인공지능수학. Ⅳ. 최적화	고등학교 학번, 이름:	

0. INTRO

0.1. 소개

인공지능은 자료를 기반으로 합리적인 의사 결정을 내리는 기술을 제공한다. 이번 단원에서는 주어진 자료에 가장 적합한 의사결정을 내리기 위한 함수를 만들고 최적화를 통하여 문제를 해결하는 과정을 배울 것이다.

0.2. 목차

1. 산점도, 추세선

1.1 산점도

자료를 한 눈에 알아보기 위해 시각적으로 표현하는 방법에는 다양한 것들이 있다. 여기서는 **산점도**를 이용해 자료를 표현해 보도록 하자.

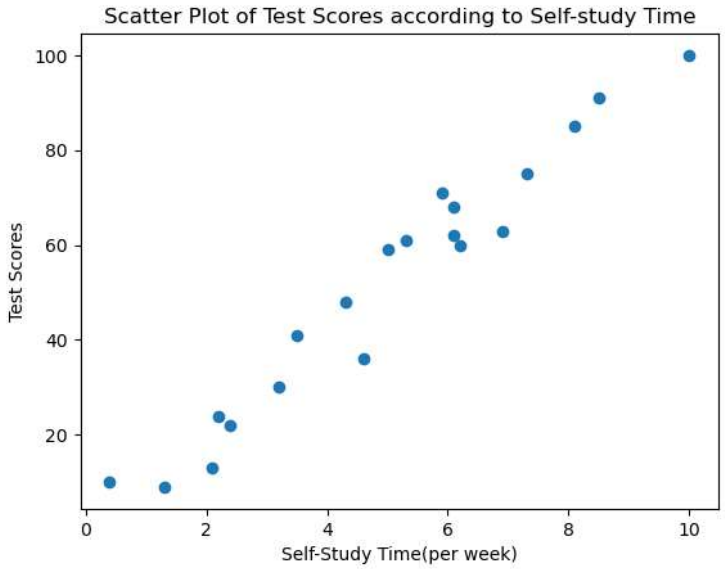
개념

산점도 : 어떤 자료에서 두 변량 x와 y의 순서쌍 (x,y)를 좌표평면 위에 점으로 나타낸 그래프를 x와 y의 산점도라고 한다.

다음은 어느 반 학생 20명의 주당 자습 시간과 그에 따른 시험 점수 자료를 표로 나타낸 것이다.

자습시간	시험점수	자습시간	시험점수	자습시간	시험점수	자습시간	시험점수
0.4	10	3.2	30	5.3	61	6.9	63
1.3	9	3.5	41	5.9	71	7.3	75
2.1	13	4.3	48	6.1	62	8.1	85
2.2	24	4.6	36	6.1	68	8.5	91
2.4	22	5	59	6.2	60	10	100

이 20개의 자료를 산점도로 나타내면 아래와 같다.



〈주당 자습시간에 따른 시험점수 산점도〉

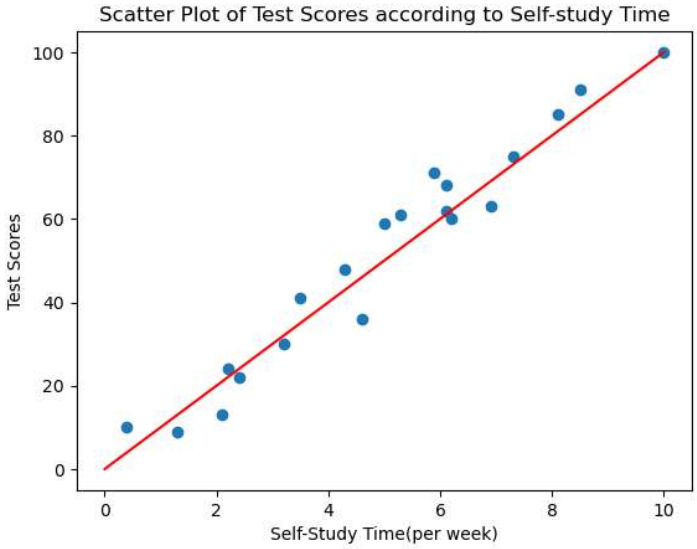
점들이 대체로 한 직선을 중심으로 그 주위에 가까이 분포되어 있음을 알 수 있다. 즉, 자습 시간과 시험점수 사이에 양의 상관관계가 있음을 짐작할 수 있다.

1.2 추세선

위 자료의 산점도에서 두 변량 사이의 상관관계를 판단하려면 점들이 어떤 직선의 주위에 모여 있는지 봐야 한다. 이와 같이 자료의 경향을 나타내는 직선을 **추세선**이라 한다.

개념

추세선 : 자료의 경향성을 나타내는 직선



〈주당 자습시간에 따른 시험점수 산점도와 그 추세선〉

상식(교육과정 외)

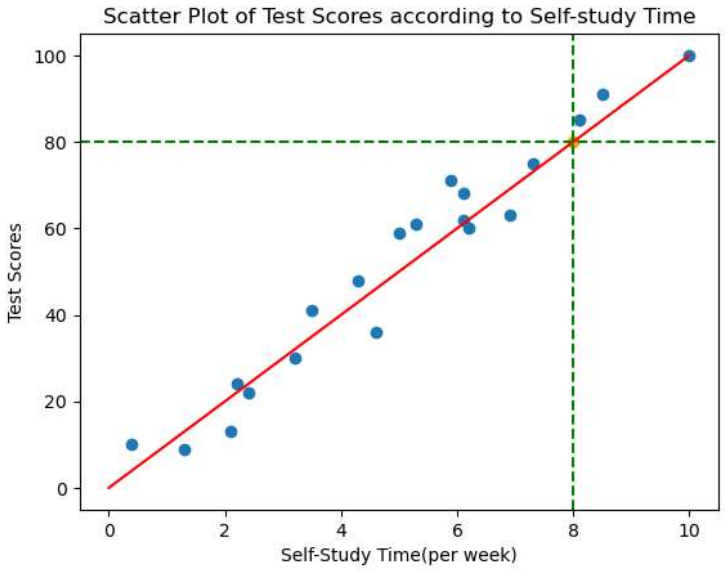
통계학에서는 이와 같이 연속적인 자료로부터 변수들 간의 상관관계를 찾는 것을 '회귀'라고 한다. 특히 그 상관관계가 선형일 경우 이를 '선형회귀'라고 하며 자료를 대표하는 직선, 즉 추세선을 구하는 과정을 자료를 '선형회귀분석 한다'고 이해해도 무방하다.

2. 추세선으로 예측값과 오차 구하기

2.1 예측값

자료의 경향성을 나타내는 추세선을 이용하면 자료에 없는 변량 x에 대응하는 측정값 y를 예측하는 것이 가능하다.

예를 들어, 자료에서 조사한 학생 20명과 동일한 환경에서 공부한 새로운 학생(21번째 학생이라 하자)의 주당 자습시간이 약 8시간이라 하자. 그러면 추세선을 이용해 21번째 학생의 시험성적을 아래와 같이 예측할 수 있다. (편의상, 21번째 학생을 추가하여도 추세선은 변함이 없다고 가정하자)



<추세선을 통한 자습시간이 약 8시간일 때의 시험성적 예측>

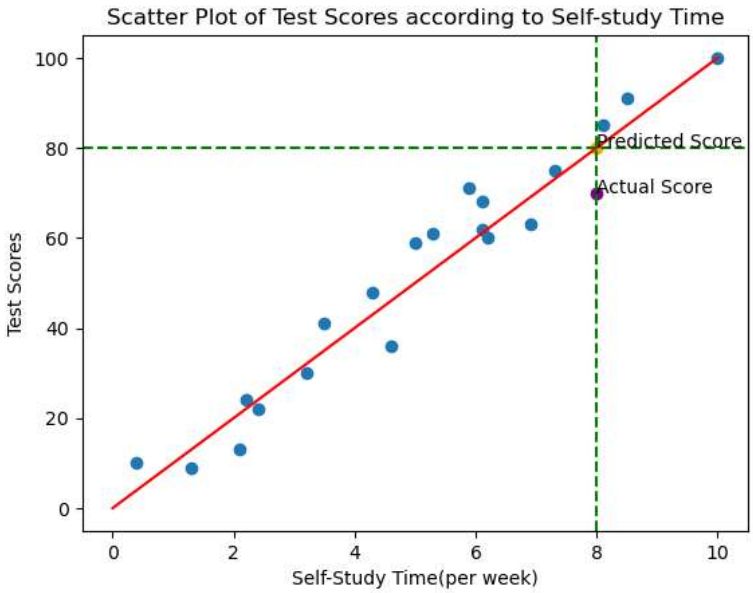
이 경우 추세선을 통해 자습시간이 약 8시간일 경우의 시험성적 예측값은 80점이라고 할 수 있다.

개념

변량 y의 **예측값** : 일반적으로 자료의 두 변량 x, y 사이의 경향을 나타내는 추세선 $f(x)=ax+b$에서 변량 x에 대한 추세선의 값 $f(x)$를 변량 y의 **예측값**이라 한다.

