1. 분포, 상관, 추세(trend)의 시각화

# 분포(Distribution)의 시각화

상관관계(Correlation)의 시각화는 X축과 Y축으로 표현된 데이터가 전반적으로 어떠한 관계를 가지는지를 표현한 시각화이다. 관계의 시각화는 단순히 X축 데이터의 분포에 따라 Y축 데이터의 분포가 어떻게 분포하는지를 위주로 표현되는데 관계의 시각화에 따라 나타나는 상관관계는 부가적인 통계 처리 정보가 보완되지 않는다면 단순 관계에 불과할 뿐 인과관계를 표현하는 것은 아니다. 상관관계의 시각화를 설명하기 위해 앞으로 졸업자수에 따른 취업자 수의 시각화를 볼 것이다. 당연히 졸업자가 많은 학과는 상대적으로 취업자 수도 많게 될 것이다. 하지만 이러한 시각화 결과 만으로 단순히 졸업자가 많은 과는 취업자도 많다는 결론에 도달해서는 안된다는 것이다. 이를 증명하기 위해 다양한 통계적 검증 과정이 필요하고 이 검증 과정을 통해 독립변수가 종속 변수를 변화시키는 원인이라는 결론에 도달해야 한다. 그렇다면 관계의 시각화는 필요없는 것일까? 그렇지 않다. 어떤 독립 변수가 어떤 종속 변수와 밀접하게 관계가 있는지를 먼저 관계의 시각화를 통해 확인하여야 통계적 검증을 수행할 수 있다. 그저 데이터가 존재한다는 이유만으로 모든 변수간의 인과 관계를 검증할 수는 없는 것이다. 과거 컴퓨터 시스템이 발달하지 못했던 떄는 여러가지 통계적 방법론을 사용하여 이 상관관계를 확인하였지만 1장에서 본 바와 같이 통계적 방법론이 동일해도 시각화의 결과 큰 관계가 없는 경우도 있다. 따라서 상관관계를 확인하기 위한 통계치와 시각화 결과를 반드시 같이 확인해야 한다.

상관관계는 양의 상관관계와 음의 상관관계로 나뉜다. 양의 상관관계는 독립변수(보통 X축 변수)가 증가할수록 종속변수(보통 Y축변수)가 증가하는 관계이고 음의 상관관계는 독립변수가 증가할수록 독립변수가 감소하는 관계를 말한다. 이러한 상관관계는 상관계수라는 수치로 얼마나 강한 상관관계를 가지는지를 표현한다. 보통 0.7이상의 상관계수는 매우 강한 상관관계가 있다고 간주된다.[[1]](#footnote-20)



그림 5-1 상관관계와 상관계수

데이터의 상관관계를 확인하기 위한 시각화를 살펴보기 전에 이번 장에서 사용할 데이터를 다음과 같이 준비한다.

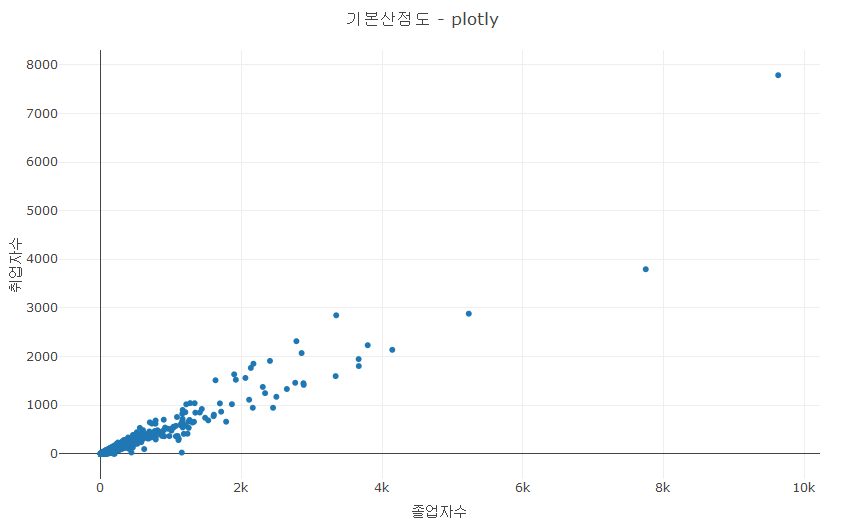
## 과정구분의 순서를 맞추기 위해 과정구분을 팩터로 설정하고 레벨의 순서를 설정  
df\_취업률$과정구분 <- fct\_relevel(df\_취업률$과정구분, '전문대학과정', '대학과정', '대학원과정')  
  
## 대계열의 순서를 맞추기 위해 대계열을 팩터로 설정하고 레벨의 순서를 설정  
df\_취업률$대계열 <- fct\_relevel(df\_취업률$대계열, '인문계열', '사회계열', '교육계열', '자연계열', '공학계열', '의약계열', '예체능계열')  
  
names(df\_취업률)[10:12] <- c('졸업자수', '취업률', '취업자수')

## 탐색적 데이터 분석의 시작은 산점도부터

산점도는 x, y 두개의 변수에 따른 데이터의 위치를 점으로 표현한 시각화 방법이다. 산점도를 통해 데이터의 전반적 분포와 X축의 독립변수에 따른 Y축의 종속변수가 어떻게 분포하는지를 한눈에 확인할 수 있고 그 분포 모양에 따라 상관관계와 상관강도를 알아볼 수 있다.

앞선 장에서 살펴본 바와 같이 산점도는 add\_markers()나 add\_trace()를 사용하여 쉽게 만들 수 있다.

df\_취업률 |>   
 plot\_ly() |>  
 add\_trace(type = 'scatter', type = 'markers', x = ~졸업자수, y = ~취업자수) |>  
 layout(title = list(text = '기본산점도 - plotly'),   
 xaxis = list(title = '졸업자수'),   
 yaxis = list(title = '취업자수'),   
 margin = margins)



실행결과 5-기본 산점도

산점도를 만들때 주의해야 할 것이 산점도에 너무 많은 점이 한곳에 표현되는 오버플로팅(Over-Ploting)이 일어나지 않도록 해야한다는 점이다. 위의 산점도를 보면 다행이 선형 상관관계가 눈에 보이지만 좌측 하단 구간은 데이터가 집중되다보니 검정색 지역만 눈에 보인다. 그 지역에서도 데이터가 특히 밀집된 지역이 존재하겠지만 데이터가 오버플로팅되어 이러한 데이터의 밀집을 전혀 확인할 수 없다. 이런 오버플로팅을 해결하는 방법은 다음과 같다.

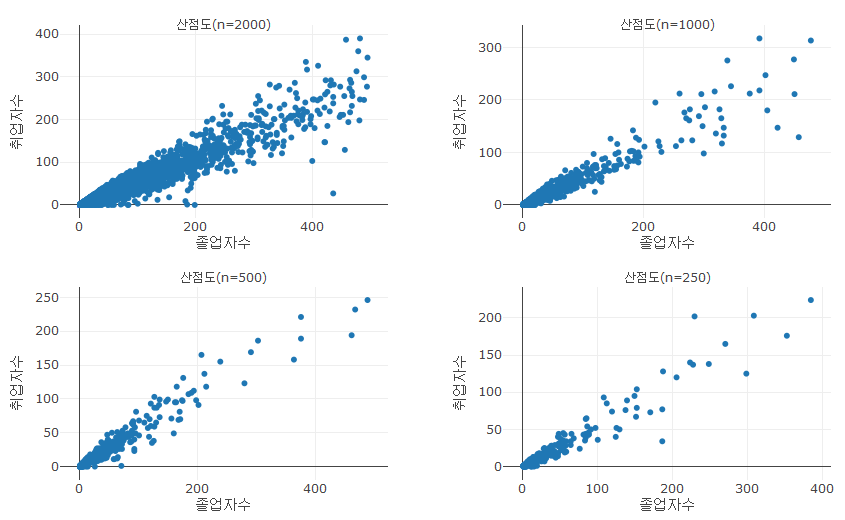
### overplotting의 제거 - 데이터 사이즈의 축소

플로팅되는 데이터를 랜덤 샘플링과 같은 방법을 통해 축소시키는 방법이다. 다음은 오버플로팅 된 데이터를 랜덤 샘플링을 통해 2000, 1000, 500개로 축소한 산점도이다.

## 랜덤 샘플링을 위한 난수 설정  
set.seed(123)  
  
## df\_취업률에서 졸업자가 500명 이하인 학과 2000개 샘플링  
df\_취업률\_2000 <- df\_취업률 |>   
 filter(졸업자수 < 500)  
  
## df\_취업률에서 졸업자가 500명 이하인 학과 2000개 샘플링  
df\_취업률\_1000 <- df\_취업률 |>   
 filter(졸업자수 < 500) |>   
 sample\_n(1000)  
  
## df\_취업률에서 일부 데이터를 필터링하고 2000개 데이터 샘플링  
df\_취업률\_500 <- df\_취업률 |>   
 filter(졸업자수 < 500) |>   
 sample\_n(500)  
  
## df\_취업률에서 일부 데이터를 필터링하고 1000개 데이터 샘플링  
df\_취업률\_250 <- df\_취업률 |>   
 filter(졸업자수 < 500) |>   
 sample\_n(250)

plotly를 사용하여 데이터를 축소를 통한 오버플로팅 제거는 다음과 같다.

df\_취업률\_2000 |>   
 plot\_ly() |>  
 add\_markers(x = ~졸업자수, y = ~취업자수) |>   
 layout(title = '산점도(n=2000)',  
 margin = margins  
 )  
  
df\_취업률\_1000 |>   
 plot\_ly() |>  
 add\_markers(x = ~졸업자수, y = ~취업자수) |>   
 layout(title = '산점도(n=1000)',  
 margin = margins  
 )  
  
df\_취업률\_500 |>   
 plot\_ly() |>  
 add\_markers(x = ~졸업자수, y = ~취업자수) |>   
 layout(title = '산점도(n=500)',  
 margin = margins  
 )  
  
df\_취업률\_250 |>   
 plot\_ly() |>  
 add\_markers(x = ~졸업자수, y = ~취업자수) |>   
 layout(title = '산점도(n=250)',  
 margin = margins  
 )



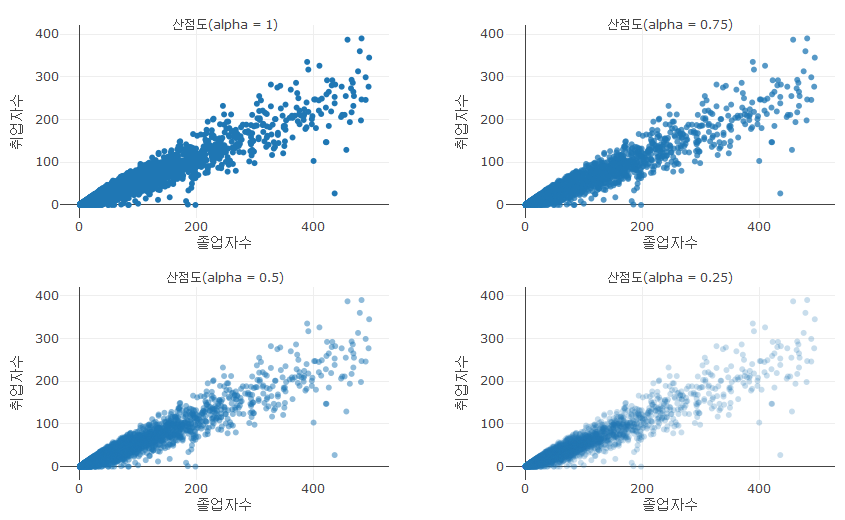
실행결과 5- 샘플수 조절들을 통한 오버플로팅 제거 - plotly

### overplotting의 제거 - 표시점의 투명도(alpha) 조절

위의 실행결과를 보면 샘플수 조절만으로는 오버플로팅이 완전히 해결되지는 않아 보인다. 이때 사용할 수 있는 또 하나의 방법이 표시점의 투명도를 조절하는 방법이다. 샘플수 축소와 투명도 조절을 잘 사용하면 효과적으로 오버플로팅을 제거할 수 있다.

plotly를 사용하여 점의 투명도 조절를 통한 오버플로팅 제거는 다음과 같다.

