機器學習< Assignment #2 - Regression>

姓名: 翁佳煌　　學號: 409430030

要求 1:

下圖 1.1 為要求 1 的 code。

1. 在第 6 行設定了隨機種子，確保在每次運行程式時生成的隨機數序列都是相同的，以方便此次實驗的觀察。

2. 接下來第 8 行，使用 NumPy 的 linspace 函數在區間[-3, 3]中均勻生成 15 個數字，並將結果儲存在變數 x 中。

3. 第 11 行使用 NumPy 的 normal 函數來生成一個包含 15 個隨機數的數組，其符合均值為 0，標準差為 1 的高斯分佈，並將結果儲存在變數 noise 中。

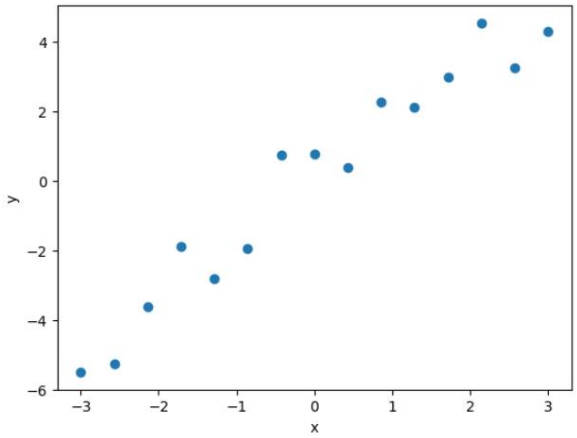
4. 接著 13 行，將噪聲加到 x 中，並通過線性模型生成對應的 y 值。

5. 15~19 行，我在這裡使用 Matplotlib 庫，將數據點以散點圖的形式呈現出來。

```
1 #Requirement 1
2
3 import numpy as np
4 import matplotlib.pyplot as plt
5
6 np.random.seed(42)
7 # Generate 15 data points with equal spacing in the interval [-3, 3]
8 x = np.linspace(-3, 3, num=15)
9
10 # Generate the noise samples from a zero-mean Gaussian distribution with standard deviation 1
11 noise = np.random.normal(loc=0, scale=1, size=15)
12 # Compute the corresponding y values using the linear model
13 y = 2 * x + noise
14
15 # Plot the data points as a scatter plot
16 plt.scatter(x, y)
17 plt.xlabel('x')
18 plt.ylabel('y')
19 plt.show()
20
```

▲圖 1.1

下圖 1.2 為要求 1 的執行結果:



▲圖 1.2

要求 1 問題討論:

一開始遇到要加上高斯 noise，之前在機率學過，但幾乎要忘光了，於是又多花了點時間去搞懂這部分。另外，透過將高斯噪聲加入到數據集中，可以使生成的數據更加逼近現實情況，從而使得線性回歸等機器學習模型更能夠有效地處理這些數據。

要求 2:

下圖 2.1 為要求 2 的 code。

1. 1~10 行直接採用要求 1 的 code 去做延伸，生成帶有高斯噪聲的數據。使用 np.linspace 生成-3 到 3 之間的 15 個等距 x 值，然後生成對應的 y 值，並且通過給定的線性模型 y = 2x + ε 和高斯噪聲 ε 來計算 y 值。

2. 13 行創建設計矩陣。將剛生成的 x 和一個全為 1 的向量，按列組成一個矩陣，即設計矩陣 X。

3. 計算權重向量。使用課堂上提到的 Linear Regression 方程式求解權重向量 W，即 W = (X.T * X)-1 * X.T * y。

4. 計算訓練誤差。使用設計矩陣 X 和權重向量 W，計算預測值 y_pred_train，並且通過平均平方誤差(MSE)來計算訓練誤差 mse_train。

5. 計算五倍交叉驗證誤差。使用 sklearn 中的 KFold 函數將數據集分成 5 份，然後對每個部分進行循環，每次選擇其中一份作為驗證集，其餘部分作為訓練集。對於每次分割，使用訓練集計算權重向量 W_cv，然後在驗證集上進行預測，計算預測值 y_pred_cv 和 MSE，並且將 MSE 添加到 mse_cv 列表中。

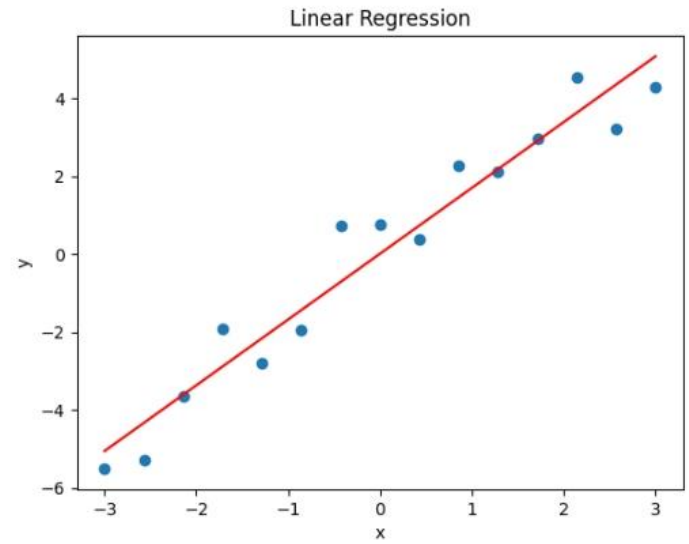
6. 繪製擬合線和數據。使用 plt.scatter 和 plt.plot 函數繪製數據點和擬合線。

