OW ZI LYNN

200615190

ST2195 Coursework 2021/2

Table of Contents

1.0 Load Data

Create data frames called *airports* with airports.csv, *carriers* with carriers.csv, *planes* with plane-data.csv and *ontime* with 2006.csv.bz2 and 2007.csv.bz2. The csv and csv.bz2 files can be downloaded from https://doi.org/10.7910/DVN/HG7NV7

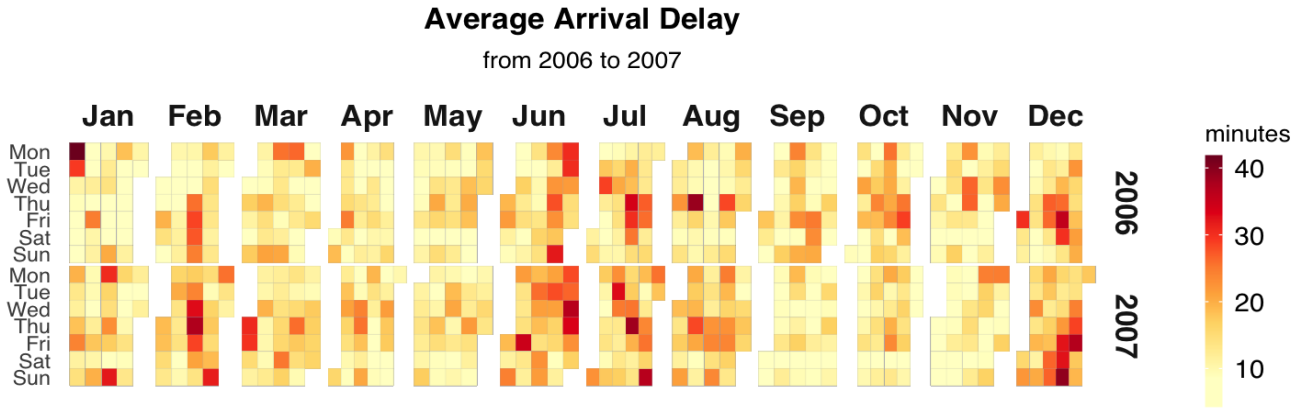
1.1 When is the best time of day, day of the week, and time of year to fly to minimize delays?

Idea: Only arrival delays are considered as departure delay does not necessarily result in arrival delay. Average arrival delays are computed for each date, month, day of week and each time interval to identify the period with shortest average arrival delay to fly to minimize delays.

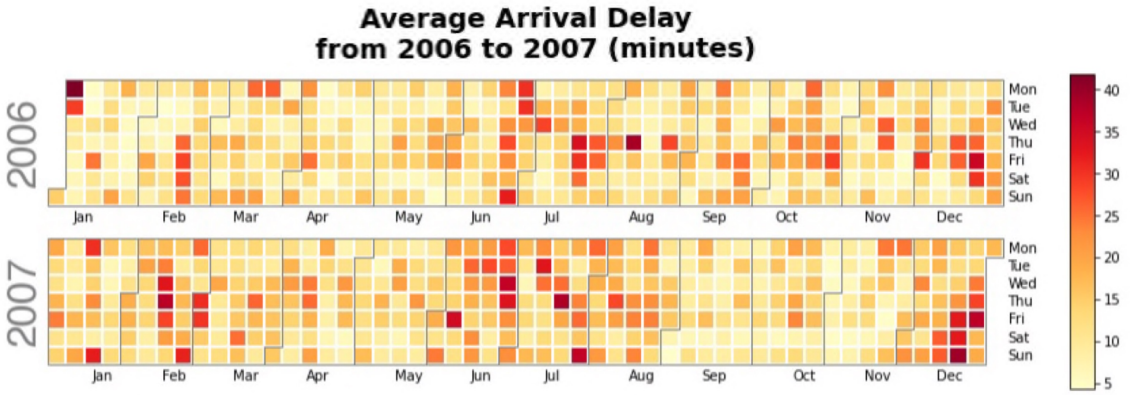
Approach: For *ontime*, convert all column names to lowercase. Remove duplicate rows. There are scheduled and actual departure and arrival time. Convert all time from 2400 to 0000 and remove time more than 2400. In R, create new columns to store time as POSIXct objects with as.*POSIXct()* function in year-month-day hour:minutes format in EST time zone. In Python, time is stored as datetime in the same format in R with *pd.to_datetime()* function. Add columns to group time in 4-hour intervals with *cut()* function in R and Python. Convert month in 1,2,..12 to Jan, Feb and Dec with *lubridate*::*month()* function in R and *calendar.month_abbr[ ]* in Python. Convert day of week in 1,2,..7 to Mon, Tue, .. Sun with *lubridate*::*wday()* function in R and *calendar.day_abbr[ ]* in Python. In R, identify the week of the year with *isoweek()* function. Last few days of December might be the first week of the following year. Convert them from 1st to 53rd week. First few days of January might be the 52nd or 53rd week of the previous year. Convert them to the 1st week.

For computation of arrival delay, drop non-applicable arrival delay values in *ontime.* Filter out diverted and canceled flights. If the flight is early, the delay values would be negative, convert them to 0 to indicate no delay. For calendar plots, group by date to compute average daily arrival delay. Each rectangle of the calendar represents a day of the year. The color of each rectangle is determined by the length of average daily arrival delays. Shade of dark red demonstrates long average daily arrival delay whereas shade of light yellow depicts short average daily arrival delays [Figure 1a & 1b]. In R, arrange the x-axis based on week, y-axis based on day of week in inverted manner. Each column is for each distinct month whereas each row is for each distinct year. In Python, use the *calplot* package to generate a calendar plot with data arranged in series such that date is the index and daily average arrival delay are the values. For bar plots [Figure 1c & 1d], group by month to compute monthly average arrival delay. For bar plots [Figure 1e & 1f], group by day of week to compute average arrival delay of day of week. For bar plots [Figure 1g & 1h] , group by 4-hour time intervals to calculate average arrival delay of each time interval.