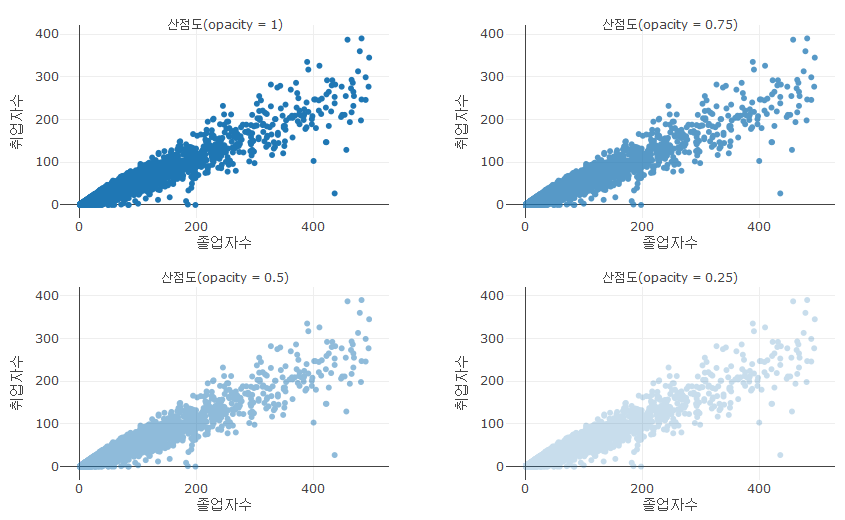
df\_취업률\_2000 |>   
 plot\_ly() |>  
 add\_markers(x = ~졸업자수, y = ~취업자수, alpha = 1) |>   
 layout(title = '산점도(alpha = 1)',  
 margin = margins  
 )  
  
df\_취업률\_2000 |>   
 plot\_ly() |>  
 add\_markers(x = ~졸업자수, y = ~취업자수, alpha = 0.75) |>   
 layout(title = '산점도(alpha = 0.75)',  
 margin = margins  
 )  
  
df\_취업률\_2000 |>   
 plot\_ly() |>  
 add\_markers(x = ~졸업자수, y = ~취업자수, alpha = 0.5) |>   
 layout(title = '산점도(alpha = 0.5)',  
 margin = margins  
 )  
  
df\_취업률\_2000 |>   
 plot\_ly() |>  
 add\_markers(x = ~졸업자수, y = ~취업자수, alpha = 0.25) |>   
 layout(title = '산점도(alpha = 0.25)',  
 margin = margins  
 )



실행결과 5- 투명도 조절을 통한 오버플로팅 제거 - plotly

plotly에서는 alpha 속성과 함께 opacity를 사용하여 오버플로팅을 제거할 수 있다. opacity는 점들이 겹쳐질 때 겹쳐지는 부분의 투명도를 계속해서 유지하는 투명도 설정이다. 이 속성은 서로 겹치는 부분이 얼마나 겹쳐있는지를 알 수 없기 때문에 분포의 집중도를 알 수 없다는 단점이 있다.

df\_취업률\_2000 |>   
 plot\_ly() |>  
 add\_markers(x = ~졸업자수, y = ~취업자수, opacity = 1) |>   
 layout(title = '산점도(opacity = 1)',  
 xaxis = list(title = '졸업자수'),   
 xaxis = list(title = '취업자수'),   
 margin = margins  
 )  
  
df\_취업률\_2000 |>   
 plot\_ly() |>  
 add\_markers(x = ~졸업자수, y = ~취업자수, opacity = 0.75) |>   
 layout(title = '산점도(opacity = 0.75)',  
 xaxis = list(title = '졸업자수'),   
 xaxis = list(title = '취업자수'),   
 margin = margins  
 )  
  
df\_취업률\_2000 |>   
 plot\_ly() |>  
 add\_markers(x = ~졸업자수, y = ~취업자수, opacity = 0.5) |>   
 layout(title = '산점도(opacity = 0.5)',  
 xaxis = list(title = '졸업자수'),   
 xaxis = list(title = '취업자수'),   
 margin = margins  
 )  
  
df\_취업률\_2000 |>   
 plot\_ly() |>  
 add\_markers(x = ~졸업자수, y = ~취업자수, opacity = 0.25) |>   
 layout(title = '산점도(opacity = 0.25)',  
 xaxis = list(title = '졸업자수'),   
 xaxis = list(title = '취업자수'),   
 margin = margins  
 )



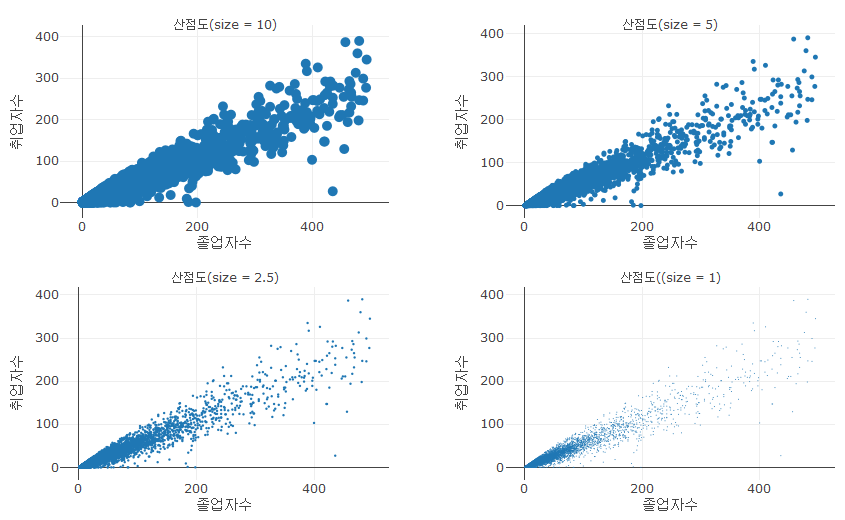
실행결과 5- opacity 조절을 통한 오버플로팅 제거 - plotly

### overplotting의 제거 - 표시점의 크기(size) 조절

산점도의 오버플로팅을 제거하는데 사용할 수 있는 또 하나의 방법은 표시점의 크기를 조절하는 것이다. 크기를 작게하면 단위 면적당 점을 더 많이 표시할 수 있기 때문에 오버플로팅을 다소 해소할 수 있다.

plotly를 사용하여 점의 크기 조절를 통한 오버플로팅 제거는 다음과 같다. ggplot2와 달리 plotly의 size는 0보다 큰 값이어야 한다.

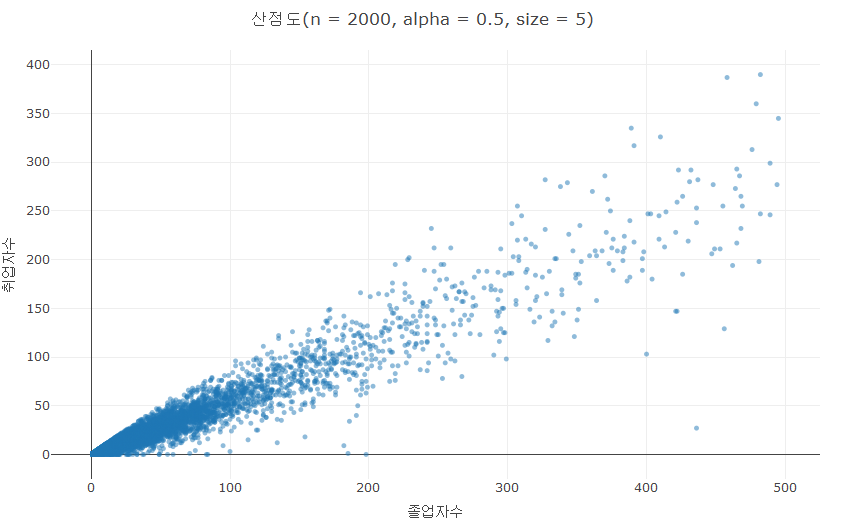
df\_취업률\_2000 |>   
 plot\_ly() |>  
 add\_markers(x = ~졸업자수, y = ~취업자수,   
 marker = list(size = 10)) |>   
 layout(title = '산점도(size = 10)',  
 margin = margins  
 )  
  
df\_취업률\_2000 |>   
 plot\_ly() |>  
 add\_markers(x = ~졸업자수, y = ~취업자수,   
 marker = list(size = 5)) |>   
 layout(title = '산점도(size = 5)',  
 margin = margins  
 )  
  
df\_취업률\_2000 |>   
 plot\_ly() |>  
 add\_markers(x = ~졸업자수, y = ~취업자수,   
 marker = list(size = 5)) |>   
 layout(title = '산점도(size = 2.5)',  
 margin = margins  
 )  
  
df\_취업률\_2000 |>   
 plot\_ly() |>  
 add\_markers(x = ~졸업자수, y = ~취업자수,   
 marker = list(size = 2)) |>   
 layout(title = '산점도(size = 1)',  
 margin = margins  
 )



실행결과 5-size 조절을 통한 오버플로팅 제거 - plotly

위의 세가지 방법을 모두 사용하여 오버플로팅을 다음과 같이 제거할 수 있다.

df\_취업률\_2000 |>   
 plot\_ly() |>  
 add\_markers(x = ~졸업자수, y = ~취업자수, alpha = 0.5,  
 marker = list(size = 5)) |>   
 layout(title = '산점도(n = 2000, alpha = 0.5, size = 5)',  
 xaxis = list(title = '졸업자수'),   
 xaxis = list(title = '취업자수'),   
 margin = margins  
 )



실행결과 5- 세가지 미적요소 조절을 통한 오버플로팅 제거 - plotly

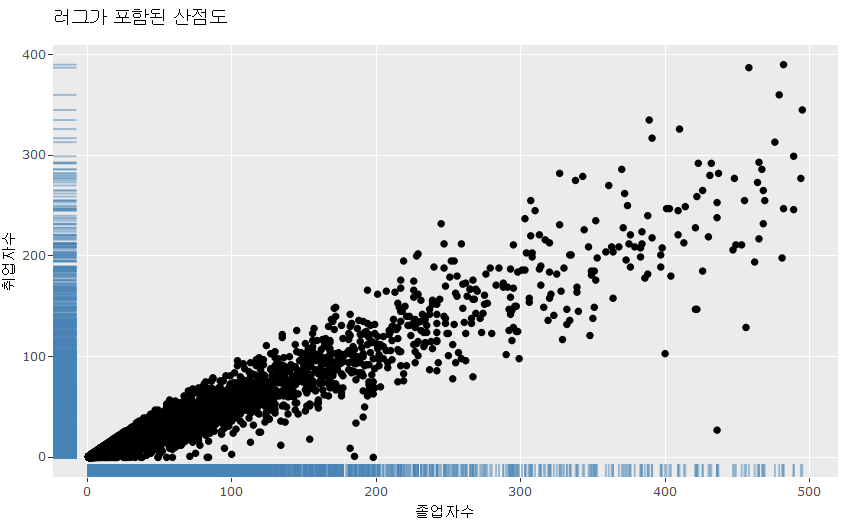
오버플로팅을 해결하는 방법으로 추가적으로 사용할 수 있는 방법은 밀도 분포 시각화를 사용하는 방법이 있다. 이는 다음 장에서 추가적으로 설명하겠다.

