통계분석A 1강

2022.02.08 (화) 14:30 ~ 16:30

통계에서 코드를 배워야 하는 이유

현대의 데이터 기반 작업들은 대부분, 대용량의 데이터를 다룸

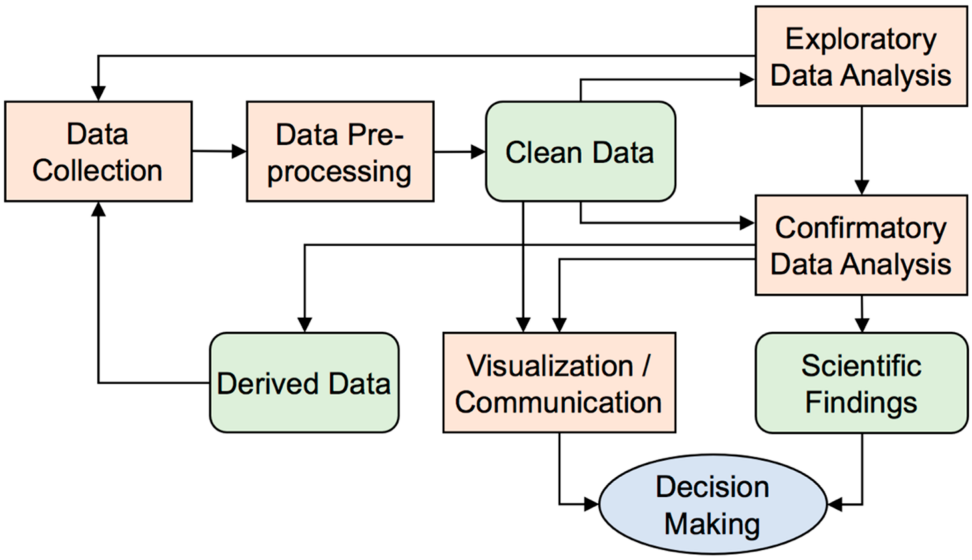
- 통계 분석, 데이터 사이언스, 데이터 엔지니어링, 머신러닝, 딥러닝, 자연어처리 기타 등등
- 코드 혹은 프로그래밍은 반복 작업을 통해 대용량 데이터의 처리를 가능하게 함
- 기본적인 코드를 짜는 능력(=코딩)은 통계의 기본

→ 통계 관련 코딩의 기본기를 배워보자

통계 5W 1H

💡 **Why? 데이터에 대한 통찰을 얻고, 통찰을 기반으로 의사결정을 하기 위해**

- Who - 수강생 여러분이
- What - 각자의 전문 분야의 데이터를
- Where - 컴퓨터가 있는 곳 어디서든
- When - 각자가 필요할 때에
- **How - 파이썬과 기타 툴 들을 활용하여 데이터 통계를 다룰 수 있는 기본 기술을 습득한다**



https://www.mdpi.com/2220-9964/6/11/368/html

강의 진행 방법

- python 사용
- 이론이나 배경 설명 + 간단한 코드 설명
- IDE (Integrated Development Environment) (= 개발 툴)
  - Google Colab 사용
    - 복잡한 로컬 설정없이 브라우저 상에서 python 코드 작성 및 실행 가능
  - 간단한 세팅 및 라이브러리 import 후 사용 안내 예정
- 목적
  - 초심자 - 통계 처리가 생각보다 쉽다(?)를 경험
  - 숙련자 - 실습 + 현장의 실제 데이터 혹은 실무 포인트
- 문의 사항

    - 강의 중 질문
        - 코드 중 XXX가 안돼요! 형식의 질문을 제외한 모든 질문
    - 이메일 질문
        - chwhint@gmail.com
        - 강의와 관련된 모든 것
        - 질문시 메일 제목을 [통계 분석 특강] 으로 시작하도록 작성 부탁드립니다.
- **당부 말씀**
    - 파이썬은 띄워쓰기(indent)가 매우 중요합니다.
        - indent 단위: 공백 4칸 or 탭 1번 (전자를 권고)
        - 보통은 enter(return)을 입력할 경우 IDE에서 자동으로 해줍니다.
    - 직접 손으로 코딩 + 분석 해보시길 바랍니다

Warming up

Hans Rosling: 한스 로슬링이 이제껏 보지 못했던 최고의 통계를 보여준다.

