迴歸分析 期末報告

世界幸福解釋性數據分析

World Happiness Exploratory Data Analysis

應數三	王常騰
應數四	溫宏岳
應數四	張俊翔
應數四	陳曦
應數四	胡家宏

Contents

PART 1

關於「世界幸福報告」

一、甚麼是「世界幸福報告」？

世界幸福報告（World Happiness Report）為聯合國為衡量國家幸福的可持續發展方案，於網路出版的國際調查報告。

聯合國標誌

世界幸福報告的標誌

二、計算標準

《世界幸福報告》的排名是使用蓋洛普世界民意調查的數據，這是由民意調查中提出的生活評估問題的回答，它要求受訪者想出一個階梯，對他們來說最好的是 10，最壞的是 0，這被稱為 Cantril 階梯。
評分的細項共有10點 ：

1. 人均GDP
2. 社會支援
3. 人生抉擇的自由
4. 預期的健康壽命
5. 慷慨
6. 正面影響
7. 負面影響
8. 家庭收入報告的基尼係數
9. 世界銀行提供的基尼指數
10. 對於政府機關的信任程度

二、計算標準(續)

這十點細項將表現出以下六大因素，這六大因素中的每一個皆有助於評估每個國家。

GDP 水平

社會支持

預期壽命

自由

慷慨

腐敗

三、殘差

除了上面這些指標外，因為統計方法的緣故，分數裡還加了一項「 Dystopia + 殘值 」這個數字。

Dystopia ：「作為標準的虛擬國家」，擁有上面六大指標的最低分數。

「殘值」：以迴歸分析算出來的值跟實際值的差異。

簡單來說，可以視為這個統計模型中「無法解釋的部分」。

四、樣本來源

每年每個國家的典型樣本為 1,000 人，聯合國使用最近三年的回復來提供最新的生活評估，所以如果一個典型的國家每年進行調查，樣本量將是 3,000，其樣本數夠多，可以減少隨機抽樣誤差。

但是目前還有許多國家沒有進行年度調查。

五、數據“浪潮”

蓋洛普將每個日曆年收集的調查作為當年調查浪潮的一部分。在絕大多數情況下，浪潮對應於日曆年，但也有一些例外；例如 ： 一些在 2022 年初完成的調查被認為是 2021 年浪潮的一部分。

並非每個國家每年都接受調查。因此，調查波的規模每年也不同。

PART 2

題目以及過程

Step 1. 資料匯入與預處理

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
from collections import Counter
from pandas import DataFrame

import sklearn
import statsmodels.api as sm
from sklearn.linear_model import LinearRegression
from statsmodels.sandbox.regression.predstd import wls_prediction_std
from sklearn.model_selection import train_test_split
from patsy import dmatrices
from statsmodels.stats.outliers_influence import variance_inflation_factor

import scipy.stats as stats
from sklearn.metrics import mean_squared_error

import seaborn as sns
import matplotlib.mlab as mlab

sns.set_style("whitegrid")
sns.set_context("paper")
df = pd.read_csv("2021a.csv")
```

```
df.info()
print('\n')
print('len = ' ,len(df))
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 149 entries, 0 to 148
Data columns (total 20 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   CountryName                       149 non-null    object
 1   RegionalIndicator                 149 non-null    object
 2   LadderScore                       149 non-null    float64
 3   StandardErrorOfLadderScore        149 non-null    float64
 4   upperwhisker                      149 non-null    float64
 5   lowerwhisker                      149 non-null    float64
 6   LoggedGDPPerCapita                149 non-null    float64
 7   SocialSupport                     149 non-null    float64
 8   HealthyLifeExpectancy             149 non-null    float64
 9   FreedomToMakeLifeChoices          149 non-null    float64
 10  Generosity                        149 non-null    float64
 11  PerceptionsOfCorruption           149 non-null    float64
 12  LadderScoreInDystopia             149 non-null    float64
 13  ExplainedbyLogGDPpercapita        149 non-null    float64
 14  ExplainedbySocialsupport          149 non-null    float64
 15  ExplainedbyHealthylifeexpectancy  149 non-null    float64
 16  ExplainedbyFreedomtomakelifechoices  149 non-null  float64
 17  ExplainedbyGenerosity             149 non-null    float64
 18  ExplainedbyPerceptionsofcorruption  149 non-null  float64
 19  Dystopiaresidual                  149 non-null    float64
dtypes: float64(18), object(2)
memory usage: 23.4+ KB


len =  149
```

a. 檢查資料是否有缺失值

```
df.isnull().sum(axis=0)
```

```
CountryName                         0
RegionalIndicator                   0
LadderScore                         0
StandardErrorOfLadderScore          0
upperwhisker                        0
lowerwhisker                        0
LoggedGDPPerCapita                  0
SocialSupport                       0
HealthyLifeExpectancy               0
FreedomToMakeLifeChoices            0
Generosity                          0
PerceptionsOfCorruption             0
LadderScoreInDystopia               0
ExplainedbyLogGDPpercapita          0
ExplainedbySocialsupport            0
ExplainedbyHealthylifeexpectancy    0
ExplainedbyFreedomtomakelifechoices 0
ExplainedbyGenerosity               0
ExplainedbyPerceptionsofcorruption  0
Dystopiaresidual                    0
dtype: int64
```

b. One hot encoding ： 可以處理數字但不能直接處理字串值， 需先將字串對應成數值。

```
data_one_hot = pd.get_dummies(df)
data_one_hot.head()
```

	LadderScore	StandardErrorOfLadderScore	upperwhisker	lowerwhisker	LoggedGDPPerCapita	SocialSupport	HealthyLifeExpectancy	FreedomToMakeLifeChoic
0	7.842	0.032	7.904	7.780	10.775	0.954	72.0	0.9
1	7.620	0.035	7.687	7.552	10.933	0.954	72.7	0.9
2	7.571	0.036	7.643	7.500	11.117	0.942	74.4	0.9
3	7.554	0.059	7.670	7.438	10.878	0.983	73.0	0.9
4	7.464	0.027	7.518	7.410	10.932	0.942	72.4	0.9

5 rows × 177 columns

c. Drop無關值 ：

```
df = df.drop(['CountryName','RegionalIndicator','upperwhisker','lowerwhisker','ExplainedbyLogGDPpercapita',
              'ExplainedbySocialsupport','ExplainedbyHealthylifeexpectancy','ExplainedbyFreedomtomakelifechoices',
              'ExplainedbyGenerosity','ExplainedbyPerceptionsofcorruption','Dystopiaresidual'], axis = 1)
df.head()
```

	LadderScore	StandardErrorOfLadderScore	LoggedGDPPerCapita	SocialSupport	HealthyLifeExpectancy	FreedomToMakeLifeChoices	Generosity	PerceptionsC
0	7.842	0.032	10.775	0.954	72.0	0.949	-0.098	
1	7.620	0.035	10.933	0.954	72.7	0.946	0.030	
2	7.571	0.036	11.117	0.942	74.4	0.919	0.025	
3	7.554	0.059	10.878	0.983	73.0	0.955	0.160	
4	7.464	0.027	10.932	0.942	72.4	0.913	0.175	

Step 2. 複迴歸模型

score會透過R^2來判定我們模型的精準程度；如果訓練集的分數很高，但測試集的分數卻很低，那就是過度擬和

a. 從理論推導

```
yhat = model.predict(X)
```

```
SS_Residual = sum((y-yhat)**2)
SS_Total = sum((y-np.mean(y))**2)
r_squared = 1 - (float(SS_Residual))/SS_Total
adjusted_r_squared = 1 - (1-r_squared)*(len(y)-1)/(len(y)-X.shape[1]-1)
print(r_squared, adjusted_r_squared)
```

```
0.7558471374226854 0.7437260733231024
```

b. 使用 sklearn linear_model 計算

雖然無法直接從文檔中找到任何計算 adjusted R^2方式的函數。

```
print(model.score(X, y), 1 - (1-model.score(X, y))*(len(y)-1)/(len(y)-X.shape[1]-1))
```

```
0.7558471374226855 0.7437260733231026
```

c. 使用 statsmodels 計算，手動添加截距

```
# 通過手動添加截距後使用 statsmodels 計算
import statsmodels.api as sm
X1 = sm.add_constant(X)
result = sm.OLS(y, X1).fit()
#print dir(result)
print(result.rsquared, result.rsquared_adj)
```

```
0.7558471374226855 0.7455308192856158
```

d. 使用 statsmodels 計算，使用公式的另一種方法

```
# 使用 statsmodels的另一種公式計算
import statsmodels.formula.api as sm
#result = sm.ols(formula="Ladder score ~ NumberofEmployees + ValueofContract", data=df).fit()
#print result.summary()
print(result.rsquared, result.rsquared_adj)
```

```
0.7558471374226855 0.7455308192856158
```

```
print('   R^2                    adj R^2')
print('a:', r_squared1, adjusted_r_squared1)
print('b:', result2_rsquared, result2_rsquared_adj)
print('c:', result3.rsquared, result3.rsquared_adj)
print('d:', result4.rsquared, result4.rsquared_adj)
```

```
   R^2                    adj R^2
a: 0.7558471374226854 0.7437260733231024
b: 0.7558471374226855 0.7437260733231026
c: 0.7558471374226855 0.7455308192856158
d: 0.7558471374226855 0.7455308192856158
```

=> 由上述4種方式得知調整後的R^2最好的是c、d的模型

Step 3. OLS 線性關係判斷

線性回歸模型，顧名思義，首先要保證自變量與因變量之間存在線性關係。關於線性關係的判斷，我們可以通過圖形或Pearson相關係數來識別。

一般情況下的評判標準 ：

當相關係數 低於0.4 ，則表明變量之間存在弱相關關係；

當相關係數在 0.4~0.6之間 ，則說明變量之間存在中度相關關係；

當相關係數在 0.6以上時 ，則反映變量之間存在強相關關係。

a. 創建線性迴歸最小平方法模型

