Chapter 4

Regression

Sejong Oh

DKU DANKOOK UNIVERSITY

Contents

- Machine learning 분류
  - 지도학습 (supervised learning)
    - 회귀(regression)
    - 분류(classification) 등
  - 비지도학습(unsupervised learning)
    - 군집화(clustering)
  - 강화학습(Reinforcement learning)

1. Simple linear regression

- 정의
  - 종속 변수(y) 와 독립변수(x) 사이의 선형 관계를 파악하고 이를 예측에 활용하는 방법
  - 예) 기온(x) 과 아이스 크림 판매량(y) 사이의 관계식을 찾아낸다. 이를 이용하여 내일의 예상 기온으로 부터 예상 아이스크림 판매량을 예측한다. => 필요한 아이스크림 재료의 양을 예측할 수 있다.
  - 기온(x) 과 아이스 크림 판매량(y) 사이의 관계식을 모델(model) 이라고 한다. (회귀 모델, 예측 모델)
  - 단순 선형 회귀식은 다음과 같은 형태

$$y = Wx + b$$

상수인 W와 b 를 찾는 것이 학습(훈련) 목표

1. Simple linear regression

- 정의
  - 기온(x) 과 아이스 크림 판매량(y) 사이의 관계식이 다음과 같다고 하자

$$y = 2500\,x + 45$$

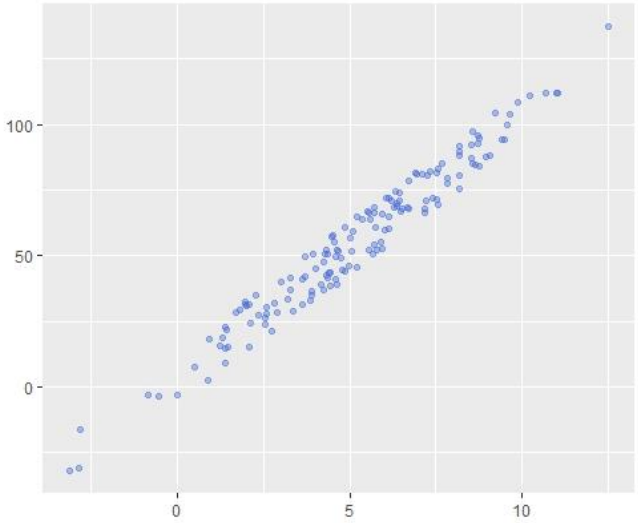
  - 내일의 예상 기온이 32 도 이면 내일의 아이스크림 예상 판매량은 다음과 같이 예측될 수 있다

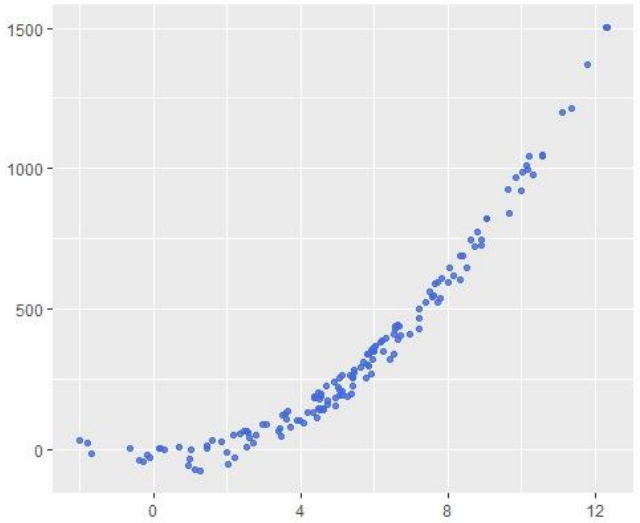
$$y = 2500 \times 32 + 45 = 80045$$

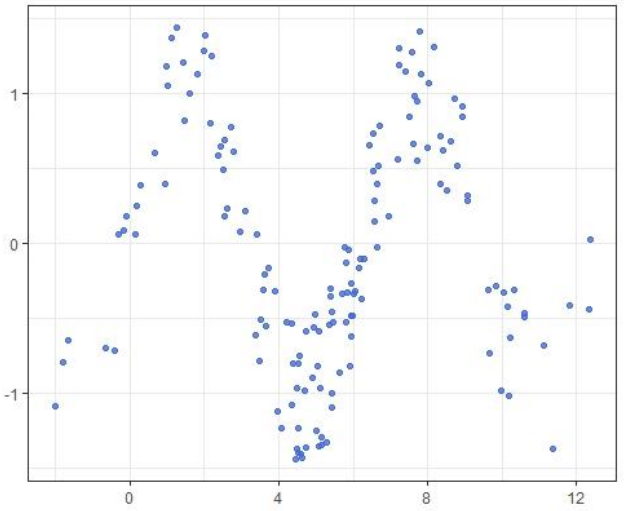
  - 현실세계에서는 두 변수가 선형 관계에 있는 경우가 많아서 선형회귀분석이 유용하다
  - 두변수가 선형 관계에 있는지 알아보는 방법 : 산점도, 상관계수

Simple linear regression

1. Simple linear regression



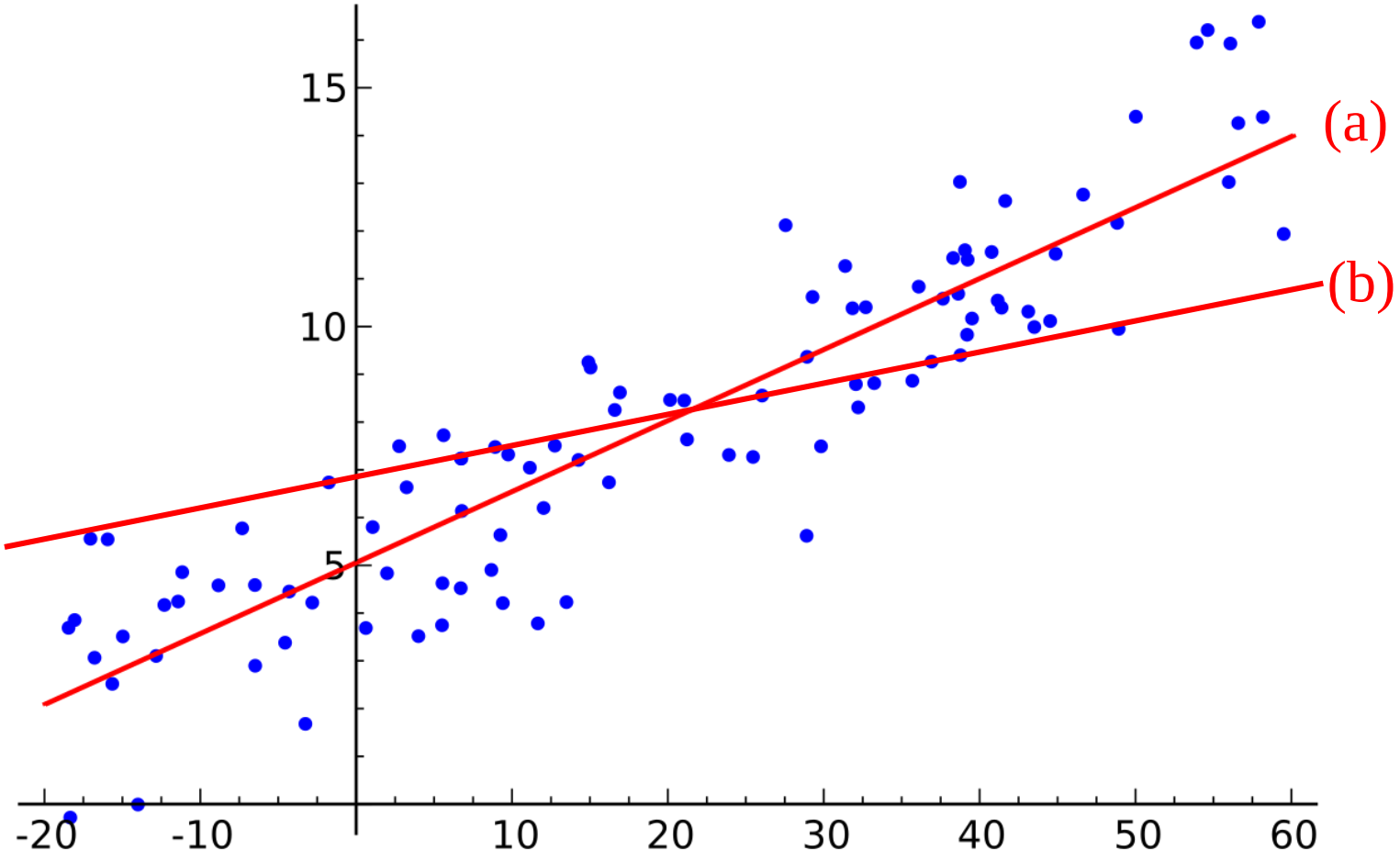




https://wikidocs.net/34065

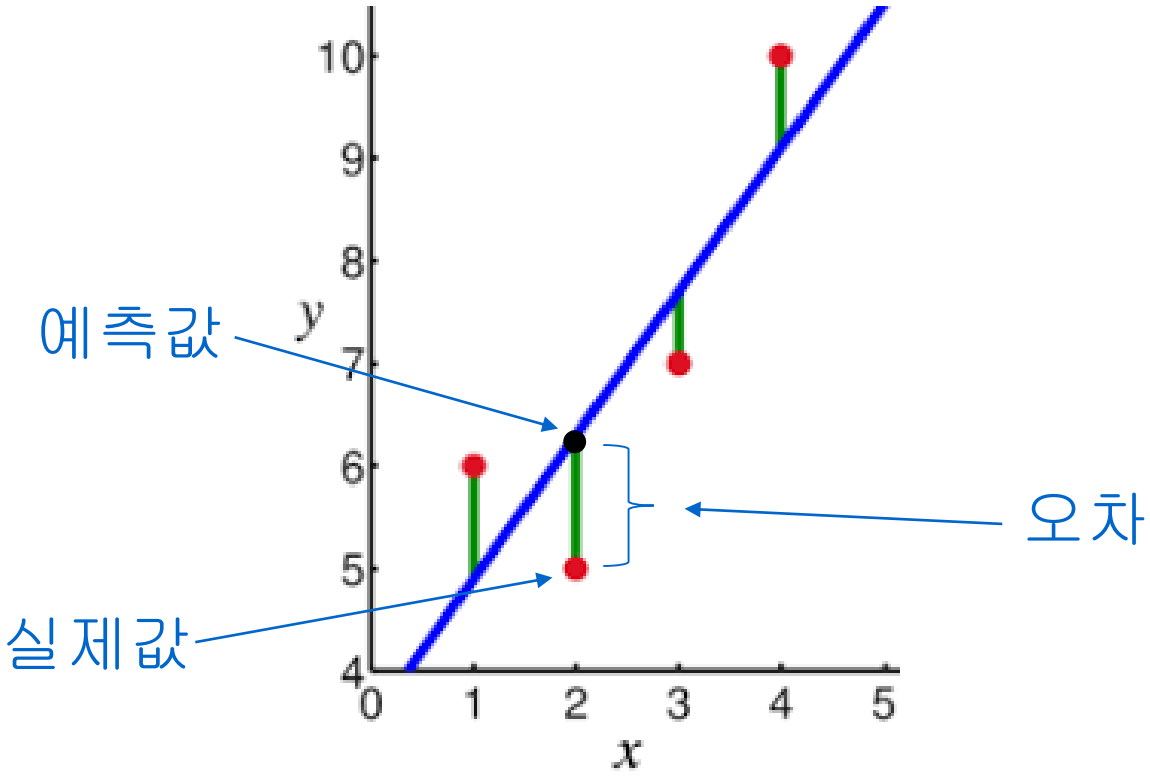
1. Simple linear regression

- 회귀식에서 W와 b를 찾는 방법



1. Simple linear regression

- 회귀식에서 W와 b를 찾는 방법



"최소 제곱법"

1. Simple linear regression

- Example problem: car speed vs stopping distance

	A	B
1	speed	dist
2	4	2
3	4	10
4	7	4
5	7	22
6	8	16
7	9	10
8	10	18
9	10	26
10	10	34
11	11	17
12	11	28

cars.csv

- speed: Speed (mph)
- dist: Stopping distance (ft)

https://grimmermotors.co.nz/car-brakes-take-longer-to-stop-repairs-hamilton/

1. Simple linear regression

- Python code

04.regression_simple.py

```
# load required modules

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

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
```

1. Simple linear regression

```
# prepare dataset
cars = pd.read_csv('D:/data/cars.csv')
print(cars)
speed = cars['speed']
dist  = cars['dist']
```

```
In [49]: print(cars)
    speed  dist
0       4     2
1       4    10
2       7     4
3       7    22
4       8    16
5       9    10
6      10    18
7      10    26
8      10    34
9      11    17
10     11    28
11     12    14
12     12    20
```

```
In [55]: speed
Out[55]:
0      4
1      4
2      7
3      7
4      8
5      9
6     10
7     10
8     10
9     11
10    11
11    12
12    12
```

```
In [57]: dist
Out[57]:
0      2
1     10
2      4
3     22
4     16
5     10
6     18
7     26
8     34
9     17
10    28
11    14
12    20
```

50x2

50x1

50x1

1. Simple linear regression

```
# data frame to np.array
speed = np.array(speed).reshape(50,1)
dist  = np.array(dist).reshape(50,1)
```

```
In [61]: np.array(speed)
Out[61]:
array([ 4,  4,  7,  7,  8,  9, 10, 10, 10, 11, 11, 12, 12, 12, 12, 13, 13,
       13, 13, 14, 14, 14, 14, 15, 15, 15, 16, 16, 17, 17, 17, 18, 18, 18,
       18, 19, 19, 19, 20, 20, 20, 20, 20, 22, 23, 24, 24, 24, 24, 25],
      dtype=int64)
```

1차원 배열 (원소수 50)

```
In [62]: np.array(speed).reshape(50,1)
Out[62]:
array([[ 4],
       [ 4],
       [ 7],
       [ 7],
       [ 8],
       [ 9],
       [10],
       [10],
       [10],
       [11],
       [11],
       [12],
       [12],
```

2차원 배열 (50x1)

1. Simple linear regression

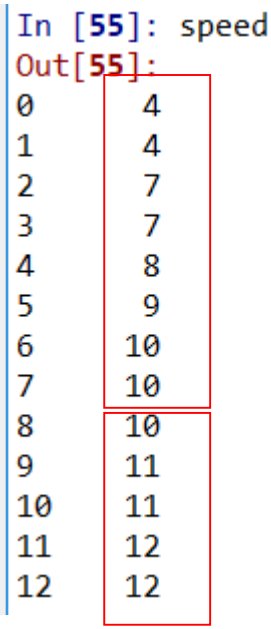
```
# Split the data into training/testing sets
train_X, test_X, train_y, test_y = \
    train_test_split(speed, dist, test_size=0.2, random_state=123)
```

\ 하나의 명령문을 여러줄에 걸쳐서 작성할 때 연결기호

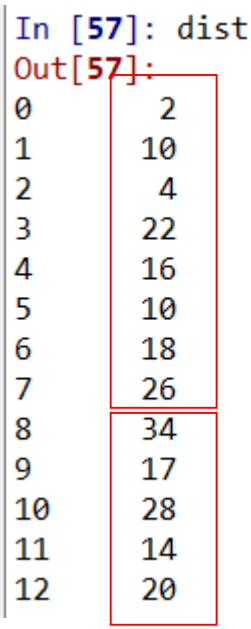
random_state: seed

train_X

test_X



```
In [55]: speed
Out[55]:
0      4
1      4
2      7
3      7
4      8
5      9
6     10
7     10
8     10
9     11
10    11
11    12
12    12
```



```
In [57]: dist
Out[57]:
0      2
1     10
2      4
3     22
4     16
5     10
6     18
7     26
8     34
9     17
10    28
11    14
12    20
```

train_y

test_y

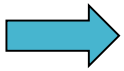
1. Simple linear regression

```
# Dfine learning method
model =  LinearRegression()

# Train the model using the training sets
model.fit(train_X, train_y)

# Make predictions using the testing set
pred_y = model.predict(test_X)
print(pred_y)
```

```
In [67]: test_X
Out[67]:
array([[11],
       [12],
       [17],
       [24],
       [13],
       [ 4],
       [20],
       [12],
       [17],
       [10]], dtype=int64)
```



```
In [66]: print(pred_y)
[[25.221513  ]
 [29.18014184]
 [48.97328605]
 [76.68368794]
 [33.13877069]
 [-2.48888889]
 [60.84917258]
 [29.18014184]
 [48.97328605]
 [21.26288416]]
```

1. Simple linear regression

```
# prediction test
print(model.predict([[13]]))    # when speed=13
print(model.predict([[20]]))    # when speed=20
```

```
In [68]: print(model.predict([[13]]))    # when speed=13
[[33.13877069]]

In [69]: print(model.predict([[20]]))    # when speed=20
[[60.84917258]]
```

```
In [70]: shape([13])
Out[70]: (1,)

In [71]: shape([[13]])
Out[71]: (1, 1)
```

1. Simple linear regression

```
# The coefficients & Intercept
print('Coefficients: {0:.2f}, Intercept {1:.3f}'\
      .format(model.coef_[0][0], model.intercept_[0]))
```

```
In [72]: print('Coefficients: {0:.2f}, Intercept {1:.3f}'\
    ...:       .format(model.coef_[0][0], model.intercept_[0]))
    ...:
Coefficients: 3.96, Intercept -18.323
```

$$y = Wx + b$$

$$y = 3.96x - 18.323$$

$$dist = 3.96 \times speed - 18.323$$

1. Simple linear regression

- Model evaluation 1

```
# The mean squared error
print('Mean squared error: {0:.2f}'.\
      format(mean_squared_error(test_y, pred_y)))
```

```
In [74]: print('Mean squared error: {0:.2f}'.\
    ...:       format(mean_squared_error(test_y, pred_y)))
Mean squared error: 86.11
```

$$MSE = \sum_{k=0}^{n} (test\_y_k - pred\_y_k)^2$$

실제값 모델에 의한 예측값

이 값을 작을 수록 정확한 모델임

1. Simple linear regression

- Model evaluation 2

```
# The coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: %.2f'
      % r2_score(test_y, pred_y))
```

```
In [75]: print('Coefficient of determination: %.2f'
    ...:       % r2_score(test_y, pred_y))
    ...:
Coefficient of determination: 0.84
```

R^2 score : 최대값은 1 (모델의 예측 결과와 실제값이 완전히 일치할 때)
이 값이 1 에 가까울수록 좋은 모델
완전히 잘못된 모델은 (-)값이 나올수도 있음

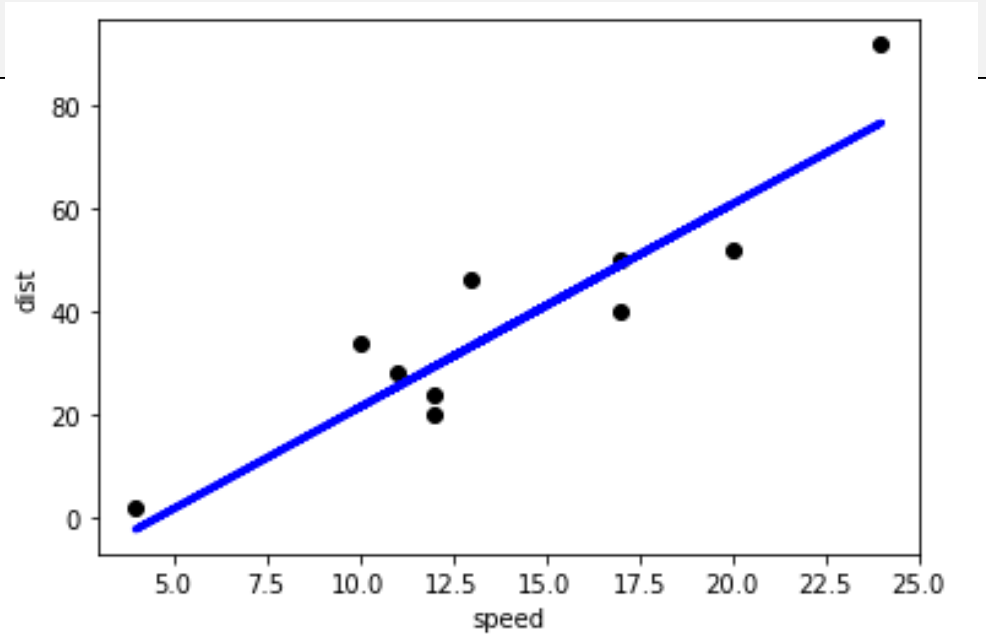
1. Simple linear regression

- 모델 시각화

```
# Plot outputs
plt.scatter(test_X, test_y,  color='black')
plt.plot(test_X, pred_y, color='blue', linewidth=3)

plt.xlabel('speed')
plt.ylabel('dist')

plt.show()
```



2. Multiple linear regression

- Definition
  - Multiple linear regression
  - 독립변수(설명변수)가 2개 이상인 경우
  - 예) 키(x1) 와 몸무게(x2)를 가지고 혈당수치(y)를 를 예측
    - 독립변수(설명변수) : 키, 몸무게
    - 종속변수(반응변수) : 혈당수치
  - 중선형 회귀식의 형태

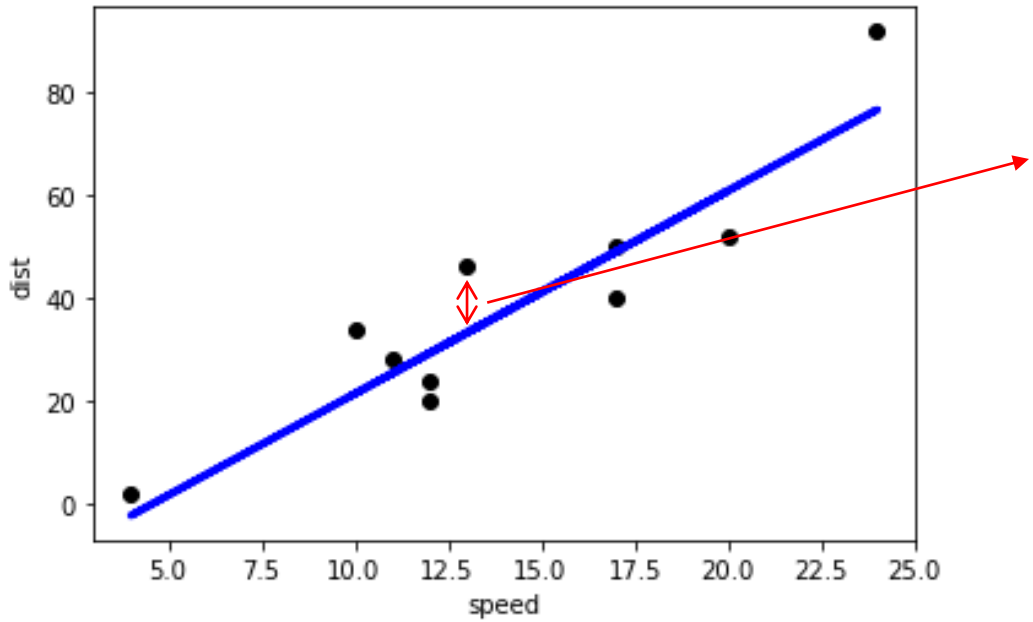
$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots \beta_k x_k + \epsilon.$$

intercept

coefficients

2. Multiple linear regression

- Motivation



이 차이는 어디서 발생을 하는가?

제동거리는 단순히 주행속도에 의해서만 결정이 되는가

타이어 년수
브레이크 성능
당일 풍속
…

단순 선형회귀 : (주행속도) → (제동거리)
중 선형회귀 : (주행속도,타이어년수,당일풍속) → (제동거리)

2. Multiple linear regression

- 사례: 연봉 예측 모델
  - 특정 직군의 연봉을 3가지 변수(교육년수, 여성비율, 평판)를 가지고 예측해보자
  - 데이터셋 : prestige.csv

	A	B	C	D	E	F	G
1		education	income	women	prestige	census	type
2	gov.administrators	13.11	12351	11.16	68.8	1113	prof
3	general.managers	12.26	25879	4.02	69.1	1130	prof
4	accountants	12.77	9271	15.7	63.4	1171	prof
5	purchasing.officers	11.42	8865	9.11	56.8	1175	prof
6	chemists	14.62	8403	11.68	73.5	2111	prof
7	physicists	15.64	11030	5.13	77.6	2113	prof
8	biologists	15.09	8258	25.65	72.6	2133	prof
9	architects	15.44	14163	2.69	78.1	2141	prof
10	civil.engineers	14.52	11377	1.03	73.1	2143	prof

직업

교육년수

수입(연봉)

여성비율

직업에 대한 평판

2. Multiple linear regression

04.regression_multiple.py

```
import pandas as ps
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
```

2. Multiple linear regression

```
# Load the prestge dataset
df = ps.read_csv('D:/data/prestige.csv')
print(df)
df_X = df[['education','women','prestige']]
df_y = df['income']
```

```
In [127]: print(df)
              Unnamed: 0  education  income  women  prestige  census  type
0     gov.administrators      13.11   12351  11.16      68.8    1113  prof
1       general.managers      12.26   25879   4.02      69.1    1130  prof
2             accountants      12.77    9271  15.70      63.4    1171  prof
3    purchasing.officers      11.42    8865   9.11      56.8    1175  prof
4               chemists      14.62    8403  11.68      73.5    2111  prof
..                   ...        ...     ...    ...       ...     ...   ...
97          bus.drivers       7.58    5562   9.47      35.9    9171    bc
98         taxi.drivers       7.93    4224   3.59      25.1    9173    bc
99         longshoremen       8.37    4753   0.00      26.1    9313    bc
100         typesetters      10.00    6462  13.58      42.2    9511    bc
101         bookbinders       8.55    3617  70.87      35.2    9517    bc

[102 rows x 7 columns]
```

2. Multiple linear regression

```
In [128]: df_X
Out[128]:
     education  women  prestige
0        13.11  11.16      68.8
1        12.26   4.02      69.1
2        12.77  15.70      63.4
3        11.42   9.11      56.8
4        14.62  11.68      73.5
..         ...    ...       ...
97        7.58   9.47      35.9
98        7.93   3.59      25.1
99        8.37   0.00      26.1
100      10.00  13.58      42.2
101       8.55  70.87      35.2

[102 rows x 3 columns]
```

```
In [129]: df_y
Out[129]:
0      12351
1      25879
2       9271
3       8865
4       8403
       ...
97      5562
98      4224
99      4753
100     6462
101     3617
Name: income, Length: 102, dtype: int64
```

2. Multiple linear regression

```
# Split the data into training/testing sets
train_X, test_X, train_y, test_y = \
    train_test_split(df_X, df_y, test_size=0.2)

# Dfine learning model
model =  LinearRegression()

# Train the model using the training sets
model.fit(train_X, train_y)

# Make predictions using the testing set
pred_y = model.predict(test_X)
print(pred_y)
```

```
In [130]: print(pred_y)
[ 9726.86845309  1477.0993624   2856.46952728 11994.73980403
  8286.61100978   147.24557196 12372.18736326  8829.44744925
  1994.42105212  4553.65395295  8486.76942636 14270.62966948
  8766.69756943  4036.98370326 13760.21223628  9194.5642717
  8705.39606181  1970.9237636   5943.05740924 10537.86854041
  2419.7463034 ]
```

