ST2195 Programming for Data Science Coursework Report

Name : Jessica Lawrence Gunawan
Student Number : 210421794

Table of Contents

I. Introduction

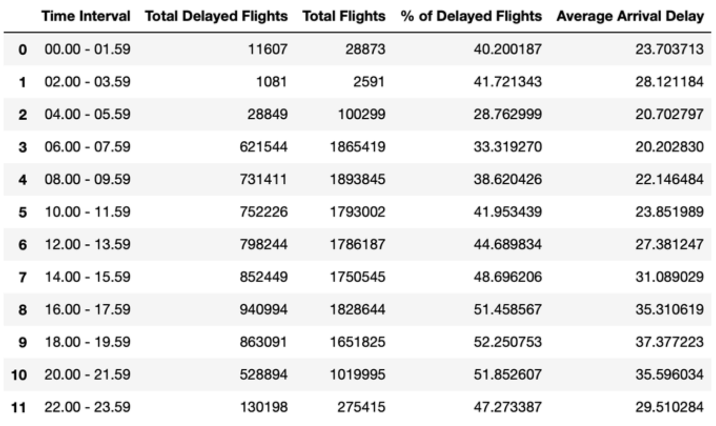
The main purpose of this report is to analyze The 2009 ASA Statistical Computing and Graphics Data Expo from Harvard Dataverse and provide answers to all five questions given. Here, both R and Python are used as the programming languages that will facilitate the analysis of the dataset. We chose two consecutive years of flight data, which are the year 2005 and 2006. Also, there are three additional data, which are airports, carriers, and plane-data, to complete the analysis.

II. Answering Questions

To start the analysis, we must import and load all the required Python packages and R libraries, such as *Pandas* and *Matplotlib* for Python & *Dplyr* and *Ggplot2* for R. Subsequently, we load all the datasets needed and create a new table called "flights", which is a combination of flight data from 2005 and 2006. We then look at the general information of the flights data, we can see what each variable's data type is as well as the names of all the columns. One of the most important steps is to check whether there are null values (missing values) inside the data. Any rows with null values in columns that the analysis requires will need to be filtered out. After completing the preparation steps, we may begin to answer the questions. All the complete packages or libraries, codes, tables, and graph visualizations are provided in the RMarkdown (for R) and Jupyter Notebooks (for Python) of each question.

A. Question 1: When is the best time of day, day of the week, and time of year to fly to minimise delays?

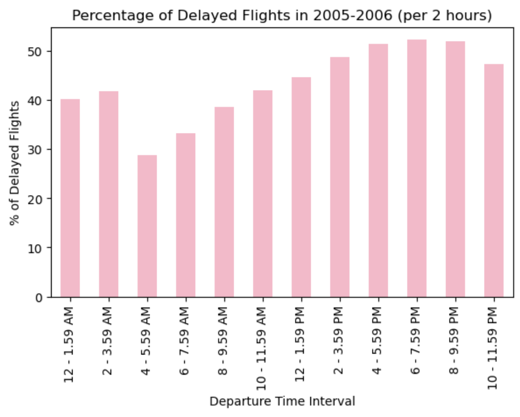
To answer this question, we will assume that delays are referred to as "ArrDelay" (Arrival Delays), as arrival time is what most people are concerned about. There can be a departure delay, but the plane still arrives at its destination on time. As a result, delays will be considered when "ArrDelay" > 0, since negative values indicate an earlier arrival than anticipated. We will separate the answers to this question into three parts, which are the best time of day, the best day of the week, and the best time of year. In order to identify the times or days with the fewest delays, we will analyze the percentage of delayed flights and the average delay duration. A lower percentage of delayed flights indicates a better time to fly, as it signifies fewer delays occurring. This goes the same with average delay, which denotes how long a delay is on average for a specific time.

