SIM-UOL

ST2195 Programming for Data Science

Student ID 210472637

The datasets used in this project can be downloaded from the Harvard Dataverse at https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/HG7NV7. Flight data from 2004 and 2005 as well as all supplementary data (airport, plane and carrier data) must be downloaded for this project. It must be said that not all supplementary data will be used for the completion of the project. Nevertheless, they will be kept in the database in case they are required for future use.

I chose flight data from 2004 and 2005 because I wanted to work with more recent data. Although I could use more data from the previous years, I did not see any need for it and given the size of the two datasets alone, I saw no compelling reason to add more. All operations for this project are performed in R and Python. Many packages in R and libraries in Python are required to perform these operations. By using the DBI package in R and SQLite3 library in Python, I was able to create a database to hold all flight, plane, airport and carrier data. This database will be named "airline2.db".

It must be mentioned that this project was first completed in R, then Python. In order to understand this report clearly, I would recommend viewing the RMarkdown file first.

Dataset and Database Information

I will combine both 2004 and 2005 flight data into a single table named "ontime" with R and Python. This combined dataset yields at least 14.2 million observations and 29 variables. Needless to say, the size of this table is overkill for any data processing operation; a strong computer with an extremely large RAM will be needed and this is costly. Additionally, the processing times can be long, and this is time-consuming. To avoid this, I will draw a sample of 200,000 observations with fewer variables. The supplementary information will have tables of their own, straightforwardly named as "planes", "airports" and "carriers". All three tables are also added into the database and can be accessed when needed for querying.

Before creating the database, the code will first check if the database file exists and delete it. This is because the code will always create and store tables into the database file and this file will be stored locally on the computer. However, the file cannot be deleted if another process is using it and a *PermissionError* will appear. To avoid this, ensure that other programs using the database are closed. While the code for this coursework is provided in both R and Python, they cannot be run simultaneously, or this error will appear.

Setting up

Although there are tables in the database, we will not be using them directly. Instead, we will be using and manipulating data frames. To quickly explain, when reading the csv files in R/Python, the data from the files are extracted and placed into a table called a data frame. This data frame can be stored into the database with dbWriteTable() function in R or df.to_sql() in Python. Any changes made to the data frame will not affect the table that exists in the database.

A copy of the 'ontime' data frame will be made and named 'ontime2'. This is to protect the integrity of the original data frame. Many copies of data frames will be created in this project, and this is to allow for easy pinpointing/fixing should something go wrong. Variables that do not appear relevant in answering the questions for this project will be removed. The following variables were dropped: `ActualElapse`, `CRSActualElapse`, `TaxiIn`, `TaxiOut`, `CarrierDelay`, `NASDelay`, `WeatherDelay`, `SecurityDelay`. This should leave us with an *ontime2* data frame with 22 variables. I will also add two new variables, `FlightDate` and `Day`, as these will be needed to answer the first question. Note that this process should be done much later -- as I have done so in Python -- because it is more efficient, however for a better presentation in this report, I chose to do it earlier in R.

Creating the sample

As mentioned earlier, a sample named *o* will be created to answer the questions. This sample shall be drawn from the *ontime2* data frame because it contains the relevant variables. Any future samples that are needed in this project will always be drawn from this table. The sample will be drawn with the stratified sampling method because after exploring and analysing the data, I have found that the days and months act as subgroups, and I want a sample that best represents the population, i.e., the original table.

```
#Drawing a stratified sample of roughly 200,000 with proportional size for each day
table(ontime2$Day)/nrow(ontime2) * 200000
```

```
##
##    Friday    Monday  Saturday    Sunday  Thursday   Tuesday Wednesday
##  29606.89  29341.21  25553.83  27776.49  29544.69  29028.33  29148.54
```

Figure 1. A line of code in R to find the ideal size for each subgroup.

```
table(o$Day)
```

```
##
##    Friday    Monday  Saturday    Sunday  Thursday   Tuesday Wednesday
##     30419     30146     26255     28538     30355     29825     29948
```

Figure 2. The sample's subgroups.

```
#What the proportion of the sample should look like
ontime2.DayOfWeek.value_counts()/ontime.shape[0]*200000
```

```
5    29606.893295
4    29544.692291
1    29341.214557
3    29148.542810
2    29028.331450
7    27776.490683
6    25553.834913
Name: DayOfWeek, dtype: float64
```

```
#Check proportions of sample
o.DayOfWeek.value_counts()
```

```
5    30419
4    30355
1    30146
3    29948
2    29825
7    28538
6    26255
Name: DayOfWeek, dtype: int64
```

Figure 3. The same process performed in Jupyter Notebook.

The proportions for both sample and population are similar, so the size is acceptable. Since we added the *Day* variable in R earlier, we can tell from a glance which days have the least flights and the most flights. This is harder to do with the Python image because `DayOfWeek`, which has integers, was used instead. Keep in mind that the samples drawn in R and Python are unique and do not share the same set of observations.

After drawing the sample, further examination should be done to determine if the sample truly is a close representation of the population. This can be done by checking the summary of statistics for each variable and the variance of important variables. In this case, these variables would be `DepDelay` and `ArrDelay` because they will be used heavily later. If the sample is satisfactory, any variables we want to create should be done at this point. With fewer observations to work with, the processing time will be shorter.

Unlike with R, I created three new variables – `FlightDate`, `Day` and `NameOfMonth` – after drawing the sample with Python.

