ST2195 Coursework Project Aug 2024

Student ID 220165356

Table of Contents

Please refer to codes available in the attached Python and RMarkdown files.

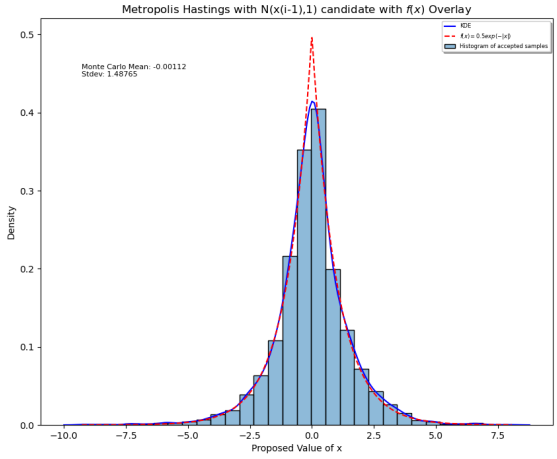
Part 1: Metropolis Hastings

Metropolis Hastings (MH) is a classic method of sampling using the Monte Carlo Markov chain (MCMC) method to predict the posterior distribution by generating random numbers for the distribution with probability density function, f(x) = 1/2 exp(−|x|). The steps taken include:

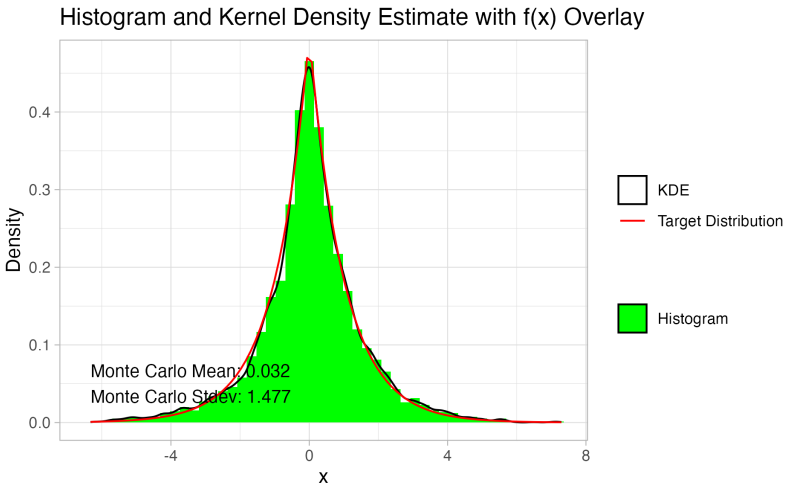
1. We start by selecting a candidate for the new value of x randomly between 0 and 1, x_new.
2. Define the acceptance ratio(A). Using the given ratio A = min(1, $P(x_*)/P(x_{i-1})$).
3. Generate another uniform random number (u) between 0 and 1.
4. If log(u)<log(A), we accept the new value and set x=x_new, else x will be the previous value. Log is used to maintain the stability.
5. Record the final value of x for this sample. Using the samples to plot histogram and kernel density plot.
6. Repeat the steps above by generating chains, keep on sampling while discarding the initial burn in phase(N=1000) as the chain has not reached its stationary phase.

A) The diagrams below represents MH performed with N=10000, s = 1 and the generated samples (without burn in)

Python representation:

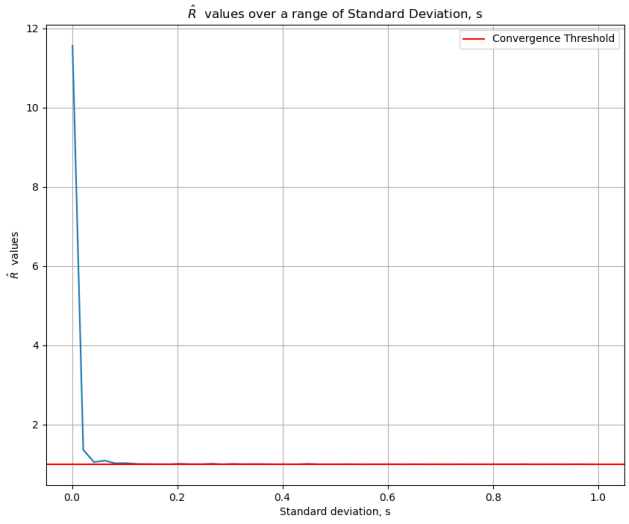


RStudio representation:

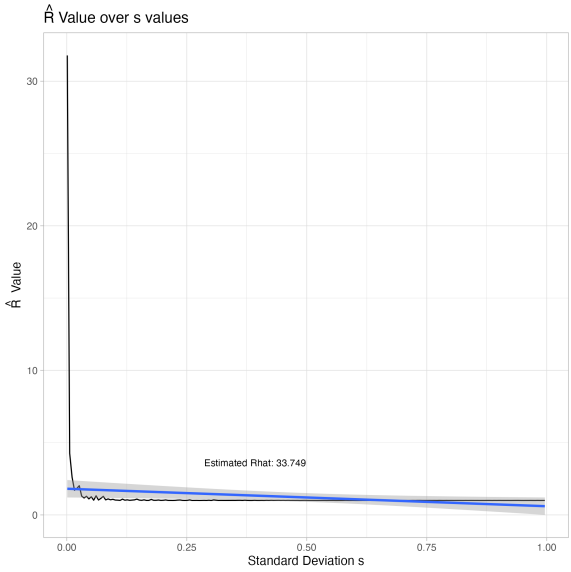


B) Graphs below shows the convergence of Rhat to 1 that shows a high level of correlation and that it is a suitable model.

Python representation:



RStudio representation:



Part 2: Data Analysis of commercial flights in the USA

Step 1 – Import necessary libraries (stated inside the Python and Markdown)

Step 2 – Load the data: I have chosen data from 1999.csv to 2008.csv, and its supplementary information downloaded from *Harvard Dataverse* at https://doi.org/10.7910/DVN/HG7NV7.

Step 3 – Identify/Handling missing values by removing null values.

Step 4 – Verify missing values

Step 5 – Perform Regression and further analysis

****Please note the codes are in bold, refer to Python and RMarkdown files for more detailed coding****

1. Setup and connection to SQLite3

A "Data10yrs.db" database in DB SQLite has been created to facilitate these datasets. For connection, please see the RMarkdown and Jupyter notebooks for steps. To connect:

Python: conn = sqlite3.connect('Data10yrs.db')

R: conn <- dbConnect(RSQLite::SQLite(), dbname = 'Data10yrs.db')

2. Data cleaning and shaping

A. Sampling of data

Python: Looking at the 10 years consecutive data 1999.csv to 2008.csv, I have used random sampling of the 10 years data by taking subsets of each year using approximate percentage. An example for data '1999.csv' is below, and is done for all ten datasets.

```
n = 100  # every 100th line = 1% of the lines
df_1999 = pd.read_csv('1999.csv', header=0, skiprows=lambda i: i % n != 0)
```

R: Using random sampling, a total of 10,000 samples are selected .

```
subset_of_years <- read_csv(all_years_files, id="file", n_max = 100000,skip_empty_rows = TRUE, show_col_types=FALSE)
```

B. Creating new dataset

A new dataset (subset_of_years) consisting of the 10 sampled data has been created. This will be the main source of data to be used.

C. Removing null values

After loading the dataset as a dataframe, it is important to check the data for its integrity by looking for any error or null values.

Python: subset_of_years = subset_of_years.dropna(axis=0, thresh = 26)

R: df_2b_clean <- na.omit(df_2b)

Question 2A: What are the best times and days of the week to minimise delays each year?

Using the "variable-descriptions.csv" folder to match the variables and information, to answer question 2A we will need the Month, DayofWeek, CRSDepTime, ArrDelay and DepDelay.

Using SQL query codes, to find the best times with minimal delay for each year:

```
timeOfYear=c.execute('''
   SELECT Year,CRSDepTime, COUNT(CRSDepTime) AS Frequency, ArrDelay, DepDelay
   FROM subset_of_years
   WHERE DepDelay = 0 AND ArrDelay = 0
   GROUP BY Year
   ORDER BY Frequency DESC
   ''').fetchall()
```

The best days of week for each year with minimal delay:

```
DayOfWeek=c.execute('''
   SELECT Year, DayOfWeek, COUNT(DayOfWeek) AS Frequency, ArrDelay, DepDelay
   FROM subset_of_years
   WHERE DepDelay = 0 AND ArrDelay = 0
   GROUP BY Year
   ORDER BY Frequency DESC
   ''').fetchall()
```

