# HW1 game of thrones

## basic method

### 資料前處理：

**讀取資料**

```
from google.colab import drive
drive.mount('/content/drive')
import pandas as pd

# 讀取 test.csv
test_df = pd.read_csv('/content/drive/MyDrive/碩一上課堂/1131_d

# 讀取 train.csv
train_df = pd.read_csv('/content/drive/MyDrive/碩一上課堂/1131_
```

**將兩張表格空值設為0**

```
# 將 train.csv 中的空值替換為 0
train_df.fillna(0, inplace=True)

# 將 test.csv 中的空值替換為 0
test_df.fillna(0, inplace=True)
```

**將三個代表死亡的欄位取「Death Year」並設為 binary data**

```
# 只保留 Death Year 欄位，並將其重命名為 'death'
train_df['death'] = train_df['Death Year'].apply(lambda x: 1

# 刪除 'Death Year', 'Book of Death', 'Death Chapter' 這三個欄位
train_df.drop(columns=['Death Year', 'Book of Death', 'Death
```

**將「Alegiances」作虛擬變數轉換**

```
train_df = pd.get_dummies(train_df, columns=['Allegiances'], 
```

**將test資料做相同轉換，之後輸入模型會比較方便。**

```
# 先對 test.csv 進行 dummy 特徵轉換
test_df = pd.get_dummies(test_df, columns=['Allegiances'], dr

# 確保 test_df 中的 dummy 特徵與 train_df 保持一致
# 找出 train_df 中有而 test_df 中沒有的特徵，並補上這些特徵，設置為 0
missing_cols = set(train_df.columns) - set(test_df.columns)
for col in missing_cols:
    test_df[col] = 0

# 保持 test_df 的列順序與 train_df 一致
test_df = test_df[train_df.columns.drop('death')]
```

**把train data分為訓練及與驗證集**

```
from sklearn.model_selection import train_test_split

# 將 train_df 拆分成 75% 的訓練集和 25% 的測試集
X = train_df.drop(columns=['death'])  # 特徵資料
y = train_df['death']  # 目標標籤

# 使用 train_test_split 進行隨機拆分
X_train, X_valid, y_train, y_valid = train_test_split(X, y, t

# 檢查拆分後的資料集大小
print(f"訓練集大小: {X_train.shape}, 測試集大小: {X_valid.shape}'
```

```
訓練集大小: (515, 30), 測試集大小: (172, 30)
```

