SAMSUNG SW AI ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

삼성청년 SW·AI아카데미

0-1. AI를 위한 파이썬

SAMSUNG SW AI ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

0-1 AI를 위한 파이썬

SAMSUNG SW AI ACADEMY FOR YOUTH

INDEX

SAMSUNG SW AI ACADEMY FOR YOUTH

• INDEX

SAMSUNG SW AI ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

과정 소개

SAMSUNG SW AI ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

시작하기 전에

시작하기 전에

Confidential

Q) AI 왜 배워야 하나요?

- 이미 AI 라는 거대한 파도는 오고 있고, 피할 수 없으면 올라타야 한다.

Q) AI를 배워도 고학력(석/박사)만 취업 가능한 거 아닌가요?

- Research(연구)에만 해당하는 아주 큰 오해
- 실제 서비스를 만들기 위해서는 “이미 만들어진 모델” 을 가져오거나, “프레임 워크”를 이용해 고객의 요구사항이나 서비스를 만들어주는 사람이 결국 필요

➢ 결국 실제 문제를 해결하는 능력과 도구를 다루는 실무 역량이 훨씬 중요

➢ 그렇기에 수학? 통계? 을 몰라도 충분히 학습이 가능 (기본적인 내용은 같이 학습하면서 진행할 예정)
~~(사실 알고 있으면 깊은 이해가 가능하고, 알면 무조건 좋습니다..)~~

시작하기 전에

Confidential

우리는 이번 한달 간의 AI 학습 과정에서 아래 내용을 우선 숙지할 것을 목표로 설정

- 첫 술에 배부를 수 없다!
- 모든 내용을 디테일하게 암기하고, 이해하려하지 말고, 전체 워크플로우를 끝까지 따라가는 '**완주**'를 목표로 공부하기
- 상황에 맞는 다양한 모델을 다뤄보는 경험을 통해, 내가 만들 서비스에 적재적소의 모델을 사용하는 능력 갖추기
- 디테일한 부분에 집중하면, 결국 끝까지 완주할 수 없다.

시작하기 전에

좋은 학습 방법

- 전체적인 워크플로우 이해하고 ,경험하기
- 다양한 모델을 사용해보기
- 암기가 아닌, 자연스러운 과정을 이해하기

좋지 않은 학습 방법

- 복잡한 수식을 암기하고, 증명을 찾아보기
- 모델의 구체적인 내부 동작 원리를 이해하기 위해서 하루를 꼬박 날리기
- 매우 디테일한 질문에 사로잡혀, 해결할 때까지 공부해서 다음 진도 포기하기

물론 학습에 여유가 있거나, 앞선 학습을 잘 따라오고 있다면,
수식 / 내부 동작 원리에 대해서 학습하는 것도 아주 훌륭한 학습 방법!

SAMSUNG SW AI ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

실습 커리큘럼

실습 커리큘럼

AI를 위한 파이썬

- python, numpy, pandas 학습
- 워크플로우(문제 정의 -> 데이터 준비 -> 모델훈련 -> 평가)에 대한 간단한 완주

데이터 EDA(탐색적 데이터 분석) 및 모델 학습

- 데이터의 특징, 분포를 시각화를 통해 이해
- 모델 학습을 위한 데이터 전처리
- 예측 모델을 만들고, 성능을 평가
- 차원 축소 및 군집화를 통한 숨겨진 구조 찾기

실습 커리큘럼

MLP 구현

- PyTorch를 사용해 다층 퍼셉트론(MLP)이라는 기본적인 딥러닝 모델을 직접 설계, 학습, 평가하는 과정을 학습

토큰화/임베딩

- 자연어 처리(NLP) 모델의 발전사를 직접 코드로 구현하면서 학습
- 토큰화/임베딩 -> RNN -> LSTM -> 어텐션 -> 트랜스포머

합성 데이터

- LLM을 활용해 학습 데이터를 인공적으로 생성
- 합성 데이터의 품질을 자동으로 평가하는 전체 과정을 학습

실습 커리큘럼

CNN(합성공 신경망, 주로 이미지 학습)을 활용한 전이 학습

- 이미 만들어진 모델을 가져와서 내가 원하는 형태로 바꾸기
- 모델의 성능을 향상시키기 위한 기법(데이터 증강, 학습률 스케쥴러)
- 전통의 CNN과 최신의 ViT 을 경험
- AI 모델과 데이터셋이 모여있는 HuggingFace 경험

이미지 생성

- 여러 이미지 모델을 활용해보고, 직접 이미지를 생성해보기

실습 커리큘럼

Confidential

- **RAG**
  - 실시간 검색을 통한 답변을 생성하는 파이프라인 학습
  - LangChain 체험
- **Agent 서비스**
  - AI의 논리적 사고를 위한 ReAct 프레임워크(reasoning + Acting) 체험
- **PEFT(파라미터 효율적 튜닝)**
  - 적은 컴퓨팅 자원으로 LLM을 효율적으로 튜닝하는 방법 및 워크플로우 학습(LoRA, Unsloth , fine-tuning)
- **임베디드 모델 기반 서비스 구현**
  - 소형 언어 모델(SLM)을 파인튜닝하는 전체 워크플로우를 학습

SAMSUNG SW AI ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

Numpy

Numpy

- Numerical Python
- 파이썬에서 계산을 빠르게 하기 위한 라이브러리
- 빠른 연산 속도, 다양한 수학, N차원 배열 객체(ndarray)를 지원
- **$ pip install numpy 로 설치 후 사용**

```
n = 1_000_000

# 파이썬 리스트
py_list = list(range(n))

result_list = [x * 2 for x in py_list]  # 0.042010 초
```

대략 50배 가량 빠름

```
# NumPy 배열
np_array = np.arange(n)

result_array = np_array * 2   # 0.000779 초
```

Numpy - ndarray (1/2)

- N-dimensional array
- 대규모 숫자 데이터를 빠르고 효율적으로 처리하기 위해 만들어진 특수한 데이터 구조
- 모든 원소를 하나의 연속된 메모리 공간에 저장하기 때문에 캐시를 활용해서 극적인 속도를 보여줌
- 내부의 모든 원소들은 반드시 같은 데이터 타입을 저장해야 함
- N-차원(1차원, 2차원, 3차원, …) 등 원하는 만큼의 차원을 직관적으로 표현하고 다룰 수 있음
- 머신러닝에서의 Pandas, TensorFlow, PyTorch와 같은 다른 라이브러리들의 기반이 되는 핵심 데이터 구조

```python
import numpy as np

# 1차원 배열 생성
arr = np.array([1, 2, 3, 4, 5])
print("\n1차원 배열:\n", arr, '\n')

# 다차원 배열 생성
arr2d = np.array([[10, 20, 30], [40, 50, 60]])
print("2차원 배열:\n", arr2d)
```

```
1차원 배열:
 [1 2 3 4 5]

2차원 배열:
 [[10 20 30]
 [40 50 60]]
```

Numpy - ndarray (2/2)

- 배열 자체에 대한 중요한 정보들을 속성으로 가지고 있음
- .ndim
  - 배열의 차원 수
- .shape
  - 각 차원의 크기 (예: (3, 4)는 3행 4열)
- .size
  - 전체 원소의 개수
- .dtype
  - 원소의 데이터 타입

```python
import numpy as np

data = [[1, 2, 3],
        [4, 5, 6]]

arr = np.array(data)

print(arr, '\n')

print(f"차원: {arr.ndim}")
print(f"모양: {arr.shape}")
print(f"원소 개수: {arr.size}")
print(f"데이터 타입: {arr.dtype}")
```

```
[[1 2 3]
 [4 5 6]]

차원: 2
모양: (2, 3)
원소 개수: 6
데이터 타입: int32
```

SAMSUNG SW AI ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

numpy 함수

Numpy 함수 (1/2)

