본 페이지는 편집 시 유의사항을 담고 있습니다.

페이지 수 오류를 방지하기 위해 실제 편집 작업 시에는 지워 주시기 바랍니다.

이 파일은 B5로 제작을 원하는 저자분들께서 사용하시면 됩니다.

편집 과정중 꽉차게 들어가는 이미지, 배경색상이 있을 경우 하얀색 종이가 보이지 않도록 이미지, 색상 크기 반영하셔야 하는 점 알려드립니다.

이파일은 실제 제작시 필요한 사방여백 3mm가 포함된 크기입니다.(188x263mm)

상하좌우 3mm는 실제 제작시 재단되어 반영되지 않으니 참고해주세요!

**어린 왕자**(제목을 적어주세요.)

**어린 왕자(제목을 적어주세요)**

**발 행**｜2023년 00월 00일

**저 자**｜생텍쥐 페리(저자명, 필명을 적어주세요)

**펴낸이**｜한건희

**펴낸곳**｜주식회사 부크크

**출판사등록**｜2014.07.15(제2014-16호)

**주 소**｜서울특별시 금천구 가산디지털1로 119 SK트윈타워 A동 305호

**전 화**｜1670-8316

**이메일**｜info@bookk.co.kr

**ISBN**｜979-11-410-0000-0

**www.bookk.co.kr**

**ⓒ 셍텍쥐 페리 2023**

본 책은 저작자의 지적 재산으로서 무단 전재와 복제를 금합니다.

어

린

왕

자

Evan, Sarah 지음

**CONTENT**

내용

[Chapter 1. 집필 동기 11](#_Toc133532565)

[1. 이 책을 쓰게 된 동기 11](#_Toc133532566)

[A. 강사 입장에서 11](#_Toc133532567)

[B. 수강생 입장에서 11](#_Toc133532568)

[2. 무엇을 배울 수 있는가 11](#_Toc133532569)

[Chapter 2. 개발환경설정 12](#_Toc133532570)

[1. Github 회원가입 12](#_Toc133532571)

[2. Streamlit 회원가입 12](#_Toc133532572)

[3. Git 설치 12](#_Toc133532573)

[4. Python 설치 12](#_Toc133532574)

[5. Visual Studio Code 설치 12](#_Toc133532575)

[6. 가상환경 Virtualenv 설치 12](#_Toc133532576)

[7. requirements.txt 파일 작성 12](#_Toc133532577)

[8. Streamlit, Hello World 12](#_Toc133532578)

[Chapter 3. Python 기본문법 13](#_Toc133532579)

[1. Python Basic Tutorial 13](#_Toc133532580)

[A. Python 프로그래밍 시작 13](#_Toc133532581)

[B. 주석처리 13](#_Toc133532582)

[C. Variables 14](#_Toc133532583)

[D. 사칙연산 17](#_Toc133532584)

[E. 비교 연산자Comparison Operators 18](#_Toc133532585)

[F. 논리 연산자Logical Operators 19](#_Toc133532586)

[G. String 21](#_Toc133532587)

[H. 리스트List 29](#_Toc133532588)

[I. 튜플Tuple 35](#_Toc133532589)

[J. 딕셔너리Dictionary 38](#_Toc133532590)

[K. 조건문 41](#_Toc133532591)

[L. 반복문Iteration 43](#_Toc133532592)

[M. 사용자 정의 함수User-Defined Function 49](#_Toc133532593)

[2. pandas 55](#_Toc133532594)

[A. 데이터 불러오기 57](#_Toc133532595)

[B. 컬럼 선택 57](#_Toc133532596)

[C. 행 선택 58](#_Toc133532597)

[D. .loc와 iloc 59](#_Toc133532598)

[E. describe() 61](#_Toc133532599)

[F. rename() 62](#_Toc133532600)

[G. value\_counts() 64](#_Toc133532601)

[H. isin() 65](#_Toc133532602)

[I. 시계열Time Series 데이터 핸들링 66](#_Toc133532603)

[J. GroupBy 연산 71](#_Toc133532604)

[3. Matplotlib 74](#_Toc133532605)

[A. 선 그래프(Line Plot) 74](#_Toc133532606)

[B. 막대 그래프(Bar Plot) 77](#_Toc133532607)

[C. 박스플롯(Boxplot) 79](#_Toc133532608)

[4. Seaborn 83](#_Toc133532609)

[A. 선 그래프(Line Chart) 83](#_Toc133532610)

[B. 막대 그래프(Bar Plot) 86](#_Toc133532611)

[C. 박스플롯(Boxplot) 92](#_Toc133532612)

[D. Matplotlib와 Seaborn의 관계 96](#_Toc133532613)

[5. Plotly 97](#_Toc133532614)

[A. Graph Objects와 Plotly Express 차이점 97](#_Toc133532615)

[B. 선 그래프(Line Chart) 101](#_Toc133532616)

[C. 막대 그래프(Bar Plot) 104](#_Toc133532617)

[D. 박스플롯(Boxplot) 105](#_Toc133532618)

[6. scikit-learn 107](#_Toc133532619)

[A. 머신러닝Machine Learning 프로세스 108](#_Toc133532620)

[B. 회귀와 분류, 평가지표 109](#_Toc133532621)

[C. 시계열 데이터 예측Time Series Forecasting 110](#_Toc133532622)

[D. ARIMA를 활용한 주가 데이터 예측 114](#_Toc133532623)

[E. LightGBM을 활용한 시계열 데이터 예측 126](#_Toc133532624)

[F. Prophet을 활용한 Airline 데이터 예측 135](#_Toc133532625)

[Chapter 4. 지도 시각화 143](#_Toc133532626)

[1. 배경지식Background Knowledge 145](#_Toc133532627)

[2. GeoPandas 149](#_Toc133532628)

[A. 기본문법 149](#_Toc133532629)

[B. 데이터 매핑 155](#_Toc133532630)

[C. Geometry 데이터 다루기 158](#_Toc133532631)

[Chapter 5. Streamlit 162](#_Toc133532632)

[3. Slider bar 162](#_Toc133532633)

[Chapter 6. 공공데이터 수집 163](#_Toc133532634)

[1. 서울열린데이터광장 163](#_Toc133532635)

[2. 공공데이터포털 163](#_Toc133532636)

[Chapter 7. 부동산 실거래가 대시보드 164](#_Toc133532637)

[1. 데이터 수집 164](#_Toc133532638)

[2. Home 화면 꾸미기 164](#_Toc133532639)

[3. 탐색적 자료분석 화면 꾸미기 164](#_Toc133532640)

[4. 머신러닝 모형 만들기 164](#_Toc133532641)

[5. Prediction 화면 꾸미기 164](#_Toc133532642)

[6. OpenAI 챗봇 구현하기 164](#_Toc133532643)

[Chapter 8. 배포 165](#_Toc133532644)

[1. secrets.toml 165](#_Toc133532645)

[2. 배포 165](#_Toc133532646)

[3. 마무리 165](#_Toc133532647)

[인용 자료 166](#_Toc133532648)

작가의 말 242

레옹 베르트에게 이 책을 바칩니다.

이 책을 어른에게 바친 데 대해 어린이들에게 용서를 바랍니다.

나에게는 그럴 만한 중요한 이유가 있습니다.

그것은 무엇보다도 그 사람은 이 세상에서 나와 가장 친한

친구이기 때문라는 점입니다. 그리고 그는 무엇이든지

알아들을 수 있으며 어린이들을 위한 책까지도 다 이해한다는 점입니다.

세 번째 이유는 그가 프랑스에 살고 있는데

그 곳에서 추위와 굶주림에 떨고 있다는 사실입니다.

그는 위로받아야 할 처지에 있는 것입니다.

그래도 이 모든 이유들이 부족하다면 예전 어린 시절의

그에게 이 책을 바치겠습니다. 어른들도 모두 한때는 어린이였으니까요.

(물론 그것을 기억하는 어른은 별로 없지만.)

그래서 바치는 글을 이렇게 고쳐 씁니다.

'어린 시절의 레옹 베르트에게 이 책을 바칩니다

# Chapter 1. 집필 동기

## 이 책을 쓰게 된 동기

### 강사 입장에서

### 수강생 입장에서

## 무엇을 배울 수 있는가

# Chapter 2. 개발환경설정

## Github 회원가입

### Streamlit 회원가입

### Git 설치

### Python 설치

### Visual Studio Code 설치

### 가상환경 Virtualenv 설치

### requirements.txt 파일 작성

### Streamlit, Hello World

# Chapter 3. Python 기본문법

## Python Basic Tutorial

### Python 프로그래밍 시작

프로그래밍의 첫번째 시작은 “Hello World!”를 출력하는 것에서부터 시작한다. Python에서는 다음과 같이 print()함수를 사용한다.

|  |
| --- |
| print("Hello, Streamlit!")  [결과]  Hello, Streamlit! |

### 주석처리

주석처리는 크게 1줄 주석처리와 다중 주석처리가 존재한다. 다음 코드를 통해 확인해 본다. 아래 코드를 실행하면 주석 처리된 곳은 생략하고, print( ) 함수만 실행되는 것을 확인할 수 있다.

|  |
| --- |
| # 한 줄 주석 처리  """  abc  다중 주석 처리  1234556  """  print("Hello, Streamlit!")  [결과]  Hello, Streamlit! |

### Variables

프로그래밍에서 변수Variables를 저장하는 방법은 크게 4가지(Camel case, Pascal case, Kebab case, Snake case) 방식으로 구분한다. 원 단어가 임의의 학생을 저장하는 변수 my student가 있다고 가정한다. 각자 원하는 방식을 사용하도록 한다. 다음 예시를 통해서 관련 코드를 익히도록 한다.

|  |
| --- |
| myStudent = “sarah” # Camel Case  MyStudent = “sarah” # Pascal Case  my-student = “sarah” # Kebab Case  my\_student = “sarah” # Snake Case |

Python에서 Data Structures를 말할 때, Primitive 자료형과 Non-Primitive 자료형으로 구분한다 (Jaiswal, 2017).[[1]](#footnote-1)

Table 1. Python Data Structures

|  |  |
| --- | --- |
| Primitive | Non-Primitive |
| Integer  Float  String  Boolean | Array  List  Tuple  Dictionary  Set  File |

여기에서 List는 크게 아래와 같이 Linear와 Non-Linear와 구분할 수 있다.

Table . List Data Types[[2]](#footnote-2)

|  |  |
| --- | --- |
| Linear | Non-Linear |
| Stacks  Queues | Graphs  Trees |

#### Integer

Integer는 정수형을 말하며, 4, 5, -1과 같은 숫자를 의미한다. 각자의 나이를 입력하고 출력하는 코드를 작성한다.

|  |
| --- |
| num\_int = 30  print(num\_int)  print(type(num\_int))  [결과]  30  <class 'int'> |

#### Float

Float는 실수형을 말하며, 일반적으로 1.11또는 3.11과 같은 소수를 표현한다. 이번에는 본인의 키를 입력하고 출력하는 코드를 작성한다.

|  |
| --- |
| my\_height = 181.8  print(my\_height)  print(type(my\_height))  [결과]  181.8  <class 'float'> |

#### String

String은 문자열을 말하며, 알파벳, 단어 또는 기타 문자의 모음이다. Python에서는 한쌍의 작은 따옴표(‘’) 또는 큰 따옴표(“”) 안에 일련의 문자를 포함하여 문자열을 생성할 수 있다. 우선 다음 코드를 확인해본다. 임의의 글자를 입력하고 출력하는 코드를 작성한다. 문자열을 다루는 방법에 대해서는 추후에 한번 더 다룰 예정이다.

|  |
| --- |
| my\_name = "evan"  print(my\_name)  print(type(my\_name))  [결과]  evan  <class 'str'> |

#### Boolean

일반적으로 True(참) 또는 False(거짓)을 표현하는 데이터 타입이며, True는 1로, False는 0으로 변환되기도 한다. 해당 데이터 타입은 특히 조건식 및 비교식에 매우 유용하게 활용할 수 있다. 임의의 두개의 숫자를 비교하는 코드를 작성하고, 결괏값을 확인해본다.

|  |
| --- |
| x = 5  y = 3  z = x == y  print(z)  print(type(z))  [결과]  False  <class 'bool'> |

### 사칙연산

일반적으로 형변환이라고 부른다. 임의의 두개를 변수를 만들고 사칙연산을 만들어 보도록 한다. 이 때 주의해서 봐야 할 것은 나눗셈이다. 두 개의 정수를 나눌 때, 정수가 실수형으로 변한다는 것에 주의해야 한다.

|  |
| --- |
| x = 6  y = 3  print(x + y)  print(x - y)  print(x \* y)  print(x / y)  [결과]  9  3  18  2.0 |

이번에는 사칙연산을 통해 나머지, 몫, 제곱을 구하는 연산자를 확인한다.

|  |
| --- |
| x = 6  y = 3  print(x % y)  print(x // y)  print(x \*\* y)  [결과]  0  2  216 |

### 비교 연산자Comparison Operators

비교 연산자를 통해서 알고자 하는 것은 비교의 결괏값이 True인지, False인지 구분하기 위함이다. 우선 비교 연산자는 크게 6개가 존재 한다. 아래 표를 통해서 확인하도록 한다.

