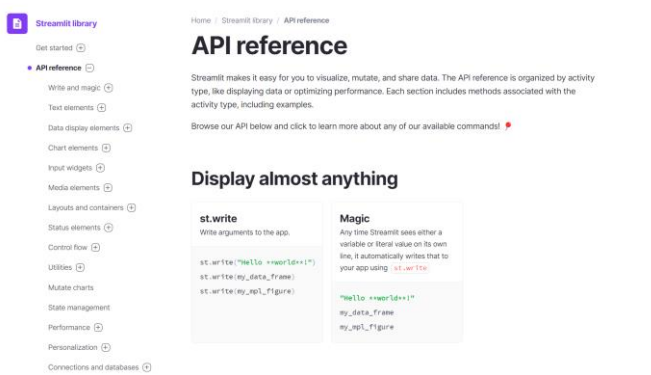
## 3. Streamlit 주요 위젯

본 장에서는 프로젝트에서 사용한 위젯 위주로 간단하게 설명하려고 한다. 각 위젯의 화면을 먼저 보여주고 화면 아래에 코드를 보여주는 식으로 구성하였다. 실제 강의 시, 일반적으로 app.py 파일를 생성하고 하나씩 나열하면서 코드 실행 후, 업데이트 화면을 계속적으로 보여주면서 진행하였지만, 본 장에서는 각 위젯별로 파일을 만들었다. 파일을 실행할 때는 파일이 있는 곳에서 다음과 같이 실행한 후, http://localhost:8501에서 확인한다.

```
/ch04 (main) $ streamlit run app file_name.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://192.168.45.246:8501
```

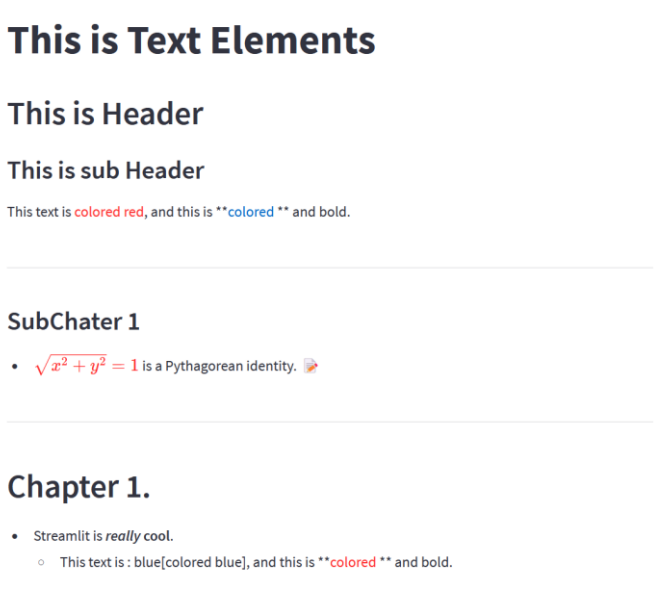
주요 위젯에 대한 설명은 공식문서에 잘 나와 있다. 공식문서에서는 모든 위젯 설명에 대해 크게 아래와 같이 구성되어 있다.[66] 앞으로 진행할 주요 코드는 기본적으로는 공식 문서의 코드를 활용하고 추가적인 부분은 필자가 응용하였다.




[66] https://docs.streamlit.io/library/api-reference

### A. Text Elements

여기에서는 st.title(), st.header(), st.subheader(), st.markdown()만 보여준다. 특히, 유심히 살펴봐야 하는 것은 st.markdown()를 잘 활용하면 매우 유용하게 텍스트를 표시할 수 있다.




코드는 아래와 같이 작성한다.

파일위치 : ch05/01_text.py

```
# -*- coding:utf-8 -*-
import streamlit as st

def main():
    st.title("This is Text Elements")
    st.header("This is Header")
    st.subheader("This is sub Header")
```

```
    st.markdown("This text is :red[colored red], and this is
**:blue[colored] ** and bold.")
    st.write("-" * 50)
    st.markdown("""
    ### SubChater 1
     - :red[$\sqrt{x^2+y^2}=1$] is a Pythagorean identity. :pencil:
    """)
    st.write("-" * 50)
    st.markdown("## Chapter 1. \n"
                "- Streamlit is **_really_ cool**.\n"
                "   * This text is : blue[colored blue], and this is
**:red[colored] ** and bold.")


if __name__ == "__main__":
    main()
```

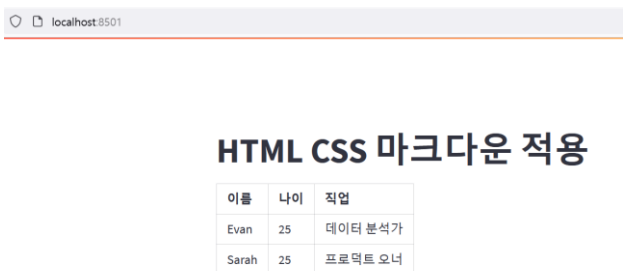
공식문서를 확인해보면 마크다운을 작성할 때 기본 문법은 github의 양식을 따라간다.[67] 문서에 따르면 github 마크다운 양식은 크게 emoji 단축키를 지원한다. 단축키에 관한 설명은 『Streamlit emoji shortcodes』에서 확인한다.[68] 마크다운에서 수식을 작성할 때 LaTeX를 이용한다. 보통 표현식을 '$' or '$$'로 감싸서 표현한다. 지원되는 LaTeX 함수는 각주를 참고한다.[69] 컬러 텍스트는 :color[색칠할 텍스트] 구문을 사용하며, 여기서 색상은 파란색, 녹색, 주황색, 빨간색, 보라색 중 지원되는 색상으로 대체해야 한다.

[67] https://github.github.com/gfm/

[68] https://streamlit-emoji-shortcodes-streamlit-app-gwckff.streamlit.app/

[69] https://katex.org/docs/supported.html

이번에는 HTML과 CSS를 markdown에 적용하도록 해본다. 화면부터 확인한다.




코드를 보면 다음과 같다.

