1. 데이터 전처리

A. 데이터 전처리

1) 결측값(Missing Value)의 처리

너무 많은 항목이 비어 있는 변수나 너무 많은 항목이 비어 있는 레코드는 그 자체를 삭제

기타 나머지 항목에 대해서는 일반적으로 다음과 같은 값으로 대체

평균값(Mean) / 중앙값(Median) / 최빈치(Mode)

평균값 : (1+2+3+4+4+5+6+6+6+7+8) / 11 = 4.727

중앙값 : 5

최빈치 : 6

2) 정성적 변수의 정량화

각 속성은 단일변수값(atomic value)을 갖도록 수정

정성적 변수의 경우, 0/1의 binary code로 변환해야 추후 해석이 가능

예) 주소의 변환, 성별의 변환 등

3) 이상치(Outlier)의 제거

상식적으로 말이 안되거나 잘못 입력된 것으로 추정되는 변수값을 조정

일괄적으로 상위10%와 하위 10%에 해당하는 값들을 단일값으로 부여하는 경우도 있음

예) 체중 80Kg이상은 무조건 80Kg로,체중 45Kg 이하는 무조건 45Kg으로

4) 새로운 파생변수 개발

기존의 변수를 조합하여 새로운 변수를 개발

본래는 비율변수인 변수를 의미있는 정보로 구간화하여, 새로운 명목변수로 만듦

5) 정규화(Normalization)

모든 입력변수의 값이 최소 0에서 최대 1사이의 값을 갖도록 조정하거나, 평균 0을 갖는 표준정규분포를 갖도록 값을 조정하는 것

정규화 공식 (Min-Max Normalization)

(x-최소값) / (최대값-최소값)

예를 들어 전체 고객 중 체중이 가장 작은 사람이 40Kg, 가장 큰 사람이 120Kg이라고 하면,

40Kg → 0으로 변환

120Kg → 1로 변환

80Kg → (80-40)/(120-40) = 40/80 = 0.5로 변환

6) 자료의 구분

① 과적합화(Overfitting)의 발생 가능성

다음날의 주가지수를 예측하는 모형 A와 B가 있다.

A는 모형을 구축한 날까지의 주가(과거주가)는 99.99% 맞춘다. 그런데, 그 다음

날부터 주가지수를 예측시켜보니 70%를 맞추었다.

모형 B는 과거주가는 83% 맞추는데, 미래주가는 78% 맞춘다.

A, B중 더 잘 구축된 모형은?

② 과적합화의 예방법 : 모형 구축시, hold-out data의 개념을 도입

Hold-out data (검증) : 모형이 일반성을 갖는지 확인하기 위해 남겨두는 unknown data

통계 모형을 구축할 때, 전체 데이터가 100이라면 학습:검증=8:2 혹은 7:3의 비중으로 자료를 미리 나누어 둠

③ 0/1 예측의 경우 0과 1의 비중이 각 데이터셋마다 1:1의 비중이 되도록 섞어야 함

7) 모형에 들어갈 후보 입력변수 선정

카이제곱 검정(Chi-square Test)

독립표본 t검정 (t-Test) – 이분류 모형의 경우에 사용

분산분석 (ANOVA) – 다분류 모형의 경우에 사용

기법	대상변수A	대상변수B	적용 예
카이제곱검정	이산형	이산형	성별과 구매여부사이에 유의한 관계가 있는가?
독립표본t검정	이산형 (2그룹)	연속형	체중과 구매여부 사이에 유의한 관계가 있는가? (구매자와 비구매자의 평균 체중이 크게 다른가?)
일원배치 분산분석	이산형 (3그룹 이상)	연속형	체중과 고객등급 사이에 유의한 관계가 있는가? (고객등급에 따라 평균 체중이 크게 다른가?)

B. 실습예제

1) 결측값 처리

```
import pandas as pd
df=pd.read_csv('c:/data/test/sample.csv')
df
```

```
df.isnull() #결측값 여부 확인
```

```
#pip install missingno
import missingno as msno
import matplotlib.pyplot as plt

msno.matrix(df)
#흰색 - 결측값
#스파크라인(spark line) - 각 샘플의 데이터 완성도를 표현
```

```
msno.bar(df) #필드별 데이터 완성도
```

```
import seaborn as sns
titanic = sns.load_dataset("titanic")
titanic.tail()
```

```
#     survived : 생존 여부
#     pclass : 승객의 클래스
#     sex : 성별. male, female로 표기
#     sibsp : 형제 혹은 자매의 수
#     parch : 부모 혹은 자녀의 수
#     fare : 탑승 요금
#     embarked : 출발지의 고유 이니셜
#     class : 선실의 클래스
#     who : male, female을 man, woman으로 표기
#     adult_male : 성인 남성 인지 아닌지 여부
#     deck : 선실 고유 번호의 가장 앞자리 알파벳(A ~ G)
#     embark_town : 출발지
#     alive : 생존 여부 데이터를 yes 혹은 no로 표기
#     alone : 가족이 없는 경우 True
```

```
msno.matrix(titanic)
#age,deck 등의 필드에 결측값이 많음
```

```
msno.bar(titanic) #필드별로 결측값 확인
```

```
titanic.dropna() #결측값이 있는 모든 행을 삭제
```

```
#결측값이 있는 필드 제거
titanic.dropna(axis=1)
```

```
#7개 이상 비결측 데이터가 있는 필드만 남기고 제거
titanic.dropna(thresh=7, axis=1)
```

```
# 결측값이 50% 이상인 필드를 삭제
titanic = titanic.dropna(thresh=int(len(titanic) * 0.5),
axis=1)
msno.matrix(titanic)
```

```
from sklearn.impute import SimpleImputer
# 결측값을 mean 평균값으로, median 중위수로, most_freqeunt
최빈수로 대체
# 일반적으로 실수형 연속값인 경우 평균 또는 중위수
#   정규분포인 경우 평균을 사용하는 것이 유리하고 비정규분포인
경우 중위수가 유리함
# 카테고리인 경우 최빈값을 사용하는 것이 좋음
imputer = SimpleImputer(strategy="most_frequent")
titanic = pd.DataFrame(imputer.fit_transform(titanic), colu
mns=titanic.columns)
titanic
```

```
#출발지
sns.countplot(titanic.embark_town)
plt.title("embark_town")
```

```
from sklearn.impute import SimpleImputer
#출발지는 범주형이므로 최빈수가 적당함
imputer_embark_town = SimpleImputer(strategy="most_frequent
")
#출발지(fit_transform() 함수에는 2차원 배열을 입력해야 함)
titanic["embark_town"] = imputer_embark_town.fit_transform
(titanic[["embark_town"]])
#출발지의 고유 이니셜
titanic["embarked"] = imputer_embark_town.fit_transform(tit
anic[["embarked"]])

msno.matrix(titanic)
```

```
plt.hist(titanic.age)
plt.title("age")
#비대칭(비정규분포)
```

```
#비대칭인 경우는 중위수를 사용함
imputer_age = SimpleImputer(strategy="median")
titanic["age"] = imputer_age.fit_transform(titanic[["age"
]])

msno.matrix(titanic)
```

2) 스케일링

```
from patsy import demo_data
import pandas as pd
#임의의 실수형 데이터
df = pd.DataFrame(demo_data("x1", "x2", "x3", "x4", "x5"))
df

df.boxplot()

from sklearn.preprocessing import StandardScaler
#평균 0, 표준편차 1이 되도록 스케일링
scaler = StandardScaler()
df2=scaler.fit_transform(df)
df3=pd.DataFrame(df2, columns=df.columns)
df3
df3.boxplot()
```

```
import numpy as np
X = np.arange(7).reshape(7, 1)   #7행 1열로 변환
X
```

```
from sklearn.preprocessing import StandardScaler
#평균 0, 표준편차 1이 되도록 스케일링
scaler = StandardScaler()
X2=scaler.fit_transform(X)
X2
```

```
# 이상치(outlier)가 존재할 경우
X2 = np.vstack([X, [[1000]]]) #배열을 세로로 쌓는 함수
X2
```

```
#아웃라이어가 존재할 경우 스케일링을 했을 때 0에 수렴하지 않고 멀어지는 현상이 발생할 수 있다.
#이것은 기계학습 모형의 예측력을 떨어뜨릴 수 있는 요인이 될 수 있다.
scaler.fit_transform(X2)
```

```
#이상치가 많은 데이터의 경우 RobustScaler를 사용한다.
#중앙값 0, IQR(InterQuartile Range)이 1이 되도록 변환하므로 아웃라이어가 있어도
# 대부분의 데이터가 0 주위로 모이게 된다.
from sklearn.preprocessing import RobustScaler

scaler = RobustScaler()
scaler.fit_transform(X2)
```

3) 범주형 데이터의 전처리

```
# 범주형 데이터(카테고리형 데이터) - 성별, 혈액형, 주소 등

# 기계학습을 위해서는 숫자로 변환해야 함
```

```
import pandas as pd
df1 = pd.DataFrame(["Male", "Female"], columns=["x"])
df1
```

```
#더미변수
df2=pd.get_dummies(df1['x'], prefix='gender')
df2
```

```
df3 = pd.DataFrame(["A", "B", "AB", "O"], columns=["x"])
df3
```

```
df4=pd.get_dummies(df3['x'], prefix='blood')
df4
```

2. 회귀분석

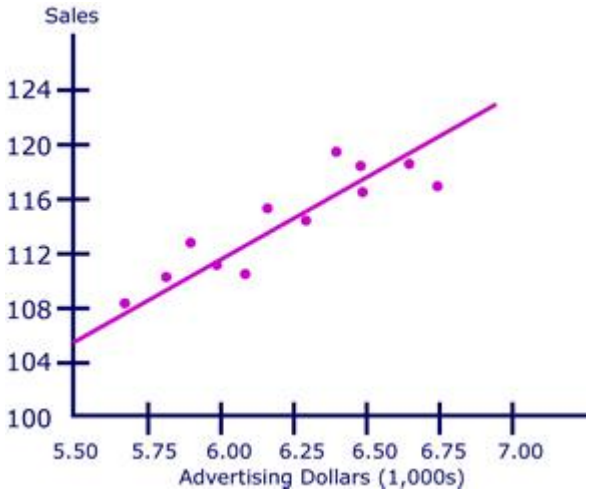
A. 회귀분석(Regression)이란?

1) 독립변수(X)와 종속변수(Y)의 관계식을 구하는 기법

독립변수가 한단위 증가할 때 종속변수에 미치는 영향을 측정하기 위한 통계적 예측 모형

2) 추정 : 회귀식, 회귀계수

3) 검정 : 독립변수의 영향력, 모형의 적합성(R2) 등



B. 회귀분석의 종류

1) 단순회귀분석: 독립변수가 1개인 회귀모형

(예) 기업의 광고집행액(X)을 이용하여 그 기업의 매출액(Y)을 예측하는 모형

2) 중회귀분석: 독립변수가 2개 이상인 회귀모형

(예) 어린이의 연령(X1)과 하루 평균 학습시간(X2)을 이용하여 그 어린이의 성적(Y)을 예측하는 모형

기법	대상변수A	대상변수B	적용 예
카이제곱 검정	이산형	이산형	성별과 결혼유무 사이에 유의한 관계가 있는가?
독립표본 t검정	이산형 (2그룹/독립)	연속형	성별에 따른 평균 취업률의 차이가 있는가?
대응표본 t검정	이산형 (2그룹/Pair)	연속형	보충수업 후 성적의 향상이 있는가?
일원배치 분산분석	이산형 (3그룹 이상)	연속형	거주지역에 따른 평균소득액의 차이가 있는가?
회귀분석	연속형	연속형	가계 수입과 사교육비 지출 사이에 유의한 관계가 있는가?

C. 회귀분석 프로세스

1) 분석을 위한 주제 결정

ex) 교육시간이 직원의 업무 수행에 영향을 주는가?
식사시간이 아이의 두뇌발달에 영향을 주는가?

2) 독립변수와 종속변수 선정

독립변수 : 교육시간
종속변수 : 업무능력

3) 가설 설정

귀무가설(H0) : 교육시간이 업무 능력 점수에 영향을 주지 않는다.
대립가설(H1) : 교육시간이 업무 능력 점수에 영향을 준다.

4) 데이터 수집

5) 데이터 전처리 : 결측값 처리, 데이터 중에서 특이하거나 이상한 데이터의 제거(이상치 제거), 표준화 등

6) 모델을 적용하여 데이터 분석

7) 결과 해석

p-value가 0.05보다 작으면 대립가설(H1) 채택
결정계수가 0~1 사이의 값을 가지며 0.65~0.7 이상이어야 좋은 회귀모형이라고 할 수 있음

D. 단순 회귀분석 실습

1) 전기생산량과 전기사용량 예측

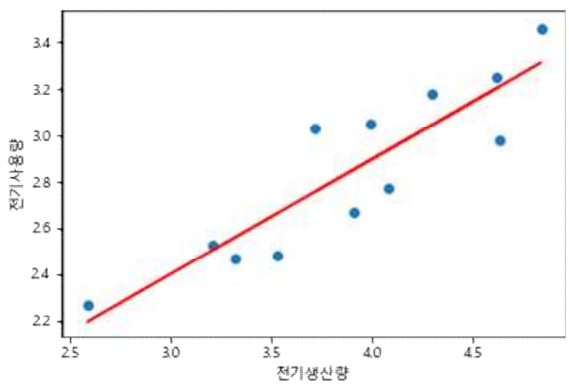
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np

from scipy import stats
from matplotlib import font_manager, rc

#한글 처리를 위해 폰트 설정
font_name = font_manager.FontProperties(fname="c:/Windows/F
onts/malgun.ttf").get_name()
rc('font', family=font_name)
```

```
#회귀분석 : 1.전기생산량과 소비량
#독립변수(전기생산량), 종속변수(전기소비량)
#독립변수가 1개이므로 단순회귀분석(선형회귀분석) 사용
#귀무가설 : 전기생산량과 전기소비량 간에 상관관계가 없다.
#대립가설 : 전기생산량과 전기소비량 간에 상관관계가 있다.
#월별 전기생산금액(억원)
X  =[3.52,2.58,3.31,4.07,4.62,3.98,4.29,  4.83,  3.71,  4.61,
3.90,3.20]
#월별 전기 사용량(백만kwh)
y  =[2.48,2.27,2.47,2.77,2.98,3.05,3.18,  3.46,  3.03,  3.25,
2.67,2.53]
#기울기(slope), 절편(intercept),
#상관계수(rvalue), pvalue(예측 불확실성의 정도를 나타내는 값)
#에러의 표준편차(stderr) - 실제값과 예측값의 평균적인 차이
#p-value는 0.05 미만일 때 통계학적으로 유의미
```

```
# linregress(독립변수,종속변수) 선형회귀분석
result=stats.linregress(X, y)
result
#p_value : 9.238421943157891e-05으로 0.05보다 작으므로 통계적으로 유의미함
#귀무가설을 기각하고 대립가설을 채택한다.
#결론 : 전기생산량과 전기소비량 간에 상관관계가 있다.
#상관분석 : 두 변수 간에 어떤 선형적 관계가 있는지 분석
#상관계수(rvalue) : 상관관계의 정도를 파악하는 값( -1에서 1사이)
```

```
LinregressResult(slope=0.4956032360182905,
intercept=0.9195814280689418, rvalue=0.8929235125385305,
pvalue=9.238421943157891e-05, stderr=0.07901935226531728)
```

```
slope, intercept, r_value, p_value, stderr = stats.linregress(X, y)
x1 = np.array(X)
#산점도 그리기
plt.scatter(X,y)
#회귀선 그리기
plt.plot(x1, slope*x1 +intercept, c="red")
plt.xlabel("전기생산량")
plt.ylabel("전기사용량")
```



```
#생산량이 4일 때 전기사용량 예측
4 * slope + intercept
```

```
2.901994372142104
```

2) p-value의 계산

```
from math import sqrt
from numpy import mean
from scipy.stats import sem
from scipy.stats import t

# 독립표본 t검정: 표본으로부터 측정된 분산, 표준편차를 이용하여 두 모집단의 평균의 차이를 검정하는 방법
def independent_ttest(data1, data2):
    # 평균값
    mean1, mean2 = mean(data1), mean(data2)
    # 모집단이 큰 경우 표본을 여러번 추출하게 되는데 추출할 때마다 표본들의 평균값이 달라지게 됨, 표본평균의 표준편차를 표준오차라고 함(sem 함수로 계산)
    se1, se2 = sem(data1), sem(data2)
    # 표본간의 차이에 대한 표준오차 계산
    sed = sqrt(se1**2 + se2**2)
    # t 통계량(t statistic) 계산
    t_stat = (mean1 - mean2) / sed
    # 자유도(degrees of freedom) 계산(샘플개수-1)
    # 표본수가 n개인 표본이 있다면 표본값 중 자유롭게 변할 수 있는 값은 n-1개의 표본
    # 시험 3회 응시, 평균 80점이라고 할 때 2개 시험에서 70점,90점을 받았다면 나머지 1개는 80점이 되어야 함, 표본수가 3이고 자유롭게 변할 수 있는 값은 2
    df = len(data1) + len(data2) - 2
    # p-value 계산(cdf 누적분포함수)
    p = (1.0 - t.cdf(abs(t_stat), df)) * 2.0
    return t_stat, p
```