7. 輸出訓練誤差和交叉驗證誤差。使用 print 函數輸出訓練誤差和交叉驗證誤差。

```python
#Requirement 2
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold

# Generate the data
np.random.seed(42)
x = np.linspace(-3, 3, num=15)
noise = np.random.normal(loc=0, scale=1, size=15)
y = 2 * x + noise

# Create the design matrix
X = np.column_stack((np.ones_like(x), x))

# Compute the weight vector using the normal equation
W = np.linalg.inv(X.T @ X) @ X.T @ y

# Compute the predicted values and the training error
y_pred_train = X @ W

mse_train = np.mean((y - y_pred_train)**2)

# Compute the five-fold cross-validation errors
kf = KFold(n_splits=5, shuffle=True, random_state=42)
mse_cv = []
for train_index, test_index in kf.split(X):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]
        W_cv = np.linalg.inv(X_train.T @ X_train) @ X_train.T @ y_train
        y_pred_cv = X_test @ W_cv
        mse_cv.append(np.mean((y_test - y_pred_cv)**2))

# Plot the fitting line and the data
plt.scatter(x, y)
plt.plot(x, y_pred_train, color='red')
plt.title('Linear Regression')
plt.xlabel('x')
plt.ylabel('y')
plt.show()

# Print the training error and the cross-validation errors
print('Training error:', mse_train)
print('Cross-validation errors:', mse_cv)

```

▲圖 2.1

下圖 2.2 為要求 2 的執行結果:




```
Training error: 0.5899826233860157
Cross-validation errors: [0.3800533653381555, 0.9589254301285632, 0.7188262893606258, 0.3393074612157955, 1.9638207608102423]
```

▲圖 2.2

要求 2 問題討論:

我發現 Linear Regression 其實可以直接 from sklearn.linear_model import LinearRegression 來做使用，但我知道老師一定是要我們實作這部分，我現在才漸漸清楚很多模型其實是可以直接套模來使用的。

此外，根據執行結果，訓練誤差（training error）為 0.5899826233860157，感覺在訓練集上的表現不是很好，存在一定程度的誤差，而且交叉驗證的平均誤差(cross-validation errors）比訓練誤差（training error）要大，這表示模型可能過度擬合了訓練資料。

要求 3:

下圖 3.1 為要求 3 的 code。

1. 前面 1~10 行都與要求 1 相同，不多加贅述。

2. 創建了一個函數 create_design_matrix，該函數接受 x 軸數據和多項式的次數 degree 作為輸入。它通過將 x 的各次方加入到一個數組中來創建一個矩陣 X，使其成為一個多項式，函式的第一行創建一個形狀為(len(x),1)的矩陣 X，每個元素都初始化為 1，這是因為任何自變量的零次冪都是 1。接下來的 for 循環從 1 迭代到 degree，每次將一個新的冪次向量 x.reshape(-1,1)**i 添加到 X 的左側，以形成新的設計矩陣。最後，函式返回設計矩陣 X。。

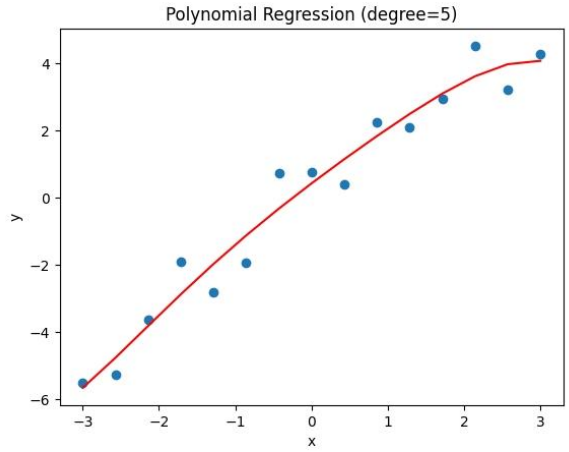
3. 使用 5 倍交叉驗證來計算模型的平均交叉驗證誤差。為此，我們將數據分為五個不同的集合，每個集合用於訓練模型的不同版本。然後，我們使用這些不同版本來計算交叉驗證誤差，這是指使用不同數據子集訓練模型並測試其在不同數據子集上的表現。

