4주. Regression			
학번	32153180	이름	이상민

※ 이번 실습에 사용된 데이터셋은 공지에 있는 데이터셋 압축파일에 포함되어 있음

BostonHousing 데이터셋은 보스턴 지역의 지역정보 및 평균주택 가격 (medv) 정보를 담고 있다.

BostonHousing dataset을 가지고 단순 선형 회귀 분석을 하고자 한다.

Q1 lstat (소득분위가 하위인 사람들의 비율) 로 mdev (주택가격)을 예측하는 단순 선형회귀 모델을 만드시오 (tain, test 나누지 않음). 모델의 내용을 보이시오

Source code :

```
# 과제에 필요한 패키지
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
# 모델 내용을 보여주기 위한 statsmodels package
import statsmodels.api as sm

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_squared_error, accuracy_score
from sklearn.model_selection import train_test_split

boston = pd.read_csv('C:/Users/sangmin/Desktop/학교생활/4-2/딥러닝클라우드/dataset/BostonHousing.csv')
lstat = np.array(boston["lstat"]).reshape(506, 1)
medv = np.array(boston["medv"]).reshape(506, 1)

model = LinearRegression()
model.fit(lstat, medv)

sm_model = sm.OLS(medv, sm.add_constant(lstat)).fit()
sm_model.summary()
```

실행화면 캡쳐:

```
In [18]: model.fit(lstat, medv)
Out[18]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)

In [19]: sm_model = sm.OLS(medv, sm.add_constant(lstat)).fit()

In [20]: sm_model.summary()
Out[20]:
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.544
Model:                            OLS   Adj. R-squared:                  0.543
Method:                 Least Squares   F-statistic:                     601.6
Date:                Sat, 26 Sep 2020   Prob (F-statistic):           5.08e-88
Time:                        15:12:15   Log-Likelihood:                -1641.5
No. Observations:                 506   AIC:                             3287.
Df Residuals:                     504   BIC:                             3295.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         34.5538      0.563     61.415      0.000      33.448      35.659
x1            -0.9500      0.039    -24.528      0.000      -1.026      -0.874
==============================================================================
Omnibus:                      137.043   Durbin-Watson:                   0.892
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              291.373
Skew:                           1.453   Prob(JB):                     5.36e-64
Kurtosis:                       5.319   Cond. No.                         29.7
==============================================================================
```

Q2. 모델에서 만들어진 회귀식을 쓰시오 (mdev = W x lstat + b 의 형태)

```
In [21]:
    ...: print('Coefficients: {0:.2f}, Intercept: {1:.3f}'\
    ...:       .format(model.coef_[0][0], model.intercept_[0]))
    ...: print('medv = {0:.2f} * lstat + {1:.3f}'\
    ...:       .format(model.coef_[0][0], model.intercept_[0]))
Coefficients: -0.95, Intercept: 34.554
medv = -0.95 * lstat + 34.554
```

Q3. 회귀식을 이용하여 lstat 의 값이 각각 2.0, 3.0, 4.0, 5.0 일 때 mdev 의 값을 예측하여 제시하시오.

Source code :

```
test = np.array([2.0, 3.0, 4.0, 5.0]).reshape(4, -1)

for i in range(4):
    print('lstat : {0} -> medv predict value : {1:.3f}'\
      .format(test[i][0], model.predict(test)[i][0]))
```

실행화면 캡쳐:

```
In [26]:
    ...: for i in range(4):
    ...:     print('lstat : {0} -> medv predict value : {1:.3f}'\
    ...:       .format(test[i][0], model.predict(test)[i][0]))
lstat : 2.0 -> medv predict value : 32.654
lstat : 3.0 -> medv predict value : 31.704
lstat : 4.0 -> medv predict value : 30.754
lstat : 5.0 -> medv predict value : 29.804
```

Q4. 데이터셋의 모든 lstat 값을 회귀식에 넣어 mdev 의 값을 예측 한 뒤 mean square error를 계산하여 제시하시오

Source code :

```
pred = model.predict(lstat)
print('Mean Squared Error: {:.2f}'\
     .format(mean_squared_error(medv, pred)))
```

실행화면 캡쳐:

```
In [29]:
    ...: pred = model.predict(lstat)
    ...: print('Mean Squared Error: {:.2f}'\
    ...:       .format(mean_squared_error(medv, pred)))
Mean Squared Error: 38.48
```

BostonHousing dataset을 가지고 다중 선형 회귀 분석을 하고자 한다.

