선형회귀분석

종속변수와 하나 이상의 독립변수간의 선형 관계를 모델링하는 방법

- 독립변수가 하나인 경우 단순 선형회귀
- 독립변수가 여러개인 경우 다중선형회귀

선형회귀분석 4가지 가정

- 1) 독립변수-종속변수의 선형성
- 2) 오차의 정규성 --> shapiro-wilk검정
- 3) 오차의 등분산성
- 4) 오차의 독립성 --> 더빈왓슨검정

단순 선형회귀분석

```
import seaborn as sns
import pandas as pd
df = sns.load_dataset('iris')
df.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

```
# 독립변수 : petal_length, 종속변수 : sepal_length 회귀분석 시행

#1.선형성 확인
# 절댓값 크기가 0.7이상 -> 강한 상관관계 / 0.2~0.4이하면 약한 상관관계 / 0.2 이하면 상관관계 없다.
df[['petal_length','sepal_length']].corr()
```

	petal_length	sepal_length
petal_length	1.000000	0.871754
sepal_length	0.871754	1.000000

```
X= df[['petal_length']]
y= df['sepal_length']
```

```
from statsmodels.formula.api import ols
lr = ols('sepal_length ~ petal_length', data = df).fit()
```

```
# 회귀계수
lr.params
```

```
Intercept       4.306603
petal_length    0.408922
dtype: float64
```

```
# 예측
y_pred = lr.predict(X)
y_pred
```

```
0       4.879095
1       4.879095
2       4.838202
3       4.919987
4       4.879095
          ...
145     6.432999
146     6.351215
147     6.432999
```

```
148    6.514784
149    6.392107
Length: 150, dtype: float64
```

```
# 평가 (mse, rmse)
# 숫자가 작을수록 좋음
res = y-y_pred
mse = (res**2).sum() / len(y)
import numpy as np
rmse = np.sqrt(mse)
```

```
lr.summary()
```

OLS Regression Results

Dep. Variable:	sepal_length	**R-squared:**	0.760
Model:	OLS	**Adj. R-squared:**	0.758
Method:	Least Squares	**F-statistic:**	468.6
Date:	Tue, 21 May 2024	**Prob (F-statistic):**	1.04e-47
Time:	08:14:07	**Log-Likelihood:**	-77.020
No. Observations:	150	**AIC:**	158.0
Df Residuals:	148	**BIC:**	164.1
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	4.3066	0.078	54.939	0.000	4.152	4.462
petal_length	0.4089	0.019	21.646	0.000	0.372	0.446

Omnibus:	0.207	**Durbin-Watson:**	1.867
Prob(Omnibus):	0.902	**Jarque-Bera (JB):**	0.346
Skew:	0.069	**Prob(JB):**	0.841
Kurtosis:	2.809	**Cond. No.**	10.3

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

```
#sepal_length = 4.3066 + petal_length * 0.4089
```

다중회귀분석

필요 가정

- 독립변수와 종속변수간의 선형성
- 오차의 정규성 --> shapiro.
- 오차의 독립성 --> durbin_watson
- 오차의 등분산성 --> bartlett
- 다중공선성 위배 X --> variance_inflation_factor

[ ] *숨겨진 셀 8개*

sklearn 사용

[ ] *숨겨진 셀 8개*

확장된 선형회귀 (릿지,라쏘, 엘라스틱넷)

목적 : 과적합 방지, 변수선택, 차원 축소

- 릿지 : L2 정규화 -> 회귀 계수의 제곱의 합에 패널티(alpha)를 적용
- 라쏘 : L1 정규화 -> 회귀 계수의 절대값의 합에 패널티(alpha)를 적용
- 엘라스틱 넷 : L1과 L2 정규화 두 개 조합 -> 릿지와 라쏘의 장점을 결합하여, L1 패널티(alpha)로 변수 선택과 L2 패널티로 계수 축소를 동시에 수행

패널티(alpha)

알파가 크면 패널티를 강하게 적용하기 때문에 과적합을 막을 수 있지만

너무 커지면 과소적합의 가능성이 존재. 따라서 최적의 알파를 찾아야함 -> "그리드 서치 알고리즘"

그리드 서치(GridSerchCV)

주어진 하이퍼파라미터 목록의 조합을 모두 탐색하여 최적의 파라미터를 찾는 알고리즘