```
import statsmodels.formula.api as smf
import statsmodels.api as sm

model = sm.OLS(y,X)

results = model.fit()

results.summary()
```

OLS Regression Results

Dep. Variable:	LadderScore	**R-squared:**	0.756
Model:	OLS	**Adj. R-squared:**	0.746
Method:	Least Squares	**F-statistic:**	73.27
Date:	Mon, 13 Jun 2022	**Prob (F-statistic):**	5.06e-41
Time:	17:09:58	**Log-Likelihood:**	-116.50
No. Observations:	149	**AIC:**	247.0
Df Residuals:	142	**BIC:**	268.0
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
LoggedGDPPerCapita	0.2795	0.087	3.219	0.002	0.108	0.451
SocialSupport	2.4762	0.668	3.706	0.000	1.155	3.797
HealthyLifeExpectancy	0.0303	0.013	2.274	0.024	0.004	0.057
FreedomToMakeLifeChoices	2.0105	0.495	4.063	0.000	1.032	2.989
Generosity	0.3644	0.321	1.134	0.259	-0.271	0.999
PerceptionsOfCorruption	-0.6051	0.291	-2.083	0.039	-1.179	-0.031
LadderScoreInDystopia	-0.9207	0.259	-3.548	0.001	-1.434	-0.408

Omnibus:	12.908	**Durbin-Watson:**	1.614
Prob(Omnibus):	0.002	**Jarque-Bera (JB):**	13.688
Skew:	-0.667	**Prob(JB):**	0.00107
Kurtosis:	3.650	**Cond. No.**	1.05e+03

[1] 標準誤差假設正確指定了誤差的共變異數矩陣。
[2] 條件數很大，1.05e+03。 這可能表明存在強多重共線性或其他數值問題。

b. LadderScore與自變量之間的相關係數

```
df.corrwith(df['LadderScore'])
```

```
LadderScore                    1.000000
StandardErrorOfLadderScore    -0.470787
LoggedGDPPerCapita             0.789760
SocialSupport                  0.756888
HealthyLifeExpectancy          0.768099
FreedomToMakeLifeChoices       0.607753
Generosity                    -0.017799
PerceptionsOfCorruption       -0.421140
LadderScoreInDystopia               NaN
dtype: float64
```

經過對比發現，LadderScore與Generosity之間的為弱相關關係，可以不考慮將該變量納入模型。當然，變量之間不存在線性關係並不代表不存在任何關係，可能是二次函數關係、對數關係等，所以一般還需要進行檢驗和變量轉換。

相關係數較大的是Logged GDP per capita、Healthy life expectancy、Social support、Freedom to make life choices。

b. LadderScore與自變量之間的相關係數

```
df.corrwith(df['LadderScore'])
```

```
LadderScore                    1.000000
StandardErrorOfLadderScore    -0.470787
LoggedGDPPerCapita             0.789760
SocialSupport                  0.756888
HealthyLifeExpectancy          0.768099
FreedomToMakeLifeChoices       0.607753
Generosity                    -0.017799
PerceptionsOfCorruption       -0.421140
LadderScoreInDystopia               NaN
dtype: float64
```

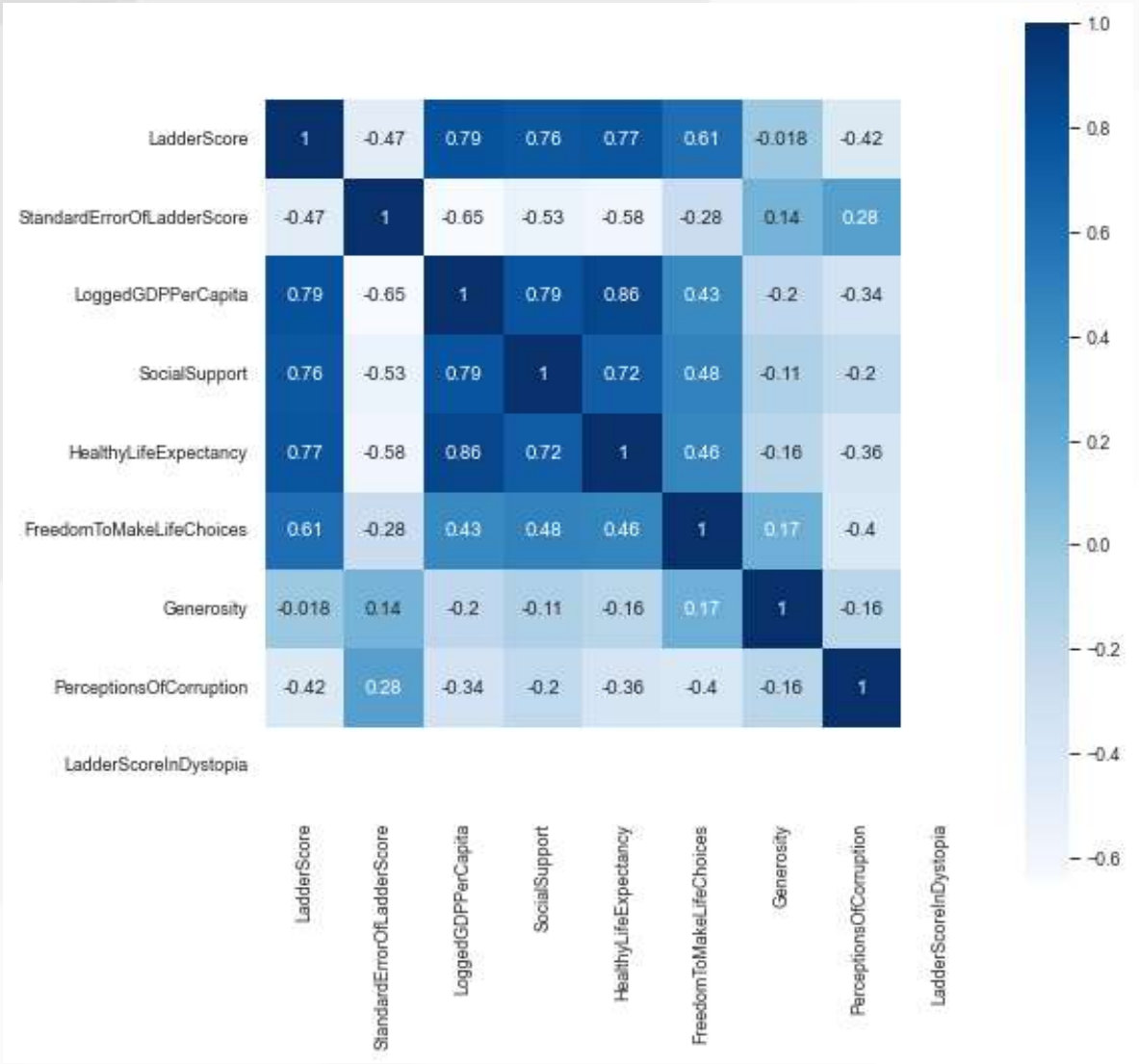
經過對比發現，LadderScore與Generosity之間的為弱相關關係，可以不考慮將該變量納入模型。當然，變量之間不存在線性關係並不代表不存在任何關係，可能是二次函數關係、對數關係等，所以一般還需要進行檢驗和變量轉換。

相關係數較大的是Logged GDP per capita、Healthy life expectancy、Social support、Freedom to make life choices。

Step 4. 多重共線性判斷

a. 相關性(使用heatmap)

```
plt.figure(figsize=(8,8))
sns.heatmap(df.corr(), annot=True, vmax=1, square=True, cmap='Blues')
plt.show()
```



b. 擬合模型-模型顯著性和參數顯著性判斷

```
Train,Test = train_test_split(df, train_size = 0.8, random_state=0)
fit = smf.ols('LadderScore~ LoggedGDPPerCapita + SocialSupport + \
            HealthyLifeExpectancy + FreedomToMakeLifeChoices + Generosity + \
            PerceptionsOfCorruption + LadderScoreInDystopia'
            , data = Train).fit()

fit.summary()
```

```
d = [ ["原始資料", fit.rsquared, fit.rsquared_adj]]

print(tabulate(d, headers=["R^2", "adj R^2"]))
```

```
               R^2    adj R^2
--------  --------  ---------
原始資料  0.752083   0.738802
```

OLS Regression Results

Dep. Variable:	LadderScore	**R-squared:**	0.752
Model:	OLS	**Adj. R-squared:**	0.739
Method:	Least Squares	**F-statistic:**	56.63
Date:	Mon, 13 Jun 2022	**Prob (F-statistic):**	1.14e-31
Time:	17:09:59	**Log-Likelihood:**	-97.408
No. Observations:	119	**AIC:**	208.8
Df Residuals:	112	**BIC:**	228.3
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-0.3658	0.108	-3.389	0.001	-0.580	-0.152
LoggedGDPPerCapita	0.2185	0.111	1.968	0.052	-0.002	0.438
SocialSupport	2.8474	0.799	3.565	0.001	1.265	4.430
HealthyLifeExpectancy	0.0424	0.016	2.689	0.008	0.011	0.074
FreedomToMakeLifeChoices	1.6991	0.568	2.990	0.003	0.573	2.825
Generosity	0.3891	0.358	1.088	0.279	-0.320	1.098
PerceptionsOfCorruption	-0.5927	0.333	-1.779	0.078	-1.253	0.067
LadderScoreInDystopia	-0.8889	0.262	-3.389	0.001	-1.409	-0.369

Omnibus:	7.976	**Durbin-Watson:**	2.318
Prob(Omnibus):	0.019	**Jarque-Bera (JB):**	7.660
Skew:	-0.592	**Prob(JB):**	0.0217
Kurtosis:	3.375	**Cond. No.**	7.23e+17

=> 通過上面結果我們清楚看到：
7個回歸係數的t統計量p值除了Generosity、PerceptionsOfCorruption、LoggedGDPPerCapita其餘的都<0.05，說明剩下的迴歸係數較顯著，因此需要Drop掉P值最大的Generosity重新建模

c. 第一次重新建模

```
fit2 = smf.ols('LadderScore~ LoggedGDPPerCapita \
                + SocialSupport + HealthyLifeExpectancy + \
                FreedomToMakeLifeChoices + PerceptionsOfCorruption + \
                LadderScoreInDystopia'
                ,data = Train.drop('Generosity', axis = 1)).fit()

fit2.summary()
```

OLS Regression Results

Dep. Variable:	LadderScore	**R-squared:**	0.749
Model:	OLS	**Adj. R-squared:**	0.738
Method:	Least Squares	**F-statistic:**	67.61
Date:	Mon, 13 Jun 2022	**Prob (F-statistic):**	2.35e-32
Time:	17:09:59	**Log-Likelihood:**	-98.033
No. Observations:	119	**AIC:**	208.1
Df Residuals:	113	**BIC:**	224.7
Df Model:	5		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-0.3448	0.106	-3.244	0.002	-0.555	-0.134
LoggedGDPPerCapita	0.1996	0.110	1.818	0.072	-0.018	0.417
SocialSupport	2.9098	0.797	3.649	0.000	1.330	4.490
HealthyLifeExpectancy	0.0413	0.016	2.623	0.010	0.010	0.072
FreedomToMakeLifeChoices	1.8121	0.559	3.241	0.002	0.704	2.920
PerceptionsOfCorruption	-0.6498	0.329	-1.973	0.051	-1.302	0.003
LadderScoreInDystopia	-0.8379	0.258	-3.244	0.002	-1.350	-0.326

Omnibus:	8.416	**Durbin-Watson:**	2.282
Prob(Omnibus):	0.015	**Jarque-Bera (JB):**	8.142
Skew:	-0.606	**Prob(JB):**	0.0171
Kurtosis:	3.419	**Cond. No.**	7.22e+17