4. 使用 matplotlib 繪製數據和模型的擬合線，將模型的訓練誤差和交叉驗證誤差印出。

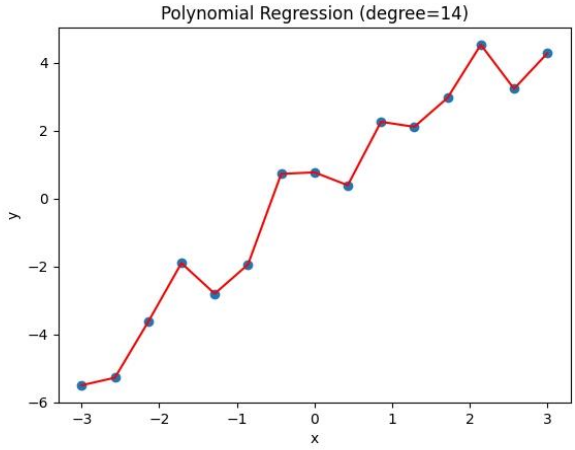
```python
#Requirement 3
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold

# Generate the data
np.random.seed(42)
x = np.linspace(-3, 3, num=15)
noise = np.random.normal(loc=0, scale=1, size=15)
y = 2 * x + noise

# Create the design matrix for polynomial regression
def create_design_matrix(x, degree):
        X = np.ones((len(x), 1))
        for i in range(1, degree+1):
                X = np.hstack((x.reshape(-1,1)**i,X))
                #print(X)
        return X

# Perform polynomial regression for degrees 5, 10, and 14
degrees = [5, 10, 14]
for degree in degrees:
        X = create_design_matrix(x, degree)

        # Compute the weight vector using the normal equation
        W = np.linalg.inv(X.T @ X) @ X.T @ y

        # Compute the predicted values and the training error
        y_pred_train = X @ W
        mse_train = np.mean((y - y_pred_train)**2)

        # Compute the five-fold cross-validation errors
        kf = KFold(n_splits=5, shuffle=True, random_state=42)
        mse_cv = []
        for train_index, test_index in kf.split(X):
                X_train, X_test = X[train_index], X[test_index]
                y_train, y_test = y[train_index], y[test_index]
                W_cv = np.linalg.inv(X_train.T @ X_train) @ X_train.T @ y_train
                y_pred_cv = X_test @ W_cv
                mse_cv.append(np.mean((y_test - y_pred_cv)**2))

        # Plot the fitting line and the data
        plt.scatter(x, y)
        plt.plot(x, y_pred_train, color='red')
        plt.title('Polynomial Regression (degree={})'.format(degree))
        plt.xlabel('x')
        plt.ylabel('y')
        plt.show()

        # Print the training error and the cross-validation errors
        print('Training error (degree={}): {}'.format(degree, mse_train))
        print('Cross-validation errors (degree={}): {}'.format(degree, mse_cv))
        print("\n")
```

▲圖 3.1

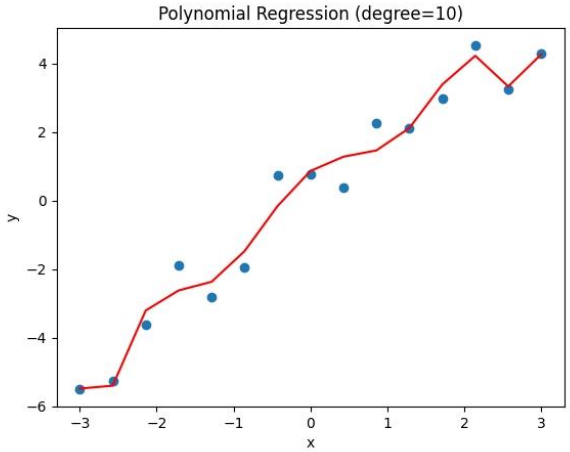
下圖 3.2 為要求 3 的執行結果:




```
Training error (degree=5): 0.4131074093617368
Cross-validation errors (degree=5): [1.3027950844371305, 1.4754684497517088, 1.464826352207493, 0.46432560015532703, 2.3611186816936254]
```




```
Training error (degree=14): 7.49195250613944e-12
Cross-validation errors (degree=14): [77945605.08393377, 7844.280466886881, 16110260.183059225, 1.6757993082741234, 8386.818920431358]
```




```
Training error (degree=10): 0.2415907284176918
Cross-validation errors (degree=10): [3592.7984800121717, 31.820851449712634, 31548.9423018607, 0.3109663883687654, 2.0309111007186798]
```

▲圖 3.2

要求 3 問題討論:

在一開始看到要實作 Polynomial Regression 的時候有腦袋卡一下，看到了 pdf 上的矩陣表示後才發現並不是很難實作，另外，他也可以直接拿 Sklearn 裡面的模型來直接使用。

另外，從上述輸出結果可以觀察到，隨著多項式回歸的次數增加，訓練誤差逐漸減小，但交叉驗證誤差卻不是一直減小，而是在某一個次數後開始增加，這就是過度擬合的現象。對於次數較低的多項式回歸（degree=5），訓練誤差和交叉驗證誤差都不算太高，這時模型可以較好地擬合數據，泛化能力較好。但是對於次數較高的多項式回歸（degree=10, 14），雖然訓練誤差很小，甚至趨近於 0，但是交叉驗證誤差卻很高，這時模型過度擬合，對新數據的泛化能力很差。

我認為這種現象的原因在於，當模型的複雜度過高時，模型會過度擬合訓練數據，從而導致模型對新數據的預測能力下降。此外，當模型的複雜度過高時，模型中的參數會變得非常多，使得模型容易產生噪聲，進而影響模型的預測能力。因此，對於多項式回歸模型，需要在模型複雜度和預測能力之間進行權衡，避免過度擬合的現象發生。

要求 4:

下圖 4.1 為要求 4 的 code。

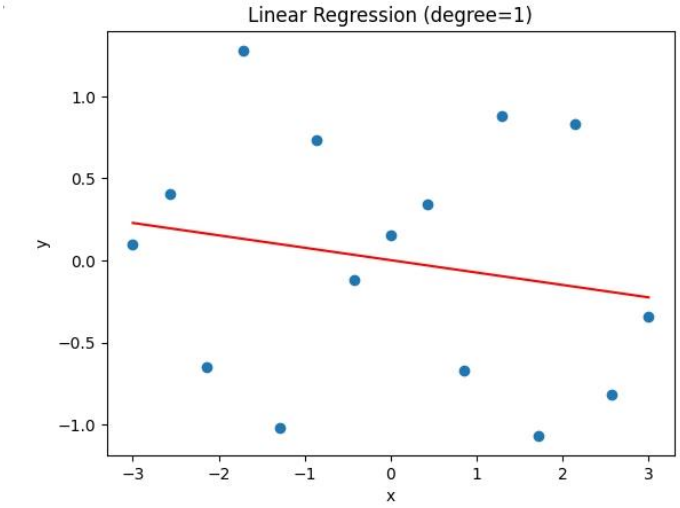
1. 與要求 1~3 線性函數不同，這裡使用非線性的函數 y=sin(2πx)+ε 作為樣本資料，其中 ε 是服從平均值為 0、標準差為 0.2 的常態分佈。
   在第 12 行修改成 y = np.sin(2 * np.pi * x) + noise

2. 其餘部分皆與要求 2 和 3 相同，唯一不同的是，題目要求比對 linear 和 polynomial regression 的結果，因此 degrees[]的部分多添加了 1 進去做觀察和比較 。

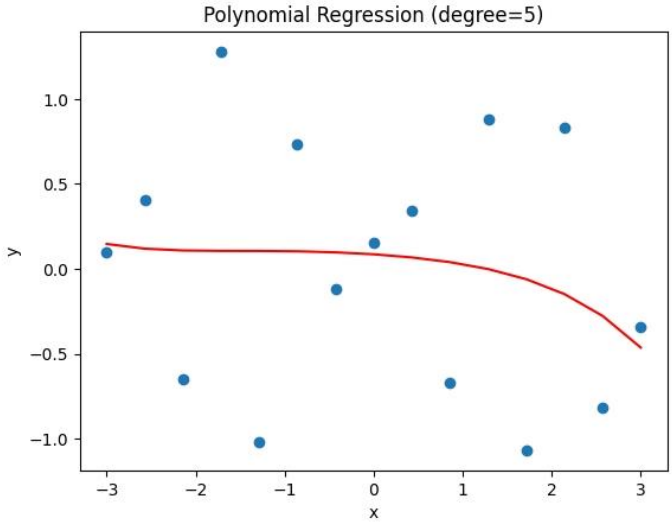
```
1 #Requirement 4
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from sklearn.model_selection import KFold
5
6 # Generate the data
7 np.random.seed(42)
8 x = np.linspace(-3, 3, num=15)
9
10 #y = sin(2πx) +ε with the noise ε~ N(0, 0.04)
11 noise = np.random.normal(loc=0, scale=0.2, size=15)
12 y = np.sin(2 * np.pi * x) + noise
13
14
15 # Create the design matrix for polynomial regression
16 def create_design_matrix(x, degree):
17         X = np.ones((len(x), 1))
18         for i in range(1, degree+1):
19                 X = np.hstack((x.reshape(-1,1)**i,X))
20                 #print(X)
21         return X
22
23 # Perform polynomial regression for degrees 5, 10, and 14
24 degrees = [1, 5, 10, 14]
25 for degree in degrees:
26         X = create_design_matrix(x, degree)
27
28         # Compute the weight vector using the normal equation
29         W = np.linalg.inv(X.T @ X) @ X.T @ y
30
31         # Compute the predicted values and the training error
32         y_pred_train = X @ W
33         mse_train = np.mean((y - y_pred_train)**2)
34
35         # Compute the five-fold cross-validation errors
36         kf = KFold(n_splits=5, shuffle=True, random_state=42)
37         mse_cv = []
38         for train_index, test_index in kf.split(X):
39                 X_train, X_test = X[train_index], X[test_index]
40                 y_train, y_test = y[train_index], y[test_index]
41                 W_cv = np.linalg.inv(X_train.T @ X_train) @ X_train.T @ y_train
42                 y_pred_cv = X_test @ W_cv
43                 mse_cv.append(np.mean((y_test - y_pred_cv)**2))
44
45         # Plot the fitting line and the data
46         plt.scatter(x, y)
47         plt.plot(x, y_pred_train, color='red')
48         if(degree==1):
49             plt.title('Linear Regression (degree={})'.format(degree))
50         else:
51             plt.title('Polynomial Regression (degree={})'.format(degree))
52         plt.xlabel('x')
53         plt.ylabel('y')
54         plt.show()
55
56         # Print the training error and the cross-validation errors
57         print('Training error (degree={}): {}'.format(degree, mse_train))
58         print('Cross-validation errors (degree={}): {}'.format(degree, mse_cv))
59         print("\n")
```

