진해여자고등학교 AI 창의 융합

독서 프로젝트 활동 강의자료 (7월 18일)

이 자료는 진해여자고등학생을 대상으로 한 코딩 수업에서 실제로 사용되었던 Jupyter Notebook 기반의 활동들을 정리한 것입니다. 학생들이 수학적 개념, 프로그래밍 사고력, 그리고 데이터 분석 및 시각화 도구의 기초 사용법을 함께 익힐 수 있도록 구성되었습니다.

각 활동은 다음과 같은 교육적 목표를 기반으로 설계되었습니다:

- 함수와 도함수의 개념을 실제 문제인 경사하강법에 적용하며 이해한다.
- 유클리드 거리 개념을 통해 데이터 간의 '거리'라는 개념을 직관적으로 받아들인다.
- KNN 알고리즘의 작동 원리를 이해하고 반복적인 계산을 통해 분류 또는 예측이 어떻게 이루어지는지 경험한다.
- 시각화, 수치 연산, 마크다운을 통한 설명을 통합적으로 활용하는 연습을 한다.

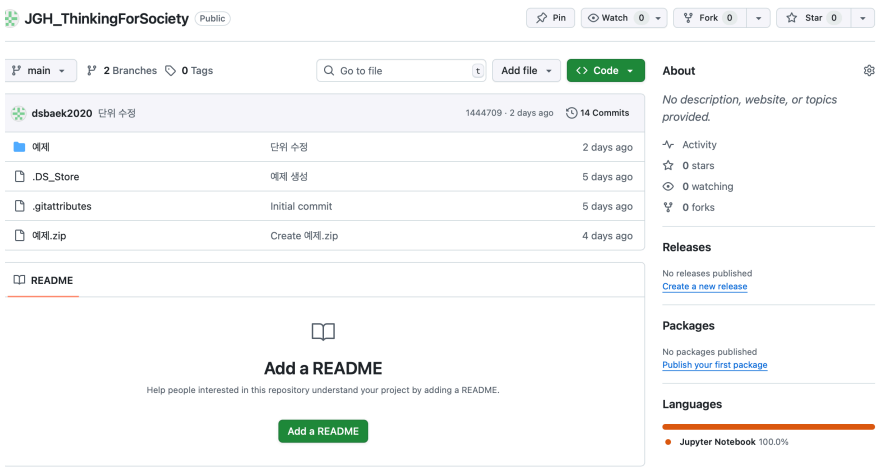
이 자료는 교사가 직접 작성한 마크다운과 Python 코드 셀을 중심으로 구성되어 있으며, 학생 스스로 탐구하고 실습하며 내용을 확장할 수 있도록 설계되어 있습니다.

각 활동은 단편적인 코드 제공이 아닌 알고리즘적 사고 과정과 단계별 구현을 통해 개념을 자연스럽게 습득할 수 있도록 의도되었습니다. 이를 통해 수학, 과학, 컴퓨터 사이의 연계를 느끼고, 실제 데이터에 대한 감각도 익힐 수 있습니다.

실습 코드 및 설명자료는 아래의 깃허브 소스코드 공유 사이트에 공개 하였습니다.

해당자료를 다운받아, 구글 코랩에서 실행하거나, 컴퓨터에 아나콘다 파이썬을 설치하고 주피터랩을 실행하여 실습합니다.

https://github.com/dsbaek2020/JGH_ThinkingForSociety/tree/main



✍ 작성자

백대성
젤리코딩학원(창원) 운영 중.

경사하강법(Gradient Descent)

🎯 목표

우리는 어떤 직선이 주어진 데이터에 가장 잘 맞는지를 찾아보고자 합니다.
즉, **예측값과 실제값의 차이(오차)**를 줄이는 방향으로 파라미터(기울기와 절편)를 조정해 나가는 과정을 공부합니다.
이때 사용하는 대표적인 알고리즘이 바로 **경사하강법**입니다.

1️⃣ 직선부터 시작해 보자!

중학교 때 배운 **직선의 식**을 기억하나요?
y = a * x + b
여기서:

- a: 기울기
- b: y축과 만나는 점 (절편)

이 식은 어떤 **규칙이나 경향**을 설명할 때 아주 많이 사용됩니다.
예를 들어, 다음과 같은 데이터를 생각해 봅시다:

공부 시간 (x)	시험 점수 (y)
1시간	52점
2시간	60점
3시간	68점
4시간	75점

이 데이터를 **하나의 직선으로 표현할 수 있을까요?**

2️⃣ 점들을 가장 잘 통과하는 직선은?

직선은 무한히 많지만, 우리는 점들을 **가장 잘 통과**하는 단 하나의 직선을 원합니다.
예를 들어 아래 두 직선을 비교해보면…

- 직선 A: 거의 모든 점 근처를 지나간다 → 좋은 예측 가능!
- 직선 B: 대부분 점에서 멀리 떨어져 있다 → 예측 정확도 낮음

이때 우리가 사용하려는 **좋은 직선을 찾는 방법**이 바로
오차(Error)를 줄이는 방향으로 직선을 조정하는 방법입니다.

3️⃣ 우리가 찾고 싶은 직선은?

