1. plotly 시각화 만들기

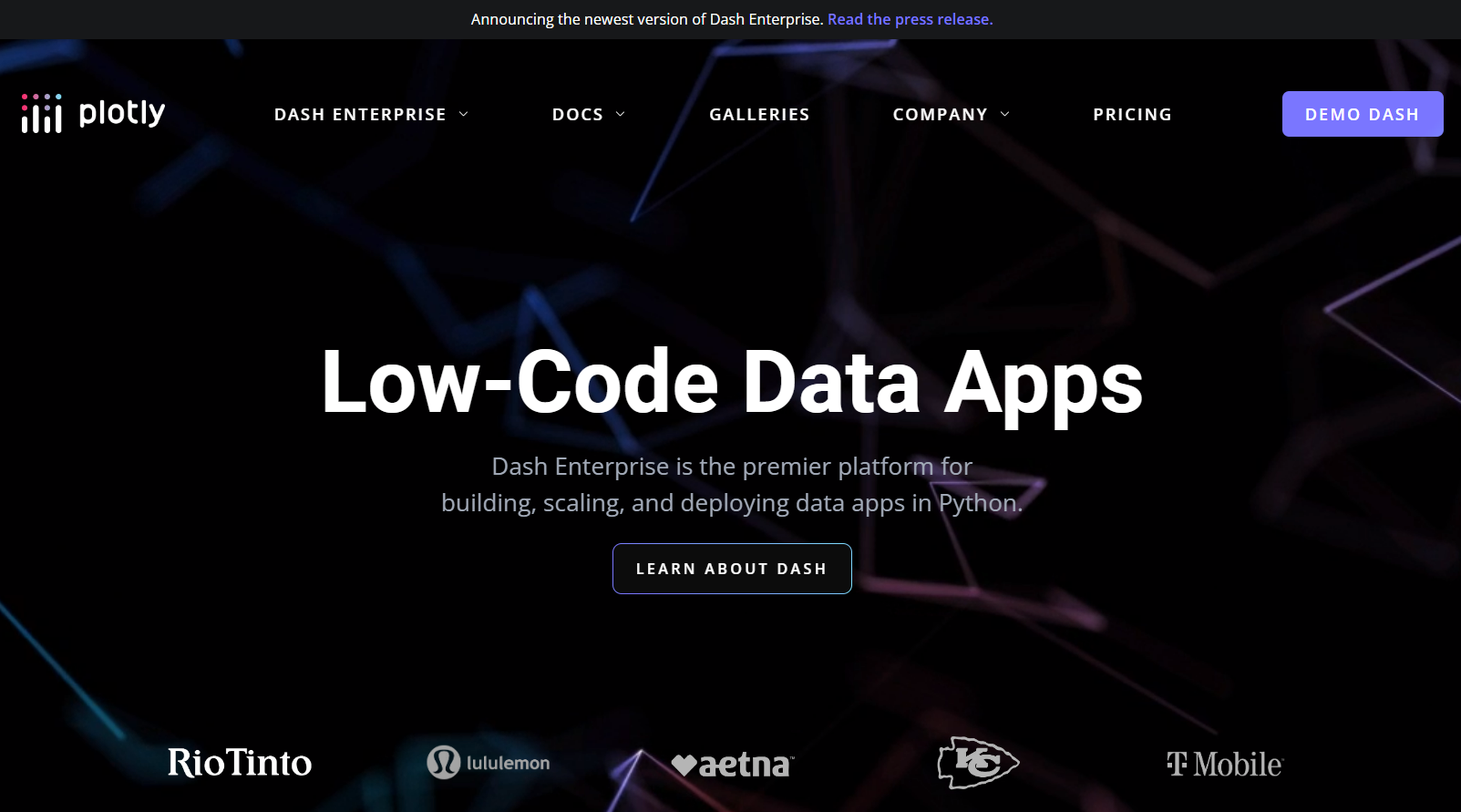
데이터 분석을 다루는 많은 교육코스나 서적에서 데이터의 시각화는 R의 경우 R base나 ggplot2 패키지를 사용하여는 방법, 파이썬의 경우 matplotlib이나 seaborn 패키지를 사용하는 방법을 위주로 설명한다. 이러한 방법들은 데이터 시각화 방법이 간단하고 그 품질이 좋은 편이기 때문에 많이 사용되고 있지만 정적(Static) 시각화라는 한계를 가진다. 정적 시각화는 대부분 이미지로 저장되며 인포그래픽(Infographic)이라고도 불리며, 일반적으로 문서나 인쇄물에 많이 사용되고 웹에 게시되는 이미지로도 사용된다. 그렇기 때문에 대부분 png, jpg, pdf 등의 벡터 혹은 픽셀 이미지 파일로 제공된다. 정적 데이터 시각화는 데이터 분석가의 의도에 맞춰 작성되기 때문에 데이터 분석가의 데이터 분석 관점에 의존적일 수 밖에 없으며 시각화를 사용하는 사용자의 의도에 따른 해석은 매우 제한될 수 밖에 없다.

이러한 제한점을 극복하기 위해 사용되는 데이터 시각화 방법이 동적(Dynamic) 시각화 혹은 인터랙티브(Interactive) 시각화라고 하는 방법이다. 이 동적 시각화는 시각화를 사용하는 사용자의 의도에 따라 데이터를 다각적 관점에서 살펴볼 수 있다는 점이 특징이다. 사용자의 의도에 따라 데이터가 동적으로 변동되어야 하기 때문에 데이터 시각화가 많이 사용되는 인쇄물 형태 매체에서 사용이 어렵고 웹을 통해 사용되어야 그 장점을 충실히 사용할 수 있다. 따라서 동적 시각화는 웹 사이트에서 제공하는 대시보드(DashBoard)의 형태로 제공되는 것이 일반적이기 때문에 동적 시각화를 위해서는 R이나 파이썬에서 주로 사용되는 시각화 패키지가 아닌 동적 시각화 전용 패키지를 사용해야 한다. R에서는 동적 시각화를 위해 rbokeh, highcharter 등이 사용되었고 파이썬에서는 bokeh, hvplot 등을 사용되었다. 하지만 R과 파이썬 모두에서 사용되는 plotly 패키지가 등장함으로써 R과 파이썬의 동적 시각화 패키지 시장에서 매우 빠르게 사용자층을 넓혀가고 있다.

하지만 정적 시각화와 동적 시각화의 어느것이 더 효용성이 있는지를 단언할 수 없다. 데이터 시각화가 사용되는 매체, 데이터 시각화를 보는 대상, 데이터 시각화에서 보여주고자 하는 스토리에 따라서 정적 시각화를 사용해야 할 때와 동적 시각화를 사용해야 할 때를 적절히 선택해야 한다.

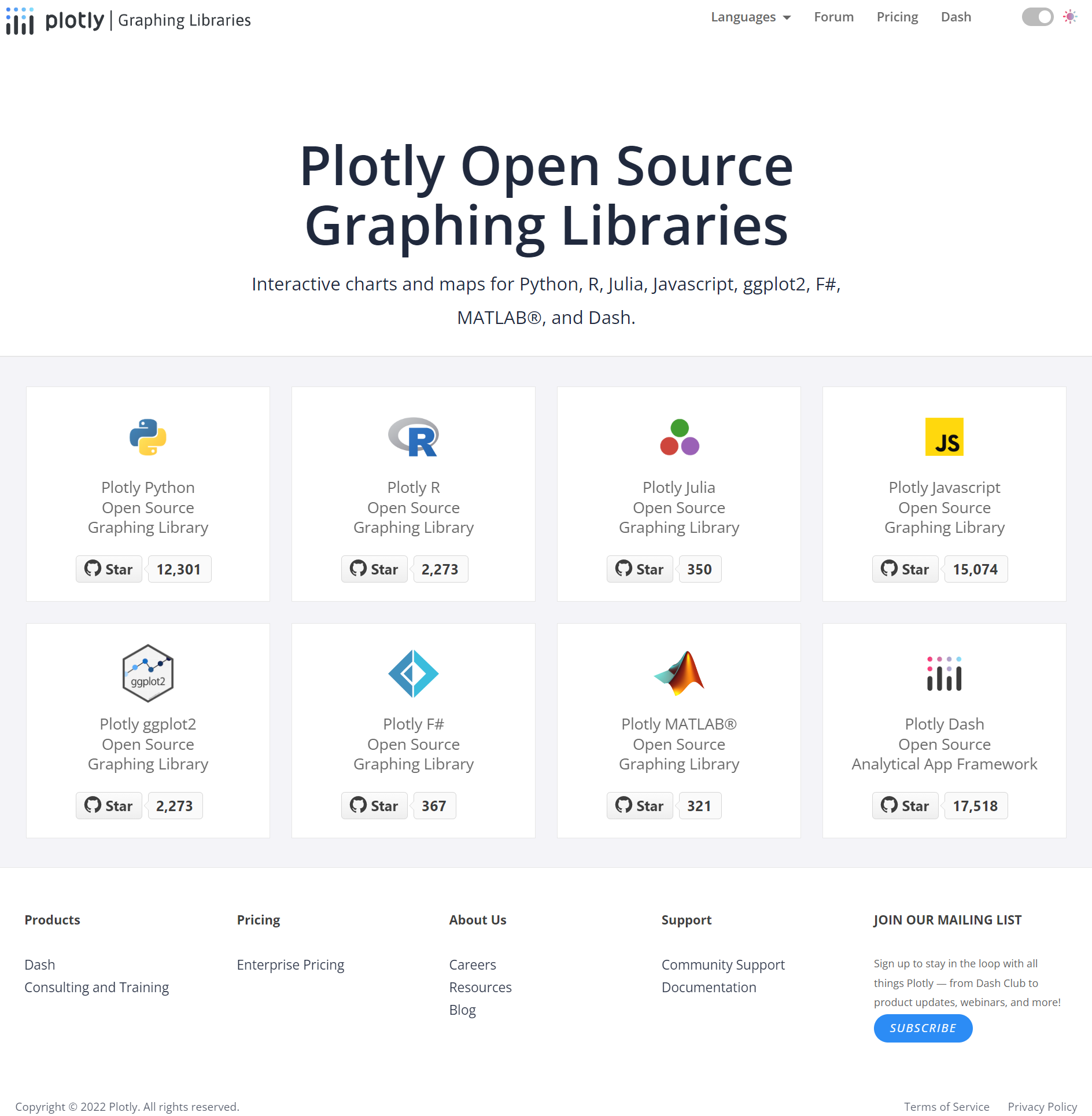
# plotly란?

