FINAL YEAR REPORT

Machine Learning-Based Flight Delay Prediction

By

SOE CHOU KIT

TP058323

APD3F2305CS(DA)

A project submitted in partial fulfilment of the requirements of Asia Pacific University of Technology and Innovation for the degree of

B.Sc. (Hons) Computer Science with a specialism in Data Analytics

Supervised by Dr. Vazeerudeen Hameed

2nd Marker: Ms. Lai Chew Ping

10-January-2024

DECLARATION OF THESIS CONFIDENTIALITY

Author's full name: **SOE CHOU KIT**

IC No./Passport No.: **001009010825**

Thesis/Project title: **Machine Learning-Based Flight Delay Prediction**

I declare that this thesis is classified as:

☐ CONFIDENTIAL

☐ RESTRICTED

☑ OPEN ACCESS

I acknowledged that Asia Pacific University of Technology & Innovation (APU) reserves the right as follows:

1. The thesis is the property of Asia Pacific University of Technology & Innovation (APU).
2. The Library of Asia Pacific University of Technology & Innovation (APU) has the right to make copies for the purpose of research only.
3. The Library has the right to make copies of the thesis for academic exchange.

Author's Signature: 

Date: 30 September 2023

Supervisor's Name: **DR. VAZEERUDEEN ABDUL HAMEED**

Date: 20 December 2023

Signature: .....Vazeer.....

First Name: Soe
Middle Name (only if applicable) :
Last Name: Chou Kit
Title of the Final Year Project / Dissertation / Thesis: Machine Learning-Based Flight Delay Prediction
Abstract: This study addresses the pervasive issue of flight delays in the rapidly evolving landscape of civil aviation, acknowledging their significant impact on both passengers and the aviation industry. Flight delays disrupt travel plans, causing inconvenience, financial setbacks, and heightened stress levels for passengers. Moreover, these delays impose substantial financial burdens on airlines, affecting operational expenses, customer satisfaction, and overall profitability. In response to this challenge, the study employs machine learning techniques, specifically Random Forest, Decision Tree, XGBoost and LightGBM, to predict flight delays accurately. The source of data is from Kaggle.com which named Combined_Flights_2021 dataset. The research emphasizes data pre-processing techniques such as cleaning, transformation, under sampling, and feature selection to enhance model robustness. The findings reveal XGBoost as the optimal algorithm for flight delay prediction. Feature importance analysis highlights the crucial role of time-related variables, airline choice, and monthly patterns in predicting delays. This knowledge empowers airlines and airports to proactively manage and optimize resources, ultimately enhancing operational efficiency and customer satisfaction. The study's significance lies in its potential to positively impact the aviation industry, providing valuable insights for stakeholders to make informed decisions, reduce operational costs, and improve overall performance.
A few keywords associated with the work: Flight, Delay, Machine Learning, Aviation Industry, Customer Satisfaction, Travels
General Subject: Computer Science in Data Analytics
Date of Submission: 10th January 2024

Acknowledgement

First, I would like to grateful to Asia Pacific University of Technology & Innovation (APU) for providing me a chance to complete my Final Year Report. I grateful to my supervisor, Dr. Vazeerudeen Hameed, for freely imparting his vast knowledge and experience. His direction and support were essential in helping me to improve the quality of my work and get a better grasp of my topic. The direction of my study has been greatly influenced by Dr. Vazeerudeen Hameed's vast knowledge, skill, and passion. His guidance went beyond the expected bounds, and he continually went above and above to make sure I understood my study field completely.

Secondly, I would also like to thank Ms. Lai Chew Ping as my second marker, for her valuable insights and constructive feedback. Her thorough evaluation significantly enriched the overall assessment of my project. I am honoured to have been a part of the APU community and to have had the privilege of working with such dedicated and supportive individuals. His insightful discussions, thought-provoking questions, and constructive feedback have been instrumental in refining my project's direction and enhancing its overall quality.

Lastly, I am thankful to Dr. Dewi Octaviani, my FYP manager, for her invaluable guidance and mentorship throughout this journey. Dr. Dewi Octaviani's expertise and dedication were instrumental in teaching us how to produce a comprehensive and high-quality Investigation Report. Her insightful feedback and continuous support greatly contributed to shaping the success of my project. I'm very thankful for her efforts in arranging a suitable supervisor and second marker for my project.

Table of Contents

List of Figures

List of Tables

CHAPTER 1: INTRODUCTION

1.1 Background to the project

Due to the swift progress of civil aviation, flight delays have emerged as a significant concern and challenge for air transportation systems worldwide. Continuously, the aviation industry faces financial losses due to these delays. Air travel now plays a crucial role in our contemporary society, linking individuals and businesses globally. Nonetheless, flight delays can be extremely inconvenient for passengers, resulting in missed connections, complicated rescheduling, and time waste (Yu et al., 2019). Airlines and airports are constantly looking for new ways to improve their operations and reduce delays, and one promising approach is to use machine learning methods.

1.2 Problem Statement

1 Travel plans disrupted and passengers inconvenienced

Passengers who miss their connecting flight, important appointments, momentous occasions, or must spend a significant amount of time waiting at the airport may experience significant disruptions and inconveniences. These delays may result in detrimental consequences like financial setbacks, missed opportunities, and elevated stress levels. Due to scheduling conflicts and missed professional and social obligations, passengers may incur financial losses (CAA, 2014). Additionally, the overall travel experience as well as people's well-being may be impacted by the increased stress and annoyance brought on by extended wait times and derailed travel plans.

2 Financial impact on airlines and the aviation industry

Airlines as well as the larger aviation industry are significantly impacted financially by flight delays (Anupkumar, n.d.). They result in increased operational expenses, including additional crew hours, higher fuel consumption, and elevated aircraft maintenance costs, thereby placing strain on airline profitability. Furthermore, delays have the potential to negatively affect customer satisfaction, damage brand reputation, and reduce customer loyalty. These consequences can lead to a decrease in passenger demand and a subsequent decline in revenue. Hence, the financial impact of flight delays extends beyond immediate operational costs, encompassing long-term effects on

customer perception and the sustainable operation of businesses within the aviation sector.

3 Decreased operational efficiency for airlines and airports.

Flight delays have the potential to set off a chain reaction, impacting the smooth operation of both airlines and airports. This results in congestion at terminals, longer turnaround times for aircraft, and disrupted schedules for the crew, leading to inefficiencies and underutilization of resources. Consequently, this incurs higher expenses for the airline, diminishes customer satisfaction, and presents overall operational challenges. Increased congestion at hub airports affects on-time airline performance on-time to the detriment of customer satisfaction and may have substantially negative repercussions for airlines in a hypercompetitive environment. These operational shortcomings highlight the significance of timely and effective operations, as delays can have far-reaching effects on the entire airline and airport ecosystem.

1.3 Aim and Objectives

1.3.1 Aim

The aim of this study is to determine machine learning techniques can be used to increase the accuracy of flight delay prediction. The purpose of this study is to significantly contribute to the aviation sector.

1.3.2 Objectives

1. To compare different machine learning algorithms and methodologies for effectively predicting flight delays.
2. To construct prediction models using historical flight data, weather data, and airport infrastructure data.
3. To establish techniques and suggestions to improve the reliability and efficacy of machine learning-based flight delay prediction models.
4. To provide valuable information to the current knowledge base and allow airlines, airports, and other key stakeholders beneficial information so they may make intelligent choices.

1.4 Scope

The scope of this study includes a thorough investigation and prediction of departure delays for several airlines operating in the United States. The major purpose is to use a dataset that captures the relevant information of aircraft departures and arrivals, exploring into factors like historical flight data. The difficulties in this study occur from the many and dispersed sources of data, possibly limited access, and worries about data quality. The availability, consistency, and correctness of the acquired data will be ensured by attempting to address these problems.

The study entails thorough pre-processing of the data to improve its quality and relevance, such as feature selection, data transformation, and missing value removals. Training effective prediction models that can identify between delayed and on-time flights is the goal, with a focus on classification machine learning methods including XGBoost, LightGBM, Random Forest, and Decision Trees. It is noteworthy that the predictive model just relies on the binary classification of delay or not delay, without providing an accurate delay time prediction.

1.5 Rationale

Predicting flight delays is critical to enhancing client loyalty and happiness. By precisely projecting flight delays, airlines can efficiently manage customers' expectations, actively communicate with them, and take appropriate action to reduce annoyance. Airlines can also reduce the incidence of incidents in their operations by utilizing resources and making preventive adjustments. This will increase productivity, save costs, and improve overall performance. Airlines can minimize disruptions to flight schedules and the need for last-minute maintenance interventions by planning ahead and scheduling maintenance within predicted downtime. In addition, detailed monitoring of aircraft delays helps airlines to utilize resources, including fuel, gates and land handling equipment, more effectively and efficiently.

1.6 Potential benefits

1.6.1 Tangible benefits

There are many benefits of predicting flight delay like reduced costs, better crew management and better control of air traffic. The most important benefit is cost savings. When airlines enable predict flight delays, they may decide to take preventative steps like rescheduling or rerouting flights to cut down on the costs. Furthermore, smart distribution of resources can cut down on inefficiencies and boost cost-effectiveness by implement predictions for flight delays. It helps controllers fine-tune flight plans, change routes, and handle traffic more efficiently. This improves management of airspace, lowers air traffic, and makes flights safer. Airlines may also benefit from this foresight by setting up employee's schedules well, following the rules, and putting employee welfare first.

1.6.2 Intangible benefits

Airlines may be able to improve their brand image and reputation by using systems that monitor flights all the time and make accurate predictions about when flights will be delay. This shows that an airline is dedicated to operating at a high level and making sure passengers are satisfied. These ensuing implications can make customers more loyal and give the company an edge in the market. Airlines maintain passenger trust and confidence by the regular provision of precise predictions and effective management. The technique that can predict flight delays help airlines make better decisions in many areas of their business and give them useful information. Airlines can more efficiently plan routes, manage capacity, make well-informed infrastructure expenditures, and distribute resources by analysing past delay trends and determining the underlying reasons.

1.7 Target users

Predictions of flight delays are useful for many users in the aviation industry, such as airlines, airports, travellers, and travel agencies. This prediction can help airlines be more efficient, make smart decisions, and lessen the damage that delays due to their schedules, resources, and overall efficiency. Hence, airports can manage facilities and resources effectively. For example, airports can adjust staffing at security checkpoints to prevent long queues and maintain a smooth flow of passengers. People are informed in a timely manner

about possible flight delays, which gives them the chance to plan, make other arrangements, or change their schedules as needed. Travel agents get accurate information about delays, which helps them provide better customer service. This lets them offer affected passengers a range of travel options and quickly change people's plans as needed.

1.8 Overview of FYP

Chapter 1: Introduction to the study

Chapter 1 included introduction of this project's background. The problem statements are related to the domain. The aim and objective of this project also included in this section. The target users can be benefit in this project which can divide to tangible and intangible. The scope and rationale have mentioned the main task of this project.

Chapter 2: Literature review

Literature review plays as an important role for generate a good and successful report. This chapter offers a comprehensive examination of existing research and scholarly literature pertaining to a specific subject. It entails a methodical evaluation and analysis of published studies, books, articles, and other sources to identify essential discoveries, concepts, theories, methodologies, and areas where knowledge gaps exist. The domain research has explained and investigation the domain that related to this project. The programming language which chosen for this project has been mentioned in this section. The library and operating system also included in this section.

Chapter 3: Methodology

Chapter 3 provides introduction of the methodologies. This chapter compare two methodologies and select the best methodology. The selected methodology has been utilized in this project as a guide to assist in carrying out the project. The explanation of the step selected methodology described and how this project implemented.

Chapter 4: Design and Implementation

Chapter 4 is the implementation of the project. The source of data mentioned in this section. Data understanding has been applied in this section by using EDA. Data pre-processing also applied to make sure the data is clean and accept by machine learning algorithms. Four types of the machine learning built in this section.

Chapter 5: Result and Discussion

Chapter 5 discuss about the evaluation of machine learning models. After comparison, the best machine learning algorithm has been selected good for flight delay prediction. The feature important also discovered in this section. System deployment also executed based on the best machine learning algorithm.

Chapter 6: Conclusion

Chapter 6 focus on the conclusion of the whole project. The achievements, contribution of the system to the public and strengths had shown in this chapter. Lastly, the limitations have been faced in this project are also addressed. The related recommendations have been given for future works.

1.9 Project Plan

Task Name	Duration	Start Date	End Date	Status
INTRODUCTION TO THE STUDY				
Background To the Project	3 hours	15-July-2023	15-July-2023	Completed
Problem Statement	3 hours	15-July-2023	15-July-2023	Completed
Rationale	3 hours	16-July-2023	16-July-2023	Completed
Potential Benefits	2 hours	16-July-2023	16-July-2023	Completed
Target Users	2 hours	16-July-2023	16-July-2023	Completed
Aim and Objectives	2 hours	17-July-2023	17-July-2023	Completed
Overview of the Report	2 hours	17-July-2023	18-July-2023	Completed
LITERATURE REVIEW				
Introduction	1 hour	17-July-2023	18-July-2023	Completed
Domain Research				
Aviation Industry	5 hours	18-July-2023	18-July-2023	Completed
Application of Machine Learning in Aviation Industry	10 hours	19-July-2023	19-July-2023	Completed
Machine Learning Techniques	10 hours	20-July-2023	20-July-2023	Completed
Similar Work	30 hours	21-July-2023	23-July-2023	Completed
Summary	5 hours	24-July-2023	24-July-2023	Completed
TECHNICAL RESEARCH				
Programming Language	2 hours	24-July-2023	24-July-2023	Completed
IDE	1 hours	24-July-2023	24-July-2023	Completed
Libraries/Tools	1 hours	24-July-2023	24-July-2023	Completed
Operating System	1 hours	24-July-2023	24-July-2023	Completed
Summary	1 hours	24-July-2023	24-July-2023	Completed
METHODOLOGY				
Introduction	2 hours	25-July-2023	25-July-2023	Completed
Methods	6 hours	26-July-2023	27-July-2023	Completed
Preliminary Data Analysis				
Data Collection	2 hours	30-July-2023	30-July-2023	Completed
Data Understanding	46 hours	1-August-2023	3-August-2023	Completed
Data Pre-Processing	24 hours	4-August-2023	6-August-2023	Completed
Summary	2 hours	7-August-2023	7-August-2023	Completed
CONCLUSION				
Conclusion	2 hours	7-August-2023	7-August-2023	Completed

Figure 1:Project Plan Sem 1

Task Name	Duration	Start Date	End Date	Status
INTRODUCTION TO THE STUDY				
Background To the Project	3 hours	15-July-2023	15-July-2023	Completed
Problem Statement	3 hours	15-July-2023	15-July-2023	Completed
Aim and Objectives	3 hours	16-July-2023	16-July-2023	Completed
Scope	2 hours	16-July-2023	16-July-2023	Completed
Rationale	2 hours	16-July-2023	16-July-2023	Completed
Potential Benefits	2 hours	17-July-2023	17-July-2023	Completed
Target Users	2 hours	17-July-2023	18-July-2023	Completed
Overview	1 hour	31-December-2023	31-Decemebr-2023	Completed
Project Plan	1 hour	31-December-2023	31-December-2023	Completed
LITERATURE REVIEW				
Domain Research	25 hours	18-July-2023	20-July-2023	Completed
Similar Work	30 hours	21-July-2023	23-July-2023	Completed
Technical Research	5 hours	24-July-2023	24-July-2023	Completed
METHODOLOGY				
Introduction	2 hours	25-July-2023	25-July-2023	Completed
Methods	6 hours	26-July-2023	27-July-2023	Completed
Summary	2 hours	28-July-2023	28-July-2023	Completed
DESIGN AND IMPLEMENTATION				
Introduction	1 hour	29-July-2023	29-July-2023	Completed
Data Collection and Data Understanding	48 hours	30-July-2023	3-August-2023	Completed
Data Pre-processing	24 hours	4-August-2023	6-August-2023	Completed
Model Building	180 hours	20-Novemeber-2023	10-December-2023	Completed
Summary	1 hour	11-December-2023	11-December-2023	Completed
RESULT AND DISCUSSION				
Introduction	2 hours	13-December-2023	13-December-2023	Completed
Model Evaluations and Discussions	48 hours	14-December-2023	17-December-2023	Completed
Feature Importance	5 hours	18-December-2023	19-December-2023	Completed
Model Deployment	24 hours	19-December-2023	21-December-2023	Completed
Summary	1 hour	22-December-2023	22-December-2023	Completed
CONCLUSION				
Critical Evaluation	24 hours	24-December-2023	26-December-2023	Completed
Limitations	10 hours	27-December-2023	28-December-2023	Completed
Recommendations	10 hours	29-December-2023	30-December-2023	Completed

Figure 2:Project Plan Sem 2

CHAPTER 2: LITERATURE REVIEW

2.1 Domain Research

2.1.1 Problem faced by Aviation Industry

The aviation industry is very globalized and benefits people, businesses, and cultures from all over the world to connect with each other. Professionals in the industry who are devoted to enhancing aviation technology and the convenience of the flying experience. This industry not only fosters economic growth and job prospects, but it also plays an important role in enabling worldwide commerce and tourism. The aviation sector has grown significantly throughout time as air travel has become more accessible and inexpensive, resulting in a rise in the number of passengers and flights on a worldwide scale. However, the industry has faced a proportional rise in challenges and problems. This growth has given rise to various challenges, including issues like flight delays and cancellations, air traffic congestion, security considerations, and numerous others.

The aviation sector must make complicated decisions on service quality, safety, and financial performance. When assessing service quality, decision-makers must examine tangibility, dependability, safety, security, speed, reservation and ticketing service, check-in and boarding operations, and financial performance. Furthermore, the sector must resolve problems such as aircraft procurement, route planning, and maintaining safety and security while considering a broad variety of quantitative and qualitative aspects (Dožić, 2019).

Besides that, experience for passengers is also very important. However, passengers in the airline sector confront a variety of challenges, including legacy systems that result in delays, such as the necessity for human validation at check-in and constraints on ticket pricing depending on the number of English alphabet rather than specific needs. Other issues include the failure to dynamically modify change costs, a lack of customized goods and services, fragmented procedures because of unavailable and non-comparable data, and the lack of a single view of the customer. Passengers are often inconvenienced by computer problems and outages, which cause severe travel delays (Silling, 2019).

2.1.1.1 Flight Delay

Flight disruption stands as a crucial concern within the contemporary aviation sector. Flight delays occur when flights depart or arrive at the airport later than 15 minutes of their scheduled time. The smooth operation of flights faces numerous challenges due to the

involvement of various stakeholders. Some factors that directly affect the airline's operations include managing aircraft turnover, ensuring passenger punctuality, utilizing technology effectively, and maintaining crew performance. However, there are numerous additional factors beyond the control of the airline, including weather conditions, air traffic control, security measures, and airport conditions (Busson, 2020).

According to Zámková et al. (2022), airport restrictions are mostly to responsible for the longer delays that happen in the afternoon and at night. These prolonged delays are often attributed to operational issues with the airline, air traffic control delays, supplier delays (related to handling, fuelling, and catering), delays linked to handling passengers and luggage, operational requirements, and crew duty rules.

2.1.1.2 Weather

The aviation sector relies heavily on weather conditions since directly impacts the capacity of airports and airspace resources, thereby affecting the operations of airlines, airports, and air traffic. Unfavourable weather conditions frequently lead to a mismatch between demand and capacity, resulting in flight delays and cancellations that decrease the overall on-time performance (OTP).

De Oliveira et al. (2021) have noted that numerous studies have investigated the impact of weather conditions on flight operations, which are well-documented in the literature. Within Europe, nearly 50% of regulated airport traffic delays can be attributed to adverse weather conditions. Additionally, capacity loss brought on by outside causes like weather has a more significant effect when airports are operating at or close to their full capacity. Multiple weather-related variables may considerably slow down arrival times and have an impact on the whole Air Traffic Management (ATM) system during airport operations (Rodríguez-Sanz et al. 2021).

The influence of typical weather events such as thunderstorms, snow, or fog on flight delays is extensively established. Meanwhile, space weather can also cause delays in flights because it can affect communication and navigation systems. These disturbances could cause planes to change routes and make communication less clear, both of which could cause delays. Weather conditions, particularly space weather, may thus significantly affect aircraft operations departing from the origin airport (Kauristie, et al., 2021).

2.1.1.3 Passenger Satisfaction

Passenger satisfaction is very important for airlines because it affects customers plans to behave, their loyalty, and the overall performance of the business. Passengers who are happy with an airline are more likely to do good things, like using the airline again and telling others about it. Several elements of airline service quality have an impact on passenger satisfaction. Shah et al. (2020) indicates that passenger happiness and behavioural intentions are impacted by airline service quality factors such tangibles, certainty, responsiveness, and empathy.

Flight delays can significantly impact passenger dissatisfaction. Research indicates that passengers often experience negative emotions such as displeasure, uncertainty, and disappointment when facing delays, and these emotions tend to intensify as the duration of the delay increases. The impact of delays on passenger dissatisfaction is also influenced by the severity of the situation, as passengers may be unaware of the reasons for delays or may not even realize they are delayed until shortly before the boarding process. Furthermore, the length of the delay is positively correlated with the cost of the delay, which can contribute to increased dissatisfaction among passengers (Eftymiou et al., 2018). Therefore, flight delays can lead to heightened dissatisfaction due to the emotional impact, lack of information, and the costs associated with the delay.

Song et al. (2020) found that flight delays are the main reason cause most passengers are unhappy. Passenger experience is very important to airlines since airline services include a wide range of product features. It is obvious that flight delays do not remain true to consumer expectations. People who experience flight delays are more likely to feel negatively, and the proportion of people who feel negatively much outweighs the proportion of people who feel positively. Furthermore, following a flight delay, passengers pay closer attention to service aspects, and their satisfaction with the airport significantly declines.

2.2.2 Machine Learning Application

Machine learning techniques is very important for build prediction model. Based on previous occurrences, predictive analytics focuses on identifying and leveraging correlations between explanatory and predictor factors to estimate unknown outcomes (Sarker, 2021). In essence, machine learning involves assigning a computer program to perform specific tasks (Ray, 2019). As the machine gains experience and demonstrates improved measurable

performance in these tasks, it can be deemed to have learned from the experience. There are several methods that can be utilized, including reinforcement learning, unsupervised learning, semi-supervised learning, and supervised learning. The choice of technique is made based on the specific types and categories of training data.

Unsupervised learning falls under the realm of machine learning, aiming to uncover patterns, structures, and connections within unlabelled data. This type of learning is particularly beneficial for tasks like grouping similar data points (clustering), reducing data complexity (dimensionality reduction), detecting anomalies, and learning association rules. It works without clear target labels or directions on how to interpret data. They examine the facts on their own and arrive to insightful conclusions without having any prior understanding of the anticipated results. One drawback of unsupervised learning is that it does not offer clear instruction, which may result in outcomes that are less accurate or unexpected when compared to supervised learning. Additionally, unsupervised learning may require more complex algorithms to identify patterns in unstructured data, and the interpretation of the results can be more challenging due to the absence of labelled data (Naeem et al., 2023). K-means clustering, hierarchical clustering, and Gaussian Mixture Models (GMM) are the few popular unsupervised learning techniques.

Supervised learning is a machine learning technique that involves training an algorithm using a collection of labelled instances. Input features and their target class labels are shown in the observations. The primary objective is to create a connection or pattern between the input attributes and the target labels so that the method may be used to previously unknown data and produce predictions or classifications. Regression and classification are the main types of supervised learning. Regression is the analysis of making predictions about numbers that change over time and lead to specific numbers. On the other hand, classification involves putting examples into separate groups, with the target variable representing specific classes or labels (Shetty et al., 2022). One of the benefits of supervised learning is that it allows user make predictions and learn from past data, which makes behaviour smarter. Supervised learning also helps user train models on labelled data and test them on unlabelled data, which is important for tasks like fraud detection, getting information, and diagnosing medical conditions. (Dridi, 2021). Support vector machines, decision trees, and logistic regression are a few of the popular algorithms that are implemented in supervised learning (Shetty et al., 2022).

2.2.2.1 Type of Classification Task

Machine learning can be identified into four basic classification problems, including unbalanced, multi-class, multi-label, and binary classifications. The objective of the multi-class classification is to predict which class a given input sample belongs to, using at least two class labels that are mutually incompatible. We attempt to anticipate zero or more classes for each input case in multi-label classification challenges. Mutual exclusion is not applicable in this scenario since the input example is permitted to possess multiple labels. An imbalanced distribution of examples occurs in each class for the imbalanced classification, which means that the training data may have a greater number of instances from one class than the others. Identifying the input data into two distinct groups that are not mutually exclusive is the objective of a binary classification problem. The training data is designated in a binary format, which varies depending on the task at hand: true and false; positive and negative; O and 1; delay and no delay (Keita, 2022).

The utilisation of binary classification in flight delay prediction has notable benefits owing to its straightforwardness and unambiguous delineation of results. The objective of this assignment is to classify input data into two distinct and non-delayed categories. In contrast to multi-class or multi-label classifications, binary classification emphasises a binary format to simplify the prediction process. This direct methodology optimises the process of training the model, resulting in a more comprehensible and effective outcome. Furthermore, binary classification is very suitable for rectifying the issue of unbalanced datasets by guaranteeing an equitable distribution of occurrences classified as delay and non-delay. In general, the classification task's binary structure enables accurate and succinct predictions, making it a desirable option for situations involving flight delay prediction.

2.2.2.2 Application in Aviation

The aviation sector has experienced a significant shift due to technological advancements and the emergence of new digital capabilities. Intelligent solutions can make the industrial sector more efficient, reduce expenses, and boost production. Modern systems use cutting-edge technology such as automation, robots, machine learning, Internet of Things (IoT) and artificial intelligence (AI). Machine learning is essential to the digitization process because it identifies and extracts traits, patterns, and trends to facilitate smart decision-making (Jiang et al., 2023).

In the fields of aeronautical architecture and transportation systems, several machine learning models have been implemented (Guo et al., 2021). Recently, airline professional and many researchers have explored in this area such as using natural language processing (NLP) in analyzing passenger feedback and customer reviews to do passenger sentiment analysis to improve their customer experiences. Kumar & Zymbler (2019) has proposed machine learning algorithms to using tweets to discover out how customers feel for the airline service.

Machine learning techniques are also employed to build a sentiment classification model for Twitter data, focusing on several popular airlines worldwide. Furthermore, Yılmaz et al. (2021) has applied machine learning approach to forecast fuel tankering by implementing various algorithms. Fuel consumption has become increasingly significant and has a direct impact on companies actively involved in air transportation. To capitalize on the differences in fuel prices between departure and arrival airports, cost-saving techniques known as fuel tankering have been employed. Other more applications are weather forecasting for flight planning, predicting aircraft potential faults and air traffic management.

According to Gui et al. (2020), air traffic congestion in constrained air space brought on by increasing demand for air travel and the corresponding rapid development of flights will impact air traffic management (ATM). Machine learning algorithms have been implemented in time series analysis due to their significant power in identifying complex structures and matching inherent links between data. Support vector machines (SVR) can predict the flow with high accuracy because of their better performance when handling data with observable trends. Long Short-Term Memory performed better than the SVR-based predictor in the circumstance under study, showing that the LSTM-based predictor can handle aberrant points of flow fluctuation better. Those can prove that machine learning has been widely utilize in aviation industry.

2.2.2.3 Flight Delay Prediction

In recent times, flight delay prediction has gained significance since it may decrease passenger dissatisfaction, prevent economic extra expenses for airlines and relieve aircraft delays. Monitoring real-time flight delays requires prediction models with a high degree of accuracy, since those with less precision may spread false information airport operators and result in ineffective efforts to mitigate aircraft delays. Aviation experts may minimise the adverse effects of undesired traffic, enhance planning, and monitor flight delays with more

precision by developing prediction models that recognize the likely processes of such delays in depth (Li & Jing, 2021). Therefore, the importance of flight delay prediction lies in its ability to improve daily operations for airports, support decision-making processes for arranging business operations, avoid potential service costs, and generate optimal flight scheduling for airlines.

As a result, data-driven analytics has been a prominent way for attempting to anticipate flight delays directly by utilising machine learning techniques, data mining, and statistical analysis. Many well-known data-driven strategies have been employed to forecast aircraft delays, including log probability, deep learning, random forest algorithms and artificial neural networks (Bojia et al. 2020). It's possible to find patterns and factors that cause flight delays by using machine learning algorithms. This valuable information can then be utilized for predicting potential delays for specific flights. The predictive capability empowers airlines to proactively respond by adjusting schedules or redirecting passengers, thereby mitigating the potential negative effects of delays (Thakur, 2023).Recently, there are rising research on this similar topic. Hatıpoğlu et al. (2022) and Anusha et al. (2022) has employed machine learning to predict flight delays.

2.2.3 Flight Delay Prediction Algorithms

The selection of an effective machine learning approach for flight delay prediction is critical due to the capacity to provide predictions that yield insights into the level of uncertainty in the estimated delays. By considering the unpredictability of the projected delays for specific flights, probabilistic predictions may assist planners in effectively organising flight operations, including gate and runway assignments, as well as providing passengers with appropriate backup plans (Zoutendijk & Mitici, 2021). An appropriate algorithm should possess the capability to efficiently classify flights as delayed or not delayed in the setting of flight delays (Tang, 2021). The system furnishes airport planners with a quantification of the confidence associated with the estimated flight delays of all arriving planes. As a result, airports and airlines will be able to provide superior service by means of an enhanced capacity for predicting flight delays; this may be accomplished with the assistance of the appropriate machine learning algorithm.

2.2.3.1 Decision Tree

To develop classification systems or prediction models based on numerous variables to reach certain outcomes, the decision tree is a frequently used data mining tool. With the use of this technique, the data is divided into many branches, resulting in a tree-like structure with a root node, internal nodes, and leaf nodes. Without the need for in-depth parameter explanations, it makes it easy to manage large and complex datasets. Data processing, interpretation, prediction, and classification are just a few of the activities that may be performed with the help of decision trees, an efficient statistical tool. One of their primary characteristics is that they are simple and easy to understand. It is crucial to use care when dealing with smaller datasets to avoid possible issues like overfitting or underfitting (Song & Lu, 2015).

The aviation sector has used decision trees for prediction. According to Tang (2021), decision tree algorithm was successfully employed to predict flight delays with a good accuracy of 0.9777. Kuşkapan et al. (2021) classified aviation accidents using the decision tree algorithm and reported that it produced the best results out of all the algorithms they tested. Additionally, the decision tree model was used for forecasting fuel tankering in the aviation sector (Ylmaz et al. 2021).

2.2.3.2 Random Forest

Random forest is a popular machine learning technique for building prediction models. It utilises classification and regression trees, which are fundamental models that leverage binary splits on predictor variables to forecast outcomes. A number of classification and regression trees are constructed by using training datasets and groups of selected variables that are selected at random. The predictions from each tree are then combined to provide predictions for specific data. This technique usually produces improved accuracy as compared to utilising a single decision tree model while preserving some of the advantageous characteristics of tree models (Speiser et al., 2019).

Researchers have used random forests in the aviation sector for a variety of purposes. For instance, Nogueira et al. (2023) developed a predictive model for identifying failures brought on by human factors using random forest as a methodology. According to the study, random forest produced the best outcomes. Anusha et al. (2022) used random forest to forecast flight delays. Setiawan et al. (2023) employed the random forest machine learning analysis

approach, researchers looked at the on-time performance of domestic flights in Indonesia, showing great accuracy in their results.

2.2.3.3 LightGBM (Light Gradient Boosting Machine)

LightGBM, short for Light Gradient Boosting Machine, is a tool developed by Microsoft and offered as an open-source framework. It uses the idea of gradient boosting, where weak prediction models (like decision trees) are combined to create a stronger and more accurate model. What makes LightGBM unique is its leaf-wise growth strategy, which focuses on building leaves in decision trees that lead to the most significant reduction in loss, rather than the traditional level-by-level approach. This approach improves its efficiency and performance. It reduces training timeframes while retaining a high degree of accuracy in comparison to level-wise progression. It is possible to handle large datasets and move swiftly through training without compromising accuracy.

Due to its proficiency in handling big data volumes and delivering top performance, LightGBM is well recognised and often utilised in the aviation sector. LightGBM has been effectively employed in a number of aviation applications by researchers in this area. For instance, Ye et al., (2020) created a flight delay prediction model using LightGBM and discovered that it performed better than conventional models. Bati and Withington (2019) developed a precise method for determining the safety of the National Airspace System (NAS), notably on airport grounds. LightGBM was employed to assist with this procedure. These instances demonstrate how LightGBM is used as a potent tool in aviation research, enabling increased precision in predicting flight delays and the creation of cutting-edge safety indicators.

2.2.3.4 XGBoost (Extreme Gradient Boosting)

XGBoost is an algorithm classified as a gradient boosting method. Its purpose is to optimise the predictive potential of models using distinct trees constructed from several cores and data organisation that minimises lookup times. As a result, the training time is reduced and performance is enhanced. XGBoost is renowned for its efficient execution and rapidity, especially when confronted with structured or tabular datasets that need classification, regression, and predictive modelling (Santhanam et al., 2017).

XGBoost has gained significant recognition for its exceptional performance in categorization challenges. It has been seen to outperform other classification algorithms on structured or tabular datasets and has been demonstrated to be quicker and more accurate in several studies (Santhanam et al., 2017). Many industries have applied XGBoost to implement their prediction. Li and Zhang (2019) developed classification and prediction models for orthopaedic auxiliary components using the XGBoost method. The study's objective was to develop a prediction model for orthopaedic auxiliary categorization using the XGBoost method. The resulting model exhibited enhanced precision, faster computation, and broader suitability in the context of orthopaedic clinical data. The study has applied XGBoost to estimate the status of flights based on the dataset containing all flights of an international airline for a year (Hatıpoğlu et al., 2022).

2.2 Similar Work

Lambelho et al.(2020) explored using machine learning-based predictions for flight delays and cancellations to examine strategic flying plans at London Heathrow Airport. Strategic flight schedules at London Heathrow Airport from 2013 to 2018 are the dataset used in this research. Random forests, LightGBM, and multilayer perceptrons (MLP) are some of the machine learning techniques that have been used by the research to forecast aircraft delays and cancellations (RF). These algorithms are used six months ahead of the day the flight is scheduled to occur to categorise flight cancellations and delays. Recursive feature elimination (RFE) approach, in which weak features are eliminated by feature selection, is one of the pre-processing techniques used in the research. The target encoding technique is also used to encode category characteristics like airline, terminal, aircraft, and nation. The performance of the algorithms was also evaluated in the research by using a 5-fold cross-validation method. the research employs a variety of assessment indicators. The area under the ROC curve, recall, f1-score, accuracy, and precision are some of these measurements (AUC) to measure how well machine learning algorithms perform in predicting aircraft delays and cancellations. Based on the findings, it can be concluded that all classifiers manage to classify flights with delays with an accuracy of at least 0.75 and cancellations with an accuracy of at least 0.98. On the other hand, LightGBM has the greatest area under the ROC curve (AUC) among the models, suggesting that it is better at determining whether aircraft are really delayed or not. Thus, it can

be concluded from this research that LightGBM is the algorithm that performs the best for predicting aircraft delays and cancellations.

This research is mainly about a new mixed method called Random Forest Regression and Maximal Information Coefficient (RFR-MIC) that can be used to predict when a flight will be delay. It focuses on using flight data from several air routes and shows that the model works better than other prediction methods like linear regression, k-nearest neighbours, artificial neural networks, and standard Random Forest Regression. The researchers used departure flight dataset which was collected from Beijing Capital International Airport (PEK). This study used standard data pre-processing steps like cleaning the data, handling the data, normalising the data, and extracting features from the data. The RFR-MIC method was more accurate than other prediction methods in terms of correlation coefficient (R), mean absolute error (MAE), and root mean square error (RMSE). As a whole, 97.1 % of the predicted values were within a 30-minute variance from the actual values, demonstrating the effectiveness of the RFR-MIC approach for departure delay prediction. The lack of reliable essential data on arrival airport information is one of the study's limitations which may affect departure flight delays. Besides that, cloud computing and parallel computing, which shorten calculation times and provide real-time forecasts, were not used in this research. This study offers various recommendations for further research to address the constraints, such as improving the suggested model by including information about departure and arrival airports. The utilization of parallel computing and cloud computing for real-time predictions also can apply (Guo et al., 2021).

The study discussed the implementation of supervised machine learning models to predict flight delays. The research presents an overall comparison of seven machine learning algorithms designed for binary classification of flight delays: Logistic Regression, K-Nearest Neighbor, Gaussian Naïve Bayes, Decision Tree, Support Vector Machine, Random Forest, and Gradient Boosted Tree. The research makes use of flight departure data from JFK airport over a duration of one year. In the data pre-processing stage, integer encoding was used to transform categorical variables into numeric values so that the chosen machine learning algorithms could operate efficiently on the dataset utilised in this research. The study evaluates the effectiveness of these algorithms through the utilisation of metrics including accuracy, precision, recall, and f1-score. The research showed that the Decision Tree algorithm had superior performance compared to the other algorithms, attaining the highest scores of 0.9778 for accuracy, precision, recall, and f1-score. The lowest performance model is K-Nearest Neighbor (KNN) algorithm with an f1-score of 0.8039. Based on a comparison of the machine

learning algorithms used in the study, the Decision Tree algorithm turned out to be the best model for predicting flight delays. The research proposed that future study may investigate at the importance of additional tree-based ensemble algorithms in predicting flight delays, expanding on the discovery that these algorithms perform better with unbalanced data sets. Furthermore, the research indicates using approaches such as SMOTE to address the data set's unbalanced distribution, thereby enhancing prediction accuracy (Tang, 2021).

The paper offered a technique for predicting accumulated flight departure delays in airports using supervised learning approaches. The dataset used in the study was operational data from the Nanjing Lukou International Airport in China. The accuracy and predictability of four common supervised learning approaches were evaluated in the study: multiple linear regression, support vector machine, very randomised trees, and LightGBM. Predictive modelling required acquiring four types of aggregate characteristics connected to airports by data pre-processing. The dataset division method used in the study involved a 5-fold cross-validation method. The hyperparameter tuning technique was utilised to improve the performance of the machine learning models. The study used the R-score, mean square error (MSE) and mean absolute error (MAE) as performance measures. The results showed that the LightGBM model provided the best performance indicated the lowest MAE and higher prediction accuracy (0.8655) compared to the other models. The LightGBM model consistently outperformed the other models in terms of prediction accuracy and capability of handling large-scale data. They recommended that the following study may look at other explanatory factors such national weather, city-pair, and network states to improve the accuracy of airline departure delay predictions (Ye et al., 2020).

Khaksar & Sheikholeslami (2017) presented a study of machine learning techniques for predicting flight delays and identifying the factors that influence delay occurrences in US and Iranian airline networks. They utilized machine learning algorithms such as Bayesian modeling, decision tree, cluster classification, random forest, and hybrid methods. The study used datasets from the US flight network and the Iranian airline network. The pre-processing steps involved data cleaning, data integration, and feature selection. The large Iranian airline dataset required data pre-partition and feature selection to effectively predict flight delays using various machine learning algorithms. The best-performing method is the Hybrid Method with achieved accuracy levels of 71.39% and 76.44% in predicting delay. The class precision for this method ranged from 60.28% to 79.37% and 74.00% to 77.01%. The worst-performing method is Cluster Classification with accuracy levels ranging from 62.27% to 69.63% and class

precision ranging from 53.27% to 76.44%. There are some limitations in this study included the need for continuous updates to the model due to changing airline operations, the potential for overfitting, and the challenge of handling imbalanced datasets. They recommended exploring other data mining methods and combining the hybrid method with robust flight scheduling for further research.

The research proposes that probabilistic forecasting methods, more precisely Random Forest regression and Mixture Density Networks, be used to predict aircraft delays at an airport in Europe. The weather dataset and the aircraft schedule dataset are the two primary datasets used in the research. Data pre-processing involves feature selection and encoding. Pearson Correlation Coefficient is a technique for feature selection. After calculated the correlation between features and the target variable (flight delay), the feature with the smallest correlation with the target variable is removed. A binary delay threshold of 15 minutes is used to target encode categorical characteristics. The study proposes several evaluation metrics including the Continuous Ranked Probability Score (CRPS), RMSE, MAE. For MDN, the Continuous Ranked Probability Score (CRPS) for departures was 9.12, and for arrivals, it was 10.95, with mean CRPS standard deviations of 19.15 and 17.59 for departures and arrivals, respectively. On the other hand, RFR yielded a CRPS of 8.86 for departures and 10.85 for arrivals, with mean CRPS standard deviations of 18.15 and 17.49 for departures and arrivals, respectively. In the context of the provided data, the Random Forest Regression (RFR) algorithm generally yielded lower CRPS for both departures and arrivals compared to the Mixture Density Networks (MDN). Therefore, RFR appears to be the most effective algorithm for probabilistic flight delay prediction in this study (Zoutendijk & Mitici, 2021). The study does not provide any limitations and suggestion.

Flight delay generation and prediction in aviation industry have been shown by the research. It offers methods for predicting flight delays for airports that account for both local and network impacts. The study also focuses on the application of Random Forest algorithms for classification and regression. This study did not mention the process of pre-processing. The evaluation metrics used in the research are accuracy, precision, recall score, area under the ROC curve (AUC score), coefficient of determination, Mean Absolute Error (MAE), and Root Mean Square Error (RMSE). The results of the machine learning models used for flight delay prediction include achieving higher prediction performance compared to existing studies. The accuracy, precision and recall scores for the classification model reached 96.48%, 94.39%, and 90.26% when classifying delays within 15 minutes. The researc also analyzed the importance

of input factors for flight delay prediction using Random Forest algorithms. The research found that the demand-capacity mismatch exerted the most influence, with the congestion of the aviation network following suit. In contrast, the crowdedness of the airport was shown to be the least significant component. The study acknowledges limitations related to the scope of the analysis, such as focusing solely on U.S. domestic flights in July 2018 and not considering extreme weather conditions. They suggested that further studies could expand the analysis to include both domestic and international flights over multiple years and incorporate extreme weather conditions to enhance the prediction model (Li & Jing, 2021).

Wang et al. (2022) used machine learning techniques to forecast the distributions of aircraft delays. The article discusses the use of several machine learning methods, such as random forest, multilayer perceptron (MLP) and LightGBM in the prediction of flight delays. The research paper uses empirical data from Guangzhou Baiyun International Airport. The data includes features such as aircraft, airport, year, day of the week, airline, scheduled departure time, scheduled arrival time, flight frequency, flying time, and hourly flights. It discussed the process of data pre-processing which involves determining and encoding features using corresponding methods. The evaluation metrics used in the research paper include prediction accuracy under given confidence levels, regional area under given interval levels and Wasserstein distance. The results of all the machine learning algorithms are presented the accuracy of prediction of departure delay around 0.65 confidence level and arrival delay at a 0.50 confidence level can be over 0.80. Additionally, the regional area of the arrival delay reaches 0.7 at an interval level of 55 minutes, and the regional area of the departure delay reaches 0.9. The study indicates MLP (quantile) algorithm showed the best performance in predicting departure delay distributions. The research indicated high similarity between the predicted and actual delay distributions. The limitation is the inability to predict cancellations due to data limitations. As for future research, they suggested that more sophisticated models could be constructed to enhance prediction performance. The authors also suggest generating probabilistic delay prediction to the flight-to-gate assignment problem to make the solution more stable.

Hatıpoğlu et al. (2022) has proposed a study on the application of machine learning techniques, including XGBoost, LightGBM, and CatBoost to predict flight delays. The research highlights how crucial it is to predict flight delays in part to avoid bad effects including financial consequences, customer satisfaction problems and environmental problem. The research employed a dataset of flight data from a private airline operator that includes all flights

made throughout the course of the year. The dataset included weather information from the time closest to the aircraft. The dataset used in this work was unbalanced, thus the Synthetic Minority Over-Sampling Technique (SMOTE) was used to filter out observations intended for training before they were used in the training phase. Additionally, the categorical variables were transformed into numerical "one-hot-encoding" values using the transformation technique. The study employed Bayesian hyperparameter optimization for hyperparameter tuning in machine learning models. The study evaluated the performance of the developed models using various performance measures such as accuracy, recall, receiver operating characteristic (ROC) score, and Cohen's Kappa characteristic (ROC) score. LightGBM performed better accuracy (0.967), recall (0.929), Cohen's Kappa (0.924), and F2 score (0.9707). XGBoost had a slightly worse performance. Even though Catboost has the worst results, it is important to note that these results are around 0.90. The recommended model based on the study's findings is LightGBM compared to XGBoost and CatBoost. The study suggests that more research could be done to predict delays in connecting flights or find other ways to transport urgently needed cargo instead of cargo flights.

The journal discusses the use of machine learning algorithms to predict flight delays and study the factors causing them. The data used in this research are the flight data of Hong Kong International Airport collected between March 2018 and April 2018 from the website FlightStats. Several machine learning algorithms used in predicting flight delays, including Support Vector Machine (SVM), Random Forest, K-nearest neighbor, Reinforcement Learning, Naïve Bayes, Decision Tree, and Artificial Neural Networks (ANNs). Data pre-processing included feature selection and data cleaning for their data mining. The train to test ratio has been set as 8:2. The algorithm with highest accuracy is ANNs at 83.17%, while the least accurate is Naïve Bayes at 80.75%. The average of all algorithms is 81.82%. They also investigated the most important factors feature which is rainfall. Significant impact is also observed for the day of departure and type of airline. The size of the aircraft showed less importance on flight delay. The study concluded ANNs is the most effective in predicting flight delay. The limitations of the study are data might be imbalanced and only included only one month data. They recommend future analysis can investigate the underlying reason behind the performance of each algorithm (Yiu et al., 2021).

Summary (Similar Works)

Among the various machine learning algorithms discussed in the provided studies on predicting flight delays, three algorithms that frequently occurred and demonstrated noteworthy performance are LightGBM, Random Forest (RF) and Decision Tree. These algorithms were consistently utilized across multiple studies, showcasing their relevance and effectiveness in the domain of flight delay prediction.

In terms of overall performance, LightGBM and Random Forest stand out as two machine learning types that have demonstrated superior capabilities in predicting flight delays. LightGBM is a gradient boosting framework that has shown high accuracy, recall, and area under the ROC curve (AUC) in many studies. This makes it a popular and useful choice for predicting flight delays. Random Forest is an ensemble learning method that has been used in many studies and has shown to be very good at working with large datasets, with high accuracy and effectiveness.

According to the similar work, most of the research has implemented data pre-processing. The common approaches involve data cleaning, normalization, and feature selection. Recursive feature elimination (RFE) and target encoding for categorical features are two feature selection strategies that are often used to increase the predictive ability of the models. Furthermore, the training data biases have been reduced by using methods like the Synthetic Minority Over-Sampling Technique (SMOTE) to handle unbalanced datasets.

Lastly, evaluation metrics commonly used across the studies include accuracy, precision, recall, F1-score, area under the ROC curve (AUC), mean absolute error (MAE), and root mean square error (RMSE). These metrics provide a comprehensive assessment of model performance, addressing aspects such as classification accuracy, sensitivity, specificity, and the ability to handle regression tasks effectively.

2.3 Technical Research

2.3.1 Programming Language

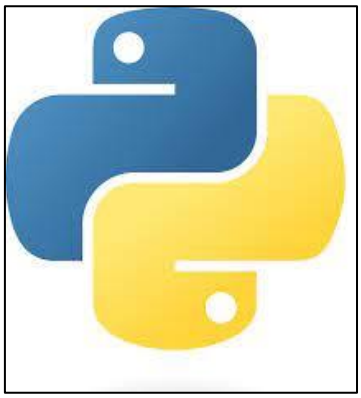


Figure 3:Logo of Python

In the dynamic and ever-evolving realm of machine learning, the choice of programming language plays a pivotal role in determining the effectiveness and efficiency of constructing predictive models. Python and R emerge as formidable candidates, each boasting distinctive advantages tailored to specific facets of the data science journey. This analysis delves into the strengths of Python and R, with the aim of selecting the language that best aligns with my predictive machine learning objectives. By evaluating their respective features, libraries, ease of use, and community support, I aspire to identify the optimal tool for building robust and accurate predictive models that drive data-informed decision-making.

In the present day, Python finds extensive applications across a diverse range of platforms, including popular ones like YouTube, Instagram, Facebook, Dropbox, and Quora, among others. Its widespread adoption in numerous IT organizations enables seamless collaboration and coordination among development teams. Python's versatility is further complemented by an extensive library ecosystem, particularly crucial for data manipulation and management tasks. For instance, leveraging tools like Scikit-learn allows users to effortlessly perform data pre-processing and analysis, unlocking the potential for unprecedented AI capabilities with Python. Additionally, engineers benefit from the Pandas package, which facilitates the creation of robust data structures and simplifies data evaluation, thereby significantly reducing development timelines (Odhiambo et al. 2022). According to Manasa & Velayutham (2023) and Satish et al. (2022), many research has been selected Python as their programming language to develop their machine learning model for predict flight delay.

Figure 4:Logo of R Programming

R programming, developed primarily by statisticians, is purposefully tailored for analysts, evident from its syntax, which emphasizes predictability. R boasts a rich array of invaluable libraries and tools, specifically designed to cater to statistical tasks. Leveraging these resources, R users can capitalize on a growing repository of data analyses, spanning pre- and post-presentation stages, while streamlining cumbersome tasks in the research process, including model validation and data visualization. This feature simplifies the analysis of datasets, empowering data scientists to make informed scientific decisions based on the outcomes of their statistical investigations. Consequently, R stands as an essential programming tool utilized by data scientists to draw meaningful inferences and gain insights from statistical analyses (Odhiambo et al. 2022). This study has been used R programming to predict flight delay (Alla et al., 2021).

R and Python offer distinct advantages and cater to diverse applications depending on the nature of the research. After careful evaluation, I opted for Python as my programming language to build a model for this study due to its superior integration capabilities. Python's functionality surpasses that of R, making it an ideal choice for my research. Moreover, Python-based stacks are seamlessly adaptable to new projects, facilitating the integration of other essential tasks that data researchers might need to perform. Nonetheless, R remains more suitable for statistical analysis and exploratory work. For projects involving initial exploratory work on the measurement model, I would recommend using R. These considerations led me to choose Python after a thorough comparison between the two languages.

2.3.2 IDE (Interactive Development Environment)

Figure 5:Logo of Jupyter Notebook

Jupyter Notebook has been selected as the main Integrated Development Environment for this project. Jupyter Notebook offers a cell-based programming environment that enables users to write and execute code, visualize code outputs, and incorporate hyperlinks, comments, and images. Python code can seamlessly run within this environment. Additionally, various Python packages such as pandas, plotly, and numpy can be imported, granting access to potent data analysis tools through advanced programming commands (Fleischer et al., 2022). Jupyter notebooks provide a structured framework for constructing Machine Learning (ML) models and acquiring high-quality data. The success of these models heavily relies on top-notch data for their training, allowing the creation of shareable models that effectively evaluate the interaction properties of compounds (Smajić et al., 2022).

Figure 6:Logo of Google Colab

Google Colab is a second IDE that is used. Google Colab, which is short for Google Colaboratory, is a free cloud service that gives users access to a fully cloud based Jupyter notebook environment. It is designed to support machine learning and artificial intelligence

research by providing access to free GPUs and TPUs, which are essential for handling the computational demands of these fields. This allows users to leverage significant computational power without the need for expensive hardware (Naik et al., 2022).

Google Colaboratory (Colab) is a useful tool for building models and pre-processing data for machine learning and deep learning applications. Colab notebooks are pre-configured with crucial libraries for machine learning and artificial intelligence including TensorFlow, Matplotlib, and Keras. This allows users to easily write and execute code for model building and data pre-processing without the need to install and configure these libraries themselves. Colab supports the creation of rich, interactive documents that can include live code, visualizations, and narrative text. This is useful for documenting the entire machine learning workflow, including data pre-processing, model building, and evaluation (Carneiro, et al., 2018).

Additionally, the GPU-accelerated runtime provided by Colab can significantly speed up the training of machine learning models, making it faster than using traditional CPU-based systems. However, it's important to note that there are limitations, such as the lack of CPU cores and the 12-hour limit on the virtual machine lifetime, which may impact certain types of processing and model building (Carneiro, et al., 2018).

