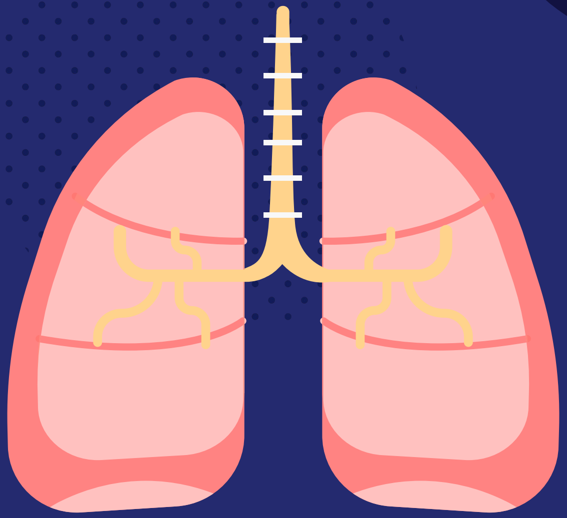


LUNG CANCER PREDICTION

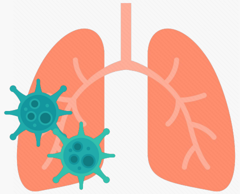
工資管113　郭哲豪

工資管113　王正廷

TABLE OF CONTENTS

MOTIVATION

隨著生活型態的轉變，人們的外在壓力日趨龐大， 肺癌患者與日遽增 。根據美國CDC顯示，吸煙者罹患肺癌或死於肺癌的機率是不吸煙者的 15 至 30 倍。英國癌症協會也 說「男性在一生中罹患肺癌的機率約為 1/15，而女性則約為 1/17 。吸煙的人風險是要高得多 。」但這兩則資訊都沒有提供它們的數據來源，所以萌生了我們想從 其他面向探討 罹患肺癌風險的想法。除此之外，若能藉由資訊預測肺癌，將能及時制定對策，使我們免受病魔之苦。