```
#독립변수(입력값)
X=[3.52, 2.58, 3.31, 4.07, 4.62, 3.98, 4.29, 4.83, 3.71,
4.61, 3.9, 3.2] #전기생산량
#종속변수(출력값)
y=[2.48, 2.27, 2.47, 2.77, 2.98, 3.05, 3.18, 3.46, 3.03,
3.25, 2.67, 2.53] #전기사용량

#95% 신뢰수준
alpha = 0.05
t_stat, p = independent_ttest(X, y)
print('t=%.3f, p-value=%f' % (t_stat, p))

if p > alpha:
    print('p-value > 0.05, 귀무가설 채택, 대립가설 기각')
else:
    print('p-value < 0.05, 귀무가설 기각, 대립가설 채택')

###################################
```

```
#참고
#확률분포: 어떤 사건에 어느 정도의 확률이 할당되었는지를 묘사한 것
#누적분포함수(cumulative distribution function))
#  모든 사건에 대해 구간을 정의하기가 어려우므로 시작점을 마이너스무한대로 설정하고
#  마이너스무한대 ~ -1, 마이너스무한대 ~ 0, 마이너스무한대 ~ 1 식으로 구간을 정의하는 방법
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import scipy as sp

xx = np.linspace(-8, 8, 100)
#print(xx)
#정규분포 생성, loc 분포의 기대값, scale 분포의 표준편차
rv = sp.stats.norm(loc=1, scale=2)
#누적분포함수
cdf = rv.cdf(xx)
#확률밀도함수(누적분포함수를 미분한 함수)
pdf = rv.pdf(xx)

print(xx[:5])
print(cdf[:5])
plt.plot(xx, pdf)
plt.plot(xx, cdf)
plt.show()
```

3) 회귀함수

```
from sklearn.datasets import make_regression
#가상의 회귀분석 데이터 생성 함수
#n_features : 독립변수의 갯수, bias: y절편,
#noise: y의 표준편차
#coef: 회귀계수(기울기) 사용 여부, True이면 w값을 리턴함
X, y, w = make_regression(
    n_samples=50, n_features=1, bias=100, noise=10,
    random_state=0, coef=True
)
print(X[:5].flatten())
print(y[:5])
print(w)
```

```
import numpy as np
xx = np.linspace(-3, 3, 100) # -3~3, 100등분
y0 = w * xx + 100
```

```
%matplotlib inline
import matplotlib.pyplot as plt
plt.plot(xx, y0, "r-")
plt.scatter(X, y, s=100)
plt.xlabel("x")
plt.ylabel("y")
plt.title("regression")
```

```
from scipy import stats
stats.linregress(X.flatten(),y.flatten())
```

4) 최소자승법(OLS)

실제값과 예측값의 차 : 잔차(Residual)

잔차의 제곱의 합을 최소로 하는 방법

```
#OLS, Ordinary Least Squares
from sklearn.datasets import make_regression

bias = 100
X, y, w = make_regression(
    n_samples=200,   n_features=1,   bias=bias,   noise=10,
coef=True, random_state=1
)
print(X[:5])
print(y[:5])
print(w)
```

```
#사이킷런 패키지의 회귀분석 함수
from sklearn.linear_model import LinearRegression

model = LinearRegression().fit(X, y)
#절편(상수항), 가중치(기울기)
print(model.intercept_, model.coef_)
```

```
#새로운 입력값에 대한 예측(2차원 배열로 입력해야 함)
model.predict([[-2], [-1], [0], [1], [2]])
```

```
print(X.flatten()[:5])
print(y[:5])
```

```
import pandas as pd
df = pd.DataFrame({'x':X.flatten(), 'y':y})
df
```

```
import statsmodels.api as sm

X = df[["x"]]
y = df[["y"]]

#최소자승법(OLS, Ordinary Least Squares)
model = sm.OLS(y, X)
result = model.fit()
result.summary()
```

```
#새로운 값 예측
result.predict([-2, -1, 0, 1, 2])
```

```
#가중치(기울기)
result.params
```

```
#잔차(실제값과 예측값의 차이)
result.resid
```

```
#잔차 벡터 그래프
%matplotlib inline
import matplotlib.pyplot as plt

result.resid.plot(style="o")
plt.xlabel("index")
plt.ylabel("Residual")
```

5) 회귀분석 모형의 성능

```
from sklearn.datasets import make_regression
import statsmodels.api as sm
import pandas as pd

X, y, coef = make_regression(
    n_samples=100,   n_features=1,   noise=30,   coef=True,
random_state=0)
dfX = pd.DataFrame(X, columns=["X"])
#dfX = sm.add_constant(dfX0)
dfy = pd.DataFrame(y, columns=["Y"])
df = pd.concat([dfX, dfy], axis=1)

model = sm.OLS.from_formula("Y ~ X", data=df)
result = model.fit()
result.predict(dfX)
```

```
# Total Sum of Square(종속변수 y의 분산)
print("TSS = ", result.uncentered_tss)
# Explained Sum of Square(예측값의 분산)
print("ESS = ", result.mse_model)
# Residual Sum of Square(잔차의 분산, 오차의 크기)
#   0에 가까울수록 좋은 모형
print("RSS = ", result.ssr)
# 결정계수: 모형의 설명력(0~1 사이의 값)
# 1- RSS/TSS=ESS/TSS
print("R squared=", result.rsquared)
```

```
print(result.summary())
```

6) 온도에 따른 오존량 예측

```
from scipy import stats
import pandas as pd
# 1973년 뉴욕의 공기의 질을 측정한 데이터셋

#회귀분석 예제2 : 오존 데이터셋(온도에 따른 오존량 예측)
#독립변수: 온도, 종속변수: 오존량
#귀무가설 : 온도가 오존량에 영향을 미치지 않는다.
#대립가설 : 온도가 오존량에 영향을 미친다.
# csv 파일을 로딩
df = pd.read_csv("c:/data/ozone/ozone.csv")
 # 데이터의 컬럼명 변경
print(df.head())
```

```
   Ozone  Solar.R  Wind  Temp  Month  Day
0   41.0    190.0   7.4    67      5    1
1   36.0    118.0   8.0    72      5    2
2   12.0    149.0  12.6    74      5    3
3   18.0    313.0  11.5    62      5    4
4    NaN      NaN  14.3    56      5    5
```

```
#결측값이 있는 행 제거
df2=df.dropna(axis=0)
df2.head()
```

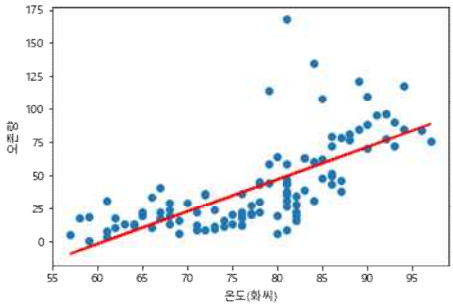
```
#기울기(slope), 절편(intercept), 상관계수(rvalue), p-value(예
측 불확실성의 정도를 나타내는 값)
#에러의 표준편차(stderr)
#p-value는 0.05 미만일 때 통계적으로 유의함
x2 = df2["Temp"].values
```

```
print(x2)
y2 = df2["Ozone"].values
print(y2)
```

```
result=stats.linregress(x2, y2)
result
#모형의 적합도 :  R2값이 0이 아니므로 추정하는 회귀함수가 유의미하다.
# R2 값이 0.69  이 모델은 69%의 설명력이 있음
#귀무가설 : 온도가 오존량에 영향을 미치지 않는다.
#대립가설 : 온도가 오존량에 영향을 미친다.
#p_value : 1.552677229392932e-17는 0.05보다 작으므로 통계적으로 유의함
# 따라서 귀무가설을 기각하고 대립가설을 채택한다.
#결론 : 온도가 오존량에 영향을 미친다.
```

LinregressResult(slope=2.439109905529362, intercept=-147.64607238059494, rvalue=0.6985414096486389, pvalue=1.552677229392932e-17, stderr=0.23931937849409174)

```
import matplotlib.pyplot as plt

slope, intercept, r_value, p_value, stderr = stats.linregress(x2, y2)
#산점도 그리기
plt.scatter(x2,y2)
#회귀선 그리기
plt.plot(x2, slope*x2 +intercept, c="red")
plt.xlabel("Temp")
plt.ylabel("Ozone")
```




```
#온도가 화씨 80도일 때 오존량 예측
80 * slope + intercept
```

```
47.48272006175401
```

7) 붓꽃 품종

```
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from sklearn import datasets
```

```
# 붓꽃 데이터 로딩
iris = datasets.load_iris()
print(iris)
```

```
#변수명 확인
print(iris['feature_names'])
```

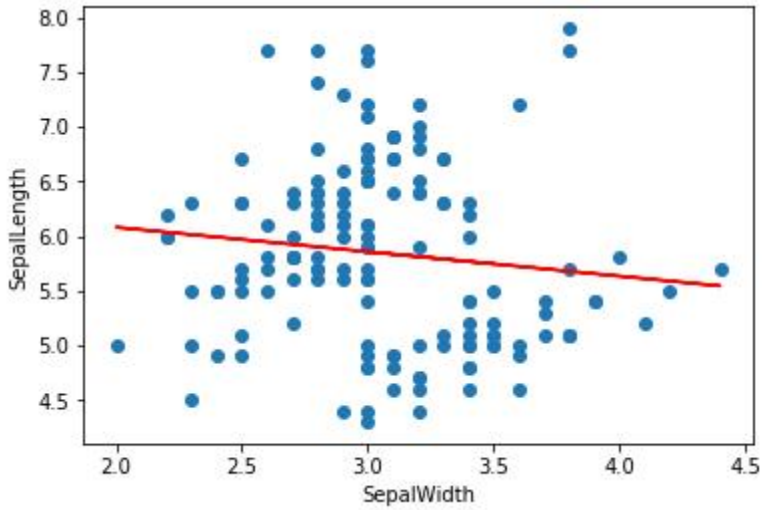
```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',
'petal width (cm)']
```

```
# SepalWidth(꽃받침의 너비)로 SepalLength(꽃받침의 길이)를 예측
# 귀무가설 : 꽃받침의 너비와 꽃받침의 길이는 상관관계가 없다.
# 대립가설 : 꽃받침의 너비와 꽃받침의 길이는 상관관계가 있다.
X = iris.data[:, 1]
y = iris.data[:, 0]
```

```
result=stats.linregress(X, y)
result
# pvalue가 0.05보다 크므로 모델이 유의하지는 않음, 모델의 설명력 -11.7%
# 결론 : 꽃받침의 너비와 꽃받침의 길이는 상관관계가 없다.
```

```
LinregressResult(slope=-0.2233610611298999,
intercept=6.526222550894481, rvalue=-0.11756978413300208,
pvalue=0.15189826071144782, stderr=0.15508092994240763)
```

```
slope, intercept, r_value, p_value, stderr = stats.linregre
ss(X, y)
x1 = np.array(X)
#산점도 그리기
plt.scatter(X,y)
#회귀선 그리기
plt.plot(x1, slope*x1 +intercept, c="red")
plt.xlabel("SepalWidth")
plt.ylabel("SepalLength")
```

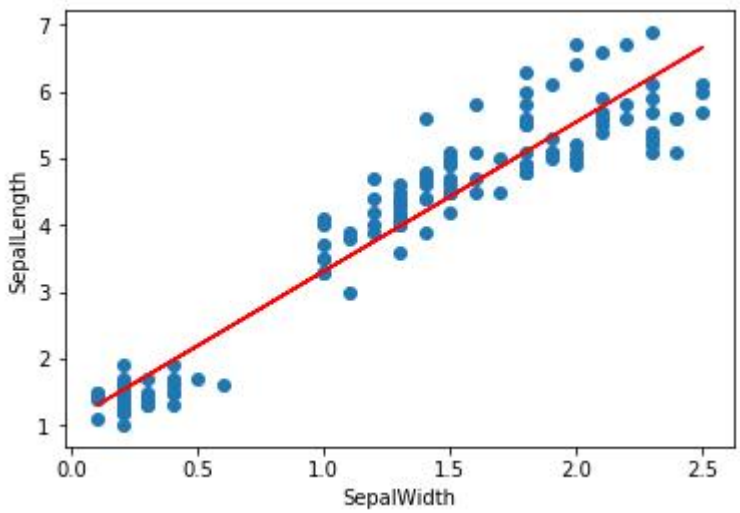



```
# Petal.Width(꽃잎의 너비)로 Petal.Length(꽃잎의 길이)를 예측
# 귀무가설 : 꽃잎의 너비와 꽃잎의 길이는 상관관계가 없다.
# 대립가설 : 꽃잎의 너비와 꽃잎의 길이는 상관관계가 있다.
X = iris.data[:, 3]
y = iris.data[:, 2]
```

```
result=stats.linregress(X, y)
result
# pvalue가 0.05보다 작으므로 모델이 유의함, 모델의 설명력 96.2%
# 결론 : 꽃잎의 너비와 꽃잎의 길이는 강한 양의 상관관계가 있다.
```

```
LinregressResult(slope=2.229940495121865,
intercept=1.08355803285051, rvalue=0.9628654314027963,
pvalue=4.6750039073255014e-86, stderr=0.0513962314651412)
```

```
slope, intercept, r_value, p_value, stderr = stats.linregre
ss(X, y)
x1 = np.array(X)
#산점도 그리기
plt.scatter(X,y)
#회귀선 그리기
plt.plot(x1, slope*x1 +intercept, c="red")
plt.xlabel("SepalWidth")
plt.ylabel("SepalLength")
```



8) 항공운항 지연 시간 예측

```
#항공운항 데이터셋
#분석할 필드가 적은 편
#로딩 시간이 많이 걸려서 가장 레코드수가 적은 1987년 데이터로 실습
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
```

```
df = pd.read_csv("c:/data/airline/1987.csv")

df.head()
```

```
df.info()
```

```
df2=df.loc[:, ["Distance","DepDelay","ArrDelay"]]
```

```
df2.shape
```

```
(1311826, 3)
```

```
#결측값 제거
df3=df2.dropna(axis=0)
```

```
df3.shape
```

```
(1287333, 3)
```

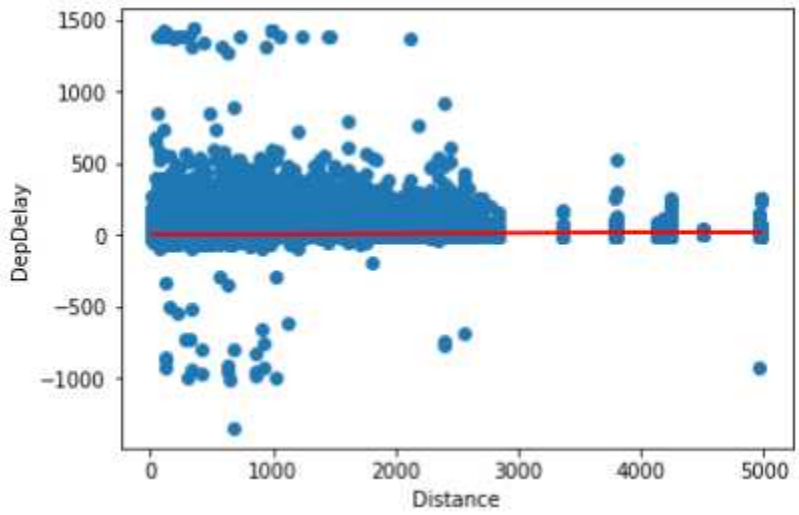
```
X = df3["Distance"] # 운항거리
```

```
y = df3["DepDelay"] # 출발지연시간
print(X.head())
print(y.head())
```

```
#단순회귀분석
#기울기(slope), 절편(intercept), 상관계수(rvalue), p-value(예
측 불확실성의 정도를 나타내는 값)
#에러의 표준편차(stderr)
#p-value는 0.05 미만일 때 통계적으로 유의함
x2 = df3["Distance"].values
y2 = df3["DepDelay"].values
result=stats.linregress(x2, y2)
result
```

```
LinregressResult(slope=0.002611912591979618,
intercept=6.423342199979843, rvalue=0.05505749093590335,
pvalue=0.0, stderr=4.174821057208909e-05)
```

