Machine Learning 2 - Project Report

Flight Delay Prediction

Group 5:

Fuqian Zou, Glenys Lion, Iris Lee, Kavya Bhat, Mingze Bian

Table of Contents

1. Introduction

Airplane flight delays pose significant operational challenges to airlines, airports, and passengers when it happens. Airlines face financial losses with increased fuel costs, crew overtime expenses, and potentially regulatory fines. Airports can experience logistic complication and congestion issues that can later cascade into more delayed flights. Passengers also have to endure longer wait times and potentially missed connecting flights that negatively impact travel experience. In this project, we develop binary classification models to classify whether a flight will be delayed or not, allowing all parties to take proactive measures to mitigate disruptions to the minimum. Airlines can optimize crew assignments, airports can allocate resources to improve operational efficiency, and passengers can be notified ahead of time to make informed travel decisions.

In this paper, we emphasize not only accurately identifying delays, but also effectively handling the inherent class imbalance in the dataset. Since delays are relatively infrequent compared to on-time flights, it is critical that the models do not overly favor the majority class. We experiment with multiple machine learning models that we have learned in class, including Logistic Regression, Neural Networks, Decision Trees, K-Nearest Neighbors (KNN), Generalized Additive Models (GAM), Random Forest, and Gradient Boosting, applying various class weighting ratios to give more importance to the minority class. Because Boosted Tree indicates the strongest performance result, we explored using XGBoost, which is the enhanced version of Boosted Tree. Our goal is to determine the most robust approach for predicting delays while ensuring a fair balance between both classes. To evaluate model performance, we prioritize metrics such as Precision, Recall, and F1-score, as simple accuracy can be misleading in an imbalanced dataset. Recall is particularly important in our context, as it measures the model's ability to

correctly identify delayed flights, helping us foresee potential disruptions and mitigate them effectively.

2. Data Collection

The airplane flight dataset was originally collected from the Bureau of Transportation Statistics and National Centers for Environmental Information (NOAA) and then published on Kaggle. This dataset has detailed airline, weather, airport and employment information. It consists of 6 million rows and 24 columns of predictors, both categorical and numerical data. Below is the list of variables:

Category	Variable Name	Description
Response Variable	DEP_DEL15	Flight delay >15 min (1) or on-time (0)
Categorical Variables	DEP_TIME_BLK	Departure Time Block (e.g., 09001000 means 9-10 AM)
Categorical Variables	CARRIER_NAME	Airline carrier name
Categorical Variables	DEPARTING_AIRPORT	Airport departing from
Categorical Variables	PREVIOUS_AIRPORT	Previous airport before this flight
Numerical Variables - Flight Schedule	MONTH	Month of the flight
Numerical Variables - Flight Schedule	DAY_OF_WEEK	Day of the week
Numerical Variables - Flight Characteristics	DISTANCE_GROUP	Flight distance group
Numerical Variables - Flight Characteristics	SEGMENT_NUMBER	Segment number of flight
Numerical Variables - Flight Characteristics	CONCURRENT_FLIGHTS	Number of concurrent flights
Numerical Variables - Flight Characteristics	NUMBER_OF_SEATS	Total number of seats on flight
Numerical Variables - Airline and Airport Performance	AIRPORT_FLIGHTS_MONTH	Number of flights at airport per month
Numerical Variables - Airline and Airport Performance	AIRLINE_FLIGHTS_MONTH	Number of flights by airline per month
Numerical Variables - Airline and Airport Performance	AIRLINE_AIRPORT_FLIGHTS_MONTH	Number of flights by airline at a specific airport per month
Numerical Variables - Airline and Airport Performance	AVG_MONTHLY_PASS_AIRPORT	Average monthly passengers at airport
Numerical Variables - Airline and Airport Performance	AVG_MONTHLY_PASS_AIRLINE	Average monthly passengers by airline
Numerical Variables - Operational Details	FLT_ATTENDANTS_PER_PASS	Flight attendants per passenger
Numerical Variables - Operational Details	GROUND_SERV_PER_PASS	Ground service staff per passenger
Numerical Variables - Operational Details	PLANE_AGE	Age of the aircraft
Numerical Variables - Geographic Coordinates	LATITUDE	Latitude of departure airport
Numerical Variables - Geographic Coordinates	LONGITUDE	Longitude of departure airport
Numerical Variables - Weather Conditions	PRCP	Precipitation levels
Numerical Variables - Weather Conditions	SNOW	Snowfall levels
Numerical Variables - Weather Conditions	SNWD	Snow depth
Numerical Variables - Weather Conditions	TMAX	Maximum temperature
Numerical Variables - Weather Conditions	AWND	Average wind speed

3. Exploratory Data Analysis

Understanding the patterns and factors contributing to flight delays is essential for building an effective predictive model. This section examines class distribution, temporal trends, weather conditions, aircraft characteristics, and correlation analysis to identify the most relevant predictors.

3.1 Class Distribution

The dataset consists of 6,489,062 flight records, with 81.1% of flights departing on time and 18.9% experiencing delays. This significant class imbalance poses a challenge for predictive modeling, as most standard machine learning algorithms tend to favor the majority class, leading to poor recall for delayed flights. To mitigate this issue, class weighting techniques were applied during model training to ensure that delayed flights were adequately represented in the learning process.

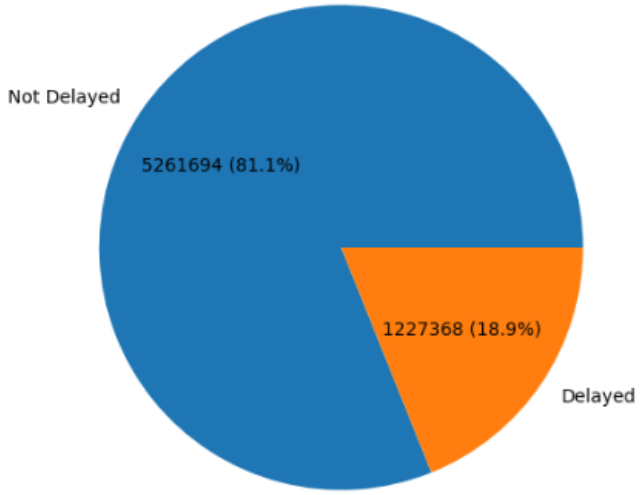

Figure 1: Proportion of delayed and Non delayed Flights

3.2 Temporal Patterns in Flight Delays

Flight delays exhibit clear seasonal and time-of-day trends, which are critical considerations for predictive modeling. Monthly analysis reveals that delays are not evenly distributed throughout the year. The highest average delay times occur in June and July, coinciding with peak summer travel demand. February and December also show increased delay rates, likely influenced by winter weather conditions and holiday congestion. These seasonal variations suggest that incorporating the month of departure as a predictor could enhance model performance.

Time-of-day analysis further reinforces the influence of operational factors on delays. Flights scheduled for the evening and night experience the highest delay rates, while early morning and late-night flights have the lowest probability of delay. This pattern suggests a cumulative effect, where minor operational inefficiencies and air traffic congestion throughout the day contribute to worsening delays as the day progresses. The strong correlation between departure time and delay probability indicates that this feature plays a significant role in delay prediction.

