III. 분포의 시각화 – 미완성

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

R 에서 시각화의 종류에 따른 함수는 `plotly` 함수에 포함된 `add_*()`를 사용한다. 각각의 'trace' 종류에 따라 사용하는 함수들이 제공되는데 먼저 `plot_ly()`로 객체를 초기화하고 표현하고자 하는 'trace'를 `add_*()`를 사용하여 계속 'trace'를 추가하는 방식으로 시각화를 구현한다.

python 에서는 `plotly` 패키지의 서브모듈인 `plotly.express` 에서 제공하는 함수들을 사용할 수 있다. `plotly.express` 에서 제공하는 함수들은 앞 장에서 살펴본 `plotly.graphic_object` 의 `add_trace()` 보다 직관적이고 사용하기 쉽다. `plotly` 를 제공하는 제작사에서는 `plotly` 로 시각화 객체를 만들때 `plotly.graphic_object` 를 사용하기 보다는 `plotly.express` 를 사용하기를 권장한다. `plotly.express` 로 만든 객체도 결국 `plotly.graphic_object` 를 리턴하기 때문에 `plotly.graphic_object` 의 기능을 사용하여 다시 세부적인 설정을 할 수 있다. 하지만 `plotly.express` 는 결정적인 몇가지 단점이 있다. 첫 번째 단점은 'mesh'나 'isosurface'와 같은 3 차원 시각화는 아직 `plotly.express` 는 지원하지 않는다. 두 번째는 여러개의 trace 를 가지는 서브플롯, 다중 축의 사용, 여러개의 trace 를 가지는 패싯(facet)과 같은 시각화는 `plotly.express` 로 생성하는데 다소 어려움이 있다. 따라서 `plotly.express` 로는 최상위 시각화를 그리고 `add_trace()`를 사용하여 `plotly.graphic_object` 의 trace 추가 함수를 사용하여 계속 추가해주는 방식으로 구현하여야 한다. 또 `plotly.graphic_object` 에서 제공하는 함수와 `plotly.express` 에서 제공하는 함수의 속성도 다소 차이가 있기 때문에 `plotly.express` 는 사용이 간편하긴 하지만 사용할 때는 사용법을 잘 확인하고 사용해야 한다.[1]

---

[1] 본 책에서는 `plotly.graph_object` 위주로 설명한다. `plotly.express` 의 사용법은 https://plotly.com/python-api-reference/ 을 참조하라.

1. 산점도(scatter chart)

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

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

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

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

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

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

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

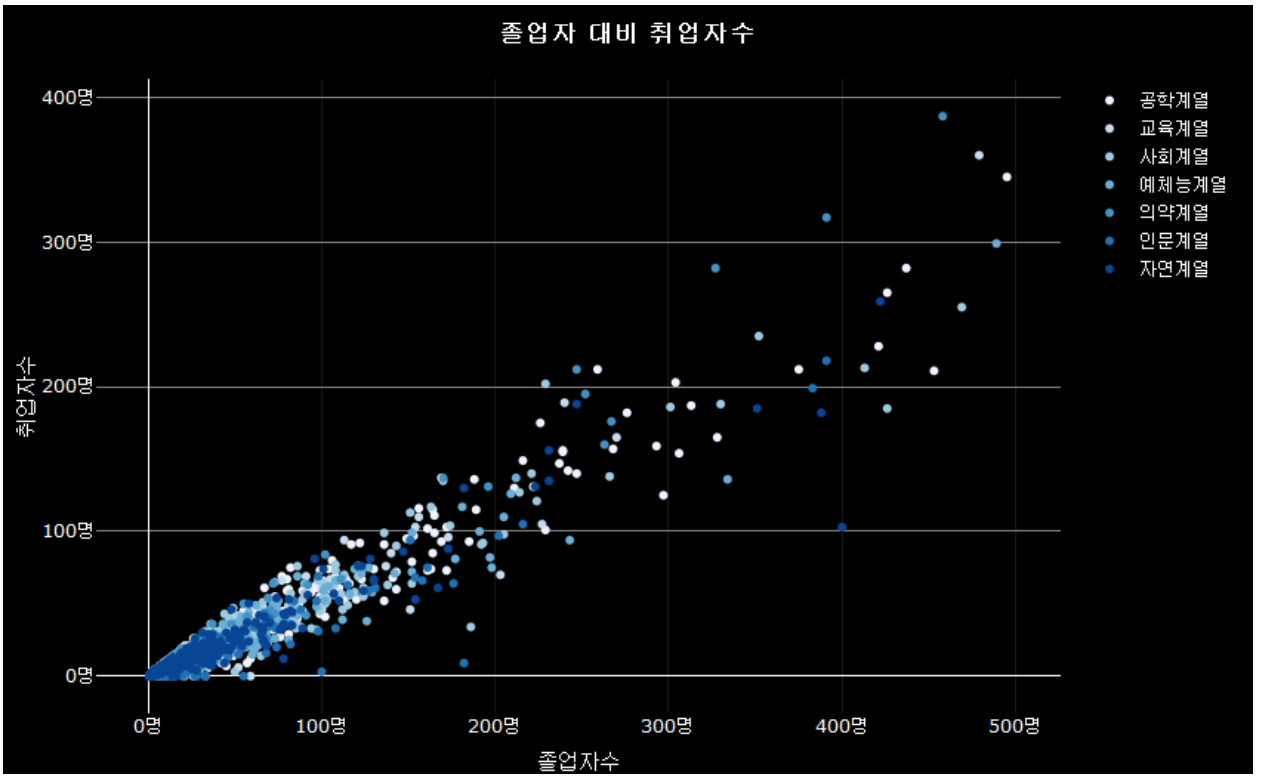
1.1. 기본 산점도 그리기

산점도를 그리는데 앞 장에서 `plotly` 를 그리기 위해 사용했던 `add_trace()`를 사용할 수 있지만 R 과 python 모두 산점도를 위한 전용 함수를 제공한다.

- R : add_markers()

R 에서 산점도는 `add_markers()`를 사용하여 만들 수 있다. `add_markers()`에서 사용하는 속성은 `add_trace()`에서 사용하는 속성 그대로 사용할 수 있다. 만약 이 중 여러개를 동시에 그리기 위해서는 R 자체적으로 제공하는 파이프(`|>`)나 `tidyverse` 에서 지원하는 파이프(`%>%`)로 계속 겹쳐서 그릴 수 있다.

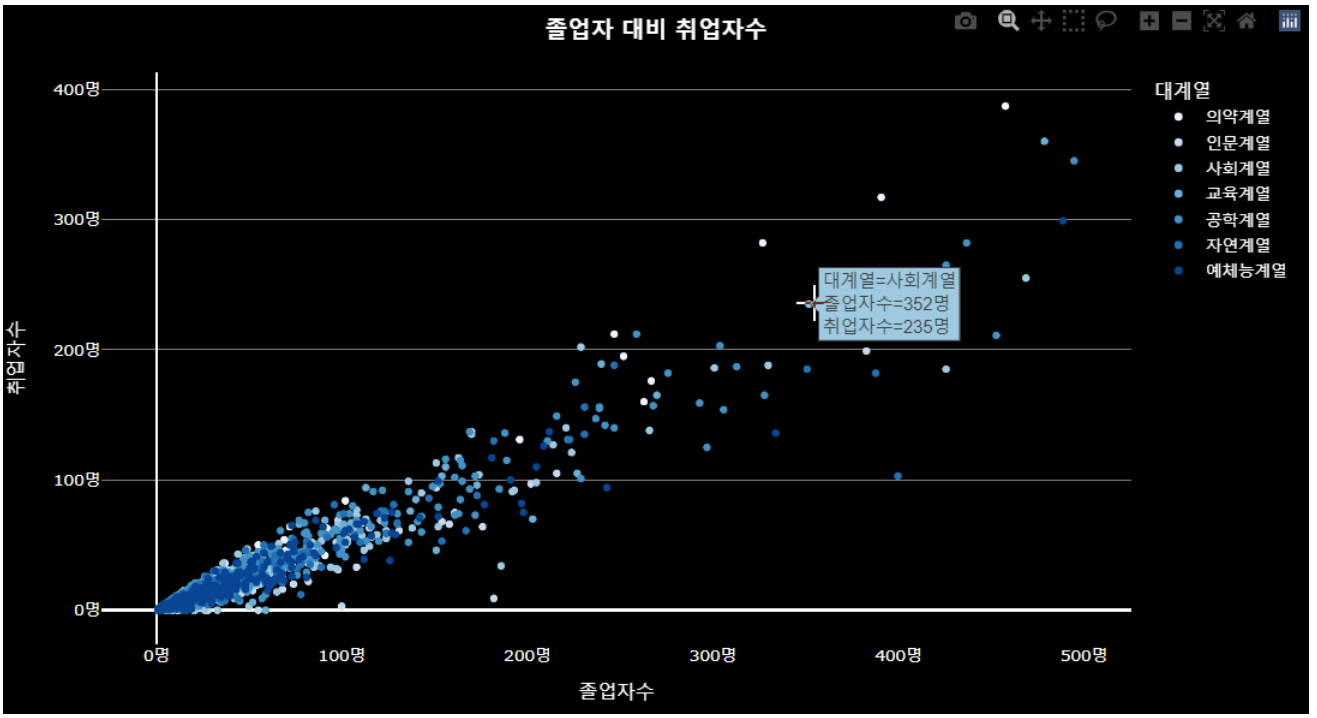
```
df_취업률_2000 |>
  plot_ly() |>
  add_markers(x = ~졸업자수, y = ~취업자수, color = ~대계열) |>
  layout(title = list(text = '<b>졸업자 대비 취업자수</b>', font = list(color = 'white')),
         margin = list(t = 50, b = 25, l = 25, r = 25),
         paper_bgcolor = 'black', plot_bgcolor = 'black',
         xaxis = list(color = 'white', ticksuffix = '명'),
         yaxis = list(color = 'white', gridcolor = 'gray', ticksuffix = '명', dtick = 100),
         legend = list(font = list(color = 'white')))
```



실행결과 2- 1. color 매핑 결과

- python : plotly.express.scatter()

python 에서는 `plotly.express.scatter()`로 산점도를 그릴 수 있다. 다만 `plotly.express.scatter()`에서 사용하는 매개변수는 `add_trace(plolty.graph_object.Scatter())`와 다소 다르다.



1.2. 추세선 그리기

이번에는 `plotly` 를 사용하여 직접 추세선을 그리는 과정을 알아보자. `plotly` 에서 추세선의 추가는 python 에서는 매우 간단하게 추가할 수 있지만 R 에서는 직접적으로 추세선을 그리는 기능을 제공하지 않는다.