Table . Comparison Operators

|  |  |
| --- | --- |
| Operator | Meaning |
| A == B | A와 B가 같다 |
| A != B | A와 B가 같지 않다 |
| A > B | A가 B보다 크다 |
| A < B | A가 B보다 작다 |
| A >= B | A가 B보다 크거나 같다 |
| A <= B | A가 B보다 작거나 같다 |

Python 코드의 예시를 통해 살펴보도록 한다.

|  |
| --- |
| x = 10  y = 11  print(x == y)  print(x != y)  print(x > y)  print(x < y)  print(x >= y)  print(x <= y)  [결과]  False  True  False  True  False  True |

### 논리 연산자Logical Operators

논리 연산자는 주어진 조건식이 True인지 False인지 확인하는데 사용하며, 의사 결정 시에 유용하게 사용된다.

|  |
| --- |
| a = 3  b = 5  print((a > 2) and (b >= 6)  [결과]  False |

위 코드는 Python에서 논리 연산자를 사용하는 예제이다. 논리 연산자는 크게 and, or, not 3개가 존재한다. 논리 연산자의 기본적인 개념을 표로 정리하였다.

Table . Logical Operators[[3]](#footnote-3)

|  |  |  |
| --- | --- | --- |
| Operator | Example | Meaning |
| and | X and Y | X와 Y가 모두 True이면 True |
| or | X or Y | X와 Y중 한 개라도 True이면 True |
| not | not X | X가 False이면 True |

다음은 위 논리 연산자의 모든 결과를 보여주는 표이다.

Table . Truth Table

|  |  |  |  |
| --- | --- | --- | --- |
| X | Y | X and B | A or B |
| True | True | True | True |
| True | False | False | True |
| False | True | False | True |
| False | False | False | False |

위 표가 실제로 Python에서도 구현이 되는지 코드로 확인한다. 먼저 and 연산자를 구현한 코드이며, 결과는 양쪽이 모두 True일 때, True를 반환한다.

|  |
| --- |
| print(True and True)  print(True and False)  print(False and True)  print(False and False)  [결과]  True  False  False  False |

이번에는 or 연산자를 구현한 코드이며, 결과는 양쪽 중 하나만 True여도 True로 결과가 반환이 된다.

|  |
| --- |
| print(True or True)  print(True or False)  print(False or True)  print(False or False)  [결과]  True  True  True  False |

not 연산자를 사용하면 반대로 결괏값이 반환되는 것도 같이 확인한다.

|  |
| --- |
| print(not True)  print(not False)  [결과]  False  True |

### String

앞에서 String에 대해서 간단하게 다뤘지만, 일반적으로 문자열은 보다 자세히 다뤄야 할 정도로 매우 중요하다.

#### String Operators

우선 문자열에는 덧셈 연산자와 곱셈 연산자 사용이 가능하다. 덧셈 연산자는 두개의 문자를 이어서 내보내고, 곱셈 연산자는 숫자만큼 문자열이 반복된다. 코드를 통해 확인해본다. 먼저 덧셈 연산자는 아래와 같이 표현할 수 있다.

|  |
| --- |
| a = "boy"  b = "loves"  c = "girl"  letter = a + " " + b + " " + c + " forever"  print(letter)  [결과]  boy loves girl forever |

이번에는 곱셈 연산자를 확인하는 코드를 작성한다.

|  |
| --- |
| x = "Love "  lyrics = x \* 5  print(lyrics)  [결과]  Love Love Love Love Love |

#### 인덱싱Indexing

인덱싱은 각각의 문자열 안에서 범위를 지정하는 것을 의미한다. 다음 이미지[[4]](#footnote-4)를 확인하면 문자열은 앞쪽에서는 0부터 시작하고, 뒤쪽에서는 -1부터 시작하는 것을 확인할 수 있다. 이 때, 기억해야 하는 것은 이번에 배울 인덱싱은 문자열, List, Tuple, array와 같은 자료형에도 동일하게 사용할 수 있다.

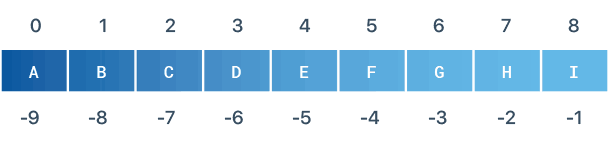


Figure . String Indexing

위 이미지를 토대로 간단하게 Streamlit Dashboard 문자열을 가지고 인덱싱을 진행하도록 한다 (독자도 임의대로 코드를 작성하여 실행하도록 해본다). 임의의 텍스트를 X 변수에 담고, X[인덱스번호]과 같은 형태로 코드를 실행하면 해당 인덱스번호의 글자가 조회된다.

|  |
| --- |
| X = "Streamlit Dashboard"  print(X[0]) # 0번째 글자 가져오기  print(X[5]) # 5번째 글자 가져오기  print(X[-1]) # -1번째 글자 가져오기  print(X[-3]) # -3번째 글자 가져오기  [결과]  S  m  d  a |

그런데, 만약 주어진 텍스트의 인덱스 번호를 초과하면 다음과 같은 에러가 발생이 된다.

|  |
| --- |
| print(X[100]) |

텍스트이(가) 표시된 사진

자동 생성된 설명

위 에러 메시지에서 주의 깊게 살펴봐야 하는 것은 ①『----> 1 print(X[100])』와 ②『**IndexError**: string index out of range』이다. ① 메시지는 어떤 코드에서 에러가 나타났는지를 표시해주는 것이고, ② 메시지는 에러코드의 내용이다. 간단히 요약하면, 1번째 print() 코드에서 **IndexError** 에러가 발생했다고 알려주는 것이다. 즉, X가 담고 있는 텍스트는 인덱스 번호가 100까지는 없기 때문에 조회할 것이 없다는 뜻이기도 하다. X 텍스트가 가지고 있는 인덱스 번호를 재 확인하여 입력하면 에러는 발생이 되지 않을 것이다.

#### 슬라이싱Slicing

인덱싱은 각 문자열의 문자를 한 개씩 추출한다면 슬라이싱은 범위를 지정하여 추출하는 것을 말한다. 즉, 기존 데이터에서 일부의 범위를 지정하여 하위 집합을 추출하는 것을 말한다. 이 슬라이싱의 문법은 일반적으로 [start:end:step]의 일반적인 패턴을 따르게 된다.

* start : 슬라이싱의 첫번째 인덱스이며, 해당 인덱스 번호부터 조회를 시작한다. 만약에, 생략을 하면 인덱스는 0번째부터 시작을 한다.
* end : 슬라이싱의 마지막 인덱스를 말한다. 그런데, 해당 인덱스 번호를 포함되지 않고 그 전까지만 조회가 된다. 만약에, 생략을 하면 해당 문자열의 전체 길이가 기본값으로 설정된다.
* step : 슬라이싱의 각 인덱스 사이에서 건너뛸 인덱스의 수를 말하며, 생략할 경우 기본값은 1이다.

주어진 예제를 확인하도록 한다. 이 때, 해당 인덱스 번호를 바꿔서 진행해서 원하는 대로 결과가 나오는지 확인한다.

|  |
| --- |
| greeting = "Hello Streamlit"  print(greeting[:])  print(greeting[6:])  print(greeting[:6])  print(greeting[3:8])  print(greeting[0:9:2])  [결과]  Hello Streamlit  Streamlit  Hello  lo St  HloSr |

인덱싱과 슬라이싱의 원리는 향후에 나오게 되는 List, Tuple 등 다른 자료형에도 그대로 사용이 되기 때문에, 반드시 기억하도록 한다.

#### 문자열 수정

주어진 문자열을 수정하는 코드를 작성한다. 일반적으로는 다음과 같은 코드 형태로 작성할 수 있다. 주어진 문자열 중 H를 A로 바꾸는 코드를 작성한다. 그런데 다음과 같은 에러가 난다.

|  |
| --- |
| greeting = 'Hello, Streamlit'  greeting[0] = 'A'  print(greeting) |

텍스트이(가) 표시된 사진

자동 생성된 설명

우선, 에러가 나는 이유는 문자열은 불변immutable 객체로 인식되기 때문이다. 불변 객체는 객체 생성 이후 내부의 상태가 변하지 않는 것을 말한다. 그렇다면, 새로운 객체를 만들려면 새로운 변수명을 만드는 방법으로 우회할 수 있다.

|  |
| --- |
| greeting = "Hello Streamlit"  new\_greeting = 'A' + greeting[1:]  print(new\_greeting)  [결과]  Aello, Streamlit |

#### String Methods

String 클래스 내부에는 다양한 메서드Methods들이 존재한다.[[5]](#footnote-5) 여기에서 자주 사용하는 여러 메서드들을 확인하도록 한다.

1. str.upper() : 영어 문자들을 모두 대문자로 변경한다.

|  |
| --- |
| sampleText = "hello, streamlit"  upper\_text = sampleText.upper()  print(upper\_text)  [결과]  HELLO, STREAMLIT |

2. str.upper() : 영어 문자들을 모두 소문자로 변경한다.

|  |
| --- |
| sampleText = "HELLO, STREAMLIT"  lower\_text = sampleText.lower()  print(lower\_text)  [결과]  hello, streamlit |

3. str.capitalize() : 영어 문자의 첫번째 글자만 대문자로 변경한다.

|  |
| --- |
| sampleText = "hello, streamlit"  cap\_text = sampleText.capitalize()  print(cap\_text)  [결과]  Hello, streamlit |

4. str.title() : 문자열 내의 각 단어들의 첫번째 글자만 대문자로 변경한다.

|  |
| --- |
| sampleText = "hello, streamlit"  title\_text = sampleText.title()  print(title\_text)  [결과]  Hello, streamlit |

5. str.strip() : 주어진 문자열의 앞뒤 공백이 있다면 모두 제거한다.

|  |
| --- |
| sampleText = " hello, streamlit "  stripped\_text = sampleText.strip()  print(stripped\_text)  [결과]  hello, streamlit |

6. str.replace(old, new) : 기존(old) 문자열을 새로운(new) 문자열로 바꾼다. 주어진 텍스트 hello에서 welcome으로 변경하는 코드를 작성한다.

|  |
| --- |
| sampleText = "hello, streamlit"  new\_text = sampleText.replace("hello", "welcome")  print(new\_text)  [결과]  welcome, streamlit |

7. str.split(sep=None, maxsplit=-1) : 문자열을 sep 파라미터에 입력한 분리 기준점으로 분리한 후 List[[6]](#footnote-6)로 반환한다.

|  |
| --- |
| sampleText = "hello, streamlit"  words = sampleText.split(sep=",")  print(words)  [결과]  ['hello', ' streamlit'] |

8. str.join(iterable) : List나 Tuple내의 값을 하나의 문자열로 변환한다.

|  |
| --- |
| words = ["hello", "streamlit"]  text = ", ".join(words)  print(text)  [결과]  hello, streamlit |

### 리스트List

Python에 존재하는 자료형으로 시퀀스Sequence 데이터를 다룬다. 시퀀스를 갖는 것은 데이터에 순서가 있다는 뜻이며, 순서가 존재하기 때문에 인덱스와 슬라이싱을 사용할 수 있다. 아래 코드는 주어진 객체가 시퀀스인지 확인하는 코드이다. Sequnece 클래스에 대한 설명은 본 책에서는 생략한다.[[7]](#footnote-7)

|  |
| --- |
| from collections.abc import Sequence  my\_num = 100  my\_list = [1, 2, 3]  my\_string = "hello"  is\_num = isinstance(my\_num, Sequence)  is\_list = isinstance(my\_list, Sequence)  is\_string = isinstance(my\_string, Sequence)  print(is\_num)  print(is\_list)  print(is\_string)  [결과]  False  True  True |

위 결과를 보면 알 수 있듯이 리스트와 문자열은 시퀀스 데이터로 판정을 받았지만, 수치형은 시퀀스 데이터가 아님을 알 수 있다. 그 외 다른 데이터 자료형을 정의하고, 테스트를 해보는 것은 독자에게 맡기도록 한다.