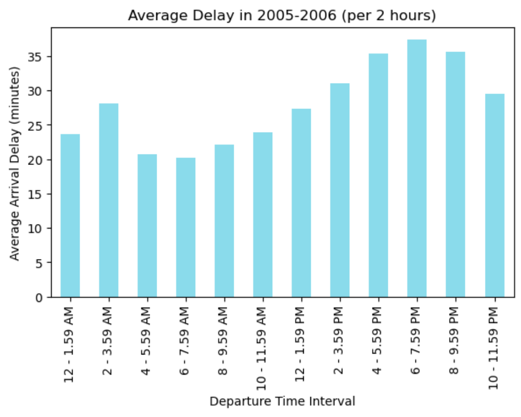


	Time Interval	Total Delayed Flights	Total Flights	% of Delayed Flights	Average Arrival Delay
0	00.00 - 01.59	11607	28873	40.200187	23.703713
1	02.00 - 03.59	1081	2591	41.721343	28.121184
2	04.00 - 05.59	28849	100299	28.762999	20.702797
3	06.00 - 07.59	621544	1865419	33.319270	20.202830
4	08.00 - 09.59	731411	1893845	38.620426	22.146484
5	10.00 - 11.59	752226	1793002	41.953439	23.851989
6	12.00 - 13.59	798244	1786187	44.689834	27.381247
7	14.00 - 15.59	852449	1750545	48.696206	31.089029
8	16.00 - 17.59	940994	1828644	51.458567	35.310619
9	18.00 - 19.59	863091	1651825	52.250753	37.377223
10	20.00 - 21.59	528894	1019995	51.852607	35.596034
11	22.00 - 23.59	130198	275415	47.273387	29.510284

In the best time of day, we group the departure time into 2-hour intervals within a 24-hour period and create a new column in the flights dataset called "deptime_interval". Afterwards, we calculate the percentage of delayed flights for each departure time interval by dividing the number of flights where "ArrDelay" > 0 by the total number of flights in the interval and multiplying by 100. To find the average delay time for each interval, we filter the flights dataset for those with an arrival delay greater than 0, group them by "deptime_interval," and aggregate the mean arrival delay. Finally, we combine this information using the merge() function in Python or left_join() in R to create a new table called "best_time_of_day".

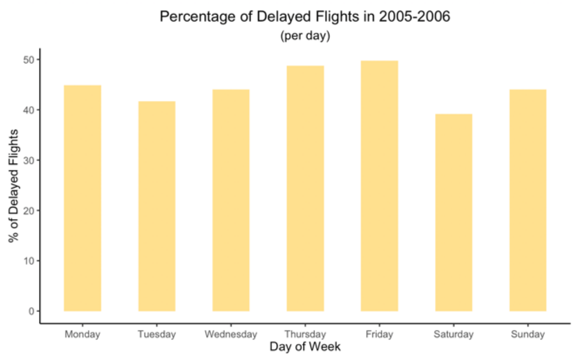
Then, we plot bar graphs to visualize the percentage of delayed flights and average arrival delay using Matplotlib for Python and Ggplot2 for R.

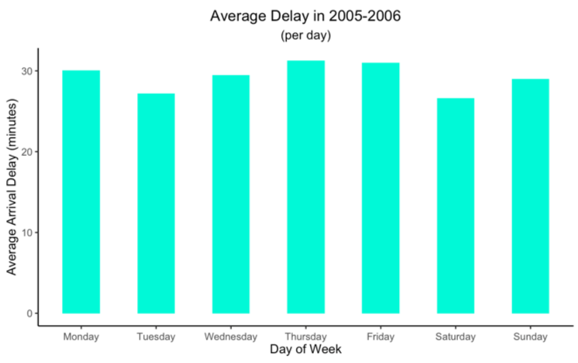




From the table and bar graphs above, we can see that the interval between 4 AM and 5.59 AM has the lowest percentage of delayed flights and the shortest average arrival delay, followed by the interval between 6 AM and 7.59 AM. Conversely, we observe that departure times after noon (12 PM) have higher percentages of delayed flights and average arrival delay. Therefore, it is recommended to book flights in the morning to minimize the risk of delays.

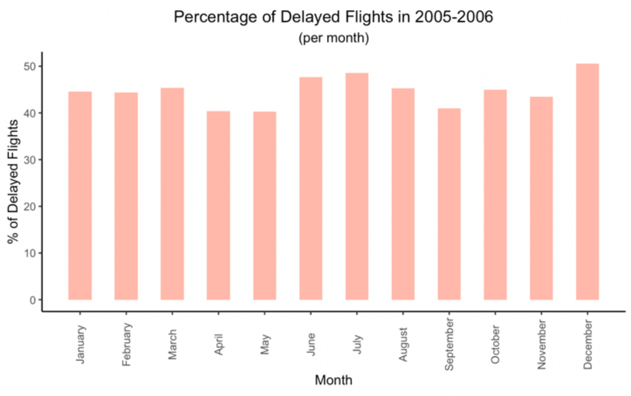
In the best day of the week, we perform similar calculations for the percentage of delayed flights and average arrival delay, but group the data by the seven days of the week (1 for Monday, 2 for Tuesday, and so on).