- 직접적으로 값을 입력해 배열을 만드는 방식 외에 다양한 생성 함수를 제공함
- **np.zeros()**
  - 모든 원소가 0으로 채워진 배열을 생성
  - 주로 배열의 틀을 미리 만들고, 나중에 값을 채우는 용도로 사용
- **np.ones()**
  - 모든 원소가 1로 채워진 배열을 생성
  - 초기화, 효율적인 수학 연산을 위한 용도로 사용
- **np.full(shape, fill_value)**
  - 지정한 모양의 배열을 만들고, 모든 요소를 지정한 값으로 채워줌
  - shape: 만들려는 배열의 모양
  - fill_value: 채우고 싶은 값

```python
import numpy as np

# 1. np.zeros(): 모든 원소가 0인 배열 생성
# 2행 4열의 모양으로, 모든 값이 0인 float 타입의 배열을 만듭니다.
zeros_array = np.zeros((2, 4))
print(zeros_array, '\n')
"""
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]]
"""

# 2. np.ones(): 모든 원소가 1인 배열 생성
# 3행 3열의 모양으로, 모든 값이 1인 integer(정수) 타입의 배열을 만듭니다.
ones_array = np.ones((3, 3), dtype=int)
print(ones_array, '\n')
"""
[[1 1 1]
 [1 1 1]
 [1 1 1]]
"""

# 3. np.full(): 지정한 모양의 배열을 만들고 ,값을 채우기
# 2행3열의 모양으로, 모든 값이 2인 배열을 생성
full_array = np.full((2, 3), 2)
print(full_array)
"""
[[2 2 2]
 [2 2 2]]
"""
```

Numpy 함수 (2/2)

- **np.arrange()**
  - 연속적인 숫자 배열 생성
  - 파이썬의 range() 와 동일하게 동작하되, 결과를 ndarray로 반환

- **np.linspace()**
  - 시작점부터 끝점까지 지정한 개수만큼 균일한 간격의 배열 생성
  - 주로 데이터 시각화, 함수 계산, 머신러닝 알고리즘에서 사용

```
import numpy as np

# 4. np.arange(): 연속적인 값을 가진 배열 생성
# 10부터 30 이전까지 5씩 건너뛰는 숫자로 배열을 만듭니다.
arange_array = np.arange(10, 30, 5)
print(arange_array, '\n')
"""
[10 15 20 25]
"""


# 5. np.linspace(): 균일한 간격을 가진 배열 생성
# 0부터 1까지의 구간을 총 5개의 원소로 균일하게 나눕니다.
linspace_array = np.linspace(0, 1, 5)
print(linspace_array, '\n')
"""
[0.   0.25 0.5  0.75 1.  ]
"""
```

SAMSUNG SW AI ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

인덱싱과 슬라이싱

인덱싱과 슬라이싱

인덱싱(indexing) (1/5)

- 1차원 배열
  - 파이썬 리스트와 사용법이 완전히 동일
- 2차원 배열
  - [행, 열]의 형태로 접근하여 특정 위치의 요소를 선택
  - 파이썬 리스트처럼 [행][열] 의 형태로도 접근할 수 있지만, 표준이 아니며 비효율적임

```
import numpy as np

# 2차원 배열 생성 (3행 4열)
arr2d = np.array([[1, 2, 3, 4],
                  [5, 6, 7, 8],
                  [9, 10, 11, 12]])

# 1행 2열의 요소에 접근 (0부터 시작하므로 두 번째 행, 세 번째 열)
element = arr2d[1, 2]  # 7

# 파이썬 리스트처럼 두 번 접근하는 것도 가능하지만,
# 콤마(,)를 사용하는 것이 NumPy의 표준 방식이며 더 효율적
list_element = arr2d[1][2] # 7
```

인덱싱과 슬라이싱

슬라이싱(slicing) (2/5)

- 1차원 배열
  - 파이썬 리스트와 사용법이 완전히 동일
- 2차원 배열
  - [행, 열]의 형태로 각각 슬라이싱할 수 있음
- 주의사항
  - Python
    - 슬라이싱을 하면 새로운 객체가 생성되고, 원본에 영향을 미치지 않음
  - Numpy
    - 슬라이싱을 하면 새로운 객체를 생성하지 않음
    - 대신, **원본 데이터의 일부를 들여다보는 뷰(View)**를 반환
    - 그렇기 때문에 슬라이싱한 객체를 수정하면, 원본도 수정됨

```python
import numpy as np

arr2d = np.array([[1, 2, 3, 4],
                  [5, 6, 7, 8],
                  [9, 10, 11, 12]])

# 첫 두 행과 1열부터 2열까지
sub_arr = arr2d[:2, 1:3]
# [[2 3]
#  [6 7]]

# 특정 행 전체를 가져오기
row = arr2d[1, :]  # [5 6 7 8]

# 1. 파이썬 예시
python_list = [10, 20, 30, 40, 50]

sliced_list = python_list[1:4]  # [20, 30, 40]
sliced_list[0] = 999

print(python_list)  # 변경되지 않음 ( [10, 20, 30, 40, 50])

# 1. 넘파이 예시
numpy_array = np.array([10, 20, 30, 40, 50])

sliced_array_view = numpy_array[1:4]  # [20 30 40]
sliced_array_view[0] = 999

print(numpy_array)  # 변경 됨([ 10 999  30  40  50])
```

Boolean Indexing (3/5)

- 조건식을 사용해 True인 요소만 추출하는 방법

```
import numpy as np

data = np.array([[1, 2],
                 [3, 4],
                 [5, 6]])

# 조건에 맞는 boolean 배열 생성
bool_mask = data > 3
print("Boolean 마스크:\n", bool_mask)
# [[False False]
#  [False  True]
#  [ True  True]]

# 마스크를 사용해 True 위치의 값만 추출 (1차원 배열로 반환됨)
print("\n3보다 큰 값들:", data[bool_mask]) # 출력: [4 5 6]

# 조건을 직접 인덱스에 넣어도 동일하게 동작합니다.
print("짝수만 추출:", data[data % 2 == 0]) # 출력: [2 4 6]
```

인덱싱과 슬라이싱

Fancy Indexing (4/5)

- 인덱스를 담은 배열을 사용하여 원하는 요소만 가져오는 방법
- 비연속적으로 요소들을 선택할 때 사용

```
import numpy as np

arr = np.array([10, 20, 30, 40, 50, 60, 70])

# 인덱스 0, 2, 5에 해당하는 요소를 선택
indices = [0, 2, 5]
print("팬시 인덱싱 결과:", arr[indices]) # 출력: [10 30 60]

# 2차원 배열에서도 가능
arr2d = np.arange(9).reshape(3, 3)
print("\n원본 2차원 배열:\n", arr2d)
# [[0 1 2]
#  [3 4 5]
#  [6 7 8]]

# 행 인덱스 [0, 2], 열 인덱스 [1, 2]를 선택
# 즉, (0, 1) 위치의 요소와 (2, 2) 위치의 요소를 선택
print("\n2D 팬시 인덱싱:", arr2d[[0, 2], [1, 2]]) # 출력: [1 8]
```

인덱싱과 슬라이싱

Reshape(행, 열) (5/5)

