**PREDICTING FLIGHT DELAYS**

**IN THE UNITED STATES**

CAPSTONE PROJECT IN BUSINESS ANALYTICS

Authors: Harshitha Mayya, Sofiya Koretskaya,

Navya Gangadharappa Ramesh, and Yada So

Submission Date: December 05, 2023

**Contents**

[**1.** **Abstract** 4](#_Toc152685780)

[**2.** **Introduction** 4](#_Toc152685781)

[**3.** **Description of Datasets and Data Exploration** 4](#_Toc152685782)

[**4.** **Literature Review** 7](#_Toc152685783)

[**4.1. Flight Delay Prediction From Spatial And Temporal Perspective** 7](#_Toc152685784)

[**4.2 A Deep Learning Approach for Flight Delay Prediction Through Time-Evolving Graphs** 8](#_Toc152685785)

[**4.3 How Machine Learning can be utilized for flight delay prediction** 8](#_Toc152685786)

[**4.4 Investigating the Costs and Economic Impact of Flight Delays in the United States** 9](#_Toc152685787)

[**4.5 Flight Delay Prediction Based on Aviation Big Data and Machine Learning** 9](#_Toc152685788)

[**4.6 Flight delay prediction based with Machine Learning** 9](#_Toc152685789)

[**4.7 A Deep Learning Approach to Flight Delay Prediction** 10](#_Toc152685790)

[**4.8 A Data Mining Approach to Flight Arrival Delay Prediction for American Airlines** 10](#_Toc152685791)

[**5.** **Data Pre-Processing** 10](#_Toc152685792)

[**6.** **Data Mining Models and Evaluations** 13](#_Toc152685793)

[**6.1 Random Forests** 13](#_Toc152685794)

[**6.2 Decision Tree** 14](#_Toc152685795)

[**6.3 Logistic Regression** 16](#_Toc152685796)

[**6.4 Naïve Bayesian** 17](#_Toc152685797)

[**6.5 SVM** 18](#_Toc152685798)

[**6.6 Neural Networks** 19](#_Toc152685799)

[**6.7 Top 3 Performing Models** 20](#_Toc152685800)

[**7.** **Discussion** 21](#_Toc152685801)

[**7.1 Domain Knowledge** 21](#_Toc152685802)

[**7.2 Methodological Contributions** 21](#_Toc152685803)

[**8.** **Conclusions** 22](#_Toc152685804)

[**9.** **References** 22](#_Toc152685805)

[**10.** **Appendices** 24](#_Toc152685806)

[**10.1 Data Dictionary** 24](#_Toc152685807)

[**10.2 Python Code** 25](#_Toc152685808)

[a. Data Pre-Processing - Merging Weather Data of Different Airports 25](#_Toc152685809)

[b. Data Pre-Processing - Weather Data 27](#_Toc152685810)

[c. Data Pre-Processing - Flight Data 32](#_Toc152685811)

[d. Data Pre-Processing - Merging Weather and Flight Data 43](#_Toc152685812)

[e. Model Data - Original Data, then Data Imbalance 47](#_Toc152685813)

[f. Model Data - Data Imbalance, then Primary Component Analysis 50](#_Toc152685814)

[g. Model Data - Data Imbalance, then Feature Selection 60](#_Toc152685815)

[h. Model Data - PCA, then Data Imbalance 68](#_Toc152685816)

[i. Model Data - Feature Selection, then Data Imbalance 70](#_Toc152685817)

[j. Selecting the Best Dataset 72](#_Toc152685818)

[k. Parameter Tuning for SVM 86](#_Toc152685819)

[m. Parameter Tuning for Neural Network 88](#_Toc152685820)

[n. Selecting the Best Models 91](#_Toc152685821)

# **Abstract**

Flight delays can have a significant impact on the aviation industry's economy, making it essential to develop efficient air traffic management strategies. This research paper presents a study on predicting flight delays for six major airlines in the United States using machine learning models.

The study evaluates the impact of data balancing techniques and feature reduction on predictive performance. The results show that the machine learning models can accurately predict flight delays, and the use of data balancing techniques and feature reduction can improve the models' performance. These findings can help the aviation industry develop more efficient air traffic management strategies to reduce flight delays and their economic costs. By predicting flight delays accurately, airlines can take proactive measures to minimize the impact of delays on passengers and the industry.

The study's methodology involved collecting flight data from six major airlines in the United States and using machine learning algorithms to predict flight delays. The models' performance was evaluated using various metrics, including accuracy, precision, recall, and F1 score.

Overall, our study demonstrates the potential of machine learning models in predicting flight delays and improving air traffic management strategies. The findings can help airlines and other stakeholders in the aviation industry make informed decisions to reduce flight delays and their economic costs.

# **Introduction**

Flight delays can incur a significant amount of costs to the economy. To reduce this wasted cost, analysis and prediction of air traffic delays is required to build more efficient and mitigating air traffic management strategies.

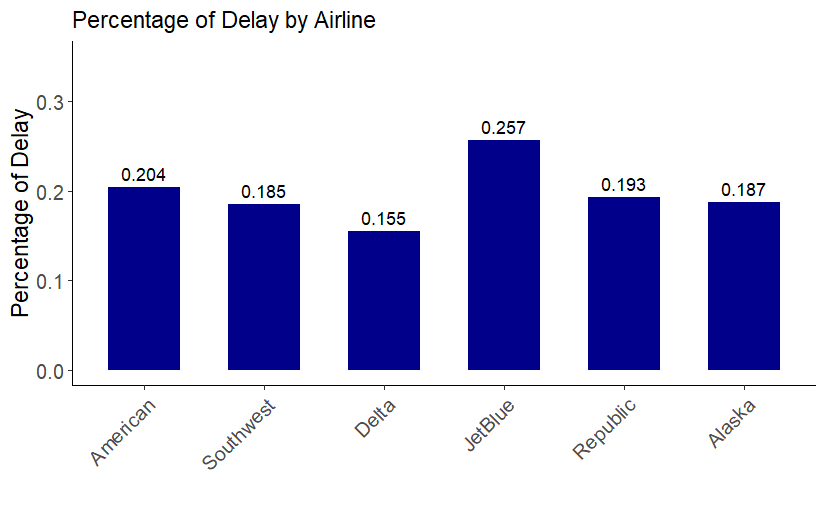
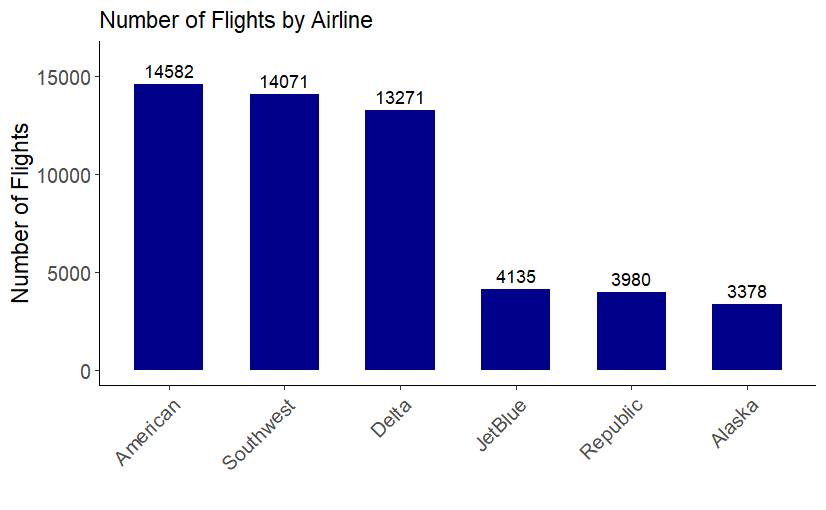
In this study, we will build machine learning models for predicting the flight delays of 6 airlines operated in United States. We will also study whether our results can be improved after we balance our dataset using Naïve Random Over-Sampling and SMOTE.

The US Bureau of Transportation Statistics has received reports on around 200 million domestic US flights, which are included in the Reporting Carrier On-Time Performance Dataset. The dataset includes fundamental statistics regarding every flight, including date, time, destination, and departure airports) and, if relevant, the duration of the flight was postponed and details regarding the cause of the postponement. Utilizing this dataset allows you to estimate the probability that a flight will arrive on schedule.

# **Description of Datasets and Data Exploration**

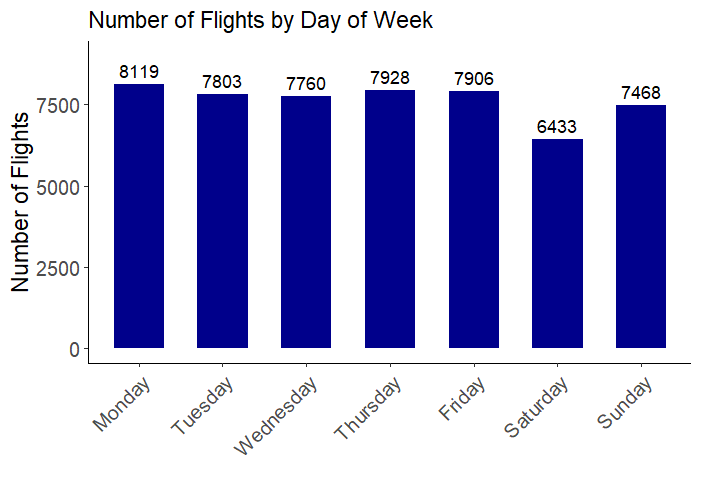
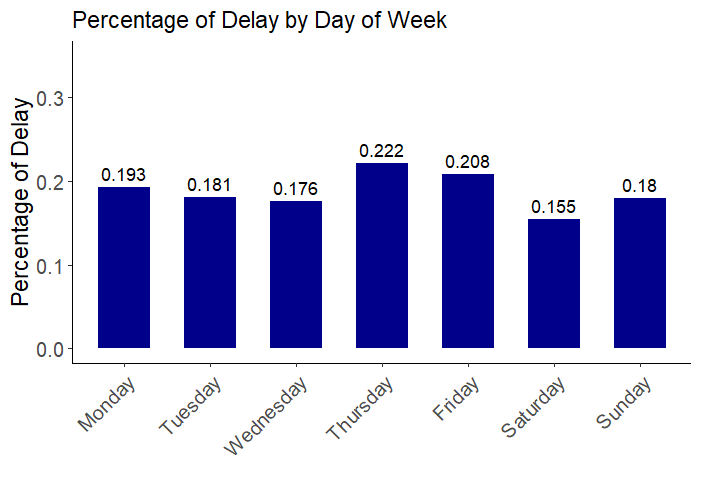
The flight data can be downloaded from the U.S. Department of Transportation. However, since the original dataset was too big for our personal computer to process, we downloaded the smaller sample dataset created by IBM for data scientists to practice machine learning. To further ease the computation time on our computer, we selected only flight data from six airlines. We gathered the weather data from the Integrated Surface Database (ISD) of National Oceanic and Atmospheric Administration (NOAA). The weather conditions at the origin and the destination airports should be important factors for predicting flight delays. Thus, in our model, we merged the flight data with the weather data to get the weather conditions at both the origin and the destination airports. The below table shows the details of the flight data and weather data used.

|  |  |
| --- | --- |
| Airlines | Southwest, Delta, American Airlines, Republic, JetBlue, and Alaska |
| Time Period | Jan 2018 – Dec 2020 |
| Attributes of Flight Data | Year, Month, Day of Month, Day of Week, Airlines, Origin Airport, Destination Airport, Departure Time Bulk, Arrival Time Bulk |
| Attributes of Weather Data | Windspeed, Cloud height, Visibility, Temperature, Rain, Snow |
| Classification (Dependent Variable) | Arrival Delay (1 if the flight is delayed for 15 minutes or more, and 0 otherwise) |



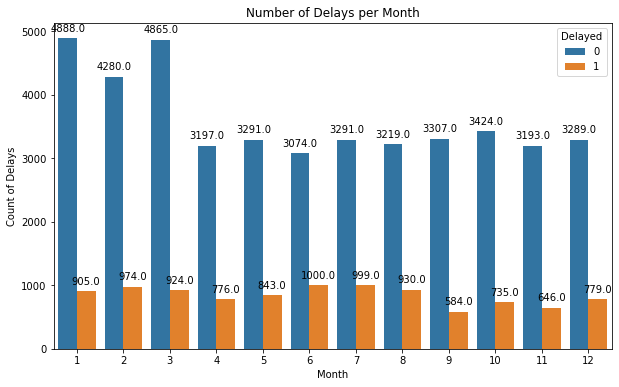
Graph 1 Graph 2

Graph 1 shows that American Airlines has the largest number of flights operated from 2018 to 2020, followed by Southwest Airlines and Delta Airlines. The other three airlines are relatively small compared with those 3 majors airlines in the United States. Graph 2 shows the percentage of delays by each airline, calculated by dividing the number of delayed flights by the number of total flights. Delta Airlines has the lowest percentage of delays and the worst airline in managing delays is JetBlue.



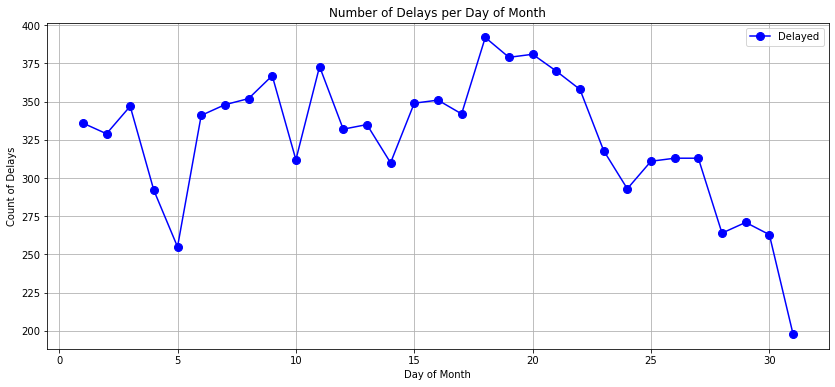
Graph 3 Graph 4

Graph 3 shows the number of flights by day of the week, and we can understand that weekdays are relatively busier than weekends. Graph 4 shows the percentage of delays by day of the week. It clearly demonstrates that Thursdays and Fridays are the worst days to travel, given that they have the highest percentage of delays.



Graph 5

Graph 5 shows the number of delays per month. We can infer that the number of delays is more in the winter quarter, especially in January and March. This could be due to the weather conditions.



Graph 6

Graph 6 shows the number of delays for day of the month. We can infer that the number of delays is more during the 18th – 20th day of the month. Both customers and the airlines/airports can plan better during these days well in advance.

# **Literature Review**

## **4.1. Flight Delay Prediction From Spatial And Temporal Perspective**

This study, which was published in November 2022. Random Forest, LSTM method (a kind of recurrent neural network (RNN) architecture that processes input data in both forward and backward directions), and Complex Network Theory were employed by the authors in this instance. The aviation network's spatial properties will be extracted at the edge, node, and network levels using complex network theory in order to predict flight delays. Random Forest is one such method. Furthermore, taking into account the temporal association between weather and airport crowdedness on flight delays, they create a prediction framework based on LSTM units to extract the temporal attribute of crowdedness and weather condition. In this study which was collected departure and arrival flight data that traveled through Chinese airspace in June and August of 2016 using VariFlight (https://data.variflight.com). 762,415 samples connecting 260 airports make up the data set. This data contains a wide range of information, such as the carriers, the origin and destination airports, the tail number, the real departure and arrival times in addition to the scheduled schedules, and the weather. All flights experienced a cumulative delay of 40.47 minutes throughout the research period.

The entire data set is randomly divided into three sub-datasets for this study: 10% is used for model validation, 10% is used for model testing, and the remaining 80% is used for training the model. Our suggested model's accuracy is 92.39%. About 86% of the samples that are received on time are correctly classified; for the samples that are received later, the classification accuracy is up to 95%. Long-short term memory (LSTM) architecture was used in this work to capture the temporal correction. In order to address the vanishing gradient issue with conventional RNNs, Hochreiter and Schmidhuber (1997) introduced the LSTM, a unique type of Recurrent Neural Network (RNN).

## **4.2 A Deep Learning Approach for Flight Delay Prediction Through Time-Evolving Graphs**

The flight delay prediction problem is investigated from a network perspective (i.e., multi-airport scenario). To model the time-evolving and periodic graph-structured information in the airport network, a flight delay prediction approach based on the graph convolutional neural network (GCN) is developed in this paper. The data used in this paper are provided by CAAC, comprising all domestic flights from April 1, 2018 to October 31, 2018 (i.e., Summer-Autumn flight season). Considering the sparse flight schedules at spoke airports, 224 civil airports are ranked according to the handling capacity and the top 74 busy airports in China in 2018. The raw data are normalized by Z-Score method and 70% of the dataset is employed for training, 15% are used for testing while the remaining 15% for validation. Here all experiments are tested on a Linux cluster (i.e., CPU: Intel (R) Xeon (R) Gold 6126 CPU @ 2.60GHz, GPU: NVIDIA TITAN RTX). Along with that they have used the below models in their paper: For Single-Airport Scenario Models they have used ARIMA which is Auto-Regressive Integrated Moving Average, which is one of the most popular methods in time-series prediction tasks. Also, SVR which is Support Vector Regression with Radial Basis Function Kernel for the penalty term. For Multi-Airport Scenario Models, they have used DCRNN that is Diffusion Convolutional Recurrent Neural Network (A neural sequence model that learns to forecast on a directed graph) and STGCN - Spatial-temporal Graph Convolutional Network (the common convolution on graphs which considers the spatial dependencies between nodes in the graph).

## **4.3 How Machine Learning can be utilized for flight delay prediction**

Flight delays are a major pain point in the aviation industry, affecting over 23% of domestic flights in the US alone. They disrupt schedules, incur costs, and cause frustration. Predicting these disruptions is a challenge, but machine learning is stepping up to the plate.

By analyzing historical data, including flight schedules, weather, and air traffic control information, ML models can accurately predict the likelihood of delays. This information empowers airlines and passengers:

**For Airlines:**

* Improved operational planning: Rebook passengers proactively before delays occur, minimizing disruption and cost.
* Resource allocation optimization: Optimize crew and aircraft scheduling based on predicted delays.

**For Passengers:**

* Informed travel planning: Choose flights with lower probabilities of delay.
* Alternative flight options: Explore alternative routes or airlines if delays are likely.

Machine learning is unlocking a new era of efficiency and predictability in air travel. By harnessing its power, we can all experience smoother, less stressful journeys.

## **4.4 Investigating the Costs and Economic Impact of Flight Delays in the United States**

A 2019 study by Shahrivar, Hall, and Peterson investigates the impact of flight delays in the US. Analyzing data from multiple sources, they estimate both direct (e.g., fuel costs) and indirect (e.g., lost productivity) costs.

Their findings are stark: flight delays cost the US economy a staggering $31.2 billion annually. The burden isn't evenly distributed, with smaller airlines, smaller airports, and frequent business travelers disproportionately affected. The study urges policymakers and airlines to take action. Investments in infrastructure and improved management practices can help reduce delays. Airlines should consider compensating passengers for the indirect costs they incur.

By tackling this issue, we can make air travel smoother and generate significant economic benefits.

## **4.5 Flight Delay Prediction Based on Aviation Big Data and Machine Learning**

This paper explores a broader scope of factors which may potentially influence the flight delay and compares several machine learning-based models in designed generalized flight delay prediction tasks. To build a dataset for the proposed scheme, automatic dependent surveillance broadcast (ADS-B) messages are received, pre-processed, and integrated with other information such as weather conditions, flight schedule, and airport information. They experimented with different models such as K-NN, SVM, naïve Bayes classifier and random forests. It showed that long short-term memory (LSTM) can handle the obtained aviation sequence data. For the LSTM, 3, 5 and 7 memory depths were chosen, and the testing dataset accuracy (36.53%, 32.8% and 31.2% respectively) got worse dramatically compared to those for the training set resulting in overfitting. Further addition of a dropout layer alleviated the overfitting problem to some extent. However, it suggests that the overfitting problem persisted due to the limited training data. Further, the researchers established that Random forest-based models can obtain higher prediction accuracy of 90.2% for the binary classification and can overcome the overfitting problem. The overfitting problem is due to the limited training data. The ADS-B message dataset used is divided by flight number and date, and then is filtered according to the flight’s height (data below 1500 feet are ignored). The full dataset generated includes 5761 items and their period is from December 2018 to May 2019.

## **4.6 Flight delay prediction based with Machine Learning**

The researchers highlight that correct predictions of flight delays result in concrete advantages such as lower costs, higher customer satisfaction and improve the aviation business. This paper uses three advanced Gradient Boosting Machine learning techniques such as XGBoost, LightGBM, and CatBoost. They used Bayesian technique for hyper-parameter settings. In addition, the Synthetic Minority Over-Sampling Technique (SMOTE) technique is also used. The results are analyzed and shared with and without SMOTE. All three models worked better without SMOTE. This could be due to implementation of cross-validation or the use of advanced models. XGBoost performed the best without SMOTE with an accuracy of 96.9% followed by LightGBM and CatBoost with an accuracy of 96.7% and 94.7% respectively. They have used flight data of a Turkish airline company. The data set consists of the daily flights of the company from 2018. This data is merged with weather condition information that is matched with the flights, which includes instant weather information that occurs in the closest time zone to the flight departure time. The aim of the study is to predict whether a planned flight will be delayed or not. According to the international rules, if the time difference between actual departure and scheduled departure is greater than 15 minutes the flight is labeled as delayed. There are 18148 observations in which 5717 are delayed and 12431 are on-time flights.

## **4.7 A Deep Learning Approach to Flight Delay Prediction**

Kim et al were the first to apply deep learning algorithms to the analysis of air traffic data. Since Recurrent Neural Networks (RNN) has shown great accuracy in modeling sequential data, it was adopted in their analysis. Specifically, Long Short-Term Memory (LSTM) networks were selected because they are faster and more accurate than standard RNNs. Their proposed model has two stages. First, they predict daily delay status at each airport using deep RNN. Then, they predict delays of individual flights using daily delay status, which is the output from the first stage, historical on-time flight performance data, and weather data. They collected flight and weather data from 10 major airports, which are Atlanta, Los Angeles, Chicago, Dallas, Denver, New York, San Francisco, Charlotte, Las Vegas, Phoenix from Jan 2010 until Aug 2015. The authors observed that increasing the number of layers improved accuracy, and increasing epochs also improved accuracy. Their model achieved the best accuracy of 87.42%, when the number of layers is 5 (number of hidden nodes for each layer are: 133, 300, 200, 100, 15 respectively) and the number of epochs is 228.

## **4.8 A Data Mining Approach to Flight Arrival Delay Prediction for American Airlines**

Chakrabarty analyzed flight information of US domestic flights operated by American Airlines, covering top 5 busiest airports of the US, and predicted possible arrival delays of the flight using the Gradient Boosting Classifier Model, achieving a maximum accuracy of 85.73%. Chakrabarty selected the American Airlines because it was one of the world largest airlines in terms of number of destinations served but it didn’t live up to its expectation in terms of punctuality or on-time performance. The 5 airports included in this analysis are Hartsfield-Jackson Atlanta International Airport, Los Angeles International Airport, Ohare International Airport, Dallas/Fort Worth International Airport and John F. Kennedy International Airport. The raw data was downloaded from US Department of Transportation’s Bureau of Transportation Statistics from 2015 to 2016, containing information such as Year, Quarter, Month, Day of Month, Day of Week, Flight Number, Origin Airport, Destination Airport, etc. The data has 97,360 samples with 12 attributes and 1 dependent variable, which is arrival delay. Randomized-Synthetic Minority Over-Sampling Technique was used to resolve the data imbalance issue of the dependent variable, arrival delay. Grid Search was also employed to find the best set of hyper-parameters, and Chakrabarty found that 300 estimators and maximum depth of 5 were the best set. The accuracy of the model for test data was 85.73% using the SMOTE data set. However, without SMOTE, the accuracy dropped to 80.18%. In conclusion, Chakrabarty concluded that Over-Sampling the imbalanced dataset helped improve the model performance.

# **Data Pre-Processing**

The weather data is available for downloading for one airport for one year only at a time. We downloaded the data for each airport for each year individually and then merged them together because they had the same columns. All missing values for windspeed, cloud height, visibility, temperature, rain, and snow information were replaced by their respective modes.

In our original flight dataset, we had 18 airlines. To ease the processing burden on our computer when running machine learning models, we selected only 6 airlines, which are Southwest Airlines, Delta Airlines, American Airlines, Republic Airlines, JetBlue Airways, and Alaska Airlines. We also reduced the number of airports from 363 to 39 only.

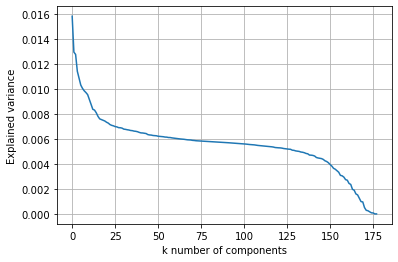
After cleaning both the weather dataset and the flight dataset, we merged the two datasets together using date, hour, and airport. Then, we had to convert our categorical variables to dummy variables using one-hot encoding. Those variables were Year, Month, Day of Month, Day of Week, Airline, Origin Airport, Destination Airport, Departure Time Bulk, and Arrival Time Bulk. After this, our dataset has 53,413 rows, 178 attributes, and 1 dependent variable (Arrival Delay).

However, in our dataset, the number of samples with the value of Arrival Delay equaling to 1 is only 10,095, compared with 43,318 when the value of Arrival Delay equaling to 0. Without balancing this dataset, we tried to run Random Forests and achieved a very low score for recall, which is a typical result for imbalanced dataset.

To resolve this issue, we tried two techniques. The first one is Naïve Random Over-Sampling and the other one is Synthetic Minority Over-Sampling Technique (SMOTE). Either technique increased the number of samples with the value of Arrival Delay equaling to 1, from 10,095 to 43,318. We will compare which technique is better for our dataset.

To further reduce the computation time when running machine learning models, we also tried to reduce the number features in our dataset using Primary Component Analysis, and feature selection (feature importance).

After running the Primary Component Analysis, the relationship between the number of components and the explained variance can be shown in the graph below. In this case, 50 seems to be the elbow point in the graph. Thus, we selected 50 components.



Then, we tested whether the new dataset with only 50 features can produce comparable results with the original data with 178 features. Using Random Forests, we observed that both datasets produced an accuracy of 81%.

Similarly, we reduced the number of features to 50 with feature selection so that we can compare whether feature selection or Primary Component Analysis perform better. Using Random Forests again, the dataset from feature selection also produced an accuracy of 81%.

In selecting the best dataset for our models, we have several choices. First, we need to choose between our original dataset, our balanced dataset using Naïve Random Over-Sampling, and our balanced dataset using SMOTE. In reducing the number of features, we need to choose between feature selection and Primary Component Analysis. We also have to consider whether we should balance the dataset first then reduce the number of features, or whether we should reduce the number of features first then balance the data.

Combining K-Fold Cross Validation (n=5) with Random Forests, we tested all available datasets, and their scores are reported below.

|  |  |  |
| --- | --- | --- |
| **No.** | **Description of Dataset** | **Score (Cross Validation)** |
| 1 | Original Dataset | 0.81 |
| 2 | Dataset with Naïve Random only | 0.97 |
| 3 | Dataset with SMOTE only | 0.87 |
| 4 | Dataset with PCA only | 0.81 |
| 5 | Dataset with PCA first, then Naïve Random | 0.98 |
| 6 | Dataset with PCA first, then SMOTE | 0.89 |
| 7 | Dataset with Naïve Random first, then PCA | 0.98 |
| 8 | Dataset with SMOTE first, then PCA | 0.88 |
| 9 | Dataset with feature selection only | 0.81 |
| 10 | Dataset with FS first, then Naïve Random | 0.97 |
| 11 | Dataset with FS first, then SMOTE | 0.85 |
| 12 | Dataset with Naïve Random first, then FS | 0.97 |
| 13 | Dataset with SMOTE first, then FS | 0.86 |

From the results above, we can see that the 5th dataset and the 7th dataset produce similar results. For the dataset to be used for selecting the best machine learning algorithms for our dataset, we will use the 5th dataset, which is the result from running the PCA first, followed by Naïve Random Over-Sampling.

# **Data Mining Models and Evaluations**

We have performed the below machine learning models on our dataset. We have used Random Forests, Logistic Regression, Decision tree, Naïve Bayes, SVM, and Neural Networks. Our computer specifications are 11th Gen Intel(R) Core (TM) i5-1135G7 @ 2.40GHz 2.40 GHz with 8.00 GB Installed RAM.

## **6.1 Random Forests**

First, we use our 4th dataset (with PCA only) to train our Random Forests model, then use the same dataset to calculate accuracy, recall, and precision for both the training data and test data. The results are shown below.

|  |  |  |
| --- | --- | --- |
| Results from Training Data | | |
| Arrival Delays | Precision | Recall |
| 0 | 1.00 | 1.00 |
| 1 | 1.00 | 1.00 |
| Accuracy | | 1.00 |

|  |  |  |
| --- | --- | --- |
| Results from Test Data | | |
| Arrival Delays | Precision | Recall |
| 0 | 0.81 | 1.00 |
| 1 | 0.61 | 0.01 |
| Accuracy | | 0.81 |

We can clearly see that without balancing the data, our results are very poor in terms of recall. As shown above, recall (Arrival Delays = 1) is only 1%. Next, we will run the model using our 5th dataset (with PCA first, followed by Naïve Random) to see how our results will improve.

|  |  |  |
| --- | --- | --- |
| Results from Training Data | | |
| Arrival Delays | Precision | Recall |
| 0 | 1.00 | 1.00 |
| 1 | 1.00 | 0.98 |
| Accuracy | | 1 |

|  |  |  |
| --- | --- | --- |
| Results from Test Data | | |
| Arrival Delays | Precision | Recall |
| 0 | 0.99 | 0.98 |
| 1 | 0.93 | 0.97 |
| Accuracy | | 0.98 |

As shown above, we observed significant improvement when we used the balanced dataset to train our model. It is worth noting that we only used the balanced dataset to train our model but switched to our 4th dataset (with PCA only without balancing our dataset) to calculate the accuracy, precision, and recall for both our training data and test data.

## **6.2 Decision Tree**

Similarly, we used the 4th dataset (with PCA only) to train our Decision Tree model, and then used the same dataset to calculate accuracy, precision, and recall. Results are shown below.

|  |  |  |
| --- | --- | --- |
| Results from Training Data | | |
| Arrival Delays | Precision | Recall |
| 0 | 1.00 | 1.00 |
| 1 | 1.00 | 1.00 |
| Accuracy | | 1.00 |

|  |  |  |
| --- | --- | --- |
| Results from Test Data | | |
| Arrival Delays | Precision | Recall |
| 0 | 0.82 | 0.80 |
| 1 | 0.23 | 0.25 |
| Accuracy | | 0.70 |

The accuracy from the training data was 99%, while the accuracy for the test data was only 70%. This suggests that our Decision Tree model has overfitted the data. Again, we will run the model using our 5th dataset (with PCA first, followed by Naïve Random) to see how our results will improve.

|  |  |  |
| --- | --- | --- |
| Results from Training Data | | |
| Arrival Delays | Precision | Recall |
| 0 | 1.00 | 0.99 |
| 1 | 0.97 | 0.99 |
| Accuracy | | 0.99 |

|  |  |  |
| --- | --- | --- |
| Results from Test Data | | |
| Arrival Delays | Precision | Recall |
| 0 | 0.99 | 0.83 |
| 1 | 0.57 | 0.98 |
| Accuracy | | 0.86 |

Like the Random Forests model, we observed significant improvement when we used the balanced dataset to train our model.

## **6.3 Logistic Regression**

Results from using the 4th dataset (with PCA only) to train our Logistic Regression model are shown below.

|  |  |  |
| --- | --- | --- |
| Results from Training Data | | |
| Arrival Delays | Precision | Recall |
| 0 | 0.81 | 1.00 |
| 1 | 0.66 | 0.01 |
| Accuracy | | 0.81 |

|  |  |  |
| --- | --- | --- |
| Results from Test Data | | |
| Arrival Delays | Precision | Recall |
| 0 | 0.81 | 1.00 |
| 1 | 0.67 | 0.01 |
| Accuracy | | 0.81 |

Unlike Random Forests and Decision Tree model, we didn’t see any overfitting problem. Accuracy for both the training data and test data is 81%. Using our 5th dataset (with PCA first, followed by Naïve Random) to train the model, we obtained the following results.

|  |  |  |
| --- | --- | --- |
| Results from Training Data | | |
| Arrival Delays | Precision | Recall |
| 0 | 0.87 | 0.60 |
| 1 | 0.27 | 0.63 |
| Accuracy | | 0.60 |

|  |  |  |
| --- | --- | --- |
| Results from Test Data | | |
| Arrival Delays | Precision | Recall |
| 0 | 0.88 | 0.59 |
| 1 | 0.27 | 0.65 |
| Accuracy | | 0.60 |

It is interesting to observe the results using the balanced dataset are worse than those using the imbalanced dataset. The balanced dataset produced an accuracy of 60% only for the test data, while the imbalanced dataset produced 81%.

## **6.4 Naïve Bayesian**

Results from using the 4th dataset (with PCA only) to train our Naïve Bayesian model are shown below.

|  |  |  |
| --- | --- | --- |
| Results from Training Data | | |
| Arrival Delays | Precision | Recall |
| 0 | 0.82 | 0.98 |
| 1 | 0.36 | 0.05 |
| Accuracy | | 0.80 |

|  |  |  |
| --- | --- | --- |
| Results from Test Data | | |
| Arrival Delays | Precision | Recall |
| 0 | 0.82 | 0.98 |
| 1 | 0.34 | 0.05 |
| Accuracy | | 0.80 |

Like Logistic Regression model, Naïve Bayesian model didn’t produce any overfitting problems. Using our 5th dataset (with PCA first, followed by Naïve Random) to train the model, we obtained the following results.

|  |  |  |
| --- | --- | --- |
| Results from Training Data | | |
| Arrival Delays | Precision | Recall |
| 0 | 0.87 | 0.62 |
| 1 | 0.27 | 0.60 |
| Accuracy | | 0.62 |

|  |  |  |
| --- | --- | --- |
| Results from Test Data | | |
| Arrival Delays | Precision | Recall |
| 0 | 0.87 | 0.62 |
| 1 | 0.27 | 0.61 |
| Accuracy | | 0.62 |

We observed that the results using the balanced dataset are worse than those using the imbalanced dataset. The balanced dataset produced an accuracy of 62% only for the test data, while the imbalanced dataset produced 80%.

## **6.5 SVM**

For Support Vectop Machine (SVM) we decided to use PCA dataset, which overall took 13,568.68 seconds (about 4 hours) to train. This is a relatively long training time, but it is worth it for the high accuracy that the model achieves.

At first, we used a gamma value of 1000, and the entire code produced a few results and then stopped. Afterwards, we decided to change values of C and g to the once below:

C\_vals = np.logspace(-3,2,10)  
g\_vals = np.logspace(-3,2,10)

The best model result for SVM had the accuracy of 99%. This is a very good score, and it indicates that the model is able to generalize well to new data.

The best model has the following parameters:

* Kernel: rbf
* Gamma: 0.5994842503189409
* C: 27.825594022071257

The rbf kernel is a nonlinear kernel that is well-suited for learning complex relationships in the  
data. The gamma parameter controls the smoothness of the kernel function, and the C parameter  
controls the trade-off between model accuracy and complexity.

Moreover, the test results show that the SVM model is able to learn from a relatively small  
dataset. The model was trained on a dataset of only 10 candidates, and it was able to achieve an  
accuracy of over 99%.

Overall, the results of the SVM cross-validation test are very positive. The model has a high  
accuracy and can generalize well to new data. This suggests that the SVM model could be  
used to effectively classify new candidates.

## **6.6 Neural Networks**

We experimented with various combinations of both single-layer and two-layer configurations. The optimal hyperparameter combination resulting in the best performance is as follows:

Best Estimator:

The most effective MLP classifier is configured with:

* Activation function: 'tanh'
* Hidden layer sizes: (125, 150)
* Maximum number of iterations: 21,000

Best Score:

The MLP classifier, utilizing the above hyperparameters, achieved the highest cross-validated score of 0.89.

Total Run Time:

The Randomized Search process, spanning 4930.51 seconds (approximately 1.5 hours), reflects the exhaustive exploration of the specified hyperparameter space.

In summary, our investigation revealed that the MLP classifier, employing a hyperbolic tangent ('tanh') activation function, two hidden layers with 125 and 150 neurons in each hidden layer, and a maximum of 21,000 iterations, produced the optimal cross-validated score. The extended run time underscores the thorough search conducted across the hyperparameter configurations. It is worth noting that we only used the balanced dataset to train our model but switched to our 4th dataset (with PCA only without balancing our dataset) to calculate the accuracy, precision, and recall for both our training data and test data.

## **6.7 Top 3 Performing Models**

When comparing machine learning models, K-Fold Cross Validation is a valuable technique for model evaluation because it provides a more reliable and robust estimate of a model's performance, maximizes data usage, and helps detect overfitting. It is widely used in the machine learning community to ensure that the reported performance metrics are representative of a model's ability to generalize to new, unseen data.

The mean score of each model and the computation time are reported in the table below.

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Model | Mean Score (Fold =5) | Computation Time (seconds) |
| 1 | SVM | 0.9941 | 10,000 |
| 2 | Random Forests | 0.9852 | 300 |
| 3 | Neural Networks | 0.8953 | 250 |
| 4 | Decision Tree | 0.8867 | 21 |
| 5 | Logistic Regression | 0.6129 | 1 |
| 6 | Naïve Bayesian | 0.6115 | 1 |

Based on the results above, our top 3 models are SVM, Random Forests, and Neural Networks. SVM produced the highest mean score among all models, but its computation time is more than 30 times higher than that of Random Forests. Taking processing time into consideration, Random Forests seems to be the best choice since its mean score is close to that of SVM.

# **Discussion**

## **7.1 Domain Knowledge**

The integration of predictive models for flight delay anticipation introduces transformative possibilities across various facets of the aviation industry. Within the operational resource allocation, predictive insights empower airlines to proactively manage those resources, gearing up for expected delays by strategically allocating staff and optimizing ground services. This pre-emptive approach ensures a more streamlined pre-flight process, encompassing check-in, security, and boarding, thereby contributing to heightened operational efficiency and passenger contentment. The model's predictive capabilities also facilitate advanced communication strategies, enabling ground and cabin crews to adeptly navigate potential delays and effectively manage passenger expectations.

In the realm of air traffic management, the application of flight delay prediction takes on a strategic role in mitigating the cascading impact of delays on the broader air traffic system. Real-time adjustments to flight routes and dynamic reassignment of takeoff and landing slots become feasible, allowing air traffic control to navigate congestion in affected airspace and minimize the likelihood of subsequent delays. Enhanced communication protocols between air traffic control, airlines, and airports foster collaborative decision-making, enabling a coordinated response to predicted delays. This proactive stance not only optimizes the utilization of available routes but also fortifies the air traffic system against the compounding effects of delays, fostering resilience and adaptability.

## **7.2 Methodological Contributions**

We had a unique original dataset with 178 features, but only 12 of them were continuous variables, while the other 166 were dummy variables. The reason for having so many dummy variables was that we had to convert categorical variables to dummies, such as Year, Month, Day of Month, Day of Week, Airlines, Origin Airports, Destination Airports, Departure Time, and Arrival Time. Moreover, in our original dataset, the number of samples with the value of Arrival Delay equaling to 1 is only 10,095, compared with 43,318 when the value of Arrival Delay equaling to 0. Thus, balancing the dataset is also required.

We were facing some challenges when we tried to reduce the number of features and balance our dataset. Should we balance our dataset first, then reduce the number of features? Or should we reduce the number of features, then balance our dataset? Should we select Naïve Random Over-Sampling or SMOTE to balance our dataset? Should we use feature selection or Primary Component Analysis to reduce the number of features?

In deciding which method works best for us, we decided to run Random Forests model on each dataset candidate and checked the results using K-Fold cross validation. We found that using Naïve Random Over-Sampling technique with Primary Component Analysis produced the highest mean scores and the order of which one to do first did not change the results.

# **Conclusions**

From this study, we have shown that balancing the data using Naïve Random Over-Sampling can improve the results significantly, especially recall values. SMOTE also improved our results but Naïve Random Over-Sampling produced much higher accuracy values. For reducing the number of features, our results show that Primary Component Analysis performs slightly better than feature selection by feature importance.

Using the dataset after applying Primary Component Analysis with Naïve Random Over-Sampling to train 6 machine learning algorithms in predicting flight delays, we found that SVM algorithms produced the highest accuracy, but its computation time was rather impractical. Random Forests, on the other hand, produced similar results with much less computation time. Thus, we concluded that Random Forests is the best model for predicting flight delays using our datasets.

Future work should involve the prediction of flight delays in other countries. In this study, only flight data in the United State was analyzed. In addition, our dataset is relatively small, compared with the full dataset that can be downloaded from the Department of Transportation. The full dataset contains millions of rows while our dataset has only around fifty thousand rows. Applying cloud computation to train machine learning algorithms using the full dataset can yield more accurate results.

# **References**

1. Dai, W., Zhang, M., & Low, K. H. (2022). Data-Efficient Modeling for Precise Power Consumption Estimation of Quadrotor Operations Using Ensemble Learning. *Journal of Latex Class Files.*
2. Anupkumar, A. (2023). Investigating the Costs and Economic Impact of Flight Delays in the Aviation Industry and the Potential Strategies for Reduction. *Electronic Theses, Projects, and Dissertations*, 1653.
3. Kim, Y. J., Choi, S., Briceno, S., & Mavris, D. (2016). A Deep Learning Approach to Flight Delay Prediction. In *35th Digital Avionics Systems Conference (DASC)*.
4. Chakrabarty, N. (2019). A Data Mining Approach to Flight Arrival Delay Prediction for American Airlines. In *9th Annual Information Technology, Electromechanical Engineering and Microelectronics Conference (IEMECON)*.
5. Gui, G., Liu, F., Sun, J., Yang, J., Zhou, Z., & Zhao, D. (2020). Flight Delay Prediction Based on Aviation Big Data and Machine Learning. *IEEE Transactions on Vehicular Technology*, 69(1), 140-150. doi: 10.1109/TVT.2019.2954094.
6. Hatıpoğlu, I., Tosun, Ö., & Tosun, N. (2022). Flight Delay Prediction Based on Machine Learning. *Logforum, 18*(1), 8. doi: 10.17270/J.LOG.2022.655.
7. Li, Q., & Jing, R. (November 2022). Flight Delay Prediction from Spatial and Temporal Perspective.
8. Cai, K., Li, Y., Fang, Y. P., & Zhu, Y. (August 2022). A Deep Learning Approach for Flight Delay Prediction Through Time-Evolving Graphs.
9. Airline Reporting Carrier On-Time Performance Dataset. (2020, November 23). IBM Developer. Retrieved October 02, 2023, from <https://developer.ibm.com/exchanges/data/all/airline/>

# **Appendices**

## **10.1 Data Dictionary**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| No. | Variable | Description | Domain Value | Data Type | Length | Example Value | Null Ratio |
| 1 | Arrival Delay | Arrival Delay Indicator, 15 Minutes or More (1=Yes) | 0 or 1 | Binary | 1 | 0 | 0% |
| 2 | Year | The year in which the flight took place. | 2018, 2019, 2020 | Nominal | 4 | 2019 | 0% |
| 3 | Month | The month in which the flight took place. | 1-12 | Nominal | 2 | 6 | 0% |
| 4 | Day of Month | The day of the month in which the flight took place. | 1-31 | Nominal | 2 | 5 | 0% |
| 5 | Day of Week | The day of the week in which the flight took place. | 1-7 | Nominal | 1 | 4 | 0% |
| 6 | Airlines | The name of the airline that was reported in the data. |  | Nominal |  | Alaska Airlines | 0% |
| 7 | Origin Airport | The unique identifier of the airport from which the flight originated. |  | Nominal |  | Sea-Tac | 0% |
| 8 | Destination Airport | The unique identifier of the airport at which the flight arrived. |  | Nominal |  | Sea-Tac | 0% |
| 9 | Departure Time Bulk | CRS Departure Time Block, Hourly Intervals. | 00:00-23:59 | Nominal |  | 13:00-13:59 | 0% |
| 10 | Arrival Time Bulk | CRS Arrival Time Block, Hourly Intervals. | 00:00-23:59 | Nominal |  | 13:00-13:59 | 0% |
| 11 | Windspeed | The rate of horizontal travel of air past a fixed point in meters per second. | 0000-0900 | Numeric | 4 | 0250 | 6.1% |
| 12 | Cloud Height | The height above ground level of the lowest cloud in meters. | 00000-22000 | Numeric | 5 | 22000 | 15.5% |
| 13 | Visibility | The horizontal distance at which an object can be seen and identified in meters. | 000000-160000 | Numeric | 6 | 160000 | 6.1% |
| 14 | Temperature | The temperature of the air in Celsius degrees. | -0932 to +0618 | Numeric | 5 | +0050 | 6.1% |
| 15 | Rain | The depth of liquid precipitation that is measured at the time of an observation in millimeters. | 0000-9998 | Numeric | 4 | 0111 | 22.4% |
| 16 | Snow | The depth of snow and ice on the ground in centimeters. | 0000-1200 | Numeric | 4 | 0010 | 76.5% |

## **10.2 Python Code**

### a. Data Pre-Processing - Merging Weather Data of Different Airports

import pandas as pd

import os

################################

#1. Loading the data

################################

#Data sets used in thie file can be downloaded using Seattle University email account from

#https://redhawks-my.sharepoint.com/:f:/g/personal/yso1\_seattleu\_edu/EinYTJ-7CdRKkTtXgMQjRgQBiY4eEkGgkFytrsZD7Xto3A?e=okv4Bl

#Change directory to where we will stote the weather data files.

new\_directory = "C:/Users/yada/Documents/3. School/Classses/11. Capstone/Project/Dataset/Data from NOAA"

os.chdir(new\_directory)

#We can only download weather data for one airport for one year at a time so there are more than

#200 weather files since we have more than 30 airports and we downloaded the data for 6 years.

#Read the data from the first weather file and select only relevant columns.

df = pd.read\_csv('Weather (1).csv')

df = df[['DATE','NAME','WND','CIG','VIS','TMP','DEW','SLP','AA1','AJ1']]

n\_rows = len(df)

#Read weather files one by one and concatenate them together.

for i in range(2,259):

filename = 'Weather '+'('+str(i)+').csv'

print (filename)

df\_m=pd.read\_csv(filename)

n\_rows = n\_rows + len(df\_m)

#Some files have no AJ1 column (snow information).

try:

df\_m = df\_m[['DATE','NAME','WND','CIG','VIS','TMP','DEW','SLP','AA1','AJ1']]

#When there is no snow information column "AJ1" in the weather file, then we create

#"AJ1" column and set it to 0.

except:

df\_m = df\_m[['DATE','NAME','WND','CIG','VIS','TMP','DEW','SLP','AA1']]

df\_m['AJ1'] = 0

df = pd.concat([df, df\_m], axis=0, ignore\_index=True)

#Create new column names

new\_column\_names = {

'WND': 'Wind',

'CIG': 'Cloud',

'VIS': 'Visibility',

'TMP': 'Air Temp',

'DEW': 'Air Temp2',

'SLP': 'Atmosphere',

'AA1': 'Rain',

'AJ1': 'Snow'

}

# Renaming columns using the rename() function

df.rename(columns=new\_column\_names, inplace=True)

# Specify the file name

file\_name = 'weather\_2018-2023\_full data set.csv'

# Writing to CSV file using pandas

df.to\_csv(file\_name, index=False)

### b. Data Pre-Processing - Weather Data

import pandas as pd

import os

################################

#1. Loading the data

################################

#Data sets used in thie file can be downloaded using Seattle University email account from

#https://redhawks-my.sharepoint.com/:f:/g/personal/yso1\_seattleu\_edu/EinYTJ-7CdRKkTtXgMQjRgQBiY4eEkGgkFytrsZD7Xto3A?e=okv4Bl

#Change directory to where we will stote the weather data files.

new\_directory = "C:/Users/yada/Documents/3. School/Classses/11. Capstone/Project/Dataset/Data from NOAA"

os.chdir(new\_directory)

df0 = pd.read\_csv('weather\_2018-2023\_full data set.csv')

df = df0.copy()

df.isnull().sum(axis=0)

#Replacing null values in Windspeed by 0.

df['Windspeed'] = df['Wind'].str[8:12]

df\_wind = df.groupby('Windspeed').size().reset\_index(name='Count')

df['Windspeed'] = df['Windspeed'].astype(int)

df.loc[df['Windspeed'] == 9999, 'Windspeed'] = 0

df\_wind2 = df.groupby('Windspeed').size().reset\_index(name='Count')

df['Windspeed'].isnull().sum(axis=0)

#Replacing null values in Cloudheight by 22000 because this is the mode of cloudheight.

df['Cloudheight'] = df['Cloud'].str[0:5]

df\_cloud = df.groupby('Cloudheight').size().reset\_index(name='Count')

df['Cloudheight'] = df['Cloudheight'].astype(int)

df.loc[df['Cloudheight'] == 99999, 'Cloudheight'] = 22000

df\_cloud2 = df.groupby('Cloudheight').size().reset\_index(name='Count')

df['Cloudheight'].isnull().sum(axis=0)

#Replacing null values in Visibility by 16093 because this is the mode of visibility.

df['Visible Distance'] = df['Visibility'].str[0:6]

df\_visible = df.groupby('Visible Distance').size().reset\_index(name='Count')

df['Visible Distance'] = df['Visible Distance'].astype(int)

df.loc[df['Visible Distance'] == 999999, 'Visible Distance'] = 16093

df\_visible2 = df.groupby('Visible Distance').size().reset\_index(name='Count')

df['Visible Distance'].isnull().sum(axis=0)

#Replacing null values in air temperature by 228 because this is the mode.

df['Temperate'] = df['Air Temp'].str[0:5]

df\_temp = df.groupby('Temperate').size().reset\_index(name='Count')

df['Temperate'] = df['Temperate'].astype(int)

df.loc[df['Temperate'] == 9999, 'Temperate'] = 228

df\_temp2 = df.groupby('Temperate').size().reset\_index(name='Count')

df['Temperate'].isnull().sum(axis=0)

#Replacing null values in rain by 0 because this is the mode.

df.isnull().sum(axis=0)

df2 = df[df['Rain'].isnull()]

df2['Rain2'] = None

df1 = df.dropna(subset=['Rain'])

df1.isnull().sum(axis=0)

df1['Rain2'] = df1['Rain'].str[3:7]

df1\_rain = df1.groupby('Rain2').size().reset\_index(name='Count')

df1['Rain2'] = df1['Rain2'].astype(int)

df1.loc[df1['Rain2'] == 9999, 'Rain2'] = 0

df1\_rain2= df1.groupby('Rain2').size().reset\_index(name='Count')

df1['Rain2'].isnull().sum(axis=0)

df = pd.concat([df1, df2], axis=0, ignore\_index=True)

df.loc[df['Rain2'].isnull(), 'Rain2'] = 0

df['Rain2'].isnull().sum(axis=0)

#Replacing null values in snow by 0 because this is the mode.

df.isnull().sum(axis=0)

df2 = df[df['Snow'].isnull()]

df2['Snow2'] = None

df1 = df.dropna(subset=['Snow'])

df1.isnull().sum(axis=0)

df1\_Snow = df1.groupby('Snow').size().reset\_index(name='Count')

df1.loc[df1['Snow'] == 0, 'Snow2'] = 0

df1.loc[df1['Snow'] != 0, 'Snow2'] = df1['Snow'].str[:4]

df1\_Snow = df1.groupby('Snow2').size().reset\_index(name='Count')

df1['Snow2'] = df1['Snow2'].astype(int)

df1.loc[df1['Snow2'] == 9999, 'Snow2'] = 0

df1\_Snow2= df1.groupby('Snow2').size().reset\_index(name='Count')

df1['Snow2'].isnull().sum(axis=0)

df = pd.concat([df1, df2], axis=0, ignore\_index=True)

df.loc[df['Snow2'].isnull(), 'Snow2'] = 0

df['Snow2'].isnull().sum(axis=0)

#Convert date to Year.

df['date2']=pd.to\_datetime (df['DATE'])

df['Year']=df['date2'].dt.year

######################################

#Select only data in 2018, 2019 and 2020 and drop data in 2021, 2022 and 2023

######################################

#Select only 2018, 2019, and 2020.

df1 = df[df['Year'].isin([2018,2019,2020])]

#Obtain date and hour from the original column.

df1['ymd']=df1['date2'].dt.date

df1['HourofDay']=df1['date2'].dt.hour

#Names of airports in the weather data and names of airports in the flight data are different.

#The following csv file tell us the difference in names.

df\_airport = pd.read\_csv('weather\_airport.csv')

#Merger the weather data with the airport name data.

df1 = pd.merge(df1, df\_airport, on='NAME', how='left')

#Create a variable called "AirportTime" containing airport name, data and hour for merging

#with the flight data later.

df1['AirportTime'] = df1['Ori\_Airport\_name'] + '\_' + df1['ymd'].astype(str) + '\_' + df1['HourofDay'].astype(str)

# Find rows where values in the AirportTime column are duplicates

df1['duplicate'] = df1.duplicated(subset='AirportTime', keep='first')

df1.loc[df1['duplicate'] == True, 'duplicate'] = None

###Count the number of nulls for each column

df1.isnull().sum(axis=0)

df1 = df1.dropna(subset=['duplicate'])

df1.isnull().sum(axis=0)

#Select only relevant column

df2= df1[['AirportTime','Windspeed','Cloudheight','Visible Distance','Temperate','Rain2','Snow2']]

df2.isnull().sum(axis=0)

# Specify the file name

file\_name = 'weather\_2018-2020\_v2.csv'

# Writing to CSV file using pandas

df2.to\_csv(file\_name, index=False)

### c. Data Pre-Processing - Flight Data

import pandas as pd

import numpy as np

import os

from datetime import datetime, timedelta

################################

#1. Loading the data

################################

#Data sets used in thie file can be downloaded using Seattle University email account from

#https://redhawks-my.sharepoint.com/:f:/g/personal/yso1\_seattleu\_edu/EinYTJ-7CdRKkTtXgMQjRgQBiY4eEkGgkFytrsZD7Xto3A?e=okv4Bl

new\_directory = "C:/Users/yada/Documents/3. School/Classses/11. Capstone/Project/Dataset/Data from DOT"

os.chdir(new\_directory)

#Loading the dataset

df = pd.read\_csv('flight data 2018\_2020.csv')

#Loading the dataset containing airport ID and airport full names

df\_ori\_airport = pd.read\_csv('L\_AIRPORT.csv')

#Loading the dataset containing airline ID and airline full names

df\_airline = pd.read\_csv('L\_UNIQUE\_CARRIERS.csv')

################################

#2. Merging the data

################################

#Make of a copy of the dataset

df\_Dest\_airport = df\_ori\_airport.copy()

#Create new column names

new\_column\_names = {

'Code': 'Reporting\_Airline',

'Description': 'Airline\_name'

}

# Renaming columns using the rename() function

df\_airline.rename(columns=new\_column\_names, inplace=True)

#Create new column names

new\_column\_names = {

'Code': 'Origin',

'Description': 'Ori\_Airport\_name'

}

# Renaming columns using the rename() function

df\_ori\_airport.rename(columns=new\_column\_names, inplace=True)

new\_column\_names = {

'Code': 'Dest',

'Description': 'Dest\_Airport\_name'

}

# Renaming columns using the rename() function

df\_Dest\_airport.rename(columns=new\_column\_names, inplace=True)

# Merging the two datasets so that we have airport full names and airline full names in df

df = pd.merge(df, df\_airline, on='Reporting\_Airline', how='left')

df = pd.merge(df, df\_ori\_airport, on='Origin', how='left')

df = pd.merge(df, df\_Dest\_airport, on='Dest', how='left')

# #Select only relevant columns

df2 = df[['Year','Month','DayofMonth', 'FlightDate','DayOfWeek', 'Airline\_name', 'Ori\_Airport\_name', 'Dest\_Airport\_name', 'DepTimeBlk', 'ArrTimeBlk', 'DepDel15','ArrDel15','ArrDelayMinutes','Diverted']]

###Count the number of nulls for each column

df2.isnull().sum(axis=0)

#If a flight is Diverted, then change the value of ArrDel15 to 1.

df2.loc[(df2['ArrDel15'].isnull()) & (df2['Diverted'] == 1), 'ArrDel15'] = 1

#Drop all rows with null values

df2.dropna(axis=0, how='any',inplace=True)

df2.isnull().sum(axis=0)

################################

#3. Exploring the data before selecting airlines

################################

#Create a copy of the data set

df3 = df2.copy()

#Restrict data to only flights with delayed departure

df4 = df3[df3['DepDel15'] == 1]

#Restrict data to only flights with delayed arrival

df5 = df3[df3['ArrDel15'] == 1]

df3['DepDel15'].value\_counts()

df3['ArrDel15'].value\_counts()

#3.1 Exploring variables

#Calculate number of flights and delayed flights in each Year

df3\_Year = df3.groupby('Year').size().reset\_index(name='Count')

df5\_Year = df5.groupby('Year').size().reset\_index(name='Delay Count')

df3\_Year = pd.merge(df3\_Year, df5\_Year, on='Year', how='left')

df3\_Year['Year Delay Ratio'] = round(df3\_Year['Delay Count'] / df3\_Year['Count'],3)

#Calculate number of flights and delayed flights in each Month

df3\_Month = df3.groupby('Month').size().reset\_index(name='Count')

df5\_Month = df5.groupby('Month').size().reset\_index(name='Delay Count')

df3\_Month = pd.merge(df3\_Month, df5\_Month, on='Month', how='left')

df3\_Month['Month Delay Ratio'] = round(df3\_Month['Delay Count'] / df3\_Month['Count'],3)

#Calculate number of flights and delayed flights in each day of Month

df3\_DayofMonth = df3.groupby('DayofMonth').size().reset\_index(name='Count')

df5\_DayofMonth = df5.groupby('DayofMonth').size().reset\_index(name='Delay Count')

df3\_DayofMonth = pd.merge(df3\_DayofMonth, df5\_DayofMonth, on='DayofMonth', how='left')

df3\_DayofMonth['DayofMonth Delay Ratio'] = round(df3\_DayofMonth['Delay Count'] / df3\_DayofMonth['Count'],3)

#Calculate number of flights and delayed flights in each day of week

df3\_dofweek = df3.groupby('DayOfWeek').size().reset\_index(name='Count')

df5\_dofweek = df5.groupby('DayOfWeek').size().reset\_index(name='Delay Count')

df3\_dofweek = pd.merge(df3\_dofweek, df5\_dofweek, on='DayOfWeek', how='left')

df3\_dofweek['dofweek Delay Ratio'] = round(df3\_dofweek['Delay Count'] / df3\_dofweek['Count'],3)

#Calculate number of flights and delayed flights of each airline

df3\_airline = df3.groupby('Airline\_name').size().reset\_index(name='Count')

df5\_airline = df5.groupby('Airline\_name').size().reset\_index(name='Delay Count')

df3\_airline = pd.merge(df3\_airline, df5\_airline, on='Airline\_name', how='left')

df3\_airline['airline Delay Ratio'] = round(df3\_airline['Delay Count'] / df3\_airline['Count'],3)

df3\_airline = df3\_airline.sort\_values(by='Count', ascending=False)

df3\_airline.reset\_index(drop=True, inplace=True)

# Calculate the sum of each column and create a new row

sum\_row = pd.Series(df3\_airline.sum(), name='Total')

# Append the new row to the DataFrame

df3\_airline = df3\_airline.append(sum\_row)

#Calculate number of flights and delayed flights of each Origin airport. Here, we use df4 instead of df5

#because we are counting number of delayed departure, not delayed arrival.

df3\_Ori\_Airport\_name = df3.groupby('Ori\_Airport\_name').size().reset\_index(name='Count')

df4\_Ori\_Airport\_name = df4.groupby('Ori\_Airport\_name').size().reset\_index(name='Delay Count')

df3\_Ori\_Airport\_name = pd.merge(df3\_Ori\_Airport\_name, df4\_Ori\_Airport\_name, on='Ori\_Airport\_name', how='left')

df3\_Ori\_Airport\_name['Ori\_Airport\_name Delay Ratio'] = round(df3\_Ori\_Airport\_name['Delay Count'] / df3\_Ori\_Airport\_name['Count'],3)

#Calculate number of flights and delayed flights of each Destination airport

df3\_Dest\_Airport\_name = df3.groupby('Dest\_Airport\_name').size().reset\_index(name='Count')

df5\_Dest\_Airport\_name = df5.groupby('Dest\_Airport\_name').size().reset\_index(name='Delay Count')

df3\_Dest\_Airport\_name = pd.merge(df3\_Dest\_Airport\_name, df5\_Dest\_Airport\_name, on='Dest\_Airport\_name', how='left')

df3\_Dest\_Airport\_name['Dest\_Airport\_name Delay Ratio'] = round(df3\_Dest\_Airport\_name['Delay Count'] / df3\_Dest\_Airport\_name['Count'],3)

#Calculate number of flights and delayed flights of each time interval. Here, we use df4 instead of df5

#because we are counting number of delayed departure, not delayed arrival.

df3\_DepTime = df3.groupby('DepTimeBlk').size().reset\_index(name='Count')

df4\_DepTime = df4.groupby('DepTimeBlk').size().reset\_index(name='Delay Count')

df3\_DepTime = pd.merge(df3\_DepTime, df4\_DepTime, on='DepTimeBlk', how='left')

df3\_DepTime['DepTime Delay Ratio'] = round(df3\_DepTime['Delay Count'] / df3\_DepTime['Count'],3)

#Calculate number of flights and delayed flights of each time interval

df3\_ArrTime = df3.groupby('ArrTimeBlk').size().reset\_index(name='Count')

df5\_ArrTime = df5.groupby('ArrTimeBlk').size().reset\_index(name='Delay Count')

df3\_ArrTime = pd.merge(df3\_ArrTime, df5\_ArrTime, on='ArrTimeBlk', how='left')

df3\_ArrTime['ArrTime Delay Ratio'] = round(df3\_ArrTime['Delay Count'] / df3\_ArrTime['Count'],3)

################################

#4. Selecting 3 Big Airlines and 3 Local Airlines

################################

################################

#4.1 Big Airlines

################################

#Select only the 3 big airlines

airline\_big = df3\_airline.head(3)

#Restrict the data to only the 3 big airlines, and count the number of flight by each airports

df6 = df3[df3['Airline\_name'].isin(airline\_big['Airline\_name'])]

df6\_ori\_airport = df6.groupby('Ori\_Airport\_name').size().reset\_index(name='Ori\_Count')

df6\_Dest\_airport = df6.groupby('Dest\_Airport\_name').size().reset\_index(name='Dest\_Count')

#Create new column names to merge the data

new\_column\_names = {

'Dest\_Airport\_name': 'Ori\_Airport\_name'

}

# Renaming columns using the rename() function

df6\_Dest\_airport.rename(columns=new\_column\_names, inplace=True)

#Merging two datasets and count the total number of flights of Origin airports and Destination airports

df6\_airport = pd.merge(df6\_ori\_airport, df6\_Dest\_airport, on='Ori\_Airport\_name', how='outer')

df6\_airport['Count'] = df6\_airport['Ori\_Count'] + df6\_airport['Dest\_Count']

df6\_airport = df6\_airport.sort\_values(by='Count', ascending=False)

df6\_airport.reset\_index(drop=True, inplace=True)

#Select only the biggest 25 airports that the 3 big airlines are using

airport\_big = df6\_airport.head(25)

################################

#4.2 Local Airlines

################################

#Select only the 3 local airlines

airline\_local = df3\_airline.iloc[[5, 7, 9]]

#Restrict the data to only the 3 local airlines, and count the number of flight by each airports

df7 = df3[df3['Airline\_name'].isin(airline\_local['Airline\_name'])]

df7\_ori\_airport = df7.groupby('Ori\_Airport\_name').size().reset\_index(name='Ori\_Count')

df7\_Dest\_airport = df7.groupby('Dest\_Airport\_name').size().reset\_index(name='Dest\_Count')

#Create new column names to merge the data

new\_column\_names = {

'Dest\_Airport\_name': 'Ori\_Airport\_name'

}

# Renaming columns using the rename() function

df7\_Dest\_airport.rename(columns=new\_column\_names, inplace=True)

#Merging two datasets and count the total number of flights of Origin airports and Destination airports

df7\_airport = pd.merge(df7\_ori\_airport, df7\_Dest\_airport, on='Ori\_Airport\_name', how='outer')

df7\_airport['Count'] = df7\_airport['Ori\_Count'] + df7\_airport['Dest\_Count']

df7\_airport = df7\_airport.sort\_values(by='Count', ascending=False)

df7\_airport.reset\_index(drop=True, inplace=True)

#Select only the biggest 25 airports that the 3 local airlines are using

airport\_local = df7\_airport.head(25)

################################

#4.3 Big & Local Airlines

################################

#Dataframe of 3 big airlines and 3 local airlines

airline\_big\_local = pd.merge(airline\_big, airline\_local, on=['Airline\_name'],how='outer')

#Dataframe of the 25 biggest airports that the 3 big airlines are using

#and the 25 biggest airports that the 3 local airlines are using .

#Some airports might overlap.

airport\_big\_local = pd.merge(airport\_big, airport\_local, on=['Ori\_Airport\_name'],how='outer')

#Restrict the data to only airlines and airports in the 2 data frames above.

df8 = df3[df3['Airline\_name'].isin(airline\_big\_local['Airline\_name'])]

df8 = df8[df8['Ori\_Airport\_name'].isin(airport\_big\_local['Ori\_Airport\_name'])]

df8 = df8[df8['Dest\_Airport\_name'].isin(airport\_big\_local['Ori\_Airport\_name'])]

# Specify the file name

file\_name = 'flight\_6al\_2018-2020.csv'

# Writing to CSV file using pandas

df8.to\_csv(file\_name, index=False)

################################

#5. Exploring the data after selecting airlines

################################

#Restrict data to only flights with delayed departure

df9 = df8[df8['DepDel15'] == 1]

#Restrict data to only flights with delayed arrival

df10 = df8[df8['ArrDel15'] == 1]

df8['DepDel15'].value\_counts()

df8['ArrDel15'].value\_counts()

#5.1 Exploring variables

#Calculate number of flights and delayed flights in each Year

df8\_Year = df8.groupby('Year').size().reset\_index(name='Count')

df10\_Year = df10.groupby('Year').size().reset\_index(name='Delay Count')

df8\_Year = pd.merge(df8\_Year, df10\_Year, on='Year', how='left')

df8\_Year['Year Delay Ratio'] = round(df8\_Year['Delay Count'] / df8\_Year['Count'],3)

#Calculate number of flights and delayed flights in each Month

df8\_Month = df8.groupby('Month').size().reset\_index(name='Count')

df10\_Month = df10.groupby('Month').size().reset\_index(name='Delay Count')

df8\_Month = pd.merge(df8\_Month, df10\_Month, on='Month', how='left')

df8\_Month['Month Delay Ratio'] = round(df8\_Month['Delay Count'] / df8\_Month['Count'],3)

df8\_Month = df8\_Month.sort\_values(by='Month Delay Ratio', ascending=False)

df8\_Month.reset\_index(drop=True, inplace=True)

#Calculate number of flights and delayed flights in each day of Month

df8\_DayofMonth = df8.groupby('DayofMonth').size().reset\_index(name='Count')

df10\_DayofMonth = df10.groupby('DayofMonth').size().reset\_index(name='Delay Count')

df8\_DayofMonth = pd.merge(df8\_DayofMonth, df10\_DayofMonth, on='DayofMonth', how='left')

df8\_DayofMonth['DayofMonth Delay Ratio'] = round(df8\_DayofMonth['Delay Count'] / df8\_DayofMonth['Count'],3)

#Calculate number of flights and delayed flights in each day of week

df8\_dofweek = df8.groupby('DayOfWeek').size().reset\_index(name='Count')

df10\_dofweek = df10.groupby('DayOfWeek').size().reset\_index(name='Delay Count')

df8\_dofweek = pd.merge(df8\_dofweek, df10\_dofweek, on='DayOfWeek', how='left')

df8\_dofweek['dofweek Delay Ratio'] = round(df8\_dofweek['Delay Count'] / df8\_dofweek['Count'],3)

df8\_dofweek = df8\_dofweek.sort\_values(by='dofweek Delay Ratio', ascending=False)

df8\_dofweek.reset\_index(drop=True, inplace=True)

# Specify the file name

file\_name = 'dataex\_dayofweek.csv'

# Writing to CSV file using pandas

df8\_dofweek.to\_csv(file\_name, index=False)

#Calculate number of flights and delayed flights of each airline

df8\_airline = df8.groupby('Airline\_name').size().reset\_index(name='Count')

df10\_airline = df10.groupby('Airline\_name').size().reset\_index(name='Delay Count')

df8\_airline = pd.merge(df8\_airline, df10\_airline, on='Airline\_name', how='left')

df8\_airline['airline Delay Ratio'] = round(df8\_airline['Delay Count'] / df8\_airline['Count'],3)

df8\_airline = df8\_airline.sort\_values(by='airline Delay Ratio', ascending=False)

df8\_airline.reset\_index(drop=True, inplace=True)

# Specify the file name

file\_name = 'dataex\_airline.csv'

# Writing to CSV file using pandas

df8\_airline.to\_csv(file\_name, index=False)

#Calculate number of flights and delayed flights of each Origin airport. Here, we use df9 instead of df10

#because we are counting number of delayed departure, not delayed arrival.

df8\_Ori\_Airport\_name = df8.groupby('Ori\_Airport\_name').size().reset\_index(name='Count')

df9\_Ori\_Airport\_name = df9.groupby('Ori\_Airport\_name').size().reset\_index(name='Delay Count')

df8\_Ori\_Airport\_name = pd.merge(df8\_Ori\_Airport\_name, df9\_Ori\_Airport\_name, on='Ori\_Airport\_name', how='left')

df8\_Ori\_Airport\_name['Ori\_Airport\_name Delay Ratio'] = round(df8\_Ori\_Airport\_name['Delay Count'] / df8\_Ori\_Airport\_name['Count'],3)

#Calculate number of flights and delayed flights of each Destination airport

df8\_Dest\_Airport\_name = df8.groupby('Dest\_Airport\_name').size().reset\_index(name='Count')

df10\_Dest\_Airport\_name = df10.groupby('Dest\_Airport\_name').size().reset\_index(name='Delay Count')

df8\_Dest\_Airport\_name = pd.merge(df8\_Dest\_Airport\_name, df10\_Dest\_Airport\_name, on='Dest\_Airport\_name', how='left')

df8\_Dest\_Airport\_name['Dest\_Airport\_name Delay Ratio'] = round(df8\_Dest\_Airport\_name['Delay Count'] / df8\_Dest\_Airport\_name['Count'],3)

df8\_Dest\_Airport\_name = df8\_Dest\_Airport\_name.sort\_values(by='Dest\_Airport\_name Delay Ratio', ascending=False)

df8\_Dest\_Airport\_name.reset\_index(drop=True, inplace=True)

#Calculate number of flights and delayed flights of each time interval. Here, we use df9 instead of df10

#because we are counting number of delayed departure, not delayed arrival.

df8\_DepTime = df8.groupby('DepTimeBlk').size().reset\_index(name='Count')

df9\_DepTime = df9.groupby('DepTimeBlk').size().reset\_index(name='Delay Count')

df8\_DepTime = pd.merge(df8\_DepTime, df9\_DepTime, on='DepTimeBlk', how='left')

df8\_DepTime['DepTime Delay Ratio'] = round(df8\_DepTime['Delay Count'] / df8\_DepTime['Count'],3)

#Calculate number of flights and delayed flights of each time interval

df8\_ArrTime = df8.groupby('ArrTimeBlk').size().reset\_index(name='Count')

df10\_ArrTime = df10.groupby('ArrTimeBlk').size().reset\_index(name='Delay Count')

df8\_ArrTime = pd.merge(df8\_ArrTime, df10\_ArrTime, on='ArrTimeBlk', how='left')

df8\_ArrTime['ArrTime Delay Ratio'] = round(df8\_ArrTime['Delay Count'] / df8\_ArrTime['Count'],3)

df8\_ArrTime = df8\_ArrTime.sort\_values(by='ArrTime Delay Ratio', ascending=False)

df8\_ArrTime.reset\_index(drop=True, inplace=True)

### d. Data Pre-Processing - Merging Weather and Flight Data

import pandas as pd

import numpy as np

import os

from datetime import datetime, timedelta

################################

#6. Merging with the weather dataset

################################

new\_directory = "C:/Users/yada/Documents/3. School/Classses/11. Capstone/Project/Dataset/Data from DOT"

os.chdir(new\_directory)

#Loading the flight dataset

df8 = pd.read\_csv('flight\_6al\_2018-2020.csv')

#Create a new variable "AirportTime" so that we can use it to merge with the weather data.

df8['FlightDate2'] = pd.to\_datetime(df8['FlightDate'])

df8['DepHour'] = df8['DepTimeBlk'].str[:2]

df8['DepHour'] = df8['DepHour'].astype(int)

df8['Depymd']=df8['FlightDate2'].dt.date

df8['AirportTime'] = df8['Ori\_Airport\_name'] + '\_' + df8['Depymd'].astype(str) + '\_' + df8['DepHour'].astype(str)

new\_directory = "C:/Users/yada/Documents/3. School/Classses/11. Capstone/Project/Dataset/Data from NOAA"

os.chdir(new\_directory)

#Loading the weather dataset.

df\_weather = pd.read\_csv('weather\_2018-2020\_v2.csv')

#Create new column names

new\_column\_names = {

'Windspeed': 'Ori\_Windspeed',

'Cloudheight': 'Ori\_Cloudheight',

'Visible Distance': 'Ori\_Visible Distance',

'Temperate': 'Ori\_Temperature',

'Rain2': 'Ori\_Rain',

'Snow2': 'Ori\_Snow'

}

# Renaming columns using the rename() function

df\_weather.rename(columns=new\_column\_names, inplace=True)

#Merging the two datasets to get weather information at the origin airports.

df11 = pd.merge(df8, df\_weather, on='AirportTime', how='left')

df11.isnull().sum(axis=0)

#Getting weather data for destination airports

#Create a new variable "AirportTime" so that we can use it to merge with the weather data.

df11['ArrHour'] = df11['ArrTimeBlk'].str[:2]

df11['ArrHour'] = df11['ArrHour'].astype(int)

df11['Arrymd']=df11['FlightDate2'].dt.date

#If arrival hour is ealier than departure hour then arrival data must be the next day.

df11.loc[df11['ArrHour'] >= df11['DepHour'], 'Arrymd'] = df11['Arrymd']

df11.loc[df11['ArrHour'] < df11['DepHour'], 'Arrymd'] = df11['Arrymd'] + timedelta(days=1)

df11['AirportTime'] = df11['Dest\_Airport\_name'] + '\_' + df11['Arrymd'].astype(str) + '\_' + df11['ArrHour'].astype(str)

#Loading the weather dataset again.

df\_weather = pd.read\_csv('weather\_2018-2020\_v2.csv')

#Create new column names

new\_column\_names = {

'Windspeed': 'Dest\_Windspeed',

'Cloudheight': 'Dest\_Cloudheight',

'Visible Distance': 'Dest\_Visible Distance',

'Temperate': 'Dest\_Temperature',

'Rain2': 'Dest\_Rain',

'Snow2': 'Dest\_Snow'

}

# Renaming columns using the rename() function

df\_weather.rename(columns=new\_column\_names, inplace=True)

#Merging the two datasets to get weather information at the destination airports.

df11 = pd.merge(df11, df\_weather, on='AirportTime', how='left')

#Select only relevant column

df11= df11[['Year','Month','DayofMonth', 'DayOfWeek', 'Airline\_name', 'Ori\_Airport\_name', 'Dest\_Airport\_name', 'DepTimeBlk',

'ArrTimeBlk', 'ArrDelayMinutes','ArrDel15','Ori\_Windspeed','Ori\_Cloudheight','Ori\_Visible Distance','Ori\_Temperature',

'Ori\_Rain','Ori\_Snow','Dest\_Windspeed','Dest\_Cloudheight','Dest\_Visible Distance','Dest\_Temperature','Dest\_Rain','Dest\_Snow']]

df11.isnull().sum(axis=0)

df12 = df11[df11.isnull().any(axis=1)]

#Drop all rows with null values

df11.dropna(axis=0, how='any',inplace=True)

df11.isnull().sum(axis=0)

new\_directory = "C:/Users/yada/Documents/3. School/Classses/11. Capstone/Project/Dataset/Data\_Model"

os.chdir(new\_directory)

# Specify the file name

file\_name = 'flight\_weather\_2018-2020.csv'

# Writing to CSV file using pandas

df11.to\_csv(file\_name, index=False)

### e. Model Data - Original Data, then Data Imbalance

##########################################################

#Data Imbalance

##########################################################

from imblearn.over\_sampling import RandomOverSampler

from imblearn.over\_sampling import SMOTE

import numpy as np

import pandas as pd

import os

new\_directory = "C:/Users/yada/Documents/3. School/Classses/11. Capstone/Project/Dataset/Data\_Model"

os.chdir(new\_directory)

data = pd.read\_csv('flight\_weather\_2018-2020.csv')

data = data.drop('ArrDelayMinutes', axis=1)

# Convert nominal categorical columns to binary (dummy variables)

categorical\_columns = ['Year', 'Month', 'DayofMonth', 'DayOfWeek', 'Airline\_name',

'Ori\_Airport\_name', 'Dest\_Airport\_name', 'DepTimeBlk', 'ArrTimeBlk']

data\_binary = pd.get\_dummies(

data, columns=categorical\_columns, drop\_first=True)

# Define target variable for classification

y = data\_binary['ArrDel15'] # Predicting departure delay

# Select features for classification

X = data\_binary.drop(['ArrDel15'], axis=1)

X.shape

y.shape

print(f'Original data: {np.unique(y, return\_counts=1)}')

df = pd.concat([y, X], axis=1, ignore\_index=False)

# Specify the file name

file\_name = 'flight\_weather\_original.csv'

# Writing to CSV file using pandas

df.to\_csv(file\_name, index=False)

# Create an instance of RandomOverSampler

ros = RandomOverSampler(random\_state=0)

X\_rs, y\_rs = ros.fit\_resample(X, y)

X\_rs.shape

y\_rs.shape

print(f'Over-sampled data: {np.unique(y\_rs, return\_counts=1)}')

df\_rs = pd.concat([y\_rs, X\_rs], axis=1, ignore\_index=False)

# Specify the file name

file\_name = 'flight\_weather\_random.csv'

# Writing to CSV file using pandas

df\_rs.to\_csv(file\_name, index=False)

# Create an instance of SMOTE

sm = SMOTE(random\_state=0)

X\_sm, y\_sm = sm.fit\_resample(X, y)

X\_sm.shape

y\_sm.shape

print(f'Over-sampled data: {np.unique(y\_sm, return\_counts=1)}')

df\_sm = pd.concat([y\_sm, X\_sm], axis=1, ignore\_index=False)

# Specify the file name

file\_name = 'flight\_weather\_smote.csv'

# Writing to CSV file using pandas

df\_sm.to\_csv(file\_name, index=False)

### f. Model Data - Data Imbalance, then Primary Component Analysis

# Import the relevant libaries

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification\_report

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

import matplotlib.pyplot as plt

import os

import pandas as pd

new\_directory = "C:/Users/yada/Documents/3. School/Classses/11. Capstone/Project/Dataset/Data\_Model"

os.chdir(new\_directory)

data = pd.read\_csv('flight\_weather\_original.csv')

data\_rs = pd.read\_csv('flight\_weather\_random.csv')

data\_sm = pd.read\_csv('flight\_weather\_smote.csv')

# Define target variable for classification

y = data['ArrDel15'] # Predicting departure delay

# Select features for classification

X = data.drop(['ArrDel15'], axis=1)

#RandomOverSampler

# Define target variable for classification

y\_rs = data\_rs['ArrDel15'] # Predicting departure delay

# Select features for classification

X\_rs = data\_rs.drop(['ArrDel15'], axis=1)

#SMOTE

# Define target variable for classification

y\_sm = data\_sm['ArrDel15'] # Predicting departure delay

# Select features for classification

X\_sm = data\_sm.drop(['ArrDel15'], axis=1)

##################################################

#PCA Using Original Data

##################################################

# z\_score normalize the data

scaler = StandardScaler()

Xn = scaler.fit\_transform(X)

# Create an instance PCA and build the model using Xn.

# We start from the same number of components as the number of original

# features.

pca\_prep = PCA().fit(Xn)

pca\_prep.n\_components\_

# We want to find out how many components

# we want to use without losing much information.

pca\_prep.explained\_variance\_

pca\_prep.explained\_variance\_ratio\_

# Well, those numbers are difficult to understand. A scree plot would be

# effective. Find an "elbow" or an inflection point on the plot.

plt.plot(pca\_prep.explained\_variance\_ratio\_)

plt.xlabel('k number of components')

plt.ylabel('Explained variance')

plt.grid(True)

plt.show()

# Alternative plot using cumulative ratios

plt.plot(np.cumsum(pca\_prep.explained\_variance\_ratio\_))

plt.xlabel('k number of components')

plt.ylabel('cumulative explained variance')

plt.grid(True)

plt.show()

#50 seems to be the elbow point. So, we select 50 components.

n\_pc = 50

pca = PCA(n\_components= n\_pc).fit(Xn)

Xp = pca.transform(Xn)

print(f'After PCA, we use {pca.n\_components\_} components.\n')

Xp = pd.DataFrame(Xp)

df = pd.concat([y, Xp], axis=1, ignore\_index=False)

# Specify the file name

file\_name = 'flight\_weather\_original\_pca.csv'

# Writing to CSV file using pandas

df.to\_csv(file\_name, index=False)

# Split the data into training and testing subsets.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size =.2,

random\_state=1234,stratify=y)

Xp\_train, Xp\_test, yp\_train, yp\_test = train\_test\_split(Xp,y,test\_size =.2,

random\_state=1234,stratify=y)

# Create two random forest models: one using the original and the other using

# the transformed data.

rfcm = RandomForestClassifier().fit(X\_train, y\_train)

rfcm\_p = RandomForestClassifier().fit(Xp\_train, yp\_train)

# Predict the flight delay using the train data.

y\_train\_pred = rfcm.predict(X\_train)

y\_train\_pred\_p = rfcm\_p.predict(Xp\_train)

# Predict the flight delay using the test data.

y\_pred = rfcm.predict(X\_test)

y\_pred\_p = rfcm\_p.predict(Xp\_test)

# Compare the performance of each model.

report\_original\_train = classification\_report(y\_train, y\_train\_pred)

report\_pca\_train = classification\_report(yp\_train, y\_train\_pred\_p)

report\_original = classification\_report(y\_test, y\_pred)

report\_pca = classification\_report(yp\_test, y\_pred\_p)

print(f'Classification Report - original - train\n{report\_original\_train}')

print(f'Classification Report - original - test\n{report\_original}')

print(f'Classification Report - pca - train\n{report\_pca\_train}')

print(f'Classification Report - pca - test\n{report\_pca}')

##################################################

#PCA Using Naive Random Over-Sampling Data

##################################################

# z\_score normalize the data

scaler = StandardScaler()

Xn = scaler.fit\_transform(X\_rs)

# Create an instance PCA and build the model using Xn.

# We start from the same number of components as the number of original

# features.

pca\_prep = PCA().fit(Xn)

pca\_prep.n\_components\_

# We want to find out how many components

# we want to use without losing much information.

pca\_prep.explained\_variance\_

pca\_prep.explained\_variance\_ratio\_

# Well, those numbers are difficult to understand. A scree plot would be

# effective. Find an "elbow" or an inflection point on the plot.

plt.plot(pca\_prep.explained\_variance\_ratio\_)

plt.xlabel('k number of components')

plt.ylabel('Explained variance')

plt.grid(True)

plt.show()

# Alternative plot using cumulative ratios

plt.plot(np.cumsum(pca\_prep.explained\_variance\_ratio\_))

plt.xlabel('k number of components')

plt.ylabel('cumulative explained variance')

plt.grid(True)

plt.show()

#50 seems to be the elbow point. So, we select 50 components.

n\_pc = 50

pca = PCA(n\_components= n\_pc).fit(Xn)

Xp = pca.transform(Xn)

print(f'After PCA, we use {pca.n\_components\_} components.\n')

Xp = pd.DataFrame(Xp)

df = pd.concat([y\_rs, Xp], axis=1, ignore\_index=False)

# Specify the file name

file\_name = 'flight\_weather\_random\_pca.csv'

# Writing to CSV file using pandas

df.to\_csv(file\_name, index=False)

# Split the data into training and testing subsets.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_rs,y\_rs,test\_size =.2,

random\_state=1234,stratify=y\_rs)

Xp\_train, Xp\_test, yp\_train, yp\_test = train\_test\_split(Xp,y\_rs,test\_size =.2,

random\_state=1234,stratify=y\_rs)

# Create two random forest models: one using the original and the other using

# the transformed data. Of course, you can use other algorithms.

rfcm = RandomForestClassifier().fit(X\_train, y\_train)

rfcm\_p = RandomForestClassifier().fit(Xp\_train, yp\_train)

# Predict the delay using the train data.

y\_train\_pred = rfcm.predict(X\_train)

y\_train\_pred\_p = rfcm\_p.predict(Xp\_train)

# Predict the delay using the test data.

y\_pred = rfcm.predict(X\_test)

y\_pred\_p = rfcm\_p.predict(Xp\_test)

# Compare the performance of each model.

report\_original\_train = classification\_report(y\_train, y\_train\_pred)

report\_pca\_train = classification\_report(yp\_train, y\_train\_pred\_p)

report\_original = classification\_report(y\_test, y\_pred)

report\_pca = classification\_report(yp\_test, y\_pred\_p)

print(f'Classification Report - original - train\n{report\_original\_train}')

print(f'Classification Report - original - test\n{report\_original}')

print(f'Classification Report - pca - train\n{report\_pca\_train}')

print(f'Classification Report - pca - test\n{report\_pca}')

##################################################

#PCA Using SMOTE Data

##################################################

# z\_score normalize the data

scaler = StandardScaler()

Xn = scaler.fit\_transform(X\_sm)

# Create an instance PCA and build the model using Xn.

# We start from the same number of components as the number of original

# features.

pca\_prep = PCA().fit(Xn)

pca\_prep.n\_components\_

# We want to find out how many components

# we want to use without losing much information.

pca\_prep.explained\_variance\_

pca\_prep.explained\_variance\_ratio\_

# Well, those numbers are difficult to understand. A scree plot would be

# effective. Find an "elbow" or an inflection point on the plot.

plt.plot(pca\_prep.explained\_variance\_ratio\_)

plt.xlabel('k number of components')

plt.ylabel('Explained variance')

plt.grid(True)

plt.show()

# Alternative plot using cumulative ratios

plt.plot(np.cumsum(pca\_prep.explained\_variance\_ratio\_))

plt.xlabel('k number of components')

plt.ylabel('cumulative explained variance')

plt.grid(True)

plt.show()

#50 seems to be the elbow point. So, we select 50 components.

n\_pc = 50

pca = PCA(n\_components= n\_pc).fit(Xn)

Xp = pca.transform(Xn)

print(f'After PCA, we use {pca.n\_components\_} components.\n')

Xp = pd.DataFrame(Xp)

df = pd.concat([y\_sm, Xp], axis=1, ignore\_index=False)

# Specify the file name

file\_name = 'flight\_weather\_smote\_pca.csv'

# Writing to CSV file using pandas

df.to\_csv(file\_name, index=False)

# Split the data into training and testing subsets.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_sm,y\_sm,test\_size =.2,

random\_state=1234,stratify=y\_sm)

Xp\_train, Xp\_test, yp\_train, yp\_test = train\_test\_split(Xp,y\_sm,test\_size =.2,

random\_state=1234,stratify=y\_sm)

# Create two random forest models: one using the original and the other using

# the transformed data. Of course, you can use other algorithms.

rfcm = RandomForestClassifier().fit(X\_train, y\_train)

rfcm\_p = RandomForestClassifier().fit(Xp\_train, yp\_train)

# Predict the delay using the train data.

y\_train\_pred = rfcm.predict(X\_train)

y\_train\_pred\_p = rfcm\_p.predict(Xp\_train)

# Predict the delay using the test data.

y\_pred = rfcm.predict(X\_test)

y\_pred\_p = rfcm\_p.predict(Xp\_test)

# Compare the performance of each model.

report\_original\_train = classification\_report(y\_train, y\_train\_pred)

report\_pca\_train = classification\_report(yp\_train, y\_train\_pred\_p)

report\_original = classification\_report(y\_test, y\_pred)

report\_pca = classification\_report(yp\_test, y\_pred\_p)

print(f'Classification Report - original - train\n{report\_original\_train}')

print(f'Classification Report - original - test\n{report\_original}')

print(f'Classification Report - pca - train\n{report\_pca\_train}')

print(f'Classification Report - pca - test\n{report\_pca}')

### g. Model Data - Data Imbalance, then Feature Selection

# =============================================

# Feature Selection using Feature-Importances

# =============================================

# Import relevant libraries.

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.feature\_selection import SelectFromModel

from sklearn import metrics

import matplotlib.pyplot as plt

import os

new\_directory = "C:/Users/yada/Documents/3. School/Classses/11. Capstone/Project/Dataset/Data\_Model"

os.chdir(new\_directory)

data = pd.read\_csv('flight\_weather\_original.csv')

data\_rs = pd.read\_csv('flight\_weather\_random.csv')

data\_sm = pd.read\_csv('flight\_weather\_smote.csv')

# Define target variable for classification

y = data['ArrDel15'] # Predicting departure delay

# Select features for classification

X = data.drop(['ArrDel15'], axis=1)

#RandomOverSampler

# Define target variable for classification

y\_rs = data\_rs['ArrDel15'] # Predicting departure delay

# Select features for classification

X\_rs = data\_rs.drop(['ArrDel15'], axis=1)

#SMOTE

# Define target variable for classification

y\_sm = data\_sm['ArrDel15'] # Predicting departure delay

# Select features for classification

X\_sm = data\_sm.drop(['ArrDel15'], axis=1)

##################################################

#FS Using Original Data

##################################################

fn = X.columns

print(f'Originally, we have {len(fn)} features.')

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size =.3, stratify=y)

# Create an instance (object) for classification and build a model.

rfcm = RandomForestClassifier().fit(X\_train, y\_train)

# Make predictions using the test data

y\_pred = rfcm.predict(X\_test)

# Show the Classification Report.

print('\nClassification Report\n')

print(metrics.classification\_report(y\_test,y\_pred))

# %matplotlib auto

# %matplotlib inline

importances = rfcm.feature\_importances\_

np.sum(importances)

plt.barh(fn,importances)

# Draw a bar chart to see the sorted importance values with feature names.

df\_importances = pd.DataFrame(data=importances, index=fn, columns=['importance\_value'])

df\_importances.sort\_values(by = 'importance\_value', ascending=False, inplace=True)

plt.barh(df\_importances.index,df\_importances.importance\_value)

df\_importances50 = df\_importances.iloc[0:50,:]

#Select only the most 50 important features

selected\_columns = df\_importances50.index

X\_reduced = X[selected\_columns]

df = pd.concat([y, X\_reduced], axis=1, ignore\_index=False)

# Specify the file name

file\_name = 'flight\_weather\_original\_fs.csv'

# Writing to CSV file using pandas

df.to\_csv(file\_name, index=False)

# Now, we are ready to build a model using those reduced number of features.

X\_reduced\_train, X\_reduced\_test, y\_reduced\_train, y\_reduced\_test = train\_test\_split(X\_reduced,y,test\_size =.3, stratify=y)

# Build a model with the reduced number of features.

rfcm2 = RandomForestClassifier().fit(X\_reduced\_train, y\_reduced\_train)

y\_reduced\_pred = rfcm2.predict(X\_reduced\_test)

print('\nClassification Report after feature reduction\n')

print(metrics.classification\_report(y\_reduced\_test,y\_reduced\_pred))

##################################################

#FS Using Naive Random Oversampling Dataset

##################################################

fn = X\_rs.columns

print(f'Originally, we have {len(fn)} features.')

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_rs,y\_rs,test\_size =.3, stratify=y\_rs)

# Create an instance (object) for classification and build a model.

rfcm = RandomForestClassifier().fit(X\_train, y\_train)

# Make predictions using the test data

y\_pred = rfcm.predict(X\_test)

# Show the Classification Report.

print('\nClassification Report\n')

print(metrics.classification\_report(y\_test,y\_pred))

# %matplotlib auto

# %matplotlib inline

importances = rfcm.feature\_importances\_

np.sum(importances)

plt.barh(fn,importances)

# Draw a bar chart to see the sorted importance values with feature names.

df\_importances = pd.DataFrame(data=importances, index=fn, columns=['importance\_value'])

df\_importances.sort\_values(by = 'importance\_value', ascending=False, inplace=True)

plt.barh(df\_importances.index,df\_importances.importance\_value)

df\_importances50 = df\_importances.iloc[0:50,:]

#Select only the most 50 important features

selected\_columns = df\_importances50.index

X\_reduced = X\_rs[selected\_columns]

df = pd.concat([y\_rs, X\_reduced], axis=1, ignore\_index=False)

# Specify the file name

file\_name = 'flight\_weather\_random\_fs.csv'

# Writing to CSV file using pandas

df.to\_csv(file\_name, index=False)

# Now, we are ready to build a model using those reduced number of features.

X\_reduced\_train, X\_reduced\_test, y\_reduced\_train, y\_reduced\_test = train\_test\_split(X\_reduced,y\_rs,test\_size =.3, stratify=y\_rs)

# Build a model with the reduced number of features.

rfcm2 = RandomForestClassifier().fit(X\_reduced\_train, y\_reduced\_train)

y\_reduced\_pred = rfcm2.predict(X\_reduced\_test)

print('\nClassification Report after feature reduction\n')

print(metrics.classification\_report(y\_reduced\_test,y\_reduced\_pred))

##################################################

#FS Using SMOTE Dataset

##################################################

fn = X\_sm.columns

print(f'Originally, we have {len(fn)} features.')

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_sm,y\_sm,test\_size =.3, stratify=y\_sm)

# Create an instance (object) for classification and build a model.

rfcm = RandomForestClassifier().fit(X\_train, y\_train)

# Make predictions using the test data

y\_pred = rfcm.predict(X\_test)

# Show the Classification Report.

print('\nClassification Report\n')

print(metrics.classification\_report(y\_test,y\_pred))

# %matplotlib auto

# %matplotlib inline

importances = rfcm.feature\_importances\_

np.sum(importances)

plt.barh(fn,importances)

# Draw a bar chart to see the sorted importance values with feature names.

df\_importances = pd.DataFrame(data=importances, index=fn, columns=['importance\_value'])

df\_importances.sort\_values(by = 'importance\_value', ascending=False, inplace=True)

plt.barh(df\_importances.index,df\_importances.importance\_value)

df\_importances50 = df\_importances.iloc[0:50,:]

#Select only the most 50 important features

selected\_columns = df\_importances50.index

X\_reduced = X\_sm[selected\_columns]

df = pd.concat([y\_sm, X\_reduced], axis=1, ignore\_index=False)

# Specify the file name

file\_name = 'flight\_weather\_smote\_fs.csv'

# Writing to CSV file using pandas

df.to\_csv(file\_name, index=False)

# Now, we are ready to build a model using those reduced number of features.

X\_reduced\_train, X\_reduced\_test, y\_reduced\_train, y\_reduced\_test = train\_test\_split(X\_reduced,y\_sm,test\_size =.3, stratify=y\_sm)

# Build a model with the reduced number of features.

rfcm2 = RandomForestClassifier().fit(X\_reduced\_train, y\_reduced\_train)

y\_reduced\_pred = rfcm2.predict(X\_reduced\_test)

print('\nClassification Report after feature reduction\n')

print(metrics.classification\_report(y\_reduced\_test,y\_reduced\_pred))

### h. Model Data - PCA, then Data Imbalance

##########################################################

#Data Imbalance

##########################################################

from imblearn.over\_sampling import RandomOverSampler

from imblearn.over\_sampling import SMOTE

import numpy as np

import pandas as pd

import os

new\_directory = "C:/Users/yada/Documents/3. School/Classses/11. Capstone/Project/Dataset/Data\_Model"

os.chdir(new\_directory)

data = pd.read\_csv('flight\_weather\_original\_pca.csv')

# Define target variable for classification

y = data['ArrDel15']

# Select features for classification

X = data.drop(['ArrDel15'], axis=1)

X.shape

y.shape

print(f'Original data: {np.unique(y, return\_counts=1)}')

# Create an instance of RandomOverSampler

ros = RandomOverSampler(random\_state=0)

X\_rs, y\_rs = ros.fit\_resample(X, y)

X\_rs.shape

y\_rs.shape

print(f'Over-sampled data: {np.unique(y\_rs, return\_counts=1)}')

df\_rs = pd.concat([y\_rs, X\_rs], axis=1, ignore\_index=False)

# Specify the file name

file\_name = 'flight\_weather\_pca\_random.csv'

# Writing to CSV file using pandas

df\_rs.to\_csv(file\_name, index=False)

# Create an instance of SMOTE

sm = SMOTE(random\_state=0)

X\_sm, y\_sm = sm.fit\_resample(X, y)

X\_sm.shape

y\_sm.shape

print(f'Over-sampled data: {np.unique(y\_sm, return\_counts=1)}')

df\_sm = pd.concat([y\_sm, X\_sm], axis=1, ignore\_index=False)

# Specify the file name

file\_name = 'flight\_weather\_pca\_smote.csv'

# Writing to CSV file using pandas

df\_sm.to\_csv(file\_name, index=False)

### i. Model Data - Feature Selection, then Data Imbalance

##########################################################

#Data Imbalance

##########################################################

from imblearn.over\_sampling import RandomOverSampler

from imblearn.over\_sampling import SMOTE

import numpy as np

import pandas as pd

import os

new\_directory = "C:/Users/yada/Documents/3. School/Classses/11. Capstone/Project/Dataset/Data\_Model"

os.chdir(new\_directory)

data = pd.read\_csv('flight\_weather\_original\_fs.csv')

# Define target variable for classification

y = data['ArrDel15'] # Predicting departure delay

# Select features for classification

X = data.drop(['ArrDel15'], axis=1)

X.shape

y.shape

print(f'Original data: {np.unique(y, return\_counts=1)}')

# Create an instance of RandomOverSampler

ros = RandomOverSampler(random\_state=0)

X\_rs, y\_rs = ros.fit\_resample(X, y)

X\_rs.shape

y\_rs.shape

print(f'Over-sampled data: {np.unique(y\_rs, return\_counts=1)}')

df\_rs = pd.concat([y\_rs, X\_rs], axis=1, ignore\_index=False)

# Specify the file name

file\_name = 'flight\_weather\_fs\_random.csv'

# Writing to CSV file using pandas

df\_rs.to\_csv(file\_name, index=False)

# Create an instance of SMOTE

sm = SMOTE(random\_state=0)

X\_sm, y\_sm = sm.fit\_resample(X, y)

X\_sm.shape

y\_sm.shape

print(f'Over-sampled data: {np.unique(y\_sm, return\_counts=1)}')

df\_sm = pd.concat([y\_sm, X\_sm], axis=1, ignore\_index=False)

# Specify the file name

file\_name = 'flight\_weather\_fs\_smote.csv'

# Writing to CSV file using pandas

df\_sm.to\_csv(file\_name, index=False)

### j. Selecting the Best Dataset

##########################################################

#1. Selecting the best dataset

##########################################################

import numpy as np

import pandas as pd

import os

from sklearn.model\_selection import cross\_val\_score

from sklearn.ensemble import RandomForestClassifier

import time

new\_directory = "C:/Users/yada/Documents/3. School/Classses/11. Capstone/Project/Dataset/Data\_Model"

os.chdir(new\_directory)

data\_original = pd.read\_csv('flight\_weather\_original.csv')

data\_rs = pd.read\_csv('flight\_weather\_random.csv')

data\_sm = pd.read\_csv('flight\_weather\_smote.csv')

data\_pca = pd.read\_csv('flight\_weather\_original\_pca.csv')

data\_random\_pca = pd.read\_csv('flight\_weather\_random\_pca.csv')

data\_smote\_pca = pd.read\_csv('flight\_weather\_smote\_pca.csv')

data\_pca\_random = pd.read\_csv('flight\_weather\_pca\_random.csv')

data\_pca\_smote = pd.read\_csv('flight\_weather\_pca\_smote.csv')

data\_fs = pd.read\_csv('flight\_weather\_original\_fs.csv')

data\_random\_fs = pd.read\_csv('flight\_weather\_random\_fs.csv')

data\_smote\_fs = pd.read\_csv('flight\_weather\_smote\_fs.csv')

data\_fs\_random = pd.read\_csv('flight\_weather\_fs\_random.csv')

data\_fs\_smote = pd.read\_csv('flight\_weather\_fs\_smote.csv')

#######################################################

##1 Original Data

#######################################################

data = data\_original

# Define target variable for classification

y = data['ArrDel15']

# Select features for classification

X = data.drop(['ArrDel15'], axis=1)

# Create a model (object) for classification

model = RandomForestClassifier()

start = time.time()

# Use cross\_val\_score

# The default scoring value is accuracy.

score = np.mean(cross\_val\_score(model,X,y,cv=5))

# Print the scores

print('\*\*\n Mean Scores (Accuracies) for random forest\*\*')

print(f'Mean Score: {score:.4f}')

end = time.time()

print(f'Total run time: {(end - start):.2f} seconds')

#######################################################

##2 Naive Ramdom Dataset

#######################################################

data = data\_rs

# Define target variable for classification

y = data['ArrDel15']

# Select features for classification

X = data.drop(['ArrDel15'], axis=1)

# Create a model (object) for classification

model = RandomForestClassifier()

start = time.time()

# Use cross\_val\_score

# The default scoring value is accuracy.

score = np.mean(cross\_val\_score(model,X,y,cv=5))

# Print the scores

print('\*\*\n Mean Scores (Accuracies) for random forest\*\*')

print(f'Mean Score from naive random dataset: {score:.4f}')

end = time.time()

print(f'Total run time: {(end - start):.2f} seconds')

#######################################################

##3 SMOTE Dataset

#######################################################

data = data\_sm

# Define target variable for classification

y = data['ArrDel15']

# Select features for classification

X = data.drop(['ArrDel15'], axis=1)

# Create a model (object) for classification

model = RandomForestClassifier()

start = time.time()

# Use cross\_val\_score

# The default scoring value is accuracy.

score = np.mean(cross\_val\_score(model,X,y,cv=5))

# Print the scores

print('\*\*\n Mean Scores (Accuracies) for random forest\*\*')

print(f'Mean Score from smote dataset: {score:.4f}')

end = time.time()

print(f'Total run time: {(end - start):.2f} seconds')

#######################################################

##4 Original PCA Dataset

#######################################################

data = data\_pca

# Define target variable for classification

y = data['ArrDel15']

# Select features for classification

X = data.drop(['ArrDel15'], axis=1)

# Create a model (object) for classification

model = RandomForestClassifier()

start = time.time()

# Use cross\_val\_score

# The default scoring value is accuracy.

score = np.mean(cross\_val\_score(model,X,y,cv=5))

# Print the scores

print('\*\*\n Mean Scores (Accuracies) for random forest\*\*')

print(f'Mean Score from original PCA dataset: {score:.4f}')

end = time.time()

print(f'Total run time: {(end - start):.2f} seconds')

#######################################################

##5 Naive Random, then PCA Dataset

#######################################################

data = data\_random\_pca

# Define target variable for classification

y = data['ArrDel15']

# Select features for classification

X = data.drop(['ArrDel15'], axis=1)

# Create a model (object) for classification

model = RandomForestClassifier()

start = time.time()

# Use cross\_val\_score

# The default scoring value is accuracy.

score = np.mean(cross\_val\_score(model,X,y,cv=5))

# Print the scores

print('\*\*\n Mean Scores (Accuracies) for random forest\*\*')

print(f'Mean Score from naive random, then PCA dataset: {score:.4f}')

end = time.time()

print(f'Total run time: {(end - start):.2f} seconds')

#######################################################

##6 SMOTE, then PCA Dataset

#######################################################

data = data\_smote\_pca

# Define target variable for classification

y = data['ArrDel15']

# Select features for classification

X = data.drop(['ArrDel15'], axis=1)

# Create a model (object) for classification

model = RandomForestClassifier()

start = time.time()

# Use cross\_val\_score

# The default scoring value is accuracy.

score = np.mean(cross\_val\_score(model,X,y,cv=5))

# Print the scores

print('\*\*\n Mean Scores (Accuracies) for random forest\*\*')

print(f'Mean Score from smote, then PCA dataset: {score:.4f}')

end = time.time()

print(f'Total run time: {(end - start):.2f} seconds')

#######################################################

##7 PCA, then Naive Random Dataset

#######################################################

data = data\_pca\_random

# Define target variable for classification

y = data['ArrDel15']

# Select features for classification

X = data.drop(['ArrDel15'], axis=1)

# Create a model (object) for classification

model = RandomForestClassifier()

start = time.time()

# Use cross\_val\_score

# The default scoring value is accuracy.

score = np.mean(cross\_val\_score(model,X,y,cv=5))

# Print the scores

print('\*\*\n Mean Scores (Accuracies) for random forest\*\*')

print(f'Mean Score from PCA, then naive random dataset: {score:.4f}')

end = time.time()

print(f'Total run time: {(end - start):.2f} seconds')

#######################################################

##8 PCA, then smote Dataset

#######################################################

data = data\_pca\_smote

# Define target variable for classification

y = data['ArrDel15']

# Select features for classification

X = data.drop(['ArrDel15'], axis=1)

# Create a model (object) for classification

model = RandomForestClassifier()

start = time.time()

# Use cross\_val\_score

# The default scoring value is accuracy.

score = np.mean(cross\_val\_score(model,X,y,cv=5))

# Print the scores

print('\*\*\n Mean Scores (Accuracies) for random forest\*\*')

print(f'Mean Score from PCA, then smote dataset: {score:.4f}')

end = time.time()

print(f'Total run time: {(end - start):.2f} seconds')

#######################################################

##9 Original FS dataset

#######################################################

data = data\_fs

# Define target variable for classification

y = data['ArrDel15']

# Select features for classification

X = data.drop(['ArrDel15'], axis=1)

# Create a model (object) for classification

model = RandomForestClassifier()

start = time.time()

# Use cross\_val\_score

# The default scoring value is accuracy.

score = np.mean(cross\_val\_score(model,X,y,cv=5))

# Print the scores

print('\*\*\n Mean Scores (Accuracies) for random forest\*\*')

print(f'Mean Score from original FS dataset: {score:.4f}')

end = time.time()

print(f'Total run time: {(end - start):.2f} seconds')

#######################################################

##10 Naive Random, then FS dataset

#######################################################

data = data\_random\_fs

# Define target variable for classification

y = data['ArrDel15']

# Select features for classification

X = data.drop(['ArrDel15'], axis=1)

# Create a model (object) for classification

model = RandomForestClassifier()

start = time.time()

# Use cross\_val\_score

# The default scoring value is accuracy.

score = np.mean(cross\_val\_score(model,X,y,cv=5))

# Print the scores

print('\*\*\n Mean Scores (Accuracies) for random forest\*\*')

print(f'Mean Score from naive random, then FS dataset: {score:.4f}')

end = time.time()

print(f'Total run time: {(end - start):.2f} seconds')

#######################################################

##11 SMOTE, then FS dataset

#######################################################

data = data\_smote\_fs

# Define target variable for classification

y = data['ArrDel15']

# Select features for classification

X = data.drop(['ArrDel15'], axis=1)

# Create a model (object) for classification

model = RandomForestClassifier()

start = time.time()

# Use cross\_val\_score

# The default scoring value is accuracy.

score = np.mean(cross\_val\_score(model,X,y,cv=5))

# Print the scores

print('\*\*\n Mean Scores (Accuracies) for random forest\*\*')

print(f'Mean Score from smote, then FS dataset: {score:.4f}')

end = time.time()

print(f'Total run time: {(end - start):.2f} seconds')

#######################################################

##12 FS, then naive random dataset

#######################################################

data = data\_fs\_random

# Define target variable for classification

y = data['ArrDel15']

# Select features for classification

X = data.drop(['ArrDel15'], axis=1)

# Create a model (object) for classification

model = RandomForestClassifier()

start = time.time()

# Use cross\_val\_score

# The default scoring value is accuracy.

score = np.mean(cross\_val\_score(model,X,y,cv=5))

# Print the scores

print('\*\*\n Mean Scores (Accuracies) for random forest\*\*')

print(f'Mean Score from FS, then naive random dataset: {score:.4f}')

end = time.time()

print(f'Total run time: {(end - start):.2f} seconds')

#######################################################

##13 FS, then smote dataset

#######################################################

data = data\_fs\_smote

# Define target variable for classification

y = data['ArrDel15']

# Select features for classification

X = data.drop(['ArrDel15'], axis=1)

# Create a model (object) for classification

model = RandomForestClassifier()

start = time.time()

# Use cross\_val\_score

# The default scoring value is accuracy.

score = np.mean(cross\_val\_score(model,X,y,cv=5))

# Print the scores

print('\*\*\n Mean Scores (Accuracies) for random forest\*\*')

print(f'Mean Score from FS, then smote dataset: {score:.4f}')

end = time.time()

print(f'Total run time: {(end - start):.2f} seconds')

### k. Parameter Tuning for SVM

##########################################################

#1. Classification

##########################################################

import numpy as np

import pandas as pd

import os

import time

new\_directory = "C:/Users/yada/Documents/3. School/Classses/11. Capstone/Project/Dataset/Data\_Model"

os.chdir(new\_directory)

data\_rs = pd.read\_csv('flight\_weather\_pca\_random.csv')

#RandomOverSampler

# Define target variable for classification

y\_rs = data\_rs['ArrDel15']

# Select features for classification

X\_rs = data\_rs.drop(['ArrDel15'], axis=1)

print(f'Over-sampled data: {np.unique(y\_rs, return\_counts=1)}')

#######################################################

## SVM with Random Search

#######################################################

from sklearn.svm import SVC

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import StratifiedKFold

from sklearn.model\_selection import RandomizedSearchCV

###Naive Random Over-Sampling Data

scaler = StandardScaler()

Xn\_rs = scaler.fit\_transform(X\_rs)

# You can change the C\_vals and g\_vals as you see fit.

C\_vals = np.logspace(-3,2,10)

g\_vals = np.logspace(-3,2,10)

params = dict(gamma=g\_vals, C=C\_vals, kernel=['poly','sigmoid','rbf'])

kfolds = StratifiedKFold(n\_splits=5)

start = time.time()

svc = SVC()

rand\_src = RandomizedSearchCV(estimator= svc, param\_distributions = params, cv=kfolds, n\_iter=10, verbose=3)

rand\_src.fit(Xn\_rs, y\_rs)

end = time.time()

print(f'Total run time for SVM: {(end - start):.2f} seconds')

# svcm can be used for predictions or others.

print(f'Best score is {rand\_src.best\_score\_:.4f}')

print(rand\_src.best\_params\_)

results = pd.DataFrame(rand\_src.cv\_results\_)

svc = rand\_src.best\_estimator\_

# {'kernel': 'rbf', 'gamma': 0.046415888336127795, 'C': 2.1544346900318843}

# {'kernel': 'rbf', 'gamma': 0.599484, 'C': 27.8256}, score = 0.9687

### m. Parameter Tuning for Neural Network

# Neural Network Classification

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.neural\_network import MLPClassifier

import pandas as pd

from sklearn.model\_selection import RandomizedSearchCV

import time

import os

new\_directory = "C:/Users/yada/Documents/3. School/Classses/11. Capstone/Project/Dataset/Data\_Model"

os.chdir(new\_directory)

data\_rs = pd.read\_csv('flight\_weather\_pca\_random.csv')

X = data\_rs.drop('ArrDel15', axis=1)

y = data\_rs['ArrDel15']

# Standardize the feature data

scaler = StandardScaler()

Xn = scaler.fit\_transform(X)

nnm\_r = MLPClassifier()

params = {'hidden\_layer\_sizes':[(7), (25), (50), (75), (100), (125), (150), (200),

(125,7), (125,25),(125,50),(125,75),(125,100),

(125,125),(125,150)],

'activation':['tanh','logistic', 'relu'],

'max\_iter': [21000]}

start\_r = time.time()

rand\_src = RandomizedSearchCV(estimator= nnm\_r, param\_distributions = params, cv=5,

n\_iter=1, verbose= 3)

rand\_src.fit(Xn,y)

end\_r = time.time()

# Generate a Report

print('\n\n \*\*Report\*\*')

print(f'The best estimator: {rand\_src.best\_estimator\_}')

print(f'The best parameters:\n {rand\_src.best\_params\_}')

print(f'The best score: {rand\_src.best\_score\_:.4f}')

print(f'Total run time for RandomizedSearchCV: {(end\_r - start\_r):.2f} seconds')

# Check the details of search

results\_rgs = pd.DataFrame(rand\_src.cv\_results\_)

results\_rgs

# Split the data into training and testing subsets.

X\_train, X\_test,y\_train, y\_test = \

train\_test\_split(Xn, y, test\_size =.3,random\_state=1234, stratify=y)

# Create a model (object) for classification

nnm = MLPClassifier(hidden\_layer\_sizes=(30), activation='tanh',

max\_iter=5000, random\_state=1234)

# Build a neural network model

nnm.fit(X\_train, y\_train)

# Calcuate the probability for each class

nnm.predict\_proba(X\_test)

# Based on the probabilities, make a prediction

y\_pred = nnm.predict(X\_test)

# Calculate the accuracy

score\_ = nnm.score(X\_test, y\_test)

print(f'Accuracy: {score\_:.4f}')

### n. Selecting the Best Models

##########################################################

# 1. Classification

##########################################################

from sklearn.neural\_network import MLPClassifier

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

from sklearn.naive\_bayes import GaussianNB

from sklearn.tree import DecisionTreeClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import cross\_val\_score

from sklearn import metrics

from sklearn.model\_selection import train\_test\_split

import numpy as np

import pandas as pd

import os

import time

new\_directory = "C:/Users/yada/Documents/3. School/Classses/11. Capstone/Project/Dataset/Data\_Model"

os.chdir(new\_directory)

data = pd.read\_csv('flight\_weather\_original\_pca.csv')

data\_rs = pd.read\_csv('flight\_weather\_pca\_random.csv')

# Define target variable for classification

y = data['ArrDel15']

# Select features for classification

X = data.drop(['ArrDel15'], axis=1)

print(f'Original data: {np.unique(y, return\_counts=1)}')

# RandomOverSampler

# Define target variable for classification

y\_rs = data\_rs['ArrDel15']

# Select features for classification

X\_rs = data\_rs.drop(['ArrDel15'], axis=1)

print(f'Over-sampled data: {np.unique(y\_rs, return\_counts=1)}')

# Create training data and test data

X\_train, X\_test, y\_train, y\_test = \

train\_test\_split(X, y, test\_size=.3, random\_state=1234, stratify=y)

X\_rs\_train, X\_rs\_test, y\_rs\_train, y\_rs\_test = \

train\_test\_split(X\_rs, y\_rs, random\_state=1234, stratify=y\_rs)

#######################################################

# 1.1Random Forest

#######################################################

# Create a model (object) for classification

model = RandomForestClassifier()

model\_rs = RandomForestClassifier()

# Build a random forest classification model

model.fit(X\_train, y\_train)

model\_rs.fit(X\_rs\_train, y\_rs\_train)

# Make predictions using the test data

y\_pred = model.predict(X\_test)

y\_pred\_rs = model\_rs.predict(X\_test)

# Make predictions using the train data

y\_pred\_train = model.predict(X\_train)

y\_pred\_train\_rs = model\_rs.predict(X\_train)

print('\nClassification Report for training data from original data using Random Forest\n')

print(metrics.classification\_report(y\_train, y\_pred\_train))

print('\nClassification Report for test data from original data using Random Forest\n')

print(metrics.classification\_report(y\_test, y\_pred))

print('\nClassification Report for training data from oversampled model using Random Forest\n')

print(metrics.classification\_report(y\_train, y\_pred\_train\_rs))

print('\nClassification Report for test data from oversampled model using Random Forest\n')

print(metrics.classification\_report(y\_test, y\_pred\_rs))

start = time.time()

# Use cross\_val\_score

model\_rs\_mean\_score = np.mean(cross\_val\_score(model\_rs, X\_rs, y\_rs, cv=5))

# Print the scores

print('\*\*\n Mean Scores (Accuracies) for random forest\*\*')

print(f'Mean Score from naive random data: {model\_rs\_mean\_score:.4f}')

end = time.time()

print(f'Total run time: {(end - start):.2f} seconds')

#######################################################

# 1.2 Logistic Regression

#######################################################

# Classification using Logistic Regression

model = LogisticRegression()

model\_rs = LogisticRegression()

# Build a logistics regression classification model

model.fit(X\_train, y\_train)

model\_rs.fit(X\_rs\_train, y\_rs\_train)

# Make predictions using the test data

y\_pred = model.predict(X\_test)

y\_pred\_rs = model\_rs.predict(X\_test)

# Make predictions using the train data

y\_pred\_train = model.predict(X\_train)

y\_pred\_train\_rs = model\_rs.predict(X\_train)

print('\nClassification Report for training data from original data using logistic regression\n')

print(metrics.classification\_report(y\_train, y\_pred\_train))

print('\nClassification Report for test data from original data using logistic regression\n')

print(metrics.classification\_report(y\_test, y\_pred))

print('\nClassification Report for training data from oversampled model using logistic regression\n')

print(metrics.classification\_report(y\_train, y\_pred\_train\_rs))

print('\nClassification Report for test data from oversampled model using logistic regression\n')

print(metrics.classification\_report(y\_test, y\_pred\_rs))

start = time.time()

# Cross Validation Scores

model\_rs\_mean\_score = np.mean(cross\_val\_score(model\_rs, X\_rs, y\_rs, cv=5))

# Print the scores

print('\*\*\n Mean Scores (Accuracies) for Logistic Regression\*\*')

print(f'Mean Score from naive random data: {model\_rs\_mean\_score:.4f}')

end = time.time()

print(f'Total run time: {(end - start):.2f} seconds')

#######################################################

# 1.3 Decision Tree

#######################################################

# Classification using Decision Tree

model = DecisionTreeClassifier()

model\_rs = DecisionTreeClassifier()

# Build a decision tree classification model

model.fit(X\_train, y\_train)

model\_rs.fit(X\_rs\_train, y\_rs\_train)

# Make predictions using the test data

y\_pred = model.predict(X\_test)

y\_pred\_rs = model\_rs.predict(X\_test)

# Make predictions using the train data

y\_pred\_train = model.predict(X\_train)

y\_pred\_train\_rs = model\_rs.predict(X\_train)

print('\nClassification Report for training data from original data using Decision Tree\n')

print(metrics.classification\_report(y\_train, y\_pred\_train))

print('\nClassification Report for test data from original data using Decision Tree\n')

print(metrics.classification\_report(y\_test, y\_pred))

print('\nClassification Report for training data from oversampled model using Decision Tree\n')

print(metrics.classification\_report(y\_train, y\_pred\_train\_rs))

print('\nClassification Report for test data from oversampled model using Decision Tree\n')

print(metrics.classification\_report(y\_test, y\_pred\_rs))

start = time.time()

# Cross Validation Scores

model\_rs\_mean\_score = np.mean(cross\_val\_score(model\_rs, X\_rs, y\_rs, cv=5))

# Print the scores

print('\*\*\n Mean Scores (Accuracies) for Decision Tree\*\*')

print(f'Mean Score from naive random data: {model\_rs\_mean\_score:.4f}')

end = time.time()

print(f'Total run time: {(end - start):.2f} seconds')

#######################################################

# 1.4 Naive Bayesian

#######################################################

# Classification using Naive Bayesian

model = GaussianNB()

model\_rs = GaussianNB()

# Build a NB classification model

model.fit(X\_train, y\_train)

model\_rs.fit(X\_rs\_train, y\_rs\_train)

# Make predictions using the test data

y\_pred = model.predict(X\_test)

y\_pred\_rs = model\_rs.predict(X\_test)

# Make predictions using the train data

y\_pred\_train = model.predict(X\_train)

y\_pred\_train\_rs = model\_rs.predict(X\_train)

print('\nClassification Report for training data from original data using Naive Bayesian\n')

print(metrics.classification\_report(y\_train, y\_pred\_train))

print('\nClassification Report for test data from original data using Naive Bayesian\n')

print(metrics.classification\_report(y\_test, y\_pred))

print('\nClassification Report for training data from oversampled model using Naive Bayesian\n')

print(metrics.classification\_report(y\_train, y\_pred\_train\_rs))

print('\nClassification Report for test data from oversampled model using Naive Bayesian\n')

print(metrics.classification\_report(y\_test, y\_pred\_rs))

start = time.time()

# Cross Validation Scores

model\_rs\_mean\_score = np.mean(cross\_val\_score(model\_rs, X\_rs, y\_rs, cv=5))

# Print the scores

print('\*\*\n Mean Scores (Accuracies) for Naive Bayesian\*\*')

print(f'Mean Score from naive random data: {model\_rs\_mean\_score:.4f}')

end = time.time()

print(f'Total run time: {(end - start):.2f} seconds')

####################################

# 1.5 SVM

####################################

scaler = StandardScaler()

Xn\_train = scaler.fit\_transform(X\_train)

Xn\_test = scaler.fit\_transform(X\_test)

Xn\_rs\_train = scaler.fit\_transform(X\_rs\_train)

Xn\_rs\_test = scaler.fit\_transform(X\_rs\_test)

start = time.time()

model = SVC(kernel='rbf', C=27.8256, gamma=0.599484,

random\_state=1234).fit(Xn\_train, y\_train)

end = time.time()

print(f'Total run time for SVM: {(end - start):.2f} seconds')

start = time.time()

model\_rs = SVC(kernel='rbf', C=27.8256, gamma=0.599484,

random\_state=1234).fit(Xn\_rs\_train, y\_rs\_train)

end = time.time()

print(f'Total run time for SVM: {(end - start):.2f} seconds')

# Make predictions using the test data

y\_pred = model.predict(Xn\_test)

y\_pred\_rs = model\_rs.predict(Xn\_test)

# Make predictions using the train data

y\_pred\_train = model.predict(Xn\_train)

y\_pred\_train\_rs = model\_rs.predict(Xn\_train)

print('\nClassification Report for training data from original data using SVM\n')

print(metrics.classification\_report(y\_train, y\_pred\_train))

print('\nClassification Report for test data from original data using SVM\n')

print(metrics.classification\_report(y\_test, y\_pred))

print('\nClassification Report for training data from oversampled model using SVM\n')

print(metrics.classification\_report(y\_train, y\_pred\_train\_rs))

print('\nClassification Report for test data from oversampled model using SVM\n')

print(metrics.classification\_report(y\_test, y\_pred\_rs))

start = time.time()

# Cross Validation Scores

model\_rs = SVC(kernel='rbf', C=27.8256, gamma=0.599484, random\_state=1234)

model\_rs\_mean\_score = np.mean(cross\_val\_score(model\_rs, X\_rs, y\_rs, cv=5))

# Print the scores

print('\*\*\n Mean Scores (Accuracies) for SVM\*\*')

print(f'Mean Score from naive random data: {model\_rs\_mean\_score:.4f}')

end = time.time()

print(f'Total run time: {(end - start):.2f} seconds')

####################################

# 1.6 Neural Network

####################################

scaler = StandardScaler()

Xn\_train = scaler.fit\_transform(X\_train)

Xn\_test = scaler.fit\_transform(X\_test)

Xn\_rs\_train = scaler.fit\_transform(X\_rs\_train)

Xn\_rs\_test = scaler.fit\_transform(X\_rs\_test)

model = MLPClassifier(hidden\_layer\_sizes=(125, 150), activation='tanh',

max\_iter=21000, random\_state=1234)

model\_rs = MLPClassifier(hidden\_layer\_sizes=(125, 150), activation='tanh',

max\_iter=21000, random\_state=1234)

start = time.time()

# Build a random forest classification model

model.fit(Xn\_train, y\_train)

end = time.time()

print(f'Total run time for SVM: {(end - start):.2f} seconds')

start = time.time()

# Build a random forest classification model

model\_rs.fit(Xn\_rs\_train, y\_rs\_train)

end = time.time()

print(f'Total run time for SVM: {(end - start):.2f} seconds')

# Make predictions using the test data

y\_pred = model.predict(Xn\_test)

y\_pred\_rs = model\_rs.predict(Xn\_test)

# Make predictions using the train data

y\_pred\_train = model.predict(Xn\_train)

y\_pred\_train\_rs = model\_rs.predict(Xn\_train)

print('\nClassification Report for training data from original data using NN\n')

print(metrics.classification\_report(y\_train, y\_pred\_train))

print('\nClassification Report for test data from original data using NN\n')

print(metrics.classification\_report(y\_test, y\_pred))

print('\nClassification Report for training data from oversampled model using NN\n')

print(metrics.classification\_report(y\_train, y\_pred\_train\_rs))

print('\nClassification Report for test data from oversampled model using NN\n')

print(metrics.classification\_report(y\_test, y\_pred\_rs))

start = time.time()

# Cross Validation Scores

model\_rs\_mean\_score = np.mean(cross\_val\_score(model\_rs, X\_rs, y\_rs, cv=5))

# Print the scores

print('\*\*\n Mean Scores (Accuracies) for SVM\*\*')

print(f'Mean Score from naive random data: {model\_rs\_mean\_score:.4f}')

end = time.time()

print(f'Total run time: {(end - start):.2f} seconds')