우리가 찾고자 하는 직선은 다음과 같이 씁니다:

y = w * x + b

- w: 기울기 (slope)
- b: 절편 (intercept)

이 직선을 통해
"입력값 x를 주었을 때 y를 예측하는 함수"를 만들고 싶은 것입니다.

4️⃣ 오차(Error)란?

우리가 예측한 값은 **y_hat = w * x + b**이고,
실제 값은 y입니다. 이 둘의 차이를 ****오차(error)****라고 합니다.
데이터가 여러 개 있을 경우, 각각의 오차를 모두 더한 후 평균을 구해줍니다.
이때 많이 사용하는 방법이 **평균제곱오차(MSE)**입니다:
L(w, b) = (1/n) * Σ (w * x_i + b - y_i)^2

- n: 데이터 개수
- x_i: 입력값
- y_i: 실제 출력값

이 함수는 w와 b를 바꿔가며 오차를 계산해주는 함수입니다.

5️⃣ 경사하강법이란?

오차 함수 L(w, b)는 어떤 값에서 **최소가 되는 지점**이 있습니다.
즉, 오차가 가장 작아지는 w와 b의 조합을 찾고 싶은 것입니다.
이걸 찾기 위해 사용하는 방법이 바로 **경사하강법 (Gradient Descent)**입니다.
비유하자면,

"미끄러운 산 위에 공이 놓여 있을 때,
공은 가장 낮은 쪽으로 굴러가게 되죠?"

이때 '내려가는 방향'을 알려주는 것이 바로 **기울기(미분)**이고,
그 기울기를 따라 내려가며 오차를 줄여나가는 것이 경사하강법입니다.

🔁 경사하강법의 수식

경사하강법은 다음과 같은 수식을 반복하여 사용합니다:

w := w - α * ∂L/∂w
b := b - α * ∂L/∂b

α (alpha): 학습률 (learning rate) — 얼마나 크게 이동할지를 결정하는 값입니다.
너무 크면 정확한 지점을 지나칠 수 있고,
너무 작으면 도달하는 데 오래 걸립니다.

📊 데이터를 통해 직선을 찾는 이유는 무엇일까?

우리는 다양한 분야에서 **데이터**를 관측하거나 측정합니다. 이 데이터들을 보면 일정한 **패턴**이나 **추세**가 있을 수 있고, 이런 경향성을 **하나의 직선**으로 표현하면 해석과 예측이 쉬워집니다.

🌌 예시 1: 천문학

천문학자들은 어떤 별의 **거리(x)**와 그 별에서 오는 **밝기(y)**를 측정합니다. 여러 별에 대해 데이터를 수집하다 보면, **거리가 멀수록 밝기가 낮아지는 경향**이 보일 수 있습니다.

- 이때 우리는 "밝기 = 직선(거리)"이라는 관계식을 찾고자 합니다.
- 이 직선은 어떤 법칙이나 자연 현상을 설명해주는 **모델**이 됩니다.

🌍 예시 2: 지질학

지질학자들은 특정 지역의 **지진 횟수(x)**와 **진도(y)**를 수십 년간 기록해 놓습니다. 시간이 흐르며 지진 강도가 서서히 커지는 추세가 있다면?

- 과거 데이터를 직선으로 표현하면 **미래 지진의 강도 예측**에도 도움을 줍니다.
- 직선은 단순하지만 **방향성과 평균적 변화율**을 보여줍니다.

🧃 예시 3: 일상 생활

아침에 마신 커피 양(x)과 하루 집중력 점수(y)를 기록했다고 가정해봅시다. 며칠간 데이터를 모으다 보면 **커피를 많이 마신 날일수록 집중력이 높다**는 경향이 있을 수 있습니다.

- 이때 "집중력 = 직선(커피 양)"이라는 관계를 찾으면 내일 아침 **몇 잔의 커피가 최적인지 예측**할 수 있습니다.

📈 직선이 의미하는 것

직선을 찾는다는 것은 다음을 의미합니다:

- **두 변수 간의 관계**를 단순한 형태로 표현하려는 것
- 변화하는 정도(기울기)와 기준점(절편)을 파악하는 것
- 데이터를 설명하거나, **미래를 예측할 수 있는 수학적 표현**을 만드는 것

📎 이런 과정을 통해 "수많은 점들을 가장 잘 설명해주는 직선"을 찾는 것이 바로
선형 회귀(linear regression)이며,

[16]:
```python
# 데이터 준비
x_data = [1, 2, 3, 4, 5]
y_data = [4, 3, 5.8, 7, 17]

# 초기값
w = 0.0
b = 0.0
learning_rate = 0.01
epochs = 1000

# 경사하강법 학습
for epoch in range(epochs):
    dw = 0.0
    db = 0.0
    n = len(x_data)

    for i in range(n):
        y_pred = w * x_data[i] + b
        error = y_pred - y_data[i]
        dw += 2 * error * x_data[i]
        db += 2 * error

    dw /= n
    db /= n

    w -= learning_rate * dw
    b -= learning_rate * db

    if epoch % 100 == 0:
        print(f"Epoch {epoch}: w = {w:.4f}, b = {b:.4f}")

print(f"\n최종 결과: y = {w:.2f}x + {b:.2f}")
```

