UNIVERSITY OF LONDON

THE LONDON SCHOOL OF ECONOMICS AND POLITICAL SCIENCE

# FLIGHT DATA ANALYSIS

CREATED BY: BRENDON TREVIN THAPARARATNAM

PAGE COUNT: 10 PAGES

*(EXCLUDING COVER PAGE, TABLE OF CONTENTS AND APPENDIX)*

# Contents

## Introduction

The following report is a detailed analysis conducted on a subset of two datasets from 2006 and 2007 containing raw data regarding all commercial flights from major carriers that occurred between the above time frame within the USA. In addition to the main flight records, supplementary datasets have been used which contain information regarding airplanes and airports. All questions have been reproduced in Python and R and the outputs from both are included in this report. Prior to commencing the analysis a general cleaning was conducted on the main dataset and this was used as the starting point for all questions. The analysis has a centred focus on the delays in flights and evaluates the factors causing delays. A few areas of focus include the best time of year, week and day to fly minimizing delays, the relationship of plane manufacture year and delays, the change in number of people flying between states within the year, the cascading effect and downstream delay propagation and finally building a model which predicts delays.

Please refer appendix for glossary of terminology and variable descriptions.

## Cleaning the dataset

Firstly, the required packages were loaded (numpy and pandas on python and dplyr and rmarkdown on R) and the 2006 and 2007 flight datasets were imported. After conducting a check on whether the datasets have an equal number of columns they were both merged to form a single dataset which can then be used to analyze more effectively. The dimensions and column names of the newly created full dataset were then retrieved to identify whether the concatenation was successful and to identify if any irregular columns were introduced or were already present. As the full dataset had an equal number of columns (29 columns) to that of the 2006 and 2007 datasets it was confirmed that the concatenation was indeed successful.

The full dataset was then checked for missing values (NULL values). It was found that the DepTime, ArrTime, ActualElapsedTime, CRSElapsedTime, AirTime, ArrDelay, DepDelay and CancellationCode columns contained missing values. Since the Arrival Delay (ArrDelay) and Departure Delay (DepDelay) were the main focus of our analysis their null values were selected for removal. This removal was justified as the percentage of missing values in the ArrDelay and Depdelay column compared to the entire dataset was only 2.16% and 1.94% respectively, which is fairly neglible and hence will not impact our results immensely. Thus, the rows containing missing values in the ArrDelay column were removed from the full dataset. After this removal all the missing values in the other columns tallied upto zero except for the CancellationCode column which presented 14279089 null values. It was found that the Cancelled column was entirely comprised of 0 values indicating that no flights were cancelled, but the cancellation code column had a single entry denoting 'B', indicating that a flight was cancelled due to bad weather. This was contradictory, hence due to this and the fact that the CancellationCode will be insignificant in our analysis, the entire CancellationCode column was removed from the full dataset. The dataset is now entirely free from missing values.

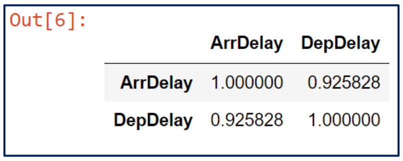
The full dataset was also checked for duplicate rows and 33 duplicates were identified and removed. Finally the types of variables were checked to make sure no irregularities were present. Upon confirming this the thoroughly cleaned dataset was saved and was made use of to answer the given questions.

Packages/libraries which were used in previous questions were carried forward without reinstalling in each scipt.

## QUESTION 1 - When is the best time of day, day of the week, and time of year to fly to minimise delays?

### *Assumptions*

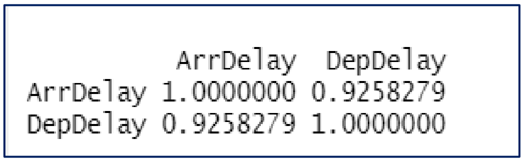
From the dataset there are a few attributes such as ArrDelay and DepDelay which can be used classify a delay. However, for this question we will be considering the Arrival Delay (ArrDelay) as the main indicator of a delay. One reason for this is because even though a flight departs late it can still arrive at the destination airport on time. Hence, from a passenger's point of view they will not have succumbed to any form of delay. Apart from that, as seen in figure-1 the correlation between ArrDelay and DepDelay is approximately 0.926, which is quite high, indicating that using either of them will not produce significantly different results. Therefore, using the ArrDelay (which is in fact the ArrTime minus the CRSArrTime) to estimate the best time to fly is justified to be logical. Furthermore we compare both the mean and median ArrDelays together in all cases as the latter is a good statistical indicator which is not affected by outliers, thus providing a wholistic analysis.



Out[6]:

| | ArrDelay | DepDelay |
|---|---|---|
| **ArrDelay** | 1.000000 | 0.925828 |
| **DepDelay** | 0.925828 | 1.000000 |

*Figure 1 - Python*

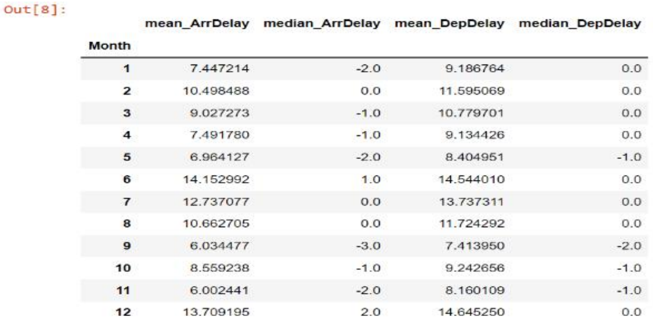


```
         ArrDelay  DepDelay
ArrDelay 1.0000000 0.9258279
DepDelay 0.9258279 1.0000000
```

*Figure 1 – R*

## Best time of year to fly

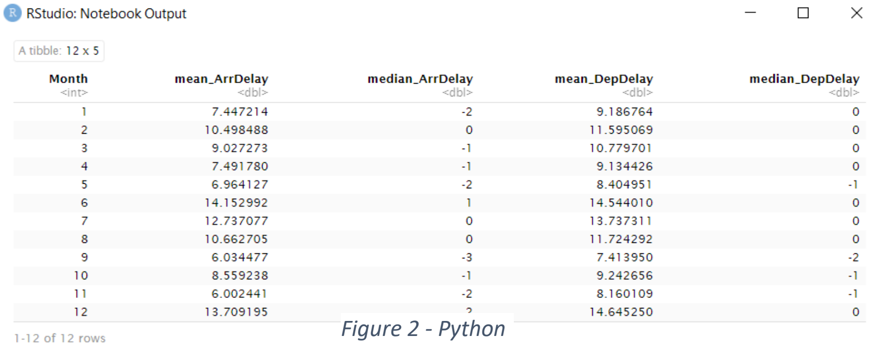
The required python packages (NumPy, pandas, seaborn, matplotlib) and R packages (rmarkdown, dplyr, plotly, ggplot2, tidyverse, tidyr) were imported along with the full dataset. The unnamed/X column was removed as it provided no benefit to the analysis. The correlation matrix was then plotted for ArrDelay and DepDelay to justify the use of ArrDelay. First, we checked the minimum, maximum and list of values from the Month column to ensure that there are no outliers or non-integer values. Then a table was constructed showing the mean and median ArrDelay and DepDelay which was grouped by each month (figure-2). To visualize this, a line graph was also constructed showing the average ArrDelay for each month. In order to make comparisons easier, the overall average delay line was also drawn for reference purposes (figure-3). Moreover, a similar average line graph was plotted to showcase how much each of the causal factors: CarrierDelay, WeatherDelay, NASDelay, SecurityDelay and LateAircraftDelay affect the ArrDelay (figure-4).



