II. 비교(Compare), 차이(difference)와 구성(Composition)의 시각화 - 미완성

1. 비교의 시각화

비교와 분포의 시각화는 데이터를 구성하는 특정 변수의 변량에 따라 데이터 값들간의 순서를 비교하거나 데이터의 구성 비율에 대한 시각화이다. 이 종류의 시각화는 다른 시각화와 다른 두가지 특징이 있다.

첫번째 특징은 시각화 그래프 내에서 한번 더 통계처리가 필요하다는 특성을 가진다. 보통 비교의 시각화는 데이터에 값에 따라 시각화 한 후 그 값들을 다시 정렬함으로써 시각화를 완성하게 된다. 따라서 데이터 시각화 이전이나 이후에 시각화 대상 데이터나 시각화 객체에서 정렬과 같은 통계 처리가 한번 더 일어나게 된다.

두번째 특징은 그 관심의 대상이 데이터 자체의 값 보다는 비교되는 대상 내에서의 상대적 위치에 더 쏠려있다는 점이다. 비교되는 대상 중에 가장 값이 크거나 작은 변량이 무엇인지에 관심이 있는 시각화이다. 비교되는 대상들의 상대적 위치가 명확하게 구분되어야 하기 때문에 비교되는 대상들이 명확하게 구분되어야 하고 이 대상들을 명확하게 구분하기 위해 비교를 위한 변수는 이산형 변수를 사용하는 것이 일반적이다.

비교의 시각화는 막대 그래프가 많이 사용되는데 순위 막대 그래프, 롤리팝 그래프, 도트 그래프 등을 많이 사용한다.

1.1. 막대 그래프

1.1.1. 순위 막대 그래프

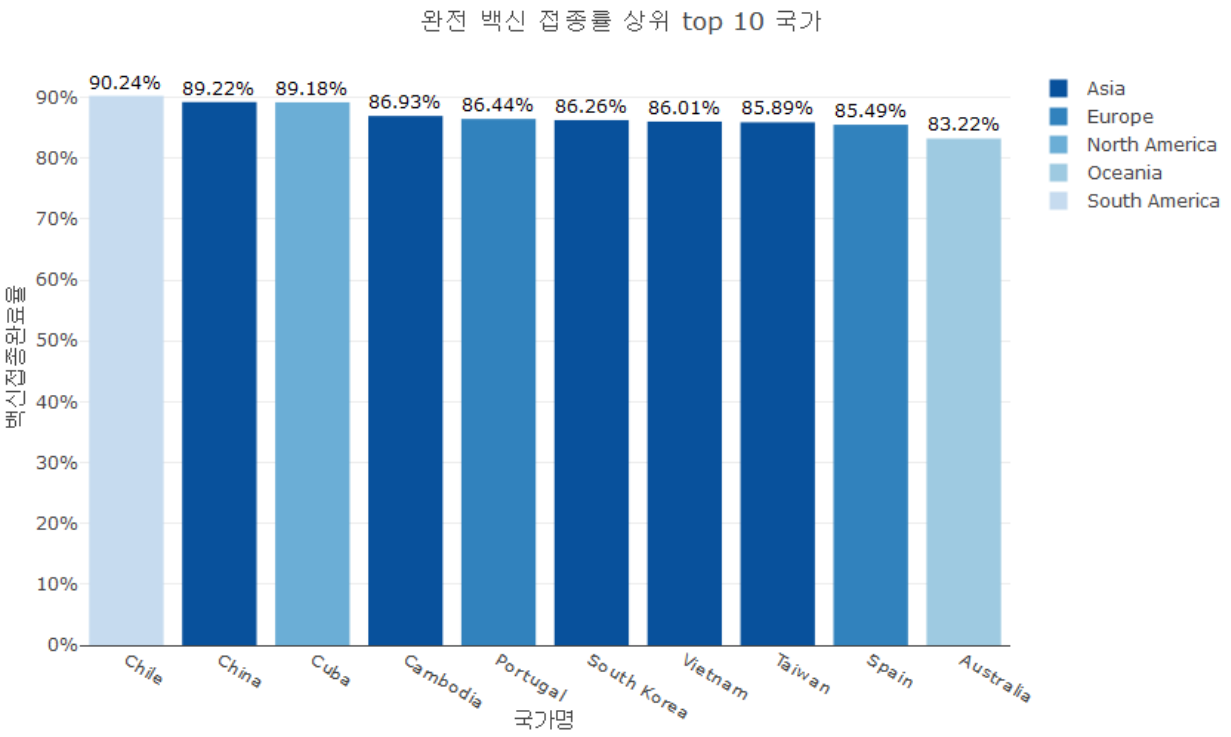
앞서 설명한 바와 같이 비교를 위한 시각화는 특정한 이산형 변수에 의한 수치형 변수를 정렬한 후에 가장 높은 순서부터 혹은 가장 낮은 순서부터 시각화하는 것을 말한다. 이 비교률 위한 시각화로 가장 많이 사용되는 시각화가 막대 그래프이다. `plotly` 로 막대 그래프를 만들기 위해서는 막대 trace 를 추가함으로써 그릴 수 있다. 비교에 사용되는 변수의 변량이

많지 않은 경우는 X 축에 이산형 변수를 매핑하고 Y 축에 비교하기 위한 수치 변수를 매핑하지만 비교에 사용되는 변수의 변량이 많은 경우는 Y 축에 이산형 변수를 매핑하고 X 축에 비교를 위한 변수를 매핑하여 아래쪽으로 수형으로 막대가 그려지는 수평형 막대 그래프를 사용한다.

코로나 19 데이터 셋에서 인구 100 명당 완전 백신 접종자의 수가 가장 많은 10 개의 국가를 시각화하기 위해서는 먼저 인구 100 명당 완전 백신 접종자를 내림차순으로 정렬하고 이중 상위 10 개국에 대해 시각화를 그려준다. `plotly` 에서는 자체적으로 데이터를 정렬하는 기능을 제공하지 않는다. 따라서 정렬해야하는 데이터를 `reorder()`를 사용하여 매핑해줌으로써 정렬이 가능하다. 여기서 인구수가 너무 적은 국가는 백신접종률에 의미가 떨어지기 때문에 인구수가 천만명 이상의 국가를 대상으로 하여 백신접종률이 가장 높은 10 개 국가를 시각화하는 코드는 다음과 같다.

- R

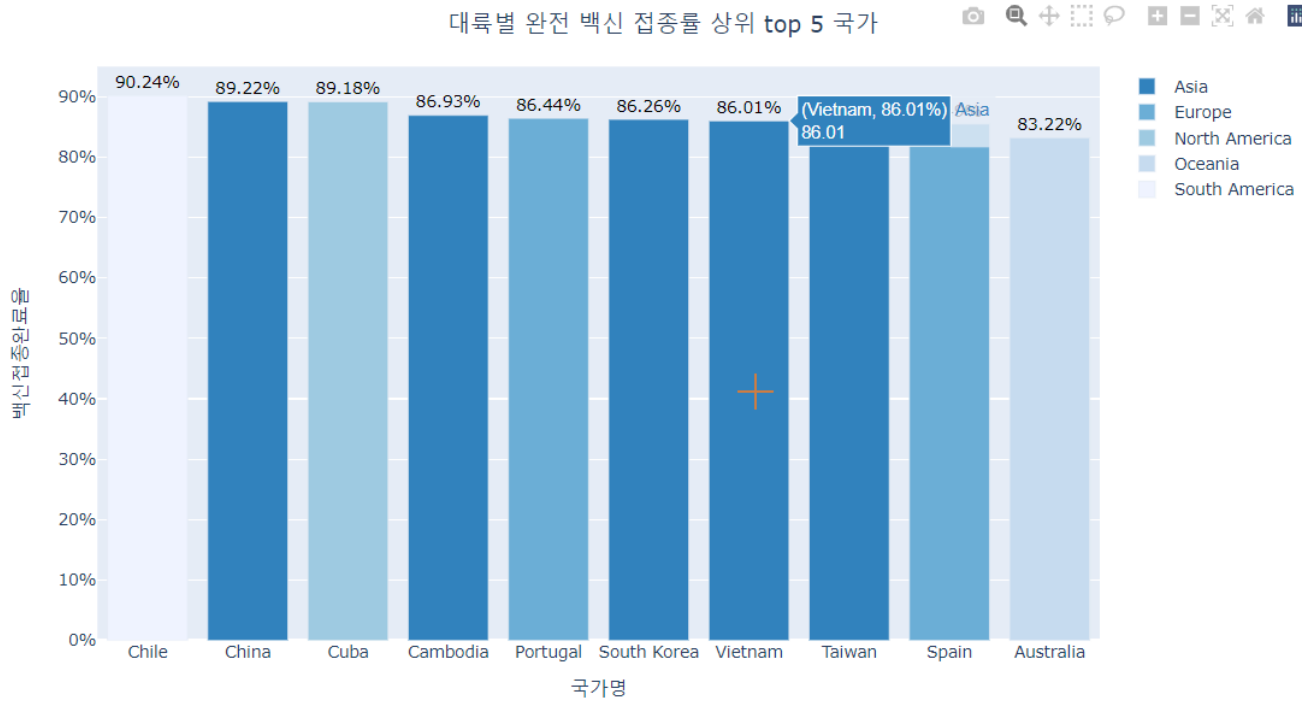
```
vaccine_top10 <- df_covid19_stat |>
 filter(인구수 > 10000000) |>
 top_n(10, 인구백명당백신접종완료률)

vaccine_top10 |>
 plot_ly() |>
 add_trace(type = 'bar',
        x = ~location,
        y = ~인구백명당백신접종완료률,
        color = ~continent, text = ~인구백명당백신접종완료률,
        textposition = 'outside', texttemplate = '%{text}%',
        textfont = list(color = 'black')) |>
 layout(title = '완전 백신 접종률 상위 top 10 국가',
      xaxis = list(title = '국가명', categoryorder = 'total descending'),
      yaxis = list(title = '백신접종완료율',
             ticksuffix = '%'),
      margin = margins)
```



실행결과 6-1 데이터가 정렬된 막대그래프

- python

```
vaccine_top10 = df_covid19_stat.loc[df_covid19_stat['인구수'] >
10000000].sort_values(by=['인구백명당백신접종완료률'], ascending=False).head(10).reset_index()

fig = go.Figure()

for 대륙, group in vaccine_top10.groupby('continent'):
    fig.add_trace(go.Bar(
        x = group['location'], y = group['인구백명당백신접종완료률'],
        name = 대륙,
        text = group['인구백명당백신접종완료률'], textposition = 'outside', texttemplate =
'%{text}%',
        textfont = dict(color = 'black')
    ))
fig.update_layout(title = dict(text = '완전 백신 접종률 상위 top 10 국가', x = 0.5),
         xaxis = dict(title = '국가명', categoryorder = 'total descending'),
         yaxis = dict(title = '백신접종완료율',
                      ticksuffix = '%'),
         margin = margins)

fig.show()
```



1.1.2. 수평 막대 그래프

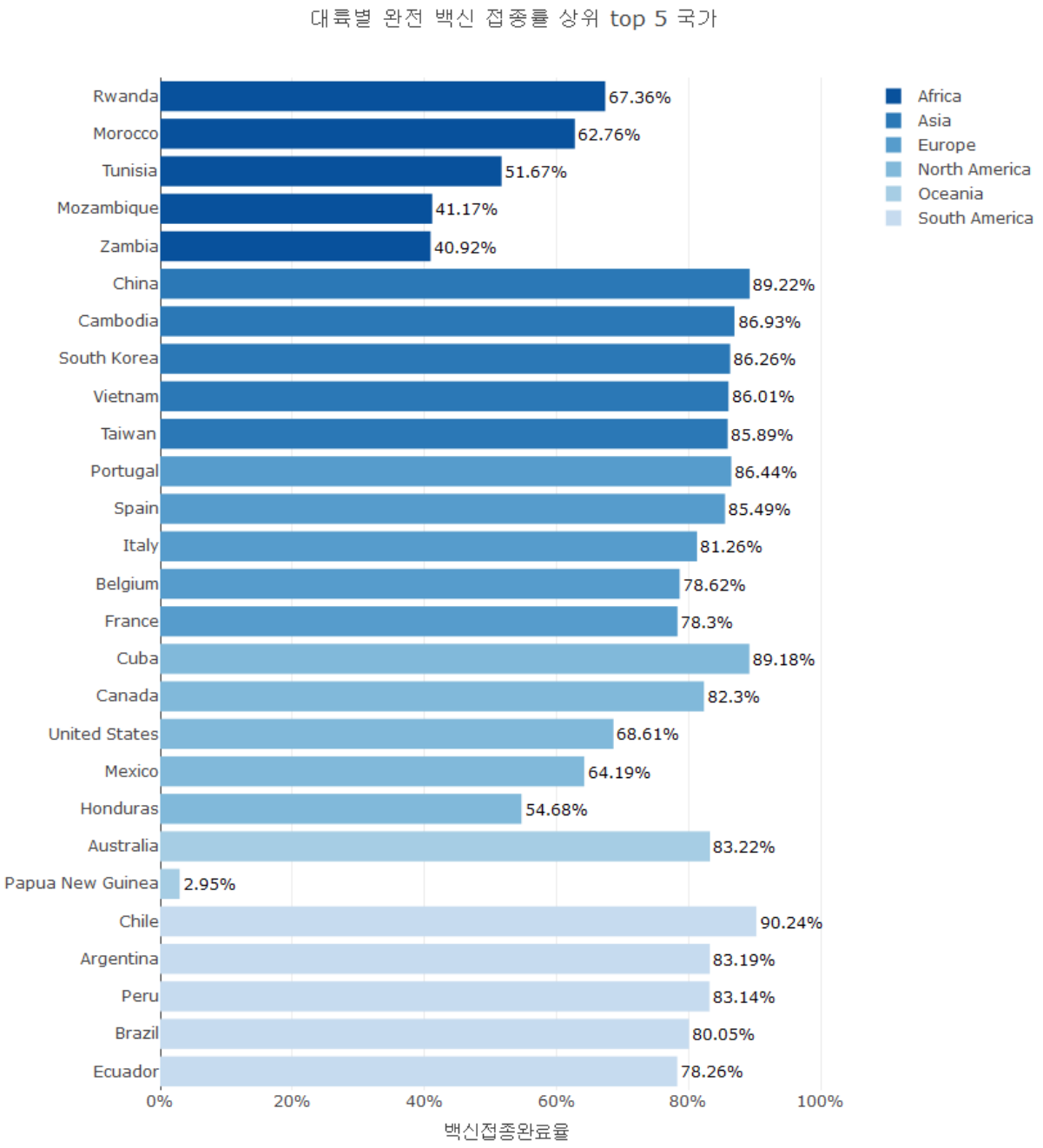
앞서 살펴본 전세계 국가 중 백신 접종률 top 10 은 백신 접종의 전체 현황은 살펴볼 수 있지만 각 대륙별 현황은 알아보기 어렵다. 이렇게 비교가 필요한 그룹간의 데이터를 비교하기 위해서는 각 그룹별로 상위 혹은 하위 데이터를 먼저 전처리 한 후 이 데이터를 사용해 시각화하는 방법을 사용해야 한다. 이번에는 각 대륙별 백신 접종률 top 5 를 비교해보자.

이를 위해서는 먼저 데이터를 전처리해야 한다. 각 대륙별로 그룹화하여 이 그룹별로 top 5 를 산출해준다. 이 시각화는 앞선 시각화와는 몇 가지 차이가 있는데 가장 큰 차이는 수평 막대 그래프라는 점이다. 각 대륙별 top 5 를 산출하면 6 대륙의 5 개 국가이기 때문에 30 개 국가가 산출되게 된다. 하재만 앞서와 같이 인구수가 너무 작은 국가를 제외하다보니 오세아니아 대륙은 1 개 국가만이 필터링되어 총 26 개국이 나오게 된다. 이렇게 많은 막대를 표현하기에는 좌우 폭은 너무 좁다. 따라서 이 시각화는 수평 막대 그래프로 표현하는 것이 효과적이다.

또 하나의 차이는 각 대륙별로 그룹화하기 위해 하나의 추가적 열을 생성하였다. `plotly` 에서는 두 개의 열에 대한 정렬을 사용할 수 없기 때문에 시각화할 순서를 미리 정해주는 순차 번호가 기록되는 열이다. 이 번호를 사용하여 국가를 정렬함으로써 대륙별 백신 접종률의 상위 top 5 국가들에 대한 시각화가 완성된다. 이 순차 번호 열을 Y 축에 매핑했기 때문에 Y 축에 표시되는 문자열을 설정하기 위해 `ticktext` 와 `tickvals` 를 설정하였다.