2.2 오차

한편 21번째 학생의 실제 시험성적은 예측값과 얼마든지 다를 수 있다. 예측값과 달리 실제 시험성적은 70점이라고 해 보자. 이를 산점도에 표시하면 아래와 같다. (주황색 점 : 예측값, 보라색 점 : 실제값)



그러면 21번째 학생의 시험성적에 대한 예측은 $70-80$ 즉 -10만큼의 **오차**를 갖는다고 할 수 있다.

개념

변량 x에서의 **오차** : 일반적으로 변량 x에 대응하는 변량 y의 값을 추세선 $f(x)=ax+b$를 이용하여 예측할 때, 다음을 변량 x에서의 오차라고 한다.

(오차) $= y-f(x)= y-ax-b$

또한 산점도에서 두 점 A,B을 잇는 선분의 길이를 곧 **오차의 크기**라고 해도 될 것이다.

개념

변량 x에서의 **오차의 크기** : 변량 x에서의 오차의 절댓값

[문제1] 위 예시에서 22번째 학생의 주당 자습시간이 5.5시간이라고 하자. 실제 시험성적이 58점인 경우 오차와 오차의 크기를 각각 구하시오. (단, 추세선의 식은 $y=10x$로 한다.)

[문제2] 아래 표는 12만원짜리 어느 제품의 사용 기간별 중고 가격을 나타낸 것이다. 사용 기간을 x년, 중고 가격을 y만원이라 하고 추세선의 식을 $f(x) = -2.5x + 11$이라 하자. 중고 가격의 측정값과 예측값 사이의 오차를 구하시오. (즉, 빈칸을 채우시오.)

사용 기간 x년	중고 가격 y만원	예측값 (만 원)	오차(만 원)
1	9		
2	5		
3	4		

3. 추세선 구하기 (선형회귀) 실습

이제, 자료를 대표하는 추세선을 구하는 것이 얼마나 의미 있는 일인지 이해할 수 있을 것이다. 그러면 추세선은 어떻게 구할 수 있을까? 추세선을 구하는 구체적인 원리와 방법은 4절.손실함수에서 배우게 되겠지만 과정이 간단하지 않다. 자료의 크기가 커지면 이 과정은 더욱 번거로울 것이다.

다행히 파이썬 코딩을 이용해 자료를 입력하면 한 번에 추세선을 얻을 수 있다. 이번 절에서 파이썬을 이용해 추세선을 구하는 과정을 실습해 보고 그 다음 절에서 그 원리를 배우고자 한다. 또한 이번 실습 절에 한하여 대학교 전공 용어 (회귀 등)를 일부 사용함을 미리 양해를 구한다.

3.1 모듈 임포트 및 csv 자료 준비

CoLab을 이용한다.

In
```
import matplotlib.pyplot as plt          # 그래프 그리는 모듈 matplotlib.pyplot 임포트
import pandas as pd                      # 표 만드는 pandas 모듈 임포트
import sklearn                           # sklearn을 임포트
sklearn.__version__                      # sklearn 버전조회
```

코드 가져오기: https://cha-record.studio/인공지능수학-4단원-실습코드/

버전이 1.xx 미만이면 아래와 같이 업데이트한다.

In
```
!pip install scikit_learn –user --upgrade
```

이제 추세선을 구할 (선형회귀를 할) 데이터 셋을 csv 파일로 작업공간에 저장하자.

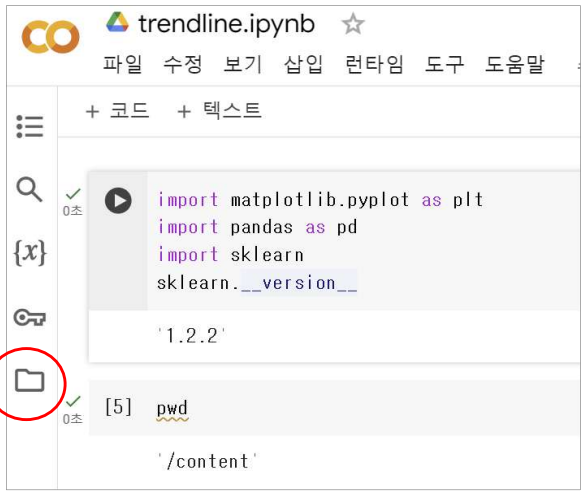
지금까지 썼던 데이터인 아래 표를 data1.csv라는 이름으로 바탕화면에 저장한다. (**2열 21행**짜리 데이터셋으로 저장.)
작업공간(coLab - [왼쪽 폴더모양 아이콘] - [content폴더 우측 동그라미 세 개 클릭] - [업로드]클릭 - 여

기에 data1.csv 업로드)에 'data1.csv'라고 저장)

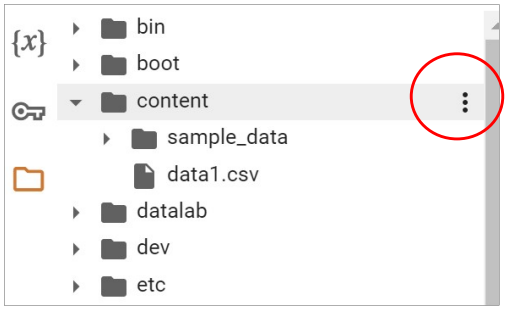
자습시간	시험점수	자습시간	시험점수	자습시간	시험점수	자습시간	시험점수
0.4	10	3.2	30	5.3	61	6.9	63
1.3	9	3.5	41	5.9	71	7.3	75
2.1	13	4.3	48	6.1	62	8.1	85
2.2	24	4.6	36	6.1	68	8.5	91
2.4	22	5	59	6.2	60	10	100

 # 작업공간에 data1.csv 업로드하기

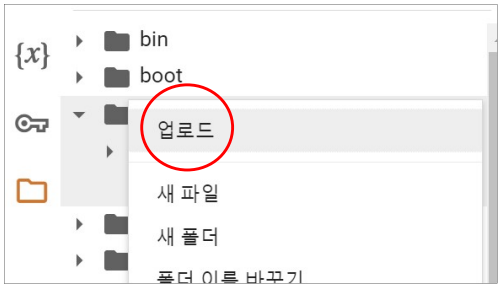
- 구글 코랩의 작업폴더에 파일을 업로드하는 방법은 아래와 같다.



(1단계) coLab 화면에서 왼쪽 폴더 모양 아이콘 클릭



(2단계) content 폴더 우측 동그라미 세 개 모양 버튼 클릭



(3단계) '업로드' 클릭하여 바탕화면에 만들어둔 data1.csv를 업로드

3.2 csv 데이터 읽기, 변수 분리

In

```
dataset = pd.read_csv('data1.csv')                    # 작업공간 내 data1.csv 읽기
dataset.head()                                        # 데이터셋 내 상위 5개 출력
```

Out

	자습시간	시험성적
0	0.4	10
1	1.3	9
2	2.1	13
3	2.2	24
4	2.4	22

데이터를 잘 가져왔음을 확인할 수 있다. 이제 이 데이터셋에서 독립변수는 X라는 변수로, 종속변수는 y라는 변수로 정의할 것이다. 지금은 고등학교 수준에서 살펴볼 수 있도록 일차원의 자료를 다루고 있지만, 원래 독립변수 X는 아래와 같이 다차원인 경우가 일반적이다.

	독립변수			종속변수
	자습시간	이전 시험성적	수면시간	시험성적
학생1	3.6	45	6.5	40
학생2	4.8	67	6	70
학생3	6.7	78	5	81

따라서 일반적으로 독립변수는 대문자 X로, 종속변수는 소문자 y로 나타내며, 데이터셋에서 변수를 분리할 때도 아래와 같이 슬라이싱을 이용한다.

In

```
X = dataset.iloc[:, :-1].values       # dataset의 첫 열부터 마지막 바로 앞 열까지 슬라이싱
y = dataset.iloc[:, -1].values        # dataset의 마지막 열
```

다루기 편하도록 X, y를 각각 리스트로 바꾸려고 한다. 변수명은 각각 xlist, ylist로 하겠다.

In

```
xlist = []
for x in X :
    x = float(x)
    xlist.append(x)

ylist = []
for y_ in y :
    ylist.append(y_)

print("xlist")       # 회색글씨 : 출력을 가로로 하기 위한 코드이니 불필요하면 생략해도 됨.
for x in xlist:
    print(x, end=' ')
print("""
""")
print("ylist")
for t in ylist :
    print(t, end=' ')
```

Out

```
xlist
0.4 1.3 2.1 2.2 2.4 3.2 3.5 4.3 4.6 5.0 5.3 5.9 6.1 6.1 6.2 6.9 7.3 8.1 8.5 10.0

ylist
10 9 13 24 22 30 41 48 36 59 61 71 62 68 60 63 75 85 91 100
```

3.3 추세선 구하기 (선형회귀 모델 생성)

이제 scikit-learn 패키지를 이용해 선형회귀를 해 보자. 정확히는 선형회귀 객체를 생성할 것이다.

In

```
from sklearn.linear_model import LinearRegression      # 선형회귀 모듈 임포트
reg = LinearRegression()                               # 선형회귀 함수 객체 생성
reg.fit(X,y)          # reg에 X,y를 학습시켜 선형회귀 모델 생성
```

코드 가져오기: https://cha-record.studio/인공지능수학-4단원-실습코드/

Out

```
LinearRegression()                    # 학습된 선형회귀 모델.
```

여전히 reg라는 변수는 방금 만들어진 학습된 선형회귀 모델인 LinearRegression()을 가리킴을 잊지 말자. (reg = LinearRegression())

3.4 예측

이제 자습시간 X를 바탕으로 우리가 만든 선형회귀 모델을 통해 시험성적을 예측해 보자.