# 模型預測：

**drop掉「Name」欄位，我覺得對預測沒有影響。**

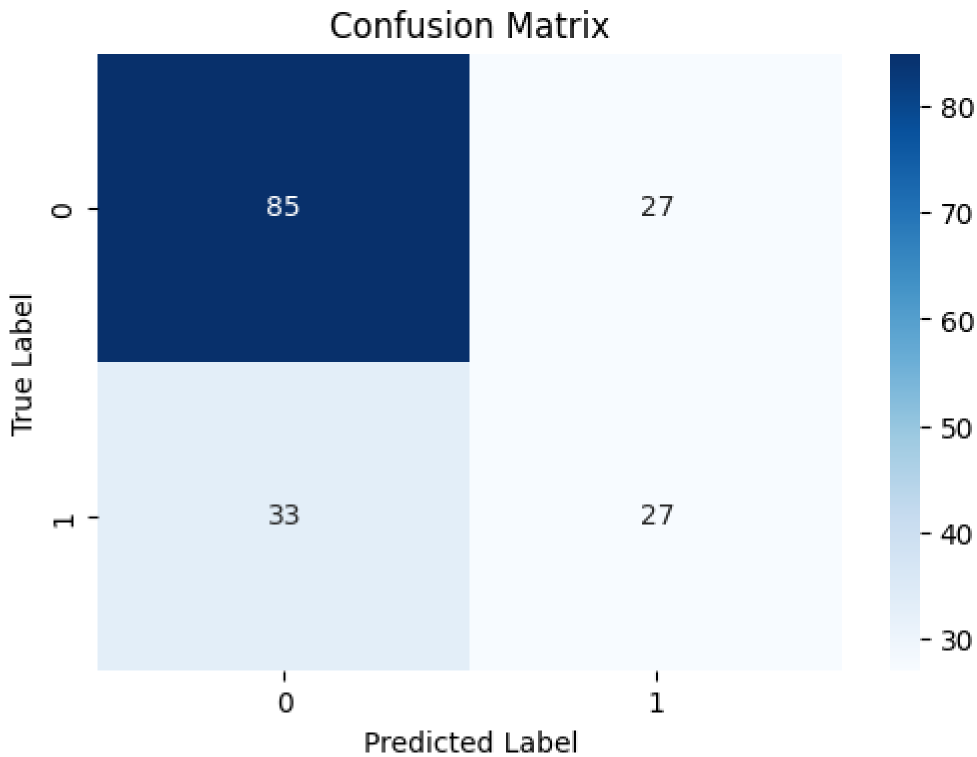
```
# 在進行模型訓練之前，刪除 'Character' 和 'Name' 欄位，因為它們對預測
X_train = X_train.drop(columns=['Character', 'Name'])
X_valid = X_valid.drop(columns=['Character', 'Name'])

# 重新進行模型訓練
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)

# 在驗證集上進行預測
y_pred = clf.predict(X_valid)

# 計算模型的準確率
accuracy = accuracy_score(y_valid, y_pred)
print(f"模型在驗證集上的準確率: {accuracy:.4f}")
```

```
模型在驗證集上的準確率: 0.6512
```

**製作混淆矩陣**

```
from sklearn.metrics import confusion_matrix, precision_score
import seaborn as sns
import matplotlib.pyplot as plt

# 生成 Confusion Matrix
conf_matrix = confusion_matrix(y_valid, y_pred)

# 可視化 Confusion Matrix
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.show()

# 計算 Precision, Recall, Accuracy
precision = precision_score(y_valid, y_pred)
recall = recall_score(y_valid, y_pred)
```

```
accuracy = accuracy_score(y_valid, y_pred)

print(f'Precision (精確率): {precision:.4f}')
print(f'Recall (召回率): {recall:.4f}')
print(f'Accuracy (準確率): {accuracy:.4f}')
```

