 속성으로 리프 노드 속성이기 때문에 바로 속성값을 설정한다. 이 속성은 각각의 트레이스에 사용되면 해당 트레이스에 대한 범례의 표시를 제어하고 ‘layout’ 속성에서 사용되면 전체 범례의 표시를 제어할 수 있다.

- R

R에서는 ‘showlegend’ 속성을 “TRUE” 또는 “FALSE”로 설정함으로써 범례를 표시하거나 없앨 수 있다.

```
df_취업률_500 |>
  plot_ly() |>
  add_trace(type = 'scatter', mode = 'markers',
            x = ~졸업자수, y = ~취업자수, name = ~대계열,
            showlegend = FALSE) ## showlegend 을 FALSE 로 설정
```

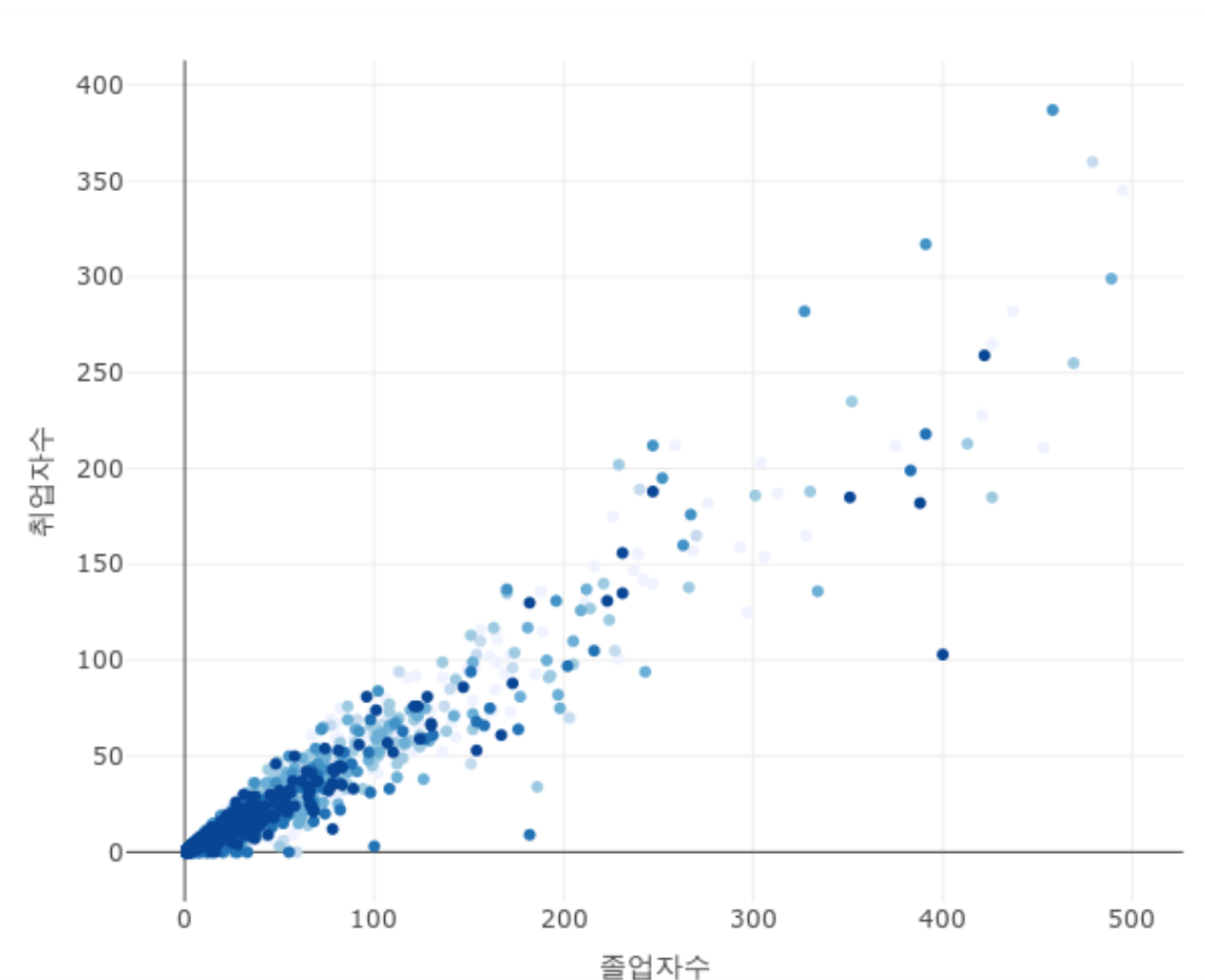
- python

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

```
fig = go.Figure()

for 대계열, group in df_취업률_500.groupby('대계열'):
    fig.add_trace({
        'type': 'scatter', 'mode': 'markers',
        'x': group['졸업자수'], 'y': group['취업자수'],
        'name': 대계열,
        'showlegend': False}) ## showlegend 을 FALSE 로 설정

fig.show()
```



실행결과 II-14. R의 *showlegend* 속성 설정

4.3. layout

지금까지는 'data' 속성을 사용하여 데이터를 표현하는 트레이스를 구성하는 방법에 대해 알아보았다. 반면 plotly 에서 트레이스와 직접적으로 연관되지 않는 다양한 시각화 속성들도 있는데 이를 'layout' 속성이라고 한다. 'layout' 속성으로 다음과 같은 시각화 요소들을 설정할 수 있다.

- 전체 플롯의 크기(Dimension)과 여백(Margin)
- 전체 플롯의 템플릿, 테마, 폰트, 색상, 호버, 모드바의 기본 설정

- 제목과 범례의 위치 설정
- 컬러 바에 연관된 색상 축(color axis) 설정
- 다중 트레이스가 사용되는 서브 플롯의 다양한 타입 설정
- annotations, shapes, images 와 같은 데이터와 관련없는 시각화 설정
- updatemenus, sliders 와 같은 사용자와 상호작용하는 컨트롤 설정

4.3.1. layout 속성 설정

R 에서 'layout' 속성을 설정하기 위해서는 `layout()` 을 사용한다. plotly 객체를 `layout()` 의 첫 번째 매개변수로 할당하여야 하고, 이 후 설정하고자 하는 'layout' 속성들을 설정함으로써 전체 'layout'을 설정하게 된다. 속성을 할당하는 방법은 `add_trace()` 와 동일하다.

python 에서는 plotly 객체에 `plotly.graph_objects.update_layout()` 을 사용하여 'layout' 속성을 설정한다. 속성을 할당하는 방법은 `plotly.graph_objects.add_trace()` 와 동일하다.

4.3.2. layout 공통 주요 속성

'layout'은 대체적으로 대부분의 속성들에 공통적으로 적용되지만 특정 트레이스에서만 사용되는 속성들도 있다. 다음은 'layout'의 공통 속성들 중에 주요 속성은 title, color, axis, legend, margin 등이 있다.

4.3.2.1. 제목(title) 설정

plotly 의 제목을 설정하는 속성은 'title'이다. 'title'의 주요 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	1이상의 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		xref'로부터의 x축 위치 설정	0부터 1사이의 수치
	xanchor		제목의 수평 정렬 설정	"auto" "left" "center" "right"
	xref		x축의 위치 설정 기준. Container는 plot의 전체, Paper는 플로팅 영역	"container" "paper"
	y		xref'로부터의 y축 위치 설정	0부터 1사이의 수치
	yanchor		제목의 수직 정렬 설정	"auto" "top" "middle" "bottom"
	xref		y축의 위치 설정 기준. Container는 plot의 전체, Paper는 플로팅 영역	"container" "paper"

이 속성은 다른 속성과는 조금 다른 성질이 있다. 원칙적으로 'title'은 'layout'의 첫 레벨 속성으로 세부 속성들의 컨테이너 역할을 하는 속성 이름이다. 하지만 'title'은 리프 노드 속성으로도 사용이 가능하다. 'title'에 설정되는 값이 'title'의 세부 속성에 대한 list나 dict라면 컨테이너 속성 노드로 취급되고 문자열이 설정되면 리프 속성 노드로서 설정된 문자열이 시각화 전체 제목으로 설정되는 동적 속성이다.

앞선 'title'의 속성 중 제목의 폰트 설정과 관련된 하위 속성은 'font', 'family', 'size'의 세 가지 속성만을 제공한다. 하지만 플롯 제목은 색상이나 굵기, 기울여쓰기 등 다양한 문자 속성의 설정이 필요하다. plotly에서는 제목과 같은 문자열을 꾸미기 위해 HTML 텍스트 태그를 지원한다. 하지만 전체 HTML 태그 중 다음의 5 가지 HTML 텍스트 태그만을 지원한다.

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

- R

R 에서 'title' 속성을 설정하기 위해서는 `layout()`에서 'title' 키워드에 'title' 하위 속성들을 list 로 설정한다. 앞서 설명했다시피 'title'에 바로 문자열을 설정한다면 'title.text'에 문자열을 설정한 것과 동일한 효과가 있다. 하지만 이 방법은 단순히 제목 문자열만 설정할 수 있을뿐 'title'의 다른 하위 속성들을 설정할 수 없다는 단점이 있다.

다음의 코드는 'title' 속성을 설정하는 R 코드이다. 'text'에 제목으로 사용할 문자열을 설정하였는데, 이를 사용하여 볼드체로 설정하였고 'x' 속성으로 x 축 방향의 위치를 전체의 중간(0.5), 'xanchor'와 'yanchor'를 "center"와 "top"으로 설정하여 제목의 상세 위치를 설정하였다.

```
R_layout_scatter <- df_취업률_500 |>
  filter(졸업자수 < 500) |>
  plot_ly() |>
  add_trace(type = 'scatter', mode = 'markers',
            x = ~졸업자수, y = ~취업자수) |>
  ## title 속성의 설정
  layout(title = list(text = '<b>졸업자 대비 취업자수</b>',
                      x = 0.5, xanchor = 'center', yanchor = 'top'))

R_layout_scatter
```

- python

python 에서 'layout'을 설정하기 위해서는 `plotly.graph_objects`의 초기화 함수인 `Figure()`에서 설정하거나 `add_trace()`로 트레이스가 설정된 plotly 객체에 `update_layout()`을 사용할 수 있다.

다음은 `Figure()`에 `fig`를 사용하여 'layout'을 설정하는 python 코드이다.

```
fig = go.Figure()

go.Figure(
    data=[
        {'type': 'scatter', 'mode': 'markers',
         'x': df_취업률_500['졸업자수'], 'y': df_취업률_500['취업자수']}
    ],
    ## fig를 사용한 title 속성의 설정
    layout={
        'title': {'text': "<b>졸업자 대비 취업자수</b>"},
        'x': 0.5, 'xanchor': 'center', 'yanchor': 'top'
    }
)
```

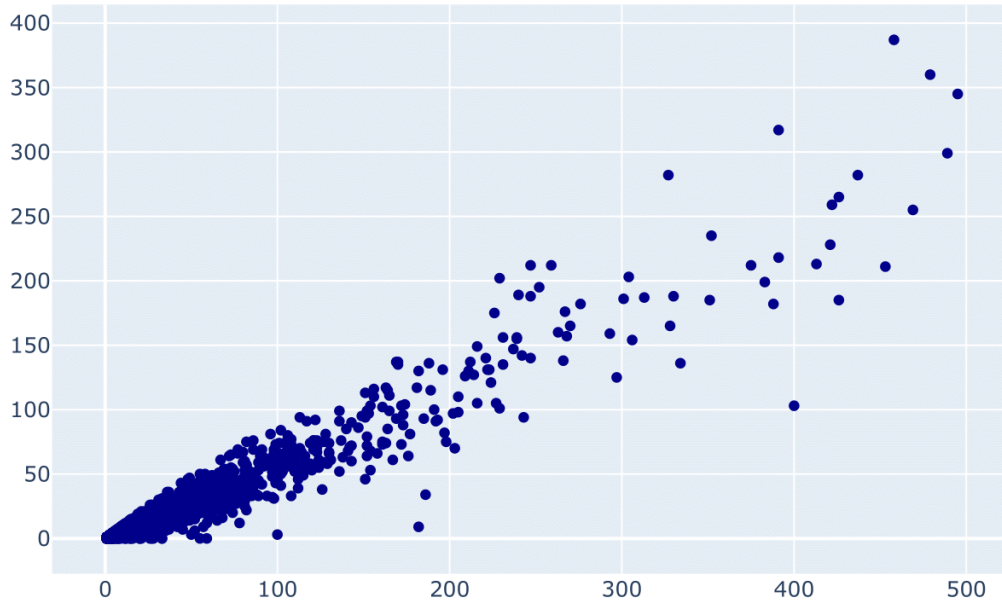
`update_layout()`를 사용할 때 주의해야 하는 것은 `add_trace()`와 같이 'layout' 키워드가 필요없다는 것이다. `Figure()`에서는 'data' 속성과 'layout' 속성을 모두 정의할 수 있기 때문에 'layout' 속성에는 반드시 "layout" 키워드를 붙여주었지만 `update_layout()`은 'layout' 전용 함수이기 때문에 구지 'layout' 키워드를 붙여줄 필요가 없다는 것이다. 다음은 `update_layout()`에서 `dict()`를 사용하여 딕셔너리를 구성하는 python 코드이다.

```
fig_scatter = go.Figure()

fig_scatter.add_trace({
    'type': 'scatter', 'mode': 'markers',
    'x': df_취업률_500['졸업자수'], 'y': df_취업률_500['취업자수'],
    'marker': {'color': 'darkblue'}})

fig_scatter.update_layout(
    ### dict()를 사용한 title 속성의 설정
    title = dict(text = "<b>졸업자 대비 취업자수</b>"},
    x = 0.5, xanchor = 'center', yanchor = 'top'))
```

졸업자 대비 취업자수



실행결과 II-15. python 의 Figure()를 사용한 title 설정

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

다음의 코드는 HTML inline 속성을 사용하여 제목 문자열 스타일을 설정한 R 과 python 코드이다. ‘졸업자 대비 취업률’이라는 문자열의 ‘졸업자’의 크기를 15, 컬러를 ‘red’, 볼드체로, ‘취업자’의 크기를 15, 컬러를 ‘blue’, 볼드체로, ‘대비’는 크기 10 으로 설정하였다.

