

파이썬을 이용한 Linear Regression 문제 해결

Using Python to solve the linear regression problem

2021. 12.

임창진

인하대학교 물리학과



이학학사학위 논문

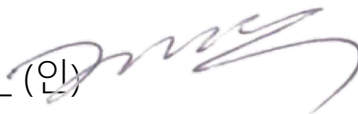
파이썬을 이용한 Linear Regression 문제 해결

Using Python to solve the linear regression problem

2021. 3.

임창진

인하대학교 물리학과

지도교수 : \_\_\_\_\_ (인) 

논문을 이학학사 학위 논문으로 제출함.

이 논문을 ○ ○ ○의 이학학사 학위논문으로 인정함.

년 월 일

논문사정위원장 (인)

논문사정위원 (인)

논문사정위원 (인)

논문사정위원 (인)

# 초 록

본 논문은 기초적인 분석방법인 Linear regression에 대한 이해도 향상을 목적으로 작성되었다. 오픈 소스 프로그래밍 언어임과 동시에 AI 분야에서 가장 각광받고 있는 Python을 활용해 자체적인 코드를 구현하면서 기존에 있던 머신 러닝 라이브러리에 대한 이해도를 높일 수 있었다. 또한 임의로 만들어진 데이터 셋과 실제로 사용될 수 있는 가상의 데이터 셋을 코드에 적용해 봄으로써 Linear regression의 현실 세계에서의 가치와 Linear regression이 프로그램 속에서 실행되는 메커니즘에 대하여 이해할 수 있었다. 또한 하나의 변수를 이용한 Linear Regression의 한계를 확인할 수 있었다.

# 목 차

## 제 1 장 서론

### 1-1.연구의 목적

### 1-2.연구에서 사용된 프로그램

## 제 2 장 이론

### 2-1.상관계수

### 2-2.Linear Regression

### 2-3.결정계수

## 제 3 장 코드

### 3-1.구현한 코드 분석

## 제 4 장 결론

### 4-1.분석 결과

### 4-2.고찰

## 참고문헌



# 제 1 장 서론

## 1-1 연구의 목적

근래 기술의 발전에 있어서 가장 핫한 키워드라고 한다면 데이터를 빼놓고 말할 수는 없을 것이다. 스마트폰과 SNS의 발전으로 인하여 이전 시대와는 비교도 할 수 없을 정도의 빠른 속도로 정보가 쌓여가고 있는데 그렇기 때문에 이러한 데이터를 분석하는 기술을 이용해 사람들의 삶 속에서 최적화된 알고리즘을 만들어 가고 있으며 이미 인류의 기술에 있어서 굉장한 효율을 발휘하고 있다.

본 논문에서는 그러한 분석기술 중의 한 방법인 Linear Regression에 대해서 다뤄볼 예정이다. Regression 분석방법은 데이터를 분석하는 알고리즘인 Machine Learning 분야에서 가장 많이 사용되고 있는 분석 방법 중 하나로서 데이터와 데이터 간의 연관성을 보여주는 분석 기법이다. 또한 Linear Regression은 그러한 분석 방법들 중 기초적인 부분이기에 Regression 분석의 메커니즘을 이해하기 쉽다.

따라서 본 논문에서는 실제로 사용되고 있는 Linear Regression 알고리즘을 이용한 코드와 Linear Regression의 이론적인 부분을 직접 구현한 코드를 비교해 봄으로써 기초적인 분석 방법인 Linear Regression이 프로그램 내부에서 구현되는 메커니즘을 이해하는 것이 목표이며 더 나아가서는 실제로 사용될 수 있는 데이터 셋과 분석 기법을 이용해 분석을 진행해 보면서 Regression 분석에 대한 기능적인 이해도를 높이고 분석을 진행할 수 있는 능력을 기르는 것이 최종적인 목표이다.



## 1-2 연구에서 사용된 프로그램

본 연구를 진행하면서 프로그래밍 언어로서 Python을 사용했다. Python은 오픈 소스 프로그래밍 언어로 AI 관련 프로그래밍 언어 중 가장 많이 사용되고 있는 언어다. 또한 Python은 내부적으로 머신러닝과 관련된 다양한 라이브러리를 사용할 수 있어 Regression 분석이 용이하다. 본 연구에서 사용된 라이브러리는 numpy, pandas, sci-kit learning으로 numpy는 선형대수 및 수치 연산을 pandas는 데이터 파일 접근 및 수정을 sci-kit learning은 머신러닝 라이브러리로 실제로 구현한 부분에서는 이 라이브러리를 대신할 수 있는 코드를 구현한 것이다. 일반적으로는 Regression의 분석을 담당하는 함수를 불러올 수 있다. 프로그래밍 개발 환경으로는 jupyter notebook을 사용했다.

