1. 분포와 시간의 시각화 - 미완성

## C:\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)

## <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

# 분포의 시각화

앞선 2장에서는 plotly의 기존 구조를 이해하기 위해 다소 복잡한 방법으로 plolty 객체를 생성하고 시각화하는 방법에 대해 알아보았다. 또 전체 구조를 이해하기 위해 가장 기본적으로 사용되는 산점도를 사용하였지만 plotly에서는 이것보다 매우 쉬운 인터페이스 함수들을 제공한다. 이 인터페이스 함수들은 시각화하고자 하는 형태에 따라 함수들이 제공되기 때문에 시각화 형태에 따라 plolty 시각화의 방법을 살펴보도록 하겠다.

R에서 시각화의 종류에 따른 함수는 plotly 함수에 포함된 add\_\*()를 사용한다. 각각의 ‘trace’ 종류에 따라 사용하는 함수들이 제공되는데 먼저 plot\_ly()로 객체를 초기화하고 표현하고자 하는 ’trace’를 add\_\*()를 사용하여 계속 ’trace’를 추가하는 방식으로 시각화를 구현한다.

python에서는 plotly 패키지의 서브모듈인 plotly.express에서 제공하는 함수들을 사용할 수 있다. plotly.express에서 제공하는 함수들은 앞 장에서 살펴본 plotly.graphic\_object의 add\_trace() 보다 직관적이고 사용하기 쉽다. plotly를 제공하는 제작사에서는 plotly로 시각화 객체를 만들때 plotly.graphic\_object를 사용하기 보다는 plotly.express를 사용하기를 권장한다. plotly.express로 만든 객체도 결국 plotly.graphic\_object를 리턴하기 때문에 plotly.graphic\_object의 기능을 사용하여 다시 세부적인 설정을 할 수 있다. 하지만 plotly.express는 결정적인 몇가지 단점이 있다. 첫 번째 단점은 ’mesh’나 ’isosurface’와 같은 3차원 시각화는 아직 plotly.express는 지원하지 않는다. 두 번째는 여러개의 trace를 가지는 서브플롯, 다중 축의 사용, 여러개의 trace를 가지는 패싯(facet)과 같은 시각화는 plotly.express로 생성하는데 다소 어려움이 있다. 따라서 plotly.express로는 최상위 시각화를 그리고 add\_trace()를 사용하여 plotly.graphic\_object의 trace 추가 함수를 사용하여 계속 추가해주는 방식으로 구현하여야 한다. 또 plotly.graphic\_object에서 제공하는 함수와 plotly.express에서 제공하는 함수의 속성도 다소 차이가 있기 때문에 plotly.express는 사용이 간편하긴 하지만 사용할 때는 사용법을 잘 확인하고 사용해야 한다.[[1]](#footnote-20)

## 산점도(scatter chart)

산점도는 plolty뿐 아니라 데이터 시각화에서 가장 기본적인 시각화 방법이다. 산점도는 X, Y축으로 구성된 데카르트 좌표계 위에 점을 사용하여 데이터의 분포를 표현하는 방법으로 매우 간단한 시각화이지만 데이터의 분포와 데이터의 관계성을 파악하는데 가장 효율적인 시각화이다. 데이터 분석을 시작할 때 대부분의 분석가들이 가장 먼저 시작하는 EDA(Exploratory Data Analysis)의 기초적인 시각화로 사용된다.

산점도는 X, Y의 2차원 축에 매핑되는 두개 혹은 세개의 데이터간의 관계성을 점으로 표현하는 시각화 방법이다. 기본적으로 2차원 공간에 흩어져(scattered) 보이는 형태의 시각화이고, 2차원 축에 매핑되어야 하기 때문에 2개의 변수가 모두 연속형 수치 변수이어야 한다. 산점도는 ’x-y 그래프’라고도 하며, 데이터의 흩어져 있는 형태의 시각화를 통해 데이터의 분포와 관계를 알아보는데 사용되는 방법중에 가장 많이 사용된다.

산점도를 통해 살펴볼 수 있는 패턴이나 상관관계는 보통 다음의 세 가지 정도이다.

* 선형 또는 비선형 상관 관계 : 선형 상관 관계는 데이터의 추세선이 직선을 형성하지만 비선형 상관 관계는 데이터의 추세선이 곡선 또는 기타 형태를 나타냄
* 강한 또는 약한 상관관계 : 강한 상관 관계는 데이터들이 추세선에 가까이 분포하지만 약한 상관 관계는 데이터 들이 추세선에 더 멀리 분포해 있음
* 양의 또는 음의 상관 관계 : 양의 상관 관계는 추세선이 우상향하고(즉, x 값이 증가할 때 y 값이 증가) 음의 상관 관계는 추세선이 우하향함(즉, x 값은 증가할 때 y 값은 감소).

<https://medium.com/@paymantaei/what-is-a-scatter-plot-and-when-to-use-one-2365e774541>

앞서 설명한 바와 같이 X, Y축에 매핑되는 결과가 산점도인데 이를 상관관계의 측면에서 풀어본다면 X축 변수는 독립변수이고 Y축 변수는 종속변수로 볼 수 있다. 하지만 많은 경우 종속 변수를 결정하는 독립 변수는 하나 이상이다. 이렇게 하나 이상의 독립 변수를 표현하기 위해 대부분의 산점도는 점의 색상이나 형태, 크기등을 사용하여 추가적인 독립변수를 표현한다. 이렇게 X축, 색, 점의 형태, 점의 크기를 모두 사용한다면 Y축에 표현되는 종속 변수는 총 4개의 독립 변수로 표현되는 산점도를 그릴 수 있는 것이다. 이중 점의 크기를 사용하는 산점도를 버블 차트라고 한다.

plotly에서 산점도는 스캐터(scatter) 트레이스를 사용하여 구현한다. plotly에서 스캐터 트레이스는 단지 산점도만을 그리는 것이 아니고 X, Y 축에 좌표상으로 표시되는 선 그래프를 포함하며 산점도와 선 그래프에 문자열을 표기하는 시각화까지 포함한다.

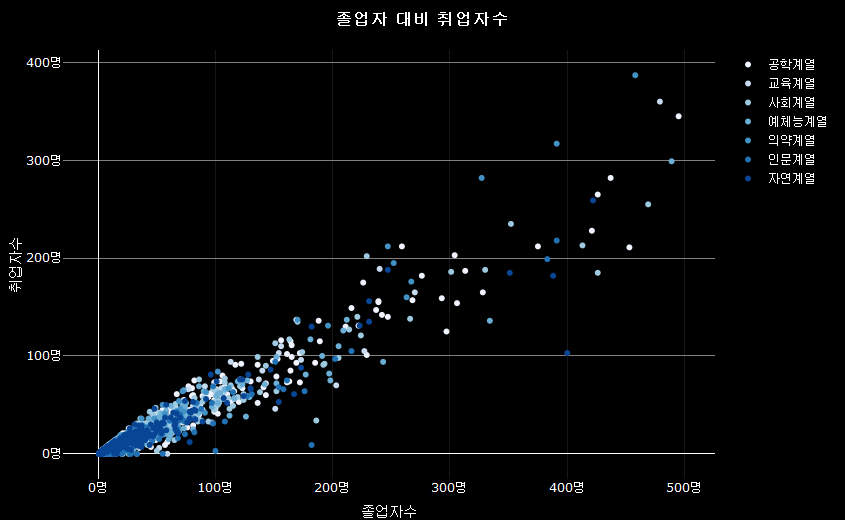
### 기본 산점도 그리기

산점도를 그리는데 앞 장에서 plotly를 그리기 위해 사용했던 add\_trace()를 사용할 수 있지만 R과 python 모두 산점도를 위한 전용 함수를 제공한다.

