파이썬을 활용한 중고차 가격 예측

20181111 박초연

I. 개요
   A. Data Set 확인
   B. 데이터 전처리
   C. 데이터 분석
   D. 분석 결과
   E. 코드 링크

II. Data Set 확인

A. 데이터 불러오기
   1. 사용한 데이터는 kaggle의 ‘Used Cars Price Prediction’ 중 train-data 사용.
   2. Url : https://www.kaggle.com/avikasliwal/used-cars-price-prediction

B. 데이터 확인

	Unnamed: 0	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_Price	Price
0	0	Maruti Wagon R LXI CNG	Mumbai	2010	72000	CNG	Manual	First	26.6 km/kg	998 CC	58.16 bhp	5.0	NaN	1.75
1	1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	Diesel	Manual	First	19.67 kmpl	1582 CC	126.2 bhp	5.0	NaN	12.50

III. 데이터 전처리

A. Column 설명
   1. Unnamed: 0 -> index
   2. Name -> 자동차 모델명
   3. Location -> 자동차의 사용 지역
   4. Year -> 출고년도
   5. Kilometers_Driven -> 자동차의 주행거리
   6. Puel_Type -> 사용되는 연료
   7. Transmission -> 변속기 종류
   8. Owner_Type -> 현 사용자가 몇번째 사용자인지
   9. Mileage -> 연비
   10. Engine -> 엔진
   11. Power -> 마력
   12. Seats -> 좌석 수
   13. New_price -> 출고가
   14. Price -> 중고가

B. 무의미한 변수 제거
   1. ‘Unnamed: 0’ 안의 값은 인덱스와 같으므로 분석에 필요하지 않다고 판단하여 제외함.

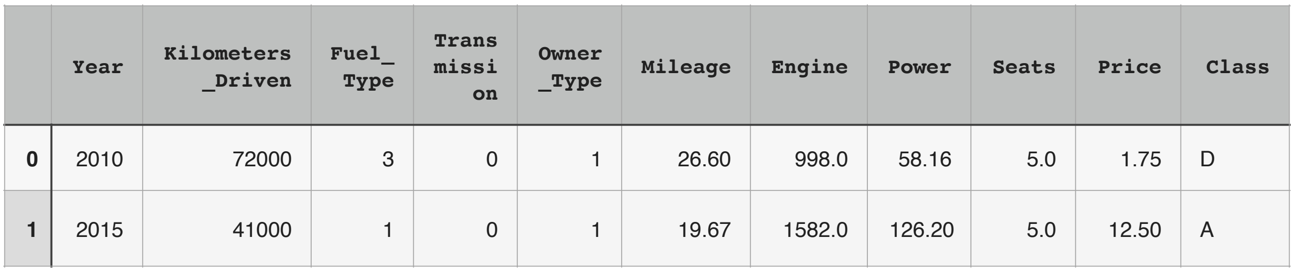
    2. ‘Name’, ‘Location’의 경우 데이터 값이 매우 다양함. 이를 통해 어떠한 특징을 이끌어내어, 중고차 가격을 예측하긴 어렵다고 판단하여 제외함.
    3. ‘New_Price’ 데이터의 값을 보면 결측값이 전체의 약 86.31%로 너무 많아 제외함.

C. 문자형 자료를 숫자형 자료로 변환
    1. ‘Mileage’, ’Engine’, ’Power’ 의 경우 숫자 정보를 가지고 있지만, 뒤에 단위가 붙어 문자형 데이터로 읽어짐. 이러한 문제를 해결하기 위해 단위를 지우고 데이터의 형태를 ‘float’으로 바꿈.
    2. ‘Power’는 공백이 포함되어 단위를 삭제해도 데이터의 형태가 안바뀜. 따라서 공백을 제거함. 또한, 결측값이 ‘null’이라는 문자로 들어가 있어 NaN 값으로 바꿈.

D. 범주형 자료를 숫자형 자료로 변환
    1. ‘Fuel_Type’, ‘Transmission’, ‘Owner_Type’은 문자로 되어 있지만, ’Price’값에 영향을 줄 수 있다고 판단하여 숫자로 변경함.
    2. 같은 value를 같은 숫자로 표현함.