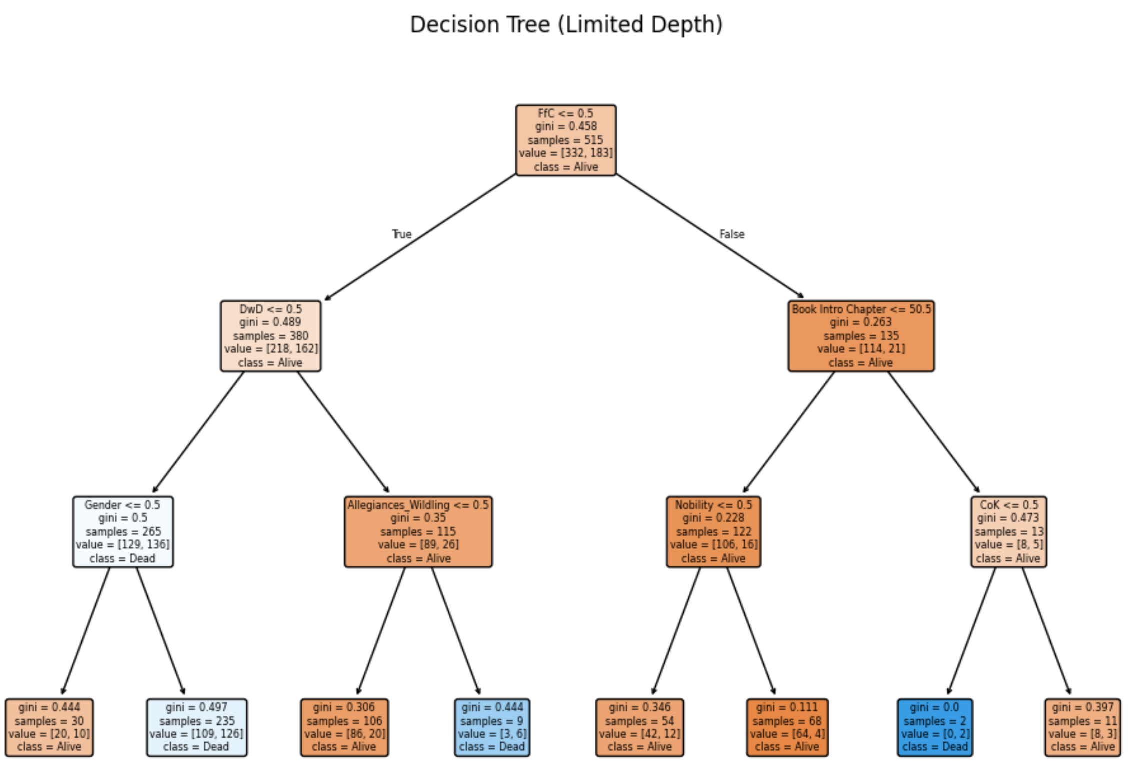


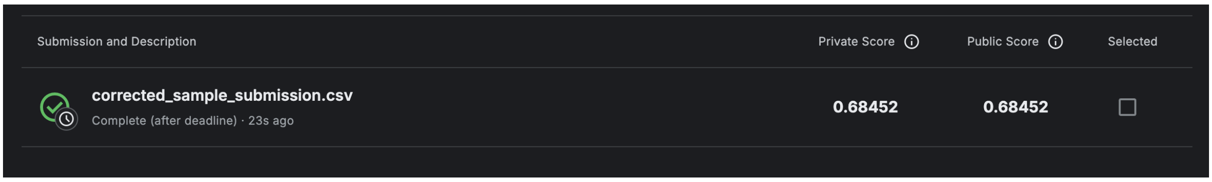

```
Precision (精確率): 0.5000
Recall (召回率): 0.4500
Accuracy (準確率): 0.6512
```

**繪製decision tree**

```
# 限制決策樹深度，這裡我們假設限制為 3
clf_limited = DecisionTreeClassifier(max_depth=3, random_stat

# 訓練模型
clf_limited.fit(X_train, y_train)

# 可視化決策樹
plt.figure(figsize=(12, 8))
```

```
tree.plot_tree(clf_limited, filled=True, feature_names=X_trai
plt.title("Decision Tree (Limited Depth)")
plt.show()
```




# advenced method

## 資料前處理：

### 戰役資料的合併：

根據家族（house）來將戰役資料合併到角色資料上，每個家族參與的戰役死亡風險被計算並加入到角色資料中。這樣你可以利用戰爭風險來影響生死的預測。

```
# 選擇相關欄位（簡單處理）
battle_risk = battles_df[['attacker_1', 'defender_1', 'major_
battle_risk['house_involved'] = battle_risk[['attacker_1', 'd

# 計算每個家族的平均戰役死亡風險
battle_risk_summary = battle_risk.groupby('house_involved')['
battle_risk_summary.rename(columns={'house_involved': 'house'
```

```
# 在合併之前，先檢查並刪除已有的 'battle_risk' 欄位
if 'battle_risk' in train_df.columns:
    train_df.drop(columns=['battle_risk'], inplace=True)

if 'battle_risk' in test_df.columns:
    test_df.drop(columns=['battle_risk'], inplace=True)

# 清理 train_df 和 test_df 中的家族名稱
train_df['house'] = train_df['house'].str.replace("House ", "
test_df['house'] = test_df['house'].str.replace("House ", "")

# 清理 battle_risk_summary 中的家族名稱
battle_risk_summary['house'] = battle_risk_summary['house'].s

# 合併戰役風險到 train_df 和 test_df
train_df = pd.merge(train_df, battle_risk_summary, on='house'
test_df = pd.merge(test_df, battle_risk_summary, on='house',

# 填補 NaN
train_df['battle_risk'].fillna(0, inplace=True)
test_df['battle_risk'].fillna(0, inplace=True)
```

