Capstone Project - Flight Delay Prediction

Jebbe Schellevis

2 December 2021

Contents

1 Introduction

One of the most popular fields within data science is machine learning. It allows data scientists to use (large) datasets to train algorithms and use them to predict future or (yet) unknown events. In this project, machine learning is applied to predict if a certain airline flight will be delayed, based on historical data. A practical application for such algorithms might be in flight booking system, to inform users on the risk of delays for certain flights. This report describes the data, methodology, and results for the machine learning project mentioned.

1.1 Goal

The goal of this project is to develop an algorithm that predicts if a certain airline flight will be delayed, given predicting factors such as the operating airline, destination, date, and time of day. There are several types of predictions possible (e.g. delay yes/no, delay 0-15 minutes, 15-30 minutes, etc. or even the probability of a delay). Finding out the best type of prediction is part of this project.

To assess the quality of the algorithm, one can look at measures like accuracy, sensitivity, specificity, and precision of the prediction. Accuracy basically refers to the percentage of cases that were predicted correctly, while sensitivity (also known as recall) and specificity are measures of true positives and true negatives respectively. Precision, lastly, measures true positives as a proportion of all positives predicted. Looking at different measures allows one to examine model performance more closely than just by the number of correct predictions. For example, the overall accuracy might be 80%, with a sensitivity of 95% but a specificity of 50%. This means that almost all cases where there is a delay are correctly predicted, but there are also many cases where a predicted delay in reality does not occur.

In this project we will look at all aforementioned measures, but take the F1 score as a preferred quality indicator. The F1 score provides a balanced indication of model performance, taking into account both precision and recall:

$F1 = 2 * (Recall * Precision)/(Recall + Precision)$

When we train multiclass classification models (predicting one of five delay time classes), we get an F1 score for each class. To aggregate to an overall score for the entire model, we can use the so-called micro-weighted F1 score, which is equal to the accuracy of the model (note 1).

And finally, we will look at a custom developed performance metric called 'ProbSpec' or 'probability specificity' to measure the models' performance in providing specific probabilities for different groups of flights (see chapter 2.7).

1.2 Dataset

The data used in this machine learning project is a dataset compiled by the United States Bureau of Transportation Statistics (BTS), downloaded from https://www.transtats.bts.gov/DL_SelectFields.asp?gnoyr_VQ=FGJ&QO_fu146_anzr=b0-gvzr. The dataset used contains records on all US domestic flights in 2019, around 7.3 million in total. For each flight, the dataset includes several characteristics (see table on next page).

For practical use, in this project the dataset is filtered to only include flights departing from San Francisco (SFO) airport. The resulting dataset contains 166,750 records. This dataset is then split into training and testing datasets, as described in more detail in chapter 2.

Table 1: Dataset preview

DepYear	DepMonth	DepDay	DepDate	UniqueCarrier	CarrierNo	FlightNo	Origin	Dest	PlanDepTime	DepTime	PlanArrTime	ArrTime	ArrDelay	Cancelled	Distance
2019	1	26	2019-01-26	9E	9E	3442	CVG	MSP	17:31	17:27	18:43	18:16	0	0	596
2019	1	7	2019-01-07	9E	9E	3442	DTW	SDF	08:35	08:30	10:26	09:48	0	0	306
2019	1	8	2019-01-08	9E	9E	3442	DTW	SDF	08:35	08:30	10:26	09:52	0	0	306
2019	1	9	2019-01-09	9E	9E	3442	DTW	SDF	08:35	08:25	10:26	09:54	0	0	306
2019	1	10	2019-01-10	9E	9E	3442	DTW	SDF	08:35	08:26	10:26	10:03	0	0	306
2019	1	11	2019-01-11	9E	9E	3442	DTW	SDF	08:35	08:30	10:26	09:48	0	0	306

1.3 Approach

The main steps in this data science project are:

1. Load and clean dataset
2. Enrich dataset with additional columns
3. Perform exploratory data analysis
4. Split dataset into training and test sets
5. Train different machine learning algorithms and evaluate their predictive power
6. Summarize methodology and results in this report

A more detailed, technical explanation of steps 1 through 6 is provided in chapter 2.

2 Methodology

This chapter describes the methodology used in this data science project. For each of the aforementioned steps, it details the methods and functions used. Some of the code used is (visibly) included in this report, but only when it supports the storyline and/or adds to the understanding of the reader. The full body of code including commentary can be found in a separate file `edx_flightdelay.r`.

2.1 General and report

This analysis was done using the language R, mostly used for statistical analysis. Scripts were developed in RStudio 1.4.1717 on Linux Debian 11. The version of R used is 4.0.4. This report was set up in R Markdown, a markup language that allows for R code and regular text to be mixed into a readable document.

Basic knowledge and methodology on R, data preparation, and different machine learning models are derived from and inspired by examples in the textbook Introduction to Data Science by Rafael A. Irizzary (2021) (note 2). References to specific other sources are made in this report where applicable; full source description is included at the end of the document.

2.2 Load and clean dataset

In order to load the dataset, first CSV files containing flights for each month in 2019 were downloaded manually from de link mentioned in chapter 1.2. Next, each file is opened, reading each line and inserting it into a tibble. Then we remove quotes around each value, set consistent, clear column titles, set the right column datatypes and filter out rows with NA values. This results in a clean, tidy dataset, ready for further enrichment, saved as a separate CSV file.

2.3 Enrich dataset

Based on the information in the dataset, several additional columns can be generated with additional information. The following columns are added:

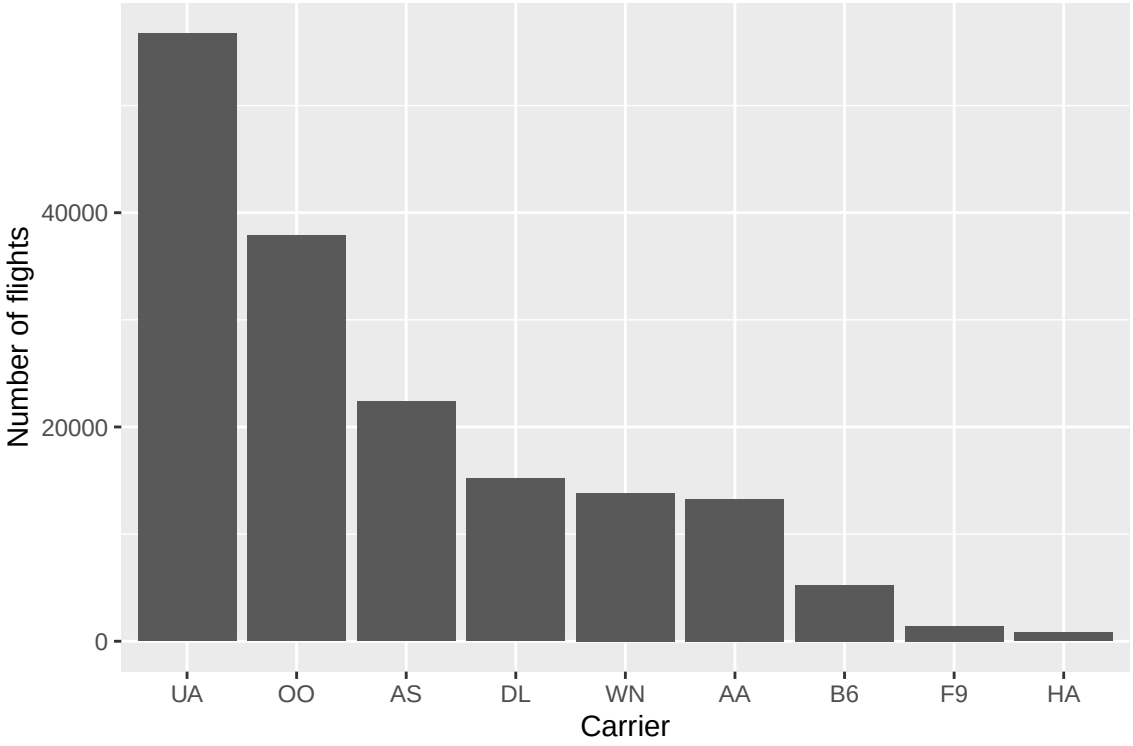
- Week number, based on the flight date
- Delay flag (0,1), based on the actual arrival delay; a flight is considered delayed when the arrival delay is more than 15 minutes
- Delay category (0-15 min, 16-30 min, 31-60 min, 61-120 min, >120 min), based on the actual arrival delay
- Flight number, based on carrier code and route number
- Planned departure time slot (rounded to 1-hour slots), based on the planned departure time

After this step, the enriched dataset is saved as a seperate CSV file.

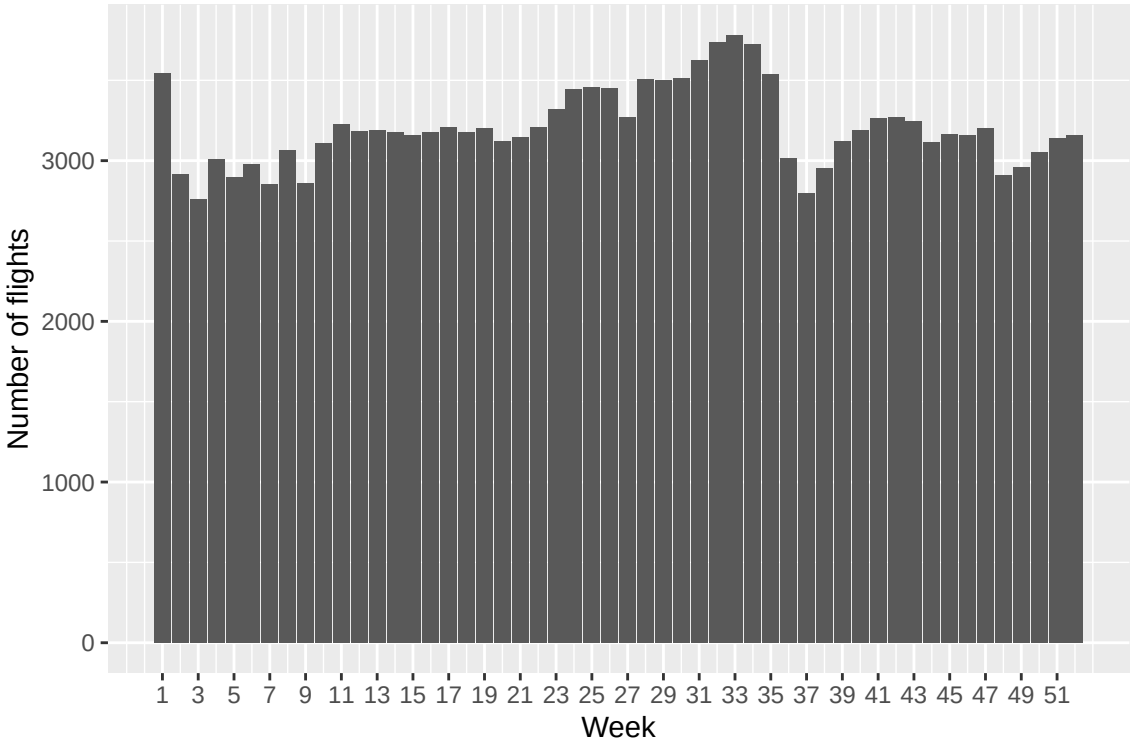
2.4 Exploratory data analysis

Having created an enriched dataset, it is time for some exploratory analysis on the dataset (EDA). This will help our understanding of the distribution of the data and can give preliminary insights into what columns may serve as good predictors for the machine learning model.

