預測天氣是否導致航班延誤，

若延誤，則延誤多久

期末報告書

學生：李育綺

指導老師：葉向原

目錄

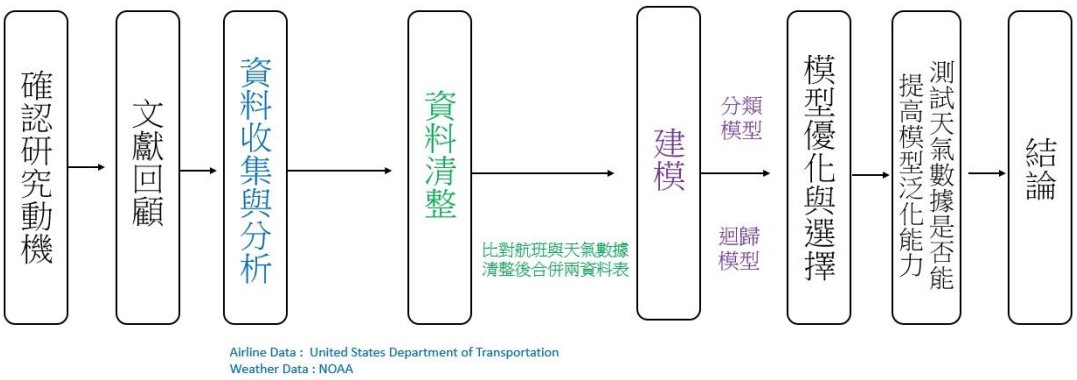
一、 摘要

隨著科技的發展，人工智慧已經應用在我們生活中許多領域。而藉由飛機往來各地也不再是一件稀有的事情。想必不少人有過這種經驗，出發前剛好遇上天候不佳，例如颱風、暴風雨等...，擔心飛機不能準時起飛，甚至開始猶豫是否該調整行程。因此若是能提早掌握資訊，就能避免等待時間的浪費。

本研究主要預測天氣如何影響航班延誤，為求分析結果具公信力，使用 NOAA 提供的公開天氣數據以及 United States Department of Transportation 的航班數據集訓練兩個模型，分別為處理第一階段航班是否因天氣延誤的二元分類模型，以及第二階段延誤多長時間的迴歸模型。

藉由本次研究，我們的目標在於能夠將此模型更廣泛應用於不同航空公司、不同國家，使更多人受益。

二、 研究流程



上圖展示了研究流程設計，首先確認研究動機，以及回顧過往文獻，再將收集回來的航班和天氣數據進行資料預處理，處理完後分別建立分類模型及迴歸模型來預測天氣對航班造成的影響，最後測試天氣數據是否真的能提高模型泛化能力。

三、 資料分析與處理

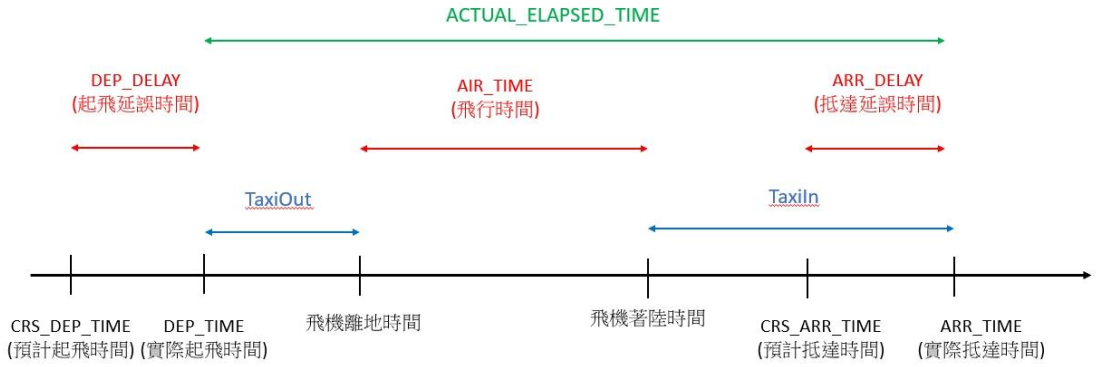
1. 資料分析

航班數據集部份，未經資料清整的原始航班數據共有 510,806 筆資料，28 欄位（YEAR、MONTH、DAY_OF_MONTH、DAY_OF_EWEEK、UNIQUE_CARRIER、FL_NUM、ORIGIN_AIRPORT_ID、ORIGIN、ORIGIN_STATE_ABR、DEST_AIRPORT_ID、DEST、DEST_STATE_ABR、CRS_DEP_TIME、DEP_TIME、DEP_DELAY、CRS_ARR_TIME、ARR_TIME、ARR_DELAY、CRS_ELAPSED_TIME、ACTUAL_ELAPSED_TIME、AIR_TIME、DISTANCE、CARRIER_DELAY、WEATHER_DELAY、NAS_DELAY、SECURITY_DELAY、LATE_AIRCRAFT_DELAY）大致分為航班靜態相關資訊(日期、出發地目的地)、航班動態相關資訊(實際/預計起飛到達時間、延誤分鐘數)、五種不同延誤因素的延誤時間。其中 ORIGN、DEST 欄位分別代表出發地機場以及目的地機場的 IATA(國際航空運輸協會機場代碼)。

1	Code	Description
5092	STA	Stauning, Denmark: Stauning Airport
5093	STB	Santa Barbara, Venezuela: Santa Barbara Airport
5094	STC	St. Cloud, MN: St. Cloud Regional
5095	STD	Santo Domingo, Venezuela: Mayor Guerrero Vivas
5096	STE	Stevens Point, WI: Stevens Point Municipal

（圖一:IATA 與相對應的機場完整名稱）

此外，依據國際航班相關作業時間標準，以及參考 Airline2008Nov Dataset Variable definition 所統整出的名詞解釋，可以推算出 ACTUAL_ELAPSED_TIME - CRS_ELAPSED_TIME + DEP_DELAY = ARR_DELAY = CARRIER_DELAY + WEATHER_DELAY + NAS_DELAY + SECURITY_DELAY + LATE_AIRCRAFT_DELAY



（圖二:參考 Airline2008Nov Dataset Variable definition 及自行統整的航班作業時間軸）

1	DEP_DELAY	ARR_DELAY	CRS_ELAPSED_TIME	ACTUAL_ELAPSED_TIME
2	-4	-18	170	156
3	-5	-3	95	97
4	-1	-12	95	84
5	-3	-13	95	85
6	-6	-16	95	85

（圖三:ACTUAL_ELAPSED_TIME - CRS_ELAPSED_TIME + DEP_DELAY = ARR_DELAY）

針對 ARR_DELAY 欄位，可以為負數、正數，或 0，當 ARR_DELAY<0，代表提前抵達目的地機場;ARR_DELAY>0，代表延誤抵達，ARR_DELAY=0，及表示準點。且當 ARR_DELAY>=15 分鐘時，才會記錄延誤原因為何(意即五個延誤原因欄位必不為空)

天氣數據集部份，未經資料清整的原始天氣數據共有 137,139 筆資料，90 個欄位，大致分為站點資訊(站點所在地機場名稱、經緯度)、紀錄時間(YYYY-MM-DD hh:mm)、每小時、每日、每月天氣數據，其中 STATION_NAME 可透過(圖一)找到相應的 IATA 代碼，進一步與航班資料集中的 ORIGN、DEST 欄位對應。天氣數據欄位中包含能見度、乾溼球溫度、露點溫度、風速、風向、相對濕度等欄位，且資料型態皆已數值型方式呈現。

2. 航班資料處理

Step1: 篩選出 ORIGN、DEST 欄位中有天氣數據的機場航班資料(意即僅針對 ORIGN、DEST 欄位為 STL 或 BNA 或 DAL 或 HOU 的航班數據進行研究)。

Step2: 刪除用不到的欄位，例如 FL_NUM(飛機編號)，使得整體資料集維度降低。

Step3: 針對缺失值位於 DEP_TIME、DEP_DELAY、ARR_TIME、ARR_DELAY、ACTUAL_ELAPSED_TIME、AIR_TIME 欄位的資料整筆刪除，原因是當這些主要欄位存在空值時，將難以獲得有效的數據進後續的預測。

Step4: 由於本次專題為預測天氣是否導致航班延誤，因此將延誤原因與"天氣"無關的航班資料刪除，刪除方式如(圖四)所示。

```
# 4. 將延誤原因與"天氣"無關的航班資料刪除(先抓)
not_weather_delay = airline[((airline['CARRIER_DELAY'] >= 0) | (airline['LATE_AIRCRAFT_DELAY'] >= 0)
              | (airline['NAS_DELAY'] >= 0) |(airline['SECURITY_DELAY'] >= 0))
              & (airline['WEATHER_DELAY'] == 0) ]

# 4. 將延誤原因與"天氣"無關的航班資料刪除(再刪)
flight = airline.drop(not_weather_delay.index)
print(flight.shape)
```

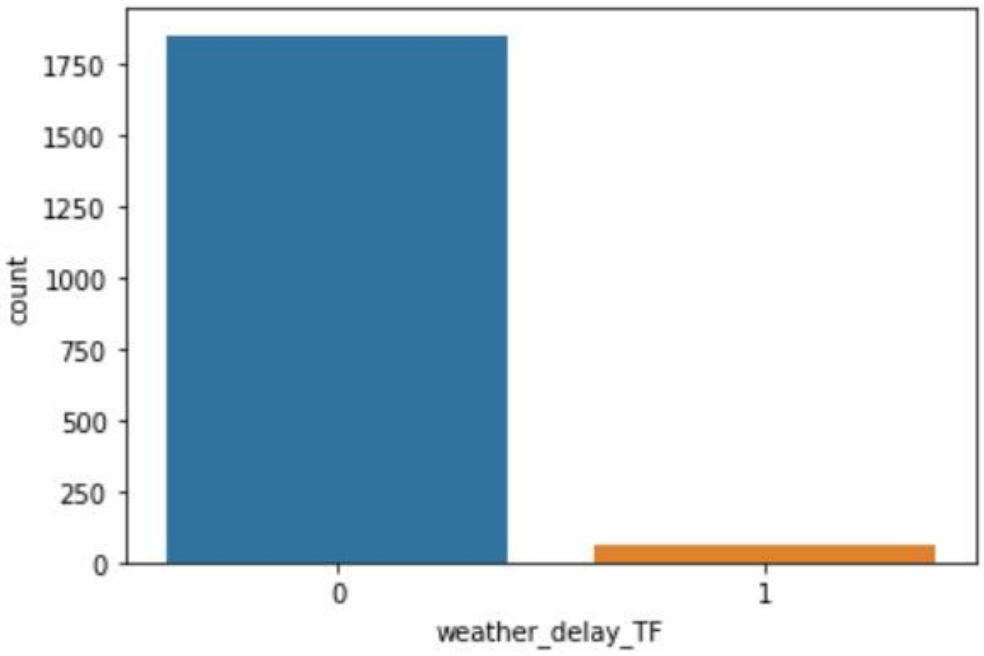
（圖四: 只要有一個延誤原因欄位不為空，就代表發生延誤，因此當發生此狀況，並且 WEATHER_DELAY 的值為 0 時，即表示此航班延誤原因不包含天氣因素）

Step5: 對 WEATHER_DELAY 欄位進行補值，分析原始數據發現，WEATHER_DELAY 會依據 ARR_DELAY 分為 3 種狀況，1. 當 ARR_DELAY<=0，代表沒延誤，此時 WEATHER_DELAY 的空值應填為 0；2. 當 0<ARR_DELAY<=15，不會標註延誤原因為何以及對應的延誤分鐘數;3. 當 ARR_DELAY>15，原數據中的 WEATHER_DELAY 皆有紀錄相對應的延誤分鐘數，因此不存在空值。
分析完空值分布後，使用 knn 對缺失值進行填充，那是因為 knn 能根據每個缺失值，藉由歐幾里得距離找出最鄰近的 k 個資料點，再計算這 k 個資料加權平均值(因為 WEATHER_DELAY 為連續型資料)來填補，由於 0<ARR_DELAY<=15 的資料筆數並非少數，若統一使用平均值或眾數等簡單的統計方法來補值，會產生偏誤，因此最終選擇用 knn 處理缺失數據。

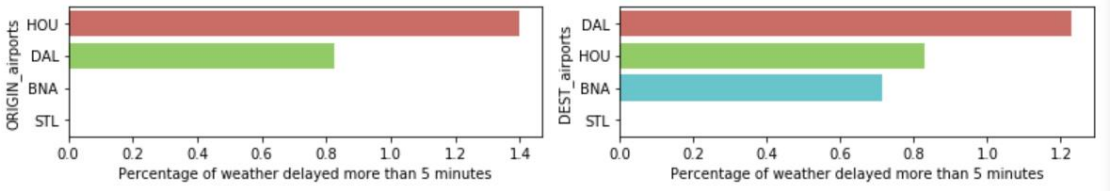
Step6: 時間欄位處理，將預計出發時間(CRS_DEP_TIME)視為此航班出發的"時";實際抵達時間(ARR_TIME)視為此航班抵達目的地的"時"。最後，根據 WEATHER_DELAY 之值是否為 0 當作判斷標準，新增一個分類模型會使用到的 weater_delay_TF 欄位來記錄此筆航班使否因天氣發生延誤(0 表沒延誤;1 表延誤)。

航班資料初步視覺化

1. 透過下圖可看出 target 的類別分布極不平均，意即航班數據集屬於 imbalanced data。



2. 分別查看 ORIGIN、DEST 的 4 個機場延誤狀況，下圖呈現各機場因天氣狀況造成航班延誤超過 5 分鐘佔其航班總數的比例。



3. 天氣資料處理

天氣數據分為兩個資料集，而每份資料集中皆包含兩個天氣測站。由於兩份資料集來源一樣，且比對過後確認欄位皆相同，因此兩份資料集將使用同一種資料清整方式。

為了與航班數據合併:

Step1: 首先將原始數據中，資料型態為(YYYY-MM-DD hh:mm) 的 DATE 欄位透過 split(" ")函式，拆分成紀錄日期的 YMD 欄位(YYYY-MM-DD)以及紀錄時間的 TIME 欄位(hh:mm)，並抓取出 TIME 欄位的小時新增為 HOUR 欄位來與航班數據的出發/抵達小時比對。

Step2: 針對用不到的欄位進行刪除，以降低資料集的維度。

Step3: 刪完後對照(圖一)將 STATION_NAME 欄位的完整機場名稱替換成能夠與航班數據中的 ORIGN、DEST 欄位對應的 IATA 代碼。

Step4: 查看欄位資料型態以及原始天氣數據，發現某些筆應當屬於數值欄位(HOURLYVISIBILITY、HOURLYDRYBULBTEMPC、HOURLYDewPointTempC、HOURLYWindSpeed、HOURLYPrecip)的數據夾雜英文字符，因此統一對數字、字符夾雜的數據，進行"刪除英文字符"的動作。

Step5: 時間欄位處理，抓取"時"作為航班起飛/降落的"小時"。

Step6: 填補空值。

Step7: 特徵組合，由於乾球溫度為一般溫度感測器所量到的溫度，而濕球是在溫度感測器綁上濕布，並泡在一小杯水中，讓水分包裹整個感測器，只要當空氣中的相對濕度小於 100%，濕球的水分會被蒸發至空氣中，而這蒸發量就是由乾溼球溫度差得知，因此我將乾球溫度、濕球溫度之差組成一個新欄位，作為蒸發量數據。

4. 合併航班與天氣資料

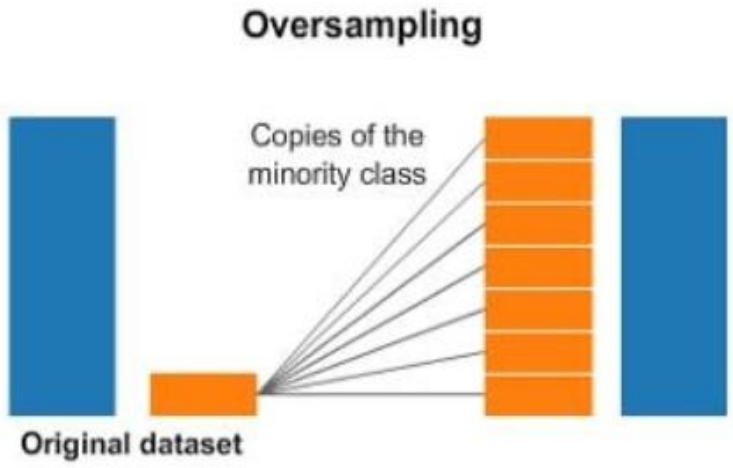
將經過資料預處理的航班及天氣數據依照出發地目的地機場、日期、時間合併，並複製成兩份，其中一份用來訓練是否會因天氣延誤的二元分類問題模型，另一份用來訓練若延誤，則延誤幾分鐘的迴歸問題模型。在第一部分，weather_delay_TF 欄位為我們的 target，第二部分的 target 則是 WEATHER_DELAY 欄位。

四、 建模

1. 預測航班是否因天氣延誤_二元分類模型

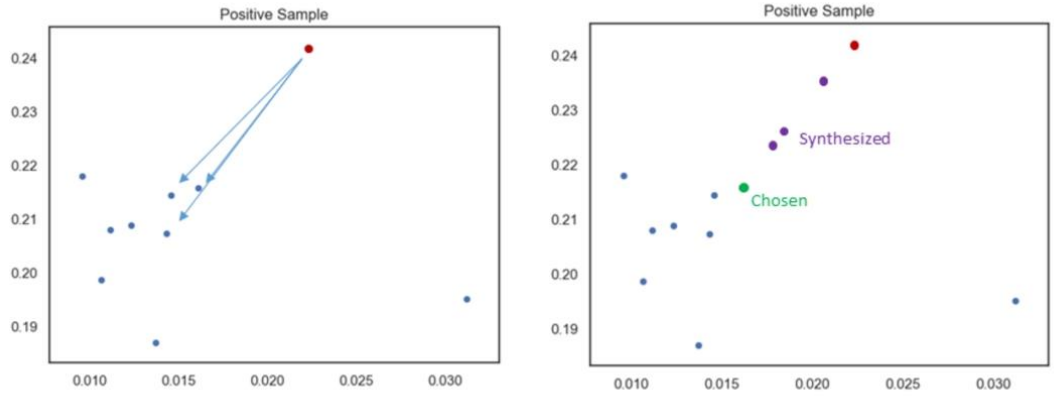
由於航班數據集屬於 imbalanced data，本階段分別嘗試採用過採樣、欠採樣、及綜合採樣來解決類別不平衡的問題，最終選擇有較高 roc_auc_score 的模型當作最終使用的分類模型。

I. 過採樣(Over-sampling)
主要概念是透過人工合成來增加數據集中代表性不足的小樣本數量，使得每類樣本比例為 1:1。



（示意圖來源:kaggle）

原始的 SMOTE()方法，其人工合成少量樣本方式為對於一個少量樣本，針對該樣本使用 knn 找出距離其最近的 k 個少數類樣本，並從中隨機挑選一個，再根據此公式創造出 N 個新樣本(k、N 自訂)，然而 SMOTE()在挑選少量樣本時是"隨機"選取，當被選取的少量樣本附近都是多數類樣本時，合成出的樣本將會使得噪聲影響更大，導致後續模型難以正確分類。但當被選取的少量樣本附近都是少數類樣本時，新合成的樣本也無法提供有用資訊，因此利用容易被錯分的邊界樣本來產生新樣本，更能有效提升模型泛化能力。

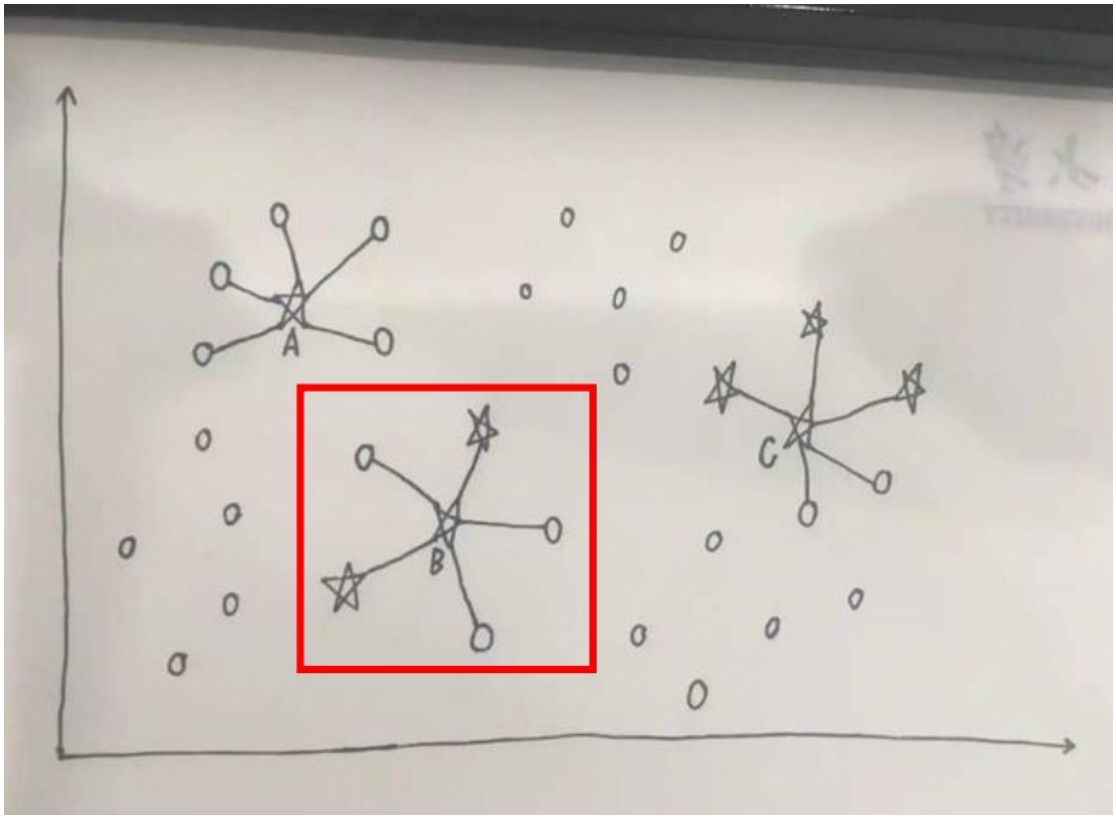


（SMOTE 示意圖。來源:SMOTE + ENN ： 解決數據不平衡建模的採樣方法）

```
model = RandomForestClassifier(n_estimators=300, max_depth=4)
over = SMOTE()

steps = [('over', over), ('model', model)]
pipeline = Pipeline(steps=steps)
# evaluate pipeline
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
scores = cross_val_score(pipeline,X_trainval, y_trainval, scoring='roc_auc', cv=cv, n_jobs=-1)
print('Mean ROC AUC: %.3f' % mean(scores))


pipeline.fit(X_trainval, y_trainval)
y_test_pred = pipeline.predict(X_test)
print(roc_auc_score(y_test, y_test_pred))
```

(smote)

所以我嘗試改採 BorderlineSMOTE()方法，針對邊界樣本進行操作。



(BorderlineSMOTE 示意圖。來源:數據嗨客 ｜ 第 6 期：不平衡數據處理)

```
model = RandomForestClassifier(n_estimators=300, max_depth=4)
over = BorderlineSMOTE()

steps = [('over', over), ('model', model)]
pipeline = Pipeline(steps=steps)
# evaluate pipeline
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
scores = cross_val_score(pipeline,X_trainval, y_trainval, scoring='roc_auc', cv=cv, n_jobs=-1)
print('Mean ROC AUC: %.3f' % mean(scores))


pipeline.fit(X_trainval, y_trainval)
y_test_pred = pipeline.predict(X_test)
print(roc_auc_score(y_test, y_test_pred))
```

(BorderlineSMOTE)

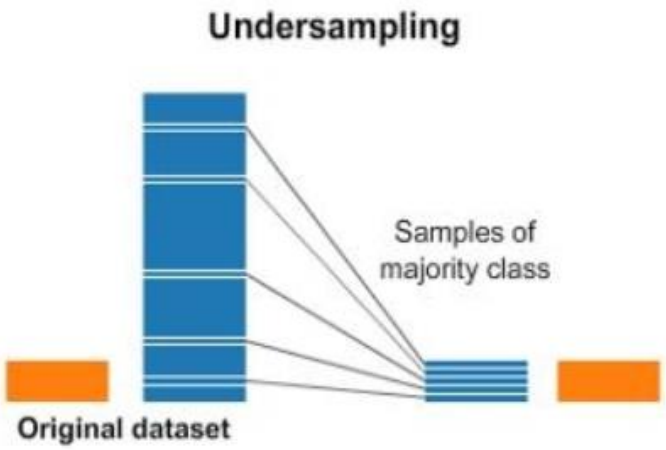
ADASYN()方法，也如同 BorderlineSMOTE()一樣，考慮了多數類樣本，但由於 ADASYN()最終不一定要使採樣結果的多數類、少數類樣本數相等，而是能依據每個少數類樣本的學習難度，合成不同數量的新樣本，意思就是對於那些更難學習的少數類樣本生成更多的新樣本。因此我們也嘗試了此種過採樣方式處理不平衡數據問題。

```
model = RandomForestClassifier(n_estimators=300, max_depth=4)
over = ADASYN()

steps = [('over', over), ('model', model)]
pipeline = Pipeline(steps=steps)
# evaluate pipeline
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
scores = cross_val_score(pipeline,X_trainval, y_trainval, scoring='roc_auc', cv=cv, n_jobs=-1)
print('Mean ROC AUC: %.3f' % mean(scores))


pipeline.fit(X_trainval, y_trainval)
y_test_pred = pipeline.predict(X_test)
print(roc_auc_score(y_test, y_test_pred))
```

(ADASYN)

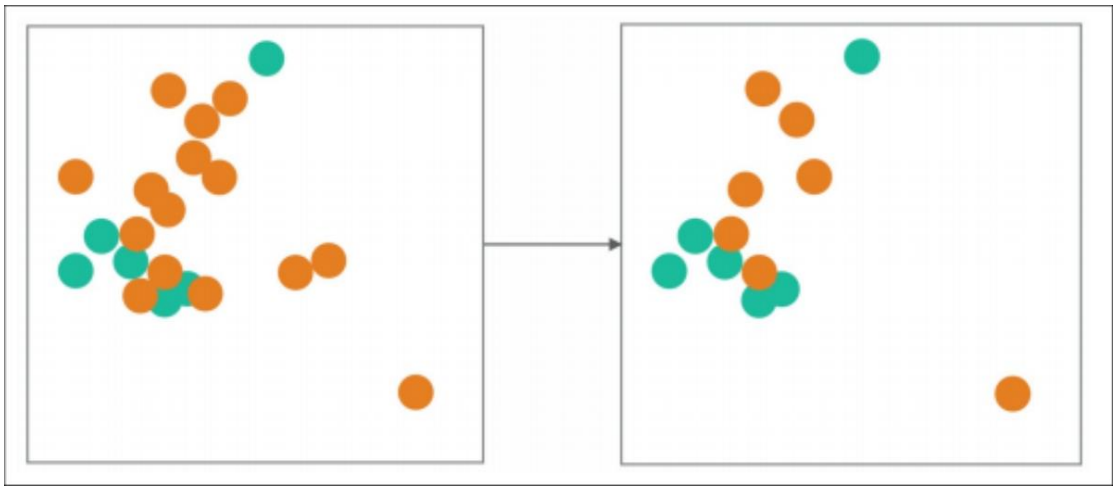
II. 欠採樣(Under-sampling)

主要概念透過減少多數類樣本數量，使得每類樣本比例為 1:1。



(示意圖來源:kaggle)

最簡單的 RandomUnderSampler 方法就是從多數類中隨機抽取樣本從而減少多數類樣本的數量，使資料達到平衡。



(RandomUnderSampler 示意圖。來源: 不平衡資料分類演算法介紹與比較)

```
model = RandomForestClassifier(n_estimators=300, max_depth=4)
under = RandomUnderSampler()

steps = [('under', under), ('model', model)]
pipeline = Pipeline(steps=steps)
# evaluate pipeline
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
scores = cross_val_score(pipeline,X_trainval, y_trainval, scoring='roc_auc', cv=cv, n_jobs=-1)
print('Mean ROC AUC: %.3f' % mean(scores))


pipeline.fit(X_trainval, y_trainval)
y_test_pred = pipeline.predict(X_test)
print(roc_auc_score(y_test, y_test_pred))
```

III. 綜合採樣

由於 SMOTE 在生成新樣本時會發生生成的少數類樣本容易與周圍的多數類樣本重疊，產生噪音，使得模型難以分類，因此藉由綜合採樣的方式先採取過採樣來平衡數據，再進行數據清理處理掉重疊的樣本。既能達到平衡類別分布，亦能防止模型過擬合。

本研究使用兩種常見的綜合採樣例子，進行實驗。

- SMOTEENN (Edited Nearest Neighbours)

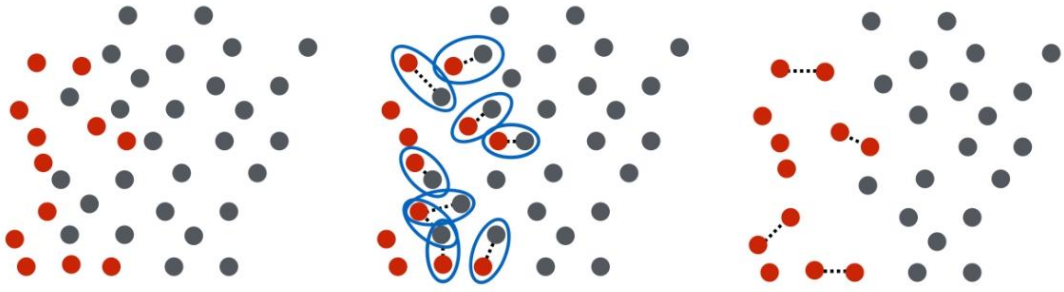
SMOTEENN()是利用 SMOTE()合成新的少數類樣本，再對新的數據集中每個樣本使用 Knn 進行數據清理，當該樣本的 k 個鄰居中，超過半數與該樣本同類，才被保留。

(SMOTEENN 示意圖。來源: 不平衡資料分類演算法介紹與比較)

```
model = RandomForestClassifier(n_estimators=300, max_depth=4)
over = SMOTEENN()

steps = [('over', over), ('model', model)]
pipeline = Pipeline(steps=steps)
# evaluate pipeline
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
scores = cross_val_score(pipeline,X_trainval, y_trainval, scoring='roc_auc', cv=cv, n_jobs=-1)
print('Mean ROC AUC: %.3f' % mean(scores))

print(confusion_matrix(y_test, y_test_pred))


pipeline.fit(X_trainval, y_trainval)
y_test_pred = pipeline.predict(X_test)
print("測試集分數:",roc_auc_score(y_test, y_test_pred))
```

- SMOTETomek()

亦是利用 SMOTE()合成新的少數類樣本，當兩樣本點互為最近鄰但屬不同類別，就刪除多數類樣本點，直到所有少數類最近鄰亦為少數類別即可停止。



(SMOTETomek 示意圖。來源:不平衡資料的二元分類 2:利用抽樣改善模型品質)

```
model = RandomForestClassifier(n_estimators=300, max_depth=4,random_state=1)
over = SMOTETomek()

steps = [('over', over), ('model', model)]
pipeline = Pipeline(steps=steps)
# evaluate pipeline
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
scores = cross_val_score(pipeline,X_trainval, y_trainval, scoring='roc_auc', cv=cv, n_jobs=-1)
print('Mean ROC AUC: %.3f' % mean(scores))

print(confusion_matrix(y_test, y_test_pred))


pipeline.fit(X_trainval, y_trainval)
y_test_pred = pipeline.predict(X_test)
print("測試集分數:",roc_auc_score(y_test, y_test_pred))
```

2. 延誤多長時間_迴歸模型

本階段使用了 XGBoost 搭配 GridSearchCV 對航班延誤時間進行預測。

整體調參流程按照以下順序進行:

1. n_estimators
   - 粗調

```
other_params = {'learning_rate':0.1,'n_estimators': 400, 'max_depth': 3, 'min_child_weight': 1,'seed':1850,
                'subsample': 0.8, 'colsample_bytree': 0.8,'colsample_bylevel':1,'reg_alpha':1,'reg_lambda':1,
                'gamma':0.2,'objective':"reg:linear"}

cv_params = {'n_estimators': [50,100,200,300,400,500,600]}
```

執行結果:

```
參数的最佳取值：: {'n_estimators': 200}
最佳模型得分: 0.2563039454265911
```

由於設定的間隔太大，又測試了一組粒度較小的參數

   - 細調

```
other_params = {'learning_rate':0.1,'n_estimators': 400, 'max_depth': 3, 'min_child_weight': 1,'seed':1850,
                'subsample': 0.8, 'colsample_bytree': 0.8,'colsample_bylevel':1,'reg_alpha':1,'reg_lambda':1,
                'gamma':0.2,'objective':"reg:linear"}

cv_params = {'n_estimators': [150,175,200,250,275]}
```

執行結果:

```
參数的最佳取值：: {'n_estimators': 175}
最佳模型得分: 0.2611058634181719
```

2. min_child_weight 以及 max_depth

```
other_params = {'learning_rate':0.1,'n_estimators': 175, 'max_depth': 3, 'min_child_weight': 1,'seed':1850,
                'subsample': 0.8, 'colsample_bytree': 0.8,'colsample_bylevel':1,'reg_alpha':1,'reg_lambda':1,
                'gamma':0.2,'objective':"reg:linear"}

cv_params = {'min_child_weight': [1,2,3,4,5],"max_depth":[3,4,5,6,7]}
```

執行結果:

```
參数的最佳取值：: {'max_depth': 5, 'min_child_weight': 1}
最佳模型得分: 0.3317431459985186
```

3. gamma

```
other_params = {'learning_rate':0.1,'n_estimators': 175, 'max_depth': 5, 'min_child_weight': 1,'seed':1850,
                'subsample': 0.8, 'colsample_bytree': 0.8,'colsample_bylevel':1,'reg_alpha':1,'reg_lambda':1,
                'gamma':0.2,'objective':"reg:linear"}

cv_params = {"gamma":[0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1]}
```

執行結果:

```
參数的最佳取值：: {'gamma': 0}
最佳模型得分: 0.3685016567270184
```

4. subsample

```
other_params = {'learning_rate':0.1,'n_estimators': 175, 'max_depth': 5, 'min_child_weight': 1,'seed':1850,
                'subsample': 0.8, 'colsample_bytree': 0.8,'colsample_bylevel':1,'reg_alpha':1,'reg_lambda':1,
                'gamma':0,'objective':"reg:linear"}

cv_params = {"subsample":[0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1]}
```

執行結果:

```
參数的最佳取值：: {'subsample': 0.5}
最佳模型得分: 0.3839494048853235
```

5. colsample_bytree

```
other_params = {'learning_rate':0.1,'n_estimators': 175, 'max_depth': 5, 'min_child_weight': 1,'seed':1850,
                'subsample': 0.5, 'colsample_bytree': 0.8,'colsample_bylevel':1,'reg_alpha':1,'reg_lambda':1,
                'gamma':0,'objective':"reg:linear"}

cv_params = {"colsample_bytree":[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]}
```

執行結果:

```
參数的最佳取值：: {'colsample_bytree': 0.7}
最佳模型得分: 0.44678463648796346
```

6. reg_alpha

```
other_params = {'learning_rate':0.1,'n_estimators': 175, 'max_depth': 5, 'min_child_weight': 1,'seed':1850,
                'subsample': 0.5, 'colsample_bytree': 0.7,'colsample_bylevel':1,'reg_alpha':1,'reg_lambda':1,
                'gamma':0,'objective':"reg:linear"}


cv_params = {"reg_alpha":[0,1,2,3,4,5,6,7,8,9,10]}
```

執行結果:

```
參数的最佳取值：: {'reg_alpha': 1}
最佳模型得分: 0.44678463648796346
```

7. reg_lambda

```
other_params = {'learning_rate':0.1,'n_estimators': 175, 'max_depth': 5, 'min_child_weight': 1,'seed':1850,
                'subsample': 0.5, 'colsample_bytree': 0.7,'colsample_bylevel':1,'reg_alpha':1,'reg_lambda':1,
                'gamma':0,'objective':"reg:linear"}


cv_params = {"reg_lambda":[0.05, 0.1, 1, 2, 3]}
```

執行結果:

```
參数的最佳取值：: {'reg_lambda': 2}
最佳模型得分: 0.45419524336056866
```

8. learning_rate

```
other_params = {'learning_rate':0.1,'n_estimators': 175, 'max_depth': 5, 'min_child_weight': 1,'seed':1850,
                'subsample': 0.5, 'colsample_bytree': 0.7,'colsample_bylevel':1,'reg_alpha':1,'reg_lambda':2,
                'gamma':0,'objective':"reg:linear"}

cv_params = {"learning_rate":[0.01, 0.05, 0.07, 0.1]}
```

執行結果:

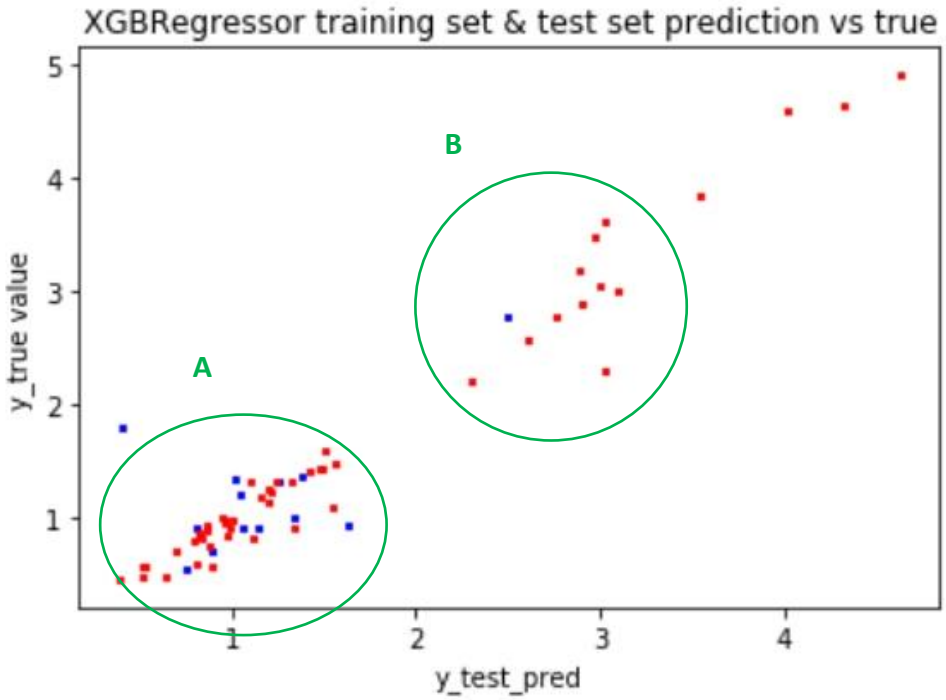
```
參数的最佳取值：: {'learning_rate': 0.1}
最佳模型得分: 0.45419524336056866
```

這邊可以看到，隨著參數不斷調優，模型分數也不斷提高。接著使用最佳參數組合來建模，判斷模型效能。

```
訓練集mes: 0.22998937422348073
訓練集r2: 0.9628199241418064
------------------------------
測試集mes: 0.47306302181892795
測試集r2: 0.25792230233665814
```

經過一連串調參後，可以看出模型對於訓練集有很好的預測能力，但測試集卻很差。

透過畫出訓練集、測試集的實際值與預測值散佈圖，發現原始資料幾乎集中在圖中 A、B 兩處。



（紅色為訓練集資料，藍色為測試集資料）

我們嘗試新增 y 的平均值以及 y 的標準差欄位當作控制項，期望透過此方式使

模型有更好的解釋能力。

```
df2['mean_y']= np.log1p(df2["WEATHER_DELAY"]).mean()
df2['std_y']= np.log1p(df2.iloc[:,-1].values).std()
```

五、 模型績效評估

1. 預測航班是否因天氣延誤_二元分類模型

第一階段二元分類模型分別使用過採樣的 BorderlineSMOTE、ADASYN，欠採樣的 RandomUnderSampler，以及綜合採樣的 SMOTEENN、SMOTETomek 共 5 種不同方式處理類別不平衡問題，在其他條件固定的情況下(例如相同演算法、相同參數) 綜合採樣 SMOTETomek 擁有較高的 roc_auc 分數，與第二高的 ADASYN 分數相距並不大(約只有 0.001)，我們推測可能由於 ADASYN 能夠根據資料集的分布情況，自動決定每個少數類樣本需產生多少新樣本，而不是像 SMOTE 及 BorderlineSMOTE 對每個少數類合成相同數量的新樣本，因此能使最終模型比其他過採樣方式擁有較高的泛化能力。

最後我們依據 roc_auc 分數高低，採用了綜合採樣 SMOTETomek 的處理類別不平衡方式以及隨機森林演算法當作第一階段二元分類模型的最終使用模型。

```
SMOTETomek():

模型平均ROC AUC分數: 0.905
測試集ROC AUC分數: 0.865
----------------------------
混淆矩陣:
[[457  97]
 [  2  19]]
```

```
ADASYN():

模型平均ROC AUC分數: 0.903
測試集ROC AUC分數: 0.864
----------------------------
混淆矩陣:
[[456  98]
 [  2  19]]
```

```
SMOTEENN():

模型平均ROC AUC分數: 0.901
測試集ROC AUC分數: 0.851
----------------------------
混淆矩陣:
[[442 112]
 [  2  19]]
```

```
BorderlineSMOTE():

模型平均ROC AUC分數: 0.904
測試集ROC AUC分數: 0.800
----------------------------
混淆矩陣:
[[464  90]
 [  5  16]]
```

```
RandomUnderSampler():

模型平均ROC AUC: 0.903
測試集ROC AUC分數: 0.797
----------------------------
混淆矩陣:
[[408 146]
 [  3  18]]
```

2. 延誤多長時間_迴歸模型

新增 y 的平均值以及 y 的標準差欄位當作控制項，再使用 XGBoost 演算法搭配 GridSearchCV 找出最佳參數組合後，模型在測試集上的分數雖然不是非常好，但能看出已有明顯提升(0.25→0.58)。

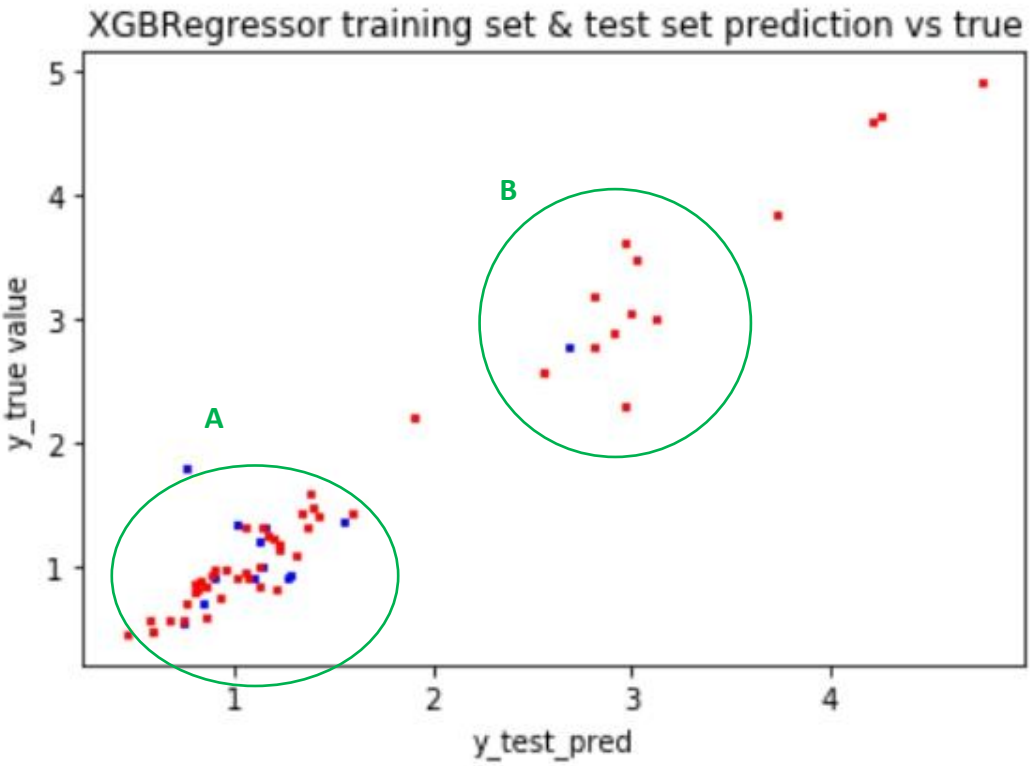
```
other_params = {'learning_rate':0.1,'n_estimators': 100, 'max_depth': 3, 'min_child_weight': 1,'seed':1850,
                'subsample': 0.8, 'colsample_bytree': 0.8,'colsample_bylevel':1,'reg_alpha':0.01,'reg_lambda':1,
                'gamma':0.2,'objective':"reg:linear"}

model = xgb.XGBRegressor(**other_params)
model.fit(X_trainval,y_trainval)  ## 默認rmse
```

（最佳參數組合）

```
訓練集mes: 0.21726556956093038
訓練集r2: 0.9668199875093347
------------------------------
測試集mes: 0.35341876074760964
測試集r2: 0.5858186153373053
```

重新畫訓練集、測試集的實際值與預測值散佈圖，能看出其實與原先尚未進行

新增 y 的平均值以及 y 的標準差欄位之分布結果無太大差異，資料始終集中在 A、B 兩處，我們推測可能是因為數據本身分布問題，導致模型在新數據集上始終表現非常糟糕，不具備良好的泛化能力。因此我們判斷此模型已為目前的最佳模型。



（紅色為訓練集資料，藍色為測試集資料）

六、 測試天氣數據是否能提高模型泛化能力

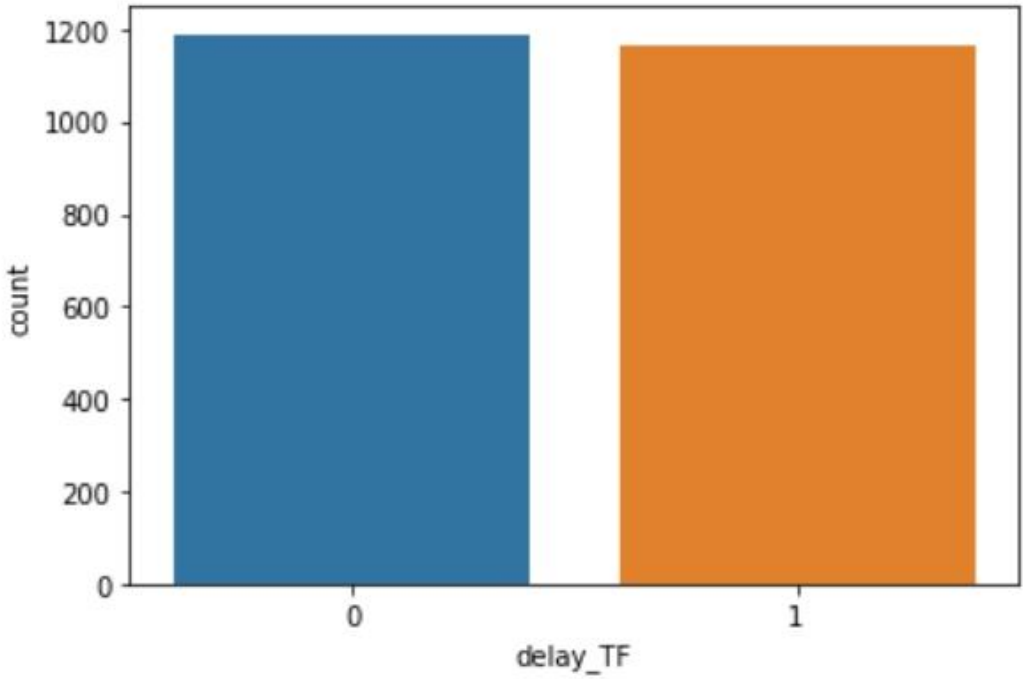
我們想研究在加入天氣相關數據的情況下，模型泛化能力是否真的能提高， 意即天氣數據能否有效幫助我們更準確預測航班的延遲狀況。

這部分我們不考慮航班因為何種原因導致延誤，而是針對所有延誤原因(也就是只看 ARR_DELAY 欄位)分析延誤狀況。

1. 預測航班是否延誤_二元分類模型

第一階段預測航班是否延誤部分我們透過 ARR_DELAY 欄位判斷航班是否延誤，並新增 delay_TF 欄位作為分類模型的 target，當 ARR_DELAY>0 時，表延誤，delay_TF 為 1 ； 當 ARR_DELAY<=0 時，表沒延誤，delay_TF 為 0。

查看資料分佈後，可看出類別比例平均，因此無須對樣本進行平衡處理。



接著使用與預測航班是否因天氣延誤之模型相同的演算法建模，發現 roc_auc 竟能達到 1。

```
model = XGBClassifier()
model.fit(X_trainval,y_trainval)
y_test_pred = model.predict(X_test)
roc_auc = roc_auc_score(y_test, y_test_pred)
print("測試集roc_auc:",roc_auc)
```

```
測試集roc_auc: 1.0
```

2. 延誤多長時間_迴歸模型

本階段亦使用 XGBoost 搭配 GridSearchCV 對航班延誤時間進行預測。整體調參流程也按照預測航班因天氣延誤多長時間的步驟進行。

下圖為經過完整調參後得出的最佳參數組合以及模型在訓練集、測試集上的分數:

```
other_params = {'learning_rate':0.1,'n_estimators': 400, 'max_depth': 3, 'min_child_weight': 3,'seed':1850,
                'subsample': 0.8, 'colsample_bytree': 1,'colsample_bylevel':1,'reg_alpha':0,'reg_lambda':1,
                'gamma':0.8,'objective':"reg:linear"}

model.fit(X_trainval,y_trainval)
```

（最佳參數組合）

```
訓練集mes: 0.6872956762988747
訓練集r2: 0.5910196332987395
----------------------------
測試集mes: 0.8481776244142936
測試集r2: 0.41930599880065356
```

（模型效能）

七、 結論

下表統整加入與未加入天氣相關數據之分類、迴歸模型在測試集上的得分(取到小數點後第 3 位):

	分類模型	迴歸模型
加入天氣相關數據	roc_auc: 0.865	mse: 0.353 r^2: 0.586
未加入天氣相關數據	roc_auc: 1	mse: 0.848 r^2: 0.419

雖然在未加入天氣數據的情況下，其分類模型的表現能力非常好，但當要預測航班延誤多長時間時，迴歸模型的泛化能力卻比有加入天氣數據的情況還來的差。

由此可知，加入了天氣數據是能夠有效幫助我們更準確預測航班的延遲狀況。