```
d = [ ["第一次建模", fit2.rsquared, fit2.rsquared_adj]]

print(tabulate(d, headers=["R^2", "adj R^2"]))
```

```
                R^2    adj R^2
----------  --------  ---------
第一次建模  0.749465   0.738379
```

=> 通過上面結果我們清楚看到：
剩下6個回歸係數的t統計量p值除了LoggedGDPPerCapita、PerceptionsOfCorruption、其餘的都<0.05，說明剩下的迴歸係數較顯著，因此需要Drop掉P值最大的LoggedGDPPerCapita繼續重新建模。

d. 第二次重新建模

```
fit3 = smf.ols('LadderScore~ SocialSupport + HealthyLifeExpectancy + \
                FreedomToMakeLifeChoices + PerceptionsOfCorruption + \
                LadderScoreInDystopia'
                ,data = Train.drop('LoggedGDPPerCapita', axis = 1)).fit()

fit3.summary()
```

```
d = [ ["第二次建模", fit3.rsquared, fit3.rsquared_adj]]

print(tabulate(d, headers=["R^2", "adj R^2"]))
```

```
                R^2    adj R^2
----------  --------  ---------
第二次建模  0.742134   0.733086
```

OLS Regression Results

Dep. Variable:	LadderScore	**R-squared:**	0.742
Model:	OLS	**Adj. R-squared:**	0.733
Method:	Least Squares	**F-statistic:**	82.02
Date:	Mon, 13 Jun 2022	**Prob (F-statistic):**	1.22e-32
Time:	17:09:59	**Log-Likelihood:**	-99.749
No. Observations:	119	**AIC:**	209.5
Df Residuals:	114	**BIC:**	223.4
Df Model:	4		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-0.3122	0.106	-2.951	0.004	-0.522	-0.103
SocialSupport	3.7287	0.665	5.610	0.000	2.412	5.045
HealthyLifeExpectancy	0.0607	0.012	5.195	0.000	0.038	0.084
FreedomToMakeLifeChoices	1.5791	0.550	2.872	0.005	0.490	2.668
PerceptionsOfCorruption	-0.7708	0.326	-2.366	0.020	-1.416	-0.125
LadderScoreInDystopia	-0.7586	0.257	-2.951	0.004	-1.268	-0.249

Omnibus:	6.199	**Durbin-Watson:**	2.225
Prob(Omnibus):	0.045	**Jarque-Bera (JB):**	5.773
Skew:	-0.524	**Prob(JB):**	0.0558
Kurtosis:	3.257	**Cond. No.**	7.15e+17

=> 通過模型反饋的結果我們可知，模型是通過顯著性檢驗的，即剩下6個迴歸係數的t統計量p值遠遠小於0.05這個閾值的，說明需要拒絕原假設(即認為模型的所有回歸係數都不全為0)。

=>模型的顯著性通過檢驗的話，並不代表每一個自變量都對因變量是重要的，所以還需要進行偏回歸係數的顯著性檢驗。通過上圖的檢驗結果顯示，除變量newspaper對應的P值超過0.05，其餘變量都低於這個閾值，說明newspaper這個廣告渠道並沒有影響到銷售量的變動，故需要將其從模型中剔除。

e. RMSE比較

```
pred = fit.predict(exog = Test)
pred2 = fit2.predict(exog = Test.drop('Generosity', axis = 1))
pred3 = fit3.predict(exog = Test.drop('LoggedGDPPerCapita', axis = 1))

RMSE = np.sqrt(mean_squared_error(Test.LadderScore, pred))
RMSE2 = np.sqrt(mean_squared_error(Test.LadderScore, pred2))
RMSE3 = np.sqrt(mean_squared_error(Test.LadderScore, pred3))

print('第一個: RMSE = %.4f\n'%RMSE)
print('第二個: RMSE = %.4f\n'%RMSE2)
print('第三個: RMSE = %.4f\n'%RMSE3)
```

	RMSE
原始資料	0.460291
第一次建模	0.460752
第二次建模	0.517379

=> 對於連續委量預測效果的好壞,我們可以信助於RMSE(均方根誤差,即真實值與預測值的均方根)來衡量,如果這個值越小就說明模型越優秀,即預測出來 的值會越接近於真實值很明顯

=> 我們發現模型1的RMSE相比於模型2、3會小一些,模型會更符合實際

f. 誤差值

```
df["pred"]=pd.Series(fit.predict())
abs_=(df['pred']-df['LadderScore']).abs()

mae_=abs_.mean()
rmse_=((abs_**2).mean())**0.5
mape_=(abs_/df['LadderScore']).mean()
mape_
```

	誤差值
原始資料	0.18341
第一次建模	0.182697
第二次建模	0.18341

g. VIF

如果自變量X與其他自變量共線性強，那麼回歸方程的R2就會較高，導致VIF也高。一般，有自變量VIF值大於10，則說明存在嚴重多重共線性，可以選擇刪除該變量或者用其他類似但VIF低的變量代替。
可以看到AT的方差膨脹因子大於10，可以刪除該變量。

```
y, X = dmatrices('LadderScore~ LoggedGDPPerCapita + SocialSupport + \
                  HealthyLifeExpectancy + FreedomToMakeLifeChoices + Generosity + \
                  PerceptionsOfCorruption + LadderScoreInDystopia'
                  , data = df, return_type='dataframe')

vif = pd.DataFrame()
vif["VIF Factor"]=[variance_inflation_factor(X.values,i) \
                   for i in range(X.shape[1])]

vif["feature"]=X.columns
vif
```

	VIF Factor	feature
0	0.000000	Intercept
1	5.104890	LoggedGDPPerCapita
2	2.972200	SocialSupport
3	4.099348	HealthyLifeExpectancy
4	1.585807	FreedomToMakeLifeChoices
5	1.180982	Generosity
6	1.367122	PerceptionsOfCorruption
7	0.000000	LadderScoreInDystopia

=> 結果顯示,所有自變量的VIF均低於10,說明自變量之間並不存在多重共線性的隱患。

多重共線性的處理方法 ：
多重共線性對於線性回歸是種災難，並且我們不可能完全消除，而只能利用一些方法來減輕它的影響。
對於多重共線性的處理方式，有以下幾種思路：
1)提前篩選變量：可以利用相關檢驗來或變量聚類的方法。
注意：決策樹和隨機森林也可以作為提前篩選變量的方法，但是它們對於多重共線性幫助不大，因為如果按照特徵重要性排序，共線性的變量很可能都排在前面。
2)子集選擇：包括逐步回歸和最優子集法。因為該方法是貪婪算法，理論上大部分情況有效，實際中需要結合第一種方法。
3)收縮方法：正則化方法，包括嶺回歸和LASSO回歸。LASSO回歸可以實現篩選變量的功能。
4)維數縮減：包括主成分回歸(PCR)和偏最小平方法迴歸(PLS)方法。

Step 5. 強影響點診斷

```
#離群點檢驗
outliers = fit.get_influence()
#高槓桿值點(帽子矩)
leverage = outliers.hat_matrix_diag
#dffts值
dffits = outliers.dffits[0]
#學生化殘差
resid_stu = outliers.resid_studentized_external
#cook距離
cook = outliers.cooks_distance[0]
#covratio值
covratio = outliers.cov_ratio
#將上面的幾種異常值檢驗統計量與原始數據集合
contat1 = pd.concat([pd.Series(leverage, name = 'leverage'),pd.Series(dffits, name = 'dffits'),
pd.Series(resid_stu,name ='resid_stu'),pd.Series(cook,name = 'cook'),
pd.Series(covratio, name = 'covratio'),], axis = 1)
df_outliers = pd.concat([df,contat1], axis = 1)
df_outliers.head()
```

a. 計算異常值數值的比例

```
#計算異常值數量的比例
outliers_ratio = sum(np.where((np.abs(df_outliers.resid_stu)>2),1,0))/df_outliers.shape[0]
print('異常值數量的比例 = ',outliers_ratio)
```

```
異常值數量的比例 =  0.04697986577181208
```

b. 刪除異常值

```
#刪除異常值
df_outliers = df_outliers.loc[np.abs(df_outliers.resid_stu)<=2,]
```

c. 第三次重新建模

```
fit4 = sm.formula.ols('LadderScore~ LoggedGDPPerCapita + SocialSupport + \
                    HealthyLifeExpectancy + FreedomToMakeLifeChoices + \
                    Generosity + PerceptionsOfCorruption +\
                    LadderScoreInDystopia'\
                    ,data = df_outliers).fit()

fit4.summary()
```

```
d = [ ["第三次建模", fit4.rsquared, fit4.rsquared_adj]]

