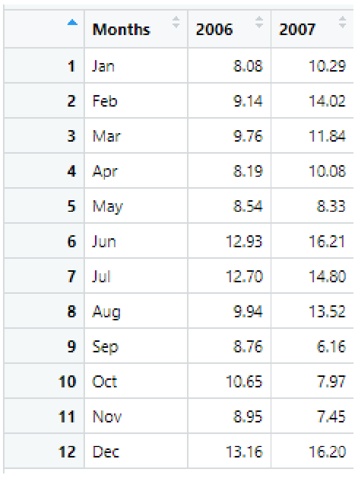
ST2195 PROGRAMMING FOR DATA SCIENCE COURSEWORK

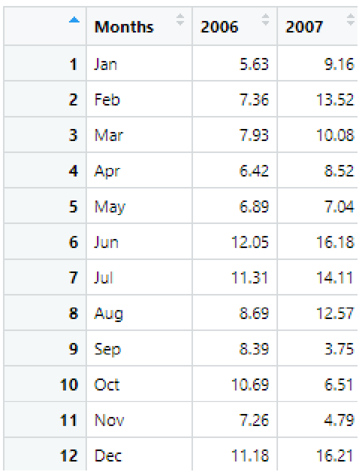
1. When is the best time of day, day of the week, and time of year to fly to minimise delays?

R

Best time of year:
To find out which time of the year is best to fly for minimization of delays, I took subsets of the dataset by months and took averages of the departure and arrival delays, and compared each month for both years 2006 and 2007.



	Months	2006	2007
1	Jan	8.08	10.29
2	Feb	9.14	14.02
3	Mar	9.76	11.84
4	Apr	8.19	10.08
5	May	8.54	8.33
6	Jun	12.93	16.21
7	Jul	12.70	14.80
8	Aug	9.94	13.52
9	Sep	8.76	6.16
10	Oct	10.65	7.97
11	Nov	8.95	7.45
12	Dec	13.16	16.20

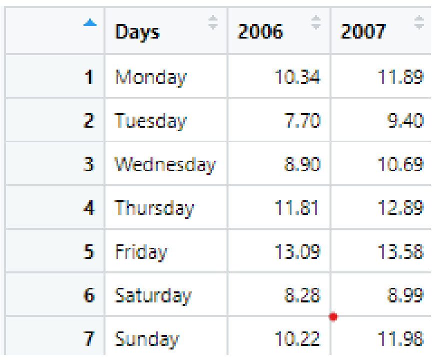


	Months	2006	2007
1	Jan	5.63	9.16
2	Feb	7.36	13.52
3	Mar	7.93	10.08
4	Apr	6.42	8.52
5	May	6.89	7.04
6	Jun	12.05	16.18
7	Jul	11.31	14.11
8	Aug	8.69	12.57
9	Sep	8.39	3.75
10	Oct	10.69	6.51
11	Nov	7.26	4.79
12	Dec	11.18	16.21

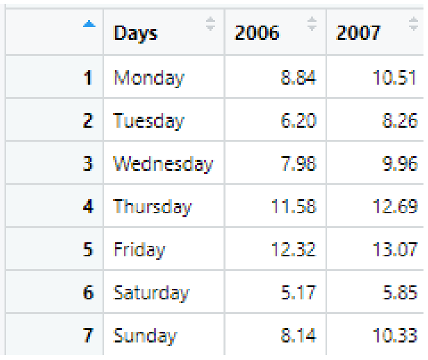
Figure 1, 2: Average Departure and Arrival delay respectively, at monthly intervals

Thus, according to the figures above, the month with the lowest average delay is September. September is hence the best time of year to travel which minimises delays.

Best day of week:
We use the subset function to take a subset of the data by the days for both 2006 and 2007 dataset so that we are able to take the average of the delays by days of the week.
Taking the average of both Departure and Arrival delays by the days Monday to Sunday allows us to compare both delays respectively by days between 2006 and 2007. This shows us which day would have on average the highest and least delays, and thus when would be the best time of the week to travel which minimises delay.



