- [1][2]：import packages

```
In [1]: import pandas as pd
        import numpy as np
        from sklearn.model_selection import train_test_split
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.tree import plot_tree

        from sklearn.feature_selection import RFECV
        from sklearn.model_selection import cross_val_score, StratifiedKFold, learning_curve, train_test_split, GridSearchCV

        from sklearn.preprocessing import LabelEncoder
        from sklearn import metrics
        import matplotlib.pyplot as plt
        import warnings
        warnings.filterwarnings('ignore')
```

```
In [2]: import sklearn
        print(sklearn.__version__)

        0.24.2
```

- [3]：讀取 game of throne data

  [4]：看各資料裡面包含甚麼欄位

```
In [3]: battle = pd.read_csv('./data/battles.csv')
        death = pd.read_csv('./data/character-deaths.csv')
        prediction = pd.read_csv('./data/character-predictions.csv')
```

```
In [4]: battle.columns
        # ['name', 'year', 'battle_number', 'attacker_king', 'defender_king', 'attacker_1', 'attacker_2', 'attacker_3', 'attacker_4',
        #  'defender_1', 'defender_2', 'defender_3', 'defender_4', 'attacker_outcome', 'battle_type', 'major_death', 'major_capture',
        #  'attacker_size','defender_size', 'attacker_commander', 'defender_commander', 'summer', 'location', 'region', 'note']

        death.columns
        # ['Name', 'Allegiances', 'Death Year', 'Book of Death', 'Death Chapter', 'Book Intro Chapter', 'Gender', 'Nobility',
        # 'GoT', 'CoK', 'SoS', 'FfC', 'DwD']

        prediction.columns
        # ['S.No', 'actual', 'pred', 'alive', 'plod', 'name', 'title', 'male', 'culture', 'dateOfBirth', 'DateoFdeath',
        #  'mother', 'father', 'heir', 'house', 'spouse', 'book1', 'book2', 'book3', 'book4', 'book5', 'isAliveMother',
        #  'isAliveFather', 'isAliveHeir', 'isAliveSpouse', 'isMarried', 'isNoble', 'age', 'numDeadRelations', 'boolDeadRelations',
        #  'isPopular', 'popularity', 'isAlive']

Out[4]: Index(['S.No', 'actual', 'pred', 'alive', 'plod', 'name', 'title', 'male',
               'culture', 'dateOfBirth', 'DateoFdeath', 'mother', 'father', 'heir',
               'house', 'spouse', 'book1', 'book2', 'book3', 'book4', 'book5',
               'isAliveMother', 'isAliveFather', 'isAliveHeir', 'isAliveSpouse',
               'isMarried', 'isNoble', 'age', 'numDeadRelations', 'boolDeadRelations',
```

- [6]：查看 character-deaths.csv 中的資料各欄位有沒有缺失值，可以看到 Death Year, Book of Death, Death Chapter 有將近 2/3 的值為 null，Book Intro Chapter 則有 12 筆資料為 null

```
In [6]: death.info()

        <class 'pandas.core.frame.DataFrame'>
        RangeIndex: 917 entries, 0 to 916
        Data columns (total 13 columns):
         #   Column              Non-Null Count  Dtype
        ---  ------              --------------  -----
         0   Name                917 non-null    object
         1   Allegiances         917 non-null    object
         2   Death Year          305 non-null    float64
         3   Book of Death       307 non-null    float64
         4   Death Chapter       299 non-null    float64
         5   Book Intro Chapter  905 non-null    float64
         6   Gender              917 non-null    int64
         7   Nobility            917 non-null    int64
         8   GoT                 917 non-null    int64
         9   CoK                 917 non-null    int64
         10  SoS                 917 non-null    int64
         11  FfC                 917 non-null    int64
         12  DwD                 917 non-null    int64
        dtypes: float64(4), int64(7), object(2)
        memory usage: 93.3+ KB
```

- [8]：將 data 中的 Death Year , Book of Death , Death Chapter，以 1/0 表示有沒有值，並分別存 has_DeathYear, has_BookofDeath , has_DeathChapter

