```
### 목표 : Diamond Dataset을 통해 Diamond 가격을 예측할 수 있는 Regression 모델 만들기
###       ---> 여러 Feature를 사용해서 예측한 Diamond 가격이 Test Dataset으로 선정된 데이터와 최소 Error를 가지도록 Regression 모델 학습 구현

# 행렬 기반 데이터 처리를 위한 numpy, pandas 라이브러리 사용
import numpy as np
import pandas as pd

# Correlation Matrix를 Heatmap으로 표현하기 위한 seaborn 라이브러리 사용
import seaborn as sns

# 그래프 라이브러리 사용
import matplotlib.pyplot as plt
from matplotlib.pylab import rcParams

# 데이터셋 분리 및 모델 Fitting을 위한 sklearn 라이브러리 사용
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import warnings
warnings.filterwarnings('ignore')

# Linear Regression Model, Ridge, Lasso, ElasticNet 모델을 사용하기 위한 sklearn 라이브러리 사용
from sklearn import linear_model
from sklearn.linear_model import Ridge, Lasso, ElasticNetCV
from sklearn.metrics import mean_squared_error  # Error값 연산을 위한 sklearn 함수 사용

from sklearn.preprocessing import LabelEncoder    # 데이터셋에서 문자열로 구성된 Feature를 숫자 형태로 Encoding하기 위한 sklearn 라이브러리
from sklearn.preprocessing import StandardScaler  # 데이터셋 내부 값을 평준화 시키기 위한 Rescaling 함수 사용

# 데이터셋 로딩
diamonds = pd.read_csv('diamonds.csv')
print('[First 5 rows of Diamond Dataset]')
print(diamonds.head())

### 01 Preprocessing (전처리) 단계 ###

# Feature 이름이 없는 데이터 Column 제거
diamonds = diamonds.drop(['Unnamed: 0'], axis=1)
print('[First 5 rows of Diamond Dataset (Without Unnamed Data)]')
print(diamonds.head())

# diamonds.head()를 통해 최초 5개의 데이터셋을 통해 'cut', 'color', 'clarity' Feature가 문자열로 구성되어있다는 것을 확인함
# 문자열로 구성된 Feature를 수학적 연산에 사용될 수 있는 실수 형태의 데이터로 Encoding함
text_cateogrical_features = ['cut', 'color', 'clarity']
le = LabelEncoder()

for i in range(len(text_cateogrical_features)):
    numeric_label = le.fit_transform(diamonds[text_cateogrical_features[i]])    # Feature의 종류마다 실수값으로 Encoding값을 생성
    diamonds[text_cateogrical_features[i]] = numeric_label                      # 해당 Feature를 가진 데이터를 데이터셋 내에서 실수 Encoding값으로 전환함

print('[First 5 rows of Diamond Dataset (Encoded Label)]')
print(diamonds.head())

# 불필요/처리 불가능 데이터 확인
print('[Number of NaN/NULL data]')
print(diamonds.isnull().sum())  # 데이터셋 내에서 NULL 또는 NaN을 확인함. 만약 NULL 또는 NaN이 존재하는 경우 해당 데이터를 데이터셋에서 제외함.