- R

R 에서 추세선을 그리는데 가장 많이 사용하는 방법은 선형 회귀를 사용하는 방법이다. R 에서 많이 사용하는 `ggplot2` 에서는 `geom_smooth()`를 사용하여 간단히 추세선을 그릴 수 있지만 `plotly` 에서는 이런 기능을 제공하지 않는다. 따라서 선형회귀(lm)나 국소회귀(loess) 모델을 만들어 추세선을 그려야 한다.

이를 위해서는 X 축과 Y 축에 설정되는 독립변수와 종속변수 간의 모델을 R base 에서 제공하는 `lm()`과 `loess()`를 사용하여 만들어야 한다. 이렇게 만든 모델에 `fitted()`나 `predict()`를 사용하여 독립변수(X 축)에 대응하는 종속변수(Y 축)에 대한 추세선을 그려준다. 만약 신뢰구간(Confidence Interval, CI)의 표현이 필요하다면 `add_ribbons()`을 사용하여 그려줄 수 있다.

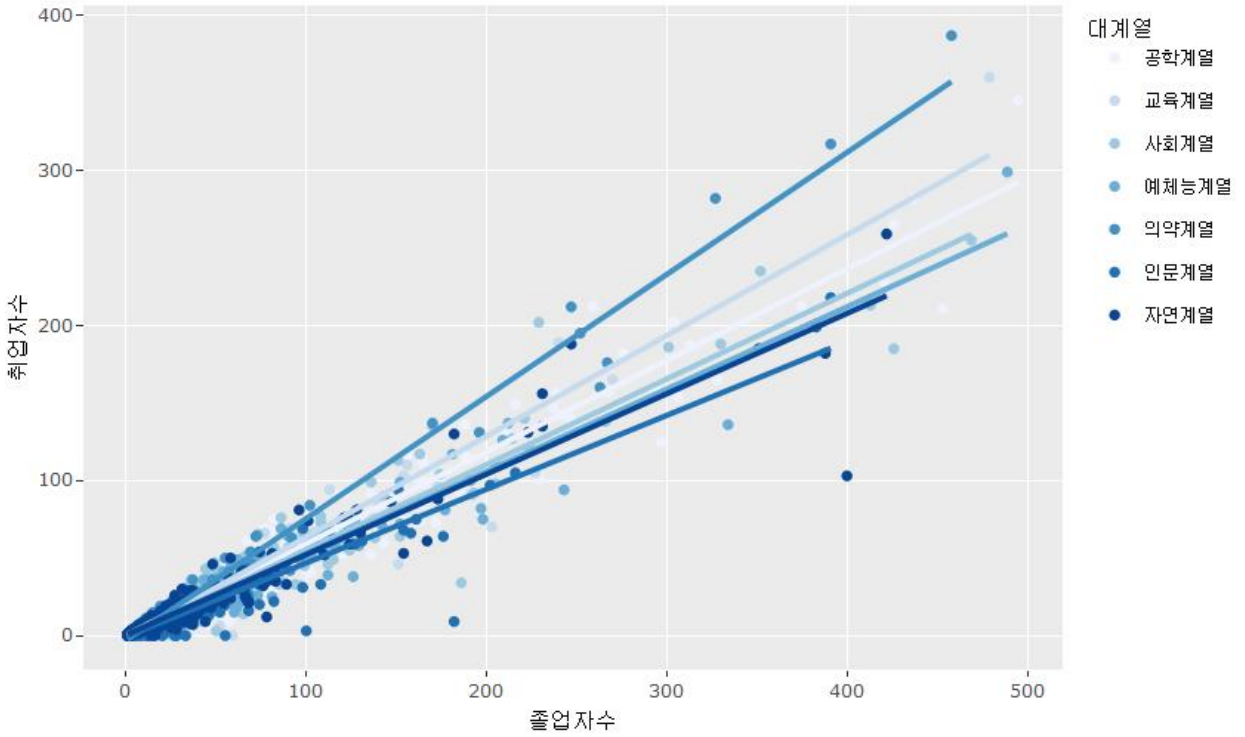
국소회귀 추세선을 `plotly` 로 그리는 방법은 앞서 설명한 바와 같이 `loess()`을 사용하여 선형회귀 모델을 만들고 이를 `fitted()`를 사용하여 해당 모델에 대한 적합치를 Y 축에

매핑함으로써 그려줄 수 있다. 다만 이 과정에서 X 축 변량의 순서대로 `fitted()` 값을 그려야 정상적인 추세선이 나타나기 때문에 이 데이터를 정렬하기 위해 임시 데이터프레임을 생성하여 사용하였다.

```
lm_trend <- lm(data = df_취업률_2000, 취업자수 ~ 졸업자수)

loess_trend <- loess(data = df_취업률_2000, 취업자수 ~ 졸업자수)

df_loess_trend <- data.frame(X = df_취업률_2000$졸업자수, Y = fitted(loess_trend)) |>
  arrange(X)

df_취업률_2000 |>
  plot_ly(type = 'scatter', mode = 'markers') |>
  add_trace(x = ~졸업자수, y = ~취업자수, showlegend = FALSE) |>
  add_trace(mode = 'lines', x = ~졸업자수, y = ~fitted(lm_trend), name = '선형 추세선') |>
  add_trace(data = df_loess_trend_emp, mode = 'lines', x = ~X, y = ~Y, name = 'loess')
```

하지만 추세선은 이렇게 전체적인 흐름을 보기위해서도 그리지만 많은 경우 세부 그룹별로 추세선을 그리는 경우도 많다. `plotly` 의 R 패키지에서 자체적으로 추세선을 지원하지 않기 때문에 세부 그룹별로 추세선을 그릴때는 `ggplot2` 로 그린후 `plotly` 로 변환하는 것이 훨씬 효율적이다.

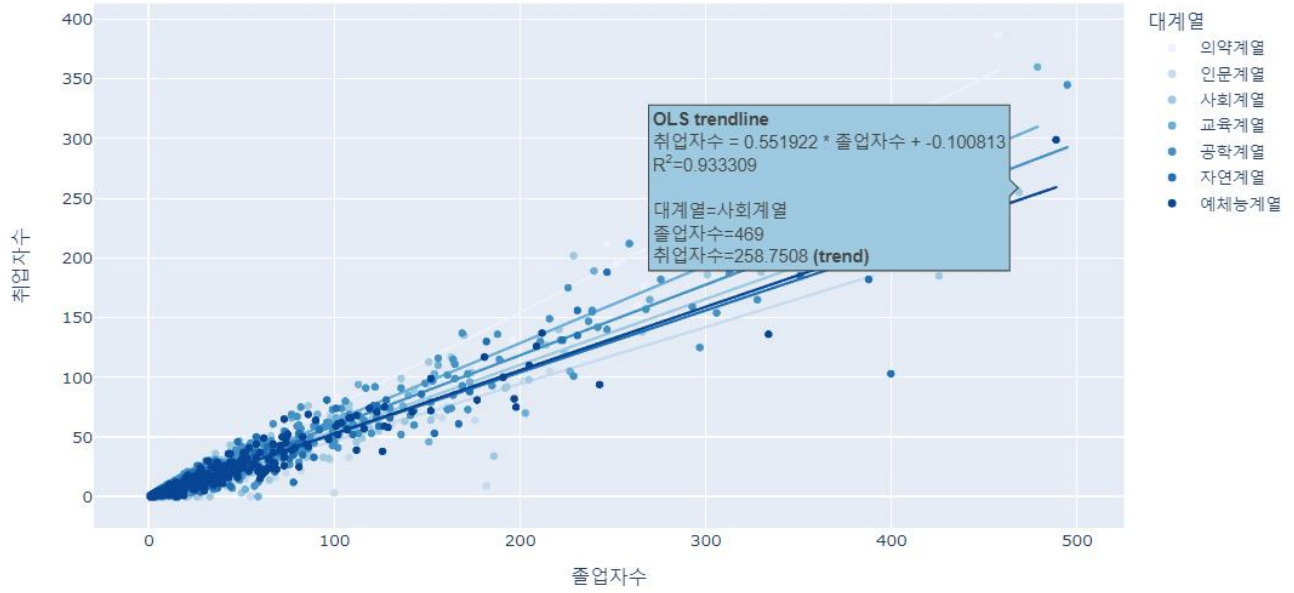
```
p <- df_취업률_2000 |>
 ggplot(aes(x = 졸업자수, y = 취업자수, color = 대계열)) +
 geom_point() +
 geom_smooth(method = 'lm', se = FALSE)

ggplotly(p)
```



- python

python 에서는 R 보다 훨씬 쉽게 추세선을 그릴 수 있다. `px.Scatter()`의 'trendline' 속성으로 간단히 그릴 수 있는데 'trendline' 속성에는 'ols'(선형회귀), 'lowess'(국소선형회귀), 'rolling'(이동평균) 등을 설정할 수 있다. 앞서 R 에서 그린 것과 같이 세부 그룹별로 추세선을 그리기 위해서는 'color' 속성으로 그룹화하고 이에 대해 'trendline'을 설정하면 간단히 그려진다.

```
fig = px.scatter(df_취업률_2000, x= '졸업자수', y="취업자수",
                 color = "대계열", trendline = 'ols')

fig.show()
```

