Bankruptcy

In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:

```
raw_df = pd.read_csv("bankruptcy.csv")
```

In [3]:

```
raw_df.head(5)
```

Out[3]:

	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	Oper I
0	0	1	0.370594	0.424389	0.405750	0.601457	0.601457	0.99
1	1	1	0.464291	0.538214	0.516730	0.610235	0.610235	0.99
2	2	1	0.426071	0.499019	0.472295	0.601450	0.601364	0.99
3	3	1	0.399844	0.451265	0.457733	0.583541	0.583541	0.99
4	4	1	0.465022	0.538432	0.522298	0.598783	0.598783	0.99

5 rows × 97 columns

In [4]:

```
raw_df.isna().sum()
```

Out[4]:

```
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 [5]:

```
x= raw_df.iloc[:,2:]
y= raw_df.iloc[:,1]
```

In [6]:

```
x
```

Out[6]:

	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	Aft Int
0	0.370594	0.424389	0.405750	0.601457	0.601457	0.998969	0.796887	0.8(
1	0.464291	0.538214	0.516730	0.610235	0.610235	0.998946	0.797380	0.8(
2	0.426071	0.499019	0.472295	0.601450	0.601364	0.998857	0.796403	0.8(
3	0.399844	0.451265	0.457733	0.583541	0.583541	0.998700	0.796967	0.8(
4	0.465022	0.538432	0.522298	0.598783	0.598783	0.998973	0.797366	0.8(
...	...	...	...	...	...	...	...	
6814	0.493687	0.539468	0.543230	0.604455	0.604462	0.998992	0.797409	0.8(
6815	0.475162	0.538269	0.524172	0.598308	0.598308	0.998992	0.797414	0.8(
6816	0.472725	0.533744	0.520638	0.610444	0.610213	0.998984	0.797401	0.8(
6817	0.506264	0.559911	0.554045	0.607850	0.607850	0.999074	0.797500	0.8(
6818	0.493053	0.570105	0.549548	0.627409	0.627409	0.998080	0.801987	0.8

6819 rows × 95 columns

In [7]:

```
from sklearn.impute import KNNImputer
```

In [8]:

```
imputer = KNNImputer(n_neighbors=2)
raw=imputer.fit_transform(x)
```

In [9]:

```
raw
```

Out[9]:

```
array([[0.37059426, 0.42438945, 0.40574977, ..., 0.56405011, 1.        ,
        0.01646874],
       [0.46429094, 0.53821413, 0.51673002, ..., 0.57017495, 1.        ,
        0.02079431],
       [0.42607127, 0.49901875, 0.47229509, ..., 0.56370608, 1.        ,
        0.01647411],
       ...,
       [0.47272461, 0.533744  , 0.52063815, ..., 0.5651584 , 1.        ,
        0.09764874],
       [0.50626432, 0.5599106 , 0.55404465, ..., 0.56530151, 1.        ,
        0.04400945],
       [0.49305319, 0.57010467, 0.54954762, ..., 0.56516694, 1.        ,
        0.23390224]])
```

In [10]:

```
columns = x.columns.tolist()
```

In [11]:

```
df = pd.DataFrame(raw, columns = columns )
```

In [12]:

```
df.isna().sum()
```

Out[12]:

```
 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
 Operating Gross Margin                                     0
 Realized Sales Gross Margin                                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: 95, dtype: int64
```

In [13]:

```
df
```

	interest before interest	and % after tax	depreciation after tax	Margin	Margin	Rate	Rate	Rate
0	0.370594	0.424389	0.405750	0.601457	0.601457	0.998969	0.796887	0.808809
1	0.464291	0.538214	0.516730	0.610235	0.610235	0.998946	0.797380	0.809301
2	0.426071	0.499019	0.472295	0.601450	0.601364	0.998857	0.796403	0.808388
3	0.399844	0.451265	0.457733	0.583541	0.583541	0.998700	0.796967	0.808966
4	0.465022	0.538432	0.522298	0.598783	0.598783	0.998973	0.797366	0.809304
...	...	...	...	...	...	...	...	...
6814	0.493687	0.539468	0.543230	0.604455	0.604462	0.998992	0.797409	0.809331
6815	0.475162	0.538269	0.524172	0.598308	0.598308	0.998992	0.797414	0.809327
6816	0.472725	0.533744	0.520638	0.610444	0.610213	0.998984	0.797401	0.809317
6817	0.506264	0.559911	0.554045	0.607850	0.607850	0.999074	0.797500	0.809399
6818	0.493053	0.570105	0.549548	0.627409	0.627409	0.998080	0.801987	0.813800

y 值 是類別，處理類別不平衡的問題

In [14]:

```
from imblearn.combine import SMOTEENN
from collections import Counter
```

In [15]:

```
counter= Counter(y)
print("before y ", counter)
```

```
before y  Counter({0: 6599, 1: 220})
```

In [16]:

```
smenn= SMOTEENN()
```

In [17]:

```
x_sm,y_sm= smenn.fit_resample(df,y)
```

In [18]:

```
counters=Counter(y_sm)
print("after y", counters)
```

```
after y Counter({1: 6285, 0: 5423})
```

In [19]:

```
print("before y ", counter)
print("after y", counters)
```

```
before y  Counter({0: 6599, 1: 220})
after y Counter({1: 6285, 0: 5423})
```

In [20]:

```
df
```

Out[20]:

	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	Aft Int
0	0.370594	0.424389	0.405750	0.601457	0.601457	0.998969	0.796887	0.8(
1	0.464291	0.538214	0.516730	0.610235	0.610235	0.998946	0.797380	0.8(
2	0.426071	0.499019	0.472295	0.601450	0.601364	0.998857	0.796403	0.8(
3	0.399844	0.451265	0.457733	0.583541	0.583541	0.998700	0.796967	0.8(
4	0.465022	0.538432	0.522298	0.598783	0.598783	0.998973	0.797366	0.8(
...	...	...	...	...	...	...	...	
6814	0.493687	0.539468	0.543230	0.604455	0.604462	0.998992	0.797409	0.8(
6815	0.475162	0.538269	0.524172	0.598308	0.598308	0.998992	0.797414	0.8(
6816	0.472725	0.533744	0.520638	0.610444	0.610213	0.998984	0.797401	0.8(
6817	0.506264	0.559911	0.554045	0.607850	0.607850	0.999074	0.797500	0.8(
6818	0.493053	0.570105	0.549548	0.627409	0.627409	0.998080	0.801987	0.8

6819 rows × 95 columns

Feature selection

In [21]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [162]:

```
model = RandomForestClassifier(n_estimators = 10)
model.fit(df,y)
importances = model.feature_importances_
```

In [23]:

```
final_df = pd.DataFrame({"Features":pd.DataFrame(df).columns,"Importances":importances})
#final_df.set_index("Importances")
final_df
```

Out[23]:

	Features	Importances
0	ROA(C) before interest and depreciation befor...	0.003017
1	ROA(A) before interest and % after tax	0.016594
2	ROA(B) before interest and depreciation after...	0.009073
3	Operating Gross Margin	0.010157
4	Realized Sales Gross Margin	0.004165
...	...	...
90	Liability to Equity	0.006228
91	Degree of Financial Leverage (DFL)	0.025248
92	Interest Coverage Ratio (Interest expense to ...	0.020463
93	Net Income Flag	0.000000
94	Equity to Liability	0.013462

95 rows × 2 columns

In [28]:

```
final= final_df.sort_values("Importances")
select = final.iloc[51:,:]
```

In [29]:

```
select
```

Out[29]:

	Features	Importances
2	ROA(B) before interest and depreciation after...	0.009073
72	Working capitcal Turnover Rate	0.009075
46	Average Collection Days	0.009081
8	Non-industry income and expenditure/revenue	0.009160
53	Working Capital to Total Assets	0.009257
7	After-tax net Interest Rate	0.009549
55	Current Assets/Total Assets	0.009789
40	Contingent liabilities/Net worth	0.009805
5	Operating Profit Rate	0.010128
32	Current Ratio	0.010144
3	Operating Gross Margin	0.010157
86	Total assets to GNP price	0.010462
48	Fixed Assets Turnover Frequency	0.010495
10	Operating Expense Rate	0.010620
77	Current Liability to Equity	0.010815
6	Pre-tax net Interest Rate	0.010832
29	Net Value Growth Rate	0.011352
74	Cash Flow to Sales	0.011893
42	Net profit before tax/Paid-in capital	0.012271
34	Interest Expense Ratio	0.012536
58	Cash/Current Liability	0.012569
78	Equity to Long-term Liability	0.013001
45	Accounts Receivable Turnover	0.013300
59	Current Liability to Assets	0.013388
94	Equity to Liability	0.013462
52	Allocation rate per person	0.013712
87	No-credit Interval	0.013868
65	Current Liabilities/Equity	0.014369
1	ROA(A) before interest and % after tax	0.016594
13	Interest-bearing debt interest rate	0.016786
33	Quick Ratio	0.017909
56	Cash/Total Assets	0.018621
36	Debt ratio %	0.019233

	Features	Importances
92	Interest Coverage Ratio (Interest expense to ...	0.020463
15	Net Value Per Share (B)	0.020594
89	Net Income to Stockholder's Equity	0.022260
37	Net worth/Assets	0.022909
9	Continuous interest rate (after tax)	0.023742
91	Degree of Financial Leverage (DFL)	0.025248
64	Working Capital/Equity	0.026073
39	Borrowing dependency	0.027214
18	Persistent EPS in the Last Four Seasons	0.032335
22	Per Share Net profit before tax (Yuan ¥)	0.032726
85	Net Income to Total Assets	0.040368

In [30]:

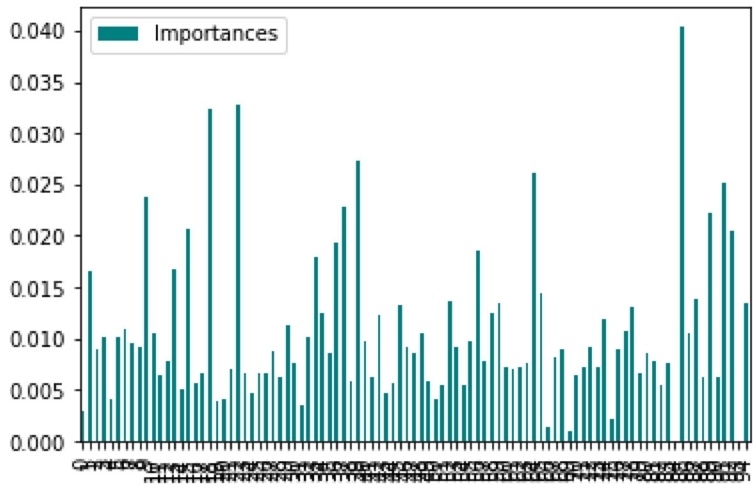
```
select.to_excel("bankruptcy_output.xlsx")
```

In [31]:

```
final_df.plot.bar(color='teal')
```

Out[31]:

<AxesSubplot:>



Diamonds

In [32]:

```
raw_df2 = pd.read_csv("daimonds.csv")
```

In [33]:

```
raw_df2.head(5)
```

Out[33]:

	Unnamed: 0	Unnamed: 0.1	carat	cut	color	clarity	depth	table	x	y	z	p
0	0	1	0.23	NaN	E	SI2	61.5	55.0	3.95	3.98	2.43	3:
1	1	2	0.21	Premium	E	SI1	59.8	NaN	3.89	3.84	2.31	3:
2	2	3	0.23	Good	E	VS1	56.9	65.0	4.05	4.07	2.31	3:
3	3	4	0.29	Premium	I	NaN	62.4	58.0	4.20	NaN	2.63	3:
4	4	5	0.31	Good	J	SI2	63.3	58.0	4.34	4.35	2.75	3:

In [34]:

```
raw_df2.isna().sum()
```

Out[34]:

```
Unnamed: 0        0
Unnamed: 0.1      0
carat           993
cut             989
color           992
clarity         995
depth           990
table           993
x               991
y               997
z               992
price           992
dtype: int64
```

category features 先encode 再進行補值

In [37]:

```
from sklearn.preprocessing import LabelEncoder
labelencoder = LabelEncoder()
```

In [38]:

```
raw_df2['cut'] = labelencoder.fit_transform(raw_df2['cut'])
raw_df2['color']= labelencoder.fit_transform(raw_df2['color'])
raw_df2['clarity']= labelencoder.fit_transform(raw_df2['clarity'])
```

In [39]:

```
from sklearn.impute import KNNImputer
```

In [40]:

```
imputer = KNNImputer(n_neighbors=2)
raw=imputer.fit_transform(raw_df2)
```

In [41]:

```
columns2 = raw_df2.columns.tolist()
```

In [42]:

```
df2 = pd.DataFrame(raw, columns = columns2 )
```

In [43]:

```
x1= df2.iloc[:,:11]
y1= df2.iloc[:,11]
```

In [44]:

```
y1 = np.array(y1, dtype=int)
```

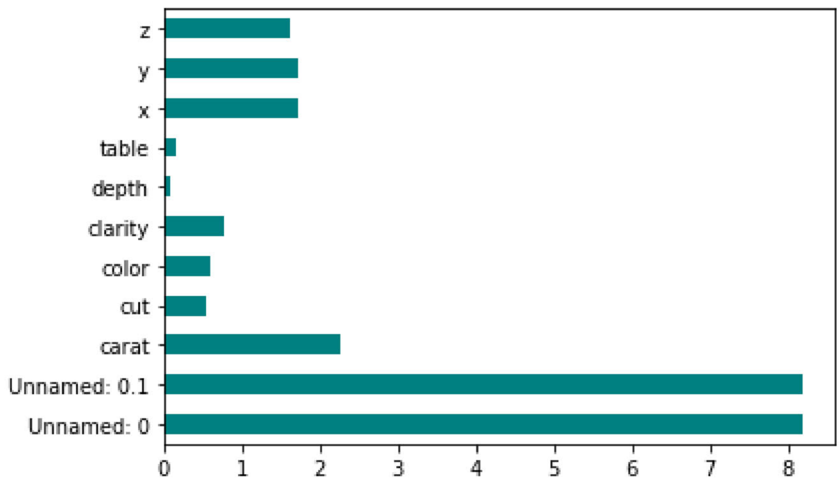
y = price 非類別不用作data imbalance

feature selection (filter)

In [45]:

```
from sklearn.feature_selection import mutual_info_classif

importances= mutual_info_classif(x1,y1)
feat_importances = pd.Series(importances,x1.columns[0:len(x1.columns)])
feat_importances.plot(kind='barh',color='teal')
plt.show()
```



由此我們可以知道那些feature是重要的

In [47]:

```
feat_importances.to_excel("Diamond_output.xlsx")
```

iris data

In [48]:

```
raw_df3 = pd.read_csv("iris_data.csv")
```

In [49]:

```
raw_df3.head(5)
```

Out[49]:

	Unnamed: 0	Sepal length	Sepal width	Petal length	Petal width	label
0	0	5.1	3.5	1.4	0.2	setora
1	1	4.9	3.0	1.4	0.2	setora
2	2	4.7	3.2	1.3	0.2	setora
3	3	4.6	3.1	1.5	0.2	setora
4	4	5.0	3.6	1.4	0.2	setora

In [50]:

```
raw_df3.isna().sum()
```

Out[50]:

```
Unnamed: 0       0
Sepal length    15
Sepal width     15
Petal length    15
Petal width     15
label            0
dtype: int64
```

In [87]:

```
x3= raw_df3.iloc[:,1:5]
y3= raw_df3.iloc[:,5]
```

補空值

In [88]:

```
from sklearn.impute import KNNImputer
```

In [89]:

```
imputer = KNNImputer(n_neighbors=2)
raw=imputer.fit_transform(x3)
```

In [90]:

```
columns3 = x3.columns.tolist()
df3 = pd.DataFrame(raw, columns = columns3 )
```

In [91]:

```
df3
```

Out[91]:

	Sepal length	Sepal width	Petal length	Petal width
0	5.1	3.5	1.4	0.20
1	4.9	3.0	1.4	0.20
2	4.7	3.2	1.3	0.20
3	4.6	3.1	1.5	0.20
4	5.0	3.6	1.4	0.20
...	...	...	...	...
145	6.7	3.0	5.2	2.30
146	6.3	2.5	5.0	1.95
147	6.5	3.0	5.2	2.00
148	6.2	3.4	5.4	2.30
149	5.9	3.0	5.1	1.80

150 rows × 4 columns

類別不平衡

In [92]:

```
from imblearn.combine import SMOTEENN
from collections import Counter
```

In [93]:

```
y3
```

Out[93]:

```
0        setora
1        setora
2        setora
3        setora
4        setora
          ...
145    virginica
146    virginica
147    virginica
148    virginica
149    virginica
Name: label, Length: 150, dtype: object
```

In [94]:

```
counter3= Counter(y3)
print("before y ", counter3,'沒有類別不平衡的問題')
```

```
before y  Counter({'setora': 50, 'versicolor': 50, 'virginica': 50}) 沒有類別
不平衡的問題
```

feature selection

In [95]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import SelectFromModel
```

In [96]:

```
logistic =LogisticRegression(C=1,penalty='l1',solver= 'liblinear',random_state= 7).fit(df3,
```

In [97]:

```
model =SelectFromModel(logistic,prefit=True)
```

In [98]:

```
x_new = model.transform(df3)
```

In [99]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [100]:

```
model1 = RandomForestClassifier(n_estimators=150)
```

In [101]:

```
model1.fit(df3,y3)
```

Out[101]:

```
RandomForestClassifier(n_estimators=150)
```

In [102]:

```
importances= model1.feature_importances_
```

In [103]:

```
final_df3=pd.DataFrame({"Features":pd.DataFrame(df3).columns,"Importances":importances})
final_df3.set_index('Importances')
```

Out[103]:

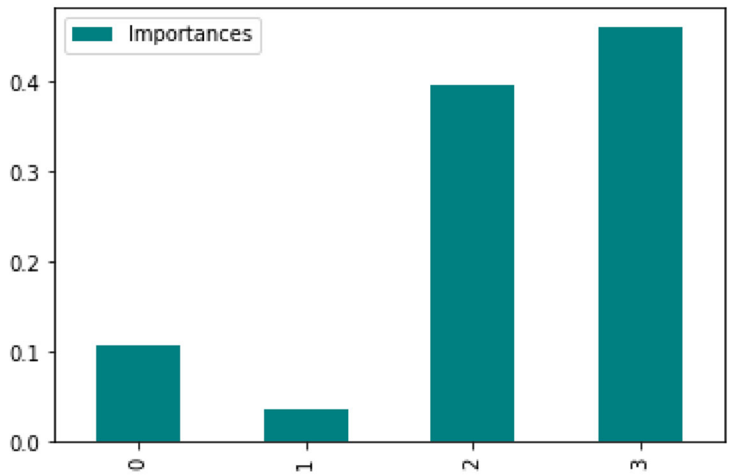
	Features
Importances	
0.108022	Sepal length
0.035845	Sepal width
0.396007	Petal length
0.460127	Petal width

