|  |
| --- |
| 비플레이스 |
| Streamlit Hands On |
| PART FIVE: DESIGNING AN APPLICATION WITH STREAMLIT |

|  |
| --- |
| 김무경  2023-10-26 |

내용

[1. Streamlit 과정 소개 4](#_Toc151644306)

[1.1. 과정을 위한 선행 요건 6](#_Toc151644307)

[1. 개발환경구성 6](#_Toc151644308)

[1.1. 개발환경 6](#_Toc151644309)

[1.2. 파이썬 설치(아나콘다) 7](#_Toc151644310)

[1.2.1. 아나콘다 다운로드 7](#_Toc151644311)

[1.2.2. 아나콘다 설치 9](#_Toc151644312)

[1.2.3. 설치 확인 15](#_Toc151644313)

[1.2. Github 서비스 가입 16](#_Toc151644314)

[1.2.1. Github 계정 만들기 16](#_Toc151644315)

[1.2.2. 리포지토리 생성하기 18](#_Toc151644316)

[1.3. 스트림릿 클라우드 가입 19](#_Toc151644317)

[1.4. 가상환경 생성 23](#_Toc151644318)

[1.4.1. 명령어 창 실행 23](#_Toc151644319)

[1.4.2. 가상환경 생성 24](#_Toc151644320)

[1.4.3. 가상환경 생성 결과 확인 25](#_Toc151644321)

[1.5. 가상환경 실행 26](#_Toc151644322)

[1.5.1. 가상환경 실행 26](#_Toc151644323)

[1.5.2. 가상환경에 설치된 패키지 확인 26](#_Toc151644324)

[1.5.3. 가상환경 실행 종료 26](#_Toc151644325)

[2. 스트림릿 소개 27](#_Toc151644326)

[2.1. 스트림릿 소개 27](#_Toc151644327)

[2.1.1. 스트림릿 공식 홈페이지 27](#_Toc151644328)

[2.1.2. App Gallery 27](#_Toc151644329)

[2.1.3. 스트림릿과 웹 개발 차이점 28](#_Toc151644330)

[2.1.4. STREAMLIT 프레임워크의 특징 28](#_Toc151644331)

[2.1.5. 일반적인 웹 페이지 구성요소 28](#_Toc151644332)

[2.1.6. HTML 폼 요소 예시 29](#_Toc151644333)

[2.2. 스트림릿 설치 31](#_Toc151644334)

[2.2.1. streamlit 패키지 설치 31](#_Toc151644335)

[2.2.2. streamlit 패키지 설치 확인 31](#_Toc151644336)

[2.3. 스트림릿 기본 사용법 32](#_Toc151644337)

[2.3.1. Streamlit 실행 32](#_Toc151644338)

[2.3.2. 방화벽 설정 해제 33](#_Toc151644339)

[2.3.3. Streamlit 실행결과 확인 33](#_Toc151644340)

[2.3.4. Streamlit 화면 구성 34](#_Toc151644341)

[2.4. 코딩 에디터 38](#_Toc151644342)

[2.4.1. 에디터 플러스 문서 편집기 설치 38](#_Toc151644343)

[3. 스트림릿 사용하기 39](#_Toc151644344)

[3.1. 스트림릿 API의 구성 39](#_Toc151644345)

[3.2. Creating Our First App 39](#_Toc151644346)

[3.2.1. Creating a Home Page 39](#_Toc151644347)

[3.2.2. Home.py 내용 수정 해보기 40](#_Toc151644348)

[3.3. Displaying Data in Streamlit 41](#_Toc151644349)

[3.3.1. Text 데이터 화면 출력하기 42](#_Toc151644350)

[3.3.2. Displaying Tabular Data 46](#_Toc151644351)

[3.3.3. Plotting Basic Graphs with Streamlit 50](#_Toc151644352)

[3.4. Custom HTML 54](#_Toc151644353)

[3.5. 스트림릿 위젯 55](#_Toc151644354)

[3.5.1. Text Input Widgets 57](#_Toc151644355)

[3.5.2. Numerical Input Widgets 58](#_Toc151644356)

[3.5.3. Date and Time Input Widgets 60](#_Toc151644357)

[3.5.4. Boolean Input Widgets 62](#_Toc151644358)

[3.5.5. Selection Widgets 64](#_Toc151644359)

[3.6. 레이아웃 (Layout Widgets) 69](#_Toc151644360)

[3.6.1. Sidebar 70](#_Toc151644361)

[3.6.2. Columns 70](#_Toc151644362)

[3.6.3. Expander 71](#_Toc151644363)

[3.6.4. Container 72](#_Toc151644364)

[3.6.5. Tabs 72](#_Toc151644365)

[3.6.6. Empty 73](#_Toc151644366)

[3.6.7. 코드 73](#_Toc151644367)

[3.7. Storing Data with st.session\_state 75](#_Toc151644368)

[3.8. 차트 77](#_Toc151644369)

[4. 데모 어플리케이션 개발 및 클라우드 환경 배포 77](#_Toc151644370)

[4.1. Building a Database Query Application 77](#_Toc151644371)

[4.1.1. Importing the Libraries 81](#_Toc151644372)

[4.1.2. Caching Data 81](#_Toc151644373)

[4.1.3. Creating our App Layout 83](#_Toc151644374)

[4.1.4. Using User Inputs to Produce a New DataFrame 85](#_Toc151644375)

[4.2. 클라우드 환경 배포 88](#_Toc151644376)

[4.2.1. Deploying an App in the Cloud with Streamlit Share 88](#_Toc151644377)

[4.2.2. Create a GitHub Account 88](#_Toc151644378)

[4.2.3. Upload Application to GitHub 88](#_Toc151644379)

[4.2.4. Connect Streamlit Share to your GitHub 93](#_Toc151644380)

[4.2.5. Create a New App 94](#_Toc151644381)

[4.2.6. Set Custom Subdomain 95](#_Toc151644382)

[5. 참조문헌 99](#_Toc151644383)

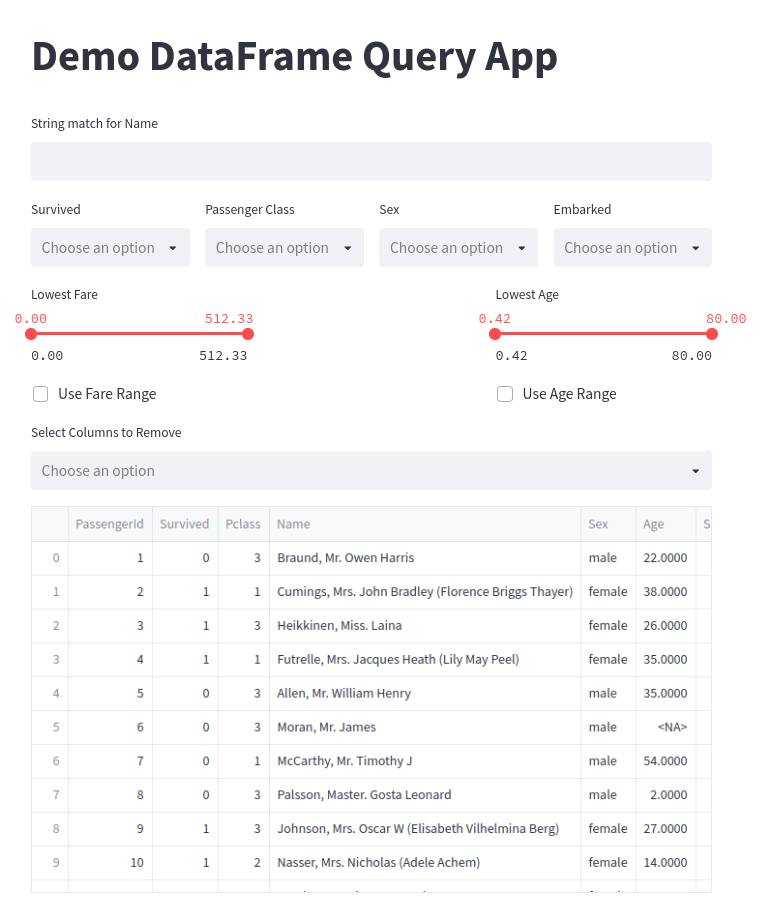
# Streamlit 과정 소개

Streamlit을 활용해서 파이썬 웹 데이터 어플리케이션을 개발하고 개발한 어플리케이션을 클라우드 환경에 배포하는 과정이다.

내용은 아래와 같습니다.

|  |  |
| --- | --- |
| 번호 | 내용 |
| 01 | 아나콘다 설치 및 가상환경 생성하기 |
| 02 | Streamlit 기초, 주요 용어 이해, 데이터 표시 방법, 주요 위젯 활용 방법 |
| 03 | 고급 기능 활용, 데이터 시각화, 레이아웃 제어, 사용자 정의 HTML |
| 04 | Pandas를 활용한 데이터베이스 쿼리 애플리케이션 개발 |
| 05 | 클라우드 환경, Streamlit Share에 어플리케이션 배포하기 |

아래는 그림은 본 과정에서 개발하고 배포할 어플리케이션의 최종 모습이다.



## 과정을 위한 선행 요건

1. 파이썬 기본 문법에 대한 이해(IF, FOR, 변수 선언, 함수 작성 등)
2. 파이썬 라이브러리 설치 방법
3. Pandas, Numpy, Matplotlib 라이브러리 사용법
4. 기본적인 Github 사용법
5. 개인 노트북
6. 이메일 계정

# 개발환경구성

## 개발환경

1. 파이썬 아나콘다 배포판
2. 파이썬 가상환경 생성 및 이용
3. Github 레파지토리
4. 스트림릿 클라우드
5. 텍스트 에디터 또는 IDE 환경
6. C:\PYLABS, C:\PYLABS\PY 폴더 생성
7. 파이썬 설치 경로는 C:\PYLABS\PY

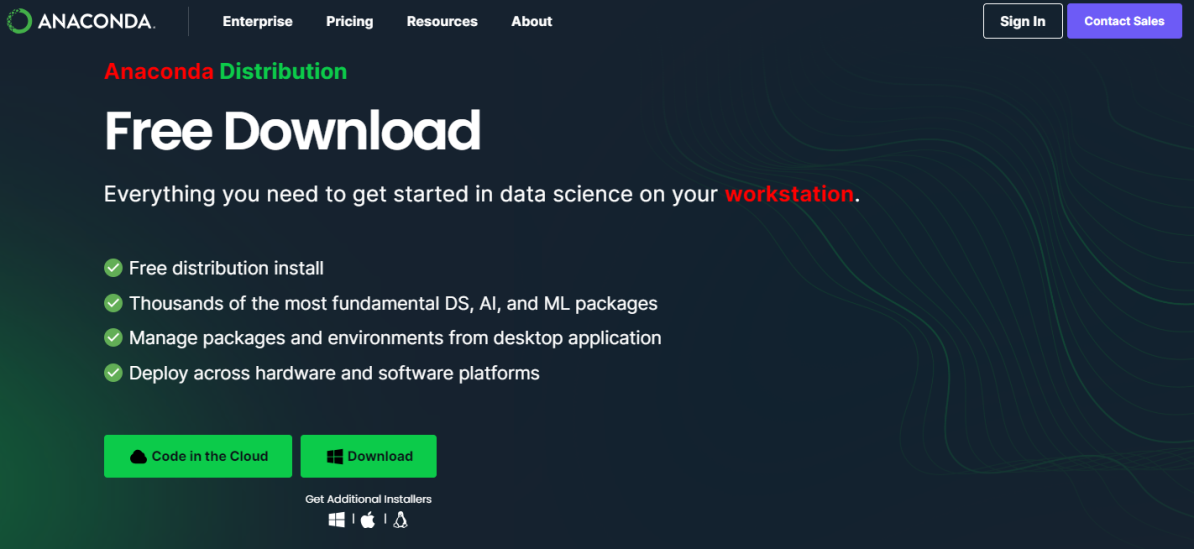
## 파이썬 설치(아나콘다)

### 아나콘다 다운로드

1. STEP 01: 아래 주소로 접속

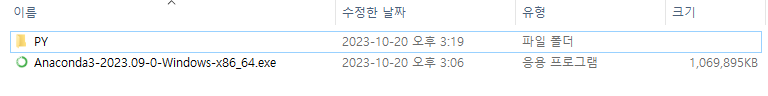
<https://www.anaconda.com/download>

1. 아나콘다 윈도우 설치 버전 선택



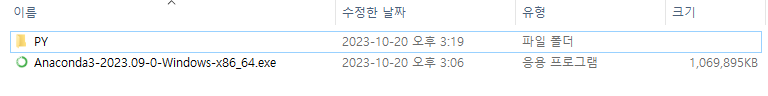


1. 설치 파일 저장



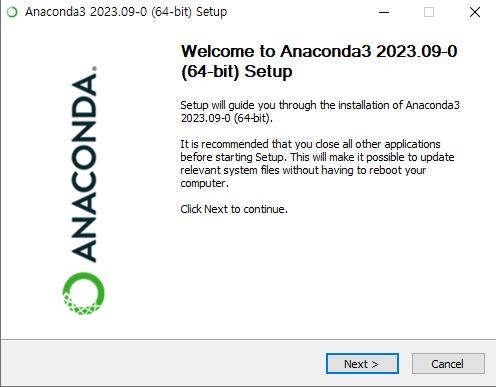
### 아나콘다 설치

1. 설치 파일 실행



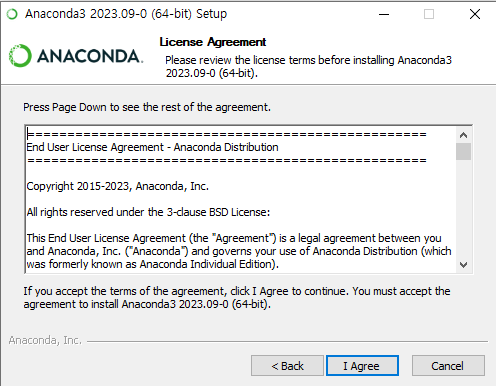
다운로드 설치 파일을 실행한다.

1. STEP 01



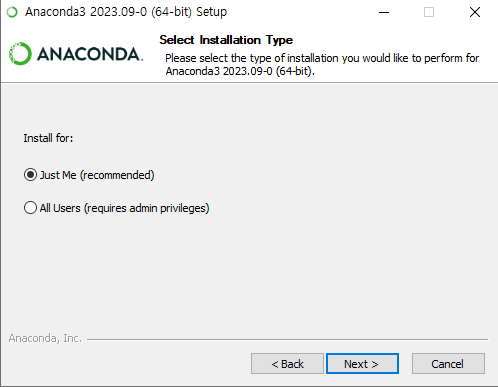
Next를 클릭한다.

1. STEP 02



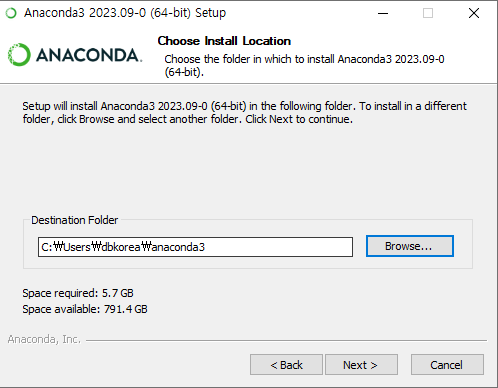
Next를 클릭한다.

1. STEP 03



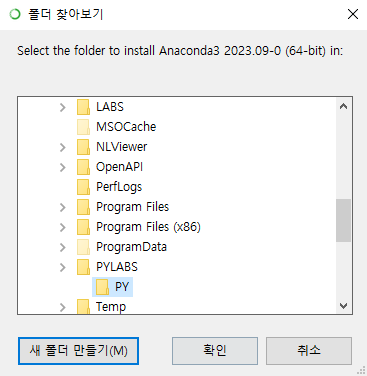
Next를 클릭한다.

1. STEP 04



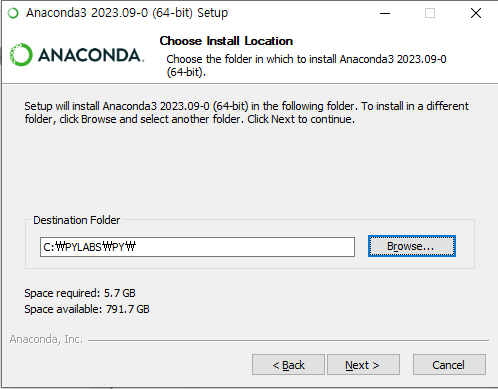
Browser 버튼을 클릭한다.

1. STEP 05



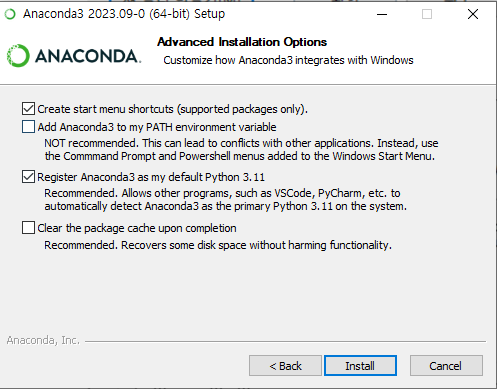
C:\PYLABS\PY 폴더를 선택한다.

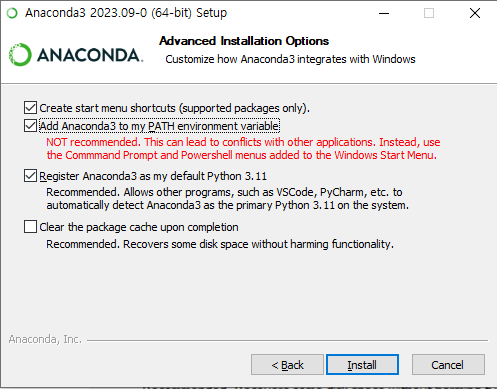
1. STEP 06



Next를 클릭한다.