- 원소의 총 개수는 유지하고, 배열의 행과 열 구조를 변경
- 원본 배열의 총 원소 개수와 변경 후 배열의 총 원소 개수가 "반드시 같아야 함"
- "-1" 을 활용하면, 남은 차원 크기를 자동으로 계산해 줌

```python
import numpy as np

# arr는 여전히 12개의 원소를 가짐
arr = np.arange(12)

# 1. 1D -> 2D로 변경 (3행 4열)
reshaped_arr = arr.reshape(3, 4)
print("reshape(3, 4) 결과 (2D):\n", reshaped_arr, reshaped_arr.shape)

# 1. 행을 4로 지정하면, 열은 알아서 3으로 계산됨 (4 * 3 = 12)
arr_m1 = arr.reshape(4, -1)
print("\nreshape(4, -1) 결과:\n", arr_m1, arr_m1.shape)

# 2. 열을 2로 지정하면, 행은 알아서 6으로 계산됨 (6 * 2 = 12)
arr_m2 = arr.reshape(-1, 2)
print("\nreshape(-1, 2) 결과:\n", arr_m2, arr_m2.shape)
```

```
reshape(3, 4) 결과 (2D):
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]] (3, 4)

reshape(4, -1) 결과:
 [[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]] (4, 3)

reshape(-1, 2) 결과:
 [[ 0  1]
 [ 2  3]
 [ 4  5]
 [ 6  7]
 [ 8  9]
 [10 11]] (6, 2)
```

SAMSUNG SW AI ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

numpy 연산

배열의 요소 별 연산

- 배열의 같은 위치에 있는 요소들끼리 하나씩 연산을 수행하는 방식
- for 루프를 사용할 때보다 코드가 훨씬 간결해지고, 속도도 훨씬 빠름

배열과 숫자 간의 연산

- 배열의 모든 요소에 연산이 적용 됨

```
arr = np.array([[1, 2, 3],
                [4, 5, 6]])

# 덧셈: 모든 요소에 10을 더하기
add_result = arr + 10
print("덧셈 결과:\n", add_result)
# [[11 12 13]
#  [14 15 16]]

# 곱셈: 모든 요소에 3을 곱하기
mul_result = arr * 3
print("\n곱셈 결과:\n", mul_result)
# [[ 3  6  9]
#  [12 15 18]]
```

배열과 배열 간의 연산

- 모양이 같은 두 배열을 연산하면, 같은 위치의 요소들끼리 연산이 이루어짐

```
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[5, 6], [7, 8]])

# 배열 간 덧셈
add_arr = arr1 + arr2
print("배열 간 덧셈:\n", add_arr)
# [[ 6  8]
#  [10 12]]

# 배열 간 곱셈
mul_arr = arr1 * arr2
print("\n배열 간 곱셈:\n", mul_arr)
# [[ 5 12]
#  [21 32]]
```

numpy 연산

- 유니버셜 함수(Universal Functions, ufunc)
  - ndarray 의 모든 요소를 대상으로 수학적인 연산을 빠르고 효율적으로 수행하는 함수 목록
  - AI 분야에서 필수적으로 사용되니 알아둘 필요가 있음
  - 주요 유니버셜 함수
    - np.sqrt
    - np.exp
    - np.log
    - np.sin
    - np.dot
    - np.eye
    - np.random.randn

유니버셜 함수(Universal Functions, ufunc)

- np.sqrt(ndarray)
  - 입력 배열에 있는 각 요소의 제곱근 구하기
- np.exp(ndarray)
  - 입력 배열에 있는 각 요소의 지수 함수 구하기
- np.log(ndarray)
  - 입력 배열에 있는 각 요소의 자연로그 구하기
- np.sin(ndarray)
  - 입력 배열에 있는 각 요소의 사인 값 구하기

```
# 1. np.sqrt() : 배열의 각 요소에 제곱근을 계산
arr_basic = np.array([1, 4, 9, 16])
sqrt_result = np.sqrt(arr_basic)
print("## 1. np.sqrt() 결과 (제곱근) ##")
print(sqrt_result, '\n') # [1. 2. 3. 4.]

# 2. np.exp() : 배열의 각 요소에 지수 함수(e^x)를 적용
# e^0=1, e^1=2.718..., e^2=7.389...
arr_exp = np.array([0, 1, 2])
exp_result = np.exp(arr_exp)
print("## 2. np.exp() 결과 (지수 함수) ##")
print(exp_result, '\n')

# 3. np.log() : 배열의 각 요소에 자연로그(ln)를 계산
# np.exp()의 결과값을 다시 넣으면 원래 값이 나옵니다 (log(e^x) = x)
log_result = np.log(exp_result)
print("## 3. np.log() 결과 (자연로그) ##")
print(log_result, '\n') # [0. 1. 2.]

# 4. np.sin() : 배열의 각 요소에 사인(sin) 값을 계산
# sin(0)=0, sin(90도)=1, sin(180도)=0
arr_angles = np.array([0, np.pi/2, np.pi]) # 0, 90도, 180도를 라디안으로 표현
sin_result = np.sin(arr_angles)
print("## 4. np.sin() 결과 (사인 값) ##")
print(sin_result, '\n') # [0.0000000e+00 1.0000000e+00 1.2246468e-16]
# 참고: np.pi가 완벽한 무한소수가 아니므로 sin(pi)의 결과가
# 0이 아닌 아주 작은 값으로 나올 수 있습니다.
```

유니버셜 함수(Universal Functions, ufunc)

- np.eye(크기)
  - 단위 행렬을 생성
- np.random.randn(행, 열)
  - 주어진 형태의 배열을 생성하고, 표준 정규 분포(평균 0, 분산 1)을 따르는 무작위 실수를 채움
- np.dot(행렬A, 행렬B)
  - 두 배열 간의 행렬 곱(내적)을 계산하는 함수
  - A@B와 동일하게 동작

```python
# 5. 3x3 크기의 단위 행렬 생성
identity_matrix = np.eye(3)
print(identity_matrix)
# [[1. 0. 0.]
#  [0. 1. 0.]
#  [0. 0. 1.]]

# 6. 2행 3열 크기의 배열을 생성하고 난수로 채우기
random_matrix = np.random.randn(2, 3)
print(random_matrix)
# [[-0.53689437  1.23293836 -0.2343209 ],
#  [ 0.8310389   0.38950982 -0.6869436 ]]


# 7. 내적
# 2x3 행렬 A 생성
matrix_a = np.array([[1, 2, 3],
                     [4, 5, 6]]) # shape: (2, 3)

# 3x2 행렬 B 생성
matrix_b = np.array([[7, 8],
                     [9, 10],
                     [11, 12]]) # shape: (3, 2)

# 행렬 곱 계산 (A @ B 와 동일)
result_matrix = np.dot(matrix_a, matrix_b) # 결과 shape: (2, 2)
print(result_matrix)
# [[ 58  64]
#  [139 154]]
```

브로드캐스팅(Broadcasting)

- 모양이 다른 배열들 간에도 연산을 가능하게 해주는 매우 강력하고 중요하고 효율적인 기능!
- 크기가 작은 배열이 자동으로 확장되어 큰 배열의 모양에 맞춰 연산이 수행되는 원리

```
arr = np.array([[1, 2, 3],
                [4, 5, 6]])

# 브로드캐스팅 발생!
result = arr + 5
print(result)
"""
[[ 6  7  8]
 [ 9 10 11]]
"""
```

- 위의 예시에서 숫자 5는 [[5, 5, 5], [5, 5, 5]] 모양으로 확장되어 모든 요소에 더해짐
- 실제로 데이터를 복제하는 것이 아니라서 메모리도 절약
- 이후 AI 학습에서 "편향(bias)을 더하는 연산"에서 핵심적인 역할 (나중에 진행)

SAMSUNG SW AI ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

numpy 집계 함수

numpy 집계 함수

집계 함수(Aggregate Function)

