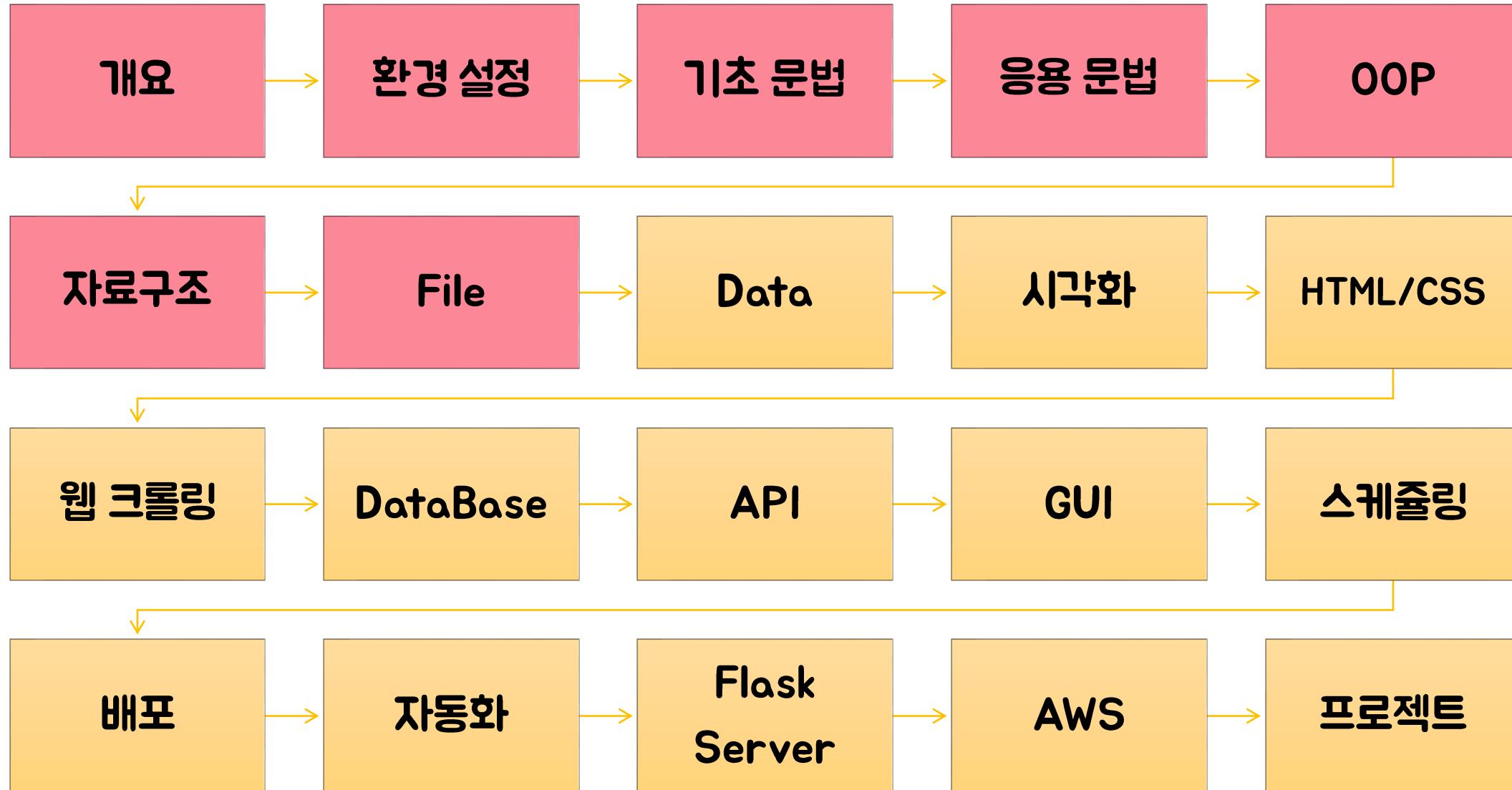




대한상공회의소
서울기술교육센터

나예호 교수



데이터 분석과 처리를 위한 강력한 라이브러리

행(Row)과 열(Column) 구조의 데이터

대규모 데이터 분석, 통계 처리, 데이터 시각화에 사용

```
!pip3 install pandas
```

```
import pandas as pd
```

Series (시리즈):

- 1차원 데이터 구조 (리스트와 비슷)
- 하나의 열(Column)과 인덱스(Index)로 구성됨

```
s = pd.Series([10, 20, 30, 40, 50])
print(s)
```

```
0    10
1    20
2    30
3    40
4    50
```

```
dtype: int64
```

DataFrame (데이터프레임):

- 2차원 데이터 구조 (엑셀 표와 유사)
- 여러 개의 시리즈(열)로 구성된 테이블 형식

```
data = {  
    "Name": ["Alice", "Bob", "Charlie"],  
    "Age": [25, 30, 28],  
    "City": ["Seoul", "Busan", "Incheon"]  
}  
  
df = pd.DataFrame(data)  
print(df)
```

```
# 상위 5개 데이터 미리 보기  
print(df.head())
```

```
# 데이터프레임 정보 확인(데이터 구조 및 타입 확인)  
print(df.info())
```

```
# 데이터 통계 요약(숫자형 데이터의 통계 요약 정보 확인)  
print(df.describe())
```

```
# CSV 파일 읽기
df2 = pd.read_csv("students.csv")
print(df2.head())
```

	Name	Age	Grade
0	Alice	20	A
1	Bob	22	B
2	Charlie	23	C
3	David	21	B
4	Eve	24	A

기본 인덱싱 및 슬라이싱

`df[column name]`

```
print(df["Name"])
```

`df[[column1, column2, ...]]`

```
print(df[["Name", "Age"]])
```

조건에 맞는 데이터 추출 : loc(location)

데이터프레임.loc[행 인덱싱 값, 열 인덱싱 값]

데이터프레임.loc[행 인덱싱 값]

```
print(df.loc[3, "Name"])
```

David

```
print(df.loc[3])
```

Name	David
Age	35
City	Daegu

조건에 맞는 데이터 추출 : loc(location)
데이터프레임.loc[행 슬라이싱, 열 슬라이싱]
데이터프레임.loc[행 슬라이싱]

```
print(df.loc[3:5, "Age":"City"])
```

	Age	City
3	35	Daegu
4	22	Gwangju
5	40	Daejeon

```
print(df.loc[3:5])
```

	Name	Age	City
3	David	35	Daegu
4	Eva	22	Gwangju
5	Frank	40	Daejeon

조건에 맞는 데이터 추출 : loc(location)
데이터프레임.loc[행 슬라이싱, 열 슬라이싱]
데이터프레임.loc[행 슬라이싱]

```
print(df.loc[:, "City"])
```

```
print(df["City"])
```

Boolean Indexing

True, False를 활용한 데이터 추출

```
df["Age"] >= 30
```

```
df.loc[df["Age"] >= 30]
```

	Name	Age	City
1	Bob	30	Busan
3	David	35	Daegu
5	Frank	40	Daejeon
7	Hannah	33	Suwon
9	Julia	31	Jeonju

Boolean Indexing

True, False를 활용한 데이터 추출

```
df.loc[(df["Age"] >= 30) & (df["City"] == "Busan")]
```

Name	Age	City
------	-----	------

1	Bob	30	Busan
---	-----	----	-------

조건에 맞는 데이터 추출 : iloc(Integer location)

데이터프레임.iloc[행 인덱싱 정수, 열 인덱싱 정수]

데이터프레임.iloc[행 인덱싱 정수]

```
print(df.loc[3, "Name"] )
```

David

```
print(df.loc[3] )
```

Name	David
Age	35
City	Daegu

```
print(df.iloc[3] )
```

Name	David
Age	35
City	Daegu

```
print(df.iloc[3, 0] )
```

David

조건에 맞는 데이터 추출 : iloc(Integer location)