## 데이터가 밀집된 구간을 찾아보자 - Marginal plot

사실 산점도는 아무리 오버플로팅을 제거하려고 노력해도 산점도 자체가 가진 한계로 인해 완전히 제거하기란 매우 힘들다. 그래서 앞서 언급한 바와 같이 샘플수를 조절한다던지 투명도를 조절한다던지 하는 방법이 사용되지만 이로 충분치는 않다. 결국 중첩되어 표현된 부분에 얼마나 많은 데이터가 중첩되는지를 알아내는 방법이 필요한데 러그 산점도가 이에 대한 대안이 될 수 있다.

러그(rug)는 ’바닥에 까는 발판’으로 X축과 Y축의 바닥에 막대를 표현함으로써 데이터의 밀도를 표현하는 방법이다. 안타깝게도 plotly에서는 러그를 사용해 데이터의 밀도를 표시하는 방법을 제공하지 않는다. 따라서 ggplot2의 geom\_rug()를 사용하여 러그 산점도를 만들고 변환하는 방법을 사용할 수 있다.

ggplotly <- df\_취업률\_2000 |>  
 ggplot() +  
 ## X축을 졸업자\_계, Y축을 취업자\_합계\_계, color를 대계열로 매핑한 geom\_point로 산점도 레이어 생성  
 geom\_point(aes(x = 졸업자수, y = 취업자수)) +   
 ## X축을 졸업자\_계, Y축을 취업자\_합계\_계, color를 대계열로 매핑한 geom\_reg 레이처 추가  
 geom\_rug(aes(x = 졸업자수, y = 취업자수), col= "steelblue", alpha=0.5) +  
 labs(title = '러그가 포함된 산점도')  
  
ggplotly(ggplotly)

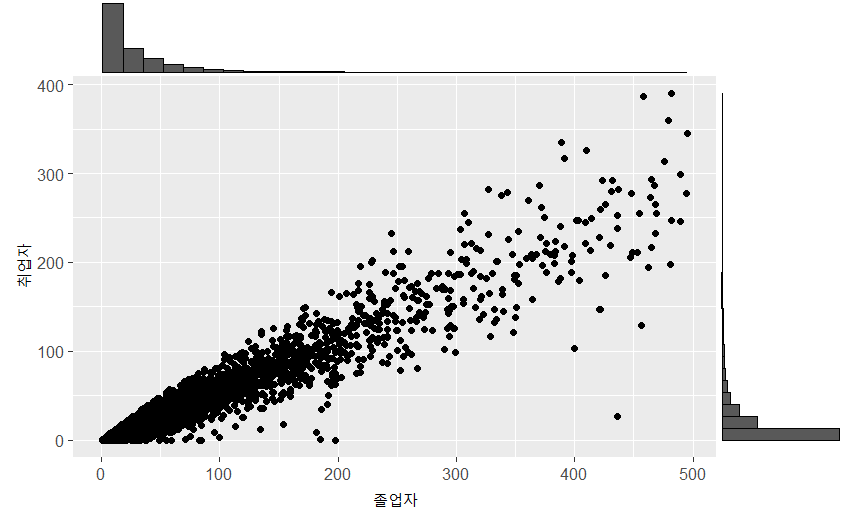


실행결과 5-ggplot2를 사용해 러그가 포함된 산점도

이렇게 데이터가 밀집된 구간을 확인할 수 있는 또 하나의 방법이 가장자리(Marginal) 플롯이다. 가장자리 플롯은 X축과 Y축의 가장자리에 히스토그램, 밀도 분포 플롯, 다음 절에서 설명할 박스플롯 등을 표현함으로서 전체 분포를 상관과 같이 보여준다.

이 가장자리 플롯은 ggplot2에서 직접적으로 사용하지는 못하고 ggExtra 패키지를 통해 사용할 수 있다. ggExtra 패키지는 ggplot2의 기능적 향상을 위해 개발된 패키지로 이 패키지의 가장 주된 기능은 ggMarginal이다. ggMarginal은 ggplot2 산점도의 X축과 Y축의 가장자리에 히스토그램, 박스플롯, 밀도 플롯을 추가시키는 함수이다.

### ggExtra 패키지 설치  
if(!require('ggExtra')) {  
 install.packages('ggExtra')  
 library(ggExtra)  
}  
  
## 산점도 생성  
p\_marginal <- df\_취업률\_2000 |>  
 ggplot() +  
 geom\_point(aes(x = 졸업자수, y = 취업자수)) +   
 labs(x = '졸업자', y = '취업자')  
  
## 가장자리 플롯으로 히스토그램을 추가  
ggMarginal(p\_marginal, x = 졸업자수, y = 취업자수, type = 'histogram')

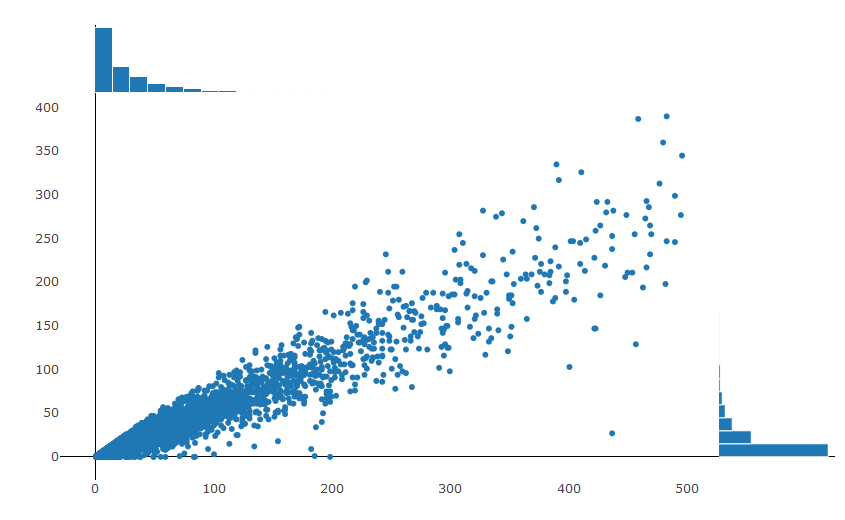


실행결과 5-타입별 가장자리 플롯

하지만 plotly에서는 러그 플롯과 가장자리 플롯을 직접적으로 지원하지 않는다. 하지만 가장자리 플롯과 유사한 시각화를 subplot()을 사용해서 만들 수 있다.

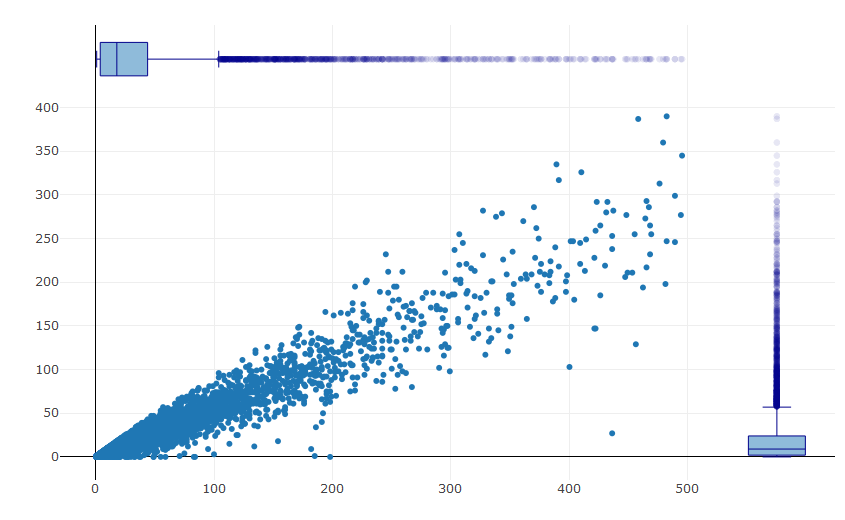
우선 가장자리 플롯을 만들기 위해서는 기본 산점도와 X축에 표현될 히스토그램, Y축에 표현될 히스토그램 3개의 plotly 객체가 필요하다. 이 세 개의 객체를 subplot()으로 위치와 크기를 적절히 설정하여 그려줌으로써 완성할 수 있다.

p1 <- df\_취업률\_2000 |>  
 plot\_ly() |>  
 add\_trace(type = 'scatter', mode = 'markers',   
 x = ~졸업자수, y = ~취업자수,   
 hovertext = ~학과명,  
 color = I('#1f77b4'), showlegend = FALSE) |>  
 layout(xaxis = list(zerolinecolor = 'black'),   
 yaxis = list(zerolinecolor = 'black'))  
  
p2 <- df\_취업률\_2000 |>  
 plot\_ly() |>  
 add\_trace(type = 'histogram', x = ~졸업자수, color = I('#1f77b4'),   
 hovertext = '졸업자 히스토그램',   
 xbins = list(size = 15), stroke = I('white'))  
  
p3 <- df\_취업률\_2000 |>  
 plot\_ly() |>  
 add\_trace(type = 'histogram', y = ~취업자수, color = I('#1f77b4'),   
 hovertext = '취업자 히스토그램',   
 ybins = list(size = 15), stroke = I('white'))  
  
subplot(  
 p2, plotly\_empty(), p1, p3,   
 nrows = 2, heights = c(0.15, 0.85),   
 widths = c(0.85, 0.15), titleX = FALSE, titleY = FALSE,   
 shareX = TRUE, shareY = TRUE, margin = 0  
) |> hide\_legend()



앞의 가장자리 플롯의 히스토그램 대신 박스 플롯을 사용하려면 다음과 같이 코딩할 수 있다.