‘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 파이썬의 plotly 패키지 외에도 R, 파이썬, 줄리아 등의 개발도구에서 사용가능한 데이터 대시보드 플랫폼인 dash를 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.



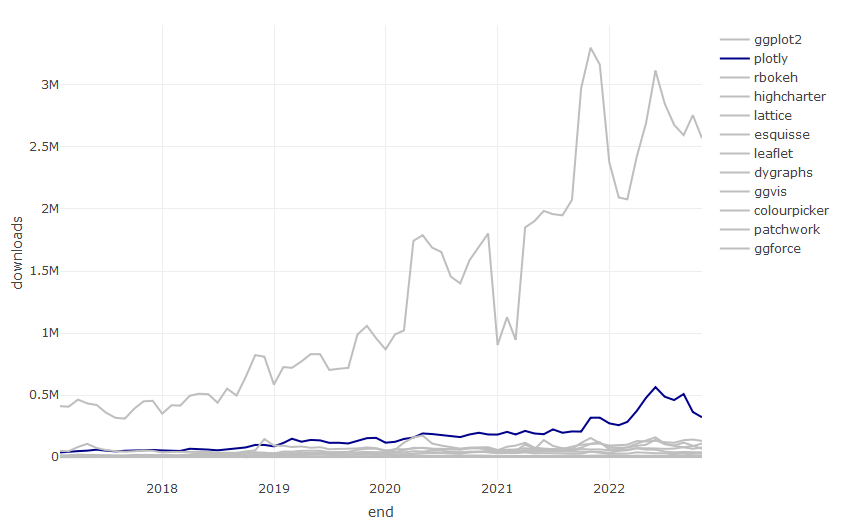
plotly 홈페이지 화면

plotly 패키지는 R, 파이썬, Julia, Java Script, F#, MATLAB 등의 다양한 언어에서 사용이 가능하도록 각각의 언어에 바인딩되는 패키지를 개발하여 제공하고 있다. plotly에서 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(Scienctific Chart), 시계열 차트, 캔들 차트와 같은 재정 차트(Finantial Chart) 등의 다양한 차트와 플롯을 제공한다.

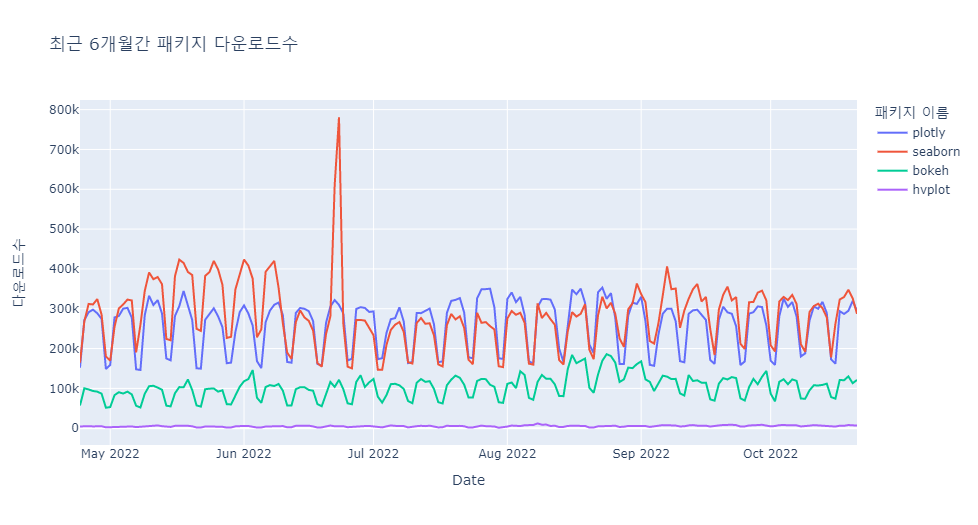


plotly 패키지 홈페이지

다음의 R 그래픽 패키지 다운로드 현황에서 보듯이 여전히 R에서 가장 많이 사용되고 있는 그래픽 패키지는 ggplot2이지만 plotly는 2021년 하반기부터 다운로드가 늘고 있고 다른 그래픽 패키지에 비해서는 압도적인 다운로드 수를 보인다.



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



파이썬 plotly 다운로드

# 예제 데이터 Import와 전처리

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

## Covid19 데이터 셋

첫번째 데이터 셋은 2020년 1월부터 기록된 전세계 국가의 코로나19 발병 관련 데이터이다. 이 데이터는 Github에서 다양한 전세계 데이터를 배포하는 ’Open World in Data’에서 제공하는 ’COVID-19 Dataset by Our World in Data’를 사용한다.[[1]](#footnote-30) 이 데이터는 온라인으로 매일 업데이트되기 때문에 다운로드 시점에 따라 시각화 결과가 책과 다소 달라질 수 있다.[[2]](#footnote-32)

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

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

## [1] "2022-07-20"

######################################   
## python 코드  
## Covid19 데이터 셋  
import pandas as pd  
from datetime import datetime, timedelta  
from pandas.api.types import CategoricalDtype  
from matplotlib import pyplot as plt  
import plotly.graph\_objects as go  
  
df\_covid19 = pd.read\_csv("https://covid.ourworldindata.org/data/owid-covid-data.csv")  
  
df\_covid19['date'] = pd.to\_datetime(df\_covid19['date'], format="%Y-%m-%d")  
  
df\_covid19\_100 = df\_covid19[(df\_covid19['iso\_code'].isin(['KOR', 'OWID\_ASI', 'OWID\_EUR', 'OWID\_OCE', 'OWID\_NAM', 'OWID\_SAM', 'OWID\_AFR'])) & (df\_covid19['date'] >= (max(df\_covid19['date']) - timedelta(days = 100)))]  
  
  
df\_covid19\_100.loc[df\_covid19\_100['location'] == 'South Korea', "location"] = '한국'  
df\_covid19\_100.loc[df\_covid19\_100['location'] == 'Asia', "location"] = '아시아'  
df\_covid19\_100.loc[df\_covid19\_100['location'] == 'Europe', "location"] = '유럽'  
df\_covid19\_100.loc[df\_covid19\_100['location'] == 'Oceania', "location"] = '오세아니아'  
df\_covid19\_100.loc[df\_covid19\_100['location'] == 'North America', "location"] = '북미'  
df\_covid19\_100.loc[df\_covid19\_100['location'] == 'South America', "location"] = '남미'  
df\_covid19\_100.loc[df\_covid19\_100['location'] == 'Africa', "location"] = '아프리카'  
  
ord = CategoricalDtype(categories = ['한국', '아시아', '유럽', '북미', '남미', '아프리카', '오세아니아'], ordered = True)  
  
df\_covid19\_100['location'] = df\_covid19\_100['location'].astype(ord)  
  
df\_covid19\_100 = df\_covid19\_100.sort\_values(by = 'date')  
  
df\_covid19\_100\_wide = df\_covid19\_100.loc[:,['date', 'location', 'new\_cases', 'people\_fully\_vaccinated\_per\_hundred']].rename(columns={'new\_cases':'확진자', 'people\_fully\_vaccinated\_per\_hundred':'백신접종완료자'})  
  
df\_covid19\_100\_wide = df\_covid19\_100\_wide.pivot(index='date', columns='location', values=['확진자', '백신접종완료자']).sort\_values(by = 'date')  
  
df\_covid19\_stat = df\_covid19.groupby(['iso\_code', 'continent', 'location']).agg(  
 인구수 = ('population', 'max'),  
 인당GDP = ('gdp\_per\_capita', 'max'),   
 전체확진자수 = ('new\_cases', 'sum'),  
 전체사망자수 = ('new\_deaths', 'sum'),   
 십만명당중환자실 = ('icu\_patients\_per\_million', 'last'),  
 재생산지수 = ('reproduction\_rate', 'last'),  
 봉쇄지수 = ('stringency\_index', 'max'),   
 전체검사자수 = ('total\_tests', 'max'),   
 신규검사자수 = ('new\_tests', 'sum'),  
 전체백신접종자수 = ('total\_vaccinations', 'max'),  
 백신접종자완료자수 = ('people\_fully\_vaccinated', 'max'),  
 부스터접종자수 = ('total\_boosters', 'max'),  
 인구백명당백신접종완료률 = ('people\_fully\_vaccinated\_per\_hundred', 'max'),  
 인구백명당부스터접종자수 = ('total\_boosters\_per\_hundred', 'max')  
)  
  
df\_covid19\_stat['십만명당사망자수'] = round(df\_covid19\_stat['전체사망자수'] / df\_covid19\_stat['인구수'] \*100000, 5)  
  
df\_covid19\_stat['백신접종완료률'] = df\_covid19\_stat['백신접종자완료자수'] / df\_covid19\_stat['인구수']

## 대학 학과 취업률 데이터 셋

최근 청년층 실업 문제가 사회적 문제로 대두됨에 따라 대학 졸업생의 취업률이 매우 중요하게 활용되고 있는 데이터이다. 이 데이터는 대학 입학을 앞둔 수험생이나 학부모에게 대학 진학을 위한 학과 선택에 중요한 데이터이고 대학 입장에서는 학생들의 진로 지도를 위해 중요하게 사용되는 데이터이다. 이 데이터는 교육통계서비스 홈페이지에서 제공한다.[[3]](#footnote-34)

취업률 데이터 셋은 다음과 같이 데이터를 로딩하고 전처리한다.