데이터프레임.iloc[행 인덱싱 정수, 열 인덱싱 정수]

데이터프레임.iloc[행 인덱싱 정수]

```
print(df.loc[3:5, "Age":"City"])
```

	Age	City
3	35	Daegu
4	22	Gwangju
5	40	Daejeon

```
print(df.iloc[3:6, 1:3])
```

	Age	City
3	35	Daegu
4	22	Gwangju
5	40	Daejeon

조건에 맞는 데이터 추출 : iloc(Integer location)

데이터프레임.iloc[행 인덱싱 정수, 열 인덱싱 정수]

데이터프레임.iloc[행 인덱싱 정수]

```
print(df.loc[:, "City"])
```

```
print(df.iloc[:, 2])
```

```
print(df["City"])
```



```
print(df[2])
```

Boolean Indexing

True, False를 활용한 데이터 추출

```
df["Age"] >= 30
```

```
df.iloc[df["Age"] >= 30]
```

기본 통계 정보 확인

평균, 표준편차, 최소/최대값 등의 통계 요약 제공

```
# 나이(Age)에 대한 통계 요약  
print(df["Age"].describe())
```

count	10.000000
mean	30.000000
std	5.142416
min	22.000000
25%	27.250000
50%	29.500000
75%	32.500000
max	40.000000

그룹화 : **groupby()**

특정 기준(열)으로 그룹화하여 통계 계산

mean(), sum(), count(), max(), min() 등 그룹 함수 사용 가능

```
# 도시별 평균 나이 계산
print(df.groupby("City")["Age"].mean())
```

그룹화 : groupby()

```
# 도시별 평균 나이 계산
print(df.groupby("City") ["Age"].mean())
```

City	
Busan	30.0
Changwon	27.0
Daegu	35.0
Daejeon	40.0
Gwangju	22.0
Incheon	28.0
Jeonju	31.0
Seoul	25.0
Suwon	33.0
Ulsan	29.0

그룹화 : groupby()

	Department	Name	Salary
0	HR	Alice	5000
1	IT	Bob	6000
2	IT	Charlie	6500
3	Finance	David	7000
4	HR	Eve	5200
5	Finance	Frank	7100

그룹화 : `groupby()`

Q. 부서별 평균 급여 계산

Department	
Finance	7050.0
HR	5100.0
IT	6250.0

```
# 부서별 평균 급여
avg_salary = df_data.groupby("Department")["Salary"].mean()
print(avg_salary)
```

그룹화 : groupby()

Q. 부서별 인원 수

Department	
Finance	2
HR	2
IT	2

```
result = df_data.groupby("Department") ["Name"].count()  
print(result)
```

그룹화 : groupby()

Q. 부서별 급여 합계와 인원수

	sum	count
Department		
Finance	14100	2
HR	10200	2
IT	12500	2

급여 합계와 인원수

```
summary = df_data.groupby("Department")["Salary"].agg(["sum", "count"])
print(summary)
```

그룹화 : groupby()

Q. 부서별로 급여의 최대값과 최소값을 계산하세요.

	max	min
Department		
Finance	7100	7000
HR	5200	5000
IT	6500	6000

```
result = df_data.groupby("Department") ["Salary"].agg( ["max", "min"] )  
print(result)
```

그룹화 : `groupby()`

Q. 부서별 급여 합계와 평균을 동시에 구하세요.

	sum	mean
Department		
Finance	14100	7050.0
HR	10200	5100.0
IT	12500	6250.0

```
result = df_data.groupby("Department") ["Salary"].agg( ["sum", "mean"] )
print(result)
```

그룹화 : groupby()

Q. 급여가 6000 이상인 사람만 필터링한 후 부서별 평균 급여를 계산하세요.

Department

Finance	7050.0
IT	6250.0

```
filtered = df_data[df_data["Salary"] >= 6000]
result = filtered.groupby("Department") ["Salary"].mean()
print(result)
```

students_exam.csv 파일을 DataFrame으로 불러와서 데이터 구조를 확인하세요.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype  
--- 
 0   Name      30 non-null    object 
 1   Age       30 non-null    int64  
 2   City      30 non-null    object 
 3   Grade     30 non-null    object 
dtypes: int64(1), object(3)
memory usage: 1.1+ KB
None
```

students_exam.csv 파일을 DataFrame으로 불러와서

상위 데이터 5개 -> head()

하위 데이터 5개 -> tail()

랜덤 데이터 -> sample()

를 확인하세요.

	Name	Age	City	Grade
0	Alice	19	Ulsan	B
1	Bob	18	Daejeon	D
2	Charlie	20	Ulsan	D
3	David	31	Ulsan	C
4	Eva	27	Seoul	D

	Name	Age	City	Grade
24	Yuri	24	Daejeon	B

	Name	Age	City	Grade
25	Zoe	30	Daegu	D
26	Liam	33	Gwangju	F
27	Sophia	23	Incheon	D
28	Noah	30	Daejeon	A
29	Isabella	19	Incheon	F

students_exam.csv 파일을 DataFrame으로 불러와서
상위 데이터 5개 -> head()
하위 데이터 5개 -> tail()
랜덤 데이터 -> sample()
를 확인하세요.

```
import pandas as pd

# CSV 파일 불러오기
df = pd.read_csv("students_exam.csv")

# 데이터 구조 확인
# print(df.info())
print(df.head())
print(df.tail())
print(df.sample())
```

students_exam.csv 자료 내
나이가 25세 이상인 학생들만 필터링하여 출력하세요.

	Name	Age	City	Grade
3	David	31	Ulsan	C
5	Frank	31	Ulsan	B
8	Ian	35	Busan	C
9	Jack	33	Ulsan	A
11	Leo	30	Daejeon	F
13	Nathan	31	Gwangju	D
14	Olivia	34	Gwangju	F
17	Rachel	33	Daegu	F
22	Will	33	Daejeon	B
25	Zoe	30	Daegu	D
26	Liam	33	Gwangju	F
28	Noah	30	Daejeon	A

students_exam.csv 자료 내
도시별 평균 나이를 계산하여 출력하세요.

City	
Busan	27.000000
Daegu	27.000000
Daejeon	26.666667
Gwangju	30.500000
Incheon	21.000000
Seoul	24.500000
Ulsan	25.875000
Name:	Age, dtype: float64

students_exam.csv 자료 내
도시별 평균 나이를 계산하여 출력하세요.

```
# 도시별 평균 나이 계산
average_age = df.groupby("City")["Age"].mean()
print(average_age)
```

특정 열(Name, City)만 추출하여 새로운 DataFrame
new_df으로 저장하세요.

