IV. 비교(Compare)와 구성(Composition)의 시각화

비교와 구성의 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 통계 처리가 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 시각화에는 막대 그래프가 많이 사용되는데 순위 막대 그래프, 롤리팝 그래프, 도트 그래프 등이 많이 사용되고 구성의 시각화는 파이 차트, 선버스트 차트, 산키 다이어그램등이 많이 사용된다.

1. 막대 그래프

막대 그래프는 이산형 데이터 또는 그룹화되어 이산형으로 집계된 데이터들을 막대의 길이를 사용하여 서로 비교하기 위해 사용되는 시각화이다. 공통 기준선에서부터 시작된 직사각형의 막대를 사용하여 각 변량의 데이터를 표현하고 막대의 길이는 데이터 값에 비례한다. 같은 기준선에서 시작한 막대들이기 때문에 막대의 끝나는 위치에 따라 데이터들이 비교될 수 있다. 또 막대의 크기에 따른 데이터의 증감을 비교할 수도 있다.

막대 그래프에는 그 유형에 따라 수직, 수평 막대 그래프, 스택 막대 그래프, 그룹 막대 그래프의 세 가지로 구분된다.

수직, 수평 막대 그래프는 이산형 데이터 각각의 변량에 각각 하나씩의 막대가 표시되는, 우리가 흔히 아는 형태의 막대 그래프이다. 다만 그 막대의 표시 방향에 따라 수직 막대 그래프와 수평 막대 그래프로 나눌 수 있다. 보통 시각화에 사용하는 보고서는 세로 사이즈보다 가로 사이즈가 작은 경우가 많고, 스크롤이 가능한 웹화면이라고 해도 세로 방향 스크롤은 익숙하지만 가로방향 스크롤은 익숙하지 않다. 따라서 막대로 표현해야 할 변량이 많다면 수직막대그래프로 표현할 수 있는 범위에 한계가 있을 수 밖에 없다. 그래서 막대의 폭이 너무 작아지게 되고 X 축에 표시되는 축 라벨이 겹치게 되는 현상이 발생할 수 밖에 없다. 따라서 표현해야 할 막대가 많을 경우에는 수평 막대 그래프를 사용하여 아래쪽으로 길게 만들면 막대의 폭도 적절히 설정가능하고 축 라벨도 정상적으로 표현할 수 있다.

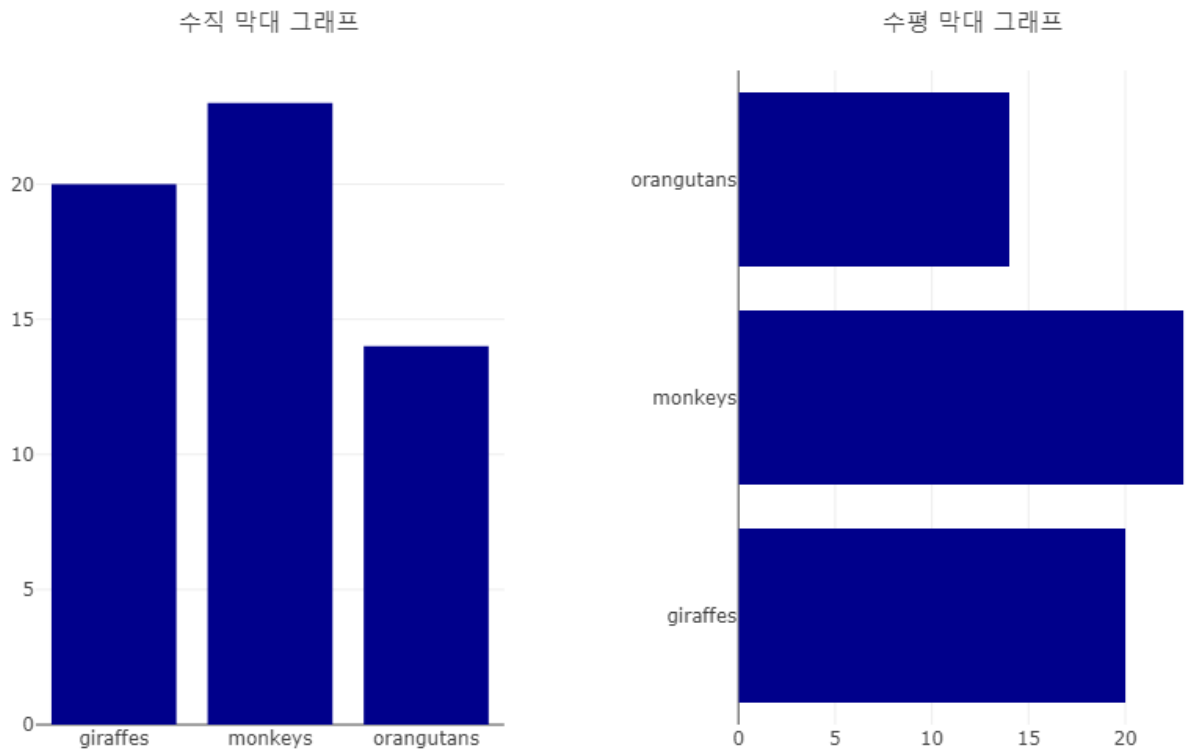


그림 IV-1. 수평 막대 그래프와 수직 막대 그래프

스택 막대 그래프와 그룹 막대 그래프는 X 축의 이산형 변수에 매핑된 변량들을 다시 또 다른 변수로 구분해서 막대를 그려야할 때 사용하는 막대 그래프이다. 스택 막대 그래프는 하나의 막대를 또 다른 변수의 변량으로 구분하여 쌓아 올리는 형태로 사용하는 방법이다. 스택 막대 그래프는 하나의 변량에 대한 세부 변수의 구성 비율을 살펴보기 쉽다는 장점이 있다. 따라서 이 방법은 비교를 위한 시각화라기 보다는 구성을 위한 시각화로 볼 수 있다. 반면 그룹 막대 그래프는 X 축의 하나의 변량에 해당하는 또 다른 변수의 변량 만큼의 막대를 묶어 그려주는

방법이다. 따라서 이 방법은 X 축의 주 변량을 구성하는 세부 변량의 크기들을 서로 비교할 수 있고 주 변량의 세부 변량들끼리도 비교할 수 있다는 장점이 있지만 구성을 비교하기 어렵다는 단점도 있다.

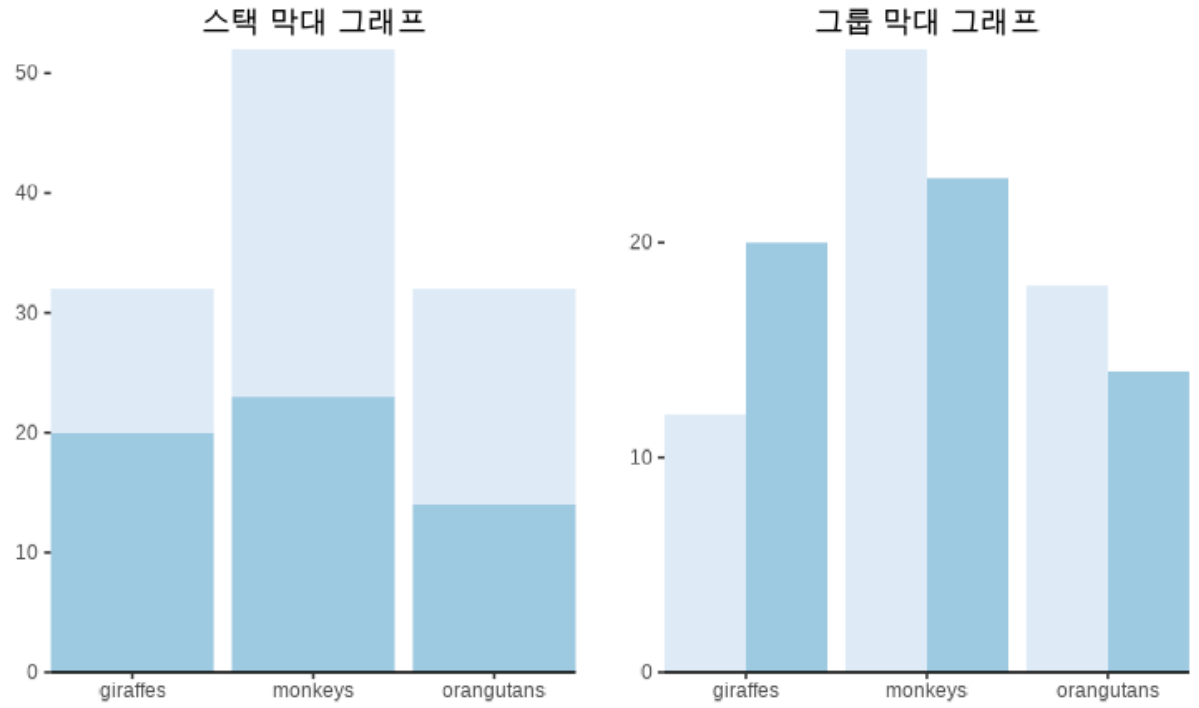


그림 IV-2. 스택 막대 그래프와 그룹 막대 그래프

막대 그래프를 그릴때는 다음과 같이 몇 가지 주의해아할 점이 있다.

막대 사이의 간격을 잘 설정해야 한다. 앞 장에서 설명한 히스토그램은 일반적으로 연속형 변수에 대한 막대 그래프이기 때문에 그 간격을 설정하지 않는다. 그 사이의 간격이 마치 연속형 변수에서 데이터가 없는 구간으로 오해될 수 있기 때문이다. 하지만 막대 그래프는 이산형 데이터에 대한 그래프이기 때문에 각각의 이산형 변량들을 구분하기 위해 적절한 간격을 두어야 한다. 이 간격을 만들때는 너무 좁은 간격을 두면 얼핏 보았을 때 데이터 변량의 구분이 어려울 수 있고 너무 넓으면 데이터 간격이 넓어 데이터의 비교에 어려움이 따를 수 있다. 보통 엑셀에서 자동적으로 생성되는 막대 그래프는 막대 너비의 30~40%의 여백을 둔다고 한다.

막대 그래프는 데이터의 비교를 쉽게 하기 위해 막대의 크기대로 정렬하는 것이 좋다. 하지만 이산형 변수의 변량값 자체에 의미가 있는 경우는 정렬하지 않는 경우도 많이 있다. 예를 들어

이산형 변수가 1 부터 5 까지 중에 선택하는 리커르트 척도나 년도, 월과 같은 시간의 경우처럼 이산형 변수 자체의 순서가 있는 경우는 정렬하지 않는다.

막대의 기준선은 0 부터 시작한다. 많은 경우 0 부터 시작하는 막대 그래프들의 크기 비교가 쉽지 않을 경우 기준선을 옮김으로써 비교가 도드라지게 시각화하는 경우가 있다. 예를 들어 어떤 두 개의 데이터가 하나는 98%, 하나는 99%라면 두 개의 막대 길이를 비교하기란 쉽지 않다. 그래서 기준선을 95%로 설정한다면 쉽게 비교될 수는 있겠지만 그 차이 때문에 사용자는 고작 1%의 차이를 매우 큰 차이로 오해하기 쉽다.

막대 그래프에서 트레이스는 막대로 표현하고자 하는 하나의 이산형 변수를 표현한다. 수평 막대 그래프나 수직 막대 그래프는 단일 이산형 변수에 따른 연속형 변수의 표현에 사용되기 때문에 하나의 bar 트레이스로 표현이 가능하다. 하지만 스택 막대 그래프나 그룹 막대 그래프는 이산형 변수를 추가하기 때문에 bar 트레이스를 추가하여야 한다.

plotly 에서 막대 그래프를 그리기 위해서는 bar 트레이스를 사용한다. R 에서는 `add_bars()`나 `add_trace(type = 'bars')`를 사용하고 python 에서는 `plotly.graph_objects.Bar()`나 `plotly.express.bar()`를 사용한다. 스택 막대 그래프나 그룹 막대 그래프의 경우는 추가적인 bar 트레이스를 `add_trace()`로 추가한 후 'layout' 속성의 'barmode'를 'stack'이나 'group'을 설정하여 스택 막대 그래프나 그룹 막대 그래프를 그릴 수 있다.

다음은 bar 트레이스에서 사용하는 주요 속성이다.

속성			속성 설명	속성값
base			막대가 그려지는 기본 위치 설정. Sets where the bar base is drawn (in position axis units). In "stack" or "relative" barmode, traces that set "base" will be excluded and drawn in "overlay" mode instead.	수치나 좌표상의 변량값
width			막대의 두께(축의 단위) 설정	0보다 큰 수치나 수치 배열
orientation			막대의 표시 방향 설정	("v" | "h")
marker	color		막대의 색상 설정	색상이나 색상 배열
	colorscale		색 스케일의 설정, marker.color가 수치형 배열일 경우에만 효과가 있음. 'colorscale'은 rgb, rgba, hex, hsl, hsv, 문자형 색 이름의 배열이여야 함. 이러한 'colorscale' 배열 대신 팔레트 이름을 쓸 수 있음 : Blackbody,Bluered,Blues,Cividis,Earth,Electric,Greens,Greys,Hot,Jet,Picnic,Portland,Rainbow,RdBu,Reds,Viridis,YlGnBu,YlOrRd.	컬러스케일
	line	color	막대 선 색을 설정.	색상이나 색상 배열
		colorscale	마커 선의 색 스케일의 설정	컬러스케일
		reversescale	True일 경우, 막대 선 색상 매핑을 역순으로 바꿈	논리값
		width	px 단위의 선 두께 설정	0보다 큰 수치나 수치 배열
	opacity		막대의 투명도 설정	0과 1사이의 수치나 배열
	pattern	bgcolor	패턴의 배경 색 설정	색상이나 색상 배열
		fgcolor	패턴의 전경 색 설정	색상이나 색상 배열
		fgopacity	패턴 전경 투명도 설정	0과 1사이의 수치
		fillmode	marker.color가 배경으로 사용되는지 전경으로 사용되는 지를 설정	("replace" | "overlay")
		shape	패턴이 채워질 모양 설정	("" | "/" | "\" | "x" | "-" | "|" | "+" | ".")
		size	픽셀 단위로 패턴이 채우질 크기의 설정	0보다 큰 수치나 수치 배열
	reversescale		True일 경우, 막대 색상 매핑을 역순으로 바꿈	논리값
insidetextanchor			박스 안쪽에 위치하는 텍스트의 위치 설정	("end" | "middle" | "start")

다음은 bar 트레이스에서 사용되는 'layout'의 주요속성이다.

속성	속성 설명	속성값
bargap	막대 사이의 간격 설정	0과 1사이의 수치
bargroupgap	막대 그룹사이의 간격 설정	0과 1사이의 수치
barmode	막대 그래프 모드 설정	("stack" | "group" | "overlay" | "relative")
barnorm	막대 트레이스의 표준화 방법 설정	("" | "fraction" | "percent")

1.1.1. 수직 막대 그래프

다음의 코드는 대학의 계열별 취업률의 평균에 대한 수직 막대 그래프를 그리는 R 과 python 의 코드이다. 대학의 계열은 총 7 개의 이산형 변수이고 취업률은 0 에서부터 100 까지의 연속형 변수이다. 따라서 X 축에는 대학의 계열을 매핑시키고 Y 축에 취업률을 매핑시키는 bar 트레이스로 그릴 수 있다. 여기에 데이터를 표시하기 위해 'text' 속성과 'textposition' 속성을 설정하였고 소수점 한 자리만 표현하기 위해 'texttemplate'를 설정하였다. 또 Y 축에 매핑되는 데이터는 백분률 데이터이기 때문에 'ticksuffix' 속성을 사용하여 눈금 라벨의 접미어를 "%"로 설정하였다.

- R

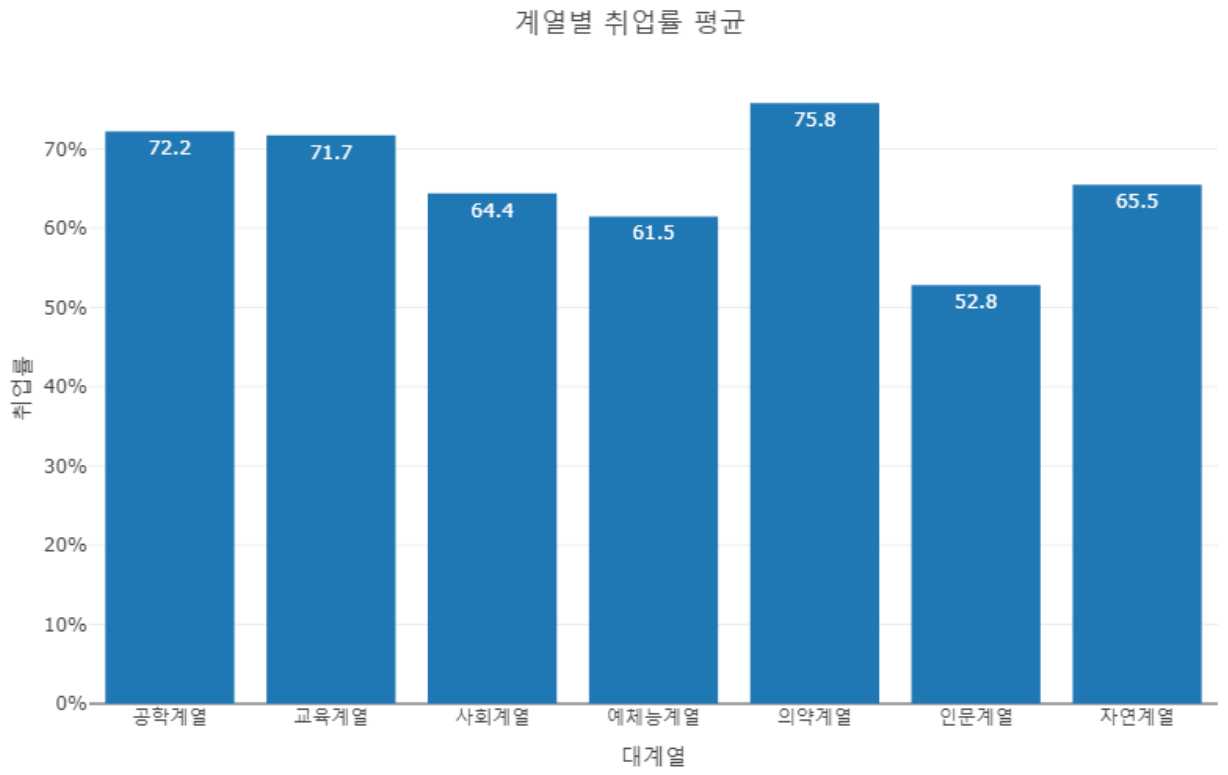
R 에서 bar 트레이스를 만들기 위해서는 `add_trace(type = 'bar')`를 사용하거나 `add_bars()`를 사용한다.

```
df_취업률 |> group_by(대계열) |>
  summarise(취업률 = mean(취업률_계)) |>
  plot_ly() |>
  ## bar 트레이스 추가
  add_trace(type = 'bar', x = ~대계열, y = ~취업률,
            ## text 와 textposition 설정
            text = ~취업률, textposition = 'inside',
            ## textemplate 설정
            texttemplate = '%{y:.1f}') |>
  layout(title = '계열별 취업률 평균',
         ## 눈금 접미어 설정
         yaxis = list(ticksuffix = '%'),
         margin = margins_R)
```

- python

python 에서 bar 트레이스를 만들기 위해서는 `add_trace()`에 `plotly.graph_object.Bar()`를 사용하거나 `plotly.express.bar()`를 사용한다.

```
df_취업률_대계열평균 = df_취업률.groupby('대계열').agg(취업률 = ('취업률_계', 'mean
'))

fig = go.Figure()

## bar 트레이스 추가
fig.add_trace(go.Bar(
    x = df_취업률_대계열평균.index, y = df_취업률_대계열평균['취업률'],
    ## text 설정
    text = df_취업률_대계열평균['취업률'],
    ## textposition, textemplate 설정
    textposition = 'inside', texttemplate = '%{y:.1f}'))

fig.update_layout(title = dict(text = '계열별 취업률 평균', x = 0.5),
                  yaxis = dict(ticksuffix = '%'))

fig.show()
```



실행결과 IV-1. R 의 수직 막대 그래프

1.2. 정렬 막대 그래프

비교를 위해 사용하는 막대 그래프는 특정한 이산형 변수에 의한 비교 데이터를 정렬한 후에 가장 높은 순서부터 혹은 가장 낮은 순서부터 그릴 때 가장 효과가 큰 시각화가 된다. 앞의 취업률 막대그래프와 같이 비교에 사용되는 변수의 변량이 많지 않은 경우는 따로 정렬을 하지 않아도 데이터의 크기를 비교하기 쉽지만 사용되는 변수의 변량이 많은 경우는 데이터 크기를 비교하기 어렵다. 이럴 경우에는 막대의 순서를 데이터의 크기별로 정렬해 주는 것이 데이터를 비교하는데 효과적이다. 이렇게 막대의 순서를 정렬해주기 위해서는 데이터 자체의 순서를 정렬한 후 막대 그래프를 그리는 방법도 있고 plotly 에서 제공하는 막대 정렬 속성을 사용하는 방법이 있다.