- 배열을 입력받아, 데이터 전체를 대표하는 하나의 값으로 요약해주는 함수
- 주요 집계 함수
  - np.sum()
  - np.mean()
  - np.std()
  - np.var()
  - np.min()
  - np.max()
  - np.argmin()
  - np.argmax()

```
arr = np.array([0, 10, 20, 30, 40])

# 주요 집계 함수 예시
print(f"합계 (sum): {np.sum(arr)}")  # 100
print(f"평균 (mean): {np.mean(arr)}")  # 20.0
print(f"최댓값 (max): {np.max(arr)}")  # 40
print(f"최솟값 (min): {np.min(arr)}")  # 0
print(f"표준편차 (std): {np.std(arr):.2f}")  # 14.14
print(f"분산 (var): {np.var(arr)}")  # 200.0
print(f"최댓값의 인덱스 (argmax): {np.argmax(arr)}")  # 4
print(f"최솟값의 인덱스 (argmin): {np.argmin(arr)}")  # 0
```

numpy 집계 함수

axis 축

- 다차원 배열에서 집계 함수를 사용할 때, 연산을 수행할 "축"을 지정하는 매개변수
- 매개변수
  - axis=0 (세로 방향): 각 열(column)에 대해 연산을 수행
  - axis=1 (가로 방향): 각 행(row)에 대해 연산을 수행
  - 지정하지 않는 경우: 배열 전체의 모든 원소를 대상으로 연산합니다.

```
arr2d = np.array([[1, 2, 3],
                  [4, 5, 6],
                  [7, 8, 9]])

# 1. 전체 합계 (axis 지정 안 함)
total_sum = arr2d.sum() # 또는 np.sum(arr2d)
print(f"전체 합계: {total_sum}") # 1+2+...+9 = 45

# 2. axis=0 (열 기준 합계)
sum_axis_0 = arr2d.sum(axis=0)
print(f"열(axis=0) 기준 합계: {sum_axis_0}") # [12 15 18]

# 3. axis=1 (행 기준 합계)
# [1+2+3, 4+5+6, 7+8+9]
print(f"행(axis=1) 기준 합계: {sum_axis_1}") # [ 6 15 24]

# 4. 각 열에서 최댓값의 인덱스 찾기
# [0, 0, 0] 열: 7 (인덱스 2)
# [1, 1, 1] 열: 8 (인덱스 2)
# [2, 2, 2] 열: 9 (인덱스 2)
max_indices = np.argmax(arr2d, axis=0)
print(f"각 열의 최댓값 인덱스: {max_indices}") # [2 2 2]
```

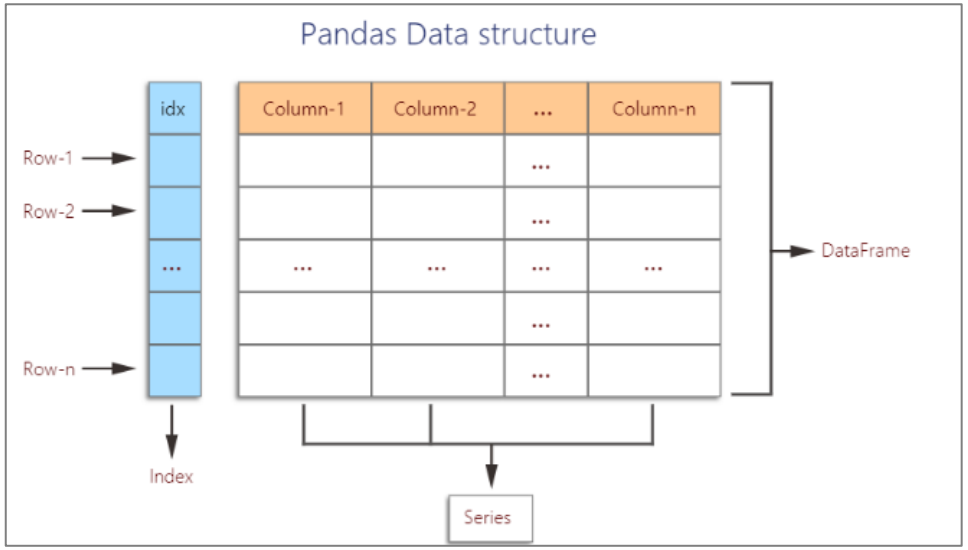
SAMSUNG SW AI ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

Pandas

Pandas

- 파이썬의 데이터 분석 라이브러리
- 2차원 배열 형태
- 엑셀 스프레드시트처럼 생긴 데이터를 다루는 데 최적화
- Series(시리즈)
  - 1차원 데이터, 하나의 열(Column)
- DataFrame(데이터프레임)
  - 2차원 데이터
  - Series가 모여 DataFrame이 됨
- **$ pip install pandas 로 설치 후 사용**




```python
import pandas as pd

data = {
    "Name": ["Alice", "Bob", "Charlie"],
    "Age": [25, 30, 35],
    "Score": [85.5, 90.3, 78.9]
}

df = pd.DataFrame(data)
print(df)
```

```
      Name  Age  Score
0    Alice   25   85.5
1      Bob   30   90.3
2  Charlie   35   78.9
```

https://lsh-story.tistory.com/56

데이터프레임 생성 방법

1. 딕셔너리를 사용하여 생성 (가장 일반적)

```
# 예시 데이터프레임 생성(ditionary 이용)
data = {
    "Name": ["Alice", "Bob", "Charlie"],
    "Age": [25, 30, 35],
    "Score": [85.5, 90.3, 78.9]
}

df = pd.DataFrame(data)
print(df)
```

2. 행 단위로 리스트를 구성하고, columns 인자를 사용하여 열 이름을 직접 지정

```
# 예시 데이터프레임 생성(list와 columns 활용)
data_list = [
    ["David", 28, 88.0],
    ["Eva", 22, 92.5],
    ["Frank", 33, 79.5]
]
columns = ["Name", "Age", "Score"]
df2 = pd.DataFrame(data_list, columns=columns)
print(df2)
```

SAMSUNG SW AI ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

Inspecting

Data Inspecting

- 데이터를 처음 만났을 때, 데이터의 구조와 내용을 파악하는 것

데이터 미리보기 (1/4)

- 데이터가 어떻게 생겼는 지 확인하는 방법
- df.head(n)
  - 데이터프레임의 처음 n 개의 행을 보여줌(default: 5)
- df.tail(n)
  - 데이터 프레임의 마지막 n개 행을 보여줌(default: 5)

```python
import pandas as pd

data = {'과목': ['영어', '수학', '과학', '사회', '국어'],
        '점수': [92, 78, 88, 95, 100],
        '응시자 수': [200, 150, 180, 210, 220]}
df = pd.DataFrame(data)
print(df)
print()
print(df.head(3)) # 처음 3개 행만 보기
print()
print(df.tail(3)) # 마지막 3개 행만 보기
```

```
   과목  점수  응시자 수
0  영어  92   200
1  수학  78   150
2  과학  88   180
3  사회  95   210
4  국어  100  220

   과목  점수  응시자 수
0  영어  92   200
1  수학  78   150
2  과학  88   180

   과목  점수  응시자 수
2  과학  88   180
3  사회  95   210
4  국어  100  220
```

데이터 구조 파악 (2/4)

- 데이터프레임의 기술적인 요약 정보를 보여줌
- 결측치를 확인하거나 데이터 타입을 파악할 때 사용
- df.info()
  - 아래 정보를 알려줌
    - 변수 타입
    - 전체 행의 개수와 인덱스 범위
    - 전체 열의 개수
    - 각 열의 이름, 데이터의 개수(결측치 제외), 데이터 타입
    - 메모리 사용량
    - 함수의 반환 값

```
import pandas as pd

data = {'과목': ['영어', '수학', '과학', '사회', '국어'],
        '점수': [92, 78, 88, 95, 100],
        '응시자 수': [200, 150, 180, 210, 220]}
df = pd.DataFrame(data)

print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   과목      5 non-null      object
 1   점수      5 non-null      int64
 2   응시자 수   5 non-null      int64
dtypes: int64(2), object(1)
memory usage: 248.0+ bytes
None
```

데이터 기술 통계 요약 (3/4)

- 숫자형 데이터를 가진 열에 대한 주요 통계량을 보여줌
- 데이터의 분포를 빠르게 파악하는 데 유용
- df.describe()
  - 아래 정보를 알려줌
    - count: 데이터 개수
    - mean: 평균
    - std: 표준편차
    - min: 최솟값
    - 25%, 50%, 75%: 사분위수
    - max: 최댓값