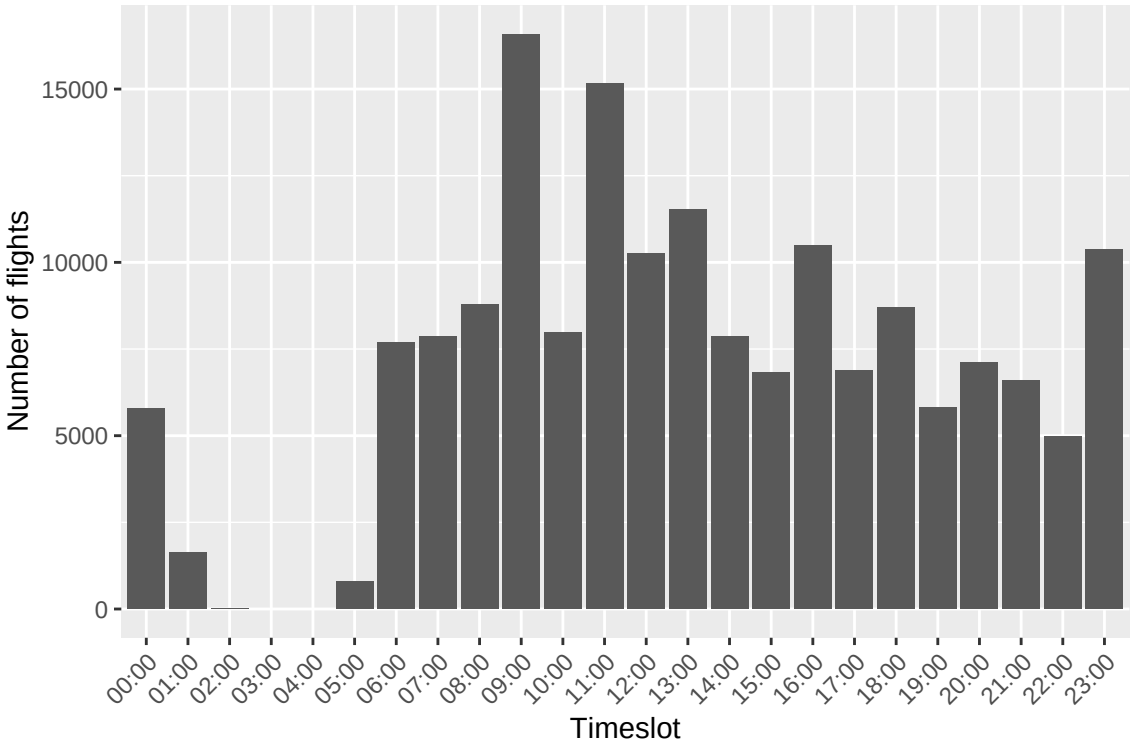
2.4.1 Flight distribution The first part of the EDA is focused on understanding the distribution of flight numbers in terms of who (carriers), when (dates/times), and where (destinations).



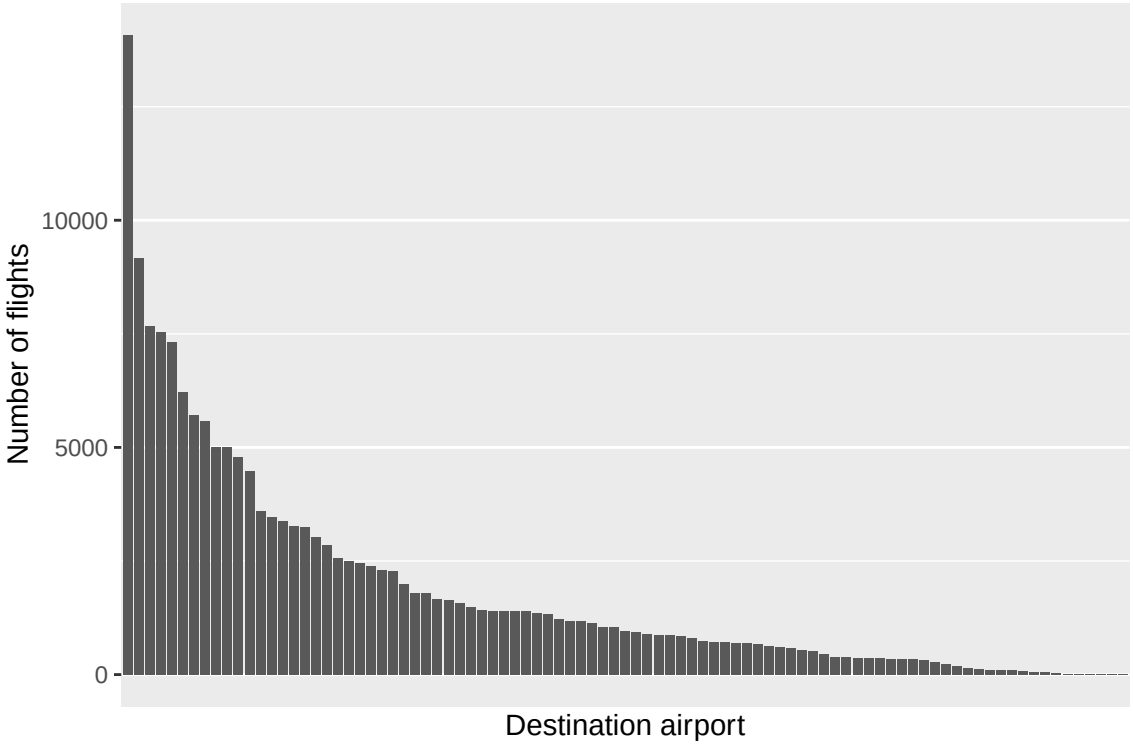
As mentioned, the dataset contains 166,750 flights, operated by 9 different carriers. The two largest carriers account for over half of all flights, with the next four carriers operating another 40%.



US domestic flights from SFO are operated throughout the year, with each week accounting for around 3,000 flights. Peaks in the above chart clearly show summer and Christmas/new year's holidays.



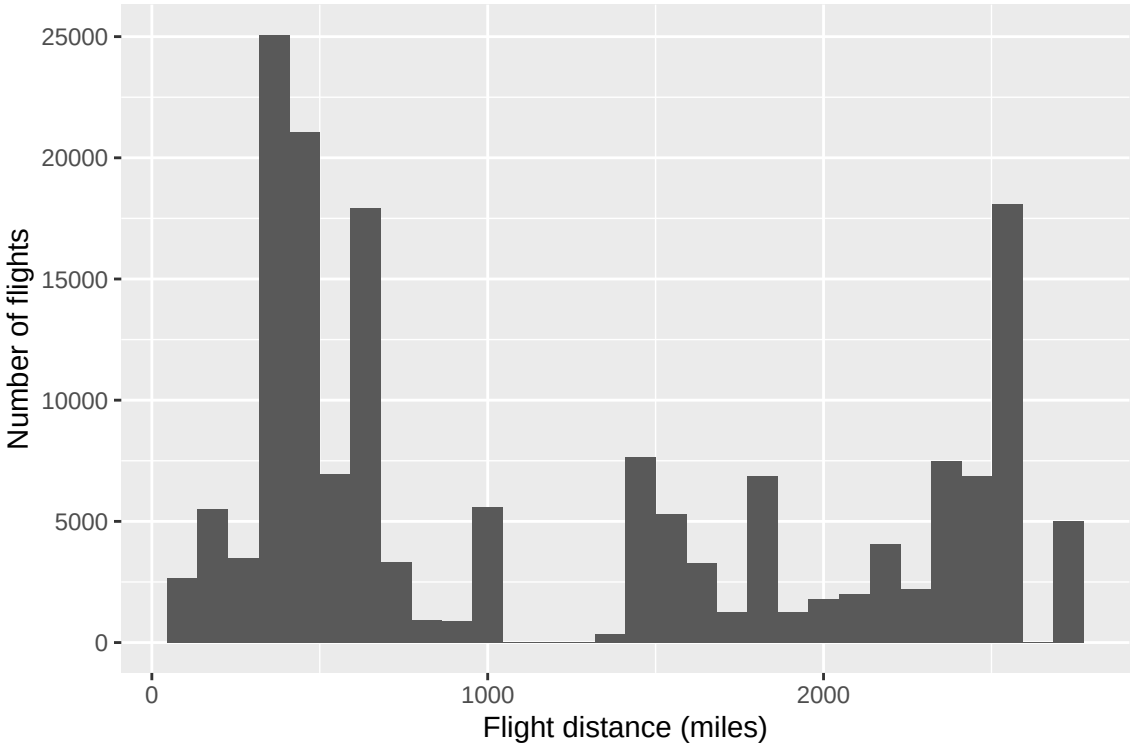
The distribution of flights over the day (planned departure time in local timezone, rounded to the nearest full hour) shows that most flights are operated in the morning (between 6 and 11 AM). Furthermore, the airport is (nearly) idle in the night between 2 to 5 AM.



From SFO flights are operated to 91 different destinations, although the histogram above shows that the majority of flights is only to a dozen or so airports. The top 10 destinations, accounting for about 45% of all flights, are the following:

Dest	Number	Share (%)
LAX	14086	8.4
SEA	9171	5.5
LAS	7677	4.6
JFK	7541	4.5
SAN	7325	4.4
ORD	6221	3.7
EWR	5715	3.4
DEN	5581	3.3
BOS	5013	3.0
PDX	5008	3.0

Not surprisingly, Los Angeles (LAX) is the top destination from San Francisco, with more than 8% of flights, followed at a distance by Seattle (SEA), Las Vegas (LAS), New York (JFK) and San Diego (SAN).



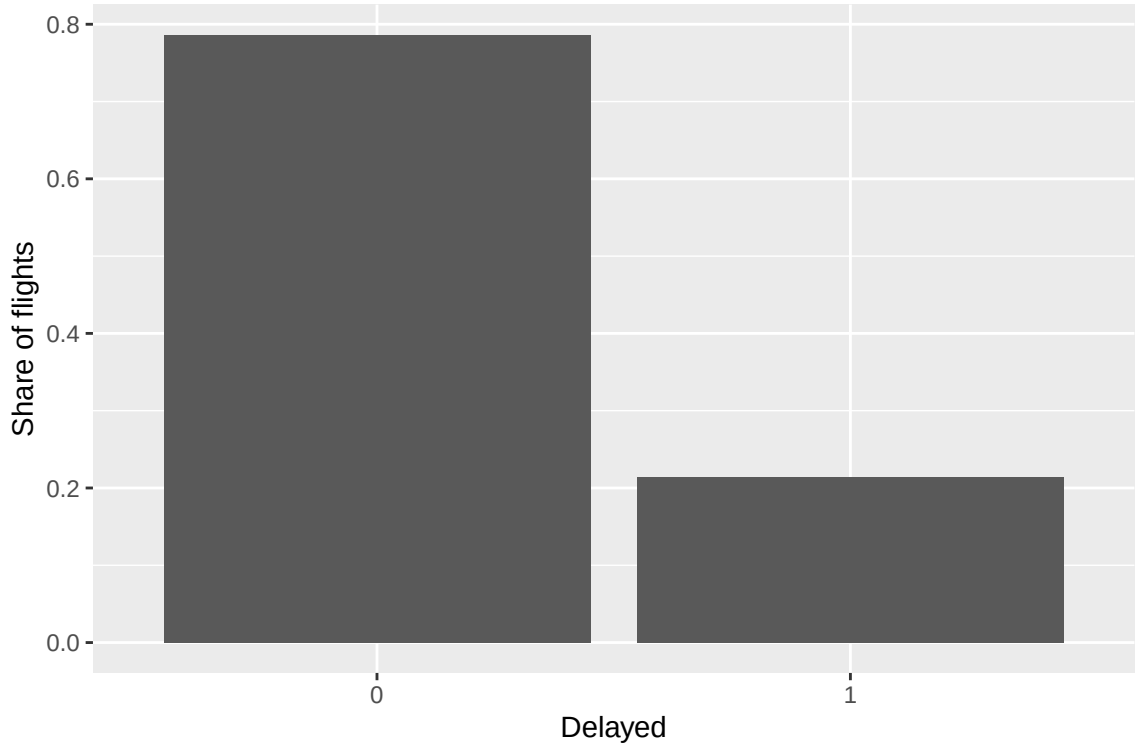
Flight distance shows a rather peculiar distribution, with large peaks around 500 miles, and (almost) no flights of 1,000 to 1,500 miles. Let's look at the cutoff points for 50% and 80% of the flights.

```
quantile(flights_sfo$Distance, c(0.51, 0.8))
```

```
##  51%  80%
##  751 2398
```

A majority of flights has a distance of less than 750 miles, while 80% of flights has a distance below 2,400 miles.

