

# 오픈소스 **SW** 기여

## 최종보고서

32182775 위성준  
32183698 이현기


## 1. 프로젝트 배경 및 목표

해당 프로젝트의 주제를 선정한 배경은 적당한 오픈소스SW에 기여하는 방식을 찾던 중에 **Gradio**라는 오픈소스 파이썬 라이브러리를 발견했고, **Gradio**에 학생 수준에서 기여할 수 있는 방법을 고민하던 중에 **Demo** 제작을 하는 방식을 선택하게 되었다.

프로젝트의 목표는 **Gradio** 홈페이지에 나와있는 **Demo**와 다른 종류의 **Demo**를 제작하여 **Gradio** 홈페이지에 제작한 **Demo**를 게시하는 것으로 잡았다.

제작한 **Demo**를 **Gradio** 홈페이지에 공개 시에 기대효과는 홈페이지에 제시된 **Demo**보다 다양하게 **Demo**를 구성할 수 있다는 것을 보여줌으로써 **Gradio** 라이브러리를 사용해보려고 고민하는 사용자들에게 정보를 주어 도움이 될 수 있을 것으로 생각하고 있다.

**Gradio**와 비슷한 라이브러리에는 **Streamlit**이 있다. **Streamlit**은 데이터 스크립트를 웹앱 형태로 바꿀 수 있게 도와주는 파이썬 라이브러리이다. 여기에서는 이 라이브러리를 사용할 수 있는 방법이 훨씬 다양하게 제시되어 있어 사용자로 하여금 라이브러리 사용에 도움을 주는 사례가 있다.

 [Cloud](#) [Gallery](#) [Components](#) [Community](#) [Docs](#) [Blog](#) [Sign in](#) [Sign up](#)

# Gallery

Below are some awesome templates, and community apps curated from [our forums](#) or [Twitter](#). Try them out, browse their source code, share with the world, and get inspired for your own projects!

CATEGORIES

**Featured**

Science & technology

NLP & language

Computer vision & images


Finance & business

Data visualization

Geography & society

Sports & fun

Education

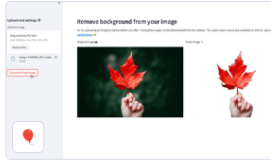


**Arup Social Data**

This is a repository for collection and analysis of open social data. This type of data can be useful to planners, NGO's,

by Jared Stock

[View source code →](#)

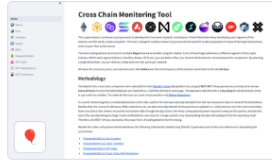


**Background Remover**

A simple Streamlit app that removes the background on an uploaded image and lets you download the result.

by Tyler Simons

[View source code →](#)



**Cross Chain Monitoring Tool**

This app lets you compare +10 blockchains in different sectors and view the performance of each blockchain.

by Ali Taslimi

[View source code →](#)

## 2. 요구사항

날씨를 알고 싶은 날짜와 날씨 요소를 입력하면 해당 날짜의 날씨 요소들을 표 형태로 수치를 보여줌과 동시에 그래프 형태로도 출력하여 시각적으로 확인할 수 있는 프로그램 모델의 Demo 형태를 제작했다.

Demo에 필요한 기상 자료는 기상자료개방포털에서 가져오고, 가져온 데이터를 활용하여 출력으로 표현될 표와 그래프는 **pandas**와 **matplotlib**을 이용했다.

