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]:

```
from sklearn.impute import KNNImputer
imputer = KNNImputer(n_neighbors=2)
raw=imputer.fit_transform(x)
```

In [7]:

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

In [8]:

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

In [9]:

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

Out[9]:

```
 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 [10]:

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

In [11]:

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

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

In [12]:

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

In [13]:

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

```
after y Counter({1: 6295, 0: 5427})
```

In [14]:

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

```
before y  Counter({0: 6599, 1: 220})
after y Counter({1: 6295, 0: 5427})
```

In [15]:

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

In [16]:

```
model = RandomForestClassifier(n_estimators = 10)
model.fit(x_sm,y_sm)
importances = model.feature_importances_
```

In [17]:

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

In [18]:

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

In [19]:

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

In [20]:

```
k=select.iloc[:,0]
```

In [21]:

```
k
```

Out[21]:

```
44                                     Total Asset Turnover
11                  Research and development expense rate
50                                       Revenue per person
71                                Quick Asset Turnover Rate
70                              Current Asset Turnover Rate
80                                    Cash Flow to Liability
20                               Revenue Per Share (Yuan ¥)
88                                      Gross Profit to Sales
0      ROA(C) before interest and depreciation befor...
68                               Total income/Total expense
94                                      Equity to Liability
29                                    Net Value Growth Rate
78                             Equity to Long-term Liability
86                                Total assets to GNP price
90                                      Liability to Equity
28                                  Total Asset Growth Rate
52                               Allocation rate per person
22               Per Share Net profit before tax (Yuan ¥)
36                                            Debt ratio %
77                              Current Liability to Equity
48                          Fixed Assets Turnover Frequency
46                                  Average Collection Days
92     Interest Coverage Ratio (Interest expense to ...
89                         Net Income to Stockholder's Equity
2      ROA(B) before interest and depreciation after...
65                               Current Liabilities/Equity
91                         Degree of Financial Leverage (DFL)
59                              Current Liability to Assets
13                       Interest-bearing debt interest rate
73                                       Cash Turnover Rate
56                                        Cash/Total Assets
37                                         Net worth/Assets
34                                    Interest Expense Ratio
33                                              Quick Ratio
67                         Retained Earnings to Total Assets
35                               Total debt/Total net worth
8             Non-industry income and expenditure/revenue
42                       Net profit before tax/Paid-in capital
18                    Persistent EPS in the Last Four Seasons
7                               After-tax net Interest Rate
9                       Continuous interest rate (after tax)
39                                     Borrowing dependency
6                                 Pre-tax net Interest Rate
85                               Net Income to Total Assets
Name: Features, dtype: object
```

In [22]:

```
k_=pd.DataFrame(x_sm,columns=k)
```

In [23]:

```
k_
```

Out[23]:

Features	Total Asset Turnover	Research and development expense rate	Revenue per person	Quick Asset Turnover Rate	Current Asset Turnover Rate	Cash Flow to Liability	Revei Sh (Yuai
0	0.100450	7.300000e+08	0.011460	9.560000e+09	1.058011e-04	0.457785	0.030
1	0.218891	5.090000e+07	0.028077	6.180000e+09	7.290000e+09	0.458954	0.042
2	0.154423	0.000000e+00	0.016383	9.840000e+09	1.026722e-04	0.462165	0.026
3	0.347826	0.000000e+00	0.033280	3.600000e+09	5.110000e+09	0.459379	0.078
4	0.076462	6.900000e+08	0.013895	2.920000e+09	4.760000e+09	0.459280	0.015
...	...	...	...	...	...	...	
11717	0.172532	1.037504e+09	0.094713	2.435496e+09	2.075007e+08	0.459763	0.044
11718	0.085077	0.000000e+00	0.017773	5.399226e+09	7.363139e+09	0.450323	0.014
11719	0.128613	4.269623e+09	0.024474	9.110656e+09	1.502774e-04	0.460511	0.012
11720	0.141242	2.518975e+09	0.079475	1.107437e-04	1.750056e-04	0.458991	0.043
11721	0.167875	7.118124e+09	0.018900	1.421188e-04	1.910839e-04	0.461538	0.060

11722 rows × 44 columns

In [24]:

```
k_y=pd.DataFrame(y_sm,columns=k)
```

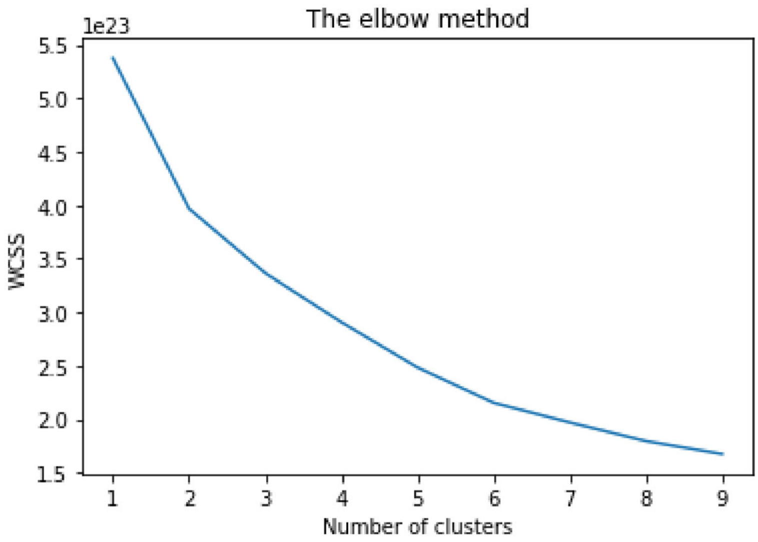
K means

In [26]:

```
from sklearn.cluster import KMeans
wcss = []

for i in range(1, 10):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init = 10, random
    kmeans.fit(k_)
    wcss.append(kmeans.inertia_)
```

In [27]:

```
plt.plot(range(1, 10), wcss)
plt.title('The elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS') #within cluster sum of squares
plt.show()
```



最佳的cluster數量是2

In [28]:

```
kmeans = KMeans(n_clusters=2)
kmeans= kmeans.fit(k_)
```

In [29]:

```
kmeans.labels_
```

Out[29]:

```
array([0, 0, 0, ..., 0, 1, 1])
```

In [30]:

```
kmeans= pd.DataFrame(kmeans.labels_, columns= ['result'])
```

In [31]:

```
kmeans
```

Out[31]:

	result
0	0
1	0
2	0
3	0
4	0
...	...
11717	1
11718	0
11719	0
11720	1
11721	1

11722 rows × 1 columns

Fuzzy c means

In [33]:

```
from fcmeans import FCM
from matplotlib import pyplot as plt
```

In [34]:

```
fcm = FCM(n_clusters=2)
fcm.fit(k_)
```

In [35]:

```
fcm_centers = fcm.centers
fcm_labels = fcm.predict(k_)
```

In [36]:

```
fcmeans= pd.DataFrame(fcm_labels, columns= ['result'])
```

In [37]:

```
fcmeans
```

Out[37]:

	result
0	1
1	1
2	1
3	1
4	1
...	...
11717	0
11718	1
11719	1
11720	0
11721	0

11722 rows × 1 columns

In [79]:

```
from sklearn import cluster, datasets
```

In [80]:

```
hclust_S = cluster.AgglomerativeClustering(linkage = 'single', affinity = 'euclidean', n_cl
hclust_C = cluster.AgglomerativeClustering(linkage = 'complete', affinity = 'euclidean', n_
hclust_A = cluster.AgglomerativeClustering(linkage = 'average', affinity = 'euclidean', n_c
```

In [81]:

```
hclust_S.fit(k_)
cluster_labels = hclust_S.labels_
print(cluster_labels)
print("---")
hclust_C.fit(k_)
cluster_labels = hclust_C.labels_
print(cluster_labels)
print("---")
hclust_A.fit(k_)
cluster_labels = hclust_A.labels_
print(cluster_labels)
print("---")
```