2.4.2 Delay distribution The second part of the EDA is focused on understanding the occurance and distribution of delays. Again we will look into who (carriers), when (dates/times), and where (destinations), but we start by how often delays occur.



Based on the delayed flag we created in the dataset, we see that slightly more than 20% of flights is delayed by more than 15 minutes at arrival (our definition of delayed). To be precise, the percentage delayed flights is:

```
# Overall pdelayed
flights_sfo %>%
    summarize(probability = mean(as.numeric(flights_sfo$Delayed) -
        1))
```

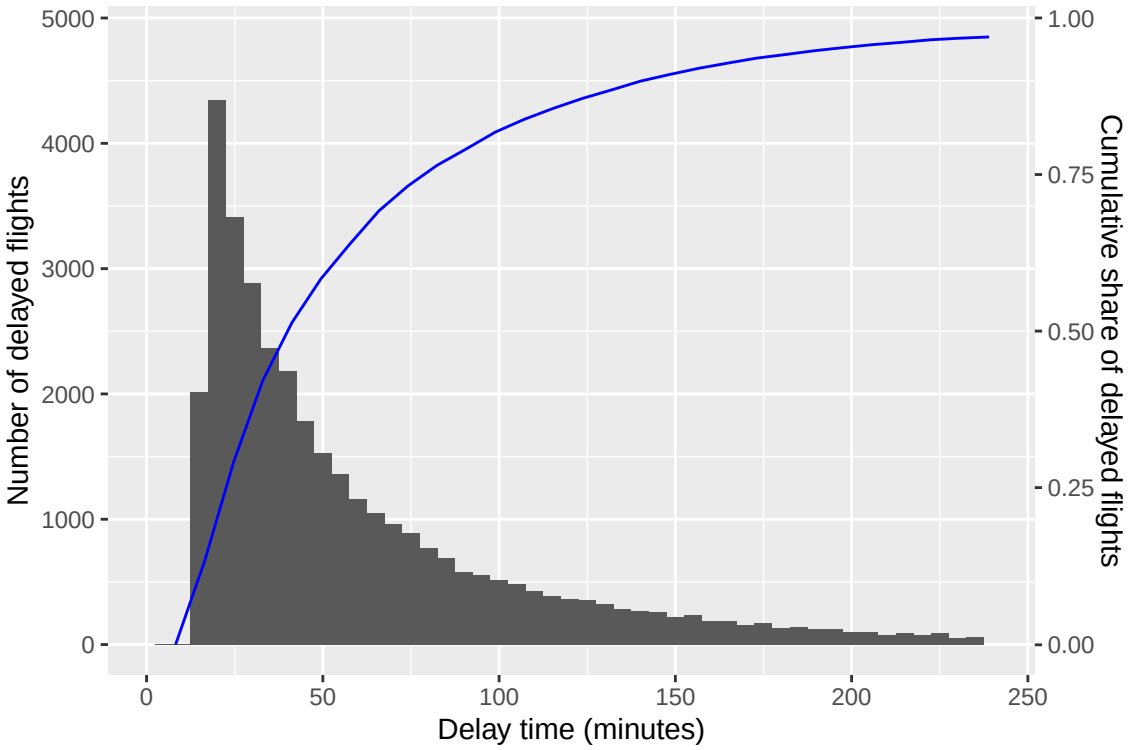
```
##   probability
## 1       0.214
```

The overall average and median delay duration (in minutes) of all delayed flights is as follows:

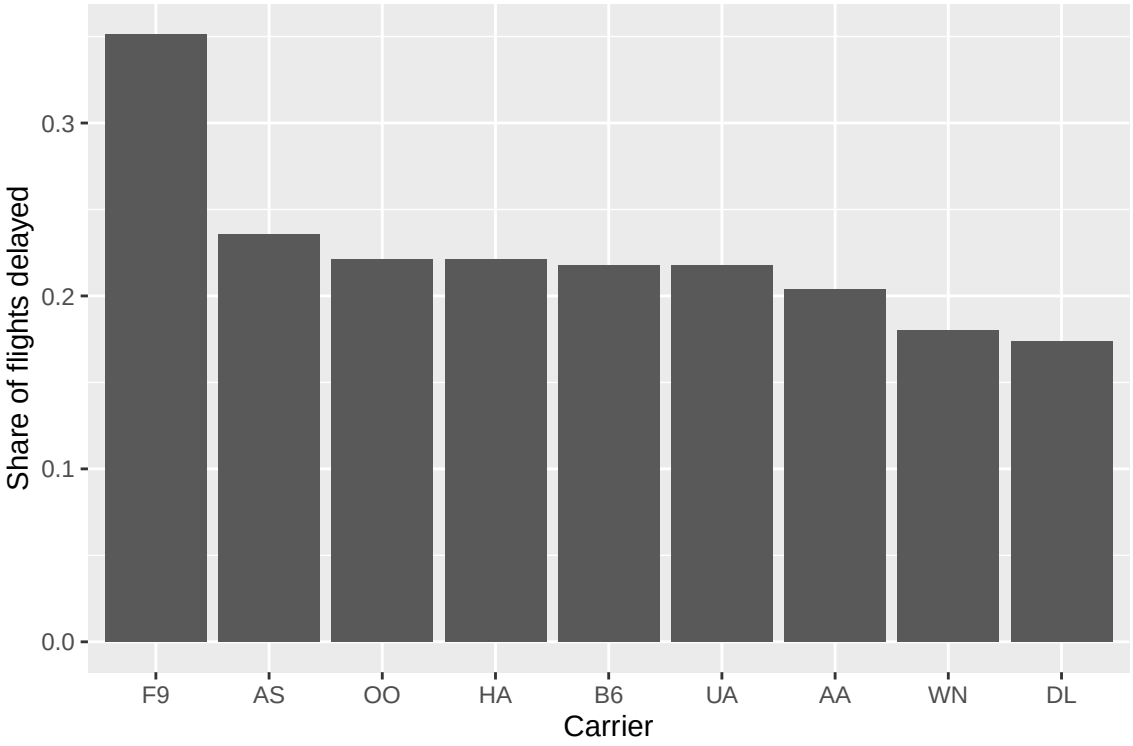
```
# Overall mean and median delay duration
flights_sfo %>%
    filter(Delayed == 1) %>%
    select(ArrDelay) %>%
    summarize(mean = mean(ArrDelay), median = median(ArrDelay))
```

```
##   mean median
## 1 70.1     44
```

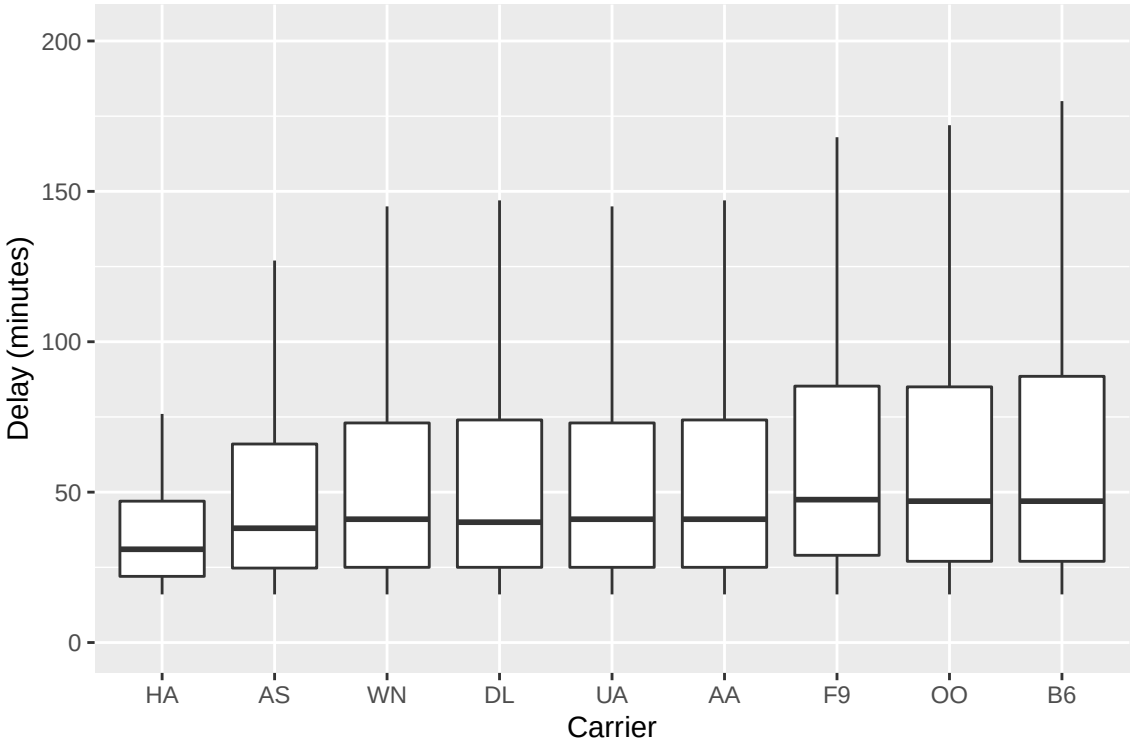
The large difference between both indicates that there are some outliers with a very long delay, which pull up the mean delay to 70 minutes. The majority of delayed flights has a delay of less than one hour, resulting in a median of 44 minutes.



When we look at the above chart, we can confirm that the majority of flights (around 60%) is delayed less than one hour. The chart shows all delayed flights (cut-off at 240 minutes for readability) in 5-minute bins. The blue line shows the cumulative percentage of flights in the histogram. Only a tiny portion of flights is delayed by more than 4 hours.

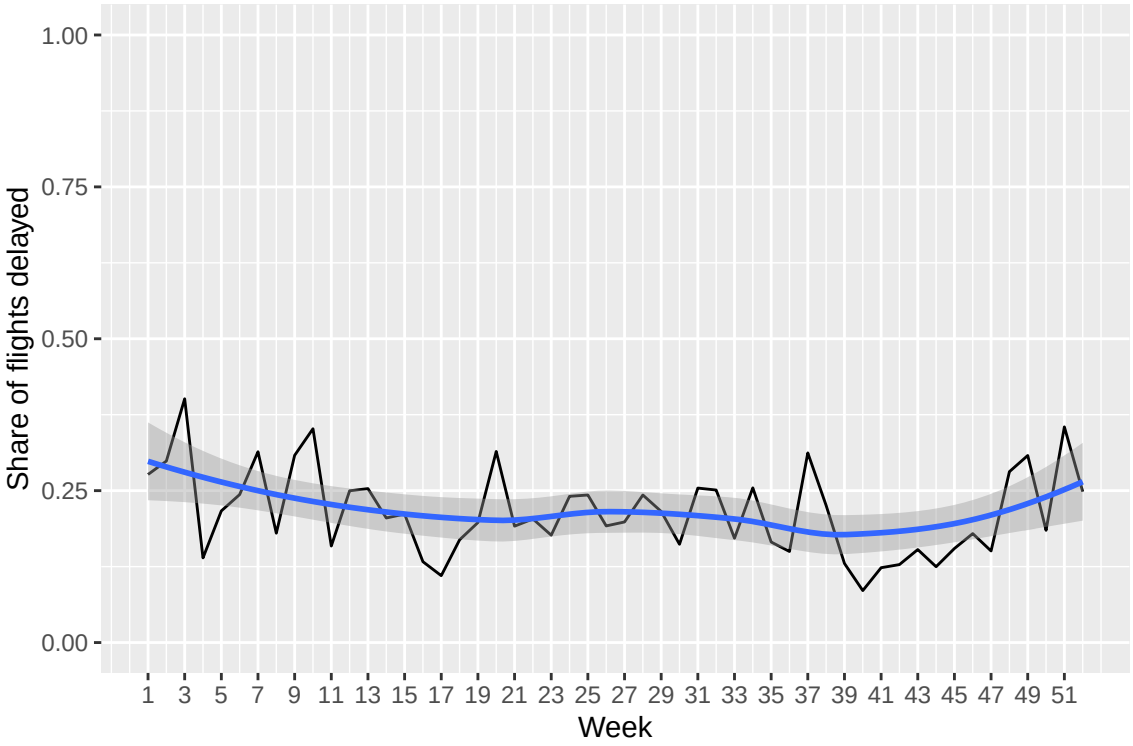


As for the share of flights delayed by carrier, only carrier F9 (Frontier Airlines, Inc.) stands out with 35%. But this carrier only operates a small number of flights, accounting for less than 1% of domestic flights from SFO. Most other carriers have a delay percentage of ~17% to 23% of flights, very near the overall delay percentage of 21%.



Looking at delay duration, there are some differences between carriers, as the above boxplot shows (cut-off at p1 and p99 for readability). Carrier HA (Hawaiian Airlines) has the lowest median delay and also a small spread between p25 and p75, indicating shorter and more consistent delays. The previous plot, however, showed that HA does not have substantially less delayed flights than other carriers.

Carriers F9 (Frontier Airlines, Inc.), OO (SkyWest Airlines), and B6 (JetBlue Airways Corporation) stand out with slightly higher median delays of nearly 50 minutes (out of all delayed flights) and also higher p75 limits. The other carriers have a median delay of close to the overall median delay of 44 minutes.

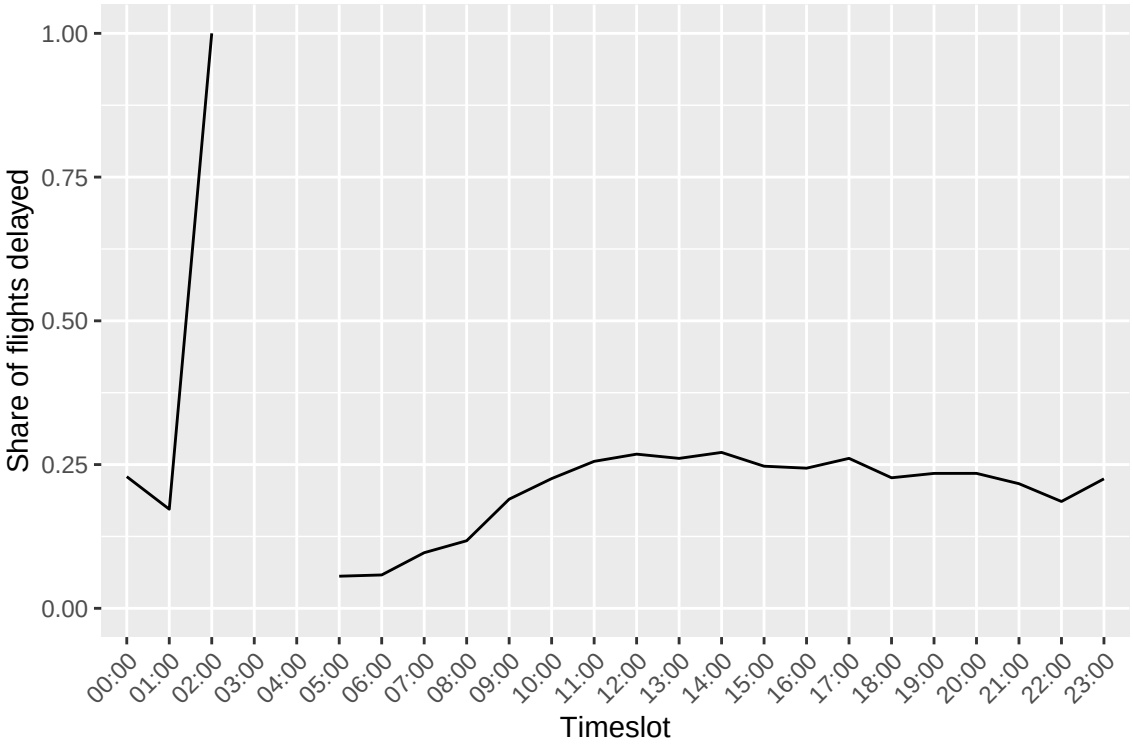


When we plot the percentage of flights delayed throughout the year (by week number), a certain seasonality effect is visible from the smoothing line added using the `loess` method. Delay is lowest in spring and autumn and higher in summer and (more pronounced) winter. This corresponds with the higher number of flights in summer and winter, as seen in chapter 2.4.1, which puts forward the question whether delays and number of flights are correlated. Let's find out.

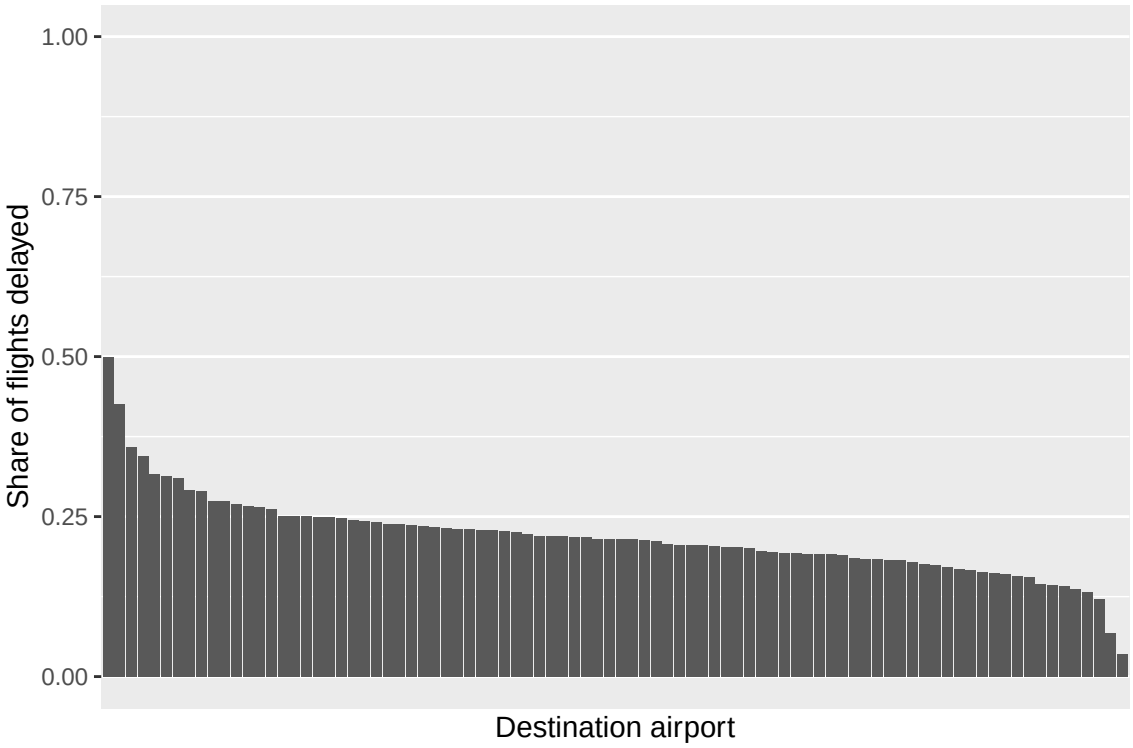
```
# Calculate correlation coefficient for delay probability
# and number of flights
cor.test(num_flights_pdelayed_by_week$pDelayed, num_flights_pdelayed_by_week$n)
```

```
##
##  Pearson's product-moment correlation
##
## data:  num_flights_pdelayed_by_week$pDelayed and num_flights_pdelayed_by_week$n
## t = -2, df = 50, p-value = 0.07
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.4940  0.0187
## sample estimates:
##    cor
## -0.256
```

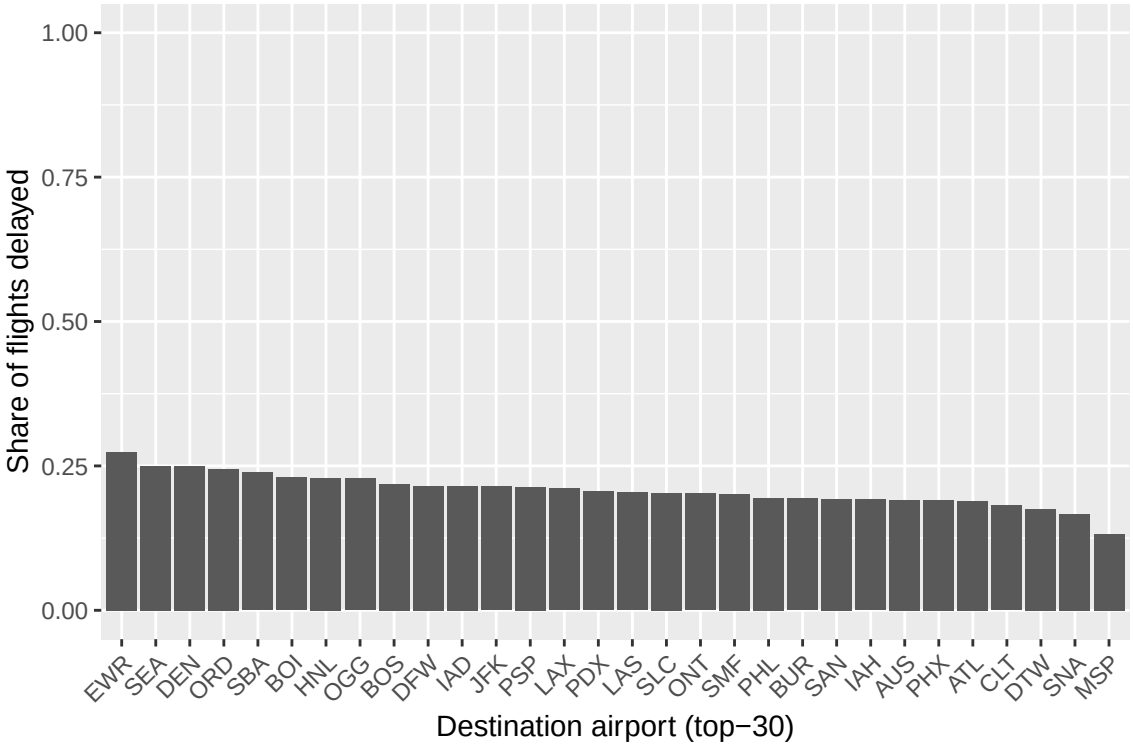
The correlation coefficient is -0.25, which actually indicates a negative correlation (less flights means higher probability for delay), but the Pearson's test shows that the relationship is not statistically significant (p-value 0.0675). Indeed the confidence interval includes a correlation coefficient of 0 (no relationship at all). This means that for the machine learning model, we need to use other predictors than the number of flights to predict flight delays.



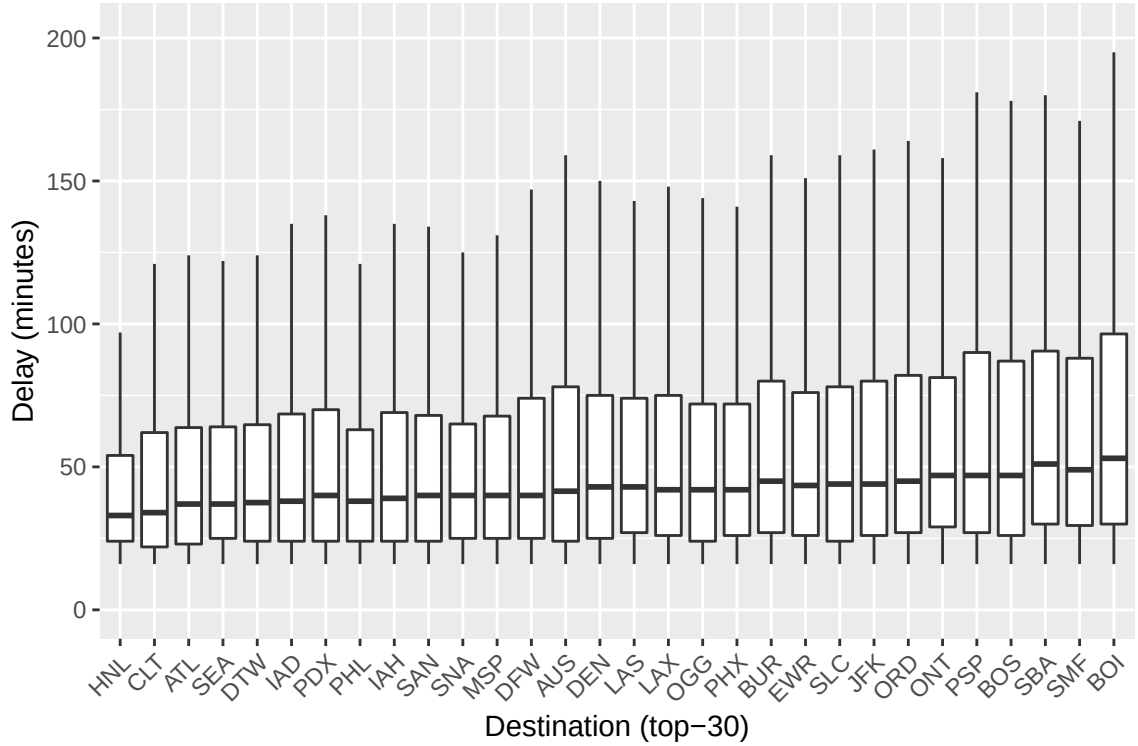
Looking at the probability of a delay at different timeslots during the day, a clear distinction shows between early-morning flights (5 AM to 8 AM) and flights during the day (9 AM to 12 AM). This could be due to a lower number of flights (although the previous analysis showed no significant relationship between number of flights and share of flights delayed). But it could also be that early morning flights don't often suffer from operational disruptions because they are at the start of the day. Note that there is no data for 3 AM and 4 AM as there are no flight departures then. Also, the peak of 100% delayed at 2 AM is an outlier due to the fact that there was only 1 flight recorded at 2 AM, which was in fact delayed.



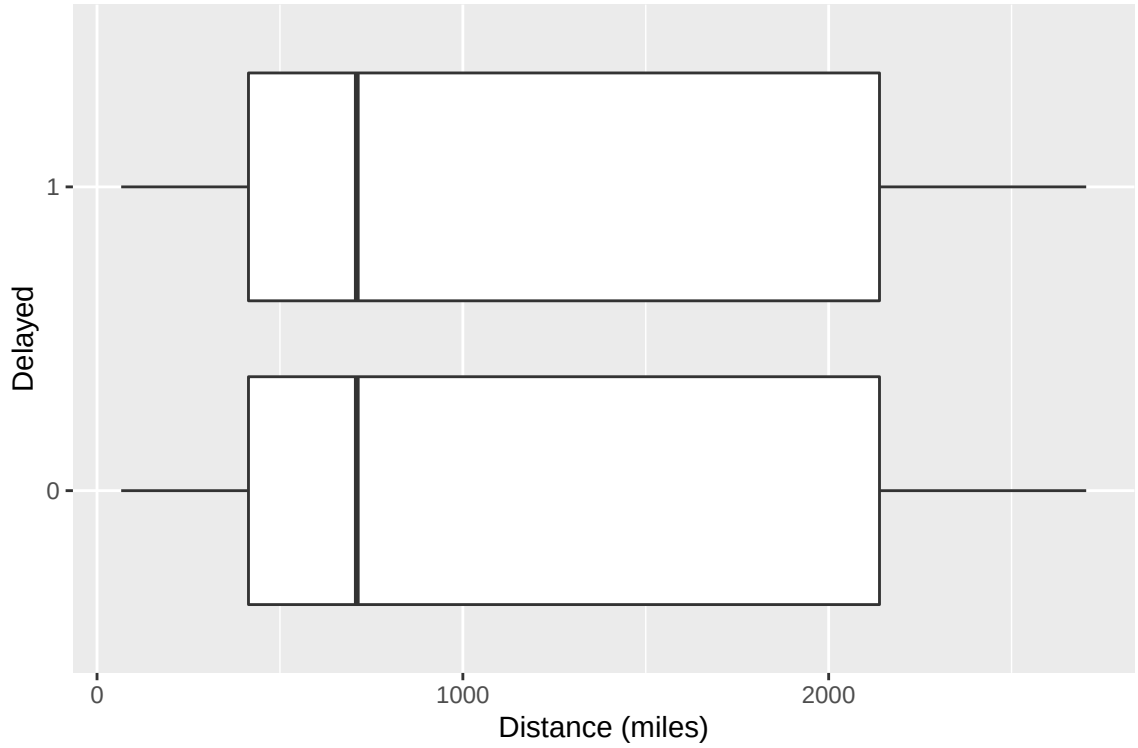
When taking a look at delay probability by destination airport, we can see some outliers on both ends. But the vast majority of destinations shows a delay probability between 10% and 20%. Off course this chart also includes many destinations with only small numbers of flights, which might not impact the overall delay prediction model nor the average traveler much. Let's recreate this chart with only the top 30 destinations.



Still the top 30 destinations (accounting for about 77% of all flights from San Francisco) shows some difference in delay probability. Several destinations are above or close to 25% of flights delayed, while there are five destinations with delay probabilities below 19%. Let's see if there are any substantial differences in delay duration.



As the above boxplot shows, the distribution of delays for the top 30 destinations looks rather similar with only small differences in median and quartile limits. Only a few destinations stand out: HNL (Honolulu, Hawaii) for the lowest median delay and also a low p75 value. This corresponds to the boxplot we saw earlier, showing lower median and p75-values for Hawaiian Airlines, which is probably the main carrier to operate the SFO-HNL route. Additionally, BOI (Boise, Idaho), ONT (Ontario, California), and SBA (Santa Barbara, California) for high medians and p75 values. But even these destinations have a median delay that's only a few minutes above the overall median delay.



Looking at flight distance, finally, a boxplot of distance for flights grouped into delayed (1) and not-delayed (0) shows almost perfectly consistent distribution. This implies there is no correlation between delay and flight distance. Let's confirm using the actual delay time at arrival (ArrDelay):

```
# Calculate correlation coefficient for delay and distance
cor.test(flights_sfo$ArrDelay, flights_sfo$Distance)
```

```
##
##  Pearson's product-moment correlation
##
## data:  flights_sfo$ArrDelay and flights_sfo$Distance
## t = -2, df = 166748, p-value = 0.02
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.01073 -0.00114
## sample estimates:
##      cor
## -0.00594
```

Indeed, as the coefficient of -0.006 shows, flight distance and delay are not correlated.

2.4.3 EDA conclusions Finalizing the exploratory data analysis, let's try and draw some conclusions from the analysis performed.

- There are observable differences between carriers, both in number of flights, in the probability of delays, and delay duration. This means carrier might be a good predictor to use in the machine learning model we'll build.

- The same holds for the timing of the flight. The week number seems to be related to the probability of a delay, representing both peak moments (e.g. holidays) and overall weather conditions changing throughout the year. The timeslot during the day also seems to be important, as there are notable differences in delay probability. Again, this means that week number and timeslot might be good predictors for our model.
- Destination airport also shows differences, although these are smaller and less pronounced. We will need to see whether this data point is indeed a predictor adding predictive value to our model.
- Flight distance shows no relation to delay class or actual delay time at arrival. This means that distance is not a good predictor for our model.

2.5 Split dataset

Having performed initial analysis on the overall set of flights from SFO (EDA), the next step is to prepare the different datasets for use in the machine learning algorithms. But before actually splitting into training and test sets, we need to take into account that the dataset is still rather large (166,750 records), and that it is imbalanced.

First, we deal with the dataset size. Because the current datasets would require too much computing resources and thus time to train models, we will use a randomly selected 50% sample of the total set of flights departing from SFO as the base set for building training and test sets. For that we can use the `createDataPartition()` function:

```
# Create random 50% subset of data for further analysis
set.seed(1, sample.kind = "Rounding")
use_flights_index <- createDataPartition(flights_sfo$Delayed,
    times = 1, p = 0.5, list = FALSE)
use_flights <- flights_sfo[use_flights_index, ]
```

Next the imbalance. As we saw in chapter 2.4, there are roughly four flights on time for every delayed flight (ratio 1:4). That means that it is harder to train algorithms in how to predict delayed flights, and also that algorithms can obtain around 80% accuracy by just predicting that no flights are delayed (although the sensitivity will be quite low then). This could lead to low performing algorithms, that will not be of much use in practical applications.

There are a number of ways to deal with imbalanced data, as discussed by several authors on machine learning (notes 3,4). For this project I will use two strategies. First, I will try different machine learning models, as not every model can handle imbalanced data well. Second, I will manually resample the training set to include the more delayed flights (by undersampling non-delayed flights).

This means that we need to create three datasets:

- A 70% sample of half of SFO flights to use as imbalanced training set (in effect 35% of all flights from SFO)
- A 30% sample of half of SFO flights to use as test set (in effect 15% of all flights from SFO)
- A resampled training set that includes a 50%/50% ratio of delayed an not-delayed flights (in effect 35% of all flights from SFO)

This is the code to prepare the datasets:

```
# Calculate how many rows we need to have of both delayed
# and not-delayed flights
nrow_parts <- ceiling(nrow(train)/2)
```

```
# Separate delayed and not-delayed flights in the SFO set
flights_sfo_delayed <- flights_sfo %>%
    filter(Delayed == 1)
flights_sfo_notdelayed <- flights_sfo %>%
    filter(Delayed == 0)

# Calculate sampling probability for not-delayed flights
p_forbalance <- nrow_parts/nrow(flights_sfo_notdelayed)

# Take random sample of not-delayed flights equal to 50% of
# the target train set size
set.seed(1, sample.kind = "Rounding")
use_flights_index <- createDataPartition(flights_sfo_notdelayed$Delayed,
    times = 1, p = p_forbalance, list = FALSE)
use_flights_notdelayed <- flights_sfo_notdelayed[use_flights_index,
    ]

# Calculate sampling probability for delayed flights
p_forbalance <- nrow_parts/nrow(flights_sfo_delayed)

# Take random sample of delayed flights equal to 50% of the
# target train set size
set.seed(1, sample.kind = "Rounding")
use_flights_index <- createDataPartition(flights_sfo_delayed$Delayed,
    times = 1, p = p_forbalance, list = FALSE)
use_flights_delayed <- flights_sfo_delayed[use_flights_index,
    ]

# Combine equal parts delayed and non-delayed flights into
# one balanced set again
train_bal <- rbind(use_flights_notdelayed, use_flights_delayed)
```

After running this code, we should end up with two equally-sized training sets and one smaller test set. Let's check:

```
# Show dimensions for created datasets
dim(train)
```

```
## [1] 58363    22
```

```
dim(train_bal)
```

```
## [1] 58365    22
```

```
dim(test)
```

```
## [1] 25013    22
```

And, let's check the ratio of delayed vs. non-delayed flights in both training sets:

```r
# Show percentage of delayed flights in each training set
mean(train$Delayed == 1)
```

```
## [1] 0.214
```

```r
mean(train_bal$Delayed == 1)
```

```
## [1] 0.5
```

2.6 Train algorithms

After preparing the data, we can start to train machine learning models and evaluate their predictive power. In this process several algorithms will be trained to maximize model performance.

Model 1: logistic regression on delayed/not-delayed classification The first model that we'll train is a logistic regression model that predicts the probability of a flight being either delayed or not delayed. It is a model that is often used for binary classification problems like this one. We'll train the model on both the imbalanced and balanced datasets using the `glm` function:

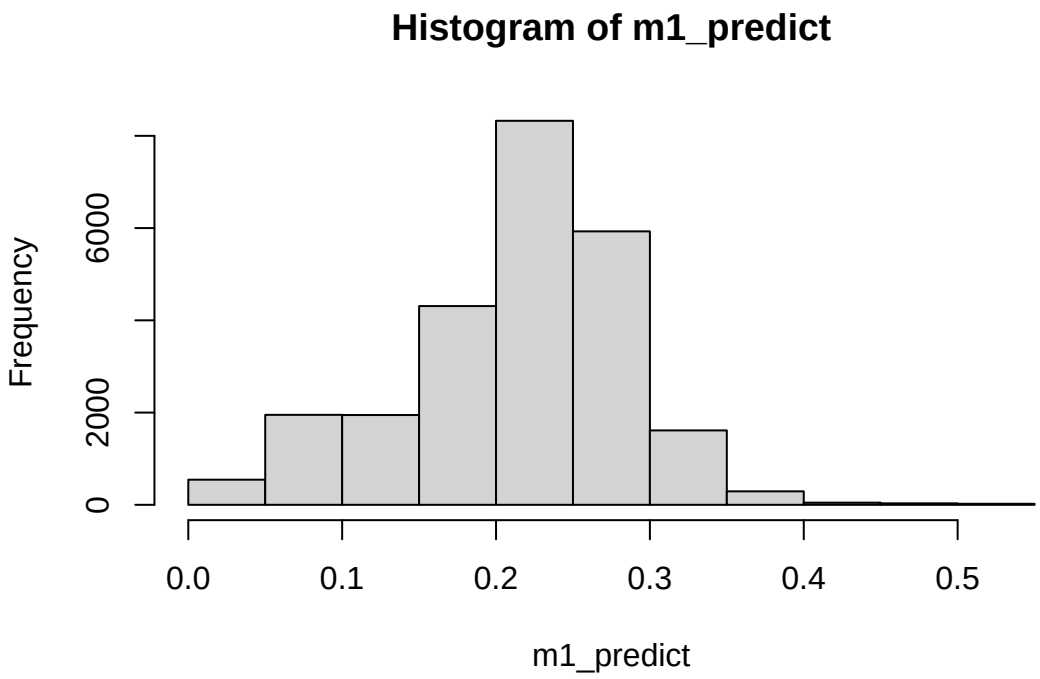
```r
## M1 - Logistic Regression model on delay yes/no

# Fit models
m1_fit <- glm(Delayed ~ DepWeek + UniqueCarrier + Dest + DepTimeslot,
    family = binomial(link = "logit"), data = train)
m1_fit_bal <- glm(Delayed ~ DepWeek + UniqueCarrier + Dest +
    DepTimeslot, family = binomial(link = "logit"), data = train_bal)

# Make prediction on test set (delay probability)
m1_predict <- predict(m1_fit, test, type = "response")
m1_predict_bal <- predict(m1_fit_bal, test, type = "response")
```

Having trained the model, let's look at the probability distribution.

```r
# Plot predicted probability
hist(m1_predict)
```



The histogram clearly shows almost no flights are predicted to have a delay probability of 0.5 or more. This is due to the imbalance in the data. This means we should not use a cut-off point of 0.5 to predict the class, but rather use the overall probability that a flight is delayed:

```
# Calculate overall pdelayed in imbalanced dataset
overall_delay_pct <- mean(train$Delayed == 1)
overall_delay_pct
```

```
## [1] 0.214
```

When we transfer probabilities to actual class predictions, we can examine the results in a so-called confusion matrix.

```
# Transfer predicted probability into delay flag (yes/no)
m1_predict_class <- as.factor(ifelse(m1_predict > overall_delay_pct,
    1, 0))  # Using overall delay probability
# Show model performance
confusionMatrix(data = m1_predict_class, reference = test$Delayed,
    mode = "prec_recall", positive = "1")
```

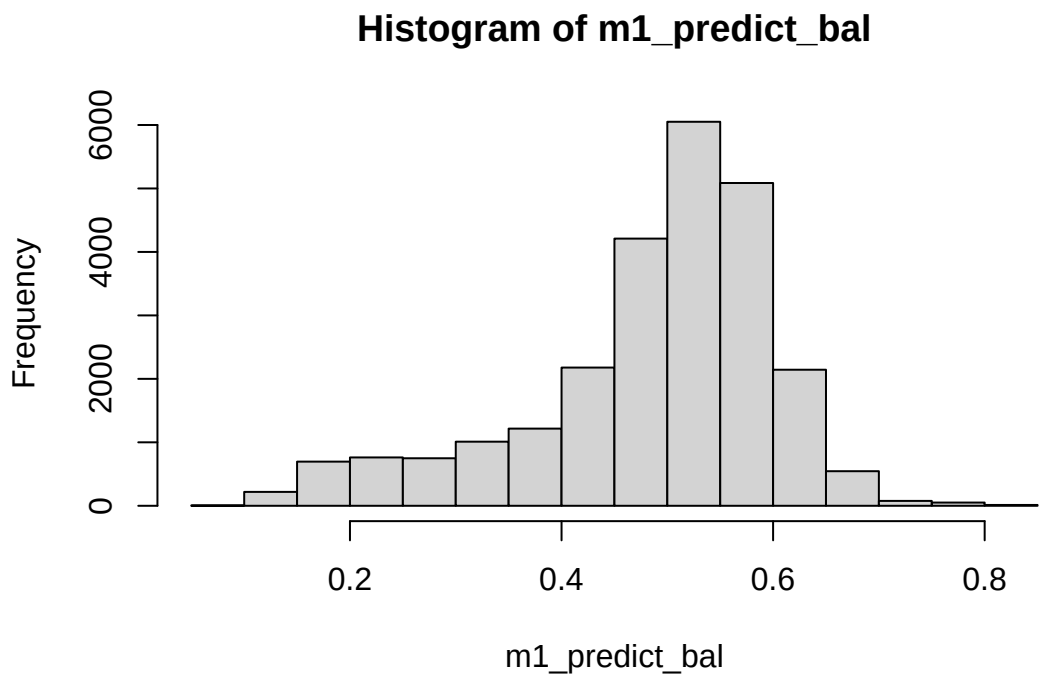
```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction     0     1
##          0  9171  1664
##          1 10492  3686
##
##                Accuracy : 0.514
```

```
##                  95% CI : (0.508, 0.52)
##     No Information Rate : 0.786
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.097
##
##  Mcnemar's Test P-Value : <2e-16
##
##               Precision : 0.260
##                  Recall : 0.689
##                      F1 : 0.378
##              Prevalence : 0.214
##          Detection Rate : 0.147
##    Detection Prevalence : 0.567
##       Balanced Accuracy : 0.578
##
##        'Positive' Class : 1
##
```

As the confusion matrix shows, the model predicts nearly the same amount of delays as no-delays. But it is quite wrong in predicting delays: of the 14,187 delays predicted, only 3,686 flights actually were delayed. This leads to a precision of 0.26. Recall, the amount of no-delays predicted correctly, actually scores quite a bit better, being right in almost 70% of cases. The balanced performance metric F1 for this model is 0.3775.

Let's try to see if we can achieve a better performance result. The next step is to look at the probability distribution from training on the balanced dataset.

```
# Plot predicted probability
hist(m1_predict_bal)
```



The histogram already shows a large change: a lot more flights are now predicted to have a delay probability over 0.5. This is because we manually biased the balanced training set to include the same amount of delayed and not-delayed flights. Let's look at the prediction performance when we transform probabilities into class predictions using the 0.5 probability cut-off point (since this dataset is balanced):

```
# Transfer predicted probability into delay flag (yes/no)
# based on the overall probability of delay
m1_predict_class_bal <- as.factor(ifelse(m1_predict_bal > 0.5,
    1, 0))  # Using 0.5 probability because set is balanced with both classes 50%

# Show model performance
confusionMatrix(data = m1_predict_class_bal, reference = test$Delayed,
    mode = "prec_recall", positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0  9323  1723
##          1 10340  3627
##
##                Accuracy : 0.518
##                  95% CI : (0.512, 0.524)
##     No Information Rate : 0.786
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.096
##
##  Mcnemar's Test P-Value : <2e-16
##
##               Precision : 0.260
##                  Recall : 0.678
##                      F1 : 0.376
##              Prevalence : 0.214
##          Detection Rate : 0.145
##    Detection Prevalence : 0.558
##       Balanced Accuracy : 0.576
##
##        'Positive' Class : 1
##
```

The confusion matrix from this set actually shows very similar results to the previous one. The resulting F1 score is 3.755, which is also very close. For this model, it seems that the balanced dataset does not give more predictive power by including more cases of delayed flights.

As a next step, let's see if training a different algorithm yields better performance.

Model 2: classification tree on delayed/not-delayed classification Besides logistic regression, classification trees are often used for binary or multiclass classification problems. Classification trees are also known to better handle imbalanced data. Let's train models on both the imbalanced and balanced datasets. We will use the `rpart` model that is included in the caret package using the `train` function, and set it to cross-validate the model in 10 iterations:

```
# Fit model on imbalanced data
m2_fit <- train(Delayed ~ DepWeek + UniqueCarrier + Dest + DepTimeslot,
    method = "rpart", data = train, trControl = trainControl(method = "cv",
        number = 10))
m2_fit_bal <- train(Delayed ~ DepWeek + UniqueCarrier + Dest +
    DepTimeslot, method = "rpart", data = train_bal, trControl = trainControl(method = "cv",
    number = 10))

# Make prediction on test set
m2_predict <- predict(m2_fit, test, type = "prob")
m2_predict_bal <- predict(m2_fit_bal, test, type = "prob")

# Transfer predicted probability into delay flag (yes/no)
m2_predict_class <- as.factor(ifelse(m2_predict$`1` > overall_delay_pct,
    1, 0))
# Using overall delay probability
m2_predict_class_bal <- as.factor(ifelse(m2_predict_bal$`1` >
    0.5, 1, 0))
# Using 0.5 probability because set is balanced with both
# classes 50%
```

Let's now look at the confusion matrix for model that is trained on the imbalanced dataset:

```
# Show model performance
confusionMatrix(m2_predict_class, test$Delayed, mode = "prec_recall",
    positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 17002  4199
##          1  2661  1151
##
##                Accuracy : 0.726
##                  95% CI : (0.72, 0.731)
##     No Information Rate : 0.786
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.089
##
##  Mcnemar's Test P-Value : <2e-16
##
##               Precision : 0.302
##                  Recall : 0.215
##                      F1 : 0.251
##              Prevalence : 0.214
##          Detection Rate : 0.046
##    Detection Prevalence : 0.152
##       Balanced Accuracy : 0.540
##
##        'Positive' Class : 1
##
```

As we can see this model predicts a 'no delay' for most observations, indeed many more than model 1 did. It actually predicts no-delay quite well, but precision and recall are quite low and the F1 score is lower than those from model 1.

Next the results for the model trained on the balanced dataset. Here the confusion matrix looks different:

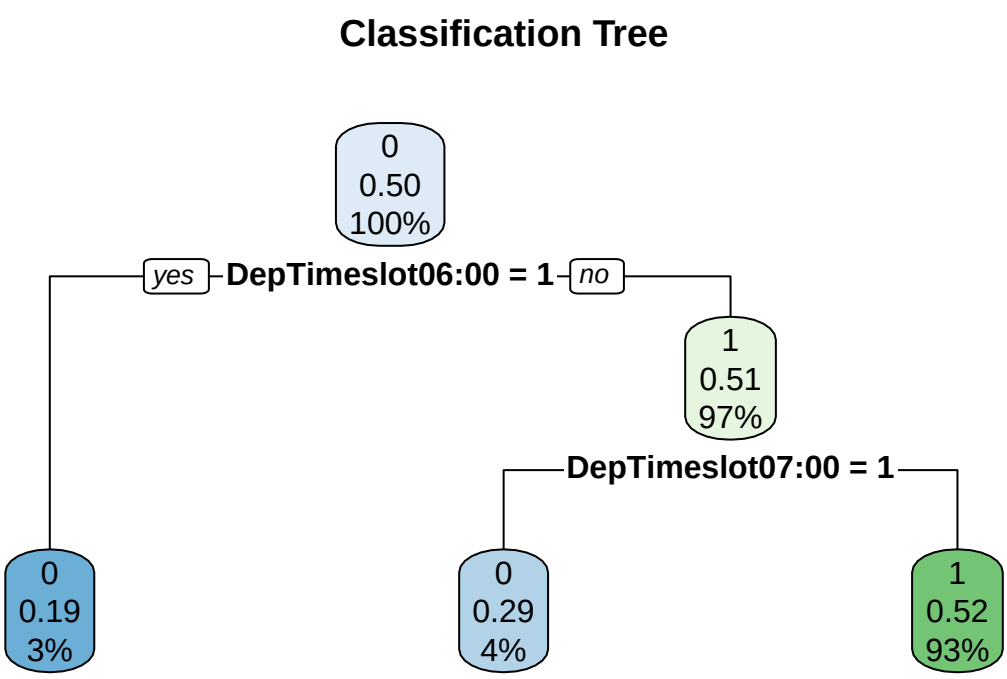
```
# Show model performance
confusionMatrix(m2_predict_class_bal, test$Delayed, mode = "prec_recall",
    positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0  2149   179
##          1 17514  5171
##
##                Accuracy : 0.293
##                  95% CI : (0.287, 0.298)
##     No Information Rate : 0.786
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.035
##
##  Mcnemar's Test P-Value : <2e-16
##
##               Precision : 0.228
##                  Recall : 0.967
##                      F1 : 0.369
##              Prevalence : 0.214
##          Detection Rate : 0.207
##    Detection Prevalence : 0.907
##       Balanced Accuracy : 0.538
##
##        'Positive' Class : 1
##
```

The model predicts a delay in most cases now. Behind the F1 score of 0.3689 are a poor 0.29 accuracy and a very high 0.97 recall. This means the model is very poor in correctly predicting a delay, but if it predicts a no-delay, you can be quite certain of this. This is a start, but as a comprehensive model still not satisfying.

The advantage of a decision tree model is that we can actually see the tree that was created in the model fit, to more closely examine the cutoffs the model uses:

```
# Plot resulting tree
rpart.plot(m2_fit_bal$finalModel, uniform = TRUE, main = "Classification Tree")
```



As we can see, the tree is rather simple. Basically it says: if the flight was scheduled to depart at 6 AM, then it was not delayed (left top branch). If it was not scheduled for 6 AM (right top branch), but instead scheduled for 7 AM (lower branch), then it was not delayed either. In all other cases, it was delayed. This obviously is a simplification of reality, which is much more dynamic than the tree suggests. But it does match the conclusions we drew from the exploratory data analysis earlier.

Model 3: random forest on delayed/not-delayed classification We will have another try to find a model that better predicts if a flight is delayed or not. The caret package we use also supports random forests. This model follows the logic of a classification tree, but then develops many different trees (a forest) and averages the predictions made by these trees. This should lead to more accurate predictions, although the downside is that it no longer produces an easy to understand decision tree.

Let's train a model on both the datasets again. This is the code to fit the model:

```
# Fit models
m3_fit <- train(Delayed ~ DepWeek + UniqueCarrier + Dest + DepTimeslot,
    method = "rf", data = train, tuneGrid = data.frame(mtry = 2))
m3_fit_bal <- train(Delayed ~ DepWeek + UniqueCarrier + Dest +
    DepTimeslot, method = "rf", data = train_bal, tuneGrid = data.frame(mtry = 2))

# Make prediction on test set
m3_predict <- predict(m3_fit, test, type = "prob")
m3_predict_bal <- predict(m3_fit_bal, test, type = "prob")

# Transfer predicted probability into delay flag (yes/no)
m3_predict_class <- as.factor(ifelse(m3_predict$`1` > overall_delay_pct,
    1, 0))  # Using overall delay probability
m3_predict_class_bal <- as.factor(ifelse(m3_predict_bal$`1` >
    0.5, 1, 0))  # Using 0.5 probability because set is balanced with both classes 50%
```

Now let's show the confusion matrix

```
# Show model performance
confusionMatrix(m3_predict_class, test$Delayed, mode = "prec_recall",
    positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 19663  5350
##          1     0     0
##
##                Accuracy : 0.786
##                  95% CI : (0.781, 0.791)
##     No Information Rate : 0.786
##     P-Value [Acc > NIR] : 0.504
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##               Precision :    NA
##                  Recall : 0.000
##                      F1 :    NA
##              Prevalence : 0.214
##          Detection Rate : 0.000
##    Detection Prevalence : 0.000
##       Balanced Accuracy : 0.500
##
##        'Positive' Class : 1
##
```

As the matrix shows, this model uses an extremely oversimplified view of reality by just predicting none of the flights is delayed. This yields a high accuracy because only about 21% of flights actually is delayed. As recall is 0, there is no F1 score. Overall, this model is clearly useless for our purpose. A quick check of the model trained on the balanced dataset shows the same results:

```
# Show confusion matrix
table(m3_predict_class_bal, test$Delayed)
```

```
##
## m3_predict_class_bal     0     1
##                    0 19663  5350
##                    1     0     0
```

The random forest model does offer some more options (parameters) to tune, which might lead to a better outcome. But given the large computational effort (it took a full day to train these models) needed to train this algorithm, we will move on and try another model and then another type of prediction.

Model 4: kNN on delayed/not-delayed classification As a final try to train a model that predicts delayed/not-delayed classes, we will use the `kNN` or k-nearest neighbors algorithm. In this algorithm, the

test case (for which an outcome should be predicted) will be placed among 'neighbors' that have similar characteristics. The class membership (in our project either delay or no-delay) of the neighbors determine the predicted class for the test case. In other words: if similar cases are delayed flights, then the prediction will also be 'delay', and the other way around. The number of neighbors to consider ('k') is a parameter between 1 and typically 5~15, which the function `train` will automatically tune to find the optimal value. The code for this model looks like this:

```
# Set parameters and fit model
ctrl <- trainControl(method = "repeatedcv", repeats = 1)
m4_fit <- train(Delayed ~ DepWeek + UniqueCarrier + Dest + DepTimeslot,
    method = "knn", data = train, trControl = ctrl, preProcess = c("center",
        "scale"), tuneLength = 5)
m4_fit_bal <- train(Delayed ~ DepWeek + UniqueCarrier + Dest +
    DepTimeslot, method = "knn", data = train_bal, trControl = ctrl,
    preProcess = c("center", "scale"), tuneLength = 5)

# Make probability prediction on test set
m4_predict <- predict(m4_fit, test, type = "prob")
m4_predict_bal <- predict(m4_fit_bal, test, type = "prob")

# Transform probability prediction to class prediction
m4_predict_class <- as.factor(ifelse(m4_predict$`1` > overall_delay_pct,
    1, 0))  # Using overall delay probability
m4_predict_class_bal <- as.factor(ifelse(m4_predict_bal$`1` >
    0.5, 1, 0))  # Using 0.5 probability because set is balanced with both classes 50%
```

After fitting, we can have a look at the results:

```
# Show model fit
m4_fit
```

```
## k-Nearest Neighbors
##
## 58363 samples
##     4 predictor
##     2 classes: '0', '1'
##
## Pre-processing: centered (118), scaled (118)
## Resampling: Cross-Validated (10 fold, repeated 1 times)
## Summary of sample sizes: 52527, 52527, 52527, 52526, 52527, 52527, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy  Kappa
##    5  0.757     0.0832
##    7  0.766     0.0762
##    9  0.771     0.0668
##   11  0.774     0.0564
##   13  0.776     0.0477
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 13.
```

```
# Show model performance
confusionMatrix(table(m4_predict_class, test$Delayed), mode = "prec_recall",
    positive = "1")
```

```
## Confusion Matrix and Statistics
##
##
## m4_predict_class     0     1
##                0 11012  2009
##                1  8651  3341
##
##                Accuracy : 0.574
##                  95% CI : (0.568, 0.58)
##     No Information Rate : 0.786
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.127
##
##  Mcnemar's Test P-Value : <2e-16
##
##               Precision : 0.279
##                  Recall : 0.624
##                      F1 : 0.385
##              Prevalence : 0.214
##          Detection Rate : 0.134
##    Detection Prevalence : 0.479
##       Balanced Accuracy : 0.592
##
##        'Positive' Class : 1
##
```

First we look at the fit. The model was trained with five different values for k, whereby a value of 13 yielded the best accuracy. The outcomes look slightly better than the previous models. Predictions are substantial for both classes, although delay-predictions are mostly inaccurate. The F1 score is actually slightly better than for previous models. Let's see the results of the model on the balanced dataset.

```
# Show model fit
m4_fit_bal
```

```
## k-Nearest Neighbors
##
## 58365 samples
##     4 predictor
##     2 classes: '0', '1'
##
## Pre-processing: centered (119), scaled (119)
## Resampling: Cross-Validated (10 fold, repeated 1 times)
## Summary of sample sizes: 52529, 52527, 52528, 52528, 52529, 52529, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy  Kappa
##    5  0.588     0.177
```

```
##    7  0.588     0.177
##    9  0.589     0.179
##   11  0.588     0.176
##   13  0.591     0.182
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 13.
```

```
# Show model performance
confusionMatrix(table(m4_predict_class_bal, test$Delayed), mode = "prec_recall",
    positive = "1")
```

```
## Confusion Matrix and Statistics
##
##
## m4_predict_class_bal     0     1
##                    0 12526  1896
##                    1  7137  3454
##
##                Accuracy : 0.639
##                  95% CI : (0.633, 0.645)
##     No Information Rate : 0.786
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.208
##
##  Mcnemar's Test P-Value : <2e-16
##
##               Precision : 0.326
##                  Recall : 0.646
##                      F1 : 0.433
##              Prevalence : 0.214
##          Detection Rate : 0.138
##    Detection Prevalence : 0.423
##       Balanced Accuracy : 0.641
##
##        'Positive' Class : 1
##
```

Again the model worked best with a value of 13 for parameter k. And the results of the model trained on the balanced dataset are better than for the imbalanced set. Distribution of delays over both classes more closely resembles actual distribution, although still not very close and most delay predictions are still inaccurate. The F1 score is again better than the previous model - but overall prediction quality is not great.

Model 5: PDA on delay category We will now look at some models to predict delay category. For that, we use the variable we created and described in chapter 2.3, which classifies each flight into five categories by actual arrival delay time: 0-15 min, 16-30 min, 31-60 min, 61-120 min, >120 min.