In [8]:
```
data = death
# data = data_attack
# data = data_defend

# 將data中的'Death Year' , 'Book of Death' , 'Death Chapter'，以1/0表示有沒有值，
# 並存為'has_DeathYear' , 'has_BookofDeath' , 'has_DeathChapter'
data['has_DeathYear'] = 1
data['has_BookofDeath'] = 1
data['has_DeathChapter'] = 1

data.loc[data['Death Year'].isna()==True, 'has_DeathYear'] = 0
data.loc[data['Book of Death'].isna()==True, 'has_BookofDeath'] = 0
data.loc[data['Death Chapter'].isna()==True, 'has_DeathChapter'] = 0
display(data[['Death Year', 'has_DeathYear' , 'Book of Death', 'has_BookofDeath' , 'Death Chapter', 'has_DeathChapter']].head(5))
```

	Death Year	has_DeathYear	Book of Death	has_BookofDeath	Death Chapter	has_DeathChapter
0	NaN	0	NaN	0	NaN	0
1	299.0	1	3.0	1	51.0	1
2	NaN	0	NaN	0	NaN	0
3	300.0	1	5.0	1	20.0	1
4	NaN	0	NaN	0	NaN	0

- [9]：將 Death Year , Book of Death , Death Chapter, Book Intro Chapter 中的 nan 以 0 填上

In [9]:
```
# 空值填上0
data['Death Year'] = data['Death Year'].fillna(0)
data['Book of Death'] = data['Book of Death'].fillna(0)
data['Death Chapter'] = data['Death Chapter'].fillna(0)
data['Book Intro Chapter'] = data['Book Intro Chapter'].fillna(0)

display(data.head(5))
```

	Name	Allegiances	Death Year	Book of Death	Death Chapter	Book Intro Chapter	Gender	Nobility	GoT	CoK	SoS	FfC	DwD	has_DeathYear	has_BookofDeath	has_DeathChapter
0	Addam Marbrand	Lannister	0.0	0.0	0.0	56.0	1	1	1	1	1	1	0	0	0	0
1	Aegon Frey (Jinglebell)	None	299.0	3.0	51.0	49.0	1	1	0	0	1	0	0	1	1	1
2	Aegon Targaryen	House Targaryen	0.0	0.0	0.0	5.0	1	1	0	0	0	0	1	0	0	0
3	Adrack	House	300.0	5.0	20.0	20.0	1	1	0	0	0	0	1	1	1	1

- [10]：用 LabelEncoder 的 fit_transform 將 data 中的 Allegiances 編碼，將原本 21 種 text 型態的名稱轉換成 0~20 的整數並存為 Alle

In [10]:
```
label = LabelEncoder()
data['Alle'] = label.fit_transform(data['Allegiances'])
display(data[['Allegiances','Alle']])
# display(data['Gender'])
```

	Allegiances	Alle
0	Lannister	12
1	None	15
2	House Targaryen	9
3	House Greyjoy	5
4	Lannister	12
...	...	...
912	None	15
913	None	15
914	None	15
915	Wildling	20

- [11]：將 Allegiances 轉成 dummy 型態，可以看到 column 數從原本的 17 變成 38(Allegiances 總共有 21 種)
  [12]：按照 0.75/0.25 的比例，隨機切分訓練集和測試集

```
In [11]: # 將資料中的Allegiances特徵轉為dummy型態
Allegiance = data['Allegiances'].unique()
print('num of Allegiances = {}'.format(len(Allegiance)))
print('original num of data columns = {}'.format(len(data.columns)))

data = pd.concat([data, pd.get_dummies(data['Allegiances'])], axis=1)
data.columns
print('new num of data columns = {}'.format(len(data.columns)))
```

```
num of Allegiances = 21
original num of data columns = 17
new num of data columns = 38
```

```
In [12]: # 切分訓練集和測試集
train, test = train_test_split(data, test_size=0.25)
print('data size = {}'.format(len(data)))
print('train size = {}'.format(len(train)))
print('test size = {}'.format(len(test)))
# display(train.head(5))
```

```
data size = 917
train size = 687
test size = 230
```