```
# 특정 열 추출
new_df = df[["Name", "City"]]
print(new_df)
```

나이가 30세 이상인 데이터 필터링 후
filtered_data에 담고
filtered_students.csv로 저장하시오

```
# 나이가 30세 이상인 데이터 필터링 후 저장
filtered_data = df[df["Age"] >= 30]
filtered_data.to_csv("filtered_students.csv", index=False)
```

데이터 변환

apply() : 전체 열에 함수를 적용

```
# 모든 이름을 대문자로 변환
def to_uppercase(name):
    return name.upper()

df["Name"] = df["Name"].apply(to_uppercase)
print(df)
```

데이터 변환

`apply()` : 전체 열에 함수를 적용

`lambda` 함수는 간단한 변환작업에 사용

```
# 모든 이름을 대문자로 변환
df["Name"] = df["Name"].apply(lambda x: x.upper())
print(df)
```

lambda 함수

lambda 매개변수 : 표현식

```
def add_two_number(x, y):  
    return x + y
```

```
add_two_number(10, 20)
```

```
30
```

```
(Lambda x, y: x + y)(10, 20)
```

```
30
```

데이터 변환

map(): 간단한 변환 작업

```
# 도시 이름을 간단히 변경
city_map = {"Seoul": "SEO", "Busan": "BSN", "Incheon": "ICN"}
df["City"] = df["City"].map(city_map)
print(df)
```

데이터프레임 복사

얕은 복사 df2 = df

-> df 데이터 변경 시 df2 반영 O

깊은 복사 df2 = df.copy()

-> df 데이터 변경 시 df2 반영 X

```
df2 = df.copy()  
print(df2)
```

```
df3 = df.copy()  
print(df3)
```

`fillna()` → 결측치(NaN)를 특정 값으로 채움

`inplace=True` → 원본 데이터에 직접 적용

```
# 결측치(NaN)를 ABC로 채우기
df2["City"].fillna("ABC", inplace=True)
print(df2)
```

`dropna()` → 결측치가 포함된 행 또는 열 삭제

`axis=0` → 행 삭제, `axis=1` → 열 삭제

```
# 결측치가 포함된 행 삭제
df3.dropna(inplace=True)
print(df3)
```

새로운 열 추가 및 가공

Age Category를 추가하여 나이에 따른 분류를 수행해보자

```
# 나이에 따라 새로운 열 추가
df["Age Category"] = df["Age"].apply(lambda x: "Senior" if x >= 30 else "Junior")
print(df)
```

sample_students.csv를 불러와
students 이름의 dataframe으로 저장하자
다음의 작업을 수행하시오

1. 이름(Name)을 모두 소문자로 변환하세요.
2. 나이(Age)가 결측치인 경우 25으로 대체하세요.
3. Grade가 C 이상이면 Pass로, D미만이라면 Fail을 표시하는
새로운 열(PassFail)을 추가하세요.
4. 가공된 dataframe을 새로운 CSV 파일(cvt_students)로
저장하세요.

```
students["Name"] = students["Name"].apply(lambda x: x.lower())
print(students)
```

```
students["Age"].fillna(25, inplace=True)
print(students)
```

```
students["PassFail"] = students["Grade"].apply(lambda x: "Pass" if x in ["A", "B", "C"] else "Fail")
print(students)
```

```
students.to_csv("cvt_students.csv", index=False)
print("데이터가 저장되었습니다!")
```

데이터가 저장되었습니다!

sample_students.csv를 불러와
students 이름의 dataframe으로 저장하자
다음의 작업을 수행하시오

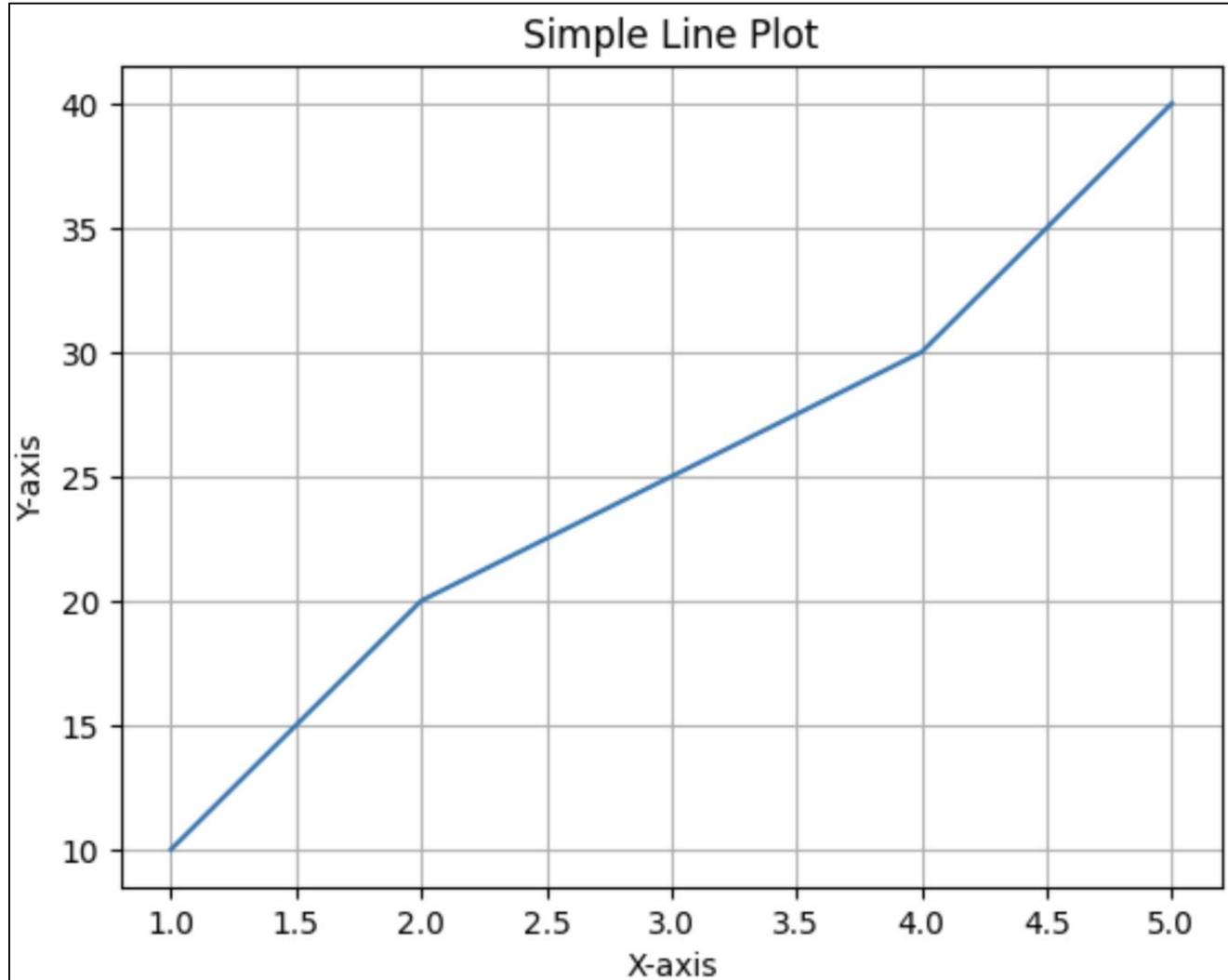
1. 이름(Name)을 모두 소문자로 변환하세요.
2. 나이(Age)가 결측치인 경우 25으로 대체하세요.
3. Grade가 C 이상이면 Pass로, D미만이라면 Fail을 표시하는
새로운 열(PassFail)을 추가하세요.
4. 가공된 dataframe을 새로운 CSV 파일(cvt_students)로
저장하세요.

matplotlib : 데이터를 다양한 방법으로 도식화하는 라이브러리

```
!pip3 install matplotlib
```

```
import matplotlib.pyplot as plt
```

선 그래프(Line Plot)



matplotlib : 데이터를 다양한 방법으로 도식화하는 라이브러리

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y = [10, 20, 25, 30, 40]

plt.plot(x, y, marker='o')
plt.title("Simple Line Plot")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.grid(True)
plt.show()
```

```
plt.plot(x, y, marker='o')
```

- 역할: x와 y 데이터로 선 그래프를 그리는 핵심 코드

- 설명:

- x = [1, 2, 3, 4, 5] (x축 좌표)
- y = [10, 20, 30, 40, 50] (y축 좌표)

- 옵션:

- marker='o' → 데이터 포인트에 동그라미(●) 마커 표시
- (기본값) → 선 그래프
- 's' → 네모, '^' → 삼각형 등 다양한 마커 지원

```
plt.title("Simple Line Plot")
```

- **역할:** 그래프의 제목을 설정

- **설명:**

- "Simple Line Plot" → 그래프 위에 표시될 제목
- 가독성을 높이기 위해 사용

```
plt.xlabel("X-axis")
```