▲圖 4.1

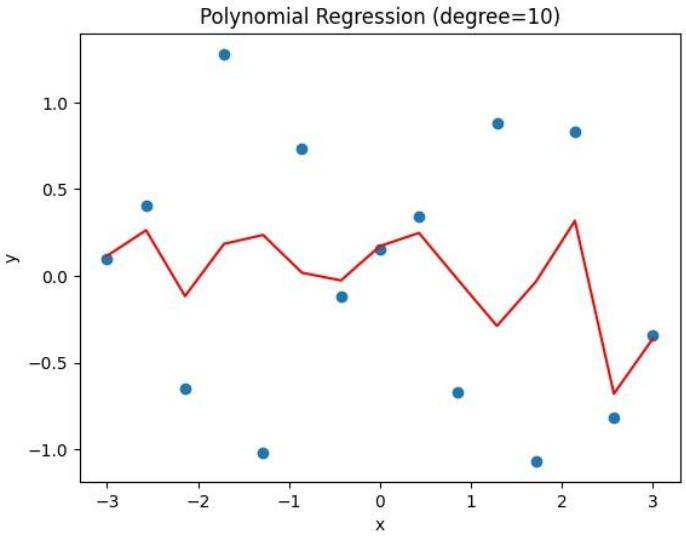
下圖 4.2 為要求 4 的執行結果:



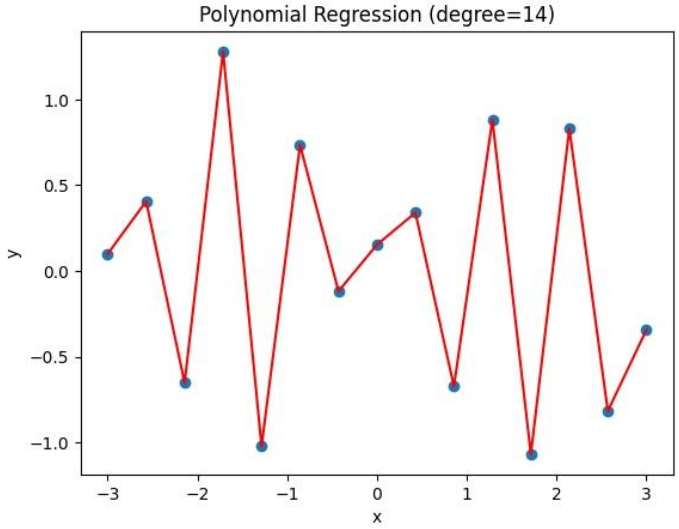

```
Training error (degree=1): 0.5018970543254088
Cross-validation errors (degree=1): [0.6561265733420149, 0.39592413855889436, 0.3034018243724104, 0.8492598315454186, 1.0209398531347447]
```




```
Training error (degree=5): 0.4934658473431223
Cross-validation errors (degree=5): [1.7422295715087044, 0.6430365726540999, 0.9204610195276764, 1.6498636506572852, 1.9858125983455022]
```




```
Training error (degree=10): 0.4510987165456404
Cross-validation errors (degree=10): [21781.48891730714, 399.9887523061586, 131888.3640710161, 5.166142236688748, 10.564750583238393]
```




```
Training error (degree=14): 6.74751768848857e-13
Cross-validation errors (degree=14): [476177727.9688049, 1283.9706317194766, 21987429.04661108, 5.473284319744681, 7666.560701332703]
```

▲圖 4.2

要求 4 問題討論:

由以上圖 4.2 執行結果可知，當 degree 為 1 時，多項式回歸模型的訓練誤差約為 0.50，交叉驗證誤差約為 0.57~1.02。當 degree 為 5 時，多項式回歸模型的訓練誤差約為 0.49，交叉驗證誤差約為 0.64~1.99。當 degree 為 10 時，多項式回歸模型的訓練誤差約為 0.45，交叉驗證誤差約為 5.17~2.18x10^5。當 degree 為 14 時，多項式回歸模型的訓練誤差為 6.75x10^-13，交叉驗證誤差約為 5.47~4.76x10^8。可以發現，當 degree=1 時，線性回歸的表現比多項式回歸好，雖然 training error 略大於多項式回歸，但看到 cross-validation errors 相對多項式回歸就小非常多。

隨著 degree 的增加，訓練誤差逐漸降低，而交叉驗證誤差則先下降後上升。當 degree 為 14 時，雖然模型的訓練誤差最小，但交叉驗證誤差反而變得非常大，這表示模型出現了過度擬合現象，對新的未見過的資料的預測能力變得非常差。因此，該選擇 degree 的適當值非常的重要，以避免過度擬合或欠擬合的問題。

要求 5:

下圖 5.1 為要求 5 的 code。

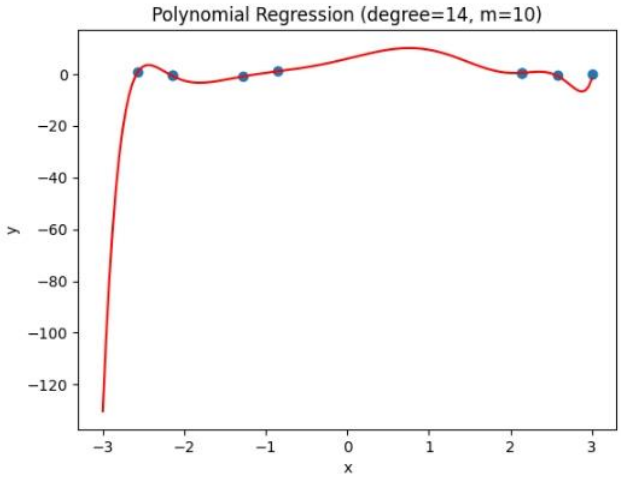
1. 在第 8 行定義 m_values = [10, 80, 320]是為了題目要求以不同的訓練數據點數量，並在 26 行這裡必須把 degrees 固定為 14，然後在第 27~28 行使用迴圈來執行不同的次方次數(degree)和不同的訓練資料筆數(m)的多項式迴歸，接下來 30 行使用 np.random.choice() 函數從 num 個數據點中隨機選取 m 個數據點的索引，並使用這些索引從 x 和 y 中選取對應的 m 個數據點，作為本次多項式迴歸的訓練數據集，之後呼叫 create_design_matrix() 函數將訓練數據集轉換為設計矩陣，接下來的部分基本上與要求 4 相同，在此不再贅述。

```
#Requirement 5

import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold

# Define the number of training data points
m_values = [10, 80, 320]
num = 15

# Generate the data
#np.random.seed(42)
#indices = np.random.choice(100, num, replace=False)
x = np.linspace(-3, 3, num=num)
noise = np.random.normal(loc=0, scale=0.2, size=num)
y = np.sin(2 * np.pi * x) + noise

# Create the design matrix for polynomial regression
def create_design_matrix(x, degree):
       X = np.ones((len(x), 1))
       for i in range(1, degree+1):
              X = np.hstack((x.reshape(-1,1)**i,X))
       return X

# Perform polynomial regression for degree 14 and vary the number of training data points
degrees = [14]
```

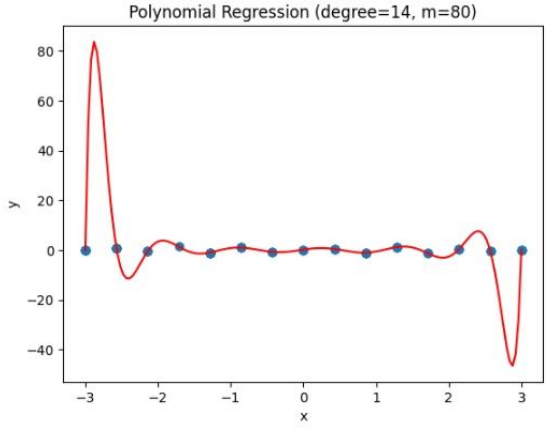
```
for degree in degrees:
       for m in m_values:
              # Randomly select m training data points
              indices = np.random.choice(num, m, replace=True)
              x_train, y_train = x[indices], y[indices]
              X_train = create_design_matrix(x_train, degree)

              # Compute the weight vector using the normal equation
              W = np.linalg.inv(X_train.T @ X_train) @ X_train.T @ y_train

              # Compute the predicted values and the training error
              y_pred_train = X_train @ W
              mse_train = np.mean((y_train - y_pred_train)**2)

              # Compute the five-fold cross-validation errors
              kf = KFold(n_splits=5, shuffle=True, random_state=42)
              mse_cv = []
              for train_index, test_index in kf.split(X_train):
                     X_train_cv, X_test_cv = X_train[train_index], X_train[test_index]
                     y_train_cv, y_test_cv = y_train[train_index], y_train[test_index]
                     W_cv = np.linalg.inv(X_train_cv.T @ X_train_cv) @ X_train_cv.T @ y_train_cv
                     y_pred_cv = X_test_cv @ W_cv
                     mse_cv.append(np.mean((y_test_cv - y_pred_cv)**2))

              # Plot the fitting line and the data
              plt.scatter(x_train, y_train)
              x_plot = np.linspace(-3, 3, num=num*10) #為了畫出平滑畫曲線
              X_plot = create_design_matrix(x_plot, degree)
              y_plot = X_plot @ W
              plt.plot(x_plot, y_plot, color='red')
              plt.title('Polynomial Regression (degree={}, m={})'.format(degree, m))
              plt.xlabel('x')
              plt.ylabel('y')
              plt.show()

              # Print the training error and the cross-validation errors
              print('Training error (degree={}, m={}): {}'.format(degree, m, mse_train))
              print('Cross-validation errors (degree={}, m={}): {}'.format(degree, m, mse_cv))
              print("\n")
```