It is often used to teach machine learning by allowing users to write Python algorithms and text explanations over the browser. It has become an important tool for virtual learning because it is portable and easy to use (Canesche et al., 2021). Those features allow users to interact with the platform easily.

2.3.3Libraries/ Tools

Python libraries consist of modules that encompass valuable code snippets and functions, sparing developers from the necessity of creating them anew. Python boasts an extensive array of libraries, numbering in the tens of thousands, which prove invaluable to machine learning specialists, data scientists, data visualization experts, and others in their respective fields of work. This study will employ a few libraries for data pre-processing, EDA and build machine learning models such as Pandas, Scikit-Learn, NumPy and many more.

The Dask Python library serves the purpose of reading a dataset comprising 6.31 million rows and 60 columns. Dask facilitates computations on large datasets that surpass the memory capacity and can be distributed across multiple nodes. The native compatibility of Dask arrays with the provided data streamlines the analysis of substantial datasets, thereby aiding MetPy users in their exploration of large data collections (May et al., 2022).

Pandas is a Python library that is often used for applications including data science, data analysis, and machine learning. It offers efficient and adaptable data structures, particularly DataFrames, tailored to handle structured data swiftly and intuitively. The prominent characteristics of Pandas encompass its clear and user-friendly syntax, abundant functionality that supports working with missing data, the capability to develop and apply functions on various datasets, high-level abstraction, and a comprehensive set of advanced data structures and manipulation utilities (Saabith et al., 2020).

NumPy holds a crucial position within the realm of machine learning, and its significance lies in its capacity to manage data through arrays, a fundamental requirement for effective mathematical and numerical operations. Notably, NumPy offers rapid precompiled functions for numerical tasks, enhances efficiency through array-based computation, supports object-oriented techniques, and provides the advantage of vectorization, resulting in quicker and more concise calculations (Saabith et al. 2020).

In this research, Matplotlib is utilized, renowned for its robust and visually captivating visualizations. Matplotlib plays a pivotal role in data visualization (Saabith et al., 2020), offering an array of plotting functions like scatter plots, histograms, box plots, and line plots. These tools provide data scientists with insightful knowledge about the distribution, connections, and patterns found in the dataset, which is vital for making wise choices when creating models. Moreover, Matplotlib proves invaluable for depicting model evaluation metrics, such as accuracy, precision, recall, ROC curves, and confusion matrices, post-training predictive models. These visual representations assist in comprehending the model's performance and identifying areas that might require improvement.

Machine learning plays a central role for this research, and Scikit-learn has been utilized as the primary tool. Scikit-learn stands as the most comprehensive open-source Python package for machine learning tasks. The Scikit-learn package relies on Python's fundamental libraries, NumPy and SciPy, and offers seamless integration with other machine learning programming libraries like NumPy and Pandas (Saabith et al., 2020). This comprehensive toolkit

encompasses four essential elements of machine learning. As a result, it serves as a valuable and flexible resource for educational and behavioral statisticians. Besides that, the research has utilized Random Forest and Decision Tree to build machine learning models. Hence, “DecisionTreeClassifier” and “RandomForestClassifier” imported form sklearn to this research. DecisionTreeClassifier in scikit-learn allows you to create and train decision tree models for classification problems. RandomForestClassifier is another class in scikit-learn that represents a random forest classifier. There are the tools, or more precisely, a machine learning algorithm used for classification tasks.

XGBoost and LightGBM are the other libraries that were used in this research for build machine learning model. XGBoost is a distributed gradient boosting library that has been built for maximum efficiency, versatility, and portability. It utilizes the Gradient Boosting framework to build machine learning algorithms. With the help of parallel tree boosting offered by XGBoost, several data science issues may be quickly and precisely resolved (xgboost 2.0.3 documentation, n.d.). LightGBM is a Python library for gradient boosting frameworks. It is an open source, distributed, high-performance implementation of gradient boosting, designed for speed and efficiency. LightGBM is particularly well-suited for large datasets and has become popular in machine learning and data science communities for its speed and scalability. The Python API makes it easy to integrate LightGBM into Python-based machine learning workflows and frameworks (LightGBM 4.2.0.99 documentation, n.d.).

Performance evaluation metrics is also very important in this research. Therefore, “accuracy_score”, “classification_report” and “confusion_matrix” are functions provided for evaluating the performance of machine learning models, particularly for classification tasks that can import from sklearn.metrics. These functions are tools in the sense that they help you assess how well your model is performing on a given dataset.

This research also deployed a simple web application by using Streamlit that user can experience the prediction. Streamlit is a contemporary, straightforward Python package that offers a scripting interface for online applications. Numerous built-in controls and display features including buttons, sliders, text fields, tables, and charts are available in Streamlit. It allows user to create a very simple simulation app. For example, users just enter basic parameter settings then press the execute button. The results will display as a table in the browser. Besides that, Streamlit provide a free hosting service for apps. They utilized Streamlit to build a share simulation models with healthcare users (Harper & Monks, 2023).

2.3.4 Operating System

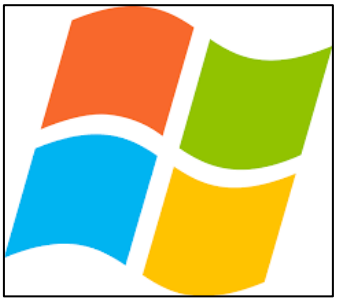


Figure 7: Logo of Windows

The Windows operating system was selected as a good and favourable alternative for machine learning research in this study. Over those years, Windows has significantly improved and solidified its support for data science and machine learning. It is a useful tool for machine learning projects since it is interoperable with several popular programming languages and machine learning packages. Windows provides a variety of tools and programs, including well-known programming environments like PyCharm and Google Colab used by machine learning engineers. In addition, it is simple to utilize for a variety of machine learning applications because to its user-friendly interface.

CHAPTER 3: METHODOLOGY

3.1 Introduction

The significance of methodology in scientific research cannot be overstated, and choosing a suitable approach that aligns with the project's objectives is of utmost importance. The main aim of selecting a methodology is to find a method that harmonizes with the desired outcomes and objectives of the study. This section examines different data mining methodologies and identifies the most appropriate ones for this project. Furthermore, the preliminary data analysis section encompasses essential aspects like data collection, exploratory data analysis (EDA), and data pre-processing.

3.2 Methods

3.2.1 Introduction of Methodology

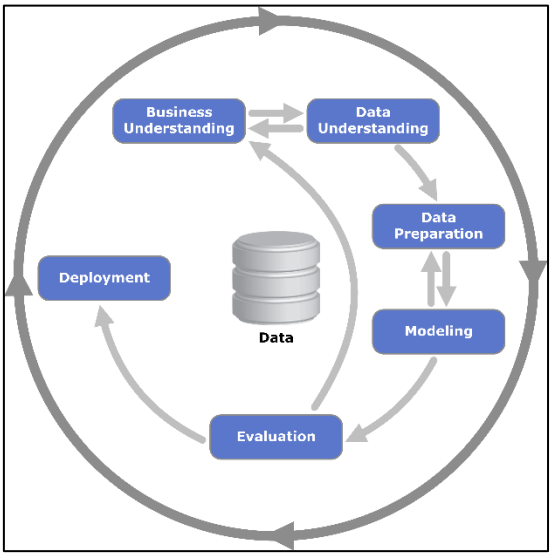


Figure 8: CRISP-DM Process Diagram

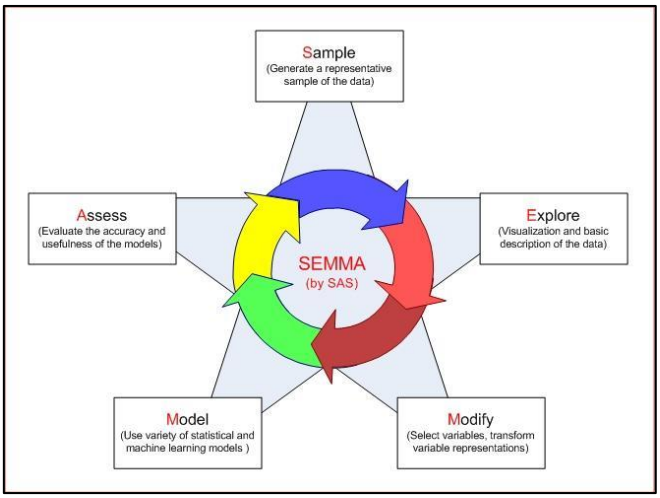


Figure 9:SEMMA

Data mining is the practice of thoroughly studying massive databases in order to identify previously unknown patterns and insights. This analytical approach finds practical uses across diverse fields. There are two methodologies for data mining projects which are CRISP-DM and SEMMA. This research paper presents a comparative analysis of these methodologies to determine the most suitable approach for the project at hand.

The SEMMA process model comprises five primary phases: Sampling, Exploration, Modification, Modeling, and Evaluation. These phases emphasize the modeling aspects of data mining. The workflow stages of the SEMMA model are specifically tailored for use with SAS Enterprise Miner software, making it one of the leading processing models for data analysis. While the SEMMA model is widely applicable, executing iterations and handling interactions among multiple tasks in the original model can present challenges (Firas, 2023). The drawback of this approach is its restricted emphasis on data preparation, as it does not sufficiently tackle the complexities of data cleaning, handling missing values, and performing feature engineering, all of which are crucial for attaining improved model performance.

CRISP-DM follows a structured approach like the waterfall life cycle model, providing a hierarchical framework for carrying out tasks effectively. The six main steps are business understanding, data understanding, data preparation, modelling, performance evaluation, and model deployment. These levels each have activities with clear inputs and outputs. These main tasks apply to all data mining applications and cover the whole data mining process (Firas, 2023). The CRISP-DM methodology comprises various distinct phases, each demanding

substantial time and effort. Consequently, in certain instances, the thoroughness of this process can result in extended project durations. Bardach et al. (2020) has employed CRISP-DM to organize their task for predicting flight delay risk.

3.2.2 Methodology Choice and Justification

After evaluating these two methodologies, CRISP-DM emerges as the more appropriate choice for this project. CRISP-DM is a comprehensive data mining methodology that offers well-defined phases, tasks, and activities. Given the project's focus on building a machine learning model, data pre-processing becomes a crucial step that requires thorough handling. Consequently, SEMMA does not fully meet the project's requirements. Moreover, SEMMA is specifically tailored for use in conjunction with the SAS Enterprise Miner tool, and its documentation primarily revolves around that context. As a result, when dealing with non-standard mining scenarios like the one presented in this study, SEMMA's limitations become more evident and less suitable.

3.2.3 CRISP-DM

CRISP-DM stands for Cross-Industry Standard Process for Data Mining, which divides the data mining process into six distinct phases, as shown in Figure 8. The framework is iterative, allowing for flexibility and adaptation to the specific needs of each project. Those six phases are described in this section below.

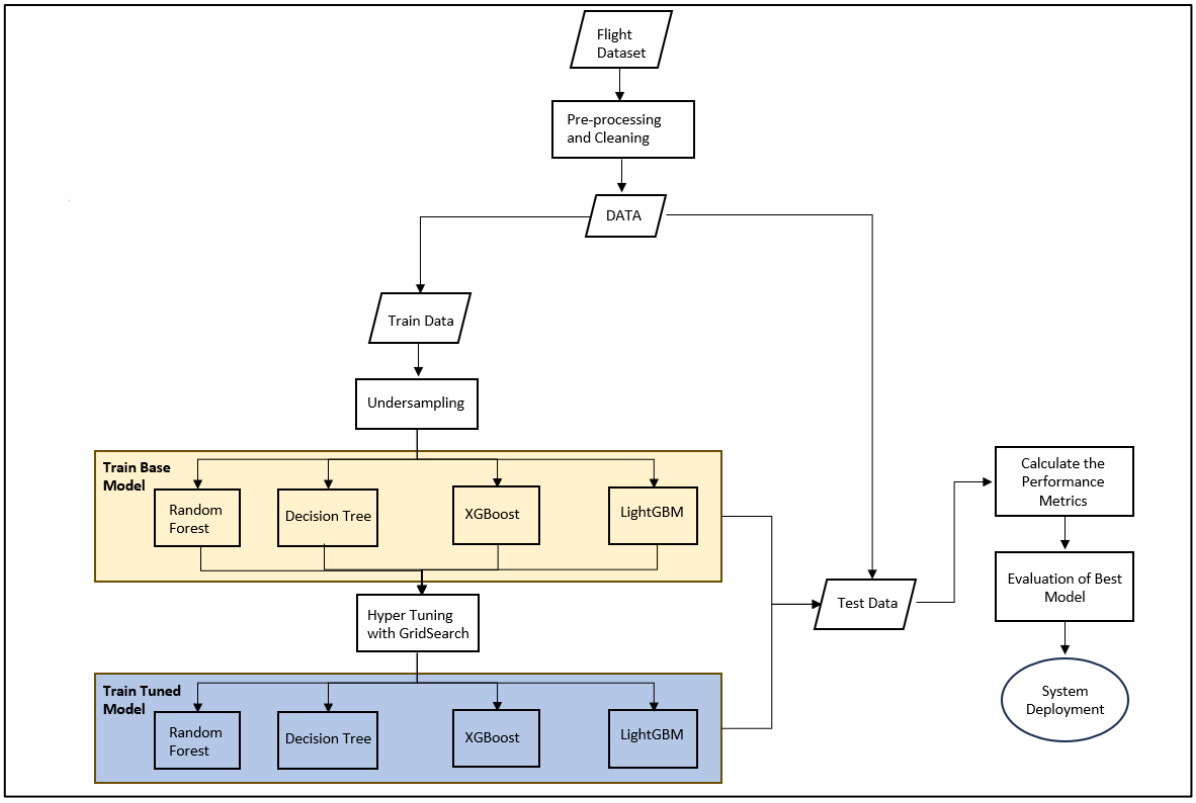


Figure 10: Flowchart

1 Business Understanding

Business understanding marks the initial phase of CRISP-DM. During this stage, it is essential to define and comprehend the project's objectives and requirements from a business standpoint, converting this understanding into a clear definition of the data mining problem. Subsequently, a project plan is developed to effectively achieve the stated objectives (Dåderman & Rosander 2018). This initial plan sets the stage for the subsequent stages of the data mining process, ensuring a structured and purposeful approach toward extracting actionable insights for informed decision-making.

The issue of flight delays has become a serious challenge within the aviation industry, impacting both airlines and passengers alike. Flight delay affect passenger experience, airline reputation and financial loss for airline. Therefore, a plan to forecast aircraft delays was put in place to reduce the impact of flight delays. This research has defined the aim is to leverage machine learning techniques to develop a robust and accurate flight delay prediction model. The objectives are important for achieve the aim. First objective is utilized and compare machine learning algorithms to select the best algorithms. The research will utilize historical flight data, and data related to airport infrastructure to build predictive models. Besides that,

the research will devise methodologies and recommendations aimed at enhancing the dependability and efficiency of flight delay prediction models based on machine learning. Lastly, the research wanted to contribute valuable insights to the current knowledge repository and offer practical information to airlines, airports, and other pertinent stakeholders, enabling them to make well-informed decisions. Chapter 1 of this research has been shown the details of objectives and aim. Based on first phase, it can help to understand the purpose of this research to all the stakeholders.

2 Data Understanding

The second phase of the methodology proposes defining criteria for data selection, acquiring, and examining the data to identify factors influencing its quality (Palacios et al., 2017). Concurrently, the identification of potential data quality issues takes precedence, as ensuring the accuracy and consistency of the dataset is paramount. Any inconsistencies, missing values, or anomalies are addressed promptly to enhance the overall quality of the data. Furthermore, a critical step involves verifying the relevance of the collected data to the specific business problem being addressed.

The research utilized a dataset from Kaggle, encompassing flight information and delays by airline for the entire year of 2021. The dataset contains all flight information including cancellation and delays by airline for dates to January 2021. There are 6.13 million rows and 61 variables included in this dataset. The data will be thoroughly explored using Python’s Exploratory Data Analysis (EDA) techniques. Every variable will be explored to make sure the feature important can be selected. By using EDA, it helps to discover the missing values and duplicate rows. The dataset in its raw form does not explicitly delineate the type of delay but the research focus of the prediction task to classify the flight delay or not. As the purpose was to predict aircraft departure delays but included some cancelled information in the dataset. The researcher better understood data and were able to prepare for the next step through this phase. Chapter 4 will describe more details of this research implementation the data understanding.

3 Data preparation

The third phase involves the selection, cleansing, and conversion of the collected data into the necessary format. As data mining processes often involve data from diverse sources and formats, data preparation becomes crucial for obtaining accurate and reliable outcomes (Bardach et al., 2020). The phase involves the careful selection of relevant variables, followed by a meticulous cleaning process to rectify any inconsistencies and inaccuracies. The data is then transformed into a format conducive to analysis, ensuring compatibility with chosen modelling techniques. After that, addressing missing values and outliers is imperative during this stage, employing methods such as imputation or removal to maintain data integrity. Moreover, to augment the analysis, new variables or features are created strategically, contributing valuable dimensions that may enhance the predictive capacity of the models.

After data understanding, the pre-processing steps primarily involve data cleaning to ensure the reliability of the analysis. Feature selection was carried out, and 13 columns were chosen for further consideration. As part of the data cleaning process, all rows corresponding to cancelled flights were removed, and duplicate entries were meticulously eliminated to enhance data integrity. The subsequent transformation involved utilizing the “delay minutes” column to categorize flights as delayed or not. To prepare the dataset for modelling, a label encoder was applied to convert object-type columns into numerical formats, a crucial step for certain machine learning models. Recognizing the imbalance in the dataset, an under-sampling technique was then employed to address this issue post data split. Chapter 4 will describe more details that research prepared the data.

4 Modelling

The fourth stage of CRISP-DM is known as the modelling phase. The modelling phase is a pivotal step in the data mining process, marked by the careful selection of modelling techniques to address the business problem and the inherent characteristics of the available data (Hotz, 2023). This involves a thoughtful consideration of various algorithms and approaches that align with the specific objectives of the analysis. Subsequently, the dataset is partitioned into distinct training and testing sets, a crucial step to ensure the model’s effectiveness and generalization to unseen data. This phase allows for the refinement and selection of the most effective model that demonstrates optimal performance and predictive capability, setting the stage for the subsequent phases in the data mining life cycle.

In the modelling phase, this research references upon previous studies that have employed machine learning for flight delay prediction. Several machine learning algorithms were suggested in earlier research, and from these, LightGBM and Random Forest were identified as having been utilized effectively. However, this investigation revealed that XGBoost and Decision Tree despite their potential, have seldom been applied in this context. As a result, these two machine learning algorithms were selected for inclusion in the modelling process. After build the models, all four machine learning algorithms and subsequently assess for potential issues such as overfitting or underfitting. Lastly, hyperparameter tuning will be conducted using a grid search method, optimizing the parameters for each algorithm to refine the performance of the models. A grid search is designed by a set of fixed parameter values which are essential in providing optimal accuracy based on n-fold cross-validation (Shekar & Dagnew, 2019). Chapter 2 has explained all the selected machine learning. Chapter 4 also included the details of model building.

5 Evaluation

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1 - Score = 2 \times \frac{Recall \times Precision}{Recall + Precision}$$

Figure 11:Calculation of Evaluation Metrics

The evaluation phase in CRISP-DM, the fifth stage, focuses on thoroughly assessing the accuracy of predictive models developed in the previous phase. The models are tested using either a separate dataset or cross-validation methods to ensure their effectiveness with new and unfamiliar data (Hotz, 2023). The performance of the constructed models is meticulously assessed using the testing data. This involves a thorough examination of how well the models generalize to new, unseen data, providing a critical measure of their effectiveness. The results obtained are then validated against the predefined business objectives, ensuring alignment with the overarching goals of the data mining project. This step is crucial in determining not only the predictive accuracy of the models but also their practical relevance and utility in addressing the identified business problem.

Flight delay prediction in this research has been identified as a classification task, where the goal is to categorize flights into delayed or non-delayed classes. In this context, accuracy is considered the primary evaluation metric, as it measures the overall correctness of the model predictions. However, acknowledging the specific nuances of flight delay scenarios, precision, recall, and F1 score are equally crucial metrics to assess the model's performance comprehensively. Figure 11 has shown the calculation of each metrics. By selecting and comparing these metrics for each machine learning model, a thorough evaluation is conducted, ensuring that the chosen model not only achieves high overall accuracy but also excels in correctly identifying and predicting flight delays. Chapter 5 will describe the performance of each algorithm.

Deployment

Deployment is the final phase of CRISP-DM after successfully built of the flight delay prediction model. The complexity of this phase varies depending on the project's requirements, ranging from a straightforward delivery of the model to a more intricate process involving the integration of the model into an operating system. Creating a deployment plan that outlines the necessary steps for the deployment process is critical during this phase (Hotz, 2023). This involves seamlessly integrating the model into the operational environment of the airline or relevant stakeholders. A comprehensive plan for ongoing monitoring and maintenance is crucial to ensure the continued accuracy and effectiveness of the model in real-world scenarios. This includes regular updates and adaptations to accommodate changing variables and evolving patterns. Additionally, to facilitate user adoption and maximize the model's impact, thorough documentation and training materials are provided for end-users.

In this research, a simple system for making real-time predictions on new data will be created. The developed system will serve as a practical tool for users, providing them with the capability to anticipate potential flight delays in the United States. The system is meticulously designed with a user-friendly interface, ensuring ease of use and accessibility for a diverse range of users. Intuitive features and a straightforward design aim to enhance the overall user experience, making it easy for individuals, whether they are frequent travellers, airline personnel, or other stakeholders, to navigate and utilize the predictive capabilities of the system seamlessly.

3.3 Summary

Chapter 3 focuses on the research's methodology and preliminary data analysis. SEMMA and CRISP-DM, two data mining approaches, are contrasted. CRISP-DM is selected as the optimal approach for this dataset after extensive deliberation. The next part of the section goes into detail on how CRISP-DM will be used to process the dataset. The examination of preliminary data is covered in the next section (4.3).

CHAPTER 4: DESIGN AND IMPLEMENTATION

4.1 Introduction

Chapter 4 focuses on preparing and understanding the data for effective model building. It starts by explaining how the data was collected, highlighting sources and methods. The chapter then moves into exploratory data analysis (EDA), where statistical insights and relationships between variables, especially with the target variable are explored. Data pre-processing process with EDA. The steps such as handling missing values and applying label encoder. This provides the basis for building models after that on, making sure that accurate predictions can be made.

4.2 Data Collection and Data Understanding

4.2.1 Data Collection

This research utilizes the dataset Combined_Flights_2021, which is available on Kaggle. The information is saved in the widely used Comma-Separated Values (CSV) format, which is known for its simplicity. This extensive dataset contains detailed flight information, including airline-specific cancellation and delay records, for dates up to and including January 2021. The dataset's origin can be traced back to the Marketing Carrier On-Time Performance data table, which is part of the TranStats data library's "On-Time" database.

Link: https://www.kaggle.com/datasets/robikscube/flight-delay-dataset-20182022?select=Combined_Flights_2021.csv

4.2.2 Data Understanding

4.2.2.1 Variables

Variables	Description
Airline	Name of Airline Company
Year	Year
Quarter	Quarter (1-4)
Month	Month
DayofMonth	Day of Month
DayOfWeek	Day of Week
FlightDate	Flight Date (yyyymmdd)
Marketing_Airline_Network	Unique Marketing Carrier Code. When the same code has been used by multiple carriers, a numeric suffix is used for earlier users, for example, PA, PA(1), PA(2). Use this field for analysis across a range of years.
Operated_or_Branded_ Code_Share_Partners	Reporting Carrier Operated or Branded Code Share Partners
SSDOT_ID_Marketing_Airline	An identification number assigned by US DOT to identify a unique airline (carrier). A unique airline (carrier) is defined as one holding and reporting under the same DOT certificate regardless of its Code, Name, or holding company/corporation.
IATA_Code_Marketing_Airline	Code assigned by IATA and commonly used to identify a carrier. As the same code may have been assigned to different carriers over time, the code is not always unique. For analysis, use the Unique Carrier Code.
Flight_Number_Marketing_Airline	Flight Number
Operating_Airline	Unique Carrier Code. When the same code has been used by multiple carriers, a numeric suffix is used for earlier users, for example, PA, PA(1), PA(2). Use this field for analysis across a range of years.
DOT_ID_Operating_Airline	An identification number assigned by US DOT to identify a unique airline (carrier). A unique airline (carrier) is defined as one holding and reporting under the same DOT certificate regardless of its Code, Name, or holding company/corporation.
IATA_Code_Operating_Airline	Code assigned by IATA and commonly used to identify a carrier. As the same code may have been assigned to different carriers over time, the code is not always unique. For analysis, use the Unique Carrier Code.
Tail_Number	Tail Number
Flight_Number_Operating_Airline	Flight Number
OriginAirportID	Origin Airport, Airport ID. An identification number assigned by US DOT to identify a unique airport. Use this field for airport analysis across a range of years because an airport can change its airport code and airport codes can be reused.

OriginAirportSeqID	Origin Airport, Airport Sequence ID. An identification number assigned by US DOT to identify a unique airport at a given point of time. Airport attributes, such as airport name or coordinates, may change over time.
OriginCityMarketID	Origin Airport, City Market ID. City Market ID is an identification number assigned by US DOT to identify a city market. Use this field to consolidate airports serving the same city market.
Origin	Origin Airport
OriginCityName	Origin Airport, City Name
OriginState	Origin Airport, State Code
OriginStateFips	Origin Airport, State Fips
OriginStateName	Origin Airport, State Name
OriginWac	Origin Airport, World Area Code
DestAirportID	Destination Airport, Airport ID. An identification number assigned by US DOT to identify a unique airport. Use this field for airport analysis across a range of years because an airport can change its airport code and airport codes can be reused.
DestAirportSeqID	Destination Airport, Airport Sequence ID. An identification number assigned by US DOT to identify a unique airport at a given point of time. Airport attributes, such as airport name or coordinates, may change over time.
DestCityMarketID	Destination Airport, City Market ID. City Market ID is an identification number assigned by US DOT to identify a city market. Use this field to consolidate airports serving the same city market.
Dest	Destination Airport
DestCityName	Destination Airport, City Name
DestState	Destination Airport, State Code
DestStateFips	Destination Airport, State Fips
DestStateName	Destination Airport, State Name
DestWac	Destination Airport, World Area Code
DepTime	Actual Departure Time (local time: hhmm)
DepDelay	Difference in minutes between scheduled and actual departure time. Early departures show negative numbers.
DepDel15	Departure Delay Indicator, 15 Minutes or More (1=Yes)
DepartureDelayGroups	Departure Delay intervals, every (15 minutes from 180)
DepTimeBlk	CRS Departure Time Block, Hourly Intervals
TaxiOut	Taxi Out Time, in Minutes
WheelsOff	Wheels Off Time (local time: hhmm)
WheelsOn	Wheels On Time (local time: hhmm)

TaxiIn	Taxi In Time, in Minutes
CRSArrTime	CRS Arrival Time (local time: hhmm)
ArrDelay	Difference in minutes between scheduled and actual arrival time. Early arrivals show negative numbers.
ArrDelayMinutes	Difference in minutes between scheduled and actual arrival time. Early arrivals set to 0.
ArrDel15	Arrival Delay Indicator, 15 Minutes or More (1=Yes)
ArrivalDelayGroups	Arrival Delay intervals, every (15-minutes from 180)
ArrTimeBlk	CRS Arrival Time Block, Hourly Intervals
Cancelled	Cancelled Flight Indicator (1=Yes)
DepDelayMinutes	Difference in minutes between scheduled and actual departure time. Early departures set to 0.
Diverted	Diverted Flight Indicator (1=Yes)
CRSElapsedTime	CRS Elapsed Time of Flight, in Minutes
ActualElapsedTime	Elapsed Time of Flight, in Minutes
Distance	Distance between airports (miles)
DistanceGroup	Distance Intervals, every 250 Miles, for Flight Segment
DivAirportLandings	Number of Diverted Airport Landings
ArrTime	Actual Arrival Time (local time: hhmm)
CRSDepTime	CRS Departure Time (local time: hhmm)
AirTime	Flight Time, in Minutes

Table 1: Description of Variables of the Flight dataset

Table 1 shows all the descriptions of the variables. There are 61 variables included in this dataset. Possible variables that affecting flight delay will be explored in EDA. However, variables which cannot affecting the target variable will be removed.

4.2.2.2 Import Library and Load Data

```
import pandas as pd
import dask.dataframe as dd
import matplotlib.pyplot as plt
import plotly.express as px
dataset_path = "/content/drive/My Drive/Dataset/Combined_Flights_2021.csv"
df = dd.read_csv(dataset_path)
```

Figure 12: Code of import library, load data show first five rows

```
df.head()
```

	FlightDate	Airline	Origin	Dest	Cancelled	Diverted	CRSDepTime	DepTime	DepDelayMinutes	DepDelay	...	WheelsOff	WheelsOn	TaxiIn	CRSArrTime	Ar
0	2021-03-03	SkyWest Airlines Inc.	SGU	PHX	False	False	724	714.0	0.0	-10.0	...	724.0	813.0	5.0	843	
1	2021-03-03	SkyWest Airlines Inc.	PHX	SGU	False	False	922	917.0	0.0	-5.0	...	940.0	1028.0	3.0	1040	
2	2021-03-03	SkyWest Airlines Inc.	MHT	ORD	False	False	1330	1321.0	0.0	-9.0	...	1336.0	1445.0	16.0	1530	
3	2021-03-03	SkyWest Airlines Inc.	DFW	TRI	False	False	1645	1636.0	0.0	-9.0	...	1703.0	1955.0	7.0	2010	
4	2021-03-03	SkyWest Airlines Inc.	PHX	BFL	False	False	1844	1838.0	0.0	-6.0	...	1851.0	1900.0	3.0	1925	

5 rows × 61 columns

Figure 13: Output of first five rows

Figure 12 is the code that imports the library that need to use. As mentioned above, the dataset that employed is Combined_Flights_2021. The size of this dataset is 2GB, so it requires more time to read. Figure 13 shows the output of the first five rows of data. “df.head()” helps to display the first five rows to make sure the data loaded successfully.

```
# Identify variables and data types
df.info()
```

```
<class 'dask.dataframe.core.DataFrame'>
Columns: 61 entries, FlightDate to DivAirportLandings
dtypes: object(18), bool(2), float64(19), int64(22)
```

Figure 14: Code and Output of Identify variables

It can be observed that there are 61 variables as mentioned above from the Figure 14. It is also observed that the data type is divided into four like bool, float64, int64 and string through this code. This shows that the data is very large. Some of the variables do not need to be used in this project. Therefore, a feature selection is applied. Data transformation has been utilized to category the type of delay based on *DepDelMinitues* for creating a new variable as target variables. **Those data pre-processing is important before EDA for better data understanding.**

4.3.2.2 Exploratory Data Analysis (EDA)

4.3.2.2.1 Understand the variable selected and data type

```
df_pandas.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6311871 entries, 0 to 183987
Data columns (total 13 columns):
 #   Column           Dtype  
---  ------           -----  
 0   Airline          object 
 1   OriginCityName   object 
 2   Diverted         bool   
 3   DistanceGroup    int64  
 4   Quarter          int64  
 5   Month            int64  
 6   DayofMonth       int64  
 7   DayOfWeek        int64  
 8   TaxiOut          float64
 9   DepDelayMinutes  float64
 10  DepTime          float64
 11  Cancelled        bool   
 12  DepTimeBlk       object 
dtypes: bool(2), float64(3), int64(5), object(3)
memory usage: 589.9+ MB
```

Figure 15: Code and Output of data type (Each Variable)

There are 13 variables that has been selected and 6311871 observations after exploring the data. Table 1 has shown the description of each variables. Figure 15 has shown the variables selected and the data types.

The “Airline”, “OriginCityName” and “DepTimeBlk” columns are the object data type. Object data type typically represents categorical variables. In the case of this dataset, these columns likely contain categorical information, such as the airline name, origin city name, and departure time block. However, some of the machine learning cannot use the object type.

Hence, convert categorical variables into numerical representations using techniques like one-hot encoding will become necessary.

The “Diverted” and “Cancelled” columns are bool data type, indicating binary values representing whether a flight was diverted or cancelled, respectively. Boolean variables are often used to denote true/false or yes/no conditions.Variable “Cancelled” will be filtered that mentioned in data pre-processing because this dataset included the flight has been cancelled. The purpose of this pre-processing phase is to guarantee that the information from cancelled flights does not interfere with the analysis or prediction work you are attempting to accomplish, which is predicting delays.

The “DistanceGroup”, “Quarter”, “Month”, “DayofMonth” and “DayOfWeek” columns are int64 data type, indicated they contain integer values. These columns likely represent different temporal and distance-related features, such as the quarter of the year, the month, the day of the month, and the day of the week.

The “TaxiOut”, “DepDelayMinutes” and “DepTime” columns are float64 data type. Float data types are used for numerical variables with decimal points. In this context, these columns likely represent features such as taxi-out time, departure delay in minutes, and departure time. “DepTime” is the actual fligh departure time. In order to facilitate calculation, analysis, and model training, the data type will be converted to a standard 4-digit format free of non-digit characters.

4.3.2.2.2 Check flight has been cancelled

```
cancelled_counts = df_pandas['Cancelled'].value_counts()

# Print the result
print(cancelled_counts)
```

```
False    6200853
True      111018
```

Figure 16:Code and Output of Cancelled Column

As mentioned at the first, the data included the flights cancelled. The result(Figure 16) has shown 111018 flights cancelled. Those rows need to be delete as this research aim is to predict flight delay. It is assumed that the cancelled flights are not related to the delay, so it is necessary to remove the rows that flight cancelled.

4.3.2.2.3 Check Missing Value

```
missing_values = df_pandas.isnull().sum()

print("Missing values in each column:")
print(missing_values)
```

```
Missing values in each column:
Airline                 0
OriginCityName          0
Diverted                0
DistanceGroup           0
Quarter                 0
Month                   0
DayofMonth              0
DayOfWeek               0
TaxiOut            110353
DepDelayMinutes    108413
DepTime            108325
Cancelled               0
DepTimeBlk              0
dtype: int64
```

Figure 17: Code and Output of missing value (Each Variables)

The code snippet in Figure 17 determines the number of columns of this dataset that have missing data. There are some missing values in the columns of "TaxiOut", "DepDelayMinutes" and "DepTime". Those missing values need to be removed at data pre-processing.

4.3.2.2.4 Check Duplicate Rows

```
# Find and count duplicate rows
duplicate_rows = df_pandas[df_pandas.duplicated()]
num_duplicates = len(duplicate_rows)

# Print the duplicate rows and the count
print(f"Number of Duplicate Rows: {num_duplicates}")
```

```
Number of Duplicate Rows: 15519
```

Figure 18: Code and Output of Check Duplicate Rows

Checking for and addressing duplicate rows is important in the process of building machine learning model. Duplicate rows can unnecessarily increase the size of the dataset, leading to inefficient use of computational resources during model training. The result (Figure 18) has shown 15519 duplicate rows. Same as missing value, the process of delete duplicate rows will be done at data pre-processing.

4.3.2.2.5 Statistics

```
df_pandas.describe()
```

	DistanceGroup	Quarter	Month	DayofMonth	DayOfWeek	TaxiOut	DepDelayMinutes	DepTime
count	6.198832e+06	6.198832e+06	6.198832e+06	6.198832e+06	6.198832e+06	6.198832e+06	6.198832e+06	6.198832e+06
mean	3.657727e+00	2.651057e+00	6.972132e+00	1.578962e+01	4.014597e+00	1.619464e+01	1.274392e+01	1.326773e+03
std	2.285997e+00	1.076522e+00	3.297312e+00	8.786910e+00	2.003847e+00	8.583950e+00	4.729910e+01	4.868175e+02
min	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	0.000000e+00	1.000000e+00
25%	2.000000e+00	2.000000e+00	4.000000e+00	8.000000e+00	2.000000e+00	1.100000e+01	0.000000e+00	9.230000e+02
50%	3.000000e+00	3.000000e+00	7.000000e+00	1.600000e+01	4.000000e+00	1.400000e+01	0.000000e+00	1.321000e+03
75%	5.000000e+00	4.000000e+00	1.000000e+01	2.300000e+01	6.000000e+00	1.900000e+01	6.000000e+00	1.729000e+03
max	1.100000e+01	4.000000e+00	1.200000e+01	3.100000e+01	7.000000e+00	2.560000e+02	3.095000e+03	2.400000e+03

Figure 19: Code and Output of the numeric variable's statistics

Figure 19's code snippet is used to create descriptive statistics for numeric variables. It summarizes the dataset's central tendency, dispersion, and distribution of numerical columns. This information is quite useful for establishing an early grasp of the distribution and properties of the data. It may assist you in identifying outliers, ensuring data integrity, and evaluating the extent and dispersion of numerical characteristics.

The table shows that variables “Month” and “DayofMonth” are complete month and day in 2021. “DistanceGroup” is based on the distance intervals which every 250 Miles. 1 represents 0-250 Miles is the smaller distance group between airport and 11 is the largest distance which included intervals is 2500 till 2750 Miles. There are four quarter as usual. The period between when an aircraft leaves the gate or parking lot and when it lifts off from the runway is known as the “taxi-out” time. “TaxiOut” can found that 16 min on average to taxi out. The longest time is 256 mins. This mean “DepDelayMinutes” show the delay time in 12 mins. The longest delay time is 51 hours.

4.3.2.2.6 DepDelayMinutes

```
delay_minutes = df_pandas.query('(DepDelayMinutes < 30)')

# Create a histogram
plt.figure(figsize=(8, 6))
plt.hist(delay_minutes['DepDelayMinutes'], bins=30, edgecolor='black')
plt.xlabel('DepDelayMinutes (min)')
plt.ylabel('Frequency')
plt.title('Histogram of DepDelayMinutes')
plt.tight_layout()

plt.show()
```

Figure 20: Code of Condition and Histogram (<30)

```
delay_minutes = df_pandas.query(
    '(DepDelayMinutes > 0) and (DepDelayMinutes < 61)')

# Create a histogram
plt.figure(figsize=(8, 6))
plt.hist(delay_minutes['DepDelayMinutes'], bins=60, edgecolor='black')
plt.xlabel('DepDelayMinutes (min)')
plt.ylabel('Frequency')
plt.title('Histogram of DepDelayMinutes (Delay > 0 min)')
plt.tight_layout()

plt.show()
```

Figure 21: Code of Condition and Histogram (>0 and <61)

Figure 20 has shown the code to create the histogram based on the condition which departure delay minutes less than 30 minutes by using query. There are some settings is for the bin that has set as 30. Figure 21 is for creating other histogram based on the condition which departure delay minutes greater than 0 minutes and less than 61 by using query after interpreting histogram 1 for more accuracy.

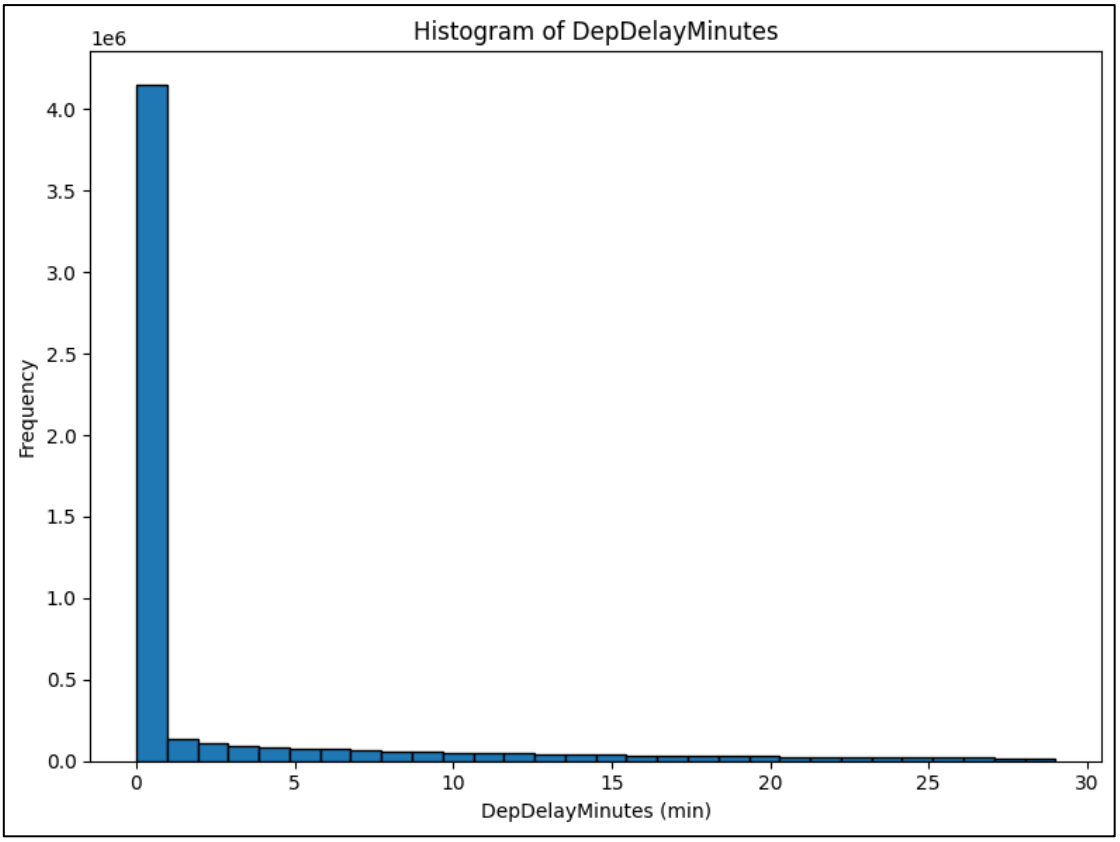


Figure 22: Histogram of DepDelayMinutes

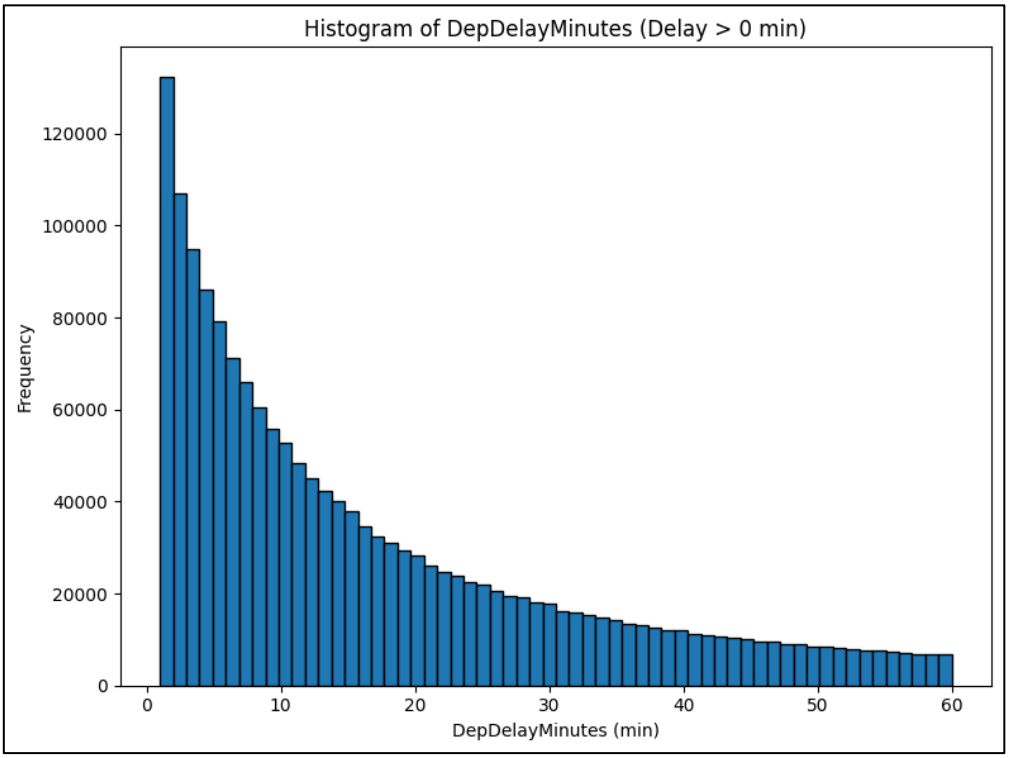


Figure 23: Histogram of DepDelayMinutes(Delay>0)

Most flights experienced no delay (0 min) as shown by the Figure 22, and it can be observed that the maximum accumulation, “0”, obscures any actual delays. The other histogram has been constructed using the code of the figure with a value larger than 0 and less than 61 to examine the delay minutes distribution more clearly. Figure 23 demonstrates that delays may range in length from a few minutes to a full hour and beyond. In this study, delays with times greater than 0 are considered delays. Data transformation will be a method to build a category variable as target variable based on “DepDelayMinutes” that is used to determine whether each aircraft is delayed or not after interpreting Figure 22 and Figure 23.

4.3.2.2.7 Target Variable (Delay Types)

```
from collections import Counter
from tabulate import tabulate

# Count the occurrences of each group
delgroups_count = Counter(df_pandas['DepDelTypes'])

# Calculate the total count
total_count = len(df_pandas['DepDelTypes'])

# Create a list of dictionaries to represent the table
table_data = [
    {'DelTypes': group, 'Count': count} for group, count in delgroups_count.items()
]
table_data.append({'DelTypes': 'Total', 'Count': total_count})

# Print the table using tabulate
print(tabulate(table_data, headers='keys', tablefmt='grid'))
```

Figure 24: Code of Condition and Histogram

Figure 24’s code snippet was used to discover the distribution of delay kinds and present them in tabular form. “DepDelTypes” column has used the “Counter” class. table_data to count the occurrences of each delay type in the. To express the total count row, add() a dictionary to the table_data list. It changes the value of “DelTypes” to “Total” and the value of “Count” to the previously determined total_count. The last code uses the tabulate function to print the table.

DelTypes	Count
0	4144885
1	2053997
Total	6198882

Figure 25: Distribution of Delay Types

Figure 25 shows the distribution of delay types based on 1 as Delay and 0 as Not Delay. There are 4144885 flights that are not delayed. However, there are 2053997 flights delayed. There are a total of 6198882 flights in this dataset after removing the noise data.

```
value_counts = df_pandas['DepDelTypes'].value_counts()

# Step 3: Create the pie chart
plt.figure(figsize=(8, 8))
plt.pie(value_counts, labels=value_counts.index,
        autopct='%1.1f%%', startangle=140)
plt.title(f'Distribution of DepDelGroups for ')
plt.axis('equal')

# Show the chart
plt.show()
```

Figure 26: Code of Create Pie Chart

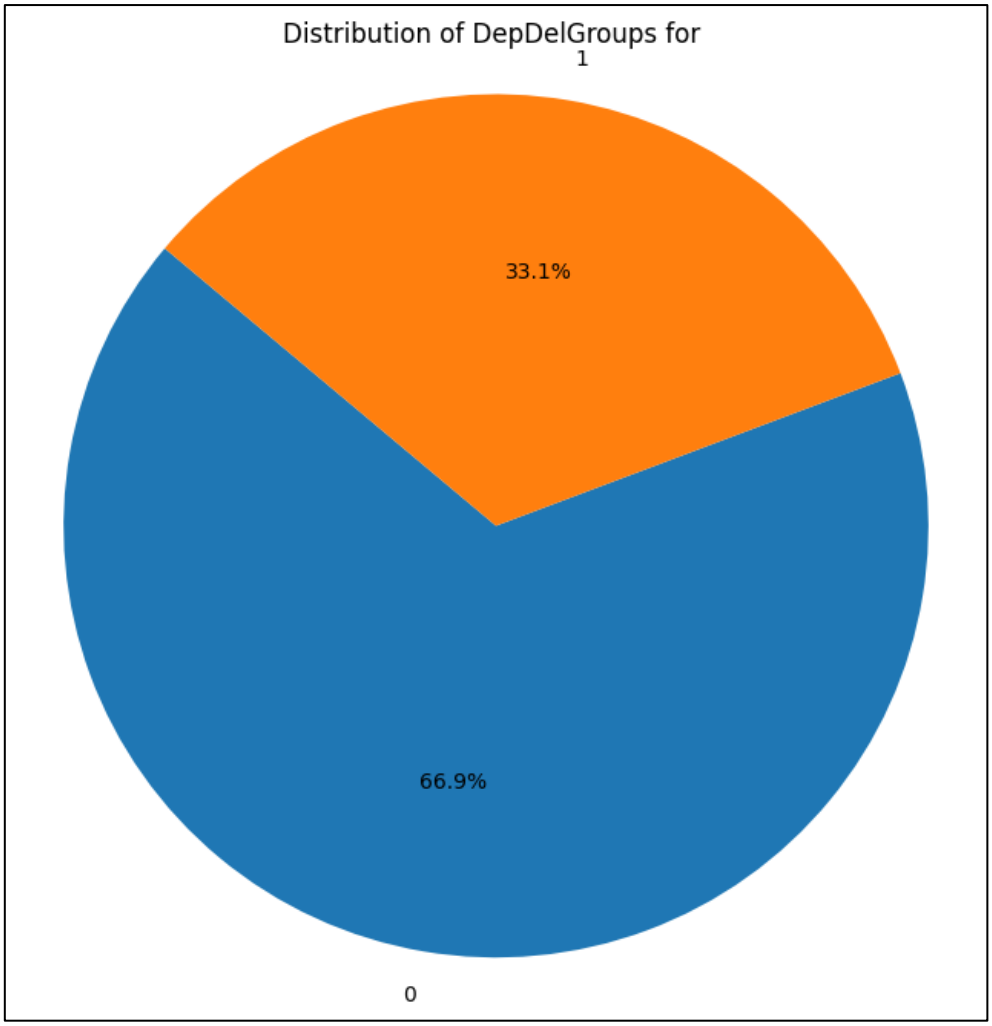


Figure 27: Pie Chart of Percentages of Delay Types

The pie chart of Figure 27 and table (Figure 25) provided can deduce that in 2021, roughly 66.9% of flights in the United States saw no delays(0), while around 33.1% of flights suffered delays(1). This implies that many flights (about one-third) were delayed. The severity of the delay is determined by several variables, including the context, industry norms, and passenger expectations. In general, a delay rate of 33.1% may be cause for worry, particularly if it suggests a major effect on travel plans and passenger experience.

4.3.2.2.8 Airline

```
# Count of AIRLINE
airline = set(df_pandas['Airline'])
airline_count = len(airline)
print("Number of airline:", airline_count)
```

```
Number of airline: 22
```

Figure 28: Code and output of count airline number

This piece of code is used to count the number of airlines and display the result in the "Airline" column. Using a set ensures that only distinct airline names remain, eliminating the need for manual culling. The "len" function counts the various air companies. According to the results, there will be 22 different airlines operating in the United States during 2021.

```
from tabulate import tabulate
airline_delay_count = df_pandas.groupby(
    ['Airline', 'DepDelTypes']).size().reset_index(name='Count')

# Pivot the table to have airlines as rows and delay groups as columns
airline_delay_pivot = airline_delay_count.pivot(
    index='Airline', columns='DepDelTypes', values='Count').fillna(0)

# Add a new column 'Total' containing the total count for each airline
airline_delay_pivot['Total'] = airline_delay_pivot.sum(axis=1)

# Sort the airlines based on the 'Total' count in descending order
airline_delay_pivot = airline_delay_pivot.sort_values(
    by='Total', ascending=False)

# Print the table using tabulate
print(tabulate(airline_delay_pivot, headers='keys', tablefmt='grid'))
```

Figure 29: Code of table by airline and delay type

The code of Figure 29 changes the data and makes a table that shows the number of each type of delay there is for each flight. The 'Total' column is used to arrange the data in decreasing order. As a result, the airlines are reordered such that the ones with the biggest total count are shown first.