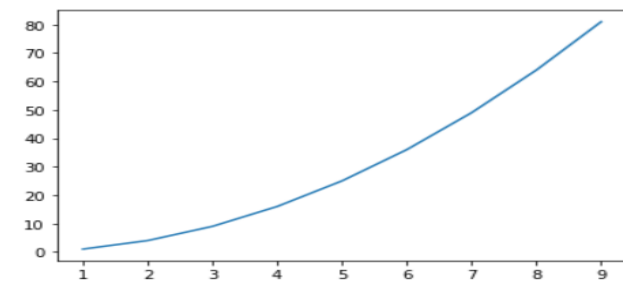
# 제 2 장 이론

## 2.1 상관계수

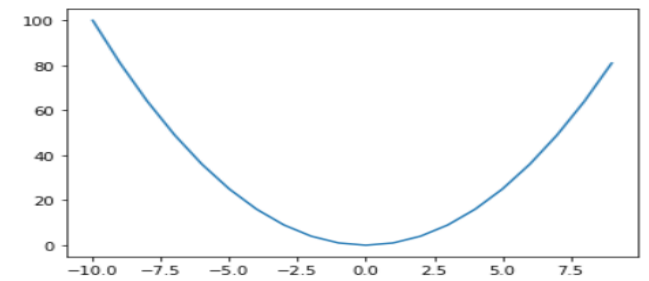
Linear Regression을 설명하기에 앞서 먼저 짚고 넘어가야 할 개념이 있는데 이는 Correlation coefficient(상관계수)이다. 상관계수는 독립 변수와 의존 변수의 관계성을 수치화해 하나의 수로 표현한 것을 의미한다. 상관계수는 함수의 선형성이 얼마나 데이터에 적합한지를 보여주는 수단이다. 상관계수는 두 변수의 공분산을 각 변수의 표준편차의 곱으로 나눠서 구하는데 식은 다음과 같다.

$$r = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}}$$

위 식을 이용해서 구한 값이 음의 값을 가지면 선형함수 일 때 변수가 반비례하는 것을 의미하고 양의 값을 가지면 비례하는 것을 의미한다. 또한 다음 그림과 같이 상관계수가 0에 가까울수록 비 선형적인 모습을 나타내고 절댓값이 1에 가까울수록 선형에 가까운 모습을 나타낸다.



correlation coefficient : 0.9752810433442548



correlation coefficient : -0.19104017997521752

**상관계수의 절댓값이 1에 가까운 함수(좌)와 0에 가까운 함수(우)의 그래프**

## 2.2 Linear Regression

Linear Regression은 한 변수에 대하여 다른 변수의 변화를 예측하려고 할 때 관계를 선형 함수 꼴로 나타내어 분석하는 것을 말한다. x 변수와 y 변수를 가정하고 두 변수를 Linear Regression으로 분석한다고 하면 다음과 같다.

$$y = ax + b$$

Regression 분석을 진행하기 위해 위의 식을 이용해서 잔차의 값을 구해야 하는데, Linear Regression 분석한 함수의 값은 실제 값이 아닌 수많은 변수들의 값을 하나로 묶은 것이므로 본래 값과는 차이가 있을 수밖에 없다. 이를 잔차라고 한다. n 개의 데이터를 Linear Regression 분석했을 때 각 변수의 잔차는 다음과 같다.

$$R_n = y_n - ax_n - b$$

여기서 잔차의 절댓값이 최소가 되는 방향으로 regression을 진행해야 한다. 그러나 잔차의 마이너스 값을 수반하므로 위 식을 제곱한 값의 총합이 최소가 되는 a, b 값을 찾는 것이 최소 제곱법이다.

$$\sum R_n^2 = \sum (y_n - ax_n - b)^2$$

a, b 에 대한 최솟값을 구하기 위해 위 식을 a와 b로 나누어 편미분을 해서 0이 되는 값을 찾아준다.

$$\frac{\partial \sum R_n^2}{\partial a} = - \sum 2x_n(y_n - ax_n - b)$$

$$\frac{\partial \sum R_n^2}{\partial b} = - \sum (y_n - ax_n - b)$$

위 식을 전개해서 a와 b의 최적화된 값을 얻는다. 그 과정은 다음과 같다.