print(tabulate(d, headers=["R^2", "adj R^2"]))
```

```
                R^2    adj R^2
----------  --------  ---------
第三次建模  0.732945   0.717684
```

Dep. Variable:	LadderScore	**R-squared:**	0.733
Model:	OLS	**Adj. R-squared:**	0.718
Method:	Least Squares	**F-statistic:**	48.03
Date:	Mon, 13 Jun 2022	**Prob (F-statistic):**	6.26e-28
Time:	17:09:59	**Log-Likelihood:**	-61.656
No. Observations:	112	**AIC:**	137.3
Df Residuals:	105	**BIC:**	156.3
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-0.0614	0.104	-0.592	0.555	-0.267	0.144
LoggedGDPPerCapita	0.2540	0.084	3.015	0.003	0.087	0.421
SocialSupport	1.9191	0.667	2.876	0.005	0.596	3.242
HealthyLifeExpectancy	0.0175	0.013	1.322	0.189	-0.009	0.044
FreedomToMakeLifeChoices	2.1116	0.500	4.220	0.000	1.119	3.104
Generosity	0.2142	0.308	0.696	0.488	-0.396	0.825
PerceptionsOfCorruption	-0.9059	0.265	-3.424	0.001	-1.431	-0.381
LadderScoreInDystopia	-0.1493	0.252	-0.592	0.555	-0.649	0.351

Omnibus:	3.786	**Durbin-Watson:**	1.632
Prob(Omnibus):	0.151	**Jarque-Bera (JB):**	3.815
Skew:	-0.432	**Prob(JB):**	0.148
Kurtosis:	2.736	**Cond. No.**	1.85e+17

d. 計算誤差

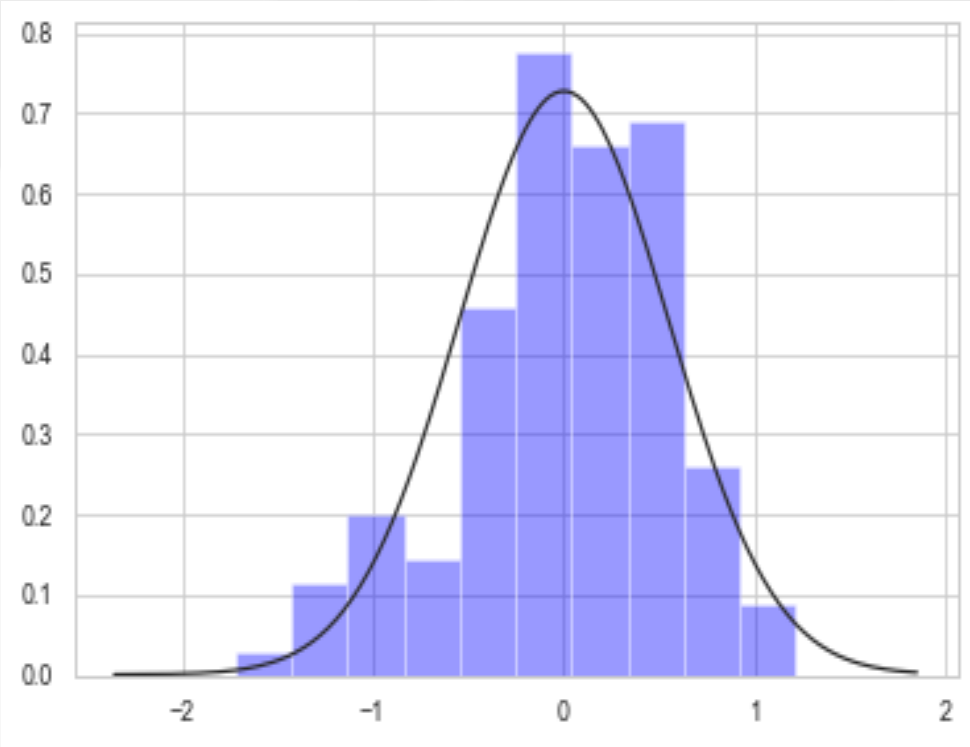
```
#總絕對深差
df_outliers["pred2"]=pd.Series(fit2.predict())
abs_= (df_outliers['pred2'] -df_outliers['LadderScore']).abs()
#mae,平均絕對溪差
mae_ = abs_.mean()
#rmse,均方根誤差
rmse_ = ((abs_**2).mean())**0.5
#mape ,平均絕對百分比課差
mape_= (abs_/df_outliers['LadderScore']).mean()
mape_
```

```
0.05540979663400268
```

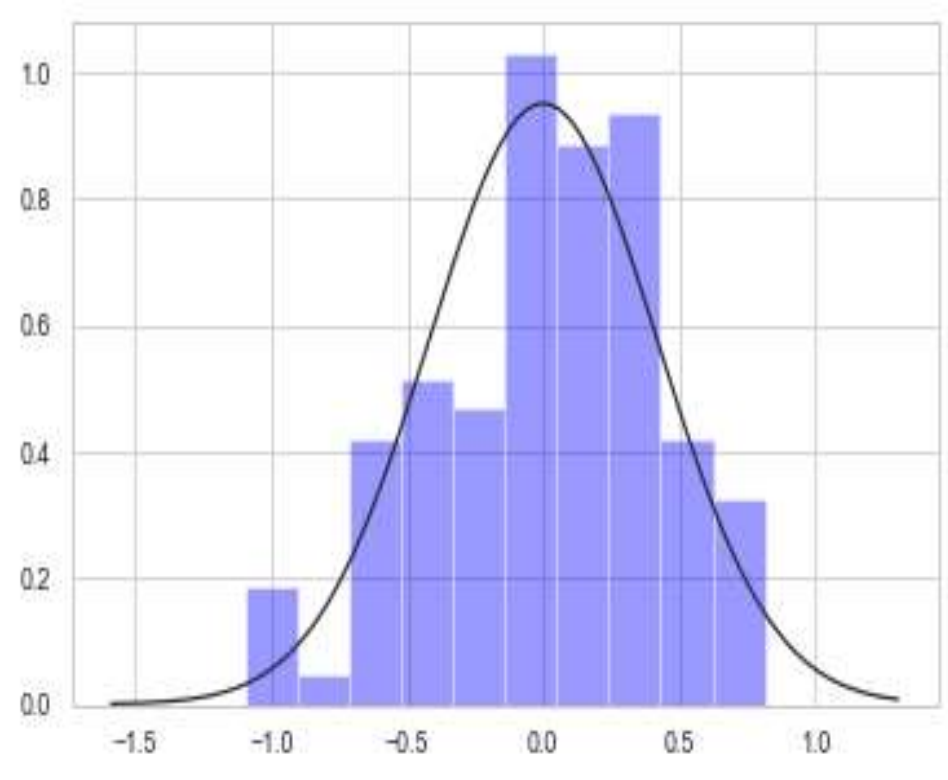
=> 通過對比f和t2,將異常值刪掉後重新建模的話效果會更理想一點具體表現為:信息準則(AIC和BIC)均變小,同時RMSE(識差均方根)也有降低

Step 6. 殘差診斷

a. KS檢驗



model1



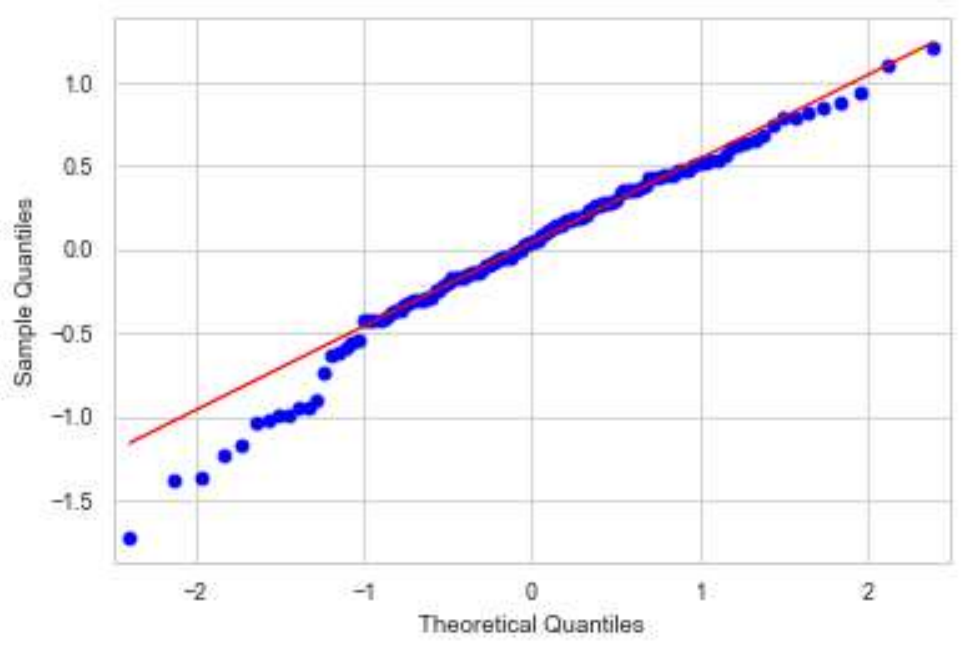
model2

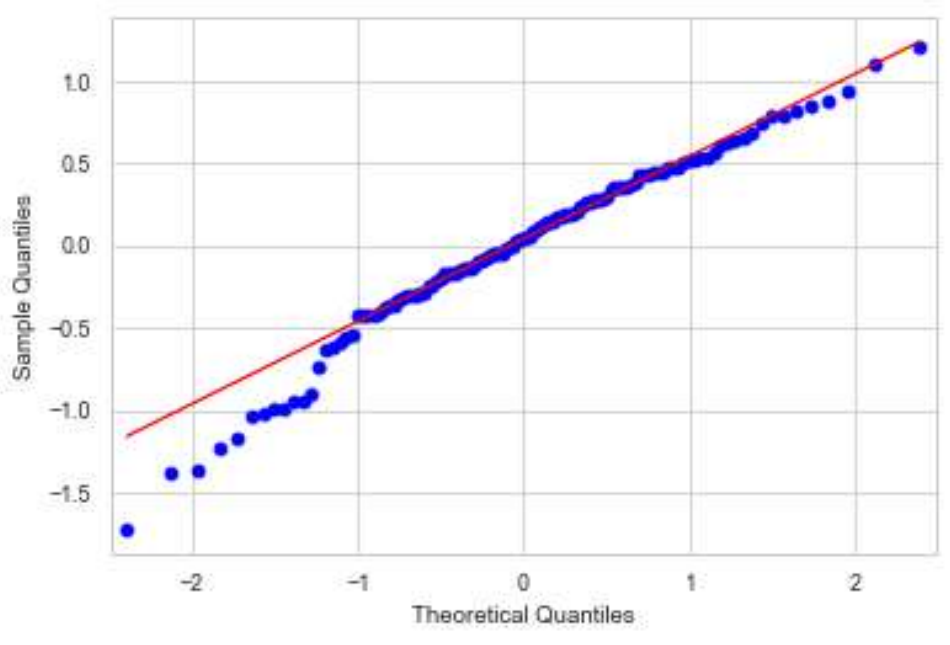
b. QQ圖

model1