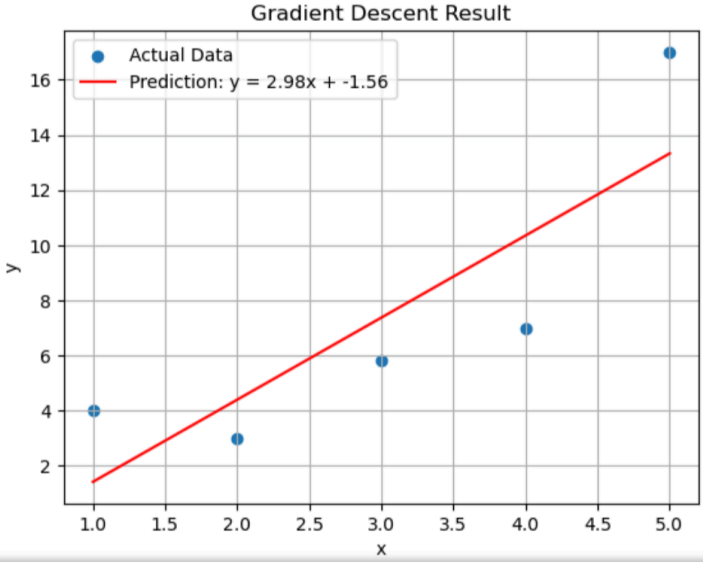
```
Epoch 0: w = 0.5616, b = 0.1472
Epoch 100: w = 2.5485, b = -0.0099
Epoch 200: w = 2.6782, b = -0.4782
Epoch 300: w = 2.7706, b = -0.8120
Epoch 400: w = 2.8365, b = -1.0499
Epoch 500: w = 2.8835, b = -1.2194
Epoch 600: w = 2.9170, b = -1.3402
Epoch 700: w = 2.9408, b = -1.4263
Epoch 800: w = 2.9578, b = -1.4877
Epoch 900: w = 2.9699, b = -1.5315

최종 결과: y = 2.98x + -1.56
```

[17]:
```python
# 그래프 그리기
import matplotlib.pyplot as plt

# 예측값 생성
y_pred = [w * x + b for x in x_data]

# 시각화
plt.scatter(x_data, y_data, label='Actual Data')
plt.plot(x_data, y_pred, color='red', label=f'Prediction: y = {w:.2f}x + {b:.2f}')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Gradient Descent Result')
plt.legend()
plt.grid()
plt.show()
```



✏️ 퀴즈 (Quiz Time!)

1. 경사하강법이란 무엇인가요? 간단히 설명해보세요.

2. 평균제곱오차(MSE)의 수식에서 오차를 제곱하는 이유는 무엇일까요?

3. 학습률(learning rate)이 너무 크면 어떤 문제가 발생할 수 있나요?

부록

✍️ 우리가 알고 싶은 건 "변화율"

함수 `y = f(x)`가 있을 때
x가 조금 변했을 때, y는 얼마나 변할까?

이걸 알려주는 것이 바로 📐 **미분**이야!

📐 예를 들어 볼까? 다음의 2차함수를 보자~

y = x^2
이 함수에서 x가 2에서 2.1로 바뀌면 y는 얼마나 바뀔까?

x = 2 → y = 4
x = 2.1 → y = 4.41
그러면 변화율은?

(4.41 − 4) / (2.1 − 2) = 0.41 / 0.1 = 4.1
이 값은 **x가 2 근처일 때의 평균 기울기**야.
이 간격을 더 작게 만들면 더 정확한 순간 기울기가 나오겠지? 😮

[18]:
```python
import numpy as np
import matplotlib.pyplot as plt

def f(x):
    return x**2  # x의 제곱, ** 은 지수 표현을 위한 프로그래밍의 기호(파이썬에서)

x0 = 2
dx = np.array([0.5, 0.2, 0.1, 0.01, 0.001])
slopes = (f(x0 + dx) - f(x0)) / dx

print("📉 평균 기울기 (x = 2 기준):")
for d, s in zip(dx, slopes):
    print(f"dx = {d:>6} → 평균 기울기 ≈ {s:.4f}")
```

```
📉 평균 기울기 (x = 2 기준):
dx =    0.5 → 평균 기울기 ≈ 4.5000
dx =    0.2 → 평균 기울기 ≈ 4.2000
dx =    0.1 → 평균 기울기 ≈ 4.1000
dx =   0.01 → 평균 기울기 ≈ 4.0100
dx =  0.001 → 평균 기울기 ≈ 4.0010
```

📏 접선(Tangent Line)이란?

어떤 곡선 위의 한 점에서 그 곡선과 딱 맞닿는 **직선**이 있어.
이걸 **접선**이라고 해. 접선의 기울기 = 바로 그 점에서의 **미분값**이야!

함수가 곡선일 땐?

예: `y = x^2`

- x = 2일 때, y = 4
- 이 점에서의 접선 기울기 = f'(2) = 4
  → 즉, 이 점에서의 변화율은 4라는 의미야!

