資料分析方法

經研一 江彥亨 R11323040

1. a

a.

In [255]:
```
#讀取圖片資料並轉為dataframe
data = []
for i in range(1,41):
    for j in range(1,11):
        data.append(list(Image.open(f'ORL Faces/{i}_{j}.png').getdata()))

data = pd.DataFrame(np.array(data))

#新增性別欄位，女性為0男性則為1
for i in data.index:
    if (i <= 9) | (70 <= i <= 79) | (90 <= i <= 99) | (310 <= i <= 319):
        data.loc[i, 'sex'] = 0
    else:
        data.loc[i, 'sex'] = 1

data
```

Out[255]:

	0	1	2	3	4	5	6	7	8	9	...	2567	2568	2569	2570	2571	2572	2573	2574	2575	sex
0	88	88	90	91	91	92	90	93	99	109	...	176	166	149	142	145	141	138	142	134	0.0
1	87	90	95	96	92	90	97	107	111	112	...	175	172	147	131	132	124	124	120	88	0.0
2	92	92	88	98	104	109	108	100	80	63	...	84	128	154	161	169	170	165	146	151	0.0
3	92	96	93	94	99	105	108	109	121	152	...	153	164	163	165	166	161	157	79	54	0.0
4	83	75	88	91	101	90	86	80	63	58	...	145	148	151	139	134	173	167	176	188	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
395	124	124	125	123	124	125	123	124	123	122	...	34	63	37	37	37	38	39	38	40	1.0
396	128	128	128	128	129	128	129	127	127	127	...	92	90	90	91	91	91	91	92	93	1.0
397	122	123	124	124	123	123	122	126	130	126	...	24	57	41	37	36	37	38	40	38	1.0
398	120	119	121	119	120	121	122	117	111	100	...	137	134	101	26	77	95	95	92	90	1.0
399	124	125	125	125	124	125	124	124	124	124	...	35	69	55	31	36	33	33	34	34	1.0

400 rows × 2577 columns

1.b

b.

由於樣本僅有400個，而自變數則高達2576個，自變數內積矩陣(xTx)非滿秩矩陣，不存在反矩陣，因此無法解出正確的線性迴歸之參數

In [268]:
```
#將性別對每格變數進行線性迴歸
X = data.iloc[:,:-1]
y = data['sex']
model = sm.OLS(y,X).fit()
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    sex   R-squared:                       1.000
Model:                            OLS   Adj. R-squared:                    nan
Method:                 Least Squares   F-statistic:                       nan
Date:                Sun, 12 Mar 2023   Prob (F-statistic):                nan
Time:                        20:42:38   Log-Likelihood:                 12972.
No. Observations:                 400   AIC:                        -2.514e+04
Df Residuals:                       0   BIC:                        -2.355e+04
Df Model:                         399
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
0              0.0001        inf          0        nan         nan         nan
1              0.0001        inf          0        nan         nan         nan
2              0.0001        inf          0        nan         nan         nan
3              0.0001        inf          0        nan         nan         nan
4            6.92e-05        inf          0        nan         nan         nan
```

1.c

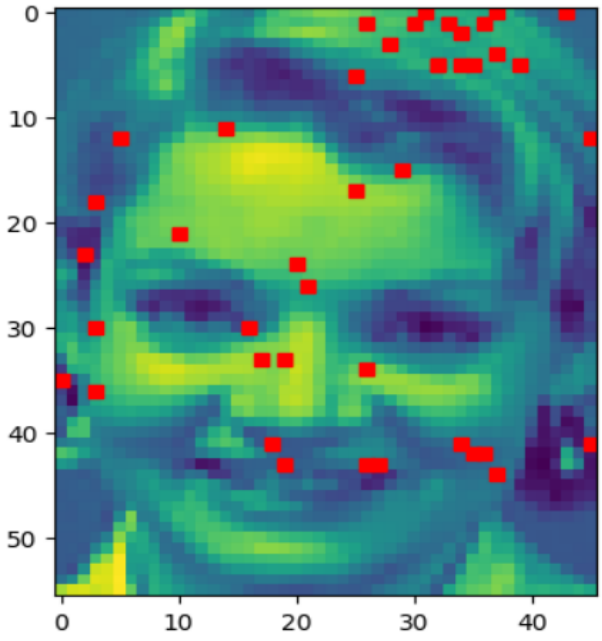
c.