$$a = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2} = \frac{n(\sum_{i=1}^n x_i y_i) - (\sum_{i=1}^n x_i)(\sum_{i=1}^n y_i)}{n(\sum_{i=1}^n x_i^2) - (\sum_{i=1}^n x_i)^2}$$

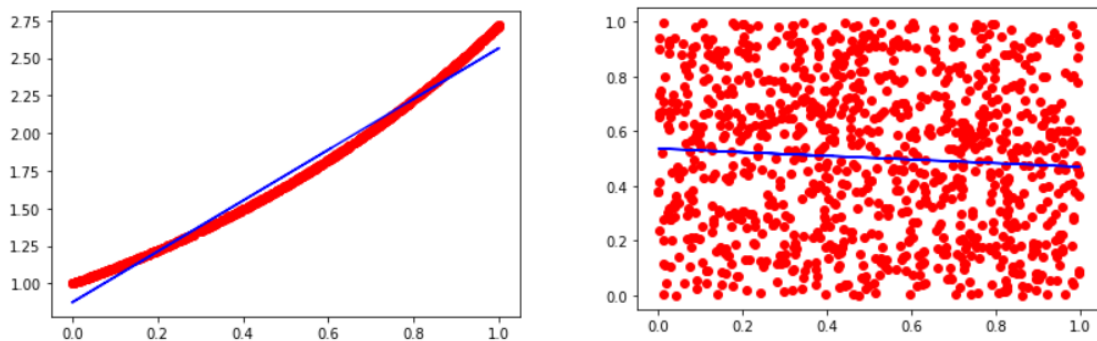
$$\bar{x}, \bar{y} : x, y \text{의 평균}$$

Regression 함수  $y=ax+b$ 는 전체 데이터 값의 각 변수  $x$ 와  $y$ 의 평균을 입력하면 잔차 없이 성립하므로  $b$ 에 대한 식은 다음과 같다.

$$b = \bar{y} - a\bar{x}$$

구해진  $a$ 와  $b$ 를 이용해 일차함수를 그리면 Linear Regression 함수를 확인할 수 있다.

## 2.3 결정계수



결정계수의 값이 1에 가까운 함수(좌)와 0에 가까운 함수(우)

결정계수  $r^2$ 은 0과 1사이의 수를 가지고 있는 값으로 결정계수의 값이 1에 가까울수록 데이터의 산포가 선에 가깝게 나타나며 이는 분석이 제대로 됨을 의미한다.

결정계수는 상관계수가 데이터 분포의 선형성을 나타내는 데에 비해 regression된 데이터가 얼마나 정확한 지 나타내는 척도이다. 결정계수는 상관계수의 제곱으로 구해지는데 다음에 따라 증명한다.

우선 Linear Regression 함수를 이용해 분산을 추정한다. Linear Regression 함수를 구하기 위해서 우리는 잔차의 제곱된 값(SSE:Error Sum of Square)의 합이 최소가 되는 값을 찾았다. 이를 사용하면 함수와 실제 데이터 사이가 얼마나 추정할 수 있는데 이를 MSE(Mean Square Error)라고 부르며 방법은 다음과 같다.

$$\text{분산: } \sigma^2 = MSE = \frac{SSE}{n - 2}$$

추정된 분산의 값은 값이 작으면 작을수록 Regression 분석이 잘 이루어졌다고 할 수 있으나 그 정도가 불분명하기에 자료로서 사용하기에 모호하다. 이를 보다 분명하게 하기 위해 다음 개념들을 사용한다. SST(Total Sum of Square)는 데이터의 의존변수에 해당하는 값과 그 평균의 차이를 제곱해서 합한 값을 말하며 다음과 같이 구할 수 있다.

$$SST = \sum_{i=1}^n (y_i - \bar{y})^2 = \left( \sum_{i=1}^n y_i^2 \right) - \left( \frac{\sum_{i=1}^n y_i}{n} \right)^2$$

SST는 독립변수를 배제한 오차의 크기를 나타내고 SSE는 독립변수를 포함한 분석에서 오차의 크기를 나타내므로 결국 SST에 비해 SSE가 작을 때 분석이 잘 이

루어졌다고 할 수 있다. 이를 보다 잘 설명하기 위해 SST를 SSE에 대한 식으로 표현할 수 있는데 다음과 같다.