파일위치 : ch05/02_html_css_markdown.py

```
# -*- coding:utf-8 -*-
import streamlit as st

def main():

    st.title("HTML CSS 마크다운 적용")
    html_css = """
    <style>
      th, td {
        border-bottom: 1px solid #ddd;
      }
    </style>

    <table>
      <thead>
        <tr>
          <th>이름</th>
          <th>나이</th>
          <th>직업</th>
        </tr>
```

```
      </thead>
      <tbody>
        <tr>
          <td>Evan</td>
          <td>25</td>
          <td>데이터 분석가</td>
        </tr>
        <tr>
          <td>Sarah</td>
          <td>25</td>
          <td>프로덕트 오너</td>
        </tr>
      </tbody>
    </table>
    """
    st.markdown(html_css, unsafe_allow_html=True)

if __name__ == "__main__":
    main()
```

HTML과 CSS를 활용하여 테이블을 만들었다. 그런데, 이 코드를 마크다운에 추가하는데, html_css 변수로 저장하여 전달한 것이다. 사용법은 기존 HTML과 CSS를 안다면, 작업하는 것은 쉽다. 그런데, 여기에서 unsafe_allow_html의 의미는 공식문서에서 다음과 같이 설명한다. 기본적으로 본문에서 발견되는 모든 HTML 태그는 이스케이프 처리되므로 일반 텍스트로 취급된다. 이 인수를 True로 설정하면 이 동작을 해제하여 HTML 코드가 동작을 하는 것이다. 그런데, 공식문서에서는 하지만 사용하지 않는 것이 좋다고 권유하며, 안전한 HTML을 작성하기는 어렵기 때문에 이 인수를 사용하면 사용자의 보안이 손상될 수 있다.[70] 만약, 커스터마이즈된 CSS를 사용하고 싶다면, style.css를 불러와서 st.markdown에 적용하는 방식을 취하도록 한다.[71]

[70] https://github.com/streamlit/streamlit/issues/152

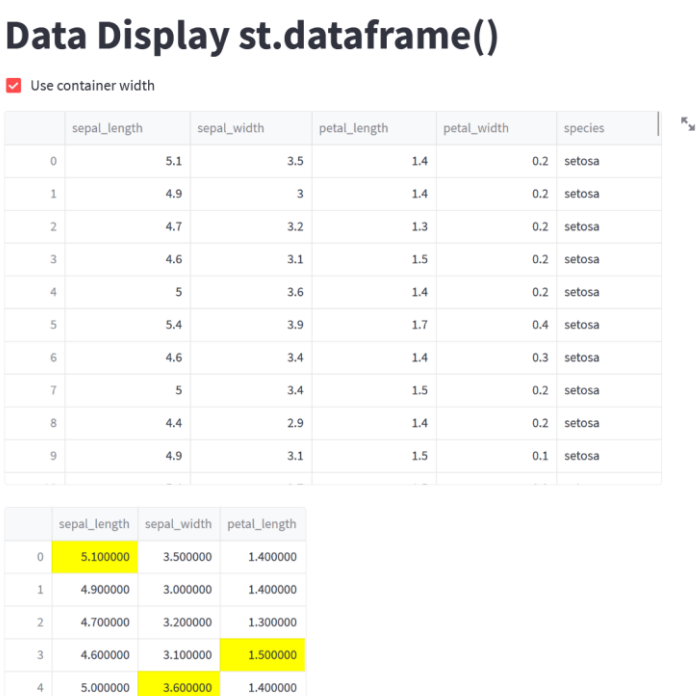
[71] https://medium.com/pythoneers/how-to-style-streamlit-metrics-in-custom-css-9a0f02b150da

### B. Data Display Elements

대시보드에서 중요한 건 메시지 전달이다. 메시지를 전달할 때는 꼭 반드시 그래프나 차트가 필요한 것은 아니다. 데이터를 테이블이나 또는 Metric으로 직접 표현하는 것도 효과적인 전달이 될 수 있다.

- st.dataframe()

두개의 테이블을 만든다. 한 개의 테이블은 use container width 체크박스 클릭 여부에 따라 컬럼의 너비가 확장되거나 축소되는 테이블이다. 다른 테이블은 pandas style 옵션을 적용하여 출력하는 형태다.




| | sepal_length | sepal_width | petal_length | petal_width | species |
|---|---|---|---|---|---|
| 0 | 5.1 | 3.5 | 1.4 | 0.2 | setosa |
| 1 | 4.9 | 3 | 1.4 | 0.2 | setosa |
| 2 | 4.7 | 3.2 | 1.3 | 0.2 | setosa |
| 3 | 4.6 | 3.1 | 1.5 | 0.2 | setosa |
| 4 | 5 | 3.6 | 1.4 | 0.2 | setosa |
| 5 | 5.4 | 3.9 | 1.7 | 0.4 | setosa |
| 6 | 4.6 | 3.4 | 1.4 | 0.3 | setosa |
| 7 | 5 | 3.4 | 1.5 | 0.2 | setosa |
| 8 | 4.4 | 2.9 | 1.4 | 0.2 | setosa |
| 9 | 4.9 | 3.1 | 1.5 | 0.1 | setosa |

| | sepal_length | sepal_width | petal_length |
|---|---|---|---|
| 0 | 5.100000 | 3.500000 | 1.400000 |
| 1 | 4.900000 | 3.000000 | 1.400000 |
| 2 | 4.700000 | 3.200000 | 1.300000 |
| 3 | 4.600000 | 3.100000 | 1.500000 |
| 4 | 5.000000 | 3.600000 | 1.400000 |

코드는 다음과 같다.

파일위치 : ch05/03_data_display_dataframe.py

```
# -*- coding:utf-8 -*-
import streamlit as st
import seaborn as sns
import pandas as pd

@st.cache_data
def load_data():
    df = sns.load_dataset('iris')
    return df

def main():
    st.title("Data Display st.dataframe()")
    st.checkbox("Use container width", value=False, key =
'use_container_width')

    iris = load_data()
    st.dataframe(iris,
use_container_width=st.session_state.use_container_width)

    # pandas style
    st.dataframe(iris.iloc[:5, 0:3].style.highlight_max(axis=0))

if __name__ == "__main__":
    main()
```

load_data() 함수는 @st.cache_data Decorator를 사용하여 정의되며, 이 데코레이터는 앱을 새로 고칠 때마다 이 함수의 출력을 캐시하여 다시 실행할 필요가 없도록 Streamlit에 지시한다. 이 함수는 Seaborn에서 iris 데이터 세트를 로드하고 이를 판다스 데이터 프레임으로 반환한다. 그 다음 사용자가 데이터 프레임에서 컨테이너 너비를 사용할지 여부를 토글할 수 있도록 st.checkbox()를 사용하여 확인란을 생성한다. 이 체크박스의 상태는 key 매개변수를 사용하여 st.session_state 객체에 저장된다. load_data() 함수를 호출하여 iris 데이터

세트를 데이터 프레임에 로드한 다음, st.dataframe()을 사용하여 데이터 프레임을 표시합니다. 사용_컨테이너_폭 매개변수는 사용자가 데이터프레임의 너비를 조정할 수 있도록 st.session_state의 체크박스 상태로 설정된다.