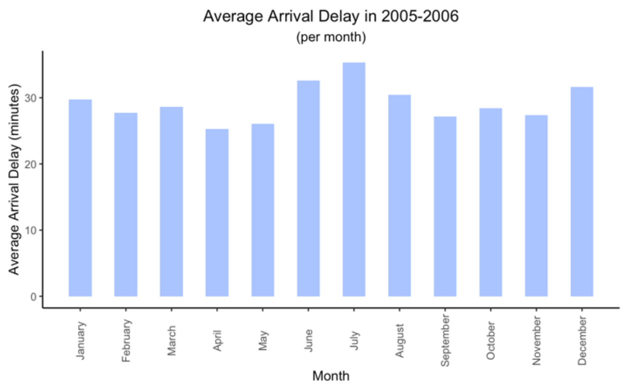




Although there are no significant differences in the percentage of delayed flights and average delay between days, we can still conclude that Saturday is the best day to fly to minimize delays. This is because on Saturday, the percentage of delayed flights is at its lowest (39.16%) and the average arrival delay is the shortest (26.6 minutes).

In the best time of year, we group the data by "Month" and observe the percentage of delayed flights as well as the average arrival delay for each month (1 for January, 2 for February, and so on).





Based on the graph, April and May are the best months to fly, with low numbers of delay occurrences and low average delays. In contrast, June, July, and December are the months with the highest number of delays and longest average delays, which may be due to the high volume of flights during the holiday season (summer and end-of-year holidays).

B. Question 2: Do older planes suffer more delays?

This question required both flights and plane_data datasets. To begin with, there are several assumptions made in order to answer this question. The assumptions are:

- 'Year' column in flights dataset refers to be the year that the plane was used to fly by passengers

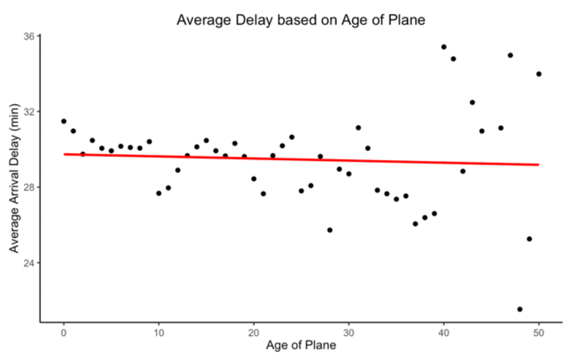
- 'year' column in plane_data dataset refers to be the year that the plane was manufactured

Therefore, the age of plane is obtained by subtracting the values in the 'Year' column of flight dataset with the values in the 'year' column in plane_data dataset. However, based on the assumptions we made earlier, there are some cases where the age appears to be negative, which does not make sense. Therefore, we will exclude these cases from our analysis.

Moreover, as previously mentioned, we define delays as arrival delays, which occur when "ArrDelay" is greater than 0. We will also consider both the percentage of delayed flights and the average delay to answer this question.

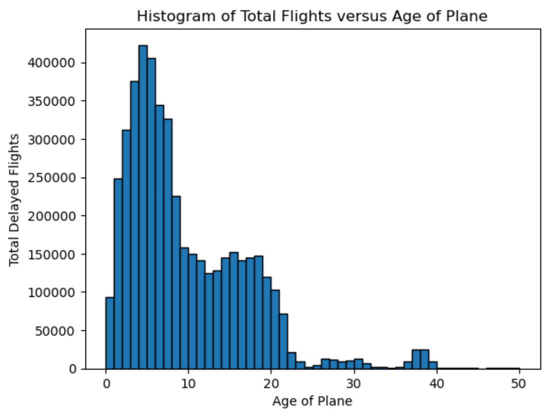
Before merging the plane_data and flights datasets, we first examine the summary of the plane_data dataset. We use the merge() function in Python and the left_join() function in R to merge both datasets based on the "TailNum" column, creating a new data frame called plane_delay. Next, we filter out all invalid values in the "year" column and convert the remaining values to integers. We need the "year" values to be integers so that we can calculate the age of the plane, as this operation only allows subtraction between two integers, not between an integer and a string. We then subtract the values in the "Year" column from the values in the "year" column to create a new column called "plane_age".