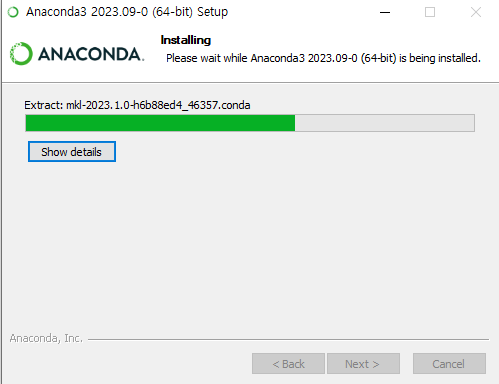
1. STEP 07

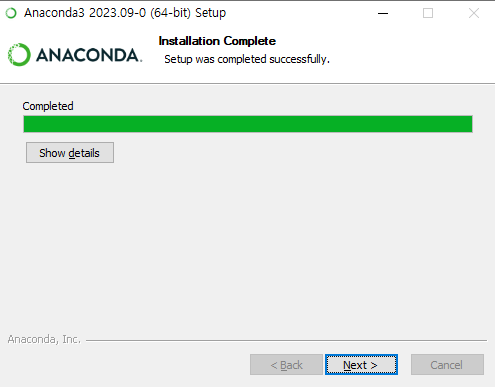




위 화면과 같이 선택하고, Next를 클릭한다.

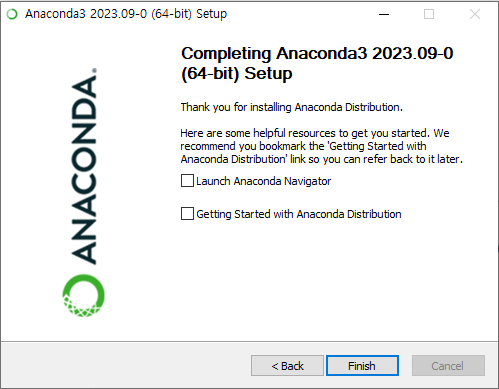
1. STEP 09





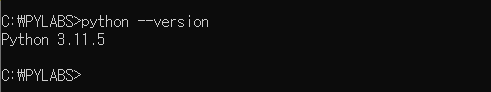


설치가 완료되면 Next를 클릭한다.



Finish를 클릭해서 설치를 완료한다.

### 설치 확인



위와 같이 명령 프롬프트를 실행하고,

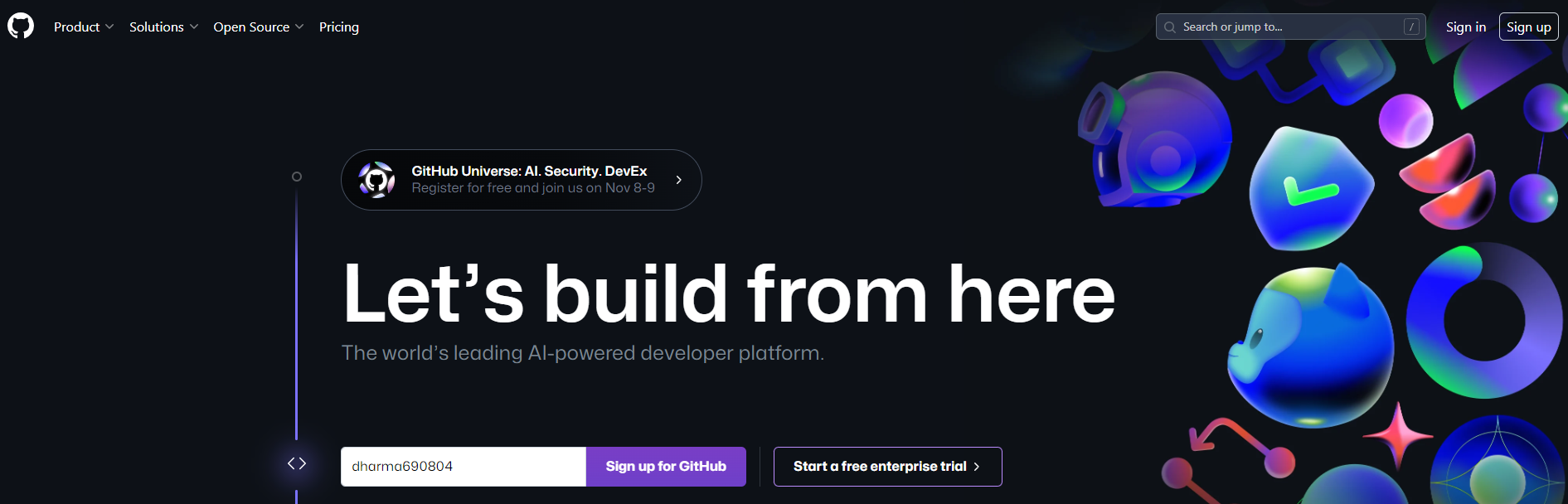
python --version

명령을 입력하고 정상적으로 파이썬 버전을 출력하면 설치가 완료된 것이다.

## Github 서비스 가입

### Github 계정 만들기

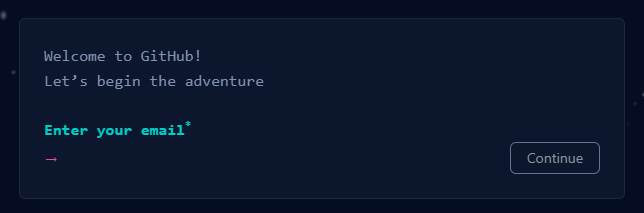
1. Github 사이트 접속(https://github.com/)

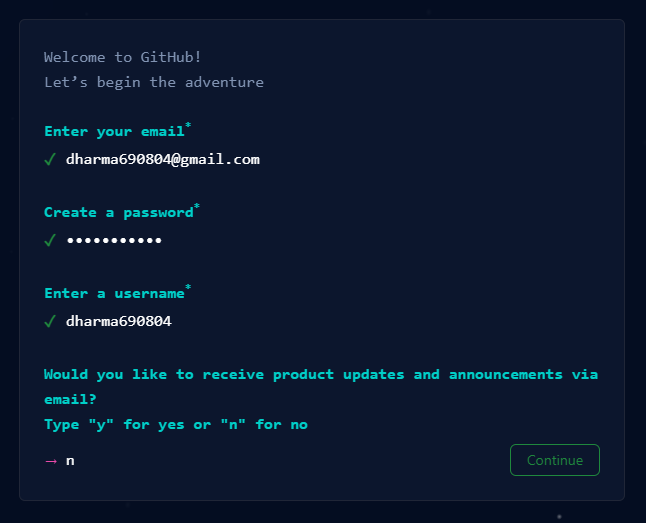
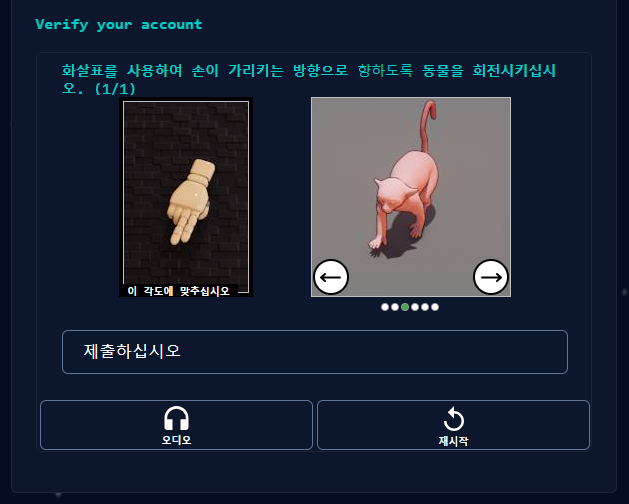


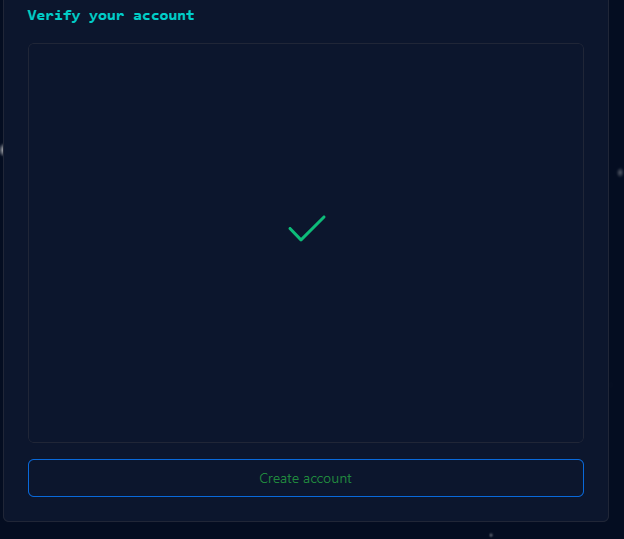
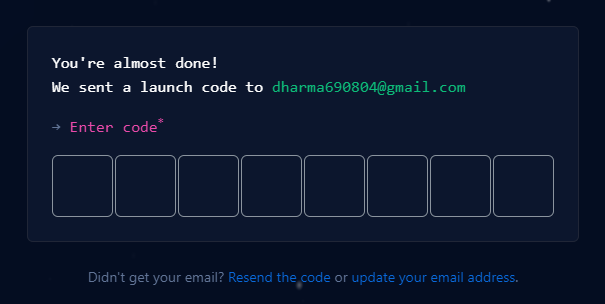
1. 계정 가입 클릭(Sign up)

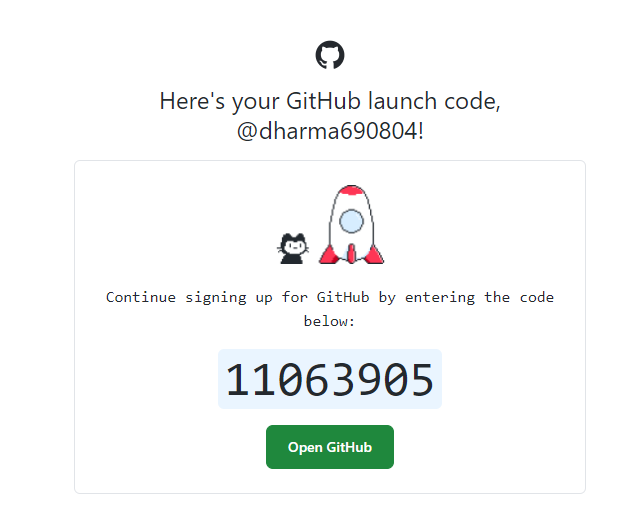


1. 이메일 입력 및 회원 가입 절차 진행



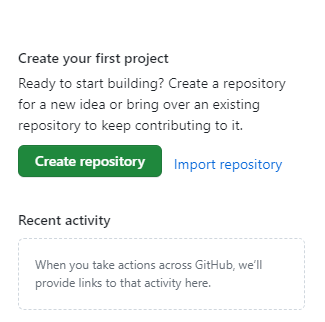
 



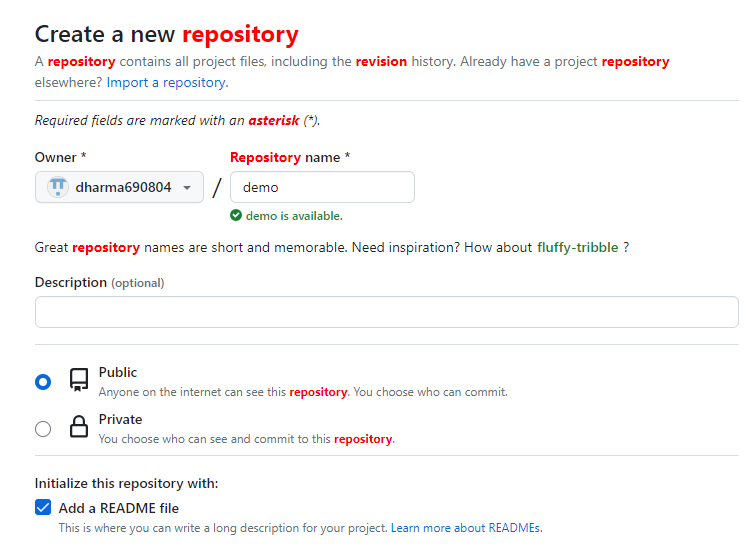
가입 시 입력한 이메일로 전달된 인증 코드를 입력하고 가입을 완료한다.

### 리포지토리 생성하기

1. Github 계정 로그인
2. 리포지토리 생성을 클릭



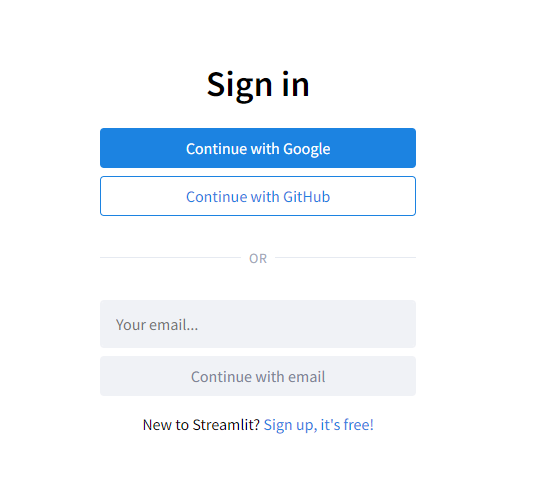
1. 리포지토리 명을 입력



생성을 클릭해서 리포지토리 생성을 완료한다.

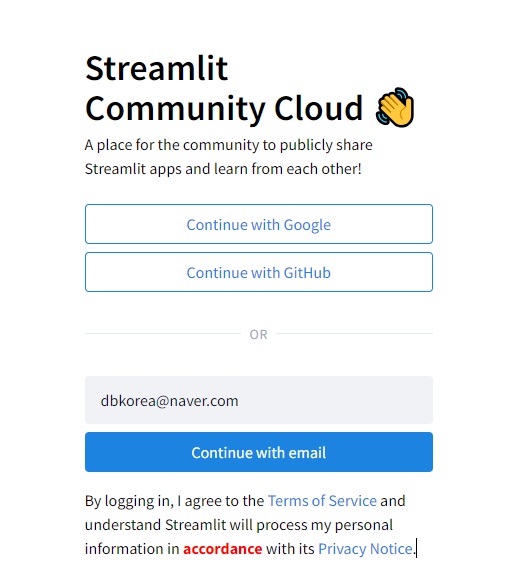
## 스트림릿 클라우드 가입

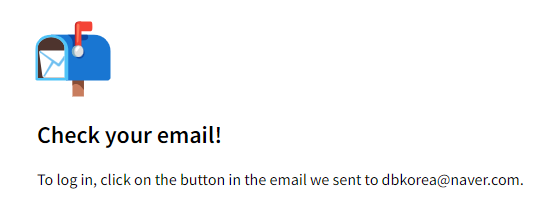
1. 스트림릿 클라우드 접속(<https://share.streamlit.io/>)



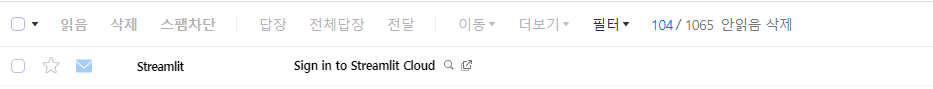
Github 계정을 이용해서 스트림릿 클라우드 서비스와 연동한다.

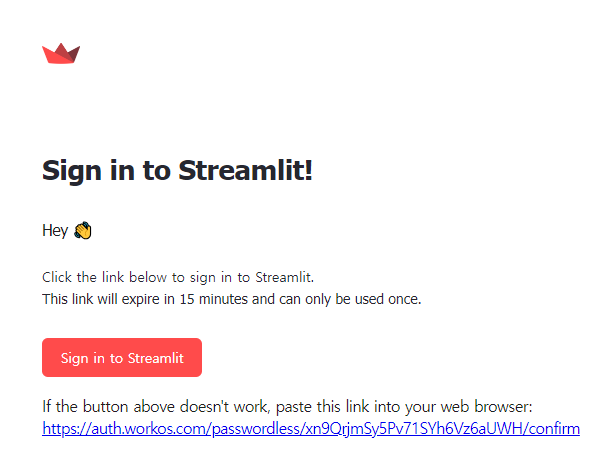
1. 이메일 주소를 입력하고 Continue with email을 클릭한다.



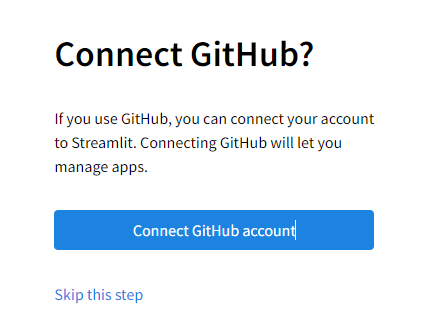


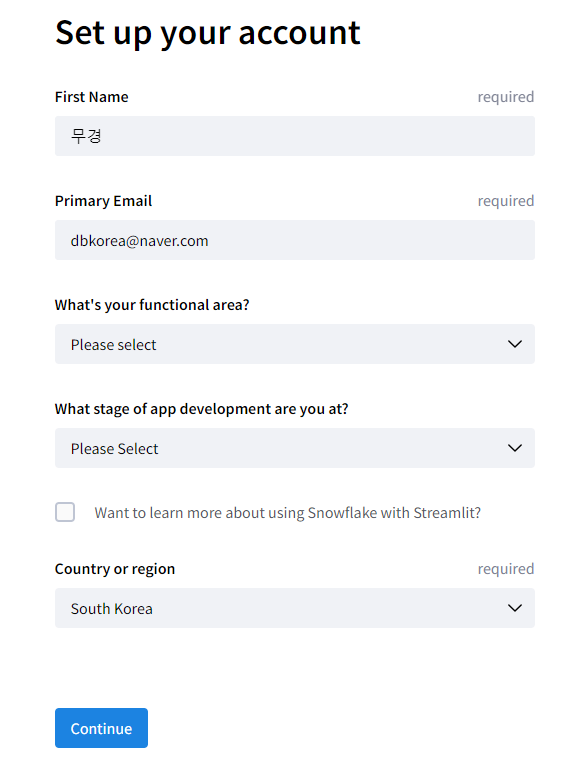
1. 이메일 확인하고 링크를 클릭한다.