❖ 사분위 수

- 전체 데이터를 크기순으로 정렬한 뒤, 데이터의 양이 똑같도록 4등하는 기준점
- 1사분위수(Q1): 전체 데이터 중 "하위 25프로"에 해당하는 값

```python
import pandas as pd

data = {'과목': ['영어', '수학', '과학', '사회', '국어'],
        '점수': [92, 78, 88, 95, 100],
        '응시자 수': [200, 150, 180, 210, 220]}
df = pd.DataFrame(data)

print(df.describe())
```

```
             점수       응시자 수
count   5.000000    5.000000
mean   90.600000  192.000000
std     8.294577   27.748874
min    78.000000  150.000000
25%    88.000000  180.000000
50%    92.000000  200.000000
75%    95.000000  210.000000
max   100.000000  220.000000
```

데이터 프레임 기본 속성 확인 (4/4)

- df.shape
  - 데이터 프레임의 모양을 튜플로 반환
- df.columns
  - 모든 열의 이름을 리스트 형태로 반환
- df.index
  - 모든 행의 인덱스를 반환

```python
import pandas as pd

data = {'과목': ['영어', '수학', '과학', '사회', '국어'],
        '점수': [92, 78, 88, 95, 100],
        '응시자 수': [200, 150, 180, 210, 220]}
df = pd.DataFrame(data)

# (행 개수, 열 개수)
print(df.shape)  # (5, 3)

# 열 이름들
print(df.columns)  # Index(['과목', '점수', '응시자 수'], dtype='object')

# 행 인덱스들
print(df.index)  # RangeIndex(start=0, stop=5, step=1)
```

SAMSUNG SW AI ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

indexing & filtering

데이터 선택 및 필터링

- 데이터를 원하는 대로 고르고, 걸러내는 작업
- 데이터 분석 및 전처리 작업에 유용

열(Column) 선택

- 특정 열의 데이터만 확인

1. 한 개의 열 선택
   - df['컬럼명']
   - Series 형태로 반환
2. 여러 개의 열 선택
   - df[['컬럼1', '컬럼2']]
   - DataFrame 형태로 반환

```python
import pandas as pd

data = {'이름': ['철수', '영희', '민준', '지아', '서준'],
        '학년': [2, 3, 1, 3, 2],
        '과목': ['수학', '영어', '수학', '영어', '과학'],
        '점수': [85, 92, 78, 88, 95]}

df = pd.DataFrame(data, index=['a', 'b', 'c', 'd', 'e'])

print(df, '\n')

# '이름' 열만 선택
names = df['이름']
print(names, '\n')

# '이름'과 '점수' 열 선택
info = df[['이름', '점수']]
print(info)
```

```
   이름  학년  과목  점수
a  철수   2  수학  85
b  영희   3  영어  92
c  민준   1  수학  78
d  지아   3  영어  88
e  서준   2  과학  95

a    철수
b    영희
c    민준
d    지아
e    서준
Name: 이름, dtype: object

   이름  점수
a  철수  85
b  영희  92
c  민준  78
d  지아  88
e  서준  95
```

행(row) 선택

- 특정 행의 데이터를 확인

1. .loc
   - 이름(label) 기반 선택
   - 슬라이싱으로 여러 행 선택 가능
   - 파이썬 슬라이싱과는 다르게 "끝점을 포함!"
2. .iloc
   - 숫자 위치(Integer Position) 기반 선택
   - 0부터 시작하는 정수 위치를 사용하여 행을 선택
   - 파이썬 슬라이싱처럼 "끝점을 포함하지 않음"

```python
data = {'이름': ['철수', '영희', '민준', '지아', '서준'],
        '학년': [2, 3, 1, 3, 2],
        '과목': ['수학', '영어', '수학', '영어', '과학'],
        '점수': [85, 92, 78, 88, 95]}

df = pd.DataFrame(data, index=['a', 'b', 'c', 'd', 'e'])

# 인덱스 이름이 'b'인 행 선택
print(df.loc['b'], '\n')

# 인덱스 이름이 'a'부터 'c'까지인 행들 선택 (c 포함!)
print(df.loc['a':'c'], '\n')

# 0번째 위치의 행 선택 (첫 번째 행)
print(df.iloc[0], '\n')

# 0번째부터 3번째 이전까지 (0, 1, 2) 위치의 행들 선택 (3 미포함!)
print(df.iloc[0:3], '\n')
```

```
이름    영희
학년     3
과목    영어
점수    92
Name: b, dtype: object

   이름  학년  과목  점수
a  철수   2  수학  85
b  영희   3  영어  92
c  민준   1  수학  78

이름    철수
학년     2
과목    수학
점수    85
Name: a, dtype: object

   이름  학년  과목  점수
a  철수   2  수학  85
b  영희   3  영어  92
c  민준   1  수학  78
```

행과 열을 함께 선택

- [행, 열] 형태로 인자를 전달하여 원하는 셀의 값을 선택

```
data = {'이름': ['철수', '영희', '민준', '지아', '서준'],
        '학년': [2, 3, 1, 3, 2],
        '과목': ['수학', '영어', '수학', '영어', '과학'],
        '점수': [85, 92, 78, 88, 95]}

df = pd.DataFrame(data, index=['a', 'b', 'c', 'd', 'e'])

# 인덱스 'c'인 학생의 '과목' 정보
subject_c = df.loc['c', '과목']
print(f"\n인덱스 'c' 학생의 과목: {subject_c}", '\n')

# 1, 2번 행과 0, 2번 열에 해당하는 데이터 조각 선택
sub_df = df.iloc[[1, 2], [0, 2]]
print(sub_df)
```

```
인덱스 'c' 학생의 과목: 수학

   이름  과목
b  영희  영어
c  민준  수학
```

Boolean Indexing 필터링

- 특정 조건에 따라 True/False 값으로 구성된 "Boolean Series"를 생성하고, 이 시리즈를 데이터프레임에 적용하여 True인 행만 추출하는 방식

```
data = {'이름': ['철수', '영희', '민준', '지아', '서준'],
        '학년': [2, 3, 1, 3, 2],
        '과목': ['수학', '영어', '수학', '영어', '과학'],
        '점수': [85, 92, 78, 88, 95]}

df = pd.DataFrame(data, index=['a', 'b', 'c', 'd', 'e'])

# '점수'가 90점 이상인 행 필터링
filt = df['점수'] >= 90

df_high_score = df[filt]
print(df_high_score)

# 또는 한 줄로: df_high_score = df[df['점수'] >= 90]
```

```
   이름  학년  과목  점수
b  영희   3  영어  92
e  서준   2  과학  95
```

복합 조건 필터링

- 여러 조건을 동시에 적용할 때 논리 연산자를 사용하여 필터링
- 각 조건은 반드시 괄호로 묶어야 함

```
data = {'이름': ['철수', '영희', '민준', '지아', '서준'],
        '학년': [2, 3, 1, 3, 2],
        '과목': ['수학', '영어', '수학', '영어', '과학'],
        '점수': [85, 92, 78, 88, 95]}

df = pd.DataFrame(data, index=['a', 'b', 'c', 'd', 'e'])

# '과목'이 '수학'이고 '점수'가 80점 이상인 행 필터링
df_seoul_high = df[(df['점수'] >= 80) & (df['과목'] == '수학')]
print(df_seoul_high, "\n")

# '점수'가 92점이거나 '과목'이 '과학'인 행 필터링
df_perfect_or_busan = df[(df['점수'] == 92) | (df['과목'] == '과학')]
print(df_perfect_or_busan, "\n")

# '과목'이 '과학'이 아닌 행 필터링
df_not_seoul = df[~(df['과목'] == '과학')]
print(df_not_seoul, "\n")
```

```
   이름  학년  과목  점수
a  철수   2  수학  85

   이름  학년  과목  점수
b  영희   3  영어  92
e  서준   2  과학  95

   이름  학년  과목  점수
a  철수   2  수학  85
b  영희   3  영어  92
c  민준   1  수학  78
d  지아   3  영어  88
```