Q2A: Data Visualisation & Analysis

Python best times of year

	Year	CRSDepTime	COUNT	ArrDelay	DepDelay
0	2003	1045	1803	0.000000	0.000000
1	2004	1358	1316	0.000000	0.000000
2	2005	637	1186	0.000000	0.000000
3	1999	1630	1054	0.000000	0.000000
4	2000	945	964	0.000000	0.000000
5	2001	621	918	0.000000	0.000000
6	2002	1255	847	0.000000	0.000000
7	2006	1930	558	0.000000	0.000000
8	2007	940	298	0.000000	0.000000
9	2008	1100	89	0.000000	0.000000

R_best_times

	Year	CRSDepTime	Frequency	ArrDelay	DepDelay
1	2002	755	5241	0	0
2	2006	1005	4456	0	0
3	2001	940	3971	0	0
4	2003	1400	3962	0	0
5	2004	840	3620	0	0
6	2005	1400	3074	0	0
7	2000	1932	2059	0	0
8	1999	635	2046	0	0
9	2007	725	438	0	0
10	2008	850	417	0	0

Python best day of year

	Year	DayOfWeek	COUNT	ArrDelay	DepDelay
0	2003	Wednesday	1803	0.000000	0.000000
1	2004	Saturday	1316	0.000000	0.000000
2	2005	Monday	1186	0.000000	0.000000
3	1999	Sunday	1054	0.000000	0.000000
4	2000	Tuesday	964	0.000000	0.000000
5	2001	Friday	918	0.000000	0.000000
6	2002	Tuesday	847	0.000000	0.000000
7	2006	Monday	558	0.000000	0.000000
8	2007	Tuesday	298	0.000000	0.000000
9	2008	Saturday	89	0.000000	0.000000

R_day_of_week

	Year	DayOfWeek	Frequency	ArrDelay	DepDelay
1	2002	Thursday	5241	0	0
2	2006	Tuesday	4456	0	0
3	2001	Saturday	3971	0	0
4	2003	Wednesday	3962	0	0
5	2004	Sunday	3620	0	0
6	2005	Monday	3074	0	0
7	2000	Monday	2059	0	0
8	1999	Wednesday	2046	0	0
9	2007	Monday	438	0	0
10	2008	Wednesday	417	0	0

For the results above, I have used subset of datasets from year 1999 to 2008, and created both SQL query results from Python and RMarkdown. It is observed from top two diagrams representing top 10 ranked Best Departure times per year , morning departures are more favourable with minimal delays, as highlighted in yellow on both diagrams. From Python result (top left), 6 out of 10 years have minimal delays with scheduled departure times in the morning. From RMarkdown result(top right), we have 7 out of 10 morning departures with no arrival and departure delays.

For the best day of the year, both results (bottom two diagrams) show an inconclusive and mixed result mostly comprising of weekdays instead of weekends, however but no specific weekday appears to be consistent. It means that it is better to travel on a weekday than on weekend to avoid delays, however no strong correlation.

The difference in the results is due to the different sampling method for the Python (approximate percentage, 1% of line) and RMarkdown where a bigger sample of 100,000 with random sampling is used.

These informations coincides to the benefits of waking up early in the morning to beat the traffic, also avoiding weekends where it is expected to have high volumes of people travelling on their day off.

Question 2B: Evaluate whether older planes suffer more delays on a year-to-year basis.

Step 1 – Import necessary libraries (stated inside the Python and Markdown)
Step 2 – Load the data: Data10yrs.db in SQLite3
Step 3 – Identify missing values
Step 4 – Handling missing values by removing all rows with more than 26 null variables.
Step 5 – Perform Polynomial Regression for each year and further analysis

Parameters

General Arrival/Departure delays are 15 minutes or more according to the US Federal Aviation Administration, thus selection will be for flights with ≥15mins delay in departure and arrival using the 'ArrDelay' and 'DepDelay' data from the datasets of all ten consecutive years 1999 to 2008.

Planes-data will be joined to each year data by 'Tailnum' and 'tailnum', and the age of the aircraft will be calculated using present year 2024 deducting its year of manufacture.

Data cleaning

Analysis is done on a year to year basis from 1999 to 2008. Data is drawn from 1999.csv to 2008.csv. The methods are described below. **Codes are in BOLD.**

Python:

Random sampling using approximate percentage, drawing every 100th line. (Where n=100,1%).

The data frame (first: df_1999) joins to plane-data using 'tailnum' and 'Tailnum'

```
age_1999 = pd.merge(df_1999, planes, left_on = 'TailNum', right_on = 'tailnum')
```

Find Aircraft age by deducting 'year' from plane.csv with present year 2024

```
age_1999['AircraftAge'] = (2024- age_1999['year'])
```

Data cleaning by removing NULL values

```
no_na_1999 = age_1999.dropna(axis=0, thresh = 26)
no_na_1999
```

Selecting relevant data with arrival and departure delays >=15mins

```
no_na_1999_delays = no_na_1999[['AircraftAge',
'ArrDelay','DepDelay']]
no_na_1999_delays
```

```
delay_1999 = no_na_1999_delays.loc[(no_na_1999_delays['ArrDelay'] >= 15) & (no_na_1999_delays['DepDelay'] >= 15)]
```

Plotting and visualising the data

R*emoving all null to prepare for regression*

```
final_1999 = delay_1999.dropna(axis= 0, how='any')
```

I have decided to evaluate the age of aircraft versus the arrival delays and departure delays more than 15mins, to study its correlation I have also included its polynomial regression line.

```
# polynomial regression line
coef = np.polyfit(final_1999['AircraftAge'], final_1999['ArrDelay'],1)
poly1d_fn = np.poly1d(coef)
```

R:

Random sampling has been done through all 10 years dataset 1999.csv to 2008.csv using this code:

```
subset_of_years <- read_csv(all_years_files, id="file", n_max = 100000,skip_empty_rows = TRUE, show_col_types=FALSE
```

Using SQL query and for loops, used arrival delay and departure delays and Age of aircraft (Present year - Year of Manufacture (year in planes.csv).
To loop through years:

```
list_of_years <- c("1999", "2000", "2001", "2002", "2003", "2004", "2005", "2006", "2007","2008")
```

Joined subset_of_years with planes table using INNER JOIN by tailnum

```
joined_tailnum <- dbGetQuery(conn,
"SELECT  subset_of_years.Year,
        planes.year AS YOM,
        subset_of_years.ArrDelay AS ArrivalDelay,
       subset_of_years.DepDelay AS DepartureDelay,
       (2024 - planes.year) AS AircraftAge
    FROM subset_of_years
    INNER JOIN planes ON subset_of_years.tailnum = planes.tailnum
    WHERE subset_of_years.ArrDelay >= 15 AND
subset_of_years.DepDelay >= 15 AND planes.year IS NOT NULL AND
planes.year != '0000' AND planes.year != '' AND planes.year != 'None'
AND planes.year > 0 AND Cancelled != 1
  ORDER BY AircraftAge DESC")
```

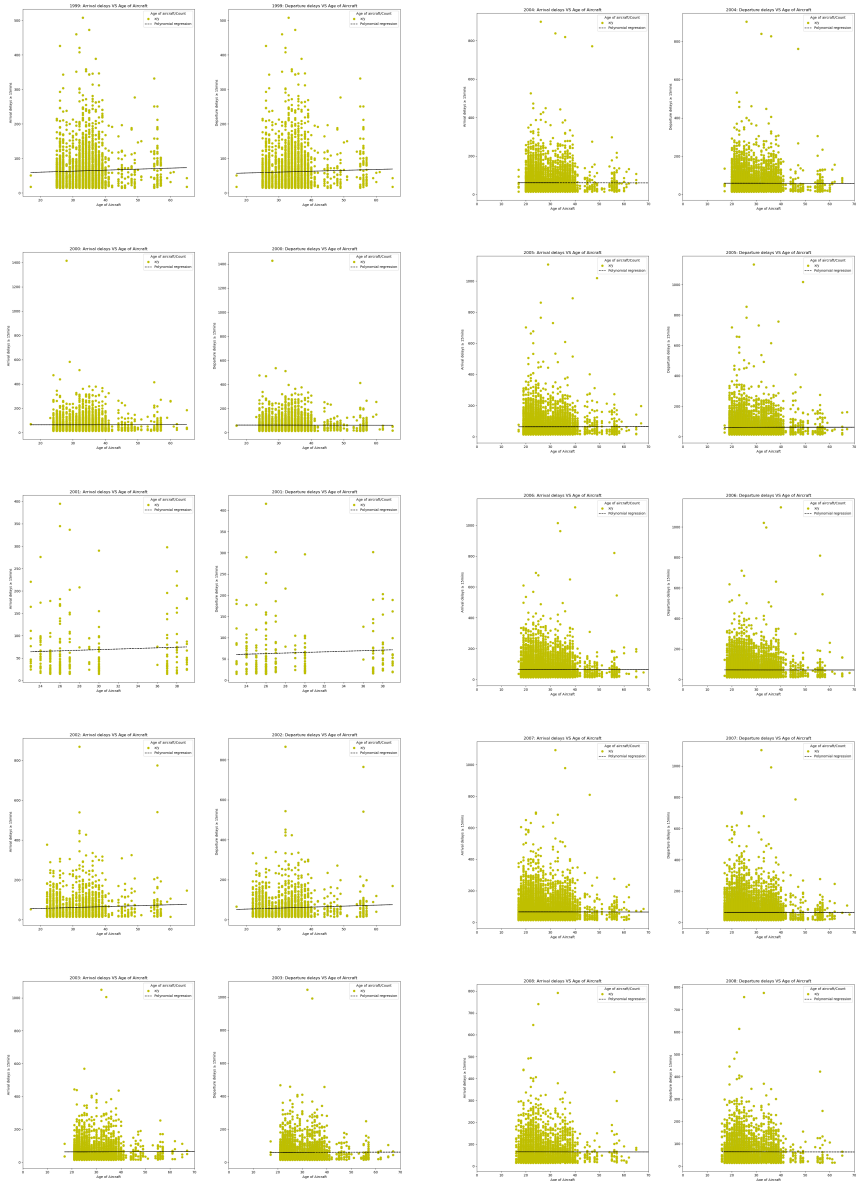
Cleaning data by removing null: **df_2b_clean <- na.omit(df_2b)**

SQL query code used:

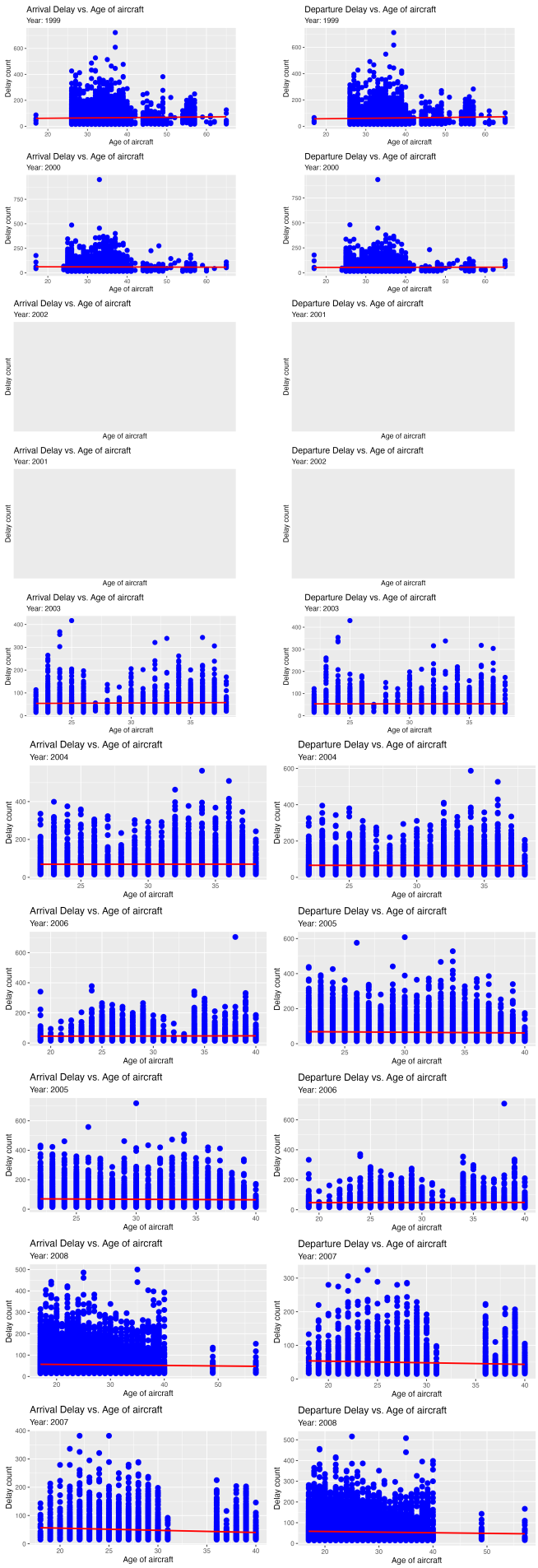
```
for (i in list_of_years) {
  p <- df_2b_clean %>%
  filter(Year == i) %>%
  ggplot(aes(AircraftAge,ArrivalDelay))+
  geom_point(color = "blue", size=3) +
  geom_smooth(method = "lm",color = "red") +
  labs(title = "Arrival Delay vs. Age of aircraft",
       subtitle = paste("Year:", i),
       x = "Age of aircraft",
       y = "Delay count")
```

Q2B: Data Visualisation & Analysis

Python results:



R results:



For Python results(plots in yellow), polynomial regression was chosen as the data points are mostly non-linear. In conclusion, based on the year on year analysis scatterplot with age of aircraft (x-axis) and Arrival delays and Departure delays as y-axis, the regression line is flat for all the graphs.

For R results(plots in blue), I have applied a regression line using the geom_smooth(method="lm") and the regression line is represented in red. It reiterates the result in Python where the polynomial regression line is flat. There was no data generated for Year 2001 and 2002, perhaps there were no significant delays in the sample set. This can also be seen in the faint plots on Python.

This shows that there is poor level of correlation between age of aircraft and delays. The reason for aircraft delays could be more probable due to weather disruptions for example during the winter months where deicing is required if snow is built up on the wings, or flooding of airports due to global warming effects.

Q2C: Logistic regression for the probability of diverted US flights using variables like departure date, scheduled departure, arrival times, the coordinates, distance between departure and planned arrival airports, and the carrier. Visualise the coefficients across the years.

Logistic regression is a powerful statistical way of modelling a binomial outcome, it models the estimate of probability p in terms of the predictor or explanatory variables x. Here the binary classification will be if the flight diverted (1) or not (0), and the other variables are departure date, the scheduled departure and arrival times, the coordinates and distance between departure and planned arrival airports, and the carrier. The logistic regression model predicts P(Y=1) as a function of X.

Step 1 – Import necessary libraries (stated inside the Python and Markdown)
Step 2 – Load the data: Data10yrs.db in SQLite3, using a combined dataset from randomly sampled to form a new data frame
Step 3 – Handling missing values by removing replacing most frequent category for categorical, median for numerical.
Step 5 – Perform Logistic Regression model and further analysis.

1. Load and preparation of data

I have selected as many features as indicated from the dataset, namely Diverted, CRSDepTime, CRSArrTime, Month, DayofMonth, Year, UniqueCarrier, Dest. Logistic regression is a method for fitting a regression curve, $y = f(x)$. We are to predict the values of y, using a set of predictors, x.

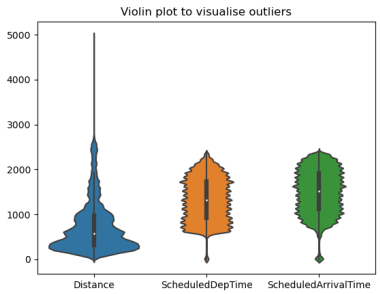
- Diverted: Diverted acts as y in the logistic regression formula. It is binary and classified as a factor.
- CRSDepTime & CRSArrtime: Scheduled departure and arrival times is changed to decimal place.
- DayofMonth, Month, Year: Combined to make the Departure date.
- Uniquecarrier: Carrier name used to as one of the attributes to study the probability of diversion ranking.

2. Data cleaning

I have used violin plot to observe for the presence of any outliers or abnormalities. The violin plot shows a tracing on Distance and it is normalised by doing scaling.

Python:

features_to_normalize = ['Distance']
MinMaxScaler(feature_range=(0, 1))
subset_of_years2[features_to_normalize] =
scaler.fit_transform(subset_of_years2[features_to_normalize])



R:

preProc <- preProcess(regression_tbl[, .(Distance)], method = c("range"))
regression_tbl[, Distance := predict(preProc, regression_tbl[, .
(Distance)])$Distance]

Below I have created a list to study the probability of diversion in terms of percentage , with all the attributes. They are ranked top 10 and bottom 10 using both R and Python. This enables us to see if there are any strong relationship , similarities or frequency patterns with the attributes and the probability of diversion. For the bottom 10 list it was hard to differentiate as there were plenty of flights that diverted only once. For the top 10 diversions, the airport STL in 1999 has significantly more diversion counts than the rest.

R_Top_ProbabilityDiversion

	Year	DivertedCount	COUNT(Diverted)	Airport	Departuredate	CRSDepTime	CRSArrTime	Distance	UniqueCarrier	prob
1	1999	1	78	STL	10612	1630	1950	934	WN	0.0016366612111293
2	2005	1	63	PHL	12805	830	1623	2378	US	0.0016366612111293
3	2006	1	42	DCA	13159	640	915	562	US	0.0016366612111293
4	2008	1	39	MDW	13885	1640	1805	395	WN	0.0016366612111293
5	2007	1	35	MCI	13525	1810	2100	718	WN	0.0016366612111293
6	2001	1	26	PHL	11343	620	706	105	US	0.0016366612111293
7	2004	1	26	MDW	12434	1040	1415	1121	WN	0.0016366612111293
8	1999	1	24	SFO	10593	810	1014	1504	UA	0.0016366612111293
9	1999	1	23	PIT	10610	945	1048	243	US	0.0016366612111293
10	1999	1	23	SLC	10614	840	1510	2125	DL	0.0016366612111293

R_bottom_ProbabilityDiversion

	Year	DivertedCount	COUNT(Diverted)	Airport	Departuredate	CRSDepTime	CRSArrTime	Distance	UniqueCarrier	prob
602	2008	1	1	PDX	13909	2100	2255	1009	WN	0.0016366612111293
603	2008	1	1	PVD	13885	720	1005	1137	WN	0.0016366612111293
604	2008	1	1	RSW	13905	1140	1530	1105	WN	0.0016366612111293
605	2008	1	1	SAN	13907	1820	2111	749	XE	0.0016366612111293
606	2008	1	1	SAT	13883	1455	1630	407	XE	0.0016366612111293
607	2008	1	1	SEA	13909	1515	1730	1107	WN	0.0016366612111293
608	2008	1	1	SFO	13903	1310	1555	1855	WN	0.0016366612111293
609	2008	1	1	SJC	13893	635	825	569	WN	0.0016366612111293
610	2008	1	1	TUL	13899	2035	2350	935	WN	0.0016366612111293
611	2008	1	1	TUS	13902	1522	1621	321	XE	0.0016366612111293

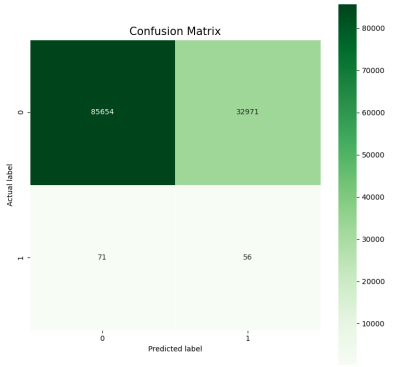
Python top 10 diversion

	Year	Index	DivertedCount	Airport	DepartureDate	ScheduledDepTime	ScheduledArrivalTime	Distance	Carrier	ProbabilityDivert%
0	2006	1	10	DFW	2006/4/7	1520	1833	1562	AA	0.014286
1	2006	1	10	IAH	2006/1/21	1515	1649	817	XE	0.014286
2	2004	1	8	ATL	2004/4/12	1851	2024	443	DL	0.011429
3	2004	1	8	IAH	2004/2/22	1845	1949	201	XE	0.011429
4	2007	1	8	DFW	2007/2/1	1700	1730	987	MQ	0.011429
5	2007	1	8	IAH	2007/3/14	1028	1443	1195	XE	0.011429
6	2005	1	6	MSP	2005/1/10	845	1419	1532	NW	0.008571
7	2007	1	6	DFW	2007/1/12	1410	1623	852	AA	0.008571
8	2005	1	5	DFW	2005/1/12	1500	1710	1068	AA	0.007143
9	2006	1	5	MSP	2006/1/27	845	951	528	NW	0.007143
10	2003	1	4	IAH	2003/7/24	615	833	1190	XE	0.005714

Python bottom 10 diversion

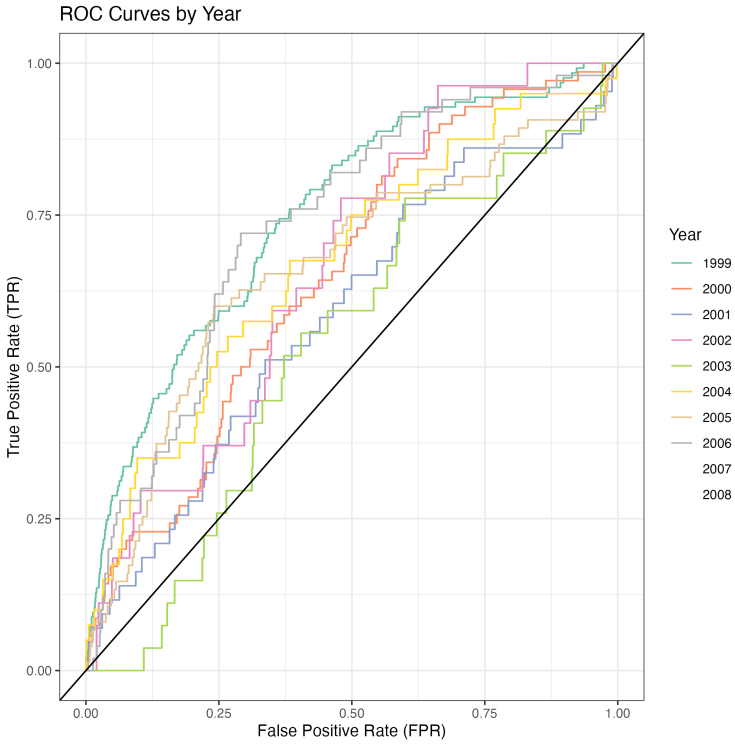
	Year	Index	DivertedCount	Airport	DepartureDate	ScheduledDepTime	ScheduledArrivalTime	Distance	Carrier	ProbabilityDivert%
483	2007	1	1	SEA	2007/9/2	830	1010	1024	UA	0.001429
484	2007	1	1	SFO	2007/10/17	900	1057	2036	DL	0.001429
485	2007	1	1	SFO	2007/1/7	1740	2100	2521	US	0.001429
486	2007	1	1	SHV	2007/6/8	1430	2400	285	9E	0.001429
487	2007	1	1	SJC	2007/2/16	650	820	342	AA	0.001429
488	2007	1	1	SJU	2007/3/7	1535	1923	2072	AA	0.001429
489	2007	1	1	SNA	2007/10/31	630	800	404	AQ	0.001429
490	2007	1	1	STL	2007/12/9	925	1211	1593	AA	0.001429
491	2007	1	1	SYR	2007/12/26	1031	1150	374	NW	0.001429
492	2007	1	1	SYR	2007/12/23	1510	1724	793	OH	0.001429
493	2007	1	1	XNA	2007/10/22	1840	1953	1147	MQ	0.001429

Python results:



Accuracy of predictions	
Accuracy	0.721756
Training set score	0.722090
Test set score	0.721756

R results:



R_coef_summary

AUC	TPR	FPR
Min. :0.5343	Min. :0.0000	Min. :0.0000
1st Qu.:0.6523	1st Qu.:0.4783	1st Qu.:0.2494
Median :0.6666	Median :0.7778	Median :0.4994
Mean :0.6747	Mean :0.6743	Mean :0.4996
3rd Qu.:0.7340	3rd Qu.:0.9200	3rd Qu.:0.7497
Max. :0.7787	Max. :1.0000	Max. :1.0000

The ROC curves for all the years in Python shows a high true positive rate (TPR), and in R all curves (except for Year 2003 that was slightly below the AUC =0.5 line in the beginning). Both graphs appears to have its highest point between 0.7 to 0.8. All that suggests the logistic regressions models are better than chance (AUC=0.5, represented by black line in both graphs) at predicting flight diversions using the selected attributes. From Python, its accuracy of predictions is 0.721 and for R has accuracy of mean = 0.6743 and median = 0.7778.

Moreover, the results of the average AUC scores across the years are all above the 0.5 mark, suggest that the models possesses moderate fit and acts as a good predictive indicator. The slight difference in AUC might be due to implementation details between the logistic regression models in Python and R, and also the slight difference in sampling method and size for Python and R where a bigger sample size is collected using R.

References:

Part 1

Metropolis-Hasting by ritvikmath

https://towardsdatascience.com/bayesian-inference-and-markov-chain-monte-carlo-sampling-in-python-bada1beabca7

https://exowanderer.medium.com/metropolis-hastings-mcmc-from-scratch-in-python-c21e53c485b7

Part 2

https://www.fly.faa.gov/flyfaa/usmap.jsp?legacy=true

https://pypi.org/project/dataframe-image/#:~:text=Pass your normal or styled,save it as an image.&text=You may also export directly,export and export_png methods, respectively.

Code for linear regression: https://python-graph-gallery.com/556-visualize-linear-regression/

https://www.w3schools.com/python/python_ml_logistic_regression.asp

https://sagnik20.medium.com/predicting-flight-delay-using-machine-learning-models-and-azure-notebook-2a8e69b94bd5

Books:

R for Data Science (Second edition) - O'Reilly

Essential Math for Data Science - Thomas Neild