p1 <- df\_취업률\_2000 |>  
 plot\_ly() |>  
 add\_trace(type = 'scatter', mode = 'markers',   
 x = ~졸업자수, y = ~취업자수,   
 hovertext = ~학과명,  
 color = I('#1f77b4'), showlegend = FALSE) |>  
 layout(xaxis = list(zerolinecolor = 'black'),   
 yaxis = list(zerolinecolor = 'black'))  
  
p2 <- df\_취업률\_2000 |>  
 plot\_ly() |>  
 add\_trace(type = 'box', x = ~졸업자수, color = I('#1f77b4'),   
 line = list(color = 'darkblue'),   
 marker = list(color = 'darkblue',   
 line = list(color = 'darkblue'),   
 opacity = 0.1),  
 hovertext = '졸업자 히스토그램',   
 xbins = list(size = 15), stroke = I('white'))  
  
p3 <- df\_취업률\_2000 |>  
 plot\_ly() |>  
 add\_trace(type = 'box', y = ~취업자수, color = I('#1f77b4'),   
 line = list(color = 'darkblue'),   
 marker = list(color = 'darkblue',   
 line = list(color = 'darkblue'),   
 opacity = 0.1),  
 hovertext = '취업자 히스토그램',   
 ybins = list(size = 15), stroke = I('white'))  
  
subplot(  
 p2, plotly\_empty(), p1, p3,   
 nrows = 2, heights = c(0.15, 0.85),   
 widths = c(0.85, 0.15), titleX = FALSE, titleY = FALSE,   
 shareX = TRUE, shareY = TRUE, margin = 0  
) |> hide\_legend()



# 상관(Correlatin)의 시각화

## 백신 접종자가 많으면 사망자가 감소하나? - 상관관계 산점도

산점도를 사용하는 가장 큰 이유는 데이터의 전체적인 분포를 보면서 데이터의 관계성을 확인하는데 있다. 이 관계성을 확인하는 과정에서 산점도는 사실 노이즈가 많이 발생한다. 이 노이즈들 사이에서 데이터의 관계성 패턴을 시각화하는 것이 추세선이다.

R에서 가장 많이 사용하는 ggplot2에서는 geom\_smooth()를 사용해서 쉽게 추세선을 그릴 수 있다. 하지만 plotly에서는 추세선을 그리는 함수를 지원하지 않는다. 그래서 plotly 객체에 추세선을 그리기 위해서는 추세선으로 많이 사용되는 회귀 함수를 사용하여 회귀 방정식을 산출하고 그 방정식에 따른 그래프를 그려줘야 한다. 일반적으로 추세선에 많이 사용되는 회귀 분석 함수는 lm()과 loess()이다.

먼저 선형 회귀인 lm()을 사용하여 회귀선을 그리기 위해서는 먼저 lm()으로 선형회귀 모델을 만들어야한다. 선형회귀 모델을 만들기 위해서는 독립 변수와 종속 변수를 결정해야 한다.

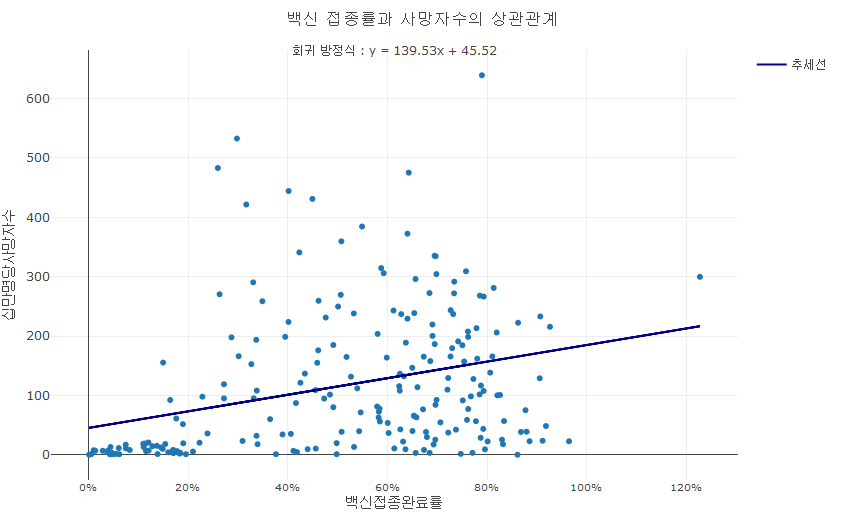
독립 변수는 말 그대로 독립된 변수, 다른 변수에 영향을 받지 않고 그 자체로 의미를 지니는 변수를 말한다. 흔히 입력 변수라고 생각하면 편할 것이다. 종속 변수는 독립 변수들의 변화에 따라 결정되는 변수를 말한다. 독립 변수의 변화에 따라 그 변화량이 종속되는 변수로 출력 변수로 생각하면 편하다. 보통 2차원 선형 회귀는 x와 y의 방정식으로 표현되는데 x에 해당하는 방정식의 오른쪽이 독립 변수, y에 해당하는 방정식의 왼쪽이 종속 변수를 표현한다.

여기서 알아보고자 하는 ’백신 접종자의 수가 많으면 사망자는 감소한다’는 가설을 증명해보기 위해서는 백신 접종자의 수와 사망자의 수간의 상관 관계를 살펴봐야한다. 이 상관관계를 확인하기 위해서는 인구 백명당 백신 접종완료자수를 독립 변수로 두고 이 독립 변수에 따른 십만명당 사망자수를 종속 변수로 설정하였다.

이렇게 결정한 독립변수와 독립변수에 대한 관계식을 lm()에 첫 번째 매개변수로 설정하여야 한다. 관계식을 표현할 떄는 ‘종속변수 ~ 독립변수`의 형태로 표현하며 독립 변수가 여러개 일때는’+’로 여러개의 독립 변수를 설정할 수 있지만 이 경우는 변수가 3개 이상이 되므로 평면에 표시되는 2차원 시각화에는 다소 어려움이 따른다.

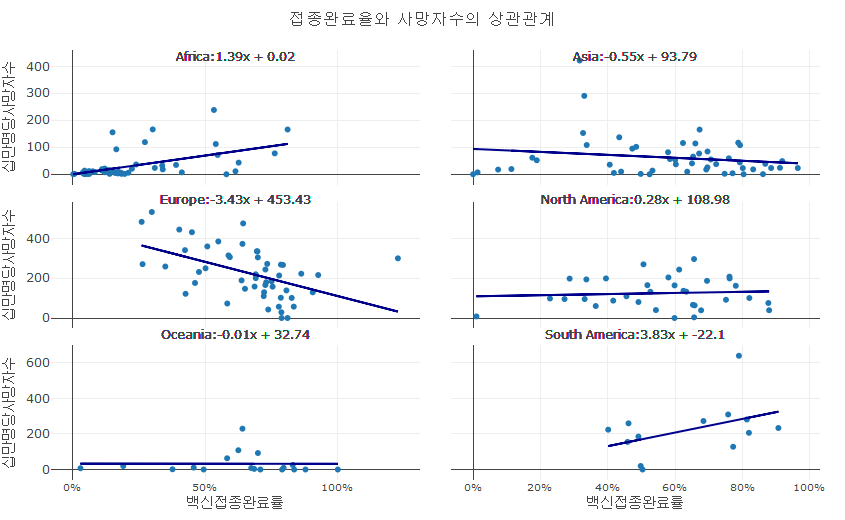
백신 접종 완료자수에 따른 사망자 수에 대한 상관 관계를 표시하기 위해서는 lm()외에 fitted()가 추가적으로 사용된다. fitted()는 회귀 모델 등 여러가지 통계 모델에 의해 입력된 독립 변수에 따른 종속 변수의 결과값을 산출하는 함수이다. 따라서 plotly의 X축에는 독립변수를 설정하고 Y축에 통계 모델에 따른 독립 변수의 결과 종속 변수 값을 산출하여 선을 그어주면 추세선을 표시할 수 있다.

df\_covid19\_stat |> filter(is.finite(백신접종완료률),   
 is.finite(십만명당사망자수),   
 !is.na(백신접종완료률),   
 !is.na(십만명당사망자수),   
 십만명당사망자수 != 0,   
 인구백명당백신접종완료률 != 0) |>  
 plot\_ly() |>  
 add\_trace(type = 'scatter', mode = 'markers',   
 x = ~백신접종완료률, y = ~십만명당사망자수,   
 text = ~location, hoverinfo = 'x+y',  
 hovertemplate = '접종률:%{x},사망자:%{y}(명/십만명)<br>국가명:%{text}<extra></extra>',  
 showlegend = FALSE) |>  
 add\_trace(type = 'scatter', mode = 'lines',   
 x = ~백신접종완료률,   
 y = ~fitted(lm(십만명당사망자수 ~ 백신접종완료률)),   
 name = '추세선',   
 color = I('darkblue')  
 ) |>  
 layout(title = '백신 접종률과 사망자수의 상관관계',   
 margin = margins,  
 showlegend = TRUE,   
 xaxis = list(tickfont = list(size = 10),  
 title = list(text = '백신접종완료률',   
 standoff = 5),   
 tickformat = ".0%"),  
 yaxis = list(title = list(text = '십만명당사망자수'),  
 tickformat = ".0f"),  
 annotations = list(x = 0.5 , y = 1.02,  
 text = ~paste0('회귀 방정식 : y = ',  
 round(lm(십만명당사망자수 ~ 백신접종완료률)$coefficients[2], 2),   
 'x + ',   
 round(lm(십만명당사망자수 ~ 백신접종완료률)$coefficients[1], 2)),  
 showarrow = F, xref='paper',  
 yref='paper', xanchor = 'center')  
 )



앞의 추세선은 전체 국가의 인구백명당 백신접종자수 대비 십만명당 사망자수의 회귀 추세선이다. 지금 이 회귀 추세선이 우리에게 알려주는 것은 인구 백명당 백신 접종 완료자수가 많아 질수록 인구 십만명당 사망자수도 증가한다는 것이다. 앞의 서브 플롯들의 제목 옆에 표기된 방정식을 보면 아마도 대부분 중학교때 배운 일차 방정식을 떠올릴 것이다. 물론 일차 방정식으로 해석해도 무방하지만 이 방정식의 통계적 의미는 조금 다르다. Y축의 수치는 결국 일차 방정식의 결과로 산출되는데 X축의 단위 1이 늘면 방정식의 X 계수만큼 Y가 증가 혹은 감소한다는 의미이다. 앞의 그래프에서 산출된 회귀 추세선의 경우 백신접종완료자가 1명 늘어나면 십만명당 사망자수가 1.4명 늘어난다는 것이다. 따라서 일단 처음 세운 가설은 맞지 않는다는 것이다. 그렇다면 이 추세선을 대륙별로 살펴보도록 하자.