E. 결측값 제거
    1. ‘Mileage’, ’Engine’, ’Power’, ‘Seats’ 안에 결측값이 포함되어 있어 결측값을 제거함.

F. ‘Price’의 범주형 자료 만들기
    1. ‘Price’는 연속형 자료이고 연속형 자료는 예측의 정확도가 떨어지기 때문에 범주를 나눠주면 정확도가 올라감.
    2. 데이터의 비율이 비슷하게 범주를 나누고 ‘Class’ 변수를 만들어 입력함.
    3. 범주는 사분위수 사용함. 75% 이상, 50% 이상, 25% 이상, 25% 이하



	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	Price	Class
0	2010	72000	3	0	1	26.60	998.0	58.16	5.0	1.75	D
1	2015	41000	1	0	1	19.67	1582.0	126.20	5.0	12.50	A

IV. 데이터 분석

A. 다중 선형 회귀 분석
    1. 다중 선형 회귀 분석을 통해 ‘Price’에 영향을 주는 가장 유의미한 변수 식을 도출함.
    2. Price = 1.0376 * Year - 1.8900 * Fuel_Type + 2.6342 * Transmission - 0.1327 * Mileage + 0.0012 * Engine + 0.1249 * Power - 0.9559 * Seats
    3. 여러차례 OLS Regression Results 확인 후, ‘Kilometers_Driven’과 ‘Owner_Type’은 ‘Price’의 변화에 큰 영향을 주지 못한다고 판단함.

B. 분석에 활용되는 Train & Test data를 변하기 않게 설정
    1. shuffle=False
    2. random_state=20181111
    3. random_state=0

C. 서포트 벡터 머신(SVM) <random_state=20181111>
    1. ‘linear’
        a) C=1, gamma=1 : accuracy = 0.7343550446998723
        b) parameter 최적화 후 C=500, gamma=0.01 : accuracy = 0.7318007662835249
        c) parameter 최적화 후 C=500, gamma=0.001 : accuracy = 0.7318007662835249
        d) parameter 최적화 후 C=500, gamma=0.0001 : accuracy = 0.7318007662835249
        e) GridSearchCV의 gamma 값을 바꿔도 accuracy가 같음.
    2. ‘rbf’
        a) C=1, gamma=1 : accuracy = 0.7692635163899532
        b) parameter 최적화 후 C=1, gamma=1 : accuracy = 0.7692635163899532

D. 신경망(MLP) <random_state=20181111>
   1. 'lbfgs'
      a) hidden_layer_sizes=(10,10) : accuracy = 0.7432950191570882
      b) parameter 최적화 후 alpha =0.1, hidden_layer_sizes=(500,) : accuracy = 0.7420178799489144
      c) parameter 최적화 후 alpha = 0.01, hidden_layer_sizes=(500,500) : accuracy = 0.7420178799489144
      d) GridSearchCV의 hidden_layer_sizes 값을 바꿔도 accuracy가 같음.

E. 서포트 벡터 머신(SVM) <random_state=0>
   1. 'linear'
      a) C=1, gamma=1 : accuracy = 0.7343550446998723
      b) parameter 최적화 후 C=500, gamma=0.01 : accuracy = 0.7318007662835249
      c) parameter 최적화 후 C=500, gamma=0.001 : accuracy = 0.7318007662835249
      d) parameter 최적화 후 C=500, gamma=0.0001 : accuracy = 0.7318007662835249
      e) GridSearchCV의 gamma 값을 바꿔도 accuracy가 같음.
   2. 'rbf'
      a) C=1, gamma=1 : accuracy = 0.7692635163899532
      b) parameter 최적화 후 C=1, gamma=1 : accuracy = 0.7692635163899532

F. 신경망(MLP) <random_state=0>
   1. 'lbfgs'
      a) hidden_layer_sizes=(10,10) : accuracy = 0.7352064708386548
      b) parameter 최적화 후 alpha =0.001, hidden_layer_sizes=(500,500) : accuracy = 0.7432950191570882

V. 분석 결과
   A. 서포트 벡터 머신 중 비선형분류 'rbf' 방식은 Train & Test data가 변하더라도 동일한 parameter와 accuracy가 나왔으며, accuracy가 가장 높게 나옴.
   B. 서포트 벡터 머신 중 비선형분류 'rbf' 방식이 'Price'를 가장 잘 예측함.