Airline	0	1	Total
Southwest Airlines Co.	484099	556905	1.041e+06
Delta Air Lines Inc.	534156	209671	743827
SkyWest Airlines Inc.	554960	184760	739720
American Airlines Inc.	472339	247373	719712
United Air Lines Inc.	282650	157961	440611
Republic Airlines	261287	65949	327236
Endeavor Air Inc.	225278	40437	265715
Envoy Air	185679	64236	249915
Comair Inc.	168333	51911	220244
JetBlue Airways	115245	84153	199398
Alaska Airlines Inc.	125783	59953	185736
Spirit Air Lines	119899	65793	185692
Mesa Airlines Inc.	107861	47128	154989
Frontier Airlines Inc.	88836	46372	135208
Allegiant Air	64469	47583	112052
Horizon Air	79908	29600	109508
Capital Cargo International	80347	17627	97974
Air Wisconsin Airlines Corp	57620	20190	77810
Commutair Aka Champlain Enterprises, Inc.	51098	23053	74151
Hawaiian Airlines Inc.	42181	18095	60276
GoJet Airlines, LLC d/b/a United Express	42767	15225	57992
Empire Airlines Inc.	90	22	112

Figure 30:Table of Delay Types (Airline)

Figure 30 shows the total number of flights, the number of flights that had no delays and the number of flights that experienced a delay for each airline. Southwest Airlines Co. had 1041140 flights with 556905 delays and 484099 flights without delay. Southwest Airlines Co. comes out as the airline with the most total flights in the dataset as well as the most delays. Southwest Airlines Co. is a key participant in the United States aviation business, operating many flights on a variety of routes and destinations.

Following that, Delta Air Lines Inc. is the second biggest airline in terms of total trips with 7443827 flights. Delta Air Lines Inc compares with Southwest with a significant

proportion experiencing no delays (534156 flights). However, there is also a notable number of delayed flights (209671), suggesting that Delta Air Lines has a mix of on-time and delayed departures. Delta is also a big player in the US airline business, with many trips to many different places.

SkyWest Airlines Inc., with 739720 flights, which is close to Delta has, making it another big player in the US airline market. Most of these flights (554960) experienced no delays, but there is still a significant number of delayed flights (184760). American Airlines has total of 719712 flights, 472339 were on time, while 247373 experienced delays.

Empire Airlines Inc. has a comparatively lower total number of flights (112) compared to the previously mentioned major airlines. Out of the 112 flights, 90 were on time, and 22 experienced delays. The proportion of delayed flights for Empire Airlines Inc. is relatively low, suggesting a better on-time performance compared to the other airlines discussed earlier.

Southwest Airlines Co. stands out with the highest delay percentage, suggesting that it faced a significant number of delays compared to the other airlines in the dataset. Delta Air Lines Inc. and SkyWest Airlines Inc. have lower delay percentages, indicating a relatively better on-time performance compared to Southwest Airlines Co. American Airlines Inc. falls in between, with a delay percentage higher than SkyWest but lower than Southwest.

```
airline_delay_count = df_pandas.groupby(
    ['Airline', 'DepDelTypes']).size().unstack(fill_value=0)

# Sort the airlines based on the total delay count in descending order
airline_delay_count['Total'] = airline_delay_count['1'] + airline_delay_count['0']
airline_delay_count = airline_delay_count.sort_values(
    by='Total', ascending=False)

# Get the top five airlines with the highest values
top_five_airlines = airline_delay_count.head(5)

# Create a grouped bar chart for the top three airlines
ax = top_five_airlines[['1', '0']].plot(
    kind='barh', stacked=False, figsize=(10, 6))
plt.xlabel('Count')
plt.ylabel('Airline')
plt.title('Top Five Airlines Delay and NoDelay Counts')
plt.legend(title='Delay Status', loc='upper right')
# plt.xticks(top_three_airlines.index, range(len(top_three_airlines)), rotation=45, ha='right')
plt.tight_layout()

plt.show()
```

Figure 31: Code of grouped bar chart

The code snippet of Figure 31 figures out the number of delayed and on-time flights for each airline, groups them by the total number of flights, picks the top five airlines, and then makes a grouped bar chart to show the number of delayed and on-time flights for these top five airlines. This chart shows how the chosen companies did in terms of delays and how often they did not have any.

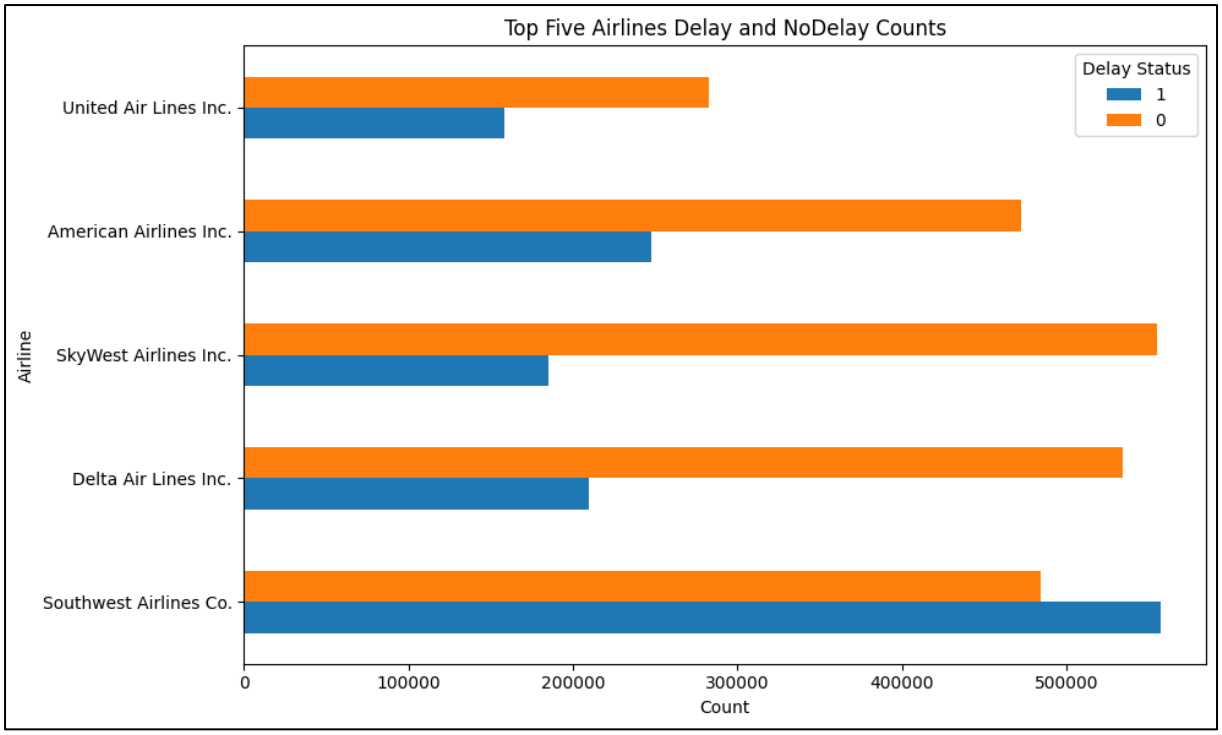


Figure 32: Horizontal Grouped Bar Chart

Based on the bar chart of Figure 32(1-delay and 0-no delay), Southwest Airlines had the most total flights as well as the most delayed flights among the top five airlines. More than half of their flights were delayed, impacting their on-time performance significantly. Delta Air Lines experienced less delays than Southwest Airlines, implying superior on-time performance. SkyWest Airlines had a lesser delay than both Southwest and Delta Airlines, suggesting that it was on time. The figure shows that the degree of delay of most flight companies in the United States accounts for about one-third of the whole flights. However, Southwest faced a big problem with delays.

Airline and Month based on Two Airline Company

```
# Filter the data to select the desired airline (e.g., 'Airline A')
selected_airline = 'Southwest Airlines Co.'
selected_airline_data = df_pandas[df_pandas['Airline'] == selected_airline]

# Group by 'Month' and 'DepDelTypes' for the selected airline, and count the occurrences
airline_month_delay_count = selected_airline_data.groupby(
    ['Month', 'DepDelTypes']).size().unstack(fill_value=0)

# Create a 'Total' column for the total count of delays and no delays for each month
# Plot the grouped bar chart for the selected airline and each month
ax = airline_month_delay_count[['1', '0']].plot(
    kind='bar', figsize=(10, 6))
plt.xlabel('Month')
plt.ylabel('Count')
plt.title(f'Delay and NoDelay Counts for {selected_airline} by Month')
plt.legend(title='Delay Status', loc='upper right')
plt.tight_layout()

plt.xticks(rotation=0)
plt.show()
```

Figure 33: Code of Specified Airline and Delay Type based on Month.

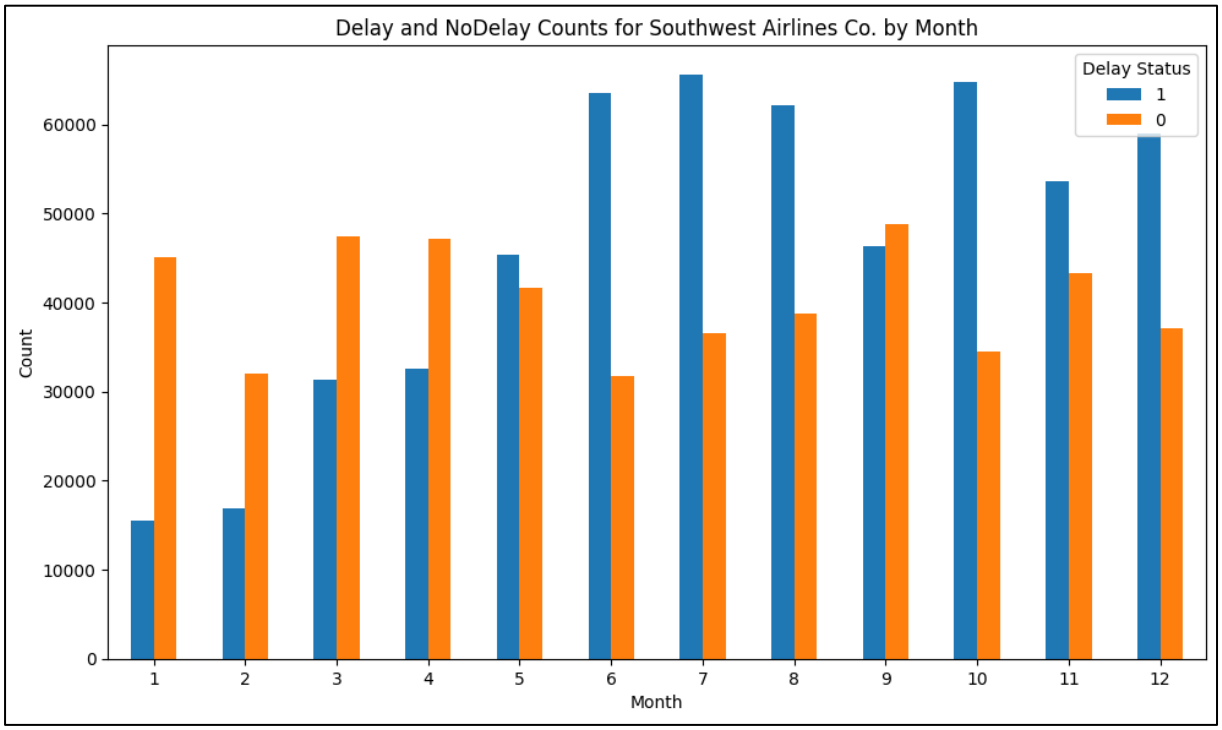


Figure 34: Grouped Bar Chart of Southwest Airline

July (7) is the month with the most delayed flights for Southwest Airline, around 65000 delayed flights. The following month is October with around 64000 flights delayed. January (1) is the month with the fewest delayed flights for Southwest Airlines. January had the lowest number of delayed flights when compared to other months, with 15487. These four months (June, July, August, and October) have the most delayed flights in comparison to other months.

These months are part of the summer season, which is a prime travel time when many individuals take vacations and travel for pleasure. The increased demand for travel during the summer months can result in increased air traffic, airport congestion, and possible delays. The chart has shown from January to October shows an upward trend in the number of delays. According to this growing trend, the number of delays has been steadily rising throughout the year.

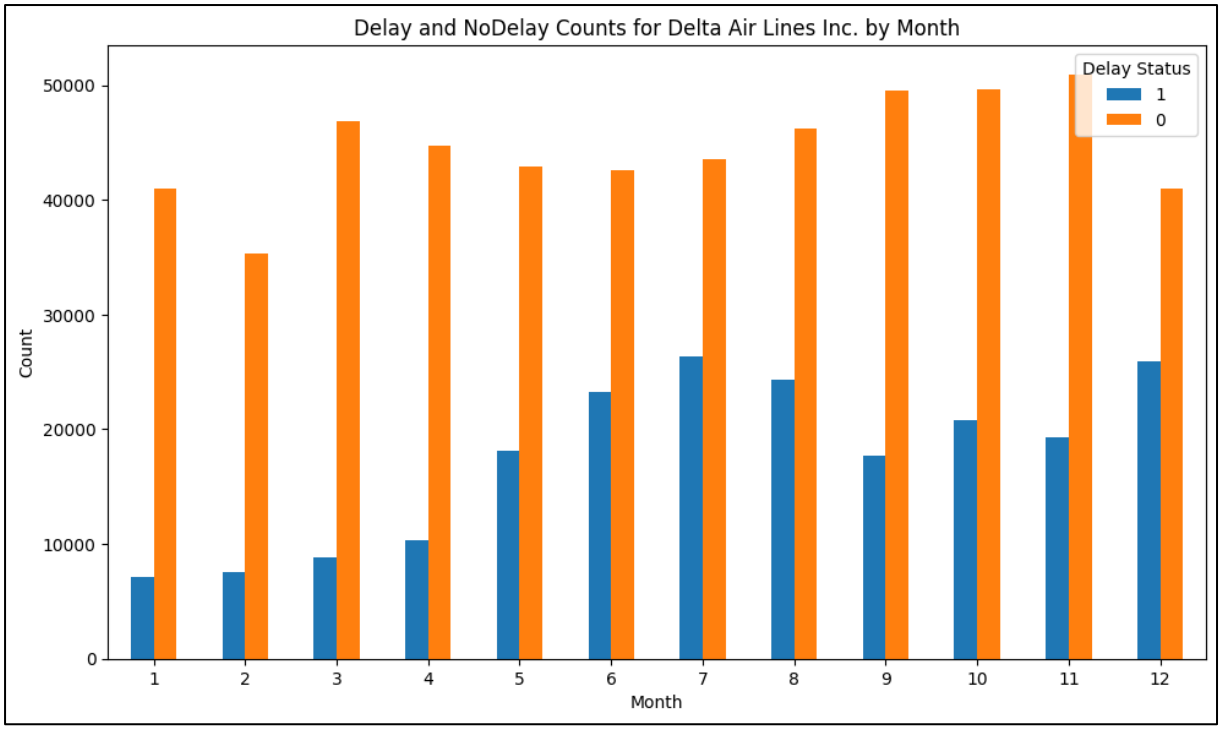


Figure 35: Grouped Bar Chart of Delta Airline

Figure 35 has shown the bar chart of Delta Airline. June and July have the most delays, with 23230 and 26379 flights, respectively. These months most likely fall around the busiest travel period of the year, which is the summer, when many people take vacations or leisure trips. With 7130 delayed flights, January is the month with the fewest delays. Since February is often a slower month for travel, there could be less delays. From January through July, there is a progressive rise in the number of delayed flights, which peaks in July. The number of delays then gradually decreases from August to December.

Justification for Airline

Southwest Airlines and Delta Air Lines exhibit a similar seasonal trend with increased delays during the summer months. The variation in the magnitude of delays may be attributed

to operational and logistical differences between the two airlines. Southwest Airlines faces challenges in managing delays, particularly during the summer months. The airline should focus on understanding the factors contributing to delays, implementing effective strategies to address them, and maintaining a commitment to ongoing improvement to ensure a positive travel experience for its passengers.

The grouped bar chart indicates that Delta Airlines may use more effective operating procedures and better resource management, which might result in fewer delays. This might include synchronized ground operations, expedited boarding procedures, and optimized scheduling. This research assumes that the capacity to manage unforeseen circumstances and reduce delays may be influenced by the knowledge and experience of the airline's workforce, including pilots, flight attendants, and ground workers. Delta Airlines's flights may have a more optimal load factor (the proportion of seats filled), enabling quicker boarding and departure times. According to the comparison, the delay is affected by the airline company. The performance of various airlines' delays might fluctuate greatly.

4.3.2.2.9 Quarter

```
quarter_delay_count = df_pandas.groupby(
    ['Quarter', 'DepDelTypes']).size().unstack(fill_value=0)

# Create a grouped bar chart for the top three airlines
ax = quarter_delay_count[['1', '0']].plot(
    kind='bar', stacked=False, figsize=(10, 6))
plt.xlabel('Quarter')
plt.ylabel('Count')
plt.title('Count of Each Delay Types with Quarter ')
plt.legend(title='Delay Status', loc='upper left')

plt.xticks(rotation=0)
plt.tight_layout()

for p in ax.patches:
    height = p.get_height()
    ax.annotate(f'{height}', (p.get_x() + p.get_width() / 2, height),
                ha='center', va='bottom', fontsize=10)
plt.show()
```

Figure 36: Code of Bar Chart by Quarter and Delay Types

The code snippet of Figure 36Figure 42 computes the number of delays for each quarter and generates a grouped bar chart to visualize the data. The result of the grouped bar chart

displays the number of Delay and No Delay flights for each quarter, making it easier to compare delay patterns between quarters.

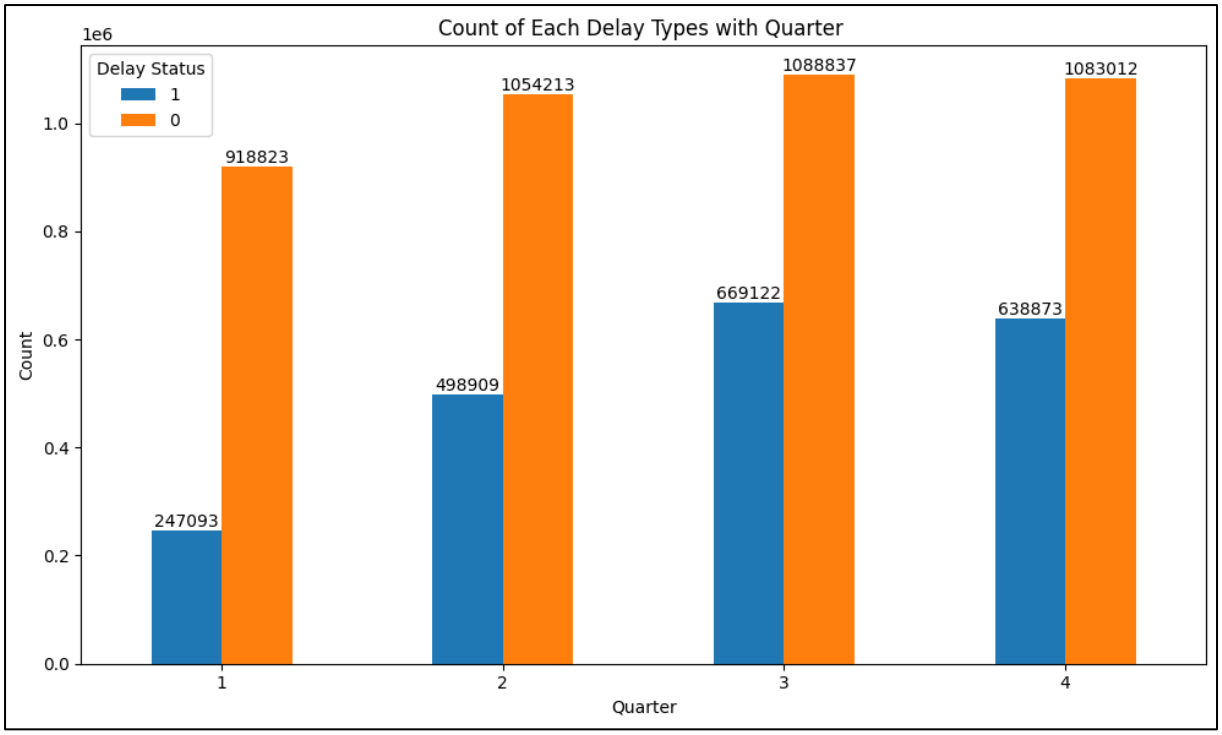


Figure 37: Grouped Bar Chart by Quarter and Delay Types

The grouped bar chart of Figure 37(1-delay and 0-no delay) demonstrates the rise in delayed flights from the first to the third quarters. The overall trend suggests an increase in delay percentages from the first quarter to the third quarter. While there is a slight decrease in the delay percentage in the fourth quarter, the values remain relatively high. This pattern indicates that delays tend to become more prevalent as the year progresses, with a potential peak in the third quarter. Quarter 3 had the most delayed flights (669122) compared to the previous quarters. Nevertheless, compared to Quarter 3, the quantity of delayed flights dropped somewhat in Quarter 4. A seasonal pattern may be seen by the increased trend in delayed flights during the third quarter and a modest decline in the fourth quarter. There might be greater air traffic or bad weather during periods of the year, which would cause longer delays. Changes in demand, the environment, or other elements might have an impact on the decline in Quarter 4. This implies that unique elements or seasonal trends throughout quarters may affect the probability of flight delays.

4.3.2.2.10 Departure Time Block

DepTimeBlk	0	1	Total
0800-0859	341149	107037	448186
0700-0759	341359	88758	430117
1100-1159	284126	131542	415668
1000-1059	290666	122879	413545
0600-0659	330660	78321	408981
1300-1359	250901	137762	388663
1400-1459	248596	139949	388545
1700-1759	235085	152854	387939
1800-1859	226911	157184	384095
1500-1559	237505	143837	381342
1200-1259	255341	125577	380918
0900-0959	276801	102859	379660
1600-1659	219637	143770	363407
1900-1959	174080	143638	317718
2000-2059	151277	111046	262323
2100-2159	101642	79789	181431
2200-2259	71957	46099	118056
0001-0559	83285	24846	108131
2300-2359	23907	16250	40157

Figure 38: Distribution of Time Block based on Delay Types

Time block “0800-0859” appears to have the highest number of delays (341149), followed closely by “0700-0759” (341359). Conversely, the time block “2300-2359” has the least delays, with only 23907 instances. This suggests that early morning hours, specifically around 8 AM, experience the most delays, while late-night hours witness the least delays. Morning time blocks may be more prone to delays due to increased air traffic, as travellers often prefer morning flights for business or convenience. On the other hand, late-night hours might experience fewer delays as air traffic tends to be lower during these periods.

4.3.2.2.11 Month

```
# Month
month_delay_count = df_pandas.groupby(
    ['Month', 'DepDelTypes']).size().reset_index(name='Count')

# Pivot the table to have airlines as rows and delay groups as columns
month_delay_pivot = month_delay_count.pivot(
    index='Month', columns='DepDelTypes', values='Count').fillna(0)

# Add a new column 'Total' containing the total count for each airline
month_delay_pivot['Total'] = month_delay_pivot.sum(axis=1)
month_delay_pivot['Delay_Percentage'] = round((month_delay_pivot['1']
                                               / month_delay_pivot['Total']) * 100, 0)
month_delay_pivot['NoDelay_Percentage'] = round((month_delay_pivot['0']
                                                 / month_delay_pivot['Total']) * 100, 0)

# Sort the airlines based on the 'Total' count in descending order
month_delay_pivot = month_delay_pivot.sort_values(
by='Total', ascending=False)


print(tabulate(month_delay_pivot, headers='keys', tablefmt='grid'))
```

Figure 39: Code of table by month and delay types

The code of Figure 39 gives detailed calculation of flight delays and no delay flights for each month, providing insight into any possible seasonal patterns or trends in flight performance. Furthermore, for each month, the delay percentage and no-delay percentages will be calculated by dividing the respective numbers by the “Total” count and multiplying by 100. Sorting the pivot table in decreasing order by “Total” count, with the months having the most total flights at the top. The tabulate function is used to structure the output in a grid-like manner, which makes the data easier to read and analyze.

Month	0	1	Total	Delay_Percentage	NoDelay_Percentage
7	339251	266135	605386	44	56
8	354068	238549	592617	40	60
10	372193	210409	582602	36	64
11	379235	193722	572957	34	66
12	331584	234742	566326	41	59
6	326808	237646	564454	42	58
9	395518	164438	559956	29	71
5	367491	150037	517528	29	71
4	359914	111226	471140	24	76
3	357595	103418	461013	22	78
1	309295	65776	375071	18	82
2	251933	77899	329832	24	76

Figure 40:Table of Month and delay types

Figure 40(1-delay and 0-no delay) displays flight performance for each month, concentrating on the number of delayed and no delay flights, total number of flights, and delay and no-delay percentages. There were 605386 flights in July with 266135 delays and 339251 no-delay flights. In July, the delay rate was about 44% which is the highest delay month. There were 592167 flights in August with 238549 delays and 354068 flights without delay. The delay rate was about 40%, while the no-delay proportion was around 60% in August. The summer months (July and August) show the highest delay percentages, aligning with increased travel and potential operational challenges. October maintains a substantial delay percentage, indicating that operational issues may persist into the fall. There were 329969 flights in February with 77899 delays and 251933 no-delay flights. The delay rate in February was roughly 24%, which is the lowest. February and January experience the least delays, possibly due to lower travel demand and more favorable weather conditions.

The overall average delay percentage of approximately 30.42% highlights the importance of considering seasonality and specific month-to-month variations in assessing delay trends. Flight volumes change from month to month, as do the percentages of delay and no delayed flights. It is critical to examine these variances to identify any seasonal patterns or trends that may affect flight delays.

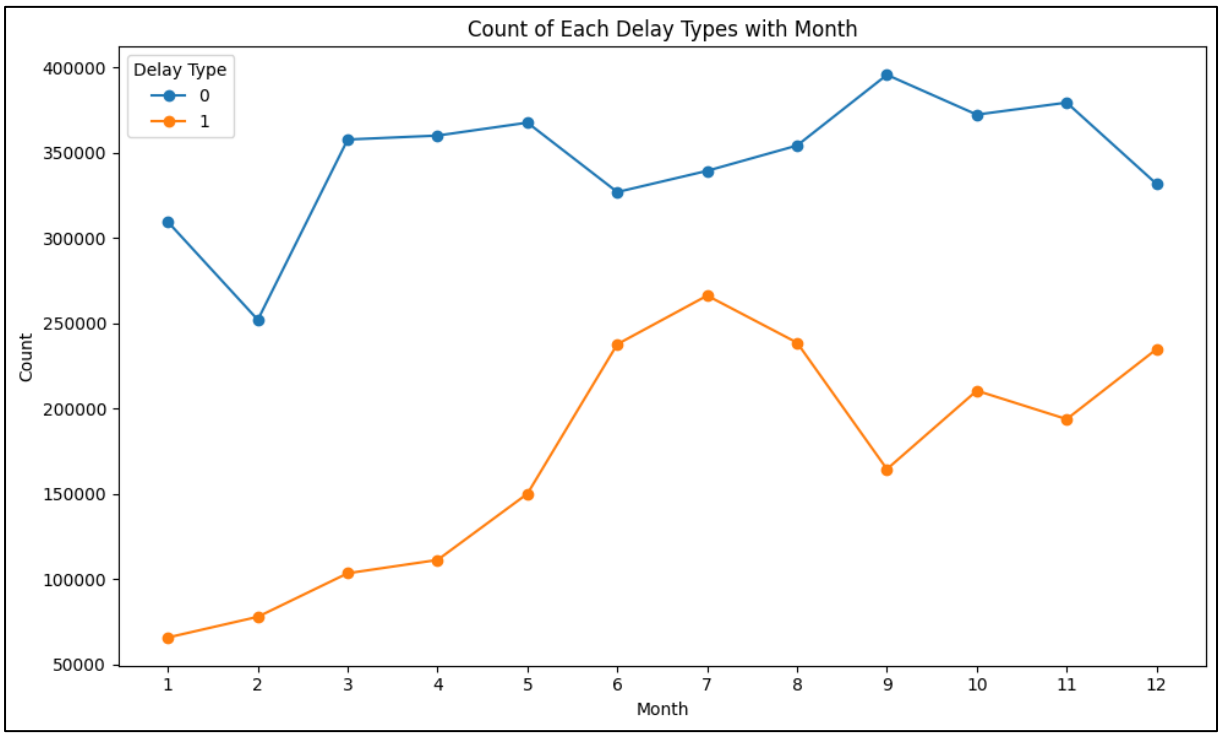


Figure 41: Line chart of delay types and month

The line chart (Figure 41) shows the trend of no-delay and delay by each month. Throughout the year, there is considerable seasonal fluctuation in flight delays. From April to July, there was a definite increased tendency in the trend of delays. When compared to the early months of the year (January to March), the number of flights is often greater in the middle of the year (June to August) and towards the end (November and December). As observed in July and August, which have more flights overall and more delays, months with more flights may encounter delays that are greater in absolute terms. According to the table 4 and line chart, the proportion of delays was greater in July and August, presumably because more people travelled during the summer vacation periods. Hence, month is an important variable that affects delay.

4.3.2.2.12 DayofMonth

```
from tabulate import tabulate
DayofMonth_delay_count = df_pandas.groupby(
    ['DayofMonth', 'DepDelTypes']).size().reset_index(name='Count')

DayofMonth_delay_pivot = DayofMonth_delay_count.pivot(
    index='DayofMonth', columns='DepDelTypes', values='Count').fillna(0)

DayofMonth_delay_pivot['Total'] = DayofMonth_delay_pivot.sum(axis=1)
DayofMonth_delay_pivot['Delay_Percentage'] = round(
    (DayofMonth_delay_pivot['1'] / DayofMonth_delay_pivot['Total']) * 100, 0)
DayofMonth_delay_pivot['NoDelay_Percentage'] = round(
    (DayofMonth_delay_pivot['0'] / DayofMonth_delay_pivot['Total']) * 100, 0)

DayofMonth_delay_pivot = DayofMonth_delay_pivot.sort_values(
    by='1', ascending=False)

print(tabulate(DayofMonth_delay_pivot, headers='keys', tablefmt='grid'))
```

Figure 42: Code of delay types and day of month

This code of Figure 42 is almost the same as Figure 39. The variable name changed in this code to create a table of count delay types based on day of month. The table has been designed based on the delay number to sort.

DayofMonth	0	1	Total	Delay_Percentage	NoDelay_Percentage
19	134151	74502	208653	36	64
18	131025	74273	205298	36	64
28	135141	73017	208158	35	65
12	135346	72962	208308	35	65
11	132304	71646	203950	35	65
27	133455	70761	204216	35	65
15	133610	70430	204040	35	65
20	134354	69279	203633	34	66
16	133190	68504	201694	34	66
14	132696	68347	201043	34	66
21	139607	67991	207598	33	67
1	130631	67810	198441	34	66
17	134712	67713	202425	33	67
3	135467	67589	203056	33	67
29	129569	67582	197151	34	66
24	136792	67225	204017	33	67
10	135074	67205	202279	33	67
26	136860	67078	203938	33	67
8	140566	66456	207022	32	68
30	123713	66383	190096	35	65
22	144268	65795	210063	31	69
13	133674	65371	199045	33	67
23	141584	65205	206789	32	68
7	139101	64911	204012	32	68

25	133280	63955	197235	32	68
2	135901	63807	199708	32	68
9	137449	63039	200488	31	69
4	142342	58991	201333	29	71
6	139792	58845	198637	30	70
5	144713	58625	203338	29	71
31	74518	38700	113218	34	66

Figure 43: Table of Day of Month and Delay Types

It is evident that certain days stand out with notable variations in on-time performance through analyzing the delay percentages across various days. Day 19 emerges as the day with the highest delay percentage, approximately 36%, where out of a total of 208653 flights, 74502 experienced delays. Following closely, Day 18 exhibits a delay percentage of around 36%, indicating a persistent trend of operational challenges. The third-highest delay percentage occurs on Day 28, with approximately 35.07% of flights experiencing delays out of a total of 208158.

On the contrary, the bottom three days with the lowest delay percentages showcase comparatively better on-time performance. Day 5 demonstrates a delay percentage of approximately 29%, while Day 6 follows closely with a delay percentage of around 29%. Day 31 stands out as the day with the lowest delay percentage among the lowest three, with approximately 34% of flights experiencing delays out of a total of 113218. These variations contribute to an overall average delay percentage of approximately 33.40% across all days, providing a baseline for assessing individual day performance.

The table (Figure 43) provides proof that the day of month may influence the proportion of flight delays. Airlines and aviation authorities should carefully monitor and analyze the trends and reasons causing delays on various days to better understand and control flight delays.

4.3.2.2.13 DayOfWeek

```
DayOfWeek_delay_count = df_pandas.groupby(
    ['DayOfWeek', 'DepDelTypes']).size().unstack(fill_value=0)

DayOfWeek_total_count = DayOfWeek_delay_count.sum(axis=1)

# Calculate the percentage of each DepDelType for each day of the week
DayOfWeek_percentage_real = DayOfWeek_delay_count.div(
    DayOfWeek_total_count, axis=0) * 100

# Create a horizontal stack bar chart
ax = DayOfWeek_percentage_real.plot(kind='barh', stacked=True, figsize=(10, 6))
plt.xlabel('Percentages')
plt.ylabel('DayOfWeek')
plt.title('DayOfWeek based on Count of Delay Types')
plt.legend(title='Delay Status', loc='upper right')
plt.tight_layout()

for patch in ax.patches:
    width, height = patch.get_width(), patch.get_height()
    x, y = patch.get_xy()
    ax.annotate(f'{width:.1f}%', (x + width / 2, y +
                height / 2), ha='center', va='center')

plt.show()
```

Figure 44: Code of horizontal bar chart based on delay types and day of week

The code of Figure 44 builds a horizontal stacked bar chart to show the percentage distribution of delay kinds (DepDelTypes) by day of the week (DayOfWeek). Some of the functions are the same as the code above, which groups the variables to count the value and the chart settings. This loop iterates across the chart's bars (ax.patches) and adds annotations to show the percentage values on the bars.

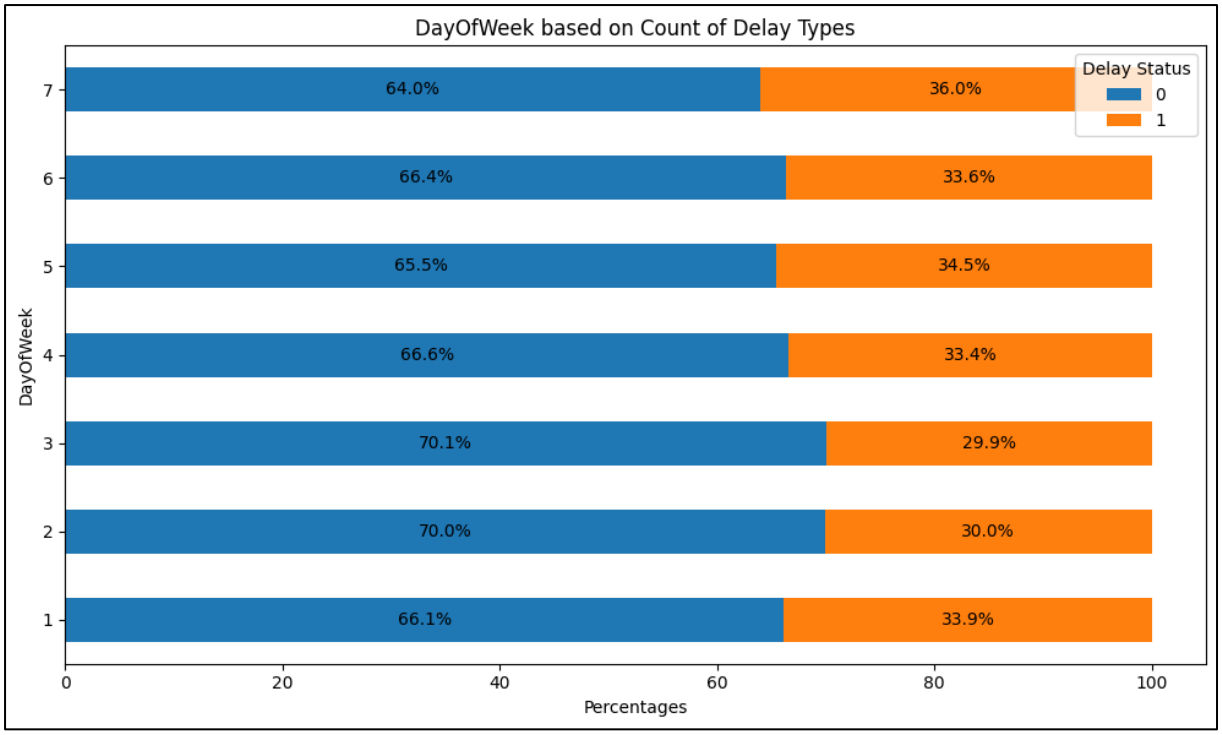


Figure 45: Horizontal Bar Chart based on delay types and day of week

The bar chart of Figure 45 shows that different days of the week have larger percentages of delays than others. Sunday has the largest proportion of delays (36%), while Tuesday and Wednesday have the lowest percentages of delays (30%). On the other hand, Sunday has the lowest no-delay proportion (64%), while Tuesday and Wednesday have the greatest no-delay percentages (70%). The days with the largest proportion of delays are typically Sunday, Tuesday, and Wednesday. Several possible variables that could cause flight delays on certain days. Business travelers often travel on Tuesdays and Wednesdays. On certain days, airlines may be extra careful about minimizing delays and business travelers may prioritize timeliness. As individuals return from their weekend getaways or holidays, particularly Sundays, tend to have greater travel volumes. Increased passenger volume may cause delays and congestion at airports.

4.3.2.2.14 Diverted

```
# Count the occurrences of delay and nondelay groups based on the diverted
Diverted_delay_count = df_pandas.groupby(
    ['Diverted', 'DepDelTypes']).size().unstack(fill_value=0)

total_diverted = Diverted_delay_count.loc[True].sum()
total_not_diverted = Diverted_delay_count.loc[False].sum()

Diverted_delay_count_percentage = Diverted_delay_count.copy()
Diverted_delay_count_percentage.loc[True] = (
    Diverted_delay_count.loc[True] / total_diverted) * 100
Diverted_delay_count_percentage.loc[False] = (
    Diverted_delay_count.loc[False] / total_not_diverted) * 100

# Create a horizontal grouped bar chart
ax = Diverted_delay_count_percentage.plot(
    kind='bar', stacked=True, figsize=(7, 6), width=0.4)
plt.xlabel('Diverted')
plt.ylabel('Percentage')
plt.title('Percentage of Delay Types Based on Diverted Status')
plt.legend(title='Delay Status', loc='upper right')
plt.tight_layout()

for patch in ax.patches:
    width, height = patch.get_width(), patch.get_height()
    x, y = patch.get_xy()
    ax.annotate(f'{height:.1f}%', (x + width / 2, y +
                height / 2), ha='center', va='center')
plt.xticks(rotation=0)
plt.show()
```

Figure 46: Code of bar chart of diverted and delay types

The code (Figure 46) provides a horizontal grouped bar chart to depict the percentage distribution of delay kinds (DepDelTypes) depending on aircraft diverted status (Diverted or Not Diverted). These lines (total_diverted) add the counts of various delay categories for the associated rows (True or False) to compute the total number of diverted and non-diverted flights.

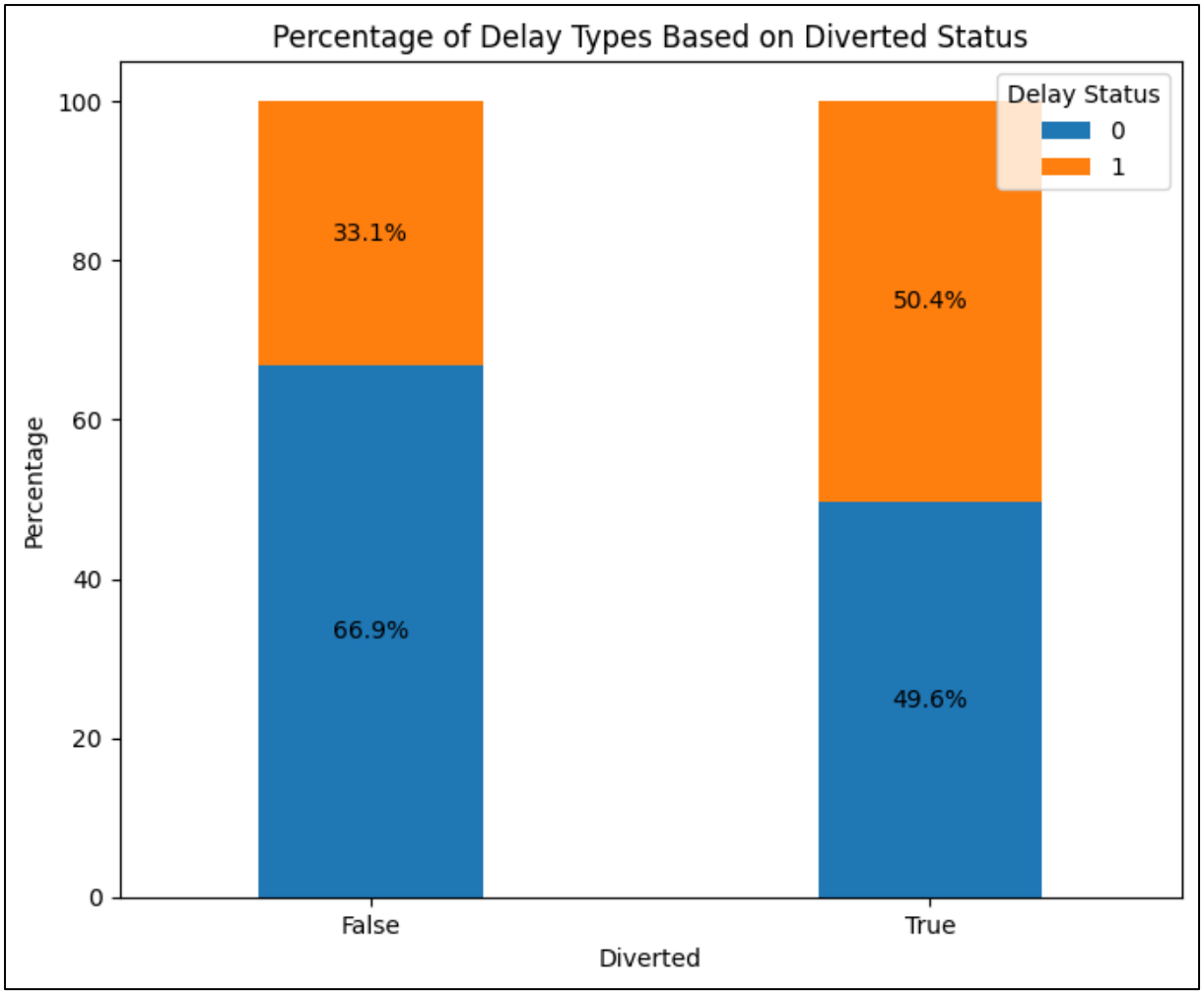


Figure 47: Stacked bar chart of diverted and delay types

Flights that were not diverted (False) encountered delays in around 33.1% of cases, while the majority (66.9%) of non-diverted aircraft experienced no delays based on figure 35. Surprisingly, a greater percentage (50.4%) of diverted (True) planes reported delays than non-diverted flights. This means that diverted planes were more likely to be delayed than non diverted flights. There is a significant variation in the proportion of delayed flights between diverted and non-diverted flights. When compared to non-diverted flights, diverted flights have a greater percentage of delays. This indicates that there is a link between diverted and delays.

4.3.2.2.15 DistanceGroup

```
DistanceGroup_delay_count = df_pandas.groupby(
    ['DistanceGroup', 'DepDelTypes']).size().reset_index(name='Count')


# Pivot the table to have airlines as rows and delay groups as columns
DistanceGroup_delay_pivot = DistanceGroup_delay_count.pivot(
    index='DistanceGroup', columns='DepDelTypes', values='Count').fillna(0)

# Add a new column 'Total' containing the total count for each airline
DistanceGroup_delay_pivot['Total'] = DistanceGroup_delay_pivot.sum(axis=1)
DistanceGroup_delay_pivot['Delay_Percentage'] = round(
    (DistanceGroup_delay_pivot['1'] / DistanceGroup_delay_pivot['Total']) * 100, 0)
DistanceGroup_delay_pivot['NoDelay_Percentage'] = round(
    (DistanceGroup_delay_pivot['0'] / DistanceGroup_delay_pivot['Total']) * 100, 0)


# Sort the airlines based on the 'Total' count in descending order
DistanceGroup_delay_pivot = DistanceGroup_delay_pivot.sort_values(
    by='1', ascending=False)

# Print the table using tabulate
print(tabulate(DistanceGroup_delay_pivot, headers='keys', tablefmt='grid',
      colalign=("right", "right", "right", "right", "center", "center")))
```

Figure 48: Code of table by distance group and delay types

The code (Figure 48) performs flight delays based on the distance grouping of the flights. The output will be a table displaying the number of delays, the number of non-delays, the total number, the delay percentages, and the non-delay percentage for each distance group. The table will be ordered in ascending order by distance group.

DistanceGroup	0	1	Total	Delay_Percentage	NoDelay_Percentage
2	1.01178e+06	426778	1.43856e+06	30	70
3	844712	392935	1.23765e+06	32	68
4	631586	368211	999797	37	63
5	428806	261670	690476	38	62
1	624527	219074	843601	26	74
7	155319	108661	263980	41	59
6	172247	101924	274171	37	63
8	69078	50923	120001	42	58
10	82230	49553	131783	38	62
11	73217	40825	114042	36	64
9	51378	33443	84821	39	61

Figure 49:Table of Distance Group and Delay Types

Table of Figure 49 has shown that eleven types of distance group. The range of distances between airports is shown by the distance group like DistanceGroup 1 represents trips up to 250 miles long, whereas DistanceGroup 2 covers airports between 251 and 500 miles long.

Beginning with Distance Group 2, which has the highest total flights at 1,438,560, a relatively low delay percentage of approximately 30% suggests a commendable on-time performance. Moving to Distance Group 3, with 1237650 total flights, the delay percentage slightly increases to around 32%, indicating a moderate impact on punctuality. However, distance group 4 experiences a more significant impact, with a delay percentage of approximately 37% out of 999797 total flights, signaling challenges in maintaining on-time performance for longer distances.

The shorter distance group 1 with 843601 total flights, demonstrates a lower delay percentage of around 26%, suggesting a more reliable on-time performance for shorter distances. The longest distances groups 10 and 11 show delay percentages of approximately 38% and 36%, respectively. While these distances still face challenges, the delay percentages are somewhat more favorable compared to certain mid-range and longer-distance groups. distance group 9, with a delay percentage of around 39%, underscores the consistent struggle to maintain on-time performance for longer flights.

Delays occurred on around 32% to 41% of the flights in this variable. The proportion of delayed flights rises gradually as travel distance increases, implying that distance between airports may be more prone to delays. Shorter distances generally exhibit better on-time performance, while longer distances tend to face higher delay percentages.

4.3.2.2.16 Taxi Out

```
# Create a taxi out (Delay) histogram
taxi_out_delay = df_pandas.query('(TaxiOut < 100) & (DepDelTypes == "1")')

plt.figure(figsize=(8, 6))
plt.hist(taxi_out_delay['TaxiOut'], bins=60, edgecolor='black')
plt.xlabel('Taxi Out (min)')
plt.ylabel('Frequency(Delay)')
plt.title('Histogram of Taxi Out')
plt.tight_layout()

plt.show()
```

Figure 50: Code of histogram by taxi out for delay

```
# Create a taxi out (No-Delay) histogram
taxi_out_delay = df_pandas.query(
    '(TaxiOut < 100) & (DepDelTypes == "0")')


plt.figure(figsize=(8, 6))
plt.hist(taxi_out_delay['TaxiOut'], bins=60, edgecolor='black')
plt.xlabel('Taxi Out (min)')
plt.ylabel('Frequency(NoDelay)')
plt.title('Histogram of Taxi Out')
plt.tight_layout()

plt.show()
```

Figure 51: Code of histogram by taxi out for no-delay

The code of Figure 50 and Figure 51 are used to generate a histogram that depicts the distribution of taxi-out times for delayed and no-delay flights. The query is used to include only delayed and no-delay flights (Table) with a taxi-out duration of less than 100 minutes. The data is separated into 60 bins to produce the histogram.

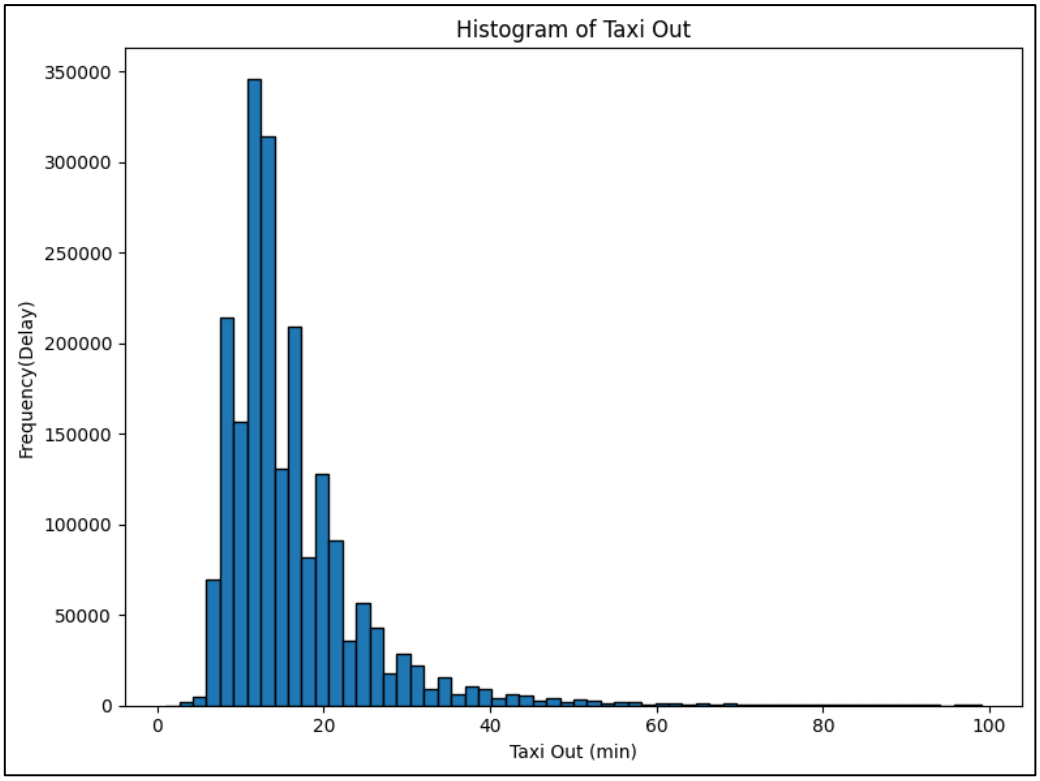


Figure 52: Histogram by taxi out for delay

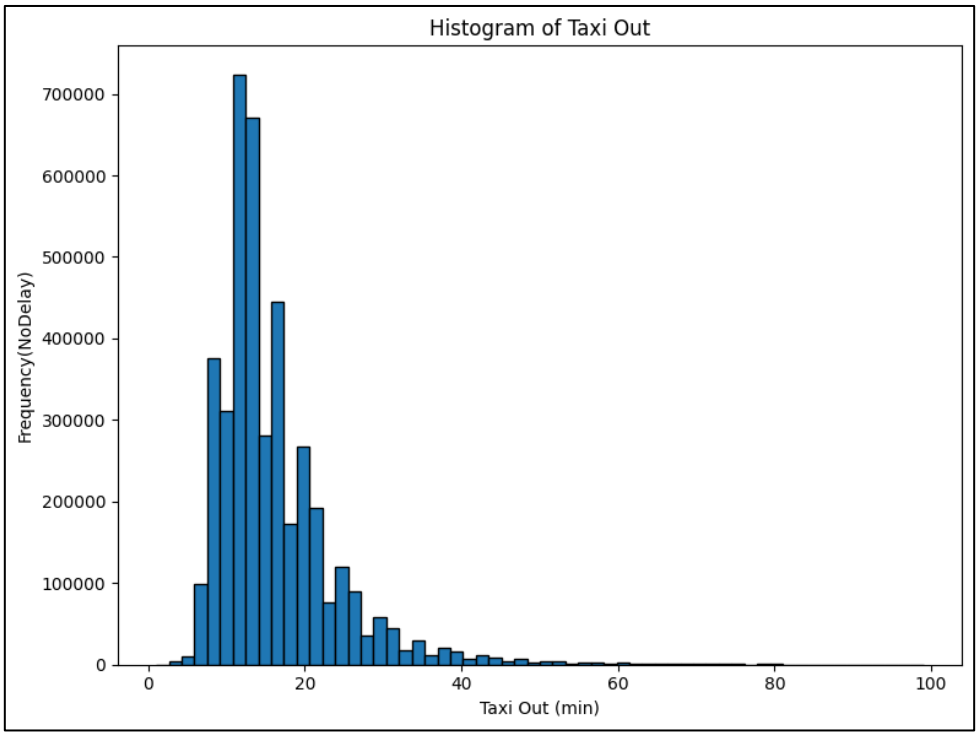


Figure 53: Histogram by taxi out for no-delay

The period between an aircraft pulling back from its gate at the departure airport and taking off and commencing flight is known as the “taxi-out”. Those histograms display right-

skewed histogram. The right-skewed histogram for both delayed and non-delayed flights shows that most flights have relatively short taxi-out times, whereas a minority number of flights have longer taxi-out durations. Both histograms indicate that taxi-out timings for both delayed and non-delayed planes are about 12 and 14 minutes. This consistency in the peaks implies that the time it takes for planes to taxi out is very constant within the stated range, regardless of whether the flight is delayed or not. The similarity of the peaks for both delay and no-delay flights shows that taxi-out time may not be a substantial contributor to flight delays in this dataset.

4.3.2.2.17 Actual Departure Time

```
# Change Datatype
# List of columns to convert to 'string'
columns_to_convert = ['DepTime']

# Convert the selected columns to 'integer' data type using pd.Categorical()
for column in columns_to_convert:
    df_pandas[column] = df_pandas[column].astype('int64')

df_pandas['DepTime'] = df_pandas['DepTime'].astype(str)

# Remove any non-digit characters (e.g., "0") from the 'DepTime' strings
df_pandas['DepTime'] = df_pandas['DepTime'].str.replace(r'\D', '', regex=True)

# Pad the 'DepTime' strings with leading zeros to ensure a 4-digit format
df_pandas['DepTime'] = df_pandas['DepTime'].str.zfill(4)
df_pandas['DepTime'] = df_pandas['DepTime'].replace('2400', '0000')
```

Figure 54:Pre-processing of DepTime

Before analysis the “DepTime”, there are some pre-processing need to be done. “DepTime” data type has been set as float. Hence, he data transformation applied to the “DepTime” column in the given code snippet seems to be aimed at converting the departure time information into a format that is more suitable for visualization. “DepTime” strings have leading zeros added to them to make sure they always have a 4-digit format. This step is probably taken to standardise departure time representation and allow proper data interpretation and display by visualisation tools. The code replaces instances of “2400” with “0000”. In the context of time, “2400” is sometimes used to represent midnight, but transforming it to “0000” ensures that the time values are consistently formatted and align with a 24-hour clock representation.

```
# Group by 'time' and 'deltype' to get the frequency of each combination
grouped_df = df_pandas.groupby(
    ['DepTime', 'DepDelTypes']).size().reset_index(name='count')

# Separate delay and no-delay data
delay_df = grouped_df[grouped_df['DepDelTypes'] == '1']
nodelay_df = grouped_df[grouped_df['DepDelTypes'] == '0']

# Set up the plot
plt.figure(figsize=(20, 6))

# Plot delay points
plt.scatter(delay_df['DepTime'], delay_df['count'],
            color='red', label='Delay', s=20)

# Plot no-delay points
plt.scatter(nodelay_df['DepTime'], nodelay_df['count'],
            color='blue', label='No-Delay', s=20)
plt.xticks(range(0, 1500, 60))
plt.yticks(range(0, max(grouped_df['count']) + 1000, 1000))
# Customize the plot
plt.xlabel('Time')
plt.ylabel('Frequency')
plt.title('Frequency of Delay Types within 1-hour Intervals')
plt.legend()

# Show the plot
plt.show()
```

Figure 55: Code of scatter plot by deptime and delay types

The code (Figure 55) is used to produce scatter plots. The Groupby() method groups objects depending on the variables used to build the chart. Each group's size (number of rows) is determined by Size(), which basically counts the number of occurrences. Time in hours is shown along the x-axis, while flight frequency is plotted along the y-axis. Each red dot represents the frequency of flights that experience delays (1) during a given period of time of one hour, while each blue dot represents the frequency of flights that experience no delays (0) over the same period.

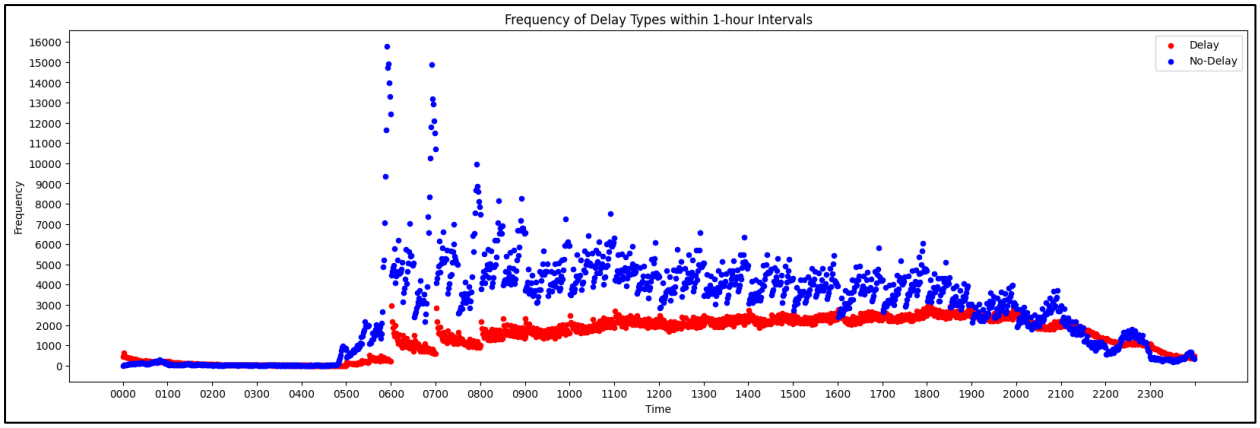


Figure 56: Scatter plot by deptime and delay types

Between the hours of 12 AM and 5 AM, both delayed and non-delayed flights are at their lowest frequency (about 4000 flights per hour). Air traffic is often at its lowest around this period. During Morning Peak (5 AM to 7 AM), both “Delay” and “No Delay” aircraft numbers start to rise considerably. The somewhat increased number of “NoDelay” flights suggests that during the morning peak, most flights are leaving on schedule. “Delay” and “No Delay” flights both increase and sustain substantially greater frequency throughout this time (7 AM to 1 PM), which displays the maximum flight density. During this busy time, there are a lot of on-time departures (“NoDelay”) and a few delays (“Delay”). While “NoDelay” flights continue to predominate with a rising frequency, the frequency of “Delay” flights has remained largely consistent at or below 3000 flights. After 6 AM, “NoDelay” flight frequency begins to decline, potentially because of increasing nighttime air traffic and other operational considerations. “Delay” flights also see a decline in frequency, although continue to operate more often than 2000 times each year. After 9 PM, the number of delayed flights begins to outnumber those without delays. This pattern persists until after midnight, suggesting that there is a larger risk of experiencing flight delays during these times as opposed to earlier in the day. Period influences the frequency of both delayed and on-time flights. There are distinct variations in flight frequency throughout the day.

4.3.2.2.18 Origin

```
# Count of City
unique_cities = set(df_pandas['OriginCityName'])
city_count = len(unique_cities)
print("Number of cities:", city_count)
```

Figure 57: Code of count the number of cities

The code snippet of Figure 57 can help to count the number of origin city. There are 374 cities. Due to too many cities, the top eleven and last two cities by total flight selected for analyze.

```
from tabulate import tabulate
city_delay_count = df_pandas.groupby(
    ['OriginCityName', 'DepDelTypes']).size().reset_index(name='Count')

# Pivot the table to have airlines as rows and delay groups as columns
city_delay_pivot = city_delay_count.pivot(
    index='OriginCityName', columns='DepDelTypes', values='Count').fillna(0)

# Add a new column 'Total' containing the total count for each city
city_delay_pivot['Total'] = city_delay_pivot.sum(axis=1)

city_delay_pivot['Delay_Percentage'] = round(
    (city_delay_pivot['1'] / city_delay_pivot['Total']) * 100, 0)
city_delay_pivot['NoDelay_Percentage'] = round(
    (city_delay_pivot['0'] / city_delay_pivot['Total']) * 100, 0)

# Sort the city based on the 'Total' count in descending order
city_delay_pivot = city_delay_pivot.sort_values(
    by='Total', ascending=False)

# Print the table using tabulate
print(tabulate(city_delay_pivot, headers='keys', tablefmt='grid'))
```

Figure 58: Code of table by cities name and delay types

The code function (Figure 58) is same as Figure 39. The variable changed to “OriginCityName”.

OriginCityName	0	1	Total	Delay_Percentage	NoDelay_Percentage
Chicago, IL	215162	127222	342384	37	63
Atlanta, GA	213464	97173	310637	31	69
Dallas/Fort Worth, TX	167787	103439	271226	38	62
Denver, CO	139254	122822	262076	47	53
Charlotte, NC	164955	58368	223323	26	74
Houston, TX	118415	78646	197061	40	60
Los Angeles, CA	109869	62263	172132	36	64
New York, NY	118179	49749	167928	30	70
Seattle, WA	109490	56106	165596	34	66
Phoenix, AZ	99053	65536	164589	40	60
Washington, DC	112384	44432	156816	28	72
Las Vegas, NV	76464	62864	139328	45	55
Orlando, FL	77855	52577	130432	40	60
Detroit, MI	96108	32903	129011	26	74
Minneapolis, MN	92056	28374	120430	24	76
Salt Lake City, UT	80210	37863	118073	32	68
Philadelphia, PA	77987	23836	101823	23	77
Newark, NJ	60436	38780	99216	39	61
San Francisco, CA	69539	28072	97611	29	71
Miami, FL	55865	34841	90706	38	62
Boston, MA	65128	24519	89647	27	73
Fort Lauderdale, FL	52109	33736	85845	39	61

Adak Island, AK	30	66	96	69	31
Gustavus, AK	67	20	87	23	77
Branson, MO	43	26	69	38	62
Bishop, CA	12	23	35	66	34
Hoolehua, HI	24	4	28	14	86
Lanai, HI	19	9	28	32	68
Pago Pago, TT	0	6	6	100	0

Figure 59: Table of City and Delay Types

There were delays on 127229 flights out of a total of 342577 departures from Chicago, representing approximately 37% of all flights based on figure 46. A total of 311238 flights departing Atlanta, 97,188 were delayed (31%), which was the second highest percentage. The

remaining 215348 flights (63%) did not experience any delays. Those cities such as Denver (47%) and Houston (40%), have relatively higher percentages of delayed aircraft than others, such as Charlotte (26%) and New York (30%). With only six flights to Lanai, HI, the percentages must be interpreted with caution. The percentages may not be representative of the overall performance due to the tiny sample size. The percentages of flight delays for the cities in the dataset range between 13% and 47%. Varied cities have varied percentages of delays, indicating that flight delay patterns can be influenced by various factors that vary by city.

4.3 Data Pre-processing

4.3.1 Feature Selection

```
necessary_variables = ['Airline', 'OriginCityName', 'Diverted', 'DistanceGroup', 'Quarter',
                       'Month', 'DayofMonth', 'DayOfWeek', 'TaxiOut', 'DepDelayMinutes', 'DepTime', 'Cancelled','DepTimeBlk']

sv = df[necessary_variables]
```

Figure 60: Code of process feature selection

```
Convert to Pandas

df_pandas = sv.compute()
```

Figure 61: Code of convert dask dataframe to pandas

Before beginning exploratory data analysis (EDA), it is important to choose the features to analyze, particularly since this dataset contains a lot of variables. To make the ensuing EDA process more controllable and targeted, feature selection (Figure 60) may assist limit the number of variables. There are 13 variables that had been selected after data understanding through the Table 1. Those variables had been assumed to affect the flight delay. After feature selection, certain data processing pipelines include a conversion from Dask to Pandas that is shown in Figure 61. Since the data is too large for Pandas to load but Dask is able to read. Dask is a library for parallel computing that allows for the decomposition of enormous datasets into manageable pieces that can then be processed in parallel. However, Pandas is a popular Python package for data manipulation that can process data in-memory and quickly for datasets that fit in RAM. Pandas is mostly because to its extensive usage and robust ecosystem. There are

various functions, charting libraries, and other data analysis tools for additional investigation and model construction after conducting feature selection in Dask on the reduced dataset.

4.3.2 Delete Rows that Flight Cancelled

```
Delete the row that flight has been cancelled

df_pandas = df_pandas.loc[df_pandas['Cancelled'] != True]
```

Figure 62: Code of delete "Cancelled" column

Figure 62 shows the code snippet of filtered the rows which flight has been cancelled as mentioned the reason above. In Pandas, *loc* technique is used for label-based indexing. This piece of code picks just the rows from the DataFrame. True value is the flight has been cancelled in this column. If the 'Cancelled' column has a True value, those rows will be filtered out. The pre-processing step of deleting rows with cancelled flights is critical to ensuring that your predictive model concentrates on the objective of effectively anticipating delays and is not influenced by the inclusion of other kinds of data.

```
df_pandas.shape
✓ 0.1s
(6200853, 12)
```

Figure 63:Output of count the rows and columns

After filtering the rows that the flight has been cancelled, there are still 6200853 observations in the data. It indicated 111018 flights have been cancelled in 2021.

```
missing_values = df_pandas.isnull().sum()

print("Missing values in each column:")
print(missing_values)
```

```
Missing values in each column:
Airline             0
OriginCityName      0
Diverted            0
DistanceGroup       0
Quarter             0
Month               0
DayofMonth          0
DayOfWeek           0
TaxiOut             0
DepDelayMinutes     0
DepTime             0
Cancelled           0
DepTimeBlk          0
DepDelTypes         0
dtype: int64
```

Figure 64: Check Missing Values (After Delete the rows that Flight Cancelled)

Figure 64 shows the result of check missing value after delete the rows that flight cancelled. This indicated most of the missing value related to the flight cancelled. The details of the flight cancelled do not record in the dataset.

4.3.3 Delete Duplicate Rows

Delete duplicate rows

```
df_pandas = df_pandas.drop_duplicates()
```

Figure 65: Code of Delete Duplicate Rows

Check Duplicate Rows (After Pre-processing)

```
# Find and count duplicate rows
duplicate_rows = df_pandas[df_pandas.duplicated()]
num_duplicates = len(duplicate_rows)

# Print the duplicate rows and the count
print(f"Number of Duplicate Rows: {num_duplicates}")
```

```
Number of Duplicate Rows: 0
```

Figure 66: Check Duplicate Rows (After Remove)

Figure 65 shows the code of delete the duplicate rows. After delete the duplicate rows, Figure 66 shows there is no more duplicate rows in this dataset.

```
df_pandas.shape
```

```
(6198882, 14)
```

Figure 67:Check the number of rows

After deleted the rows that flight cancelled and duplicate rows, there is still 6198882 rows.

4.3.4 Data Transformation

```
Create New Columns for Category Delay
```

```
df_pandas['DepDelTypes'] = None
```

```
Condition for Category
```

```
# dep delay 1=delay,0=not delay)
df_pandas['DepDelTypes'] = df_pandas['DepDelTypes'].mask(
    df_pandas['DepDelayMinutes'] <= 0, '0')
df_pandas['DepDelTypes'] = df_pandas['DepDelTypes'].mask(
    df_pandas['DepDelayMinutes'] > 0, '1')
```

Figure 68: Code of create new column and condition for category delay types.

Binding is the process of changing a numerical variable into a categorical variable depending on predetermined criteria. The numerical variable that indicates delays into two groups: 1- delay and 0-Not delayed When doing some analysis or modelling jobs that call for working with categorical data or when you wish to investigate how delays affect various parts of flight operations, this transformation might be helpful. The code of Figure 68 is set a two condition for the departure delay minutes to category which delay minute <=0 is 0-not delay and >0 is 1-delay.

Label Encoder

```
Label Encoder

from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
df_pandas['Airline'] = label_encoder.fit_transform(df_pandas['Airline'])
df_pandas['Origin'] = label_encoder.fit_transform(df_pandas['Origin'])
df_pandas['DepTimeBlk'] = label_encoder.fit_transform(df_pandas['DepTimeBlk'])
```

Figure 69: Code of Label Encoder

As mentioned above, label encoding is one of the pre-processing techniques used in this research to convert categorical data, represented as labels or strings, into numerical values. It is particularly necessary in situations where machine learning algorithms require numerical input, as many algorithms, such as those in XGBoost and LightGBM are designed to work with numerical data (Saha, 2023).

Change Data Type

```
Change Data Type

df_pandas['DepTime'] = df_pandas['DepTime'].astype(int)
df_pandas['DepDelTypes'] = df_pandas['DepDelTypes'].astype(int)
```

Figure 70: Code of change data type for the specified column

As mentioned at EDA, “DepTime” has been converted to a standard 4-digit format free of non-digit characters by using the code snippet of Figure 54 after remove missing value. Data cleaning and type conversion procedures guarantee that the “DepTime” column always contains only uniformly formatted strings. The primary purpose of these data type conversions is to prepare the data for machine learning tasks, specifically for models like XGBoost and

LightGBM. The conversion ensures that both the feature (“DepTime”) and target (“DepDelTypes”) columns are in a format compatible with these machine learning models.

4.3.5 Delete Column

Delete Column Cancelled and DepDelayMinutes

```
columns_to_drop = ['Cancelled','DepDelayMinutes']
df_pandas.drop(columns=columns_to_drop, inplace=True)
```

Figure 71: Code of drop “Cancelled” and “DepDelayMinutes” column

The “Cancelled” column had been removed after the rows with cancelled flights were sorted out. Therefore, cancelled flights are fundamentally different from delayed flights, incorporating them in the prediction job may introduce biases or false patterns into your model. A more targeted and relevant data set for predicting delays is produced by deleting the cancelled flights. “DepDelayMinutes” also has been removed from the data since this research utilized it to create the target variables.

```
df_pandas.head()
```

✓ 0.0s Python

riginCityName	Diverted	DistanceGroup	Quarter	Month	DayofMonth	DayOfWeek	TaxiOut	DepDelayMinutes	DepTime	DepDelTypes
St. George, UT	False	2	1	3	3	3	10.0	0.0	0714	NoDelay
Phoenix, AZ	False	2	1	3	3	3	23.0	0.0	0917	NoDelay
Manchester, NH	False	4	1	3	3	3	15.0	0.0	1321	NoDelay
Dallas/Fort	False	4	1	3	3	3	27.0	0.0	1636	NoDelay

Figure 72: Code of create new column and condition for category delay types

After deleting those columns, the code snippet of Figure 72 allows to check the columns available or not. Figure 72 has indicated that the columns “Cancelled” and “DepDelayMinutes” has been deleted.

```
df_pandas.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6198832 entries, 0 to 183987
Data columns (total 12 columns):
 #   Column          Dtype
---  ------          -----
 0   Airline         int64
 1   OriginCityName  int64
 2   Diverted        bool
 3   DistanceGroup   int64
 4   Quarter         int64
 5   Month           int64
 6   DayofMonth      int64
 7   DayOfWeek       int64
 8   TaxiOut         float64
 9   DepTime         int64
 10  DepTimeBlk      int64
 11  DepDelTypes     int64
dtypes: bool(1), float64(1), int64(10)
memory usage: 573.4 MB
```

Figure 73: Final Data types

The output of Figure 73 shows the data types of each column in your DataFrame after performing data transformations. Columns with data type “int64” include columns like “Airline”, “OriginCityName”, “DistanceGroup”, “Quarter”, “Month”, “DayofMonth”, “DayOfWeek”, “DepTimeBlk”, and “DepDelTypes”. The “Diverted” column has a data type of bool. The “TaxiOut” column has a data type of float64. The final data types indicated that the majority of columns have been transformed into numerical formats to make them suitable for machine learning tasks.

4.3.6 Split dataset into train set and test set

Splitting Dataset into Training and Testing Sets

```
] from sklearn.model_selection import train_test_split
  X = df_pandas.drop('DepDelTypes', axis=1)  # Features
  y = df_pandas['DepDelTypes']  # Target variable

] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
```

Figure 74: Code of Split dataset

The train test split function that allow users split a dataset into training and testing sets is loaded by the code from the sklearn.model selection package. X containing the features (independent variables) for the machine learning model. "DepDelTypes" is related to column y, which means that this column will be the one that the model attempts to predict. “Train-Test Split” splits the features (X) and target variable (y) into training and testing sets. It has been decided that 0.2 of the data will be used for testing, while 80% will be used for training.

4.3.7 Undersampling train set

Undersampling train set

```
from imblearn.under_sampling import RandomUnderSampler

# Create an instance of RandomUnderSampler
undersampler = RandomUnderSampler(sampling_strategy=1)

# Apply random undersampling to the dataset
X_train, y_train = undersampler.fit_resample(X_train,y_train)
```

Figure 75: Code of Under sampling

Figure 75 utilizes the RandomUnderSampler from the imbalanced-learn library to address class imbalance in the dataset. Under sampling is applied to the training. The number of samples in the majority class is reduced to match the number of samples in the minority class. This step is particularly important when dealing with imbalanced datasets where one class dominates the other. In these situations, models could develop biases in favour of the majority class, which would result in worse performance on the minority class. Improved model accuracy is often the result of balancing the class distribution, particularly when there is a notable class imbalance in the dataset. The reason that does not under sampling test set is to retain the original information for prediction.

```
print("Class distribution after undersampling:", Counter(y_train))
```

```
Class distribution after undersampling: Counter({0: 1642355, 1: 1642355})
```

Figure 76: Check the distribution of Delay Types

After finishing the under sampling, Figure 76 shows the result of the target variables. Each class distribution of delay and no delay become 1642355 cases.

4.4 Model Building

4.4.1 Random Forest

Random Forest

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
rf_classifier = RandomForestClassifier(n_estimators=15, random_state=42,max_depth=30)
rf_classifier.fit(X_train, y_train)
y_pred = rf_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred, digits=4)

print(f'Accuracy of Random Forest: {accuracy}')
print(f'Confusion Matrix:\n{confusion}')
print(f'Classification Report:\n{report}')
```

Figure 77: Code of build Random Forest (Base Model)

Figure 77 shows the code that imported Random Forest classifier using scikit-learn and then evaluating its performance on a test set. The random forest of this research has been set into specified parameter by setting the “n_estimators” and “max_depth”. The Random Forest classifier is trained on the balanced training set, leveraging an ensemble of decision trees to learn patterns in the data. The model's ability to generalise to new, unknown data is assessed using measures such as accuracy, confusion matrix, and classification report. These metrics provide details on how well the model performs across various classes and how well it can categorise cases. The output will be discussed and compared with other machine learning algorithms in chapter 5.

```
from sklearn.model_selection import GridSearchCV, train_test_split

# Create a Random Forest classifier
rf_classifier = RandomForestClassifier()

# Define the hyperparameter grid
param_grid = {
  'n_estimators': [15,20,30],
  'max_depth':[30,40]
}

# Create GridSearchCV object
grid_search = GridSearchCV(estimator=rf_classifier, param_grid=param_grid, cv=3, scoring='accuracy', n_jobs=-1)

# Fit the model to the training data
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print("Best Hyperparameters:", grid_search.best_params_)

# Get the best model
best_rf_model = grid_search.best_estimator_

# Make predictions on the test set
y_pred = best_rf_model.predict(X_test)

# Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

Figure 78:Code of Hyperparameter Tuning (Random Forest)

The code of Figure 78 is using scikit-learn's GridSearchCV to perform hyperparameter tuning for a Random Forest classifier. GridSearchCV (Grid Search Cross-Validation) is a technique in machine learning used for hyperparameter tuning, which involves systematically searching through a predefined hyperparameter space to find the optimal set of hyperparameters for a given model. According to Brownlee(2020), n_estimators is the important parameter for random forest. The max_depth parameter is also crucial in a Random Forest because it controls the maximum depth of the individual decision trees in the ensemble. Hence, two of the parameters has been set 3 numbers and 2 numbers for get the best parameter of random forest. The number of folds in the training data has been divided into three, and the model has been trained and validated on distinct subsets of data to guarantee a critical analysis. The code of last 5 line will print the best hyperparameters based on the accuracy that found during the search.

```
Best Hyperparameters: {'max_depth': 40, 'n_estimators': 30}
Accuracy: 0.8256849875823441
```

Figure 79: Output of Hyperparameter Tuning (Random Forest)

Figure 79 shows the results of best hyperparameter for random forest. “Max_depth” need to be set as 40 and “n_estimators” need to be set as 30.

```
rf_tuning = RandomForestClassifier(n_estimators=30, random_state=42,max_depth=40)
rf_tuning .fit(X_train, y_train)
y_pred = rf_tuning .predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred, digits=4)

print(f'Accuracy of Random Forest: {accuracy}')
print(f'Confusion Matrix:\n{confusion}')
print(f'Classification Report:\n{report}')
```

Figure 80:Code of Random Forest (Best Parameters)

The Figure 80 is utilized to build a new Random Forest classifier model named as “rf_tuning” by using specific hyperparameters obtained from a grid search. The output will be discussed and the difference with base model in chapter 5.

4.4.2 Decision Tree

```
from sklearn.tree import DecisionTreeClassifier

# Create a Decision Tree classifier
decision_tree_classifier = DecisionTreeClassifier()

# Train the classifier on the training data
decision_tree_classifier.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = decision_tree_classifier.predict(X_test)

# Calculate and print the accuracy and classification report of the classifier
accuracy = accuracy_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred, digits=4)

print("Accuracy of Decision Tree Classifier:", accuracy)
print("Confusion Matrix:\n", confusion)
print("Classification Report:\n", report)
```

Figure 81: Code of Decision Tree (Base Model)

The code of Figure 81 is utilized for build Decision Tree classifier using scikit-learn and then evaluating its performance on a test set. The parameter of this Decision Tree model has been set as the default provided by the library. The code will print the evaluation matrix, confusion matrix and report of the Decision Tree. The code of metrics has been imported at

Figure 77. The output will be discussed in chapter 5 and compared with other machine learning algorithms.

```
train_score,test_score=list(),list()
values=[i for i in range(1,31)]
for i in values:
  model=DecisionTreeClassifier(max_depth=i)
  model.fit(X_train,y_train)

  train_yhat=model.predict(X_train)
  accuracy = accuracy_score(y_train, train_yhat)
  train_score.append(accuracy)

  test_yhat = model.predict(X_test)
  test_accuracy = accuracy_score(y_test, test_yhat)
  test_score.append(test_accuracy)

plt.plot(values,train_score,'-o',label='Train')
plt.plot(values,test_score,'-o',label='Test')
plt.xlabel('Accuracy')
plt.ylabel('Max_Depth')
plt.legend()
plt.show()
```

Figure 82: Code of Hyperparameter and Visualization (Max_Depth of Decision Tree)

The code of Figure 82 is utilized to create a plot to visualize how the accuracy of a Decision Tree classifier changes with different values of the max_depth hyperparameter. The total complexity of a decision tree is managed by the hyperparameter max depth. A trade-off among an over-fitted and under-fitted decision tree may be achieved using this hyperparameter (Importance of decision tree hyperparameters on generalization — Scikit-learn course, n.d.). The purpose of this code is to explore the impact of varying the maximum depth of the decision tree on both the training and test accuracy. The range of values has been set from 1 to 30. According to this, the values of max_depth can assume some value at the hyperparameter tuning of decision tree to find the best hyperparameters.

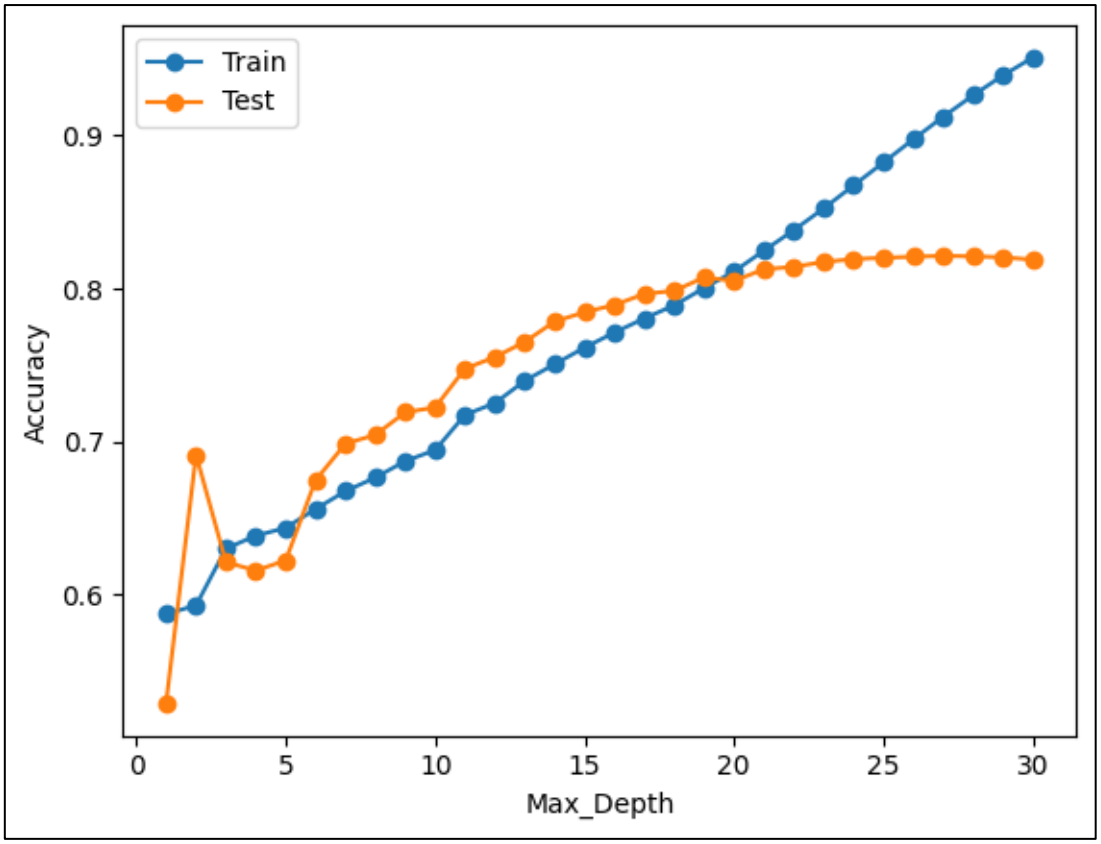


Figure 83: Output of Discover Best Max_depth of decision tree

Based on the Figure 83, both training and test accuracy initially increase with max_depth, reaching a peak around 20-30. After this point, training accuracy continues to rise slightly, but test accuracy begins to decline. The gap between training and test accuracy widens as max_depth increases beyond 30. This indicates that the model is becoming more complex and capturing more patterns in the data, leading to better performance. However, the model is starting to learn too much about the specific training data, including noise and irrelevant patterns. This results in reduced generalization to unseen data, reflected in the declining test accuracy after max_depth > 25. The selected value for set max_depth parameters around 17-22. This seems to be the optimal range for balancing model complexity with generalization performance.

Hyperparameter Tuning

```
from sklearn.tree import DecisionTreeClassifier

# Create a Decision Tree classifier
decision_tree_classifier = DecisionTreeClassifier()
# Define hyperparameters and their possible values for tuning
param_grid = {
    'max_depth': [19, 20, 22],
    'min_samples_leaf':[5,15,20],
    'max_features': ['sqrt', 'log2', None]
}

# Create GridSearchCV
grid_search = GridSearchCV(estimator=decision_tree_classifier, param_grid=param_grid, cv=3, scoring='accuracy', n_jobs=-1)

# Fit the model to find the best hyperparameters
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print("Best Hyperparameters:", grid_search.best_params_)

# Use the best model to make predictions on the test set
y_pred = grid_search.best_estimator_.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

Figure 84: Code of Hyperparameter Tuning (Decision Tree)

Figure 84 shows the code of hyperparameter tuning for a Decision Tree classifier using GridSearchCV. “Max depth”, “min samples leaf” and “max features” are the three parameters that have been specified (Importance of decision tree hyperparameters on generalization — Scikit-learn course, n.d.). “Max_depth” has been mentioned above is one of the important parameters for decision tree. Another crucial parameter is “min samples leaf” which lists potential values for the minimal quantity of samples needed at a leaf node. When splitting features, the variable “max features” indicates the range of values that may be used. It will print the set of hyperparameters that produced the optimum grid search performance, as the random forest hyperparameter tuning.

```
Best Hyperparameters: {'max_depth': 22, 'max_features': None, 'min_samples_leaf': 5}
Accuracy: 0.8149013895240838
```

Figure 85: Best Hyperparameters of Decision Tree

Figure 85 has shown that the best values for each parameter. Those values will apply for build new decision tree model.

```
from sklearn.tree import DecisionTreeClassifier
# Create a Decision Tree classifier
decision_tree_tuning = DecisionTreeClassifier(max_depth=22, max_features=None, min_samples_leaf=5)

# Train the classifier on the training data
decision_tree_tuning.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = decision_tree_tuning.predict(X_test)

# Calculate and print the accuracy and classification report of the classifier
accuracy = accuracy_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred, digits=4)

print("Accuracy of Decision Tree Classifier:", accuracy)
print("Confusion Matrix:\n", confusion)
print("Classification Report:\n", report)
```

Figure 86: Code of Decision Tree (Best Parameters)

Figure 86 is utilized to build a new Decision Tree classifier named as "decision_tree_tuning" using specific hyperparameters obtained from a grid search. The output will be discussed and the difference with base model in chapter 5.

4.4.3 XGBoost

XGBoost

```
import xgboost as xgb

xgb_classifier = xgb.XGBClassifier()

# Train the classifier on the training data
xgb_classifier.fit(X_train, y_train)

# Make predictions on the test data
y_pred = xgb_classifier.predict(X_test)

# Evaluate the classifier's accuracy
accuracy = accuracy_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred, digits=4)

print("Accuracy of XGBoost:", accuracy)
print("Confusion Matrix:\n", confusion)
print("Classification Report:\n", report)
```

Figure 87: Code of XGBoost (Base Model)

The code of Figure 87 is utilized for build XGBoost model using scikit-learn and then evaluating its performance on a test set. The parameter of this XGBoost model has been set as the default provided by the library. The code will print the evaluation metrics, confusion matrix and report of the model. The code of metrics has been imported at Figure 77. The output of XGBoost will be discussed in chapter 5 and compared with other machine learning algorithms.

Hyperparameter Tuning

```
# Define the XGBClassifier
xgb_classifier = xgb.XGBClassifier()

# Define the parameter grid for GridSearchCV
param_grid = {
    'learning_rate': [0.160, 0.076],
    'n_estimators': [200,380],
    'max_depth': [9],
    'subsample':[1,0.641],
    'gamma':[0.32,0.779],
    'min_child_weight': [0, 1]
}

# Define the scoring metric
scoring = {'Accuracy': make_scorer(accuracy_score)}

# Create the GridSearchCV object
grid_search = GridSearchCV(estimator=xgb_classifier, param_grid=param_grid, scoring=scoring, refit='Accuracy',
                           cv=3, verbose=1, n_jobs=-1)

# Fit the GridSearchCV object to the training data
grid_search.fit(X_train, y_train)

# Get the best parameters and model
best_params = grid_search.best_params_
best_xgb_classifier = grid_search.best_estimator_

# Make predictions on the test data using the best model
y_pred = best_xgb_classifier.predict(X_test)

# Evaluate the classifier's accuracy, confusion matrix, and classification report
accuracy = accuracy_score(y_test, y_pred)
```

Figure 88: Hyperparameter Tuning of XGBoost

Table 2. Hyper-parameters

Algorithm	Hyper-parameter	Range	Best value found	Best value found with SMOTE
XGBoost	Learning rate	[0.01, 1]	0.160	0.076
	Number of estimators	[100, 1000]	740	380
	Max depth	[3, 10]	9	9
	Subsample	[0, 1]	1	0.641
	Gamma	[0, 5]	0.32	0.779
	Minimum child weight	[0, 20]	0	1

Figure 89:Hyperparameters of XGBoost (Hatıpoğlu et al., 2022)

Figure 88 shows the code of hyperparameter tuning for XGBoost model using GridSearchCV. According to Figure 89, they had shown that the important parameters and the values. Therefore, this research references their research provided values to the hyperparameter tuning. There are five parameters has been selected. The step size at each iteration as one approaches the loss function's minimum is represented by the variable "learning rate". The number of boosting rounds (trees) that must be performed is indicated by "n estimators". "subsample" represents the fraction of samples used for fitting the individual trees. "Gamma" is the minimum loss reduction required to make a further partition on a leaf node. "min_child_weight" represents the minimum sum of instance weight (hessian) needed in a child. This regularisation parameter may aid in avoiding overfitting. Those parameters has been provided two different values from the previous research expect "max_depth".

```
Fitting 3 folds for each of 32 candidates, totalling 96 fits
Best Hyperparameters: {'gamma': 0.779, 'learning_rate': 0.16, 'max_depth': 9, 'min_child_weight': 0, 'n_estimators': 380, 'subsample': 0.641}
Accuracy of XGBoost Classifier: 0.8241103037078442
```

Figure 90: Best Hyperparameters of XGBoost

Figure 90 has shown that the best values for each parameter of XGBoost. Those values will apply for build new XGBoost model.

Best Parameter

```
import xgboost as xgb
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
xgb_classifier_tuning = xgb.XGBClassifier(gamma=0.779, learning_rate=0.16, max_depth=9, min_child_weight=0, n_estimators= 380, subsample=0.641)

# Train the classifier on the training data
xgb_classifier_tuning.fit(X_train, y_train)

# Make predictions on the test data
y_pred = xgb_classifier_tuning.predict(X_test)

# Evaluate the classifier's accuracy
accuracy = accuracy_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred, digits=4)

print("Accuracy of XGBoost:", accuracy)
print("Confusion Matrix:\n", confusion)
print("Classification Report:\n", report)
```

Figure 91:Code of XGBoost(Best Parameter)

Figure 91 is utilized to build a new XGBoost model named as "xgb_classifier_tuning" using specific hyperparameters obtained from a grid search. The output will be discussed and the difference with base model in chapter 5.

4.4.4 LightGBM

```
import lightgbm as lgb
from sklearn import metrics

# Create and train the LightGBM classifier
lgb_classifier = lgb.LGBMClassifier()
lgb_classifier.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = lgb_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred, digits=4)

print("Accuracy of LightGBM:", accuracy)
print("Confusion Matrix:\n", confusion)
print("Classification Report:\n", report)
```

Figure 92: Code of LightGBM (Base Model)

The code of Figure 92 is utilized for build LightGBM model using scikit-learn and then evaluating its performance on a test set. The parameter of this LightGBM model has been set as the default provided by the library. The code will print the evaluation metrics, confusion matrix and report of the model. The code of metrics has been imported at Figure 77. The output of LightGBM will be discussed in chapter 5 and compared with other machine learning algorithms.

Hyperparameter Tuning

```
lgb_classifier = lgb.LGBMClassifier()
# Define the hyperparameter grid
param_grid = {
    'num_leaves': [35,40],
    'max_bin': [400,450],           # Set the largest value for max_bin
    'learning_rate': [0.01,0.1],    # Set the lowest value for learning_rate
    'num_iterations': [200,300],
}

# Create the grid search
grid_search = GridSearchCV(estimator=lgb_classifier, param_grid=param_grid, cv=3, scoring='accuracy')

# Fit the grid search to the data
grid_search.fit(X_train, y_train)
```

Figure 93: Hyperparameter Tuning of LightGBM

Figure 93 shows the code of hyperparameter tuning for LightGBM model using GridSearchCV. According to Parameters Tuning — LightGBM 4.2.0.99 documentation (n.d.), there are four parameters can be used for increase the accuracy which are “num_leaves”, “max_bin”, “learning_rate” and “num_iterations”. The document has suggest large value for “num_leaves”, “max_bin” and num_iterations. “learning_rate” has to set small values. So, each parameter has been assigned two different values.

```
Best Hyperparameters: {'learning_rate': 0.1, 'max_bin': 450, 'num_iterations': 300, 'num_leaves': 40}
Accuracy: 0.8103643668528038
```

Figure 94: Best Parameters of LightGBM

Figure 94 has shown the output of the best values for each parameter of LightGBM. Those values will apply for build new LightGBM model.

```
# Create and train the LightGBM classifier
lgb_tuning = lgb.LGBMClassifier(learning_rate= 0.1, max_bin=450, num_iterations=300, num_leaves=40)
lgb_tuning.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = lgb_tuning.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred, digits=4)

print("Accuracy of LightGBM:", accuracy)
print("Confusion Matrix:\n", confusion)
print("Classification Report:\n", report)
```

Figure 95: Code of LightGBM(Best Parameters)

Figure 95 is utilized to build a new LightGBM model named as “lgb_tuning” using specific hyperparameters obtained from a grid search. The output will be discussed and the difference with base model in chapter 5.

4.5 Summary

Chapter 4 outlines the design and implementation of the research, with a primary focus on data collection, understanding, and pre-processing for subsequent model building. The dataset was sourced from Kaggle included extensive flight information, encompassing cancellation and delay records specific to airlines. Data understanding has been covered by explanation of variables, library importation and Exploratory Data Analysis (EDA). Notable steps in EDA involve understanding variable types, checking cancellations, handling missing values, detecting duplicates, presenting relevant statistics and the relationship between variables and target variable. Data pre-processing section involves feature selection, deletion of canceled flights and duplicates, label encoding, and creating a target variable (“DepDelTypes”) based on departure delay minutes. The dataset is then transformed, unnecessary columns are deleted, and a split into training and testing sets occurs. Under sampling is applied to the training set to address class imbalance. The chapter finishes with building the model, showing the implementations of the Random Forest, Decision Tree, XGBoost and LightGBM algorithms, establishing the framework for later assessment and analysis in following chapters.

CHAPTER 5: RESULT AND DISCUSSION

5.1 Introduction

In Chapter 5, the focus shifts to the presentation and discussion of the results obtained from the implemented models. This chapter explores the model evaluations and discussions for each algorithm employed which are Random Forest, Decision Tree, XGBoost, and LightGBM. The chapter provides a full knowledge of each model's efficacy in predicting flight delays by thoroughly analysing the performance metrics and insights obtained from each. Furthermore, a detailed comparison and discussion of the models are presented to find the best models for flight prediction. The significance of features in the predictive process is also examined, particularly with insights from XGBoost and LightGBM algorithms. At the end of the chapter, the realistic side of deploying the best machine learning model is implementation.

5.2 Model Evaluations and Discussions

5.2.1 Random Forest

```
Accuracy of Random Forest: 0.8130285771439311
Confusion Matrix:
[[691374 136752]
 [ 95049 316592]]
Classification Report:
              precision    recall  f1-score   support

           0     0.8791    0.8349    0.8564    828126
           1     0.6983    0.7691    0.7320    411641

    accuracy                         0.8130   1239767
   macro avg     0.7887    0.8020    0.7942   1239767
weighted avg     0.8191    0.8130    0.8151   1239767
```

Figure 96: Result of Random Forest Base Model

The performance evaluation of the random forest base model for flight delay prediction indicates an overall accuracy of approximately 81.3%. The confusion matrix reveals that the model correctly identified 691374 instances of no delay (class 0) and 316592 instances of delay (class 1). However, it also made 136752 false predictions of delay and 95049 false predictions of no delay.

The precision for class 0 (no delay) is 87.91%, indicating that among the instances predicted as no delay, 87.91% were truly no delays. The recall for class 0 is 83.49%, denoting

that the model captured 83.49% of the actual no-delay instances. For class 1 (delay), the precision is 69.83%, suggesting that 69.83% of the predicted delay instances were accurate, and the recall is 76.91%, indicating that the model identified 76.91% of the actual delay instances.

Learning Curve

The learning curve is a valuable tool for assessing the performance and generalization ability of a machine learning model across different sizes of training datasets after built the machine learning model. The learning curve provides insights into how well the machine learnings fits the dataset across varying sizes of training data. By observing the change in accuracy on both the training and validation sets, one can understand whether the model is overfitting, underfitting, or achieving an appropriate fit. Besides that, it can guide the optimization of model parameters. For example, if the model shows signs of overfitting, reducing the complexity.

```
from sklearn.model_selection import learning_curve
import numpy as np

rf_classifier = RandomForestClassifier(n_estimators=15, random_state=42,max_depth=30)
# Generate a learning curve based on log loss
train_sizes, train_scores, test_scores = learning_curve(rf_classifier, X_train, y_train, cv=5, scoring='accuracy', n_jobs=-1)

# Calculate mean and standard deviation for training set scores
train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)

# Calculate mean and standard deviation for test set scores
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

# Plot the learning curve based on log loss
plt.figure(figsize=(10, 6))
plt.plot(train_sizes, train_mean, color='blue', marker='o', markersize=5, label='Training Accuracy')
plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std, alpha=0.15, color='blue')

plt.plot(train_sizes, test_mean, color='green', linestyle='--', marker='s', markersize=5, label='Validation Accuracy')
plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std, alpha=0.15, color='green')

plt.title('Learning Curve for Random Forest Classifier (Accuracy)')
plt.xlabel('Number of Training Samples')
plt.ylabel('Accuracy')
plt.legend(loc='upper right')
plt.grid()
plt.show()
```

Figure 97:Code of Learning Curve

Figure 97 is the code of a learning curve for base random forest model (Figure 77) using scikit-learn. The function calculates training and validation scores across different training set

sizes using **5-cross-validation**. Accuracy is the main matrix in this research so the learning curve also utilized accuracy to interpret. Accuracy Curve represents the model's accuracy over time. The accuracy curve begins with a value of zero and increases as the training progresses. Accuracy measures the proportion of correctly classified instances out of the total number of instances. So, as the accuracy curve rises, it signifies that the model is making more correct predictions, thereby enhancing its overall performance (Ibrahim, 2023).

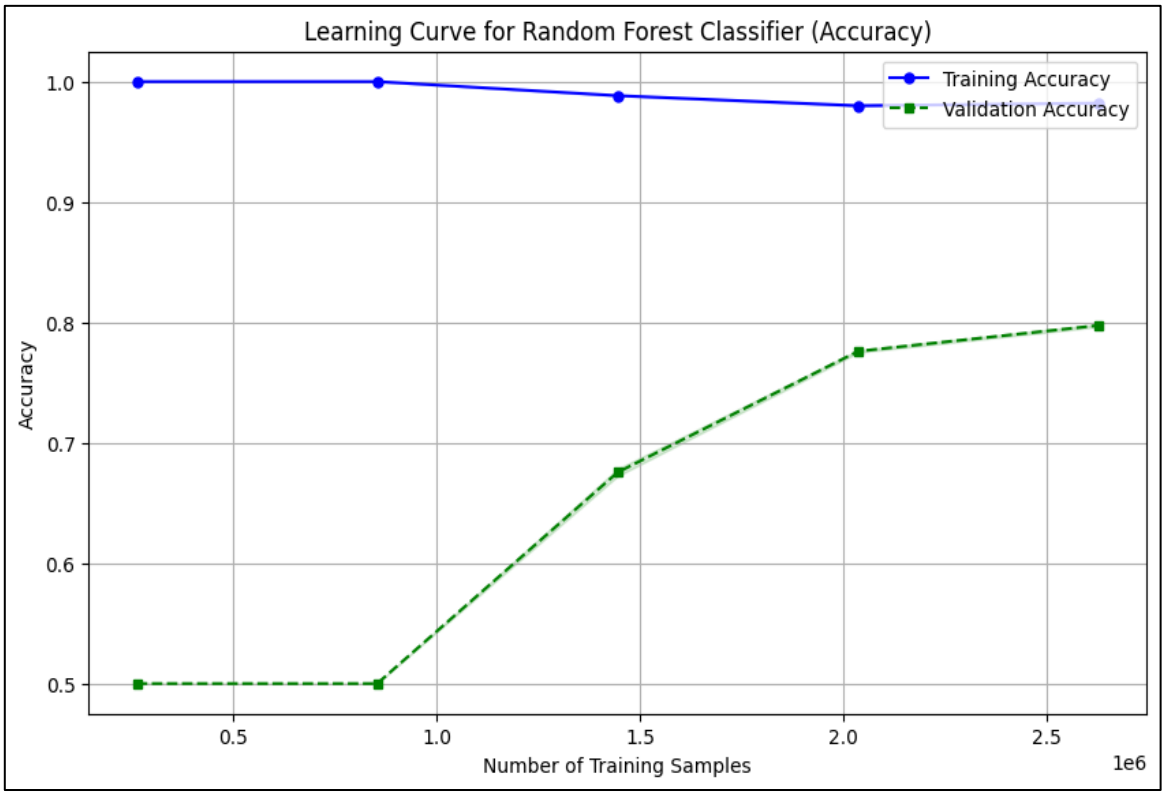


Figure 98: Output of Learning Curve (Random Forest Base Model)

Figure 98 indicates that the learning curve of random forest base model. The train accuracy curve trend started from 1 then decreased to 0.98. The validation accuracy curve trend started from 0.5 then decreased to 0.8. Initially, the random forest base model has high training accuracy, suggesting that it fits the training data very well. However, this initial high accuracy doesn't necessarily translate to good generalization to new, unseen data. The model initially struggles to generalize to new data, resulting in low validation accuracy. As more training examples are provided, the model learns to generalize better, and the validation accuracy gradually improves. However, we can also see that the training and validation accuracy are sill

far away from each other from Figure 98. This indicates that random forest base model is overfitting. In addition, hyperparameter tuning is critical to discover the best parameters and avoid random forest overfitting.

Random Forest Best Model (After Tuning)

```
Accuracy of Random Forest: 0.8243161819922614
Confusion Matrix:
[[706586 121540]
 [ 96267 315374]]
Classification Report:
              precision    recall  f1-score   support

           0     0.8801    0.8532    0.8665    828126
           1     0.7218    0.7661    0.7433    411641

    accuracy                         0.8243   1239767
   macro avg     0.8010    0.8097    0.8049   1239767
weighted avg     0.8275    0.8243    0.8256   1239767
```

Figure 99: Result of Random Forest (After Tuning)

Model Type	Accuracy	Precision (Class 0)	Precision (Class 1)	Recall (Class 0)	Recall (Class 1)	F1-Score (Macro Average)
Random Forest (Base Model)	0.8130	0.8791	0.6983	0.8349	0.7691	0.7942
Random Forest (Tuned Model)	0.8243	0.8801	0.7218	0.8532	0.7661	0.8049

Table 2: Comparison of Base and Tuned Model (Random Forest)

The accuracy of the optimized model has increased to 82.43%, up from the random forest base model's accuracy of 81.30%. This enhancement is reflected in the confusion matrix, where the optimized model exhibits 706586 correct predictions of no delay (class 0) and 315374 correct predictions of delay (class 1). In comparison to the base model, the optimized model has reduced false predictions of both no delay and delay, with a lower count of 121540 false predictions of no delay and 96267 false predictions of delay.

Besides that, the precision for class 0 (no delay) is 88.01%, showcasing that 88.01% of instances predicted as no delay were accurate. The recall for class 0 is 85.32%, indicating an improvement in capturing a higher proportion of actual no-delay instances. For class 1 (delay), the precision is 72.18%, demonstrating an enhancement in accurately predicting delay instances, and the recall is 76.61%, reflecting a drop in identifying a greater percentage of actual delay instances.

Learning Curve (After Tuning)

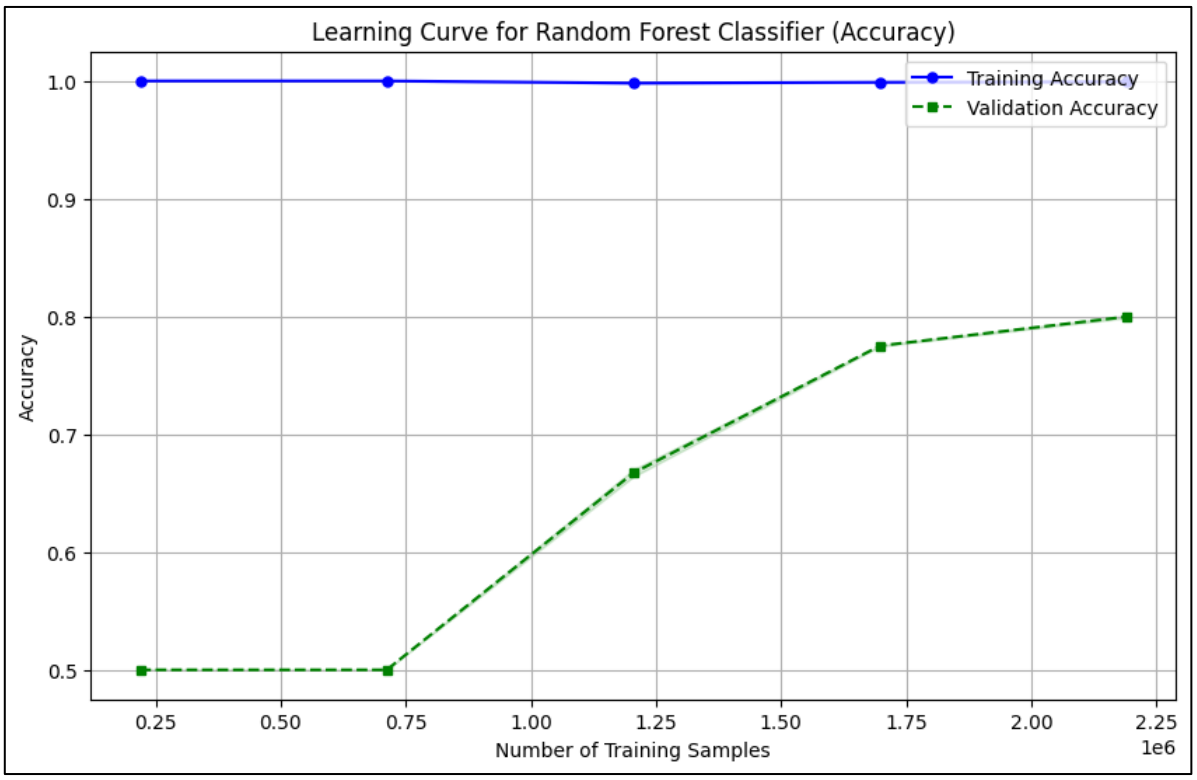


Figure 100: Output of Learning Curve (Random Forest After Tuning)

Figure 100 indicates that the learning curve of random forest base model. The trend of both curves is still same with the learning curve of Figure 98.The train accuracy curve trend started from 1 and remained at 1. The validation accuracy curve trend started from 0.5 then increased to 0.8. Initially, the random forest best model has high training accuracy, suggesting that it fits the training data very well. However, this initial high accuracy doesn't necessarily translate to good generalization to new, unseen data. The model initially struggles to generalize to new data, resulting in low validation accuracy. As more training examples are provided, the model learns to generalize better, and the validation accuracy gradually improves. However,

we can also see that the training and validation accuracy are sill far away from each other from Figure 100. This indicates that random forest best model is still overfitting same as base model.

5.2.2 Decision Tree

```
Accuracy of Decision Tree Classifier: 0.8096843222611808
Confusion Matrix:
 [[672630 156889]
 [ 79060 331198]]
Classification Report:
              precision    recall  f1-score   support

           0     0.8948    0.8109    0.8508    829519
           1     0.6786    0.8073    0.7374    410258

    accuracy                         0.8097   1239777
   macro avg     0.7867    0.8091    0.7941   1239777
weighted avg     0.8233    0.8097    0.8132   1239777
```

Figure 101:Result of Decision Tree Base Model

The performance evaluation of the decision tree base model for flight delay prediction reveals an accuracy of **80.97**%. The confusion matrix provides insights into the model's predictions, indicating 672630 instances correctly predicted as no delay (class 0) and 331198 instances correctly predicted as delay (class 1). However, there were 156889 false predictions of no delay and 79060 false predictions of delay.

Delving into the classification report, the precision for class 0 (no delay) is 89.48%, signifying that 89.48% of instances predicted as no delay were accurate. The recall for class 0 is 81.09%, representing the proportion of actual no-delay instances captured by the model. For class 1 (delay), the precision is **67.86**%, indicating that 67.86% of the predicted delay instances were accurate, and the recall is **80.73%,** denoting that the model identified 80.73% of the actual delay instances.