## 긴 형태의 100일 코로나19 데이터에서  
df\_covid19\_stat |>  
 filter(is.finite(백신접종완료률), is.finite(십만명당사망자수)) |>  
 filter(!is.na(백신접종완료률), !is.na(십만명당사망자수), !is.na(continent)) |>  
 mutate(백신접종완료률1 = 백신접종완료률\*100) |>  
 ## 국가명으로 그룹화  
 group\_by(continent) |>  
 ## 그룹화한 각각의 데이터 그룹들에 적용할 코드 설정  
 do (  
 ## 각 그룹화한 데이터를 사용해 plotly 객체 생성   
 p = plot\_ly(.) |>  
 add\_trace(type = 'scatter', mode = 'markers',   
 x = ~백신접종완료률, y = ~십만명당사망자수,  
 color = I('#1f77b4'), showlegend = FALSE,   
 text = ~location, hoverinfo = 'x+y',  
 hovertemplate = '접종률:%{x},사망자:%{y}(명/십만명)<br>국가명:%{text}<extra></extra>'  
) |>  
 add\_trace(type = 'scatter', mode = 'lines',  
 x = ~백신접종완료률,  
 y = ~fitted(lm(십만명당사망자수 ~ 백신접종완료률1)),  
 name = '추세선', color = I('darkblue')  
 ) |>  
 layout(showlegend = TRUE,   
 title = list(title = NULL),  
 xaxis = list(tickfont = list(size = 10),  
 title = list(text = '백신접종완료률',   
 standoff = 5),   
 tickformat = ".0%"),  
 yaxis = list(title = list(text = '십만명당사망자수'),  
 tickformat = ".0f"),  
 annotations = list(x = 0.5 , y = 1.02,  
 text = ~paste0(continent, ':',   
 round(lm(십만명당사망자수 ~ 백신접종완료률1)$coefficients[2], 2), 'x + ', round(lm(십만명당사망자수 ~ 백신접종완료률1)$coefficients[1], 2)),  
 showarrow = F, xref='paper',  
 yref='paper', xanchor = 'center')  
 )  
 ) |>  
 ## 생성된 plotly 객체들을 subplot 생성  
 subplot(nrows = 3, shareX = TRUE, shareY = TRUE) |>  
 ## 생성된 subplot의 layout 설정  
 layout(showlegend = FALSE,   
 title = '접종완료율와 사망자수의 상관관계',  
 margin = margins)

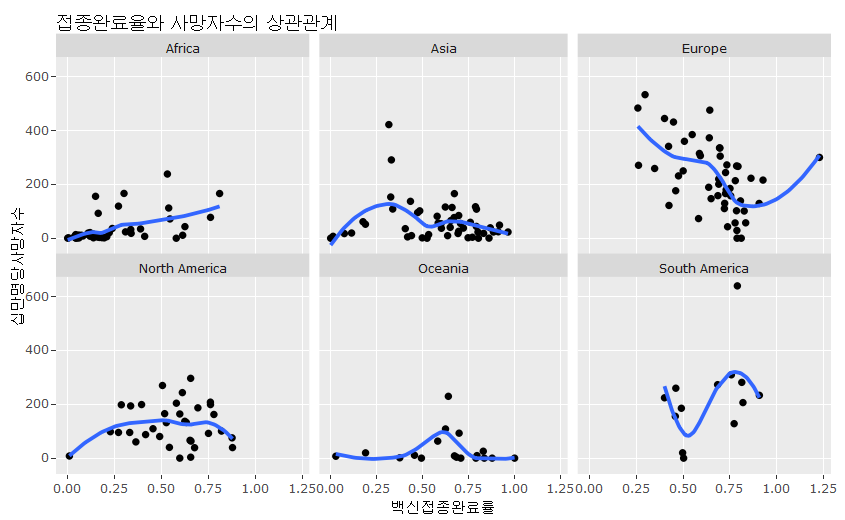


대륙별로 접종 완료자수와 사망자수의 상관 관계를 살펴보면 전체 국가의 상관관계와 조금 다르게 보이는 대륙이 보인다. 아프리카와 남미의 경우는 백신 접종자가 늘어나도 사망자의 감소세가 나타나지 않았지만 유럽과 아시아의 경우는 가설을 세운 것과 같이 백신 접종자가 늘어날수록 사망자가 감소하는 상관관계를 보인다. 또 오세아니아와 북미 대륙은 백신 접종자 수와 사망자수의 큰 상관 관계가 보이지 않는다.

여기서 하나 중요한 것은 **상관관계가 인과관계를 뜻하지 않는다는 것**이다. 즉 전체 국가의 백신 접종률이 높아질수록 높아진 사망자수 상관관계가 백신 접종률이 높았기 때문에 사망자가 늘었다는 원인과 결과관계가 아니라는 것이다. 또 비교적 강한 상관관계를 보이는 유럽과 아시아의 경우 백신 접종률이 높았기 때문에 사망자가 줄었다는 성급한 결론은 절대로 경계해야 한다는 것이다. 이를 증명하기 위해서는 데이터의 오류나 다양한 요인에 대한 검증을 필수적으로 수행하여야 한다.

앞의 추세선은 선형회귀 방정식을 생성하는 lm()을 사용했기 때문에 추세선이 직선으로 그어진다. 이번에는 추세선을 설정하는 또 하나의 방법인 loess()를 사용하는 방법을 ggplot2를 사용해 생성하고 plotly로 전환해보자.

ggplotly <- df\_covid19\_stat |>  
 filter(is.finite(백신접종완료률), is.finite(십만명당사망자수)) |>  
 filter(!is.na(백신접종완료률), !is.na(십만명당사망자수), !is.na(continent)) |>  
 ggplot() +  
 ## X축을 졸업자\_계, Y축을 취업자\_합계\_계, color를 대계열로 매핑한 geom\_point로 산점도 레이어 생성  
 geom\_point(aes(x = 백신접종완료률, y = 십만명당사망자수)) +   
 ## X축을 졸업자\_계, Y축을 취업자\_합계\_계, color를 대계열로 매핑한 geom\_smooth로 추세선 레이어 생성  
 geom\_smooth(aes(x = 백신접종완료률, y = 십만명당사망자수), method = 'loess', se = FALSE) +  
 facet\_wrap(~continent) +  
 labs(title = '접종완료율와 사망자수의 상관관계', x = '백신접종완료률', y = '십만명당사망자수')  
  
ggplotly |>  
 ggplotly()

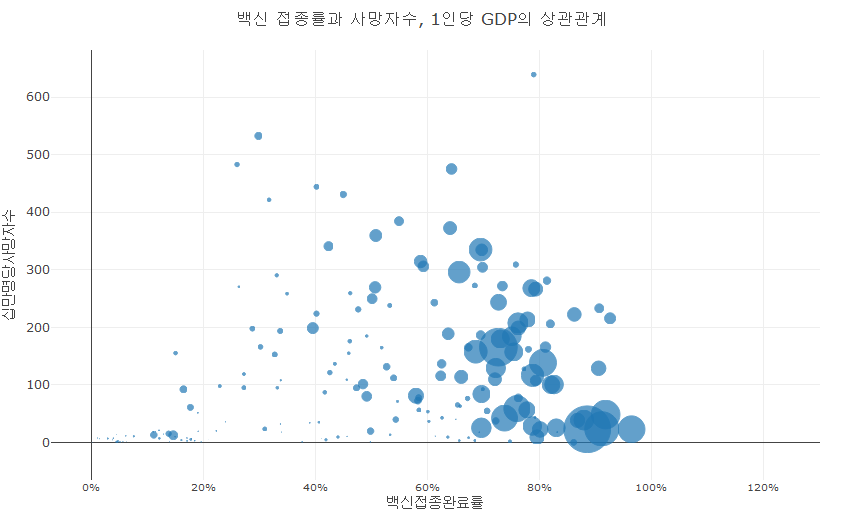


## 부자 나라의 접종완료율과 사망자수의 상관은? - 거품형 차트(Bubble Chart)

산점도는 X축과 Y축의 2차원 데이터를 사용하는 시각화 방법이다. 단 두개의 변수만의 관계를 살펴보는 것은 데이터의 관계성을 찾는데 한계가 있어 추가적인 변수들을 표현하는 방법들이 많이 제공되고 있다. 이에 대표적으로 사용되는 시각화 방법이 거품형 차트이다. 거품형 차트는 2018년 발간된 한스 로슬링의 ’팩트풀니스(Factfullness)’에서 주로 사용되며 유명세를 탄 사각화 방법이다. X축과 Y축의 변수에 기초하여 표현되는 산점도 데이터 포인트의 크기를 매핑한 추가적 변수를 사용함으로써 3차원 산점도를 표현하는 방법이다. 거품형 차트는 특별한 함수가 제공되는 것이 아니고 산점도의 점 크기를 사용해 표현한다.

다음은 앞에서 만들었던 접종 완료자수와 사망자수의 산점도에 각 국가의 1인당 GDP를 원의 크기로 표현하여 거품형 차트를 만드는 예이다.

df\_covid19\_stat |> filter(is.finite(백신접종완료률),   
 is.finite(십만명당사망자수),   
 !is.na(백신접종완료률),   
 !is.na(십만명당사망자수),   
 십만명당사망자수 != 0,   
 인구백명당백신접종완료률 != 0) |>  
 plot\_ly() |>  
 add\_trace(type = 'scatter', mode = 'markers',   
 x = ~백신접종완료률, y = ~십만명당사망자수,   
 marker = list(size = ~(인당GDP/2500)),  
 text = ~location,   
 hoverinfo = 'x+y',  
 hovertemplate = '접종률:%{x},사망자:%{y}(명/십만명)<br>국가명:%{text}<extra></extra>',   
 showlegend = FALSE) |>  
 layout(title = '백신 접종률과 사망자수, 1인당 GDP의 상관관계',   
 margin = margins,  
 showlegend = TRUE,   
 xaxis = list(tickfont = list(size = 10),  
 title = list(text = '백신접종완료률',   
 standoff = 5),   
 tickformat = ".0%"),  
 yaxis = list(title = list(text = '십만명당사망자수'),  
 tickformat = ".0f"),  
 showarrow = F, xref='paper',  
 yref='paper', xanchor = 'center')

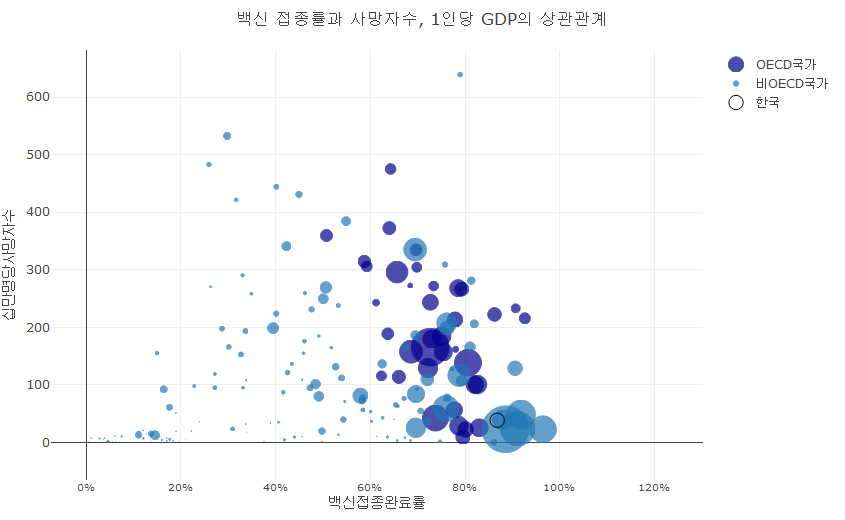


실행결과 5- 거품형 차트

앞의 그래프를 살펴보면 고소득 국가를 의미하는 큰 원들은 전반적으로 백신 접종률이 약 70%를 넘어가는 것으로 보인다. 또 이들의 십만명당 사망자수는 비교적 아래쪽으로 위치해 있음을 알 수 있다.