2. Multiple linear regression

```
# The coefficients & Intercept
print('Coefficients: {0:.2f},{1:.2f},{2:.2f} Intercept {3:.3f}'\
      .format(model.coef_[0], model.coef_[1], model.coef_[2],\
              model.intercept_))

# The mean squared error
print('Mean squared error: {0:.2f}'.\
      format(mean_squared_error(test_y, pred_y)))

# The coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: %.2f'
      % r2_score(test_y, pred_y))
```

```
Coefficients: 108.88,-48.80,168.59 Intercept -775.183
Mean squared error: 3007520.31
Coefficient of determination: 0.74
```

income = 108.88×education -48.80×women +
168.59× prestige – 775.183

2. Multiple linear regression

education	women	prestige
11.44	8.13	54.1

Income ?

```
# Test single data
my_test_x = np.array([11.44,8.13,54.1]).reshape(1,-1)
my_pred_y = model.predict(my_test_x)
print(my_pred_y)
```

```
In [150]: print(my_pred_y)
[9194.5642717]
```

reshape(1,-1):
행의갯수는 1, 열은 알아서 맞춤

Income = 9194.5642717

3. Logistic regression

- 일반적인 회귀 문제에서는 종속변수가 수치데이터(양적 자료)임
- 예측 해야 할 종속 변수가 수치데이터가 아닌 범주형 데이터 (Yes or No, Patient or Healthy) 일 때 이를 회귀방법으로 해결하고자하는 시도를 '로지스틱 회귀'라고 한다.
- 예) iris 데이터셋에서 4개의 측정 데이터로 부터 품종(Species) 를 예측

* 범주나 그룹을 예측하는 문제를 '분류(classification)' 문제 라고 한다

3. Logistic regression

- Iris 품종 예측

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

꽃받침 꽃잎 품종

- iris dataset 은 3개의 품종이 있음
- Python 의 iris 품종정보는 0,1,2 로 표시됨

3. Logistic regression

```
from sklearn import datasets
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

3. Logistic regression

```
# Load the diabetes dataset
iris_X, iris_y = datasets.load_iris(return_X_y=True)
print(iris_X.shape)   # (150, 4)

# Split the data into training/testing sets
train_X, test_X, train_y, test_y = \
    train_test_split(iris_X, iris_y, test_size=0.3,\
                     random_state=1234)
```

```
In [180]: print(iris_X.shape)   # (150, 4)
(150, 4)
```

random_state=1234

Train, test 를 나눌때 임의로 나누기 때문에 실행 할 때 마다 결과가 달라진다.
실행시 같은 결과가 나오도록 하는 것이 random_state.

3. Logistic regression

```
# Define learning model
model =  LogisticRegression()

# Train the model using the training sets
model.fit(train_X, train_y)

# Make predictions using the testing set
pred_y = model.predict(test_X)
print(pred_y)

# model evaluation: accuracy ##############
acc = accuracy_score(test_y, pred_y)
print('Accuracy : {0:3f}'.format(acc))
```

```
In [179]: print('Accuracy : {0:3f}'.format(acc))
Accuracy : 0.977778
```

$$\text{Accuracy} = \frac{\text{예측과 정답이 일치하는 instance 수}}{\text{전체 test instance 수}}$$

3. Logistic regression

- Note
  - 로지스틱 회귀의 경우에도 종속 변수가 숫자여야 하기 때문에 문자형으로 되어 있는 범주데이터는 숫자(0,1,2,..)로 변환한 후 작업을 해야 한다.

```
from sklearn.preprocessing import LabelEncoder
import numpy as np
number = LabelEncoder()

label_str = np.array(['M','F','M','F','M'])
label_num = number.fit_transform(label_str).astype('int')
print(label_str)
print(label_num)
```

```
In [198]: print(label_str)
['M' 'F' 'M' 'F' 'M']

In [199]: print(label_num)
[1 0 1 0 1]
```

알파벳 순서에 따라 F : 0, M : 1