시간과 흐름 시각화

시간과 흐름의 시각화는 시간이나 특정 이벤트, 서로 다른 데이터의 변화에 따라 발생하는 변화와 흐름을 시각화 한 것이다. 시간의 시각화는 추세(Trend)라고 하는 시간에 따른 데이터의 변화를 관찰하는데 매우 용이하다. 하지만 추세는 꼭 시간의 흐름에 종속되지는 않는다. 예를 들자면 회차(물론 이 또한 시간의 흐름과 무관하지 않지만)나 이벤트의 발생과 같은 데이터의 순차적 발생도 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

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.

선 그래프는 간단하고 이해하기 쉽고 효율적이므로 한 번에 많은 데이터 비교하거나, 시간 경과에 따른 변화 및 추세 표시, 중요한 컨텍스트 및 주석 포함하는 시각화, 예측 데이터 및 불확실성의 표시, 데이터 시리즈 내 및 데이터 시리즈 전반의 이상 징후 강조 등에 사용된다. 하지만 단순한 사물의 수량 표시, 이산형 데이터 작업 일부와 전체 비교를 위한 시각화, 데이터가 희소한 데이터 세트의 시각화에는 적합하지 않다.

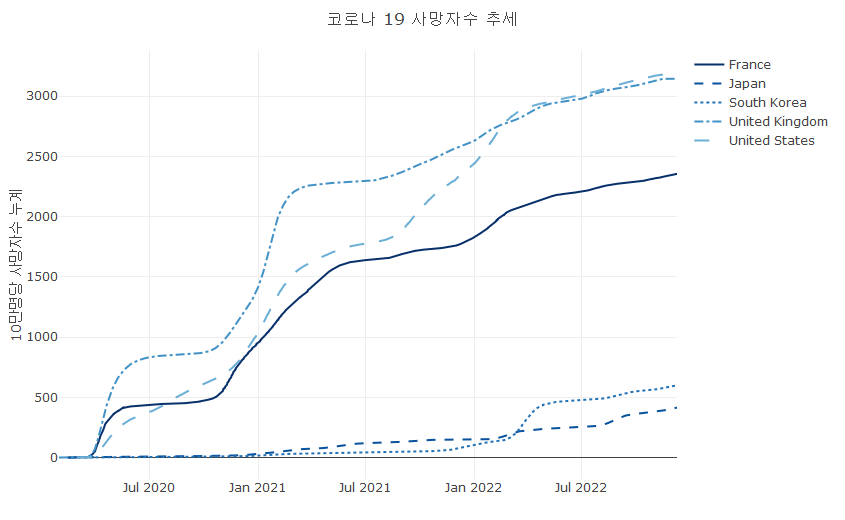
선 그래프 (또는 꺾은 선형 차트)는 각각의 시간에 관측된 데이터 포인트들을 같은 변수이나 변량끼리 선으로 연결하였기 때문에 그 기본은 산점도에 있다고 할수도 있다.

plolty에서 선 그래프를 위해 트레이스를 제공하지 않는다. 앞서 산점도를 그릴때 사용했던 scatter 트레이스의 ’mode’에 ’lines’를 추가해 줌으로써 그릴 수 있다. 다음은 코로나 19의 인구 10만명당 사망자수 누적값의 추세를 나타내는 R과 python 코드이다. 시각화를 위해 먼저 우리나라, 미국, 일본, 영국, 프랑스 다섯 국가의 데이터만 필터링한 데이터를 사용하기 위해 전처리하였다. 각 국가의 선을 보다 쉽게 구분하기 위해 선의 형태를 다르게 설정하였고, 중간중간 빈 데이터를 이어주도록 속성을 설정하였다.

* R

R에서 선 그래프를 그리기 위해서는 add\_trace(type = 'scatter', mode = 'lines', ...)을 사용하거나 add\_lines()를 사용한다.

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\_R)



* 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]),   
 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'))   
   
fig.show()

## 주석을 사용하는 선 그래프

이 시각화를 보면 5개 국가의 사망수의 흐름이 잘 표현되어 있다. 하지만 하나 아쉬운 점은 범례를 사용하여 각 선에 해당하는 국가를 나타내고 있는데 선에 따른 국가를 확인하기 위해서는 범례와 데이터 선을 번갈아 찾아야 한다는 불편함이 따른다. 그래서 선 그래프를 사용하는 많은 경우에서 범례를 사용하기 보다는 선 옆에 바로 범례를 표현하는 방법을 사용한다. 또 다른 시각화에도 많이 사용되지만 선 그래프에서는 시간의 흐름을 표시하는 경우가 많기 때문에 시간의 흐름에 따라 발생되는 다양한 이벤트를 표시하는 경우도 많다. 예를 들어 우리나라의 코로나 19 확진자수는 설이나 추석 등의 장기 연휴가 끝나면 급증하는 경향을 보인다. 또 특정한 정책이 시행되는 날의 이후 시계열적 데이터의 강한 변동이 보이는 경우도 많다. 따라서 이러한 달력상의, 사회상의 이벤트를 표시하는 경우가 많다.

이렇게 plotly에서 범례를 선 그래프 뒤에 붙여주는 기능이나 시간축상의 혹은 데이터 축상의 특정한 이벤트를 표시하는 것은 주석(annotation)으로 처리하는 방법을 써야 한다.

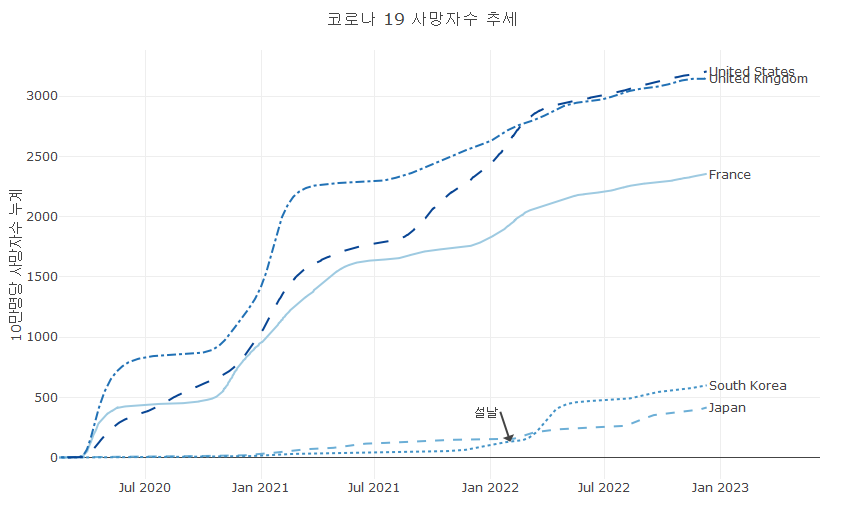
주석은 플롯 위에 표시되는 텍스트 기반 정보를 말한다. 주석이 위치하는 좌표는 플롯의 상대 좌표나 그래프의 실제 데이터 좌표를 기준으로 위치를 지정하는데, 기본적으로 주석은 주석위 정확한 위치를 표시하는 화살표를 사용한다.

plotly에서 제공하는 주석은 ‘data’로 표시되는 트레이스도, ’layout’으로 표시되는 레이 아웃에도 속하지 않는 ’layer’에 속한다. ’layer’는 ’data’나 ’layout’과 같이 plolty를 구성하는 필수적인 요소는 아니다. 따라서 ’data’와 ’layout’의 보완적 요소로써 ’annotation’, ‘shape’, ‘images’ 등의 요소들을 사용할 수 있다.

다음은 5개국의 코로나 19 누적 사망자수를 표시하는 선 그래프이다. 앞의 선 그래프와 다른 것은 범례를 사용하지 않고 데이터의 끝에 해당 데이터의 해당 데이터의 이름을 붙여주는 R과 python 코드이다. 먼저 선 그래프의 오른쪽에 범례를 표시하기 위해 Y축의 오른쪽에 여분의 공간이 필요하다. 이를 위해 마지막 날짜의 180일 후 날짜를 산출하였다. 다음으로 scatter 트레이스의 ‘line’ 모드를 설정하고 5개국의 코로나 19 누적 사망자수의 선 그래프를 그리고 범례를 없앤다. 이 후 각각 데이터의 마지막 날 데이터의 위치에 ‘xanchor’ 가 ‘left’로, ’textposition’이 ’middle right’로 설정된 각각의 데이터 이름을 주석(annotation)으로 붙여준다. 여기서 ’showarrow’를 FALSE로 설정하여 화살표를 없애고 좌표점에 데이터를 표시하도록 하였다. 다음으로 2022년 설날(’2022-02-01’)의 한국 데이터의 위치를 화살표로 ’설날’로 표시하였다.