In

```
y_pred = reg.predict(X)                    #  X에 대한 예측값
y_pred                                     # y_pred 출력
```

Out

```
array([  3.77952965,  13.15774044,  21.49392781,  22.53595123,
        24.61999807,  32.95618544,  36.08225571,  44.41844308,
        47.54451334,  51.71260703,  54.83867729,  61.09081782,
        63.17486466,  63.17486466,  64.21688808,  71.51105203,
        75.67914572,  84.01533309,  88.18342677, 103.81377809])
```

지금까지 얻은 결과를 표로 한 눈에 정리하면 다음과 같다.

In

```
y_pred_list = list(y_pred)                          # 예측값을 리스트로 변환
data_tpl = {'자습시간' : xlist,                      # 표로 만들기 위해 튜플 생성
            '시험성적예측값' : y_pred_list,
           '실제시험성적' : ylist}
# data_tpl 튜플을 표로 생성(자료 개수가 20이므로 index 범위는 range(1,21)으로.)
# 참고 : python에서 range(1,21)이란 1이상 21미만 정수를 뜻함.
df = pd.DataFrame(data_tpl, index=[i for i in range(1,21)])
df
```

Out

	자습시간	시험성적예측값	실제시험성적
1	0.4	3.779530	10
2	1.3	13.157740	9
3	2.1	21.493928	13
4	2.2	22.535951	24
5	2.4	24.619998	22
6	3.2	32.956185	30
7	3.5	36.082256	41
8	4.3	44.418443	48
9	4.6	47.544513	36
10	5.0	51.712607	59
11	5.3	54.838677	61
12	5.9	61.090818	71
13	6.1	63.174865	62
14	6.1	63.174865	68
15	6.2	64.216888	60
16	6.9	71.511052	63
17	7.3	75.679146	75
18	8.1	84.015333	85
19	8.5	88.183427	91
20	10.0	103.813778	100

이를 해석하면, 자습시간이 0.4시간인 1번 학생의 경우 우리가 만든 선형회귀 모델은 시험성적을 3.779점 정도로 예측했지만 실제 시험성적은 10점이라는 뜻이다. 1번,2번,3번 학생을 보면 오차가 상당해 보이지만 4번 학생부터 보면 예측값과 실제 시험성적이 꽤 일치함을 육안으로 관찰할 수 있다.

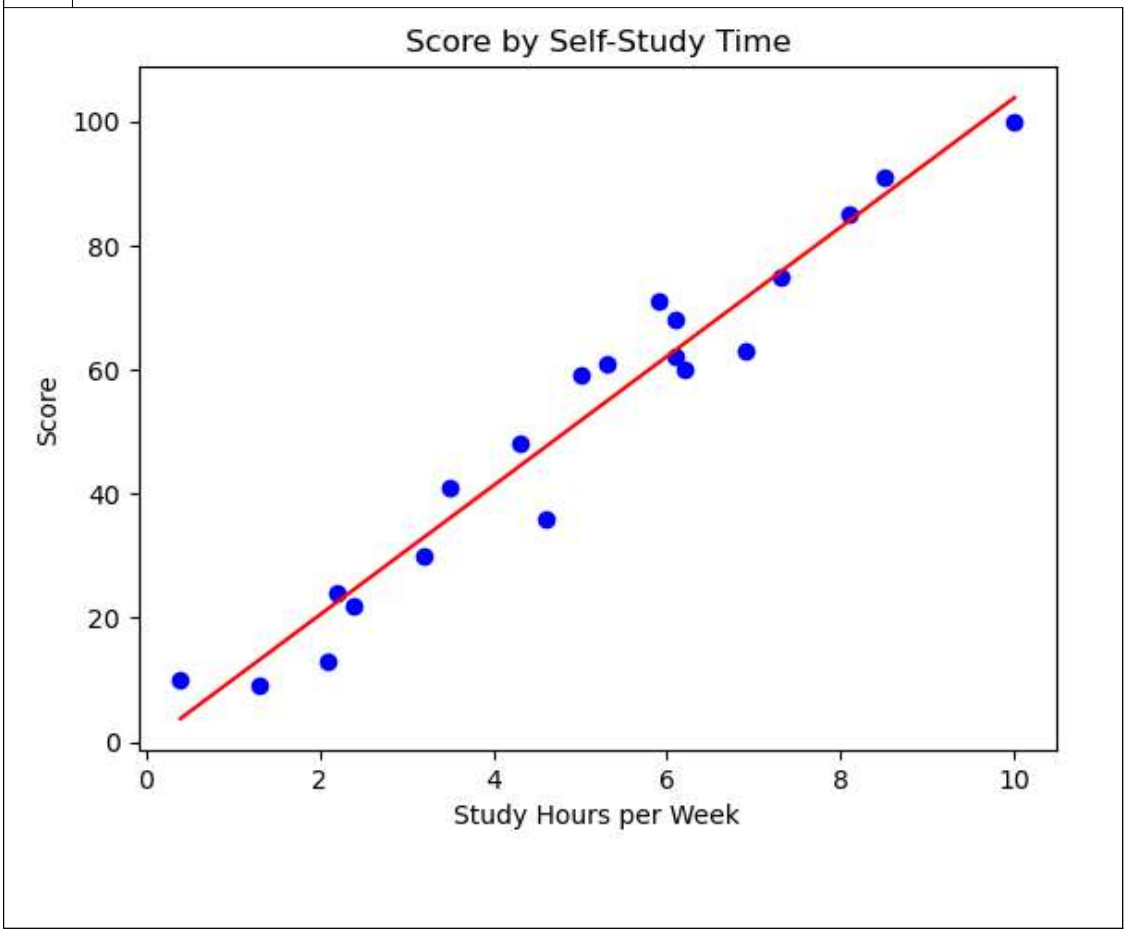
3.5 산점도 만들기 (선형회귀 예측 결과 시각화)

지금까지 작업한 것을 표로 시각화 하여 한 눈에 알아보기 쉽게 하자. 산점도와 추세선을 표현할 것이다.

In

```
plt.scatter(X, y, color = 'blue')          # 독립변수 X, 종속변수 y인 산점도 출력
plt.plot(X, y_pred, color='red')   # 독립변수 X, 예측한 값 y_pred로 꺾은선그래프 출력
                # y_pred가 곧 선형회귀 결과값이므로 꺾은선그래프는 일차함수 그래프가 된다
                # 이것이 추세선이다.
plt.title('Score by Self-study Time')          # plot 제목 생성
plt.xlable('study hours')                  # x축 이름
plt.ylable('score')                        # y축 이름
plt.show()                                 # plot 출력
```

Out



우리가 구한 선형회귀 결과, 즉 추세선이 빨간색 직선으로 나타남을 볼 수 있다. 육안으로 보아 대강 20개의 데이터를 잘 대표하고 있음을 알 수 있다.

3.6 새로 유입된 데이터에 대한 예측

자료에서 조사한 학생 20명과 동일한 환경에서 공부한 새로운 학생(21번째 학생이라 하자)의 공부시간이 6.5시간이라 하자. 이 학생의 시험성적을 우리가 만든 선형회귀 모델 reg=LinearRegression()을 이용해 예측해 보자. (역시 마찬가지로, 편의상 21번째 학생을 추가해도 추세선은 변함이 없다고 가정하자.)

In
```
print('21번째 학생의 예상 시험성적 : ', reg.predict([[6.5]]))
```

Out
```
21번째 학생의 예상 시험성적: [67.34295835]
```

우리가 만든 선형회귀모델에 따르면 (즉, 우리가 구한 추세선을 이용하여 예측하면) 21번째 학생의 시험성적 예측값은 약 67.3점임을 알 수 있다.

그런데 한 가지 아쉬움이 남는다. **우리가 구한 예측값을 신뢰할 수 있는가** 하는 것이다. 이는 곧 우리가 만든 선형회귀모델의 정확도를 어느 정도로 평가할 수 있느냐는 뜻이다.

원래는 이와 같은 의문을 해소하기 위하여, 선형회귀 모델을 생성할 때 데이터셋을 훈련데이터셋과 테스트 데이터셋으로 분리하여 모델을 생성한 후 훈련데이터셋을 통해 모델을 학습시킨 후 테스트 데이터셋을 통하여 모델의 예측 정확도를 평가하는 것이 일반적이지만 고등학교 과정에서 난이도 문제로 데이터셋을 분류하는 과정은 생략하였다.

[문제3] 자신만의 데이터 샘플(크기 30)을 구하여 위 과정을 반복하여 데이터의 추세선을 얻고 31번째의 새로운 데이터에 대한 예측값과 오차를 구해 보자.

4. 손실함수

지금까지 배운 내용에서는 모두 추세선이 이미 주어져 있거나, 또는 직접 만들더라도 컴퓨터 코딩에 의해 만들어진 것이라 추세선이 만들어지는 수학적 원리를 알 수도 없었고, 또 그 추세선이 데이터를 가장 적합하게 나타내는 것인지도 알 수 없었다.