Out[8]:

| Month | mean_ArrDelay | median_ArrDelay | mean_DepDelay | median_DepDelay |
|---|---|---|---|---|
| 1 | 7.447214 | -2.0 | 9.186764 | 0.0 |
| 2 | 10.498488 | 0.0 | 11.595069 | 0.0 |
| 3 | 9.027273 | -1.0 | 10.779701 | 0.0 |
| 4 | 7.491780 | -1.0 | 9.134426 | 0.0 |
| 5 | 6.964127 | -2.0 | 8.404951 | -1.0 |
| 6 | 14.152992 | 1.0 | 14.544010 | 0.0 |
| 7 | 12.737077 | 0.0 | 13.737311 | 0.0 |
| 8 | 10.662705 | 0.0 | 11.724292 | 0.0 |
| 9 | 6.034477 | -3.0 | 7.413950 | -2.0 |
| 10 | 8.559238 | -1.0 | 9.242656 | -1.0 |
| 11 | 6.002441 | -2.0 | 8.160109 | -1.0 |
| 12 | 13.709195 | 2.0 | 14.645250 | 0.0 |

Figure 2 - R



RStudio: Notebook Output

A tibble: 12 x 5

| Month <int> | mean_ArrDelay <dbl> | median_ArrDelay <dbl> | mean_DepDelay <dbl> | median_DepDelay <dbl> |
|---|---|---|---|---|
| 1 | 7.447214 | -2 | 9.186764 | 0 |
| 2 | 10.498488 | 0 | 11.595069 | 0 |
| 3 | 9.027273 | -1 | 10.779701 | 0 |
| 4 | 7.491780 | -1 | 9.134426 | 0 |
| 5 | 6.964127 | -2 | 8.404951 | -1 |
| 6 | 14.152992 | 1 | 14.544010 | 0 |
| 7 | 12.737077 | 0 | 13.737311 | 0 |
| 8 | 10.662705 | 0 | 11.724292 | 0 |
| 9 | 6.034477 | -3 | 7.413950 | -2 |
| 10 | 8.559238 | -1 | 9.242656 | -1 |
| 11 | 6.002441 | -2 | 8.160109 | -1 |
| 12 | 13.709195 | 2 | 14.645250 | 0 |

1-12 of 12 rows

Figure 2 - Python

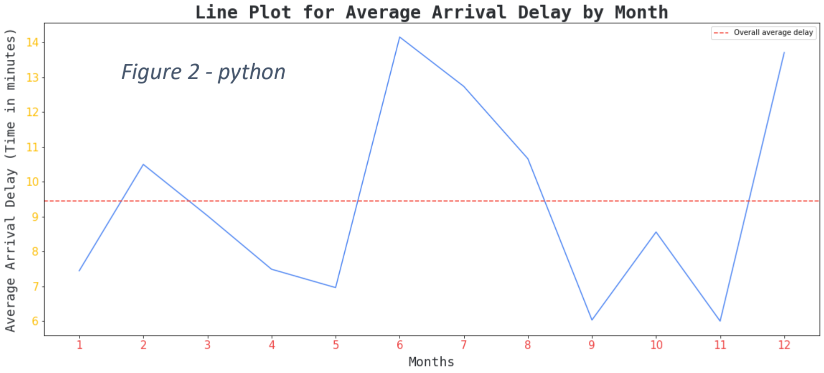


Figure 2 - python

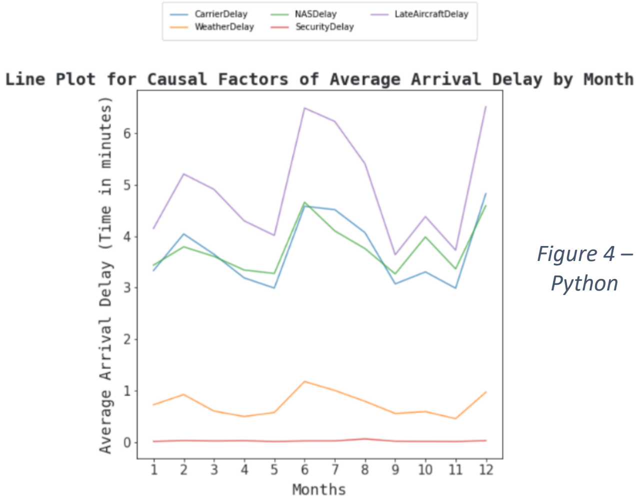


Figure 4 – Python

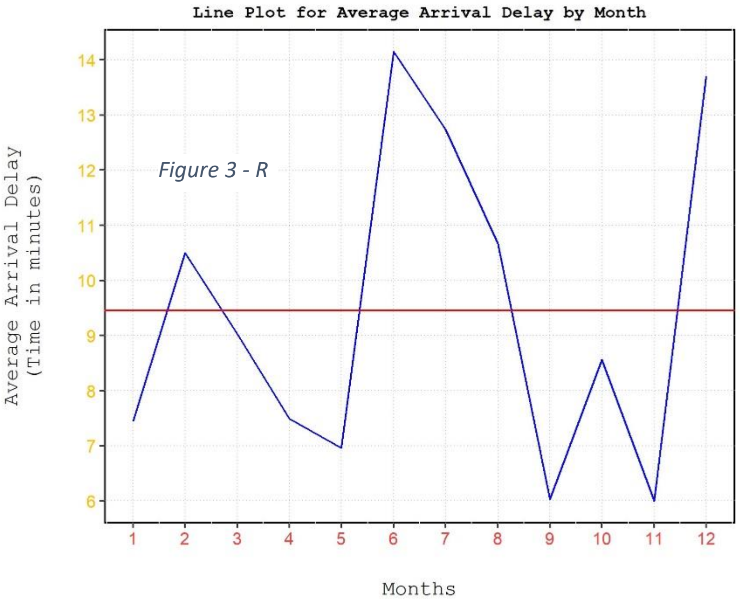


Figure 3 - R

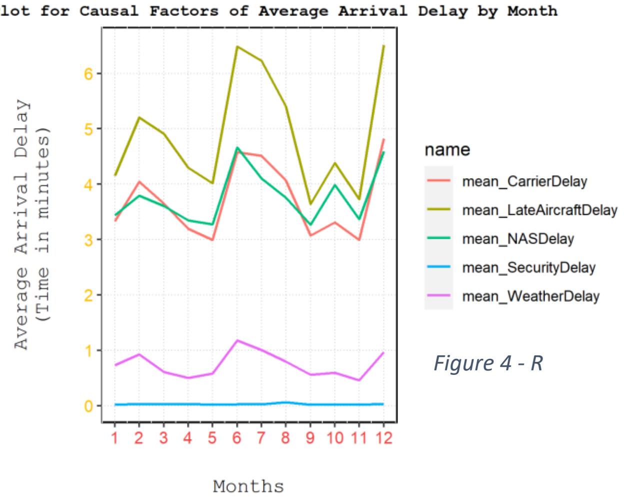
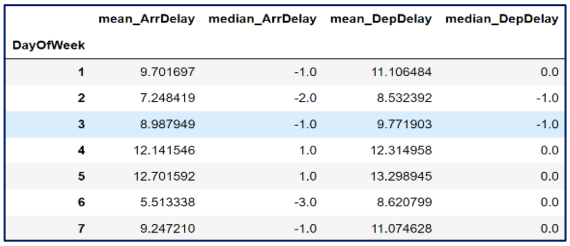


Figure 4 - R

According to figure-3 it can be seen that the months of September and November showed the lowest average ArrDelays with roughly 6 minutes each which is well below the overall average of 9.5 minutes. However, from figure-2 it can be seen that September has a lower median ArrDelay of -3 minutes compared to November with -2 minutes. Therefore, September would be the best time of year to fly to minimize delays. Moreover, the causal factors which affect ArrDelay the most would be LateAircraftDelay, NASDelay and CarrierDelay (figure-4). It is visible that these 3 delays also somewhat follow a similar trend as the average ArrDelay. Hence opting for September would also result in the minimal effect of these 3 delays on the passenger's flight.

## *Best time of week to fly*

After checking the minimum, maximum and list of values from the DayOfWeek column to ensure that there are no outliers or non-integer values, a table was constructed showing the mean and median ArrDelay and DepDelay which was grouped by each day of the week (figure-5). To visualize this, a line graph was constructed showing the average ArrDelay for each day of the week. In order to make comparisons easier, the overall average delay line was also plotted for reference purposes (figure-6). Moreover, a similar average line graph was made to showcase how much each of the causal factors: CarrierDelay, WeatherDelay, NASDelay, SecurityDelay, LateAircraftDelay affect the ArrDelay (figure-7)



| | mean_ArrDelay | median_ArrDelay | mean_DepDelay | median_DepDelay |
|---|---|---|---|---|
| **DayOfWeek** | | | | |
| **1** | 9.701697 | -1.0 | 11.106484 | 0.0 |
| **2** | 7.248419 | -2.0 | 8.532392 | -1.0 |
| **3** | 8.987949 | -1.0 | 9.771903 | -1.0 |
| **4** | 12.141546 | 1.0 | 12.314958 | 0.0 |
| **5** | 12.701592 | 1.0 | 13.298945 | 0.0 |
| **6** | 5.513338 | -3.0 | 8.620799 | 0.0 |
| **7** | 9.247210 | -1.0 | 11.074628 | 0.0 |

*Figure 5 - python*

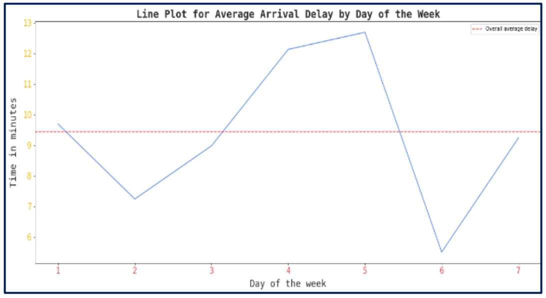


*Figure 6 - python*

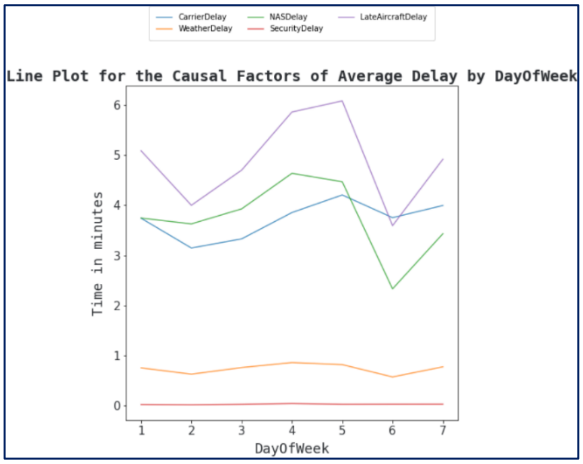
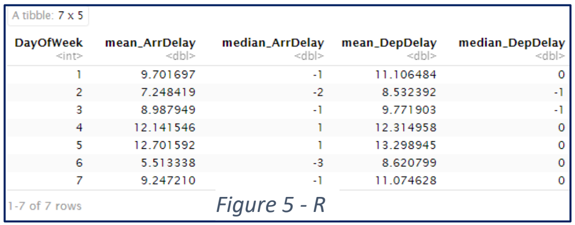


*Figure 7 - python*



A tibble: 7 x 5

| DayOfWeek<br>&lt;int&gt; | mean_ArrDelay<br>&lt;dbl&gt; | median_ArrDelay<br>&lt;dbl&gt; | mean_DepDelay<br>&lt;dbl&gt; | median_DepDelay<br>&lt;dbl&gt; |
|---|---|---|---|---|
| 1 | 9.701697 | -1 | 11.106484 | 0 |
| 2 | 7.248419 | -2 | 8.532392 | -1 |
| 3 | 8.987949 | -1 | 9.771903 | -1 |
| 4 | 12.141546 | 1 | 12.314958 | 0 |
| 5 | 12.701592 | 1 | 13.298945 | 0 |
| 6 | 5.513338 | -3 | 8.620799 | 0 |
| 7 | 9.247210 | -1 | 11.074628 | 0 |

1-7 of 7 rows

*Figure 5 - R*

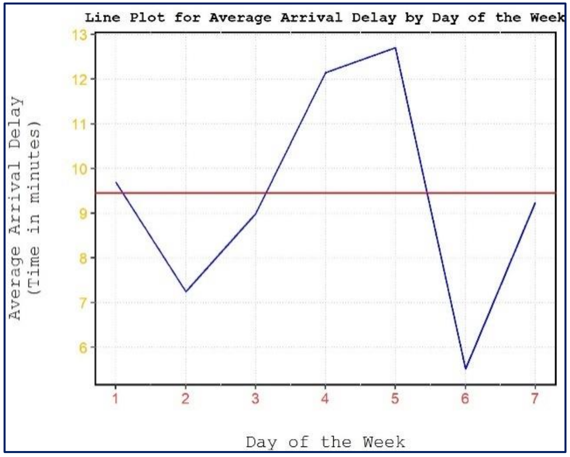


*Figure 6 – R*

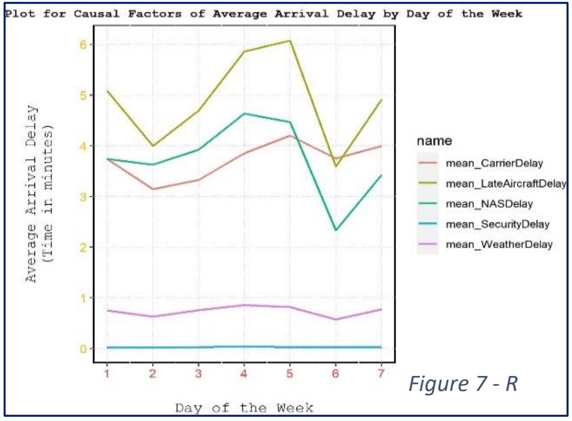
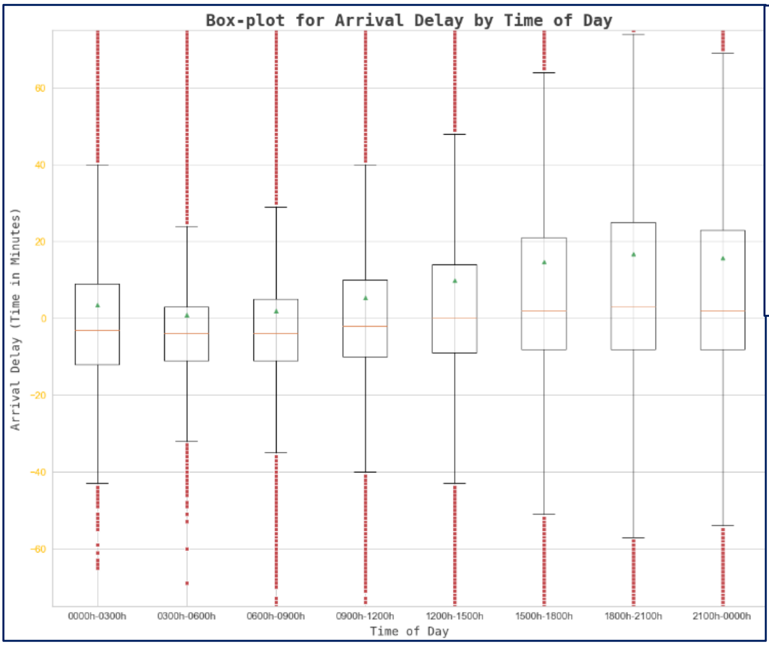


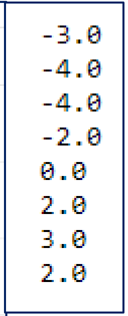
*Figure 7 - R*

According to figure-6, the 6th day of the week has the lowest average arrival delay. Assuming that the week starts on Monday, this means that Saturday has the lowest mean and median ArrDelay at 5.5 minutes and -3 minutes respectively (figure-5). Moreover, the most distinct causal factors of delay which are the LateAircraftDelay, NASDelay and CarrierDelay also show the least impact on Saturday. Hence the best time of week to fly in order to minimize delays would be on Saturday.

## *Best time of day to fly*

When finding the best time of day to fly we consider the CRSDepTime instead of the regular DepTime. This is because passengers only have access to the scheduled departure times when they book their flights. As such, any recommendation must be made based off of CRSDepTime. Having considered this, we first proceed to check for errors in the CRSDepTime column. The minimum and maximum CRSDepTime was found to make sure there were no outliers (such as 2430, 2740, -0600 etc.). The dtype of the CRSDepTime column was also found to make sure that all values contained in the column were in fact integers, thereby confirming the absence of non-numerics. Since there are 24 hours in a day, the times were binned into 8 groups (0000-0300h, 00300-0600h, 0600-0900h, 0900-1200h, 1200-1500h, 1500-1800h, 1800-2100h and 2100h-0000h) allocating a span of 3 hours for each group. This was done to reduce granularity and make visualization easier. Then the ArrDelays were stored as a multidimensional list which was grouped as specified above and boxplots were plotted and compared along with their summary statistics (Figure-8).





```
-3.0
-4.0
-4.0
-2.0
0.0
2.0
3.0
2.0
```

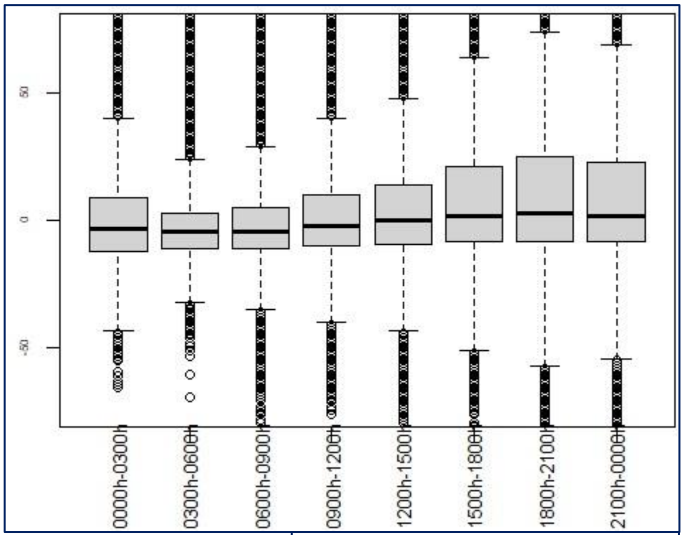
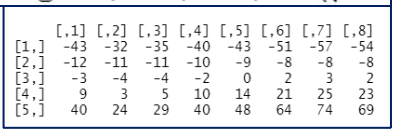


*Figure 8 - R*



```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]  -43  -32  -35  -40  -43  -51  -57  -54
[2,]  -12  -11  -11  -10   -9   -8   -8   -8
[3,]   -3   -4   -4   -2    0    2    3    2
[4,]    9    3    5   10   14   21   25   23
[5,]   40   24   29   40   48   64   74   69
```

*Figure 8 - python*

As shown in figure-8 and the adjoint summary statistics boxes, the time period of 0300-0600h and 0600-0900h have the lowest median ArrDelay with -4 minutes. (In the R output this is the row [3,]). Furthermore, the green triangles in the figure which indicate the mean ArrDelay shows that the time period of 0300-0600h has the lowest average ArrDelay at approximately 1 minute. Moreover, the 0300-0600h period has the smallest upper tail whisker, indicating that the maximum delay faced is comparatively the lowest out of all other time periods. Therefore, based on this it is evident that the best time to fly to minimize delays is between 03:00AM and 06:00AM.

## QUESTION 2 - Do older planes suffer more delays?

Firstly, the required packages were imported in Python (NumPy, pandas, seaborn, matplotlib) and R (dplyr, plotly, ggplot2). The main dataset was then imported and the unnamed/X column was removed. The supplementary dataset plane-dataset.csv was also imported making sure to replace empty values with NA in R for easy null value identification. The column names of both the full dataset and plane dataset were compared and it was found that they share a common column which is the tail number column. We then filter the datasets keeping only the 'TailNum', 'ArrDelay' and 'DepDelay' columns in the full dataset and the 'tailnum' and 'year' columns in the plane dataset. The 'year' column is our area of interest as it refers to the Plane Manufactured Year. Thereafter, the plane dataset was checked for null values and 549 of them were found in the 'year' column. This contributed to roughly 10.92% of the dataset. Since this is a fairly negligible amount and no other valid method of imputation for these unique missing values was available, they were removed from the plane dataset. The null values were removed only after selecting the required columns so that maximum data is preserved. Now a copy of the cleaned plane dataset was made only in python so as to avoid the "A value is trying to be set on a copy of a slice from a DataFrame" error. Thereafter, the plane dataset 'tailnum' column was renamed to 'TailNum' to match the 'TailNum' column in the full dataset so that merging is possible. The merge was then carried out by performing an inner join merge between the full dataset and plane dataset based on the TailNum and a new merged dataset was formed. There were no null values in the merged dataset but, the dtype of the year column was found to be an object. This meant that there was a possibility that it might contain non-numerical values and/or unfeasible year values. Hence the value counts were checked and it was found that the year column contained '0000' and 'None' values. These are irregularities which do not belong in the 'year' column and hence the rows containing these values were removed. Since the 'year' column is now solely numerical it was converted from object to integer type in order to facilitate a line plot visualization. Therefore, a variable which contains the mean ArrDelay grouped by the plane manufacture year was created and a line plot of Average Arrival Delay against Plane Manufacture Year was plotted.

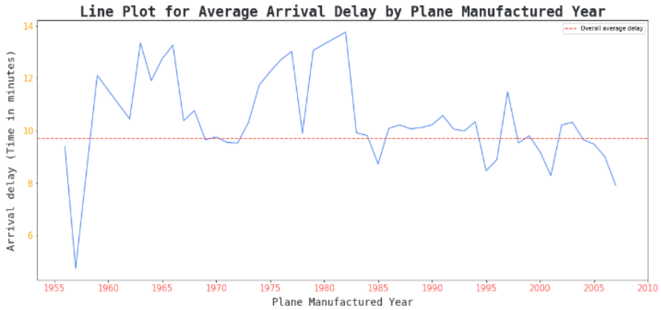


*Figure 9 -python*

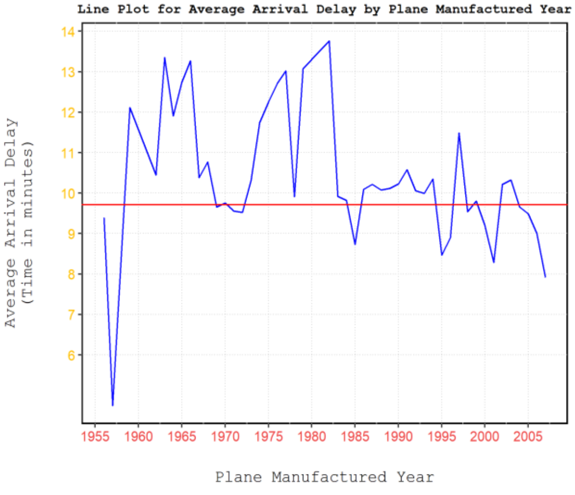


*Figure 9 - R*

As per figure-9, it can be seen that there is a lot of volatility in delays in older planes. The planes manufactured between 1955 and 1985 appear to have a lot of positive spikes in average arrival delay times, whereas the planes which were manufactured after 1985 showed less volatility and shorter spikes, and gradually settled down and plateaued near the overall average.

The average arrival delay depends on a lot of other factors and not just the plane manufacture year, but it is clearly visible here from the line plot that older planes have a higher average arrival delay than compared to newer planes.

## QUESTION 3 - Change in number of people flying between different locations over time?

### *Assumptions*

- Number of flights was used as a proxy for number of people as passenger count data was not available.
- We considered states as our choice of location over airports and cities as the latter two were very granular and hence were very difficult to compare.

The required python packages (NumPy, pandas, seaborn, matplotlib) and R packages (tidyr, dplyr, plotly, ggplot2, hrbrthemes, pheatmap, ComplexHeatmap, tidyverse) were imported along with the full dataset. The BiocManager was installed prior to loading the ComplexHeatmap library as it is a Bioconductor package. The unnamed/X column was removed from the full dataset as it provided no value to our analysis. Then the supplementary airports.csv dataset was imported making sure to replace empty values with NA in R for easy null value identification. The airports dataset was required since it contained information regarding the locations of each airport which is a primary focus of this question. Thereafter, the airports dataset was checked for null values and 12 each were found in the 'city' and 'state' column. The locations of the iata codes (airport codes) with missing values were attempted to be found by searching for duplicate iata codes, but no duplicates were found. Therefore, since there were no iata duplicates to cross-reference and fill in missing data and since all 12 null values are in the 'state' and 'city' column (which are our main areas of interest for this question) we went ahead and removed all null values in the airports dataset.

Thereafter it was required to decide upon the level of geographical scale we would be comparing the change in number of flights. Therefore, the value counts of 'city' and 'state' were computed. There were 2675 cities and 56 states. Hence to reduce clutter and granularity in the visualizations a macro level approach was considered by selecting states as the comparison factor.

The full dataset was then filtered down to only contain the Year, Origin and Dest columns. The airports dataset was filtered to only contain the iata and state column. After the filtration, in order to find both the Origin state and Destination state of each flight a deep copy of the filtered airports data frame was created. Therefore, we now have the full dataset and 2 filtered airports datasets (namely airports_1 and airports_2). The airports_1 dataframe copy was used to find the origin state and airports_2 to find the destination state. In order to facilitate the merge of both airports data frames with the full dataset the column names of the airports datasets were altered. The airports_1 'iata' column was changed to 'Origin' and the 'state' column was changed to 'Origin_State'. Similarly, the airports_2 'iata' column was changed to 'Dest' and the 'state' column was changed to 'Dest_State'. Finally, the full_dataset was merged with airports_1 using an inner join on 'Origin' and then the resulting merged dataset was again merged with airports_2 using an inner join on 'Dest'. This provided us with the Origin State and Destination State of each flight in the full dataset (now named the merged_dataset)

After checking for null values and validating proper types of the columns, the merged dataset was split into two, each containing the 2006 and 2007 flight records separately (namely the merged_dataset_2006 and the merged_dataset_2007). Then the number of flights that occurred between each state in each year was found. This was done by grouping the year-wise subset data frames by Origin State, obtaining their value counts for each Destination State, unstacking them and filling NA values with zero. It was required to sort the dataframes by 'Origin_State' and then 'Dest_State' in R to maintain consistency. Moreover, in R, the dimensions of the two data frames were 52x53 compared to python which had 52x52. This extra column (first column) was a character type column and resulted in issues when doing mathematical operations on the two dataframes in R. Therefore, to avoid this and to also avoid the "Error in FUN(left, right): non-numeric argument to binary operator" error the first column of both dataframes were made into the row indexes, hence making both dataframes in R contain only numeric values. Since now both dataframes have the same dimensions (52x52), each cell in the 2007 data frame was subtracted by the corresponding cell in the 2006 dataframe. Thus, providing us with a single dataframe (called the subtracted_dataset) containing the information regarding the change in number of flights that occurred between 2006 and 2007 between each state. This was saved as a csv file for easy viewing and cross referencing later on in Excel.

A heatmap was chosen as the ideal visualization representative. In order to standardize the change in flights to a range of -1 to +1 a weighted index was introduced. This was done by first finding the state which showed the greatest positive or negative change in number of flights. It was found to be Texas with a reduction of 20197 flights. Thereafter, each value in the subtracted_dataset was divided by 20197 thus creating a standardized set of data ranging from -1 to +1. Then a heatmap was plotted making sure that the minimum and maximum threshold of the colours was anchored at -0.25 and +0.25 respectively, so that the locations with a magnitude of change in flights that were 25% or higher than that of Texas show up as hotspots on the heatmap. This was done so that the extremely large changes do not drown out the comparatively smaller changes. Finally, the hotspots were cross-referenced to the subtracted_dataset dataframe in order to find the exact value of the change in number of flights of each location between the year 2006 and 2007.

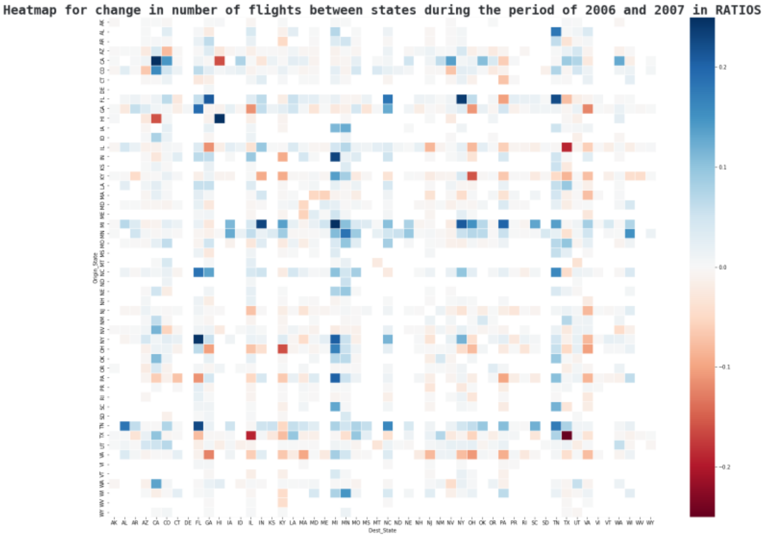
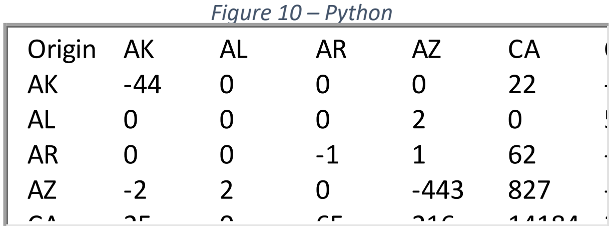


Figure 10 – Python



| Origin | AK | AL | AR | AZ | CA |
|---|---|---|---|---|---|
| AK | -44 | 0 | 0 | 0 | 22 |
| AL | 0 | 0 | 0 | 2 | 0 |
| AR | 0 | 0 | -1 | 1 | 62 |
| AZ | -2 | 2 | 0 | -443 | 827 |
| CA | 25 | 0 | 65 | 216 | 14184 |

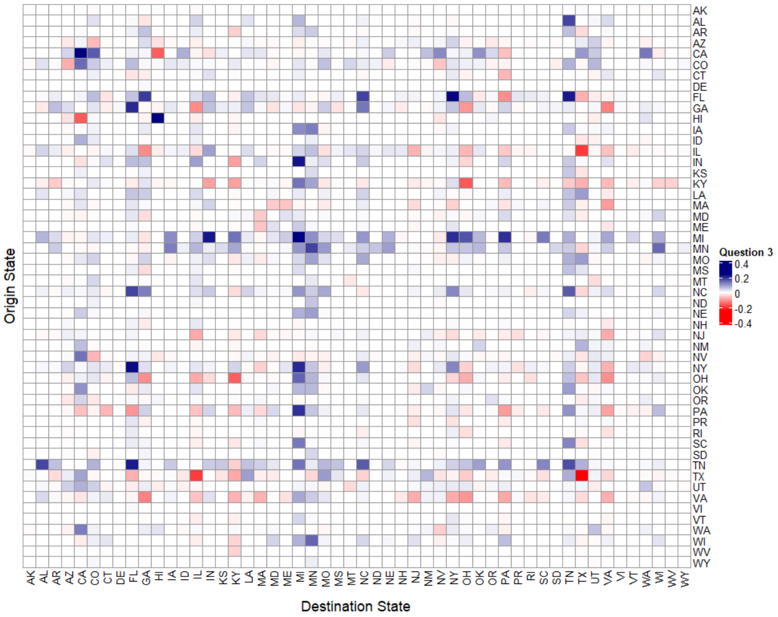
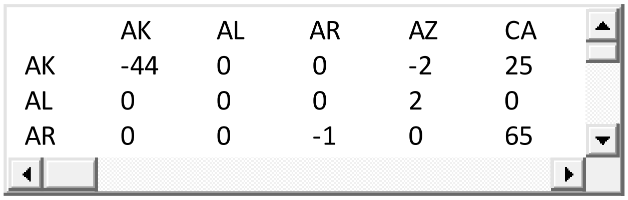


Figure 10 - R



| | AK | AL | AR | AZ | CA |
|---|---|---|---|---|---|
| AK | -44 | 0 | 0 | -2 | 25 |
| AL | 0 | 0 | 0 | 2 | 0 |
| AR | 0 | 0 | -1 | 0 | 65 |

Figure-10 shows the final heat map alongside the embedded CSV file of the subtracted_dataset. The blue spots show where there has been an increase in number of flights while the red spots show a decrease. The greatest increases in number of flights were from California-California (14184 flights), Hawaii-Hawaii (19655), Michigan-Michigan (5653), New York-Florida (4938), Michigan-Indiana (4646). The greatest decreases in number of flights were from Texas-Texas (-20197), Texas-Illinois (-3905), Ohio-Kentucky (-3259) and California-Hawaii (-3242). Therefore, it is evident that over the span of 1 year California, Hawaii, Michigan, New York and Florida had an exponential positive growth while Texas, Illinois, Ohio and Kentucky had receding/stagnant change in the number of passengers.

## QUESTION 4 - Can you detect cascading failures as delays in one airport create delays in others?

### *Assumptions*

*Total delay is used as the dependent variable in the analysis for 2 reasons. Firstly, the Arrival Delay is what airports have to deal with when planes fly in from other airports. Even if a plane had a departure delay as long as it arrives on time the DepDelay negates itself and no delay is carried forward into the arrival airport. Therefore, we include the ArrDelay in this analysis to capture the cascading effect. However, to be more wholistic, even though the Departure delay may not have as serious of an impact on other airports it still causes inefficiencies in the departing airports. Such as causing runway traffic, causing scheduling problems for air traffic control and desynchronization of airport operations which add up and affect other planes too and creates delays and causes inefficient management of the airport. Hence to capture this, DepDelay must also be included. Hence the Total Delay which is the summation of both ArrDelay and DepDelay was used as the main analytical attribute.*

Firstly, the required packages were imported in Python (NumPy, pandas, seaborn, matplotlib) and R (tidyr, dplyr, plotly, ggplot2, lubridate, stringi, janitor, vtree). The main dataset was then imported and the unnamed/X column was removed. The types of Year, Month, DayofMonth and CRSDepTime were checked to make sure they were integers. Since all 4 columns were integers, only the minimum and maximum values had to be checked to make sure there were no outliers. In order to proceed further with the analysis, the Year, Month, DayofMonth and CRSDepTime for each row had to be combined to form a column which shows the exact date and time of estimated departure of each flight saved in datetime format. We change to datetime format as it is very easy to compare dates which are in that format over dates which are in floats or integers. This was accomplished using different methods in R and Python.

In python, the minutes portion of the CRSDepTime was first extracted and converted into a string type object. Similarly, the hours portion of the CRSDepTime was also extracted and converted into a string type object. It was noted that there might be CRSDepTimes where the thousands and hundreds places do not contain a value (for example 45 which indicates 00:45AM). Therefore, these will be saved as empty values in the Hours_CRSDepTime variable. Hence, they must be located and replaced with zero to maintain consistency and avoid errors when converting to datetime64[ns] format. Thus, after confirming the presence of such values, they were replaced with the character “0”. In order for the datetime conversion function to work in pandas, all single digit days must be zero padded. Hence after converting the DayofMonth column to string type, the single digit days (1 to 9) were iterated through a lambda function and a “0” was added in front them using a string formatter and a regular expression. Thereby converting the days from 1,2,3,…,9 to 01,02,03,…,09. Finally, the individually extracted Days, Months, Hours and Minutes were combined using a pandas datetime conversion function and the exact date and time values (in datetime64[ns] format) were produced for each flight and was stored in a new column called ‘CRSDepTime_Datetime’ in the full dataset.

In R, we use the lubridate package to implement datetime conversion. We opted for the POSIXct conversion over the POSIXlt conversion as POSIXct stores date and time in seconds, with the number of seconds beginning from 1st January 1970. Storing in units of seconds is beneficial as it is optimized for use in large dataframes since it reduces computation time and processing power when performing conversions. Here we first extracted and converted the minutes portion and hours portion of CRSDepTime into a string using the stringi package and saved them as a list. As mentioned in the python conversion the extracted hours may contain empty values. Hence this was checked and empty hour values were filled with the character “0”. The day of month, month and year columns were also extracted, converted to character format and saved as separate lists. These were then pasted into a new column called "CRSDepTime_Datetime" in the full dataset as individual strings in the form of Day-Month-Year-Hour-Minutes for each row. Finally, these strings were converted to POSIXct format using the lubridate library and saved in the "CRSDepTime_Datetime" column.

A Total Delay column was created in the full dataset by adding the ArrDelay and DepDelay. The full dataset was then sorted by the newly converted CRSDepTime_Datetime so that all the flights made by each plane were arranged in a continuous organized timeline. Then a Previous Delay column was created by first grouping the Total Delay values by each tail number and then shifting/lagging the entire column down by one row. Thus, allowing for each row to show the current delay (TotalDelay) and the previous delay (PreviousDelay) of a flight of a specific airplane. Null values were caused in the PreviousDelay column due to two reasons. Firstly, due to the lag in the PreviousDelay column the top entry for each tail number had a NaN value. Secondly, the planes who had only one flight recorded will by default have no previous delay and thus showed a NaN value. Hence, these null values were found and removed. Then a scatter plot was produced to demonstrate the relationship between the magnitude of previous delays and current delays of flights.

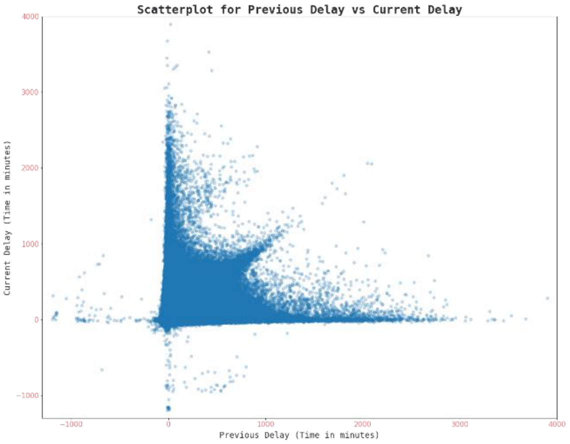


*Figure 11 - python*

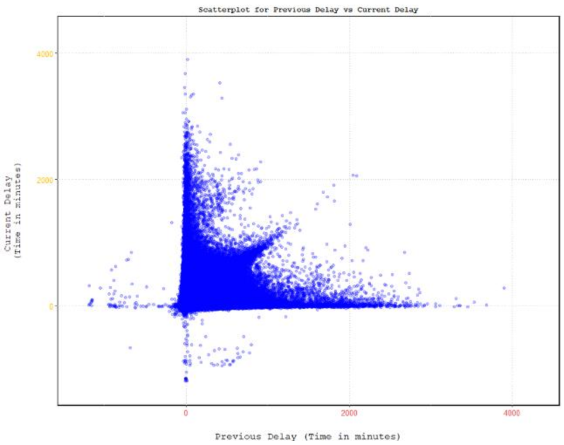
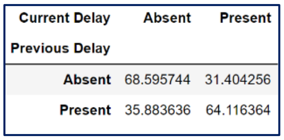


*Figure 11- R*

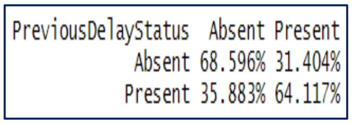
All of this was done so that we can compare how a previous delay in one flight of a specific plane affects the delay in the next flight of that same plane. Thus, analysing the cascading effect of delays caused in one airport flowing over and affecting another airport. As shown in figure-11 it can be seen that there is a slight upward trend along the 45° line. Meaning as the magnitude of previous delay increases the current delay also increases, indicating to us that the larger the previous delay the larger will be the current delay faced.

Before proceeding further, a copy of the full dataset was made in python to avoid the “A value is trying to be set on a copy of a slice from a DataFrame” error. Then a Current delay status column and Previous delay status column were created and binary inputs of ‘Present’ and ‘Absent’ were filled. ‘Present’ indicating a delay greater than zero and ‘Absent’ indicating a delay less than or equal to zero for their respective types of delays and their status columns. A cross tabulation was then formulated with the columns indicating the number of Current delays and the rows indicating the number of Previous delays. Since the previous delays affect the current delays, in accordance to the direction of causation, we found the row totals and divided each corresponding cell in that row by its row total and then multiplied it by 100, so that it can be identified how the previous delay affects the current delay as a percentage.



| Current Delay | Absent | Present |
|---|---|---|
| Previous Delay | | |
| Absent | 68.595744 | 31.404256 |
| Present | 35.883636 | 64.116364 |

*Figure 12 - python*



```
PreviousDelayStatus  Absent Present
             Absent 68.596% 31.404%
            Present 35.883% 64.117%
```

*Figure 12 - R*

As shown in figure-12 in the absence of previous delays only 31.4% current delays were present. However, in the presence of previous delays there were 64.1% current delays present. This is an indicator that the presence of previous delays leads to more frequent current delays. Hence indicating a cascading effect in delays between airports.

Another variable of interest was the LateAircraftDelay. This is defined as the arrival delay at an airport which is caused due to the late arrival of the same plane at a previous airport. A second scatter plot was then made to analyse the relationship between LateAircraftDelay and current delay.

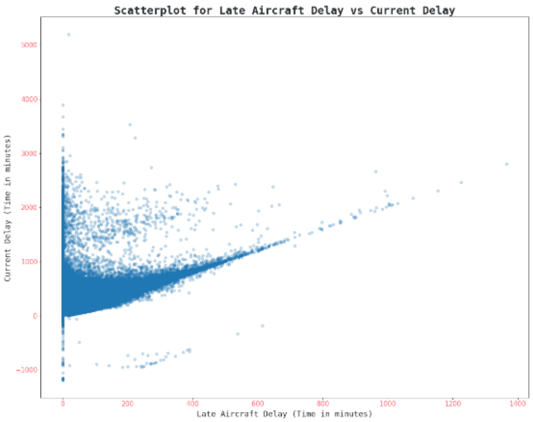


*Figure 13 - python*

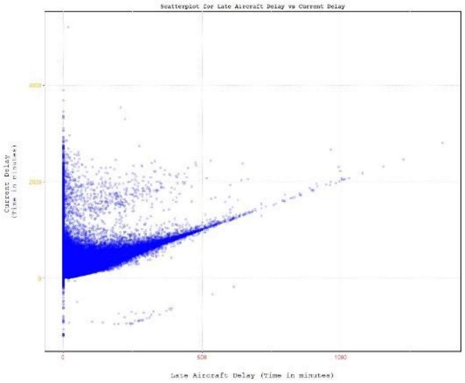


*Figure 13 - R*

As shown in figure-13, it is visible that there is a distinct upward trend between LateAircraftDelay and the current delay, indicating that an increase in LateAircraftDelay also increases the current total delay. This solidifies the fact that late arrivals of aircrafts in previous airports are affecting the delays occurring in subsequent airports.

Therefore, it is now distinctly evident that there is delay propagation and cascading ripple effects which causes delays in one airport to affect another downstream airport and create even more delays leading to a snowballing effect.

## QUESTION 5 - Use the available variables to construct a model that predicts delays.

### *Premise and Assumptions*

Delays cause a negative economic effect on airports, carriers and passengers; induces operation inefficiencies and damages the environment through excessive fuel consumption. Therefore, prediction of delays is vital to overcome these negative externalities and is the focus of this question. Here we will be predicting the arrival delay. This is because by predicting the arrival delay, the destination airport can account for any lags in flights that may happen and proactively prepare to face them without much distress. So, if all airports adopt this prediction strategy in unison, then they can mutually benefit by forecasting delays and streamlining airport activities.

The model that is to be built will be predicting the occurrence of an arrival delay. That is, when the ArrDelay is larger than zero it indicates that a delay has occurred whereas, if it is less than or equal to zero then a delay has not occurred.

The following variables have been used as explanatory features in the model: 'Year', 'Month', 'DayofMonth', 'DayOfWeek', 'CRSDepTime', 'DepDelay', 'Origin', 'Dest' and 'PlaneManufactureYear'. A majority of these variables were selected since they displayed a distinct relationship with delays in the previous questions. The departure delay (DepDelay) can be used as a predictor variable in our model since its value will be available at the time of take-off from the origin airport. This is done under the assumption that airports have an integrated system where information can be shared without restraint (especially DepDelay information) between origin and destination airports. It is in fact well known that modern day airports do share data to improve customer satisfaction by speeding up check-ins, alleviating passenger anxiety and having better passenger movement to gain higher retail sales.

The CRSDepTime (even though it is a time-based quantity) was kept as a numerical variable since using it as a categorical variable would result in one-hot encoding roughly 2400 categories which is very inefficient.

We also assume that the sample is a good representative of the population and shows all the necessary characteristics.

### *Methodology*

Firstly, the required python packages (NumPy, pandas, seaborn, matplotlib, sklearn) and R packages (dplyr, tidyr, tidyverse, mlr3, mlr3pipelines, mlr3viz, mlr3learners, rmarkdown, corrplot, future, precrec) were loaded. The main dataset was then imported and the unnamed/X column was removed. Then the supplementary dataset plane-dataset.csv was imported making sure to replace empty values with NA in R for easy null value identification. A correlation matrix heatmap (figure-14) was then plotted to find out the correlated variables so as to avoid the effect of multicollinearity when selecting features. The only noteworthy revelation through this was discovering that ArrDelay and DepDelay were highly correlated with a value of 0.93. Also, LateAircraftDelay was moderately correlated with ArrDelay and DepDelay (at approximately 0.6), hence It was concluded that ArrDelay, DepDelay and LateAircraftDelay would not be included as explanatory variables together simultaneously.

Furthermore, it was also found that none of the variables other than the CarrierDelay, WeatherDelay, NASDelay, SecurityDelay, LateAircraftDelay, DepDelay and ArrDelay show a correlation higher than 0.33. Moreover, apart from the DepDelay which is available at the time of take-off from the origin airport, the rest of the delays cannot be used as features in the model as they cannot be calculated until the plane finishes its journey. Therefore, we were forced to select the variables that were identified to affect delay from the previous 4 questions to use as features in our predictive model.

The diverted and cancelled columns were not included in the model as they were all zeros. The plane dataset columns were then filtered out to only keep the "talinum" and "year". We renamed the plane_dataset tail number column to match that of the full_dataset tail number column so that merging is possible. The "year" column was also changed to "PlaneManufactureYear" to reduce ambiguity. An inner join merge was performed between the full dataset and plane dataset based on the TailNum and hence the PlaneManufactureYear of each flight was found and saved in the newly created merged dataset. Then the merged dataset was checked for null values and 692721 were found in the PlaneManufactureYear column and hence those rows were removed. The PlaneManufactureYear column was found to be of a character type. Therefore, it was inspected further and it was found to contain "0000" and "None" values. These were also removed from the merged dataset.

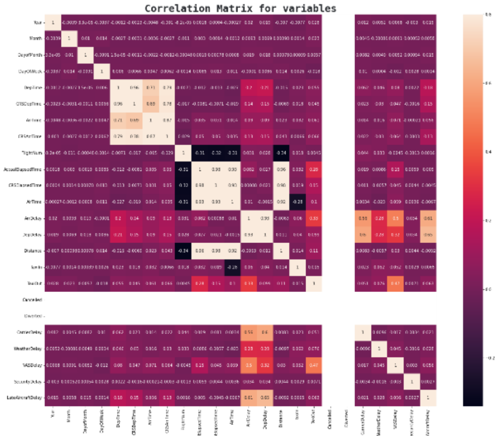


Figure 14 - python

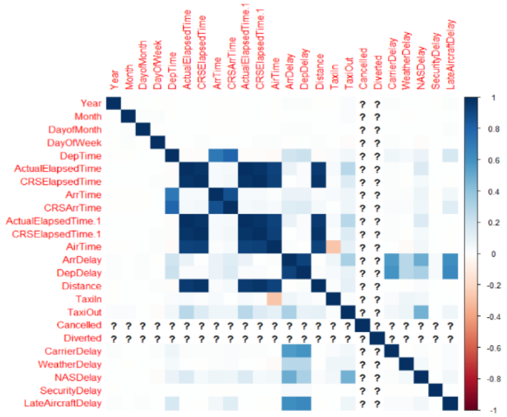


Figure 14 - R

The merged dataset was then filtered down to only contain the variables which were required for the model. The following variables were kept in the filtered dataset which was renamed as full_dataset_1: 'Year', 'Month', 'DayofMonth', 'DayOfWeek', 'CRSDepTime', 'ArrDelay', 'DepDelay', 'Origin', 'Dest' and 'PlaneManufactureYear'.

After renaming each month with its proper name instead of numerical identifiers, the Year, Month, DayofMonth, DayOfWeek columns were converted from an integer type to string (character) type so that it can be used as categorical variable. This was done because when imputing as a numerical variable the imputer uses mean and median, but using such statistics as substitutes for time-based quantities will not make sense as they do not have any numerical magnitude which can be ranked or given preference to (for instance, the 12th month is neither bigger nor better than the 1st month).

Since the scope of this model is to predict whether an arrival delay occurs or not, we create an ArrDelayStatus column. This indicates 'Present' when the ArrDelay is greater than zero and 'Absent' when the ArrDelay is smaller than or equal to zero. The ArrDelayStatus column was then converted to a factor type in R so that the "Target column 'ArrDelayStatus' must be a factor or ordered factor" error can be avoided when creating a new task. In addition, the ArrDelay column was manually removed from the full dataset in R, as we cannot use it as a feature since it is highly correlated with DepDelay. Thereafter, random samples were taken since we did not possess the computational hardware required to process 14 million rows of data.

## Python

A 50% random sample was taken from the prepared dataset. The categorical and numerical features were identified and separated and then a copy of the sample dataset only containing the columns of the explanatory features was made. A numerical pipeline was created to impute missing numerical values with either the mean or mode and then scaling them to normalize the data. Similarly, a categorical pipeline was created to impute missing categorical data using a constant or the mode and one-hot encoding was done to binarize the categorical data by creating dummy variables so that it can be used to train the predictive models. Finally, both of these pipelines were combined.

A logistic regression model was then setup. The training and testing datasets were also formed from the copy of the sample dataset created earlier. The train-test split was set at 70% training and 30% testing and we ensured that reproducibility was possible in future. After setting up the parameter grid, a grid search was conducted in order to choose the best set of optimal hyperparameters for the learning algorithm. This is called tuning the hyperparameters. Furthermore, in order to reduce the computation time, we implemented parallelization which conducts multiple parallel processes across multiple CPUs simultaneously. Thus, making the grid search more efficient. Thereafter, the tuned model was fit using the training dataset. The model was then evaluated using a ROC (Receiver Operating Characteristics) curve and confusion matrices

### R

A 5% random sample was taken from the prepared dataset and parallelization was configured. A new model building task was created using the random sample to predict the ArrDelayStatus. We then selected our evaluation method to be the Area Under the ROC Curve (AUC) and specified the type of model (learning algorithm) that we wanted to build to be a logistic regression model. Thereafter the steps in the pipeline were setup. The steps included, identifying missing data, imputing missing numerical features, scaling numerical features, imputing missing categorical features and one-hot encoding categorical features. We then put together all the steps, specified the learning algorithm, built the pipeline and created the graph object (the graph is the final object created after building the pipeline). We then encapsulated the graph object as a learner so that it can be used by mlr3 to perform resampling and do benchmarks.

A logistic regression model was then built. After ensuring replicability, the training and testing datasets were formed with a train-test split set of 70% training and 30% testing. The model was then trained using the training dataset and thereafter evaluated. Predictions were made using the model and the testing dataset and then were evaluated using the AUC measure, confusion matrix and ROC curve.

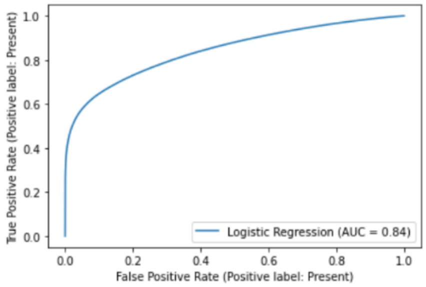


*Figure 15 - python*

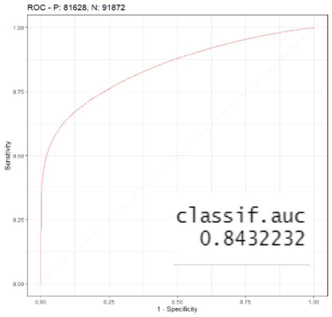


*Figure 15 - R*

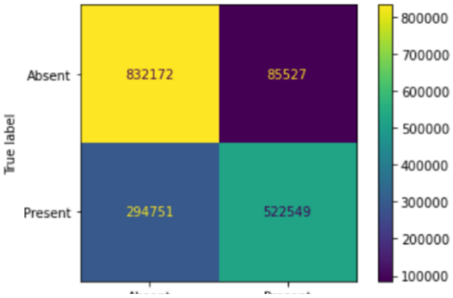
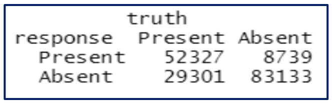


*Figure 16 - python*



```
         truth
response  Present Absent
  Present   52327   8739
  Absent    29301  83133
```

*Figure 16 - R*

|  |  | Actual values |  |
|---|---|---|---|
|  |  | Positive | Negative |
| Predicted values | Positive | TP | FP |
|  | Negative | FN | TN |

### Interpretation

The Confusion Matrix (Error Matrix) of a binary classification model is a 2 x 2 matrix which evaluates the performance of a classification model by comparing the actual target values against the predictions made by the model. It is very useful in measuring the Precision, Recall, Accuracy, Specificity, and ROC curves. There are 4 regions in the matrix: the True Positives (TP), True Negatives (TN), False Positives (FP -Type 1 error) and False Negatives (FN - Type 2 error).

$$Precision = \frac{TP}{TP + FP}$$

$$TPR = Recall = Sensitivity = \frac{TP}{TP + FN}$$

$$TNR = Specificity = \frac{TN}{TN + FP}$$

$$FPR = 1 - Specificity = \frac{FP}{TN + FP}$$

$$F1\ score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

When predicting flight delays, we want it to be as accurate as possible; but it is more of an issue when the model predicts no delay but a delay ends up occurring. Therefore, special attention must be paid to the number of False Negatives in the model which is captured by the recall (TPR). The precision of our model turned out to be 85.93% in python and 85.69% in R, which means that out of all the delays that were predicted to be present, 86% turned out to be actually present. On the other hand, the recall (TPR) was 63.93% in python and 64.31% in R, meaning 64% of all delays that were present were correctly predicted by our model. Since this is a value greater than 50% the model is of acceptable standards. Moreover, the F1-score which is the harmonic mean of precision and recall were calculated and came up to be 0.734 which is indicative that we possess good precision and recall values.

The Receiver Operator Characteristic (ROC) curve is used as an evaluation metric in binary classification models. It is a probability curve that plots the True Positive Rate (TPR) against False Positive Rate (FPR). The TPR is called the Sensitivity or Recall and the FPR is 1-Specificity, where Specificity is the True Negative Rate (TNR). The Area Under the Curve (AUC) is a measure of the ability of a classification model to distinguish between the 2 classes and is generally used as a way of summarizing the ROC curve. The higher the AUC, the better will be the performance of the model in distinguishing between positive and negative classes.

From figure-15, it can be seen that our ROC curve has an AUC of 0.84. When the AUC is between 0.5<AUC<1, there will be a high chance that the model will be able to correctly distinguish the delay ‘present’ class values from the delay ‘absent’ class values. This is because the model is able to detect larger numbers of true positives and true negatives over false positives and false negatives. Therefore, a value of 0.84 is a good indicator that our model predicts the occurrence of a delay with a high level of predictive accuracy.

# Appendix

## *Glossary of Terminology and variable descriptions*

- Year – Year of occurrence of the flight. Can be 2006 or 2007.
- Month – Month of occurrence of the flight. Takes a value between 1 and 12.
- DayofMonth – Day of the month that the flight occurred. Takes a value between 1 and 31
- DayOfWeek - Day of the week that the flight occurred. Takes a value between 1 and 7
- DepTime – The actual Departure Time of the plane (hhmm)
- CRSDepTime – The scheduled Departure Time of the plane (hhmm)
- ArrTime - The actual Arrival Time of the plane(hhmm)
- CRSArrTime - The scheduled Arrival Time of the plane(hhmm)
- UniqueCarrier – Unique Carrier Code
- FlightNum – Flight Number
- TailNum – Plane Tail Number
- ActualElapsedTime – The actual elapsed time of a flight (in minutes)
- CRSElapsedTime – The estimated elapsed time of a flight (in minutes)
- AirTime – The in-air time of the plane (in minutes)
- ArrDelay – The Arrival Delay of the flight (in minutes)
- DepDelay – The Departure Delay of the flight (in minutes)
- Origin – The origin IATA airport code
- Dest - The destination IATA airport code
- Distance – The distance travelled by the plane (in miles)
- TaxiIn – The taxi in time (in minutes)
- TaxiOut - The taxi out time (in minutes)
- Cancelled – Indicator of whether the flight was cancelled or not (1- cancelled, 0- not cancelled)
- CancellationCode – Reason for flight cancellation (A = carrier, B = weather, C = NAS, D=security)
- Diverted – Whether the flight was diverted (1=yes, 0=no)
- CarrierDelay - Carrier delay is within the control of the air carrier
- WeatherDelay - The delay which is caused by extreme or hazardous weather conditions
- NASDelay - Delay that is within the control of the National Airspace System (NAS)
- SecurityDelay - Security delay is caused by evacuation of a terminal or concourse, re-boarding of aircraft because of security breach
- LateAircraftDelay - Arrival delay at an airport due to the late arrival of the same aircraft at a previous airport.