In [273]:

```
#定義stepwise regression函數
def stepwise_regression(X,y):
    #紀錄顯著的像素
    in_var = []

    while True:

        change = False

        #Forward selection :
        out_var = list(set(X.columns)-set(in_var))

        #紀錄新像素之p-value
        new_x_pval = pd.Series(index=out_var)

        #對所有不在in_var的新像素進行迴歸並選出p-value最小的像素
        for x in out_var:
            model = sm.OLS(y, sm.add_constant(X.loc[:, in_var+[x]])).fit()
            new_x_pval[x] = model.pvalues[x]
        min_pval = new_x_pval.min()
        #若p-value小於0.01則加入in_var
        if min_pval <= 0.01:
            in_var.append(new_x_pval.idxmin())
            change = True

        #Backward selection :
        #對in_var所有像素進行迴歸並選出p-value最大者
        model = sm.OLS(y, sm.add_constant(X.loc[:, in_var])).fit()
        pval = model.pvalues[1:]
        max_pval = pval.max()

        #若p-value大於0.01則去除該像素
        if max_pval >= 0.01:
            in_var.remove(pval.idxmax())
            change = True

        #若加入的新像素不夠顯著或in_var中無不顯著的像素則終止迴圈
        if not change:
            break

    return in_var
significant_pixels = stepwise_regression(X,y)
```

In [287]:

```
#標示出顯著的像素格
img = mpimg.imread('ORL Faces/1_1.png')
plt.imshow(img)
for pixel in significant_pixels:
    plt.plot(pixel//56, pixel%46, 'rs')
```



2.

Q2

In [5]:
```
#讀取火山高度資料並找出最高點
vc = np.array(pd.read_csv('Volcano.csv'))
top = vc.max()
x,y = np.where(vc == top)
x = int(x)
y = int(y)
print(f'火山最高點位置為({x},{y})，高度為{top}')
```

```
火山最高點位置為(30,19)，高度為195
```

In [6]:
```
import warnings
warnings.filterwarnings("ignore", message="RuntimeWarning: invalid value encountered in divide")
```

In [7]:
```
#定義送帶迴歸函數，(x,y)為起始點
def IterReg(x,y):
    global path
    path = []
    max = 0
    while max < 195:
        #檢查若在邊界則往內移動
        if x == 86:
            x -= 1
        if x == 0:
            x += 1
        if y == 60:
            y -= 1
        if y == 0:
            y += 1

        #將所處的點及周圍八個點進行多元迴歸並計算出該點梯度
        X = np.array([])
        Y = np.array([])
        for row in [y-1, y, y+1]:
            for col in [x-1, x, x+1]:
                X = np.append(X,[row,col])
                Y = np.append(Y,vc[row,col])
        X = X.reshape([9,2])

        model = LinearRegression().fit(X, Y)
        coef = model.coef_.round(4)
        warnings.filterwarnings("ignore")
        gradient = model.coef_ / np.sqrt(np.sum(model.coef_ ** 2))

        #加入隨機干擾項以避免梯度消失問題
        random_noise = np.random.normal(0.1,size=2)
        gradient += random_noise

        #根據梯度值來進行移動，若係數接近0則不移動
        #左
        if (coef[0] == 0.) & (gradient[1] < 0):
            x -= 1
        #右
        elif (coef[0] == 0.) & (gradient[1] > 0):
            x += 1
        #上
        elif (gradient[0] < 0) & (coef[1] == 0.):
            y -= 1
        #下
        elif (gradient[0] > 0) & (coef[1] == 0.):
            y += 1
        #左上
        elif (gradient < 0).all():
            y -= 1
            x -= 1
        #右下
        elif (gradient > 0).all():
            y += 1
            x += 1
        #左下
        elif (gradient[0] > 0) & (gradient[1] < 0):
            x -= 1
            y += 1
        #右上
        elif (gradient[0] < 0) & (gradient[1] > 0):
            x += 1
            y -= 1
        #若係數皆為0且高度未滿100則持續朝左上移動
        elif (coef == 0.).all() & max < 100:
            y -= 1
            x -= 1
        #若係數皆為0則隨機移動到周圍1格的位置
        elif (coef == 0.).all():
            y -= np.random.randint(-1,1)
            x -= np.random.randint(-1,1)

        max = vc[y,x]

        #紀錄路徑
        path.append((y,x,max))

    #抵達最高點後終止迴圈
    if max == 195:
        print('Arrive at the highest point of the volcano!!!')
```

```
#計算花費多少時間抵達最高點
import time
start_time = time.time()

IterReg(86,60)

end_time = time.time()
run_time = end_time - start_time
print(f"程式運行時間為 {run_time} 秒")
```

```
Arrive at the highest point of the volcano!!!
程式運行時間為 9.00968074798584 秒
```

2. a

Q3

a.