이번 절에서는 추세선을 얻는 수학적 원리에 대해 배운다. 우선 이를 위해 손실함수에 대한 이해가 필요하다.

4.1 두 추세선 중 더 정확한 것은?

여기서는 오차를 구하는 것만으로는 추세선의 정확도를 가늠하기가 어려운 이유를 설명하고자 한다. 먼저 아래 문제를 살펴 보자.

[문제4] 다음은 어느 기계에 넣은 연료 $x\,\mathrm{L}$에 대한 작동 시간 y시간 사이의 관계를 표로 나타낸 것이다.

$x\,(\mathrm{L})$	y
4	6
8	5
12	15
16	12
20	20

이 자료에서 두 추세선 $f(x)=\frac{9}{8}x$, $g(x)=\frac{7}{8}x$ 에 따른 예측 작동 시간과 오차를 각각 구하여 표의 각 빈칸을 채워 보아라. (복습 : (오차) = (측정값) - (예측값))

(1) 추세선 $f(x)=\frac{9}{8}x$인 경우

$x\,(\mathrm{L})$	y	예측시간	오차
4	6	4.5	1.5
8	5		
12	15		
16	12		
20	20		

[표1]

(2) 추세선 $g(x)=\frac{7}{8}x$인 경우

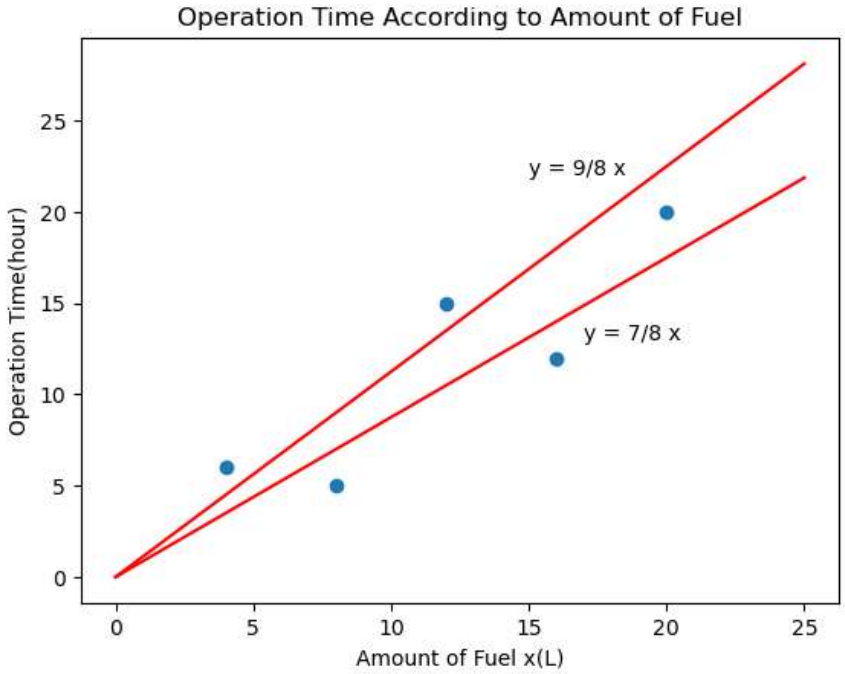
$x\,(\mathrm{L})$	y	예측시간	오차
4	6	3.5	2.5
8	5		
12	15		
16	12		
20	20		

[표2]

두 [표1], [표2]에서, $x=4, x=12$일 때는 추세선 $f(x)=\frac{9}{8}x$의 오차의 크기가 더 작고 $x=8, x=16$일 때는 추세선 $g(x)=\frac{7}{8}x$의 오차의 크기가 더 작다. 또 $x=20$일 때는 두 추세선의 오차의 크기가 같다.

따라서, 추세선의 오차의 크기를 비교하는 것만으로는 어느 추세선이 예측에 더 적합한지 판단하기 어렵다.

실제로 자료와 두 추세선을 그래프로 표현하면 아래와 같은데 육안으로 보아도 어느 추세선이 예측에 더 적합한 것인지 판단하기 어려움을 알 수 있다.



4.2. 평균제곱오차

자료의 추세선 중에서 어느 것이 더 예측에 적합한지 판단하기 위해 일반적으로 오차 대신 **오차의 제곱의 평균(평균제곱오차)**을 사용한다.

개념

오차가 $e_1, e_2, \cdots, e_n$인 데이터에 대해,

평균제곱오차(MSE, Mean Squared Error) $= \dfrac{e_1^2 + e_2^2 + \cdots + e_n^2}{n}$

[문제5] 앞의 두 [표1], [표2]에 대해 오차의 제곱의 평균을 각각 구하고 어느 추세선이 더 예측에 적합한지 구하여라.

4.3 손실함수

지금까지의 활동을 정리하면 **한 자료에서 추세선의 오차의 제곱의 평균(MSE)이 작을수록 더 적합한 추세선**이라 할 수 있다. 따라서 추세선에 따른 평균제곱오차를 그 자료의 추세선에 대한 **손실함수**라 하고 다음과 같이 계산한다.

개념

입력값이 $x_1, x_2, \cdots, x_n$에 대한 측정값이 $y_1, y_2, \cdots, y_n$이고 추세선 $f(x) = ax$ (a는 상수)에 따른 예측값이 $ax_1, ax_2, \cdots, ax_n$이라 할 때, 추세선 $f(x) = ax$에 대한 오차와 손실함수는 다음과 같다.

1) **오차** : $y_1 - ax_1, y_2 - ax_2, \cdots, y_n - ax_n$

2) **손실함수** $\mathrm{L}(a) = \dfrac{(y_1 - ax_1)^2 + (y_2 - ax_2)^2 + \cdots + (y_n - ax_n)^2}{n}$

* 주의 : 손실함수는 추세선의 기울기 a값에 따라 변하는 함수, 특히 이차함수다.

이제 손실함수를 이용하여 추세선의 적합도를 설명할 수 있게 된다.

개념

한 자료의 추세선이 $f(x)=ax+b$일 때,
손실함수 $L(a)$값이 작다 $\Leftrightarrow$ 평균제곱오차가 작다 $\Leftrightarrow$ 추세선 $f(x)$가 자료의 예측에 더 적합

[문제6] 아래는 어느 정류장에 정차하는 서로 다른 노선의 버스 6대의 배차 간격 x분과 각 버스를 기다리는 승객 수 y명 사이의 관계를 표로 나타낸 것이다. 물음에 답하시오.

x분	y명
3	5
6	8
9	11
3	3
6	4
9	5

(1) 추세선 $f(x)=ax$에 대한 손실함수 $L(a)$를 구하시오.

(2) 세 추세선 $f_1(x)=\frac{4}{3}x$, $f_2(x)=\frac{2}{3}x$, $f_3(x)=x$ 중 어느 추세선이 자료의 예측에 더 적합한지 설명하시오.

5. 최적화

앞서 배웠듯이 손실함수의 값이 최소가 되는 추세선이 예측에 적합한 추세선이라 할 수 있다. 이처럼 손실함수의 값을 최소화하는 추세선의 기울기를 찾는 과정을 **손실함수의 최적화**라고 한다. 이번 절에서는 손실함수의 최적화에 관한 두가지 방법을 배운다.

5.1 첫 번째 최적화 : 이차함수 표준형을 이용

어떤 추세선 $f(x)=ax$에 대한 손실함수가 $L(a)=2a^2-12a+20$ 이라고 하자. 이차함수의 표준형을 이용하여 최적화해 보자.

식 $L(a)$을 표준형으로 나타내면,

$$L(a)=2(a-3)^2+2$$

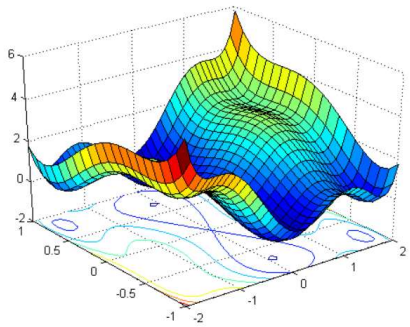
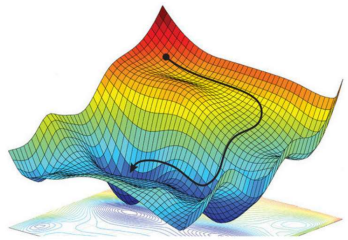
이므로, 손실함수는 $a=3$일 때 최솟값을 갖고 최적 추세선은 $f(x)=3x$이다.

[문제7] 추세선 $f(x)=ax$에 대한 손실함수가 $L(a)=3a^2+6a+8$일 때 최적 추세선 $f(x)$를 구하고 $x=-2$일 때 최적 추세선에 따른 예측값 $f(-2)$의 값을 구하시오.

5.2 두 번째 최적화 : 이차함수보다 복잡한 손실함수의 최적화: 경사하강법

추세선 $f(x)=ax$에 대한 손실함수는 반드시 a에 대한 이차함수이므로 5.1과 같이 이차함수의 성질을 이용해 손쉽게 최적화할 수 있다.

하지만 이차함수보다 복잡한 손실함수를 최적화하려면 조금 더 일반적인 최적화 기법을 사용해야 한다. 지금 소개하려는 방법인 **경사하강법**은 미분을 이용한 최적화 방법이다.