그래서 이 접선은 이렇게 쓸 수 있어:

`y = 4(x - 2) + 4`

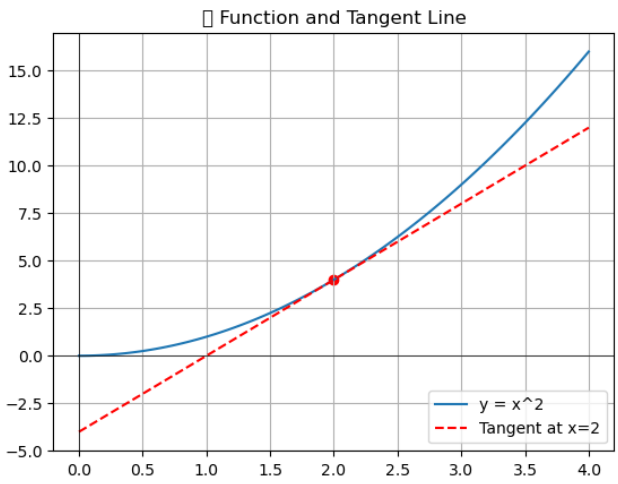
🧠 이건 수학적으로 미분을 통해 얻은 정보지!

[19]:
```python
# 접선과 함수 같이 시각화
x = np.linspace(0, 4, 100)
y = f(x)

x0 = 2
y0 = f(x0)
slope = 2 * x0  # y = x^2 이므로 f'(x) = 2x

tangent = slope * (x - x0) + y0

plt.plot(x, y, label='y = x^2')
plt.plot(x, tangent, '--r', label='Tangent at x=2')
plt.scatter([x0], [y0], color='red')
plt.axhline(0, color='black', linewidth=0.5)
plt.axvline(0, color='black', linewidth=0.5)
plt.title('📐 Function and Tangent Line')
plt.legend()
plt.grid()
plt.show()
```



📘 도함수란?

도함수는 **모든 x값에 대해 순간 변화율을 알려주는 함수**야.

예: `f(x) = x^2` 의 도함수는?

`f'(x) = 2x`
이건 x가 어떤 값이든, 그 지점에서의 기울기를 알려줘!

예시 정리 🎯

x	f(x) = x^2	f'(x) = 2x
1	1	2
2	4	4
3	9	6

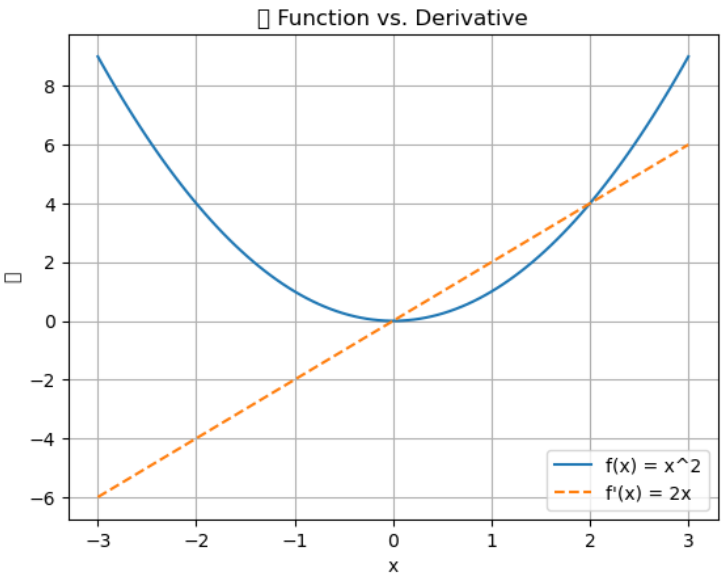
📎 도함수는 **함수의 기울기 정보 전체를 담고 있는 새로운 함수**야!

---

[20]:
```
def df(x):
    return 2 * x  # 도함수

x = np.linspace(-3, 3, 100)
y = f(x)
dy = df(x)

plt.plot(x, y, label='f(x) = x^2')
plt.plot(x, dy, '--', label="f'(x) = 2x")
plt.title('📊 Function vs. Derivative')
plt.xlabel('x')
plt.ylabel('값')
plt.grid()
plt.legend()
plt.show()
```



- 미분은 **한 점에서의 기울기**를 구하는 것
- 접선은 함수 위 한 점에서의 **기울기를 가진 직선**
- 도함수는 **모든 x에 대한 기울기 함수**
- 기울기 = 변화율 = 함수의 민감도

❓ 퀴즈

1. `y = x^2`의 x = 3에서의 접선 기울기는?
2. `f(x) = 3x^2 + 2x`의 도함수는 어떻게 될까?
3. `y = x^2`의 도함수는 증가하는 함수일까, 감소하는 함수일까?

💡 수학은 '변화'를 이해하는 언어야.
이제 함수의 숨겨진 속도를 볼 수 있게 되었어! 🚀

🙏 여러분, 끝까지 집중하며 따라와 주셔서 정말 고맙습니다!

KNN(K-Nearest Neighbors) 알고리즘

🎯 목표¶

우리는 데이터를 보고 **비슷한 패턴이나 속성**을 가진 것들을 분류하거나 추천하는 방법을 배우고자 합니다.
특히, **"가장 가까운 이웃이 누구냐?"**에 따라 결정을 내리는 **K-최근접 이웃(KNN)** 알고리즘의 원리를 익힙니다.