특수 조건 필터링

- 특정 값의 포함 여부나 결측치(NaN)을 처리할 때 사용
- .isin(리스트)
  - 해당 컬럼의 값이 주어진 리스트에 포함되는 행을 선택
- .isnull(값)
  - 해당 컬럼 값이 "결측치인 경우"에 True를 반환
- .notnull(값)
  - 해당 컬럼 값이 "결측치가 아닌 경우"에 True를 반환

```python
import pandas as pd
import numpy as np

data = {'이름': ['철수', '영희', '민준', '지아', '서준'],
        '학년': [2, 3, 1, 3, 2],
        '과목': ['수학', '영어', '수학', '영어', '과학'],
        '점수': [85, 92, 78, 88, 95]}

df = pd.DataFrame(data, index=['a', 'b', 'c', 'd', 'e'])

# '서준'의 '과목'과 '점수'를 NaN으로 설정
df.loc['e', '과목'] = np.nan
df.loc['e', '점수'] = np.nan

# '학년'이 2학년 또는 3학년인 행 필터링
target_grades = [2, 3]
filt_isin = df['학년'].isin(target_grades)
print(df[filt_isin], '\n')

# '과목'이 NaN인 행 필터링
filt_isnull = df['과목'].isnull()
df_is_null = df[filt_isnull]
print(df_is_null, '\n')

# '점수'가 NaN이 아닌 행 필터링
filt_notnull = df['점수'].notnull()
df_not_null = df[filt_notnull]
print(df_not_null)
```

```
   이름  학년   과목    점수
a  철수   2   수학  85.0
b  영희   3   영어  92.0
d  지아   3   영어  88.0
e  서준   2  NaN   NaN

   이름  학년   과목  점수
e  서준   2  NaN NaN

   이름  학년   과목    점수
a  철수   2   수학  85.0
b  영희   3   영어  92.0
c  민준   1   수학  78.0
d  지아   3   영어  88.0
```

SAMSUNG SW AI ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

데이터 전처리 및 변환

데이터 전처리 및 변환

결측치 처리

- 데이터의 신뢰도를 떨어뜨리고, 모델의 학습을 방해하는 "결측치"를 처리
- .dropna()
  - 결측치가 있는 행 또는 열을 제거
  - 데이터 손실 발생
- .fillna()
  - 결측값을 특정 값으로 대체
  - 제거보다 데이터 손실이 적어 일반적으로 선호

```python
import pandas as pd
import numpy as np

data = {'A': [1, 2, np.nan, 4, 5],
        'B': [6, np.nan, 8, np.nan, 10],
        'C': [11, 12, 13, 14, 15],
        'D': [np.nan, np.nan, np.nan, np.nan, np.nan]}

df = pd.DataFrame(data)
print(df, '\n')
# 결측치(NaN)가 하나라도 있는 행을 제거
df_dropped_rows = df.dropna()
print( df_dropped_rows, '\n')

# 결측치(NaN)가 하나라도 있는 열을 제거
df_dropped_cols = df.dropna(axis=1)
print(df_dropped_cols, '\n')  # 출력 안됨

# 모든 값이 결측치인 행을 제거
df_dropped_all_rows = df.dropna(how='all')
print(df_dropped_all_rows, '\n')
```

```
Empty DataFrame
Columns: [A, B, C, D]
Index: []

    C
0  11
1  12
2  13
3  14
4  15

     A     B   C   D
0  1.0   6.0  11 NaN
1  2.0   NaN  12 NaN
2  NaN   8.0  13 NaN
3  4.0   NaN  14 NaN
4  5.0  10.0  15 NaN
```

데이터 변환 및 수정 (1/2)

- 새로운 컬럼 형성
  - 데이터의 형태를 바꾸거나 새로운 특징을 생성하는 과정
  - 기존 컬럼들을 조합하여 새로운 컬럼 생성

```python
import pandas as pd
import numpy as np # 결측치 생성을 위해 numpy를 사용합니다.

data = {
    '이름': ['철수', '영희', '민수', '지아'],
    '수학': [95, 80, 60, 100],
    '영어': [88, 92, 70, 95]
}
df = pd.DataFrame(data)

# 기존 두 컬럼을 더하여 '총점' 컬럼 생성
df['총점'] = df['수학'] + df['영어']

# 조건에 따라 새로운 컬럼의 값 할당 (예: 90점 이상이면 'A', 아니면 'B')
df['학점'] = np.where(df['총점'] >= 180, 'A', 'B')

print(df)
```

```
   이름   수학  영어   총점 학점
0  철수   95  88  183  A
1  영희   80  92  172  B
2  민수   60  70  130  B
3  지아  100  95  195  A
```

데이터 변환 및 수정 (2/2)

- .astype()
  - 데이터 타입 변환이 잘못 지정된 경우 타입을 변경
  - ex: 숫자가 문자열로 인식된 경우

```
data2 = {
    '이름': ['철수', '영희', '민수', '지아'],
    '점수': [85.0, 92.5, np.nan, 78.0]
}

df2 = pd.DataFrame(data2)

# 1. 결측치 처리: NaN을 0으로 채웁니다.
df2['점수'] = df2['점수'].fillna(0)

# 2. '점수' 컬럼을 정수형(int)으로 변경
df2['점수'] = df2['점수'].astype(int)

# 3. 데이터 타입이 너무 커 메모리 효율을 위해 축소 (int64 -> int16)
df2['점수_int16'] = df2['점수'].astype('int16')

print(df2)
```

```
   이름  점수  점수_int16
0  철수  85        85
1  영희  92        92
2  민수   0         0
3  지아  78        78
```

SAMSUNG SW AI ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

참고

SAMSUNG SW AI ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

코랩

코랩(colab)

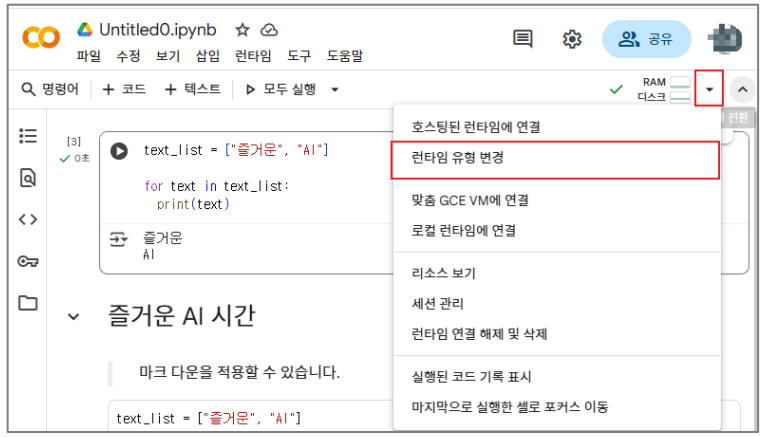
- 구글이 제공하는 무료 클라우드 기반 코딩 환경
- 웹 브라우저로 파이썬 코드를 바로 작성하고 실행할 수 있음
- 비싼 장비 없이도 복잡한 AI 모델을 훈련하고 실험할 수 있는 강력한 환경을 무료로 제공
  1. 대규모 계산을 빠르게 처리할 수 있는 **고성능 하드웨어(GPU, TPU)**를 무료로 제공 => AI 모델을 훈련시키는 데 유용
  2. AI 개발을 위한 텐서플로우(TensorFlow), 파이토치(PyTorch), 사이킷런(Scikit-learn) 등 수많은 라이브러리가 **미리 설치**되어 있음
- 코랩 노트북(.ipynb)는 구글 드라이브에 저장되어, 다른 사람과 쉽게 공유 가능
- 인터넷만 연결된다면, 다른 장소나 다른 컴퓨터로도 동일한 환경에서 작업을 이어나갈 수 있음