←실제 다양한 손실함수

이해를 돕기 위해, 여기서는 손실함수를 여전히 이차함수로 두고 경사하강법을 설명하겠다.

개념. 경사하강법

추세선 $f(x)=ax$에 대한 손실함수 $L(a)=2a^2-4a+4$라 하자.

1단계. 초깃값 설정

$a=4$를 초깃값으로 정한다.

2단계. 초깃값에서의 손실함수의 미분계수 부호 조사, 손실함수값이 감소하는 방향 찾기

손실함수의 도함수는

$$L'(a)=4a-4$$

이고, 초깃값 $a=4$에서의 미분계수 부호는

$$L'(4)=12>0$$

이므로 $a=4$에서 손실함숫값이 감소하는 방향은 음의 방향임을 알 수 있다.

3단계. 새로운 초깃값 설정

$a=4$의 음의 방향 즉, 4보다 작은 a의 값을 임의로 잡는다. $a=-1$이라 하자.
이와 같이 초깃값을 $a=4$ 대신 $a=-1$로 택하는 것을
'$a=4$**를** $a=-1$**로 이동시킨다**' 고 표현한다.

4단계. 2단계,3단계의 과정을 반복한다.

앞의 과정에 따라 a의 값을 계속 이동시키면 손실함수의 값은 점점 감소하게 되고
실제로 a의 값은 손실함수의 값이 최소가 되는 값 $a=1$에 점점 가까워진다는 것을 알 수 있다.

개념

이와 같이 미분의 성질을 이용하여 손실함수의 값이 감소하는 방향을 찾고 초깃값을 반복적으로 이동시키면서 최적화하는 방법을 **경사하강법(gradient descent algorithm)**이라고 한다.

[문제8] 추세선 $f(x)=ax$에 대한 손실함수가 $L(a)=2a^2-4a+9$이다. 이 손실함수에 대하여 경사하강법을 적용해 초깃값 $a=2$를 이동시키는 과정을 2회 반복한 결과를 구하시오. (열린 문제)

5.3. 경사하강법의 학습률

경사하강법은 초깃값에서 손실함수의 미분계수의 부호(접선의 기울기 부호)를 확인하면 초깃값을 어느 방향으로 이동시킬 것인지 확인할 수 있다. 그러나 초깃값을 얼마나 이동시켜야 하는지를 반복해서 정해야 하므로 계산 과정이 불편하다.

따라서 처음 정한 초깃값을 다음과 같은 방법으로 이동시켜서 좀 더 편리하게 최적화할 수 있다.

개념

학습률r을 이용한 경사하강법 : 아래 과정을 반복하여 초깃값을 이동시킨다. ($L(a)$은 손실함수)

① 적당한 상수 $r\,(r>0)$에 대하여 초깃값 $a=c$를 정한다.

② 초깃값 $a=c$를 새로운 초깃값 $a=c-rL'(c)$로 이동시킨다.

이 때, r의 값을 경사하강법의 **학습률**이라고 한다.

* 미분계수 $L'(c)$가 양수이면 새로운 초깃값 $a=c-rL'(c)$는 기존 초깃값 $a=c$보다 작은 값.
* 미분계수 $L'(c)$가 음수이면 새로운 초깃값 $a=c-rL'(c)$는 기존 초깃값 $a=c$보다 큰 값.

이제 손실함수 $L(a)=2a^2-4a+4$에 대하여 $r=\frac{5}{12}$일 때 경사하강법을 적용하여 초깃값을 이동시켜 보자.

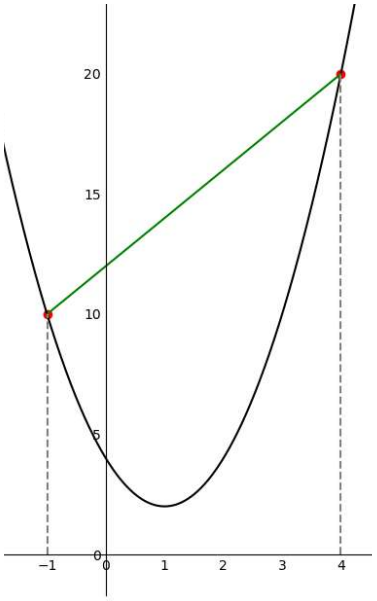
손실함수의 최적화 과정에서 학습률이 너무 크거나 작으면 어떤 문제가 생길지 아래 문제를 통하여 알아보자.

1단계) 초깃값이 $a=4$이면,

$$L'(4)=12>0$$

이므로 새로운 초깃값은 $a=4-\frac{5}{12}\times 12=-1$이다.

즉, $a=4$를 $a=4$보다 작은 $a=-1$로 이동시킨다.

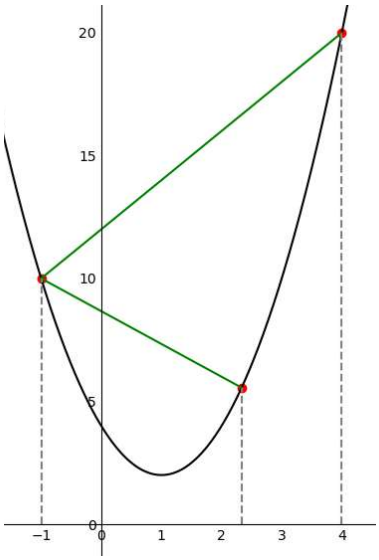


2단계) 새로 설정된 초깃값 $a=-1$에 대하여,

$$L'(-1)=-8<0$$

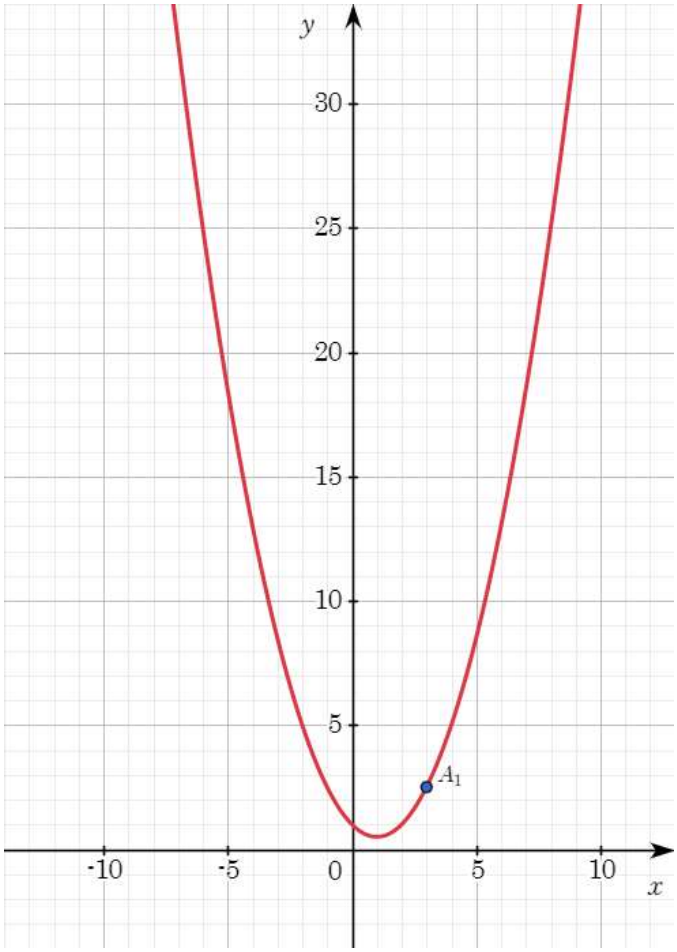
이므로 새로운 초깃값은 $a=-1-\frac{5}{12}\times(-8)=\frac{7}{3}$이다.

즉, $a=-1$을 $a=-1$보다 큰 $a=\frac{7}{3}$로 이동시킨다.

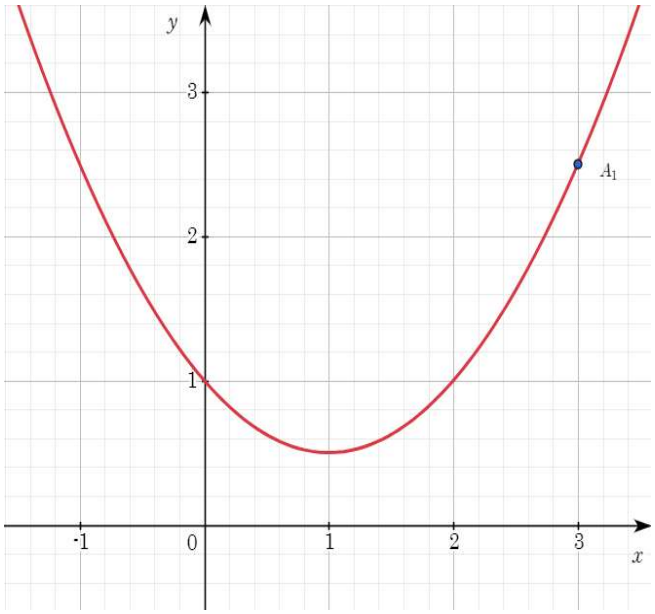


[문제9] 손실함수 $y=\frac{1}{2}x^2-x+1$의 그래프가 아래와 같다고 할 때 물음에 답하시오.