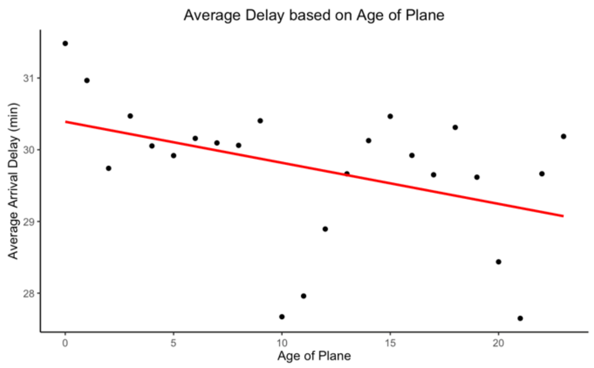
From the table we created, the age of the plane ranges from -2 to 50 years. However, based on the assumptions made earlier, negative age values do not make sense and will be filtered out of the analysis.



To investigate the relationship between plane age and average delay, we create a scatter plot using the plane_delay data frame. First, we filter the data to include only rows where the plane age was between 0 and 50, and where the arrival delay is greater than zero. We then group the data by plane age, calculating the mean delay and number of flights for each age. The resulting scatter plot shows a downward sloping line of best fit, suggesting that older planes tend to have lower average delays.

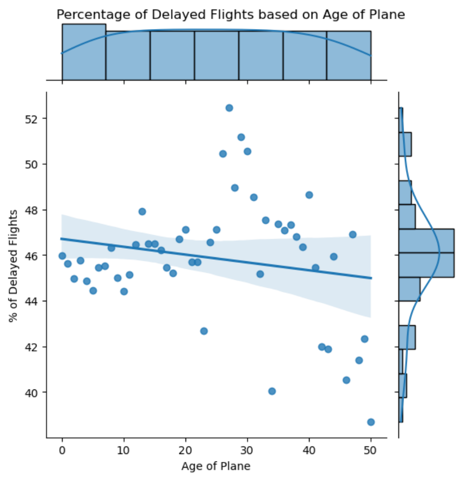
However, we also recognize that younger planes have larger sample size than old planes. To address this, we create a histogram showing the distribution of delayed flights for each age of plane. This confirms that planes with ages between 0 and 23 have much larger sample sizes than older planes. Thus, we plot another scatter plot where the age of plane is between 0 and 23 to obtain a more accurate estimation.



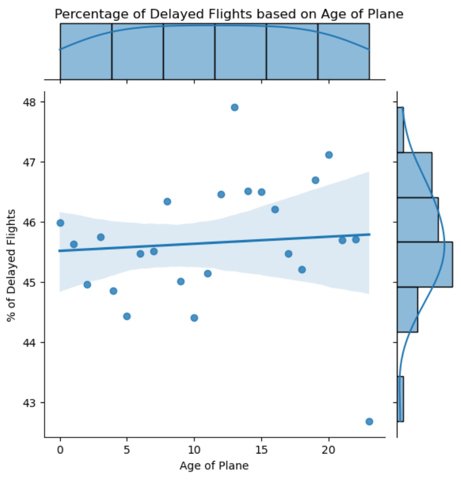


From the scatter plot, we can see that based on average delay, older planes do not suffer more delays. In fact, it has lower average delays than younger planes.

For the percentage of delayed flights, we calculate it by dividing the number of delayed flights with the total number flights then multiply by 100. We do the exact order as before, which are plotting scatter plots for age of plane between 0 and 50 as well as between 0 and 23.