- 역할: x축 레이블(이름)을 설정
- 설명:

- "X-axis" → x축 아래에 표시되는 텍스트
- x축의 의미를 설명할 때 사용
- fontsize=14 옵션으로 글씨 크기 조절 가능
- color= "blue" 옵션으로 글씨 색상 변경 가능

```
plt.ylabel("Y-axis")
```

- 역할: y축 레이블(이름)을 설정
- 설명:

- "Y-axis" → y축 왼쪽에 표시되는 텍스트
- y축 값이 무엇을 의미하는지 설명
- fontsize=14 옵션으로 글씨 크기 조절 가능
- color= "blue" 옵션으로 글씨 색상 변경 가능

plt.grid(True)

- 역할: 그래프에 격자(grid) 선을 추가
- 설명:

- True → 격자 표시
- False → 격자 제거
- 격자는 데이터의 정확한 위치를 읽기 쉽게 만들어줌
- linestyle='--' → 점선
- color='gray' → 회색
- alpha=0.7 → 투명도 조절

plt.show()

• **역할:** 그래프를 화면에 출력

• **설명:**

- Matplotlib은 여러 개의 그래프를 그릴 수 있어

- plt.show()로 최종 출력

- 반드시 필요한 단계 (없으면 그래프가 표시되지 않을 수 있음)

plt.show()

- **역할:** 그래프를 화면에 출력
- **설명:**

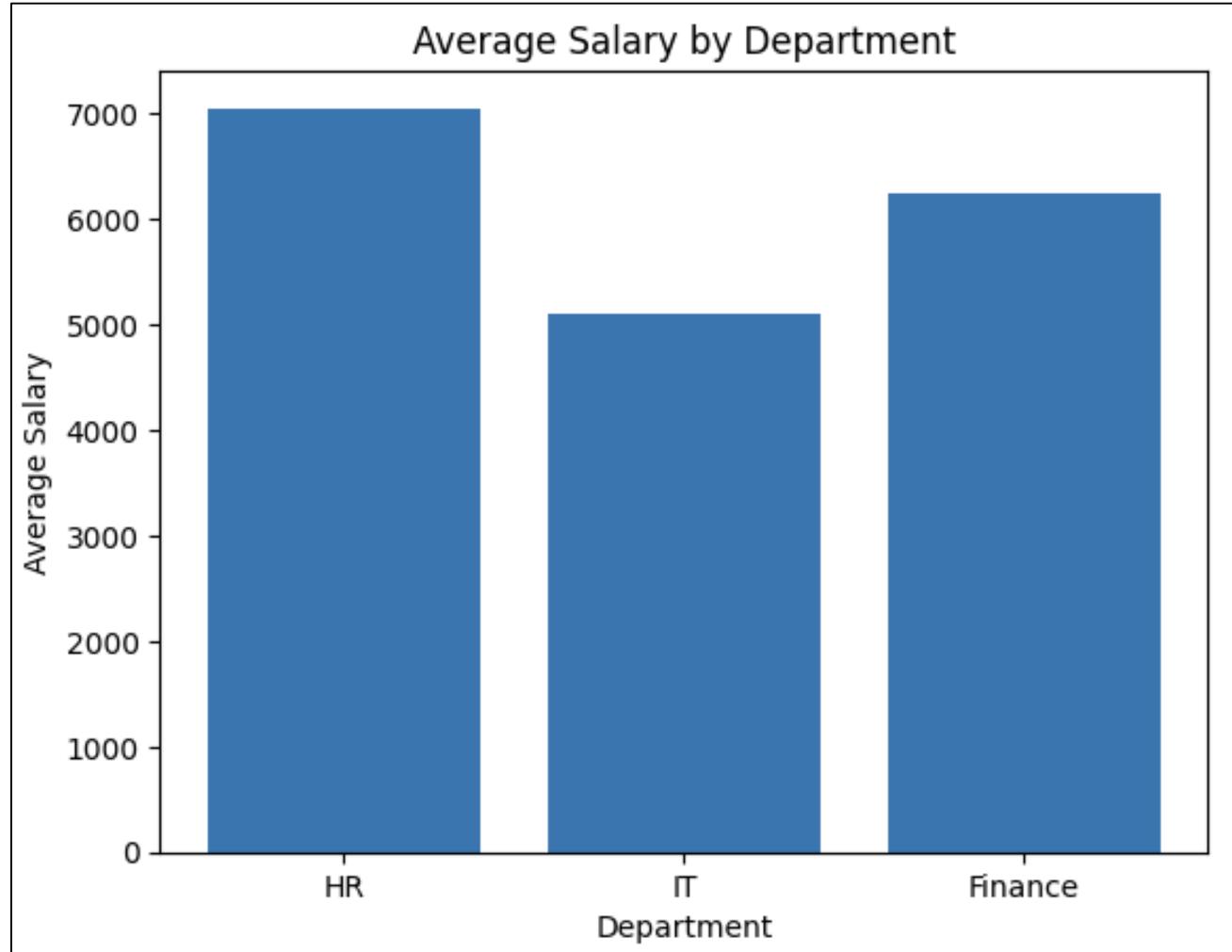
- Matplotlib은 여러 개의 그래프를 그릴 수 있어 plt.show()로 최종 출력
- 반드시 필요한 단계 (없으면 그래프가 표시되지 않을 수 있음)

plt.show()

- **역할:** 그래프를 화면에 출력
- **설명:**

- Matplotlib은 여러 개의 그래프를 그릴 수 있어 plt.show()로 최종 출력
- 반드시 필요한 단계 (없으면 그래프가 표시되지 않을 수 있음)

막대 그래프(Bar Plot)



막대 그래프(Bar Plot)

```
# 부서별 급여 시각화
departments = df_data["Department"].unique()
avg_salary = df_data.groupby("Department") ["Salary"].mean()

plt.bar(departments, avg_salary)
plt.title("Average Salary by Department")
plt.xlabel("Department")
plt.ylabel("Average Salary")
plt.show()
```