```
[0 0 0 ... 0 0 0]
---
[2 1 2 ... 2 0 0]
---
[0 0 0 ... 0 0 0]
---
```

Diamonds

In [38]:

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

In [39]:

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

Out[39]:

```
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
```

In [40]:

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

In [41]:

```
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 [42]:

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

In [43]:

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

In [44]:

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

In [58]:

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

In [59]:

```
df2.isna().sum()
```

Out[59]:

```
Unnamed: 0      0
Unnamed: 0.1    0
carat           0
cut             0
color           0
clarity         0
depth           0
table           0
x               0
y               0
z               0
price           0
dtype: int64
```

In [60]:

```
df2
```

Out[60]:

	Unnamed: 0	Unnamed: 0.1	carat	cut	color	clarity	depth	table	x	y	z	p
0	0.0	1.0	0.23	5.0	1.0	3.0	61.5	55.0	3.95	3.98	2.43	3
1	1.0	2.0	0.21	3.0	1.0	2.0	59.8	60.0	3.89	3.84	2.31	3
2	2.0	3.0	0.23	1.0	1.0	4.0	56.9	65.0	4.05	4.07	2.31	3
3	3.0	4.0	0.29	3.0	5.0	8.0	62.4	58.0	4.20	3.97	2.63	3
4	4.0	5.0	0.31	1.0	6.0	3.0	63.3	58.0	4.34	4.35	2.75	3
...	...	...	...	...	...	...	...	...	...	...	...	
53935	53935.0	53936.0	0.72	2.0	0.0	2.0	60.8	57.0	5.75	5.76	3.50	27
53936	53936.0	53937.0	0.72	1.0	0.0	2.0	63.1	55.0	5.69	5.75	3.61	27
53937	53937.0	53938.0	0.70	4.0	0.0	2.0	62.8	60.0	5.66	5.68	3.56	27
53938	53938.0	53939.0	0.86	3.0	4.0	3.0	61.0	58.0	6.15	6.12	3.74	27
53939	53939.0	53940.0	0.75	2.0	0.0	8.0	62.2	59.0	5.83	5.87	3.64	27

53940 rows × 12 columns

In [85]:

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

In [86]:

```
x1
```

Out[86]:

	Unnamed: 0	Unnamed: 0.1	carat	cut	color	clarity	depth	table	x	y	z
0	0.0	1.0	0.23	5.0	1.0	3.0	61.5	55.0	3.95	3.98	2.43
1	1.0	2.0	0.21	3.0	1.0	2.0	59.8	60.0	3.89	3.84	2.31
2	2.0	3.0	0.23	1.0	1.0	4.0	56.9	65.0	4.05	4.07	2.31
3	3.0	4.0	0.29	3.0	5.0	8.0	62.4	58.0	4.20	3.97	2.63
4	4.0	5.0	0.31	1.0	6.0	3.0	63.3	58.0	4.34	4.35	2.75
...	...	...	...	...	...	...	...	...	...	...	...
53935	53935.0	53936.0	0.72	2.0	0.0	2.0	60.8	57.0	5.75	5.76	3.50
53936	53936.0	53937.0	0.72	1.0	0.0	2.0	63.1	55.0	5.69	5.75	3.61
53937	53937.0	53938.0	0.70	4.0	0.0	2.0	62.8	60.0	5.66	5.68	3.56
53938	53938.0	53939.0	0.86	3.0	4.0	3.0	61.0	58.0	6.15	6.12	3.74
53939	53939.0	53940.0	0.75	2.0	0.0	8.0	62.2	59.0	5.83	5.87	3.64

53940 rows × 11 columns

In [87]:

```
y1.isna().sum()
```

Out[87]:

```
0
```

In [92]:

```
y2 = np.array(y1, dtype=int)
```

In [88]:

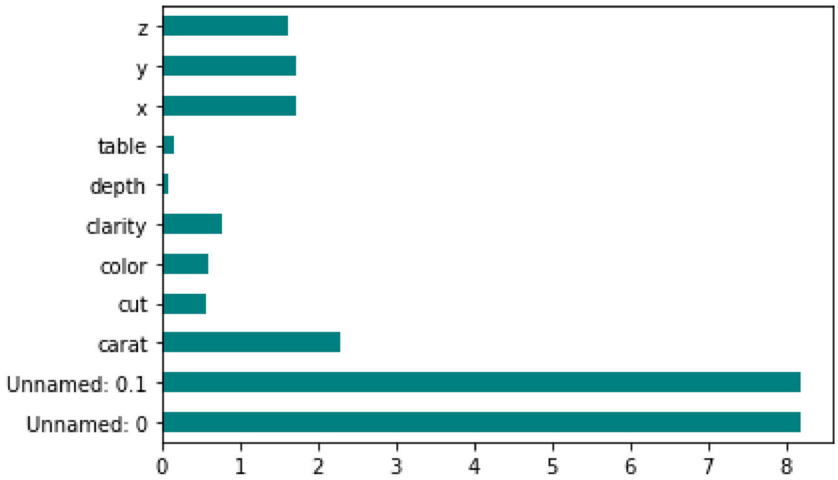
```
y1
```

Out[88]:

```
0         326.0
1         326.0
2         327.0
3         334.0
4         335.0
          ...
53935    2757.0
53936    2757.0
53937    2757.0
53938    2757.0
53939    2757.0
Name: price, Length: 53940, dtype: float64
```

In [93]:

```
from sklearn.feature_selection import mutual_info_classif

importances= mutual_info_classif(x1,y2)
feat_importances = pd.Series(importances,x1.columns[0:len(x1.columns)])
feat_importances.plot(kind='barh',color='teal')
plt.show()
```



In [94]:

```
final_df3=pd.DataFrame({"Features":pd.DataFrame(x1).columns,"Importances":importances})
final_df3.set_index('Importances')
```

Out[94]:

	Features
Importances	
8.183132	Unnamed: 0
8.181313	Unnamed: 0.1
2.280398	carat
0.569866	cut
0.598174	color
0.771725	clarity
0.084834	depth
0.168877	table
1.735349	x
1.736352	y
1.616834	z

feature selection

In [95]:

```
k_D=pd.DataFrame(x1,columns=["clarity",'color',"x","y","z",'carat'])
```

In [96]:

```
k_D
```

Out[96]:

	clarity	color	x	y	z	carat
0	3.0	1.0	3.95	3.98	2.43	0.23
1	2.0	1.0	3.89	3.84	2.31	0.21
2	4.0	1.0	4.05	4.07	2.31	0.23
3	8.0	5.0	4.20	3.97	2.63	0.29
4	3.0	6.0	4.34	4.35	2.75	0.31
...	...	...	...	...	...	...
53935	2.0	0.0	5.75	5.76	3.50	0.72
53936	2.0	0.0	5.69	5.75	3.61	0.72
53937	2.0	0.0	5.66	5.68	3.56	0.70
53938	3.0	4.0	6.15	6.12	3.74	0.86
53939	8.0	0.0	5.83	5.87	3.64	0.75

