A Report as a Part of Coursework for the Module:

Programming for Data Science

(ST2195)

Rafid Akhyara Agung

200614920

Table of Content

Preliminary steps

Before beginning to write the code, we had to obtain the data set first. The data was obtained from the "Harvard Dataverse" (https://doi.org/10.7910/DVN/HG7NV7) From here, arbitrarily, we chose two successive years to analyze, the year 2004 and 2005.

The data from these two years were imported into a relational database, from which, we will query the data.

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

For the first question, we query the 2004 and 2005 data sets and take the following attributes: the *Year*, *Month*, *DayOfWeek* that the flight flew, the *CRSDepTime* (*Scheduled* departure time) , and *ArrDelay* (Arrival delay).

From this query, we already have the "day of the week" and the "time of year", for which we will use the months that the flights flew. The only thing we are missing is the "time of day". While we do have the departure times, they are far too granular. A better measure might be to use the *hour*, but we need to extract the hour first.

However, *CRSDepTime* is a plain integer, not in any special time format that can be recognized by R or Python. (e.g. 19:28 is 1928, 06:18 is 618, 00:04 is 4, etc.) Because of this, we use Regular Expressions (Regex). Regex allows us to search for patterns in text and perform a multitude of operations on them like extract, replace, count them, etc.

Before that, I had to reformat *CRSDepTime* into a 24-hour format as some entries were in the 2600s. To do that, I tested for any entries above 2359 and subtracted such entries by 2400.

Then we apply the Regex. Now since the way the data is entered provides different lengths, we can separate the times into two groups. Those with times less than or equal to two characters were considered as hour 0. (e.g. 24 = 00:24, 1 = 00:01, all hour zero.)

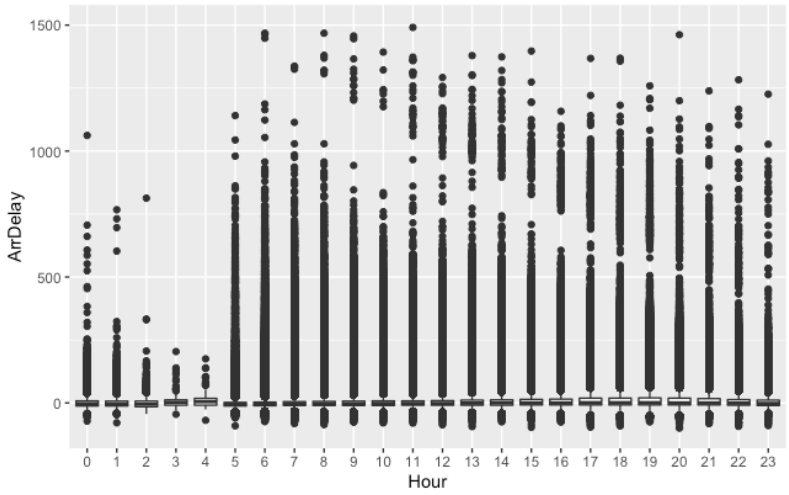
For those longer than two characters, we use the following expression: "(\\d+)\\d{2}"

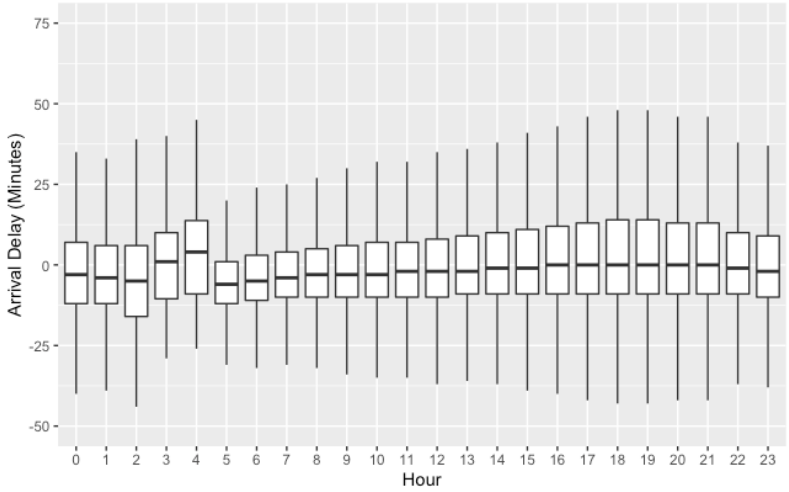
This roughly means, look for one or more digit ("(\\d+)") followed by two digits ("\\d{2}"). The parentheses around "\\d+" identifies it as a "group" and flags that part of the expression. This allows us to extract the group as the hour. So for instance 124 would be seen by the Regex as (1)24 and we extract the 1. (Or 1231 as (12)31, giving us hour 12.)

Once the hour extraction is laid out, it is relatively straight forward. The data set is filtered for any NA values (usually meaning the flight was cancelled.), we extract the hour and after that, we can create boxplots to obtain our answers.

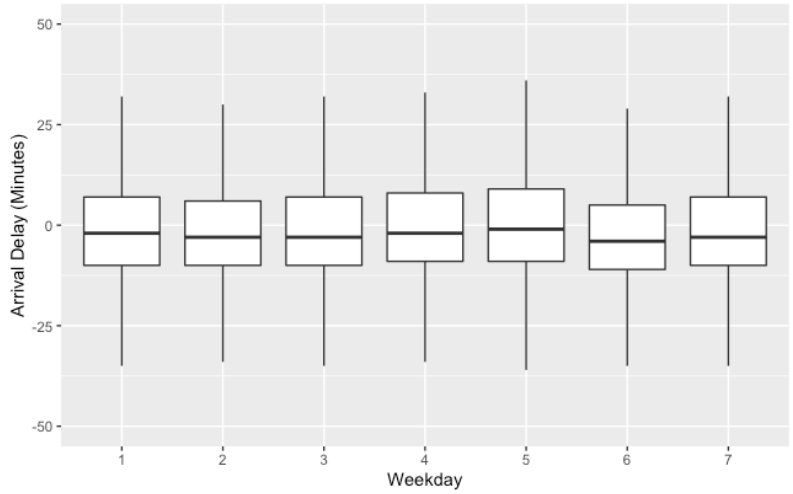
Boxplots were chosen due to the type of data being plotted, as we are trying to find the relationship between a continuous and a categorical variable.

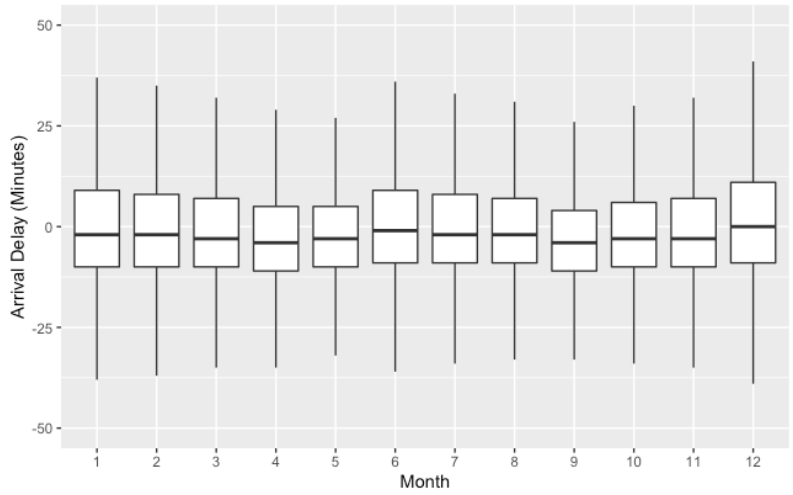
However, we will choose to look mainly at the plots without outliers as with outliers, it is too cluttered like so:





The best hour to fly according to the boxplot is around the early morning but after 4 o'clock as the delays seem to increase slightly over the day, and the range increases, and hence variance, as well, but they reset after 4.





The best weekday and month to fly though seem inconsequential as the delays do not seem to differ significantly.

Though, we can note that for the different months, the ranges are smaller for some months such as month 5 and 9 so though the median delays may be similar, the variance of those delays differ.

Do older planes suffer more delays?

For the second question, we query the 2004 and 2005 data sets and take the following attributes: *TailNum*, the tail number of a plane, unique to each aircraft, and *ArrDelay*. We also query from the planedata data set, taking the attributes: *TailNum* and *Year*, as in the year the plane started flying.

Answering this question is relatively straightforward. The first thing we need to do is a left join between the 2004-2005 data (which we set as "left") and the plane data and we join them by the *TailNum*. This pairs each *TailNum* in the 2004-2005 data with its corresponding *Year*. Doing a left join means we keep all the rows on the "left" table, regardless of whether or not it has a corresponding value in the "right" table.

For instance given these two tables:

2004-2005		Plane data	
TailNum	ArrDelay	TailNum	Year
N946UA	0	N903UA	1990
N903UA	53		
N901UA	32		

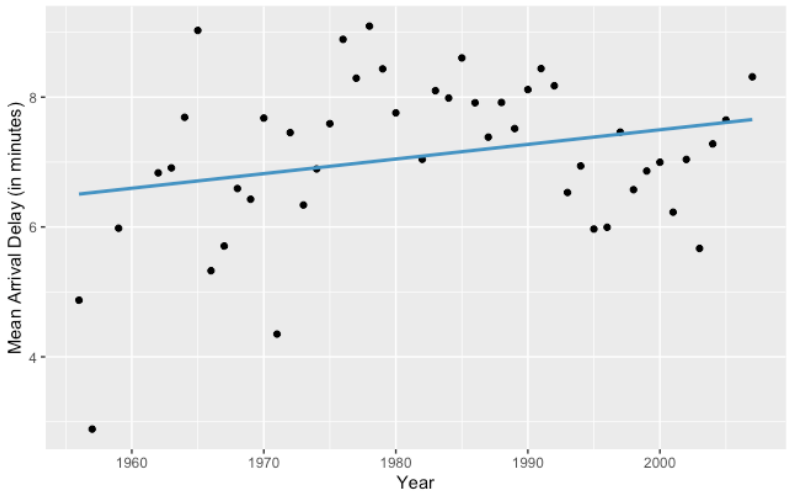
The left join-ed table would be:

N946UA	0	*NA*
N903UA	53	1990
N901UA	32	*NA*

Then we filter any NAs, and remove some mistakes like the *Year* being "None" or "0000". Then we group by *Year* and calculate the average *ArrDelay* for each *Year* group.

Then we make a scatter plot as both variables being plotted are continuous.

The code results in the below chart. We can see that as the Year increases (Age decreases), the mean delays tend to increase for the aircraft. Though from the plot it's not as strict, that is the general trend.

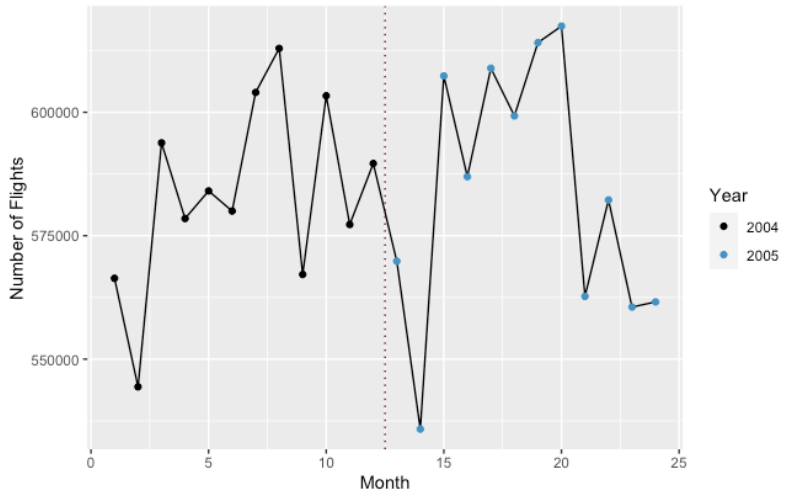


So to answer the question, no. Older planes in fact suffer less delays.

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

For the third question, we query the 2004 and 2005 data sets and take the following attributes: *Year*, *Month*, *Cancelled* (Binary value, 1 when flight was cancelled.), *Origin* (The flight's origin.) and *Dest*. (The flight's destination.)

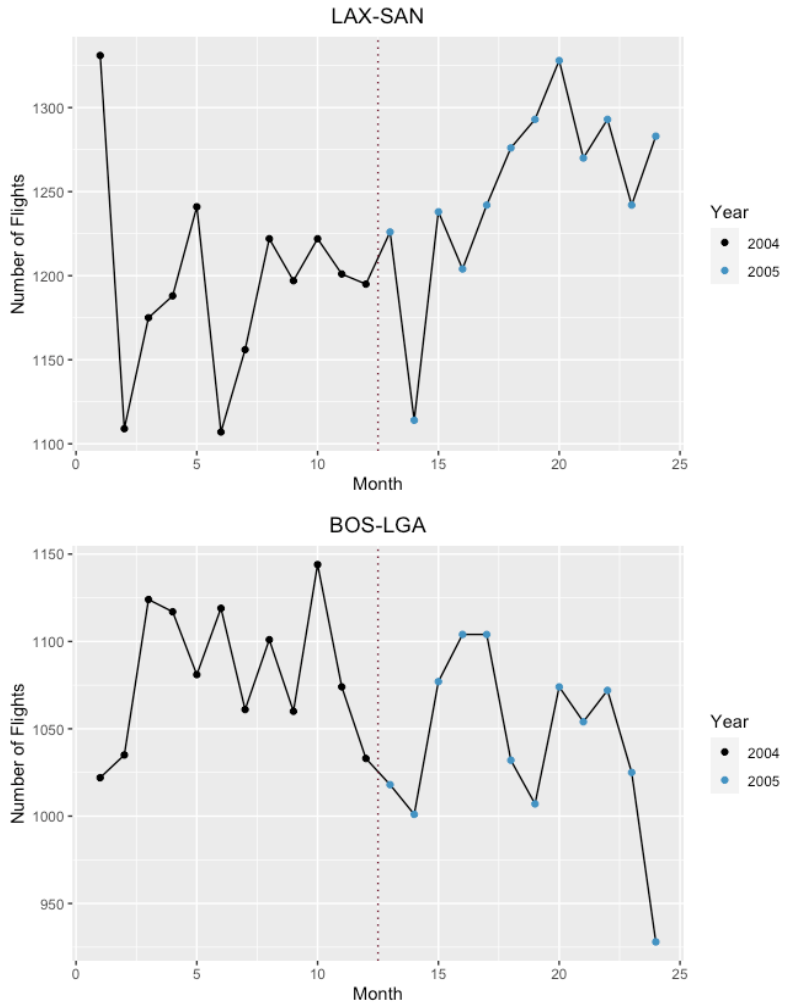
This question is also pretty straightforward. We just filter the cancelled flights, then we group by *Year* and *Month*. Then we count the number of flights in each group of *Year* and *Month*. Since we do not have the data for the exact number of passengers, using the number of flights can be used as an approximation.



The code results in the above chart. As we can observe, the number of passengers follows a general pattern over the whole year, alternating between an increase and decrease, with a general trend of increasing to a peak at the 8th month, then decreasing.

To investigate the trend of passengers *between destinations*, the code required remains relatively the same, but in the group by stage, we include *Origin* and *Dest* (Destination) as well. In order to parse out individual routes, simply use the filter() function to filter out rows with the desired *Origin* and *Dest*.

As an example, these are the number of passengers over time for two routes, "LAX-SAN" and "BOS-LGA".



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

For the fourth question, we query the 2004 and 2005 data sets and take the following attributes: *Year*, *Month*, *DayOfMonth*, *CRSDepTime*, *TailNum*, *ArrDelay*. We then do standard procedure to filter any NAs and set *CRSDepTime* to 24-hours. Then we group by Year, Month, DayofMonth, TailNum. This means each group is a flight log of each plane for each day over the span of two years.

In R, we just sort each group based on *CRSDepTime* and make a list of the ArrDelays, classified into whether the flight is delayed or not, using predefined functions from one of the R packages used extensively already, *dplyr*.

In Python, the listing process is a bit more complicated. We have to first make both the *CRSDepTime* and *ArrDelay* lists. Resulting in lists similar to this:

[[1700, 715, 1330], [1230, 1830]]

[[-5, -29, -1], [-9, 7]]

Then we use the zip() function to combine them like so:

[([1700, 715, 1330], [-5, -29, -1]), ([1230, 1830], [-9, 7])]

Then for each element in this joined list, we zip the two lists inside it together and sort by *CRSDepTime* as well:

[([715, -29], [1330, -1], [1700, -5]), ([1230, -9], [1830, 7])

We then extract the *ArrDelay*s into their own list and again classify if they are delayed or not.

After that, we check each list to see if two delays come one after another. If such an event occurs, it is an indication of a cascade.

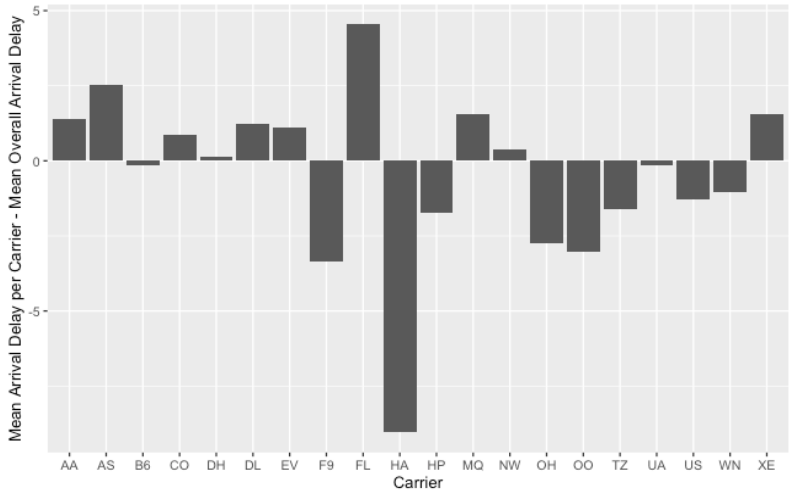
Some example rows are as follows:

Year	Month	DayofMonth	TailNum	ArrDelay	Cascade
2004	1	1	N102UW	c(FALSE, TRUE, FALSE)	FALSE
2004	1	1	N103	c(FALSE, FALSE, TRUE, TRUE, TRUE, FALSE, FALSE, TRUE)	TRUE
2004	1	1	N103UW	c(TRUE, TRUE, FALSE)	TRUE

Use the available variables to construct a model that predicts delays

In order to build this model, we will use variables we previously created in the previous sections. From question one, we take the *DayOfWeek*, *Month* and *Hour*. From question two, we take the *Age*, which is a transformed version of *Year* where we subtract 2005 by the *Year*.

Additionally we will add *UniqueCarriers* as a variable. We can see that for each carrier, there is a deviation away from the mean delays, indicating some explanatory power.



Certain variables such as *DepDelay* (Departure delay.), *ActualElapsedTime* will not be used in this model. Regardless of their correlation with *ArrDelay*, since in order to know them the flight must have already happened, such variables cannot be used for forecasting.

We then combine all these results into one table, which is our modelling data set. The type of model we will apply first is a simple linear model. The target variable of course being *ArrDelay* and the regressors being *Month*, *DayOfWeek*, *Hour*, *Age*, and *UniqueCarrier*. Any missing values will be imputed with the mean value.

To start model construction, we first subset the data to only 10 million randomly chosen rows. This is because on some computers, using the full data set would lead to an error.

Then, we split the data into training and test sets. The training set will be exclusively used for training and the test set will only be used for evaluation purposes at the end. The training set will be 70% of the data set, with the remaining being the test set.

Using the training set, we fit the model, with the variables we have determined above. Due to some of the variables being categorical, the final model will actually involve k-1 dummy variables for those variables where k is the number of categories. (e.g. Hour will end up as 23 dummy variables.)

To measure the performance of the model, we use the Root Mean Squared Error (RMSE), which is the average square root of the squared deviations from the mean. We will measure the RMSEs from both predictions based on the training set (Training Error) and based on the test set (Test Error). The RMSE values obtained are:

Type of Error	R	Python
Training Error	32.98736	33.21878985270934
Test Error	33.20547	33.16038283318835

Between R and Python, there are slight differences as when splitting the train and test sets due to the random nature of the splitting.

The low difference between the training and test error indicates that the model is not overfitted to the training set and may perform similarly when given an unknown data set.

Now, we want to try another model to see if there is an improvement. We will try to use random forests.

For this model, we subset the data again, but this time to only 100 thousand rows as this process is more computationally expensive. A key parameter we set here is the number of trees, which is set to 100, which was chosen because the results produced were similar to models with higher numbers of trees, but the fitting process took less time.

Otherwise, the process is similar to the process with linear regression.

The RMSE values obtained are:

Type of Error	R	Python
Training Error	32.49153	34.15018580925814
Test Error	30.07582	32.462770557068644

The values here differ again, due to the randomness of splitting the dataset and subsetting the data. Moreover, the “Ranger” learner in R and the RandomForestRegressor() function in Python have different parameters and different default settings. Here, we tried our best to make them as close as possible.

So are these models an improvement? It depends on the version. With Python, the random forest performs slightly worse but in R, it performs slightly better. Both models are slightly overfitted to the training set, but not substantially so as the difference between training and test error is not very large.

Moreover, one needs to consider other factors like computing power (as random forests take more time to fit) and their tolerance for error. The random forests model has the potential to perform better, but for some the additional time needed to develop the model may not outweigh the decrease in RMSE and they may go with the simpler, faster linear regression model even though the error is larger.