## R 코드  
  
df\_취업률 <- read\_excel('d:/R/data/2020년 학과별 고등교육기관 취업통계.xlsx',   
 ## '학과별' 시트의 데이터를 불러오는데,  
 sheet = '학과별',  
 ## 앞의 13행을 제외하고  
 skip = 13,   
 ## 첫번째 행은 열 이름으로 설정  
 col\_names = TRUE,   
 ## 열의 타입을 설정, 처음 9개는 문자형으로 다음 79개는 수치형으로 설정  
 col\_types = c(rep('text', 9), rep('numeric', 79)))  
  
## df\_취업률에서 첫번째부터 9번째까지의 열과 '계'로 끝나는 열을 선택하여 다시 df\_취업률에 저장  
df\_취업률 <- df\_취업률 |>   
 select(1:9, ends\_with('계'), '입대자')  
  
## df\_취업률에서 졸업자가 500명 이하인 학과 2000개 샘플링  
df\_취업률\_2000 <- df\_취업률 |>   
 filter(졸업자\_계 < 500) |>  
 mutate(id = row\_number()) |>  
 filter(row\_number() %in% seq(from = 1, to = nrow(df\_취업률), by = 4))  
  
## 열 이름을 적절히 설정  
names(df\_취업률\_2000)[10:12] <- c('졸업자수', '취업률', '취업자수')

######################################   
## python 코드  
## 대학 학과 취업률 데이터 셋  
  
df\_취업률 = pd.read\_excel("d:/R/data/2020년 학과별 고등교육기관 취업통계.xlsx",   
 sheet\_name = '학과별',  
 skiprows=(13),   
 header = 0)  
  
df\_취업률 = pd.concat([df\_취업률.iloc[:, 0:8],   
 df\_취업률.loc[:, df\_취업률.columns.str.endswith('계')],   
 df\_취업률.loc[:, '입대자']],   
 axis = 1  
 )  
  
df\_취업률\_2000 = df\_취업률.loc[(df\_취업률['졸업자\_계'] < 500)]  
  
df\_취업률\_2000 = df\_취업률\_2000.iloc[range(0, len(df\_취업률\_2000.index) , 4)]  
  
df\_취업률\_2000 = df\_취업률\_2000.rename(columns = {'졸업자\_계':'졸업자수', '취업률\_계':'취업률', '취업자\_합계\_계':'취업자수'})

# plotly의 구조

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

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

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

python의 경우는 이 속성값은 전체 경로(Path)를 ‘.’을 사용해 부모와 자식을 계속해서 연결해 설정한다. 예를 들어 layout.title.text는 ’layout’ 속성의 자식노드인 ‘title’ 속성의 자식노드인 ’text’를 지칭한다.

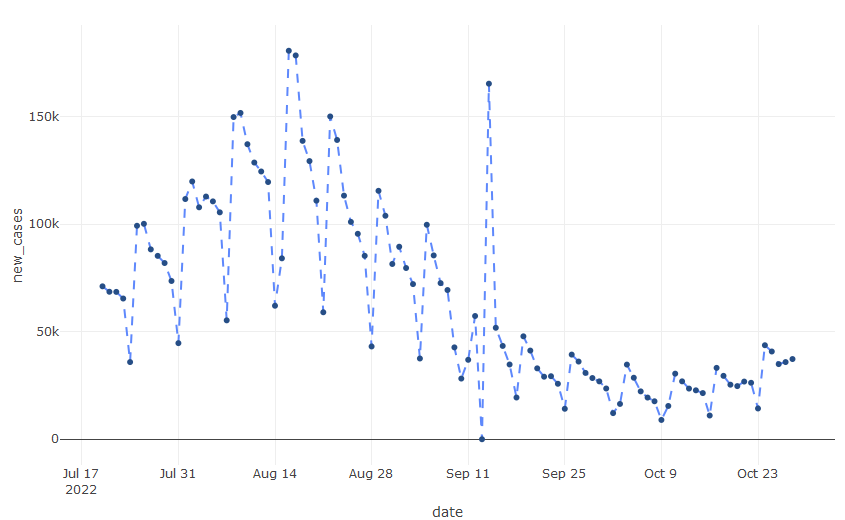
## ‘data’ 속성

plotly 트리에서 사용하는 첫 번째 레벨 속성인 ‘data’ 속성은 하위 속성들이 많지만 이 중 가장 중요한 것이 ‘trace’ 속성이다. ‘trace’ 속성은 plotly로 시각화할 수 있는 그래픽적 데이터 표현 방법을 지칭한다. plotly에서는 ‘scatter’, ‘pie’, ‘bar’ 등의 40개가 넘는 ’trace’를 제공하는데 이 ’trace’는 다음과 같은 몇 가지 특징을 가진다.

40여가지 ‘trace’ 중에 사용하고자 하는 ‘trace’를 설정하기 위해서 ’type’ 속성을 사용한다. 예를 들어 R과 python 모두 add\_trace()를 사용하여 시각화에 계속하여 ‘trace’를 추가할 수 있는데, add\_trace()를 사용하기 위해서는 반드시 ’type’ 속성으로 ‘trace’ 종류를 설정해야 한다. 하지만 ‘type’을 설정하지 않더라도 X축과 Y축에 바인딩된 변수들을 계산하여 자동적으로 설정해 주기도 각각의 ’trace’에 특화된 개별 함수를 사용한다면(예를 들어 add\_lines()나 px.line() 등) ’type’ 속성을 사용할 필요는 없다.

‘data’ 속성을 설정하기 위해서는 R과 python 모두 세 가지 방법이 제공된다. 첫 번쨰 방법은 plolty 객체를 초기화하는 함수에 직접 설정하는 방법이다. R에서는 plotly패키지의 plot\_ly()를 사용한다. plot\_ly()의 매개 변수로 세부 속성들을 설정할 수 있는데 세부 속성에 다시 하위 세부 속성들이 포함될 경우 R의 기본 데이터 타입인 ‘list’를 사용하여 묶어서 설정할 수 있다. 다음의 코드는 plot\_ly()를 사용하여 ’data’ 속성의 하부 속성인 ‘trace’를 ’scatter’, ‘mode’를 ’markers+lines’(산점도와 선그래프), ‘marker’(점) 속성을 list로 묶어 ‘color’ 속성을 설정하고 ‘line’(선) 속성을 list로 묶어 ‘color’와 ’dash’ 속성을 설정한 시각화를 보이고 있다.

## 긴 형태의 100일 코로나19 데이터에서  
df\_covid19\_100 |>   
 ## 한국 데이터만을 필터링   
 filter(iso\_code == 'KOR') |>  
 ## X축에 data, Y축에 new\_cases를 매핑하여 plot\_ly()로 시각화 생성  
 plot\_ly(type = 'scatter', x = ~date, y = ~new\_cases,   
 mode = 'markers+lines',   
 marker = list(color = '#264E86'),   
 line = list(color = '#5E88FC',   
 dash = 'dash'  
 )  
 )



실행결과 2- . plot\_ly를 사용한 plotly 시각화

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

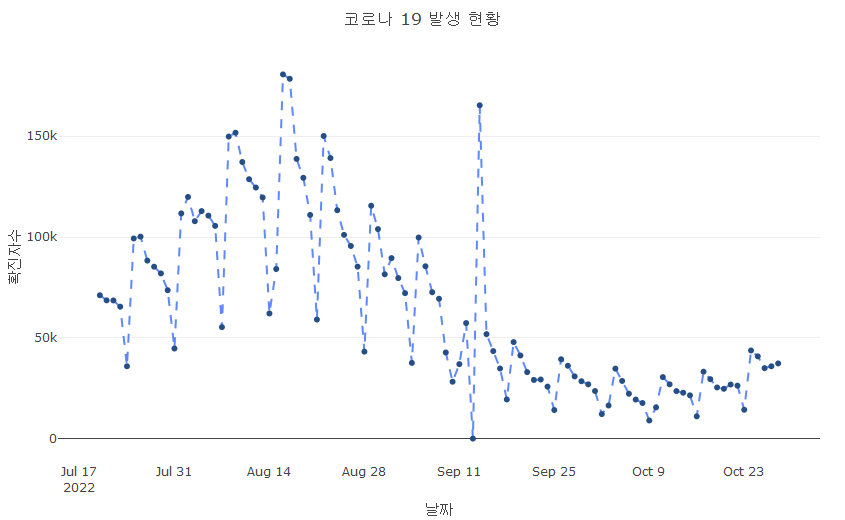
import plotly.graph\_objects as go  
  
go.Figure(  
 data = [  
 {  
 'type' : 'scatter',  
 'mode' : 'markers+lines',  
 'x' : df\_covid19\_100.loc[df\_covid19\_100['iso\_code'] == 'KOR', 'date'],  
 'y' : df\_covid19\_100.loc[df\_covid19\_100['iso\_code'] == 'KOR', 'new\_cases'],   
 'marker' : {  
 'color' : '#264E86'  
 },   
 'line' : {  
 'color' : '#5E88FC',  
 'dash' : 'dash'  
 }  
 }  
 ]  
)

## ‘layout’ 속성

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

R에서 ‘layout’ 속성을 설정하기 위해서는 layout()을 사용한다. 다만 layout()을 사용하기 위해서는 최소한 하나 이상의 ‘trace’가 설정되어야 한다. 다음의 코드는 앞서 그린 시각화에 ’title’ 속성으로 시각화 제목, ‘xaxis’ 속성으로 X축 설정, ‘yaxis’ 속성으로 Y축 설정, ‘margin’ 속성으로 여백 설정을 하고 있다. 이 중 ’xaxis’와 ’yaxis’속성은 세부 속성이 있기 때문에 다시 list로 묶어서 설정하였다.

margins\_R <- list(t = 50, b = 25, l = 25, r = 25)  
  
## 긴 형태의 100일 코로나19 데이터에서  
df\_covid19\_100 |>   
 ## 한국 데이터만을 필터링   
 filter(iso\_code == 'KOR') |>  
 ## X축에 data, Y축에 new\_cases를 매핑하여 plot\_ly()로 시각화 생성  
 plot\_ly(type = 'scatter', x = ~date, y = ~new\_cases,   
 mode = 'markers+lines',   
 marker = list(color = '#264E86'),   
 line = list(color = '#5E88FC',   
 dash = 'dash'  
 )  
 ) |>  
 layout(  
 title = "코로나 19 발생 현황",  
 xaxis = list(  
 title = "날짜",  
 showgrid = F),  
 yaxis = list(title = "확진자수"),   
 margin = margins\_R  
 )



python에서 ‘layout’ 속성의 설정은 ‘data’ 속성과 같이 Figure()에 속성들로 구성된 딕셔너리를 ’layout’에 설정한다.

margins\_python = {'t' : 50, 'b' : 25, 'l' : 25, 'r' : 25}  
  
fig = go.Figure(  
 data = [  
 {  
 'type' : 'scatter',  
 'mode' : 'markers+lines',  
 'x' : df\_covid19\_100.loc[df\_covid19\_100['iso\_code'] == 'KOR', 'date'],  
 'y' : df\_covid19\_100.loc[df\_covid19\_100['iso\_code'] == 'KOR', 'new\_cases'],   
 'marker' : {  
 'color' : '#264E86'  
 },   
 'line' : {  
 'color' : '#5E88FC',  
 'dash' : 'dash'  
 }  
 }  
 ],   
 layout = {  
 'title' : "코로나 19 발생 현황",  
 'xaxis' : {  
 'title' : "날짜",  
 'showgrid' : False},  
 'yaxis' : {'title' : "확진자수"},   
 'margin' : margins\_python  
 }  
)  
  
fig.show()

## ‘frame’ 속성

‘frame’ 속성은 plotly의 애니메이션 기능과 관련된 속성 값을 설정하는 노드이다. 이 책에서는 다루지 않겠다.