```
slope, intercept, r_value, p_value, stderr = stats.linregre
ss(x2, y2)
#산점도 그리기
plt.scatter(x2,y2)
#회귀선 그리기
plt.plot(x2, slope*x2 +intercept, c="red")
plt.xlabel("Distance")
plt.ylabel("DepDelay")
```

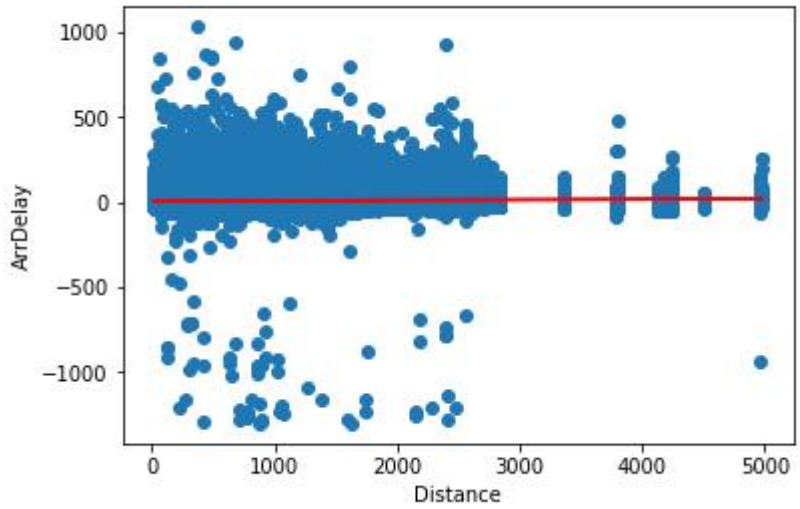



```
#단순회귀분석
#기울기(slope), 절편(intercept), 상관계수(rvalue), p-value(예
측 불확실성의 정도를 나타내는 값)
#에러의 표준편차(stderr)
#p-value는 0.05 미만일 때 통계적으로 유의함
x2 = df3["Distance"].values
y2 = df3["ArrDelay"].values
result=stats.linregress(x2, y2)
result
```

```
LinregressResult(slope=0.0021049954132008894,
intercept=8.200779337409747, rvalue=0.040634730319007245,
pvalue=0.0, stderr=4.5619425571776195e-05)
```

```
slope, intercept, r_value, p_value, stderr =
stats.linregress(x2, y2)
#산점도 그리기
plt.scatter(x2,y2)
#회귀선 그리기
```

```
plt.plot(x2, slope*x2 +intercept, c="red")
plt.xlabel("Distance")
plt.ylabel("ArrDelay")
```



9) 와인 품질 예측

```
#와인데이터셋
#https://archive.ics.uci.edu/ml/machine-learning-databases/
wine-quality/
import pandas as pd
df = pd.read_csv("c:/data/wine/winequality-red.csv",sep=";
")
df.head()
```

```
x = df["alcohol"].values
print(x)
y = df["density"].values
print(y)
```

```
from scipy import stats

result=stats.linregress(x, y)
result
#모형의 적합도 :  rvalue가 0이 아니므로 추정하는 회귀함수가 유의
미하다.
# rvalue:
#귀무가설 : 알코올 농도는 와인의 평균밀도에 영향을 미치지 않는
다.
#대립가설 : 알코올 농도는 와인의 평균밀도에 영향을 미친다.
#p_value : 0.05보다 작으므로 통계적으로 유의함
# 따라서 귀무가설을 기각하고 대립가설을 채택
#결론 :
```

```
LinregressResult(slope=0.36084176533503454,
intercept=1.8749748869971525, rvalue=0.4761663240011359,
pvalue=2.8314769747763594e-91, stderr=0.0166751602954752)
```

```
%matplotlib inline
import matplotlib.pyplot as plt
#한글 처리를 위해 폰트 설정
from matplotlib import font_manager, rc
font_name = font_manager.FontProperties(fname="c:/Windows/F
onts/malgun.ttf").get_name()
rc('font', family=font_name)

slope, intercept, r_value, p_value, stderr = stats.linregre
ss(x, y)
#산점도 그리기
plt.scatter(x,y)
#회귀선 그리기
plt.plot(x, slope*x +intercept, c="red")
plt.xlabel("알코올 농도")
plt.ylabel("와인의 밀도")
```

E. 다중회귀분석 실습

1) Guerry 데이터셋

```
#Guerry 데이터셋
#1830년도 프랑스의 사회인구학적 데이터
# Literacy: 문해율
# Crime_prop: 재산범죄당 인구
# Wealth: 재산세 순위
# Distance: 수도 파리까지의 거리(km)

# url = 'https://vincentarelbundock.github.io/Rdatasets/csv
/HistData/Guerry.csv'
```

```
import pandas as pd
df=pd.read_csv('c:/data/guerry/data.csv')
df
```

```
%matplotlib inline
df.plot()
```

```
#결측값 1개가 제거됨, inplace=True 실행 후 결과값을 현재 변수에
다시 저장
df.dropna(inplace=True)
df.shape
```

```
#변수 4개만 선택
df2 = df[['Crime_prop', 'Literacy', 'Wealth', 'Distance']]
```

```
import statsmodels.formula.api as smf
# Ordinary Least Squares(OLS) 방식
# 잔차(Residual) : 실제값과 예측값의 차
# 잔차의 제곱의 합을 최소로 하는 방식으로 회귀선을 그리는 방식
model = smf.ols(formula='Crime_prop ~ Literacy + Wealth + Distance', data=df2).fit()
model.summary()
```

```
import seaborn as sns
import matplotlib.pyplot as plt
#문해율과 범죄율(음의 상관관계)
#회귀모형그래프
sns.regplot('Literacy', 'Crime_prop', df, color='red')
```

```
#재산세 순위와 범죄율
sns.regplot('Wealth', 'Crime_prop', df, color='blue')
```

```
#수도와의 거리와 범죄율(수도와의 거리가 멀어질수록 범죄율이 높아진다)
sns.regplot('Distance', 'Crime_prop', df, color='green')
```

```
#산점도
df.plot('Literacy', 'Crime_prop', kind='scatter',color='red')
df.plot('Wealth', 'Crime_prop', kind='scatter', color='blue')
df.plot('Distance', 'Crime_prop', kind='scatter', color='green')
```

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
X = df[['Literacy', 'Wealth', 'Distance']]
y = df['Crime_prop']
model.fit(X, y)
```

```
#문해율 80, 재산세 순위 1, 수도까지의 거리 20일 때의 범죄율
literacy = 80
wealth = 1
distance = 20

regionA = [literacy, wealth, distance]
crimeA = model.predict([regionA])
crimeA
#3880명당 1건의 범죄
```

```
#문해율 10, 재산세 순위 70, 수도까지의 거리 500일 때의 범죄율
literacy = 10
wealth = 70
distance = 500

regionB = [literacy, wealth, distance]
crimeB = model.predict([regionB])
crimeB
#10964명당 1건의 범죄
```

```
#A,B 중간 정도의 동네
#문해율 50, 재산세 순위 30, 수도까지의 거리 300일 때의 범죄율
literacy = 50
wealth = 30
distance = 300

regionC = [literacy, wealth, distance]
crimeC = model.predict([regionC])
crimeC
#7073명당 1건의 범죄
```

2) 보스턴 주택 가격 예측

```
#보스턴 주택가격 데이터셋 로딩을 위한 패키지
#sklearn(사이킷런)
from sklearn.datasets import load_boston

#보스턴 주택가격 데이터셋 로딩
boston = load_boston()
#데이터셋의 형태
print(boston.data.shape)
print(type(boston.data))
print(boston.data[:3])
```

```
(506, 13)
```

```
#데이터셋에 대한 설명
print(boston.DESCR)
```

```
#회귀분석 : 단답형
#분류 : 선택형
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.DataFrame(boston.data,
columns=boston.feature_names)
df["price"] = boston.target #주택가격
#산점도 행렬( RM 방의 갯수, AGE 노후화, CRIM 범죄율)
sns.pairplot(df[["price", "RM", "AGE", "CRIM"]])
plt.show()
```

```
from sklearn.linear_model import LinearRegression

model = LinearRegression().fit(boston.data, boston.target)
predicted = model.predict(boston.data)
plt.figure(figsize=(10,6))
plt.scatter(boston.target, predicted)
plt.xlabel("real")
plt.ylabel("predict")
plt.show()
```

```
#13개의 필드, 특성끼리 곱하여 데이터 확장
#범죄율, 고속도로 접근성 => 범죄율과 고속도로 접근성의 곱도 계산
#13개의 특성을 확장하여 91개의 특성을 더하여 총 104개
import mglearn
X, y = mglearn.datasets.load_extended_boston()
print("X.shape: {}".format(X.shape))
```

```
X.shape: (506, 104)
```

종속변수

1978년 보스턴의 주택 가격

506개 타운의 주택 가격 중앙값 (단위 1,000 달러)

독립변수

CRIM: 범죄율

INDUS: 비소매상업지역 면적 비율

NOX: 일산화질소 농도

RM: 주택당 방 수

LSTAT: 인구 중 하위 계층 비율

B: 인구 중 흑인 비율

PTRATIO: 학생/교사 비율

ZN: 25,000 평방피트를 초과하는 거주 지역의 비율

```
    CHAS: 찰스강의 경계에 위치한 경우는 1, 아니면 0
    AGE: 1940년 이전에 건축된 주택의 비율
    RAD: 고속도로까지의 거리
    DIS: 고용지원센터의 거리
    TAX: 재산세율
```

```
boston.feature_names
```

```
array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE',
'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7')
```

```
X = pd.DataFrame(boston.data, columns=boston.feature_names)
y = pd.DataFrame(boston.target, columns=["MEDV"])
df = pd.concat([X, y], axis=1) #가로 방향 연결
df.tail()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.67	22.4
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	9.08	20.6
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	5.64	23.9
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	6.48	22.0
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	7.88	11.9

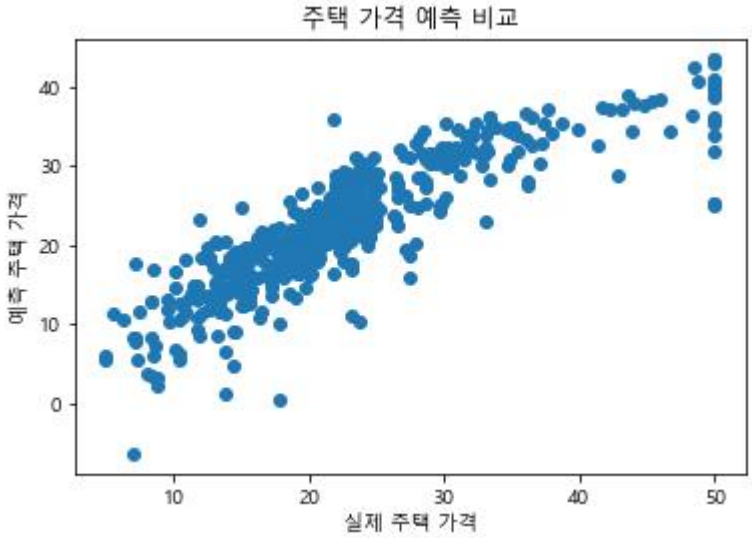
```
from sklearn.model_selection import train_test_split

#데이터셋을 나눔(학습용:검증용 = 7:3)
#random_state : 난수 발생을 위한 seed의 인자값
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=10)
model = LinearRegression().fit(X_train, y_train)
```

```
print("학습용:",model.score(X_train, y_train))
print("검증용:",model.score(X_test, y_test))
```

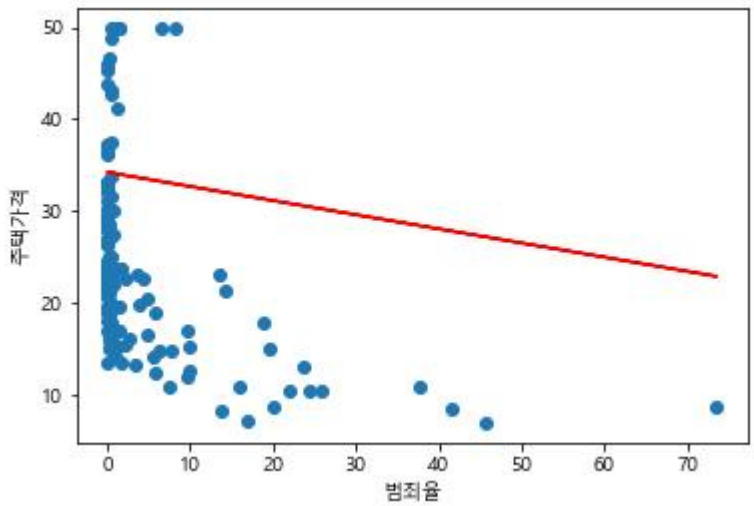
```
#상관계수
coef=model.coef_
#절편
intercept=model.intercept_

print(coef)
print(intercept)
#범죄율이 한단위 증가하면 집값은 약 153 달러 하락한다
#찰스강의 경계에 위치한 경우 집값은 약 1622 달러 상승한다.
#방갯수가 한단위 증가하면 집값은 약 3352 달러 상승한다.
#재산세율이 한단위 증가하면 집값은 약 1223 하락한다.
```

```
[[-1.53776087e-01  6.50159270e-02 -2.42597326e-02
1.62203309e+00
  -1.52999306e+01  3.35196668e+00  1.13259963e-02
-1.54810871e+00
   3.02641886e-01 -1.22317535e-02 -8.11539044e-01
1.29504798e-02
  -5.44861583e-01]]
[34.24483703]
```

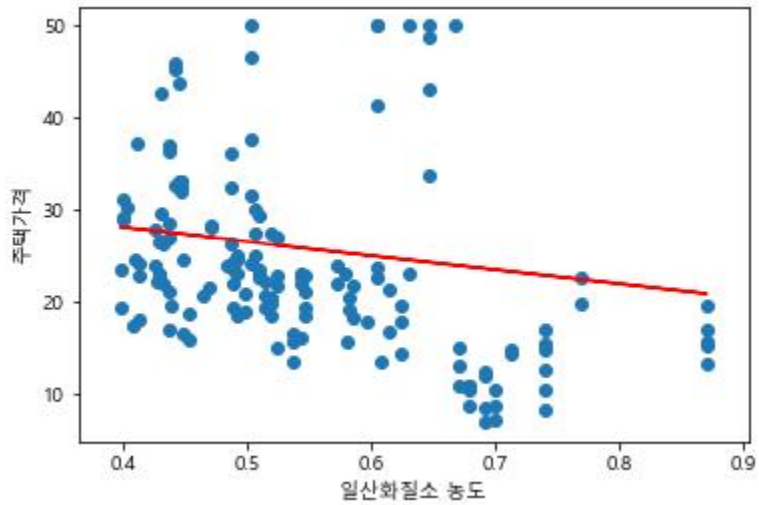
```
pred = model.predict(boston.data)

plt.scatter(boston.target, pred)
plt.xlabel("실제 주택 가격")
plt.ylabel("예측 주택 가격")
plt.title("주택 가격 예측 비교")
plt.show()
```




```
#산점도 그리기
plt.scatter(X_test["CRIM"],y_test)
#회귀선 그리기
plt.plot(X_test["CRIM"], coef[0][0]*X_test["CRIM"] +interce
pt, c="red")
plt.xlabel("범죄율")
plt.ylabel("주택가격")
```



```
#산점도 그리기
plt.scatter(X_test["NOX"],y_test)
#회귀선 그리기
plt.plot(X_test["NOX"], coef[0][4]*X_test["NOX"] +intercep
t, c="red")
plt.xlabel("일산화질소 농도")
plt.ylabel("주택가격")
```




```
#산점도 그리기
plt.scatter(X_test["RM"],y_test)
#회귀선 그리기
plt.plot(X_test["RM"], coef[0][5]*X_test["RM"] +intercept,
c="red")
plt.xlabel("방의 수")
plt.ylabel("주택가격")
```

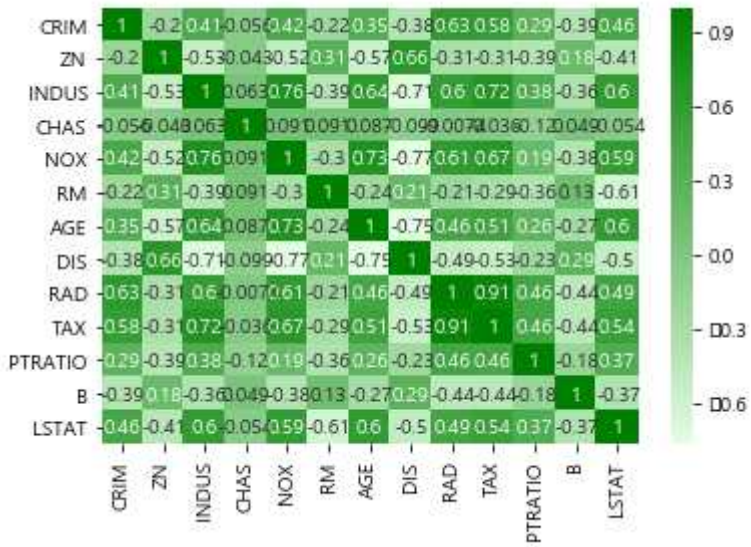
```
import pandas as pd
import statsmodels.api as sm

dfX = pd.DataFrame(boston.data, columns=boston.feature_name
s)
dfy = pd.DataFrame(boston.target, columns=["MEDV"])

model_boston2 = sm.OLS(dfy, dfX)
result_boston2 = model_boston2.fit()

print("회귀계수\n",result_boston2.params)
print("R-squared\n",result_boston2.rsquared) #회귀모델의
설명력 95.8%의 설명력
print("P values\n",result_boston2.pvalues)