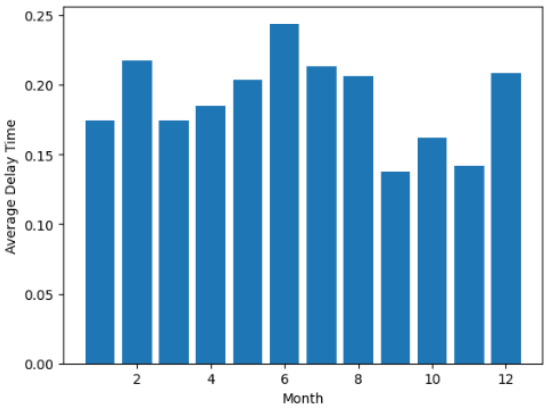

Figure 2: Average Delay time by Month

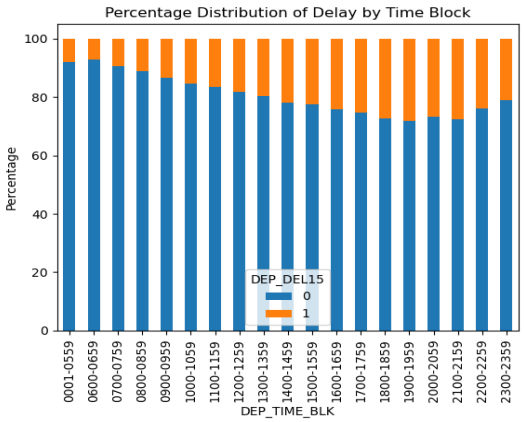

Figure 3: Average Delay time by Time of Day

3.3 Weather Conditions and Flight Delays

Weather conditions significantly impact flight delays, with precipitation and snowfall emerging as key contributors. Delayed flights consistently experience higher average precipitation levels compared to non-delayed flights, particularly during winter months and the summer storm season. The increased delay rates during these periods suggest that adverse weather disrupts airport operations, flight schedules, and air traffic control efficiency. Given this strong association, precipitation was retained as a predictor in the model.

A similar relationship is observed with snowfall, which contributes to higher delay probabilities, particularly in January and February. In contrast, during months with little to no snowfall, the difference between delayed and non-delayed flights is minimal. This finding highlights the importance of incorporating snowfall data into the model, especially when predicting delays in regions with frequent winter storms.

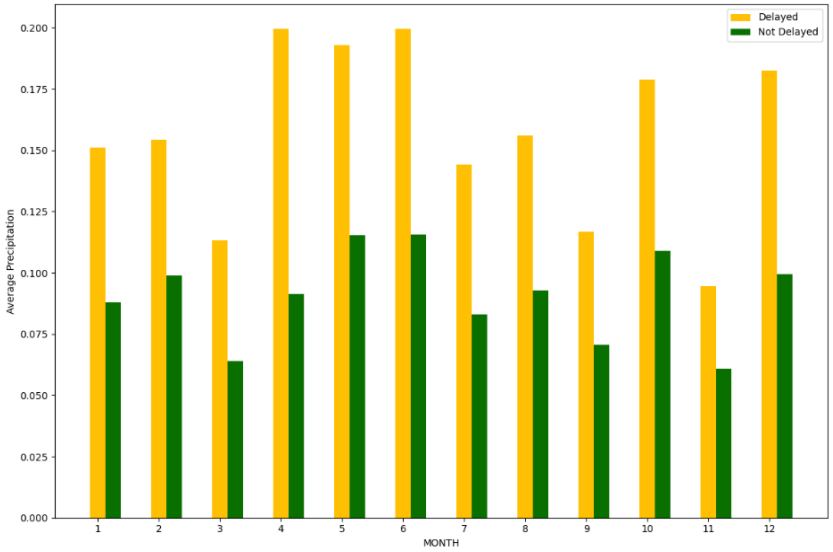

Figure 4: Average Precipitation of Delayed and Not Delayed Flights by Month

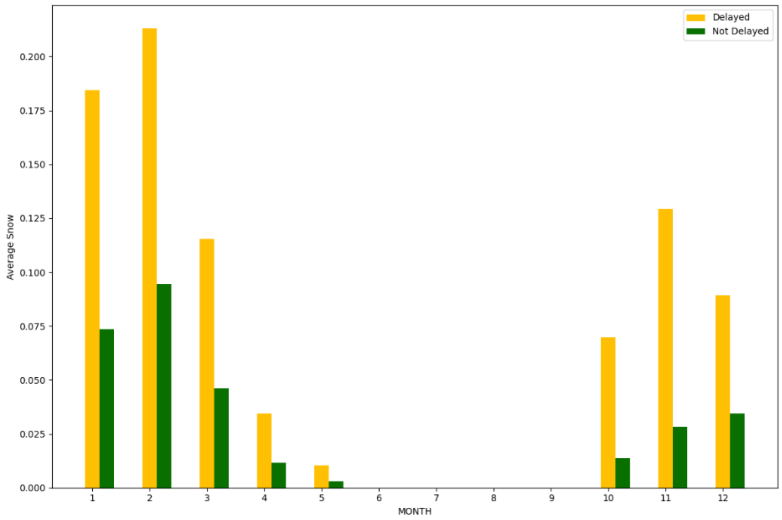

Figure 5: Average Snow of Delayed and Not Delayed Flights by Month

3.4 Aircraft Characteristics and Delay Rates

Aircraft age also plays a role in determining flight delays. Planes aged 15 to 20 years exhibit the highest delay rates, whereas older aircraft, particularly those over 25 years old, show a lower probability of delay. This non-linear relationship suggests that mid-life aircraft may face more frequent maintenance-related disruptions, whereas older aircraft are either phased out or maintained under stricter operational guidelines. The significant difference in delay patterns across aircraft age groups supports the inclusion of this feature in the model.

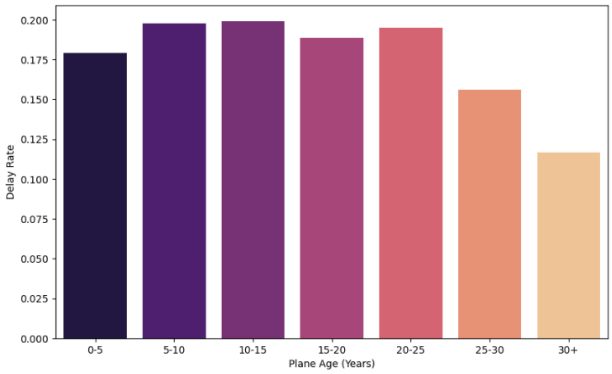

Figure 6: Average Delay Rate by Plane Age Bin

3.5 Correlation Analysis

A correlation heatmap was used to assess the relationships between numerical variables and flight delays. The analysis revealed that several operational metrics, such as the number of monthly flights per airline and airport, exhibited strong correlations with each other. To avoid redundancy, highly correlated variables were removed to improve model interpretability and prevent multicollinearity issues.

Departure time emerged as one of the strongest predictors of delays, further confirming its importance in the model. Weather-related features, including precipitation, snowfall, and wind speed, showed moderate correlations with delays, reinforcing their relevance for prediction. The combination of these insights guided feature selection, ensuring that the model retained only the most informative variables while minimizing redundancy.

Through this exploratory data analysis, key factors influencing flight delays were identified, shaping the data preprocessing and modeling decisions. By incorporating temporal trends, weather conditions, aircraft characteristics, and carefully selected features, the predictive model is better equipped to capture the complex relationships driving flight delays.

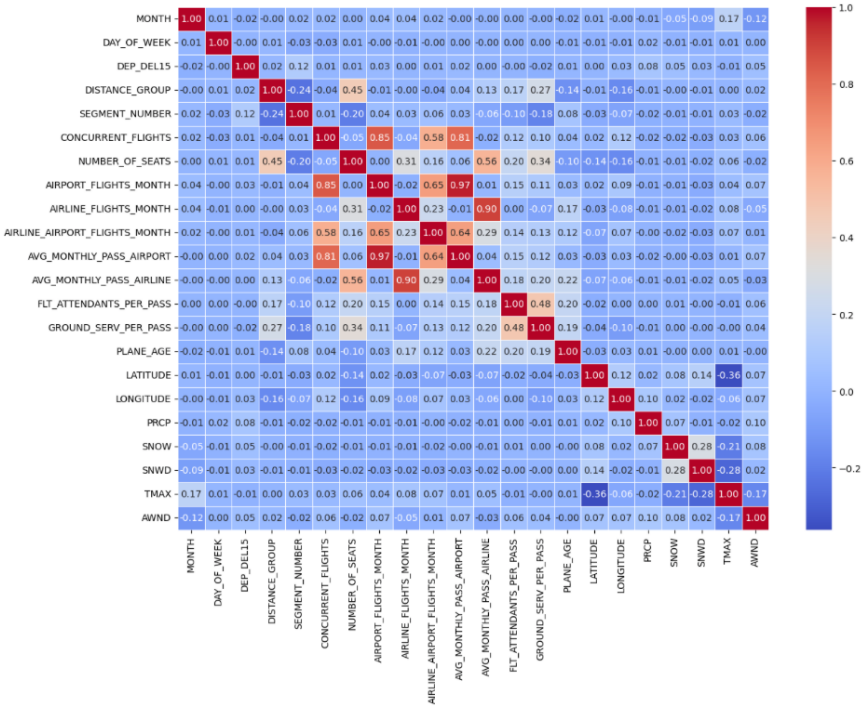

Figure 7: Correlation Heatmap

4. Data Preprocessing

Due to the large sample size, we applied stratified sampling on 1% of the data to maintain the original class distribution. We also removed duplicate records and confirmed that there were no missing values. This resulted in a final dataset of 51,685 rows, with 80.98% representing non-delayed flights and 19.02% representing delayed flights.

To prepare the data for modeling, we applied appropriate encoding techniques for categorical variables, using either one-hot encoding or label encoding depending on the nature of the variable. Additionally, we transformed the original categorical feature DEP_TIME_BLK (departure time block) into DEP_TIME_START, representing the departure hour as a numeric variable. This modification allows the model to capture potential time-based patterns in flight

delays that can have some predictive power. Finally, we retained the full set of features for initial modeling, with feature selection performed in the next step to identify the most relevant predictors.

It was observed that when including all features, complex models struggle to capture meaningful patterns in the data. This is likely due to excessive noise introduced by overlapping or redundant variables, which can hinder model performance. To address this, we implemented an initial logistic regression model to identify the most relevant features and eliminate irrelevant ones. This feature selection process helped enhance model generalization and reduced potential bias.

To refine the feature set, we conducted a Variance Inflation Factor (VIF) analysis to detect and mitigate multicollinearity. Additionally, we applied stepwise selection using the drop1 method to remove less significant predictors. These techniques enabled us to retain only the most impactful variables, ideally to enhance the model's predictive performance. The final 13 predictors selected for modeling are as follow: MONTH, DAY_OF_WEEK, DISTANCE_GROUP, SEGMENT_NUMBER, NUMBER_OF_SEATS, PLANE_AGE, GROUND_SERV_PER_PASS, PRCP, SNOW, SNWD, TMAX, AWND DEP_TIME_START, and the response variable DEP_DEL15.

The dataset was split into 80% training and 20% testing, reserved to evaluate model performances. Additionally, predictors were standardized, ensuring that features are on a comparable scale for best evaluation results. To prevent data leakage, test data was scaled using the mean and standard deviation calculated from training data. For each model, a stratified k-fold of 5 was used to preserve class proportion when tuning hyperparameters.

5. Model Selection and Training

5.1 Logistic Regression

Although it is not the focus of our class, we begin by running a logistic regression as our benchmark baseline result before implementing other nonlinear methods. Logistic regression has several advantages - it is simple and highly interpretable. The defined parametric function allows us to easily understand the relationship of each predictors and its impact. However, it only performs well when the relationship between predictors and the response variable is approximately linear. Logistic regression also does not inherently handle class imbalance unless balancing techniques are used jointly. As a result, our test performance lands at a F1 score of 1.75%, indicating a notably poor balance between precision 41.43% and recall 9.00%, and that the model almost never correctly detects delayed flights. Though implementing balancing techniques may help with performance results, we suspect there is nonlinearity in our data thus did not proceed further using logistic.

5.2 Classification Tree

As our first nonlinear model to explore, classification trees provide almost a similar level of interpretability while capturing nonlinear relationships that a logistic regression cannot. However, without proper tuning, the tree tends to overfit on the training set when it grows too deep, leading to poor generalization on unseen data. Tree is tuned at best cp where the crossed validation error is minimized with minbucket of 5. The base decision tree model performance remained subpar without proper class imbalance adjustment. The built in class weighting function was used to address this issue. By increasing weights from 2 to 4 ratio, it significantly improved recall and F1-score ending at 55.33% and 37.6%. Feature importance analysis highlights that

departure time (DEP_TIME_START), segment number, number of seats, and weather conditions (TMAX, PRCP) are the key predictors of delays.

5.3 Neural Network

Neural networks are powerful models for capturing complex, non-linear relationships between features. In our R implementation, we utilized a single hidden layer neural network architecture. Like other models, neural networks are prone to overfitting, especially with imbalanced data such as our flight delay dataset. To mitigate this challenge, we performed 5-fold cross-validation, carefully tuning the number of hidden nodes and decay values to achieve an optimal balance between model complexity and regularization.

The best performing neural network model was trained on the full training set and achieved 81.16% accuracy on the test set. However, despite performing well on predicting the majority class (on-time flights), it struggled significantly with minority class recall at just 4.27%, resulting in a low F1-score of 7.94%. This performance pattern was consistent with other models that lacked proper class weighting.

Unlike tree-based models, neural networks do not provide direct feature importance metrics, making interpretation more challenging. To address this limitation, we used Accumulate Local Effects (ALE) plots, shown in figure 8, which revealed important insights. These plots highlighted weather conditions (particularly SNOW and PRCP), NUMBER_OF_SEATS, and departure time (DEP_TIME_START) as key factors influencing flight delays. This analysis confirmed findings from other models regarding the critical impact of both operational factors and weather conditions on flight delays.