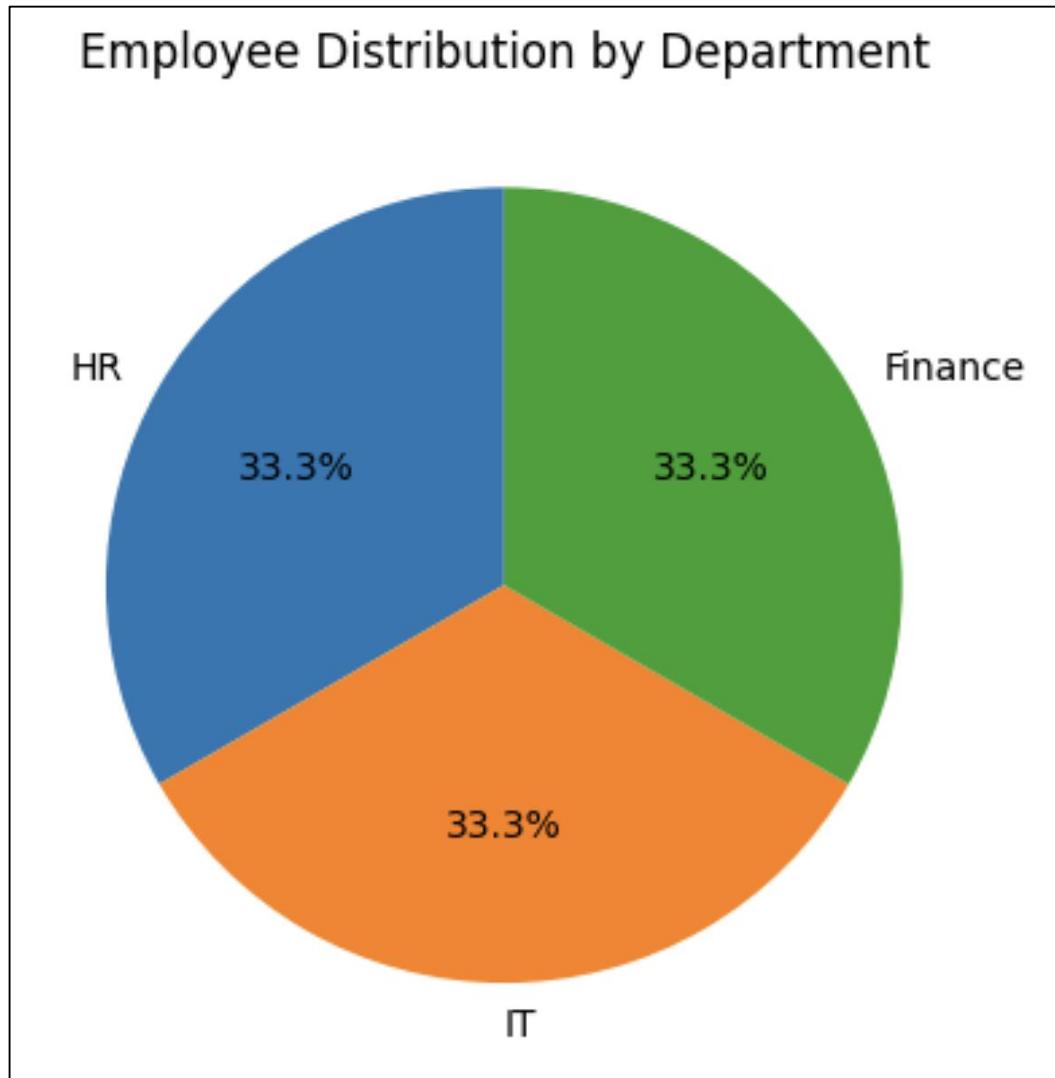
```
plt.bar(departments, avg_salary)
```

- 역할: 막대 그래프를 생성하는 핵심 코드

- 구조:

- **departments** → x축에 표시될 범주(부서 이름)
- **avg_salary** → 각 범주에 해당하는 막대의 높이(급여)

파이 차트(Pie Chart)



파이 차트(Pie Chart)

```
# 부서별 인원 비율 시각화
counts = df_data["Department"].value_counts()

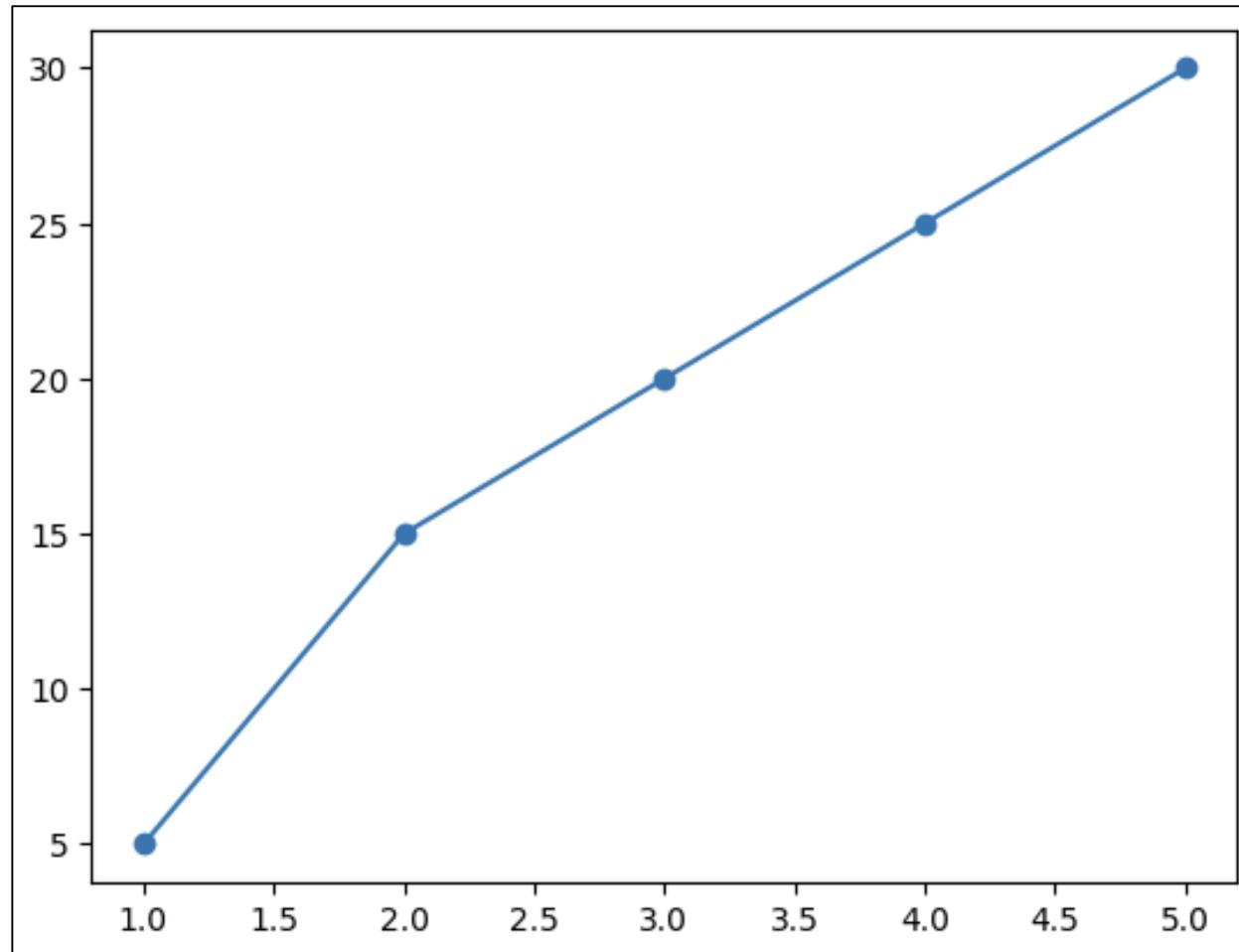
plt.pie(counts, labels=counts.index, autopct='%.1f%%', startangle=90)
plt.title("Employee Distribution by Department")
plt.show()
```

```
plt.pie(counts, labels=departments,  
autopct='%.1f%%', startangle=90)
```

- 역할: 파이 차트를 생성하는 핵심 코드
- 주요 매개변수:

- counts: 각 부서의 인원 수 → 파이 조각의 크기를 결정
- labels: 각 조각에 표시될 라벨(부서 이름)
- autopct='%.1f%%': 각 조각에 퍼센트(%) 표시
- startangle=90: 파이 차트의 시작 각도 설정 (90도부터 시작)

$x = [1, 2, 3, 4, 5]$ 와 $y = [5, 15, 20, 25, 30]$ 로
선 그래프를 그리세요.