VI. 코드 링크

<random_state=20181111>
https://colab.research.google.com/drive/1P0e1DL4_fPNSlAx-jQPNRaPwiCCfmF-F?usp=sharing

<random_state=0>
https://colab.research.google.com/drive/1EUURXWsBNB_oN0AptjA4JzuD-7GXA2yU?usp=sharing

데이터 불러오기

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, ca
```

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

```
data = pd.read_csv('/content/drive/My Drive/3학년1학기/Python_Application/
data.head(2)
```

	Unnamed: 0	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmissi
0	0	Maruti Wagon R LXI CNG	Mumbai	2010	72000	CNG	Mar
1	1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	Diesel	Mar

```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6019 entries, 0 to 6018
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Unnamed: 0         6019 non-null   int64  
 1   Name               6019 non-null   object 
 2   Location           6019 non-null   object 
 3   Year               6019 non-null   int64  
 4   Kilometers_Driven  6019 non-null   int64  
 5   Fuel_Type          6019 non-null   object 
 6   Transmission       6019 non-null   object 
 7   Owner_Type         6019 non-null   object 
 8   Mileage            6017 non-null   object 
 9   Engine             5983 non-null   object 
 10  Power              5983 non-null   object 
 11  Seats              5977 non-null   float64
 12  New_Price          824 non-null    object 
 13  Price              6019 non-null   float64
dtypes: float64(2), int64(3), object(9)
memory usage: 658.5+ KB
```

데이터 전처리

무의미한 변수 제거

> 'Unnamed: 0' 인덱스와 같음

> 'Name', 'Location' 매우 다양함. 때문에 어떤 특징을 이끌어내서 중고가와 연결 어려움.

> 'New_Price' 결측값이 너무 많음. 전체의 약 86.31%

↳ *숨겨진 셀 3개*

문자형 자료를 숫자형 자료로 변환

```python
df['Engine']=df['Engine'].str.replace('CC','').astype(float)
df['Mileage']=df['Mileage'].str.replace('kmpl','')
df['Mileage']=df['Mileage'].str.replace('km/kg','').astype(float)
df['Power']=df['Power'].str.replace('bhp','')
df['Power']=df['Power'].str.split(' ').str[0]
df['Power'][df['Power'] == 'null'] = np.NaN
df['Power']=df['Power'].astype(float)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:6: SettingWith
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs
```

```python
df.head(2)
```

	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Eng
0	2010	72000	CNG	Manual	First	26.60	9
1	2015	41000	Diesel	Manual	First	19.67	15

```python
df.dtypes
```

```
Year                   int64
Kilometers_Driven      int64
Fuel_Type             object
Transmission          object
Owner_Type            object
Mileage              float64
Engine               float64
Power                float64
Seats                float64
Price                float64
dtype: object
```

범주형 자료를 숫자형 자료로 변환

```python
df['Fuel_Type'].value_counts()
```

```
Diesel      3205
Petrol      2746
CNG           56
LPG           10
Electric       2
Name: Fuel_Type, dtype: int64
```

```python
df['Fuel_Type'][df['Fuel_Type']=='Diesel'] = 1
df['Fuel_Type'][df['Fuel_Type']=='Petrol'] = 2
df['Fuel_Type'][df['Fuel_Type']=='CNG'] = 3
df['Fuel_Type'][df['Fuel_Type']=='LPG'] = 4
df['Fuel_Type'][df['Fuel_Type']=='Electric'] = 5
df['Fuel_Type']=df['Fuel_Type'].astype(int)
df['Transmission'][df['Transmission']=='Manual'] = 0
df['Transmission'][df['Transmission']=='Automatic'] = 1
df['Transmission']=df['Transmission'].astype(int)
df['Owner_Type'][df['Owner_Type']=='First'] = 1
df['Owner_Type'][df['Owner_Type']=='Second'] = 2
df['Owner_Type'][df['Owner_Type']=='Third'] = 3
df['Owner_Type'][df['Owner_Type']=='Fourth & Above'] = 4
df['Owner_Type']=df['Owner_Type'].astype(int)
```

```python
df.head(2)
```

	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Eng
0	2010	72000	3	0	1	26.60	9
1	2015	41000	1	0	1	19.67	15

```python
df.dtypes
```

```
Year                   int64
Kilometers_Driven      int64
Fuel_Type              int64
Transmission           int64
Owner_Type             int64
Mileage              float64
Engine               float64
Power                float64
Seats                float64
Price                float64
dtype: object
```

결측값 제거

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6019 entries, 0 to 6018
Data columns (total 10 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Year               6019 non-null   int64  
 1   Kilometers_Driven  6019 non-null   int64  
 2   Fuel_Type          6019 non-null   int64  
 3   Transmission       6019 non-null   int64  
 4   Owner_Type         6019 non-null   int64  
 5   Mileage            6017 non-null   float64
 6   Engine             5983 non-null   float64
 7   Power              5876 non-null   float64
 8   Seats              5977 non-null   float64
 9   Price              6019 non-null   float64
dtypes: float64(5), int64(5)
memory usage: 470.4 KB
```

```
df1 = df.dropna()
df1 = df1.reset_index(drop=True)
df1.head(2)
```