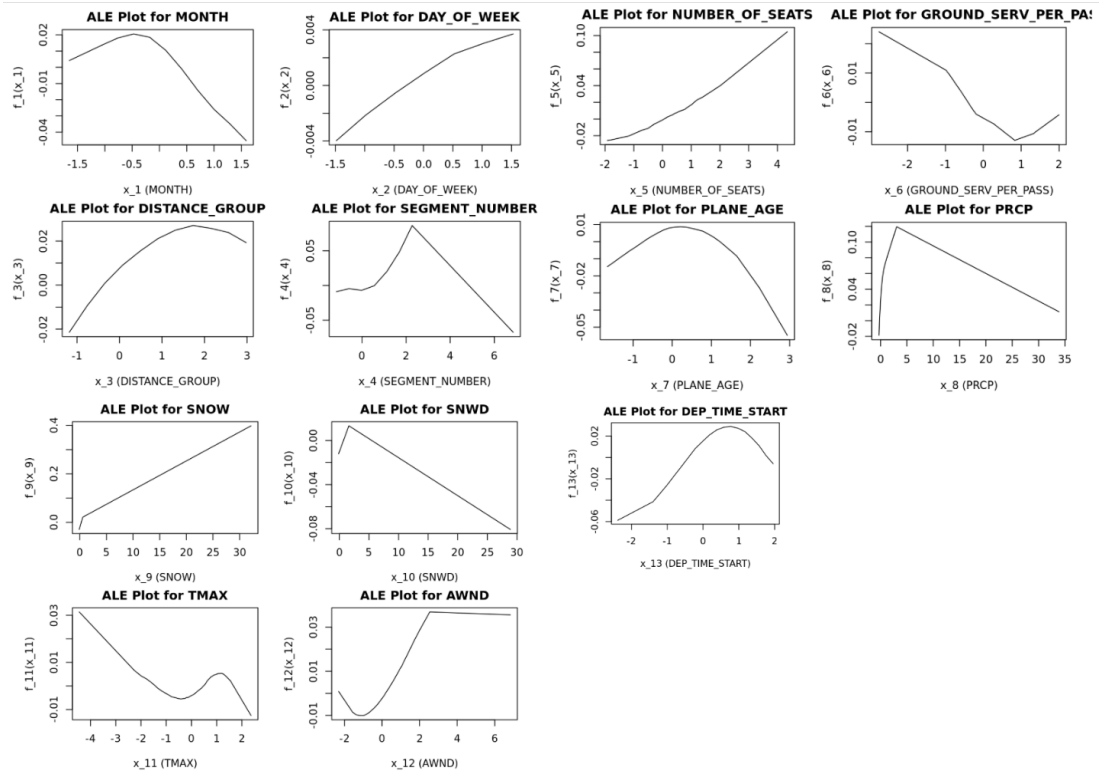

Figure 8: ALE Plots from Neural Network Model

5.4 K-Nearest Neighbors

K-Nearest Neighbors (KNN) is a distance-based, instance-based learning method that classifies new observations by a majority vote among the k closest training samples. It assumes that similar predictor values correspond to similar outcomes but struggles with high-dimensional data and large datasets due to computationally intensive distance calculations.

To optimize performance, we conducted comprehensive hyperparameter tuning, varying k from 3 to 40 in increments of 2, and implemented 5-fold cross-validation to select the optimal k-value based on the highest F1-score. The final model achieved 76.9% accuracy on the test set, with 16.28% recall and a 21.14% F1-score. While the overall accuracy was moderate, the relatively low recall indicates that KNN had difficulty identifying delayed flights accurately, showing limited

effectiveness in capturing the minority class patterns without additional adjustments like class weighting.

5.5 Generalized Additive Model

Generalized additive models (GAMs) extend the standard linear framework by applying non-linear smoothing functions to numeric predictors, offering the ability to flexibly model complex relationships while maintaining interpretability. This makes GAMs particularly useful when the relationship between predictors and the response variable is non-linear but still needs to be explained.

For our dataset, we built a GAM with spline smoothing of degree 5, applied to all numeric predictors to capture potential non-linear effects. We evaluated model performance using 5-fold cross-validation to ensure reliable assessment of the model's predictive capabilities. The final model achieved a reasonably high accuracy of 81.06% and good precision of 56.32%. However, its recall was extremely low at just 1.99%, resulting in a poor F1-score of 3.85%. This performance pattern indicates that while GAM effectively identifies non-delayed flights, it significantly struggles to detect actual delays. This limitation likely seems from the model's difficulty in capturing minority class patterns through its smoothing functions, especially in an imbalanced dataset where the non-linear relationships for delayed flights may be overwhelmed by the dominant patterns of on-time flights.

5.6 Boosted Tree

Boosted tree is an advanced ensemble technique that sequentially builds decision trees, with each new tree correcting the errors of its predecessor. We implemented a boosted tree model,

and GBM's built-in cross-validation determined that an optimal ensemble of 670 trees minimizes log loss, preventing overfitting.

The base boosted tree achieved 81.19% accuracy, but recall was extremely low (5.29%), resulting in a weak F1-score of 9.61%. Feature importance analysis (figure 9) identified DEP_TIME_START, PRCP, TMAX, SEGMENT_NUMBER, and AWND as the top predictors of flight delays.

var	rel.inf
DEP_TIME_START	25.115232
PRCP	16.122586
TMAX	8.761933
SEGMENT_NUMBER	8.248157
AWND	8.143648
NUMBER_OF_SEATS	6.690024
GROUND_SERV_PER_PASS	6.643598
MONTH	5.838251
PLANE_AGE	3.619657
DISTANCE_GROUP	3.513701
SNOW	3.484108
DAY_OF_WEEK	2.665675
SNWD	1.153431

Figure 9: Variable Importance from Boosted Tree Model

To improve recall, we applied class weighting, increasing the importance of the minority class. The best performance was achieved with a 4:1 weight ratio, boosting recall to 58.67% and the F1-score to 39.53%. Although accuracy decreased to 65.85%, the significant recall improvement indicates a better ability to detect delayed flights, leading to a more balanced performance. Overall, the boosted tree model effectively captures key predictors while leveraging class weighting to enhance sensitivity to the minority class, making it a valuable approach for flight delay prediction.

5.7 Random Forest

Random Forest is an ensemble learning algorithm that constructs multiple decision trees and aggregates their predictions to improve accuracy. Unlike boosting methods, it grows trees independently, reducing overfitting while employing bootstrap sampling and feature randomization for better generalization.

To optimize performance, we performed 5-fold cross-validation to select the best node size, ensuring the model did not overfit. Smaller node sizes increase the risk of overfitting, while larger sizes enhance generalization by reducing sensitivity to noise.

The baseline Random Forest model achieved high accuracy but suffered from low recall (6.88%), leading to a weak F1-score of 12.26%. One challenge in improving recall is that the key predictors for delayed flights ("Yes") differ from those for on-time flights ("No"). From figure 10, predicting delays relies more on PRCP, AWND, and DAY_OF_WEEK, while predicting on-time flights is influenced primarily by DEP_TIME_START, SEGMENT_NUMBER, and NUMBER_OF_SEATS. Since different feature sets drive each class, balancing precision and recall is inherently difficult, making class weighting, threshold adjustments, and feature selection necessary.

```
##                             No        Yes MeanDecreaseAccuracy MeanDecreaseGini
## MONTH                46.665682   1.211133             49.06817        965.33782
## DAY_OF_WEEK           8.352290   9.666601             12.06687        968.22392
## DISTANCE_GROUP       59.617093 -20.550144             51.14339        970.71923
## SEGMENT_NUMBER       98.181442 -75.767962             96.01273        769.64417
## NUMBER_OF_SEATS      70.498757 -24.805815             66.75949       1170.66256
## GROUND_SERV_PER_PASS 60.359977 -12.679712             59.06658        867.92560
## PLANE_AGE            51.630390 -19.827828             43.72597       1469.73670
## PRCP                 35.676266  45.128582             53.83092        952.25461
## SNOW                 18.062501   8.919377             22.76465        105.31242
## SNWD                  9.695626   5.280322             11.92881         88.99459
## TMAX                 46.114665  -8.529692             48.21230       1946.21209
## AWND                 18.704049  12.209826             23.50755       1913.50862
## DEP_TIME_START      108.752236 -41.461648            125.76073       1422.28710
```

Figure 10: Variable Importance from Random Forest Model

To address this imbalance, we applied a class-weighted system, assigning greater importance to delayed flights. The best performance was achieved with a 4:1 weight ratio, where recall increased to 8.5%, improving the F1-score to 14.63%. Although accuracy slightly decreased, the improvement in recall indicates a better ability to detect delayed flights, demonstrating a more balanced prediction approach.

5.8 Extreme Gradient Boosting

Aside from the models covered in lectures, we implemented Extreme Gradient Boosting (XGBoost) after observing that gradient boosting machines (GBM) produced the best performance metrics so far. XGBoost is an advanced tree-based method that sequentially builds decision trees to optimize performance. Unlike traditional Gradient Boosting, XGBoost incorporates L1 (Lasso) and L2 (Ridge) regularization, reducing overfitting and improving robustness. To further prevent overfitting, we performed stratified k-fold cross-validation with early stopping on log loss, while class weighting was applied to balance the dataset and enhance detection of delayed flights.

The baseline XGBoost model achieved the highest accuracy (81.4%), but recall remained low (4.68%), leading to a weak F1-score of 8.74%. While effective in predicting on-time flights, the model struggled with delays. To address this, we experimented with class weighting, and the best performance was achieved with a 4:1 weight ratio, significantly improving recall to 58.46% and the F1-score to 39.61%, demonstrating a better trade-off between precision and recall.

Feature importance from figure 11 identified DEP_TIME_START as the strongest predictor, followed by PRCP and SEGMENT_NUMBER, aligning with insights from other models. These findings highlight the significant role of departure time, weather conditions, and flight segment structure in predicting flight delays.

Feature	Gain	Cover	Frequency
<char>	<num>	<num>	<num>
DEP_TIME_START	0.313081033	0.09455181	0.10156581
PRCP	0.130172254	0.09901154	0.08675413
SEGMENT_NUMBER	0.095261097	0.05802716	0.06644096
MONTH	0.075538088	0.08406023	0.08336860
TMAX	0.075071001	0.12764090	0.13711384
GROUND_SERV_PER_PASS	0.072176389	0.10572095	0.09098603
NUMBER_OF_SEATS	0.068943901	0.10899330	0.11172239
AWND	0.062351895	0.11728749	0.12103259
DISTANCE_GROUP	0.032421892	0.04001193	0.04655099
PLANE_AGE	0.029011573	0.06354122	0.06686416
DAY_OF_WEEK	0.025181095	0.05350339	0.04909014
SNOW	0.011590278	0.03120789	0.02327550
SNWD	0.009199504	0.01644219	0.01523487

Figure 11: Variable Importance from XGBoost Model

6. Model Comparison and Evaluation

Models	Type	Accuracy	Precision	Recall	F1
Logistic Regression	Statistical	0.8092	0.4314	0.009	0.0175
Decision Tree	Tree- Based	0.8113	0.6786	0.0155	0.0302
Decision Tree Weighted Ratio 2:1		0.8022	0.4378	0.1404	0.2126
Decision Tree Weighted Ratio 3:1		0.718	0.3145	0.4089	0.3555
Decision Tree Weighted Ratio 4:1		0.6506	0.2847	0.5533	0.376
Neural Network	Neural Network	0.8116	0.5645	0.0427	0.0794
KNN	Instance-Based	0.769	0.3014	0.1627	0.2114
GAM	Semi-Parametric	0.8106	0.5632	0.0199	0.0385

Boosted Tree	Boosting	0.8119	0.5603	0.0529	0.0967
Boosted Tree Weighted Ratio 2:1		0.798	0.4324	0.1977	0.2714
Boosted Tree Weighted Ratio 3:1		0.7444	0.3515	0.4068	0.3771
Boosted Tree Weighted Ratio 4:1		0.6585	0.2981	0.5867	0.3953
Random Forest	Ensemble	0.8127	0.5633	0.0688	0.1226
Random Forest Weighted Ratio 2:1		0.8112	0.5215	0.0887	0.1516
Random Forest Weighted Ratio 3:1		0.8118	0.5327	0.0862	0.1485
Random Forest Weighted Ratio 3:1		0.8112	0.5238	0.085	0.1463
XGBoost	Gradient Boosting	0.814	0.6571	0.0468	0.0874
XGBoost Weighted Ratio 2:1		0.7982	0.4311	0.1896	0.2634
XGBoost Weighted Ratio 3:1		0.742	0.3456	0.3983	0.3701
XGBoost Weighted Ratio 4:1		0.6609	0.2995	0.5846	0.3961

7. Conclusion

Our analysis of flight delay prediction shows important findings about what makes this problem challenging. The main challenge is that most flights are not delayed, by 81% of the whole data, while only 19% are delayed. This makes it hard for models to correctly predict delays without special adjustment.

One key finding from our Random Forest model is that different factors matter when predicting delayed versus on-time flights. On-time flights are best predicted by departure time, segment number, and number of seats. However, delayed flights depend more on precipitation, wind speed, and day of the week. This difference explains why it is so difficult to predict both delayed and non-delayed flights well using the same set of features.

We found that our models worked much better when we gave extra importance to delayed flights during training, especially with a 4:1 weighted ratio. This approach helped our boosted tree models and XGBoost achieve a better balance between identifying actual delays and avoiding false alarms. Departure time consistently emerged as the most important predictor across models, with evening and night flights showing significantly higher delay rates than morning departures, based on the ALE Plot from the Neural Network model. Weather factors like snow, rain, and temperature were also consistently important predictors, confirming that both scheduled timing and weather conditions strongly affect flight delays.

For airlines and airports, we suggest using these weighted boosted tree or XGBoost models with the key factors we identified. While no model can predict delays perfectly, even reasonably accurate predictions can help airlines, airports, and passengers better prepare and adjust their plans when delays are likely to occur.

8. Appendix

8.1 Feature Selection using Logistic Regression Code

```
df <- read.csv("full_data_flightdelay.csv")
df <- df[!duplicated(df), ]

library(caret)
set.seed(1234)
index <- createDataPartition(df$DEP_DEL15, p = 0.01, list = FALSE)
df_sample <- df[index, ]
train <- df_sample

# create new feature - season
season <- c(
  "12" = "Winter", "1" = "Winter", "2" = "Winter",
  "3" = "Spring", "4" = "Spring", "5" = "Spring",
  "6" = "Summer", "7" = "Summer", "8" = "Summer",
  "9" = "Fall", "10" = "Fall", "11" = "Fall"
)
train$SEASON <- season[as.character(train$MONTH)]

# because PREVIOUS_AIRPORT has a large number of unique values, it would create high
dimension if using one hot encoding. frequency encoding is used for this variable.
train$PREVIOUS_AIRPORT_FREQ <-
table(train$PREVIOUS_AIRPORT)[train$PREVIOUS_AIRPORT]

# through EDA, we notice there is clear trend of delay probability versus departure time block.
convert time block to numeric to represent ordinal character
train$DEP_TIME_START <- as.numeric(sub("-.*", "", train$DEP_TIME_BLK))
train$DEP_TIME_START <- ifelse(train$DEP_TIME_START %% 100 == 0,
train$DEP_TIME_START / 100, train$DEP_TIME_START)
train$CARRIER_NAME_FREQ <- table(train$CARRIER_NAME)[train$CARRIER_NAME]
train$DEPARTING_AIRPORT_FREQ <-
table(train$DEPARTING_AIRPORT)[train$DEPARTING_AIRPORT]

# factorizing the categorical variables
train$DEP_DEL15 <- as.factor(train$DEP_DEL15)
train$DISTANCE_GROUP <- as.factor(train$DISTANCE_GROUP)
train$SEGMENT_NUMBER <- as.factor(train$SEGMENT_NUMBER)
```

```
train$DEP_TIME_START <- as.factor(train$DEP_TIME_START)
train$MONTH <- as.factor(train$MONTH)
train$DAY_OF_WEEK <- as.factor(train$DAY_OF_WEEK)

# standardize numeric predictors
numeric_cols <- sapply(train, is.numeric)
train[numeric_cols]<-sapply(train[numeric_cols], function(x) (x-mean(x))/sd(x))

# removing the category that we have processed it
library(dplyr)
train <- train %>%
  select(-c(DEP_TIME_BLK, CARRIER_NAME, DEPARTING_AIRPORT,
PREVIOUS_AIRPORT))

logit_model <- glm(DEP_DEL15 ~ ., data = train, family = binomial)
summary(logit_model)

drop1(logit_model, test = "Chisq")

# remove the non significant variables
train_2 <- train <- train %>%
  select(-c(SEASON))

# check the logistic regression model again
logit_model2 <- glm(DEP_DEL15 ~ ., data = train_2, family = binomial)
summary(logit_model2)

# use vif to check multicollinearity problem
library(car)
vif(logit_model2)
drop1(logit_model2, test = "Chisq")

# remove the non-important variables
train_2 <- train_2 %>%
  select(-c(AIRPORT_FLIGHTS_MONTH, AIRLINE_FLIGHTS_MONTH,
AVG_MONTHLY_PASS_AIRPORT, AVG_MONTHLY_PASS_AIRLINE,
PREVIOUS_AIRPORT_FREQ, CARRIER_NAME_FREQ, DEPARTING_AIRPORT_FREQ))

# fit the logistic regression model to check once more
logit_model3 <- glm(DEP_DEL15 ~ ., data = train_2, family = binomial)
```

```
summary(logit_model3)
drop1(logit_model3, test = "Chisq")

# remove the not significant one
data_significant <- train_2 %>%
  select(-c(CONCURRENT_FLIGHTS, AIRLINE_AIRPORT_FLIGHTS_MONTH,
FLT_ATTENDANTS_PER_PASS, LATITUDE, LONGITUDE))
# check the correlation between variables
cor_matrix <- cor(data_significant[, sapply(data_significant, is.numeric)])

# save it to csv so that everyone can use the same data for the modeling part
write.csv(data_significant, "data_for_modeling_clean.csv", row.names = FALSE)
```

8.2 Logistic Regression Model Code

```
train_control <- trainControl(method = "cv", number = 5, classProbs = TRUE, summaryFunction
= twoClassSummary)

logistic_model <- train(DEP_DEL15 ~ ., data = train_data, method = "glm", family = binomial,
trControl = train_control, metric = "ROC")

# Predict on test set
predictions <- predict(logistic_model, test_data)
conf_matrix <- confusionMatrix(predictions, test_data$DEP_DEL15, positive = "Yes")

# Store results
model_results_df <- add_model_results("Logistic Regression",
                                      conf_matrix$overall["Accuracy"],
                                      conf_matrix$byClass["Precision"],
                                      conf_matrix$byClass["Recall"],
                                      conf_matrix$byClass["F1"])
```

8.3 Classification Tree Model Code

```
# cross validation
control <- rpart.control(minbucket = 5, cp = 0.0001, maxsurrogate = 0, usesurrogate = 0, xval =
10)
tree_model <- rpart(DEP_DEL15 ~ ., data = train_data, method = "class", control = control)
best_cp <- tree_model$cptable[which.min(tree_model$cptable[, "xerror"]), "CP"]
# prune the tree
```

```
prune_tree <- prune(tree_model, cp = best_cp)
prune_tree$variable.importance
# get the prediction and the confusion matrix
test_preds <- predict(prune_tree, test_data, type = "class")
conf_matrix_tree <- confusionMatrix(test_preds, test_data$DEP_DEL15, positive = "Yes")

# compile the result
model_results_df <- add_model_results("Decision Tree",
                                      conf_matrix_tree$overall["Accuracy"],
                                      conf_matrix_tree$byClass["Precision"],
                                      conf_matrix_tree$byClass["Recall"],
                                      conf_matrix_tree$byClass["F1"])

# weighted tree model
tree_model_weighted_3 <- rpart(DEP_DEL15 ~ ., data = train_data, method = "class",
                               control = rpart.control(cp = best_cp),
                               parms = list(loss = matrix(c(0, 4, 1, 0), nrow = 2)))
test_preds_tree_weighted_3 <- predict(tree_model_weighted_3, test_data, type = "class")
conf_matrix_tree_weighted_3 <- confusionMatrix(test_preds_tree_weighted_3,
test_data$DEP_DEL15, positive = "Yes")

model_results_df <- add_model_results("Decision Tree Weighted Ration 4:1",
                                      conf_matrix_tree_weighted_3$overall["Accuracy"],
                                      conf_matrix_tree_weighted_3$byClass["Precision"],
                                      conf_matrix_tree_weighted_3$byClass["Recall"],
                                      conf_matrix_tree_weighted_3$byClass["F1"])
```

8.4 Neural Network Model Code

```
# create the cross validation function
set.seed(123)
CVInd <- function(n,K) {
 m<-floor(n/K) #approximate size of each part
 r<-n-m*K
 I<-sample(n,n) #random reordering of the indices
 Ind<- vector("list", K)
 for (k in 1:K) {
 if (k <= r) kpart <- ((m+1)*(k-1)+1):((m+1)*k)
 else kpart<-((m+1)*r+m*(k-r-1)+1):((m+1)*r+m*(k-r))
 Ind[[k]] <- I[kpart] #indices for kth part of data
 }
```

```
 return(Ind)
}
set.seed(123)
K <- 5
hidden_nodes <- c(3, 5, 10)
decay_values <- c(0.1, 0.3, 0.5)
n.models <- length(hidden_nodes) * length(decay_values)
n <- nrow(train_data)
y <- train_data$DEP_DEL15
CV_metrics <- matrix(0, n.models, 4)
model_list <- list()

Ind <- CVInd(n, K)
model_index <- 1
for (h in hidden_nodes) {
  for (d in decay_values) {
    yhat <- numeric(n)
    for (k in 1:K) {
      test_idx <- Ind[[k]]
      train_idx <- setdiff(1:n, test_idx)

      nn_model <- nnet(DEP_DEL15 ~ ., data = train_data[train_idx, ], size = h, decay = d, maxit
= 500, linout = FALSE, skip = FALSE, trace = FALSE)
      yhat[test_idx] <- predict(nn_model, train_data[test_idx, ], type = "class")
    }

    confusion <- confusionMatrix(factor(yhat), factor(y), positive = "Yes")
    CV_metrics[model_index, ] <- c(confusion$overall["Accuracy"],
confusion$byClass["Precision"], confusion$byClass["Recall"], confusion$byClass["F1"])
    model_list[[model_index]] <- list(model = nn_model, size = h, decay = d)
    model_index <- model_index + 1
  }
}

# Identify the best model
best_model_index <- which.max(CV_metrics[, 4]) # best F1 score
best_model <- model_list[[best_model_index]]$model

# store the best hidden nodes and decay
best_hideen_nodes <- model_list[[best_model_index]]$size
```

```
best_decay <- model_list[[best_model_index]]$decay

best_nn_model <- nnet(DEP_DEL15 ~ ., data = train_data, size = best_hideen_nodes, decay =
best_decay, maxit = 500, linout = FALSE, skip = FALSE, trace = FALSE)

nn_pred <- predict(best_nn_model, test_data, type = "class")
conf_matrix_nnet <- confusionMatrix(factor(nn_pred), factor(test_data$DEP_DEL15), positive
= "Yes")
model_results_df <- add_model_results("Neural Network",
                                      conf_matrix_nnet$overall["Accuracy"],
                                      conf_matrix_nnet$byClass["Precision"],
                                      conf_matrix_nnet$byClass["Recall"],
                                      conf_matrix_nnet$byClass["F1"])
# ALE Plot for Neural Network
library(ALEPlot)
ale_results <- list()
feature_names <- colnames(train_data)[colnames(train_data) != "DEP_DEL15"]

pred.fun <- function(X.model, newdata) {
  predict(best_nn_model, newdata, type = "raw")
}

par(mfrow = c(2, 2))
par(mar = c(4, 4, 2, 2))

for (feature in feature_names) {
  ale_plot <- ALEPlot(
    X = train_data[, feature_names],
    pred.fun = pred.fun,
    J = which(feature_names == feature),
    K = 50
  )

  title(main = paste("ALE Plot for", feature))
  ale_results[[feature]] <- ale_plot
}

par(mfrow = c(1, 1))
```

8.5 K Nearest Neighbors Model Code

```
set.seed(123)
K <- 5  # K-fold CV
k_values <- seq(3, 40, 2)  # Candidate values for k
n <- nrow(train_data)
y <- train_data$DEP_DEL15
X <- train_data[, colnames(train_data) != "DEP_DEL15"]
CV_metrics <- matrix(0, length(k_values), 4)

Ind <- CVInd(n, K)
for (i in 1:length(k_values)) {
  k <- k_values[i]
  yhat <- factor(rep(NA, n), levels = levels(y))
  for (fold in 1:K) {
    test_idx <- Ind[[fold]]
    train_idx <- setdiff(1:n, test_idx)
    yhat[test_idx] <- knn(train = X[train_idx, ], test = X[test_idx, ], cl = y[train_idx], k = k)
  }

  confusion <- confusionMatrix(yhat, y, positive = "Yes")
  CV_metrics[i, ] <- c(confusion$overall["Accuracy"], confusion$byClass["Precision"],
confusion$byClass["Recall"], confusion$byClass["F1"])
}

# Identify the best k
best_k_index <- which.max(CV_metrics[, 4])
best_k <- k_values[best_k_index]

# Print the best k
cat("Best k:", best_k, "\n")

X_train <- train_data[, colnames(train_data) != "DEP_DEL15"]
y_train <- train_data$DEP_DEL15
X_test <- test_data[, colnames(test_data) != "DEP_DEL15"]
y_test <- test_data$DEP_DEL15

knn_test_preds <- knn(train = X_train, test = X_test, cl = y_train, k = best_k)
conf_matrix_knn <- confusionMatrix(knn_test_preds, y_test, positive = "Yes")
```

```
model_results_df <- add_model_results("KNN",
                                      conf_matrix_knn$overall["Accuracy"],
                                      conf_matrix_knn$byClass["Precision"],
                                      conf_matrix_knn$byClass["Recall"],
                                      conf_matrix_knn$byClass["F1"])
```

8.6 Generalized Additive Model (GAM) Model Code

```
K <- 5  # K-fold CV
n <- nrow(train_data)
y <- train_data$DEP_DEL15
X <- train_data[, colnames(train_data) != "DEP_DEL15"]
CV_metrics <- matrix(0, K, 4)

Ind <- CVInd(n, K)
yhat <- factor(rep(NA, n), levels = levels(y))

numeric_vars <- names(X)[sapply(X, is.numeric)]
categorical_vars <- names(X)[sapply(X, is.factor)]

gam_formula <- as.formula(paste("DEP_DEL15 ~",
                          paste(c(
                            paste0("s(", numeric_vars, ", k=5)"),
                            categorical_vars
                          ), collapse = " + ")))

for (fold in 1:K) {
  test_idx <- Ind[[fold]]
  train_idx <- setdiff(1:n, test_idx)

  gam_model <- gam(gam_formula, data = train_data[train_idx, ], family = binomial)
  pred_probs <- predict(gam_model, train_data[test_idx, ], type = "response")
  yhat[test_idx] <- factor(ifelse(pred_probs > 0.5, "Yes", "No"), levels = levels(y))

  confusion <- confusionMatrix(yhat[test_idx], y[test_idx], positive = "Yes")
  CV_metrics[fold, ] <- c(confusion$overall["Accuracy"], confusion$byClass["Precision"],
confusion$byClass["Recall"], confusion$byClass["F1"])
}
best_gam_model <- gam(gam_formula, data = train_data, family = binomial)
gam_probs <- predict(best_gam_model, test_data, type = "response")
```

```
gam_preds <- factor(ifelse(gam_probs > 0.5, "Yes", "No"), levels =
levels(test_data$DEP_DEL15))

conf_matrix_gam <- confusionMatrix(gam_preds, test_data$DEP_DEL15, positive = "Yes")

model_results_df <- add_model_results("GAM",
                                      conf_matrix_gam$overall["Accuracy"],
                                      conf_matrix_gam$byClass["Precision"],
                                      conf_matrix_gam$byClass["Recall"],
                                      conf_matrix_gam$byClass["F1"])
```

8.7 Boosted Tree Model Code

```
gbm1 <- gbm(as.numeric(train_data$DEP_DEL15 == "Yes")~., data = train_data, distribution =
"bernoulli", n.trees = 3000,
        shrinkage = 0.1, interaction.depth = 3, bag.fraction = 0.5, train.fraction = 1,
        n.minobsinnode = 10, cv.folds = 5, keep.data = TRUE, verbose = FALSE)
best.iter <- gbm.perf(gbm1, method = "cv")
best.iter <- gbm.perf(gbm1, method = "cv")
summary(gbm1, n.trees = best.iter)
gbm1_prob <- predict(gbm1, test_data, n.trees = best.iter, type = "response")
gbm1_pred <- ifelse(gbm1_prob > 0.5, "Yes", "No")
gbm1_pred <- factor(gbm1_pred, levels = levels(test_data$DEP_DEL15))
conf_matrix_gbm1 <- confusionMatrix(gbm1_pred, test_data$DEP_DEL15, positive = "Yes")

model_results_df <- add_model_results("Boosted Tree",
                                      conf_matrix_gbm1$overall["Accuracy"],
                                      conf_matrix_gbm1$byClass["Precision"],
                                      conf_matrix_gbm1$byClass["Recall"],
                                      conf_matrix_gbm1$byClass["F1"])

# weighted boosted tree model. Here only shown the 4:1 ratio
class_weights <- ifelse(train_data$DEP_DEL15 == "Yes", 4, 1)
gbm4 <- gbm(as.numeric(train_data$DEP_DEL15 == "Yes") ~ ., data = train_data, distribution
= "bernoulli", n.trees = 3000,
        shrinkage = 0.1, interaction.depth = 3, bag.fraction = 0.5, train.fraction =
1,n.minobsinnode = 10,
        cv.folds = 5, keep.data = TRUE, verbose = FALSE,weights = class_weights)
best.iter4 <- gbm.perf(gbm4, method = "cv")
summary(gbm4, n.trees = best.iter)
gbm4_prob <- predict(gbm4, test_data, n.trees = best.iter3, type = "response")
```

```
gbm4_pred <- ifelse(gbm4_prob > 0.5, "Yes", "No")
gbm4_pred <- factor(gbm4_pred, levels = levels(test_data$DEP_DEL15))
conf_matrix_gbm4 <- confusionMatrix(gbm4_pred, test_data$DEP_DEL15, positive = "Yes")

model_results_df <- add_model_results("Boosted Tree Weighted Ratio 4:1",
                                      conf_matrix_gbm4$overall["Accuracy"],
                                      conf_matrix_gbm4$byClass["Precision"],
                                      conf_matrix_gbm4$byClass["Recall"],
                                      conf_matrix_gbm4$byClass["F1"])
```

8.7 Random Forest Model Code

```
# custom f1 score function bcs there is no f1 score on the caret rf
custom_summary <- function(data, lev = NULL, model = NULL) {
  precision <- posPredValue(data$pred, data$obs, positive = "Yes")
  recall <- sensitivity(data$pred, data$obs, positive = "Yes")
  F1 <- ifelse(precision + recall > 0, 2 * (precision * recall) / (precision + recall), 0)

  return(c(Accuracy = mean(data$pred == data$obs), Precision = precision, Recall = recall, F1 =
F1))
}

set.seed(123)

train_control <- trainControl(
  method = "cv",
  number = 5,
  classProbs = TRUE,
  summaryFunction = custom_summary,  # Use custom F1 function
  savePredictions = "final"
)

nodesize_values <- c(3, 5, 10, 15, 20)
fixed_mtry <- 4

rf_results <- data.frame(nodesize = nodesize_values, F1 = NA)

for (i in 1:length(nodesize_values)) {

  rf_model <- train(
    DEP_DEL15 ~ .,
```

```
    data = train_data,
    method = "rf",
    trControl = train_control,
    tuneGrid = expand.grid(mtry = fixed_mtry),
    metric = "F1",
    importance = TRUE,
    nodesize = nodesize_values[i],
    ntree = 500
  )

  rf_results$F1[i] <- max(rf_model$results$F1, na.rm = TRUE)
}

# Find the best nodesize based on highest F1-score
best_nodesize <- rf_results$nodesize[which.max(rf_results$F1)]

# Print results
cat("Best nodesize:", best_nodesize, "\n")
rf_model <- randomForest(DEP_DEL15 ~ ., data = train_data, mtry = 4, ntree = 500, nodesize =
best_nodesize, importance = TRUE)
plot(rf_model)
importance(rf_model)
rf_test_preds <- predict(rf_model, test_data, type = "class")
conf_matrix_rf <- confusionMatrix(as.factor(rf_test_preds), as.factor(test_data$DEP_DEL15),
positive = "Yes")

model_results_df <- add_model_results("Random Forest",
                                      conf_matrix_rf$overall["Accuracy"],
                                      conf_matrix_rf$byClass["Precision"],
                                      conf_matrix_rf$byClass["Recall"],
                                      conf_matrix_rf$byClass["F1"])
# weighted RF for 4:1 ratio
rf_model_weighted4 <- randomForest(DEP_DEL15 ~ ., data = train_data,
                                   mtry = 4, ntree = 500, nodesize = best_nodesize,
                                   importance = TRUE, classwt = c("No" = 1, "Yes" = 4))

rf_test_preds_weighted4 <- predict(rf_model_weighted4, test_data, type = "class")
conf_matrix_rf4 <- confusionMatrix(as.factor(rf_test_preds_weighted4),
as.factor(test_data$DEP_DEL15), positive = "Yes")
```

```
model_results_df <- add_model_results("Random Forest Weighted Ratio 3:1",
                                      conf_matrix_rf4$overall["Accuracy"],
                                      conf_matrix_rf4$byClass["Precision"],
                                      conf_matrix_rf4$byClass["Recall"],
                                      conf_matrix_rf4$byClass["F1"])
```

8.8 XGBoost Model Code

```
# XGBoost requires matrix input
train_matrix <- model.matrix(DEP_DEL15 ~ ., data = train_data)[,-1]  # Remove intercept
train_label <- ifelse(train_data$DEP_DEL15 == "Yes", 1, 0)

# Convert to XGBoost format
dtrain <- xgb.DMatrix(data = train_matrix, label = train_label)
params <- list(
  objective = "binary:logistic",
  eval_metric = "logloss",
  max_depth = 3,
  eta = 0.1,
  subsample = 0.8,
  colsample_bytree = 0.8
)

xgb_cv <- xgb.cv(
  params = params,
  data = dtrain,
  nrounds = 500,
  nfold = 5,
  stratified = TRUE,
  verbose = 0,
  early_stopping_rounds = 10
)

# best iter
best_nrounds <- xgb_cv$best_iteration
print(best_nrounds)
xgb_model <- xgboost(
  params = params,
  data = dtrain,
  nrounds = best_nrounds,
```

```
  verbose = 0
)

# Convert test data into XGBoost matrix
test_matrix <- model.matrix(DEP_DEL15 ~ ., data = test_data)[,-1]
test_label <- as.numeric(test_data$DEP_DEL15 == "Yes")
dtest <- xgb.DMatrix(data = test_matrix, label = test_label)
xgb_probs <- predict(xgb_model, dtest)
xgb_preds <- ifelse(xgb_probs > 0.5, "Yes", "No")
xgb_preds <- factor(xgb_preds, levels = levels(test_data$DEP_DEL15))
conf_matrix_xgb <- confusionMatrix(xgb_preds, test_data$DEP_DEL15, positive = "Yes")

model_results_df <- add_model_results("XGBoost",
                                      conf_matrix_xgb$overall["Accuracy"],
                                      conf_matrix_xgb$byClass["Precision"],
                                      conf_matrix_xgb$byClass["Recall"],
                                      conf_matrix_xgb$byClass["F1"])

# try the XGBoost with 4:1 ratio
params <- list(
  objective = "binary:logistic",
  eval_metric = "logloss",
  max_depth = 3,
  eta = 0.1,
  subsample = 0.8,
  colsample_bytree = 0.8,
  scale_pos_weight = 4
)

xgb_cv_weighted <- xgb.cv(
  params = params,
  data = dtrain,
  nrounds = 500,
  nfold = 5,
  stratified = TRUE,
  verbose = 0,
  early_stopping_rounds = 10
)

best_nrounds_weighted <- xgb_cv_weighted$best_iteration
```

```
# train the best nrounds
xgb_model_weighted <- xgboost(
  params = params,
  data = dtrain,
  nrounds = best_nrounds_weighted,
  verbose = 0
)
# Convert test data into XGBoost matrix
test_matrix <- model.matrix(DEP_DEL15 ~ ., data = test_data)[,-1]
test_label <- as.numeric(test_data$DEP_DEL15 == "Yes")
dtest <- xgb.DMatrix(data = test_matrix, label = test_label)

xgb_probs_weighted <- predict(xgb_model_weighted, dtest)
xgb_preds_weighted <- ifelse(xgb_probs_weighted > 0.5, "Yes", "No")
xgb_preds_weighted <- factor(xgb_preds_weighted, levels = levels(test_data$DEP_DEL15))

conf_matrix_xgb_weighted <- confusionMatrix(xgb_preds_weighted, test_data$DEP_DEL15,
positive = "Yes")

model_results_df <- add_model_results("XGBoost Weighted Ratio 4:1",
                                      conf_matrix_xgb_weighted$overall["Accuracy"],
                                      conf_matrix_xgb_weighted$byClass["Precision"],
                                      conf_matrix_xgb_weighted$byClass["Recall"],
                                      conf_matrix_xgb_weighted$byClass["F1"])
```