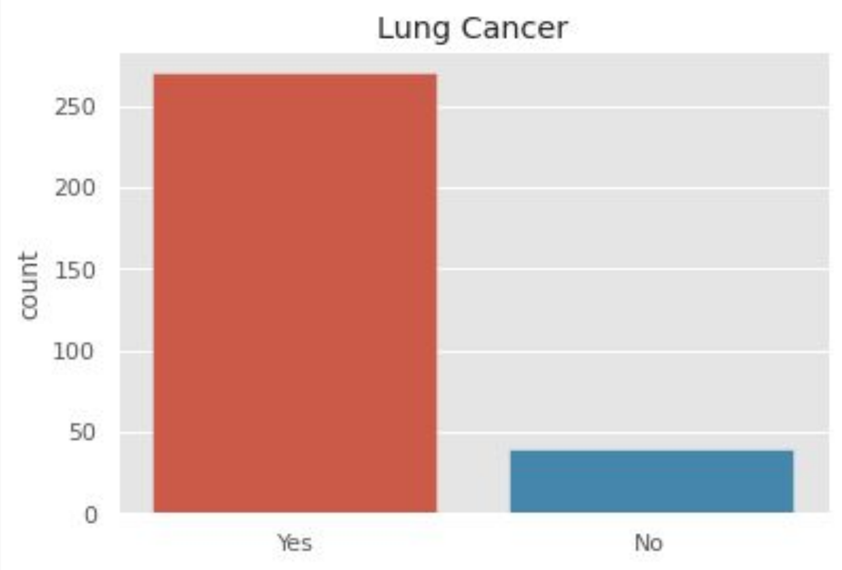
PURPOSE

藉由一些生理症狀協助人們以低成本預測罹患肺癌的風險，也幫助人們根據癌症風險狀況做出適當的決定及治療。

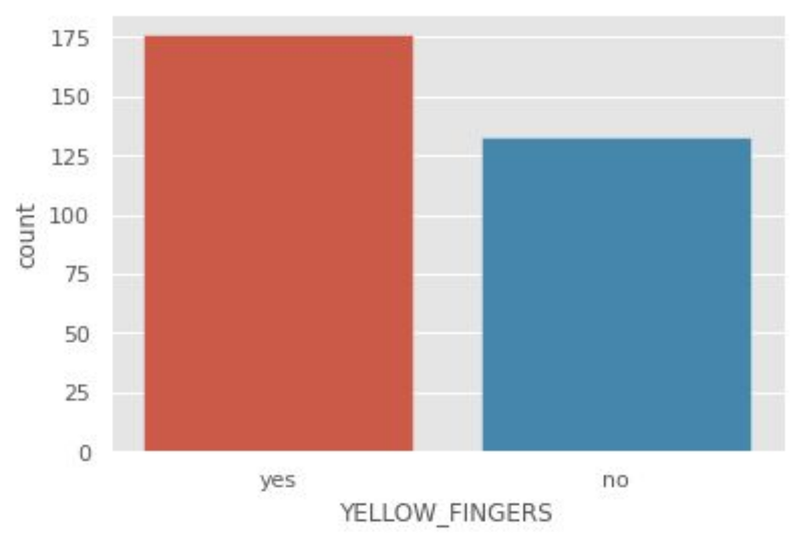
ABOUT DATASET

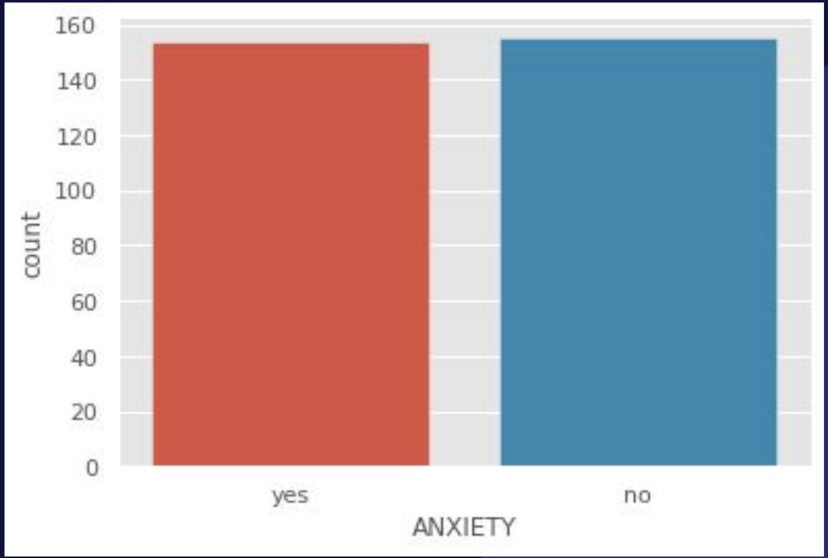
- 名稱 | Lung Cancer – Does smoking cause lung cancer
- 來源 | Kaggle
- 定義 | 本資料集共蒐集有效資料 309 筆，包含 16 種變數如：年齡、抽菸、酗酒、吞嚥困難、同儕壓力……
- 其中特別挑選黃手指、焦慮、疲憊、氣喘、咳嗽、呼吸急促、吞嚥困難、胸痛 等症狀來預測肺癌