▲圖 5.1

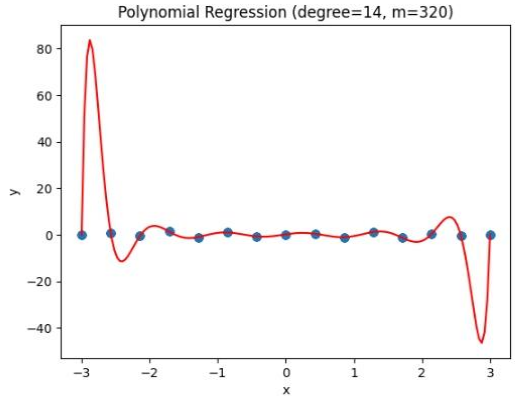
下圖 5.2 為要求 5 的執行結果:




```
Training error (degree=14, m=10): 0.37606378525229206
Cross-validation errors (degree=14, m=10): [0.8267215866286812, 0.5679213852830475, 2.0080476982653366, 363961621.038037, 48558.91463630818]
```




```
Training error (degree=14, m=80): 2.6716044696951904e-11
Cross-validation errors (degree=14, m=80): [5.4141219584534346e-11, 6.7736170788506e-11, 5.320359823140248 4e-12, 1.6881588567069829e-13, 5.9060142561137905]
```




```
Training error (degree=14, m=320): 3.694294202738933e-11
Cross-validation errors (degree=14, m=320): [1.0306388393141348e-10, 1.0103206331839158e-10, 5.649156152878312e-11, 3.7881707752820726e-11, 1.2070245455072027e-11]
```

▲圖 5.2

要求 5 問題討論:

由上圖 5.2 結果可知，這個結果顯示了隨著 m 增加，訓練誤差和交叉驗證誤差都趨於穩定，這是因為隨著 m 的增加，我們使用的樣本數增加，使得我們可以更好地捕捉到數據的真實變化。

當 m=10 時，訓練誤差為 0.37606378525229206，交叉驗證誤差則在第 4 和第 5 中非常高，分別為 363961621.038037 和 48558.91463630818。這表示當數據量太少時，模型很容易過度擬合訓練集，因此在未見過的數據上表現較差。

當 m=80 和 m=320 時，訓練誤差非常小，分別為 2.6716044696951904e-11 和 3.694294202738933e-11。交叉驗證誤差也顯著降低，並且比 m=10 時的結果要好得多。這表明當數據量增加時，模型可以更好地泛化到未見過的數據，並且不容易過度擬合訓練集。

此外，再回到**要求 4** 去查看，當使用更高的多項式次數時，訓練誤差會優於交叉驗證誤差，這是因為使用更高的多項式次數會使模型變得更複雜，更容易對訓練數據過度擬合，而無法泛化到新的數據。因此，當多項式次數過高時，模型的泛化性能將會變得較差，這就是訓練誤差和交叉驗證誤差之間差異變大的原因。

要求 6:

下圖 6.1 為要求 6 的 code。

1. 這段程式碼是由**要求 4** 做延伸，使用正則化（regularization）的方法，對一個具有 14 次多項式回歸（polynomial regression）的模型進行訓練。透過調整正則化參數 λ 的大小（第 24 行，分別設為 0、0.001/m、1/m、1000/m），來比較不同正則化強度對模型的影響。其中，m 為樣本數，本例中為 15 個。

2. 再來與**要求** 4 不同的地方為第 33 行 W 的計算方式，這裡採用的是"ridge regression"，它是一種正規化線性回歸的方法，可以有效解決過度擬合(overfitting)的問題。它的公式為：W = (X.T @ X + λ * I)^(-1) @ X.T @ y。

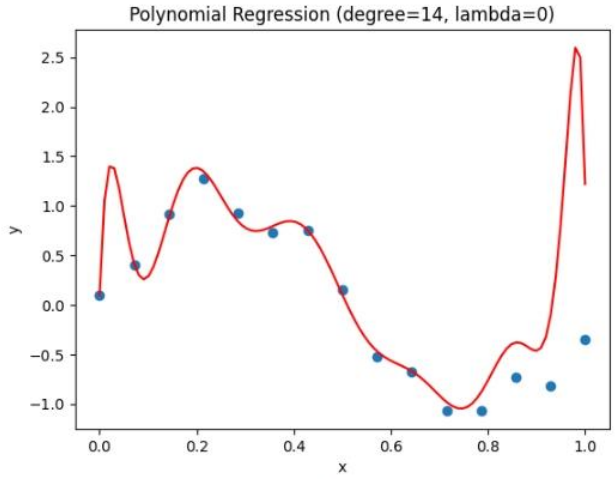
3. 接下來的步驟都相同，利用 K-fold 交叉驗證（K-Fold Cross Validation）來計算五次驗證集的均方誤差（MSE），從而評估模型的泛化能力。最後，根據模型的訓練結果，繪製相關的圖和多項式回歸的線。

```python
#Requirement 6 # 使用正則化可以幫助減少模型的過度擬合 (overfitting) 問題
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold

# Define the number of training data points
m = 15

# Generate the data
np.random.seed(42)
x = np.linspace(0, 1, num=m)
noise = np.random.normal(loc=0, scale=0.2, size=m)
y = np.sin(2 * np.pi * x) + noise

# Create the design matrix for polynomial regression
def create_design_matrix(x, degree):
        X = np.ones((len(x), 1))
        for i in range(1, degree+1):
                X = np.hstack((x.reshape(-1,1)**i,X))
        return X

# Define the degrees and regularization parameters
degrees = [14]
lambdas = [0, 0.001/m, 1/m, 1000/m]
```