- R

```
R_layout_scatter |>
  ## title 의 HTML inline 설정
  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;'>취업자</span></span>",
    x = 0.5, xanchor = 'center', yanchor = 'top'))
```

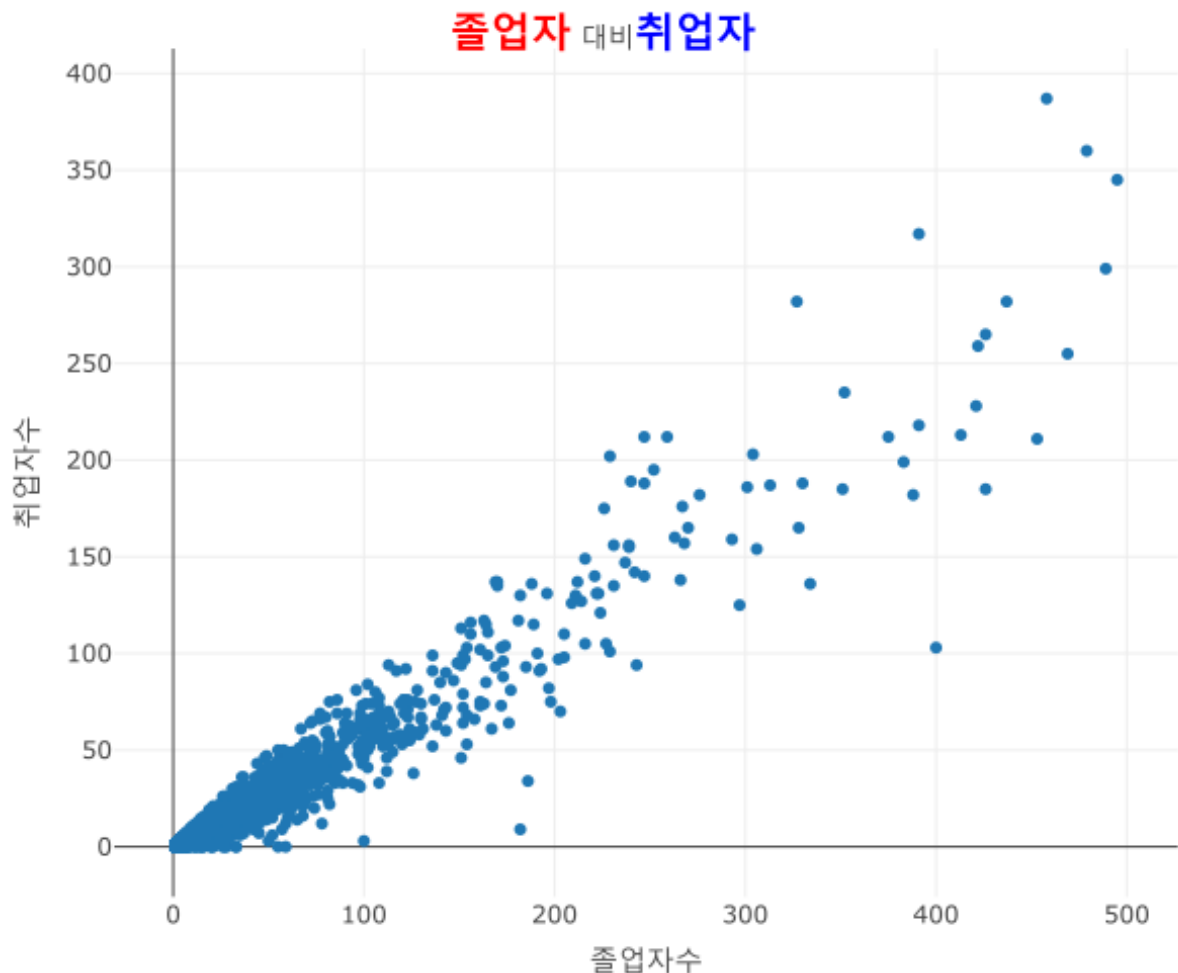
- python

python 도 ”을 사용하는 HTML inline 속성을 사용하여 CSS 의 스타일의 세부 설정이 가능하다. 다음의 python 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)
```

title 의 HTML inline 설정

```
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>",
                                     'x' : 0.5, 'xanchor' : 'center',
                                     'yanchor' : 'top'})
```



실행결과 II-16. R 의 HTML inline title 속성 설정

4.3.2.2. 색 설정

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

속성	속성 설명	속성값
paper_bgcolor	플롯이 그려지는 용지의 배경색을 설정	색상값
plot_bgcolor	x축과 y축 사이의 플로팅 영역의 배경색 설정	색상값

plotly에서 사용하는 색의 설정은 색 이름의 사용, 16 진수로 설정된 RGB 값의 사용, `rgb()` 함수를 사용한 RGB 값의 사용이 주로 사용된다. 이외에도 색조, 채도, 명도(lightness)를 사용하는 `hsl()`을 사용하는 방법, 색조, 채도, 명도(value)를 사용하는 `hsv()`를 사용하는 방법도 있다. plotly에서 사용이 가능한 색 이름은 W3.org에서 제공하는 CSS 색 이름을 사용한다.⁶

다음의 코드는 앞서 그린 산점도의 전체 배경과 플롯 영역 배경 색을 'darkgray'로 바꾸어주는 R과 python 코드이다.

- R

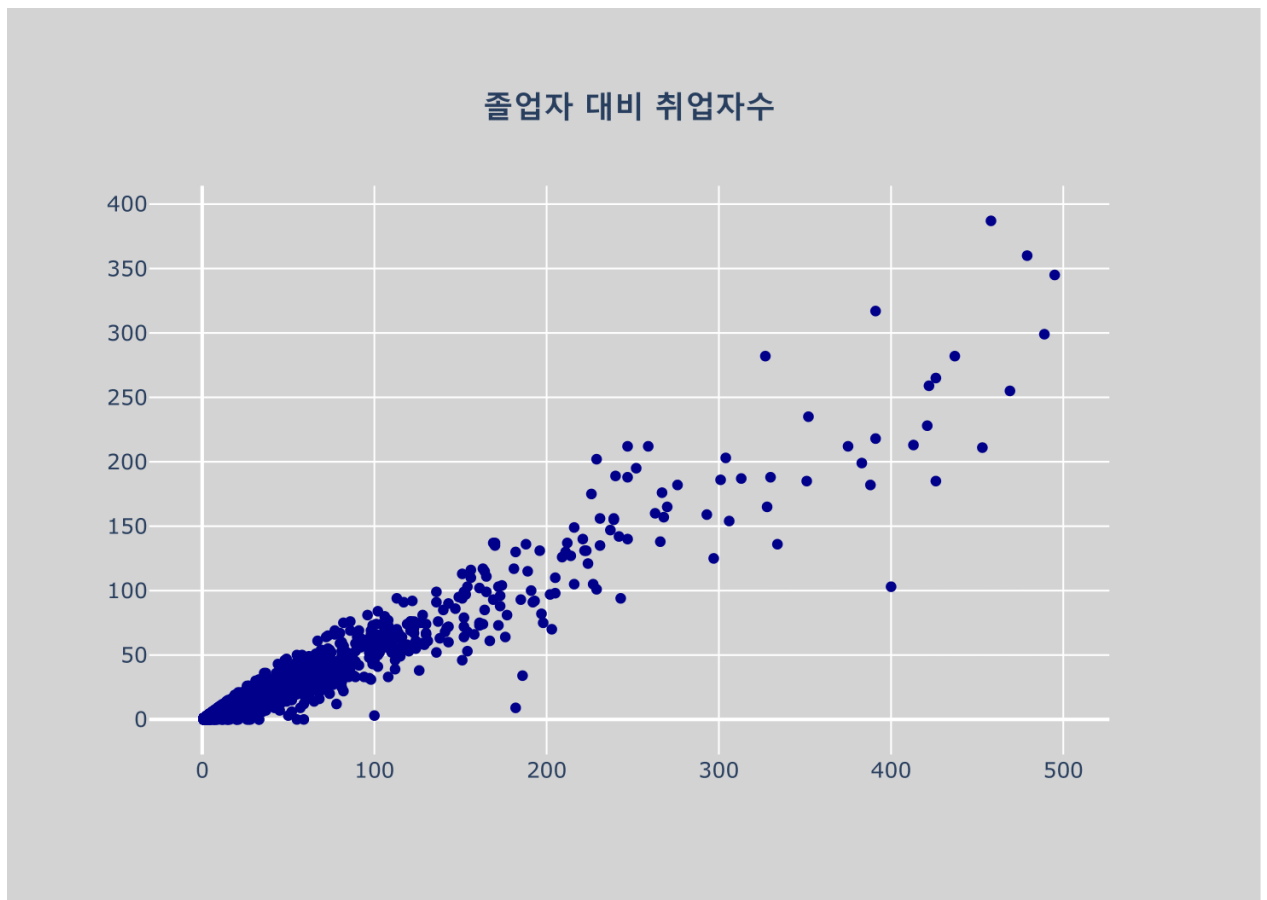
```
R_layout_scatter <- R_layout_scatter |>
  ## 페이지 배경색과 플롯 배경색의 설정
  layout(paper_bgcolor = 'lightgray', plot_bgcolor = 'lightgray')

R_layout_scatter
```

- python

```
## 페이지 배경색과 플롯 배경색의 설정
fig_scatter.update_layout(paper_bgcolor = 'lightgray', plot_bgcolor = 'lightgray')
```


⁶ https://www.w3schools.com/cssref/css_colors.asp



실행결과 II-17. python의 color 속성 설정

4.3.2.3. 축(xaxis, yaxis) 설정

plotly에서 축은 트레이스에 따라 다른 이름으로 불린다. X, Y 축을 사용하는 2 차원의 데카르트 좌표계에서는 'xaxis'와 'yaxis', 3 차원 trace에서는 '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	
xaxis(yaxis)	automargin	주 눈금라벨과 플롯간의 자동 간격 설정	"height", "width", "left", "right", "top", "bottom" 이나 "+"를 사용한 조합 또는 논리값	
	autorange	입력 데이터에 관련한 축의 범위를 설정 여부	{ True False "reversed" }	
	autotypenumbers	strict"는 프레임의 수치 문자열을 수치로 변환하지 않고 문자열로	{ "convert types" "strict" }	
	categoryarray	축에 표현될 이산형 변수의 순서 설정	list, numpy array, or Pandas series of numbers, strings, or datetimes(dataframe column, list, vector)	
	categoryorder	이산형 변수의 순서 설정 방법을 설정	{ "trace" "category ascending" "category descending" "array" "total ascending" "total descending" "min ascending" "min descending" "max ascending" "max descending" "sum ascending" "sum descending" "mean ascending" "mean descending" "median ascending" "median descending" }	
	color	축 라인, 폰트, 눈금, 그리드 등 축에 관련한 전반적 색상 설정	색상값	
	domain	전체 플롯에서의 비율로 변환된 축의 범위 설정	list	
	dtick	축에서의 눈금 간격 설정	수치나 좌표상의 변형값	
	fixedrange	축 범위를 고정, 고정값을 사용하면 값이 되지 않음	논리값	
	gridcolor	그리드  색상 설정	색상값	
	gridwidth	그리드 라인의 두께 설정	0이상의 수치	
	linecolor	축 선의 색상 설정	색상값	
	linewidth	축 선의 두께 설정	0이상의 수치	
	minor	dtick	보조 눈금의 간격 설정	수치나 좌표상의 변형값
		gridcolor	보조 눈금의 그리드 색상 설정	색상값
		gridwidth	보조 눈금의 그리드 두께 설정	0이상의 수치
		nticks	보조 눈금의 총 개수 설정	정수
		showgrid	보조 눈금의 그리드 표시 여부 설정	논리값
		tick0	보조 눈금의 시작값 설정	수치나 좌표상의 변형값
		tickcolor	보조 눈금의 눈금 색상 설정	색상값
		ticklen	보조 눈금의 눈금 길이 설정	0이상의 수치
		tickmode	보조 눈금의 눈금 모드 설정	{ "auto" "linear" "array" }
		ticks	보조 눈금의 표시 위치 설정	{ "outside" "inside" "" }
		tickvals	보조 눈금의 표시값 설정	list, numpy array, or Pandas series of numbers, strings, or datetimes(dataframe column, list, vector)
		tickwidth	보조 눈금의 눈금 두께 설정	0이상의 수치
	mirror	플롯팅 영역의 반대편에 축선과 축 눈금을 미러링 할지 여부 설정	{ True "ticks" False "all" "allticks" }	
	nticks	축에 표시할 눈금의 최대 개수 설정	0이상의 정수	
	position	플롯팅 공간에서의 축의 위치 설정	0부터 1사이의 수치	
	range	축의 범위 설정	list	
	rangebreaks	bounds	축의 단일 구간의 상한과 하한값 설정	list
		dvalue	values 배열의 크기 설정(밀리세컨 단위)	0이상의 수치
		enabled	축 단절을 사용할지 여부 설정	논리값
		pattern	절단 구간의 패턴 설정	{ "day of week" "hour" "" }
		values	축 단절과 관련한 축상의 값 설정	list
	rangemode	범위 모드의 설정	{ "normal" "tozero" "nonnegative" }	
	separatethousands	천단위 구분자 사용 여부 설정	논리값	
	showdividers	이산형 변수 간의 나눗선 사용 여부 설정	논리값	
showexponent	지수형 수치 표시 여부 설정	{ "all" "first" "last" "none" }		
showgrid	그리드 라인의 표시 여부 설정	논리값		
showline	축 선의 표시 여부 설정	논리값		
showspikes	스파이크 선의 표시 여부 설정	논리값		
showticklabels	눈금 라벨 표시 여부 설정	논리값		

4.3.2.3.1. 축 제목, 원점 선, 그리드의 설정

축의 제목을 설정하기 위해서는 'xaxis', 'yaxis'의 하위 속성인 'title'을 사용하여 설정한다. 플롯의 전체 제목 설정에 사용했던 'title'과 동일한 하위 속성을 가진다. 또 전체 제목 설정과 같이 'title'은 리프 속성 노드으로도 사용될 수 있고 하위 속성을 담는 컨테이너 속성으로도 사용될 수 있다.

2 차원 좌표계를 사용하는 시각화에서는 가급적 원점부터 데이터를 표시하여 데이터의 왜곡을 줄이는 것이 좋다고 알려져 있다. 이렇게 원점부터 데이터를 표시할 때 원점을 지나는 X, Y 축의 선을 강조할 필요가 있다면 'zerolinecolor', 'zerolinewidth'으로 설정이 가능하다.

2 차원 좌표계에서 수평선과 수직선으로 축의 데이터를 표현하는 보조선을 그리드라고 한다. plotly 에서 그리드를 설정하는 속성이 'gridcolor', 'gridwidth'를 사용하여 설정이 가능하다.

다음의 코드는 축 제목, 원점 선, 그리드를 설정하는 R 과 python 의 코드이다. X 축의 설정에서 'title' 설정은 하위 속성을 설정하는 `list()`와 `dict()`를 사용하여 설정하였고 Y 축의 설정에서 'title'의 설정은 리프 속성 노드로 설정하는 방법을 사용하였다. 축 제목은 HTML 태그를 이용하여 볼드체와 아래 첨자를 설정하였다. 또 'xaxis'와 'yaxis'의 첫 번째 레벨로써 'color'를 설정하면 line, font, tick, and grid 색상의 기본값을 설정하게 된다. 이 외에 원점 선 색('zerolinecolor')을 'black', 원점 선 두께('zerowidth')를 3, 그리드 색('gridcolor')을 'gray', 그리드 두께('gridwidth')를 1 로 설정하였다.

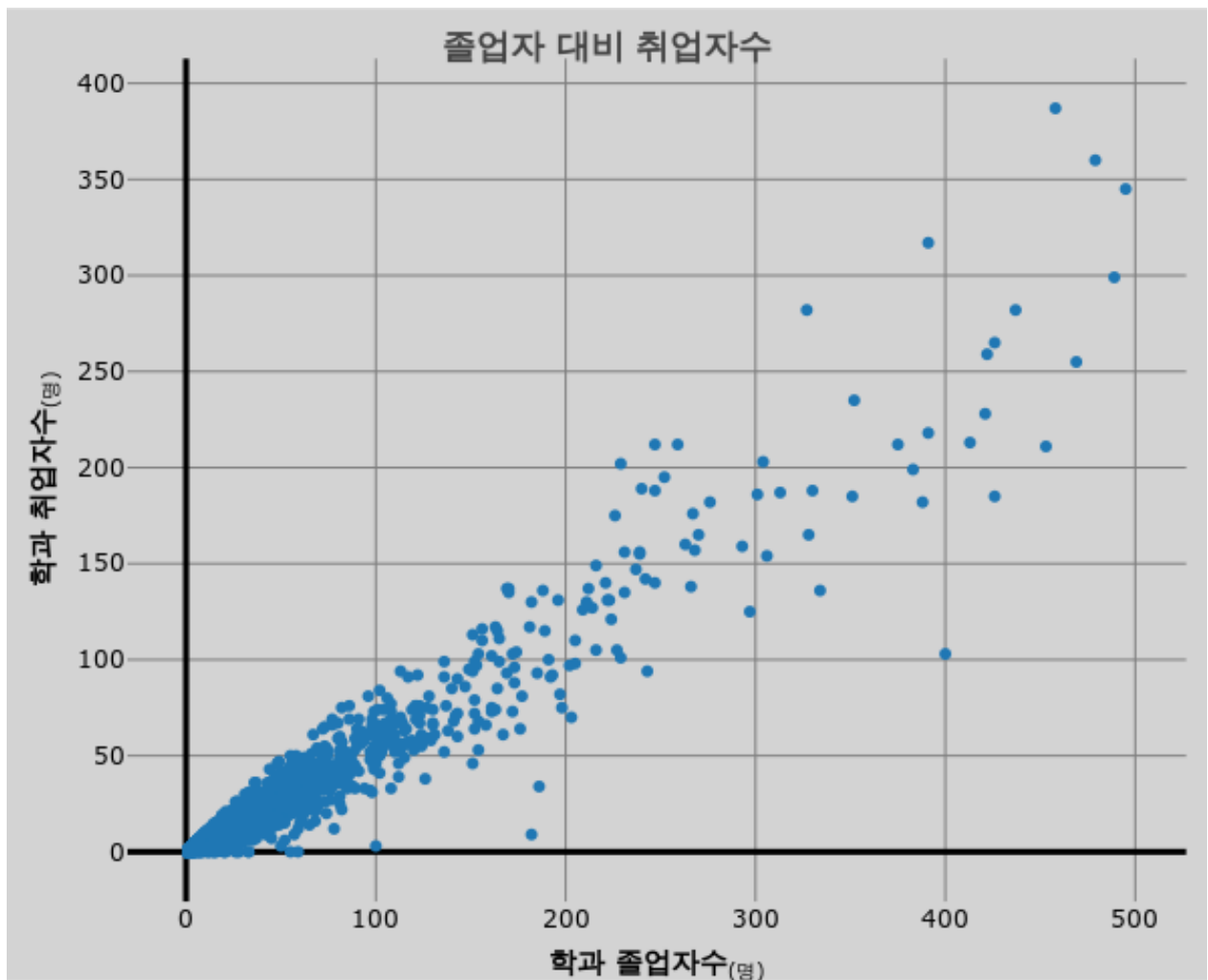
- R