Scatter Plot of Percentage of Delayed Flights based on Age of Plane Between 0 and 50

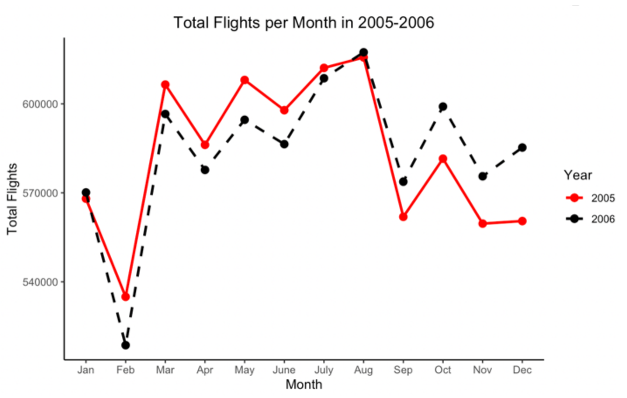


Scatter Plot of Percentage of Delayed Flights based on Age of Plane Between 0 and 23

On the right graph, the line of best fit seems to have an upward slope indicating that older planes suffer more delays. The evidence shows a slight increase in delays as planes get older, but the effect is not significant enough to draw a conclusive result. Therefore, the answer to this question is no, older planes do not suffer more delays.

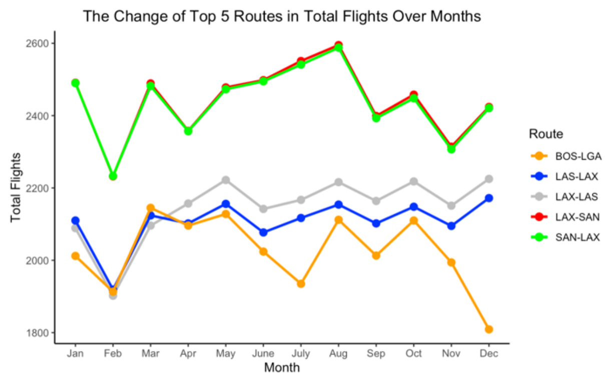
C. Question 3: How does the number of people flying between different locations change over time?

Since we lack information about the exact number of passengers per flight, we will assume a constant number of passengers for each flight. Therefore, we will count the total number of flights for different routes to analyse this question. For the "route" column, we combine "Origin" + "-" + "Dest" from the flights dataset. Our analysis begins with an overview of how the total flights for each year change over months. We will create two tables, one for each year, by filtering the flights dataset by year, grouping by month and year, and counting the total flights. To better visualize these changes, we will generate two line graphs for the years 2005 and 2006.



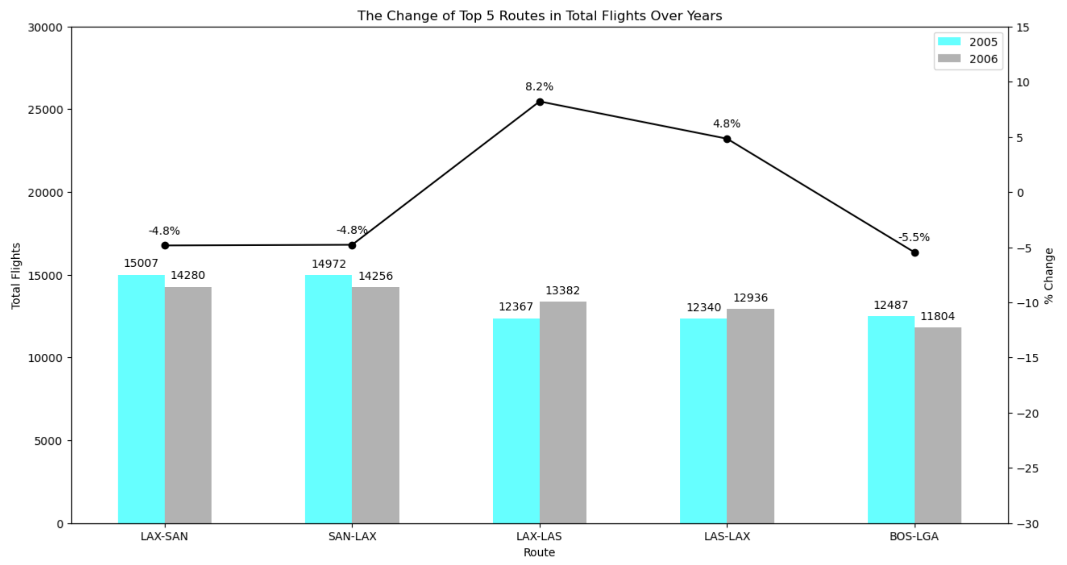
Upon analyzing the graph, it is evident that the patterns of both line graphs are identical, indicating the presence of a seasonal effect. The lowest total number of flights occurred in February for both years. The graphs fluctuate from March to July until they reach their peak in August. Furthermore, it appears that in the first half of the year, there were more flights in 2005 compared to 2006. However, towards the end of 2006, the flights were more crowded than they were in 2005.