두번째 데이터프레임은 예제로 깔끔하게 보여주기 위해 .iloc 연산자를 사용하였다. 처음 5개의 행과 3개의 열만 표시하였다. style.highlight_max() 메서드는 각 열(axis=0)의 최대값을 강조 표시하는 데 사용한다.[72]

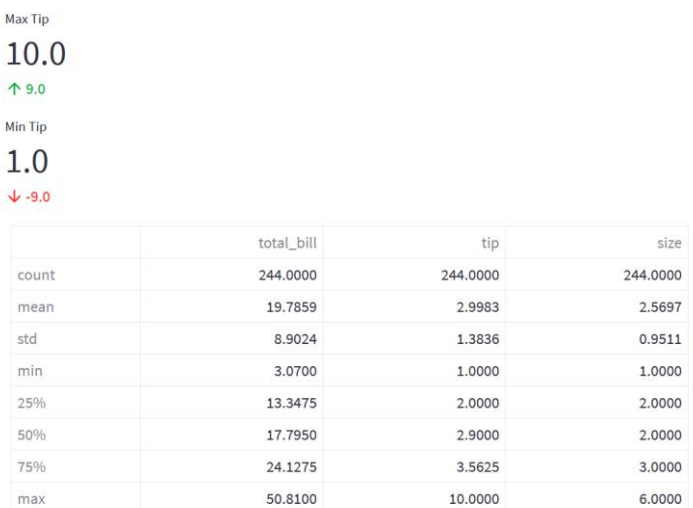
공식문서에 따르면 data 매개변수에는 입력할 수 있는 데이터 포맷은 pandas.DataFrame만 올 수 있는 것은 아니다. 클래스 기준으로 말하면, pandas.Styler, pyarrow.Table, numpy.ndarray, pyspark.sql.DataFrame, snowflake.snowpark.dataframe.DataFrame, snowflake.snowpark.table.Table, Iterable, dict 형태 등을 입력할 수 있다.

[72] 매개변수 axis=1를 적용하면 각 행의 최댓값이 노란색으로 바뀌게 된다. Table Visualization에 관한 추가적인 설명은 pandas의 공식문서를 확인한다. (https://pandas.pydata.org/docs/user_guide/style.html)

- st.table() & st.metric()

st.table() 및 st.metric()은 각각 데이터를 표 또는 metric 형식으로 표시하는 데 사용되는 Streamlit 라이브러리의 두 가지 함수이다.

st.table()은 데이터를 표 형식으로 표시하는 데 사용된다. 이 메서드는 테이블은 전체 내용이 페이지에 직접 배치되는 정적이라는 점에서 st.dataframe()과 다르다. st.metric()은 메트릭이나 핵심 성과 지표(KPI)와 같은 단일 값을 시각적으로 보기 좋은 형식으로 표시하는 데 사용된다. 이 함수는 값과 제목 및 단위와 같은 몇 가지 선택적 매개변수를 받아 카드로 표시합니다. 이 카드는 색상, 아이콘 등 다양한 매개변수를 사용하여 사용자 지정할 수 있다.




| | total_bill | tip | size |
|---|---|---|---|
| count | 244.0000 | 244.0000 | 244.0000 |
| mean | 19.7859 | 2.9983 | 2.5697 |
| std | 8.9024 | 1.3836 | 0.9511 |
| min | 3.0700 | 1.0000 | 1.0000 |
| 25% | 13.3475 | 2.0000 | 2.0000 |
| 50% | 17.7950 | 2.9000 | 2.0000 |
| 75% | 24.1275 | 3.5625 | 3.0000 |
| max | 50.8100 | 10.0000 | 6.0000 |

tips 테이블을 활용해서 tip 최댓값과 최솟값을 보여주고, 두 값의 차이를 보여주는 대시보드를 작성하였다. 또한, st.table()에는 tip의 기술 통계량을 테이블 형태로 보여주는 코드를 작성한다.

파일위치 : ch05/04_data_display_table_metric.py

```
# -*- coding:utf-8 -*-
import streamlit as st
import seaborn as sns
import pandas as pd

def main():
    tips = sns.load_dataset('tips')
    tip_max = tips['tip'].max()
    tip_min = tips['tip'].min()
    st.metric(label="Max Tip", value=tip_max, delta=tip_max - tip_min)
    st.metric(label="Min Tip", value=tip_min, delta=tip_min - tip_max)
    st.table(tips.describe())

if __name__ == "__main__":
    main()
```

Tip의 최댓값과 최솟값을 각각 구한 후 tip_max와 tip_min으로 개별적으로 변수에 할당한다. 최댓값과 최솟값을 각각 st.metric() 메서드에 저장하여 metric 카드를 생성한다. 각 매개변수에 대한 설명을 하면다음과 같다.

- label(str) : 문자열만 올 수 있다. 레이블은 선택적으로 마크다운을 포함할 수 있으며 다음 요소를 지원: 굵게, 기울임꼴, 취소선, 인라인 코드, 이모티콘 및 링크이다. 이 외에도 st.markdown()에서 지원하는 이모티콘, LaTeX, 컬러 텍스트 등을 지원한다.
- value(int, float, str, or None) : 표현하고자 하는 값이 입력된다.
- delta(int, float, str, or None) : 지표$_{metric}$가 어떻게 변경되었는지를 나타내는 지표로, 화살표로 렌더링된다. 델타가 음수(int/float)이거나 마이너스 기호(str)로 시작하는 경우 화살표는 아래쪽을 가리키고 텍스트는 빨간색이며, 그렇지 않으면 화살표는 위쪽을 가리키고 텍스트는 녹색이다. 만약 값이 없음(기본값)인 경우 델타는 표시되지 않는다.

st.table()는 pandas의 describe 메서드를 사용하여 tip 데이터 세트를 요약한 테이블을 표시하는 데 사용된다. 그 외에 더 많은 옵션은 도움말을 참조한다.[73]

[73] https://docs.streamlit.io/library/api-reference/data/st.metric

## C. Chart Elements

이번에는 Streamlit 에서 지원하는 Chart 에 대해 알아본다. 일반적으로 Streamlit 은 자체적으로 메서드를 지원하지만, Python 에서 제공하는 다양한 시각화 라이브러리들과 연계하는 것을 목표로 하고 있다. Chapter 3 장에서 배웠던 시각화 라이브러리들은 모두 지원이 가능하다. 본장에서는 주로 코드의 흐름만 이해하는 정도로만 기술할 것이고, 시각화 코드에 구체적인 설명은 여기애서는 생략하도록 한다. 그 외 자세한 설명은 Chapter 3 장의 Matplotlib, Seaborn, 그리고 Plotly 에서 확인하기를 바라며, Streamlit Documents 에서 추가적인 옵션들을 확인하기를 바란다.

아래 표는 Streamlit 라이브러리에서 제공하는 기본 시각화 메서드들이다.

| Streamlit Chart | Documents 주소 |
|---|---|
| st.line_chart() | https://docs.streamlit.io/library/api-reference/charts/st.line_chart |
| st.area_chart() | https://docs.streamlit.io/library/api-reference/charts/st.area_chart |
| st.bar_chart() | https://docs.streamlit.io/library/api-reference/charts/st.bar_chart |
| st.map() | https://docs.streamlit.io/library/api-reference/charts/st.map |

이번에는 Streamlit 에서 사용하는 외부 라이브러리에 대해 간단하게 요약하면 다음과 같다.

| Library | Streamlit Method |
|---|---|
| Matplotlib & Seaborn | st.pyplot()[74] |
| Altair | st.altair_chart()[75] |
| Vega-Lite | st.vega_lite_chart()[76] |

[74] https://docs.streamlit.io/library/api-reference/charts/st.pyplot

[75] https://docs.streamlit.io/library/api-reference/charts/st.altair_chart

[76] https://docs.streamlit.io/library/api-reference/charts/st.vega_lite_chart, vega_lite_chart는 Python에서 제공하는 라이브러리는 아니다. 기본적으로 JSON 형태로 구성되어야 하며, streamlit에서 해당 라이브러리를 API 형태로 사용할 수 있다.

| | |
|---|---|
| Plotly | st.plotly_chart()[77] |
| Bokeh | st.bokeh_chart()[78] |
| PyDeck | st.pydeck_chart()[79] |
| Grapviz | st.graphviz()[80] |

각 사용자가 즐겨 쓰는 시각화 라이브러리가 위 목록 중에 있다면, streamlit에서 쉽게 구현할 수 있다.