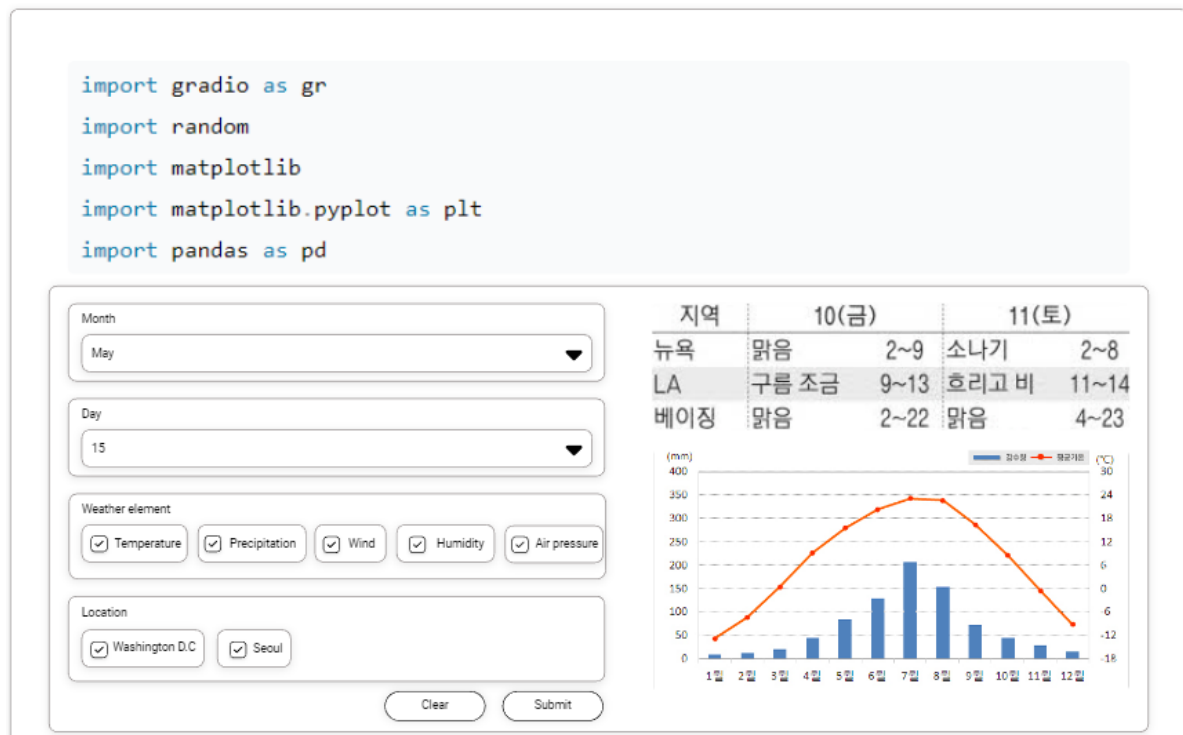


해당 Demo를 사용자가 사용할 때에 사용자 시나리오는 다음과 같다.

1. 사용자는 날짜를 Dropdown box에 있는 Month와 Day를 선택하여 입력한다.
2. 사용자는 날씨 요소를 Weather element의 Check box에 체크한다.
3. 사용자는 지역을 Location의 Radio button 중 하나를 체크한다.
4. 사용자는 입력값을 모두 설정한 후에 Submit 버튼을 누른다.
5. 오른쪽 위치에 입력값에 맞는 기상 수치표와 기상 그래프가 출력된다.

## 3. 상세 설계 내용

ui 스케치를 통해 구성한 데모이다.