```
from sklearn.linear_model import Ridge, Lasso, ElasticNet
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import GridSearchCV

#릿지
ridge = Ridge()
parameter = {'alpha' : (0.01, 0.2, 0.7, 1.5)}
ridge_regressor = GridSearchCV(ridge, parameter, scoring = 'r2')
ridge_regressor.fit(X_train, y_train)

#최적 파라미터 출력 및 성능
print(f"Best parameter Ridge : {ridge_regressor.best_params_}")
print(f"Best score Ridge : {ridge_regressor.best_score_}")
```

```
Best parameter Ridge : {'alpha': 0.01}
Best score Ridge : 0.8286498244963101
```

```
# 라쏘
lasso = Lasso()
lasso_regressor = GridSearchCV(lasso, parameter,scoring = 'r2')
lasso_regressor.fit(X_train,y_train)

#최적 파라미터 출력 및 성능
print(f"Best parameter Lasso : {lasso_regressor.best_params_}")
print(f"Best score Lasso : {lasso_regressor.best_score_}")
```

```
Best parameter Lasso : {'alpha': 0.01}
Best score Lasso : 0.814156502671422
```

```
# 엘라스틱넷
parameters = {
    'alpha' : [0.01,0.1,0.5],
    'l1_ratio' : [0.1,0.5, 0.7,1]
}
elastic_net = ElasticNet()
elastic_net_regressor = GridSearchCV(elastic_net, parameters, scoring= 'r2')

elastic_net_regressor.fit(X_train,y_train)

print(f"Best parameter Elastic : {elastic_net_regressor.best_params_}")
print(f"Best score Elastic : {elastic_net_regressor.best_score_}")
```

```
Best parameter Elastic : {'alpha': 0.5, 'l1_ratio': 1}
Best score Elastic : 0.6324542917848802
```

SVM(Support Vector Machine)를 통한 회귀

데이터들 간 가장 가까이 있도록 만드는 선을 찾고, 예측 오차가 허용 범위 이내로 들어오게 해서 예측을 시행

- kernel : linear, rbf 등

```
import pandas as pd
df = pd.read_csv('./sample_data/california_housing_train.csv')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17000 entries, 0 to 16999
Data columns (total 9 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           17000 non-null  float64
 1   latitude            17000 non-null  float64
 2   housing_median_age  17000 non-null  float64
 3   total_rooms         17000 non-null  float64
 4   total_bedrooms      17000 non-null  float64
 5   population          17000 non-null  float64
 6   households          17000 non-null  float64
 7   median_income       17000 non-null  float64
 8   median_house_value  17000 non-null  float64
```

