

2. pandas

pandas¹²는 데이터 조작 및 분석 도구를 제공하는 Python 라이브러리이다. 일반적으로 CSV 또는 Excel 파일과 같은 구조화된 데이터를 전처리(Processing)하고 다른 데이터(Hanlding)에 사용된다. Pandas를 사용하면 사용자가 간단하고 효율적인 방식으로 처리하는 불러오고, 정제하고, 필터링하고, 변환하고 시각화를 할 수 있다. 또한 데이터 단순 집계, 그룹화를 활용한 집계, 피벗 테이블에 대한 매우 유용한 기능을 제공한다. 데이터 분석가 뿐만 아니라, 데이터 사이언티스트, 데이터 엔지니어 등을 포함한 데이터를 다루는 모든 사람에게는 필수적인 도구이다.

일반적으로 pandas는 크게 두개의 객체를 가지고 있다. 먼저 Series 객체는 정수, 실수, 문자열 등 다양한 데이터 유형을 저장할 수 있는 1차원 배열과 유사한 객체이며, Series의 각 요소에 레이블을 지정하는 인덱스가 존재한다. Series를 생성하는 방법은 아래와 같다.

```
import pandas as pd

d = {'a': 1, 'b': 2, 'c': 3}
ser = pd.Series(data=d, index=['a', 'b', 'c'])
ser

[결과]
a    1
b    2
c    3
dtype: int64
```

데이터프레임 DataFrame 객체는 여러 개의 Series 또는 서로 다른 유형의 배열을 저장할 수 있는 2차원 배열과 유사한 객체로 정의할 수 있다. 각 데이터프레임의 각 요소에 레이블을 지정하는 행, 열 인덱스가 모두 존재한다. 다음은 데이터프레임의 일반적인 예시이다.

¹² <https://pandas.pydata.org/>

```
import pandas as pd

d = {'col1': [1, 2], 'col2': [3, 4]}
df = pd.DataFrame(data=d)
print(df)
```

```
[결과]
   col1  col2
0     1     3
1     2     4
```

데이터 전처리를 하면, 반환값이 Series로 나올 때도 있고, DataFrame으로 나올 때도 있기 때문에, 중간에 한번씩 확인하는 것이 좋다.

A. 데이터 불러오기

데이터를 불러오는 방법은 `pd.read_csv`(각 파일의 경로) 함수를 활용하면 사용할 수 있다. 본 프로젝트와 유사한 부동산 데이터를 불러온다.¹³

```
import pandas as pd

df_boston = pd.read_csv("./data/boston.csv")
df_boston
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	target
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2
...
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.67	22.4
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	9.08	20.6
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	5.64	23.9
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	6.48	22.0
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	7.88	11.9

506 rows × 14 columns

B. 컬럼 선택

특정한 컬럼을 선택할 때는 `List`에 특정 컬럼명을 작성하여 아래와 같이 선택할 수 있다. 독자들도 임의의 컬럼을 작성하여 추출하도록 한다.

```
cols = ['CRIM', 'ZN', 'INDUS']
result = df_boston[cols].head()
print(result)
```

¹³ Boston 데이터셋의 정보는 <http://lib.stat.cmu.edu/datasets/boston>에서 확인한다. 본 장에서는 독자가 해당 컬럼들의 정보를 이해했다는 가정에서 기술하였다.

[결과]

	CRIM	ZN	INDUS
0	0.00632	18.0	2.31
1	0.02731	0.0	7.07
2	0.02729	0.0	7.07
3	0.03237	0.0	2.18
4	0.06905	0.0	2.18

C. 행 선택

행을 선택할 때는 조건식을 통해서 처리하는 경우가 일반적이다. 예를 들어 ZN에서 18.0에 해당하는 행만 선택하는 코드는 아래와 같이 할 수 있다. 조건식을 작성할 때는 비교연산자를 통해서 작성이 가능하다.

```
result = df_boston[df_boston['ZN'] == 18.0]
cols = ['CRIM', 'ZN', 'INDUS']
print(result[cols])
```

[결과]

	CRIM	ZN	INDUS
0	0.00632	18.0	2.31

D. .loc 와 iloc

pandas에서 보편적으로 행과 열을 선택하는 방법은 .loc와 .iloc를 사용하는 것이다. 두 메서드의 가장 큰 차이는 loc는 행과 열을 선택할 때, labels/names로 접근하는 방식이고, iloc는 행과 열을 선택할 때, index/position으로 접근하는 방식이다. 만약, loc를 사용해서 행과 열을 선택할 때 매칭되는 label/name이 없다면 KeyError가 발생한다. 반면, iloc를 사용해서 행과 열을 선택할 때 매칭되는 index가 없다면 IndexError가 발생한다.

.loc를 활용하여 존재하지 않는 "CRI"를 입력하면 KeyError가 발생한다.

```
df_boston.loc['CRI']
~/anaconda3/lib/site-packages/pandas/core/indexes/range.py in get_loc(self, key, method, tolerance)
    387         raise KeyError(key) from err
    388     self._check_indexing_error(key)
--> 389     raise KeyError(key)
    390     return super().get_loc(key, method=method, tolerance=tolerance)
    391
KeyError: 'CRI'
```

.iloc를 활용하여 존재하지 않는 컬럼 인덱스 20을 다음과 같이 입력하면 IndexError가 발생한다.

```
df_boston.iloc[:, 20]
~/anaconda3/lib/site-packages/pandas/core/indexing.py in _validate_integer(self, key, axis)
    1453     len_axis = len(self.obj._get_axis(axis))
    1454     if key >= len_axis or key < -len_axis:
--> 1455         raise IndexError("single positional indexer is out-of-bounds")
    1456
    1457     # -----
```

- loc

간단한 예제를 통해 loc가 어떻게 적용되는지 확인한다. 아래 코드는 CRIM, ZN, target 컬럼만 추출하면서, CRIM은 1보다 작은 행을 조회하도록 한다.

```
cols = ['CRIM', 'ZN', 'target']
result = df_boston.loc[df_boston['CRIM'] < 1, cols]
print(result.head(3))

[결과]
CRIM      ZN  target
```

0	0.00632	18.0	24.0
1	0.02731	0.0	21.6
2	0.02729	0.0	34.7

이번에는 다중 조건을 입력하도록 한다. 위 조건에 추가적으로 target 이 24 보다 이상인 데이터만 조회하도록 한다. 아래 코드에서 &은 and 연산자를 의미하고, |은 or 연산자를 의미한다. 독자분들은 & 연산자 대신에 | 연산자를 대입하면 다른 결과가 나오는 것을 확인할 수 있다.

```
cols = ['CRIM', 'ZN', 'target']
result = df_boston.loc[(df_boston['CRIM'] < 1) &
                      (df_boston['target'] >= 24.0), cols]
print(result.head(3))
```

[결과]

	CRIM	ZN	target
0	0.00632	18.0	24.0
2	0.02729	0.0	34.7
3	0.03237	0.0	33.4

iloc

이번에는 iloc를 활용하도록 한다. 일반적으로 iloc를 활용할 때 행은 조건식이 아닌 인덱싱 및 슬라이싱을 활용하여 추출하는 것에 유의한다. 마찬가지로 컬럼도 인덱싱 및 슬라이싱을 활용하여 추출하는 것이다. 예를 들면, 0~4번째 인덱스 행과, 처음 3개의 컬럼만 조회한다고 하면 아래와 같이 작성할 수 있다.

```
result = df_boston.iloc[0:5, 0:3]
print(result)
```

[결과]

	CRIM	ZN	INDUS
0	0.00632	18.0	2.31
1	0.02731	0.0	7.07
2	0.02729	0.0	7.07

```
3 0.03237  0.0   2.18
4 0.06905  0.0   2.18
```

만약, 임의의 한 개의 행만 조회한다면, 데이터프레임에서 Series 형태로 바뀐다는 것에 주의한다.

```
result = df_boston.iloc[20, 0:3]
print(result)
print(type(result))
```

[결과]

```
CRIM      1.25179
ZN        0.00000
INDUS    8.14000
Name: 20, dtype: float64
<class 'pandas.core.series.Series'>
```

E. describe()

주어진 데이터의 기초 통계량을 구할 때 쓰는 함수이며, 평균, 표준편차, 각 컬럼의 사분위를 그릴 수 있다.

```
cols = ['CRIM', 'ZN', 'target']
print(df_boston[cols].describe())

[결과]
          CRIM           ZN       target
count  506.000000  506.000000  506.000000
mean    3.613524  11.363636  22.532806
std     8.601545  23.322453   9.197104
min    0.006320  0.000000   5.000000
25%    0.082045  0.000000  17.025000
50%    0.256510  0.000000  21.200000
75%    3.677083  12.500000  25.000000
max   88.976200 100.000000  50.000000
```

F. rename()

컬럼명을 변경할 때 사용한다. 이 때, 딕셔너리 형태로 추가한다. 우선 기존 데이터셋에서 일부 데이터만 추출한다.

```
sample_df = df_boston.iloc[:, 0:4]
sample_df.head()
```

	CRIM	ZN	INDUS	CHAS
0	0.00632	18.0	2.31	0.0
1	0.02731	0.0	7.07	0.0
2	0.02729	0.0	7.07	0.0
3	0.03237	0.0	2.18	0.0
4	0.06905	0.0	2.18	0.0

컬럼명을 바꾸는 방법은 아래와 같이 할 수 있다.

```
data.rename(columns={'Old 컬럼명' : 'New 컬럼명'})
```

추출된 sample_df 데이터에서 ZN 컬럼명을 landZone 으로 변경하는 코드를 작성하도록 한다.

```
sample_df = sample_df.rename(columns={'ZN': 'landZone'})
sample_df.head()
```

	CRIM	landZone	INDUS	CHAS
0	0.00632	18.0	2.31	0.0
1	0.02731	0.0	7.07	0.0
2	0.02729	0.0	7.07	0.0
3	0.03237	0.0	2.18	0.0
4	0.06905	0.0	2.18	0.0

만약, 이번에는 두개의 컬럼을 변경한다면 딕셔너리 안에 연속해서 key-value 형태로 값을 추가하면 된다. 기존 컬럼 CRIM 과 INDUS 을 각각 crime 과 businessRatio 로 변경하도록 한다.

```
sample_df = sample_df.rename(columns={'CRIM': 'crime',
                                         'INDUS':'businessRatio'})
sample_df.head()
```

	crime	landZone	businessRatio	CHAS
0	0.00632	18.0	2.31	0.0
1	0.02731	0.0	7.07	0.0
2	0.02729	0.0	7.07	0.0
3	0.03237	0.0	2.18	0.0
4	0.06905	0.0	2.18	0.0

G. value_counts()

각 컬럼의 여러 값(value)에 대한 모든 숫자를 세어보는 메서드로 매우 기초적인 집계함수를 제공한다. 데이터타입이, object, int, 등 상관없이 사용이 가능하다. 데이터 df_boston 에서 RAD 에 해당 함수를 적용해본다. 이 때 결과값은 DataFrame에서 Series 형태로 반환되는 것에 주의한다.

```
df_boston['RAD'].value_counts()
```

```
24.0    132
5.0     115
4.0     110
3.0      38
6.0      26
2.0      24
8.0      24
1.0      20
7.0      17
Name: RAD, dtype: int64
```

```
type(df_boston['RAD'].value_counts())
```

```
[결과]
pandas.core.series.Series
```

이 메서드 매개변수에는 normalize라는 매개변수가 존재한다. 이 매개변수를 활성화하면 빈도수가 아닌 비율로 표시된다.

```
df_boston['RAD'].value_counts(normalize=True)
```

```
24.0    0.260870
5.0     0.227273
4.0     0.217391
3.0     0.075099
6.0     0.051383
2.0     0.047431
8.0     0.047431
1.0     0.039526
7.0     0.033597
Name: RAD, dtype: float64
```

H. isin()

특정 컬럼의 다양한 값 중에서 일부의 값만 가져오는 코드를 작성할 때 유용하게 사용할 수 있다. 메서드 안에는 일반적으로 리스트로 정의한 값이 적용된다.

아래 코드를 통해 확인해본다. 가상의 데이터를 만든 후, 특정 '구'만 출력이 되도록 코드를 작성한다.

```
import pandas as pd

df = pd.DataFrame({'시군구': ['관악구', '서대문구', '강남구', '서초구'],
                   '부동산가격': [10000, 20000, 30000, 40000]})

cities = ['관악구', '서대문구']
filtered_df = df[df['시군구'].isin(cities)]
```

filtered_df

	시군구	부동산가격
0	관악구	10000
1	서대문구	20000

이번에는 숫자에 적용을 하도록 해본다. 리스트 numbers에 정의된 값만 출력이 되는 것을 확인할 수 있다.

```
numbers = [1.0, 7.0]
filtered_df = df_boston[df_boston['RAD'].isin(numbers)]
filtered_df[['CRIM', 'RAD']].head(6)
```

	CRIM	RAD
0	0.00632	1.0
193	0.02187	1.0
194	0.01439	1.0
244	0.20608	7.0
245	0.19133	7.0
246	0.33983	7.0

I. 시계열 Time Series 데이터 핸들링

pandas 라이브러리에서 시계열 데이터는 인덱스가 날짜 혹은 시간인 데이터를 말한다. 만약 가상으로 시계열 자료를 생성하려면 인덱스를 숫자가 아닌 DatetimeIndex 자료형¹⁴으로 만들어야 한다. 시계열 데이터를 만들 때 일반적으로 기본적으로 기억해야 하는 주요 함수는 pd.to_datetime, pd.date_range, pd.Timestamp 등이 있지만 여기에서는 pd.to_datetime만 집중해서 본다.

- pd.to_datetime()

일반적으로 기존 문자열, 날짜/시간, 리스트, Tuple 등 입력된 날짜를 pandas datetime 객체로 변환한다. 간단한 예를 들면 아래와 같이 작업할 수 있다. 먼저 한 개의 문제를 변경할 때는 아래와 같이 진행한다.

```
date_string = '2023-03-02'  
datetime_obj = pd.to_datetime(date_string)  
print(datetime_obj)  
print(type(datetime_obj))
```

[결과]
2023-03-02 00:00:00
<class 'pandas._libs.tslibs.timestamps.Timestamp'>

만약 리스트로 정의를 한다면 아래와 같이 처리가 된다.

```
date_list = ['2023-03-02']  
datetime_obj_list = pd.to_datetime(date_list)  
print(datetime_obj_list)  
print(type(datetime_obj_list))
```

[결과]
DatetimeIndex(['2023-03-02'], dtype='datetime64[ns]', freq=None)
<class 'pandas.core.indexes.datetimes.DatetimeIndex'>

이번에는 가상의 데이터를 불러오고 object를 datetime으로 변환하는 방법을 확

¹⁴ <https://pandas.pydata.org/docs/reference/api/pandas.DatetimeIndex.html>

인한다. (데이터셋 참조 : House Property Sales Time Series¹⁵⁾)

```
sales = pd.read_csv('data/raw_sales.csv')
sales.head(3)
```

	datesold	postcode	price	propertyType	bedrooms
0	2007-02-07 00:00:00	2607	525000	house	4
1	2007-02-27 00:00:00	2906	290000	house	3
2	2007-03-07 00:00:00	2905	328000	house	3

데이터의 정보를 확인하면 datesold는 datetime이 아니라 object인 것을 확인할 수 있다.

```
sales.info()
```

[결과]

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 29580 entries, 0 to 29579
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype  
 --- 
 0   datesold    29580 non-null   object 
 1   postcode     29580 non-null   int64  
 2   price        29580 non-null   int64  
 3   propertyType 29580 non-null   object 
 4   bedrooms     29580 non-null   int64  
dtypes: int64(3), object(2)
memory usage: 1.1+ MB
```

이제 위 데이터를 pd.to_datetime()을 활용하여 object를 datetime으로 변환한다.

```
sales['datesold'] = pd.to_datetime(sales['datesold'])
sales.info()
```

[결과]

¹⁵ <https://www.kaggle.com/datasets/htaghoddings/property-sales?resource=download>

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 29580 entries, 0 to 29579
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   datesold    29580 non-null   datetime64[ns]
 1   postcode     29580 non-null   int64  
 2   price        29580 non-null   int64  
 3   propertyType 29580 non-null   object  
 4   bedrooms     29580 non-null   int64  
dtypes: int64(3), object(2)
memory usage: 1.1+ MB
```

- 시계열의 전체 기간 길이 구하기

이번에는 위 데이터의 전체 기간의 길이를 구하도록 한다.

```
sales["datesold"].max() - sales["datesold"].min()

[결과]
Timedelta('4553 days 00:00:00')
```

- 연도, 월, 일 구하기

이번에는 주어진 날짜 데이터에서 연도, 월, 일을 구하도록 한다.

```
sales["year"] = sales["datesold"].dt.year
sales["month"] = sales["datesold"].dt.month
sales["day"] = sales["datesold"].dt.day

sales.head()
```

	datesold	postcode	price	propertyType	bedrooms	year	month	day
0	2007-02-07	2607	525000	house	4	2007	2	7
1	2007-02-27	2906	290000	house	3	2007	2	27
2	2007-03-07	2905	328000	house	3	2007	3	7
3	2007-03-09	2905	380000	house	4	2007	3	9
4	2007-03-21	2906	310000	house	3	2007	3	21

- `shift()`

`shift` 연산을 사용하면 인덱스는 그대로 두고 데이터만 이동이 가능하다. 이 때 시계열은 앞으로 또는 뒤로 지정된 기간만큼 이동한다. 다음 샘플 코드를 통해 확인해본다.

```
temp_df = sales[['datesold', 'price']].copy()
temp_df['shifted_v1'] = temp_df['price'].shift(1, fill_value=0).astype(int)
temp_df['shifted_v2'] = temp_df['price'].shift(2, fill_value=0).astype(int)

temp_df.head()
```

	datesold	price	shifted_v1	shifted_v2
0	2007-02-07	525000	0	0
1	2007-02-27	290000	525000	0
2	2007-03-07	328000	290000	525000
3	2007-03-09	380000	328000	290000
4	2007-03-21	310000	380000	328000

이번에는 `shift(-1)`, `shift(-2)`를 대입하도록 한다. 이 때 결과는 반대로 데이터의 마지막 부분을 확인해야 한다.

```
temp_df['shifted_v3'] = temp_df['price'].shift(-1, fill_value=0).astype(int)
temp_df['shifted_v4'] = temp_df['price'].shift(-2, fill_value=0).astype(int)
temp_df.tail()
```

	datesold	price	shifted_v1	shifted_v2	shifted_v3	shifted_v4
29575	2019-07-25	500000	475000	380000	560000	464950
29576	2019-07-25	560000	500000	475000	464950	589000
29577	2019-07-26	464950	560000	500000	589000	775000
29578	2019-07-26	589000	464950	560000	775000	0
29579	2019-07-26	775000	589000	464950	0	0

- `strftime()`

주어진 `datesold` 컬럼에 한글 형태로 바꿔서 저장을 하도록 하는데, 이때 `strftime()` 메서드를 이용하여 문자열을 만들 수 있다.

```
temp_df['한글날짜'] = temp_df['datesold'].dt.strftime("%Y년 %m월 %d일")
temp_df.head()
```

	datesold	price	shifted_v1	shifted_v2	shifted_v3	shifted_v4	한글날짜
0	2007-02-07	525000	0	0	290000	328000	2007년 02월 07일
1	2007-02-27	290000	525000	0	328000	380000	2007년 02월 27일
2	2007-03-07	328000	290000	525000	380000	310000	2007년 03월 07일
3	2007-03-09	380000	328000	290000	310000	465000	2007년 03월 09일
4	2007-03-21	310000	380000	328000	465000	399000	2007년 03월 21일

J. GroupBy 연산

GroupBy 연산은 하나의 데이터프레임 또는 Series에 있는 특정 컬럼안에 있는 여러개의 값(value)을 열로 그룹화한 다음 집계 함수를 수행할 수 있는 방법을 말한다. 그룹별 통계량을 확인하고자 할 때 매우 유용하게 활용할 수 있다. 아래 이미지¹⁶를 통해서 확인하면 크게 Split-Apply-Combine 3단계로 구성됨을 알 수 있다.

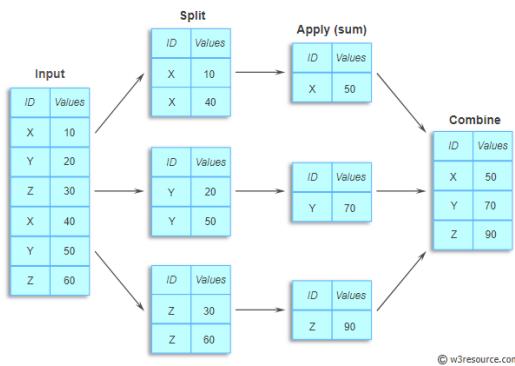


Figure 2.

Split 단계에서는 `groupby()` 컬럼에서 그룹화할 수 있는 영역을 찾고, 분리를 한다. Apply 단계에서는 독립된 그룹별 함수를 적용한다. 위 그림에서는 `sum()` 함수가 사용이 되었지만, 그 외에도 다른 집계 함수 사용이 가능하다. Combine 단계에서는 각각의 독립된 그룹별로 함수가 적용된 결과를 종합하여 다시 하나의 테이블을 합치게 된다. 아래 코드를 통해서 살펴보도록 한다.

```
agg_dict = {"price" : "mean", "bedrooms" : "mean"}  
grouped = sales.groupby("propertyType").agg(agg_dict)  
grouped
```

¹⁶ <https://www.w3resource.com/python-exercises/pandas/groupby/index.php>

	price	bedrooms
propertyType		
house	647956.128462	3.539467
unit	423106.557279	1.837510

위 코드는 sales 데이터에서 "propertyType" 열의 값에 따라 그룹화하고, 각 그룹에서 "price"와 "bedrooms" 열의 평균 값을 구하는 코드이다. 여기에서 확인해야 하는 코드는 agg_dict 변수인데, 'price' 열에 대해서는 평균값을 계산하고, 'bedroom' 열에 대해서도 평균값을 처리하도록 구조화하였고, 이를 grouped에 변수로 저장하여 출력하는 것이다. 출력된 형태는 DataFrame이며 인덱스는 propertyType으로 지정된 것을 확인할 수 있다. 만약, 인덱스로 정의된 컬럼을 일반 컬럼으로 변환하고자 한다면 reset_index()를 실행한다.

grouped.reset_index()

propertyType	price	bedrooms
0 house	647956.128462	3.539467
1 unit	423106.557279	1.837510

이번에는 컬럼별 다른 집계함수를 적용하도록 한다. 예를 들면, price 에는 전체 합계와 중간값, bedrooms 에는 평균값만 도출하도록 하는 것이다. 기존 정의된 딕셔너리에서 수정할 것은 value 의 값을 리스트로 재정의 하는 것이다.

```
agg_dict = {"price" : ["sum", "median"], "bedrooms" : "mean"}
grouped = sales.groupby("propertyType").agg(agg_dict)
grouped
```

propertyType	price bedrooms		
	sum	median	mean
house	15908618866	585000.0	3.539467
unit	2127379770	390000.0	1.837510

만약 위 컬럼을 평坦화를 하려면 결괏값으로 도출된 데이터프레임 columns에 대한 이해가 필요하다. 우선 아래 코드를 살펴보면 결괏값이 튜플형태로 출력되는 것을 확인할 수 있다.

```
grouped.columns  
  
[결과]  
MultiIndex([( 'price',      'sum'),  
             ('price', 'median'),  
             ('bedrooms',   'mean')],  
            )
```

위 개념을 확인해서 MultiIndex로 된 코드를 하나의 열로 치환하는 코드는 아래와 같다. 결과물을 확인하면 일반적인 데이터프레임 형태로 변경되었다. 추가 설명을 하면 먼저 빈 리스트를 만든 다음 반복문을 사용하는데, 이 때 핵심이 되는 코드는 각 열에 대해 isinstance(col, tuple)인지 확인하는 코드이다. 만약 튜플형태로 판정되면, new_col에 f-string 문자열의 원리를 이용하여 튜플의 값을 연결하여 저장하고, 빈리스트에 하나씩 추가하여 기존 컬럼에 대입하면 최종적으로 업데이트가 되도록 하는 코드이다.

```
new_columns = []  
for col in grouped.columns:  
    if isinstance(col, tuple):  
        new_col = f'{col[0]}_{col[1]}'  
    else:  
        new_col = col  
    new_columns.append(new_col)  
  
grouped.columns = new_columns  
grouped
```

	price_sum	price_median	bedrooms_mean
propertyType			
house	15908618866	585000.0	3.539467
unit	2127379770	390000.0	1.837510

3. Matplotlib

Streamlit 라이브러리에서는 `matplotlib.pyplot figure`가 나타나도록 지원하고 있다.¹⁷ 따라서, 기본적인 시각화를 익히도록 한다. 시각화의 차트 종류는 매우 다양하지만, 여기에서는 본 프로젝트에서 사용한 시각화 차트 위주로 간단하게 준비하였다.

메모 포함[J1]: 선, 막대, 박스플롯만 그렸는데, 더 그릴게 있는지 확인

A. 선 그래프(Line Plot)

부동산 데이터셋 중 시계열 데이터가 적용된 데이터셋을 불러온 후 간단하게 선 그래프를 그려본다. 데이터셋을 가져올 때, pandas와 다른 부분은 `read_csv()` 메서드 내 파라미터를 적용해서 `object`가 아닌 `datetime64`로 저장되고 있다는 차이점이 있다.

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

sales = pd.read_csv('data/raw_sales.csv', parse_dates=['datesold'])
sales.head(3)
```

	datesold	postcode	price	propertyType	bedrooms
0	2007-02-07	2607	525000	house	4
1	2007-02-27	2906	290000	house	3
2	2007-03-07	2905	328000	house	3

데이터셋의 기본 정보는 아래와 같다. 앞서 pandas와 다르게 여기에서는 `datetime64`로 적용된 것을 확인할 수 있다.

메모 포함[h2]: 완성시, 앞선 페이지 명시해주자!

```
print(sales.info())

[결과]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 29580 entries, 0 to 29579
```

¹⁷ <https://docs.streamlit.io/library/api-reference/charts/st.pyplot>

```
Data columns (total 5 columns):
 #  Column      Non-Null Count  Dtype  
 --- 
 0  datesold    29580 non-null   datetime64[ns]
 1  postcode     29580 non-null   int64  
 2  price        29580 non-null   int64  
 3  propertyType 29580 non-null   object 
 4  bedrooms     29580 non-null   int64  
 dtypes: datetime64[ns](1), int64(3), object(1)
 memory usage: 1.1+ MB
 None
```

이제 pandas 데이터프레임의 그룹 연산을 활용하여 연도별 평균 주택가격을 구하도록 한다.

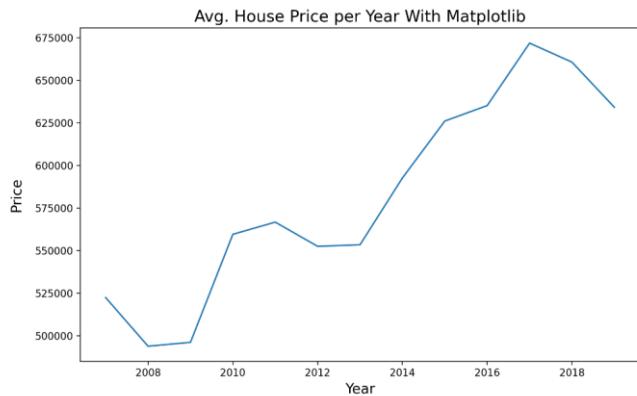
```
import numpy as np

sales['year'] = sales['datesold'].dt.year
result = np.round(sales.groupby('year')['price'].agg(np.mean), 1)
pd.DataFrame(result.head()).T
```

year	2007	2008	2009	2010	2011
price	522377.2	493814.2	496092.0	559564.8	566715.1

이번에는 선 그래프 시각화를 구현한다.

```
fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(result.index, result.values)
ax.set_title('Avg. House Price per Year With Matplotlib', size = 16)
ax.set_xlabel('Year', size = 14)
ax.set_ylabel('Price', size = 14, labelpad=12)
plt.savefig('output/matplotlib01.png', dpi=200)
plt.show()
```



위 코드에 대해서 구체적으로 설명하면 다음과 같다.¹⁸

- `fig, ax = plt.subplots(figsize=(10, 6))`가 의미하는 것은 Matplotlib에서 그래프를 작성할 때는 기본적으로 Figure 객체가 Axes객체가 필요하다.
- `ax.plot(result.index, result.values)`: ax 객체의 `plot()` 메서드를 사용하여 그래프를 그린다. 이 때, `result.index`는 x축 값, `result.values`는 y축 값으로 사용된다.
- `ax.set_title('Avg. House Price per Year With Matplotlib', size=16)`: ax 객체의 `set_title()` 메서드를 사용하여 그래프의 제목을 지정하고, `size`는 제목의 크기를 지정한다.
- `ax.set_xlabel('Year', size=14)`: ax 객체의 `set_xlabel()` 메서드를 사용하여 x축의 레이블을 지정한다.
- `ax.set_ylabel('Price', size=14, labelpad=12)`: ax 객체의 `set_ylabel()` 메서드를 사용하여 y축의 레이블을 지정합니다. `labelpad`는 레이블과 축 사이의 간격을 지정한다.
- `plt.savefig('output/matplotlib01.png', dpi=200)`: 그래프를 이미지 파일로 저장하는 명령어이다.

¹⁸ <https://delftswa.gitbooks.io/desosa-2017/content/matplotlib/chapter.html>

- 일로 저장한다. dpi는 저장된 이미지의 해상도를 지정한다
- plt.show(): 그래프를 보여준다.

B. 막대 그래프(Bar Plot)

막대 그래프 범주형 데이터를 직사각형 막대로 나타내는 그래프 유형으로, 각 막대의 길이나 높이는 해당 범주의 양 또는 빈도를 나타낸다. 막대 그래프는 서로 다른 범주의 값 비교 또는 범주의 값이 시간에 따라 어떻게 변화하는지 추적하는데 사용됩니다. 데이터 시각화에서 일반적으로 사용되며 데이터를 간단하고 쉽게 이해할 수 있는 방식으로 표시한다. 현재 주어진 데이터에서 막대 그래프를 그리기 위해 간단하게 데이터를 가공하도록 한다. 여기에서 2007년과 2008년 데이터만 추출하고, 각 월별 평균가격을 집계하도록 한다.

```
import numpy as np
import seaborn as sns

sales['year'] = sales['datesold'].dt.year
sales['month'] = sales['datesold'].dt.month

# 2007, 2008 추출
df = sales.copy()
df = df[df['year'].isin([2007, 2008])]
result = np.round(df.groupby(['year', 'month'])['price'].agg(np.mean), 1)
result.reset_index().iloc[:3, :]
```

	year	month	price
0	2007	2	407500.0
3	2007	5	339500.0
6	2007	8	505608.7
9	2007	11	505442.3
12	2008	2	531080.0
15	2008	5	522394.7
18	2008	8	449510.4
21	2008	11	489546.3

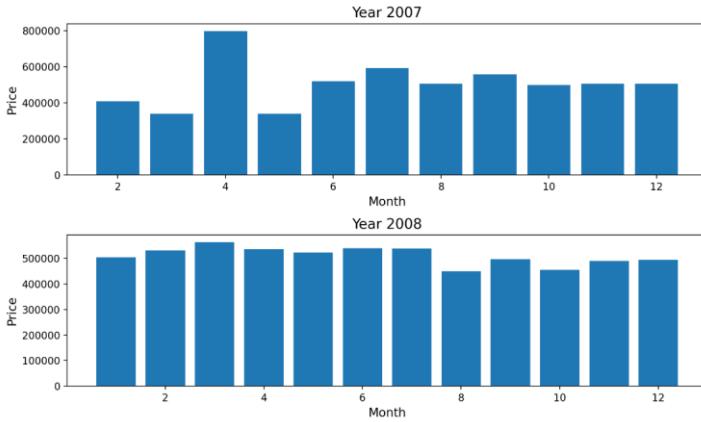
위 결괏값을 토대로 연도별 그래프를 작성한다. 우선, result에 reset_index()를 적용하여 새로운 데이터로 저장한다.

```
result = result.reset_index()
```

```
result['year'].unique()  
[결과]  
array([2007, 2008], dtype=int64)
```

이 정리된 데이터를 matplotlib를 적용하여 막대그래프를 작성하도록 한다.

```
fig, ax = plt.subplots(nrows=2, ncols=1, figsize=(10, 6))  
for i, y in enumerate(result['year'].unique()):  
    year_data = result[result['year'] == y]  
    ax[i].bar(year_data['month'], year_data['price'])  
    ax[i].set_title('Year {}'.format(y), fontsize=14)  
    ax[i].set_xlabel('Month', fontsize=12)  
    ax[i].set_ylabel('Price', fontsize=12)  
plt.tight_layout()  
plt.savefig('output/matplotlib02.png', dpi=200)  
plt.show()
```



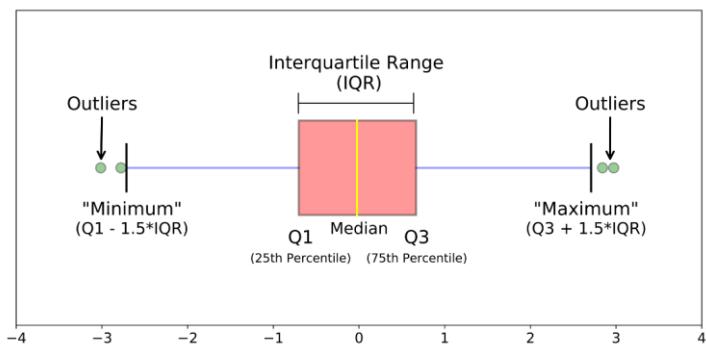
코드 설명을 하면 다음과 같다.

- `plt.subplots()` 함수를 사용하여 그림 객체와 2x1 grid의 subplot 축을 생성한다.

- for i, y in enumerate(result['year'].unique()): 코드는 result DataFrame의 year 열의 고유한 값에 대해 반복문을 실행한다. 이를 통해서 각 연도마다 하나의 subplot이 만들어진다.
- result[result['year'] == y] 구문을 사용하여 현재 연도에 해당하는 행만 선택하여 year_data 변수에 저장한다.
- ax[i].bar() 메서드를 사용하여 현재 연도의 각 월에 대한 price 값을 막대 그래프로 나타낸다.

C. 박스플롯 (Boxplot)

박스플롯은 데이터의 분포와 이상치 outliers를 시각화하는 그래프이다. 주로 수치형 변수의 분포를 살펴볼 때 사용된다. 박스플롯은 다섯 가지 요약 통계량(최소값, 1분위 값(25%), 중간값(50%), 3분위 값(75%), 최대값)을 이용하여 그려진다. 이 미지를 통해 확인하면 아래와 같다.¹⁹



이러한 요약 통계량은 상자(box) 안에 표시되며, 상자의 하단부터 차례로 1분위 값, 중간값, 3분위값까지 그린다. 상자 위와 아래에 있는 선은 각각 최소값과 최대값을 나타낸다. 이상치는 상자와 선의 범위를 벗어나 있는 값들로, 일반적으로 $1.5 \times \text{IQR}_{\text{Interquartile Range}}$ 이상 벗어나면 이상치로 판단한다. 박스플롯은 데이터의 분포와 이상치를 시각화하여 데이터의 중심 경향성과 분산정도를 파악하는 데 도움

¹⁹ https://itwiki.kr/w/박스_플롯#/media/파일:Boxplot2.png

을 준다. 또한 다른 변수와의 관계를 파악하는데도 유용하게 사용된다. 예를 들어, 그룹별 박스플롯을 그리면 그룹 간 분포의 차이나 중심 경향성의 차이를 한 눈에 파악할 수 있다. 박스플롯 그래프를 그리기 위해 price 가격에서 1,000,000 이하의 데이터만 추출하도록 한다.

```
sales = pd.read_csv('data/raw_sales.csv', parse_dates=['datesold'])
sales['year'] = sales['datesold'].dt.year
sales['month'] = sales['datesold'].dt.month
sales = sales.loc[sales['price'] < 1000000].reset_index(drop=True)
```

```
sales.head(3)
```

	datesold	postcode	price	propertyType	bedrooms	year	month
0	2007-02-07	2607	525000	house	4	2007	2
1	2007-02-27	2906	290000	house	3	2007	2
2	2007-03-07	2905	328000	house	3	2007	3

propertyType에 있는 값은 두개(house, unit)로 확인할 수 있다.

```
sales['propertyType'].unique()
```

[결과]

```
array(['house', 'unit'], dtype=object)
```

이제 박스플롯 그래프를 그려보면 아래와 같이 나타나게 된다.

```
from matplotlib.ticker import ScalarFormatter
formatter = ScalarFormatter()
formatter.set_scientific(False)

house_df = sales[sales['propertyType'] == 'house']['price']
unit_df = sales[sales['propertyType'] == 'unit']['price']

fig, ax = plt.subplots(figsize=(10, 6))
ax.boxplot([house_df, unit_df])
ax.set_xticklabels(['House', 'Unit'])
ax.set_xlabel('Property Type', fontsize=12)
ax.set_ylabel('Price', fontsize=12)
ax.set_title('Prices of Houses and Units', fontsize=14)
```

```
ax.yaxis.set_major_formatter(formatter)  
  
plt.savefig('output/matplotlib03.png', dpi=200)  
plt.show()
```



House와 Unit의 가격을 박스 플롯으로 시각화하는 코드이다. 코드에 대한 설명은 아래와 같이 작업할 수 있다.

- 'from matplotlib.ticker import ScalarFormatter'²⁰는 축의 눈금 레이블을 포맷하는데 사용된다. 특히 과학적 표기법 scientific notation으로 표시되는 숫자를 일반적인 숫자로 변환하는 데 사용한다. 이를 통해 그래프에서 축의 숫자가 더 읽기 쉽고 이해하기 쉬운 형식으로 표시할 수 있다. 여기서 설정한 변수는 추후 ax.yaxis.set_major_formatter()에 대입한다.
- sales 데이터프레임에서 'propertyType'이 'house'와 'unit'별로 price 데이터를 각각 가져온다.
- ax.boxplot([house_df, unit_df]) house_df와 unit_df 데이터프레임을 사용하여 박스 플롯을 그린다. 특히, ax.boxplot()안에 리스트로 각각의

²⁰

https://matplotlib.org/stable/api/ticker_api.html#matplotlib.ticker.ScalarFormatter

데이터를 넣어야 한다는 것에 주의한다.

- `ax.set_xticklabels(['House', 'Unit'])` x-축 레이블을 "House"와 "Unit"으로 설정한다.

4. Seaborn

Seaborn 라이브러리를 기본적으로 Matplotlib의 객체를 포함하고 있기 때문에, Streamlit 라이브러리에서 matplotlib 객체로 사용이 가능하다. 따라서, 이번에는 Seaborn 라이브러리로 그래프를 그려보도록 한다.

A. 선 그래프 (Line Chart)

먼저, 2008년과 2018년의 데이터만 추출하고, 연도, 월별의 평균 가격을 조회하는 코드를 작성하도록 한다. 그리고 이를 result로 저장한다.

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

sales = pd.read_csv('data/raw_sales.csv', parse_dates=['datesold'])
sales['year'] = sales['datesold'].dt.year
sales['month'] = sales['datesold'].dt.month

# 2007, 2008 추출
df = sales.copy()
df = df[df['year'].isin([2008, 2018])]
result = np.round(df.groupby(['year', 'month'])['price'].agg(np.mean),
1).reset_index()
result.head(12)
```

	year	month	price
0	2008	1	504428.6
1	2008	2	531080.0
2	2008	3	563500.0
3	2008	4	534204.5
4	2008	5	522394.7
5	2008	6	539092.6
6	2008	7	537453.1
7	2008	8	449510.4
8	2008	9	496778.7
9	2008	10	454822.0
10	2008	11	489546.3
11	2008	12	494410.8

이번에는 2018년 데이터만 조회한다.

```
result.tail(12)
```

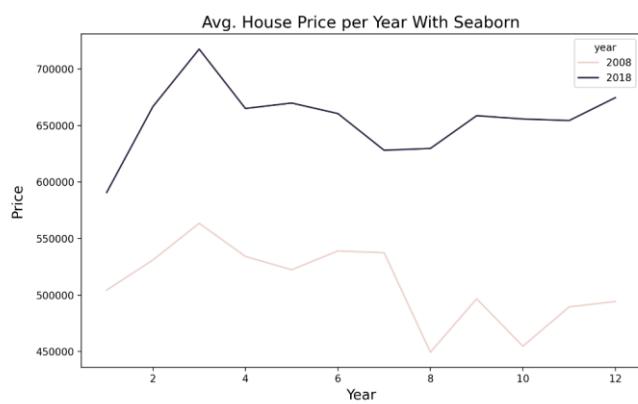
	year	month	price
12	2018	1	590753.8
13	2018	2	666738.3
14	2018	3	717516.1
15	2018	4	665040.4
16	2018	5	669840.6
17	2018	6	660528.7
18	2018	7	628074.0
19	2018	8	629684.1
20	2018	9	658647.4
21	2018	10	655744.7
22	2018	11	654334.5
23	2018	12	674591.1

전반적인 코드는 matplotlib와 크게 다른 것이 없다. 다만, seaborn의 편리성은 lineplot() 메서드안에 있는 hue라는 객체를 사용할 때 알 수 있다. 일반적으로 두개 연도별로 그래프를 작성하려면 ax.plot()를 두개 그려야하고, 각 연도별로 데이터도 별도로 추출해야 하지만, seaborn에서는 그러한 과정을 생략하는 것을 알 수 있다.

```
import matplotlib.pyplot as plt
from matplotlib.ticker import ScalarFormatter
formatter = ScalarFormatter()
formatter.set_scientific(False)

fig, ax = plt.subplots(figsize=(10, 6))
sns.lineplot(data=result, x=result.month, y=result.price, hue=result.year,
ax=ax)
ax.set_title('Avg. House Price per Year With Seaborn', size = 16)
ax.set_xlabel('Year', size = 14)
ax.set_ylabel('Price', size = 14, labelpad=12)
ax.yaxis.set_major_formatter(formatter)

plt.savefig('output/seaborn01.png', dpi=200)
plt.show()
```



B. 막대 그래프 (Bar Plot)

seaborn에는 다양한 종류의 barplot을 그릴 수 있다. 기본적으로 총 4개의 그래프가 존재하는데, 아래 예시를 통해 우선 확인해본다. seaborn 라이브러리에 존재하는 tips 데이터셋을 가져와서 간단하게 데이터를 확인한다.²¹

```
import seaborn as sns

# 데이터 가져오기
tips = sns.load_dataset("tips")
tips.head()
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

이번에는 seaborn내에 존재하는 다양한 막대 그래프를 그려보도록 한다.

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(nrows = 2, ncols = 2, figsize=(10, 6))

sns.barplot(x="day", y="total_bill", data=tips, ax = ax[0, 0])
ax[0, 0].set_title('barplot()')

sns.countplot(x="day", data=tips, ax = ax[0, 1])
ax[0, 1].set_title('countplot()')

sns.stripplot(x="day", y="total_bill", data=tips, ax = ax[1, 0])
ax[1, 0].set_title('stripplot()')

sns.pointplot(x="day", y="total_bill", data=tips, ax = ax[1, 1])
```

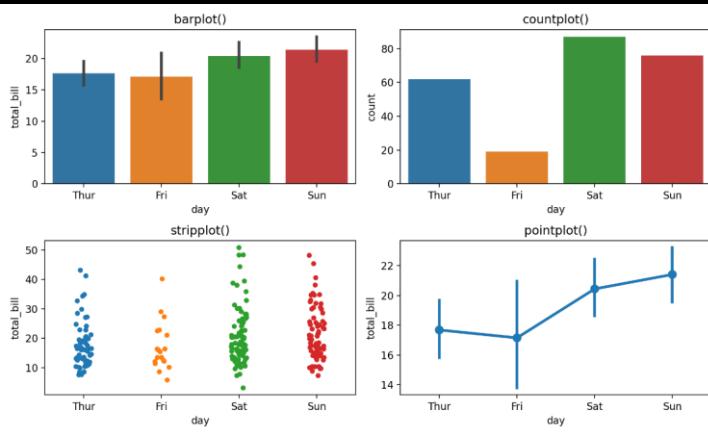
²¹ tips 데이터셋 설명: <https://rdrr.io/cran/reshape2/man/tips.html>

```

ax[1, 1].set_title('pointplot()')

plt.tight_layout()
plt.savefig('output/seaborn02.png', dpi=200)
plt.show()

```



위 그림에서 보는 것처럼 각 함수마다 결과는 다르게 나타나기 때문에, 각 사용자가 목적에 맞게 사용하면 된다. 위 4개의 그래프에 대한 개별적인 설명은 아래와 같이 정리할 수 있다.

- `sns.barplot()`²²: 이 함수는 범주형 변수의 각 범주(category)에 대한 수치형 변수의 평균을 수직 막대그래프(vertical bar plot)로 그리는 것이다. 이 때, 각 막대의 높이는 각 범주에서의 평균을 보여주는 것이며, 신뢰구간까지 보여준다. 즉, 이 함수를 통해서 각 범주 간 수치형 변수의 평균을 비교하는데 유용하다.

²² `sns.barplot()` : <https://seaborn.pydata.org/generated/seaborn.barplot.html>

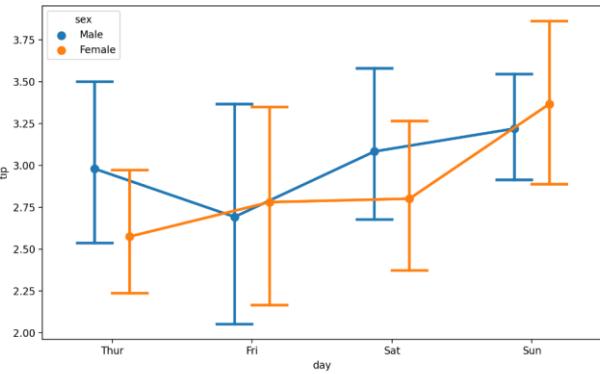
- `sns.countplot()`²³: 이 함수는 범주형 변수의 각 범주에 대한 관측치의 개수를 수직 막대그래프로 보여준다. 각 막대의 높이는 각 범주에서의 관측치의 개수에 해당한다.
- `sns.stripplot()`²⁴: 이 함수는 수치형 변수를 범주형 변수의 축에 따라 데이터 점으로 나타낸다. 이 함수는 범주형 변수의 각 범주에서의 수치형 변수의 분포를 보여주는 데 유용하다.
- `sns.pointplot()`: 이 함수는 수치형 변수를 범주형 변수의 축에 따라 수치 변수의 중심 경향성을 나타내고, 오차 막대를 사용하여 주정치 주변의 불확실성을 표시하는 시각화 기법이다. 하나 이상의 범주형 변수의 여러 수준을 비교하고, 각 변수 간의 교호작용을 표시하는데 유용하다. 공식 문서에서는 `pointplot()`이 `barplot()`보다 범주형 변수의 여러 수준을 비교할 때 더 유용하다고 말한다. 막대의 높이는 평균값을 의미한다.

`pointplot()` 함수 내부의 다양한 파라미터를 적용하여 그래프를 꾸며볼 수 있다. 각 파라미터의 설명은 다음과 같다. 먼저 x 축은 'day', y 축은 'tip'으로 설정한다. `hue` 파라미터를 이용하여 'sex' 칼럼을 기준으로 여성과 남성으로 그룹화한다. `dodge` 파라미터는 여러 개의 그룹이 있을 때 겹치지 않도록 간격을 조정하는 값이며, 소수점으로 변경할 수 있다. `capsize` 파라미터는 에러 바의 끝에 너비의 길이를 정하는 값이다. `errorbar` 파라미터는 `ci`, `pi`, `se`, `sd` 중 하나를 선택하거나 또는 튜플 형태로 작성할 수 있다. 또는 `errorbar`를 숨길 수도 있다(`None`)。

```
fig, ax = plt.subplots(figsize=(10, 6))
sns.pointplot(data=tips, x='day', y='tip', hue='sex',
               dodge=0.25, capsize=0.25, errorbar='sd', ax = ax)
plt.savefig('output/seaborn03.png', dpi=200)
plt.show()
```

²³ `sns.countplot()` : <https://seaborn.pydata.org/generated/seaborn.countplot.html>

²⁴ `sns.stripplot()`: <https://seaborn.pydata.org/generated/seaborn.stripplot.html>



본 데이터셋에 적용해서 house 와 unit 의 errobar 를 적용해서 살펴보도록 한다.
이번에는 2007~2010 년까지의 데이터만 추출하도록 한다.

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

sales = pd.read_csv('data/raw_sales.csv', parse_dates=['datesold'])
sales['year'] = sales['datesold'].dt.year
sales['month'] = sales['datesold'].dt.month

# 2007 ~ 2010 추출
df = sales.copy()
df = df[df['year'].isin([2007, 2008, 2009, 2010])]
df.head()
```

```
datesold postcode price propertyType bedrooms year month
0 2007-02-07 2607 525000 house 4 2007 2
1 2007-02-27 2901 290000 house 3 2007 2
2 2007-03-07 2905 328000 house 3 2007 3
3 2007-03-09 2905 380000 house 4 2007 3
4 2007-03-21 2906 310000 house 3 2007 3
```

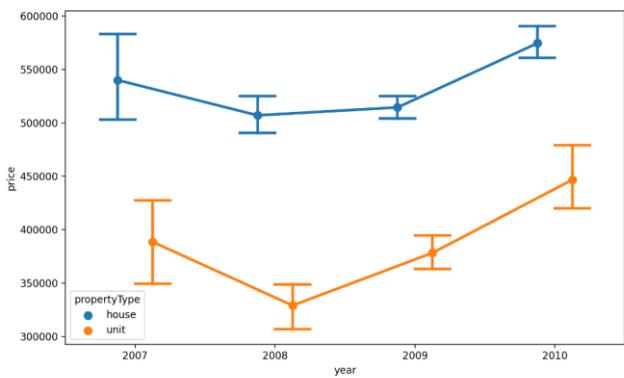
```
df['year'].unique()
```

[결과]

```
array([2007, 2008, 2009, 2010], dtype=int64)
```

위 가공된 데이터에서, x 축은 year, y 축은 price, 그리고 hue 는 propertyType 을 설정하면, year, 그리고 propertyType 으로 비교분석이 가능함을 확인할 수 있다. 그레프를 통해 확인하도록 한다.

```
fig, ax = plt.subplots(figsize=(10, 6))
sns.pointplot(data=df, x='year', y='price', hue='propertyType',
               dodge=0.25, capsize=0.25, errorbar='sd', ax=ax)
plt.savefig('output/seaborn04.png', dpi=200)
plt.show()
```



위 그래프를 보면, house의 평균가격은 항상 unit보다 높았다. 그런데, 연도별 추이를 보면, 2007년에서 2008년 사이의 평균가격을 살펴보면 unit의 하락폭이 house의 하락폭보다 더 크다는 것을 시각적으로 알 수 있다. 또한, 전체적인 그래프의 변동성을 보면 house보다는 unit의 변동성이 더 크며, 이를 통해 unit의 가격이 house의 가격보다 경기에 더 민감하다고 추정할 수 있다.

C. 박스플롯(Boxplot)

seaborn에서 박스플롯은 일종의 분포를 확인하고자 하는 것이다. seaborn에서 박스플롯을 작성할 때는 x 축만 사용할 수도 있고, x 축과 y 축만 사용할 수도 있다. 우선 price 가격의 전체적인 boxplot을 작성하면 다음과 같다. 여기에서 핵심이 되는 코드는 sns.boxplot(x=df["price"], ax = ax)이다. 기존 문법과 다르게, x 축에 series 형태로 입력한 것에 유의한다.

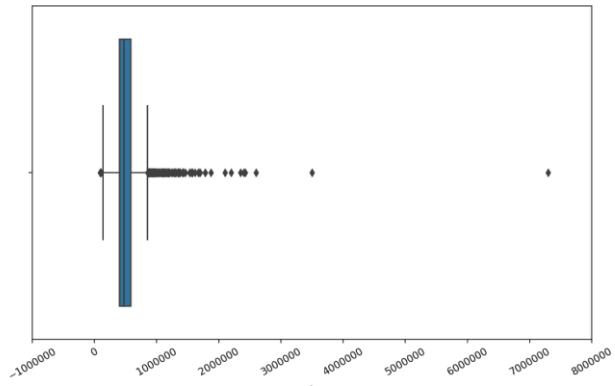
```
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.ticker import ScalarFormatter
formatter = ScalarFormatter()
formatter.set_scientific(False)

sales = pd.read_csv('data/raw_sales.csv', parse_dates=['datesold'])
sales['year'] = sales['datesold'].dt.year
sales['month'] = sales['datesold'].dt.month

# 2007 ~ 2010 추출
df = sales.copy()
df = df[df['year'].isin([2007, 2008, 2009, 2010])]

fig, ax = plt.subplots(figsize=(10, 6))
sns.boxplot(x=df["price"], ax = ax)
ax.set_xticks(ax.get_xticks())
print(ax.get_xticks())
ax.set_xticklabels(ax.get_xticks(), rotation=30)
ax.xaxis.set_major_formatter(formatter)
plt.savefig('output/seaborn05.png', dpi=200)
plt.show()

[결과]
[-1000000.  0.  1000000.  2000000.  3000000.  4000000.  5000000.
 6000000.  7000000.  8000000.]
```

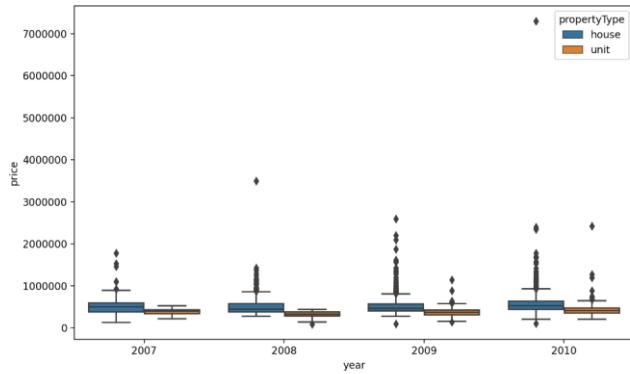


- `ax.get_xticks()`를 출력하면, x 축의 라벨을 리스트로 가져올 수 있다.
- `ax.set_xticklabels(ax.get_xticks(), rotation=30)` : x 축의 라벨을 설정한 후, 각 라벨의 값을 rotation에 맞춰서 방향을 변형할 수 있다.

이번에는 x 축은 year, y 축은 price, hue 는 propertyType 을 지정하면 새로운 연도별 박스플롯을 그릴 수 있다.

```
# 2007 ~ 2010 추출
df = sales.copy()
df = df[df['year'].isin([2007, 2008, 2009, 2010])]

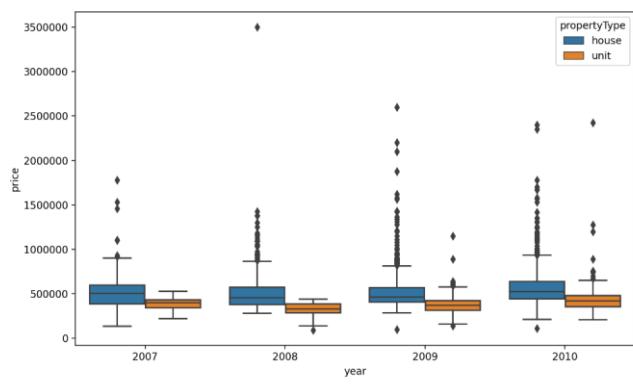
fig, ax = plt.subplots(figsize=(10, 6))
sns.boxplot(data = df, x = 'year', y = 'price',
            hue = 'propertyType', ax = ax)
ax.yaxis.set_major_formatter(formatter)
plt.savefig('output/seaborn06.png', dpi=200)
plt.show()
```



위 그래프에서 보면, 2010년에 700000 이상의 데이터가 존재한다. 일반적으로 이런 데이터가 존재하면 전체적인 분포의 상태를 파악하기 어렵기 때문에 해당 데이터를 제거하고 다시 그라프를 작성할 수 있다. 앞선 그라프보다 더 각 연도별 박스플롯의 그라프가 보다 더 자세히 나타나는 것을 확인할 수 있다.

```
# 2007 ~ 2010 추출
df2 = df[df['price'] <= 700000]

fig, ax = plt.subplots(figsize=(10, 6))
sns.boxplot(data = df2, x = 'year', y = 'price',
            hue = 'propertyType', ax = ax)
ax.yaxis.set_major_formatter(formatter)
plt.savefig('output/seaborn07.png', dpi=200)
plt.show()
```



D. Matplotlib 와 Seaborn 의 관계

지금까지 Seaborn 관련 코드를 작성하면서, Matplotlib의 코드도 계속 반복해서 사용되고 있는 것을 깨달았을 것이다. 실제로도 두개의 코드는 독립적인 라이브러리보다는 오히려 상호 보완적인 측면이 더 강하다. 간단하게 두 개의 라이브러리에 대해 비교 설명을 하면 다음과 같이 정리할 수 있다.

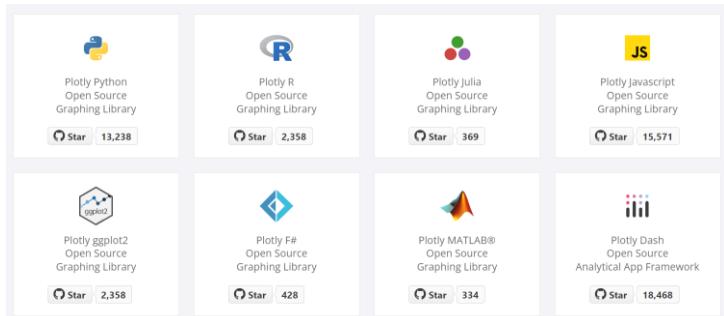
Matplotlib은 2D 및 3D 플롯을 생성하기 위한 광범위한 플롯 기능을 제공하는 저수준 Low-Level 라이브러리이다. 축 레이블, 범례, 색상 등을 포함하여 매우 사용자 지정 가능하며 플롯의 모양을 제어할 수 있다. Matplotlib은 Python에서 데이터 시각화에 필수적인 도구이며 데이터 분석가와 과학자가 광범위하게 사용한다.

반면 Seaborn은 Matplotlib 라이브러리 생태계 위에 구축되는 고수준 High-Level 데이터 시각화 라이브러리이다. 보다 사용자 친화적인 인터페이스를 제공하고 히트맵 Heatmap, 시계열 Time Series 및 회귀선 Regression 같은 일반적인 유형의 그림을 쉽게 만들 수 있습니다. Seaborn은 Violin Plot, Joint Plot, Pair Plots과 같은 고급 통계 시각화도 제공한다..

Matplotlib과 Seaborn을 효율적으로 사용하기 위해서는 각 라이브러리의 장단점을 이해하고 작업에 적합한 도구를 선택하는 것이 중요하다. Matplotlib은 시각화를 사용자에 요구사항에 맞춰서 커스터마이즈하고 복잡한 시각화를 만드는 데 더 적합한 반면, Seaborn은 일반적인 유형의 그래프, 특히 통계와 관련된 그래프를 빠르게 만드는 데 더 적합하다고 볼 수 있다.

5. Plotly

Plotly는 대화형 차트, 그래프 및 대시보드를 생성하기 위한 다양한 도구를 제공하는 데이터 시각화 회사다. 이러한 도구는 데이터 탐색, 프레젠테이션 및 협업에 사용할 수 있습니다. 2013년에 Jack, Chris, Matthew, 그리고 Alex는 캐나다 몬트리올에 이 회사를 설립한다.²⁵ 몬트리올 데이터 과학자, 분석가 및 개발자들 사이에서 인기 있는 기업으로 성장했으며, 오픈 소스 커뮤니티에 뿌리를 둔 회사로서 Plotly는 Python에 웹 기반 데이터 시각화를 도입했습니다. 현재 이 회사는 전 세계 모든 기업이 빠르고 쉽게 데이터 애플리케이션을 구축하고 확장할 수 있도록 지원하는 최고의 소프트웨어 툴과 플랫폼을 제공하는 Dash Enterprise를 제공하고 있다. Plotly는 Python뿐만 아니라, R, Julia, JavaScript 외 다른 언어에서도 사용이 가능하도록 설계 및 관리되고 있다. (그림²⁶ 참조)



A. Graph Objects 와 Plotly Express 차이점

기본적으로 Plotly Express의 모든 객체는 Graph Objects에서 만들어진 것이다. Plotly Express의 객체는 `plotly.graph_object.Figure` 클래스의 인스턴스라고 정의하고 있다. 간단하게 정리하면, Graph Objects는 저수준(Low-Level) 모듈로써, 고급 개발자가 사용하면, 시각화를 다양하게 커스텀화 할 수 있다. 초급 개발자

²⁵ <https://plotly.com/about-us/>

²⁶ <https://plotly.com/graphing-libraries/>

나 아직 `plotly` 생태계가 익숙하지 않은 독자는 `Plotly Express`를 사용하는 것을 권장하고 있다. 공식 홈페이지에서 제공하는 코드를 보면 두 모듈의 코드의 양 차이가 많이 나는 것을 확인할 수 있다.²⁷ 동일한 데이터를 활용하여 막대그래프를 작성하는 예제를 통해서 차이점을 확인하도록 한다.

- Graph Objects 그래프 예제

데이터는 아래와 같이 생성한다.

```
import pandas as pd
df = pd.DataFrame({
    "Fruit": ["Apples", "Oranges", "Bananas", "Apples", "Oranges", "Bananas"],
    "Contestant": ["Alex", "Alex", "Alex", "Jordan", "Jordan", "Jordan"],
    "Number Eaten": [2, 1, 3, 1, 3, 2],
})
df
```

	Fruit	Contestant	Number Eaten
0	Apples	Alex	2
1	Oranges	Alex	1
2	Bananas	Alex	3
3	Apples	Jordan	1
4	Oranges	Jordan	3
5	Bananas	Jordan	2

이제 `Graph Objects`를 통해서 그래프를 작성하도록 한다.

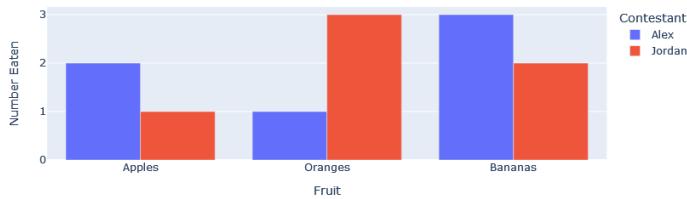
```
import plotly.graph_objects as go

fig = go.Figure()
for contestant, group in df.groupby("Contestant"):
    fig.add_trace(go.Bar(x=group["Fruit"],
                          y=group["Number Eaten"], name=contestant,
                          hovertemplate="Contestant=%s<br>Fruit=%{x}<br>Number Eaten=%{y}<br>% contestant))
```

²⁷ <https://plotly.com/python/graph-objects/>

```
fig.update_layout(title = "graph_objects plot",
                  legend_title_text = "Contestant")
fig.update_xaxes(title_text="Fruit")
fig.update_yaxes(title_text="Number Eaten")
fig.show()
```

graph_objects plot



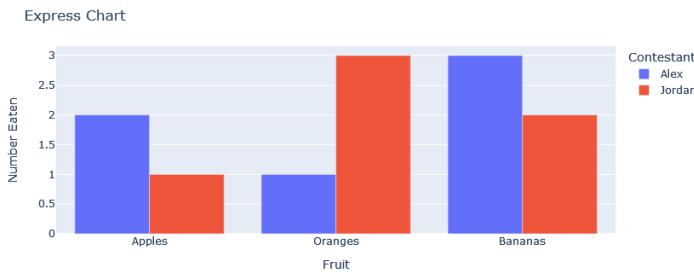
코드 설명을 한다면, 우선 `plotly.graph_objects` 모듈을 `go`로 정의하여 불러온다. 그 후, `go.Figure()`를 통해 빈 그래프 객체를 생성한다. `for` 반복문을 이용하여 `df.groupby()`를 사용하여 "Contestant"를 그룹으로 묶은 데이터를 `contestant`, `group`으로 순서대로 받아온다. 그 다음, `go.Bar()`를 이용하여 각 그룹에 해당하는 막대 그래프를 생성한다. 이 때, "Fruit"를 x축, "Number Eaten"을 y축으로 설정하고, `name`에는 해당 그룹의 "Contestant" 이름을 지정한다. 또한 `hovertemplate`을 이용하여 그래프의 마우스 오버 시 출력되는 정보를 설정한다. 각 그래프를 `fig`에 추가한 뒤, `update_layout()` 함수를 이용하여 범례 제목을 설정한다. `update_xaxes()`와 `update_yaxes()`를 이용하여 각각 x축과 y축의 제목을 설정한다. 마지막으로 `fig.show()`를 실행하여 그래프를 출력한다.

- Plotly Express 그래프 예제

이번에는 Plotly Express 그래프를 통해 작성하면 앞서 Graph Objects에서 작성한 그래프와 결과물은 동일하지만 코드의 양은 매우 절약되었음을 확인한다.

```
import plotly.express as px
fig = px.bar(df, x="Fruit",
```

```
y="Number Eaten", color="Contestant", barmode="group")
fig.update_layout(title = "Express Chart")
fig.show()
```



보통 필자는 plotly를 설명할 때, matplotlib와 seaborn과의 관계를 대입해서 Graph_Objects와 Plotly Express를 설명한다. 보다 세부적으로 그래프를 꾸미고 싶다면 Graph_Objects 모듈을 사용하고, 간단하게 그래프를 작성할 때는 Plotly Express를 사용하면 된다. 만약, Plotly Express 그래프 객체의 세부 옵션을 수정하고 싶다면, Graph_Objects의 문법을 적용하도록 한다. 본 장에서는 Plotly Express를 활용해서 그래프를 작성한다. 그래프를 작성할 때는 가급적 matplotlib와 seaborn에서 그렸던 데이터셋을 그대로 동일하게 구현하였다. 이를 통해서, 각 라이브러리마다 코드가 어떻게 다른지 독자가 판단할 수 있도록 집필 하려고 노력했다.

메모 포함[evan3]: Pandas 코드가 중복이 되는 것 같기는 한데, 그래도 그래도 쓰는게 나을지 판단 부탁

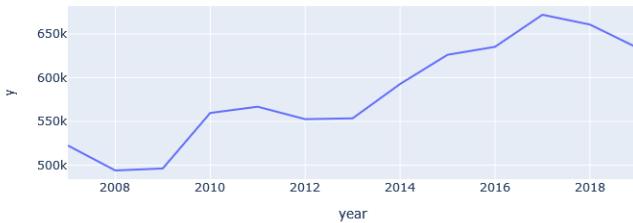
B. 선 그래프 (Line Chart)

선 그래프를 작성할 때는 px.line()²⁸ 메서드를 사용한다.

```
import pandas as pd
import plotly
import numpy as np
import plotly.express as px

sales = pd.read_csv('data/raw_sales.csv', parse_dates=['datesold'])
sales['year'] = sales['datesold'].dt.year
result = sales.groupby('year')['price'].agg(np.mean)
fig = px.line(result, x=result.index, y=result.values,
              title='Avg. House Price per Year With Plotly')
fig.show()
```

Avg. House Price per Year With Plotly



이번에는 2008년, 2018년 데이터만 추출한 뒤, 각 월별 시각화를 작성한다. 이 때, 한 그래프 안에 두개의 라인 그래프가 표현되도록 작성한다. 우선 제공된 데 이터부터 확인한다.

²⁸ 그 외 px.line()에서 사용 가능한 Parameters는 공식문서에서 참조한다.
<https://plotly.com/python-api-reference/generated/plotly.express.line.html>

```
import pandas as pd
import plotly
import numpy as np

sales = pd.read_csv('data/raw_sales.csv', parse_dates=['datesold'])
sales['year'] = sales['datesold'].dt.year
sales['month'] = sales['datesold'].dt.month

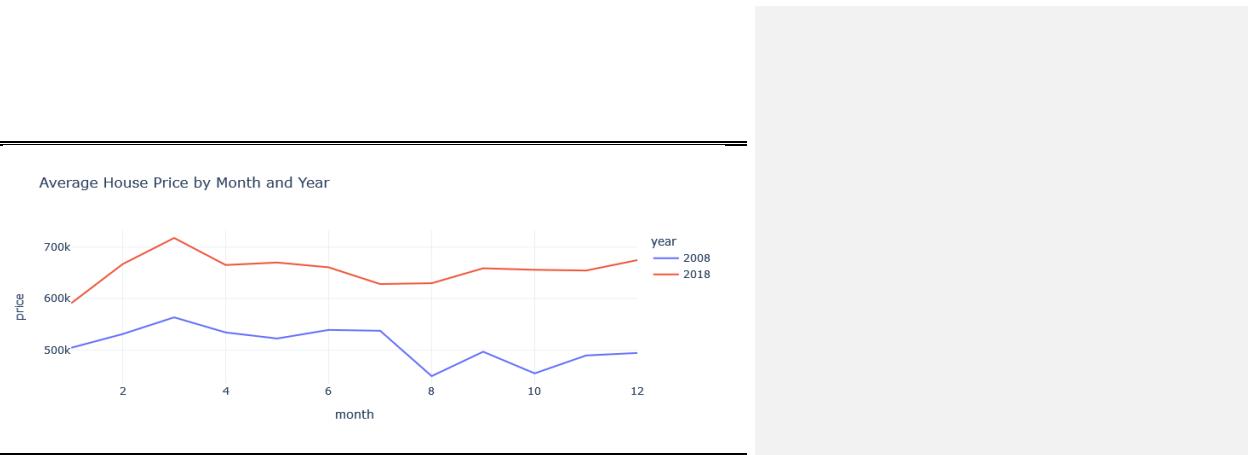
sales = sales[sales['year'].isin([2008, 2018])]
result = np.round(sales.groupby(['year',
'month'])['price'].agg(np.mean).reset_index())
result.head()
```

	year	month	price
0	2008	1	504429.0
1	2008	2	531080.0
2	2008	3	563500.0
3	2008	4	534205.0
4	2008	5	522395.0

이번에는 plotly 테마를 적용해서 시각화를 작성한다. 기존 코드에서 추가된 것은
plotly.io.templates.default 와 color = 'year'을 추가한 것이다.

```
import plotly.express as px
import plotly.io as pio

pio.templates.default = "plotly_white"
fig = px.line(result,
               x='month',
               y='price',
               color='year',
               title='Average House Price by Month and Year')
fig.show()
```



만약, 그 외의 템플릿을 추가하고 싶다면 아래 코드에서 확인할 수 있다.

```
import plotly.io as pio
pio.templates

[결과]
Templates configuration
-----
Default template: 'plotly_white'
Available templates:
['ggplot2', 'seaborn', 'simple_white', 'plotly',
'plotly_white', 'plotly_dark', 'presentation', 'xgridoff',
'ygridoff', 'gridon', 'none']
```

C. 막대 그래프 (Bar Plot)

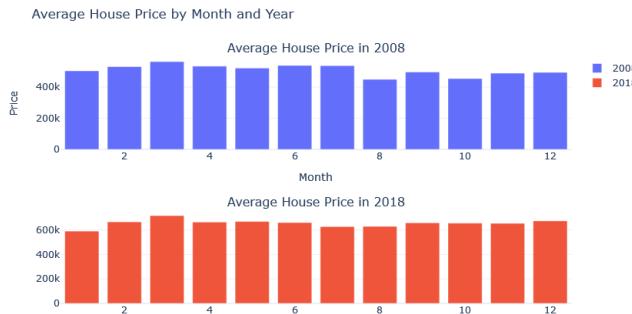
이번에 막대 그래프를 작성할 때는 2008년과 2018년별로 나누어서 그래프를 작성한다. Plotly의 make_subplots 함수는 2개의 행과 1개의 열로 이루어진 서브플롯을 만든다. subplot_titles 매개변수는 각 서브플롯의 제목을 설정하는 데 사용된다. for 루프는 [2008, 2018] 목록의 각 연도를 반복하고 go.Bar 함수를 사용하여 각 연도별로 새로운 막대 그래프를 생성한다. add_trace 함수는 각 막대 그래프를 해당 행에 row 및 col 매개변수를 사용하여 서브플롯에 추가한다. update_layout 함수는 서브플롯의 제목과 축 제목, 그리고 서브플롯의 높이를 설정하는 데 사용된다. 마지막으로, show 함수는 서브플롯을 표시하는 데 사용된다.

```
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import plotly.io as pio

pio.templates.default = "plotly_white"

fig = make_subplots(rows=2, cols=1, subplot_titles=('Average House Price in
2008', 'Average House Price in 2018'))

for i, year in enumerate([2008, 2018]):
    data = result[result['year'] == year]
    fig.add_trace(go.Bar(x=data['month'],
                         y=data['price'],
                         name=str(year)), row=i+1, col=1)
fig.update_layout(title='Average House Price by Month and Year',
                  xaxis_title='Month', yaxis_title='Price', height=500)
fig.show()
```



D. 박스플롯(Boxplot)

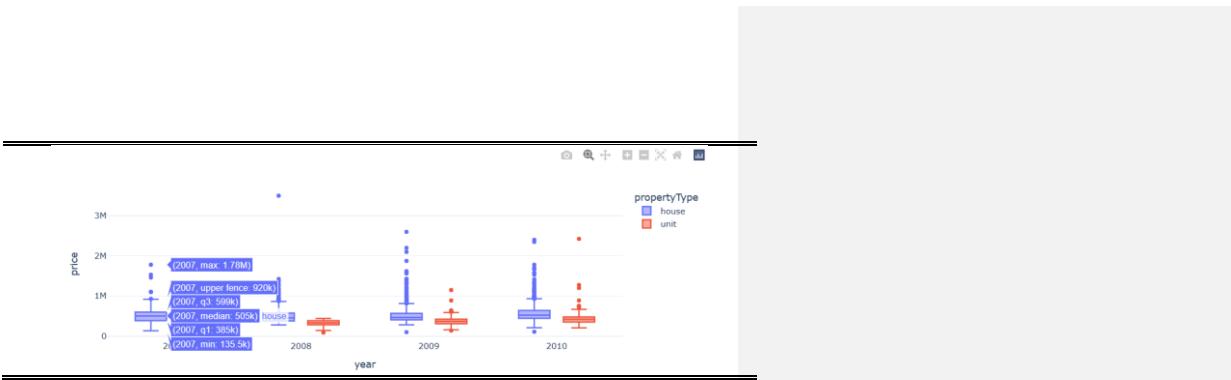
이번에는 박스플롯을 그려보도록 한다. 데이터 가공은 앞서 matplotlib에서 boxplot에서 데이터와 동일하게 나오도록 작성했다. 결과를 통해 확인한다.

```
import pandas as pd
import plotly.io as pio
import plotly.express as px

pio.templates.default = "plotly_white"

sales = pd.read_csv('data/raw_sales.csv', parse_dates=['datesold'])
sales['year'] = sales['datesold'].dt.year
sales['month'] = sales['datesold'].dt.month
sales = sales[sales['year'].isin([2007, 2008, 2009, 2010])]
sales = sales[sales['price'] <= 7000000]

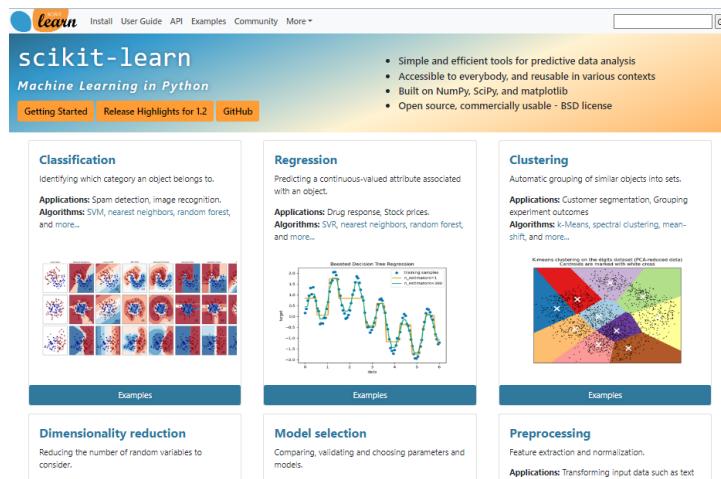
fig = px.box(sales, x="year", y="price", color="propertyType")
fig.show()
```



작성된 그래프 위에 마우스를 위치시키면 (2007년 박스플롯) 최솟값, 1사분위(q1), 2사분위(median), 3사분위(q3), 최대값 등이 나타나는 것을 확인할 수 있다.

6. scikit-learn²⁹

scikit-learn은 파이썬에서 머신러닝을 구현할 때 많이 사용되는 라이브러리 중 하나이다. scikit-learn은 다양한 머신러닝 알고리즘을 제공한다. 회귀, 분류, 군집화, 차원 축소 등의 다양한 문제를 해결할 수 있는 알고리즘이 구현되어 있다. 또한, 데이터 전처리, 모델 선택, 모델 평가 등의 다양한 기능도 제공한다. scikit-learn은 머신러닝을 구현하는 데 필요한 많은 기능을 제공한다. 예를 들어, 데이터를 나누는 함수인 `train_test_split` 함수, 모델을 만드는 함수인 `DecisionTreeClassifier`, `RandomForestClassifier`, `KMeans` 등의 함수들이 대표적이다. scikit-learn은 다른 데이터과학 라이브러리와 함께 사용할 수 있다. 예를 들어, pandas와 함께 사용하면 데이터를 불러오고 전처리하는 과정에서 유용하게 사용할 수 있다. 또한 Deep Learning 프레임워크인 Tensorflow나, Pytorch와도 함께 사용이 가능하다. scikit-learn은 오픈소스 라이브러리로서, 누구나 사용할 수 있고, 다양한 문서와 예제 코드들이 많이 제공되어 있다. 이러한 점들이 scikit-learn을 대중적으로 사용되는 머신러닝 라이브러리 중 하나로 만들었다.



The screenshot shows the official scikit-learn website at <https://scikit-learn.org/stable/>. The header includes links for 'Install', 'User Guide', 'API', 'Examples', 'Community', and 'More'. Below the header, there's a search bar and a 'Go' button. The main content area is titled 'scikit-learn Machine Learning in Python' and features a 'Getting Started' button. It lists several key features:

- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

The page is organized into several sections with examples:

- Classification**: Identifying which category an object belongs to. Includes a grid of small plots showing classification results on various datasets.
- Regression**: Predicting a continuous-valued attribute associated with an object. Includes a line plot showing a regression fit.
- Clustering**: Automatic grouping of similar objects into sets. Includes a scatter plot with colored regions representing different clusters.
- Dimensionality reduction**: Reducing the number of random variables to consider. Includes a small image of a dataset.
- Model selection**: Comparing, validating and choosing parameters and models. Includes a line plot comparing different models.
- Preprocessing**: Feature extraction and normalization. Includes a small image of a dataset.

²⁹ <https://scikit-learn.org/stable/>

A. 머신러닝 Machine Learning 프로세스

머신러닝은 명시적으로 프로그래밍하지 않고도 기계가 데이터를 통해 학습할 수 있도록 하는 학문 분야이다. 머신 러닝 모델을 만드는 과정에는 데이터 수집, 데이터 가공, 모델 선택, 모델 훈련, 모델 성능 평가 등 여러 단계가 포함됩니다.

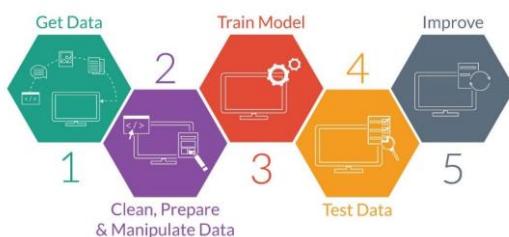
머신 러닝 프로세스의 첫 번째 단계는 데이터를 수집하는 것이다. 이 데이터는 데이터베이스, API 또는 CSV 파일, 엑셀 파일 등 다양한 곳에서 가져올 수 있다. 데이터를 수집한 후에는 데이터가 정확하고 오류나 불일치가 없는지 확인하기 위해 데이터를 정리하고 사전 처리해야 한다.

다음으로 머신러닝 모델을 선택해야 한다. 여기에는 데이터 유형, 데이터 세트의 크기, 원하는 출력과 같은 요소에 따라 당면한 문제에 가장 적합한 알고리즘이나 방법을 선택하는 것이 포함된다.

모델을 선택한 후에는 사전 처리된 데이터를 사용하여 모델을 학습시켜야 한다. 여기에는 모델에 데이터를 입력하고 원하는 출력을 정확하게 예측할 수 있을 때 까지 매개변수를 조정하여 모델이 데이터를 학습할 수 있도록 하는 과정이 포함된다.

마지막으로 예측 모델의 오차 또는 정확성을 파악하기 위해서는 모델의 성능을 평가해야 한다. 모델 평가는 분류와 회귀에 따라 달라진다.

모델이 평가되면 배포하여 새로운 데이터에 대한 예측을 수행할 수 있다. 모델을 배포하고 예측에 사용하는 이 프로세스를 추론이라고 하며, 실제 애플리케이션에서 모델을 사용할 수 있도록 하는 머신 러닝 프로세스의 중요한 부분이다.



B. 회귀와 분류, 평가지표

회귀 Regression와 분류 Classification은 모두 지도학습 Supervised Learning의 일종으로, 독립 변수 Independent Variable과 종속 변수 Dependent Variable 간의 관계를 모델링하는 데 사용된다.

회귀 모델은 일반적으로 선형 회귀 Linear Regression, 다항 회귀 Polynomial Regression 등의 알고리즘을 사용하여 구현된다. 이러한 모델은 입력 변수와 출력 변수 간의 관계를 설명하는 모델링 방법을 학습하고, 새로운 입력 변수를 제공하면 해당 출력 변수를 예측할 수 있다. 회귀 모델의 성능은 주로 평균 제곱 오차 (MSE: Mean Squared Error), 평균 절대 오차 (MAE : Mean Absolute Error), 결정 계수 (R^2 : R-Squared) 등의 지표로 평가한다.

반면, 분류 모델은 일반적으로 로지스틱 회귀 Logistic Regression 등의 알고리즘을 사용하여 구현한다. 이러한 모델은 입력 변수와 이산형 출력 변수 간의 관계를 설명하는 모델링 방법을 학습하고, 새로운 입력 변수를 제공하면 해당 출력 변수(예: 카테고리, 클래스, 레이블 등)를 예측할 수 있다. 분류 모델의 성능을 평가하기 위해서는 우선 혼동행렬 Confusion Matrix를 생성해야 한다.

		Predicted Class		
		Positive	Negative	
Actual Class	Positive	True Positive (TP) <i>Type II Error</i>	False Negative (FN)	Sensitivity $\frac{TP}{(TP + FN)}$
	Negative	False Positive (FP) <i>Type I Error</i>	True Negative (TN)	Specificity $\frac{TN}{(TN + FP)}$
	Precision $\frac{TP}{(TP + FP)}$	Negative Predictive Value $\frac{TN}{(TN + FN)}$		Accuracy $\frac{TP + TN}{(TP + TN + FP + FN)}$

성능은 주로 정확도 Accuracy, 정밀도 Precision, 재현율 Recall, F1 점수 F1 Score 등의 지표로 평가된다. 정확도는 모델이 올바르게 분류한 데이터 샘플의 비율을 나타내며, 정밀도는 모델이 예측한 양성 클래스 중에서 실제 양성 클래스인 비율을, 재현율은 실제 양성 클래스 중에서 모델이 정확하게 예측한 비율을 나타냅니다. F1 점수는 정밀도와 재현율의 조화 평균으로 계산되며, 불균형한 클래스 분포에서 유용하다.

C. 시계열 데이터 예측 Time Series Forecasting

시계열 예측은 기업에서 또는 정부기관에서 자주 사용하는 기법 중의 하나이다. 특히, 미래의 동향을 예측하고, 재고를 계획하고, 판매를 예측하고, 리소스를 최적화할 수 있도록 지원하기 때문에 비즈니스 의사 결정의 중요한 측면을 담당하고 있다. 본 프로젝트에서도 부동산 실거래가를 일자별로 받아서 다음 몇일을 예측하는 코드를 구현하였다. 시계열 예측에 적합한 알고리즘을 선택하는 것은 어려운 작업이다. 특히, 계절성, 추세 및 노이즈와 같은 데이터의 특성에 따라 서로 다른 알고리즘이 수행되기도 한다. 본 장에서는 전통적인 시계열 모델링 방법론인 ARIMA/SARIMA, 최신 머신러닝 알고리즘인 LightGBM, Facebook에서 개발한 Prophet을 사용하였다.

- ARIMA

ARIMA(Auto-Regressive Integrated Moving Average) 모델은 시계열 데이터를 기반으로 한 분석 기법이다. ARIMA 모델은 ARMA(Auto-Regressive Moving Average) 모델을 일반화한 것으로, 과거의 관측 값과 오차를 사용해서 현재의 시계열 값을 설명한다. 이 모델은 분기/반기/연간 단위로 다음 지표를 예측하거나 주간/월간 단위로 지표를 리뷰하며 트렌드에 이상치가 없는지를 모니터링하는 데 사용된다.

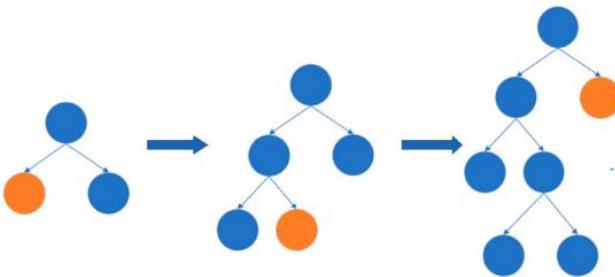
ARIMA모델인 안정적인 시계열(Stationary Series)에만 적용 가능한 ARMA 모델과 달리, 분석 대상이 다소 비안정적인 시계열(Non Stationary Series)의 특징을 보여도 적용이 가능하다. 하지만, 비안정적인 시계열에서는 로그를 이용하거나 차분을 통해 시계열을 안정적으로 변환한 뒤에 분석을 진행하는 것이 좋다.

여기에서 안정적인 시계열이란 시간의 추이와 관계없이 평균 및 분산이 불변하거나 시점 간의 공분산이 기준시점과 무관한 형태의 시계열이다. 비안정적인 시계열은 그와 반대로 시간의 추이에 따라 평균이나 분산이 변하는 형태이다. ARIMA 모델은 경제학에서 많이 사용되며, 주가 전망이나 수요 예측 등에 활용된다. ARIMA 모델은 다른 시계열 분석 기법과 함께 사용된다.

- LightGBM

LightGBM은 트리 기반 학습 알고리즘을 사용하여 광범위한 기계 학습 문제를 해결하는 오픈 소스 그레이디언트 부스팅 프레임워크이다. 마이크로소프트에서 개발했으며 대규모 데이터 세트에 높은 정확도를 제공하면서 효율적이고 빠르고 확장 가능하도록 설계되었다.

LightGBM(Light Gradient Boosting Machine)은 GBDT(Gradient Boosting Decision Tree) 알고리즘을 구현하는 프레임워크로서, 효율적인 병렬 교육, 빠른 교육 속도, 낮은 메모리 소비, 더 나은 정확도 및 대용량 데이터의 신속한 처리를 위한 분산 지원을 지원한다.³⁰ 대부분의 GBDT 도구에서 사용하는 level-wise decision 대신 깊이 제한이 있는 a leaf-wise 알고리즘을 사용한다.

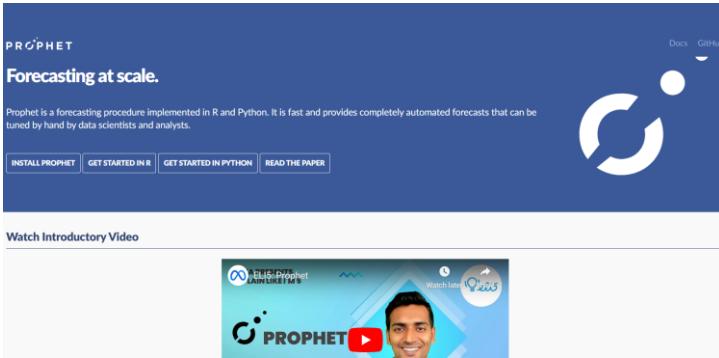


이 전략은 현재 모든 리프 노드에서 분할 이득이 가장 높은 리프를 찾은 다음 이를 분할하며, 이 사이클을 반복합니다. 같은 수의 분할에서 오류를 줄이고 정확도를 높일 수 있다는 것에 장점이 있다. 그러나 위 그림처럼 의사결정 트리가 더 깊게 자라면 과적합_{overfitting}을 초래할 수 있다는 것이 단점이다. LightGBM의 주요 기능 중 하나는 히스토그램 기반 그레이디언트 부스팅 및 잎별 트리 성장과 같은 기술을 사용하여 대규모 데이터 세트를 효율적으로 처리할 수 있다는 것이다. 이를 통해 LightGBM은 다른 그레이디언트 부스팅 프레임워크보다 빠르게 모델을 교육하면서 높은 정확도를 달성할 수 있다. LightGBM의 또 다른 주목할 만한 기능은 일회성 인코딩이나 다른 전처리 단계 없이 범주형 기능을 직접 처리할 수 있다는 것이다. 따라서 범주형 기능이 많은 모델의 메모리 요구 사항에서도 학습 시간을 줄일 수 있다. 또한 LightGBM은 누락된 값 처리, 불균형 데이터 세트 및 기타 일반적인 기계 학습 문제를 처리하기 위한 내장 지원 기능을 제공한다. 분류, 회귀 및 순위 지정을 포함한 광범위한 기계 학습 작업에 사용할 수 있다. 요약하자면, LightGBM은 다양한 기계 학습 문제에 대한 빠르고 정확한 솔루션을 제공하는 강력한 그레이디언트 부스팅 프레임워크이다.

메모 포함[evan4]: 확인 필요

³⁰ Liang, W.; Luo, S.; Zhao, G.; Wu, H. Predicting hard rock pillar stability using GBDT, XGBoost, and LightGBM algorithms. Mathematics 2020, 8, 765.

- Prophet³¹



Prophet은 Facebook에서 개발한 오픈소스 시계열 예측 라이브러리로, 시계열 데이터를 모델링하기 위한 간단하고 직관적인 인터페이스를 제공한다. 계절성, 공휴일, 기타 복잡한 패턴을 포함한 다양한 시계열 데이터를 처리할 수 있도록 설계되었다. 크게 다음과 같은 장점을 가진다.

- **직관적인 인터페이스:** Prophet은 통계나 머신 러닝에 대한 배경 지식이 없는 사용자도 쉽게 사용할 수 있는 시계열 예측용 인터페이스를 제공한다. 표준 데이터 형식을 사용하고 명확하고 간결한 설명서를 제공하므로 시계열 데이터 모델링을 쉽게 시작할 수 있다.
- **유연성:** Prophet은 계절성, 휴일 효과 및 기타 복잡한 패턴이 있는 데이터를 포함하여 광범위한 시계열 데이터를 처리하도록 설계되었습니다. 또한 누락된 데이터와 이상값을 처리할 수 있어 기존의 시계열 예측 방법보다 더 강력하다.
- **투명성:** Prophet은 사용자가 예측의 주요 동인을 이해할 수 있는 투명한 모델링 프로세스를 제공합니다. 사용자가 예측의 정확성과 불확실성을 이해하는 데 도움이 되는 진단 및 시각화를 제공한다.
- **확장성:** Prophet은 확장성이 뛰어나며 고빈도 데이터로 구성된 대규모 데이터 세트를 처리할 수 있다. 또한 장기 예측을 처리할 수 있어 미래 성장을 계획하는 기업과 조직에 유용할 수 있다.

³¹ <https://facebook.github.io/prophet/>

- 오픈 소스 및 커뮤니티 중심: Prophet은 Facebook과 광범위한 데이터 과학 커뮤니티에서 적극적으로 유지 관리하는 오픈 소스 라이브러리이다. 따라서 사용자는 지속적인 업데이트와 개선 사항 뿐만 아니라 다른 사용자의 풍부한 리소스와 지원을 이용할 수 있다.

전반적으로 Prophet은 모든 수준의 사용자에게 간단하고 직관적인 인터페이스를 제공하는 강력하고 유연한 시계열 예측 라이브러리이다. 복잡한 패턴과 대규모 데이터 집합을 처리하도록 설계되어 정확하고 신뢰할 수 있는 예측을 해야 하는 기업, 조직, 연구자에게 유용한 도구이다.

D. ARIMA 를 활용한 주가 데이터 예측

주가 데이터 예측 실습을 위해 yfinance³² 라이브러리를 활용한다. 독자가 원하는 종목을 선택한다.³³

```
import yfinance as yf
TSLA = yf.download("TSLA", start="2019-01-01", end="2023-03-31")
TSLA.head()
```

Date	Open	High	Low	Close	Adj Close	Volume
2019-01-02	20.406668	21.008667	19.920000	20.674667	20.674667	174879000
2019-01-03	20.466667	20.626667	19.825333	20.024000	20.024000	104478000
2019-01-04	20.400000	21.200001	20.181999	21.179333	21.179333	110911500
2019-01-07	21.448000	22.449333	21.183332	22.330667	22.330667	113268000
2019-01-08	22.797333	22.934000	21.801332	22.356667	22.356667	105127500

종가 Close 만 추출한 후, 앞에서 배운 선 그래프 코드를 적용하여 시각화를 진행하도록 한다 (사용 라이브러리 : Matplotlib + Seaborn).

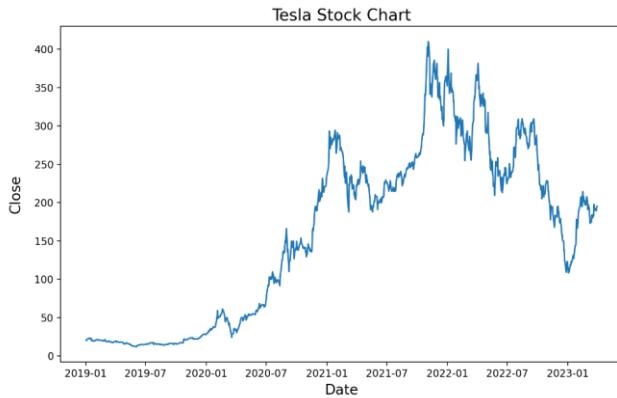
```
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.ticker import ScalarFormatter
formatter = ScalarFormatter()
formatter.set_scientific(False)

fig, ax = plt.subplots(figsize=(10, 6))
sns.lineplot(data=TSLA, x=TSLA.index, y=TSLA.Close, ax=ax)
ax.set_title('Tesla Stock Chart', size = 16)
ax.set_xlabel('Date', size = 14)
ax.set_ylabel('Close', size = 14, labelpad=12)
ax.yaxis.set_major_formatter(formatter)

plt.savefig('output/scikit-learn01.png', dpi=200)
plt.show()
```

³² <https://github.com/ranaroussi/yfinance>

³³ 여담이지만, 필자는 테슬라로 적지 않은 수익을 거둔적이 있다.



시계열 데이터 분석 할 때는 정상성^{Stationarity}을 확인한다.³⁴ 시계열 분석에서 정상성은 시계열 데이터가 시간의 흐름에 따라 통계적 특성이 일정하게 유지되는 것을 의미한다. 보다 구체적으로는 시계열 데이터의 평균과 분산이 일정하게 유지되는 것을 말하며, 정상성이 있는 시계열 데이터는 예측 모델링에 있어서 안정적이고, 일반적으로 확률적인 분석 기법을 적용하기 용이하다. 반면에, 정상성이 없는 시계열 데이터는 추세나 계절성 등이 존재하며, 예측 모델링에 어려움을 끼치는 경우가 있다. 따라서 시계열 분석에서는 정상성 검정을 통해 데이터가 정상성을 갖는지 확인하고, 이를 바탕으로 적절한 모델을 선택한다. 일반적으로 주어진 데이터가 정상성인지, 아닌지 확인하는 방법으로 Dickey-Fuller 검정, Autocorrelation, Partial Autocorrelation을 확인해야 한다. 이를 한꺼번에 확인하는 코드를 사용자 정의 함수로 구현한다.

³⁴ 시계열 분석에서 정상성에 관한 설명을 구체적으로 확인하기 원한다면, <https://otexts.com/fpp3/stationarity.html> 페이지에서 확인하도록 한다.

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import statsmodels.api as sm
import statsmodels.tsa.api as smt
import warnings
warnings.filterwarnings("ignore")

def tsplot(y, lags=None, figsize=(10, 6), tickerName = "stock", outputname =
'output/scikit-learn.png'):
    """시계열 그래프, ACF, PACF, Dickey - Fuller test 계산한다.

    Parameters:
        y : 시계열 데이터
        lags : ACF, PACF, Dickey-Fuller 검정
    """
    if not isinstance(y, pd.Series):
        y = pd.Series(y)

    p_value = np.round(sm.tsa.stattools.adfuller(y)[1], 2)

    fig = plt.figure(figsize=figsize)
    layout = (0.5, 0.5)

    spec = fig.add_gridspec(nrows=2, ncols=2, height_ratios=layout,
width_ratios=layout)

    ### 2. setting axes
    axs = []
    for i in range(len(layout)*len(layout)):
        if i >= 2:
            axs[i] = fig.add_subplot(spec[i//len(layout), i%len(layout)])
            # axs[i].text(0.5, 0.5, f"axs[{i}]",
fontdict={"horizontalalignment":"center", "color":"gray"})
        else:
            axs[i] = fig.add_subplot(spec[i//len(layout), :])

```

```
# axs[i].text(0.5, 0.5, f"axs[{i}]",
fontdict={"horizontalalignment":"center", "color":"gray"})

fig.delaxes(axs[0])

axs[1].plot(y)
axs[1].set_title(f'{tickerName} Stock Time Series Analysis Plots\nDickey-Fuller: p={p_value}')

smt.graphics.plot_acf(y, lags=lags, ax=axs[2])
axs[2].set_ylim(-1.5, 1.5)
axs[2].set_title('Autocorrelation')
smt.graphics.plot_pacf(y, lags=lags, ax=axs[3])

axs[3].set_ylim(-1.5, 1.5)
axs[3].set_title('Partial Autocorrelation')

fig.tight_layout()
plt.savefig(outputname, dpi=200)
plt.show()
```

이 코드는 주어진 시계열 데이터에 대해 시각적으로 검토하고, 그 데이터가 정상적인 시계열인지 확인할 수 있는 방법을 제공하는 함수이다.

- 함수의 인자 중 하나인 y 는 시계열 데이터를 입력으로 받는다. 이 데이터는 Pandas의 Series 형식으로 되어 있어야 하며. 만약 그렇지 않은 경우, 이 함수 내에서 자동으로 Series 형식으로 변환된다.
- 그리고 함수는 주어진 시계열 데이터의 정상성 여부를 파악하기 위해 Augmented Dickey-Fuller 검정을 수행한다. Augmented Dickey-Fuller 검정은 시계열이 정상성을 가지는지 여부를 결정하는 통계 검정 방법 중 하나다. 검정 결과의 p -value는 0과 1 사이의 값을 가지며, p -value가 낮을수록 시계열 데이터는 정상성을 가지고 있다는 것을 의미하고, 그렇지 않으면 계절성, 주기성 등을 가지고 있다고 판단한다. 일반적으로 p -value가 0.05 이하로 판단한다.
- 시계열 데이터에 대한 그래프를 작성한다. ACF와 PACF는 자기상관함수와 편자기상관함수를 나타내는 그래프를 그린다. ACF는 시계열 데이터가 시

간 간격을 두고 얼마나 상관성을 가지는지 보여주는 그래프이고, PACF는 시계열 데이터의 각 시점의 값이 그 전 시점의 값과 얼마나 상관성을 가지고 있는지를 보여주는 그래프입니다. 이 그래프를 통해 시계열 데이터가 자기회귀 모형(AR 모형)과 이동평균 모형(MA 모형)에 적합한지를 확인할 수 있다. 이 함수를 사용하면 주어진 시계열 데이터의 그래프와 정상성 여부를 확인하여, 해당 데이터를 바탕으로 안정적인 모형을 만들 수 있다.

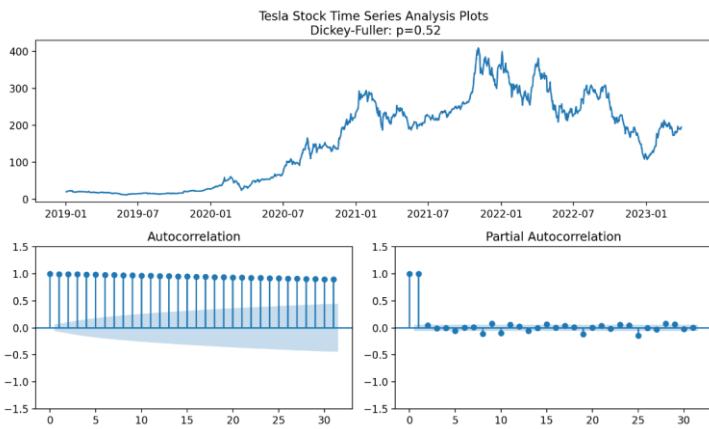
함수 내부의 Matplotlib 코드에 대해 설명하면 다음과 같다.

- 위 코드에서 `fig`은 `plt.figure(figsize=figsize)`에 의해 생성된 새로운 `figure` 객체이다. `figsize`는 생성될 `figure`의 크기를 지정한다.
- `spec = fig.add_gridspec(nrows=2, ncols=2, height_ratios=layout, width_ratios=layout)` 는 `fig`에 그리드를 생성한다. 이 그리드는 2x2의 크기이며, `height_ratios` 와 `width_ratios`는 각각 행과 열의 크기 비율을 나타낸다.
- `axs = {}`는 `spec`에 있는 서브플롯들을 저장하기 위한 딕셔너리이다.
- `fig.delaxes(axs[0])`는 첫 번째 서브플롯을 제거한다.³⁵
- `axs[1]`은 첫 번째 행의 두 번째 열 서브플롯으로, y 시계열 데이터를 그래프로 그리고, 플롯의 제목에는 `tickerName`과 Dickey-Fuller 테스트의 `p-value`가 포함하도록 한다.
- `axs[2]`은 두 번째 행의 첫 번째 열 서브플롯으로, y 시계열 데이터의 자기상관 함수(ACF)를 그래프로 그린다. `smt.graphics.plot_acf` 함수를 사용하여 그래프를 그린다. ACF는 y축을 자기상관계수, x축을 `lag`로 나타냅니다. `lags` 매개변수는 계산할 최대 `lag` 값을 지정한다. `axs[2].set_ylim(-1.5, 1.5)`를 사용하여 y축의 범위를 설정한다.
- `axs[3]`은 두 번째 행의 두 번째 열 서브플롯으로, y 시계열 데이터의 편자기상관 함수(PACF)를 그래프로 그린다.
- `smt.graphics.plot_pacf` 함수를 사용하여 그래프를 그린다. PACF는 y축을 자기상관계수, x축을 `lag`로 나타낸다. `axs[3].set_ylim(-1.5, 1.5)`를 사용하여 y축의 범위를 설정합니다.

³⁵ 이 부분은 약간의 트릭을 사용했다. 2x2 배열 형태로 처음 출력이 되는데, 이 때 상단 2개 중 한 개의 범위를 의도적으로 늘리고, 대신 다른 1개는 지워야만 원하는 형태가 나오도록 설계를 한 것이다. 주석 처리한 코드 '`axs[i].text(0.5, 0.5, f"axs[{i}]", fontdict={"horizontalalignment":"center", "color":"gray"})`' 전체 코드 작성 전, 순차적으로 해당 코드를 실행하면 이유를 알 수 있을 것이다.

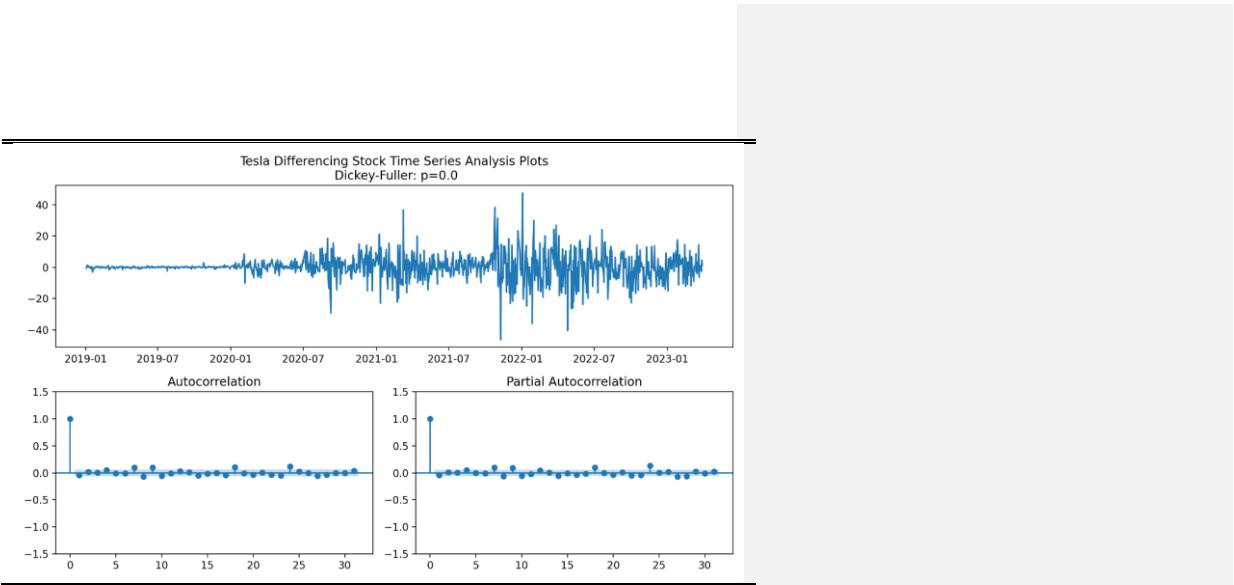
- `fig.tight_layout()`을 사용하여 서브플롯들을 조정하여 서로 중첩되지 않도록 한다.

```
tsplot(TSLA.Close, tickerName = "Tesla", outputname = 'output/scikit-learn02.png')
```



Augmented Dickey-Fuller 테스트를 진행한 결과 `p.value` 값이 0.5 이상 나오기 때문에 정상성이 있다고 판단하기 어렵다. 이런 경우, 일반적으로 차분 Differencing 을 진행한 후, 다시 테스트를 진행한다. 사용자 정의 함수를 사용했기 때문에, 코드는 두 줄이면 충분하다.

```
ts_sun_diff = (TSLA.Close - TSLA.Close.shift(1)).dropna()
tsplot(ts_sun_diff, tickerName = "Tesla Differencing", outputname = 'output/scikit-learn03.png')
```



위 결과를 보면 앞선 그레프와 다른 모양을 띠고 있고, Dickey-Fuller 검정 테스트에서도 p -value 값이 0.5 이하로 정상성을 만족하고 있다는 것을 확인할 수 있다.

- Train/Test

지도 학습에서 훈련/테스트 분할은 데이터 세트를 훈련 세트와 테스트 세트의 두 부분으로 무작위로 분할하는 것이다. 훈련 세트는 모델을 훈련하는 데 사용되며, 테스트 세트는 성능을 평가하는 데 사용된다. 분할은 일반적으로 데이터 세트의 데이터 포인트 순서를 고려하지 않고 무작위로 수행된다.

반면에 시계열에서 훈련/테스트 분할은 데이터 포인트의 순서가 중요하기 때문에 다르다. 시계열 분석에서는 과거 값을 기반으로 미래 값을 예측하는 데 관심이 있다. 따라서 시간적 순서를 보존하는 방식으로 데이터를 분할해야 한다.

가장 일반적인 접근 방식은 '롤링 윈도우 Rolling Window' 접근 방식을 사용하는 것으로, 특정 시점을 기준점으로 하여 시계열의 앞 부분을 학습에 사용하고 후반부를 테스트에 사용하는 것이다. 예를 들어, 주가 데이터의 처음 80%는 훈련에 사용하고 마지막 20%는 테스트에 사용할 수 있다. 이 접근 방식은 시계열 분석에서 중요하게 생각하는 훈련 데이터 이후에 나오는 데이터에 대해 모델을 테스트할 수 있도록 하는 것이다. 전반적으로 시계열 분석에서 훈련/테스트 분할은 데이터의 시간적 순서를 신중하게 고려해야 하므로 일반적인 지도 학습보다 더 복잡하다.

위 내용으로 근거로 모델링 하기 전에 훈련 세트와 테스트 세트로 분할하는 코드를 작성하도록 한다.

```
test_len = int(len(TSLA.Close) * 0.2)
TSLA_train, TSLA_test = TSLA.Close.iloc[:-test_len], TSLA.Close.iloc[-test_len:]
TSLA_train.shape, TSLA_test.shape
```

[결과]
((856,), (213,))

각 분할된 데이터를 확인하도록 한다. 먼저 훈련데이터를 확인한다.

```
TSLA_train[:5]
```

[결과]
Date
2019-01-02 20.674667
2019-01-03 20.024000
2019-01-04 21.179333
2019-01-07 22.330667
2019-01-08 22.356667
Name: Close, dtype: float64

이번에는 테스트 데이터를 확인한다.

```
TSLA_test[:5]
```

[결과]
Date
2022-05-25 219.600006
2022-05-26 235.910004
2022-05-27 253.210007
2022-05-31 252.753326
2022-06-01 246.789993
Name: Close, dtype: float64

- sktime

ARIMA 모델을 활용하여 예측할 때에는 회귀형 모델에서 과거 시계열 값(자동 회귀^{Autoregressive})과 과거 예측 오류(이동 평균^{Moving Average})를 모두 사용한다. 이 모델에는 세 가지 매개 변수 p , d , q 가 있다. 시계열에 가장 적합한 모델을 얻으려면 이러한 매개 변수에 대한 올바른 값을 찾아야 한다.

- p : 모델의 자동 회귀(AR) 부분의 순서로, 모델에 포함될 종속 변수(시계열)의 후행 값 수를 나타낸다. 즉, p 는 미래 값을 예측하는 데 사용되는 시계열의 과거 값의 수이다. 예를 들어 $p=2$ 인 경우 모델은 시계열의 가장 최근 두 값을 사용하여 다음 값을 예측한다.
- d : 모델에 포함된 차이의 정도이다. 차분은 시계열에서 추세 또는 계절성을 제거하여 시계열을 고정시키는 데 사용되는 기법이다. 고정 시계열은 시간에 따라 평균과 분산이 일정하므로 모델링하기가 더 쉽다. 차이 정도인 d 는 시계열의 차이 횟수를 나타냅니다. 예를 들어, $d=1$ 이면 추세를 제거하기 위해 시계열을 한 번 차분하는 것을 말한다.
- q : 모델의 이동 평균(MA) 부분의 순서로, 모델에 포함될 후행 예측 오류의 수를 나타낸다. 즉, q 는 미래 값을 예측하는 데 사용되는 과거 예측 오차의 수다. 예를 들어, $q=2$ 인 경우 모델은 가장 최근의 예측 오차 두 개를 사용하여 다음 값을 예측한다.

p , d , q 는 함께 ARIMA 모델의 구조를 결정하며 모델링하는 시계열의 특성에 따라 선택한다. 이러한 매개 변수를 선택하는 프로세스를 모델 선택이라고 하며, 일반적으로 Akaike 정보 기준(AIC) 또는 베이지안 정보 기준(BIC)과 같은 통계적 방법을 사용하여 수행한다. 이러한 Parameter를 자동으로 찾아주도록 도와주는 라이브러리가 sktime's AutoARIMA이다.

먼저 sktime³⁶는 시계열 데이터를 이용한 머신 러닝을 위한 오픈 소스 Python 라이브러리이다. 시계열 분류, 회귀, 예측, 클러스터링 등 다양한 시계열 작업을 위한 통합 프레임워크를 제공한다. sktime 머신 러닝 라이브러리인 scikit-learn을 기반으로 구축되었으며, 유사한 API를 제공하기 때문에 이미 scikit-learn에 익숙한 사람들도 쉽게 사용하고 배울 수 있도록 제공하고 있다.

³⁶ <http://www.sktime.net/en/latest/>

The screenshot shows the official website for sktime. At the top, there's a navigation bar with links to 'Get Started', 'Documentation', 'Installation', 'API Reference', 'Get Involved', and 'More'. Below the navigation is a search bar with placeholder text 'Search the docs...'. To the right of the search bar are icons for GitHub, Twitter, and LinkedIn.

Welcome to sktime

A unified framework for machine learning with time series.

Mission

sktime provides an easy-to-use, flexible and modular open-source framework for a wide range of time series machine learning tasks. It offers scikit-learn compatible interfaces and model composition tools, with the goal to make the ecosystem more usable and interoperable as a whole. We build and sustain an open, diverse and self-governing community, welcoming new contributors from academia and industry through instructive documentation, mentoring and workshops.

Scope

sktime features a unified interface for multiple time series learning tasks. Currently, we support forecasting, time series classification, time series regression and time series clustering. We have experimental support for time series annotation.

Features:

- API for machine learning with time series, for the purpose of specifying, fitting, applying and

sktime의 인기 기능 중 하나는 ARIMA 모델에서 자동 회귀(AR), 차분(I), 이동 평균(MA) 용어의 순서 선택을 자동화하는 ARIMA 모델 선택 프로세스를 구현한 AutoARIMA입니다. 이 작업은 p, d, q의 가능한 값 범위에서 AIC(Akaike 정보 기준)를 최소화하여 수행된다. AutoARIMA 함수는 시계열을 입력으로 받아 해당 시계열에 맞는 ARIMA 모델을 p, d, q에 대한 최적의 값으로 반환한다. 이 함수는 사용자가 최적의 모델을 찾기 위해 수동으로 다양한 조합을 시도할 필요가 없으므로 시계열 예측 작업에 편리한 도구이다.

이제 코드를 통해 ARIMA 모델을 학습시키는 코드를 구현한다. 최적의 ARIMA 모델은 SARIMAX(9, 1, 2)가 나타난 것을 확인할 수 있다. 모형 학습 전, TSLA_train.index의 값을 train_dates로 저장한다. 이 코드는 향후 시각화 할 때, 재사용이 될 것이다.

```
from sktime.forecasting.arima import AutoARIMA

train_dates = TSLA_train.index
forecaster = AutoARIMA(start_p=8, max_p=9, suppress_warnings=True)
TSLA_train.index = TSLA_train.index.astype(int)
forecaster.fit(TSLA_train)
forecaster.summary()
```

```

SARIMAX Results
Dep. Variable: y No. Observations: 856
Model: SARIMAX(9, 1, 2) Log Likelihood: -2921.474
Date: Mon, 17 Apr 2023 AIC: 5866.948
Time: 22:45:07 BIC: 5923.961
Sample: 0 HQIC: 5888.781
- 856
Covariance Type: opg

```

이제 예측값을 구한 후, 기존 값과 어떤 차이가 나타나는지 확인하는 함수를 작성하도록 한다. 평가모형은 MAE와 MAPE를 동시에 사용한다.

```

from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_absolute_percentage_error

fh = np.arange(test_len) + 1
forecast = forecaster.predict(fh=fh)
coverage = 0.95
forecast_int = forecaster.predict_interval(fh=fh,
coverage=coverage)[['Coverage']][coverage]

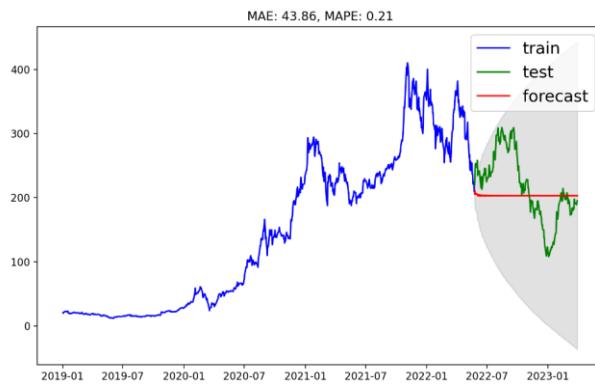
TSLA_train.index = train_dates
forecast.index = TSLA_test.index

mae = np.round(mean_absolute_error(TSLA_test, forecast), 2)
mape = np.round(mean_absolute_percentage_error(TSLA_test, forecast), 2)

fig, ax = plt.subplots(figsize=(12, 6))
ax.plot(TSLA_train.index, TSLA_train.values, label="train", color = 'b')
ax.plot(TSLA_test.index, TSLA_test.values, label="test", color = 'g')
ax.plot(forecast.index, forecast.values, label="forecast", color = 'r')
ax.fill_between(TSLA_test.index,
                forecast_int["lower"],
                forecast_int["upper"])

```

```
alpha=0.2,  
color="dimgray")  
ax.set_title(f'MAE: {mae}, MAPE: {mape}')  
plt.legend(prop={'size':16})  
plt.savefig("output/scikit-learn04.png", dpi=200)  
plt.show()
```



결과 그래프가 말해주듯이, 전통적인 시계열 모형은 잘 맞지 않는 것처럼 보인다. 시계열의 전체적인 기간을 좀 더 축소하면 정확도는 올라갈 수 있다. 그러나, 본 프로젝트에서 중요한 건, 특정한 알고리즘을 계속 테스트하는 것 보다는 다양한 알고리즘을 비교한 후, 최종 모델을 선택하는 과정을 보여주기 위한 것이기 때문에 시계열은 여기에서 일단락 하기로 한다.

- 주요 참고자료

Hyndman, R. J., & Athanasopoulos, G. (2018). *Forecasting: Principles and Practice*. (2nd ed.) OTexts. <https://otexts.org/fpp2/>

E. LightGBM 을 활용한 시계열 데이터 예측

메모 포함[evan5]: 작성 하지 말까 고민중.. 코드 작성하는데 시간 필요.....

이번에는 Nile³⁷ 데이터를 활용하여 LightGBM 모델을 만들고 예측을 하는 코드를 만들어본다. 우선 필요한 라이브러리를 불러오도록 한다.

```
import numpy as np
import pandas as pd

import lightgbm as lgb
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns

import statsmodels.api as sm
import statsmodels.formula.api as smf
import statsmodels.tsa.api as smt
from statsmodels.tsa.stattools import adfuller
from sktime.forecasting.arima import AutoARIMA
from sktime.forecasting.compose import (TransformedTargetForecaster,
                                         make_reduction)
from sktime.forecasting.model_selection import (ExpandingWindowSplitter,
                                                 ForecastingGridSearchCV)
from sktime.forecasting.trend import PolynomialTrendForecaster
from sktime.performance_metrics.forecasting import
MeanAbsolutePercentageError
from sktime.transformations.series.detrend import Deseasonalizer, Detrender
from sktime.utils.plotting import plot_series

import warnings
warnings.filterwarnings("ignore")
```

³⁷ <https://www.statsmodels.org/dev/datasets/generated/nile.html>

각 라이브러리들에 대한 설명은 아래와 같다.³⁸

- lightgbm: Microsoft에서 개발한 경량화된 Gradient Boosting Decision Tree 모델
- statsmodels: 통계 모델링과 검정 등 다양한 통계적 분석 기능을 제공하는 라이브러리
- sktime: 시계열 분석을 위한 라이브러리로, 시계열 예측, 분해, 전처리 등 다양한 기능을 제공함
- warnings: 경고 메시지를 제어하기 위한 파이썬 라이브러리

조금 더 구체적으로 sktime의 모듈에 대해 설명하면 아래와 같다.

- sktime.forecasting.arima
 - ARIMA(AutoRegressive Integrated Moving Average) 모델을 사용한 시계열 예측에 필요한 클래스 및 함수들을 제공하는 모듈이다.
 - AutoARIMA: ARIMA 모델의 하이퍼파라미터(p , d , q)를 자동으로 선택해주는 클래스입니다. Grid search 방식으로 최적의 하이퍼파라미터를 찾아준다.
- sktime.forecasting.compose
 - 시계열 예측 모델들을 조합하여 예측 성능을 높이기 위한 클래스 및 함수들을 제공하는 모듈이다.
 - TransformedTargetForecaster: 전처리(transform)된 시계열 데이터를 입력으로 받아 모델링을 수행하는 클래스이다.
 - make_reduction: 예측 결과를 집계하는(reduce) 함수를 생성해주는 함수입니다. 여러 모델의 예측 결과를 모아서 양상을 모델을 만들 때 사용된다.
- sktime.forecasting.trend
 - 시계열 데이터의 추세(trend)를 예측하기 위한 클래스 및 함수들을

³⁸ 앞에서 배운 pandas, matplotlib, seaborn 라이브러리에 대한 설명은 제외한다.

제공하는 모듈이다.

- `PolynomialTrendForecaster`: 다항식 회귀(Polynomial Regression)를 사용하여 추세를 예측하는 클래스이다.
- `sktime.performance_metrics.forecasting`
 - 시계열 예측 성능 평가에 필요한 함수들을 제공하는 모듈이다.
 - `MeanAbsolutePercentageError`: 예측값과 실제값 사이의 평균 절대 백분율 오차(Mean Absolute Percentage Error, MAPE)를 계산하는 함수이다.
- `sktime.transformations.series.detrend`
 - 시계열 데이터의 추세를 제거하기 위한 클래스 및 함수들을 제공하는 모듈이다.
 - `Deseasonalizer`: 계절성을 제거하기 위한 클래스이다.
 - `Detrender`: 추세를 제거하기 위한 클래스이다.
- `sktime.utils.plotting`
 - 시계열 데이터를 시각화하기 위한 함수들을 제공하는 모듈이다.
 - `plot_series`: 시계열 데이터를 그래프로 그려주는 함수이다.

우선 데이터를 불러오는 코드를 작성한다.

```
import statsmodels.api as sm
import pandas as pd

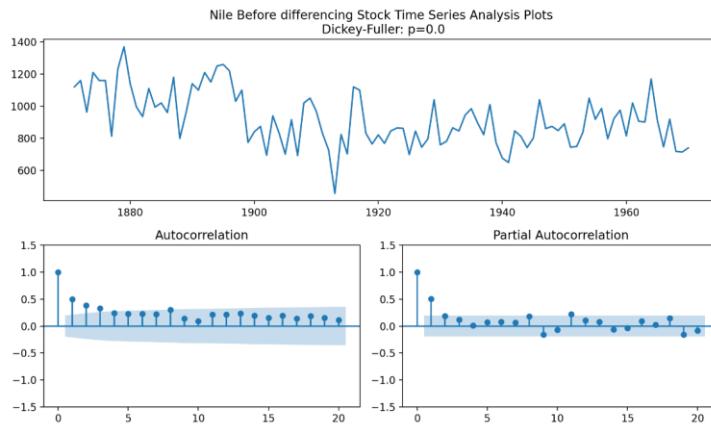
timeseries_df = sm.datasets.get_rdataset("Nile").data
timeseries_df = timeseries_df.set_index('time').value
timeseries_df.head()

[결과]
time
1871    1120
```

```
1872    1160
1873     963
1874    1210
1875    1160
Name: value, dtype: int64
```

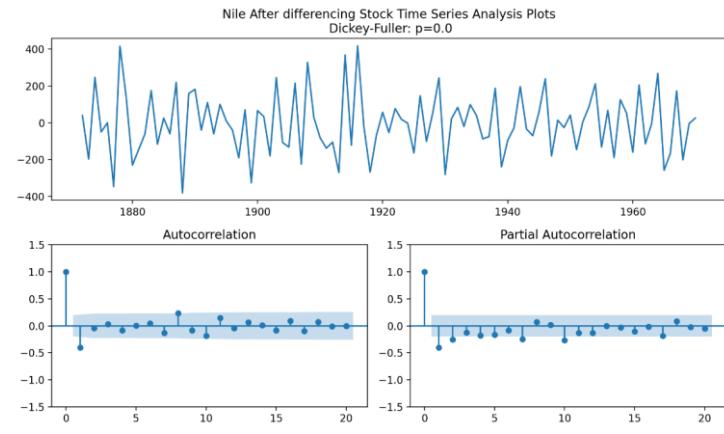
앞에서 이미 구현한 tsplot() 메서드를 호출하여 시각화를 진행한다.

```
tsplot(timeseries_df, tickerName = "Nile Before Differencing", outputname =
'output/scikit-learn05.png')
```



이번에는 차분을 진행한 후, 다시 시각화를 진행한다.

```
ts_nl_diff = (timeseries_df - timeseries_df.shift(1)).dropna()
tsplot(ts_nl_diff, tickerName = "Nile After differencing", outputname =
'output/scikit-learn06.png')
```



이번에는 머신러닝 코드를 구현하기 위해 훈련데이터와 테스트데이터로 분리한다.

```
test_len = int(len(timeseries_df) * 0.3)
train, test = timeseries_df.iloc[:-test_len], timeseries_df.iloc[-test_len:]
train.shape, test.shape
```

[결과]

```
((70,), (30,))
```

LightGBM 모델을 사용하여 Recursive Strategy를 이용한 Time Series Forecasting 모델을 만드는 코드를 구현하도록 한다.

- `lgb.LGBMRegressor()` : LightGBM의 Regressor 모델을 초기화한다.
- `make_reduction(regressor, window_length=5, strategy="recursive")` : LightGBM 모델을 사용하여 Recursive Strategy를 이용한 Time Series Forecasting 모델을 만든다. `window_length`는 이전 데이터 포인트의 수, `strategy`는 예측에 사용되는 방법을 나타낸다. Recursive Strategy는 이

전 예측값을 사용하여 다음 값을 예측하는 방법이다.³⁹

- forecaster : 생성된 Time Series Forecasting 모델을 나타낸다.

```
regressor = lgb.LGBMRegressor()
forecaster = make_reduction(regressor, window_length=5, strategy="recursive")
forecaster
```



정의한 모델을 학습하는 코드를 구현하도록 한다.

```
param_grid = {"window_length": [6, 12]}
cv = ExpandingWindowSplitter(initial_window=int(len(train) * 0.7))
gscv = ForecastingGridSearchCV(
    forecaster, strategy="refit", cv=cv, param_grid=param_grid,
    scoring=MeanAbsolutePercentageError(symmetric=True))
gscv.fit(train)
print(f"best params: {gscv.best_params_}")
```

[결과]

```
best params: {'window_length': 6}
```

코드 설명은 아래와 같다.

- 해당 코드는 Expanding Window Splitter와 Forecasting Grid Search CV를 사용하여 Recursive Strategy를 이용한 Time Series Forecasting 모델의 하이퍼파라미터 튜닝을 수행하는 코드다.
- param_grid : 튜닝할 하이퍼파라미터의 후보 값들을 딕셔너리 형태로 지정한다. 위 코드에서는 window_length 하이퍼파라미터의 후보 값으로 6과 12를 지정했다.
- ExpandingWindowSplitter(initial_window=int(len(train) * 0.7)) : Expanding Window Splitter를 초기화한다. Expanding Window Splitter는

³⁹ https://sktime-backup.readthedocs.io/en/v0.17.1/api_reference/auto_generated/sktime.forecasting.compose.make_reduction.html

Time Series 데이터를 일련의 윈도우로 분할하는 객체다. 초기 윈도우 크기는 전체 데이터의 70%로 설정한다.

- ForecastingGridSearchCV() : Grid Search Cross Validation을 사용하여 모델의 하이퍼파라미터를 튜닝한다. forecaster는 이전에 생성한 Time Series Forecasting 모델이다. strategy는 예측에 사용되는 방법을 나타낸다. cv는 Cross Validation을 수행할 Splitter를 지정한다. param_grid는 튜닝할 하이퍼파라미터의 후보 값들을 지정한다. scoring은 모델의 성능을 평가하는 지표를 지정한다. 위 코드에서는 Mean Absolute Percentage Error (MAPE) 지표를 사용한다.
- gscv.fit(train) : Grid Search Cross Validation을 수행한다. train은 학습 데이터다.
- gscv.best_params_ : 최적의 하이퍼파라미터를 출력한다.
- 따라서, 위 코드는 Recursive Strategy를 이용한 Time Series Forecasting 모델의 하이퍼파라미터 튜닝을 수행하고, 최적의 하이퍼파라미터를 출력하는 코드다.

이번에는 예측 값을 구현하도록 한다.

```
fh = np.arange(len(test)) + 1
y_pred = gscv.predict(fh=fh)
y_pred[:5]
```

[결과]

```
1941    973.096051
1942    895.924521
1943    852.563049
1944    977.141101
1945    838.803281
Name: value, dtype: float64
```

코드 설명은 다음과 같다.

- 튜닝된 Recursive Strategy를 이용한 Time Series Forecasting 모델을 사용하여 테스트 데이터에 대한 예측값을 생성하는 코드다.

- `fh = np.arange(len(test)) + 1` : 테스트 데이터의 길이에 맞게 예측할 미래 시점들을 생성한다. `len(test)`는 테스트 데이터의 길이를 나타내며, `+1`은 미래 시점을 나타내는 값을 1부터 시작하도록 지정한다.
- `gscv.predict(fh=fh)` : Grid Search Cross Validation을 통해 튜닝된 Recursive Strategy를 이용한 Time Series Forecasting 모델을 사용하여 테스트 데이터에 대한 예측값을 생성한다. `fh`는 예측할 미래 시점들을 나타냅니다.
- `y_pred[:5]` : 생성된 예측값 중 첫 5개를 출력합니다.

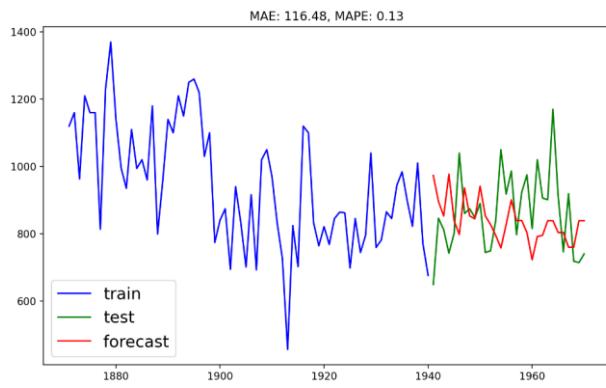
예측 결과값을 시각화로 구현하도록 한다.

```
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_absolute_percentage_error

mae = np.round(mean_absolute_error(test, y_pred), 2)
mape = np.round(mean_absolute_percentage_error(test, y_pred), 2)

fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(train.index, train.values, label="train", color = 'b')
ax.plot(test.index, test.values, label="test", color = 'g')
ax.plot(y_pred.index, y_pred.values, label="forecast", color = 'r')

ax.set_title(f'MAE: {mae}, MAPE: {mape}')
plt.legend(prop={'size':16})
plt.savefig("output/scikit-learn07.png", dpi=200)
plt.show()
```



참고 : <https://towardsdatascience.com/multi-step-time-series-forecasting-with-arima-lightgbm-and-prophet-cc9e3f95dfb0>

참고 : <https://www.kaggle.com/code/satyads/auto-regressor-lightgbm-sktime>

F. Prophet 을 활용한 Airline 데이터 예측

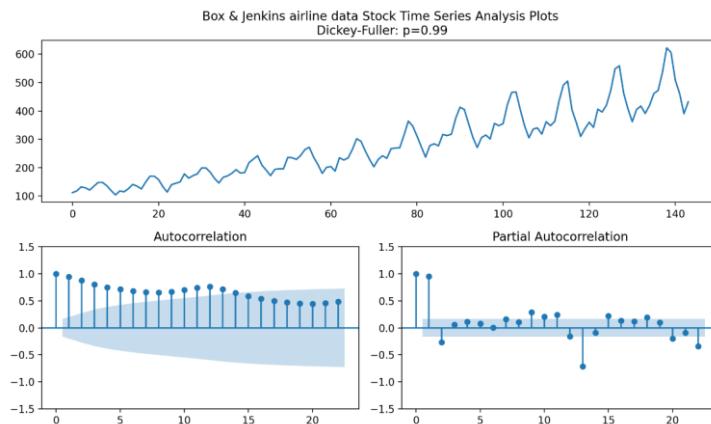
Box-Jenkins Airline 데이터셋 실습을 위해 관련 데이터셋을 불러온다.⁴⁰

```
from prophet import Prophet
from sktime.datasets import load_airline
airline_df = load_airline().to_timestamp(freq="M").reset_index()
airline_df
```

Period	Number of airline passengers	
0	1949-01-31	112.0
1	1949-02-28	118.0
2	1949-03-31	132.0
3	1949-04-30	129.0
4	1949-05-31	121.0

관련 데이터셋을 시각화하면 다음과 같이 나온다. 앞서 정의한 tsplot()을 활용해보도록 한다.

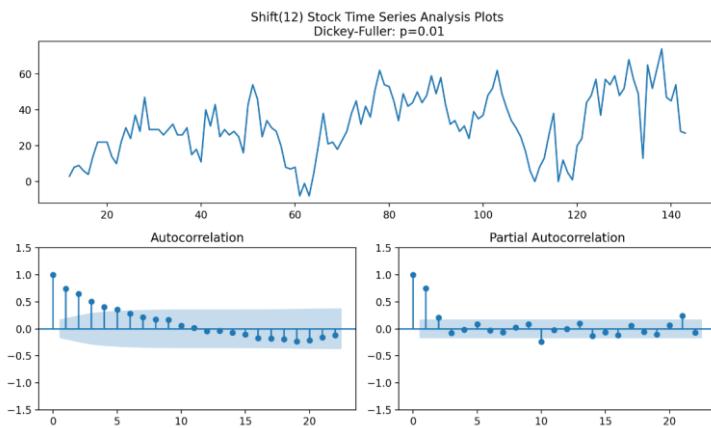
```
tsplot(airline_df['Number of airline passengers'], tickerName = "Box & Jenkins
airline data", outputname = 'output/scikit-learn08.png')
```



⁴⁰ <https://rdrr.io/r/datasets/AirPassengers.html>

위 데이터를 보면 정상성은 만족하지 못하기 때문에, 차분을 진행해야 한다. 차분을 진행할 때는 Seasonality가 보이기 때문에, shift(12)로 차분하고 진행하면 정상성을 확보할 수 있다.

```
ts_al_diff = (airline_df['Number of airline passengers'] - airline_df['Number of airline passengers'].shift(12)).dropna()
tsplot(ts_al_diff, tickerName = "Shift(12)", outputname = 'output/scikit-learn09.png')
```



훈련데이터셋과 테스트데이터셋으로 분리한다.

```
test_len = int(len(airline_df) * 0.3)
train, test = airline_df.iloc[:-test_len], airline_df.iloc[-test_len:]
train.shape, test.shape
```

```
[결과]
((101, 2), (43, 2))
```

이번에는 prophet에 맞는 데이터 형태로 변형하는 코드를 작성한다. prophet 라

이브러리는 컬럼명을 `ds`와 `y`로 지정해줘야 한다.⁴¹ 따라서 아래와 같이 데이터로 변형한다.

```
train.columns = ["ds", "y"]
train.head()
```

	ds	y
0	1949-01-31	112.0
1	1949-02-28	118.0
2	1949-03-31	132.0
3	1949-04-30	129.0
4	1949-05-31	121.0

이제 모형을 만들고 예측 데이터를 생성하도록 한다.

```
model = Prophet(
    seasonality_mode="multiplicative",
    yearly_seasonality=True,
    weekly_seasonality=False,
    daily_seasonality=False,
)
model.fit(train)

future = model.make_future_dataframe(periods=test_len, freq="M")
forecast = model.predict(future)
forecast = forecast.iloc[-test_len:]
forecast = forecast.rename(columns={"yhat_lower": "lower", "yhat_upper": "upper"})
forecast.head()
```

	23:00:56 - cmdstamPy - INFO - Chain [1] start processing								
	23:00:56 - cmdstamPy - INFO - Chain [1] done processing								
	ds	trend	lower	upper	trend_lower	trend_upper	multiplicative_terms	multiplicative_terms_lower	multiplicative_terms_upper
114	1958-07-31	397.460432	480.584854	505.430658	397.449274	397.464065	0.239731	0.239731	0.2
115	1958-08-31	400.331253	471.643891	494.993675	400.269891	400.392911	0.208316	0.208316	0.2
116	1958-09-30	403.109468	415.765220	439.682543	402.965662	403.255717	0.060466	0.060466	0.0
117	1958-10-31	405.980290	360.088647	382.840338	405.723757	406.237410	-0.084357	-0.084357	-0.0
118	1958-11-30	408.758505	311.630290	335.845675	408.378029	409.151208	-0.206039	-0.206039	-0.2

⁴¹ https://facebook.github.io/prophet/docs/quick_start.html

코드 설명을 하면 다음과 같다.

- Prophet() 생성자를 이용하여 Prophet 모델 객체를 생성한다.
seasonality_mode는 추세를 모델링하는 데 사용되는 계절성 모드를 지정하며, yearly_seasonality, weekly_seasonality, daily_seasonality는 각각 연간, 주간, 일간 계절성 구성 요소의 사용 여부를 설정한다.
- model.fit(train_pp) 메서드를 호출하여 생성된 Prophet 모델을 학습시킨다. train_pp는 학습에 사용될 시계열 데이터다.
- model.make_future_dataframe(periods=test_len, freq="M") 메서드를 호출하여 시계열 데이터에 대한 예측을 생성할 기간을 포함한 미래의 시계열 인덱스를 만든다. periods 인자는 미래 예측 기간을 설정하며, freq는 인덱스의 빈도를 설정한다.
- model.predict(future) 메서드를 호출하여 미래 시계열 데이터에 대한 예측을 수행한다. future는 앞서 생성한 미래 시계열 인덱스이다.
- forecast.iloc[-test_len:]를 이용하여 테스트 기간에 대한 예측 결과만 추출한다. -test_len:은 마지막 test_len개의 데이터를 의미한다.
- forecast.rename(columns={"yhat_lower": "lower", "yhat_upper": "upper"})를 이용하여 예측 결과의 열 이름을 변경한다. "yhat_lower"와 "yhat_upper"는 Prophet 모델의 결과 중 예측의 최소값과 최대값을 나타냅니다. rename() 메서드를 이용하여 이를 "lower"와 "upper"로 변경합니다.

위 이미지 외에도 다양한 정보를 가진 컬럼이 존재한다.

```
forecast.info()
```

[결과]

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 213 entries, 114 to 326
Data columns (total 16 columns):
 #   Column           Non-Null Count  Dtype  
 ---  --  
 0   ds               213 non-null    datetime64[ns]
 1   trend            213 non-null    float64 
 2   lower             213 non-null    float64 
 3   upper             213 non-null    float64 
 4   trend_lower       213 non-null    float64
```

```
5   trend_upper           213 non-null    float64
6   multiplicative_terms   213 non-null    float64
7   multiplicative_terms_lower 213 non-null    float64
8   multiplicative_terms_upper 213 non-null    float64
9   yearly                213 non-null    float64
10  yearly_lower          213 non-null    float64
11  yearly_upper          213 non-null    float64
12  additive_terms         213 non-null    float64
13  additive_terms_lower   213 non-null    float64
14  additive_terms_upper   213 non-null    float64
15  yhat                  213 non-null    float64
dtypes: datetime64[ns](1), float64(15)
memory usage: 26.8 KB
```

이제 마지막으로 예측이 잘 맞는지 확인하는 시각화 코드를 작성한다. 아래 코드를 설명하면 다음과 같다.

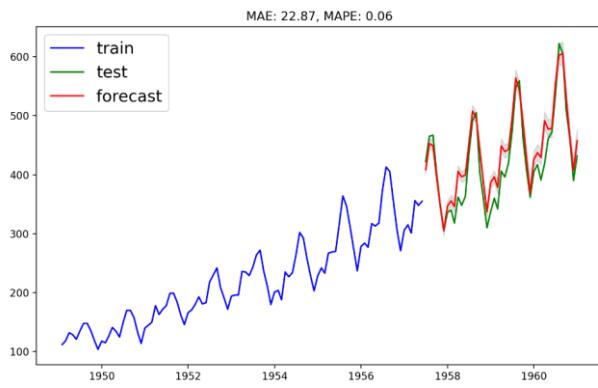
```
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_absolute_percentage_error

test.columns = ["ds", "y"]
forecast.index = test.index

mae = np.round(mean_absolute_error(test['y'], forecast['yhat']), 2)
mape = np.round(mean_absolute_percentage_error(test['y'], forecast['yhat']), 2)

fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(train['ds'], train['y'], label = 'train', color = 'b')
ax.plot(test['ds'], test['y'], label = 'test', color = 'g')
ax.plot(forecast["ds"], forecast["yhat"].values, label="forecast", color = 'r')
ax.fill_between(test["ds"],
                 forecast["lower"],
                 forecast["upper"],
                 alpha=0.2,
```

```
color="dimgray")  
  
ax.set_title(f'MAE: {mae}, MAPE: {mape}')  
plt.legend(prop={'size':16})  
plt.savefig("output/scikit-learn10.png", dpi=200)  
plt.show()  
  
test.columns = ["ds", "y"]  
forecast.index = test.index  
  
mae = np.round(mean_absolute_error(test['y'], forecast['yhat']), 2)  
mape = np.round(mean_absolute_percentage_error(test['y'], forecast['yhat']),  
2)  
  
fig, ax = plt.subplots(figsize=(10, 6))  
ax.plot(train['ds'], train['y'], label = 'train', color = 'b')  
ax.plot(test['ds'], test['y'], label = 'test', color = 'g')  
ax.plot(forecast["ds"], forecast["yhat"].values, label="forecast", color =  
'r')  
ax.fill_between(test["ds"],  
    forecast["lower"],  
    forecast["upper"],  
    alpha=0.2,  
    color="dimgray")  
  
ax.set_title(f'MAE: {mae}, MAPE: {mape}')  
plt.legend(prop={'size':16})  
plt.savefig("output/scikit-learn10.png", dpi=200)  
plt.show()
```



본 대시보드 프로젝트에서는 ARIMA와 prophet만 사용할 예정이다.

Chapter 4. 지리공간 데이터

지리공간 분석(Geospatial Analysis)은 위치나 근접성과 같은 지리적 요소를 가진 데이터를 분석하고 해석하는 프로세스이며. 여기에는 지리 정보 시스템(GIS), 통계 및 기타 분야의 기술을 사용하여 공간 데이터의 패턴, 관계 및 추세를 이해하는 것이 포함된다. 지리공간 분석을 다음과 같은 다양한 용도로 사용할 수 있다.

- 데이터의 지리적 패턴 또는 추세 파악
- 인구 밀도 또는 리소스 활용성 추정
- 질병의 확산 또는 자연재해의 영향 모델링
- 다양한 지리적 영역에서 정책 또는 프로그램의 효과 평가
- 비즈니스, 도시 계획, 공중 보건 등의 분야에서 의사 결정 지원

지리공간 분석에 사용되는 몇 가지 일반적인 기법은 다음과 같다.

- 공간 보간: 주변 데이터 포인트를 기반으로 특정 위치의 값을 추정
- 공간 클러스터링: 공간에서 서로 가까운 데이터 포인트 그룹을 식별
- 공간 회귀: 종속 변수와 하나 이상의 독립 변수 간의 관계를 모델링하는 동시에 공간적 의존성을 고려.
- 네트워크 분석: 여기에는 도로망이나 공급망과 같이 연결된 위치의 네트워크를 따라 이동 또는 흐름을 모델링.

일반적으로 지리공간 분석에서 자주 사용되는 도구는 지리 정보 시스템(GIS: Geographic Information System)을 의미한다. 예를 들면, ArcGIS⁴², QGIS⁴³와 같은 도구들이 사용된다. Python에서는 관련 라이브러리가 다수 존재한다. 간단하게 정리하면 아래와 같다.

- GeoPandas⁴⁴ : Pandas 데이터프레임을 기반으로 하는 지리공간 데이터 작업을 위한 Python 라이브러리이며. 공간 작업과 분석을 수행할 수 있는 사용하기 쉬운 인터페이스를 제공한다. 본 프로젝트에서 활용한다.
- Shapely⁴⁵ : 점, 선, 다각형과 같은 2D 기하 도형 개체로 작업하고 조작

⁴² <https://www.arcgis.com/index.html>

⁴³ <https://www.qgis.org/ko/site/>

⁴⁴ <https://geopandas.org/en/stable/>

⁴⁵ <https://shapely.readthedocs.io/en/stable/>

하기 위한 Python 라이브러리.

- Fiona⁴⁶ : Shapefile과 GeoJSON과 같은 지리공간 데이터 형식을 읽고 쓰기 위한 라이브러리.
- PyProj⁴⁷ : 지리 좌표계와 투영 좌표계간 변환 위한 Python 라이브러리
- GDAL/OGR⁴⁸ : Raster 및 Vector 데이터를 포함한 지리공간 데이터 형식으로 작업할 수 있는 강력한 라이브러리이다. 명령줄 인터페이스를 제공하며 GIS 소프트웨어에서 널리 사용된다.
- Cartopy⁴⁹ : 지도 및 기타 다른 지리공간 데이터 분석 진행을 위해 필요한 데이터 처리를 위해 설계된 라이브러리이며, 특히 proj, NumPy, Shapely, Matplotlib 라이브러리들을 활용하여 다양한 시각화 기능을 제공한다.
- Rasterio⁵⁰ : GIS는 위성 이미지 및 지형 모델과 같은 격자형 Raster 데이터 세트를 구성하고 저장하기 위해 GeoTIFF 및 기타 형식을 사용한다. Rasterio는 이러한 형식을 읽고 쓰며, NumPy N차원 배열과 GeoJSON을 기반으로 하는 Python API를 제공한다.
- PySAL⁵¹ : 지리공간 벡터 데이터에 중점을 둔, Geospatial Data Science 플랫폼 라이브러리로, 일반적으로 공간 클러스터 이상값 탐지, 공간 회귀, 통계 모델링, 공간 계량 경제학, 탐색적 시공간 데이터 분석에 활용되고 있다.
- Geopy⁵² : Geocoding과 역 Geocoding을 위한 라이브러리로 주소의 지리적 좌표를 찾거나 그 반대로 찾는데 활용된다.

⁴⁶ <https://fiona.readthedocs.io/en/latest/index.html>

⁴⁷ <https://pyproj4.github.io/pyproj/stable/>

⁴⁸ <https://gdal.org/>

⁴⁹ <https://scitools.org.uk/cartopy/docs/latest/>

⁵⁰ <https://rasterio.readthedocs.io/en/stable/>

⁵¹ <https://pysal.org/>

⁵² <https://github.com/geopy/geopy>

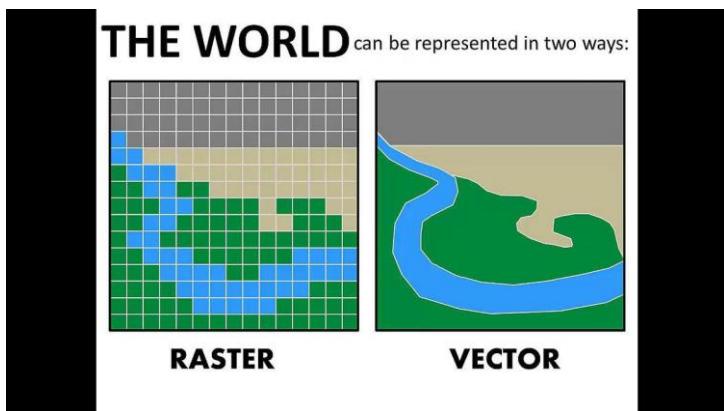
1. 배경지식 Background Knowledge

처음 입문하는 사람에게 지리 정보 시각화를 이해하는 것은 쉬운 것이 아니다.

우선 기본 데이터의 형태가 다르고, 새로운 용어도 익혀야 한다. 여기에서는 기본적인 용어에 대해 서술하도록 하였다.

- Raster vs Vector

Raster와 Vector의 비교에 관한 이미지는 구글에서 쉽게 찾을 수 있다.⁵³



먼저, Raster 데이터는 이미지를 셀 또는 픽셀 그리드로 표현하는 것이며, 각 셀에는 색상 등 특정 속성을 나타내는 값이 포함되어 있다. 특히, 고도 Elevation나 온도와 같은 연속적인 데이터에 가장 적합하며 지도, 위성 이미지, 항공사진과 같은 시각화를 만드는 데 사용한다.

반면에 Vector 데이터는 위치, 크기 모양과 같은 특정 속성을 가진 점 points, 선 lines, 다각형 polygons으로 특징을 나타낸다. 모든 점은 x, y 좌표 coordinate로 나타낸다. 선은 두개 이상의 점들을 순서대로 연결하여 표현하는 것이며, 일반적으로 도로나 강을 표현한다. 마지막으로 다각형은 3개 이상의 점을 연결한 도형이며, 빌딩의 경계나, 호수 등을 표현한다. 즉, Vector는 지구 표면의 특징과 속성을 표현

⁵³ https://storage.googleapis.com/lds-media/images/366022803_1280x720.width-1200.jpg

할 수 있고, 대부분 shapefiles(.shp) 형태로 저장되는 경우가 가장 많다. 속성에는 예를 들면, 건물 이름, 주소, 가격, 건축 날짜 등이 Polygon에 저장될 수 있다. 간단하게 표로 정리하면 아래와 같이 정리가 된다.

Raster	Vector
Pixels	Points, Lines, Polygons
.svg, .shp	.jpg, .png, .tif

- Shapefiles

Shapefile은 지리 정보 시스템(GIS) 소프트웨어에서 널리 사용되는 지리 공간 벡터 데이터 형식을 의미한다. Shapefile은 기본 이름은 같지만 확장자가 다른 세 가지 파일(.shp, .shx, .dbf)로 구성되어 있다. 먼저 .shp 파일에는 점, 선, 다각형과 같은 벡터 피처의 geometry가 포함되어 있다. .shx 파일은 GIS 소프트웨어가 .shp 파일에 빠르게 액세스할 수 있도록 하는 인덱스 파일이다. .dbf 파일에는 각 벡터 피처와 관련된 속성 데이터가 포함되어 있다. 요약하면, .shp파일과 .shx파일은 공간 데이터를 가지고 있고, .dbf 파일은 속성정보를 가지고 있다. 즉, 이 3개의 파일은 개별적인 것이 아니라 하나의 파일로 이해한다.

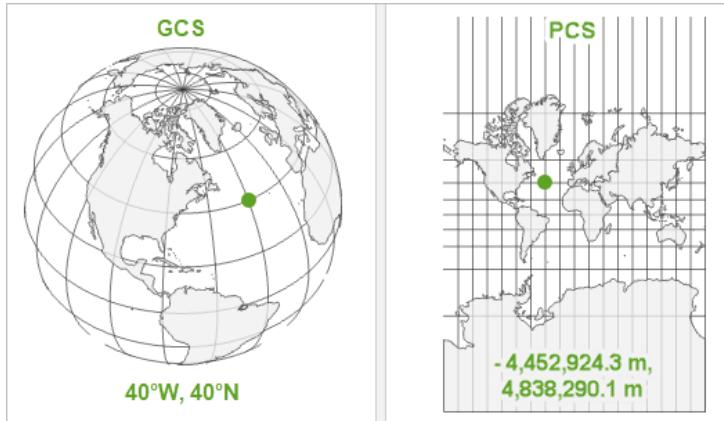
- CRS(Coordinate Reference System)

보통 좌표계로 번역된다. 지구상의 특정한 점을 우리가 알고 있는 위도와 경도의 조합을 나타내는 좌표체계를 의미한다. 우선 GIS 시스템에서 사용되는 좌표 참조 시스템(CRS)에는 여러가지 유형이 존재한다. 여기에서는 크게 세 가지 유형만 확인하도록 한다.

- GCS(Geographic Coordinate Systems) : GCS는 위도 및 경도 좌표를 사용하여 지구 표면의 지형지물의 위치를 정의하는 데 사용된다. WGS84 또는 NAD83 기반으로 하는 다양한 GCS가 존재한다. 일반적으로 GCS는 데이터가 지표면에서 어디에 위치하는지를 정의한다.
- PCS(Projected Coordinate Systems) : 투영 좌표계는 지도나 컴퓨터 화면과 같이 평평한 표면에 지리적 특징을 표현하는 데 사용된다. 투영 좌표계는 수학적 공식을 사용하여 지구의 곡면을 평평한 표면으로 변환하므로 거리, 방향 및 면적에 왜곡이 발생한다. UTM 또는 SPCS와 같은 특정 GCS 및 투영 방법을 기반으로 하는 다양한 PCS가 있다. 일반적으로 PCS는 종이 지도나 컴퓨터 화면과 같이 평평한 표면에 그리는 방법을 데

이터에 알려준다.

아래 이미지를 통해서 GCS와 PCS를 쉽게 비교하는 방법은 아래와 같다.⁵⁴



- VCS(Vertical Coordinate Systems) : VCS는 해수면과 같은 기준 표면 위 또는 아래의 고도 또는 높이를 측정하는 데 사용된다. 일반적으로 미터 또는 피트와 같은 특정 데이텀과 측정 단위를 기반으로 한다. VCS는 흙수 모델링, 항공 및 측량과 같은 애플리케이션에 중요하다. 아래 이미지를 통해 확인한다.⁵⁵



- 국내 지도에서 사용하는 좌표계는 Bessel 1841(EPSG : 4004) : 한국과 일본

⁵⁴ <https://www.esri.com/arcgis-blog/wp-content/uploads/2022/02/grid2.png>

⁵⁵ <https://www.esri.com/arcgis-blog/wp-content/uploads/2018/01/VCS1-300x149.png>

에 적합한 지역타원체를 사용한 좌표계가 있으며, GRS80 UTM-K(EPSG:5179)는 한반도 전체를 하나의 좌표계로 나타낼 때 많이 사용한다. 네이버 지도가 이 좌표계를 사용하는 것으로 알려져 있다. GRS80 중부원점(EPSG:5181)는 다음 카카오가 사용하는 좌표계로 알려져 있다.

- Georeferencing vs Geocoding

Georeferencing은 Vector나 Raster에 좌표를 할당하여 지구 표면의 모델에 투영하는 프로세스이다. 이 프로세스에는 거리 교차로나 랜드마크와 같이 이미지나 지도에서 알려진 위치를 식별하고 현실 세계의 해당 지리적 좌표와 일치시키는 작업이 포함된다. 이를 통해 이미지 또는 지도를 위성 이미지나 벡터 레이어와 같은 다른 공간 데이터와 정렬할 수 있다. Georeferencing은 일반적으로 과거 지도, 항공사진 또는 GPS가 사용되기 이전의 기타 데이터에 사용된다. Python의 Georeferencing 프로세스는 일반적으로 GDAL 라이브러리를 사용하여 수행된다. GDAL(지리공간 데이터 추상화 라이브러리)은 다양한 형식의 지리공간 데이터를 읽고, 쓰고, 조작할 수 있는 강력한 오픈소스 라이브러리이다. GDAL 라이브러리는 서로 다른 좌표계 간에 이미지와 지도를 Georeferencing하고 변환하기 위한 도구를 제공한다. Rasterio 및 OpenCV와 같은 다른 Python 라이브러리도 지오레퍼런싱에 사용할 수 있다.

반면에, Geocoding은 사람이 읽을 수 있는 주소를 일련의 좌표로 변환하는 프로세스를 말한다. 일반적으로 알고리즘과 참조 데이터의 조합을 사용하여 주어진 주소의 위도와 경도를 결정하는 Geocoding 서비스를 사용하여 수행된다. Geocoding은 일반적으로 온라인 맵핑, 내비게이션 및 위치 기반 서비스와 같은 애플리케이션에서 사용된다. Geocoding에 가장 일반적으로 사용되는 Python 라이브러리는 geopy와 geocoder입니다. geopy는 구글 맵, 빙, 오픈스트리트맵 등 다양한 Geocoding 서비스를 사용하여 Geocoding 및 Reverse Geocoding 기능을 제공하는 Python 라이브러리입니다. geocoder는 구글, 빙, 오픈스트리트맵과 같은 다양한 geocoding 서비스를 사용하여 geocoding 기능을 제공하는 또 다른 Python 라이브러리이다.

한편, geopandas 라이브러리는 다른 Python라이브러리와 함께, georeferencing과 geocoding에 모두 사용이 가능하다. Georeferencing된 Raster 데이터를 다양한 형식으로 읽고 조작할 수 있다. 비록 기능은 제한적이지만, geocoding도 geopandas를 통해 수행할 수 있다.

2. GeoPandas

GeoPandas는 기존 Pandas 라이브러리를 지리공간 데이터에 대해 확장한 것으로 이해한다. GeoPandas의 핵심 데이터 구조는 geometry 열을 저장하고 공간 연산을 수행할 수 있는 pandas.DataFrame의 서브클래스인 geopandas.GeoDataFrame로 볼 수 있다. pandas.Series의 서브 클래스인 geopandas.GeoSeries는 geometries를 처리한다. 따라서 GeoDataFrame은 기존 데이터(numeric, bool, text, etc)가 포함된 pandas.Series와 기하 도형(점, 다각형 등)이 포함된 geopandas.GeoSeries의 조합으로 구성된다. 아래 그림을 보면 GeoPandas의 구조를 이해할 수 있다.⁵⁶



A. 기본문법

먼저 geopandas와 함께 제공되는 데이터 집합인 'naturalearth_lowres'를 불러오도록 한다. 이 데이터 집합에는 전 세계 각 국가의 Geometry가 포함되어 있으며, 인구 및 GDP 추정치와 같은 몇 가지 추가 세부 정보가 함께 제공된다.

```
import geopandas as gpd  
  
world_gdf = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))  
world_gdf.head()
```

⁵⁶ https://geopandas.org/en/stable/_images/dataframe.svg

	pop_est	continent	name	iso_a3	gdp_md_est	geometry
0	889953.0	Oceania	Fiji	FJI	549	MULTIPOLYGON (((180.00000 -16.06713, 180.00000..
1	58005463.0	Africa	Tanzania	TZA	63177	POLYGON ((33.90371 -0.95000, 34.07262 -1.05982..
2	603253.0	Africa	W. Sahara	ESH	907	POLYGON ((8.66559 27.65643, -8.66512 27.58948..
3	37589262.0	North America	Canada	CAN	1736425	MULTIPOLYGON (((-122.84000 49.00000, -122.9742..
4	328239523.0	North America	United States of America	USA	2143326	MULTIPOLYGON (((-122.84000 49.00000, -120.0000..

위 데이터를 보면, geometry 컬럼만 제거하면 일반적인 pandas 데이터와 크게 다르지 않다. 이번에는 .shp 파일을 불러오는 코드를 작성한다. 데이터는 <https://www.naturalearthdata.com/downloads/>에서 The Natural Earth Quick Start kit를 다운로드 받는다.

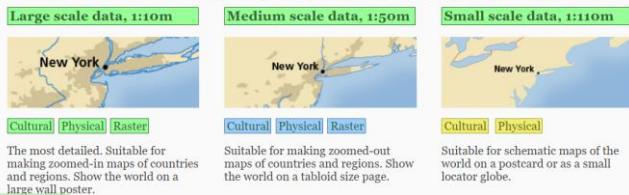
Downloads

Data themes are available in three levels of detail. For each scale, themes are listed on Cultural, Physical, and Raster category pages.

Stay up to date! Know when a new version of Natural Earth is released by subscribing to our [announcement list](#).

Overwhelmed? The Natural Earth quick start kit (219 mb) provides a small sample of Natural Earth themes styled in a QGIS document. Download all vector themes as SHP (576 mb), SQLite (423 mb), or GeoPackage (436 mb).

Natural Earth is the creation of many [volunteers](#) and is supported by [NACIS](#). It is free for use in any type of project. [Full Terms of Use »](#)



압축파일을 열면, 다양한 종류의 데이터가 있다. 그 중에서 각자 원하는 포맷을 가져오면 된다. 여기에서는 모든 미국 주 경계를 가진 Shape 파일을 가져오도록 하였다.

```
import geopandas as gpd
states = gpd.read_file('data/map/ne_110m_admin_1_states_provinces.shp')
states[['adm1_code', 'iso_3166_2', 'name', 'geometry']].head(6)
```

	adm1_code	iso_3166_2	name	geometry
0	USA-3514	US-MN	Minnesota	POLYGON ((-89.95766 47.28691, -90.13175 47.292...
1	USA-3515	US-MT	Montana	POLYGON ((-116.04823 49.00037, -113.05950 49.0...
2	USA-3516	US-ND	North Dakota	POLYGON ((-97.22894 49.00089, -97.21414 48.902...
3	USA-3517	US-HI	Hawaii	MULTIPOLYGON (((-155.93665 19.05939, -155.9080...
4	USA-3518	US-ID	Idaho	POLYGON ((-116.04823 49.00037, -115.96780 47.9...
5	USA-3519	US-WA	Washington	POLYGON ((-117.03143 48.99931, -117.02665 47.7...

데이터의 주요 특징을 확인한다. 먼저, 데이터 타입을 확인한다.

```
print(type(states))

[결과]
<class 'geopandas.geodataframe.GeoDataFrame'>
```

총 데이터의 개수는 51개이고, 컬럼 개수는 122개로 확인된다.

```
states.shape

[결과]
(51, 122)
```

컬럼의 종류는 다음과 같이 출력이 가능하다.

```
states.columns

[결과]
Index(['featurecla', 'scalerank', 'adm1_code', 'diss_me', 'iso_3166_2',
       'wikipedia', 'iso_a2', 'adm0_sr', 'name', 'name_alt',
       ...
       'FCLASS_ID', 'FCLASS_PL', 'FCLASS_GR', 'FCLASS_IT', 'FCLASS_NL',
       'FCLASS_SE', 'FCLASS_BD', 'FCLASS_UA', 'FCLASS_TLC', 'geometry'],
      dtype='object', length=122)
```

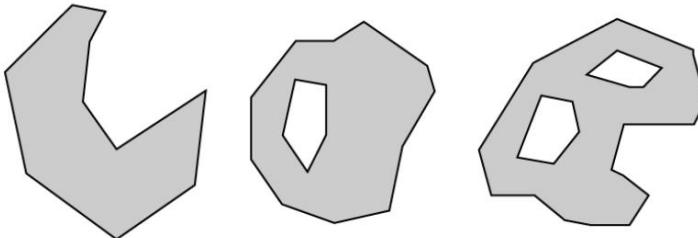
속성 테이블의 처음 5 개의 항목이 Polygon 과 MultiPolygon 으로 구성되어 있다는 것을 보여준다.

```
states.geom_type.head()
```

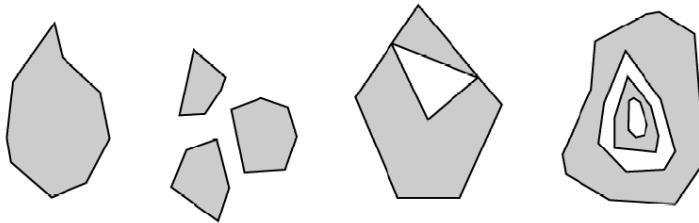
[결과]

```
0      Polygon  
1      Polygon  
2      Polygon  
3  MultiPolygon  
4      Polygon  
dtype: object
```

기본적인 정의는 약간 다르다. 먼저 Polygon 은 다각형은 1 개의 외부 경계와 0 개 이상의 내부 경계로 정의된 평면형 지표 Surface 이다. 각 내부 경계는 다각형의 구멍 Hole 을 정의한다. Polygon 의 샘플 이미지는 아래와 같다. GIS 에서 다각형은 정렬된 x,y 좌표 집합으로 정의되는 2 차원 또는 3 차원 도형이다. 다각형은 하나의 연속된 영역을 나타내는 닫힌 도형이다. 다각형의 예로는 도시 블록, 호수, 들판 등이 있다.



Multipolygon 은 복수의 Polygon 이 존재하는 형태라고 보면 된다. 이러한 Polygon 은 겹치거나, 분리되거나, 중첩될 수 있다. Multipolygon 의 예로는 토지 구획의 집합을 들 수 있는데, 각 구획은 별도의 다각형으로 표시되지만 모든 다각형이 합쳐지면 도시나 국가와 같은 더 큰 영역을 구성한다. Multipolygon 의 샘플 이미지는 아래와 같다.



Polygon 과 MultiPolygon 의 추가적인 설명은 <https://mapscaping.com/polygons-vs-multipolygons-in-gis/> 에서 살펴보기를 바란다.

이번에는 좌표계를 출력해본다. CRS는 공간 데이터셋에 관한 필수 정보를 제공한다. 가장 일반적으로 사용되는 CRS 중 하나는 WGS84 위도-경도 투영법입니다. 권한 코드 "EPSG:4326"을 사용하여 참조할 수 있다. EPSG:4326은 지구 표준 좌표계인 WGS 1984 로도 알려져 있다.⁵⁷

```
states.crs

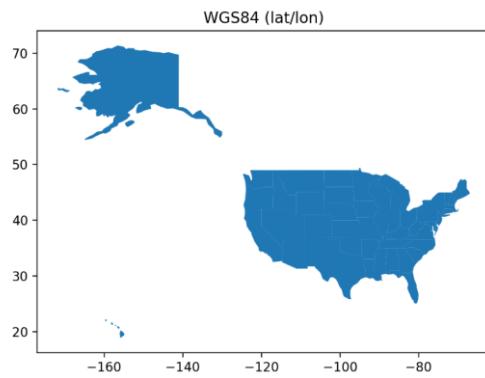
[결과]
<Geographic 2D CRS: GEOGCS["WGS 84",DATUM["WGS_1984",SPHEROID["WGS 84" ...>
Name: WGS 84
Axis Info [ellipsoidal]:
- lon[east]: Longitude (Degree)
- lat[north]: Latitude (Degree)
Area of Use:
- undefined
Datum: World Geodetic System 1984
- Ellipsoid: WGS 84
- Prime Meridian: Greenwich
```

만약 데이터프레임의 지도로 표시하고 싶다면 아래와 같이 시각화가 가능하다.

⁵⁷ 자주 사용하는 코드에 관해서는 <https://spatialreference.org/> 에서 확인한다.

```
import matplotlib.pyplot as plt

ax = states.plot()
ax.set_title("WGS84 (lat/lon)");
plt.savefig('output/map01.png', dpi=200)
plt.show()
```



이번에는 geojson 파일로 내보내기를 해본다.

```
states.to_file('data/my_file.json', driver='GeoJSON')
```

만약, geojson 파일을 불러오고 싶다면 아래와 같이 불러올 수 있다.

```
import geopandas as gpd
new_df = gpd.read_file('data/my_file.json')
new_df[['adm1_code', 'iso_3166_2', 'name', 'geometry']].head(6)
```

	adm1_code	iso_3166_2	name	geometry
0	USA-3514	US-MN	Minnesota	POLYGON ((-89.95766 47.28691, -90.13175 47.292...
1	USA-3515	US-MT	Montana	POLYGON ((-116.04823 49.00037, -113.05950 49.0...
2	USA-3516	US-ND	North Dakota	POLYGON ((-97.22894 49.00089, -97.21414 48.902...
3	USA-3517	US-HI	Hawaii	MULTIPOLYGON (((-155.93665 19.05939, -155.9080...
4	USA-3518	US-ID	Idaho	POLYGON ((-116.04823 49.00037, -115.96780 47.9...
5	USA-3519	US-WA	Washington	POLYGON ((-117.03143 48.99931, -117.02665 47.7...

B. 데이터 맵핑

지도 맵과 가상의 데이터를 합치는 코드를 작성한 후, 시각화를 한다. 먼저, 샘플 지리 정보 데이터를 가져온다. geopandas 라이브러리를 사용하여 Natural Earth 데이터 세트에서 세계 국가의 shape 파일을 읽어온다. 구체적으로, 국가 경계가 있는 세계 지도의 저해상도 버전이 포함된 "naturalearth_lowres" 데이터 세트를 불러와서 "world"라는 GeoDataFrame 객체를 만든다. 코드가 실행되면 '세계' GeoDataFrame에는 각 국가의 모양에 대한 정보와 국가 이름, 인구 또는 기타 통계와 같은 관련 속성이 포함된다.

```
import geopandas as gpd
import pandas as pd
import numpy as np
world = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))
world.head()
```

pop_est	continent	name	iso_a3	gdp_md_est	geometry
889953.0	Oceania	Fiji	FJI	5496	MULTIPOLYGON (((180.00000 -16.06713, 180.00000...
58005463.0	Africa	Tanzania	TZA	63177	POLYGON ((33.90371 -0.95000, 34.07262 -1.05982...
603253.0	Africa	W. Sahara	ESH	907	POLYGON ((-8.66559 27.65643, -8.66512 27.58948...
37589262.0	North America	Canada	CAN	1736425	MULTIPOLYGON (((-122.84000 49.00000, -122.9742...
328239523.0	North America	United States of America	USA	21433226	MULTIPOLYGON (((-122.84000 49.00000, -120.0000...