#요약 결과 출력
print(result_boston2.summary())
```

```
전진선택법 : 공집합인 상태에서 변수를 1개씩 추가하면서 전체
모델의 성능이 좋아질때까지 변수를 계속 증가시키는 방법
후진제거법 : 전체 변수를 대상으로 미리 정의한 모든 통계
수치(유의수준, 다중공선성 등)를 만족할 때까지 반복적으로 변수를
하나씩 제거하는 방법

변수들 중 탈락시킬 변수 선정
['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS',
'RAD','TAX', 'PTRATIO', 'B', 'LSTAT']
pvalue가 0.05보다 큰 값을 탈락시킨다.
INDUS(2) NOX(4) AGE(6)
INDUS > AGE > NOX
```

```
#변수들의 상관관계
#상관계수 행렬
dfX.corr()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
CRIM	1.000000	-0.200469	0.406583	-0.055892	0.420972	-0.219247	0.352734	-0.379670	0.625505	0.582764	0.289946	-0.385064	0.455621
ZN	-0.200469	1.000000	-0.533828	-0.042697	-0.516604	0.311991	-0.569537	0.664408	-0.311948	-0.314563	-0.391679	0.175520	-0.412995
INDUS	0.406583	-0.533828	1.000000	0.062938	0.763651	-0.391676	0.644779	-0.708027	0.595129	0.720760	0.383248	-0.356977	0.603800
CHAS	-0.055892	-0.042697	0.062938	1.000000	0.091203	0.091251	0.086518	-0.099176	-0.007368	-0.035587	-0.121515	0.048788	-0.053929
NOX	0.420972	-0.516604	0.763651	0.091203	1.000000	-0.302188	0.731470	-0.769230	0.611441	0.668023	0.188933	-0.380051	0.590879
RM	-0.219247	0.311991	-0.391676	0.091251	-0.302188	1.000000	-0.240265	0.205246	-0.209847	-0.292048	-0.355501	0.128069	-0.613808
AGE	0.352734	-0.569537	0.644779	0.086518	0.731470	-0.240265	1.000000	-0.747881	0.456022	0.506456	0.261515	-0.273534	0.602339
DIS	-0.379670	0.664408	-0.708027	-0.099176	-0.769230	0.205246	-0.747881	1.000000	-0.494588	-0.534432	-0.232471	0.291512	-0.496996
RAD	0.625505	-0.311948	0.595129	-0.007368	0.611441	-0.209847	0.456022	-0.494588	1.000000	0.910228	0.464741	-0.444413	0.488676
TAX	0.582764	-0.314563	0.720760	-0.035587	0.668023	-0.292048	0.506456	-0.534432	0.910228	1.000000	0.460853	-0.441808	0.543993
PTRATIO	0.289946	-0.391679	0.383248	-0.121515	0.188933	-0.355501	0.261515	-0.232471	0.464741	0.460853	1.000000	-0.177383	0.374044
B	-0.385064	0.175520	-0.356977	0.048788	-0.380051	0.128069	-0.273534	0.291512	-0.444413	-0.441808	-0.177383	1.000000	-0.366087
LSTAT	0.455621	-0.412995	0.603800	-0.053929	0.590879	-0.613808	0.602339	-0.496996	0.488676	0.543993	0.374044	-0.366087	1.000000

```
import seaborn as sns
plt.figure(figsize=(15,10))
cmap = sns.light_palette("green", as_cmap=True)
sns.heatmap(dfX.corr(), annot=True, cmap=cmap)
plt.show()
```



```
#다중공선성은 VIF 값으로 확인할 수 있다.
#다른 변수에 의존적일수록 VIF가 커진다(다중공선성)
#독립변수가 서로 의존하게 되면 과적합화 문제가 발생하여 모델의
안정성이 떨어질 수 있다.
#VIF(Variance Inflation Factor)

from statsmodels.stats.outliers_influence import variance_i
nflation_factor

vif = pd.DataFrame()
vif["VIF"] = [variance_inflation_factor(
    dfX.values, i) for i in range(dfX.shape[1])]
vif["변수"] = dfX.columns
vif = vif.sort_values("VIF").reset_index(drop=True)
vif
#vif값은 작을수록 좋은데 5보다 큰 값은 다중공선성이 큰 변수라고
볼 수 있다.
```

	VIF	변수
0	1.152952	CHAS
1	2.100373	CRIM
2	2.844013	ZN
3	11.102025	LSTAT
4	14.485758	INDUS
5	14.699652	DIS
6	15.167725	RAD
7	20.104943	B
8	21.386850	AGE
9	61.227274	TAX
10	73.894947	NOX
11	77.948283	RM
12	85.029547	PTRATIO

```
# INDUS 필드 제거
arr=boston.data[:,[0,1,3,4,5,6,7,8,9,10,11,12]]
arr
dfX = pd.DataFrame(arr, columns=['CRIM', 'ZN', 'CHAS',
'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B',
'LSTAT'])
dfy = pd.DataFrame(boston.target, columns=["MEDV"])

model_boston3 = sm.OLS(dfy, dfX)
result_boston3 = model_boston3.fit()

print("회귀계수\n",result_boston3.params)
print("R-squared\n",result_boston3.rsquared) #회귀모델의
설명력 95.8%의 설명력
print("P values\n",result_boston3.pvalues)

#요약 결과 출력
print(result_boston3.summary())
```

```
# AGE 필드 제거
arr=boston.data[:,[0,1,3,4,5,7,8,9,10,11,12]]
arr

dfX = pd.DataFrame(arr, columns=['CRIM', 'ZN', 'CHAS',
'NOX', 'RM', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT'])
dfy = pd.DataFrame(boston.target, columns=["MEDV"])

model_boston3 = sm.OLS(dfy, dfX)
result_boston3 = model_boston3.fit()

print("회귀계수\n",result_boston3.params)
print("R-squared\n",result_boston3.rsquared) #회귀모델의
설명력 95.8%의 설명력
print("P values\n",result_boston3.pvalues)

#요약 결과 출력
print(result_boston3.summary())
```

```
# NOX 필드 제거
arr=boston.data[:,[0,1,3,5,7,8,9,10,11,12]]
arr

dfX = pd.DataFrame(arr, columns=['CRIM', 'ZN', 'CHAS',
'RM', 'DIS', 'RAD','TAX', 'PTRATIO', 'B', 'LSTAT'])
dfy = pd.DataFrame(boston.target, columns=["MEDV"])

model_boston3 = sm.OLS(dfy, dfX)
result_boston3 = model_boston3.fit()

print("회귀계수\n",result_boston3.params)
print("R-squared\n",result_boston3.rsquared) #회귀모델의
설명력 95.8%의 설명력
print("P values\n",result_boston3.pvalues)

#요약 결과 출력
print(result_boston3.summary())
```

```
# PTRATIO  필드를 제거하니 오히려 설명력이 95.7%로 약간
떨어지므로
# PTRATIO 필드부터는 제거하지 않는다.
arr=boston.data[:,[0,1,3,5,7,8,9,11,12]]
arr

dfX = pd.DataFrame(arr, columns=['CRIM', 'ZN', 'CHAS',
'RM', 'DIS', 'RAD','TAX', 'B', 'LSTAT'])
dfy = pd.DataFrame(boston.target, columns=["MEDV"])

model_boston3 = sm.OLS(dfy, dfX)
result_boston3 = model_boston3.fit()

print("회귀계수\n",result_boston3.params)
print("R-squared\n",result_boston3.rsquared) #회귀모델의
설명력 95.7%의 설명력
print("P values\n",result_boston3.pvalues)

#요약 결과 출력
print(result_boston3.summary())
```

결론

수정 R 제곱 : 0.958에서 개선되지 않았다.
개별변수의 p value : 모든 값이 유의함(유의하지 않은 변수 3개가 제거됨)

3) 다중공선성과 변수선택

```
from statsmodels.datasets.longley import load_pandas
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
#미국의 거시경제지표 데이터셋
#     TOTEMP - Total Employment
#     GNPDEFL - GNP deflator
#     GNP - GNP
#     UNEMP - Number of unemployed
#     ARMED - Size of armed forces
#     POP - Population
#     YEAR - Year (1947 - 1962)

dfy = load_pandas().endog
dfX = load_pandas().exog
df = pd.concat([dfy, dfX], axis=1)
df.head()
```

```
sns.pairplot(dfX)
plt.show()
#독립변수들간의 상관관계가 강한 데이터셋
```

```
dfX.corr()
```

```
%matplotlib inline
cmap = sns.light_palette("darkgray", as_cmap=True)
sns.heatmap(dfX.corr(), annot=True, cmap=cmap)
```

```
from sklearn.model_selection import train_test_split
import statsmodels.api as sm

def get_model1(seed):
    df_train, df_test = train_test_split(df, test_size=0.5,
random_state=seed)
    model = sm.OLS.from_formula("TOTEMP ~ GNPDEFL + POP +
GNP + YEAR + ARMED + UNEMP", data=df_train)
    return df_train, df_test, model.fit()



df_train, df_test, result1 = get_model1(3)
result1.summary()
#다중공선성 문제로 인하여 조건수(conditional number)가 증가한
다.
```

```
# Total Sum of Square(종속변수 y의 분산)
# Residual Sum of Square(잔차의 분산, 오차의 크기)
def calc_r2(df_test, result):
    target = df.loc[df_test.index].TOTEMP #실제값
    predict_test = result.predict(df_test) #예측값
    RSS = ((predict_test - target)**2).sum() #잔차의 분산
    TSS = ((target - target.mean())**2).sum() #실제값의 분산
    return 1 - RSS / TSS #결정계수


train1 = []
test1 = []
for i in range(10):
    df_train, df_test, result = get_model1(i)
    train1.append(calc_r2(df_train, result))
    test1.append(calc_r2(df_test, result))
#과적합 문제가 있음
print(train1)
print(test1)
```

```
#다중 공선성을 해결하는 방법

#변수 선택으로 의존적인 변수 제거
#PCA(주성분분석)
#정규화
```

```
#VIF(Variance Inflation Factor) : 다른 변수에 의존적일수록 커진다.
```

```
from statsmodels.stats.outliers_influence import variance_i
nflation_factor

vif = pd.DataFrame()
vif["VIF Factor"] = [variance_inflation_factor(
    dfX.values, i) for i in range(dfX.shape[1])]
vif["features"] = dfX.columns
vif
```

```
# VIF와 pvalue가 높은 GNPDEFL,POP,YEAR 변수들을 제거하고 스케
일링 처리한 모형
def get_model2(seed):
    df_train, df_test = train_test_split(df, test_size=0.5,
random_state=seed)
    model = sm.OLS.from_formula("TOTEMP ~ scale(GNP) +
scale(ARMED) + scale(UNEMP)", data=df_train)
    return df_train, df_test, model.fit()


df_train, df_test, result2 = get_model2(3)
print(result2.summary())
# 성능이 향상됨
```

```
#모형의 정확도가 개선됨, 과적합이 해소됨
test2 = []
for i in range(10):
    df_train, df_test, result = get_model2(i)
    test2.append(calc_r2(df_test, result))

test2
```

```
#다중공선성 제거 전
plt.figure(figsize=(15,10))
plt.subplot(121)
plt.plot(test1, 'ro', label="Test")
plt.hlines(result1.rsquared, 0, 9, label="Train")
plt.legend()
plt.xlabel("seed")
plt.ylabel("R-squared")
plt.ylim(0.5, 1.2)

#다중공선성 제거 후
plt.subplot(122)
plt.plot(test2, 'ro', label="Test")
plt.hlines(result2.rsquared, 0, 9, label="Train")
plt.legend()
plt.xlabel("seed")
plt.ylabel("R-squared")
plt.ylim(0.5, 1.2)

plt.tight_layout()
plt.show()
```

```
from sklearn.datasets import load_boston
import numpy as np

boston = load_boston()

dfX0 = pd.DataFrame(boston.data, columns=boston.feature_nam
es)

from patsy import dmatrix

formula = "scale(CRIM) + scale(I(CRIM ** 2)) + " + \
    "scale(ZN) + scale(I(ZN ** 2)) + scale(INDUS) + " + \
    "scale(NOX) + scale(RM) + scale(AGE) + " + \
    "scale(np.log(DIS)) + scale(RAD) + scale(TAX) + " + \
    "scale(np.log(PTRATIO)) + scale(B) + scale(np.log(LSTA
T)) + CHAS"
dfX = dmatrix(formula, dfX0, return_type="dataframe")
dfy = pd.DataFrame(boston.target, columns=["MEDV"])
```

```
dfX.corr()
```

```
plt.figure(figsize=(15,10))
cmap = sns.light_palette("black", as_cmap=True)
sns.heatmap(dfX.corr(), annot=True, fmt='3.1f', cmap=cmap)
plt.show()
```

```
vif = pd.DataFrame()
vif["VIF Factor"] = [variance_inflation_factor(
    dfX.values, i) for i in range(dfX.shape[1])]
vif["features"] = dfX.columns
vif = vif.sort_values("VIF Factor").reset_index(drop=True)
vif
```

```
#VIF가 높은 값들을 제거하기 전의 모형
model_boston1 = sm.OLS(np.log(dfy), dfX)
result_boston1 = model_boston1.fit()
print(result_boston1.summary())
```

```
#VIF가 높은 변수들을 제거하고 최종적으로 선택한 독립변수들로
만든 모형
cols = ["Intercept", "CHAS", "scale(B)", "scale(CRIM)", "sc
ale(AGE)",
       "scale(np.log(PTRATIO))", "scale(RM)", "scale(np.log
(LSTAT))"]

model_boston2 = sm.OLS(np.log(dfy), dfX[cols])
result_boston2 = model_boston2.fit()
print(result_boston2.summary())
#결정계수: 81.7% => 77.4%
#조건수: 13.7 => 6.71
```

4) 스케일링

```
from sklearn.datasets import load_boston
import pandas as pd

boston = load_boston()

dfX = pd.DataFrame(boston.data, columns=boston.feature_name
s)
dfy = pd.DataFrame(boston.target, columns=["MEDV"])
df = pd.concat([dfX, dfy], axis=1)
```

```
import statsmodels.api as sm

model = sm.OLS.from_formula("MEDV ~ " + "+".join(boston.fea
ture_names), data=df)
result = model.fit()
result.summary()
```

```
#[2] The condition number is large, 1.51e+04. This might indicate that there are
#strong multicollinearity or other numerical problems.

# 조건수(Cond. No.)가 너무 커서 강한 다중공선성(multicollinearity) 또는 다른 수치적 문제가 있을 수 있다.
# 이것은 모형의 예측 정확도가 떨어지는 원인이 될 수 있다.

# 행렬의 조건수란 가장 큰 값과 가장 작은값의 비율을 의미함
# 조건수가 커지는 원인
#   변수들의 단위 차이 : 숫자의 스케일이 크게 달라짐 => 스케일링
#   다중공선성(상관관계가 큰 독립변수들이 있는 경우)
#     => 변수 제거, 주성분분석(pca) 등으로 해결
# 이 데이터는 변수 간의 편차가 큰 데이터셋이다.
```

```
#변수들의 표준편차 비교
dfX.describe().loc["std"]
```

```
%matplotlib inline
dfX.boxplot()
```

```
#Standardization 표준화, 평균 0, 표준편차 1
# 데이터를 0을 중심으로 양쪽으로 분포시키는 방법
# 평균을 기준으로 얼마나 떨어져 있는지를 나타내는 값
# (측정값-평균) / 표준편차
# k-means, svm 등 거리 기반의 모델에서 주로 사용함
# 변수의 단위가 다른 경우 가중치가 부여되어 노이즈가 생길 수 있다
# 1000원과 1달러(가중치가 1000배가 되지 않도록 해야 함)

# 변수들의 평균값 계산
mean_on_train = dfX.mean(axis=0)
# 변수들의 표준편차값 계산
std_on_train = dfX.std(axis=0)

# 값에서 평균을 빼고 표준 편차로 나누면
# 평균 0, 표준편차 1인 데이터로 변환됨
dfX_scaled = (dfX - mean_on_train) / std_on_train
#평균 0, 표준편차 1로 조정됨
print(dfX_scaled.head())
print(dfX_scaled.describe())
```

```
dfX_scaled.boxplot()
```

```
df_scaled = pd.concat([dfX_scaled, dfy], axis=1)
```

```
model = sm.OLS.from_formula("MEDV ~ " + "+".join(boston.fea
ture_names), data=df_scaled)
result = model.fit()
result.summary()
```

```
#스케일링을 한 이후 조건수가 9.82로 감소하였다.
```