## ‘plotly’ 구조 확인

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

R에서 plotly의 구조를 확인하기 위해서는 plotly\_json()을 사용한다.

df\_covid19\_100 |>   
 ## 한국 데이터만을 필터링   
 filter(iso\_code == 'KOR') |>  
 ## X축에 data, Y축에 new\_cases를 매핑하여 plot\_ly()로 시각화 생성  
 plot\_ly(type = 'scatter', x = ~date, y = ~new\_cases,   
 mode = 'markers+lines',   
 marker = list(color = '#264E86'),   
 line = list(color = '#5E88FC',   
 dash = 'dash'  
 )  
 ) |>  
 plotly\_json()

## {  
## "visdat": {  
## "25a434ed55": ["function () ", "plotlyVisDat"]  
## },  
## "cur\_data": "25a434ed55",  
## "attrs": {  
## "25a434ed55": {  
## "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": ["2022-07-20", "2022-07-21", "2022-07-22", "2022-07-23", "2022-07-24", "2022-07-25", "2022-07-26", "2022-07-27", "2022-07-28", "2022-07-29", "2022-07-30", "2022-07-31", "2022-08-01", "2022-08-02", "2022-08-03", "2022-08-04", "2022-08-05", "2022-08-06", "2022-08-07", "2022-08-08", "2022-08-09", "2022-08-10", "2022-08-11", "2022-08-12", "2022-08-13", "2022-08-14", "2022-08-15", "2022-08-16", "2022-08-17", "2022-08-18", "2022-08-19", "2022-08-20", "2022-08-21", "2022-08-22", "2022-08-23", "2022-08-24", "2022-08-25", "2022-08-26", "2022-08-27", "2022-08-28", "2022-08-29", "2022-08-30", "2022-08-31", "2022-09-01", "2022-09-02", "2022-09-03", "2022-09-04", "2022-09-05", "2022-09-06", "2022-09-07", "2022-09-08", "2022-09-09", "2022-09-10", "2022-09-11", "2022-09-12", "2022-09-13", "2022-09-14", "2022-09-15", "2022-09-16", "2022-09-17", "2022-09-18", "2022-09-19", "2022-09-20", "2022-09-21", "2022-09-22", "2022-09-23", "2022-09-24", "2022-09-25", "2022-09-26", "2022-09-27", "2022-09-28", "2022-09-29", "2022-09-30", "2022-10-01", "2022-10-02", "2022-10-03", "2022-10-04", "2022-10-05", "2022-10-06", "2022-10-07", "2022-10-08", "2022-10-09", "2022-10-10", "2022-10-11", "2022-10-12", "2022-10-13", "2022-10-14", "2022-10-15", "2022-10-16", "2022-10-17", "2022-10-18", "2022-10-19", "2022-10-20", "2022-10-21", "2022-10-22", "2022-10-23", "2022-10-24", "2022-10-25", "2022-10-26", "2022-10-27", "2022-10-28"],  
## "y": [71109, 68579, 68521, 65433, 35883, 99268, 100182, 88296, 85275, 81944, 73589, 44689, 111700, 119866, 107849, 112857, 110610, 105507, 55292, 149819, 151734, 137196, 128671, 124515, 119603, 62078, 84128, 180652, 178480, 138741, 129350, 110944, 59046, 150098, 139165, 113281, 101064, 95538, 85295, 43142, 115519, 103919, 81499, 89528, 79623, 72144, 37548, 99737, 85484, 72599, 69389, 42724, 28214, 36938, 57309, 0, 165336, 51832, 43400, 34764, 19407, 47864, 41231, 32972, 29081, 29315, 25792, 14168, 39367, 36126, 30846, 28466, 26913, 23597, 12150, 16423, 34710, 28603, 22259, 19379, 17654, 8981, 15476, 30503, 26928, 23562, 22757, 21469, 11040, 33190, 29482, 25369, 24709, 26823, 26256, 14302, 43714, 40805, 34950, 35887, 37296],  
## "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"  
## }

python에서는 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 시각화 만들기

앞서 설명한 바와 같이 plotly 객체의 구조를 초기화하는 plot\_ly()나 Figure()를 사용해서 plotly 시각화를 그릴수 있다. 하지만 이 방법을 사용하면 코드의 길이가 길어지고 리스트와 딕셔너리 등 plotly 구조에 맞게 괄호들을 설정해야 하기 때문에 매우 복잡해진다.

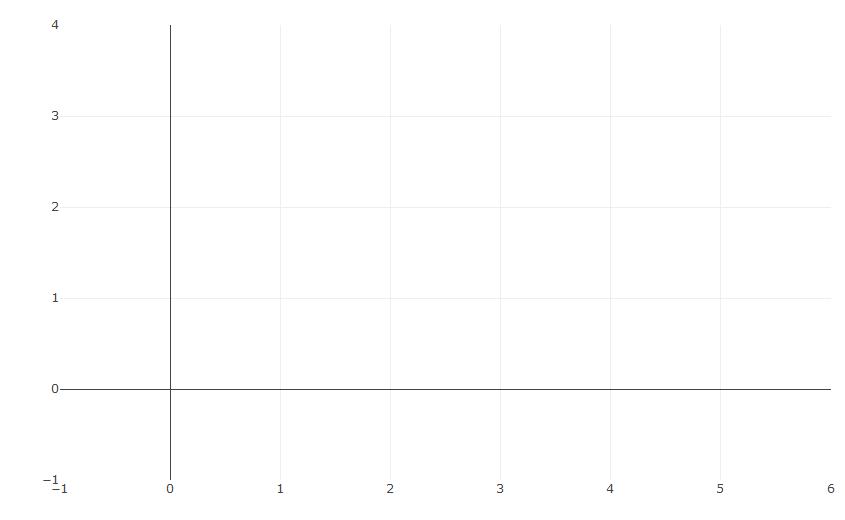
따라서 이 방법 보다는 ‘trace’를 추가하는 add\_trace()나 ’trace’에 특화된 자체 함수를 사용하는 방법이 많이 사용된다. 이 두 가지 방법중에 어떤 방법을 사용하더라도 같은 시각화가 그려진다. 여기서 중요한 것은 두 함수에 설정하는 시각화의 다양한 속성들이다. 이 속성은 앞에서 언급한데로 ’data’ 속성에 해당하는 ‘trace’ 관련한 속성들과 ’layout’과 관련한 속성들로 나뉜다.

## plotly 초기화

add\_trace()와 ‘trace’ 특화 함수를 사용하기 위해서는 먼저 plotly 객체를 초기화하여야 한다. R에서는 plot\_ly(), python에서는 plotly.graphic\_object의 객체에 Figure()를 사용한다.

R에서는 plotly의 초기화 단계에서 사용할 데이터프레임을 바인딩할 수 있다. tidyverse에서 제공하는 %>%나 R base에서 제공하는 |>를 사용하여 plot\_ly()에 사용할 데이터프레임을 설정하면 이 후 추가되는 ’trace’들에 모두 같은 데이터프레임을 사용할 수 있다.

## R에서 plotly 객체 초기화  
df\_covid19\_100 |>   
 plot\_ly()



실행결과 2- . plot\_ly()를 사용한 plotly 초기화

python에서는 plotly.graph\_objects 라이브러리 객체의 Figure()를 사용하여 초기화한다.

## python에서 plotly 객체 초기화  
go.Figure()

## trace의 공통 속성

이번 절에서는 ’trace’를 사용할 때 공통적으로 사용하는 속성들에 대해 설명한다. 실습을 위한 코드는 add\_trace()를 사용하도록 하겠다.

### type

‘trace’ 설정에 가장 중요한 속성이 ‘type’ 속성이다. ‘type’ 속성은 데이터를 표현할 방식을 설정하기 때문에 시각화의 가장 중요한 형태를 결정하는 속성이다. plotly에서 지원하는 주요 ’trace’는 다음과 같다.

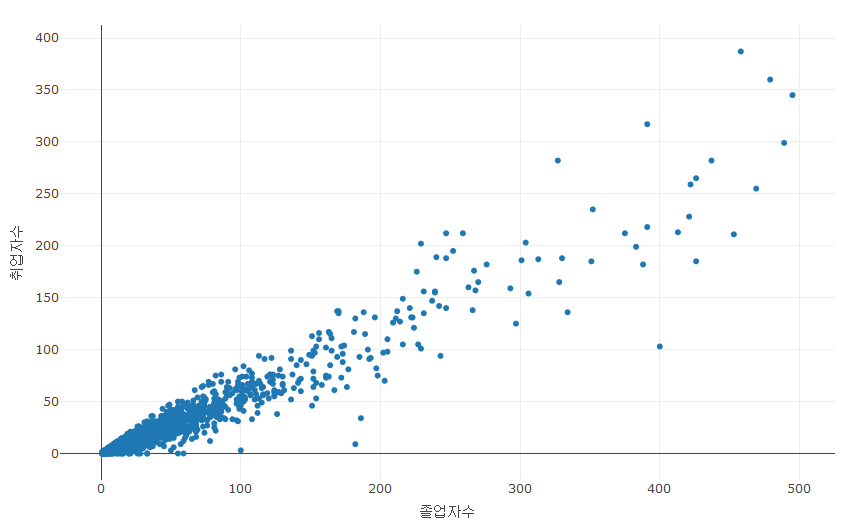
* scatter 타입 : ‘scatter’, ‘scattergl’
* bar 타입 : ‘bar’, ‘funnel’, ‘waterfall’
* 집계된 bar 타입 : ‘histogram’
* 1차원 분포 타입 : ‘box’, ‘violin’
* 2차원 밀도 분포 타입 : ‘histogram2d’, ‘histogram2dcontour’
* 매트릭스 타입 : ‘image’, ‘heatmap’, ‘contour’
* 주가 타입 : ‘olhc’, ‘candlestick’

### x, y

‘trace’ 설정에서 2차원 데카르트 좌표축에 데이터를 매핑하는 속성이 x, y이다.