DATA DISTRIBUTION

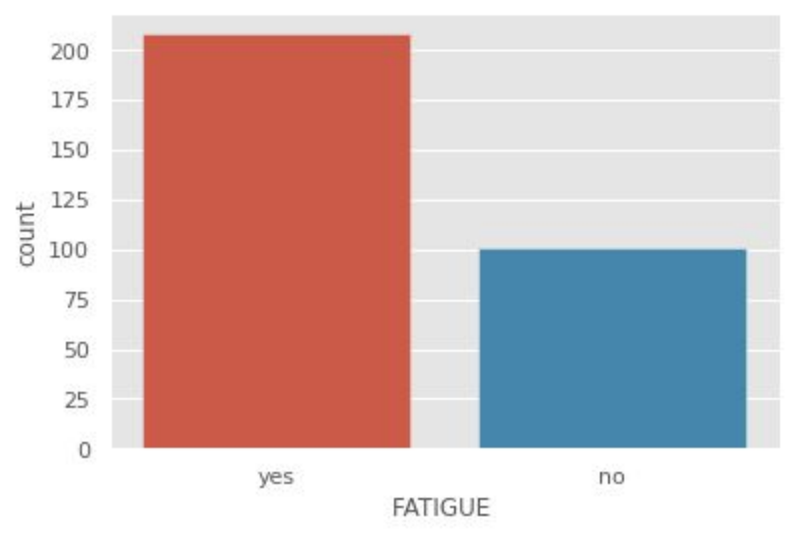


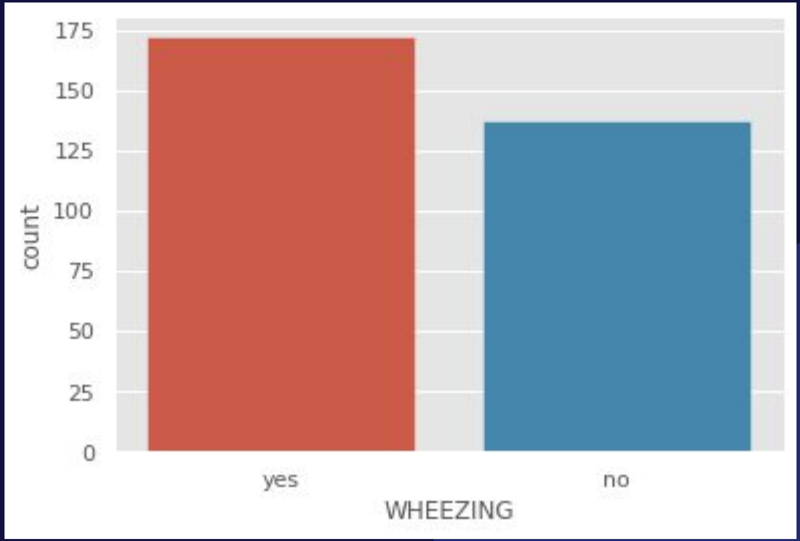
DATA DISTRIBUTION



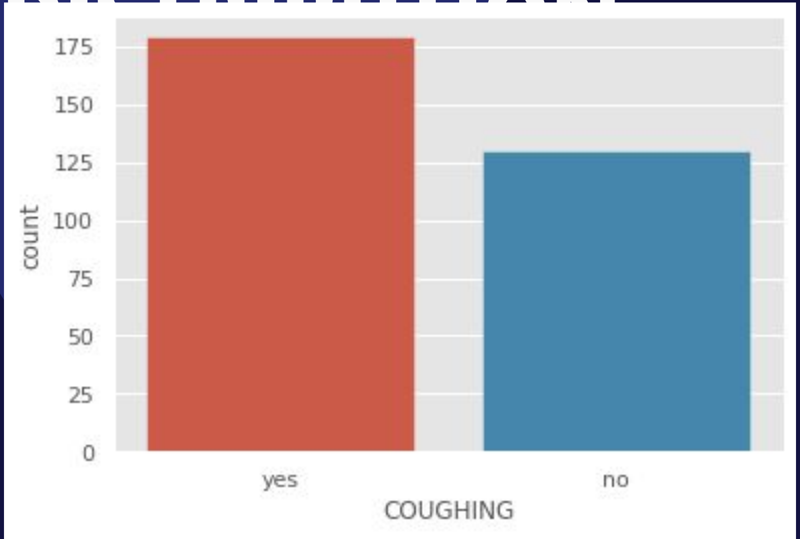


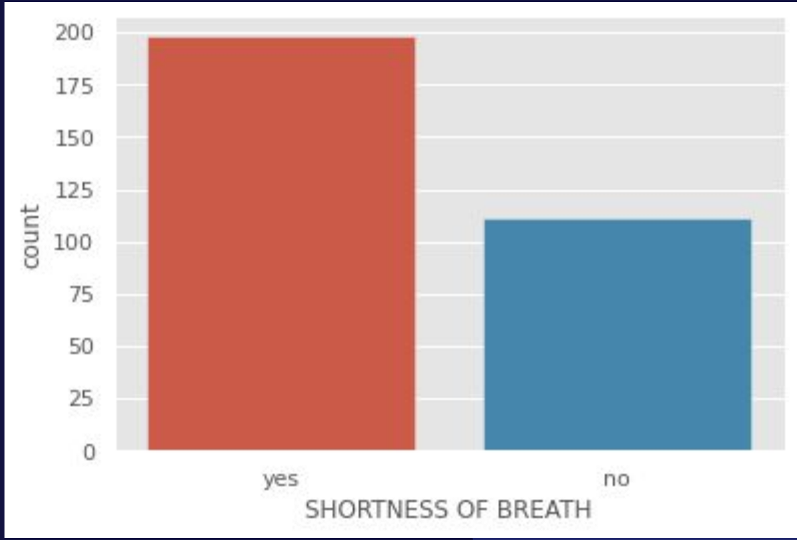
DATA DISTRIBUTION



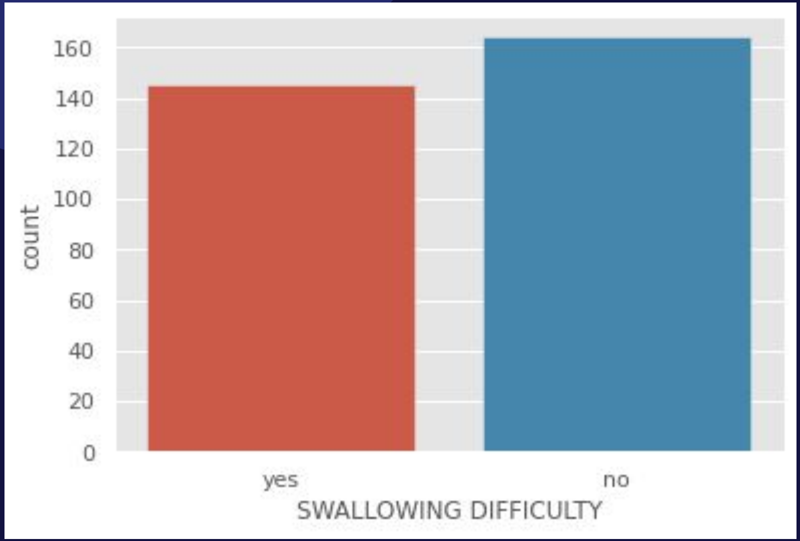


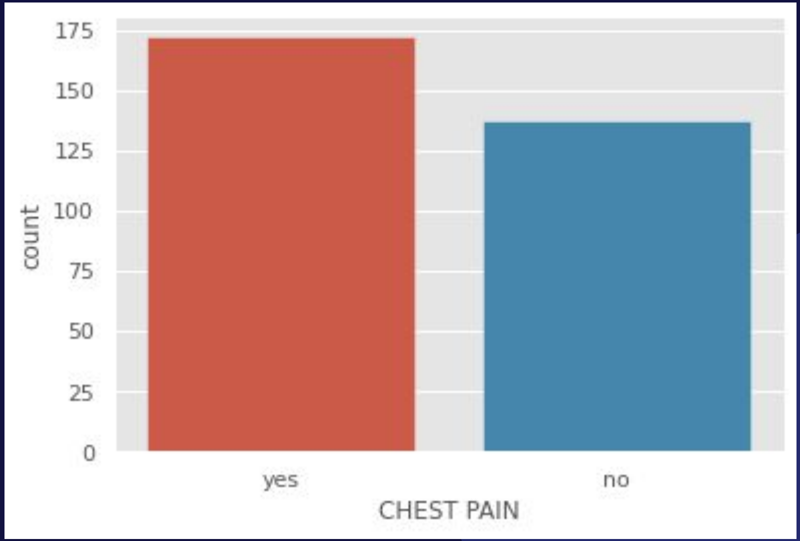
DATA DISTRIBUTION





DATA DISTRIBUTION





資料前處理

```
#資料分析
train_x = train_df[['YELLOW_FINGERS','ANXIETY', 'FATIGUE ', 'WHEEZING', 'COUGHING', 'SHORTNESS OF BREATH',
            'SWALLOWING DIFFICULTY', 'CHEST PAIN']]          # 取出訓練資料需要分析的資料欄位

train_y = train_df['LUNG_CANCER']

# 數值型態資料前處理
from sklearn.preprocessing import StandardScaler # 匯入標準化的工具


# 類別型態資料前處理
# 匯入 Label Encoder
from sklearn.preprocessing import LabelEncoder

# 創造 Label Encoder
la = LabelEncoder()
# 給予train_y類別一個數值
la.fit(train_y)
# 轉換train_y類別成為數值
train_y = la.transform(train_y)
```

資料狀態

	YELLOW_FINGERS	ANXIETY	FATIGUE	WHEEZING	COUGHING	SHORTNESS OF BREATH	SWALLOWING DIFFICULTY	CHEST PAIN
0	2	2	2	2	2	2	2	2
1	1	1	2	1	1	2	2	2
2	1	1	2	2	2	2	1	2
3	2	2	1	1	1	1	2	2
4	2	1	1	2	2	2	1	1

使用模型

- Decision Tree
- Random Forest
- SVM
- KNN
- Logistic Regression
- Adaptive Boosting

Decision Tree

```
# 匯入決策樹模型
from sklearn.tree import DecisionTreeClassifier
# 匯入準確度計算工具
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score

# 創造決策樹模型
# 設定最佳化方法為 Gini Index
# 設定最大深度為 2
# 設定最多葉子個數為 4
model = DecisionTreeClassifier(
        criterion='gini',
        max_depth=4,
        max_leaf_nodes=2 ** 4
)
# 訓練決策樹模型
model.fit(train_x, train_y)
```

```
# 訓練決策樹模型
model.fit(train_x, train_y)

# 確認模型是否訓練成功
pred_y = model.predict(train_x)
# 計算準確度
acc = accuracy_score(train_y, pred_y)
f1 = f1_score(train_y, pred_y)

# 輸出準確度
print('accuracy: {}'.format(acc))
print('F-score: {}'.format(f1))
```

