ST2195 Programming for Data Science Coursework
DataSet Analysis

Saraf Vidhi

210500099

Singapore Institute of Management

3rd April 2023

Table of Contents

Data Sorting And Wrangling

To analyze the datasets from “The 2009 ASA Statistical Computing and Graphics Data Expo”, we first imported the necessary packages to execute the code efficiently. Next, we uploaded and read the files using "pd.read_csv", and combined the data from the years 2005 and 2006. We then identified unique values, missing values, and the data types present to perform a detailed analysis. To handle missing data, we used the interpolate() function to fill in the gaps, enabling more efficient data processing.

Q1. When is the best time of day, day of the week, and time of year to fly to minimise delays?

1.1 Python

To determine the best time, day, and year of arrival for a set of flights, the delays between the actual arrival and departure times compared to the scheduled departure time needed to be analyzed. In order to accomplish this, the "CRSDepTime" and "CRSArrTime" columns were first converted into a time format, allowing for a better understanding of the delay distribution. The "CRSDepHour" column was also created by extracting the first two integers of the "CRSDepTime" column using the slice method. This step was taken to round off the departure times to the nearest hour, making it easier to create a graph of the delays by hour of the day.

To generate a graph of the delays by day of the week, the weeks were converted from numbers to the names of the days. Similarly, the months were converted from numbers to the names of the month to create a graph of the delays by month of the year. This technique allowed for the creation of more visually appealing and accessible graphs, making it easier to understand and interpret the data. The departure and arrival time was taken from “DepDelay” and “ArrDelay”

```
# Convering data into time format
df1['CRSDepTime'] = df1['CRSDepTime'].apply(Timeformat)
df1['CRSArrTime'] = df1['CRSArrTime'].apply(Timeformat)
df1[['CRSDepTime', 'CRSArrTime']].head()

# Slicing the Departure time to hour format
df1['CRSDepHour'] = df1['CRSDepTime'].apply(lambda x: x[:2])
df1['CRSArrHour'] = df1['CRSArrTime'].apply(lambda x: x[:2])
```

Part of code used to obtain the results for the following question

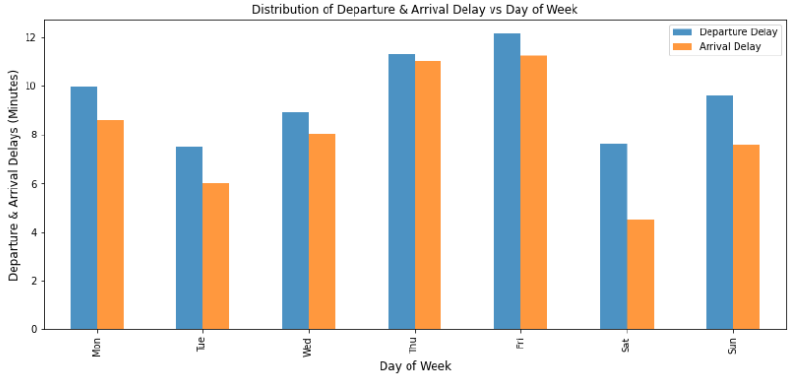
	CRSDepTime	CRSArrTime	CRSDepHour
0	16:05	17:59	16
1	16:05	17:59	16
2	16:10	18:05	16
3	16:05	17:59	16
4	19:00	22:32	19

Output obtained from running thecode on the left

BEST TIME IN THE DAY TO TRAVEL

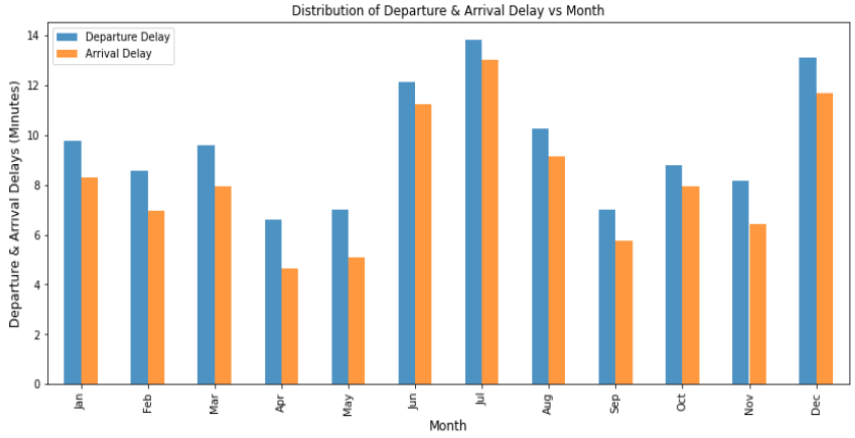
As per the results above we see that the best time to travel is between 05:00 to 07:00.

BEST DAY IN THE WEEK TO TRAVEL



As per the graph plotted above the best day of the week to travel is Saturday.

BEST MONTH TO TRAVEL



As per the graph plotted above the best month to travel is April or May.

1.2 R Analysis

For R we tried to use the same method leading us to data and analysis similar to that of the data above.

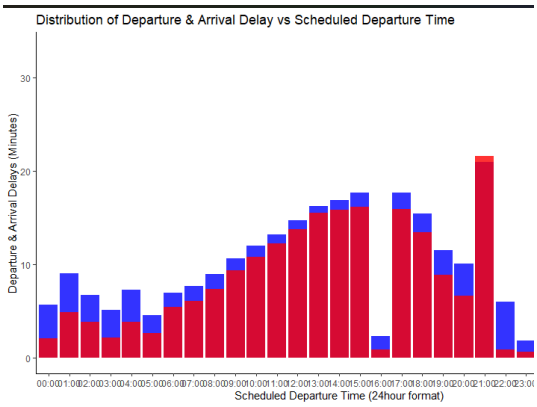
```
# Converting CRSDepTime, CRSArrTime into time format
df1$CRSDepTime <- sapply(df1$CRSDepTime, Timeformat)
df1$CRSArrTime <- sapply(df1$CRSArrTime, Timeformat)
df1[,c("CRSDepTime", "CRSArrTime")]


# Converting Scheduled Departure time into hours format
df1$CRSDepHour <- substr(df1$CRSDepTime, 1, 2)
df1$CRSArrHour <- substr(df1$CRSArrTime, 1, 2)
```

Part of code used to obtain the results for the following question

CRSDepTime	CRSArrTime	CRSDepHour
<chr>	<chr>	<chr>
16:05	17:59	16
16:05	17:59	16
16:10	18:05	16
16:05	17:59	16
19:00	22:32	19
19:00	22:32	19
19:00	22:32	19
19:00	22:32	19
19:00	22:23	19
19:00	22:23	19

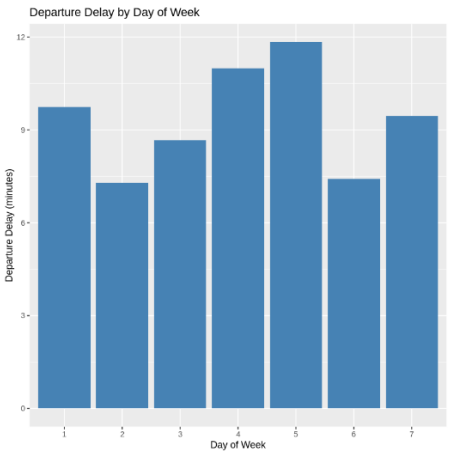
Output obtained from running the pat of the code on the left

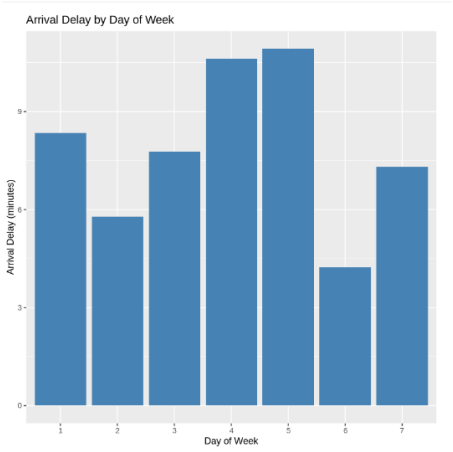
BEST TIME IN THE DAY TO TRAVEL



As per the results above we see that the best time to travel is between 05:00 to 07:00.

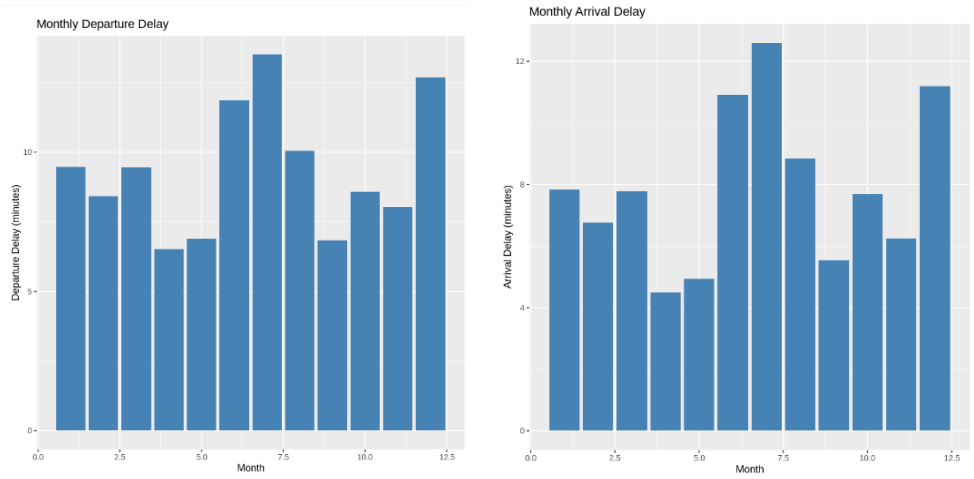
BEST DAY IN THE WEEK TO TRAVEL





As per the graph plotted above the best day of the week to travel is Saturday.

BEST MONTH TO TRAVEL



As per the graph plotted above the best month to travel is April or May.

Q2.Do older planes suffer more delays?

In order to assess the potential impact of aging planes on flight delays, we performed an analysis that required integrating data from the "planes.csv" Excel sheet with our pre-existing data set. In Python we merged the "TailNum" column from "planes.csv" with "df," which enabled a comprehensive analysis of pertinent variables. To prevent any potential ambiguity arising from an existing "years" column in our "df" data set, we swiftly renamed the "years" column in "planes.csv" to "YearOfManufacture."We then added the sum of the “DepDelay” and “ArrDelay” according to the “YearOfManufacture”, in order to get the average delay in time. Similarly, in R, we renamed the column names prior to merging the primary data frame, "df_planes," with the "planes_data" data frame using the "left_join()" function. The resulting merged data set was then utilized to create advanced visualizations, thereby enabling us to discern significant patterns and trends in flight delays, and to present the findings in a visually appealing format.

2.1Python

```
# Sum of Depature Delay and Arrival Delay respectively
df2planes_sum = df2_planes.groupby('YearOfManufacture')[['DepDelay','ArrDelay']].sum()
```

```
df2planes_sum.head()
df2planes_sum = df2planes_sum.reset_index()
```

Arrival Delay vs Year of Manufacture

Departure Delay Vs Year of Manufacture

2.2 R Analysis

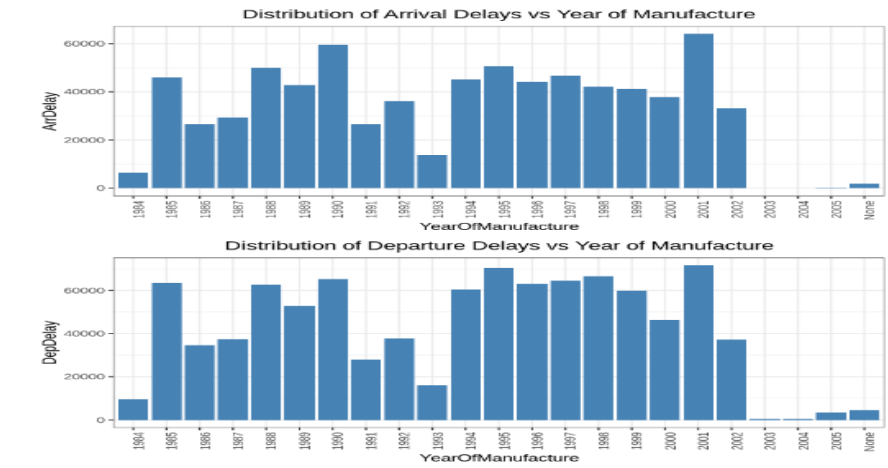
```
[ ] # Merge df with df_planes dataset
    colnames(df_planes)[colnames(df_planes)=="year"] <- "YearOfManufacture"
    colnames(df_planes)[colnames(df_planes)=="tailnum"] <- "TailNum"
    df2_planes <- merge(df, df_planes[c("TailNum","YearOfManufacture")], by="TailNum", all.x=TRUE)
```

```
[ ] # Sum of Departure Delay and Arrival Delay respectively
    df2planes_sum <- aggregate(cbind(DepDelay, ArrDelay) ~ YearOfManufacture, data=df2_planes, FUN=sum)
```

```
[ ] # Distribution of Arrival Delays vs Year of Manufacture
    p1 <- ggplot(df2planes_sum, aes(x=YearOfManufacture, y=ArrDelay)) +
      geom_bar(stat="identity", fill="steelblue") +
      theme_bw() +
      ggtitle("Distribution of Arrival Delays vs Year of Manufacture") +
      theme(plot.title = element_text(hjust = 0.5)) +
      theme(axis.text.x = element_text(angle = 90, hjust = 1))
```

```
[ ] # Distribution of Departure Delays vs Year of Manufacture
    p2 <- ggplot(df2planes_sum, aes(x=YearOfManufacture, y=DepDelay)) +
      geom_bar(stat="identity", fill="steelblue") +
      theme_bw() +
      ggtitle("Distribution of Departure Delays vs Year of Manufacture") +
      theme(plot.title = element_text(hjust = 0.5)) +
      theme(axis.text.x = element_text(angle = 90, hjust = 1))
```

```
# Combine the two plots
grid.arrange(p1, p2, nrow=2)
```



The bar chart shows that the majority of delays are associated with newer aircraft models. Based on our analysis, we conclude that newer aircraft models are primarily responsible for the majority of delays, and older planes do not suffer from more delays. Interestingly, our results also suggest that the latest models made post 2002 have shown less delay.

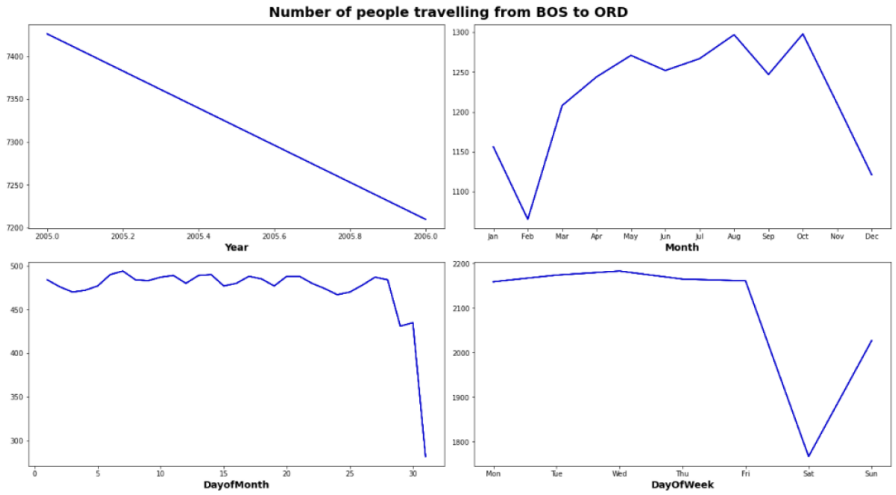
Furthermore, the bar chart highlights that the highest number of flight delays occurs in planes that were manufactured between the years of 1994 and 2002. Conversely, flight delays from planes manufactured between the years 1990 to 1993 and post-2002 were significantly lower.

Q3. How does the number of people flying between different locations change over time?

To gain a deeper understanding of the data, we limit our focus to “BOS” and “ORD”, disregarding the rest of the dataset due to its vastness. After restricting our analysis to only include flights originating from Boston (BOS) and arriving at Chicago (ORD), we proceed to calculate the number of passengers traveling between the two cities over different time intervals. To better visualize these trends, we construct a graph that plots the changes in passenger volume over time. This enables us to discern any notable patterns or shifts in passenger traffic between Boston and Chicago.

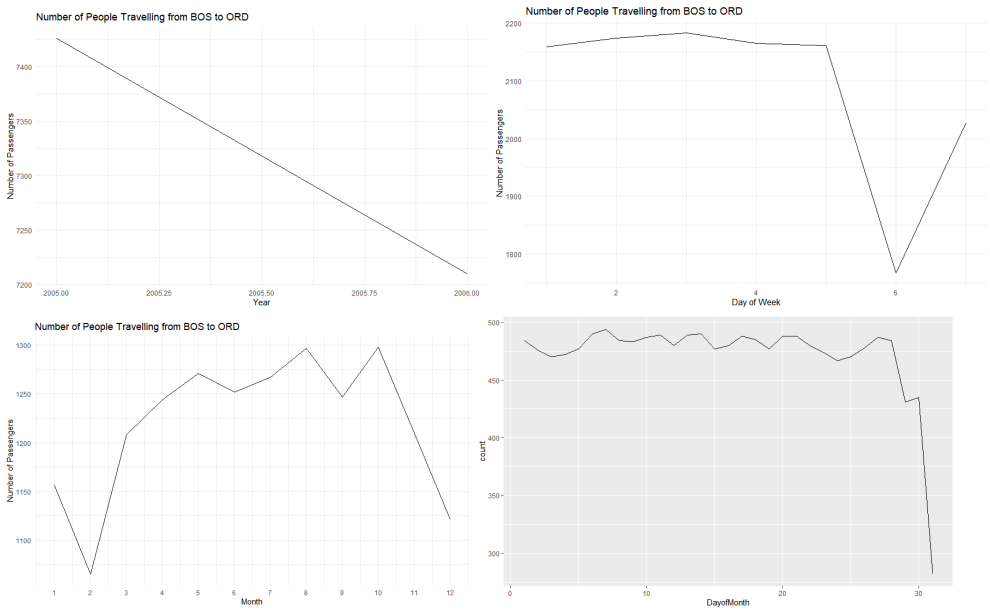
3.1 Python

```
fig, axs = plt.subplots(2, 2, figsize=(18, 10))
axs[0, 0].plot(df3_location.groupby(['Year']).count(), color='mediumblue')
axs[0, 0].set_xlabel('Year', fontsize=14, weight = "bold")
axs[0, 1].plot(df3_location.groupby(['Month']).count().reindex(daysofmonth), color='mediumblue')
axs[0, 1].set_xlabel('Month', fontsize=14, weight = "bold")
axs[1, 0].plot(df3_location.groupby(['DayofMonth']).count(), color='mediumblue')
axs[1, 0].set_xlabel('DayofMonth', fontsize=14, weight = "bold")
axs[1, 1].plot(df3_location.groupby(['DayOfWeek']).count().reindex(daysofweek), color='mediumblue')
axs[1, 1].set_xlabel('DayOfWeek', fontsize=14, weight = "bold")
plt.suptitle('Number of people travelling from BOS to ORD', fontsize=20, weight = "bold")
plt.tight_layout()
plt.show()
```



3.2 R Analysis

```
# Finding the count of people travelling from the city of Boston(BOS) to Chicago(ORD)
df3_location <- df3[df3$Origin == 'BOS' & df3$Dest == 'ORD', ]
n_passengers <- nrow(df3_location)
# Plotting the number of passengers by year, month, day, and day of week
ggplot(df3_location, aes(x = Year)) +
  geom_line(stat = 'count') +
  labs(x = 'Year', y = 'Number of Passengers', title = 'Number of People Travelling from BOS to ORD') +
  theme_minimal()

ggplot(df3_location, aes(x = Month)) +
  geom_line(stat = 'count') +
  scale_x_continuous(breaks = 1:12) +
  labs(x = 'Month', y = 'Number of Passengers', title = 'Number of People Travelling from BOS to ORD') +
  theme_minimal()

ggplot(df3_location, aes(x = DayofMonth)) +
  geom_line(stat = 'count') +
  labs(x = 'Day of Month', y = 'Number of Passengers', title = 'Number of People Travelling from BOS to ORD') +
  theme_minimal()

ggplot(df3_location, aes(x = DayOfWeek)) +
 geom_line(stat = 'count') +
  labs(x = 'Day of Week', y = 'Number of Passengers', title = 'Number of People Travelling from BOS to ORD') +
  theme_minimal()
```



Based on the graphs presented above, it is apparent that there was a notable decline in the number of people traveling from BOS to ORD between 2005 and 2006. The peak period for travel was observed in August and October, with approximately 1,300 passengers recorded during these months. In contrast, February saw the lowest number of travellers, with only around 1,000 passengers recorded. On a daily basis, there was relatively consistent traffic between the two cities, with approximately 450 passengers recorded each day. However, there was a sharp decline in passenger volume observed around the 27th of each month. Finally, the data revealed that passenger traffic remained relatively constant throughout the week, with the exception of Saturday, which experienced a significant decline in the number of travellers.

Q4. Can you detect cascading failures as delays in one airport create delays in others?

To extract the required dataset, we begin by making a copy of the original dataset and selecting the relevant columns:
'Date','UniqueCarrier','TailNum','Origin','Dest','Distance','CRSDepTime','DepTime','DepDelay','CRSArrTime','ArrTime','ArrDelay'. We then sort the "Origin" and "Dest" columns to only include flights originating from Boston (BOS) and arriving at Chicago (ORD).

To study the impact of delays on individual flights, we filter the dataset to only include flights with a positive "ArrDelay" and "DepDelay" and select one of the identified "TailNum" values, in this case, "N935UA". By analyzing the delay data for this specific aircraft, we can better understand how delays impact individual flights and the potential ripple effect they can have on subsequent flights.

4.1 Python

```
df4_analysis = df4_data[(df4_data['ArrDelay'] > 0) & (df4_data['DepDelay'] > 0)]
```

```
df4_analysis[(df4_analysis['TailNum']=='N326UA')].head(2)
```

```
df4_analysis[(df4_analysis['Origin'] == "ORD") & (df4_analysis['TailNum']=='N326UA')].head(2)
```

	Date	UniqueCarrier	TailNum	Origin	Dest	Distance	CRSDepTime	DepTime	DepDelay	CRSArrTime	ArrTime	ArrDelay
37	2005-01-09	UA	N326UA	BOS	ORD	867	1705	1853.0	108.0	1902	2138.0	156.0
42	2005-01-14	UA	N326UA	BOS	ORD	867	1705	1712.0	7.0	1902	1919.0	17.0

	Date	UniqueCarrier	TailNum	Origin	Dest	Distance	CRSDepTime	DepTime	DepDelay	CRSArrTime	ArrTime	ArrDelay
60	2005-01-09	UA	N326UA	ORD	SAT	1041	1950	2205.0	135.0	2250	133.0	163.0
65	2005-01-14	UA	N326UA	ORD	SAT	1041	1950	2042.0	52.0	2250	2335.0	45.0

4.2 R Analysis

```
#Q4. Can you detect cascading failures as delays in one airport create delays in others?
# Make a copy of dataset for Q4
df4 <- df
df4$Month <- sapply(df4$Month, function(x) as.numeric(format(strptime(x, "%b"), "%m")))
# Adding a date column to the dataset
df4$Date <- as.Date(paste(df4$Year, df4$Month, df4$DayofMonth, sep="-"))
df4 <- df4[, c("Date", colnames(df4)[!(names(df4) %in% c("Year", "Month", "DayofMonth"))])]
head(df4)
# Subsetting columns required to test for cascading failures
df4_data <- df4[, c("Date", "UniqueCarrier", "TailNum", "Origin", "Dest", "Distance", "CRSDepTime", "DepTime",
"DepDelay", "CRSArrTime", "ArrTime", "ArrDelay")]
head(df4_data)
df4_analysis <- subset(df4_data, ArrDelay > 0 & DepDelay > 0)
head(subset(df4_analysis, TailNum == "N326UA"), 2)
head(subset(df4_analysis, Origin == "ORD" & TailNum == "N326UA"), 2)
```

This figure illustrates flight data for Tail Number “N326UA” on January 9th, 2005. The scheduled departure time for the flight from 'BOS' to 'ORD' was 1705 (5.05 pm), but it departed at 1853 (6.53 pm), resulting in a departure delay of 108 minutes. Similarly, the scheduled arrival time was 1902 (7.02 pm), but the actual arrival time was 2138 (9.38 pm), leading to an arrival delay of 156 minutes. Furthermore, Flight "N326UA" scheduled to depart from 'ORD' to 'SAT' on the same date was delayed by 135 minutes, with the scheduled departure time of 1950 (7.50 pm) and the actual departure time of 2205 (10.05 pm). This highlights that there is an impact of the departure delay at 'BOS' airport on the subsequent departure delay at 'ORD' airport on the same day.

Q5. Use the available variables to construct a model that predicts delays.

To begin with, the dataframe was subset to include only the relevant variables and the first 10,000 rows. A new column named 'DepartureDelay' was created with binary values of 0 or 1 based on the presence of delays in the 'DepDelay' column. The independent variables were defined, and the 'DepartureDelay' column was set as the dependent variable. The data was then split into a 70:30 ratio of train and test sets. The logistic regression model was fitted on the train set and used to make predictions on the test set. The performance of the model was evaluated using the classification report module from sklearn.metrics.

5.1 Python

```
# Model Evaluation using Confusion Matrix
conf = metrics.confusion_matrix(y_test, y_predict)
conf
```

```
array([[1576,    6],
       [   8, 1410]], dtype=int64)
```

```
# Checking Classification Report
model_summary = classification_report(y_test, y_predict)
print(model_summary)
```

```
              precision    recall  f1-score   support

           0       0.99      1.00      1.00      1582
           1       1.00      0.99      1.00      1418

    accuracy                           1.00      3000
   macro avg       1.00      1.00      1.00      3000
weighted avg       1.00      1.00      1.00      3000
```

```
print("Accuracy:",metrics.accuracy_score(y_test, y_predict))
print("Precision:",metrics.precision_score(y_test, y_predict))
print("Recall:",metrics.recall_score(y_test, y_predict))
```

```
Accuracy: 0.9953333333333333
Precision: 0.9957627118644068
Recall: 0.9943582510578279
```

5.2 R Analysis:

```
# Model Evaluation using Confusion Matrix
conf <- table(y_test, y_predict)
conf
```

```
      y_predict
y_test    0    1
     0 1548    0
     1    0 1452
```

```
# Checking Classification Report
model_summary <- caret::confusionMatrix(factor(y_predict, levels = c(0,1)), factor(y_test, levels = c(0,1)))
print(model_summary)
```

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 1548    0
         1    0 1452

               Accuracy : 1
                 95% CI : (0.9988, 1)
    No Information Rate : 0.516
    P-Value [Acc > NIR] : < 2.2e-16
```