To investigate the trend of the number of flights between destinations, we will use the "route" column. The analysis will be separated into two parts, analysis by month and analysis by year. In the analysis by month, we calculate the total flights for each route and sort it in descending order based on the total number of flights. We will focus on the top 5 routes with the highest number of total flights: LAX-SAN, SAN-LAX, LAX-LAS, LAS-LAX, and BOS-LGA. For each route, we create a table consisting of the total flights for each month. Then, we combine all the information and visualize it through five lines on a graph using Matplotlib in Python or ggplot2 in R.



Generally, a route and its reverse direction tend to have similar patterns, such as LAX-SAN and SAN-LAX, or LAX-LAS and LAS-LAX. The total flights for these routes fluctuate moderately throughout the year, as indicated by the relatively small range of change in total flights from January to December. However, for the BOS-LGA route, there is a significant decrease in total flights from October to December.

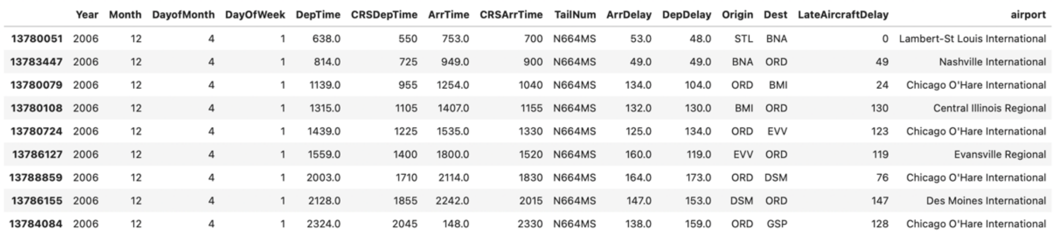
Regarding analysis by year, we are interested in how the amount of travellers between different areas has changed between 2005 and 2006. We grouped the flights dataset by "Year" and "route" for each year, creating two new tables with additional calculation of Total flights of each route. Afterwards, we inner join both tables and determine the difference between the total flights in 2005 and total flights in 2006. Not only that, we also calculate the percentage change by dividing the difference with total flights in 2005 and multiply it by 100. We compute the percentage change since total flights of each route may vary greatly. Negative value in percentage change indicates that there is a decrease in total flights from 2005 to 2006. Just like what we did previously in analysis by month, we select top 5 routes with highest total flights in both years to analyse. Here, we generate a side-by-side bar graph for each route, where the cyan bar represents the total flights in 2005 and the grey bar represents the total flights in 2006. In R, we need to melt the data first using the melt() function before creating the bar graph. We also include a line graph which refers to the percentage change of total flights of each route.



It appears that there is no discernible change in the overall number of flights between 2005 and 2006 on these 5 routes. Only a 0 to 10% change is shown. Also, it should be mentioned that for a route and its reverse direction, such as LAX-SAN and SAN-LAX or LAX-LAS and LAS-LAX, both routes exhibited the same changes, either both increasing or both decreasing.