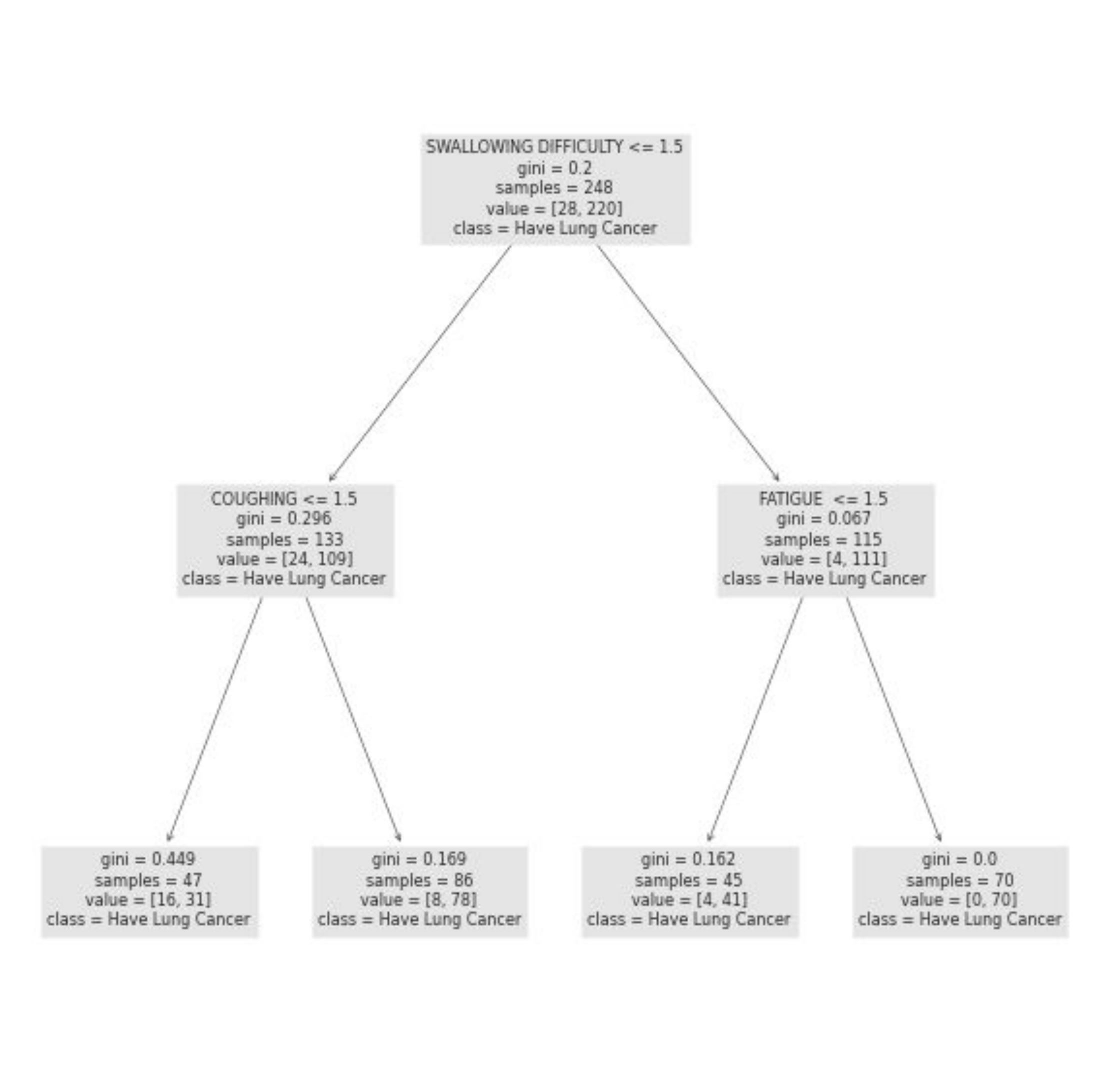
```
accuracy: 0.9029126213592233
F-score: 0.9454545454545454
```

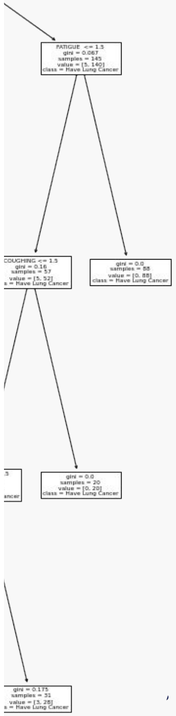
Decision Tree

```
#  交叉驗證

SEED=1024
np.random.seed(SEED)                                    #固定隨機變數，確保每次結果一樣

from  sklearn.model_selection  import  KFold               #  匯入  K  次交叉驗證工具

kf  =  KFold(n_splits=5,                                             #  設定  K  值
            random_state=SEED,
            shuffle=True)
kf.get_n_splits(train_x)                                        #  給予資料範圍

train_acc_list  =  []                                                  #  儲存每次訓練模型的準確度
valid_acc_list  =  []                                                  #  儲存每次驗證模型的準確度

for  train_index,  valid_index  in  kf.split(train_x):       #  每個迴圈都會產生不同部份的資料
        train_x_split  =  train_x.iloc[train_index]                  #  產生訓練資料
        train_y_split  =  train_y[train_index]                     #  產生訓練資料標籤
        valid_x_split  =  train_x.iloc[valid_index]                  #  產生驗證資料
        valid_y_split  =  train_y[valid_index]                     #  產生驗證資料標籤

        model  =  DecisionTreeClassifier(
        random_state=SEED,
        criterion='gini',                                    #設定模型超參數
        max_depth=10,
        max_leaf_nodes=2  **  5
        )                                                      #  創造決策樹模型
        model.fit(train_x_split,  train_y_split)               #  訓練決策樹模型

        train_pred_y  =  model.predict(train_x_split)                      #  確認模型是否訓練成功
        train_acc  =  accuracy_score(train_y_split,  train_pred_y)      #  計算訓練資料準確度
        valid_pred_y  =  model.predict(valid_x_split)                      #  驗證模型是否訓練成功
        valid_acc  =  accuracy_score(valid_y_split,  valid_pred_y)      #  計算驗證資料準確度

        train_acc_list.append(train_acc)
        valid_acc_list.append(valid_acc)
```

```
average train accuracy: 0.9466044142614601
average valid accuracy: 0.8933897408778424

max train accuracy: 0.9554655870445344
max valid accuracy: 0.9508196721311475

min train accuracy: 0.9392712550607287
min valid accuracy: 0.8709677419354839
```