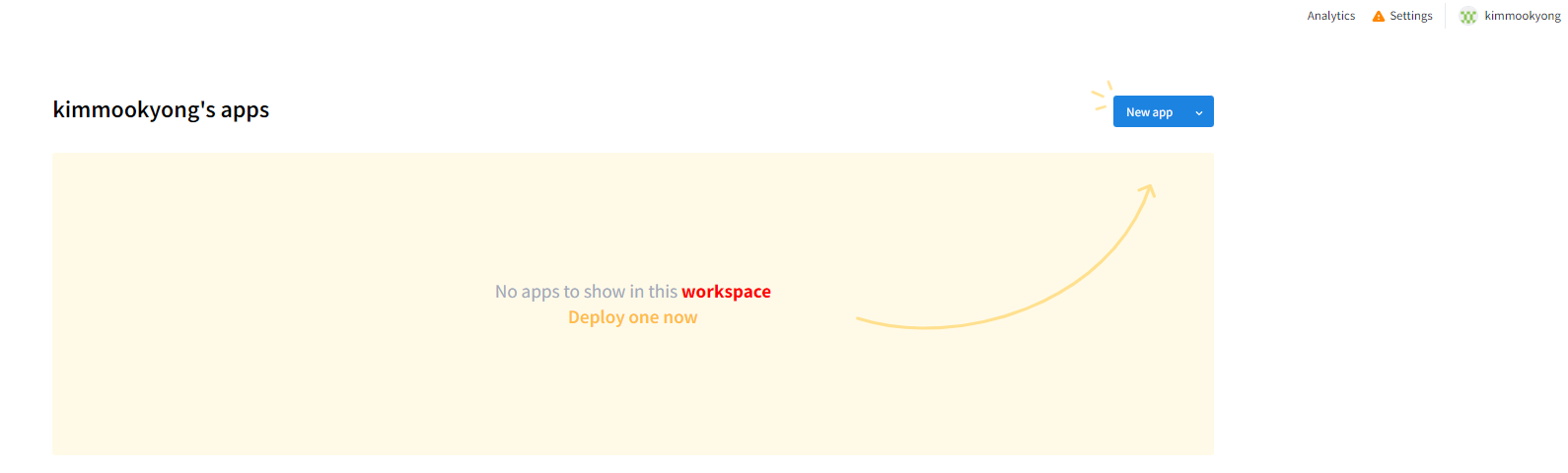


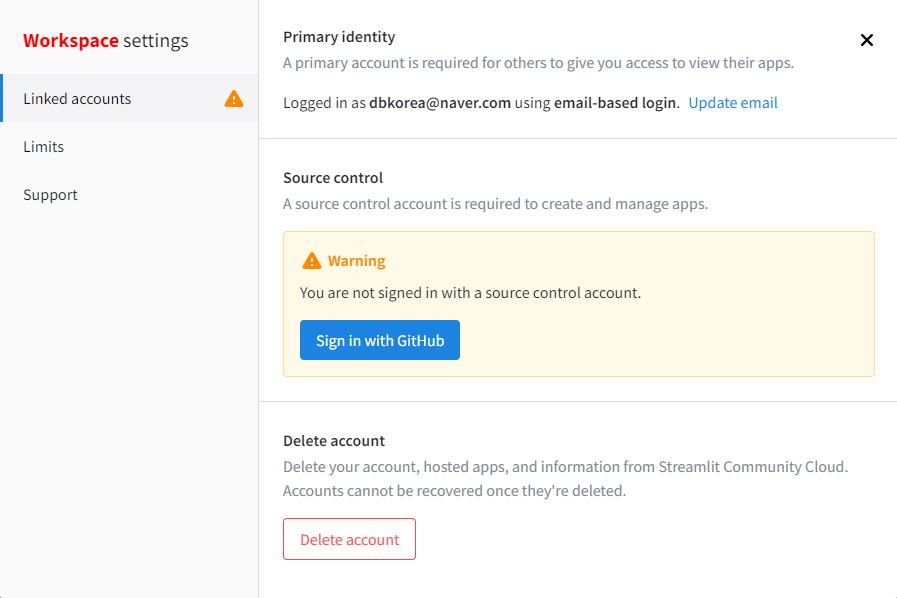


1. Github 계정 연동



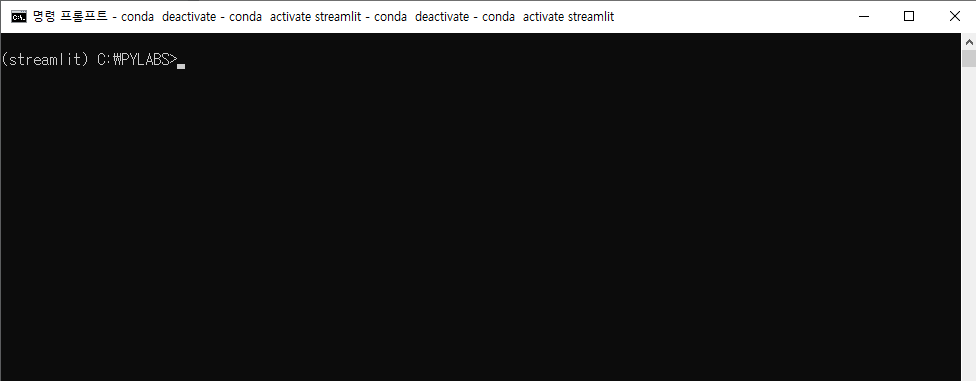






## 가상환경 생성

### 명령어 창 실행



### 가상환경 생성

C:\PYLABS>conda create --name streamlit python=3.8

Downloading and Extracting Packages

Preparing transaction: done

Verifying transaction: done

Executing transaction: done

#

# To activate this environment, use

#

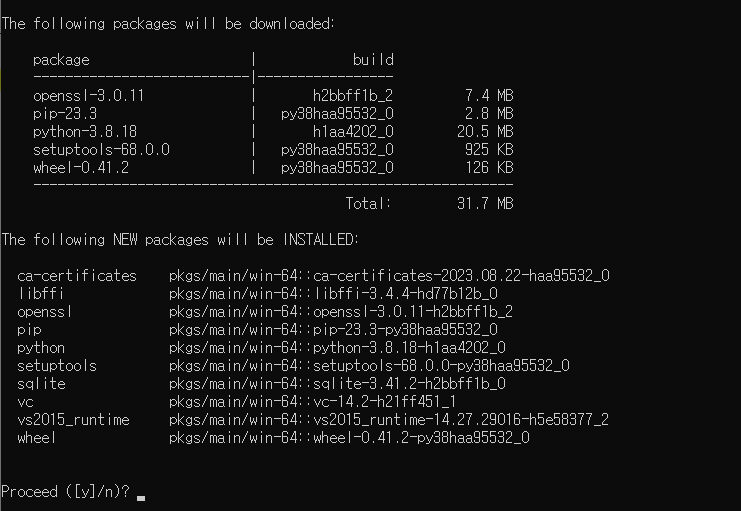
# $ conda activate streamlit

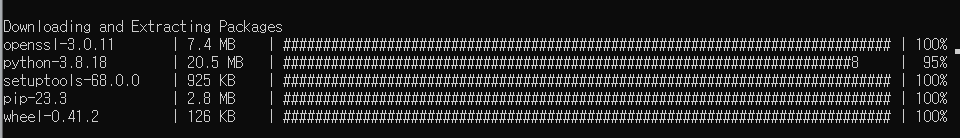
#

# To deactivate an active environment, use

#

# $ conda deactivate





### 가상환경 생성 결과 확인

C:\PYLABS>conda env list

# conda environments:

#

base C:\PYLABS\PY

streamlit C:\PYLABS\PY\envs\streamlit

## 가상환경 실행

### 가상환경 실행

C:\PYLABS>conda activate streamlit

(streamlit) C:\PYLABS>

### 가상환경에 설치된 패키지 확인

(streamlit) C:\PYLABS>conda list

# packages in environment at C:\PYLABS\PY\envs\streamlit:

#

# Name Version Build Channel

ca-certificates 2023.08.22 haa95532\_0

libffi 3.4.4 hd77b12b\_0

openssl 3.0.11 h2bbff1b\_2

pip 23.3 py38haa95532\_0

python 3.8.18 h1aa4202\_0

setuptools 68.0.0 py38haa95532\_0

sqlite 3.41.2 h2bbff1b\_0

vc 14.2 h21ff451\_1

vs2015\_runtime 14.27.29016 h5e58377\_2

wheel 0.41.2 py38haa95532\_0

(streamlit) C:\PYLABS>

### 가상환경 실행 종료

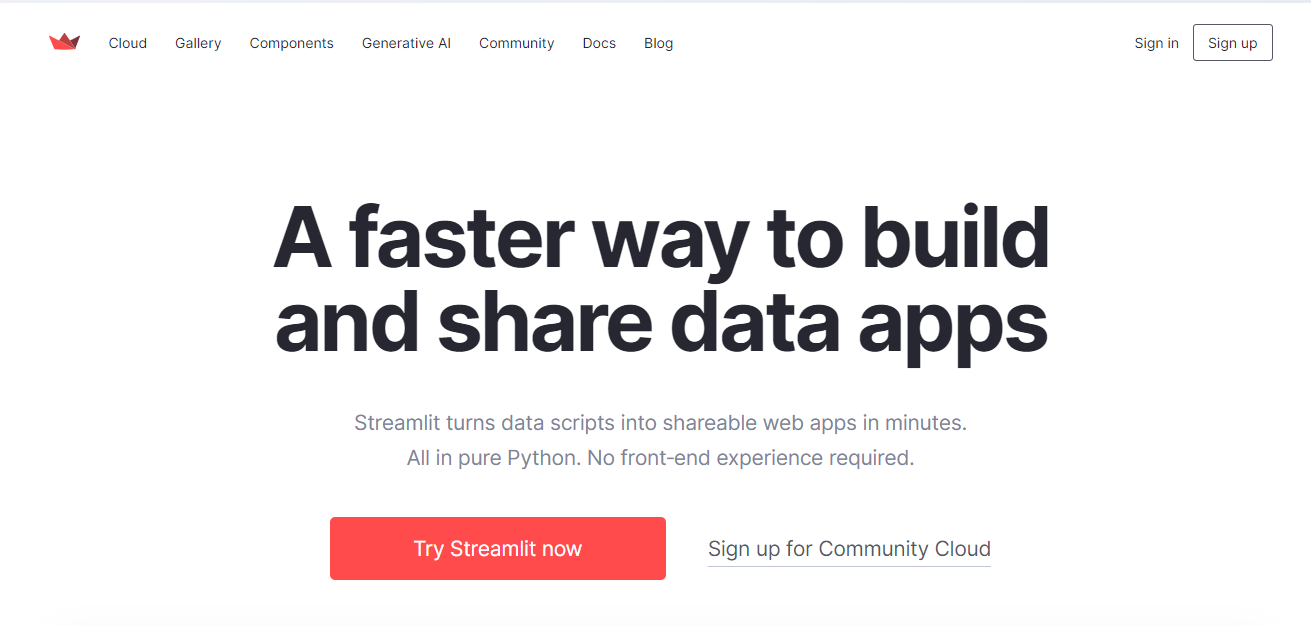
(streamlit) C:\PYLABS>conda deactivate

# 스트림릿 소개

## 스트림릿 소개

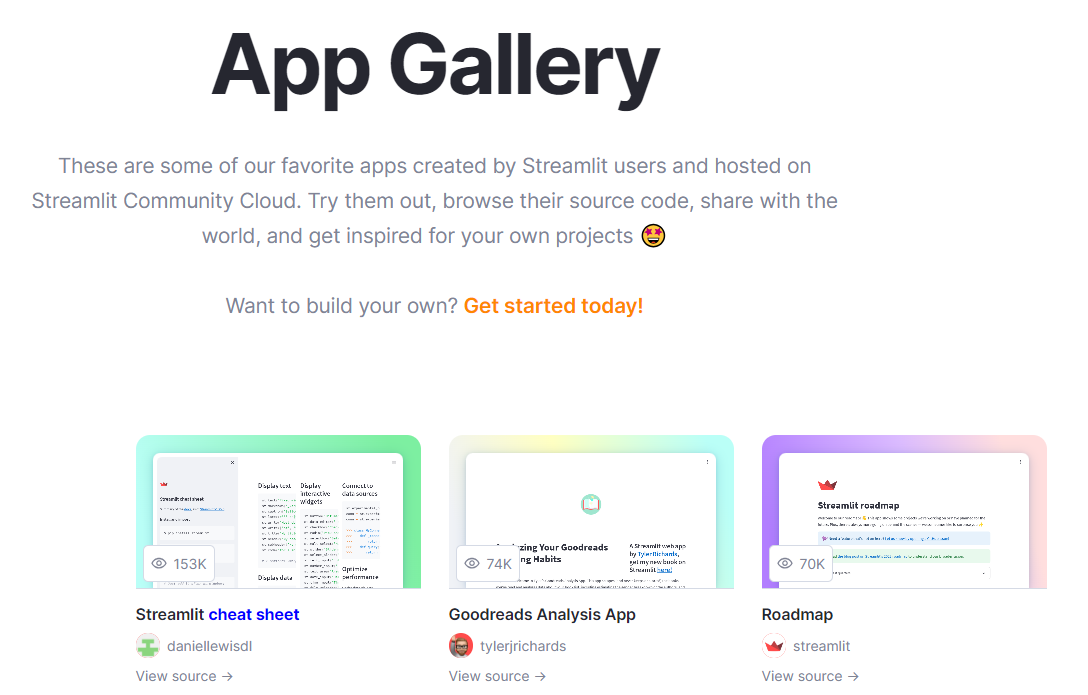
### 스트림릿 공식 홈페이지

<https://streamlit.io/>



### App Gallery

<https://streamlit.io/gallery>



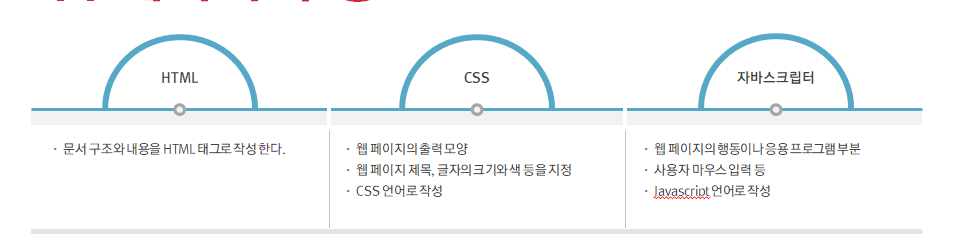
### 스트림릿과 웹 개발 차이점

|  |  |  |
| --- | --- | --- |
| 구분 | STREAMLIT | 웹 개발 |
| 목적 및 사용 사례 | 데이터 시각화, 머신 러닝 모델 시연, 대화형 데이터 애플리케이션 빠르게 작성 | 웹 사이트, 웹 애플리케이션, 전자 상거래 플랫폼 개발 |
| 기술 스택 | 주로 파이썬 언어 사용, 데이터 과학 라이브러리 통합 | HTML, CSS, JavaScript 및 다양한 백엔드 프레임워크 사용 |
| 복잡성 | 대화형 애플리케이션에 중점, 비교적 간단한 작성 방식 | 웹 애플리케이션에 복잡한 구조와 기능 포함 |
| 사용자 수와 규모 | 개별 데이터 과학 프로젝트, 작은 규모 사용자 그룹 | 대규모 사용자 그룹 대상 웹 사이트 및 애플리케이션 개발 |

### STREAMLIT 프레임워크의 특징

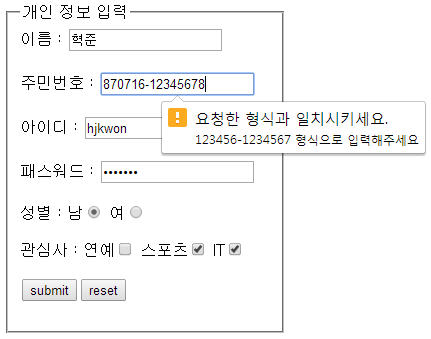
1. 웹 화면에 변경 사항(버튼 클릭 등의 이벤트) 발생시 전체 페이지를 로딩하는 방식이다.
2. 웹 화면 UI 구성을 위해서 콘텐츠를 출력하는 함수형 API와 값을 반환하는 위젯(Widgets)을 제공한다.
3. 전체 페이지를 로딩하는 방식으므로 성능 향상을 위해서 적절히 데이터 캐싱 기능을 이용해야 한다.
4. 전체 페이지 로딩시 필요한 정보는 적절히 세션 객체에 저장해야 한다.

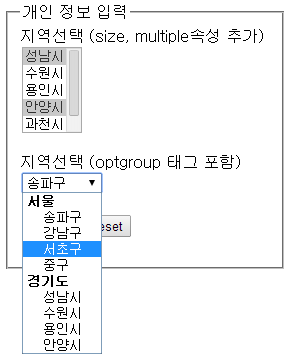
### 일반적인 웹 페이지 구성요소

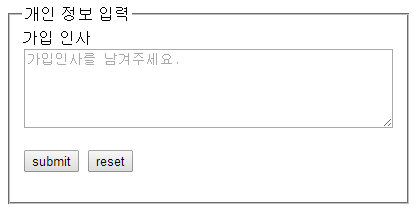




### HTML 폼 요소 예시







## 스트림릿 설치

### streamlit 패키지 설치

(streamlit) C:\PYLABS>pip install streamlit

Collecting streamlit

Downloading streamlit-1.27.2-py2.py3-none-any.whl.metadata (8.1 kB)

Collecting altair<6,>=4.0 (from streamlit)

Downloading altair-5.1.2-py3-none-any.whl.metadata (8.6 kB)

...

Downloading rpds\_py-0.10.6-cp38-none-win\_amd64.whl (185 kB)

---------------------------------------- 185.8/185.8 kB 11.0 MB/s eta 0:00:00

Downloading smmap-5.0.1-py3-none-any.whl (24 kB)

Installing collected packages: pytz, zipp, watchdog, validators, urllib3, tzdata, typing-extensions, tornado, toolz, tom

l, tenacity, smmap, six, rpds-py, pygments, protobuf, pkgutil-resolve-name, pillow, packaging, numpy, mdurl, MarkupSafe,

idna, colorama, charset-normalizer, certifi, cachetools, blinker, backports.zoneinfo, attrs, tzlocal, requests, referen

cing, python-dateutil, pyarrow, markdown-it-py, jinja2, importlib-resources, importlib-metadata, gitdb, click, rich, pyd

eck, pandas, jsonschema-specifications, gitpython, jsonschema, altair, streamlit

Successfully installed MarkupSafe-2.1.3 altair-5.1.2 attrs-23.1.0 backports.zoneinfo-0.2.1 blinker-1.6.3 cachetools-5.3.