R에서는 x, y에 변수를 매핑할 때는 반드시 ~를 변수명 앞에 붙여줘야한다. 매핑하는 변수로 plot\_ly() 설정에서 매핑한 데이터프레임의 열을 설정할수도 있고 벡터를 사용할 수도 있다.

## df\_취업률\_2000에서   
df\_취업률\_2000 |>   
 ## X축은 졸업자수, Y축은 취업자수로 매핑한 plotly 객체 생성  
 plot\_ly() |>  
 add\_trace(type = 'scatter', mode = 'markers',   
 x = ~졸업자수, y = ~취업자수)



실행결과 2- . plotly 기본 산점도 생성

python에서는 ‘x’, ‘y’ 속성에 데이터프레임 열을 설정하거나 데이터프레임을 인덱싱한 벡터를 설정한다.

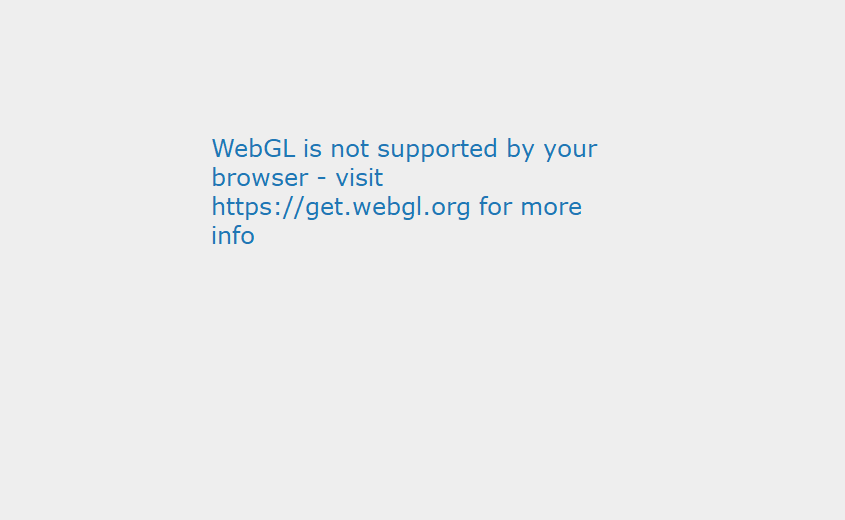
fig = go.Figure()  
  
fig.add\_trace(  
 {'type' : 'scatter',  
 'mode' : 'markers',  
 'x': df\_취업률\_2000['졸업자수'],  
 'y': df\_취업률\_2000['취업자수']  
 }  
)  
fig.show()

### z

plotly는 3차원 그래픽을 지원하기 때문에 3차원 데카르트 좌표계를 설정할 수 있다. 3차원 그래픽 시각화를 사용하기 위해서는 3차원 시각화를 지원하는 ‘trace’를 사용하여야하고 ’z’ 속성을 사용하여 3차원 좌표축을 설정하여야 한다. 동적 시각화는 사용자가 시각화를 자신이 원하는 방향으로 설정하여 관찰할 수 있기 때문에 3차원 효과가 효율적일 수 있다. 하지만 일반적으로 데이터 시각화에서는 3차원 을 사용하는 것은 크게 효과적이지 않다고 알려져 있기 때문에 3차원의 활용은 주의할 필요가 있다.

R에서는 ‘x’, ‘y’의 설정과 동일한 방법으로 ’z’ 속성을 설정할 수 있다.

df\_취업률\_2000 |>  
 ## X축은 졸업자수, Y축은 취업자수, Z축은 대계열로 매핑된 plotly 객체 생성  
 plot\_ly() |>  
 add\_trace(type = 'scatter3d', mode = 'markers',   
 x = ~졸업자수, y = ~취업자수, z = ~대계열)



실행결과 2- . 3차원 plotly

반면 python에서도 ‘x’, ‘y’의 설정과 유사한 방법으로 ’z’ 속성을 설정할 수 있다.

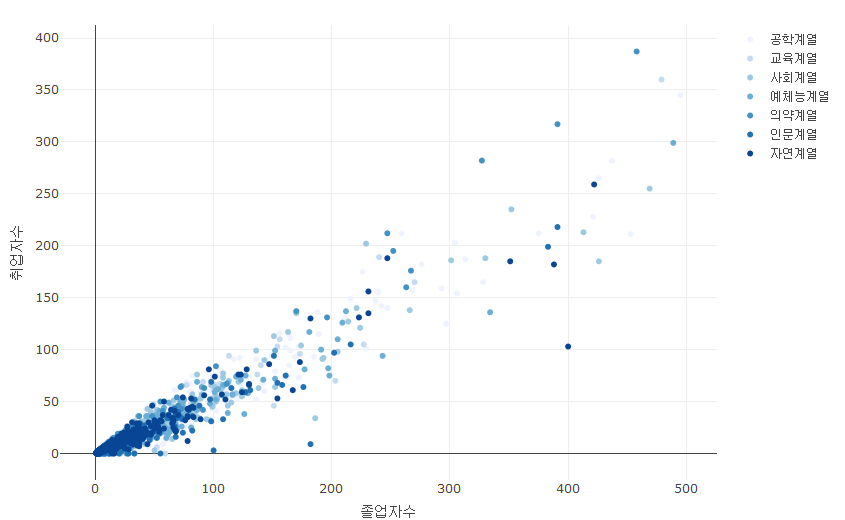
fig = go.Figure()  
  
fig.add\_trace(  
 {'type' : 'scatter3d',  
 'mode' : 'markers',  
 'x': df\_취업률\_2000['졸업자수'],  
 'y': df\_취업률\_2000['취업자수'],   
 'z': df\_취업률\_2000['대계열']  
 }  
)  
fig.show()

### name

’name’은 plotly에서 추가되는 각각의 ’trace’에 대한 이름을 설정한다. 이 이름은 범례 아이템과 마우스 포인터가 데이터 점에 위치할때 나타나는 호버(Hover)에 표기되는 이름으로도 사용된다.

R에서는 ‘name’ 속성에 ‘x’, ’y’과 같이 plotly에 바인딩된 데이터 프레임의 열을 사용할 수도 있고 벡터를 사용할 수도 있다. 다만 벡터를 사용한다면 plotly포 표시되는 데이터에 1:1 매핑이 가능한 벡터를 설정하여야 한다.

df\_취업률\_2000 |>  
 ## X축은 졸업자수, Y축은 취업자수, name은 대계열로 매핑한 plotly 객체 생성  
 plot\_ly() |>  
 add\_trace(type = 'scatter', mode = 'markers',   
 x = ~졸업자수, y = ~취업자수, name = ~대계열)



실행결과 2- . trace 이름 매핑

python에서 ‘name’ 속성을 사용하는데는 R보다는 다소 번거롭다. python에서 ’name’에 설정이 가능한 데이터 타입은 단일 string만을 설정할 수 있다. 따라서 여러 개의 ’trace’에 대한 루프를 사용해야 한다.

fig = go.Figure()  
  
for 대계열, group in df\_취업률\_2000.groupby('대계열'):  
 fig.add\_trace({  
 'type' : 'scatter',  
 'mode' : 'markers',  
 'x': group['졸업자수'],  
 'y': group['취업자수'],   
 'name' : 대계열  
 })  
fig.show()

### text, textposition, texttemplate, textfont

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

‘text’ 속성은 데이터로 표시되어야 하는 문자열을 매핑하거나 설정하는 속성이다. ’text’에 단일 문자열을 설정하면 모든 데이터에 설정된 문자열이 표시되고 문자열 벡터가 설정되면 문자열 벡터와 표현되는 데이터가 1:1로 매핑되어 해당 데이터에 매핑된 문자열이 표시된다.

## 긴 형태의 100일간 코로나19 데이터 중에  
df\_covid19\_100 |>  
 ## 국가명으로 그룹화  
 group\_by(location) |>  
 ## 확진자수의 합계를 new\_cases로 산출  
 summarise(new\_cases = sum(new\_cases)) |>  
 ## X축을 location, Y축과 text를 new\_case로 매핑  
 plot\_ly() |>  
 add\_trace(type = 'bar', x = ~location, y = ~new\_cases, text = ~new\_cases)

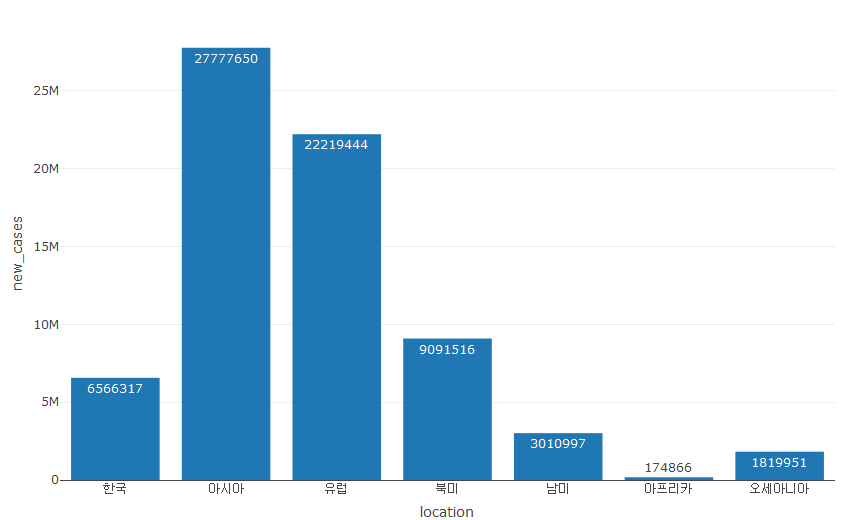
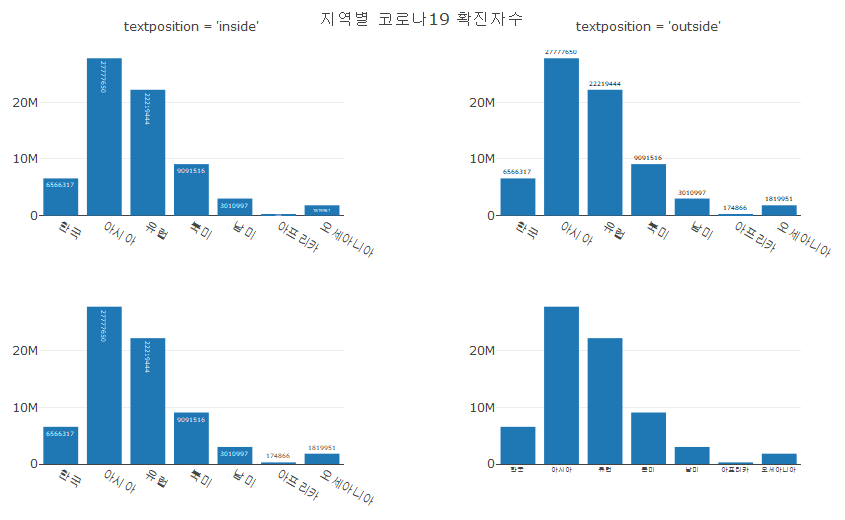