그렇다면 세계 경제를 이끌고 가는 국제기구인 OECD에 가입된 국가들의 상황을 알아보고자 하면 어떻게 할까? 다음과 같이 OECD 국가의 벡터를 만들고 이들 국가들의 거품색을 차이나게 해줌으로써 거품형 차트에 추가적인 정보를 표현할 수 있다. 그리고 여기에 한국의 위치를 추가적으로 표현하는 코드는 다음과 같다.

oecd\_country <- c('United States', 'Canada', 'United Kingdom', 'Denmark',   
 'Iceland', 'Norway', 'Turkey', 'Spain', 'Portugal',   
 'France', 'Ireland', 'Belgium', 'Germany', 'Greece', 'Sweden',   
 'Austria', 'Italy', 'Switzerland', 'Luxembourg', 'Netherlands',   
 'Australia', 'Chile', 'Colombia', 'Costa Rica', 'Czechia',   
 'Estonia', 'Finland', 'Hungary', 'Israel', 'Japan',   
 'South Korea', 'Latvia', 'Lithuania', 'Mexico',   
 'New Zealand', 'Poland', 'Slovakia', 'Slovenia')  
  
oecd <- df\_covid19\_stat |> filter(is.finite(백신접종완료률),   
 is.finite(십만명당사망자수),   
 !is.na(백신접종완료률),   
 !is.na(십만명당사망자수),   
 십만명당사망자수 != 0,   
 인구백명당백신접종완료률 != 0) |>  
 filter(location %in% oecd\_country, location != 'South Korea')  
  
nooecd <- df\_covid19\_stat |> filter(is.finite(백신접종완료률),   
 is.finite(십만명당사망자수),   
 !is.na(백신접종완료률),   
 !is.na(십만명당사망자수),   
 십만명당사망자수 != 0,   
 인구백명당백신접종완료률 != 0) |>  
 filter(!(location %in% oecd\_country), location != 'South Korea')  
  
korea <- df\_covid19\_stat |> filter(location == 'South Korea')  
  
  
 plot\_ly() |>  
 add\_trace(data = oecd, type = 'scatter', mode = 'markers',   
 x = ~백신접종완료률, y = ~십만명당사망자수,   
 name = 'OECD국가', color = I('darkblue'),   
 marker = list(size = ~(인당GDP/2500)  
 ),  
 text = ~location,   
 hoverinfo = 'x+y',  
 hovertemplate = '<b>%{text}</b><br>접종률:%{x},사망자:%{y}(명/십만명)<br>인당GDP:%{marker.size}<extra></extra>',   
 showlegend = T) |>  
 add\_trace(data = nooecd, type = 'scatter', mode = 'markers',   
 x = ~백신접종완료률, y = ~십만명당사망자수,   
 name = '비OECD국가', color = I('#1f77b4'),  
 marker = list(size = ~(인당GDP/2500)  
 ),  
 text = ~location,   
 hoverinfo = 'x+y',  
 hovertemplate = '<b>%{text}</b><br>접종률:%{x},사망자:%{y}(명/십만명)<br>인당GDP:%{marker.size}<extra></extra>',   
 showlegend = T) |>  
 add\_trace(data = korea, type = 'scatter', mode = 'markers',   
 x = ~백신접종완료률, y = ~십만명당사망자수,   
 name = '한국', color = I('black'),  
 marker = list(size = ~(인당GDP/2500), symbol = 100  
 ),  
 text = ~location,   
# hoverinfo = 'x+y',  
 hovertemplate = '<b>%{text}</b><br>접종률:%{x},사망자:%{y}(명/십만명)<br>인당GDP:%{marker.size:,}<extra></extra>',   
 showlegend = T) |>  
 layout(title = '백신 접종률과 사망자수, 1인당 GDP의 상관관계',   
 margin = margins,  
 showlegend = TRUE,   
 xaxis = list(tickfont = list(size = 10),  
 title = list(text = '백신접종완료률',   
 standoff = 5),   
 tickformat = ".0%"),  
 yaxis = list(title = list(text = '십만명당사망자수'),  
 tickformat = ".0f"))



실행결과 5-색을 사용한 거품형 차트

이 거품형 차트를 보면 OECD 국가들은 백신접종완료율이 60~80%, 십만명당 사망자수가 300명 이하에 대부분 위치하는 것으로 보인다. 이에 비해 우리나라는 80%를 훨씬 넘는 백신접종완료율에 50명도 안되는 십만명당 사망자수를 보이는 것으로 나타난다.

# 추세(Trend)의 시각화

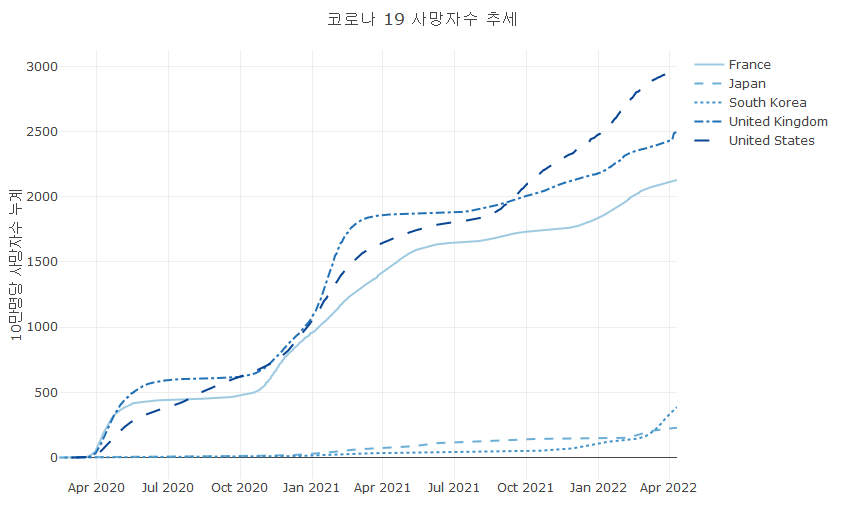
추세(Trend)란 데이터의 흐름에 따라 발생하는 데이터의 변화를 말한다. 보통 추세는 시간적 흐름에 따라 발생한 데이터를 말하는 경우가 많지만 추세가 꼭 시간의 흐름에 종속되지는 않는다. 예를 들자면 회차(물론 이 또한 시간의 흐름과 무관하지 않지만)나 이벤트의 발생과 같은 흐름도 추세에 속할 수 있다. 하지만 추세에 측정에 있어 하나 중요한 것은 그것이 시간이든 회차이든 특정 이벤트이던 그들의 흐름을 측정하는 간격이나 성질이 일정해야 한다는 것이다. 시간의 경우 추세를 측정하기위해서는 시간적 간격, 즉, 연도별, 월별, 일별 등의 간격이 동일해야 하고 회차의 경우 1회, 2회와 같이 연속된 회차로 기록되어야 유의미하다. 만약 시간의 간격이 어느 구간에서는 연도별, 어느 구간에서는 월별로 표현된다면 추세를 정확히 파악하기 어렵다. 따라서 추세에는 데이터의 흐름, 특히 흐름의 측정 간격이 매우 중요하다.

추세를 시각화할 때는 데이터의 포인트와 해당 데이터의 바로 전 데이터와 다음 데이터를 연결하는 선 그래프가 많이 사용되지만 막대그래프도 많이 사용된다.

## 주요 나라의 코로나19 사망자 추세는? - 선 그래프

선 그래프 (또는 꺾은 선형 차트)는 특정한 변량의 흐름에 따라 변화되는 데이터 값들을 선으로 연결하여 그 변화량을 보여주는 시각화 방법이다. 이 선그래프가 가장 효과적으로 사용되는 시각화가 시간의 흐름에 따라 변화하는 시계열 데이터에 대한 시각화 방법이다. 각각의 시간에 관측된 데이터 포인트들을 같은 변수이나 변량끼리 선으로 연결하였기 때문에 그 기본은 산점도에 있다고 할수도 있다.

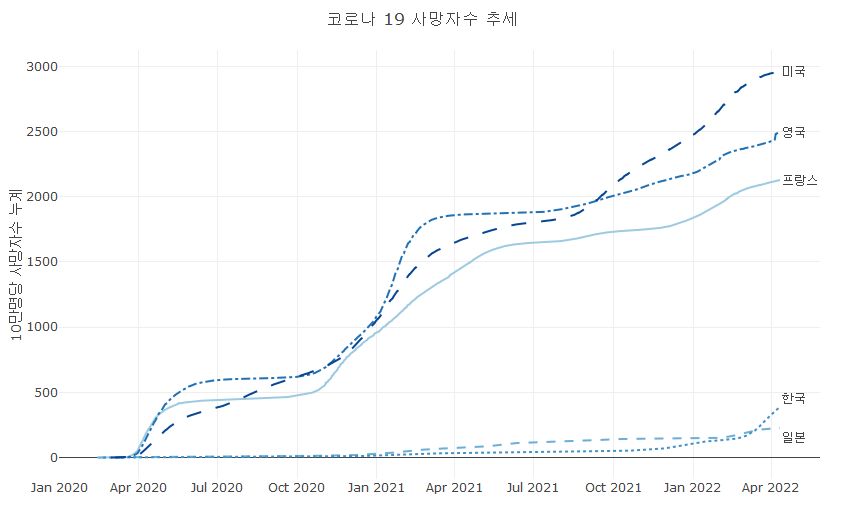
total\_deaths\_5\_nations\_by\_day <- df\_covid19 |>   
 filter((iso\_code %in% c('KOR', 'USA', 'JPN', 'GBR', 'FRA'))) |>  
 filter(!is.na(total\_deaths\_per\_million))  
  
total\_deaths\_5\_nations\_by\_day |>  
 ## plotly 객체 생성  
 plot\_ly() |>  
 add\_trace(type = 'scatter', mode = 'lines',   
 x = ~date, y = ~total\_deaths\_per\_million , linetype = ~location, connectgaps = T) |>  
 layout(title = '코로나 19 사망자수 추세',   
 xaxis = list(title = ''),   
 yaxis = list(title = '10만명당 사망자수 누계'),   
 margin = margins)



앞의 시각화를 보면 주요 5개국의 10만명당 사망자수 누계의 추세를 보이고 있다. 2020년 4~5월 경부터 영국, 프랑스의 사망자수가 급격히 증가하고 이 시기부터 미국의 사망자수도 증가하였지만 초기의 사망자 추세는 영국, 프랑스보다는 증가 추세가 높지 않았다. 하지만 2020년 연말에 접어들면서 이 세 나라의 증가세가 비슷해지기 시작했고 이후 미국의 증가세는 꾸준히 증가한 반면 프랑스와 영국은 2021년 상반기부터 증가 추세가 낮아지기 시작했다. 반면 우리나라와 일본의 경우 2021년까지 매우 낮은 증가세를 모이지만 꾸준히 증가하였고 2022년에 들어서 우리나라의 증가세가 급격히 늘어나기 시작한 것으로 나타나고 있다.