```
# Create a Decision Tree classifier
decision_tree_classifier = DecisionTreeClassifier()
# Generate a learning curve based on log loss
train_sizes, train_scores, test_scores = learning_curve(
    decision_tree_classifier, X_train, y_train, cv=5, scoring='accuracy', n_jobs=-1)


# Calculate mean and standard deviation for training set scores
train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)

# Calculate mean and standard deviation for test set scores
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

# Plot the learning curve based on log loss
plt.figure(figsize=(10, 6))
plt.plot(train_sizes, train_mean, color='blue', marker='o', markersize=5, label='Training Accuracy')
plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std, alpha=0.15, color='blue')

plt.plot(train_sizes, test_mean, color='green', linestyle='--', marker='s', markersize=5, label='Validation Accuracy')
plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std, alpha=0.15, color='green')

plt.title('Learning Curve for Decision Tree Classifier (Accuracy)')
plt.xlabel('Number of Training Samples')
plt.ylabel('Accuracy')
plt.legend(loc='upper right')
plt.grid()
plt.show()
```

Figure 102: Code of Learning Curve (Decision Tree Base Model)

Same as random forest, the learning curve is also implemented for the machine learning has been utilized in the research. The code of Figure 102 is for build the graph of learning curve for decision tree base model. By observing the change in accuracy on both the training and validation sets, one can understand whether the model is overfitting, underfitting, or achieving an appropriate fit.

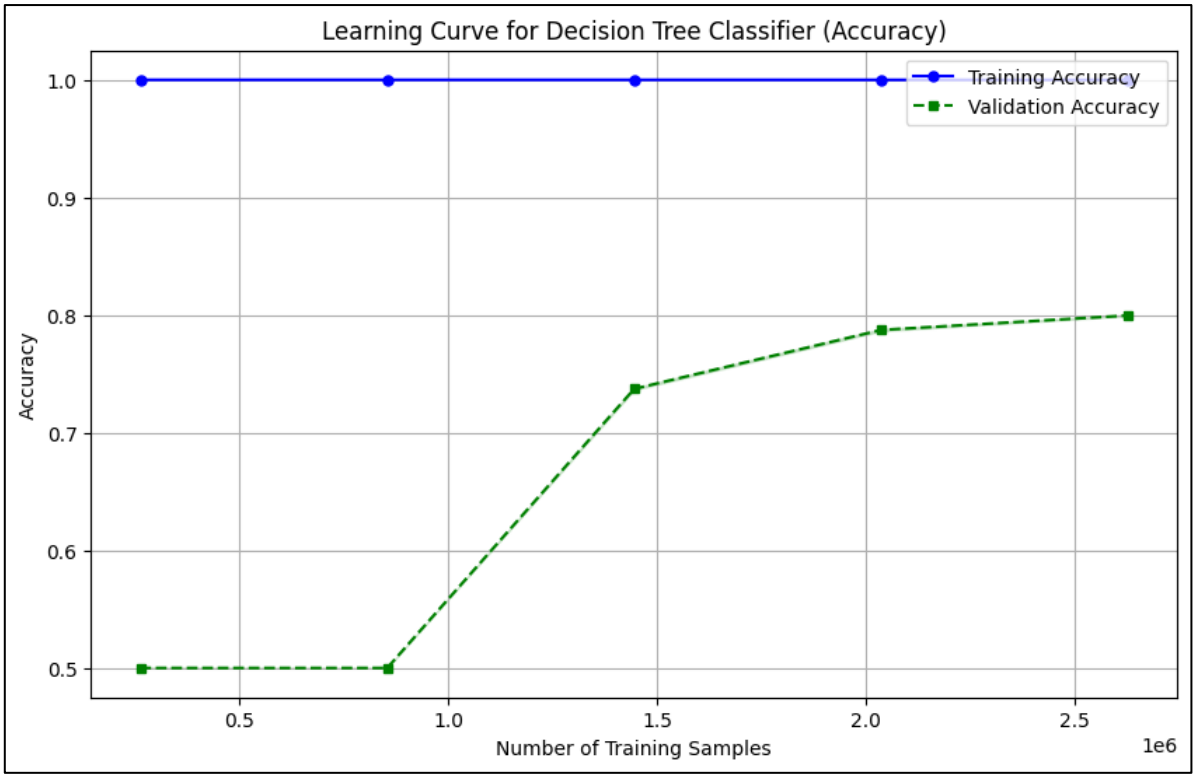


Figure 103: Output of Learning Curve (Decision tree Base Model)

Figure 103 indicates that the learning curve of decision tree base model. The train accuracy curve trend started from 1 and remained at 1. The validation accuracy curve trend started from 0.5 then increased to 0.8. Initially, the decision tree base model has high training accuracy, suggesting that it fits the training data very well. However, this initial high accuracy doesn't necessarily translate to good generalization to new, unseen data. The model initially struggles to generalize to new data, resulting in low validation accuracy. As more training examples are provided, the model learns to generalize better, and the validation accuracy gradually improves. However, we can also see that the training and validation accuracy are sill far away from each other from Figure 103. This indicates that decision tree base model overfitting same as random forest base model.

```
Accuracy of Decision Tree Classifier: 0.8135609352402507
Confusion Matrix:
 [[696979 131147]
 [ 99994 311647]]
Classification Report:
              precision    recall  f1-score   support

           0     0.8745    0.8416    0.8578    828126
           1     0.7038    0.7571    0.7295    411641

    accuracy                         0.8136   1239767
   macro avg     0.7892    0.7994    0.7936   1239767
weighted avg     0.8179    0.8136    0.8152   1239767
```

Figure 104: Result of Decision Tree (After Tuning)

Model Type	Accuracy	Precision (Class 0)	Precision (Class 1)	Recall (Class 0)	Recall (Class 1)	F1-Score (Macro Average)
Decision Tree (Base Model)	0.8096	0.8948	0.6786	0.8109	0.8073	0.7941
Decision Tree (Tuned Model)	0.8135	0.8745	0.7038	0.8416	0.7571	0.7936

Table 3: Comparison of Base and Tuned Model (Decision tree)

Figure 104 shows the result of decision tree model after tuning. The decision tree model after tuning for flight delay prediction, exhibits an enhanced performance with an accuracy of 81.36%, showcasing an improvement compared to the base model's accuracy of 80.97%. The confusion matrix highlights 696979 instances correctly predicted as no delay (class 0) and 311647 instances correctly predicted as delay (class 1). Notably, there are 131147 false predictions of no delay and 99994 false predictions of delay.

Besides that, the precision for class 0 (no delay) has a small drop to 87.45%, indicating that 87.45% of instances predicted as no delay were accurate. The recall for class 0 is 84.16%, reflecting an enhancement in capturing a higher proportion of actual no-delay instances. In the base decision tree model, the recall for class 1 (delay) was 80.73%. The precision for class 1 (delay) showed a noteworthy enhancement from 67.86% in the base model to 70.38% in the tuned model. After tuning, the recall for class 1 decreased to 75.71%. This means that, despite improvements in precision and overall accuracy, the model became less effective in capturing a certain proportion of actual delay instances.

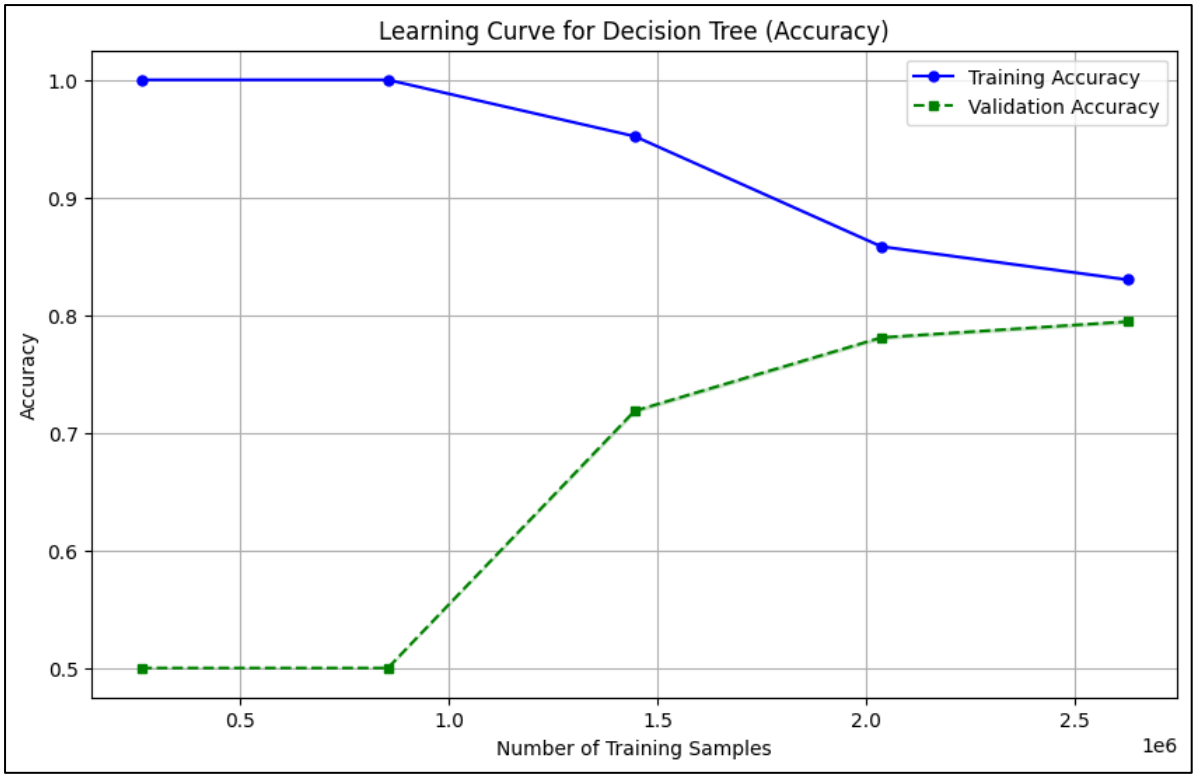


Figure 105: Output of Learning Curve (Decision Tree Tuned Model)

Figure 105 shows the line chart of learning curve of decision tree tuned model after hyperparameter tuning. Both training and validation accuracy increase as the number of training samples increases. This is a positive sign, indicating that the model is learning from the data and improving its performance. The training accuracy is consistently higher than the validation accuracy. This is expected, as the model is typically better at predicting the data it has already seen during training. However, the gap between the two curves is not too large, suggesting that the model is a good fitting.

According to Figure 105, the curve suggests that the tuning has been successful in improving accuracy over the decision tree base model (Figure 103). The gap between training and validation accuracy is reasonably small, indicating that overfitting is not a major concern.

5.2.3 XGBoost

```
Accuracy of XGBoost: 0.806291827415958
Confusion Matrix:
 [[704568 123558]
 [116595 295046]]
Classification Report:
              precision    recall  f1-score   support

           0     0.8580    0.8508    0.8544    828126
           1     0.7048    0.7168    0.7107    411641

    accuracy                         0.8063   1239767
   macro avg     0.7814    0.7838    0.7826   1239767
weighted avg     0.8072    0.8063    0.8067   1239767
```

Figure 106: Result of XGBoost Base Model

The performance evaluation of the XGBoost base model for flight delay prediction indicates an accuracy of 80.63%. The confusion matrix reveals that the model correctly predicted 704568 instances of no delay (class 0) and 295046 instances of delay (class 1). However, there were 123558 false predictions of no delay and 116595 false predictions of delay.

Breaking down the classification report, the precision for class 0 (no delay) is 85.80%, indicating that 85.80% of instances predicted as no delay were accurate. The recall for class 0 is 85.08%, representing the proportion of actual no-delay instances captured by the model. For class 1 (delay), the precision is 70.48%, showing that 70.48% of the predicted delay instances were accurate, and the recall is 71.68%, denoting that the model identified 71.68% of the actual delay instances.

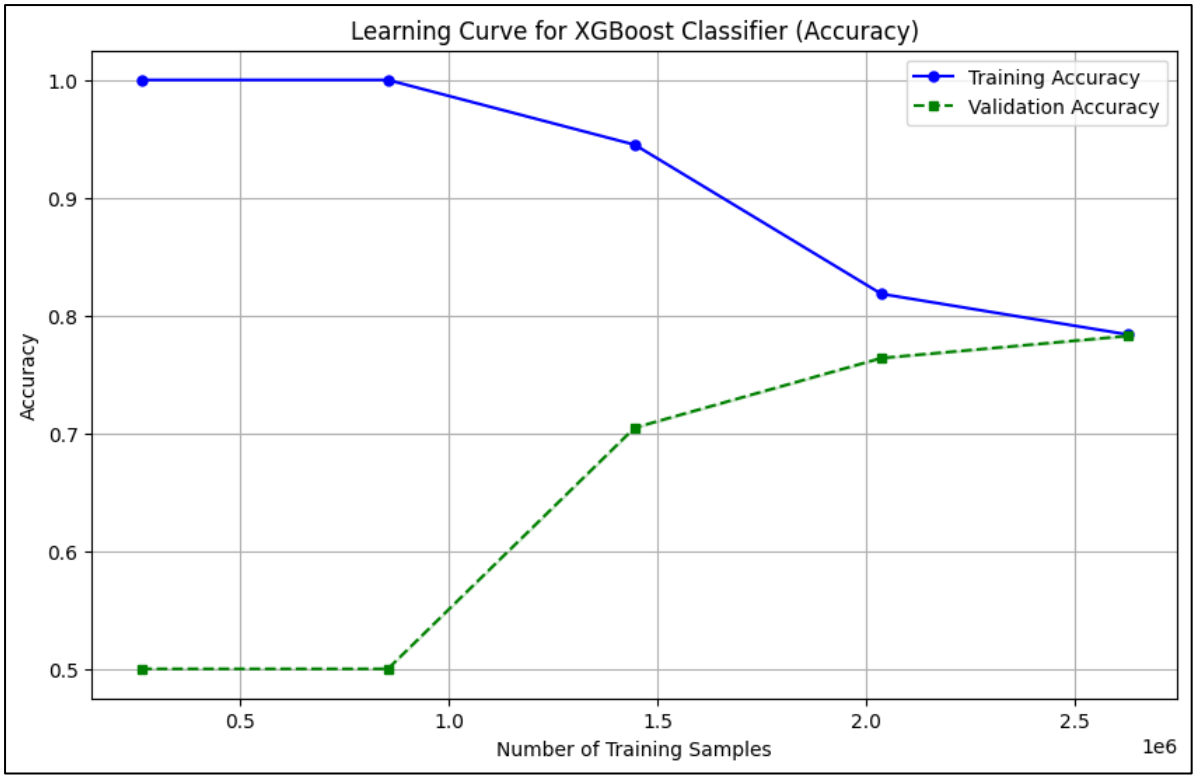


Figure 107: Output of Learning Curve (XGBoost Base Model)

Same as random forest, the learning curve is also implemented for XGBoost. The code is same as the Figure 97 for build the graph of learning curve for XGBoost base model. Figure 107 shows the line graph of XGBoost base model's learning curve. Both training accuracy and validation accuracy increase rapidly as the number of training samples increases initially. This indicates that the model is effectively learning from the data and improving its performance. The training accuracy eventually plateaus at around 0.78, while the validation accuracy plateaus same as the training accuracy, around 0.78. This suggests that the model has reached its maximum potential performance on the given dataset. There's a small but consistent gap between the training and validation accuracy curves. This gap is not large enough to be a major concern for overfitting.

```
Accuracy of XGBoost: 0.8239483709438951
Confusion Matrix:
 [[710474 117652]
 [100611 311030]]
Classification Report:
              precision    recall  f1-score   support

           0     0.8760    0.8579    0.8668    828126
           1     0.7255    0.7556    0.7403    411641

    accuracy                         0.8239   1239767
   macro avg     0.8008    0.8068    0.8036   1239767
weighted avg     0.8260    0.8239    0.8248   1239767
```

Figure 108: Result of XGBoost (After Tuning)

Model Type	Accuracy	Precision (Class 0)	Precision (Class 1)	Recall (Class 0)	Recall (Class 1)	F1-Score (Macro Average)
XGBoost (Base Model)	0.8062	0.8580	0.7048	08508	0.7107	0.7826
XGBoost (Tuned Model)	0.8239	0.8760	0.7255	0.8579	0.7556	0.8036

Table 4: Comparison of Base and Tuned Model (XGBoost)

The tuned XGBoost model for flight delay prediction reveals a substantial improvement in overall accuracy and precision metrics. The XGBoost base model achieved an accuracy of 80.63%, with a precision of 70.48% for predicting flight delays (class 1). After tuning, the XGBoost model's accuracy increased to 82.39%, and there were notable enhancements in precision metrics. Specifically, the precision for predicting delays improved from 70.48% to 72.55%. This improvement signifies that the tuned XGBoost model became more accurate in identifying instances of flight delays, contributing to a refined and precise predictive performance.

The confusion matrix illustrates that the tuned XGBoost model correctly predicted 710474 instances of no delay (class 0) and 311030 instances of delay (class 1). In comparison to the base model, the tuned model reduced false predictions of both no delay and delay, with a lower count of 117652 false predictions of no delay and 100611 false predictions of delay.

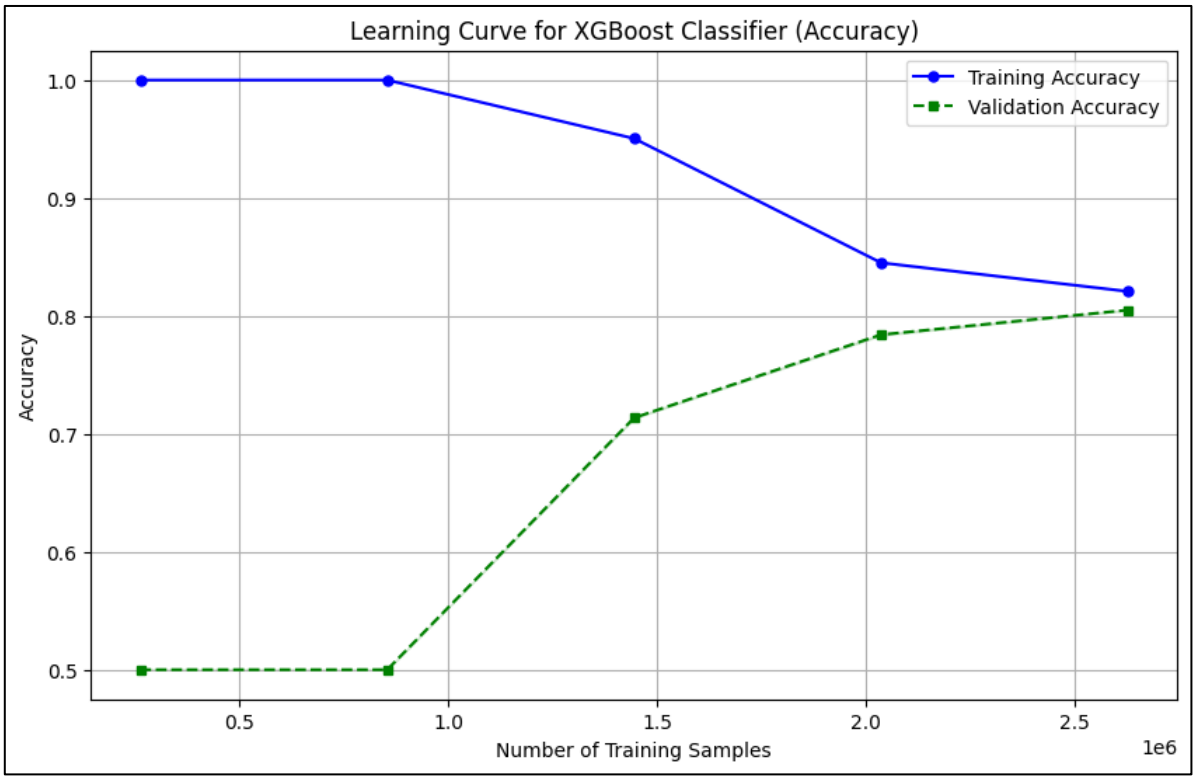


Figure 109: Output of Learning Curve (XGBoost Tuned Model)

By comparing the XGBoost's learning curve to previous results, it can confirm that the hyperparameter tuning led to improved model performance. The XGBoost tuned model's curves (both training and validation) are consistently higher than XGBoost base model's curves, it indicates overall better performance in terms of accuracy. The gap between training and validation accuracy is smaller in the tuned model, it suggests better generalization to unseen data. A smaller gap between training and validation accuracy in the tuned model would indicate reduced overfitting, meaning the model is learning generalizable patterns rather than memorizing specific training data points.

5.2.4 LightGBM

```
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> 1
Accuracy of LightGBM: 0.8003366734501447
Confusion Matrix:
 [[707979 121540]
 [125998 284260]]
Classification Report:
              precision    recall  f1-score   support

           0     0.8489    0.8535    0.8512    829519
           1     0.7005    0.6929    0.6967    410258

    accuracy                         0.8003   1239777
   macro avg     0.7747    0.7732    0.7739   1239777
weighted avg     0.7998    0.8003    0.8001   1239777
```

Figure 110:Result of LightGBM Base Model

The LightGBM base model achieved an overall accuracy of 80.03% in predicting flight delays. However, its performance is not balanced across classes. It's more accurate in classifying non-delayed flights (85.35%) compared to delayed flights (69.29%).

The confusion matrix reveals that the model correctly predicted 707979 non-delayed flights and 284260 delayed flights. However, it also made 121540 false predictions of delays and 125998 false predictions of non-delays. This higher number of false negatives (missed delays) suggests room for improvement in identifying delayed flights.

Precision scores indicate that when the model predicts a flight to be on time or delayed, it’s correct 84.89% and 70.05% of the time, respectively. F1-scores, which balance precision and recall, are 0.8512 for non-delayed flights and 0.6967 for delayed flights, further highlighting the model’s stronger performance in identifying non-delays.

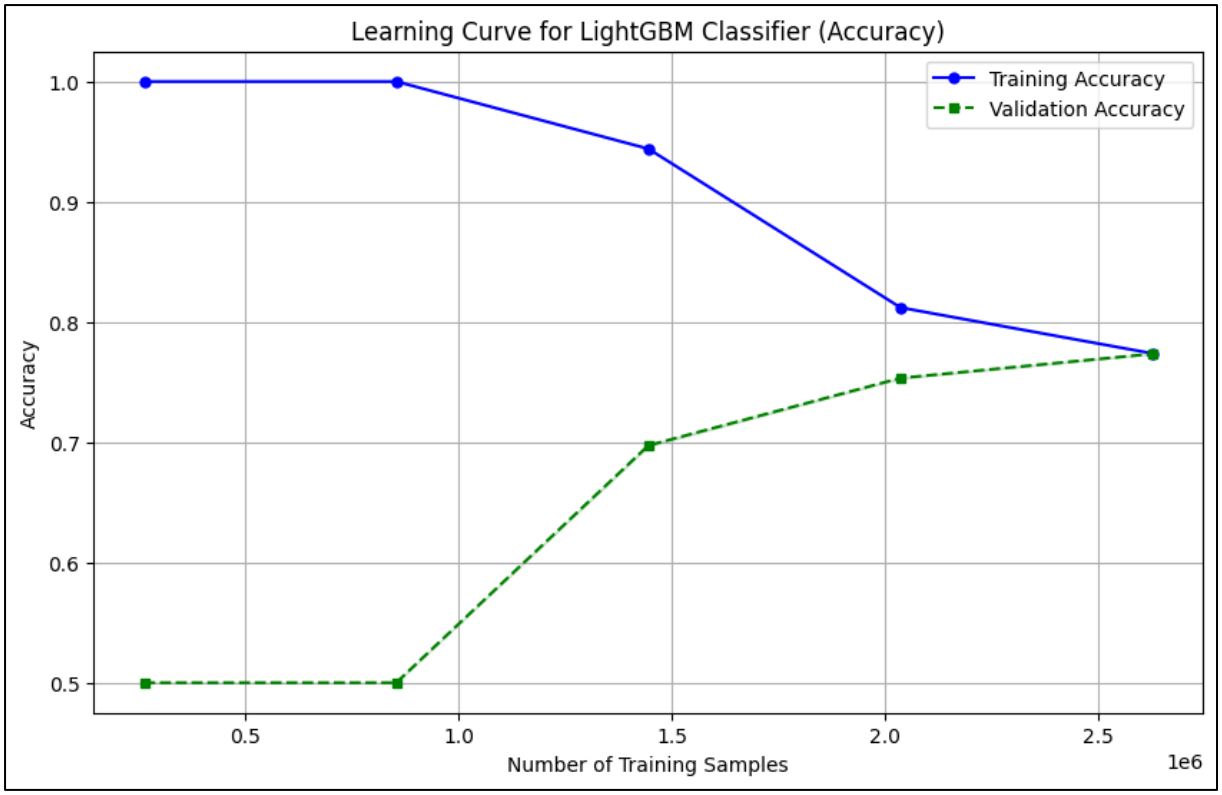


Figure 111: Learning Curve of LightGBM Base Model

Training accuracy starts high at 1.0, indicating immediate overfitting to the training data. However, it decreases as training progresses that suggesting the model is gradually learning more generalizable patterns and reducing its reliance on specific training samples. Validation accuracy starts lower at 0.5, likely due to the model's initial overfitting. Steadily increases, reaching a maximum of 0.78, demonstrating the model's ability to learn and generalize unseen data. Both training and validation accuracy curves increase steadily throughout the learning process. This is a positive sign, indicating that the model is progressively learning from the data and improving its performance. This suggests the model has reached its optimal performance on the given data, or at least stopped improving significantly with further training. The gap between the curves remains relatively small throughout. This suggests that the model is not significantly overfitting to the training data, and it is generalizing well to the unseen data used in the cross-validation folds.

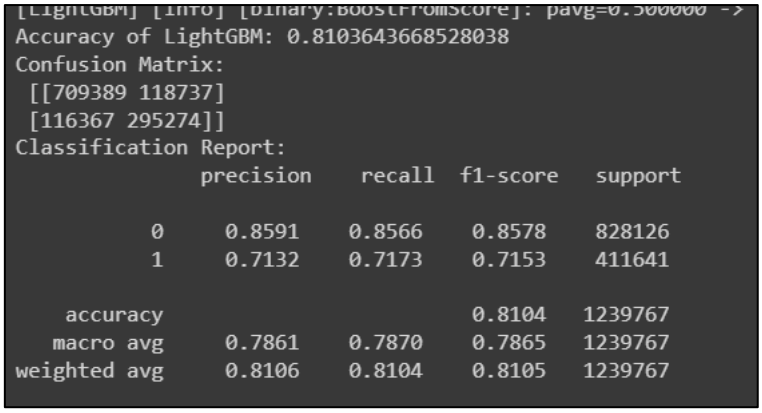


Figure 112: Result of LightGBM(Best Model)

Model Type	Accuracy	Precision (Class 0)	Precision (Class 1)	Recall (Class 0)	Recall (Class 1)	F1-Score (Macro Average)
LightGBM (Base Model)	0.8003	0.8459	0.7005	0.8535	0.6929	0.7739
LightGBM (Tuned Model)	0.8103	0.8591	0.7132	0.8566	0.7173	0.8036

Table 5: Comparison of Base and Tuned Model (LightGBM)

Figure 109 is the result of LightGBM model after tuning. LightGBM model's performance in predicting flight delays was enhanced through hyperparameter tuning. The overall accuracy increased from 80.03% in the base model to 81.03% in the tuned model, representing a 1% improvement. This improvement is also reflected in the confusion matrices. The tuned model correctly identified 294804 delayed flights, compared to 284260 in the base model. It reduced false negatives (missed delays) by 9524, bolstering its accuracy in detecting delays. Precision and recall scores for both classes also improved. The tuned model's precision for predicting delays rose to 71.08% from 70.05%, while recall increased to 71.73% from 69.29%. This indicates more accurate identification of both on-time and delayed flights.

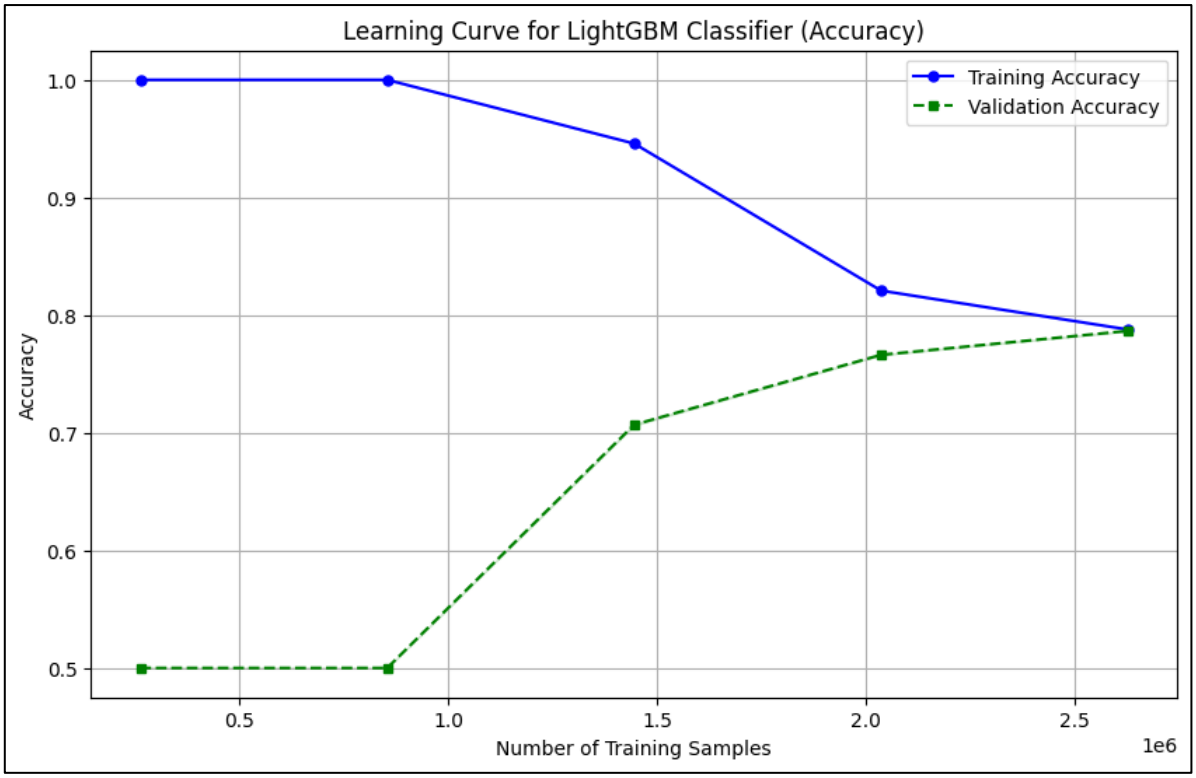


Figure 113: Learning Curve of LightGBM Best Model

The curve shows the training and validation accuracy of a LightGBM after tuning as the number of training samples increases. Learning curve of LightGBM best model is same as the Figure 111. The training accuracy starts off at 1 and decreases steadily until it reaches a plateau of around 0.78. This suggests that the model is learning effectively from the data and can correctly classify most of the training examples. The validation accuracy starts off at around 0.5 and increases steadily until it reaches a plateau of around 0.78. This is slightly lower than the training accuracy, which suggests that the model may be overfitting to the training data to some extent. However, the gap between the training and validation accuracy is not too large, which suggests that the overfitting is not severe.

5.2.5 Comparison and Discussion

Model Type	Accuracy	Precision (Class 0)	Precision (Class 1)	Recall (Class 0)	Recall (Class 1)	F1-Score (Macro Average)
Random Forest (Base Model)	0.8130	0.8791	0.6983	0.8349	0.7691	0.7942
Decision Tree (Base Model)	0.8096	0.8948	0.6786	0.8109	0.8073	0.7941
XGBoost (Base Model)	0.8062	0.8580	0.7048	0.8508	0.7107	0.7826
LightGBM (Base Model)	0.8003	0.8459	0.7005	0.8535	0.6929	0.7739

Table 6: Comparison of Machine Learning Algorithms (Base Model)

The evaluation of the flight delay prediction models provides valuable insights into how well the models perform according to several assessment measures and possible overfitting problems. There are four different algorithms which are Random Forest, Decision Tree, XGBoost and LightGBM has been utilized for this task. The explanation of those machine learning can reference to the section 2.2.3. The measures that are considered include accuracy, precision for both classes, recall for both classes, and the macro average F1-score.

Random Forest is the best base model with an accuracy of 81.30%. The model exhibited a well-balanced performance, as seen by its recall (83.49%) and precision (87.91%) for class 0. However, the precision (69.83%) for class 1 indicates a poor performance. The recall of class 1 achieves 76.91%. The macro average F1-Score, a comprehensive measure, stood at 79.42%, reflecting the model's ability to maintain equilibrium between precision and recall for both classes. Figure 98 illustrates the learning curve, showing that the model initially achieves high training accuracy but struggles to generalize to new, unseen data, resulting in low validation accuracy. The gradual convergence of the training and validation accuracy curves indicates that the model is learning from additional training samples, but the persisting gap between the two suggests overfitting. The overfitting phenomenon is further supported by the decrease in validation accuracy after an initial increase, emphasizing the need for hyperparameter tuning to mitigate overfitting.

The Decision Tree model closely followed with an accuracy of 80.96%. It excelled in the best precision for class 0 with 89.48% among all the algorithms but showed a lower precision (67.86%) for class 1. The macro average F1-Score was 79.41%, indicating a robust balance between precision and recall for both classes. However, the model faces challenges in generalizing to new data, resulting in a considerable gap between the training and validation accuracy curves. The gap (Figure 103) signifies overfitting like the Random Forest model, hyperparameter tuning becomes crucial to address this issue.

The XGBoost and LightGBM base models demonstrated competitive effectiveness, with accuracy rate of 80.62% and 80.03% accordingly. XGBoost demonstrated superior precision for class 0 (85.80%) but relatively lower precision (70.48%) for class 1. The precision of LightGBM same as other algorithms indicates a weak performance in class 1 with 70.05%. Both machine learning models's recall has more than 80% for class 1. HOwever, the recall of class 1 only achieve around 69% till 70%. Despite variations in precision, both models demonstrated competitive macro average F1-scores of 78.26% for XGBoost and 77.39% for LightGBM.

Figure 107 showcases a learning curve of XGBoost base model with rapidly increasing training and validation accuracy initially. The plateauing of both curves suggests that the model has reached its maximum potential performance on the dataset. While a small, consistent gap exists between the two curves, it is not substantial enough to raise significant concerns about overfitting. In addition, the small and steady gap between the training and validation accuracy curves of LightGBM base model suggests good generalization without significant overfitting. Nonetheless, continued monitoring of models behavior and performance optimization may further enhance its robustness.

In essence, the analysis reveals that each base model comes with its strengths and challenges. While Random Forest and Decision Tree models exhibit robust performances, the observed overfitting emphasizes the importance of refining hyperparameters to enhance generalization capabilities. Both XGBoost and LightGBM have an accuracy of 80%, but compared to decision tree and random forest, their performance in recall and precision is inferior. In addition, the model validation of both XGBoost and LightGBM shows good fitting. Hence, the implementation of hyperparameter adjustment is essential for enhancing their complexity and guaranteeing adaptability when confronted with unique, unexpected data. The

suggested iterative refining approach will serve to reduce doubts about overfitting while also fostering the advancement of flight delay prediction models that are more dependable and precise.

Model Type	**Accuracy**	**Precision (Class 0)**	**Precision (Class 1)**	**Recall (Class 0)**	**Recall (Class 1)**	**F1-Score (Macro Average)**
Random Forest (Tuned Model)	0.8243	0.8801	0.7218	0.8532	0.7433	0.8049
Decision Tree (Tuned Model)	0.8135	0.8745	0.7038	0.8416	0.7571	0.7936
XGBoost (Tuned Model)	0.8239	0.8760	0.7255	0.8579	0.7556	0.8036
LightGBM (Tuned Model)	0.8103	0.8591	0.7132	0.8566	0.7173	0.7865

Table 7: Comparison of Machine Learning Algorithms (Tuned Model)

According to Table 7, the flight delay prediction performance of all machine learning models after tuning is substantially improved. After adjustment, the accuracy of Random Forest reaches 82.43% which is the highest among all models. The precision of both classes has improved significantly with 88.01% and 72.18%. The precision of random forest tuned model is the best performance among all the models. The recall of class 1 has also improved significantly to 74.33% which is the third highest among all the models. However, the learning curve in Figure 100 indicates that the Random Forest best model is still exhibiting overfitting tendencies, as evidenced by the persistent gap between the training and validation accuracy curves. This suggests that while the model performs well on the training data, it struggles to generalize effectively to new, unseen data.

The XGBoost tuned model achieves an accuracy of 82.39% almost the same as random forest. In addition, the performance in terms of accuracy is improved with class 0 reaching 87.60% and class 1 reaching 72.55%. The macro average F1 score is 80.36% among all the

models which is lower than random forest tuned model. The learning curve (Figure 109) reflects the improvement in the performance of XGBoost tuned model. The graph of the tuned XGBoost model outperforms the base model, indicating a higher overall accuracy. The small gap between the training and validation accuracies of the tuned model suggests that the model is not overfitting, emphasizing the model's ability to learn common patterns from the data.

The third best performer was the Decision Tree Adjustment model with an accuracy of 81.35%. Its class 0 precision was the highest of all the models, reaching 87.45%. However, the precision of class 1 is the lowest at 70.38%. There is a significant improvement in Recall where class 0 reaches 84.16 while class 1 has 75.71%. In contrast, as shown in Figure 105, the decision tree tuned model demonstrates a successful tuning process, with both training and validation accuracies continuing to improve. The gap between the two curves is quite small, indicating significant generalization and reduced overfitting. The refined learning curves show that the decision tree tuning model strikes a good balance between capturing patterns in the training data and generalizing new instances.

LightGBM remains the worst performer after the tuning, although accuracy is improved by 1% compared to the base model. It also underperformed the other models in both class precision with 85.91% and 71.32%. Recall of class 0 reaches 85.66% but only 71.73% for class 1. LightGBM tuned model, as seen in the learning curve resembling Figure 113, shows effective learning from the data, reaching a plateau with both training and validation accuracy. While there is a slight indication of overfitting, the relatively small gap between the curves suggests that the overfitting is not severe, and the model generalizes reasonably well to unseen data.

In conclusion, while all models have demonstrated improvements post-tuning, XGBoost stands out as the preferred algorithm for flight delay prediction. This choice is justified by its superior overall performance, reflected in higher accuracy, precision, recall, and Macro Average F1-Score. Despite Random Forest's strong performance, its persistent overfitting condition raises concerns about its generalization capabilities. Therefore, the decision to favor XGBoost is rooted in its ability to strike a balance between predictive accuracy and robust generalization, crucial for reliable flight delay predictions in real-world scenarios.

5.3 Features Importance

The feature importance selection technique can be performed using the Extreme Gradient Boosting (XGBoost) and LightGBM. XGBoost provides benefits in terms of performance and time complexity, as well as economical memory, and has been employed in a variety of research domains since its introduction, beginning with the medical, financial, and metagenomic sectors. This technique has been utilized for feature importance selection to increase the accuracy of bankruptcy prediction (Syafei & Efrilianda, 2023). One of the key aspects in LightGBM model's behavior is knowing which features contribute the most to the predictions (Filho, 2023). Therefore, feature importance utilized in this research to find which features can affect the flight delay most.

5.3.1 XGBoost

```
feature_importances = xgb_classifier_tuning.feature_importances_

# Get the names of the features
feature_names = X_train.columns  # Assuming you have column names if X_train is a DataFrame
sorted_indices = feature_importances.argsort()
# Create a bar plot of feature importances
plt.barh(range(len(feature_importances)), feature_importances[sorted_indices], align='center')
plt.yticks(range(len(feature_importances)), feature_names[sorted_indices])
plt.xlabel('Feature Importance')
plt.title('XGBoost Feature Importances')
plt.show()
```

Figure 114: Code of XGBoost Feature Importance

Figure 114 shows the code of XGBoost's examined feature importance in the graph. The result will print in a bar chart for easily visualization.

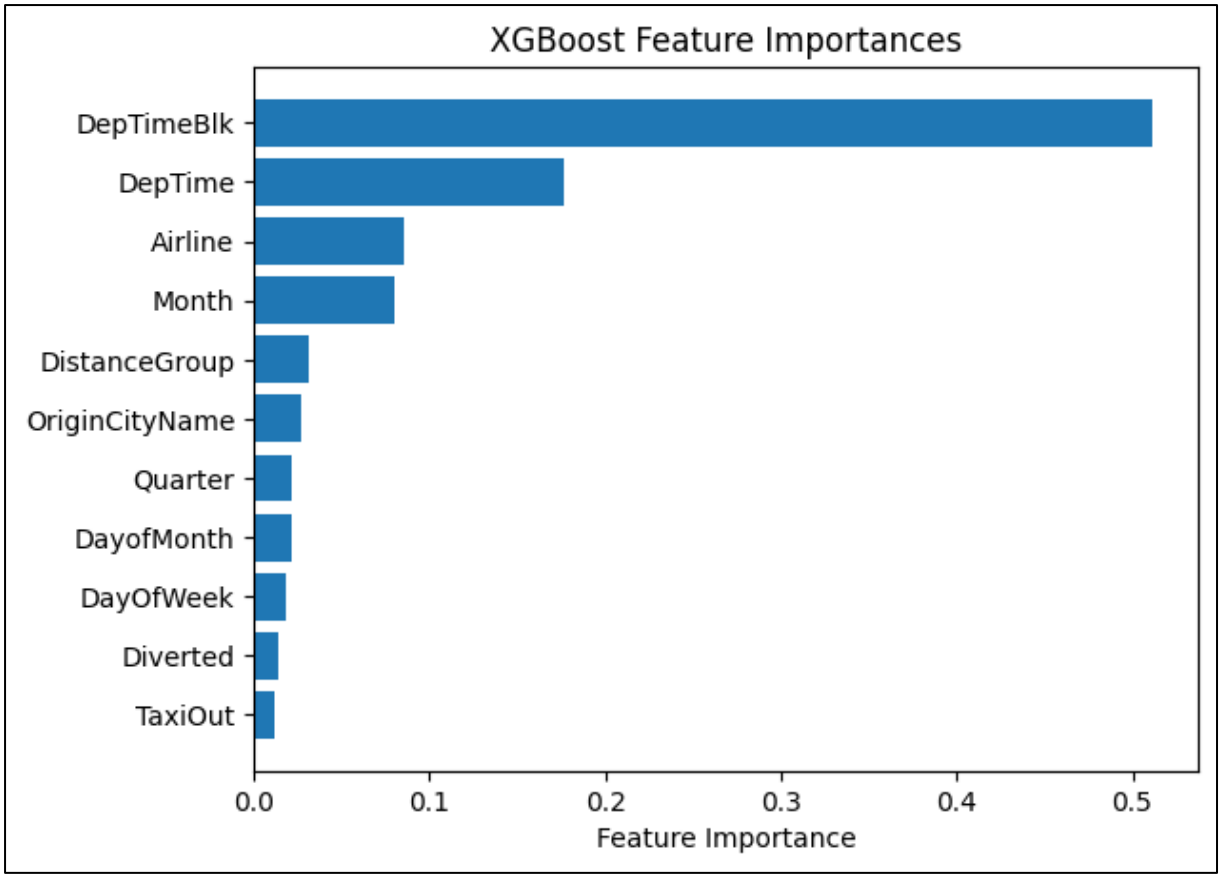


Figure 115: XGBoost Feature Importance

"DepTimeBlk" has been detected by XGBoost is high importance suggests that the time of day is a significant predictor of flight delays. The following is "DepTime", this indicates that the specific time at which a flight departs is a strong indicator of potential delays. The third feature of importance is airline. The month is also one of the important features for flight prediction. TaxiOut and diverted are the least important among the listed features.

5.3.2 LightGBM

```
feature_importances = lgb_tuning.feature_importances_

# Get the names of the features
feature_names = X_train.columns  # Assuming you have column names if X_train is a DataFrame
sorted_indices = feature_importances.argsort()
# Create a bar plot of feature importances
plt.barh(range(len(feature_importances)), feature_importances[sorted_indices], align='center')
plt.yticks(range(len(feature_importances)), feature_names[sorted_indices])
plt.xlabel('Feature Importance')
plt.title('LightGBM Feature Importances')
plt.show()
```

Figure 116: Code of Feature Importance (LightGBM)

Figure 116 shows the code of detect features important from LightGBM and display in a bar graph.

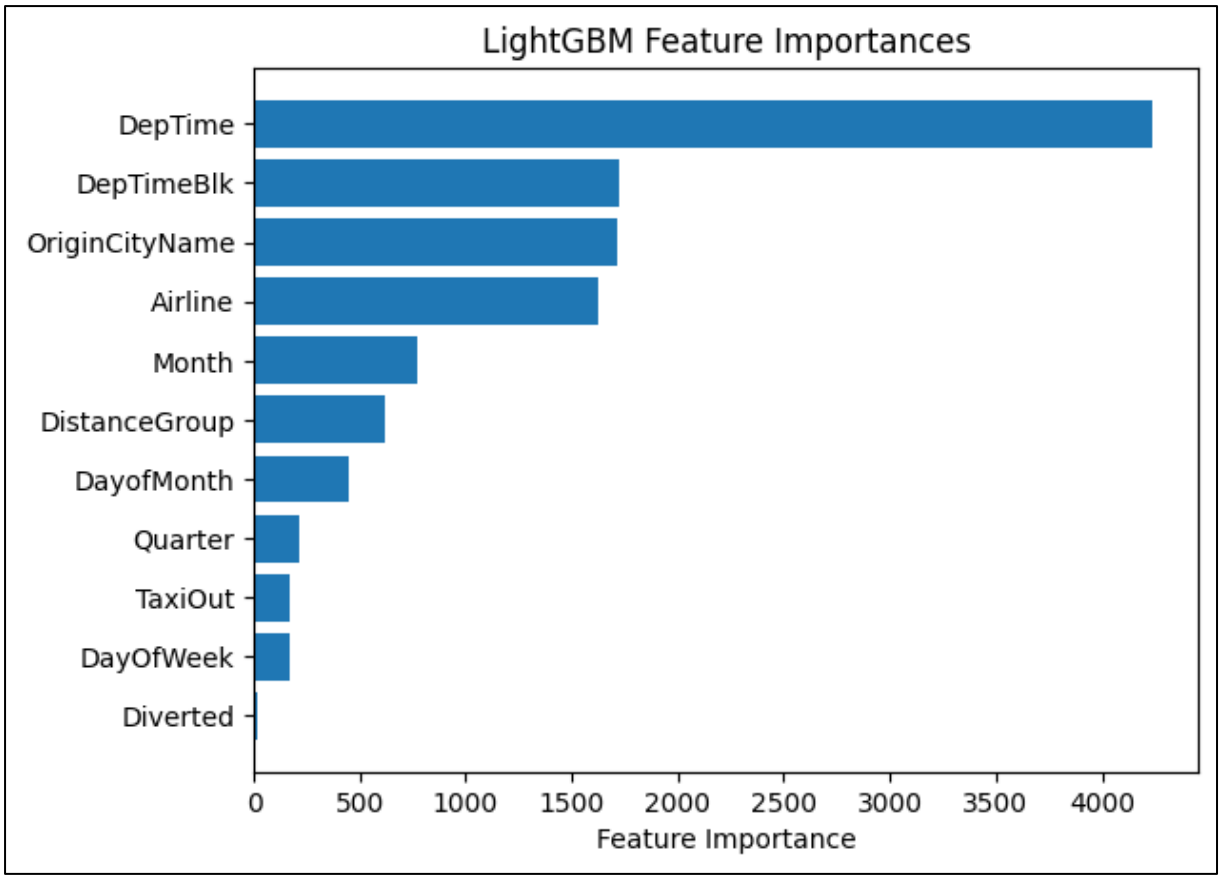


Figure 117: LightGBM Features Importance

"DepTime" has the highest importance score, indicating that the departure time of flights is the most significant predictor in the LightGBM model. The departure time block (which could represent a specific time range during the day) is the second most important feature. The name of the origin city is also a relatively important feature, suggesting that the location from which a flight departs has a significant impact on the model's predictions. The airline operating the flight is next in importance, which implies that the model finds the airline to be a useful predictor.

The significance of "TaxiOut" is quite low. The relevance score for the day of the week is low, suggesting that it is not a good determinant in the model. The lowest relevance score on the graph indicates that whether or not a flight was diverted is the least helpful predictor among the characteristics offered.

5.3.3 Analysis

Features importance analysis in the domain of XGBoost and LightGBM flight delay prediction provides detailed insights into the performance of many characteristics. Both models place significant importance on time-related variables, labeled as "DepTime." It underscores the significance of departure time in predicting flight delays and is closely related to DepTimeBlk. There are several variables that may affect the exact moment a flight takes off, including air traffic control, airport operations, and aircraft readiness. Airlines may detect possible delays and identify trends linked with certain departure times by analysing the actual departure time. Subsequently, they can implement suitable steps to reduce the problem. “DepTimeBlk” also has been listed as a significant impact on flight delays. Delays may occur during certain time periods due to increased air traffic, congestion, or poor weather. This information may be used by airlines and airports to enhance flight schedules, efficiently allocate personnel, and proactively handle predict delays that may occur during certain time periods.

The airline industry is identified as a crucial predictor due to the many daily operations that have a substantial effect on delays. This is a significant aspect since the efficiency and punctuality of different airlines may vary. Given the significance of this characteristic, selecting an airline may substantially impact the probability of encountering a delay. This information may assist passengers and trip planners in making well-informed judgments on airline selection.

Flight delays may be attributed to several factors such as weather patterns, travel demand, and monthly variations that are often associated with seasonal changes. For instance, poor weather may cause greater delays during the winter months, and increased traffic volume during the summer may cause congestion. Through the use of monthly prediction, airports and airlines are able to predict possible patterns of delays, allocate resources correspondingly, and execute initiatives that mitigate disruptions across certain months.

There are factors that can affect the delay of airplanes as well as features that are not so influential. The time takes for an aircraft to taxi from the gate to the runway is a factor that contributes to overall flight efficiency. However, it has a lower importance in this model, longer taxi times can still be indicative of potential delays. The less factors that affect the flight delays

are diverted. A diverted flight indicates that the original flight plan had to be changed, which could be due to various reasons such as weather conditions, air traffic congestion, or mechanical issues. Although they may not have a strong impact on delay prediction compared to other features, it is still worth considering in understanding the overall flight performance.

5.4 Model Deployment

```
# saving the model
import pickle
pickle_out = open("xgbclassifier.pkl", mode = "wb")
pickle.dump(xgb_classifier_tuning, pickle_out)
pickle_out.close()
```

Figure 118: Code of Save Model

The code snippet of Figure 118 is to save a trained XGBoost model (xgb_classifier_tuning) into a binary file which named as “xgbclassifier.pkl” using the pickle module in Python. The saved model file can allow users loaded anytime and used for predictions without the need to retrain the model. This is a common practice in machine learning to persistently store models for deployment or future use. It is a crucial step in the machine learning pipeline to deploy a system. It allows the preservation of a trained model so that it can be reused without the need to retrain it every time.

```
import pickle
import streamlit as st

# loading the trained model
#pickle_in = open('xgbclassifier.pkl', 'rb')
#classifier = pickle.load(pickle_in)
with open('xgbclassifier.pkl', 'rb') as file:
    # Load the pickled model
    classifier = pickle.load(file)
st.cache_data

# defining the function which will make the prediction using the data which the user inputs
def prediction(Airline, OriginCityName, Diverted, DistanceGroup, Quarter,
               Month, DayofMonth, DayOfWeek, TaxiOut, DepTime,DepTimeBlk):

    # Pre-processing user input
    if Airline == "Air Wisconsin Airlines Corp":
        Airline = 0
    elif Airline == "Alaska Airlines Inc.":
        Airline = 1
    elif Airline == "Allegiant Air":
        Airline = 2
    elif Airline == "American Airlines Inc.":
        Airline = 3
    elif Airline == "Capital Cargo International":
        Airline = 4
    elif Airline == "Comair Inc.":
        Airline = 5
    elif Airline == "Commutair Aka Champlain Enterprises, Inc.":
        Airline = 6
    elif Airline == "Delta Air Lines Inc.":
```

```
def main():
    # front end elements of the web page
    html_temp = """
    <div style ="background-color:blue;padding:13px">
      <h1 style ="color:black;text-align:center;">Fligt Delay Prediction System</h1>
    </div>
    """
    # display the front end aspect
    st.markdown(html_temp , unsafe_allow_html = True)

    # following lines create boxes in which user can enter data required to make prediction
    Airline = st.selectbox('Airline',("Air Wisconsin Airlines Corp","Alaska Airlines Inc.","Allegiant Air","American Airlines Inc.",
    "Capital Cargo International","Comair Inc.",
                                      "Commutair Aka Champlain Enterprises, Inc.","Delta Air Lines Inc.","Empire Airlines Inc.",
                                      "Endeavor Air Inc.",
                                      "Envoy Air","Frontier Airlines Inc.","GoJet Airlines LLC d/b/a United Express","Hawaiian Airlines
                                      Inc.","Horizon Air",
                                      "JetBlue Airways", "Mesa Airlines Inc.","Republic Airlines", "SkyWest Airlines Inc.","Southwest
                                      Airlines Co.","Spirit Air Lines","United Air Lines Inc."))
    OriginCityName = st.selectbox('OriginCityName',("Aberdeen, SD","Abilene, TX","Adak Island, AK","Aguadilla, PR","Akron, OH","Alamosa,
    CO ","Albany, GA","Albany, NY","Albuquerque, NM",
                                                    "Alexandria, LA","Allentown/Bethlehem/Easton, PA","Alpena, MI","Amarillo, TX",
                                                    "Anchorage, AK","Appleton, WI","Arcata/Eureka, CA","Asheville, NC",
                                                    "Ashland, WV","Aspen, CO","Atlanta, GA","Atlantic City, NJ","Augusta, GA","Austin,
```

```
    Diverted= st.selectbox('Diverted',("True","False"))
    DistanceGroup = st.selectbox('Distance Group',("1(0-250 Miles)","2(251-500 Miles)","3(501-750 Miles)","4(751-1000 Miles)","5
    (1001-1250 Miles)",
                                                   "6(1251-1500 Miles)","7(1501-1750 Miles)","8(1751-2000 Miles)","9(2001-2250 Miles)",
                                                   "10(2501-2750 Miles)","11(2751-3000 Miles)"))
    Quarter=st.selectbox('Quarter',("1(Jan-Mar)","2(Apr-Jun)","3(Jul-Sep)","4(Oct-Dec)"))
    Month=st.selectbox('Month',("January","February","March","April","May","June","July","August","September","October", "November",
    "December"))
    DayofMonth = st.number_input("Date", min_value=1, max_value=31, value=1, step=1)
    DayOfWeek = st.selectbox('Day of Week',("Monday","Tuesday","Wednesday","Thrusday","Friday","Saturday","Sunday"))
    TaxiOut = st.number_input("Taxi Out(Min)")
    DepTime = st.number_input("Departure Time",min_value=0, max_value=2359, value=0000, step=1)
    DepTimeBlk = st.selectbox('Time Block',("0001-0559","0600-0659","0700-0759","0800-0859","0900-0959","1000-1059","1100-1159",
                                            "1200-1259","1300-1359","1400-1459","1500-1559","1600-1659","1700-1759", "1800-1859",
                                            "1900-1959","2000-2059","2100-2159","2200-2259","2300-2359"))
    result =""

    # when 'Predict' is clicked, make the prediction and store it
    if st.button("Predict"):
        result = prediction(Airline, OriginCityName, Diverted, DistanceGroup, Quarter,Month, DayofMonth, DayOfWeek, TaxiOut, DepTime,
        DepTimeBlk)

        if result == 1:
            st.error('Your Flight is Potentially Delay !!!!')
        else:
           st.success('Your Flight is No Delay.')

if __name__=='__main__':
    main()
```

Figure 119: Code of Web Application Design

In this flight delay prediction application, GitHub and Streamlit were utilized for deployment. The application is designed to predict whether a flight will be delayed or not based on user-input data. The predictive model, saved as “xgbclassifier.pkl”, is loaded using the pickle library.

The code of Figure 119 is to develop a Streamlit web application for predicting flight delays using a pre-trained XGBoost classifier stored in a pickled file (“xgbclassifier.pkl”). Users can engage with the application by providing specific details about a flight, including the airline, origin city, whether the flight was diverted, distance group, quarter, month, day of

the month, day of the week, taxi-out time, departure time, and departure time block. The Python script along with the pickled model file is saved on a GitHub repository. This allows for easy access and retrieval of the files for deployment. The Streamlit Cloud platform is then employed to host the application. Streamlit Cloud facilitates the seamless deployment of Streamlit apps directly from a GitHub repository.

After that, Streamlit Cloud is accessed, and the application is deployed by connecting it to the GitHub repository. This connection allows the application receiving updates automatically in response to modifications made to the GitHub repository. Users can interact with the mounted app by entering information about flights and getting predictions about possible delays. It is easy and quick to make machine learning apps accessible to more people when utilizing GitHub and Streamlit together.

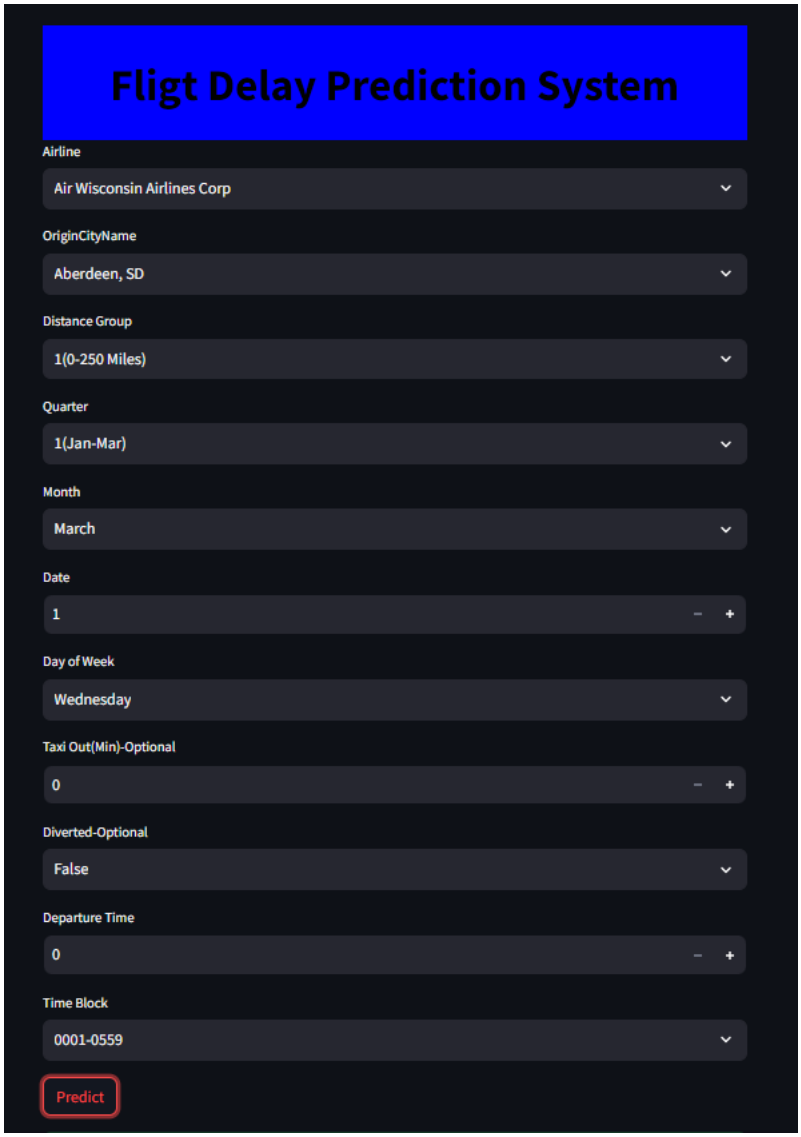


Figure 120: User Interface of Flight Delay Prediction Application

The user interface of flight delay prediction application (Figure 120) is designed to facilitate user interaction through a set of dropdown menus and input boxes.Users are provided with a variety of alternatives for certain features through the usage of dropdown menus. There are some selections that has been insert in the code of Figure 119 for each dropdown menus. For instance, the “Airline” dropdown menu allows users to choose from a list of airlines, and similarly, the “OriginCityName” dropdown provides options for selecting the origin city. Furthermore, numerical input boxes are incorporated for more precise input. Users can input the day of the month using a numerical input box, while the “Taxi Out Time” and “Departure Time” require numerical inputs for time. The taxi out and diverted will be provided the explanation of the word. Hence, some of the users are passenger, they may not be familiar with

those word. Those two columns set as optional input which user does not have this kind of information.

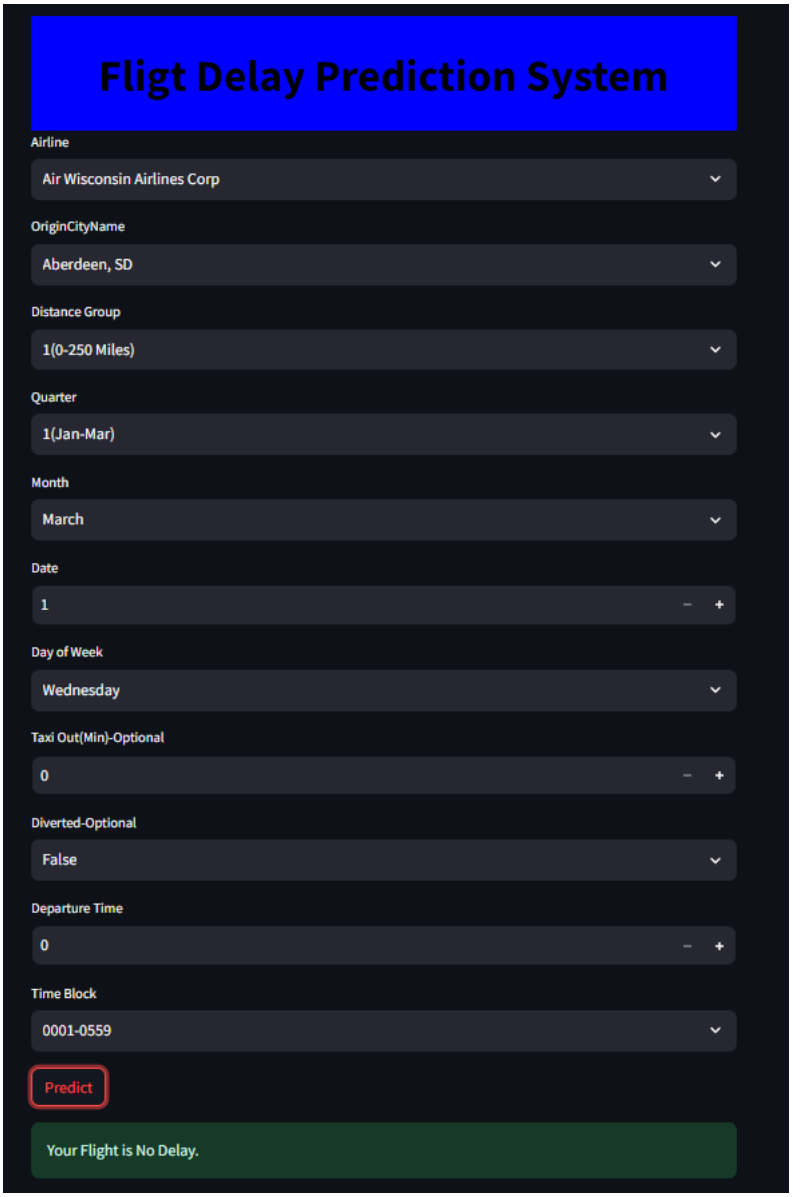


Figure 121: Delay Detected Example1

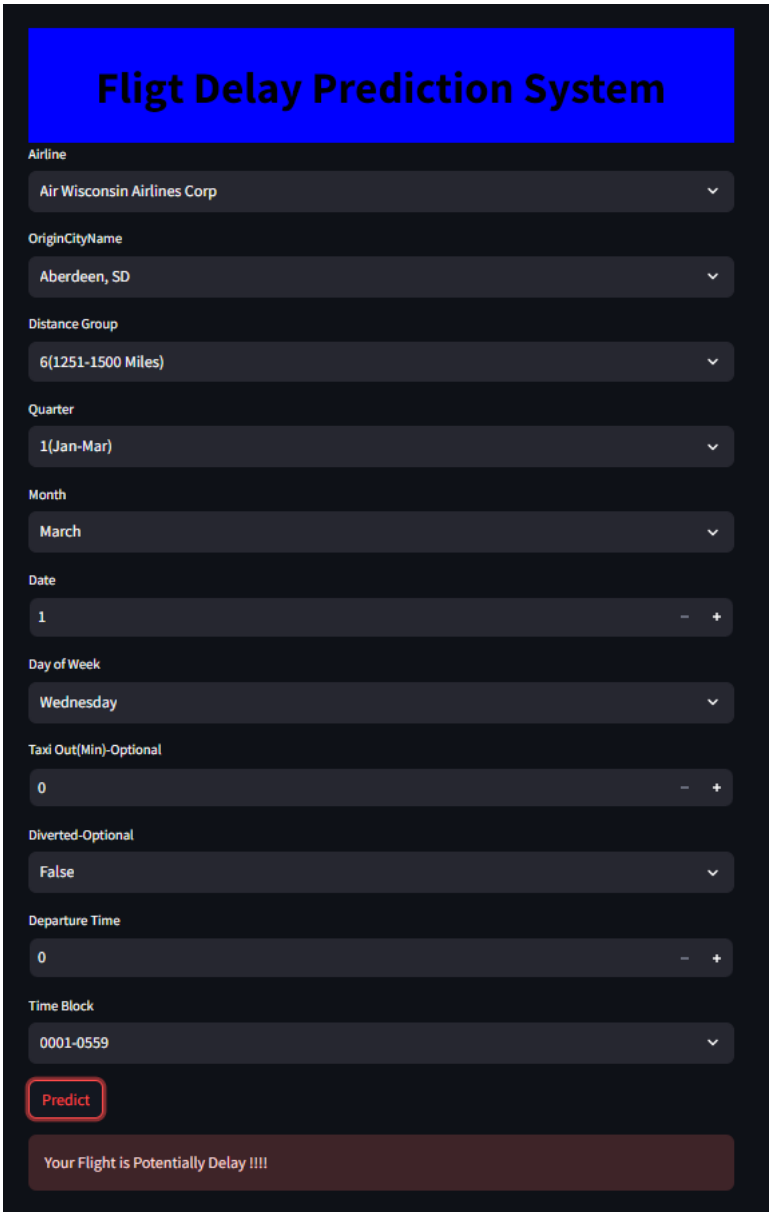


Figure 122: Delay Detected Example2

After the users have provided all of the required information, they are able to begin the process of prediction by pressing the “Predict” button. The app then makes use of the XGBoost classifier that has been pre-trained in order to interpret the input data and presents the projected result, which indicates whether or not the flight is expected to be delayed. . Figure 121 has detected the flight does not delay. The system displays a green box that inform the user the flight does not delay. If the flight detected as delay, the system pops up a red box that inform user the flight is potentially delay. User can access the system through this LINK.

5.5 Summary

The research presents the findings and has a thorough evaluation of the prediction models effectiveness in Chapter 5. The base models initial accuracy is found to be between 80 and 81 %, with precision for Class 1 (delay) consistently lower than that for Class 0 (no delay), suggesting that effectively anticipating delays is a challenging task. Additionally, tuning improves Class 1 accuracy, reducing the difference between the two classes. The use of learning curves facilitates a more detailed evaluation of model performance by offering insightful information about possible overfitting scenarios. Hyperparameter tuning executed through grid search for leads to improved accuracy in all the models. All the model's accuracy range improved to 81%-82%. However, the discussion highlights overfitting in the Random Forest model, despite its commendable accuracy. The important component of the features significance study reveals that although variables like taxi out and diverted flights have comparatively less of an influence, departure time, time block, and airline are critical in predicting flight delays. After the comparison and discussion, XGBoost has been selected as the most effective model post-tuning, showcasing superior performance.

CHAPTER 6: CONCLUSION

6.1 Critical Evaluation

6.1.1 Achievement

The project has accomplished significant success in achieving its main objectives, which primarily revolve around comparing machine learning algorithms, constructing reliable prediction models, offering valuable insights for enhancement, and tackling challenges inherent in the domain of flight delay prediction. The examination of the usefulness of machine learning algorithms, such as XGBoost, Random Forest, and Decision Trees, in predicting flight delays was conducted with great focus on accuracy. By doing so, the initiative not only made a valuable contribution to the scholarly discussion, but also provided tangible benefits for the stakeholders of the aviation industry.

The study demonstrated a practical use of machine learning methods by using historical flight data to develop prediction models. The use of classification techniques such as XGBoost resulted in the models demonstrating a notable degree of accuracy in differentiating between aircraft that were delayed and those that were on-time This accomplishment provides airlines and airports with a potent tool, enabling them to actively monitor operations, optimize resources, and improve overall efficiency.

The project's emphasis on methodologies and recommendations for improvement reflects its dedication to not just developing predictive models but also improving them for practical implementations. The study has provided valuable information on improving the dependability and effectiveness of machine learning-based flight delay prediction models. This knowledge may be immediately used by airlines, airports, and other relevant parties. This feature amplifies the project's influence by connecting the divide between theoretical research and practical solutions.

Lastly, the project's achievement is overcome the constraints related to the study scope, such as different data sources, limited access, and worries about data quality, demonstrates its resilience and flexibility. The recognition and efficient management of difficulties concerning the acquisition of reliable data, handling datasets with uneven distribution, and resolving technological problems in real-time forecasts enhance the project's credibility and suitability in the ever-changing aviation industry.

6.1.2 Contribution of the project towards community/ industries

The research on flight delay prediction significantly contributes to enhancing operational efficiency and overall performance within the aviation industry, thereby positively impacting both the industry and the wider community. The developed prediction models serve as powerful tools for airlines and airports, offering strategic insights that optimize operations. This optimization not only leads to a reduction in operational costs but also improves crew management and air traffic control. By enabling proactive decision-making, the research empowers stakeholders to navigate complex scenarios with greater ease.

In terms of customer satisfaction, the ability to predict flight delays facilitates effective communication with passengers. This, in turn, allows travelers to plan and adjust their schedules accordingly, mitigating the inconvenience caused by unforeseen delays. The emphasis on customer satisfaction contributes to building passenger loyalty, fostering a positive travel experience, and aligning with the aviation industry's commitment to providing reliable and satisfactory services to the community.

Addressing the financial impact of flight delays is a tangible benefit derived from this research. The project's insights translate into cost savings for airlines and improved profitability for the aviation industry. By strategically managing resources, optimizing routes, and minimizing disruptions, airlines can achieve better financial outcomes, thereby ensuring the sustainability and growth of the industry.

Furthermore, the research provides decision support tools that offer valuable insights for stakeholders in the aviation sector. From route planning to capacity management and resource allocation, decision-makers now have access to data-driven solutions. This not only streamlines operational processes but also contributes to the long-term planning and sustainable development of the aviation industry.

6.1.3 Strength

This research’s wide and diverse approach to aviation flight delay prediction is its strength. The study is notable for various factors that enhance its effect and relevance. The research stands out in several key aspects, each contributing significantly to the overall impact and significance of the study. The main highlight is the thorough comparison of several machine learning methods, such as Decision Trees, LightGBM, Random Forest, and XGBoost.

This comparative study contributes to the growth of knowledge in the industry by helping to identify the most successful approaches for anticipating flight delays and by offering insightful information for future research and practical applications.

Furthermore, the study extends beyond theoretical investigation by putting practical solutions into practice with the development of prediction models. Built using actual data, these models provide useful tools for all parties involved in the aviation sector, including airlines, airports, and travelers. This practical approach guarantees that the research is relevant and immediately applicable in solving industrial concerns, rather than being restricted to academic domains.

The research further exhibits honesty and trustworthiness by clearly recognizing and resolving issues related to flight delay prediction. Problems like unbalanced datasets and technological difficulties with real-time predictions are addressed, which enhances the validity of the results and offers helpful advice for future researchers who encounter such difficulties.

The usefulness of the study is further enhanced by the user-focused interface, which takes the shape of an application that predicts flight delays. This interface, which was designed to be user-friendly, enables a wide variety of stakeholders to engage with and get advantages from the prediction models that have been established. As a result, the research is accessible and has an influence on a variety of user groups.

Finally, the study adds a great deal to the body of knowledge by offering fresh perspectives and useful data to airports, airlines, and other stakeholders. This addition strengthens the research's long-lasting effects by promoting continuing industry practice improvement and broadening the general understanding of flight delay prediction.

6.2 Limitations

While this study provides valuable insights into flight delay prediction, it is not without its limitations. First, the research confronts significant challenges emanating from computational resources and hardware constraints. The dataset included 6.13 million rows imposes substantial demands on computing power essential for training intricate machine learning models, including XGBoost, Random Forest, LightGBM, and Decision Trees. This limitation results from the resource-intensive processing of large datasets, which may place a burden on the hardware and computing resources that are available. When the computing

infrastructure is not strong, model training might take a long time, which slows the development of increasingly complex models.

In addition, the study faces significant hardware limitations, mostly related to memory and processing speed. Some devices may not be able to handle the large number of data relevant to flight delay prediction. This real-world limitation forces a careful evaluation of the trade-off between computing efficiency and model complexity when choosing and implementing machine learning methods. As such, careful consideration of the deployment device's constraints becomes crucial.

Since, the study only included data from 2021, it has some important limits related to its time frame. Patterns and factors that affect flight delays often change over time. Limiting the information to a single year could mean missing out on changing trends and seasonal effects. At the same time, the fact that the study only at the United States limits its regional scope, the factors that cause flight delays may be different in different areas. This makes it harder for the prediction models to work in a wider setting, which shows how important it is to be careful when interpreting and thinking about possible changes in how flight delays happen.

Another acknowledged limitation is the lack of meteorological conditions in the dataset. The influence of weather on flight delays is well acknowledged and excluding it from the prediction system undermines its accuracy. The complicated interaction between weather-related events and flight operations introduces a level of intricacy, and the absence of this crucial information results in an insufficient depiction of the many elements that cause flight delays. It is essential to address these limitations in order to ensure the robustness and applicability of the findings in flight delay analysis.

6.3 Recommendations

A number of recommendations are made in considering the primary limitations that this flight delay prediction study experienced in order to improve the validity and relevance of further research in this area. Initially, it is critical to solve computing limitations in order to investigate more complex models and handle large datasets. The training of complex machine learning models, such as XGBoost, Random Forest, LightGBM, and Decision Trees, requires highly competent computer resources, which researchers should aim to achieve. It is possible to alleviate the burden on local hardware and accelerate model training operations by partnering with organizations or cloud computing services.

Future study efforts should take into consideration a more comprehensive temporal scope and many years of data to precisely capture changing patterns and seasonality impacts to solve temporal and geographical restrictions. Furthermore, broadening the scope of the research to include other geographical areas and aviation settings outside of the US will enhance our comprehension of the factors that influence flight delays. By avoiding the loss of generalizability seen in exclusive, one-year, single-country research, this method helps to increase the prediction models' wider application.

It is critical that future databases have complete weather data, given the significant impact that weather conditions have on flight delays. While weather-related events and flight operations interact so intricately, this feature guarantees a more comprehensive picture of the many elements leading to flight delays. This restriction may be overcome by working with weather organizations or adding trustworthy weather APIs to the dataset, which will improve the forecasting model's precision and applicability.

Overall, the main suggestions for further study in flight delay prediction are to minimize computing difficulties, prioritize algorithmic efficiency, broaden the scope in terms of both time and space, and include detailed meteorological data. By addressing these suggestions, this area will improve and more predictive models with substantial ramifications for the aviation sector will be developed, which will be more reliable, accurate, and broadly applicable, among other things.

References

Alla, H., Moumoun, L., & Balouki, Y. (2021). A Multilayer Perceptron Neural Network with Selective-Data Training for Flight Arrival Delay Prediction. *Scientific Programming*, 1. doi:10.1155/2021/5558918

Anupkumar, A. (n.d.). *INVESTIGATING THE COSTS AND ECONOMIC IMPACT OF FLIGHT DELAYS IN THE AVIATION INDUSTRY AND THE POTENTIAL STRATEGIES FOR REDUCTION.* Retrieved from CSUSB ScholarWorks: https://scholarworks.lib.csusb.edu/etd/1653

Anusha, A., Kumar, B. R., Reddy, D. N., Kumar, D. S., & Nayak, K. S. (2022). PREDICTION OF FLIGHT DELAY USING MACHINE LEARNING. *International Research Journal of Modernization in Engineering Technology and Science*, 2486-2491.

Bardach, M., Gringinger, E., Schrefl, M., & Schuetz, C. G. (2020). Predicting Flight Delay Risk Using a Random Forest Classifier Based on Air Traffic Scenarios and Environmental Conditions. doi:10.1109/dasc50938.2020.9256474

Brownlee, J. (2020, August 27). *Tune hyperparameters for classification machine learning algorithms*. Retrieved from MachineLearningMastery.com: https://machinelearningmastery.com/hyperparameters-for-classification-machine-learning-algorithms/

Busson, T. (2020, September 15). *Why is My Flight Delayed? The 20 Main Reasons for Flight Delays*. Retrieved from The ClaimCompass Blog: https://www.claimcompass.eu/blog/why-is-my-flight-delayed/#howtofindthereasonofthedelay

CAA. (2014). *Passenger experiences during flight disruption Consumer research report.* Civil Aviation Authority.

Canesche, M., Braganca, L., Neto, O. P., Nacif, J. A., & Ferreira, R. (2021). Google Colab CAD4U: Hands-On Cloud Laboratories for Digital Design. *IEEE Xplore*. doi:10.1109/iscas51556.2021.9401151

Carneiro, T., Nobrega, R. V., Nepomuceno, T., Bian, G.-B., Albuquerque, V. H., & Gupta, D. (2018). Performance analysis of Google Colaboratory as a tool for accelerating deep learning applications. *IEEE Access*, 61677-61685. doi:10.1109/access.2018.2874767

Dožić, S. (2019). Multi-criteria decision making methods: Application in the aviation industry. *Journal of Air Transport Management*, 101683. doi:10.1016/j.jairtraman.2019.101683

Dridi, S. (2021). Supervised learning-a systematic literature review. doi:10.31219/osf.io/tysr4

Eftymiou, M., Njoya, E. T., Lo, P. L., Παπαθεοδώρου, A., & Randall, D. (2018). The Impact of Delays on Customers' Satisfaction: an Empirical Analysis of the British Airways On-Time Performance at Heathrow Airport. *Journal of Aerospace Technology and Management*. doi:10.5028/jatm.v11.977

Filho, M. (2023, September 19). *How to get feature importance in LightGBM (Python example).* Retrieved from Forecastegy: https://forecastegy.com/posts/feature-importance-lightgbm-python-example/

Firas, O. (2023). A combination of SEMMA & CRISP-DM models for effectively handling big data using formal concept analysis based knowledge discovery: A data mining approach. *World Journal of Advanced Engineering Technology and Sciences*, 9-14. doi:10.30574/wjaets.2023.8.1.0147

Fleischer, Y., Biehler, R., & Schulte, C. (2022). TEACHING AND LEARNING DATA-DRIVEN MACHINE LEARNING WITH EDUCATIONALLY DESIGNED JUPYTER NOTEBOOKS. *Statistics Education Research Journal*, 7. doi:10.52041/serj.v21i2.61

Gui, G., Zhou, Z., Wang, J., Liu, F., & Sun, J. (2020). Machine Learning Aided Air Traffic Flow Analysis Based On Aviation Big Data. *IEEE Transactions on Vehicular Technology*, 4817-4826. doi:10.1109/tvt.2020.2981959

Guo, Z., Yu, B., Hao, M., Wang, W., & Jiang, Y. (2021). A novel hybrid method for flight departure delay prediction using Random Forest Regression and Maximal Information Coefficient. *Aerospace Science and Technology*, 106822. doi:10.1016/j.ast.2021.106822

Harper, A., & Monks, T. (2023). A Framework to Share Healthcare Simulations on the Web Using Free and Open Source Tools and Python. *Proceedings of the Operational Research Society Simulation Workshop 2023*. doi:10.36819/sw23.030

Hatıpoğlu, I., Tosun, O., & Tosun, N. (2022). FLIGHT DELAY PREDICTION BASED WITH MACHINE LEARNING. *LogForum*, 97-107. doi:10.17270/j.log.2022.655

Hotz, N. (2023, January 19). *What is CRISP DM? - Data Science Process Alliance*. Retrieved from Data Science Process Alliance: https://www.datascience-pm.com/crisp-dm-2/

Ibrahim, M. (2023, October). *Weights & biases*. Retrieved from W&B: https://wandb.ai/mostafaibrahim17/ml-articles/reports/A-Deep-Dive-Into-Learning-Curves-in-Machine-Learning--Vmlldzo0NjA1ODY0

Importance of decision tree hyperparameters on generalization — Scikit-learn course. (n.d.). Retrieved from Github: https://inria.github.io/scikit-learn-mooc/python_scripts/trees_hyperparameters.html

Jiang, Y., Tran, T. H., & Williams, L. (2023). Machine learning and mixed reality for smart aviation: Applications and challenges. *Journal of Air Transport Management*, 102437. doi:10.1016/j.jairtraman.2023.102437

Kauristie, K., Andries, J., Beck, P., Berdermann, J., Berghmans, D., Cesaroni, C., . . . Laitine, T. V. (2021). Space Weather Services for Civil Aviation—Challenges and Solutions. *Remote Sensing*, 3685. doi:10.3390/rs13183685

Keita, Z. (2022, September 21). *Classification in Machine Learning: An Introduction*. Retrieved from DataCamp: https://www.datacamp.com/blog/classification-machine-learning

Khaksar, H., & Sheikholeslami, A. (2017). Airline delay prediction by machine learning algorithms. *Scientia Iranica*, 0. doi:10.24200/sci.2017.20020

Kumar, S., & Zymbler, M. (2019). A machine learning approach to analyze customer satisfaction from airline tweets. *Journal of Big Data*. doi:10.1186/s40537-019-0224-1

Kuşkapan, E., Sahraei, M. A., & Codur, M. Y. (2022). Classification of aviation accidents using data mining algorithms. *Balkan journal of electrical & computer engineering*, 10-15. doi:10.17694/bajece.793368

Lambelho, M., Mitici, M., Pickup, S., & Marsden, A. (2020). Assessing strategic flight schedules at an airport using machine learning-based flight delay and cancellation

predictions. *Journal of Air Transport Management*, 101737. doi:10.1016/j.jairtraman.2019.101737

Li, Q., & Jing, R. (2021). Generation and prediction of flight delays in air transport. *Iet Intelligent Transport Systems*, 740-753. doi:10.1049/itr2.12057

Li, S., & Zhang, X. (2019). Research on orthopedic auxiliary classification and prediction model based on XGBoost algorithm. *Neural Computing and ApplicationsNeural Computing and Applications*, 1971-1979. doi:10.1007/s00521-019-04378-4

LightGBM 4.2.0.99 documentation. (n.d.). Retrieved from Python-package Introduction: https://lightgbm.readthedocs.io/en/latest/Python-Intro.html

Manasa, C., & Velayutham, P. (2023). PREDICTING FLIGHT DELAYS WITH ERROR CALCULATION USING MACHINE LEARNED CLASSIFIERS . *Journal of Engineering Sciences*, 93-97.

May, R. M., Goebbert, K. H., Thielen, J., Leeman, J. R., Camron, M. D., Bruick, Z. S., . . . Marsh, P. T. (2022). MetPY: a meteorological Python library for data analysis and visualization. *Bulletin of the American Meteorological Society*, E2273-E2284. doi:10.1175/bams-d-21-0125.1

Naeem, S., Ali, A., Anam, S., & Ahmed, M. M. (2023). An Unsupervised Machine Learning Algorithms: Comprehensive Review. *International Journal of Computing and Digital Systems*, 911-921. doi:10.12785/ijcds/130172

Naik, P., Naik, G., & M.B.Patil, M. (2022). *Conceptualizing Python in Google COLAB.*

Nogueira, R. P., Melício, R., Valério, D., & Santos, L. F. (2023). Learning methods and predictive modeling to identify failure by human factors in the aviation industry. *Applied sciences*, 4069. doi:10.3390/app13064069

Odhiambo, J. O., Onsongo, W. M., & Osman, S. (2020). An Analytical Comparison between Python Vs R Programming Languages Which one is the best for Machine Learning and Deep Learning?

Oliveira, M. d., Eufraasio, A. B., Guterres, M. X., Murça, M. C., & Gomes, R. e. (2021). Analysis of airport weather impact on on-time performance of arrival flights for the

Brazilian domestic air transportation system. *Journal of Air Transport Management*, 101974. doi:10.1016/j.jairtraman.2020.101974

Parameters Tuning — LightGBM 4.2.0.99 documentation. (n.d.). Retrieved from LightGBM: https://lightgbm.readthedocs.io/en/latest/Parameters-Tuning.html#for-better-accuracy

Ray, S. (2019). A Quick Review of Machine Learning Algorithms. *IEEE*.

Rodríguez-Sanz, Á., Cano, J., & Fernández, B. R. (2021). Impact of weather conditions on airport arrival delay and throughput. *IOP conference series*, 012107. doi:10.1088/1757-899x/1024/1/012107

Saabith, A., T.Vinothraj, & MMM.Fareez. (2020). POPULAR PYTHON LIBRARIES AND THEIR APPLICATION DOMAINS . *International Journal of Advance Engineering and Research Development* .

Saha, S. (2023, August 8). *XGBoost vs LightGBM: How Are They Different*. Retrieved from neptune.ai: https://neptune.ai/blog/xgboost-vs-lightgbm

Santhanam, R., Uzir, N., Raman, S., & Banerjee, S. (2017). Experimenting XGBoost Algorithm for Prediction and Classification of Different Datasets. *ResearchGate*, 651-662.

Sarker, I. H. (2021). Machine learning: algorithms, Real-World applications and research directions. *SN Computer Science*. doi:10.1007/s42979-021-00592-x

Satish, N., lPremkumar, M., Karthikeyan, A., Senthilkumar, V. M., Sathyaseelan, K., Kalaiyarasi, V., & Saranya, E. (2022). Flight Delay Prediction Using Machine Learning. *Neuroquantology*, 2152-2155. doi:10.48047/Nq.2022.20.17.Nq880278

Setiawan, A., Efendi, E., Mubarok, A., Prasetyo, K. T., & Wibowo, U. L. (2023). Check On-Time performance of domestic airlines using random Forest machine learning analysis. *Technium Social Sciences Journal*, 570-583. doi:10.47577/tssj.v43i1.8792

Shah, F. T., Syed, Z., Imam, A., & Raza, A. (2020). The impact of airline service quality on passengers' behavioral intentions using passenger satisfaction as a mediator. *Journal of Air Transport Management*, 101815. doi:10.1016/j.jairtraman.2020.101815

Shekar, B. H., & Dagnew, G. (2019). Grid Search-Based Hyperparameter Tuning and Classification of microarray cancer data. *2019 Second International Conference on Advanced Computational and Communication Paradigms (ICACCP)*. doi:10.1109/icaccp.2019.8882943

Shetty, S. H., Shetty, S., Singh, C., & Rao, A. (2022). Supervised Machine Learning: : Algorithms and Applications. *Fundamentals and Methods of Machine and Deep Learning*, 1-16. doi:10.1002/9781119821908.ch1

Silling, U. (2019). Aviation of the Future: What needs to change to get aviation fit for the Twenty-First Century. In *IntechOpen eBooks.*

Smajić, A., Grandits, M., & Ecker, G. (2022). Using Jupyter Notebooks for re-training machine learning models. *Journal of Cheminformatics*. doi:10.1186/s13321-022-00635-2

Song, C., Guo, J., & Zhuang, J. (2020). Analyzing passengers' emotions following flight delays- a 2011–2019 case study on SKYTRAX comments. *Journal of Air Transport Management*, 101903. doi:10.1016/j.jairtraman.2020.101903

Song, Y.-y., & Lu, Y. (2015). Decision tree methods: applications for classification and prediction. *Shanghai Arch Psychiatry*, 130-135.

Speiser, J. L., Miller, M. E., Tooze, J. A., & Ip, E. H. (2019). A comparison of random forest variable selection methods for classification prediction modeling. *Expert Systems with Applications*, 93-101. doi:10.1016/j.eswa.2019.05.028

Syafei, R. M., & Efrilianda, D. A. (2023). Machine Learning Model Using Extreme Gradient Boosting (XGBoost) Feature Importance and Light Gradient Boosting Machine (LightGBM) to Improve Accurate Prediction of Bankruptcy. *Recursive Journal of Informatics*, 64-72. doi:10.15294/rji.v1i2.71229

Tang, Y. (2021). Airline Flight Delay Prediction Using Machine Learning Models. *ISBN*, 151-154. doi:10.1145/3497701.3497725

Thakur, U. (2023, March 4). *Top 6 Use Cases of Machine Learning in the Aviation Industry that you should know about*. Retrieved from Linkedin: https://www.linkedin.com/pulse/top-6-use-cases-machine-learning-aviation-industry-you-umair-thakur/

Wang, Z., Liao, C., Hang, X., Li, L., Delahaye, D., & Hansen, M. (2022). Distribution Prediction of Strategic Flight Delays via Machine Learning Methods. *Sustainability*, 15180. doi:10.3390/su142215180

xgboost 2.0.3 documentation. (n.d.). Retrieved from XGBoost Documentation: https://xgboost.readthedocs.io/en/stable/

Ye, B., Liu, B., Tian, Y., & Wan, L. (2020). A methodology for predicting aggregate flight departure delays in airports based on supervised learning. *Sustainability*, 2749. doi:10.3390/su12072749

Yilmaz, İ. G., Kartal, E., Özen, Z., & Gülseçen, S. (2021). Prediction of Fuel Tankering in Aviation Industry with Machine Learning Algorithms. *Journal of Aeronautics and Space Technologies*, 19-34.

Yiu, C. Y., Ng, K. M., Kwok, K. C., Lee, W. T., & Mo, H. T. (2021). Flight delay predictions and the study of its causal factors using machine. *2021 IEEE 3rd International Conference on Civil Aviation Safety and Information Technology (ICCASIT)*. doi:10.1109/iccasit53235.2021.9633571

Yu, B., Guo, Z., Asian, S., Wang, H., & Chen, G. (2019). Flight delay prediction for commercial air transport: A deep learning approach. *Transportation Research Part E-logistics and Transportation Review*, 203-221. doi:10.1016/j.tre.2019.03.013

Zámková, M., Rojík, S., Prokop, M., & Stolín, R. (2022). Factors Affecting the International Flight Delays and Their Impact on Airline Operation and Management and Passenger Compensations Fees in Air Transport Industry: Case Study of a Selected Airlines in Europe. *Sustainability*, 14763. doi:10.3390/su142214763

Zoutendijk, M., & Mitici, M. (2021). Probabilistic Flight Delay Predictions Using Machine Learning and Applications to the Flight-to-Gate Assignment Problem. *Aerospace*, 152. doi:10.3390/aerospace8060152

Appendices

Appendix A: PPF- Machine Learning-Based Flight Delay Prediction

DRAFT PROJECT PROPOSAL FORM

Proposal ID :

Supervisor :
1) Assoc. Prof. Dr. Nirase Fathima Abubacker
2) Ms. Minnu Helen Joseph
3) Dr. Vazeerudeen Abdul Hamed
4) Mr. Zailan Arabee Abdul Salam
5) Assoc. Prof. Dr. Imran Medi (Mr.)

Student Name : Soe Chou Kit

Student No : TP058323

Email Address : TP058323@mail.apu.edu.my & choukitsoe@gmail.com

Programme Name : Bachelor in Computer Science (DA)

Title of project : Machine Learning-Based Flight Delay Prediction

Please record which module(s) your topic is related to:

Data Mining and Predictive Modelling(CT119-3-2-DMPM)
Research Methods for Computing and Technology (CT098-3-2-RMCT)
Data Management (CT075-3-2-DTM)

1. Introduction

Air travel has become an essential part of our modern society, connecting everyone and businesses around the world. However, flight delays can be a major inconvenience for passengers, leading to missed connections, rescheduling complications, and wasted time (Yu et al., 2019). Airlines and airports are constantly seeking ways to improve their operations and minimize delays, and one promising approach is the application of machine learning techniques.

Machine learning has revolutionized various industries by enabling predictive modeling and accurate decision-making based on large and complex datasets. In the field of aviation, machine learning can be utilize to examine historical flight data, air traffic patterns, weather conditions and many more factors to enhance the prediction of flight delays. By doing so, airlines can proactively manage their operations, allocate resources efficiently, and provide timely updates to passengers (Herbas, 2021).

This research is to examine using of machine learning in predicting flight delays. We will delve into the challenges associated with flight delay prediction and highlight the potential benefits it offers to the aviation industry. Additionally, we will discuss the various data sources that can be utilized, such as flight data records and airport infrastructure data, to build accurate predictive models.

The implications of accurate flight delay prediction are far-reaching. Passengers can be informed in advance about potential delays, allowing them to adjust their travel plans accordingly. Airlines can optimize their crew scheduling, gate assignments, and maintenance operations, leading to improved efficiency and customer satisfaction. Airports can enhance their capacity planning and better manage the flow of passengers through their terminals.

This paper will examine various machine learning algorithms that have been utilized for the prediction of flight delays. We will assess the characteristics and constraints of each approach and deliberate on the significance of feature engineering and data preprocessing in attaining precise predictions.

2. Problem Statement

1. **Disrupted travel plans and inconveniences for passengers**

Flight delays can create considerable disturbances and inconveniences for travelers, leading to missed connecting flights, crucial engagements, significant events, and prolonged waiting periods at airports. Such delays can have adverse consequences, including financial setbacks, missed opportunities, and heightened stress levels. Passengers may face monetary losses due to rescheduling expenses or missed business and personal engagements (CAA, 2014). Moreover, the added stress and frustration caused by extended waiting times and disrupted travel plans can impact the overall travel experience and well-being of individuals.

2. **Economic impact on airlines and the aviation industry**

Flight delays carry significant economic ramifications for both airlines and the wider aviation sector (Anupkumar, n.d.). They impose augmented operational expenses, encompassing crew overtime, escalated fuel consumption, and heightened aircraft maintenance costs, thereby exerting pressure on airline profitability. Additionally, delays have the potential to adversely impact customer satisfaction, tarnish brand reputation, and diminish customer loyalty. These repercussions can contribute to reduced passenger demand and subsequent revenue decline. Therefore, the financial consequences of flight delays extend beyond immediate operational costs, encompassing long-term effects on customer perception and business sustainability within the aviation industry.

3. **Decreased operational efficiency for airlines and airports**

Flight delays can create a domino effect, impacting the smooth functioning of both airlines and airports. The resulting congestion in terminals, prolonged aircraft turnaround times, and disrupted crew schedules lead to inefficiencies and underutilization of resources. This, in turn, leads to heightened expenses for the airline, decreased customer satisfaction, and overall operational challenges. Increased costs arise from compensations,

accommodation, and meals, while reduced customer satisfaction stems from inconvenience and disrupted travel plans (Peterson et al., 2013). These operational inefficiencies highlight the significance of timely and efficient operations, as delays can have wide-ranging implications throughout the airline and airport ecosystem.

3. Project Aim and Objectives

The aim is to explore the application of machine learning techniques in predicting flight delays. This study is to make a valuable contribution to the progress of the aviation sector. It aims to achieve this by harnessing the power of machine learning algorithms to augment the precision of flight delay forecasts.

The objectives of this project:

1. To examine and assess the obstacles linked to predicting flight delays within the aviation sector.
2. To examine and compare different machine learning algorithms and approaches for predict flight delays.
3. To investigate the availability and integration of relevant data sources, such as historical flight data, weather information, and airport infrastructure data, for building predictive models.
4. To provide strategies and provide recommendations for enhancing the dependability and effectiveness of machine learning-based flight delay prediction models.
5. To make a valuable contribution to the existing knowledge base and offer insights that can aid airlines, airports, and relevant stakeholders in making informed decisions and optimizing their operations to mitigate the occurrence of flight delays.

4. Literature Review

In order to assess the selected subject, it is necessary to present the pertinent information, facts, and prior investigations to explore and substantiate the topic. Existing literature on the flight delay predominantly focuses on broad discussions, business processes, technical aspects, and specific domains. Gathering and analyzing all the essential data aims to address the limitations identified in previous studies, enabling the researcher to propose innovative solutions to the identified problems.

According to Yu et al.'s (2019b) research, flight delays have become prevalent throughout the aviation sector. Accurate prediction of delays is crucial to alleviate airport. The researchers utilized a novel deep belief network (DBN) approach to uncover intricate patterns in flight delays. They introduced and examined new influential factors like air route conditions and airport congestion levels, based on flight and passenger volumes. The findings emphasized the significant relevance of these factors in enhancing the accuracy of flight delay prediction. However, this study was limited to a case study, as it didn't incorporate arrival and international flight data due to data unavailability. Further research should be conducted to improve the model's accuracy by considering both domestic and international flight information for arrivals and departures.

In their research, Hatıpoğlu et al. (2022) analyzed a comprehensive dataset consisting of all flights (18,148 flights) from an international airline over the course of a year. The study revealed that flights can be predicted with a high level of accuracy. The researchers employed three machine learning techniques: XGBoost, LightGBM, and CatBoost. Based on various performance metrics such as recall, Cohen's Kappa, F2 score and accuracy, LightGBM emerged as the most suitable technique for the given scenario. Although XGBoost exhibited slightly lower performance, it should be noted that the results were still quite good, hovering around 0.90. Although CatBoost yielded the least favorable outcomes, it is worth emphasizing that even these relatively lower results remained within an acceptable range. Consequently, the recommended model from their analysis was LightGBM (Hatıpoğlu et al., 2022).

Chen and Liu (2019) proposed a novel machine learning approach for predicting air traffic delays in their research. They created a framework that integrated random forest classification with an estimated delay model. Their purpose was to enhance the precision of predictions by employing an optimal procedure for selecting features. It was found that departure delays and

late arrival aircraft delays were key factors in accurately predicting flight delays. By considering the initial departure delay, the chain model demonstrated the ability to predict subsequent delays in the same aircraft trip. This methodology showcased improved accuracy and practicality when applied to predicting delays in daily air traffic operations (Chen & Liu, 2019).

Based on Bojia et al. (2020), this research conducted a new method using supervised learning techniques to predict aggregate flight departure delays in airports. With the intention of enhancing both prediction accuracy and dependability, they utilized several well-known supervised learning techniques, including support vector machine, multiple linear regression, LightGBM, and extremely randomized trees. The results revealed that the LightGBM model outperformed the other approaches, achieving an accuracy rate of 0.8655 and a mean absolute error of 6.65 minutes when predicting for a 1-hour timeframe. The researchers suggested future investigations should incorporate additional factors like national weather, city-pair, and network states for a more comprehensive analysis (Bojia et al., 2020).

According to Lambelho et al. (2020), they extensively evaluated strategic schedules by incorporating forecasts for arrival and departure flight delays in this research. There are three machine learning classification algorithms. LightGBM, multilayer perceptron, and random forests were utilized to classify flight delays six months in advance of the actual flight execution. The results indicated that all three classifiers achieved an accuracy rate of 0.75 or higher in categorizing flight delays. Particularly, LightGBM demonstrated the best performance, exhibiting an accuracy of 0.794, precision of 0.567, recall of 0.553, and an area under the ROC curve of 0.786 for both departure and arrival delays. Consequently, the researchers recommend future investigations to expand the range of features employed in the prediction algorithms, with the aim of further improving prediction accuracy (Lambelho et al., 2020).

Drawing upon the accessible search outcomes, the literature review pertaining to machine learning-based flight delay prediction provides a classification system and condenses the diverse approaches implemented to tackle this issue. The review accentuates the growing utilization of machine learning techniques and underscores the connections between flight delay prediction challenges and prevailing research trends. This comprehensive literature

6. References

Yu, B., Guo, Z., Asian, S., Wang, H., & Chen, G. (2019). Flight delay prediction for commercial air transport: A deep learning approach. *Transportation Research Part E-logistics and Transportation Review*, *125*, 203–221. https://doi.org/10.1016/j.tre.2019.03.013

Anupkumar, A. (n.d.). INVESTIGATING THE COSTS AND ECONOMIC IMPACT OF FLIGHT DELAYS IN THE AVIATION INDUSTRY AND THE POTENTIAL STRATEGIES FOR REDUCTION. CSUSB ScholarWorks. https://scholarworks.lib.csusb.edu/etd/1653

CAA. (2014). Passenger experiences during flight disruption Consumer research report. Civil Aviation Authority.

Peterson, E. B., Neels, K., Barczi, N., & Graham, T. (2013). The economic cost of airline flight delay. Journal of Transport Economics and Policy (JTEP), 47(1), 107-121.

Hatipoğlu, I., Tosun, Ö., & Tosun, N. (2022). Flight delay prediction based with machine learning. LogForum, 18(1), 97–107. https://doi.org/10.17270/j.log.2022.655

Chen, J., & Liu, L. (2019). Chained Predictions of Flight Delay Using Machine Learning. In AIAA Scitech 2019 Forum. https://doi.org/10.2514/6.2019-1661

Bojia, Y., Liu, B., Tian, Y., & Wan, L. (2020). A Methodology for Predicting Aggregate Flight Departure Delays in Airports Based on Supervised Learning. Sustainability, 12(7), 2749. https://doi.org/10.3390/su12072749

Lambelho, M., Mitici, M., Pickup, S., & Marsden, A. (2020). Assessing strategic flight schedules at an airport using machine learning-based flight delay and cancellation predictions. Journal of Air Transport Management, 82, 101737. https://doi.org/10.1016/j.jairtraman.2019.101737

Herbas, J. (2021, December 16). Using Machine Learning to Predict Flight Delays - Analytics Vidhya - Medium. Medium. https://medium.com/analytics-vidhya/using-machine-learning-to-predict-flight-delays-e8a50b0bb64c

review provides valuable insights into the scope, data, computational methods, and emerging trends in the field of machine learning-based flight delay prediction.

5. Deliverables

Airlines and airports can benefit from the project by gaining insights into flight delays and improving their operational efficiency. By accurately predicting flight delays, airlines can optimize their resources, adjust schedules, and minimize disruptions. This leads to smoother operations, improved on-time performance, and better resource allocation.

Passengers can benefit from the project through improved travel planning and a better overall experience. By providing accurate flight delay predictions, passengers can make informed decisions about their itineraries, such as adjusting connecting flights, rescheduling ground transportation, or modifying travel plans. This reduces the inconvenience caused by unexpected delays and increases passenger satisfaction.

The implementation of a system that incorporates the developed predictive models. This system should be capable of processing real-time or historical data inputs and generating flight delay predictions. The system will incorporate the following techniques and approaches:

1. Allow users can access accurate and reliable predictions of flight delays for specific flights or routes.
2. Allow users monitor the schedules of flights and receive notifications.
3. Allow users to explore alternative flight routes or connections based on the predicted delays.
4. Allow airlines and airports can use the system to optimize resource allocation, such as gate assignments, crew scheduling, and aircraft utilization.
5. Allow users can assess the potential impacts of flight delays.
6. Allow users with reports and analytics.
7. Allow facilitates collaboration and communication among different stakeholders.

Figure 123: Figure of PPF

Appendix B: Ethics Forms (Fast Track)

Office Record	Receipt – Fast-Track Ethical Approval
Date Received:	Student name: Student number:
Received by whom:	Received by: Date:

APU / APIIT FAST-TRACK ETHICAL APPROVAL FORM (STUDENTS)

Tick one box (level of study):

- ☐ POSTGRADUATE (PhD / MPhil / Masters)
- ☑ UNDERGRADUATE (Bachelors degree)
- ☐ FOUNDATION / DIPLOMA / Other categories

Tick one box (purpose of approval):

- ☑ Thesis / Dissertation / FYP project
- ☐ Module assignment
- ☐ Other: _____

Title of Programme on which enrolled BSc (Hons) in Computer Science With a specialism in Data Analytics

Tick one box: ☑ Full-Time Study or ☐ Part-Time Study

Title of project / assignment Machine Learning-Based Flight Delay Prediction

Name of student researcher Soe Chou Kit

Name of supervisor / lecturer Dr. Vazeerudeen Hameed

Student Researchers- please note that certain professional organisations have ethical guidelines that you may need to consult when completing this form.

Supervisors/Module Lecturers - please seek guidance from the Chair of the APU Research Ethics Committee if you are uncertain about any ethical issue arising from this application.

		YES	NO	N/A
1	Will you describe the main procedures to participants in advance, so that they are informed about what to expect?			√
2	Will you tell participants that their participation is voluntary?			√
3	Will you obtain written consent for participation?			√
4	If the research is observational, will you ask participants for their consent to being observed?			√
5	Will you tell participants that they may withdraw from the research at any time and for any reason?			√
6	With questionnaires and interviews will you give participants the option of omitting questions they do not want to answer?			√
7	Will you tell participants that their data will be treated with full confidentiality and that, if published, it will not be identifiable as theirs?			√
8	Will you give participants the opportunity to be debriefed i.e. to find out more about the study and its results?			√

If you have ticked **No** to any of Q1-8 you should complete the full Ethics Approval Form.

		YES	NO	N/A
9	Will your project/assignment deliberately mislead participants in any way?		√	
10	Is there any realistic risk of any participants experiencing either physical or psychological distress or discomfort?		√	
11	Is the nature of the research such that contentious or sensitive issues might be involved?		√	

If you have ticked **Yes** to 9, 10 or 11 you should complete the full Ethics Approval Form. In relation to question 10 this should include details of what you will tell participants to do if they should experience any problems (e.g. who they can contact for help). You may also need to consider risk assessment issues.

			YES	NO	N/A
12	Does your project/assignment involve work with animals?			√	
13	Do participants fall into any of the following special groups? **Note that you may also need to obtain satisfactory clearance from the relevant authorities**	Children (under 18 years of age)		√	
		People with communication or learning difficulties			
		Patients			
		People in custody			
		People who could be regarded as vulnerable			
		People engaged in illegal activities (eg drug taking)			
14	Does the project/assignment involve external funding or external collaboration where the funding body or external collaborative partner requires the University to provide evidence that the project/assignment had been subject to ethical scrutiny?			√	

If you have ticked **Yes** to 12, 13 or 14 you should complete the full Ethics Approval Form. There is an obligation on student and supervisor to bring to the attention of the APU Research Ethics Committee any issues with ethical implications not clearly covered by the above checklist.

STUDENT RESEARCHER

Provide in the boxes below (plus any other appended details) information required in support of your application, THEN SIGN THE FORM.

Please Tick Boxes

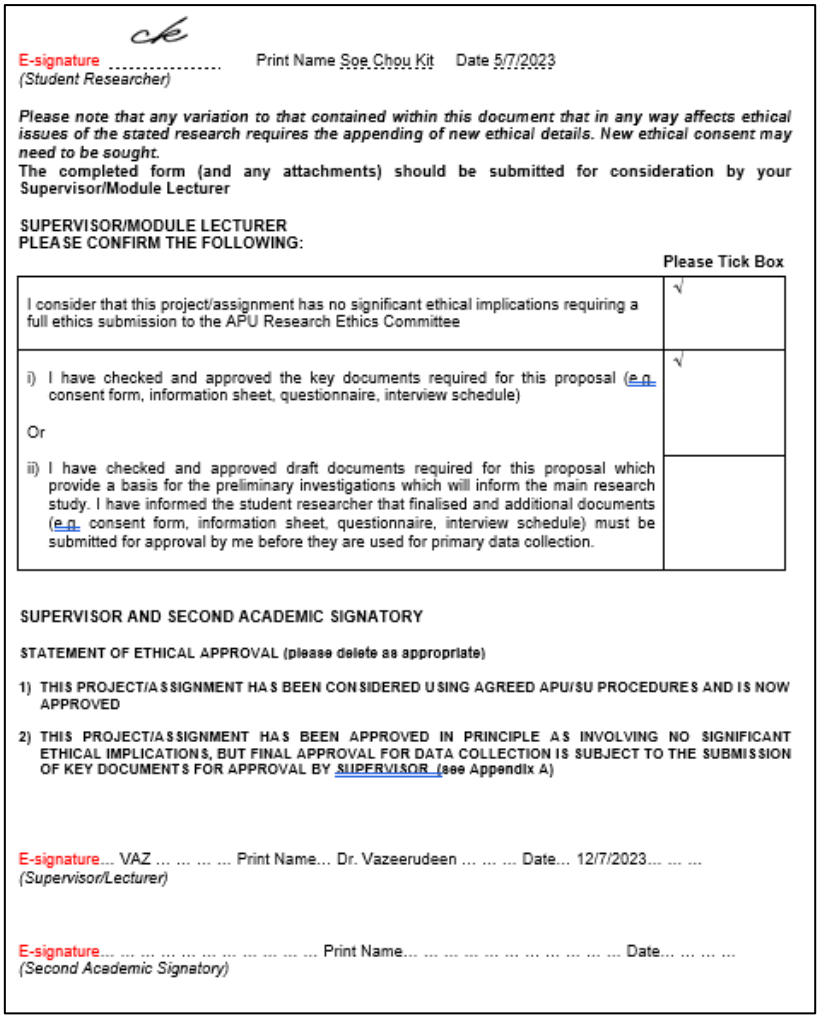
I consider that this project/assignment has no significant ethical implications requiring a full ethics submission to the APU Research Ethics Committee.	√

Give a brief description of participants and procedure (methods, tests used etc) in up to 150 words.

This project to predict flight delay by using machine learning technique. This project will utilize an open data source dataset source from Kaggle with 6.31M observations and 61 variables. The link of the dataset: https://www.kaggle.com/datasets/robikscube/flight-delay-dataset-20182022?select=raw

The dataset will do data pre-processing including handling missing value, data normalization and remove unnecessary variables. Random forest, XG Boost and Decision tree will be considered as the machine learning methods to train the models. The model will be evaluated using accuracy, precision and recall and F1 score. There is no survey or interview to be conducted for the project.

I also confirm that: i) All key documents e.g. consent form, information sheet, questionnaire/interview are appended to this application.	N/A
Or ii) Any key documents e.g. consent form, information sheet, questionnaire/interview schedules which need to be finalised following initial investigations will be submitted for approval by the project/assignment supervisor/module lecturer before they are used in primary data collection.	



E-signature ..... Print Name Soe Chou Kit Date 5/7/2023
(Student Researcher)

Please note that any variation to that contained within this document that in any way affects ethical issues of the stated research requires the appending of new ethical details. New ethical consent may need to be sought.
The completed form (and any attachments) should be submitted for consideration by your Supervisor/Module Lecturer

SUPERVISOR/MODULE LECTURER
PLEASE CONFIRM THE FOLLOWING:

	Please Tick Box
I consider that this project/assignment has no significant ethical implications requiring a full ethics submission to the APU Research Ethics Committee	√
i) I have checked and approved the key documents required for this proposal (e.g. consent form, information sheet, questionnaire, interview schedule) Or	√
ii) I have checked and approved draft documents required for this proposal which provide a basis for the preliminary investigations which will inform the main research study. I have informed the student researcher that finalised and additional documents (e.g. consent form, information sheet, questionnaire, interview schedule) must be submitted for approval by me before they are used for primary data collection.	

SUPERVISOR AND SECOND ACADEMIC SIGNATORY

STATEMENT OF ETHICAL APPROVAL (please delete as appropriate)

1) **THIS PROJECT/ASSIGNMENT HAS BEEN CONSIDERED USING AGREED APU/SU PROCEDURES AND IS NOW APPROVED**

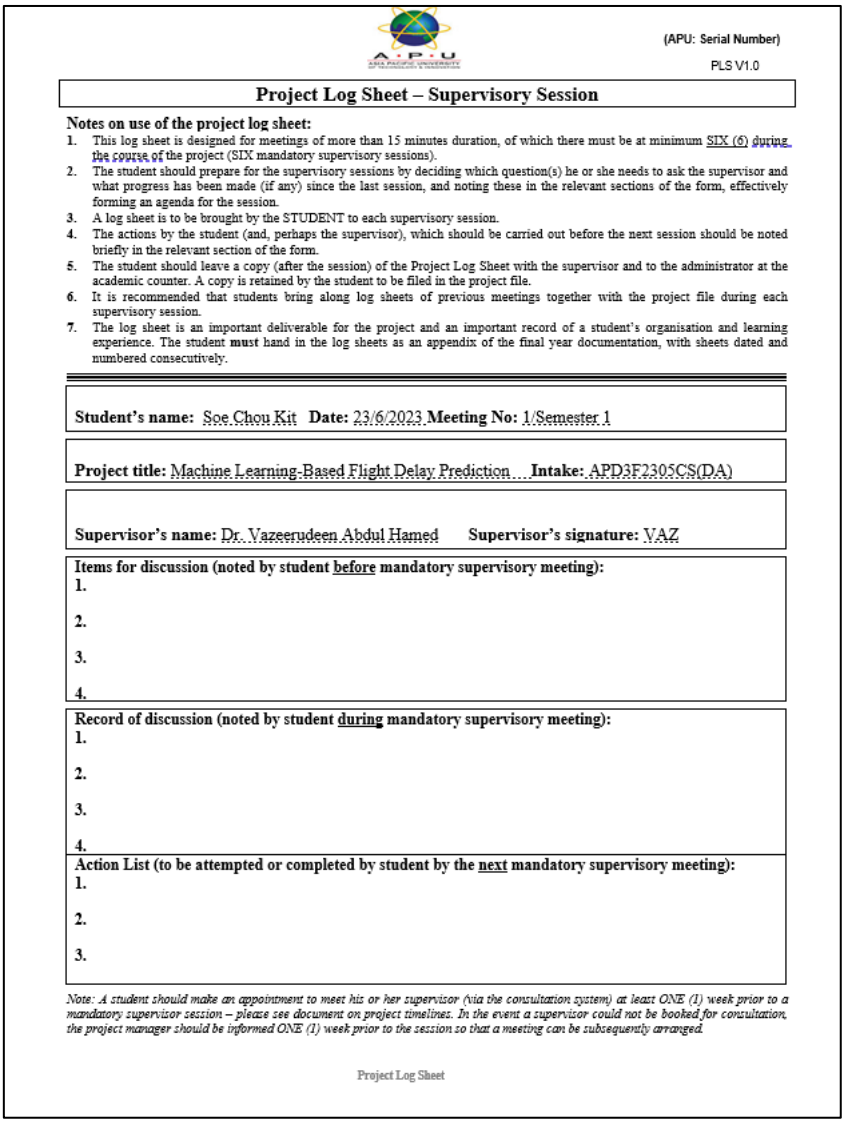
2) **THIS PROJECT/ASSIGNMENT HAS BEEN APPROVED IN PRINCIPLE AS INVOLVING NO SIGNIFICANT ETHICAL IMPLICATIONS, BUT FINAL APPROVAL FOR DATA COLLECTION IS SUBJECT TO THE SUBMISSION OF KEY DOCUMENTS FOR APPROVAL BY SUPERVISOR (see Appendix A)**

E-signature... VAZ ... ... Print Name... Dr. Vazeerudeen ... ... Date... 12/7/2023... ... ...
(Supervisor/Lecturer)

E-signature... ... ... Print Name... ... ... Date... ... ...
(Second Academic Signatory)

Figure 124: Figure of Ethics Forms

Appendix C: Log Sheets



(APU: Serial Number)

PLS V1.0

Project Log Sheet – Supervisory Session

Notes on use of the project log sheet:

1. This log sheet is designed for meetings of more than 15 minutes duration, of which there must be at minimum SIX (6) during the course of the project (SIX mandatory supervisory sessions).
2. The student should prepare for the supervisory sessions by deciding which question(s) he or she needs to ask the supervisor and what progress has been made (if any) since the last session, and noting these in the relevant sections of the form, effectively forming an agenda for the session.
3. A log sheet is to be brought by the STUDENT to each supervisory session.
4. The actions by the student (and, perhaps the supervisor), which should be carried out before the next session should be noted briefly in the relevant section of the form.
5. The student should leave a copy (after the session) of the Project Log Sheet with the supervisor and to the administrator at the academic counter. A copy is retained by the student to be filed in the project file.
6. It is recommended that students bring along log sheets of previous meetings together with the project file during each supervisory session.
7. The log sheet is an important deliverable for the project and an important record of a student's organisation and learning experience. The student **must** hand in the log sheets as an appendix of the final year documentation, with sheets dated and numbered consecutively.

Student's name: Soe Chou Kit **Date:** 23/6/2023 **Meeting No:** 1/Semester 1

Project title: Machine Learning-Based Flight Delay Prediction **Intake:** APD3F2305CS(DA)

Supervisor's name: Dr. Vazeerudeen Abdul Hamed **Supervisor's signature:** VAZ

Items for discussion (noted by student before mandatory supervisory meeting):

1.
2.
3.
4.

Record of discussion (noted by student during mandatory supervisory meeting):

1.
2.
3.
4.

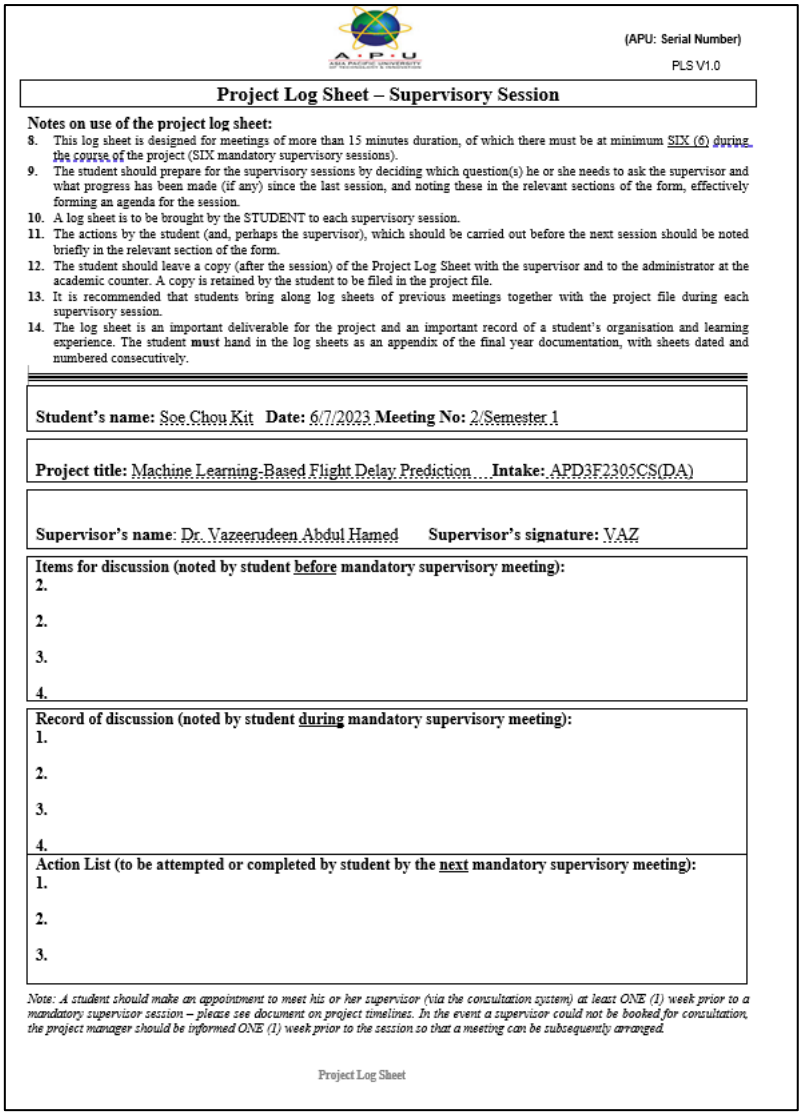
Action List (to be attempted or completed by student by the next mandatory supervisory meeting):

1.
2.
3.

Note: A student should make an appointment to meet his or her supervisor (via the consultation system) at least ONE (1) week prior to a mandatory supervisor session – please see document on project timelines. In the event a supervisor could not be booked for consultation, the project manager should be informed ONE (1) week prior to the session so that a meeting can be subsequently arranged.

Project Log Sheet

Figure 125: Project Log Sheet 1



(APU: Serial Number)

PLS V1.0

Project Log Sheet – Supervisory Session

Notes on use of the project log sheet:

8. This log sheet is designed for meetings of more than 15 minutes duration, of which there must be at minimum SIX (6) during the course of the project (SIX mandatory supervisory sessions).
9. The student should prepare for the supervisory sessions by deciding which question(s) he or she needs to ask the supervisor and what progress has been made (if any) since the last session, and noting these in the relevant sections of the form, effectively forming an agenda for the session.
10. A log sheet is to be brought by the STUDENT to each supervisory session.
11. The actions by the student (and, perhaps the supervisor), which should be carried out before the next session should be noted briefly in the relevant section of the form.
12. The student should leave a copy (after the session) of the Project Log Sheet with the supervisor and to the administrator at the academic counter. A copy is retained by the student to be filed in the project file.
13. It is recommended that students bring along log sheets of previous meetings together with the project file during each supervisory session.
14. The log sheet is an important deliverable for the project and an important record of a student's organisation and learning experience. The student **must** hand in the log sheets as an appendix of the final year documentation, with sheets dated and numbered consecutively.

Student's name: Soe Chou Kit **Date:** 6/7/2023 **Meeting No:** 2/Semester 1

Project title: Machine Learning-Based Flight Delay Prediction **Intake:** APD3F2305CS(DA)

Supervisor's name: Dr. Vazeerudeen Abdul Hamed **Supervisor's signature:** VAZ

Items for discussion (noted by student before mandatory supervisory meeting):

2.

2.

3.

4.

Record of discussion (noted by student during mandatory supervisory meeting):

1.

2.

3.

4.

Action List (to be attempted or completed by student by the next mandatory supervisory meeting):

1.

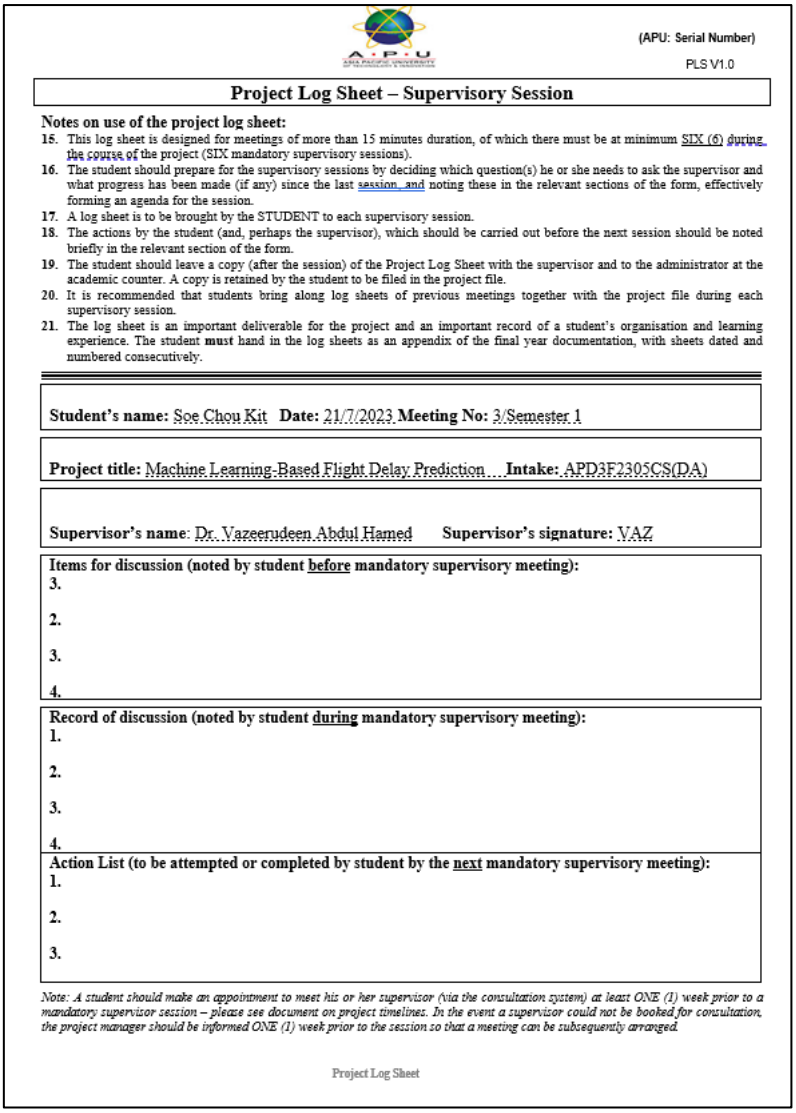
2.

3.

Note: A student should make an appointment to meet his or her supervisor (via the consultation system) at least ONE (1) week prior to a mandatory supervisor session – please see document on project timelines. In the event a supervisor could not be booked for consultation, the project manager should be informed ONE (1) week prior to the session so that a meeting can be subsequently arranged.

Project Log Sheet

Figure 126: Project Log Sheet 2



(APU: Serial Number)

PLS V1.0

Project Log Sheet – Supervisory Session

Notes on use of the project log sheet:

15. This log sheet is designed for meetings of more than 15 minutes duration, of which there must be at minimum SIX (6) during the course of the project (SIX mandatory supervisory sessions).
16. The student should prepare for the supervisory sessions by deciding which question(s) he or she needs to ask the supervisor and what progress has been made (if any) since the last session, and noting these in the relevant sections of the form, effectively forming an agenda for the session.
17. A log sheet is to be brought by the STUDENT to each supervisory session.
18. The actions by the student (and, perhaps the supervisor), which should be carried out before the next session should be noted briefly in the relevant section of the form.
19. The student should leave a copy (after the session) of the Project Log Sheet with the supervisor and to the administrator at the academic counter. A copy is retained by the student to be filed in the project file.
20. It is recommended that students bring along log sheets of previous meetings together with the project file during each supervisory session.
21. The log sheet is an important deliverable for the project and an important record of a student's organisation and learning experience. The student **must** hand in the log sheets as an appendix of the final year documentation, with sheets dated and numbered consecutively.

Student's name: Soe Chou Kit **Date:** 21/7/2023 **Meeting No:** 3/Semester 1

Project title: Machine Learning-Based Flight Delay Prediction **Intake:** APD3F2305CS(DA)

Supervisor's name: Dr. Vazeerudeen Abdul Hamed **Supervisor's signature:** VAZ

Items for discussion (noted by student before mandatory supervisory meeting):

3.

2.

3.

4.

Record of discussion (noted by student during mandatory supervisory meeting):

1.

2.

3.

4.

Action List (to be attempted or completed by student by the next mandatory supervisory meeting):

1.

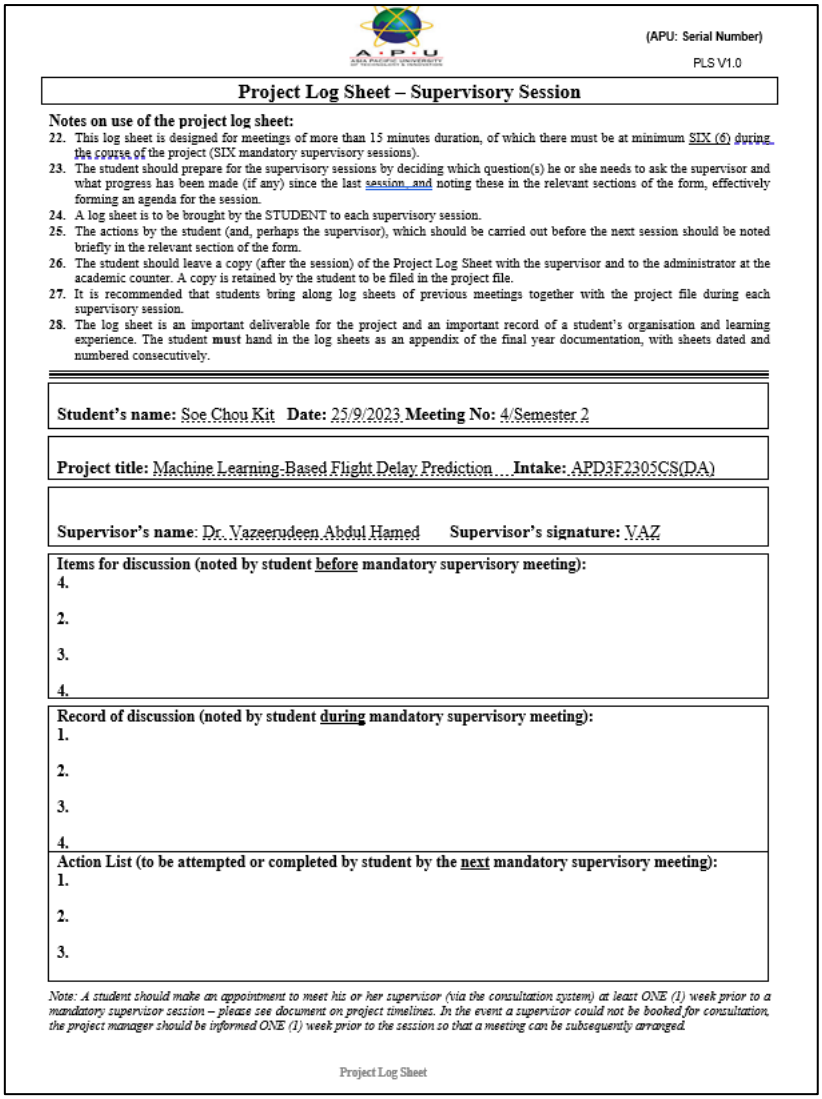
2.

3.

Note: A student should make an appointment to meet his or her supervisor (via the consultation system) at least ONE (1) week prior to a mandatory supervisor session – please see document on project timelines. In the event a supervisor could not be booked for consultation, the project manager should be informed ONE (1) week prior to the session so that a meeting can be subsequently arranged.

Project Log Sheet

Figure 127: Project Log Sheet 3



(APU: Serial Number)

PLS V1.0

Project Log Sheet – Supervisory Session

Notes on use of the project log sheet:

22. This log sheet is designed for meetings of more than 15 minutes duration, of which there must be at minimum SIX (6) during the course of the project (SIX mandatory supervisory sessions).
23. The student should prepare for the supervisory sessions by deciding which question(s) he or she needs to ask the supervisor and what progress has been made (if any) since the last session, and noting these in the relevant sections of the form, effectively forming an agenda for the session.
24. A log sheet is to be brought by the STUDENT to each supervisory session.
25. The actions by the student (and, perhaps the supervisor), which should be carried out before the next session should be noted briefly in the relevant section of the form.
26. The student should leave a copy (after the session) of the Project Log Sheet with the supervisor and to the administrator at the academic counter. A copy is retained by the student to be filed in the project file.
27. It is recommended that students bring along log sheets of previous meetings together with the project file during each supervisory session.
28. The log sheet is an important deliverable for the project and an important record of a student's organisation and learning experience. The student **must** hand in the log sheets as an appendix of the final year documentation, with sheets dated and numbered consecutively.

Student's name: Soe Chou Kit **Date:** 25/9/2023 **Meeting No:** 4/Semester 2

Project title: Machine Learning-Based Flight Delay Prediction … **Intake:** APD3F2305CS(DA)

Supervisor's name: Dr. Vazeerudeen Abdul Hamed **Supervisor's signature:** VAZ

Items for discussion (noted by student before mandatory supervisory meeting):

4.

2.

3.

4.

Record of discussion (noted by student during mandatory supervisory meeting):

1.

2.

3.

4.

Action List (to be attempted or completed by student by the next mandatory supervisory meeting):

1.

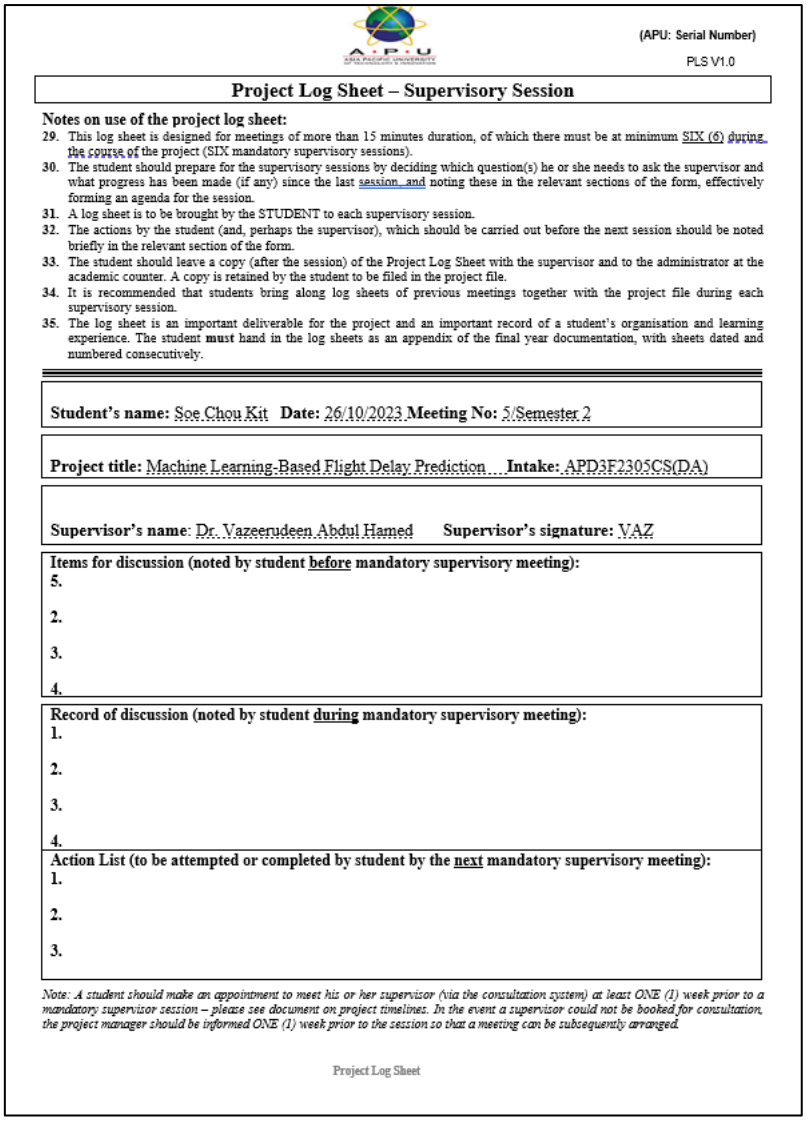
2.

3.

Note: A student should make an appointment to meet his or her supervisor (via the consultation system) at least ONE (1) week prior to a mandatory supervisor session – please see document on project timelines. In the event a supervisor could not be booked for consultation, the project manager should be informed ONE (1) week prior to the session so that a meeting can be subsequently arranged.

Project Log Sheet

Figure 128: Project Log Sheet 4



(APU: Serial Number)

PLS V1.0

Project Log Sheet – Supervisory Session

Notes on use of the project log sheet:

29. This log sheet is designed for meetings of more than 15 minutes duration, of which there must be at minimum SIX (6) during the course of the project (SIX mandatory supervisory sessions).
30. The student should prepare for the supervisory sessions by deciding which question(s) he or she needs to ask the supervisor and what progress has been made (if any) since the last session, and noting these in the relevant sections of the form, effectively forming an agenda for the session.
31. A log sheet is to be brought by the STUDENT to each supervisory session.
32. The actions by the student (and, perhaps the supervisor), which should be carried out before the next session should be noted briefly in the relevant section of the form.
33. The student should leave a copy (after the session) of the Project Log Sheet with the supervisor and to the administrator at the academic counter. A copy is retained by the student to be filed in the project file.
34. It is recommended that students bring along log sheets of previous meetings together with the project file during each supervisory session.
35. The log sheet is an important deliverable for the project and an important record of a student's organisation and learning experience. The student **must** hand in the log sheets as an appendix of the final year documentation, with sheets dated and numbered consecutively.

Student's name: Soe Chou Kit **Date:** 26/10/2023 **Meeting No:** 5/Semester 2

Project title: Machine Learning-Based Flight Delay Prediction **Intake:** APD3F2305CS(DA)

Supervisor's name: Dr. Vazeerudeen Abdul Hamed **Supervisor's signature:** VAZ

Items for discussion (noted by student before mandatory supervisory meeting):

5.

2.

3.

4.

Record of discussion (noted by student during mandatory supervisory meeting):

1.

2.

3.

4.

Action List (to be attempted or completed by student by the next mandatory supervisory meeting):

1.

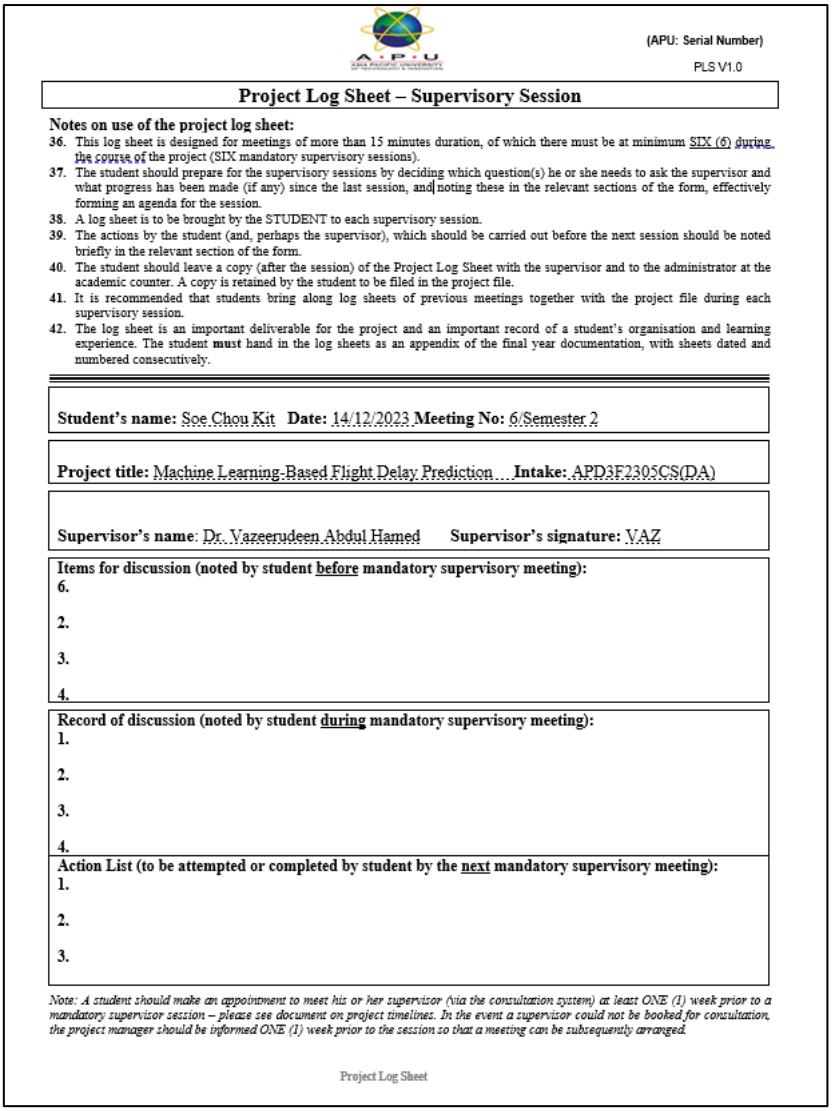
2.

3.

Note: A student should make an appointment to meet his or her supervisor (via the consultation system) at least ONE (1) week prior to a mandatory supervisor session – please see document on project timelines. In the event a supervisor could not be booked for consultation, the project manager should be informed ONE (1) week prior to the session so that a meeting can be subsequently arranged.

Project Log Sheet

Figure 129: Project Log Sheet 5



(APU: Serial Number)

PLS V1.0

Project Log Sheet – Supervisory Session

Notes on use of the project log sheet:

36. This log sheet is designed for meetings of more than 15 minutes duration, of which there must be at minimum SIX (6) during the course of the project (SIX mandatory supervisory sessions).
37. The student should prepare for the supervisory sessions by deciding which question(s) he or she needs to ask the supervisor and what progress has been made (if any) since the last session, and noting these in the relevant sections of the form, effectively forming an agenda for the session.
38. A log sheet is to be brought by the STUDENT to each supervisory session.
39. The actions by the student (and, perhaps the supervisor), which should be carried out before the next session should be noted briefly in the relevant section of the form.
40. The student should leave a copy (after the session) of the Project Log Sheet with the supervisor and to the administrator at the academic counter. A copy is retained by the student to be filed in the project file.
41. It is recommended that students bring along log sheets of previous meetings together with the project file during each supervisory session.
42. The log sheet is an important deliverable for the project and an important record of a student's organisation and learning experience. The student **must** hand in the log sheets as an appendix of the final year documentation, with sheets dated and numbered consecutively.

Student's name: Soe Chou Kit **Date:** 14/12/2023 **Meeting No:** 6/Semester 2

Project title: Machine Learning-Based Flight Delay Prediction **Intake:** APD3F2305CS(DA)

Supervisor's name: Dr. Vazeerudeen Abdul Hamed **Supervisor's signature:** VAZ

Items for discussion (noted by student before mandatory supervisory meeting):

6.

2.

3.

4.

Record of discussion (noted by student during mandatory supervisory meeting):

1.

2.

3.

4.

Action List (to be attempted or completed by student by the next mandatory supervisory meeting):

1.

2.

3.

Note: A student should make an appointment to meet his or her supervisor (via the consultation system) at least ONE (1) week prior to a mandatory supervisor session – please see document on project timelines. In the event a supervisor could not be booked for consultation, the project manager should be informed ONE (1) week prior to the session so that a meeting can be subsequently arranged.

Project Log Sheet

Figure 130: Project Log Sheet 6

Appendix D: Poster

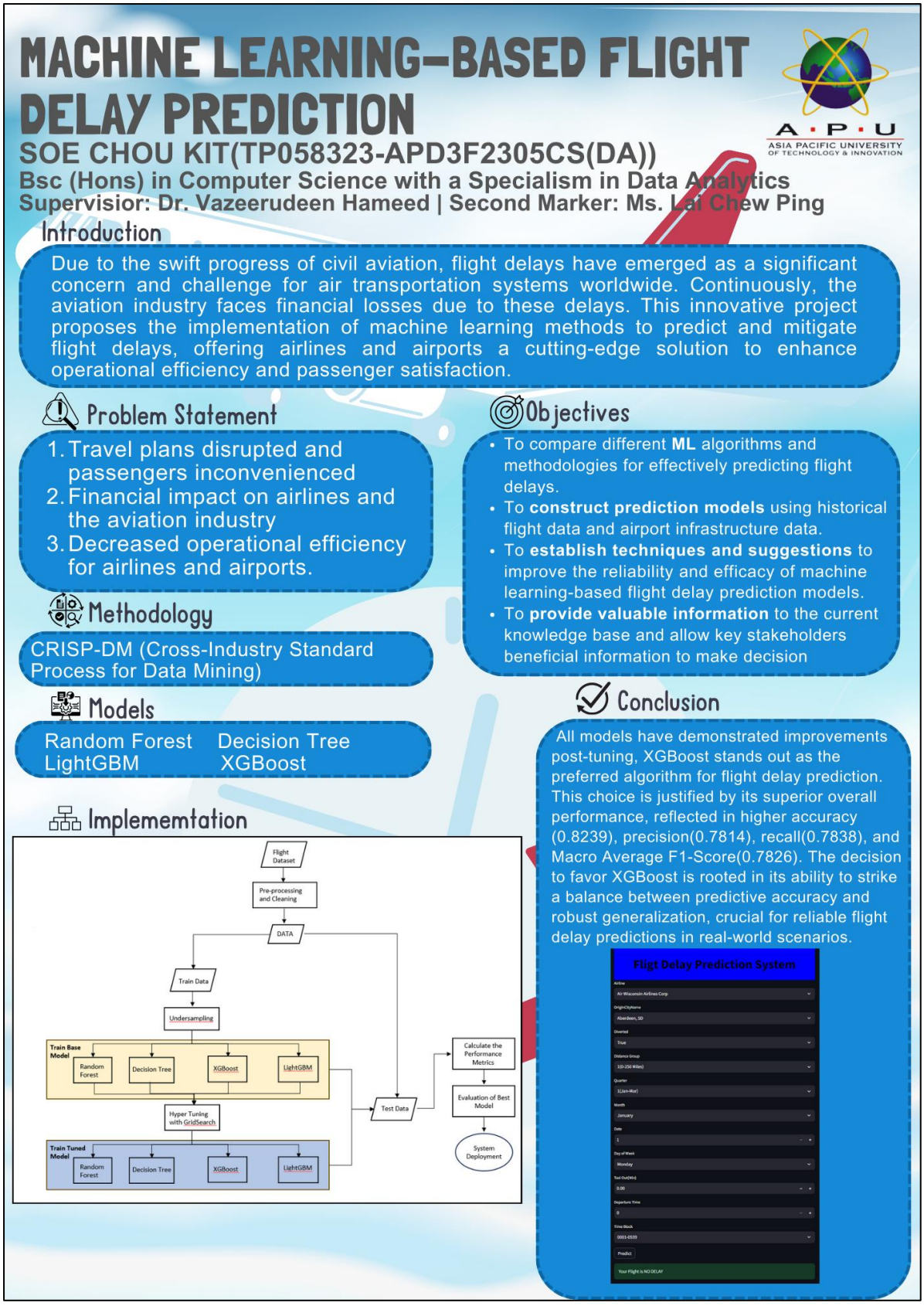
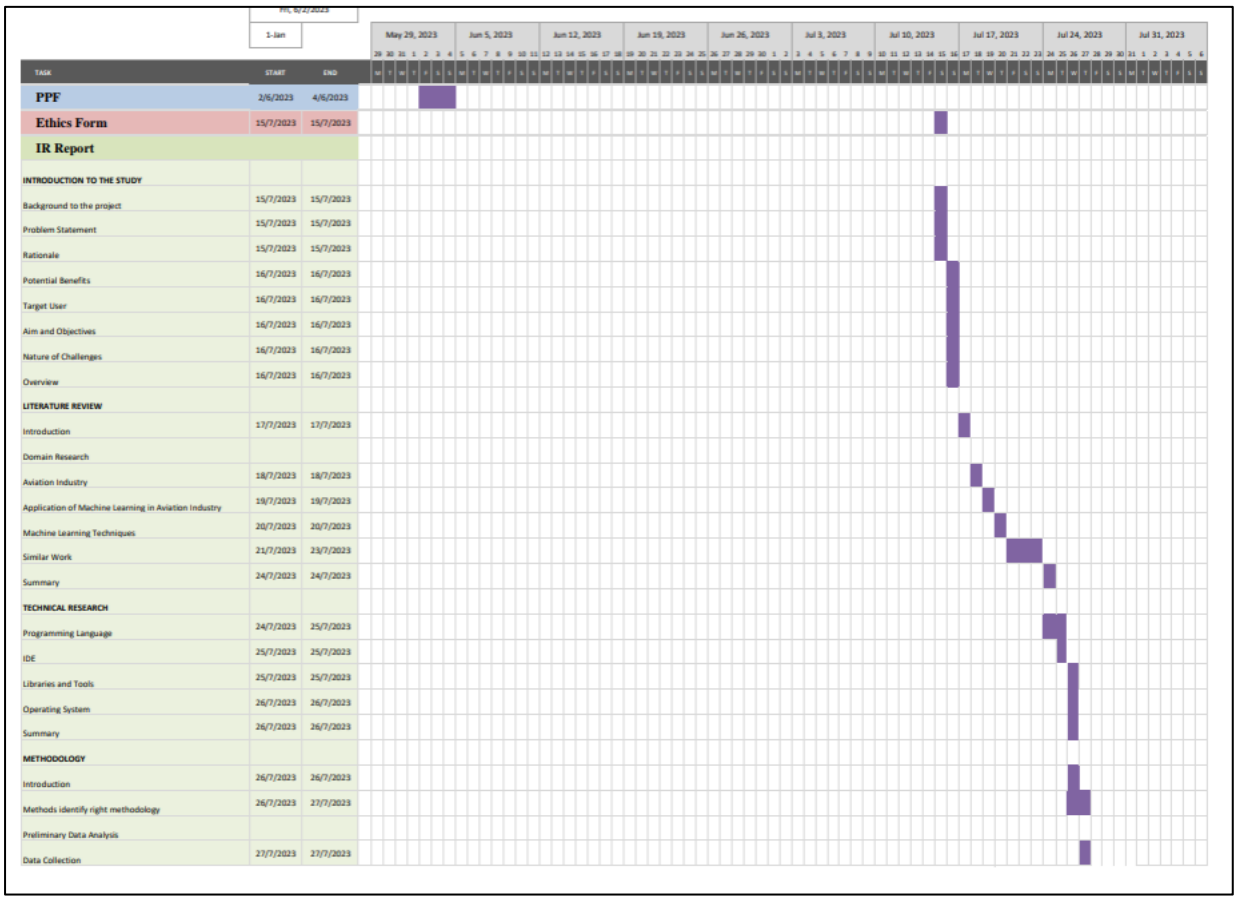


Figure 131: Poster

Appendix E: Gantt Chart

Gantt Chart for FYP IR Semester 1



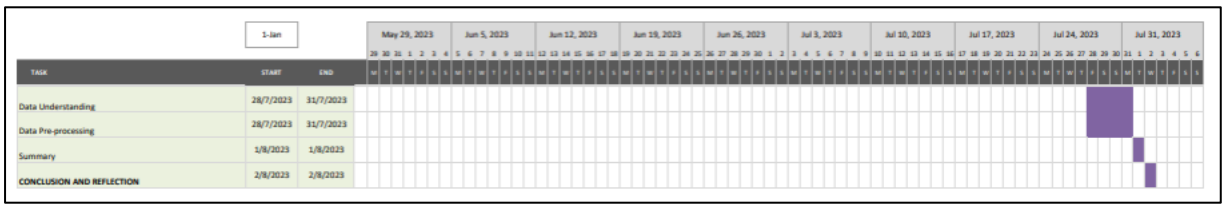


Figure 132: Gantt Chart of Semester 1

Gantt Chart for FYP Semester 2

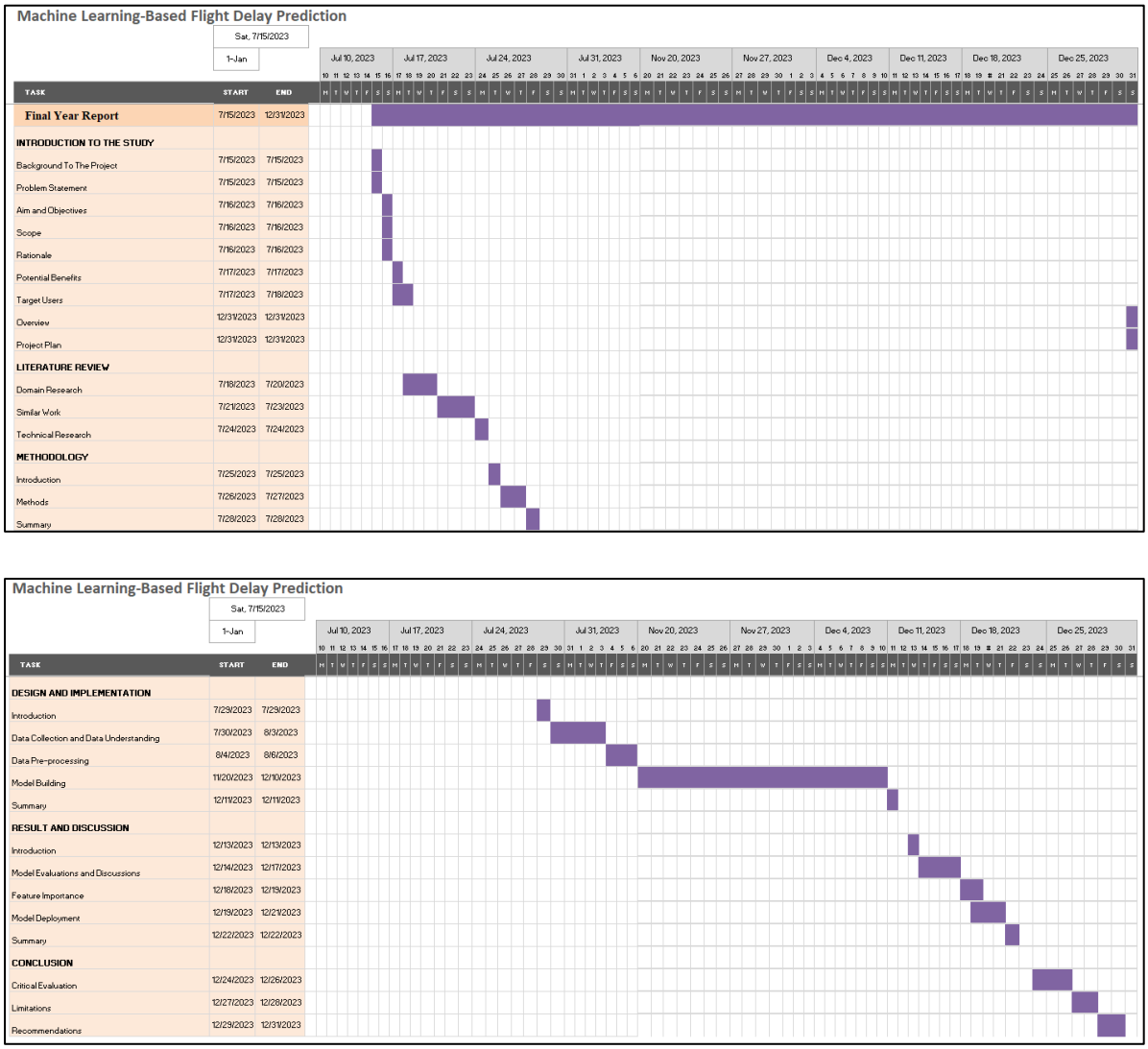


Figure 133: Gantt Chart of Semester 2

Appendix F: Sample Code Implementation

Mount

```
[ ] # Mount your Google Drive.
    from google.colab import drive
    drive.mount("/content/drive")
```

```
Mounted at /content/drive
```

```
import pandas as pd
import dask.dataframe as dd
import matplotlib.pyplot as plt
import numpy as np
import xgboost as xgb
import lightgbm as lgb
import pickle
import streamlit as st
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from imblearn.under_sampling import RandomUnderSampler
from collections import Counter
from tabulate import tabulate
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import make_scorer, accuracy_score, classification_report, confusion_matrix
from sklearn.model_selection import learning_curve
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier, plot_tree


dataset_path = "/content/drive/My Drive/Dataset/Combined_Flights_2021.csv"
df = dd.read_csv(dataset_path)
```

```
[ ] df.head()
```

LoadData

```
[ ] # Identify variables and data types
    df.info()
```

Pre-Processing

Feature Selection

```
[ ] necessary_variables = ['Airline', 'OriginCityName', 'Diverted', 'DistanceGroup', 'Quarter',
                           'Month', 'DayofMonth', 'DayOfWeek', 'TaxiOut', 'DepDelayMinutes', 'DepTime', 'Cancelled','DepTimeBlk']

    sv = df[necessary_variables]
```

Convert to Pandas

```
[ ] df_pandas = sv.compute()
```

```
[ ] df_pandas.shape
```

```
[ ] df_pandas.head()
```

Delete the row that flight has been cancelled

```
[ ] df_pandas = df_pandas.loc[df_pandas['Cancelled'] != True]
```

Check Missing Value (After Pre-processing)

```
[ ] missing_values = df_pandas.isnull().sum()

    print("Missing values in each column:")
    print(missing_values)
```

Delete duplicate rows

```
[ ] df_pandas = df_pandas.drop_duplicates()
```

Check Duplicate Rows (After Pre-processing)

```
[ ] # Find and count duplicate rows
    duplicate_rows = df_pandas[df_pandas.duplicated()]
    num_duplicates = len(duplicate_rows)

    # Print the duplicate rows and the count
    print(f"Number of Duplicate Rows: {num_duplicates}")
```

Create New Columns for Category Delay

```
[ ] df_pandas['DepDelTypes'] = None
```

Condition for Category

```
# dep delay 1=delay,0=not delay)
df_pandas['DepDelTypes'] = df_pandas['DepDelTypes'].mask(
    df_pandas['DepDelayMinutes'] <= 0, '0')
df_pandas['DepDelTypes'] = df_pandas['DepDelTypes'].mask(
    df_pandas['DepDelayMinutes'] > 0, '1')
```

Delete Column Cancelled and DepDelayMinutes

```
[ ] columns_to_drop = ['Cancelled','DepDelayMinutes']
    df_pandas.drop(columns=columns_to_drop, inplace=True)
```

Change Data Type

```
[ ] df_pandas['DepTime'] = df_pandas['DepTime'].astype(int)
    df_pandas['DepDelTypes'] = df_pandas['DepDelTypes'].astype(int)
```

```
[ ] df_pandas.head()
```

Label Encoder

```
[ ] label_encoder = LabelEncoder()
    df_pandas['Airline'] = label_encoder.fit_transform(df_pandas['Airline'])
    df_pandas['OriginCityName'] = label_encoder.fit_transform(df_pandas['OriginCityName'])
    df_pandas['DepTimeBlk'] = label_encoder.fit_transform(df_pandas['DepTimeBlk'])
```

```
[ ] df_pandas.info()
```

Splitting Dataset into Training and Testing Sets

```
[ ] X = df_pandas.drop('DepDelTypes', axis=1)  # Features
    y = df_pandas['DepDelTypes']  # Target variable
```

```
[ ] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
```

Undersampling train set

```
[ ] # Create an instance of RandomUnderSampler
    undersampler = RandomUnderSampler(sampling_strategy=1)

    # Apply random undersampling to the dataset
    X_train, y_train = undersampler.fit_resample(X_train,y_train)
```

```
[ ] print("Class distribution after undersampling:", Counter(y_train))
```

EDA

Variables Selected and data types

```
[ ] df_pandas.info()
```

Check Missing Value (Before Pre-processing)

```
[ ] missing_values = df_pandas.isnull().sum()

    print("Missing values in each column:")
    print(missing_values)
```

```
[ ] cancelled_counts = df_pandas['Cancelled'].value_counts()

    # Print the result
    print(cancelled_counts)
```

Check Duplicate Rows (Before Pre-processing)

```
[ ] # Find and count duplicate rows
    duplicate_rows = df_pandas[df_pandas.duplicated()]
    num_duplicates = len(duplicate_rows)

    # Print the duplicate rows and the count
    print(f"Number of Duplicate Rows: {num_duplicates}")
```

```
[ ] df_pandas.shape
```

Statistics

```
[ ] df_pandas.describe()
```

Departure Delay in Minutes

```
[ ] delay_minutes = df_pandas.query('(DepDelayMinutes < 30)')

    # Create a histogram
    plt.figure(figsize=(8, 6))
    plt.hist(delay_minutes['DepDelayMinutes'], bins=30, edgecolor='black')
    plt.xlabel('DepDelayMinutes (min)')
    plt.ylabel('Frequency')
    plt.title('Histogram of DepDelayMinutes')
    plt.tight_layout()

    plt.show()
```

```
delay_minutes = df_pandas.query(
    '(DepDelayMinutes > 0) and (DepDelayMinutes < 61)')

# Create a histogram
plt.figure(figsize=(8, 6))
plt.hist(delay_minutes['DepDelayMinutes'], bins=60, edgecolor='black')
plt.xlabel('DepDelayMinutes (min)')
plt.ylabel('Frequency')
plt.title('Histogram of DepDelayMinutes (Delay > 0 min)')
plt.tight_layout()

plt.show()
```

Target Variable

```
[ ] # Count the occurrences of each group
    delgroups_count = Counter(df_pandas['DepDelTypes'])

    # Calculate the total count
    total_count = len(df_pandas['DepDelTypes'])

    # Create a list of dictionaries to represent the table
    table_data = [
        {'DelTypes': group, 'Count': count} for group, count in delgroups_count.items()
    ]
    table_data.append({'DelTypes': 'Total', 'Count': total_count})

    # Print the table using tabulate
    print(tabulate(table_data, headers='keys', tablefmt='grid'))
```

Pie Chart (Distribution of Target Variable)

```
[ ] value_counts = df_pandas['DepDelTypes'].value_counts()

    # Step 3: Create the pie chart
    plt.figure(figsize=(8, 8))
    plt.pie(value_counts, labels=value_counts.index,
            autopct='%1.1f%%', startangle=140)
    plt.title(f'Distribution of DepDelGroups for ')
    plt.axis('equal')

    # Show the chart
    plt.show()
```

Airline

```
[ ] # Count of AIRLINE
    airline = set(df_pandas['Airline'])
    airline_count = len(airline)
    print("Number of airline:", airline_count)
```

Distribution of Delay Types for Each Airline

```
    airline_delay_count = df_pandas.groupby(
        ['Airline', 'DepDelTypes']).size().reset_index(name='Count')

    # Pivot the table to have airlines as rows and delay groups as columns
    airline_delay_pivot = airline_delay_count.pivot(
        index='Airline', columns='DepDelTypes', values='Count').fillna(0)

    # Add a new column 'Total' containing the total count for each airline
    airline_delay_pivot['Total'] = airline_delay_pivot.sum(axis=1)

    # Sort the airlines based on the 'Total' count in descending order
    airline_delay_pivot = airline_delay_pivot.sort_values(
        by='Total', ascending=False)

    # Print the table using tabulate
    print(tabulate(airline_delay_pivot, headers='keys', tablefmt='grid'))
```

```
[ ]  airline_delay_count = df_pandas.groupby(
         ['Airline', 'DepDelTypes']).size().reset_index(name='Count')

     # Pivot the table to have airlines as rows and delay groups as columns
     airline_delay_pivot = airline_delay_count.pivot(
         index='Airline', columns='DepDelTypes', values='Count').fillna(0)

     # Add a new column 'Total' containing the total count for each airline
     airline_delay_pivot['Total'] = airline_delay_pivot.sum(axis=1)

     # Sort the airlines based on the 'Total' count in descending order
     airline_delay_pivot = airline_delay_pivot.sort_values(
         by='Total', ascending=False)

     # Print the table using tabulate
     print(tabulate(airline_delay_pivot, headers='keys', tablefmt='grid'))
```

Grouped Horizontal Bar Chart (Distribution of Delay Types for Top 5 Airline)

```
[ ]  airline_delay_count = df_pandas.groupby(
         ['Airline', 'DepDelTypes']).size().unstack(fill_value=0)

     # Sort the airlines based on the total delay count in descending order
     airline_delay_count['Total'] = airline_delay_count['1'] + airline_delay_count['0']
     airline_delay_count = airline_delay_count.sort_values(
         by='Total', ascending=False)

     # Get the top five airlines with the highest values
     top_five_airlines = airline_delay_count.head(5)

     # Create a grouped bar chart for the top three airlines
     ax = top_five_airlines[['1', '0']].plot(
         kind='barh', stacked=False, figsize=(10, 6))
     plt.xlabel('Count')
     plt.ylabel('Airline')
     plt.title('Top Five Airlines Delay and NoDelay Counts')
     plt.legend(title='Delay Status', loc='upper right')
     # plt.xticks(top_three_airlines.index, range(len(top_three_airlines)), rotation=45, ha='right')
     plt.tight_layout()

     plt.show()
```

Bar Chart (Distribution of Delay Types for Southwest AirLines Co.)

```
# Filter the data to select the desired airline (e.g., 'Airline A')
selected_airline = 'Southwest Airlines Co.'
selected_airline_data = df_pandas[df_pandas['Airline'] == selected_airline]

# Group by 'Month' and 'DepDelTypes' for the selected airline, and count the occurrences
airline_month_delay_count = selected_airline_data.groupby(
    ['Month', 'DepDelTypes']).size().unstack(fill_value=0)

# Create a 'Total' column for the total count of delays and no delays for each month
# Plot the grouped bar chart for the selected airline and each month
ax = airline_month_delay_count[['1', '0']].plot(
    kind='bar', figsize=(10, 6))
plt.xlabel('Month')
plt.ylabel('Count')
plt.title(f'Delay and NoDelay Counts for {selected_airline} by Month')
plt.legend(title='Delay Status', loc='upper right')
plt.tight_layout()

plt.xticks(rotation=0)
plt.show()
```

Bar Chart (Distribution of Delay Types for Delta Air Lines Inc.)

```
[ ] # Filter the data to select the desired airline (e.g., 'Airline A')
selected_airline = 'Delta Air Lines Inc.'
selected_airline_data = df_pandas[df_pandas['Airline'] == selected_airline]

# Group by 'Month' and 'DepDelTypes' for the selected airline, and count the occurrences
airline_month_delay_count = selected_airline_data.groupby(
    ['Month', 'DepDelTypes']).size().unstack(fill_value=0)

# Create a 'Total' column for the total count of delays and no delays for each month
# Plot the grouped bar chart for the selected airline and each month
ax = airline_month_delay_count[['1', '0']].plot(
    kind='bar', figsize=(10, 6))
plt.xlabel('Month')
plt.ylabel('Count')
plt.title(f'Delay and NoDelay Counts for {selected_airline} by Month')
plt.legend(title='Delay Status', loc='upper right')
plt.tight_layout()

plt.xticks(rotation=0)
plt.show()
```

Quarter

```
[ ]  quarter_delay_count = df_pandas.groupby(
        ['Quarter', 'DepDelTypes']).size().reset_index(name='Count')

    # Pivot the table to have airlines as rows and delay groups as columns
    quarter_delay_pivot = quarter_delay_count.pivot(
        index='Quarter', columns='DepDelTypes', values='Count').fillna(0)

    # Add a new column 'Total' containing the total count for each airline
    quarter_delay_pivot['Total'] = quarter_delay_pivot.sum(axis=1)
    quarter_delay_pivot = quarter_delay_pivot.sort_values(
        by='Quarter', ascending=True)

    # Print the table using tabulate
    print(tabulate(quarter_delay_pivot, headers='keys', tablefmt='grid'))
```

Bar Chart (Distribution of Delay Types for Each Quarter)

```
    quarter_delay_count = df_pandas.groupby(
        ['Quarter', 'DepDelTypes']).size().unstack(fill_value=0)

    # Create a grouped bar chart for the top three airlines
    ax = quarter_delay_count[['1', '0']].plot(
        kind='bar', stacked=False, figsize=(10, 6))
    plt.xlabel('Quarter')
    plt.ylabel('Count')
    plt.title('Count of Each Delay Types with Quarter ')
    plt.legend(title='Delay Status', loc='upper left')

    plt.xticks(rotation=0)
    plt.tight_layout()

    for p in ax.patches:
        height = p.get_height()
        ax.annotate(f'{height}', (p.get_x() + p.get_width() / 2, height),
                    ha='center', va='bottom', fontsize=10)
    plt.show()
```

Dep Time Block

```
[ ]  time_blk = df_pandas.groupby(['DepTimeBlk','DepDelTypes']).size().reset_index(name='Count')

    time_blk_pivot = time_blk.pivot(
        index='DepTimeBlk', columns='DepDelTypes', values='Count').fillna(0)
    # Calculate the total sum of 'DepDelTypes'

    # Add a new column 'Total' containing the total count for each airline
    time_blk_pivot['Total'] = time_blk_pivot.sum(axis=1)


    # Sort the deptimeblock based on the 'Total' count in descending order
    time_blk_pivot = time_blk_pivot.sort_values(
        by='Total', ascending=False)

    # Print the table using tabulate
    print(tabulate(time_blk_pivot, headers='keys', tablefmt='grid'))
```

```
    time_blk = df_pandas.groupby(['DepTimeBlk','DepDelTypes']).size().reset_index(name='Count')

    time_blk_pivot = time_blk.pivot(
        index='DepTimeBlk', columns='DepDelTypes', values='Count').fillna(0)
    # Calculate the total sum of 'DepDelTypes'

    # Add a new column 'Total' containing the total count for each airline
    time_blk_pivot['Total'] = time_blk_pivot.sum(axis=1)


    # Sort the deptimeblock based on the 'Total' count in descending order
    time_blk_pivot = time_blk_pivot.sort_values(
        by='Total', ascending=False)

    # Print the table using tabulate
    print(tabulate(time_blk_pivot, headers='keys', tablefmt='grid'))
```

Month

```
# Month
month_delay_count = df_pandas.groupby(
    ['Month', 'DepDelTypes']).size().reset_index(name='Count')

# Pivot the table to have airlines as rows and delay groups as columns
month_delay_pivot = month_delay_count.pivot(
    index='Month', columns='DepDelTypes', values='Count').fillna(0)

# Add a new column 'Total' containing the total count for each airline
month_delay_pivot['Total'] = month_delay_pivot.sum(axis=1)
month_delay_pivot['Delay_Percentage'] = round((month_delay_pivot['1']
                                               / month_delay_pivot['Total']) * 100, 0)
month_delay_pivot['NoDelay_Percentage'] = round((month_delay_pivot['0']
                                                 / month_delay_pivot['Total']) * 100, 0)

# Sort the airlines based on the 'Total' count in descending order
month_delay_pivot = month_delay_pivot.sort_values(
by='Total', ascending=False)


print(tabulate(month_delay_pivot, headers='keys', tablefmt='grid'))
```

Line Chart (Count of Delay Types with Month)

```
[ ] # Count the occurrences of each delay type for each quarter
    Month_delay_count = df_pandas.groupby(
        ['Month', 'DepDelTypes']).size().unstack(fill_value=0)

    # Create the line plots
    ax = Month_delay_count.plot(kind='line', marker='o', figsize=(10, 6))
    plt.xlabel('Month')
    plt.ylabel('Count')
    plt.title('Count of Each Delay Types with Month')
    plt.legend(title='Delay Type', loc='upper left')
    plt.xticks(df_pandas['Month'].unique())
    plt.tight_layout()

    plt.show()
```

DayofMonth

```
[ ] DayofMonth_delay_count = df_pandas.groupby(
        ['DayofMonth', 'DepDelTypes']).size().reset_index(name='Count')

    DayofMonth_delay_pivot = DayofMonth_delay_count.pivot(
        index='DayofMonth', columns='DepDelTypes', values='Count').fillna(0)

    DayofMonth_delay_pivot['Total'] = DayofMonth_delay_pivot.sum(axis=1)
    DayofMonth_delay_pivot['Delay_Percentage'] = round(
        (DayofMonth_delay_pivot['1'] / DayofMonth_delay_pivot['Total']) * 100, 0)
    DayofMonth_delay_pivot['NoDelay_Percentage'] = round(
        (DayofMonth_delay_pivot['0'] / DayofMonth_delay_pivot['Total']) * 100, 0)

    DayofMonth_delay_pivot = DayofMonth_delay_pivot.sort_values(
        by='1', ascending=False)

    print(tabulate(DayofMonth_delay_pivot, headers='keys', tablefmt='grid'))
```

Day of Week

```
DayofWeek_delay_count = df_pandas.groupby(
    ['DayOfWeek', 'DepDelTypes']).size().reset_index(name='Count')
DayofWeek_delay_pivot = DayofWeek_delay_count.pivot(
    index='DayOfWeek', columns='DepDelTypes', values='Count').fillna(0)
DayofWeek_delay_pivot['Total'] = DayofWeek_delay_pivot.sum(axis=1)
DayofWeek_delay_pivot['Delay_Percentage'] = round(
    (DayofWeek_delay_pivot['1'] / DayofWeek_delay_pivot['Total']) * 100, 0)
DayofWeek_delay_pivot['NoDelay_Percentage'] = round(
    (DayofWeek_delay_pivot['0'] / DayofWeek_delay_pivot['Total']) * 100, 0)
print(tabulate(DayofWeek_delay_pivot, headers='keys', tablefmt='grid'))
```

Stack Bar Chart for DayOfWeek based on Count of Delay Types

```
[ ] DayOfWeek_delay_count = df_pandas.groupby(
        ['DayOfWeek', 'DepDelTypes']).size().unstack(fill_value=0)

    DayOfWeek_total_count = DayOfWeek_delay_count.sum(axis=1)

    # Calculate the percentage of each DepDelType for each day of the week
    DayOfWeek_percentage_real = DayOfWeek_delay_count.div(
        DayOfWeek_total_count, axis=0) * 100

    # Create a horizontal stack bar chart
    ax = DayOfWeek_percentage_real.plot(kind='barh', stacked=True, figsize=(10, 6))
    plt.xlabel('Percentages')
    plt.ylabel('DayOfWeek')
    plt.title('DayOfWeek based on Count of Delay Types')
    plt.legend(title='Delay Status', loc='upper right')
    plt.tight_layout()

    for patch in ax.patches:
        width, height = patch.get_width(), patch.get_height()
        x, y = patch.get_xy()
        ax.annotate(f'{width:.1f}%', (x + width / 2, y +
                    height / 2), ha='center', va='center')

    plt.show()
```

Diverted

```
diverted_true_data = df_pandas[df_pandas['Diverted'] == False]


# Count the occurrences of delay and nondelay groups for diverted flights
diverted_delay_count = df_pandas.groupby(
    ['Diverted', 'DepDelTypes']).size().reset_index(name='Count')

diverted_delay_pivot = diverted_delay_count.pivot(
    index='Diverted', columns='DepDelTypes', values='Count').fillna(0)

# Print the table
print(diverted_delay_pivot)
```

```
# Count the occurrences of delay and nondelay groups based on the diverted
Diverted_delay_count = df_pandas.groupby(
    ['Diverted', 'DepDelTypes']).size().unstack(fill_value=0)

total_diverted = Diverted_delay_count.loc[True].sum()
total_not_diverted = Diverted_delay_count.loc[False].sum()

Diverted_delay_count_percentage = Diverted_delay_count.copy()
Diverted_delay_count_percentage.loc[True] = (
    Diverted_delay_count.loc[True] / total_diverted) * 100
Diverted_delay_count_percentage.loc[False] = (
    Diverted_delay_count.loc[False] / total_not_diverted) * 100

# Create a horizontal grouped bar chart
ax = Diverted_delay_count_percentage.plot(
    kind='bar', stacked=True, figsize=(7, 6), width=0.4)
plt.xlabel('Diverted')
plt.ylabel('Percentage')
plt.title('Percentage of Delay Types Based on Diverted Status')
plt.legend(title='Delay Status', loc='upper right')
plt.tight_layout()

for patch in ax.patches:
    width, height = patch.get_width(), patch.get_height()
    x, y = patch.get_xy()
    ax.annotate(f'{height:.1f}%', (x + width / 2, y +
                height / 2), ha='center', va='center')
plt.xticks(rotation=0)
plt.show()
```

DistanceGroup

```
[ ] DistanceGroup_delay_count = df_pandas.groupby(
        ['DistanceGroup', 'DepDelTypes']).size().reset_index(name='Count')


    # Pivot the table to have airlines as rows and delay groups as columns
    DistanceGroup_delay_pivot = DistanceGroup_delay_count.pivot(
        index='DistanceGroup', columns='DepDelTypes', values='Count').fillna(0)

    # Add a new column 'Total' containing the total count for each airline
    DistanceGroup_delay_pivot['Total'] = DistanceGroup_delay_pivot.sum(axis=1)
    DistanceGroup_delay_pivot['Delay_Percentage'] = round(
        (DistanceGroup_delay_pivot['1'] / DistanceGroup_delay_pivot['Total']) * 100, 0)
    DistanceGroup_delay_pivot['NoDelay_Percentage'] = round(
        (DistanceGroup_delay_pivot['0'] / DistanceGroup_delay_pivot['Total']) * 100, 0)


    # Sort the airlines based on the 'Total' count in descending order
    DistanceGroup_delay_pivot = DistanceGroup_delay_pivot.sort_values(
        by='1', ascending=False)

    # Print the table using tabulate
    print(tabulate(DistanceGroup_delay_pivot, headers='keys', tablefmt='grid',
          colalign=("right", "right", "right", "right", "center", "center")))
```

TaxiOut

```
# Create a taxi out (Delay) histogram
taxi_out_delay = df_pandas.query('(TaxiOut < 100) & (DepDelTypes == "1")')

plt.figure(figsize=(8, 6))
plt.hist(taxi_out_delay['TaxiOut'], bins=60, edgecolor='black')
plt.xlabel('Taxi Out (min)')
plt.ylabel('Frequency(Delay)')
plt.title('Histogram of Taxi Out')
plt.tight_layout()

plt.show()
```

```
[ ] # Create a taxi out (No-Delay) histogram
    taxi_out_delay = df_pandas.query(
        '(TaxiOut < 100) & (DepDelTypes == "0")')


    plt.figure(figsize=(8, 6))
    plt.hist(taxi_out_delay['TaxiOut'], bins=60, edgecolor='black')
    plt.xlabel('Taxi Out (min)')
    plt.ylabel('Frequency(NoDelay)')
    plt.title('Histogram of Taxi Out')
    plt.tight_layout()

    plt.show()
```

Frequency of Delay Types within 1-hour Intervals

```
# Change Datatype
# List of columns to convert to 'string'
columns_to_convert = ['DepTime']

# Convert the selected columns to 'integer' data type using pd.Categorical()
for column in columns_to_convert:
    df_pandas[column] = df_pandas[column].astype('int64')

df_pandas['DepTime'] = df_pandas['DepTime'].astype(str)

# Remove any non-digit characters (e.g., "0") from the 'DepTime' strings
df_pandas['DepTime'] = df_pandas['DepTime'].str.replace(r'\D', '', regex=True)

# Pad the 'DepTime' strings with leading zeros to ensure a 4-digit format
df_pandas['DepTime'] = df_pandas['DepTime'].str.zfill(4)
df_pandas['DepTime'] = df_pandas['DepTime'].replace('2400', '0000')
```

```
[ ] # Group by 'time' and 'deltype' to get the frequency of each combination
    grouped_df = df_pandas.groupby(
        ['DepTime', 'DepDelTypes']).size().reset_index(name='count')

    # Separate delay and no-delay data
    delay_df = grouped_df[grouped_df['DepDelTypes'] == '1']
    nodelay_df = grouped_df[grouped_df['DepDelTypes'] == '0']

    # Set up the plot
    plt.figure(figsize=(20, 6))

    # Plot delay points
    plt.scatter(delay_df['DepTime'], delay_df['count'],
                color='red', label='Delay', s=20)

    # Plot no-delay points
    plt.scatter(nodelay_df['DepTime'], nodelay_df['count'],
                color='blue', label='No-Delay', s=20)
    plt.xticks(range(0, 1500, 60))
    plt.yticks(range(0, max(grouped_df['count']) + 1000, 1000))
    # Customize the plot
    plt.xlabel('Time')
    plt.ylabel('Frequency')
    plt.title('Frequency of Delay Types within 1-hour Intervals')
    plt.legend()

    # Show the plot
    plt.show()
```

Origin

```
    # Count of City
    unique_cities = set(df_pandas['OriginCityName'])
    city_count = len(unique_cities)
    print("Number of cities:", city_count)
```

```
[ ] city_delay_count = df_pandas.groupby(
        ['OriginCityName', 'DepDelTypes']).size().reset_index(name='Count')

    # Pivot the table to have CITY as rows and delay groups as columns
    city_delay_pivot = city_delay_count.pivot(
        index='OriginCityName', columns='DepDelTypes', values='Count').fillna(0)

    # Add a new column 'Total' containing the total count for each city
    city_delay_pivot['Total'] = city_delay_pivot.sum(axis=1)

    city_delay_pivot['Delay_Percentage'] = round(
        (city_delay_pivot[1] / city_delay_pivot['Total']) * 100, 0)
    city_delay_pivot['NoDelay_Percentage'] = round(
        (city_delay_pivot[0] / city_delay_pivot['Total']) * 100, 0)

    # Sort the city based on the 'Total' count in descending order
    city_delay_pivot = city_delay_pivot.sort_values(
        by='Total', ascending=False)

    # Print the table using tabulate
    print(tabulate(city_delay_pivot, headers='keys', tablefmt='grid'))
```

Machine Learning

Random Forest

```
    rf_classifier = RandomForestClassifier(n_estimators=15, random_state=42,max_depth=30)
    rf_classifier.fit(X_train, y_train)
    y_pred = rf_classifier.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    confusion = confusion_matrix(y_test, y_pred)
    report = classification_report(y_test, y_pred, digits=4)

    print(f'Accuracy of Random Forest: {accuracy}')
    print(f'Confusion Matrix:\n{confusion}')
    print(f'Classification Report:\n{report}')
```

Default Parameter (Random Forest)

```
[ ] rf_default_params = rf_classifier.get_params()
    print(rf_default_params)
```

Learning Curve

```
rf_classifier = RandomForestClassifier(n_estimators=15, random_state=42,max_depth=30)
# Generate a learning curve based on log loss
train_sizes, train_scores, test_scores = learning_curve(rf_classifier, X_train, y_train, cv=5, scoring='accuracy', n_jobs=-1)

# Calculate mean and standard deviation for training set scores
train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)

# Calculate mean and standard deviation for test set scores
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

# Plot the learning curve based on log loss
plt.figure(figsize=(10, 6))
plt.plot(train_sizes, train_mean, color='blue', marker='o', markersize=5, label='Training Accuracy')
plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std, alpha=0.15, color='blue')

plt.plot(train_sizes, test_mean, color='green', linestyle='--', marker='s', markersize=5, label='Validation Accuracy')
plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std, alpha=0.15, color='green')

plt.title('Learning Curve for Random Forest Classifier (Accuracy)')
plt.xlabel('Number of Training Samples')
plt.ylabel('Accuracy')
plt.legend(loc='upper right')
plt.grid()
plt.show()
```

Hyperparameter Tuning

```
[ ] # Create a Random Forest classifier
    rf_classifier = RandomForestClassifier()

    # Define the hyperparameter grid
    param_grid = {
      'n_estimators': [15,20,30],
      'max_depth':[30,40]
    }

    # Create GridSearchCV object
    grid_search = GridSearchCV(estimator=rf_classifier, param_grid=param_grid, cv=3, scoring='accuracy', n_jobs=-1)

    # Fit the model to the training data
    grid_search.fit(X_train, y_train)

    # Print the best hyperparameters
    print("Best Hyperparameters:", grid_search.best_params_)

    # Get the best model
    best_rf_model = grid_search.best_estimator_

    # Make predictions on the test set
    y_pred = best_rf_model.predict(X_test)

    # Evaluate the model
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("Classification Report:\n", classification_report(y_test, y_pred))
```

Best Parameter

```
[ ] rf_tuning = RandomForestClassifier(n_estimators=30, random_state=42,max_depth=40)
    rf_tuning .fit(X_train, y_train)
    y_pred = rf_tuning .predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    confusion = confusion_matrix(y_test, y_pred)
    report = classification_report(y_test, y_pred, digits=4)

    print(f'Accuracy of Random Forest: {accuracy}')
    print(f'Confusion Matrix:\n{confusion}')
    print(f'Classification Report:\n{report}')
```

Learning Curve(Tuning)

```
[ ] rf_tuning = RandomForestClassifier(n_estimators=30, random_state=42,max_depth=40)
    # Generate a learning curve based on log loss
    train_sizes, train_scores, test_scores = learning_curve(rf_tuning, X_train, y_train, cv=3, scoring='accuracy', n_jobs=-1)


    # Calculate mean and standard deviation for training set scores
    train_mean = np.mean(train_scores, axis=1)
    train_std = np.std(train_scores, axis=1)

    # Calculate mean and standard deviation for test set scores
    test_mean = np.mean(test_scores, axis=1)
    test_std = np.std(test_scores, axis=1)

    # Plot the learning curve based on log loss
    plt.figure(figsize=(10, 6))
    plt.plot(train_sizes, train_mean, color='blue', marker='o', markersize=5, label='Training Accuracy')
    plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std, alpha=0.15, color='blue')

    plt.plot(train_sizes, test_mean, color='green', linestyle='--', marker='s', markersize=5, label='Validation Accuracy')
    plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std, alpha=0.15, color='green')

    plt.title('Learning Curve for Random Forest Classifier (Accuracy)')
    plt.xlabel('Number of Training Samples')
    plt.ylabel('Accuracy')
    plt.legend(loc='upper right')
    plt.grid()
    plt.show()
```

XGBoost

```
[ ] pip install --upgrade xgboost
```

```
[ ] xgb_classifier = xgb.XGBClassifier()

    # Train the classifier on the training data
    xgb_classifier.fit(X_train, y_train)

    # Make predictions on the test data
    y_pred = xgb_classifier.predict(X_test)

    # Evaluate the classifier's accuracy
    accuracy = accuracy_score(y_test, y_pred)
    confusion = confusion_matrix(y_test, y_pred)
    report = classification_report(y_test, y_pred, digits=4)

    print("Accuracy of XGBoost:", accuracy)
    print("Confusion Matrix:\n", confusion)
    print("Classification Report:\n", report)
```

Default Parameter (XGBoost)

```
[ ] xgb_default_params = xgb_classifier.get_params()
    print(xgb_default_params)
```

Learning Curve

```
# Create an XGBoost classifier
xgb_classifier = xgb.XGBClassifier()

# Generate a learning curve based on log loss
train_sizes, train_scores, test_scores = learning_curve(
    xgb_classifier, X_train, y_train, cv=5, scoring='accuracy', n_jobs=-1)

# Calculate mean and standard deviation for training set scores
train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)

# Calculate mean and standard deviation for test set scores
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

# Plot the learning curve based on log loss
plt.figure(figsize=(10, 6))
plt.plot(train_sizes, train_mean, color='blue', marker='o', markersize=5, label='Training Accuracy')
plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std, alpha=0.15, color='blue')

plt.plot(train_sizes, test_mean, color='green', linestyle='--', marker='s', markersize=5, label='Validation Accuracy')
plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std, alpha=0.15, color='green')

plt.title('Learning Curve for XGBoost Classifier (Accuracy)')
plt.xlabel('Number of Training Samples')
plt.ylabel('Accuracy')
plt.legend(loc='upper right')
plt.grid()
plt.show()
```

Hyperparameter Tuning

```
# Define the XGBClassifier
xgb_classifier = xgb.XGBClassifier()

# Define the parameter grid for GridSearchCV
param_grid = {
    'learning_rate': [0.160, 0.076],
    'n_estimators': [200,380],
    'max_depth': [9],
    'subsample':[1,0.641],
    'gamma':[0.32,0.779],
    'min_child_weight': [0, 1]
}

# Define the scoring metric
scoring = {'Accuracy': make_scorer(accuracy_score)}

# Create the GridSearchCV object
grid_search = GridSearchCV(estimator=xgb_classifier, param_grid=param_grid, scoring=scoring, refit='Accuracy',
                           cv=3, verbose=1, n_jobs=-1)

# Fit the GridSearchCV object to the training data
grid_search.fit(X_train, y_train)

# Get the best parameters and model
best_params = grid_search.best_params_
best_xgb_classifier = grid_search.best_estimator_

# Make predictions on the test data using the best model
y_pred = best_xgb_classifier.predict(X_test)

# Evaluate the classifier's accuracy, confusion matrix, and classification report
accuracy = accuracy_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred, digits=4)

# Print the results
print("Best Hyperparameters:", best_params)
print("Accuracy of XGBoost Classifier:", accuracy)
print("Confusion Matrix:\n", confusion)
print("Classification Report:\n", report)
```

Best Parameter

```
[ ] xgb_classifier_tuning = xgb.XGBClassifier(gamma=0.779, learning_rate=0.16, max_depth=9, min_child_weight=0, n_estimators= 380, subsample=0.641)

    # Train the classifier on the training data
    xgb_classifier_tuning.fit(X_train, y_train)

    # Make predictions on the test data
    y_pred = xgb_classifier_tuning.predict(X_test)

    # Evaluate the classifier's accuracy
    accuracy = accuracy_score(y_test, y_pred)
    confusion = confusion_matrix(y_test, y_pred)
    report = classification_report(y_test, y_pred, digits=4)

    print("Accuracy of XGBoost:", accuracy)
    print("Confusion Matrix:\n", confusion)
    print("Classification Report:\n", report)
```

Learning Curve (Tuning)

```
xgb_classifier_tuning = xgb.XGBClassifier(gamma=0.779, learning_rate=0.16, max_depth=9, min_child_weight=0, n_estimators= 380, subsample=0.641)
# Generate a learning curve based on log loss
train_sizes, train_scores, test_scores = learning_curve(
    xgb_classifier_tuning, X_train, y_train, cv=5, scoring='accuracy', n_jobs=-1)


# Calculate mean and standard deviation for training set scores
train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)

# Calculate mean and standard deviation for test set scores
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

# Plot the learning curve based on log loss
plt.figure(figsize=(10, 6))
plt.plot(train_sizes, train_mean, color='blue', marker='o', markersize=5, label='Training Accuracy')
plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std, alpha=0.15, color='blue')

plt.plot(train_sizes, test_mean, color='green', linestyle='--', marker='s', markersize=5, label='Validation Accuracy')
plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std, alpha=0.15, color='green')

plt.title('Learning Curve for XGBoost Classifier (Accuracy)')
plt.xlabel('Number of Training Samples')
plt.ylabel('Accuracy')
plt.legend(loc='upper right')
plt.grid()
plt.show()
```

Feature Important

```
[ ] feature_importances = xgb_classifier_tuning.feature_importances_

    # Get the names of the features
    feature_names = X_train.columns  # Assuming you have column names if X_train is a DataFrame
    sorted_indices = feature_importances.argsort()
    # Create a bar plot of feature importances
    plt.barh(range(len(feature_importances)), feature_importances[sorted_indices], align='center')
    plt.yticks(range(len(feature_importances)), feature_names[sorted_indices])
    plt.xlabel('Feature Importance')
    plt.title('XGBoost Feature Importances')
    plt.show()
```

Decision Tree

```
[ ] # Create a Decision Tree classifier
    decision_tree_classifier = DecisionTreeClassifier()

    # Train the classifier on the training data
    decision_tree_classifier.fit(X_train, y_train)

    # Make predictions on the testing data
    y_pred = decision_tree_classifier.predict(X_test)

    # Calculate and print the accuracy and classification report of the classifier
    accuracy = accuracy_score(y_test, y_pred)
    confusion = confusion_matrix(y_test, y_pred)
    report = classification_report(y_test, y_pred, digits=4)

    print("Accuracy of Decision Tree Classifier:", accuracy)
    print("Confusion Matrix:\n", confusion)
    print("Classification Report:\n", report)
```

Default Parameter (Decision Tree)

```
[ ] dt_default_params = decision_tree_classifier.get_params()
    print(dt_default_params)
```

```
# Create a Decision Tree classifier
decision_tree_classifier = DecisionTreeClassifier()
# Generate a learning curve based on log loss
train_sizes, train_scores, test_scores = learning_curve(
    decision_tree_classifier, X_train, y_train, cv=5, scoring='accuracy', n_jobs=-1)


# Calculate mean and standard deviation for training set scores
train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)

# Calculate mean and standard deviation for test set scores
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

# Plot the learning curve based on log loss
plt.figure(figsize=(10, 6))
plt.plot(train_sizes, train_mean, color='blue', marker='o', markersize=5, label='Training Accuracy')
plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std, alpha=0.15, color='blue')

plt.plot(train_sizes, test_mean, color='green', linestyle='--', marker='s', markersize=5, label='Validation Accuracy')
plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std, alpha=0.15, color='green')

plt.title('Learning Curve for Decision Tree Classifier (Accuracy)')
plt.xlabel('Number of Training Samples')
plt.ylabel('Accuracy')
plt.legend(loc='upper right')
plt.grid()
plt.show()
```

```
[ ] train_score,test_score=list(),list()
    values=[i for i in range(1,31)]
    for i in values:
      model=DecisionTreeClassifier(max_depth=i)
      model.fit(X_train,y_train)

      train_yhat=model.predict(X_train)
      accuracy = accuracy_score(y_train, train_yhat)
      train_score.append(accuracy)

      test_yhat = model.predict(X_test)
      test_accuracy = accuracy_score(y_test, test_yhat)
      test_score.append(test_accuracy)

    plt.plot(values,train_score,'-o',label='Train')
    plt.plot(values,test_score,'-o',label='Test')
    plt.xlabel('Max_Depth')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.show()
```

Hyperparameter Tuning

```
[ ] # Create a Decision Tree classifier
    decision_tree_classifier = DecisionTreeClassifier()
    # Define hyperparameters and their possible values for tuning
    param_grid = {
        'max_depth': [19, 20, 22],
        'min_samples_leaf':[5,15,20],
        'max_features': ['sqrt', 'log2', None]
    }

    # Create GridSearchCV
    grid_search = GridSearchCV(estimator=decision_tree_classifier, param_grid=param_grid, cv=3, scoring='accuracy', n_jobs=-1)

    # Fit the model to find the best hyperparameters
    grid_search.fit(X_train, y_train)

    # Print the best hyperparameters
    print("Best Hyperparameters:", grid_search.best_params_)

    # Use the best model to make predictions on the test set
    y_pred = grid_search.best_estimator_.predict(X_test)

    # Evaluate the model
    accuracy = accuracy_score(y_test, y_pred)
    print("Accuracy:", accuracy)
```

Best Parameter

```
[ ] # Create a Decision Tree classifier
    decision_tree_tuning = DecisionTreeClassifier(max_depth=22, max_features=None, min_samples_leaf=5)

    # Train the classifier on the training data
    decision_tree_tuning.fit(X_train, y_train)

    # Make predictions on the testing data
    y_pred = decision_tree_tuning.predict(X_test)

    # Calculate and print the accuracy and classification report of the classifier
    accuracy = accuracy_score(y_test, y_pred)
    confusion = confusion_matrix(y_test, y_pred)
    report = classification_report(y_test, y_pred, digits=4)

    print("Accuracy of Decision Tree Classifier:", accuracy)
    print("Confusion Matrix:\n", confusion)
    print("Classification Report:\n", report)
```

Learning Curve (Tuning)

```
[ ] decision_tree_tuning = DecisionTreeClassifier(max_depth=22, max_features=None, min_samples_leaf=5)
    # Generate a learning curve based on log loss
    train_sizes, train_scores, test_scores = learning_curve(
        decision_tree_tuning, X_train, y_train, cv=5, scoring='accuracy', n_jobs=-1)


    # Calculate mean and standard deviation for training set scores
    train_mean = np.mean(train_scores, axis=1)
    train_std = np.std(train_scores, axis=1)

    # Calculate mean and standard deviation for test set scores
    test_mean = np.mean(test_scores, axis=1)
    test_std = np.std(test_scores, axis=1)

    # Plot the learning curve based on log loss
    plt.figure(figsize=(10, 6))
    plt.plot(train_sizes, train_mean, color='blue', marker='o', markersize=5, label='Training Accuracy')
    plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std, alpha=0.15, color='blue')

    plt.plot(train_sizes, test_mean, color='green', linestyle='--', marker='s', markersize=5, label='Validation Accuracy')
    plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std, alpha=0.15, color='green')

    plt.title('Learning Curve for Decision Tree (Accuracy)')
    plt.xlabel('Number of Training Samples')
    plt.ylabel('Accuracy')
    plt.legend(loc='upper right')
    plt.grid()
    plt.show()
```

LightGBM

```
[ ] # Create and train the LightGBM classifier
    lgb_classifier = lgb.LGBMClassifier()
    lgb_classifier.fit(X_train, y_train)

    # Make predictions on the testing data
    y_pred = lgb_classifier.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    confusion = confusion_matrix(y_test, y_pred)
    report = classification_report(y_test, y_pred, digits=4)

    print("Accuracy of LightGBM:", accuracy)
    print("Confusion Matrix:\n", confusion)
    print("Classification Report:\n", report)
```

Default Parameter (LightGBM)

```
[ ] lgm_default_params = lgb_classifier.get_params()
    print(lgm_default_params)
```

Learning Curve

```
lgb_classifier = lgb.LGBMClassifier()
# Create and train the LightGBM classifier
# Generate a learning curve based on log loss
train_sizes, train_scores, test_scores = learning_curve(
    lgb_classifier, X_train, y_train, cv=5, scoring='accuracy', n_jobs=-1)


# Calculate mean and standard deviation for training set scores
train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)

# Calculate mean and standard deviation for test set scores
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

# Plot the learning curve based on log loss
plt.figure(figsize=(10, 6))
plt.plot(train_sizes, train_mean, color='blue', marker='o', markersize=5, label='Training Accuracy')
plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std, alpha=0.15, color='blue')

plt.plot(train_sizes, test_mean, color='green', linestyle='--', marker='s', markersize=5, label='Validation Accuracy')
plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std, alpha=0.15, color='green')

plt.title('Learning Curve for LightGBM Classifier (Accuracy)')
plt.xlabel('Number of Training Samples')
plt.ylabel('Accuracy')
plt.legend(loc='upper right')
plt.grid()
plt.show()
```

Hyperparameter Tuning

```
# Create and train the LightGBM classifier
lgb_classifier = lgb.LGBMClassifier()
# Define the hyperparameter grid
param_grid = {
    'num_leaves': [35,40],
    'max_bin': [400,450],           # Set the largest value for max_bin
    'learning_rate': [0.01,0.1],    # Set the lowest value for learning_rate
    'num_iterations': [200,300],
}

# Create the grid search
grid_search = GridSearchCV(estimator=lgb_classifier, param_grid=param_grid, cv=3, scoring='accuracy')

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print("Best Hyperparameters:", grid_search.best_params_)

# Use the best model to make predictions on the test set
y_pred = grid_search.best_estimator_.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

Best Parameter

```
[ ] # Create and train the LightGBM classifier
    lgb_tuning = lgb.LGBMClassifier(learning_rate= 0.1, max_bin=450, num_iterations=300, num_leaves=40)
    lgb_tuning.fit(X_train, y_train)

    # Make predictions on the testing data
    y_pred = lgb_tuning.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    confusion = confusion_matrix(y_test, y_pred)
    report = classification_report(y_test, y_pred, digits=4)

    print("Accuracy of LightGBM:", accuracy)
    print("Confusion Matrix:\n", confusion)
    print("Classification Report:\n", report)
```

Learning Curve (Tuning)

```
# Generate a learning curve based on log loss
lgb_tuning = lgb.LGBMClassifier(learning_rate= 0.1, max_bin=450, num_iterations=300, num_leaves=40)
train_sizes, train_scores, test_scores = learning_curve(
    lgb_tuning, X_train, y_train, cv=5, scoring='accuracy', n_jobs=-1)

# Calculate mean and standard deviation for training set scores
train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)

# Calculate mean and standard deviation for test set scores
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

# Plot the learning curve based on log loss
plt.figure(figsize=(10, 6))
plt.plot(train_sizes, train_mean, color='blue', marker='o', markersize=5, label='Training Accuracy')
plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std, alpha=0.15, color='blue')

plt.plot(train_sizes, test_mean, color='green', linestyle='--', marker='s', markersize=5, label='Validation Accuracy')
plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std, alpha=0.15, color='green')

plt.title('Learning Curve for LightGBM Classifier (Accuracy)')
plt.xlabel('Number of Training Samples')
plt.ylabel('Accuracy')
plt.legend(loc='upper right')
plt.grid()
plt.show()
```

Feature Important

```
[ ] feature_importances = lgb_tuning.feature_importances_

    # Get the names of the features
    feature_names = X_train.columns  # Assuming you have column names if X_train is a DataFrame
    sorted_indices = feature_importances.argsort()
    # Create a bar plot of feature importances
    plt.barh(range(len(feature_importances)), feature_importances[sorted_indices], align='center')
    plt.yticks(range(len(feature_importances)), feature_names[sorted_indices])
    plt.xlabel('Feature Importance')
    plt.title('LightGBM Feature Importances')
    plt.show()
```

Deployment

Save the best machine learning model

```
# saving the model
pickle_out = open("xgbclassifier.pkl", mode = "wb")
pickle.dump(xgb_classifier_tuning, pickle_out)
pickle_out.close()
```

```
import pickle
import streamlit as st

# loading the trained model
#pickle_in = open('xgbclassifier.pkl', 'rb')
#classifier = pickle.load(pickle_in)
with open('xgbclassifier.pkl', 'rb') as file:
    # Load the pickled model
    classifier = pickle.load(file)
st.cache_data

# defining the function which will make the prediction using the data which the user inputs
def prediction(Airline, OriginCityName, Diverted, DistanceGroup, Quarter,
                    Month, DayofMonth, DayOfWeek, TaxiOut, DepTime,DepTimeBlk):

    # Pre-processing user input
    if Airline == "Air Wisconsin Airlines Corp":
        Airline = 0
    elif Airline == "Alaska Airlines Inc.":
        Airline = 1
    elif Airline == "Allegiant Air":
        Airline = 2
    elif Airline == "American Airlines Inc.":
        Airline = 3
    elif Airline == "Capital Cargo International":
        Airline = 4
    elif Airline == "Comair Inc.":
        Airline = 5
    elif Airline == "Commutair Aka Champlain Enterprises, Inc.":
        Airline = 6
    elif Airline == "Delta Air Lines Inc.":
```

```
def main():
    # front end elements of the web page
    html_temp = """
    <div style ="background-color:blue;padding:13px">
      <h1 style ="color:black;text-align:center;">Fligt Delay Prediction System</h1>
    </div>
    """
    # display the front end aspect
    st.markdown(html_temp , unsafe_allow_html = True)

    # following lines create boxes in which user can enter data required to make prediction
    Airline = st.selectbox('Airline',("Air Wisconsin Airlines Corp","Alaska Airlines Inc.","Allegiant Air","American Airlines Inc.",
    "Capital Cargo International","Comair Inc.",
                                      "Commutair Aka Champlain Enterprises, Inc.","Delta Air Lines Inc.","Empire Airlines Inc.",
                                      "Endeavor Air Inc.",
                                      "Envoy Air","Frontier Airlines Inc.","GoJet Airlines LLC d/b/a United Express","Hawaiian Airlines
                                      Inc.","Horizon Air",
                                      "JetBlue Airways", "Mesa Airlines Inc.","Republic Airlines", "SkyWest Airlines Inc.","Southwest
                                      Airlines Co.","Spirit Air Lines","United Air Lines Inc."))
    OriginCityName = st.selectbox('OriginCityName',("Aberdeen, SD","Abilene, TX","Adak Island, AK","Aguadilla, PR","Akron, OH","Alamosa,
    CO ","Albany, GA","Albany, NY","Albuquerque, NM",
                                                    "Alexandria, LA","Allentown/Bethlehem/Easton, PA","Alpena, MI","Amarillo, TX",
                                                    "Anchorage, AK","Appleton, WI","Arcata/Eureka, CA","Asheville, NC",
                                                    "Ashland, WV","Aspen, CO","Atlanta, GA","Atlantic City, NJ","Augusta, GA","Austin,
```

```
    Diverted= st.selectbox('Diverted',("True","False"))
    DistanceGroup = st.selectbox('Distance Group',("1(0-250 Miles)","2(251-500 Miles)","3(501-750 Miles)","4(751-1000 Miles)","5
    (1001-1250 Miles)",
                                                   "6(1251-1500 Miles)","7(1501-1750 Miles)","8(1751-2000 Miles)","9(2001-2250 Miles)",
                                                   "10(2501-2750 Miles)","11(2751-3000 Miles)"))
    Quarter=st.selectbox('Quarter',("1(Jan-Mar)","2(Apr-Jun)","3(Jul-Sep)","4(Oct-Dec)"))
    Month=st.selectbox('Month',("January","February","March","April","May","June","July","August","September","October", "November",
    "December"))
    DayofMonth = st.number_input("Date", min_value=1, max_value=31, value=1, step=1)
    DayOfWeek = st.selectbox('Day of Week',("Monday","Tuesday","Wednesday","Thrusday","Friday","Saturday","Sunday"))
    TaxiOut = st.number_input("Taxi Out(Min)")
    DepTime = st.number_input("Departure Time",min_value=0, max_value=2359, value=0000, step=1)
    DepTimeBlk = st.selectbox('Time Block',("0001-0559","0600-0659","0700-0759","0800-0859","0900-0959","1000-1059","1100-1159",
                                            "1200-1259","1300-1359","1400-1459","1500-1559","1600-1659","1700-1759", "1800-1859",
                                            "1900-1959","2000-2059","2100-2159","2200-2259","2300-2359"))
    result =""

    # when 'Predict' is clicked, make the prediction and store it
    if st.button("Predict"):
        result = prediction(Airline, OriginCityName, Diverted, DistanceGroup, Quarter,Month, DayofMonth, DayOfWeek, TaxiOut, DepTime,
        DepTimeBlk)

        if result == 1:
            st.error('Your Flight is Potentially Delay !!!!')
        else:
           st.success('Your Flight is No Delay.')

if __name__=='__main__':
    main()
```

Figure 134: Code Implementation