ST 2195
Coursework Report

Student Number: 220572875

Contents

Part 1

For part 1a), the task was to simulate random numbers for a given probability density function. After setting up the required values, a “for loop” was used to generate random numbers for the Normal Distribution and Uniform Distribution, and the results are visualized in Figure 1. The histogram and kernel density plot (red line) represents the values of the Metropolis-Hastings algorithm, while the blue line represents the values of the original probability density function. The mean and standard deviation of the algorithm are also displayed accordingly

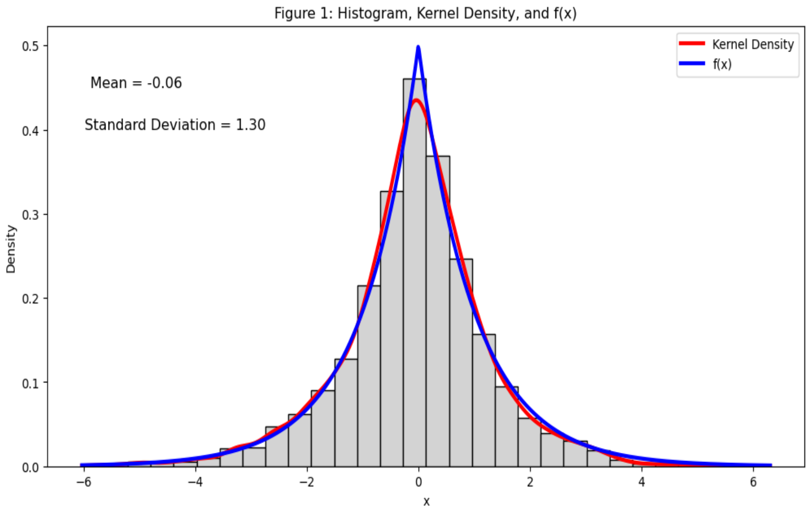


Figure 1: Histogram, Kernel Density, and f(x)

For part 1b), the task was to find the *R-hat* values for the Metropolis Hastings algorithm. A “for loop” was created to generate J sequences of length N, and inside this for loop, the “for loop” in part 1a) was used to generate random numbers. The required mean and variance in the question are then calculated, and these are all placed inside a function. A vector is then created to store the *R-hat* values over *s* values which will be done using another “for loop”. Figure 2 visualizes the results. The generated *R-hat* values gradually converges over the *s* values (increment set to 0.01).

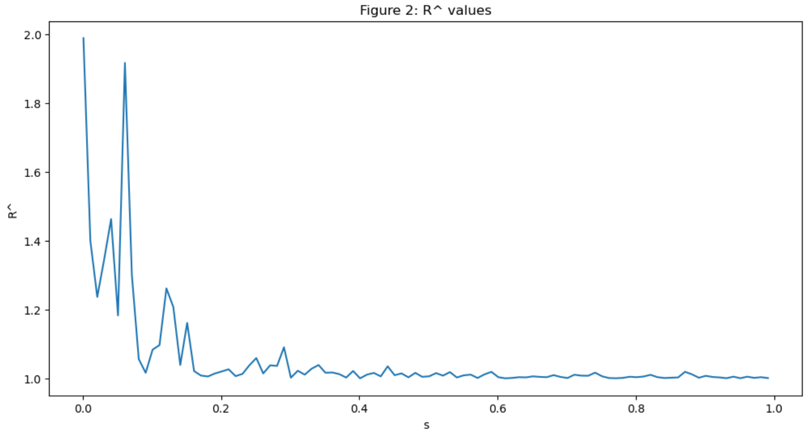


Figure 2: R^ values

Part 2

Introduction

The purpose of this report is to use the flight data provided by the 2009 ASA Statistical Computing and Graphics Data Expo to answer the following questions using R and Python:

a) What are the best times and days of the week to minimise delays each year?
b) Evaluate if older planes suffer more delays on a year-to-year basis.
c) Fit a logistics regression each year for the probability of diverted US flights using as many features as possible from attributes of departure date, scheduled departure date and arrival times, coordinates and distance between departure and planned arrival airports and the carrier. Visualize the coefficients across years.

For this report, data from the years 1995 to 2004 will be used to answer the questions. As the same operations were done in R and Python, **the report shall only display the graphs which are better looking of the two languages.**

Part 2a)

Before answering this question, a database named "dataverse.db" was created in DB Browser to store 10 consecutive years of data available in the dataset given. The years chosen are 1995 to 2004 and only these 10 years are loaded into the database. Four tables are created to store data, namely:

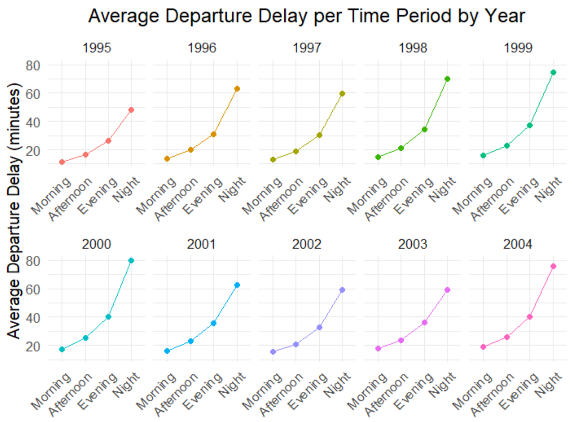
- "airports", this table contains data about airports such as location, IATA code.
- "carriers", this table contains data about carriers and their carrier code.
- "flights", this table contains flight data from 1995 to 2004.
- "planes", this table contains data about planes such as tail number and manufacturer.

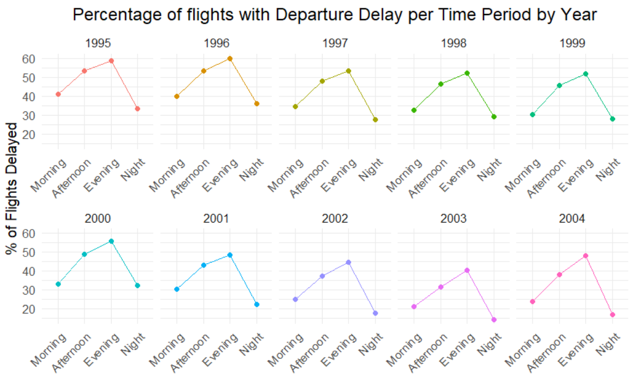
Average departure delay is used to answer this question because departure delay directly affects arrival delay, and from the point of view of a traveller, getting delayed on the ground is much more frustrating than being delayed while in the air because at least they are still "travelling" while in the air.

After this, we set the working directory to where the data is located, relevant packages are loaded into both R and Python, and SQL queries are used to fetch the data from the database. To summarise the SQL queries for the best times to minimise delays, the departure time for flights was fetched and grouped into "Morning", "Afternoon", "Evening" and "Night", depending on the values, while other invalid values are grouped into "Invalid". Departure time from 0600hrs to 1200 hrs is grouped as "Morning", 1200hrs to 1800hrs as "Afternoon", etc. When fetching the data, only flights that are not cancelled, not diverted, departure delay > 0 and grouping is not "Invalid" are taken. This is then used to find the average departure delay for each period of the day. A similar query is used to fetch the data for best days of the week to minimise delays, except days are grouped into "Mon", "Tue", "Wed" etc. A similar query is also used to find the percentage of delayed flights for each period of the day as well as day of the week, but instead of average departure delay, the number of flights with departure delay is expressed as a percentage of the total number of flights. Results are visualized using ggplot2 for R and seaborn + matplotlib for Python, shown below.

Visualisation

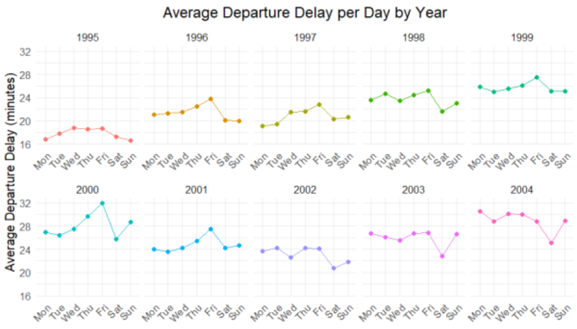
Average Departure Delay per Time Period / Percentage of Delayed Flights

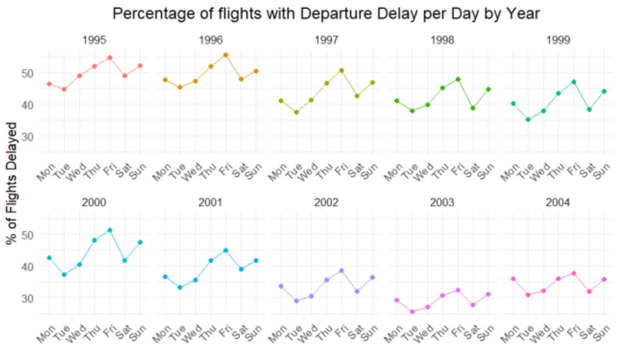




The facet grid on the left shows the average departure delay for each year. It can be seen from the line chart for each year that average departure delay is the lowest in the *Morning* and increases over time till *Night* for all 10 years. From this chart, it can be deduced that the best times to minimise delays will be to travel in the *Morning*. However, if we look at the facet grid on the right, it shows the percentage of delayed flights for each period of the day, and for all 10 years, *Night* has the lowest percentage of delayed flights. So, even though the average departure delay for *Night* is the highest, *Night* has the lowest percentage of having delayed flights. As such, if a traveller wants to minimise average departure delay, he should travel during *Morning*, but if he is trying to reduce the chances of having delayed flights, he should travel during *Night*.

Average Departure Delay per Day / Percentage of Delayed Flights





The facet grid on the left shows us that for all the years with 1995 and 1996 as the exception, the lowest average departure delay is on *Saturday* and the highest is on *Friday*. Therefore, if the traveller is looking to minimise average departure delay, he should be travelling on *Saturday*. The facet grid on the right shows us the percentage of delayed flights for each day. *Tuesday* has the lowest percentage of delayed flights, while *Friday* has the highest.

Conclusion

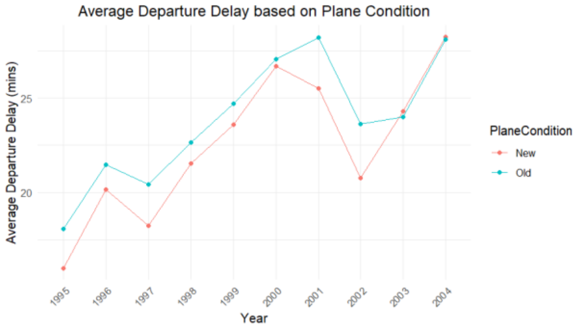
Based on the visualizations above, it can be concluded that that if a traveller wants to minimise his delay when travelling, the best time to travel will be on *Saturday Morning*. If the traveller wants to minimise the chances of having delays when travelling, the best time to travel will be *Tuesday Night*.

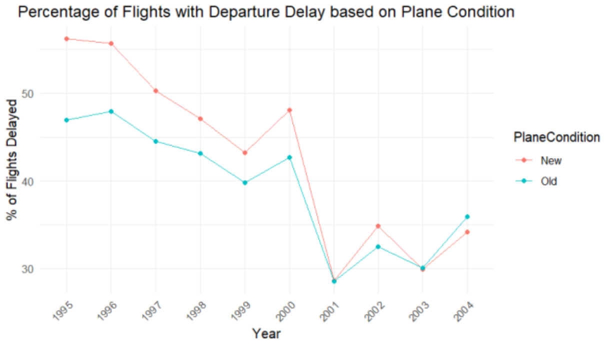
Part 2b)

The same database in Part 2a) is used to answer this question. To fetch the data from the database, an SQL query which combines “flights” table and “planes” table is created. These two tables are joined by the column “TailNum” which can be found in both tables. The ‘year’ column in the planes table is assumed to be the manufacturing year of the plane, and planes manufactured in the year 1988 and before are considered old planes. Thereafter, the query also takes the average departure delay of the planes, and removes the values 'None’, ‘0’ and NULL values. It also ensures that the departure delay is more than zero, the flights are not cancelled nor diverted. The same thing is done to find out the percentage of flights delayed due to the plane condition, but instead taking average departure delay, the number of flights with departure delay is expressed as a percentage of the total number of flights. Then, the result of the query is visualized.

Visualization

Average Departure Delay based on Plane Condition / Percentage of Delayed Flights





The graph on the left shows us that older planes have consistently experienced more average departure delays than newer planes, and as the years goes by, the average departure delays of both old and new planes have been increasing. It can be deduced from this graph that older planes have longer average departure delays compared to newer planes. Interestingly, the graph on the right shows us that older planes have lower percentage of flights delayed, and the percentage of flights being delayed has being dropping over the 10 years. This may be due to other factors such as increased efficiency of air transport management and technological advancements.

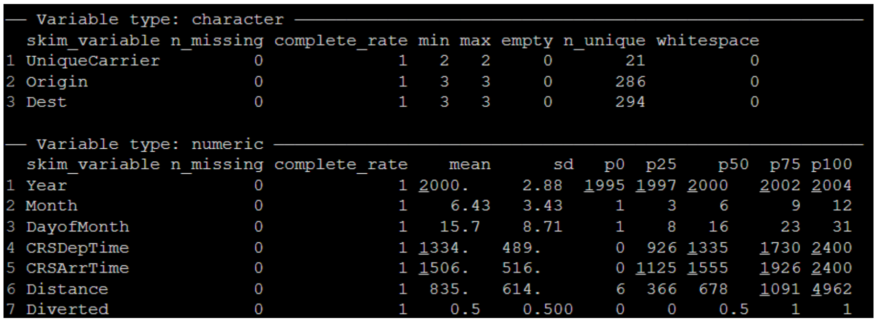
Conclusion

Based on the results above, it can be concluded that *older planes suffer longer average departure delays than newer planes over the years*, but *older planes do not suffer more departure delays than newer planes.*

Part 2c)

For this question, “dataverse.db” is also used. Starting off, we set the working directory to where the data is located, and the relevant packages are imported into both R and Python. Due to the large dataset, only a sample of flight data is used to fit the logistic regression model. An SQL query is first used to fetch all the diverted flights with the features ‘Year’, ‘Month’, ‘DayofMonth’, ‘CRSDepTime’, CRSArrTime’, ‘Distance’, ‘UniqueCarrier’, ‘Origin’, ‘Dest’ and ‘Diverted’. This query also removes flights that are cancelled and with NULL distance value. Next, another query is used to randomly fetch the same number of non-diverted flights, with the rest of the conditions remaining the same. These two queries are put into separate data frames before being combined into a single data frame. This data frame will be the sample that is used to fit the logistic regression model for each year, adding up to 248162 rows of data.

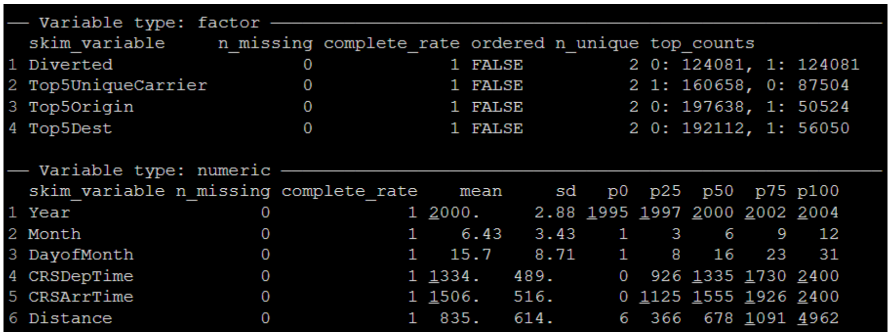
Then, the skim() function from the skimr library for R (.info() for Python) is used to view the structure of the data frame, and it shows that ‘UniqueCarrier’, ‘Origin’ and ‘Dest’ are characters while the rest are numeric.



```
── Variable type: character ──────────────────────────────────────────────────
  skim_variable n_missing complete_rate min max empty n_unique whitespace
1 UniqueCarrier         0             1   2   2     0       21          0
2 Origin                0             1   3   3     0      286          0
3 Dest                  0             1   3   3     0      294          0

── Variable type: numeric ────────────────────────────────────────────────────
  skim_variable n_missing complete_rate    mean      sd   p0  p25    p50  p75 p100
1 Year                  0             1 2000.     2.88 1995 1997 2000   2002 2004
2 Month                 0             1    6.43   3.43    1    3    6      9   12
3 DayofMonth            0             1   15.7    8.71    1    8   16     23   31
4 CRSDepTime            0             1 1334.   489.       0  926 1335   1730 2400
5 CRSArrTime            0             1 1506.   516.       0 1125 1555   1926 2400
6 Distance              0             1  835.   614.       6  366  678   1091 4962
7 Diverted              0             1    0.5    0.500    0    0    0.5    1    1
```

Before changing these three variables into factors, the top five most occurring values of the variables are extracted. This is so that when they are changed into factors, it becomes Top5 = 1, while the rest are = 0. As such, one-hot encoding is performed on these three variables, and they are converted into factors along with ‘Diverted’. Then, skim() is used again to confirm the changes have been made.

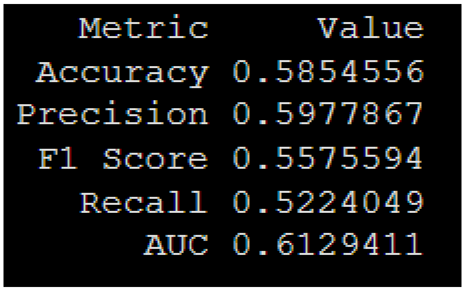


```
── Variable type: factor ─────────────────────────────────────────────────────
  skim_variable     n_missing complete_rate ordered n_unique top_counts
1 Diverted                  0             1 FALSE          2 0: 124081, 1: 124081
2 Top5UniqueCarrier         0             1 FALSE          2 1: 160658, 0: 87504
3 Top5Origin                0             1 FALSE          2 0: 197638, 1: 50524
4 Top5Dest                  0             1 FALSE          2 0: 192112, 1: 56050

── Variable type: numeric ────────────────────────────────────────────────────
  skim_variable n_missing complete_rate    mean     sd   p0  p25  p50  p75 p100
1 Year                  0             1 2000.     2.88 1995 1997 2000 2002 2004
2 Month                 0             1    6.43   3.43    1    3    6    9   12
3 DayofMonth            0             1   15.7    8.71    1    8   16   23   31
4 CRSDepTime            0             1 1334.   489.      0  926 1335 1730 2400
5 CRSArrTime            0             1 1506.   516.      0 1125 1555 1926 2400
6 Distance              0             1  835.   614.      6  366  678 1091 4962
```

The data is then split into training and test sets, with 70% of the data being the training set and 30% being the test set.

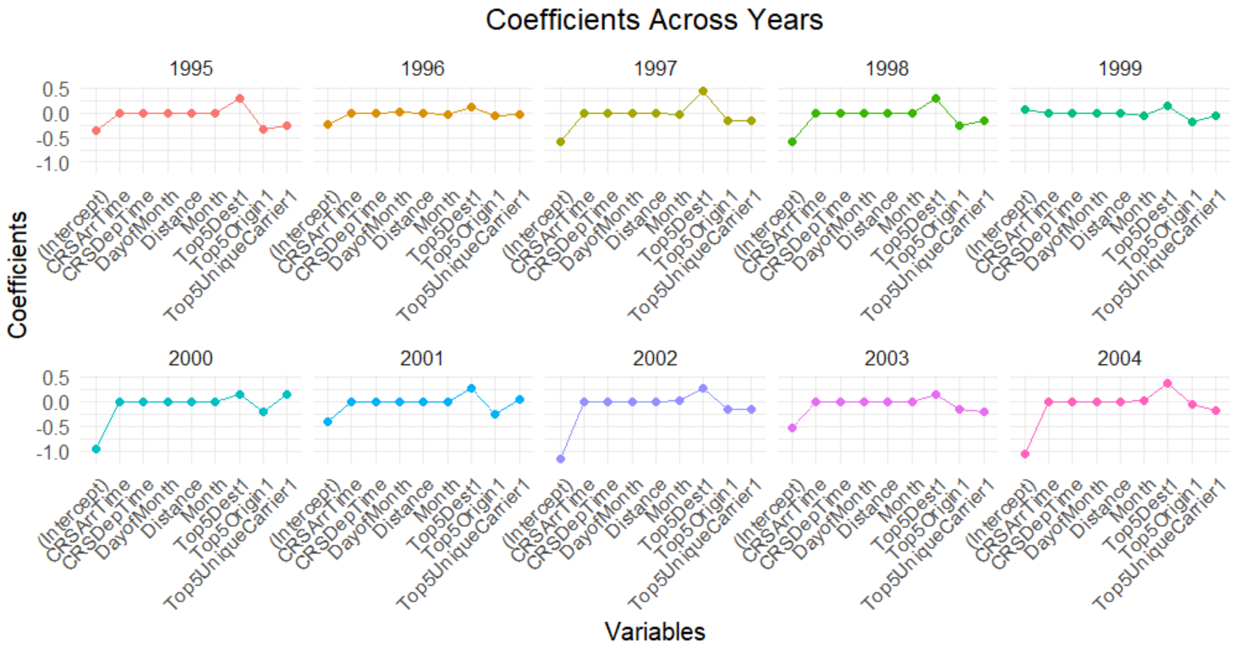
Now, we can run the logistics regression model and obtain the evaluation metrics of the model.



```
   Metric     Value
 Accuracy 0.5854556
Precision 0.5977867
 F1 Score 0.5575594
   Recall 0.5224049
      AUC 0.6129411
```

From the metric score, it can be inferred that there is an approximate 58.5% chance that the model correctly predicts whether a flight is diverted. When it does predict a flight is diverted, there is a 59.7% chance that it is correct. The model can correctly identify 52.2% of flights that are diverted, and a F1 score of 0.557 mean that the model is has reasonable ability to correctly identify flights that are actually diverted vs those that are labelled to be diverted but actually not. An AUC score of 0.612 means that the model can differentiate between diverted and non-diverted flights, but the ability can be improved.

To get the coefficients of the model, a function is created before running the model again, but this time the data used is a subset of the training data where Year = year. Then the 'Year', variable names and coefficient values are combined into a data frame so that we can continue with plotting the graph. The result is shown below:



It can be inferred from the graph that 'CRSArrTime', 'CRSDepTime', 'DayofMonth', 'Distance' and 'Month' barely affect whether a flight is diverted or not since their coefficients are very close to 0. 'Top5Dest', 'Top5Origin' and 'Top5UniqueCarrier' seem to be able to affect whether a flight is diverted or not.