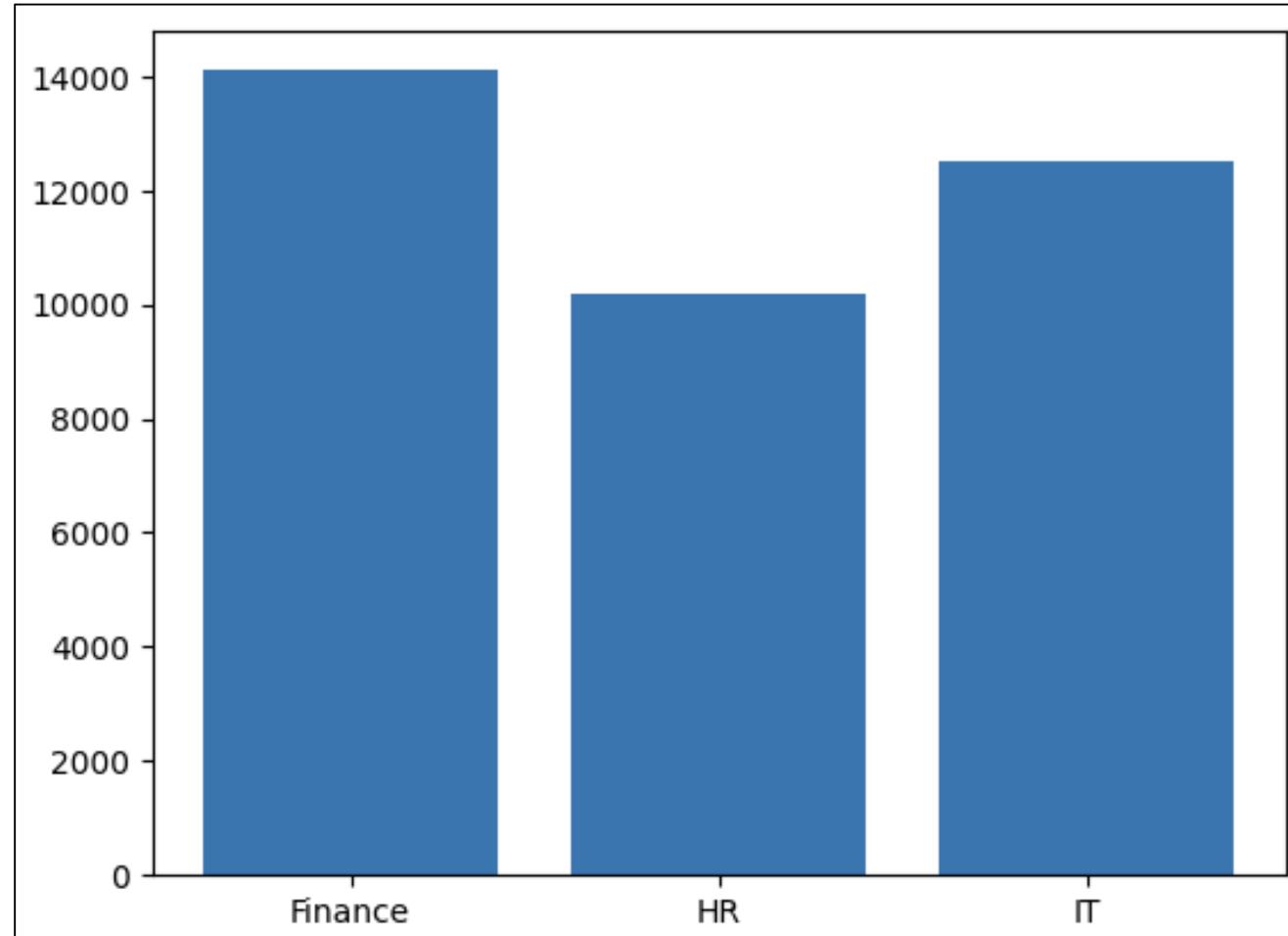


$x = [1, 2, 3, 4, 5]$ 와 $y = [5, 15, 20, 25, 30]$ 로
선 그래프를 그리세요.

```
x = [1, 2, 3, 4, 5]
y = [5, 15, 20, 25, 30]
```

```
plt.plot(x, y, marker='o')
plt.show()
```

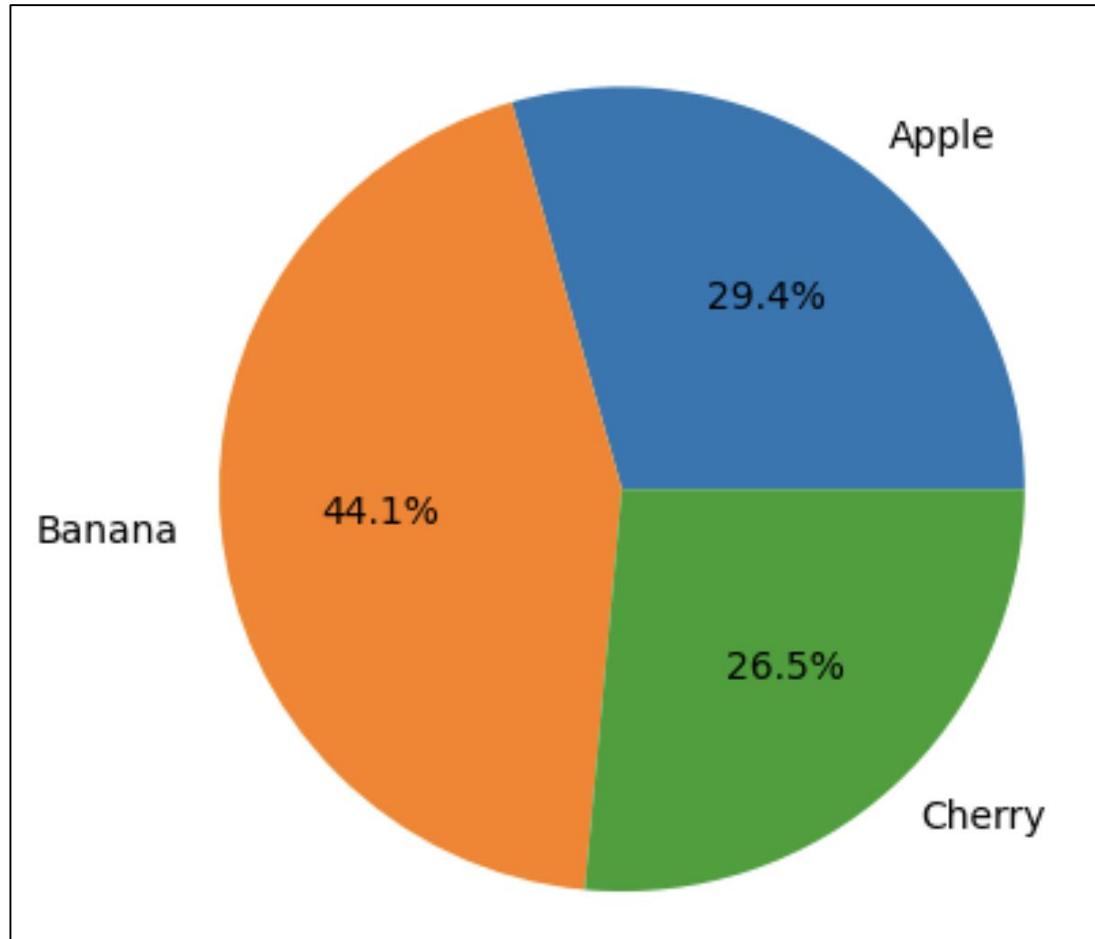
부서별 총 급여를 막대 그래프로 시각화하세요.



부서별 총 급여를 막대 그래프로 시각화하세요.

```
sum_salary = df_data.groupby("Department")["Salary"].sum()  
plt.bar(sum_salary.index, sum_salary.values)  
plt.show()
```

과일 판매량(Apple:100, Banana:150, Cherry:90)을
파이 차트로 시각화하세요.