fig = go.Figure()  
  
temp = df\_covid19\_100.groupby('location').agg(new\_cases = ('new\_cases', 'sum'))  
  
fig.add\_trace({  
 'type' : 'bar',  
 'x': temp.index,  
 'y': temp['new\_cases'],  
 'text' : temp['new\_cases']  
 }  
 )  
  
fig.show()

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

## 긴 형태의 100일간 코로나19 데이터 중에  
df\_covid19\_100 |>  
 ## 국가명으로 그룹화  
 group\_by(location) |>  
 ## 확진자수의 합계를 new\_cases로 산출  
 summarise(new\_cases = sum(new\_cases)) |>  
 ## X축을 location, Y축과 text를 new\_case로 매핑  
 plot\_ly() |>  
 add\_trace(type = 'bar', x = ~location, y = ~new\_cases, text = ~new\_cases,   
 ## textposition을 '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,   
 ## textposition을 '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,   
 ## textposition을 '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,   
 ## textposition을 'none'으로 설정  
 textposition = 'none')

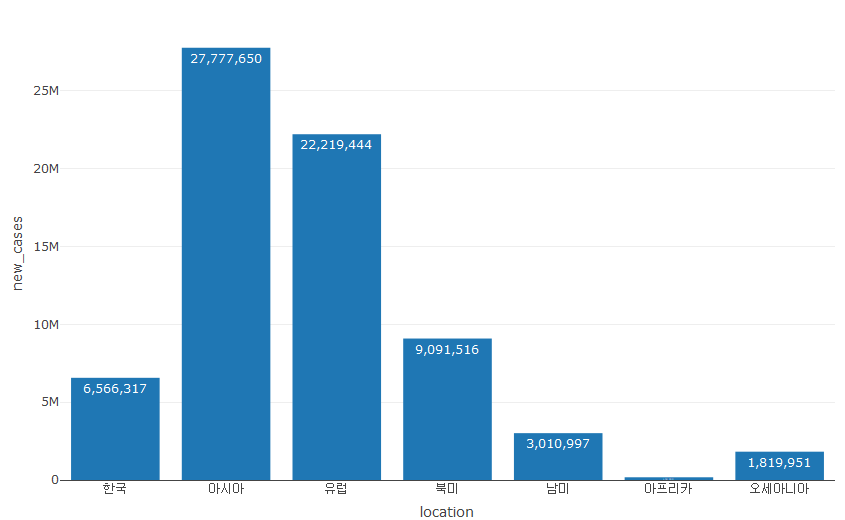
temp = df\_covid19\_100.groupby('location').agg(new\_cases = ('new\_cases', 'sum'))  
  
## textposition을 'inside'로 설정  
fig = go.Figure()  
  
fig.add\_trace({  
 'type' : 'bar',  
 'x': temp.index,  
 'y': temp['new\_cases'],  
 'text' : temp['new\_cases'],   
 'textposition' : 'inside'  
 }  
 )  
  
fig.show()  
  
## textposition을 'outside'로 설정  
fig = go.Figure()  
  
fig.add\_trace({  
 'type' : 'bar',  
 'x': temp.index,  
 'y': temp['new\_cases'],  
 'text' : temp['new\_cases'],   
 'textposition' : 'outside'  
 }  
 )  
  
fig.show()  
  
## textposition을 'auto'로 설정  
fig = go.Figure()  
  
fig.add\_trace({  
 'type' : 'bar',  
 'x': temp.index,  
 'y': temp['new\_cases'],  
 'text' : temp['new\_cases'],   
 'textposition' : 'auto'  
 }  
 )  
  
fig.show()  
  
## textposition을 'none'로 설정  
fig = go.Figure()  
  
fig.add\_trace({  
 'type' : 'bar',  
 'x': temp.index,  
 'y': temp['new\_cases'],  
 'text' : temp['new\_cases'],   
 'textposition' : 'none'  
 }  
 )  
  
fig.show()



실행결과 2- . textposition에 설정 결과

texttemplate은 텍스트가 표시되는 형태를 설정하는 type이다. texttemplate는 hovertemplate의 정의와 같이 사용되는데 ‘%{변수:변수포맷}’의 형태로 사용된다. 변수 포맷에서 사용하는 포맷은 자바 스크립트의 d3 format[[4]](#footnote-48)을 사용한다. 앞의 예에서 일반대학의 입학생수의 포맷에 천단위 콤마를 넣는다면’%{text:,}‘로 설정하고 소수점 아래 두째 자리까지 표기한다면’%{text:2f}’로 설정한다.

## 긴 형태의 100일간 코로나19 데이터 중에  
df\_covid19\_100 |>  
 ## 국가명으로 그룹화  
 group\_by(location) |>  
 ## 확진자수의 합계를 new\_cases로 산출  
 summarise(new\_cases = sum(new\_cases)) |>  
 ## X축을 location, Y축과 text를 new\_case로 매핑  
 plot\_ly() |>  
 add\_trace(type = 'bar', x = ~location, y = ~new\_cases, text = ~new\_cases,   
 ## textposition을 'inside'로 설정  
 textposition = 'inside',  
 ## texttemplate를 설정  
 texttemplate = '%{text:,}'  
 )



실행결과 2- . texttemplate 설정 결과

fig = go.Figure()  
  
fig.add\_trace({  
 'type' : 'bar',  
 'x': temp.index,  
 'y': temp['new\_cases'],  
 'text' : temp['new\_cases'],   
 'textposition' : 'inside',  
 ## texttemplate를 설정  
 'texttemplate' : '%{text:,}'  
 }  
 )  
  
fig.show()

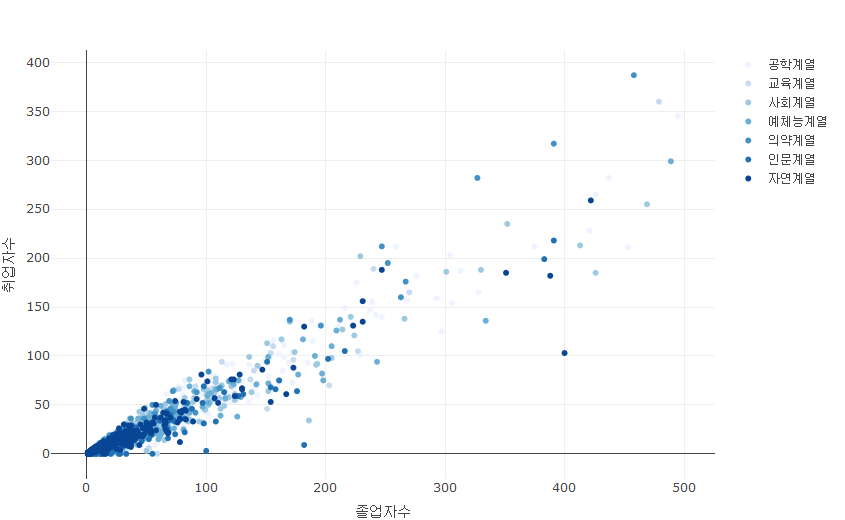
### hovertext, hoverinfo, hovermode, hovertemplete

plotly와 같은 동적 시각화에서는 대부분 마우스 포인터를 데이터가 표시된 점이나 선에 위치하면 해당 위치의 데이터가 표시된다. plotly에서는 이렇게 데이터의 정보를 표시하는 말풍선을 ’hover’라고 한다. ’hover’는 시각화를 설계하는 사용자에 따라 표시할 정보를 설정할 수 있는데 이 정보를 설정하는 속성이 hover\*이다.

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

df\_취업률\_2000 |>  
 plot\_ly() |>  
 add\_trace(type = 'scatter', mode = 'markers',   
 x = ~졸업자수, y = ~취업자수, name = ~대계열,  
 ## hovertext를 학과명으로 매핑  
 hovertext = ~학과명)

fig = go.Figure()  
  
for 대계열, group in df\_취업률\_2000.groupby('대계열'):  
 fig.add\_trace({  
 'type' : 'scatter',  
 'mode' : 'markers',  
 'x': group['졸업자수'],  
 'y': group['취업자수'],   
 'name' : 대계열,  
 'hovertext' : df\_취업률\_2000['학과명']  
 })  
fig.show()