- 주요 시각화 라이브러리 설명

본 장에서 다루지 않은 시각화 라이브러리에 대한 개별적인 설명은 다음과 같다.

- Altair: 간결하고 직관적인 구문으로 대화형 시각화를 만들 수 있는 Python의 선언적 시각화 라이브러리입니다. 홈페이지: https://altair-viz.github.io/
- Vega-Lite: 웹용 대화형 시각화를 생성할 수 있는 Vega 위에 구축된 고급 시각화 문법입니다. 홈페이지: https://vega.github.io/vega-lite/
- Bokeh: 웹용 인터랙티브 브라우저 기반 시각화를 생성할 수 있는 Python 라이브러리입니다. 홈페이지: https://bokeh.org/
- PyDeck: WebGL 및 Deck.gl을 사용하여 대화형 고성능 시각화를 만들기 위한 Python 라이브러리입니다. 홈페이지: https://pydeck.gl/index.html
- Graphviz: DOT 언어를 사용하여 다이어그램 및 시각화를 생성할 수 있는 Python 라이브러리입니다. 홈페이지: https://graphviz.org/

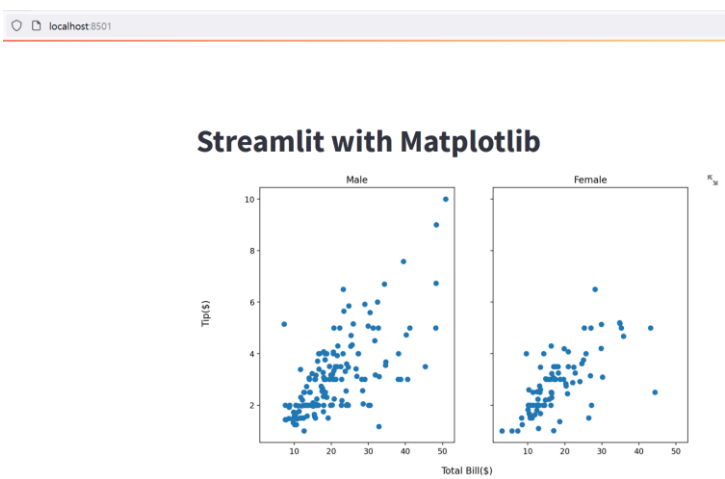
본 장에서는 Chapter 3장에서 배웠던 Matplotlib, Seaborn, Plotly를 활용하여 각각 대시보드로 표현하도록 한다.

---

[77] https://docs.streamlit.io/library/api-reference/charts/st.plotly_chart
[78] https://docs.streamlit.io/library/api-reference/charts/st.bokeh_chart
[79] https://docs.streamlit.io/library/api-reference/charts/st.pydeck_chart
[80] https://docs.streamlit.io/library/api-reference/charts/st.graphviz_chart

- Matplotlib

먼저 matplotlib 라이브러리를 활용하여 Streamlit 라이브러리에 표현한다. tips 데이터를 불러와서 Male과 Female별로 x축은 total_bill, y축은 tip으로 하여 산점도를 표현한 것이다.




위 코드를 시각화로 구현하면 다음과 같다.

파일위치 : ch05/05_matplotlib.py

```
# -*- coding:utf-8 -*-
import streamlit as st
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt

def main():
    st.title("Streamlit with Matplotlib")
```

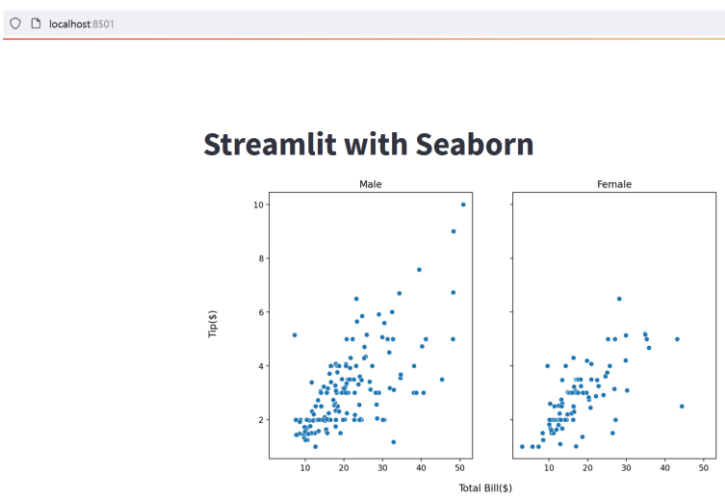
```
    tips = sns.load_dataset('tips')
    m_tips = tips.loc[tips['sex'] == 'Male']
    f_tips = tips.loc[tips['sex'] == 'Female']
    fig, ax = plt.subplots(ncols=2, figsize=(10, 6), sharex=True,
sharey=True)
    ax[0].scatter(x = m_tips['total_bill'], y = m_tips['tip'])
    ax[0].set_title('Male')
    ax[1].scatter(x = f_tips['total_bill'], y = f_tips['tip'])
    ax[1].set_title('Female')
    fig.supxlabel('Total Bill($)')
    fig.supylabel('Tip($)')
    st.pyplot(fig)

if __name__ == "__main__":
    main()
```

기존에 설명했던 코드는 생략하고, 마지막 3줄만 설명하면 다음과 같다. 전체적인 Figure의 전체적인 title과 X축과 Y축 라벨을 적용할 때는 FigureBase 클래스의 하위 메서드를 이용한다. 좀 더 자세한 설명은 Documents를 참조한다.[81] 본 장에서의 핵심은 st.pyplot으로 전달하는 객체로 ax가 아닌 fig를 선택해야 하는 것을 기억한다.

[81] https://matplotlib.org/stable/gallery/subplots_axes_and_figures/figure_title.html

- Seaborn

Chapter 3장에서 시각화 파트를 잘 읽었다면, Matplotlib와 Seaborn은 사실상 하나의 라이브러리로 봐도 무방하다는 것을 알 수 있을 것이다. 동일하게 코드로 구현한다.




위 시각화 코드는 다음과 같다.