1️⃣ 직관적으로 생각해보자!¶

다음과 같은 상황을 상상해봅시다:

나는 새로운 도시에 이사 왔어요.
이 동네에서 가장 인기 있는 음식은 뭘까?

가장 빠른 방법은?

→ **근처에 사는 사람들**(이웃)에게 물어보는 것이겠죠.
가까운 사람들이 햄버거를 좋아한다면 나도 그럴 가능성이 높습니다.

이게 바로 KNN의 핵심입니다:

"가까운 이웃들이 어떤 선택을 했는가?"에 따라 나의 선택을 결정한다!

2️⃣ KNN이란?¶

KNN(K-Nearest Neighbors)은 다음과 같이 작동합니다:

1. 새로운 데이터가 들어오면
2. 기존 데이터 중에서 **가장 가까운 K개를 고르고**
3. 그들 중 가장 많은 클래스를 보고 **분류** 또는 **추천**합니다.

이 방법은 분류(classification)뿐만 아니라 추천 시스템, 작물 선택, 소비자 분석 등 **현실 문제에 자주 사용됩니다**.

3️⃣ 시각적으로 연습해보자

아래는 단순한 2차원 공간(온도 vs 강수량)에 여러 도시가 존재할 때,
"최근의 기후 조건이 주어졌을 때 어떤 작물을 추천할 것인가?"를
KNN으로 분류해보는 예제입니다.

기본 아이디어는 이렇습니다:

- 도시별로 **기온, 강수량 데이터**를 가지고 있음
- 각 작물은 **선호하는 기후 조건**이 다름
- 과거 데이터를 학습하여, **현재 조건과 가장 비슷한 조건을 가진 도시들**의 작물 추천을 따릅니다

4️⃣ 작물 추천 예시: 왜 KNN이 좋은가?

기존의 회귀 방식은 수학적 식을 이용해 예측합니다.
하지만 KNN은 **"이웃을 직접 보고 결정"**합니다.

예를 들어:

- 올해 진주의 기후가 과거 창원의 2017년과 비슷하다면?
  → 그때 추천되었던 작물을 참고할 수 있습니다.

따라서 KNN은 **"경험 기반"**의 알고리즘이라고 할 수 있으며,
기후 변화처럼 **예측이 어렵고 선형성이 보장되지 않는 문제**에서 특히 유용합니다.

5️⃣ 실제 데이터 예시와 구조

이 노트북에서는 다음과 같은 흐름을 따릅니다:

1. 간단한 2D KNN 분류 시각화 (파란색/빨간색 점들 중 어디에 가까운가)
2. 경남 5개 도시의 15년간 기후 데이터를 생성
3. 각 작물의 기후 선호 조건 정의
4. 각 도시-연도별 작물 점수 계산
5. 가장 적합한 작물을 `BestCrop`으로 레이블링
6. KNN 분류기로 학습 → 최신년도 데이터를 기반으로 작물 추천 실행
7. 결과 시각화 (Temperature vs Rainfall 평면에 추천된 작물 분포)

✨ 핵심 개념 정리

- KNN은 단순하지만, **직관적이며 해석이 쉬운 분류 알고리즘**입니다.
- 비슷한 조건(이웃)을 찾아 분류하기 때문에, **복잡한 모델 없이도 효과적인 결과**를 얻을 수 있습니다.
- 다양한 분야(의료, 농업, 마케팅 등)에서 실제로 활용되고 있습니다.

🧠 생각해볼 질문¶

1. 왜 "가장 가까운 이웃"이 나와 비슷하다고 생각할 수 있을까요?
2. 이웃의 수 K를 3개, 5개, 10개로 바꾸면 결과는 어떻게 달라질까요?
3. 작물 추천 문제에서 KNN보다 회귀(직선 모델)가 더 적절한 상황은 언제일까요?
4. 기후 데이터 외에 어떤 요소를 넣으면 더 나은 추천이 가능할까요?

이제 여러분은 KNN이 단순한 알고리즘이지만,

---

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# 시군 이름
cities = ['Changwon', 'Jinju', 'Miryang', 'Hamyang', 'Geoje']

# 연도
years = np.arange(2010, 2025)

# 지역별 기온 및 강수량 생성 (도시별 편차 존재)
np.random.seed(42)
data = []
for city in cities:
    temp_base = 13.5 + np.random.rand() * 1.5  # 도시마다 온도 베이스 다름
    rain_base = 1000 + np.random.rand() * 200

    for year in years:
        temperature = temp_base + 0.03 * (year - 2010) + np.random.normal(0, 0.1)
        rainfall = rain_base + 5 * np.sin((year - 2010) * 0.4) + np.random.normal(0, 20)
        data.append([city, year, round(temperature, 2), round(rainfall, 2)])