## - Gradio components

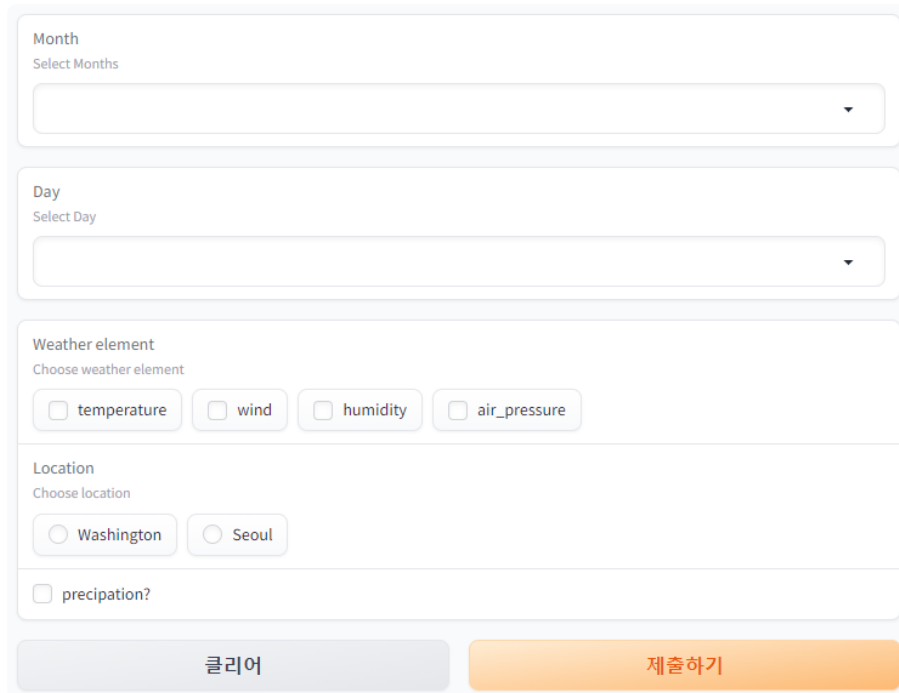
Month, Day -> **Dropdown**

Weather elements (temperature, wind, humidity, air pressure)

-> **CheckboxGroup**

Location (Washington, Seoul) -> **Radio**

Precipitation -> **Checkbox**



Month  
Select Months

Day  
Select Day

Weather element  
Choose weather element

☐ temperature ☐ wind ☐ humidity ☐ air\_pressure

Location  
Choose location

☐ Washington ☐ Seoul

☐ precipitation?

클리어 제출하기

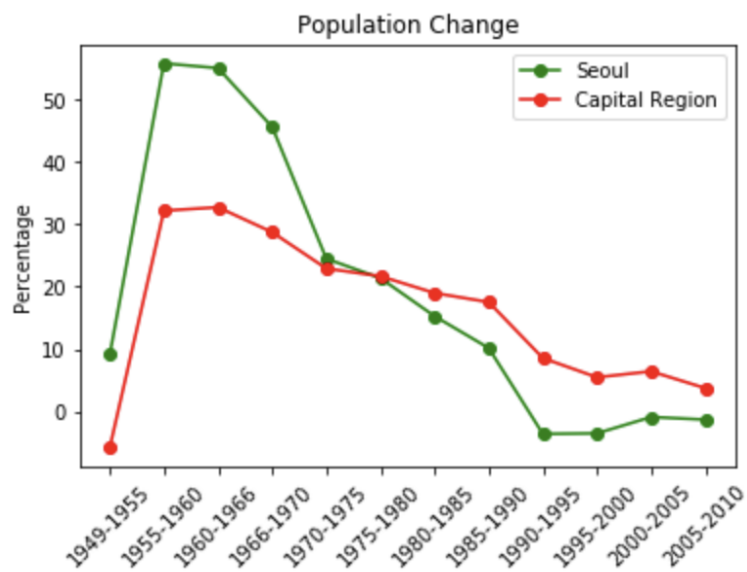
## - Matplotlib components

선 그래프 (기온, 풍속, 기압) -> matplotlib.pyplot의 **plot method** 이용.

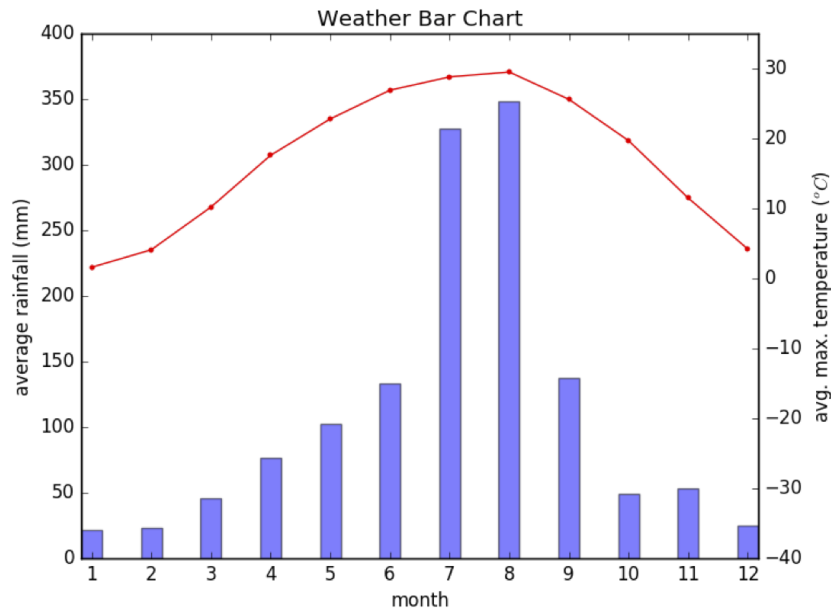
막대 그래프 (습도) -> matplotlib.pyplot의 **bar method** 이용.