```
pq = sm.ProbPlot(residual)

pq.qqplot(line='q')
```

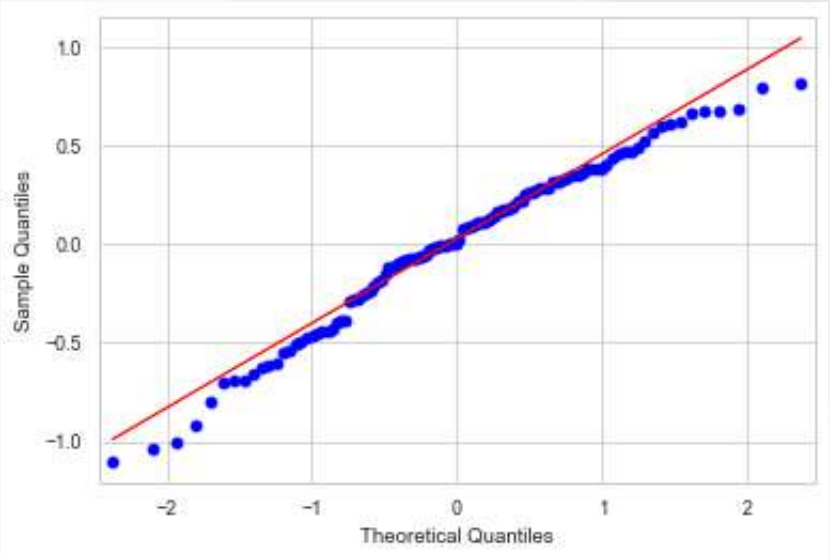


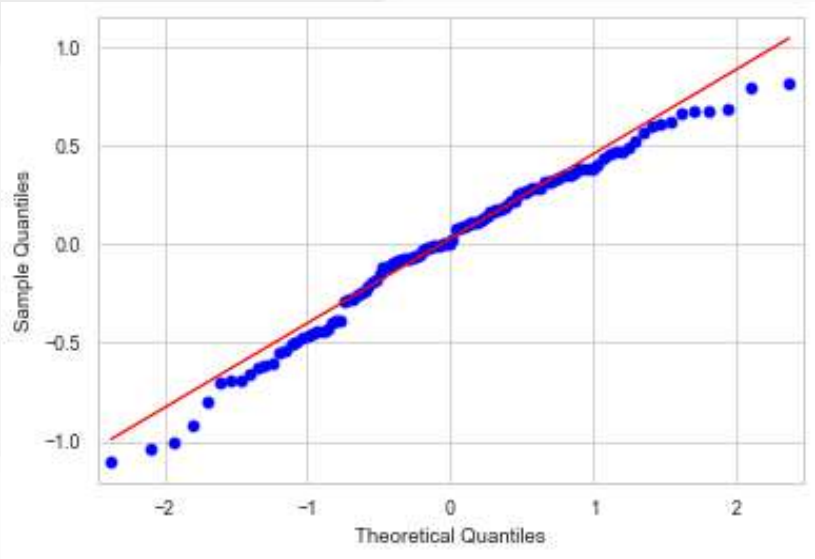


model2

```
pq = sm.ProbPlot(residual2)

pq.qqplot(line='q')
```





c. KS檢驗比較

```
standard_resid=(residual-np.mean(residual))/np.std(residual)
stats.kstest(standard_resid, 'norm')
```

```
KstestResult(statistic=0.067465695059197 5, pvalue=0.626097297867154 6)
```

```
standard_resid2=(residual2-np.mean(residual2))/np.std(residual2)
stats.kstest(standard_resid2, 'norm')
```

```
KstestResult(statistic=0.075660212873908 95, pvalue=0.518447847926760 7)
```

d. 第四次重新建模 ： 對Y進行轉換

```
import scipy.stats as stats
#找到box-cox變換得lambda係數
lamd = stats.boxcox_normmax(df_outliers.LadderScore, method = 'mle')
#對Y進行變換
df_outliers['trans_y'] = stats.boxcox(df_outliers.LadderScore, lamd)
fit5 =sm.formula.ols('trans_y~LoggedGDPPerCapita + SocialSupport + \
                      HealthyLifeExpectancy + FreedomToMakeLifeChoices + \
                      Generosity + PerceptionsOfCorruption + \
                      LadderScoreInDystopia'
                      , data = df_outliers).fit()

fit5.summary()
```

```
d = [ ["第四次建模", fit5.rsquared, fit5.rsquared_adj]]

print(tabulate(d, headers=["R^2", "adj R^2"]))
```

```
                R^2    adj R^2
----------  --------  ---------
第四次建模  0.733056   0.717802
```

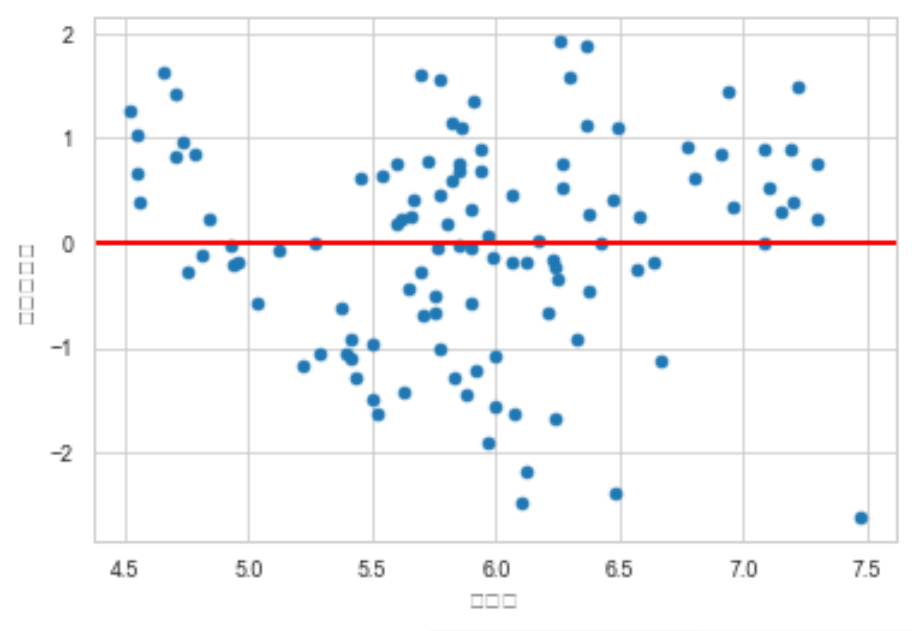
Dep. Variable:	trans_y	**R-squared:**	0.733
Model:	OLS	**Adj. R-squared:**	0.718
Method:	Least Squares	**F-statistic:**	48.06
Date:	Mon, 13 Jun 2022	**Prob (F-statistic):**	6.12e-28
Time:	16:30:32	**Log-Likelihood:**	317.70
No. Observations:	112	**AIC:**	-621.4
Df Residuals:	105	**BIC:**	-602.4
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	0.0933	0.004	26.599	0.000	0.086	0.100
LoggedGDPPerCapita	0.0080	0.003	2.820	0.006	0.002	0.014
SocialSupport	0.0680	0.023	3.014	0.003	0.023	0.113
HealthyLifeExpectancy	0.0008	0.000	1.871	0.064	-5e-05	0.002
FreedomToMakeLifeChoices	0.0758	0.017	4.479	0.000	0.042	0.109
Generosity	0.0035	0.010	0.340	0.735	-0.017	0.024
PerceptionsOfCorruption	-0.0201	0.009	-2.244	0.027	-0.038	-0.002
LadderScoreInDystopia	0.2268	0.009	26.599	0.000	0.210	0.244

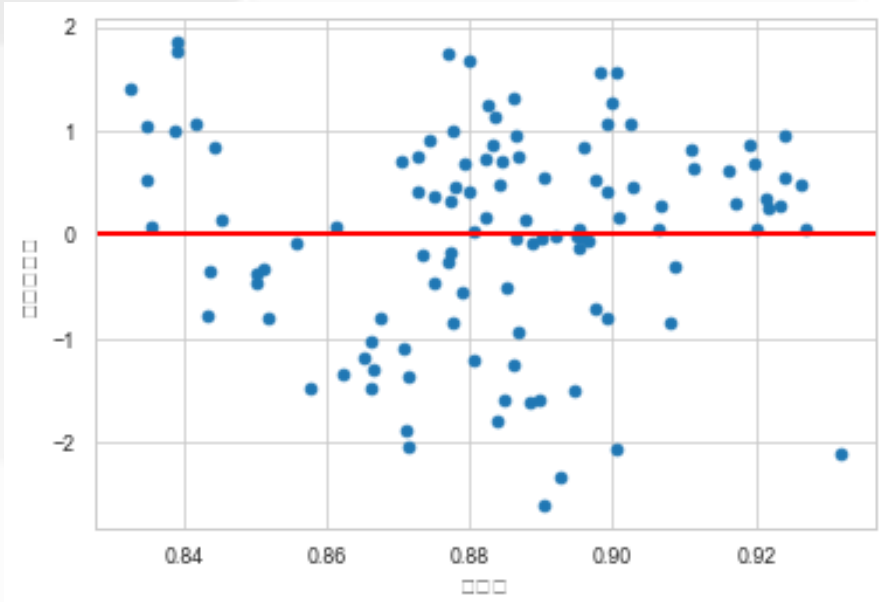
Omnibus:	5.151	**Durbin-Watson:**	1.676
Prob(Omnibus):	0.076	**Jarque-Bera (JB):**	5.168
Skew:	-0.491	**Prob(JB):**	0.0755
Kurtosis:	2.624	**Cond. No.**	4.68e+17

Step 7. 標準化殘差檢定圖

```
#圖示法完成方差齊性的判斷
#標準化殘差與預測值之間的散點圖
plt.scatter(fit2.predict(), (fit2.resid-fit2.resid.mean())/fit2.resid.std())
plt.xlabel('預測值')
plt.ylabel('標準化殘差')
#添加水平參考線
plt.axhline(y = 0, color = 'r',linewidth = 2)
plt.show()
```




```
#圖示法完成方差齊性的判斷
#標準化殘差與預測值之間的散點圖
plt.scatter(fit3.predict(), (fit3.resid-fit3.resid.mean())/fit3.resid.std())
plt.xlabel('預測值')
plt.ylabel('標準化殘差')
#添加水平參考線
plt.axhline(y = 0, color = 'r',linewidth = 2)
plt.show()
```



a. 對數變換