#### 리스트 생성

먼저 리스트를 생성하는 다양한 방법에 대해 살펴보도록 한다. 개발을 하면 종종임의의 빈 리스트Empty List를 생성할 때가 있기 때문에, 관련 코드는 한번씩 숙지하면 좋다. 마지막 변수 e에 저장된 리스트는 리스트 안에 또다른 리스트가 존재하는 것으로 중첩리스트Nested List라고 부르기도 한다.[[8]](#footnote-8)

|  |
| --- |
| a = []  b = list()  c = [2]  d = ['streamlit']  e = [1, 2, ['hello streamlit']]  print(a)  print(b)  print(c)  print(d)  print(e)  [결과]  []  []  [2]  ['streamlit']  [1, 2, ['hello streamlit']] |

#### 인덱싱 & 슬라이싱

이미 문자열에서 인덱싱과 슬라이싱에 관한 개념적인 설명은 문자열 인덱싱 및 슬라이싱에서 진행했기 때문에 여기에서는 간단하게 다루도록 한다.

|  |
| --- |
| a = [1, 2, 3]  print(a[0]) # 첫번째 요소  print(a[1]) # 두번째 요소  print(a[2]) # 세번째 요소  print(a[-1]) # 마지막 요소  [결과]  1  2  3  3 |

중첩리스트를 다루는 방법은 []를 연속해서 사용하는 것이다. 아래와 같이 리스트를 생성하도록 한다. 이 때, be with you만 조회하는 코드를 작성하면 다음과 같이 할 수 있다.

|  |
| --- |
| my\_list = [1, 2, ["I", "want", "to", "be", "with", "you"], 4, 5]  # be with you 조회  my\_list[2][3:]  [결과]  ['be', 'with', 'you'] |

my\_list의 인덱스에 2번째 인덱스를 선택하면 ["I", "want", "to", "be", "with", "you"]만 남게 되고, 그 후에 3번째 인덱스부터 마지막까지 지정하면 원하는 결과를 조회할 수 있다.

#### append()

기존 정의된 리스트에 새로운 값 1개를 추가할 때 사용한다.

|  |
| --- |
| letter = ["evan", "likes"]  letter.append("sarah's smile")  letter  [결과]  ['evan', 'likes', "sarah's smile"] |

#### extend()

append() 메서드는 리스트의 한 개의 값만 추가가 가능하다. 만약 여러 개의 값을 추가하고 싶다면 extend() 메서드를 사용한다.

|  |
| --- |
| letter = ["Do", "You"]  letter.extend(["Like", "Evan", "?"])  letter  [결과]  ['Do', 'You', 'Like', 'Evan', '?'] |

#### insert()

insert() 메서드는 특정 인덱스의 목록에 새로운 값을 추가한다.

|  |
| --- |
| letter = ["Evan", "has", "on", "sarah"]  letter.insert(2, "a crush")  letter  [결과]  ['Evan', 'has', 'a crush', 'on', 'sarah'] |

#### 리스트 값 변경

리스트는 변경가능한Mutable 객체이다. 연산자 “=”를 활용하면 새 값을 할당하여 리스트의 항목을 변경할 수 있다.

|  |
| --- |
| letter = ["Evan", "is", "with", "someone"]  letter[3] = "Sarah"  letter  [결과]  ['Evan', 'has', 'a crush', 'on', 'sarah'] |

#### 리스트 값 삭제

리스트 내의 값을 삭제하는 방법은 remove()를 이용할 수 있다. 이 때, remove() 메서드 안에는 리스트 내의 임의의 값을 입력하여 제거 할 수 있다. 임의의 숫자를 생성한 후, 두 개의 값을 연속해서 제거한다.

|  |
| --- |
| # 임의의 숫자 리스트를 생성한다.  dates = [302, 401, 418, 430, 529]    # 401를 제거한다.  dates.remove(401)  print(dates)  # 529를 제거한다.  dates.remove(529)  print(dates)  [결과]  [302, 418, 430, 529]  [302, 418, 430] |

그 외에도 pop()함수를 사용하거나 del 메서드를 사용하여 리스트 내의 값을 제거할 수 있다.

#### 리스트 값 정렬

리스트 값을 정렬하는 메서드로 sort()가 존재한다. 문자 및 숫자에 각각 적용한 후, 결과를 확인해본다. 먼저 문자는 가나다 또는 알파벳순으로 정렬된다.

|  |
| --- |
| letters = ["sarah", "loves", "evan"]  letters.sort()  letters  [결과]  ['evan', 'loves', 'sarah'] |

숫자는 오름차순으로 정렬된다.

|  |
| --- |
| dates = [329, 322, 317, 302, 217, 201]  dates.sort()  dates  [결과]  [201, 217, 302, 317, 322, 329] |

### 튜플Tuple

Python의 튜플은 리스트와 유사하다. 두 객체의 차이점은 튜플내의 값들이 한번 할당되면 변경할 수 없는 반면, 리스트는 변경 가능하다. 튜플을 생성하는 방법은 (값1, 값2) 형태로 저장한다. 아래 예시를 통해 확인하도록 한다.

|  |
| --- |
| my\_tuple = ("Hi", 1, 3.12)  print(my\_tuple)  my\_nested\_tuple = ("Hi", [1, 2, 3], (3.12, 1.23, 4.21))  print(my\_nested\_tuple)  [결과]  ('Hi', 1, 3.12)  ('Hi', [1, 2, 3], (3.12, 1.23, 4.21)) |

그런데, 소괄호( )가 없이 튜플을 생성할 수도 있다.

|  |
| --- |
| my\_tuple = 1, 2, 3  print(my\_tuple)  [결과]  (1, 2, 3) |

#### 인덱싱 및 슬라이싱

튜플에서도 인덱싱 및 슬라이싱은 가능하다. 인덱싱과 슬라이싱의 기본원리는 문자열, 리스트와 동일하다. 중첩 튜플(Nested Tuple)을 생성하고, 인덱싱과 슬라이싱을 진행한다.

|  |
| --- |
| my\_tuple = (1, 2, ("I", "want", "to", "be", "with", "you"), 4, 5)  # be with you 조회  my\_tuple[2][3:]  [결과]  ('be', 'with', 'you') |

#### 튜플 Methods

리스트에서 제공하는 append(), remove() 메서드 사용은 불가능하다. 그러나, count(), index() 메서드는 사용이 가능하다. 메서드 count()는 튜플 내의 특정 단어가 몇번 출현하는지 빈도를 파악하는 함수이고, index()는 튜플 내의 찾고자 하는 단어가 첫번째 등장하는 인덱스 값을 반환하는 함수이다.

|  |
| --- |
| my\_tuple = ("Love", "Love", "Love", "Hate", "Love", "Love")  print(my\_tuple.count('Love'))  print(my\_tuple.index('Hate'))  [결과]  5  3 |

#### 리스트와 튜플의 차이점[[9]](#footnote-9)

일반적으로 튜플은 리스트와 유사하며, 비슷한 상황에서 사용이 가능하다. 그런데, 아래와 같은 상황에서 약간의 차이점이 존재한다.

* 일반적으로 이기종Heterogeneous(다른) 데이터 유형에는 튜플을 사용하고 동종Homogeneous(유사)데이터 유형에는 리스트를 사용한다.
* 리스트가 수정가능한 객체인 반면 튜플은 불변이기 때문에 튜플을 통해서 반복문을 구성하는 것이 약간 속도가 빠르다고 알려져 있다.
* 불변인 객체 튜플은 딕셔너리 객체의 key값으로 구성이 될 수 있지만, 리스트는 딕셔너리 객체의 key값으로 구성이 될 수 없다.
* 만약, 변경할 필요가 없는 데이터를 가지고 있다면, 이를 만약에 튜플로 구현한다면 데이터가 쓰기-금지write-protected 상태로 유지될 수 있다.

### 딕셔너리Dictionary

딕셔너리는 key-value 값으로 이루어진 데이터로, 숫자 범위로 인덱싱되는 시퀀스와 달리 딕셔너리는 key로 인덱싱되며, key는 모든 불변 유형이며, 문자열과 숫자가 key값으로 사용될 수 있다. 튜플 또한 key값으로 사용할 수 있다. 그런데, 이 때에도 마찬가지로 튜플 내에서도 문자열과, 숫자가 있는 경우에만 가능하면, 다른 객체가 들어있는 경우 key값으로 사용될 수 없다. 리스트와 달리 key값은 슬라이싱, append(), extend()과 같은 리스트 관련 메서드를 사용할 수 없다는 점에 유의한다. 기본적으로 딕셔너리는 {} 형태로 저장된다.

|  |
| --- |
| my\_dic = {'name':'sarah', 'age': 20, 'job': 'Data Analyst',  'shopping':[1,2,3]}  print(my\_dic)  print(my\_dic['name'])  print(my\_dic['age'])  print(my\_dic['shopping'])  [결과]  {'name': 'sarah', 'age': 20, 'job': 'Data Analyst', 'shopping': [1, 2, 3]}  sarah  20  [1, 2, 3] |

기존 정의된 딕셔너리 my\_dic에 새로운 key-value값을 추가하는 방법은 아래와 같이 할 수 있다.

|  |
| --- |
| my\_dic['country'] = 'S. Korea'  print(my\_dic)  [결과]  {'name': 'sarah', 'age': 20, 'job': 'Data Analyst', 'shopping': [1, 2, 3], 'country': 'S. Korea'} |

기존에 정의된 값을 바꾸려면, 기존 key값에 새로운 value값을 입력한다. 만약, shopping key에 정보를 가방으로 바꾸도록 한다.

|  |
| --- |
| my\_dic['shopping'] = '가방'  print(my\_dic)  [결과]  {'name': 'sarah', 'age': 20, 'job': 'Data Analyst', 'shopping': '가방', 'country': 'S. Korea'} |

만약에 특정 value를 삭제하고 싶다면, del을 활용한다. 가방을 사지 않기로 가정하고 key값 shopping을 제거한다.

|  |
| --- |
| del my\_dic['shopping']  print(my\_dic)  [결과]  {'name': 'sarah', 'age': 20, 'job': 'Data Analyst', 'country': 'S. Korea'} |

#### 딕셔너리 메서드

딕셔너리 메서드는 다양하게 존재하지만, 자주 사용하는 것 3가지만 간단하게 확인한다. 먼저 정의된 딕셔너리에서 key값만 조회하는 메서드는 keys() 메서드가 있다.

|  |
| --- |
| my\_dic.keys()  [결과]  dict\_keys(['name', 'age', 'job', 'country']) |

이번에는 value값만 조회하는 메서드 values()를 확인해본다.

|  |
| --- |
| my\_dic.values()  [결과]  dict\_values(['sarah', 20, 'Data Analyst', 'S. Korea']) |

이번에는 key-value값을 동시에 조회하는 메서드 items()를 확인해본다. 이때 주의해야 하는 것은 key-value값은 튜플 형태로 묶여서 나온다.

|  |
| --- |
| my\_dic.items()  [결과]  dict\_items([('name', 'sarah'), ('age', 20), ('job', 'Data Analyst'), ('country', 'S. Korea')]) |

### 조건문

조건문이란, 특정한 조건이 True이면 해당 조건에 부합하는 코드 블록을 실행하는 것을 말한다. 그런데 때로는 여러 조건을 확인해야 하는 경우, 여러 구문을 사용할 수 있다. 먼저 단일 조건인 경우에는 아래와 같이 코드 작성이 가능하다.

|  |
| --- |
| a = 38  b = 31  if a >= b:  print("a가 b보다 크거나 같다")  else:  print("b가 a보다 작다")  [결과]  a가 b보다 크거나 같다 |

만약 다중 조건일 때는 if-elif-else 구문을 사용한다. 다음 문법을 통해 확인해본다.

|  |
| --- |
| a = 31  b = 31  if a > b:  print("a가 b보다 크다")  elif a == b:  print("a가 b와 같다")  else:  print("a가 b보다 작다")  [결과]  a가 b와 같다 |

지금까지 비교를 한 것은 숫자끼리의 비교이다. 그러나, 조건문은 단순히 숫자끼리의 비교 뿐만이 아니라, 문자열 매칭을 조건문으로 사용할 수 있다. 우선 아래 코드를 확인하면, 주어진 문자열에 let이 있는지 여부를 체크한 후, 해당 코드 블록이 실행되는 것을 확인할 수 있다.

|  |
| --- |
| my\_string = "Don't let them in, don't let them see"  if "let" in my\_string:  print("문자열 let이 존재한다.")  else:  print("문자열 let이 존재하지 않는다")  [결과]  문자열 let이 존재한다. |

Python 문법 중 한가지 특징은 조건문을 한 줄로 표현할 수도 있다. 기존에 작성했던 a, b 숫자 비교하는 것을 한 줄로 처리하도록 해본다.

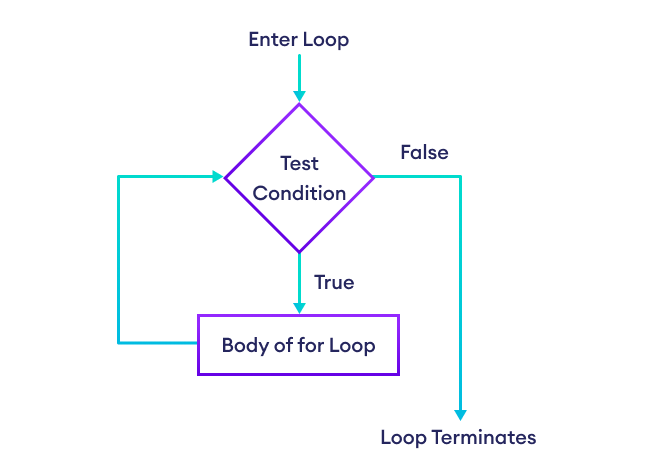
|  |
| --- |
| a = 38  b = 31  result = "a가 b보다 크거나 같다" if a >= b else "b가 a보다 작다"  print(result)  [결과]  a가 b보다 크거나 같다 |

### 반복문Iteration

반복문이란, 특정 코드 블록을 여러 번 반복하는 과정을 말한다. 일반적으로 반복문은 for-loop 구문과, while 구문으로 나누어서 살펴볼 수 있다.

#### for Loop

Python에서 for-loop는 특정 횟수 동안의 코드 블록을 실행하는 데 사용된다. 일반적으로 리스트, 튜플, 문자열 등의 시퀀스를 반복하는데 사용된다.



for-loop 반복문의 기본 코드는 아래와 같이 작성할 수 있다.

|  |
| --- |
| for idx in sequence\_data:  # 코드 |

위 코드에서 idx는 시퀀스 데이터가 가지고 있는 각 값에 순차적으로 접근할 수 있다. for-loop는 시퀀스데이터의 전체 길이만큼 반복된 후 종료된다.

간단하게 아래 코드를 실행해본다.

|  |
| --- |
| programming = ['파이썬', 'HTML', 'CSS', '자바']  for idx in programming:  print(idx)  [결과]  파이썬  HTML  CSS  자바 |

이번에는 range() 메서드를 활용하여 반복문을 실행해본다. Python에서 제공하는 range() 메서드는 인덱스 번호가 0번째부터 시작을 한다. 만약 range(2)를 사용한다면, 인덱스 번호는 0, 1 두개가 출력이 된다. 다음 코드로 확인한다.

|  |
| --- |
| for i in range(2):  print(i)  [결과]  0  1 |

Python에서는 for loop에도 else 구문이 추가될 수 있다. 위 코드에 else 구문을 추가해본다.

|  |
| --- |
| for x in range(2):  print(x)  else:  print("반복문 완전 종료!")  [결과]  0  1  반복문 완전 종료! |

이번에는 enumerate() 메서드를 활용해본다. enumerate() 메서드를 사용할 경우, 시퀀스를 반복하면서 각 항목의 인덱스를 추적할 수 있다.

|  |
| --- |
| my\_tuple = ("Hi", 1, 3.12)  for i, item in enumerate(my\_tuple):  print(i, item)  [결과]  0 Hi  1 1  2 3.12 |

이번에는 zip() 메서드를 활용해본다. 해당 메서드의 가장 큰 장점은 서로 다른 두개의 리스트에서 연산을 수행할 수도 있다.

|  |
| --- |
| total\_sales = [5000, 4500, 300]  prices = [2.5, 3.3, 1.9]  for sales, price in zip(total\_sales, prices):  revenue = sales \* price  print(f'전체 매출액 : {revenue}')  [결과]  전체 매출액 : 12500.0  전체 매출액 : 14850.0  전체 매출액 : 570.0 |

#### List Comprehension

반복문을 한줄로 처리하는 방식이다. List Comprehension은 아래 그림[[10]](#footnote-10)과 같이 크게 3가지 구문으로 구성되어 있다.

차트이(가) 표시된 사진

자동 생성된 설명

주어진 리스트에서 문자열 매칭 코드를 작성해본다. 주어진 이름에서 알파벳 a가 들어간 이름만 나오도록 한다.

|  |
| --- |
| names = ["Evan", "Sarah", "John", "Issac"]  newlist = []  for name in names:  if "a" in name:  newlist.append(name)  print(newlist)  [결과]  ['Evan', 'Sarah', 'Issac'] |

위 복잡한 코드를 한줄로 정의할 수 있다.

|  |
| --- |
| names = ["Evan", "Sarah", "John", "Issac"]  newlist = [name for name in names if "a" in name]  print(newlist)  [결과]  ['Evan', 'Sarah', 'Issac'] |

#### While 문

While 조건문은 특정 조건이 참인 전제에서 코드 블록을 반복적으로 실행할 수 있는 구조이다. While 루프의 구문은 다음과 같이 작성할 수 있다.

|  |
| --- |
| while condition:  # 코드 블록 |

While Loop의 조건은 반복문이 실행되기전에 진행된다. 여기에서 조건이 참이면 코드 블록으로 넘어가고, 코드 블록이 실행된 후, 조건을 재확인한 후, 코드 블록이 실행되는 구조다. 만약, 조건이 참으로 유지되는 한 반복문은 계속 유지가 된다. 다음은 while Loop를 사용하여 1부터 5까지의 숫자를 출력하도록 한다.

|  |
| --- |
| i = 1  while i <= 3:  print(i)  i += 1  [결과]  1  2  3 |

이번에는 간단하게 로그인을 구현하는 예제를 만들어본다. 즉, 문자열을 활용하여 while loop를 만들어본다.

|  |
| --- |
| username = input("사용자 이름을 입력해주세요..!")  password = ''  while password != 'Sarah':  print('정확한 비밀번호가 필요합니다.')  password = input("비밀번호를 입력해주세요..!")  print(f'{username}님 환영합니다')  [결과]  사용자 이름을 입력해주세요..! Evan  정확한 비밀번호가 필요합니다.  비밀번호를 입력해주세요..! a  정확한 비밀번호가 필요합니다.  비밀번호를 입력해주세요..! Sarah  Evan님 환영합니다 |

### 사용자 정의 함수User-Defined Function

사용자 정의 함수는 자주 사용할 것 같은 기능을 분석가가 또는 개발자가 입맛에 재구성을 하는 것을 말한다. 일반적으로 사용자 정의 함수를 만들 때, 기본 함수들을 이용하거나, 동일 코드가 여러 번 반복할 것 같은데, 작성하는 것이 좋다. Python에서는 def 키워드를 제공하고 있으며, 아래와 같은 기본 문법으로 정의할 수 있다. 아래 코드에서 docstring은 소스 코드에 포함된 문서(document)라고 볼 수 있다. 즉, 함수에 대한 설명을 남기는 것이라고 보면 된다.

|  |
| --- |
| def 함수이름(매개변수):  “””docstring”””  # 코드  # 코드  return 반환값 |

이제 간단하게 함수를 구현하고 실행해본다.

|  |
| --- |
| def hello():  """이 함수는 'Hello World!'를 출력한다!"""  print("Hello World!") |

이 함수를 실행하면 Hello World!가 나올 것이다.

|  |
| --- |
| hello()  [결과]  Hello World! |

여기에서 help(hello)를 호출하면 아래와 같이 docstring[[11]](#footnote-11)에 정의된 구문을 확인할 수 있다.

|  |
| --- |
| help(hello)  [결과]  Help on function hello in module \_\_main\_\_:  hello()  이 함수는 'Hello World!'를 출력한다! |

이번에는 사용자 정의 함수에 매개변수를 추가하도록 한다.

|  |
| --- |
| def hello(name):  print("안녕!", name) |

위 정의된 함수에 임의의 이름을 입력한다.

|  |
| --- |
| hello("Sarah")  [결과]  안녕! Sarah |

숫자를 입력해도 일단 출력이 된다.

|  |
| --- |
| hello(123)  [결과]  안녕! 123 |

그런데, 숫자를 name으로 처리하는 적절하지 않기 때문에, string만 받도록 처리하는 함수를 구현한다. 특히 프로젝트를 진행할 때에도 아래와 같이 함수를 작성하는 것을 권장한다.

|  |
| --- |
| def hello(name: str) -> None:  if not isinstance(name, str):  raise TypeError("Name must be a string.")  print(f"안녕, {name}!") |

함수 안에 있는 isinstance 함수는 입력된 name이 str인지 아닌지 체크하는 함수이고, 만약, name이 str이 아니라면, TypeError 오류 메시지가 출력되도록 하는 함수이다. 만약, name이 정상적으로 str이라면, 그 다음 코드로 넘어가도록 설계하는 것이다. 실제로 테스트를 해보도록 한다. 기존 문자가 입력된 것은 특별한 에러 없이 잘 출력이 되지만, 두번째 코드는 아래와 같이 에러가 나타나는 것을 확인 할 수 있다.

|  |
| --- |
| hello("Sarah")  hello(111)  [결과]  안녕, Sarah! |

텍스트이(가) 표시된 사진

자동 생성된 설명

이번에는 가변 매개변수인 args에 대해 살펴보도록 한다. 일반적으로 사용자 정의함수에는 다양한 매개변수를 정의할 수 있다.

|  |
| --- |
| def add(a, b, c):  return a + b + c  print(add(1, 2, 3))  [결과]  6 |

그런데, 아래와 같이 한 개의 매개변수를 지정하지 않으면 에러가 발생한다.

|  |
| --- |
| print(add(1, 2)) |

텍스트이(가) 표시된 사진

자동 생성된 설명

이렇게 가변적인 상황에 대응할 수 있도록 도와주는 것이 args 이다. 함수를 재정의하고, 어떻게 데이터가 입력이 되는지 확인한다.

|  |
| --- |
| def add(\*args):  print(args, type(args))  return args  add(1, 2, 3)  [결과]  (1, 2, 3) <class 'tuple'>  (1, 2, 3) |

들어온 모든 값을 더하기 위해서는 반복문을 사용해야 한다. 함수 내부를 업그레이드 한다.

|  |
| --- |
| def add(\*args):  addition = 0  for number in args:  addition += number  return addition  print(add(1, 2))  print(add(1, 2, 3))  print(add(1, 2, 3, 4))  [결과]  3  6  10 |

그런데, 여전히 함수의 코드가 복잡하다. 좀 더 간결하게 하기 위해서 Python에서 제공하는 기본 함수 sum()을 활용한다.

|  |
| --- |
| def add(\*args):  return sum(args)  print(add(1, 2))  print(add(1, 2, 3))  print(add(1, 2, 3, 4))  [결과]  3  6  10 |

이번에는 \*\*kwargs에 대해 알아본다. 일반적으로 함수에서 정의되지 않은 매개변수를 받을 때 사용되는데, 딕셔너리 형식으로 전달하게 된다. 아래 예시 코드를 통해 확인한다.

|  |
| --- |
| def temp(\*\*kwargs):  for key, value in kwargs.items():  print(key, value)  temp(name="Sarah", age=30, city="Seoul")  [결과]  name Sarah  age 30  city Seoul |

이번에는 개별적인 문자열만 받아서, 하나의 문장으로 완성하는 코드를 작성하는 함수를 만들어본다.

|  |
| --- |
| def whatIsLove(\*\*kwargs):  result = ""  for arg in kwargs.values():  result += arg + ", "  return result  print(whatIsLove(a="Love is patient",  b="love is kind",  c="It does not envy"))  [결과]  Love is patient, love is kind, It does not envy, |

만약, 다른 텍스트를 더 추가하더라도, 계속적으로 문장을 이어갈 수 있다.

## pandas

pandas[[12]](#footnote-12)는 데이터 조작 및 분석 도구를 제공하는 Python 라이브러리이다. 일반적으로 CSV 또는 Excel 파일과 같은 구조화된 데이터를 전처리(Processing)하고 다루는(Handling)데 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 싸이언티스트, 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 |

데이터프레임DataFrame 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.

|  |
| --- |
| 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 나올때도 있기 때문에, 중간에 한번씩 확인하는 것이 좋다.

### 데이터 불러오기

데이터를 불러오는 방법은 pd.read\_csv(각 파일의 경로) 함수를 활용하면 사용할 수 있다. 본 프로젝트와 유사한 부동산 데이터를 불러온다.[[13]](#footnote-13)

|  |
| --- |
| import pandas as pd  df\_boston = pd.read\_csv("./data/boston.csv")  df\_boston |

테이블이(가) 표시된 사진

자동 생성된 설명

### 컬럼 선택

특정한 컬럼을 선택할 때는 List에 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)  [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 |

### 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 |

### .loc와 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 key error가 발생한다. 반면, iloc를 사용해서 행과 열을 선택할 때 매칭되는 index가 없다면 index error가 발생한다.

.loc를 활용하여 존재하지 않는 “CRI”를 입력하면 KeyError가 발생한다.

|  |
| --- |
| df\_boston.loc['CRI'] |

텍스트이(가) 표시된 사진

자동 생성된 설명

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

|  |
| --- |
| df\_boston.iloc[:, 20] |

텍스트이(가) 표시된 사진

자동 생성된 설명

#### 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 target이 24보다 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'> |

### 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 |

### 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() |

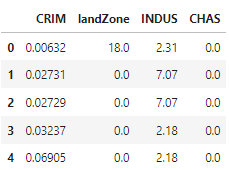


컬럼명을 바꾸는 방법은 아래와 같이 할 수 있다.

|  |
| --- |
| data.reanme(columns={‘Old컬럼명’ : ‘New컬럼명’}) |

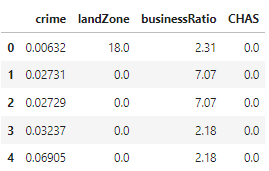
추출된 sample\_df데이터에서 ZN 컬럼명을 landZone으로 변경하는 코드를 작성하도록 한다.

|  |
| --- |
| sample\_df = sample\_df.rename(columns={'ZN': 'landZone'})  sample\_df.head() |



만약, 이번에는 두개의 컬럼을 변경한다면 딕셔너리 안에 연속해서 key-value 형태로 값을 추가하면 된다. 기존 컬럼 CRIM과 INDUS을 각각 crime과 businessRatio로 변경하도록 한다.

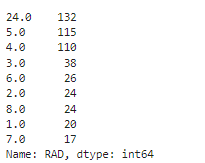
|  |
| --- |
| sample\_df = sample\_df.rename(columns={'CRIM': 'crime',  'INDUS':'businessRatio'})  sample\_df.head() |



### 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\_boston에서 RAD에 해당 함수를 적용해본다. 이 때 결괏값은 DataFrame에서 Series 형태로 반환되는 것에 주의한다.

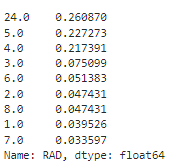
|  |
| --- |
| df\_boston['RAD'].value\_counts() |



|  |
| --- |
| type(df\_boston['RAD'].value\_counts())  [결과]  pandas.core.series.Series |

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

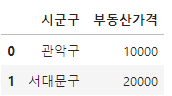
|  |
| --- |
| df\_boston['RAD'].value\_counts(normalize=True) |



### 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 |



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) |



### 시계열Time Series 데이터 핸들링

pandas 라이브러리에서 시계열 데이터는 인덱스가 날짜 혹은 시간인 데이터를 말한다. 만약 가상으로 시계열 자료를 생성하려면 인덱스를 숫자가 아닌 DatatimeIndex 자료형[[14]](#footnote-14)으로 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 확인한다. (데이터셋 참조 : House Property Sales Time Series[[15]](#footnote-15))

|  |
| --- |
| sales = pd.read\_csv('data/raw\_sales.csv')  sales.head(3) |

테이블이(가) 표시된 사진

자동 생성된 설명

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]  <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() |

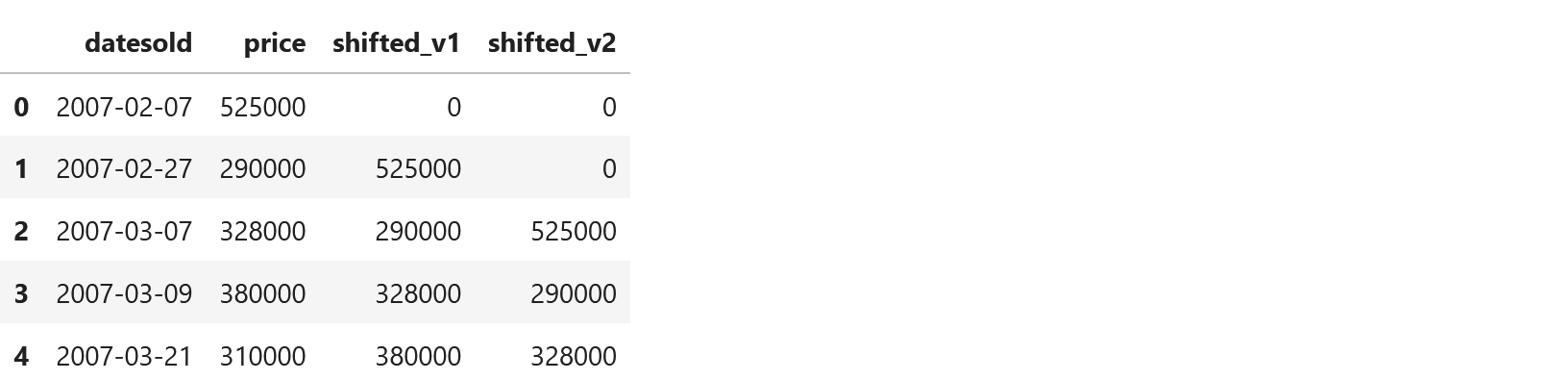
테이블이(가) 표시된 사진

자동 생성된 설명

#### 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() |



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() |

테이블이(가) 표시된 사진

자동 생성된 설명

#### 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() |

테이블이(가) 표시된 사진

자동 생성된 설명

### 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 이미지[[16]](#footnote-16)를 통해서 확인하면 크게 Split-Apply-Combine 3단계로 구성됨을 알 수 있다.

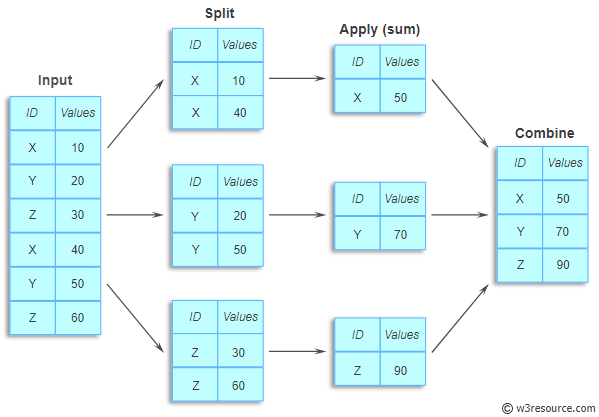
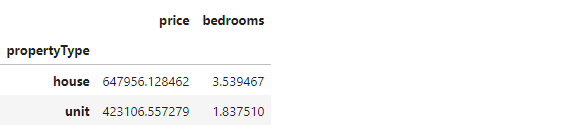


Figure .

Split단계에서는 groupby() 컬럼에서 그룹화할 수 있는 영역을 찾고, 분리를 한다. Apply단계에서는 독립된 그룹별 함수를 적용한다. 위 그림에서는 sum()함수가 사용이 되었지만, 그 외에도 다른 집계 함수 사용이 가능하다. Combine단계에서는 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 |



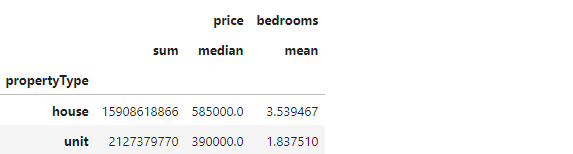
위 코드는 sales 데이터에서 "propertyType" 열의 값에 따라 그룹화하고, 각 그룹에서 "price"와 "bedrooms" 열의 평균 값을 구하는 코드이다. 여기에서 확인해야 하는 코드는 agg\_dict 변수인데, ‘price’ 열에 대해서는 평균값을 계산하고, ‘bedroom’ 열에 대해서도 평균값을 처리하도록 구조화하였고, 이를 grouped에 변수로 저장하여 출력하는 것이다. 출력된 형태는 DataFrame이며 인덱스는 properType으로 지정된 것을 확인할 수 있다. 만약, 인덱스로 정의된 컬럼을 일반 컬럼으로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

|  |
| --- |
| grouped.reset\_index() |



이번에는 컬럼별 다른 집계함수를 적용하도록 한다. 예를 들면, price에는 전체 합계와 중간값, bedrooms에는 평균값만 도출하도록 하는 것이다. 기존 정의된 딕셔너리에서 수정할 것은 value의 값을 리스트로 재정의 하는 것이다.

|  |
| --- |
| agg\_dict = {"price" : ["sum", "median"], "bedrooms" : "mean"}  grouped = sales.groupby("propertyType").agg(agg\_dict)  grouped |

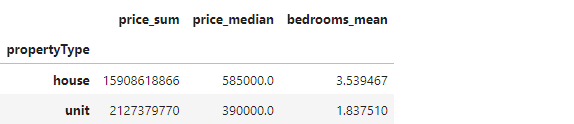


만약 위 컬럼을 평탄화를 하려면 결괏값으로 도출된 데이터프레임 columns에 대한 이해가 필요하다. 우선 아래 코드를 살펴보면 결괏값이 튜플형태로 출력되는 것을 확인할 수 있다.

|  |
| --- |
| 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-string문자열의 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 |



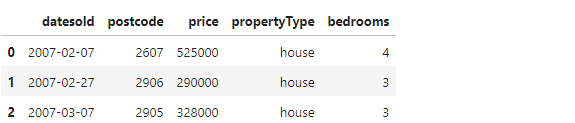
## Matplotlib

Streamlit 라이브러리에서는 matplotlib.pyplot figure가 나타나도록 지원하고 있다.[[17]](#footnote-17) 따라서, 기본적인 시각화를 익히도록 한다. 시각화의 차트 종류는 매우 다양하지만, 여기에서는 본 프로젝트에서 사용한 시각화 차트 위주로 간단하게 준비하였다.

### 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 object가 아닌 datetime64로 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) |



데이터셋의 기본 정보는 아래와 같다. 앞서 pandas와 다르게 여기에서는 datetime64로 적용된 것을 확인할 수 있다.

|  |
| --- |
| print(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 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 |



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() |
| 차트이(가) 표시된 사진  자동 생성된 설명 |

위 코드에 대해서 구체적으로 설명하면 다음과 같다.[[18]](#footnote-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.index는 x축 값, result.values는 y축 값으로 사용된다.
* ax.set\_title('Avg. House Price per Year With Matplotlib', size=16): ax 객체의 set\_title() 메서드를 사용하여 그래프의 제목을 지정하고. size는 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. labelpad는 레이블과 축 사이의 간격을 지정한다.
* plt.savefig('output/matplotlib01.png', dpi=200): 그래프를 이미지 파일로 저장한다. dpi는 저장된 이미지의 해상도를 지정한다
* plt.show(): 그래프를 보여준다.

### 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 추적하는 데 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, :] |



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() |
| 차트이(가) 표시된 사진  자동 생성된 설명 |

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.

### 박스플롯(Boxplot)

박스플롯은 데이터의 분포와 이상치outliers를 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. 이미지를 통해 확인하면 아래와 같다.[[19]](#footnote-19)

차트이(가) 표시된 사진

자동 생성된 설명

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\*IQRInterquartile Range 이상 벗어나면 이상치로 판단한다. 박스플롯은 데이터의 분포와 이상치를 시각화하여 데이터의 중심 경향성과 분산정도를 파악하는 데 도움을 준다. 또한 다른 변수와의 관계를 파악하는데도 유용하게 사용된다. 예를 들어, 그룹별 박스플롯을 그리면 그룹 간 분포의 차이나 중심 경향성의 차이를 한눈에 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) |

테이블이(가) 표시된 사진

자동 생성된 설명

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’[[20]](#footnote-20)는 축의 눈금 레이블을 포맷하는데 사용된다. 특히 과학적 표기법scientific 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 price데이터를 각각 가져온다.
* ax.boxplot([house\_df, unit\_df]) house\_df와 unit\_df 데이터프레임을 사용하여 박스 플롯을 그린다. 특히, ax.boxplot()안에 리스트로 각각의 데이터를 넣어야 한다는 것에 주의한다.
* ax.set\_xticklabels(['House', 'Unit']) x-축 레이블을 "House"와 "Unit"으로 설정한다.

## Seaborn

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

### 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) |



이번에는 2018년 데이터만 조회한다.

|  |
| --- |
| result.tail(12) |



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() |
| 차트이(가) 표시된 사진  자동 생성된 설명 |

### 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]](#footnote-21)

|  |
| --- |
| import seaborn as sns  # 데이터 가져오기  tips = sns.load\_dataset("tips")  tips.head() |



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])  ax[1, 1].set\_title('pointplot()')  plt.tight\_layout()  plt.savefig('output/seaborn02.png', dpi=200)  plt.show() |
| 차트이(가) 표시된 사진  자동 생성된 설명 |

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]](#footnote-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.
* sns.countplot()[[23]](#footnote-23): 이 함수는 범주형 변수의 각 범주에 대한 관측치의 개수를 수직 막대그래프로 보여준다. 각 막대의 높이는 각 범주에서의 관측치의 개수에 해당한다.
* sns.stripplot()[[24]](#footnote-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 추정치 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 x축은 ‘day', y축은 '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 errorbar를 숨길수도 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() |
| 차트이(가) 표시된 사진  자동 생성된 설명 |

본 데이터셋에 적용해서 house와 unit의 errobar를 적용해서 살펴보도록 한다. 이번에는 2007~2010년까지의 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 ~ 2010추출  df = sales.copy()  df = df[df['year'].isin([2007, 2008, 2009, 2010])]  df.head() |

테이블이(가) 표시된 사진

자동 생성된 설명

|  |
| --- |
| df['year'].unique()  [결과]  array([2007, 2008, 2009, 2010], dtype=int64) |

위 가공된 데이터에서, x축은 year, y축은 price, 그리고 hue는 propertyType을 설정하면, year, 그리고 propertyType으로 비교분석이 가능함을 확인할 수 있다. 그래프를 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.

### 박스플롯(Boxplot)

seaborn에서 박스플롯은 일종의 분포를 확인하고자 하는 것이다. seaborn에서 박스플롯을 작성할 때는 x축만 사용할 수도 있고, x축과 y축만 사용할 수도 있다. 우선 price가격의 전체적인 boxplot을 작성하면 다음과 같다. 여기에서 핵심이 되는 코드는 sns.boxplot(x=df["price"], ax = ax)이다. 기존 문법과 다르게, x축에 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 ~ 2010추출  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, x축의 라벨을 리스트로 가져올 수 있다.
* ax.set\_xticklabels(ax.get\_xticks(), rotation=30) : x축의 라벨을 설정한 후, 각 라벨의 값을 rotation에 맞춰서 방향을 변형할 수 있다.

이번에는 x축은 year, y축은 price, hue는 propertyType을 지정하면 새로운 연도별 박스플롯을 그릴 수 있다.

|  |
| --- |
| # 2007 ~ 2010추출  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 700000이상의 데이터가 존재한다. 일반적으로 이런 데이터가 존재하면 전체적인 분포의 상태를 파악하기 어렵기 때문에 해당 데이터를 제거하고 다시 그래프를 작성할 수 있다. 앞선 그래프보다 더 각 연도별 박스플롯의 그래프가 보다 더 자세히 나타나는 것을 확인할 수 있다.

|  |
| --- |
| # 2007 ~ 2010추출  df2 = df[df['price'] <= 7000000]  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() |
|  |

### Matplotlib와 Seaborn의 관계

지금까지 Seaborn관련 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.

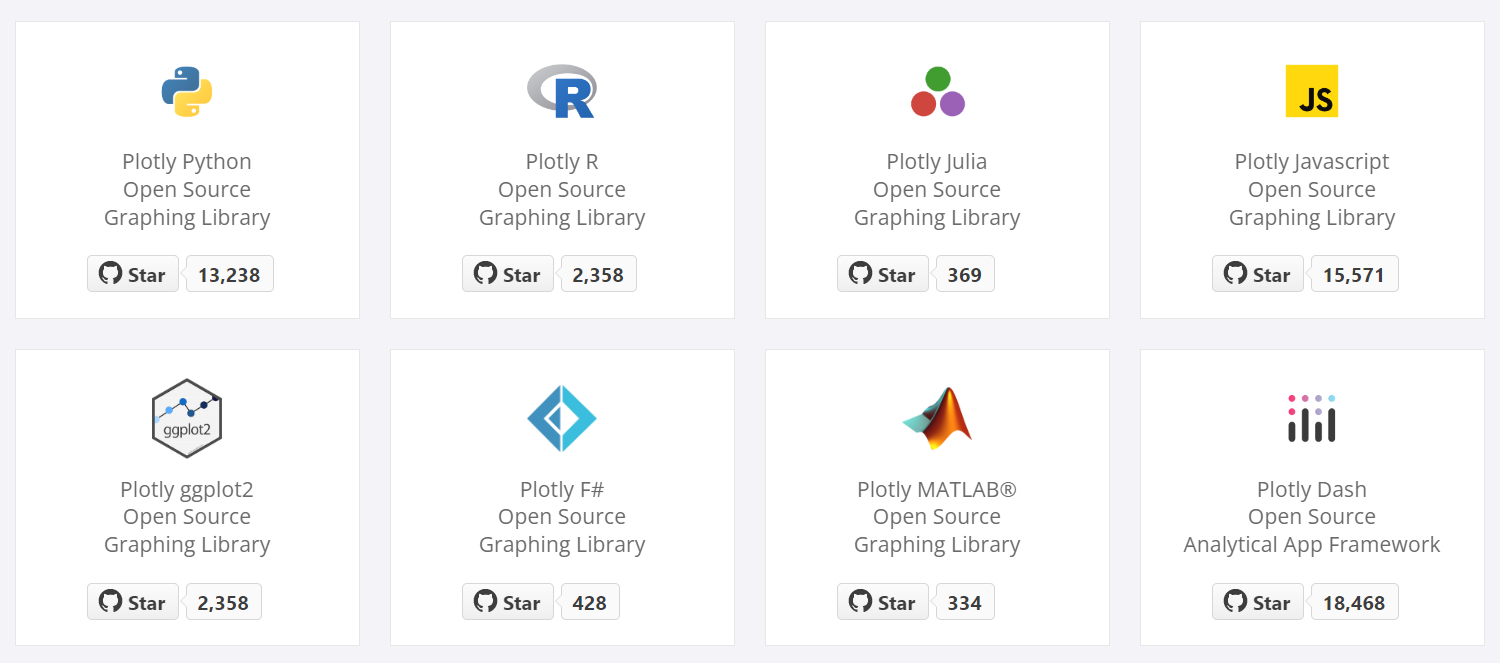
Matplotlib은 2D 및 3D 플롯을 생성하기 위한 광범위한 플롯 기능을 제공하는 저수준Low-Level 라이브러리이다. 축 레이블, 범례, 색상 등을 포함하여 매우 사용자 지정 가능하며 플롯의 모양을 제어할 수 있다. Matplotlib은 Python에서 데이터 시각화에 필수적인 도구이며 데이터 분석가와 과학자가 광범위하게 사용한다.

반면 Seaborn은 Matplotlib 라이브러리 생태계 위에 구축되는 고수준High-Level 데이터 시각화 라이브러리이다. 보다 사용자 친화적인 인터페이스를 제공하고 히트맵Heatmap, 시계열Time Series 및 회귀선Regression 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.

## 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]](#footnote-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. (그림[[26]](#footnote-26) 참조)



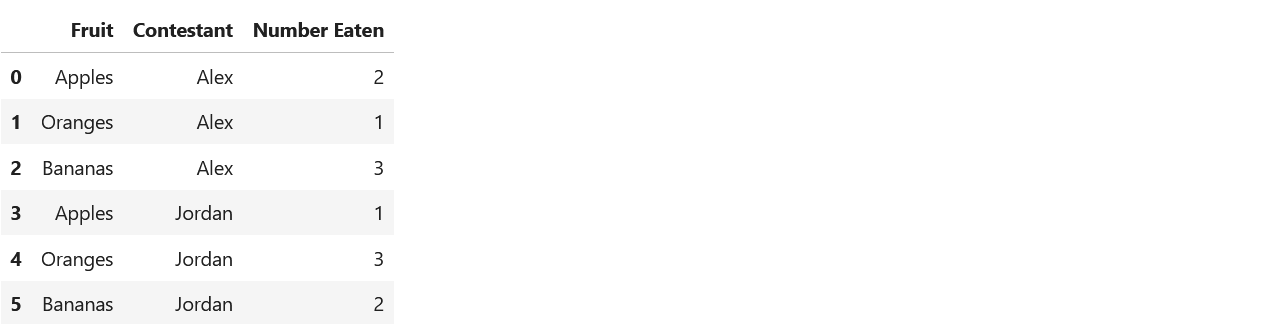
### Graph Objects와 Plotly Express 차이점

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 개발자나 아직 plotly 생태계가 익숙하지 않은 독자는 Plotly Express를 사용하는 것을 권장하고 있다. 공식 홈페이지에서 제공하는 코드를 보면 두 모듈의 코드의 양 차이가 많이 나는 것을 확인할 수 있다.[[27]](#footnote-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 |



이제 Graph Objects를 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}<extra></extra>"% contestant))  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() |
|  |

코드 설명을 한다면, 우선 plotly.graph\_objects 모듈을 go로 정의하여 불러온다. 그 후, go.Figure()를 통해 빈 그래프 객체를 생성한다. for 반복문을 이용하여 df.groupby()을 사용하여 "Contestant"를 그룹으로 묶은 데이터를 contestant, group으로 순서대로 받아온다. 그 다음, go.Bar()를 이용하여 각 그룹에 해당하는 막대 그래프를 생성한다. 이 때, "Fruit"을 x축, "Number Eaten"을 y축으로 설정하고, name에는 해당 그룹의 "Contestant" 이름을 지정한다. 또한 hovertemplate을 이용하여 그래프의 마우스 오버 시 출력되는 정보를 설정한다. 각 그래프를 fig에 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 집필하려고 노력했다.

### 선 그래프(Line Chart)

선 그래프를 작성할 때는 px.line()[[28]](#footnote-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() |
| 차트이(가) 표시된 사진  자동 생성된 설명 |

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

|  |
| --- |
| 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() |



이번에는 plotly 테마를 적용해서 시각화를 작성한다. 기존 코드에서 추가된 것은 plotly.io.templates.default와 color = ‘year’를 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'] |

### 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() |
|  |

### 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.

## scikit-learn[[29]](#footnote-29)

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.

텍스트, 스크린샷, 웹사이트, 웹 페이지이(가) 표시된 사진

자동 생성된 설명

### 머신러닝Machine 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.

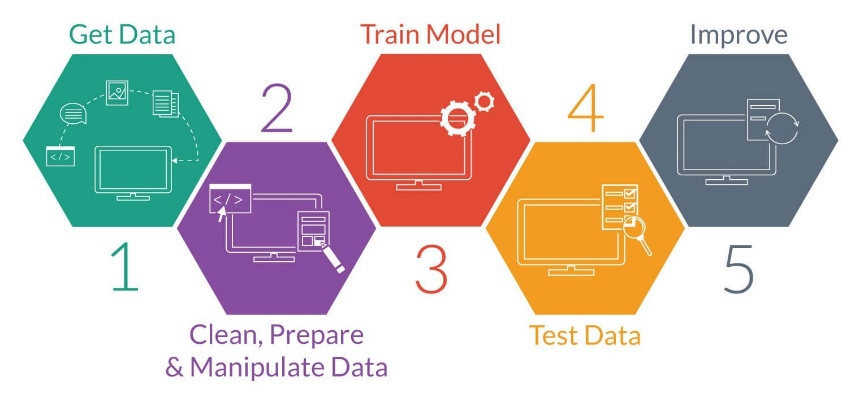
머신 러닝 프로세스의 첫 번째 단계는 데이터를 수집하는 것이다. 이 데이터는 데이터베이스, API 또는 CSV파일, 엑셀 파일 등 다양한 곳에서 가져올 수 있다. 데이터를 수집한 후에는 데이터가 정확하고 오류나 불일치가 없는지 확인하기 위해 데이터를 정리하고 사전 처리해야 한다.

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 때까지 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.



### 회귀와 분류, 평가지표

회귀Regression와 분류Classification는 모두 지도학습Supervised Learning의 일종으로, 독립 변수Independent Variable와 종속 변수Dependent Variable 간의 관계를 모델링하는 데 사용된다.

회귀 모델은 일반적으로 선형 회귀Linear Regression, 다항 회귀Polynomial Regression등의 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(R² : R-Squared) 등의 지표로 평가한다.

반면, 분류 모델은 일반적으로 로지스틱 회귀Logistic Regression등의 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 혼동행렬Confusion Matrix를 생성해야 한다.

텍스트, 스크린샷, 폰트, 도표이(가) 표시된 사진

자동 생성된 설명

성능은 주로 정확도Accuracy, 정밀도Precision, 재현율Recall, F1 점수F1 Score등의 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.

### 시계열 데이터 예측Time 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(LightGradient 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]](#footnote-30) 대부분의 GBDT 도구에서 사용하는 level-wise decision 대신 깊이 제한이 있는 a leaf-wise 알고리즘을 사용한다.

도표, 원, 디자인이(가) 표시된 사진

자동 생성된 설명

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 과적합Overfitting을 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 학습시간을 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.

#### Prophet[[31]](#footnote-31)

텍스트, 스크린샷, 인간의 얼굴, 온라인 광고이(가) 표시된 사진

자동 생성된 설명

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.
* 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.

### ARIMA를 활용한 주가 데이터 예측

주가 데이터 예측 실습을 위해 yfinance[[32]](#footnote-32) 라이브러리를 활용한다. 독자가 원하는 종목을 선택한다.[[33]](#footnote-33)

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

텍스트, 스크린샷, 폰트, 영수증이(가) 표시된 사진

자동 생성된 설명

종가Close만 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() |
|  |

시계열 데이터 분석 할 때는 정상성Stationary을 확인한다.[[34]](#footnote-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.

|  |
| --- |
| 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\n Dickey-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 y는 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 시간 간격을 두고 얼마나 상관성을 가지는지 보여주는 그래프이고, PACF는 시계열 데이터의 각 시점의 값이 그 전 시점의 값과 얼마나 상관성을 가지는지 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 는 plt.figure(figsize=figsize) 에 의해 생성된 새로운 figure 객체이다. figsize는 생성될 figure의 크기를 지정한다.
* spec = fig.add\_gridspec(nrows=2, ncols=2, height\_ratios=layout, width\_ratios=layout) 는 fig에 그리드를 생성한다. 이 그리드는 2x2의 크기이며, height\_ratios 와 width\_ratios는 각각 행과 열의 크기 비율을 나타낸다.
* axs = {}는 spec에 있는 서브플롯들을 저장하기 위한 딕셔너리이다.
* fig.delaxes(axs[0])는 첫 번째 서브플롯을 제거한다.[[35]](#footnote-35)
* axs[1]은 첫 번째 행의 두 번째 열 서브플롯으로, y 시계열 데이터를 그래프로 그리고, 플롯의 제목에는 tickerName과 Dickey-Fuller 테스트의 p-value가 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 lag로 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 lag로 나타낸다. axs[3].set\_ylim(-1.5, 1.5)를 사용하여 y축의 범위를 설정합니다.
* 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 차분Differencing을 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 그래프와 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 윈도우Rolling 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 회귀Autoregressive)과 과거 예측 오류(이동 평균Moving Average)를 모두 사용한다. 이 모델에는 세 가지 매개 변수 p, d, q가 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, p는 미래 값을 예측하는 데 사용되는 시계열의 과거 값의 수이다. 예를 들어 p=2인 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 d는 시계열의 차이 횟수를 나타냅니다. 예를 들어, d=1이면 추세를 제거하기 위해 시계열을 한 번 차분하는 것을 말한다.
* q: 모델의 이동 평균(MA) 부분의 순서로, 모델에 포함될 후행 예측 오류의 수를 나타낸다. 즉, q는 미래 값을 예측하는 데 사용되는 과거 예측 오차의 수다. 예를 들어, q=2인 경우 모델은 가장 최근의 예측 오차 두 개를 사용하여 다음 값을 예측한다.

p, d, q는 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 sktime[[36]](#footnote-36)는 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.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

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.idex의 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() |

텍스트, 스크린샷, 영수증, 폰트이(가) 표시된 사진

자동 생성된 설명

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/>

### LightGBM을 활용한 시계열 데이터 예측

이번에는 Nile[[37]](#footnote-37) 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") |

각 라이브러리들에 대한 설명은 아래와 같다.[[38]](#footnote-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 앙상블 모델을 만들 때 사용된다.
* sktime.forecasting.trend
  + 시계열 데이터의 추세(trend)를 예측하기 위한 클래스 및 함수들을 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\_length는 이전 데이터 포인트의 수, strategy는 예측에 사용되는 방법을 나타낸다. Recursive Strategy는 이전 예측값을 사용하여 다음 값을 예측하는 방법이다.[[39]](#footnote-39)
* forecaster : 생성된 Time Series Forecasting 모델을 나타낸다.

|  |
| --- |
| regressor = lgb.LGBMRegressor()  forecaster = make\_reduction(regressor,window\_length=5,strategy="recursive")  forecaster |

텍스트, 폰트, 화이트, 대수학이(가) 표시된 사진

자동 생성된 설명

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 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, +1은 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. fh는 예측할 미래 시점들을 나타냅니다.
* y\_pred[:5] : 생성된 예측값 중 첫 5개를 출력합니다.

예측 결괏값을 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

### Prophet을 활용한 Airline 데이터 예측

Box-Jenkins Airline 데이터셋 실습을 위해 관련 데이터셋을 불러온다.[[40]](#footnote-40)

|  |
| --- |
| from prophet import Prophet  from sktime.datasets import load\_airline  airline\_df = load\_airline().to\_timestamp(freq="M").reset\_index()  airline\_df |

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

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') |
|  |

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 라이브러리는 컬럼명을 ds와 y로 지정해줘야 한다.[[41]](#footnote-41) 따라서 아래와 같이 데이터로 변형한다.

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

텍스트, 스크린샷, 영수증, 폰트이(가) 표시된 사진

자동 생성된 설명

이제 모형을 만들과 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() |

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

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 분석Geospatial Analysis은 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 가용성 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, ArcGIS[[42]](#footnote-42), QGIS[[43]](#footnote-43)와 같은 도구들이 사용된다. Python에서는 관련 라이브러리가 다수 존재한다. 간단하게 정리하면 아래와 같다.

* GeoPandas[[44]](#footnote-44) : Pandas 데이터프레임을 기반으로 하는 지리공간 데이터 작업을 위한 Python 라이브러리이며. 공간 작업과 분석을 수행할 수 있는 사용하기 쉬운 인터페이스를 제공한다. 본 프로젝트에서 활용한다.
* Shapely[[45]](#footnote-45) : 점, 선, 다각형과 같은 2D 기하 도형 개체로 작업하고 조작하기 위한 Python 라이브러리.
* Fiona[[46]](#footnote-46) : Shapefile과 GeoJSON과 같은 지리공간 데이터 형식을 읽고 쓰기 위한 라이브러리.
* PyProj[[47]](#footnote-47) : 지리 좌표계와 투영 좌표계간 변환 위한 Python 라이브러리
* GDAL/OGR[[48]](#footnote-48) : Raster 및 Vector 데이터를 포함한 지리공간 데이터 형식으로 작업할 수 있는 강력한 라이브러리이다. 명령줄 인터페이스를 제공하며 GIS 소프트웨어에서 널리 사용된다.
* Cartopy[[49]](#footnote-49) : 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.
* Rasterio[[50]](#footnote-50) : 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.
* PySAL[[51]](#footnote-51) : 지리공간 벡터 데이터에 중점을 둔, Geospatial Data Science 플랫폼 라이브러리로, 일반적으로 공간 클러스터 이상값 탐지, 공간 회귀, 통계 모델링, 공간 계량 경졔학, 탐색적 시공간 데이터 분석에 활용되고 있다.
* Geopy[[52]](#footnote-52) : Geocoding과 역 Geocoding을 위한 라이브러리로 주소의 지리적 좌표를 찾거나 그 반대로 찾는데 활용된다.

## 배경지식Background 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]](#footnote-53)

텍스트, 스크린샷, 지도, 도표이(가) 표시된 사진

자동 생성된 설명

먼저, Raster 데이터는 이미지를 셀 또는 픽셀 그리드로 표현하는 것이며, 각 셀에는 색상 등 특정 속성을 나타내는 값이 포함되어 있다. 특히, 고도Elevation나 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 점points, 선lines, 다각형polygons으로 특징을 나타낸다. 모든 점은 x, y 좌표Coordinate로 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 표현할 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

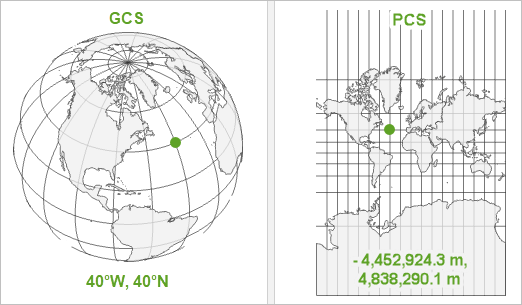
Shapefile은 지리 정보 시스템(GIS) 소프트웨어에서 널리 사용되는 지리 공간 벡터 데이터 형식을 의미한다. Shapefile은 기본 이름은 같지만 확장자가 다른 세 가지 파일(.shp, .shx, .dbf)로 구성되어 있다. 먼저 .shp 파일에는 점, 선, 다각형과 같은 벡터 피처의 geometry가 포함되어 있다. .shx 파일은 GIS 소프트웨어가 .shp 파일에 빠르게 액세스할 수 있도록 하는 인덱스 파일이다. .dbf 파일에는 각 벡터 피처와 관련된 속성 데이터가 포함되어 있다. 요약하면, .shp파일과 .shx파일은 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 데이터에 알려준다.

아래 이미지를 통해서 GCS와 PCS를 쉽게 비교하는 방법은 아래와 같다.[[54]](#footnote-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 홍수 모델링, 항공 및 측량과 같은 애플리케이션에 중요하다. 아래 이미지를 통해 확인한다.[[55]](#footnote-55)

스크린샷이(가) 표시된 사진

자동 생성된 설명

* 국내 지도에서 사용하는 좌표계는 Bessel 1841(EPSG : 4004) : 한국과 일본에 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

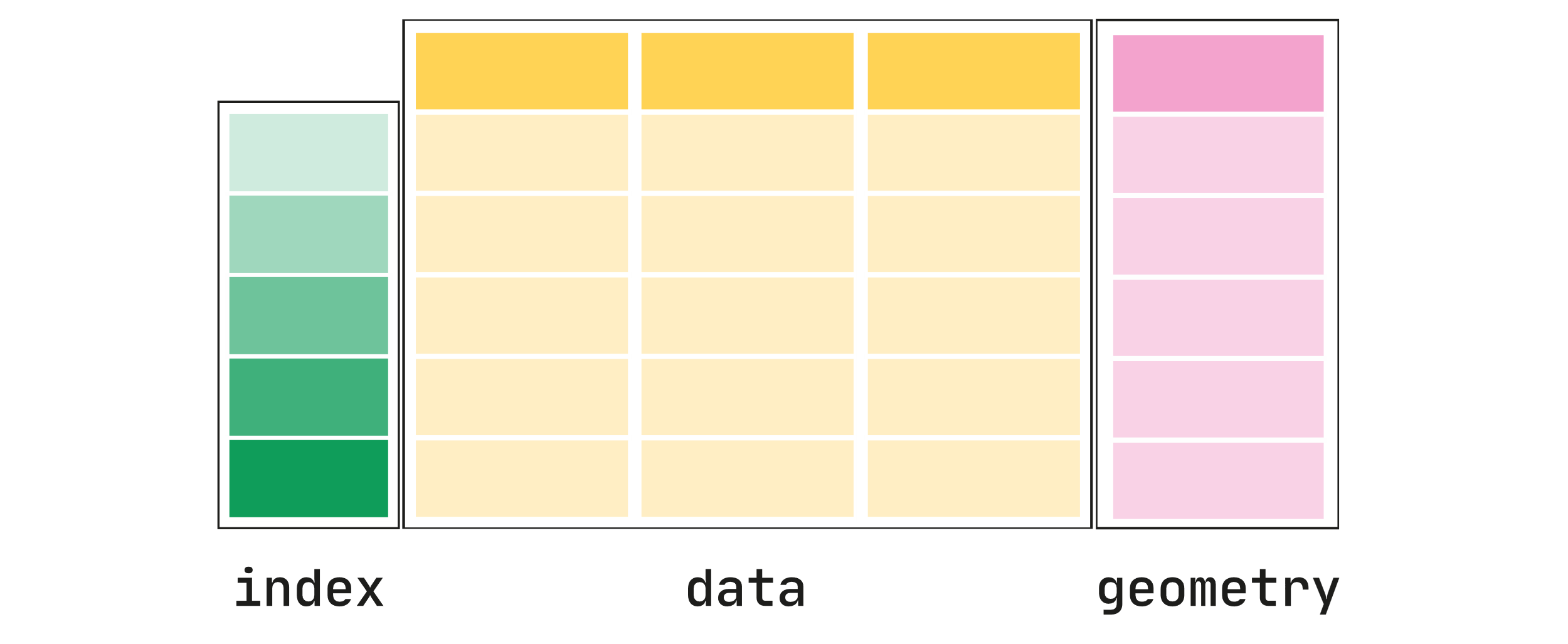
Georeferencing은 Vector나 Raster에 좌표를 할당하여 지구 표면의 모델에 투영하는 프로세스이다. 이 프로세스에는 거리 교차로나 랜드마크와 같이 이미지나 지도에서 알려진 위치를 식별하고 현실 세계의 해당 지리적 좌표와 일치시키는 작업이 포함된다. 이를 통해 이미지 또는 지도를 위성 이미지나 벡터 레이어와 같은 다른 공간 데이터와 정렬할 수 있다. Georeferencing은 일반적으로 과거 지도, 항공사진 또는 GPS가 사용되기 이전의 기타 데이터에 사용된다. Python의 Georeferencing 프로세스는 일반적으로 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 매핑, 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, geoferencing과 geocoding에 모두 사용이 가능하다. Georeferencing된 Raster 데이터를 다양한 형식으로 읽고 조작할 수 있다. 비록 기능은 제한적이지만, geocoding도 geopandas를 통해 수행할 수 있다.

## 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]](#footnote-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() |

텍스트, 스크린샷, 폰트, 화이트이(가) 표시된 사진

자동 생성된 설명

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.

텍스트, 지도, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

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) |

텍스트, 스크린샷, 폰트, 문서이(가) 표시된 사진

자동 생성된 설명

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) |

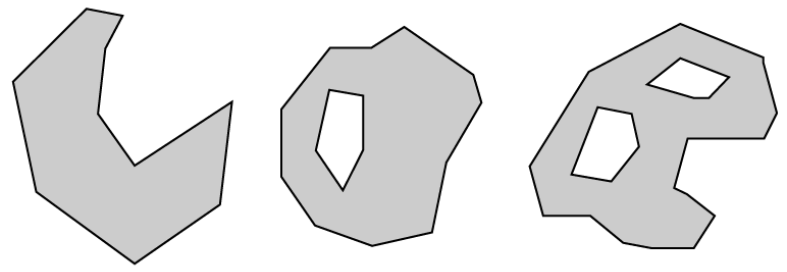
컬럼의 종류는 다음과 같이 출력이 가능하다.

|  |
| --- |
| 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) |

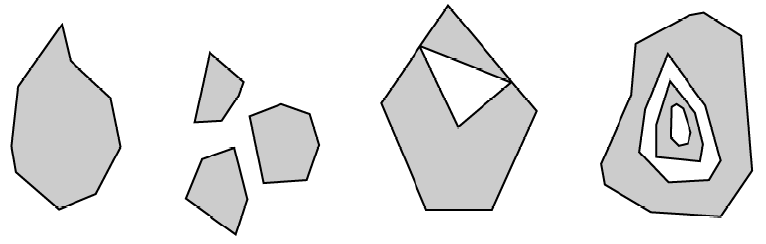
속성 테이블의 처음 5개의 항목이 Polygon과 MultiPolygon으로 구성되어 있다는 것을 보여준다.

|  |
| --- |
| states.geom\_type.head()  [결과]  0 Polygon  1 Polygon  2 Polygon  3 MultiPolygon  4 Polygon  dtype: object |

기본적인 정의는 약간 다르다. 먼저 Polygon은 다각형은 1개의 외부 경계와 0개 이상의 내부 경계로 정의된 평면형 지표Surface이다. 각 내부 경계는 다각형의 구멍Hole을 정의한다. Polygon의 샘플 이미지는 아래와 같다. GIS에서 다각형은 정렬된 x,y 좌표 집합으로 정의되는 2차원 또는 3차원 도형이다. 다각형은 하나의 연속된 영역을 나타내는 닫힌 도형이다. 다각형의 예로는 도시 블록, 호수, 들판 등이 있다.



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



Polygon과 MultiPolygon의 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]](#footnote-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.

|  |
| --- |
| 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, geojson파일을 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) |

텍스트, 스크린샷, 폰트, 문서이(가) 표시된 사진

자동 생성된 설명

### B. 데이터 매핑

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() |

텍스트, 스크린샷, 폰트, 화이트이(가) 표시된 사진

자동 생성된 설명

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() |

텍스트, 스크린샷, 영수증, 폰트이(가) 표시된 사진

자동 생성된 설명

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 key값의 이름이 동일하지는 않다. 따라서, 한쪽에서 맞춰서 컬럼명을 바꿔줘야 한다.

|  |
| --- |
| world\_df = world.merge(data, on = 'name')  world\_df.head() |

텍스트, 폰트, 스크린샷, 영수증이(가) 표시된 사진

자동 생성된 설명

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() |
|  |

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 지정할 때, ‘left’, ‘right’, ‘top’, ‘bottom’ 등을 지정할 수 있다. shring키워드 인수는 색상표가 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\_type은 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) 등의 속성을 가지고 있다. 또한, 한 국가의 영토 등을 여러 개의 점을 이은 Polygon으로 표현할 수 있다. 이 때 데이터 타입은 shapely.geometry.polygon.Polygon으로 확인된다. 만약, Polygon에 대해 이해하려면, Shapely Docs를 확인해야 한다.[[58]](#footnote-58)

|  |
| --- |
| type(world.geometry[1])  [결과]  shapely.geometry.polygon.Polygon |

간단하게 그래프를 그려보면 다음과 같다.

|  |
| --- |
| world.geometry[1] |



위 geometry를 출력하면 Polygon 데이터 값을 뽑을 수 있다.

|  |
| --- |
| print(world.geometry[1]) |

텍스트, 흑백, 폰트, 화이트이(가) 표시된 사진

자동 생성된 설명

이번에는 world 데이터에서 한반도를 출력하도록 한다. 먼저 world 데이터에서 name컬럼의 South Korea, North Korea를 isin() 메서드를 활용하여 필터링 한 후, geometry 컬럼만 GeoSeries 객체로 저장한다. GeoSeries에 squeeze 메서드를 적용하여 단순한 Shapely LineString 객체로 변환한다. (단일 LineString 객체가 포함된 경우). 이는 boundary가 입력에 따라 GeoSeries 또는 GeoDataFrame을 반환할 수 있기 때문이다. squeeze는 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() |

텍스트, 스크린샷, 폰트, 화이트이(가) 표시된 사진

자동 생성된 설명

cities 객체에서 서울과 평양만을 추출한다.

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



위 정보를 이용하여 그래프를 출력한다. kor\_geom.plot() 내부의 color 매개변수는 Polygon 채우기의 색을 흰색으로 설정하고, edgecolor는 Polygon 가장자리의 색을 검정색으로 설정한다. 다음으로, 이름 열이 '서울' 또는 '평양'과 일치하는 행만 포함하도록 cities GeoDataFrame을 필터링한다. 그런 다음 plot 메서드를 사용하여 동일한 축에 GeoDataFrame를. marker 매개 변수는 마커의 모양을 원으로 설정하고, color는 색상을 빨간색으로 설정하며, markersize는 마커의 크기를 설정한다. 축에서 set\_axis\_off 메서드를 호출하여 x축, y축을 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

## Slider bar

# Chapter 6. 공공데이터 수집

## 서울열린데이터광장

## 공공데이터포털

# Chapter 7. 부동산 실거래가 대시보드

## 데이터 수집

## Home 화면 꾸미기

## 탐색적 자료분석 화면 꾸미기

## 머신러닝 모형 만들기

## Prediction 화면 꾸미기

## OpenAI 챗봇 구현하기

# Chapter 8. 배포

## secrets.toml

## 배포

## 마무리

# 인용 자료

Jaiswal, S. (2017, Dec). *Python Data Structures Tutorial*. Retrieved from DataCamp: https://www.datacamp.com/tutorial/data-structures-python

1. Jaiswal, S. (2017, Dec). *Python Data Structures Tutorial*. Retrieved from DataCamp: https://www.datacamp.com/tutorial/data-structures-python [↑](#footnote-ref-1)
2. Python 코딩테스트에서는 Stacks, Queues, Graphs, Trees와 같은 개념을 알아야 풀 수 있는 문제들이 있어서, 해당 문법들을 공부할 필요가 있다. 본 책에서는 다루지 않는다. [↑](#footnote-ref-2)
3. 아래 표에서 X와 Y는 모두 조건식을 의미하며, 조건식의 결과가 True 또는 False인 상태를 의미한다. [↑](#footnote-ref-3)
4. https://www.learnbyexample.org/wp-content/uploads/python/String-Indexing.png [↑](#footnote-ref-4)
5. https://docs.python.org/3/library/stdtypes.html#string-methods [↑](#footnote-ref-5)
6. List와 Tuple, Dictionary와 같은 기초 문법은 다음 페이지부터 순차적으로 다룬다. [↑](#footnote-ref-6)
7. 더 읽어보기 : https://docs.python.org/3/library/collections.abc.html [↑](#footnote-ref-7)
8. 일반적으로 리스트에는 파이썬에서 존재하는 다양한 값을 저장할 수 있다. 숫자, 문자열 뿐만 아니라, 아직 배우지 않은 데이터프레임 객체, 함수, 클래스 등도 넣을 수 있어서, 활용범위가 매우 무궁무진하다. [↑](#footnote-ref-8)
9. https://www.programiz.com/python-programming/tuple [↑](#footnote-ref-9)
10. https://scaler.com/topics/images/list-comprehenstion.webp [↑](#footnote-ref-10)
11. Docstring에 관한 자세한 설명은 <https://peps.python.org/pep-0257/> 참고한다. [↑](#footnote-ref-11)
12. https://pandas.pydata.org/ [↑](#footnote-ref-12)
13. Boston 데이터셋의 정보는 <http://lib.stat.cmu.edu/datasets/boston> 에서 확인한다. 본 장에서는 독자가 해당 컬럼들의 정보를 이해했다는 가정에서 기술하였다. [↑](#footnote-ref-13)
14. https://pandas.pydata.org/docs/reference/api/pandas.DatetimeIndex.html [↑](#footnote-ref-14)
15. https://www.kaggle.com/datasets/htagholdings/property-sales?resource=download [↑](#footnote-ref-15)
16. https://www.w3resource.com/python-exercises/pandas/groupby/index.php [↑](#footnote-ref-16)
17. https://docs.streamlit.io/library/api-reference/charts/st.pyplot [↑](#footnote-ref-17)
18. https://delftswa.gitbooks.io/desosa-2017/content/matplotlib/chapter.html [↑](#footnote-ref-18)
19. https://itwiki.kr/w/박스\_플롯#/media/파일:Boxplot2.png [↑](#footnote-ref-19)
20. https://matplotlib.org/stable/api/ticker\_api.html#matplotlib.ticker.ScalarFormatter [↑](#footnote-ref-20)
21. tips 데이터셋 설명: https://rdrr.io/cran/reshape2/man/tips.html [↑](#footnote-ref-21)
22. sns.barplot() : https://seaborn.pydata.org/generated/seaborn.barplot.html [↑](#footnote-ref-22)
23. sns.countplot() : https://seaborn.pydata.org/generated/seaborn.countplot.html [↑](#footnote-ref-23)
24. sns.stripplot(): https://seaborn.pydata.org/generated/seaborn.stripplot.html [↑](#footnote-ref-24)
25. https://plotly.com/about-us/ [↑](#footnote-ref-25)
26. https://plotly.com/graphing-libraries/ [↑](#footnote-ref-26)
27. https://plotly.com/python/graph-objects/ [↑](#footnote-ref-27)
28. 그 외 px.line()에서 사용가능한 Parameters는 공식문서에서 참조한다. https://plotly.com/python-api-reference/generated/plotly.express.line.html [↑](#footnote-ref-28)
29. https://scikit-learn.org/stable/ [↑](#footnote-ref-29)
30. Liang, W.; Luo, S.; Zhao, G.; Wu, H. Predicting hard rock pillar stability using GBDT, XGBoost, and LightGBM algorithms. Mathematics 2020, 8, 765. [↑](#footnote-ref-30)
31. https://facebook.github.io/prophet/ [↑](#footnote-ref-31)
32. https://github.com/ranaroussi/yfinance [↑](#footnote-ref-32)
33. 여담이지만, 필자는 테슬라로 적지 않은 수익을 거둔적이 있다. [↑](#footnote-ref-33)
34. 시계열 분석에서 정상성에 관한 설명을 구체적으로 확인한기 원한다면, <https://otexts.com/fpp3/stationarity.html> 페이지에서 확인하도록 한다. [↑](#footnote-ref-34)
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. [↑](#footnote-ref-35)
36. http://www.sktime.net/en/latest/ [↑](#footnote-ref-36)
37. https://www.statsmodels.org/dev/datasets/generated/nile.html [↑](#footnote-ref-37)
38. 앞에서 배운 pandas, matplotlib, seaborn 라이브러리에 대한 설명은 제외한다. [↑](#footnote-ref-38)
39. https://sktime-backup.readthedocs.io/en/v0.17.1/api\_reference/auto\_generated/sktime.forecasting.compose.make\_reduction.html [↑](#footnote-ref-39)
40. https://rdrr.io/r/datasets/AirPassengers.html [↑](#footnote-ref-40)
41. https://facebook.github.io/prophet/docs/quick\_start.html [↑](#footnote-ref-41)
42. https://www.arcgis.com/index.html [↑](#footnote-ref-42)
43. https://www.qgis.org/ko/site/ [↑](#footnote-ref-43)
44. https://geopandas.org/en/stable/ [↑](#footnote-ref-44)
45. https://shapely.readthedocs.io/en/stable/ [↑](#footnote-ref-45)
46. https://fiona.readthedocs.io/en/latest/index.html [↑](#footnote-ref-46)
47. https://pyproj4.github.io/pyproj/stable/ [↑](#footnote-ref-47)
48. https://gdal.org/ [↑](#footnote-ref-48)
49. https://scitools.org.uk/cartopy/docs/latest/ [↑](#footnote-ref-49)
50. https://rasterio.readthedocs.io/en/stable/ [↑](#footnote-ref-50)
51. https://pysal.org/ [↑](#footnote-ref-51)
52. https://github.com/geopy/geopy [↑](#footnote-ref-52)
53. https://storage.googleapis.com/lds-media/images/366022803\_1280x720.width-1200.jpg [↑](#footnote-ref-53)
54. https://www.esri.com/arcgis-blog/wp-content/uploads/2022/02/grid2.png [↑](#footnote-ref-54)
55. https://www.esri.com/arcgis-blog/wp-content/uploads/2018/01/VCS1-300x149.png [↑](#footnote-ref-55)
56. https://geopandas.org/en/stable/\_images/dataframe.svg [↑](#footnote-ref-56)
57. 자주 사용하는 코드에 관해서는 <https://spatialreference.org/> 에서 확인한다. [↑](#footnote-ref-57)
58. https://shapely.readthedocs.io/en/stable/manual.html#polygons [↑](#footnote-ref-58)