	Days	2006	2007
1	Monday	10.34	11.89
2	Tuesday	7.70	9.40
3	Wednesday	8.90	10.69
4	Thursday	11.81	12.89
5	Friday	13.09	13.58
6	Saturday	8.28	8.99
7	Sunday	10.22	11.98



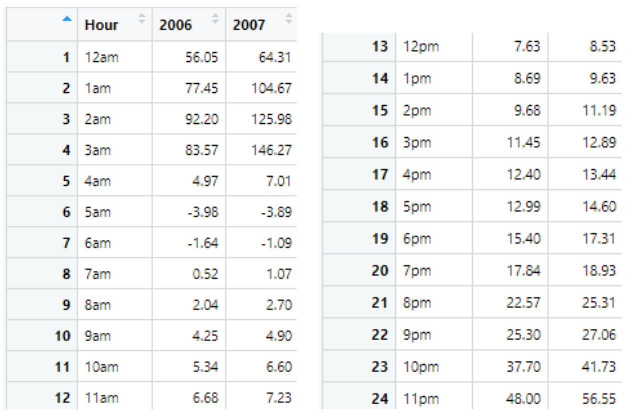
	Days	2006	2007
1	Monday	8.84	10.51
2	Tuesday	6.20	8.26
3	Wednesday	7.98	9.96
4	Thursday	11.58	12.69
5	Friday	12.32	13.07
6	Saturday	5.17	5.85
7	Sunday	8.14	10.33

Figure 3,4: Average Departure and Arrival delays respectively, at daily intervals

Hence, the best day to travel is Saturdays as both departure and arrival delay is the lowest on average. Furthermore, the difference between 2006 and 2007 is the least which means it is also more reliable as changes are not too large over a span of 1 year.

Best time of day to travel:

Using the subset function again, subsetting by hours using the actual departure time in 2006 and 2007, we can find the average departure delay at different time intervals in both years by looking at the hourly interval. As seen in previous cases, there is a clear positive relationship between the departure and arrival delay as when there is high departure delay, there is high arrival delay in the destination state as well. Thus, I only used averages of the departure delay to derive my answer.



	Hour	2006	2007
1	12am	56.05	64.31
2	1am	77.45	104.67
3	2am	92.20	125.98
4	3am	83.57	146.27
5	4am	4.97	7.01
6	5am	-3.98	-3.89
7	6am	-1.64	-1.09
8	7am	0.52	1.07
9	8am	2.04	2.70
10	9am	4.25	4.90
11	10am	5.34	6.60
12	11am	6.68	7.23
13	12pm	7.63	8.53
14	1pm	8.69	9.63
15	2pm	9.68	11.19
16	3pm	11.45	12.89
17	4pm	12.40	13.44
18	5pm	12.99	14.60
19	6pm	15.40	17.31
20	7pm	17.84	18.93
21	8pm	22.57	25.31
22	9pm	25.30	27.06
23	10pm	37.70	41.73
24	11pm	48.00	56.55

Figure 5,6: Average Departure and Arrival delays respectively, at hourly intervals

Thus, with just the hourly comparison of departure delay in both 2006 and 2007, we can clearly tell that the best timing of the day, or the best hour to travel with the least delays is within 5am, where there is in fact a negative average value for delay. A negative value means that with the number of flights happening at the 5am range till before 6am, there is an average of the flights being early instead of late.

Python

Best time of year:
Firstly, import and load the packages and required data into the Jupyter notebook. After importing the data, duplicates can be removed using drop_duplicates. The data was then filtered into subsets by months, where the average can then be taken by the months. This would show which time of the year has the lowest delays and thus the best time of year to fly. The best period to fly to minimise delays during the year would be in September as delays are at the lowest.

Best day of week:
Subsetting 2006 and 2007 by the days Monday to Sunday, we are able to find the average departure and arrival delay of the days in the years 2006 and 2007. The best day to travel with minimal delay is Saturday.