$$\sum_{i=1}^n (y_i - \bar{y})^2 = \sum_{i=1}^n (y_i - \hat{y}_i)^2 + \sum_{i=1}^n (\hat{y}_i - \bar{y})^2$$

$$SST = SSE + SSR$$

SSR(Regression Sum of Square)은 SST를 SSE로 분해했을 때 나오는 항으로 Linear Regression 함수의 값과 전체 데이터의 평균의 차이를 나타낸다. 이는 결국 SST에서 SSR이 차지하는 비율이 크면 클 수록 분석이 잘 되었음을 의미한다. 결정계수  $r^2$ 은 SSR을 SST로 나눈 것으로 설명되고 이를 정리해 보면 상관계수의 제곱과 같은 값이 나온다.

$$r^2 = \frac{SSR}{SST} = \left( \frac{n(\sum_{i=1}^n x_i y_i) - (\sum_{i=1}^n x_i)(\sum_{i=1}^n y_i)}{\sqrt{n(\sum_{i=1}^n x_i^2) - (\sum_{i=1}^n x_i)^2} \sqrt{n(\sum_{i=1}^n y_i^2) - (\sum_{i=1}^n y_i)^2}} \right)^2$$

## 제 3 장 코드

### 3.1 구현한 코드 분석

본 코드에서 사용한 데이터는 중고차 가격 측정 모델로 본래 데이터에는 다양한 변수가 존재하지만 이번 Linear Regression에서 사용할 변수는 주행거리(Kilometers)와 가격(Price)이다. 다음 코드는 데이터 전처리 부분과 Linear

Regression의 실행 부분으로 나뉘져있다.

# 데이터 전처리 : 데이터를 분석하기 전 지저분한 데이터를 정제하는 과정이다.

# 라이브러리 불러오기

```
import pandas as pd
```

```
import numpy as np
```

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

# 데이터 파일 불러오기

```
pre_data = pd.read_csv("C:/Users/cod/Desktop/data.csv",index_col=0)
```

```
pre_data
```

	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_Price	Price
0	Maruti Wagon R LXI CNG	Mumbai	2010	72000	CNG	Manual	First	26.6 km/kg	998 CC	58.16 bhp	5.0	NaN	1.75
1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	Diesel	Manual	First	19.67 kmpl	1582 CC	126.2 bhp	5.0	NaN	12.50
2	Honda Jazz V	Chennai	2011	46000	Petrol	Manual	First	18.2 kmpl	1199 CC	88.7 bhp	5.0	8.61 Lakh	4.50
3	Maruti Ertiga VDI	Chennai	2012	87000	Diesel	Manual	First	20.77 kmpl	1248 CC	88.76 bhp	7.0	NaN	6.00
4	Audi A4 New 2.0 TDI Multitronic	Coimbatore	2013	40670	Diesel	Automatic	Second	15.2 kmpl	1968 CC	140.8 bhp	5.0	NaN	17.74

## 데이터 확인

# 데이터 복제 : 초기 데이터와의 비교 및 문제가 생겼을 때를 대비하여 저장한다.

```
refined_data = pre_data.copy(deep=True)
```

# 데이터 확인

refined\_data.head()

	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_Price	Price
0	Maruti Wagon R LXI CNG	Mumbai	2010	72000	CNG	Manual	First	26.6 km/kg	998 CC	58.16 bhp	5.0	NaN	1.75
1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	Diesel	Manual	First	19.87 kmpl	1582 CC	126.2 bhp	5.0	NaN	12.50
2	Honda Jazz V	Chennai	2011	46000	Petrol	Manual	First	18.2 kmpl	1199 CC	88.7 bhp	5.0	8.61 Lakh	4.50
3	Maruti Ertiga VDI	Chennai	2012	87000	Diesel	Manual	First	20.77 kmpl	1248 CC	88.76 bhp	7.0	NaN	6.00
4	Audi A4 New 2.0 TDI Multitronic	Coimbatore	2013	40670	Diesel	Automatic	Second	15.2 kmpl	1968 CC	140.8 bhp	5.0	NaN	17.74

복사한 데이터 확인

# 쓸모없는 data 삭제

```
refined_data.drop(["Name","Location","Year","Fuel_Type","Transmission","Owner_Type",  
,"Mileage","Power","Engine","Seats","New_Price"],axis=1,inplace=True)
```

refined\_data.head()

	Kilometers_Driven	Price
0	72000	1.75
1	41000	12.50
2	46000	4.50
3	87000	6.00
4	40670	17.74

Kilometers\_Driven과 Price를 제외한 columns 삭제

# null 확인 : 결측치를 확인하는 과정인데 사용된 데이터에서는 결측치가 확인되지 않았다. 만약 결측치가 확인된다면 지워주거나 평균값을 입력해 분석에 영향을 주지 않도록 한다.

refined\_data.isnull().sum()



```
Kilometers    0
Price         0
dtype: int64
```