Plot 구성 -> 기상 요소 개수에 따라 그래프 추가 (최대 plot 2개)

예시) 1번 plot X축 : 시간, Y축 : 기온, 풍속



예시) 2번 plot X축 : 시간, Y축 : 습도, 기압



※ 강수량은 일일 데이터의 양이 적기 때문에 그래프에 표시 X.

title = 2022년 기상 그래프

xlabel = 해당 date, ylabel = 해당 weather element

x축 rotation = 45

## - Pandas components

기상데이터 표 -> **DataFrame**

DataFrame 구성

index : hide, column : Location, date, time, Weather elements

데이터 전처리 후 Gradio를 이용하여 표 출력

In [33]: data

Out[33]:

	Area Abbreviation	Area Code	Area	Item Code	Item	Element Code	Element	Unit	latitude	longitude	...	Y2004	Y2005	Y2006	Y2007	Y2008	Y2009
0	AF	2	Afghanistan	2511	Wheat and products	5142	Food	1000 tonnes	33.94	67.71	...	3249.0	3486.0	3704.0	4164.0	4252.0	4538.0
1	AF	2	Afghanistan	2805	Rice (Milled Equivalent)	5142	Food	1000 tonnes	33.94	67.71	...	419.0	445.0	546.0	455.0	490.0	415.0
2	AF	2	Afghanistan	2513	Barley and products	5521	Feed	1000 tonnes	33.94	67.71	...	58.0	236.0	262.0	263.0	230.0	379.0
3	AF	2	Afghanistan	2513	Barley and products	5142	Food	1000 tonnes	33.94	67.71	...	185.0	43.0	44.0	48.0	62.0	55.0
4	AF	2	Afghanistan	2514	Maize and products	5521	Feed	1000 tonnes	33.94	67.71	...	120.0	208.0	233.0	249.0	247.0	195.0
5	AF	2	Afghanistan	2514	Maize and products	5142	Food	1000 tonnes	33.94	67.71	...	231.0	67.0	82.0	67.0	69.0	71.0
6	AF	2	Afghanistan	2517	Millet and products	5142	Food	1000 tonnes	33.94	67.71	...	15.0	21.0	11.0	19.0	21.0	18.0
7	AF	2	Afghanistan	2520	Cereals, Other	5142	Food	1000 tonnes	33.94	67.71	...	2.0	1.0	1.0	0.0	0.0	0.0
8	AF	2	Afghanistan	2531	Potatoes and products	5142	Food	1000 tonnes	33.94	67.71	...	276.0	294.0	294.0	260.0	242.0	250.0
9	AF	2	Afghanistan	2536	Sugar cane	5521	Feed	1000 tonnes	33.94	67.71	...	50.0	29.0	61.0	65.0	54.0	114.0
10	AF	2	Afghanistan	2537	Sugar beet	5521	Feed	1000 tonnes	33.94	67.71	...	0.0	0.0	0.0	0.0	0.0	0.0