In [104]:

```
final_df3.plot.bar(color='teal') # sepal width 比較不重要
```

Out[104]:

<AxesSubplot:>



In [147]:

```
final3=final_df3.sort_values("Importances")
```

In [149]:

```
final3.to_excel("iris_output.xlsx")
```

Stroke

In [105]:

```
raw_df4 = pd.read_csv("stroke.csv")
```

In [106]:

```
raw_df4.head(5)
```

Out[106]:

named: 0	id	gender	age	hypertension	heart_disease	ever_married	work_type	Reside
0	9046	Male	67.0	0	1	Yes	Private	
1	51676	Female	61.0	0	0	Yes	Self-employed	
2	31112	Male	80.0	0	1	Yes	Private	
3	60182	Female	49.0	0	0	Yes	Private	
4	1665	Female	79.0	1	0	Yes	Self-employed	

In [131]:

```
raw_df4.isna().sum()
```

Out[131]:

```
Unnamed: 0             0
id                     0
gender                 0
age                    0
hypertension           0
heart_disease          0
ever_married           0
work_type              0
Residence_type         0
avg_glucose_level      0
bmi                  201
smoking_status         0
stroke                 0
dtype: int64
```

In [132]:

```
x4= raw_df4.iloc[:,2:12]
y4= raw_df4.iloc[:,12]
```

先encoding 再進行補值

In [133]:

```
x4['gender'] = labelencoder.fit_transform(x4['gender'])
x4['ever_married'] = labelencoder.fit_transform(x4['ever_married'])
x4['work_type']= labelencoder.fit_transform(x4['work_type'])
x4['Residence_type']= labelencoder.fit_transform(x4['Residence_type'])
x4['smoking_status']= labelencoder.fit_transform(x4['smoking_status'])
```

In [134]:

```
from sklearn.impute import KNNImputer
```

In [135]:

```
imputer = KNNImputer(n_neighbors=2)
raw=imputer.fit_transform(x4)
```

In [136]:

```
columns4 = x4.columns.tolist()
```

In [137]:

```
df4 = pd.DataFrame(raw, columns = columns4 )
```

In [138]:

```
y4 = np.array(y4, dtype=int)
```

y 值類別不平衡

In [150]:

```
from imblearn.combine import SMOTEENN
from collections import Counter
```

In [151]:

```
counter4= Counter(y4)
print("before y ", counter4)
```

```
before y  Counter({0: 4861, 1: 249})
```

In [152]:

```
smenn= SMOTEENN()
```

In [153]:

```
x_sm4,y_sm4= smenn.fit_resample(df4,y4)
```

In [154]:

```
counters=Counter(y_sm4)
print("after y", counters)
```

```
after y Counter({1: 4565, 0: 3662})
```

In [155]:

```
print("before y ", counter4)
print("after y", counters)
```

```
before y  Counter({0: 4861, 1: 249})
after y Counter({1: 4565, 0: 3662})
```

feature selection

In [156]:

```
model4 = RandomForestClassifier(n_estimators = 10)
model4.fit(df4,y4)
importances = model4.feature_importances_
```

In [157]:

```
final_df4 = pd.DataFrame({"Features":pd.DataFrame(df4).columns,"Importances":importances})
#final_df.set_index("Importances")
final_df4
```

Out[157]:

	Features	Importances
0	gender	0.027727
1	age	0.234486
2	hypertension	0.028371
3	heart_disease	0.020468
4	ever_married	0.013618
5	work_type	0.050271
6	Residence_type	0.028193
7	avg_glucose_level	0.301028
8	bmi	0.235493
9	smoking_status	0.060345

In [158]:

```
final4= final_df4.sort_values("Importances")
```

In [159]:

```
final4.to_excel("stroke_output.xlsx")
```

In [ ]:

```
```