df = pd.DataFrame(data, columns=["City", "Year", "Temperature", "Rainfall"])
df.head()
```

	City	Year	Temperature	Rainfall
0	Changwon	2010	14.13	1220.60
1	Changwon	2011	14.07	1187.41
2	Changwon	2012	14.28	1209.08
3	Changwon	2013	14.10	1205.65
4	Changwon	2014	14.14	1185.83

[2]:
```python
# 작물 종류와 선호 조건 (기온, 강수량 범위)
crops = {
    'SweetPotato':  {'T_opt': 16,  'R_opt': 1000},
    'Apple':        {'T_opt': 14,  'R_opt': 900},
    'Blueberry':    {'T_opt': 13.5,'R_opt': 1100},
    'Peach':        {'T_opt': 15.5,'R_opt': 950},
    'Garlic':       {'T_opt': 13,  'R_opt': 850}
}
```

[3]:
```python
# 작물 선호 점수 계산 함수 (온도와 강수량에서의 거리 기반)
def crop_score(temp, rain, T_opt, R_opt):
    t_score = np.exp(-((temp - T_opt)**2) / 2)
    r_score = np.exp(-((rain - R_opt)**2) / 5000)
    return t_score * r_score

# 작물별 점수 계산
for crop, params in crops.items():
    df[crop] = df.apply(lambda row: crop_score(row['Temperature'], row['Rainfall'], params['T_opt'], params['R_opt']), axis=1)

df.head()
```

[3]:

	City	Year	Temperature	Rainfall	SweetPotato	Apple	Blueberry	Peach	Garlic
0	Changwon	2010	14.13	1220.60	0.000010	1.171100e-09	0.044721	1.707062e-07	6.210766e-13
1	Changwon	2011	14.07	1187.41	0.000138	6.668061e-08	0.184418	4.573867e-06	7.292777e-11
2	Changwon	2012	14.28	1209.08	0.000036	4.845142e-09	0.068297	7.024516e-07	2.784677e-12
3	Changwon	2013	14.10	1205.65	0.000035	7.643697e-09	0.089601	7.898802e-07	5.632969e-12
4	Changwon	2014	14.14	1185.83	0.000178	7.933727e-08	0.186718	5.856504e-06	8.350139e-11

[4]:
```python
from sklearn.neighbors import KNeighborsClassifier

# 각 행마다 가장 적합한 작물 이름을 라벨로 부여 (정답값)
df['BestCrop'] = df[crops.keys()].idxmax(axis=1)

# 학습용 입력 (기온과 강수량만 사용)
X_train = df[['Temperature', 'Rainfall']]
y_train = df['BestCrop']
```

[5]:
```python
# KNN 모델 (k=3 사용)
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

# 예측을 위한 최신년도 기후 데이터
X_test = df[df['Year'] == df['Year'].max()][['Temperature', 'Rainfall']]
cities_test = df[df['Year'] == df['Year'].max()]['City']

# 예측 실행
y_pred = knn.predict(X_test)

# 결과 출력
results = pd.DataFrame({
    'City': cities_test.values,
    'PredictedCrop_KNN': y_pred
})

results
```

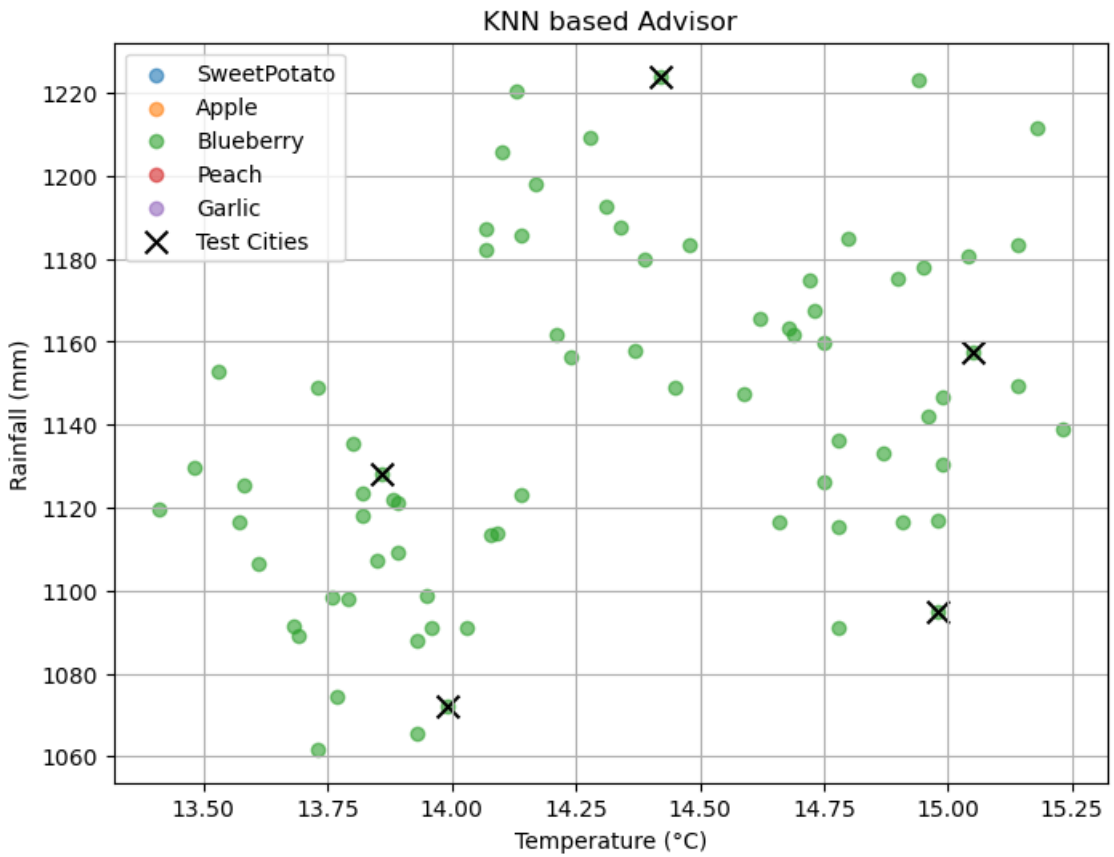
	City	PredictedCrop_KNN
0	Changwon	Blueberry
1	Jinju	Blueberry
2	Miryang	Blueberry
3	Hamyang	Blueberry
4	Geoje	Blueberry

[6]:
```python
plt.figure(figsize=(8, 6))

