作業概述：

本資料集的大小為 32561 rows × 15 columns，然後我們想要去預測的目標是 income。而由於 income 的結果只有(<=50K 或>50K 兩種)，屬於分類問題，因此採用 bagging 中的隨機森林演算法，選擇 10-fold 進行預測(不可使用套件)，並將兩者放入一個函示中方便呼叫使用。

程式碼：

資料前處理

```
# 導入套件
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from collections import Counter
```

```
# 讀入作業的csv檔並存到df
df = pd.read_csv('HW2data.csv')
df
```

	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relationship	race	sex	capital_gain	capital_loss	hours_per_week
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40
...	...	...	...	...	...	...	...	...	...	...	...	...	...
32556	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife	White	Female	0	0	38
32557	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	0	0	40
32558	58	Private	151910	HS-grad	9	Widowed	Adm-clerical	Unmarried	White	Female	0	0	40
32559	22	Private	201490	HS-grad	9	Never-married	Adm-clerical	Own-child	White	Male	0	0	20
32560	52	Self-emp-inc	287927	HS-grad	9	Married-civ-spouse	Exec-managerial	Wife	White	Female	15024	0	40

32561 rows × 15 columns

```
# 看一下df的資料型態以及有無空值
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   age             32561 non-null  int64 
 1   workclass       32561 non-null  object
 2   fnlwgt          32561 non-null  int64 
 3   education       32561 non-null  object
 4   education_num   32561 non-null  int64 
 5   marital_status  32561 non-null  object
 6   occupation      32561 non-null  object
 7   relationship    32561 non-null  object
 8   race            32561 non-null  object
 9   sex             32561 non-null  object
 10  capital_gain    32561 non-null  int64 
 11  capital_loss    32561 non-null  int64 
 12  hours_per_week  32561 non-null  int64 
 13  native_country  32561 non-null  object
 14  income          32561 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

```
# 取df中income以外的欄位並存到df1
df1 = df.iloc[:, 0:-1]
df1
```

	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relationship	race	sex	capital_gain	capital_loss	hours_per_week
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40
...	...	...	...	...	...	...	...	...	...	...	...	...	...
32556	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife	White	Female	0	0	38
32557	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	0	0	40
32558	58	Private	151910	HS-grad	9	Widowed	Adm-clerical	Unmarried	White	Female	0	0	40
32559	22	Private	201490	HS-grad	9	Never-married	Adm-clerical	Own-child	White	Male	0	0	20
32560	52	Self-emp-inc	287927	HS-grad	9	Married-civ-spouse	Exec-managerial	Wife	White	Female	15024	0	40

32561 rows × 14 columns

```python
# 取出df中的income欄位並存到Y
Y = df.iloc[:, -1]
Y
```

```
0         <=50K
1         <=50K
2         <=50K
3         <=50K
4         <=50K
          ...
32556     <=50K
32557      >50K
32558     <=50K
32559     <=50K
32560      >50K
Name: income, Length: 32561, dtype: object
```

```python
# 看一下df1有哪些特徵
df1.columns
```

```
Index(['age', 'workclass', 'fnlwgt', 'education', 'education_num',
       'marital_status', 'occupation', 'relationship', 'race', 'sex',
       'capital_gain', 'capital_loss', 'hours_per_week', 'native_country'],
      dtype='object')
```

```python
# 看一下df1中'age'和 'workclass'這一行內勺內的資料型態
print(df1['age'].dtypes)
print(df1['workclass'].dtypes)
```

```
int64
object
```

```python
# 試試看能不能使用這個判斷式，因為想用迴圈取出需要做預處理的特徵
df1['workclass'].dtypes == 'object'
```

```
True
```

```python
# 取出需要做預處理的特徵
unprocess_feature = []
for i in df1.columns:
    if df1[i].dtypes == 'object':
        unprocess_feature.append(i)
print(unprocess_feature)
```

```
['workclass', 'education', 'marital_status', 'occupation', 'relationship', 'race', 'sex', 'native_country']
```

```python
# 看一下需要做預處理的特徵之內容
for i in unprocess_feature:
    print(f'{i}這一欄')
    print(Counter(df1[i]), end='\n\n')
```