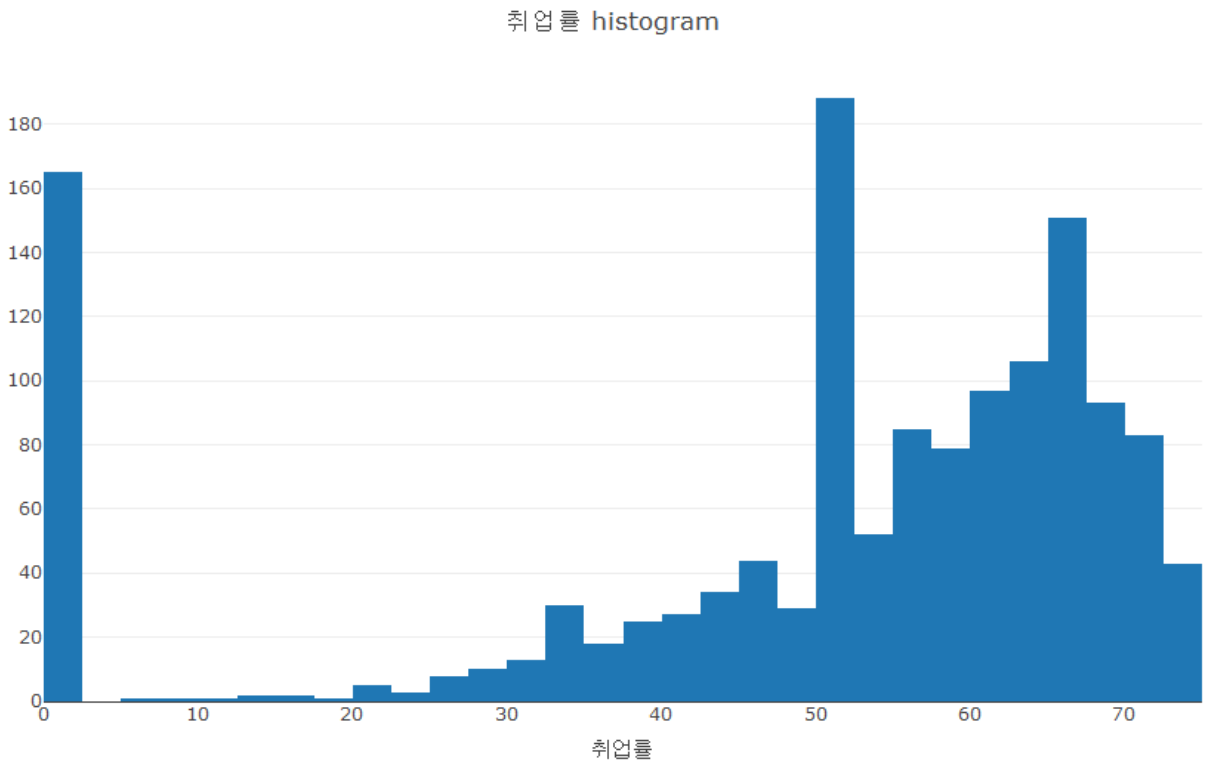


2. 히스토그램(histogram)

2.1. 기본 히스토그램

- R

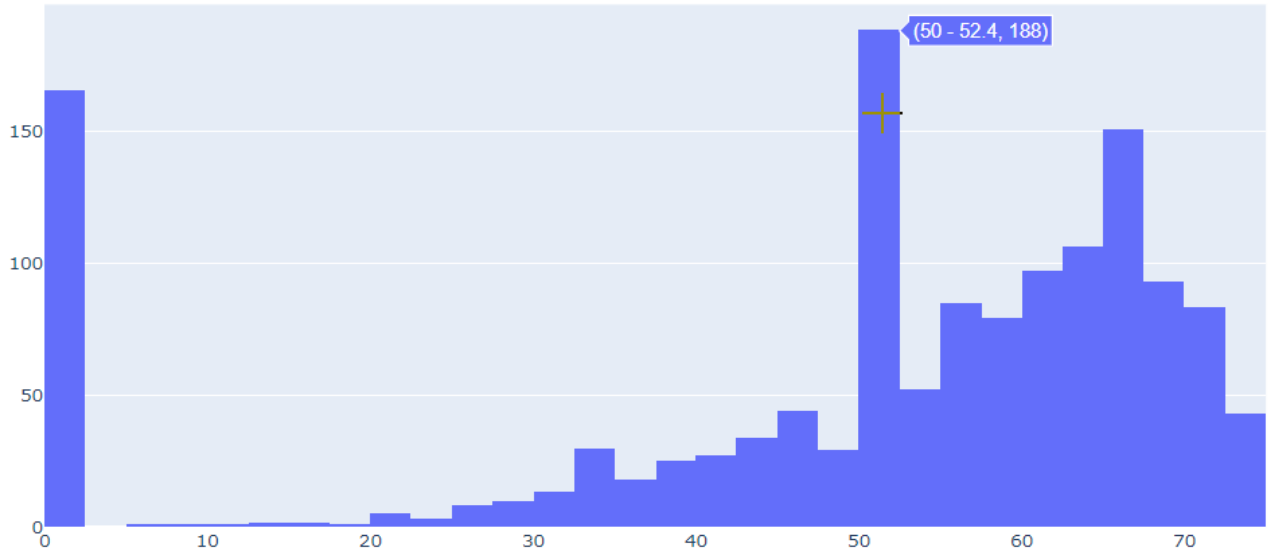
```
## 취업률 데이터를 사용해 plotly 객체 생성
p_histogram <- df_취업률_2000 |> plot_ly()

p_histogram |>
 ## histogram trace 로 X 축을 취업률로 매핑, name 을 취업률로 설정
 add_histogram(x = ~취업률, name = '취업률',
        xbins = list(start = 0, end = 75, size = 2.5)) |>
 ## 제목과 여백 설정
 layout(title = '취업률 histogram', margin = list(t = 50, b = 25, l = 25, r = 25))
```



실행결과2- 2. 히스토그램 trace 시각화

- python

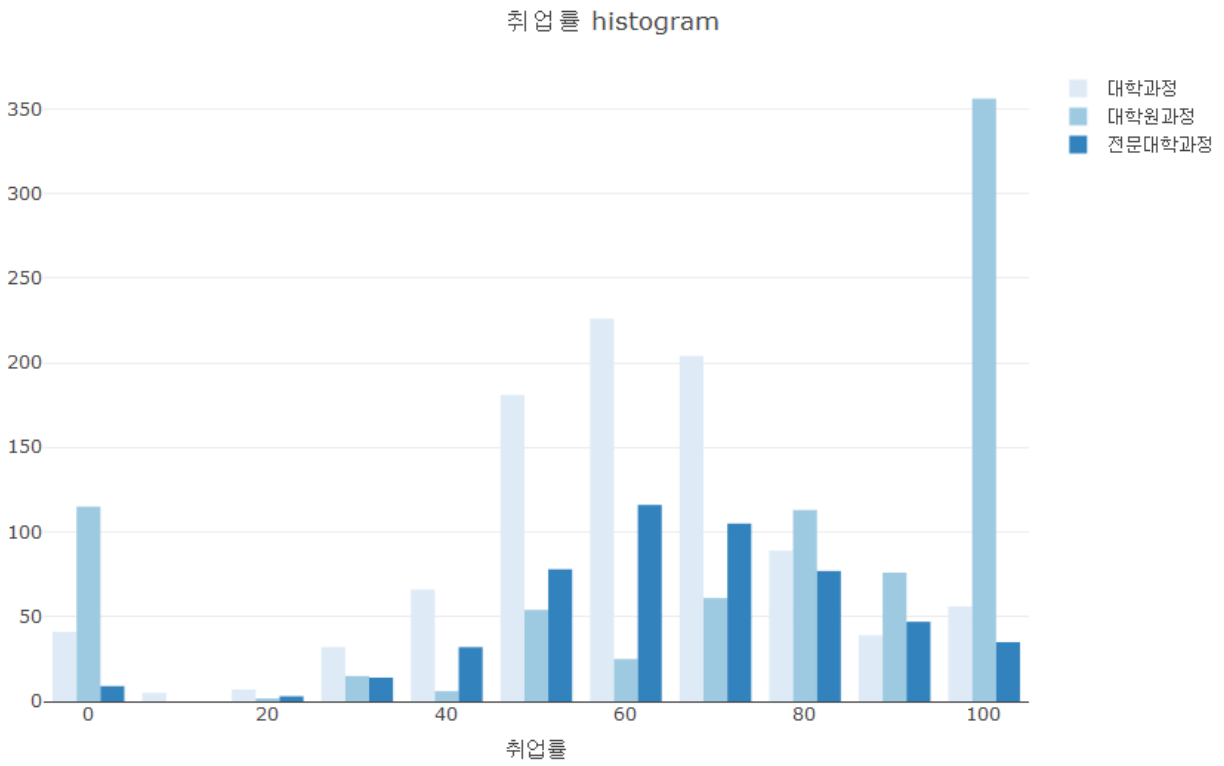
```
fig = go.Figure()

fig.add_trace(go.Histogram(x = df_취업률_2000['취업률'], name = '취업률',
            xbins = dict(start = 0, end = 75, size = 2.5)))

fig.show()
```



2.2. 다중 히스토그램

- R

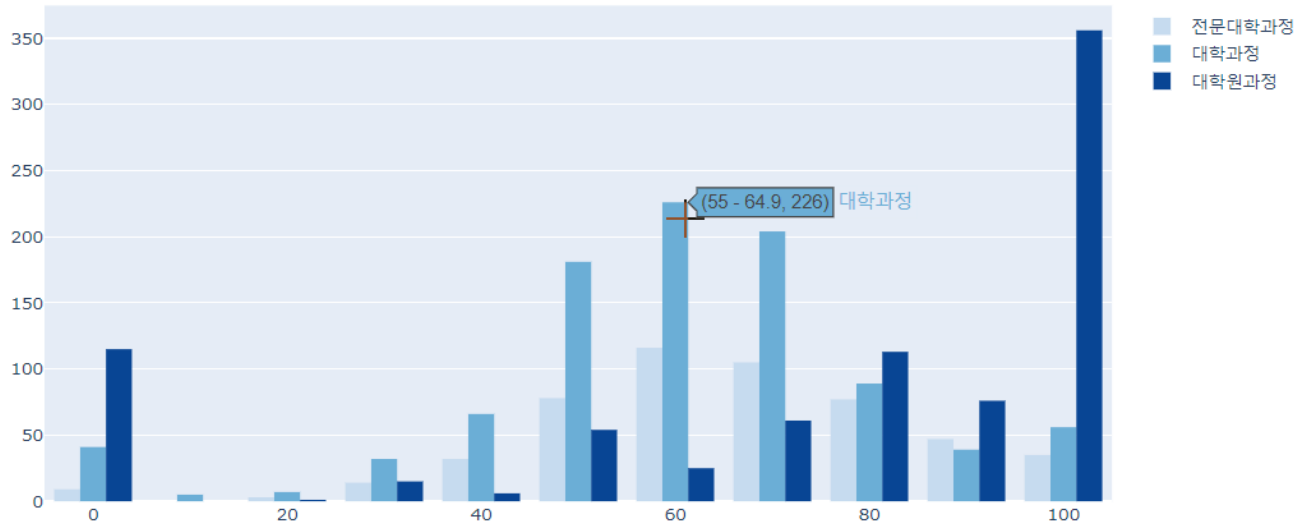
```
## 취업률 데이터를 사용해 plotly 객체 생성
p_histogram <- df_취업률_2000 |> plot_ly()

p_histogram |>
 ## histogram trace 로 X 축을 취업률로 매핑, name 을 취업률로 설정
 add_histogram(x = ~취업률, color = ~과정구분,
         xbins = list(size = 10)) |>
 ## 제목과 여백 설정
 layout(title = '취업률 histogram',
       margin = list(t = 50, b = 25, l = 25, r = 25)
)
```