```
import math
df_outliers['log_y']=df_outliers["LadderScore"].map(lambda x: math.log(x,15))
df_outliers['log_StandardErrorOfLadderScore']=df_outliers["StandardErrorOfLadderScore"].map(lambda x: math.log(x,15))
df_outliers['log_LoggedGDPPerCapita']=df_outliers["LoggedGDPPerCapita"].map(lambda x: math.log(x,15))
df_outliers['log_SocialSupport']=df_outliers["SocialSupport"].map(lambda x: math.log(x,15))
df_outliers['log_HealthyLifeExpectancy']=df_outliers["HealthyLifeExpectancy"].map(lambda x: math.log(x,15))
df_outliers['log_FreedomToMakeLifeChoices']=df_outliers["FreedomToMakeLifeChoices"].map(lambda x: math.log(x,15))
df_outliers['log_PerceptionsOfCorruption']=df_outliers["PerceptionsOfCorruption"].map(lambda x: math.log(x,15))
df_outliers['log_LadderScoreInDystopia']=df_outliers["LadderScoreInDystopia"].map(lambda x: math.log(x,15))

df_outliers.head()
```

b. 第五次重新建模 ： 使用對數建模

```
fit6_log = sm.formula.ols('log_y~LoggedGDPPerCapita + SocialSupport + \
                          HealthyLifeExpectancy + FreedomToMakeLifeChoices + \
                          Generosity + PerceptionsOfCorruption + \
                          LadderScoreInDystopia'
                          , data = df_outliers).fit()

fit6_log.summary()
```

Dep. Variable:	log_y	**R-squared:**	0.734
Model:	OLS	**Adj. R-squared:**	0.719
Method:	Least Squares	**F-statistic:**	48.38
Date:	Mon, 13 Jun 2022	**Prob (F-statistic):**	4.76e-28
Time:	16:30:33	**Log-Likelihood:**	250.80
No. Observations:	112	**AIC:**	-487.6
Df Residuals:	105	**BIC:**	-468.6
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	0.0344	0.006	5.396	0.000	0.022	0.047
LoggedGDPPerCapita	0.0152	0.005	2.931	0.004	0.005	0.025
SocialSupport	0.1215	0.041	2.964	0.004	0.040	0.203
HealthyLifeExpectancy	0.0013	0.001	1.616	0.109	-0.000	0.003
FreedomToMakeLifeChoices	0.1345	0.031	4.376	0.000	0.074	0.195
Generosity	0.0096	0.019	0.509	0.612	-0.028	0.047
PerceptionsOfCorruption	-0.0456	0.016	-2.803	0.006	-0.078	-0.013
LadderScoreInDystopia	0.0836	0.015	5.396	0.000	0.053	0.114

Omnibus:	4.721	**Durbin-Watson:**	1.663
Prob(Omnibus):	0.094	**Jarque-Bera (JB):**	4.734
Skew:	-0.470	**Prob(JB):**	0.0938
Kurtosis:	2.638	**Cond. No.**	4.68e+17

```
d = [ ["第五次建模", fit6_log.rsquared, fit6_log.rsquared_adj]]

print(tabulate(d, headers=["R^2", "adj R^2"]))
```

```
                R^2    adj R^2
----------  --------  ---------
第五次建模  0.734345   0.719165
```

c. 計算誤差

```
df_outliers["pred4"]=fit6_log.predict()
abs3_ = (df_outliers['pred4'] -df_outliers['log_y']).abs()
#mae,平均絕對誤差
mae3_= abs3_.mean()
#rmse,均方根誤差
rmse3_ = ((abs3_**2).mean())**0.5
#mape,平均絕對百分比誤差
mape3_ = (abs3_/df_outliers['log_y']).mean()

print('誤差值 = ', mape3_)
```

```
誤差值 =  0.032270335249057994
```

Step 8. ANOVA Table

```
import pandas as pd
import researchpy as rp
rp.summary_cont(df['LadderScore'])
```

	Variable	N	Mean	SD	SE	95% Conf.	Interval
0	LadderScore	149.0	5.5328	1.0739	0.088	5.359	5.7067

```
model = sm.formula.ols('LadderScore~LoggedGDPPerCapita + SocialSupport + \
                        HealthyLifeExpectancy + FreedomToMakeLifeChoices + \
                        Generosity + PerceptionsOfCorruption + \
                        LadderScoreInDystopia'
                        , data = df_outliers).fit()

result = sm.stats.anova_lm(model, type=2)
print('原始建模')
print(result)
```

原始建模

	df	sum_sq	mean_sq	F	\
LoggedGDPPerCapita	1.0	39.479700	39.479700	210.213134	
SocialSupport	1.0	4.534560	4.534560	24.144663	
HealthyLifeExpectancy	1.0	0.930714	0.930714	4.955666	
FreedomToMakeLifeChoices	1.0	6.595014	6.595014	35.115733	
Generosity	1.0	0.380142	0.380142	2.024097	
PerceptionsOfCorruption	1.0	2.201754	2.201754	11.723433	
LadderScoreInDystopia	1.0	0.434498	0.434498	2.313524	
Residual	105.0	19.719836	0.187808	NaN	

	PR(>F)
LoggedGDPPerCapita	8.165657e-27
SocialSupport	3.305200e-06
HealthyLifeExpectancy	2.814180e-02
FreedomToMakeLifeChoices	3.992819e-08
Generosity	1.577851e-01
PerceptionsOfCorruption	8.813401e-04
LadderScoreInDystopia	1.312588e-01
Residual	NaN

```
fit2 = smf.ols('LadderScore~ LoggedGDPPerCapita + SocialSupport + \
                HealthyLifeExpectancy + FreedomToMakeLifeChoices + \
                PerceptionsOfCorruption + LadderScoreInDystopia'
                , data = Train.drop('Generosity', axis = 1)).fit()

result2 = sm.stats.anova_lm(fit2, type=2)
print('第一次重新建模')
print(result2)
```

第一次重新建模

	df	sum_sq	mean_sq	F	PR(>F)
LoggedGDPPerCapita	1.0	88.128918	88.128918	275.155235	4.653080e-32
SocialSupport	1.0	8.650668	8.650668	27.009031	9.084253e-07
HealthyLifeExpectancy	1.0	4.597534	4.597534	14.354375	2.444682e-04
FreedomToMakeLifeChoices	1.0	5.644001	5.644001	17.621643	5.394397e-05
PerceptionsOfCorruption	1.0	1.247321	1.247321	3.894374	5.088927e-02
LadderScoreInDystopia	1.0	0.207096	0.207096	0.646593	4.230218e-01
Residual	113.0	36.192543	0.320288	NaN	NaN

```
fit3 = smf.ols('LadderScore~ SocialSupport + HealthyLifeExpectancy + \
                FreedomToMakeLifeChoices + PerceptionsOfCorruption + \
                LadderScoreInDystopia'
                ,data = Train.drop('LoggedGDPPerCapita', axis = 1)).fit()

result3 = sm.stats.anova_lm(fit3, type=2)
print('第二次重新建模')
print(result3)
```

第二次重新建模

	df	sum_sq	mean_sq	F	PR(>F)
SocialSupport	1.0	85.702821	85.702821	262.274276	2.449714e-31
HealthyLifeExpectancy	1.0	14.892938	14.892938	45.576500	6.457762e-10
FreedomToMakeLifeChoices	1.0	4.784159	4.784159	14.640847	2.128068e-04
PerceptionsOfCorruption	1.0	1.829523	1.829523	5.598846	1.965953e-02
LadderScoreInDystopia	1.0	0.214852	0.214852	0.657506	4.191314e-01
Residual	114.0	37.251543	0.326768	NaN	NaN

```
fit4 = sm.formula.ols('LadderScore~ LoggedGDPPerCapita + SocialSupport + \
                HealthyLifeExpectancy + FreedomToMakeLifeChoices + \
                Generosity + PerceptionsOfCorruption +\
                LadderScoreInDystopia'\
                ,data = df_outliers).fit()

result4 = sm.stats.anova_lm(fit4, type=2)
print('第三次重新建模')
print(result4)
```

第三次重新建模

	df	sum_sq	mean_sq	F	PR(>F)
LoggedGDPPerCapita	1.0	39.479700	39.479700	210.213134	8.165657e-27
SocialSupport	1.0	4.534560	4.534560	24.144663	3.305200e-06
HealthyLifeExpectancy	1.0	0.930714	0.930714	4.955666	2.814180e-02
FreedomToMakeLifeChoices	1.0	6.595014	6.595014	35.115733	3.992819e-08
Generosity	1.0	0.380142	0.380142	2.024097	1.577851e-01
PerceptionsOfCorruption	1.0	2.201754	2.201754	11.723433	8.813401e-04
LadderScoreInDystopia	1.0	0.434498	0.434498	2.313524	1.312588e-01
Residual	105.0	19.719836	0.187808	NaN	NaN

```
import scipy.stats as stats
lamd = stats.boxcox_normmax(df_outliers.LadderScore, method = 'mle')

df_outliers['trans_y'] = stats.boxcox(df_outliers.LadderScore, lamd)
fit5 =sm.formula.ols('trans_y~LoggedGDPPerCapita + SocialSupport + \
                HealthyLifeExpectancy + FreedomToMakeLifeChoices + \
                Generosity + PerceptionsOfCorruption + \
                LadderScoreInDystopia'
                , data = df_outliers).fit()

result5 = sm.stats.anova_lm(fit5, type=2)
print('第四次重新建模')
print(result5)
```

第四次重新建模

	df	sum_sq	mean_sq	F	PR(>F)
LoggedGDPPerCapita	1.0	0.046163	0.046163	215.072395	3.644097e-27
SocialSupport	1.0	0.005971	0.005971	27.819553	7.190708e-07
HealthyLifeExpectancy	1.0	0.001571	0.001571	7.317284	7.970173e-03
FreedomToMakeLifeChoices	1.0	0.006962	0.006962	32.434535	1.132338e-07
Generosity	1.0	0.000142	0.000142	0.661809	4.177631e-01
PerceptionsOfCorruption	1.0	0.001081	0.001081	5.035963	2.692447e-02
LadderScoreInDystopia	1.0	0.000204	0.000204	0.951058	3.316916e-01
Residual	105.0	0.022537	0.000215	NaN	NaN

```
fit6_log = sm.formula.ols('log_y~LoggedGDPPerCapita + SocialSupport + \
                HealthyLifeExpectancy + FreedomToMakeLifeChoices + \
                Generosity + PerceptionsOfCorruption + \
                LadderScoreInDystopia'
                , data = df_outliers).fit()

result6 = sm.stats.anova_lm(fit6_log, type=2)
print('第五次重新建模')
print(result6)
```

第五次重新建模

	df	sum_sq	mean_sq	F	PR(>F)
LoggedGDPPerCapita	1.0	0.152210	0.152210	214.753746	3.840635e-27
SocialSupport	1.0	0.018640	0.018640	26.298873	1.343605e-06
HealthyLifeExpectancy	1.0	0.004380	0.004380	6.179798	1.449868e-02
FreedomToMakeLifeChoices	1.0	0.024059	0.024059	33.944358	6.278676e-08
Generosity	1.0	0.000862	0.000862	1.216372	2.725953e-01
PerceptionsOfCorruption	1.0	0.005569	0.005569	7.856964	6.030677e-03
LadderScoreInDystopia	1.0	0.001054	0.001054	1.487553	2.253291e-01
Residual	105.0	0.074420	0.000709	NaN	NaN

Step 9. R語言CP = Python決策樹

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score,accuracy_score
from sklearn import preprocessing
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 = 5)
x_train.shape,x_test.shape
```

```
((104, 8), (45, 8))
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from IPython.display import Image

model = DecisionTreeClassifier(criterion = 'entropy',max_depth=3)
clf = model.fit(X,y.astype('int'))

y_pred = model.predict(X)

print("決策樹預測率:"+str(model.score(X,y.astype('int'))))
```

```
決策樹預測率:0.6711409395973155
```