* R

last\_day = max(distinct(total\_deaths\_5\_nations\_by\_day, date) |> pull()) + 180  
  
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) |>  
 add\_annotations(   
 x =~ (total\_deaths\_5\_nations\_by\_day |> filter(date == max(date)) |> select(date) |> pull()),   
 y = ~(total\_deaths\_5\_nations\_by\_day |> filter(date == max(date)) |> select(total\_deaths\_per\_million) |> pull()),  
 text = ~(total\_deaths\_5\_nations\_by\_day |> filter(date == max(date)) |> select(location) |> pull()),   
 textposition = 'middle right', xanchor = 'left', showarrow = FALSE  
 ) |>  
 add\_annotations(   
 x = '2022-02-01',   
 y = ~(total\_deaths\_5\_nations\_by\_day |> filter(date == '2022-02-01', iso\_code == 'KOR') |> select(total\_deaths\_per\_million) |> pull()),  
 text = '설날',   
 textposition = 'middle right', xanchor = 'right'  
 ) |>  
 layout(title = '코로나 19 사망자수 추세',   
 xaxis = list(title = '', range = c('2020-02-15', format(last\_day, format="%Y-%m-%d"))),   
 yaxis = list(title = '10만명당 사망자수 누계'),   
 margin = margins\_R,  
 showlegend = FALSE)



실행결과 IV - R. 범례를 데이터 옆에 직접 표기한 선 그래프

* python

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]),   
 name = location,  
 connectgaps = True, showlegend = False  
 ))  
 fig.add\_trace(go.Scatter(  
 mode = 'text',  
 x = group.loc[group['date'] == group['date'].max(), 'date'],   
 y = group.loc[group['date'] == group['date'].max(), 'total\_deaths\_per\_million'],  
 text = group.loc[group['date'] == group['date'].max(), 'location'],   
 showlegend = False,   
 textposition = 'middle right'  
 ))   
  
fig.add\_annotation(  
 x = '2022-02-01',   
 y = total\_deaths\_5\_nations\_by\_day.loc[(total\_deaths\_5\_nations\_by\_day['date'] == '2022-02-01') & (total\_deaths\_5\_nations\_by\_day['iso\_code'] == 'KOR'), 'total\_deaths\_per\_million'].values[0],  
 text = '설날', showarrow=True, arrowhead=1, arrowsize = 1.5  
)   
  
fig.update\_layout(title = dict(text = '코로나 19 사망자수 추세', x = 0.5),   
 xaxis = dict(title = '',   
 range = [total\_deaths\_5\_nations\_by\_day['date'].min(), total\_deaths\_5\_nations\_by\_day['date'].max() + timedelta(days=150)]),   
 yaxis = dict(title = '10만명당 사망자수 누계'))   
fig.show()

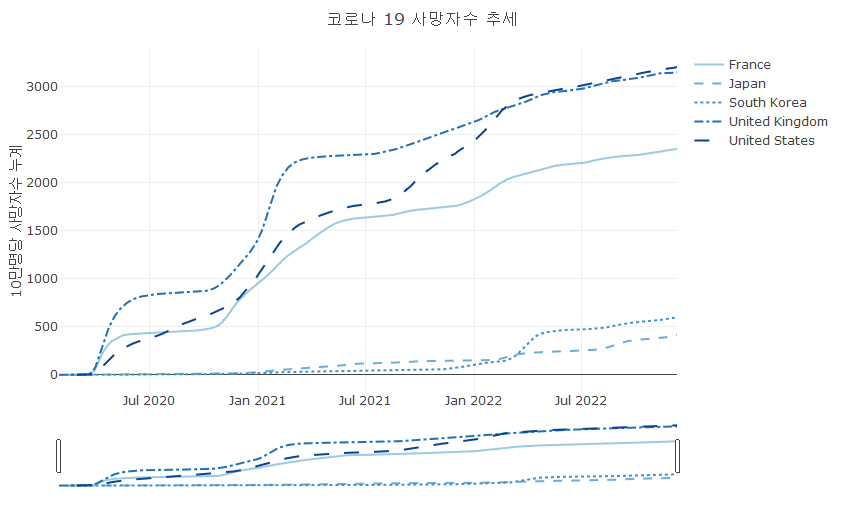
## rangeslider를 사용한 선 그래프

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.

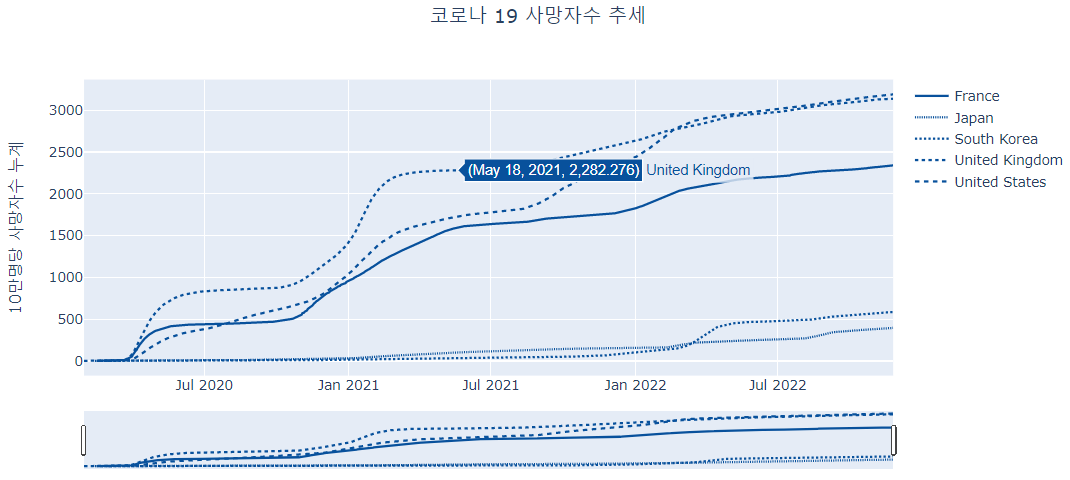
* R

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\_R,   
 title='Time Series with Rangeslider',  
 margin = margins\_R)



* python

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]),   
 name = location,  
 connectgaps = True  
 ))  
  
fig.update\_layout(title = dict(text = '코로나 19 사망자수 추세', x = 0.5),   
 xaxis = dict(title = '', rangeslider = dict(visible = True)),   
 yaxis = dict(title = '10만명당 사망자수 누계'))   
   
fig.show()



## 기간 설정 버튼(rangeselector)을 사용한 선 그래프

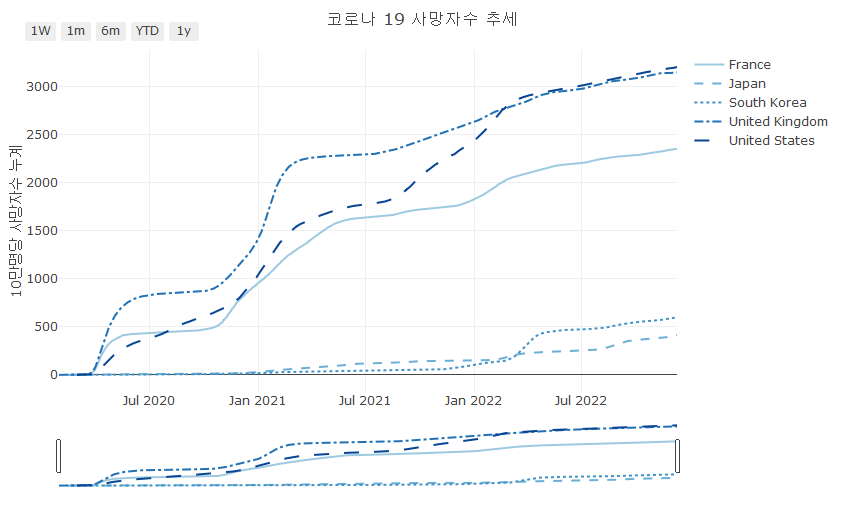
앞서 설명한 rangeslider는 전체 기간중에 특정 기간을 사용자가 직접 설정할 수 있는 장점이 있지만 정확한 기간을 설정하기는 어렵다. 예를 들어 최근 30일, 최근 6개월과 같은 명확한 기간을 설정하고자 할 때는 효과적이지 못하다. 이런 경우를 대비하여 plotly에서 제공하는 기능이 rangeselecor이다. rangeselector는 버튼으로 제공되는데 최근 일에서부터 거꾸로 얼마의 기간 범위를 설정할지를 결정할 수 있다. rangeselector의 button 속성을 설정하기 위해 사용하는 주요 속성은 다음과 같다.

| 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.

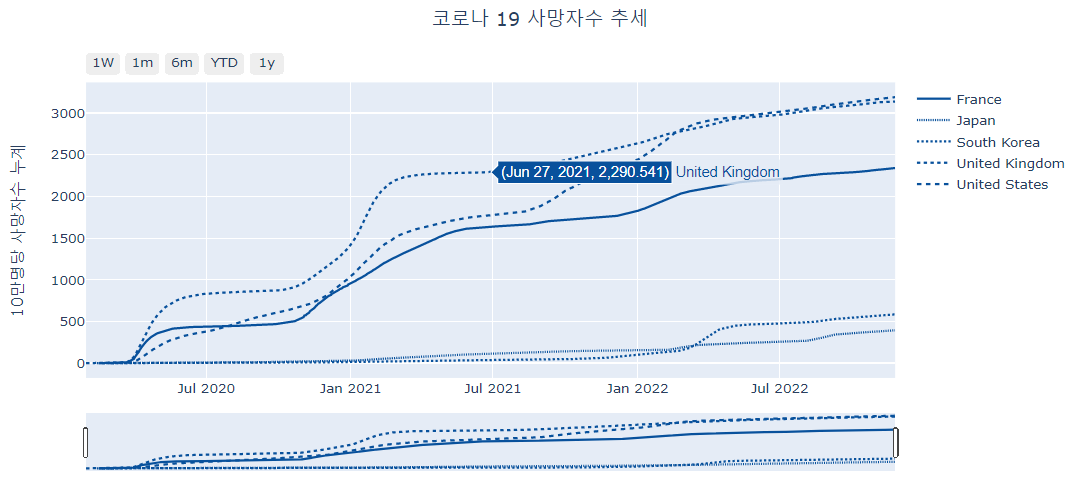
* R

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\_R  
 )