- R

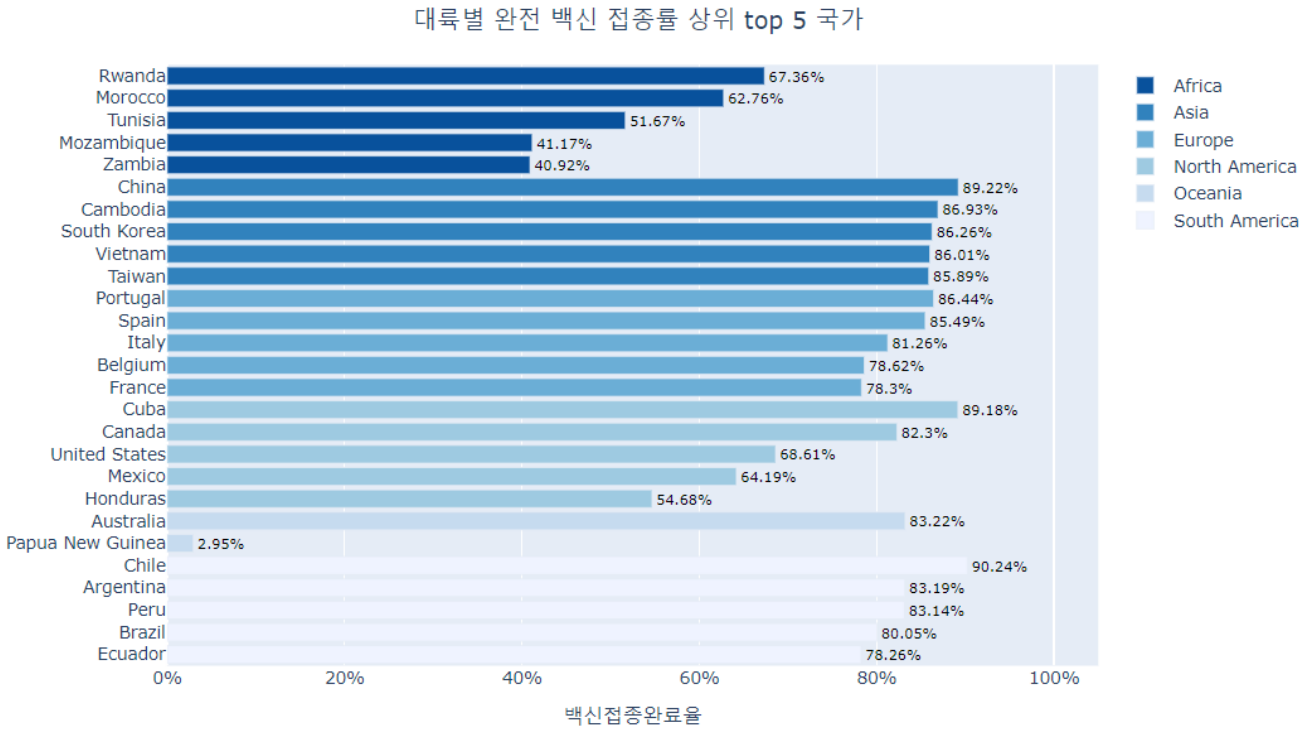
```
vaccine_top5_by_continent <- df_covid19_stat |>
 filter(인구수 > 10000000, !is.na(continent)) |>
 group_by(continent) |>
 top_n(5, 인구백명당백신접종완료률) |>
 arrange(continent, desc(인구백명당백신접종완료률)) |>
 ungroup() |>
 mutate(seq = as.factor(seq(1:n())))

vaccine_top5_by_continent |>
 plot_ly() |>
 add_trace(type = 'bar',
        y = ~seq, x = ~인구백명당백신접종완료률,
        color = ~continent,
        text = ~인구백명당백신접종완료률, textposition = 'outside', texttemplate = '%{text}%',
        textfont = list(color = 'black'),
        orientation = 'v'
        ) |>
 layout(barmode = 'group', title = '대륙별 완전 백신 접종률 상위 top 5 국가',
      xaxis = list(title = '백신접종완료율',
             ticksuffix = '%', range = c(0, 105)),
      yaxis = list(title = '', autorange = 'reversed',
             tickvals = ~seq,
             ticktext = ~location),
      margin = margins)
```



- python

```
fig = go.Figure()
colors = {'South America':"#EFF3FF", 'Oceania':'#C6DBEF', 'North America':'#9ECAE1',
'Europe':'#6BAED6', 'Asia':'#3182BD', 'Africa':'#08519C'}

for continent, group in vaccine_top10.groupby('continent'):
    fig.add_trace(go.Bar(
        y = group['location'], x = group['인구백명당백신접종완료률'],
        name = continent,
```

```
        marker_color = colors[continent],
        text = group['인구백명당백신접종완료률'], textposition = 'outside', texttemplate =
'%{text}%',
        textfont = dict(color = 'black'), orientation = 'h'
    ))

fig.update_layout(title = dict(text = '대륙별 완전 백신 접종률 상위 top 5 국가', x = 0.5),
         xaxis = dict(title = '백신접종완료율',
                      ticksuffix = '%', range = (0, 105)),
         yaxis = dict(title = '', autorange = 'reversed'),
         margin = dict(t = 50, b = 25, l = 25, r = 25))

fig.show()
```



대륙별 완전 백신 접종률 상위 top 5 국가의 결과를 보면 대부분 70% 후반의 백신 접종률을 보이지만 아프리카와 북미 지역은 상대적으로 백신 접종률이 떨어지는 것으로 나타난다. 아프리카는 대륙의 국가들의 소득이 낮기 때문에 그럴수 있으리라 추정도 가능하지만 북미 지역의 경우 쿠바와 캐나다만 접종률이 높고 나머지 국가들은 60%중반을 넘기지 못하는 것으로 나타난다. 특히 미국의 백신 접종률이 60% 중반밖에 되지 못한다. 또하나 특징적인 부분이 오세아니아인데 오세아니아는 호주만 표기되었다. 이는 데이터 전처리 과정에서 인구 천만명이상의 국가를 대상으로 하였기 때문에 오세아니아 대륙에 인구 천만명 이상 국가가 호주밖에 없기 때문에 이러한 결과가 나왔다.

1.1.3. 이중 축 막대 그래프

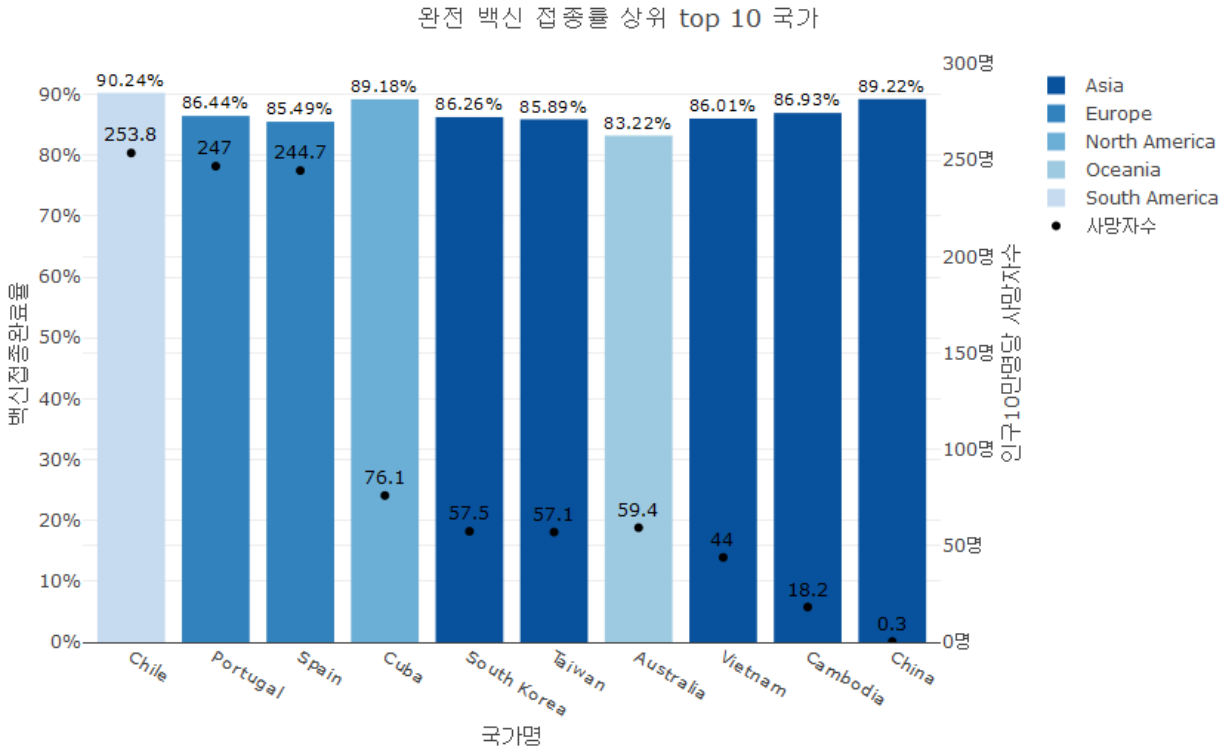
시각화를 하다보면 하나의 그래프에 여러 개의 데이터를 표현해야하는 경우가 많다. 다른 종류의 데이터를 표현하는데에는 서로 동일한 시각화 방법을 사용할 수도 있고 다른 시각화 방법을 사용할 수도 있다. 가장 흔하게 사용하는 경우가 막대 그래프와 선그래프를 혼용하는 경우이다. 이렇게 하나 이상의 데이터 계열을 표현하는 경우에는 보통 X, Y 축 중 한 축의 스케일은 공유하지만 나머지 한 축은 각각의 데이터에 대한 스케일을 가질 수 있다. 따라서 추가적인 축의 설정이 필요하다. `plotly` 에서는 기본적으로 설정되는 X 축은 xaxis, Y 축은 yaxis 로 `layout()`에서 설정이 가능하고 추가적으로 설정하는 축은 뒤에 숫자를 붙여 설정할 수 있다. X 축을 추가적으로 설정하려면 xaxis2, Y 축을 추가적으로 설정하려면 yaxis2 로 설정한다. 이렇게 설정된 추가 축을 사용하는 trace 에 `xaxis`, `yaxis` 속성을 사용하여 해당 trace 가 참조해야하는 축을 매칭한다. 이 두 속성을 설정하지 않으면 `layout()`에서 설정되는 `xaxis` 와 `yaxis` 가 매칭되지만 `xaxis` 와 `yaxis` 에 'x2', 'y2'로 설정하면 `layout()`의 `xaxis2`, `yaxis2` 에 매칭된다. 기본 축 설정인 `xaxis` 와 `yaxis` 의 속성 설정과 다른 하나는 `overlaying` 인데 `overlaying` 에 공유해야 할 축의 id 를 설정함으로써 해당 축에 부가적인 축이라는 것을 설정해야 한다. 서로 축이 겹쳐서 그려지지 않도록 `side` 속성을 사용하여 축이 그려지는 위치를 설정할 수 있다.

이중 축의 사례를 살펴보기 위해 앞에서 그렸던 완전 백신 접종률 상위 10 국가 막대 그래프에 인구 10 만명당 사망자수를 점으로 표시하는 스캐터 trace 를 추가해보도록 하겠다. 백신 접종률의 Y 축 스케일은 0%부터 100%까지의 스케일을 가지지만 해당 국가의 인구 10 만명당 사망자수는 0.3 부터 281 까지의 스케일을 가지기 때문에 추가적인 Y 축이 필요하게 된다. 따라서 인구 10 만명당 사망자수를 표시하는 `add_trace()`의 `yaxis` 속성에 'y2'를 설정하여 추가적인 축을 사용하는 trace 로 설정하고 `layout` 에 'y2'에 해당하는 축에 대한 `yaxis2` 를 설정한다. 그리고 `overlaying` 속성에 'y'를 설정하여 Y 축에 대한 보조축이라는 점을 설정하였고 `side` 를 'right'로 설정하여 `yaxis2` 가 오른쪽에 표시되도록 설정하였다.

- R

```
vaccine_top10 |>
  plot_ly() |>
  add_trace(type = 'bar',
            x = ~location, y = ~인구백명당백신접종완료률,
            color = ~continent, text = ~인구백명당백신접종완료률,
            textposition = 'outside', texttemplate = '%{text}%',
            textfont = list(color = 'black')) |>
  add_trace(type = 'scatter', mode = 'markers+text',
            name = '10 만명당 사망자수', yaxis = "y2",
```

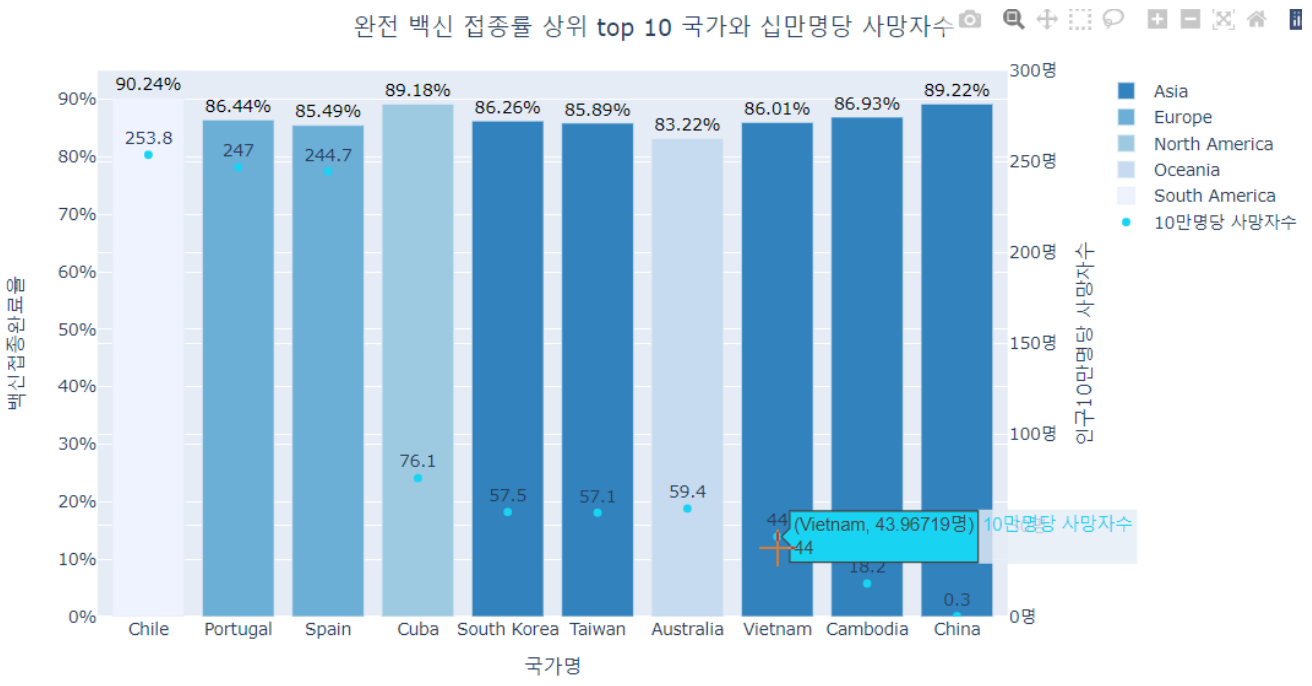
```
        x = ~location,
        y = ~십만명당사망자수, text = ~round(십만명당사망자수, 1),
        textposition = 'top'
        )|>
layout(title = '완전 백신 접종률 상위 top 10 국가',
       xaxis = list(title = '국가명', categoryorder = 'total descending'),
       yaxis = list(title = '백신접종완료율',
                    ticksuffix = '%'),
       yaxis2 = list(title = '인구 10 만명당 사망자수',
                     side = "right", overlaying = "y",
                     range = c(0, 300), ticksuffix = '명'),
       margin = list(r = 100, t = 50),
       legend = list(x = 1.1))
```



- python

```
vaccine_top10 = df_covid19_stat.loc[df_covid19_stat['인구수'] >
10000000].sort_values(by=['인구백명당백신접종완료률'], ascending=False).head(10).reset_index()

fig = go.Figure()

for 대륙, group in vaccine_top10.groupby('continent'):
    fig.add_trace(go.Bar(
        x = group['location'], y = group['인구백명당백신접종완료률'],
        name = 대륙,
        text = group['인구백명당백신접종완료률'], textposition = 'outside', texttemplate =
```

```
'%{text}%',
        textfont = dict(color = 'black')
    ))

fig.add_trace(go.Scatter(mode = 'markers+text',
            name = '10 만명당 사망자수', yaxis = "y2",
            x = vaccine_top10['location'],
            y = vaccine_top10['십만명당사망자수'], text =
round(vaccine_top10['십만명당사망자수'], 1),
            textposition = 'top center')
             )
fig.update_layout(title = dict(text = '완전 백신 접종률 상위 top 10 국가와 십만명당 사망자수', x
= 0.5),
         xaxis = dict(title = '국가명', categoryorder = 'total descending'),
         yaxis = dict(title = '백신접종완료율',
                      ticksuffix = '%'),
         yaxis2 = dict(title = '인구 10 만명당 사망자수',
                       side = "right", overlaying = "y",
                       range = (0, 300), ticksuffix = '명'),
         margin = dict(r = 100, t = 50),
         legend = dict(x = 1.1))

fig.show()
```