	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Eng
0	2010	72000	3	0	1	26.60	9
1	2015	41000	1	0	1	19.67	15

```
df1.isnull().sum()
```

```
Year                 0
Kilometers_Driven    0
Fuel_Type            0
Transmission         0
Owner_Type           0
Mileage              0
Engine               0
Power                0
Seats                0
Price                0
dtype: int64
```

Price의 범주형 자료 만들기

> 75% 이상, 50% 이상, 25% 이상, 25% 이하

> 연속형 자료는 예측의 정확도가 떨어짐.

```
df1.Price.describe()
```

```
count    5872.000000
mean        9.603919
std        11.249453
min         0.440000
25%         3.517500
50%         5.750000
75%        10.000000
max       160.000000
Name: Price, dtype: float64
```

```
n = df1.shape[0]
label = pd.DataFrame(index=df1.index,columns=['Class'])
for i in range(5872):
  j = df1.Price.loc[i]
  if j >= 10.000000:
    label.loc[i,'Class'] = "A"
  elif j >= 5.750000:
    label.loc[i,'Class'] = "B"
  elif j >= 3.517500:
    label.loc[i,'Class'] = "C"
  else:
    label.loc[i,'Class'] = "D"
```

```
label['Class'].value_counts()
```

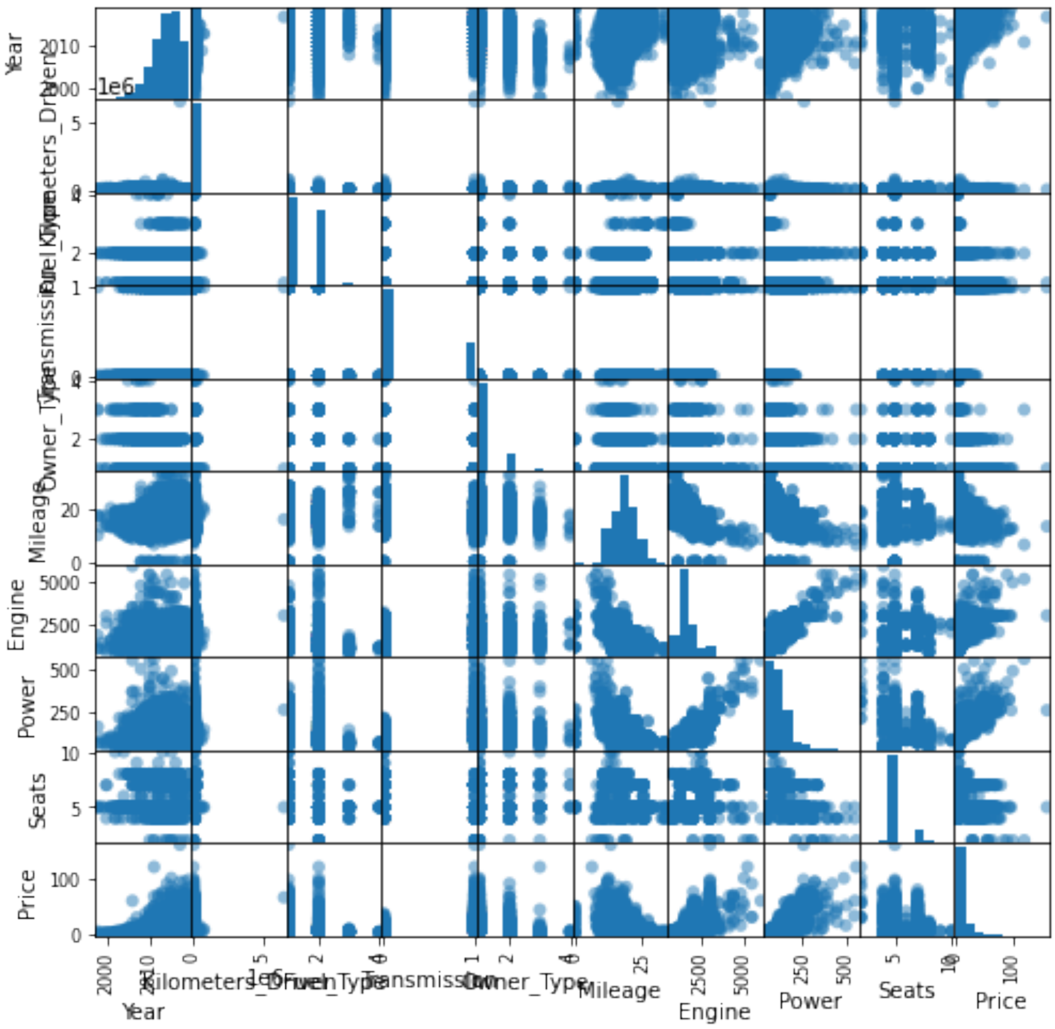
```
A    1485
D    1468
C    1466
B    1453
Name: Class, dtype: int64
```

```
df2 = pd.merge(df1,label,left_on=df1.index,right_on=label.index).iloc[:,
df2.head(2)
```