이 같은 데이터를 본 적이 없을 것이다. 드라마틱한 이야기 전개와 스포츠 캐스터 같은 열의있는 발표를 통해, 통계 전문가 한스 로슬링이 소위 말하는 개발 도상국에 관한 통계를 완전 해부한다.

https://www.ted.com/talks/hans_rosling_the_best_stats_you_ve_ever_seen?language=ko

Hans Rosling(1948 ~ 2017)

- 스웨덴 출신 의사, 보건 통계학자
- **FACTFULLNESS**
  “생각보다 인류는 잘 살아 왔고, 우리는 발전하는 세계에 살고 있다.”

  https://gapminder.org/

데이터는 오픈되어 있다

- 공공 데이터 포털
    - 우리나라 공공기관에서 제공하는 데이터.
    - 공공 데이터 포털에 없을 경우, 데이터 요청도 가능
- Kaggle

    - google 에서 운영. 각종 기계학습, 딥러닝 문제와 함께 computing resource를 제공
- Awesome Public Datasets
- Or google it!

→ **본인이 필요한 것을 찾아서 해보자!**

파이썬 복습

- reference: 점프 투 파이썬

조건문

- 기본 구조

```
If <조건문>:
    <수행할 문장1>
    <수행할 문장2>
    ...
elif <조건문>:
    <수행할 문장1>
    <수행할 문장2>
    ...
elif <조건문>:
    <수행할 문장1>
    <수행할 문장2>
    ...
...
else:
   <수행할 문장1>
   <수행할 문장2>
   ...
```

- 예제 코드

```
pocket = ['paper', 'cellphone']
card = True
if 'money' in pocket:
     print("택시를 타고가라")
elif card:
     print("택시를 타고가라")
else:
     print("걸어가라")

# (결과)
# 택시를 타고가라
```

반복문

while문

- 기본 구조

```
while <조건문>:
    <수행할 문장1>
    <수행할 문장2>
    <수행할 문장3>
    ...
```

- 예제 코드

```
treeHit = 0
while treeHit < 10:
    treeHit = treeHit +1
    print("나무를 %d번 찍었습니다." % treeHit)
    if treeHit == 10:
        print("나무 넘어갑니다.")

# (결과)
# 나무를 1번 찍었습니다.
# 나무를 2번 찍었습니다.
# 나무를 3번 찍었습니다.
# 나무를 4번 찍었습니다.
# 나무를 5번 찍었습니다.
# 나무를 6번 찍었습니다.
# 나무를 7번 찍었습니다.
# 나무를 8번 찍었습니다.
# 나무를 9번 찍었습니다.
# 나무를 10번 찍었습니다.
# 나무 넘어갑니다.
```

- while 문 강제 탈출

  Loop 내에서 `break` 사용

```
# 커피 자판기가 동작하는 방식
coffee = 10
while True:
    money = int(input("돈을 넣어 주세요: "))
    if money == 300:
        print("커피를 줍니다.")
        coffee = coffee -1
    elif money > 300:
        print("거스름돈 %d를 주고 커피를 줍니다." % (money -300))
        coffee = coffee -1
    else:
        print("돈을 다시 돌려주고 커피를 주지 않습니다.")
        print("남은 커피의 양은 %d개 입니다." % coffee)
    if coffee == 0:
        print("커피가 다 떨어졌습니다. 판매를 중지 합니다.")
        break
```

- while 문의 처음으로 이동

  `continue` 사용

```
# 홀수의 출력
a = 0
while a < 10:
    a = a + 1
    if a % 2 == 0: continue # 짝수 판별시
    print(a)

# (결과)
# 1
# 3
# 5
# 7
# 9
```

for 문

- 기본구조

```
for 변수 in 리스트(또는 튜플, 문자열):
    수행할 문장1
    수행할 문장2
    ...
```

- 예제 코드

```
test_list = ['one', 'two', 'three']
for i in test_list:
    print(i)

# (결과)
# one
# two
# three
```