5) 교차검증

```
from sklearn.datasets import load_boston
import pandas as pd
import numpy as np

boston = load_boston()
dfX = pd.DataFrame(boston.data, columns=boston.feature_name
s)
dfy = pd.DataFrame(boston.target, columns=["MEDV"])
df = pd.concat([dfX, dfy], axis=1)
#학습용과 검증용을 7:3으로 구분
N = len(df)
ratio = 0.7
np.random.seed(0)
idx_train = np.random.choice(np.arange(N), np.int64(ratio *
N),replace=False)
idx_test = list(set(np.arange(N)).difference(idx_train))

df_train = df.iloc[idx_train]
df_test = df.iloc[idx_test]
```

```
import statsmodels.api as sm

model = sm.OLS.from_formula("MEDV ~ " + "+".join(boston.fea
ture_names), data=df_train)
result = model.fit()
print(result.summary())
```

```
#검증용 데이터셋으로 모형 평가
pred = result.predict(df_test)
# Residual Sum of Square(잔차의 분산, 오차의 크기)
rss = ((df_test.MEDV - pred) ** 2).sum()
# Total Sum of Square(종속변수 y의 분산)
tss = ((df_test.MEDV - df_test.MEDV.mean())** 2).sum()
# 결정계수: 모형의 설명력(0~1 사이의 값)
rsquared = 1 - rss / tss
rsquared
```

```
from sklearn.model_selection import train_test_split

#학습용,검증용으로 구분
df_train, df_test = train_test_split(df, test_size=0.3,
random_state=0)
df_train.shape, df_test.shape
```

```
#학습용X,y 검증용 X,y로 구분
dfX_train, dfX_test, dfy_train, dfy_test = train_test_split
(dfX, dfy, test_size=0.3, random_state=0)
dfX_train.shape, dfy_train.shape, dfX_test.shape, dfy_test.
shape
```

```
from sklearn.model_selection import KFold

scores = np.zeros(5)
cv = KFold(5, shuffle=True, random_state=0)
for i, (idx_train, idx_test) in enumerate(cv.split(df)):
    df_train = df.iloc[idx_train]
    df_test = df.iloc[idx_test]

    model = sm.OLS.from_formula("MEDV ~ " + "+".join(boston.
feature_names), data=df_train)
    result = model.fit()

    pred = result.predict(df_test)
    rss = ((df_test.MEDV - pred) ** 2).sum()
    tss = ((df_test.MEDV - df_test.MEDV.mean())** 2).sum()
    rsquared = 1 - rss / tss

    scores[i] = rsquared
    print(f"학습용 R2 = {result.rsquared:.3f}, 검증용 R2 = {r
squared:.3f}")
```

```
from sklearn.metrics import r2_score,mean_squared_error,mea
n_absolute_error

scores1 = np.zeros(5)
scores2 = np.zeros(5)
scores3 = np.zeros(5)
cv = KFold(5, shuffle=True, random_state=0)
for i, (idx_train, idx_test) in enumerate(cv.split(df)):
    df_train = df.iloc[idx_train]
    df_test = df.iloc[idx_test]

    model = sm.OLS.from_formula("MEDV ~ " + "+".join(boston.
feature_names), data=df_train)
    result = model.fit()

    pred = result.predict(df_test)
    #결정계수를 구하는 함수
    rsquared = r2_score(df_test.MEDV, pred)
    scores1[i] = rsquared
    #평균제곱오차(Mean Squared Error) - 오차의 제곱의 합계의
평균값
    mse = mean_squared_error(df_test.MEDV, pred)
    scores2[i] = mse
    #평균절대오차(Mean Absolute Error) - 오차의 합계의 평균값
    mae = mean_absolute_error(df_test.MEDV, pred)
    scores3[i] = mae



print(scores1)
print(scores2)
print(scores3)
```

```
from sklearn.base import BaseEstimator, RegressorMixin
import statsmodels.formula.api as smf
import statsmodels.api as sm

class StatsmodelsOLS(BaseEstimator, RegressorMixin):
    def __init__(self, formula):
        self.formula = formula
        self.model = None
        self.data = None
        self.result = None

    def fit(self, dfX, dfy):
        self.data = pd.concat([dfX, dfy], axis=1)
        self.model = smf.ols(self.formula, data=self.data)
        self.result = self.model.fit()

    def predict(self, new_data):
        return self.result.predict(new_data)
```

```
from sklearn.model_selection import cross_val_score

model = StatsmodelsOLS("MEDV ~ " + "+".join(boston.feature_
names))
cv = KFold(5, shuffle=True, random_state=0)
cross_val_score(model, dfX, dfy, scoring="r2", cv=cv)

#평균제곱오차로 평가하는 경우
result=cross_val_score(model, dfX, dfy, scoring='neg_mean_s
quared_error', cv=cv)
#음수로 나온 결과값을 양수로 변환
rmse_score = np.sqrt(-result)
rmse_score
```

6) 주택 가격 예측2

```
# https://www.kaggle.com/anthonypino/price-analysis-and-lin
ear-regression
import pandas as pd
import numpy as np
%matplotlib inline
import seaborn as sns
import matplotlib.pyplot as plt
```

```
df =  pd.read_csv("c:/data/house_regress/data.csv")
```

```
df.head()
```

```
df.shape
```

```
(19740, 21)
```

```
df.columns
```

```
Index(['Suburb', 'Address', 'Rooms', 'Type', 'Price',
'Method', 'SellerG', 'Date', 'Distance', 'Postcode',
'Bedroom2', 'Bathroom', 'Car','Landsize', 'BuildingArea',
'YearBuilt', 'CouncilArea', 'Lattitude','Longtitude',
'Regionname', 'Propertycount'],
      dtype='object')
```

```
X= df.loc[ : , ['Rooms', 'Distance', 'Bedroom2',
'Bathroom', 'Car', 'Landsize', 'BuildingArea',
'Propertycount']]
y = df["Price"]
df2=pd.concat([X,y],axis=1)

from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy="mean")
df3 = pd.DataFrame(imputer.fit_transform(df2),
columns=df2.columns)
df3

cols=df3.columns[:-1]
X=df3[cols]
y=df3['Price']
```

```
#다중 회귀분석
import statsmodels.api as sm

model = sm.OLS(y, X)
result = model.fit()

print("회귀계수\n",result.params)
print("R-squared\n",result.rsquared)
print("P values\n",result.pvalues)

#요약 결과 출력
print(result.summary())
```

```
#가장 유의하지 않은 Landsize 제외
X= df3.loc[ : , ['Rooms', 'Car','Distance', 'Bedroom2',
'Bathroom', 'BuildingArea', 'Propertycount']]
y = df3["Price"]
```

```
model = sm.OLS(y, X)
result = model.fit()
print(result.summary())
```

```
# Bedroom2 제외
X= df3.loc[ : , ['Rooms', 'Car','Distance', 'Bathroom',
'BuildingArea', 'Propertycount']]
y = df3["Price"]
```

```
model = sm.OLS(y, X)
result = model.fit()
print(result.summary())
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
#데이터셋을 나눔(학습용:검증용 = 8:2)
#random_state : 난수 발생을 위한 seed의 인자값
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=10)
model = LinearRegression().fit(X_train, y_train)
```

```
print("학습용:",model.score(X_train, y_train))
print("검증용:",model.score(X_test, y_test))
```

7) 범주형 독립변수

```
#1920~1939년 노팅엄 지역 월 평균 기온 데이터
import pandas as pd
df = pd.read_csv('c:/data/degree/data.csv')
df
```

```
%matplotlib inline
#boxplot(y,x)
df.boxplot("value", "month")
```

```
#카테고리형 변수인 월이 연속숫자형으로 인식되지 않도록 C() 함수 사용
```

```
import statsmodels.api as sm
# +0을 붙이면 1월~12월이 평균기온으로 처리됨
# coef는 월의 평균기온
model = sm.OLS.from_formula("value ~ C(month)+0", df)
result = model.fit()
result.summary()
```

```
# +0이 없으면 첫번째 변수인 1월은 제외하고 1월을 기준으로 각 월의 평균 기온이
# 1월보다 얼마나 더 높은지 낮은지를 나타내는 값이 회귀모형의 계수가 된다.
model = sm.OLS.from_formula("value ~ C(month)", df)
result = model.fit()
result.summary()
```

8) 부분회귀 플롯

```
# 새로운 독립변수를 추가하여 다시 회귀분석을 하면 기존 가중치 벡터의 값은 변경된다.
# 부분회귀플롯:독립변수가 여러개일 때 특정한 하나의 독립변수의 영향력을 시각화하는 방법
```

```
from sklearn.datasets import load_boston
import pandas as pd

boston = load_boston()

dfX = pd.DataFrame(boston.data, columns=boston.feature_names)
dfy = pd.DataFrame(boston.target, columns=["MEDV"])
df = pd.concat([dfX, dfy], axis=1)
df
```

```
import statsmodels.api as sm

model = sm.OLS(dfy, dfX)
result = model.fit()
result.summary()
```

```
%matplotlib inline
import seaborn as sns
#age와 medv는 음의 상관관계가 있는 것처럼 보인다.
#이 부분플롯은 순수하게 age와 medv의 상관관계를 표현한 것이 아님.
sns.regplot(x="AGE", y="MEDV", data=df)
```

```
# A.difference(B) A-B 차집합
others = list(set(df.columns).difference(set(["MEDV",
"AGE"])))
others
```

```
from statsmodels.graphics.regressionplots import
plot_partregress
import matplotlib.pyplot as plt
# others : AGE를 제외한 나머지 독립변수들
# obs_labels : 데이터 라벨링 여부
plot_partregress("MEDV", "AGE", others, data=df,
obs_labels=False)
plt.show()
#부분회귀 플롯을 볼 때 age와 medv는 상관관계가 없다.
#가로축의 값은 독립변수 자체의 값이 아닌
#  어떤 독립변수에서 다른 독립변수의 영향을 제거한 값
```

```
#전체 데이터에 대해 한번에 부분회귀 플롯을 그리는 함수
from statsmodels.graphics.regressionplots import plot_partr
egress_grid
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(8, 20))
# result: 학습 완료된 모형
plot_partregress_grid(result, fig=fig)
plt.plot()
```

9) 보험료 예측

```
#회귀분석(보험료 예측)
# https://www.kaggle.com/mirichoi0218/insurance/downloads/insurance.csv/1
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
import seaborn as sns
```

```
data = pd.read_csv('c:/data/insurance/insurance.csv')
data.head()
```

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520

```
data.corr()['charges'].sort_values()
```

```
children    0.067998
bmi         0.198341
age         0.299008
charges     1.000000
Name: charges, dtype: float64
```

```
#전처리된 파일(전처리 실습)
data2 = pd.read_csv('c:/data/insurance/insurance2.csv')
data2.head()
```

	age	sex	bmi	children	smoker	southwest	southeast	northeast	northwest	charges
0	19	0	27.900	0	1	1	0	0	0	16884.92400
1	18	1	33.770	1	0	0	1	0	0	1725.55230
2	28	1	33.000	3	0	0	1	0	0	4449.46200
3	33	1	22.705	0	0	0	0	0	1	21984.47061
4	32	1	28.880	0	0	0	0	0	1	3866.85520

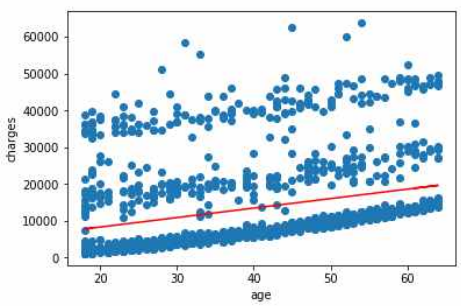
```
data2.corr()['charges'].sort_values()
```

```
southwest   -0.043210
northwest   -0.039905
northeast    0.006349
sex          0.057292
children     0.067998
southeast    0.073982
bmi          0.198341
age          0.299008
smoker       0.787251
charges      1.000000
Name: charges, dtype: float64
```

```
#단순회귀분석
from scipy import stats
#나이와 보험료와의 관계
X=data2["age"].values
#종속변수
y=data2["charges"].values
result=stats.linregress(X, y)
result
```

```
  <class 'numpy.ndarray'>
[19 18 28 ... 18 21 61]
[16884.924    1725.5523   4449.462   ...   1629.8335   2007.945
29141.3603]
LinregressResult(slope=257.7226186668955,
intercept=3165.885006063025, rvalue=0.2990081933306477,
pvalue=4.886693331718491e-29, stderr=22.5023892867703)
```

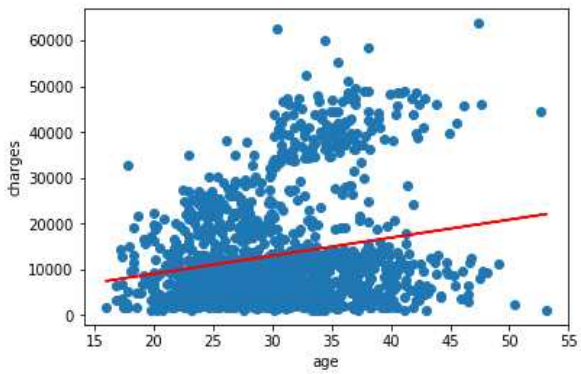
```
import matplotlib.pyplot as plt
slope, intercept, r_value, p_value, stderr = stats.linregre
ss(X, y)
#산점도 그리기
plt.scatter(X,y)
#회귀선 그리기
plt.plot(X, slope*X +intercept, c="red")
plt.xlabel("age")
plt.ylabel("charges")
```



```
from scipy import stats
#bmi와 보험료와의 관계
X=data2["bmi"].values
#종속변수
y=data2["charges"].values
result=stats.linregress(X, y)
result
```

```
LinregressResult(slope=393.87303079739524,
intercept=1192.9372089611497, rvalue=0.1983409688336289,
pvalue=2.459085535116683e-13, stderr=53.25073835210321)
```

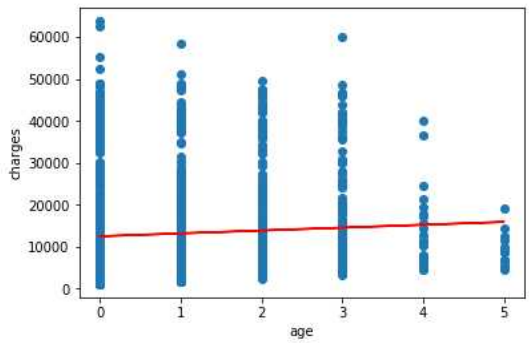
```
import matplotlib.pyplot as plt

slope, intercept, r_value, p_value, stderr = stats.linregre
ss(X, y)
#산점도 그리기
plt.scatter(X,y)
#회귀선 그리기
plt.plot(X, slope*X +intercept, c="red")
plt.xlabel("age")
plt.ylabel("charges")
```



```
from scipy import stats
#자녀수와 보험료와의 관계
X=data2["children"].values
#종속변수
y=data2["charges"].values
result=stats.linregress(X, y)
result
```

```
LinregressResult(slope=683.0893824813649,
intercept=12522.495549644096, rvalue=0.06799822684790487,
pvalue=0.012852128520136412, stderr=274.2018326126803)
```

```
import matplotlib.pyplot as plt

slope, intercept, r_value, p_value, stderr = stats.linregre
ss(X, y)
#산점도 그리기
plt.scatter(X,y)
#회귀선 그리기
plt.plot(X, slope*X +intercept, c="red")
plt.xlabel("children")
plt.ylabel("charges")
```



```
# 다중 회귀분석

#명목형 변수는 제외하고 연속형 변수만 독립변수로 선택
#독립변수 : 연속형 변수만 사용하고 범주형 변수는 사용하지 않음
df2=data2.iloc[:,[0,2,3]]
print(df2.head())
#종속변수
y2=data2.iloc[:,[9]]
print(y2.head())
```

```
   age     bmi  children
0   19  27.900         0
1   18  33.770         1
2   28  33.000         3
3   33  22.705         0
4   32  28.880         0
       charges
0  16884.92400
1   1725.55230
2   4449.46200
3  21984.47061
4   3866.85520
```

```
import statsmodels.api as sm

model = sm.OLS(y2, df2)
result = model.fit()

print("회귀계수\n",result.params)
print("R-squared\n",result.rsquared) #회귀모델의 설명력 59.5%
의 설명력
print("P values\n",result.pvalues)

#요약 결과 출력
print(result.summary())
```

```
#유의하지 않은 children 변수 제거
df2=data2.iloc[:,[0,2]]
print(df2.head())
#종속변수
y2=data2.iloc[:,[9]]
print(y2.head())

model = sm.OLS(y2, df2)
result = model.fit()

print("회귀계수\n",result.params)
print("R-squared\n",result.rsquared)
print("P values\n",result.pvalues)

#요약 결과 출력
print(result.summary())
```