이번에는 주어진 국가 이름에 가상의 데이터를 생성하도록 한다. 'name' 열은 세계 각 국가의 이름이 포함된 이전에 정의된 세계 GeoDataFrame에서 'name' 열을 선택하여 생성한다. '데이터' 열은 데이터 프레임의 각 행에 대해 100에서 1000 사이의 임의의 정수를 생성하는 numpy의 random.randint() 함수를 사용하여 생성된다. 데이터 열의 길이는 len() 함수를 사용하여 세계 GeoDataFrame의 행 수와 동일하도록 설정한다.

```
data = pd.DataFrame({
    'name': world['name'],
    'data': np.random.randint(100, 1000, len(world))
})
```

```
data.head()
```

	name	data
0	Fiji	491
1	Tanzania	990
2	W. Sahara	892
3	Canada	864
4	United States of America	288

두 개의 데이터 프레임을 하나로 합치는 방법은 여러가지가 있다. 그러나 여기에서는 merge() 메서드를 사용할 것이다. merge() 메서드는 'on' 매개 변수를 사용하여 지정된 공통 열을 기반으로 두 데이터 원본을 결합하는 데 사용된다. 위 데이터에서는 world(GeoDataFrame) 객체와 data(DataFrame) 객체 모두에 있는 'name' 열을 병합하고 있습니다. 일반적으로 실무에서는 두개의 공통된 key 값의 이름이 동일하지는 않다. 따라서, 한쪽에서 맞춰서 컬럼명을 바꿔줘야 한다.

```
world_df = world.merge(data, on = 'name')
world_df.head()
```

	pop_est	continent	name	iso_a3	gdp_md_est	geometry	data
0	889953.0	Oceania	Fiji	FJI	5496	MULTIPOLYGON (((180.00000 -16.06713, 180.00000... 491	
1	58005463.0	Africa	Tanzania	TZA	63177	POLYGON ((39.90371 -0.95000, 34.07262 -1.05982... 990	
2	603253.0	Africa	W. Sahara	ESH	907	POLYGON ((-8.66559 27.65643, -8.66512 27.58948... 892	
3	37589262.0	North America	Canada	CAN	1736425	MULTIPOLYGON (((-122.84000 49.00000, -122.9742... 864	
4	328239523.0	North America	United States of America	USA	21433226	MULTIPOLYGON (((-122.84000 49.00000, -120.0000... 288	

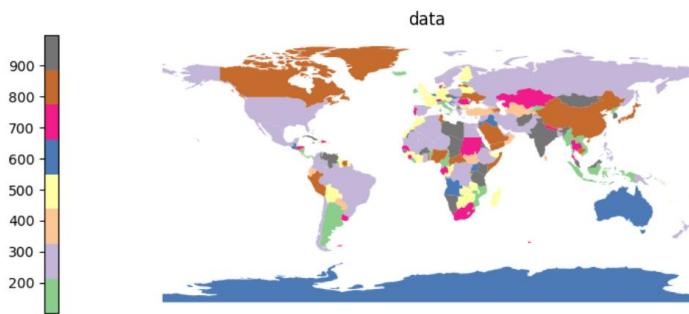
데이터가 합쳐졌기 때문에, 이제 시각화를 작성할 것이다. 먼저 코드와 시각화부터 확인한다.

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(10, 6))
world_df.plot(ax=ax, column="data", cmap="Accent", legend=False, alpha=0.9)
```

```
ax.set_title('data')
ax.set_axis_off()

cb = fig.colorbar(ax.collections[0], ax=ax, location = 'left', shrink = 0.6)
plt.savefig('output/map02.png', dpi=200)
plt.show()
```



메모 포함[evan6]: 이 시각화는 고치지 마세요.... 캡쳐 해서 붙여 넣기 해야 해요.

세계 지도가 있는 그림과 그림 왼쪽에 색상 막대가 있는 그림을 만든다. 먼저, plot() 메서드 안에의 파라미터를 설명하면 다음과 같다. ax 는 하위 플롯을 지정하고, column는 world_df 데이터프레임의 열을 지정하며, cmap 인수는 사용할 컬러맵을 지정하는 세 가지 파라미터를 받는다. 범례 인수는 범례가 표시되지 않도록 하기 위해 False로 설정하고, 선의 투명도를 높이기 위해 alpha 인수를 0.9로 설정한다. 파라미터 location는 컬러맵의 위치를 지정한다. 컬러맵의 위치를 지정 할 때, 'left', 'right', 'top', 'bottom' 등을 지정할 수 있다. shrink키워드 인수는 색상 표가 차지할 공간의 양을 지정합니다. 파라미터 shrink는 컬러맵을 축소한다. 이 때 가능한 값은 0과 1 사이의 실수입니다. 위 코드에서 shrink를 0.6으로 설정하여 컬러맵을 기본 크기의 60%로 축소한다.

C. Geometry 데이터 다루기

지도 맵과 가상의 데이터를 다루도록 한다.

- `geom_type()`

`world.geom_type`은 GeoDataFrame 각 행의 Geometry 유형을 나타내며 GeoSeries 객체로 반환한다. 이 때, Geometry 유형에는 Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon이 올 수 있다.

```
world.geom_type[:5]
```

```
[결과]
0    MultiPolygon
1        Polygon
2        Polygon
3    MultiPolygon
4    MultiPolygon
dtype: object
```

- `geometry`

Geometry타입의 데이터는 area(넓이), boundary(테두리), centroid(중앙지점) 등
의 속성을 가지고 있다. 또한, 한 국가의 영토 등을 여러 개의 점을 이은
Polygon으로 표현할 수 있다. 이 때 데이터 타입은
`shapely.geometry.polygon.Polygon`으로 확인된다. 만약, Polygon에 대해 이해하
려면, Shapely Docs를 확인해야 한다.⁵⁸

```
type(world.geometry[1])
```

```
[결과]
shapely.geometry.polygon.Polygon
```

⁵⁸ <https://shapely.readthedocs.io/en/stable/manual.html#polygons>

간단하게 그래프를 그려보면 다음과 같다.

```
world.geometry[1]
```



위 geometry를 출력하면 Polygon 데이터 값을 뽑을 수 있다.

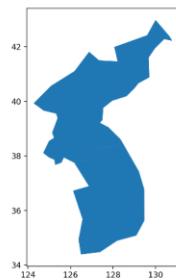
```
print(world.geometry[1])
```

이번에는 world 데이터에서 한반도를 출력하도록 한다. 먼저 world 데이터에서 name 컬럼의 South Korea, North Korea 를 isin() 메서드를 활용하여 필터링 한 후, geometry 컬럼만 GeoSeries 객체로 저장한다. GeoSeries 에 squeeze 메서드를 적용하여 단순한 Shapely LineString 객체로 변환한다. (단일 LineString 객체가 포함된 경우). 이는 boundary 가 입력에 따라 GeoSeries 또는 GeoDataFrame 을 반환할 수 있기 때문이다. squeeze 는 길이 1 축을 스칼라로 축소하는 메서드이다.

```
kor_geom = world[world['name'].isin(['South Korea', 'North Korea'])].geometry  
kor_geom.boundary.squeeze()  
  
[결과]  
95      MULTILINESTRING ((130.78000 42.22001, 130.7800...  
96      LINESTRING (126.17476 37.74969, 126.23734 37.8...  
dtype: geometry
```

이제 한반도를 출력하도록 한다.

```
fig, ax = plt.subplots(figsize=(10, 6))
kor_geom.plot(ax=ax)
plt.savefig('output/map03.png', dpi=200)
plt.show()
```



- Points

이번에는 두 국가의 수도를 위 그래프에 표시하도록 한다. 그러기 위해서는 cities 데이터를 불러와야 한다.

```
cities = gpd.read_file(gpd.datasets.get_path('naturalearth_cities'))
cities.head()
```

	name	geometry
0	Vatican City	POINT (12.45339 41.90328)
1	San Marino	POINT (12.44177 43.93610)
2	Vaduz	POINT (9.51667 47.13372)
3	Lobamba	POINT (31.20000 -26.46667)
4	Luxembourg	POINT (6.13000 49.61166)

cities 객체에서 서울과 평양만을 추출한다.

```
cities[cities['name'].isin(['Seoul', 'Pyongyang'])]
```

name	geometry
154	Pyongyang POINT (125.75274 39.02138)
194	Seoul POINT (126.99779 37.56829)

위 정보를 이용하여 그래프를 출력한다. `kor_geom.plot()` 내부의 `color` 매개변수는 `Polygon` 채우기의 색을 흰색으로 설정하고, `edgecolor`는 `Polygon` 가장자리의 색을 검정색으로 설정한다. 다음으로, 이름 열이 '서울' 또는 '평양'과 일치하는 행만 포함하도록 `cities GeoDataFrame`을 필터링한다. 그런 다음 `plot` 메서드를 사용하여 동일한 축에 `GeoDataFrame`를. `marker` 매개 변수는 마커의 모양을 원으로 설정하고, `color`는 색상을 빨간색으로 설정하며, `markersize`는 마커의 크기를 설정한다. 축에서 `set_axis_off` 메서드를 호출하여 x 축, y 축을 제거한다.

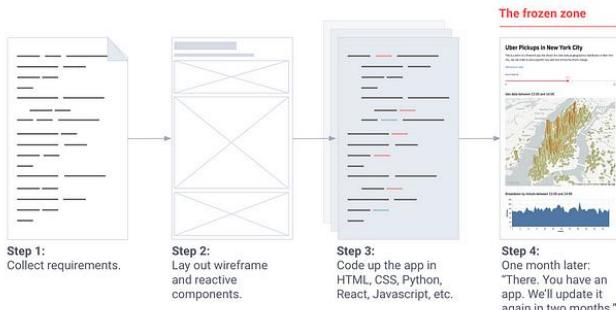
```
fig, ax = plt.subplots(figsize=(10, 6))
ax = kor_geom.plot(color='white', edgecolor="k", ax = ax)
ax = cities[cities['name'].isin(['Seoul', 'Pyongyang'])].plot(ax = ax,
marker='o', color='red', markersize=7)
ax.set_axis_off()
plt.savefig('output/map04.png', dpi=200)
plt.show()
```



Chapter 5. Streamlit

1. Streamlit 라이브러리

Streamlit은 Google 엔지니어였던 Adrien Treuille가 2019년 10월 1일에 세상에 나왔다. 자세한 내용은 그의 첫번째 글을 읽어볼 것을 권한다.⁵⁹ Streamlit은 머신 러닝과 데이터 과학을 위한 멋진 맞춤형 웹 앱을 쉽게 만들고 공유할 수 있는 오픈 소스 Python 라이브러리이다. 짧은 시간 안에 대시보드를 빌드하고 배포할 수 있도록 되어 있다. Streamlit은 Jinja 템플릿 엔진과 React JavaScript 라이브러리를 기반으로 구축되었다. 대회형 웹 앱을 만들기 위한 간단하고 직관적인 구문을 제공한다. 또한, 플로팅, 표, 양식과 같은 일반적인 데이터 과학 작업을 위해 미리 빌드된 여러 위젯이 포함되어 있다.

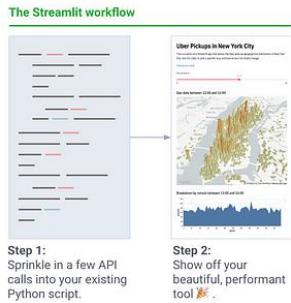


Streamlit이 나온 배경은 웹개발에 능숙하지 못한 데이터 분석가⁶⁰를 위해서다. 위 그림의 Step3 을 보면, 실제 코드 작성은 위해서는 HTML, CSS, Python, React, Javascript, etc 언어들이 활용되어야 한다. 즉, 분석가가 웹개발을 하려면, 기

⁵⁹ <https://medium.com/towards-data-science/coding-ml-tools-like-you-code-ml-models-ddba3357eace>

⁶⁰ 데이터 분석가, 데이터 사이언티스트, 머신러닝 엔지니어 등 다양한 직군에서 대시보드가 필요하지만, 여기에서는 통칭하여 데이터 분석가로 정리하였다.

본적으로 배워야 할 것들이 많다. HTML, CSS, JavaScript, 각 언어별 웹개발 프레임워크 등을 배우라는 것인데, 현실적으로 이 부분이 쉬운 것은 아니다. 그렇다면, 데이터 분석가는 필연적으로 웹개발자와 함께 일을 해야하는데, 일반적으로 작은 회사에서는 인력을 충원하는 것이 쉬운 것은 아니다. 또한 실제 프로젝트를 진행한다 하더라도 분석가와 웹개발팀 사이에 Communication 문제가 항상 존재한다. Adrien Treuille은 개발팀을 tools team이라고 지칭 했는데, 분석가와 개발팀 사이에 존재하는 긴장감을 서술하기도 했다. 이러한 불편함을 해소하기 위해 나온 라이브러리가 Streamlit 인 것이다. 아래 그림에서 볼 수 있는 것처럼 기존 4단계를 2단계로 획기적으로 줄인 것을 확인할 수 있다.



2. Streamlit 핵심 원리

Streamlit의 핵심 원리는 크게 3가지로 요약할 수 있다.⁶¹

- Embrace Python Scripting

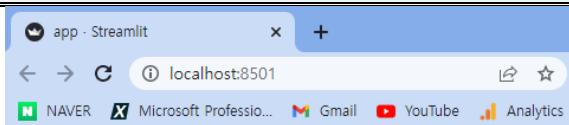
일반적으로 분석가는 Python에서 Jupyter Notebook을 활용해서 진행한다. Jupyter Notebook을 사용하면, 코드 입력-실행을 통해 즉시 결과를 확인할 수 있다. 예를 들면 아래와 같이 코드를 실행할 수 있다.⁶²

파일위치 : ch05/app.py

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

def main():
    st.write("Hello World")

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



Hello World

⁶¹ <https://medium.com/towards-data-science/coding-ml-tools-like-you-code-ml-models-ddba3357eace> 블로그 내용의 일부를 발췌하여 추가적으로 첨언하였다.

⁶² 실행 방법은 해당 파일이 있는 곳에서 터미널에서 streamlit run app.py 를 실행하면 된다. 참고 : <https://docs.streamlit.io/knowledge-base/using-streamlit/how-do-i-run-my-streamlit-script>

- Treat widgets as Variable

대시보드 관련 라이브러리는 종류가 매우 다양하다. R의 Shiny⁶³나 Python의 Dash⁶⁴ 라이브러리에서는 모두 callback 기능을 사용해야 한다. 간단하게 Dash callback 기능을 보여주면 아래와 같다. 코드 하단에 보면 @app.callback 항목이 보인다. 이 영역을 적절하게 처리해야 한다.

```
# -*- coding:utf-8 -*-
from dash import Dash, dcc, html, Input, Output

app = Dash(__name__)

app.layout = html.Div([
    html.H6("Change the value in the text box to see callbacks in action!"),
    html.Div([
        "Input: ",
        dcc.Input(id='my-input', value='initial value', type='text')
    ]),
    html.Br(),
    html.Div(id='my-output'),
])

@app.callback(
    Output(component_id='my-output', component_property='children'),
    Input(component_id='my-input', component_property='value')
)
def update_output_div(input_value):
    return f'Output: {input_value}'

if __name__ == '__main__':
    app.run_server(debug=True)
```

⁶³ <https://shiny.rstudio.com/reference/shiny/1.6.0/onbookmark>

⁶⁴ <https://dash.plotly.com/basic-callbacks>

Streamlit 라이브러리를 활용하면 callback 기능이 없어도 즉시 표현할 수 있다. 아래 코드를 통해 확인해본다. main() 함수의 두 번째 줄인 `x = st.slider('x')`는 사용자가 변수 x의 값을 선택할 수 있는 슬라이더 위젯을 생성한다. main() 함수의 세 번째 줄인 `st.write(x, "x 2 = ", x * 2)`는 앱에 x의 값과 x * 2의 값을 표시한다.

파일위치 : ch05/app.py

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

def main():
    st.title("Hello World")
    x = st.slider('x')
    st.write(x, "x 2 = ", x * 2)

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



Hello World



- Reuse data and computation

데이터를 가지고 다양한 곳에서 연산을 해야한다고 가정할 때, 해당 정보를 안전하게 보존할 수 있도록 도와주는 기능을 cache라고 한다. Cache에 대해 간단하게 설명하면 다음과 같이 정리할 수 있다. Streamlit의 캐시는 데이터를 메모리에

저장하여 더 빠르게 액세스할 수 있도록 하는 방법이다. 이는 대용량 데이터 세트나 자주 업데이트되는 데이터와 같이 자주 액세스하는 데이터에 유용하다. Streamlit에는 두 가지 유형의 캐시가 있다

- 로컬_{Local} 캐시: 사용자의 브라우저에 저장되는 캐시이다. 이 캐시는 사용자가 브라우저를 닫으면 지워진다.
- 글로벌_{Global} 캐시: 서버에 저장되는 캐시이다. 이 캐시는 사용자가 브라우저를 닫을 때 지워지지 않는다.

캐시를 사용하려면 `@st.cache` Decorator⁶⁵를 사용할 수 있습니다. 이 데코레이터는 함수의 출력을 캐시에 저장합니다. 다음에 함수를 호출하면 출력이 다시 계산되는 대신 캐시에서 검색된다. 아래 코드를 통해 확인한다.

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

@st.cache
def get_data():
    data = load_data()
    return data
```

프로젝트의 규모가 커질수록 Cache의 장점은 극대화 된다. 예를 들면, 데이터를 불러오는 데 걸리는 시간을 줄여 앱의 성능을 향상시킬 수 있다. 자주 액세스하는 데이터를 캐싱하여 서버의 부하를 줄일 수 있다. 자주 업데이트 되는 데이터를 캐싱하여 앱의 응답성을 높일 수 있다.

본 책에서는 간단한 대시보드를 제작할 것이기 때문에, cache를 적용하지는 않을 것이다.

⁶⁵ <https://dojang.io/mod/page/view.php?id=2427>

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 파일을 생성하고 하나씩 나열하면서 코드 실행 후, 업데이트 화면을 계속적으로 보여주면서 진행하였지만, 본 장에서는 각 위젯 별로 파일을 만들었다. 파일을 실행할 때는 파일이 있는 곳에서 다음과 같이 실행한 후, <http://localhost:8501>에서 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.

The screenshot shows the Streamlit API reference page. On the left, there's a sidebar with a tree view of API categories: Get started, API reference (selected), Write and magic, Text elements, Data display elements, Chart elements, Input widgets, Media elements, Layouts and containers, Status elements, Control flow, Utilities, Mutate charts, State management, Performance, Personalization, and Connections and databases. The main content area has a title "API reference" and a sub-section "Display almost anything". It contains two code snippets: one for "st.write" and one for "Magic".

```
st.write
Write arguments to the app.

st.write("Hello +world++!")
st.write(my_data_frame)
st.write(my_np1_figure)

Magic
Any time Streamlit sees either a variable or literal value on its own line, it automatically writes that to your app using st.write()

"Hello +world++!"
my_data_frame
my_np1_figure
```

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.

This is Text Elements

This is Header

This is sub Header

This text is colored red, and this is **colored** and bold.

SubChater 1

- $\sqrt{x^2 + y^2} = 1$ is a Pythagorean identity. 📸
-

Chapter 1.

- Streamlit is *really cool*.
 - This text is : blue[colored blue], and this is **colored** and bold.

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.

localhost:8501

HTML CSS 마크다운 적용

이름	나이	직업
Evan	25	데이터 분석가
Sarah	25	프로젝트 오너

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>프로젝트 오너</td>
        </tr>
    </tbody>
</table>
    
```

```
</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_html`의 의미는 공식문서에서 다음과 같이 설명한다. 기본적으로 본문에서 발견되는 모든 HTML 태그는 이스케이프 처리되므로 일반 텍스트로 취급된다. 이 인수를 `True`로 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, 커스터マイ즈된 CSS를 사용하고 싶다면, `style.css`를 불러와서 `st.markdown`에 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.

Data Display st.dataframe()

Use container width

	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_state`의 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.DataFrame`만 올 수 있는 것은 아니다. 클래스 기준으로 말하면, `pandas.Styler`, `pyarrow.Table`, `numpy.ndarray`, `pyspark.sql.DataFrame`, `snowflake.snowpark.DataFrame`, `snowflake.snowpark.table.Table`, `Iterable`, `dict` 형태 등을 입력할 수 있다.

⁷² 매개변수 `axis=1`를 적용하면 각 행의 최댓값이 노란색으로 바뀌게 된다. `Table Visualization`에 관한 추가적인 설명은 `pandas`의 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.

Max Tip

10.0

↑ 9.0

Min Tip

1.0

↓ -9.0

	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 `tip`의 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 하면 다음과 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マイ너스 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 Streamlit에서 지원하는 Chart에 대해 알아본다. 일반적으로 Streamlit은 자체적으로 메서드를 지원하지만, Python에서 제공하는 다양한 시각화 라이브러리들과 연계하는 것을 목표로 하고 있다. Chapter 3장에서 배웠던 시각화 라이브러리들은 모두 지원이 가능하다. 본장에서는 주로 코드의 흐름만 이해하는 정도로만 기술할 것이고, 시각화 코드에 구체적인 설명은 여기에서는 생략하도록 한다. 그 외 자세한 설명은 Chapter 3장의 Matplotlib, Seaborn, 그리고 Plotly에서 확인하기를 바라며, Streamlit Documents에서 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 Streamlit에서 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	<code>st.plotly_chart()</code> ⁷⁷
Bokeh	<code>st.bokeh_chart()</code> ⁷⁸
PyDeck	<code>st.pydeck_chart()</code> ⁷⁹
Graphviz	<code>st.graphviz()</code> 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

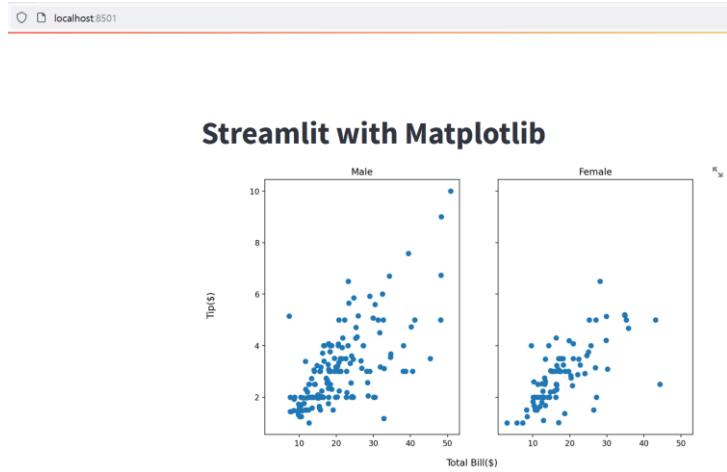
⁷⁸ https://docs.streamlit.io/library/api-reference/charts/st.bokeh_chart

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.

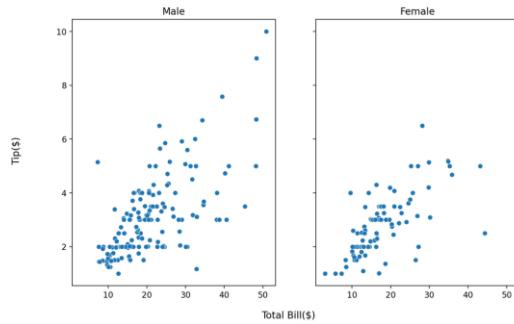
⁸¹ https://matplotlib.org/stable/gallery/subplots_axes_and_figures/figure_title.html

- Seaborn

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

localhost:8501

Streamlit with Seaborn



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 의미한다. .set() 메서드는 Matplotlib에서 객체의 여러 속성을 한 번에 설정할 수 있는 방법이다.

- **Plotly**

이번에는 plotly 라이브러리를 활용하여 streamlit 대시보드에 표현하도록 한다.



Streamlit with Plotly



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 추가하려면 `st.radio()` 함수를 사용한다.

- 파일 업로더: 사용자가 컴퓨터에서 파일을 업로드할 수 있다. Streamlit 앱에 파일 업로더 위젯을 추가하려면 `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.

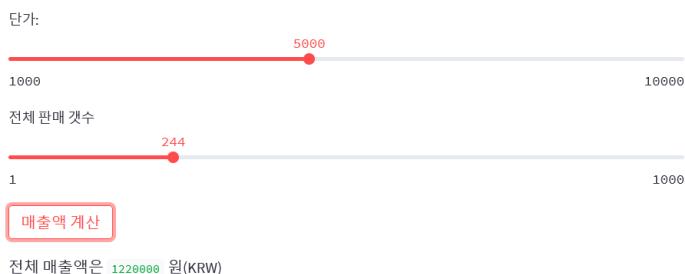
색상 선택기: 사용자가 색상을 선택할 수 있다. Streamlit 앱에 색상 선택기 위젯을 추가하려면 `st.color_picker()` 함수를 사용한다.

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.

Sales Revenue Calculator



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.

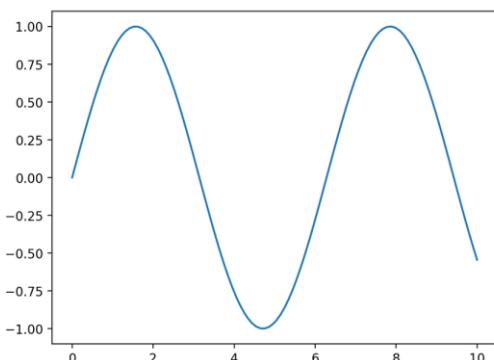
- 먼저, `calculate_sales_revenue` 함수를 정의 한다. 이 함수는 `price`와 `total_sales`라는 두 개의 매개변수를 받는다. 이 함수는 두 인수를 곱하고 결과를 반환한다.
- `st.slider()`을 활용하여 가격에 대한 슬라이더 위젯을 만든다. 슬라이더의 최소값은 1000, 최대값은 10000, 기본값은 5000으로 정했다.
- `st.slider()`을 활용하여 판매된 총 품목 수에 대한 슬라이더 위젯을 생성한다. 슬라이더의 최소값은 1, 최대값은 1000, 기본값은 500으로 정했다.
- `st.button()`함수를 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.checkbox`를 활용해서 사용자가 시각화를 제어하도록 한다.

Check Box Control

시각화 보여주기



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()
```

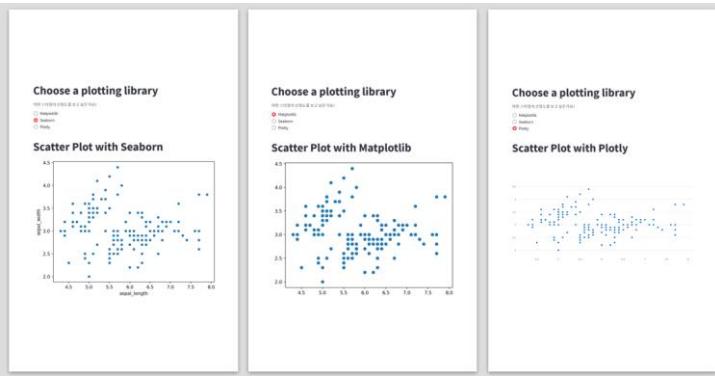
코드 설명은 다음과 같다. 시각화를 작성한 후, if문을 사용하여 show_plot의 값을 확인한다. True인 경우(즉, checkbox가 선택된 경우), st.pyplot()을 사용하여 시각화가 Streamlit에 표시됩니다. 만약 checkbox를 클릭하지 않는다면, 시각화는 사라지게 된다.

Check Box Control

시각화 보여주기

- 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 매개 변수를 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`, `Plotly`이다. 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.

	sepal_length	sepal_width	petal_length	petal_width	species
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
10	5.4	3.7	1.5	0.2	setosa
11	4.8	3.4	1.6	0.2	setosa
12	4.8	3	1.4	0.1	setosa
13	4.3	3	1.1	0.1	setosa
14	5.8	4	1.2	0.2	setosa
15	5.7	4.4	1.5	0.4	setosa
16	5.4	3.9	1.3	0.4	setosa
...

index	sepal_width	species	petal_length
7	3.5	setosa	1.4
8	3	setosa	1.4
9	3.2	setosa	1.3
10	3.1	setosa	1.5
11	3.6	setosa	1.4
12	3.9	setosa	1.7
13	4.9	versicolor	1.4
14	4.4	versicolor	1.4
15	5.2	versicolor	1.3
16	5.8	versicolor	1.3
...

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("1개의 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 HTML 문서에서 내용을 구분하거나 주제의 변화를 정의할 때 쓰는 HTML 태그이다. 데이터 집합의 species 컬럼에 있는 고유 값을 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.

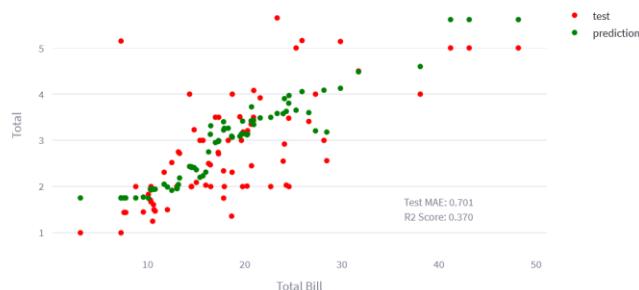
그 후, 데이터의 컬럼 값만 추출하여 st.multiselect() 옵션으로 사용한다. 사용자가 multiselect 상자에서 그림과 같이 선택하면 복수의 컬럼값이 cols로 저장된다. 그 후에 iris.loc 문법을 활용하여 특정 컬럼만 조회가 되도록 하였다.

- 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.



Tip Prediction with RandomForestRegressor



파일위치 : ch05/12_slider_select_slider.py

```
# -*- coding:utf-8 -*-
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 하나
씩 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 상호 작용할 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.