- 다양한 for 문의 활용

```
# 합격 여부 판단
marks = [90, 25, 67, 45, 80]

number = 0
for mark in marks:
    number = number +1
    if mark >= 60:
        print("%d번 학생은 합격입니다." % number)
    else:
        print("%d번 학생은 불합격입니다." % number)

# (결과)
```

```
# 1번 학생은 합격입니다.
# 2번 학생은 불합격입니다.
# 3번 학생은 합격입니다.
# 4번 학생은 불합격입니다.
# 5번 학생은 합격입니다.
```

- for 문과 range의 활용

```
# 구구단
# range(2, 10) : 2~9까지의 숫자를 iteration 할수 있게 해줌
for i in range(2,10):        # ①번 for문
    for j in range(1, 10):   # ②번 for문
        print(i*j, end=" ")
    print('') # 줄바꿈

# (결과)
# 2 4 6 8 10 12 14 16 18
# 3 6 9 12 15 18 21 24 27
# 4 8 12 16 20 24 28 32 36
# 5 10 15 20 25 30 35 40 45
# 6 12 18 24 30 36 42 48 54
# 7 14 21 28 35 42 49 56 63
# 8 16 24 32 40 48 56 64 72
# 9 18 27 36 45 54 63 72 81
```

함수

- 반복되는 코드 → 코드의 재사용성을 높이기 위해 코드 사용
- 함수 정의와 호출을 통해 사용(0~n개의 입력변수, 0~n개의 결과 사용 가능)
- 종류
  - 내장 함수 : 이미 정의되어 있는 함수. 필요 시 적절한 함수 호출하여 사용
  - 사용자 정의 함수 : 필요에 의해 사용자가 정의
- 함수의 매개 변수는 해당 함수 내에서만 유효
  - 함수의 시작과 함께 생성되어 함수 종료 시 소멸
- 함수의 정의와 호출

```
# 함수의 정의
def do_nothing():
  pass
do_nothing() # 괄호 주의!

#(결과)
#
```

```
# 매개변수 없이 단순 동작
def do_something():
  print("야호")

do_something()


# 매개변수 없이 함수값 반환
def do_something2():
  return '메롱'

result = do_something2()
print(result)

result2 = do_something2()
print(result2)

# (결과)
# 야호
# 메롱
# 메롱
```

  - 사람이 모니터에서 값을 보는 것과, 프로그램 입장에서 반환 값을 받는 것은 다르다.
  - 프로그램 입장에서는 유효한 반환값을 확인해야 반환 값이 있는 것.

- 인수와 매개변수

```
# 인수와 매개변수
def echo(anything):
  return anything + " " + anything
# 매개변수(함수 내부) = anything
# 인수(외부의 input) = ABCD

echo("ABCD")

# (결과)
# ABCD ABCD
```

- 매개변수의 유효범위

```
# 매개변수의 유효 범위

def echo(anything):
  print(anything) # 유효한 매개변수 (함수 내)
  return anything + ' ' + anything
echo("ABCD")
print(anything) # 유효하지 않은 매개변수(함수 밖)

# ABCD : test1 만 실행됨
# ---------------------------------------------------------------------------
# NameError                                 Traceback (most recent call last)
# <ipython-input-19-1d7e1d8f4e88> in <module>()
#       7 # 인수(외부의 input) = ABCD
#       8 echo("ABCD")
```

```
# ----> 9 print(anything)

# NameError: name 'anything' is not defined
```

- 위치 인수 / 키워드 인수
  - 인수명을 명시하지 않음
  - 매개변수 순서대로 값이 할당됨

```
# 위치 인수

def get_purchase_advice(price, budget, loan):
  "가격, 예산, 대출을 통해 물건을 구입시 조언을 출력하는 함수"
  # docstring : 함수 정의에 붙이는 문서

  statement = None
  if price > budget + loan :
    statement = "Give it up."

  elif (price <= budget + loan) and (price > budget):
    statement = "Buy it with a loan."

  elif price <= budget and price >= 0:
    statement = "Buy it without a loan"

  else:
    statement = "Something wrong"

  print(f"price:{price}, budget:{budget}, loan:{loan}")

  return statement


# 인수 명시 없이 입력 = 위치 인수 사용 = 입력 변수를 매개변수 순서대로 할당
result1 = get_purchase_advice(500, 200, 300)
print(result1)
print()

# 키워드 인수 입력
result2 = get_purchase_advice(price=500, budget=200, loan=300)
print(result2)
print()

# 입력되는 키워드 인수 순서를 매개변수 순서와 다르게 입력
result3 = get_purchase_advice(price=500, loan=300, budget=200)
print(result3)
print()

# result 1,2,3 은 동일한 결과
```