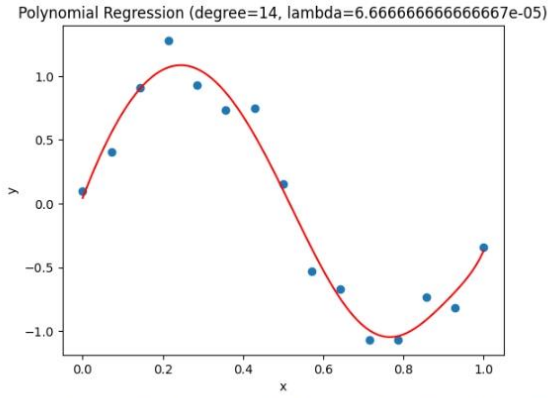
```python

# Define the degrees and regularization parameters
degrees = [14]
lambdas = [0, 0.001/m, 1/m, 1000/m]

# Loop over the degrees and regularization parameters
for degree in degrees:
      for lambda_val in lambdas:
            # Create the design matrix
            X = create_design_matrix(x, degree)

            # Compute the weight vector using "ridge regression"
            W = np.linalg.inv(X.T @ X + lambda_val * np.identity(degree+1)) @ X.T @ y

            # Compute the five-fold cross-validation errors
            kf = KFold(n_splits=5, shuffle=True, random_state=42)
            mse_cv = []
            for train_index, test_index in kf.split(X):
                  X_train_cv, X_test_cv = X[train_index], X[test_index]
                  y_train_cv, y_test_cv = y[train_index], y[test_index]
                  W_cv = np.linalg.inv(X_train_cv.T @ X_train_cv + lambda_val * np.identity(degree+1)) @ X_train_cv.T @ y_train_cv
                  y_pred_cv = X_test_cv @ W_cv
                  mse_cv.append(np.mean((y_test_cv - y_pred_cv)**2))

            # Plot the fitting line and the data
            plt.scatter(x, y)
            x_plot = np.linspace(0, 1, num=100)
            X_plot = create_design_matrix(x_plot, degree)
            y_plot = X_plot @ W
            plt.plot(x_plot, y_plot, color='red')
            plt.title('Polynomial Regression (degree={}, lambda={})'.format(degree, lambda_val))
            plt.xlabel('x')
            plt.ylabel('y')
            plt.show()

            # Print the cross-validation errors
            print('Cross-validation errors (degree={}, lambda={}): {}'.format(degree, lambda_val, mse_cv))
            print("\n")
```

▲圖 6.1

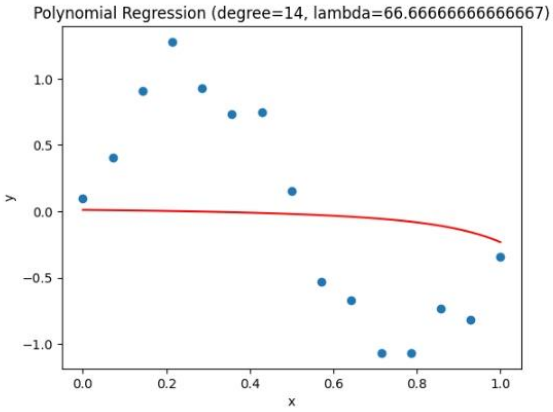
下圖 6.2 為要求 6 的執行結果:



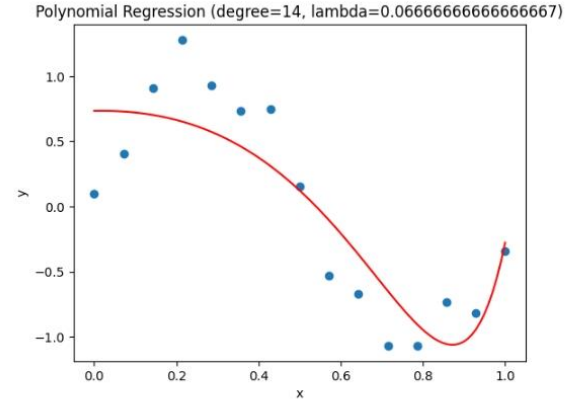
Cross-validation errors (degree=14, lambda=0): [48.31465162839259, 81.25297133025838, 35673.57605268293, 0.20034660134456603, 1.082948748821802]



Cross-validation errors (degree=14, lambda=6.666666666666667e-05): [0.02466172140856922, 0.16977939482026985, 2.209767516593763, 0.012972387506546293, 0.10004041654933112]



Cross-validation errors (degree=14, lambda=66.66666666666667): [0.505181504392254, 0.43368926302853367, 0.34456439000434186, 0.6436076934214195, 0.8905367072871458]



Cross-validation errors (degree=14, lambda=0.06666666666666667): [0.3414482819088513, 0.11114912316618564, 0.2045812295087921, 0.1376413793372324, 0.37360611711014907]

▲圖 6.2

要求 6 問題討論:

由上圖 6.2 觀察得知，當 lambda 為 0 時，模型沒有進行正則化，結果出現了明顯的過度擬合，因為交叉驗證誤差很大。當 lambda 增加到 0.001/m 時，交叉驗證誤差明顯減小，表示正則化有助於減少過擬合。當 lambda 增加到 1/m 時，交叉驗證誤差繼續減小，但是當 lambda 增加到 1000/m 時，交叉驗證誤差又有所增加，這可能是因為太強的正則化導致欠擬合的原因。因此得知，可以根據交叉驗證誤差選擇合適的 lambda 值來平衡擬合效果和泛化性能。