```
workclass這一欄
Counter({' Private': 22696, ' Self-emp-not-inc': 2541, ' Local-gov': 2093, ' ?': 1836, ' State-gov': 1298, ' Self-emp-inc': 1
116, ' Federal-gov': 960, ' Without-pay': 14, ' Never-worked': 7})

education這一欄
Counter({' HS-grad': 10501, ' Some-college': 7291, ' Bachelors': 5355, ' Masters': 1723, ' Assoc-voc': 1382, ' 11th': 1175, '
Assoc-acdm': 1067, ' 10th': 933, ' 7th-8th': 646, ' Prof-school': 576, ' 9th': 514, ' 12th': 433, ' Doctorate': 413, ' 5th-6t
h': 333, ' 1st-4th': 168, ' Preschool': 51})

marital_status這一欄
Counter({' Married-civ-spouse': 14976, ' Never-married': 10683, ' Divorced': 4443, ' Separated': 1025, ' Widowed': 993, ' Mar
ried-spouse-absent': 418, ' Married-AF-spouse': 23})

occupation這一欄
Counter({' Prof-specialty': 4140, ' Craft-repair': 4099, ' Exec-managerial': 4066, ' Adm-clerical': 3770, ' Sales': 3650, ' O
ther-service': 3295, ' Machine-op-inspct': 2002, ' ?': 1843, ' Transport-moving': 1597, ' Handlers-cleaners': 1370, ' Farming
-fishing': 994, ' Tech-support': 928, ' Protective-serv': 649, ' Priv-house-serv': 149, ' Armed-Forces': 9})

relationship這一欄
Counter({' Husband': 13193, ' Not-in-family': 8305, ' Own-child': 5068, ' Unmarried': 3446, ' Wife': 1568, ' Other-relative':
```

```
# 取出不需要做預處理的特徵
# 取出特徵下的內容存成X0
process_feature = []
for i in df1.columns:
    if df1[i].dtypes == np.int64:
        process_feature.append(i)
print(process_feature)

X0 = df1[process_feature]
X0
```

```
['age', 'fnlwgt', 'education_num', 'capital_gain', 'capital_loss', 'hours_per_week']
```

	age	fnlwgt	education_num	capital_gain	capital_loss	hours_per_week
0	39	77516	13	2174	0	40
1	50	83311	13	0	0	13
2	38	215646	9	0	0	40
3	53	234721	7	0	0	40
4	28	338409	13	0	0	40
...	...	...	...	...	...	...
32556	27	257302	12	0	0	38
32557	40	154374	9	0	0	40
32558	58	151910	9	0	0	40
32559	22	201490	9	0	0	20
32560	52	287927	9	15024	0	40

32561 rows × 6 columns

```
# 從df1取出age這個欄位，方便合併表格，之後會刪除(因為會重複)
X_temp1 = df1['age']
# 做one-hot-encoding，並將結果存到X_temp1
for i in unprocess_feature:
    X_temp2 = pd.get_dummies(df1[i], prefix=i)
    X_temp1 = pd.concat([X_temp1, X_temp2], axis=1)
X_temp1
```

	age	workclass_ ?	workclass_ Federal-gov	workclass_ Local-gov	workclass_ Never-worked	workclass_ Private	workclass_ Self-emp-inc	workclass_ Self-emp-not-inc	workclass_ State-gov	workclass_ Without-pay	...	native_country_ Portugal	native_countr Puerto-Ri
0	39	0	0	0	0	0	0	0	1	0	...	0	
1	50	0	0	0	0	0	0	1	0	0	...	0	
2	38	0	0	0	0	1	0	0	0	0	...	0	
3	53	0	0	0	0	1	0	0	0	0	...	0	
4	28	0	0	0	0	1	0	0	0	0	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
32556	27	0	0	0	0	1	0	0	0	0	...	0	
32557	40	0	0	0	0	1	0	0	0	0	...	0	
32558	58	0	0	0	0	1	0	0	0	0	...	0	
32559	22	0	0	0	0	1	0	0	0	0	...	0	
32560	52	0	0	0	0	0	1	0	0	0	...	0	

32561 rows × 103 columns

```
# 將處理好的特徵做合併，接著存到X
X = pd.concat([X0, X_temp1.iloc[:, 1:]], axis=1)
X
```

	age	fnlwgt	education_num	capital_gain	capital_loss	hours_per_week	workclass_ ?	workclass_ Federal-gov	workclass_ Local-gov	workclass_ Never-worked	...	native_country_ Portugal	native_ Pue
0	39	77516	13	2174	0	40	0	0	0	0	...	0	
1	50	83311	13	0	0	13	0	0	0	0	...	0	
2	38	215646	9	0	0	40	0	0	0	0	...	0	
3	53	234721	7	0	0	40	0	0	0	0	...	0	
4	28	338409	13	0	0	40	0	0	0	0	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
32556	27	257302	12	0	0	38	0	0	0	0	...	0	
32557	40	154374	9	0	0	40	0	0	0	0	...	0	
32558	58	151910	9	0	0	40	0	0	0	0	...	0	
32559	22	201490	9	0	0	20	0	0	0	0	...	0	
32560	52	287927	9	15024	0	40	0	0	0	0	...	0	

32561 rows × 108 columns

```
# 看一下X的資料型態，發現已無Object型態，代表處理成功
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Columns: 108 entries, age to native_country_ Yugoslavia
dtypes: int64(6), uint8(102)
memory usage: 4.7 MB
```