(1) 학습률 r을 3, 초깃값 $a=3$으로 둔 경우 경사하강법을 이용해 초깃값을 이동시키는 과정을 2회 반복하고 오른쪽 그래프 상에 이동시킨 초깃값을 표시하시오.



(2) 학습률 r을 0.1, 초깃값 $a=3$으로 둔 경우 경사하강법을 이용해 초깃값을 이동시키는 과정을 2회 반복하고 오른쪽 그래프 상에 이동시킨 초깃값을 표시하시오.



(3) 학습률이 너무 크거나 작으면 어떤 어려움이 있는지 자유롭게 말해 봅시다.

5.4 경사하강법 실습

경사하강법은 시각적으로 확인하여 직관적으로 이해하는 것이 큰 도움이 될 수 있다. 아래 웹앱에 접속하여 초깃값과 학습률을 임의로 입력하면서 경사하강법을 역동적으로 이해해 보자.

도구 바로가기: https://cha-record.studio/인공지능수학-경사하강법실습/

5.5. 프로그래밍으로 경사하강법 구현하기

아래 과정은 프로그래밍을 통해 경사하강법을 구현하는 과정이다. 프로그래밍 역량이 있는 학생은 따라해 보도록 하자. (코드 가져오기: https://cha-record.studio/인공지능수학-4단원-실습코드/)

실습1. CoLab을 이용한 경사하강법 시뮬레이션

1단계. 패키지 임포트 및 독립변수와 종속변수 분리

In

```
import numpy as np                              # 필요한 패키지 임포트
import matplotlib.pyplot as plt
import pandas as pd

data_set = pd.read_csv('data1.csv')             # 작업공간 내 data1.csv 읽기

X = data_set.iloc[:, :-1].values                # 독립변수 분리
y = data_set.iloc[:, -1].values                 # 종속변수 분리
```

2단계. 다루기 쉽도록 독립변수값, 종속변수 값을 담은 각각의 리스트 xlist, ylist 생성

In

```
xlist = []
for x in X :
    x = float(x)
    xlist.append(x)

ylist = []
for y_ in y :
    ylist.append(y_)

xlist, ylist
```

아래와 같이 간단한 리스트가 만들어졌음을 볼 수 있다.

Out

```
([0.4,  1.3,  2.1,  2.2,  2.4,  3.2,  3.5,  4.3,  4.6,  5.0,  5.3,  5.9,  6.1,  6.1,
6.2,  6.9,  7.3,  8.1,  8.5,  10.0],
 [10,  9,  13,  24,  22,  30,  41,  48,  36,  59,  61,  71,  62,  68,  60,  63,  75,
85,  91,  100])
```

3단계. 손실함수 만들기

In

```
import sympy as sy              # 식을 다루는 데 적합한 패키지 sympy 임포트
x = sy.symbols('x')             # x가 식을 구성하는 변수임을 선언
a=0
for x_, y_ in zip(xlist, ylist) :
    a += (y_ - x*x_)**2         # 오차의 제곱 (y_k − x_k × x)^2을 차곡차곡 a에 더함

a = a / len(xlist)              # 이를 자료 개수만큼 나눔. 따라서 a는 평균제곱오차.

loss_func = sy.expand(a)                # a를 x에 대해 내림차순으로 정리한 식을
                                        # loss_func 변수에 저장. 즉, 이것이 손실함수
print('loss_func:', loss_func)          # 손실함수 출력

c2 = loss_func.coeff(x,2)               # 손실함수의 최고차항 계수를 c2라고 함
c1 = loss_func.coeff(x,1)               # 손실함수의 최고차항 계수를 c1라고 함
c0 = loss_func.coeff(x,0)               # 손실함수의 최고차항 계수를 c0라고 함
```

실제로 아래와 같이 손실함수가 출력됨을 볼 수 있다.

Out

```
loss_func: 30.926*x**2 - 640.65*x + 33501/10
```

이제 경사하강법을 단계적으로 시뮬레이션해보려고 한다. 코드는 아래와 같다. (시뮬레이션 기회는 20번. 도중에 'q'를 입력하면 멈추도록 설계하였다.)

In

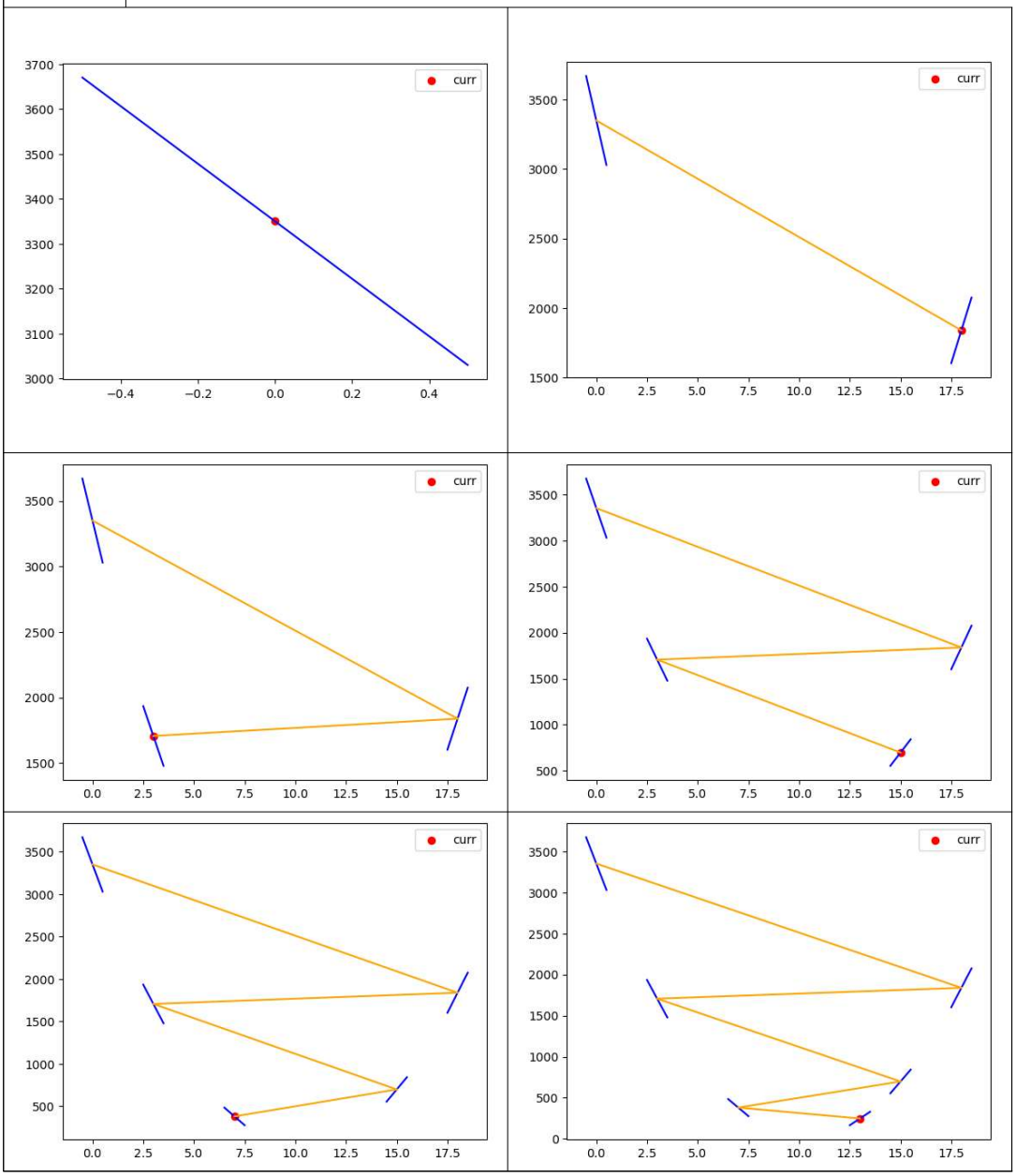
```
opp = 20                                                                    1
alist = []                                                                  2
curr_points = []                                                            3
while opp>0 :                                                               4
    m = input("추세선 기울기 a 값을 입력 (멈추려면 q): ")                    5
    if m == 'q' :                                                           6
        break                                                               7
    m=float(m)                                                              8
    alist.append(m)                                                         9
```

```
    x = np.array(range(-5,11))                                                                  10
    for m in alist :                                                                            11
        plt.plot([m-0.5, m+0.5],                                                                12
        [c2*m**2+(c1-c2)*m -0.5*c1 +c0, c2*m**2+(c1+c2)*m+c0+0.5*c1], color='blue')              13
    mlist = [alist[-1]]                                                                         14
    x = np.array(mlist)                                                                         15
    plt.scatter(x, c2*x**2 + c1*x +c0, color='red', label = 'curr')                             16
    curr_points.append([x,c2*x**2+c1*x +c0])                                                    17
#     print(opp, curr_points)                                                                   18
    if opp <= 19 :                                                                              19
        for a in range(0, len(curr_points)-1) :                                                 20
            plt.plot([curr_points[a][0], curr_points[a+1][0]],                                  21
                    [curr_points[a][1], curr_points[a+1][1]], color = 'orange')                 22
    plt.legend()                                                                                23
    plt.show()                                                                                  24
    opp -= 1                                                                                    25
```