* R

R에서 산점도는 add\_markers()를 사용하여 만들 수 있다. add\_markers()에서 사용하는 속성은 add\_trace()에서 사용하는 속성 그대로 사용할 수 있다. 만약 이 중 여러개를 동시에 그리기 위해서는 R 자체적으로 제공하는 파이프(|>)나 tidyverse에서 지원하는 파이프(%>%)로 계속 겹쳐서 그릴 수 있다.

df\_취업률\_2000 |>   
 plot\_ly() |>  
 add\_markers(x = ~졸업자수, y = ~취업자수, color = ~대계열) |>  
 layout(title = list(text = '<b>졸업자 대비 취업자수</b>', font = list(color = 'white')),   
 margin = list(t = 50, b = 25, l = 25, r = 25),   
 paper\_bgcolor = 'black', plot\_bgcolor = 'black',   
 xaxis = list(color = 'white', ticksuffix = '명'),   
 yaxis = list(color = 'white', gridcolor = 'gray', ticksuffix = '명', dtick = 100),   
 legend = list(font = list(color = 'white')))



실행결과 2- . color 매핑 결과

* python

python에서는 plotly.express.scatter()로 산점도를 그릴 수 있다. 다만 plotly.express.scatter()에서 사용하는 매개변수는 add\_trace(plolty.graph\_object.Scatter())와 다소 다르다.

import plotly.express as px  
fig = px.scatter(df\_취업률\_2000, x= '졸업자수', y="취업자수",   
 color = "대계열", color\_discrete\_sequence = ("#EFF3FF", "#C6DBEF","#9ECAE1","#6BAED6","#4292C6","#2171B5","#084594"))  
fig.update\_layout(title = dict(text = '<b>졸업자 대비 취업자수</b>', x = 0.5, font = dict(color = 'white')),   
 margin = dict(t = 50, b = 25, l = 25, r = 25),   
 paper\_bgcolor = 'black', plot\_bgcolor = 'black',   
 xaxis = dict(color = 'white', ticksuffix = '명', showgrid = False),   
 yaxis = dict(color = 'white', gridcolor = 'gray', ticksuffix = '명', dtick = 100),   
 legend = dict(font = dict(color = 'white')))  
fig.show()

### 추세선 그리기

이번에는 plotly를 사용하여 직접 추세선을 그리는 과정을 알아보자. plotly에서 추세선의 추가는 python에서는 매우 간단하게 추가할 수 있지만 R에서는 직접적으로 추세선을 그리는 기능을 제공하지 않는다.

* R

R에서 추세선을 그리는데 가장 많이 사용하는 방법은 선형 회귀를 사용하는 방법이다. R에서 많이 사용하는 ggplot2에서는 geom\_smooth()를 사용하여 간단히 추세선을 그릴 수 있지만 plotly에서는 이런 기능을 제공하지 않는다. 따라서 선형회귀(lm)나 국소회귀(loess) 모델을 만들어 추세선을 그려야 한다.

이를 위해서는 X축과 Y축에 설정되는 독립변수와 종속변수 간의 모델을 R base에서 제공하는 lm()과 loess()를 사용하여 만들어야 한다. 이렇게 만든 모델에 fitted()나 predict()를 사용하여 독립변수(X축)에 대응하는 종속변수(Y축)에 대한 추세선을 그려준다. 만약 신뢰구간(Confidence Interval, CI)의 표현이 필요하다면 add\_ribbons()을 사용하여 그려줄 수 있다.

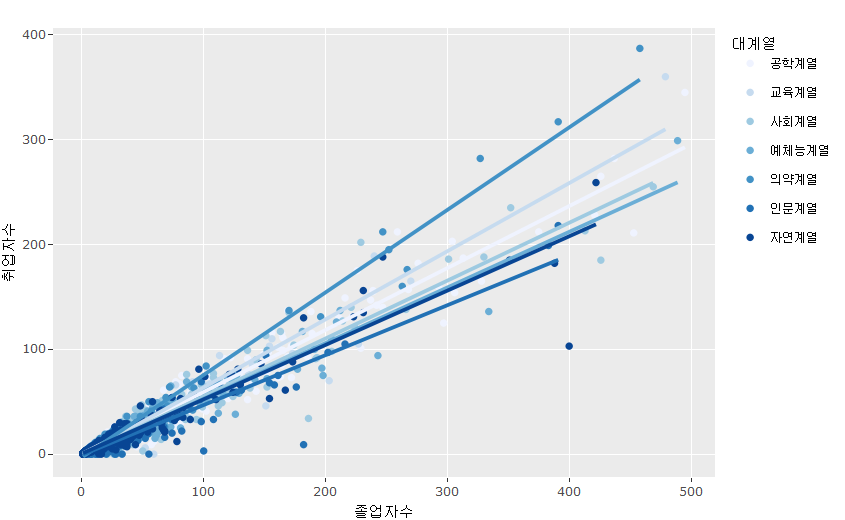
국소회귀 추세선을 plotly로 그리는 방법은 앞서 설명한 바와 같이 loess()을 사용하여 선형회귀 모델을 만들고 이를 fitted()를 사용하여 해당 모델에 대한 적합치를 Y축에 매핑함으로써 그려줄 수 있다. 다만 이 과정에서 X축 변량의 순서대로 fitted() 값을 그려야 정상적인 추세선이 나타나기 때문에 이 데이터를 정렬하기 위해 임시 데이터프레임을 생성하여 사용하였다.

lm\_trend <- lm(data = df\_취업률\_2000, 취업자수 ~ 졸업자수)  
  
loess\_trend <- loess(data = df\_취업률\_2000, 취업자수 ~ 졸업자수)  
  
df\_loess\_trend <- data.frame(X = df\_취업률\_2000$졸업자수, Y = fitted(loess\_trend)) |>  
 arrange(X)  
  
df\_취업률\_2000 |>  
 plot\_ly(type = 'scatter', mode = 'markers') |>  
 add\_trace(x = ~졸업자수, y = ~취업자수, showlegend = FALSE) |>  
 add\_trace(mode = 'lines', x = ~졸업자수, y = ~fitted(lm\_trend), name = '선형 추세선') |>  
 add\_trace(data = df\_loess\_trend\_emp, mode = 'lines', x = ~X, y = ~Y, name = 'loess')

lm\_trend <- lm(data = df\_취업률\_2000, 취업자수 ~ 졸업자수)  
  
loess\_trend <- loess(data = df\_취업률\_2000, 취업자수 ~ 졸업자수)  
  
df\_loess\_trend <- data.frame(X = df\_취업률\_2000$졸업자수, Y = fitted(loess\_trend)) |>  
 arrange(X)  
  
df\_취업률\_2000 |>  
 plot\_ly(type = 'scatter', mode = 'markers') |>  
 add\_trace(x = ~졸업자수, y = ~취업자수, showlegend = FALSE) |>  
 add\_trace(mode = 'lines', x = ~졸업자수, y = ~fitted(lm\_trend), name = '선형 추세선', color = I('blue')) |>  
 add\_trace(data = df\_loess\_trend\_emp, mode = 'lines', x = ~X, y = ~Y, name = 'loess', color = I('skyblue'), dashline = 'dash')

하지만 추세선은 이렇게 전체적인 흐름을 보기위해서도 그리지만 많은 경우 세부 그룹별로 추세선을 그리는 경우도 많다. plotly의 R 패키지에서 자체적으로 추세선을 지원하지 않기 때문에 세부 그룹별로 추세선을 그릴때는 ggplot2로 그린후 plotly로 변환하는 것이 훨씬 효율적이다.

p <- df\_취업률\_2000 |>  
 ggplot(aes(x = 졸업자수, y = 취업자수, color = 대계열)) +  
 geom\_point() +   
 geom\_smooth(method = 'lm', se = FALSE)  
  
ggplotly(p)



* python

python에서는 R보다 훨씬 쉽게 추세선을 그릴 수 있다. px.Scatter()의 ‘trendline’ 속성으로 간단히 그릴 수 있는데 ‘trendline’ 속성에는 ‘ols’(선형회귀), ‘lowess’(국소선형회귀), ‘rolling’(이동평균) 등을 설정할 수 있다. 앞서 R에서 그린 것과 같이 세부 그룹별로 추세선을 그리기 위해서는 ‘color’ 속성으로 그룹화하고 이에 대해 ’trendline’을 설정하면 간단히 그려진다.

