

機器學習導論期末分析報告書

巨資四 B, 04170217, 胡弘林

巨資四 B, 04170228, 楊文瀚

巨量析 2, 06571006, 蘇成恩

1 Introduction

本報告書分析的資料為 Kaggle 上的 New York City Taxi Fare Prediction 競賽，Training data 共有 5 千 5 百多萬筆資料，其中共有 8 個欄位，如下圖

	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	2009-06-15 17:26:21.000000100	4.5	2009-06-15 17:26:21	-73.844311	40.721319	-73.841610	40.712278	1
1	2010-01-05 16:52:16.000000200	16.9	2010-01-05 16:52:16	-74.016048	40.711303	-73.979268	40.782004	1

分別為 ID、車費、搭乘時間、搭乘地點經緯度、抵達地點經緯度以及載客數量，而最後要預測的變項為車費。為了節省分析時間以及電腦效能，因此我們切出 Training data 其中的 10 萬筆資料來做資料的前處理以及 EDA，最後再套用回 5000 多萬筆的總資料。

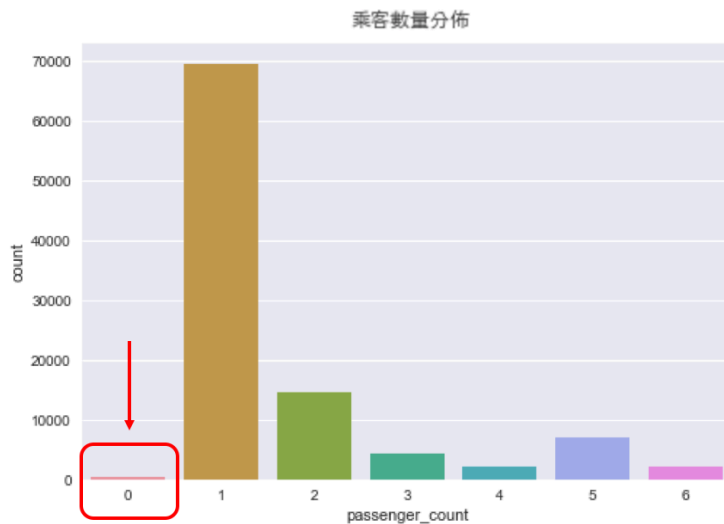
2 Exploratory data analysis for 10 萬筆資料

(a) 基本統計指標

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	99999.000000	99999.000000	99999.000000	99999.000000	99999.000000	99999.000000
mean	11.354708	-72.494667	39.914473	-72.490952	39.919044	1.673827
std	9.716809	10.693986	6.225716	10.471438	6.213457	1.300175
min	-44.900000	-736.550000	-74.007670	-84.654241	-74.006377	0.000000
25%	6.000000	-73.992041	40.734996	-73.991215	40.734182	1.000000
50%	8.500000	-73.981789	40.752765	-73.980000	40.753243	1.000000
75%	12.500000	-73.966982	40.767258	-73.963433	40.768164	2.000000
max	200.000000	40.787575	401.083332	40.851027	404.616667	6.000000

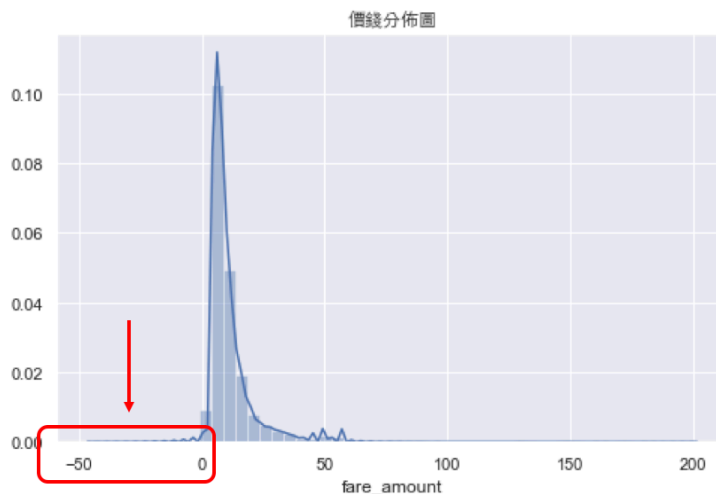
→ 從上述可以發現在經緯度的資料當中存在異常值，經度(longitude)的範圍應在-180~180 之間，而緯度(latitude)的範圍應在-90~90 之間，因此我們首先去除資料中有異常經緯度的筆數，在 10 萬筆當中約有 15 筆異常值。

(b) 載客數量分佈圖



→ 乘客數量以 1~6 人為主要的分佈，不過也出現的 0 的少量異常值，初步猜想是人為輸入錯誤或是儀器錯誤。

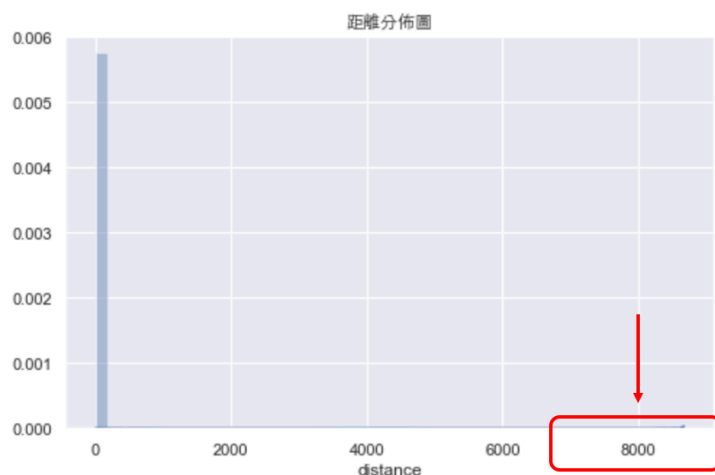
(c) 價錢分佈圖



→ 價錢分佈大多為 0~20 元左右，其中也出現的價格為負的異常值

(d) 新增距離欄位(distance)

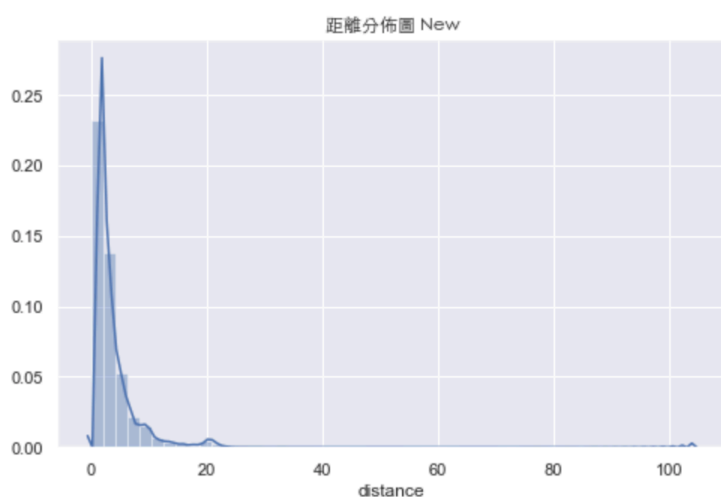
→ 利用搭車地點的經緯度以及下車地點的經緯度做距離轉換，雖然算法是以兩者的直線距離為轉換結果，無法正確表達出計程車實際的行駛距離，但我們初步認為仍具有一定的參考性，就地點與地點間的直線距離越遠，價格越高，反之越近則價格越低。



→ 查看距離的分佈圖後發現有距離異常高的狀況，反查資料後發現如下

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	distance
472	2.5	0.000000	0.000000	-74.005433	40.726685	2	8667.655928
1260	5.7	-73.973907	40.754743	0.000000	0.000000	2	8665.602660
2396	45.0	0.000000	0.000000	-74.010230	40.714553	2	8667.810452

→ 出現位於赤道的異常經緯度，因此對於經緯度的處理不能只有限制在正確的範圍內而已，我們的處理方式為先查看 Testing data 中的經緯度範圍為何，接著我們只要有包含到 Testing data 的經緯度資料就好，其他全部予以刪除，而我們限縮的經度範圍為-74.5~-72.8，而緯度範圍為 40.5~41.8，如此一來不僅能去除定位錯誤的異常值，也能涵蓋 Testing data 的經緯度，下圖則為新的距離分佈圖



→ 從上圖可以看出大部分搭乘的距離集中在 5~15 公里的範圍內

(e) 新增時間欄位

→ 我們也從搭車時間 (pickup_datetime) 的欄位當中取出年份、月份、星期以及小時當作新的欄位變項，如下圖示例。

	pickup_datetime	year	month	weekday	hour
0	2009-06-15 17:26:21	2009	6	Monday	17
1	2010-01-05 16:52:16	2010	1	Tuesday	16
2	2011-08-18 00:35:00	2011	8	Thursday	0

3 Data pre-processing for all data

(a) 資料轉換

→ 所有的 Training data 總共有 5.7GB，匯入 python 當中總共需要 1min 28s 左右，不只緩慢也佔了不少的記憶體，因此我們將此份 csv 轉檔成 feather 格式，空間可以壓縮至 1.6GB 左右，減少記憶體的佔據，同時匯入 python 僅需 23 秒左右，增加分析的時間。

(b) 刪除遺失值

→ 原先的 10 萬筆資料中並無出現遺失值，而所有資料當中共出現了 376 筆遺失值，因此先予以刪除。

(c) 限制乘客數量的範圍為 1~6 人

→ 原先的 10 萬筆資料中有出現 0 人的情況，而在所有資料中又更加凌亂了，有出現 200 多人等的異常值，因此同樣予以刪除。

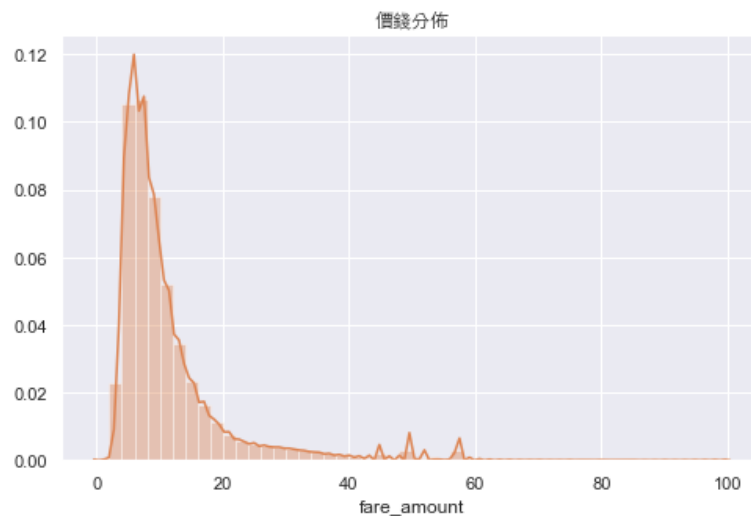
(d) 刪除價格為負的資料

(e) 限制經緯度範圍在 (-74.5, -72.8, 40.5, 41.8) 之間

(f) 新增距離、年份、月份、星期、小時欄位變項

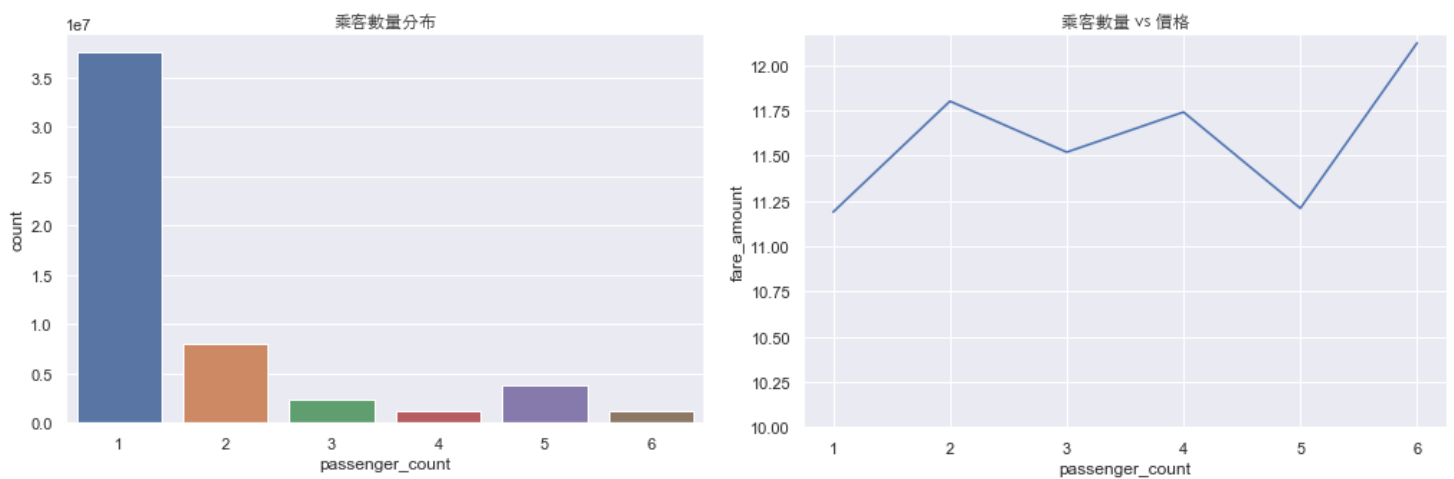
4 Exploratory data analysis for all data

(a) Feature 1 – 價格



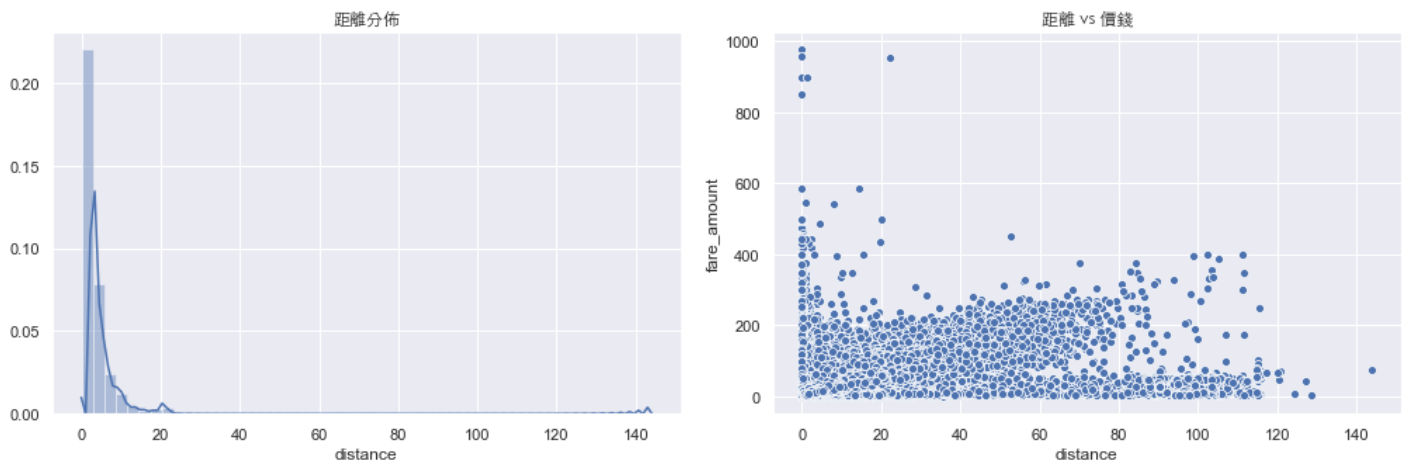
→ 在所有資料中價錢分佈與 10 萬筆類似，大部分都是集中在 0~20 元之中

(b) Feature 2 – 乘客數量



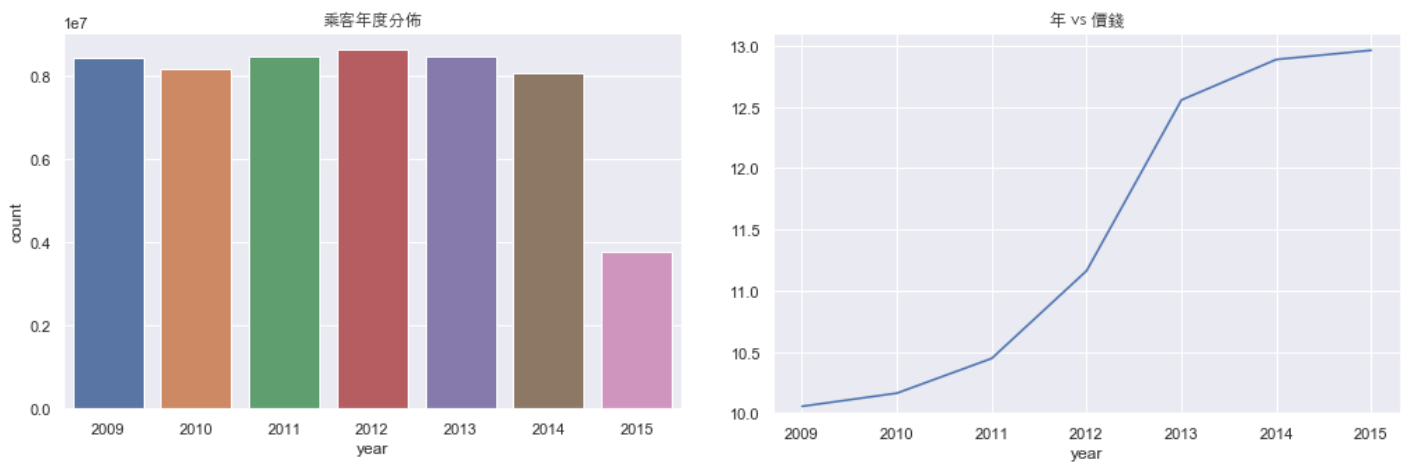
→ 在左圖中我們可以看出此份資料大部分的乘客都是 1 人，而右圖可以看出載客數量的不同確實有影響到價格的不同

(c) Feature 3 – 距離



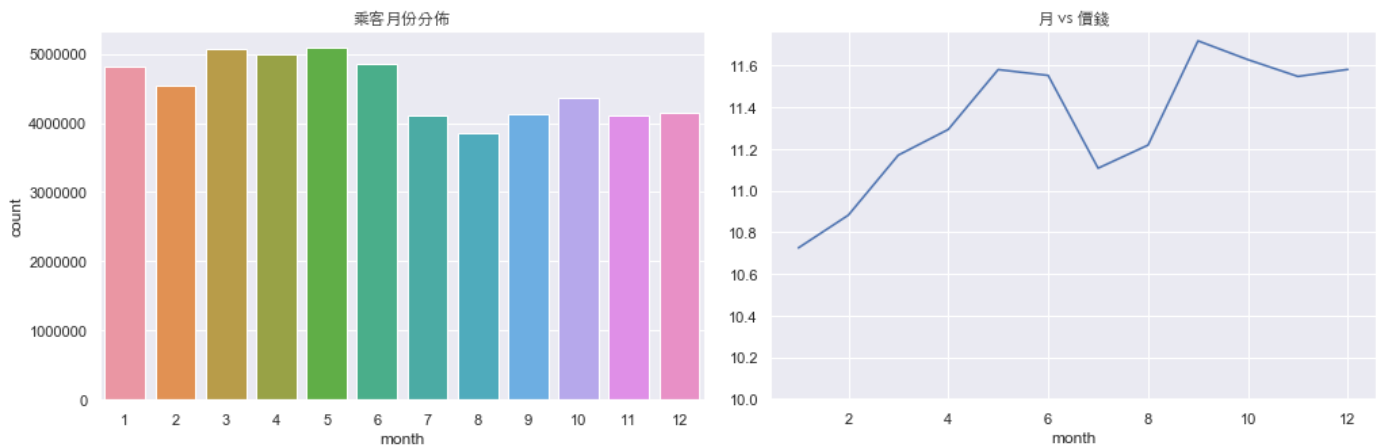
→ 左圖與原先的 10 萬筆也類似，搭乘距離幾乎集中在 20 公里內左右，右圖則無法明顯看出距離與價格存在什麼樣的關係

(d) Feature 4 – 年份



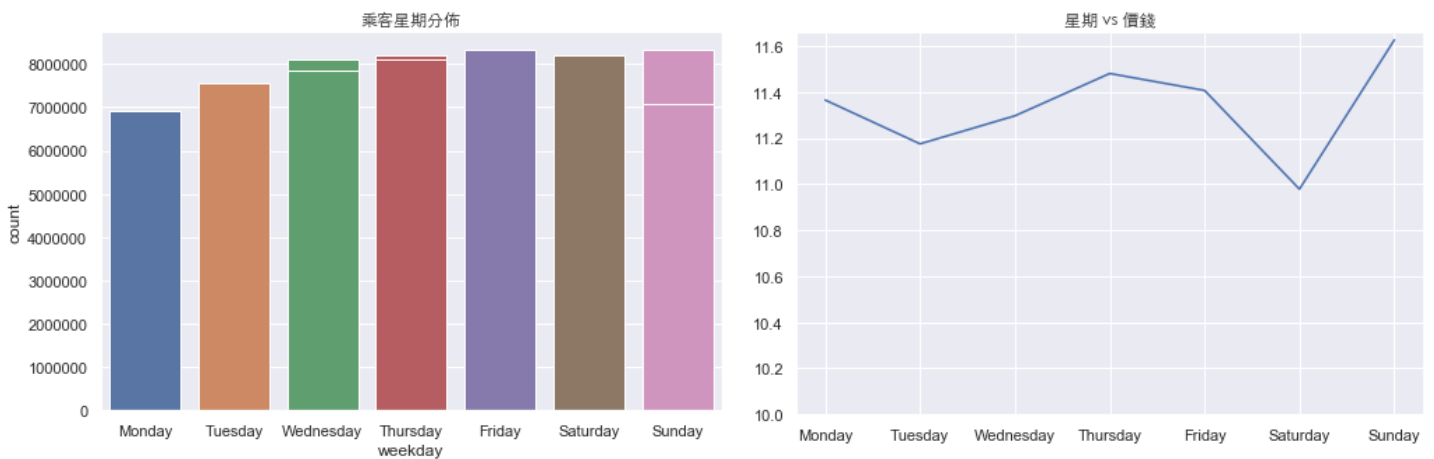
→ 左圖可以看出個年度的乘車人數大致相同，無差距太大，2015 年只剩一半是因為此份資料並無搜集完 2015 年的資料，而右圖明顯可以看出計程車的車資隨著年份而上漲，可以代表年份對於預測車資而言會是一個很好的變項。

(e) Feature 5 – 月份



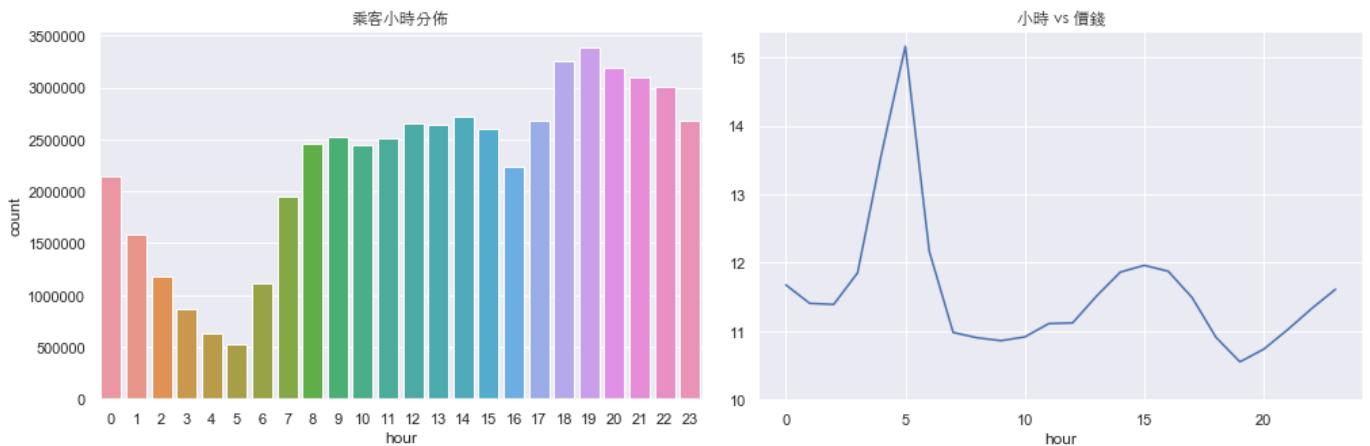
→ 左圖可以看出搭乘計程車的人數在 3~6 月是比較多的，而右圖我們發現計程車車資也是會隨著月份的增加而大致上漲，唯獨在 7 月的時候平均價格較低，是一個有趣的觀點！同時也告訴我們月份對於價格也是存有影響力。

(f) Feature 6 – 星期



→ 左圖可以看出在星期一和二的人數相對較少，右圖則發現星期六時平均的車資為最低的時候，但星期日平均卻為最高。

(g) Feature 7 – 小時



→ 左圖可以看出搭車人數在晚間 6~8 點時為最尖峰的時刻，凌晨則為最少，但從右圖中可以發現在凌晨 5 點之時的車資卻為最高，也是一個蠻有趣的觀點。

5 Data modeling

(a) Model 1 – Random Forest Regression for 100 萬 training data

→ 我們訓練過全部的 5000 多萬筆資料但發現電腦負載量過大，需花許多時間建成模型，因此先隨機挑選其中的 100 萬筆資料出來建置模型，至少可以先利用些指標來評估模型的好壞，模型建置大約需 1min 30s 左右，評估指標如下

```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test.values, y_predict_rfr)
print("MSE : ",mse)
```

MSE : 14.968006064436619

```
from math import sqrt
rms = sqrt(mean_squared_error(y_test.values, y_predict_rfr))
print("RMSE : ",rms)
```

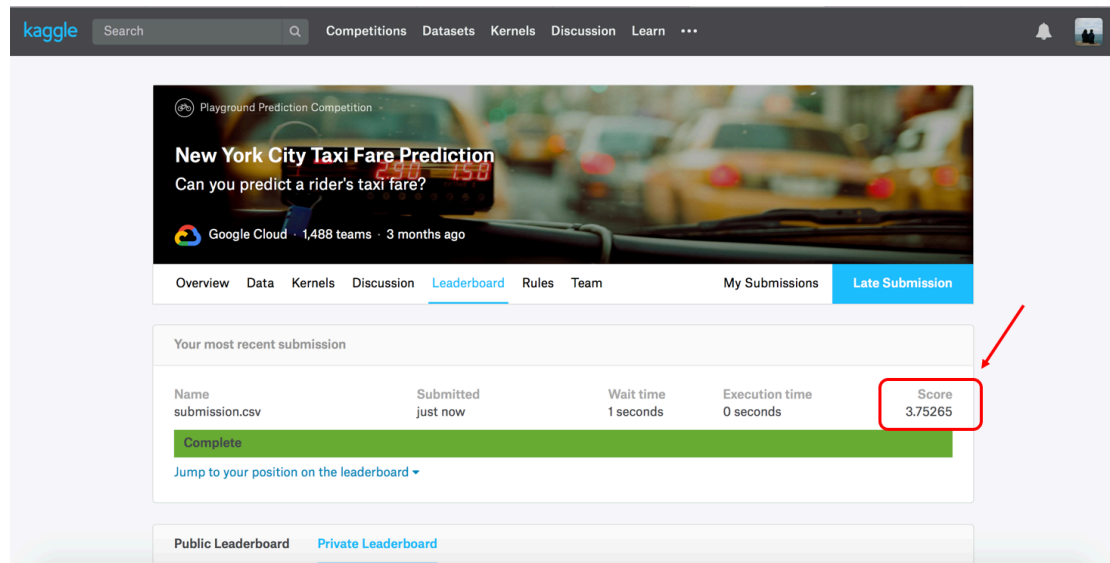
RMSE : 3.8688507420727176

```
R_2 = rfr.score(X_train, y_train)
adj_R_2 = R_2 - (1 - R_2) * (X_train.shape[1] / (X_train.shape[0] + 1))
print("Adjusted R-squared : ",adj_R_2)
```

Adjusted R-squared : 0.9699264753216053

→ MSE 和 RMSE 誤差因還沒有其他對照組比較，因此無法確定其好壞標準，

但從調整後的判定係數來看，可以發現我們的變項篩選確實是很具有代表性的，透過這 10 個變項解釋力竟高達 96%，我們也將第一版的模型預測 testing data 並丟上 kaggle 查看分數，而分數為 3.75265，大約排名在 800 左右。



(b) Model 2 – Random Forest Regression for 500 萬 training data

→ 我們透過同樣的演算法新增訓練的資料至 500 萬筆，模型建置大約需 9min 44s 左右，模型評估指標如下

```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test.values, y_predict_rfr)
print("MSE : ",mse)
```

MSE : 15.124094325419247

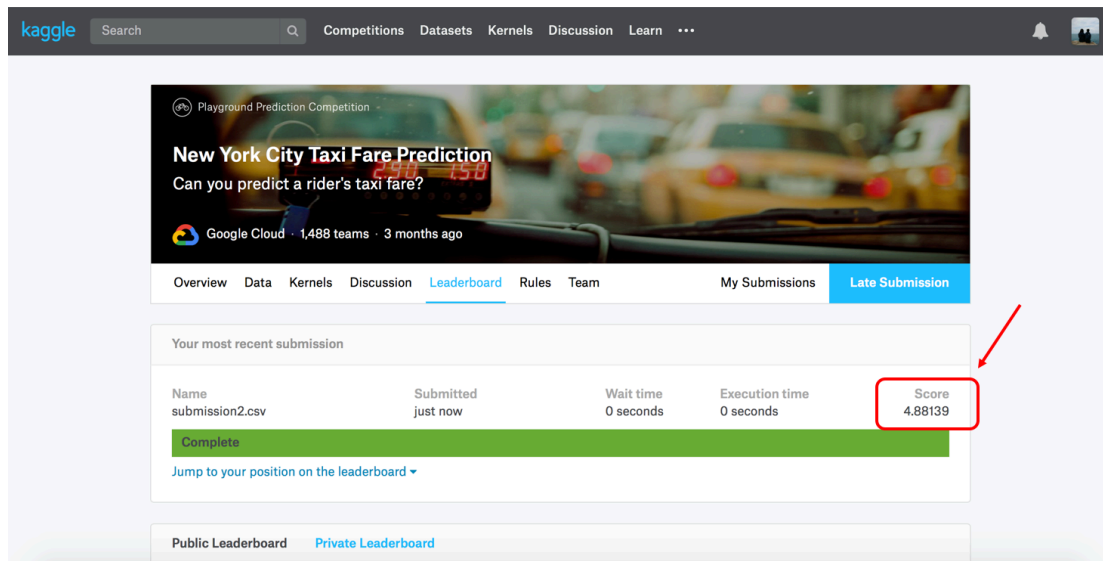
```
from math import sqrt
rms = sqrt(mean_squared_error(y_test.values, y_predict_rfr))
print("RMSE : ",rms)
```

RMSE : 3.8889708568487946

```
R_2 = rfr.score(X_train, y_train)
adj_R_2 = R_2 - (1 - R_2) * (X_train.shape[1] / (X_train.shape[0]
print("Adjusted R-squared : ",adj_R_2)
```

Adjusted R-squared : 0.9701757324592529

→ MSE 和 RMSE 我們可以看到比原先的模型來的稍微高一些，與我們原先預期的會下降並不同，而在調整後的判定係數上仍維持 96~97%的解釋力，與我們預期的相同，畢竟是使用同樣的變項建置模型，最後丟上 kaggle 查看分數



→ 意外的是分數竟然比原先的還要低，而且低了不少，我們猜測是因為抽樣誤差的關係導致

(c) Model 3– XGBoost Regression for 500 萬 training data

→ 我們嘗試換了一種模型 XGBoost Regression，訓練資料一樣給予 500 萬筆 data，模型建置大約需 8min 30s，模型評估指標如下

```
from sklearn.metrics import mean_squared_error
# mse = mean_squared_error(y_test.values, y_predict_rfr)
mse = mean_squared_error(y_test.values, y_predict_xgb)
print("MSE : ",mse)

MSE : 15.754254

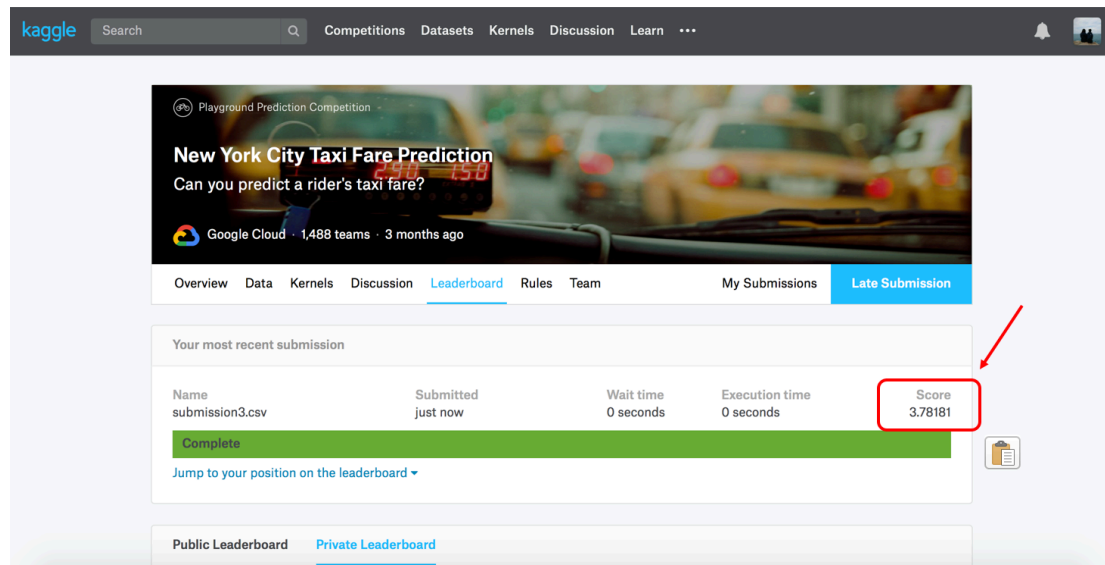
from math import sqrt
# rms = sqrt(mean_squared_error(y_test.values, y_predict_rfr))
rms = sqrt(mean_squared_error(y_test.values, y_predict_xgb))
print("RMSE : ",rms)

RMSE : 3.969162927006838

# R_2 = rfr.score(X_train, y_train)
R_2 = xgb.score(X_train.values, y_train.values)
adj_R_2 = R_2 - (1 - R_2) * (X_train.shape[1] / (X_train.shape[0] + 1))
print("Adjusted R-squared : ",adj_R_2)

Adjusted R-squared : 0.8296406535499343
```

→ 調整後的判定係數在 XGBoost 中表現的沒有比 Random Forest Regression 來的好，丟上 kaggle 後的分數為 3.78181，也沒有模型 1 的效果來的好



(d) Model 4 – XGBoost Regression for 所有 5400 萬 data

→ 我們在最後使用的所有的資料去訓練模型，總共花了約 4 個半小時完成，模型評估指標如下

```
from sklearn.metrics import mean_squared_error
# mse = mean_squared_error(y_test.values, y_predict_rfr)
mse = mean_squared_error(y_test.values, y_predict_xgb)
print("MSE : ",mse)
```

MSE : 18.442097

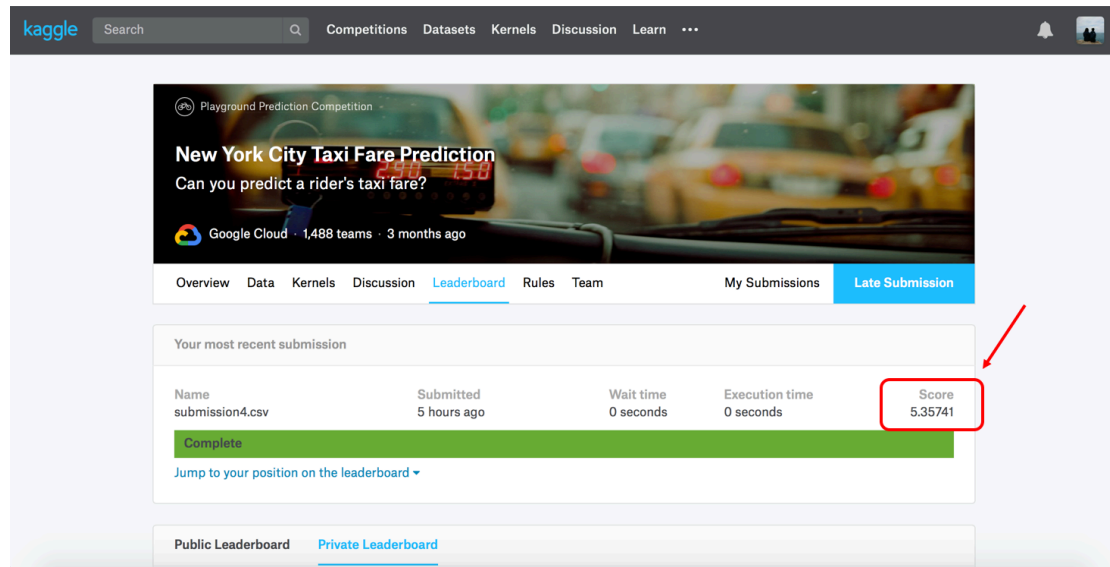
```
from math import sqrt
# rms = sqrt(mean_squared_error(y_test.values, y_predict_rfr))
rms = sqrt(mean_squared_error(y_test.values, y_predict_xgb))
print("RMSE : ",rms)
```

RMSE : 4.294426237602071

```
# R_2 = rfr.score(X_train, y_train)
R_2 = xgb.score(X_train.values, y_train.values)
adj_R_2 = R_2 - (1 - R_2) * (X_train.shape[1] / (X_train.shape[0] + 1))
print("Adjusted R-squared : ",adj_R_2)
```

Adjusted R-squared : 0.8035955453841719

→ 我們發現調整後的判定係數在 XGBoost 中皆不如原本使用的 Random Forest Regression，但 Kaggle 上的分數只有 5.3 左右令我們比較意外



6 Conclusion

→ 我們在分析當中得出了幾個結論

1. 我們認為訓練比數的增加不代表能讓模型的預測能力越準確，訓練越多反而越有可能 overfitting
2. 我們並未完全了解紐約市的計程車生態，也許像是機場到某個景點間是有固定的收費價格的，抑或是中間會經過固定的收費站等等，因此日後若要持續分析，我們並不能單單使用兩者的直線距離當變相而已，也許能匯入 google 地圖去判斷經緯度到經緯度之間會經過的路而算出真正距離，抑或是還有其他我們目前未發現到的車資規則也可以加入
3. 我們目前並未調整參數，也許能藉由調整模型參數而提高分數，而從我們的結果分析來看使用 Random Forest Regression 會是效果最好的演算法