**對熱門程度作min-Max處理**

```
# Min-Max Scaler 處理
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(-1, 1))
```

```
train_df['popularity'] = scaler.fit_transform(train_df[['popu
test_df['popularity'] = scaler.transform(test_df[['popularity
```

**對年齡做分箱處理**

```
# 設定年齡的分箱區間和標籤
bins = [0, 18, 30, 45, 60, 100]  # 分別表示年齡的區間
labels = ['0-18', '19-30', '31-45', '46-60', '60+']

# 檢查 train_df 是否有 'age' 欄位
if 'age' in train_df.columns:
    # 對 train_df 進行年齡分箱
    train_df['age_group'] = pd.cut(train_df['age'], bins=bins
    test_df['age_group'] = pd.cut(test_df['age'], bins=bins,

    # 將年齡分箱結果轉換為虛擬變數
    train_df = pd.get_dummies(train_df, columns=['age_group']
    test_df = pd.get_dummies(test_df, columns=['age_group'],

    # 確保 test_df 和 train_df 的欄位一致
    missing_cols = set(train_df.columns) - set(test_df.column
    for col in missing_cols:
        test_df[col] = 0

    test_df = test_df[train_df.columns.drop('death')]
```

## 模型建置，利用XGboost：

**利用XGboost並且搭配cross validation查看模型的泛化能力**

```
from sklearn.model_selection import RandomizedSearchCV
import xgboost as xgb

# 設定要調整的參數空間，增加複雜度
param_distributions = {
    'n_estimators': [400, 500, 600, 700],          # 增加樹的數
    'max_depth': [8, 9, 10, 12],                   # 增加樹的最
    'learning_rate': [0.01, 0.05, 0.1],            # 降低學習率
    'min_child_weight': [1, 2, 3],                 # 調整最小樣
```

```
    'subsample': [0.8, 0.9, 1.0],                    # 樣本抽樣比
    'colsample_bytree': [0.7, 0.8, 0.9],             # 特徵抽樣比
    'gamma': [0, 0.1, 0.2],                          # 控制葉子節
    'reg_alpha': [0, 0.01, 0.1],                     # L1 正則化
    'reg_lambda': [1, 1.5, 2.0]                      # L2 正則化
}

# 初始化 XGBoost 模型
xgb_clf = xgb.XGBClassifier(use_label_encoder=False, eval_met

# 設置 RandomizedSearchCV
random_search = RandomizedSearchCV(estimator=xgb_clf, param_d
                                   n_iter=100,  # 搜尋 100 組隨
                                   cv=5,        # 5折交叉驗證
                                   verbose=2,   # 顯示搜索過程
                                   random_state=42,
                                   n_jobs=-1)   # 使用所有可用的

# 準備訓練資料
X = train_df.drop(columns=['Character', 'Name', 'death'])
y = train_df['death']

# 處理類別變數並填補缺失值
X = pd.get_dummies(X)
X.fillna(0, inplace=True)

# 進行超參數調整
random_search.fit(X, y)

# 打印最佳參數和最佳交叉驗證結果
print("Best parameters found: ", random_search.best_params_)
print("Best cross-validation score: ", random_search.best_sco
```

```
Fitting 5 folds for each of 50 candidates, totalling 250 fits
/usr/local/lib/python3.10/dist-packages/xgboost/core.py:158: UserWarning: [13:46:52] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
Best parameters found:  {'subsample': 1.0, 'n_estimators': 200, 'min_child_weight': 5, 'max_depth': 6, 'learning_rate': 0.3,
Best cross-validation score:  0.7234422934518142
```

找到最佳超參數以及經過cross validation後得到的分數為0.723

**輸出後結果：**


分數為0.73166