* python

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]),   
 name = location,  
 connectgaps = True  
 ))  
  
fig.update\_layout(title = dict(text = '코로나 19 사망자수 추세', x = 0.5),   
 xaxis = dict(title = '',   
 rangeslider = dict(visible = True),  
 rangeselector=dict(  
 ################################################################# 주의  
 buttons=list([  
 dict(count=7, label="1W", step="day", stepmode="backward"),  
 dict(count=1, label="1m", step="month", stepmode="backward"),  
 dict(count=6, label="6m", step="month", stepmode="backward"),  
 dict(count=1, label="YTD", step="year", stepmode="todate"),  
 dict(count=1, label="1y", step="year", stepmode="backward")  
 ])  
 )  
 ),   
 yaxis = dict(title = '10만명당 사망자수 누계'))   
   
fig.show()

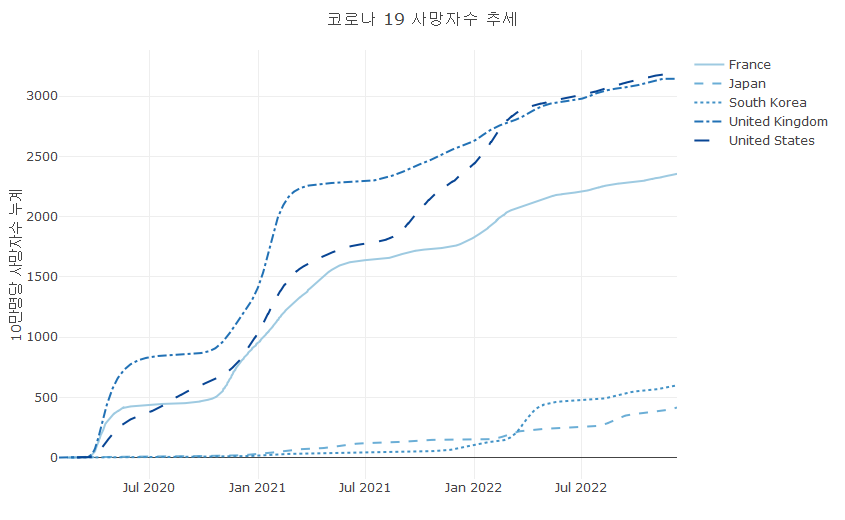


## 호버모드를 사용한 선 그래프

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.

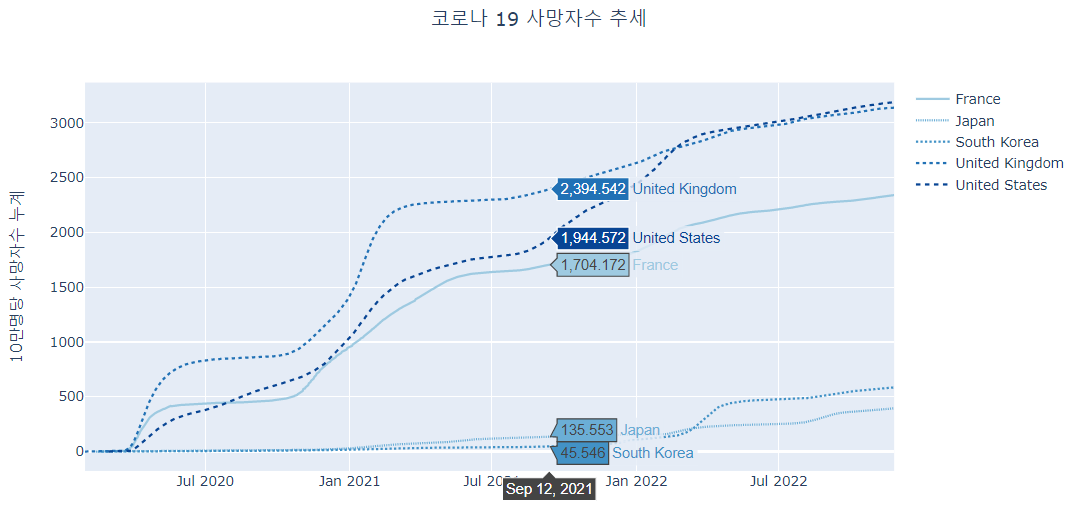
* R

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\_R,   
 hovermode="x")



* python

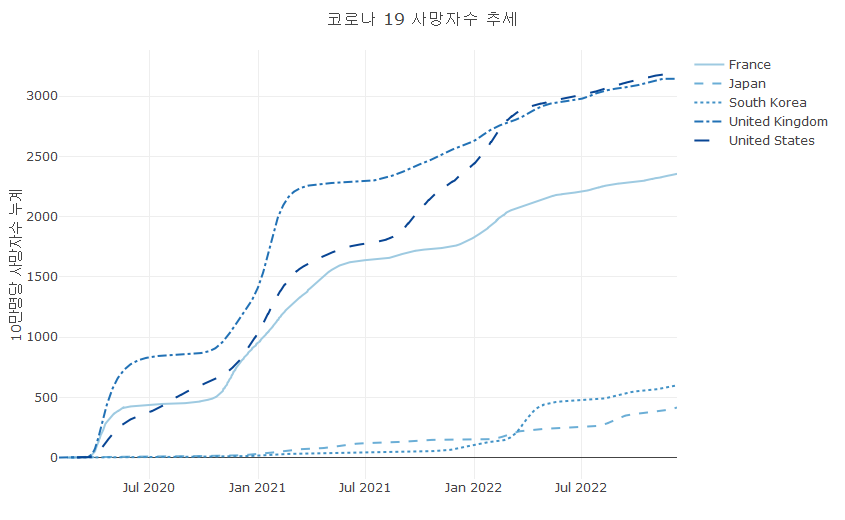
fig = go.Figure()  
for location, group in total\_deaths\_5\_nations\_by\_day.groupby('location'):  
 fig.add\_trace(go.Scatter(  
 mode = 'lines',   
 x = group['date'],   
 y = group['total\_deaths\_per\_million'],   
 line = dict(dash = nations[location]),   
 name = location,  
 connectgaps = True  
 ))  
fig.update\_layout(title = dict(text = '코로나 19 사망자수 추세', x = 0.5),   
 xaxis = dict(title = ''),   
 yaxis = dict(title = '10만명당 사망자수 누계'),  
 hovermode="x")   
fig.show()



## 스파크모드을 사용한 선 그래프

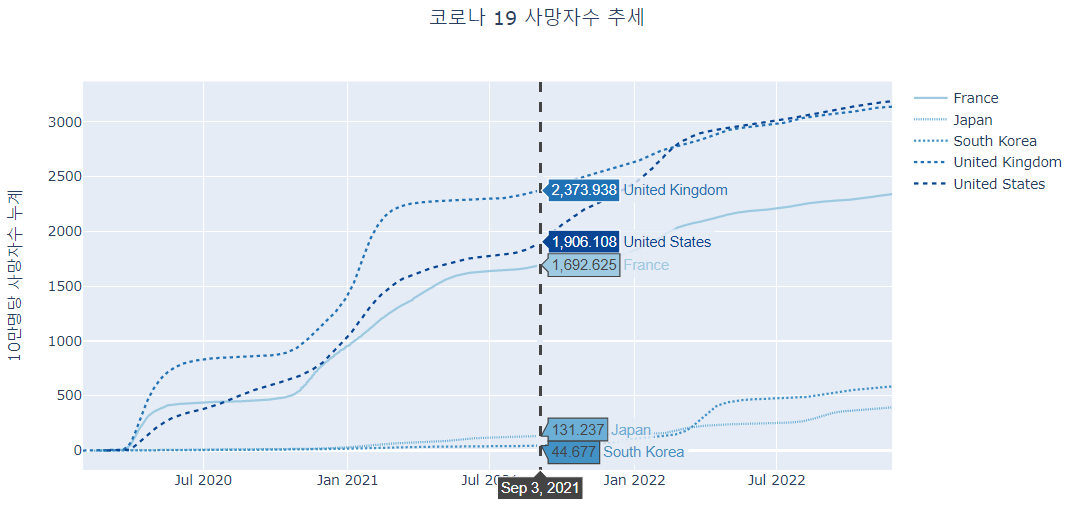
* R

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 = '',   
 spikemode = 'toaxis+across'),   
 yaxis = list(title = '10만명당 사망자수 누계'),   
 margin = margins\_R,   
 hovermode="x")



* python

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]),   
 name = location,  
 connectgaps = True  
 ))  
fig.update\_layout(title = dict(text = '코로나 19 사망자수 추세', x = 0.5),   
 xaxis = dict(title = '',   
 spikemode = 'toaxis+across'),   
 yaxis = dict(title = '10만명당 사망자수 누계'),   
 hovermode="x")   
   
fig.show()



# 시간축의 설정

# 캔들 스틱 차트

우선 삼성전자의 최근 100일 주가를 가져오도록 하겠다.

* R

library(tqk)  
library(lubridate)  
code <- code\_get()  
  
## 삼성전자 코드값을 가져온다.   
sse\_code <- code |> filter(name == '삼성전자') |>  
 select(code) |>   
 pull()  
  
samsung <- tqk\_get(sse\_code, from=today() - 100, to=today())

* python

from pandas\_datareader import data as pdr  
import yfinance as yf  
yf.pdr\_override()  
end\_today = datetime.today()  
start\_day = end\_today - timedelta(days=100)  
end\_today = end\_today.strftime("%Y-%m-%d")  
start\_day = start\_day.strftime("%Y-%m-%d")  
samsung\_stock = pdr.get\_data\_yahoo("005930.KS", start=start\_day, end=end\_today)

plotly에서는 주식차트에서 사용하는 캔들스틱 트레이스를 지원한다. 캔들스틱 트레이스를 사용하기 위해서는 add\_trace()의 ‘type’ 속성을 ‘candlestick’으로 설정하고 ’open’, ‘close’, ‘high’, ‘low’ 값으로 그릴 값이 저장된 열을 매핑해주면 간단히 그려진다.

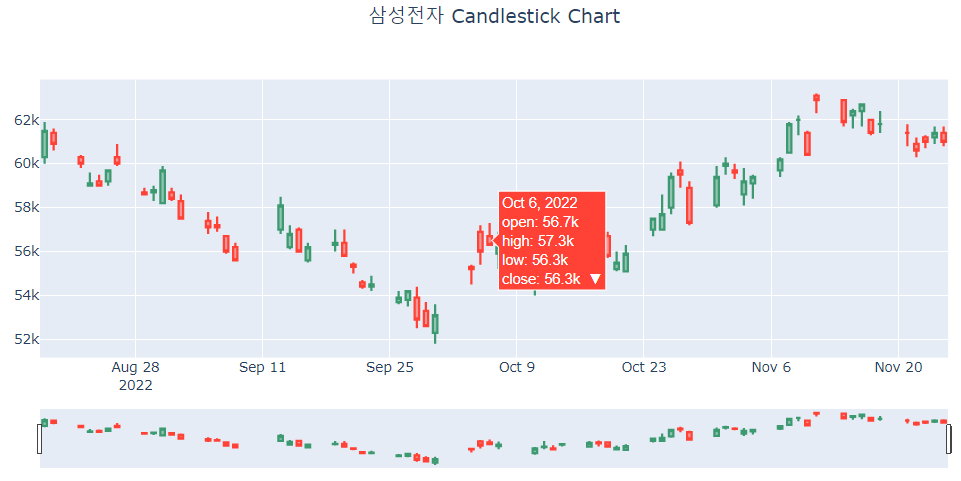
* R

samsung |> plot\_ly() |>  
 add\_trace(  
 type="candlestick", x = ~date,  
 open = ~open, close = ~close,  
 high = ~high, low = ~low) |>   
 layout(title = "삼성전자 Candlestick Chart")



* python

fig = go.Figure()  
fig.add\_trace(go.Candlestick(  
 x = samsung\_stock.index,  
 open = samsung\_stock['Open'], close = samsung\_stock['Close'],  
 high = samsung\_stock['High'], low = samsung\_stock['Low']  
))  
fig.update\_layout(title = dict(text = "삼성전자 Candlestick Chart", x = 0.5))



위의 캔들스틱 그래프에서 보이듯이 기본적으로 시가, 종가, 고가, 저가가 표시되는 캔들스틱 그래프가 만들어졌는데 캔들스틱 그래프 아래에는 rangeslider가 자동적으로 그려져서 주가 기간을 쉽게 설정할 수 있도록 되어 있다. 만약 rangeslider가 없는 캔들스틱 그래프는 다음과 같이 그릴수 있다.

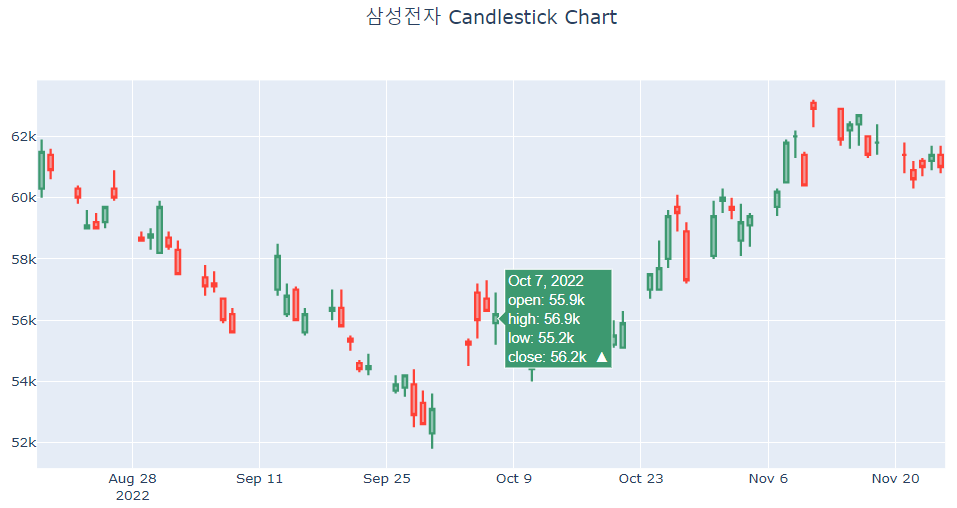
* R

samsung |> plot\_ly() |>  
 add\_trace(  
 type="candlestick", x = ~date,  
 open = ~open, close = ~close,  
 high = ~high, low = ~low) |>   
 layout(title = "삼성전자 Candlestick Chart",   
 xaxis = list(rangeslider = list(visible = F)))



* python

fig = go.Figure()  
fig.add\_trace(go.Candlestick(  
 x = samsung\_stock.index,  
 open = samsung\_stock['Open'], close = samsung\_stock['Close'],  
 high = samsung\_stock['High'], low = samsung\_stock['Low']  
))  
fig.update\_layout(title = dict(text = "삼성전자 Candlestick Chart", x = 0.5),   
 xaxis = dict(rangeslider = dict(visible = False)))

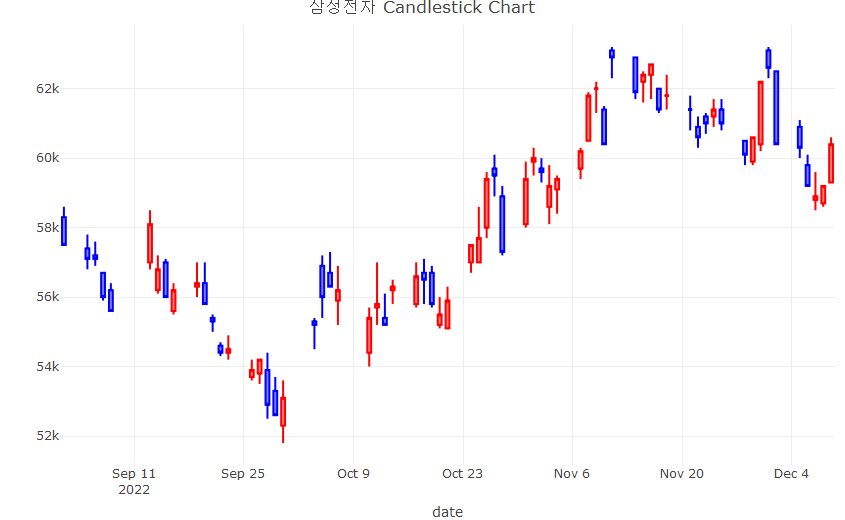


## 캔들 스틱 색 변경

앞에서 그린 캔들스틱은 우리가 흔히 보는 주가 그래프와 조금 다른 점이 있다. 캔들스틱의 색인데 우리나라에서는 파란색과 붉은색으로 표시되는데 plotly에서는 초록색과 붉은색으로 표시된다. 이를 파란색과 붉은색으로 바꾸기 위해서는 ‘increasing’과 ’decreasing’ 속성을 사용한다.

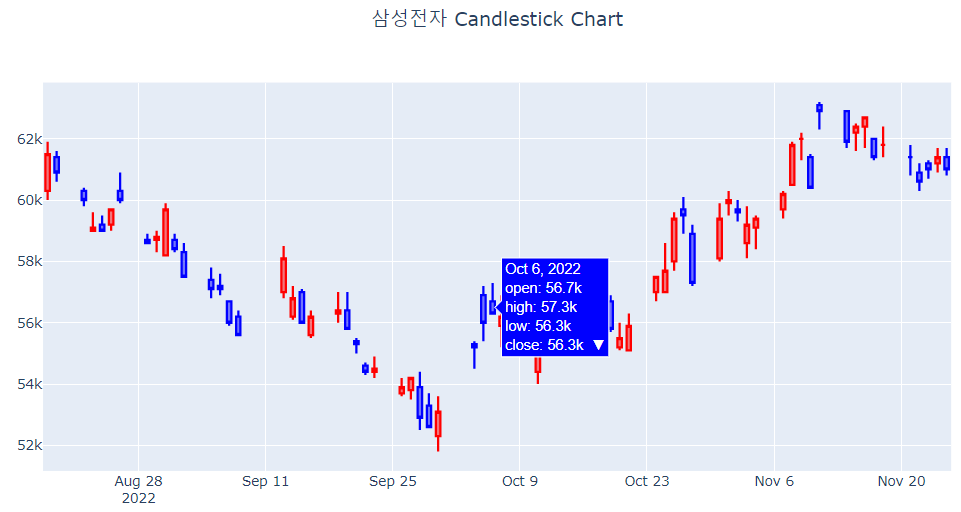
* R

samsung |> plot\_ly() |>  
 add\_trace(  
 type="candlestick", x = ~date,  
 open = ~open, close = ~close,  
 high = ~high, low = ~low,   
 increasing = list(line = list(color = 'red')),   
 decreasing = list(line = list(color = 'blue'))  
 ) |>   
 layout(title = "삼성전자 Candlestick Chart",   
 xaxis = list(rangeslider = list(visible = F)))



* python

fig = go.Figure()  
fig.add\_trace(go.Candlestick(  
 x = samsung\_stock.index,  
 open = samsung\_stock['Open'], close = samsung\_stock['Close'],  
 high = samsung\_stock['High'], low = samsung\_stock['Low'],   
 increasing = dict(line = dict(color = 'red')),   
 decreasing = dict(line = dict(color = 'blue'))  
))  
fig.update\_layout(title = dict(text = "삼성전자 Candlestick Chart", x = 0.5),   
 xaxis = dict(rangeslider = dict(visible = False)))

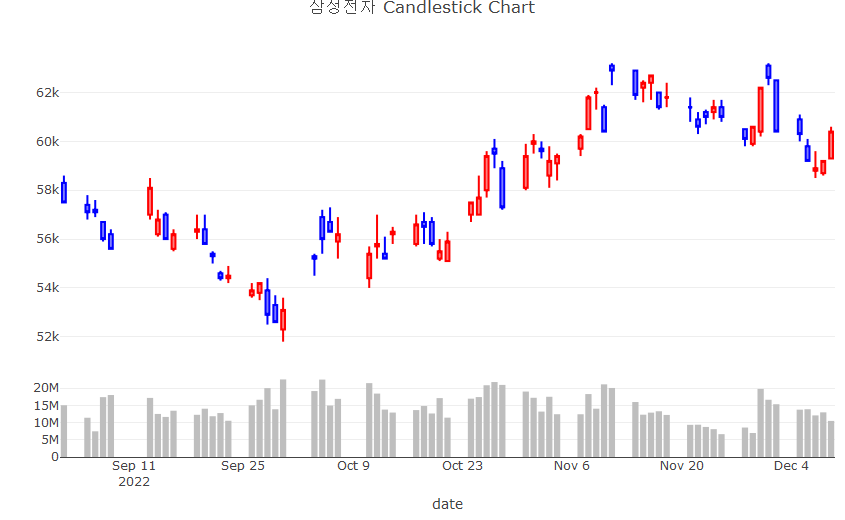


## 거래량 그래프 추가

이제 그래프 하단에 거래량 그래프를 추가해보도록 하겠다. 거래량 그래프를 추가하기 위해서는 subplot()을 사용하여 두 개의 트레이스를 붙여서 그려야 한다. 그래서 막대 트레이스로 거래량 그래프를 그리고 이 그래프를 캔들스틱 차트 아래에 붙이도록 하겠다. 일반적으로 거래량 그래프는 아래쪽에 위치하고 크기는 전체 높이의 20%정도로 설정하고 캔들스틱 차트를 70%, 여백으로 10%정도를 설정하겠다.

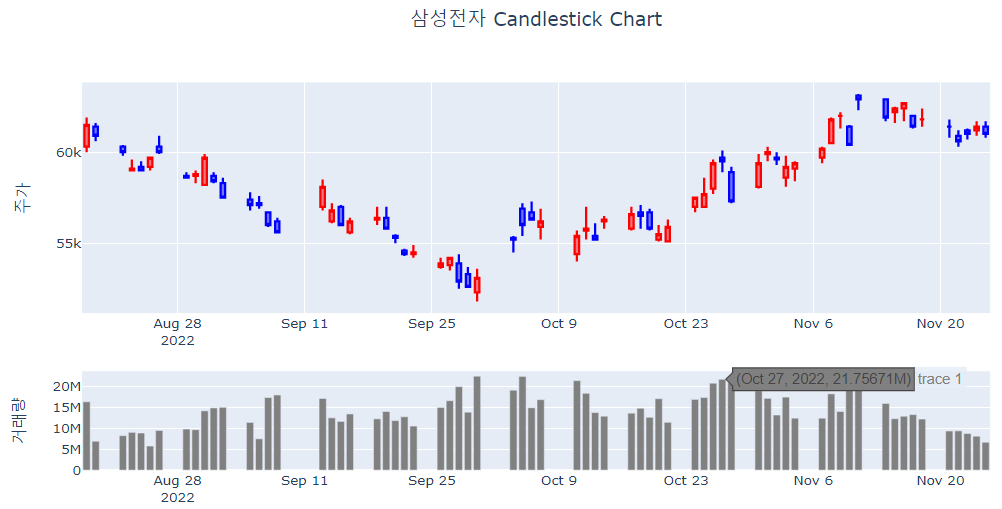
* R

fig1 <- samsung |> plot\_ly() |>  
 add\_trace(  
 type="candlestick", x = ~date,  
 open = ~open, close = ~close,  
 high = ~high, low = ~low,   
 increasing = list(line = list(color = 'red')),   
 decreasing = list(line = list(color = 'blue'))  
 ) |>   
 layout(title = "삼성전자 Candlestick Chart",   
 xaxis = list(rangeslider = list(visible = F)),  
 yaxis = list(title = '주가'),  
 showlegend = FALSE)  
  
fig2 <- samsung %>% plot\_ly() |>  
 add\_trace(type = 'bar', x=~date, y=~volume, type='bar',  
 color = I('gray'), showlegend = FALSE) |>  
 layout(yaxis = list(title = '거래량'))  
  
subplot(fig1, fig2, heights = c(0.7,0.2), nrows=2,  
 shareX = TRUE)



* python

from plotly.subplots import make\_subplots  
  
fig = make\_subplots(rows = 2, cols = 1, row\_heights=[0.7, 0.3])  
  
fig.add\_trace(go.Candlestick(  
 x = samsung\_stock.index,  
 open = samsung\_stock['Open'], close = samsung\_stock['Close'],  
 high = samsung\_stock['High'], low = samsung\_stock['Low'],   
 increasing = dict(line = dict(color = 'red')),   
 decreasing = dict(line = dict(color = 'blue'))  
),   
 row = 1, col = 1)  
  
fig.add\_trace(go.Bar(  
 x = samsung\_stock.index,  
 y = samsung\_stock['Volume'],   
 marker = dict(color = 'gray')  
),   
 row = 2, col =1)  
  
fig.update\_yaxes(title\_text="주가", row=1, col=1)  
fig.update\_yaxes(title\_text="거래량", row=2, col=1)  
  
fig.update\_layout(title = dict(text = "삼성전자 Candlestick Chart", x = 0.5),  
 xaxis = dict(rangeslider = dict(visible = False)),   
 showlegend = False  
)



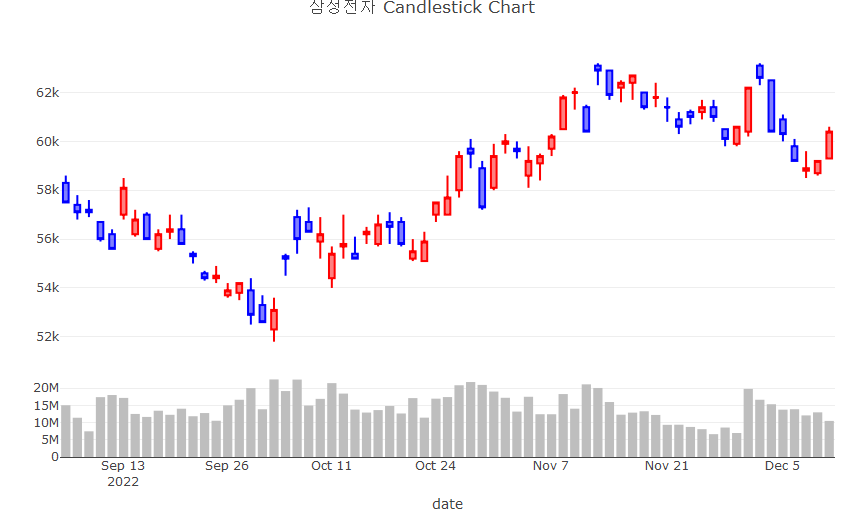
## 주말 효과가 제거된 선 그래프

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 |  |

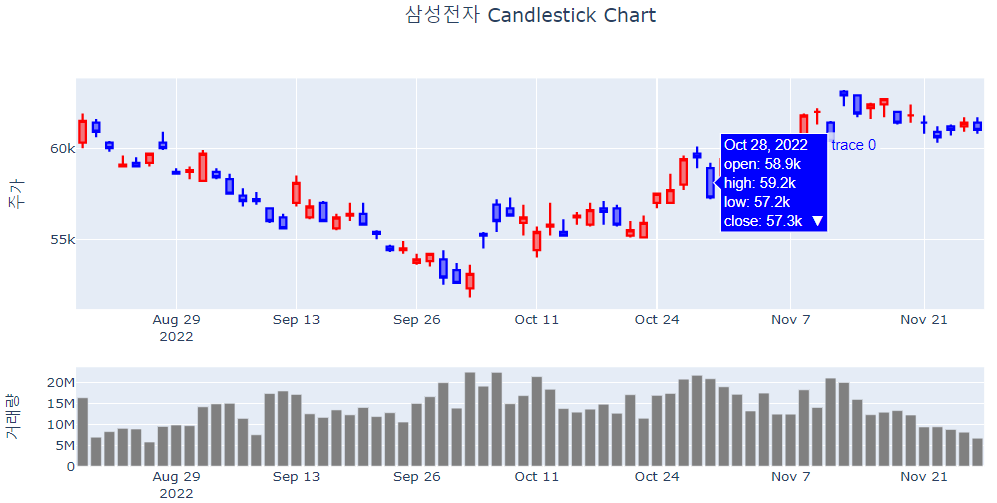
* R

fig1 <- samsung |> plot\_ly() |>  
 add\_trace(  
 type="candlestick", x = ~date,  
 open = ~open, close = ~close,  
 high = ~high, low = ~low,   
 increasing = list(line = list(color = 'red')),   
 decreasing = list(line = list(color = 'blue'))  
 ) |>   
 layout(title = "삼성전자 Candlestick Chart",   
 xaxis = list(rangeslider = list(visible = F),   
 rangebreaks=list(  
 list(bounds=list("sat", "mon")),   
 list(values = list("2022-09-09", "2022-09-12", "2022-10-03", "2022-10-10"))  
 )  
 ),  
 yaxis = list(title = '주가'),  
 showlegend = FALSE)  
  
fig2 <- samsung %>% plot\_ly() |>  
 add\_trace(type = 'bar', x=~date, y=~volume, type='bar',  
 color =I('gray'), showlegend = FALSE) |>  
 layout(xaxis = list(rangebreaks=list(  
 list(bounds=list("sat", "mon")),   
 list(values = list("2022-09-09", "2022-09-12", "2022-10-03", "2022-10-10"))  
 )  
 ),  
 yaxis = list(title = '거래량'))  
  
subplot(fig1, fig2, heights = c(0.7,0.2), nrows=2,  
 shareX = TRUE)



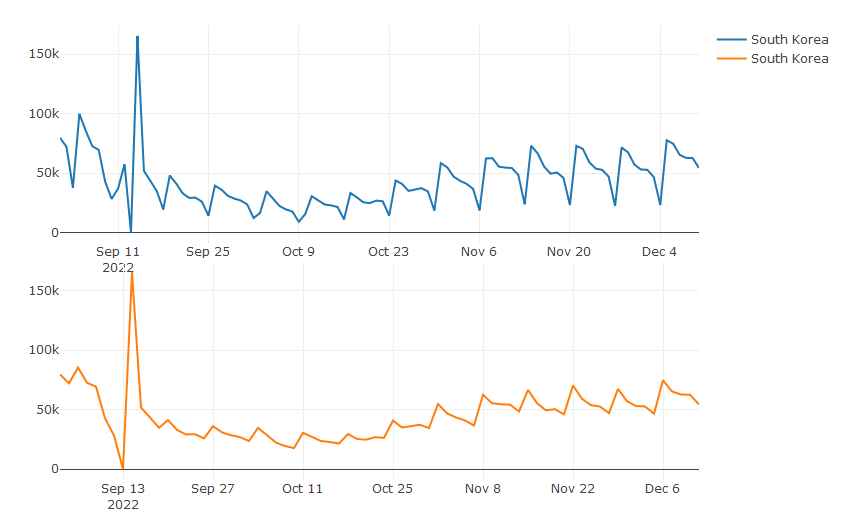
* python

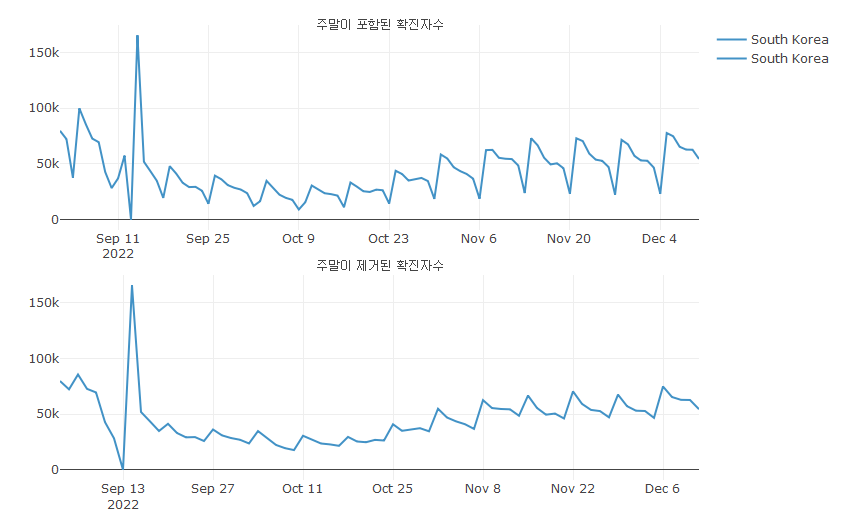
fig = make\_subplots(rows = 2, cols = 1, row\_heights=[0.7, 0.3])  
  
fig.add\_trace(go.Candlestick(  
 x = samsung\_stock.index,  
 open = samsung\_stock['Open'], close = samsung\_stock['Close'],  
 high = samsung\_stock['High'], low = samsung\_stock['Low'],   
 increasing = dict(line = dict(color = 'red')),   
 decreasing = dict(line = dict(color = 'blue'))  
),   
 row = 1, col = 1)  
  
fig.add\_trace(go.Bar(  
 x = samsung\_stock.index,  
 y = samsung\_stock['Volume'],   
 marker = dict(color = 'gray')  
),   
 row = 2, col =1)  
fig.update\_xaxes(rangeslider = dict(visible = False),   
 rangebreaks = [  
 dict(bounds=["sat", "mon"]), #hide weekends  
 dict(values=["2022-09-09", "2022-09-12", "2022-10-03", "2022-10-10"]) # hide Christmas and New Year's  
 ],   
 row = 1, col = 1)  
fig.update\_xaxes(rangeslider = dict(visible = False),   
 rangebreaks = [  
 dict(bounds=["sat", "mon"]), #hide weekends  
 dict(values=["2022-09-09", "2022-09-12", "2022-10-03", "2022-10-10"]) # hide Christmas and New Year's  
 ],   
 row = 2, col = 1)  
fig.update\_yaxes(title\_text="주가", row=1, col=1)  
fig.update\_yaxes(title\_text="거래량", row=2, col=1)  
  
fig.update\_layout(title = dict(text = "삼성전자 Candlestick Chart", x = 0.5),  
 showlegend = False  
)



* R

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")



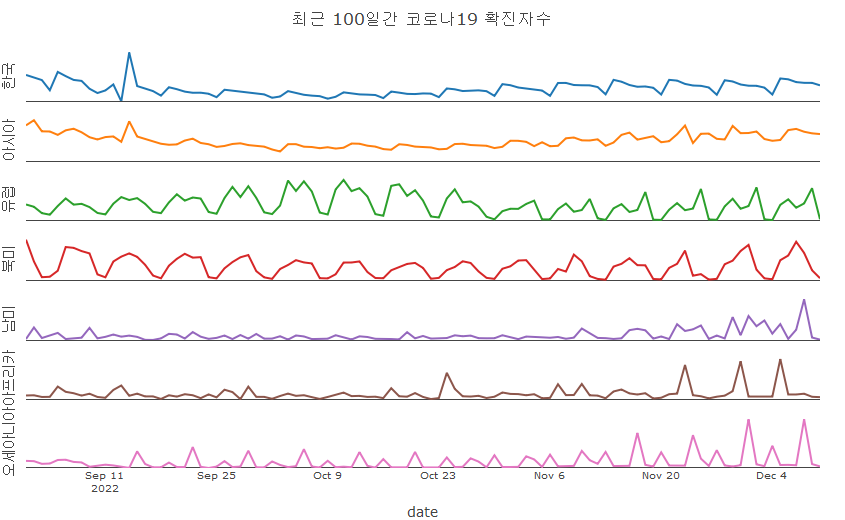


# 스파크라인

스파크라인 그래프는 X, Y축이 생략되고 데이터 흐름이 표현된 선만으로 표시되는 선 그래프를 말한다. 이 그래프는 그 시각화 자체로 사용되기 보다는 다른 정보, 특히 표의 컬럼에 표현되거나 다른 시각화의 보조적 정보의 제공 형태로 사용된다. 특히 많은 선 그래프를 표현하지만 서브 플롯도 다소 많다고 느끼는 경우 사용할 수 있다.

다음은 아시아 국가들의 최근 100일간 코로나19 신규 확진자에 대한 스파크라인 그래프를 그리는 코드이다.

df\_covid19\_100 |>  
 ## 국가명으로 그룹화  
 group\_by(location) |>  
 ## 그룹화한 각각의 데이터 그룹들에 적용할 코드 설정  
 do(  
 ## 각 그룹화한 데이터를 사용해 plotly 객체 생성   
 p = plot\_ly(.) |>   
 ## line 모드의 스캐터 trace 추가  
 add\_trace(type = 'scatter', mode = 'lines',  
 ## X, Y축에 변수 매핑, color를 설정  
 x = ~date, y = ~new\_cases, name = ~location) |>  
 ## layout으로 X, Y축을 설정  
 layout(title = list(title = NULL),  
 xaxis = list(tickfont = list(size = 10),   
 showgrid = FALSE),   
 yaxis = list(title = list(text = ~location),   
 showticklabels = F,   
 showgrid = FALSE,   
 rangemode = 'tozero'))  
 ) |>  
 ## 생성된 plotly 객체들을 subplot 생성  
 subplot(nrows = 7, shareX = TRUE, shareY = TRUE) |>  
 ## 생성된 subplot의 layout 설정  
 layout(showlegend = FALSE,   
 title = '최근 100일간 코로나19 확진자수',  
 margin = margins\_R)



# 퍼널 차트

인구 연령별 인구수를 표현 그래프에서 많이 사용된다. 하지만 plotly에서는 피라미드 그래프라는 이름으로 사용되지 않고 퍼널(깔때기, funnel) trace라는 이름으로 사용된다. 퍼널 trace는 변량의 크기를 길이로 표현한 막대로 표현하고 각 막대의 중간을 맞춰 양쪽으로 퍼져나가는 깔때기 형태의 trace이다. 이 퍼널 차트는 영업 및 마케팅 부서에서 자주 사용되는데 여러 단계별로 값이나 지표가 변화하는 것을 한 눈에 표현하기 위한 목적으로 사용한다.