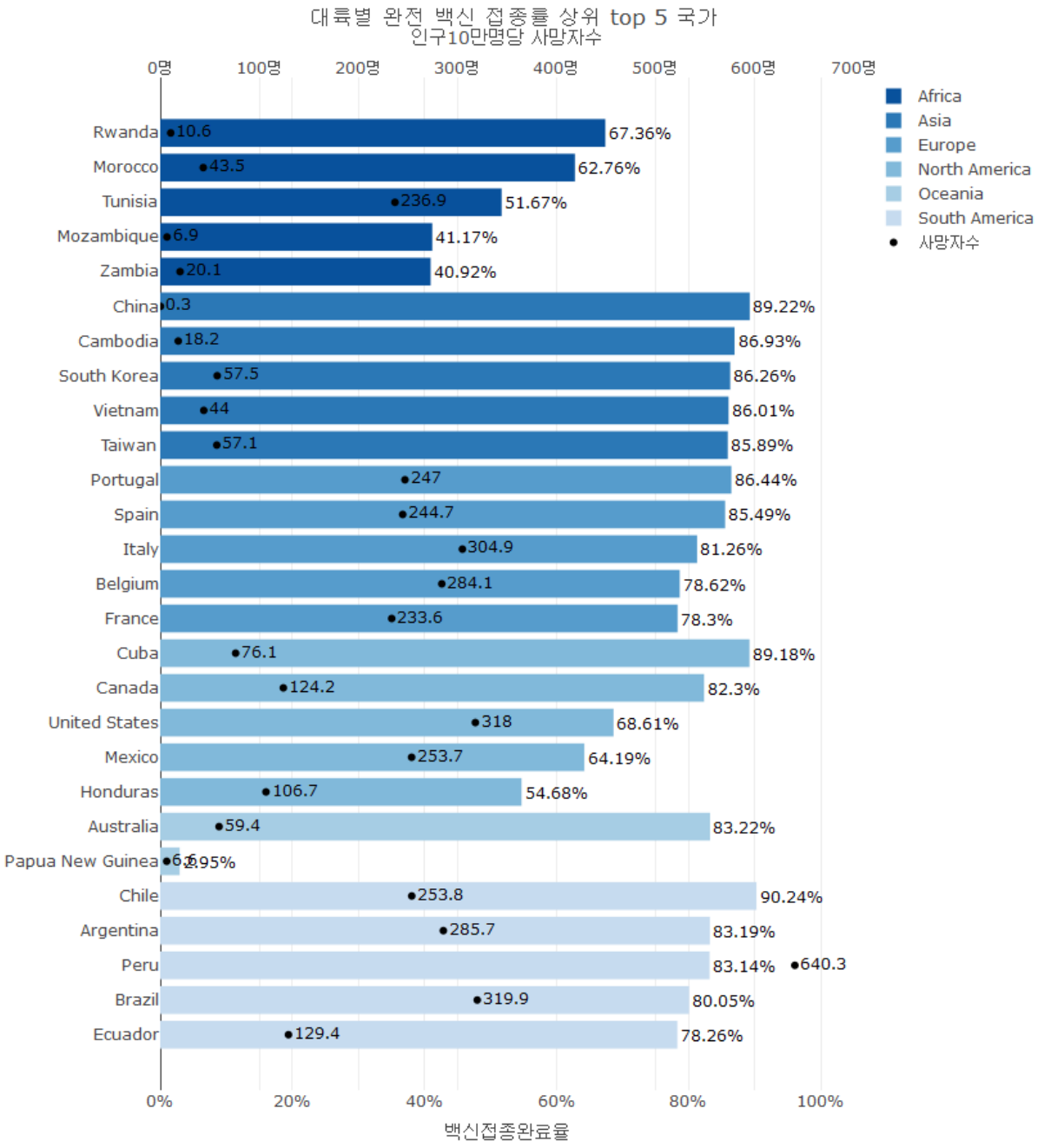


백신 접종 완료율과 인구 10 만명당 사망자수를 동시에 표기해보니 포르투갈, 칠레, 스페인, 아르헨티나는 높은 백신 접종률에도 불구하고 10 만명당 200 명 이상의 사망자가 나왔다. 하지만 쿠바, 우리나라, 캄보디아, 호주등의 국가는 높은 접종률을 보이면서 10 만명당 사망자수도 낮게 나타난다.

앞서 그렸던 대륙별 완전 백신 접종률 상위 top 5 국가에 인구 10 만명당 사망률을 추가적으로 표시하고 축을 추가하는 방법은 다음과 같다.

- R

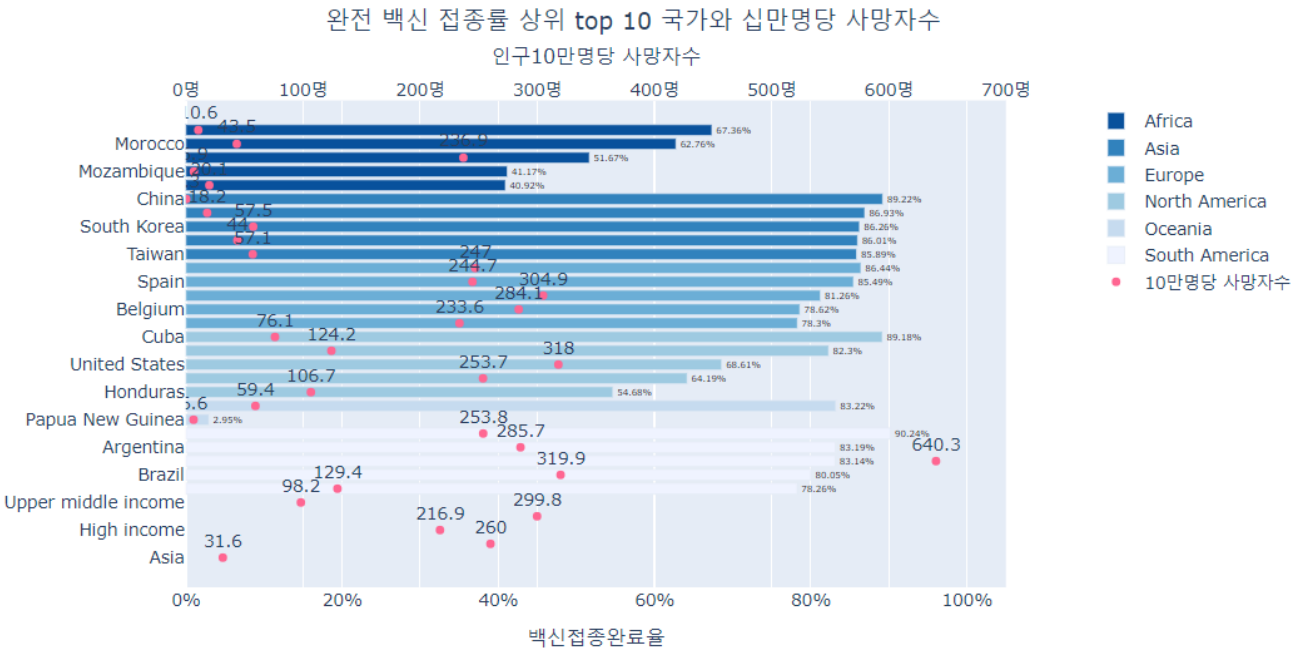
```
vaccine_top5_by_continent |>
 plot_ly() |>
 add_trace(type = 'bar',
        y = ~seq, x = ~인구백명당백신접종완료률,
        color = ~continent,
        text = ~인구백명당백신접종완료률, textposition = 'outside', texttemplate = '%{text}%',
        textfont = list(color = 'black'),
        orientation = 'v'
        ) |>
 add_trace(type = 'scatter', mode = 'markers+text',
        name = '사망자수', xaxis = "x2",
        y = ~seq, x = ~십만명당사망자수, color = I('black'),
        text = ~round(십만명당사망자수, 1),
        textposition = 'right'
        )|>
 layout(barmode = 'group',
      title = list(text = '대륙별 완전 백신 접종률 상위 top 5 국가',
                y = 0.97, yref = 'container'),
      xaxis = list(title = '백신접종완료율', range = c(0, 105),
                ticksuffix = '%'),
      yaxis = list(title = '', autorange = 'reversed',
                tickvals = ~seq,
                ticktext = ~location),
      xaxis2 = list(title = list(text = '인구 10 만명당 사망자수',
                           standoff = 1),
                side = "top",
                overlaying = "x",
                range = c(0, 700),
                ticksuffix = '명'),
      margin = list(r = 100, t = 80),
      size = list(height = 900)
      )
```



- python

```
vaccine_top10 = df_covid19_stat.loc[df_covid19_stat['인구수'] >
10000000].sort_values(by=['인구백명당백신접종완료률'], ascending=False).head(10).reset_index()
colors = {'South America':"#EFF3FF", 'Oceania':'#C6DBEF', 'North America':'#9ECAE1',
'Europe':'#6BAED6', 'Asia':'#3182BD', 'Africa':'#08519C'}

fig = go.Figure()

for 대륙, group in vaccine_top10.groupby('continent'):
```

```
    fig.add_trace(go.Bar(
        x = group['location'], y = group['인구백명당백신접종완료률'],
        name = 대륙,
        text = group['인구백명당백신접종완료률'], textposition = 'outside', texttemplate =
'%{text}%',
        textfont = dict(color = 'black')
    ))

fig.add_trace(go.Scatter(mode = 'markers+text',
            name = '10 만명당 사망자수', yaxis = "y2",
            x = vaccine_top10['location'],
            y = vaccine_top10['십만명당사망자수'], text =
round(vaccine_top10['십만명당사망자수'], 1),
            textposition = 'top center')
             )
fig.update_layout(title = dict(text = '완전 백신 접종률 상위 top 10 국가와 십만명당 사망자수', x
= 0.5),
         xaxis = dict(title = '국가명', categoryorder = 'total descending'),
         yaxis = dict(title = '백신접종완료율',
                      ticksuffix = '%'),
         yaxis2 = dict(title = '인구 10 만명당 사망자수',
                       side = "right", overlaying = "y",
                       range = (0, 300), ticksuffix = '명'),
         margin = dict(r = 100, t = 50),
         legend = dict(x = 1.1))

fig.show()
```



대륙별 완전 백신 접종률 상위 국가들의 인구 10 만명당 사망자수를 살펴보면 아시아 국가의 접종률이 높은 국가들은 전반적으로 낮은 사망자수를 나타냈고 유럽과 남미는 백신 접종률이 높았지만 사망자 수도 높게 나타나고 있음을 알 수 있다.

1.2. 롤리팝 그래프

보통 우리가 보는 시각화는 인쇄물 형태이던 웹 브라우저 등의 모니터 화면으로 보던 좌우의 확장보다는 상하의 확장에 더 유연하다. 따라서 좌우로 표현되어야 하는 막대가 많이지면 앞서와 같이 상하 확정을 사용하는 수평형 막대 그래프를 사용한다. 하지만 한눈에 보이는 추세와 비교를 같이 표시하기 위해서는 막대의 너비가 다소 부담스러울 수 있다. 이런 경우 효과적으로 사용되는 시각화가 바로 롤리팝 그래프이다.

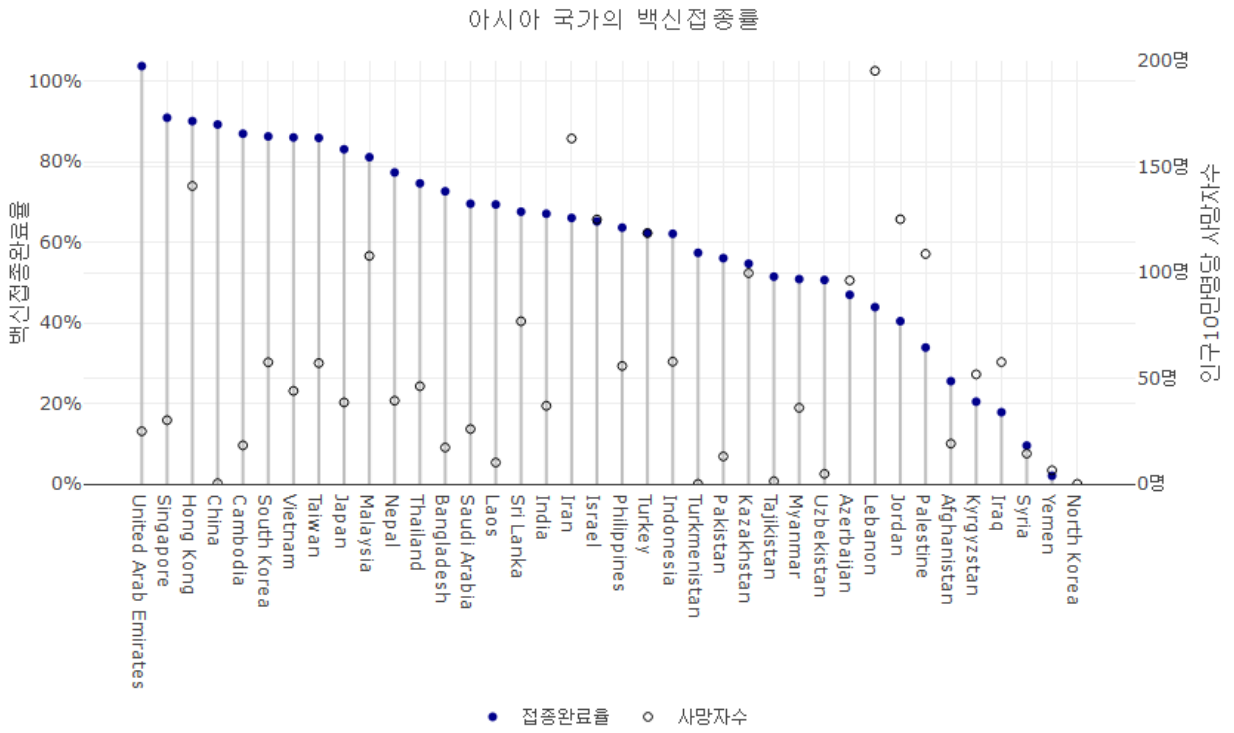
롤리팝 그래프는 막대사탕의 상품명에서 유래된 그래프이다. 막대그래프와 유사하지만 그 표현을 막대가 아닌 막대사탕처럼 표현한다는데에서 유래했다. 원으로 표시된 데이터 점으로부터 축까지를 선으로 이어 데이터를 표현하는 방식의 그래프이다.

`plotly` 에서는 이 롤리팝 그래프를 trace 타입으로 제공하지는 않는다. 따라서 롤리팝 그래프를 그리기 위해서는 `scatter` trace 의 `markers` 와 `add_segments()`를 사용하여 그려야 한다. 이 과정에서 하나 주의해야 하는 것은 `markers` trace 를 그리기 전에 `add_segements()`를 사용하여 선을 먼저 그려야 한다는 것이다. 만약 `markers` 를 먼저 그리게 되면 점 안으로 선이

보이기 때문에 시각화가 보기 좋지 않다. 다음은 인구수 오백만명 이상의 아시아 지역 국가들의 백신 접종률을 롤리팝 그래프로 그린 코드이다.

- R

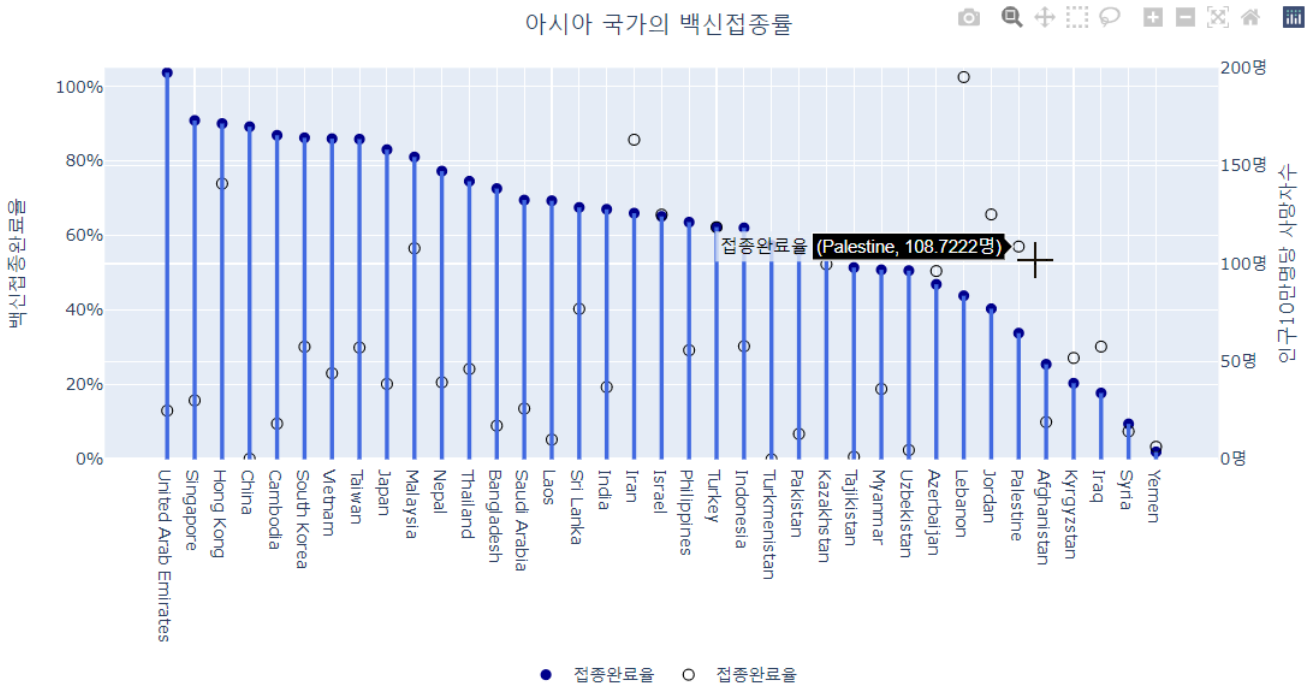
```
df_lolipop <- df_covid19_stat |>
 filter(인구수 > 5000000, continent == 'Asia') |>
 arrange(desc(인구백명당백신접종완료률))

df_lolipop |>
 plot_ly(x = ~reorder(location, desc(인구백명당백신접종완료률))) |>
 add_segments(xend = ~reorder(location, desc(인구백명당백신접종완료률)),
          y = ~인구백명당백신접종완료률,
          yend = 0, color = I('gray'),
          showlegend = FALSE
          ) |>
 # add_segments(xend = ~reorder(location, desc(인구백명당백신접종완료률)),
 #          y = ~십만명당사망자수,
 #          yend = 0, color = I('gray'),  yaxis = "y2",
 #          showlegend = FALSE
 #          ) |>
 add_trace(type = 'scatter', mode = 'markers', name = '접종완료율',
        y = ~인구백명당백신접종완료률, color = I('darkblue')) |>
 add_trace(type = 'scatter', mode = 'markers',
        symbol = I('circle-open'),
        name = '사망자수', yaxis = "y2",
        y = ~십만명당사망자수, color = I('black'),
        text = ~round(십만명당사망자수, 1),
        textposition = 'right'
        )|>
 layout(barmode = 'group',
      title = list(text = '아시아 국가의 백신접종률',
              y = 0.97, yref = 'container'),
      yaxis = list(title = '백신접종완료율', range = c(0, 105),
              ticksuffix = '%'),
      xaxis = list(title = ''),
      yaxis2 = list(title = list(text = '인구 10 만명당 사망자수',
                      standoff = 10),
               side = "right", overlaying = "y",
               range = c(0, 200), ticksuffix = '명'),
      margin = list(t = 50, b = 25, l = 25, r = 70),
      legend = list(orientation = 'h', y = -0.5, x = 0.5,
              yref = 'container', xanchor = 'center'),
       showlegend = T)
```