### null이 없음을 확인

```
# columns 변경
```

```
columns=list(refined_data.columns)
```

```
for i in range(len(columns)):
```

```
    columns[i]=columns[i].split('_',1)[0]
```

```
refined_data.columns=columns
```

```
refined_data.head()
```

	Kilometers	Price
0	72000.0	1.75
1	41000.0	12.50
2	46000.0	4.50
3	87000.0	6.00
4	40670.0	17.74

### Kilometers\_Driven => Kilometers

# type변경 : 데이터가 저장된 형식을 파악해 계산에 문제가 없도록 float(부동소수점)로 바꿔준다.

```
refined_data.dtypes
```

```
Kilometers    int64
Price         float64
dtype: object
```

### Kilometers의 형식이 int임을 확인

```
refined_data.Kilometers=refined_data.Kilometers.astype("float")
```

```
Kilometers    float64
Price         float64
dtype: object
```

### **Kilometers의 형식을 float으로 변경**

# 데이터 확인 : 데이터를 산점도로 확인해보는 과정이다. 위의 그림에서처럼 과하게 벗어난 데이터가 있을 경우 분석이 제대로 이루어지지 않는다.(Anscombe's

quartet)

```
kilometers=refined_data.Kilometers.values
```

```
price=refined_data.Price.values
```

```
fig, ax = plt.subplots(1,2,figsize=(16,4))
```

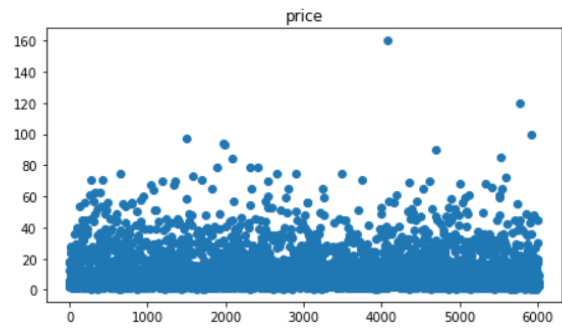
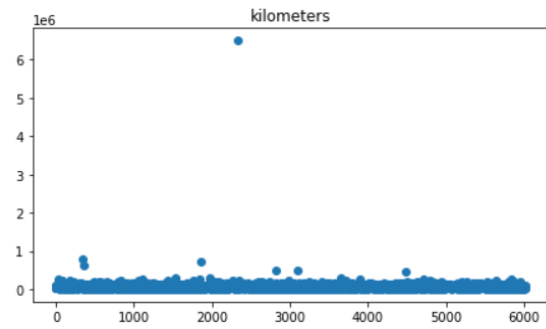
```
ax[0].plot(kilometers)
```

```
ax[0].set_title('kilometers')
```

```
ax[1].plot(price)
```

```
ax[1].set_title('price')
```

```
plt.show()
```



**kilometers(주행거리)의 단위 : km/price(가격)의 단위 : \$ (x축은 인덱스)**

# 확인된 데이터 제거

check=[]

for i in range(len(kilometers)):

    if kilometers[i]> 400000:

        check.append(i)

for i in range(len(price)):

    if price[i]>150:

        check.append(i)

refined\_data.drop(check,inplace=True)

kilometers=refined\_data.Kilometers.values

price=refined\_data.Price.values

# 데이터 재 확인

fig, ax = plt.subplots(1,2,figsize=(10,4))

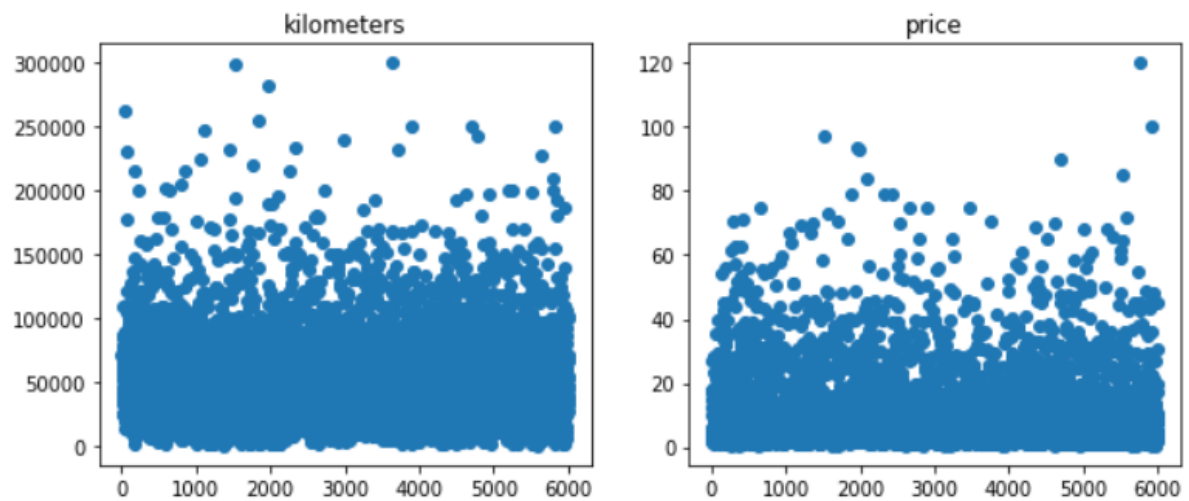
```
ax[0].plot(kilometers,'o')
```

```
ax[0].set_title('kilometers')
```

```
ax[1].plot(price)
```

```
ax[1].set_title('price')
```

```
plt.show()
```