for crop_name in crops.keys():
    sub = df[df['BestCrop'] == crop_name]
    plt.scatter(sub['Temperature'], sub['Rainfall'], label=crop_name, alpha=0.6)

# 테스트 샘플(최근년도) 표시
plt.scatter(X_test['Temperature'], X_test['Rainfall'], marker='x', color='black', label='Test Cities', s=100)

plt.title("KNN based Advisor")
plt.xlabel("Temperature (°C)")
plt.ylabel("Rainfall (mm)")
plt.grid(True)
plt.legend()
plt.show()
```



🧠 재미있는 질문 3가지: KNN을 탐정처럼 생각해보자!¶

❓ 질문 1. "내 옆집은 왜 중요한가요?"¶

여러분이 전학 간 학교에서, **같은 반 친구들 중 누가 여러분과 비슷한 성향일지** 알아보려면 어떻게 하시겠어요?
그냥 "느낌적으로" 고르기보다는,
나랑 취미, 말투, 앉은 자리, 점심시간 동선이 비슷한 친구를 먼저 찾을 거예요.

Q.
KNN 알고리즘은 이런 상황에서 어떤 식으로 친구를 추천해줄까요?
누가 '가장 가까운 이웃'인지, 어떤 기준으로 판단하나요?

❓ 질문 2. "블루베리 도시의 음모"¶

작년까지만 해도 사과가 잘 자라던 도시에서,
갑자기 블루베리만 추천되고 있어요! 🍇

그 도시의 기후는 **살짝 더워지고, 비가 조금 더 왔다고** 해요.

Q.
KNN은 어떻게 이런 기후 변화에 반응해 작물 추천을 바꿨을까요?
과연 이건 "진짜 기후의 영향"일까요, 아니면 "데이터상의 착시"일까요?

❓ 질문 3. "탐정 K의 의심"¶

KNN 탐정은 새로 등장한 미지의 작물 **'Z토마토'🍅**에 대해 궁금해졌습니다.
그래서 Z토마토가 자란 지역과 기후를 보고,
기존 데이터에 가장 가까운 작물들을 찾으려 해요.

Q.
탐정 K는 어떻게 'Z토마토'의 기후 선호 조건을 추측할 수 있을까요?
그리고 만약 K값을 너무 크게 잡으면, 어떤 실수가 생길 수 있을까요?

🔍 부록: KNN은 어떤 기준으로 이웃을 고를까?

KNN 알고리즘은 이렇게 작동해요:

1. 새로운 데이터가 주어졌을 때
2. 기존 데이터들과의 **유클리드 거리**를 계산하고
3. 그 중에서 **가장 가까운 K개의 점**을 고릅니다.
4. 그 K개 중 가장 많은 라벨(클래스)을 선택하여 **예측**해요!

🎯 중요한 포인트!

- 유클리드 거리 계산이 핵심!
- 피타고라스 정리를 이용해서 "누가 제일 가까운가"를 판단

📐 유클리드 거리(Euclidean Distance)란?

우리는 KNN 알고리즘에서 **어떤 데이터가 가장 가까운 이웃인지**를 찾아야 해요.
이때 사용하는 거리 계산 방법이 바로 **유클리드 거리**입니다.

📏 두 점 사이 거리 공식 (2차원 기준)

거리 $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$
이건 바로 **피타고라스 정리**와 같아요! 🧠

$빗변^2 = 밑변^2 + 높이^2 \rightarrow c^2 = a^2 + b^2$
따라서 유클리드 거리는 바로 **두 점 사이의 직선 거리**를 나타냅니다.

[8]:
```python
#💻 코드 셀 ①: 유클리드 거리 직접 계산해보기
import numpy as np

# 두 점 (x1, y1), (x2, y2)
p1 = np.array([2, 3])
p2 = np.array([6, 6])

# 유클리드 거리 계산
distance = np.sqrt(np.sum((p1 - p2)**2))
print(f"유클리드 거리: {distance:.4f}")
```

```
유클리드 거리: 5.0000
```

[11]:
```python
#🖍️ 코드 셀 ②: 거리 계산으로 가장 가까운 이웃 찾기
# 기존 데이터
data = np.array([[1, 2], [2, 3], [3, 3], [5, 6]])
labels = ['apple', 'apple', 'orange', 'orange']