Q5. lstat (소득분위가 하위인 사람들의 비율), ptratio(초등교사비율), tax(세금), rad(고속도로접근성)로 mdev (주택가격)을 예측하는 단순 선형회귀 모델을 만드시오 (tain, test 나누지 않음). 모델의 내용을 보이시오

Source code :

```
boston_X = boston[['lstat', 'ptratio', 'tax', 'rad']]
boston_y = boston["medv"]

model = LinearRegression()
model.fit(boston_X, boston_y)

sm_model = sm.OLS(boston_y, sm.add_constant(boston_X)).fit()
sm_model.summary()
```

실행화면 캡쳐:

```
In [47]: sm_model = sm.OLS(boston_y, sm.add_constant(boston_X)).fit()

In [48]: sm_model.summary()
Out[48]:
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                   medv   R-squared:                       0.623
Model:                            OLS   Adj. R-squared:                  0.620
Method:                 Least Squares   F-statistic:                     207.2
Date:                Sat, 26 Sep 2020   Prob (F-statistic):          9.65e-105
Time:                        15:22:24   Log-Likelihood:                -1593.2
No. Observations:                 506   AIC:                             3196.
Df Residuals:                     501   BIC:                             3218.
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         58.5457      2.412     24.271      0.000      53.807      63.285
lstat         -0.8141      0.043    -19.059      0.000      -0.898      -0.730
ptratio       -1.2304      0.134     -9.168      0.000      -1.494      -0.967
tax           -0.0151      0.004     -4.018      0.000      -0.023      -0.008
rad            0.3317      0.071      4.703      0.000       0.193       0.470
==============================================================================
Omnibus:                      145.710   Durbin-Watson:                   1.000
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              369.178
Skew:                           1.442   Prob(JB):                     6.82e-81
Kurtosis:                       6.032   Cond. No.                     4.24e+03
==============================================================================
```

Q6. 모델에서 만들어진 회귀식을 쓰시오

```
In [49]:
    ...: print('Coefficients: {0:.2f}, {1:.2f}, {2:.2f}, {3:.2f} / Intercept {4:.3f}'\
    ...:       .format(model.coef_[0], model.coef_[1], model.coef_[2], model.coef_[3], model.intercept_))
    ...: print('medv = {0:.2f}*lstat + {1:.2f}*ptratio + {2:.2f}*tax + {3:.2f}*rad + {4:.3f}'\
    ...:       .format(model.coef_[0], model.coef_[1], model.coef_[2], model.coef_[3], model.intercept_))
Coefficients: -0.81, -1.23, -0.02, 0.33 / Intercept 58.546
medv = -0.81*lstat + -1.23*ptratio + -0.02*tax + 0.33*rad + 58.546
```

Q7. lstat, ptratio, tax, rad 의 값이 다음과 같을 때 mdev 의 예측값을 보이시오.

lstat	ptratio	tax	rad
2.0	14	296	1
3.0	15	222	2
4.0	15	250	3

Source code :

```
test = np.array([[2.0, 14, 296, 1], [3.0, 15, 222, 2], [4.0, 15, 250, 3]]).reshape(3, -1)

for i in range(3):
    print('lstat : {0}, ptratio : {1}, tax : {2}, rad : {3} -> medv predict value : {4:.3f}'\
      .format(test[i][0],   test[i][1],   test[i][2],   test[i][3], model.predict(test)[i]))
```

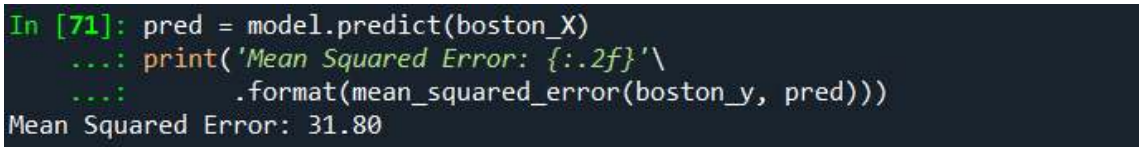
실행화면 캡쳐:

```
In [70]:
    ...: test = np.array([[2.0, 14, 296, 1], [3.0, 15, 222, 2], [4.0, 15, 250, 3]]).reshape(3, -1)
    ...:
    ...: for i in range(3):
    ...:     print('lstat : {0}, ptratio : {1}, tax : {2}, rad : {3} -> medv predict value : {4:.3f}'\
    ...:       .format(test[i][0], test[i][1], test[i][2], test[i][3], model.predict(test)[i]))
lstat : 2.0, ptratio : 14.0, tax : 296.0, rad : 1.0 -> medv predict value : 35.548
lstat : 3.0, ptratio : 15.0, tax : 222.0, rad : 2.0 -> medv predict value : 34.954
lstat : 4.0, ptratio : 15.0, tax : 250.0, rad : 3.0 -> medv predict value : 34.049
```

Q8. 데이터셋의 모든 lstat, ptratio, tax, rad 값을 회귀식에 넣어 mdev 의 값을 예측한 뒤 mean square error를 계산하여 제시하시오.

```
pred = model.predict(boston_X)
print('Mean Squared Error: {:.2f}'\
      .format(mean_squared_error(boston_y, pred)))
```

실행화면 캡쳐:



Q9. lstat 하나만 가지고 모델을 만든 경우와 4개 변수를 가지고 모델을 만든 경우 어느쪽이 더 좋은 모델이라고 할수 있는가? 그 이유는?