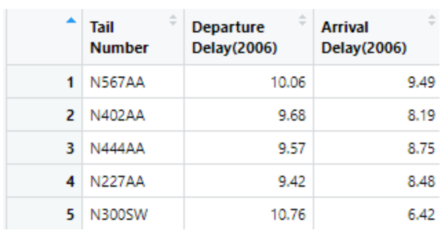
Best time of day:
Subsetting by the actual departure times hourly, we can compare the delays that occur at every hour interval. The best timing to depart where there is the least delay is the interval of 5am to before 6am. As there is a clear positive relationship between

departing and arriving,due to there being a clear indication that there is high delay in arrival whenever there is high departure delay, I only used departure delays to derive my answer as mentioned under the R portion of Question 1.

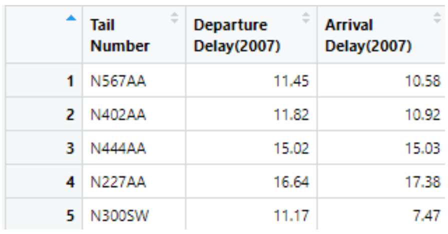
2. Do older planes suffer more delay?

R

Import plane data into the R global environment first (since the 2006 and 2007 data on flights is already inside) then drop useless data, which are rows in which the important columns are empty or are NA. Then filter the top 5 planes which were issued first, meaning the oldest 5 planes that were used. Filter also the top 5 planes with the latest issue date, ie. the 5 newest planes used in 2006 and 2007. We filter out flights with top 5 oldest and newest planes according to the tail number (number specific to every plane used) of planes in both 2006 and 2007, then find the average departure and arrival delay for both old and new planes. (note: there was no use for N843AL in 2006, thus the value is NA.

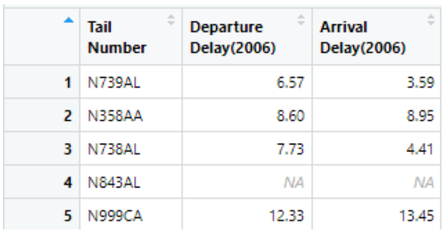


	Tail Number	Departure Delay(2006)	Arrival Delay(2006)
1	N567AA	10.06	9.49
2	N402AA	9.68	8.19
3	N444AA	9.57	8.75
4	N227AA	9.42	8.48
5	N300SW	10.76	6.42

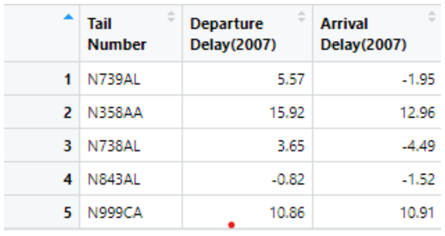


	Tail Number	Departure Delay(2007)	Arrival Delay(2007)
1	N567AA	11.45	10.58
2	N402AA	11.82	10.92
3	N444AA	15.02	15.03
4	N227AA	16.64	17.38
5	N300SW	11.17	7.47

Figure 7,8: Comparison of old planes in 2006 and 2007 respectively



	Tail Number	Departure Delay(2006)	Arrival Delay(2006)
1	N739AL	6.57	3.59
2	N358AA	8.60	8.95
3	N738AL	7.73	4.41
4	N843AL	NA	NA
5	N999CA	12.33	13.45



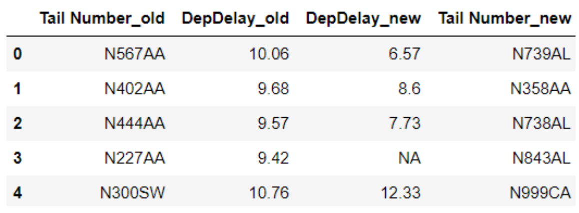
	Tail Number	Departure Delay(2007)	Arrival Delay(2007)
1	N739AL	5.57	-1.95
2	N358AA	15.92	12.96
3	N738AL	3.65	-4.49
4	N843AL	-0.82	-1.52
5	N999CA	10.86	10.91

Figure 8,9: Comparison of new planes in 2006 and 2007 respectively

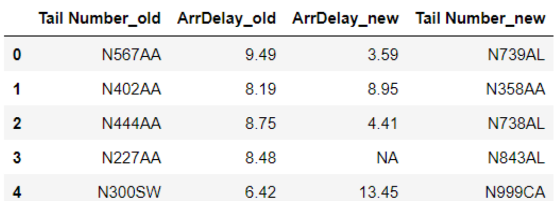
Even though there are some anomalies in the delays for newer planes, they do face lesser delays on average while the old planes have consistently faced higher levels of delay for both departure and arrival. Furthermore, new planes even have negative delays which indicates that they are early, and faster in general thus in comparison to old planes, old planes do suffer more delay.

Python

Import plane data and the 2006 and 2007 flights datasets, then filter out the top 5 oldest and newest planes according to the issue date. Then subset older and newer planes used in the 2006 and 2007 flights data. Then find the mean of departure and arrival delay for each plane in 2006 and 2007. We can then compare the delay between new and old planes like in figures 10 to 13 as shown below.

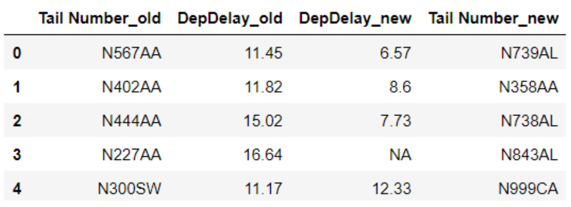


	Tail Number_old	DepDelay_old	DepDelay_new	Tail Number_new
0	N567AA	10.06	6.57	N739AL
1	N402AA	9.68	8.6	N358AA
2	N444AA	9.57	7.73	N738AL
3	N227AA	9.42	NA	N843AL
4	N300SW	10.76	12.33	N999CA

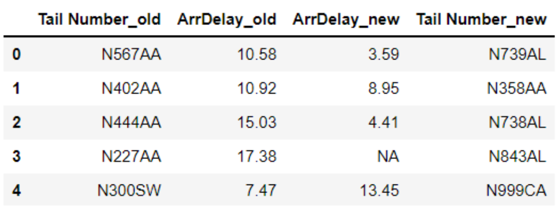


	Tail Number_old	ArrDelay_old	ArrDelay_new	Tail Number_new
0	N567AA	9.49	3.59	N739AL
1	N402AA	8.19	8.95	N358AA
2	N444AA	8.75	4.41	N738AL
3	N227AA	8.48	NA	N843AL
4	N300SW	6.42	13.45	N999CA

Figure 10,11: Comparison of delay in 2006



	Tail Number_old	DepDelay_old	DepDelay_new	Tail Number_new
0	N567AA	11.45	6.57	N739AL
1	N402AA	11.82	8.6	N358AA
2	N444AA	15.02	7.73	N738AL
3	N227AA	16.64	NA	N843AL
4	N300SW	11.17	12.33	N999CA



	Tail Number_old	ArrDelay_old	ArrDelay_new	Tail Number_new
0	N567AA	10.58	3.59	N739AL
1	N402AA	10.92	8.95	N358AA
2	N444AA	15.03	4.41	N738AL
3	N227AA	17.38	NA	N843AL
4	N300SW	7.47	13.45	N999CA

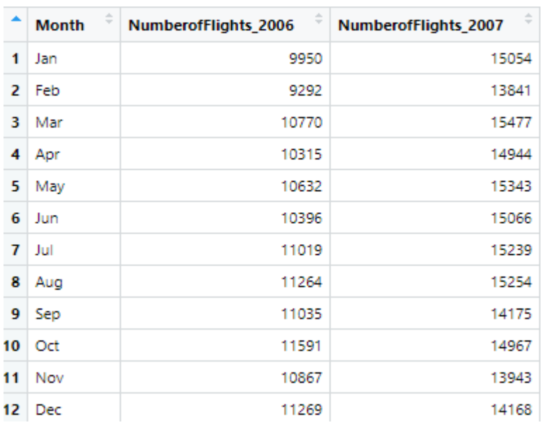
Figure 12,13: Comparison of delay in 2007 (error in DepDelay_new column, values are 5.57,15.92,3.65,-0.82,10.86. Another error in ArrDelay_new column, values are -1.95,12.96,-4.48,-1.52,10.91 instead)

Looking at the comparisons between old and new planes for each delay throughout both years, aside from a few anomalies, the older planes generally have a higher average delay for both departure and arrival. Thus older planes do in fact suffer more delay.

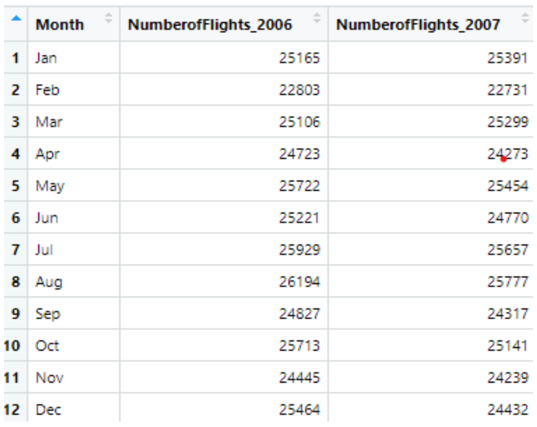
3. How does the number of people flying between different locations change over time?

R

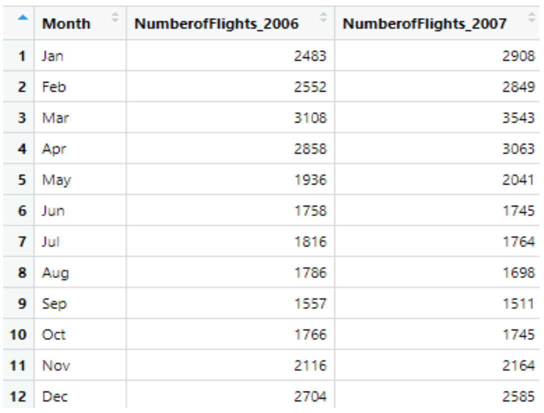
Since the question did not specify what locations are, we assume that the question is asking about the different states in the USA.
Firstly, we need to import the airport data into the global environment. Then load the necessary libraries and ensure that the flights dataset is working well by using summary() function. Then we randomly subset a few destination states for which I have chosen: DTW(Michigan), DFW(Texas) and RSW(Florida), and then subset them by the months again in both 2006 and 2007.
To gauge the number of people flying, we look at the number of flights recorded. We then compare the number of flights that arrived in the respective states to see if there is any increasing or decreasing trend from January to December in both years as well.



	Month	NumberofFlights_2006	NumberofFlights_2007
1	Jan	9950	15054
2	Feb	9292	13841
3	Mar	10770	15477
4	Apr	10315	14944
5	May	10632	15343
6	Jun	10396	15066
7	Jul	11019	15239
8	Aug	11264	15254
9	Sep	11035	14175
10	Oct	11591	14967
11	Nov	10867	13943
12	Dec	11269	14168



	Month	NumberofFlights_2006	NumberofFlights_2007
1	Jan	25165	25391
2	Feb	22803	22731
3	Mar	25106	25299
4	Apr	24723	24273
5	May	25722	25454
6	Jun	25221	24770
7	Jul	25929	25657
8	Aug	26194	25777
9	Sep	24827	24317
10	Oct	25713	25141
11	Nov	24445	24239
12	Dec	25464	24432



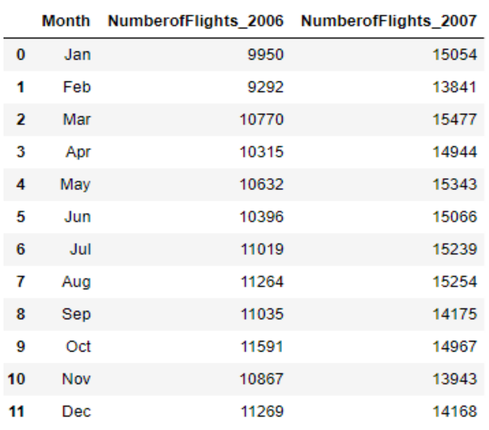
	Month	NumberofFlights_2006	NumberofFlights_2007
1	Jan	2483	2908
2	Feb	2552	2849
3	Mar	3108	3543
4	Apr	2858	3063
5	May	1936	2041
6	Jun	1758	1745
7	Jul	1816	1764
8	Aug	1786	1698
9	Sep	1557	1511
10	Oct	1766	1745
11	Nov	2116	2164
12	Dec	2704	2585

Figure 14(top left),15(top right),16(bottom left): Comparison of monthly number of flights arriving in Michigan, Texas and Florida respectively in 2006 and 2007

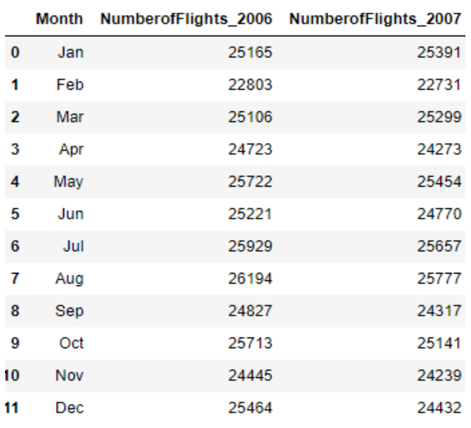
Thus, from the figures shown above, it is seen that there are more people visiting Michigan and Florida as time passes while there is a slight decrease in people visiting Texas comparing the number of flights in 2006 to 2007. However, there is an average increase in the number of people travelling. Thus there is an increased number of people flying between different locations over time.

Python

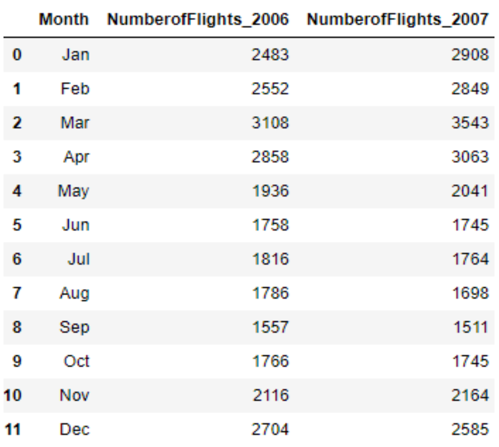
First import the necessary libraries and also the airport data, into Jupyter notebook. Load the airport and both years data with pandas read_csv function again. Since we have randomly chosen in R as shown above, we use the same three destination airport IATA codes DTW, DFW and RSW, Michigan, Texas and Florida respectively. We then subset each state in both years' data, and then we subset each state by months, so that we can see how many flights, also known as the number of people which are travelling to said states. Doing so, we can already see the total number of flights to the three states by the months and thus I just put the values in the tables as shown below in Figure 17 18 and 19. The figures compare the number of people flying between different locations to the three states and it is showing an overall increase seen in the increase both by month and from 2006 to 2007. Thus we can say that there is an increase in the number of people flying between different locations over time.



	Month	NumberofFlights_2006	NumberofFlights_2007
0	Jan	9950	15054
1	Feb	9292	13841
2	Mar	10770	15477
3	Apr	10315	14944
4	May	10632	15343
5	Jun	10396	15066
6	Jul	11019	15239
7	Aug	11264	15254
8	Sep	11035	14175
9	Oct	11591	14967
10	Nov	10867	13943
11	Dec	11269	14168



	Month	NumberofFlights_2006	NumberofFlights_2007
0	Jan	25165	25391
1	Feb	22803	22731
2	Mar	25106	25299
3	Apr	24723	24273
4	May	25722	25454
5	Jun	25221	24770
6	Jul	25929	25657
7	Aug	26194	25777
8	Sep	24827	24317
9	Oct	25713	25141
10	Nov	24445	24239
11	Dec	25464	24432



	Month	NumberofFlights_2006	NumberofFlights_2007
0	Jan	2483	2908
1	Feb	2552	2849
2	Mar	3108	3543
3	Apr	2858	3063
4	May	1936	2041
5	Jun	1758	1745
6	Jul	1816	1764
7	Aug	1786	1698
8	Sep	1557	1511
9	Oct	1766	1745
10	Nov	2116	2164
11	Dec	2704	2585

Figure 17(top left),18(top right),19(bottom left): Comparison of monthly number of flights arriving in Michigan, Texas and Florida respectively in 2006 and 2007

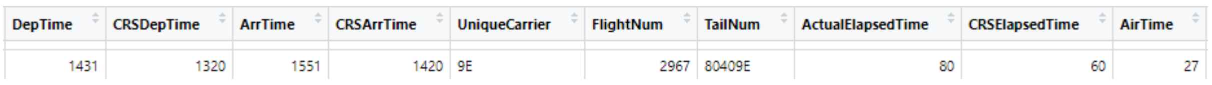
4. Can you detect cascading failures as delays in one airport create delays in others?

R

Load the libraries and ensure our datasets are working well using the summary() function. Then, arrange DepDelay timings in descending order, we can see that the maximum Arrival delay was also in line with the highest Departure delay timing. Thus now we just need to find delayed flights at a certain airport, then at the destination airport of the same flight, filter departing flights which have a significant delay. This will then show that any delay in arrival flights at a particular airport will lead to a departure delay out of that same airport. Then, we filter different airports to find the trend.
To find out if cascading failures can be detected, we need to find out if there are flights on the same day which have delayed arrival flights to a certain destination, and then at the same destination airport, if there are any departing flights delayed after the delayed arrival flight has landed.

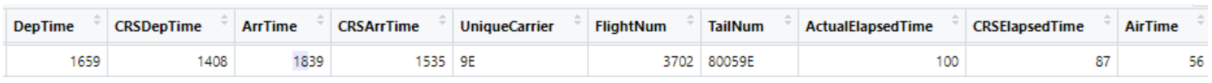
The data shown below indicates that a delay for a plane with tail number 80409E, arriving at DTW, was 91 minutes late when it was supposed to reach at 1420 hours. Then at the destination airport DTW, a departing flight was delayed as it was supposed to depart at 1408 hours, but was delayed and only left at 1659 hours.

1. 18.2.2007 80409E FWA—>DTW(dep:71 min, arr:91 min)



DepTime	CRSDepTime	ArrTime	CRSArrTime	UniqueCarrier	FlightNum	TailNum	ActualElapsedTime	CRSElapsedTime	AirTime
1431	1320	1551	1420	9E	2967	80409E	80	60	27

18.2.2007 80059E DTW—> MDT (dep:171, arr:184)

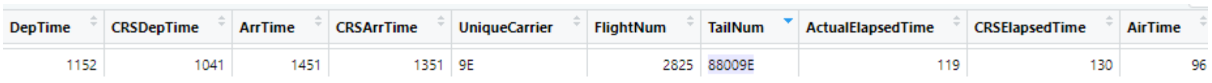


DepTime	CRSDepTime	ArrTime	CRSArrTime	UniqueCarrier	FlightNum	TailNum	ActualElapsedTime	CRSElapsedTime	AirTime
1659	1408	1839	1535	9E	3702	80059E	100	87	56

→ 80409E caused delay in 80059E

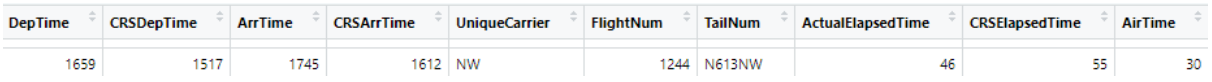
This is another example of the above events occurring.

2. 25.3.2007 88009E BHM—>DTW (dep:71 , arr:60)



DepTime	CRSDepTime	ArrTime	CRSArrTime	UniqueCarrier	FlightNum	TailNum	ActualElapsedTime	CRSElapsedTime	AirTime
1152	1041	1451	1351	9E	2825	88009E	119	130	96

25.3.2007 N613NW DTW—>CMH (dep: 102, arr:93



DepTime	CRSDepTime	ArrTime	CRSArrTime	UniqueCarrier	FlightNum	TailNum	ActualElapsedTime	CRSElapsedTime	AirTime
1659	1517	1745	1612	NW	1244	N613NW	46	55	30

As shown above, it is clear that cascading failure can be detected as seen by the arrival delays leading to the next departure delay in a certain airport.

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

R

We first import libraries and load the data needed. The data that we are using to build our predictive model has already been imported in Question 1. We now just need to make sure it is imported and working well, we use head() function for both 2006 and 2007 flights data. Once data is loaded properly, we need to clean the data and impute missing values in the dataset or remove the NA values which I have chosen to do. We then set up a task then choose a learner. To join together the above operations, mlr3 creates a pipe for each of these tasks, and joining it together creates a pipeline using mlr3pipelines which then can be turned into a combined learner. Now, we see the combined learner as a separate algorithm and we have to evaluate its performance now. We should now split the data into training and test sets where we can allocate 70% as the training set and 30% as the test set. While training the model, the test set will not be touched. We then select a measure, in which I selected mse. Now we train the model. We then tune the hyperparameters and we need to set up the tuning environment. We need to specify the hyperparameters and the range we want to use. We then put everything together into a new learner where now we train the model and evaluate the performance. We then compare different learners using the benchmark function. Set the benchmark design and run the comparisons. We can then visualise the comparisons with boxplots using autoplot() and we can then see the overall measure for each learner using bmr$aggregate(). (UOL VLE, 2022)

Python

Firstly, we import the libraries and load the data needed into Jupyter notebook. Then read the data using pandas read_csv().

We are in the pre-processing stage, where we can first start to build pipelines with a few pre-processing steps for both continuous and categorical variables in the datasets. Our numerical features that we need are both the Departure and Arrival delay timings, and we can impute–insertion of an estimated value– the missing values using SimpleImputer. We then apply scaling with the function StandardScaler, which takes away the mean divides with the standard deviation. Our categorical features are the flight numbers, month and the day of month of the year. We apply SimpleImputer again like previously and then create suitable dummy variables using the OneHotEncoder function. After separately preprocessing the variables, we can merge them together using the function ColumnTransformer(), where we state that the 2 pipelines to be merged using the transformers argument. The merged pipeline can now be used as the starting point for different machine learners as it already contains the preprocessing steps. Then, we are ready to attach a learner to the pipeline created above , which would complete the machine learning pipeline. Next, we split the data into training and test sets using the train_test_split function, by which we can train the pipeline and do evaluation on its prediction performance. The test size value indicates how much data is allocated to the test set, and thus the remaining, which is the majority, is the training set. The data needs to be split so as to be able to test the machine learning model on the test set, which was not touched while building the learner. Now, we tune the pipeline using the grid search with GridSearchCV(). To use the function, a parameter grid is needed which means a dictionary containing the names of each hyperparameter with its range of values is required. A hyperparameter is often used to help estimate the model parameters. We then compare two machine learning pipelines using plot Receiver Operating Characteristic curve. (UOL VLE, 2022)