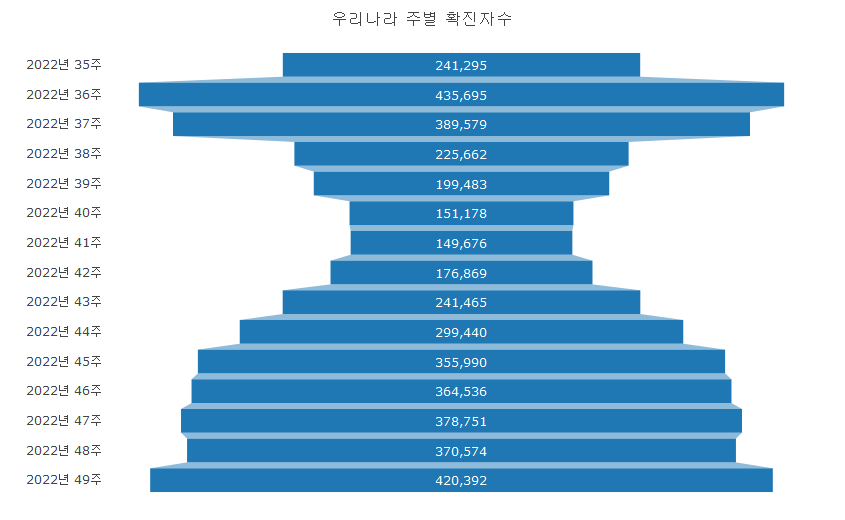
코로나19 데이터를 사용하여 퍼널 차트를 만들기 위해 일간 코로나19 데이터를 주(week) 단위로 요약하여 주 단위로 값의 변화를 살펴보겠다. 먼저 최근 100일간의 우리나라 코로나19 신규 확진자 데이터를 주 단위로 요약하는 전처리는 다음과 같다.

* R

df\_funnel <-   
 df\_covid19\_100 |>   
 filter(iso\_code == 'KOR') |>  
 ## date의 월 단위 열을 yearmonth에 저장  
 mutate(date\_by\_week = lubridate::floor\_date(date, "week"),   
 yearweekth = paste0(lubridate::year(date\_by\_week), '년 ',   
 lubridate::week(date\_by\_week), '주')) |>   
 ## iso\_code, yearmonth로 그룹화  
 group\_by(iso\_code, date\_by\_week, yearweekth) |>  
 ## new\_cases 합계 산출  
 summarise(new\_cases = sum(new\_cases))

전처리된 데이터를 사용하여 퍼널 차트를 그려본다. X축은 데이터 값인 신규 확진자 수를 매핑하고 Y축은 단계로 구분했던 연도의 주(Week) 차수를 매핑함으로서 퍼널 차트를 쉽게 그릴 수 있다.

df\_funnel |>  
 plot\_ly() |>  
 add\_trace(type = 'funnel', x = ~new\_cases, y = ~date\_by\_week,   
 text = ~new\_cases, texttemplate = '%{text:,.0f}') |>  
 layout(title = '우리나라 주별 확진자수',   
 yaxis = list(title = '',   
 tickvals = ~date\_by\_week,   
 ticktext = ~yearweekth),   
 margin = margins\_R)



실행결과 IV - R. 우리나라 주별 확진자 퍼널 차트

* python

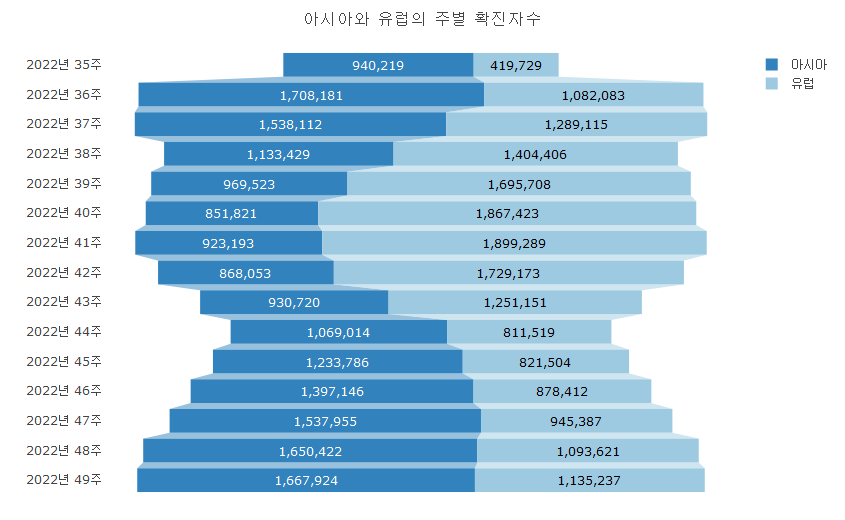
import datetime  
  
df\_funnel = df\_covid19\_100.copy()  
df\_funnel = df\_funnel.loc[(df\_funnel['iso\_code'] == 'KOR')]  
df\_funnel['date\_by\_week'] = df\_funnel['date'].dt.floor('7D')  
df\_funnel['yearweekth'] = df\_funnel['date'].dt.isocalendar().year.astype(str) + '년' + df\_funnel['date'].dt.isocalendar().week.astype(str) + '주'  
df\_funnel = df\_funnel.groupby(yearweekth)['new\_cases'].agg([('new\_cases', 'sum')])  
fig = go.Figure()  
fig.add\_trace(go.Funnel(  
 x = df\_funnel['new\_cases'], y = df\_funnel.index,   
 text = df\_funnel['new\_cases'], texttemplate = '%{text:,.0f}'  
))  
  
fig.update\_layout(title = dict(text = '우리나라 주별 확진자수', x = 0.5),   
 yaxis = dict(title = '',   
 ticktext = df\_funnel.index),   
 margin = margins\_P)

앞선 퍼널 차트에서 보면 우리나라의 신규 확진자 수는 2022년 4주차부터 늘어나기 시작해서 2022년 11주차(3.13~3.19)까지 급격히 증가하고 감소하는 상황이라는 것이 한눈에 보인다.

만약 이 퍼널 차트를 전체 대륙이 한꺼번에 나타나도록 그리기 위해서는 어떻게 해야할까? 여러 변량을 추가하려면 add\_trace()를 추가하면 간단히 그려진다. 다음은 아시아와 유럽의 신규 확진자 주별 데이터에 대한 퍼널 차트이다.

* R

df\_funnel\_Asia <-   
 df\_covid19\_100 |>   
 filter(location == '아시아') |>  
 ## date의 월 단위 열을 yearmonth에 저장  
 mutate(date\_by\_week = lubridate::floor\_date(date, "week"),   
 yearweekth = paste0(lubridate::year(date\_by\_week), '년 ',   
 lubridate::week(date\_by\_week), '주')) |>   
 ## iso\_code, yearmonth로 그룹화  
 group\_by(iso\_code, date\_by\_week, yearweekth) |>  
 ## new\_cases 합계 산출  
 summarise(new\_cases = sum(new\_cases))  
  
df\_funnel\_Europe <-   
 df\_covid19\_100 |>   
 filter(location == '유럽') |>  
 ## date의 월 단위 열을 yearmonth에 저장  
 mutate(date\_by\_week = lubridate::floor\_date(date, "week"),   
 yearweekth = paste0(lubridate::year(date\_by\_week), '년 ',   
 lubridate::week(date\_by\_week), '주')) |>   
 ## iso\_code, yearmonth로 그룹화  
 group\_by(iso\_code, date\_by\_week, yearweekth) |>  
 ## new\_cases 합계 산출  
 summarise(new\_cases = sum(new\_cases))  
  
df\_funnel\_Asia |>   
 plot\_ly() |>  
 add\_trace(type = 'funnel', name = '아시아',   
 x = ~new\_cases, y = ~date\_by\_week,   
 text = ~new\_cases, texttemplate = '%{text:,.0f}') |>  
 add\_trace(data = df\_funnel\_Europe,type = 'funnel', name = '유럽',  
 x = ~new\_cases, y = ~date\_by\_week,   
 text = ~new\_cases, texttemplate = '%{text:,.0f}') |>  
 layout(title = '아시아와 유럽의 주별 확진자수',   
 yaxis = list(title = '',   
 tickvals = ~date\_by\_week,   
 ticktext = ~yearweekth),   
 margin = margins\_R)