파일위치 : ch05/06_seaborn.py

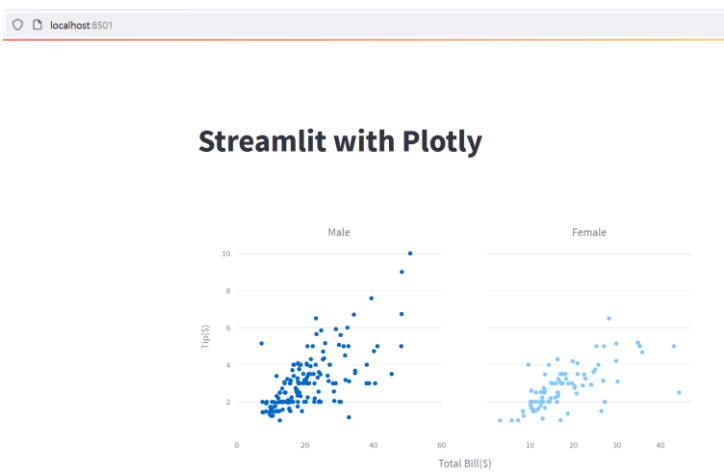
```
# -*- coding:utf-8 -*-
import streamlit as st
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt

def main():
    st.title("Streamlit with Seaborn")
    tips = sns.load_dataset('tips')
    m_tips = tips.loc[tips['sex'] == 'Male']
```

```
    f_tips = tips.loc[tips['sex'] == 'Female']
    fig, ax = plt.subplots(ncols=2, figsize=(10, 6), sharex=True,
sharey=True)
    sns.scatterplot(data=m_tips, x = 'total_bill', y = 'tip', ax=ax[0])
    ax[0].set_title('Male')
    sns.scatterplot(data=f_tips, x = 'total_bill', y = 'tip', ax=ax[1])
    ax[0].set(xlabel=None, ylabel=None)
    ax[1].set_title('Female')
    ax[1].set(xlabel=None, ylabel=None)
    fig.supxlabel('Total Bill($)')
    fig.supylabel('Tip($)')
    st.pyplot(fig)

if __name__ == "__main__":
    main()
```

`ax[0/1].set(xlabel=None, ylabel=None)`은 하위 플롯의 x축 및 y축 레이블 속성을 설정하는 것이다. 특히, xlabel=None 및 ylabel=None은 x축 레이블 및 y축 레이블을 비워두도록 설정하는 것인데, 간단하게 말하면 x축 또는 y축에 레이블이 표시되지 않도록 설정하는 것을 의미힌다. .set() 메서드는 Matplotlib에서 객체의 여러 속성을 한 번에 설정할 수 있는 방법이다.

- Plotly

이번에는 plotly 라이브러리를 활용하여 streamlit 대시보드에 표현하도록 한다.




코드는 아래와 같이 작성하였다.

파일위치 : ch05/07_plotly.py

```
# -*- coding:utf-8 -*-
import streamlit as st
import plotly.graph_objs as go
from plotly.subplots import make_subplots
import seaborn as sns

def main():
    st.title("Streamlit with Plotly")
    tips = sns.load_dataset('tips')
    m_tips = tips.loc[tips['sex'] == 'Male']
    f_tips = tips.loc[tips['sex'] == 'Female']
```

```
    fig = make_subplots(rows = 1,
                        cols = 2,
                        subplot_titles=('Male', 'Female'),
                        shared_yaxes=True,
                        shared_xaxes=True,
                        x_title='Total Bill($)'
                        )
    fig.add_trace(go.Scatter(x = m_tips['total_bill'], y = m_tips['tip'],
mode='markers'), row=1, col=1)
    fig.add_trace(go.Scatter(x = f_tips['total_bill'], y = f_tips['tip'],
mode='markers'), row=1, col=2)
    fig.update_yaxes(title_text="Tip($)", row=1, col=1)
    fig.update_xaxes(range=[0, 60])
    fig.update_layout(showlegend=False)

    # Display visualization
    st.plotly_chart(fig, use_container_width=True)

if __name__ == "__main__":
    main()
```

기본적으로 plotly는 크게 두개의 클래스를 활용하여 그래프를 작성한다(graph_objects, express). 그런데, streamlit에서 입력받는 그래프 객체는 plotly.graph_objects.Figure 또는 plotly.graph_objects.Data로 입력받는다.[82] 좀 더 간단하게 말하면, 기존 plotly.express 객체는 streamlit에서 활용하지 못하기 때문에, 이를 graph_objects나 data로 변형을 해야하는데, 이렇게 하면, 오히려 번거롭기 때문에 처음 시각화를 할 때부터 graph_objects 방식으로 작성하는 것을 권한다. 코드 설명은 make_subplots()부터 확인한다.

1. make_subplots 함수에 다음 인수를 사용하여 새 하위 플롯을 만든다.
   - rows=1: 하위 플롯의 한 행

[82] https://docs.streamlit.io/library/api-reference/charts/st.plotly_chart

- cols=2: 하위 플롯의 두 열
- subplot_titles: 각 서브플롯에 대한 제목을 튜플 형태로 저장
- shared_yaxes=True: 두 하위 플롯에서 y축 공유
- shared_xaxes=True: 두 하위 플롯에서 x축 공유
- x_title: 두 하위 플롯의 x축에 대한 제목

2. 하위 플롯 개체의 add_trace 함수를 사용하여 하위 플롯에 두 개의 트레이스(산점도 차트)를 추가한다:
- go.Scatter는 x 및 y 값으로 산점도 차트를 만드는 데 사용된다.
- mode='markers'는 산점도 플롯에 마커만 표시하도록 지정하는 데 사용된다.
- row=1, col=1은 첫 번째 트레이스가 첫 번째 하위 플롯(즉, 왼쪽에 있는 플롯)에 추가되도록 지정한다.
- row=1, col=2는 두 번째 트레이스가 두 번째 하위 플롯(즉, 오른쪽에 있는 플롯)에 추가되도록 지정한다.

3. 다음 인수와 함께 update_yaxes 함수를 사용하여 첫 번째 하위 플롯의 y축 제목을 업데이트합니다
- title_text="Tip($)": 첫 번째 하위 플롯의 y축 제목(이 경우 "Tip($)")

4. 다음 인수와 함께 update_xaxes 함수를 사용하여 두 하위 플롯의 x축 범위를 업데이트 한다.
- range=[0, 60]: x축의 범위를 0에서 60으로 설정한다.
- row=1, col=1: 첫 번째 하위 플롯에 적용하도록 지정한다.

5. update_layout 함수에 기존 차트의 레이아웃을 업데이트 한다
- showlegend=False: 그래프에서 범례를 숨긴다.

6. 다음 인수와 함께 st.plotly_chart 함수를 사용하여 차트를 표시한다.
- fig: 표시할 차트를 의미한다.
- use_container_width=True: 그래프 너비를 컨테이너 너비로 설정한다.

## D. Input Widgets

Streamlit은 사용자가 시각화 및 입력 데이터와 상호 작용할 수 있는 다양한 위젯을 제공한다. 일반적으로 사용되는 위젯은 다음과 같다.