53940 rows × 6 columns

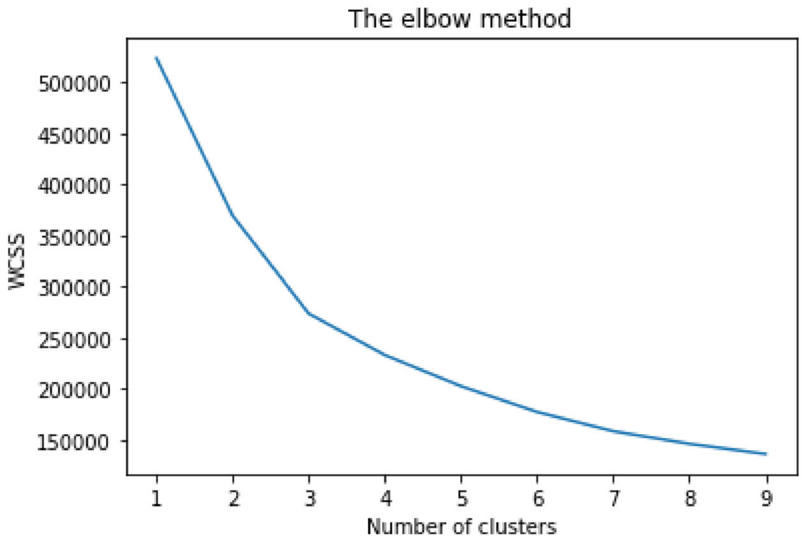
K means

In [97]:

```
from sklearn.cluster import KMeans
wcss = []

for i in range(1, 10):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init = 10, random
    kmeans.fit(k_D)
    wcss.append(kmeans.inertia_)
```

In [98]:

```
plt.plot(range(1, 10), wcss)
plt.title('The elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS') #within cluster sum of squares
plt.show()
```



In [99]:

```
kmeans2 = KMeans(n_clusters=3)
kmeans2= kmeans2.fit(k_D)
```

In [100]:

```
kmeans2.labels_
```

Out[100]:

```
array([0, 0, 1, ..., 0, 2, 1])
```

In [101]:

```
kmeans2_D= pd.DataFrame(kmeans2.labels_, columns= ['result'])
```

In [102]:

```
kmeans2_D
```

Out[102]:

	result
0	0
1	0
2	1
3	1
4	2
...	...
53935	0
53936	0
53937	0
53938	2
53939	1

53940 rows × 1 columns

fuzzy c means u

先做**PCA**

In [103]:

```
from sklearn.preprocessing import StandardScaler

x_standard_bank = pd.DataFrame(StandardScaler().fit_transform(k_D))

# check mean and std
#x_stand_descr_bank = x_standard_bank.describe().loc[['mean', 'std']]
#x_stand_descr_bank.style.format("{:.1f}")
```

In [91]:

```
from sklearn.decomposition import PCA

pca2 = PCA(n_components = 2, random_state = 18)
PCA_result_bank2 = pca2.fit_transform(x_standard_bank)
PCA_skl_bank_df2 = pd.DataFrame(np.hstack((PCA_result_bank2, y1.to_numpy().reshape(-1, 1)))
PCA_skl_bank_df2
```

Out[91]:

	PC1	PC2	category
0	-2.965780	-0.990163	326.0
1	-3.080053	-1.434004	326.0
2	-3.041527	-0.546504	327.0
3	-2.668392	2.579268	334.0
4	-1.873406	0.700771	335.0
...	...	...	...
53935	-0.189792	-1.754529	2757.0
53936	-0.144475	-1.754813	2757.0
53937	-0.242610	-1.755527	2757.0
53938	0.742062	0.044872	2757.0
53939	-0.431452	0.906404	2757.0

53940 rows × 3 columns

In [104]:

```
xpts = PCA_skl_bank_df2.iloc[:, 0]
ypts = PCA_skl_bank_df2.iloc[:, 1]
labels = PCA_skl_bank_df2.iloc[:, 2]
```

In [105]:

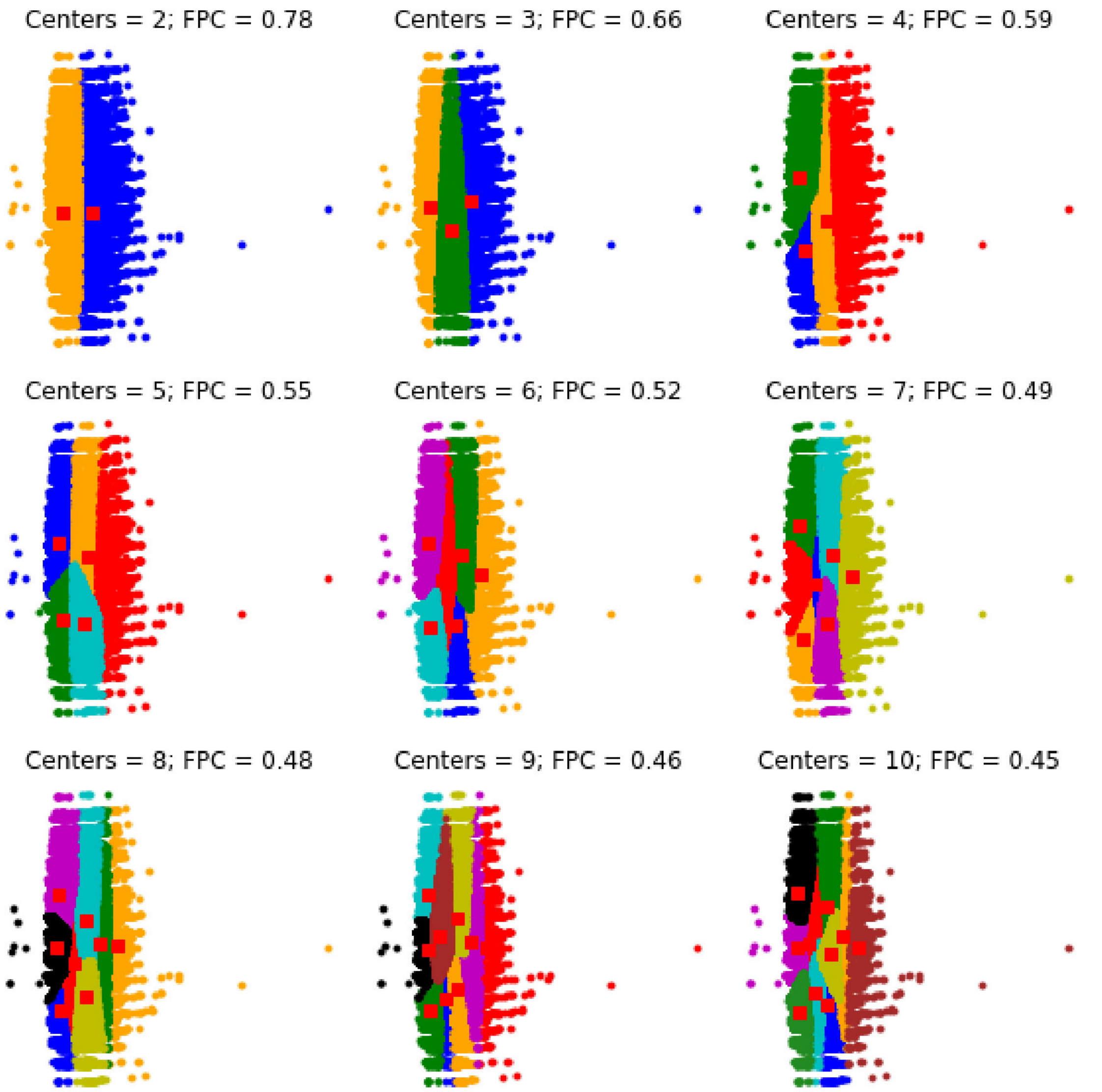
```
import skfuzzy as fuzz
```

In [107]:

```
colors = ['b', 'orange', 'g', 'r', 'c', 'm', 'y', 'k', 'Brown', 'ForestGreen']
```

In [108]:

```
fig1, axes1 = plt.subplots(3, 3, figsize=(8, 8))
alldata = np.vstack((xpts, ypts))
fpcs = []

for ncenters, ax in enumerate(axes1.reshape(-1), 2):
    cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(
        alldata, ncenters, 2, error=0.005, maxiter=1000, init=None)

    # Store fpc values for later
    fpcs.append(fpc)

    # Plot assigned clusters, for each data point in training set
    cluster_membership = np.argmax(u, axis=0)
    for j in range(ncenters):
        ax.plot(xpts[cluster_membership == j],
                ypts[cluster_membership == j], '.', color=colors[j])

    # Mark the center of each fuzzy cluster
    for pt in cntr:
        ax.plot(pt[0], pt[1], 'rs')

    ax.set_title('Centers = {0}; FPC = {1:.2f}'.format(ncenters, fpc))
    ax.axis('off')

fig1.tight_layout()
```