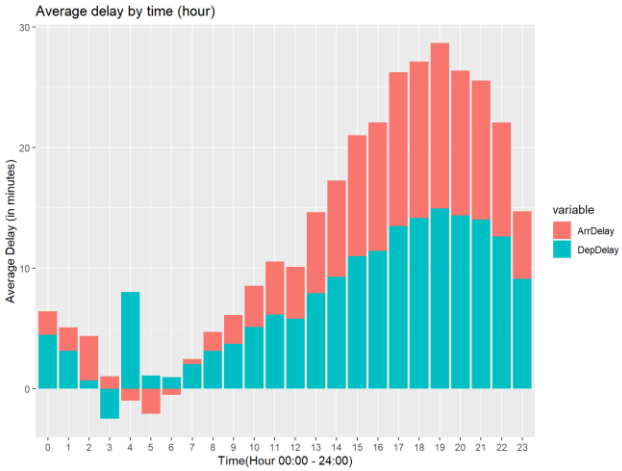
Question 1

What is the best time of the day, day of the week, time of year to fly to minimise delays?

To answer this question, we will break it into 3 parts. Firstly, what is the best time of the day to fly?

Since we are dealing with time, the time variables must be converted into a *time* class/datatype first. This is to allow R/Python to recognise these values as time values which will enable them to bin properly. We will then bin the variable `CRSDepTime` into time intervals of one hour. We use the `CRSDepTime` variable here instead of `DepTime` because `DepTime` represents *Actual Departure* times and we want the *Scheduled Departure* times instead. This is due to the scheduled departure time being known in advanced, but the actual departure time will always be unknown.

Since this question involves delays, the `DepDelay` and `ArrDelay` variables must be used. We will opt to find the average because we want to know what the typical value is for each hour.



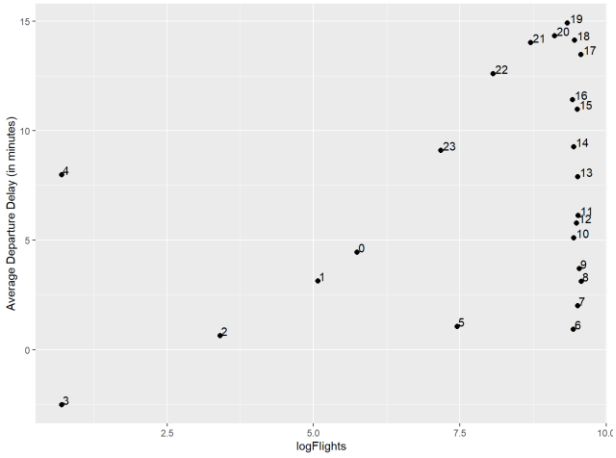
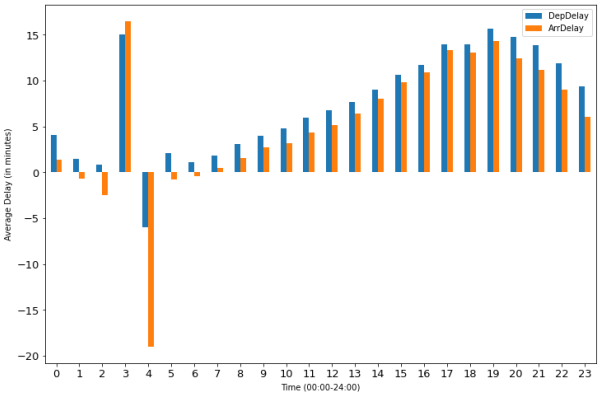
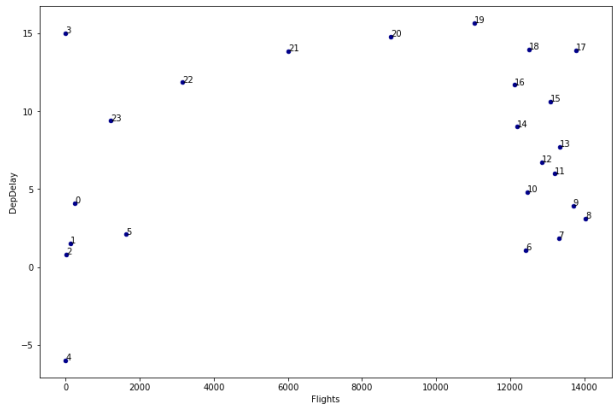


Figure 4. Plots created in R. The scatterplot uses log-transformed values.

Figure 5. Plots created in Python. The scatterplot uses original values.

For clarity, I will explain what the time bins mean. 0 represents scheduled departure times from 12am





to 1am. 1 represents scheduled departure times from 1am to 2am and so on and so forth.

The graphs above show that **5am to 7am** is the best time of the day to fly because the average delay during these hours is less than or around 5 minutes. Past 7am is when the average Arrival delay starts to noticeably and steadily increase. This is probably due to the increase of delays in other airports. Arguably, **1am to 3am** is also a contender, but the second graph shows that not a lot of flights occur during these hours, understandably due to lack of working staff as it is during the night.

I chose two different approaches with the scatterplots: I log-transformed the number of `Flights` variable in R whereas I did not in Python. I used log transformation because there is a lot of variation in this variable: during certain hours there are as little as 1 flight occurring and as many as 10,000 flights occurring. I wanted to see how both graphs compared and it would seem that the log-transformed graph is easier to read.

It should be noted that the average delay in the hours from **2am to 5am** will always differ between samples because of how few flights occur during these hours.

```
## # A tibble: 7 x 4
##   Day       DepDelay ArrDelay NumOfFlights
##   <fct>        <dbl>    <dbl>        <int>
## 1 Saturday      5.93     2.85        26255
## 2 Tuesday       6.55     5.15        29825
## 3 Wednesday     7.91     7.06        29947
## 4 Sunday        8.42     6.55        28540
## 5 Monday        8.78     7.41        30145
## 6 Thursday      9.50     8.89        30355
## 7 Friday        9.84     8.75        30419
```

Table 1. avg_time_delay table in R.

	Day	DepDelay	ArrDelay	Flights
0	Saturday	6.196722	3.007846	25747
1	Tuesday	6.731171	5.573312	29197
2	Wednesday	7.736614	6.974712	29303
3	Sunday	8.849907	6.755898	27976
4	Thursday	9.105842	8.407014	29714
5	Monday	9.162531	8.005864	29502
6	Friday	9.741041	8.819483	29831

Table 2. avg_time_delay table in Python.