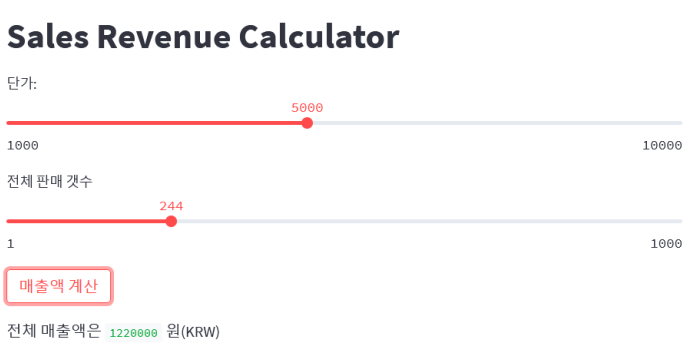
- 텍스트 입력: 사용자가 텍스트 값을 입력할 수 있다. Streamlit 앱에 텍스트 입력 위젯을 추가하려면 st.text_input() 함수를 사용한다.
- 숫자 입력: 사용자가 숫자 값을 입력할 수 있다. Streamlit 앱에 숫자 입력 위젯을 추가하려면 st.number_input() 함수를 사용한다.
- 슬라이더 입력: 사용자가 트랙을 따라 핸들을 밀어서 값을 선택할 수 있다. Streamlit 앱에 슬라이더 입력 위젯을 추가하려면 st.slider() 함수를 사용한다.
- 드롭다운 입력: 사용자가 드롭다운 목록에서 값을 선택할 수 있다. Streamlit 앱에 드롭다운 입력 위젯을 추가하려면 st.selectbox() 또는 st.multiselect() 함수를 사용한다.
- 체크박스 입력: 사용자가 옵션 목록에서 하나 이상의 옵션을 선택할 수 있다. Streamlit 앱에 체크박스 입력 위젯을 추가하려면 st.checkbox() 또는 st.multicheckbox() 함수를 사용한다.
- 라디오 버튼 입력: 사용자가 옵션 목록에서 하나의 옵션을 선택할 수 있다. Streamlit 앱에 라디오 버튼 입력 위젯을 추가 시, st.radio() 함수를 사용한다.
- 파일 업로더: 사용자가 컴퓨터에서 파일을 업로드할 수 있다. Streamlit 앱에 파일 업로더 위젯을 추가 시, st.file_uploader() 함수를 사용한다.
- 날짜 입력: 사용자가 달력에서 날짜를 선택할 수 있다. Streamlit 앱에 날짜 입력 위젯을 추가하려면 st.date_input() 함수를 사용한다.
- 시간 입력: 사용자가 시간을 선택할 수 있다. Streamlit 앱에 시간 입력 위젯을 추가하려면 st.time_input() 함수를 사용한다.

색상 선택기: 사용자가 색상을 선택할 수 있다. Streamlit 앱에 색상 선택기 위젯을 추가하려면 st.color_picker() 함수를 사용한다.

요구 사항에 따라 이러한 위젯의 레이블, 기본값 및 기타 속성을 사용자 지정할 수 있다. 이러한 위젯은 인터랙티브하고 사용자 친화적인 데이터 시각화 및 입력 양식을 만드는 데 매우 유용하다. Streamlit 앱에 입력 위젯을 추가하려면 해당 함수를 호출하고 레이블, 기본값, 범위 등 필요한 매개변수를 전달한다. 본격적으로 각 위젯을 테스트 하도록 한다.

- Widget-1 : st.slider & st.button

이번에는 매출액을 계산하는 대시보드를 생성하도록 한다. 여기에서 사용한 위젯은 st.slider(), st.button()이 사용되었다.




파일위치 : ch05/08_button.py

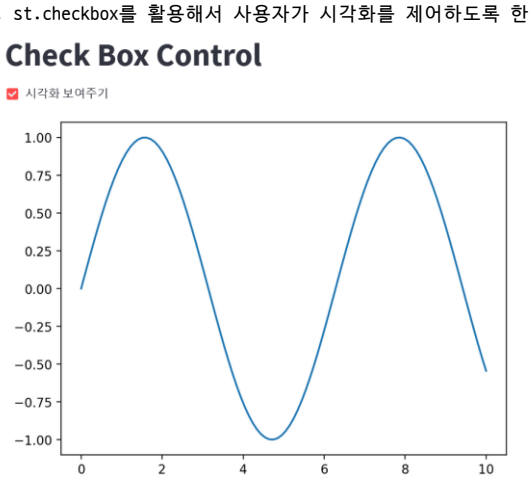
```
# -*- coding:utf-8 -*-
import streamlit as st

def calculate_sales_revenue(price, total_sales):
    revenue = price * total_sales
    return revenue
def main():
    st.title("Sales Revenue Calculator")
    price = st.slider("단가:", 1000, 10000, value=5000)
    total_sales = st.slider("전체 판매 갯수", 1, 1000, value=500)

    if st.button("매출액 계산"):
        revenue = calculate_sales_revenue(price, total_sales)
        st.write("전체 매출액은", revenue, "원(KRW)")

if __name__ == "__main__":
    main()
```

코드 설명은 다음과 같다.

- 먼저, calculate_sales_revenue 함수를 정의 한다. 이 함수는 price와 total_sales라는 두 개의 매개변수를 받는다. 이 함수는 두 인수를 곱하고 결과를 반환한다.
- st.slider()을 활용하여 가격에 대한 슬라이더 위젯을 만든다. 슬라이더의 최소값은 1000, 최대값은 10000, 기본값은 5000으로 정했다. .
- st.slider()을 활용하여 판매된 총 품목 수에 대한 슬라이더 위젯을 생성한다. 슬라이더의 최소값은 1, 최대값은 1000, 기본값은 500으로 정했다.
- st.button()함수를 활용하여 '매출액 계산'(수익 계산) 버튼이 클릭되었는지 확인한다. 버튼을 클릭한 경우 가격 및 총 판매갯수 슬라이더 위젯의 값을 사용하여 calculate_sales_revenue 함수가 호출됩니다. 그러면 함수 결과가 Streamlit 앱에 표시된다.

- Widget-2 : st.checkbox

st.checkbox 위젯은 사용자가 하나 이상의 옵션을 선택할 수 있도록 하는 데 사용된다. 서비스 약관에 동의하는지, 이메일 업데이트를 수신할지 등 사용자의 의견을 받는 데 유용하다. 또한 사용자가 체크박스에 체크 표시를 한 경우에만 특정 콘텐츠를 표시하는 등 앱의 흐름을 제어하는 데에도 st.checkbox 위젯을 사용할 수 있다. st.checkbox를 활용해서 사용자가 시각화를 제어하도록 한다.