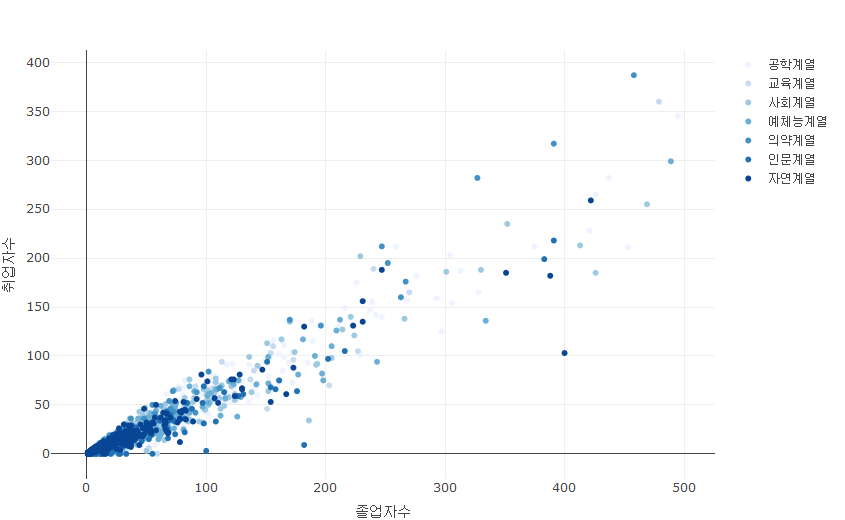


실행결과 2- . hovertext 매핑 결과

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

df\_취업률\_2000 |>  
 plot\_ly() |>  
 add\_trace(type = 'scatter', mode = 'markers',   
 x = ~졸업자수, y = ~취업자수, name = ~대계열,  
 hovertext = ~학과명, hoverinfo = 'x+y+text')

fig = go.Figure()  
  
for 대계열, group in df\_취업률\_2000.groupby('대계열'):  
 fig.add\_trace({  
 'type' : 'scatter',  
 'mode' : 'markers',  
 'x': group['졸업자수'],  
 'y': group['취업자수'],   
 'name' : 대계열,  
 'hovertext' : df\_취업률\_2000['학과명'],   
 'hoverinfo' : 'x+y+text'  
 })  
fig.show()

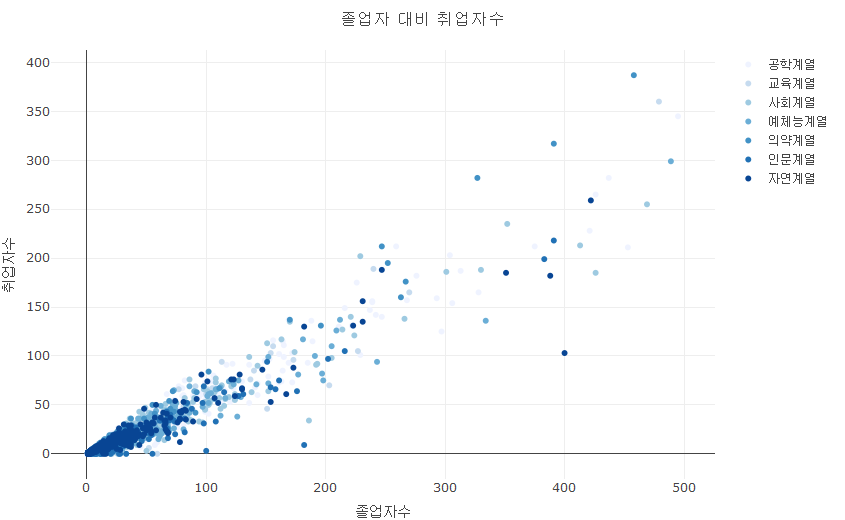


실행결과 2- . hoverinfo 설정 결과

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

df\_취업률\_2000 |>   
 plot\_ly() |>  
 add\_trace(type = 'scatter', mode = 'markers',   
 x = ~졸업자수, y = ~취업자수, name = ~대계열,  
 ## hoverinfo를 설정하고 hovertext를 중계열로 매핑  
 hoverinfo = 'x+y+text', hovertext = ~중계열)

fig = go.Figure()  
  
for 대계열, group in df\_취업률\_2000.groupby('대계열'):  
 fig.add\_trace({  
 'type' : 'scatter',  
 'mode' : 'markers',  
 'x': group['졸업자수'],  
 'y': group['취업자수'],   
 'name' : 대계열,  
 'hoverinfo' : 'x+y+text',   
 'hovertext' : df\_취업률\_2000['중계열']  
 })  
  
fig.show()

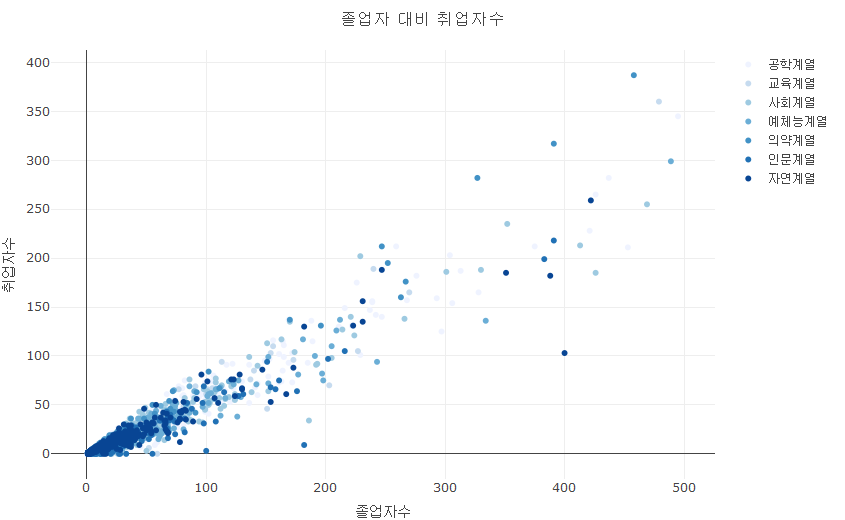


실행결과 2- . hoverinfo와 hovertext 동시 매핑 결과

다음의 결과는 ’hoverinfo’없이 ’hovertext’를 설정 했을때의 결과이다.

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

fig = go.Figure()  
  
for 대계열, group in df\_취업률\_2000.groupby('대계열'):  
 fig.add\_trace({  
 'type' : 'scatter',  
 'mode' : 'markers',  
 'x': group['졸업자수'],  
 'y': group['취업자수'],   
 'name' : 대계열,  
 'hovertext' : df\_취업률\_2000['중계열']  
 })  
  
fig.show()

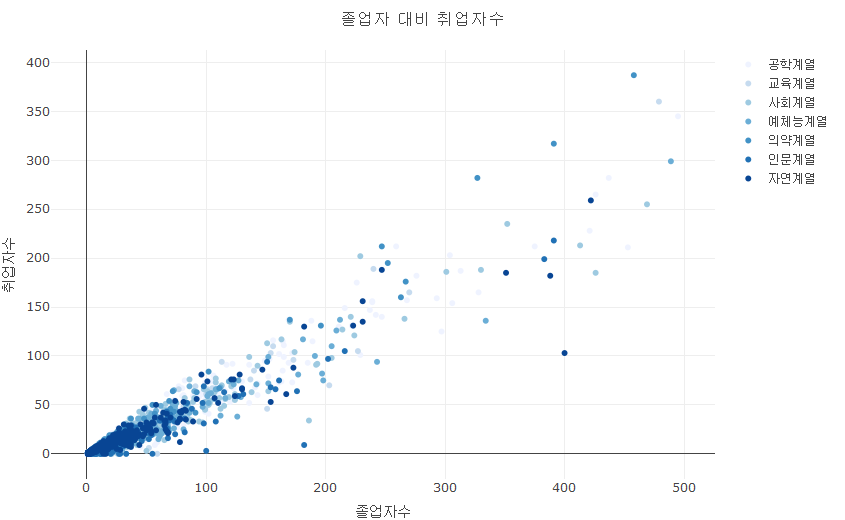


실행결과 2- . hoverinfo 없이 hovertext 매핑 결과

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

df\_취업률\_2000 |>   
 plot\_ly() |>  
 add\_trace(type = 'scatter', mode = 'markers',   
 x = ~졸업자수, y = ~취업자수, name = ~대계열,  
 hoverinfo = 'x+y+text', text = ~중계열,  
 ## hovertamplate의 설정  
 hovertemplate = ' 졸업자:%{x}, 취업자:%{y}, 중계열:%{text}')

fig = go.Figure()  
  
for 대계열, group in df\_취업률\_2000.groupby('대계열'):  
 fig.add\_trace({  
 'type' : 'scatter',  
 'mode' : 'markers',  
 'x': group['졸업자수'],  
 'y': group['취업자수'],   
 'name' : 대계열,  
 'hoverinfo' : 'x+y+text',   
 'text' : group['중계열'],  
 ## hovertamplate의 설정  
 'hovertemplate' : ' 졸업자:%{x}, 취업자:%{y}, 중계열:%{text}'  
 })  
  
fig.show()



실행결과 2- . hovertemplate 설정 결과

### color, colors

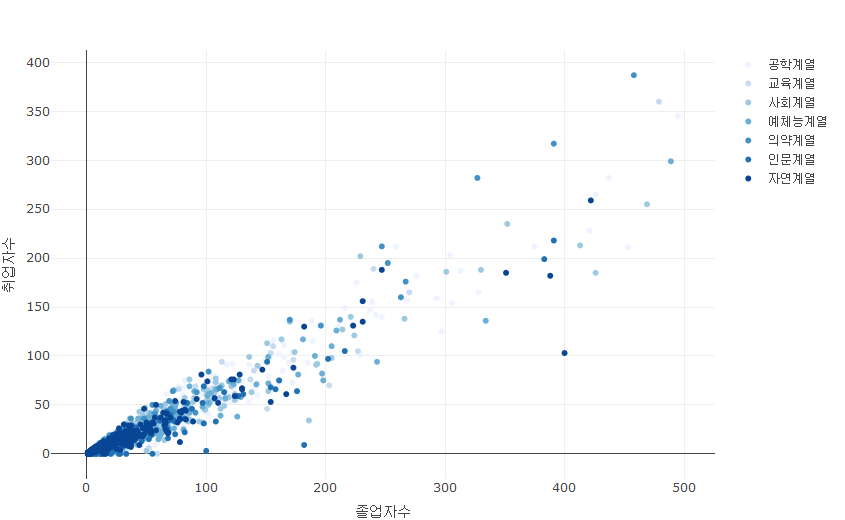
‘color’는 ’trace’에 표현되는 점, 선, 문자의 내부 색을 설정하는 속성이다. 내부 색을 설정할 때는 먼저 색을 변수에 매핑할지, 특정 색상으로 설정할 지를 결정해야 한다. 이를 설정하는 매개변수가 ’color’이다. 이 ’color’ 속성은 R과 python의 사용방법이 다소 다르다.

R에서는 add\_trace()의 속성 매개변수인 ’color’에 데이터프레임의 열을 설정하거나 표시되는 데이터에 1:1로 매핑된 벡터를 설정할 수 있다. ’color’에 설정된 데이터프레임 열이나 벡터는 반드시 컬러값일 필요는 없고 factor나 문자열도 가능하다. R에서는 이 factor나 문자열에 컬러값을 지정된 팔레트에 자동으로 매핑하여 색이 표현된다.

df\_취업률\_2000 |>   
 ## X, Y축의 매핑과 color를 대계열로 매핑  
 plot\_ly() |>  
 add\_trace(type = 'scatter', mode = 'markers',   
 x = ~졸업자수, y = ~취업자수, color = ~대계열)

하지만 python에서는 add\_trace()의 딕셔너리에 ‘color’ 속성이 존재하지 않는다. 따라서 ’color’를 설정하기 위해서는 데이터가 표현되는 도형에 관련된 속성의 하위속성인 ’color’를 설정하여야 한다. 산점도의 예를 보면 데이터를 표현하는 ’marker’의 하위 속성인 ’color’에 색을 설정하여야 한다. 또 하나 주의 해야할 것은 R에서는 카테고리 데이터 타입이나 문자열 데이터 타입의 경우 자동적으로 변환하여 색을 설정해주지만 python에서는 반드시 숫자형 변수만을 설정할 수 있다는 것이다. 따라서 문자열을 설정하는 경우는 다음과 같이 ’category’로 변환후 ’value’의 ’codes’를 매핑해 주어야 한다.

fig = go.Figure()  
  
fig.add\_trace(  
 {'type' : 'scatter',  
 'mode' : 'markers',  
 'x': df\_취업률\_2000['졸업자수'],  
 'y': df\_취업률\_2000['취업자수'],   
 'marker' : {'color' : df\_취업률\_2000["대계열"].astype('category').values.codes}  
 }  
)  
fig.show()



실행결과 2- . color 매핑 결과

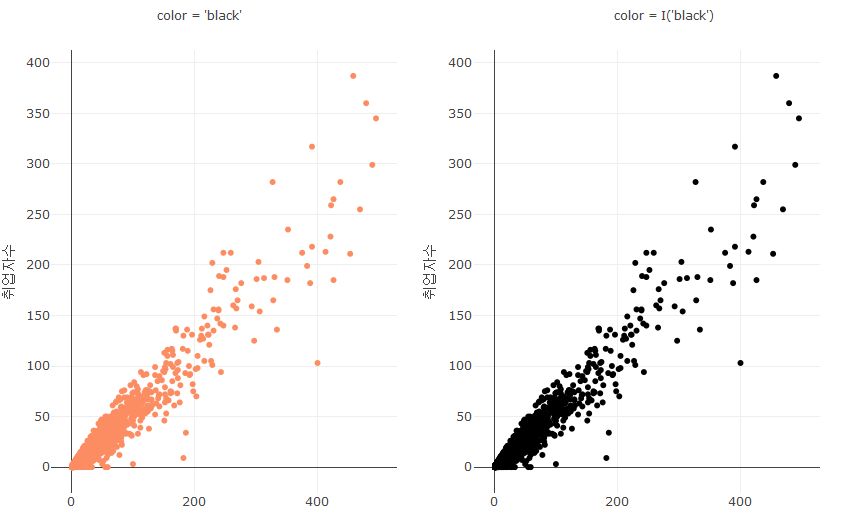
다음으로 R과 python이 차이나는 것은 변수로 ‘color’ 속성을 설정하는 것이 아닌 색상값을 직접 지정할 때이다. 앞의 변수 설정에는 python이 다소 번거로왔다면 직접 색상을 설정할 때는 R이 다소 번거롭다.

R에서 특정한 색으로 설정할 때 color에 특정 색 이름을 직접 설정하면 모든 marker가 지정된 색으로 나타나지 않는다. 다음과 같이 색상을 ’black’으로 설정했더라도 검정색으로 표현되지 않는다.

df\_취업률\_2000 |>   
 ## plot\_ly()에 color를 색 이름 설정  
 plot\_ly() |>  
 add\_trace(x = ~졸업자수, y = ~취업자수, color = 'black')

R에서 plotly에서 색의 사용은 기본적으로 변수의 매핑을 전제로 사용된다. 따라서 앞의 코드에서 color = 'black'으로 설정하는 것은 색을 검정색으로 지정하는 것이 아니고 ‘black’이라는 이름으로 매핑된 색 배열을 불러온다는 의미이다. 하지만 미리 매핑되어 정의된 ’black’ 색 배열이 없기 때문에 plotly의 디폴트 색 팔레트를 사용하여 색이 설정된다. 자신이 원하는 색을 직접 설정하기 위해서는 ’asis’를 의미하는 I()를 사용하여 색을 설정하여야 한다.

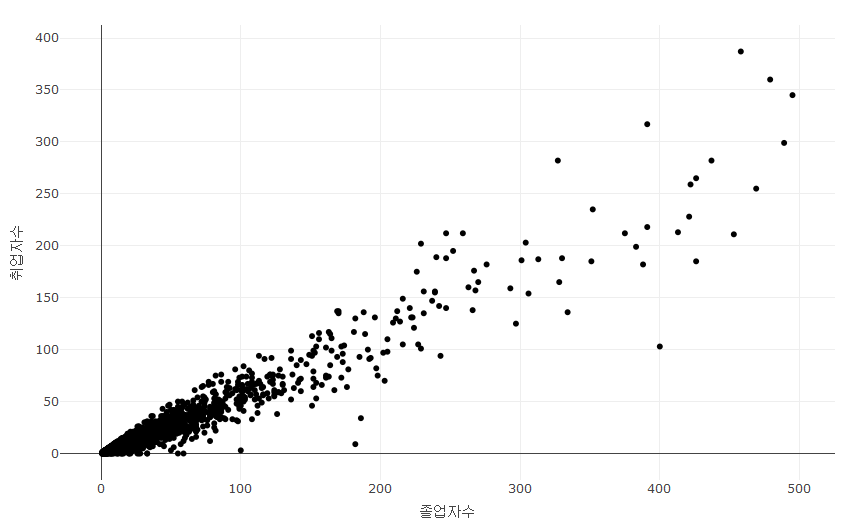
df\_취업률\_2000 |>   
 ## plot\_ly()에 color를 I()사용하여 색 이름 설정  
 plot\_ly() |>  
 add\_trace(x = ~졸업자수, y = ~취업자수, color = I('black'))



실행결과 2- . I()를 사용한 color 설정 결과

하지만 python에서는 간단히 색상명이나 색상 코드를 ‘color’ 속성에 설정함으로써 전체적 색을 설정할 수 있다.

fig = go.Figure()  
  
fig.add\_trace({  
 'type' : 'scatter',  
 'mode' : 'markers',  
 'x': df\_취업률\_2000['졸업자수'],  
 'y': df\_취업률\_2000['취업자수'],   
 'marker' : {'color' : 'black'}  
 })  
  
fig.show()

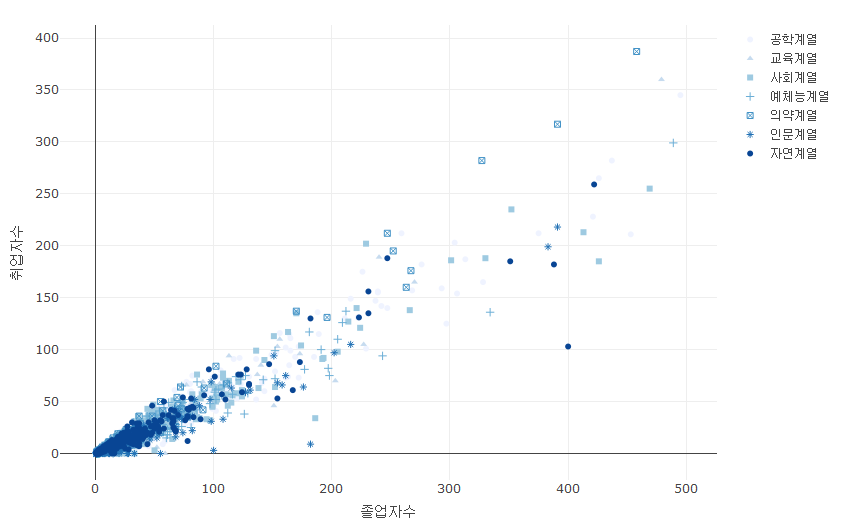


### symbol, symbols

symbol은 점의 형태를 설정하는 속성이다. 앞서 color와 같이 symbol도 매핑할 변수를 설정하는 속성이고 symbols는 변수 카테고리에 따라 설정하는 점의 형태를 설정하는 속성이다.

df\_취업률\_2000 |>   
 plot\_ly(x = ~졸업자수, y = ~취업자수,   
 ## symbol을 연속형 수치 변수에 매핑  
 symbol = ~대계열) |>  
 layout(title = '졸업자 대비 취업자수', margin = margins)

fig = go.Figure()  
  
df\_취업률\_2000['대계열\_level'] = df\_취업률\_2000["대계열"].astype('category').values.codes  
symbols = {'공학계열':'1', '교육계열':'2', '사회계열':'3', '예체능계열':'4', '의약계열':'5', '인문계열':'6', '자연계열':'7'}  
  
for 대계열, group in df\_취업률\_2000.groupby('대계열'):  
 fig.add\_trace({  
 'type' : 'scatter',  
 'mode' : 'markers',  
 'x': group['졸업자수'],  
 'y': group['취업자수'],   
 'name' : 대계열,   
 'marker' : {'symbol' : symbols[대계열]}  
 })  
fig.show()



실행결과 2- . symbol 매핑 결과

1. <https://github.com/owid/covid-19-data/blob/master/public/data/owid-covid-data.csv> [↑](#footnote-ref-30)
2. 필자의 블로그에 업로드된 데이터를 활용하면 책과 동일한 결과를 얻을 수 있다. [↑](#footnote-ref-32)
3. 해당 데이터는 교육통계 서비스 홈페이지<https://kess.kedi.re.kr/contents/dataset?itemCode=04&menuId=m_02_04_03_02&tabId=m3>에서 다운로드를 받거나 필자의 블로그(2stndard.tistory.com)에서 다운로드 받을 수 있다. [↑](#footnote-ref-34)
4. <https://github.com/d3/d3-format/tree/v1.4.5#d3-format> [↑](#footnote-ref-48)