- 사용자 정의 함수

1. 기상 데이터를 찾아주는 함수

**dataSearch(Month, Day, Weather\_elements, Location, Precipitation)**

pandas로 기상데이터.csv 파일 오픈

date에 맞는 형식으로 문자열 생성

매개변수와 매칭되는 data들만 DataFrame 형태로 추출

**return DataFrame**

2. 그래프와 표를 출력하는 함수

**showOutput(Month, Day, Weather\_elements, Location, Precipitation)**

영문 month를 숫자로 매핑

weatherTable = dataSearch(매핑된 지역변수들)

weatherTable 기반으로 matplotlib.pyplot 사용해서 weatherPlot 생성

**return [DataFrame, figure]**

위 컴포넌트들을 다 구성하여 `gradio.interface(showOutput, Gradio components, Matplotlib·Pandas components, examples)`로 `gradio demo`를 구성했다. `examples`는 입력 예시를 보여주는 테이블을 넣어 사용에 도움을 주는 역할을 한다. Demo를 다 구성했으면 `gradio.interface.launch()`로 실행한다.

#### 4. Demo 구현 코드

```
import gradio as gr
import pandas as pd
import matplotlib.pyplot as plt

# search weather data that is in csv file
def dataSearch(month, day, weather_elements, location, precipitation):
    if location=='Seoul':
        df = pd.read_csv('Seoul.csv')
    elif location=='Washington':
        df = pd.read_csv('Washington.csv')
    if precipitation:
        weather_elements.append('precipitation(mm)')
    if day in ['1','2','3','4','5','6','7','8','9']:
        today = '2022-'+month+'-0'+day
    else:
        today = '2022-'+month+'-'+day

    df1 = df[df.date == today]
    columns = ['location', 'date', 'time'] + weather_elements
    df2 = df1.loc[:, columns]
```

```

    return df2

# show weather data in plot using matplotlib
def showOutput(month, day, weather_elements, location, precipitation):
    if month=='January':
        month = '01'
    elif month=='February':
        month = '02'
    elif month=='March':
        month = '03'
    elif month=='April':
        month = '04'
    elif month=='May':
        month = '05'
    elif month=='June':
        month = '06'
    elif month=='July':
        month = '07'
    elif month=='August':
        month = '08'
    elif month=='September':
        month = '09'
    elif month=='October':
        month = '10'
    elif month=='November':
        month = '11'
    elif month=='December':
        month = '12'

    weatherTable = dataSearch(month, day, weather_elements, location,
precipitation)

    if precipitation:
        weather_elements.remove('precipitation(mm)')

    if day in ['1','2','3','4','5','6','7','8','9']:
        xname = '2022-'+month+'-0'+day
    else:
        xname = '2022-'+month+'-'+day

    y_value=[0]*len(weather_elements)

```

```

x_value = weatherTable['time']

if 'humidity(%)' in weather_elements:
    humidity_index = weather_elements.index('humidity(%)')
    if weather_elements[humidity_index] != weather_elements[-1]:
        temp = weather_elements[humidity_index]
        weather_elements[humidity_index] = weather_elements[-1]
        weather_elements[-1] = temp
    for i in range(len(weather_elements)):
        y_value[i] = weatherTable[weather_elements[i]]

if len(weather_elements) == 1:
    weatherPlot = plt.figure(figsize=(10,10))
    plt.title("2022 Weather Graph", fontsize=20, fontweight='bold')
    plt.xlabel(xname, labelpad=5, fontsize=15)
    plt.ylabel(weather_elements[0], labelpad=15, fontsize=15)
    plt.xticks(size=10, rotation=45)

    plt.bar(x_value, y_value[-1], color='skyblue',
label=weather_elements[0])
    plt.legend(loc = "upper left")

elif len(weather_elements) == 2:
    weatherPlot, ax1 = plt.subplots(figsize=(10,10))
    plt.title("2022 Weather Graph", fontsize=20, fontweight='bold')
    plt.xticks(size=10, rotation=45)

    ax1.bar(x_value, y_value[-1], color='skyblue',
label=weather_elements[1])
    ax1.set_xlabel(xname, labelpad=5, fontsize=15)
    ax1.set_ylabel(weather_elements[1], labelpad=15, fontsize=15)
    ax1.legend(loc='upper left')

    ax1_sub = ax1.twinx()
    ax1_sub.plot(x_value, y_value[0], color='red', marker = "o",
label=weather_elements[0])
    ax1_sub.set_ylabel(weather_elements[0], labelpad=25, fontsize=15,
rotation=270)
    ax1_sub.legend(loc='upper right')

elif len(weather_elements) == 3:

```



```

        weatherPlot, (ax1, ax2) = plt.subplots(2, 1, figsize=(10,15),
constrained_layout=True)

        line1 = ax1.plot(x_value, y_value[0], color='red', marker = "o",
label=weather_elements[0])
        ax1.set_xlabel(xname, labelpad=5, fontsize=15)
        ax1.set_ylabel(weather_elements[0], labelpad=15, fontsize=15)
        ax1.set_title("2022 Weather Graph", fontsize=20,
fontWeight='bold')
        ax1.tick_params(axis='x', rotation=45, labels=10)

        ax1_sub = ax1.twinx()
        line2 = ax1_sub.plot(x_value, y_value[1], color='blue', marker =
"o", label=weather_elements[1])
        ax1_sub.set_ylabel(weather_elements[1], labelpad=25, fontsize=15,
rotation=270)

        lines = line1 + line2
        labels = [l.get_label() for l in lines]
        ax1.legend(lines, labels, loc='upper left')

        ax2.bar(x_value, y_value[-1], color='skyblue',
label=weather_elements[-1])
        ax2.set_xlabel(xname, labelpad=5, fontsize=15)
        ax2.set_ylabel(weather_elements[-1], labelpad=15, fontsize=15)
        ax2.tick_params(axis='x', rotation=45, labels=10)
        ax2.legend(loc='upper right')

    elif len(weather_elements) == 4:
        weatherPlot, (ax1, ax2) = plt.subplots(2,1, figsize=(10,15),
constrained_layout=True)

        line1 = ax1.plot(x_value, y_value[0], color='red', marker = "o",
label=weather_elements[0])
        ax1.set_xlabel(xname, labelpad=5, fontsize=15)
        ax1.set_ylabel(weather_elements[0], labelpad=15, fontsize=15)
        ax1.set_title("2022 Weather Graph", fontsize=20,
fontWeight='bold')
        ax1.tick_params(axis='x', rotation=45, labels=10)

        ax1_sub = ax1.twinx()
        line2 = ax1_sub.plot(x_value, y_value[1], color='blue', marker =
"o", label=weather_elements[1])

```

```

        ax1_sub.set_ylabel(weather_elements[1], labelpad=25, fontsize=15,
rotation=270)

        lines = line1 + line2
        labels = [l.get_label() for l in lines]
        ax1.legend(lines, labels, loc='upper left')

        ax2.bar(x_value, y_value[-1], color='skyblue',
label=weather_elements[-1])
        ax2.set_xlabel(xname, labelpad=5, fontsize=15)
        ax2.set_ylabel(weather_elements[-1], labelpad=15, fontsize=15)
        ax2.tick_params(axis='x', rotation=45, labels=10)
        ax2.legend(loc='upper left')

        ax2_sub = ax2.twinx()
        ax2_sub.plot(x_value, y_value[2], color='gray', marker = "o",
label=weather_elements[2])
        ax2_sub.set_ylabel(weather_elements[2], labelpad=25, fontsize=15,
rotation=270)
        ax2_sub.legend(loc='upper right')

    else:
        for i in range(len(weather_elements)):
            y_value[i] = weatherTable[weather_elements[i]]

        if len(weather_elements) == 1:
            weatherPlot = plt.figure(figsize=(10,10))
            plt.title("2022 Weather Graph", fontsize=20, fontweight='bold')
            plt.xlabel(xname, labelpad=5, fontsize=15)
            plt.ylabel(weather_elements[0], labelpad=15, fontsize=15)
            plt.xticks(size=10, rotation=45)
            plt.plot(x_value, y_value[0], color='red', marker='o',
label=weather_elements[0])
            plt.legend(loc = "upper left")

        elif len(weather_elements) == 2:
            weatherPlot, ax1 = plt.subplots(figsize=(10,10))
            plt.title("2022 Weather Graph", fontsize=20, fontweight='bold')
            plt.xticks(size=10, rotation=45)

            line1 = ax1.plot(x_value, y_value[0], color='red', marker='o',
label=weather_elements[0])