이 시각화를 보면 범례를 사용하여 각 선에 해당하는 국가를 나타내고 있다. 하지만 선에 따른 국가를 확인하기 위해서는 범례와 데이터 선을 번갈아 찾아야 하기 때문에 다소 불편함이 따른다. 선 옆에 바로 국가명을 표현해 주면 이러한 불편함이 다소 감소될 수 있다.

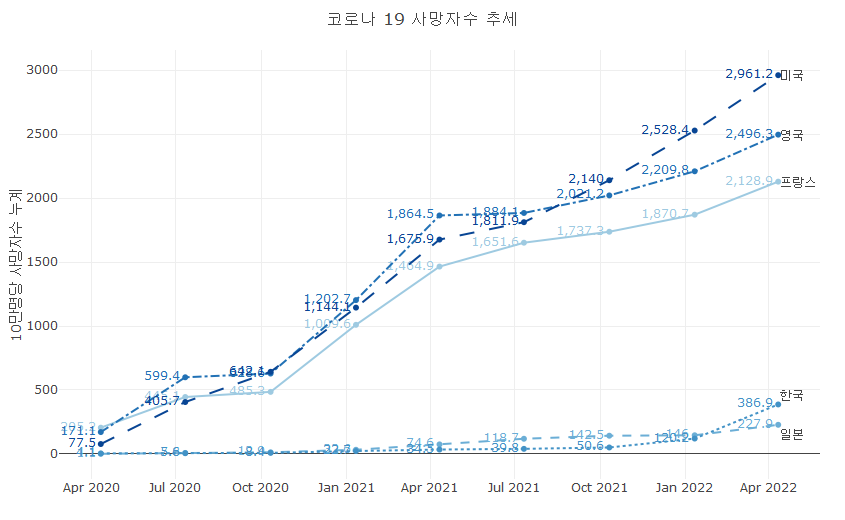
total\_deaths\_5\_nations\_by\_day |>  
 ## plotly 객체 생성  
 plot\_ly() |>  
 add\_trace(type = 'scatter', mode = 'lines+text',   
 x = ~date, y = ~total\_deaths\_per\_million , linetype = ~location, connectgaps = T) |>  
 add\_annotations(text = '프랑스',   
 ## 프랑스 trace의 마지막 위치에 주석 추가  
 x = total\_deaths\_5\_nations\_by\_day |> filter(location == 'France', date == max(date)) |>  
 select(date) |> pull(),   
 y = total\_deaths\_5\_nations\_by\_day |> filter(location == 'France', date == max(date)) |>  
 select(total\_deaths\_per\_million) |> pull(),  
 xanchor = 'left', showarrow = FALSE  
 ) |>  
 add\_annotations(text = '일본',   
 ## 일본 trace의 마지막 위치에 주석 추가  
 x = total\_deaths\_5\_nations\_by\_day |> filter(location == 'Japan', date == max(date)) |>  
 select(date) |> pull(),   
 y = total\_deaths\_5\_nations\_by\_day |> filter(location == 'Japan', date == max(date)) |>  
 select(total\_deaths\_per\_million) |> pull(),  
 xanchor = 'left', yanchor = 'top', showarrow = FALSE  
 ) |>  
 add\_annotations(text = '영국',   
 ## 영국 trace의 마지막 위치에 주석 추가  
 x = total\_deaths\_5\_nations\_by\_day |> filter(location == 'United Kingdom', date == max(date)) |>  
 select(date) |> pull(),   
 y = total\_deaths\_5\_nations\_by\_day |> filter(location == 'United Kingdom', date == max(date)) |>  
 select(total\_deaths\_per\_million) |> pull(),  
 xanchor = 'left', showarrow = FALSE  
 ) |>  
 add\_annotations(text = '미국',   
 ## 미국 trace의 마지막 위치에 주석 추가  
 x = total\_deaths\_5\_nations\_by\_day |> filter(location == 'United States', date == max(date)) |>  
 select(date) |> pull(),   
 y = total\_deaths\_5\_nations\_by\_day |> filter(location == 'United States', date == max(date)) |>  
 select(total\_deaths\_per\_million) |> pull(),  
 xanchor = 'left', showarrow = FALSE  
 ) |>  
 add\_annotations(text = '한국',   
 ## 한국 trace의 마지막 위치에 주석 추가  
 x = total\_deaths\_5\_nations\_by\_day |> filter(location == 'South Korea', date == max(date)) |>  
 select(date) |> pull(),   
 y = total\_deaths\_5\_nations\_by\_day |> filter(location == 'South Korea', date == max(date)) |>  
 select(total\_deaths\_per\_million) |> pull(),  
 xanchor = 'left', yanchor = 'bottom', showarrow = FALSE  
 ) |>  
 layout(title = '코로나 19 사망자수 추세',   
 xaxis = list(title = ''),   
 yaxis = list(title = '10만명당 사망자수 누계'),   
 margin = margins,  
 showlegend = FALSE)



앞의 시각화에서 전반적인 데이터의 흐름은 보이지만 구체적인 데이터 값은 보이지 않는다. 데이터 값을 표기하기 위해서는 앞선 장에서 설명했다시피 mode 속성에 ‘+text’를 포함하면 된다. 또 보통 데이터 값을 표기할 때에는 정확한 데이터 값의 위치를 같이 표기해 주는 것이 일반적이기 때문에’+markers’까지 넣어주는 것이 좋다. 하지만 데이터가 일별 데이터이기 때문에 mode에 ’lines+text+markers’를 설정해주면 도대체 알 수 없는 시각화가 나온다. 따라서 이런 경우에는 데이터를 일정한 주기별로 정제하여 사용하는 것이 좋다.

이렇게 mode에 ‘text’를 설정할 때는 하여 정확한 데이터를 표시하는 방법도 있지만 마우스의 이동에 따라 X, Y축의 정확한 위치를 표시해주는 보조선을 사용하는 방법도 있다. 이런 보조선은 layout()의 hovermode속성을 사용하여 설정할 수 있다. hovermode는 ’x unified’, ’y unified’로 설정하면 X, Y축의 수직, 수평선의 보조선이 생성되고 이 선에 해당하는 데이터에 대한 정보가 표시된다.

total\_deaths\_5\_nations\_by\_3month <-   
 total\_deaths\_5\_nations\_by\_day |>  
 ## 가장 최근 데이터부터 가장 오래된 데이터까지 3개월 단위 데이터 필터링  
 filter(date %in% seq.Date(max(date), min(date), by = '-3 month'))  
  
total\_deaths\_5\_nations\_by\_3month |>  
 plot\_ly() |>  
 ## 데이터 trace 추가  
 add\_trace(type = 'scatter', mode = 'lines+text+markers',   
 x = ~date, y = ~total\_deaths\_per\_million,   
 text = ~round(total\_deaths\_per\_million, 1),   
 textposition = "left",   
 texttemplate = '%{text:,}',  
 linetype = ~location, connectgaps = T) |>  
 add\_annotations(text = '프랑스',   
 x = total\_deaths\_5\_nations\_by\_3month |> filter(location == 'France', date == max(date)) |>  
 select(date) |> pull(),   
 y = total\_deaths\_5\_nations\_by\_3month |> filter(location == 'France', date == max(date)) |>  
 select(total\_deaths\_per\_million) |> pull(),   
 xanchor = 'left', showarrow = FALSE  
 ) |>  
 add\_annotations(text = '일본',   
 x = total\_deaths\_5\_nations\_by\_3month |> filter(location == 'Japan', date == max(date)) |>  
 select(date) |> pull(),   
 y = total\_deaths\_5\_nations\_by\_3month |> filter(location == 'Japan', date == max(date)) |>  
 select(total\_deaths\_per\_million) |> pull(),   
 xanchor = 'left', yanchor = 'top', showarrow = FALSE  
 ) |>  
 add\_annotations(text = '영국',   
 x = total\_deaths\_5\_nations\_by\_3month |> filter(location == 'United Kingdom', date == max(date)) |>  
 select(date) |> pull(),   
 y = total\_deaths\_5\_nations\_by\_3month |> filter(location == 'United Kingdom', date == max(date)) |>  
 select(total\_deaths\_per\_million) |> pull(),   
 xanchor = 'left', showarrow = FALSE  
 ) |>  
 add\_annotations(text = '미국',   
 x = total\_deaths\_5\_nations\_by\_3month |> filter(location == 'United States', date == max(date)) |>  
 select(date) |> pull(),   
 y = total\_deaths\_5\_nations\_by\_3month |> filter(location == 'United States', date == max(date)) |>  
 select(total\_deaths\_per\_million) |> pull(),  
 xanchor = 'left', showarrow = FALSE  
 ) |>  
 add\_annotations(text = '한국',   
 x = total\_deaths\_5\_nations\_by\_3month |> filter(location == 'South Korea', date == max(date)) |>  
 select(date) |> pull(),   
 y = total\_deaths\_5\_nations\_by\_3month |> filter(location == 'South Korea', date == max(date)) |>  
 select(total\_deaths\_per\_million) |> pull(),  
 xanchor = 'left', yanchor = 'bottom', showarrow = FALSE  
 ) |>  
 layout(title = '코로나 19 사망자수 추세',   
 xaxis = list(title = ''),   
 yaxis = list(title = '10만명당 사망자수 누계'),   
 margin = margins,  
 showlegend = FALSE,   
 hovermode = 'x unified')

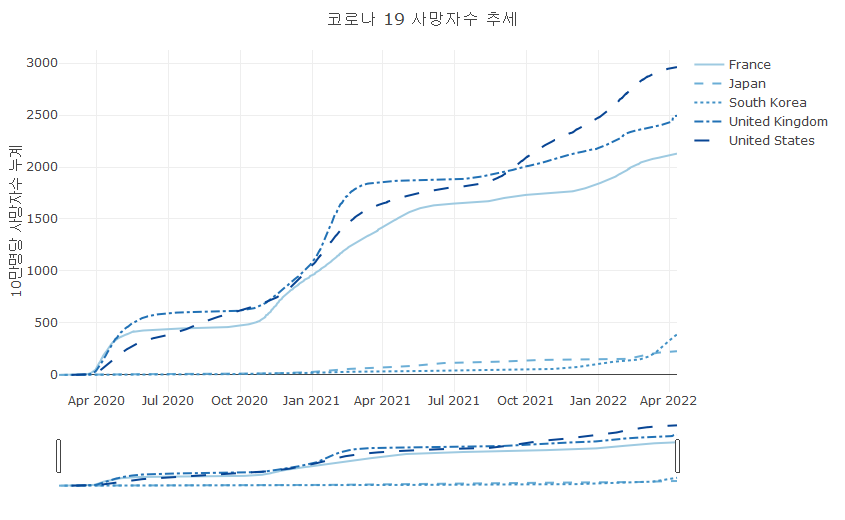


## 시간을 줄이고 늘이고.. - rangeslider

지금까지 그려본 plotly 선 그래프는 사실 정적 시각화로도 그릴 수 있는 그래프이다. 물론 plotly가 modebar나 마우스를 사용한 상호작용과 같이 기본적으로 제공하는 동적 시각화 기능을 사용하면 시각화를 다양하게 사용할 수 있지만 plotly에서만 제공하는 선 그래프의 특별한 기능들이 있다. 그 중에 하나가 ’rangeslider’이다.