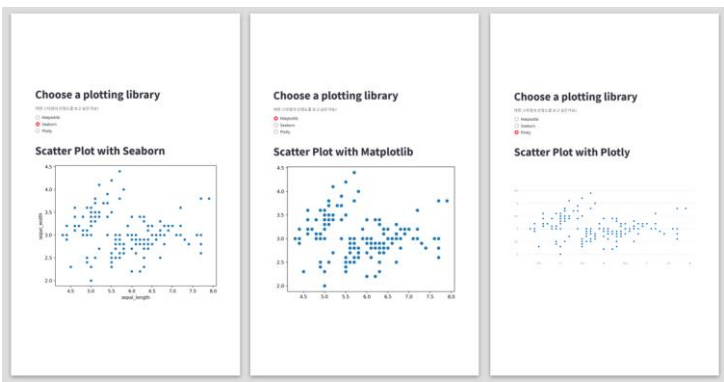
파일위치 : ch05/09_checkbox.py

```
# -*- coding:utf-8 -*-
import streamlit as st
import matplotlib.pyplot as plt
import numpy as np

def main():
    st.title('Check Box Control")
    x = np.linspace(0, 10, 100)
    y = np.sin(x)

    show_plot = st.checkbox("시각화 보여주기")

    fig, ax = plt.subplots()
    ax.plot(x, y)

    if show_plot:
        st.pyplot(fig)

if __name__ == '__main__':
    main()
```

코드 설명은 다음과 같다. 시각화를 작성한 후, if문을 사용하여 show_plot의 값을 확인한다. True인 경우(즉, checkbox가 선택된 경우), st.pyplot()을 사용하여 시각화가 Streamlit에 표시됩니다. 만약 checkbox를 클릭하지 않는다면, 시각화는 사라지게 된다.

- Widget-3 : st.radio

st.checkbox 위젯은 사용자가 여러 옵션 그룹에서 하나의 옵션을 선택할 수 있도록 하는 데 사용된다. 앞서 살펴본, st.checkbox 위젯과 비슷하지만, 사용자는 한 번에 하나의 옵션만 선택할 수 있다. 기본적으로 st.checkbox는 두 개의 매개변수를 입력받는다. 첫번째 매개변수는 옵션 그룹에 대한 레이블이며, 두번째 매개변수는 옵션 목록이다. st.radio 위젯은 선택한 옵션의 인덱스를 반환한다. 아래 그림은 matplotlib, seaborn, plotly가 선택되면, 선택된 항목이 시각화가 되도록 설계하였다.




코드는 다음과 같다.

파일위치 : ch05/10_radio.py

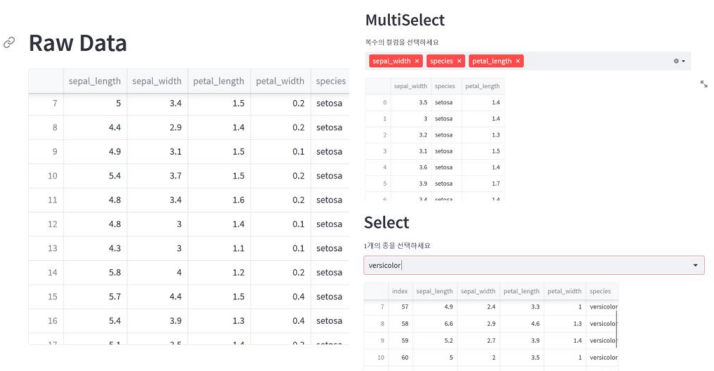
```
# -*- coding:utf-8 -*-
import streamlit as st
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.graph_objects as go

# 데이터 불러오기
iris = sns.load_dataset('iris')

def plot_matplotlib():
    st.title('Scatter Plot with Matplotlib')
```

```
    fig, ax = plt.subplots()
    ax.scatter(iris['sepal_length'], iris['sepal_width'])
    st.pyplot(fig)

def plot_seaborn():
    st.title('Scatter Plot with Seaborn')
    fig, ax = plt.subplots()
    sns.scatterplot(iris, x = 'sepal_length', y = 'sepal_width')
    st.pyplot(fig)

def plot_plotly():
    st.title('Scatter Plot with Plotly')
    fig = go.Figure()
    fig.add_trace(
        go.Scatter(x = iris['sepal_length'],
                   y = iris['sepal_width'],
                   mode='markers')
    )
    st.plotly_chart(fig)

def main():
    st.title("Choose a plotting library")
    plot_type = st.radio(
        "어떤 스타일의 산점도를 보고 싶은가요?",
        ("Matplotlib", "Seaborn", "Plotly"))

    if plot_type == "Matplotlib":
        plot_matplotlib()
    elif plot_type == "Seaborn":
        plot_seaborn()
    elif plot_type == "Plotly":
        plot_plotly()

if __name__ == '__main__':
    main()
```

코드가 복잡해 보이지만, 이 코드는 서로 다른 Python 시각화 라이브러리를 사용하여 산점도 그래프를 생성하는 세 가지 함수, plot_matplotlib(), plot_seaborn(), plot_plotly()를 정의한다. 각 함수의 시각화 라이브러리는 각각 Matplotlib, Seaborn, Plotly이다. 그런 다음 정의된 함수를 호출하기 위해 Streamlit에서 제공하는 라디오 버튼 위젯을 통해 사용자가 이 세 가지 플로팅 라이브러리 중 어떤 것을 사용할지 선택할 수 있도록 하였다. 사용자가 그래프 유형을 선택하고 라디오 버튼을 클릭하면 해당 함수가 실행되고 그래프 유형에 따라 Streamlit의 st.pyplot() 또는 st.plotly_chart() 함수를 사용하여 그래프를 Streamlit 페이지에 표시한다.
요약하면, 구조화해서 살펴보면 크게 3개의 서로 다른 시각화 함수를 정의한 것이고, 각 radio 버튼에 대응할 수 있도록 조건문을 통해 구현한 것에 불과하다.

- Widget-4 : st.selectbox & st.multiselect

st.selectbox를 사용하면 사용자가 옵션 드롭다운 목록에서 단일 옵션을 선택할 수 있는 반면, st.multiselect를 사용하면 사용자가 체크박스를 사용하여 옵션 목록에서 여러 옵션을 선택할 수 있다. 예를 들어 사용자가 카테고리 목록에서 단일 카테고리를 선택할 수 있도록 하려는 경우 st.selectbox가 유용하고, 사용자가 한 번에 여러 카테고리를 선택할 수 있도록 하려는 경우 st.multiselect가 유용할 수 있다. 두 위젯 모두 보다 대화형 방식으로 데이터를 필터링하고 탐색하는 데 유용할 수 있으며, 특정 사용 사례와 사용자 요구 사항에 따라 위젯을 선택할 수 있다. 먼저 iris 데이터를 불러와서 행과 열이 선택되도록 하였다.