Deci


SWALLOWING DIFFICULTY <= 1.5
gini = 0.2
samples = 248
value = [28, 220]
class = Have Lung Cancer
COUGHING <= 1.5
gini = 0.296
samples = 133
value = [24, 109]
class = Have Lung Cancer
FATIGUE <= 1.5
gini = 0.067
samples = 115
value = [4, 111]
class = Have Lung Cancer
gini = 0.449
samples = 47
value = [16, 31]
class = Have Lung Cancer
gini = 0.169
samples = 86
value = [8, 78]
class = Have Lung Cancer
gini = 0.162
samples = 45
value = [4, 41]
class = Have Lung Cancer
gini = 0.0
samples = 70
value = [0, 70]
class = Have Lung Cancer



Random Forest

```
#匯入隨機森林模型
from sklearn.ensemble i mport RandomForestClassifier

#匯入準確度計算工具
from sklearn.metrics import accuracy_score
#創造隨機森林模型
model = RandomForestClassifier(random_state=1024)
#訓練隨機森林模型
model.fit(train_x, train_y)
#確認模型是否訓練成功
pred_y = model.predict(train_x)
#計算準確度、f1 score
acc = accuracy_score(train_y, pred_y)
f1 = f1_score(train_y, pred_y)

# 輸出準確度
print('accuracy: {}'.format(acc))
print('F-score: {}'.format(f1))
```

```
accuracy: 0.9449838187702265
F-score: 0.9689213893967092
```

Random Forest

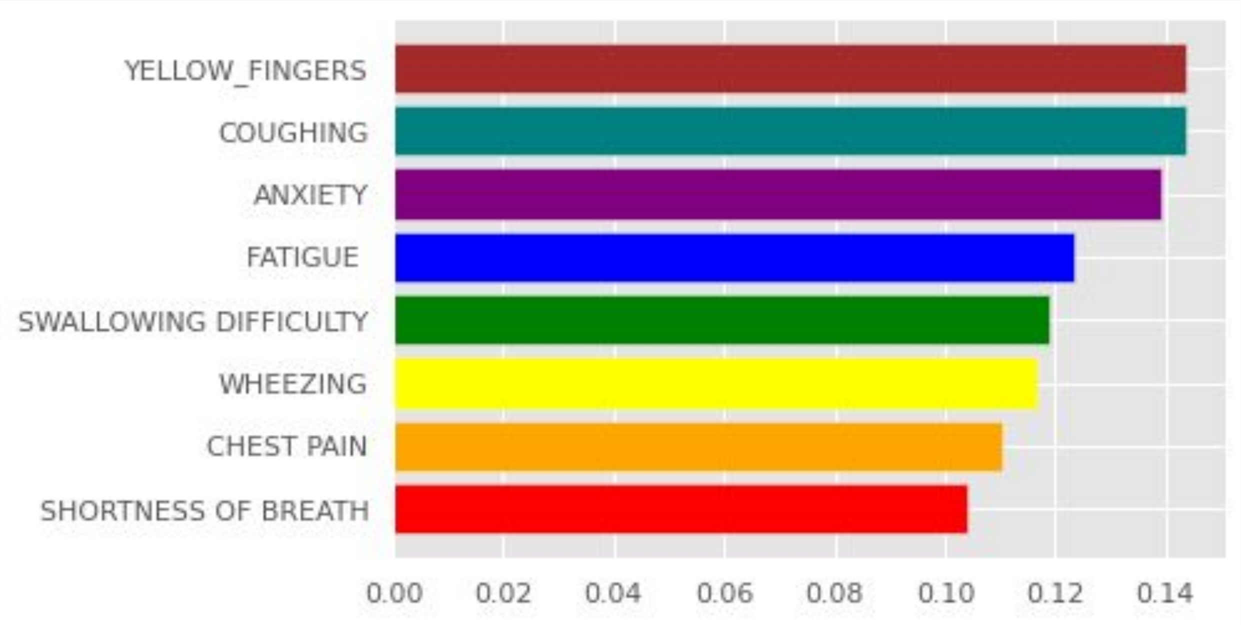
```
#交叉驗證

#匯入K次交叉驗證工具

from sklearn.model_selection import KFold

#設定K值

kf = KFold(n_splits=5, random_state=1012, shuffle=True)

#給予資料範圍

kf.get_n_splits(train_x)

#每個迴圈都會產生不同部份的資料

train_acc_list = []                                    # 儲存每次訓練模型的準確度
valid_acc_list = []                                    # 儲存每次驗證模型的準確度

for train_index, valid_index in kf.split(train_x):     # 每個迴圈都會產生不同部份的資料
    train_x_split = train_x.iloc[train_index]          # 產生訓練資料
    train_y_split = train_y[train_index]               # 產生訓練資料標籤
    valid_x_split = train_x.iloc[valid_index]          # 產生驗證資料
    valid_y_split = train_y[valid_index]               # 產生驗證資料標籤

#使用隨機森林模型

model = RandomForestClassifier(random_state=1024)

model.fit(train_x_split, train_y_split)

train_pred_y = model.predict(train_x_split)                    # 確認模型是否訓練成功
train_acc = accuracy_score(train_y_split, train_pred_y)        # 計算訓練資料準確度
valid_pred_y = model.predict(valid_x_split)                    # 驗證模型是否訓練成功
valid_acc = accuracy_score(valid_y_split, valid_pred_y)        # 計算驗證資料準確度

train_acc_list.append(train_acc)
valid_acc_list.append(valid_acc)
```

```
average train accuracy: 0.9475806451612904
average valid accuracy: 0.9344262295081968

max train accuracy: 0.9475806451612904
max valid accuracy: 0.9344262295081968

min train accuracy: 0.9475806451612904
min valid accuracy: 0.9344262295081968
```

Random Forest

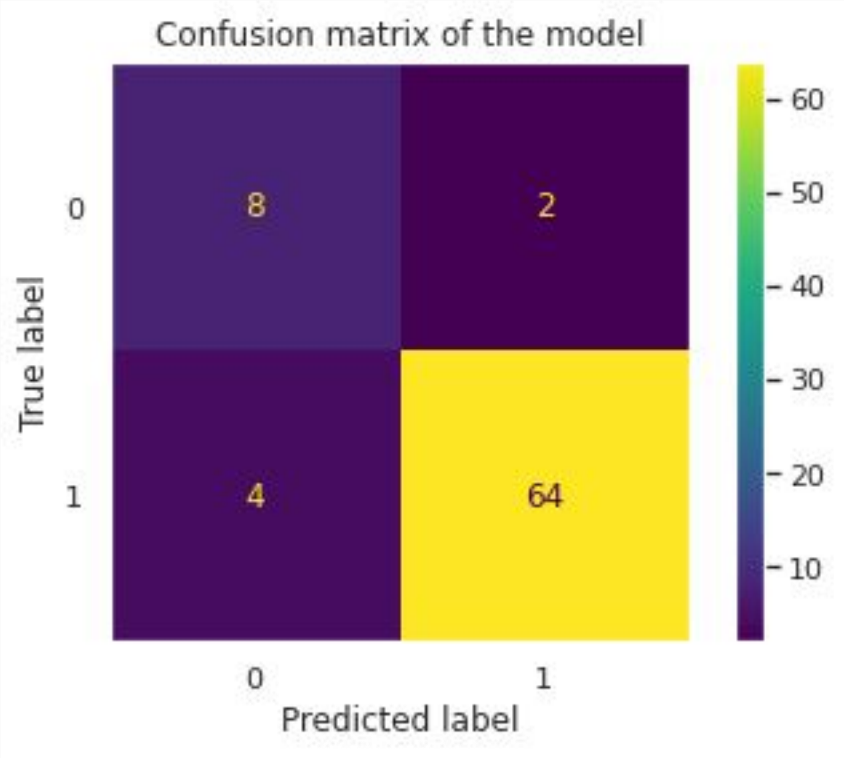
```python
# 匯入計算feature重要程度的工具
from sklearn.inspection import permutation_importance

# 計算重要程度
result = permutation_importance(model, train_x, train_y, random_state=1012)
# 排序
perm_sorted_idx = result.importances_mean.argsort()
tree_importance_sorted_idx = np.argsort(model.feature_importances_)
tree_indices = np.arange(0, len(model.feature_importances_)) + 0.5

# 繪圖
plt.barh(tree_indices,
              model.feature_importances_[tree_importance_sorted_idx],
              tick_label = train_x.columns[tree_importance_sorted_idx],
              color = ['red','orange','yellow','green','blue','purple','teal','brown']
              )
plt.style.use('ggplot')
plt.show()
```

Random Forest



Random Forest

- **準確度熱點圖**



SVM

```
#匯入支援向量機模型

from sklearn.svm import SVC

#匯入準確度計算工具

from sklearn.metrics import accuracy_score
#創造支援向量機模型
model = SVC(random_state=1012)
#訓練支援向量機模型
model.fit(train_x, train_y)
#確認模型是否訓練成功
pred_y = model.predict(train_x)
#計算準確度、f1 score
acc = accuracy_score(train_y, pred_y)
f1 = f1_score(train_y, pred_y)

# 輸出準確度
print('accuracy: {}'.format(acc))
print('F-score: {}'.format(f1))
```

```
accuracy: 0.9158576051779935
F-score: 0.9525547445255474
```

SVM

```
#匯入K次交叉驗證工具

from sklearn.model_selection import KFold

#設定K值

kf = KFold(n_splits=5, random_state=1012, shuffle=True)

#給予資料範圍

kf.get_n_splits(train_x)

#每個迴圈都會產生不同部份的資料

train_acc_list = []                                           # 儲存每次訓練模型的準確度
valid_acc_list = []                                           # 儲存每次驗證模型的準確度

for train_index, valid_index in kf.split(train_x):        # 每個迴圈都會產生不同部份的資料
        train_x_split = train_x.iloc[train_index]              # 產生訓練資料
        train_y_split = train_y[train_index]                # 產生訓練資料標籤
        valid_x_split = train_x.iloc[valid_index]              # 產生驗證資料
        valid_y_split = train_y[valid_index]                # 產生驗證資料標籤

#使用支援向量機模型

model = SVC(random_state=1012)

model.fit(train_x_split, train_y_split)

train_pred_y = model.predict(train_x_split)                        # 確認模型是否訓練成功
train_acc = accuracy_score(train_y_split, train_pred_y)          # 計算訓練資料準確度
valid_pred_y = model.predict(valid_x_split)                        # 驗證模型是否訓練成功
valid_acc = accuracy_score(valid_y_split, valid_pred_y)          # 計算驗證資料準確度

train_acc_list.append(train_acc)
valid_acc_list.append(valid_acc)
```

```
average train accuracy: 0.907258064516129
average valid accuracy: 0.8852459016393442

max train accuracy: 0.907258064516129
max valid accuracy: 0.8852459016393442

min train accuracy: 0.907258064516129
min valid accuracy: 0.8852459016393442
```

KNN

```python
#匯入近鄰演算法模型
from sklearn.neighbors import KNeighborsClassifier

#匯入準確度計算工具
from sklearn.metrics import accuracy_score
#創造近鄰演算法模型
model = KNeighborsClassifier(n_neighbors=3)
#訓練近鄰演算法模型
model.fit(train_x, train_y)
#確認模型是否訓練成功
pred_y = model.predict(train_x)
#計算準確度、f1 score
acc = accuracy_score(train_y, pred_y)
f1 = f1_score(train_y, pred_y)

# 輸出準確度
print('accuracy: {}'.format(acc))
print('F-score: {}'.format(f1))
```

```
accuracy: 0.9288025889967637
F-score: 0.9597069597069596
```

KNN

```
#交叉驗證

#匯入K次交叉驗證工具

from sklearn.model_selection import KFold

#設定K值

kf = KFold(n_splits=5, random_state=1012, shuffle=True)

#給予資料範圍

kf.get_n_splits(train_x)

#每個迴圈都會產生不同部份的資料

train_acc_list = []                                          # 儲存每次訓練模型的準確度
valid_acc_list = []                                          # 儲存每次驗證模型的準確度

for train_index, valid_index in kf.split(train_x):       # 每個迴圈都會產生不同部份的資料
        train_x_split = train_x.iloc[train_index]             # 產生訓練資料
        train_y_split = train_y[train_index]                # 產生訓練資料標籤
        valid_x_split = train_x.iloc[valid_index]             # 產生驗證資料
        valid_y_split = train_y[valid_index]                # 產生驗證資料標籤

#使用近鄰演算法模型

model = KNeighborsClassifier(n_neighbors=3)

model.fit(train_x_split, train_y_split)

train_pred_y = model.predict(train_x_split)                       # 確認模型是否訓練成功
train_acc = accuracy_score(train_y_split, train_pred_y)        # 計算訓練資料準確度
valid_pred_y = model.predict(valid_x_split)                       # 驗證模型是否訓練成功
valid_acc = accuracy_score(valid_y_split, valid_pred_y)        # 計算驗證資料準確度

train_acc_list.append(train_acc)
valid_acc_list.append(valid_acc)
```

```
average train accuracy: 0.9395161290322581
average valid accuracy: 0.9344262295081968

max train accuracy: 0.9395161290322581
max valid accuracy: 0.9344262295081968

min train accuracy: 0.9395161290322581
min valid accuracy: 0.9344262295081968
```

Logistic Regression

```python
#匯入邏輯迴歸模型
from sklearn.linear_model import LogisticRegression

#匯入準確度計算工具
from sklearn.metrics import accuracy_score
#創造邏輯迴歸模型
model = LogisticRegression(random_state=1024)
#訓練邏輯迴歸模型
model.fit(train_x, train_y)
#確認模型是否訓練成功
pred_y = model.predict(train_x)
#計算準確度、f1 score
acc = accuracy_score(train_y, pred_y)
f1 = f1_score(train_y, pred_y)

# 輸出準確度
print('accuracy: {}'.format(acc))
print('F-score: {}'.format(f1))
```

```
accuracy: 0.9029126213592233
F-score: 0.9458483754512635
```

Logistic Regression

```
#交叉驗證

#匯入K次交叉驗證工具

from  sklearn.model_selection  import  KFold

#設定K值

kf  =  KFold(n_splits=5,  random_state=1012,  shuffle=True)

#給予資料範圍

kf.get_n_splits(train_x)

#每個迴圈都會產生不同部份的資料

train_acc_list  =  []                                                    #  儲存每次訓練模型的準確度
valid_acc_list  =  []                                                    #  儲存每次驗證模型的準確度

for  train_index,  valid_index  in  kf.split(train_x):       #  每個迴圈都會產生不同部份的資料
        train_x_split  =  train_x.iloc[train_index]                  #  產生訓練資料
        train_y_split  =  train_y[train_index]                   #  產生訓練資料標籤
        valid_x_split  =  train_x.iloc[valid_index]                  #  產生驗證資料
        valid_y_split  =  train_y[valid_index]                   #  產生驗證資料標籤

#使用邏輯迴歸模型

model  =  LogisticRegression(random_state=1024)

model.fit(train_x_split,  train_y_split)

train_pred_y  =  model.predict(train_x_split)                               #  確認模型是否訓練成功
train_acc  =  accuracy_score(train_y_split,  train_pred_y)              #  計算訓練資料準確度
valid_pred_y  =  model.predict(valid_x_split)                               #  驗證模型是否訓練成功
valid_acc  =  accuracy_score(valid_y_split,  valid_pred_y)              #  計算驗證資料準確度

train_acc_list.append(train_acc)
valid_acc_list.append(valid_acc)
```

```
average train accuracy: 0.9032258064516129
average valid accuracy: 0.8852459016393442

max train accuracy: 0.9032258064516129
max valid accuracy: 0.8852459016393442

min train accuracy: 0.9032258064516129
min valid accuracy: 0.8852459016393442
```

Adaptive Boosting

```
#  匯入AdaBoost模型
from  sklearn.ensemble  import  AdaBoostClassifier
#匯入準確度計算工具
from  sklearn.metrics  import  accuracy_score
#創造隨機AdaBoost
model  =  AdaBoostClassifier(random_state=1024)
#訓練隨機AdaBoost
model.fit(train_x,  train_y)
#確認模型是否訓練成功
pred_y  =  model.predict(train_x)
#計算準確度、f1  score
acc  =  accuracy_score(train_y,  pred_y)
f1  =  f1_score(train_y,  pred_y)

#  輸出準確度
print('accuracy:  {}'.format(acc))
print('F-score:  {}'.format(f1))
```

```
accuracy:  0.8932038834951457
F-score:  0.9396709323583181
```

Adaptive Boosting

```
#交叉驗證

#匯入K次交叉驗證工具

from sklearn.model_selection import KFold

#設定K值

kf = KFold(n_splits=5, random_state=1012, shuffle=True)

#給予資料範圍

kf.get_n_splits(train_x)

#每個迴圈都會產生不同部份的資料

train_acc_list = []                                    # 儲存每次訓練模型的準確度
valid_acc_list = []                                    # 儲存每次驗證模型的準確度

for train_index, valid_index in kf.split(train_x):     # 每個迴圈都會產生不同部份的資料
    train_x_split = train_x.iloc[train_index]          # 產生訓練資料
    train_y_split = train_y[train_index]               # 產生訓練資料標籤
    valid_x_split = train_x.iloc[valid_index]          # 產生驗證資料
    valid_y_split = train_y[valid_index]               # 產生驗證資料標籤

#使用AdaBoost模型

model = AdaBoostClassifier(random_state=1024)

model.fit(train_x_split, train_y_split)

train_pred_y = model.predict(train_x_split)                    # 確認模型是否訓練成功
train_acc = accuracy_score(train_y_split, train_pred_y)        # 計算訓練資料準確度
valid_pred_y = model.predict(valid_x_split)                    # 驗證模型是否訓練成功
valid_acc = accuracy_score(valid_y_split, valid_pred_y)        # 計算驗證資料準確度

train_acc_list.append(train_acc)
valid_acc_list.append(valid_acc)
```