1 certifi-2023.7.22 charset-normalizer-3.3.0 click-8.1.7 colorama-0.4.6 gitdb-4.0.10 gitpython-3.1.40 idna-3.4 importlib

-metadata-6.8.0 importlib-resources-6.1.0 jinja2-3.1.2 jsonschema-4.19.1 jsonschema-specifications-2023.7.1 markdown-it-

py-3.0.0 mdurl-0.1.2 numpy-1.24.4 packaging-23.2 pandas-2.0.3 pillow-10.1.0 pkgutil-resolve-name-1.3.10 protobuf-4.24.4

pyarrow-13.0.0 pydeck-0.8.1b0 pygments-2.16.1 python-dateutil-2.8.2 pytz-2023.3.post1 referencing-0.30.2 requests-2.31.0

rich-13.6.0 rpds-py-0.10.6 six-1.16.0 smmap-5.0.1 streamlit-1.27.2 tenacity-8.2.3 toml-0.10.2 toolz-0.12.0 tornado-6.3.

3 typing-extensions-4.8.0 tzdata-2023.3 tzlocal-5.1 urllib3-2.0.7 validators-0.22.0 watchdog-3.0.0 zipp-3.17.0

### streamlit 패키지 설치 확인

(streamlit) C:\PYLABS>conda list

# packages in environment at C:\PYLABS\PY\envs\streamlit:

#

# Name Version Build Channel

rpds-py 0.10.6 pypi\_0 pypi

setuptools 68.0.0 py38haa95532\_0

six 1.16.0 pypi\_0 pypi

smmap 5.0.1 pypi\_0 pypi

sqlite 3.41.2 h2bbff1b\_0

streamlit 1.27.2 pypi\_0 pypi

tenacity 8.2.3 pypi\_0 pypi

toml 0.10.2 pypi\_0 pypi

toolz 0.12.0 pypi\_0 pypi

tornado 6.3.3 pypi\_0 pypi

typing-extensions 4.8.0 pypi\_0 pypi

tzdata 2023.3 pypi\_0 pypi

tzlocal 5.1 pypi\_0 pypi

urllib3 2.0.7 pypi\_0 pypi

validators 0.22.0 pypi\_0 pypi

## 스트림릿 기본 사용법

### Streamlit 실행

(streamlit) C:\PYLABS>streamlit hello

Welcome to Streamlit. Check out our demo in your browser.

Local URL: http://localhost:8501

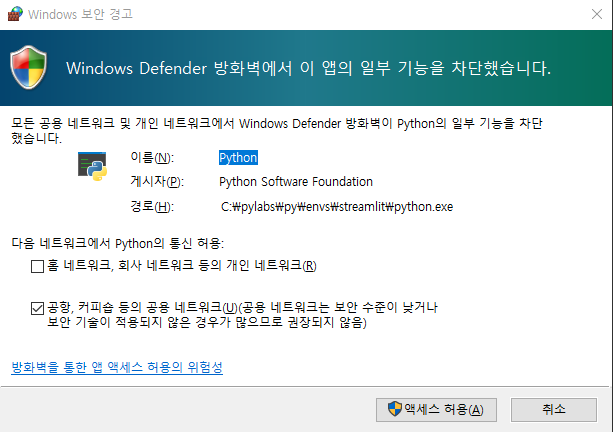
Network URL: http://172.30.1.5:8501

Ready to create your own Python apps super quickly?

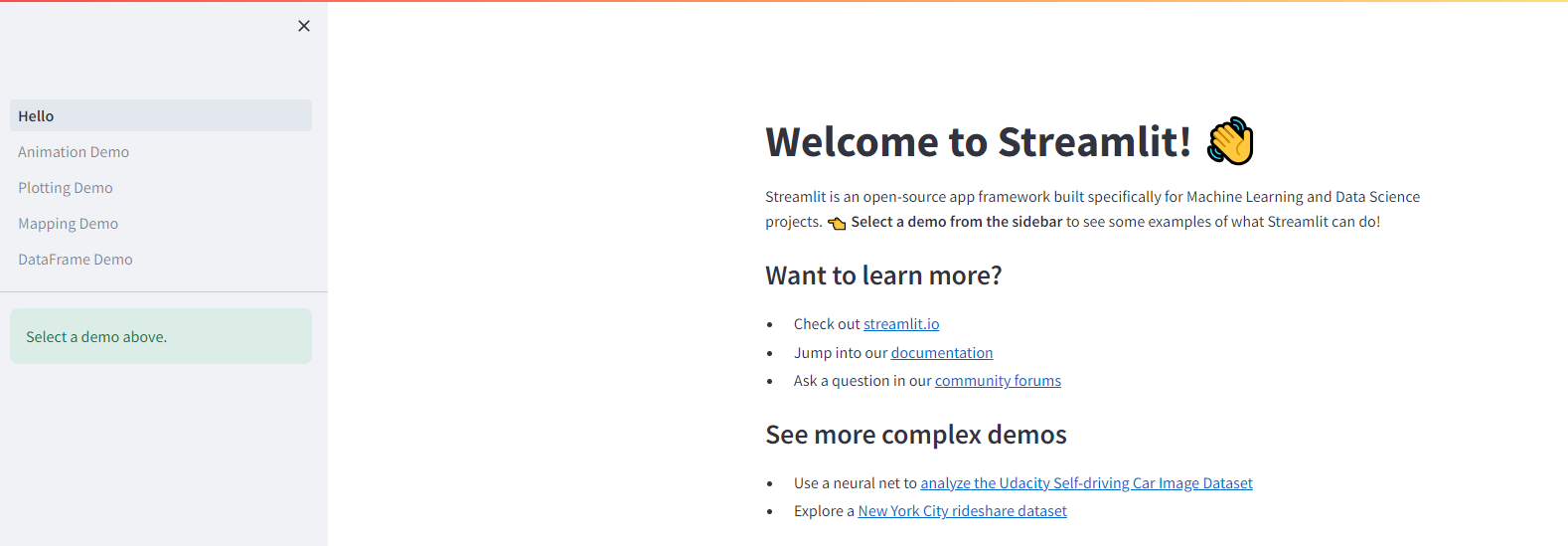
Head over to https://docs.streamlit.io

May you create awesome apps!

### 방화벽 설정 해제

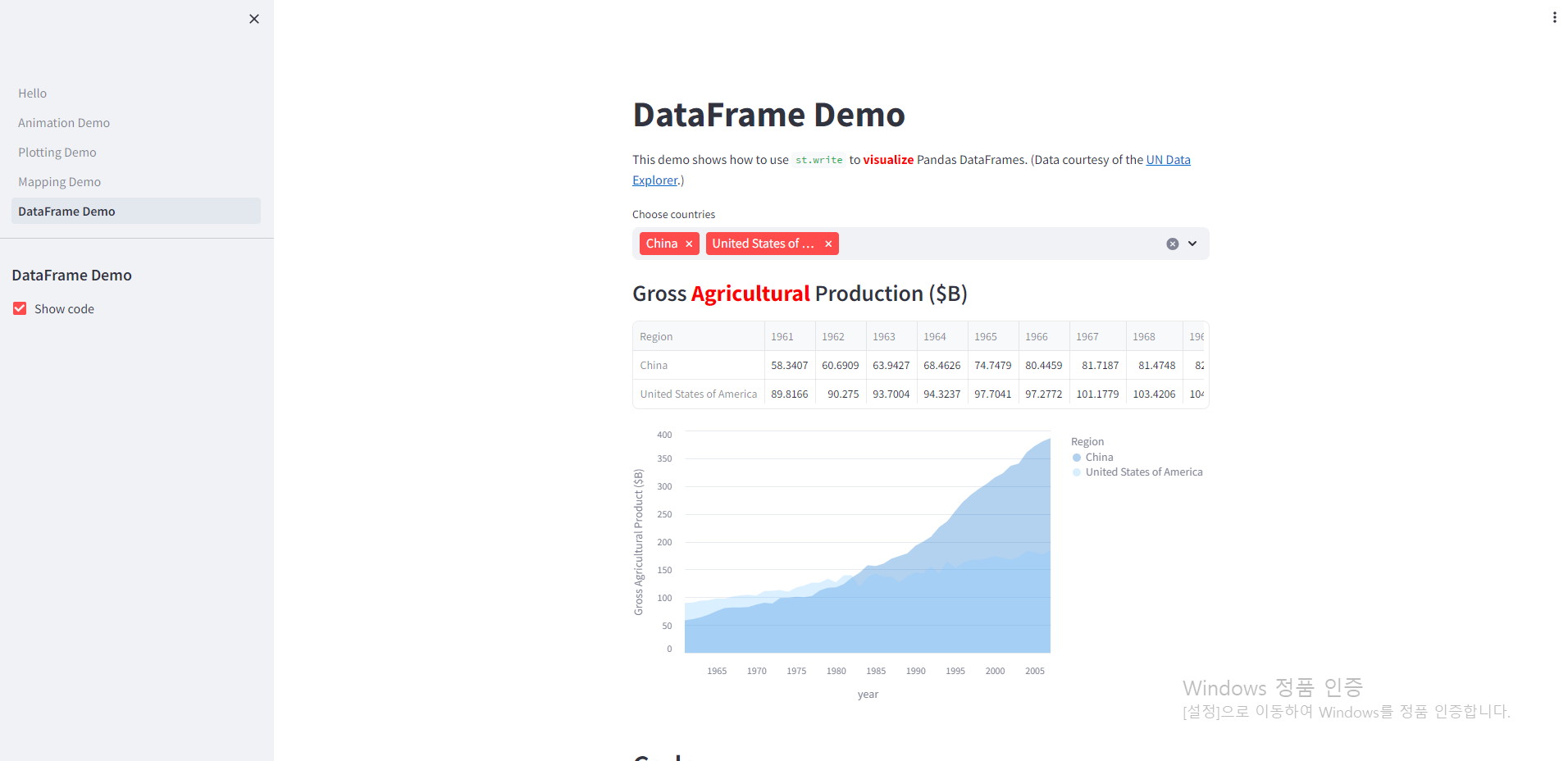


### Streamlit 실행결과 확인

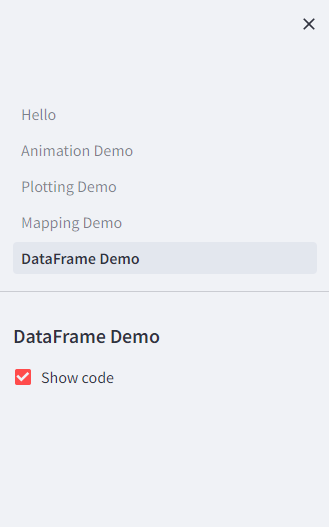


### Streamlit 화면 구성

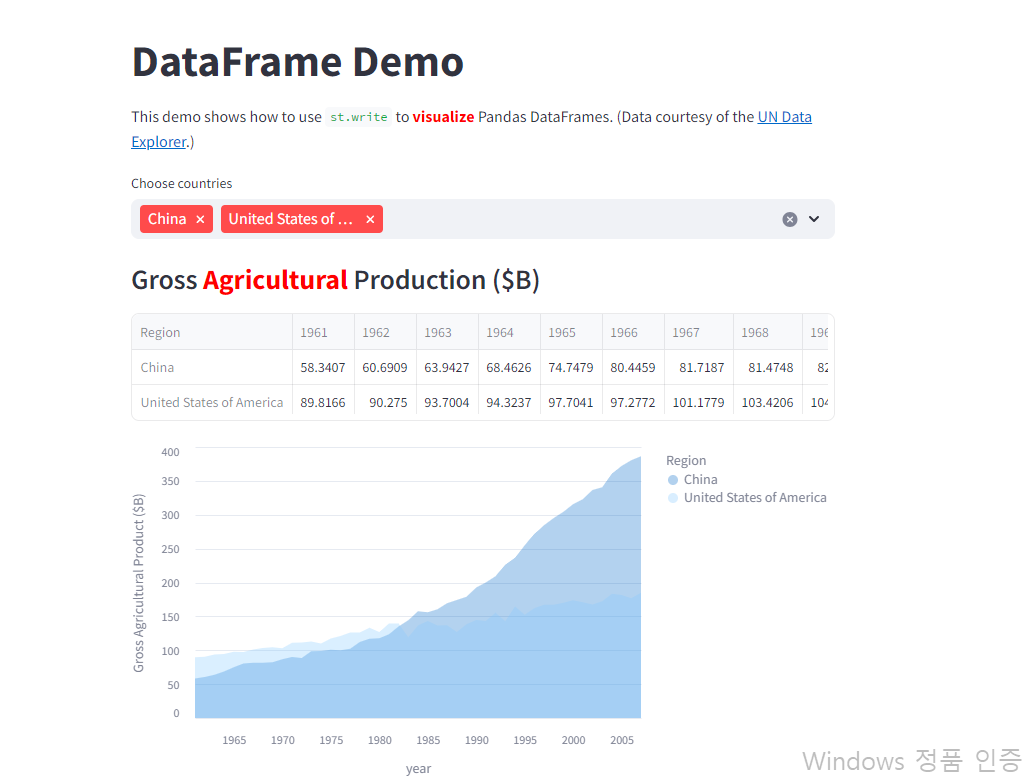
#### 전체화면



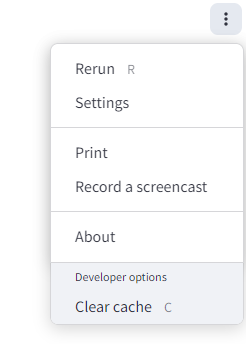
#### 사이드 바 화면



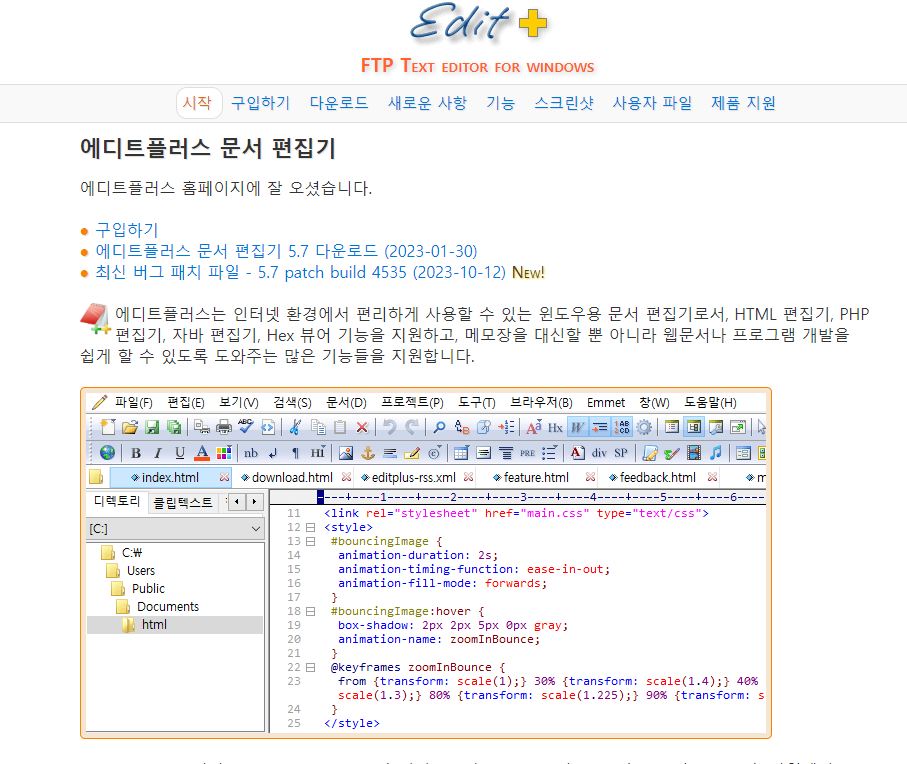
#### 메인 화면(콘텐츠 영역)



#### 환경 설정 메뉴



## 코딩 에디터



### 에디터 플러스 문서 편집기 설치

* 다운로드 및 설치

<https://www.editplus.com/kr/download.html>

# 스트림릿 사용하기

## 스트림릿 API의 구성



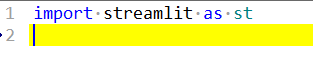
## Creating Our First App

### Creating a Home Page

Streamlit 시작하기 위해서 파이썬 파일 하나가 필요하다. 여기서는 Home.py 파일 생성한다.

파일 최상단에 아래 내용을 추가한다.

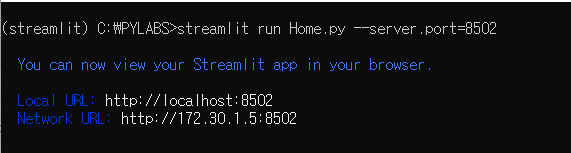
import streamlit as st



파일 생성 후 아래의 명령어 창에 아래의 명령어를 실행한다.

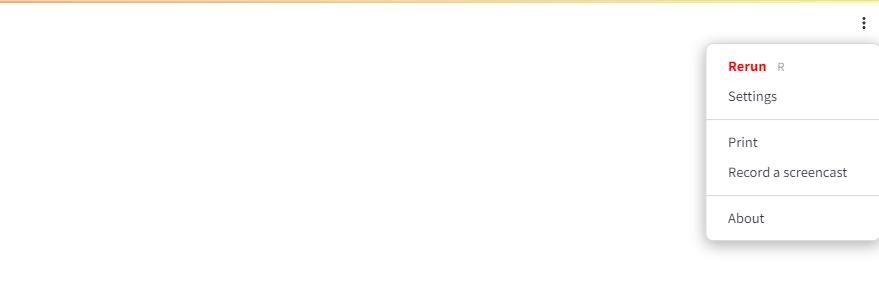
streamlit run Home.py --server.port=8502

아래와 같이 로컬 환경에 서버가 실행된다.



웹 브라우저를 실행해서 아래의 결과를 확인한다.

<http://172.30.1.28:8502/>



### Home.py 내용 수정 해보기

import streamlit as st

st.title('This a Title')

st.code('''

st.title('This a Title')

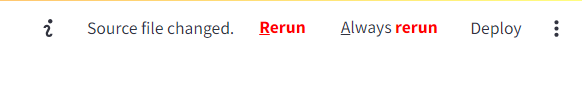
''')

st.header('This a Header')

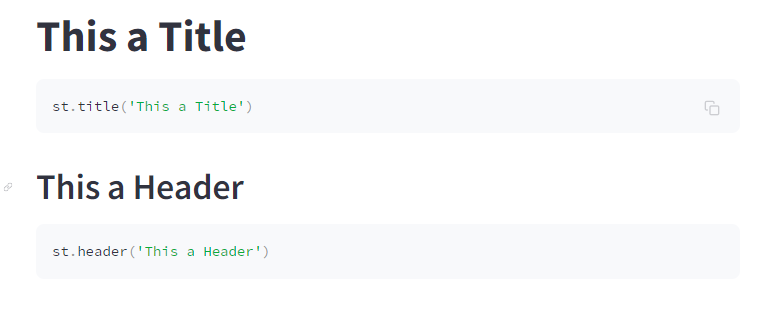
st.code('''

st.header('This a Header')

''')



기존의 Home.py 소스코드 내용을 변경하면 위의 화면이 보입니다. 여기서 F5 또는 Rerun 클릭해서 변경된 화면을 확인합니다.



## Displaying Data in Streamlit

* Displaying Text to Users
  + st.title
  + st.header
  + st.subheader
  + st.write
  + st.caption
  + st.markdown
* Displaying Python Data Structures
  + Data Structures with st.write()
  + Data Structures with st.json()
* Displaying Tabular Data
  + Tabular Data with st.write()
  + Tabular Data with st.dataframe()
  + Tabular Data with st.table()
  + Tabular Data with st.markdown()
* Displaying Multimedia in Streamlit
  + Images
  + Audio
  + Video

### Text 데이터 화면 출력하기

#### st.title

st.title("This a Title.")

#### st.header

st.header("This a Header.")

파이썬 파일에 위치하는 순서대로 화면에 출력된다.

#### st.subheader

st.subheader("This a Subheader.")

#### st.write

st.write("This is text.")

Streamlit’s st.write is quite powerful. As we will see below, it can display data structures, such as lists and dictionaries, as well as entire dataframes, automatically.

#### st.caption

st.caption("This a Caption.")

#### st.markdown

with open("./contents/README.md", "r") as f:

markdown\_text = f.read()

st.markdown(markdown\_text, unsafe\_allow\_html =True)

As we will learn later in this book, st.markdown can also be used to display HTML which makes it even more powerful.

HTML 사용을 허용하려면, unsafe\_allow\_html =True 로 설정해줘야 한다.

README.md 파일은 배포된 자료에 contents 폴더 하위에 있음.

#### 전체 코드 및 결과

import streamlit as st

st.title("This a Title.")

st.code('''

st.title("This a Title.")

''')

st.header("This a Header.")

st.code('''

st.header("This a Header.")

''')

st.subheader("This a Subheader.")

st.code('''

st.subheader("This a Subheader.")

''')

st.write("This is text.")

st.code('''

st.write("This is text.")

''')

st.caption("This a Caption.")

st.code('''

st.caption("This a Caption.")

''')

with open("./contents/README.md", "r", encoding="UTF8") as f:

markdown\_text = f.read()

st.markdown(markdown\_text, unsafe\_allow\_html =True)

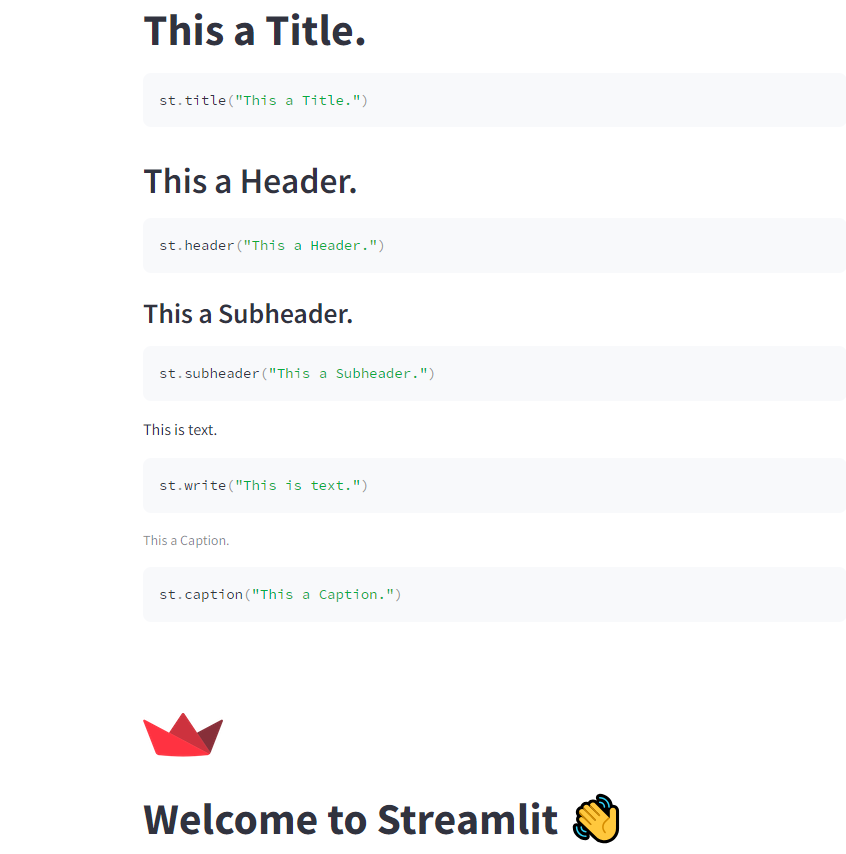
st.code('''

with open("./contents/README.md", "r") as f:

markdown\_text = f.read()

st.markdown(markdown\_text, unsafe\_allow\_html =True)

''')



### Displaying Tabular Data

There are four ways to display tabular data within Streamlit.

* st.write() (defaults to st.dataframe())
* st.dataframe()
* st.table()
* st.markdown()

While on the surface, these may all seem to display the same data, understanding what each does is important so that you can have your application do precisely what you wish.

#### Tabular Data with st.write()

If you are trying to test an application quickly and just want to display tabular data without any extra customization, then st.write() is perfectly suitable. If Streamlit detects a Pandas DataFrame as the object that is being passed to st.write(), it will automatically output that data via st.dataframe().

st.write(df)

#### Tabular Data with st.dataframe()

If you want to have more control over how your tabular data is displayed in your application, you may want to use st.dataframe instead. By using st.dataframe, you can control the width and height of the displayed dataframe.

st.dataframe(df, height=200)

The output will look like this in your application:



With both st.write() and st.dataframe, users will be given a Streamlit dataframe display. This means that they can highlight certain parts of the dataframe, expand cells to read longer text, and sort the data. In other words, the dataframe is an entirely interactive display widget.

#### Tabular Data with st.table()

One of the downsides the st.dataframe display is that the interactivity comes at the cost of aesthetics. If you are working with humanities data, you may have a lot of text in your tables. That text can be difficult for viewers to read in the standard st.dataframe output. In these situations, st.table may be more appropriate.

미관상 보기가 좋다.

st.table(df)

#### Tabular Data with st.markdown()

A key limitation of both the st.table() and st.dataframe is that they do not offer a way to display images. With markdown, we can easily display images within our tables. This, however, comes at the cost of not being able to sort the output. We will learn how to do this later when we work with custom HTML in our Streamlit application. For now, understand that you can convert a Pandas DataFrame to markdown by using the to\_markdown() method.

이미지 출력 지원, 데이터프레임을 마크다운 코드로 변환(to\_markdown())

st.markdown(df.to\_markdown())

##### 전체 코드 및 결과

import streamlit as st

import pandas as pd

# 예제 데이터 생성

data = {

'이름': ['홍길동', '김철수', '이영희', '박영수'],

'나이': [30, 25, 28, 32],

'성별': ['남성', '남성', '여성', '남성']

}

# 데이터 프레임 생성

df = pd.DataFrame(data)

# 데이터 프레임 출력

st.write(df)

st.dataframe(df, height=200)

st.table(df)

st.markdown(df.to\_markdown())

st.code('''

st.write(df)

st.dataframe(df, height=200)

st.table(df)

st.markdown(df.to\_markdown())

''')



### Plotting Basic Graphs with Streamlit

We can plot basic graphs in Streamlit by passing a Pandas dataframe to different chart widgets in Streamlit. The first basic plot we can create is a line chart which we can create with the Streamlit widget st.line\_chart(). We will be working with the Titanic dataset here that we first met in Part Two of this textbook. To prepare the data for visualization, we need to modify it a bit and group everything by the specific value that we want to plot. In our case, we want to visualize the number of survivors for different age groups on the Titanic. We can prepare our dataframe with the code below.

import streamlit as st

import pandas as pd

# 데이터 준비

df = pd.read\_csv("data/titanic.csv")

df = df[["Age", "Survived"]]

chart\_df = df.groupby(["Age"]).sum()

chart\_df["Age"] = chart\_df.index

st.dataframe(df)

st.dataframe(chart\_df)



#### Line Charts with st.line\_chart()

Once we have created our new chart\_df, we can pass it to st.line\_chart(). Here, we will pass the entire dataframe as the first argument and specify our x axis and y axis on the graph. In our case, we want to view the Age column on the x axis and the Survived column on the y axis.

import streamlit as st

import pandas as pd

# 데이터 준비

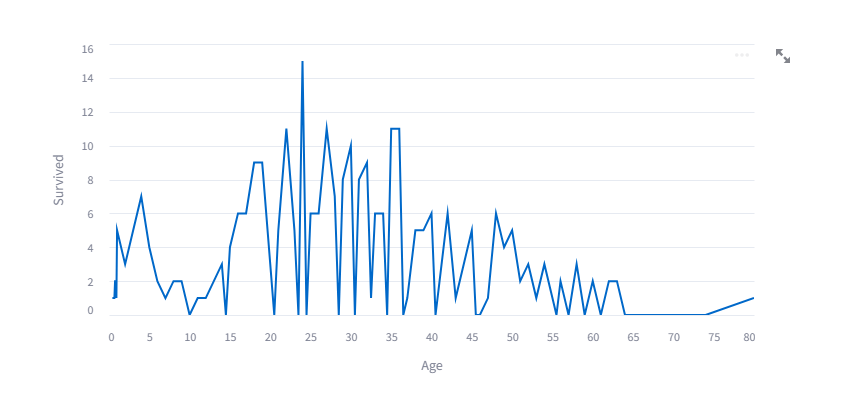
df = pd.read\_csv("data/titanic.csv")

df = df[["Age", "Survived"]]

chart\_df = df.groupby(["Age"]).sum()

chart\_df["Age"] = chart\_df.index

st.line\_chart(chart\_df, x="Age", y=["Survived"])



#### Bar Charts with st.bar\_chart()

Likewise, we can present this same data as a bar\_chart with the widget st.bar\_chart(). This will take the same arguments as above.

import streamlit as st

import pandas as pd

# 데이터 준비

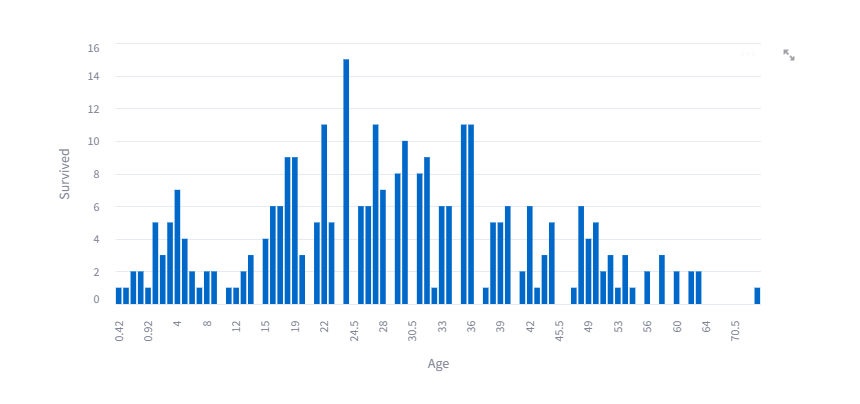
df = pd.read\_csv("data/titanic.csv")

df = df[["Age", "Survived"]]

chart\_df = df.groupby(["Age"]).sum()

chart\_df["Age"] = chart\_df.index

st.bar\_chart(chart\_df, x="Age", y=["Survived"])



#### Area Charts with st.area\_chart()

And finally we can also use the same arguments to create an area chart with the st.area\_char() widget.

import streamlit as st

import pandas as pd

# 데이터 준비

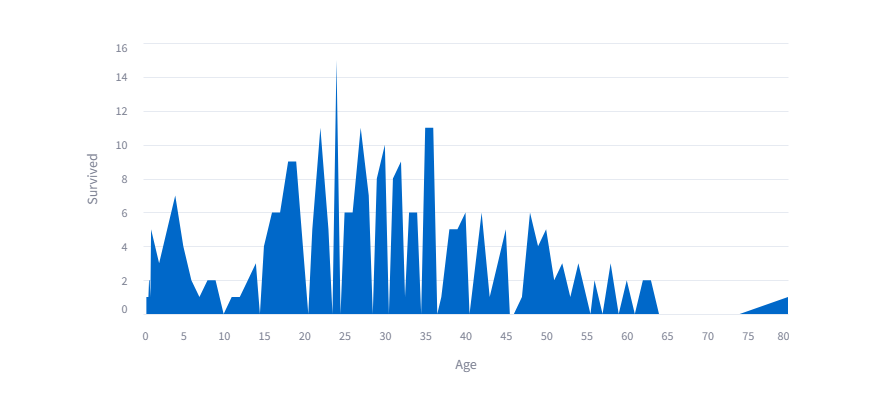
df = pd.read\_csv("data/titanic.csv")

df = df[["Age", "Survived"]]

chart\_df = df.groupby(["Age"]).sum()

chart\_df["Age"] = chart\_df.index

st.area\_chart(chart\_df, x="Age", y=["Survived"])



## Custom HTML

Adding custom HTML in Streamlit allows you to develop more customized applications that fit your need. While it is not always necessary, it is useful to understand how to embed HTML within an application.

Custom HTML can be added via st.markdown(). In order for your HTML to appear on the page, however, you must pass a keyword argument unsafe\_allow\_html=True. This allows the HTML to be rendered. Let’s look at a basic example where we want to display text with a background color of yellow. We can do this by wrapping our text in an a tag in HTML and setting the style’s background color to yellow.

html = """

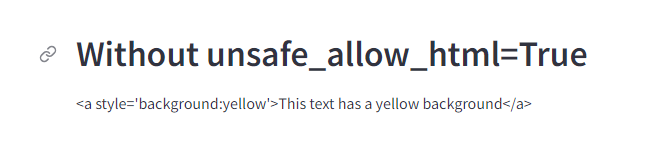
<a style='background:yellow'>This text has a yellow background</a>

"""

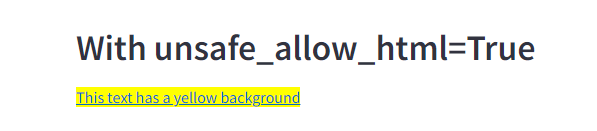
st.header("Without unsafe\_allow\_html=True")

st.markdown(html)

If we do not set unsafe\_allow\_html to True, then our result will look like this:



If we do set it to True, then our result will look like this:



## 스트림릿 위젯

* text input
* numerical input
* date input
* boolean input
* selection input

Cheatsheet for Widgets in Streamlit

|  |  |  |
| --- | --- | --- |
| **Widget** | **Return** | **Description** |
| text\_input | string | A small text region |
| text\_area | string | A large text region |
| number\_input | integer or floa | A typed number input |
| slider | integer or floa | A slider for number input |
| date\_input | timeseries | A calendar for selecting a date |
| time\_input | timeseries | A dropdown menu for selecting a time |
| checkbox | Boolean | A checkbox for marking something as true or false |
| button | Boolean | A button for triggering an event |
| radio | string | A selection for a single option |
| selectbox | string | A selection for a single option (dropdown menu) |
| multiselect | list | A selection for multiple options (dropdown menu) |

* Text Input Widgets
  + st.text\_input()
  + st.text\_area()
* Numerical Input Widgets
  + st.number\_input()
  + st.slider()
* Date and Time Input Widgets
  + st.date\_input()
  + st.time\_input()
* Boolean Input Widgets
  + st.checkbox()
  + st.button()
* Selection Widgets
  + st.radio()
  + st.selectbox()
  + st.multiselect()

### Text Input Widgets

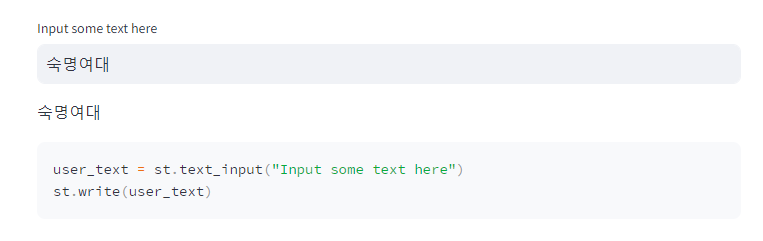
Streamlit offers two ways to allow users to input textual data into an application. You can either use st.text\_input() or st.text\_area(). Both essentially do precisely the same thing, that is, return a string from the user; but each should be used in specific situations. Streamlit’s st.text\_input() is designed for shorter text (such as names, queries, etc.), while st.text\_area() should be used for longer string input data, such as text that can be processed via a spaCy pipeline.

#### st.text\_input()

사용자로부터 짧은 데이터를 입력 받을 때 사용

user\_text = st.text\_input("Input some text here")

st.write(user\_text)



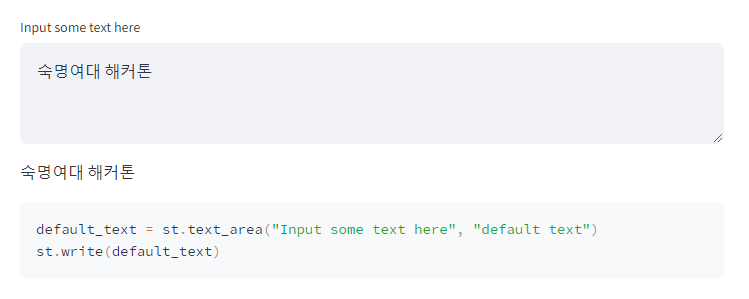
#### st.text\_area()

To create a text area style input, you can use the precise same code, but replace text\_input with text\_area. Both of these classes also let us pass an additional argument for some default text as the second parameter. We can add some default text that will prepopulate the text field with a predetermined string.

default\_text = st.text\_area("Input some text here", "default text")

st.write(default\_text)

첫 번째 파라미터는 라벨, 두 번째는 기본값



### Numerical Input Widgets

While you could let a user input numerical data inside of st.text\_input()or st.text\_area(), this really would be inappropriate. Both of these input options return strings. This means that if you gave the user the ability to input numerical data, you would have to convert it to either an integer or float. In addition to this, the text input widgets do not offer any special keyword arguments that you can pass to the widget that are specific to numerical data, such as minimum value and maximum value.

It is far better in these scenarios to use one of two options for numerical input.

#### st.number\_input()

The first option is st.number\_input(). This feature lets a user input a numerical data that they can tick up or down via a minus or plus sign in the widget. The widget will return an integer or float, depending on how you structure the widget.

Users can also manually type in a specific number. Another feature of this widget is the ability to specify a minimum value (with the min\_value argument) and maximum value (with the max\_value argument). You can also give the user a default value by setting the value argument to a specific number. Finally, you can even provide a step argument which will step up every n-numbers, so a step of 2 would increase the value by two each time the user clicks the plus symbol in the widget.

user\_number = st.number\_input("Input Number",

min\_value=1,

max\_value=10,

value=5,

step=1)

st.write(user\_number)



#### st.slider()

Another way to let a user input data is via the st.slider() widget. This widget will also return an integer or a float, depending on if your values are in decimal form.

slider\_number = st.slider("Select your Number",

min\_value=1,

max\_value=10,

value=5,

step=1)

st.write(slider\_number)



### Date and Time Input Widgets

Working with dates and times is essential in a lot of applications and Streamlit has two widgets for receiving time-series data. Both will require the use of the built-in library datetime, so be sure to import this if you intend to work time time-series data in your application.

#### st.date\_input()

The first widget is st.date\_input(), this allows you to receive a date object which will allow you to structure robust logic, such as finding all data that fall between a start date and end date. When creating the widget, you can simply use the defaults, but if you expect all your data to fall between two dates, it may be wise to set minimum and maximum values. You can do this via the datetime library date method. You should ensure that all values in the st.date\_input() widget conform to the following format:

user\_date = st.date\_input("Select your Date",

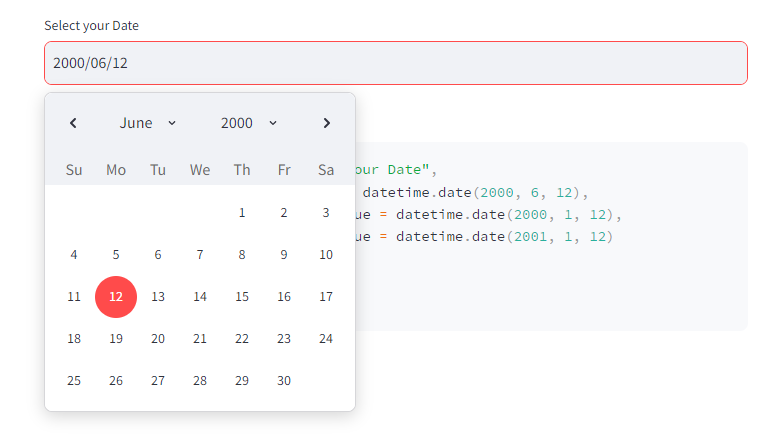
value = datetime.date(2000, 6, 12),

min\_value = datetime.date(2000, 1, 12),

max\_value = datetime.date(2001, 1, 12)

)

st.write(user\_date)



#### st.time\_input()

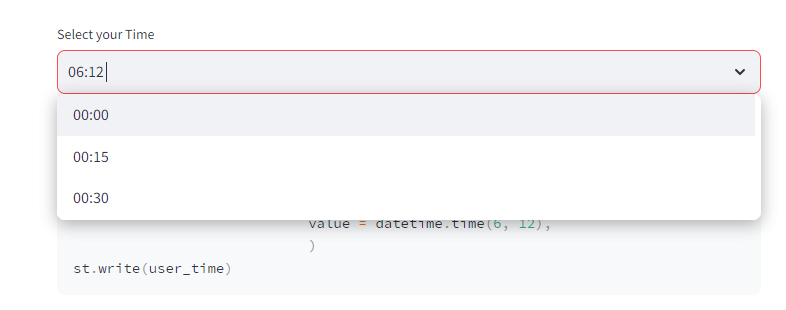
You can do precisely the same thing with time via the st.time\_input() widget. With time, however, we cannot set min or max values. Also, unlike the st.date\_input(), the st.time\_input() will use the time method from the datetime library.

user\_time = st.time\_input("Select your Time",

value = datetime.time(6, 12),

)

st.write(user\_time)



### Boolean Input Widgets

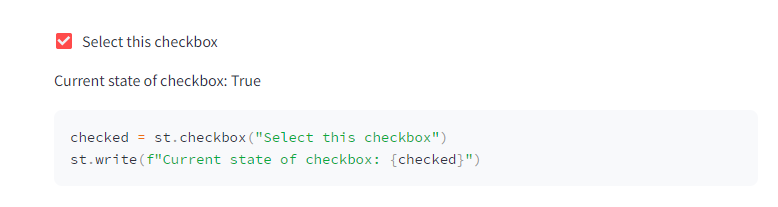
Another important feature for controlling the logic of your application is understanding Boolean (True or False) values from a user input. In Streamlit, we have two ways of using Boolean inputs to control the logic and they both function a bit differently.

#### st.checkbox()

The first is st.checkbox(). This allows us to create a checkbox. Its state can be constantly changed. So a box can be checked or unchecked. As a user changes the state of the checkbox, the Boolean output from the widget will change. We can create a checkbox widget and write out the results with the following lines of code.

checked = st.checkbox("Select this checkbox")

st.write(f"Current state of checkbox: {checked}")

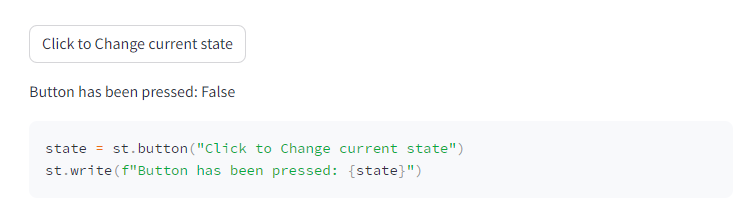


#### st.button()

Unlike the st.checkbox() widget, the st.button() widget will have a continuous state. This means that once the button is clicked, its Boolean output value will forever change unless you specifically change it in your script. In other words, the button’s state at the start of the application is False, but once a user clicks the button, that state will be True continuously. This is important because it means that the click of a button can trigger a one-time event, such as draw a map or run a machine learning model.

state = st.button("Click to Change current state")

st.write(f"Button has been pressed: {state}")



### Selection Widgets

The final collection of important widgets in Streamlit are the selection widgets. These allow you to give users a set of options to choose from. There are three types of selection widgets.

#### st.radio()

The first is st.radio(). This widget allows you to give the user the ability to second one item from a list of options. Only one option can be selected by the user.

options = ["Red", "Blue", "Yellow"]

radio\_selection = st.radio("Select Color", options)

st.write(f"Color selected is {radio\_selection}")



The st.radio() widget will return a string of the selected option.

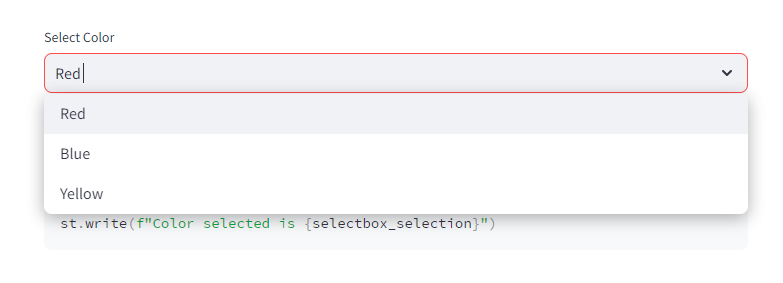
#### st.selectbox()

The same logic holds true for the st.selectbox() widget. The only difference between this and the st.radio() widget is the aesthetic way the options are presented. A selectbox is often more appropriate if you are presenting a user with a larger selection of options, which would be clunky with radio buttons. This will return a string of the selected item.

options = ["Red", "Blue", "Yellow"]

selectbox\_selection = st.selectbox("Select Color", options)

st.write(f"Color selected is {selectbox\_selection}")



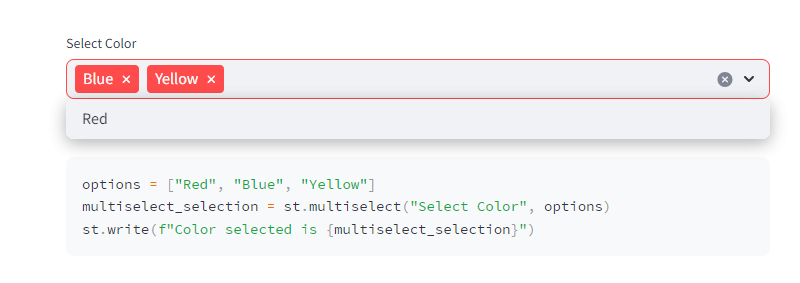
#### st.multiselect()

The final selection widget is the st.multiselect() widget which allows a user to select multiple items from a selectbox. This will return a list of the selected items in the options.

options = ["Red", "Blue", "Yellow"]

multiselect\_selection = st.multiselect("Select Color", options)

st.write(f"Color selected is {multiselect\_selection}")



#### 전체 코드 및 결과

import streamlit as st

import datetime

user\_text = st.text\_input("Input some text here")

st.write(user\_text)

default\_text = st.text\_area("Input some text here", "default text")

st.write(default\_text)

user\_number = st.number\_input("Input Number",

min\_value=1,

max\_value=10,

value=5,

step=1)

st.write(user\_number)

slider\_number = st.slider("Select your Number",

min\_value=1,

max\_value=10,

value=5,

step=1)

st.write(slider\_number)

user\_date = st.date\_input("Select your Date",

value = datetime.date(2000, 6, 12),

min\_value = datetime.date(2000, 1, 12),

max\_value = datetime.date(2001, 1, 12)

)

st.write(user\_date)

user\_time = st.time\_input("Select your Time",

value = datetime.time(6, 12),

)

st.write(user\_time)

checked = st.checkbox("Select this checkbox")

st.write(f"Current state of checkbox: {checked}")

state = st.button("Click to Change current state")

st.write(f"Button has been pressed: {state}")

options = ["Red", "Blue", "Yellow"]

radio\_selection = st.radio("Select Color", options)

st.write(f"Color selected is {radio\_selection}")

options = ["Red", "Blue", "Yellow"]

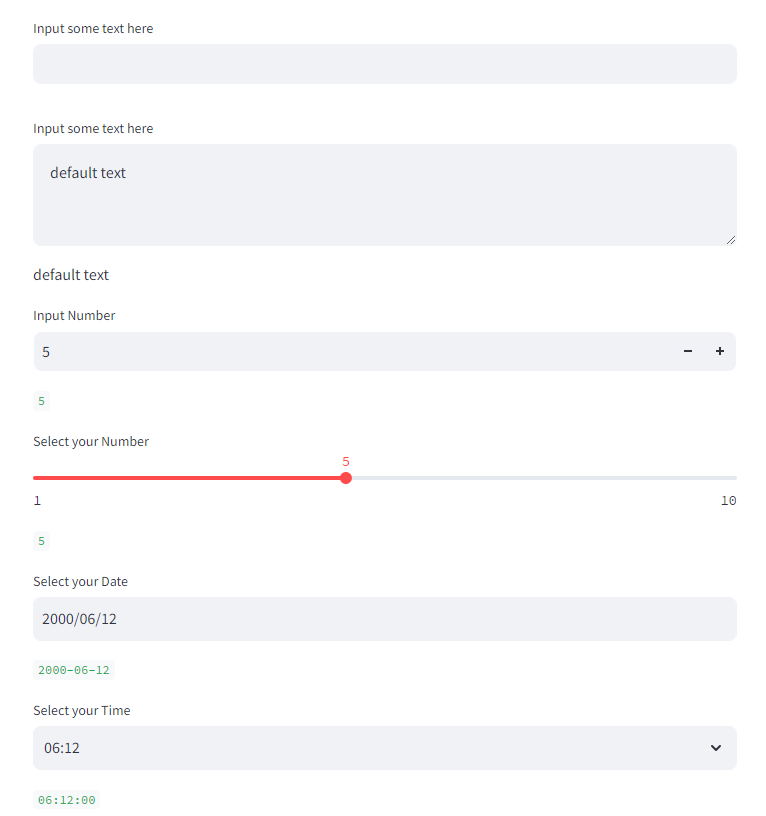
selectbox\_selection = st.selectbox("Select Color", options)

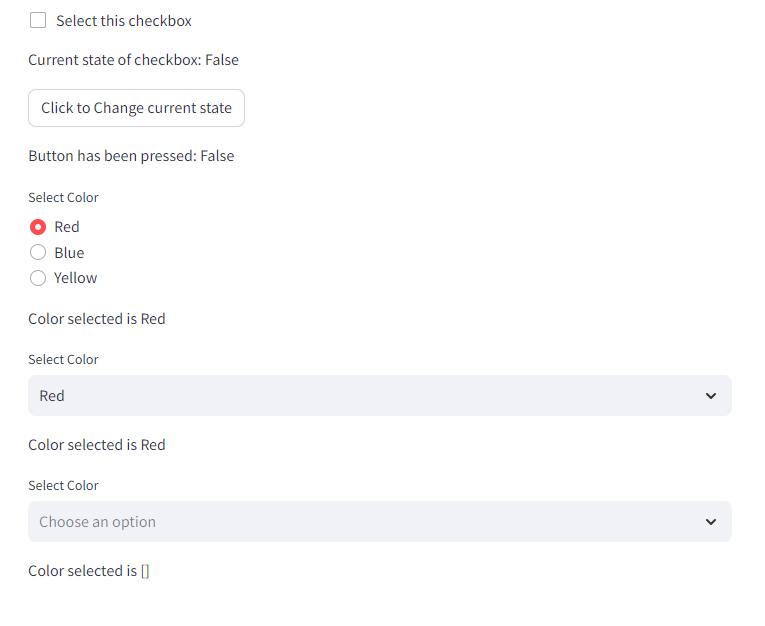
st.write(f"Color selected is {selectbox\_selection}")

options = ["Red", "Blue", "Yellow"]

multiselect\_selection = st.multiselect("Select Color", options)

st.write(f"Color selected is {multiselect\_selection}")





## 레이아웃 (Layout Widgets)

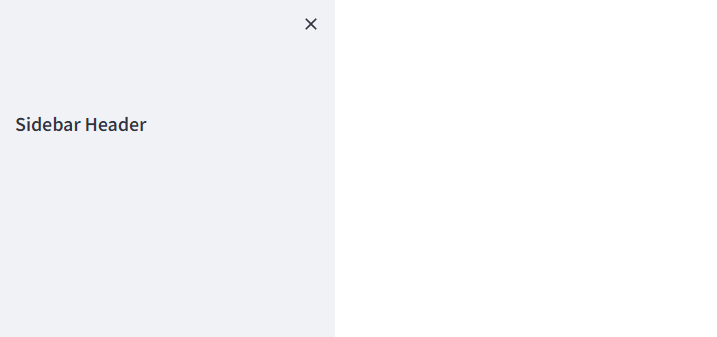
* Sidebar
* Columns
* Expander
* Container
* Tabs
* Empty

### Sidebar

스트릿에서 Sidebar 위젯은 주로 웹사이트나 어플리케이션의 사이드(왼쪽) 영역에 배치되어 네비게이션(메뉴) 및 부가 정보를 제공하는데 주로 사용한다.

import streamlit as st

st.sidebar.header("Sidebar Header")



### Columns

스트릿의 Columns 위젯은 다양한 콘텐츠를 나란히 효과적으로 배치하거나 복잡한 레이아웃을 손쉽게 구성하는 데 사용한다.

import streamlit as st

st.sidebar.header("Sidebar Header")

st.header("Columns")

cols = st.columns(2)

cols[0].write("Column 1")

cols[1].write("Column 2")



### Expander

스트릿의 Expander 위젯은 특정 콘텐츠를 감추거나 확장하여 사용자에게 더 많은 정보를 제공하는 데 사용한다.

import streamlit as st

import pandas as pd

st.sidebar.header("Sidebar Header")

st.header("Columns")

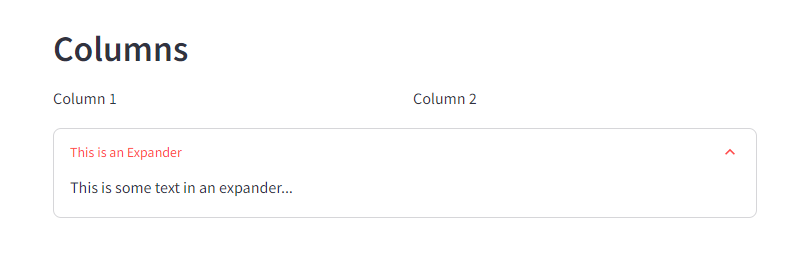
cols = st.columns(2)

cols[0].write("Column 1")

cols[1].write("Column 2")

expander = st.expander("This is an Expander")

expander.write("This is some text in an expander...")



### Container

스트릿의 Container 위젯은 다양한 요소들을 그룹화하여 특정 영역에 배치하고 스타일을 적용하는데 사용한다.

### Tabs

스트림릿(Tabs) 위젯은 여러 탭을 만들어 다양한 섹션 간에 전환하거나 다른 콘텐츠를 효과적으로 구성하는 데 사용한다.

import streamlit as st

import pandas as pd

st.sidebar.header("Sidebar Header")

st.header("Columns")

cols = st.columns(2)

cols[0].write("Column 1")

cols[1].write("Column 2")

expander = st.expander("This is an Expander")

expander.write("This is some text in an expander...")

st.header("Container")

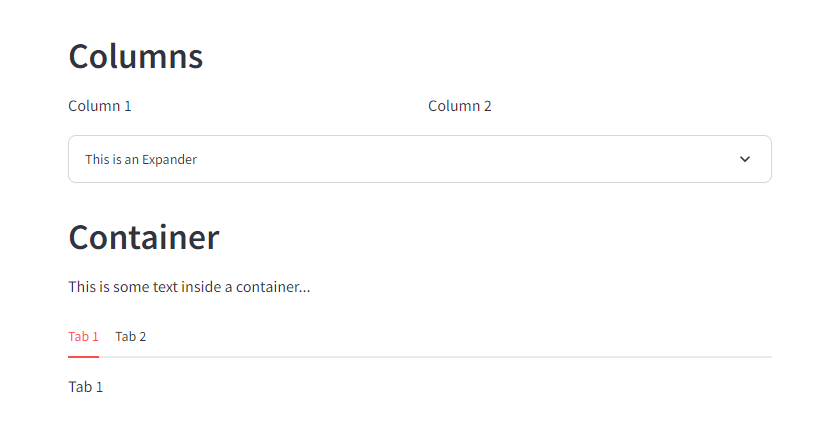
container = st.container()

container.write("This is some text inside a container...")

tabs = st.tabs(["Tab 1", "Tab 2"])

for i, tab in enumerate(tabs):

tabs[i].write(f"Tab {i+1}")



### Empty

스트림릿(Empty) 위젯은 빈 공간을 만들어 특정 레이아웃 조작이나 시각적 효과를 위해 사용한다.

특정 화면을 최신 데이트로 갱신하는 기능을 수행한다.

화면이 감빡이는 문제가 있다.

특정 영역을 실시간으로 갱신해야 하는 경우에 사용하면 된다.

### 코드

import streamlit as st

st.sidebar.header("Sidebar Header")

st.header("Columns")

cols = st.columns(2)

cols[0].write("Column 1")

cols[1].write("Column 2")

expander = st.expander("This is an Expander")

expander.write("This is some text in an expander...")

st.header("Container")

container = st.container()

container.write("This is some text inside a container...")

tabs = st.tabs(["Tab 1", "Tab 2"])

for i, tab in enumerate(tabs):

tabs[i].write(f"Tab {i+1}")

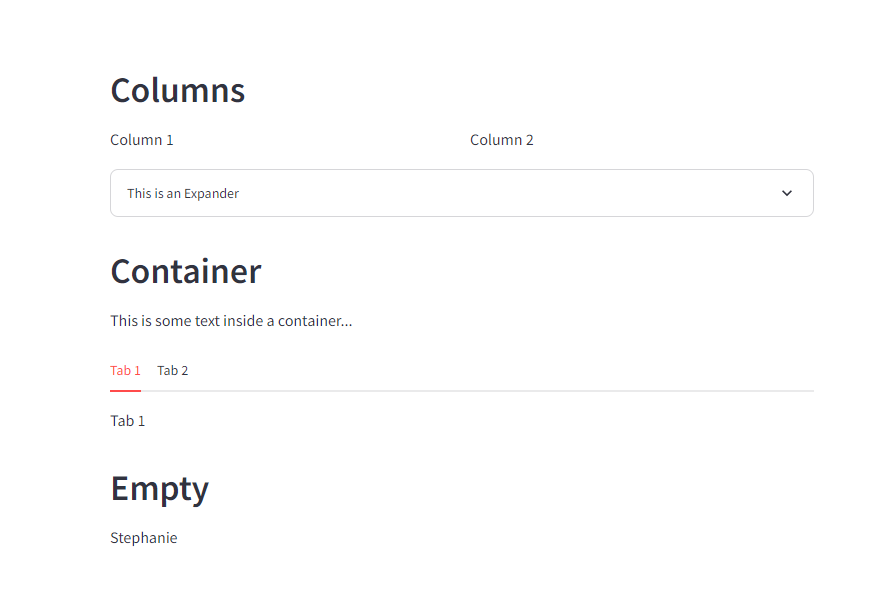
st.header("Empty")

empty = st.empty()

items = ["Tom", "Fred", "Stephanie"]

for item in items:

empty.write(item)



## Storing Data with st.session\_state

Aside from storing large data with cache, we can also store previous states of data with the st.session\_state. The Streamlit Session State gives greater flexibility to an application. It functions as a dictionary that stores data that remains the same during any given state of the app. This means that if your app is rerun by the user because they interacted with the application, then the variable stored in the session state would remain the same.

This is essential for more complex data-driven applications. Let’s consider the simple example that we saw earlier in this chapter when we examined the st.metric() widget.

import streamlit as st

if "prev\_word\_count" not in st.session\_state:

st.session\_state["prev\_word\_count"] = 5

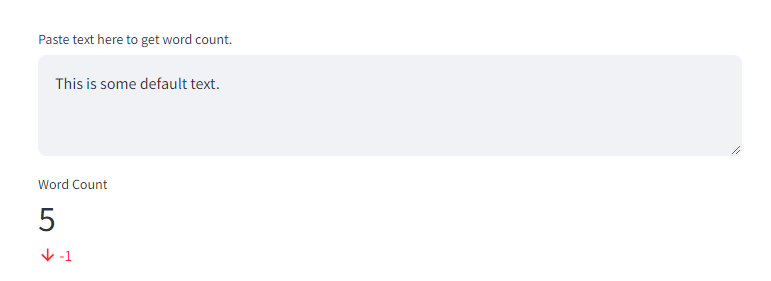
text = st.text\_area("Paste text here to get word count.", "This is some default text.")

word\_count = len(text.split())

change = word\_count - st.session\_state.prev\_word\_count

st.metric("Word Count", word\_count, change)

st.session\_state.prev\_word\_count = word\_count



In this sample, we start off with a conditional:

if "prev\_word\_count" not in st.session\_state:

This line looks to see if a variable name that we want to use is stored in our session state. If it is not stored there, then we want to create that new key. We do that with the following line:

st.session\_state["prev\_word\_count"] = 5

Here we are setting the prev\_word\_count key to 5.

Next, we give the user the ability to input some text for which they wish to receive a word count. In order for the metric to know if the new metric is higher or lower than the previous one, we must store the previous text’s total word count. To do this, we access the previous session state in the final line of the snippet below.

text = st.text\_area("Paste text here to get word count.", "This is some default text.")

word\_count = len(text.split())

change = word\_count-st.session\_state.prev\_word\_count

Once we have populated those results, we then can update the st.session\_state.previous\_word\_count value to the new value. This allows us to always know the state of the previous word count, so that when we display the change value, we know precisely how much our metric has changed.

st.metric("Word Count", word\_count, change)

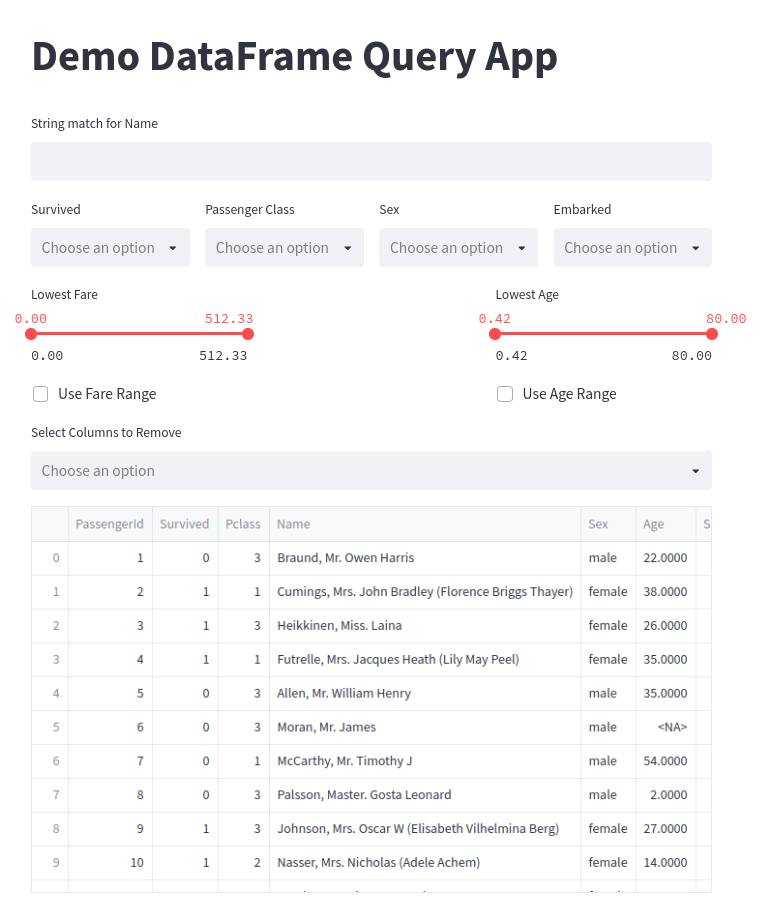
st.session\_state.prev\_word\_count = word\_count

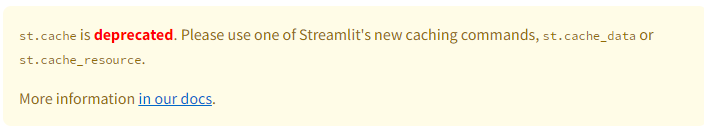
## 차트

# 데모 어플리케이션 개발 및 클라우드 환경 배포

## Building a Database Query Application

Throughout this section, we will be developing an application in Streamlit that looks like this:





We will be working with the following code:

import streamlit as st

import pandas as pd

# Cache our data

@st.cache\_data()

def load\_df():

df = pd.read\_csv("./data/titanic.csv")

# 생존 여부

survival\_options = df.Survived.unique()

# 객실

p\_class\_options = df.Pclass.unique()

# 성별

sex\_options = df.Sex.unique()

# 출발

embark\_options = df.Embarked.unique()

# 요금

min\_fare = df.Fare.min()

max\_fare = df.Fare.max()

# 나이

min\_age = df.Age.min()

max\_age = df.Age.max()

return df, survival\_options, p\_class\_options, sex\_options, embark\_options, min\_fare, max\_fare, min\_age, max\_age

def check\_rows(column, options):

return res.loc[res[column].isin(options)]

st.title("Demo DataFrame Query App")

df, survival\_options, p\_class\_options, sex\_options, embark\_options, min\_fare, max\_fare, min\_age, max\_age = load\_df()

res = df

name\_query = st.text\_input("String match for Name")

cols = st.columns(4)

survival = cols[0].multiselect("Survived", survival\_options)

p\_class = cols[1].multiselect("Passenger Class", p\_class\_options)

sex = cols[2].multiselect("Sex", sex\_options)

embark = cols[3].multiselect("Embarked", embark\_options)

range\_cols = st.columns(3)

min\_fare\_range, max\_fare\_range = range\_cols[0].slider("Lowest Fare", float(min\_fare), float(max\_fare),

[float(min\_fare), float(max\_fare)])

min\_age\_range, max\_age\_range = range\_cols[2].slider("Lowest Age", float(min\_age), float(max\_age),

[float(min\_age), float(max\_age)])

if name\_query != "":

res = res.loc[res.Name.str.contains(name\_query)]

if survival:

res = check\_rows("Survived", survival)

if p\_class:

res = check\_rows("Pclass", p\_class)

if sex:

res = check\_rows("Sex", sex)

if embark:

res = check\_rows("Embarked", embark)

if range\_cols[0].checkbox("Use Fare Range"):

res = res.loc[(res.Fare > min\_fare\_range) & (res.Age < max\_fare\_range)]

if range\_cols[2].checkbox("Use Age Range"):

res = res.loc[(res.Age > min\_age\_range) & (res.Age < max\_age\_range)]

removal\_columns = st.multiselect("Select Columns to Remove", df.columns.tolist())

for column in removal\_columns:

res = res.drop(column, axis=1)

st.write(res)

By the end of this chapter, you will be able to understand and parse each line of this code.

* Importing the Libraries
* Caching Data
* Creating our App Layout
* Using User Inputs to Produce a New DataFrame

### Importing the Libraries

At the start of our Python file, we first import the required libraries. We are using Streamlit for the app development and Pandas for working with our data.

import streamlit as st

import pandas as pd

### Caching Data

Let’s examine the next section of code.

@st.cache()

def load\_df():

df = pd.read\_csv("./data/titanic.csv")

survival\_options = df.Survived.unique()

p\_class\_options = df.Pclass.unique()

sex\_options = df.Sex.unique()

embark\_options = df.Embarked.unique()

min\_fare = df.Fare.min()

max\_fare = df.Fare.max()

min\_age = df.Age.min()

max\_age = df.Age.max()

return df, survival\_options, p\_class\_options, sex\_options, embark\_options, min\_fare, max\_fare, min\_age, max\_age

The first line is:

@st.cache\_data()

This decorator followed by st.cache\_data() establishes that resulting objects from the function that proceeds it should be cached into memory.

The next line begins the creation of our function.

def load\_df():

Once we have defined our function we begin working with our data. First, we load the data:

df = pd.read\_csv("./data/titanic.csv")

Next, our application will leverage four st.multiselect() widgets for four different columns in our dataframe. In order to populate a list of options for users to select, we need to know the unique values of each column. We can grab each unique value with .unique() on each column.

survival\_options = df.Survived.unique()

p\_class\_options = df.Pclass.unique()

sex\_options = df.Sex.unique()

embark\_options = df.Embarked.unique()

Next, our application will also leverage two sliders: one for Age and one for Fare. These will allow a user to find results based on a person’s age or the fare of their ticket. We need to know the max value and the minimum value for each of these so that we can automatically set the slider minimum and maximum values.

min\_fare = df.Fare.min()

max\_fare = df.Fare.max()

min\_age = df.Age.min()

max\_age = df.Age.max()

Finally, we return all of these values so that when the function is called each of these will be returned.

return df, survival\_options, p\_class\_options, sex\_options, embark\_options, min\_fare, max\_fare, min\_age, max\_age

Once we have created this function, we can call it and create all the objects that we need with the following code:

df, survival\_options, p\_class\_options, sex\_options, embark\_options, min\_fare, max\_fare, min\_age, max\_age = load\_df()

We will also create another object whose variable name will be res. This will be the dataframe that gets manipulated by the user and populates the results in the app.

res = df

### Creating our App Layout

After preparing all the data, now comes the time to design the general layout of our application. We use the following code to that:

name\_query = st.text\_input("String match for Name")

cols = st.columns(4)

survival = cols[0].multiselect("Survived", survival\_options)

p\_class = cols[1].multiselect("Passenger Class", p\_class\_options)

sex = cols[2].multiselect("Sex", sex\_options)

embark = cols[3].multiselect("Embarked", embark\_options)

range\_cols = st.columns(3)

min\_fare\_range, max\_fare\_range = range\_cols[0].slider("Lowest Fare", float(min\_fare), float(max\_fare),

[float(min\_fare), float(max\_fare)])

min\_age\_range, max\_age\_range = range\_cols[2].slider("Lowest Age", float(min\_age), float(max\_age),

[float(min\_age), float(max\_age)])

Let’s break down this section of the code. In the first line, we create an object name\_query. This will be a string that is returned from a st.text\_input() widget. We will use this input to query the Name field in the dataframe.

name\_query = st.text\_input("String match for Name")

Next, we will create four columns that we can populate with our st.multiselect() widgets.

cols = st.columns(4)

Now that we have our columns, we can create our four st.multiselect() widgets. The user will be able to select which items that want to see returned for each field in the dataframe. Each of these will return a list of options. We will gather data for four fields: Survived, Pclass, Sex, and Embarked.

survival = cols[0].multiselect("Survived", survival\_options)

p\_class = cols[1].multiselect("Passenger Class", p\_class\_options)

sex = cols[2].multiselect("Sex", sex\_options)

embark = cols[3].multiselect("Embarked", embark\_options)

Next, we need three new columns so that we can populate our two range sliders. We are using three columns here so that there is a large gap between the two sliders.

range\_cols = st.columns(3)

We will populate the first and last slider, we will place two st.slider() widgets. We will use the minimum and maximum values for the Age and Fare fields in the dataframe.

min\_fare\_range, max\_fare\_range = range\_cols[0].slider("Lowest Fare", float(min\_fare), float(max\_fare),

[float(min\_fare), float(max\_fare)])

min\_age\_range, max\_age\_range = range\_cols[2].slider("Lowest Age", float(min\_age), float(max\_age),

[float(min\_age), float(max\_age)])

### Using User Inputs to Produce a New DataFrame

With the general layout designed, we can then work with the input from the user to modify our res dataframe. The following code manipulates the dataframe through a set of conditions we generate from the user input.

if name\_query != "":

res = res.loc[res.Name.str.contains(name\_query)]

if survival:

res = check\_rows("Survived", survival)

if p\_class:

res = check\_rows("Pclass", p\_class)

if sex:

res = check\_rows("Sex", sex)

if embark:

res = check\_rows("Embarked", embark)

if range\_cols[0].checkbox("Use Fare Range"):

res = res.loc[(res.Fare > min\_fare\_range) & (res.Age < max\_fare\_range)]

if range\_cols[2].checkbox("Use Age Range"):

res = res.loc[(res.Age > min\_age\_range) & (res.Age < max\_age\_range)]

removal\_columns = st.multiselect("Select Columns to Remove", df.columns.tolist())

for column in removal\_columns:

res = res.drop(column, axis=1)

st.write(res)

First, we check to see if the user has written anything in the st.text\_input() widget. If it is, then we will narrow the dataframe down to anything that matches the user’s string.

if name\_query != "":

res = res.loc[res.Name.str.contains(name\_query)]

For each of the st.multiselect() widget inputs, we check to see if the user has selected how to narrow the dataframe:

if survival:

res = check\_rows("Survived", survival)

if p\_class:

res = check\_rows("Pclass", p\_class)

if sex:

res = check\_rows("Sex", sex)

if embark:

res = check\_rows("Embarked", embark)

Next, we will use the input from the minimum and maximum values of the sliders for Age and Fare. So that we do not ignore results where Age or Fare are NaN in the dataset, we want to give the user the ability to check an st.checkbox() widget. This will allow the user to activate or deactivate the sliders.

if range\_cols[0].checkbox("Use Fare Range"):

res = res.loc[(res.Fare > min\_fare\_range) & (res.Age < max\_fare\_range)]

if range\_cols[2].checkbox("Use Age Range"):

res = res.loc[(res.Age > min\_age\_range) & (res.Age < max\_age\_range)]

We also want to give the user one final input, the ability to narrow down and delete columns from the dataset. The reason for this is because not all fields will be relevant to every user and since this is a large dataset, it makes sense to give them the ability to limit which fields they are seeing to the ones they want.

removal\_columns = st.multiselect("Select Columns to Remove", df.columns.tolist())

for column in removal\_columns:

res = res.drop(column, axis=1)

As we have manipulated the res dataframe throughout each of these lines, we are finally ready to display the data:

st.write(res)

## 클라우드 환경 배포

### Deploying an App in the Cloud with Streamlit Share

Once you have designed an application and have tested it locally, it comes time to share it with others. We can do this with Streamlit by leveraging several different cloud-based services. Fortunately, Streamlit offers a free Streamlit Share service that lets users share their apps for free.

* Create a GitHub Account
* Upload Application to GitHub
* Connect Streamlit Share to your GitHub
* Create a New App
* Set Custom Subdomain

### Create a GitHub Account

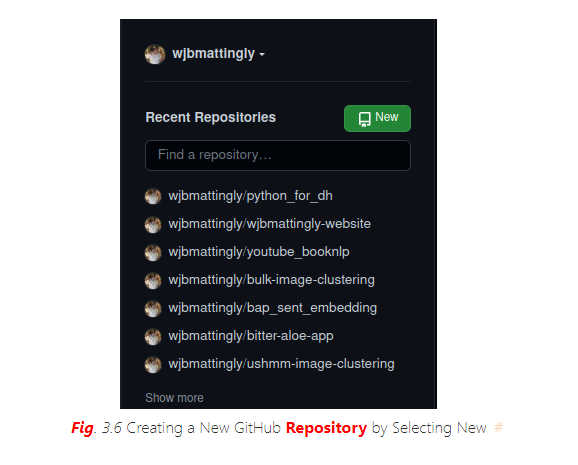
In order to deploy your application, you will need a place to store your application’s source code. The easiest way to do this is via GitHub which Streamlit and many other application hosting services support.

### Upload Application to GitHub

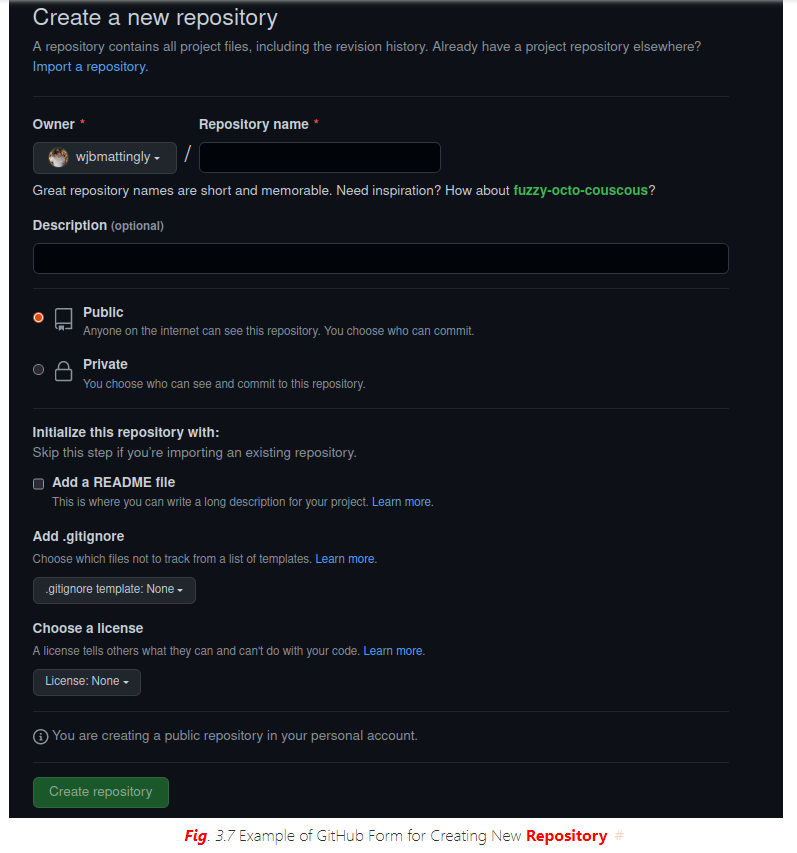
Once you have created your GitHub account, you can create a repository and upload the code into the repository. If you are just starting out with GitHub, getting used to Git can be a bit daunting. Git is powerful, but has a steep learning curve. It allows you to maintain your code, especially in teams, and version control everything. This means that your code is always backed up and you can access previous versions of your code at any single time.

To upload your application formally, you would want to be familiar with Git and how to perform basic tasks, such as cloning a repository, adding files to it, committing those changes, and then pushing them to the repository. While this is the better approach, it is not the only one. For now, you can get your application up and running entirely in your browser.

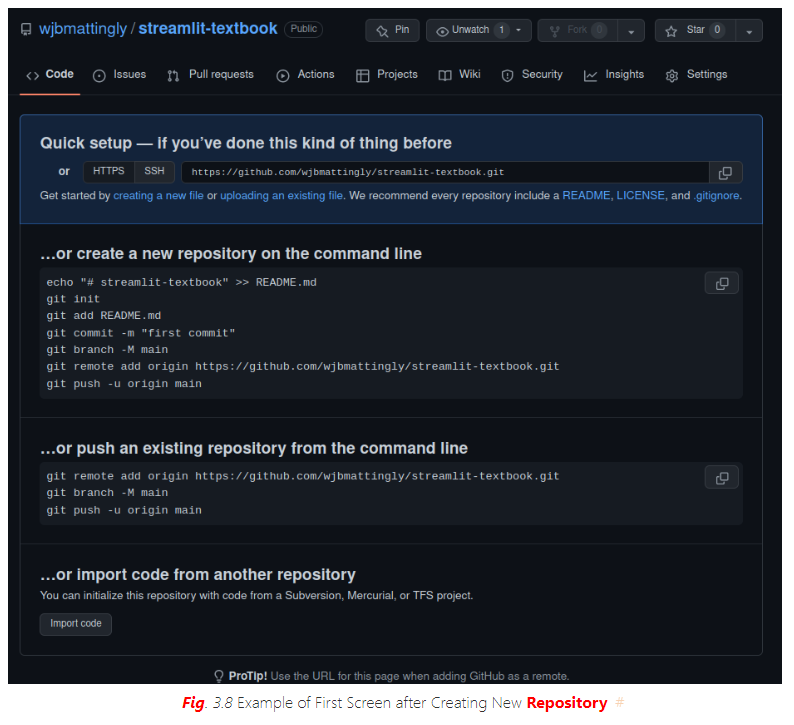
First, we will create a new repository. To do this, we will go to our GitHub main page and select New, found in the top-left side of your screen.



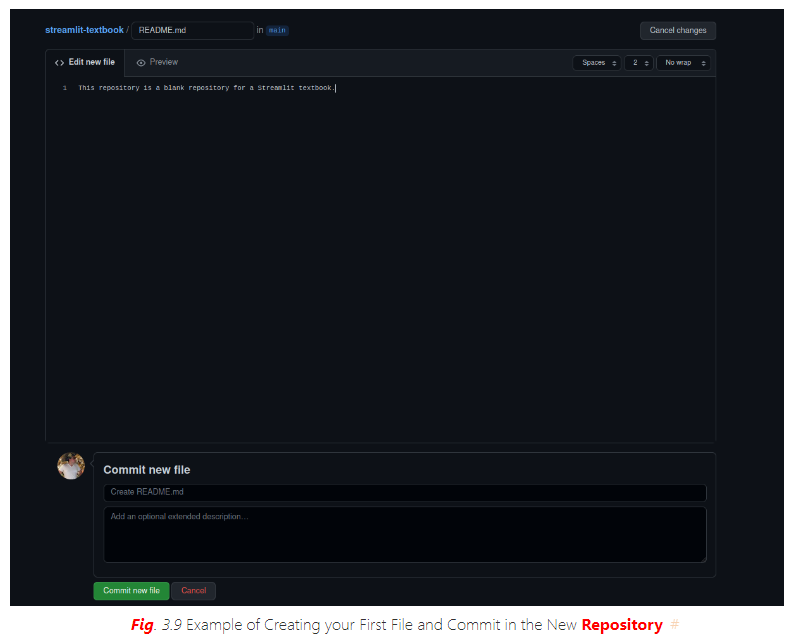
Once you click New, you will see the following screen:



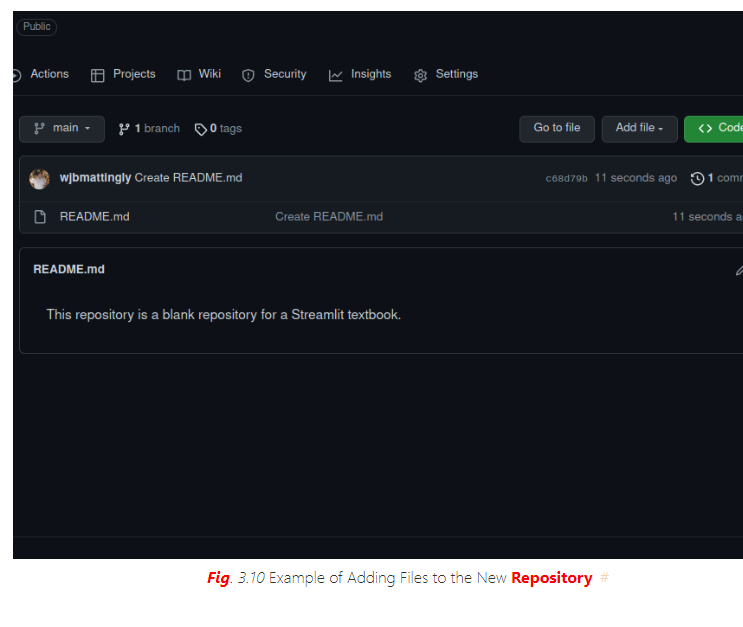
Here, we will fill out the details of our repository. We need to first give it a unique name. We can keep all other settings the default for now. Once done, we will click Create repository. After this, you will see a new screen that looks like this:



You will now select creating a new file. Once you do, you will see a screen that looks like this:



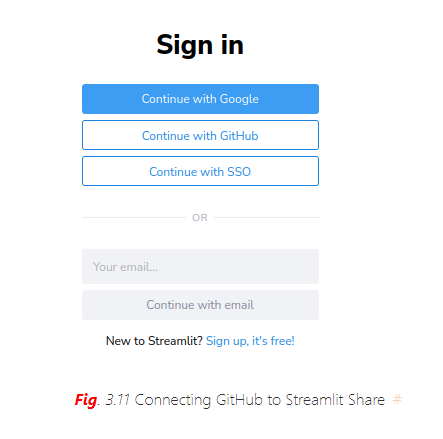
Here, you will create a file called README.md and assign some text to it. This will ideally describe your application. It will be what users see when they visit your GitHub repository. Once you have a description you are happy with, click Commit new file. This will lead to a final new screen.



Next, we need upload our application and data to our repository. To do this, we will click Add file and upload our local files that are necessary for running our Streamlit application. You will also want to make a requirements.txt file lists all required libraries, such as Pandas.

### Connect Streamlit Share to your GitHub

Now that we have our repository created, we need to link Streamlit Share to our GitHub. To do so, you will need to visit https://share.streamlit.io/ Once on this page, you will see a screen that looks like this:

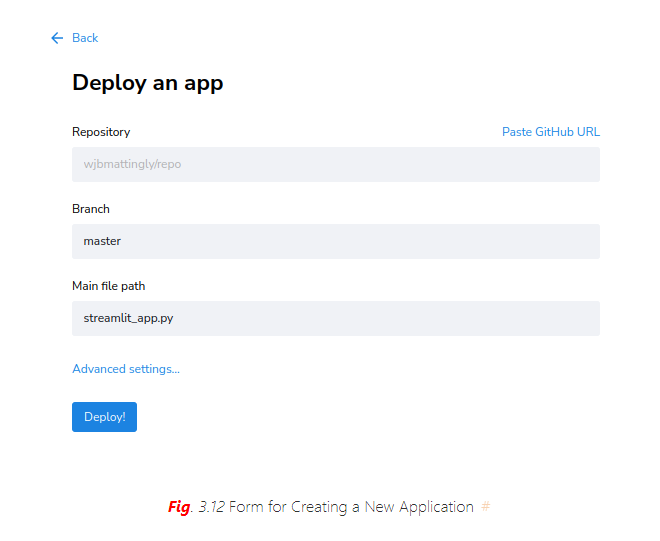


Select Continue with GitHub and use your GitHub credentials for login verification. Once you are logged in, you will see a screen that looks like this at the top:

````{figure} `../images/streamlit/streamlit\_mainpage.png Main Screen of Streamlit Share

### Create a New App

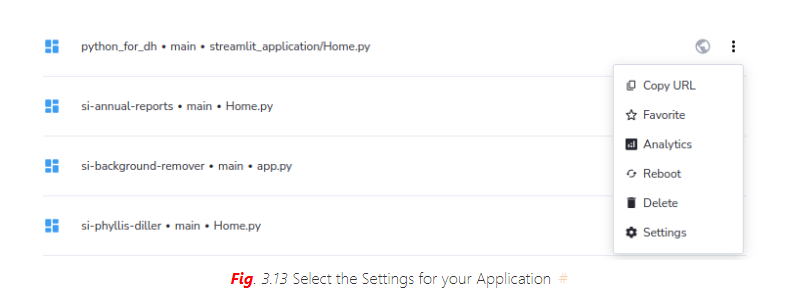
Select New App. After a few seconds, you will be taken to a page that looks like this:



Click on each of these fields. First, select your repository. Streamlit will be connected to your GitHub, so you will be able to see all your repositories, both private and public. Next, select the branch of your repository. We have not covered branches in this textbook. Branches are a function of Git. By default, your application will be on the main branch of your repository. Finally, select the main Python file in which your application rests. Once you have filled out these three fields, click Deploy!.

### Set Custom Subdomain

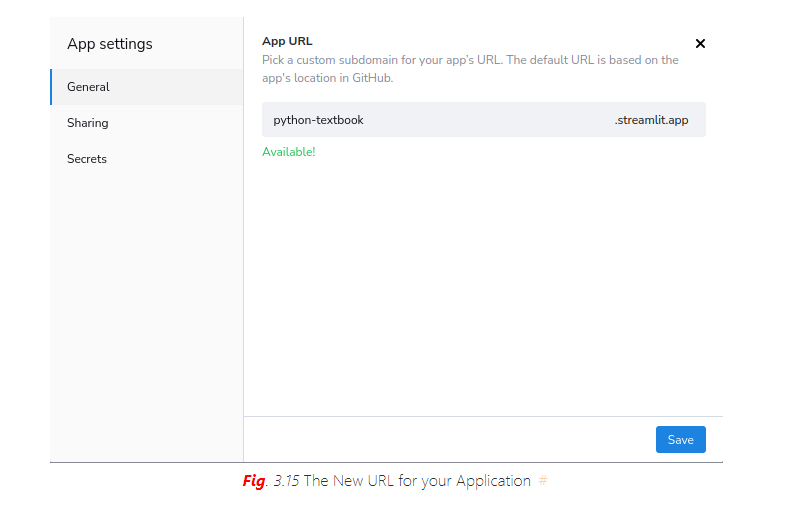
It will take several minutes (depending on the complexity of your application and the number of required libraries you wish to have installed). At this point, Streamlit Share is building an environment on a server to host your application in the cloud. Once complete, it will provide you with a unique domain for your application with the extension. This will be based on your repository name and your GitHub username. We can create a custom subdomain for free by returning to our Streamlit Share homepage and selecting the three dots next to our application’s name.



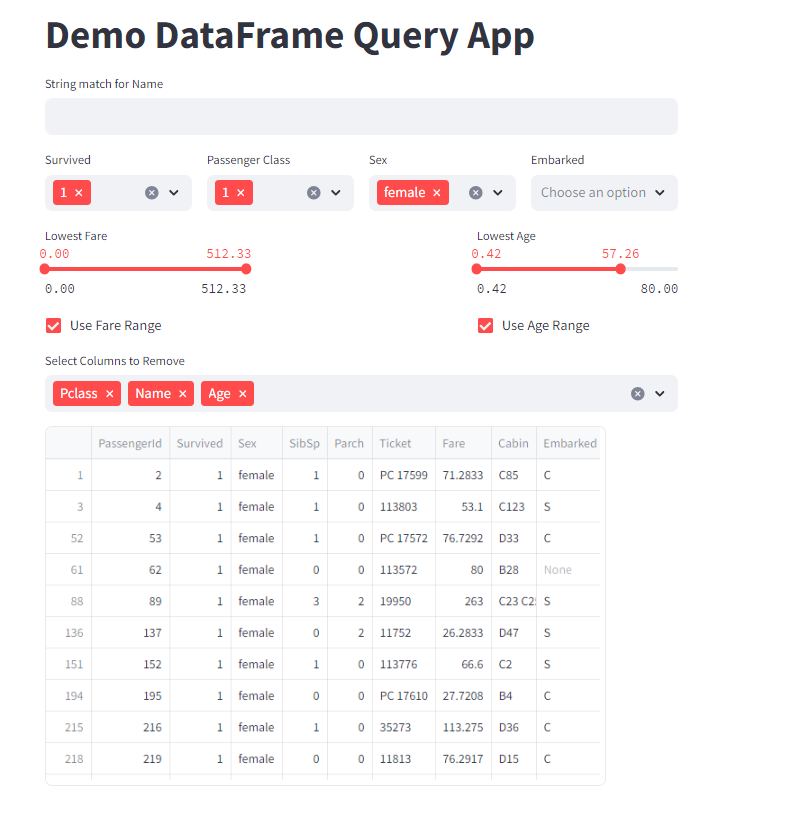
Once here, select Settings and you will see a page that looks like this:



Note that in the middle of the screen, you will see your long subdomain. You can now change it to something unique and easier to remember for distributing to users. After you change your subdomain, click Save.



Now, your app is in the cloud with a unique and easy-to-remember subdomain!



# 참조문헌

<https://python-textbook.pythonhumanities.com/05_streamlit/05_01_02_streamlit.html>

<https://github.com/FelipeAdeT/PythonforHumanities/tree/master>