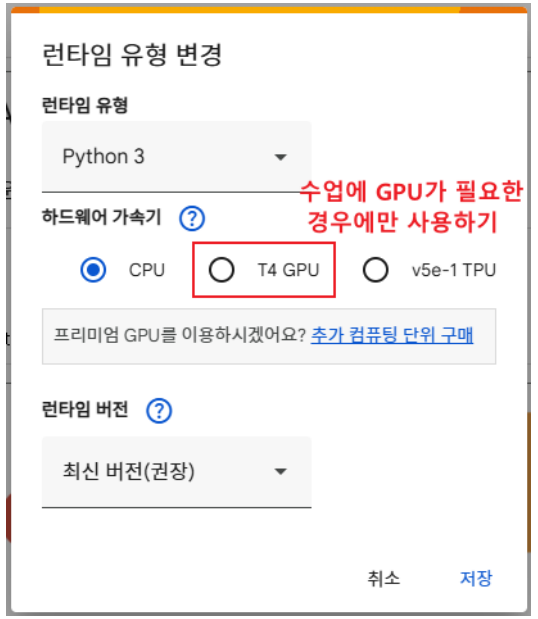
ipynb(Interactive Python Notebook)

- 주피터 노트북(Jupyter Notebook) 파일의 확장자
- 코드 외에 **코드 실행 결과, 설명 텍스트, 이미지, 차트** 등을 모두 하나의 문서에 담을 수 있음
- 특징
  - 전체 코드를 한 번에 실행하지 않고도 특정 부분(셀, cell)의 결과를 즉시 확인할 수 있음
  - 코드 외의 다른 양식(설명, 수식, 이미지 등)을 함께 작성할 수 있음
  - 시각적인 결과를 확인할 수 있음

런타임(runtime)

- 사용자의 코드를 실행하기 위해 구글이 할당해주는 클라우드 위의 **가상 컴퓨터**
- 각 코랩 노트북(ipynb)는 독립적인 런타임을 가지며, 서로 다른 노트북에 영향을 주지 않음
- 영구적이지 않으며, 일정 시간동안 활동이 없거나 최대 사용 시간에 도달하면 연결이 끊김
  (설치한 라이브러리, 변수, 업로드 파일 모두 삭제)
- 무료 사용량이 정해져있기 때문에 수업에 GPU가 필요한 경우에만 GPU 연결해서 사용하기(default: CPU)





SAMSUNG SW AI ACADEMY FOR YOUTH

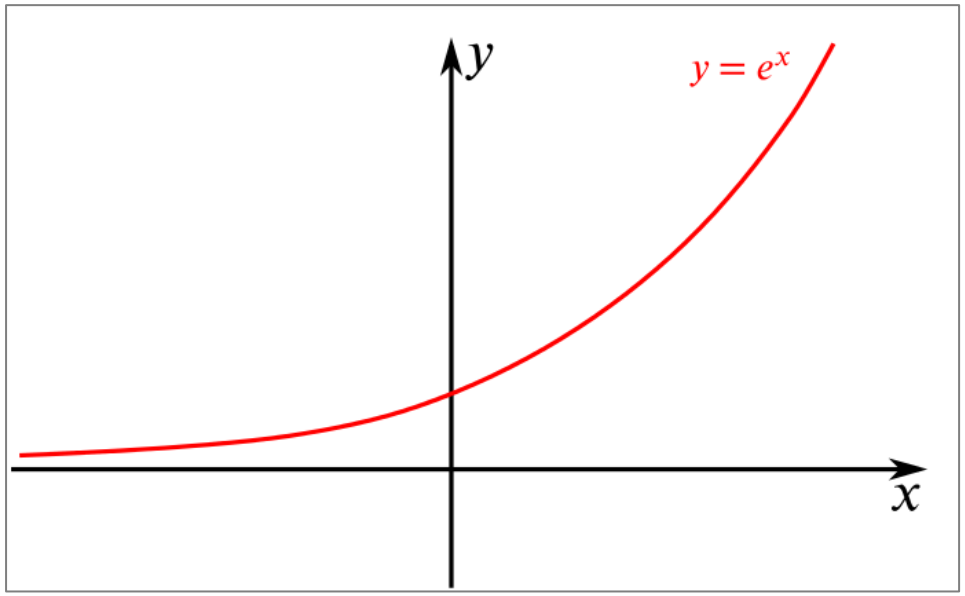
함께가요 미래로!
Enabling People

지수 함수

지수 함수

지수 함수(exponential function)

- $y = e^x$
- e(자연상수)
  - 약2.718…
  - 자연스러운 성장의 기준이 되는 숫자
- 작은 차이를 확실한 차이로 바꿔줌
  - 2점 => $e^2 \approx 7.4$, 5점 => $e^5 \approx 148.4$
- e^x 는 음수를 항상 양수로 만들어줌



- 위의 특징을 활용해 AI 에서 Softmax, Sigmoid 의 핵심으로 지수 함수가 활용
- **미분**해도 자기 자신을 유지

https://undergroundmathematics.org/combining-functions/r5595/solution

지수 함수

왜 하필 2.718 인가요 ?

왜 하필 2.718... 인가요? 🧐

이해하기 가장 좋은 방법은 **'복리 이자'** 예시입니다.

1,000원에 **연 100%** 이자를 주는 은행이 있다고 가정해 봅시다.

1. 이자를 1년에 한 번 준다면?

- 1년 후, 원금 1,000원 + 이자 1,000원 = **2,000원**이 됩니다. (원금의 **2**배)

2. 이자를 6개월마다 50%씩 두 번 준다면? (더 유리해집니다)

- **6개월 후**: 1,000원 + 500원(50%) = 1,500원
- **1년 후**: 1,500원 + 750원(1,500원의 50%) = **2,250원**이 됩니다. (원금의 **2.25**배)

3. 이자를 한 달마다 8.33%씩 열두 번 준다면?

- 1년 후, 원금은 약 **2.613...**배가 됩니다.

4. 이자를 주는 횟수를 무한대로 늘린다면? (가장 이상적인 성장)

이자를 매일, 매시간, 매분, 매초, 그리고 그보다 더 잘게 쪼개서 **'끊임없이(continuously)'** 이자가 붙는 가장 이상적인 상황을 상상해 보세요.

이때 원금은 무한정 커지지 않고, 정확히 어떤 **'한계점(limit)'**에 수렴하게 됩니다. 수학자들이 계산해 보니 그 한계점이 바로 **약 2.71828...**배였고, 이 중요한 숫자를 **`e`**라고 이름 붙였습니다.

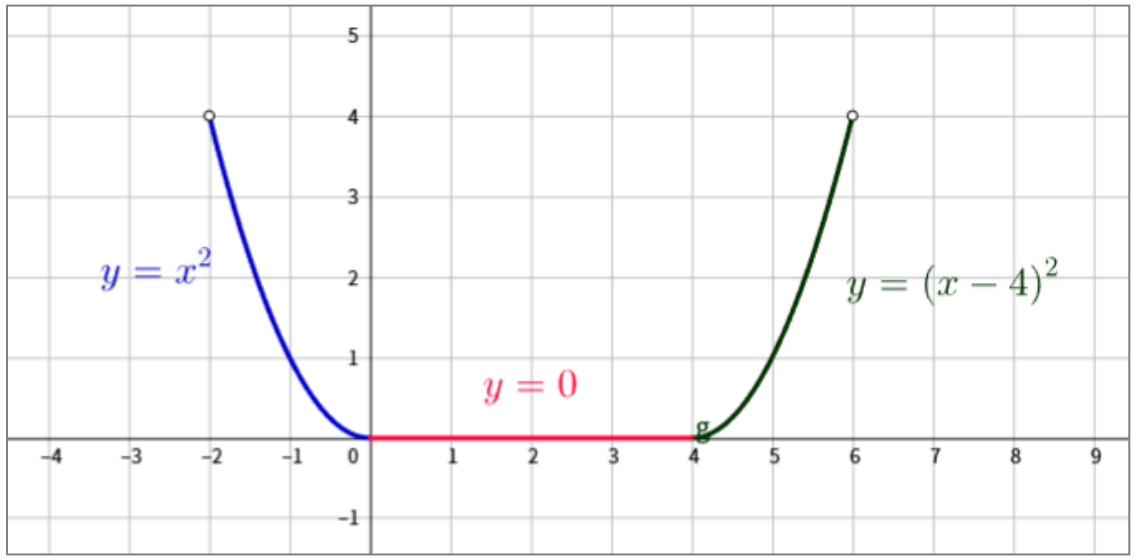
SAMSUNG SW AI ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

미분

[참고] 미분이란?