- python

```
df_lolipop = df_covid19_stat.reset_index()

df_lolipop = df_lolipop.loc[(df_lolipop['인구수'] > 5000000) & (df_lolipop['continent'] ==
'Asia') & (df_lolipop['인구백명당백신접종완료률'].isna() == False)].sort_values(by =
'인구백명당백신접종완료률', ascending=False)

fig = go.Figure()

for index, row in df_lolipop.iterrows():
    fig.add_shape(type="line", xref="x", yref="y",
        x0=row['location'], y0=0, x1=row['location'], y1=row['인구백명당백신접종완료률'],
        line=dict(color="RoyalBlue",width=3)
    )

fig.add_trace(go.Scatter(
    mode = 'markers', name = '접종완료율',
    x = df_lolipop['location'], y = df_lolipop['인구백명당백신접종완료률'],
    marker = dict(color = 'darkblue', symbol = 0, size = 8)
))

fig.add_trace(go.Scatter(
    mode = 'markers', name = '접종완료율',
    x = df_lolipop['location'], y = df_lolipop['십만명당사망자수'], yaxis = "y2",
```

```
    marker = dict(color = 'black', symbol = 100, size = 8)
))

fig.update_layout(
        title = dict(text = '아시아 국가의 백신접종률', x = 0.5),
        xaxis = dict(title = ''),
        yaxis = dict(title = '백신접종완료율', range = (0, 105),
                     ticksuffix = '%'),
        yaxis2 = dict(title = dict(text = '인구 10 만명당 사망자수',
                                   standoff = 10),
                      side = "right", overlaying = "y",
                      range = (0, 200), ticksuffix = '명'),
        margin = dict(t = 50, b = 25, l = 25, r = 70),
        legend = dict(orientation = 'h', y = -0.5, x = 0.5, xanchor = 'center'),
        showlegend = True)
```



아시아 지역 인구 5 백만명 이상 국가들의 접종 완료율과 사망자수를 롤리팝 그래프로 보면 전반적으로 접종률이 높은 아랍에미레이트, 싱가포르, 우리나라, 캄보디아, 일본 등의 국가는 사망다수도 비교적 낮게 나타난다. 반면 접종률이 낮게 나타나는 이라크, 키르기스탄, 레바논, 팔레스타인, 요르단, 아제르바이잔 등의 국가는 사망자수이 높게 나타나고 있다.

2. 차이(difference)의 시각화

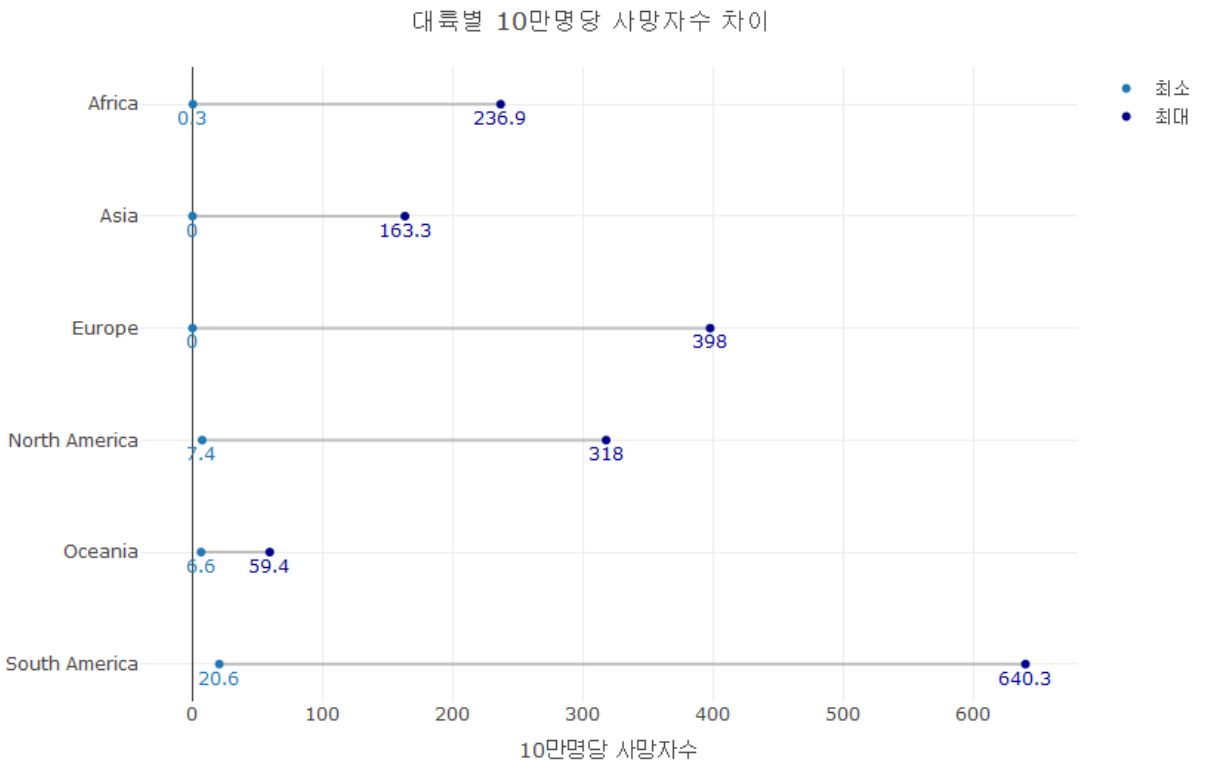
앞에서 설명한 비교의 시각화는 특정 변수의 변량값들을 서로 비교함으로써 상대적인 위치를 찾아내기 위한 시각화 방법이다. 하지만 동일 시각화 대상들의 시점이나 조건의 변화에 따른 변량 값의 차이를 확인하기 위해 사용하는 시각화 방법이 차이의 시각화이다. 시계열 그래프가 시간 변화의 조건에 따른 변량의 차이를 표현하는 좋은 예이기는 하지만 보통 차이의 시각화는 두 시점이나 두 조건의 차이를 보여주는 것이 효과적이다.

2.1. 덤델 차트

덤벨(Dumbbell) 그래프는 동일한 변수의 두개의 값을 비교하기 위해 사용하는 차트이다. 일반적으로 양쪽 끝을 둥글게 만들고 그 사이를 선으로 연결하여 생긴 형태가 운동할때 쓰는 아령과 같이 생겨서 붙여진 이름이다.앞서 롤리팝 그래프와 같이 `plotly` 는 덤벨 그래프를 위한 함수를 제공하지 않기 때문에 `scatter` trace 의 `markers` 와 `add_segments()`를 사용하여 그려야 한다. 다음은 대륙별 인구수가 백만 이상의 국가 중에 십만명당 사망자수가 가장 작은 나라와 가장 큰 나라의 차이를 시각화하는 코드이다.

- R

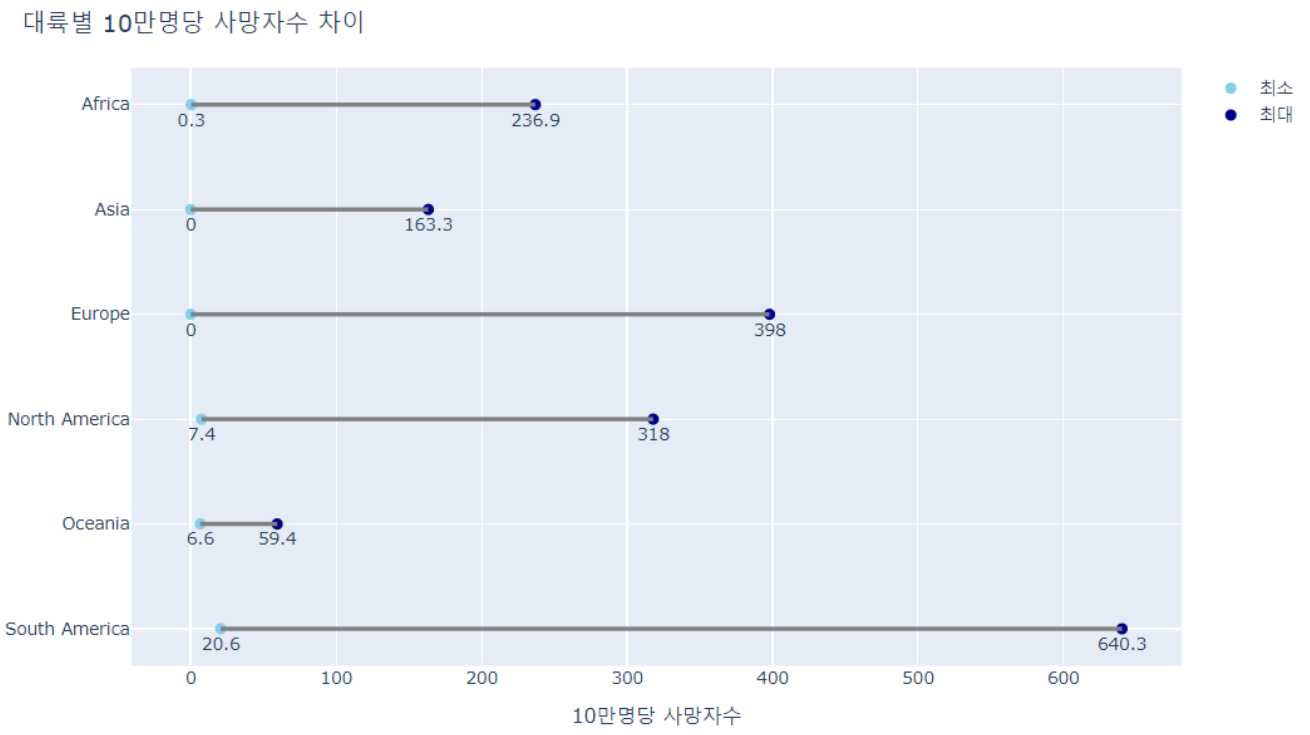
```
df_covid19_stat |>
 filter(!is.na(continent), 인구수 > 10000000) |>
 group_by(continent) |>
 summarise(min = min(십만명당사망자수), max = max(십만명당사망자수)) |>
 plot_ly() |>
 add_segments(
        x = ~min, xend = ~max, y = ~continent, yend = ~continent,
        showlegend = FALSE,
        color = I('gray')) |>
 add_trace(type = 'scatter', mode = 'markers+text',
        x = ~min, y = ~continent, name = '최소',
        text = ~round(min, 1), textposition = 'bottom center',
        color = I('#1f77b4')) |>
 add_trace(type = 'scatter', mode = 'markers+text',
        x = ~max, y = ~continent, name = '최대',
        text = ~round(max, 1), textposition = 'bottom center',
        color = I('darkblue')) |>
 layout(title = '대륙별 10 만명당 사망자수 차이',
      xaxis = list(title = '10 만명당 사망자수'),
      yaxis = list(title = '', autorange = 'reversed'),
      margin = margins
      )
```



- python

```
df_dumbbell = df_covid19_stat.copy().reset_index()
df_dumbbell = df_dumbbell.loc[(df_dumbbell['continent'].isna() == False) & (df_dumbbell['인구수']
> 10000000)].groupby('continent')['십만명당사망자수'].agg([('min', 'min'), ('max',
'max')]).reset_index()

fig = go.Figure()
fig.add_trace(go.Scatter(
    mode = 'markers+text', name = '최소',
    x = df_dumbbell['min'], y = df_dumbbell['continent'],
    text = round(df_dumbbell['min'], 1), textposition = 'bottom center',
    marker = dict(size = 8, color = 'skyblue')
))
fig.add_trace(go.Scatter(
    mode = 'markers+text', name = '최대',
    x = df_dumbbell['max'], y = df_dumbbell['continent'],
    text = round(df_dumbbell['max'], 1), textposition = 'bottom center',
    marker = dict(size = 8, color = 'darkblue')
))
for index, row in df_dumbbell.iterrows():
    fig.add_shape(type="line", xref="x", yref="y",
        x0=row['min'], y0=row['continent'], x1=row['max'], y1=row['continent'],
        line=dict(color="gray",width=3)
    )
```

```
fig.update_layout(title = '대륙별 10 만명당 사망자수 차이',
         xaxis = dict(title = '10 만명당 사망자수'),
         yaxis = dict(title = '', autorange = 'reversed'),
         margin = margins
         )
fig.show()
```



대륙별 10 만명당 사망자수 차이 시각화를 보면 남미의 경우 최소 사망자수 국가와 최대 사망자수 국가의 차이가 약 620 명으로 가장 크게 나타난다. 반면 아시아 국가의 경우 최대와 최소의 차이가 약 165 명정도로 가장 작게 나타나고 있음을 알 수 있다.

2.2. 퍼널 차트

피라미드 그래프는 보통 인구 연령별 인구수를 표현하는 그래프에서 많이 사용된다. 하지만 `plotly` 에서는 피라미드 그래프라는 이름으로 사용되지 않고 퍼널(깔때기, funnel) trace 라는 이름으로 사용된다. 퍼널 trace 는 변량의 크기를 길이로 표현한 막대로 표현하고 각 막대의 중간을 맞춰 양쪽으로 퍼져나가는 깔때기 형태의 trace 이다. 이 퍼널 차트는 영업 및 마케팅 부서에서 자주 사용되는데 여러 단계별로 값이나 지표가 변화하는 것을 한 눈에 표현하기 위한 목적으로 사용한다.

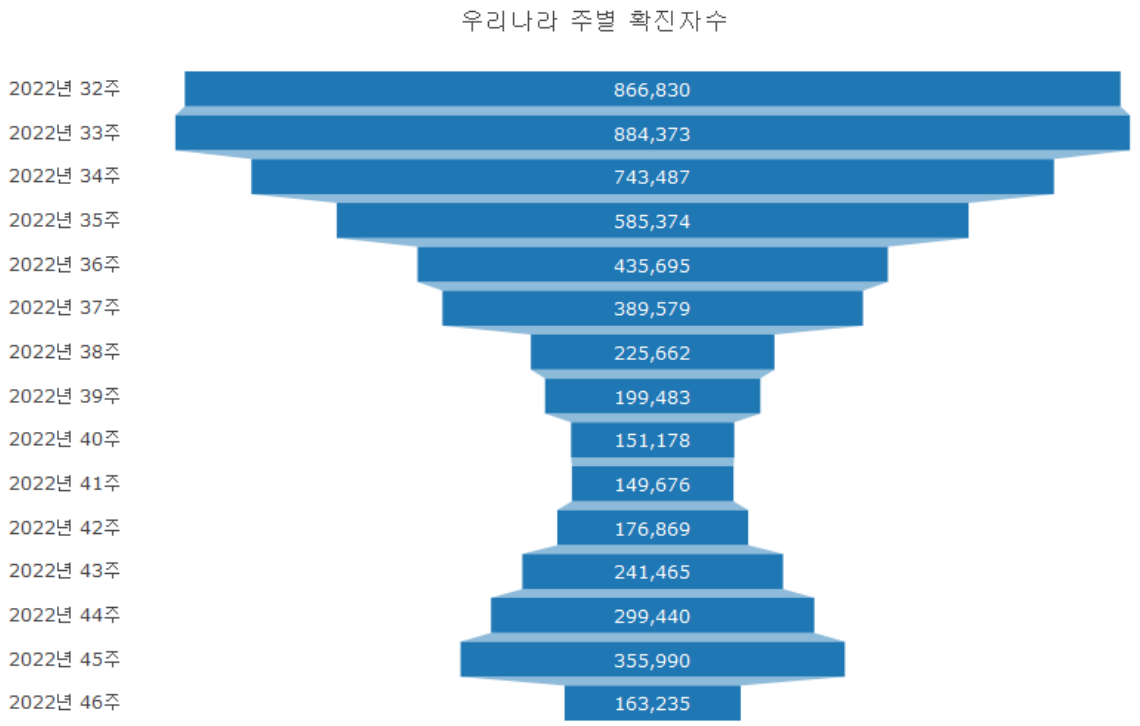
코로나 19 데이터를 사용하여 퍼널 차트를 만들기 위해 일간 코로나 19 데이터를 주(week) 단위로 요약하여 주 단위로 값의 변화를 살펴보겠다. 먼저 최근 100 일간의 우리나라 코로나 19 신규 확진자 데이터를 주 단위로 요약하는 전처리는 다음과 같다.

- R

```
df_funnel <-
 df_covid19_100 |>
 filter(iso_code == 'KOR') |>
 ## date 의 월 단위 열을 yearmonth 에 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, yearmonth 로 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. X 축은 데이터 값인 신규 확진자 수를 매핑하고 Y 축은 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)
```



실행결과 6-2 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)
```

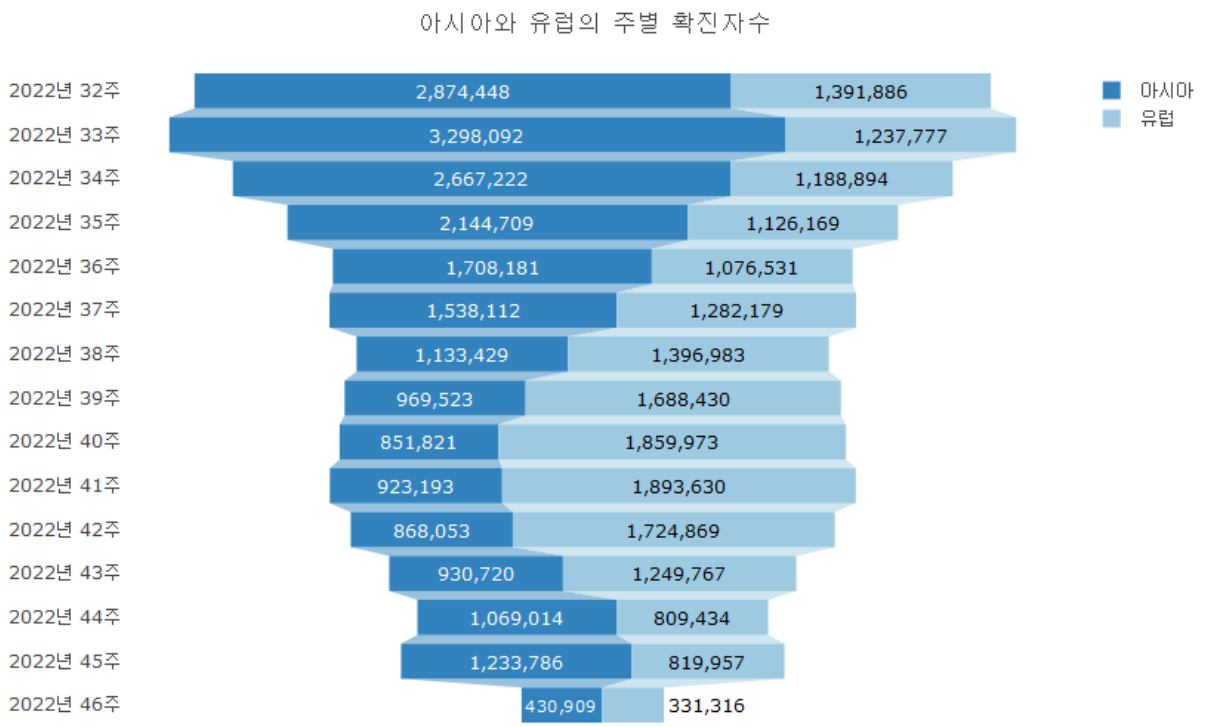
앞선 퍼널 차트에서 보면 우리나라의 신규 확진자 수는 2022 년 4 주차부터 늘어나기 시작해서 2022 년 11 주차(3.13~3.19)까지 급격히 증가하고 감소하는 상황이라는 것이 한눈에 보인다.

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') |>
 ## date 의 월 단위 열을 yearmonth 에 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, yearmonth 로 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') |>
 ## date 의 월 단위 열을 yearmonth 에 저장
```

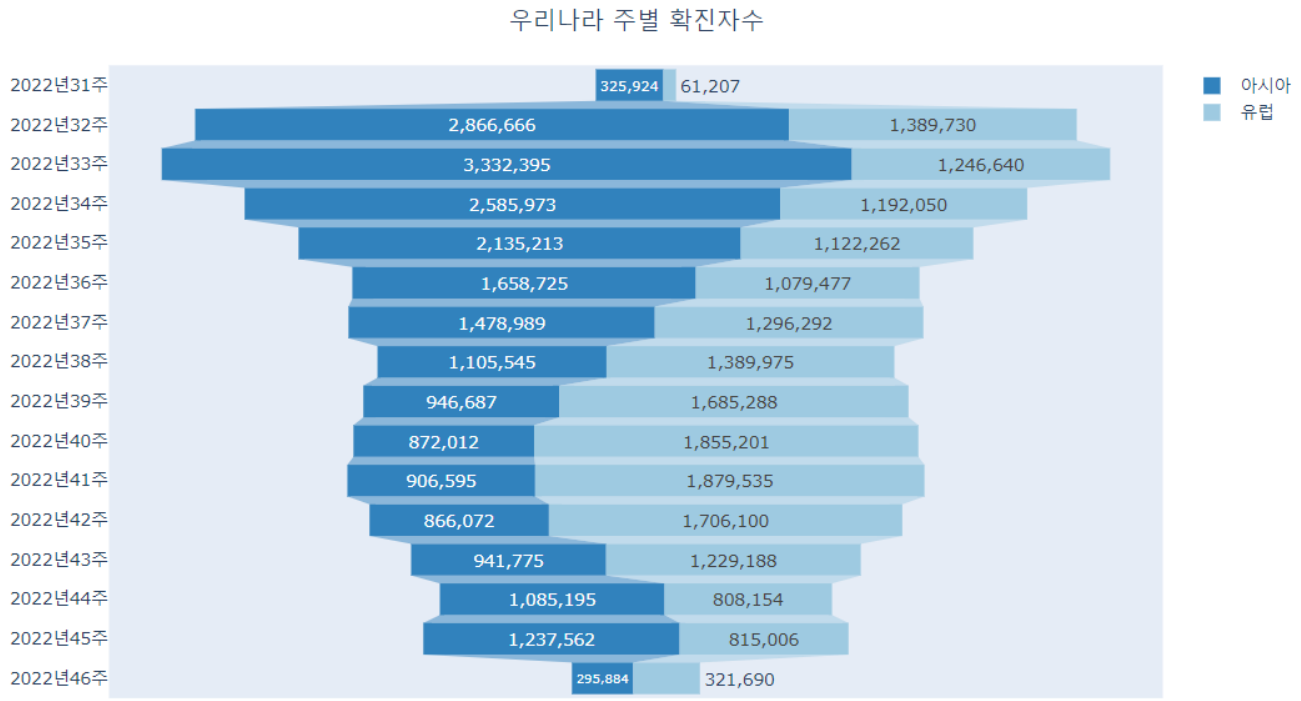
```
  mutate(date_by_week = lubridate::floor_date(date, "week"),
         yearweekth =  paste0(lubridate::year(date_by_week), '년 ',
                              lubridate::week(date_by_week), '주')) |>
  ## iso_code, yearmonth 로 그룹화
  group_by(iso_code, date_by_week, yearweekth) |>
  ## new_cases 합계 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)
```



- 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)
```



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.

2.3. 비율 막대 그래프

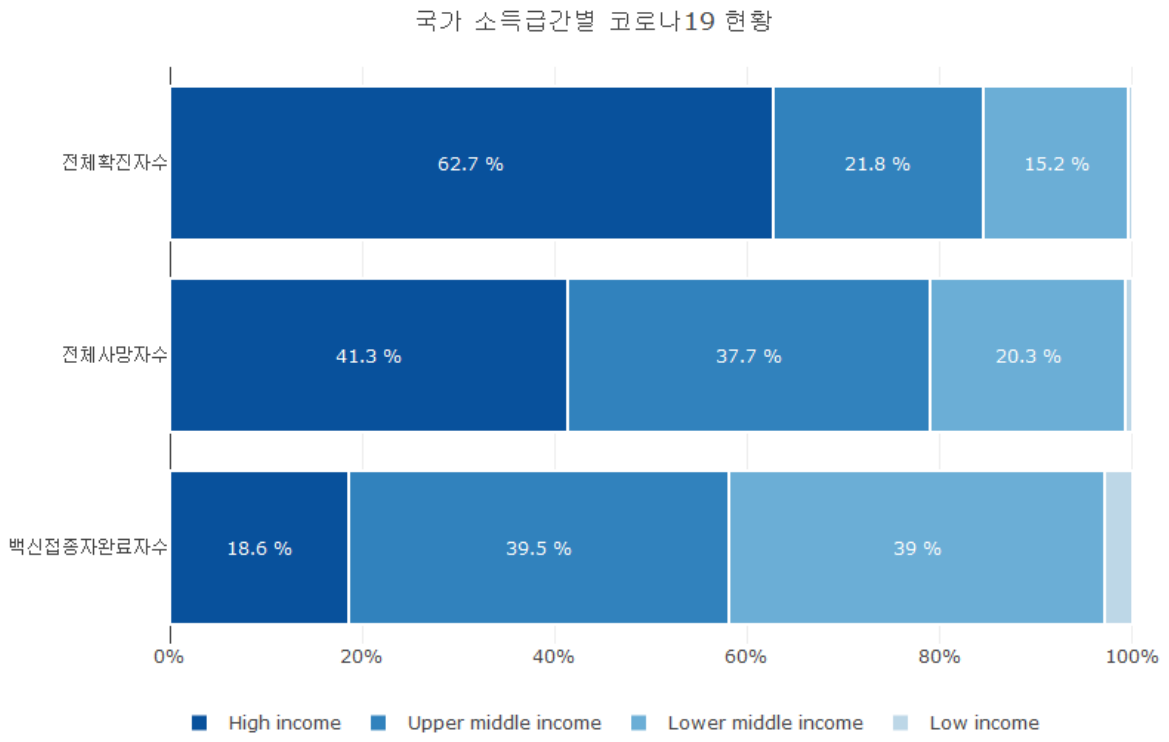
앞 장에서 `plotly` 가 지원하는 막대 trace 의 `barmode` 는 stack, group, overlay, ralative 의 네 가지가 있다고 설명하였다. 막대 그래프의 stack 을 사용하면 구성의 시각화에 효과적으로 사용된다. 이를 비율 막대 그래프라고 하는데 막대 길이가 모두 1 로 같게 그리고 막대를 구성하는 변량은 전체 대비 비율로 표현하는 막대 그래프이다. 따라서 전체 데이터 값의 비교가 불가능하고 단지 그 세부 분류의 비율만을 확인할 수 있다.

`ggplot2` 에서는 비율 막대 그래프를 매개변수 설정만으로 간단히 만들수 있다. 그러나 `plotly` 에서는 비율 누적 막대 그래프를 지원하지 않기 때문에 비율 막대 그래프로 표현할 각 데이터들을 비율로 바꾸어 주는 전처리 과정이 필요하다. 전처리된 비율을 사용하여 `barmode = 'stack'` 으로 막대 trace 를 사용하면 비율 막대 그래프가 만들어진다. 그리고 만들어진 비율 막대 그래프에 비율을 표시하기 위해서는 비율을 주석으로 적절한 위치에 표시하여야 한다.

다음은 전세계 국가들의 소득별로 그룹화한 통계를 사용하여 '전체 확진자', '전체 사망자', '백신접종완료자수'를 비율 막대로 표시하는 코드이다.

- R

```
df_covid19_stat |>
 filter(iso_code %in% c('OWID_HIC', 'OWID_LIC', 'OWID_LMC', 'OWID_UMC')) |>
 select(3, 6, 7, 14) |>
 pivot_longer(cols = c(2, 3, 4)) |>
 pivot_wider(names_from = location) |>
 group_by(name) |>
 mutate(sum = (`High income`+`Low income`+`Lower middle income`+`Upper middle income`)) |>
 mutate(`High income` = `High income` / sum,
      `Low income` = `Low income` / sum,
      `Lower middle income` = `Lower middle income` / sum,
      `Upper middle income` = `Upper middle income` / sum) |>
 plot_ly(type = 'bar', x = ~`High income`, y = ~name,
       orientation = 'h', name = 'High income',
       marker = list(line = list(color = 'white',
                      width = 2))
       ) |>
 add_trace(x = ~`Upper middle income`, name = 'Upper middle income') |>
 add_trace(x = ~`Lower middle income`, name = 'Lower middle income') |>
 add_trace(x = ~`Low income`, name = 'Low income') |>
 add_annotations(xref = 'x', yref = 'y',
            x = ~`High income` / 2, y = ~name,
            text = ~paste(round(`High income`*100, 1), '%'),
            font = list(color = 'white'),
            showarrow = FALSE) |>
 add_annotations(xref = 'x', yref = 'y',
            x = ~`High income` + `Upper middle income` / 2, y = ~name,
            text = ~paste(round(`Upper middle income`*100, 1), '%'),
            font = list(color = 'white'),
            showarrow = FALSE) |>
 add_annotations(xref = 'x', yref = 'y',
            x = ~`High income` + `Upper middle income` + `Lower middle income` / 2, y = ~name,
            text = ~paste(round(`Lower middle income`*100, 1), '%'),
            font = list(color = 'white'),
            showarrow = FALSE) |>
 layout(barmode = 'stack',
      title = '국가 소득급간별 코로나 19 현황',
      xaxis = list(title = '', tickformat = '.0%'),
      yaxis = list(title = ''),
      legend = list(orientation = 'h', traceorder = 'normal'),
      margin = margins)
```



3. 구성의 시각화

3.1. 선버스트 차트

'sunburst'의 사전적 의미는 '햇살'이다. 햇살처럼 퍼저나가는 데이터를 표현하기 위해 사용되는 형태의 시각화이다. `plotly` 에서 지원하는 선버스트 trace 는 루트(root)에서 잎(leaf)까지 원의 방사형 바깥쪽으로 퍼져나가면서 계층화된 데이터를 시각화한다. 따라서 선버스트 trace 를 그리기 위해서는 계층화된 데이터가 필요하다. 선버스트 trace 의 각 섹터는 `labels` 과 `parents` 의 속성에서 설정하는데 `label` 은 선버스트 trace 에서 표시되는 섹터의 표시 라벨의 벡터가 설정되고 `parents` 는 각 섹터의 부모 섹터를 가리키는 라벨을 설정한다. 만약 `parents` 에 설정된 부모 섹터가"이면 가장 안쪽의 노드인 루트 노드가 된다.

선버스트 trace 에서 추가적으로 알아두어야 할 속성이 `branchvalues` 와 `insidetextorientation` 이다. `branchvalues` 는 `values` 로 설정된 아이템의 합계 산출 방법을 설정한다. "total"로 설정하면 `values` 의 아이템이 모든 하위 항목의 값으로 간주된다. 예를 들어 부모 섹터의 `values` 가 50, 2 개의 자식 섹터의 `values` 가 각각 20 라면 부모 섹터의 40%씩을 차지하게 된다. 반면 "remainer"는 부모 섹터와 자식 섹터에 해당하는 `values` 를 모두

더해서 비율을 산출하게 된다. 앞의 예처럼 부모 섹터가 50, 2 개의 자식 섹터가 각각 20 이라면 20/(50+20+20)이므로 22%가 산출된다.

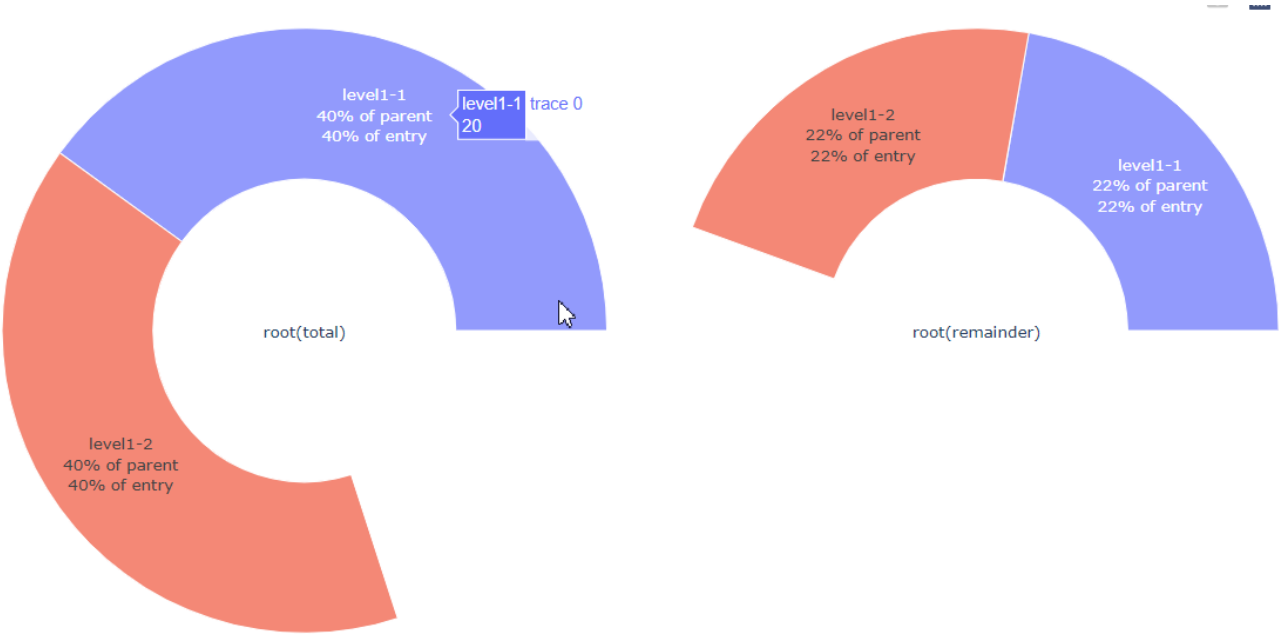
- R

```
fig1 <- plot_ly(
  labels = c("root", "level1-1", "level1-2"),
  parents = c("", "root", "root"),
  values = c(50, 20, 20),
  type = 'sunburst', textinfo = 'label+percent parent+percent entry',
  branchvalues = 'reminder',
  marker= list(colors = RColorBrewer::brewer.pal(3, 'Blues')),
  domain = list(x = c(0.51, 1), y = c(0, 1))
)

fig2 <- plot_ly(
  labels = c("root", "level1-1", "level1-2"),
  parents = c("", "root", "root"),
  values = c(50, 20, 20),
  type = 'sunburst',textinfo = 'label+percent parent+percent entry',
  branchvalues = 'total',
  domain = list(x = c(0, 0.49), y = c(0, 1))
)


subplot(fig1 |>
        add_annotations(x = 0.5, y = 1, xref = 'paper', yref = 'paper',
                        xanchor = 'center',
                        text = "branchvalues = 'total'", showarrow = F),
      fig2  |>
        add_annotations(x = 0.5, y = 1, xref = 'paper', yref = 'paper',
                        xanchor = 'center',
                        text = "branchvalues = 'reminder'", showarrow = F)
)
```

- python

```
fig = go.Figure()

fig.add_trace(go.Sunburst(
    labels = ("root(total)", "level1-1", "level1-2"),
    parents = ("", "root(total)", "root(total)"),
    values = (50, 20, 20),
    textinfo = 'label+percent parent+percent entry',
    branchvalues = 'total',
    domain=dict(column=0)
))

fig.add_trace(go.Sunburst(
    labels = ("root(remainder)", "level1-1", "level1-2"),
    parents = ("", "root(remainder)", "root(remainder)"),
    values = (50, 20, 20),
    textinfo = 'label+percent parent+percent entry',
    branchvalues = 'remainder',
    domain=dict(column=1)
))

fig.update_layout(
    grid= dict(columns=2, rows=1),
    margin = dict(t=0, l=0, r=0, b=0)
```

```
)

fig.show()
```



`insidetextorientation` 은 선버스트 trace 에 표시되는 내부 정보 문자열이 표시되는 각도를 설정한다. `insidetextorientation` 는 'radial'과 'horizontal'의 두 가지 방법이 사용되는데 'radial'은 섹터의 각도에 따라 문자가 회전하고 'horizontal'은 섹터의 각도와 관계없이 수평으로 표기된다.

다음은 선버스트 trace 를 위해 필요한 데이터의 전처리 과정이다. df_sunburst_cases 데이터프레임을 만들기 위해서 두 개의 데이터프레임을 `rbind()`로 붙여 하나의 데이터프레임으로 만들었다. 첫 번째 데이터프레임은 각 대륙별 전체 확진자수 데이터프레임이고 두 번째 데이터프레임은 각 대륙별로 전체 확진자수가 많은 5 개 국가의 데이터프레임이다.

- R

```
df_sunburst_cases <- rbind(
 ## 각 대륙별 전체 확진자수 데이터프레임
 df_covid19_stat |>
 filter(iso_code %in% c('OWID_AFR', 'OWID_ASI', 'OWID_EUR', 'OWID_NAM', 'OWID_OCE', 'OWID_SAM')) |>
 select(continent, location, 전체확진자수),
 ## 전체 확진자가 많은 대륙별 5 개국 데이터 프레임
 df_covid19_stat |>
  filter(!is.na(continent)) |>
  group_by(continent) |> top_n(5, wt = 전체확진자수) |>
```

```
  select(continent, location, 전체확진자수)
)

head(df_sunburst_cases, 10)

## # A tibble: 10 x 3
##    continent    location     전체확진자수
##    <chr>        <chr>           <dbl>
##  1 <NA>         Africa        12387706
##  2 <NA>         Asia         190549741
##  3 <NA>         Europe        237839549
##  4 <NA>         North America   116202256
##  5 <NA>         Oceania         12808969
##  6 <NA>         South America    64400672
##  7 South America Argentina       9721718
##  8 Oceania      Australia       10520099
##  9 South America Brazil         34803252
## 10 North America Canada          4424802
```

- python

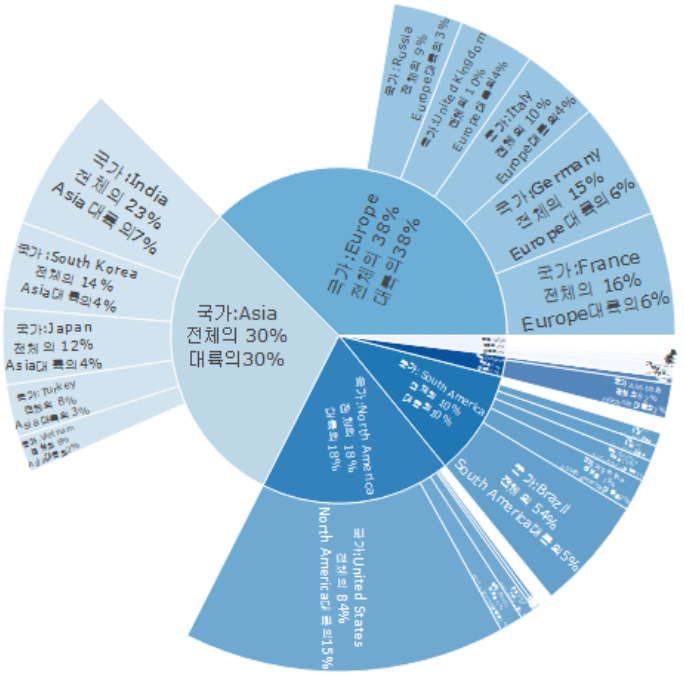
```
df_sunburst_cases = df_covid19_stat.copy().reset_index()
df_sunburst_cases = df_sunburst_cases[(df_sunburst_cases['iso_code'].isin(['OWID_ASI',
'OWID_EUR', 'OWID_OCE', 'OWID_NAM', 'OWID_SAM', 'OWID_AFR']))]
df_sunburst_cases = df_sunburst_cases[['iso_code', 'continent', 'location', '전체확진자수']]

df_sunburst_cases_1 = df_covid19_stat.copy().reset_index()
df_sunburst_cases_1 = df_sunburst_cases_1.dropna(subset = ['continent'])
df_sunburst_cases_1 = df_sunburst_cases_1.sort_values(by = '전체확진자수',
ascending=False).groupby('continent').head(5)
df_sunburst_cases_1 = df_sunburst_cases_1[['iso_code', 'continent', 'location', '전체확진자수']]

pd.concat([df_sunburst_cases, df_sunburst_cases_1])
```

전처리된 데이터프레임을 보면 대륙의 데이터를 나타내는 행에는 'continent' 열이 비어있다. 각 대륙의 확진자가 많은 5 개국은 'continent' 열에 대륙 이름이 기록되었다. 따라서 `parents` 속성이 'continent'열로 설정되고 전체 섹터의 이름으로 설정되는 `label` 은 'location'열로 설정되었다. 이들 섹터의 값은 전체 확진자 열로 설정된다.

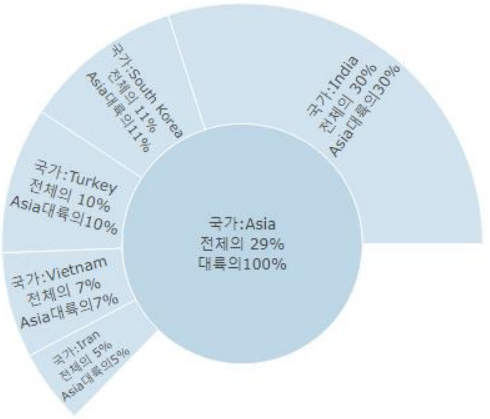
이제 전처리된 데이터를 사용하여 선버스트 차트를 만든다. `add_trace()`의 `type` 에 'sunburst'를 설정함으로써 해당 trace 가 선버스트 trace 임을 설정하고 선버스트 trace 에 필수적으로 필요한 `labels`, `parents`, `values` 속성에 전처리한 데이터 중 해당 열을 매핑하여 완성하였다.

- R

```
df_sunburst_cases |>
  plot_ly() |>
  add_trace(type = 'sunburst',
          labels = ~location,
          parents = ~continent,
          values = ~전체확진자수,
          branchvalues = 'total',
          insidetextorientation='radial',
          textinfo = 'label+percent parent+percent entry',
          texttemplate = '국가:%{label}<br>전체의 %{percentParent}<br>%{parent}대륙의%{percentEntry}'
          )
```



다른 trace 와는 달리 선버스트 trace 는 마우스 클릭에 따른 추가적인 사용자 반응 작용이 있다. 선버스트 차트에서 특정 섹터를 클릭하면 해당 부분 섹터가 확대되어 표시됨으로써 세부 데이터를 확인하기 쉬워진다.



아시아 섹터를 클릭한 선버스트 차트의 반응 결과

- python

```
fig = go.Figure()

fig.add_trace(go.Sunburst(
    labels = df_sunburst_cases['location'],
    parents = df_sunburst_cases['continent'],
    values = df_sunburst_cases['전체확진자수'],
    branchvalues = 'total',
    insidetextorientation='radial',
    textinfo = 'label+percent parent+percent entry',
    texttemplate = '국가:%{label}<br>전체의 %{percentParent}<br>%{parent}대륙의%{percentEntry}'
))
```

3.2. 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 `plotly` 에서 제공하는 trace 에 `sankey` 를 제공하기 때문에 추가적인 패키지의 도움없이 산키 다이어그램을 만들수 있지만 `ggplot2` 의 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 3 열(과정구분, 왼쪽 노드로 사용)과 12 열, 21 열부터 26 열(오른쪽 노드로 사용)까지를 선택
 select(3, 12, 21:26) |>
 ## 과정구분 열을 사용하여 그룹화
 group_by(과정구분) |>
 ## 전체 열에 대해 `sum`을 적용(summarise_all 은 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 105904 명이라는 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 from 에 저장
from <- unique(as.character(df_sankey$과정구분))

## 오른쪽 노드로 사용할 변량을 to 에 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_sankey 에서 '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 R 에서 사용하는 인덱스는 일반적으로 1 부터 시작하지만 여기서 사용하는 인덱스는 0 부터 시작한다는 것이다.

세번째로 노드 데이터 정보를 생성한다. 앞서 데이터를 전처리 할 때 노드 링크의 순서에 따라 데이터를 정리해 놓았기 때문에 df_sankey 의 학생수 열을 사용할 수 있다.

이제 데이터의 전처리가 끝났으니 `plotly` 의 `sankey` trace 를 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)
```

실행결과 6-3 산키 다이어그램

3.3. 스캐터폴라 차트

레이다 차트(Radar Chart)는 특정 단위 데이터를 여러 평가 항목에 따라 평가를 할 때 주로 사용된다. 평가 항목 수에 따른 다각형을 중심으로부터 측정 단위에 따라 일정 간격으로 척도를 재는 칸을 나누어 평가 항목 간 결과를 한눈에 볼 수 있도록 해주는 차트이다. 여러 측정 항목을 함께 겹쳐 놓아 비교하기에도 편리하고 항목 간 비율뿐만 아니라 균형과 경향을 직관적으로 알 수 있어 편리하다.

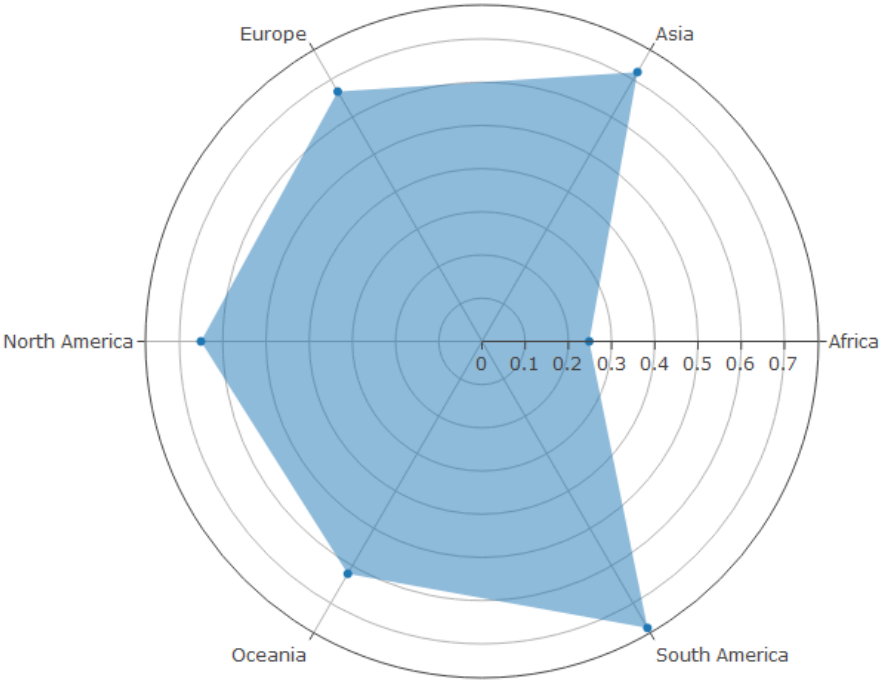
`plotly` 에서 레이다 차트를 그리는 타입이 레이다(radar) trace 가 아닌 스캐터폴라(scatterpolar) trace 라는 이름으로 사용한다. 스캐터폴라 trace 는 단지 레이더 차트를 그리기 위해 사용되는 trace 가 아니고 축을 X, Y 축이 아닌 각도(angular)와 반지름(radial)를 사용하는 극 좌표계(Ploar)를 사용하는 스캐터 trace 를 말한다. 따라서 `mode` 를 'markers'나 'text'로 사용하는 스캐터폴라 trace 를 사용할 수도 있다.

스캐터폴라 trace 에서 중요하게 사용되는 항목이 `theta` 와 `r`, `fill` 이다. `theta` 는 각도로 표현되는 축을 설정하는 속성으로 레이더 차트에서 평가 항목으로 설정해야하는 변수나 벡터를 설정한다. `r` 은 반지름으로 표현되는 축을 설정하는 속성으로 레이터 차트에서 평가

항목의 측정값을 설정한다. `fill` 은 단색으로 채울 영역을 설정하는 속성이다. 이 속성은 'none', 'toself', 'tonext'의 세가지 값을 가지는데 'toself'는 trace 의 시작점과 끝점을 연결해 닫힌 모양으로 만들고 해당 trace 에 설정된 색을 채우고 trace 가 여러 개일 경우 trace 가 겹치는 곳의 색은 양 trace 의 색이 겹쳐 표현된다. 'tonext'도 시작점과 끝점을 연결해 닫힌 모양으로 만들고 내부를 채우는데 완전히 겹쳐지는 trace 의 색을 설정된 색을 사용한다는 것이 차이이다. 'none'은 내부를 채우지 않다.

- R

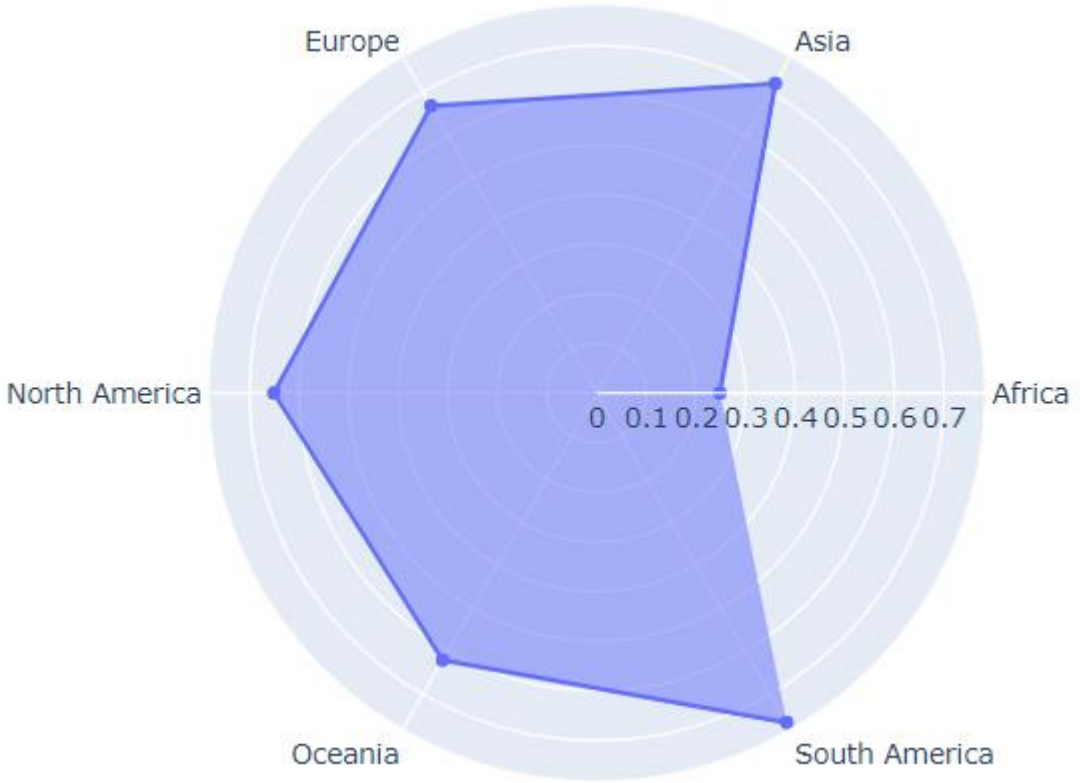
```
df_radar_veccine <- df_covid19_stat |>
 filter(iso_code %in% c('OWID_AFR', 'OWID_ASI', 'OWID_EUR', 'OWID_NAM', 'OWID_OCE', 'OWID_SAM')) |>
 select(continent, location, 백신접종완료률)

df_radar_veccine |>
 plot_ly() |>
 add_trace(type = 'scatterpolar',
        theta = ~location,
        r = ~백신접종완료률,
        fill = 'toself'
        )
```



- python

```
df_radar_veccine = df_covid19_stat.copy().reset_index()
df_radar_veccine = df_radar_veccine[(df_radar_veccine['iso_code'].isin(['OWID_ASI', 'OWID_EUR',
'OWID_OCE', 'OWID_NAM', 'OWID_SAM', 'OWID_AFR']))]
```

```
df_radar_veccine
fig = go.Figure()

fig.add_trace(go.Scatterpolar(
     theta = df_radar_veccine['location'],
            r = df_radar_veccine['백신접종완료률'],
            fill = 'toself'

))
```



3.4. 트리맵

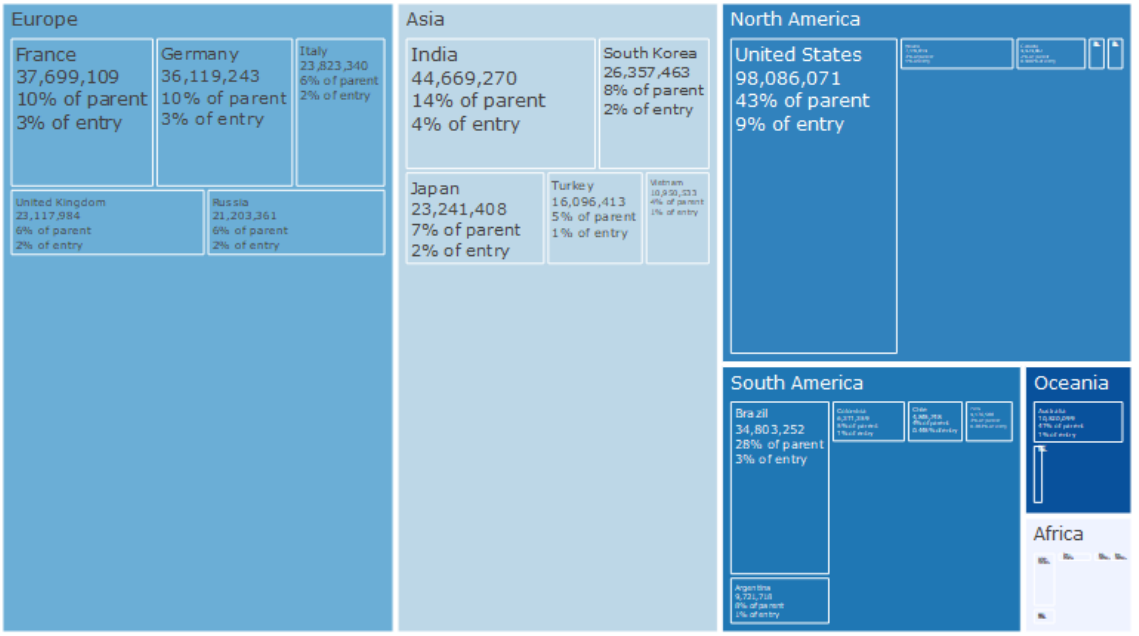
구성의 시각화 방법을 보다보면 하나 특징적인 것이 원으로 데이터의 비율을 표현하는 시각화 방법이 많다는 점이다. 이중 가장 대표적인 것이 파이차트, 도넛차트 등이며 선버스트나 레이더 차트도 원형이다. 하지만 시각화는 네모난 종이위에 표현되거나 네모난 화면위에 표현된다. 따라서 원형으로 표현되는 차트들 네모난 종이나 화면에 표현하면 시각화에 사용되지 못하는 공간이 많아져서 전체 영역을 호과적으로 사용하지 못한다. 이러한 단점을 극복하기 위해 네모난 사각형을 사용하여 전체의 비율을 표현한 시각화 방법이 트리맵이다.

트리맵은 잎(및/또는 외부 가지)에서 루트를 향하는 계층적 데이터를 직사각형으로 시각화한다. 앞서 설명한 선버스트 차트의 사각형 버전이다. 그래서 선버스트에서 사용한 `labels`, `parents`, `values` 속성을 동일하게 사용할 수 있다.

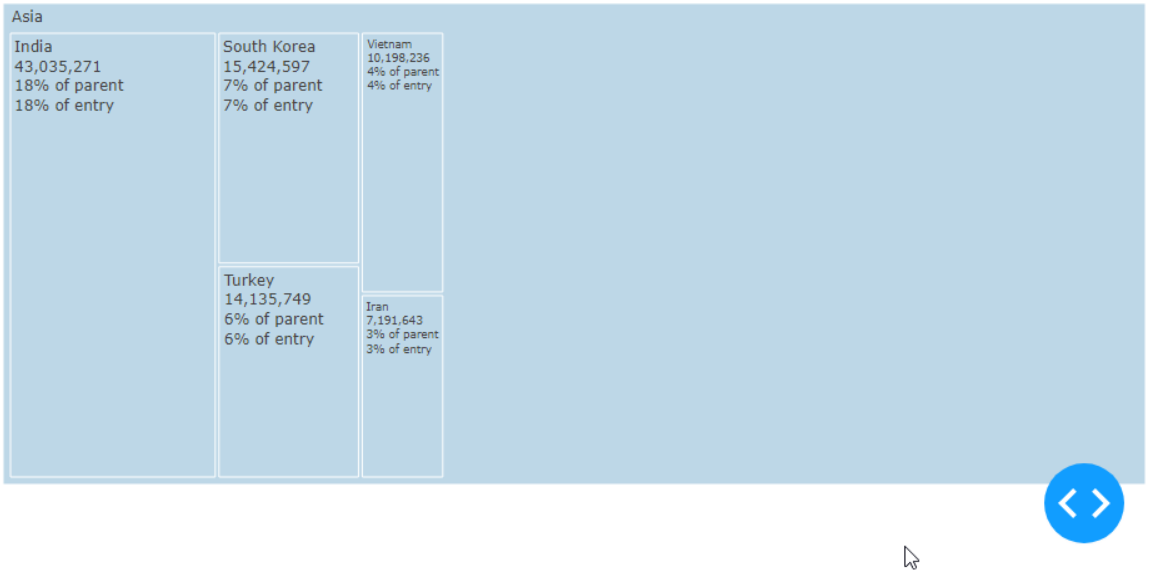
다음은 앞서 만들었던 선버스트 trace 를 그대로 트리맵 trace 로 만든 코드이다.

- R

```
df_sunburst_cases |>
 plot_ly() |>
 add_trace(type = 'treemap',
        labels = ~location,
        parents = ~continent,
        values = ~전체확진자수,
        textinfo = 'label+value+percent parent+percent entry'
        )
```



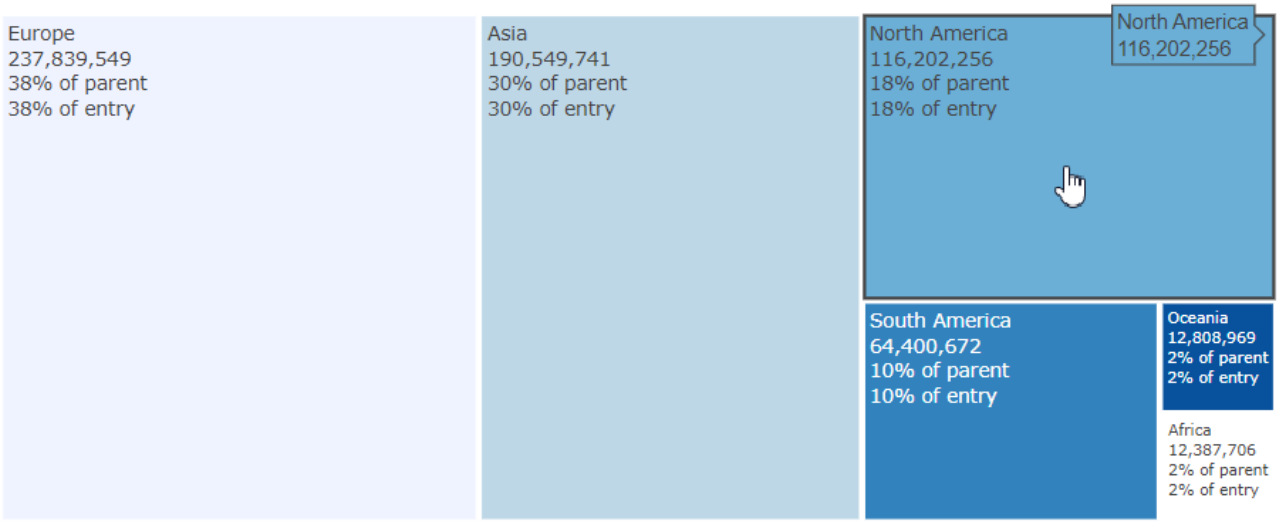
트리맵도 선버스트와 같이 마우스 클릭으로 세부 섹터를 확대하여 사용할 수 있다.



아시아 섹터를 클릭한 트리맵 차트의 반응 결과

- python

```
fig = go.Figure()
fig.add_trace(go.Treemap(
    labels = df_sunburst_cases['location'],
    parents = df_sunburst_cases['continent'],
    values = df_sunburst_cases['전체확진자수'],
    textinfo = 'label+value+percent parent+percent entry'

))
```



3.5. 벤 다이어그램

벤다이어그램은 아마도 중학교 시절 한번식은 그려본 차트일 것이다. 사실 이 차트는 지금까지 줄기차게 설명했던 데이터의 시각화와는 좀 다른 형태의 시각화이다. 우선 데이터의 양에 직접적인 관계가 없고 데이터의 성질에 관계가 있다는 점에서 정량적 시각화가 아닌 정성적 시각화이다. 각 데이터의 크기를 비교하는 것이 아닌 데이터간의 관계를 표현한다는 점도 다른 시각화와 매우 다르다. 하지만 데이터의 전반적 구조를 표현하거나 데이터 분석을 통해 도출된 인사이트를 전달하는 과정에서 효과적으로 사용될 수 있는 시각화이다.

R 의 많은 사용자들은 데이터와 직접적 관계가 없는 벤다이어그램을 R 에서 그린다는게 어색할 것이다. 보통 이렇게 데이터와 직접적 관계가 없는 시각화는 파워포인트와 같은 프리젠테이션 전용 툴이나 일러스트레이터와 같은 전문 그래픽 툴을 사용하는 것이 일반적이다. 하지만 R 마크다운을 사용하여 R 에서 직접적으로 문서를 만들거나 그래픽 툴에 익숙하지 않은 데이터 분석가들을 위해 R 에서 벤다이어그램을 그릴 수 있도는 패키지가 제공되고 있다.

아쉽게도 `plotly` 나 `ggplot2` 에서는 벤다이어그램을 지원하지 않는다. 하지만 R 에서 벤다이어그램을 만들 수 있도록 `ggplot2` 의 확장 패키지로 `ggVennDiagram` 패키지가

지원된다.[1] 이 패키지에서 제공하는 `ggVennDiagram()`을 이용해서 벤다이어그램을 만들 수 있다.

벤다이어그램을 그리기 위해서는 먼저 `ggVennDiagram` 패키지를 설치한다.

```
## ggVennDiagram 설치
if(!require(ggVennDiagram)) {
  install.packages('ggVennDiagram')
  library(ggVennDiagram)
}
```

`ggVennDiagram` 패키지에서 제공하는 `ggVennDiagram()`의 사용법은 다음과 같다.

ggVennDiagram(x, category.names = names(x), show_intersect = FALSE, set_color = "black", set_size = NA, label = c("both", "count", "percent", "none"), label_alpha = 0.5, label_geom = c("label", "text"), label_color = "black", label_size = NA, label_percent_digit = 0, label_txtWidth = 40, edge_lty = "solid", edge_size = 1, ...)

- x : 벤다이어그램에 사용 할 벡터 리스트
- category.names : 각각의 원에 해당하는 카테고리의 이름 설정
- show_intersect : 인터랙티브 플롯(plotly)으로 생성할지 여부를 결정하는 논리값
- set_color : 벤다이어그램 안의 텍스트 라벨 색 설정
- set_size : 벤다이어그램 안의 텍스트 라벨 크기 설정
- label : 텍스트 라벨의 표시 형태 설정, "both", "count", "percent", "none" 중에 하나 설정
- label_alpha : 벤다이어그램 안의 텍스트 라벨 투명도 설정
- label_geom : 벤다이어그램 안의 텍스트 기하요소 설정, "label", "text" 중에 하나 설정
- label_color : 벤다이어그램 안의 텍스트 색 설정
- label_size : 벤다이어그램 안의 라벨 크기 설정
- label_percent_digit : 라벨 타입이 'percent'일 경우 소수점 아래 몇자리까지 출력할지 설정
- label_txtWidth : 벤다이어그램 교차부분의 텍스트 너비 설정

---

[1] 이 패키지외에 `VennDiagram` 패키지도 있지만 `VennDiagram` 패키지에서는 한글을 지원하지 못한다는 결정적 단점이 있다.

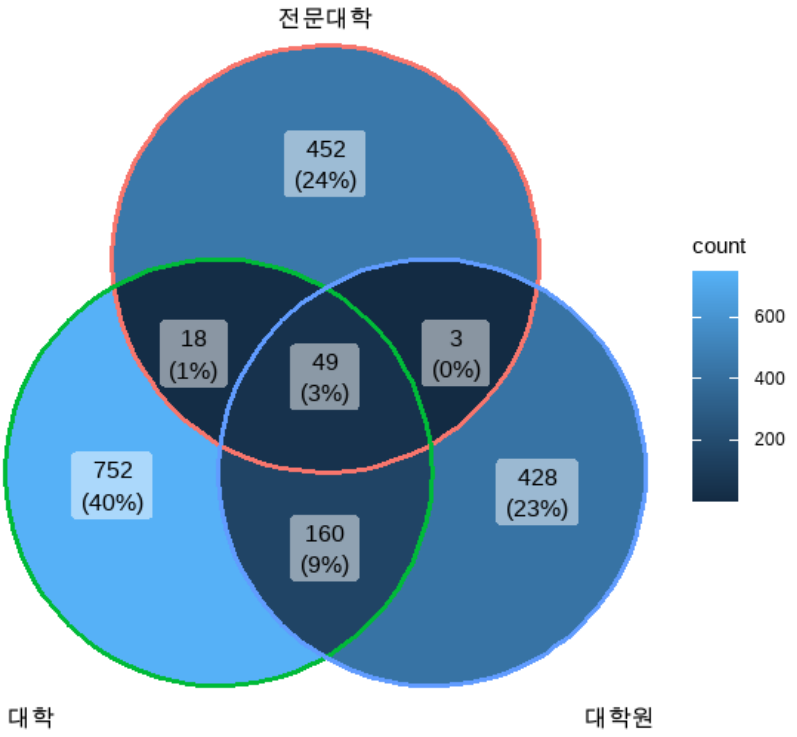
- edge_lty : 벤다이어그램 가장자리 라인 타입 설정

- edge_size : 벤다이어그램 가장자리 라인 굵기 설정

벤다이어그램은 보통 3 개의 원으로 표현되는 것이 효율적이다. 그 이상의 원으로 표현은 가능하겠지만 오히려 데이터의 특징을 전반적으로 확인하는데 방해되는 경우도 많다.

벤다이어그램을 그리기 위해서는 먼저 벤다이어그램에 사용할 데이터 벡터를 생성해야 한다. 여기서는 공학계열의 학과들을 과정구분별로 필터링해서 세개의 벡터를 생성하고 이에 대한 벤다이어그램을 그리도록 한다.

```
## 전문대학과정의 공학계열 학과명을 벡터로 저장
vec_전문대학과명 <- df_취업률 |>
 filter(대계열 == '공학계열', 과정구분 == '전문대학과정') |>
 select(학과명) |>
 ## 하나의 열만 선택하더라도 여전히 데이터프레임이기 때문에 pull()로 벡터화
 pull()

## 대학과정의 공학계열 학과명을 벡터로 저장
vec_대학학과명 <- df_취업률 |>
 filter(대계열 == '공학계열', 과정구분 == '대학과정') |>
 select(학과명) |>
 pull()

## 대학원과정의 공학계열 학과명을 벡터로 저장
vec_대학원학과명 <- df_취업률 |>
 filter(대계열 == '공학계열', 과정구분 == '대학원과정') |>
 select(학과명) |>
 pull()

## 벤다이어그램의 제목 설정과 데이터를 연결한 리스트 생성
list_venn_diagram <- list(전문대학 = vec_전문대학과명, 대학 = vec_대학학과명, 대학원 = vec_대학원학과명)

## 벤다이어그램 생성
ggVennDiagram(list_venn_diagram)
```



실행결과 6-4 기본 벤다이어그램

3.6. 업셋 그래프

중복이 허용되면서 변량이 많지 않은 데이터를 시각화하는 방법으로 많이 활용되는 것이 벤다이어그램이다. 벤다이어그램은 10 장에서 설명하겠지만 원으로 변량을 표현해 중복에 대한 표현이 가능한 장점이 있다. 하지만 원(변량)이 3 개를 넘어가면 매우 혼란스러워진다. 이 벤다이어그램의 단점을 극복하기 위해 2014 년에 제안된 시각화 방법이 업셋 그래프이다.[2]

업셋 그래프는 그래프의 아래쪽에 해당 데이터를 구성하는 세트의 조합이 표현되고 위쪽에는 해당 세트의 조합에 대한 데이터 값을 표현하는 막대 그래프가 표현된다. 이렇게 구성함으로써 단일 세트만 가능했던 막대 그래프에 변량의 세트에 대한 표현이 가능해져서 벤다이어그램에서 해석이 어려웠던 다양한 데이터의 조합에 대한 시각화에 매우 효과적이다.

---

[2] Alexander Lex, Nils Gehlenborg, Hendrik Strobelt, Romain Vuillemot, Hanspeter Pfister
UpSet: Visualization of Intersecting Sets
IEEE Transactions on Visualization and Computer Graphics (InfoVis), 20(12): 1983–1992, doi:10.1109/TVCG.2014.2346248, 2014.

업셋 그래프는 3 개 이상 30 개 미만의 집합 데이터에 가장 적합하다 . 4 세트 미만의 경우 익숙한 벤 다이어그램이 친숙하다. 또 업셋 그래프는 집합 데이터의 분포를 분석하는 데 적합하기때문에 교차하는 집합의 조합의 빈도확인에 매우 효과적이다.[3]

그럼 이제 업셋 그래프를 만들어 보겠다. df_취업통계 데이터는 각 학과를 7 개의 학제(전문대학(2 년제), 전문대학(3 년제), 대학교, 산업대학 등)로 구분되어 있다. 따라서 같은 학과명을 가지더라도 서로 학제가 달라 최대 7 개의 행이 있을 수 있다. 그래서 동일한 학과명이 여러 학제에 걸쳐 존재하는 경우가 많은데 이 분포를 알아내기 위해 업셋 그래프를 만든다.

이 업셋 그래프를 만들기 위해 사용하는 데이터는 이 df_취업통계 데이터로 각각의 학과명이 속한 학제를 리스트로 만든 열이 필요하다. 이 열을 기준으로 빈도를 표현한 막대 그래프를 그릴 것이다. 이를 위해 다음과 같이 데이터를 전처리 한다.

```
## df_취업통계에서
df_과정구분_upset <- df_취업률 |>
 ## 학과명으로 그룹화
 group_by(학과명) |>
 ## 학제를 리스트로 만든 학과리스트 열을 생성(각각의 그룹의 행에 학과리스트 열은 모두 같은 값을 가지기 때문에 맨 아래에서 중복을 제거)
 mutate(학제리스트 = list(unique(sort(학제)))) |>
 ## 필요한 열만 선택
 select(3:6, 8, 학제리스트, 11) |>
 ## 중복을 제거
 unique()
```

전처리가 끝났으면 본격적으로 업셋 그래프를 만든다. 이를 위해 먼저 `ggupset` 패키지를 설치해야한다.

```
## ggupset 패키지 설치
if(!require(ggupset)) {
 install.packages('ggupset')
 library(ggupset)
}
```

업셋 그래프를 그리는 코드는 다음과 같다. 먼저 학과명의 조합 리스트로 구성된 학과명을 X 축으로 매핑한 geom_bar 레이어를 생성한다. 이후 X 축의 스케일를 업셋 스케일로

---

[3] https://upset.app/

바꾸어주면 업셋의 기본 그래프가 생성된다. 축의 스케일을 업셋 스케일로 바꾸려면 `scale_x_upset()`을 사용한다.

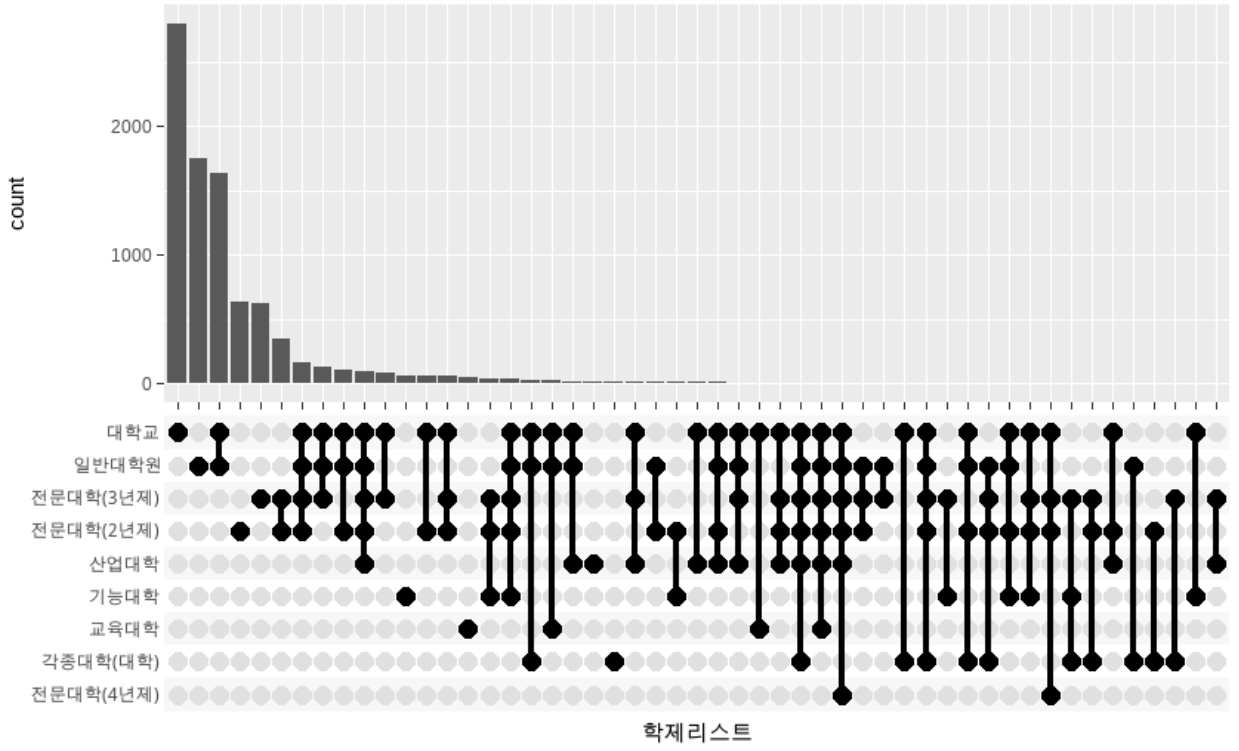
scale_x_upset(order_by = c("freq", "degree"), n_sets = Inf, n_intersections = Inf, sets = NULL,

intersections = NULL, reverse = FALSE, ytrans = "identity", ..., position = "bottom")

- order_by : X 축의 순서 설정을 어떻게 할지 설정
- n_sets : 전체 셋의 최대값 설정
- n_intersections : 표시될 교차 셋의 최대값 설정
- sets : 표시하고 싶은 셋의 이름 벡터 설정
- intersections : 표시될 교차 셋의 이름 벡터 설정
- reverse : 교차값의 역순으로 설정할 논리값 설정
- ytrans : Y 축의 변환 설정
- ... : 이산형 스케일과 관련한 추가 매개변수
- position : X 축을 아래에 위치할 것인지 위에 위치할 것인지 설정

빈도수를 표현하는 `geom_bar()`의 X 축 스케일을 `scale_x_upset()`으로 설정하여 업셋 그래프를 그리는 코드는 다음과 같다.

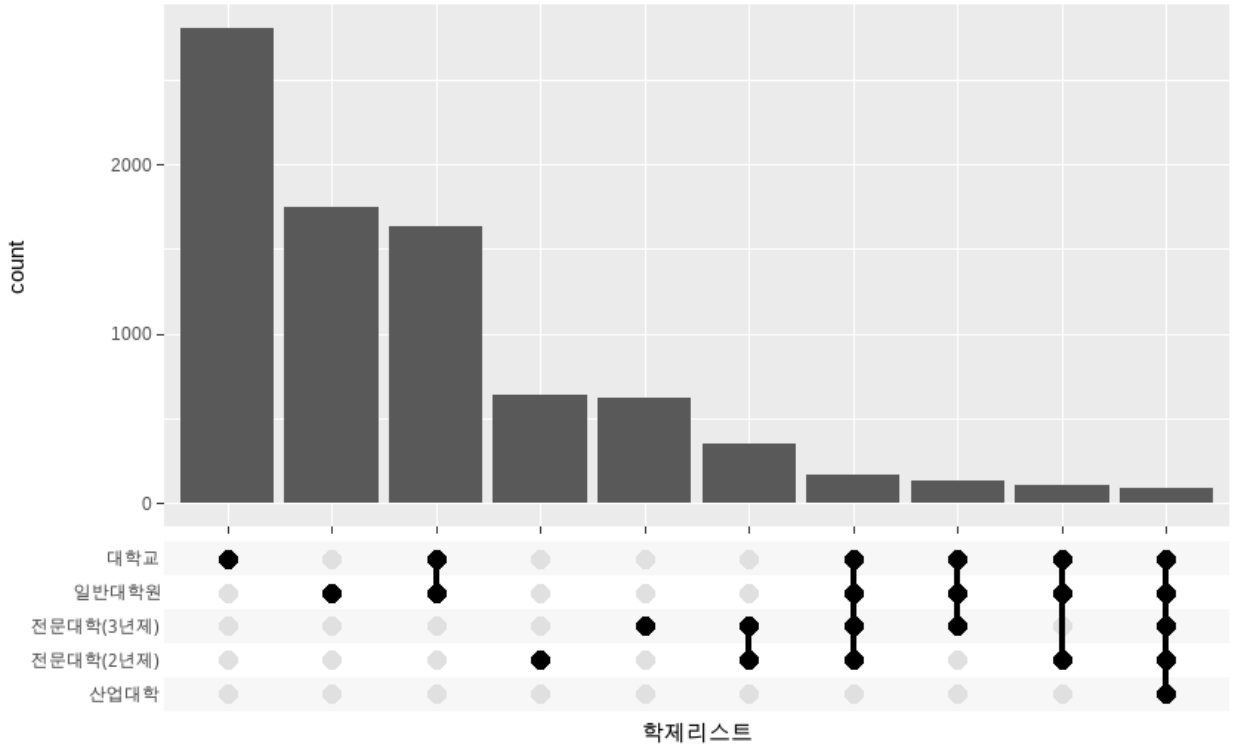
```
library(ggupset)

df_과정구분_upset |> ggplot() +
 ## X 축을 학제리스트로 매핑한 geom_bar 레이어 생성
 geom_bar(aes(x=학제리스트)) +
 ## X 축 스케일을 업셋 스케일로 설정
 scale_x_upset()
```



실행결과 6-5 기본 업셋 그래프

7 개의 학제에 대한 조합이 표현되다보니 조합의 수가 너무 많아 보인다. 이 중 빈도가 많은 10 개의 조합만 표시하도록하는 코드는 다음과 같다.

```
ggplot(df_과정구분_upset, aes(x=학제리스트)) +
 geom_bar() +
 ## n_intersection 을 10 으로 설정해서 X 축 변량을 10 개로 한정
 scale_x_upset(n_intersections = 10)
```

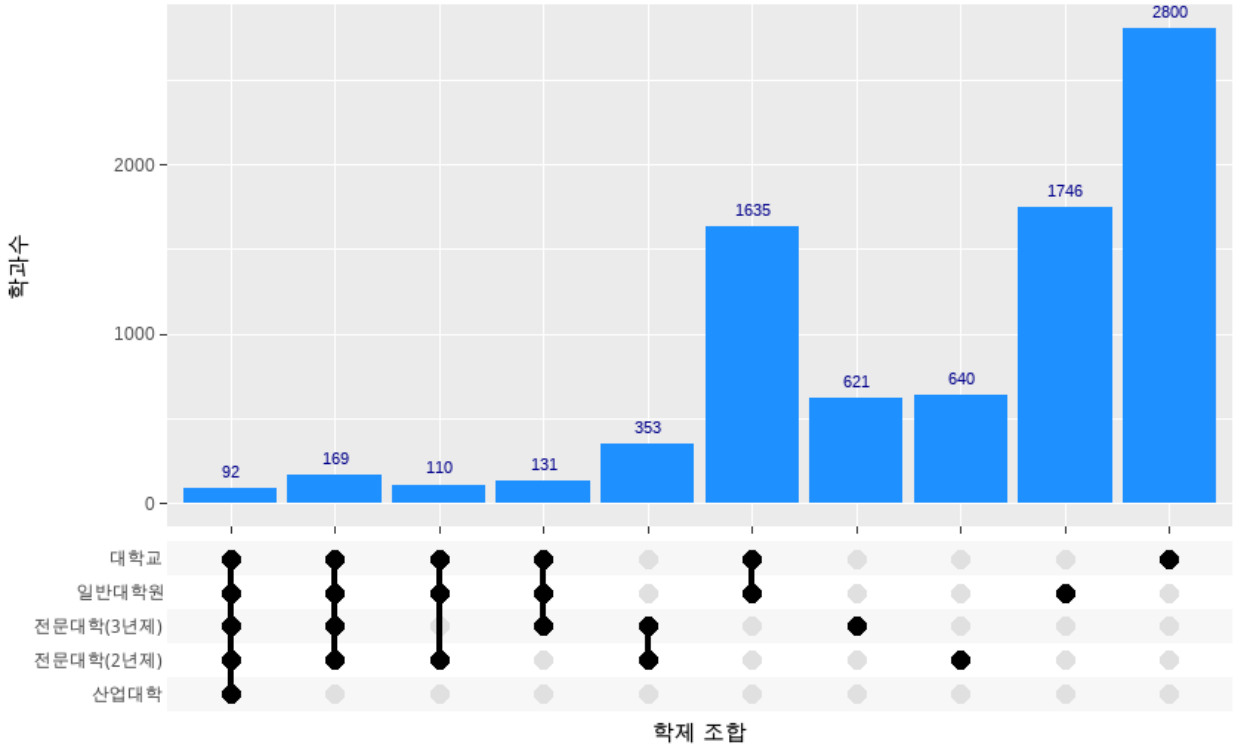


실행결과 6-6 기본 업셋 그래프

앞의 그래프를 보면 대학교에만 존재하는 학과가 가장 많고 다음은 일반대학원이다. 세번째로 많은 분포는 대학교와 일반대학원에 같이 존재하는 학과가 있다는 것을 알 수 있다.

앞의 업셋 그래프의 조합 매트릭스의 조합이 많은 순으로 정렬 순서를 바꾸면 다음과 같이 바꿀 수 있다.

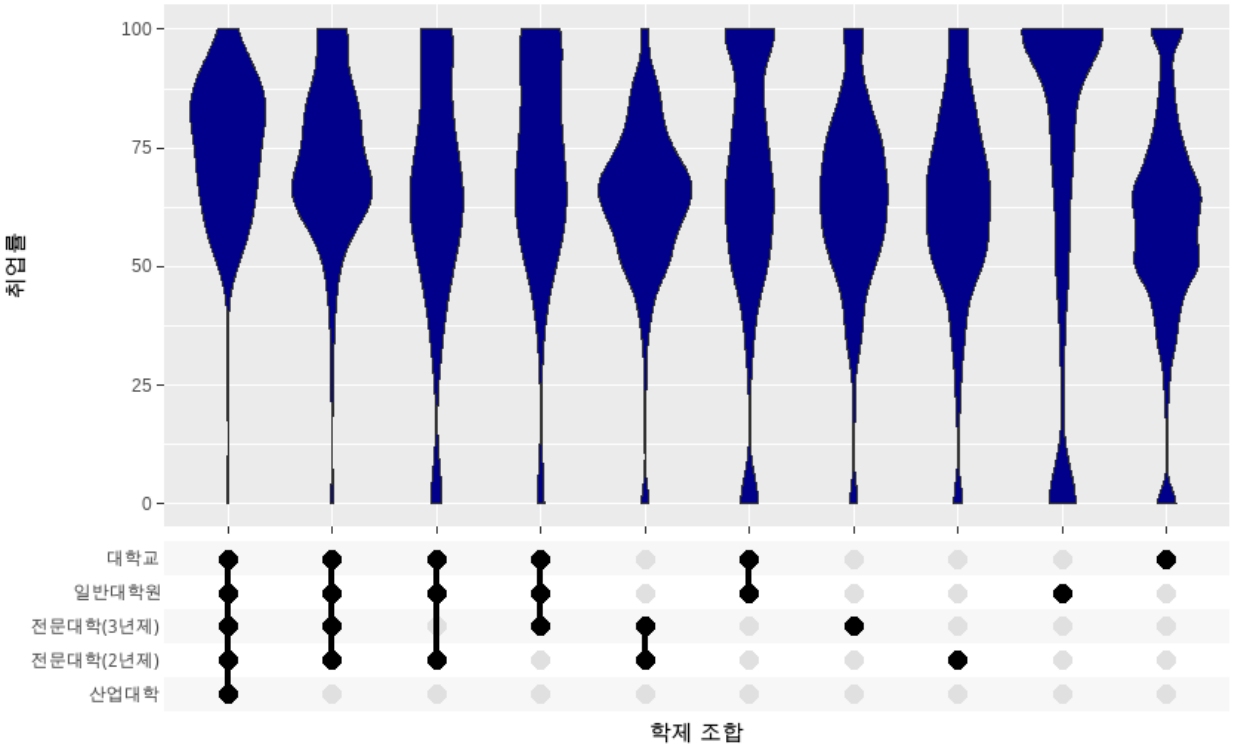
```
df_과정구분_upset |>
 ggplot() +
 geom_bar(aes(x=학제리스트), fill = "dodgerblue") +
 geom_text(aes(x=학제리스트, label = ..count..), stat = 'count', vjust = -1, size = 3, color = "darkblue") +
 ## order_by 를 'degree'로 설정하여 매트릭스 조합을 정렬 기준으로 맞추고 내림차순으로 설정하기 위해
reverse 를 설정
 scale_x_upset(n_intersections = 10,
           order_by="degree",
           reverse = TRUE) +
 ylab("학과수") + xlab("학제 조합")
```



실행결과 6-7 조합순으로 정렬된 업셋 그래프

업셋 그래프에 사용되는 상위 그래프는 막대 그래프외에 모든 그래프를 사용할 수 있다. 다만 X 축의 조합 매트릭스를 설정함으로써 데이터를 잘 설명할 수 있는지 고려하여 설정하는 것이 좋다. 다음은 앞의 빈도수의 막대 그래프를 취업률에 대한 바이올린 그래프로 바꾼 코드이다.

```
df_과정구분_upset |>
 ggplot() +
 ## 상부 그래프의 레이어를 geom_violin 레이어로 생성
 geom_violin(aes(x=학제리스트, y = 취업률_계), fill = "darkblue") +
 scale_x_upset(n_intersections = 10,  order_by="degree", reverse = TRUE) +
 ylab("취업률") + xlab("학제 조합")
```



실행결과 6-8 바이올린 업셋 그래프