10) 레버리지와 이상치

```
#레버리지(leverage, 지렛대/영향력) : 실제값이 예측값에 미치는 영향을 나타낸 값
# 0~1 사이의 값
```

```
from sklearn.datasets import make_regression

# 100개의 데이터 생성
X0, y, coef = make_regression(n_samples=100, n_features=1, noise=20, coef=True, random_state=1)
```

```
import numpy as np
import statsmodels.api as sm
# 가상의 outlier 추가
data_100 = (4, 300)
data_101 = (3, 150)
X0 = np.vstack([X0, np.array([data_100[:1], data_101[:1]])])
# 회귀분석에서 수식을 간단하게 만들기 위해
#   독립변수의 첫번째 필드에 상수항 1을 추가(augmentation)
X = sm.add_constant(X0)
y = np.hstack([y, [data_100[1], data_101[1]]])
```

```
%matplotlib inline
import matplotlib.pyplot as plt

plt.scatter(X0, y)
plt.xlabel("x")
plt.ylabel("y")
plt.title("Regression")
```

```
import pandas as pd
model = sm.OLS(pd.DataFrame(y), pd.DataFrame(X))
result = model.fit()
print(result.summary())
```

```
#영향도 값
influence = result.get_influence()
# 레버리지 벡터의 값
hat = influence.hat_matrix_diag
#print(hat)
#막대그래프와 비슷하지만 폭이 없는 그래프
plt.stem(hat)
#0.02에 가로 점선 추가
plt.axhline(0.02, c="g", ls="--")
plt.title("leverage")
#마지막에 추가한 이상치를 제외하면 대부분의 값들은 0.02 근처의 낮은 값들
```

```
hat.sum() #합계는 2에 수렴함
```

```
#레버리지 값이 0.05보다 큰 샘플 강조
ax = plt.subplot()
plt.scatter(X0, y)
sm.graphics.abline_plot(model_results=result, ax=ax)

idx = hat > 0.05
plt.scatter(X0[idx], y[idx], s=300, c="r", alpha=0.5)
plt.show()
```

```
#레버리지가 높은 데이터가 회귀분석에 미치는 영향
model2 = sm.OLS(y[:-1], X[:-1]) #마지막 샘플 제외
result2 = model2.fit()
ax = plt.subplot()
plt.scatter(X0, y)
#레버리지가 큰 데이터를 포함한 모형
sm.graphics.abline_plot(model_results=result,
                    c="r", linestyle="--", ax=ax, label='A')
#레버리지가 큰 데이터를 포함하지 않은 모형
sm.graphics.abline_plot(model_results=result2,
                       c="g", alpha=0.7, ax=ax, label='B')
# ms(marker size), mew(marker edge width)
#이상치
plt.plot(X0[-1], y[-1], marker='x', c="m", ms=20, mew=5)
plt.legend()
plt.show()
```

```
# 각 샘플의 잔차
plt.stem(result.resid)
plt.show()
```

```
# 각 샘플의 표준화 잔차(잔차를 스케일링한 값, 일반적으로 2~4보다
크면 아웃라이어로 본다.)
plt.figure(figsize=(10, 2))
plt.stem(result.resid_pearson)
plt.axhline(3, c="g", ls="--")
plt.axhline(-3, c="g", ls="--")
plt.show()
```

```
#레버리지와 잔차의 크기가 모두 큰 데이터들을 보기 위한 그래프
#x축: 표준화 잔차의 제곱, y축: 레버리지값
#숫자가 표시된 데이터들 확인
sm.graphics.plot_leverage_resid2(result)
plt.show()
```

```
#레버리지와 잔차의 크기가 모두 큰 데이터들을 시각적으로 표현
sm.graphics.influence_plot(result, plot_alpha=0.3)
plt.show()
#샘플의 인덱스 확인(인덱스 100,101번이 outlier)
```

```
from sklearn.datasets import load_boston
boston = load_boston()
```

```
dfX0 = pd.DataFrame(boston.data, columns=boston.feature_nam
es)
# 회귀분석에서 수식을 간단하게 만들기 위해
#   독립변수의 첫번째 필드에 상수항 1을 추가(augmentation)
dfX = sm.add_constant(dfX0)
dfy = pd.DataFrame(boston.target, columns=["MEDV"])
```

```
model_boston = sm.OLS(dfy, dfX)
result_boston = model_boston.fit()
print(result_boston.summary())
```

```
dfy.boxplot()
```

```
#주택가격이 40이상일 때 40으로 낮추는 방법
dfy.loc[dfy.MEDV >= 40, 'MEDV']=40
model_boston2=sm.OLS(dfy, dfX)
result_boston2=model_boston2.fit()
result_boston2.summary()
#78.2%
```

```
# 이상치라고 생각되는 MEDV >= 40 이상인 값들
idx = np.where(dfy >= 40)[0]
idx
```

```
#아웃라이어를 제외한 후 회귀 분석을 한 결과
idx2 = list(set(range(len(dfX))).difference(idx))
#행인덱스를 새롭게 부여하고 인덱스 필드는 제거
dfX = dfX.iloc[idx2, :].reset_index(drop=True)
dfy = dfy.iloc[idx2, :].reset_index(drop=True)
model_boston3 = sm.OLS(dfy, dfX)
result_boston3 = model_boston2.fit()
print(result_boston3.summary())
# R-squared: 74.1% => 77.1%
```

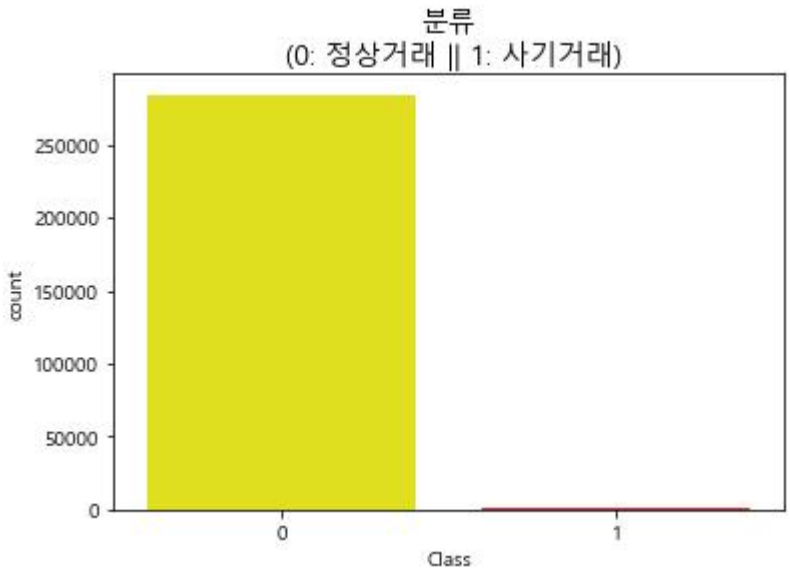
11) 신용카드 거래 금액 예측

```
# 원본 데이터셋 출처

# https://www.kaggle.com/janiobachmann/credit-fraud-dealing
-with-imbalanced-datasets/data

# 2013년 9월 유럽 카드 소지자가 신용카드로 거래한 내용, 284807
건의 거래 가운데 492건의 사기거래
# 변수 v1~v28, Amount 거래 금액, Class 0/1 정상거래/사기거래
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# 신용카드 거래 데이터 csv 파일을 로딩(원본)
df = pd.read_csv("c:/data/creditcard/creditcard.csv")

df.head()
```

```
#한글 처리를 위해 폰트 설정
from matplotlib import font_manager, rc
font_name = font_manager.FontProperties(fname="c:/Windows/F
onts/malgun.ttf").get_name()
rc('font', family=font_name)

colors = ["yellow", "red"]

sns.countplot('Class', data=df, palette=colors)
plt.title('분류\n (0: 정상거래 || 1: 사기거래)', fontsize=14)
#불균형 데이터셋
```



```
print("사기거래")
print(df.Amount[df.Class == 1].describe())
print()
print("정상거래")
print(df.Amount[df.Class == 0].describe())
# 사기거래 492건, 정상거래 284315건
```

```
사기거래
count     492.000000
mean      122.211321
std       256.683288
min         0.000000
25%         1.000000
50%         9.250000
75%       105.890000
max      2125.870000
Name: Amount, dtype: float64

정상거래
count    284315.000000
mean         88.291022
std         250.105092
min           0.000000
25%           5.650000
50%          22.000000
75%          77.050000
max       25691.160000
Name: Amount, dtype: float64
```

```
train_cols = df.columns[1:29]
print(train_cols)
X = df[train_cols] # 독립변수
y = df["Amount"]
print(X.head())
print(y.head())
```

```
from sklearn.model_selection import train_test_split
#데이터셋을 나눔(학습용:검증용 = 8:2)
#random_state : 난수 발생을 위한 seed의 인자값
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=10)
```

```
#다중 회귀분석
import statsmodels.api as sm

model = sm.OLS(y_train, X_train)
result = model.fit()

print("회귀계수\n",result.params)
print("R-squared\n",result.rsquared)
print("P values\n",result.pvalues)

#요약 결과 출력
print(result.summary())
# 모델의 설명력 91.3%
# p-value를 확인해 볼 때 모든 변수가 유의한 것으로 나타남
```

```
from sklearn.metrics import r2_score
pred=result.predict(X_test)
r2_score(y_test, pred)
```

```
from sklearn.linear_model import LinearRegression
model = LinearRegression().fit(X_train, y_train)
```

```
print("학습용:",model.score(X_train, y_train))
print("검증용:",model.score(X_test, y_test))
```

학습용: 0.9165786704366631

검증용: 0.920111967752019

12) HDI(인간개발지수) 데이터 실습

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
```

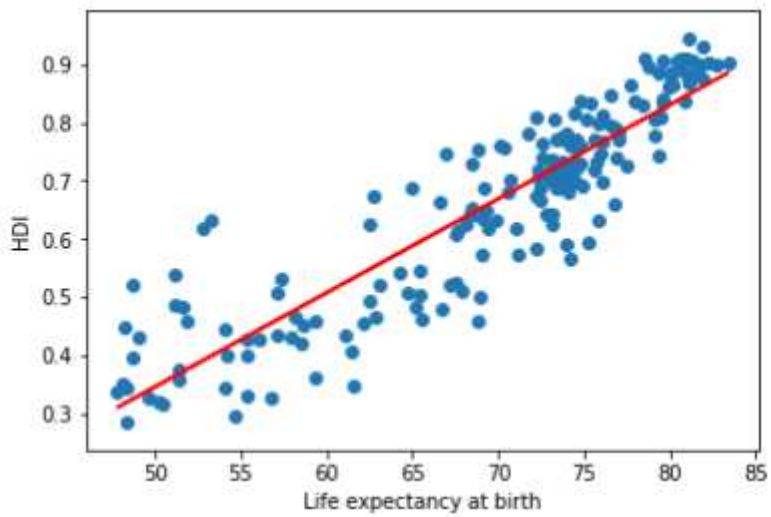
```
# 전처리 과정을 거친 데이터 로딩
df = pd.read_csv("c:/data/hdi/hdi-data2.csv")

df.head()
```

```
train_cols = df.columns[0:5]
print(train_cols)
X = df[train_cols] # 독립변수
y = df["HDI"]
print(df[train_cols].head())
print(y.head())
```

```
#단순회귀분석
#기울기(slope), 절편(intercept), 상관계수(rvalue), p-value(예측 불확실성의 정도를 나타내는 값)
#에러의 표준편차(stderr)
#p-value는 0.05 미만일 때 통계적으로 유의함
x2 = df["Life expectancy at birth"].values
y2 = df["HDI"].values
result=stats.linregress(x2, y2)
result
```

```
LinregressResult(slope=0.016124893707708518,
intercept=-0.4599942026967768, rvalue=0.9015129749487512,
pvalue=3.284775456269812e-69, stderr=0.000569086352615306)
```

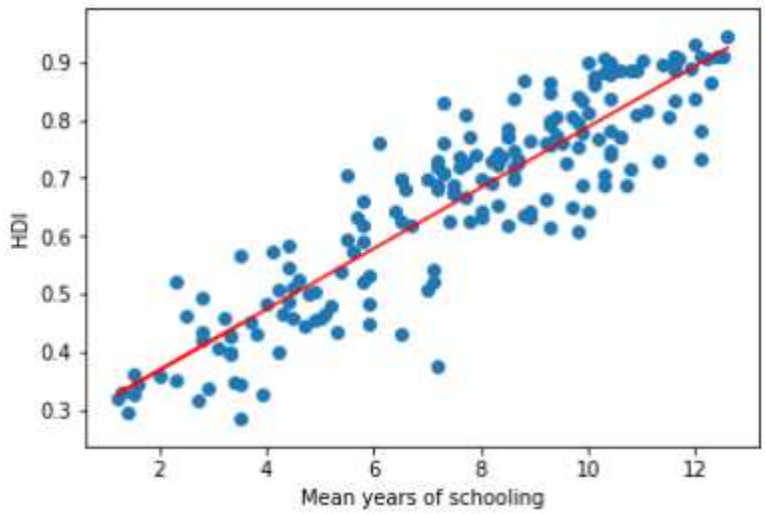
```
slope, intercept, r_value, p_value, stderr = stats.linregre
ss(x2, y2)
#산점도 그리기
plt.scatter(x2,y2)
#회귀선 그리기
plt.plot(x2, slope*x2 +intercept, c="red")
plt.xlabel("Life expectancy at birth")
plt.ylabel("HDI")
```




```
x2 = df["Mean years of schooling"].values
y2 = df["HDI"].values
result=stats.linregress(x2, y2)
result
```

```
LinregressResult(slope=0.05232467201381997,
intercept=0.26456717569801413, rvalue=0.9005296673824348,
pvalue=7.857605621580009e-69, stderr=0.0018574055433612294)
```

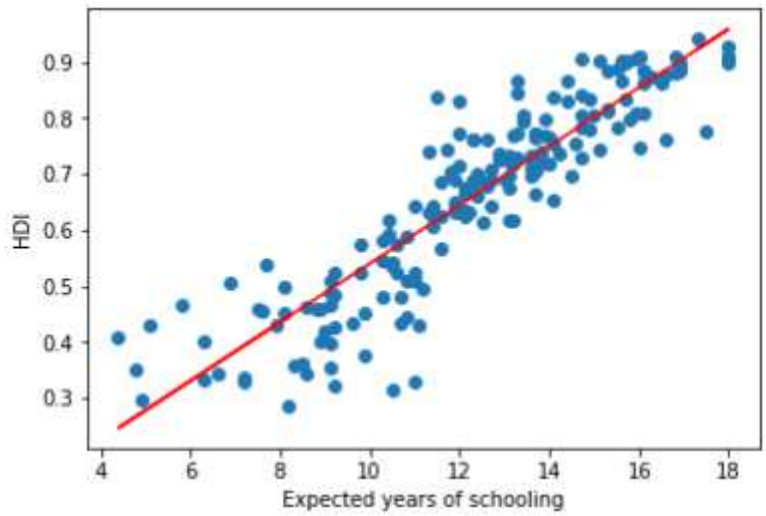
```
slope, intercept, r_value, p_value, stderr = stats.linregre
ss(x2, y2)
#산점도 그리기
plt.scatter(x2,y2)
#회귀선 그리기
plt.plot(x2, slope*x2 +intercept, c="red")
plt.xlabel("Mean years of schooling")
plt.ylabel("HDI")
```




```
x2 = df["Expected years of schooling"].values
y2 = df["HDI"].values
result=stats.linregress(x2, y2)
result
```