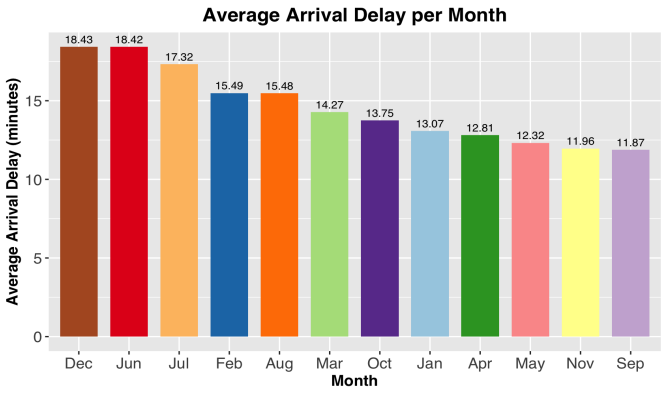
Results : The best timing to fly to minimize delay is identified based on the shortest average arrival delay. September is the best month, Saturday is the best day of week and the time interval between 8am to 12pm is the best time of the day to fly to minimize delays.



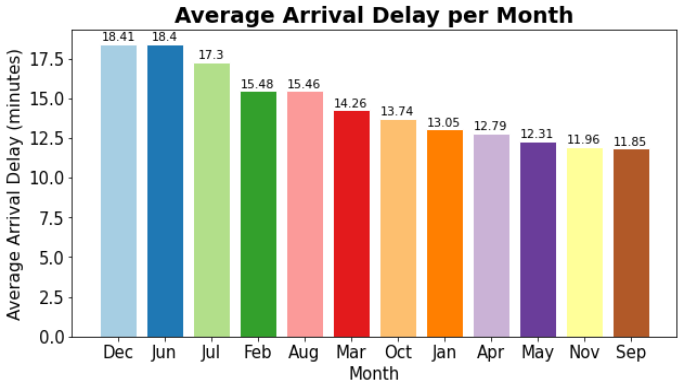
[Figure 1a] R: RStudio



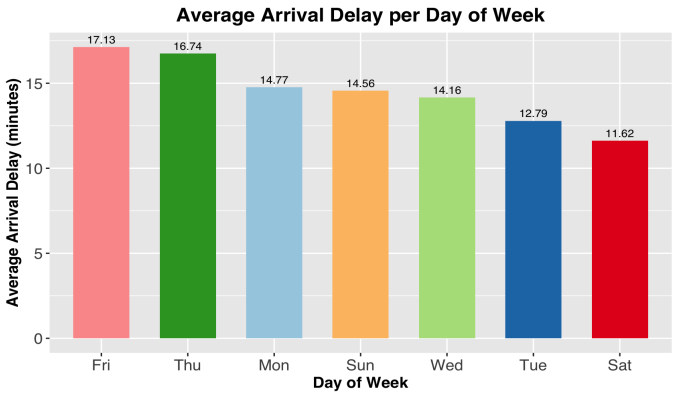
[Figure 1b] Python : Jupyter Notebook



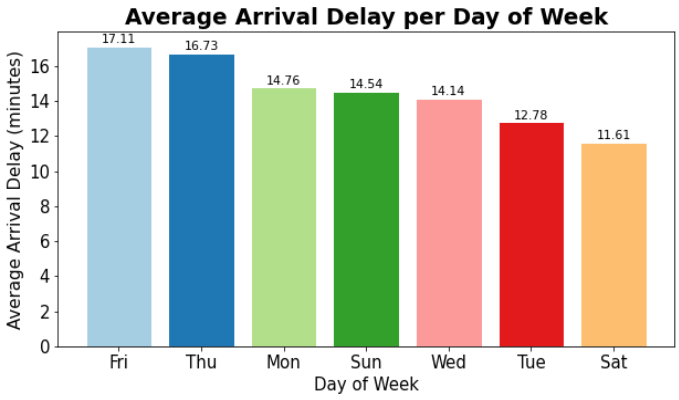
[Figure 1c] R: RStudio



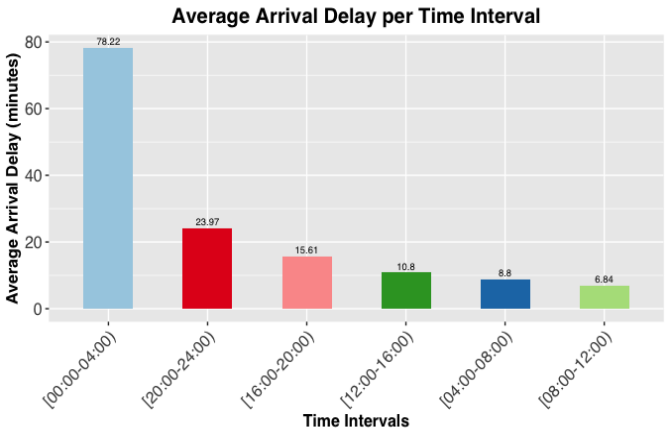
[Figure 1d] Python : Jupyter Notebook



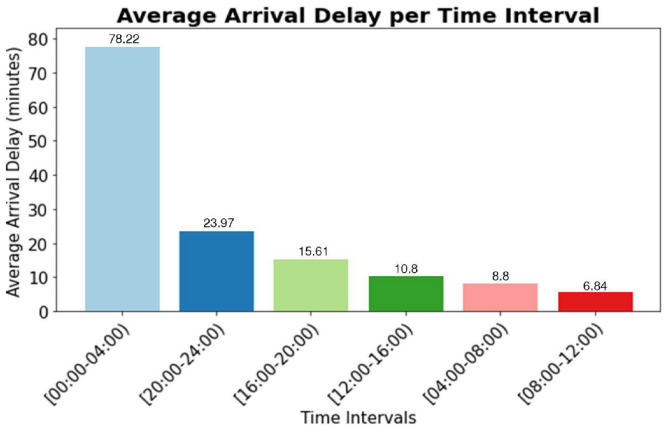
[Figure 1e] R: RStudio



[Figure 1f] Python : Jupyter Notebook



[Figure 1g] R: RStudio



[Figure 1h] Python : Jupyter Notebook

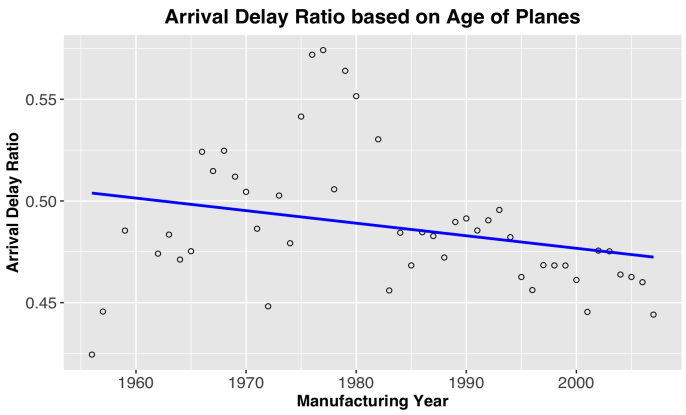
1.2 Do older planes suffer more delays?

Idea: Determine age of plane with manufacturing year. Arrival delay ratio for each manufacturing year is computed and plotted with scatter plots fitted with the best fit line. Results from the linear regression model are analyzed to recognise the relationship between manufacturing year and arrival delay ratio.

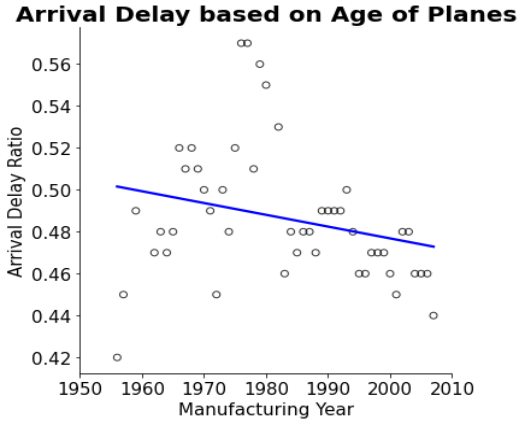
Approach: The age of the plane is determined by the manufacturing year, which is the year column in *planes.* Rename the year column in *planes* as *manufacturingyear.* Remove blanks and “None” in *manufacturingyear* and convert the values to numeric to select manufacturing year more than 0. Create new columns, *biarrdelay* in *ontime,* where value of 1 is given for arrival delay more than 0. Else, 0 is assigned. Inner join *planes* with *ontime1* where diverted and canceled flights are filtered out from *ontime* by *tailnum* for computation of arrival delay ratio.

Arrival delay ratio is calculated based on manufacturing year. Firstly, compute the number of arrival delays, no arrival delay and sum of arrival and no arrival delay for each *manufacturingyear*. Then, divide the number of arrival delays by sum of arrival and no arrival delay for arrival delay ratio. Scatter plots are plotted [Figure 2a & 2b] with a best fit line to demonstrate the linear relationship between age of the plane and its arrival delay ratio. Linear regression in R is conducted with *lm() function* and in Python with *statsmodel.add_constant() , statsmodel.OLS()* and *fit()* function. Manufacturing year is the predictor variable whereas arrival delay ratio is the response variable. The summary of the results is obtained with *the summary()* function in R and Python.