- 기본 매개변수 값을 할당할 때

```
# 기본 매개변수 값 지정하기

def get_purchase_advice2(price, budget, loan=100): #loan=100
```

```
  "가격, 예산, 대출을 통해 물건을 구입할 수 있는지 여부를 확인하는 함수2"

  statement = None
  if price > budget + loan :
    statement = "Give it up."

  elif (price <= budget + loan) and (price > budget):
    statement = "Buy it with a loan."

  elif price <= budget and price >= 0:
    statement = "Buy it without a loan"

  else:
    statement = "Something wrong"

  print(f"price:{price}, budget:{budget}, loan:{loan}")

  return statement

# 인수가 입력되지 않으면 매개변수의 기본값을 사용
result1 = get_purchase_advice2(500, 200)
print(result1)
print()

# 기본값이 지정된 매개변수에 인수가 입력되면 입력된 인수를 사용
result2 = get_purchase_advice2(500, 200, 300)
print(result2)
print()
```

- docstring 확인시

```
# docstring 확인
print(get_purchase_advice.__doc__)
print(get_purchase_advice2.__doc__)
```

- 위치 인수의 개수를 모를때
  - 기본 구조
    - * 을 사용
    - 관례적으로 `args` 라는 매개변수를 사용

```
def 함수이름(*매개변수):
    <수행할 문장>
    ...
```

  - 예제 코드

```
def add_many(*args):
    result = 0
    for i in args:
        result = result + i
```

```
    return result


result1 = add_many(1,2,3)
print(result1)

result2 = add_many(1,2,3,4,5,6,7,8,9,10)
print(result2)

# (결과)
# 입력 인수의 개수에 관계없이 사용가능
# 6
# 55
```

- 키워드 인수의 개수를 모를 때
  - 기본 구조
    - ** 을 사용
    - 역시 관례적으로 `kwargs` 라는 매개변수 이름을 사용
  - 예제 코드

```
def print_kwargs(**kwargs):
    print(kwargs)

print_kwargs(a=1)
print_kwargs(name='foo', age=3)

# (결과)
# {'a': 1}
# {'age': 3, 'name': 'foo'}
```

클래스

클래스의 정의

- 변수와 함수를 묶은 객체

클래스의 쉽게 이해하기

- 대표적인 클래스 문제 : 쌀집 계산기
- (조건) 계산기는 이전 계산 값을 어딘가에 기억하고 있어야 한다.
- 계산기의 구현

```
# 계산기
result = 0

def add(num):
    global result # 함수 밖에서도 결과를 기억할수 있도록 전역 변수를 사용
```

```
    result += num
    return result

print(add(3))
print(add(4))

# (결과)
# 3
# 7
```

- 두 대의 계산기가 필요하다면?
  - Add 하나만으로 결과 값 유지가 어려움
  - 두 개의 함수를 구현하는 것이 필요함

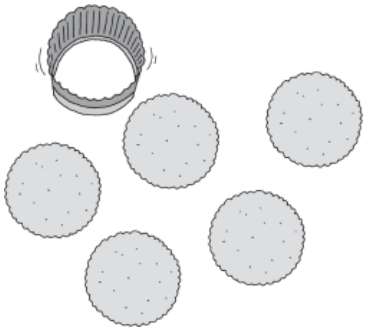
```
# 두 대의 계산기
result1 = 0
result2 = 0

def add1(num):
    global result1
    result1 += num
    return result1

def add2(num):
    global result2
    result2 += num
    return result2

print(add1(3))
print(add1(4))
print(add2(3))
print(add2(7))

# (결과)
# 3
# 7
# 3
# 10
```