* python

import datetime  
  
df\_funnel\_asia = df\_covid19\_100.copy()  
df\_funnel\_asia = df\_funnel\_asia.loc[(df\_funnel\_asia['location'] == '아시아')]  
df\_funnel\_asia['date\_by\_week'] = df\_funnel\_asia['date'].dt.floor('7D')  
df\_funnel\_asia['yearweekth'] = df\_funnel\_asia['date'].dt.isocalendar().year.astype(str) + '년' + df\_funnel\_asia['date'].dt.isocalendar().week.astype(str) + '주'  
df\_funnel\_asia = df\_funnel\_asia.groupby('yearweekth')['new\_cases'].agg([('new\_cases', 'sum')])  
  
df\_funnel\_europe = df\_covid19\_100.copy()  
df\_funnel\_europe = df\_funnel\_europe.loc[(df\_funnel\_europe['location'] == '유럽')]  
df\_funnel\_europe['date\_by\_week'] = df\_funnel\_europe['date'].dt.floor('7D')  
df\_funnel\_europe['yearweekth'] = df\_funnel\_europe['date'].dt.isocalendar().year.astype(str) + '년' + df\_funnel\_europe['date'].dt.isocalendar().week.astype(str) + '주'  
df\_funnel\_europe = df\_funnel\_europe.groupby('yearweekth')['new\_cases'].agg([('new\_cases', 'sum')])  
  
fig = go.Figure()  
  
fig.add\_trace(go.Funnel(  
 x = df\_funnel\_asia['new\_cases'], y = df\_funnel\_asia.index, name = '아시아',   
 text = df\_funnel\_asia['new\_cases'], texttemplate = '%{text:,.0f}'  
))  
  
fig.add\_trace(go.Funnel(  
 x = df\_funnel\_europe['new\_cases'], y = df\_funnel\_europe.index, name = '유럽',  
 text = df\_funnel\_europe['new\_cases'], texttemplate = '%{text:,.0f}',   
))  
  
fig.update\_layout(title = dict(text = '우리나라 주별 확진자수', x = 0.5),   
 yaxis = dict(title = '', ticktext = df\_funnel.index),  
 margin = margins\_P)

퍼널 차트의 또하나의 표현 방식은 퍼널 면적(funnel area) 차트이다. 퍼널 면적 차트가 완전한 피라미드 형태의 차트이다. 앞선 퍼널 차트에서 각각의 막대 사이의 공간을 없애고 삼각형 형태로 표현되며 면적으로 데이터 값을 표현한다.

# 산키 다이어그램

산키 다이어그램은 두개 혹은 두개 이상의 변수간의 데이터 흐름을 잘 보여주는 다이어그램이다. 각각의 변수 항목의 변량들은 왼쪽과 오른쪽에 네모 박스로 표현하고 변량들의 데이터가 연관된 항목간의 데이터 량에 따라 굵기가 다른 선으로 이어지는 형태로 표현되는 다이어그램으로 비교적 최근부터 사용되기 시작한 그래프 형태이다.

최근 plotly에서 제공하는 trace에 sankey를 제공하기 때문에 추가적인 패키지의 도움없이 산키 다이어그램을 만들수 있지만 ggplot2의 경우는 확장 패키지로 만들어진 ggsankey 패키지를 통해 ggplot 객체의 산키 다이어그램을 만들 수 있다. 이외에도 networkD3 패키지를 사용할 수 있다.

산키 다이어그램을 생성하기 위해서는 세가지 데이터가 필요하다.

첫번째는 (네모 박스로 표현되는) 각각 노드의 이름, 두번째는 각각의 노드들이 연결되는 링크에 대한 정보, 세번째는 링크의 굵기가 표현될 데이터 정보이다.

이번 절에서 만들어 볼 산키 다이어그램은 취업통계 데이터를 사용하여 대학 과정(전문대학과정, 대학과정, 대학원과정)에 따른 졸업자가 졸업 후 어떤 진로를 선택했는지(취업, 진학 등)의 흐름을 연결하는 다이어그램이다. 이 산키 다이어그램을 생성하기 위해서는 앞서 설명한 세가지 데이터를 만들기 위해 다음과 같이 전처리하였다.

df\_sankey <- df\_취업률 |>   
 ## 열 중에서 3열(과정구분, 왼쪽 노드로 사용)과 12열, 21열부터 26열(오른쪽 노드로 사용)까지를 선택  
 select(3, 12, 21:26) |>   
 ## 과정구분 열을 사용하여 그룹화  
 group\_by(과정구분) |>  
 ## 전체 열에 대해 `sum`을 적용(summarise\_all은 전체 열에 동일한 요약함수를 적영하는 함수임)  
 summarise\_all(sum) |>  
 ## 열이름을 적절히 변경  
 rename(c('취업' = '취업자\_합계\_계', '진학' = '진학자\_계', '취업불가' = '취업불가능자\_계', '외국인' = '외국인유학생\_계', '제외인정' = '제외인정자\_계', '기타' = '기타\_계', '미상' = '미상\_계')) |>  
 ## 첫번째 열을 제외하고 나머지 열들에 긴 형태의 데이터로 변환, 열 이름이 들어간 열은 '구분'으로 데이터 값이 들어간 열은 '학생수'열로 설정  
 pivot\_longer(cols = 2:8, names\_to = '졸업구분', values\_to = '학생수') |>  
 ## 과정구분 열과 구분 열의 순서설정을 위해 팩터 레벨 설정  
 mutate(과정구분\_node = case\_when(  
 과정구분 == '전문대학과정' ~ 0,   
 과정구분 == '대학과정' ~ 1,  
 과정구분 == '대학원과정' ~ 2),  
 졸업구분\_node = case\_when(  
 졸업구분 == '취업' ~ 3,   
 졸업구분 == '진학' ~ 4,   
 졸업구분 == '취업불가' ~ 5,   
 졸업구분 == '외국인' ~ 6,   
 졸업구분 == '제외인정' ~ 7,   
 졸업구분 == '기타' ~ 8,   
 졸업구분 == '미상' ~ 9)  
 ) |>  
 arrange(과정구분\_node, 졸업구분\_node)  
  
head(df\_sankey, 10)

## # A tibble: 10 x 5  
## 과정구분 졸업구분 학생수 과정구분\_node 졸업구분\_node  
## <chr> <chr> <dbl> <dbl> <dbl>  
## 1 전문대학과정 취업 105904 0 3  
## 2 전문대학과정 진학 11616 0 4  
## 3 전문대학과정 취업불가 73 0 5  
## 4 전문대학과정 외국인 1588 0 6  
## 5 전문대학과정 제외인정 2024 0 7  
## 6 전문대학과정 기타 46058 0 8  
## 7 전문대학과정 미상 1247 0 9  
## 8 대학과정 취업 178215 1 3  
## 9 대학과정 진학 21700 1 4  
## 10 대학과정 취업불가 87 1 5

이 데이터에서 뜻하는 것은 ‘과정구분’(전문대학, 대학, 대학원)의 졸업자가 ‘졸업구분’(취업, 진학 등)으로 몇 명이 배출되었는지를 한 행에 나타낸다. 예를 들어 첫 번쨰 행의 경우 전문대학과정 졸업자 중에 취업으로 간 학생은 105904명이라는 의미이다.

이 데이터에서 앞서 설명한 세가지 데이터 벡터를 생성한다.

첫 번째 벡터는 노드의 이름을 가지는 벡터이다. 노드의 이름은 ‘과정구분’ 열과 ‘졸업구분’ 열의 각각의 변량을 벡터로 만들었다.

## 왼쪽 노드로 사용할 변량을 from에 저장  
from <- unique(as.character(df\_sankey$과정구분))  
  
## 오른쪽 노드로 사용할 변량을 to에 저장  
to <- unique(as.character(df\_sankey$졸업구분))  
  
## 전체 노드 벡터 생성  
node <- c(from, to)  
  
node

## [1] "전문대학과정" "대학과정" "대학원과정" "취업" "진학"   
## [6] "취업불가" "외국인" "제외인정" "기타" "미상"

두번째로는 노드 링크 정보를 생성한다. 노드 링크 정보는 앞서 전처리한 df\_sankey에서 ‘과정구분’ 열에서 ‘졸업구분’열로 이동하는 의미로 구성하였고 각각의 노드에 대한 번호를 만들었 때문에 ’source’를 ’과정구분\_node’ 열 벡터로, ‘target’을 ’졸업구분\_node’ 열로 설정한다.

여기서 중요한 것이 R에서 사용하는 인덱스는 일반적으로 1부터 시작하지만 여기서 사용하는 인덱스는 0부터 시작한다는 것이다.

세번쨰로 노드 데이터 정보를 생성한다. 앞서 데이터를 전처리 할 때 노드 링크의 순서에 따라 데이터를 정리해 놓았기 때문에 df\_sankey의 학생수 열을 사용할 수 있다.

이제 데이터의 전처리가 끝났으니 plotly의 sankey trace를 사용하여 산키 다이어그램을 생성한다.

df\_sankey |> plot\_ly(  
 type = "sankey",  
 orientation = "h",  
  
 node = list(  
 label = node,  
 color = c(rep('lightblue', 3), rep('darkblue', 7)),  
 pad = 15,  
 thickness = 20,  
 line = list(  
 color = "black",  
 width = 0.5  
 )  
 ),  
  
 link = list(  
 source = ~과정구분\_node,  
 target = ~졸업구분\_node,  
 value = ~학생수  
 )  
 ) |>  
 layout(title = '대학과정별 졸업자의 졸업 후 진로',   
 margin = margins\_R)



실행결과 IV - R. 산키 다이어그램