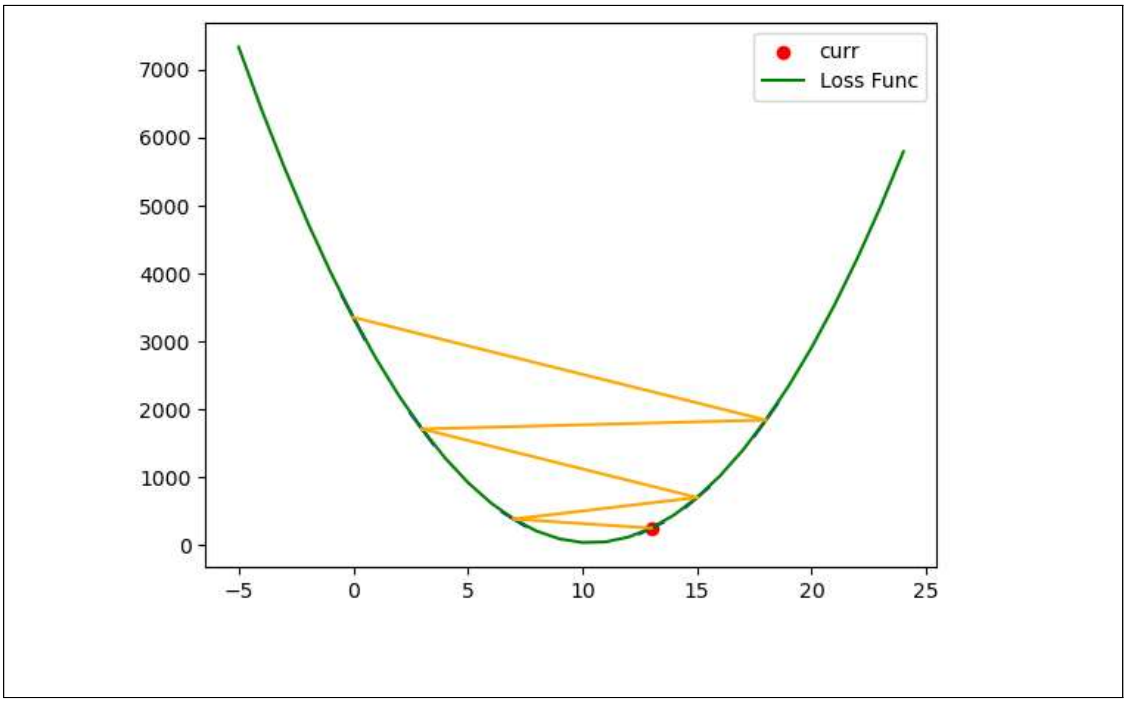
* 12,13번째 줄 코드는 줄바꿈 없이 이어 써야 한다.

단계별 결과는 아래와 같다. (아래 예시의 경우 손실함수 기울기 값을 차례로 0, 18, 3, 15, 7, 13 으로 입력하였다.)

Out



이 데이터셋의 경우 추세선 기울기 값 a가 7.5, 12.5 사이인 어느 지점에서 최적화됨을 알 수 있다. 지금까지의 시뮬레이션을 손실함수 그래프와 함께 표현하면 아래와 같다.



Jupyter Notebook에서 실습하려면 아래 코드를 따르면 된다.

실습2. Jupyter Notebook을 이용한 시뮬레이션
- 추세선의 기울기값 a와 손실함수, 그에 따른 산점도와 추세선

In

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

data_set = pd.read_csv('data1.csv')

X = data_set.iloc[:, :-1].values
y = data_set.iloc[:, -1].values
```

In

```
xlist = []
for x in X :
    x = float(x)
    xlist.append(x)

ylist = []
for y_ in y :
    ylist.append(y_)

xlist, ylist
```

Out

```
([0.4,  1.3,  2.1,  2.2,  2.4,  3.2,  3.5,  4.3,  4.6,  5.0,  5.3,  5.9,  6.1,  6.1,
6.2,  6.9,  7.3,  8.1,  8.5,  10.0],
 [10,  9,  13,  24,  22,  30,  41,  48,  36,  59,  61,  71,  62,  68,  60,  63,  75,
85,  91,  100])
```

In

```
import sympy as sy
x = sy.symbols('x')
a=0
for x_, y_ in zip(xlist, ylist) :
    a += (y_ - x*x_)**2
a = a / len(xlist)

loss_func = sy.expand(a)
print('loss_func:', loss_func)

c2 = loss_func.coeff(x,2)
c1 = loss_func.coeff(x,1)
c0 = loss_func.coeff(x,0)
```

여기까지는 앞 시뮬레이션과 같다.

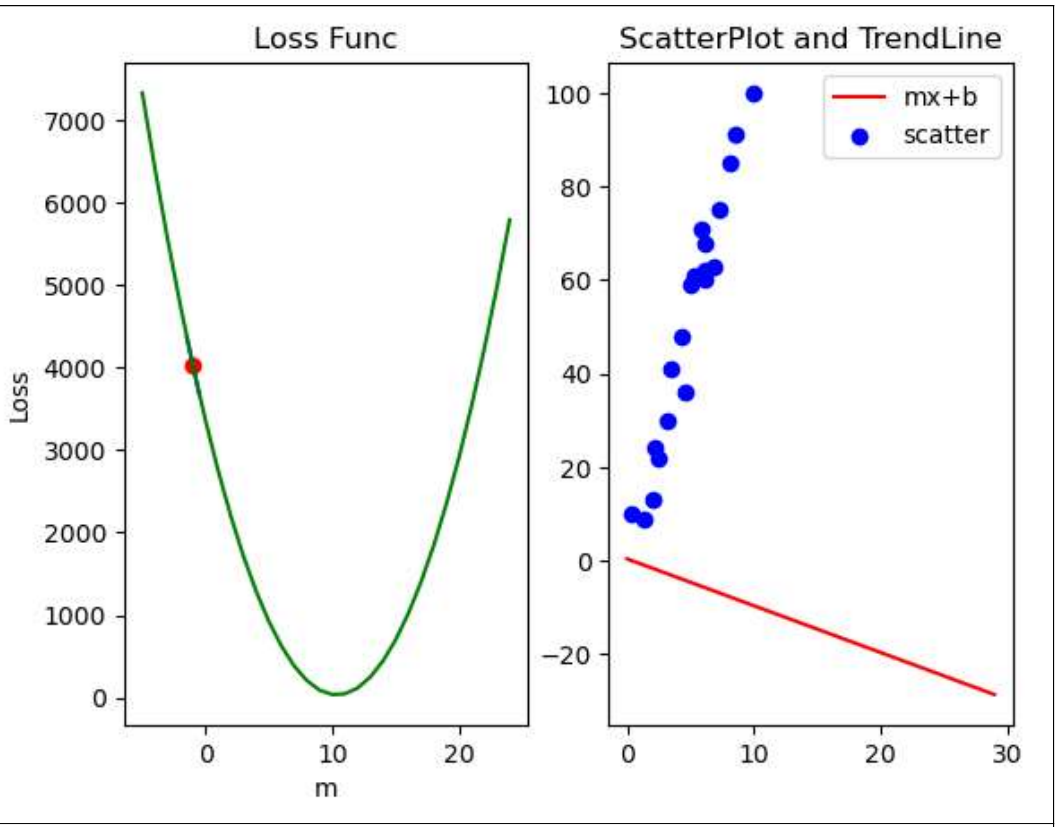
In

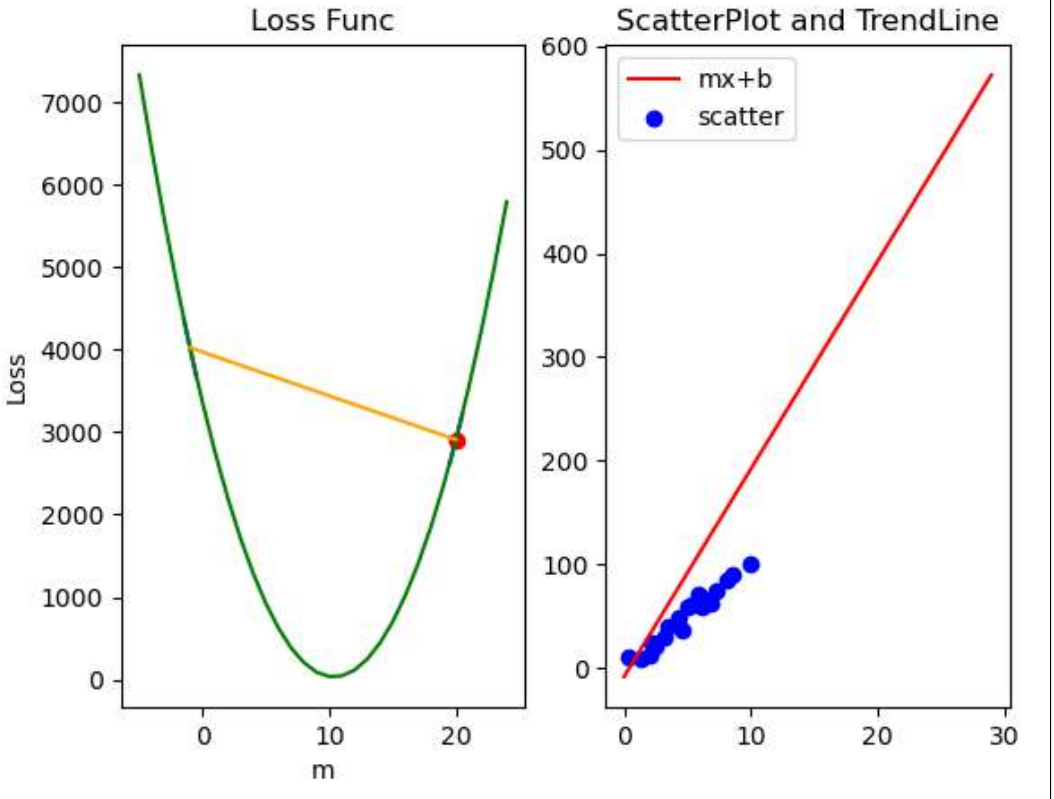
```
# 모드 구현 : 경사하강법 없이 그냥 손실함수와 산점도만 보고 싶으면 'yet'을,
# 경사하강법 시뮬레이션을 시작하려면 's'를 입력하면 되도록 모드 구현
mode = input("""Input 'yet' to see the Loss Func Graph and Data ScatterPlot,
or Input 's' to start GRADIENT DESCENT """)
# 하다가 잘 안되면 https://chapenguin.com/ 과제제출 게시판에 문의바랍니다.
# Copyright by 차민경 (코드 무단 복사 및 배포 금지. 출처 표기하여 공유 가능)