Step 10. AIC and BIC => Forward 、 Backward and Step-Wise

a. AIC and BIC

```
import time
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LassoLarsIC
from sklearn.pipeline import make_pipeline

start_time = time.time()
lasso_lars_ic = make_pipeline(
    StandardScaler(), LassoLarsIC(criterion="aic", normalize=False)
).fit(X, y)

fit_time = time.time() - start_time
```

```
results = pd.DataFrame(
    {
        "alphas": lasso_lars_ic[-1].alphas_,
        "AIC criterion": lasso_lars_ic[-1].criterion_,
    }
).set_index("alphas")

alpha_aic = lasso_lars_ic[-1].alpha_
```

```
lasso_lars_ic.set_params(lassolarsic__criterion="bic").fit(X, y)
results["BIC criterion"] = lasso_lars_ic[-1].criterion_
alpha_bic = lasso_lars_ic[-1].alpha_
```

```
def highlight_min(x):
    x_min = x.min()
    return ["font-weight: bold" if v == x_min else "" for v in x]

results.style.apply(highlight_min)
```

alphas	AIC criterion	BIC criterion
0.8452906453888189	666.703705	666.703705
0.6814192343098944	543.475045	546.478991
0.6800720505260539	544.446544	550.454436
0.45802032533781484	398.357027	407.368866
0.18034937246735258	271.593594	283.609379
0.05012510644857145	246.803836	261.823567
0.0	245.292447	263.316125

b. Forward selection

```
def forward_selection(data, target, significance_level=0.05):
    initial_features = data.columns.tolist()
    best_features = []
    while (len(initial_features)>0):
        remaining_features = list(set(initial_features)-set(best_features))
        new_pval = pd.Series(index=remaining_features)
        for new_column in remaining_features:
            model = sm.OLS(target, sm.add_constant(data[best_features+[new_column]])).fit()
            new_pval[new_column] = model.pvalues[new_column]
        min_p_value = new_pval.min()
        if(min_p_value<significance_level):
            best_features.append(new_pval.idxmin())
        else:
            break
    return best_features
```

```
forward_selection(X,y)
```

```
#importing the necessary libraries
import mlxtend
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from sklearn.linear_model import LinearRegression
# Sequential Forward Selection(sfs)
sfs = SFS(LinearRegression(),
          k_features=8,
          forward=True,
          floating=False,
          scoring = 'r2',
          cv = 0)

sfs.fit(X,y)
sfs.k_feature_names_
```

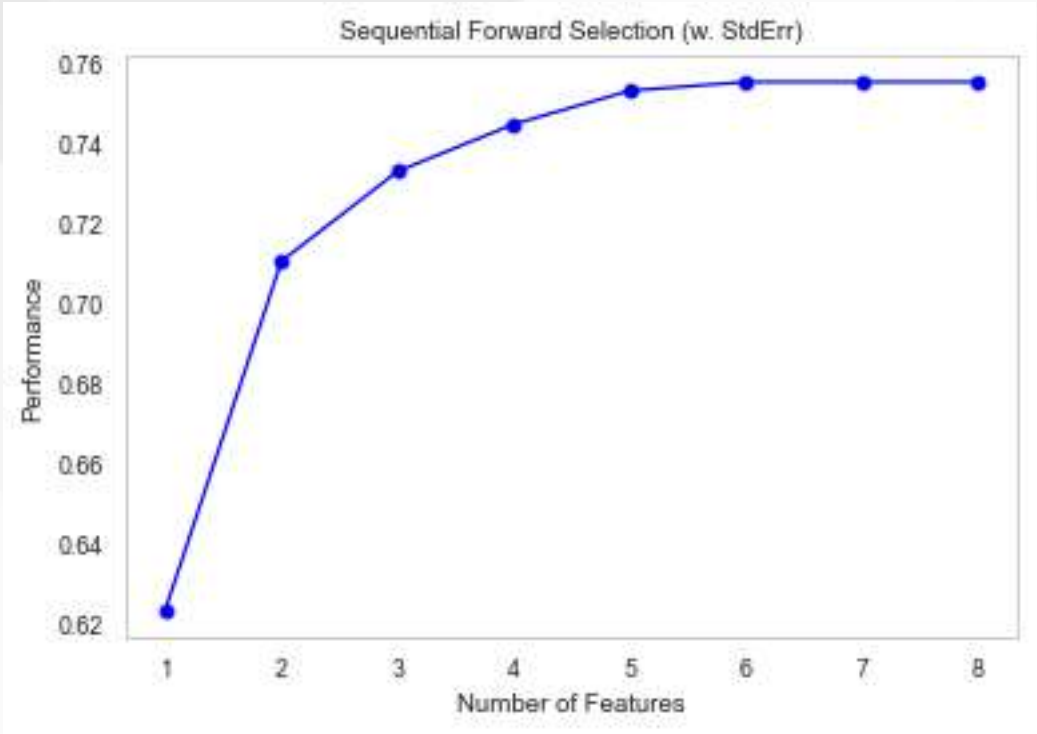
```
('Intercept',
 'LoggedGDPPerCapita',
 'SocialSupport',
 'HealthyLifeExpectancy',
 'FreedomToMakeLifeChoices',
 'Generosity',
 'PerceptionsOfCorruption',
 'LadderScoreInDystopia')
```

```
df_SFS_results = pd.DataFrame(sfs.subsets_ ).transpose()
df_SFS_results
```

	feature_idx	cv_scores	avg_score	feature_names
1	(1,)	[0.6237203782313991]	0.62372	(LoggedGDPPerCapita,)
2	(1, 4)	[0.7109512261766451]	0.710951	(LoggedGDPPerCapita, FreedomToMakeLifeChoices)
3	(1, 2, 4)	[0.7334321909172843]	0.733432	(LoggedGDPPerCapita, SocialSupport, FreedomToM...
4	(1, 2, 4, 6)	[0.7452626912764992]	0.745263	(LoggedGDPPerCapita, SocialSupport, FreedomToM...
5	(1, 2, 3, 4, 6)	[0.753634549709 28]	0.753635	(LoggedGDPPerCapita, SocialSupport, HealthyLif...
6	(1, 2, 3, 4, 5, 6)	[0.7558471374226853]	0.755847	(LoggedGDPPerCapita, SocialSupport, HealthyLif...
7	(0, 1, 2, 3, 4, 5, 6)	[0.7558471374226855]	0.755847	(Intercept, LoggedGDPPerCapita, SocialSupport,...
8	(0, 1, 2, 3, 4, 5, 6, 7)	[0.7558471374226856]	0.755847	(Intercept, LoggedGDPPerCapita, SocialSupport,...

```
fig = plot_sfs(sfs.get_metric_dict(), kind='std_err')

plt.title('Sequential Forward Selection (w. StdErr)')
plt.grid()
plt.show()
```



c. Backward elimination

```
def backward_elimination(data, target,significance_level = 0.05):
    features = data.columns.tolist()
    while(len(features)>0):
        features_with_constant = sm.add_constant(data[features])
        p_values = sm.OLS(target, features_with_constant).fit().pvalues[1:]
        max_p_value = p_values.max()
        if(max_p_value >= significance_level):
            excluded_feature = p_values.idxmax()
            features.remove(excluded_feature)
        else:
            break
    return features
```

```
backward_elimination(X,y)
```

```
['Intercept',
 'LoggedGDPPerCapita',
 'SocialSupport',
 'HealthyLifeExpectancy',
 'FreedomToMakeLifeChoices',
 'PerceptionsOfCorruption',
 'LadderScoreInDystopia']
```

```
sbs = SFS(LinearRegression(),
          k_features=8,
          forward=False,
          floating=False,
          cv=0)

sbs.fit(X, y)
sbs.k_feature_names_
```

```
('Intercept',
 'LoggedGDPPerCapita',
 'SocialSupport',
 'HealthyLifeExpectancy',
 'FreedomToMakeLifeChoices',
 'Generosity',
 'PerceptionsOfCorruption',
 'LadderScoreInDystopia')
```

```
df_SBS_results = pd.DataFrame(sbs.subsets_ ).transpose()
df_SBS_results
```

	avg_score	cv_scores	feature_idx	feature_names
8	0.755847	[0.7558471374226856]	(0, 1, 2, 3, 4, 5, 6, 7)	(Intercept, LoggedGDPPerCapita, SocialSupport,...

d. Step-wise

```
def stepwise_selection(data, target,SL_in=0.05,SL_out = 0.05):
    initial_features = data.columns.tolist()
    best_features = []
    while (len(initial_features)>0):
        remaining_features = list(set(initial_features)-set(best_features))
        new_pval = pd.Series(index=remaining_features)
        for new_column in remaining_features:
            model = sm.OLS(target, sm.add_constant(data\
                                                   [best_features+[new_column]])).fit()
            new_pval[new_column] = model.pvalues[new_column]
        min_p_value = new_pval.min()
        if(min_p_value<SL_in):
            best_features.append(new_pval.idxmin())
            while(len(best_features)>0):
                best_features_with_constant = sm.add_constant(data[best_features])
                p_values = sm.OLS(target, best_features_with_constant).fit().pvalues[1:]

                max_p_value = p_values.max()
                if(max_p_value >= SL_out):
                    excluded_feature = p_values.idxmax()
                    best_features.remove(excluded_feature)
                else:
                    break
        else:
            break
    return best_features
```

```
stepwise_selection(X,y)
```

```
sffs = SFS(LinearRegression(),
           k_features=8,
           forward=True,
           floating=True,
           cv=0)
sffs.fit(X, y)
sffs.k_feature_names_
```