# 데이터셋 Scaling 작업
# 데이터셋 내 각 Feature들은 각자 고유의 단위를 사용함. 이로 인해 각 Feature들의 단위별 변화량이 서로 다름.
# 이러한 단위별 변화량이 다르게 되면 각 Feature들이 학습에 주는 영향이 상대적이게됨.
# 큰 단위를 사용하면서 작은 변화량을 가진 Feature를 작은 단위를 사용하면서 큰 변화량을 가진 Feature보다 모델의 학습에 비교적 적은 영향을 주게되거나 무시될 수 있음.
# 그러므로 모든 Feature가 모델에 대해 동일한 영향을 줄 수 있도록 Rescaling을 수행하여 표준화시킴.
features_X = diamonds[['carat', 'depth', 'table', 'x', 'y', 'z', 'clarity', 'cut', 'color']]    # Feature 데이터 (X) 선정
target_y = diamonds[['price']]  # Target 데이터 (Diamond 가격 / Y) 선정

scaler = StandardScaler()
scaler.fit(features_X)
features_X = scaler.transform(features_X)       # Standard Scaler를 사용하여 Feature 데이터(X)에 대해 단위 표준화 수행

### 02 데이터셋 분석 단계 ###

# Feature로 선정된 데이터들 간에 서로 Multi-Colinearity(다중공정성)이 존재하는 지 파악해야함.
# 만약 Feature간에 Multi-Colinearity가 존재하면, Feature 간에 서로 선형성이 존재하기에 Feature (X) 변화에 따른 Target (Y) 결과 예측이 매우 힘들어짐.
# Multi-Colinearity가 존재하면 Linear Regression이 제대로된 Fitting된 모델을 생성할 수 없음.
# 그러므로 Feature간 Multi-Colinearity를 파악하기 위해 Feature들 간의 Correlation Matrix를 만들어야함.
```

```
# 그리고 Mutli-Colinearity가 높은 요소에 대해서는 Colinearity를 최소화시키는 Regularization(Shrinkage)을 적용해야함.

# Feature 간의 Correlation Matrix를 만들고, Heatmap으로 표현하여 Multi-Colinearity가 존재하는지 파악함.
DIAMONDS_X = pd.DataFrame(features_X,
                          columns=['carat', 'depth', 'table', 'x', 'y', 'z', 'clarity', 'cut', 'color'],
                          index=range(len(features_X)))

correlation_matrix = DIAMONDS_X.corr()  # Rescale된 Diamond Dataset을 pandas DataFrame으로 전환 후 coor() API를 통해 Correlation Matrix 산출함

# Feature (X) 사이의 Correlation Matrix를 Heatmap 그래프로 표현함
plt.figure(figsize=(12, 12))
plt.title('Correlation Matrix Heatmap of Diamond Dataset', pad=15, fontsize='x-large')
sns.heatmap(data=correlation_matrix, square=True, annot=True, cbar=True)
plt.show()
# --> Heatmap을 확인하면 Diamond Dataset은 일부 Feature들 사이에 0.9 단위의 Correlation을 가지는 것을 볼 수 있음.
#     Multi-Colinearity가 존재하기에 그에 상응하는 Regularization 기법을 적용해야함

### 03 데이터셋 준비 단계 ###
# 전체 Diamond Dataset의 Feature (X)와 Target (y)을 특정 비율로 Train Dataset과 Test Dataset으로 중복없이 분리함.
X_train, X_test, y_train, y_test = train_test_split(features_X, target_y, test_size=0.25, random_state=101)     # sklearn train_test_split()을 사용하여 Train과 Test를 75%:25%로 중복없이 분리함
parameters = {'alpha': np.concatenate((np.arange(0.1,2,0.1), np.arange(2, 5, 0.5), np.arange(5, 100, 1)))}      # Regularization에 사용할 Parameter 준비

### 04 Regression 모델 학습 및 성능 확인 단계 ###

# [Linear Regression] ##########################################################################
# Mutli-Colinearity 상황에서 비교를 위해 Linear Regression 모델 준비
linear = linear_model.LinearRegression()        # Linear Regression 모델 객체 생성

linear.fit(X_train, y_train)                    # Train Dataset으로 Linear Regression 모델 학습

y_predict = linear.predict(X_test)              # 학습 후 만들어진 Linear Regression 모델로 Test Dataset의 Feature (X)를 적용해서 Diamond 가격 예측을 수행함

print('----------------------------------')
print('Linear Score : ', linear.score(X_test, y_test))          # 학습된 Linear Regression 모델의 R^2 (결정계수, Coefficient of Determination) 출력하여 모델의 정확도 파악