```
LinregressResult(slope=0.05241654962397508,
intercept=0.015688076086450953, rvalue=0.9013827197759505,
pvalue=3.6890187212890467e-69, stderr=0.0018513333268186474)
```

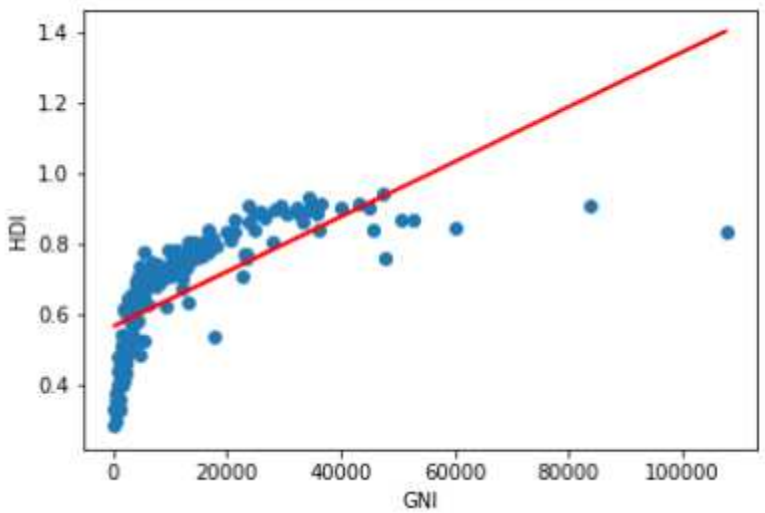
```
slope, intercept, r_value, p_value, stderr = stats.linregre
ss(x2, y2)
#산점도 그리기
plt.scatter(x2,y2)
#회귀선 그리기
plt.plot(x2, slope*x2 +intercept, c="red")
plt.xlabel("Expected years of schooling")
plt.ylabel("HDI")
```




```
x2 = df["GNI"].values
y2 = df["HDI"].values
result=stats.linregress(x2, y2)
result
```

```
LinregressResult(slope=7.789292151991572e-06,
intercept=0.5632858689341738, rvalue=0.7003465074525178,
pvalue=6.793224486797103e-29, stderr=5.836836801269087e-07)
```

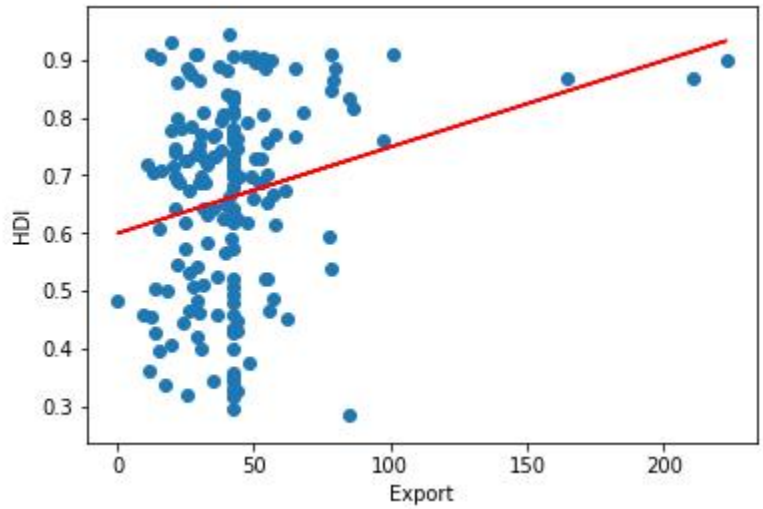
```
slope, intercept, r_value, p_value, stderr = stats.linregre
ss(x2, y2)
#산점도 그리기
plt.scatter(x2,y2)
#회귀선 그리기
plt.plot(x2, slope*x2 +intercept, c="red")
plt.xlabel("GNI")
plt.ylabel("HDI")
```




```
x2 = df["Export"].values
y2 = df["HDI"].values
result=stats.linregress(x2, y2)
result
```

```
LinregressResult(slope=0.0014930000842694835,
intercept=0.5994367855376995, rvalue=0.22835956515234107,
pvalue=0.0016695578488607908, stderr=0.0004679753471524697)
```

```
slope, intercept, r_value, p_value, stderr = stats.linregre
ss(x2, y2)
#산점도 그리기
plt.scatter(x2,y2)
#회귀선 그리기
plt.plot(x2, slope*x2 +intercept, c="red")
plt.xlabel("Export")
plt.ylabel("HDI")
```



```
#다중 회귀분석
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=10)

import statsmodels.api as sm
model = sm.OLS(y_train, X_train)
result = model.fit()

print("회귀계수\n",result.params)
print("R-squared\n",result.rsquared)
print("P values\n",result.pvalues)

#요약 결과 출력
print(result.summary())
# 모델의 설명력
# p-value를 확인해 볼 때 모든 변수가 유의한 것으로 나타남

from sklearn.metrics import r2_score
pred=result.predict(X_test)
rsquared=r2_score(y_test, pred)
rsquared
```

```
from sklearn.linear_model import LinearRegression
model = LinearRegression().fit(X_train, y_train)
```

```
print("학습용:",model.score(X_train, y_train))
print("검증용:",model.score(X_test, y_test))
```

학습용: 0.9795835622122138

검증용: 0.9431322148074397

13) 난방효율성 데이터 분석

```
#에너지 효율성 데이터셋
#건축 구조의 기본 요소인 건물 표면적, 벽과 지붕 면적, 높이, 사각지대, 건물 외형의 간결성,
#건물의 난방과 냉방 효율의 관계 등을 조사한 데이터
#18가지의 건축 특성을 지닌 12가지의 건축 속성, 총 768채의 주택 조사

# X1 : 상대적 크기
# X2 : 건축 표면적
# X3 : 벽체 면적
# X4 : 지붕 면적
# X5 : 전체 높이
# X6 : 건물의 방위
# X7 : 유리창 면적
# X8 : 유리창 면적의 분산
# Y1 : 난방 하중
# Y2 : 냉방 하중
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
```

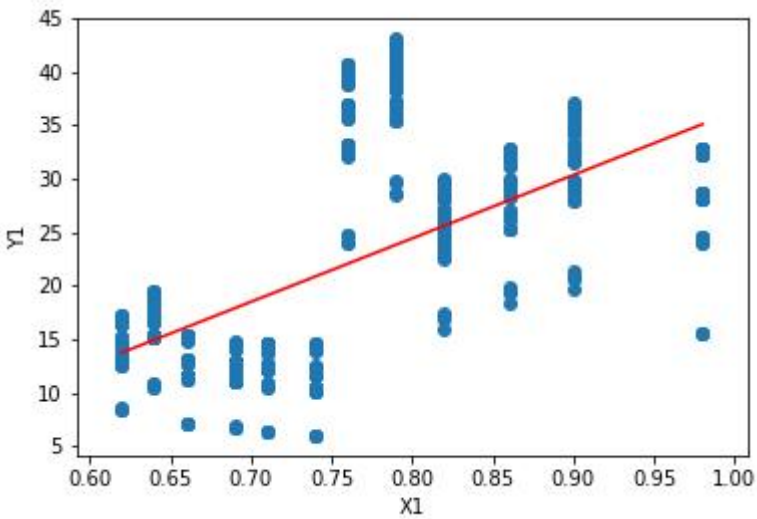
```
df = pd.read_csv("c:/data/energy/ENB2012_data.csv")

df.head()
```

```
train_cols = df.columns[0:8]
print(train_cols)
X = df[train_cols] # 독립변수
y = df["Y1"] # 난방 하중
print(df[train_cols].head())
print(y.head())
```

```
#단순회귀분석
#기울기(slope), 절편(intercept), 상관계수(rvalue), p-value(예측 불확실성의 정도를 나타내는 값)
#에러의 표준편차(stderr)
#p-value는 0.05 미만일 때 통계적으로 유의함
x2 = df["X1"].values
y2 = df["Y1"].values
result=stats.linregress(x2, y2)
result
```

```
LinregressResult(slope=59.35902736778322,
intercept=-23.05298955938101, rvalue=0.6222721790646625,
pvalue=1.5912736997125491e-83, stderr=2.69800874779432)
```

```
slope, intercept, r_value, p_value, stderr = stats.linregre
ss(x2, y2)
#산점도 그리기
plt.scatter(x2,y2)
#회귀선 그리기
plt.plot(x2, slope*x2 +intercept, c="red")
plt.xlabel("X1")
plt.ylabel("Y1")
```



```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=10)
```

```
#다중 회귀분석
import statsmodels.api as sm

model = sm.OLS(y_train, X_train)
result = model.fit()

print("회귀계수\n",result.params)
print("R-squared\n",result.rsquared)
print("P values\n",result.pvalues)

#요약 결과 출력
print(result.summary())
# 모델의 설명력 98.5%
# p-value : X6(건물의 방위)은 유의하지 않음
# 해석
# X1(상대적 크기) : 건물이 크면 난방 비용이 감소한다.
# X7(유리창 면적) : 유리창 면적이 크면 난방 비용이 증가한다.
# X5(전체 높이) : 건물 높이가 높으면 난방 비용이 증가한다.
```

```
from sklearn.linear_model import LinearRegression
model = LinearRegression().fit(X_train, y_train)
```

```
print("학습용:",model.score(X_train, y_train))
print("검증용:",model.score(X_test, y_test))
```

학습용: 0.9123488471973692

검증용: 0.9313289941747572

```
# X6을 제외한 모델

train_cols = df.columns[[0,1,2,3,4,6,7]]
print(train_cols)
X = df[train_cols] # 독립변수
y = df["Y1"] # 난방 하중
print(df[train_cols].head())
print(y.head())
```

```
#다중 회귀분석
import statsmodels.api as sm

model = sm.OLS(y, X)
result = model.fit()

print("회귀계수\n",result.params)
print("R-squared\n",result.rsquared)
print("P values\n",result.pvalues)

#요약 결과 출력
print(result.summary())
# 모델의 설명력 98.5%(개선되지는 않음)
# p-value : 모든 변수가 유의함
```

14) 냉방효율성 데이터분석

```
#에너지 효율성 데이터셋
#건축 구조의 기본 요소인 건물 표면적, 벽과 지붕 면적, 높이, 사각지대, 건물 외형의 간결성,
#건물의 난방과 냉방 효율의 관계 등을 조사한 데이터
#18가지의 건축 특성을 지닌 12가지의 건축 속성, 총 768채의 주택 조사


# X1 : 상대적 크기
# X2 : 건축 표면적
# X3 : 벽체 면적
# X4 : 지붕 면적
# X5 : 전체 높이
# X6 : 건물의 방위
# X7 : 유리창 면적
# X8 : 유리창 면적의 분산
# Y1 : 난방 하중
# Y2 : 냉방 하중
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
```

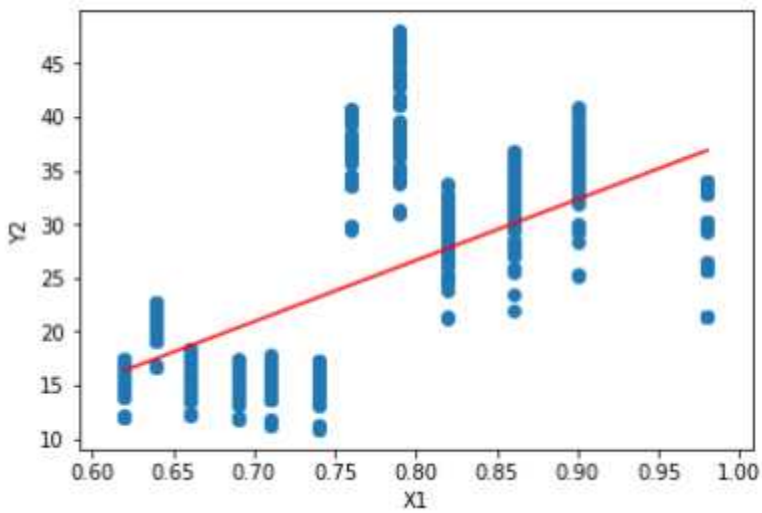
```
df = pd.read_csv("c:/data/energy/ENB2012_data.csv")


df.head()
```

```
train_cols = df.columns[0:8]
print(train_cols)
X = df[train_cols] # 독립변수
y = df["Y2"] # 냉방 하중
print(df[train_cols].head())
print(y.head())
```

```
#단순회귀분석
#기울기(slope), 절편(intercept), 상관계수(rvalue), p-value(예측 불확실성의 정도를 나타내는 값)
#에러의 표준편차(stderr)
#p-value는 0.05 미만일 때 통계적으로 유의함
x2 = df["X1"].values
y2 = df["Y2"].values
result=stats.linregress(x2, y2)
result
```

```
LinregressResult(slope=57.05053290659375,
intercept=-19.008355146122057, rvalue=0.6343390663353593,
pvalue=1.0608759912586174e-87, stderr=2.5120843906893957)
```

```
slope, intercept, r_value, p_value, stderr = stats.linregre
ss(x2, y2)
#산점도 그리기
plt.scatter(x2,y2)
#회귀선 그리기
plt.plot(x2, slope*x2 +intercept, c="red")
plt.xlabel("X1")
plt.ylabel("Y2")
```



```
#다중 회귀분석
import statsmodels.api as sm

model = sm.OLS(y, X)
result = model.fit()

print("회귀계수\n",result.params)
print("R-squared\n",result.rsquared)
print("P values\n",result.pvalues)

#요약 결과 출력
print(result.summary())
# 모델의 설명력 98.5%
# p-value : X2(건축 표면적),X6(건물의 방위),X8(유리창 면적의 분산)은 유의하지 않음
# 해석
# X1(상대적 크기) : 건물이 크면 냉방비용이 감소된다.
# X7(유리창 면적) : 유리창 면적이 크면 냉방비용이 증가한다.
# X5(전체 높이) : 건물 높이가 높으면 냉방비용이 증가한다.
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
#데이터셋을 나눔(학습용:검증용 = 8:2)
#random_state : 난수 발생을 위한 seed의 인자값
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=10)
model = LinearRegression().fit(X_train, y_train)
```

```
print("학습용:",model.score(X_train, y_train))
print("검증용:",model.score(X_test, y_test))
```

학습용: 0.8824724641607591

검증용: 0.907978623553135

15) 놀이기구 만족도 데이터 분석

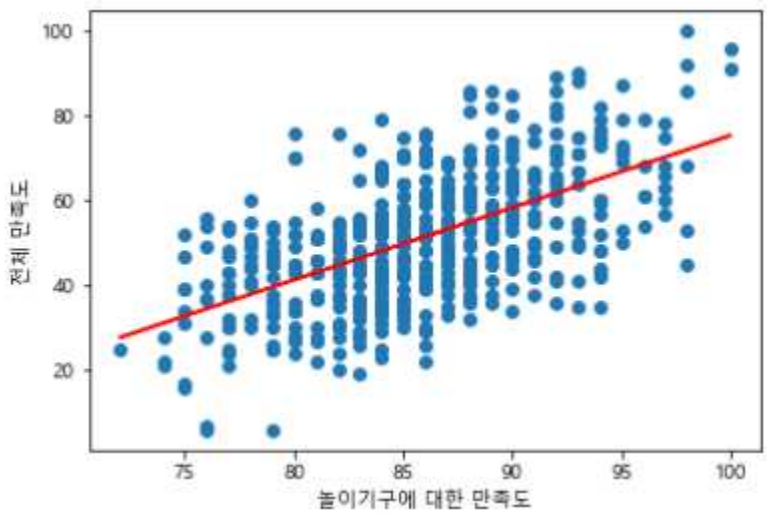
```
#귀무가설 : 놀이기구에 대한 만족도는 전체만족도에 영향을 미치지 않는다.
#대립가설 : 놀이기구에 대한 만족도는 전체만족도에 영향을 미친다.