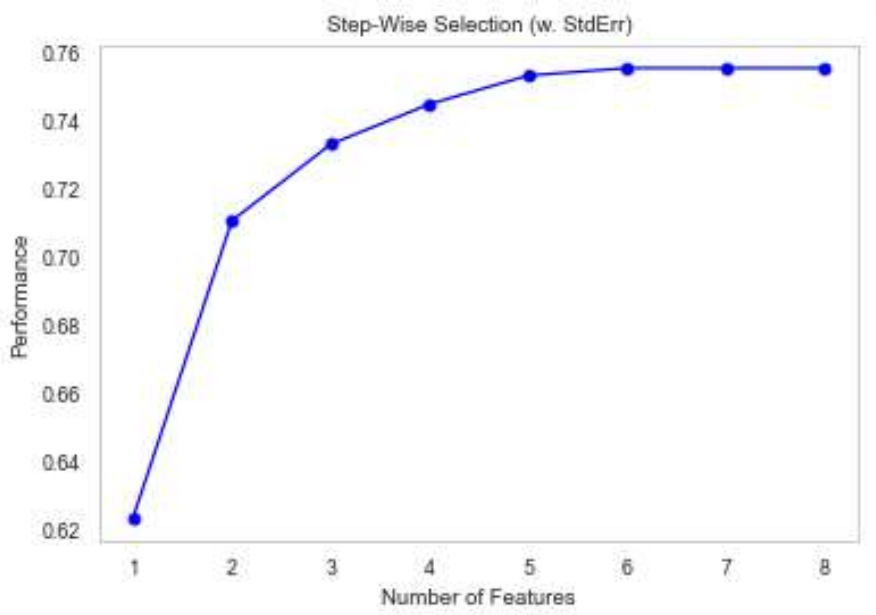
```
('Intercept',
 'LoggedGDPPerCapita',
 'SocialSupport',
 'HealthyLifeExpectancy',
 'FreedomToMakeLifeChoices',
 'Generosity',
 'PerceptionsOfCorruption',
 'LadderScoreInDystopia')
```

```
df_SFFS_results = pd.DataFrame(sffs.subsets_ ).transpose()
df_SFFS_results
```

	feature_idx	cv_scores	avg_score	feature_names
1	(1,)	[0.6237203782313991]	0.62372	(LoggedGDPPerCapita,)
2	(1, 4)	[0.7109512261766451]	0.710951	(LoggedGDPPerCapita, FreedomToMakeLifeChoices)
3	(1, 2, 4)	[0.7334321909172843]	0.733432	(LoggedGDPPerCapita, SocialSupport, FreedomToM...
4	(1, 2, 4, 6)	[0.7452626912764992]	0.745263	(LoggedGDPPerCapita, SocialSupport, FreedomToM...
5	(1, 2, 3, 4, 6)	[0.75363454970928]	0.753635	(LoggedGDPPerCapita, SocialSupport, HealthyLif...
6	(1, 2, 3, 4, 5, 6)	[0.7558471374226853]	0.755847	(LoggedGDPPerCapita, SocialSupport, HealthyLif...
7	(0, 1, 2, 3, 4, 5, 6)	[0.7558471374226855]	0.755847	(Intercept, LoggedGDPPerCapita, SocialSupport,...
8	(0, 1, 2, 3, 4, 5, 6, 7)	[0.7558471374226856]	0.755847	(Intercept, LoggedGDPPerCapita, SocialSupport,...

```
fig = plot_sfs(sffs.get_metric_dict(), kind='std_err')

plt.title('Step-Wise Selection (w. StdErr)')
plt.grid()
plt.show()
```



Step 11. 結論

```
d = [ ["原始資料", fit.rsquared, fit.rsquared_adj, fit.rsquared-fit.rsquared_adj, mape1_],
      ["第一次建模", fit2.rsquared, fit2.rsquared_adj, fit2.rsquared-fit2.rsquared_adj, mape2_],
      ["第二次建模", fit3.rsquared, fit3.rsquared_adj, fit3.rsquared-fit3.rsquared_adj, mape3_],
      ["第三次建模", fit4.rsquared, fit4.rsquared_adj, fit4.rsquared-fit4.rsquared_adj, mape4_],
      ["第四次建模", fit5.rsquared, fit5.rsquared_adj, fit5.rsquared-fit5.rsquared_adj, mape5_],
      ["第五次建模", fit6_log.rsquared, fit6_log.rsquared_adj, fit6_log.rsquared-fit6_log.rsquared_adj, mape6_]]

print(tabulate(d, headers=["R^2", "adj R^2", "R^2 - adj R^2", "誤差值"]))
```

```
                R^2    adj R^2    R^2 - adj R^2     誤差值
----------  --------  ---------  ---------------  ---------
原始資料    0.752083   0.738802       0.0132813   0.18341
第一次建模  0.749465   0.738379       0.0110856   0.182697
第二次建模  0.742134   0.733086       0.00904792  0.18341
第三次建模  0.732945   0.717684       0.0152603   0.0554098
第四次建模  0.733056   0.717802       0.0152539   0.85039
第五次建模  0.734345   0.719165       0.0151803   0.0322703
```

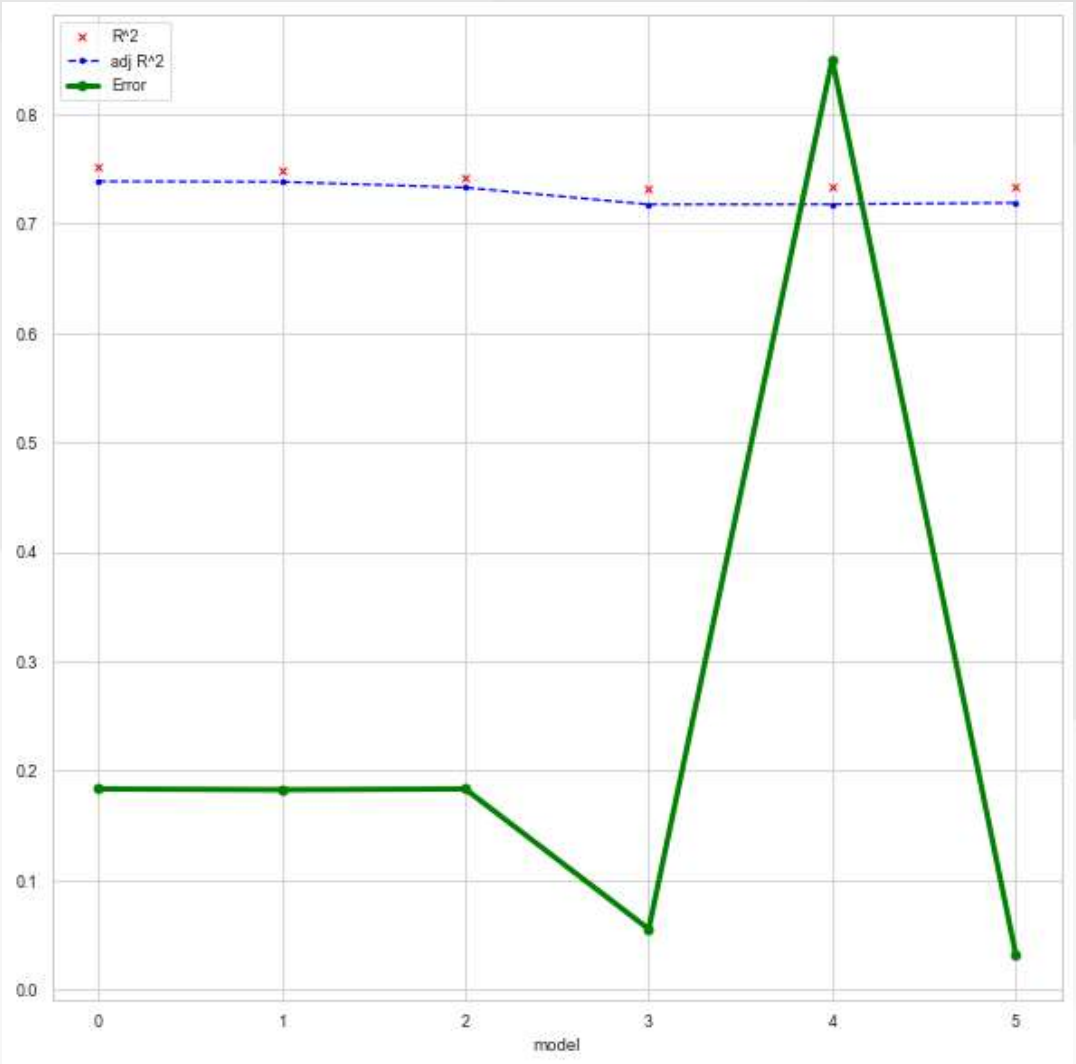
```
# r代表紅色，x代表用'x'來表示點，且不畫線
plt.figure(figsize=(10,10))
plt.plot([fit.rsquared, fit2.rsquared, fit3.rsquared, fit4.rsquared, fit5.rsquared, fit6_log.rsquared], 'rx')

# b代表藍色，'.'代表用單個小點表示一個點，'--'表示用虛線(dashed line)來畫線
plt.plot([fit.rsquared_adj, fit2.rsquared_adj, fit3.rsquared_adj, fit4.rsquared_adj, fit5.rsquared_adj, fit6_log.rsquared_adj],\
         'b.--')

# g代表綠色，o代表實心圓，linewidth表示粗度修為3
plt.plot([mape1_, mape2_, mape3_, mape4_, mape5_, mape6_], 'go-', linewidth=3)

plt.legend(('R^2', 'adj R^2', 'Error'), loc='upper left') # 畫圖例及決定位置
plt.xlabel('model')

plt.grid(True) # 畫出網格
plt.show()
```



=> 由上述2張圖表可以得知，誤差最小的是使用"模型三"，然而R^2-adj R^2最小的，也是調整過後最接近R^2的是"模型二"="第二次建模"。但因為前面2個模型P值皆"小於"顯著水準 0.05，拒絕虛無假設，表示各組的變異數不完全相等，這邊只是想演示老師上課有教到的模型來嘗試看看而已，因此是不需要往後做剩餘的模型。

=> 結論是選擇"模型二"會是最好的解決方式!

PART 3

參考資料及工作分配表

一、參考資料

1. Kaggle : World Happiness Expanatory Data Analysis
2. 維基百科 : 世界幸福報告
3. 世界幸福報告官網

二、工作分配表

	資料整理	Word檔製作	程式碼	PPT製作	報告
王常騰		✓			
溫宏岳	✓		✓		✓
張俊翔				✓	
陳曦					✓
胡家宏	✓	✓			

THANKS!