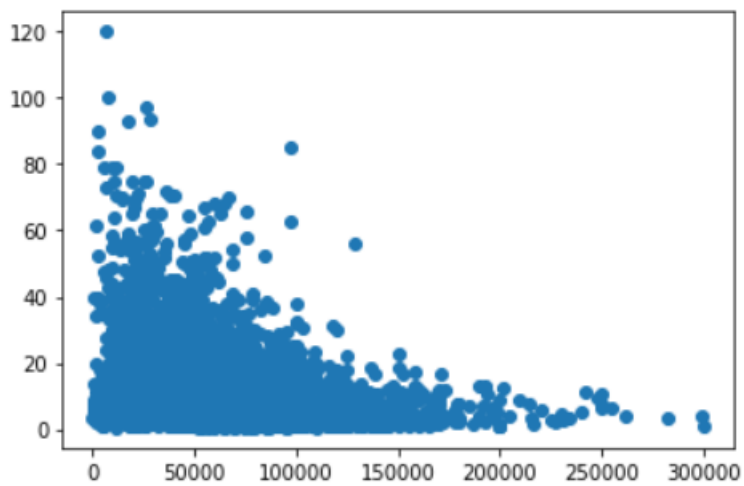


**kilometers(주행거리)의 단위 : km/price(가격)의 단위 : \$ (x축은 인덱스)**

# x : kilometers y : price 그래프로 확인

```
plt.plot(kilometers,price,'o')
```

```
plt.show()
```



x(주행거리)의 단위 : km/y(가격)의 단위 : \$

# 제거된 데이터가 있으므로 정확한 데이터의 갯수 파악을 위해 인덱스(서순)를 재정렬 해준다.

refined\_data

	Kilometers	Price
0	72000.0	1.75
1	41000.0	12.50
2	46000.0	4.50
3	87000.0	6.00
4	40670.0	17.74
...	...	...
6014	27385.0	4.75
6015	100000.0	4.00
6016	55000.0	2.90
6017	46000.0	2.65
6018	47000.0	2.50

6011 rows × 2 columns

6011개의 행이 있지만 6018 까지 인덱스가 설정되어 있다.

refined\_data=refined\_data.reset\_index()

```
refined_data.drop(refined_data.iloc[:,[0]],axis=1,inplace=True)
```

refined\_data

	Kilometers	Price
0	72000.0	1.75
1	41000.0	12.50
2	46000.0	4.50
3	87000.0	6.00
4	40670.0	17.74
...	...	...
6006	27365.0	4.75
6007	100000.0	4.00
6008	55000.0	2.90
6009	46000.0	2.65
6010	47000.0	2.50

6011 rows × 2 columns

서순 변경

```
# Linear Regression 분석
```

```
# 라이브러리 불러오기
```

```
import numpy as np
```

```
import pandas as pd
```

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

```
from sklearn.linear_model import LinearRegression
```

```
from data import refined_data # 정제시킨 데이터를 불러온다.
```

```
#데이터 전처리
```

정제시킨 데이터를 구현한 코드에 맞춰서 다시 한번 전처리한다.

```
# columns 변경
```

```
columns=["x","y"]
```

```
chart.columns=columns
```

```
# xy columns 추가
```

```
chart['xy']=chart.x*chart.y
```

x와 y를 곱한 값을 추가시킨다.

	x	y	xy
0	72000.0	1.75	126000.00
1	41000.0	12.50	512500.00
2	46000.0	4.50	207000.00
3	87000.0	6.00	522000.00
4	40670.0	17.74	721485.80