```
# 將X, Y做合併存到XY，等等要放入我們做好的函示中
XY = pd.concat([X, Y], axis=1)
XY
```

	age	fnlwgt	education_num	capital_gain	capital_loss	hours_per_week	workclass_ ?	workclass_ Federal-gov	workclass_ Local-gov	workclass_ Never-worked	...	native_country_ Puerto-Rico	native_
0	39	77516	13	2174	0	40	0	0	0	0	...	0	
1	50	83311	13	0	0	13	0	0	0	0	...	0	
2	38	215646	9	0	0	40	0	0	0	0	...	0	
3	53	234721	7	0	0	40	0	0	0	0	...	0	
4	28	338409	13	0	0	40	0	0	0	0	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
32556	27	257302	12	0	0	38	0	0	0	0	...	0	
32557	40	154374	9	0	0	40	0	0	0	0	...	0	
32558	58	151910	9	0	0	40	0	0	0	0	...	0	
32559	22	201490	9	0	0	20	0	0	0	0	...	0	
32560	52	287927	9	15024	0	40	0	0	0	0	...	0	

32561 rows × 109 columns

k-fold cross-validation

```
# 建立k-fold cross-validation的函示，並用print(XXXX.index)來看看使用出錯
def K_fold_and_RF(data, num):
    from sklearn.ensemble import RandomForestClassifier
    clf = RandomForestClassifier(max_depth=10)
    score = []
    s = int(len(data)/num)
    for i in range(1, num+1):
        train_index = []
        X_test = data.iloc[(i-1)*s:(i*s-1)]
        X_test = X_test.drop(['income'], axis=1)
        Y_test = data['income'].iloc[X_test.index]
        for i in data.index:
            if i not in X_test.index:
                train_index.append(i)
        X_train = data.iloc[train_index]
        X_train = X_train.drop(['income'], axis=1)
        Y_train = data['income'].iloc[train_index]

        print(X_train.index)
        print(X_test.index)
        print(Y_train.index)
        print(Y_test.index)
        print('--------------------------------------------------------------------------------')
```

```
# 舉3-fold為例子，發現運作正確
K_fold_and_RF(XY, 3)
```

```
Int64Index([    0,     1,     2,     3,     4,     5,     6,     7,     8,
                9,
            ...
            10842, 10843, 10844, 10845, 10846, 10847, 10848, 10849, 10850,
            10851],
           dtype='int64', length=10852)
--------------------------------------------------------------------------------
Int64Index([    0,     1,     2,     3,     4,     5,     6,     7,     8,
                9,
            ...
            32551, 32552, 32553, 32554, 32555, 32556, 32557, 32558, 32559,
            32560],
           dtype='int64', length=21709)
RangeIndex(start=10853, stop=21705, step=1)
Int64Index([    0,     1,     2,     3,     4,     5,     6,     7,     8,
                9,
            ...
            32551, 32552, 32553, 32554, 32555, 32556, 32557, 32558, 32559,
            32560],
           dtype='int64', length=21709)
```

k-fold cross-validation&隨機森林

```python
# 延長剛剛的函示，放入隨機森林(最大深度限制為10層)，並用迴圈算出每次的準確率然後求平均
def K_fold_and_RF(data, num):
    from sklearn.ensemble import RandomForestClassifier
    clf = RandomForestClassifier(max_depth=10)
    scores = []
    sumofscores = 0
    s = int(len(data)/num)
    for i in range(1, num+1):
        train_index = []
        X_test = data.iloc[(i-1)*s:(i*s-1)]
        X_test = X_test.drop(['income'], axis=1)
        Y_test = data['income'].iloc[X_test.index]
        for i in data.index:
            if i not in X_test.index:
                train_index.append(i)
        X_train = data.iloc[train_index]
        X_train = X_train.drop(['income'], axis=1)
        Y_train = data['income'].iloc[train_index]

        clf.fit(X_train, Y_train)
        scores.append(clf.score(X_test, Y_test))

    print(f'這{num}次跑出來的準確率分別為：\n{scores}')
    for i in scores:
        sumofscores += i
    print('算出來的平均準確率為{}'.format(sumofscores/num))
```

```python
# 計算作業所要求的10-fold的準確率
K_fold_and_RF(XY, 10)
```

```
這10次跑出來的準確率分別為：
[0.8423963133640553, 0.8620583717357911, 0.8651305683563748, 0.8454685099846391, 0.8620583717357911, 0.8571428571428571, 0.8522273425499232, 0.8632872503840245, 0.8623655913978494, 0.8522273425499232]
算出來的平均準確率為0.8564362519201227
```

結果分析：

分析出來的準確率大概是八成五左右，寫此 code 當中最大的問題也就是需要手刻 k-fold，我也覺得這是一個很好的練習，雖然需要花費更多時間，但透過手刻可以對演算法的運算更加的熟悉，而不是只會用而不知道背後的原理！