# csv 파일을 로딩
import pandas as pd

df = pd.read_csv("c:/data/rides/rides.csv")

print(df.head())
```

```
#단순회귀분석 : 놀이기구 만족도와 전체 만족도와의 관계
x = df["rides"].values
print(x[:10])
y = df["overall"].values
print(y[:10])
```

```
from scipy import stats

result=stats.linregress(x, y)
result

#slope : 기울기, intercept : 절편, rvalue : 상관계수
#p-value : 예측 불확실성의 정도를 나타내는 값
#stderr : 에러의 표준편차
#p-value는 0.05 미만일 때 유의함

#모형의 적합도 :  rvalue 0이 아니므로 추정하는 회귀함수가 유의미하다.
# rvalue : 0.58  이 모델은 58%의 설명력이 있음

#p_value : 0.05보다 작으므로 통계적으로 유의함

# 따라서 귀무가설을 기각하고 대립가설을 채택한다.
#결론 : 놀이기구에 대한 만족도는 전체만족도에 영향을 미친다.
```

```
LinregressResult(slope=1.7032854834102056,
intercept=-94.96224560883252, rvalue=0.5859862820034282,
pvalue=1.9715137881102242e-47, stderr=0.10554615174616558)
```

```
%matplotlib inline
import matplotlib.pyplot as plt
from matplotlib import font_manager, rc

#한글 처리를 위해 폰트 설정
font_name = font_manager.FontProperties(\
fname="c:/Windows/Fonts/malgun.ttf").get_name()
rc('font', family=font_name)

slope, intercept, r_value, p_value, stderr = stats.linregre
ss(x, y)
#산점도 그리기
plt.scatter(x,y)
#회귀선 그리기
plt.plot(x, slope*x +intercept, c="red")
plt.xlabel("놀이기구에 대한 만족도")
plt.ylabel("전체 만족도")
```



```
#놀이기구에 대한 만족도가 90일 때 전체 만족도 예측
90 * slope + intercept
```

```
58.33344789808598
```

```
#범주형 변수 처리(주말여부)
df2=pd.get_dummies(df['weekend'],prefix='weekend')
df=pd.concat([df,df2],axis=1)
del df['weekend']
df.head()
```

```
#다중회귀분석
#독립변수
X=df[['num.child', 'distance', 'rides', 'games', 'wait',
'clean', 'weekend_no', 'weekend_yes']]
#종속변수
y=df['overall']
```

```
import statsmodels.api as sm

model = sm.OLS(y, X)
result = model.fit()

result.summary()
# 모델의 설명력 68.3%
# p-value : games(게임종류) 유의하지 않음
# 해석
# 만족도에 가장 큰 영향을 미치는 변수는 자녀수이다.
```

```
#자녀수, 거리, 놀이기구점수, 게임점수, 대기시간점수, 청결도, 평일여부, 주말여부
result.predict([2, 20, 80, 70, 80, 90, 0, 1])
```

```
result.predict([3, 10, 80, 70, 80, 90, 1, 0])
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
#데이터셋을 나눔(학습용:검증용 = 8:2)
#random_state : 난수 발생을 위한 seed의 인자값
X_train, X_test, y_train, y_test = train_test_split(\
X, y, test_size=0.2, random_state=10)
model = LinearRegression().fit(X_train, y_train)
```

```
print("학습용:",model.score(X_train, y_train))
print("검증용:",model.score(X_test, y_test))
```

```
#자녀수, 거리, 놀이기구점수, 게임점수, 대기시간점수, 청결도, 평일여부, 주말여부
model.predict([[2, 20, 80, 70, 80, 90, 0, 1]])
```

```
model.predict([[3, 10, 80, 70, 80, 90, 1, 0]])
```

```
import joblib
joblib.dump(model, 'd:/data/rides/rides_regress.model')
```

16) 놀이동산 만족도 예측 모형 불러오기

```
import joblib
model=joblib.load('d:/data/rides/rides_regress.model')
```

```
#자녀수, 거리, 놀이기구점수, 게임점수, 대기시간점수, 청결도, 평일여부, 주말여부
model.predict([[2, 20, 80, 70, 80, 90, 0, 1]])
```

```
model.predict([[3, 10, 80, 70, 80, 90, 1, 0]])
```

17) 모형의 진단과 수정

```
#기계학습모형의 성능을 진단하고 성능을 개선하는 과정
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_boston
import pandas as pd

boston = load_boston()
dfX = pd.DataFrame(boston.data, columns=boston.feature_name
s)
dfy = pd.DataFrame(boston.target, columns=["MEDV"])
df_boston = pd.concat([dfX, dfy], axis=1)
#두 변수는 비선형 관계임(인구중 하위계층 비율과 주택가격)
sns.scatterplot(x="LSTAT", y="MEDV", data=df_boston)
plt.show()
```

```
import statsmodels.api as sm

model1 = sm.OLS.from_formula("MEDV ~ LSTAT", data=df_bosto
n)
result1 = model1.fit()
print(result1.summary())
```

```
#예측값과 실제값 비교
y_hat1 = result1.predict(df_boston)
df1 = pd.concat([y_hat1, df_boston.LSTAT], axis=1).sort_val
ues("LSTAT")
df1.columns = ["Prediction", "LSTAT"]
df1.plot(x="LSTAT", style="r-", lw=3)
plt.plot(df_boston.LSTAT, df_boston.MEDV, "bo", alpha=0.5)
plt.show()
#예측력이 약한 모형
```

```
#LSTAT을 제곱한 비선형 독립변수를 추가한 모형
model2 = sm.OLS.from_formula("MEDV ~ LSTAT + I(LSTAT**2)",
data=df_boston)
result2 = model2.fit()
print(result2.summary())
#성능이 향상됨 54.4% => 64.1%
```

```
y_hat2 = result2.predict(df_boston)
df2 = pd.concat([y_hat2, df_boston.LSTAT], axis=1).sort_val
ues("LSTAT")
df2.columns = ["Prediction", "LSTAT"]
df2.plot(x="LSTAT", style="r-", lw=3)
plt.plot(df_boston.LSTAT, df_boston.MEDV, "bo", alpha=0.5)
plt.show()
#선형모형보다 좀더 적합한 모형
```

```
#범주형 변수의 예

# 방의 개수가 아주 적거나 아주 많으면 선형모형이 잘 맞지 않는다.
sns.scatterplot(x="RM", y="MEDV", data=df_boston)
plt.show()
print(df_boston['RM'])
```

```
model3 = sm.OLS.from_formula("MEDV ~ RM", data=df_boston)
result3 = model3.fit()
print(result3.summary())
```

```
import numpy as np
df_boston["CAT_RM"] = np.round(df_boston.RM)
sns.barplot(x="CAT_RM", y="MEDV", data=df_boston)
```

```
#실수형을 정수형으로 바꾸고 더미변수를 사용한 모형
model4 = sm.OLS.from_formula("MEDV ~ C(np.round(RM))+0",
data=df_boston)
result4 = model4.fit()
print(result4.summary())
# 성능이 개선됨 48.4 => 53.7%
```

```
#독립변수가 시간인 경우 특정 시점에서 경과된 시간값으로 변형해야
함
#일자별 전력수요량
df=pd.read_csv('d:/data/energy/data.csv')
df.head()
```

```
import datetime as dt

df['Date'] = pd.to_datetime(df['Date'])
# toordinal 서기 1년 1월 1일을 기준으로 경과한 날짜
df["Ordinal"] = df.Date.map(dt.datetime.toordinal)
df["Timestamp"] = df.Date.map(dt.datetime.timestamp)
df.head()
```

```
#일단위 날짜를 기준으로 분석, 날짜는 단위가 크므로 스케일링이 필
요함
model5 = sm.OLS.from_formula("Demand ~ scale(Ordinal)",
data=df)
result5 = model5.fit()
print(result5.summary())
```

```
#시간에서 활용할 수 있는 추가적인 필드들
df["Year"] = df.Date.dt.year
df["Month"] = df.Date.dt.month
df["DayOfYear"] = df.Date.dt.dayofyear #연중 몇일차
df["DayOfMonth"] = df.Date.dt.daysinmonth #일수
df["DayOfWeek"] = df.Date.dt.dayofweek
df["WeekOfYear"] = df.Date.dt.weekofyear #연중 몇주차
df["Weekday"] = df.Date.dt.weekday #요일코드
df["IsMonthStart"] = df.Date.dt.is_month_start
df["IsMonthEnd"] = df.Date.dt.is_month_end
df.tail()
```

```
formula = """
Demand ~ scale(Ordinal) + C(Month)+0 + DayOfYear +
        C(DayOfMonth)+0 + C(DayOfWeek)+0 + C(Weekday)+0 +
C(IsMonthStart)+0 + C(IsMonthEnd)+0
"""
model6 = sm.OLS.from_formula(formula, data=df)
result6 = model6.fit()
print(result6.summary())
#3.1%에서 53.7%로 향상됨
```

```
#독립변수 뿐 아니라 상황에 따라서는 종속변수도 변형할 수 있음
#실제주택가격과 예측값의 비교
plt.scatter(boston.target, y_hat1)
plt.xlabel('Real')
plt.ylabel('Predict')
plt.show()
#선형적으로 설명하기 어려운 모형
```

```
#종속변수를 제곱근 처리
model11 = sm.OLS.from_formula("np.sqrt(MEDV) ~ LSTAT",
data=df_boston)
result11 = model11.fit()
print(result11.summary())
```

```
plt.subplot(121)
plt.scatter(boston.target, y_hat1)
plt.title("MEDV ~ LSTAT")
plt.subplot(122)
plt.scatter(boston.target, (result11.predict(df_boston))**
2)
plt.title("np.sqrt(MEDV) ~ LSTAT")
plt.tight_layout()
plt.show()
```

```
#로그를 취한 모형
model12 = sm.OLS.from_formula("np.log(MEDV) ~ LSTAT",
data=df_boston)
result12 = model12.fit()
print(result12.summary())
```

```
plt.subplot(121)
plt.scatter(boston.target, y_hat1)
plt.title("MEDV ~ LSTAT")
plt.subplot(122)
plt.scatter(boston.target, np.exp(result12.predict(df_bosto
n)))
plt.title("np.log(MEDV) ~ LSTAT")
plt.tight_layout()
plt.show()
```

```
#독립변수와 종속변수 모두 로그를 취한 모형(가장 성능이 좋은 모
형)
model13=sm.OLS.from_formula("np.log(MEDV) ~ np.log(LSTAT)",
data=df_boston)
result13 = model13.fit()
print(result13.summary())
```

```
plt.subplot(121)
plt.scatter(boston.target, y_hat1)
plt.title("MEDV ~ LSTAT")
plt.subplot(122)
plt.scatter(boston.target, np.exp(result13.predict(df_bosto
n)))
plt.title("np.log(MEDV) ~ np.log(LSTAT)")
plt.tight_layout()
plt.show()
```

18) 오존량 예측(다중회귀분석)

```
import pandas as pd

df = pd.read_csv("d:/data/ozone/ozone.csv")

df.head()
```

```
#오존량, 일조량 결측값 처리
%matplotlib inline
import missingno as msno
import matplotlib.pyplot as plt

msno.matrix(df)
#흰색 - 결측값
#스파크라인(spark line) - 각 샘플의 데이터 완성도를 표현
```

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy="mean")
df = pd.DataFrame(imputer.fit_transform(df), columns=df.col
umns)
df
```

```
X=df[['Solar.R','Wind','Temp']]
y=df['Ozone']
```

```
import statsmodels.api as sm

model = sm.OLS(y, X)
result = model.fit()
result.summary()
```

```
#일조량 92, 풍량 15.5, 화씨온도 84 일때의 오존량 예측
result.predict([92,15.5,84])
```

```
result.save("d:/data/ozone/ozone_regress.model")
```

19) 오존량 예측 모형 불러오기

```
from statsmodels.regression.linear_model import OLSResults
model = OLSResults.load("d:/data/ozone/ozone_regress.model
")
```

```
#일조량 92, 풍량 15.5, 화씨온도 84 일때의 오존량 예측
model.predict([92,15.5,84])
```

```
#일조량 80, 풍량 5.5, 화씨온도 64 일때의 오존량 예측
model.predict([80,5.5,64])
```

3. 회귀분석 모형의 활용

PyCharm 프로젝트 생성 : 프로젝트 이름 - pythonweb

Python 웹 응용 프로그램을 지원하는 대표적인 Framework
  Django - 다양하고 풍부한 라이브러리, 현재 호환성 문제로 tensorflow와 keras의 최신버전이 실행되지 않는 문제가 있음
  Flask - Django에 비해 사용법이 쉽고 tensorflow, keras와의 연동도 잘 됨

A. 구구단 예제

패키지 만들기 : test

1) test/gugu.py

```
from flask import Flask, Markup, render_template, request
app = Flask(__name__)

@app.route('/',methods=['GET'])
def main():
    return render_template('gugu.html')

@app.route('/gugu_result', methods=['POST'])
def gugu_result():
    dan = int(request.form['dan'])
    result=''
    for i in range(1,10):
        result += f'{dan}x{i}={dan*i}<br>'
    #html 태그를 인식하게 하는 함수
```

```
    result=Markup(result)
    return render_template('gugu_result.html',
result=result)

if __name__ == '__main__':
    #threaded를 True로 설정하면 신경망 모형을 불러오는 코드에서
에러가 발생하므로 False로 설정
    app.run(port=8000, threaded=False)
```

2) test/templates/gugu.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h2>구구단</h2>
<form method="post" action="gugu_result">
    단
    <input type="text" name="dan">
    <input type="submit" value="확인">
</form>
</body>
</html>
```

3) test/templates/gugu_result.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h2>구구단 계산 결과</h2>
{{result}}

<a href="/">gugu main</a>
</body>
</html>
```

B. 오존량 예측

패키지 만들기 : regress

1) regress/ozone.py

```
from flask import Flask,render_template,request
from statsmodels.regression.linear_model import OLSResults

app = Flask(__name__)

@app.route('/',methods=['GET'])
def main():
    return render_template('ozone/input.html')

@app.route('/result',methods=['POST'])
def result():
    model = OLSResults.load("d:/data/ozone/ozone_regress.mo
del")
    a = float(request.form['a'])
    b = float(request.form['b'])
    c = float(request.form['c'])
    test_set = [a,b,c]
    ozone= model.predict(test_set)
    return render_template('ozone/result.html',
ozone='{:.2f}'.format(ozone[0]), a=a, b=b, c=c)

if __name__ == '__main__':
    #웹브라우저에서 실행할 때 http://localhost로 하면 느림
    #서버를 실행시킨 후 http://127.0.0.1:8000 링크를 클릭
    app.run(port=8000, threaded=False)
```

2) regress/templates/ozone/input.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h2>오존량 예측</h2>
<form method="post" action="result">
<table>
    <tr>
        <td>일조량</td>
        <td><input type="text" name="a"></td></tr>
    <tr>
        <td>풍량</td>
        <td><input type="text" name="b"></td></tr>
    <tr>
        <td>화씨온도</td>
        <td><input type="text" name="c"></td>
    </tr>
    <tr>
        <td align="center" colspan="2">
            <input type="submit" value="확인">
        </td>
    </tr>
</table>
</form>
</body>
</html>
```

3) regress/templates/ozone/result.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h2>오존량 예측 결과 : {{ozone}} </h2>
일조량: {{a}}<br>
풍량: {{b}}<br>
온도: {{c}}<br>
<a href="/">Home</a>
</body>
</html>
```

C. 놀이동산 만족도

1) regress/rides.py

```
from flask import Flask,render_template,request
import joblib

app = Flask(__name__)

@app.route('/',methods=['GET'])
def main():
    return render_template('rides/input.html')

@app.route('/result',methods=['POST'])
def result():
    model = joblib.load('d:/data/rides/rides_regress.model
')
    a = int(request.form['a'])
    b = int(request.form['b'])
    c = int(request.form['c'])
    d = int(request.form['d'])
    e = int(request.form['e'])
    f = int(request.form['f'])
    g = int(request.form['g'])
    if g==0:
        h=1
    elif g==1:
        h=0
    # 2차원 배열로 입력해야 함
    test_set = [[a,b,c,d,e,f,g,h]]
    result= model.predict(test_set)
```

```
    return render_template('rides/result.html',
point='{:.2f}'.format(result[0]), a=a, b=b, c=c, d=d, e=e,
f=f, g=g, h=h)

if __name__ == '__main__':
    #웹브라우저에서 실행할 때 http://localhost로 하면 느림
    #서버를 실행시킨 후 http://127.0.0.1:8000 링크를 클릭
    app.run(port=8000, threaded=False)
```

2) regress/templates/rides/input.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h2>놀이동산 만족도</h2>
<form method="post" action="result">
<table>
    <tr>
        <td>자녀수</td>
        <td><input type="text" name="a"></td>
    </tr>
    <tr>
        <td>거리</td>
        <td><input type="text" name="b"></td>
    </tr>
    <tr>
        <td>놀이기구점수</td>
        <td><input type="text" name="c"></td>
    </tr>
    <tr>
        <td>게임점수</td>
        <td><input type="text" name="d"></td>
    </tr>
    <tr>
        <td>대기시간점수</td>
        <td><input type="text" name="e"></td>
```

```
    </tr>
    <tr>
        <td>청결도</td>
        <td><input type="text" name="f"></td>
    </tr>
    <tr>
        <td>주말여부</td>
        <td>
            <input type="radio" name="g" value="0"> 평일
            <input type="radio" name="g" value="1"> 주말
        </td>
    </tr>
    <tr>
        <td align="center" colspan="2">
            <input type="submit" value="확인">
        </td>
    </tr>
</table>
</form>

</body>
</html>
```

3) regress/templates/rides/result.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h2>종합 만족도 예측 결과 : {{point}} </h2>
자녀수: {{a}}<br>
거리: {{b}}<br>
놀이기구점수: {{c}}<br>
게임점수: {{d}}<br>
대기시간점수: {{e}}<br>
청결도: {{f}}<br>
평일여부: {{g}}<br>
주말여부: {{h}}<br>
<a href="/">Home</a>
</body>
</html>
```