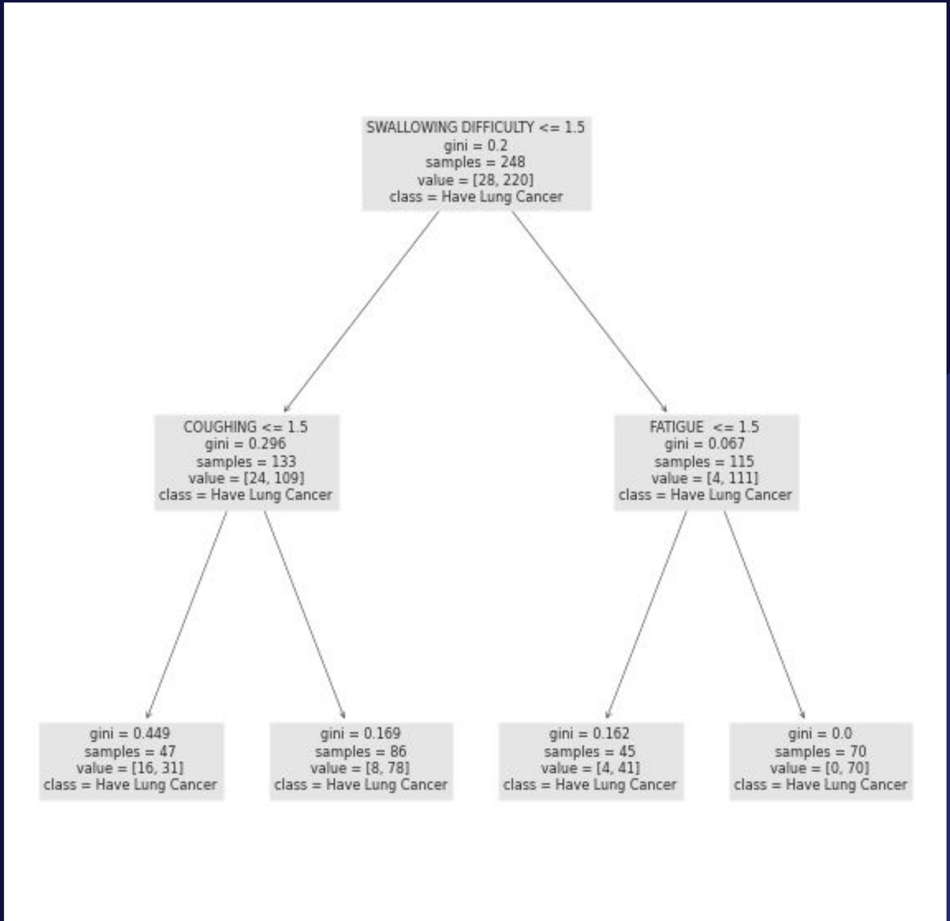
```
average train accuracy: 0.8870967741935484
average valid accuracy: 0.9016393442622951

max train accuracy: 0.8870967741935484
max valid accuracy: 0.9016393442622951

min train accuracy: 0.8870967741935484
min valid accuracy: 0.9016393442622951
```

綜合比較

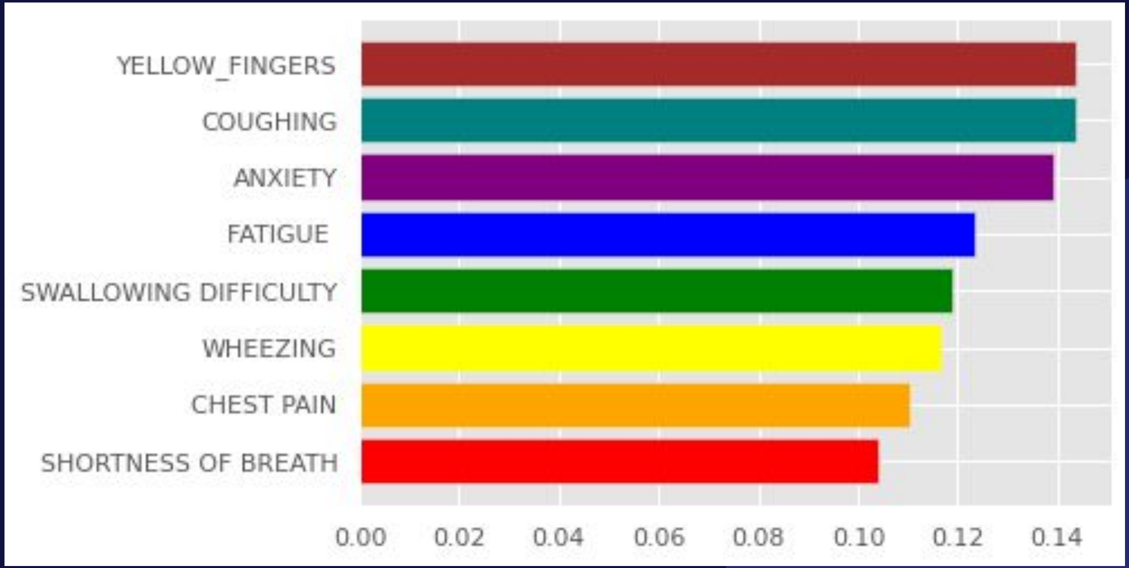
	Decision Tree	Random Forest	SVM	KNN	Logistic Regression	Adaptive Boosting
Average Train Accuracy	0.9466	0.9476	0.9073	0.9395	0.9032	0.8871
Average Valid Accuracy	0.8934	0.9344	0.8852	0.9344	0.8852	0.9016

結論

此資料集有肺癌者比例遠大於無肺癌者，導致決策樹的預測結果皆為有肺癌，若繼續向下延伸可預測：沒有吞嚥困難、沒有咳嗽、沒有氣喘等症狀的人沒有肺癌



經由綜合比較得知 **隨機森林的預測能力最佳**，並由其計算出的特徵重要程度可得知，**黃手指、咳嗽、焦慮** 等症狀為預測肺癌的最重要依據，此外各個症狀的預測能力差異並不顯著



參考資料

- Lung cancer prediction with symptoms | Kaggle
- GitHub - IKMLab/course_material: 上課教材的大集合！！！