**Kilometers => x/Price => y/xy삽입**

```
# 상관계수 계산과 Linear Regression 분석을 위한 변수 설정
```

```
n=len(chart.iloc[:,0])
```

```
x=chart.x.values
```

```
y=chart.y.values
```

```
xsum=chart.x.sum()
```

```
ysum=chart.y.sum()
```

```
xSsum=np.square(chart.x).sum()
```

```
ySsum=np.square(chart.y).sum()
```

```
xysum=chart.xy.sum()
```

```
sig_xy=n*xysum-xsum*ysum
```

```
sig_x=np.sqrt(n*xSsum-xsum**2)
```

```
sig_y=np.sqrt(n*ySsum-ysum**2)
```

```
y_bar=ysum/n
```

```
x_bar=xsum/n
```

```
# 상관계수
```

```
r=sig_xy/(sig_x*sig_y)
```

```
-0.18519881309327973
```

```
r : 상관계수
```

```
r
```

```
# 결정계수
```

```
np.square(r)
```

```
0.03429860037115956
```

```
r2: 결정계수
```

```
# a(계수),b(절편) 정의
```

```
coef=r*sig_y/sig_x
```

```
coef
```

```
-6.0331688063697593e-05
```

```
coef : 독립변수의 계수
```

```
intercept=y_bar-coef*x_bar
```

```
intercept
```

```
12.895921468095686
```



**intercept : 의존변수의 절편**

# Linear Regression 함수 정의

```
def hommade_regression(x):
```

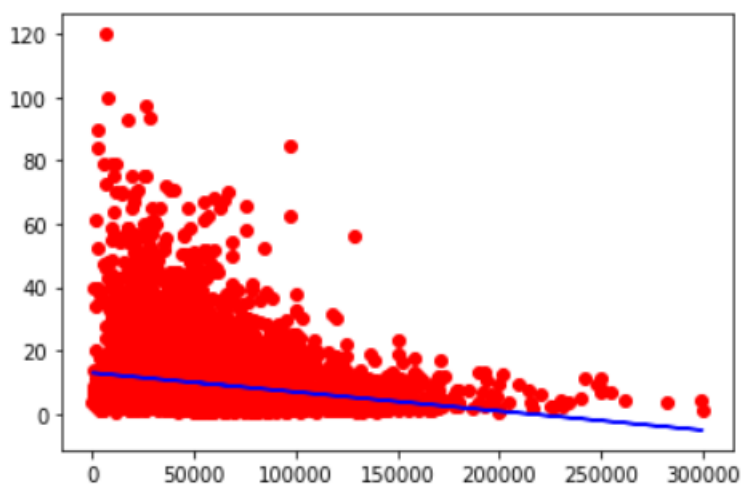
```
    yy=intercept+coef*x
```

```
    return yy
```