- 구현은 가능하지만 매우 비효율적 → 클래스를 이용해 구현!

```
# 계산기의 클래스를 이용한 구현
class Calculator:
    def __init__(self):
        self.result = 0

    def add(self, num):
        self.result += num
        return self.result

cal1 = Calculator() # 인스턴스 할당 = 객체 생성
cal2 = Calculator()

print(cal1.add(3))
```

```
print(cal1.add(4))
print(cal2.add(3))
print(cal2.add(7))

# (실행)
# 3
# 7
# 3
# 10
```

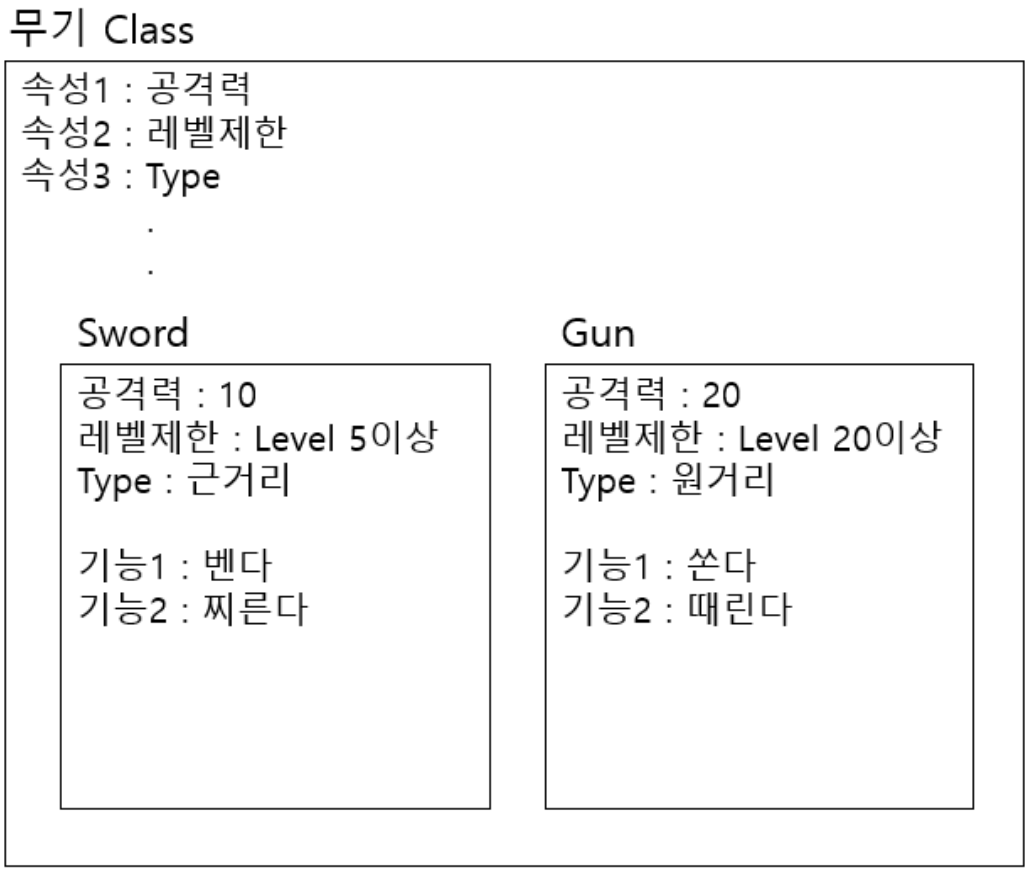
- 클래스의 이해
  - **과자틀과 과자**



> 과자 틀 → 클래스 (class)
> 과자 틀에 의해서 만들어진 과자 → 객체 (object)

> 과자 틀로 만든 과자에 구멍을 뚫거나 조금 베어 먹더라도 다른 과자에는 아무 영향이 없는 것과 마찬가지로 동일한 클래스로 만든 객체들은 서로 전혀 영향을 주지 않는다.

  - 게임 캐릭터
    - 각각의 캐릭터는 “속성”과 “기능”을 가지고 있음.