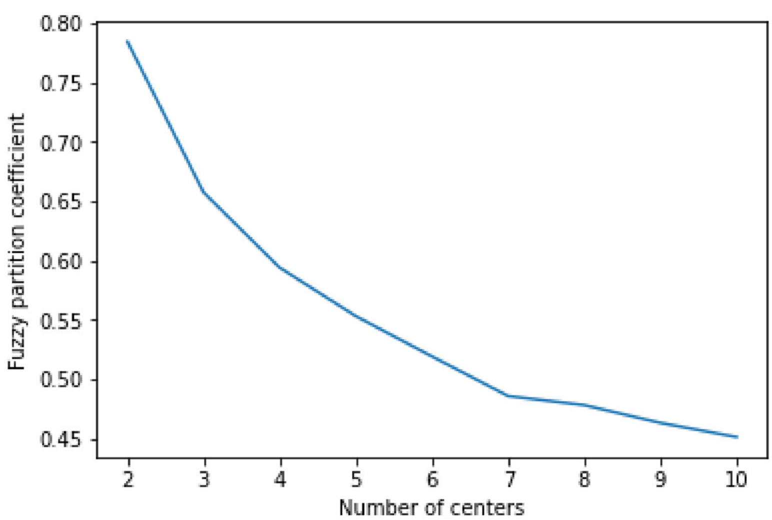


In [109]:

```
fig2, ax2 = plt.subplots()
ax2.plot(np.r_[2:11], fpcs)
ax2.set_xlabel("Number of centers")
ax2.set_ylabel("Fuzzy partition coefficient")
```

Out[109]:

```
Text(0, 0.5, 'Fuzzy partition coefficient')
```



最佳的cluster數量是3

In [110]:

```
from sklearn import cluster, datasets
```

In [115]:

```
hclust_S = cluster.AgglomerativeClustering(linkage = 'single', affinity = 'euclidean', n_cl
hclust_C = cluster.AgglomerativeClustering(linkage = 'complete', affinity = 'euclidean', n_
hclust_A = cluster.AgglomerativeClustering(linkage = 'average', affinity = 'euclidean', n_c
```

In [117]:

```
hclust_S.fit(k_D)
cluster_labels = hclust_S.labels_
print(cluster_labels)
print("---")
```

```
[0 0 0 ... 0 0 0]
---
```

In [116]:

```
hclust_C.fit(k_D)
cluster_labels_C = hclust_C.labels_
print(cluster_labels_C)
print("---")
```

```
---------------------------------------------------------------------------
MemoryError                               Traceback (most recent call last)
<ipython-input-116-9fd5b5d13392> in <module>
----> 1 hclust_C.fit(k_D)
      2 cluster_labels_C = hclust_C.labels_
      3 print(cluster_labels_C)
      4 print("---")
      5 hclust_A.fit(k_D)

c:\users\user\venv\lib\site-packages\sklearn\cluster\_agglomerative.py in fi
t(self, X, y)
    895         )
    896
--> 897         out = memory.cache(tree_builder)(X, connectivity=connectivit
y,
    898                                          n_clusters=n_clusters,
    899                                          return_distance=return_dist
ance,

c:\users\user\venv\lib\site-packages\joblib\memory.py in __call__(self, *arg
s, **kwargs)
    350
    351     def __call__(self, *args, **kwargs):
--> 352         return self.func(*args, **kwargs)
    353
    354     def call_and_shelve(self, *args, **kwargs):

c:\users\user\venv\lib\site-packages\sklearn\cluster\_agglomerative.py in _c
omplete_linkage(*args, **kwargs)
    607 def _complete_linkage(*args, **kwargs):
    608     kwargs['linkage'] = 'complete'
--> 609     return linkage_tree(*args, **kwargs)
    610
    611

c:\users\user\venv\lib\site-packages\sklearn\cluster\_agglomerative.py in li
nkage_tree(X, connectivity, n_clusters, linkage, affinity, return_distance)
    491             out = _hierarchical.single_linkage_label(mst)
    492         else:
--> 493             out = hierarchy.linkage(X, method=linkage, metric=affini
ty)
    494         children_ = out[:, :2].astype(int, copy=False)
    495

c:\users\user\venv\lib\site-packages\scipy\cluster\hierarchy.py in linkage
(y, method, metric, optimal_ordering)
   1058                           'matrix looks suspiciously like an unconden
sed '
   1059                           'distance matrix')
-> 1060         y = distance.pdist(y, metric)
   1061     else:
   1062         raise ValueError("`y` must be 1 or 2 dimensional.")
```

```
c:\users\user\venv\lib\site-packages\scipy\spatial\distance.py in pdist(X, m
etric, *args, **kwargs)
   2021     out = kwargs.pop("out", None)
   2022     if out is None:
-> 2023         dm = np.empty((m * (m - 1)) // 2, dtype=np.double)
   2024     else:
   2025         if out.shape != (m * (m - 1) // 2,):

MemoryError: Unable to allocate 10.8 GiB for an array with shape (145473483
0,) and data type float64
```