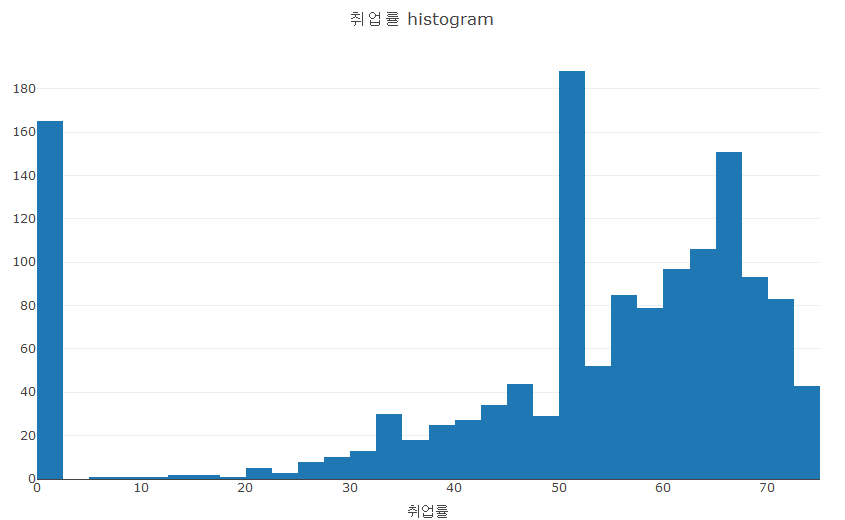
fig = px.scatter(df\_취업률\_2000, x= '졸업자수', y="취업자수",   
 color = "대계열", trendline = 'ols')  
  
fig.show()

## 히스토그램(histogram)

### 기본 히스토그램

* R

## 취업률 데이터를 사용해 plotly 객체 생성  
p\_histogram <- df\_취업률\_2000 |> plot\_ly()  
  
p\_histogram |>   
 ## histogram trace로 X축을 취업률로 매핑, name을 취업률로 설정  
 add\_histogram(x = ~취업률, name = '취업률',   
 xbins = list(start = 0, end = 75, size = 2.5)) |>  
 ## 제목과 여백 설정  
 layout(title = '취업률 histogram', margin = list(t = 50, b = 25, l = 25, r = 25))



실행결과 2- . 히스토그램 trace 시각화

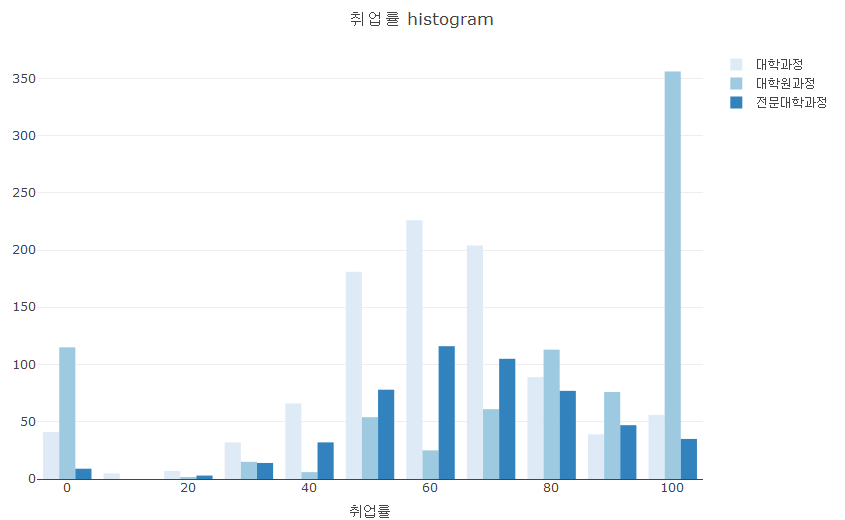
* python

fig = go.Figure()  
  
fig.add\_trace(go.Histogram(x = df\_취업률\_2000['취업률'], name = '취업률',   
 xbins = dict(start = 0, end = 75, size = 2.5)))  
  
fig.show()

### 다중 히스토그램

* R

## 취업률 데이터를 사용해 plotly 객체 생성  
p\_histogram <- df\_취업률\_2000 |> plot\_ly()  
  
p\_histogram |>   
 ## histogram trace로 X축을 취업률로 매핑, name을 취업률로 설정  
 add\_histogram(x = ~취업률, color = ~과정구분,   
 xbins = list(size = 10)) |>  
 ## 제목과 여백 설정  
 layout(title = '취업률 histogram',   
 margin = list(t = 50, b = 25, l = 25, r = 25)  
)



실행결과 2- . 히스토그램 trace 시각화

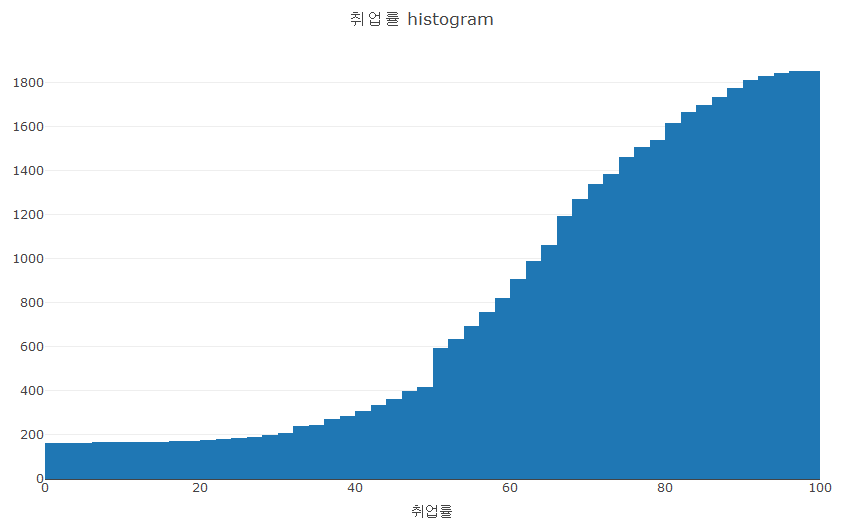
* python

fig = go.Figure()  
  
fig.add\_trace(go.Histogram(x = df\_취업률\_2000.loc[df\_취업률\_2000['과정구분'] == '전문대학과정', '취업률'],   
 name = "전문대학과정", xbins = dict(size = 10)  
 )  
 )  
  
fig.add\_trace(go.Histogram(x = df\_취업률\_2000.loc[df\_취업률\_2000['과정구분'] == '대학과정', '취업률'],   
 name = '대학과정', xbins = dict(size = 10)  
 )  
 )  
  
fig.add\_trace(go.Histogram(x = df\_취업률\_2000.loc[df\_취업률\_2000['과정구분'] == '대학원과정', '취업률'],   
 name = '대학원과정', xbins = dict(size = 10)  
 )  
 )  
  
fig.show()