	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Enç
0	2010	72000	3	0	1	26.60	9
1	2015	41000	1	0	1	19.67	15

▾ 다중 선형 회귀분석

```
# 산점도 행렬
p = pd.plotting.scatter_matrix(df2, figsize=(8, 8), marker='o')
```



```
from statsmodels.formula.api import ols
reg_simp = ols('Price ~ Year + Fuel_Type + Transmission + Mileage + Engi
reg_simp.summary()
```

```
                          OLS Regression Results
    Dep. Variable:     Price                  R-squared:       0.695
         Model:        OLS               Adj. R-squared:       0.695
        Method:        Least Squares          F-statistic:     1912.
         Date:         Tue, 07 Jul 2020 Prob (F-statistic):   0.00
         Time:         13:23:31          Log-Likelihood:       -19054.
 No. Observations:     5872                       AIC:         3.812e+04
    Df Residuals:      5864                       BIC:         3.818e+04
       Df Model:       7
 Covariance Type:      nonrobust
                    coef      std err     t      P>|t|   [0.025     0.975]
     Intercept  -2086.1610  56.300  -37.054  0.000  -2196.530  -1975.792
        Year     1.0376      0.028   36.902   0.000  0.982       1.093
     Fuel_Type  -1.8900      0.201   -9.408   0.000  -2.284      -1.496
 Transmission   2.6342       0.243   10.850   0.000  2.158       3.110
      Mileage   -0.1327      0.030   -4.448   0.000  -0.191      -0.074
      Engine    0.0012       0.000   2.958    0.003  0.000       0.002
       Power    0.1249       0.004   31.797   0.000  0.117       0.133
       Seats    -0.9559      0.137   -6.984   0.000  -1.224      -0.688
      Omnibus:       4452.126   Durbin-Watson:     2.039
 Prob(Omnibus):      0.000      Jarque-Bera (JB):  359805.135
        Skew:        2.983          Prob(JB):      0.00
      Kurtosis:      40.882         Cond. No.      1.82e+06


 Warnings:
 [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
 [2] The condition number is large, 1.82e+06. This might indicate that there are
 strong multicollinearity or other numerical problems.
```

Train & Test data

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

```
### 입력변수(x)와 출력변수(y)
x = df2[['Year','Fuel_Type','Transmission','Mileage','Engine','Power','S
x_std = StandardScaler().fit_transform(x)
y = df2.Class
```

```
### Train & Test data
x_train, x_test, y_train, y_test = train_test_split(x_std,y,test_size=0.
                                                    random_state=2018111
```

서포트 벡터 머신(SVM)

```
from sklearn.svm import SVC
```

선형분류(linear)

```
### SVM
svc = SVC(kernel='linear', C=1, gamma=1)
model = svc.fit(x_train, y_train)
```

```
### 예측
y_pred = model.predict(x_test)
y_pred
```

```
array(['B', 'C', 'C', ..., 'B', 'D', 'D'], dtype=object)
```

```
### accuracy
model.score(x_test, y_test)
```

```
0.7343550446998723
```

```
### 교차표
pd.crosstab(y_test, y_pred)
```

col_0	**A**	**B**	**C**	**D**
Class				
A	498	69	3	0
B	54	404	131	3
C	8	114	373	108
D	1	14	119	450

```
from sklearn.model_selection import GridSearchCV
# Set the parameters by cross-validation
tuned_parameters = {'kernel': ['linear'],
                    'C': [0.01, 0.1, 1, 10, 50, 100, 500 ,1000],
                    'gamma': [0.01, 0.05, 0.1, 0.5, 1, 10 , 50]}
```

```
grid = GridSearchCV(SVC(), tuned_parameters)
%time grid.fit(x_train, y_train)
```

```
CPU times: user 12min 23s, sys: 116 ms, total: 12min 23s
Wall time: 12min 24s
GridSearchCV(cv=None, error_score=nan,
             estimator=SVC(C=1.0, break_ties=False, cache_size=200,
                           class_weight=None, coef0=0.0,
                           decision_function_shape='ovr', degree=3,
                           gamma='scale', kernel='rbf', max_iter=-1,
                           probability=False, random_state=None, shrinking=
                           tol=0.001, verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid={'C': [0.01, 0.1, 1, 10, 50, 100, 500, 1000],
                         'gamma': [0.001, 0.01, 0.05, 0.1, 0.5, 1, 10, 50],
                         'kernel': ['linear']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