What is the best day of the week to fly?

Thanks to the `Day` variable created earlier the coding for this part of the question is quite simple. First, a subset of the sample is made with relevant variables to answer this question. The resulting table is shown on the right. From there, we create our graphs.

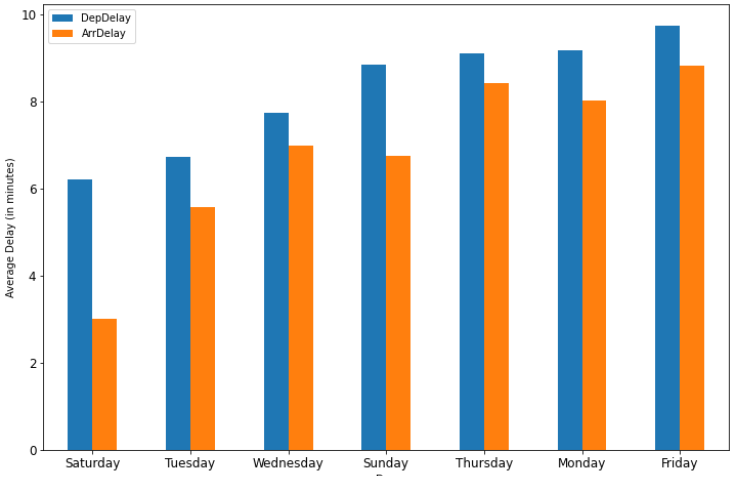


Figure 6. Bar graph from Python.

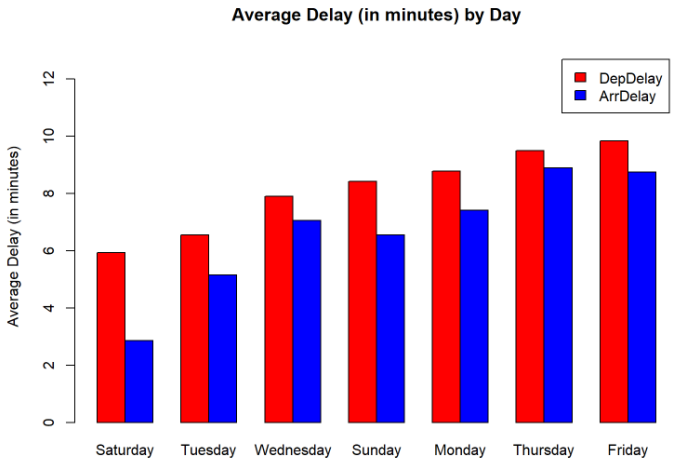


Figure 7. Bar graph from R.

The best day to fly is **Saturday** or **Tuesday** as shown on these graphs above.

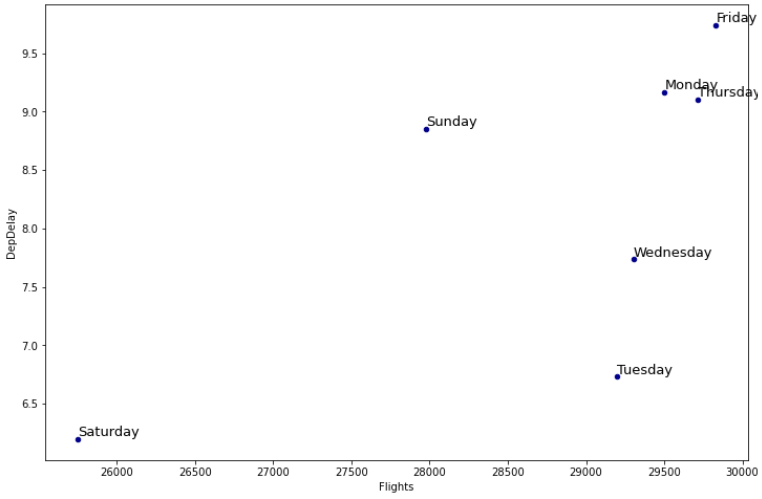


Figure 8. Scatterplot from Python.

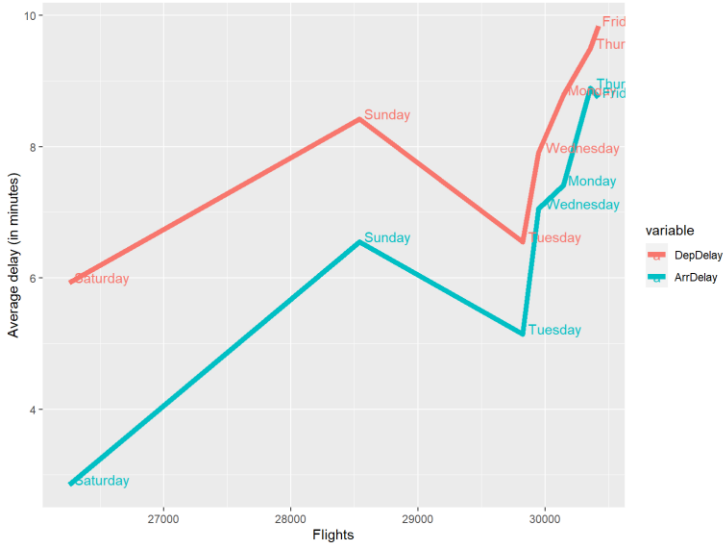


Figure 9. Scatterplot from R.

Note that the least number of flights occur on **Saturday** which explains why it has the lowest average delay. **Tuesday** has the almost the same number of flights as the other days but has the lowest average delay despite being as busy.

This graph on the right demonstrates that `DepDelay` and `ArrDelay` share a relationship as their respective lines are very similar to each other. This suggests that departure delay causes arrival delay. For flights that have departure delay but no or less arrival delay, it could be speculated that the plane adjusted its speed to match its scheduled arrival time.

What is the best time of the year to fly?

September or **April** is the best time of the year to fly.

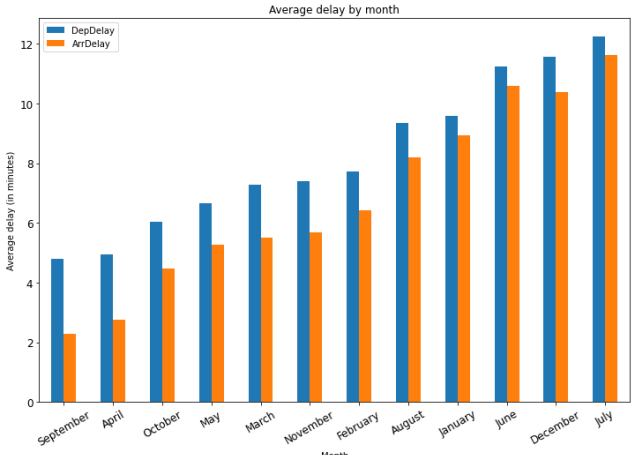


Figure 10. Bar graph from Python.

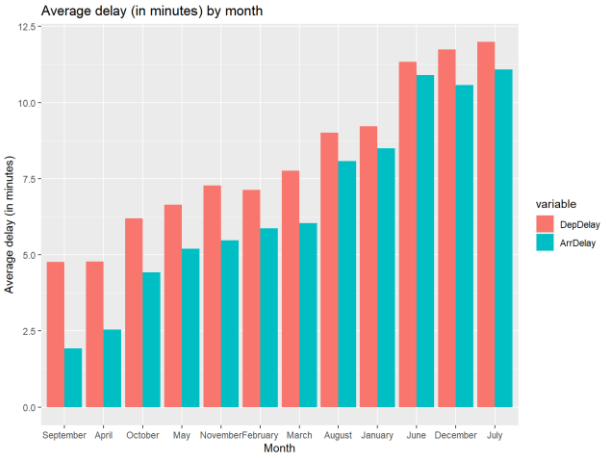


Figure 11. Bar graph from R.

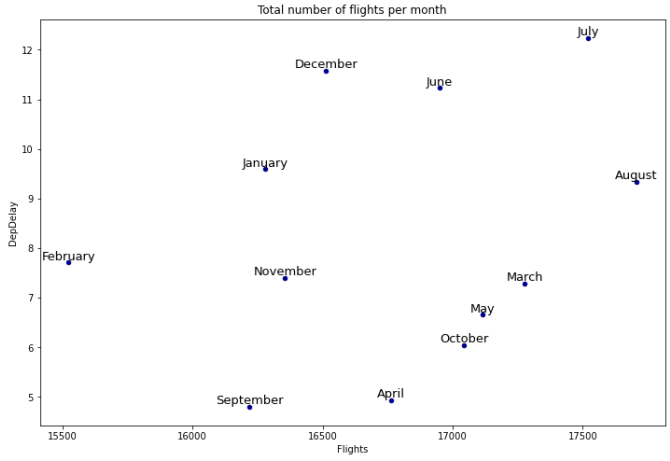


Figure 12. Scatterplot from Python.

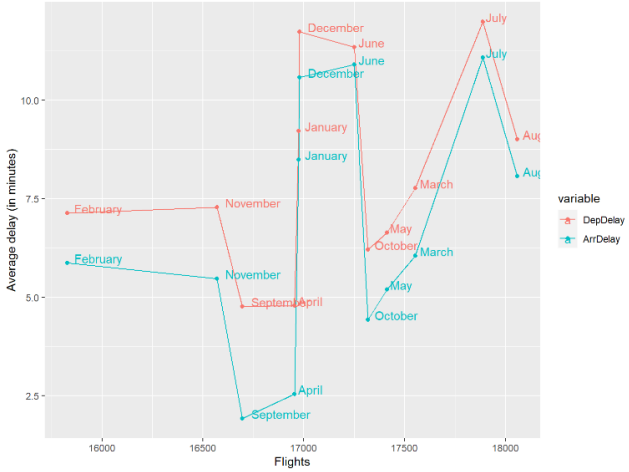


Figure 13. Line plot from R.

The graphs above show the number of flights that occur for each month and the average delay for each month. Again, we see that there is a correlation between `DepDelay` and `ArrDelay`. Unlike the graphs using Day as the x-axis, we also see that there is no relationship between the number of flights and the average delay.

In summary, the best time of the day, day of the week and time of the year to fly to minimise delays is **5am to 7am on a Saturday or Tuesday in September or April.**

Question 2

Do older planes suffer more delays?

This question requires the planes table to answer. Taking a quick look at the planes table, we can find missing values in certain variables. This includes the `issue_date` variable which we will use to determine a plane's age. Before we start, we must clean the planes table by dropping rows with invalid/empty issue date. Then we will need to convert the `issue_date` variable to a *date* class, so that we can properly extract the year when sub-setting later.

Oddly enough, there are certain models that have an issue date beyond 2005. We can assume that this is not an error, but rather that the planes are being used for a different purpose. We are only interested in planes with an issue date equal to or less than 2005 however, so these will be dropped.

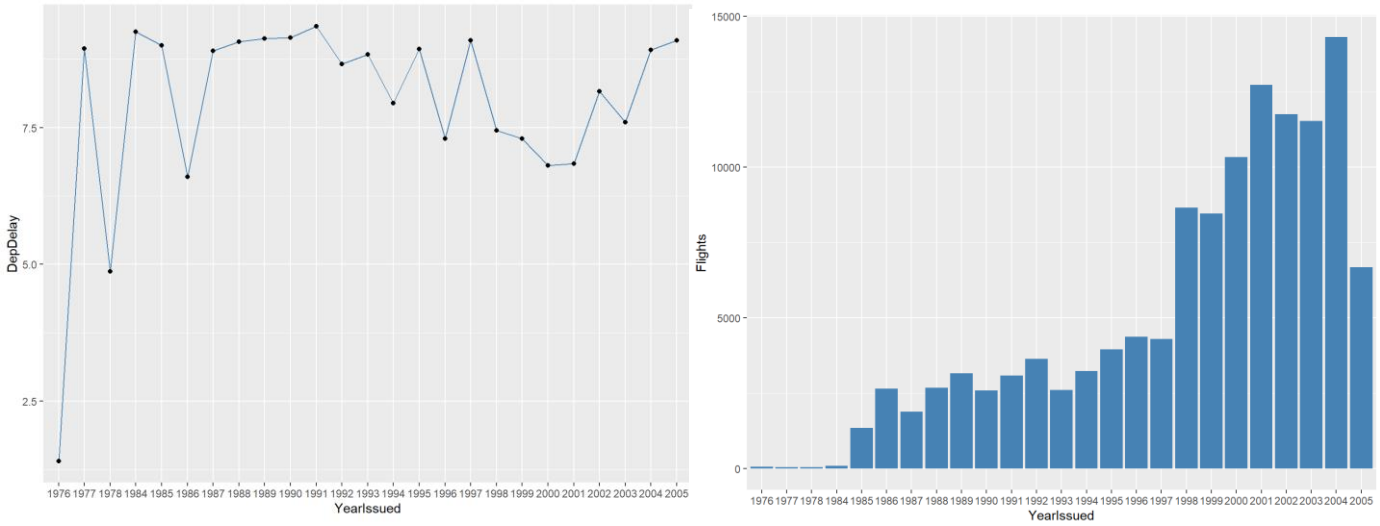


Figure 14. Line plot and bar graph from R.

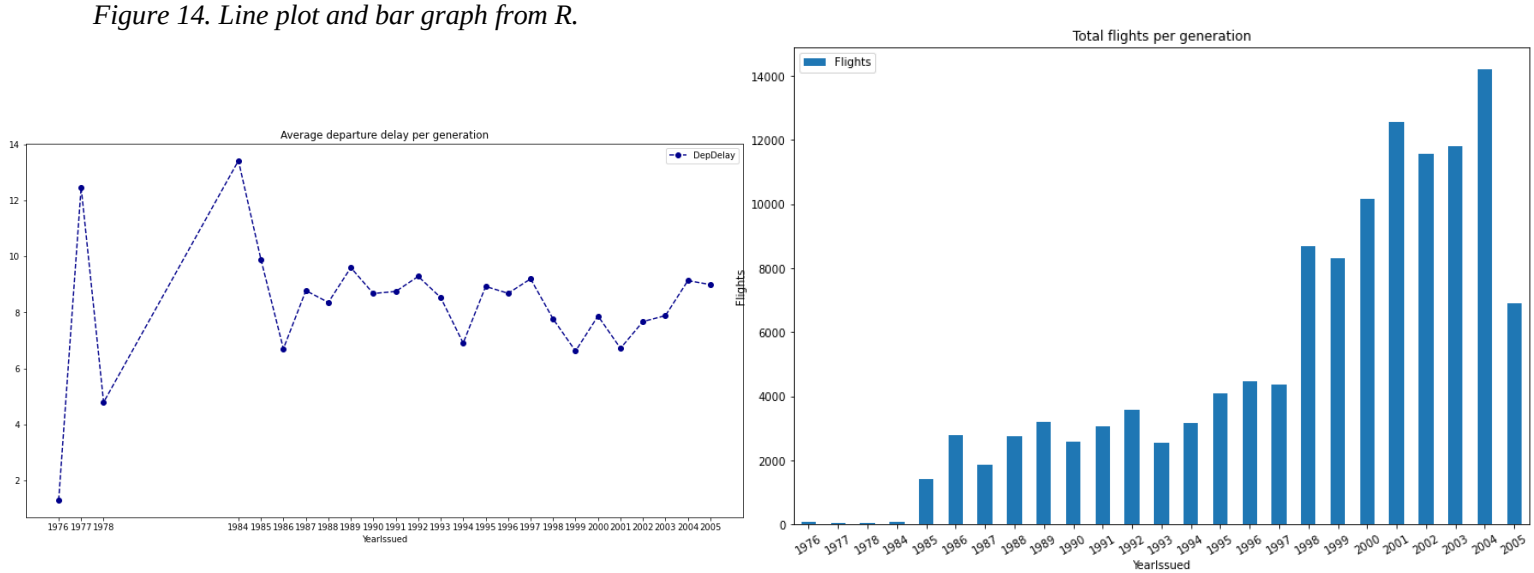


Figure 15. Line plot and bar graph from Python.

The line plots on the left show that the oldest planes (1976-1978) have less average delay than younger planes. However, if you look at the bar graphs on the right, we can see that those same planes do not fly very often. In fact, planes younger than 1997 fly more frequently.

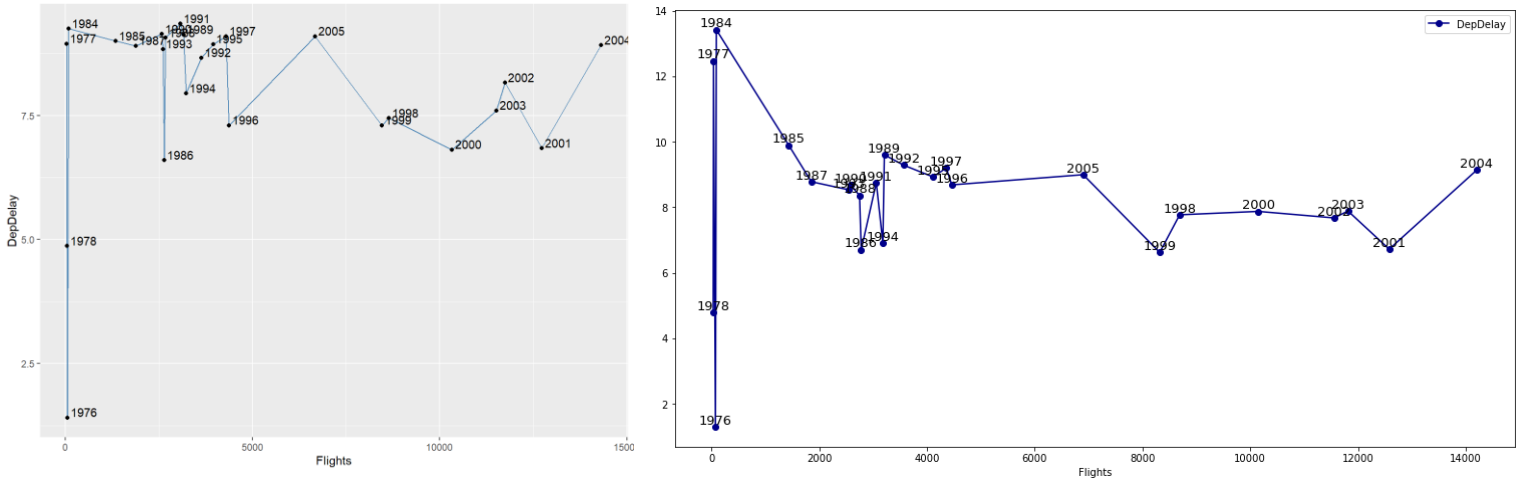


Figure 16. Line plot from R and Python.

The line graphs above combines all 3 variables, and we can see that older planes, despite flying less frequently, have higher average delay than younger planes. Going by a ratio measurement of flight:delay, **we can safely say that older planes do suffer more delays**. The only exception is planes issued in 2005: they fly the least in the 1998-2005 group yet have a higher average delay.

Question 3

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

Before we begin, we will drop flights that were cancelled or diverted due to their association with NA values and because we can treat these flights as if they never flew. We are only interested in flights that have filled out the variables `DepTime` and `ArrTime` because this would mean that the planes have flown and reached their destinations.

To answer this question, we must figure out the number of flights that happen for each month in their respective years. This requires grouping the rows, but we are unable to do this unless we remove the day of the month in the date. However, by removing the day in the date, R will not recognise the month/year date as a *date* class because it is incomplete. We will substitute the 1st day of the month instead to resolve this.

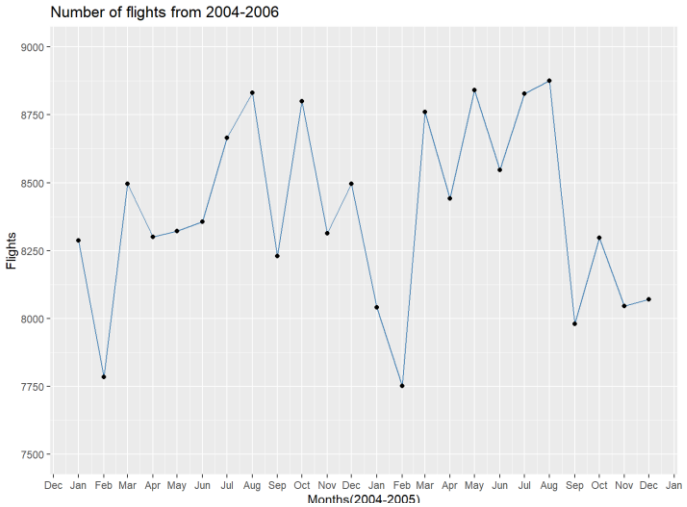


Figure 17. Line plot from R.

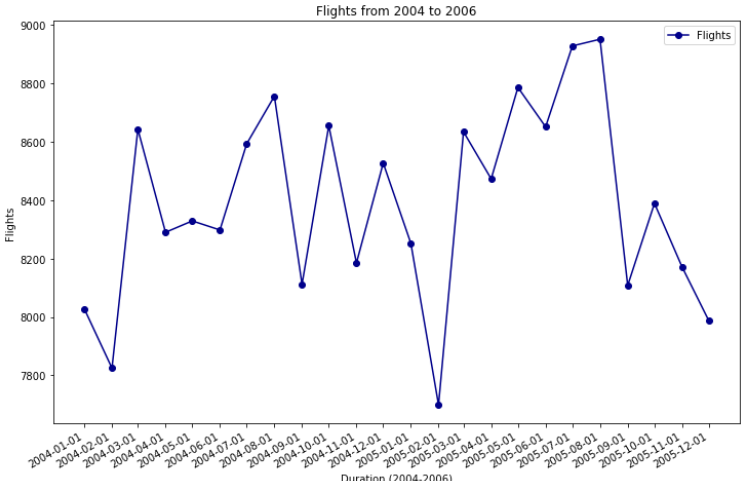


Figure 18. Line plot from Python.

In the graphs above, people tend to fly less near the end of or beginning of the year (Nov-Feb). There are also certain months where the number of people flying dips, creating valleys in the graph. This phenomenon occurs in February, September and November, and remains consistent in both years.

Similarly, there are months where the number of people flying peaks and this happens in March, August and October.

It also appears that in 2004, people flew more in October to December, but in 2005 less people flew in those months. People also flew less in May to June of 2004, but more people flew in May to August of 2005.

Question 4

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

We subset and filter rows with positive `LateAircraftDelay` values because presumably this variable explains delays in the airport. I also decided to use different approaches in R and Python: I will look at an airport's inbound flights in R and I will look at an airport's outbound flights in Python.

	Origin	airport	SumDelay	Flights
0	ORD	Chicago O'Hare International	65720	1383
1	ATL	William B Hartsfield-Atlanta Intl	56072	1265
2	DFW	Dallas-Fort Worth International	27428	774
3	LAS	McCarran International	26400	669
4	EWR	Newark Intl	23909	496
5	PHL	Philadelphia Intl	21478	472
6	LAX	Los Angeles International	19557	508
7	IAD	Washington Dulles International	17664	347
8	DEN	Denver Intl	17615	404
9	PHX	Phoenix Sky Harbor International	16472	490
10	LGA	LaGuardia	16447	323
11	MCO	Orlando International	13884	341
12	BWI	Baltimore-Washington International	13339	318
13	BOS	Gen Edw L Logan Intl	13217	306
14	IAH	George Bush Intercontinental	12895	288

```
mostdelays
```

```
## # A tibble: 84 x 4
## # Groups:   Origin [84]
##    Origin airport                          SumDelay Flights
##    <chr>  <chr>                               <int>   <int>
##  1 ORD    Chicago O'Hare International        66242    1399
##  2 ATL    William B Hartsfield-Atlanta Intl   54572    1157
##  3 DFW    Dallas-Fort Worth International     29966     843
##  4 EWR    Newark Intl                         26699     494
##  5 LAS    McCarran International              25113     659
##  6 PHL    Philadelphia Intl                   21930     459
##  7 IAD    Washington Dulles International     19900     361
##  8 DEN    Denver Intl                         15965     392
##  9 LAX    Los Angeles International           15595     452
## 10 LGA    LaGuardia                           14596     336
## # ... with 74 more rows
```

Table 3. The mostdelays table from both Python and R.

ORD and ATL are the two airports with the highest amount of late aircraft delays. Other notable airports are DFW, EWR, LAS, PHL, IAD, LGA, DEN, LAX, PHX and MDW.

If we look at the airports individually, notice that most of the top 9 airports that cause the highest delays in those airports are in the top 12 airports of highest delays in the *mostdelays* table.From this, we can assume that yes, delays in one airport can cause delays in others.

I also a constructed a map on the right in R to visualise flight connections of airports with delays more than 1500.

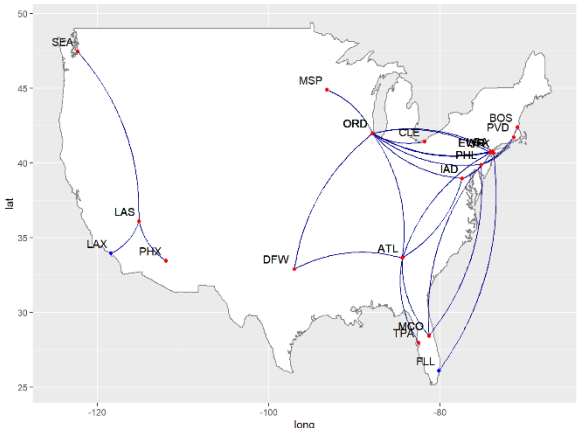


Figure 19. Map from R.

ORD Airport

```
##   Dest  Origin SumDelay Flights
##   <chr> <chr>     <int>   <int>
## 1 ORD   ATL        1857      36
## 2 ORD   LGA        1836      35
## 3 ORD   EWR        1693      22
## 4 ORD   IAD        1407      28
## 5 ORD   PHL        1369      33
## 6 ORD   DEN         952      23
## 7 ORD   DSM         901      21
## 8 ORD   PIT         789      22
## 9 ORD   MIA         738      20
```

	Origin	Dest	SumDelay	Flights
0	ORD	MSP	2705	49
1	ORD	CLE	2068	36
2	ORD	EWR	1866	28
3	ORD	PHL	1832	38
4	ORD	DFW	1821	41
5	ORD	DTW	1688	32
6	ORD	CLT	1646	23
7	ORD	DEN	1630	29
8	ORD	LGA	1556	40

MSP Airport

```
##    Dest  Origin SumDelay Flights
##    <chr> <chr>     <int>   <int>
##  1 MSP   ORD        1961      47
##  2 MSP   DEN        1112      17
##  3 MSP   ATL         924      23
##  4 MSP   EWR         565      14
##  5 MSP   MDW         458      10
##  6 MSP   OMA         367       9
##  7 MSP   DFW         322      12
##  8 MSP   PHX         318       9
##  9 MSP   LGA         293       9
## 10 MSP   DCA         217       9
```

	Origin	Dest	SumDelay	Flights
0	MSP	ORD	1152	23
1	MSP	DEN	593	13
2	MSP	IAH	445	9
3	MSP	EWR	388	6
4	MSP	ATL	352	10
5	MSP	CLE	269	5
6	MSP	PHX	174	5
7	MSP	DTW	142	8
8	MSP	MDW	142	5

EWR Airport

	Dest	Origin	SumDelay	Flights
	<chr>	<chr>	<int>	<int>
1	EWR	ORD	1567	35
2	EWR	IAD	1511	24
3	EWR	ATL	989	29
4	EWR	RDU	798	20
5	EWR	CLT	792	20
6	EWR	MCO	619	10
7	EWR	DTW	450	12
8	EWR	FLL	376	11
9	EWR	SFO	370	9
10	EWR	BOS	356	9
11	EWR	SJU	231	9

	Origin	Dest	SumDelay	Flights
0	EWR	RDU	1183	28
1	EWR	LAX	1109	14
2	EWR	ATL	1074	24
3	EWR	PIT	1057	15
4	EWR	IAD	981	23
5	EWR	CLT	885	23
6	EWR	MIA	749	11
7	EWR	FLL	736	13
8	EWR	ALB	634	11

RDU Airport

	Dest	Origin	SumDelay	Flights
	<chr>	<chr>	<int>	<int>
1	RDU	ATL	1312	26
2	RDU	PHL	812	23
3	RDU	EWR	715	15
4	RDU	LGA	526	15
5	RDU	ORD	525	10
6	RDU	BWI	315	9
7	RDU	BOS	291	8
8	RDU	DCA	222	8
9	RDU	DTW	210	10

	Origin	Dest	SumDelay	Flights
0	RDU	BOS	1050	16
1	RDU	DFW	725	9
2	RDU	ATL	642	15
3	RDU	PHL	565	13
4	RDU	BNA	353	6
5	RDU	BWI	294	6
6	RDU	LGA	291	11
7	RDU	JFK	277	6
8	RDU	CMH	225	6

Question 5

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

We will utilize the `caret` package in R and `sci-kit` library in Python for the models.
We need to create 2 separate tables: one without `DepDelay` and one without `ArrDelay`.
This is because both variables are strongly correlated and `ArrDelay` depends heavily on `DepDelay`.
Using both would give us a high R2 score and better predictive power in our models, but realistically this would not be possible because both variables are also unknown variables, meaning that you would not know in advance what the `DepDelay` or `ArrDelay` is.We will use specific variables, namely the ones that will be known to us in advance: `Month`, `DayOfWeek`, `CRSDepTime`, `CRSArrTime`.

We also want to include `Origin` and `Dest`, however these variables are categorical and due to the nature of the sample (it is quite large), it is impossible to bin these without long processing times and a large memory. We can simply use the `Distance` variable as a substitute, but we must keep in mind that some airport connections share the same distance.

Initially, I wanted to do a backward selection to choose variables where I start with using all the variables and remove the ones with the lowest p-values one-by-one because they signify that it is not important to the model, but this was not possible without using a large amount of memory due to the size of the sample. I tried drawing another sample of 10,000 observations and used this sample to fit the models but found that the R2 score did not improve much as it remained below 1%. Finding no improvement, I reverted to using the sample of 200,000 observations. The first set of models will use Multi-Linear Regression.

```
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -7.9756834  0.2932277 -27.200  < 2e-16 ***
## Month        0.0959107  0.0194056   4.942 7.72e-07 ***
## DayofMonth   0.0503656  0.0075483   6.672 2.52e-11 ***
## CRSDepTime   0.0065811  0.0002306  28.542  < 2e-16 ***
## CRSArrTime   0.0038093  0.0002234  17.049  < 2e-16 ***
## Distance     0.0002713  0.0001168   2.323   0.0202 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
predict(dep_lr_model, head(dep_o5))
```

```
##         1         2         3         4         5         6
## -0.4251394 15.4906038  8.5936041  8.4539079 13.1261109  4.3465836
```

Figure 20. Summary of Multi-Linear Regression model output and predictions in R.

```
Month          0.076216
DayofMonth     0.062695
CRSDepTime     0.007237
CRSArrTime     0.003263
Distance       0.000554
dtype: float64
```

```
R2 = 0.024821667827259586
MSE = 850.8077184817662
RMSE = 29.168608442669427
```

	Predicted DepDelay
0	11.642691
1	4.538113
2	4.962679
3	3.091069
4	13.191388

Figure 24. Multi-Linear Regression model coefficients, accuracy scores and predictions in Python.

According to the summary of the `DepDelay` model on the left, all variables chosen are significant, although `Distance` has the weakest p-value. The R2 score for this model is 0.23% with an RMSE of 29.78. This means that the model does not perform well, and predictions may be inaccurate. The figure on the right shows that the final model's result from Python is also quite similar with a R2 score of 0.25% and a RMSE of 29.1686. Knowing that all the variables used are significant, this suggests that we are not using enough variables.

```
## UniqueCarrierMQ -1.0375821  0.3416398  -3.037 0.002389 **
## UniqueCarrierNW -3.3889216  0.3342392 -10.139  < 2e-16 ***
## UniqueCarrierOH -1.3422906  0.3704498  -3.623 0.000291 ***
## UniqueCarrierOO -2.8566083  0.3443526  -8.296  < 2e-16 ***
## UniqueCarrierTZ -4.6231117  0.7629257  -6.060 1.37e-09 ***
## UniqueCarrierUA -1.1823787  0.3248837  -3.639 0.000273 ***
## UniqueCarrierUS -1.8937861  0.3489924  -5.426 5.76e-08 ***
## UniqueCarrierWN  0.4391709  0.2834723   1.549 0.121322
## UniqueCarrierXE -2.7755333  0.3644657  -7.615 2.64e-14 ***
## Distance         0.0003738  0.0001316   2.840 0.004513 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 29.73 on 201332 degrees of freedom
## Multiple R-squared:  0.02715,    Adjusted R-squared:  0.02704
## F-statistic: 234.1 on 24 and 201332 DF,  p-value: < 2.2e-16
```

Figure 22. Summary of model output in R.

As a demonstration, I will add the variable `UniqueCarrier`. Thanks to this new addition, the model has improved slightly with a RMSE of 29.73 compared to the previous 29.78 and a R2 score of 0.27% compared to the previous 0.23%, although it is still not at an acceptable level.

Next, we will try building Ridge Regression models.

```
## Ridge Regression
##
## 140951 samples
##      6 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 10 times)
## Summary of sample sizes: 126856, 126856, 126856, 126855, 126857, 126856, ...
## Resampling results across tuning parameters:
##
##   lambda  RMSE      Rsquared    MAE
##   0e+00   29.85533  0.02723497  15.26858
##   1e-04   29.85533  0.02723501  15.26857
##   1e-01   29.85633  0.02721629  15.26063
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was lambda = 1e-04.
```

```
predict(dep_rr_model, head(test_dep))
```

```
##        1        2        3        4        5        6
##  4.369639  3.462671  6.148625 14.797682 13.562287 14.252603
```

Figure 20. Summary of Ridge Regression model output and predictions in R.

```
Month          0.076216
DayofMonth     0.062695
CRSDepTime     0.007237
CRSArrTime     0.003263
Distance       0.000554
dtype: float64
```

	Predicted DepDelay
0	11.642691
1	4.538113
2	4.962679
3	3.091069
4	13.191388

```
R2 = 0.024821667826107285
MSE = 850.8077184827715
RMSE = 29.16860844268666
```

Figure 24. Ridge Regression model coefficients, accuracy scores and predictions in Python.

Using Ridge Regression does not improve model performance as the R2 score and RMSE are similar to the previous Multi-Linear Regression model.