D. Question 4: Can you detect cascading failures as delays in one airport create delays in others?

The airports dataset will be used to answer this question. To investigate the occurrence of cascading failures, we will examine whether a delay in the departure time of a plane results in subsequent delays for the plane's next schedules. We will consider it proof of cascading failures if the following flight's schedule is delayed as a result of the prior flight's late arrival, as indicated by the "LateAircraftDelay" variable. We will collect three samples to aid in the analysis. It is important to point out that the "LateAircraftDelay" variable measures the length of time that a flight is delayed as a result of using the same aircraft as the earlier, delayed trip. Initially, we left join the flights dataset and airports dataset by "Origin" on flights equals to "iata" on airports creating a new table "cf". We query only necessary columns such as "Year", "Month", "DayofMonth", "DayOfWeek", "DepTime", "CRSDepTime", "ArrTime", "CRSArrTime", "TailNum", "ArrDelay", "DepDelay", "Origin", "Dest", "LateAircraftDelay", and "airport" that will be needed throughout the process. The following step is to select rows where "LateAircraftDelay" is larger than 0 but less than 120 minutes. This is because a larger "LateAircraftDelay" might lead to cancellation of the next plane's schedule. Then, we take 3 samples randomly after the table has been sorted by "LateAircraftDelay" in descending order. The table below is one of the samples taken.

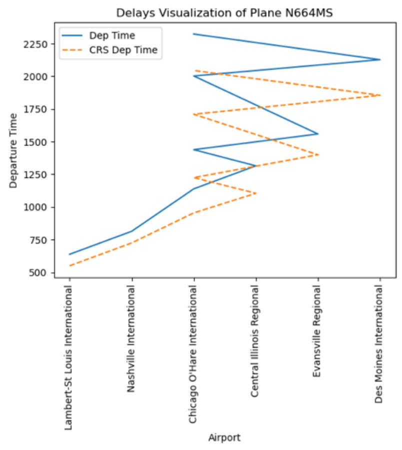


	Year	Month	DayofMonth	DayOfWeek	DepTime	CRSDepTime	ArrTime	CRSArrTime	TailNum	ArrDelay	DepDelay	Origin	Dest	LateAircraftDelay	airport
13780051	2006	12	4	1	638.0	550	753.0	700	N664MS	53.0	48.0	STL	BNA	0	Lambert-St Louis International
13783447	2006	12	4	1	814.0	725	949.0	900	N664MS	49.0	49.0	BNA	ORD	49	Nashville International
13780079	2006	12	4	1	1139.0	955	1254.0	1040	N664MS	134.0	104.0	ORD	BMI	24	Chicago O'Hare International
13780108	2006	12	4	1	1315.0	1105	1407.0	1155	N664MS	132.0	130.0	BMI	ORD	130	Central Illinois Regional
13780724	2006	12	4	1	1439.0	1225	1535.0	1330	N664MS	125.0	134.0	ORD	EVV	123	Chicago O'Hare International
13786127	2006	12	4	1	1559.0	1400	1800.0	1520	N664MS	160.0	119.0	EVV	ORD	119	Evansville Regional
13788859	2006	12	4	1	2003.0	1710	2114.0	1830	N664MS	164.0	173.0	ORD	DSM	76	Chicago O'Hare International
13786155	2006	12	4	1	2128.0	1855	2242.0	2015	N664MS	147.0	153.0	DSM	ORD	147	Des Moines International
13784084	2006	12	4	1	2324.0	2045	148.0	2330	N664MS	138.0	159.0	ORD	GSP	128	Chicago O'Hare International

Flight Table of a Plane with Tail Number N664MS

Looking at the table above, the first scheduled departure time of the plane was at 5.50 AM. However, the plane departed late at 6.38 AM from Lambert-St Louis International Airport which caused a forty eight minutes departure delay. Then, it arrived late at Nashville International at 7.53 AM. This 53 minutes arrival delay results in another departure delay in Nashville International Airport where the plane's next schedule was supposed to depart at 7.25 AM. Afterwards, it arrived late again at Chicago O'Hare International Airport at 9.49 AM which cause another 104 minutes departure delay for the next schedule. This continuity in delays happened for the plane's seven subsequent schedules.

It is proven from the line graph visualization that cascading failures do occur. A delay in the first departure time resulted in a continuing departure delay in another airport for the plane's next few schedules, as can be seen from the actual departure time (Dep Time) line graph being above the scheduled departure time (CRS Dep Time) line graph.