# Linear Regression 함수 그래프 확인

```
plt.plot(x,y,'ro')
```

```
plt.plot(x,hommade_regression(x),'b-')
```



**x(주행거리)의 단위 : km/y(가격)의 단위 : \$**

# 임의의 예측값 설정 및 확인

```
12.895774741430316
```

**임의로 설정한 값을 x에 대입해 y값 예측**

```
hommade_regression(2.432)
```

# 기존에 존재하는 라이브러리를 사용한 결과 확인

# 상관계수

```
-0.1851988130932797
```

**위에서 확인한 상관계수 값과 일치함을 확인**

```
np.corrcoef(x, y)[0,1]
```

**#결정계수**

```
np.square(np.corrcoef(x, y)[0,1])
```

```
0.03429860037115955
```

**위에서 확인한 결정계수 값과 일치함을 확인**

**# sklearn 라이브러리에 따른 데이터 처리**

```
x=x.reshape(-1,1)
```

**# Linear Regression 분석 진행**

```
model=LinearRegression()
```

```
model.fit(x,y)
```

**# a(계수),b(절편) 확인**

```
model.coef_
```

```
[-6.03316881e-05
```

**위에서 확인한 독립변수의 계수 값과 일치함을 확인**

```
model.intercept_
```

```
12.895921468095686
```

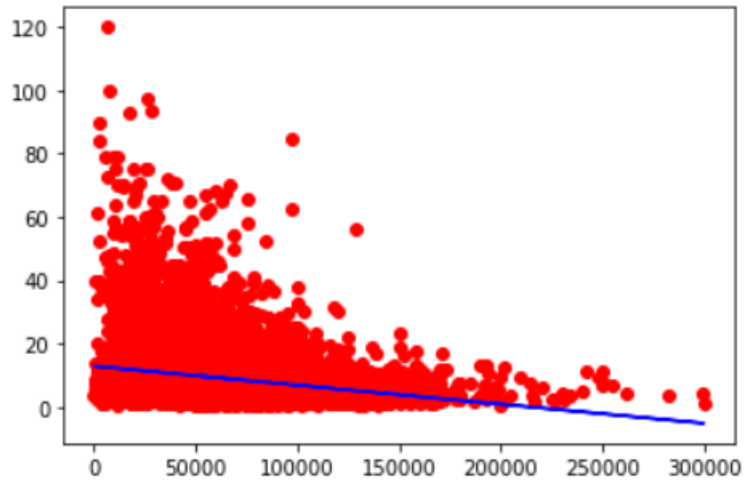
**위에서 확인한 독립변수의 계수 값과 일치함을 확인**

**# 그래프 확인**

```
plt.plot(x,y,'ro')
```

```
plt.plot(x,model.predict(x),'b-')
```

```
plt.show()
```



위에서 확인한 그래프와 일치함을 확인

x(주행거리)의 단위 : km/y(가격)의 단위 : \$

# 예측값 확인

```
predict_value=model.predict([[2.432]])
```

```
predict_value
```

```
12.89577474
```

위에서 확인한 예측값과 일치함을 확인

## 제 4 장 결론

### 4.1 분석 결과

데이터를 이용해 프로그램을 실행 시켰을 때, 상관계수의 값 -0.185와 결정계수

의 값 0.0343의 값을 얻을 수 있었다. 여기서 구해진 상관계수와 결정계수 모두 0에 가까운 값을 보였다. 이는 두 변수의 선형적인 관계성이 거의 없음을 의미하는데 그 모습은 산점도와 함수 그래프로도 확인할 수 있었다. 값을 예측할 때  $x$  값의 편차에 비해서 극히 작은 값인 2.432를 예측변수로 설정했는데 그 이유는 산점도에서 확인할 수 있듯이 낮은 주행거리에 해당하는 중고차들의 가격이 천차만별이기 때문이다.

하지만 실제에서 앞서 확인했던 통계학적인 자료가 아닌 통념적으로 생각해 보았을 때 중고차의 주행거리와 가격은 반비례하며 중고차라고는 하지만 차종이나 브랜드에 따라 가격이 천차만별일 것이므로 두 변수는 분명 관계성이 있다. 통계학적인 자료는 두 변수의 상관관계가 그다지 없는 것처럼 보이지만 실제로는 존재한다. 이 부분은 Linear Regression 했을 때의 상관계수와 결정계수의 한계점을 시사한다. 선형적인 관계성이 불분명할지라도 분명 두 변수는 관계가 있을 수 있는 것이다.

직접 구현한 코드와 이미 사용되고 있는 코드 사이의 차이는 본 자료를 활용했을 때는 거의 없었다. 10만개 이상의 데이터를 이용해서 테스트 해보았을 때도 큰 차이를 보이지 않았기에 성능상에 차이는 거의 없다고 볼 수 있다. 라이브러리의 내부적인 코드를 확인할 순 없었지만 구현한 코드와 큰 차이는 보이지 않을 것으로 여겨진다.

## 4.2 고찰

분석을 실행함에 있어서 자료로 쓰인 데이터는 하나의 독립변수만을 가지고

Linear Regression 분석을 진행했을 때의 한계를 명확히 보여주는 사례이다. 앞서 확인한 바와 같이 프로그램을 실행했을 때 출력되는 자료들은 두 값의 상관관계가 없는 것처럼 보였지만 실제로는 그렇지 않았다. 이러한 문제점을 해결하기 위해 다양한 변수를 함께 취급할 수 있다. 우리가 사용한 데이터에서는 차의 연비나 소음의 정도 마력 등을 주행거리와 함께 사용한다면 훨씬 정확한 값을 얻을 수 있을 것으로 보인다.

Linear Regression은 경제 산업 전반에서 굉장히 많이 사용되고 있는 분석 방법의 하나이다. 매분 매초마다 데이터가 쏟아져 나오고 있는 세상에서 Linear Regression 분석 방법은 각 산업에서 추정이나 예측 모델을 통해 새로운 방향을 제시할 것으로 생각된다.

## 참고문헌

- [1] Marc Peter Deisenroth, A.Aldo Faisal, Cheng Soon Ong. Mathematics For Machine Learning(2020)
- [2] 김용대, 등. 통계학개론. 제 5개정판. 영지문화사(2008)