```
dtypes: float64(9)
memory usage: 1.2 MB
```

```
#median_hose_value 예측
df.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_val
0	-114.31	34.19	15.0	5612.0	1283.0	1015.0	472.0	1.4936	6690
1	-114.47	34.40	19.0	7650.0	1901.0	1129.0	463.0	1.8200	8010
2	-114.56	33.69	17.0	720.0	174.0	333.0	117.0	1.6509	8570
3	-114.57	33.64	14.0	1501.0	337.0	515.0	226.0	3.1917	7340
4	-114.57	33.57	20.0	1454.0	326.0	624.0	262.0	1.9250	6550

```
y = df['median_house_value']
X =df.drop('median_house_value', axis = 1)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train,y_test = train_test_split(X,y,test_size = 0.3, random_state = 42)
X_train
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
9173	-119.03	35.32	12.0	2721.0	549.0	1294.0	523.0	2.5575
16528	-122.66	39.03	27.0	1446.0	329.0	594.0	255.0	1.1650
4819	-118.09	33.94	33.0	1976.0	404.0	1379.0	395.0	3.8542
6818	-118.30	34.06	20.0	1782.0	896.0	1749.0	823.0	2.2094
7717	-118.38	34.07	16.0	4814.0	1381.0	1897.0	1209.0	3.3725
...	...	...	...	...	...	...	...	...
11284	-121.13	37.74	21.0	2376.0	475.0	1175.0	441.0	3.6016
11964	-121.38	38.62	41.0	774.0	144.0	356.0	150.0	3.5625
5390	-118.15	33.91	38.0	901.0	205.0	760.0	208.0	2.9643
860	-117.07	32.56	9.0	3648.0	895.0	3293.0	840.0	3.0992
15795	-122.41	37.75	52.0	2452.0	623.0	1932.0	549.0	2.3903

11900 rows × 8 columns

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train_sc = pd.DataFrame(sc.fit_transform(X_train), columns = X_train.columns)
X_test_sc = pd.DataFrame(sc.transform(X_test), columns = X_train.columns)
X_test_sc
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
0	-0.648904	1.000668	-1.565599	0.995477	0.891252	0.863894	1.086240	-0.140764
1	0.712317	-0.711972	1.848829	0.044641	-0.277535	-0.240187	-0.225916	2.218276
2	-0.240040	0.631000	-1.009762	0.481499	0.466452	0.866428	0.503632	-0.698127
3	1.071320	-0.754086	-1.565599	1.275374	0.802068	1.332727	0.918675	0.191534
4	0.652483	-0.777483	1.848829	-0.595450	-0.455905	-0.247789	-0.359967	-1.218458
...	...	...	...	...	...	...	...	...
5095	1.046389	-0.735369	0.340128	-0.573675	-0.627233	-0.467423	-0.702829	-0.578726
5096	0.069102	0.448505	-1.089167	-0.577304	-0.549783	-0.144731	-0.496596	-1.096027
5097	-0.429514	-0.318907	1.134181	-0.282890	-0.460599	-0.629614	-0.494019	-0.096054
5098	0.712317	-0.871070	1.610613	0.067323	0.351451	-0.115165	0.377315	-0.299652
5099	1.210932	-1.371760	1.213586	-0.900752	-0.756316	-0.480094	-0.746654	-1.008905

5100 rows × 8 columns

```
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error

svm_regressor = SVR(kernel ='rbf')

svm_regressor.fit(X_train_sc, y_train)
```

```
▾ SVR
SVR()
```

```
y_pred_train = svm_regressor.predict(X_train_sc)
mse_train = mean_squared_error(y_train, y_pred_train)
print('Train MSE : ',mse_train)

y_pred_test = svm_regressor.predict(X_test_sc)
mse_test = mean_squared_error(y_test, y_pred_test)
print('Test MSE : ',mse_test)
```

```
Train MSE :  13971430770.149454
Test MSE :  14777723073.742445
```

```
from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV
param_grid = [{'kernel' : ['linear'], 'C' : [10, 15, 20, 30]},
              {'kernel' : ['rbf'], 'C' : [1,3,10],
               'gamma' : [0.02,0.1,0.3]}]

svm_grid = GridSearchCV(SVR(), param_grid, cv = 5)
svm_grid.fit(X_train_sc, y_train)

# 최적 파라미터 출력
print('최적 파라미터 : ', svm_grid.best_params_)

y_pred_test = svm_grid.best_estimator_.predict(X_test_sc)

mse_test = mean_squared_error(y_test, y_pred_test)
print('after_gird_MSE : ', mse_test)
```

```
최적 파라미터 :  {'C': 30, 'kernel': 'linear'}
after_gird_MSE :  6585221450.626026
```