파일위치 : ch05/11_select_multiselect.py

```
# -*- coding:utf-8 -*-
import streamlit as st
import pandas as pd
import seaborn as sns

# 데이터 불러오기
iris = sns.load_dataset('iris')

def main():

    st.markdown("## Raw Data")
```

```
    st.dataframe(iris)

    st.markdown("<hr>", unsafe_allow_html=True)
    st.markdown("## Select")
    species = st.selectbox("1 개의 종을 선택하세요", iris.species.unique())
    st.dataframe(iris[iris['species']==species].reset_index())

    st.markdown("<hr>", unsafe_allow_html=True)
    st.markdown("## MultiSelect")
    cols = st.multiselect("복수의 컬럼을 선택하세요", iris.columns)
    filtered_iris = iris.loc[:, cols]
    st.dataframe(filtered_iris)

if __name__ == "__main__":
    main()
```

여기에서는 pandas 라이브러리 문법이 자주 사용되었다. iris 데이터를 불러와서 가공되지 않은 iris 데이터를 보여준다. 중간에 <hr> 태그는 HTML문서에서 내용을 구분하거나 주제의 변화를 정의할 때 쓰는 HTML 태그이다. 데이터 집합의 species 컬럼에 있는 고유 값은 iris.species.unique()를 사용하여 가져오고, 이 값을 옵션으로 사용하여 st.selectbox()를 사용하여 선택 상자를 만든다. st.selectbox()를 활용하여 종 선택 시, 특정 종에 해당되는 행, 즉, 데이터만 추출되도록 하였다.
그 후, 데이터의 컬럼 값만 추출하여 st.multiselect() 옵션으로 사용한다. 사용자가 multiselect 상자에서 그림과 같이 선택하면 복수의 컬럼값이 cols로 저장된다. 그 후에 iris.loc 문법을 확용하여 특정 컬럼만 조회가 되도록 하였다.

- Widget-5 : st.slider & st.select_slider

st.slider와 st.select_slider는 사용자가 범위 또는 옵션 목록에서 값을 선택할 수 있는 Streamlit 위젯이다. 본 위젯을 사용해서 머신러닝 기법 중 하이퍼파라미터 튜닝을 적용하고 하이퍼파라미터 값에 따라 예측값이 움직이도록 하였다. Tips 데이터셋을 활용하여 모델을 만든 후, 각 파라미터를 지정하면, 예측 결괏값이 변하도록 대시보드를 구성할 수 있다.


파일위치 : ch05/12_slider_select_slider.py

```
import streamlit as st
import pandas as pd
import seaborn as sns
import numpy as np
```

**메모 포함[evan7]:** 여기 코드 보고 살짝 정리 해줘요.. 넘 길면, 변수명을 바꾸던가, 아니면 매개변수를 하나씩 나열하던가 해야할 듯요.

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.metrics import r2_score, mean_absolute_error
import plotly.graph_objects as go
@st.cache_data
def load_data():
    # 데이터 불러오기
    tips = sns.load_dataset('tips')

    return tips

@st.cache_resource
def run_model(data, max_depth, min_samples_leaf):
    # 특성과 타겟 분리
    y = data['tip']
    X = data[['total_bill', 'size']]

    # 훈련, 테스트 데이터 분리
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
    st.write('선택된 max_depth:', max_depth, '& min_samples_leat:',
min_samples_leaf)

    random_search = {'max_depth': [i for i in range(max_depth[0],
max_depth[1])],
                     'min_samples_leaf': [min_samples_leaf]}

    clf = RandomForestRegressor()
    model = RandomizedSearchCV(estimator = clf, param_distributions =
random_search, n_iter = 10,
                                    cv = 4, verbose= 1, random_state= 101,
n_jobs = -1)
    return model.fit(X_train,y_train), X_test, y_test

def prediction(model, X_test, y_test):
    # 예측
```

```
    y_test_pred = model.predict(X_test)

    # 성능 평가
    test_mae = mean_absolute_error(y_test, y_test_pred)
    r2 = r2_score(y_test, y_test_pred)

    return y_test_pred, test_mae, r2

def prediction_plot(X_test, y_test, y_test_pred, test_mae, r2):
    # 그래프 그리기
    fig = go.Figure()
    fig.add_trace(
        go.Scatter(x=X_test['total_bill'], y=y_test, mode='markers',
name='test', marker=dict(color='red'))
    )
    fig.add_trace(
        go.Scatter(x=X_test['total_bill'], y=y_test_pred, mode='markers',
name='prediction', marker=dict(color='green'))
    )

    fig.update_layout(
        title='Tip Prediction with RandomForestRegressor',
        xaxis_title='Total Bill',
        yaxis_title='Total',
        annotations=[go.layout.Annotation(x=40, y=1.5,
                                          text=f'Test MAE:
{test_mae:.3f}<br>R2 Score: {r2:.3f}',
                                          showarrow=False)]
    )

    st.plotly_chart(fig)

def main():
    # Hyperparameters
    max_depth = st.select_slider("Select max depth", options=[i for i in
range(2, 30)], value=(5, 10))
```

```
    min_samples_leaf = st.slider("Minimum samples leaf", min_value=2,
max_value=20)

    tips = load_data()
    model, X_test, y_test = run_model(tips, max_depth, min_samples_leaf)
    y_test_pred, test_mae, r2 = prediction(model, X_test, y_test)
    prediction_plot(X_test, y_test, y_test_pred, test_mae, r2)

if __name__ == "__main__":
    main()
```

코드의 세부적인 내용은 생략하고, 전반적인 코드의 흐름만 기술한다.
이 코드는 Streamlit 프레임워크를 사용하여 RandomForestRegressor 모델을 구현한 것이다. 이 모델의 독립변수는 total_bill, size이고, 종속변수는 tip을 설정한 후, tip을 예측하도록 학습된 것이다. 사용자는 슬라이더와 선택 슬라이더를 사용하여 모델의 하이퍼파라미터를 선택할 수 있다.
load_data 함수는 Seaborn에서 tips 데이터 세트를 로드하여 반환한다. 이 함수는 데이터를 캐시하기 위해 @st.cache_data로 설정하였고, 사용자가 앱과 상호작용할 때마다 다시 로드되지 않는다.
run_model 함수는 로드된 데이터 세트, Max Depth, min_samples_leaf를 입력 파라미터로 받는다. 그런 다음 데이터를 훈련 및 테스트 세트로 분할하고, 하이퍼파라미터 사전을 정의하고, RandomForestRegressor 모델을 생성하고, RandomizedSearchCV를 수행하여 최적의 하이퍼파라미터를 찾습니다. 이 함수는 학습된 모델, 테스트 세트 및 테스트 세트에 대한 예측 값을 반환한다. 이 함수는 모델을 캐시하기 위해 @st.cache_resource로 장식되어 있으므로 사용자가 앱과 상호 작용할 때마다 모델을 재학습하지 않습니다. 그러나, 파라미터가 변동이 되면 재학습을 시작하는 것이다.
예측 함수는 학습된 모델, 테스트 세트 및 테스트 세트에 대한 예측 값을 입력 파라미터로 사용합니다. 평균 절대 오차(MAE) 및 R2 점수를 계산하여 테스트 세트에 대한 예측 값과 함께 이 값을 반환한다.
prediction_plot 함수는 테스트 세트, 테스트 세트의 예측 값, MAE 및 R2 점수를 입력 파라미터로 사용한다. 이 함수는 total_bill를 X축에, tip을 Y축에, 실젯값은 빨간색 점으로, 예측값은 녹색 점으로 표시하여 Plotly를 사용하여 산점도 차트를 만듭니다. 또한 플롯에 MAE 및 R2 점수도 주석으로 표시했다.