- [13]：採用 feature 作為決策樹建模的參考特徵，predict_var 則為預測欄位

```
In [13]: # 使用決策樹
predict_var = 'Death Year' , 'Book of Death' , 'Death Chapter'
# feature = ['Book Intro Chapter', 'Gender', 'Nobility', 'GoT', 'CoK', 'SoS', 'FfC',
#            'DwD', 'has_DeathYear', 'has_BookofDeath', 'has_DeathChapter', 'Arryn',
#        'Baratheon', 'Greyjoy', 'House Arryn', 'House Baratheon',
#        'House Greyjoy', 'House Lannister', 'House Martell', 'House Stark',
#        'House Targaryen', 'House Tully', 'House Tyrell', 'Lannister',
#        'Martell', 'Night\'s Watch', 'None', 'Stark', 'Targaryen', 'Tully',
#        'Tyrell', 'Wildling']
feature = ['Alle','has_DeathYear', 'has_BookofDeath', 'has_DeathChapter']
# feature = ['Alle','has_DeathYear', 'has_BookofDeath', 'has_DeathChapter','GoT','CoK','SoS','FfC','DwD','Gender','Nobility',
#            'Book Intro Chapter','']
```

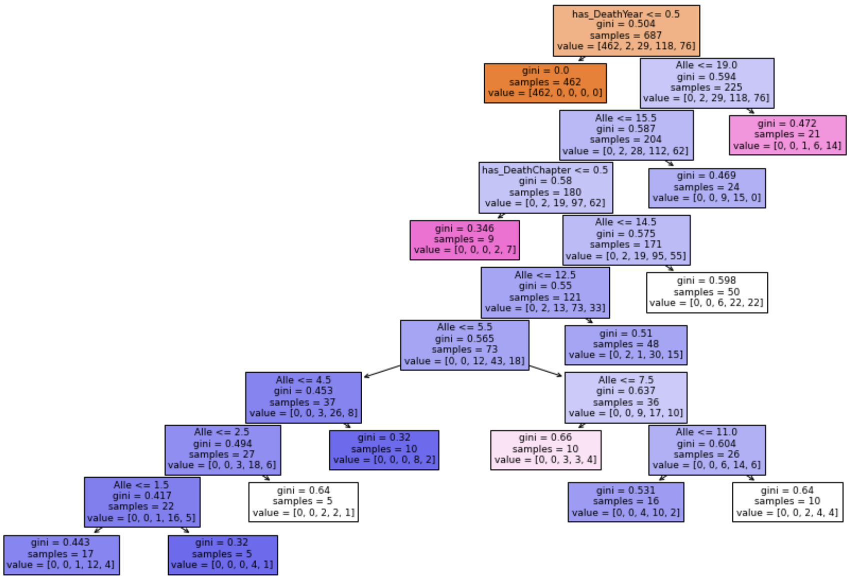
- [14]：迴圈 50 次，隨機切分訓練集和測試集，x1 為訓練資料(訓練集中的訓練參考特徵)，y1 為 x1 訓練資料的答案(Death Year 的答案)。clf_1 為決策樹宣告，將最大深度設定為 15，並 fit x1 和 y1 做訓練。最後 y1_prediction 為使用該模型去預測測試集的結果，並將 accuracy, precision, recall 存入 list 中。最後將 50 次的結果平均作為輸出，可以看到測試集的 Death Year 預測準確率 accuracy, precision, recall 均介於 0.84 左右。

```
In [14]: result_acc, result_prec, result_rec = [], [], []

for i in range(50):
    train, test = train_test_split(data, test_size=0.25)

    x1 = train[feature]
    y1 = train[predict_var[0]]
    clf_1 = DecisionTreeClassifier(random_state=i,max_depth=15, min_samples_leaf=5)
    clf_1 = clf_1.fit(x1,y1)

    y1_prediction = clf_1.predict(test[feature])


    accuracy_1 = metrics.accuracy_score(test[predict_var[0]], y1_prediction).round(4)
    precision_1 = metrics.precision_score(test[predict_var[0]], y1_prediction,average='weighted').round(4)
    recall_1 = metrics.recall_score(test[predict_var[0]], y1_prediction,average='weighted').round(4)

    result_acc.append(accuracy_1)
    result_prec.append(precision_1)
    result_rec.append(recall_1)

print(statistics.mean(result_acc))
print(statistics.mean(result_prec))
print(statistics.mean(result_rec))
```

```
0.838344
0.835744
0.838344
```

- [15]：利用 plot_tree 畫出決策樹(預測欄位為 Death Year)

In [15]:
```python
plt.figure(figsize=(15, 10))
plot_tree(clf_1, feature_names=feature, filled=True)
plt.show()
```



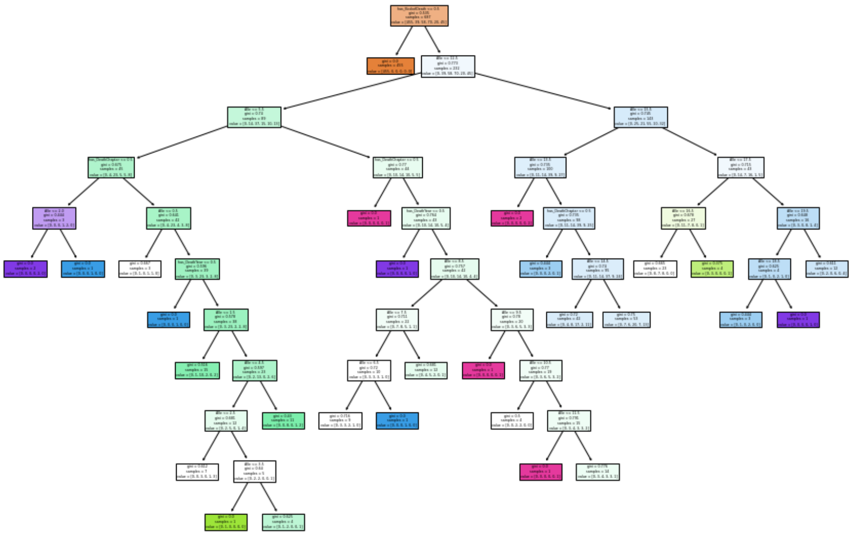
- [16]：以同樣的方法預測測試集的 Book of Death，可以看到 accuracy, precision, recall 均介於 0.79 左右。

In [16]:
```python
result_acc, result_prec, result_rec = [], [], []

for i in range(50):
    train, test = train_test_split(data, test_size=0.25)
    x2 = train[feature]
    y2 = train[predict_var[1]]

    clf_2 = DecisionTreeClassifier(random_state=i, max_depth=15)
    clf_2 = clf_2.fit(x2,y2)

    y2_prediction = clf_2.predict(test[feature])

    accuracy_2 = metrics.accuracy_score(test[predict_var[1]], y2_prediction).round(4)
    precision_2 = metrics.precision_score(test[predict_var[1]], y2_prediction,average='weighted').round(4)
    recall_2 = metrics.recall_score(test[predict_var[1]], y2_prediction,average='weighted').round(4)

    result_acc.append(accuracy_2)
    result_prec.append(precision_2)
    result_rec.append(recall_2)

print(statistics.mean(result_acc))
print(statistics.mean(result_prec))
print(statistics.mean(result_rec))
```

```
0.78679
0.792498
0.78679
```

- [17]：利用 plot_tree 畫出決策樹(預測欄位為 Book of Death)

In [17]:
```
plt.figure(figsize=(15, 10))
plot_tree(clf_2, feature_names=feature, filled=True)
plt.show()
```



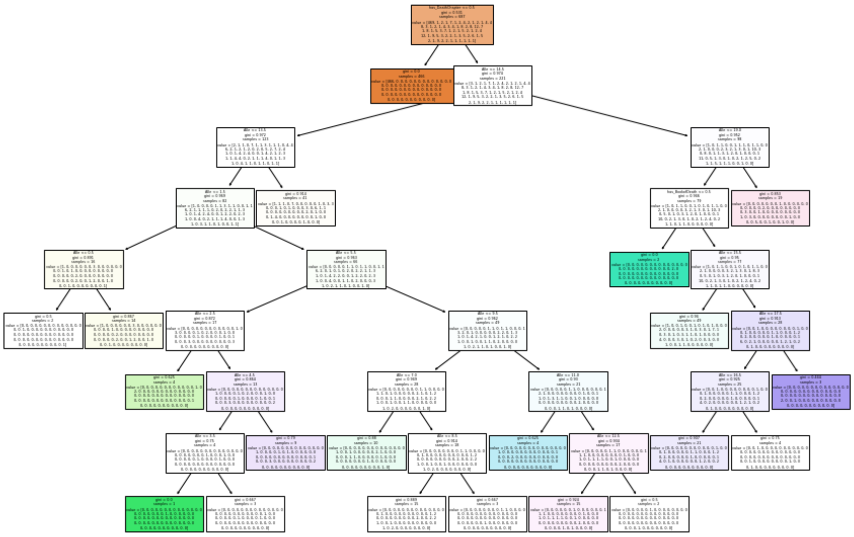
- [18]：以同樣的方法預測測試集的 Death Chapter，可以看到 accuracy, precision, recall 均介於 0.70 左右。

In [18]:
```
result_acc, result_prec, result_rec = [], [], []

for i in range(50):
    train, test = train_test_split(data, test_size=0.25)

    x3 = train[feature]
    y3 = train[predict_var[2]]

    clf_3 = DecisionTreeClassifier(max_depth=15)
    clf_3 = clf_3.fit(x3,y3)

    y3_prediction = clf_3.predict(test[feature])

    accuracy_3 = metrics.accuracy_score(test[predict_var[2]], y3_prediction).round(4)
    precision_3 = metrics.precision_score(test[predict_var[2]], y3_prediction,average='weighted').round(4)
    recall_3 = metrics.recall_score(test[predict_var[2]], y3_prediction,average='weighted').round(4)

    result_acc.append(accuracy_3)
    result_prec.append(precision_3)
    result_rec.append(recall_3)

print(statistics.mean(result_acc))
print(statistics.mean(result_prec))
print(statistics.mean(result_rec))
```

```
0.714166
0.69807
0.714166
```

- [19]：利用 plot_tree 畫出決策樹(預測欄位為 Death Chapter)

```
In [19]: plt.figure(figsize=(15, 10))
         plot_tree(clf_3, feature_names=feature, filled=True)
         plt.show()
```



- [20]：增加建模參考特徵，加入 GoT(Appeared in first book), CoK(Appeared in second book), SoS(Appeared in third book), FfC(Appeared in fourth book), DwD(Appeared in fifth book)等更多特徵。

```
In [13]: # 使用決策樹
         predict_var = 'Death Year' , 'Book of Death' , 'Death Chapter'
         # feature = ['Book Intro Chapter', 'Gender', 'Nobility', 'GoT', 'CoK', 'SoS', 'FfC',
         #           'DwD', 'has_DeathYear', 'has_BookofDeath', 'has_DeathChapter', 'Arryn',
         #        'Baratheon', 'Greyjoy', 'House Arryn', 'House Baratheon',
         #        'House Greyjoy', 'House Lannister', 'House Martell', 'House Stark',
         #        'House Targaryen', 'House Tully', 'House Tyrell', 'Lannister',
         #        'Martell', 'Night\'s Watch', 'None', 'Stark', 'Targaryen', 'Tully',
         #        'Tyrell', 'Wildling']
         # feature = ['Alle','has_DeathYear', 'has_BookofDeath', 'has_DeathChapter']
         feature = ['Alle','has_DeathYear', 'has_BookofDeath', 'has_DeathChapter','GoT','CoK','SoS', 'FfC','DwD']

         # feature = ['Alle','has_DeathYear', 'has_BookofDeath', 'has_DeathChapter','GoT','CoK','SoS','FfC','DwD','Gender','Nobility',
         #           'Book Intro Chapter']
```