```
R_layout_scatter <- R_layout_scatter |>
  layout(xaxis = list(
    title = list(text = '<b>학과 졸업자수</b><sub>(명)</sub>'), ## 정상적 방법
    color = 'black', zerolinecolor = 'black', zerolinewidth = 3,
    gridcolor = 'gray', gridwidth = 1),
  yaxis = list(
    title = '<b>학과 취업자수</b><sub>(명)</sub>',
    color = 'black', zerolinecolor = 'black', zerolinewidth = 3,
    gridcolor = 'gray', gridwidth = 1) ## 약식 방법
  )

R_layout_scatter
```

- python

```
fig_scatter.update_layout(
  xaxis = dict(
    title = dict(text = '<b>학과 졸업자수</b><sub>(명)</sub>'), ## 정상적 방법
    color = 'black', zerolinecolor = 'black', zerolinewidth = 3,
    gridcolor = 'grey', gridwidth = 1),
  yaxis = dict(
    title = '<b>학과 취업자수</b><sub>(명)</sub>', ## 약식 방법
    color = 'black', zerolinecolor = 'black', zerolinewidth = 3,
    gridcolor = 'grey', gridwidth = 1))
```



실행결과 II-18. R의 axis title, zeroline, grid 속성 설정

4.3.2.3.2. 눈금 라벨, 눈금 간격 설정

축의 설정에서 매우 많이 사용되는 설정이 바로 눈금에 대한 설정이다. 특히 눈금 라벨과 눈금 간격을 어떻게 설정하는가에 따라 해당 시각화에서 제공하는 정보가 매우 달라진다.

plotly에서 눈금 라벨과 눈금 간격을 설정하기 위해 'tickmode', 'tick0', 'nticks', 'dtick', 'ticktext', 'tickvals' 속성을 이용해 설정할 수 있다. 이 속성들은 모두 'xaxis'와 'yaxis'의 첫 번째 레벨 속성 노드이기 때문에 바로 속성값을 설정한다.

'tickmode'는 눈금이 표시되는 방법을 설정한다. 'tickmode'는 "auto", "linear", "array"의 세 가지 속성값을 가진다. "auto"는 'nticks'의 속성값으로 설정된 개수만큼 자동적으로 눈금이 표시된다. "linear"는 'tick0'에서부터 'dtick'만큼의 간격으로 눈금이 표시된다.

"array"는 'tickvals'와 'ticktext'에 설정된 눈금 배열만큼 눈금이 표시된다.

다음은 'xaxis'를 "array" 모드로 'ticktext'와 'tickvals'를 사용해 눈금 라벨과 눈금 간격을 설정하였다. 눈금에 사용하는 라벨은 숫자 대신 숫자가 의미하는 텍스트를 사용해 사용자에게 추가적 정보를 줄 수 있도록 설정하였다. 'yaxis'를 "linear" 모드로 'tick0'와 'dtick'을 사용하여 설정하는 R 과 python 코드이다.

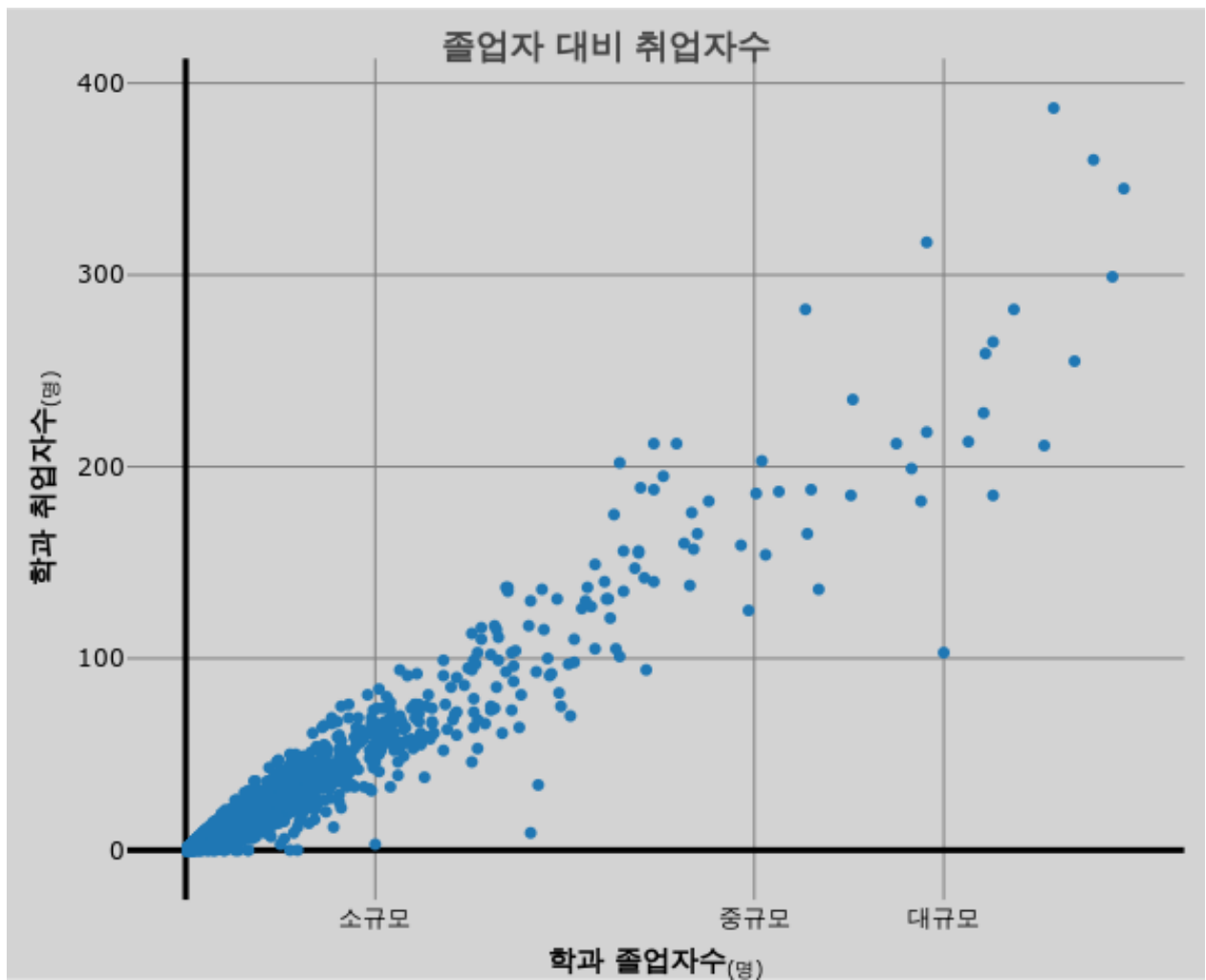
- R

```
R_layout_scatter <- R_layout_scatter |>
  layout(xaxis = list(
    tickmode = 'array', ## tickmode 를 "array"로 설정
    ticktext = c('소규모', '중규모', '대규모'), ## ticktext 설정
    tickvals = c(100, 300, 400)), ## tickvals 설정
  yaxis = list(
    tickmode = 'linear', ## tickmode 를 "linear"로 설정
    tick0 = 100, ## tick0 설정
    dtick = 100)) ## dtick 설정
```

R_layout_scatter

- python

```
fig_scatter_temp.update_layout(
  xaxis = dict(tickmode = 'array', ## tickmode 를 "array"로 설정
    ticktext = ('소규모', '중규모', '대규모'), ## ticktext 설정
    tickvals = (100, 300, 400) ## tickvals 설정
  ),
  yaxis = dict(tickmode = 'linear', ## tickmode 를 "linear"로 설정
    tick0 = 100, ## tick0 설정
    dtick = 100) ## dtick 설정
)
```



실행결과 II-19. python의 눈금라벨, 눈금 간격 속성 설정

4.3.2.3.3. 축 범위 설정

plotly에서는 트레이스에 할당된 데이터들이 다 표현되도록 각각의 축의 범위를 자동적으로 설정한다. 하지만 시각화를 하다보면 축의 일부를 강조하거나 확대하기 위해 전체 축의 범위중에 일부의 범위에 한정하여 데이터를 표현해야할 경우가 있다. 이렇게 축의 범위를 한정하거나 설정할 때 사용하는 속성이 'range'와 'rangemode'이다.

또 plotly는 'range'에 관련된 'rangeslider'나 'rangeselector'와 같은 축 범위 설정과 관련한 컨트롤을 제공하는데 이는 '시간의 시각화'에서 설명하도록 한다.

'range'는 표현하고자 하는 축의 최소값과 최대값으로 구성된 배열이나 벡터를 사용하여 축의 범위를 설정해주는 속성이다. 'range' 속성을 사용하여 Zoom In 이나 Zoom Out 과 유사한 효과를 낼 수 있다.

'rangemode'는 축 범위의 효과를 설정하는 속성으로 "normal", "tozero", "nonnegative"의 세 가지 속성값을 가진다. "normal"은 plotly의 기본값으로 데이터가 표현되는 전체를 축의 범위로 설정한다. "tozero"는 축의 시작점을 0부터 시작하도록 설정하고, "nonnegative"는 축에서 음수의 범위를 표시하지 않도록 설정한다.

다음은 X 축과 Y 축의 'range'와 'rangemode'를 설정하는 R과 python의 코드이다. X 축은 0부터 350까지, Y 축은 0부터 300까지로 설정하고 X 축의 'rangemode'는 "nonnegative"로 Y 축의 'rangemode'는 "tozero"로 설정하였다. 만약 'range'는 음수부터 시작하고 'rangemode'는 "nonnegative"인 경우와 같이 'range'와 "rangemode"의 설정이 서로 배치되는 경우에는 'range' 속성이 우선한다. 이렇게 설정하면 눈금 라벨과 축과의 거리가 너무 가까워지기 때문에 'margin' 속성의 'pad' 속성을 사용하여 약간의 여백을 주었다.

- R

R에서 'range' 속성을 설정하기 위해 `c()`를 사용하여 최소값과 최대값으로 구성된 벡터를 설정하였다.

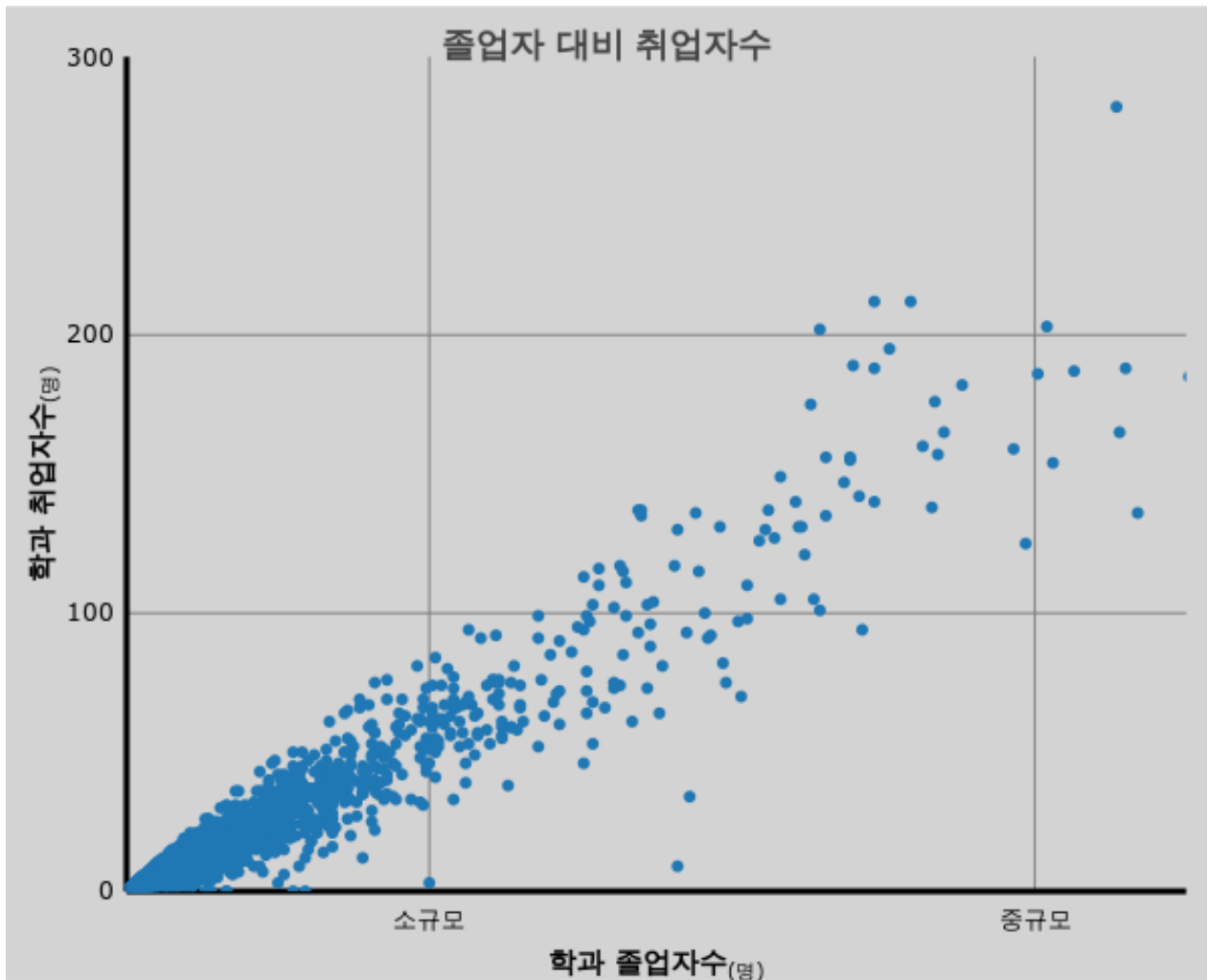
```
R_layout_scatter |>
  layout(xaxis = list(range = c(0, 350), ## X 축의 range 설정
                     rangemode = 'nonnegative'), ## X 축의 rangemode 설정
        yaxis = list(range = c(0, 300), ## Y 축의 range 설정
                     rangemode = 'tozero'), ## Y 축의 rangemode 설정
        margin = list(pad = 5))
```

- python

python에서 'range' 속성을 설정하기 위해서는 배열이나 튜플을 사용하여 최소값과 최대값으로 구성된 벡터를 설정한다. 배열을 사용한다면 `[]`로 최소값과 최대값을 묶고 튜플을 사용한다면 `()`로 최소값과 최대값을 묶어준다.

```
fig_scatter_temp = go.Figure(fig_scatter)
fig_scatter_temp.update_layout(
    xaxis = dict(range = (0, 350), ## 배열을 사용한 X 축의 range 설정
                 rangemode = 'nonnegative'), ## X 축의 rangemode 설정
    yaxis = dict(range = [0, 300], ## 튜플을 사용한 Y 축의 range 설정
                 rangemode = 'tozero'), ## Y 축의 rangemode 설정
```

```
margin = dict(pad = 5)
)
```



실행결과 II-20. R의 축 범위 속성 설정

4.3.2.4. 범례(legend) 설정

`layout()`에서 범례 설정과 관련된 속성은 'showlegend'와 'legend'뿐 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 하위 속성의 컨테이너 속성이다. '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	0이상의 수치	
	entrywidth		범례의 두께 설정	0이상의 수치	
	font	color	범례 문자색 설정	색상값	
		family	범례 문자 폰트 설정	폰트명	
		size	범례 문자 크기 설정	1이상의 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	1이상의 수치
		side	범례 제목의 위치 설정	"top" "left" "top left"	
		text	범례 제목 문자열 설정	문자열	
	traceorder		범례 순서 설정	"reversed", "grouped", "reversed+grouped", "normal"	
	valign		텍스트에 연관된 심볼의 수직 정렬 설정	"top" "middle" "bottom"	
	x		범례의 X축 위치	-2에서 3까지의 수치	
	xanchor		범례의 수평 위치 앵커를 설정	"auto" "left" "center" "right"	
y		범례의 X축 위치	-2에서 3까지의 수치		
yanchor		범례의 수평 위치 앵커를 설정	"auto" "top" "middle" "bottom"		

다음의 코로나 19의 확진자 수를 넓은 형태의 데이터프레임에서 선형 차트로 그리는 R과 python 코드이다. 각각의 대륙을 `add_trace()`로 추가하여 총 4개의 트레이스를 추가하였고 이중 아시아 트레이스에는 'showlegend'를 "FALSE"로 설정하여 아시아 트레이스의 범례를 제거하였다. 또 'layout'에서 범례의 방향을 가로방향, 범례의 테두리 색을 회색, 테두리 두께를 2, x와 y의 위치를 0.95, xanchor를 "right"로 설정했다.

- R

초기화 및 한국 확진자 선 scatter 트레이스 생성

```
R_layout_line <- df_covid19_100_wide |>
plot_ly() |>
add_trace(type = 'scatter', mode = 'lines',
          x = ~date, y = ~확진자_한국, name = '한국')
```

아시아 확진자 선 scatter 트레이스 추가

```
R_layout_line <- R_layout_line |>
```

```
add_trace(type = 'scatter', mode = 'lines',
          x = ~date, y = ~확진자_아시아, name = '아시아', showlegend = FALSE)
```

유럽 확진자 선 scatter 트레이스 추가

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

북미 확진자 선 scatter 트레이스 추가

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

범례 layout 설정

```
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

초기화

```
fig_line = go.Figure()
```

한국 확진자 선 scatter 트레이스 생성

```
fig_line.add_trace({
  'type' : 'scatter', 'mode' : 'lines',
  'x' : df_covid19_100_wide.index, 'y' : df_covid19_100_wide['확진자_한국'],
  'name' : '한국'
})
```


아시아 확진자 선 scatter 트레이스 추가

```
fig_line.add_trace({  
    'type' : 'scatter', 'mode' : 'lines',  
    'x' : df_covid19_100_wide.index, 'y' : df_covid19_100_wide['확진자_아시아'],  
    'name' : '아시아'  
})
```

유럽 확진자 선 scatter 트레이스 추가

```
fig_line.add_trace({  
    'type' : 'scatter', 'mode' : 'lines',  
    'x' : df_covid19_100_wide.index, 'y' : df_covid19_100_wide['확진자_유럽'],  
    'name' : '유럽'  
})
```

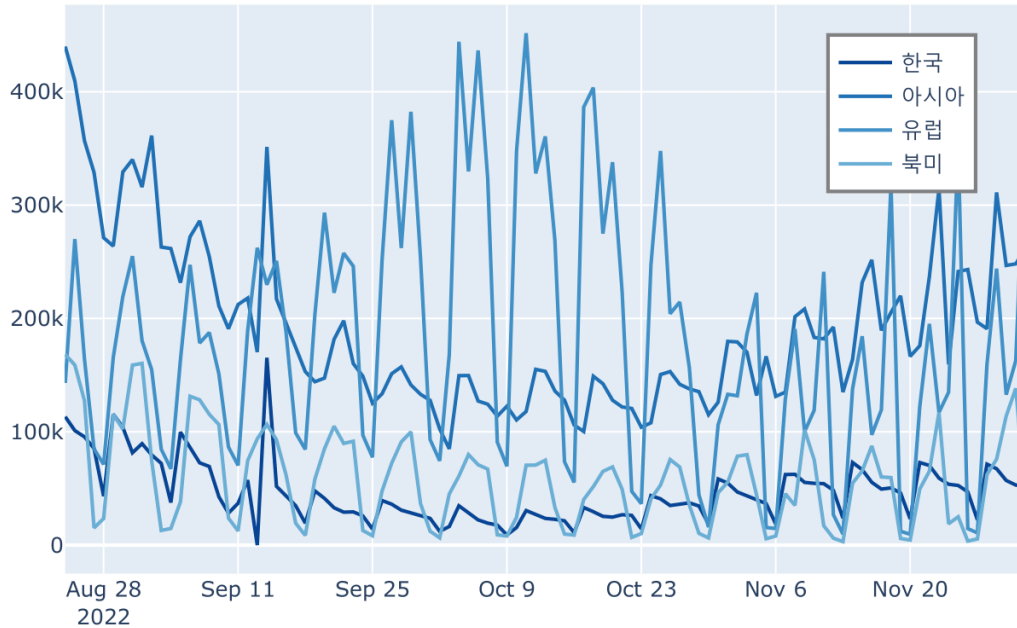
북미 확진자 선 scatter 트레이스 추가

```
fig_line.add_trace({  
    'type' : 'scatter', 'mode' : 'lines',  
    'x' : df_covid19_100_wide.index, 'y' : df_covid19_100_wide['확진자_북미'],  
    'name' : '북미'  
})
```

범례 layout 설정

```
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



실행결과 II-21. python 의 범례 설정

4.3.2.5. 여백(margin) 설정

'layout'의 하위 속성 중 여백의 설정과 관련된 속성은 'margin'이다. 'margin'은 컨테이너 속성으로 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	0이상의 수치
	l	왼쪽 여백 설정	0이상의 수치
	r	오른쪽 여백 설정	0이상의 수치
	t	위쪽 여백 설정	0이상의 수치

다음은 여백의 설정을 위한 R 과 python 코드이다.

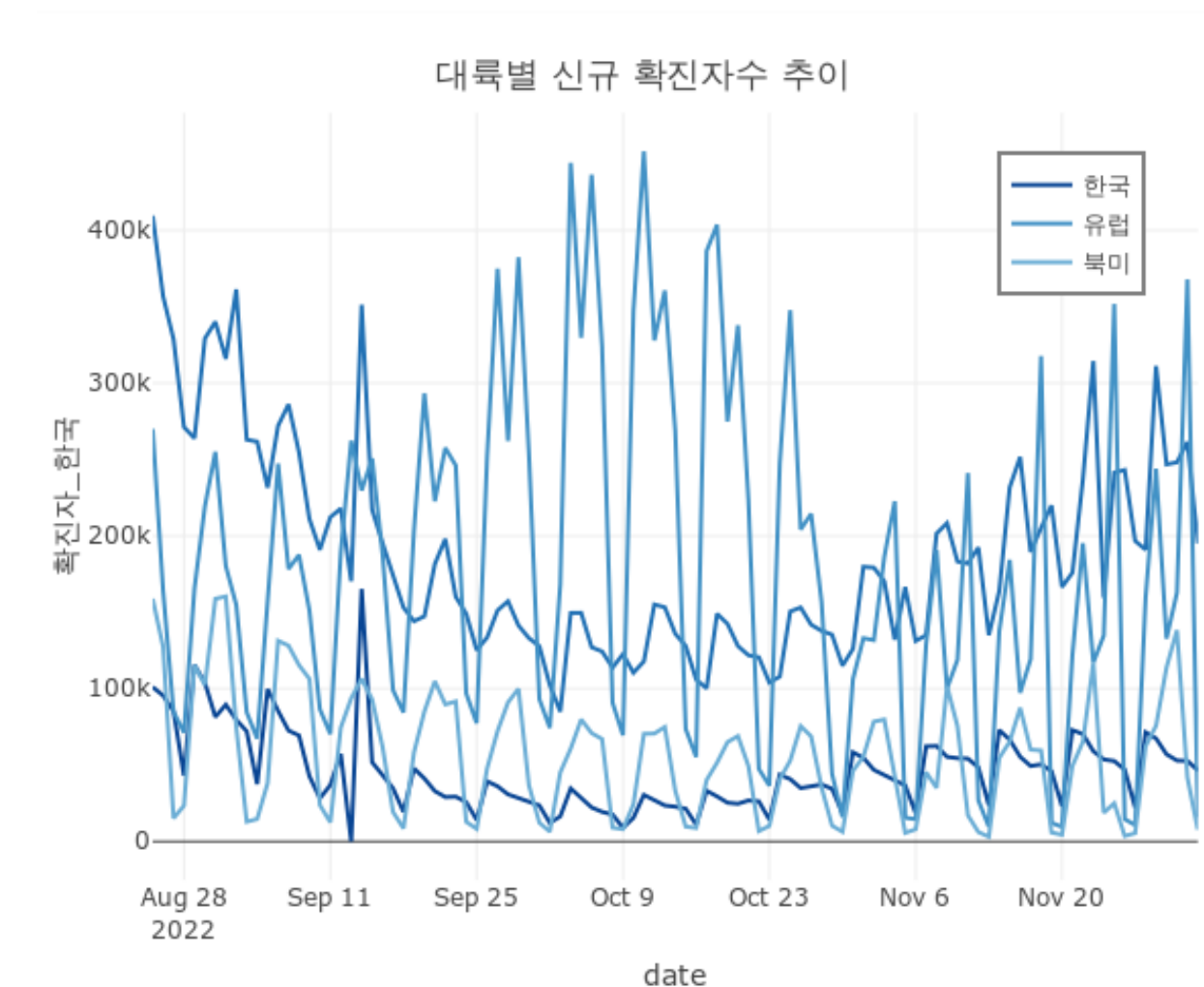
- R

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

R_layout_line

- python

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



실행결과 II-22. R의 여백 설정

4.3.2.6. 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	10이상의 수치
height	플롯의 높이 설정	10이상의 수치

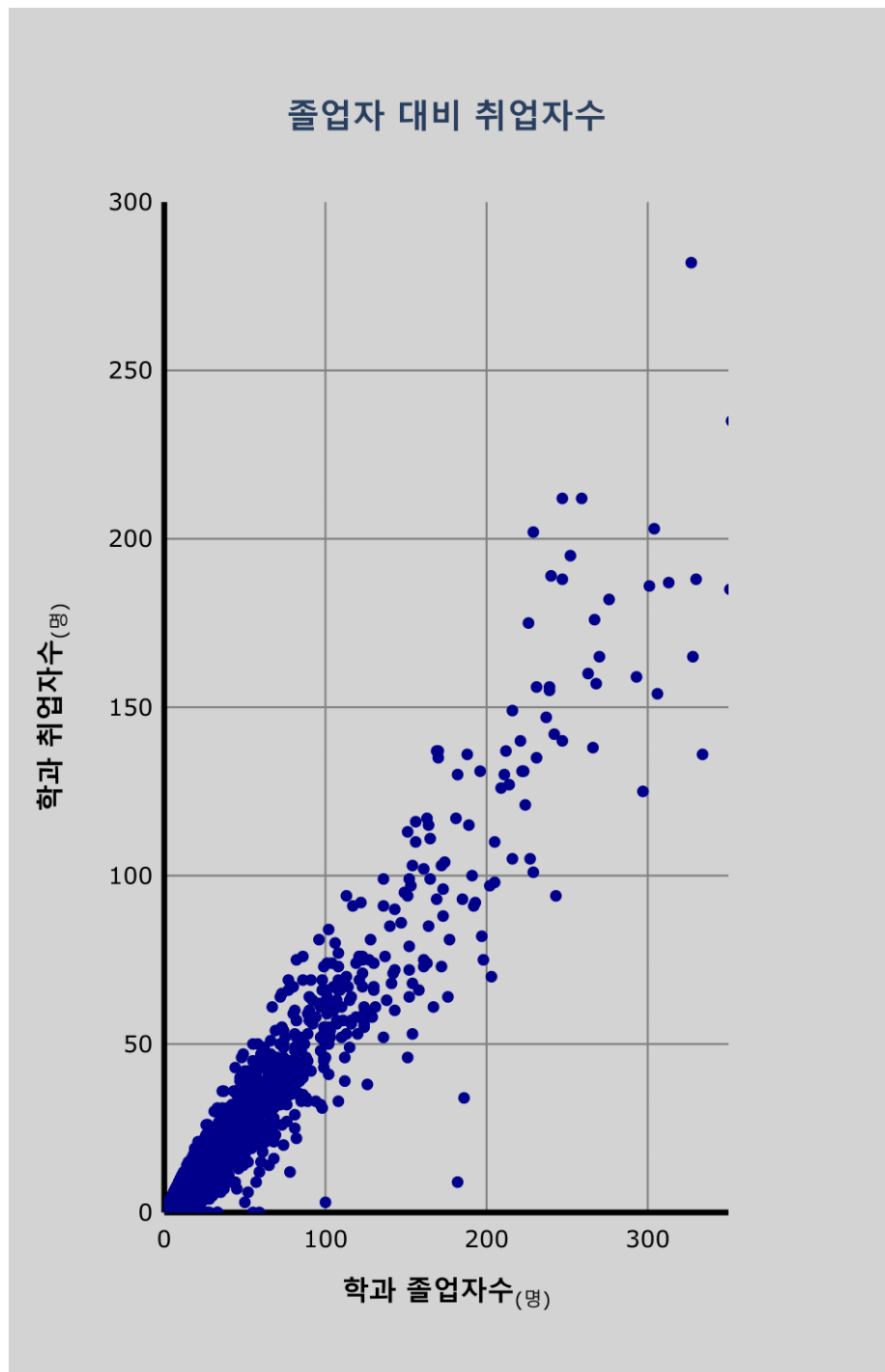
다음은 세로로 긴 플롯 사이즈를 설정하는 R 과 python 코드이다.

- R

```
R_layout_scatter |>  
## 플롯 사이즈 설정  
layout(width = 450, height = 700)
```

- python

```
## 플롯 사이즈 설정  
fig_scatter_temp.update_layout(width = 450, height = 700)
```



실행결과 II-23. python 의 플롯 크기 속성 설정

4.3.2.7. 폰트 설정

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

공통적으로 사용되는 속성이다. 하지만 'layout'의 첫 레벨로써 'font' 설정은 plotly 전체적인 폰트를 설정하게 된다.

다음은 플롯 전체에 적용되는 폰트 설정에 대한 R 과 python 코드이다.

- R

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

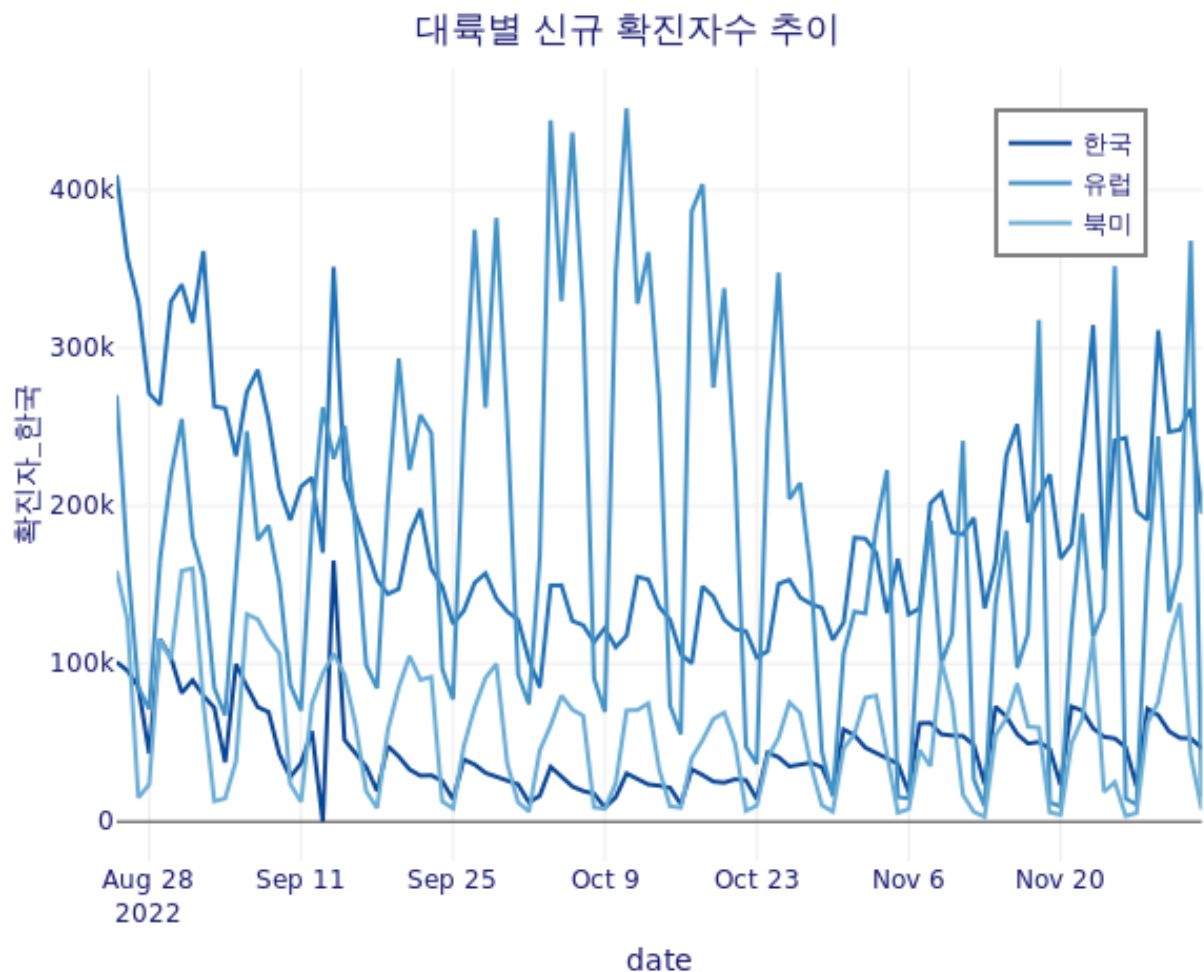
```
R_layout_line <- R_layout_line |>
  ## 폰트 설정
  layout(font = list(family = "나눔고딕", color = 'MidnightBlue', size = 12))

R_layout_line
```

- python

python 에서 문자열에 대한 폰트 설정 코드는 다음과 같다. 폰트명은 OS 에 설치된 폰트명을 사용하고 색상과 크기를 설정하였다.

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



실행결과 II-24. R의 폰트 설정

5. 서브 플롯

데이터를 시각화할 때 하나의 시각화에 여러 개의 데이터를 표시하는 경우는 매우 빈번하게 발생한다. 보통 이런 경우 각각의 데이터를 구별하기 위해 색상, 크기, 라인타입 등의 다양한 그래픽적 특성을 사용하여 데이터를 구분한다. 하지만 구분되어야 하는 데이터가 많은 경우는 데이터 구분 속성을 알아보기 힘들거나 일부 구간에 중복되어 인식하기 어려운 경우도 많다. 이런 경우 각각의 데이터들을 따로 떼서 작은 시각화를 만들어 주면 이런 현상을 해소할 수 있다.

plotly에서도 이렇게 여러 개의 작은 플롯을 만드는 기능을 서브 플롯이라는 이름으로 지원한다. 서브 플롯은 동일한 트레이스를 사용하는 플롯으로 구성할 수도 있고 서브 플롯마다 각자의 트레이스를 사용해서 서로 다른 여러개의 트레이스로 구성할 수도 있다.

5.1. 서브 플롯 생성과 제목 설정

plotly 에서 서브 플롯을 생성하기 위해서 사용하는 함수는 R 에서는 `subplot()`, python 에서는 `plotly.subplots` 모듈의 `make_subplots()`이다. 앞서 'data'와 'layout'의 경우 R 과 python 이 유사한 방법을 사용했지만 서브 플롯에서는 두 언어간 구현 방식이 다소 다르다.

서브 플롯의 사용시에 하나 주의해야할 부분은 서브 플롯은 서브 플롯을 담는 전체 플롯과 서브 플롯 들로 구성된다는 것이다. 'layout'의 속성을 사용하여 플롯의 'layout' 속성을 설정하면 이 속성은 전체 플롯에도, 서브 플롯에도 적용될 수 있는데 이 때는 전체 플롯에 우선 적용되고 서브 플롯에는 적용되지 않는다. 예를 들어 플롯의 제목을 설정하는 'title'의 경우 세부 서브 플롯들의 제목과 전체 플롯의 제목이 모두 'layout'의 'title' 속성으로 설정된다. 하지만 서브 플롯에 'layout'의 'title'로 세부 설정을 한다고 해도 결국은 전체 제목으로 설정된다는 것이다. 하지만 'xaxis'나 'yaxis'와 같이 전체 플롯에는 없는 설정의 경우는 각각의 세부 플롯에서 설정된 값들이 유지된다.

- R

R 에서 plotly 가 서브 플롯을 구현되는 방식은 먼저 서브 플롯에 사용될 각각의 plotly 객체를 생성하고 이 객체들을 `subplot()`으로 전체 플롯을 구성한다. `subplot()`에서는 plotly 객체 뿐 아니라 `ggplot2` 로 만들어진 객체도 서브 플롯으로 구성할 수 있다.

또 `subplot()`에서 사용되는 매개변수들은 앞서 `add_trace()`나 `layout()`과 같이 속성들의 리스트가 아닌 함수의 매개변수 형태로 사용한다. R 에서 서브 플롯을 설정할 때 가장 중요한 것이 'nrows' 이다. 'nrows'는 서브 플롯의 행의 개수를 설정하는 매개변수인데 'nrows'가 설정되면 전체 서브 플롯의 수에 따라 자동적으로 열의 수가 결정된다.

`subplot()`의 주요 매개변수는 다음과 같다.

`subplot(..., nrows = 1, widths = NULL, heights = NULL, margin = 0.02, shareX = FALSE, shareY = FALSE, titleX = shareX, titleY = shareY, which_layout = "merge")`

- ... : plotly 나 ggplot2 객체의 이름, 리스트, tibble
- nrows : subplot 의 행의 개수
- widths : 각각의 subplot 의 0 부터 1 사이의 상대적 열 너비
- heights : 각각의 subplot 의 0 부터 1 사이의 상대적 열 높이
- margin : 0 부터 1 사이의 단일 수치나 4 개의 수치(왼쪽, 오른쪽, 위, 아래)의 여백

- shareX : X 축을 공유할지에 대한 논리값
- shareY : Y 축을 공유할지에 대한 논리값
- titleX : X 축의 제목을 남길지에 대한 논리값
- titleY : Y 축의 제목을 남길지에 대한 논리값
- which_layout : 어떤 플롯의 레이아웃에 적용할지 결정, 기본값은 “merge”로 뒤 플롯의 레이아웃 옵션이 앞 옵션보다 우선함을 의미

다음은 대륙별 확진자수의 선 그래프를 서브 플롯으로 구성하는 코드이다. 먼저 전체적인 plotly 객체를 초기화하고 이 초기화된 객체를 사용하여 한국, 아시아, 유럽, 북미, 남미, 오세아니아, 아프리카의 plotly 객체를 각각 생성한다. 이 객체들을 `subplot()`을 사용하여 서브 플롯을 만들어주고 서브 플롯의 'layout'을 설정하였다.

서브플롯 생성을 위한 기본 plotly 객체 생성

```
p_line_wide <- df_covid19_100_wide |> plot_ly()
```

첫 번째 서브 플롯 생성

```
p1 <- p_line_wide |>
  add_trace(type = 'scatter', mode = 'lines', x = ~date,
            y = ~확진자_한국, name = '한국') |>
  layout(xaxis = list(tickfont = list(size = 10)),
         yaxis = list(title = list(text = '확진자수')))
```

두 번째 서브 플롯 생성

```
p2 <- p_line_wide |>
  add_trace(type = 'scatter', mode = 'lines', x = ~date,
            y = ~확진자_아시아, name = '아시아') |>
  layout(xaxis = list(tickfont = list(size = 10)),
         yaxis = list(title = list(text = '확진자수')))
```

세 번째 서브 플롯 생성

```
p3 <- p_line_wide |>
  add_trace(type = 'scatter', mode = 'lines', x = ~date,
            y = ~확진자_유럽, name = '유럽') |>
```

```
layout(xaxis = list(tickfont = list(size = 10)),  
       yaxis = list(title = list(text = '확진자수')))
```

네 번째 서브 플롯 생성

```
p4 <- p_line_wide |>  
  add_trace(type = 'scatter', mode = 'lines', x = ~date,  
            y = ~확진자_북미, name = '북미') |>  
  layout(xaxis = list(tickfont = list(size = 10)),  
        yaxis = list(title = list(text = '확진자수')))
```

다섯 번째 서브 플롯 생성

```
p5 <- p_line_wide |>  
  add_trace(type = 'scatter', mode = 'lines', x = ~date,  
            y = ~확진자_남미, name = '남미') |>  
  layout(xaxis = list(tickfont = list(size = 10)),  
        yaxis = list(title = list(text = '확진자수')))
```

여섯 번째 서브 플롯 생성

```
p6 <- p_line_wide |>  
  add_trace(type = 'scatter', mode = 'lines', x = ~date,  
            y = ~확진자_아프리카, name = '아프리카') |>  
  layout(xaxis = list(tickfont = list(size = 10)),  
        yaxis = list(title = list(text = '확진자수')))
```

일곱 번째 서브 플롯 생성

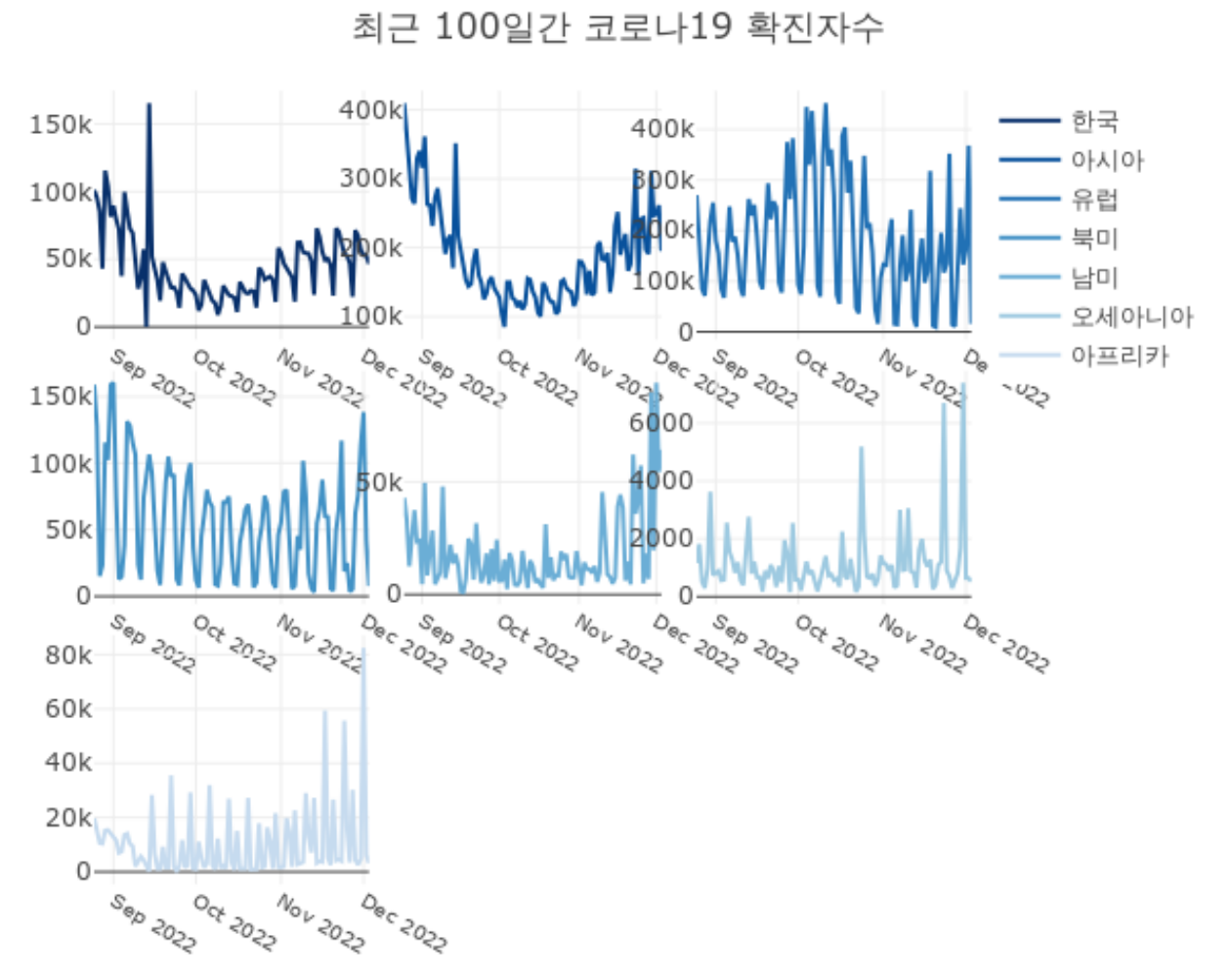
```
p7 <- p_line_wide |>  
  add_trace(type = 'scatter', mode = 'lines', x = ~date,  
            y = ~확진자_오세아니아, name = '오세아니아') |>  
  layout(xaxis = list(tickfont = list(size = 10)),  
        yaxis = list(title = list(text = '확진자수')))
```

```
subplots <- subplot(p1, p2, p3, p4, p5, p6, p7, nrow = 3) |>
```

```
layout(## 전체 제목 설정  
       title = '최근 100 일간 코로나 19 확진자수',  
       ## 전체 여백 설정
```

```
margin = margins_R)
```

subplots



실행결과 II-25. R의 서브플롯 설정

앞의 `subplot()`에서의 주의깊게 보아야하는 것은 제목의 설정이다. 앞의 코드에서 각각의 서브 플롯을 만들때 'layout'의 'title' 속성을 사용하여 제목을 설정하였다. 하지만 결과 서브 플롯에서는 세부 서브 플롯들의 제목들이 반영되지 않는다. 앞서 설명했듯이 'title'과 같이 전체 플롯과 서브 플롯에 공통적으로 적용되는 속성들은 전체 플롯에 우선되기 때문에 이 코드에서는 결국 전체 플롯의 제목이 '한국'에서 '아시아'로, '아시아'에서 '유럽'으로 바뀌어 최종적으로 마지막 'layout'의 'title'인 '최근 100 일간 코로나 19 확진자수'로 설정된다. 만약 마지막 'layout'에서 'title' 속성이 설정되지 않는다면 마지막 'title' 속성인 '아프리카'로 제목이

설정된다. 제대로 된 서브 플롯의 제목을 설정하기 위해서는 `add_annotation()`을 사용하여 주석(annotation)을 제목처럼 붙여주어야 한다.

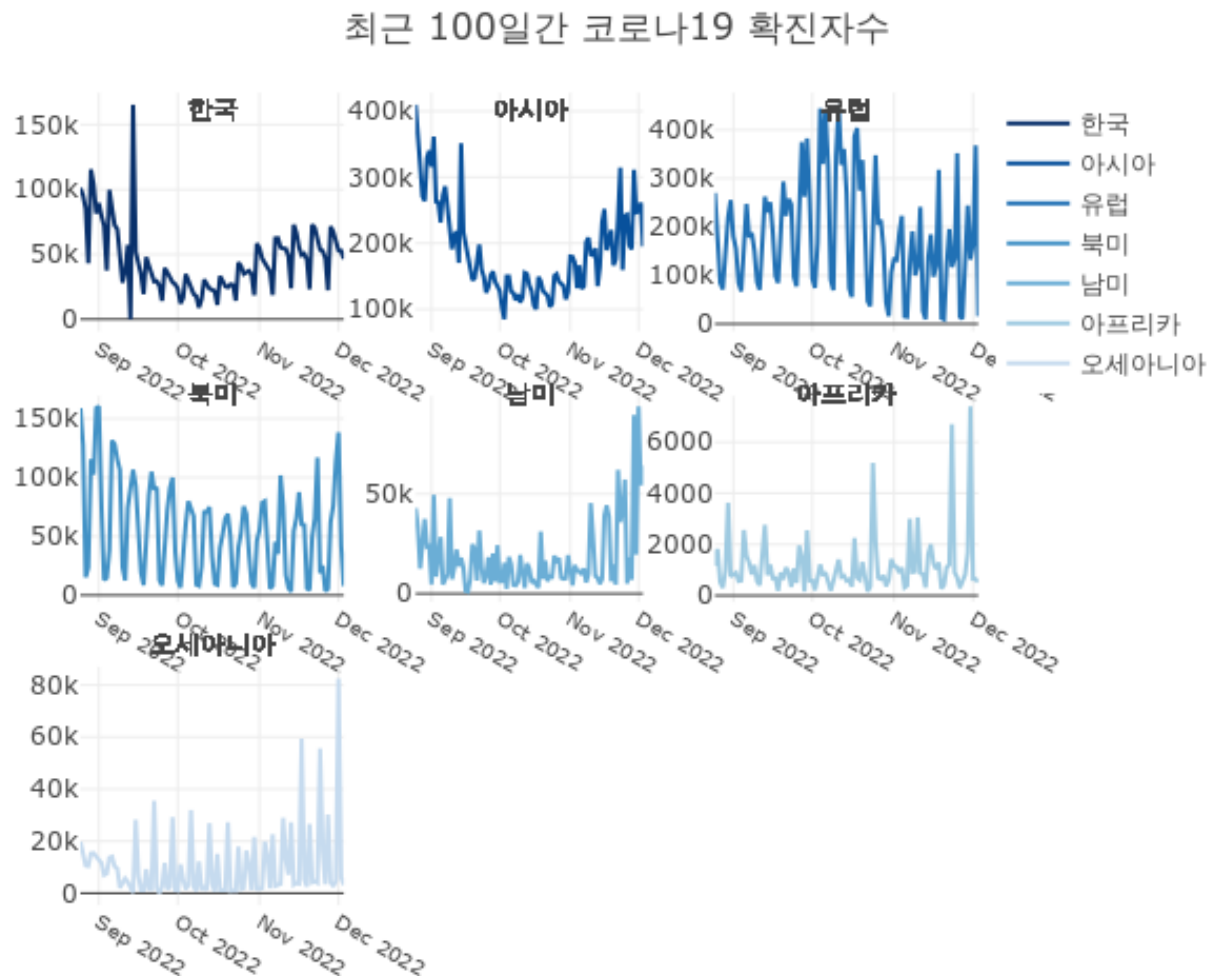
또 앞의 예와 같이 서브 플롯이 몇 개 되지 않을때는 각각 plotly 객체를 만들어주고 `subplot()`으로 묶어 전체 서브플롯을 만들어 주는 것이 가능하겠지만, 서브 플롯이 많을 때는 이러한 작업이 매우 어려워진다. 특히 긴 형태의 데이터프레임의 경우 각각의 서브 플롯을 만들기 위해 데이터를 필터링하고 plotly 객체를 만들어야하기 때문에 매우 번거로운 작업이 수반된다.

이럴 경우는 `group_by()`로 데이터프레임을 그룹화하고 `.do()`을 사용하여 각각의 그룹화된 데이터 그룹별로 plotly 객체를 만드는 방식으로 간단히 서브 플롯을 만들어 줄 수 있다.

이 때 `do`를 사용하여 각각의 그룹화된 데이터 그룹별 plotly 객체를 만드는 코드를 적용하여야 한다. 다음의 코드는 `group_by()`와 `do`를 사용하여 서브 플롯을 만드는 코드인데 각각의 서브 플롯에 `add_annotation()`을 사용하여 서브 플롯의 제목을 설정하였다.

```
df_covid19_100 |>
  ## 국가명으로 그룹화
  group_by(location) |>
  ## 그룹화한 각각의 데이터 그룹들에 적용할 코드 설정
  do(
    ## 각 그룹화한 데이터를 사용해 plotly 객체 생성
    p = plot_ly(.) |>
    ## line 모드의 스캐터 trace 추가
    add_trace(type = 'scatter', mode = 'lines',
      ## X, Y 축에 변수 매핑, color 를 설정
      x = ~date, y = ~new_cases, name = ~location) |>
    add_annotations(x = 0.5, y = 1.02, text = ~location,
      showarrow = F, xref='paper',
      yref='paper', xanchor = 'center') |>
    ## layout 으로 X, Y 축을 설정
    layout(xaxis = list(tickfont = list(size = 10)),
      yaxis = list(title = list(text = '확진자수')))
  ) |>
  ## 생성된 plotly 객체들을 subplot 생성
  subplot(nrows = 3, margin = 0.04) |>
  ## 생성된 subplot 의 layout 설정
```

```
layout(showlegend = TRUE,  
       title = '최근 100 일간 코로나 19 확진자수',  
       margin = margins_R)
```



실행결과 II-26. R의 서브플롯 제목 설정

- python

python에서 서브 플롯을 만들기 위해서는 `Figure()`로 초기화하지 않고 `make_subplots()`로 초기화하는데, 이 과정에서 서브 플롯의 전체적 구조를 설정해야 한다. 서브 플롯을 만들기 위해 가로, 세로로 몇 개의 서브 플롯을 배치할지 결정해서 'row', 'col' 매개변수로 서브 플롯의 구조를 초기화해야 한다.

`make_subplots()`의 주요 매개변수는 다음과 같다.

```
make_subplots(rows=1, cols=1, shared_xaxes=False, shared_yaxes=False,  
start_cell='top-left', print_grid=False, horizontal_spacing=None, vertical_spacing=None,  
subplot_titles=None, column_widths=None, row_heights=None, specs=None,  
insets=None, column_titles=None, row_titles=None, x_title=None, y_title=None,  
figure=None, **kwargs)
```

- rows, cols : subplot의 행과 열의 개수
- shared_xaxes, shared_yaxes : X, Y 축을 공유할지에 대한 논리값
- horizontal_spacing, vertical_spacing : 각각의 subplot의 가로, 세로 간격
- subplot_titles : 각각의 subplot의 제목의 설정
- column_widths, row_heights : 서브 플롯의 너비와 높이
- specs : 서브 플롯의 타입에 대한 설정
- x_title, y_title : 서브 플롯들의 아래와 왼쪽에 표기할 타이틀

다음은 대륙별 확진자수의 선 그래프를 서브 플롯으로 구성하는 코드이다. 먼저 `plotly.subplots`를 `make_subplots()`로 import 하고, 이를 사용하여 가로, 세로 3 개씩의 서브 플롯으로 구성되는 plotly 객체를 초기화한다. 다음으로 서브 플롯에 들어갈 트레이스를 만드는데 각각의 트레이스에 'row'와 'col' 속성을 사용하여 서브 플롯에서의 위치를 지정한다. python에서는 R과는 달리 각각의 서브 플롯의 제목을 설정할 수 있다. 다만 각각의 서브 플롯의 'layout' 설정에서 제목을 설정하는 것이 아닌 `make_subplot()`의 매개변수로 각각의 서브 플롯 제목을 'subplot_titles'을 사용하여 설정한다.

```
## 서브플롯 생성을 위한 라이브러리 импорт  
from plotly.subplots import make_subplots  
  
## 서브플롯 생성을 위한 기본 plotly 객체 생성  
fig_subplot = make_subplots(  
    rows=3, cols=3,  
    ## 서브 플롯 제목 설정  
    subplot_titles = ('한국', '아시아', '유럽', '북미', '남미', '오세아니아', '아프리카'))  
  
## 첫 번째 서브 플롯 생성  
fig_subplot.add_trace({
```

```
'type' : 'scatter', 'mode' : 'lines',  
'x': df_covid19_100_wide.index, 'y': df_covid19_100_wide['확진자_한국'],  
'name':'한국'},  
row=1, col=1)
```

두 번째 서브 플롯 생성

```
fig_subplot.add_trace({  
    'type' : 'scatter', 'mode' : 'lines',  
    'x': df_covid19_100_wide.index, 'y': df_covid19_100_wide['확진자_아시아'],  
    'name':'아시아'},  
row=1, col=2)
```

세 번째 서브 플롯 생성

```
fig_subplot.add_trace({  
    'type' : 'scatter', 'mode' : 'lines',  
    'x': df_covid19_100_wide.index, 'y': df_covid19_100_wide['확진자_유럽'],  
    'name':'유럽'},  
row=1, col=3)
```

네 번째 서브 플롯 생성

```
fig_subplot.add_trace({  
    'type' : 'scatter', 'mode' : 'lines',  
    'x': df_covid19_100_wide.index, 'y': df_covid19_100_wide['확진자_북미'],  
    'name':'북미'},  
row=2, col=1  
)
```

다섯 번째 서브 플롯 생성

```
fig_subplot.add_trace({  
    'type' : 'scatter', 'mode' : 'lines',  
    'x': df_covid19_100_wide.index, 'y': df_covid19_100_wide['확진자_남미'],  
    'name':'남미'},  
row=2, col=2)
```

여섯 번째 서브 플롯 생성

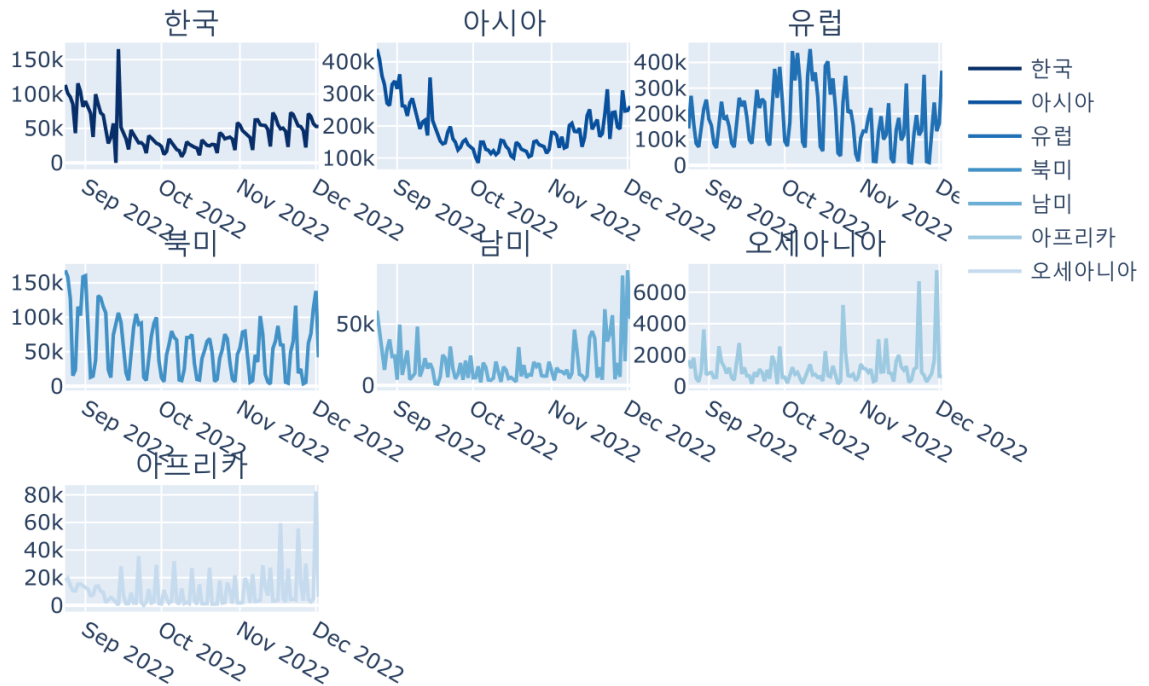
```
fig_subplot.add_trace({  
    'type' : 'scatter', 'mode' : 'lines',  
    'x': df_covid19_100_wide.index, 'y': df_covid19_100_wide['확진자_아프리카'],  
    'name':'아프리카'},  
    row=2, col=3)
```

일곱 번째 서브 플롯 생성

```
fig_subplot.add_trace({  
    'type' : 'scatter', 'mode' : 'lines',  
    'x': df_covid19_100_wide.index, 'y': df_covid19_100_wide['확진자_오세아니아'],  
    'name':'오세아니아'},  
    row=3, col=1)
```

```
fig_subplot.update_layout(title=dict(text = "최근 100 일간 코로나 19 확진자",  
    x = 0.5))
```


최근 100일간 코로나19 확진자



실행결과 II-27. python 의 서브 플롯 생성

5.2. 서브 플롯 범례 설정

앞선 예에서는 서브 플롯으로 사용되는 플롯들에 세부 제목을 붙여줌으로써 각각의 서브 플롯의 의미를 부여하였다. 그렇기 때문에 우측에 표현되는 범례가 큰 의미가 없다. 이렇게 서브 플롯의 범례는 세부 플롯마다 범례가 표시되는 것이 아니라 전체 플롯의 하나의 범례로 표시된다. 따라서 모아진 범례를 표시하거나 표시하지 않는 것을 결정하는 것은 전체 플롯의 'layout'의 'showlegend' 속성을 사용한다.

다음은 서브 플롯의 범례를 없애는 R 과 python 코드이다.

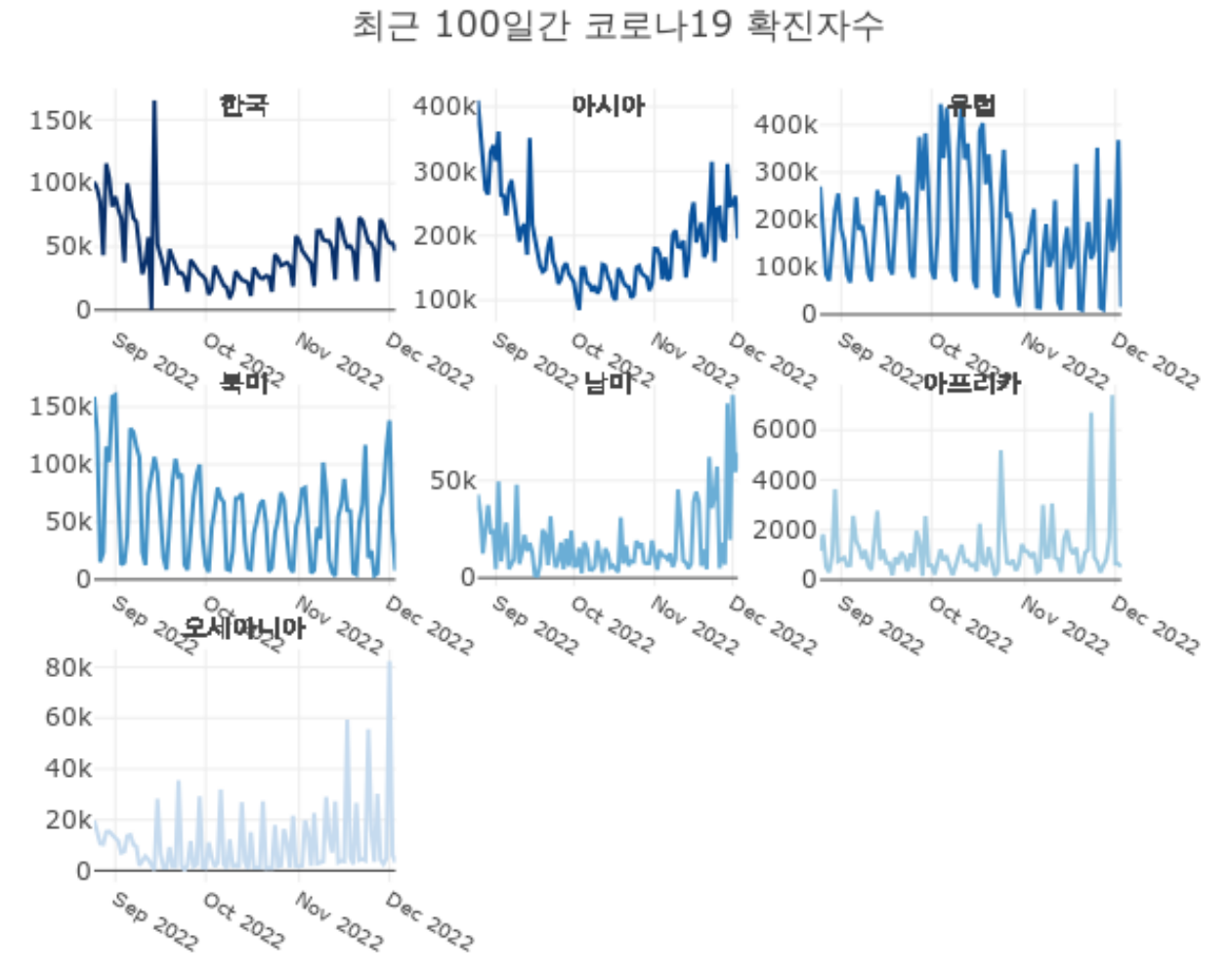
- R

```
subplots |>
  ## 범례는 제거
  layout(showlegend = FALSE)
```

- python

```
## 범례는 제거
```

```
fig_subplot.update_layout(showlegend = False)
```



실행결과 II-28. R의 서브 플롯 범례 설정

5.3. 서브 플롯 위치 배치와 편집

서브 플롯은 일반적으로 행열의 수만큼 격자형으로 배치하는 것이 일반적이다. 그러나 서브 플롯에 표현되는 데이터의 특성에 따라 격자형 배치가 아닌 사용자 정의형 배치를 사용해야 할 때도 있다. plotly에서는 서브 플롯의 크기나 위치를 편집하는 기능을 제공한다. 이 설정도 R과 python에서의 방법이 다르다.

- R

R에서의 서브 플롯은 서브 플롯안에 포함되는 플롯들의 위치를 `subplots()`에서 2 차원 격자형 형태로 배치할 수 있다. 행의 수를 설정하기 위해서는 'nrows' 속성을 사용하지만 열의 수를 설정할 수는 없다는 점을 주의해야 한다.

기본적으로 서브 플롯에 포함되는 플롯들의 크기는 서브 플롯에 설정된 행과 열의 수에 따라 전체 높이와 너비를 동일한 비율로 공유한다. 하지만 이 비율은 'heights', 'widths'를 사용하여 변경할 수 있는데, 각각의 플롯의 크기를 변경함으로써 플롯들의 위치와 크기를 사용자가 원하는 대로 편집할 수 있다. 'heights'와 'widths'는 전체 플롯의 높이와 너비를 1로 보고 상대적인 크기를 설정하며, 여러개의 'heights'와 'widths'가 필요하다면 `c()`를 사용하여 벡터로 만들어야 한다.

서브 플롯을 원하는 대로 배치하고 크기를 설정하기 위해서는 서브 플롯이 배치되는 방향을 잘 고려해야 한다. 앞서 살펴본 서브 플롯의 경우 p1 부터 p7 까지를 순서대로 `subplot()`의 매개변수로 설정했다. 그리고 'nrows'를 3으로 설정했기 때문에 7개의 플롯을 표시하기 위해서는 자동적으로 열의 수가 3으로 설정된다. 전체 플롯을 9개로 등분하고 각각의 매개변수 호출 순서대로 열 방향으로 배치하게 된다. 전체 플롯을 9등분 하기 때문에 2개의 등분은 빈 공간으로 남게 된다. 만약 빈 공간을 마지막 행에 두지 않고 원하는 위치에 두고자 한다면, 원하는 위치에 `plotly_empty()`로 빈 plotly 객체를 설정해 준다.

앞의 예에서는 1 행에 p1(한국), p2(아시아), p3(유럽), 2 행에 p4(북미), p5(남미), p6(오세아니아)가 배치되고 마지막 행에는 하나 남은 p7(아프리카)가 1 열에 배치되며 나머지 2 열은 비게 된다. 만약 p1 플롯을 좀 부각시키기 위해서는 다음과 같이 설정할 수 있다.

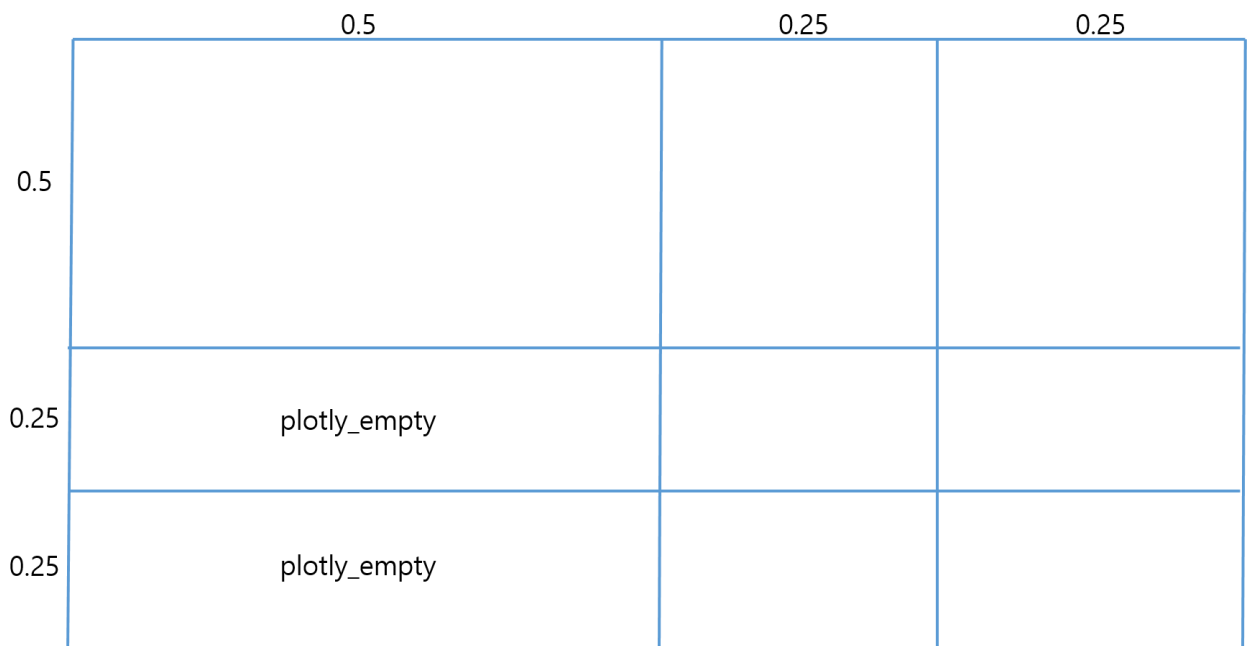
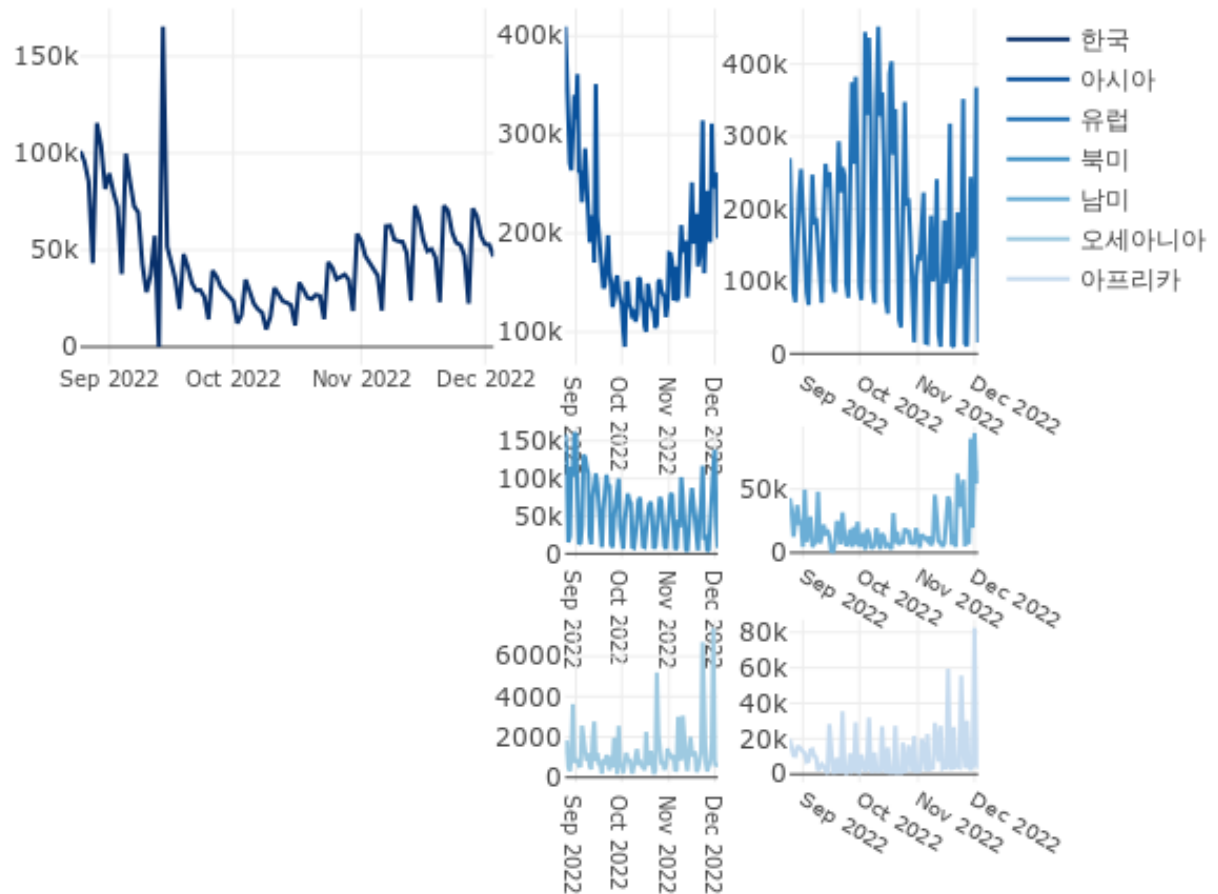


그림 11-6. 서브플롯의 배치

전체 9 개의 서브플롯 공간의 높이와 넓이를 0.5, 0.25, 0.25 로 각각 설정한다. 그러면 앞의 그림과 같이 길이와 너비가 각각 설정된 9 개의 서브 플롯 공간이 생기는데 여기서 비워둘 공간은 `plotly_empty()`를 사용하고 각각의 위치에 들어갈 plotly 객체를 `subplot()`에서 배치한다.

```
subplot(
  p1, p2, p3, plotly_empty(), p4, p5, plotly_empty(), p6, p7,
  ## 서브플롯은 3 개의 열로 설정
  nrows = 3,
  ## 서브플롯간의 여백 설정
  heights = c(0.5, 0.25, 0.25),
  widths = c(0.5, 0.25, 0.25),
  margin = 0.04) |>
  ## 범례는 제거
  layout(showlegend = TRUE,
    ## 전체 제목 설정
    title = '최근 100 일간 코로나 19 확진자수',
    ## 전체 여백 설정
    margin = margins_R)
```

최근 100일간 코로나19 확진자수



실행결과 II-29. R의 서브 플롯의 크기와 배치 설정

앞의 서브 플롯을 보면 한국 플롯이 크게 표현되어 강조되는 효과를 내기는 했지만 같은 행, 같은 열에 있는 플롯들은 한쪽으로 길게 표시되어 보기에 어색해 보인다. 이를 해결하기 위해서는 `subplot()`을 중첩해서 해결할 수 있다. `subplot()`을 중첩하여 사용한다는 것은 `subplot()`으로 완성된 서브 플롯을 다시 `subplot()`으로 배열한다는 것이다.

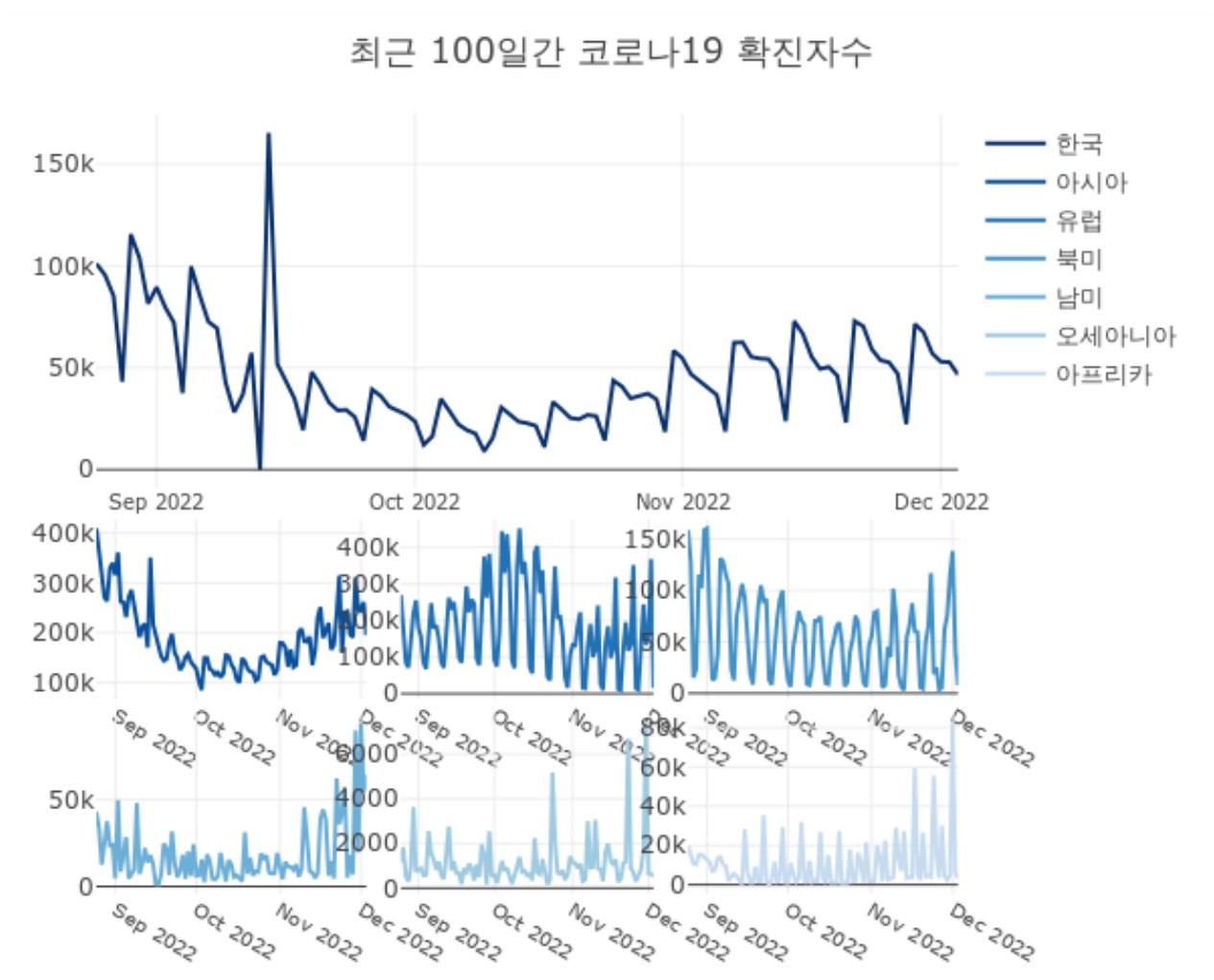
다음은 한국을 위에 길게 표현하고 아래에 6 개의 플롯을 2 행으로 배열하는 예이다. 다음의 예에서는 p1(한국)과 p2 부터 p7 까지의 플롯을 `subplot()`으로 하나의 plotly 로 생성하여 이 두개의 플롯을 다시 `subplot()`으로 붙여주는 코드이다.

```
subplot(
  ## 아래의 nrow 가 2 이기 때문에 맨 위 열에 p1 하나를 위치시킴
  p1,
  ## subplot()으로 p2 부터 p7 까지를 묶어 하나의 플롯으로 만들
```

```

subplot(p2, p3, p4, p5, p6, p7,
## 서브플롯은 2 개의 열로 설정함으로써 2 행 3 열 서브 플롯 생성
nrows = 2),
## 전체 서브 플롯은 2 열로 구성
nrows = 2
) |>
## 범례는 제거
layout(showlegend = TRUE,
## 전체 제목 설정
title = '최근 100 일간 코로나 19 확진자수',
## 전체 여백 설정
margin = margins_R)

```



실행결과 II-30. R의 서브 플롯의 크기와 배치 설정

- python

python에서는 R과 같이 서브 플롯의 행과 열의 높이와 너비를 설정하는 방법을 사용할 수 있다. 그리고 또 하나의 방법을 제공하는데 `make_subplots()`에서 'specs'를 사용하여 서브 플롯의 위치와 크기를 편집할 수도 있다.

다음은 서브 플롯의 행과 열의 높이와 너비를 설정하는 방법이다. 행의 높이를 설정하기 위해서는 'row_heights', 열의 너비를 설정하기 위해서는 'column_widths'를 사용하는데 행과 열의 수에 맞게 각각의 서브플롯의 행과 열의 높이와 너비가 설정되어야 하며 높이와 너비는 전체 값을 1 하여 상대적 높이와 너비를 설정한다. 다음에서는 행의 높이를 [0.5, 0.25, 0.25], 열의 너비를 [0.5, 0.25, 0.25]로 설정하였다.