Results: The p-value of manufacturing year in both R and Python are both larger than 5%, indicating that manufacturing year or age of plane is not statistically significant. R-squared of 7% illustrates that the predictor variable is not explaining the variation of the response variable well [Figure 2c & 2d]. In short, we cannot conclude that older planes suffer more delays.



[Figure 2a] R: RStudio



[Figure 2b] Python : Jupyter Notebook

```
Call:
lm(formula = arrratio ~ manufacturingyear, data = carrdelayage)

Residuals:
      Min        1Q    Median        3Q       Max
-0.079417 -0.016746 -0.005246  0.010591  0.083248

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)       1.7110058  0.6435376   2.659   0.0108 *
manufacturingyear -0.0006171  0.0003245  -1.902   0.0635 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.03276 on 46 degrees of freedom
Multiple R-squared:  0.07288,   Adjusted R-squared:  0.05273
F-statistic: 3.616 on 1 and 46 DF,  p-value: 0.0635
```

[Figure 2c] R: RStudio

OLS Regression Results

Dep. Variable:	arrdelayratio	R-squared:	0.066
Model:	OLS	Adj. R-squared:	0.046
Method:	Least Squares	F-statistic:	3.262
Date:	Wed, 23 Mar 2022	Prob (F-statistic):	0.0775
Time:	18:29:43	Log-Likelihood:	98.975
No. Observations:	48	AIC:	-193.9
Df Residuals:	46	BIC:	-190.2
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	1.6020	0.618	2.593	0.013	0.359	2.845
manufacturingyear	-0.0006	0.000	-1.806	0.077	-0.001	6.44e-05

Omnibus:	7.094	Durbin-Watson:	0.647
Prob(Omnibus):	0.029	Jarque-Bera (JB):	6.569
Skew:	0.620	Prob(JB):	0.0375
Kurtosis:	4.321	Cond. No.	2.70e+05

[Figure 2d] Python : Jupyter Notebook

1.3 How does the number of people flying between different locations change over time?

Idea: Investigate how the number of people change monthly from 2006 to 2007 between the destination city with highest number of inbound count and top 10 origin cities with highest number of outbound count to this destination city. This step is repeated with cities with 2nd and 3rd highest inbound count. The number of people is estimated with the number of outbound flights. Each flight is assumed to have 100 people.

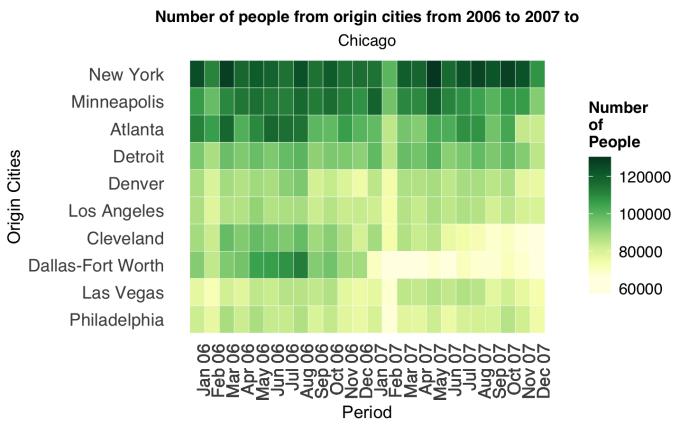
Approach: To compute the number of people between cities, canceled flights are excluded. To identify the top 3 destination cities with the highest number of inbound counts, destination cities are included in *topinbound,* which is the inner join of *ontime2,* which excluded canceled flights from *ontime and* airports by *dest* and *iata* columns. Then, group by destination cities and compute the inbound count for each city. Retain data of the top 3 cities with the highest number of inbound count which are Chicago, Atlanta and Dallas-Fort Worth for *topinbound.*

destorigin is set up for origin and destination data. *ontime2* is joined with airports by *dest* = *iata*. Then, repeat this step with *origin = iata.* To ensure that *destorigin* only contains data from top 3 cities with highest inbound count, create *destorigin2* which is the inner join of *destorigin* and *topinbound.* Create a column called *ym*, by concatenating the values from the *year* and *month* columns.

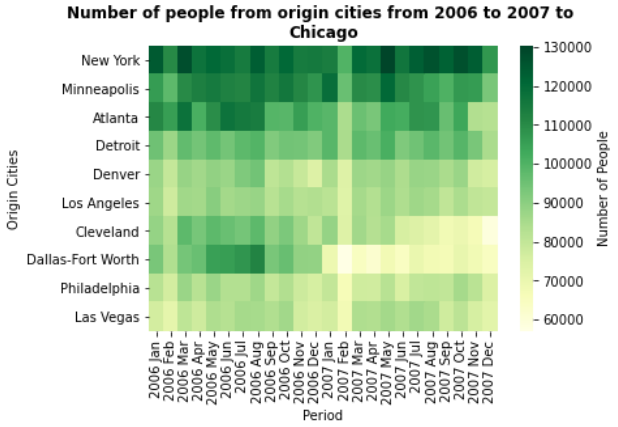
To identify the top 10 origin cities with highest outbound count to Chicago, group by origin cities and compute the outbound count of each city. Ensure the destination city of *destorigin* only contains Chicago. This step is repeated for Atlanta and Dallas-Fort Worth with a for loop. The outputs of Chicago with top 10 origin cities, Atlanta with top 10 origin cities and Dallas with top 10 origin cities are stored as 3 new dataframes in R with *assign()* function. Then, merge these 3 data frames into 1. In python, 3 data frames are stored in a dictionary and could be accessed with 3 different keys.

To identify the monthly outbound count of each 10 origin cities to Chicago, group by origin cities and *ym* for the computation of outbound count using the merged data frame in R or dictionary in Python. Ensure the destination city is Chicago. This step is repeated with another 2 cities. The outbound count is then multiplied by 100 to represent the number of people. Heatmaps are plotted to show how the number of people change between cities over time. Each rectangle demonstrates the number of people. Dark green represents a higher number of people whereas light yellow illustrates a lower number of people [Figure 3a to 3f].

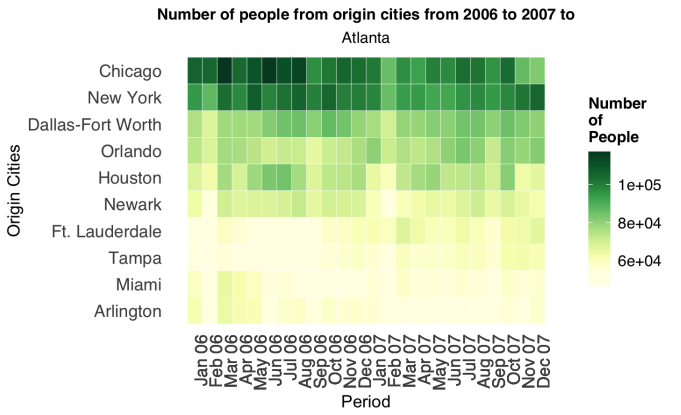
Results: The number of people drastically dropped in 2007 compared to 2006 from Dallas to Chicago [Figure 3a & 3b]. The number of people from Chicago to Dallas were comparatively lower in 2007 compared to 2006 [Figure 3e & 3f]. Further investigations could be conducted of why the number of people from Dallas to Chicago and Chicago to Dallas decreased. The number of people to Chicago, Atlanta and Dallas were usually low in February [Figure 3a to 3f] . Discounts could be offered for February flights so that more people would choose to fly in February instead of the peak months to reduce congestion at the airports.



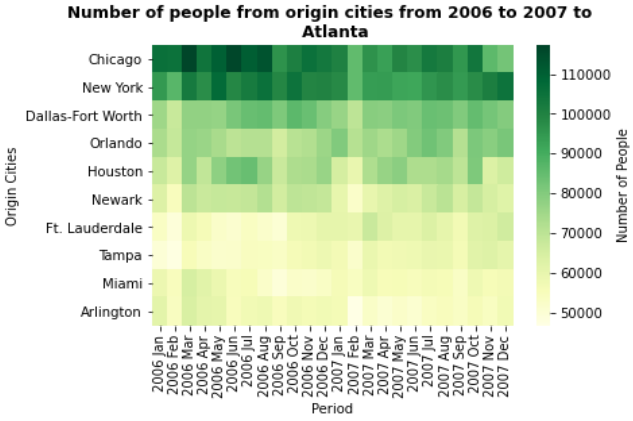
[Figure 3a] R: RStudio



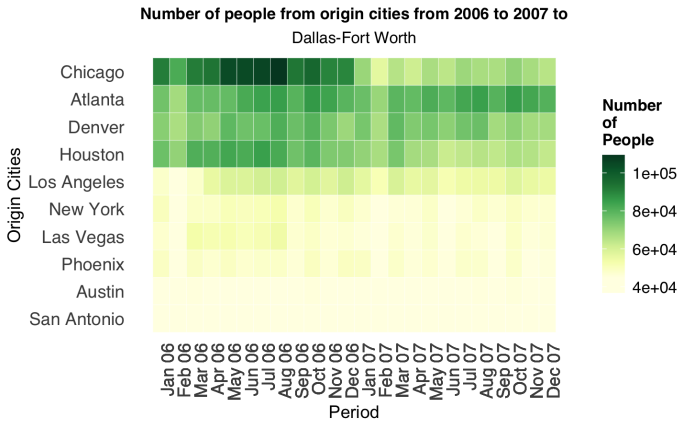
[Figure 3b] Python : Jupyter Notebook



[Figure 3c] R: RStudio



[Figure 3d] Python : Jupyter Notebook



[Figure 3e] R: RStudio

[Figure 3f] Python : Jupyter Notebook

1.4. Can you detect cascading failures as delays in one airport create delays in others?

Idea: Study how departure delay of planes from an airport led to cascading arrival and departure delays of planes to other airports on the day where average arrival delay is the longest.

Approach: To identify cascading failures, canceled and diverted flights are excluded. For origin and destination airports data, *destorigin3* is set up. Join *ontime1* from *ontime* which excludes canceled and diverted flights with *airports* by *dest* = *iata*. Then, repeat this step with *origin* = *iata.* Ensure this data frame only contains the data from the date with the longest average arrival delay, which is on 2006-01-02 [Figure 1a & 1b]. Arrange the data by tail number and actual departure time in ascending order. Ensure that arrival time is later than departure time, departure time is later than previous arrival time. Select top 6 planes with highest number of flights on 2006-01-02. The top plane had 13 flights whereas the 2nd to 6th planes had 12 flights on 2006-01-02. Filter the data such that only data with departure and arrival delay are selected. Set up a data frame for original data and another data frame for filtered data with departure and arrival delay. To identify flights with cascading departure and arrival delay, compare the index of the original data frame with the filtered data frame and remove all rows below if there are missing indexes in the middle of the data frames. Row with missing index implies that plane did not have delay for that timing, hence breaking the cascading effect.

Results: Out of the 6 planes with the highest number of flights on 2006-01-02, only 2 planes had more than 3 consecutive departure and arrival delays respectively. These 2 planes were N226SW which had 11 consecutive departure and arrival delays and N271YV which had 12 consecutive departure and arrival delays. Parts of the cascading delays for these 2 planes are shown in Table 4a to 4d. To study the relationship between airports for cascading delays, network analysis is conducted. The nodes are the airports and their sizes increase with degree centrality. The node size is bigger when an airport has more connections to other airports and connections from other airports to itself. The edges are the paths between 2 airports and their weights increase with the number of connections between 2 airports. From figure 4e to 4h, for both N226SW and N271YV with cascading delays, Los Angeles International airport was the most central airport which had the highest sum of connections to other airports and connections of other airports to itself. Furthermore, the paths from San Diego International-Lindbergh to Los Angeles International and vice versa were the more frequent paths used during the cascading delays. For N271YV, the paths from Ontario International to Los Angeles International and vice versa were also frequently cruised.

	tailnum <chr>	flightnum <dbl>	crsdatedeptime <S3: POSIXct>	datedeptime <S3: POSIXct>	crsdatearrtime <S3: POSIXct>	datearrtime <S3: POSIXct>	originairport <chr>	destairport <chr>
1	N226SW	6068	2006-01-02 07:00:00	2006-01-02 07:01:00	2006-01-02 07:43:00	2006-01-02 08:09:00	San Diego International-Lindbergh	Los Angeles International
2	N226SW	6051	2006-01-02 08:00:00	2006-01-02 08:30:00	2006-01-02 09:06:00	2006-01-02 09:30:00	Los Angeles International	Fresno Yosemite International
3	N226SW	6175	2006-01-02 09:42:00	2006-01-02 09:58:00	2006-01-02 10:47:00	2006-01-02 11:40:00	Fresno Yosemite International	Los Angeles International
4	N226SW	6096	2006-01-02 11:15:00	2006-01-02 12:20:00	2006-01-02 12:00:00	2006-01-02 13:09:00	Los Angeles International	San Diego International-Lindbergh
5	N226SW	6096	2006-01-02 12:30:00	2006-01-02 13:37:00	2006-01-02 13:13:00	2006-01-02 14:28:00	San Diego International-Lindbergh	Los Angeles International

[Table 4a] R: RStudio

	tailnum	flightnum	datecrsdeptime	datedeptime	datecrsarrtime	datearrtime	originairport	destairport
0	N226SW	6068	2006-01-02 07:00:00	2006-01-02 07:01:00	2006-01-02 07:43:00	2006-01-02 08:09:00	San Diego International-Lindbergh	Los Angeles International
1	N226SW	6051	2006-01-02 08:00:00	2006-01-02 08:30:00	2006-01-02 09:06:00	2006-01-02 09:30:00	Los Angeles International	Fresno Yosemite International
2	N226SW	6175	2006-01-02 09:42:00	2006-01-02 09:58:00	2006-01-02 10:47:00	2006-01-02 11:40:00	Fresno Yosemite International	Los Angeles International
3	N226SW	6096	2006-01-02 11:15:00	2006-01-02 12:20:00	2006-01-02 12:00:00	2006-01-02 13:09:00	Los Angeles International	San Diego International-Lindbergh
4	N226SW	6096	2006-01-02 12:30:00	2006-01-02 13:37:00	2006-01-02 13:13:00	2006-01-02 14:28:00	San Diego International-Lindbergh	Los Angeles International

[Table 4b] Python : Jupyter Notebook

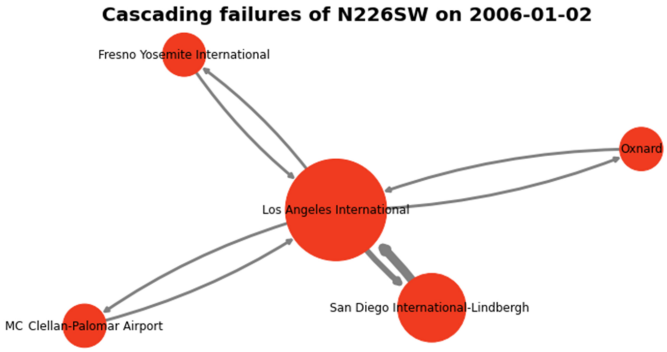
	tailnum <chr>	flightnum <dbl>	crsdatedeptime <S3: POSIXct>	datedeptime <S3: POSIXct>	crsdatearrtime <S3: POSIXct>	datearrtime <S3: POSIXct>	originairport <chr>	destairport <chr>
1	N271YV	6098	2006-01-02 06:38:00	2006-01-02 06:42:00	2006-01-02 07:10:00	2006-01-02 07:24:00	Los Angeles International	Ontario International
2	N271YV	6098	2006-01-02 07:31:00	2006-01-02 07:42:00	2006-01-02 07:59:00	2006-01-02 08:25:00	Ontario International	Los Angeles International
3	N271YV	6142	2006-01-02 08:29:00	2006-01-02 08:49:00	2006-01-02 08:59:00	2006-01-02 09:43:00	Los Angeles International	John Wayne /Orange Co
4	N271YV	6142	2006-01-02 09:43:00	2006-01-02 10:21:00	2006-01-02 10:15:00	2006-01-02 11:23:00	John Wayne /Orange Co	Los Angeles International
5	N271YV	6100	2006-01-02 10:31:00	2006-01-02 11:53:00	2006-01-02 11:05:00	2006-01-02 12:38:00	Los Angeles International	Ontario International

[Table 4c] R: RStudio

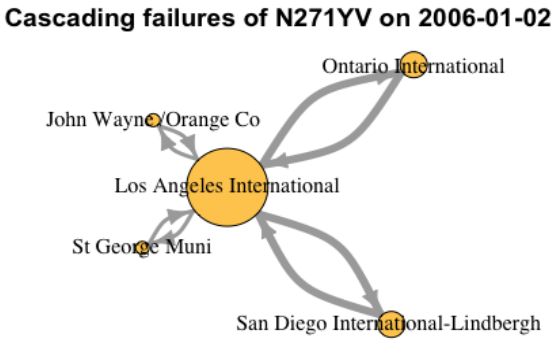
	tailnum	flightnum	datecrsdeptime	datedeptime	datecrsarrtime	datearrtime	originairport	destairport
0	N271YV	6098	2006-01-02 06:38:00	2006-01-02 06:42:00	2006-01-02 07:10:00	2006-01-02 07:24:00	Los Angeles International	Ontario International
1	N271YV	6098	2006-01-02 07:31:00	2006-01-02 07:42:00	2006-01-02 07:59:00	2006-01-02 08:25:00	Ontario International	Los Angeles International
2	N271YV	6142	2006-01-02 08:29:00	2006-01-02 08:49:00	2006-01-02 08:59:00	2006-01-02 09:43:00	Los Angeles International	John Wayne /Orange Co
3	N271YV	6142	2006-01-02 09:43:00	2006-01-02 10:21:00	2006-01-02 10:15:00	2006-01-02 11:23:00	John Wayne /Orange Co	Los Angeles International
4	N271YV	6100	2006-01-02 10:31:00	2006-01-02 11:53:00	2006-01-02 11:05:00	2006-01-02 12:38:00	Los Angeles International	Ontario International

[Table 4d] Python : Jupyter Notebook

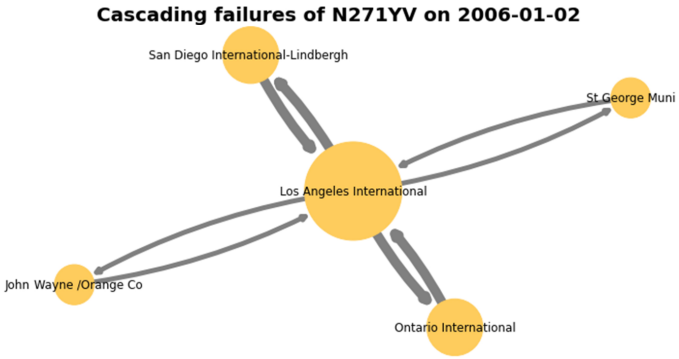
[Figure 4e] R: RStudio



[Figure 4f] Python : Jupyter Notebook



[Figure 4g] R: RStudio



[Figure 4h] Python : Jupyter Notebook

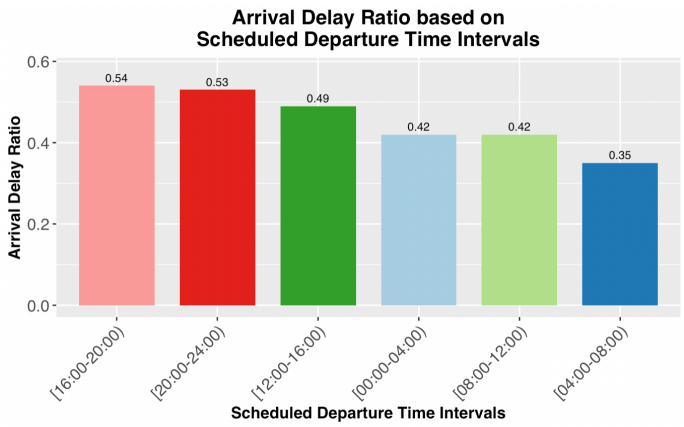
1.5 Use the available variables to construct a model that predicts delays.

Idea: A target variable is chosen and exploratory data analysis is conducted for features selection. Data is sampled to process datasets. Different classification models are trained with training sets and their performance is evaluated by running the trained models on test sets. Grid search with cross validations are conducted using a training set for hyperparameter optimisation. Classification error and ROC (Receiver Operating Characteristics) curves with AUC(Area Under The Curve) are measures used to compare performance of the classification models.

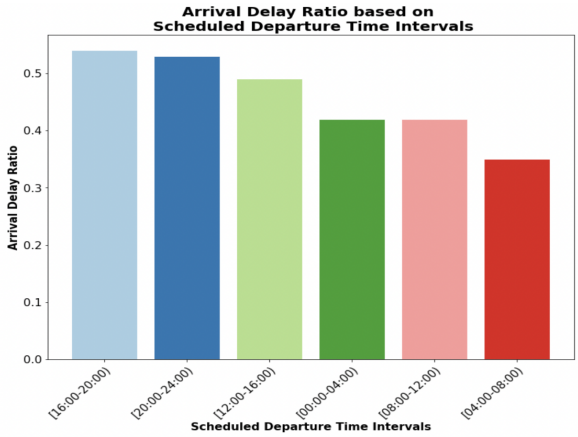
Approach: Conduct exploratory data analysis on features and select features that are useful to predict the target variable. Features such as scheduled departure and scheduled arrival time intervals, carrier, destination airport, origin airport, month and day of week are chosen as they seem to have an effect on target variable, *biarrdelay.* If the arrival delay is more than 0, *biarrdelay* = 1. Else, *biarrdelay* = 0. One of the visualizations for data exploration is shown in Figure 5a & 5b. These features are selected as their values could be known before flights take off. A data frame, ontime3 is set up by filtering out canceled and diverted flights from *ontime* and containing all the features and target variable needed for predictive modeling. Based on the summary of the data, *ontime3 is free of non-applicable values.*

The data from 2006 to 2007 is approximately 14 million rows. 20% of the data is sampled such that the distribution of all variables of the sampled data is the same with original data [Figure 5c & 5d]. In R, convert variables as ordered and unordered factors. For non-ordered factors, remove the first factor level, leaving n-1 columns using method = "treatment" to avoid multicollinearity issues. For ordered factors, convert the ordered factors to integers. In Python, convert all categorical variables as dummy variables. Remove the last dummy variables of each variable. From the sampled data, 75% of the data are selected randomly as training set and the rest for test set. After training the predictive model with the training set, run the trained models on the test set to evaluate their performance. It is important to have both training and test set to avoid overfitting. Use grid search with cross validations to identify hyperparameters of each model for most accurate predictions with the training sets. In R, select classification errors as a measure for performance comparisons of different learners, In Python, use ROC with AUC as measures. Logistic regression, penalized logistic regression, gradient boosting, classification tree and random forest are the chosen predictive models for response variable which is binary categorical.

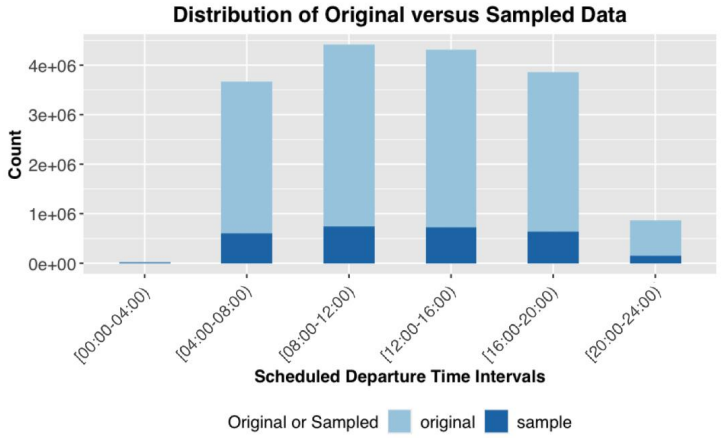
Results: Based on Figure 5e, in terms of median and range, the performance of random forest is the best with lowest classification error of 40%. Based on Figure 5f, gradient boosting is the best model as it has the highest AUC, indicating the largest area under the probability curve called ROC. The result of the performance comparison of different models using different measures is different. The difference might be due to imbalanced datasets [Figure 5g & 5h]. For the target variable of arrival delay, there are less arrival delays in the dataset. If the model predicts every plane as no delays, it would have less classification error. Hence, the ROC curve which plots true positives against false positives is preferred for performance comparison. Gradient boosting is more likely to predict arrival delays or no arrival delays more accurately in the future at 62% chance [Figure 5f].



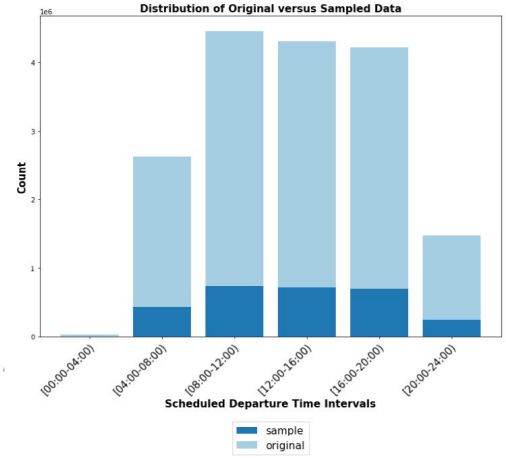
[Figure 5a] R: RStudio



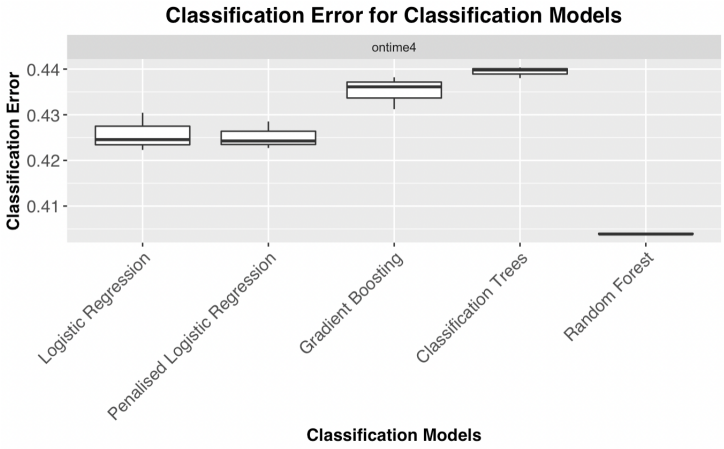
[Figure 5b] Python : Jupyter Notebook



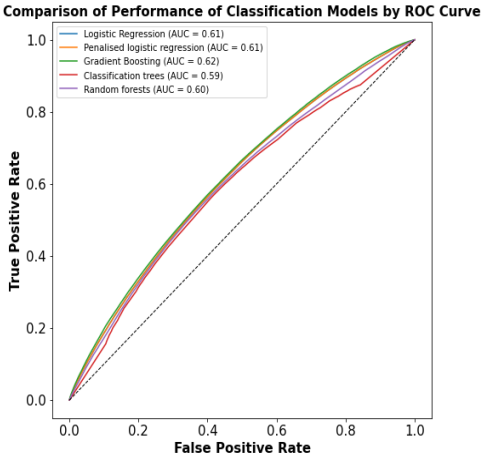
[Figure 5c] R: RStudio



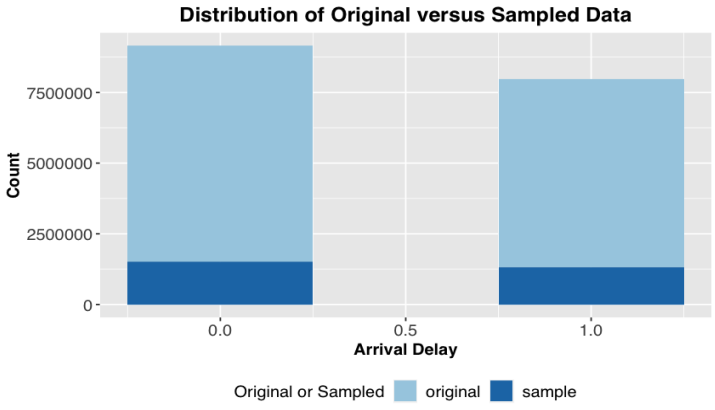
[Figure 5d] Python : Jupyter Notebook



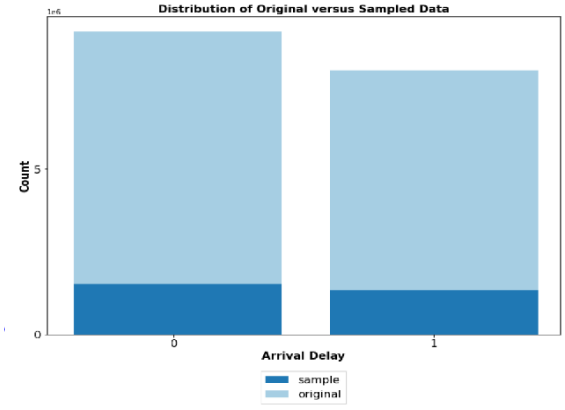
[Figure 5e] R: RStudio



[Figure 5f] Python : Jupyter Notebook



[Figure 5g] R: RStudio



[Figure 5h] Python : Jupyter Notebook