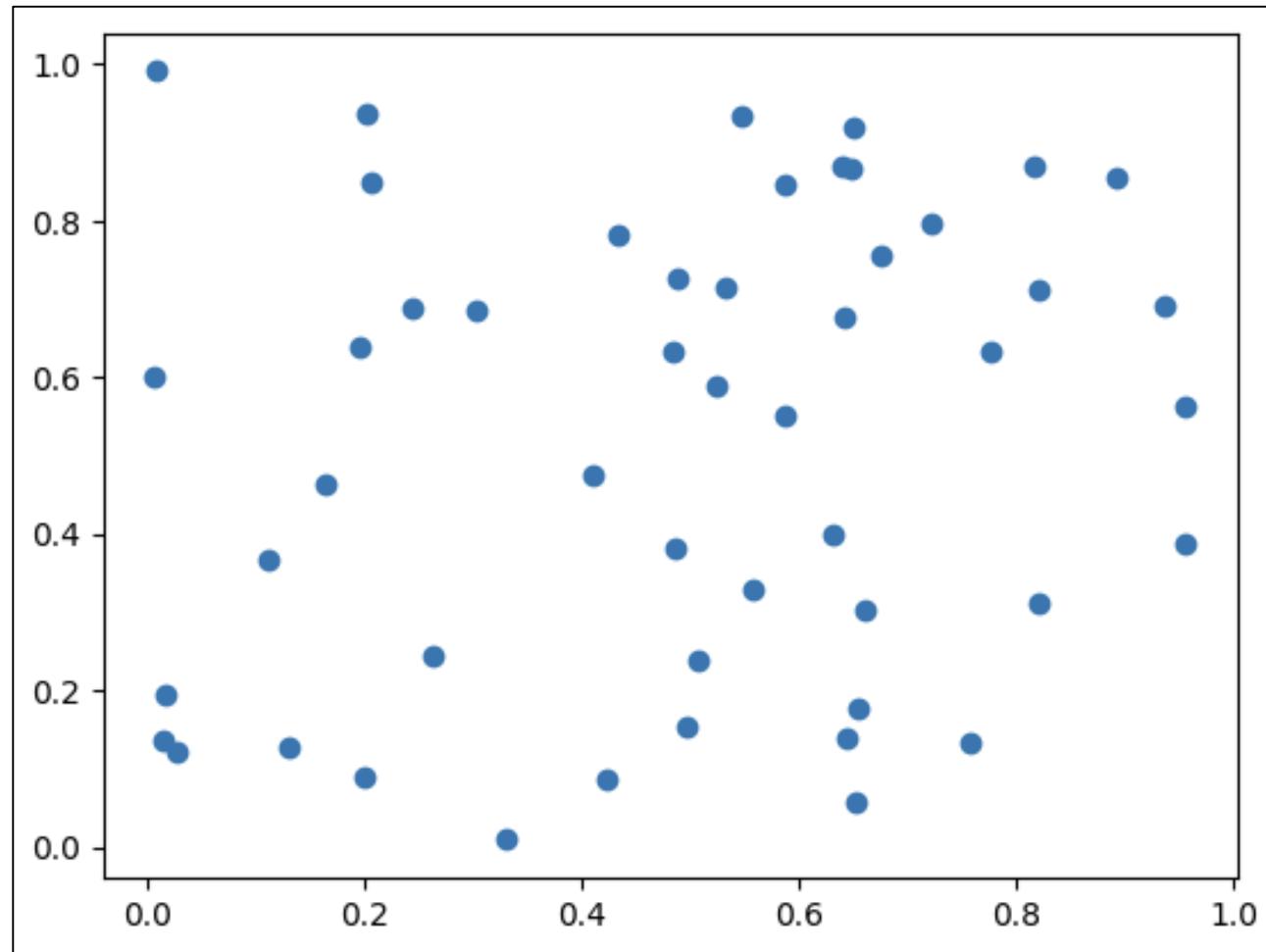


과일 판매량(Apple:100, Banana:150, Cherry:90)을
파이 차트로 시각화하세요.

```
labels = ["Apple", "Banana", "Cherry"]
sizes = [100, 150, 90]

plt.pie(sizes, labels=labels, autopct="%1.1f%%")
plt.show()
```

그 외 사용되는 시각화 : 산점도



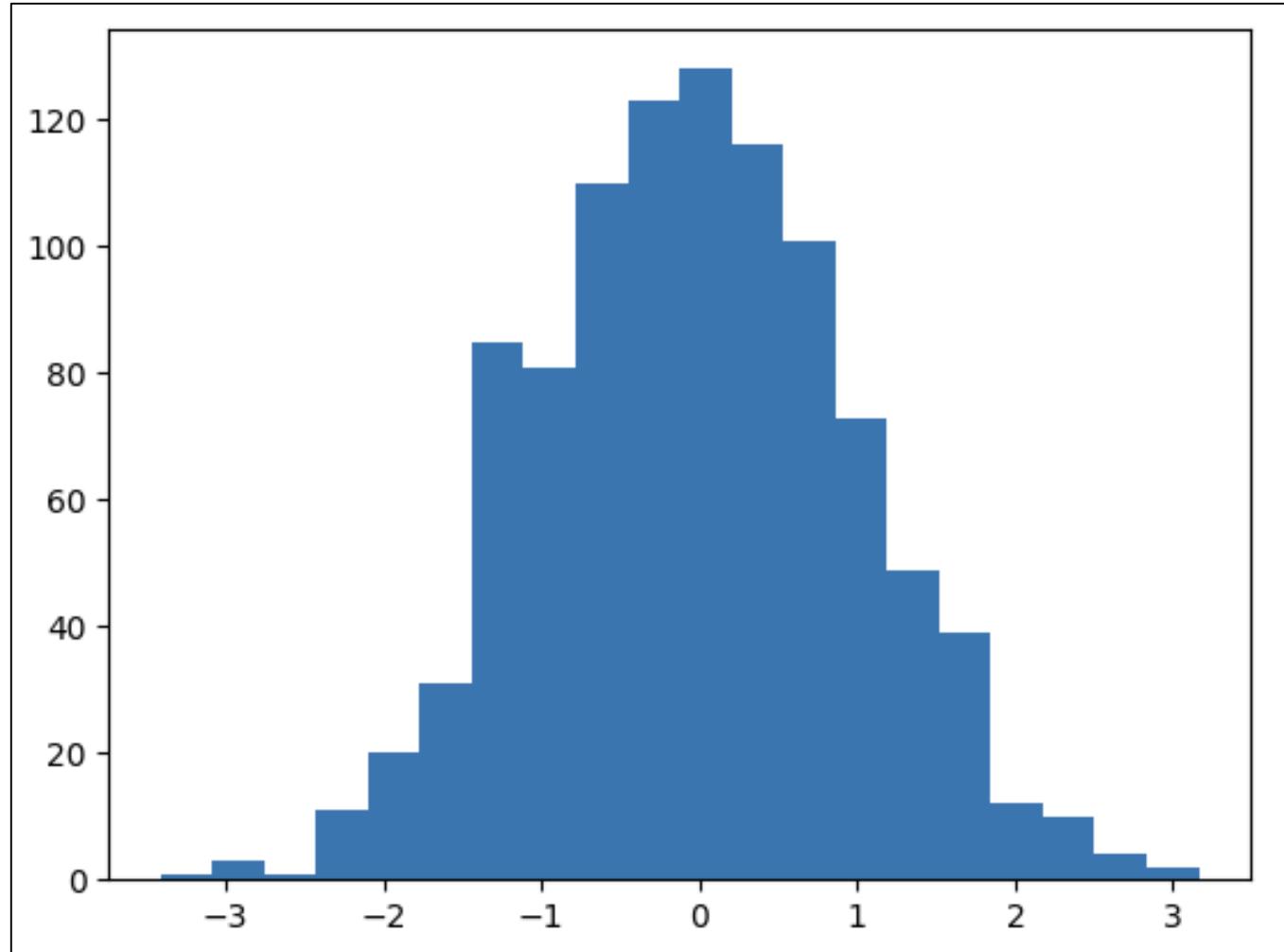
그 외 사용되는 시각화 : 산점도

```
import numpy as np

x = np.random.rand(50)
y = np.random.rand(50)

plt.scatter(x, y)
plt.show()
```

그 외 사용되는 시각화 : 히스토그램



기술 통계량 확인

count: 데이터 개수

mean: 평균

std: 표준편차

min ~ max: 최소값, 최대값

```
# 기술 통계량 요약
summary = df_data.describe()
print(summary)
```

상관계수 분석

상관계수(Correlation Coefficient)

- > 두 변수 간의 관계를 수치로 표현
- > 1에 가까울수록 양의 상관관계
- > -1에 가까울수록 음의 상관관계

```
# 상관계수 분석 (Correlation)
df = pd.read_csv("sample_employee_data.csv")
correlation = df[["Salary", "Age"]].corr()
print(correlation)
```

	Salary	Age
Salary	1.000000	-0.041122
Age	-0.041122	1.000000

employee_data.csv를 활용해
다음의 상관관계 계수를 확인하시오

	Age	Salary	Experience
Age	1.000000	-0.251485	-0.270488
Salary	-0.251485	1.000000	0.273276
Experience	-0.270488	0.273276	1.000000

employee_data.csv를 활용해
다음의 상관관계 계수를 확인하시오

```
employee_data = pd.read_csv("employee_data.csv")
# print(employee_data)
corr = employee_data[["Age", "Salary", "Experience"]].corr()
print(corr)
```

**student_performance.csv를 활용해
다음의 상관관계 계수를 확인하시오**

	Study_Hours	English_Score	Science_Score	Math_Score
Study_Hours	1.00000	-0.206310	-0.282470	-0.302840
English_Score	-0.20631	1.000000	0.214706	0.239768
Science_Score	-0.28247	0.214706	1.000000	0.384893
Math_Score	-0.30284	0.239768	0.384893	1.000000

student_performance.csv를 활용해 다음의 상관관계 계수를 확인하시오

```
student_performance = pd.read_csv("student_performance.csv")
# print(student_performance)
corr = student_performance[["Study_Hours", "English_Score", "Science_Score", "Math_Score"]].corr()
print(corr)
```

**sales_data.csv를 활용해
다음의 상관관계 계수를 확인하시오**

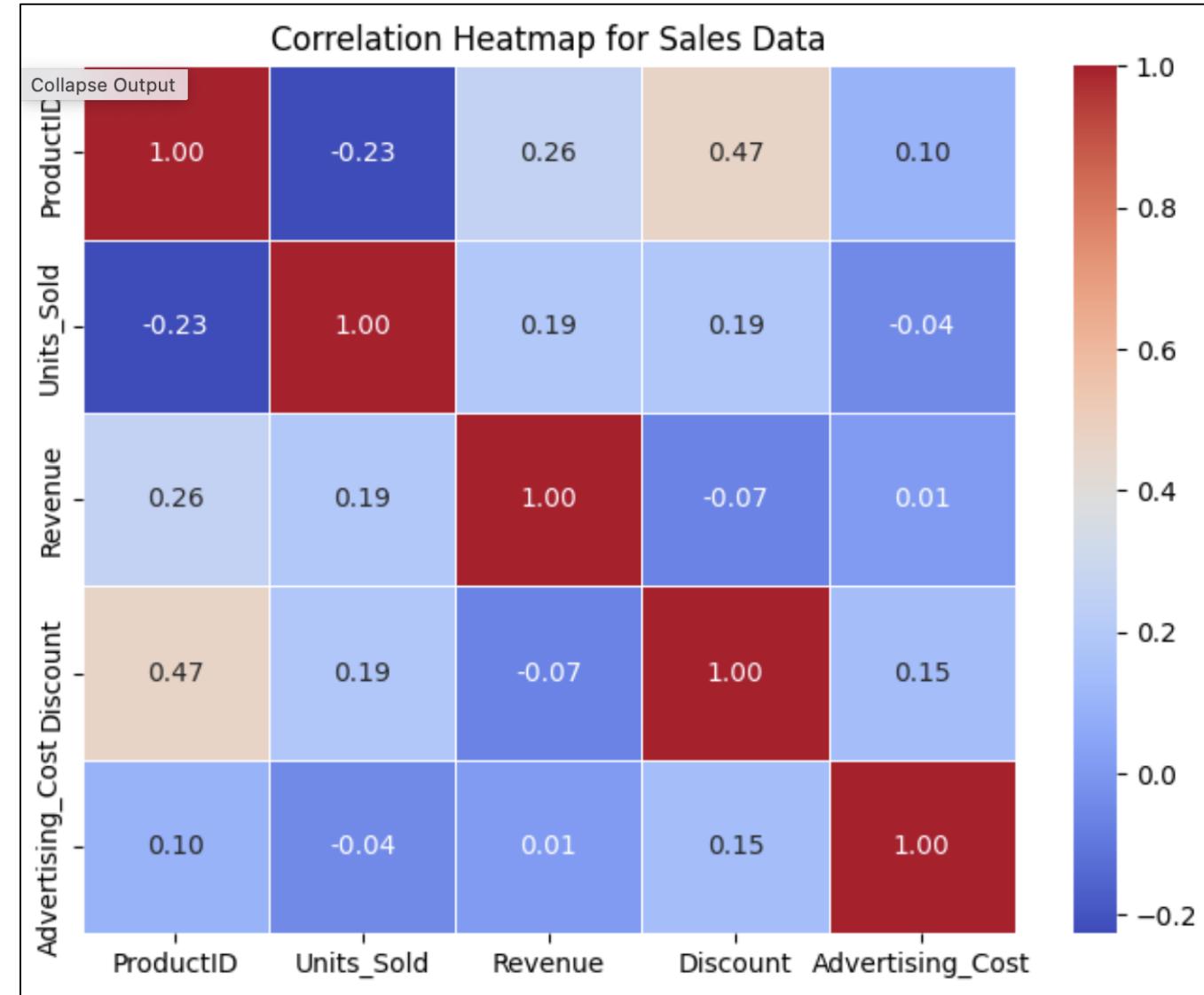
	ProductID	Units_Sold	Revenue	Discount	Advertising_Cost
ProductID	1.000000	-0.225686	0.261634	0.467962	0.095440
Units_Sold	-0.225686	1.000000	0.193679	0.188862	-0.041606
Revenue	0.261634	0.193679	1.000000	-0.065890	0.011998
Discount	0.467962	0.188862	-0.065890	1.000000	0.146099
Advertising_Cost	0.095440	-0.041606	0.011998	0.146099	1.000000

sales_data.csv를 활용해 다음의 상관관계 계수를 확인하시오

```
sales_data = pd.read_csv("sales_data.csv")
# print(sales_data)
corr = sales_data[["ProductID", "Units_Sold", "Revenue", "Discount", "Advertising_Cost"]].corr()
print(corr)
```

seaborn

```
!pip3 install seaborn
```



seaborn

```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6)) # 그래프 크기 설정
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)

plt.title("Correlation Heatmap for Sales Data") # 제목 추가
plt.show() # 그래프 표시
```

seaborn

- 히트맵은 상관관계를 직관적으로 확인할 수 있는 강력한 도구
- 색상으로 쉽게 구분 가능
- 필요에 따라 색상, 레이아웃, 스타일을 커스터마이징할 수 있음

•corr: 상관계수 행렬 (Pandas의 corr() 결과)

•annot=True: 각 셀에 상관계수 값 표시

•cmap='coolwarm': 색상 팔레트 설정

→ (coolwarm = 파란색(음의 상관) ~ 빨간색(양의 상관))

•fmt=".2f": 소수점 두 자리까지 표시

•linewidths=0.5: 셀 경계선 표시

seaborn

진한 빨간색(+) → 강한 양의 상관관계 (값이 1에 가까움)

진한 파란색(-) → 강한 음의 상관관계 (값이 -1에 가까움)

흰색(0 근처) → 상관관계 거의 없음

ProductID <-> Discount: +0.47

→ 빨간색 (중간 정도의 양의 상관관계)

Revenue <-> Advertising_Cost: +0.01

→ 거의 흰색 (상관관계 없음)