print('Linear MSE : ', mean_squared_error(y_test, y_predict))   # 학습된 Linear Regression 모델의 Test Dataset의 Feature에 대한 예측값과 Test Dataset의 정답값과 Mean Squared Error를 산출하여 모델의 정확도 파악

# Linear Regression 학습 결과 (Test Dataset의 정답값 vs Test Dataset의 Feature에 대한 예측값) 그래프 전시
plt.subplot(2, 2, 1)
plt.title('Diamond Price Prediction using Linear Regression \n (R^2 Score : {:2f} / MSE : {:2f})'.format(linear.score(X_test, y_test), mean_squared_error(y_test, y_predict)))
plt.grid()
plt.plot(range(30000), range(30000), color='red', linestyle='dashed')
plt.xlim(0, 30000)
plt.ylim(0, 30000)
plt.scatter(y_test, y_predict)

# [Ridge = RSS + L2 Penalty] ###################################################################
# Diamond Dataset의 Feature (X) 사이에 Multi-Colinearity가 존재하기 때문에
# Colinear한 Feature를 L2 Penalty로 약화시킬 수 있는 Regularization 기법을 반영한 Ridge 사용

ridge = linear_model.Ridge()            # Ridge 모델 객체 생성

gridridge = GridSearchCV(ridge, parameters, scoring='r2')       # Ridge의 Tuning Parameter를 전체적으로 탐색하여 최적의 Tuning Parameter를 찾기 위해 Exhaustive Search를 수행하는 GridSearchCV 사용함

gridridge.fit(X_train, y_train)         # 전체 Tuning Parameter 범위에 걸쳐서 Train Dataset으로 Ridge 모델 학습

y_predict = gridridge.predict(X_test)   # 학습 후 만들어진 Ridge 모델로 Test Dataset의 Feature (X)를 적용해서 Diamond 가격 예측을 수행함

print('----------------------------------')
print('Ridge Best Parameters : ', gridridge.best_params_)       # 학습된 Ridge 모델이 최적을 성능을 가질 수 있게하는 Tuning Parameter값 출력
print('Ridge Score : ', gridridge.score(X_test, y_test))        # 학습된 Ridge 모델의 R^2 (결정계수, Coefficient of Determination) 출력하여 모델의 정확도 파악
print('Ridge MSE : ', mean_squared_error(y_test, y_predict))    # 학습된 Ridge 모델의 Test Dataset의 Feature에 대한 예측값과 Test Dataset의 정답값과 Mean Squared Error를 산출하여 모델의 정확도 파악

# Ridge 학습 결과 (Test Dataset의 정답값 vs Test Dataset의 Feature에 대한 예측값) 그래프 전시
plt.subplot(2, 2, 2)
plt.title('Diamond Price Prediction using Ridge \n (R^2 Score : {:2f} / MSE : {:2f})'.format(gridridge.score(X_test, y_test), mean_squared_error(y_test, y_predict)))
plt.grid()
plt.plot(range(30000), range(30000), color='red', linestyle='dashed')
plt.xlim(0, 30000)
plt.ylim(0, 30000)
plt.scatter(y_test, y_predict, color='orange')

# [Lasso = RSS + L1 Penalty] ###################################################################
# Diamond Dataset의 Feature (X) 사이에 Multi-Colinearity가 존재하기 때문에
# Colinear한 Feature를 L1 Penalty로 Zero 수렴시켜서 약화시킬 수 있는 Regularization 기법을 반영한 Lasso 사용
# Colinear한 Feature를 부적합 Feature로 취급하여 0으로 수렴시키게 만듬으로서 유효한 Feature만 살아남게하는 'Feature Selection' 역할을 수행함
```

```
lasso = linear_model.Lasso()    # Lasso 모델 객체 생성

gridlasso = GridSearchCV(lasso, parameters, scoring='r2')       # Lasso의 Tuning Parameter를 전체적으로 탐색하여 최적의 Tuning Parameter를 찾기 위해 Exhaustive Search를 수행하는 GridSearchCV 사용함

gridlasso.fit(X_train, y_train) # 전체 Tuning Parameter 범위에 걸쳐서 Train Dataset으로 Lasso 모델 학습