# 새 데이터 포인트
new_point = np.array([3.5, 4.0])

# 유클리드 거리 계산
distances = np.linalg.norm(data - new_point, axis=1)

# 거리와 레이블 함께 출력
print("새 데이터 포인터로와 기존 데이터의 거리를 계산합니다.")
for i, (d, label) in enumerate(zip(distances, labels)):
    print(f"📌 Sample {i}: {label}, 거리 = {d:.4f}")
```

```
새 데이터 포인터로와 기존 데이터의 거리를 계산합니다.
📌 Sample 0: apple, 거리 = 3.2016
📌 Sample 1: apple, 거리 = 1.8028
📌 Sample 2: orange, 거리 = 1.1180
📌 Sample 3: orange, 거리 = 2.5000
```

🤔 또 다른 알고리즘인 K-Mean 클러스터링과 KNN은 다를까?

둘 다 **데이터 간의 거리**를 이용하지만 방식이 달라요!

구분	KNN	클러스터링 (예: K-Means)
지도 학습 여부	✅ 지도학습 (라벨 필요)	❌ 비지도학습 (라벨 없음)
목적	새로운 데이터를 분류(Classify)	데이터를 그룹으로 나누기
거리 사용	기존 라벨과 가까운 이웃 판단에 사용	중심점과의 거리로 군집 형성
반복 계산	❌ 거리만 1회 계산	✅ 중심점 갱신하며 반복계산 수행

🎯 정리하자면:

- KNN은 분류 알고리즘입니다. (새 데이터가 어떤 클래스인지!)
- 클러스터링은 군집을 스스로 형성합니다. (데이터 구조 파악)

둘 다 **피타고라스 정리로부터 유래한 거리 개념**을 사용해요! 📐

📏 그래서~~^^ 데이터 과학에서 '거리'란?

데이터 과학에서 **'거리(distance)'**라는 말은 단순히 두 장소 사이의 거리만을 뜻하지 않아요.
여기서는 **숫자들로 표현된 데이터들 간의 차이**를 표현하는 데 쓰여요.
좀 더 쉽게 설명할게요. 😊

🧊 숫자로 된 점들, 공간 위에 놓여 있다고 상상해요!

예를 들어, 친구 두 명의 수학 점수와 영어 점수를 가지고 있다고 해봐요:

이름	수학 점수	영어 점수
지민	80	70
유나	85	90

이 점수는 숫자 두 개로 이루어진 데이터예요.
이걸 **(수학, 영어)**처럼 묶어서 생각하면:

- 지민은 (80, 70)
- 유나는 (85, 90)

이렇게 두 점이 되는 거예요.
이 점들은 마치 2차원 좌표평면 위에 놓인 점처럼 생각할 수 있어요!
📈 수학 점수는 x축, 영어 점수는 y축이라고 하면, 두 점 사이의 거리를 생각할 수 있겠죠?

📐 유클리드 거리 (다시한번 정리)

두 점 사이의 거리를 구하는 데 가장 많이 쓰는 방법이 바로 **유클리드 거리**예요.
이건 우리가 중학교 때 배운 **피타고라스 정리**와 똑같아요!

> 유클리드 거리
> $= \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

💻 예제 코드 (Python)

```python
import math

x1, y1 = 80, 70  # 지민
x2, y2 = 85, 90  # 유나

distance = math.sqrt((x2 - x1)**2 + (y2 - y1)**2)
print(f"두 사람의 점수 거리: {distance:.2f}")
```

출력 결과:

```
두 사람의 점수 거리: 20.62
```

즉, 점수 기준으로는 **20.62만큼 떨어져 있다**고 볼 수 있어요.

🤔 왜 이게 중요할까?

데이터 과학에서는 비슷한 데이터를 찾을 때 이 '거리' 개념을 많이 써요.

- 거리 ≈ 데이터 간의 **차이**
- 거리가 가까우면 ≈ **비슷하다**
- 거리가 멀면 ≈ **다르다**

예를 들어 **K-최근접이웃(KNN)** 알고리즘은 거리 개념을 사용해서
가장 가까운 친구들을 찾아서 분류하는 방식이에요.

🧭 정리

개념	실제 의미
유클리드 거리	두 숫자 데이터 사이의 공간상 거리 (피타고라스 정리)
데이터 간 거리	서로 얼마나 비슷하거나 다른지를 나타냄
수의 공간	여러 개의 숫자를 좌표로 생각한 공간 (n차원 공간)

👉 이처럼 **데이터 사이의 거리를 계산하는 것**은,
단순한 수학을 넘어서 우리가 **비슷한 것을 찾고, 분류하고, 예측**하는 데 아주 중요한 역할을 해요.
그리고 그 시작은 **수의 축에서 점들이 얼마나 떨어져 있나?**라는 질문이에요.
"수학 + 공간 + 비교 = 데이터 거리" 🧮📊

💬 수고했어요! 고마워요! 🙌

그리고 K-mean 알고리즘도 도전해 보세요 ^^

[ ]: