Explore Delay Patterns and Attempt to Build Predictive Models Using Historical Flight Data

Table of Contents

Introduction 2
Data 2
1. In R 2
1.1. Cleaning Data 2
1.2. When is the best time of day, day of week, and time of year to fly to minimise delays? 2
1.4. How does the number of people flying between different locations change over time? 4
1.5. Can you detect cascading failures as delays in one airport create delays in others? 5
1.6. Use the available variables to construct a model that predicts delays. 5
2. In Python 6
2.1. Data Cleaning 6
2.2. When is the best time of day, day of week, and time of year to fly to minimise delays? 6
2.3. Do Older Planes Suffer More Delays? 7
2.4. How does the number of people flying between different locations change over time? 8
2.5. Can you detect cascading failures as delays in one airport create delays in others? 9
2.6. Use the available variables to construct a model that predicts delays. 9
Appendix and References 11

Introduction

Flight delays are a serious problem in the industry which cost millions of dollars of the airlines, airports, and passengers. According to the Federal Aviation Administration (FAA), a flight is considered delayed when it departs or arrives at its destination 15 minutes after its scheduled time. This research serves to answer these questions.

1. When is the best time of day, day of the week, and time of year to fly to minimise delays?
2. Do older planes suffer more delays?
3. How does the number of people flying between different locations change over time?
4. Can you detect cascading failures as delays in one airport create delays in others?
5. Use the available variables to construct a model that predicts delays.

Data

In order to answer those questions, flight data from years 2004 & 2005 as well as airports, carriers, and plane-data has been downloaded from the Harvard Dataverse, at https://doi.org/10.7910/DVN/HG7NV7 and will be used in this research.

1. In R

1.1. Cleaning Data

Each entry of the 2004/2005.csv file corresponds to a flight and each flight is described according to 31 variables. However, we will only be focusing on those variables that will help up with this research such as:

- **Year, Month, DayofMonth, DayOfWeek**: dates of the flight
- **Origin** and **Dest**: code attributed by IATA to identify the airports
- **CRSDepTime & CRSArrTime** : scheduled times of take-off and landing
- **DepTime & ArrTime**: real times at which take-off and landing took place
- **ArrDelay & DepDelay**: difference (in minutes) between planned and real times

This process is show in Figure 1 where the unnecessary data columns are dropped to make the data we work with smaller. Also, flights that have been cancelled have also been removed to help streamline the data even more.

```
df_ontime_db = select(df_ontime_db, -c('CRSElapsedTime', 'AirTime','Distance','TaxiIn', 'TaxiOut', 'Diverted','CancellationCode', 'UniqueCarrier',
                                       'CarrierDelay','WeatherDelay','NASDelay', 'SecurityDelay', 'LateAircraftDelay'))
df_ontime_db = df_ontime_db[df_ontime_db$Cancelled != 1,] #removed flights that were cancelled, to lower number of entries and wont affect result
```

Figure 1

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

```
flight_sum<- df_ontime_db %>%
  #group flight cancellation and flight delay into one level
  mutate(Delay = ifelse(DepDelay >= 15 | is.na(DepDelay) == TRUE, 1, 0),) %>%
  #select relevant variables and save to a new data table
  select(Delay, Year, Month, DayofMonth, DayOfWeek,Origin, CRSDepTime)
flight_sum$CRSDepHour = flight_sum$CRSDepTime %/% 100

flight_sum_delay <- df_ontime_db %>%
  #group flight cancellation and flight delay into one level
  mutate(Delay = ifelse(DepDelay >= 15 | is.na(DepDelay) == TRUE, 1, 0),) %>%
  #select relevant variables and save to a new data table
  select(Delay, Year, Month, DayofMonth, DayOfWeek,Origin, CRSDepTime)
flight_sum_delay = flight_sum_delay[flight_sum_delay$Delay !=0,]
flight_sum_delay$CRSDepHour = flight_sum_delay$CRSDepTime %/% 100
```

Figure 2

Two dataframes called *flight_sum* and *flight_sum_delay* is created. A new column called *Delay* created. Its values are determined by the values in the *DepDelay* column, if the values in *DepDelay* is more than '15', the

corresponding *Delay* column will have a value of '1', else, a '0'. This then allows us to remove flights that were not delayed from flight_sum_delay and focused on those that were. At the same time, another new column is added using *flight_sum$CRSDepHour = flight_sum$CRSDepTime %/% 100*. This new column allows us to extract the hour of the scheduled flights.

```
flight_sum_day = flight_sum %>%group_by(CRSDepHour) %>% summarize(n_Delays = n())
flight_sum_delay_day = flight_sum_delay %>%group_by(CRSDepHour) %>% summarize(n_Delays = n())
ggplot() +
  geom_col(data=flight_sum_day, aes(x= CRSDepHour, y = n_Delays, colour='Flights', fill='Flights', label=n_Delays))+
  geom_text(nudge_y = 1)+
  geom_col(data=flight_sum_delay_day, aes(x= CRSDepHour, y = n_Delays, colour='Delayed', fill='Delayed', label=n_Delays)) +
  geom_text(nudge_y = 1)+
  scale_x_discrete(limits = 0:23) +
  ggtitle("Distribution of Flights & Delays by Time of Day") +
  labs(x="Time (in 24Hr Format)", y='Count', color ="Legend", fill = "Legend")
```

Figure 3: Data Manipulated to Show Best Day to Fly

```
flight_sum_week = flight_sum %>%group_by(DayOfWeek) %>% summarize(n_Delays = n())
flight_sum_delay_week = flight_sum_delay %>% group_by(DayOfWeek) %>% summarize(n_Delays = n())
ggplot() +
  geom_col(data=flight_sum_week, aes(x= DayOfWeek, y = n_Delays, colour='Flights', fill='Flights', label=n_Delays))+
  geom_text(nudge_y = 1)+
  geom_col(data=flight_sum_delay_week, aes(x= DayOfWeek, y = n_Delays, colour='Delayed', fill='Delayed', label=n_Delays)) +
  geom_text(nudge_y = 1)+
  scale_x_discrete(limits = 1:7) +
  ggtitle("Distribution of Flights & Delays by Day of Week") +
  labs(x="Day of Week", y='Count', color ="Legend", fill = "Legend")
```

Figure 4: Data Manipulated to Show Best Week to Fly

```
flight_sum_month = flight_sum %>%group_by(Month) %>% summarize(n_Delays = n())
flight_sum_delay_month = flight_sum_delay %>% group_by(Month) %>% summarize(n_Delays = n())
ggplot() +
  geom_col(data=flight_sum_month, aes(x= Month, y = n_Delays, colour='Flights', fill='Flights', label=n_Delays))+
  geom_text(nudge_y = 1)+
  geom_col(data=flight_sum_delay_month, aes(x= Month, y = n_Delays, colour='Delayed', fill='Delayed', label=n_Delays)) +
  geom_text(nudge_y = 1)+
  scale_x_discrete(limits = 1:12) +
  ggtitle("Distribution of Flights & Delays by Month") +
  labs(x="Month", y='Count', color ="Legend", fill = "Legend")
```

Figure 5: Data Manipulated to Show Best Month to Fly

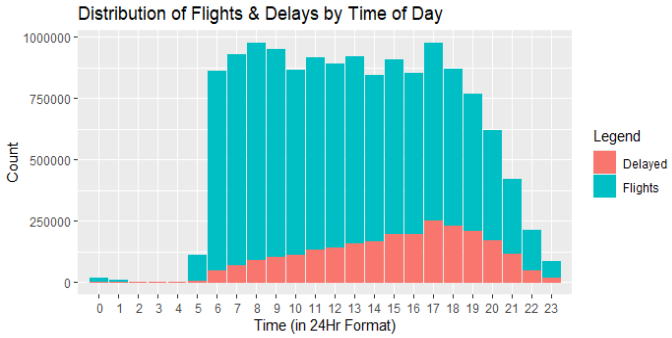

Figure 6: Graph of Distribution of Flights & Delays by Time of Day

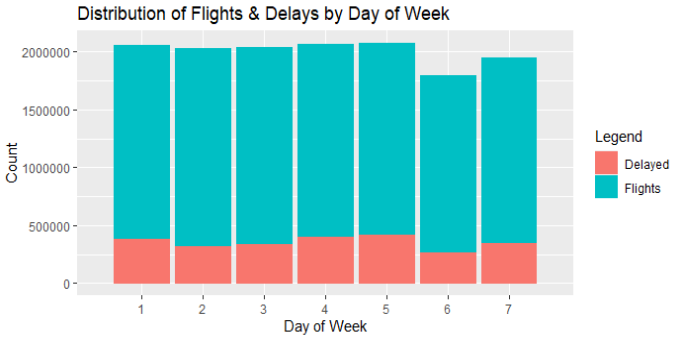

Figure 7: Graph of Distribution of Flights & Delays by Day of Week

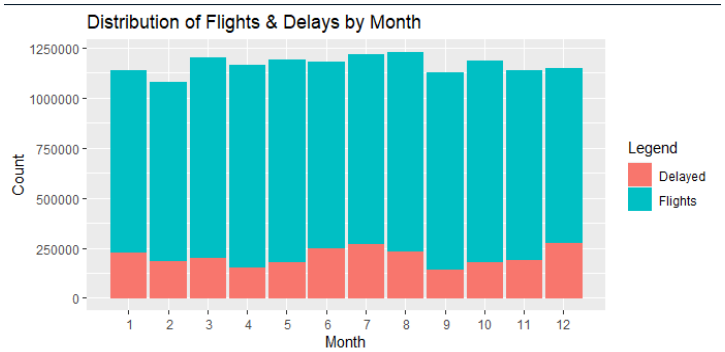

Figure 8: Graph of Distribution of Flights & Delays by Month

Upon viewing the plots, from Figure 6, we are able to conclude that flights departing between 6AM and 9AM has a lower ratio of delayed flight to total flights and therefore would be the better time to try and avoid delays. From Figure 7, we are able to conclude that flights on Tuesdays and Wednesday has a lower ratio of delayed flight to total flights and therefore would be the better days to travel to try and avoid delays. Lastly, from Figure 8, it would appear that months April and September has a lower ratio of delayed flight to total flights and therefore would be the better month to travel to try and avoid delays.

1.3. Do Older Planes Suffer More Delays?

```
flight_sum_old<- flight_sum_old %>%
  #group flight cancellation and flight delay into one level
  mutate(Delay = ifelse(DepDelay >= 15 |ArrDelay >= 15| is.na(DepDelay) == TRUE, 1, 0),)
join_by = colnames(planes)[1]
names(join_by)=colnames(flight_sum_old)[10]
older_planes_delay = left_join(flight_sum_old,planes, by=join_by)
#CHI SQUARE TESTING
chisq.test(older_planes_delay$year, older_planes_delay$Delay, correct=FALSE)
```

Figure 9

In Figure 9, a new data frame called *flight_sum_old* is created and like the previous *df_ontime_db* data frame, unwanted columns and cancelled flights are cancelled and a new column called *Delay* is created however in this instance, *Delay* is a combination of arrival delay and departure delay. *older_planes_delay* is then created by joining *flight_sum_old* together with *planes* by their *tailnum* column.

```
data:  older_planes_delay$year and older_planes_delay$Delay
X-squared = 8982.5, df = 49, p-value < 2.2e-16
```

Figure 10

With this new data frame, a chi-square test is performed using *Delay* and *year* column which results in a p-value of less that 0.001. It is therefore statistically significant and indicates strong evidence against the null hypothesis. Therefore, we reject the null hypothesis and conclude that older planes do suffer more delays.

Data in “years” column, the date the aircraft model is launched, and not “issue_date” is used to calculate aircraft age as “issue_date” is the date the airline receives the aircraft, as it is normal for aircraft to be bought and sold multiple times by carriers, it might not be the original delivery date, therefore the true age of the plane is unknown and may affect the results, also, many aircrafts are bought and delivered close to launch date, this means that launch date, the data in the “year” column, will give more accurate results in terms of if older aircraft causes more delays.

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

```
q1_2004_db = dbGetQuery(conn, 'SELECT Origin, Dest FROM ontime
                               WHERE (ontime.Year = "2004")
                               AND (ontime.Origin ="ATL")
                               AND ((ontime.Month = "1") OR (ontime.Month = "2") OR (ontime.Month = "3"))')
```

Figure 11

As there is too much data to be individually managed, I have decided to create multiple data frames for different time periods. I have decided to split the individual data frame into 1st, 2nd, 3rd, and 4th quarter of years 2004 and 2005. We will only observe flight from Atlanta Airport as Atlanta has the highest number of departing flights. Figure 8 is an example of one data frame, I again used *dbGetQuery* but this time I selected the “Origin” column containing flight data from Atlanta from the *ontime* table only from year 2004 and between the first three months. This is method is repeated for the remaining data.

Next, the function *par(mfrow=c(2,4))* is used to create a plot that has 2 rows and 4 columns.

```
q1_2004_db = q1_2004_db %>% group_by(Dest) %>% summarize(n_count = n()) %>% arrange(desc(n_count))%>% slice(1:10)
q12004=ggplot(q1_2004_db, aes(reorder(x=Dest, -n_count), y=n_count, colour = Dest, fill = Dest), stat="identity") +
  geom_col() + geom_text(aes(label = n_count), colour='black')+
  ggtitle("Top 10 Most Travelled Location From Atlanta Airport in First Quarter") +
  ylab('Count') + xlab('Airport')
```

Figure 12

The data frame seen in Figure 12 is then piped through different functions. First, the number observations are counted and sorted by “Dest” in a descending order and only the top 10 are selected. This new data frame is called *q1_2004_db* and a bar graph is plotted. This step is repeated for the rest of the data and is collated into the 2x4 plot seen in Figure 13.

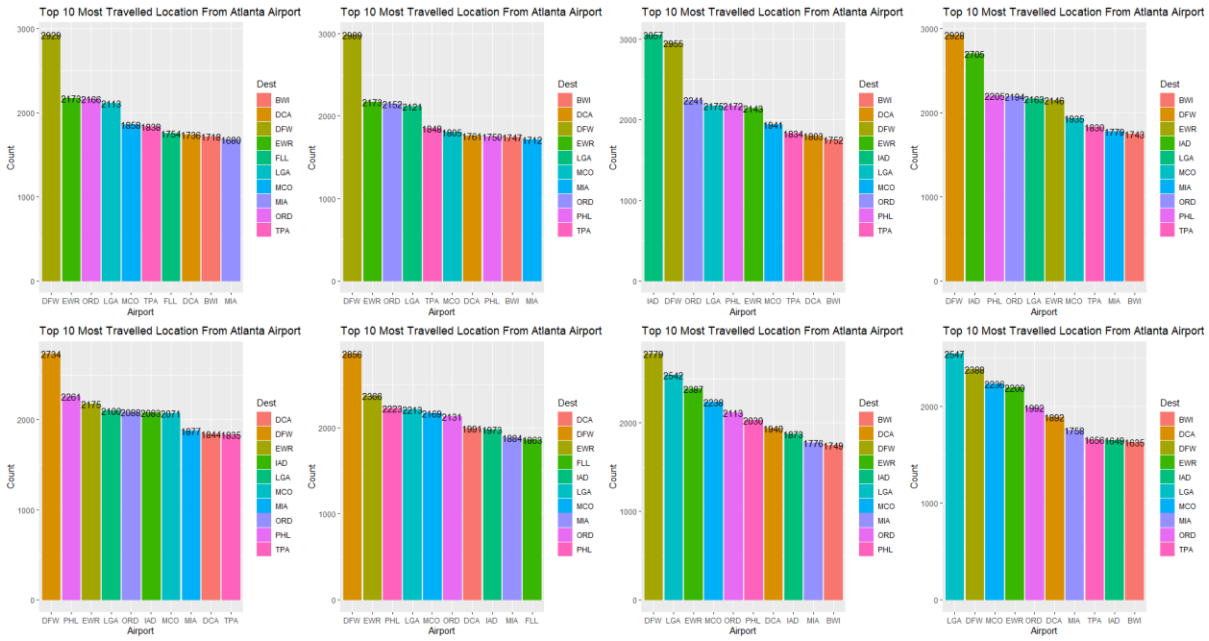

Figure 13

After analysing the plot, we can observe that with exception of the third quarter of 2004 and the last quarter in 2005, DFW airport has constantly been the most location. In the third quarter of 2004, there was a large and sudden increase to flight to IAD before a constant dip in flights which continues to the end of 2005. Flights to EWR, ORD have remained relatively constant. Flight to LGA & MCO has been increasing steadily. Flights to TPA has been steadily decreasing and flights to MIA was on an increasing trend until the second quarter of 2005 before it started to dip.

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

Year	Month	DayofMonth	DepTime	CRSDepTime	ArrTime	CRSArrTime	UniqueCarrier	TailNum	ArrDelay	DepDelay	Origin	Dest
2004	1	3	726	610	1112	920	UA	N326UA	112	76	GEG	DEN
2004	1	3	1208	1008	1503	1240	UA	N326UA	143	120	DEN	MCI
2004	1	3	1623	1330	1736	1456	UA	N326UA	160	173	MCI	ORD
2004	1	3	1854	1715	2200	1958	UA	N326UA	122	99	ORD	BOI

Figure 14

Yes, we are able to detect cascading failure. In Figure 14, United Airlines Flight N326UA was scheduled to depart GEG airport at 610 and arrive at DEN airport at 920, however a delay at GEG airport causes the flight to depart at 726, a delay of 76 minutes, this causes the aircraft to arrive at DEN airport at 1112, 112 minutes late. The same flight was then supposed to depart DEN airport at 1008 to MCI airport, however due to the delay and late arrival time, flight N326UA was delay by 120 minutes. This then cause the flight to arrive at MCI airport 143 minutes late causing a delay to the subsequent flight and so on. The closer we look into the data then more examples we can find that shows cascading failures.

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

In order to answer this, a model is built, seen in Figure 15. The dataset is split into a 70-30 ratio for training and testing and seed number 100 is given to ensure that the results are reproduceable. The data used is only data from the first month of year 2004 and also only from the top 10 airports with the most departing flights.

Figure 16 below shows the results of the summary and also shows that all the recorded forms of delay are a significant causes of flight delays since they all have p-values lesser than 0.001. Also, by seeing how the coefficient of the delays are greater than 0, naturally a conclusion can be made that increasing the delay time will result in an increased probability that the flight will be delayed.

```
set.seed(100)
n <- nrow(flight_sum)
train_idx <- sample(1:n, ceiling(0.7*n))
length(train_idx)

train <- flight_sum[train_idx, ]
test <- flight_sum[-train_idx, ]
dim(train)
```

Figure 15

```
Coefficients: (2 not defined because of singularities)
                  Estimate Std. Error z value Pr(>|z|)
(Intercept)     -3.960e+00  6.484e-02 -61.072  < 2e-16
Year                    NA         NA      NA       NA
Month                   NA         NA      NA       NA
DayofMonth      -1.967e-02  1.447e-03 -13.591  < 2e-16
DayOfWeek        1.807e-02  6.609e-03   2.735  0.00624
OriginCVG       -7.021e-02  5.404e-02  -1.299  0.19390
OriginDEN       -2.109e-01  6.442e-02  -3.273  0.00106
OriginDFW        3.289e-01  4.393e-02   7.486 7.09e-14
OriginEWR       -2.554e-01  6.499e-02  -3.930 8.50e-05
OriginIAH       -5.206e-01  6.080e-02  -8.562  < 2e-16
OriginLAS        1.604e-01  5.763e-02   2.782  0.00540
OriginLAX       -5.874e-02  5.384e-02  -1.091  0.27524
OriginORD        5.639e-01  4.308e-02  13.088  < 2e-16
OriginPHX        3.512e-01  5.386e-02   6.521 6.97e-11
CRSDepTime       6.232e-04  2.879e-05  21.643  < 2e-16
CarrierDelay     2.419e-01  2.316e-03 104.413  < 2e-16
WeatherDelay     2.269e-01  4.757e-03  47.704  < 2e-16
NASDelay         3.204e-02  7.145e-04  44.845  < 2e-16
SecurityDelay    2.285e-01  1.680e-02  13.595  < 2e-16
LateAircraftDelay 2.781e-01 3.571e-03 77.878  < 2e-16
```

Figure 16

```
> #Area under the ROC curve
> roc_rf$auc
Area under the curve: 0.946
```

Figure 17

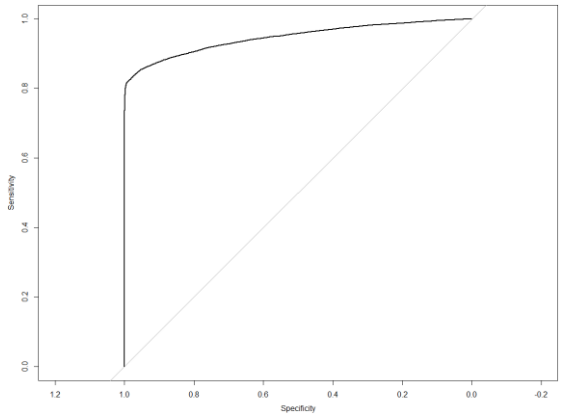

Figure 18

Next, a random forest model with 100 tress is created and given that the model has an AUC (area under the ROC curve) of 0.946. I can conclude that this model is good at predicting flight delays.

2. In Python

2.1. Data Cleaning

```
#removed unused columns to make dataset smaller
ontime_db = ontime_db.drop(columns=['ActualElapsedTime','FlightNum', 'CRSElapsedTime', 'AirTime','Distance',
                                    'TaxiIn', 'TaxiOut', 'Diverted','CancellationCode', 'CarrierDelay',
                                    'WeatherDelay', 'NASDelay', 'SecurityDelay', 'LateAircraftDelay'])
ontime_db = ontime_db[ontime_db['Cancelled'] != 1] #drop flights that have been cancelled to make sample size smaller
```

Figure 19

Flights that have been cancelled as well as unused columns are dropped to make the data frame smaller easier to handle.

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

```
ontime_db['Delay'] = np.where(ontime_db['DepDelay'] >= 15, 1, 0 )
flight_sum = ontime_db
flight_sum_delay = ontime_db[ontime_db['Delay']!= 0]
flight_sum['CRSDepHour'] = flight_sum.CRSDepTime // 100 % 24
flight_sum_delay['CRSDepHour'] = flight_sum.CRSDepTime // 100 % 24
```

Figure 20

A new column in data frame *ontime_db* created call *Delay.* next, a new data frame called *flight_sum* and *flight_sum_delayed* is created. *flight_sum_delayed* contains only the data of flights that have been delayed. In both *flight_sum* and *flight_sum_delayed*, a new column id created called *CRSDepHour*, this shows the hour that all the flights depart at.

```
ax=sns.countplot(x='CRSDepHour', hue='Delay', data=flight_sum)
ax.set_ylabel('Count')
ax.set_xlabel('Time (in 24Hr format)')
ax.title.set_text("Distribution of Flights & Delays By Time of Day")
L = plt.legend()
L.get_texts()[0].set_text('Flghts')
L.get_texts()[1].set_text('Delayed')
```

Figure 21: Data Manipulated to Show Best Day to Fly

```
ax=sns.countplot(x='DayOfWeek', hue='Delay', data=flight_sum)
ax.set_ylabel('Count')
ax.set_xlabel('Day of Week')
ax.title.set_text("Distribution of Flights & Delays By Day of Week")
L = plt.legend()
L.get_texts()[0].set_text('Flghts')
L.get_texts()[1].set_text('Delayed')
```

Figure 22: Data Manipulated to Show Best Week to Fly

```
ax=sns.countplot(x='Month', hue='Delay', data=flight_sum)
ax.set_ylabel('Count')
ax.set_xlabel('Month')
ax.title.set_text("Distribution of Flights & Delays By Month")
L = plt.legend()
L.get_texts()[0].set_text('Flghts')
L.get_texts()[1].set_text('Delayed')
```

Figure 23: Data Manipulated to Show Best Month to Fly

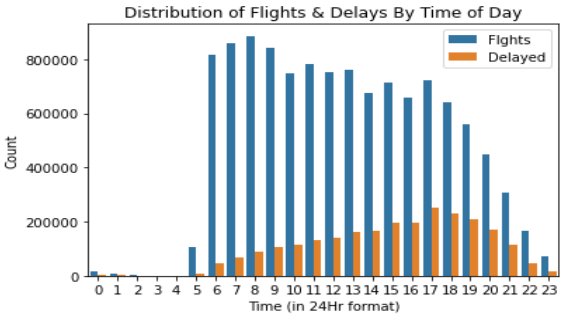

Figure 24: Graph of Distribution of Flights & Delays by Time of Day

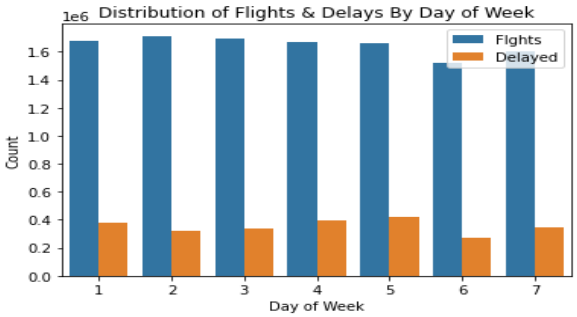

Figure 25: Graph of Distribution of Flights & Delays by Day of Week

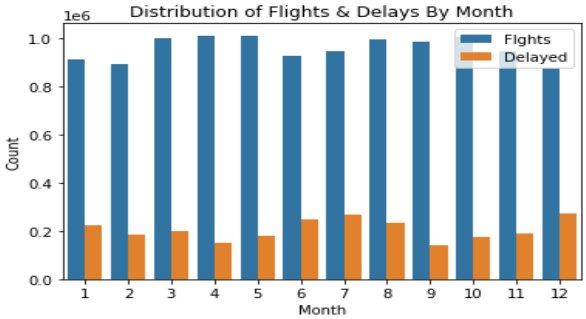

Figure 26: Graph of Distribution of Flights & Delays by Month

Upon viewing the plots, from Figure 24, we are able to conclude that flights departing between 6AM and 9AM has a lower ratio of delayed flight to total flights and therefore would be the better time to fly to avoid delays. From Figure 25, we are able to conclude that the ratio of delayed flights to total flights on Tuesdays and Wednesday is lower as compared to other days and therefore would be the better days to travel to try and avoid delays. Lastly, from Figure 26, it would appear that months April and September has some of the lowest ratio of delayed flight to total flights and therefore would be the better month to travel to try and avoid delays.

2.3. Do Older Planes Suffer More Delays?

```
planes_df1=planes[planes['type'].isnull()==False]
older_planes_delay = ontime_db.merge(planes_df1[['tailnum','year']],how='left',left_on='TailNum',right_on='tailnum')
```

Figure 27

As with answering the question in R, a new data frame named *older_planes_delay* is created. This data frame is formed by merging *ontime_db* and *planes* by the columns *tailnum*.

```
older_planes_delay.replace([np.inf, -np.inf], np.nan, inplace=True)
older_planes_delay = older_planes_delay.fillna(0)
older_planes_delay.loc[(older_planes_delay!=0).any(1)]
older_planes_delay['ArrDelay'] = older_planes_delay['ArrDelay'].astype(int)
older_planes_delay['Delayed_Total']= np.where((older_planes_delay['DepDelay']>=15) |(older_planes_delay['ArrDelay']>=15),1,0)
chi,p =chisquare(np.array(older_planes_delay.groupby(['year'])['Delayed_Total'].sum()))
```

Figure 28

Next, any rows or columns with *inf* values is changed to a *NA* value and then dropped to ensure better results. *'ArrDelay'* column is then converted to *int* format before a new column called *'Delayed_Total'* is added. A 1 represents that either a departure or an arrival delay has occurred with a 0 represents that there was no delay in both departure and arrival. A chi-squared analysis is then done between the sum of the total delay that occurred for that particular year against each year.

```
P value 0.000000
There sufficient evidence to reject null hypothesis
```

Figure 29

This then gives a p-value of less the 0.001. We therefore reject the null hypothesis and conclude that older planes do suffer more delay as compared to new planes.

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

```
q1_2004_db = ontime_db.loc[(ontime_db['Year']==2004) & (ontime_db['Month'].between(1,3,inclusive = True)) & (ontime_db['Origin']=='ATL')]
q2_2004_db = ontime_db.loc[(ontime_db['Year']==2004) & (ontime_db['Month'].between(4,6,inclusive = True)) & (ontime_db['Origin']=='ATL')]
q3_2004_db = ontime_db.loc[(ontime_db['Year']==2004) & (ontime_db['Month'].between(7,9,inclusive = True)) & (ontime_db['Origin']=='ATL')]
q4_2004_db = ontime_db.loc[(ontime_db['Year']==2004) & (ontime_db['Month'].between(10,12,inclusive = True)) & (ontime_db['Origin']=='ATL')]
q1_2005_db = ontime_db.loc[(ontime_db['Year']==2005) & (ontime_db['Month'].between(1,3,inclusive = True)) & (ontime_db['Origin']=='ATL')]
q2_2005_db = ontime_db.loc[(ontime_db['Year']==2005) & (ontime_db['Month'].between(4,6,inclusive = True)) & (ontime_db['Origin']=='ATL')]
q3_2005_db = ontime_db.loc[(ontime_db['Year']==2005) & (ontime_db['Month'].between(7,9,inclusive = True)) & (ontime_db['Origin']=='ATL')]
q4_2005_db = ontime_db.loc[(ontime_db['Year']==2005) & (ontime_db['Month'].between(10,12,inclusive = True)) & (ontime_db['Origin']=='ATL')]
```

Figure 30

As with the method done in R, only flights departing from Atlanta Airport will be observed as there will be too much data to go through if every flight is taken into consideration. Multiple new data frames are created separating the data into four quarters of both year 2004 and 2005. The data is then sorted by destinations with the highest traffic and the top 10 locations are selected.

```
fig, ax = plt.subplots(2, 4, figsize=(30, 15))
ax[0][0].bar(q1_2004.index, q1_2004.values, alpha = 0.6,width = 0.5, color = my_palette)
ax[0][0].set_ylabel('Number of Travellers')
ax[0][0].set_xlabel('Airport')
ax[0][0].title.set_text("Top 10 Destinations from Atlanta in the First Quarter of 2004")
```

Figure 31

Next, all the data is plotted and labelled into a 2x4 plot as seen in Figure 32. After analysing the plot, we can observe that with exception of the third quarter of 2004 and the last quarter in 2005, DFW airport has constantly been the most location. In the third quarter of 2004, there was a large and sudden increase to flight to IAD before a constant dip in flights which continues to the end of 2005. Flights to EWR, ORD have remained relatively constant. Flight to LGA & MCO has been increasing steadily. Flights to TPA has been steadily decreasing and flights to MIA was on an increasing trend until the second quarter of 2005 before it started to dip.

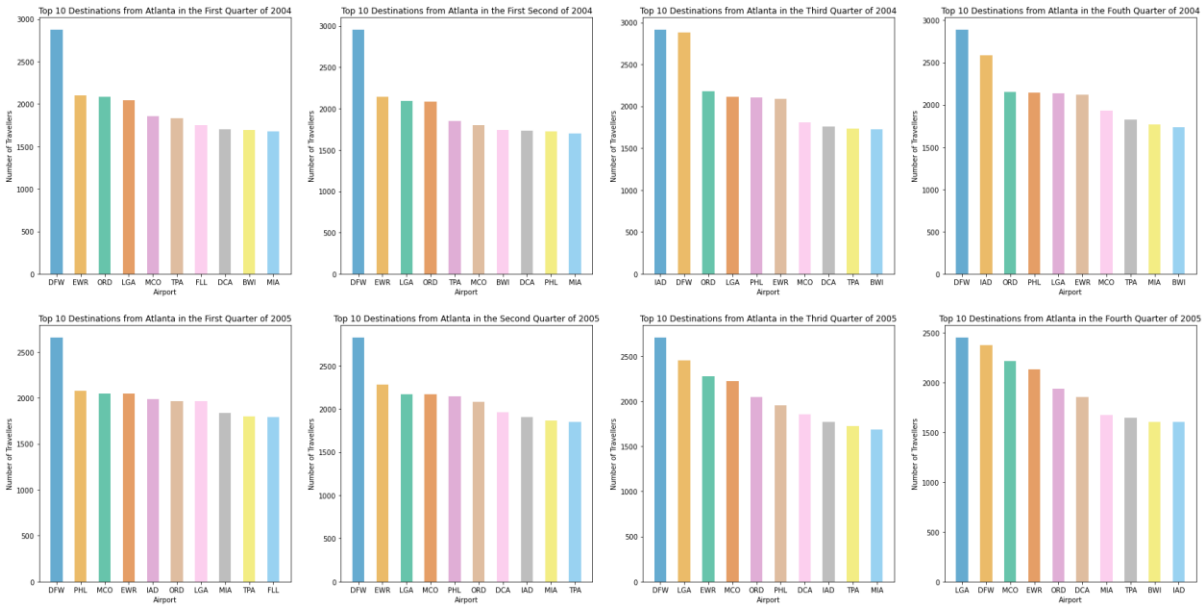

Figure 32

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

Year	Month	DayofMonth	DepTime	CRSDepTime	ArrTime	CRSArrTime	UniqueCarrier	TailNum	Origin	Dest
2004	1	3	808	735	959	918	UA	N344UA	BOI	DEN
2004	1	3	1045	1005	1540	1452	UA	N344UA	DEN	CLE
2004	1	3	1617	1537	1639	1607	UA	N344UA	CLE	ORD
2004	1	3	1730	1700	2019	1945	UA	N344UA	ORD	IAH

Figure 33

Cascading failures can be detected as delays in one airport causes delays in another airport. Using the example seen in Figure 33 above, we can track United Airline number N344UA's journey. A 33 minute departure delay from BOI airport caused a 41 minute arrival delay at DEN airport. This subsequently cause the next flight to CLE airport to be delayed by 40 minutes and also the flights thereafter. Therefore, we can conclude that cascading failure can be detected.

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

```
from sklearn.model_selection import train_test_split
train_x, test_x, train_y, test_y = train_test_split(features.drop('Delay', axis=1),
                                                    features['Delay'], test_size=0.3, random_state=42)
```

Figure 34

I first split the data into a 70 – 30 ratio for the training set and test set using the random state of 42 to ensure it is reproducible. Next, 100 trees were grown using the RandomForestClassifier model. This resulted in a model score of 0.944 (Figure 35).

```
In [209]: model.score(test_x, test_y)
Out[209]: 0.9440226264306609
```

Figure 35

	coef	std err	z	P>|z|	[0.025	0.975]
Month	0.0116	0.001	15.722	0.000	0.010	0.013
DayOfWeek	0.0205	0.001	16.037	0.000	0.018	0.023
DayofMonth	0.0049	0.000	16.953	0.000	0.004	0.005
CRSDepTime	0.0006	5.64e-06	113.992	0.000	0.001	0.001
CarrierDelay	0.2329	0.000	511.016	0.000	0.232	0.234
WeatherDelay	0.2326	0.001	190.363	0.000	0.230	0.235
NASDelay	0.0310	0.000	234.248	0.000	0.031	0.031
SecurityDelay	0.2296	0.004	56.805	0.000	0.222	0.238
LateAircraftDelay	0.2854	0.001	372.334	0.000	0.284	0.287
Origin_ATL	-4.1903	0.014	-307.953	0.000	-4.217	-4.164
Origin_CVG	-4.3829	0.015	-288.868	0.000	-4.413	-4.353
Origin_DEN	-4.5891	0.016	-285.395	0.000	-4.621	-4.558
Origin_DFW	-4.3181	0.014	-310.056	0.000	-4.345	-4.291
Origin_EWR	-4.4100	0.016	-279.835	0.000	-4.441	-4.379
Origin_IAH	-4.7260	0.016	-304.180	0.000	-4.756	-4.696
Origin_LAS	-3.9880	0.015	-264.736	0.000	-4.018	-3.958
Origin_LAX	-4.3891	0.015	-297.204	0.000	-4.418	-4.360
Origin_ORD	-4.1340	0.014	-304.690	0.000	-4.161	-4.107
Origin_PHX	-3.9411	0.015	-270.557	0.000	-3.970	-3.913

Figure 36

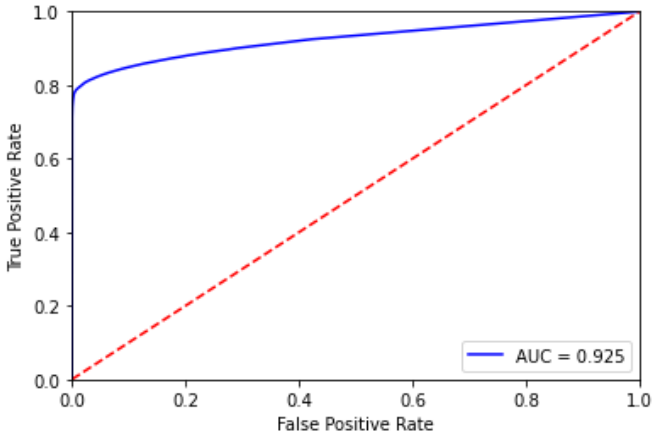

Figure 37

Figure 30 show the result and summary. As all the forms of recorded delay have p-values lesser than 0.001, it can be concluded that they are all a significant cause of delay. Also, as they coefficient are all greater than 0, it can be concluded that an increase in any form of delays will cause an increase to the flight being delayed.

Lastly, as the area under the ROC curve is 0.925, I can conclude that this model is good at predicting flight delays.

Appendix and References

1. Yen Tran. , 2019. Predicting Flight Delays and Cancellations – Report, [online]

   Available at: https://rpubs.com/tranyen/496061 [Accessed 23 February 2022]

2. Fabien Daniel. , 2018 Predicting flight delays [Tutorial], [online]

   Available at: < https://www.kaggle.com/code/fabiendaniel/predicting-flight-delays-tutorial/notebook> [Accessed 2 February 2022]