y_predict = gridlasso.predict(X_test)   # 학습 후 만들어진 Lasso 모델로 Test Dataset의 Feature (X)를 적용해서 Diamond 가격 예측을 수행함

print('----------------------------------')
print('Lasso Best Parameters : ', gridlasso.best_params_)       # 학습된 Lasso 모델이 최적을 성능을 가질 수 있게하는 Tuning Parameter값 출력
print('Lasso Score : ', gridlasso.score(X_test, y_test))        # 학습된 Lasso 모델의 R^2 (결정계수, Coefficient of Determination) 출력하여 모델의 정확도 파악
print('Lasso MSE : ', mean_squared_error(y_test, y_predict))    # 학습된 Lasso 모델의 Test Dataset의 Feature에 대한 예측값과 Test Dataset의 정답값과 Mean Squared Error를 산출하여 모델의 정확도 파악

# Lasso 학습 결과 (Test Dataset의 정답값 vs Test Dataset의 Feature에 대한 예측값) 그래프 전시
plt.subplot(2, 2, 3)
plt.title('Diamond Price Prediction using Lasso \n (R^2 Score : {:.2f} / MSE : {:.2f})'.format(gridlasso.score(X_test, y_test), mean_squared_error(y_test, y_predict)))
plt.grid()
plt.plot(range(30000), range(30000), color='red', linestyle='dashed')
plt.xlim(0, 30000)
plt.ylim(0, 30000)
plt.scatter(y_test, y_predict, color='black')

# [ElasticNet = Ridge + Lasso] ###############################################################################
# ElasticNet은 Ridge와 Lasso의 혼합형 모델임.
# ElasticNet은 Feature (X) 사이에 Multi-Colinearity가 존재하는 상황에서 Lasso의 Feature Zeroing을 Ridge로 밸런스로 맞추게함으로서 좀 더 일반화된 Regression 모델을 생성함.

elasticNet = linear_model.ElasticNetCV(cv=5, random_state=12, l1_ratio=np.arange(0, 1, 0.01), alphas=np.arange(0.1, 100, 0.1))
# Lasso에 대한 Penalty 비율 (L1 Penalty Ratio), Ridge에 대한 Penalty 비율 (L2 Penalty Ratio), Tuning Parameter 범위 전체에 걸쳐서 학습할 수 있는 ElasticNet 모델 객체 생성

elasticNet.fit(X_train, y_train)        # L1 Penalty Ratio, L2 Penalty Ratio, Tuning Parameter 범위에 걸쳐서 Train Dataset으로 ElasticNet 모델 학습

y_predict = elasticNet.predict(X_test)  # 학습 후 만들어진 ElasticNet 모델로 Test Dataset의 Feature (X)를 적용해서 Diamond 가격 예측을 수행함

l1_ratio = elasticNet.l1_ratio_         # 학습된 ElasticNet 모델이 최적을 성능을 가질 수 있게하는 L1 : L2 Penalty Ratio (Ridge, Lasso 사용 비율)을 저장함

print('----------------------------------')
print('ElasticNet')
print('L1 penalty ratio : ', l1_ratio)                                  # 학습된 ElasticNet 모델이 최적을 성능을 가질 수 있게하는 L1:L2 Penalty Ratio (Ridge, Lasso 사용 비율) 출력
print('ElasticNet Score : ', elasticNet.score(X_test, y_test))          # 학습된 ElasticNet 모델의 R^2 (결정계수, Coefficient of Determination) 출력하여 모델의 정확도 파악
print('ElasticNet MSE : ', mean_squared_error(y_test, y_predict))       # 학습된 ElasticNet 모델의 Test Dataset의 Feature에 대한 예측값과 Test Dataset의 정답값과 Mean Squared Error를 산출하여 모델의 정확도 파악

# ElasticNet 학습 결과 (Test Dataset의 정답값 vs Test Dataset의 Feature에 대한 예측값) 그래프 전시
plt.subplot(2, 2, 4)
plt.title('Diamond Price Prediction using ElasticNet \n (L1 Penalty : {:.2f} / L2 Penalty : {:.2f}) \n (R^2 Score : {:.2f} / MSE : {:.2f})'.format(elasticNet.l1_ratio_, 1 - elasticNet.l1_ratio_, elasticNet.score(X_test, y_test),
mean_squared_error(y_test, y_predict)))
plt.grid()
plt.plot(range(30000), range(30000), color='red', linestyle='dashed')
plt.xlim(0, 30000)
plt.ylim(0, 30000)
plt.scatter(y_test, y_predict, color='green')
plt.show()

