# 班機抵達延誤分析

| 陳光源 | 109034402 |

Prediction of Flight Arrival Delays

指導教授: 邱銘傳教授


CONTENTS

# 目錄

1

PART ONE

# 研究背景

5W1H

# 研究背景

航班延誤
高額的賠償、客戶滿意度、機場鬧事
致使航空公司不僅償付賠款還會面臨客戶流失的風險

神經網路建立航班抵達延誤預測模型

幫助消費者選擇較少班機延誤的航班與時段

協助航空公司了解班機延誤的主因並提早進行預防以降抵損失

# 5W1H

**WHAT**
班機抵達延誤預測

**WHERE**
美國運輸部交通統計局

**WHEN**
購買機票時可根據預測結果的協助做出選擇

**WHY**
2007 年美國 NAS 預估班機延誤造成航空公司損失 330 億美金

**WHO**
欲購買機票的個人或旅行社業者

**HOW**
透過深度學習分析資料，建立航班抵達延誤預測模型

2

PART TWO

# 研究方法

# 研究方法

美國運輸部交通統計局

在kaggle的公開資料

資料來源

01

資料前處裡

02

資料轉換與刪除

以基本統計與視覺化數據

釐清資料資訊結構與特性

探索性資料分析

03

資料預處理

04

資料編碼

資料樣本區分

標準化

# 資料集

```
RangeIndex: 1048575 entries, 0 to 1048574
Data columns (total 31 columns):
 #   Column               Non-Null Count    Dtype
---  ------               --------------    -----
 0   YEAR                 1048575 non-null  int64
 1   MONTH
 2   DAY
 3   DAY_OF_WEEK
 4   AIRLINE
 5   FLIGHT_NUMBE
 6   TAIL_NUMBER
 7   ORIGIN_AIRPO
 8   DESTINATION_
 9   SCHEDULED_DE
 10  DEPARTURE_TI
 11  DEPARTURE_DELAY      1009060 non-null  float64
 12  TAXI_OUT             1008346 non-null  float64
 13  WHEELS_OFF           1008346 non-null  float64
 14  SCHEDULED_TIME       1048573 non-null  float64
 15  ELAPSED_TIME         1005504 non-null  float64
 16  AIR_TIME             1005504 non-null  float64
 17  DISTANCE             1048575 non-null  int64
 18  WHEELS_ON            1007279 non-null  float64
 19  TAXI_IN              1007279 non-null  float64
 20  SCHEDULED_ARRIVAL    1048575 non-null  int64
 21  ARRIVAL_TIME         1007279 non-null  float64
 22  ARRIVAL_DELAY        1005504 non-null  float64
 23  DIVERTED             1048575 non-null  int64
 24  CANCELLED            1048575 non-null  int64
 25  CANCELLATION_RE
 26  AIR_SYSTEM_DELA
 27  SECURITY_DELAY
 28  AIRLINE_DELAY
 29  LATE_AIRCRAFT_D
 30  WEATHER_DELAY
dtypes: float64(16),
```

```
In [4]: print(air.info())
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14 entries, 0 to 13
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   IATA_CODE   14 non-null     object
 1   AIRLINE     14 non-null     object
dtypes: object(2)
```

```
In [6]: print(airport.info())
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 322 entries, 0 to 321
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   IATA_CODE   322 non-null    object
 1   AIRPORT     322 non-null    object
 2   CITY        322 non-null    object
 3   STATE       322 non-null    object
 4   COUNTRY     322 non-null    object
 5   LATITUDE    319 non-null    float64
 6   LONGITUDE   319 non-null    float64
dtypes: float64(2), object(5)
```

airlines.csv

14筆資料

2 個欄位

flights.csv

1048574 筆資料

31 個欄位

airports.csv

332 筆資料

7 個欄位

# 資料前處理 – 資料轉換與刪除

```
df['SCHEDULED_DEPARTURE'] = create_flight_time(df, 'SCHEDULED_DEPARTURE')
df['DEPARTURE_TIME'] = df['DEPARTURE_TIME'].apply(format_hour)
df['SCHEDULED_ARRIVAL'] = df['SCHEDULED_ARRIVAL'].apply(format_hour)
df['ARRIVAL_TIME'] = df['ARRIVAL_TIME'].apply(format_hour)


# 時間：年月日周
print(df['DAY_OF_WEEK'].value_counts())
print(df['MONTH'].value_counts())
import calendar
df['DAY_OF_WEEK'] = df['DAY_OF_WEEK'].apply(lambda x: list(calendar.day_name)[x-1])

df['MONTH'] = df['MONTH'].apply(lambda x: calendar.month_abbr[x])
```

### 時間欄位

將日期時間欄位YEAR, MONTH, DAY轉換成DATE，並將月和星期轉換為各自對應的月份與星期名稱；以便後續探索性資料分析圖形化使用。

檢查各時間相關欄位後發現年(YEAR)只有2015年；月(MONTH)僅有Jan, Feb, Mar三種類型;星期(DAY_OF_WEEK)則有完整的種類。

```
print('Number of cancelled flights(df["CANCELLED"] = 1): ')
print(df['CANCELLED'].value_counts())
# 因為已取消航班沒有起飛降落 -> 刪除已取消航班的row
df = df[(df['CANCELLED'] == 0)]
# 再次檢查有無已取消航班
# 班機數訊息可從AIRLINE得知量 -> 刪除
print(df['CANCELLED'].value_counts())
df = df.drop(columns=['CANCELLED'])
```

### 已取消航班

因為已取消航班沒有起飛降落，所以刪除已取消航班的對應資料，並再次檢查。

```
Check Total Number of Airlines in flights.csv: 14
Check Total Number of Airlines in airlines.csv: 14
```

### 合併資料集

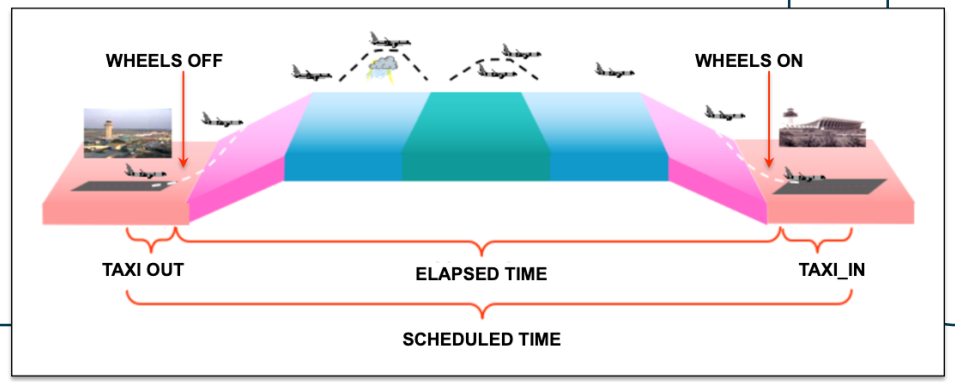
檢查兩資料集航空公司數量，都有14家；並合併資料，使原有資料航空公司名稱完整。

# 資料前處理 – 資料轉換與刪除

```
# DIVERTED: 轉機
print('Number of diverted flights(df["DIVERTED"] = 1): ')
print(df['DIVERTED'].value_counts())
df_DIV = df[df['DIVERTED'] == 1]
print(df_DIV['DIVERTED'].nunique())
print(df_DIV['ARRIVAL_DELAY'].nunique())
# 因與班級延遲資料意義一樣 -> 刪除轉機
# 與DEPARTURE_TIME 和 ARRIVAL_TIME 意義一樣 -> 刪除滑行、滑出、起飛
df = df.drop(columns=['DIVERTED', 'TAXI_OUT', 'WHEELS_OFF', 'WHEELS_ON', 'TAXI_IN'])
```

### 轉機

經過檢查發現與航班抵達延誤資料欄意義相同，所以刪除轉機資料欄位。同時刪除滑行、滑出與起飛時間等可從航班抵達延誤欄為獲取相同意義的資料欄位。

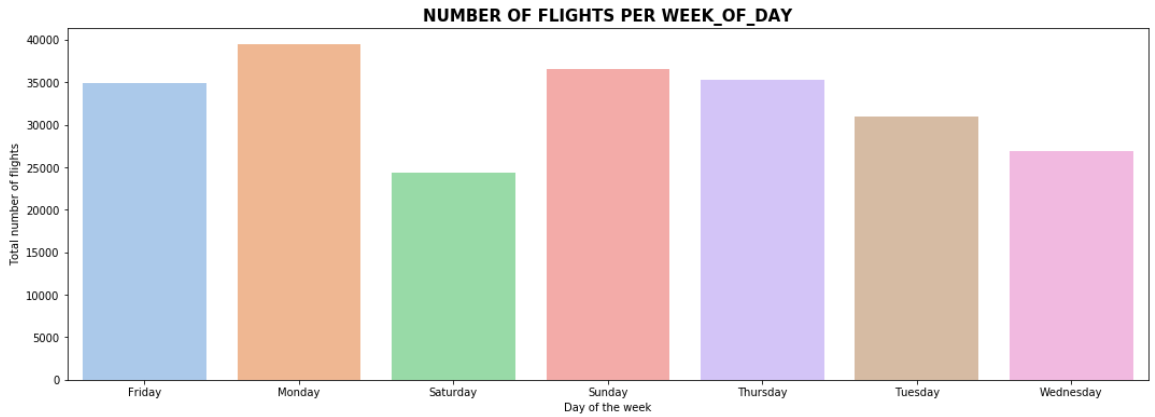



```
# Dealing with missing values
print('Summation of missing values in flights')
print(df.isnull().sum())
df = df.dropna()
df = df.reset_index()
df = df.drop(columns=['index'])
df.info()
print('New shape of df ', df.shape)
```

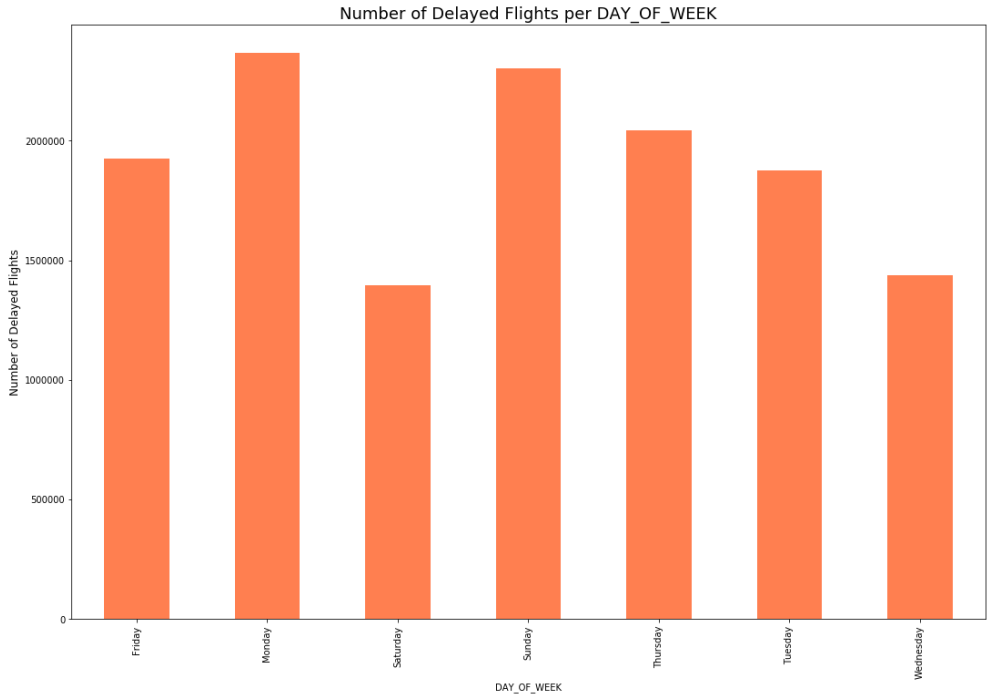
### 刪除缺失值

因所有缺失值為飛機飛行時間與航班延誤原因，為無法替補的資料，所以直接刪除有缺失值的資料。經過資料前處裡後的資料集有228528筆與17欄欄位。

# 資料視覺化



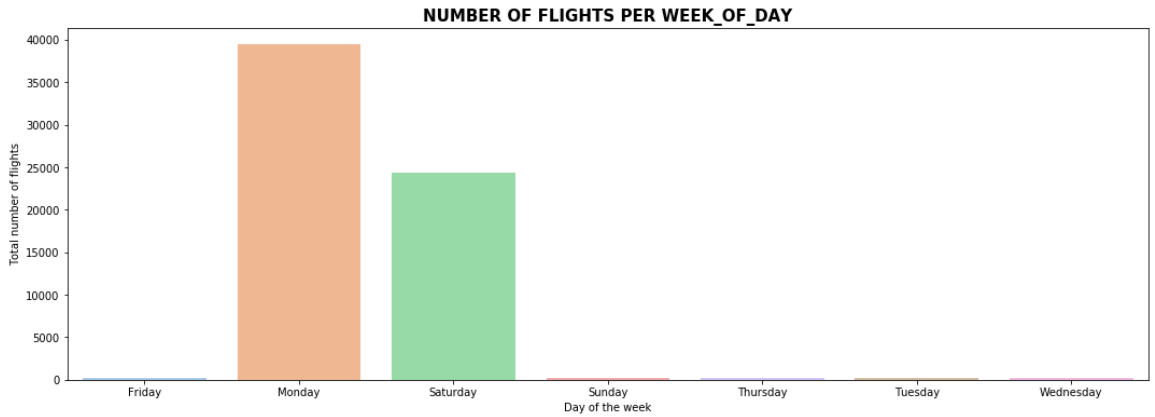

航班數量與每週天數圖



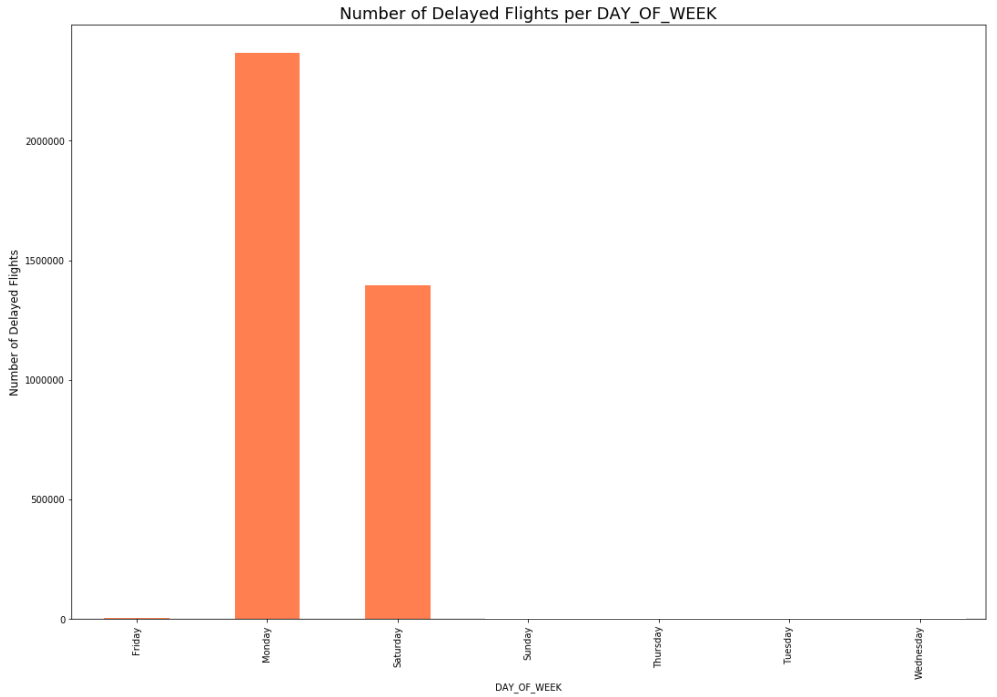

班機抵達延誤與每週天數

週一有較多的航班，周日的航班卻是最少。較多航班同樣也有較多的班機延誤數量，與一般認知相符。

# 資料視覺化



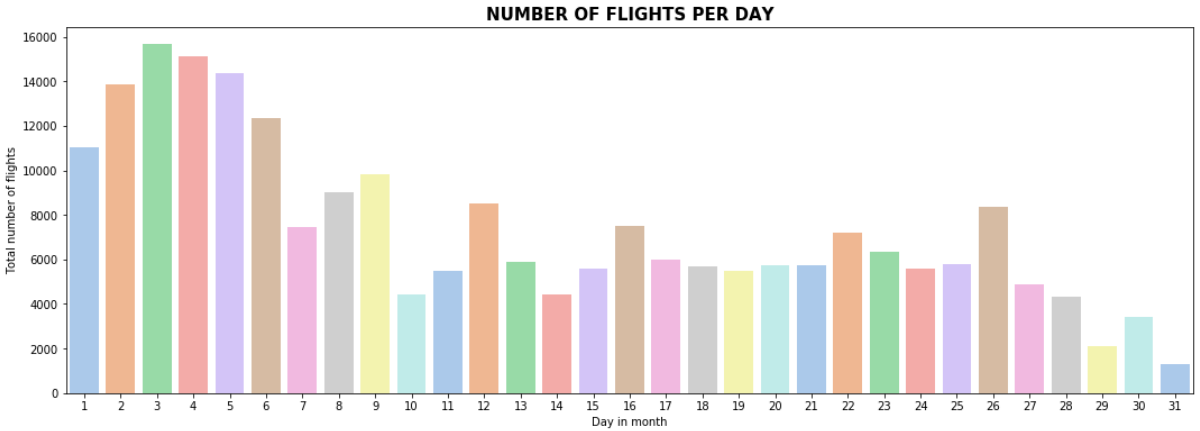

航班數量與每週天數圖



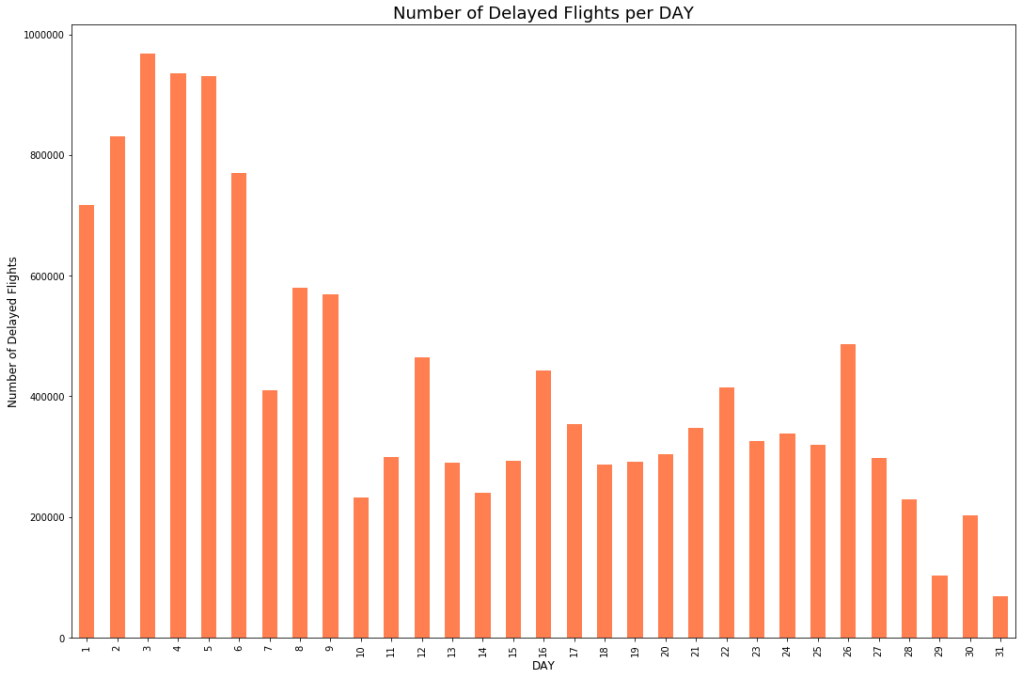

班機抵達延誤與每週天數

週一有較多的航班，周日的航班卻是最少。較多航班同樣也有較多的班機延誤數量，與一般認知相符。

# 資料視覺化



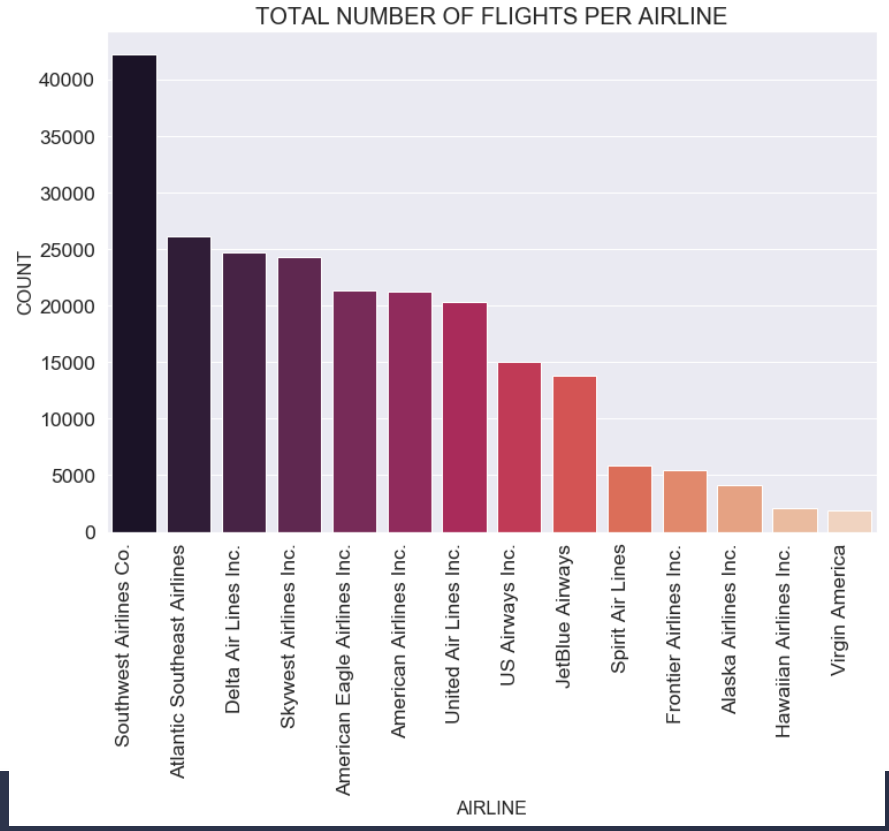

航班數量與每月天數

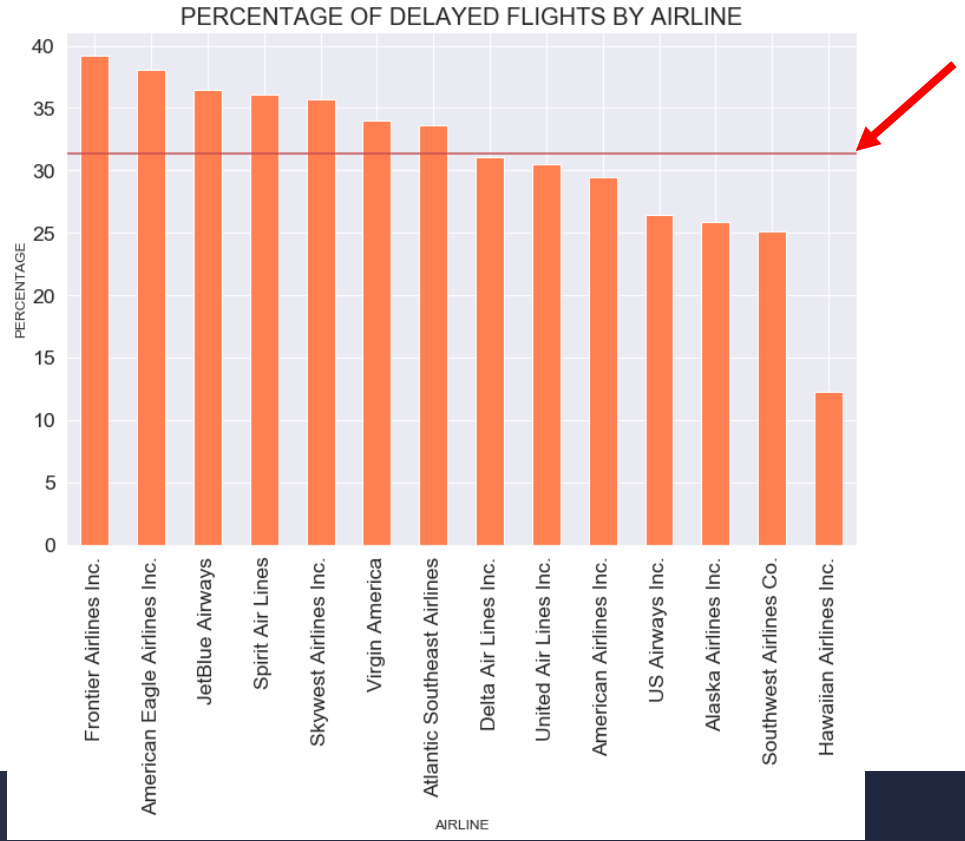



班機抵達延誤與每月天數

月初與月尾有較多的航班，而月初又比月尾航班數量多出許多，月中航班卻是最少。同樣較多航班的天數有較多的班機延誤數量。

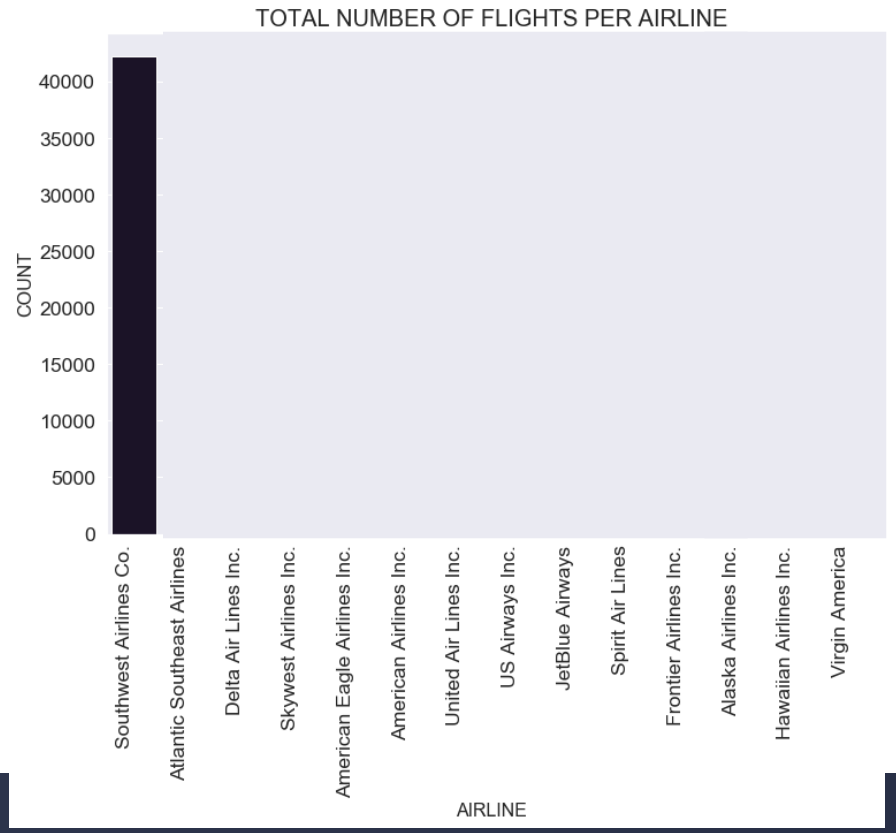
# 資料視覺化

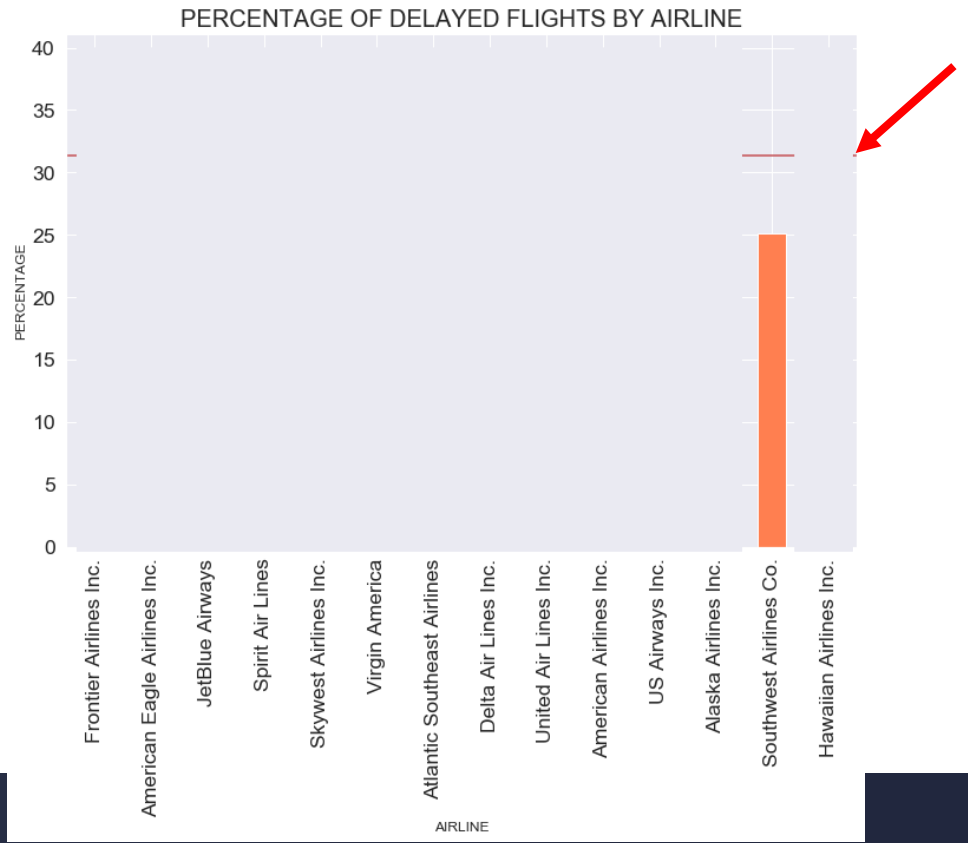



7家航空公司班機延誤數量高於平均。Southwest 航空公司雖有最多的航班其延誤航班數卻低於平均許多，相反Frontier 航空公司的航班是相對較少的，卻是航班延誤作多的公司之一。航點與航班數量資料所帶來的訊息相同，後續資料分析可以只取一個。

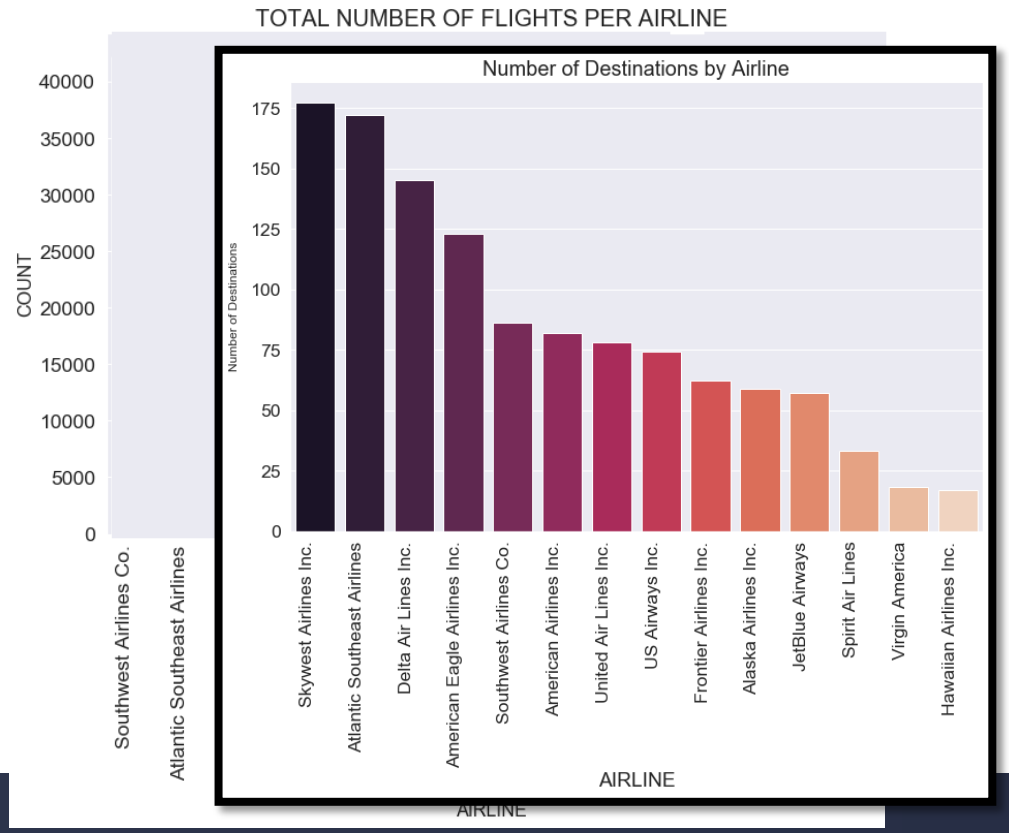
# 資料視覺化

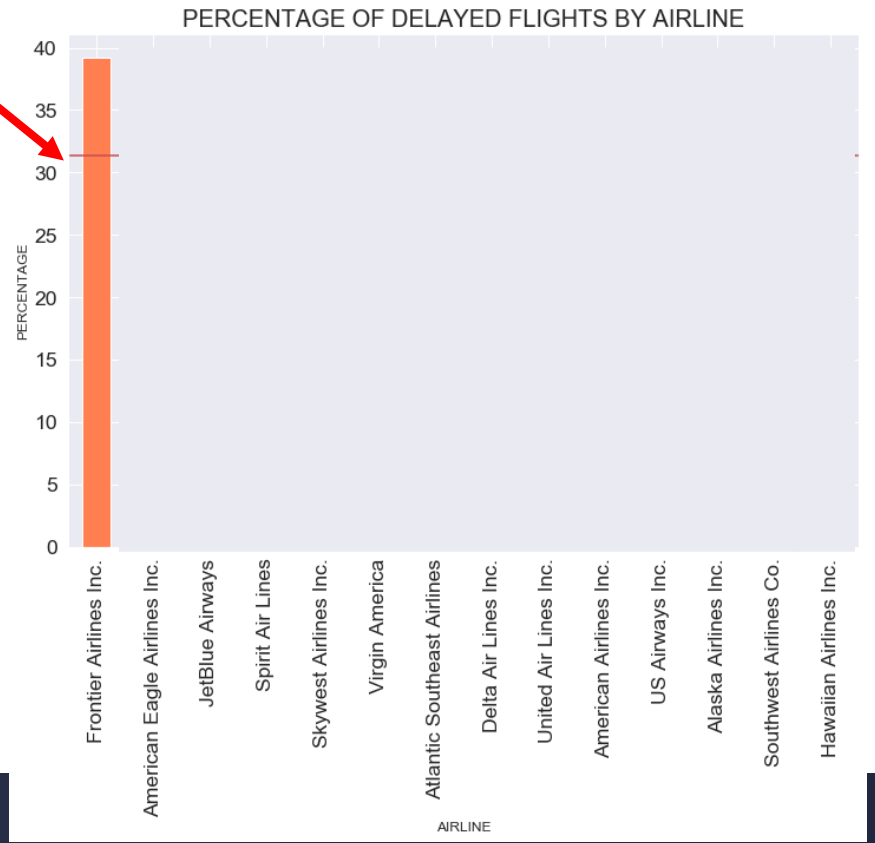



7家航空公司班機延誤數量高於平均。Southwest 航空公司雖有最多的航班其延誤航班數卻低於平均許多，相反Frontier 航空公司的航班是相對較少的，卻是航班延誤作多的公司之一。航點與航班數量資料所帶來的訊息相同，後續資料分析可以只取一個。

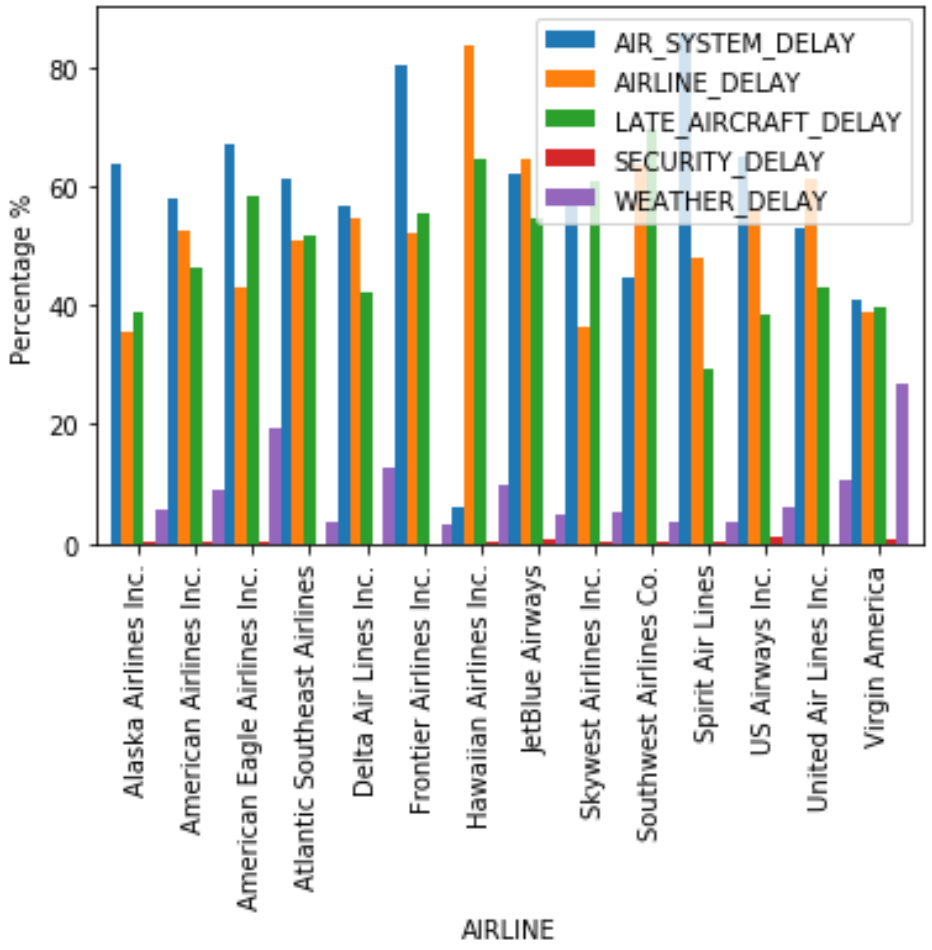
# 資料視覺化



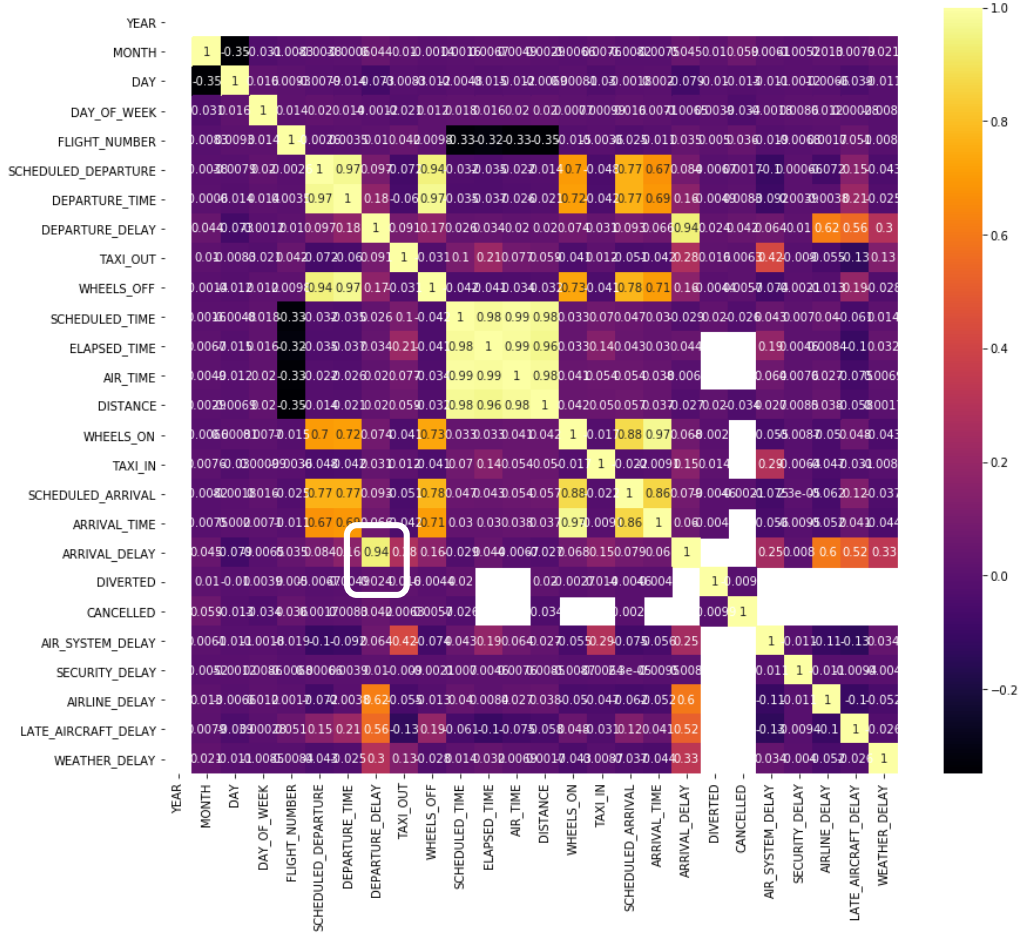




7家航空公司班機延誤數量高於平均。Southwest 航空公司雖有最多的航班其延誤航班數卻低於平均許多，相反Frontier 航空公司的航班是相對較少的，卻是航班延誤作多的公司之一。航點與航班數量資料所帶來的訊息相同，後續資料分析可以只取一個。

# 資料視覺化




各航空公司班機延誤原因所占百分比




特徵值間的關係圖

# 資料預處理

## 資料編碼
LabelEncoding

```
# Label encoding
# 資料編碼 -> label encoding: 把每個類別 mapping 到某個整數, 不增加新欄位
le = LabelEncoder()
df['AIRLINE']= le.fit_transform(df['AIRLINE'])
df['ORIGIN_AIRPORT'] = le.fit_transform(df['ORIGIN_AIRPORT'])
df['DESTINATION_AIRPORT'] = le.fit_transform(df['DESTINATION_AIRPORT'])
df = df.drop(columns=['DATE'])
#df['DATE'] = le.fit_transform(df['DATE'])
df['MONTH'] = le.fit_transform(df['MONTH'])
df['DAY_OF_WEEK'] = le.fit_transform(df['DAY_OF_WEEK'])
df['SCHEDULED_DEPARTURE'] = le.fit_transform(df['SCHEDULED_DEPARTURE'])
df['DEPARTURE_TIME'] = le.fit_transform(df['DEPARTURE_TIME'])
df['SCHEDULED_ARRIVAL'] = le.fit_transform(df['SCHEDULED_ARRIVAL'])
df['ARRIVAL_TIME'] = le.fit_transform(df['ARRIVAL_TIME'])
df['target'] = le.fit_transform(df['target'])
```

## 資料樣本區分
增加目標欄位(target)
train_test_split()

```
# Createing target column
df['target'] = df['ARRIVAL_DELAY'] > df['ARRIVAL_DELAY'].mean()
print('Percentage of delayed flights per airline:')
print(df['target'].value_counts(normalize=True))
print('False means arrive ontime. True means delayed.')
```


## 標準化
StandartScaler()

```
# Splitting y and x
# 因為 ARRIVAL_DELAY == target 且 大多DEPARTURE DELAY 就會有 ARRIVAL DELAY
# X 不取 DEPARTURE DELAY 與 ARRIVAL DELAY
X = df.drop(['target', 'ARRIVAL_DELAY', 'DEPARTURE_DELAY'],axis = 1)
y = df['target']

# Splitting into train and test data set
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state = 2)
# 標準化資料
sc1=StandardScaler()
X_train_sc=sc1.fit_transform(X_train)
X_test_sc=sc1.transform(X_test)
num_features = len(X_train.columns)
```

4

# 模型訓練與參數調整

PART FOUR

# 模型訓練

1 Liner Regression

```
Linear Regression
Mean Absolute Error: 0.26339660172746254
Mean Squared Error: 0.1144467636731898
Root Mean Squared Error: 0.3382998132916863
R2 :  0.46761802741449865
Accuracy:  0.46761802741449865
```

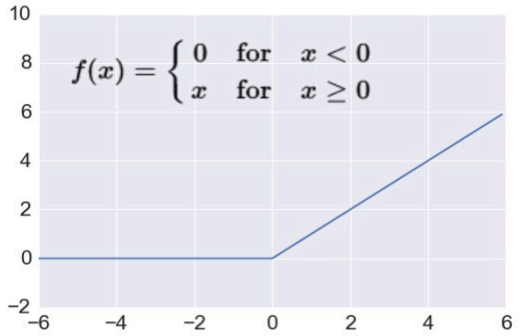
2 全連結層深度神經網絡

```
Accuracy: 68.72 %
Precision score: 0.0 %
Recall score: 0.0 %
F1 score: 0.0 %
```

# 模型參數

## 激活函數

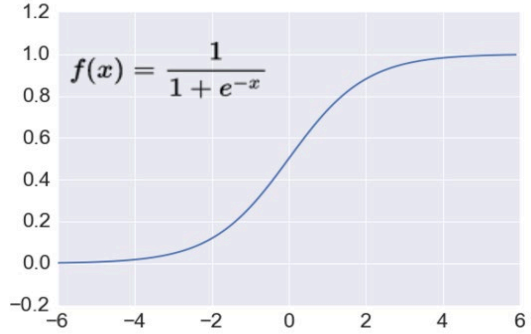
ReLU




輸出值介於 [0,∞] 之間

$$f(x) = \max(x, 0)$$

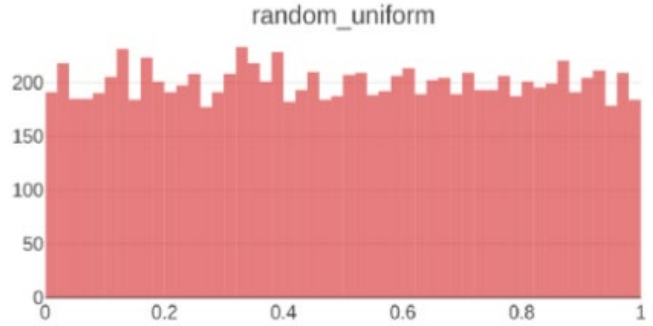
Sigmoid




輸出值介於 [0,1] 之間，且分布兩極化(1或0)，適合二分法

$$f(x) = \frac{1}{1+e^{-x}}$$

## 模型所用之初始化方法

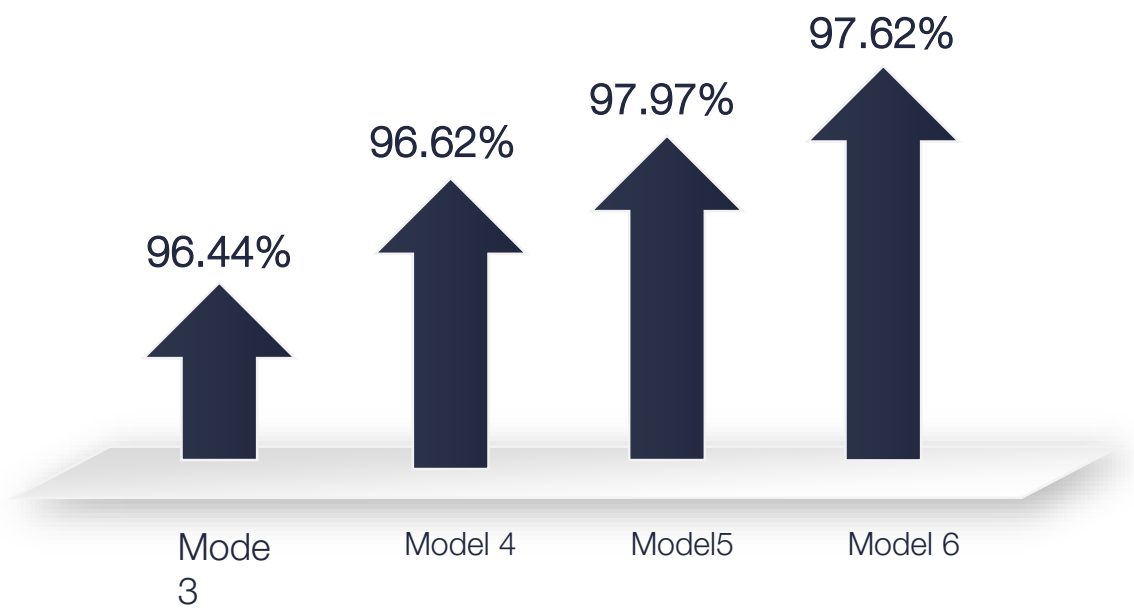



## 模型所用之正則化參數

$$R(w) = ||w||_1 = \sum_i |w_i|$$

$$R(w) = ||w||_2^2 = \sum |w_i^2|$$

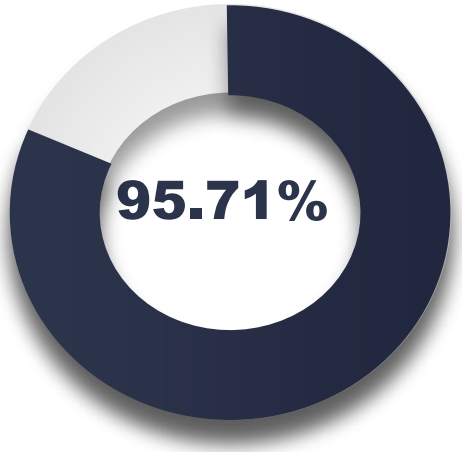
# 參數調整



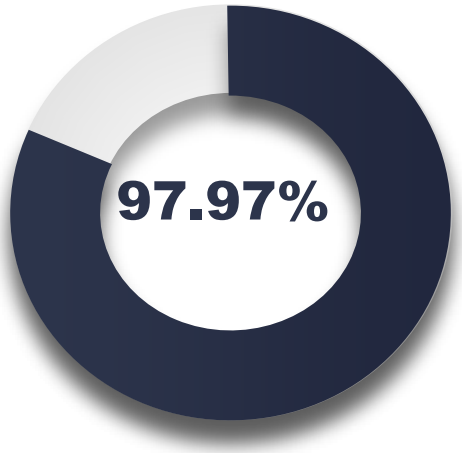

Top 4 model accuracy on testing set

| # | Activation function | epoch | Optimizer function | Loss function | Batch size | initializer and regularize | Max. performance |
|---|---|---|---|---|---|---|---|
| 1 | Relu and sigmoid | 10 | adam | mean_squared_error | 64 | No | 68.64 %<br>68.92 % |
| 2 | Relu and sigmoid | 16 | adam | binary_crossentropy | 64 | No | 31.36 %<br>31.08 % |
| 3 | Relu and sigmoid | 16 | adagrad | binary_crossentropy | 64 | No and l1 | 96.17 %<br>96.44 % |
| 4 | Relu and sigmoid | 16 | adagrad | binary_crossentropy | 64 | random_uniform<br>and l2 | 96.05 %<br>96.62 % |
| 5 | Relu and sigmoid | 16 | adam | mean_squared_error | 64 | random_uniform<br>and l2 | 95.71 %<br>97.97 % |
| 6 | Relu and sigmoid | 16 | adagrad | binary_crossentropy | 64 | random_uniform<br>and l2 | 96.66 %<br>97.62 % |

# 參數調整




On Training Set




On Testing Set

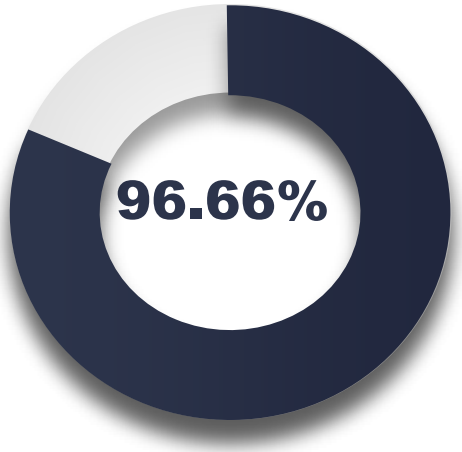
**94.99%**

```
Maximum accuracy in Training set: 95.71
epoch:  13
Maximum accuracy in Testing set: 97.97
epoch:  10
Minimum loss in Training set: 21.39
epoch:  0
Minimum loss in Testing set: 11.08
epoch:  0
```

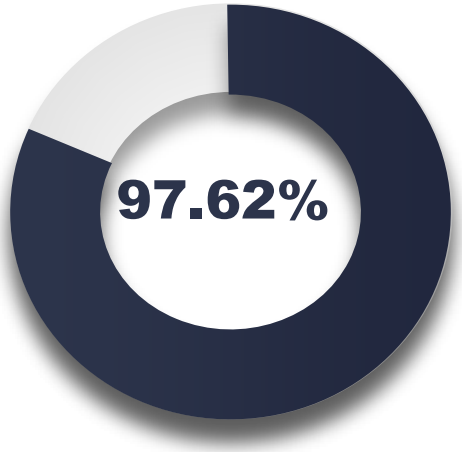
```
On Testing Set
Accuracy: 94.99 %
Precision score: 86.39 %
Recall score: 99.67 %
F1 score: 92.56 %
```

| # | Activation function | epoch | Optimizer function | Loss function | Batch size | initializer and regularize | Max. performance |
|---|---|---|---|---|---|---|---|
| 1 | Relu and sigmoid | 10 | adam | mean_squared_error | 64 | No | 68.64 %<br>68.92 % |
| 2 | Relu and sigmoid | 16 | adam | binary_crossentropy | 64 | No | 31.36 %<br>31.08 % |
| 3 | Relu and sigmoid | 16 | adagrad | binary_crossentropy | 64 | No and l1 | 96.17 %<br>96.44 % |
| 4 | Relu and sigmoid | 16 | adagrad | binary_crossentropy | 64 | random_uniform and l2 | 96.05 %<br>96.62 % |
| 5 | Relu and sigmoid | 16 | adam | mean_squared_error | 64 | random_uniform and l2 | 95.71 %<br>97.97 % |
| 6 | Relu and sigmoid | 16 | adagrad | binary_crossentropy | 64 | random_uniform and l2 | 96.66 %<br>97.62 % |

# 參數調整




On Training Set




On Testing Set

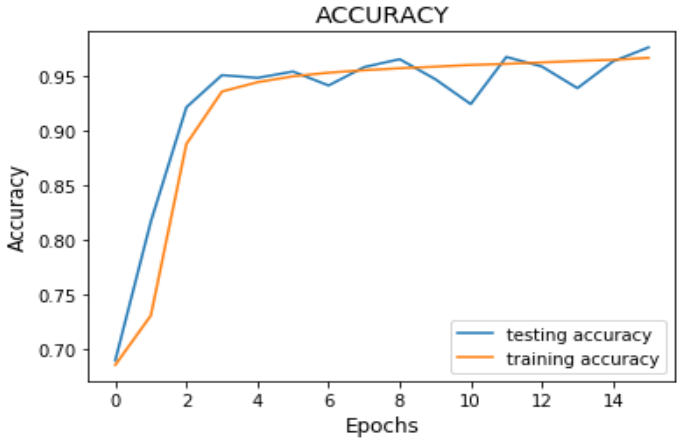
**97.53%**

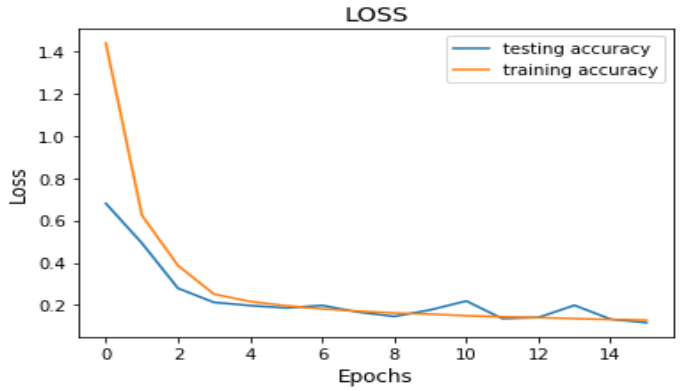
```
Maximum accuracy in Training set: 96.66
epoch:  15
Maximum accuracy in Testing set: 97.62
epoch:  15
Minimum loss in Training set: 12.89
epoch:  15
Minimum loss in Testing set: 11.67
epoch:  15
```

```
On Testing Set
Accuracy: 97.53 %
Precision score: 94.87 %
Recall score: 97.38 %
F1 score: 96.1 %
```

| # | Activation function | epoch | Optimizer function | Loss function | Batch size | initializer and regularize | Max. performance |
|---|---|---|---|---|---|---|---|
| 1 | Relu and sigmoid | 10 | adam | mean_squared_error | 64 | No | 68.64 %<br>68.92 % |
| 2 | Relu and sigmoid | 16 | adam | binary_crossentropy | 64 | No | 31.36 %<br>31.08 % |
| 3 | Relu and sigmoid | 16 | adagrad | binary_crossentropy | 64 | No and l1 | 96.17 %<br>96.44 % |
| 4 | Relu and sigmoid | 16 | adagrad | binary_crossentropy | 64 | random_uniform<br>and l2 | 96.05 %<br>96.62 % |
| 5 | Relu and sigmoid | 16 | adam | mean_squared_error | 64 | random_uniform<br>and l2 | 95.71 %<br>97.97 % |
| 6 | Relu and sigmoid | 16 | adagrad | binary_crossentropy | 64 | random_uniform<br>and l2 | 96.66 %<br>97.62 % |

# 最佳模型




## 混淆矩陣

```
[[45981  1130]
 [  563 20885]]
```

## 泛化程度

```
On Testing Set
Accuracy: 97.53 %
Precision score: 94.87 %
Recall score: 97.38 %
F1 score: 96.1 %
```

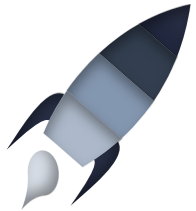
```
model_6 = Sequential()
model_6.add(Dense(256, activation='relu', input_shape=(num_features,),
                  kernel_initializer='random_uniform',
                  bias_initializer=initializers.Zeros(),
                  kernel_regularizer=regularizers.l2(0.01)))
model_6.add(Dense(128, activation='relu', input_shape=(num_features,),
                  kernel_initializer='random_uniform',
                  bias_initializer=initializers.Zeros(),
                  kernel_regularizer=regularizers.l2(0.01)))
model_6.add(Dense(128, activation='relu', input_shape=(num_features,),
                  kernel_initializer='random_uniform',
                  bias_initializer=initializers.Zeros(),
                  kernel_regularizer=regularizers.l2(0.01)))
model_6.add(Dense(1, activation='sigmoid',
                  kernel_initializer='random_uniform',
                  bias_initializer=initializers.Zeros(),
                  kernel_regularizer=regularizers.l2(0.01)))

model_6.compile(loss='binary_crossentropy', optimizer='adagrad', metrics=['accuracy'])
```

5

PART FIVE

# 結論

# 結論 - 未來展望



使用深度學習的全連結層神經網路建立航班抵達延誤預測模型，其模型預測結果具良好的效度。未來可以獲取更多月份或時間性資料進行分析，以取得根據連假出遊變動的航班需求資訊，更有效的預測結果；此外在模型建構上可再做更多參數的調適，提升模型效度。

激活函數
Relu
sigmoid

損失函數
binary_crossentropy
激活函數
adagrad

模型初始化
random
uniform
正則化參數
L2

3

PART THREE

# 參考資料

## 模型訓練參數調整參考


激活函數: https://ithelp.ithome.com.tw/articles/10191725

Kernel initializers: https://keras.io/api/layers/initializers/

https://becominghuman.ai/priming-neural-networks-with-an-appropriate-initializer-7b163990ead

Kernel regularizers: https://keras.io/api/layers/regularizers/

https://stats.stackexchange.com/questions/383310/what-is-the-difference-between-kernel-bias-and-activity-regulizers-and-when-t

https://www.itread01.com/content/1513589905.html


## 論文參考

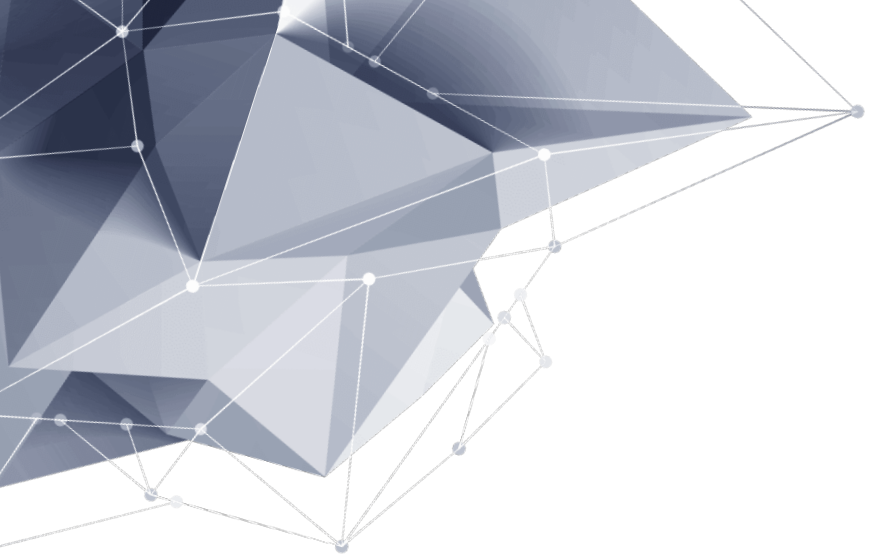

Chakrabarty, Navoneel. "A Data Mining Approach to Flight Arrival Delay Prediction for American Airlines." *2019 9th Annual Information Technology, Electromechanical Engineering and Microelectronics Conference (IEMECON)*. IEEE, 2019.

Kim, Young Jin, et al. "A deep learning approach to flight delay prediction."*2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*. IEEE, 2016.

張惟帆. "航班時間可靠度衡量指標與影響因素分析." *交通大學運輸與物流管理學系學位論文* 2016 年 (2016): 1-56.

何學庸, and 张鈥崴. "航班取消, 延誤及超額訂位致旅客無法登機賠償法規發展之研究-以欧盟, 英國, 美國及台灣航空運輸市場為例." *休閒與遊憩研究* 4.2 (2010): 73-100.

黄明光. "借鉴外国经验 降低民航班机延误及乘客闹事率." *漯河職業技術學院學報* 14.6 (2015): 118-119.



# THANK YOU

## Q & A