- [14]：測試集的 Death Year 預測準確率(acc, prec, rec)由 0.84 上升至約 0.95，加入人物在第幾本書中有沒有出現這個特徵對預測人物 Death Year 很有幫助

```
In [14]: result_acc, result_prec, result_rec = [], [], []

         for i in range(50):
             train, test = train_test_split(data, test_size=0.25)

             x1 = train[feature]
             y1 = train[predict_var[0]]
             clf_1 = DecisionTreeClassifier(random_state=i,max_depth=15, min_samples_leaf=5)
             clf_1 = clf_1.fit(x1,y1)

             y1_prediction = clf_1.predict(test[feature])


             accuracy_1 = metrics.accuracy_score(test[predict_var[0]], y1_prediction).round(4)
             precision_1 = metrics.precision_score(test[predict_var[0]], y1_prediction,average='weighted').round(4)
             recall_1 = metrics.recall_score(test[predict_var[0]], y1_prediction,average='weighted').round(4)

             result_acc.append(accuracy_1)
             result_prec.append(precision_1)
             result_rec.append(recall_1)

         print(statistics.mean(result_acc))
         print(statistics.mean(result_prec))
         print(statistics.mean(result_rec))

         0.94878
         0.948964
         0.94878
```

- [16]：測試集的 Book of Death 預測準確率(acc, prec, rec)由 0.79 上升至約 0.97，加入人物在第幾本書中有沒有出現這個特徵對預測人物 Book of Death 非常有幫助(有出現就不可能比那本書其他人物更早死亡)

```
In [16]: result_acc, result_prec, result_rec = [], [], []

         for i in range(50):
             train, test = train_test_split(data, test_size=0.25)
             x2 = train[feature]
             y2 = train[predict_var[1]]

             clf_2 = DecisionTreeClassifier(random_state=i, max_depth=15)
             clf_2 = clf_2.fit(x2,y2)

             y2_prediction = clf_2.predict(test[feature])

             accuracy_2 = metrics.accuracy_score(test[predict_var[1]], y2_prediction).round(4)
             precision_2 = metrics.precision_score(test[predict_var[1]], y2_prediction,average='weighted').round(4)
             recall_2 = metrics.recall_score(test[predict_var[1]], y2_prediction,average='weighted').round(4)

             result_acc.append(accuracy_2)
             result_prec.append(precision_2)
             result_rec.append(recall_2)

         print(statistics.mean(result_acc))
         print(statistics.mean(result_prec))
         print(statistics.mean(result_rec))

         0.971488
         0.9737480000000001
         0.971488
```

- [18]：測試集的 Death Chapter 預測準確率(acc, prec, rec)由 0.70 上升至約 0.73，新特徵幫助比較不大

```
In [18]: result_acc, result_prec, result_rec = [], [], []

         for i in range(50):
             train, test = train_test_split(data, test_size=0.25)

             x3 = train[feature]
             y3 = train[predict_var[2]]

             clf_3 = DecisionTreeClassifier(max_depth=15)
             clf_3 = clf_3.fit(x3,y3)

             y3_prediction = clf_3.predict(test[feature])

             accuracy_3 = metrics.accuracy_score(test[predict_var[2]], y3_prediction).round(4)
             precision_3 = metrics.precision_score(test[predict_var[2]], y3_prediction,average='weighted').round(4)
             recall_3 = metrics.recall_score(test[predict_var[2]], y3_prediction,average='weighted').round(4)

             result_acc.append(accuracy_3)
             result_prec.append(precision_3)
             result_rec.append(recall_3)

         print(statistics.mean(result_acc))
         print(statistics.mean(result_prec))
         print(statistics.mean(result_rec))

         0.738864
         0.733422
         0.738864
```