E. Question 5: Use the available variables to construct a model that predicts delays.

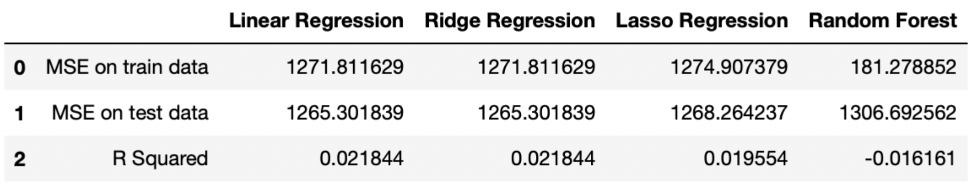
Firstly, we import necessary packages to answer this question. We use the scikit-learn package in Python and the mlr3 package in R. The same as previously, we will construct a model to forecast arrival delay ("ArrDelay") based on the assumption that the delay in question is the arrival delay. We will build 4 models: the Linear Regression Model, the Ridge Regression Model, the Lasso Regression Model, and the Random Forest Model (using samples). For the random forest model, we will only utilize samples because using a huge dataset might lead to errors.

We need to choose variables which are going to be the regressors with "ArrDelay" as the target variable. We have to choose those variables that are available before the flights happen. This is so that the model can be used in practice for predicting future delays. The chosen variables are "CRSDepTime", "CRSArrTime", "CRSElapsedTime", "Year", "Month", "DayofMonth", "DayOfWeek", and "plane_age". The "plane_age" variable used is the same as the one we used before to answer question 2 where we left join flights data and plane_data data and we subtract the values in 'Year' column with the values in 'year' column. Also, we only want rows where the age of plane is greater than 0.

In Python, we need to specify the explanatory variables and the target variable. Then, we have to split them into train set and test set with 70% being the train set and the remaining 30% being the test set. Since all the features we chose are numerical features, we apply the SimpleImputer() function to do imputation on the missing values and apply the StandardScaler() function for scaling. Next we build a pipeline for each prediction model and fit in into the model. Set a parameter grid to specify a grid of hyperparameters to search over during the tuning process for each pipeline. Afterwards, we perform a grid search using the GridSearchCV function.

In R, we select all those variables that we are going to use for the prediction. We create a task and specify “ArrDelay” as the target variable. Then, we set MSE (Mean Squared Error) as the measurement tool. To ensure the model is reproducible, we use set.seed() function. Same as in Python, we split the data into training set and test set with a percentage of 70% and 30%, respectively. Afterwards, using the 'regr.glmnet' method, we generate learner objects for each prediction model, setting the alpha to 0 for ridge regression and 1 for lasso regression. We also do imputation and scaling using po(‘imputmean’) and po(‘scale’) functions respectively. For ridge regression, lasso regression, and random forest, we need to set up the tuning environment before we do hyperparameter tuning for each model.

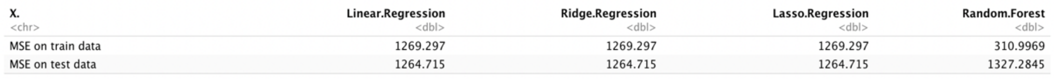
Lastly, to evaluate the performance of the four prediction models, both in Python and R, we use Mean Squared Error as well as R squared value. Lower MSE values indicate better model performance, as the model is making fewer errors in its prediction. On the other hand, the better the model fits the data, the nearer the R-squared value is to 1.



		Linear Regression	Ridge Regression	Lasso Regression	Random Forest
0	MSE on train data	1271.811629	1271.811629	1274.907379	181.278852
1	MSE on test data	1265.301839	1265.301839	1268.264237	1306.692562
2	R Squared	0.021844	0.021844	0.019554	-0.016161

In Python

The models with the lowest MSE for Python are linear regression and ridge regression, both of which have a value of 1265.301839. They also have a higher value in R squared, supporting the claim that both models are superior to other models in terms of their ability to forecast delays. For Random Forest model, it seems that there is a big difference between the MSE on train data and MSE on test data. This tells us that the data is overfitting to the train data.



X. <chr>	Linear.Regression <dbl>	Ridge.Regression <dbl>	Lasso.Regression <dbl>	Random.Forest <dbl>
MSE on train data	1269.297	1269.297	1269.297	310.9969
MSE on test data	1264.715	1264.715	1264.715	1327.2845

In R

It appears that the MSE values for the Linear, Ridge, and Lasso Regression models in R are all 1264.715. Moreover, its value is less than Random Forest’s.

In conclusion, Linear Regression and Ridge Regression models are the best models to forecast delays based on the MSE values for both R and Python.

III. References

ST2195 Subject Guide

Kaggle