`plolty` 의 bar 트레이스에서 막대의 순서를 정렬하기 위해서는 'categoryorder'를 사용할 수 있다. 'categoryorder'에는 여러가지 속성값이 있는데 "total ascending"과 "total descending"는 전체 데이터의 오름차순과 내림차순으로 정렬, "category ascending"과 "category descending"은 변량의 이름으로 정렬, "array"는 사용자가 원하는 정렬 순서를 'categoryarray' 속성을 사용해서 지정해줄 수 있다.

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 막대 그래프를 그리는 R 과 python 코드는 다음과 같다. 여기서 인구수가 너무 적은 국가는 백신접종률에 의미가 떨어지기 때문에 인구수가 천만명 이상의 국가를 대상으로 하였고, 'texttemplate'를 사용하여 표시되는 데이터에 "%"를 붙여주었고 정렬을 위해 'categoryorder'를 "total desceding"으로 설정하여 정렬하였다.

- R

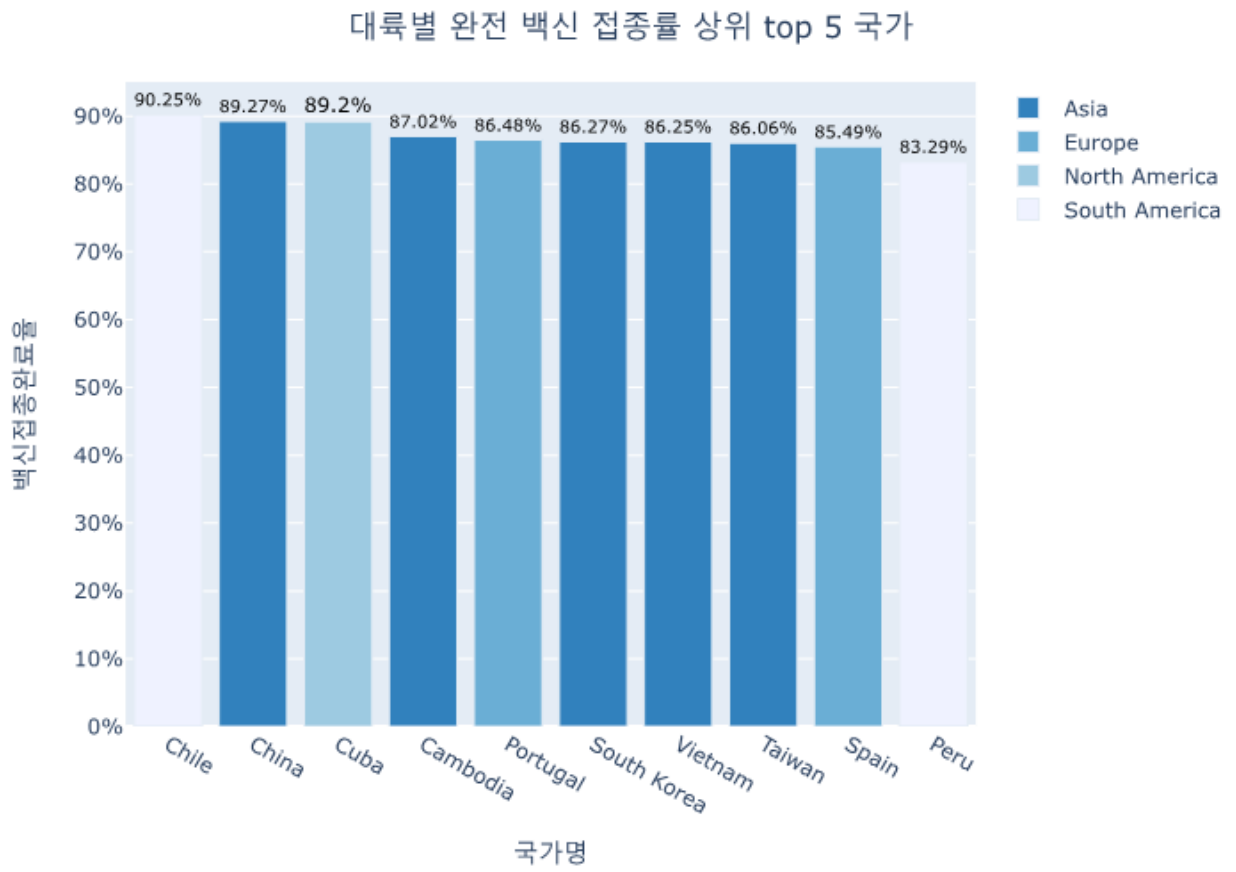
```r
## 인구수가 백만명 이상의 국가중에 인구백명당접종완료율 top 10 필터링
vaccine_top10 <- df_covid19_stat |>
  filter(인구수 > 10000000) |>
  top_n(10, 인구백명당백신접종완료율)

vaccine_top10 |>
  plot_ly() |>
  add_trace(type = 'bar',
            x = ~location, y = ~인구백명당백신접종완료율,
            color = ~continent, text = ~인구백명당백신접종완료율,
            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_R)
```

- python

python 에서는 앞서 scatter 트레이스에서와 마찬가지로 색상을 지정하는 'color' 속성에 배열이나 리스트를 설정할 수 없다. 따라서 `for` 루프를 사용하여 각각의 대륙별로 색상을 지정한 bar 트레이스를 추가하는 형태로 코딩해야 한다.

```python
## 인구수가 백만명 이상의 국가중에 인구백명당접종완료율 top 10 필터링
vaccine_top10 = df_covid19_stat.loc[df_covid19_stat['인구수'] > 10000000].sort_va
lues(by=['인구백명당백신접종완료율'], ascending=False).head(10).reset_index()

fig = go.Figure()

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

```
        name = continent,
        text = group['인구백명당백신접종완료율'], textposition = 'outside', texttem
plate = '%{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_P)

fig.show()
```



실행결과 IV-2. python 의 정렬 막대 그래프

1.3. 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 대륙별로 그룹화된 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. 하지만 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 앞서 설명한 바와 같이 표현되는 변량의 수가 많은 막대 그래프는 수평 막대 그래프로 표현하는 것이 효과적이다.

`plolty` 에서 수평 막대 그래프를 그리기 위해서는 bar 트레이스 속성의 'orientation'을 "v"로 설정해야 한다. 또 이산형 변수를 Y 축에 매핑시키고 연속형 변수를 X 축에 매핑시켜 일반적인 막대 그래프의 매핑과 반대로 설정해야 한다.

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' 속성을 설정하였다. 또 X 축에 여유를 주기 위해 'layout'의 'xaxis'를 0 부터 105 까지로 설정하였고, Y 축에 눈금 라벨을 설정하기 위해 'tickvals'와 'ticktext'를 설정하였다.

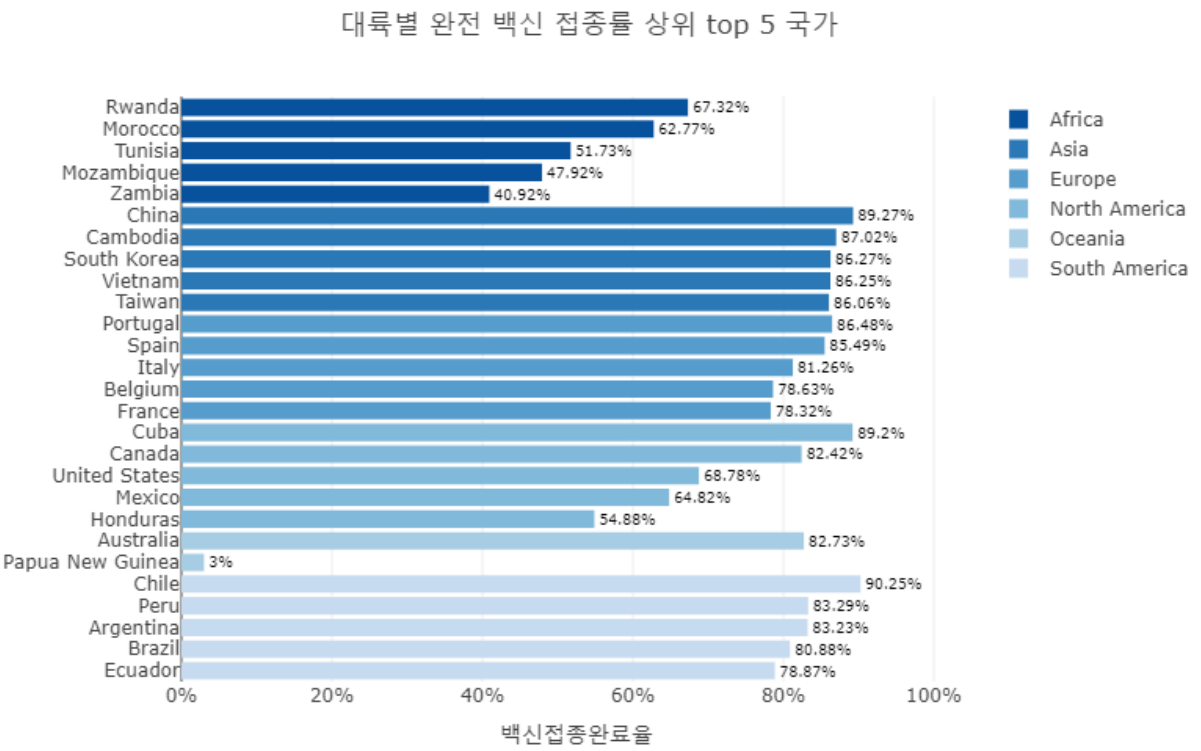
- R

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

vaccine_top5_by_continent |>
  plot_ly() |>
  add_trace(type = 'bar',
```

```
            y = ~seq, x = ~인구백명당백신접종완료율,
            color = ~continent,
            text = ~인구백명당백신접종완료율, textposition = 'outside',
            texttemplate = '%{text}%',
            textfont = list(color = 'black'),
            orientation = 'v'
            ) |>
  layout(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_R, size = list(height = 900)
)
```

- python

```
## 대륙별 백신접종완료율 top 5 필터링
vaccine_top5_by_continent = df_covid19_stat.loc[df_covid19_stat['인구수'] > 10000
000].sort_values(by=['continent', '인구백명당백신접종완료율'], ascending=False).gro
upby('continent').head(5).reset_index()

fig = go.Figure()

for continent, group in vaccine_top5_by_continent.groupby('continent'):
    fig.add_trace(go.Bar(
        y = group['location'], x = group['인구백명당백신접종완료율'],
        name = continent,
        text = group['인구백명당백신접종완료율'], textposition = 'outside', texttem
plate = '%{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 = margins_P, height = 800)

fig.show()
```



실행결과 IV-3. R 의 정렬 막대 그래프

1.4. 그룹 막대 그래프

앞에서 그린 수평 막대 그래프는 대륙별로 그룹화되어 있긴 하지만 전체 데이터를 하나의 데이터프레임으로 만들고 표시될 순서를 열로 만들어 마치 그룹 막대 그래프처럼 보이게 만든 단순 수평 막대 그래프이다. 하지만 그룹 막대 그래프는 일반적으로 긴 형태의 데이터보다는 넓은 형태의 데이터에 적합한 시각화이다. 각각의 열로 표현된 bar 트레이스를 여러개 추가함으로써 그룹 막대 그래프를 그린다.

다음은 대학의 계열에 따른 과정별 그룹 막대 그래프를 그리는 R 과 python 코드이다. 우선 그룹 막대 그래프를 그리기 위해서 먼저 긴 형태의 취업률 데이터프레임 중 시각화에 필요한 데이터만으로 그룹화하고 그룹별로 평균값을 산출한다.이 데이터를 다시 넓은 형태로 만들어서 '전문대학과정', '대학과정', '대학원과정'의 세 개의 열을 가지는 넓은 데이터프레임을 만든다. 이 넓은 데이터프레임의 각 열을 Y 축으로 매핑하는 세 개의 bar 트레이스를 하나의 plotly 객체에 추가해주고 'layout' 속성의 'barmode'를 "group"으로 설정해서 그룹 막대 그래프를 만든다. 추가적으로 'bargroupgap'을 0.2 로 설정해주었는데 막대 그룹내의 막대간의

간격을 설정해주었다. 만약 'bargroupgap'이 설정되지 않는다면 막대 그룹 내의 막대들은 서로 붙어서 표현된다.

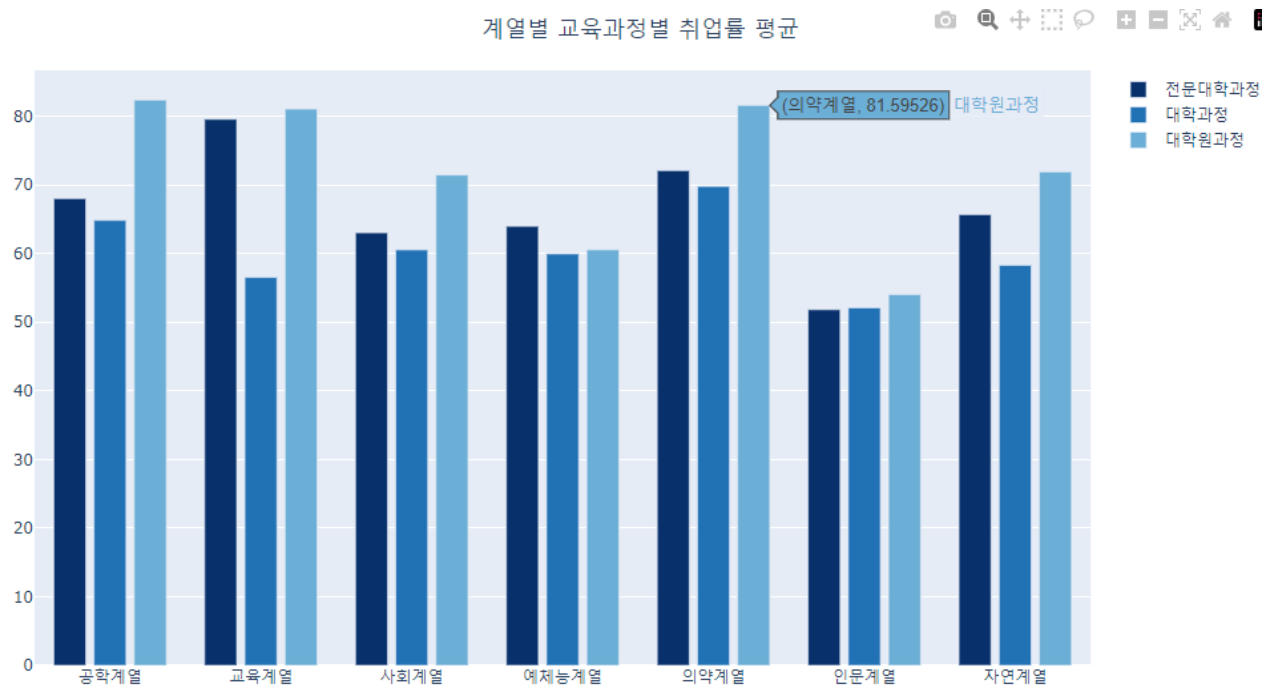
- R

```r
## 계열별 취업률을 넓은 데이터 형태로 전처리
취업률_by_계열 <- df_취업률 |>
  group_by(과정구분, 대계열) |>
  summarise(취업률 = mean(취업률_계)) |>
  pivot_wider(names_from = 과정구분, values_from = 취업률)

취업률_by_계열 |> plot_ly() |>
  ## 과정별로 bar 트레이스 추가
  add_trace(type = 'bar', x = ~대계열, y = ~ 전문대학과정, name = '전문대학과정') |
>
  add_trace(type = 'bar', x = ~대계열, y = ~ 대학과정, name = '대학과정') |>
  add_trace(type = 'bar', x = ~대계열, y = ~대학원과정, name = '대학원과정') |>
  ## barmode, bargroupgap 설정
  layout(barmode = 'group', bargroupgap = 0.2,
         title = '계열별 교육과정별 취업률 평균',
         margin = margins_R)
```

- python

```python
## 계열별 취업률을 넓은 데이터 형태로 전처리
취업률_by_계열 = df_취업률.groupby(['과정구분', '대계열']).agg(취업률 = ('취업률_계
', 'mean')).reset_index().pivot(index = '대계열', columns='과정구분', values='취업
률')
취업률_by_계열

fig = go.Figure()

## 과정별로 bar 트레이스 추가
fig.add_trace(go.Bar(
    x = 취업률_by_계열.index,
    y = 취업률_by_계열['전문대학과정'],
    name = '전문대학과정'))

fig.add_trace(go.Bar(
    x = 취업률_by_계열.index,
```

```
    y = 취업률_by_계열['대학과정'],
    name = '대학과정'))

fig.add_trace(go.Bar(
    x = 취업률_by_계열.index,
    y = 취업률_by_계열['대학원과정'],
    name = '대학원과정'))

## barmode, bargroupgap 설정
fig.update_layout(barmode = 'group', bargroupgap = 0.2,
                  title = '계열별 교육과정별 취업률 평균',
                  title_x = 0.5, margin = margins_P)

fig.show()
```



실행결과 IV-4. python 의 정렬 막대 그래프

1.5. 이중 축 막대 그래프

시각화를 하다보면 하나의 그래프에 여러 개의 데이터를 표현해야하는 경우가 많다. 서로 다른 데이터를 표현하는데 서로 동일한 트레이스를 사용할 수도 있고 다른 트레이스를 병합하여 혼용할 수도 있다. 우리가 흔하게 만나는 경우가 막대 그래프와 선그래프를 혼용하는 경우이다. 이렇게 하나 이상의 데이터를 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 에서 기본적으로 설정되는 X 축은 'xaxis', Y 축은 'yaxis'로 'layout' 속성에서 설정이 가능하다. 여기에 추가적으로 설정하는 축은 뒤에 숫자를 붙여 설정할 수 있다. X 축을 추가적으로 설정하려면 'xaxis2', Y 축을 추가적으로 설정하려면 'yaxis2'로 설정하고 'layout'에서 세부 속성을 설정한다. 이렇게 설정된 추가 축은 해당 축을 사용하는 trace 에 'xaxis'나 'yaxis' 속성에 'layout'에 설정한 추가 축의 이름을 설정하여 서로 참조하도록 매칭한다. `plotly` 에서는 트레이스에서 'xaxis'와 'yaxis'에 "x2", "y2"로 설정하면 'layout'의 'xaxis2', 'yaxis2'에 매칭되도록 설계되어 있다. 추가적인 축이 표현되는데에는 'overlaying'과 'side'의 두 가지 속성의 설정이 필요하다. 'overlaying'은 기존 디폴트 축과 겹쳐져서 두 번째 축이 표시되고 'side'는 표현되는 곳을 지정하도록 하는 방법이다.