```

```

ax1.set_xlabel(xname, labelpad=5, fontsize=15)
ax1.set_ylabel(weather_elements[0], labelpad=15, fontsize=15)

ax1_sub = ax1.twinx()
line2 = ax1_sub.plot(x_value, y_value[1], color='skyblue',
marker='o', label=weather_elements[1])
ax1_sub.set_ylabel(weather_elements[1], labelpad=25, fontsize=15,
rotation=270)

lines = line1 + line2
labels = [l.get_label() for l in lines]
ax1.legend(lines, labels, loc='upper left')

elif len(weather_elements) == 3:
    weatherPlot, (ax1, ax2) = plt.subplots(2,1, figsize=(10,15),
constrained_layout=True)

    line1 = ax1.plot(x_value, y_value[0], color='red', marker = "o",
label=weather_elements[0])
    ax1.set_xlabel(xname, labelpad=5, fontsize=15)
    ax1.set_ylabel(weather_elements[0], labelpad=15, fontsize=15)
    ax1.set_title("2022 Weather Graph", fontsize=20,
fontweight='bold')
    ax1.tick_params(axis='x', rotation=45, labelsiz=10)

    ax1_sub = ax1.twinx()
    line2 = ax1_sub.plot(x_value, y_value[1], color='skyblue', marker
= "o", label=weather_elements[1])
    ax1_sub.set_ylabel(weather_elements[1], labelpad=25, fontsize=15,
rotation=270)

    lines = line1 + line2
    labels = [l.get_label() for l in lines]
    ax1.legend(lines, labels, loc='upper left')

    ax2.plot(x_value, y_value[2], color='gray', marker = "o",
label=weather_elements[2])
    ax2.set_xlabel(xname, labelpad=5, fontsize=15)
    ax2.set_ylabel(weather_elements[2], labelpad=15, fontsize=15)
    ax2.tick_params(axis='x', rotation=45, labelsiz=10)
    ax2.legend(loc='upper left')

return [weatherTable, weatherPlot]