### 누적 히스토그램

* R

## 취업률 데이터를 사용해 plotly 객체 생성  
p\_histogram <- df\_취업률\_2000 |> plot\_ly()  
  
p\_histogram |>   
 ## histogram trace로 X축을 취업률로 매핑, name을 취업률로 설정  
 add\_histogram(x = ~취업률, name = '취업률',  
 xbins = list(start = 0, end = 100, size = 2),   
 cumulative = list(enabled=TRUE)) |>  
 ## 제목과 여백 설정  
 layout(title = '취업률 histogram', margin = list(t = 50, b = 25, l = 25, r = 25))



실행결과 2- . 히스토그램 trace 시각화

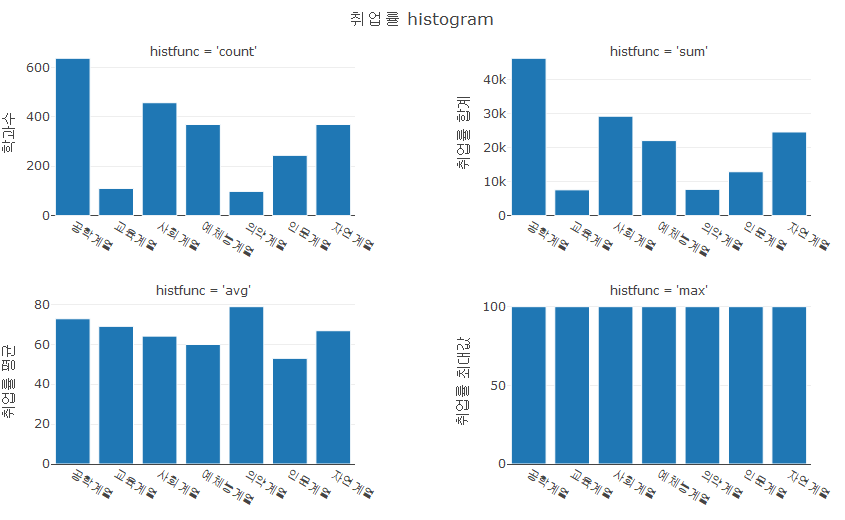
* python

fig = go.Figure()  
  
fig.add\_trace(go.Histogram(x = df\_취업률\_2000['취업률'], name = '취업률',   
 xbins = dict(start = 0, end = 100, size = 2),   
 cumulative = dict(enabled = True)))  
  
fig.show()

### 히스토그램 함수

* R

p\_histogram |>   
 add\_trace(type = 'histogram', ## add\_histogram()과 동의 함수   
 x = ~대계열,   
 ## stroke를 흰색으로, 히그토그램 막대 값을 'count'로 설정  
 stroke = I('white'), histfunc = 'count') |>  
 layout(title = '취업률 histogram', margin = margins,   
 yaxis = list(title = list(text = '학과수')))  
  
p\_histogram |>   
 add\_trace(type = 'histogram', x = ~대계열, y = ~as.character(취업률),   
 ## stroke를 흰색으로, 히그토그램 막대 값을 'sum'으로 설정  
 histfunc = 'sum', stroke = I('white')) |>  
 ## Y축을 선형으로 설정  
 layout(yaxis=list(type='linear',   
 title = list(text = '취업률 합계')),   
 title = '취업률 histogram', margin = margins)  
  
p\_histogram |>   
 add\_trace(type = 'histogram', x = ~대계열, y = ~as.character(취업률),   
 ## stroke를 흰색으로, 히그토그램 막대 값을 'average'로 설정  
 histfunc = 'avg', stroke = I('white')) |>  
 ## Y축을 선형으로 설정  
 layout(yaxis=list(type='linear',   
 title = list(text = '취업률 평균')),   
 title = '취업률 histogram', margin = margins)  
  
p\_histogram |>   
 add\_trace(type = 'histogram', x = ~대계열, y = ~as.character(취업률),  
 ## stroke를 흰색으로, 히그토그램 막대 값을 'max'로 설정  
 histfunc = 'max', stroke = I('white')) |>  
 ## Y축을 선형으로 설정  
 layout(yaxis=list(type='linear', title = list(text = '취업률 최대값')),   
 title = '취업률 histogram', margin = list(t = 50, b = 25, l = 25, r = 25))



실행결과 2- . histfunc에 따른 히스토그램 결과

* python

#################  
fig = go.Figure()  
fig.add\_trace(go.Histogram(x = df\_취업률\_2000['대계열'], y = df\_취업률\_2000['취업률'],  
 histfunc = 'count', showlegend = False))  
fig.show()  
  
#################  
fig = go.Figure()  
fig.add\_trace(go.Histogram(x = df\_취업률\_2000['대계열'], y = df\_취업률\_2000['취업률'],  
 histfunc = 'sum', showlegend = False))  
fig.show()  
  
#################  
fig = go.Figure()  
fig.add\_trace(go.Histogram(x = df\_취업률\_2000['대계열'], y = df\_취업률\_2000['취업률'],  
 histfunc = 'avg', showlegend = False))  
fig.show()  
  
#################  
fig = go.Figure()  
fig.add\_trace(go.Histogram(x = df\_취업률\_2000['대계열'], y = df\_취업률\_2000['취업률'],  
 histfunc = 'max', showlegend = False))  
fig.show()

## 박스(Box) 플롯

박스 trace는 박스 플롯을 생성하기 위해 사용되는 trace이다. 앞 장에서 설명했듯이 박스 플롯은 데이터의 전체적 분포를 4분위수(quantile)과 IQR(Inter Quartile Range)를 사용하여 표시하는 시각화로 연속형 변수와 이산형 변수의 시각화에 사용되는 방법이다. 박스 trace를 사용해 박스 플롯을 생성하기 위해서는 add\_trace(type = 'box')를 사용하거나 add\_boxplot()을 사용한다.

### 평균, 표준편차가 포함된 박스 플롯

df\_취업률 |>   
 plot\_ly() |>   
 add\_boxplot(x = ~대계열, y = ~취업률\_계, boxmean = 'sd', notched = TRUE)|>  
 ## boxmode를 group으로 설정  
 layout(title = list(text = '대학 계열별 취업률 분포'),   
 margin = list(t = 50, b = 25, l = 25, r = 25))