이중 축의 사례를 살펴보기 위해 앞에서 그렸던 완전 백신 접종률 상위 10 국가 막대 그래프에 인구 10 만명당 사망자수를 점으로 표시하는 scatter 트레이스를 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 설정하다. 두번째 Y 축의 설정을 위해 'layout' 속성의 'y2'에 해당하는 축에 대한 'yaxis2' 속성을 설정한다. 그리고 'overlaying' 속성에 "y"를 설정하여 Y 축에 오버레이되는 축이라는 점을 설정하였고 'side'를 "right"로 설정하여 'yaxis2'가 오른쪽에 표시되도록 설정하였다. 두번째 Y 축의 설정으로 인해 범례와의 간격이 좁아졌다. 그래서 범례를 조금 오른쪽으로 옮겨주었다.

- R

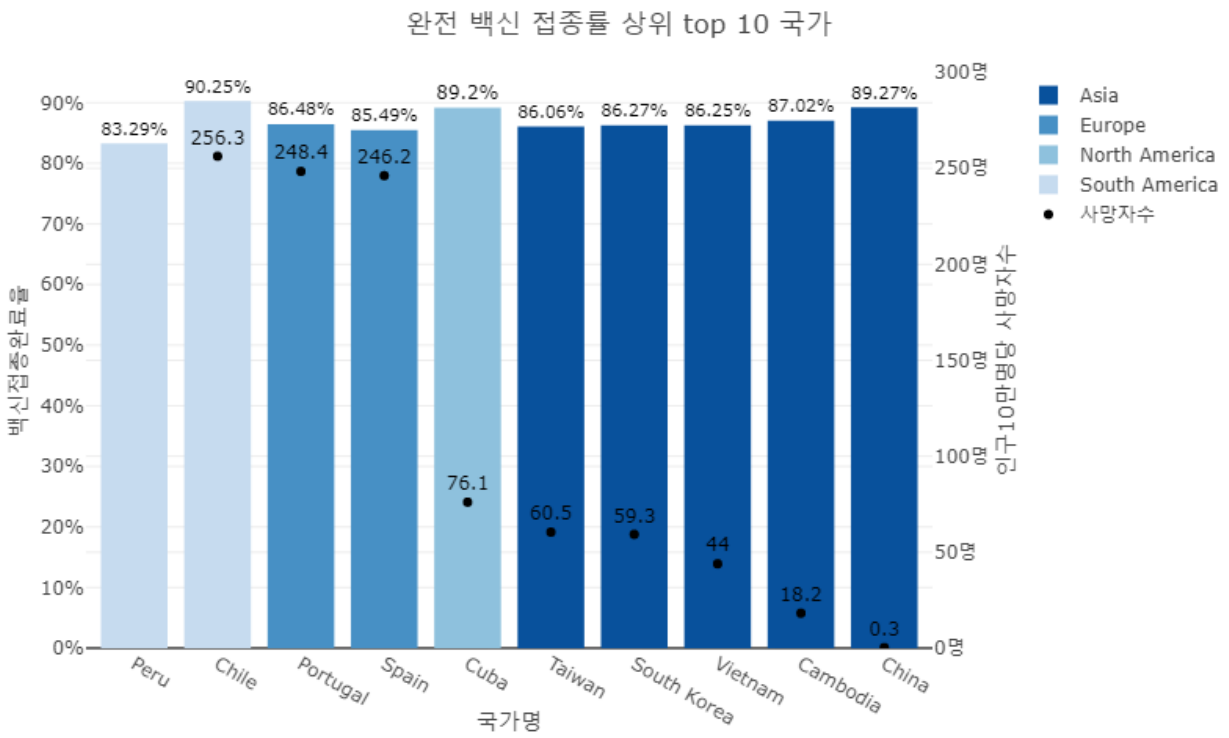
```r
vaccine_top10 |>
  plot_ly() |>
  ## bar 트레이스 추가
  add_trace(type = 'bar',
            x = ~location, y = ~인구백명당백신접종완료율,
            color = ~continent, text = ~인구백명당백신접종완료율,
            textposition = 'outside', texttemplate = '%{text}%',
            textfont = list(color = 'black')) |>
  ## markers+text 모드인 scatter 트레이스 생성
  add_trace(type = 'scatter', mode = 'markers+text',
            ## yaxis 를 "y2"로 설정
            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 = '%'),
         ## y2 축의 설정
         yaxis2 = list(title = '인구 10 만명당 사망자수',
                       side = "right", overlaying = "y",
                       range = c(0, 300), ticksuffix = '명'),
         margin = margins_R,  legend = list(x = 1.1))
```

- python

```
fig = go.Figure()

## bar 트레이스 추가, 색상 설정을 위해 continent 로 그룹화하고 for 루프 실행
for 대륙, group in vaccine_top10.groupby('continent'):
    fig.add_trace(go.Bar(
        x = group['location'], y = group['인구백명당백신접종완료율'],
        name = 대륙,
        text = group['인구백명당백신접종완료율'], textposition = 'outside', texttem
plate = '%{text}%',
        textfont = dict(color = 'black')))

## markers+text 모드인 scatter 트레이스 생성
fig.add_trace(go.Scatter(mode = 'markers+text',
            ## yaxis 를 "y2"로 설정
            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 = '%'),
```

```
        ## y2 축의 설정
        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()
```



실행결과IV-5. R 의 이중 축 막대 그래프

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

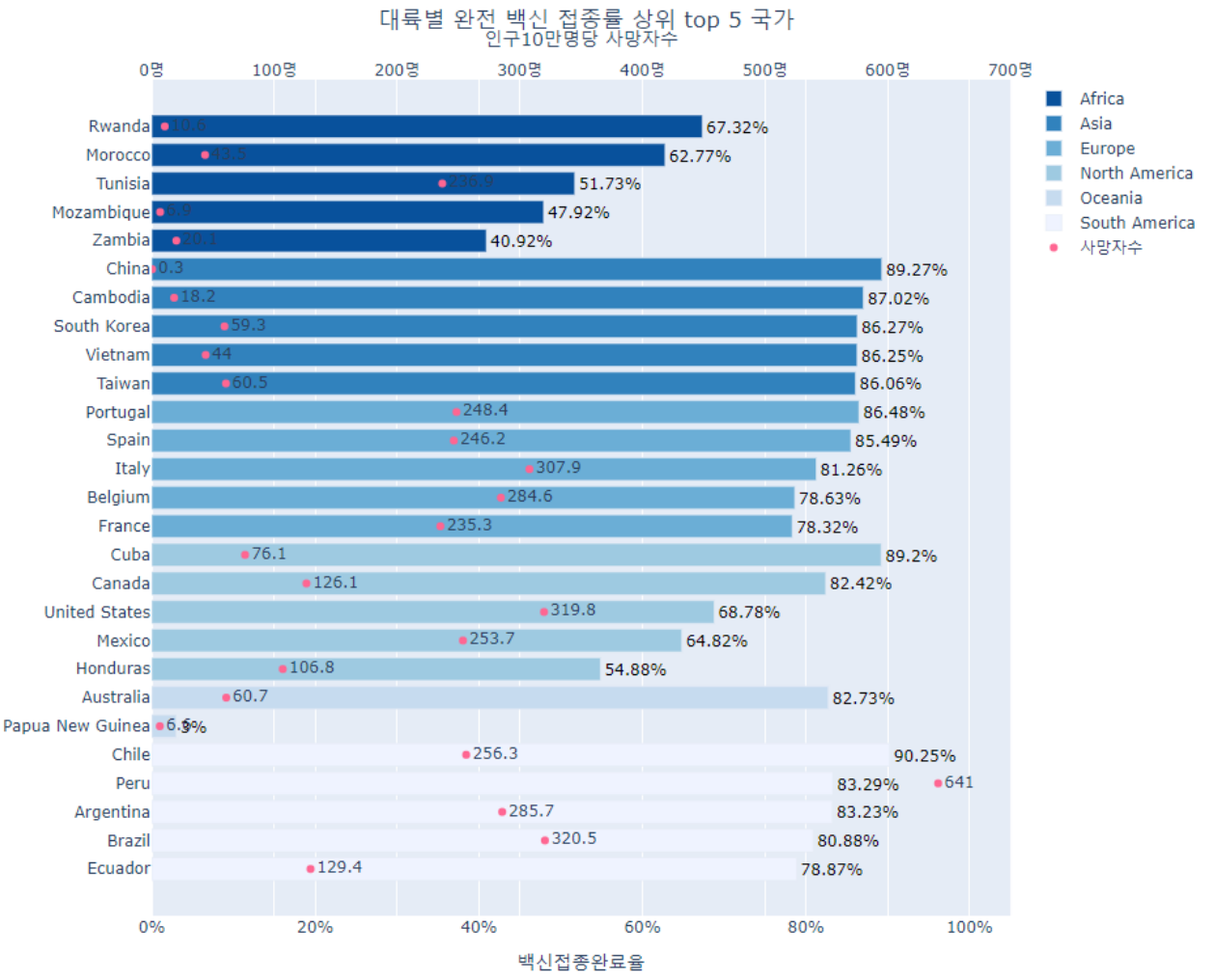
```
vaccine_top5_by_continent |>
  plot_ly() |>
```

```
  ## bar 트레이스 추가
  add_trace(type = 'bar',
            y = ~seq, x = ~인구백명당백신접종완료율, color = ~continent,
            text = ~인구백명당백신접종완료율, textposition = 'outside',
            texttemplate = '%{text}%',
            textfont = list(color = 'black'), orientation = 'v'
            ) |>
  ## markers+text 모드인 scatter 트레이스 생성
  add_trace(type = 'scatter', mode = 'markers+text',
            ## xaxis 를 "x2"로 설정
            name = '사망자수', xaxis = "x2",
            y = ~seq, x = ~십만명당사망자수, color = I('black'),
            text = ~round(십만명당사망자수, 1),
            textposition = 'middle 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 축의 설정
         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

```
fig = go.Figure()

## bar 트레이스 추가, 색상 설정을 위해 continent 로 그룹화하고 for 루프 실행
for continent, group in vaccine_top5_by_continent.groupby('continent'):
    fig.add_trace(go.Bar(
        y = group['location'], x = group['인구백명당백신접종완료율'],
        name = continent,
        text = group['인구백명당백신접종완료율'], textposition = 'outside', texttem
plate = '%{text}%',
        textfont = dict(color = 'black'), orientation = 'h'
    ))

## markers+text 모드인 scatter 트레이스 생성
```

```
fig.add_trace(go.Scatter(mode = 'markers+text',
    ## xaxis 를 "x2"로 설정
    xaxis = "x2",
    name = '사망자수',
    x = vaccine_top5_by_continent['십만명당사망자수'],
    y = vaccine_top5_by_continent['location'], text = round(vaccine_top5_by_cont
inent['십만명당사망자수'], 1),
    textposition = 'middle right'))

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)),
         ## xaxis2 축의 설정
         xaxis2 = dict(title = dict(text = '인구 10 만명당 사망자수', standoff =
1),
                       side = "top", overlaying = "x",
                       range = (0, 700), ticksuffix = '명'),
         yaxis = dict(title = '', autorange = 'reversed'),
         margin = dict(r = 100, t = 80), height = 800)

fig.show()
```



실행결과 IV-6. python 의 이중 축 막대 그래프 2

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 세로로 긴 형태가 적절치 않은 경우가 있다. 따라서 막대의 너비를 최소화한 직선의 형태로 표현한 막대 그래프를 롤리팝 그래프라고 한다.

롤리팝 그래프는 막대사탕의 상품명에서 유래된 그래프이다. 막대 그래프와 유사하지만 그 표현을 막대가 아닌 막대 사탕처럼 점과 직선을 사용하여 표현한다는데에서 유래했다. 점으로 표시된 데이터 점으로부터 축까지를 선으로 이어 데이터를 표현하는 방식의 그래프이다.

plotly 에서는 이 롤리팝 그래프를 트레이스로 제공하지는 않는다. 따라서 롤리팝 그래프를 그리기 위해서는 scatter 트레이스의 'markers'와 scatter 트레이스의 선이 아닌 도형으로서의 선을 축까지 그려줌으로서 만들 수 있다. 점을 그리는 scatter 트레이스는 R 과 python 이 유사한 방법을 사용하지만 선 도형을 넣는 것은 다소 차이가 있다. 다음은 앞서 아시아 지역 국가들의 십만명당 사망자수가 표현된 대륙별 백신접종율을 롤리팝 그래프로 그린 R 과 python 코드이다.

- R

R 에서 도형으로서의 선은 `add_segments()`를 사용하거나 `add_annotations()`의 type 속성을 "line"으로 설정하여 그릴 수 있다. 롤리팝 그래프를 그리기 위해서 먼저 데이터를 전처리하였다. 아시아 지역 데이터로 필터링하였고 인구수도 5 백만명 이상의 국가로 한정하였다. 이 후 plotly 객체를 초기화하였는데 모든 트레이스에서 공통적으로 사용하는 X 축 매핑을 `plot_ly()`에 넣어주어 추가되는 트레이스에 중복적으로 설정되는 현상을 방지하였다. 다음으로 `add_segments()`로 수직선을 만들어주는데 'xend'는 X 축에 매핑되는 값고 동일하게 하였고 Y 축에 매핑되는 값은 선의 시작점인 'y'를 국가별 인구 백명당접종완료율로, 선의 끝점인 'yend'는 0 로 Y 축까지 선이 그어지도록 설정하였다. 여기에 scatter 트레이스로 접종완료율과 십만명당 사망자수를 점으로 표시하였다. 마지막으로 'layout'을 설정하는데 'title', 'yaxis', 'xaxis', 'yaxis2', 'margin', 'legend' 속성을 설정하였다.

```
## 롤리팝 그래프를 위한 데이터 전처리
df_lolipop <- df_covid19_stat |>
  filter(인구수 > 5000000, continent == 'Asia') |>
  arrange(desc(인구백명당백신접종완료율))

df_lolipop |>
  plot_ly(x = ~reorder(location, desc(인구백명당백신접종완료율))) |>
  ## 세그먼트 레이어 추가
  add_segments(xend = ~reorder(location, desc(인구백명당백신접종완료율)),
               y = ~인구백명당백신접종완료율,
               yend = 0, color = I('gray'),
               showlegend = FALSE) |>
  ## markers 모드인 scatter 트레이스 추가
  add_trace(type = 'scatter', mode = 'markers', name = '접종완료율',
            y = ~인구백명당백신접종완료율, color = I('darkblue')) |>
  add_trace(type = 'scatter', mode = 'markers',
            symbol = I('circle-open'),
```

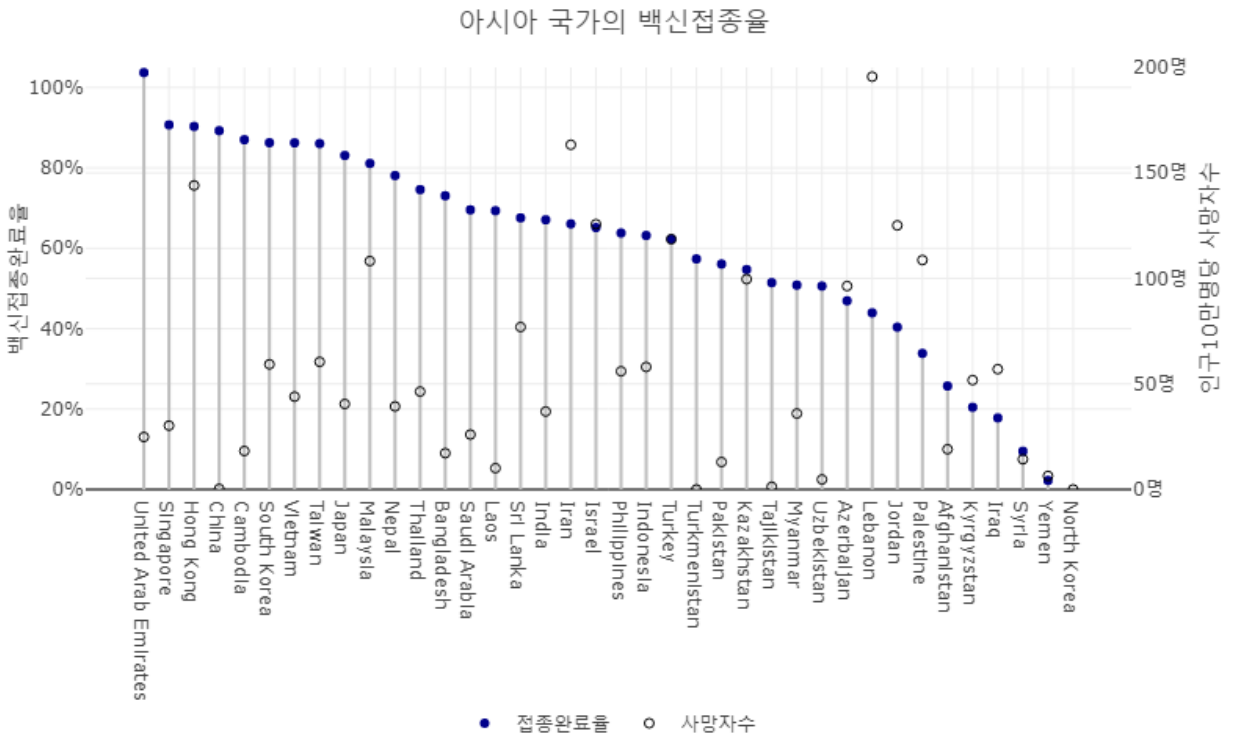
```
            name = '사망자수', yaxis = "y2",
            y = ~십만명당사망자수, color = I('black'),
            text = ~round(십만명당사망자수, 1),
            textposition = 'right')|>
  layout(barmode = 'group',
         title = list(text = '아시아 국가의 백신접종율',
                      y = 0.97, yref = 'container'),
         yaxis = list(title = '백신접종완료율', range = c(0, 105),
                      ticksuffix = '%'),
         xaxis = list(title = ''),
         ## 두 번째 Y 축의 설정
         yaxis2 = list(title = list(text = '인구 10 만명당 사망자수',
                                    standoff = 10),
                       side = "right", overlaying = "y",
                       range = c(0, 200), ticksuffix = '명'),
         margin = margins_R,
         legend = list(orientation = 'h', y = -0.5, x = 0.5,
                       yref = 'container', xanchor = 'center'),
         showlegend = T)
```

- python

python 에서 선 도형을 그리기 위해서는 `add_shape()`의 'type'을 'line'으로 설정하고 X 축의 시작점을 'x0', 끝점을 'x1', Y 축의 시작점을 'y0', 끝점을 'y1'을 설정하여 그렸고 선의 속성을 설정하기 위해 'line' 속성의 하위 속성을 설정하였다. 다만 'x0', 'x1', 'y0', 'y1'는 벡터형 변수를 설정할 수 없기 때문에 `for` 루프를 사용하여 각각의 국가별로 선을 그려주었다. 이후 접종율과 사망자수를 표시하는 scatter 트레이스를 추가해주었고 'layout' 속성으로 'title', 'xaxis', 'yaxis', 'yaxis2', 'margin', 'legend', 'showlegend'를 설정해주었다.

```
## 롤리팝 그래프를 위한 데이터 전처리
df_lolipop = df_covid19_stat.reset_index()

df_lolipop = df_lolipop.loc[(df_lolipop['인구수'] > 5000000) & (df_lolipop['conti
nent'] == 'Asia') & (df_lolipop['인구백명당백신접종완료율'].isna() == False)].sort_
values(by = '인구백명당백신접종완료율', ascending=False)

fig = go.Figure()

## 라인 세그컨트 레이어를 추가
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['인구백명당백신접종완
료율'],
        line=dict(color='RoyalBlue',width=3))

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

fig.add_trace(go.Scatter(
    mode = 'markers', name = '사망자수',
    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 백신접종율', x = 0.5),
         xaxis = dict(title = ''),
         yaxis = dict(title = '백신접종완료율', range = (0, 105),
                      ticksuffix = '%'),
         ## 두 번째 Y 축의 설정
         yaxis2 = dict(title = dict(text = '인구 10 만명당 사망자수',
                                    standoff = 10),
                       side = "right", overlaying = "y",
                       range = (0, 200), ticksuffix = '명'),
         margin = margins_P, showlegend = True,
         legend = dict(orientation = 'h', y = -0.5, x = 0.5, xanchor = 'center
'))

fig.show()
```



실행결과 IV-7. R 의 롤리팝 그래프

3. 레이더 차트

레이다 차트(Radar Chart)는 특정 객체나 이벤트를 여러 평가 항목에 따라 평가를 한 결과를 시각화할 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. 원점을 중심으로 둥글게 펼쳐진 각각의 데이터 축에 표현되다보니 마치 레이더망과 같이 생겨 붙여진 이름이다. 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) 트레이스가 아닌 scatterpolar 트레이스라는 이름의 트레이스를 사용한다. scatterploar 트레이스는 단지 레이더 차트를 그리기 위해 사용되는 트레이스가 아니고 축을 X, Y 축이 아닌 각도(angular)와 반지름(radial)를 사용하는 극 좌표계(Ploar)를 사용하는 scatter 트레이스를 말한다. 따라서 'mode'를 'markers'나 'text'로 사용하는 scatterpolar 트레이스를 사용할 수도 있다.

scatterpolar 트레이스에서 중요하게 사용되는 속성이 '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 트레이스 설정된 색을 채우고 트레이스가 여러 개일 경우 트레이스가 겹치는 곳의 색은 양 트레이스의 색이 겹쳐 표현된다. "tonext"도 시작점과 끝점을 연결해 닫힌 모양으로 만들고 내부를 채우는데 완전히 겹쳐지는 트레이스의 색을 설정된 색을 사용한다는 것이 차이이다. "none"은 내부를 채우지 않는다.

다음은 scatterploar 트레이스에서 사용되는 주요 속성들이다.

속성			속성 설명	python 속성값(R 속성값)	R 속성값
mode			스캐터 trace의 그리기 모드를 결정. 선, 점, 문자 중 하나가 오거나 +를 사용하여 두가지 이상을 동시에 표현할 수 있음	플래그 문자열. "lines", "markers", "text" 를"+"로 조합. 또는 "none".	
r			좌표계의 반지름 설정	list, numpy array, or Pandas series of numbers, strings, or datetimes/dataframe column, list,	
r0			r의 대안으로 r좌표계의 선형 공간을 만드는 시작좌표 설정	수치나 좌표상의 변량값	
dr			r0가 설정될 때 r좌표계의 간격 설정	수치	
theta			각좌표의 설정	list, numpy array, or Pandas series of numbers, strings, or datetimes/dataframe column, list,	
theta0			theta의 대안으로 theta 좌표계의 선형 공간을 만드는 시각좌표 설정	수치나 좌표상의 변량값	
dtheta			theta0가 설정될때 theta 좌표계의 간격 설정	수치	
marker	color		마커의 색상 설정. 특정한 색상이나 colorscale의 최대값과 최소값(또는 marker.cmax와 marker.cmin)에 상대적으로 매핑되는 수치 배열이 설정될 수 있다.	색상 또는 색상 배열	
	colorscale		색 스케일의 설정. marker.color가 수치형 배열일 경우에만 효과가 있음. 'colorscale'은 rgb, rgba, hex, hsl, hsv, 문자형 색 이름의 배열이여야 함. 이러한 'colorscale' 배열 대신 팔레트 이름을 쓸 수 있음 : Blackbody,Bluered,Blues,Cividis,Earth,Electric,Greens,Greys,Hot,Jet,Picnic,Portland,Rainbow,RdBu,Reds,Viridis,YlGnBu,YlOrRd	컬러 스케일	
	line	color	마커 선 색을 설정. 특정한 색상이나 colorscale의 최대값과 최소값(또는 marker.cmax와 marker.cmin)에 상대적으로 매핑되는 수치 배열이 설정될 수 있다.	색상 또는 색상 배열	
		colorscale	마커 선의 색 스케일의 설정. marker.line.color가 수치형 배열일 경우에만 효과가 있음. 'colorscale'은 rgb, rgba, hex, hsl, hsv, 문자형 색 이름의 배열이여야 함. 이러한 'colorscale' 배열 대신 팔레트 이름을 쓸 수 있음 : Blackbody,Bluered,Blues,Cividis,Earth,Electric,Greens,Greys,Hot,Jet,Picnic,Portland,Rainbow,RdBu,Reds,Viridis,YlGnBu,YlOrRd	컬러 스케일	
		reversescale	True일 경우, 마커 선 색상 매핑을 역순으로 바꾼다. Marker.line.color가 수치형 배열일 경우에만 효과를 낸다.	논리값	
		width	px 단위의 선 두께 설정	0과 같거나 큰 수치나 수치 배열	
	opacity		마커의 투명도 설정	0과 같거나 큰 수치나 수치 배열	
	reversescale		True일 경우, 마커 색상 매핑을 역순으로 바꾼다. Marker.line.color가 수치형 배열일 경우에만 효과를 낸다.	논리값	
	size		px 단위의 마커 크기 설정	0과 같거나 큰 수치나 수치 배열	
	symbol		마커 형태(symbol) 타입을 설정. 기본 도형의 번호에 100을 더하면 '-open'형 심볼 이름이 되고, 200을 더하면 '-dot'형 심볼 이름이 되고, 300을 더하면 '-open-dot'으로 심볼이름이 설정됨	열거형 타입 또는 열거형 타입 배열	
line	backoff				
	color		선의 색을 설정	색상	
	dash		선의 타입을 설정. 선 타입 이름을 설정("solid", "dot", "dash", "longdash", "dashdot", or "longdashdot")하거나 px단위의 점선 길이 리스트를 설정	문	
	shape		라인 모양을 결정함. 'spline'은 spline 보간법을 사용한 선을 그림. 다른 값은 단계 형태의 선 값("hv" | "vh" | "hvh" | "vhv")을 가짐	("linear" | "spline")	
	smoothing		shape가 spline일 경우 평활 정도를 설정	0부터 1.3사이의 수치	
	width		px 단위의 선 두께 설정	0과 같거나 큰 수치	
connectgaps			데이터간의 갭을 연결할지 여부를 결정	논리값	
fill			내부 색상의 채워지는 형태 설정	("none" | "toself" | "tonext")	
fillcolor			라인 색, 마커 색, 마커 라인 색의 내부 색상 설정	색상	

plotly 에서 scatterploar 트레이스를 사용할 때는 지금까지 사용했던 X, Y 축의 데카르트 좌표계가 아닌 극좌표계(polar)를 사용하기 때문에 'layout'을 설정할 때 축 설정과 관련한 속성들이 다소 차이가 있다. 극좌표계는 'layout' 속성의 'polar' 속성으로 설정한다. 'polar' 좌표도 'angularaxis'와 'radialaxis'의 두 개의 축으로 구성된다. 다음의 그림은 극좌표계의 주요 구성을 보이고 있다.

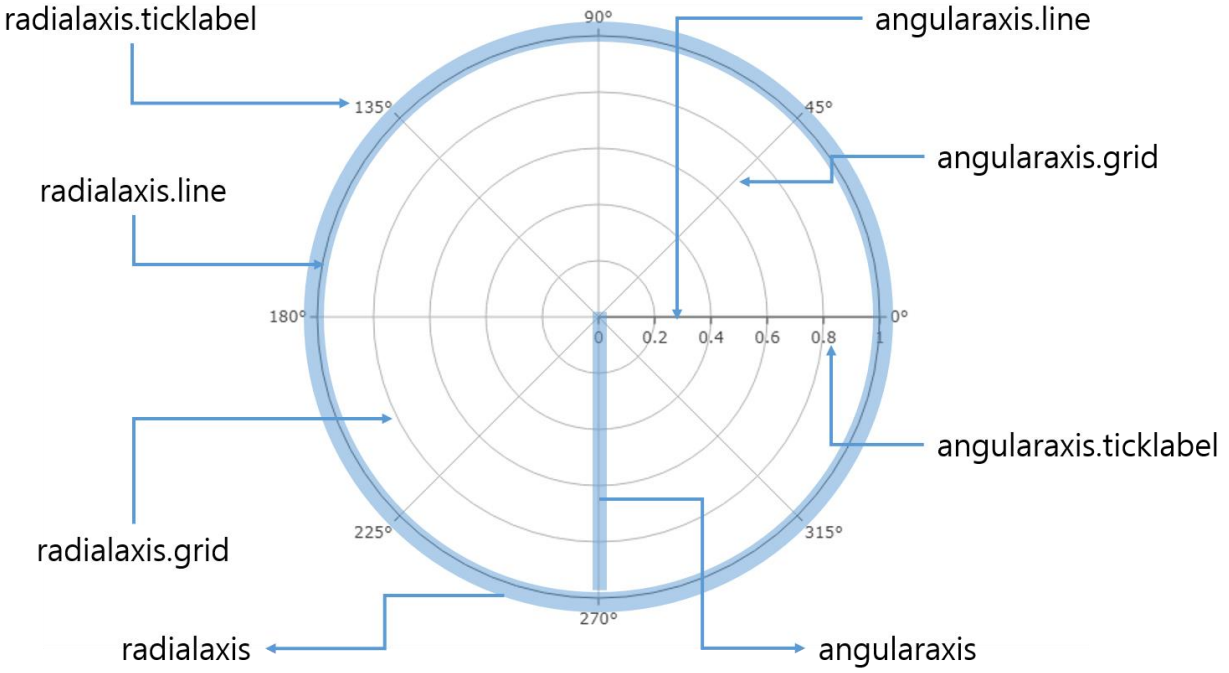
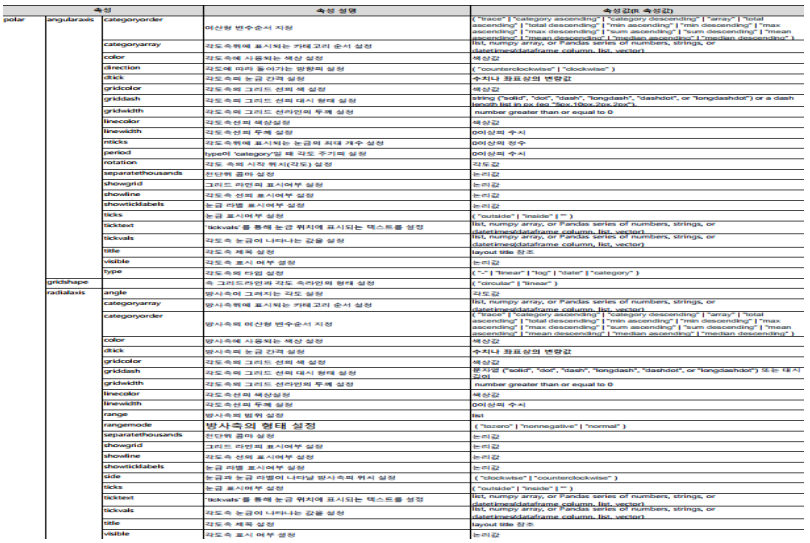


그림 IV-3. 레이더 차트의 구조

이들을 설정하는 'polar' 속성도 크게 'angularaxis'와 'radialaxis'의 두 가지 하위 속성으로 설정이 가능하다. 다음은 극좌표계에서 사용되는 좌표계 설정을 위한 주요 'layout'속성들이다.



다음은 대륙별 백신접종완료율을 레이더 차트로 시각화하는 R 과 python 코드이다.

- R

R 에서 scatterpolar 트레이스를 만들기 위해서는 `add_trace(type = 'scatterpolar', ...)`를 사용한다.

```
## 레이더 차트를 위한 데이터 전처리
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, 인구백명당백신접종완료율)

df_radar_veccine |>
  plot_ly() |>
  ## scatterpolar 트레이스 추가
  add_trace(type = 'scatterpolar',
            theta = ~location, r = ~인구백명당백신접종완료율, fill = 'toself') |>
  ## polar 속성 설정
  layout(polar = list(
    ##  angularaxis 속성 설정
    angularaxis = list(ticktext = c('아프리카', '아시아', '유럽', '북미', '오세아니아', '남미'),
                       tickvals = c('Africa', 'Asia', 'Europe', 'North America', 'Oceania', 'South America'),
                       linewidth = 2, linecolor = 'black', gridcolor = 'gray'),
    ##  radialaxis 속성 설정
    radialaxis = list(linewidth = 2, linecolor = 'dodgerblue', gridcolor = 'skyblue',
```

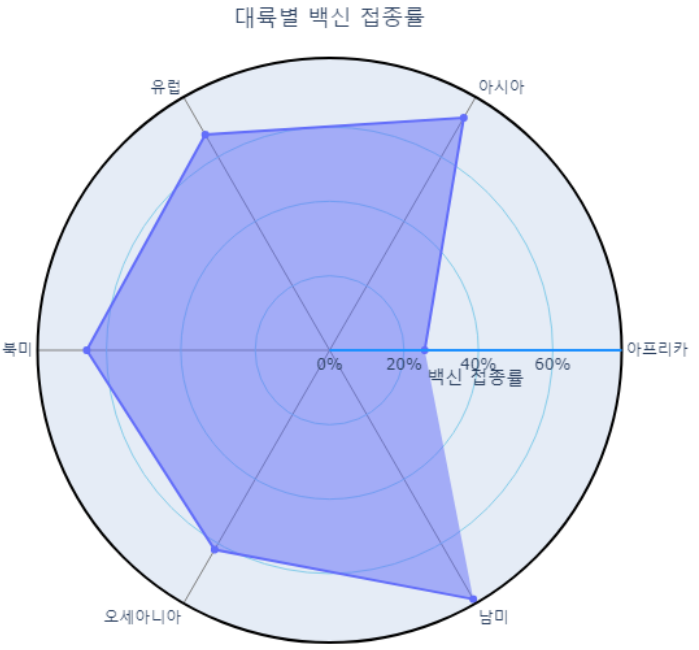
```
                    nticks = 5, ticksuffix = '%', title = '백신 접종률')),
         title = list(text = '대륙별 백신 접종률', x = 0.5),
         margin = margins_R)
```

- python

python 에서 scatterpolar 트레이스를 만들기 위해서는 `add_trace()`에 `plotly.graph_objects.Scatterploar()`를 사용하거나 `plotly.express.scatter_ploar()`를 사용한다.

```
## 레이더 차트를 위한 데이터 전처리
df_radar_veccine = df_covid19.groupby(['iso_code', 'continent', 'location'], dro
pna=False).agg(
    인구수 = ('population', 'max'),
    전체사망자수 = ('new_deaths', 'sum'),
    백신접종자완료자수 = ('people_fully_vaccinated', 'max'),
    인구백명당백신접종완료율 = ('people_fully_vaccinated_per_hundred', 'max'),
    인구백명당부스터접종자수 = ('total_boosters_per_hundred', 'max')
).reset_index()
df_radar_veccine = df_radar_veccine[(df_radar_veccine['iso_code'].isin(['OWID_AS
I', 'OWID_EUR', 'OWID_OCE', 'OWID_NAM', 'OWID_SAM', 'OWID_AFR']))]

fig = go.Figure()

## scatterpolar 트레이스 추가
fig.add_trace(go.Scatterpolar(
    theta = df_radar_veccine['location'],
    r = df_radar_veccine['인구백명당백신접종완료율'],
    fill = 'toself'
))

## polar 속성 설정
fig.update_layout(polar =
    ##  angularaxis 속성 설정
    dict(angularaxis =
            dict(ticktext = ('아프리카', '아시아', '유럽', '북미', '오세아니아', '남
미'),
                 tickvals = ('Africa', 'Asia', 'Europe', 'North America', 'Oceani
a', 'South America'),
                  linewidth = 2, linecolor = 'black', gridcolor = 'gray'),
    ##  radialaxis 속성 설정
         radialaxis =
```

```
            dict(linewidth = 2, linecolor = 'dodgerblue', gridcolor = 'skyblue
',
                 nticks = 5, ticksuffix = '%', title = '백신 접종률')),
    title = dict(text = '대륙별 백신 접종률', x = 0.5),
    margin = margins_P)

fig.show()
```



실행결과 IV-8. python 의 레이더 차트

4. 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. 롤리팝 그래프와 유사하게 보이지만 롤리팝 그래프는 막대 그래프의 변형으로 시작점을 모두 기준선에 맞추어 시작하지만 덤벨 차트는 시작점이 일정하지 않기 때문에 보통 두 개의 데이터를 비교하고 어떤 변량에서 차이가 더 큰지를 비교할 때 많이 사용한다.

plotly 는 덤벨 그래프를 위한 트레이스를 제공하지 않기 때문에 롤리팝 그래프와 같이 scatter 트레이스의 “markers” 타입으로 각 변량에 따른 두 개의 데이터를 표시하고 이 사이를 선 도형으로 이어서 그린다.

다음은 대륙별 인구수가 백만 이상의 국가 중에 십만명당 사망자수가 가장 작은 나라와 가장 큰 나라의 차이를 덤벨 차트로 시각화하는 R 과 python 코드이다. 데이터 전처리 단계에서는 먼저 전체 데이터를 각 대륙별로 인구수가 천만명이 넘는 국가를 필터링하고 그룹화하여 최대값과 최소값을 구한 데이터프레임을 만들었다. 이 데이터를 설명한 바와 같이 최대값을 표시하는 scatter 트레이스와 최소값을 표시하는 scatter 트레이스를 만들었고 이 사이를 선으로 잇는 선 도형을 만들었다.

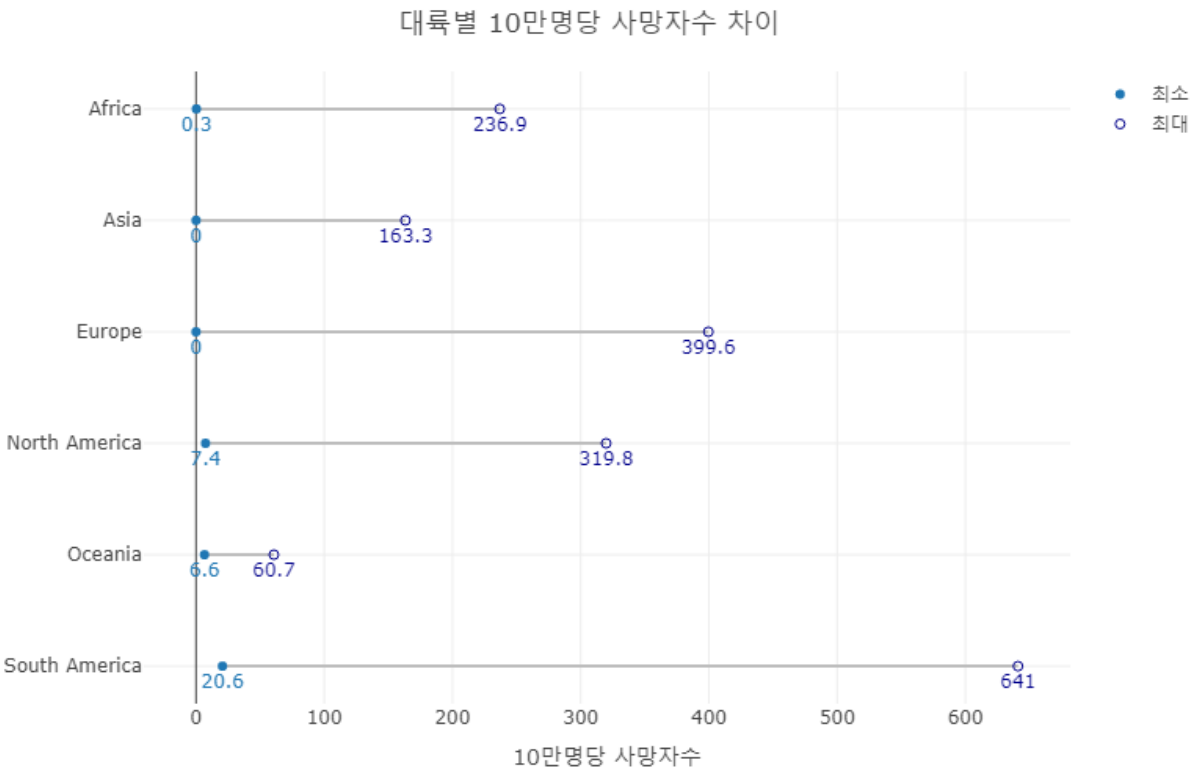
- R

```r
##  덤벨 차트를 위한 데이터 전처리
df_covid19_stat |>
  filter(!is.na(continent), 인구수 > 10000000) |>
  group_by(continent) |>
  summarise(min = min(십만명당사망자수), max = max(십만명당사망자수)) |>
  plot_ly() |>
  ## 덤벨 차트용 세그먼트 추가
  add_segments(
            x = ~min, xend = ~max, y = ~continent, yend = ~continent,
            showlegend = FALSE,
            color = I('gray')) |>
  ## 최소값 트레이스 추가
  add_trace(type = 'scatter', mode = 'markers+text',
            x = ~min, y = ~continent, name = '최소',
            text = ~round(min, 1), textposition = 'bottom center',
            color = I('#1f77b4')) |>
  ## 최대값 트레이스 추가
  add_trace(type = 'scatter', mode = 'markers+text',
            x = ~max, y = ~continent, name = '최대',
            text = ~round(max, 1), textposition = 'bottom center',
            color = I('darkblue'), symbol = I('circle-open')) |>
  layout(title = '대륙별 10 만명당 사망자수 차이',
         xaxis = list(title = '10 만명당 사망자수'),
         yaxis = list(title = '', autorange = 'reversed'),
         margin = margins_R)
```

- python

```python
##  덤벨 차트를 위한 데이터 전처리
df_dumbbell = df_covid19_stat.copy().reset_index()
df_dumbbell = df_dumbbell.loc[(df_dumbbell['continent'].isna() == False) & (df_d
umbbell['인구수'] > 10000000)].groupby('continent')['십만명당사망자수'].agg([('min
', 'min'), ('max', 'max')]).reset_index()

fig = go.Figure()
```

```
## 최소값 트레이스 추가
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')))

## 최대값 트레이스 추가
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', symbol = 'circle-open')))

## 덤벨 차트용 세그먼트 추가
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 = dict(text = '대륙별 10 만명당 사망자수 차이', x = 0.5),
         xaxis = dict(title = '10 만명당 사망자수'),
         yaxis = dict(title = '', autorange = 'reversed'),
         margin = margins_P)

fig.show()
```



실행결과 IV-9. R 의 덤벨 그래프

5. 비율 막대 그래프

앞 장에서 plotly 가 지원하는 bar 트레이스는 수평, 수직 막대 그래프, 그룹 막대 그래프, 스택 막대 그래프의 네 가지 유형을 살펴보았다. 하지만 막대 그래프를 사용하여 변량 간의 데이터를 비교하는 것이 아닌 각 변량을 구성하고 있는 구성 성분의 비율을 살펴보기 위해서는 스택 막대 그래프를 백분률로 표시하면 그 구성 비율이 쉽게 표현된다. 이를 비율 막대 그래프라고 하는데 막대 길이가 모두 100 으로 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.

비율 막대 그래프를 그리기 위해서는 먼저 표현하고자 하는 변수의 각 변량 데이터를 백분률로 전처리한 후 이 비율을 bar 트레이스의 스택 막대 그래프로 그려준다.

다음은 전세계 국가들의 소득별로 그룹화한 통계를 사용하여 '전체 확진자', '전체 사망자', '백신접종완료자수'를 비율 막대로 표시하는 R 과 python 코드이다. 먼저 데이터 중에 소득별 데이터만 필터링하고 행은 '전체확진자수', '전체 사망자수', '백신접종자완료자수'로,

열은 'High income', 'Low income', 'Lower middle income', 'Upper middle income'을 가지도록 변환하였다. 이 후 각 열의 합계값을 가지는 'sum' 열을 만들고 각각의 열을 'sum' 열로 나누어 행 전체의 비율로 각각의 열을 계산하였다. 이 비율로 비율 막대 그래프를 그리는데 수평 막대 그래프를 그리고 'barmode'를 "stack"으로 설정하여 누적형 스택 막대그래프로 설정하였다. 이렇게 그려진 비율 막대 그래프에 각 비율 텍스트를 넣어주는데 plotly 에서는 이 비율을 넣어주는 기능을 제공하지 않기 때문에 'annotation'을 추가하여 각각의 비율을 표기해 주었다. 마지막으로 'layout' 속성의 'tickformat'을 사용하여 X 축의 스케일을 백분률로 바꾸어 주었다.

- R

R 에서 비율 스택 막대 그래프를 그릴때는 설명한대로 막대의 구성으로 이루어지는 bar 트레이스들을 추가함으로써 그릴 수 있다. 여기서는 각각의 Y 변수에 따라 그 구성비(High income, Low income, Lower middle income, Upper middle income)에 해당하는 4 개의 bar 트레이스를 추가하였다. 여기에 'barmode'를 "stack"으로 설정해서 각각 스택 막대 그래프로 만들었고 각각의 비율 텍스트를 표기하기 위해 `add_annotations()`를 사용하였다. 텍스트의 X 위치를 결정할 때 각각의 변량의 중간의 위치하도록 하는데 두 번째, 세 번째, 네 번째 텍스트는 앞에 표시된 변량의 누적량만큼을 더해서 X 위치를 결정하였다. 여기에 'layout' 속성의 'xaxis', 'yaxis', 'legend', 'margin'을 설정하였다. 이 중 'legend'는 가로형태로 표현하기 위해 'orientation'을 "h"로 설정하였다.

```
##  비율 막대 그래프를 위한 데이터 전처리
df_covid19_stat |>
  filter(iso_code %in% c('OWID_HIC', 'OWID_LIC', 'OWID_LMC', 'OWID_UMC')) |>
  select(3, 5, 6, 7) |>
  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 i
ncome`)) |>
  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() |>
  ##  'High income'을 위한 bar 트레이스 추가
  add_trace(type = 'bar', x = ~`High income`, y = ~name,
            name = 'High income', orientation = 'h',
            marker = list(line = list(color = 'white', width = 2))) |>
  ##  'Upper middle income'을 위한 bar 트레이스 추가
  add_trace(type = 'bar', x = ~`Upper middle income`, y = ~name,
            name = 'Upper middle income', orientation = 'h',
```

```
            marker = list(line = list(color = 'white', width = 2))) |>
  ##  'Lower middle income'을 위한 bar 트레이스 추가
  add_trace(type = 'bar', x = ~`Lower middle income`, y = ~name,
            name = 'Lower middle income', orientation = 'h',
            marker = list(line = list(color = 'white', width = 2))) |>
  ##  'Low income'을 위한 bar 트레이스 추가
  add_trace(type = 'bar', x = ~`Low income`, y = ~name,
            name = 'Low income', orientation = 'h',
            marker = list(line = list(color = 'white', width = 2))) |>
  ##  'High income' 값 표시를 위한 주석 레이어 추가
  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) |>
  ##  'High income' 값 표시를 위한 주석 레이어 추가
  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) |>
  ##  'High income' 값 표시를 위한 주석 레이어 추가
  add_annotations(xref = 'x', yref = 'y',
                  x = ~`High income` + `Upper middle income` + `Lower middle inc
ome` / 2,
                  y = ~name,
                  text = ~paste(round(`Lower middle income`*100, 1), '%'),
                  font = list(color = 'white'),
                  showarrow = FALSE) |>
  add_annotations(xref = 'x', yref = 'y',
                  x = ~`High income` + `Upper middle income` + `Lower middle inc
ome` + `Low income` / 2, y = ~name,
                  text = ~paste(round(`Lower middle income`*100, 1), '%'),
                  font = list(color = 'white'),
                  showarrow = FALSE) |>
  layout(barmode = 'stack',
         title = '국가 소득급간별 코로나19 현황',
         xaxis = list(title = '', tickformat = '.0%'),
         yaxis = list(title = ''),
         legend = list(orientation = 'h', traceorder = 'normal'),
         margin = margins_R)
```

- python

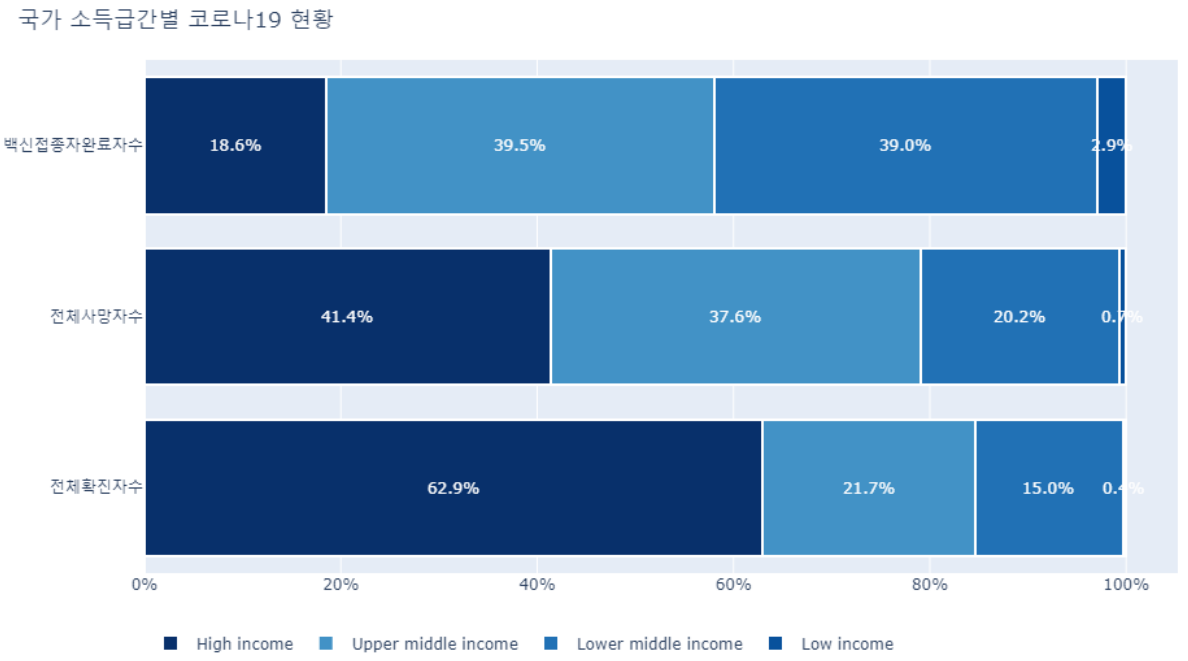
python 에서 비율 스택 막대 그래프를 그릴때도 설명한대로 막대의 구성으로 이루어지는 bar 트레이스들을 추가함으로써 그릴 수 있다. 여기서도 각각의 Y 변수에 따라 그 구성비(High income, Low income, Lower middle income, Upper middle income)에 해당하는 4 개의 bar

트레이스를 추가하였다. 여기에 'barmode'를 "stack"으로 설정하였고, 각각의 비율 텍스트를 표기하기 위해 `add_annotation()`를 사용하였다. `add_annotation()`의 속성중 표시될 텍스트를 지정하는 'text' 속성은 R 과는 달리 단일 문자열만 설정이 가능해 `for` 루프를 사용하여 각각의 행의 'annotation'을 추가하는 방법을 사용하였다. 텍스트의 X 위치를 결정할 때 각각의 변량의 중간의 위치하도록 하는데 두 번째, 세 번째, 네 번째 텍스트는 앞에 표시된 변량의 누적량만큼을 더해서 X 위치를 결정하였다. 여기에 'layout' 속성의 'xaxis', 'yaxis', 'legend', 'margin'을 설정하였다. 이 중 'legend'는 가로형태로 표현하기 위해 'orientation'을 "h"로 설정하였다.

```
##  비율 막대 그래프를 위한 데이터 전처리
df_covid19_stat_tmp = df_covid19.groupby(['iso_code', 'continent', 'location'],
dropna=False).agg(
    인구수 = ('population', 'max'),
    전체사망자수 = ('new_deaths', 'sum'),
    백신접종자완료자수 = ('people_fully_vaccinated', 'max'),
    인구백명당백신접종완료율 = ('people_fully_vaccinated_per_hundred', 'max'),
    인구백명당부스터접종자수 = ('total_boosters_per_hundred', 'max')
).reset_index()

df_covid19_stat_tmp = df_covid19_stat_tmp.loc[df_covid19_stat_tmp['iso_code'].is
in(['OWID_HIC', 'OWID_LIC', 'OWID_LMC', 'OWID_UMC']), ['location', '전체사망자수',
 '백신접종자완료자수', '인구백명당백신접종완료율']].transpose()
df_covid19_stat_tmp = df_covid19_stat_tmp.rename(columns=df_covid19_stat_tmp.ilo
c[0]).drop(df_covid19_stat_tmp.index[0])

df_covid19_stat_tmp['sum'] = df_covid19_stat_tmp.sum(axis = 1)
df_covid19_stat_tmp['High income'] = df_covid19_stat_tmp['High income'] / df_cov
id19_stat_tmp['sum']
df_covid19_stat_tmp['Low income'] = df_covid19_stat_tmp['Low income'] / df_covid
19_stat_tmp['sum']
df_covid19_stat_tmp['Lower middle income'] = df_covid19_stat_tmp['Lower middle i
ncome'] / df_covid19_stat_tmp['sum']
df_covid19_stat_tmp['Upper middle income'] = df_covid19_stat_tmp['Upper middle i
ncome'] / df_covid19_stat_tmp['sum']

fig = go.Figure()

##  'High income'을 위한 bar 트레이스 추가
fig.add_trace(go.Bar(
    x = df_covid19_stat_tmp['High income'],
    y = df_covid19_stat_tmp.index, orientation = 'h', name = 'High income',
    marker = dict(line = dict(color = 'white', width = 2))))
```

```
##  'Upper middle income'을 위한 bar 트레이스 추가
fig.add_trace(go.Bar(
    x = df_covid19_stat_tmp['Upper middle income'],
    y = df_covid19_stat_tmp.index, orientation = 'h', name = 'Upper middle incom
e',
    marker = dict(line = dict(color = 'white', width = 2))))

##  'Lower middle income'을 위한 bar 트레이스 추가
fig.add_trace(go.Bar(
    x = df_covid19_stat_tmp['Lower middle income'],
    y = df_covid19_stat_tmp.index, orientation = 'h', name = 'Lower middle incom
e',
    marker = dict(line = dict(color = 'white', width = 2))))

##  'Low income'을 위한 bar 트레이스 추가
fig.add_trace(go.Bar(
    x = df_covid19_stat_tmp['Low income'],
    y = df_covid19_stat_tmp.index, orientation = 'h', name = 'Low income',
    marker = dict(line = dict(color = 'white', width = 2))))

for index, row in df_covid19_stat_tmp.iterrows():
    ##  'High income' 값 표시를 위한 주석 레이어 추가
    fig.add_annotation(
        xref = 'x', yref = 'y',
        x = row['High income'] / 2, y = index,
        text = str(round(row['High income'] * 100, 1)) + '%',
        showarrow = False, font = dict(color = 'white'))
    ##  'Upper middle income' 값 표시를 위한 주석 레이어 추가
    fig.add_annotation(
        xref = 'x', yref = 'y',
        x = (row['Upper middle income'] / 2) + row['High income'], y = index,
        text = str(round(row['Upper middle income'] * 100, 1)) + '%',
        showarrow = False, font = dict(color = 'white'))
    ##  'Lower middle income' 값 표시를 위한 주석 레이어 추가
    fig.add_annotation(
        xref = 'x', yref = 'y',
        x = (row['Lower middle income'] / 2) + row['High income'] + row['Upper m
iddle income'], y = index,
        text = str(round(row['Lower middle income'] * 100, 1)) + '%',
        showarrow = False, font = dict(color = 'white'))
    ##  'Low income' 값 표시를 위한 주석 레이어 추가
    fig.add_annotation(
        xref = 'x', yref = 'y',
        x = (row['Low income'] / 2) + row['High income'] + row['Upper middle inc
ome'] + row['Lower middle income'], y = index,
        text = str(round(row['Low income'] * 100, 1)) + '%',
```

```
        showarrow = False, font = dict(color = 'white'))

fig.update_layout(barmode = 'stack',
         title = dict(text = '국가 소득급간별 코로나 19 현황', x = 0.5),
         xaxis = dict(title = '', tickformat = '.0%'),
         yaxis = dict(title = ''),
         legend = dict(orientation = 'h', traceorder = 'normal'),
         margin = margins_P)

fig.show()
```



실행결과IV-10. python 의 비율 막대 그래프

6. 파이 차트

파이 차트는 원 그래프라고도 말하는 차트이다. 파이 차트는 비율 막대 그래프와 같이 데이터 전체에 대한 각 변량의 비율만큼의 슬라이스 모양으로 백분율을 표시한 차트이다. 각 부분의 중심 각도를 비율로 변환해서 슬라이스를 만들어 데이터의 구성을 표현하는데, 전체적인 비율을 쉽게 파악할 수 있다는 장점이 있다. 일부 연구 에 따르면 사람으 인식은 면적보다는 길이에 더 민감하기 때문에 파이 차트에서의 각 부분들에 대한 크기 비교는 막대 차트의 크기 비교에 비해 어렵다고 전해진다. 특히 파이 차트에 표시되는 변량이 많은 경우와 변량의 크기가 유사한 경우에는 특히 파이 차트가 적절치 않다. 하지만 일부 변량에 데이터가 집중되는 경우에는 막대 그래프보다 원형 파이 차트로 구성하는게 효과적일수 있다.

plotly 에서는 파이 차트를 만들기 위해 pie 트레이스를 제공한다. pie 트레이스는 X, Y 축으로 구성되는 데카르트 좌표계를 사용하지 않기 때문에 'x', 'y' 속성이 사용되지 않는다. 대신 'values'와 'labels' 속성을 사용하여 데이터를 표현한다. 'values' 속성에 설정된 수치 배열에 따라 원을 각각의 슬라이스로 구분하게 되고 'labels'에 설정된 베열이 각각의 트레이스 이름으로 설정된다. 또 원 그래프에 표시되는 데이터 값은 기본적으로 백분률 값이 표시된다. 다음은 pie 트레이스에서 사용되는 주요 속성이다.

속성			속성 설명	속성값(R 속성값)
title	font	color	제목 글자 색상 설정	색상
		family	제목 글자 HTML 폰트 설정	폰트명
		size	제목 글자 크기 설정	1이상의 수치
	position		제목 위치 설정	("top left" | "top center" | "top right" | "middle center" | "bottom left" | "bottom center" | "bottom right")
	text		제목 문자열 설정	문자열
values			섹터의 값 설정	list, numpy array, or Pandas series of numbers, strings, or datetimes.
labels			섹터의 라벨 설정	list, numpy array, or Pandas series of numbers, strings, or datetimes.
pull			중심으로부터 밖으로 빠지는 비율 설정	number or array of numbers between or equal to 0 and 1
marker	colors		파이 색상 설정	list, numpy array, or Pandas series of numbers, strings, or datetimes.
	line	color	파이의 선 색상 설정	color or array of colors
		width	파이의 선 두께 설정	number or array of numbers greater than or equal to 0
textinfo			파이 차트에 표시할 텍스트 정보 설정	flaglist string. Any combination of "label", "text", "value", "percent" joined with a "+" OR "none".
direction			파이 차트에 표시하는 방향의 설정	("clockwise" | "counterclockwise")
hole			파치 차트 중간의 구멍 크기 설정	number between or equal to 0 and 1
insidetextfont	color		파이 차트 내부 글자 색상 설정	색상
	family		파이 차트 내부 폰트 설정	폰트명
	size		파이 차트 내부 폰트 사이즈 설정	1이상의 수치
insidetextorientation			파이차트 내부 텍스트 방향 설정	("horizontal" | "radial" | "tangential" | "auto")
outsidetextfont	color		파이 차트 외부 글자 색상 설정	색상
	family		파이 차트 외부 폰트 설정	폰트명
	size		파이 차트 외부 폰트 사이즈 설정	1이상의 수치
rotation				
sort				

plotly 의 pie 트레이스는 기본적으로 시계 반대방향으로 'values'의 순서대로 표시된다. 만약 시계 방향으로 표시하기를 원한다면 'direction'을 'clockwise'로 설정해준다. 정렬을 할지 여부는 'sort' 속성으로 사용하는데 파이 차트에서는 가급적 정렬된 상태로 사용하는 것이 효과적이다.

다음은 대학의 졸업자들이 각 계열별로 어떻게 구성되는지를 표시하는 파이 차트를 만드는 R 과 python 코드이다. 파이 차트를 위해 먼저 전체 졸업자수를 계열별로 합산한 데이터를 만들어주었고 'values'를 졸업자수로, 'labels'를 대계열 이름으로 설정하였다.

- R

R 에서 pie 트레이스를 만들기 위해서는 `add_trace(type = 'pie')`를 사용하거나 `add_pie()`를 사용한다.

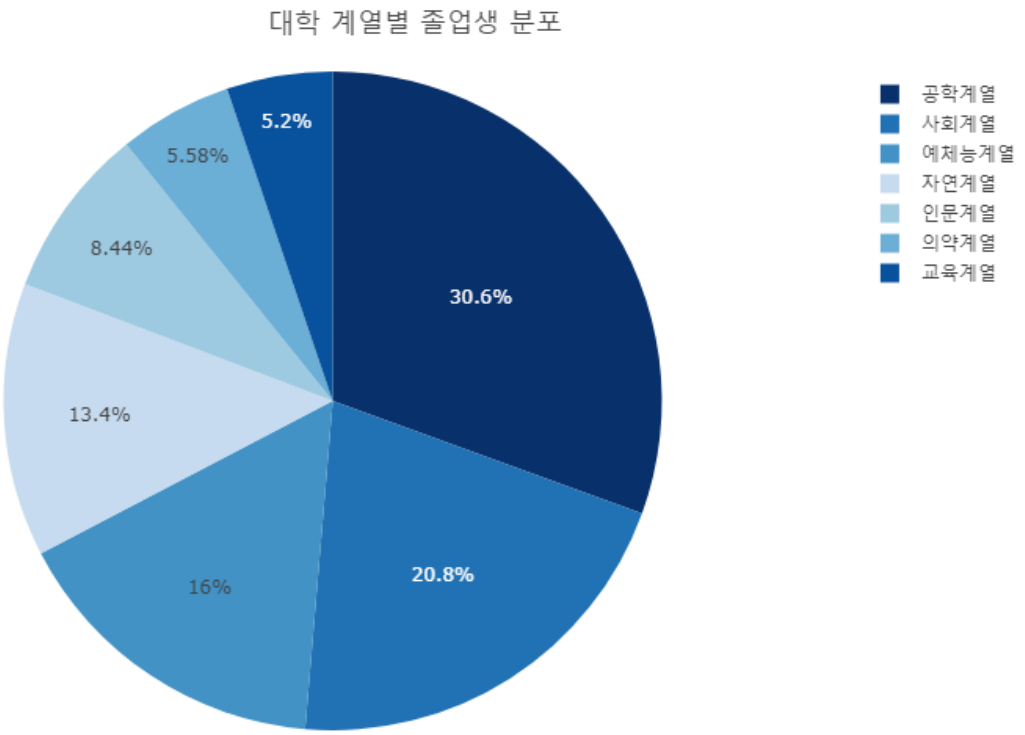
```
df_취업률_500 |> group_by(대계열) |>
  summarise(졸업자수 = sum(졸업자수)) |>
  plot_ly() |>
  ## value 와 labels 를 매핑한 파이 trace 추가
```

```
  add_trace(type = 'pie', values = ~졸업자수, labels = ~대계열, direction = 'clock
wise') |>
  layout(title = list(text = '대학 계열별 졸업생 분포'),
         margin = margins_R)
```

- python

python 에서 pie 트레이스를 만들기 위해서는 `add_trace()`에서 `plotly.graph_objects.Pie()`를 사용하거나 `plotly.express.pie()`를 사용한다.

```
df_pie = df_취업률_500.groupby('대계열').sum('졸업자수').reset_index()[['대계열', '
졸업자수']]

fig = go.Figure()

## value 와 labels 를 매핑한 파이 trace 추가
fig.add_trace(go.Pie(
    values = df_pie['졸업자수'], labels = df_pie['대계열'],
    direction = 'clockwise',  sort = True))

fig.show()
```



실행결과 IV-11. R 의 파이 그래프

6.1. 파이 차트의 라벨 표시

pie 트레이스는 기본적으로 백분율을 자동 계산하여 표시해 준다. 하지만 'textinfo' 속성을 사용하면 백분율 외에도 데이터 자체 값이나 백분율과 데이터 값을 혼용할 수 있다. 다음은 'textinfo' 속성을 달리하여 파이 차트를 그리는 R 과 python 코드이다.

- R

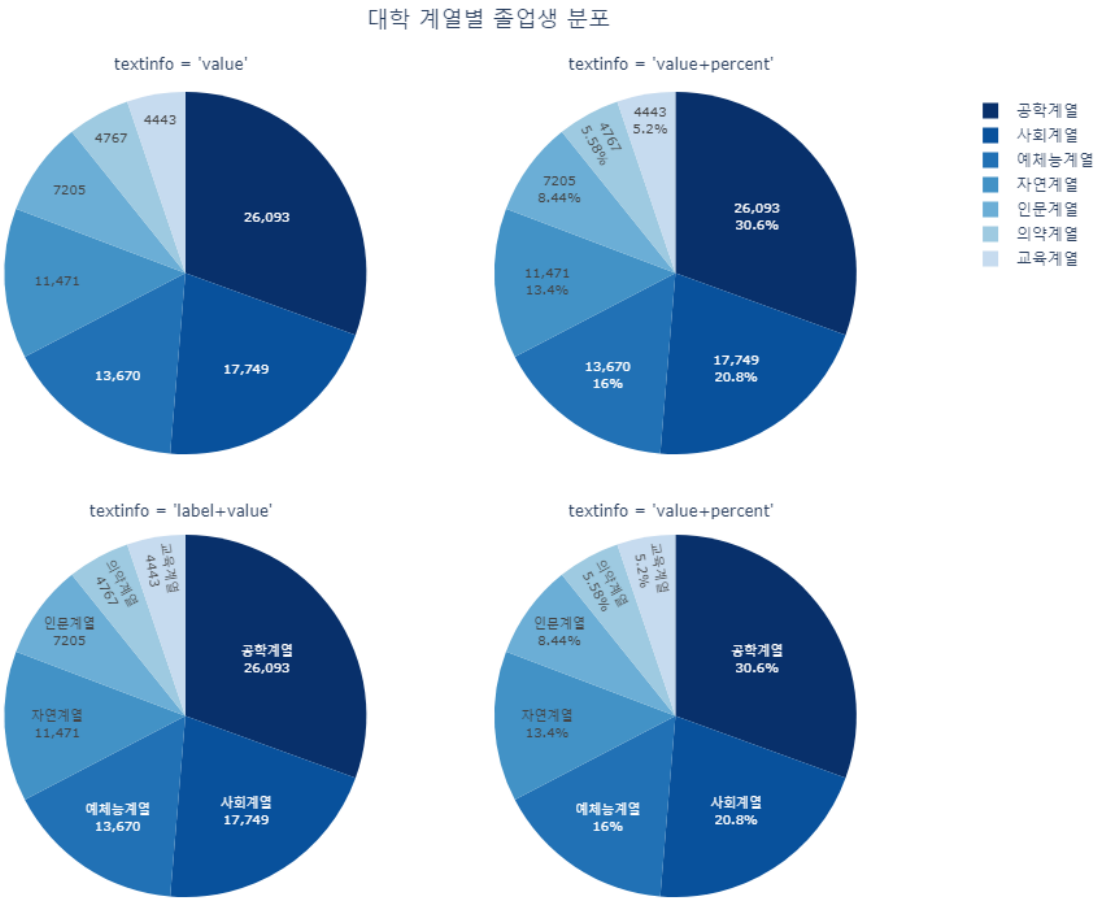
```
p_pie <- df_취업률_500 |> group_by(대계열) |>
  summarise(졸업자수 = sum(졸업자수)) |>
  plot_ly()

p_pie |>
  ## value 를 매핑한 파이 trace 추가
  add_trace(type = 'pie', values = ~졸업자수, labels = ~대계열, direction = 'clockwise',
            textinfo = 'value') |>
  layout(title = list(text = '대학 계열별 취업률 분포'),
         margin = margins_R)

p_pie |>
  ## value 와 percent 를 매핑한 파이 trace 추가
  add_trace(type = 'pie', values = ~졸업자수, labels = ~대계열, direction = 'clockwise',
            textinfo = 'value + percent') |>
  layout(title = list(text = '대학 계열별 취업률 분포'),
         margin = margins_R)

p_pie |>
  ## value 와 labels 를 매핑한 파이 trace 추가
  add_trace(type = 'pie', values = ~졸업자수, labels = ~대계열, direction = 'clockwise',
            textinfo = 'label+value') |>
  layout(title = list(text = '대학 계열별 취업률 분포'),
         margin = margins_R)

p_pie |>
  ## value 와 percent 를 매핑한 파이 trace 추가
  add_trace(type = 'pie', values = ~졸업자수, labels = ~대계열, direction = 'clockwise',
            textinfo = 'label+percent') |>
```

```
  layout(title = list(text = '대학 계열별 취업률 분포'),
         margin = margins_R)
```

- python

```
#####################################
fig = go.Figure()

## value 를 매핑한 파이 trace 추가
fig.add_trace(go.Pie(
    values = df_pie['졸업자수'], labels = df_pie['대계열'],
    direction = 'clockwise', textinfo = 'value'))

fig.update_layout(title = dict(text = '대학 계열별 졸업생 분포', x = 0.5))

fig.show()

#####################################
fig = go.Figure()

## value 와 percent 를 매핑한 파이 trace 추가
fig.add_trace(go.Pie(
    values = df_pie['졸업자수'], labels = df_pie['대계열'],
    direction = 'clockwise', textinfo = 'value + percent'))

fig.update_layout(title = dict(text = '대학 계열별 졸업생 분포', x = 0.5))

fig.show()

#####################################
fig = go.Figure()

## value 와 label 를 매핑한 파이 trace 추가
fig.add_trace(go.Pie(
    values = df_pie['졸업자수'], labels = df_pie['대계열'],
    direction = 'clockwise', textinfo = 'label+value'))

fig.update_layout(title = dict(text = '대학 계열별 졸업생 분포', x = 0.5))

fig.show()

#####################################
fig = go.Figure()