실행결과2- 3. 히스토그램 trace 시각화

- python

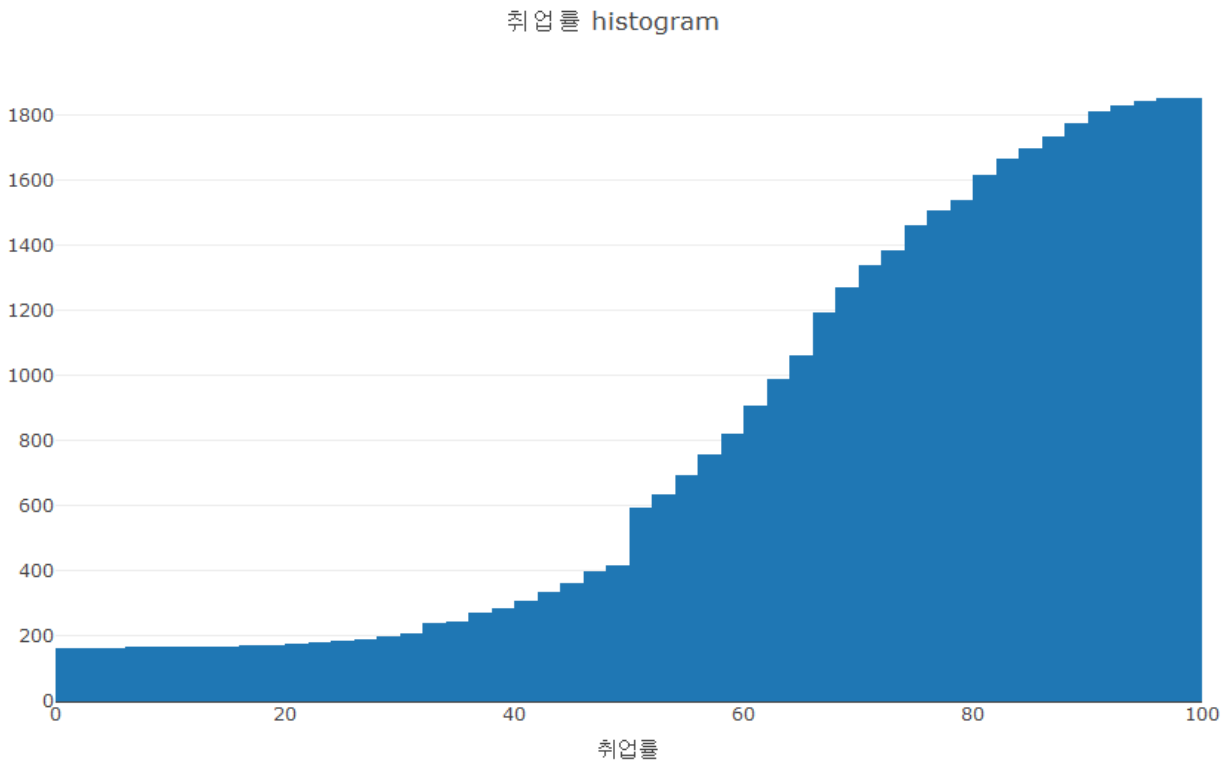
```
fig = go.Figure()

fig.add_trace(go.Histogram(x = df_취업률_2000.loc[df_취업률_2000['과정구분'] == '전문대학과정',
'취업률'],
                           name = "전문대학과정", xbins = dict(size = 10)
                          )
             )

fig.add_trace(go.Histogram(x = df_취업률_2000.loc[df_취업률_2000['과정구분'] == '대학과정',
'취업률'],
                           name = '대학과정', xbins = dict(size = 10)
                          )
             )

fig.add_trace(go.Histogram(x = df_취업률_2000.loc[df_취업률_2000['과정구분'] == '대학원과정',
'취업률'],
                           name = '대학원과정', xbins = dict(size = 10)
                          )
             )

fig.show()
```



2.3. 누적 히스토그램

- R

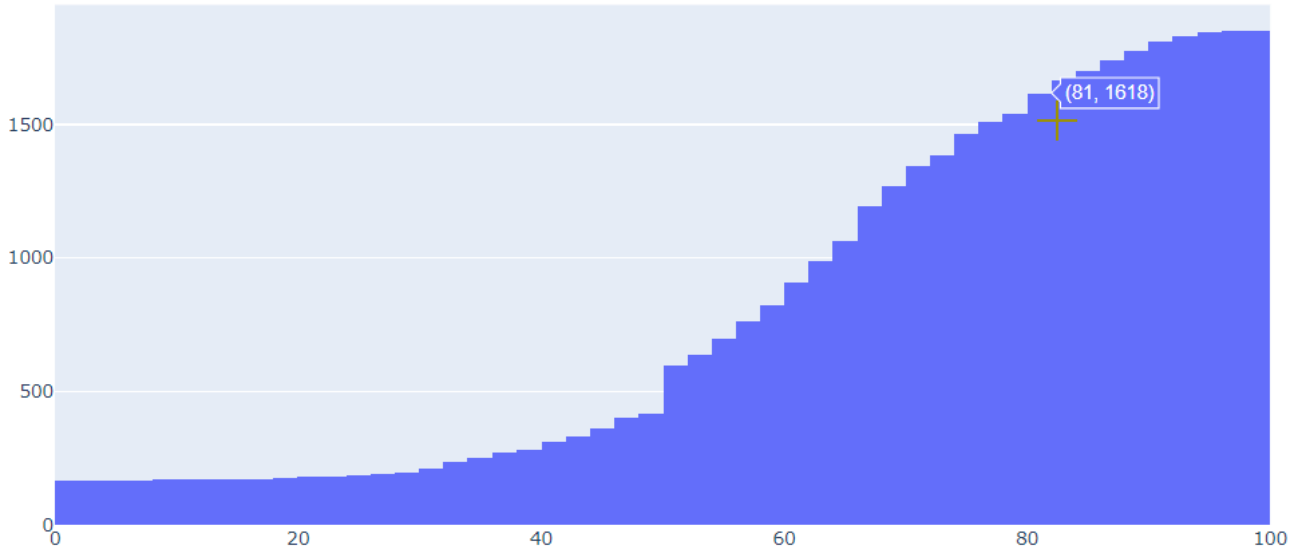
```
## 취업률 데이터를 사용해 plotly 객체 생성
p_histogram <- df_취업률_2000 |> plot_ly()

p_histogram |>
 ## histogram trace 로 X 축을 취업률로 매핑, name 을 취업률로 설정
 add_histogram(x = ~취업률, name = '취업률',
        xbins = list(start = 0, end = 100, size = 2),
        cumulative = list(enabled=TRUE)) |>
 ## 제목과 여백 설정
 layout(title = '취업률 histogram', margin = list(t = 50, b = 25, l = 25, r = 25))
```



실행결과 2- 4. 히스토그램 trace 시각화

- python

```
fig = go.Figure()

fig.add_trace(go.Histogram(x = df_취업률_2000['취업률'], name = '취업률',
                           xbins = dict(start = 0, end = 100, size = 2),
                           cumulative = dict(enabled = True)))

fig.show()
```

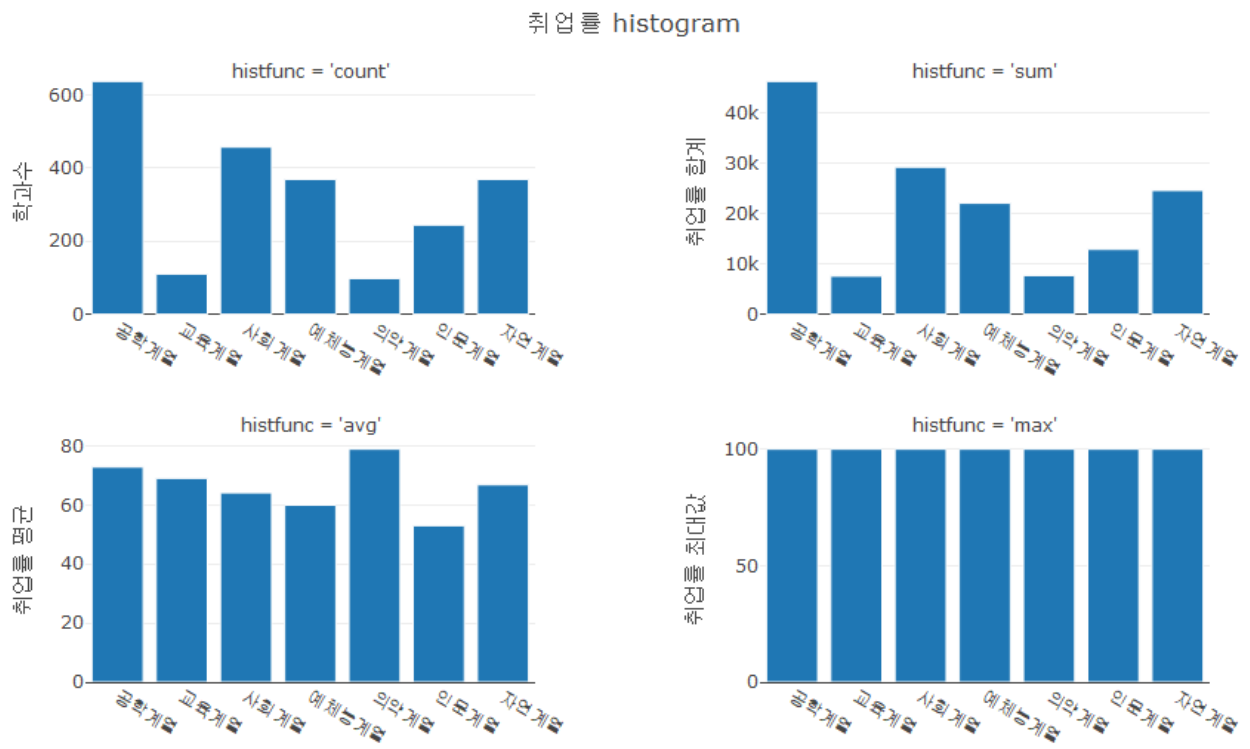


2.4. 히스토그램 함수

- R

```
p_histogram |>
 add_trace(type = 'histogram',  ## add_histogram()과 동의 함수
        x = ~대계열,
        ## stroke 를 흰색으로, 히그토그램 막대 값을 'count'로 설정
        stroke = I('white'), histfunc = 'count') |>
 layout(title = '취업률 histogram', margin = margins,
      yaxis = list(title = list(text = '학과수')))

p_histogram |>
 add_trace(type = 'histogram', x = ~대계열, y = ~as.character(취업률),
        ## stroke 를 흰색으로, 히그토그램 막대 값을 'sum'으로 설정
        histfunc = 'sum', stroke = I('white')) |>
 ## Y 축을 선형으로 설정
 layout(yaxis=list(type='linear',
              title = list(text = '취업률 합계')),
      title = '취업률 histogram', margin = margins)

p_histogram |>
 add_trace(type = 'histogram', x = ~대계열, y = ~as.character(취업률),
        ## stroke 를 흰색으로, 히그토그램 막대 값을 'average'로 설정
        histfunc = 'avg', stroke = I('white')) |>
 ## Y 축을 선형으로 설정
 layout(yaxis=list(type='linear',
              title = list(text = '취업률 평균')),
      title = '취업률 histogram', margin = margins)

p_histogram |>
 add_trace(type = 'histogram', x = ~대계열, y = ~as.character(취업률),
```

```
        ## stroke 를 흰색으로, 히그토그램 막대 값을 'max'로 설정
        histfunc = 'max', stroke = I('white')) |>
## Y 축을 선형으로 설정
layout(yaxis=list(type='linear', title = list(text = '취업률 최대값')),
    title = '취업률 histogram', margin = list(t = 50, b = 25, l = 25, r = 25))
```