rangeslider는 선 그래프의 전체적인 형태를 유지하면서 사용자가 직접 X축에 매핑된 시간 축을 이동, 확대, 축소하기 위한 작은 서브플롯차트를 제공하는 기능이다. 이 서브 플롯의 왼쪽 막대와 오른쪽 막대를 움직이면서 X축의 범위를 사용자가 직접 설정할 수 있다. 이 rangeslider는 X축에만 제공되는 속성인 rangeslider의 세부 속성인 visible을 ’TRUE’로 설정하면 나타나고 세부 설정을 위한 다양한 속성들을 제공한다.

total\_deaths\_5\_nations\_by\_day |>  
 ## plotly 객체 생성  
 plot\_ly() |>  
 add\_trace(type = 'scatter', mode = 'lines',   
 x = ~date, y = ~total\_deaths\_per\_million ,   
 linetype = ~location, connectgaps = T  
) |>  
 layout(title = '코로나 19 사망자수 추세',   
 xaxis = list(title = '', rangeslider = list(visible = T)),   
 yaxis = list(title = '10만명당 사망자수 누계'),   
 showlegend = T, margin = margins,   
 title='Time Series with Rangeslider',  
 margin = margins)



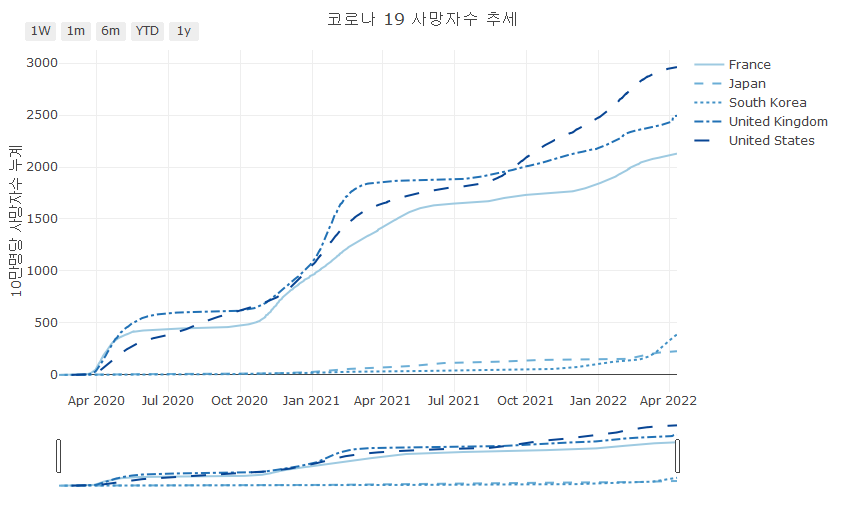
## 기간 설정을 버튼으로.. - rangeselector

앞서 설명한 rangeslider는 전체 기간중에 특정 기간을 사용자가 직접 설정할 수 있는 장점이 있지만 정확한 기간을 설정하기는 어렵다. 예를 들어 최근 30일, 최근 6개월과 같은 명확한 기간을 설정하고자 할 때는 효과적이지 못하다. 이런 경우를 대비하여 plotly에서 제공하는 기능이 rangeselecor이다. rangeselector는 버튼으로 제공되는데 최근 일에서부터 거꾸로 얼마의 기간 범위를 설정할지를 결정할 수 있다. rangeselector의 button 속성을 설정하기 위해 사용하는 주요 속성은 다음과 같다.

| 속성 | 설명 | 속성값 | 세부속성 |
| --- | --- | --- | --- |
| count | step으로 설정된 단위를 얼마나 shift할지 설정 | 0이상의 수치 |  |
| label | 버튼의 표시 문자열 | 문자열 |  |
| step | count의 값에서 사용될 시간 간격 설정 | ‘month’, ‘year’, ‘day’, ‘hour’, ‘minute’, ‘second’, ‘all’ |  |
| stepmode | 범위 업데이트 모드의 설정 | ‘backward’, ‘todate’ |  |
| visible | 버튼을 표시할지 설정 | 논리값 |  |

다음의 코드를 보면 총 5개의 버튼을 생성하였다. 첫 번째 버튼은 step을 ’day’로 설정하고 count를 7로 설정하였기 때문에 범위를 최근일로부터 7일전부터 최근일까지를 설정한다. 네 번째 버튼에서 보면 stepmode가 다른 버튼과 달리 ’todate’로 설정되어 있다. 반면 다섯 번째 버튼은 네 번째 버튼과 stepmode외에는 동일한 속성들을 가진다. stepmode가 ’todate’로 설정되면 step이 count만큼의 설정되는 범위에서 가장 가까운 타임스탬프로 위치한다. 따라서 stepmode가 ’todate’로 설정되면 현재로부터 1년전의 1월 1일로 범위가 설정된다. 반면 stepmode가 ’bakcward’로 설정되면 현재로부터 1년전까지만 설정이 된다. 예를 들어 X축의 마지막 날짜가 2022년 3월 1일이라면 2021년 3월 1일로 범위가 설정되게 된다.

total\_deaths\_5\_nations\_by\_day |>  
 ## plotly 객체 생성  
 plot\_ly() |>  
 add\_trace(type = 'scatter', mode = 'lines',   
 x = ~date, y = ~total\_deaths\_per\_million , linetype = ~location, connectgaps = T) |>  
 layout(title = '코로나 19 사망자수 추세',   
 yaxis = list(title = '10만명당 사망자수 누계'),   
 xaxis = list(title = '',   
 range = c(min(total\_deaths\_5\_nations\_by\_day$date),  
 max(total\_deaths\_5\_nations\_by\_day$date)),  
 rangeslider = list(visible = T, autorange = F, range = c(min(total\_deaths\_5\_nations\_by\_day$date),  
 max(total\_deaths\_5\_nations\_by\_day$date))  
 ),   
 rangeselector=list(  
 buttons=list(  
 list(count=7, label="1W", step="day", stepmode="backward"),  
 list(count=1, label="1m", step="month", stepmode="backward"),  
 list(count=6, label="6m", step="month", stepmode="backward"),  
 list(count=1, label="YTD", step="year", stepmode="todate"),  
 list(count=1, label="1y", step="year", stepmode="backward")  
 )  
 )  
 ),  
 showlegend = T, margin = margins  
 )

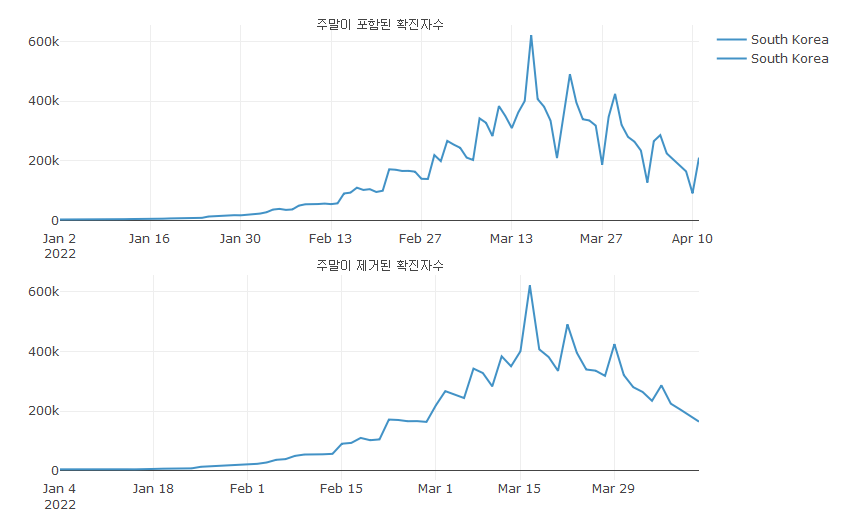


## 주말 효과를 제거해보자 - rangebreaks

코로나19 데이터의 1주일 이상의 장기 데이터를 한번이라도 본 경험이 있다면 일요일과 월요일에 확진자 수가 급감했다가 화요일부터 다시 증가한다는 계절성을 보았을 것이다. 토요일과 일요일에 검사 건수가 적어지는 주말 효과에 의해 검사 결과가 나오는 일요일과 월요일의 확진자가 감소했다가 월요일부터 다시 검사 건수가 늘어나기 때문에 이 검사 결과가 나오는 화요일부터 확진자가 증가한다. 따라서 이 주말효과는 데이터의 전반적 추세를 살펴보는데 다소 방해가 되는 요소이다. plotly는 이와 같은 달력 상의 특정 주기나 특정 날짜를 제거해주는 기능을 rangebreaks를 통해 설정할 수 있다. rangebreaks를 사용할 때 하나 주의해야하는 것은 rangebreaks의 세부속성을 모두 리스트로 만들어 주어야 한다는 것이다.

| 속성 | 설명 | 속성값 | 세부속성 |
| --- | --- | --- | --- |
| bounds | rangebreaks를 설정할 최소, 최대값을 설정, 패턴을 설정할 수 있음 | 리스트 |  |
| dvalue | values에 설정하는 크기 설정. 밀리세컨드로 설정 | 0이상의 수치 |  |
| enable | rangebreaks를 설정할지 여부 설정 | 논리값 |  |
| pattern | rangebreaks로 설정할 타임라인 패턴 설정 | ‘day of week’, ‘hour’, ’’ |  |
| values | rangebreak에 해당하는 좌표 값을 설정 | 리스트 |  |

total\_deaths\_5\_nations\_since\_100day <-   
 total\_deaths\_5\_nations\_by\_day |>  
 ## 한국 데이터만 필터링  
 filter((iso\_code %in% c('KOR'))) |>  
 ## 주말 효과를 확인하기 위해 최근 100일 데이터만 필터링  
 filter(date > max(date)-100)   
  
## 주말효과가 있는 선 trace 추가  
p1 <- total\_deaths\_5\_nations\_since\_100day |>  
 plot\_ly() |>  
 add\_trace(type = 'scatter', mode = 'lines',   
 x = ~date, y = ~new\_cases ,   
 linetype = ~location, connectgaps = T  
)  
  
## 주말효과가 없는 선 trace 추가  
p2 <- total\_deaths\_5\_nations\_since\_100day |>  
 plot\_ly() |>  
 add\_trace(type = 'scatter', mode = 'lines',   
 x = ~date, y = ~new\_cases ,   
 linetype = ~location, connectgaps = T) |>   
 layout(xaxis = list(  
 ## rangebreaks의 설정  
 rangebreaks=list(  
 ## 제거기간을 일요일부터 화요일 이전까지 패턴 설정  
 list(bounds=list("sun", "tue")),   
 ## 제거날짜에 크리스마스 포함  
 list(values=list('2022-03-02'))  
 )  
 )  
 )  
  
subplot(p1, p2, nrows = 2) |>  
 layout(title = '',   
 hovermode = "x unified")



1. <https://www.reneshbedre.com/blog/correlation-analysis-r.html> [↑](#footnote-ref-20)