以下資料來自kaggle 的 Home Credit Default Risk 的訓練資料集，並且將變數縮減為2個和樣本數縮減為50000個
網址 https://www.kaggle.com/competitions/home-credit-default-risk/data?select=application_train.csv

In [208]:
```
df = pd.read_csv('application_train.csv')
df
```

Out[208]:

	TARGET	AMT_INCOME_TOTAL	AMT_CREDIT
0	1	202500.0	406597.5
1	0	270000.0	1293502.5
2	0	67500.0	135000.0
3	0	135000.0	312682.5
4	0	121500.0	513000.0
...	...	...	...
49995	0	126000.0	1125000.0
49996	0	112500.0	900000.0
49997	0	270000.0	820638.0
49998	0	117000.0	254700.0
49999	0	67500.0	343800.0

50000 rows × 3 columns

由迴歸結果可看出，AMT_INCOME_TOTAL越高則違約可能性越高，AMT_CREDIT越高則違約可能性越低

In [218]:
```
X = df[['AMT_INCOME_TOTAL', 'AMT_CREDIT']]
y = df['TARGET']
model = sm.OLS(y,sm.add_constant(X)).fit()
print(model.summary())
print()
print(f'MSE is {model.mse_resid}')
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 TARGET   R-squared:                       0.001
Model:                            OLS   Adj. R-squared:                  0.001
Method:                 Least Squares   F-statistic:                     30.66
Date:                Sun, 12 Mar 2023   Prob (F-statistic):           4.94e-14
Time:                        17:40:42   Log-Likelihood:                -5836.4
No. Observations:               50000   AIC:                         1.168e+04
Df Residuals:                   49997   BIC:                         1.171e+04
Df Model:                           2
Covariance Type:            nonrobust
===================================================================================
                      coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------
const               0.0929      0.002     42.246      0.000       0.089       0.097
AMT_INCOME_TOTAL  6.757e-09   2.29e-09      2.948      0.003    2.26e-09    1.12e-08
AMT_CREDIT       -2.254e-08   3.03e-09     -7.441      0.000   -2.85e-08   -1.66e-08
==============================================================================
Omnibus:                    30368.913   Durbin-Watson:                   2.010
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           195927.494
Skew:                           3.079   Prob(JB):                         0.00
Kurtosis:                      10.492   Cond. No.                     1.37e+06
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.37e+06. This might indicate that there are
strong multicollinearity or other numerical problems.

MSE is 0.07395027912876959
```

3.b

b.

根據以下結果，梯度下降無法得到和OLS一樣的結果，且即便初始參數已很接近OLS得到的參數，MSE仍舊隨著迭帶次數增加而上升，
顯示此資料不適合使用梯度下降法尋找最佳參數

In [219]:
```
#定義梯度下降函數
def gradient_decent(X, y, beta=[], learning_rate = 10e-12, max_iteration = 10):
    n = len(X) #樣本數
    p = len(X.columns) #變數個數
    beta = np.array(beta) #初始參數

    beta_lst = []
    mse_lst = []
    for i in range(max_iteration):
        #計算預測值
        y_pred = X.values.dot(beta)
        #計算誤差項
        error = y_pred - y.values
        #計算梯度
        gradient = X.values.T.dot(error)/n
        #更新參數
        beta = beta - learning_rate*gradient
        #紀錄每次更新的參數和MSE
        beta_lst.append(beta)
        mse_lst.append(np.sum(np.square(error)))

    plt.plot(mse_lst)
    return beta_lst
#加入常數項
X.insert(0, 'intercept', np.ones(len(X)))

gradient_decent(X, y, beta=[9e-02, 6e-09, -2e-08])
```

Out[219]:
```
[array([ 9.00000000e-02,  1.03061395e-08, -1.50749304e-08]),
 array([ 9.0000000e-02, -4.5963066e-09, -4.0886022e-08]),
 array([9.00000000e-02, 5.72554556e-08, 8.53970737e-08]),
 array([ 9.00000000e-02, -2.21915164e-07, -5.19487675e-07]),
 array([9.00000000e-02, 1.07905151e-06, 2.35781519e-06]),
 array([ 9.00000000e-02, -5.05215338e-06, -1.12972386e-05]),
 array([9.00000001e-02, 2.39542718e-05, 5.34563810e-05]),
 array([ 0.09      , -0.00011345, -0.00025353]),
 array([0.09      , 0.00053774, 0.00120173]),
 array([ 0.08999999, -0.00254884, -0.00569665])]
```