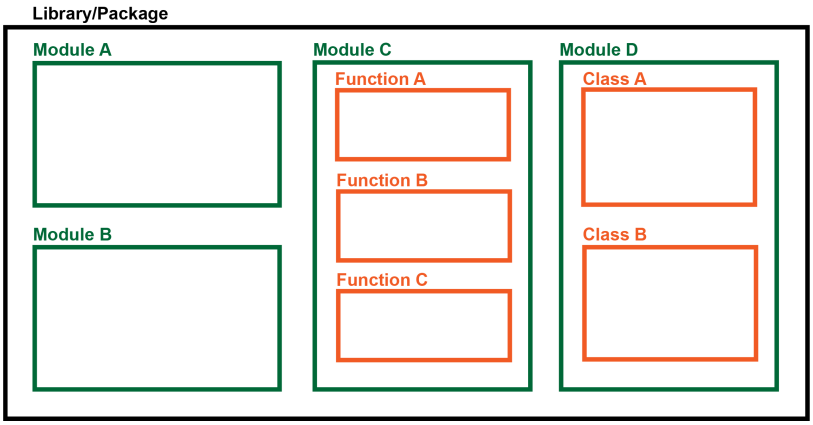
https://ybworld.tistory.com/20

모듈과 패키지

- 모듈 : (적절하게 묶인) 함수 혹은 클래스들의 집합
- 패키지(혹은 라이브러리) : (적절하게 묶인) 모듈로 이뤄진 집합
  - 하나의 목적이나 기능을 가지고 만들어짐

    ex) Pathlib : 경로와 관련된 클래스/함수의 집합

    datetime: 기본 날짜와 시간 데이터 변환/연산
- 프레임워크: 라이브러리와 비슷하나, 좀 더 큰 느낌의 추상적인 개념



https://thinkreen.github.io/python/py-FunctionModuleClass/

- 모듈/함수 불러오기
  - 예제

```
# 점심메뉴 선택

menu_list = ["짜장면", "김밥", "햄버거", "마라탕", "초밥", "치킨"]

# 모듈 import
import random # 여러가지 random 과 관련한 library
print(random.choice(menu_list))

# 함수 import
from random import choice
print(choice(menu_list))

# 이름 바꿔서 함수 import 하기
from random import choice as c
print(c(menu_list))

# (결과)
# 김밥
# 햄버거
# 김밥
# random을 사용하기 때문에 결과가 다르게 나올 수 있습니다. 그러나 셋의 기능은 모두 같습니다.
```

  - 모듈 탐색 경로
    - 절대 경로 / 상대 경로
      - import 하고자 하는 모듈이 어디에 있는가?(참고 링크)

        탐색 우선순위

        1. 첫 번째 경로는 지금 실행한 파일이 있는 폴더
        2. lib/python3.XX은 표준 라이브러리가 있는 폴더
        3. site-packages는 외부패키지가 일반적으로 저장되는 폴더
      - 탐색 Case

        - 같은 디렉토리에 있는 경우

          `from . import 모듈이름`
        - 바로 상위 디렉토리에 있는 경우

          `from .. import 모듈이름`
        - 상위 디렉토리의 다른 하위 디렉토리(A)에 있는 경우

          `from ..A import 모듈이름`
        - 차상위 디렉토리

          pathlib 라이브러리 사용(참고 링크)
    - Colab에서의 모듈 탐색 경로
        - 구글 드라이브와 연동, 구글 드라이브의 경로를 기준으로 참조(참고 링크)
    - 경로 탐색은 파일 입출력시에도 많이 사용.

      (이번 강의에서는 주로 모듈보다는 파일 경로를 찾기 위해 많이 사용 될 것)
- 파이썬 표준 라이브러리
    - 파이썬 설치 시 함께 내장되는 라이브러리
    - 가능한 표준 라이브러리가 제공하는 기능을 사용하는 쪽으로 코드를 작성하는 것이 좋음
- 통계에서 많이 쓰이는 라이브러리
    - Numpy, Scikit-learn, Pandas, Matplotlib...
    - Pandas

        - 표 형식 데이터를 다루는데 특화된 라이브러리
        - DataFrame이라는 자료형 (=클래스)를 사용하여 편리하게 요약, 연산 가능
        - Numpy(수학 연산) matplotlib(시각화) 라이브러리와 밀접한 dependency