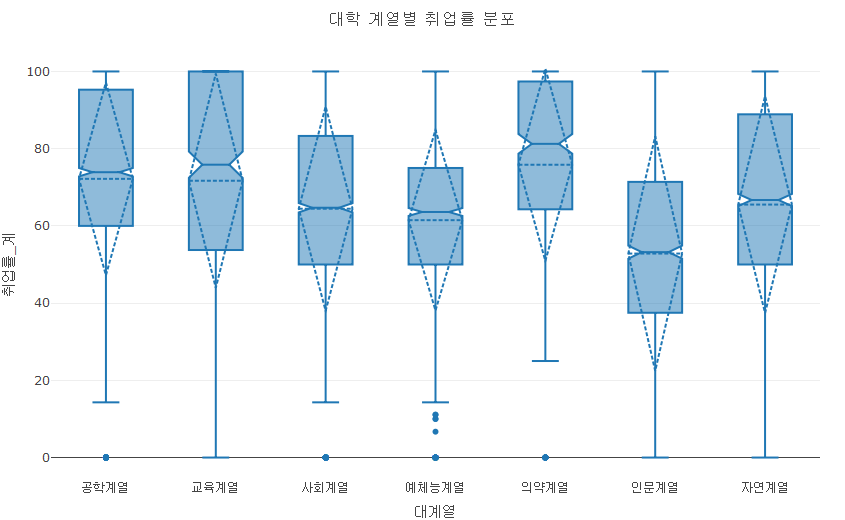


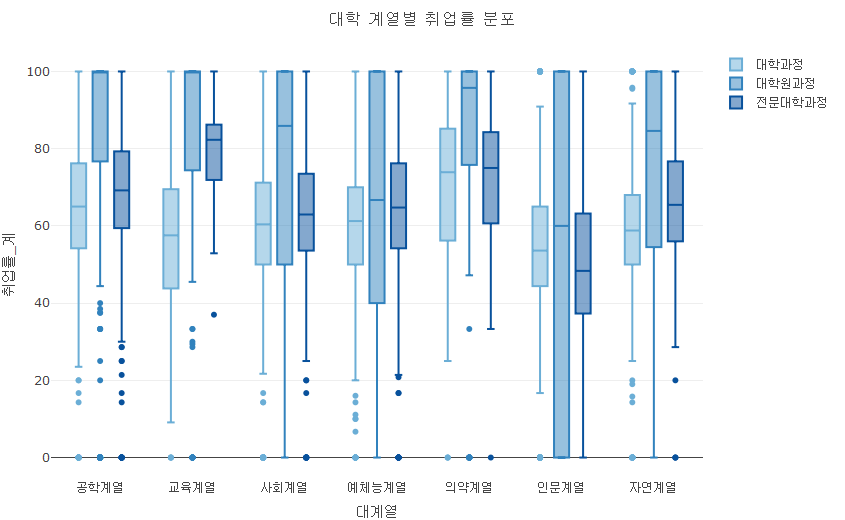
fig = go.Figure()  
fig.add\_trace(go.Box(  
 x = df\_취업률['대계열'], y = df\_취업률['취업률\_계'],   
 boxmean = 'sd', notched = True))

### 그룹 박스 플롯

박스 trace도 막대 trace와 같이 그룹화한 변수를 color나 linetype등의 속성에 매핑시켜서 다수의 박스 trace를 한번에 만들거나 각각의 박스 trace를 추가하여 그릴 수 있다. 이렇게 여러개의 박스 trace를 그릴 때 그 구성 형태를 결정하기 위해서는 차후 설명할 layout()의 boxmode 속성을 설정함으로써 가능하다.[[2]](#footnote-32)

boxmode 속성은 ’group’과 ’overlay’의 두 가지를 설정할 수 있다. ’group’은 각각의 박스들이 옆으로 배치되면서 전체 박스 플롯이 완성되고 ’overlay’는 각각의 박스들이 겹쳐져 그려지면서 완성된다. 다음의 코드는 color로 과정구분을 매핑하여 ’group’형 박스 플롯을 생성하는 코드이다.

df\_취업률 |>   
 plot\_ly() |>   
 add\_boxplot(x = ~대계열, y = ~취업률\_계,   
 ## color를 과정구분으로 매핑  
 color = ~과정구분)|>  
 ## boxmode를 group으로 설정  
 layout(boxmode = "group", title = list(text = '대학 계열별 취업률 분포'),   
 margin = list(t = 50, b = 25, l = 25, r = 25))

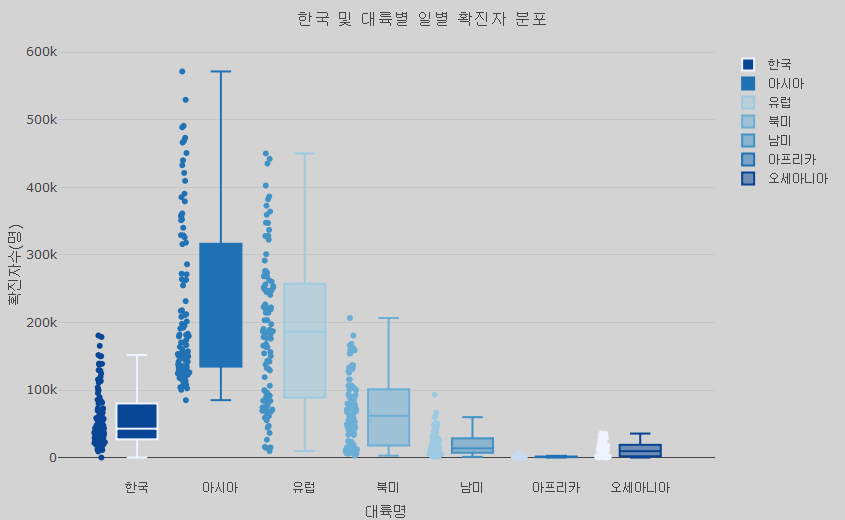


실행결과 2- . 박스 trace의 group mode 실행 결과

fig = go.Figure()  
fig.add\_trace(go.Box(  
 x = df\_취업률.loc[df\_취업률['과정구분'] == '전문대학과정', '대계열'], y = df\_취업률['취업률\_계'],   
 name = '전문대학과정')  
 )  
  
fig.add\_trace(go.Box(  
 x = df\_취업률.loc[df\_취업률['과정구분'] == '대학과정', '대계열'], y = df\_취업률['취업률\_계'],   
 name = '대학과정')  
 )  
  
fig.add\_trace(go.Box(  
 x = df\_취업률.loc[df\_취업률['과정구분'] == '대학원과정', '대계열'], y = df\_취업률['취업률\_계'],   
 name = '대학원과정')  
 )  
  
fig.update\_layout(boxmode = 'group',   
 title = dict(text = '학위과정별 취업률 분포', x = 0.5, xanchor = 'center')  
 )

### 지터(jitter) 박스 플롯

fig <- df\_covid19\_100\_wide |> plot\_ly()  
  
fig <- fig |>   
 add\_boxplot(y = ~확진자\_한국, name = '한국', boxpoints = "all", jitter = 0.3,  
 pointpos = -1.8)  
  
fig <- fig |>   
 add\_boxplot(y = ~확진자\_아시아, name = '아시아', boxpoints = "all", jitter = 0.3,  
 pointpos = -1.8)  
  
fig <- fig |>   
 add\_boxplot(y = ~확진자\_유럽, name = '유럽', boxpoints = "all", jitter = 0.3,  
 pointpos = -1.8)  
  
fig <- fig |>   
 add\_boxplot(y = ~확진자\_북미, name = '북미', boxpoints = "all", jitter = 0.3,  
 pointpos = -1.8)  
  
fig <- fig |>   
 add\_boxplot(y = ~확진자\_남미, name = '남미', boxpoints = "all", jitter = 0.3,  
 pointpos = -1.8)  
  
fig <- fig |>   
 add\_boxplot(y = ~확진자\_아프리카, name = '아프리카', boxpoints = "all", jitter = 0.3,  
 pointpos = -1.8)  
  
fig <- fig |>   
 add\_boxplot(y = ~확진자\_오세아니아, name = '오세아니아', boxpoints = "all", jitter = 0.3,  
 pointpos = -1.8)  
  