- 순간 변화량을 의미
- 함수의 특정 지점에서 순간적으로 얼마나 빠르게 변하는 지(기울기)를 알아내는 방법



https://www.geogebra.org/m/ctF6UJtX

SAMSUNG SW AI ACADEMY FOR YOUTH

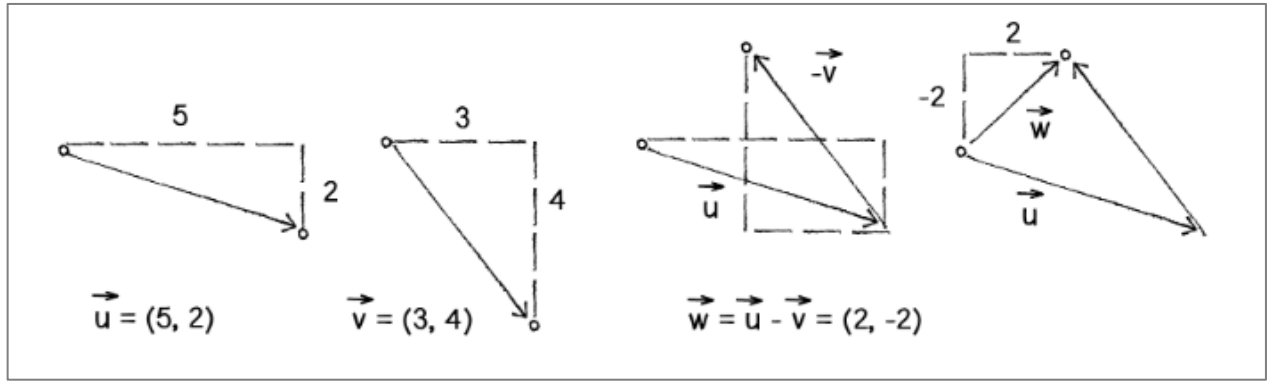
함께가요 미래로!
Enabling People

벡터와 내적

벡터와 내적

벡터

- “방향과 크기”를 나타내는 수학적 표현
- 1차원 벡터, [v]
  - 실수선에서 한 점을 가리키는 화살표
- 2차원 벡터, [x, y]
  - 2D 평면에서 "오른쪽 x, 위로 y" 을 가리키는 화살표
- 3차원 벡터, [x, y, z]
  - 3D 평면에서 "x축, y축, z축"을 가리키는 화살표
- 고차원 벡터, [v1, v2, v3, v4, … , vn]
  - N개의 실수 성분의 "크기와 방향"
  - 문서에 1,000 개의 단어가 있다면, 1,000차원 벡터가 문서를 나타낸다고 볼 수 있음



https://ko.khanacademy.org/computing/computer-programming/programming-natural-simulations/programming-vectors/a/more-vector-math

벡터와 내적

노름(Norm)

- 벡터의 크기 또는 길이를 측정하는 방법
- 원점(0, 0)으로부터 벡터의 끝점까지의 거리를 의미
- 모델의 오차 측정이나 정규화를 위해 사용
- ‖v‖ 와 같이 표기
- 예) L2 노름(유클리드 거리, 일반적으로 생각하는 두 점 사이의 직선 거리)
  - v = [a, b]
  - $\|v\| = \sqrt{a^2 + b^2}$

벡터와 내적

- 내적(Dot Product)
  - 두 벡터가 얼마나 같은 방향으로 향해 있는지를 반영한 값 (유사도, 상관 관계)
  - 각 성분을 곱한 뒤 모두 더한 하나의 값
  - 예)
    - 3차원에서 벡터 A(3, 3, 3), B(4, -1, 0)이 있을 때, A, B의 내적은?
    - $A \cdot B = (3*4) + (3*-1) + (3*0) = 12 + (-3) + (0) = 9$

행렬 곱

- 두 행렬을 곱해 새로운 행렬을 만드는 연산
- 단순한 요소별 곱셈이 아니라, “선형 변환”을 합성하는 중요한 계산
  - 선형 변환
    - 원점은 유지한채로 공간을 찌그리거나 늘리거나 하는 것
    - 확대, 축소, 회전 등이 선형 변환의 예시
    - 선형 변환을 통해 특징을 강조하면서, 데이터를 잘 표현할 수 있음
- 내적을 반복 수행하는 연산이며, 그로 인해 유사도를 계산하여 새로운 좌표를 부여한다고 이해하면 됨
- 계산 방식은 앞 행렬의 “행”과 뒤 행렬의 “열”을 곱해준다.
  => 즉, 앞 행렬의 열의 개수와 뒷 행렬의 행 개수가 같아야 계산 가능

$$\begin{bmatrix}1 & 3\\2 & 5\end{bmatrix}\begin{bmatrix}0\\3\end{bmatrix} = \begin{bmatrix}1\times 0 + 3\times 3\\2\times 0 + 5\times 3\end{bmatrix} = \begin{bmatrix}9\\15\end{bmatrix}$$

$$\begin{bmatrix}1 & 3\\2 & 5\end{bmatrix}\begin{bmatrix}0 & 1\\3 & 2\end{bmatrix} = \begin{bmatrix}9 & 7\\15 & 12\end{bmatrix}$$

SAMSUNG SW AI ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

통계 기초

평균(Mean)

- 데이터 집합의 중심을 나타냄
- 모든 데이터의 합 / 데이터의 개수

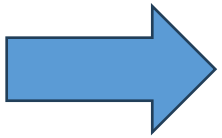
```python
import numpy as np

scores = np.array([70, 85, 90, 95, 100])

mean_score = np.mean(scores)

print(f"점수 데이터: {scores}")  # [ 70  85  90  95 100]
print(f"평균 점수: {mean_score}") # 88.0
```

분산(Variance)

- 데이터가 평균으로부터 얼마나 퍼져있는지를 나타내는 지표(데이터의 분포 정도를 측정)
- 데이터의 분산을 1로 만든다는 것은 서로 다른 단위를 같은 단위로 표현하겠다는 의미
- 의미
  - 분산이 작다: 데이터들이 평균 주변에 빽빽하게 모여 있음
  - 분산이 크다: 데이터들이 평균에서 멀리 흩어져 있음
- 공식
  1. 데이터의 평균을 구하고, 각 데이터 값에서 평균을 뺀다. (편차)
  2. 각 편차를 제곱해서 더한다. (제곱을 함으로써 음수가 사라지고, 멀리 떨어진 값에 더 큰 가중치를 부여)
  3. 제곱한 값들의 평균을 구함

제곱근을 씌운 것이 표준편차

$$\sigma^2 = \frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n}$$



$$\text{표준편차 } (\sigma) = \sqrt{\text{분산}} = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n}}$$

SAMSUNG SW AI ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

다음 시간에 만나요!

삼성 청년 SW · AI 아카데미