- lstat 하나만 가지고 모델을 만든 경우 MSE(mean squared error) 38.48인 반면 4개의 변수를 갖고 모델을 만든 경우 MSE는 31.80이다. MSE가 낮을수록 정확한 모델이기 때문에 후자를 더 좋은 모델이라고 볼 수 있을 것 같다. 보통 다중 선형회귀가 다양한 변수를 사용하여 모델을 만들기 때문에 높은 정확도를 보이지만, 이 예제에서는 단순 선형회귀 모델보다 정확도가 낮은 것으로 보아 변수들이 종속 변수 medv를 예측하는 데 많은 기여를 하지 못한 것 같다.

ucla_admit.csv 파일은 미국 UCLA 의 대학원 입학에 대한 정보를 담고 있다. 컬럼(변수)에 대한 설명은 다음과 같다.
admit : 합격여부 (1:합격, 0:불합격)
gre : GRE 점수
gpa : GPA 점수
rank : 성적 석차

이 데이터셋에 대해 다음의 문제를 해결하시오

Q10. **gre**, **gpa**, **rank**를 가지고 합격여부를 예측하는 logistic regression 모델을 만드시오. (train, test를 나누되 test 의 비율은 30% 로 하고 random_state 는 1234 로 한다)

```
ucla = pd.read_csv('C:/Users/sangmin/Desktop/학교생활/4-2/딥러닝클라우드
/dataset/ucla_admit.csv')
ucla_X = ucla[["gre", "gpa", "rank"]]
ucla_y = ucla["admit"]

train_X, test_X, train_y, test_y = \
    train_test_split(ucla_X, ucla_y, test_size=0.3, random_state=1234)

model = LogisticRegression()
model.fit(train_X, train_y)
```

실행화면 캡쳐:

```
In [72]:
    ...: ucla = pd.read_csv('C:/Users/sangmin/Desktop/학교생활/4-2/딥러닝클라우드/dataset/ucla_admit.csv')
    ...: ucla_X = ucla[["gre", "gpa", "rank"]]
    ...: ucla_y = ucla["admit"]
    ...:
    ...: train_X, test_X, train_y, test_y = \
    ...:     train_test_split(ucla_X, ucla_y, test_size=0.3, random_state=1234)
    ...:
    ...: model = LogisticRegression()
    ...: model.fit(train_X, train_y)
Out[72]:
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

Q11. 모델을 테스트 하여 training accuracy 와 test accuracy를 보이시오

```
train_pred = model.predict(train_X)
test_pred = model.predict(test_X)
train_acc = accuracy_score(train_y, train_pred)
test_acc = accuracy_score(test_y, test_pred)
print('Train accuracy: {0:.3f}'.format(train_acc))
print('Test accuracy: {0:.3f}'.format(test_acc))
```

실행화면 캡쳐:

```
In [73]:
    ...: train_pred = model.predict(train_X)
    ...: test_pred = model.predict(test_X)
    ...: train_acc = accuracy_score(train_y, train_pred)
    ...: test_acc = accuracy_score(test_y, test_pred)
    ...: print('Train accuracy: {0:.3f}'.format(train_acc))
    ...: print('Test accuracy: {0:.3f}'.format(test_acc))
Train accuracy: 0.671
Test accuracy: 0.742
```

Q12. gre, gpa, rank 가 다음과 같을 때 합격 여부를 예측하여 보이시오

gre	gpa	rank
400	3.5	5
550	3.8	2
700	4.0	2

```
test = np.array([[400, 3.5, 5], [550, 3.8, 2], [700, 4.0, 2]]).reshape(3, -1)

for i in range(3):
    print('gre : {0}, gpa : {1}, rank : {2} -> admit predict value : {3}'.format(test[i][0], test[i][1], test[i][2], model.predict(test)[i]))
```

실행화면 캡쳐:

```
In [76]:
    ...: test = np.array([[400, 3.5, 5], [550, 3.8, 2], [700, 4.0, 2]]).reshape(3, -1)
    ...:
    ...: for i in range(3):
    ...:     print('gre : {0}, gpa : {1}, rank : {2} -> admit predict value : {3}'\
    ...:         .format(test[i][0], test[i][1], test[i][2], model.predict(test)[i]))
gre : 400.0, gpa : 3.5, rank : 5.0 -> admit predict value : 0
gre : 550.0, gpa : 3.8, rank : 2.0 -> admit predict value : 0
gre : 700.0, gpa : 4.0, rank : 2.0 -> admit predict value : 0
```

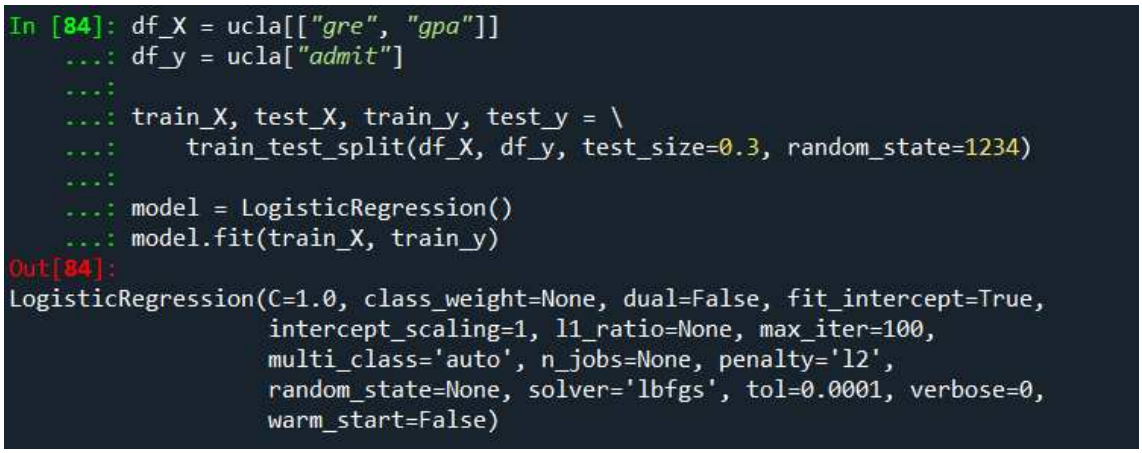
Q13.이번에는 **gre, gpa만 가지고 합격 여부를 예측하는 모델을 만드시오**
(train, test를 나누되 test 의 비율은 30% 로 하고 random_state 는 1234 로 한다)

```
df_X = ucla[["gre", "gpa"]]
df_y = ucla["admit"]

train_X, test_X, train_y, test_y = \
    train_test_split(df_X, df_y, test_size=0.3, random_state=1234)

model = LogisticRegression()
model.fit(train_X, train_y)
```

실행화면 캡쳐:



Q14. 모델을 테스트 하여 training accuracy 와 test accuracy를 보이시오

```
train_pred = model.predict(train_X)
test_pred = model.predict(test_X)
train_acc = accuracy_score(train_y, train_pred)
test_acc = accuracy_score(test_y, test_pred)
print('Train accuracy: {0:.3f}'.format(train_acc))
print('Test accuracy: {0:.3f}'.format(test_acc))
```

실행화면 캡쳐:

```
In [86]:
    ...: train_pred = model.predict(train_X)
    ...: test_pred = model.predict(test_X)
    ...: train_acc = accuracy_score(train_y, train_pred)
    ...: test_acc = accuracy_score(test_y, test_pred)
    ...: print('Train accuracy: {0:.3f}'.format(train_acc))
    ...: print('Test accuracy: {0:.3f}'.format(test_acc))
Train accuracy: 0.625
Test accuracy: 0.825
```

Q15. 3가지 변수로 모델을 만든 경우와 2가지 변수로 모델을 만든 경우를 비교하여 어떤 모델이 더 좋은 모델인지 자신의 의견을 제시하시오

- 3가지 변수로 모델을 만든 경우 test accuracy가 0.742인 반면 2가지 변수로 모델을 만든 경우의 test accuracy는 0.825이다. 단순하게 이 정확도만을 놓고 봤을 때에는 2가지 변수로 모델을 만든 경우가 더 좋은 모델이라고 볼 수 있다. 하지만 두 모델 다 train accuracy보다 test accuracy가 높게 나온 것으로 보아 underfitting이 발생했다고 생각한다. 두 모델은 rank 변수를 독립 변수로 사용했는지 여부의 차이가 있는데, 종속 변수 admit을 예측할 때 rank 변수가 큰 기여를 하지 않는다고 결론지을 수 있다.