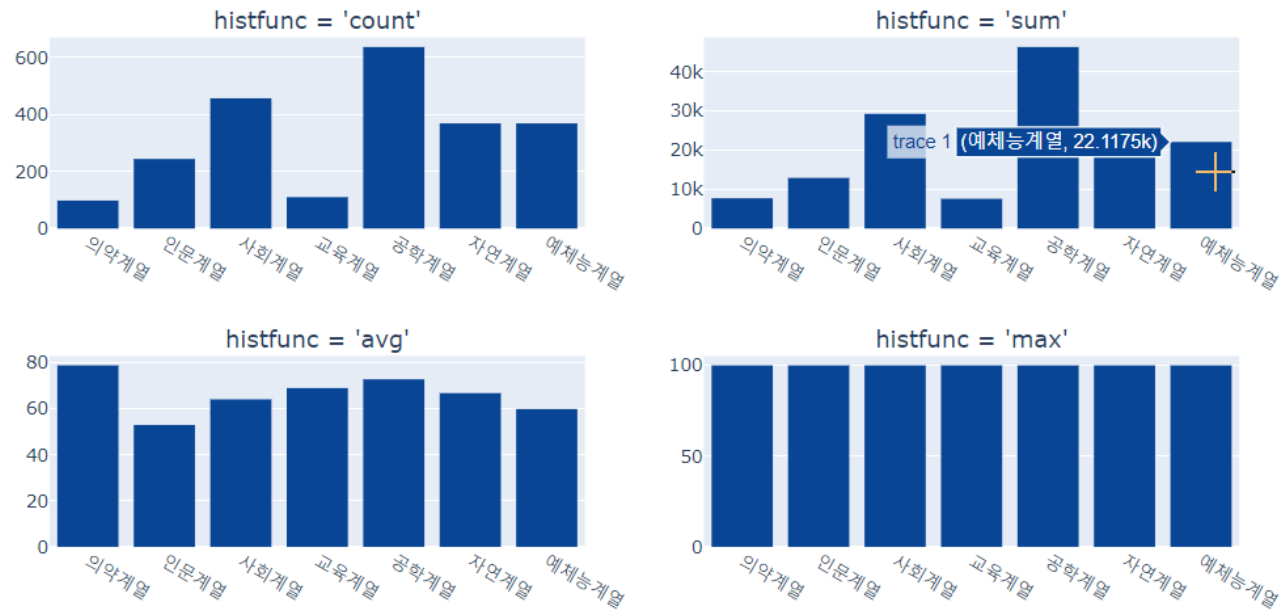


실행결과 2- 5. histfunc 에 따른 히스토그램 결과

- python

```
#################
fig = go.Figure()
fig.add_trace(go.Histogram(x = df_취업률_2000['대계열'], y = df_취업률_2000['취업률'],
                           histfunc = 'count', showlegend = False))
fig.show()

#################
fig = go.Figure()
fig.add_trace(go.Histogram(x = df_취업률_2000['대계열'], y = df_취업률_2000['취업률'],
                           histfunc = 'sum', showlegend = False))
fig.show()

#################
fig = go.Figure()
fig.add_trace(go.Histogram(x = df_취업률_2000['대계열'], y = df_취업률_2000['취업률'],
```

```
                            histfunc = 'avg', showlegend = False))
fig.show()

#################
fig = go.Figure()
fig.add_trace(go.Histogram(x = df_취업률_2000['대계열'], y = df_취업률_2000['취업률'],
                            histfunc = 'max', showlegend = False))
fig.show()
```



3. 박스(Box) 플롯

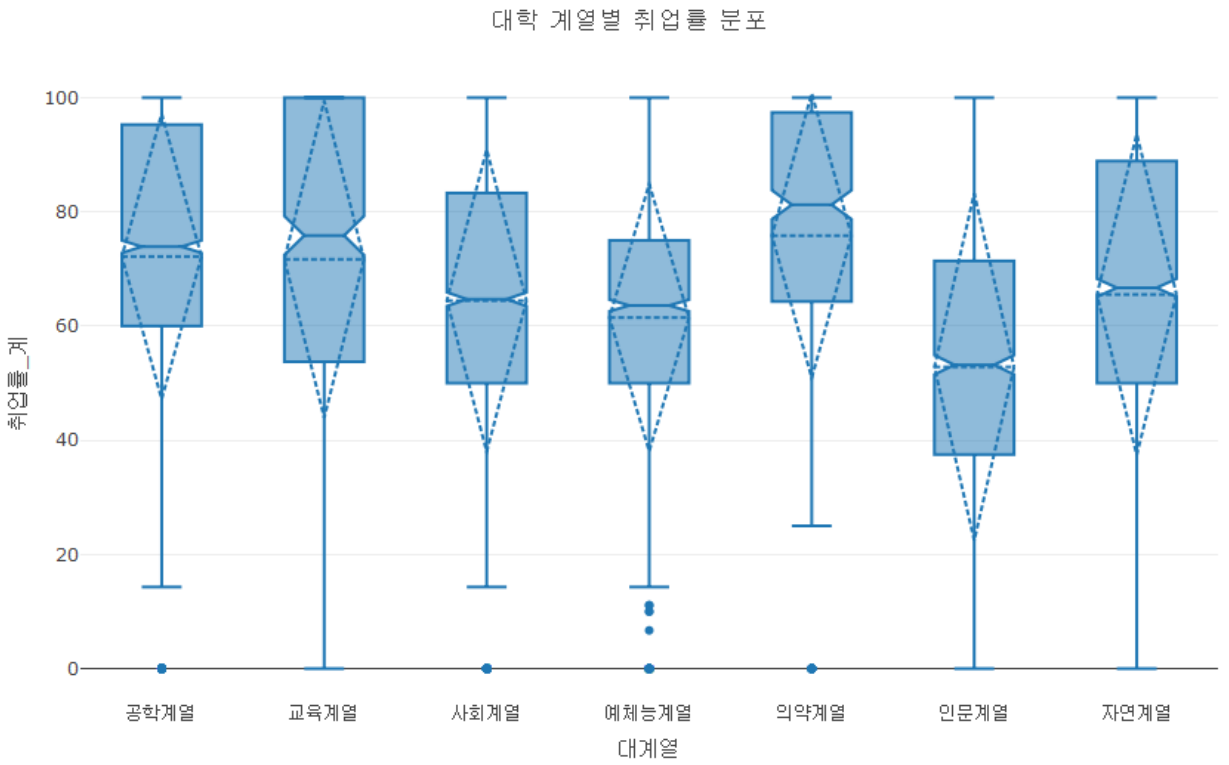
박스 trace 는 박스 플롯을 생성하기 위해 사용되는 trace 이다. 앞 장에서 설명했듯이 박스 플롯은 데이터의 전체적 분포를 4 분위수(quantile)과 IQR(Inter Quartile Range)를 사용하여 표시하는 시각화로 연속형 변수와 이산형 변수의 시각화에 사용되는 방법이다. 박스 trace 를 사용해 박스 플롯을 생성하기 위해서는 `add_trace(type = 'box')`를 사용하거나 `add_boxplot()`을 사용한다.

3.1. 평균, 표준편차가 포함된 박스 플롯

- R

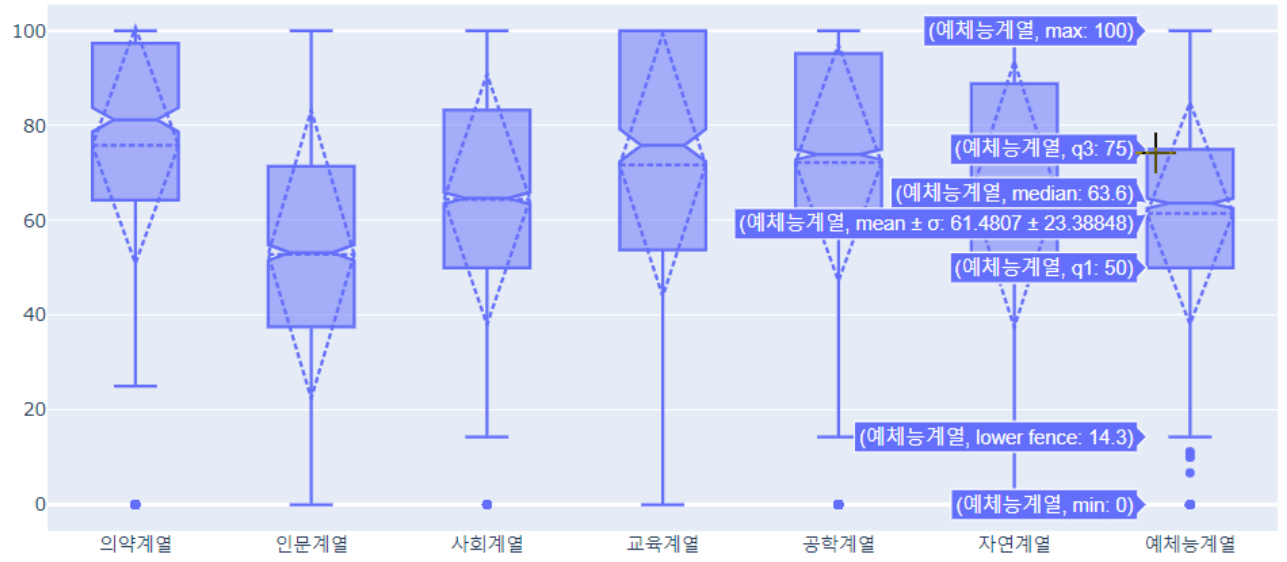
```
df_취업률 |>
  plot_ly() |>
  add_boxplot(x = ~대계열, y = ~취업률_계, boxmean = 'sd', notched = TRUE)|>
```

```
## boxmode 를 group 으로 설정
layout(title = list(text = '대학 계열별 취업률 분포'),
    margin = list(t = 50, b = 25, l = 25, r = 25))
```



