Name: Clare Kuan Sze Sing

Student number: 220456793

US flight pattern analysis

Programming for data science

Contents

[1.0 Finding best times to minimise delays 2](#_Toc162784358)

[1.0.1 Data wrangling and manipulation 2](#_Toc162784359)

[1.0.2 10 years data 2](#_Toc162784360)

[1.0.3 Extra data wrangling 3](#_Toc162784361)

[1.0 Finding best days of the week to minimise delays 3](#_Toc162784362)

[1.1.1 Data wrangling and manipulation and plotting 10 years data 3](#_Toc162784363)

[1.1.2 Extra data wrangling 3](#_Toc162784364)

[2. Evaluate whether older planes suffer more delays than younger delays 2.0.1 Data wrangling and manipulation 4](#_Toc162784365)

[2.0.2 Plotting 10 years data in 2 different ways 4](#_Toc162784366)

[2.0.3 Extra data wrangling 5](#_Toc162784367)

[3. Fitting a logistic regression model and visualising the coefficients 5](#_Toc162784368)

[3.0.1 Data wrangling and manipulation and plotting 5](#_Toc162784369)

[3.0.2 Extra data wrangling 7](#_Toc162784370)

[Appendix 8](#_Toc162784371)

# Finding best times to minimise delays[[1]](#footnote-1)

### 1.0.1 Data wrangling and manipulation

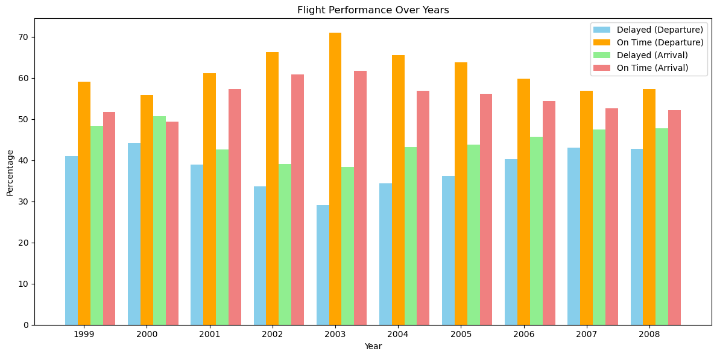
Step 1: We import relevant packages in both python and r for manipulation and plotting of the data, afterwards we read each of the 10 years csvs and group them into a single list that contains data across the 10 years

Step 2: In order to grasp a better understanding of flight patterns, we break down the list to only focus on delay timings which include departure delay (DepDelay) and arrival delay (ArrDelay). In the same process, we also filter out both diverted and cancelled flights to avoid any discrepancy or NaN values in data. Just a cautionary process, we also keep the Year column to ensure all years are in the dataframe

Step 3: Since we are looking to minimise delays, we ensure that the both delay variables are more than zero as values lesser or equivalent to zero are classified as ontime flights.

Step 4: To visualise how the 2 delay variables compare with each other, we calculate the percentage of flights that are delayed and the ontime flights percentage for each year via a loop function. Hence there will be 4 bar graphs per year; on time departure vs on time arrival and delayed departure vs delayed arrival as seen in the Jupyter notebook and R markdown.

Python’s visualisation



The graph shows that 1. There’s a higher % of ontime departure and a lower % of delayed departure flights. Hence, to find best time we will be looking specifically in the arrival related columns

### 1.0.2 10 years data

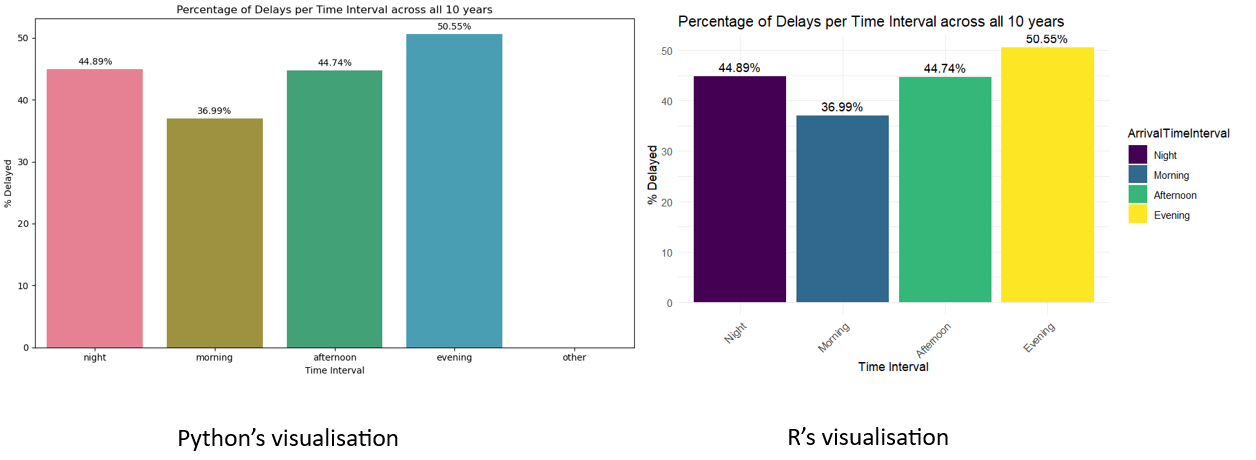
For finding best times, we repeat step 2 but instead of focusing on departure delay and arrival delay, we instead narrow down the list to focus on arrival delay and scheduled arrival time (CRSArrTime).

Step 2.1: After filtering, we combine the list into a singular dataframe.

Step 5: We categorise the scheduled timings into different parts of the day using a 24hr format as inputted in the dataframe[[2]](#footnote-2). This is done via making a categorise\_time function to separate and categorise the timings into the newly created column ArrivalTimeInterval. This step also combines all the years’ scheduled timings into their relevant interval hence, there will be no specific year plotting for the result

Step 6: Repeat Step 3 and 4 but remove calculation for percentage of ontime flights

Step 7: We plot the data into a graph for visualisation



Both graphs shows that the best time for flights to schedule an arrival is during the mornings. The mornings experience the lowest percent of delayed flights with evenings experiencing the highest percent of delayed flights.

1.0.3 Extra data wrangling

For individual years we do further manipulate and clean out the data by extracting anomalies. This is done by setting upper and lower limit that does not remove any potential delays that are low but more than zero. This is also done to clean up the graph as for individual years we are plotting a line plot of CRSArrTime against the average ArrDelay. In addition, each year’s line plot will have a red dot as a marker to indicate where the lowest delay occurs at first glance. Further information can be found in both Jupyter notebook and R markdown labelled as ST2195 Coursework part 2 a). Labels and names may differ between the 2 programs but the structure is the same.

# Finding best days of the week to minimise delays[[3]](#footnote-3)

### 1.1.1 Data wrangling and manipulation and plotting 10 years data

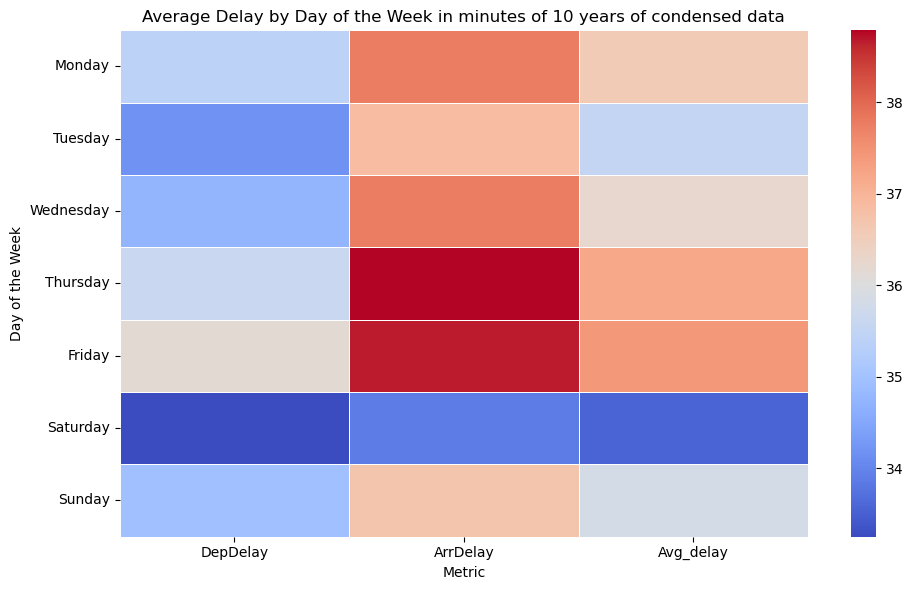
Firstly, we repeat Step 1 – 2.1 but, in step 2 we add another column called DayOfWeek as we are focusing on finding days with lowest delays

For finding best days, we need not only investigate ArrDelays like best times. Hence in this section we would also include DepDelay and use these 2 delays for further manipulation / creation of column(s)

Step 8: We map out the day numbers to which day of the week they represent as in the dataframe the DayOfWeek column is inputted by numbers instead of names. Then we can effectively convert the DayOfWeek column to have the day names. We proceed to repeat Step 3.

Step 9: Since we have both DepDelay and ArrDelay, this step involves finding the mean DepDelay and ArrDelay for each day first then we proceed to add the mean of both delays up and divide by 2 to find the overall average delay per day.

Lastly, we repeat Step 7 but with different coding and we plot out DepDelay, ArrDelay and Avg\_Delay all in 1 diagram



We can observe that Saturday's DepDelay experiences the lowest overall delay while Friday's ArrDelay experiences the highest overall delay

In DepDelay, Saturday experiences the lowest delay while Friday experiences the highest

in ArrDelay, Saturday experiences the lowest delay while Friday experiences the highest

in Avg\_delay, Saturday experiences the lowest delay while Friday experiences the highest

Overall, it’s clear that Saturday experiences the lowest delays while Friday experiences the highest delays

Python’s visualisation

### 1.1.2 Extra data wrangling

For individual years, for the diagram, we will not code a heatmap to plot out DepDelay and ArrDelay. Instead, we will only be plotting the Avg\_delay in the form of a bar graph both in R and python. Labels and names may differ between the 2 programs but the structure is the same.

2. Evaluate whether older planes suffer more delays than younger delays[[4]](#footnote-4)  
2.0.1 Data wrangling and manipulation

Firstly, we repeat Step 1 and in addition read the plane-data.csv. Afterwards, we repeat Step 2 but only filtering out the diverted and cancelled flights in the year csv.

Before we can proceed to repeat Step 2.1, we insert a step where after reading the plane-data.csv we must filter out any invalid values in their year column. Next, we convert the years in year column into a year-month-date format for easier checking after Step 2.1. Lastly, before Step 2.1, we rename the talinum column in the plane data to TailNum to match the year csv’s column. Now since both TailNum columns are similarly capitalised, we can merge the plane with our 10-year data list based on their shared column TailNum and then finally repeat Step 2.1.

We must repeat Step 1, 2 and 2.1 as this section of coding will be in a separate Jupyter notebook and R markdown.

Step 10: To find the age of the planes we will use the most recent year in our dataframe (2008) for subtraction in the year column (year is in the plane data, while Year is found in the eg 1999.csv). Before subtraction, we remove any NaN values as a precaution and then convert the data in the year column into integers (4-digit year). Then we can subtract.

Step 11: We want to categorise the planes into older and younger. To do so, we have to set a threshold age (20 years) and any plane older than 20 years will be grouped into older planes. [[5]](#footnote-5)

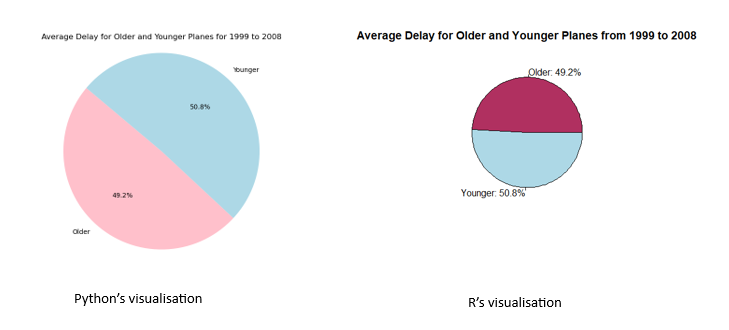
### 2.0.2 Plotting 10 years data in 2 different ways

Step 12: We create a new column called Delay that is the addition of DepDelay and ArrDelay

Step 13: From the singular dataframe, we separate into 2 sub dataframes one for older and one for younger planes

Step 14: We remove any NaN values in the Delay column for both older and younger planes’ own dataframe. Afterwards, we proceed to find the mean of the Delay in both dataframes.

Lastly, we repeat Step 7 but with different coding



We can observe there’s generally little to no difference between younger and older planes

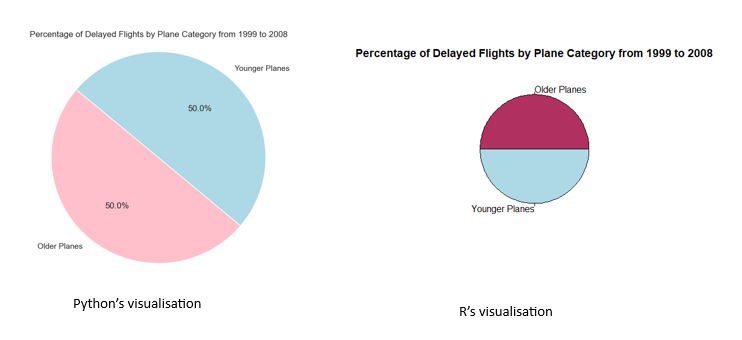
But to be more specific, younger planes suffer from more average delays by about 1 %

Step 12- 14 is looking into average total delays without removing ontime or earlier flights. Hence the 2nd part below will be looking specifically into delayed flights to give a more focused look into older and younger planes

Before we move on to Step 15, we must repeat the Steps mentioned in 2.0.1 and Steps 12-14

Step 15: In each of the younger and older dataframe, we filter to keep Delay values that are more than zero. (not filtering DepDelay or ArrDelay itself more than zero but its summation)

Step 16: We repeat Step 4 but instead having an ontime flight percentage its only delayed flights percentage

Lastly, we repeat Step 7 but with different coding 

We can observe there’s no difference between younger and older planes

Hence older planes do not suffer more delays than younger delays

### 2.0.3 Extra data wrangling

For individual years, the plane age calculation and categorisation differ from the 10 years calculation. For each year, to calculate the plane age, we take the year we are specifically look at in that section and subtract it from the year column. So, let’s say we are looking at only 2002, the plane age for the 2002 coding section will be 2002 minus year column in plane data. Furthermore, since the plane age will always differ between years by using such subtraction method, the threshold\_age also differs. Hence for every year we will print out the plane age to determine a good threshold for classification. Further information about the different threshold ages can be found in both Jupyter notebook and R markdown labelled as ST2195 Coursework part 2 b). Lastly, another modification is the graph, where instead of a pie chart it is a bar graph where the percentage values are shown at the top of each bar. Labels and names may differ between the 2 programs but the structure is the same.

# 3. Fitting a logistic regression model and visualising the coefficients[[6]](#footnote-6)

### 3.0.1 Data wrangling and manipulation and plotting

Since this will be greatly different from the other 2 parts, we will restart the Step numbers.

Step 1 to 14 will be plotting logistic regression model and finding coefficients for numerical features

Step 1: Import relevant packages in python and r then read the 1999 to 2003 data csv, plane-data csv and airports.csv

Step 2: Combine the 1999 to 2003 read csv into 1 list and then merge it into a dataframe

Step 3: We merge airports.csv and plane-data.csv into the 5-year dataframe. We merge the Origin, Dest column in airports with the 5-year dataframe via the common column iata. The last merge will be the UniqueCarrier column in plane with the airport and 5-year dataframe using the common code column.

Step 4: We filter out cancelled flights from the fully merged dataframe

Step 5: To get the coordinate feature, we calculate the midpoint latitude and midpoint longitude. Each midpoint is obtained by adding their respective origin and dest latitude and longitude and dividing it by 2

Step 6: We create 2 new columns in the fully merged dataframe to store each midpoint as latitude and longitude. Since the machine learning pipeline is unable to process coordinates format, the latitude and longitude will be the coordinates representation but separated.

Step 7: We separate the features into numerical and categorical. This step will be putting the numerical features into the machine learning pipeline

Step 7.1: We define what our features are; CRSDepTime, CRSArrTime, Distance, latitude and longitude. Then we define that x = features and y = whether the flights diverted or not

Step 7.2: Next, we will preprocess the data before feeding it into the predictive / machine learning model. In this preprocessing stage, we create numerical transformers that fill in missing values and standardises the numerical features

Step 7.3: We split the dataset into training and testing sets and allocate 50% of the data for testing. We also set up a random\_state to consistently generate the same number of random numbers.

Step 8: We set up the parameters for tuning in the logistic regression model

Step 9: We then calculate the predicted probabilities of the test sets and computes the true positive and false positive rates. At the same time, we also calculate the auc score to determine how good the model is at distinguishing between positive classes and negative classes. Finally, we can then plot the logistic regression model for numerical features

Step 10: Now, we proceed to find the coefficients of the numerical features.

Step 11: We fit a grid search to find the best logistic regression model by cross validating the training set

Step 12: After getting the best model, we can retrieve the coefficient corresponding to each feature implemented into the model

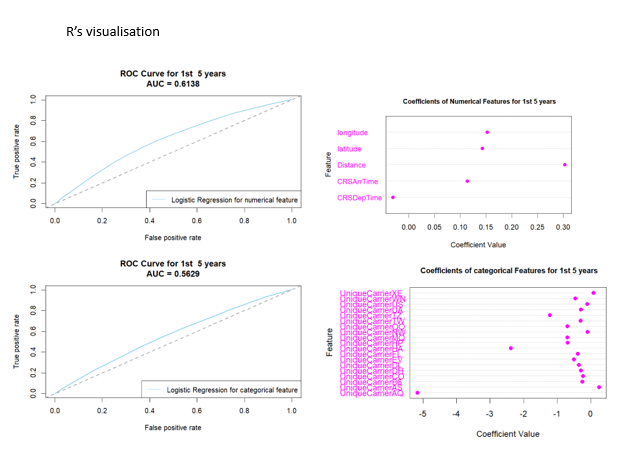
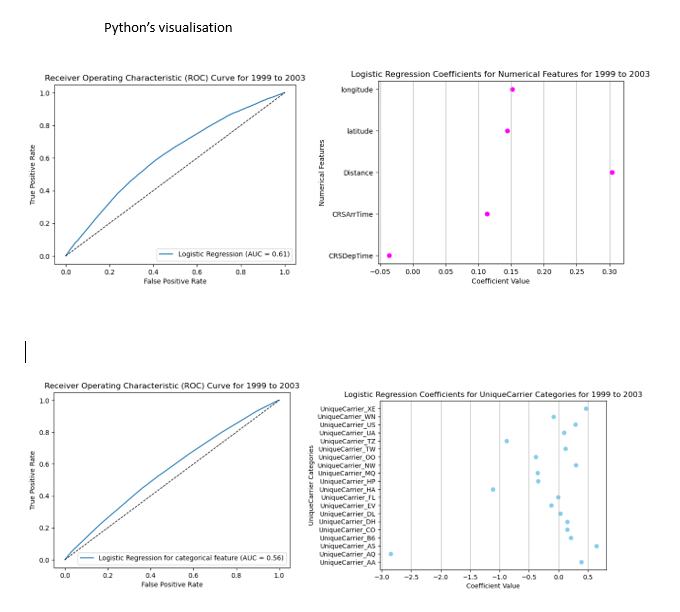
Step 13: We align / map each feature to their respective coefficients

Step 14: Plot the numerical feature coefficients out

Step 15: We use the fully merged dataframe in Step 4 and do not follow through Step 5 and 6

Step 16: We repeat step 7 to 14 but our feature is now only UniqueCarrier and the numerical transformer will be renamed to categorical transformer and the categorical feature will be converted into binary variables by the OneHotEncoder. At the end both graphs will be visualising the categorical features

Step 17: For the 2nd 5 years; 2004 to 2008 repeat Step 1 to 16 but do take note that in Step 1 and 2 we are reading year csv of 2004 to 2008 and combing these 5 years into another list for merging



We can observe both python and r the auc score is the same of 0.61 for numerical and 0.56 for categorical.

The auc score shows that the numerical logistic regression is better at distinguishing positive and negative classes than categorical

For the numerical coefficients, we can see that its between R and python it is generally the same where only CRSDepTime has a negative coefficient while the rest are positive This shows that overall numerical features have a positive relationship with the predicting model. When the numerical values increase, its likelihood of them being classified into positive classes increases as well

However, for the categorical coefficients, it greatly differs between R and python. Even though its distribution is generally the same, its actual value is not. Categorical coefficients in python mostly have a positive relationship with the predicting model while categorical coefficients in R have mostly an inverse relationship with the predicting model. This is where an increase in value causes the higher likelihood of the features to be classified in the negative class.

Even so, there is still some similarities like UniqueCarrier\_AQ in both python and R indicates lowest coefficient (most negative) and UniqueCarrier\_AS shows highest coefficient (most positive)

### 3.0.2 Extra data wrangling

In R and only in R, there will be extra lines of codes to remove columns that are not part of the pipeline process for both numerical and categorical features across all individual years and the grouped years. In addition, for the 2nd 5 years (2004 to 2008), around Step 9, an error would pop out but right below there are added codes to help investigate and deal with the error to continue plotting the logistic regression model. This is the same for the individual year of 2004. The error removal codes only apply to R and not Python. There will be no modifications in plotting for individual years. Labels and names may differ between the 2 programs but the structure is the same.

# Appendix

1. (htt) Parts of the day. Available at : <https://www.britannica.com/dictionary/eb/qa/parts-of-the-day-early-morning-late-morning-etc>

2. (jets, n.d.) FAQ: Is the age of an aircraft a safety factor? Available at: <https://www.paramountbusinessjets.com/faq/age-of-aircraft-safety-factor#:~:text=An%20aircraft's%20age%20is%20based,Standard%20aircraft%20%3D%2010%2D20%20years>

1. Found in Jupyter notebook and R markdown labelled as ST2195 Coursework part 2 a) [↑](#footnote-ref-1)
2. https://www.britannica.com/dictionary/eb/qa/parts-of-the-day-early-morning-late-morning-etc [↑](#footnote-ref-2)
3. Done in Jupyter notebook and R markdown labelled as ST2195 Coursework part 2 a) [↑](#footnote-ref-3)
4. Done in Jupyter notebook and R markdown labelled as ST2195 Coursework part 2 b) [↑](#footnote-ref-4)
5. https://www.paramountbusinessjets.com/faq/age-of-aircraft-safety-factor#:~:text=An%20aircraft's%20age%20is%20based,Standard%20aircraft%20%3D%2010%2D20%20years [↑](#footnote-ref-5)
6. Done in Jupyter notebook and R markdown labelled as ST2195 Coursework part 2 c) [↑](#footnote-ref-6)