if mode == 'yet' :
```

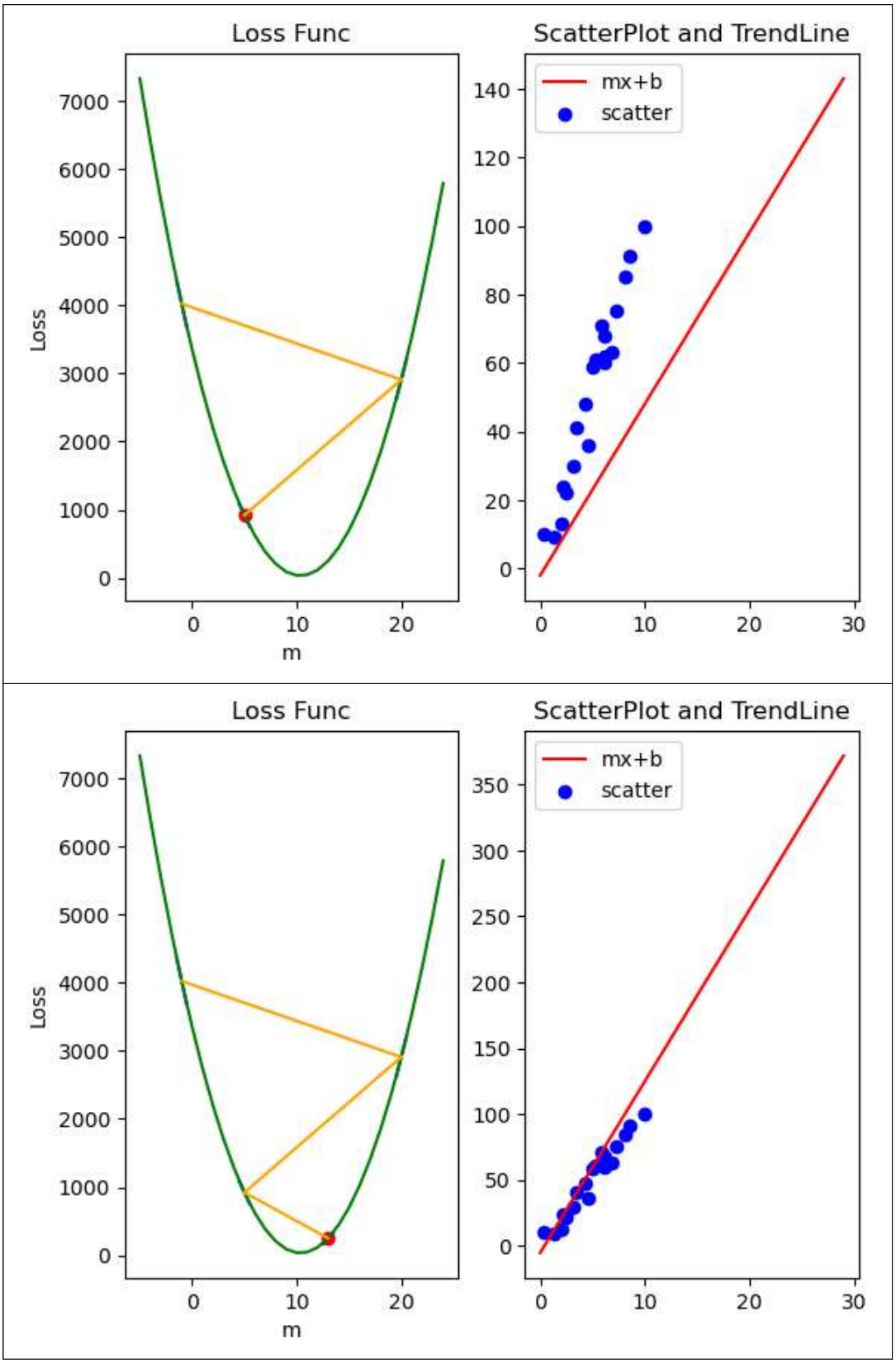
```
    plt.subplot(121)                                                                    9
    x = np.array(range(-7,20))                                                          10
    plt.plot(x, c2*x**2 + c1*x + c0, color='green', label='Loss Func')                  11
    plt.title("Loss Func")                                                              12
    plt.xlabel("m")                                                                     13
    plt.ylabel("Loss")                                                                  14
                                                                                        15
    plt.subplot(122)                                                                    16
    plt.scatter(X, y, color='blue', label='scatter')                                    17
    x = np.array(range(-5,25))                                                          18
    plt.title("ScatterPlot and TrendLine")                                              19
    plt.legend()                                                                        20
    plt.show()                                                                          21
                                                                                        22
elif mode =='s' :                                                                       23
                                                                                        24
    opp = 20                                                                            25
    alist = []                                                                          26
    curr_points = []                                                                    27
                                                                                        28
    while opp > 0 :                                                                     29
        m=input("Input the value m of the Loss Function L(m) : ")                       30
        if m == 'q' :                                                                   31
            break                                                                       32
        m = float(m)                                                                    33
        alist.append(m)                                                                 34
                                                                                        35
        plt.subplot(121)                                                                36
        x = np.array(range(-7, 25))                                                     37
        for m in alist :                                                                38
            plt.plot([m-0.5, m+0.5],                                                    39
[c2*m**2+(c1-c2)*m -0.5*c1 +c0, c2*m**2+(c1+c2)*m+c0+0.5*c1], color='blue')              40
        # 37,38 줄은 줄바꿈하지 말고 붙여 써야 함.                                      41
        mlist = [alist[-1]]                                                             42
        x = np. array(mlist)                                                            43
        plt.scatter(x, c2*x**2 + c1*x + c0, color='red', label = 'curr')                44
        curr_points.append([x, c2*x**2+c1*x + c0])                                      45
        x = np.array(range(-5,25))                                                      46
        plt.plot(x, c2*x**2 + c1*x + c0, color='green', label='Loss Func')              47
        if opp <= 19 :                                                                  48
            for a in range(0, len(curr_points)-1) :                                     49
                plt.plot([curr_points[a][0], curr_points[a+1][0]],                      50
                        [curr_points[a][1], curr_points[a+1][1]], color='orange')       51
        opp -= 1                                                                        52
```

```
        plt.title("Loss Func")
        plt.xlabel("m")
        plt.ylabel("Loss")

        plt.subplot(122)
        x = np.array(range(0,30))
        x1 = xlist[0]
        y1 = ylist=[0]
        plt.plot(x, m*x+ y1 - m*x1, label='mx+b', color = 'red')


# 여기서는 코딩의 편의상 추세선이 반드시 첫 번째 데이터를 나타내는 점을 지나도록
# 설계하였다. 실제 추세선은 반드시 첫 번째 데이터 점을 지난다는 보장이 없다.
# 하지만 실제 방식대로 추세선을 구현하려면 epoch 등 교육과정 외 내용이 필요하여
# 편의상 단순화하였으니, 참고하도록 하자.


        plt.scatter(X, y, color='blue', label='scatter')
        plt.title("ScatterPlot and TrendLine")

        plt.legend()
        plt.show()
```

모드값 (yet or s)과 손실함수 기울기값을 조정해 가며 손실함수 기울기 값에 따른 추세선을 시각적으로 확인할 수 있다. 아래는 모드를 s로 택하고 단계적으로 경사하강법을 실시한 결과다.


Loss Func
Loss
m
ScatterPlot and TrendLine
mx+b
scatter


Loss Func
Loss
m
ScatterPlot and TrendLine
mx+b
scatter



왼쪽의 손실함수 그래프 상에서 경사하강법을 반복하여 손실이 최소화되는 점으로 이동할수록 오른쪽 산점도에서 추세선이 자료(파란색)를 더 적합한 방향으로 대표하게 됨을 볼 수 있다.

자료 제작 및 문의 : 교사 차민경 (https://cha-record.studio/)
자료 출처 : 미래엔 인공지능수학 교과서 (2015개정교육과정)