In [1]:

```
#Null (or Missing) Values Estimation [Please fill the values without highlight in the blank
#Data Balance [Only for “classification problem”, and please state the original and final a
#Feature Selection [Please state the original and final amount of features of each dataset]
#After finishing 1) 2) 3), please save your file to .xlsx for this assignment, and save to
```

In [2]:

```
#conda update scikit-learn
```

In [3]:

```
#pip install -U imbalanced-learn
```

In [4]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import classification_report
```

In [5]:

```
#load the data
data=pd.read_csv("bankruptcy.csv")
```

In [6]:

```
#realize the shape
data.shape
```

Out[6]:

```
(6819, 97)
```

In [7]:

```
#glace data values
#總共有97個col，前兩個分別為編號和是否破產，剩下95個為feature數量
data.head()
```

Out[7]:

	Unnamed: 0	Bankrupt?	ROA(C) before interest and depreciation before interest	ROA(A) before interest and % after tax	ROA(B) before interest and depreciation after tax	Operating Gross Margin	Realized Sales Gross Margin	Operating Profit Rate
0	0	1	0.370594	0.424389	0.405750	0.601457	0.601457	0.998969
1	1	1	0.464291	0.538214	0.516730	0.610235	0.610235	0.998946
2	2	1	0.426071	0.499019	0.472295	0.601450	0.601364	0.998857
3	3	1	0.399844	0.451265	0.457733	0.583541	0.583541	0.998700
4	4	1	0.465022	0.538432	0.522298	0.598783	0.598783	0.998973

5 rows × 97 columns

In [8]:

```
#checking null data
print(data.isnull().sum())
```

```
Unnamed: 0                                                  0
Bankrupt?                                                   0
 ROA(C) before interest and depreciation before interest   98
 ROA(A) before interest and % after tax                   100
 ROA(B) before interest and depreciation after tax         98
                                                         ...
 Liability to Equity                                      100
 Degree of Financial Leverage (DFL)                        99
 Interest Coverage Ratio (Interest expense to EBIT)        99
 Net Income Flag                                          100
 Equity to Liability                                      100
Length: 97, dtype: int64
```

In [9]:

```
#using 最後觀察值推估法(Last Observation Carried Forward(LOCF)) ffill& 下個觀察值推估法(Next C
#同時使用兩個方法，以防遺漏
#將處理完的data改名為data_fill
data_fill=data.fillna(method='ffill')
data_fill=data_fill.fillna(method='bfill')
data_fill
```

Out[9]:

	Unnamed: 0	Bankrupt?	ROA(C) before interest and depreciation before interest	ROA(A) before interest and % after tax	ROA(B) before interest and depreciation after tax	Operating Gross Margin	Realized Sales Gross Margin	Operatir Pro Ra
0	0	1	0.370594	0.424389	0.405750	0.601457	0.601457	0.99896
1	1	1	0.464291	0.538214	0.516730	0.610235	0.610235	0.99894
2	2	1	0.426071	0.499019	0.472295	0.601450	0.601364	0.99885
3	3	1	0.399844	0.451265	0.457733	0.583541	0.583541	0.99870
4	4	1	0.465022	0.538432	0.522298	0.598783	0.598783	0.99897
...	...	...	...	...	...	...	...	
6814	6814	0	0.493687	0.539468	0.543230	0.604455	0.604462	0.99899
6815	6815	0	0.475162	0.538269	0.524172	0.598308	0.598308	0.99899
6816	6816	0	0.472725	0.533744	0.520638	0.610444	0.610213	0.99898
6817	6817	0	0.506264	0.559911	0.554045	0.607850	0.607850	0.99907
6818	6818	0	0.493053	0.570105	0.549548	0.627409	0.627409	0.99808

6819 rows × 97 columns

In [10]:

```
#check null data
#not null data
print(data_fill.isnull().sum())
```

```
Unnamed: 0                                                 0
Bankrupt?                                                  0
 ROA(C) before interest and depreciation before interest   0
 ROA(A) before interest and % after tax                    0
 ROA(B) before interest and depreciation after tax         0
                                                          ..
 Liability to Equity                                       0
 Degree of Financial Leverage (DFL)                        0
 Interest Coverage Ratio (Interest expense to EBIT)        0
 Net Income Flag                                           0
 Equity to Liability                                       0
Length: 97, dtype: int64
```

In [11]:

```
data_fill.columns
```

Out[11]:

```
Index(['Unnamed: 0', 'Bankrupt?',
       ' ROA(C) before interest and depreciation before interest',
       ' ROA(A) before interest and % after tax',
       ' ROA(B) before interest and depreciation after tax',
       ' Operating Gross Margin', ' Realized Sales Gross Margin',
       ' Operating Profit Rate', ' Pre-tax net Interest Rate',
       ' After-tax net Interest Rate',
       ' Non-industry income and expenditure/revenue',
       ' Continuous interest rate (after tax)', ' Operating Expense Rat
e',
       ' Research and development expense rate', ' Cash flow rate',
       ' Interest-bearing debt interest rate', ' Tax rate (A)',
       ' Net Value Per Share (B)', ' Net Value Per Share (A)',
       ' Net Value Per Share (C)', ' Persistent EPS in the Last Four Seas
ons',
       ' Cash Flow Per Share', ' Revenue Per Share (Yuan ¥)',
       ' Operating Profit Per Share (Yuan ¥)',
       ' Per Share Net profit before tax (Yuan ¥)',
       ' Realized Sales Gross Profit Growth Rate',
       ' Operating Profit Growth Rate', ' After-tax Net Profit Growth Rat
e',
       ' Regular Net Profit Growth Rate', ' Continuous Net Profit Growth
Rate',
       ' Total Asset Growth Rate', ' Net Value Growth Rate',
       ' Total Asset Return Growth Rate Ratio', ' Cash Reinvestment %',
       ' Current Ratio', ' Quick Ratio', ' Interest Expense Ratio',
       ' Total debt/Total net worth', ' Debt ratio %', ' Net worth/Asset
s',
       ' Long-term fund suitability ratio (A)', ' Borrowing dependency',
       ' Contingent liabilities/Net worth',
       ' Operating profit/Paid-in capital',
       ' Net profit before tax/Paid-in capital',
       ' Inventory and accounts receivable/Net value', ' Total Asset Turn
over',
       ' Accounts Receivable Turnover', ' Average Collection Days',
       ' Inventory Turnover Rate (times)', ' Fixed Assets Turnover Freque
ncy',
       ' Net Worth Turnover Rate (times)', ' Revenue per person',
       ' Operating profit per person', ' Allocation rate per person',
       ' Working Capital to Total Assets', ' Quick Assets/Total Assets',
       ' Current Assets/Total Assets', ' Cash/Total Assets',
       ' Quick Assets/Current Liability', ' Cash/Current Liability',
       ' Current Liability to Assets', ' Operating Funds to Liability',
       ' Inventory/Working Capital', ' Inventory/Current Liability',
       ' Current Liabilities/Liability', ' Working Capital/Equity',
       ' Current Liabilities/Equity', ' Long-term Liability to Current As
sets',
       ' Retained Earnings to Total Assets', ' Total income/Total expens
e',
       ' Total expense/Assets', ' Current Asset Turnover Rate',
       ' Quick Asset Turnover Rate', ' Working capictal Turnover Rate',
       ' Cash Turnover Rate', ' Cash Flow to Sales', ' Fixed Assets to As
```

```
sets',
        ' Current Liability to Liability', ' Current Liability to Equity',
        ' Equity to Long-term Liability', ' Cash Flow to Total Assets',
        ' Cash Flow to Liability', ' CFO to Assets', ' Cash Flow to Equit
y',
        ' Current Liability to Current Assets', ' Liability-Assets Flag',
        ' Net Income to Total Assets', ' Total assets to GNP price',
        ' No-credit Interval', ' Gross Profit to Sales',
        ' Net Income to Stockholder's Equity', ' Liability to Equity',
        ' Degree of Financial Leverage (DFL)',
        ' Interest Coverage Ratio (Interest expense to EBIT)',
        ' Net Income Flag', ' Equity to Liability'],
      dtype='object')
```

In [12]:

```
#區分出預測值和feature
y=data_fill["Bankrupt?"]
x=data_fill.drop(["Unnamed: 0","Bankrupt?"],axis=1)
```

In [13]:

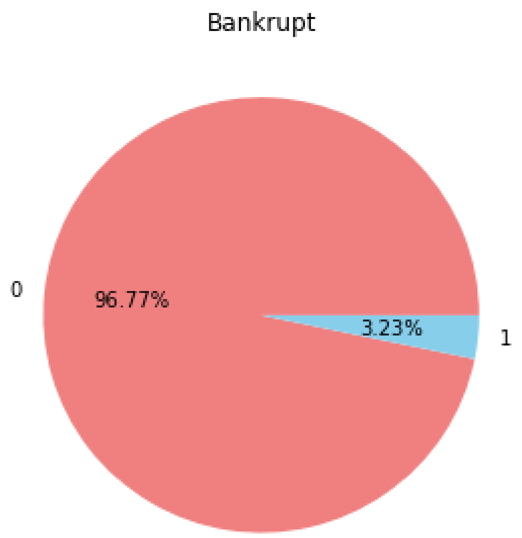
```
y.value_counts()
#0 6599
#1 220
```

Out[13]:

```
0    6599
1     220
Name: Bankrupt?, dtype: int64
```

In [14]:

```
#觀察資料佔比差異，建議使用過採樣處理
plt.figure( figsize=(10,5) )
y.value_counts().plot( kind='pie', colors=['lightcoral','skyblue'], autopct='%1.2f%%' )
plt.title( 'Bankrupt' )  # 圖標題
plt.ylabel( '' )
plt.show()
```

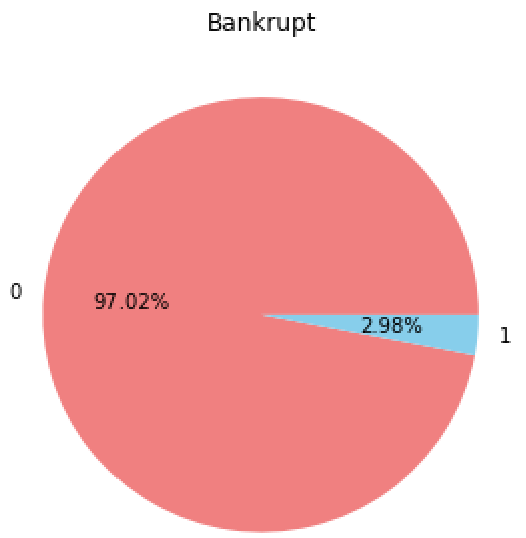


In [15]:

```
#將資料區分為訓練集和測試集
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3,random_state=42)
```

In [16]:

```
#確定取樣後的訓練集為均勻的
plt.figure( figsize=(10,5) )
y_train.value_counts().plot( kind='pie', colors=['lightcoral','skyblue'], autopct='%1.2f%%'
plt.title( 'Bankrupt' )  # 圖標題
plt.ylabel( '' )
plt.show()
```



In [17]:

```
y_train.shape
```

Out[17]:

```
(4773,)
```

In [18]:

```
x_train
```

Out[18]:

	ROA(C) before interest and depreciation before interest	ROA(A) before interest and % after tax	ROA(B) before interest and depreciation after tax	Operating Gross Margin	Realized Sales Gross Margin	Operating Profit Rate	Pre-tax net Interest Rate	After-tax net Interest Rate	Non inc expenditure
5632	0.500951	0.584169	0.553777	0.600614	0.600549	0.999025	0.797528	0.809438	
903	0.479647	0.588748	0.530007	0.609666	0.609666	0.998971	0.797330	0.809240	
2666	0.516502	0.580953	0.568714	0.610682	0.610682	0.999128	0.797571	0.809469	
109	0.446302	0.531509	0.499545	0.595771	0.595771	0.998895	0.797324	0.809286	
5316	0.575245	0.560565	0.547781	0.600001	0.600001	0.999034	0.797440	0.809349	
...	...	...	...	...	...	...	...	...	

In [19]:

```
y_train
```

Out[19]:

```
5632    0
903     0
2666    0
109     0
5316    0
       ..
3772    0
5191    0
5226    0
5390    0
860     0
Name: Bankrupt?, Length: 4773, dtype: int64
```

In [20]:

```
#進行過採樣處理
from imblearn.over_sampling import SMOTE
x_train, y_train = SMOTE().fit_resample(x_train, y_train)
```

In [21]:

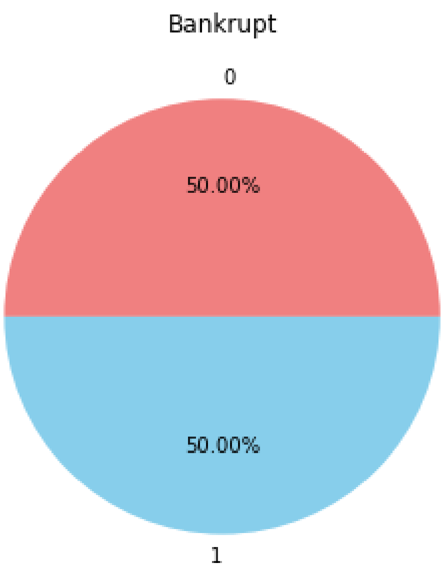
```
y_train.shape
```

Out[21]:

```
(9262,)
```

In [22]:

```
#觀察新的資料佔比差異
plt.figure( figsize=(10,5) )
y_train.value_counts().plot( kind='pie', colors=['lightcoral','skyblue'], autopct='%1.2f%%'
plt.title( 'Bankrupt' )  # 圖標題
plt.ylabel( '' )
plt.show()
```



In [23]:

```python
#合成新的資料
data_balance=pd.concat([x_train,y_train],axis=1)
data_balance
```

Out[23]:

	ROA(C) before interest and depreciation before interest	ROA(A) before interest and % after tax	ROA(B) before interest and depreciation after tax	Operating Gross Margin	Realized Sales Gross Margin	Operating Profit Rate	Pre-tax net Interest Rate	After-tax net Interest Rate	Non inc expenditure
0	0.500951	0.584169	0.553777	0.600614	0.600549	0.999025	0.797528	0.809438	
1	0.479647	0.588748	0.530007	0.609666	0.609666	0.998971	0.797330	0.809240	
2	0.516502	0.580953	0.568714	0.610682	0.610682	0.999128	0.797571	0.809469	
3	0.446302	0.531509	0.499545	0.595771	0.595771	0.998895	0.797324	0.809286	
4	0.575245	0.560565	0.547781	0.600001	0.600001	0.999034	0.797440	0.809349	
...	...	...	...	...	...	...	...	...	

In [24]:

```python
from sklearn.ensemble import AdaBoostClassifier
from sklearn.datasets import make_classification
clf = AdaBoostClassifier(n_estimators=120, random_state=0)


# clf = tree.DecisionTreeClassifier()
# clf = clf.fit(X, Y)
model = clf
#先觀察都沒做feature selection前的mode準確度
#由於是分類問題，適用LogisticRegression
# model = LogisticRegression(max_iter=1000)
model.fit(x_train,y_train)
```

Out[24]:

```
AdaBoostClassifier(n_estimators=120, random_state=0)
```

In [25]:

```
#觀察迴歸模型的準確度
print(classification_report(y_train, model.predict(x_train)))
print(classification_report(y_test, model.predict(x_test)))
```

```
              precision    recall  f1-score   support

           0       0.99      0.97      0.98      4631
           1       0.97      0.99      0.98      4631

    accuracy                           0.98      9262
   macro avg       0.98      0.98      0.98      9262
weighted avg       0.98      0.98      0.98      9262

              precision    recall  f1-score   support

           0       0.98      0.96      0.97      1968
           1       0.36      0.54      0.43        78

    accuracy                           0.95      2046
   macro avg       0.67      0.75      0.70      2046
weighted avg       0.96      0.95      0.95      2046
```

In [26]:

```
#feature selection
#使用filter法中的皮爾森相關係數方法作為篩選
#設在相關係數前n大的數字
def feature_selection_n(n):
    featuresCorr = data_balance.corr()
    targetCorr=abs(featuresCorr["Bankrupt?"])
    targetCorr = targetCorr.drop("Bankrupt?")
    result =pd.Series.argsort(-targetCorr)
    targetCorr=targetCorr[result]
    selectedFeatures=targetCorr[0:n]
    #print(f"Number of selected features: {len(selectedFeatures)} \n\nHighly relative featu
    return selectedFeatures
```

In [27]:

```
selectedFeatures=feature_selection_n(4)
selectedFeatures
```

Out[27]:

```
 Net worth/Assets                                   0.598710
 Debt ratio %                                       0.595721
 Per Share Net profit before tax (Yuan ¥)           0.570813
 ROA(C) before interest and depreciation before interest    0.569048
Name: Bankrupt?, dtype: float64
```

In [28]:

```
x_train=data_balance[selectedFeatures.index]
x_train
```

Out[28]:

	Net worth/Assets	Debt ratio %	Per Share Net profit before tax (Yuan ¥)	ROA(C) before interest and depreciation before interest
0	0.888200	0.111800	0.184454	0.500951
1	0.847689	0.152311	0.166755	0.479647
2	0.966895	0.033105	0.181969	0.516502
3	0.906310	0.093690	0.167960	0.446302
4	0.882396	0.117604	0.179935	0.575245
...	...	...	...	...
9257	0.824272	0.175728	0.161288	0.467220
9258	0.786659	0.213341	0.174660	0.494583
9259	0.806011	0.193989	0.193313	0.494676
9260	0.872011	0.127989	0.155082	0.417124
9261	0.837775	0.162225	0.166181	0.463788

9262 rows × 4 columns

In [29]:

```
y_train = data_balance["Bankrupt?"]
data_new=pd.concat([y_train,x_train],axis=1)
data_new
```

Out[29]:

	Bankrupt?	Net worth/Assets	Debt ratio %	Per Share Net profit before tax (Yuan ¥)	ROA(C) before interest and depreciation before interest
0	0	0.888200	0.111800	0.184454	0.500951
1	0	0.847689	0.152311	0.166755	0.479647
2	0	0.966895	0.033105	0.181969	0.516502
3	0	0.906310	0.093690	0.167960	0.446302
4	0	0.882396	0.117604	0.179935	0.575245
...	...	...	...	...	...
9257	1	0.824272	0.175728	0.161288	0.467220
9258	1	0.786659	0.213341	0.174660	0.494583
9259	1	0.806011	0.193989	0.193313	0.494676
9260	1	0.872011	0.127989	0.155082	0.417124
9261	1	0.837775	0.162225	0.166181	0.463788

9262 rows × 5 columns

In [30]:

```
data_new.to_csv('bankruptcy_HW1_train.csv')
```

In [31]:

```
#觀察select後的準確度
```

In [32]:

```
clf = AdaBoostClassifier(n_estimators=120, random_state=0)


# clf = tree.DecisionTreeClassifier()
# clf = clf.fit(X, Y)
model = clf
#先觀察都沒做feature selection前的mode準確度
#由於是分類問題，適用LogisticRegression
# model = LogisticRegression(max_iter=1000)
model.fit(x_train,y_train)
```

Out[32]:

```
AdaBoostClassifier(n_estimators=120, random_state=0)
```

In [33]:

```
x_test_selection=x_test[selectedFeatures.index]

x_test_selection
```

Out[33]:

	Net worth/Assets	Debt ratio %	Per Share Net profit before tax (Yuan ¥)	ROA(C) before interest and depreciation before interest
239	0.956193	0.043807	0.156963	0.434456
2850	0.870000	0.130000	0.188522	0.542534
2687	0.921365	0.078635	0.202305	0.584897
6500	0.772682	0.227318	0.151842	0.436942
2684	0.817305	0.182695	0.182345	0.506898
...	...	...	...	...
4315	0.904979	0.095021	0.176546	0.487739
2228	0.873265	0.126735	0.188747	0.517720
1083	0.905463	0.094537	0.184982	0.524204
3355	0.902984	0.097016	0.201928	0.576074
861	0.839859	0.160141	0.198539	0.516453

2046 rows × 4 columns

In [34]:

```
data_test=pd.concat([y_test,x_test_selection],axis=1)
data_test
```

Out[34]:

	Bankrupt?	Net worth/Assets	Debt ratio %	Per Share Net profit before tax (Yuan ¥)	ROA(C) before interest and depreciation before interest
239	0	0.956193	0.043807	0.156963	0.434456
2850	0	0.870000	0.130000	0.188522	0.542534
2687	0	0.921365	0.078635	0.202305	0.584897
6500	1	0.772682	0.227318	0.151842	0.436942
2684	0	0.817305	0.182695	0.182345	0.506898
...	...	...	...	...	...
4315	0	0.904979	0.095021	0.176546	0.487739
2228	0	0.873265	0.126735	0.188747	0.517720
1083	0	0.905463	0.094537	0.184982	0.524204
3355	0	0.902984	0.097016	0.201928	0.576074
861	0	0.839859	0.160141	0.198539	0.516453

2046 rows × 5 columns

In [36]:

```
data_test.to_csv('bankruptcy_HW1_test.csv')
```

In [37]:

```
print(classification_report(y_train, model.predict(x_train)))
print(classification_report(y_test, model.predict(x_test_selection)))
```

```
              precision    recall  f1-score   support

           0       0.91      0.85      0.88      4631
           1       0.86      0.92      0.89      4631

    accuracy                           0.88      9262
   macro avg       0.89      0.88      0.88      9262
weighted avg       0.89      0.88      0.88      9262

              precision    recall  f1-score   support

           0       0.99      0.84      0.91      1968
           1       0.18      0.86      0.29        78

    accuracy                           0.84      2046
   macro avg       0.59      0.85      0.60      2046
weighted avg       0.96      0.84      0.89      2046
```