利用 pandas 套件將資料讀取進來，透過 info()來看 character-deaths.csv 的資訊，如欄位的型態(object、float64…等)、資料的數量…等。

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
from pandas import Series,DataFrame
import matplotlib.pyplot as plt

data = pd.read_csv("character-deaths.csv")
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 917 entries, 0 to 916
Data columns (total 13 columns):
Name                  917 non-null object
Allegiances           917 non-null object
Death Year            305 non-null float64
Book of Death         307 non-null float64
Death Chapter         299 non-null float64
Book Intro Chapter    905 non-null float64
Gender                917 non-null int64
Nobility              917 non-null int64
GoT                   917 non-null int64
CoK                   917 non-null int64
SoS                   917 non-null int64
FfC                   917 non-null int64
DwD                   917 non-null int64
dtypes: float64(4), int64(7), object(2)
memory usage: 93.3+ KB
```

再來進入資料前處理的階段；先利用 isna().sum()來看哪些欄位是有空值存在。

```
In [2]: #把空值以0替代
        data.isna().sum()
```

```
Out[2]: Name                    0
        Allegiances             0
        Death Year            612
        Book of Death         610
        Death Chapter         618
        Book Intro Chapter     12
        Gender                  0
        Nobility                0
        GoT                     0
        CoK                     0
        SoS                     0
        FfC                     0
        DwD                     0
        dtype: int64
```

透過 fillna()將有空值的欄位全都補上 0

```
In [3]: select_data = pd.DataFrame(data)
        select_data['Death Year'] = select_data['Death Year'].fillna(0)
        select_data['Book of Death'] = select_data['Book of Death'].fillna(0)
        select_data['Death Chapter'] = select_data['Death Chapter'].fillna(0)
        select_data['Book Intro Chapter'] = select_data['Book Intro Chapter'].fillna(0)
        select_data
```

Out[3]:

	Name	Allegiances	Death Year	Book of Death	Death Chapter	Book Intro Chapter	Gender	Nobility	GoT	CoK	SoS	FfC	DwD
0	Addam Marbrand	Lannister	0.0	0.0	0.0	56.0	1	1	1	1	1	1	0
1	Aegon Frey (Jinglebell)	None	299.0	3.0	51.0	49.0	1	1	0	0	1	0	0
2	Aegon Targaryen	House Targaryen	0.0	0.0	0.0	5.0	1	1	0	0	0	0	1
3	Adrack Humble	House Greyjoy	300.0	5.0	20.0	20.0	1	1	0	0	0	0	1
4	Aemon Costayne	Lannister	0.0	0.0	0.0	0.0	1	1	0	0	1	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
912	Zollo	None	0.0	0.0	0.0	21.0	1	0	0	0	1	0	0
913	Yurkhaz zo Yunzak	None	300.0	5.0	59.0	47.0	1	0	0	0	0	0	1
914	Yezzan Zo Qaggaz	None	300.0	5.0	57.0	25.0	1	1	0	0	0	0	1
915	Torwynd the Tame	Wildling	300.0	5.0	73.0	73.0	1	0	0	0	1	0	0
916	Talbert Serry	Tyrell	300.0	4.0	29.0	29.0	1	1	0	0	0	1	0

917 rows × 13 columns

在 Death Year ， Book of Death ， Death Chapter 三者當中，我選擇 Death Year 這個欄位，透過 np.where()將原本非零數值全部轉成 1

```
In [4]: # Death Year , Book of Death , Death Chapter三者取一個，將有數值的轉成1
select_data['Death Year'] = np.where(select_data['Death Year'] > 0, 1.0, 0.0)
select_data
```

Out[4]:

	Name	Allegiances	Death Year	Book of Death	Death Chapter	Book Intro Chapter	Gender	Nobility	GoT	CoK	SoS	FfC	DwD
0	Addam Marbrand	Lannister	0.0	0.0	0.0	56.0	1	1	1	1	1	1	0
1	Aegon Frey (Jinglebell)	None	1.0	3.0	51.0	49.0	1	1	0	0	1	0	0
2	Aegon Targaryen	House Targaryen	0.0	0.0	0.0	5.0	1	1	0	0	0	0	1
3	Adrack Humble	House Greyjoy	1.0	5.0	20.0	20.0	1	1	0	0	0	0	1
4	Aemon Costayne	Lannister	0.0	0.0	0.0	0.0	1	1	0	0	1	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
912	Zollo	None	0.0	0.0	0.0	21.0	1	0	0	0	1	0	0
913	Yurkhaz zo Yunzak	None	1.0	5.0	59.0	47.0	1	0	0	0	0	0	1
914	Yezzan Zo Qaggaz	None	1.0	5.0	57.0	25.0	1	1	0	0	0	0	1
915	Torwynd the Tame	Wildling	1.0	5.0	73.0	73.0	1	0	0	0	1	0	0
916	Talbert Serry	Tyrell	1.0	4.0	29.0	29.0	1	1	0	0	0	1	0

917 rows × 13 columns

將 Allegiances 轉成 dummy 特徵，底下有幾種分類就會變成幾個特徵，值會是 0 或 1。