 ## boxmode를 group으로 설정  
fig |> layout(title = list(text = '한국 및 대륙별 일별 확진자 분포'),   
 xaxis = list(title = '대륙명'),  
 yaxis = list(title = '확진자수(명)'),  
 margin = list(t = 50, b = 25, l = 25, r = 25),   
 paper\_bgcolor='lightgray', plot\_bgcolor='lightgray')



* python

fig = go.Figure()  
fig.add\_trace(go.Box(  
 y = df\_covid19\_100\_wide['확진자\_한국'], name = '한국',  
 boxpoints = "all", jitter = 0.3, pointpos = -1.8))  
   
fig.add\_trace(go.Box(  
 y = df\_covid19\_100\_wide['확진자\_아시아'], name = '아시아',  
 boxpoints = "all", jitter = 0.3, pointpos = -1.8))  
   
fig.add\_trace(go.Box(  
 y = df\_covid19\_100\_wide['확진자\_유럽'], name = '유럽',  
 boxpoints = "all", jitter = 0.3, pointpos = -1.8))  
   
fig.add\_trace(go.Box(  
 y = df\_covid19\_100\_wide['확진자\_북미'], name = '북미',  
 boxpoints = "all", jitter = 0.3, pointpos = -1.8))  
   
fig.add\_trace(go.Box(  
 y = df\_covid19\_100\_wide['확진자\_남미'], name = '남미',  
 boxpoints = "all", jitter = 0.3, pointpos = -1.8))  
   
fig.add\_trace(go.Box(  
 y = df\_covid19\_100\_wide['확진자\_아프리카'], name = '아프리카',  
 boxpoints = "all", jitter = 0.3, pointpos = -1.8))  
   
fig.add\_trace(go.Box(  
 y = df\_covid19\_100\_wide['확진자\_오세아니아'], name = '오세아니아',  
 boxpoints = "all", jitter = 0.3, pointpos = -1.8))  
   
fig.update\_layout(title = dict(text = '한국 및 대륙별 일별 확진자 분포', x = 0.5),  
 xaxis = dict(title = '대륙명'),  
 yaxis = dict(title = '확진자수(명)'),  
 margin = dict(t = 50, b = 25, l = 25, r = 25),   
 paper\_bgcolor='lightgray', plot\_bgcolor='lightgray')

## 바이올린(Violin) 플롯

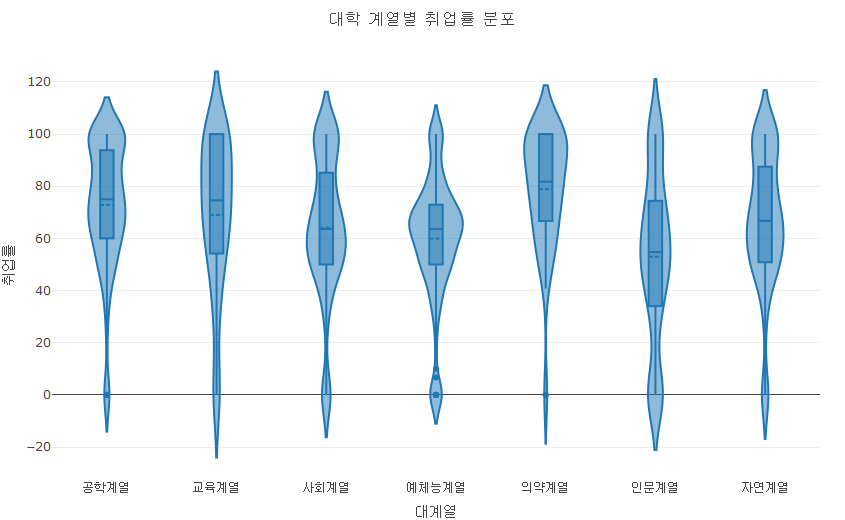
바이올린 trace는 바이올린 플롯을 생성하기 위해 사용되는 trace이다. 앞 장에서 설명했듯이 바이올린 플롯은 박스 플롯에서 확인하기 어려운 데이터의 분포를 확인할 수 있는 시각화 방법이다.

바이올린 trace를 사용해 바이올린 플롯을 생성하기 위해서는 add\_trace(type = 'violin')를 사용하여야하고 래핑함수를 제공하지 않는다.

add\_trace(p, type = ‘violin’, …, data = NULL, inherit = TRUE)  
- p : plot\_ly()로 생성한 plotly 객체  
- type : trace 타입을 ’violin’로 설정  
- … : 바이올린 trace의 line 모드에 설정할 수 있는 속성 설정  
- data : 시각화할 데이터프레임  
- inherit : plot\_ly()에 설정된 속성 type을 상속할지를 결정하는 논리값

### 박스 플롯이 포함된 바이올린 플롯

df\_취업률\_2000 |>   
 plot\_ly() |>   
 ## 바이올린 trace 추가  
 add\_trace(type = 'violin', x = ~대계열, y = ~취업률,   
 box = list(visible = T),  
 meanline = list(visible = T)) |>  
 layout(title = list(text = '대학 계열별 취업률 분포'),   
 margin = list(t = 50, b = 25, l = 25, r = 25))



실행결과 2- . 바이올린 trace 생성

* python

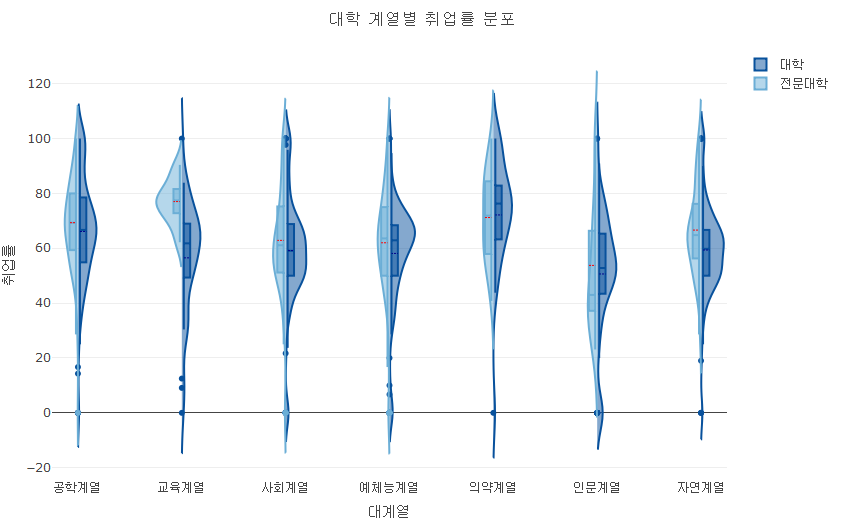
fig = go.Figure()  
fig.add\_trace(go.Violin(  
 x = df\_취업률['대계열'], y = df\_취업률['취업률\_계'],   
 box = dict(visible = True),  
 meanline = dict(visible = True)  
))

### 분리된 바이올린 플롯

앞의 예에서 대학과 전문대학의 바이올린 trace를 각각 추가함으로써 그룹화된 바이올린 플롯을 그렸다. 그런데 이 두 바이올린 플롯을 반씩 붙여서 그리면 바이올린 박스가 반으로 줄기 때문에 데이터를 확인하기 쉬울 것이다.

이렇게 두개의 바이올린 플롯을 반씩 잘라 붙이는 속성이 side이다. side는 바이올린의 양쪽을 다 사용하는 ‘both’, 왼쪽 부분을 사용하는 ‘negative’, 오른쪽 부분을 사용하는 ’positive’를 설정할 수 있다. 그리고 이 두 바이올린을 붙이기 위해 layout()의 violinmode를 overlay로 설정한다. 여기에 앞서 설정한 box와 meanline 설정하면 박스 trace와 평균선도 반으로 그려서 붙여줄 수 있다. 앞서 그렸던 대학과 전문대학의 계열별 바이올린 플롯을 붙이는 코드는 다음과 같다.

p\_violin |>   
 ## 대학과정을 필터링한 데이터 설정  
 add\_trace(data = df\_취업률\_2000 |> filter(과정구분 == '대학과정'),  
 ## 바이올린 trace로 추가  
 type = 'violin', x = ~대계열, y = ~취업률, name = '대학',   
 ## side, box의 설정  
 side = 'positive', box = list(visible = TRUE, width = 0.5),   
 ## meanline의 속성 설정  
 meanline = list(visible = TRUE, color = 'red', width = 1)) |>  
 ## 전문대학과정을 필터링한 데이터 설정  
 add\_trace(data = df\_취업률\_2000 |> filter(과정구분 == '전문대학과정'),   
 type = 'violin', x = ~대계열, y = ~취업률, name = '전문대학',   
 side = 'negative', box = list(visible = TRUE, width = 0.5),   
 meanline = list(visible = TRUE, color = 'red', width = 1)) |>   
 layout(violinmode = "overlay",   
 title = list(text = '대학 계열별 취업률 분포'),   
 margin = margins)



실행결과 2- . 바이올린 trace와 박스 trace의 side 병합

* python

fig = go.Figure()  
fig.add\_trace(go.Violin(  
 x = df\_취업률.loc[df\_취업률['과정구분'] == '전문대학과정', '대계열'], y = df\_취업률['취업률\_계'],   
 name = '전문대학',   
 side = 'positive', box = dict(visible = True, width = 0.5),  
 meanline = dict(visible = True, color = 'red', width = 1)  
))  
  
fig.add\_trace(go.Violin(  
 x = df\_취업률.loc[df\_취업률['과정구분'] == '대학과정', '대계열'], y = df\_취업률['취업률\_계'],   
 name = '대학',   
 side = 'negative', box = dict(visible = True, width = 0.5),  
 meanline = dict(visible = True, color = 'red', width = 1)  
))  
fig.update\_layout(title = dict(text = '대학 계열별 취업률 분포', x = 0.5),   
 margin = dict(t = 50, b = 25, l = 25, r = 25))

fig = go.Figure()  
fig.add\_trace(go.Violin(  
 x = df\_취업률.loc[df\_취업률['과정구분'] == '전문대학과정', '대계열'], y = df\_취업률['취업률\_계'],   
 name = '전문대학',   
 side = 'positive', box = dict(visible = True, width = 0.5),  
 meanline = dict(visible = True, color = 'red', width = 1)  
))  
  
fig.add\_trace(go.Violin(  
 x = df\_취업률.loc[df\_취업률['과정구분'] == '대학과정', '대계열'], y = df\_취업률['취업률\_계'],   
 name = '대학',   
 side = 'negative', box = dict(visible = True, width = 0.5),  
 meanline = dict(visible = True, color = 'red', width = 1)  
))  
fig.update\_layout(title = dict(text = '대학 계열별 취업률 분포', x = 0.5),   
 margin = dict(t = 50, b = 25, l = 25, r = 25),  
 colorway = ('#08519C', '#6BAED6'))

# 시간(Time Series)의 시각화

시간의 시각화는 시간의 흐름에 따른 데이터의 변화를 시각화 한 것이다. 시간의 시각화는 추세(Trend)라고 하는 시간에 따른 데이터의 변화가 발생하는데 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 시간의 흐름에 따른 추세의 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.

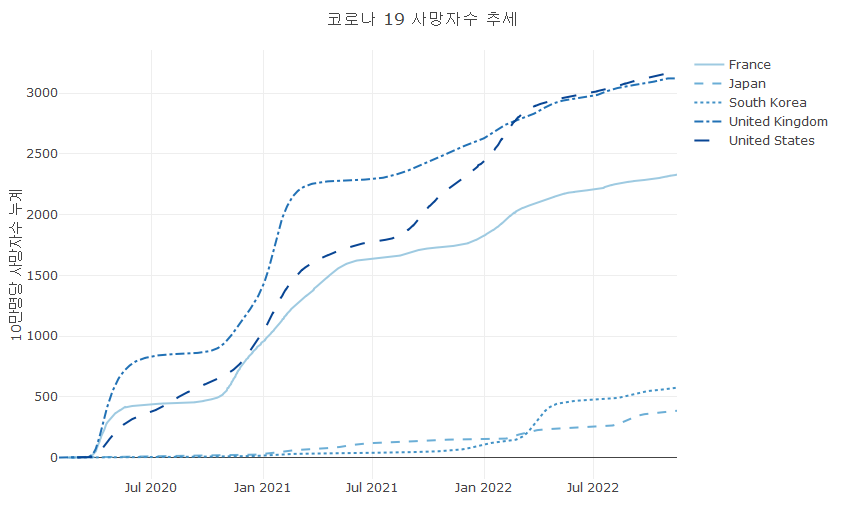
시간을 시각화할 때는 데이터의 포인트와 해당 데이터의 바로 전 데이터와 다음 데이터를 연결하는 선 그래프가 많이 사용되지만 막대 그래프도 많이 사용된다.

## 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.

* R

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)



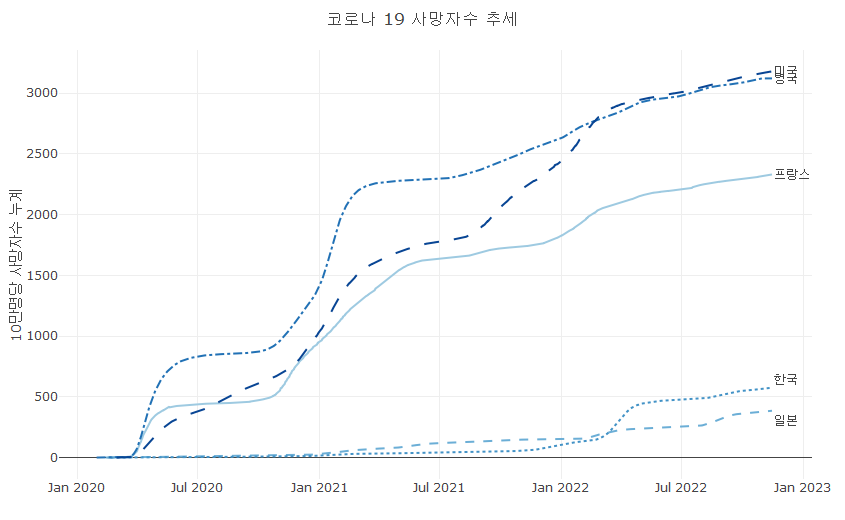
* python

total\_deaths\_5\_nations\_by\_day = df\_covid19.copy()  
total\_deaths\_5\_nations\_by\_day = total\_deaths\_5\_nations\_by\_day[(total\_deaths\_5\_nations\_by\_day['iso\_code'].isin(['KOR', 'USA', 'JPN', 'GBR', 'FRA']))].dropna(subset = ['total\_deaths\_per\_million'])  
  
nations = {'France':'0', 'Japan':'1', 'South Korea':'2', 'United Kingdom':'3', 'United States':'4'}  
  
fig = go.Figure()  
for location, group in total\_deaths\_5\_nations\_by\_day.groupby('location'):  
 fig.add\_trace(go.Scatter(  
 mode = 'lines',   
 x = group['date'],   
 y = group['total\_deaths\_per\_million'],   
 line = dict(dash = nations[location], color = "#08519C"),   
 name = location,  
 connectgaps = True  
 ))  
  
fig.update\_layout(title = dict(text = '코로나 19 사망자수 추세', x = 0.5),   
 xaxis = dict(title = ''),   
 yaxis = dict(title = '10만명당 사망자수 누계'),   
 margin = margins  
 )   
   
fig.show()

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.

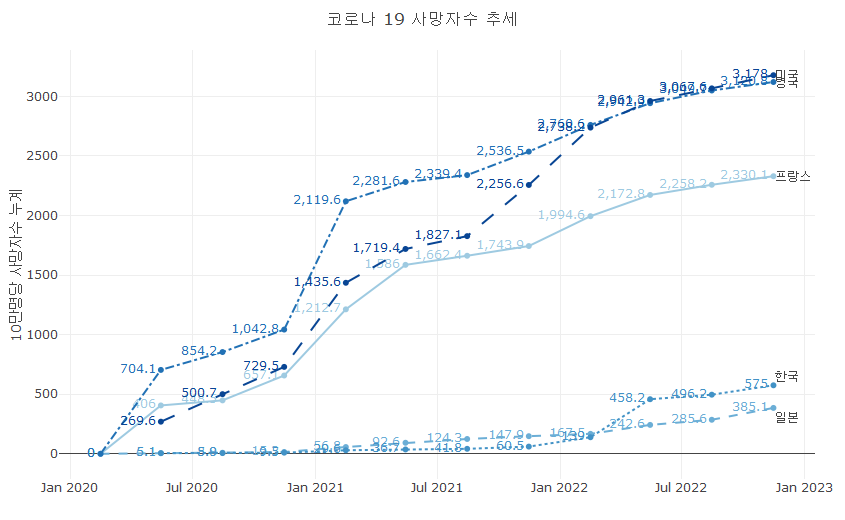
total\_deaths\_5\_nations\_by\_day |>  
 ## plotly 객체 생성  
 plot\_ly() |>  
 add\_trace(type = 'scatter', mode = 'lines+text',   
 x = ~date, y = ~total\_deaths\_per\_million , linetype = ~location, connectgaps = T) |>  
 add\_annotations(text = '프랑스',   
 ## 프랑스 trace의 마지막 위치에 주석 추가  
 x = total\_deaths\_5\_nations\_by\_day |> 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')

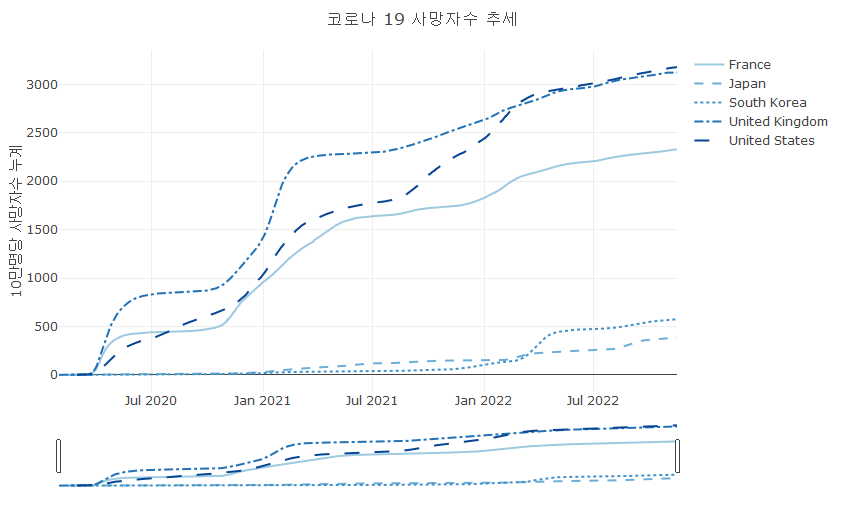


## rangeslider를 사용한 선 그래프

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)



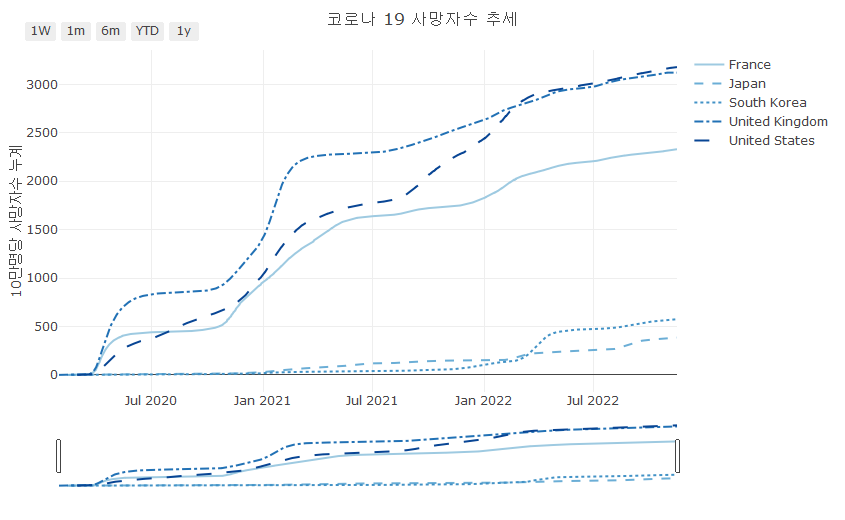
## 기간 설정 버튼(rangeselector)을 사용한 선 그래프

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  
 )

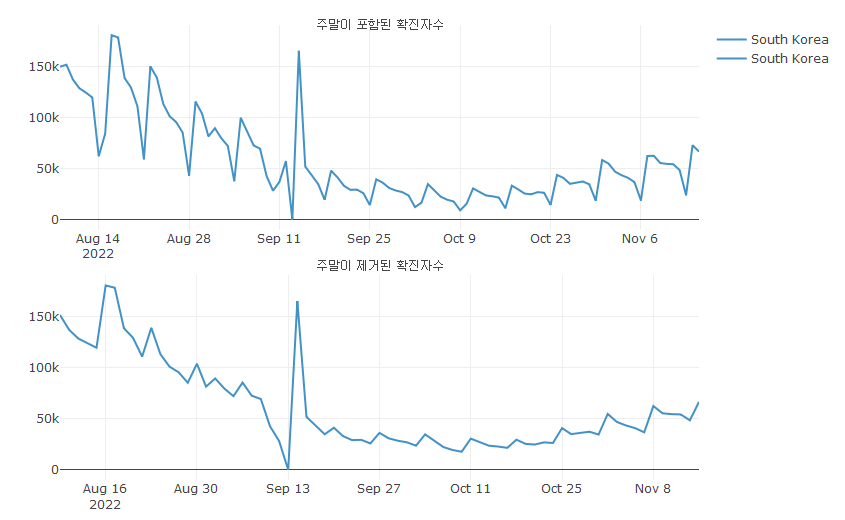


## 주말 효과가 제거된 선 그래프

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")



## 주가를 표현하는 선 그래프

1. 본 책에서는 plotly.graph\_object위주로 설명한다. plotly.express의 사용법은 <https://plotly.com/python-api-reference/> 을 참조하라. [↑](#footnote-ref-20)
2. boxmode를 사용할 때 ’Warning message’를 내는 경우가 있는데 이는 Plotly Community Forum에서도 적절치 않은 경고 메세지로 지적되고 있는데 무시해도 되는 메세지이다. [↑](#footnote-ref-32)