```
14777723073.742445 - 6585221450.626026
```

```
8192501623.116419
```

의사결정나무를 통한 회귀

- 데이터를 분할해서 예측을 수행
- 각 분할된 영역에서 목표변수 평균값을 사용해서 예측
- 하이퍼파라미터 조정을 통해 최적 모델을 구축할 수 있음

장점

- 데이터 비선형관계를 잘 모델링 할 수 있음
- 트리 구조가 직관적이고 이해가 쉬움
- 스케일링과 같은 전처리 필요 X

단점

- 트리의 깊이가 너무 깊을 때는 훈련데이터에 과적합 가능성 존재

```
import seaborn as sns
mpg = sns.load_dataset('mpg')
mpg.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   mpg           398 non-null    float64
 1   cylinders     398 non-null    int64
 2   displacement  398 non-null    float64
 3   horsepower    392 non-null    float64
 4   weight        398 non-null    int64
```

```
 5   acceleration  398 non-null    float64
 6   model_year    398 non-null    int64
 7   origin        398 non-null    object
 8   name          398 non-null    object
dtypes: float64(4), int64(3), object(2)
memory usage: 28.1+ KB
```

```
mpg.head()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	name
0	18.0	8	307.0	130.0	3504	12.0	70	usa	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693	11.5	70	usa	buick skylark 320
2	18.0	8	318.0	150.0	3436	11.0	70	usa	plymouth satellite
3	16.0	8	304.0	150.0	3433	12.0	70	usa	amc rebel sst
4	17.0	8	302.0	140.0	3449	10.5	70	usa	ford torino

```
mpg.dropna(inplace = True)
```

```
mpg.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 392 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   mpg           392 non-null    float64
 1   cylinders     392 non-null    int64
 2   displacement  392 non-null    float64
 3   horsepower    392 non-null    float64
 4   weight        392 non-null    int64
 5   acceleration  392 non-null    float64
 6   model_year    392 non-null    int64
 7   origin        392 non-null    object
 8   name          392 non-null    object
dtypes: float64(4), int64(3), object(2)
memory usage: 30.6+ KB
```

```
mpg1 = mpg.drop('name',axis = 1)
mpg1 = pd.get_dummies(mpg1, columns = ['origin'])
mpg1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 392 entries, 0 to 397
Data columns (total 10 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   mpg            392 non-null    float64
 1   cylinders      392 non-null    int64
 2   displacement   392 non-null    float64
 3   horsepower     392 non-null    float64
 4   weight         392 non-null    int64
 5   acceleration   392 non-null    float64
 6   model_year     392 non-null    int64
 7   origin_europe  392 non-null    bool
 8   origin_japan   392 non-null    bool
 9   origin_usa     392 non-null    bool
dtypes: bool(3), float64(4), int64(3)
memory usage: 25.6 KB
```

```
X = mpg1.drop('mpg', axis = 1)
y = mpg1['mpg']
X
y
```

```
0      18.0
1      15.0
2      18.0
3      16.0
4      17.0
       ...
393    27.0
394    44.0
395    32.0
396    28.0
397    31.0
Name: mpg, Length: 392, dtype: float64
```

```
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 = 1)

from sklearn.tree import DecisionTreeRegressor
dt = DecisionTreeRegressor(max_depth = 10, random_state = 1, criterion = 'squared_error')
dt.fit(X_train, y_train)
print(dt.score(X_test, y_test))
```

```
0.7793050599320761
```

```
from sklearn.metrics import mean_squared_error
import numpy as np

y_test_pred = dt.predict(X_test)
print('dicision tree MSE',mean_squared_error(y_test, y_test_pred))
print('dicision tree RMSE', np.sqrt(mean_squared_error(y_test, y_test_pred)))
```

```
dicision tree MSE 15.299075387496774
dicision tree RMSE 3.911403250432864
```

코딩을 시작하거나 AI로 코드를 생성하세요.