```
In [5]: #將Allegiances轉成dummy特徵(底下有幾種分類就會變成幾個特徵，值是0或1，本來的資料集就會再增加約20種特徵)
select_data_2 = pd.get_dummies(select_data['Allegiances'])
select_data_2
```

Out[5]:

	Arryn	Baratheon	Greyjoy	House Arryn	House Baratheon	House Greyjoy	House Lannister	House Martell	House Stark	House Targaryen	...	House Tyrell	Lannister	Martell	Night's Watch	None	Stark	Targar
0	0	0	0	0	0	0	0	0	0	0	...	0	1	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	1	0	
2	0	0	0	0	0	0	0	0	0	1	...	0	0	0	0	0	0	
3	0	0	0	0	0	1	0	0	0	0	...	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	1	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
912	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	1	0	
913	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	1	0	
914	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	1	0	
915	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
916	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	

917 rows × 21 columns

接著要將新增的特徵透過 join()加到原本的資料集當中，資料集的特徵就會變成 34 個。

```
In [6]: select_data_3 = select_data.join(select_data_2)
select_data_3.info()
```

接著亂數拆成訓練集(75%)與測試集(25%)，random_state 設為 None 是避免測試集和訓練集資料重複。

```
In [7]: #亂數拆成訓練集(75%)與測試集(25%)
from sklearn.model_selection import train_test_split
X = select_data_3.iloc[:,5:34]
y = select_data_3.loc[:,"Death Year"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=None)
X_test
```

使用 scikit-learn 的 DecisionTreeClassifier 進行預測，樹的深度限制為 3，進行高度限制是怕訓練出的模型會有 overfitting 的問題；在此條件下，預測出的結果準確率大概為 72%

In [8]:
```
#使用scikit-Learn的DecisionTreeClassifier進行預測
#criterion : optional (default="gini") or Choose attribute selection measure: This parameter allows us to use the different
#           attribute selection measure. Supported criteria are "gini" for the Gini index and "entropy" for the information gai
#max_depth : int or None, optional (default=None) or Maximum Depth of a Tree: The maximum depth of the tree.
#           The higher value of maximum depth causes overfitting, and a lower value causes underfitting (Source).
from sklearn import tree
from sklearn import metrics
clf = tree.DecisionTreeClassifier(criterion="entropy", max_depth=3)
game_clf = clf.fit(X_train, y_train)
predicted = game_clf.predict(X_test)
accuracy = metrics.accuracy_score(predicted, y_test)
print(accuracy)
```

```
0.717391304347826
```

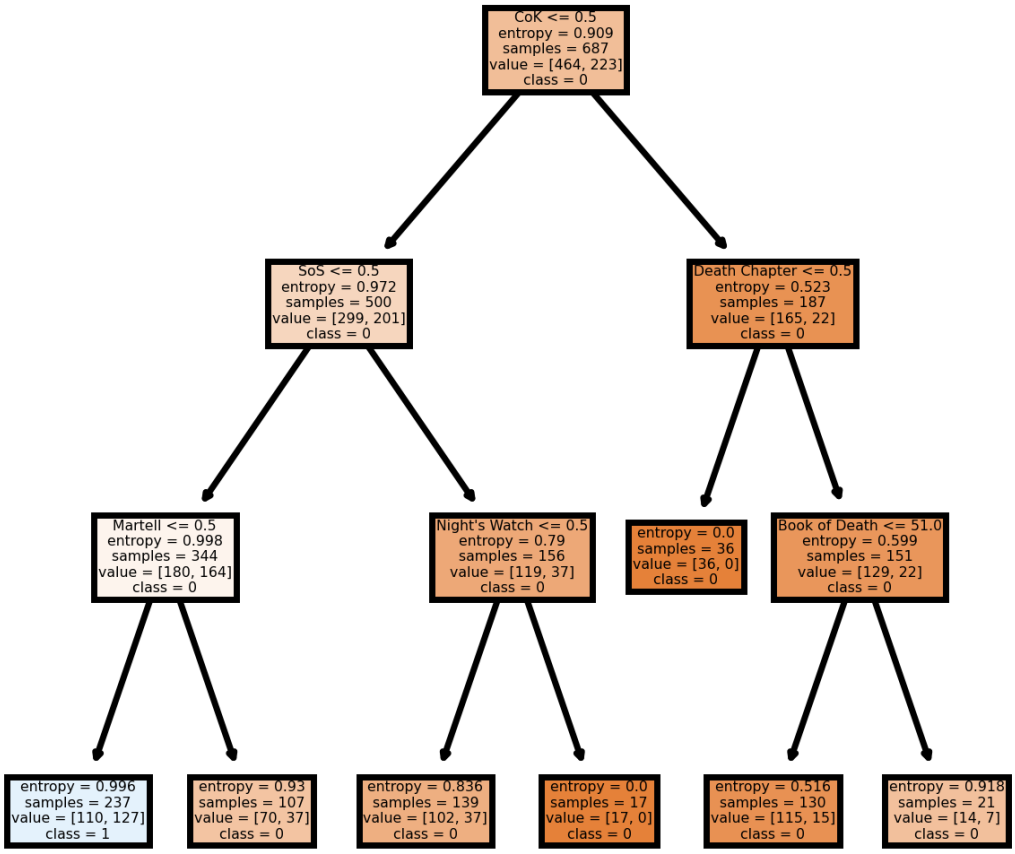
利用 sklearn.metrics 做出 Confusion Matrix，並計算出 Precision, Recall, Accuracy

In [9]:
```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, predicted))
print(classification_report(y_test, predicted))
```

```
[[113  35]
 [ 30  52]]
              precision    recall  f1-score   support

         0.0       0.79      0.76      0.78       148
         1.0       0.60      0.63      0.62        82

    accuracy                           0.72       230
   macro avg       0.69      0.70      0.70       230
weighted avg       0.72      0.72      0.72       230
```

利用 matplotlib.pyplot 產出決策樹的圖

In [10]:
```
#產出決策樹的圖(限制樹的深度=3)
fn = ["Book of Death","Death Chapter","Book Intro Chapter","Gender","Nobility","GoT","CoK","SoS","FfC","DwD","Arryn","Baratheon",
cn = ["0","1"]
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (3,3), dpi=500)

tree.plot_tree(game_clf,feature_names = fn,
               class_names=cn,
               filled = True);
fig.savefig('imagename.png')
```


CoK <= 0.5
entropy = 0.909
samples = 687
value = [464, 223]
class = 0
SoS <= 0.5
entropy = 0.972
samples = 500
value = [299, 201]
class = 0
Death Chapter <= 0.5
entropy = 0.523
samples = 187
value = [165, 22]
class = 0
Martell <= 0.5
entropy = 0.998
samples = 344
value = [180, 164]
class = 0
Night's Watch <= 0.5
entropy = 0.79
samples = 156
value = [119, 37]
class = 0
entropy = 0.0
samples = 36
value = [36, 0]
class = 0
Book of Death <= 51.0
entropy = 0.599
samples = 151
value = [129, 22]
class = 0
entropy = 0.996
samples = 237
value = [110, 127]
class = 1
entropy = 0.93
samples = 107
value = [70, 37]
class = 0
entropy = 0.836
samples = 139
value = [102, 37]
class = 0
entropy = 0.0
samples = 17
value = [17, 0]
class = 0
entropy = 0.516
samples = 130
value = [115, 15]
class = 0
entropy = 0.918
samples = 21
value = [14, 7]
class = 0

不過當樹的深度調整為 4 時，準確率並無明顯提升。

```
from sklearn import tree
from sklearn import metrics
clf = tree.DecisionTreeClassifier(criterion="entropy", max_depth=4)
game_clf = clf.fit(X_train, y_train)
predicted = game_clf.predict(X_test)
accuracy = metrics.accuracy_score(predicted, y_test)
print(accuracy)
```

```
0.7043478260869566
```

In [19]:
```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, predicted))
print(classification_report(y_test, predicted))
```

```
[[110  41]
 [ 27  52]]
              precision    recall  f1-score   support

         0.0       0.80      0.73      0.76       151
         1.0       0.56      0.66      0.60        79

    accuracy                           0.70       230
   macro avg       0.68      0.69      0.68       230
weighted avg       0.72      0.70      0.71       230
```