- Python

```
fig = go.Figure()
fig.add_trace(go.Box(
    x = df_취업률['대계열'], y = df_취업률['취업률_계'],
    boxmean = 'sd', notched = True))
```



3.2. 그룹 박스 플롯

박스 trace 도 막대 trace 와 같이 그룹화한 변수를 `color` 나 `linetype` 등의 속성에 매핑시켜서 다수의 박스 trace 를 한번에 만들거나 각각의 박스 trace 를 추가하여 그릴 수 있다. 이렇게 여러개의 박스 trace 를 그릴 때 그 구성 형태를 결정하기 위해서는 차후 설명할 `layout()`의 `boxmode` 속성을 설정함으로써 가능하다.[2]

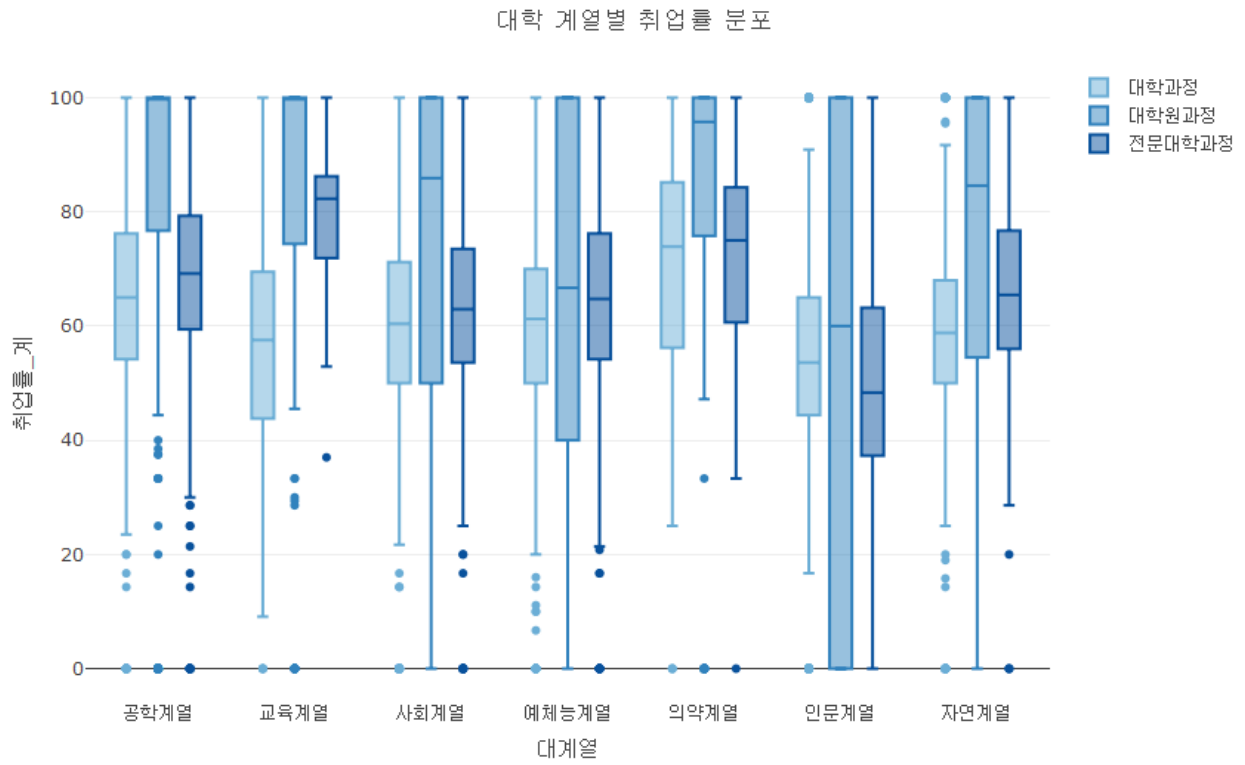
`boxmode` 속성은 'group'과 'overlay'의 두 가지를 설정할 수 있다. 'group'은 각각의 박스들이 옆으로 배치되면서 전체 박스 플롯이 완성되고 'overlay'는 각각의 박스들이 겹쳐져 그려지면서 완성된다. 다음의 코드는 `color` 로 과정구분을 매핑하여 'group'형 박스 플롯을 생성하는 코드이다.

- R

```
df_취업률 |>
 plot_ly() |>
 add_boxplot(x = ~대계열, y = ~취업률_계,
        ## color 를 과정구분으로 매핑
        color = ~과정구분)|>
 ## boxmode 를 group 으로 설정
```

[2] boxmode 를 사용할 때 'Warning message'를 내는 경우가 있는데 이는 Plotly Community Forum 에서도 적절치 않은 경고 메세지로 지적되고 있는데 무시해도 되는 메세지이다.

```
layout(boxmode = "group", title = list(text = '대학 계열별 취업률 분포'),
    margin = list(t = 50, b = 25, l = 25, r = 25))
```

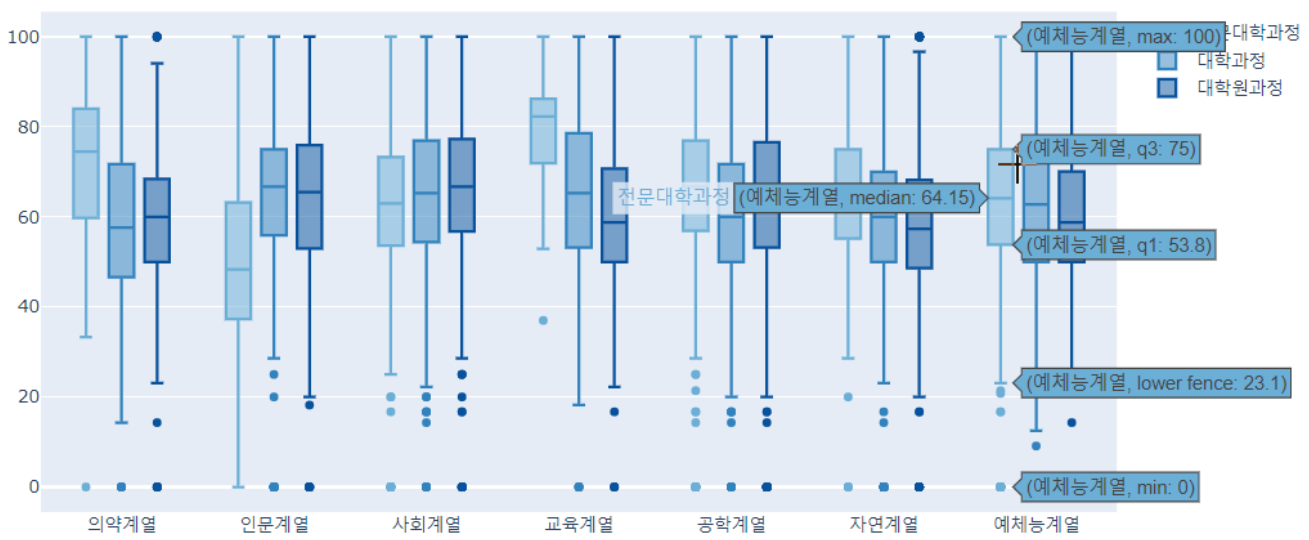


실행결과 2- 6. 박스 trace 의 group mode 실행 결과

- Python

```
fig = go.Figure()
fig.add_trace(go.Box(
    x = df_취업률.loc[df_취업률['과정구분'] == '전문대학과정', '대계열'], y =
df_취업률['취업률_계'],
    name = '전문대학과정')
            )

fig.add_trace(go.Box(
    x = df_취업률.loc[df_취업률['과정구분'] == '대학과정', '대계열'], y = df_취업률['취업률_계'],
    name = '대학과정')
            )

fig.add_trace(go.Box(
    x = df_취업률.loc[df_취업률['과정구분'] == '대학원과정', '대계열'], y =
df_취업률['취업률_계'],
    name = '대학원과정')
            )
```

```
fig.update_layout(boxmode = 'group',
                  title = dict(text = '학위과정별 취업률 분포', x = 0.5, xanchor = 'center')
                 )
```

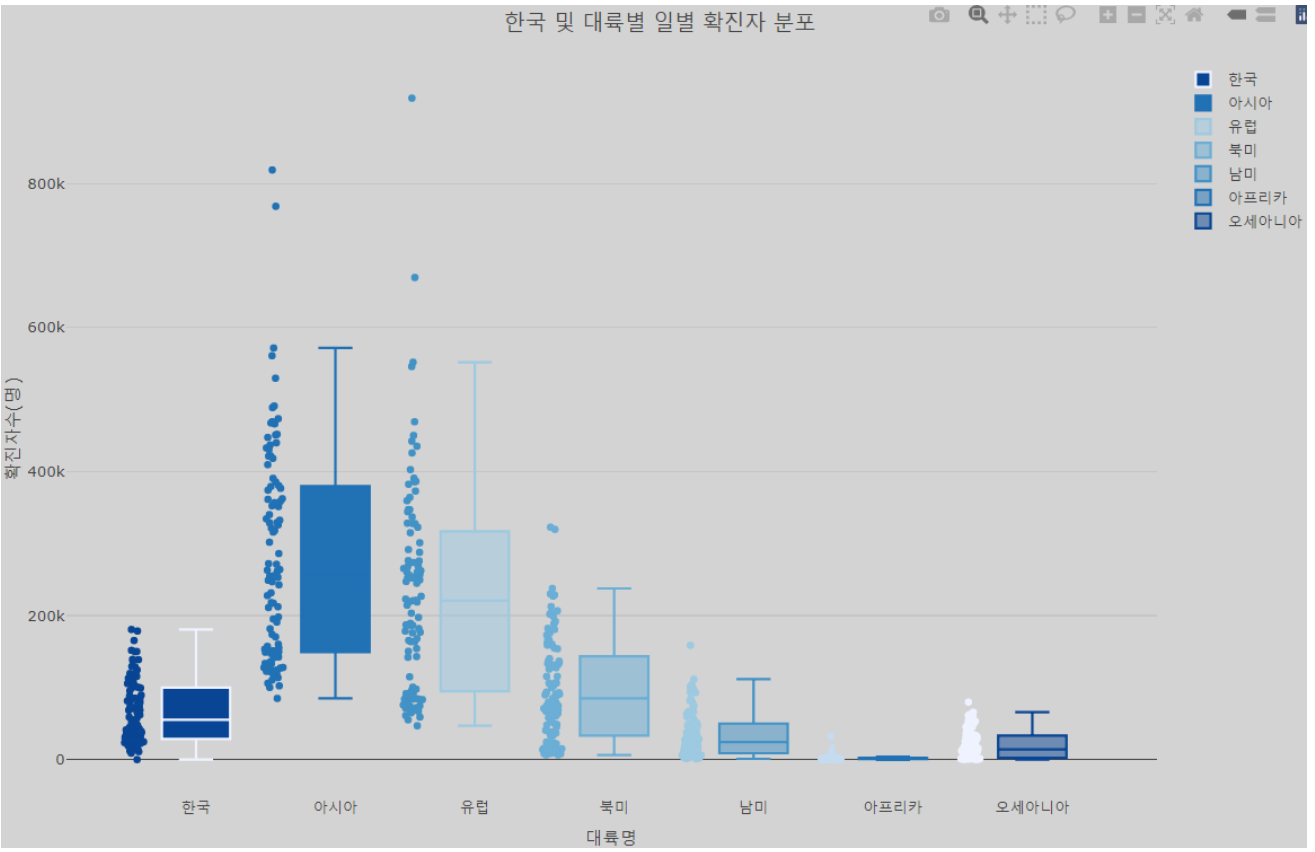


3.3. 지터(jitter) 박스 플롯

- R

```
fig <- df_covid19_100_wide |> plot_ly()

fig <- fig |>
 add_boxplot(y = ~확진자_한국, name = '한국', boxpoints = "all", jitter = 0.3,
     pointpos = -1.8)

fig <- fig |>
 add_boxplot(y = ~확진자_아시아, name = '아시아', boxpoints = "all", jitter = 0.3,
     pointpos = -1.8)

fig <- fig |>
 add_boxplot(y = ~확진자_유럽, name = '유럽', boxpoints = "all", jitter = 0.3,
     pointpos = -1.8)

fig <- fig |>
 add_boxplot(y = ~확진자_북미, name = '북미', boxpoints = "all", jitter = 0.3,
     pointpos = -1.8)

fig <- fig |>
 add_boxplot(y = ~확진자_남미, name = '남미', boxpoints = "all", jitter = 0.3,
     pointpos = -1.8)
```

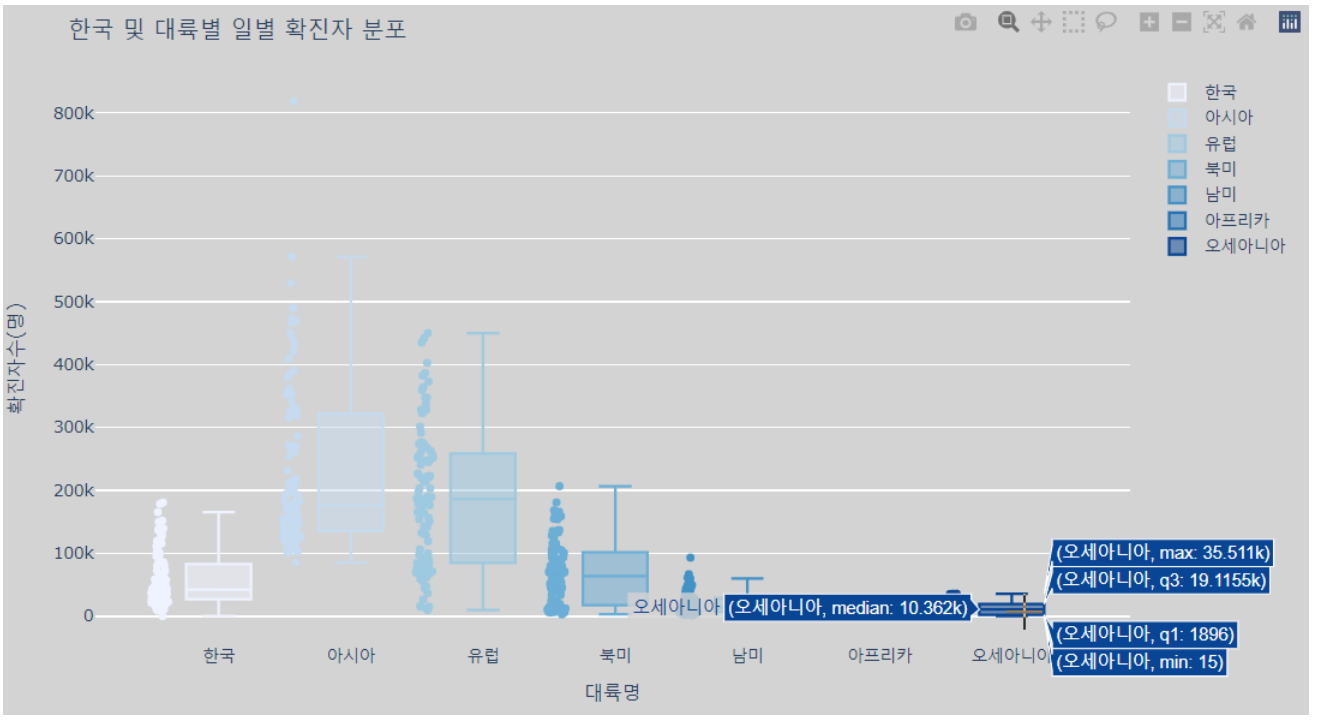
```
fig <- fig |>
 add_boxplot(y = ~확진자_아프리카, name = '아프리카', boxpoints = "all", jitter = 0.3,
     pointpos = -1.8)

fig <- fig |>
 add_boxplot(y = ~확진자_오세아니아, name = '오세아니아', boxpoints = "all", jitter = 0.3,
     pointpos = -1.8)

 ## boxmode 를 group 으로 설정
fig |>  layout(title = list(text = '한국 및 대륙별 일별 확진자 분포'),
          xaxis = list(title = '대륙명'),
          yaxis = list(title = '확진자수(명)'),
       margin = list(t = 50, b = 25, l = 25, r = 25),
       paper_bgcolor='lightgray', plot_bgcolor='lightgray')
```



- python

```
fig = go.Figure()
fig.add_trace(go.Box(
  y = df_covid19_100_wide['확진자_한국'], name = '한국',
  boxpoints = "all", jitter = 0.3, pointpos = -1.8))

fig.add_trace(go.Box(
  y = df_covid19_100_wide['확진자_아시아'], name = '아시아',
```

```
  boxpoints = "all", jitter = 0.3, pointpos = -1.8))

fig.add_trace(go.Box(
  y = df_covid19_100_wide['확진자_유럽'], name = '유럽',
  boxpoints = "all", jitter = 0.3, pointpos = -1.8))

fig.add_trace(go.Box(
  y = df_covid19_100_wide['확진자_북미'], name = '북미',
  boxpoints = "all", jitter = 0.3, pointpos = -1.8))

fig.add_trace(go.Box(
  y = df_covid19_100_wide['확진자_남미'], name = '남미',
  boxpoints = "all", jitter = 0.3, pointpos = -1.8))

fig.add_trace(go.Box(
  y = df_covid19_100_wide['확진자_아프리카'], name = '아프리카',
  boxpoints = "all", jitter = 0.3, pointpos = -1.8))

fig.add_trace(go.Box(
  y = df_covid19_100_wide['확진자_오세아니아'], name = '오세아니아',
  boxpoints = "all", jitter = 0.3, pointpos = -1.8))

fig.update_layout(title = dict(text = '한국 및 대륙별 일별 확진자 분포', x = 0.5),
                  xaxis = dict(title = '대륙명'),
                  yaxis = dict(title = '확진자수(명)'),
                  margin = dict(t = 50, b = 25, l = 25, r = 25),
                  paper_bgcolor='lightgray', plot_bgcolor='lightgray')
```



4. 바이올린(Violin) 플롯

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

바이올린 trace 를 사용해 바이올린 플롯을 생성하기 위해서는 `add_trace(type = 'violin')`를 사용하여야하고 래핑함수를 제공하지 않는다.

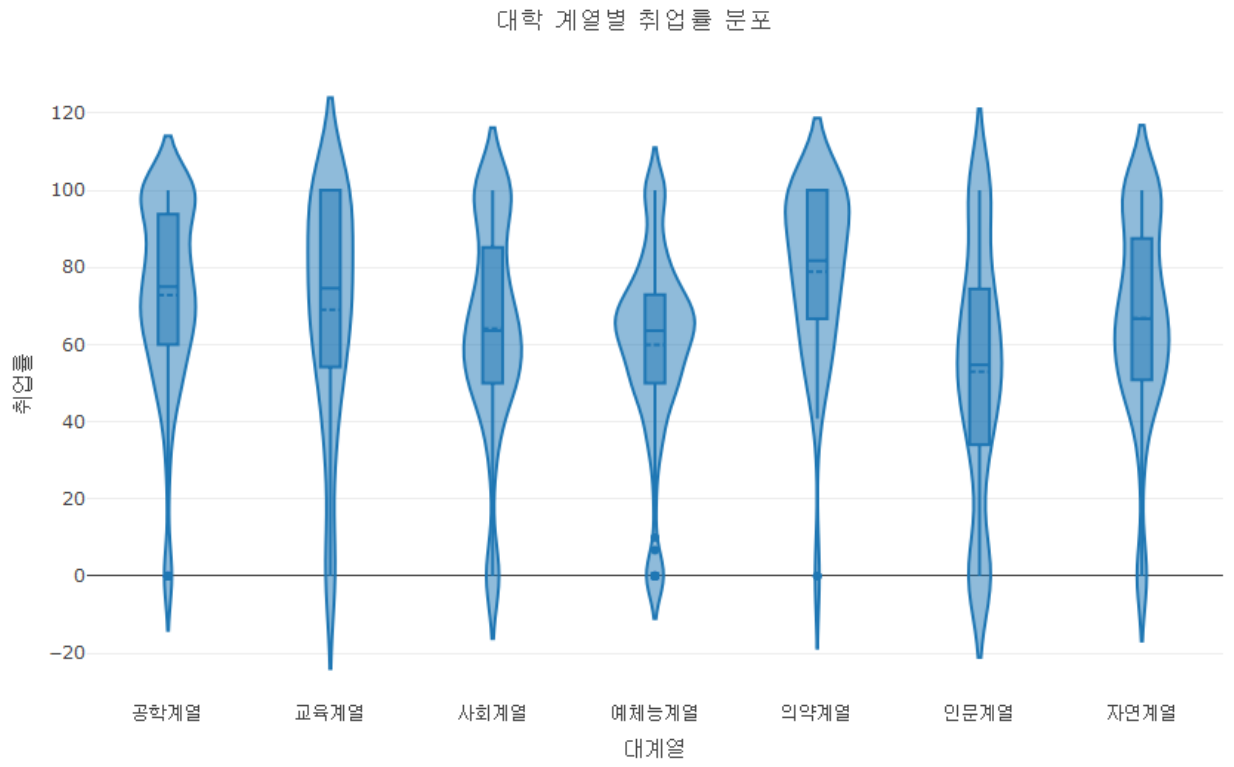
add_trace(p, type = 'violin', ..., data = NULL, inherit = TRUE)

- p : plot_ly()로 생성한 plotly 객체
- type : trace 타입을 'violin'로 설정
- ... : 바이올린 trace 의 line 모드에 설정할 수 있는 속성 설정
- data : 시각화할 데이터프레임
- inherit : plot_ly()에 설정된 속성 type 을 상속할지를 결정하는 논리값

4.1. 박스 플롯이 포함된 바이올린 플롯

- R

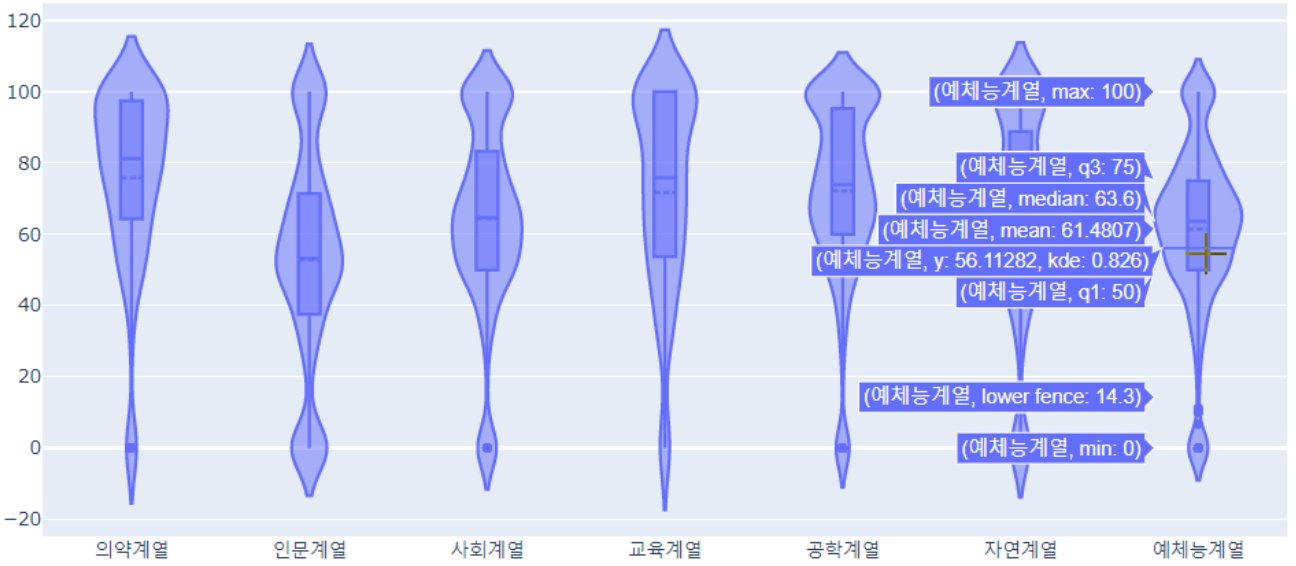
```
df_취업률_2000 |>
 plot_ly() |>
 ## 바이올린 trace 추가
 add_trace(type = 'violin', x = ~대계열, y = ~취업률,
        box = list(visible = T),
        meanline = list(visible = T)) |>
 layout(title = list(text = '대학 계열별 취업률 분포'),
      margin = list(t = 50, b = 25, l = 25, r = 25))
```



실행결과 2- 7. 바이올린 trace 생성

- python

```
fig = go.Figure()
fig.add_trace(go.Violin(
    x = df_취업률['대계열'], y = df_취업률['취업률_계'],
    box = dict(visible = True),
    meanline = dict(visible = True)
))
```



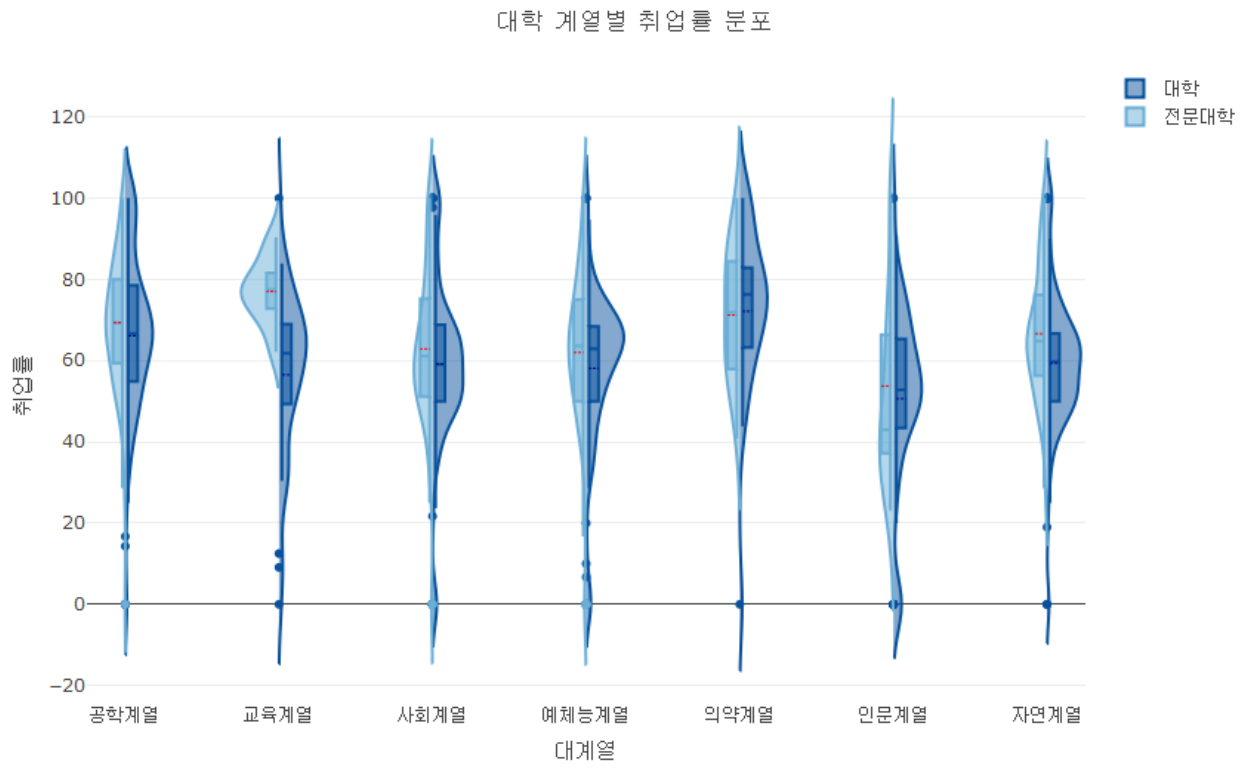
4.2. 분리된 바이올린 플롯

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

이렇게 두개의 바이올린 플롯을 반씩 잘라 붙이는 속성이 `side` 이다. `side` 는 바이올린의 양쪽을 다 사용하는 'both', 왼쪽 부분을 사용하는 'negative', 오른쪽 부분을 사용하는 'positive'를 설정할 수 있다. 그리고 이 두 바이올린을 붙이기 위해 `layout()`의 `violinmode` 를 `overlay` 로 설정한다. 여기에 앞서 설정한 `box` 와 `meanline` 설정하면 박스 trace 와 평균선도 반으로 그려서 붙여줄 수 있다. 앞서 그렸던 대학과 전문대학의 계열별 바이올린 플롯을 붙이는 코드는 다음과 같다.

```
p_violin |>
 ## 대학과정을 필터링한 데이터 설정
 add_trace(data = df_취업률_2000 |> filter(과정구분 == '대학과정'),
        ## 바이올린 trace 로 추가
        type = 'violin', x = ~대계열, y = ~취업률, name = '대학',
        ## side, box 의 설정
        side = 'positive', box = list(visible = TRUE, width = 0.5),
        ## meanline 의 속성 설정
        meanline = list(visible = TRUE, color = 'red', width = 1)) |>
 ## 전문대학과정을 필터링한 데이터 설정
 add_trace(data = df_취업률_2000 |> filter(과정구분 == '전문대학과정'),
        type = 'violin', x = ~대계열, y = ~취업률, name = '전문대학',
        side = 'negative', box = list(visible = TRUE, width = 0.5),
```

```
        meanline = list(visible = TRUE, color = 'red', width = 1)) |>
 layout(violinmode = "overlay",
     title = list(text = '대학 계열별 취업률 분포'),
     margin = margins)
```



실행결과 2- 8. 바이올린 trace 와 박스 trace 의 side 병합

- python

```
fig = go.Figure()
fig.add_trace(go.Violin(
    x = df_취업률.loc[df_취업률['과정구분'] == '전문대학과정', '대계열'], y =
df_취업률['취업률_계'],
    name = '전문대학',
    side = 'positive', box = dict(visible = True, width = 0.5),
    meanline = dict(visible = True, color = 'red', width = 1)
))

fig.add_trace(go.Violin(
    x = df_취업률.loc[df_취업률['과정구분'] == '대학과정', '대계열'], y = df_취업률['취업률_계'],
    name = '대학',
    side = 'negative', box = dict(visible = True, width = 0.5),
    meanline = dict(visible = True, color = 'red', width = 1)
))
fig.update_layout(title = dict(text = '대학 계열별 취업률 분포', x = 0.5),
         margin = dict(t = 50, b = 25, l = 25, r = 25))
```

```
fig = go.Figure()
fig.add_trace(go.Violin(
    x = df_취업률.loc[df_취업률['과정구분'] == '전문대학과정', '대계열'], y =
df_취업률['취업률_계'],
    name = '전문대학',
    side = 'positive', box = dict(visible = True, width = 0.5),
    meanline = dict(visible = True, color = 'red', width = 1)
))

fig.add_trace(go.Violin(
    x = df_취업률.loc[df_취업률['과정구분'] == '대학과정', '대계열'], y = df_취업률['취업률_계'],
    name = '대학',
    side = 'negative', box = dict(visible = True, width = 0.5),
    meanline = dict(visible = True, color = 'red', width = 1)
))
fig.update_layout(title = dict(text = '대학 계열별 취업률 분포', x = 0.5),
         margin = dict(t = 50, b = 25, l = 25, r = 25),
                 colorway = ('#08519C', '#6BAED6'))
```