## label 과 percent 를 매핑한 파이 trace 추가
```

```
fig.add_trace(go.Pie(
    values = df_pie['졸업자수'], labels = df_pie['대계열'],
    direction = 'clockwise', textinfo = 'label+percent'))

fig.update_layout(title = dict(text = '대학 계열별 졸업생 분포', x = 0.5))

fig.show()
```



실행결과 IV-12. python 의 파이 차트의 라벨 표시

6.2. 도넛 차트

도넛 차트는 도넛 형태로 생긴 파이 차트의 변형 차트이다. 파이 차트에 중앙 홀을 만듦으로 만들 수 있는 차트이다. 중앙 홀은 pie 트레이스의 'hole' 속성을 사용해 간단히 만들 수 있다. 'hole'은 0 부터 1 까지의 수치를 가지는데 파이 차트의 반지름을 1 로한 상대적 비율을

설정한다. 하지만 중요한 것은 이 중간 공간을 어떻게 이용할지이다. 보통 이 중앙 홀에 해당 파이 차트를 대표하는 간단한 문구를 넣어주면 효율적으로 사용이 가능하다. 이 문구는 비율 막대 그래프에서 사용했던 annotation 으로 넣어줄 수 있다.

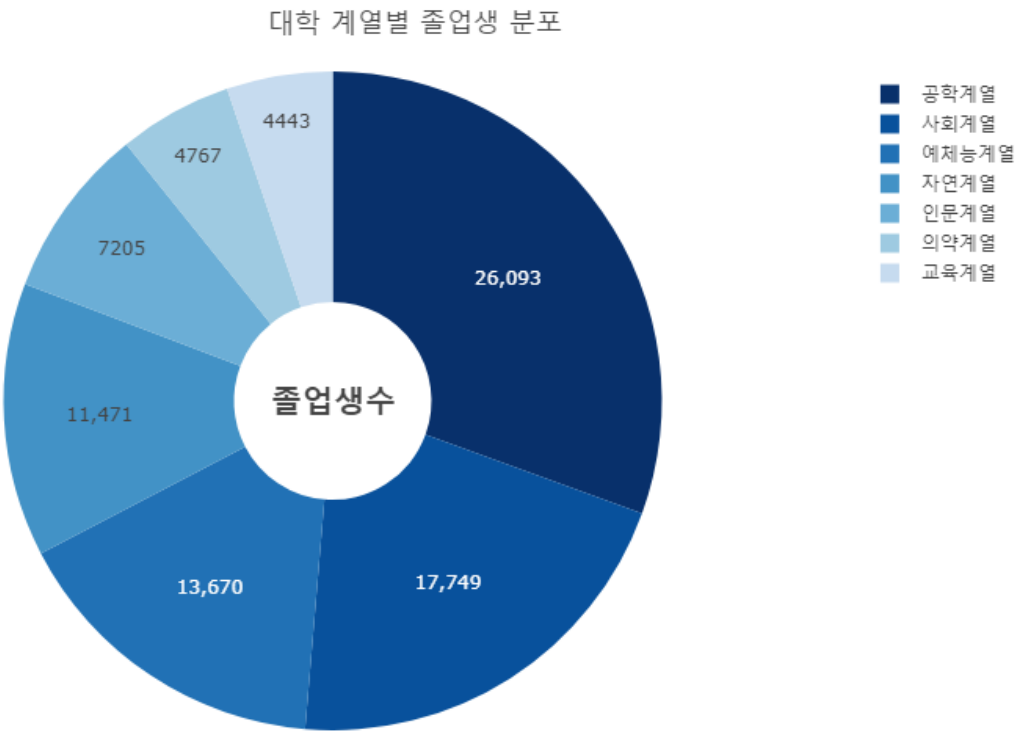
다음은 도넛 차트를 만들고 중앙 홀에 '졸업생수' 문구를 넣는 R 과 python 코드이다.

- R

```r
p_pie |>
  add_trace(type = 'pie', values = ~졸업자수, labels = ~대계열, direction = 'clockwise',
  ## hole 을 사용한 도넛 차트
            textinfo = 'value', hole = 0.3) |>
  add_annotations(x = 0.5, y = 0.5, text = '<b>졸업생수</b>',
                  showarrow = FALSE, xanchor = 'center',
                  font = list(size = 20)) |>
  layout(title = list(text = '대학 계열별 졸업생 분포'),
         margin = margins_R)
```

- python

```python
fig = go.Figure()

fig.add_trace(go.Pie(
    values = df_pie['졸업자수'], labels = df_pie['대계열'],
    direction = 'clockwise', sort = True,
    ## hole 을 사용한 도넛 차트
    hole = 0.3))

fig.add_annotation(x = 0.5, y = 0.5, text = '<b>졸업생수</b>',
                   showarrow = False, xanchor = 'center',
                   font = dict(size = 20))

fig.update_layout(title = dict(text = '대학 계열별 졸업생 분포', x = 0.5))

fig.show()
```



실행결과 IV-13. R 의 도넛 차트

6.3. 파이 차트의 강조

보통 시각화에서 특정한 부분을 강조하고 싶을 때 해당 부분의 색을 도드라지게 해서 강조하는 것이 일반적 방법이다. 하지만 파이 차트에서는 색을 통해 강조하는 방법도 가능하겠지만 마치 피자의 한 조각을 떼낸 것처럼 강조하고자 하는 부분의 슬라이스를 조금 바깥쪽으로 빼서 강조하는 경우도 있다. 이를 위해서 pie 트레이스에서 'pull' 속성을 사용한다. 'pull'에는 0 부터 1 까지의 수치나 수치 배열이 오는데 수치가 올 경우 설정한 수치만큼 슬라이스를 밖으로 빼준다.

다음은 교육계열을 강조하는 졸업생수 도넛 차트의 R 과 python 코드이다.

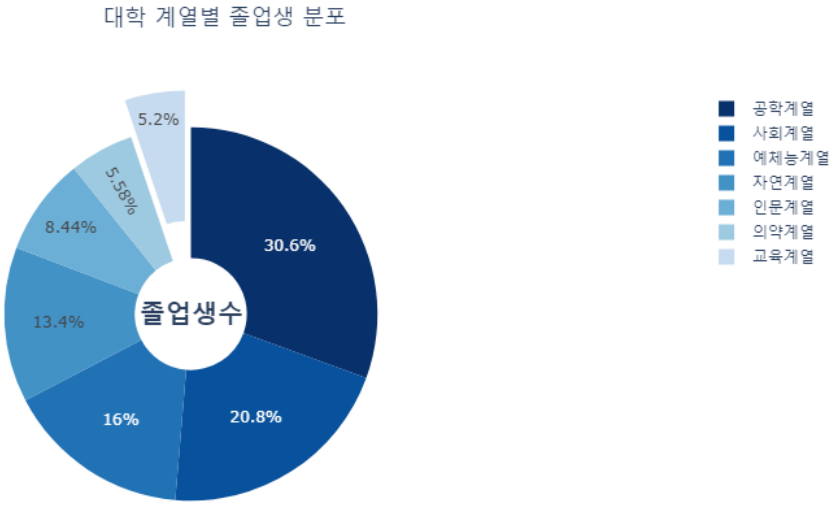
- R

```
p_pie |>
  add_trace(type = 'pie', values = ~졸업자수, labels = ~대계열, direction = 'clockwise',
            textinfo = 'value', hole = 0.3,
            ## 파이 차트 강조 설정
            pull = c(0, 0.2, 0, 0, 0, 0, 0)) |>
  add_annotations(x = 0.5, y = 0.5, text = '<b>졸업생수</b>',
```

```
                  showarrow = FALSE, xanchor = 'center',
                  font = list(size = 20)) |>
  layout(title = list(text = '대학 계열별 졸업생 분포'),
         margin = margins_R)
```

- python

```
fig = go.Figure()

fig.add_trace(go.Pie(
    values = df_pie['졸업자수'], labels = df_pie['대계열'],
    direction = 'clockwise', sort = True, hole = 0.3,
    ## 파이 차트 강조 설정
    pull = (0, 0.2, 0, 0, 0, 0, 0)))

fig.add_annotation(x = 0.5, y = 0.5, text = '<b>졸업생수</b>',
                   showarrow = False, xanchor = 'center',
                   font = dict(size = 20))

fig.update_layout(title = dict(text = '대학 계열별 졸업생 분포', x = 0.5))

fig.show()
```



실행결과IV-14. python 의 파이 차트의 강조

7. 선버스트 차트

선버스트(sunburst)의 사전적 의미는 '햇살'이다. 햇살처럼 퍼저나가는 데이터를 표현하기 위해 사용되는 형태의 시각화이다. plotly 에서 지원하는 sunburst 트레이스는 루트(root)에서 잎(leaf)까지 원의 방사형 방향의 바깥쪽으로 퍼져나가면서 계층화된 데이터를 시각화한다. 따라서 sunburst 트레이스를 그리기 위해서는 계층화된 데이터가 필요하다. sunburst 트레이스의 각 섹터는 'labels'와 'parents'의 속성에서 설정하는데 'labels'은 선버스트 trace 에서 표시되는 모든 섹터의 표시 라벨의배열(python 은 리스트)를 설정한다. 'parents'는 'labels'와 길이가 같은 배열(리스트)로 설정하여 모든 'labels'와 1:1 로 매칭되는데, 각 'labels'의 부모 섹터를 가리키는 'labels' 값을 설정한다. 만약 'parents'에 설정된 부모 섹터가''이면 가장 안쪽의 노드인 루트 노드가 된다. 'values'는 해당 섹터에 표시될 값을 설정한다. 'values' 값에 따라 섹터의 크기도 결정된다. 다음은 sunburst 트레이스에서 사용하는 주요 속성이다.

속성			속성 설명	속성값(R 속성값)
parents			각 섹터의 부모 섹터 설정	list, numpy array, or Pandas series of numbers, strings, or datetimes.(dataframe column, list, vector)
values			각 섹터의 값 설정	list, numpy array, or Pandas series of numbers, strings, or datetimes.(dataframe column, list, vector)
labels			각 섹터의 라벨 설정	list, numpy array, or Pandas series of numbers, strings, or datetimes.(dataframe column, list, vector)
marker	colors		해당 트레이스의 색상값 설정	list, numpy array, or Pandas series of numbers, strings, or datetimes.(dataframe column, list, vector)
	colorscale		해당 트레이스의 컬러스케일 설정	컬러스케일
	line	color	각 섹터를 둘러싼 선 색 설정	색상 또는 색상 배열
		width	각 섹터를 둘러싼 선 두께 설정	0이상의 수치나 수치 배열
	reversescale		색상 매핑을 역순으로 설정	논리값
textinfo			그래프에 표현될 텍스트 정보 설정	"label", "text", "value", "current path", "percent root", "percent entry", "percent parent" 의 문자열이나 +를 사용한 조합 문자
branchvalues			values 속성의 값이 어떻게 더해질지를 설정	("remainder" | "total")
count			values 속성이 설정되지 않을 경우 무엇을 표기해야할지 설정	"branches", "leaves"의 문자열이나 +를 사용한 조합문자열
insidetextfont	color		파이 차트 내부 글자 색상 설정	색상
	family		파이 차트 내부 폰트 설정	폰트명
	size		파이 차트 내부 폰트 사이즈 설정	1이상의 수치
insidetextorientation			파이차트 내부 텍스트 방향 설정	("horizontal" | "radial" | "tangential" | "auto")
outsidetextfon	color		파이 차트 외부 글자 색상 설정	색상값
	family		파이 차트 외부 폰트 설정	폰트명
	size		파이 차트 외부 폰트 사이즈 설정	1이상의 수치
root	color		sunburst/treemap/icicle 트레이스의 루트 노드 색상 설정	색상값
leaf	opacity		리프노드의 투명도 설정	0이상 1이하의 수치
level			해당 트레이스 계층 구조의 레벨 설정	수치나 좌표상의 변량값
maxdepth			계층 구조 레벨의 최대 깊이 설정	정수
rotation			반시계방향으로의 회전 각도 설정	각도
sort			섹터의 값에 따른 정렬 여부의 설정	논리값

다음은 대학의 졸업자 구성을 대계열, 중계열의 계층적으로 표현한 선버스트 차트를 만드는 R 과 python 코드이다. 다음의 그림은 선버스트 차트에 필요한 'labels', 'parents', 'values'의 구조를 도식한 것이다.

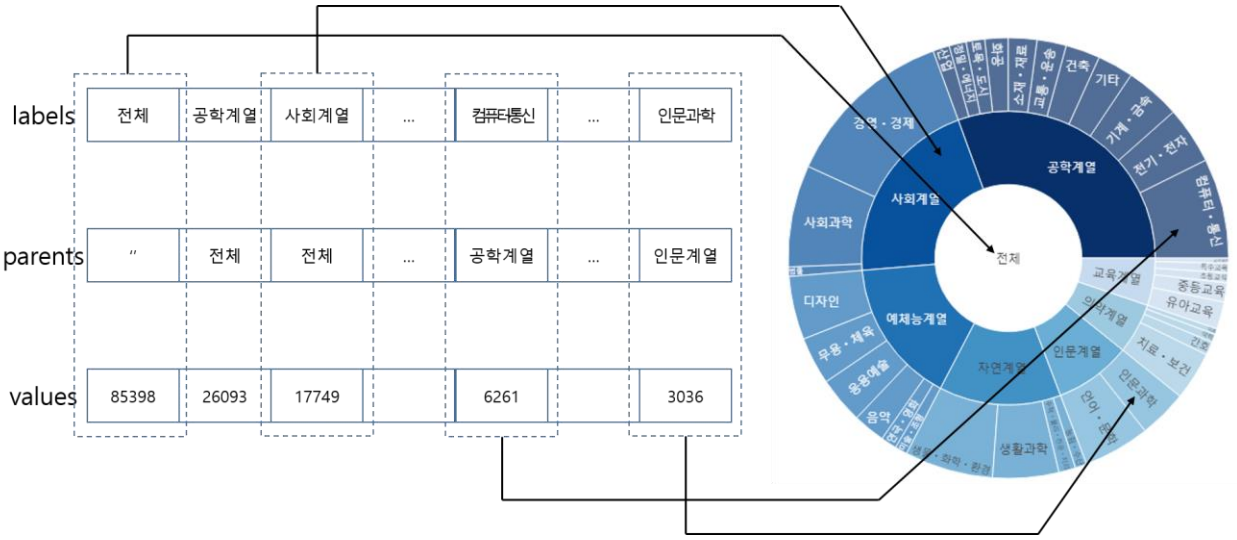


그림 IV-4. 선버스트 차트의 구조

앞의 그림에서 보면 'lables'에는 총 43 개의 원소(전체 1 개, 대계열 7 개, 중계열 35 개)가 필요하다. 이들의 리스트를 만들어서 'labels'에 설정하였다. 다음으로 'parents'에 'labels'의 각각의 원소에 해당하는 부모노드를 설정하는데 'labels'의 첫 번째 원소인 '전체'는 루트노드이기 때문에 'parents'의 첫 번째 원소를''로 설정한다. 이후 대계열에 속하는 'labels'는 루트노드인 '전체'로 설정하고 다음부터 나오는 중계열은 해당 중계열이 속하는 대계열 이름을 설정하였다. 마지막으로 각각의 값을 설정하기 위해 합계값을 구한다. 루트 노드에 설정할 전체 합계 값, 각각의 대계열에 설정할 합계값, 중계열의 합계값들을 구해 이들 배열(리스트)를 연결하여 전체 배열(리스트)를 만들어 준다.

- R

R 에서 sunburst 트레이스를 만들기 위해서는 `add_trace(type = 'sunburst', ...)`를 사용한다.

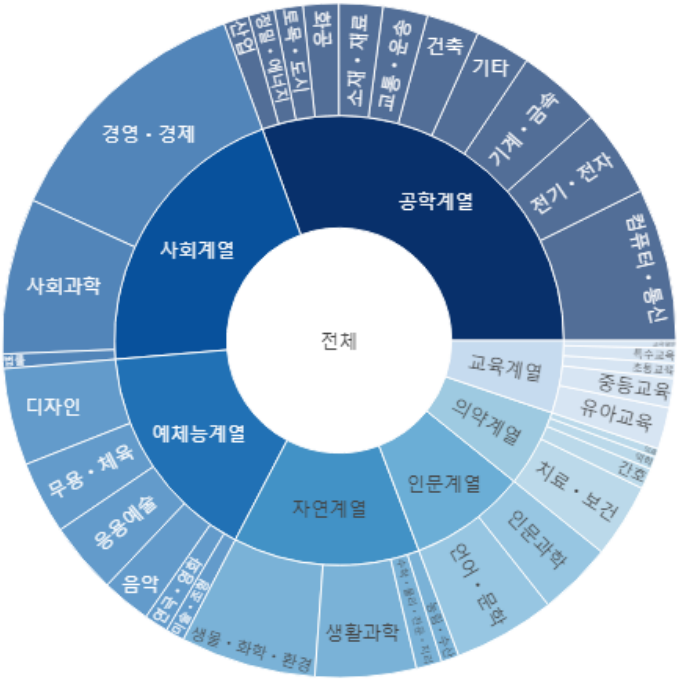
```
## 선버스트 차트를 위한 데이터 전처리
df_sunburst <- df_취업률_500 |> group_by(대계열, 중계열) |>
  summarise(졸업자수 = sum(졸업자수))

all_sum <- sum(df_sunburst$졸업자수)
계열_sum <- df_sunburst |> group_by(대계열) |>
  summarise(sum = sum(졸업자수)) |>
  select(sum) |> pull()

df_sunburst |> plot_ly() |>
```

```
  add_trace(type = 'sunburst',
            ## sunburst 트레이스의 labels 설정
            labels = c('전체', unique(df_sunburst$대계열), df_sunburst$중계열),
            ## sunburst 트레이스의 parents 설정
            parents = c('', rep('전체', 7), df_sunburst$대계열),
            ## sunburst 트레이스의 values 설정
            values = c(all_sum, 계열_sum, df_sunburst$졸업자수),
  branchvalues = 'total')
```

- python

python 에서 sunburst 트레이스를 만들기 위해서는 `add_trace()`에 `plotly.graph_objects.Sunburst()`를 사용하거나 `plotly.express.sunburst()`를 사용한다.

