組別：鎮霆與他的可愛小夥伴

組員：劉鎮霆、簡宏曄、邢皓霆

一、問題分析及資料的觀察

- **題目設定**

  這是一個飯店 / 旅館預訂的資料集，並總共有 91531 筆資料，每筆資料有 hotel, is_canceled 等 32 個特徵，而我們的目標為預測收益 ADR 及客房取消率 is_canceleded。

- **簡單統計分析及觀察**

  在開始利用機器學習之技法前，我們先簡易的看過資料對其有初步的認識，包括檢查每筆資料的所有特徵資訊是否完整，以及計算出每個特徵之平均值、標準差、及對目標之相關係數。

- **預先處理**

  為了讓我們的分析具有顯著性及可驗證性，以及降低模型的複雜度以增加模型執行效率，我們根據資料特徵的分析，先對資料做過處理，處理後的資料才真正進入機器學習之階段，並根據預先處理過的資料比較若干個機器學習之模型表現，選出表現最好之模型。而後再利用這個選出之模型，重新檢視是否有更多的特徵得先經過預處理，並重新訓練所選定之模型以增加表現。

二、預處理（分為刪除之特徵　特徵之轉換　及增加之特增）

1. 刪除之資料

- 我們發現其中 children, country, agent company 這四項特徵之值有缺漏，且缺漏之比例分別為 0.000034, 0.004087, 0.136862, 0.943069。
- 對於 children 缺漏之資料，我們都將此筆數值設為 0 ，其原因為 children 為 0 的資料佔了 92.8049 %，因此將其設為 0 對整體資料的分佈影響性極低。
- 對於 country 之缺漏，因為所佔比例仍不算高，然而對於 country 的各個類別中，所佔比例最高的為 26%，並不像 0 個 children 一般佔了那麼極端的比例。因此我們這裡多新增了一個為 “NAN” 的類別，將資料缺失的部分都填上 “NAN” 的值作為其數據。
- agent 之缺漏，其所佔比例 13% 已經不容忽視。而我們相信其數據會缺失可能存在某些原因，因此資料缺失就也可以視為一個類型，並納入分析的考量。例如有可能具有值的特徵皆是旅行社，而不具有值的則為自由行的旅客等。
- 對於 company，因為缺失之資料高達 94%，我們難以確定這些資料為何缺失及分佈為何。因此在資料筆數太小的情況下我們直接刪除這個特徵。

2. 資料之轉換：label-encoding ＆ one-hot-encoding

- **實作方法：**

  為了能進行符合機器學習的輸入資料所需要的格式，我們將類別的特徵，例如國家是德國、西班牙、荷蘭，藉由 label-encoding 或 one-hot-encoding 轉換成數值。

  而兩個特徵轉換方式之時間及預測表現列於下表。而這裡需注意的事情是，因為我們認為 tree base 的演算法較適合處理這題（將於第三部分時解釋），因此這裡我們先選擇將這兩個資料處理方式套用到較基本的 tree base 的模型 decision tree 上做測試。

  由比較表格中可以發現，就時間上無論是 Ein、Eout，使用 one-hot-encoding 的時間皆大約為 label encoding 的十倍左右。然而在預測的準確度上，one-hot-encoding 之 Eout 卻僅為 label encoding 之三分之一到四分之一左右。雖然所需時間大了不少，然而實際時間大約為 10 秒內。而所預測的目標為一天之 ADR ，因此不會要重複高頻率的執行此預測，在此情況下 one-hot-encoding 的執行時間仍然能夠讓人接受。然而犧牲了一點程式執行效率所換來的，則是明顯顯著提升的準確度。

因此在這樣的權衡下，我們認為 one-hot-encoding 明顯相較於 label encoding 是較好的將類別特徵轉換成數值的方式。

⑩ **效能表現**

Time Comparison （單位：seconds）

	Training (ADR)	Predicting (ADR)	Training (is canceled)	Predicting (is canceled)
Label	0.687217	0.006828	0.628856	0.003813
One-hot	9.471948	0.043643	10.338217	0.039281

Performance Comparison （單位：label mae）

	E_in	E_out (public)	E_out (private)	E_out (average)
Label	0.015625	2.105263	2.000000	2.0526315
One-hot	0.015625	0.657895	0.415584	0.5367395

3. 新增之特徵

⑩ **特徵說明**

除了原先資料就已經有的特徵以外，我們還結合了原有特徵，而產生了其他兩個新的特徵。一個是這筆住房訂單的天數，另一個則為所預定的房型是否吻合。

關於第一個新加的特徵代表訂單的天數，是由原有的兩個特徵"weekday"和 "weekend" 所組成（也就是平日和假日各待了多少天）。然而原先資料卻無間直接顯現說這筆訂單的客人總共待了多少天，而待的時間長短可能隱含了顧客的類型等等資訊，因此我們將它額外加上去。

至於第二新加的特徵房型是否吻合，則為檢視原有的特徵中「所欲訂的房間」跟「實際給的房間是否一樣」。前原因是是否滿足客人要求可能會影響滿意度等，而對訂單收益等造成影響。

而實際加上這兩個特徵進行驗證後，在程式執行時間上，多加了這兩個特徵讓執行時間少量增加，然而仍在接受範圍內。而至於準確性可以發而外新增兩個特徵後 Eout 在平均的表現下雖有稍微提升， Eout 從 0.53 降為 0.5，然而幅度卻不是非常明顯。此外在 Eout (private) 裡加了這兩個特徵後反而變差從 0.41 增加為 0.44。因此我們認為新加的特徵對結果之影響不太顯著。

⑩ **效能表現**

Time Comparison （單位：seconds）

	Training (ADR)	Predicting (ADR)	Training (is canceled)	Predicting (is canceled)
w/o column	9.471948	0.043643	10.338217	0.039281
column	11.257811	0.083577	11.521708	0.049964

Performance Comparison （單位：label mae）

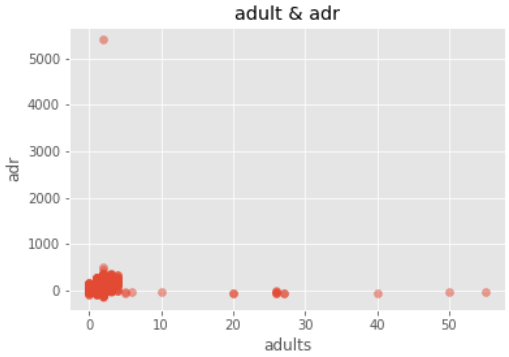
	E_in	E_out (public)	E_out (private)	E_out (average)
w/o column	0.015625	0.657895	0.415584	0.5367395

column	0.015625	0.565789	0.441558	0.5036735

三、三個學習方法之嘗試

⑩ 模型說明

我們在一開始時，先把所有特徵對於我們所要預測的 ADR 畫出散佈圖，例如下圖為 adults 對於 ADR 之散佈圖。觀察後我們發現特徵與目標之間非線性關係，因此我們認為此題的判斷與決策過程較適合使用 tree base 的演算法來訓練 model。



而我們實際使用的三個 tree base 模型為 decision tree, random forest 及 Xgboost，而三個演算法我們皆直接使用 sklearn 及 xgboost 裡的套件。我們將三個模型的參數的資訊列於下面，需注意的是為求版面簡潔，我們把同一個模型預測 adr 和 is_canceled 的部分結合，其中若那一個參數是預測 adr 或 is_canceled 都一樣的話就不做另外標示，若該行只屬於預測 adr (is_cancel) ，就在後面括號 adr (is_cancel)，其中需要注意的地方是，decision tree 的深度不限；random forest 裡有 100 顆樹，每顆樹的深度上限為 20；Xgboost 裡有 100 顆數，而深度上限為 6。而選擇這些參數的原因是經 5-fold cross validation 實驗後認為這些參數設定的表現會較好。

⑩ 參數設定

- **Random Forest**

```
params = {'bootstrap': True,
 'ccp_alpha': 0.0,
 'criterion': 'mse',
 'max_depth': 20,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
regr = RandomForestRegressor(**params)
```

```
params = {'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': 20,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
clf = RandomForestClassifier(**params)
```

- **XGBoosting**

```
regr = XGBRegressor(
    objective='reg:squarederror',
    n_estimators=100,
    learning_rate=.3,
    max_depth=6,
    min_child_weight=1,
    colsample_bytree=.5,
    subsample=.5,
    gamma=10,
    n_jobs=-1,
    random_state=1126,
    tree_method='gpu_hist',
    predictor='gpu_predictor',
    deterministic_histogram=False
)
```

```
clf = XGBClassifier(
    objective='count:poisson',
    n_estimators=100,
    learning_rate=.3,
    max_depth=6,
    subsample=.7,
    colsample_bytree=.7,
    gamma=1,
    max_delta_step=0.699999988,
    n_jobs=-1,
    random_state=1126,
    tree_method='gpu_hist',
    predictor='gpu_predictor',
    deterministic_histogram=False
)
```

- **Decision Tree**

```
params = {'ccp_alpha': 0.0,
 'criterion': 'mse',
 'max_depth': None,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'presort': 'deprecated',
 'random_state': None,
 'splitter': 'best'}
regr = DecisionTreeRegressor(**params)
```

```
params = {'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'presort': 'deprecated',
 'random_state': None,
 'splitter': 'best'}
clf = DecisionTreeClassifier(**params)
```

⑩ 效能表現

Time Comparison （單位：seconds）

	Training (ADR)	Predicting (ADR)	Training (is canceled)	Predicting (is canceled)	Total
Decision Tree	11.257811	0.083577	11.521708	0.049964	22.913060
Random Forest	504.716167	1.153261	37.657469	0.889433	544.416330
Xgboost	67.931374	0.202746	71.622965	0.202000	140.059085

Performance Comparison （單位：label mae）

	E_in	E_out (public)	E_out (private)	E_out (average)
Decision Tree	0.015625	0.565789	0.441558	0.5036735
Random Forest	0.353125	0.460526	0.467532	0.464029
Xgboost	0.2234375	0.394737	0.363636	0.3791865

最後我們把三個模型分為四個點去做比較，分別為 efficiency、scalability、popularity、interpretability。

- **Efficiency：**

意即所花的時間，根據我們的實驗結果 decision tree 的所需時間皆最少。至於 Random forest 及 Xgboost 之比較，因為訓練出一個 rank 的結果我們須分別訓練和預測過 ADR 及 is_canceled，所以我們比較時把四個數值加總，得到 decision tree 所需要的時間為 0.5036735， random forest 的所需時間為 544.416330，而 Xgboost 的所需時間為 140.059085，因此 Xgboost 在這個問題的情況下較 random forest 有效率。因此三個模型效率之排序為 decision tree > Xgboost > random forest。

- **Scalability：**

也就是模型在不同資料大小的表現，我們針對不同的資料數量做出了比較，如下表。從表中可以發現，無論在什麼資料大小下執行效率皆是 decision tree > Xgboost > random forest，亦即其順序跟 efficiency 之順序一樣。至於記憶體的使用量，則為 Xgboost > decision tree > random forest，但差異相對較小。

Time Comparison（單位：sec）

	Training (ADR)	Training (is canceled)
Decision Tree (100% dataset)	11.257811	11.521708

Decision Tree (10% dataset)	0.470403	0.395853
Decision Tree (1% dataset)	0.024861	0.021504
Random Forest (100% dataset)	504.716167	37.657469
Random Forest (10% dataset)	22.706029	2.200745
Random Forest (1% dataset)	1.382033	0.263528
Xgboost (100% dataset)	67.931374	71.622965
Xgboost (10% dataset)	8.486898	9.948263
Xgboost (1% dataset)	0.642588	0.677164

Maximum memory usage（單位：mb）

	Training (ADR)	Training (is canceled)
Decision Tree	2371.941406	2372.414062
Random Forest	2607.175781	2821.289062
Xgboost	3150.773437	2934.132812

- **Popularity**

  我們用這幾個方法的名稱在網路上有多少相關資料以及被多少人搜尋過，來衡量他被廣為人知或受喜愛的程度。因此我們用這個詞彙在 google 搜尋後跳出的資料數，以及在 google trend 上所顯示出的被搜尋的次數，作為衡量的指標。關於 google 跳出的資料數為：decision tree 有 824,000,000 筆，random forest 有 606,000,000 筆，Xgboost 則有 1,170,000 筆。至於在 google trend 被搜尋的次數，若為全世界的加總，其結果跟資料數所顯示的一樣，搜尋次數的多寡為：decision tree > random forest > Xgboost。然而搜尋次數若細看不同區域或國家，國與國之間卻有著極大的差異。例如以世界三大強國為例，搜尋的比例（ decision tree：random forest : xgboost) 分別是：美國為 48:31:21，中國為 11:11:78，台灣則為 28 : 35 : 37。可見不同區域對不同模型的偏好程度有著極度的差異。

- **Interpretability**

  也就是代表最為受大眾理解且能直觀解釋其意義的程度。我們認為這三個 model 的 interpretability 的程度為 decision tree > random forest > Xgboost。其原因為決策樹便為模擬人類的決策過程，按照選項慢慢縮小範圍找到答案，因此也最為容易受人理解。而 random forest 則如同民主制度投票一樣，因此其運作方式也較容易理解，相對起來 Xgboost 就較難用生活經驗做為比擬。而在資料的佐證上，延續在 popularity 的討論，也就是在 google 上的資料數及搜尋數，我們認為越容易理解的模型，其越會廣為人知及喜愛，因此會有越多之搜尋及資料數。因此 interpretability 的排序會跟 popularity 的排序一樣。最後將全世界對於三個數的搜尋次數繪圖於下：

而最後將 efficiency、scalability、popularity、interpretability 之內容彙整如下。

	Efficiency	Scability (space)	Popularity	Interpretability
Decision Tree	最佳	最佳	最佳	最佳
Random Forest	最差	次之	次之	次之
Xgboost	次之	最差	最差	最差

四、此問題設定下推薦之最佳模型

根據以上種種分析，在這題問題的設定下，我們推薦使用 Xgboost。首先就優點而言，其有著最好的 Eout 也就是準確性最高，雖然訓練的時間比 decision tree 較長，但相比 random forest 還是有很不錯的表現。另外 Xgboost 在處理大筆資料時，也會有很好的表現，也就是有好的 scalibility。此外，Xgboost 因為只注重 example 的 ordering，所以離群值對於整體表現的影響不大。雖然在 efficiency、scalibility、popularity 及 interpretability 上未有突出的表現，但我們認為相比準確率的提升，這些仍在可接受的範圍內。

然而就缺點而言，Xgboost 有著較多的參數，因此不見得那麼好入手。此外 Xgboost 容易 overfitting，而也是這原因我們這次模型的設計相較於 random forest 每棵樹能有 20 層，Xgboost 卻得壓到 6 層以下，因此使用上更加容易出錯。最後 Xgboost 也較 decision tree 和 random forest 而顯得更不直觀難以解釋，而此次題目因為設計的應該是公司內部系統等，而非需要向廣大民眾解釋的模型，才犧牲了可解釋性而追求較好的表現。然而在病情分析等得告知病人因果的情況下，這樣的模型可能就較為不適用。

五、工作分配

- 劉鎮霆：初期程式建構及資料搜尋
- 簡宏曄：中後期程式優化及參數調整
- 邢皓霆：執行表現分析及報告撰寫

六、引用及參考 (Citation & Reference)

- 我們使用了 sklearn 的 package，因此在 citation 上附上 sklearn 的 paper：
  Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.

- 另外我們也使用了 xgboost 的 package，因此在 citation 上附上 xgboost 的 paper：
  XGBoost: A Scalable Tree Boosting System, Tianqi Chen, Carlos Guestrin, arXiv:1603.02754v3 [cs.LG] 10 Jun 2016

- 在預處理的時候參考了：
  https://www.kaggle.com/prasannavijayakumar/predicting-hotel-booking-cancellation-lstm

- 在選擇 model 的時候參考了：
  https://www.kaggle.com/kaushikmurthi/hotel-booking-eda-and-prediction?fbclid=IwAR1PZH0AMAtiVXZhdtdBMZTfaSwO4-8D5LSexITLgEATxxJRP4wG7xo3MQQ