```
from plotly.subplots import make_subplots

fig_subplot = make_subplots(
    rows=3, cols=3,
    column_widths=[0.5,0.25,0.25], ## 열 방향 너비 설정
    row_heights=[0.5,0.25,0.25], ## 행 방향 너비 설정
    subplot_titles=('한국', '아시아', '유럽', '', '북미', '남미', '', '오세아니아', '아프리카'))

fig_subplot.add_trace({
    'type': 'scatter', 'mode': 'lines',
    'x': df_covid19_100_wide.index, 'y': df_covid19_100_wide['확진자_한국'],
    'name': '한국'},
    row=1, col=1)

fig_subplot.add_trace({
    'type': 'scatter', 'mode': 'lines',
    'x': df_covid19_100_wide.index, 'y': df_covid19_100_wide['확진자_아시아'],
    'name': '아시아'},
    row=1, col=2)

fig_subplot.add_trace({
    'type': 'scatter', 'mode': 'lines',
    'x': df_covid19_100_wide.index, 'y': df_covid19_100_wide['확진자_유럽'],
```

```
'name':'유럽'},  
row=1, col=3)
```

```
fig_subplot.add_trace({  
    'type' : 'scatter', 'mode' : 'lines',  
    'x': df_covid19_100_wide.index, 'y': df_covid19_100_wide['확진자_북미'],  
    'name':'북미'},  
row=2, col=2)
```

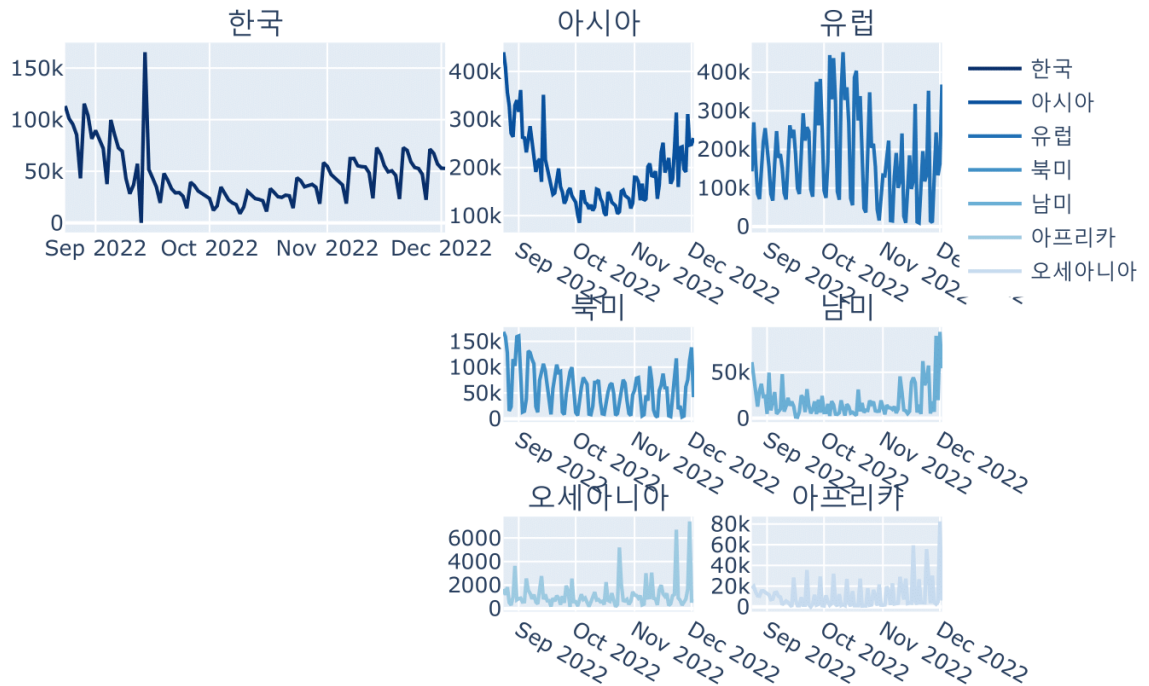
```
fig_subplot.add_trace({  
    'type' : 'scatter', 'mode' : 'lines',  
    'x': df_covid19_100_wide.index, 'y': df_covid19_100_wide['확진자_남미'],  
    'name':'남미'},  
row=2, col=3)
```

```
fig_subplot.add_trace({  
    'type' : 'scatter', 'mode' : 'lines',  
    'x': df_covid19_100_wide.index, 'y': df_covid19_100_wide['확진자_아프리카'],  
    'name':'아프리카'},  
row=3, col=2)
```

```
fig_subplot.add_trace({  
    'type' : 'scatter', 'mode' : 'lines',  
    'x': df_covid19_100_wide.index, 'y': df_covid19_100_wide['확진자_오세아니아'],  
    'name':'오세아니아'},  
row=3, col=3)
```

```
fig_subplot.update_layout(title=dict(text = "최근 100 일간 코로나 19 확진자",  
                                     x = 0.5))
```


최근 100일간 코로나19 확진자



실행결과 II-31. python 의 서브 플롯의 크기와 배치 설정

두 번째 방법은 'specs'를 사용하는 방법이다. 'specs'는 서브 플롯 별로 옵션을 설정하는 매개변수로서 'rows'와 'cols'로 만들어지는 2 차원 서브 플롯 그리드에 일치하는 list 로 구성된다.

'specs'로 해당 서브 플롯 그리드 셀의 길이('rowspan')와 너비('colspan')을 설정한다. 셀의 길이와 너비는 셀 단위의 크기를 설정하는데 예를 들어 특정 서브 플롯이 2 칸의 행, 3 칸의 열 크기를 사용한다면 'colspan'이 3 로, 'rowspan'이 2 으로 설정된 딕셔너리로 해당 셀을 설정해야 한다. 만약 셀을 비워야 한다면 'None' 키워드를 사용한다.

다음은 한국의 서브 플롯을 상단 2 행, 3 열의 크기로 설정하고 나머지 서브 플롯을 배치하는 python 코드이다. 여기서 'colspan'과 'rowspan'이 모두 1 로 설정된다면 설정을 생략하고 {}만 설정해도 가능하다.

```
from plotly.subplots import make_subplots
```

```
fig_subplot = make_subplots(
```

```

rows=4, cols=3,
## specs 를 사용한 서브플롯 구성 설정
specs = [
    [{'colspan' : 3, 'rowspan' : 2}, None, None],
    [None, None, None],
    [{}, {}, {}],
    [{}, {}, {}]
],
subplot_titles = ('한국', '아시아', '유럽', '북미', '남미', '오세아니아', '아프리카'))

```

```

fig_subplot.add_trace({
    'type' : 'scatter', 'mode' : 'lines',
    'x': df_covid19_100_wide.index, 'y': df_covid19_100_wide['확진자_한국'],
    'name': '한국'},
    row=1, col=1)

```

```

fig_subplot.add_trace({
    'type' : 'scatter', 'mode' : 'lines',
    'x': df_covid19_100_wide.index, 'y': df_covid19_100_wide['확진자_아시아'],
    'name': '아시아'},
    row=3, col=1)

```

```

fig_subplot.add_trace({
    'type' : 'scatter', 'mode' : 'lines',
    'x': df_covid19_100_wide.index, 'y': df_covid19_100_wide['확진자_유럽'],
    'name': '유럽'},
    row=3, col=2)

```

```

fig_subplot.add_trace({
    'type' : 'scatter', 'mode' : 'lines',
    'x': df_covid19_100_wide.index, 'y': df_covid19_100_wide['확진자_북미'],
    'name': '북미'},
    row=3, col=3)

```

```
fig_subplot.add_trace({
    'type' : 'scatter', 'mode' : 'lines',
    'x': df_covid19_100_wide.index, 'y': df_covid19_100_wide['확진자_남미'],
    'name': '남미'},
    row=4, col=1)

fig_subplot.add_trace({
    'type' : 'scatter', 'mode' : 'lines',
    'x': df_covid19_100_wide.index, 'y': df_covid19_100_wide['확진자_아프리카'],
    'name': '아프리카'},
    row=4, col=2)

fig_subplot.add_trace({
    'type' : 'scatter', 'mode' : 'lines',
    'x': df_covid19_100_wide.index, 'y': df_covid19_100_wide['확진자_오세아니아'],
    'name': '오세아니아'},
    row=4, col=3)

fig_subplot.update_layout(title=dict(text = "최근 100 일간 코로나 19 확진자",
    x = 0.5))
```

최근 100일간 코로나19 확진자

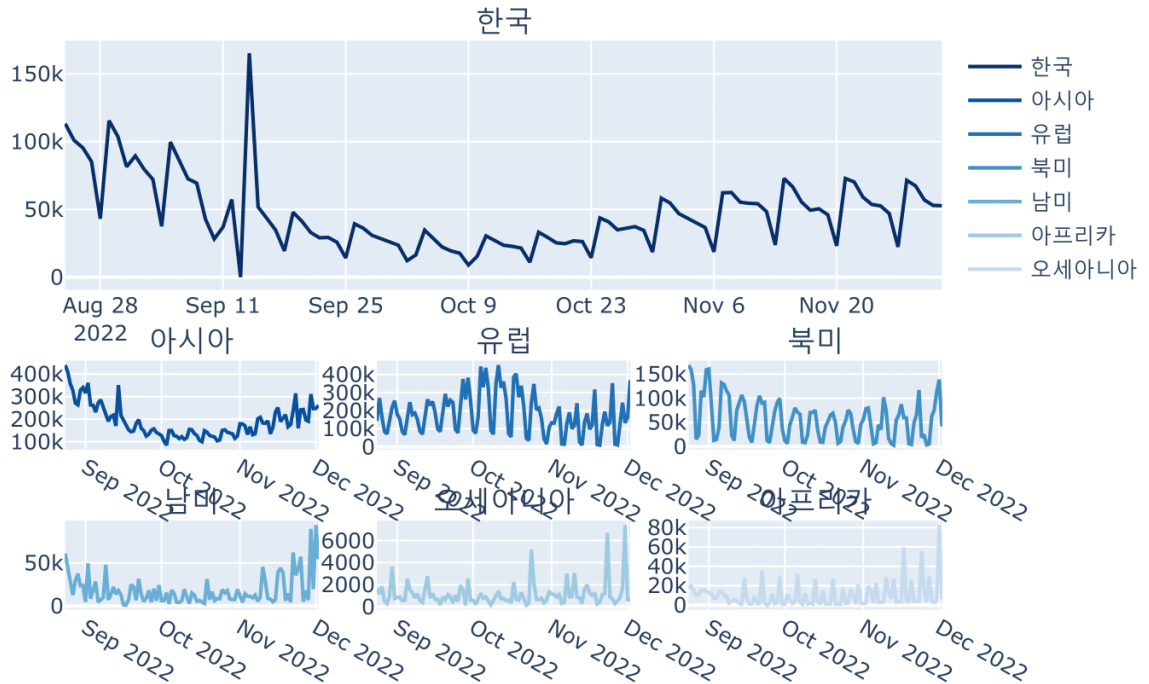


그림 II-32. python의 spec을 사용한 서브 플롯의 크기와 배치 설정

5.4. 축 공유

서브 플롯에서 중요하게 사용되는 속성값이 몇 개 있는데 그 중 꼭 알아두어야 하는 속성 값이 축 공유에 관련된 속성이다. R의 경우 'shareX'와 'shareY', python의 경우 'shared_xaxes', 'shared_yaxes'이다. 이 두 속성은 서브 플롯에 표시되는 각각의 플롯의 X, Y 축을 공유할지를 결정하는 속성이다. 이 속성값은 R의 경우 `subplot()`에서, python의 경우 `make_subplots()`에서 설정하는데 'shareX'나 'shared_xaxes' 값이 TRUE로 설정되면 같은 열의 서브 플롯들의 X 축은 모두 같은 스케일로 공유된다. 하지만 FALSE로 설정할 경우 모두 각각의 서브 플롯에 적합한 축으로 표현되기 때문에 서브 플롯별로 X 축의 스케일이 달라질 수 있다.

마찬가지로 'shareY'나 'shared_yaxes'가 TRUE로 설정되면 같은 행에 표현된 서브 플롯은 같은 스케일의 Y 축을 공유한다.

다음은 X, Y 축을 공유하는 서브플롯을 그리는 R과 python 코드이다.

- R

```

subplot(
    p1,
    subplot(p2, p3, p4, p5, p6, p7,
        nrows = 2,
        ## shareX, shareY를 TRUE로 설정하여 축 공유
        shareX = TRUE, shareY = TRUE),
    nrows = 2, margin = 0.04
) |>
layout(showlegend = TRUE,
    title = '최근 100일간 코로나 19 확진자수',
    margin = margins_R)

```

- python

```

fig_subplot = make_subplots(
    rows=4, cols=3,
    specs = [
        [{'colspan' : 3, 'rowspan' : 2}, None, None],
        [None, None, None],
        [{}, {}, {}],
        [{}, {}, {}]
    ],
    ## shareX, shareY를 TRUE로 설정하여 축 공유
    shared_xaxes = True, shared_yaxes = True,
    subplot_titles = ('한국', '아시아', '유럽', '북미', '남미', '오세아니아', '아프리카'))

fig_subplot.add_trace({
    'type' : 'scatter', 'mode' : 'lines',
    'x': df_covid19_100_wide.index, 'y': df_covid19_100_wide['확진자_한국'],
    'name': '한국'},
    row=1, col=1)

fig_subplot.add_trace({
    'type' : 'scatter', 'mode' : 'lines',
    'x': df_covid19_100_wide.index, 'y': df_covid19_100_wide['확진자_아시아'],
    'name': '아시아'},

```

```
row=3, col=1)
```

```
fig_subplot.add_trace({  
    'type' : 'scatter', 'mode' : 'lines',  
    'x': df_covid19_100_wide.index, 'y': df_covid19_100_wide['확진자_유럽'],  
    'name':'유럽'},  
    row=3, col=2)
```

```
fig_subplot.add_trace({  
    'type' : 'scatter', 'mode' : 'lines',  
    'x': df_covid19_100_wide.index, 'y': df_covid19_100_wide['확진자_북미'],  
    'name':'북미'},  
    row=3, col=3)
```

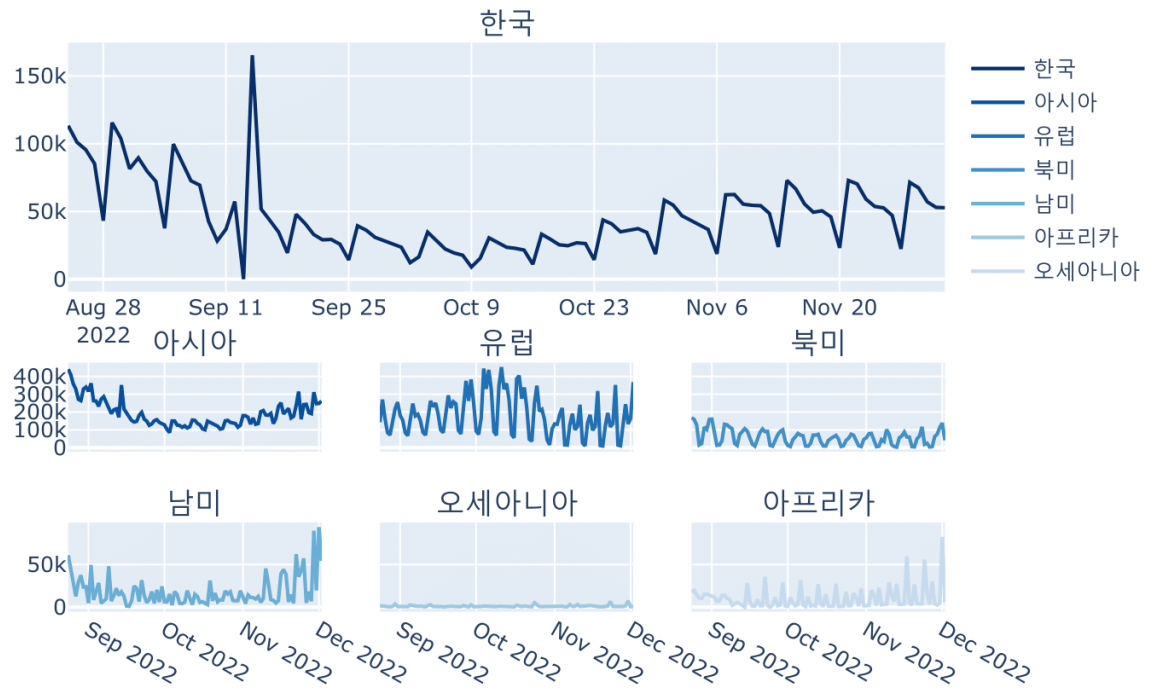
```
fig_subplot.add_trace({  
    'type' : 'scatter', 'mode' : 'lines',  
    'x': df_covid19_100_wide.index, 'y': df_covid19_100_wide['확진자_남미'],  
    'name':'남미'},  
    row=4, col=1)
```

```
fig_subplot.add_trace({  
    'type' : 'scatter', 'mode' : 'lines',  
    'x': df_covid19_100_wide.index, 'y': df_covid19_100_wide['확진자_아프리카'],  
    'name':'아프리카'},  
    row=4, col=2)
```

```
fig_subplot.add_trace({  
    'type' : 'scatter', 'mode' : 'lines',  
    'x': df_covid19_100_wide.index, 'y': df_covid19_100_wide['확진자_오세아니아'],  
    'name':'오세아니아'},  
    row=4, col=3)
```

```
fig_subplot.update_layout(title=dict(text = "최근 100 일간 코로나 19 확진자",  
                                     x = 0.5))
```

최근 100일간 코로나19 확진자



실행결과 II-33. python 의 축 공유가 설정된 서브 플롯