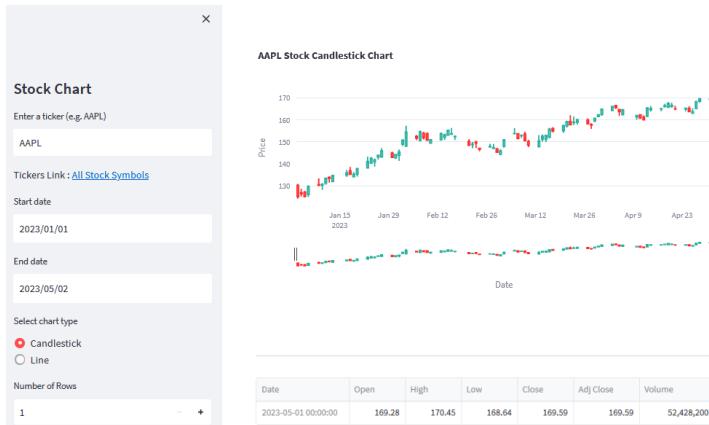
- Widget-6 : st.date_input & st.number_input

st.date_input은 사용자가 특정 날짜를 선택할 수 있는 날짜 선택 위젯을 생성하는 Streamlit에서 제공하는 함수이다. 이 함수는 label, min_value 및 max_value 날짜와 같은 선택적 매개변수를 받는다. 선택한 날짜는 추가 계산이나 시각화에서 사용할 수 있다. 특히, 시계열 관련된 데이터를 다룰 때 자주 활용된다.

st.number_input은 사용자가 특정 숫자를 입력할 수 있는 숫자 입력 위젯을 생성하는 Streamlit에서 제공하는 또 다른 메서드이다. 이 메서드는 label, min_value, max_value, value와 같은 값을 입력받는다. 입력한 숫자는 사칙 연산과 같은 추가 계산이나 데이터 가공 등에 활용할 수 있다.

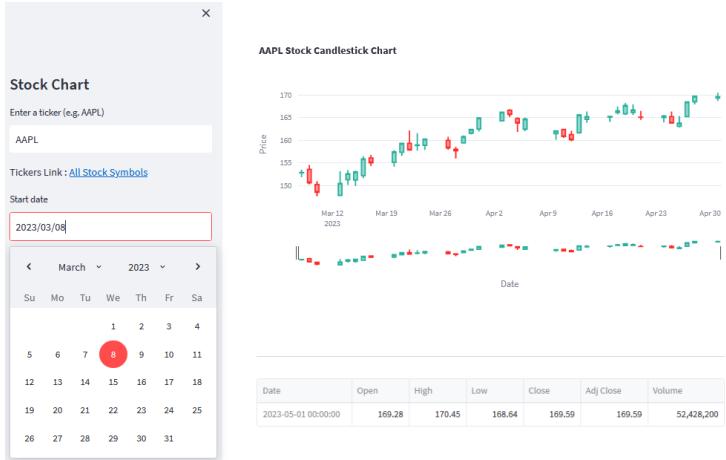
이번에는 날짜 또는 숫자를 입력받아서 다이나믹하게 차트가 변경되도록 하는 대시보드를 구현한다. 주식 관련 대시보드를 간단하게 만들어본다.

- 시각화 전체 화면



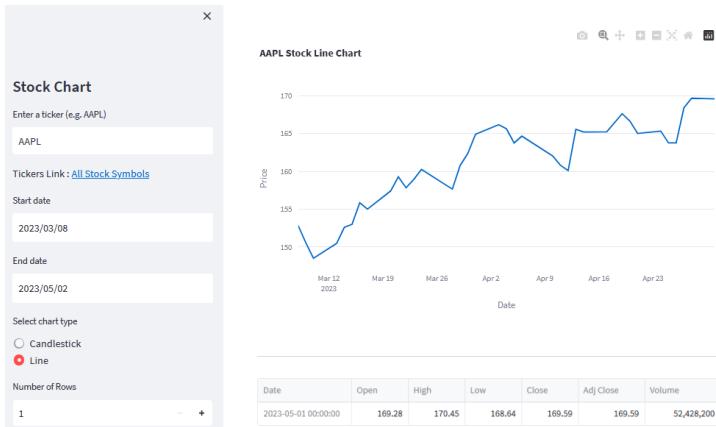
- 날짜 입력 화면

날짜 입력탭을 클릭하면 아래 이미지와 같이 날짜가 선택되어지고, 변경된 날짜에 맞춰서 시각화도 같이 변경되는 것을 확인할 수 있다.



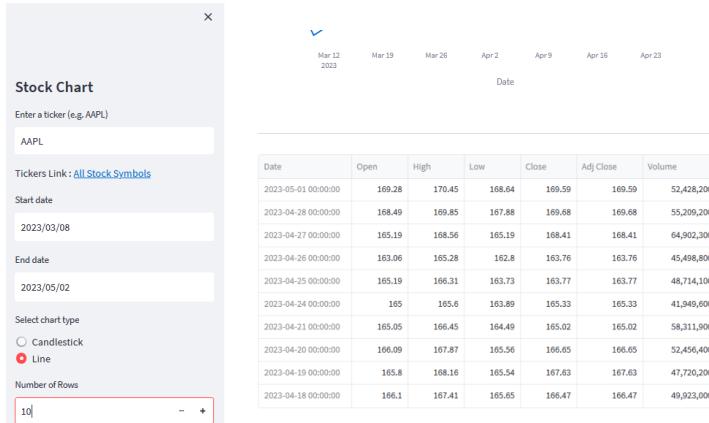
- Chart Type

CandleStick 외에 Line Chart를 선택하면 차트가 변경된다.



- 데이터의 행 갯수 조정

행 개수를 숫자에 따라 입력하면 자동으로 개수가 늘어나도록 조정하였다.



파일위치 : ch05/13_number_date_input.py

```
# -*- coding:utf-8 -*-
import plotly.graph_objects as go
import pandas as pd
import streamlit as st
import yfinance as yf

def main():
    st.sidebar.title("Stock Chart")

    ticker = st.sidebar.text_input("Enter a ticker (e.g. AAPL)",
    value="AAPL")
    st.sidebar.markdown('Tickers Link : [All Stock
Symbols](https://stockanalysis.com/stocks/)')
    start_date = st.sidebar.date_input("Start date",
    value=pd.to_datetime('2023-01-01'))
```

```
end_date = st.sidebar.date_input("End date",
value=pd.to_datetime('today'))

data = yf.download(ticker, start=start_date, end=end_date)
chart_type = st.sidebar.radio("Select chart type", ("Candlestick",
"Line"))

candlestick = go.Candlestick(x=data.index, open=data['Open'],
high=data['High'], low=data['Low'], close=data['Close'])
line = go.Scatter(x=data.index, y=data['Close'], mode='lines',
name='Close')

if chart_type == "Candlestick":
    fig = go.Figure(candlestick)
elif chart_type == "Line":
    fig = go.Figure(line)
else:
    pass

fig.update_layout(title=f"{ticker} Stock {chart_type} Chart",
xaxis_title="Date", yaxis_title="Price")

# Plot the figure
st.plotly_chart(fig)
st.markdown("<hr>", unsafe_allow_html=True)

num_row = st.sidebar.number_input('Number of Rows', min_value=1,
max_value=len(data))
st.dataframe(data[-num_row:].reset_index().sort_index(ascending=False).set_index('Date'),
use_container_width=True)

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

먼저, 본 코드에 필요한 주요 라이브러리를 불러온다. Widget-7에서 st.sidebar에 대해 설명을 자세히 하겠지만, 간단하게 설명하면, st.sidebar 다음에 여러 위젯을 사용하면, sidebar에 해당 위젯이 적용이 된다. 따라서, 여기에서 st.sidebar에 대해서는 크게 신경 쓸 필요 없이 다음에 따라오는 위젯, 즉 input widgets에만 집중하면 된다. 코드가 길기 때문에 코드를 분할해서 설명을 하면 다음과 같다.

```
def main():
    st.sidebar.title("Stock Chart")

    ticker = st.sidebar.text_input("Enter a ticker (e.g. AAPL)",
    value="AAPL")
    st.sidebar.markdown('Tickers Link : [All Stock
Symbols](https://stockanalysis.com/stocks/)')
    start_date = st.sidebar.date_input("Start date",
    value=pd.to_datetime('2023-01-01'))
    end_date = st.sidebar.date_input("End date",
    value=pd.to_datetime('today'))

    data = yf.download(ticker, start=start_date, end=end_date)
```

메모 포함[evan8]: 이거 제거하고, 일괄적으로 볼일지
(indentation제거) 아니면 이대로 할지 고민 필요해요

main() 함수 내에서, 사이드바에 제목을 추가하려면 st.sidebar.title을, 티커 심볼에 대한 사용자 입력을 가져오려면 st.sidebar.text_input을, 주식 데이터의 시작일과 종료일을 가져오려면 st.sidebar.date_input을 사용하여 사이드바를 설정한다. yf.download 함수를 사용하여 지정된 티커 및 날짜 범위에 대한 주식 데이터를 가져온다.

```
chart_type = st.sidebar.radio("Select chart type", ("Candlestick",
"Line"))

candlestick = go.Candlestick(x=data.index, open=data['Open'],
high=data['High'], low=data['Low'], close=data['Close'])
line = go.Scatter(x=data.index, y=data['Close'], mode='lines',
name='Close')
```

```
if chart_type == "Candlestick":  
    fig = go.Figure(candlestick)  
elif chart_type == "Line":  
    fig = go.Figure(line)  
else:  
    pass  
  
fig.update_layout(title=f"{ticker} Stock {chart_type} Chart",  
xaxis_title="Date", yaxis_title="Price")  
  
# Plot the figure  
st.plotly_chart(fig)  
st.markdown("<hr>", unsafe_allow_html=True)
```

st.sidebar.radio를 사용하여 캔들형 차트와 꺾은선형 차트 중에서 차트 유형을 선택할 수 있는 라디오 버튼을 사이드바에 추가한다. 각각 go.Candlestick 및 go.Scatter 메서드를 사용하여 Candlestick 개체와 Line 차트 개체를 만든다. 선택한 차트 유형을 확인하고 적절한 차트 유형으로 go.Figure 객체를 생성한다. fig.update_layout를 사용하여 차트 제목과 축 제목을 설정합니다. 이 때, Chart 제목에는 주식의 ticker 이름과 Chart 이름이 같이 출력되도록 지정하였다. st.plotly_chart를 사용하여 웹 애플리케이션에 차트를 표시하고 st.markdown을 사용하여 차트 뒤에 가로줄을 추가한다.

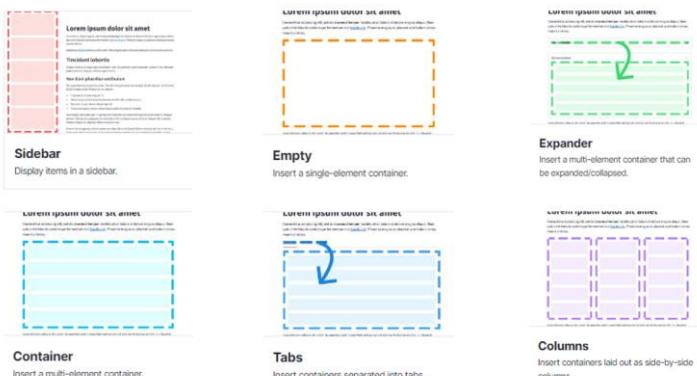
```
num_row = st.sidebar.number_input('Number of Rows', min_value=1,  
max_value=len(data))  
st.dataframe(data[-  
num_row:].reset_index().sort_index(ascending=False).set_index('Date'),  
use_container_width=True)
```

데이터 테이블에 표시할 행 수를 가져오기 위해 st.sidebar.number_input을 추가하고, st.dataframe을 사용하여 주식 데이터의 마지막 num_row 행을 날짜별로 내림차순으로 정렬된 테이블 형식으로 표시한다.

E. Layouts and Containers

대시보드의 레이아웃을 제어하는 것은 미적 관점과 프로그래밍 관점 모두에서 필수적이다. Streamlit은 다양한 옵션을 활용하여 대시보드의 레이아웃을 제어한다. 기본적으로 크게 6개의 메뉴로 구성되어 있다. Sidebar, Columns, Tabs, Expander, Container, 마지막으로 Empty 등이 있다. 여기에서 Sidebar와 Container는 추가적인 설명이 필요하기 때문에, 별도로 다루기로 한다. 나머지 구성 요소는 간략하게만 정리를 한 후, 대시보드 화면에서 어떤식으로 활용되는지 살펴보도록 한다.

- Intro⁸³



`st.sidebar` 위젯은 앱 왼쪽에 사이드바를 만든다. 사이드바는 템색, 입력 필드 및 앱의 기본 콘텐츠 영역에 있을 필요가 없는 기타 컨트롤과 같은 항목을 배치하기에 좋다.

`st.columns` 위젯은 여러 열이 있는 레이아웃을 만든다. 열을 사용하여 콘텐츠를 가로로 정렬할 수 있습니다.

`st.tabs` 위젯은 여러 개의 탭이 있는 레이아웃을 만든다. 탭을 사용하여 여러 페이지의 콘텐츠를 만들 수 있다.

⁸³ <https://docs.streamlit.io/library/api-reference/layout>

`st.expander` 위젯은 확장과 축소가 가능한 컨테이너를 만든다. 사용자가 항상 볼 필요가 없는 콘텐츠를 숨기는 데 사용할 수 있다.

`st.container` 위젯은 컨테이너를 만든다. 컨테이너를 사용하여 관련 콘텐츠를 함께 그룹화할 수 있다.

`st.empty` 위젯은 빈 공간을 만든다. 빈 공간은 앱에 시각적 여유 공간을 추가하는 데 사용할 수 있다.

- **Widget-1 : `st.sidebar`**

사이드바는 그림과 같이 화면의 왼쪽 영역을 일컫는다.



앞서 배운 Input Widgets들은 동일하게 sidebar에도 적용할 수 있다. 좀 더 자세하게 말하면, `st.sidebar` 객체는 사용자가 슬라이더, 버튼, 드롭다운 메뉴 등 다양한 위젯과 상호작용할 수 있는 사이드바를 Streamlit 애플리케이션 내에 생성하는 데 사용된다는 뜻이다. 문법은 크게 두가지로 두가지로 표현이 된다.

```
# Object notation  
st.sidebar.[element_name]  
  
# "with" notation  
with st.sidebar:
```

st.[element_name]

여기에서 element_name이라고 하는 것은 input widgets을 말한다. 다양한 메뉴들이 올 수 있는데, 개념적으로는 어려운 것인 아니지만, 문법적으로는 약간의 혼동이 올 수도 있다. 다음 그림을 확인해본다.



동일하게 st.slider() 메서드를 사용했다. 그런데, 왼쪽 사이드바에는 슬라이더가 두 개가 표현되어 있고, 오른쪽 메인 화면에는 한 개의 slider가 존재한다. 코드로 보면 좀더 명확하게 확인할 수 있다.

파일위치 : ch05/14_sidebar.py

```
# -*- coding:utf-8 -*-
import streamlit as st
def main():
    value1 = st.sidebar.slider('Select a Value object notation', 0, 100)
    st.sidebar.write(value1)

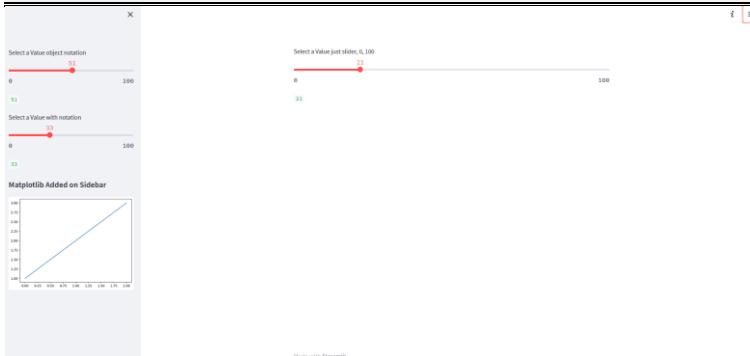
    with st.sidebar:
        value2 = st.slider('Select a Value with notation', 0, 100)
        st.write(value2)
    value3 = st.slider("Select a Value just slider, 0, 100")
    st.write(value3)

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

기억하면 좋은 것이 sidebar에는 시각화 차트도 추가할 수 있다. 다음 코드를 추가하고 결과를 확인한다.

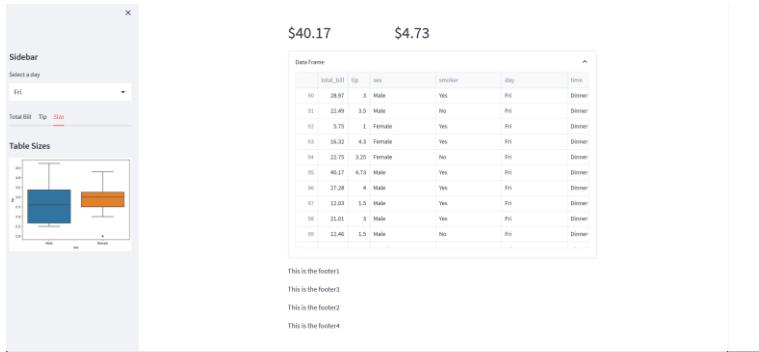
파일위치 : ch05/14_sidebar.py

```
# -*- coding:utf-8 -*-
import streamlit as st
import matplotlib.pyplot as plt
def main():
    ...
    with st.sidebar:
        st.markdown('### Matplotlib Added on Sidebar')
        fig, ax = plt.subplots()
        ax.plot([1, 2, 3])
        st.pyplot(fig)
if __name__ == "__main__":
    main()
```



- Widget-2 : Layouts

본 장에서는 주어진 Layouts and Containers를 모두 활용하여 대시보드를 꾸며보도록 한다. 먼저 전체 화면을 보면 다음과 같다.



메모 포함[evan9]: 이 그림에 각 메서드 표시해야하는데, 그럴려면 이미지 작업을 좀 해야 할 듯요. PPT로 가능할까요?

전체 코드를 확인한다.

파일위치 : ch05/15_layouts.py

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

def main():
    with st.sidebar:
        st.header("Sidebar")
        day = st.selectbox("Select a day", ["Thur", "Fri", "Sat", "Sun"])

    tips = sns.load_dataset("tips")
    filtered_tips = tips[tips["day"] == day]
    top_bill = filtered_tips["total_bill"].max()
```

```
top_tip = filtered_tips["tip"].max()

tab1, tab2, tab3 = st.sidebar.tabs(["Total Bill", "Tip", "Size"])

with tab1:
    fig, ax = plt.subplots()
    st.header("Total Bill Amounts")
    sns.histplot(filtered_tips["total_bill"], kde=False, ax=ax)
    st.pyplot(fig)

with tab2:
    fig, ax = plt.subplots()
    st.header("Tip Amounts")
    sns.histplot(filtered_tips["tip"], kde=False, ax=ax)
    st.pyplot(fig)

with tab3:
    fig, ax = plt.subplots()
    st.header("Table Sizes")
    sns.boxplot(data=filtered_tips, x="sex", y="tip", ax=ax)
    st.pyplot(fig)

container = st.container()
col1, col2 = container.columns([1, 2])

with col1:
    st.metric("Top Bill", f"${top_bill:.2f}")

with col2:
    st.metric("Top Tip", f"${top_tip:.2f}")

with container:
```

```
with st.expander("Data Frame"):
    st.dataframe(filtered_tips)
st.write("This is the footer1")

st.write("This is the footer2")

with container:
    st.write("This is the footer3")

with container:
    st.empty()

st.write("This is the footer4")

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

각 기능별로 나눠서 살펴보도록 한다. 먼저 주요 라이브러리를 불러온다. 여기에서는 matplotlib와 seaborn 두개의 시각화 라이브러리를 가져온다.

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

먼저 sidebar에서 selectbox에서 day를 선택하는 항목을 만들고, 사용자가 특정한 day를 선택하면 day 객체로 저장을 한다. 그 다음 tips 데이터를 불러온 후, 각 day에 맞게 행 추출한 뒤, total_bill과 tip의 최댓값을 구한 뒤, 각각 top_bill과 top_tip으로 각각 객체로 저장한다.

```
def main():
    with st.sidebar:
```

```
st.header("Sidebar")
day = st.selectbox("Select a day", ["Thur", "Fri", "Sat", "Sun"])

tips = sns.load_dataset("tips")
filtered_tips = tips[tips["day"] == day]
top_bill = filtered_tips["total_bill"].max()
top_tip = filtered_tips["tip"].max()
```

이번에는 tabs 메서드를 사용하는데, sidebar에 탭을 사용하기로 한다.⁸⁴ 각 탭에는 탭의 이름을 지정할 수 있다. 그 후에 with절과 함께 각 탭 안에 원하는 시각화 코드를 작성하면 된다. 시각화 코드에 대한 설명은 여기에서는 생략한다.

```
tab1, tab2, tab3 = st.sidebar.tabs(["Total Bill", "Tip", "Size"])

with tab1:
    fig, ax = plt.subplots()
    st.header("Total Bill Amounts")
    sns.histplot(filtered_tips["total_bill"], kde=False, ax=ax)
    st.pyplot(fig)

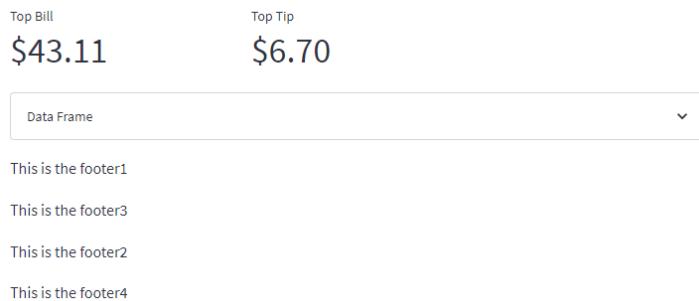
with tab2:
    fig, ax = plt.subplots()
    st.header("Tip Amounts")
    sns.histplot(filtered_tips["tip"], kde=False, ax=ax)
    st.pyplot(fig)

with tab3:
    fig, ax = plt.subplots()
    st.header("Table Sizes")
    sns.boxplot(data=filtered_tips, x="sex", y="tip", ax=ax)
    st.pyplot(fig)
```

⁸⁴ sidebar에는 사실상 모든 객체가 다 올 수 있다고 봐도 무방하다. 다만, UX/UI원리에 대해 배우고 나면, 자주 사용하는 메서드를 기억한 뒤, 곧바로 응용하는 것이 좋다.

- Widget-3 : Containers and Empty

먼저, container가 적용된 화면만 별도로 확인해본다.



언뜻 보면, 기존 메서드와 별다른 차이점을 느끼지 못한다. 그런데, 하나 특이
것이 있는데, This is the footer라인이다. 이 부분을 이해하면 container와
empty가 어떻게 활용되는지 이해할 수 있을 것이다. 우선 공식 문서에 있는
st.container의 개념부터 확인하도록 한다.

Insert a multi-element container.

Inserts an invisible container into your app that can be used to hold multiple elements. This allows you to, for example, insert multiple elements into your app out of order.

To add elements to the returned container, you can use "with" notation (preferred) or just call methods directly on the returned object.

다중 요소 컨테이너를 삽입한다.

여러 요소를 담는 데 사용할 수 있는 보이지 않는 컨테이너를 앱에 삽입한다.
이를 통해 예를 들어 앱에 여러 요소를 순서대로 삽입할 수 있다.

반환된 컨테이너에 요소를 추가하려면 "with" 표기법(기본값)을 사용하거나 반
환된 객체에서 메서드를 직접 호출할 수 있다.

위 설명만 보면, 어떤 내용인지 사실 확인하기가 어렵다. 아래 코드를 통해

container의 역할을 확인해야 한다. 먼저, columns(2)는 화면에서 세로로, 즉 1 x 2 배열 형태로 생성한다. with col1과 with col2는 앞에서 저장한 top_bill과 top_tip을 st.metric()에 추가한다.

```
container = st.container()
col1, col2 = container.columns([1, 2])

with col1:
    st.metric("Top Bill", f"${top_bill:.2f}")

with col2:
    st.metric("Top Tip", f"${top_tip:.2f}")
```

이제 This is the footer 라인의 코드를 확인해본다. 코드에서 This is the footer는 흐름상 순차적으로 작성되어 있다.

```
with container:
    with st.expander("Data Frame"):
        st.dataframe(filtered_tips)
        st.write("This is the footer1")

    st.write("This is the footer2")

    with container:
        st.write("This is the footer3")

    with container:
        st.empty()

    st.write("This is the footer4")
```

그런데 결과는 아래 그림을 보면 숫자 기준 1, 3, 2, 4 순으로 되어 있다.

Top Bill
\$43.11

Top Tip

\$6.70

Data Frame

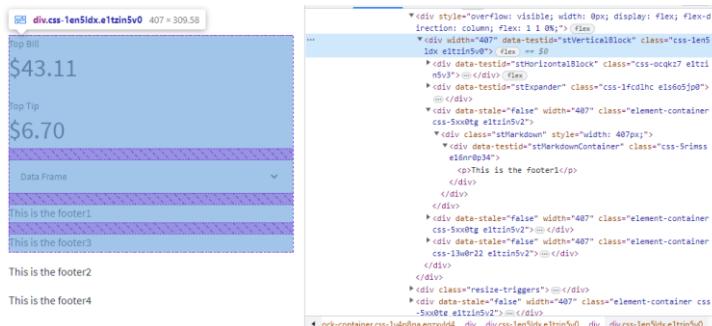
This is the footer1

This is the footer3

This is the footer2

This is the footer4

왜 이런 결과가 나타났을까? This is the footer 1, 3 번은 st.container 로 묶여 있는 반면, This is the footer 2, 4 번은 st.write 가 사용되었다. 즉, 코드의 순서는 1, 2, 3, 4 순이지만, container 로 묶여 있는지 여부에 따라 결과는 달라짐을 확인할 수 있다. 즉, container 의 역할은 일종의 그룹화를 한다고 보면 된다. 실제 개발자 모드에서도 이를 확인할 수 있다.⁸⁵



The screenshot displays a browser window with developer tools open. On the left, the rendered view shows a top section with 'Top Bill' and '\$43.11', followed by a 'Top Tip' section with '\$6.70'. Below these are four footer sections: 'Data Frame', 'This is the footer1', 'This is the footer3', 'This is the footer2', and 'This is the footer4'. The 'Data Frame' section has a dropdown arrow. The right side of the screenshot shows the DOM tree. It starts with a main container 'div.css-1en5idx.e1tzin5v0' with a width of 407px. Inside are two flex items: one for the top section and another for the tip section. The tip section contains a 'stHorizontalBlock' container with a 'stExpander' component. Below these are two 'stMarkdownContainer' components, each containing a 'p' tag with the text 'This is the footer1' or 'This is the footer3'. The bottom section of the DOM tree shows two 'stEmpty' containers, each with a 'resize-trigger' element. The entire structure is wrapped in a 'ock-container' div.

st.empty()는 앞서 그룹화한 container 를 초기화 시키는 코드이다. 만약 코드 중간에 그룹화를 해야 할 필요가 있다면, 다시 container 를 활성화하여 그룹화를 진행하면 된다.

⁸⁵ 개발자 모드는 F12를 누르면 된다.