```
grid.best_params_
```

```
{'C': 500, 'gamma': 0.001, 'kernel': 'linear'}
```

```
### SVM
# svc = SVC(kernel="linear",C=500, gamma=0.01)
svc = SVC(kernel="linear",C=grid.best_params_['C'], gamma=grid.best_para
model = svc.fit(x_train, y_train)
```

```
### 예측
y_pred = model.predict(x_test)
```

```
### accuracy
model.score(x_test, y_test)
```

```
0.7318007662835249
```

```
### 교차표
pd.crosstab(y_test, y_pred)
```

col_0	A	B	C	D
Class				
A	495	73	2	0
B	54	401	134	3
C	8	112	377	106
D	1	14	123	446

비선형분류(rbf)

```
### SVM
svc = SVC(kernel="rbf",C=1, gamma=1)
model = svc.fit(x_train, y_train)
```

```
### 예측
y_pred = model.predict(x_test)
```

```
### accuracy
model.score(x_test, y_test)
```

```
0.7692635163899532
```

```
### 교차표
pd.crosstab(y_test, y_pred)
```

col_0	A	B	C	D
Class				
A	519	49	2	0
B	52	402	135	3
C	8	81	434	80
D	5	12	115	452

```
from sklearn.model_selection import GridSearchCV
tuned_parameters = {'kernel': ['rbf'],
                    'C': [0.01, 0.1, 1, 10, 50],
                    'gamma': [0.1, 0.5, 1, 10, 50]}
```

```
grid = GridSearchCV(SVC(), tuned_parameters)
%time grid.fit(x_train, y_train)
```

```
CPU times: user 53.4 s, sys: 15 ms, total: 53.4 s
Wall time: 53.5 s
GridSearchCV(cv=None, error_score=nan,
             estimator=SVC(C=1.0, break_ties=False, cache_size=200,
                           class_weight=None, coef0=0.0,
                           decision_function_shape='ovr', degree=3,
                           gamma='scale', kernel='rbf', max_iter=-1,
                           probability=False, random_state=None, shrinking=
                           tol=0.001, verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid={'C': [0.01, 0.1, 1, 10, 50],
                         'gamma': [0.1, 0.5, 1, 10, 50], 'kernel': ['rbf']}
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

```
grid.best_params_
```

```
{'C': 1, 'gamma': 1, 'kernel': 'rbf'}
```

```
### SVM
# svc = SVC(kernel="rbf",C=10, gamma=0.1)
svc = SVC(kernel="rbf",C=grid.best_params_['C'], gamma=grid.best_params_
model = svc.fit(x_train, y_train)
```

```
### 예측
y_pred = model.predict(x_test)
```

```
### accuracy
model.score(x_test, y_test)
```

```
0.7692635163899532
```

```
### 교차표
pd.crosstab(y_test, y_pred)
```

col_0	A	B	C	D
Class				
A	519	49	2	0
B	52	402	135	3
C	8	81	434	80
D	5	12	115	452

신경망(MLP)

```
from sklearn.neural_network import MLPClassifier
```

```
print(x_std.mean(axis=0), x_std.var(axis=0))
```

```
[ 2.29081059e-14  7.62332976e-17 -3.63015703e-17 -4.17468058e-17
 -1.08904711e-17 -2.00868689e-16  5.25162717e-16] [1. 1. 1. 1. 1. 1. 1.]
```

```
### [0,1] 조정
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.4)
# x_train
x_train_min = x_train.min(axis=0)
x_train_max = (x_train - x_train_min).max(axis=0)
x_train = (x_train - x_train_min) / x_train_max
# x_test
x_test = (x_test - x_train_min) / x_train_max
```

```
### MLP
mlp = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(10,10))
model = mlp.fit(x_train, y_train)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_p
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
```

```
### 예측
y_pred = model.predict(x_test)
```

```
### accuracy
model.score(x_test, y_test)
```

```
0.7432950191570882
```

```
### 교차표
pd.crosstab(y_test, y_pred)
```

col_0	**A**	**B**	**C**	**D**
Class				
A	551	49	3	0
B	55	387	76	3
C	8	157	345	109
D	1	14	128	463

```
from sklearn.model_selection import GridSearchCV
tuned_parameters = {'solver':['lbfgs'],
                   'alpha':[0.01,0.1,1,10],
                   'hidden_layer_sizes':[(5,5),(10,10),(100,),(100,100),
```

```
grid = GridSearchCV(MLPClassifier(),tuned_parameters)
%time grid.fit(x_train, y_train)
```

```
grid.best_params_
```

```
{'alpha': 0.01, 'hidden_layer_sizes': (500, 500), 'solver': 'lbfgs'}
```

```
### MLP
mlp = MLPClassifier(solver='lbfgs', alpha=grid.best_params_['alpha'],
                    hidden_layer_sizes=grid.best_params_['hidden_layer_s
model = mlp.fit(x_train, y_train)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_p
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
```

```
### 예측
y_pred = model.predict(x_test)
```

```
### accuracy
model.score(x_test, y_test)
```

```
0.7420178799489144
```

```
### 교차표
pd.crosstab(y_test, y_pred)
```

col_0	**A**	**B**	**C**	**D**
Class				
A	548	53	2	0
B	59	371	89	2
C	9	148	353	109
D	1	13	121	471

데이터 불러오기

↳ 숨겨진 셀 4개

데이터 전처리

↳ 숨겨진 셀 22개

다중 선형 회귀분석

↳ 숨겨진 셀 2개

Train & Test data

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

```
### 입력변수(x)와 출력변수(y)
x = df2[['Year','Fuel_Type','Transmission','Mileage','Engine','Power','S
x_std = StandardScaler().fit_transform(x)
y = df2.Class
```

```
### Train & Test data
x_train, x_test, y_train, y_test = train_test_split(x_std,y,test_size=0.
                                                    random_state=0)
```

서포트 벡터 머신(SVM)

```
from sklearn.svm import SVC
```

선형분류(linear)

```
### SVM
svc = SVC(kernel='linear', C=1, gamma=1)
model = svc.fit(x_train, y_train)
```

```
### 예측
y_pred = model.predict(x_test)
y_pred
```

```
array(['B', 'C', 'C', ..., 'B', 'D', 'D'], dtype=object)
```

```
### accuracy
model.score(x_test, y_test)
```

```
0.7343550446998723
```

```
### 교차표
pd.crosstab(y_test, y_pred)
```

```
col_0    A    B    C    D
Class
    A  498   69    3    0
    B   54  404  131    3
    C    8  114  373  108
    D    1   14  119  450
```

```
from sklearn.model_selection import GridSearchCV
# Set the parameters by cross-validation
tuned_parameters = {'kernel': ['linear'],
                    'C': [0.1, 1, 10, 50, 100, 500 ,1000],
                    'gamma': [0.01, 0.05, 0.1, 0.5, 1, 10]}
```

```
grid = GridSearchCV(SVC(), tuned_parameters)
%time grid.fit(x_train, y_train)
```

```
CPU times: user 9min 44s, sys: 67.8 ms, total: 9min 44s
Wall time: 9min 44s
GridSearchCV(cv=None, error_score=nan,
             estimator=SVC(C=1.0, break_ties=False, cache_size=200,
                           class_weight=None, coef0=0.0,
                           decision_function_shape='ovr', degree=3,
                           gamma='scale', kernel='rbf', max_iter=-1,
                           probability=False, random_state=None, shrinking=
                           tol=0.001, verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid={'C': [0.1, 1, 10, 50, 100, 500, 1000],
                         'gamma': [0.01, 0.05, 0.1, 0.5, 1, 10],
                         'kernel': ['linear']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

```
grid.best_params_
```

```
{'C': 500, 'gamma': 0.01, 'kernel': 'linear'}
```

```
### SVM
# svc = SVC(kernel="linear",C=500, gamma=0.01)
svc = SVC(kernel="linear",C=grid.best_params_['C'], gamma=grid.best_para
model = svc.fit(x_train, y_train)
```

```
### 예측
y_pred = model.predict(x_test)
```

```
### accuracy
model.score(x_test, y_test)
```

```
0.7318007662835249
```

```
### 교차표
pd.crosstab(y_test, y_pred)
```

```
col_0    A    B    C    D
Class
  A    495   73    2    0
  B     54  401  134    3
  C      8  112  377  106
  D      1   14  123  446
```

비선형분류(rbf)

```
### SVM
svc = SVC(kernel="rbf",C=1, gamma=1)
model = svc.fit(x_train, y_train)
```

```
### 예측
y_pred = model.predict(x_test)
```

```
### accuracy
model.score(x_test, y_test)
```

```
0.7692635163899532
```

```
### 교차표
pd.crosstab(y_test, y_pred)
```

```
col_0    A    B    C    D
Class
  A    519   49    2    0
  B     52  402  135    3
  C      8   81  434   80
  D      5   12  115  452
```

```python
from sklearn.model_selection import GridSearchCV
tuned_parameters = {'kernel': ['rbf'],
                    'C': [0.01, 0.1, 1, 10, 50],
                    'gamma': [0.1, 0.5, 1, 10, 50]}
```

```python
grid = GridSearchCV(SVC(), tuned_parameters)
%time grid.fit(x_train, y_train)
```

```
CPU times: user 54.7 s, sys: 18.9 ms, total: 54.8 s
Wall time: 54.8 s
GridSearchCV(cv=None, error_score=nan,
             estimator=SVC(C=1.0, break_ties=False, cache_size=200,
                           class_weight=None, coef0=0.0,
                           decision_function_shape='ovr', degree=3,
                           gamma='scale', kernel='rbf', max_iter=-1,
                           probability=False, random_state=None, shrinking=
                           tol=0.001, verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid={'C': [0.01, 0.1, 1, 10, 50],
                         'gamma': [0.1, 0.5, 1, 10, 50], 'kernel': ['rbf']}
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

```python
grid.best_params_
```

```
{'C': 1, 'gamma': 1, 'kernel': 'rbf'}
```

```python
### SVM
# svc = SVC(kernel="rbf",C=10, gamma=0.1)
svc = SVC(kernel="rbf",C=grid.best_params_['C'], gamma=grid.best_params_
model = svc.fit(x_train, y_train)
```

```python
### 예측
y_pred = model.predict(x_test)
```

```python
### accuracy
model.score(x_test, y_test)
```

```
0.7692635163899532
```

```
### 교차표
pd.crosstab(y_test, y_pred)
```

col_0	A	B	C	D
Class				
A	519	49	2	0
B	52	402	135	3
C	8	81	434	80
D	5	12	115	452

신경망(MLP)

```
from sklearn.neural_network import MLPClassifier
```

```
print(x_std.mean(axis=0), x_std.var(axis=0))
```

```
[ 2.29081059e-14  7.62332976e-17 -3.63015703e-17 -4.17468058e-17
 -1.08904711e-17 -2.00868689e-16  5.25162717e-16] [1. 1. 1. 1. 1. 1. 1.]
```

```
### [0,1] 조정
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.4)
# x_train
x_train_min = x_train.min(axis=0)
x_train_max = (x_train - x_train_min).max(axis=0)
x_train = (x_train - x_train_min) / x_train_max
# x_test
x_test = (x_test - x_train_min) / x_train_max
```

```
### MLP
mlp = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(10,10))
model = mlp.fit(x_train, y_train)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_p
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
```

```
### 예측
y_pred = model.predict(x_test)
```

```
### accuracy
model.score(x_test, y_test)
```

```
0.7352064708386548
```

```
### 교차표
pd.crosstab(y_test, y_pred)
```

col_0	A	B	C	D
Class				
A	530	61	2	1
B	64	375	125	4
C	11	122	368	83
D	1	23	125	454

```
from sklearn.model_selection import GridSearchCV
tuned_parameters = {'solver':['lbfgs'],
                   'alpha':[0.0001,0.001,0.01,0.1,1,10],
                   'hidden_layer_sizes':[(100,),(100,100),(500,),(500,50
```

```
grid = GridSearchCV(MLPClassifier(),tuned_parameters)
%time grid.fit(x_train, y_train)
```

```
grid.best_params_
```

```
{'alpha': 0.001, 'hidden_layer_sizes': (500, 500), 'solver': 'lbfgs'}
```

```
### MLP
mlp = MLPClassifier(solver='lbfgs', alpha=grid.best_params_['alpha'],
                    hidden_layer_sizes=grid.best_params_['hidden_layer_s
model = mlp.fit(x_train, y_train)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_p
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
```

```
### 예측
y_pred = model.predict(x_test)
```

```
### accuracy
model.score(x_test, y_test)
```

```
0.7432950191570882
```

```
### 교차표
pd.crosstab(y_test, y_pred)
```

```
col_0    A    B    C    D
Class
  A    537   52    2    3
  B     63  381  120    4
  C     10  130  368   76
  D      1   20  122  460
```