# Linear Regression, Ridge, Lasso 3개를 비교했을 때, Ridge가 Score값이 제일 높고, Lasso가 MSE값이 제일 낮을 것을 볼 수 있음.
# Multi-Colinearity 상황에서 Ridge 또는 Lasso를 적용해서 더 나은 정확도를 가진 Regression 모델을 산출할 수 있음.
# ElasticNet의 경우 Ridge와 Lasso를 통합해서 사용할 시 L1:L2 Penalty 비율이 0.99:0.01일 때 최적을 성능을 낸다는 것을 볼 수 있음.
# ==> Ridge를 Lasso 보다 더 많은 Penalty를 부여하여 Feature Zeroing 되는 것을 약화시켜서 밸런스를 맞춤.

### 05 학습 결과 Coefficient 구성 그래프 출력 ###
# Ridge, Lasso, ElasticNet의 Tuning Parameter 또는 Penalty Ratio 변화에 따라 Coefficient가 어떻게 변하는 지 확인함

# Tuning Parameter 준비
alphaRidge = parameters['alpha']
alphaLasso = np.arange(0, 20, 1)
alphaElasticNet = parameters['alpha']

# Coefficient (Feature X의 weight) 준비
coefficient_Ridge = []
coefficient_Lasso = []
coefficient_ElasticNet = []

# Ridge 모델에서 Tuning Parameter의 변화에 따른 Coefficient 정보 저장
for alpha in alphaRidge:
    ridge = linear_model.Ridge(alpha=alpha)
    ridge.fit(X_train, y_train)
    coefficient_Ridge.append(ridge.coef_[0])
```

```
# Lasso 모델에서 Tuning Parameter의 변화에 따른 Coefficient 정보 저장
for alpha in alphaLasso:
    lasso = linear_model.Lasso(alpha=alpha)
    lasso.fit(X_train, y_train)
    coefficient_Lasso.append(lasso.coef_)

# ElasticNet 모델에서 최적의 L1:L2 Penalty Ratio 사용시 Tuning Parameter의 변화에 따른 Coefficient 정보 저장
for alpha in alphaElasticNet:
    elasticNet = linear_model.ElasticNetCV(cv=5, random_state=12, l1_ratio=L1_ratio, alphas=[alpha])
    elasticNet.fit(X_train, y_train)
    coefficient_ElasticNet.append(elasticNet.coef_)

plt.cla
plt.subplot(1, 3, 1)
plt.plot(alphaRidge, coefficient_Ridge) # Ridge의 Coefficient 변화 그래프 출력
plt.title('Ridge Coefficients')
plt.xlabel('alpha')
plt.ylabel('coefficients')

plt.subplot(1, 3, 2)
plt.plot(alphaLasso, coefficient_Lasso) # Lasso의 Coefficient 변화 그래프 출력
plt.title('Lasso Coefficients')
plt.xlabel('alpha')
plt.ylabel('coefficients')

plt.subplot(1, 3, 3)
plt.plot(alphaElasticNet, coefficient_ElasticNet)       # ElasticNet의 Coefficient 변화 그래프 출력
plt.title('ElasticNet Coefficients \n (L1 Penalty : {:2f} / L2 Penalty : {:2f})'.format(L1_ratio, 1-L1_ratio))
plt.xlabel('alpha')
plt.ylabel('coefficients')
plt.show()

# ElasticNet에서 Ridge와 Lasso를 통합하여 사용하면서 Feature에 대한 Weight/Coefficient가 0으로 수렴하는 것을 좀 더 부드럽게 만듬.
```