In [118]:

```
hclust_A.fit(k_D)
cluster_labels_A = hclust_A.labels_
print(cluster_labels_A)
print("---")
```

```
---------------------------------------------------------------------------
MemoryError                               Traceback (most recent call last)
<ipython-input-118-75c1a340ae4b> in <module>
----> 1 hclust_A.fit(k_D)
      2 cluster_labels_A = hclust_A.labels_
      3 print(cluster_labels_A)
      4 print("---")

c:\users\user\venv\lib\site-packages\sklearn\cluster\_agglomerative.py in fi
t(self, X, y)
    895         )
    896
--> 897         out = memory.cache(tree_builder)(X, connectivity=connectivit
y,
    898                                          n_clusters=n_clusters,
    899                                          return_distance=return_dist
ance,

c:\users\user\venv\lib\site-packages\joblib\memory.py in __call__(self, *arg
s, **kwargs)
    350
    351     def __call__(self, *args, **kwargs):
--> 352         return self.func(*args, **kwargs)
    353
    354     def call_and_shelve(self, *args, **kwargs):

c:\users\user\venv\lib\site-packages\sklearn\cluster\_agglomerative.py in _a
verage_linkage(*args, **kwargs)
    612 def _average_linkage(*args, **kwargs):
    613     kwargs['linkage'] = 'average'
--> 614     return linkage_tree(*args, **kwargs)
    615
    616

c:\users\user\venv\lib\site-packages\sklearn\cluster\_agglomerative.py in li
nkage_tree(X, connectivity, n_clusters, linkage, affinity, return_distance)
    491             out = _hierarchical.single_linkage_label(mst)
    492         else:
--> 493             out = hierarchy.linkage(X, method=linkage, metric=affini
ty)
    494         children_ = out[:, :2].astype(int, copy=False)
    495

c:\users\user\venv\lib\site-packages\scipy\cluster\hierarchy.py in linkage
(y, method, metric, optimal_ordering)
   1058                           'matrix looks suspiciously like an unconden
sed '
   1059                           'distance matrix')
-> 1060         y = distance.pdist(y, metric)
   1061     else:
   1062         raise ValueError("`y` must be 1 or 2 dimensional.")

c:\users\user\venv\lib\site-packages\scipy\spatial\distance.py in pdist(X, m
etric, *args, **kwargs)
```

```
   2021     out = kwargs.pop("out", None)
   2022     if out is None:
-> 2023         dm = np.empty((m * (m - 1)) // 2, dtype=np.double)
   2024     else:
   2025         if out.shape != (m * (m - 1) // 2,):

MemoryError: Unable to allocate 10.8 GiB for an array with shape (145473483
0,) and data type float64
```

資料集太大，跑不出來

In [ ]: