ST2195 Coursework Report

Programming for Data Science

Student Name: Fatema Mansoor Ali Ashoor

SRN: 230713176

Candidate Number: P04139

2024-2025 October Submission

Table of Contents

Introduction

This report presents the steps and results of answering Part 1 and Part 2 of the coursework. First, starting with the application of Metropolis-Hastings algorithm (a popular Markov Chain Monte Carlo method in statistics) to simulate random samples from the Laplace distribution. Second, we utilizes the 2009 ASA Statistical Computing and Graphics Data Expo dataset to analyze flight data, allowing us to reveal delay .

Both Python and R are used for the analysis, with the use of some important libraries such as NumPy, Pandas and Matplotlib in Python and dplyr, ggplot2 and tidyr in R. To ensure clarity for readers who may not be familiar with coding, this report will break down the implementation into simple step-by-step instructions.

Part 1 (a)

In this part, we apply the Random Walk Metropolis-Hastings algorithm to generate samples and plot a histogram and a kernel density plot, using the steps that were provided in the coursework question sheet.

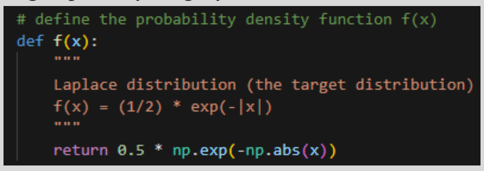
Step 1: Load necessary libraries for data manipulation and creating visualizations, and set a random seed for reproducibility.

$\Rightarrow$ NumPy, Matplotlib Pyplot and SciPy

Step 2: Define the probability density function $f(x) = \frac{1}{2}\exp(-|x|)$.

Step 3: Create the algorithm first by creating an array of zeros to store the generated x values, and then "for" a loop to iteratively generate the x values. Within the loop, we calculate the acceptance ratio for each proposed x value in a "for" loop. A random number from the uniform distribution is drawn and based on the ratio, the value is either accepted or rejected.

A glimpse: defining a function



```
# define the probability density function f(x)
def f(x):
    """
    Laplace distribution (the target distribution)
    f(x) = (1/2) * exp(-|x|)
    """
    return 0.5 * np.exp(-np.abs(x))
```

Step 4: Apply the algorithm to the parameters given in the question N = 10000 (number of samples) and s = 1 (standard deviation).

Step 5: Calculate the sample statistics (mean, standard deviation) and the acceptance ratio.

Step 4: Creating a visualization for the results using Matplotlib by using the generated samples, and overlaying the histogram with the kernel density plot (KDE).

Part 1 (b)

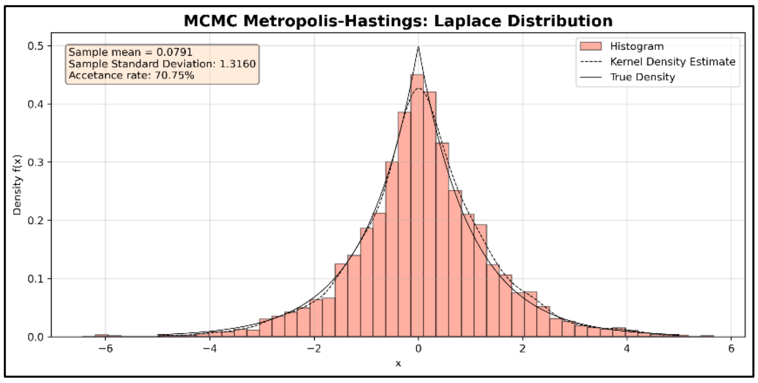
Following the results in part (a), we apply a convergence diagnostics using a the R-hat which is a common diagnostic tool that measures the convergence of MCMC chains. Values close to 1 suggest good convergence, while values greater than 1 indicate that the chains have not converged (meaning the estimates differ significantly).

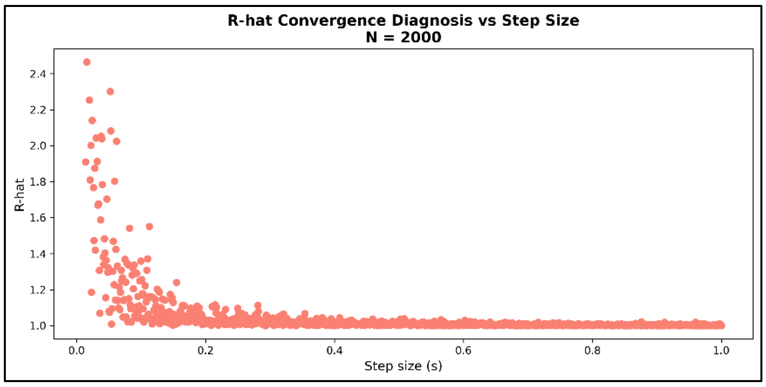
Step 1: Define a function for R-hat using number of samples, step size and number of chains.

Step 2: Calculate R-hat for parameters given in the question.

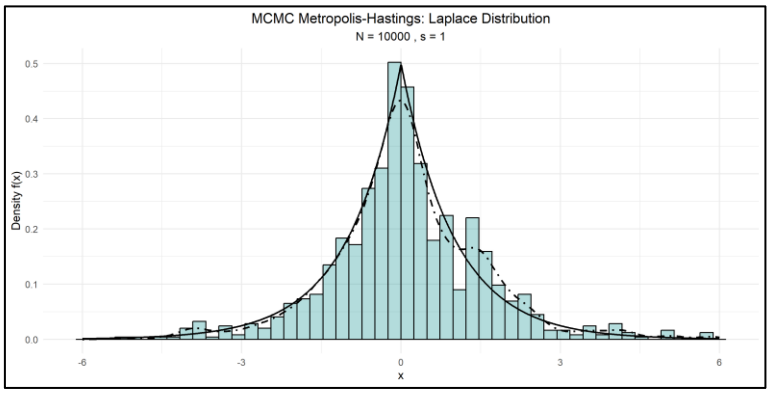
Step 3: Visualize results using a scatter plot.

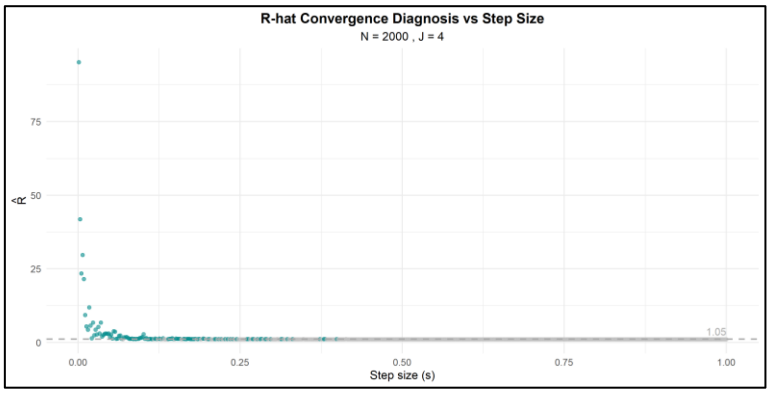
Results in **Python**:





Results in **R**:





Part 2

Question (a)

a.1 Best <u>times</u> of the day to minimize delays.

Step 1: Load flight data for each year (2000-2004), as well as plane data csv files.

Step 2: Check the data before cleaning to get a general idea of the dataset. Combine the data frames for all years and drop the rows with missing values.

Step 3: Combine arrival delay with departure delay to get a single delay variable.

Step 4: Find average delay, sort the values and convert time format to get the best flight time of the day.

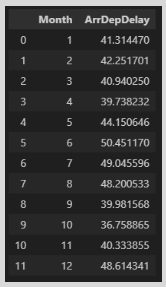
Step 5: Visualize the results.

a.2 Best <u>days</u> of the week to minimize delays.

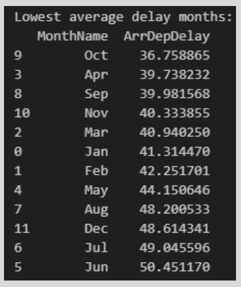
a.3 Best <u>months</u> of the year to minimize delays. (BONUS)

We follow the same procedure here, with some minor adjustments such as assigning names for the days of the week (instead of numbers). Then then we visualize the results using a bar chart since it's most appropriate for a representing categorical data.

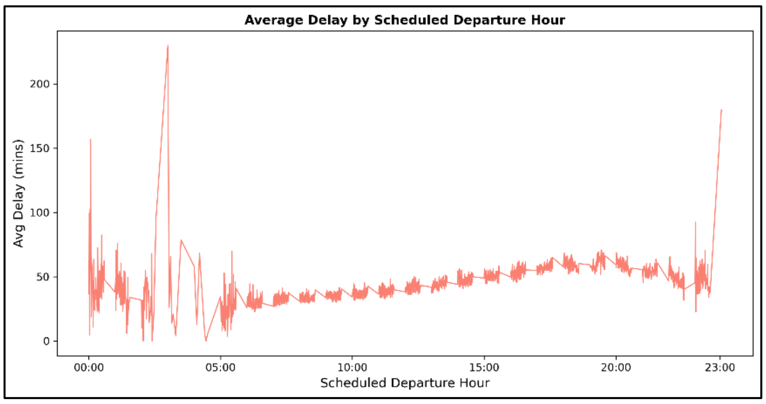
A glimpse: from numbers to month names



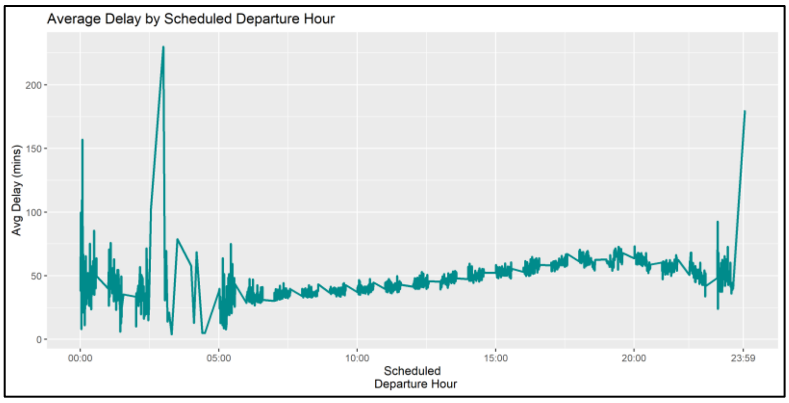
	Month	ArrDepDelay
0	1	41.314470
1	2	42.251701
2	3	40.940250
3	4	39.738232
4	5	44.150646
5	6	50.451170
6	7	49.045596
7	8	48.200533
8	9	39.981568
9	10	36.758865
10	11	40.333855
11	12	48.614341



```
Lowest average delay months:
   MonthName  ArrDepDelay
9        Oct    36.758865
3        Apr    39.738232
8        Sep    39.981568
10       Nov    40.333855
2        Mar    40.940250
0        Jan    41.314470
1        Feb    42.251701
4        May    44.150646
7        Aug    48.200533
11       Dec    48.614341
6        Jul    49.045596
5        Jun    50.451170
```



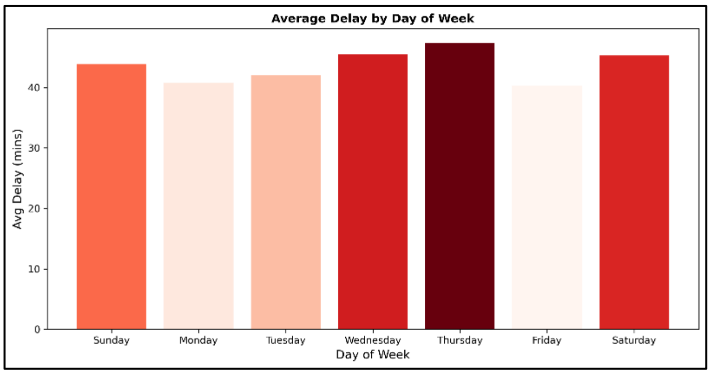
Python vs. R



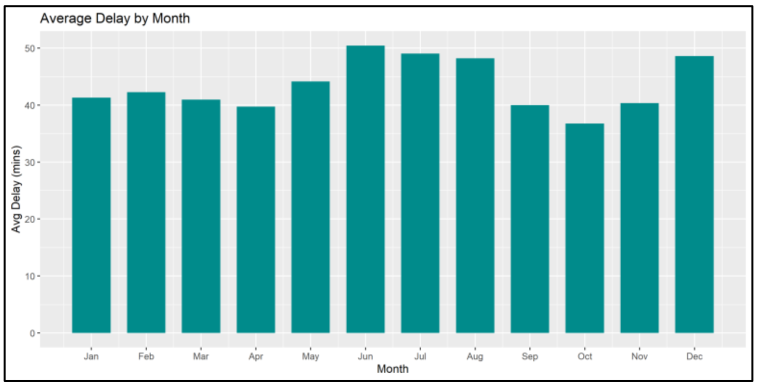
A glimpse: normalizing the delays to apply color gradient (light color = low, dark=high), and using matplotlib to visualize in Python

```
norm_day = plt.Normalize(vmin=day_delay['ArrDepDelay'].min(), vmax=day_delay['ArrDepDelay'].max())
gradient_day = plt.cm.Reds(norm_day(day_delay['ArrDepDelay']))

plt.bar(day_delay['DayName'], day_delay['ArrDepDelay'], color=gradient_day)
plt.xlabel('Day of Week', fontsize=12)
plt.ylabel('Avg Delay (mins)', fontsize= 12)
plt.tight_layout()
plt.title('Average Delay by Day of Week', fontsize=12, fontweight='bold')
```



Python vs. R



Question (b)

Evaluate whether **older planes** suffer more **delays** on a year-to-year basis.

In this part, we explore whether there is a correlation between plane age and delays, more specifically, whether older planes tend to experience more delays.

Null hypothesis *H0:* There is no significant difference in the delay rates of older planes compared to newer ones on a year-to-year basis.

Step 1: Merge the csv file *plane-data.csv* with 2000-2004 data frame

A glimpse: merging plane data with yearly flight data in Python

```
merged = df.merge(airplanes.rename(columns={'tailnum': 'TailNum'}), how='left', on='TailNum', copy=False)
```

Step 2: Calculate plane age by subtracting manufacturing year from scheduled flight year.

Step 3: Perform some data cleaning steps such as removing invalid manufacturing years.

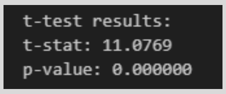
Step 4: Find average delay, and sort by manufacturing years with the lowest delay.

Step 5: Visualize results, scale the size of bubbles by number of flights.

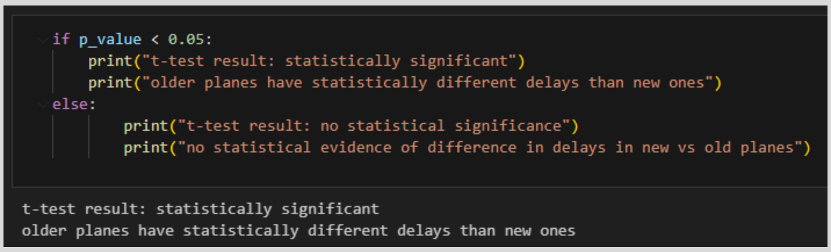
Step 6: Perform a statistical analysis by first splitting planes into two categories: "Old" and "New" using the threshold 20 years, and calculate the proportion of delays in each.

Step 7: Perform a t-test and find the t statistic and p value.

A glimpse: t-test results



```
t-test results:
t-stat: 11.0769
p-value: 0.000000
```

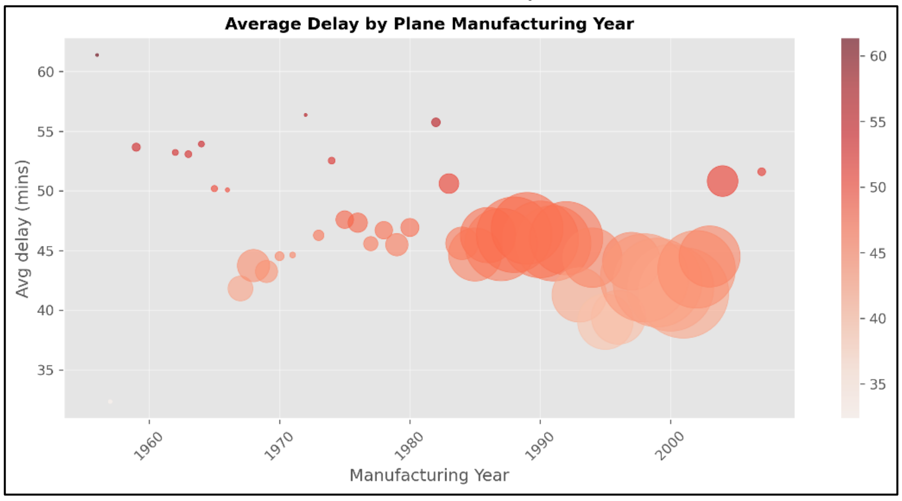


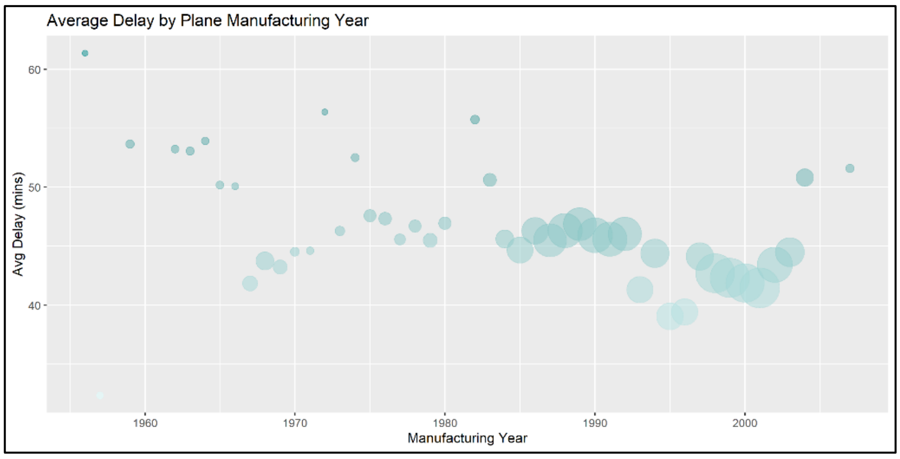
```
if p_value < 0.05:
    print("t-test result: statistically significant")
    print("older planes have statistically different delays than new ones")
else:
        print("t-test result: no statistical significance")
        print("no statistical evidence of difference in delays in new vs old planes")
```

```
t-test result: statistically significant
older planes have statistically different delays than new ones
```

Step 8: Conduct a year-by-year analysis for extra details and a more thorough analysis, as well as correlation classification.

Visualization results in **Python** and **R**:





Question (c)

For each year, fit a logistic regression model for the probability of diverted US flights using as many features as possible from attributes of the departure date, the scheduled departure and arrival times, the coordinates and distance between departure.

Finally In this part, we aim to develop a logistic regression model to predict flight diversions, using features from the dataset such as Month, DayOfWeek, Distance and Origin.

Step 1: Filter for US flights.

Step 2: Merge airport locations with flight records.

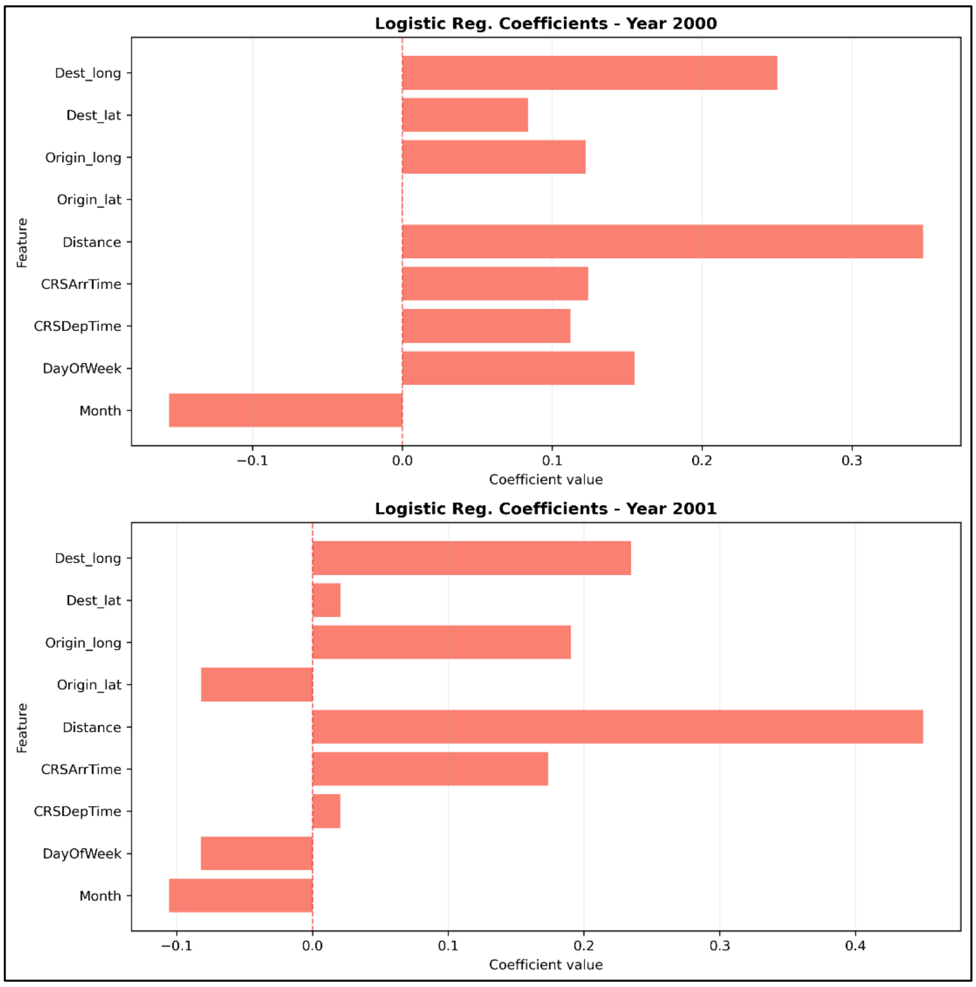
Step 3: Start training the model by assigning features (x) and the target (y) which is the Diverted flights.

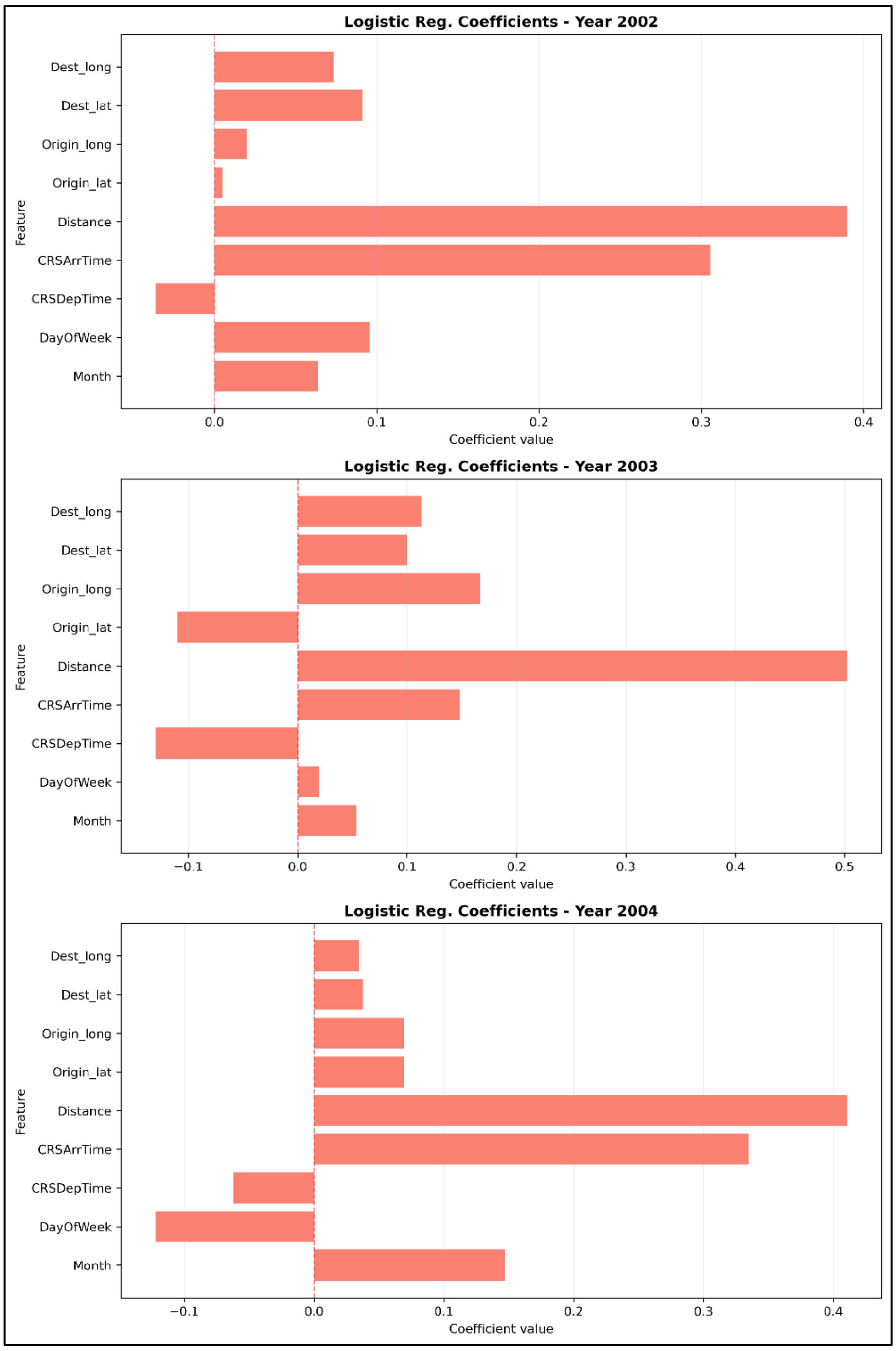
Step 4: Split data, 20% for testing and 80% for training. Make sure to track model performance and test accuracy.

Step 5: Visualize the results for all years by creating subplots plotting features against coefficient value.

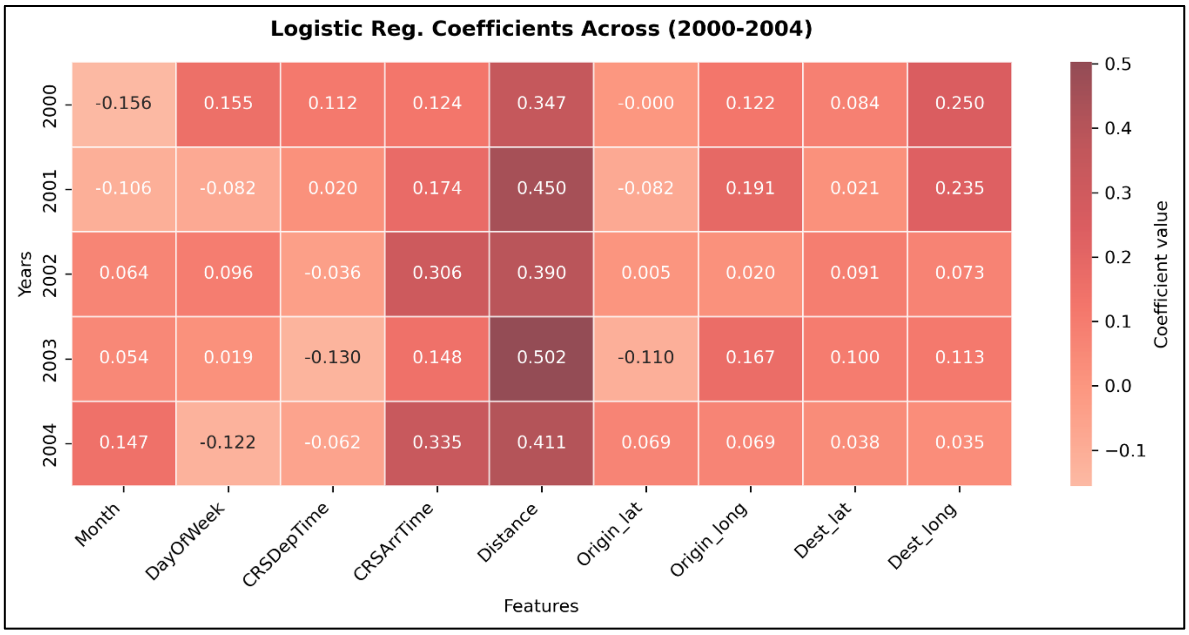
Step 6: Create a heatmap to visualize coefficient comparison across years.

Python output:





From the figures above, we discover the most significantly influential features across years, i.e. Distance and CRSArrTime in 2004.



Years	Month	DayOfWeek	CRSDepTime	CRSArrTime	Distance	Origin_lat	Origin_long	Dest_lat	Dest_long
2000	-0.156	0.155	0.112	0.124	0.347	-0.000	0.122	0.084	0.250
2001	-0.106	-0.082	0.020	0.174	0.450	-0.082	0.191	0.021	0.235
2002	0.064	0.096	-0.036	0.306	0.390	0.005	0.020	0.091	0.073
2003	0.054	0.019	-0.130	0.148	0.502	-0.110	0.167	0.100	0.113
2004	0.147	-0.122	-0.062	0.335	0.411	0.069	0.069	0.038	0.035

Key features from the figure above: the heatmap shows that distance is a consistently strong positive predictor of flight diversion probability across all years (high coefficient values, darker color), indicating that longer flight distances are strongly associated with a higher probability of diversion. While most features maintain stable coefficient signs, their magnitudes vary over time. Other variables, such as scheduled departure and arrival times and geographic coordinates, show smaller and more variable effects across years.

From this year-to year analysis, we discover that the coefficients change yearly, reflecting that factors influencing flight diversion are not static and are affected by other external factors such as operational conditions and scheduling practices.

Conclusion

This report analyzed the steps done in Python and R and presented the analysis' findings in a simple manner for non-programmers. First we implemented the Metropolis-Hastings algorithm to sample from the Laplace probability density function, and assessing convergence using R-hat that lead to finding values close to 1, indicating successful convergence. This shows how the algorithm's can be used for sampling from complex distributions. In the second part, we explored flight data to uncover patterns behind delays to find best flight times. Then we tested impact of aircraft age on performance. Lastly, we applied logistic regression to assess the relationships between factors influencing flight delays and US flight diversions.

*only python code examples were used in the report for simplicity.