```

```

output1 = gr.Dataframe()
output2 = gr.Plot()

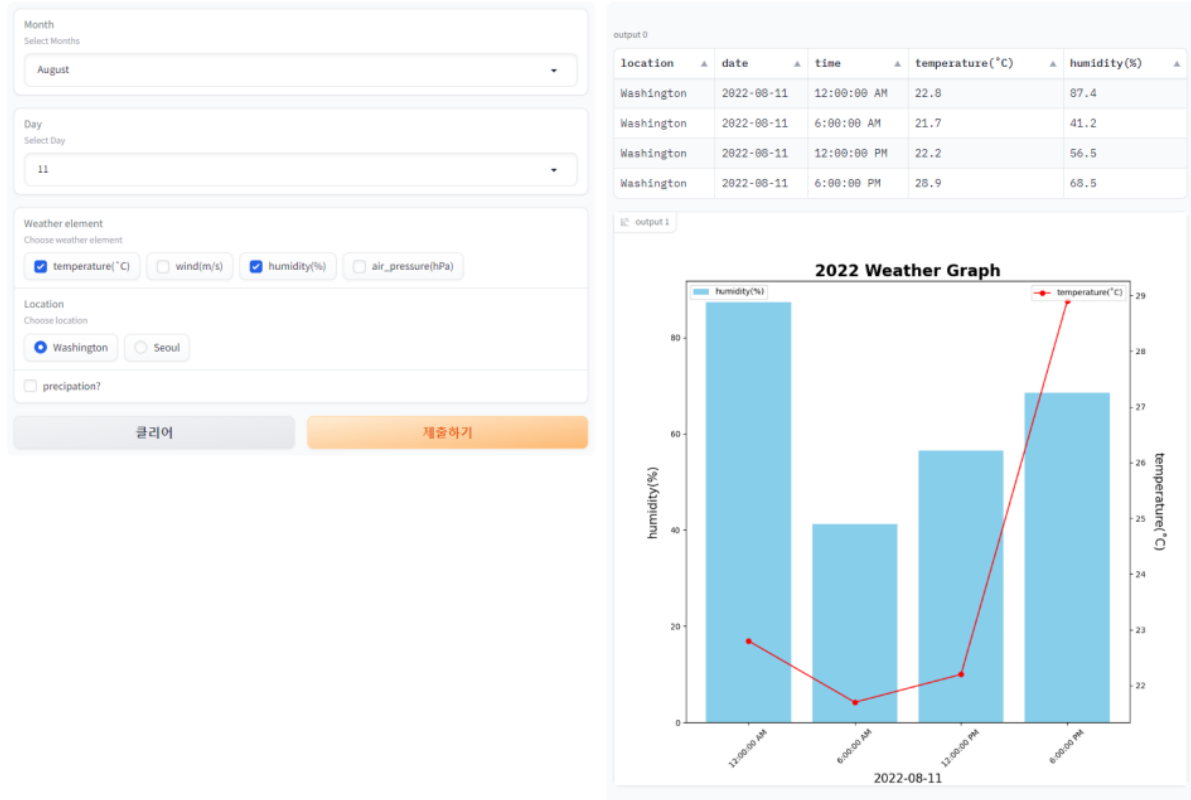
# make gradio interface
demo = gr.Interface(
    fn=showOutput,
    inputs=[
        gr.Dropdown(["January", "February", "March", "April", "May",
"June",
                    "July", "August", "September", "October",
"November", "December"],
                    label="Month", info="Select Months"),
        gr.Dropdown(["1", "2", "3", "4", "5", "6", "7", "8", "9", "10",
"11", "12", "13", "14", "15", "16", "17", "18",
"19", "20",
                    "21", "22", "23", "24", "25", "26", "27", "28",
"29", "30", "31"], label="Day", info="Select Day"),
        gr.CheckboxGroup(["temperature (°C)", "wind (m/s)",
"humidity (%)", "air_pressure (hPa)"],
                    label="Weather element", info="Choose weather
element"),
        gr.Radio(["Washington", "Seoul"],
                    label="Location", info="Choose location"),
        gr.Checkbox(label="precipitation?"),
        outputs=[output1, output2]
    )

if __name__ == "__main__":
    demo.launch()

```

## 5. Demo 실행 화면

Demo를 실행한 결과는 다음과 같다.



## 6. 추진 전략

아래와 같은 전략으로 프로젝트를 진행하였다.

- 1) Demo 제작을 빠르게 진행하면서 Demo에 대한 구체적인 설명이 포함된 문서들을 작성한다.
- 2) Demo에 대해 설명 시 Demo를 직접 사용해보는 animated gif나 영상을 만들어 gradio 측에서 쉽게 이해할 수 있게 한다.
- 3) 문서 같은 경우에는 영어로 작성하여 github에 pull request를 보내고, gradio 측과 소통을 하며 요구에 맞는 형태로 수정 및 보완을 진행한다. 이때에 gradio 측의 답변이 늦어질 수 있으므로 최대한 빠르게 pull request 한다.
- 4) Demo 제작에 사용한 코드(ipynb파일) 및 문서는 github에 올려 모든 사람들이 볼 수 있게 한다.

## 7. 기여 결과

gradio의 awesome-demos repo에 위 데모를 추가하여 Pull request를 해서 awesome-demos에 있는 README.md 파일 내용 안에 Tabular Data & Plots 카테고리 Weather Data & Graph in 2022의 이름으로 데모가 추가되었다. 그리고 프로젝트의 목표인 gradio 홈페이지에 데모가 게시되는것에 대해서는 gradio 측 개발자분과 논의 중에 있다.

## 8. 참고 문헌

- 1) Streamlit : <https://streamlit.io/>
- 2) 기상자료개방포털 : <https://data.kma.go.kr/data/ogd/selectGtsRltmList.do?pgmNo=658>
- 3) Gradio-app/gradio : <https://github.com/gradio-app/gradio>
- 4) Gradio-app/awesome-demos : <https://github.com/gradio-app/awesome-demos>