```
## 선버스트 차트를 위한 데이터 전처리
df_sunburst = df_취업률_500.groupby(['대계열', '중계열']).sum()['졸업자수'].reset_index()

all_sum = df_취업률_500.sum()['졸업자수']
계열_sum = df_취업률_500.groupby('대계열').sum().reset_index()['졸업자수'].values.tolist()

fig = go.Figure()

fig.add_trace(go.Sunburst(
    ## sunburst 트레이스의 labels 설정
    labels = ['전체'] + df_sunburst['대계열'].unique().tolist() + df_sunburst['중계열'].tolist(),
    ## sunburst 트레이스의 parents 설정
    parents = [''] + ['전체'] * 7 + df_sunburst['대계열'].tolist(),
    ## sunburst 트레이스의 values 설정
    values = [all_sum] + 계열_sum + df_sunburst['졸업자수'].values.tolist(),
    branchvalues = 'total'))

fig.show()
```



실행결과 IV-15. R 의 선버스트 차트

sunburst 트레이스에서 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

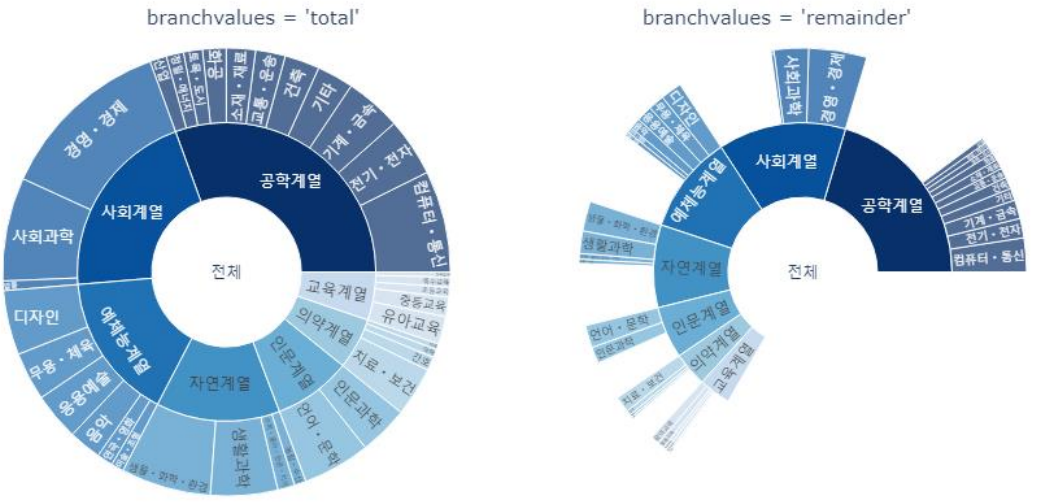
```
df_sunburst <- df_취업률_500 |> group_by(대계열, 중계열) |>
  summarise(졸업자수 = sum(졸업자수))

all_sum <- sum(df_sunburst$졸업자수)
계열_sum <- df_sunburst |> group_by(대계열) |>
  summarise(sum = sum(졸업자수)) |>
  select(sum) |> pull()

df_sunburst |> plot_ly() |>
  add_trace(type = 'sunburst',
```

```
            labels = c('전체', unique(df_sunburst$대계열), df_sunburst$중계열),
            parents = c('', rep('전체', 7), df_sunburst$대계열),
            values = c(all_sum, 계열_sum, df_sunburst$졸업자수),
  branchvalues = 'total', maxdepth = 3, domain = list(x = c(0, 0.45))) |>
  add_trace(type = 'sunburst',
            labels = c('전체', unique(df_sunburst$대계열), df_sunburst$중계열),
            parents = c('', rep('전체', 7), df_sunburst$대계열),
            values = c(all_sum, 계열_sum, df_sunburst$졸업자수),
  branchvalues = 'reminder', maxdepth = 3, domain = list(x = c(0.55, 1))) |>
  add_annotations(x = 0.25, y = 1, text = "branchvalues = 'total'",
                  showarrow = F, xanchor = 'center')  |>
  add_annotations(x = 0.75, y = 1, text = "branchvalues = 'reminder'",
                  showarrow = F, xanchor = 'center')
```

- python

```
df_sunburst = df_취업률_500.groupby(['대계열', '중계열']).sum()['졸업자수'].reset_index()

all_sum = df_취업률_500.sum()['졸업자수']
계열_sum = df_취업률_500.groupby('대계열').sum().reset_index()['졸업자수'].values.tolist()

fig = go.Figure()
fig.add_trace(go.Sunburst(
    labels = ['전체'] + df_sunburst['대계열'].unique().tolist() + df_sunburst['중계열'].tolist(),
    parents = [''] + ['전체'] * 7 + df_sunburst['대계열'].tolist(),
    values = [all_sum] + 계열_sum + df_sunburst['졸업자수'].values.tolist(),
    branchvalues = 'total', domain = dict(x = (0, 0.45))
))
fig.add_trace(go.Sunburst(
    labels = ['전체'] + df_sunburst['대계열'].unique().tolist() + df_sunburst['중계열'].tolist(),
    parents = [''] + ['전체'] * 7 + df_sunburst['대계열'].tolist(),
    values = [all_sum] + 계열_sum + df_sunburst['졸업자수'].values.tolist(),
    branchvalues = 'remainder', domain = dict(x = (0.55, 1))
))
fig.add_annotation(x = 0.25, y = 1.1, text = "branchvalues = 'total'",
                   showarrow = False, xanchor = 'center',
                   font = dict(size = 15)
)
fig.add_annotation(x = 0.75, y = 1.1, text = "branchvalues = 'remainder'",
                   showarrow = False, xanchor = 'center',
```

```
                  font = dict(size = 15)
)

fig.show()
```



실행결과 IV-16. python 선버스트 차트의 branchvalue 별 결과

`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.

- R

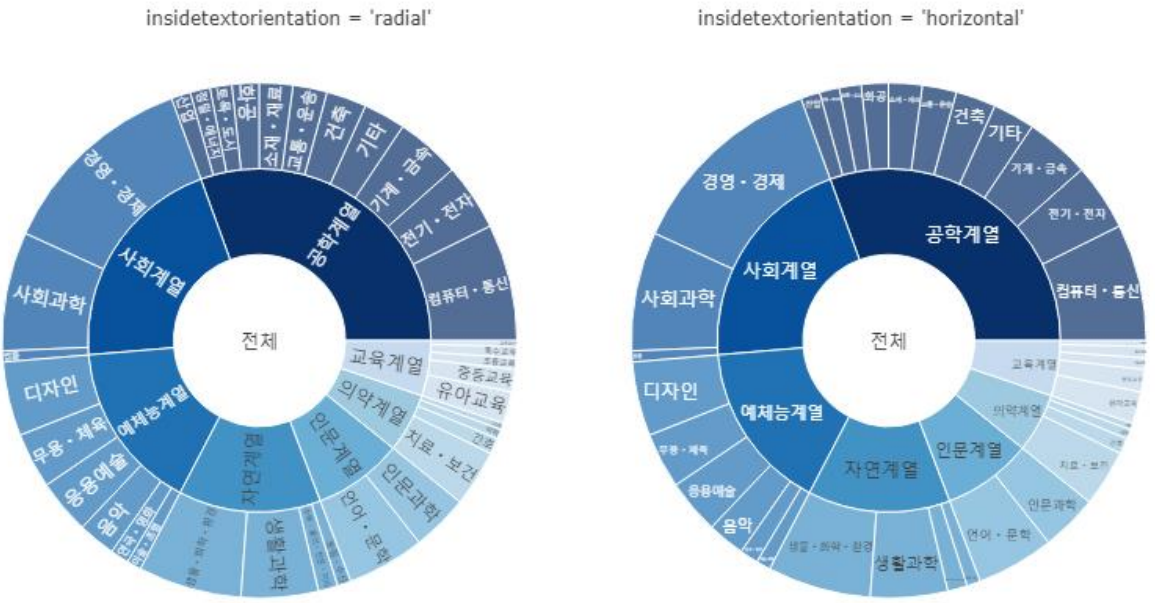
```
df_sunburst <- df_취업률_500 |> group_by(대계열, 중계열) |>
  summarise(졸업자수 = sum(졸업자수))

all_sum <- sum(df_sunburst$졸업자수)
계열_sum <- df_sunburst |> group_by(대계열) |>
  summarise(sum = sum(졸업자수)) |>
  select(sum) |> pull()

df_sunburst |> plot_ly() |>
  add_trace(type = 'sunburst',
            labels = c('전체', unique(df_sunburst$대계열), df_sunburst$중계열),
```

```
            parents = c('', rep('전체', 7), df_sunburst$대계열),
            values = c(all_sum, 계열_sum, df_sunburst$졸업자수),
  branchvalues = 'total', insidetextorientation = 'radial', maxdepth = 3, domain
 = list(x = c(0, 0.45))) |>
  add_trace(type = 'sunburst',
            labels = c('전체', unique(df_sunburst$대계열), df_sunburst$중계열),
            parents = c('', rep('전체', 7), df_sunburst$대계열),
            values = c(all_sum, 계열_sum, df_sunburst$졸업자수),
  branchvalues = 'total', insidetextorientation = 'horizontal',maxdepth = 3, dom
ain = list(x = c(0.55, 1))) |>
  add_annotations(x = 0.25, y = 1, text = "insidetextorientation = 'radial'",
                  showarrow = F, xanchor = 'center')  |>
  add_annotations(x = 0.75, y = 1, text = "insidetextorientation = 'horizontal'
",
                  showarrow = F, xanchor = 'center')
```

- python

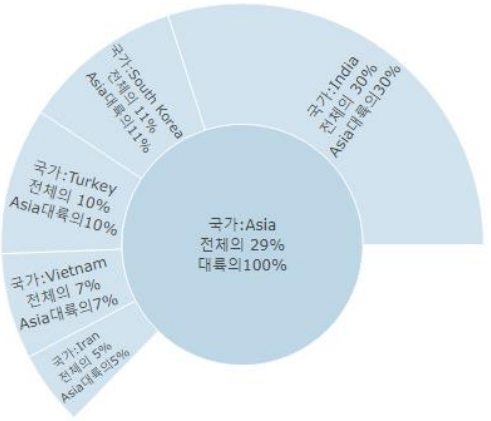
```
df_sunburst = df_취업률_500.groupby(['대계열', '중계열']).sum()['졸업자수'].reset_i
ndex()

all_sum = df_취업률_500.sum()['졸업자수']
계열_sum = df_취업률_500.groupby('대계열').sum().reset_index()['졸업자수'].values.t
olist()

fig = go.Figure()
fig.add_trace(go.Sunburst(
    labels = ['전체'] + df_sunburst['대계열'].unique().tolist() + df_sunburst['중
계열'].tolist(),
    parents = [''] + ['전체'] * 7 + df_sunburst['대계열'].tolist(),
    values = [all_sum] + 계열_sum + df_sunburst['졸업자수'].values.tolist(),
    branchvalues = 'total', insidetextorientation = 'radial', domain = dict(x =
(0, 0.45))
))
fig.add_trace(go.Sunburst(
    labels = ['전체'] + df_sunburst['대계열'].unique().tolist() + df_sunburst['중
계열'].tolist(),
    parents = [''] + ['전체'] * 7 + df_sunburst['대계열'].tolist(),
    values = [all_sum] + 계열_sum + df_sunburst['졸업자수'].values.tolist(),
    branchvalues = 'total', insidetextorientation = 'horizontal', domain = dict
(x = (0.55, 1))
))
fig.add_annotation(x = 0.25, y = 1.1, text = "nsidetextorientation = 'radial'",
                   showarrow = False, xanchor = 'center',
```

```
                    font = dict(size = 15)
)
fig.add_annotation(x = 0.75, y = 1.1, text = "insidetextorientation = 'horizonta
l'",
                    showarrow = False, xanchor = 'center',
                    font = dict(size = 15)
)

fig.show()
```



실행결과 IV-17. R 선버스트 차트의 insidetextorientation 결과

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



실행결과 IV-18. 아시아 섹터를 클릭한 python 선버스트 차트의 반응 결과

8. 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 차트를 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 treemap 트레이스의 주요 속성이다.

속성			속성 설명	속성값(R 속성값)
parents			각 섹터의 부모 섹터 설정	list, numpy array, or Pandas series of numbers, strings, or datetimes (dataframe column, list,
values			각 섹터의 값 설정	list, numpy array, or Pandas series of numbers, strings, or datetimes (dataframe column, list,
labels			각 섹터의 라벨 설정	list, numpy array, or Pandas series of numbers, strings, or datetimes (dataframe column, list,
marker	colors		해당 트레이스의 색상값 설정	list, numpy array, or Pandas series of numbers, strings, or datetimes (dataframe column, list,
	colorscale		해당 트레이스의 컬러스케일 설정	컬러스케일
	depthfade		Determines if the sector colors are faded towards the background from the leaves up to the headers.	(True | False | "reversed")
	line	color	각 섹터를 둘러싼 선 색 설정	색상 또는 색상 배열
		width	각 섹터를 둘러싼 선 두께 설정	0이상의 수치나 수치 배열
	reversescale		색상 매핑을 역순으로 설정	논리값
branchvalues			values 속성의 값이 어떻게 더해질지를 설정	("remainder" | "total")
count			values 속성이 설정되지 않을 경우 무엇을 표기해야할지 설정	"branches", "leaves"의 문자열이나 +를 사용한 조합 문자열
insidetextfont	color		파이 차트 내부 글자 색상 설정	색상
	family		파이 차트 내부 폰트 설정	폰트명
	size		파이 차트 내부 폰트 사이즈 설정	1이상의 수치
outsidetextfont	color		파이 차트 외부 글자 색상 설정	색상값
	family		파이 차트 외부 폰트 설정	폰트명
	size		파이 차트 외부 폰트 사이즈 설정	1이상의 수치
root	color		sunburst/treemap/icicle 트레이스의 구트 노드 색상 설정	색상값
level			해당 트레이스 계층 구조의 레벨 설정	수치나 좌표상의 변량값
maxdepth			계층 구조 레벨의 최대 깊이 설정	정수
sort			섹터의 값에 따른 정렬 여부의 설정	논리값

다음은 앞서 만들었던 sunburst 트레이스를 그대로 treemap 트레이스로 만든 R 과 python 코드이다.

- R

R 에서 treemap 트레이스를 만들기 위해서는 `add_trace(type = 'treemap', ...)`을 사용한다.

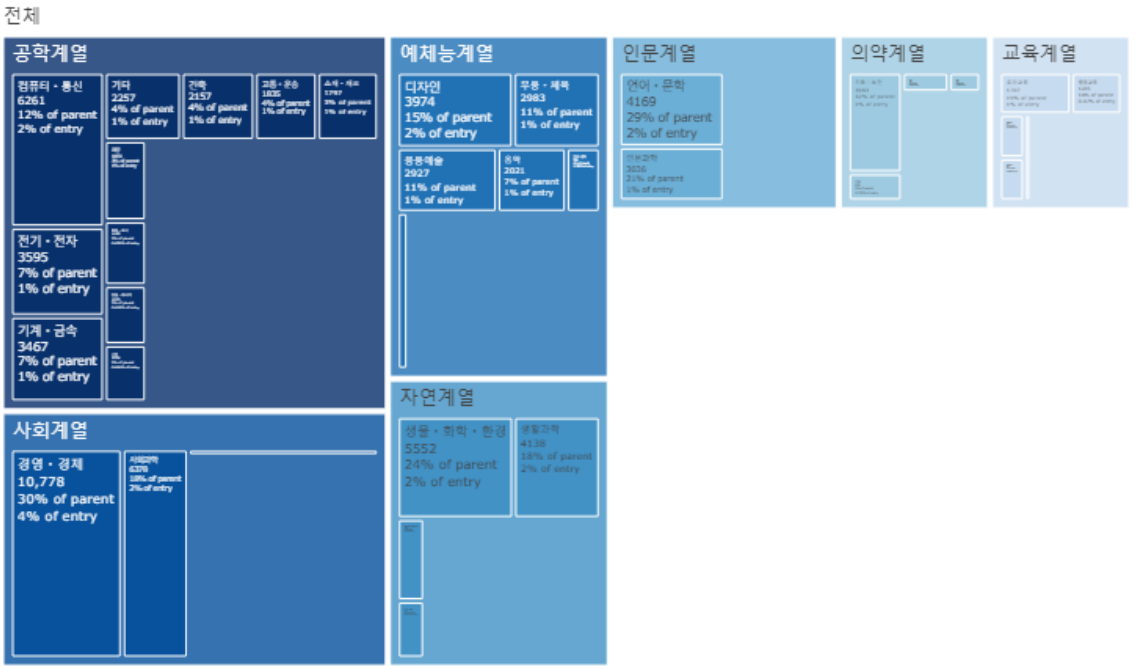
```r
plot_ly() |>
  add_trace(type = 'treemap',
            ## treemap 트레이스의 labels 설정
            labels = c('전체', unique(df_sunburst$대계열), df_sunburst$중계열),
            ## treemap 트레이스의 parents 설정
            parents = c('', rep('전체', 7), df_sunburst$대계열),
            ## treemap 트레이스의 values 설정
            values = c(all_sum, 계열_sum, df_sunburst$졸업자수),
            textinfo = 'label+value+percent parent+percent entry')
```

- python

python 에서 treemap 트레이스를 만들기 위해서는 `add_trace()`에 `plotly.graph_objects.Treemap()`을 사용하거나 `plotly.express.treemap()`을 사용한다.

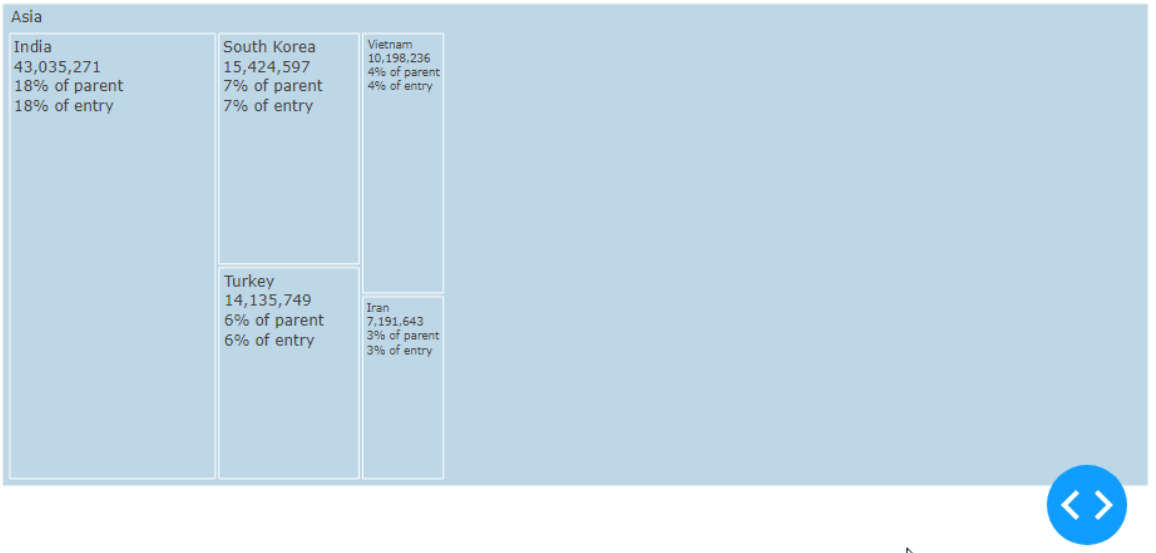
```python
fig = go.Figure()
fig.add_trace(go.Treemap(
    ## treemap 트레이스의 labels 설정
    labels = ['전체'] + df_sunburst['대계열'].unique().tolist() + df_sunburst['중계열'].tolist(),
```

```
    ## treemap 트레이스의 parents 설정
    parents = [''] + ['전체'] * 7 + df_sunburst['대계열'].tolist(),
    ## treemap 트레이스의 values 설정
    values = [all_sum] + 계열_sum + df_sunburst['졸업자수'].values.tolist(),
    textinfo = 'label+value+percent parent+percent entry'))

fig.show()
```



실행결과 IV-18. 아시아 섹터를 클릭한 R 트리맵 차트의 반응 결과

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



실행결과 IV-19. 아시아 섹터를 클릭한 트리맵 차트의 반응 결과