The first model to consider is PDA or Penalized Discriminant Analysis, a variant of linear discriminant analysis whereby the model tries to find (linear) combinations of the predictive features for each class (delay category in our case). To train the model, this is the code we use:

```r
# Fit models
control <- trainControl(method = "cv", number = 10, repeats = 2,
    classProbs = TRUE, summaryFunction = multiClassSummary)
m5_fit <- train(DelayCategory ~ DepWeek + UniqueCarrier + Dest +
    DepTimeslot, method = "pda", data = train, trControl = control)
m5_fit_bal <- train(DelayCategory ~ DepWeek + UniqueCarrier +
    Dest + DepTimeslot, method = "pda", data = train_bal, trControl = control)

# Make prediction on test set (delay probability)
m5_predict <- predict(m5_fit, test, type = "raw")
m5_predict_bal <- predict(m5_fit_bal, test, type = "raw")
```

Then, to show the model results:

```r
# Show model performance with explicit levels due to
# missing predictions
confusionMatrix(table(m5_predict, test$DelayCategory), mode = "prec_recall",
    positive = "1")
```

```
## Confusion Matrix and Statistics
##
##
## m5_predict     a     b     c     d     e
##          a 19654  1700  1642  1274   730
##          b     2     0     1     1     0
##          c     0     0     0     0     0
##          d     7     1     1     0     0
##          e     0     0     0     0     0
##
## Overall Statistics
##
##                Accuracy : 0.786
##                  95% CI : (0.781, 0.791)
##     No Information Rate : 0.786
##     P-Value [Acc > NIR] : 0.559
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: a Class: b Class: c Class: d Class: e
## Precision               0.786  0.00000       NA  0.00000       NA
## Recall                  1.000  0.00000   0.0000  0.00000   0.0000
## F1                      0.880      NaN       NA      NaN       NA
## Prevalence              0.786  0.06800   0.0657  0.05097   0.0292
## Detection Rate          0.786  0.00000   0.0000  0.00000   0.0000
## Detection Prevalence    0.999  0.00016   0.0000  0.00036   0.0000
## Balanced Accuracy       0.500  0.49991   0.5000  0.49981   0.5000
```

As we can see, class a (0-15 min delay) is predicted for nearly all cases. By now we know this does not match the dataset and reality very well, which explains why there is not even an F1 score for each class. Let's see if the balanced dataset yields better results:

```
# Show model performance with explicit levels due to
# missing predictions
confusionMatrix(table(m5_predict_bal, test$DelayCategory), mode = "prec_recall",
    positive = "1")
```

```
## Confusion Matrix and Statistics
##
##
## m5_predict_bal     a     b     c     d     e
##              a 19564  1689  1630  1264   727
##              b    15     4     5     1     1
##              c    38     7     6     6     2
##              d    14     1     1     3     0
##              e    32     0     2     1     0
##
## Overall Statistics
##
##                Accuracy : 0.783
##                  95% CI : (0.778, 0.788)
##     No Information Rate : 0.786
##     P-Value [Acc > NIR] : 0.909
##
##                   Kappa : 0.003
##
##  Mcnemar's Test P-Value : <2e-16
##
## Statistics by Class:
##
##                      Class: a Class: b Class: c Class: d Class: e
## Precision               0.787  0.15385  0.10169  0.15789   0.0000
## Recall                  0.995  0.00235  0.00365  0.00235   0.0000
## F1                      0.879  0.00463  0.00705  0.00464      NaN
## Prevalence              0.786  0.06800  0.06573  0.05097   0.0292
## Detection Rate          0.782  0.00016  0.00024  0.00012   0.0000
## Detection Prevalence    0.994  0.00104  0.00236  0.00076   0.0014
## Balanced Accuracy       0.501  0.50070  0.50069  0.50084   0.4993
```

Again, the model predicts almost only no-delays and then obviously does not yield good results.

Model 6: HDDA on delay category Next, we'll try High Dimensional Discriminant Analysis (HDDA), an algorithm optimized for high dimensional data. Given the high number of categorical predictors (that are translated into dummy variables when training a model), our dataset could benefit from this model. This is the code to train:

```
# Fit model on imbalanced training set
m6_fit <- train(DelayCategory ~ DepWeek + UniqueCarrier + Dest +
    DepTimeslot, method = "hdda", data = train, trControl = control)
m6_fit_bal <- train(DelayCategory ~ DepWeek + UniqueCarrier +
    Dest + DepTimeslot, method = "hdda", data = train_bal, trControl = control)

# Make prediction on test set (delay probability)
m6_predict <- predict(m6_fit, test, type = "raw")
m6_predict_bal <- predict(m6_fit_bal, test, type = "raw")
```

Let's see the results:

```
# Show model performance with explicit levels due to
# missing predictions
confusionMatrix(table(m6_predict, test$DelayCategory), mode = "prec_recall",
    positive = "1")
```

```
## Confusion Matrix and Statistics
##
##
## m6_predict     a     b     c     d     e
##          a 18219  1558  1487  1149   639
##          b    51     6     6     0     3
##          c   285    39    34    20     7
##          d    93    11    13     8     7
##          e  1015    87   104    98    74
##
## Overall Statistics
##
##                Accuracy : 0.733
##                  95% CI : (0.728, 0.739)
##     No Information Rate : 0.786
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.021
##
##  Mcnemar's Test P-Value : <2e-16
##
## Statistics by Class:
##
##                      Class: a Class: b Class: c Class: d Class: e
## Precision               0.790  0.09091  0.08831  0.06061  0.05370
## Recall                  0.927  0.00353  0.02068  0.00627  0.10137
## F1                      0.853  0.00679  0.03351  0.01137  0.07021
## Prevalence              0.786  0.06800  0.06573  0.05097  0.02918
## Detection Rate          0.728  0.00024  0.00136  0.00032  0.00296
## Detection Prevalence    0.922  0.00264  0.01539  0.00528  0.05509
## Balanced Accuracy       0.512  0.50048  0.50283  0.50053  0.52383
```

Indeed the results are different from model 5, predicting substantial amounts of cases in all five delay categories. Predictions for the first category (0-15 min) are quite good, with an F1 score of 0.853. But the performance in the other categories is not as good. Let's see if the balanced dataset gives better results:

```
# Show model performance with explicit levels due to
# missing predictions
confusionMatrix(table(m5_predict_bal, test$DelayCategory), mode = "prec_recall",
    positive = "1")
```

```
## Confusion Matrix and Statistics
##
##
## m5_predict_bal     a     b     c     d     e
##              a 19564  1689  1630  1264   727
```

```
##              b    15     4     5     1     1
##              c    38     7     6     6     2
##              d    14     1     1     3     0
##              e    32     0     2     1     0
##
## Overall Statistics
##
##                Accuracy : 0.783
##                  95% CI : (0.778, 0.788)
##     No Information Rate : 0.786
##     P-Value [Acc > NIR] : 0.909
##
##                   Kappa : 0.003
##
##  Mcnemar's Test P-Value : <2e-16
##
## Statistics by Class:
##
##                      Class: a Class: b Class: c Class: d Class: e
## Precision               0.787  0.15385  0.10169  0.15789   0.0000
## Recall                  0.995  0.00235  0.00365  0.00235   0.0000
## F1                      0.879  0.00463  0.00705  0.00464      NaN
## Prevalence              0.786  0.06800  0.06573  0.05097   0.0292
## Detection Rate          0.782  0.00016  0.00024  0.00012   0.0000
## Detection Prevalence    0.994  0.00104  0.00236  0.00076   0.0014
## Balanced Accuracy       0.501  0.50070  0.50069  0.50084   0.4993
```

The confusion matrix shows that overall accuracy has dropped from 73% to 51%, and that individual F1 scores have changed - but overall not improved. This may be due to the fact that the balanced dataset was balanced on delay vs. no-delay and not balanced to represent all five delay classes equally.

Model 7: kNN on delay category Although it does not seem that predicting delay category rather than delay/no-delay yields better results, we will have a final try with one of the better performing algorithms. We will again use the k-Nearest neighbors model, but this time to predict delay category. Here is the model training code:

```
# Fit model on imbalanced training set
ctrl <- trainControl(method = "repeatedcv", repeats = 1)
m7_fit <- train(DelayCategory ~ DepWeek + UniqueCarrier + Dest +
    DepTimeslot, method = "knn", data = train, trControl = ctrl,
    preProcess = c("center", "scale"), tuneLength = 5)
m7_fit_bal <- train(DelayCategory ~ DepWeek + UniqueCarrier +
    Dest + DepTimeslot, method = "knn", data = train_bal, trControl = ctrl,
    preProcess = c("center", "scale"), tuneLength = 5)

# Make prediction on test set (delay probability)
m7_predict <- predict(m7_fit, test)
m7_predict_bal <- predict(m7_fit_bal, test)
```

Let's see the results for this algorithm on the imbalanced dataset:

```r
# Show model performance with explicit levels due to
# missing predictions
confusionMatrix(table(m7_predict, test$DelayCategory), mode = "prec_recall",
    positive = "1")
```

```
## Confusion Matrix and Statistics
##
##
## m7_predict     a     b     c     d     e
##          a 19615  1684  1629  1260   722
##          b    18     9     4     5     1
##          c    12     6     9     8     3
##          d     7     1     1     1     3
##          e    11     1     1     1     1
##
## Overall Statistics
##
##                Accuracy : 0.785
##                  95% CI : (0.78, 0.79)
##     No Information Rate : 0.786
##     P-Value [Acc > NIR] : 0.67
##
##                   Kappa : 0.009
##
##  Mcnemar's Test P-Value : <2e-16
##
## Statistics by Class:
##
##                      Class: a Class: b Class: c Class: d Class: e
## Precision               0.787  0.24324  0.23684 0.076923  0.06667
## Recall                  0.998  0.00529  0.00547 0.000784  0.00137
## F1                      0.880  0.01036  0.01070 0.001553  0.00268
## Prevalence              0.786  0.06800  0.06573 0.050973  0.02918
## Detection Rate          0.784  0.00036  0.00036 0.000040  0.00004
## Detection Prevalence    0.996  0.00148  0.00152 0.000520  0.00060
## Balanced Accuracy       0.504  0.50204  0.50212 0.500139  0.50040
```

This is a picture we have seen many times before now. The model just predicts a no-delay for almost all cases, thereby yielding a rather poor outcome. We can see if the balanced dataset produces better results.

```r
# Show model performance with explicit levels due to
# missing predictions
confusionMatrix(table(m7_predict_bal, test$DelayCategory), mode = "prec_recall",
    positive = "1")
```

```
## Confusion Matrix and Statistics
##
##
## m7_predict_bal     a     b     c     d     e
##              a 17440  1202  1146   924   528
##              b   834   280   169    97    52
##              c   764   117   229   102    58
```

```
##              d   439    65    74   118    36
##              e   186    37    26    34    56
##
## Overall Statistics
##
##                Accuracy : 0.725
##                  95% CI : (0.719, 0.73)
##     No Information Rate : 0.786
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.148
##
##  Mcnemar's Test P-Value : <2e-16
##
## Statistics by Class:
##
##                      Class: a Class: b Class: c Class: d Class: e
## Precision               0.821   0.1955  0.18031  0.16120  0.16519
## Recall                  0.887   0.1646  0.13929  0.09255  0.07671
## F1                      0.853   0.1787  0.15717  0.11759  0.10477
## Prevalence              0.786   0.0680  0.06573  0.05097  0.02918
## Detection Rate          0.697   0.0112  0.00916  0.00472  0.00224
## Detection Prevalence    0.849   0.0573  0.05077  0.02926  0.01355
## Balanced Accuracy       0.588   0.5576  0.54737  0.53334  0.53253
```

As the F1 scores for each class suggest, this outcome is actually better than on the imbalanced dataset. But again the model is mostly good for predicting no-delays (class a) and is less able to specifically predict other classes.

2.7 Model performance and practical application

Below table summarizes the results from the previous step, describing the performance of each of the models for the different dataset types.

Table 2: Model results (F1 and Accuracy)

Model	Dataset	F1	Accuracy
M1 - Logistic Regression on yes/no	Imbalanced	0.378	0.514
M1 - Logistic Regression on yes/no	Balanced	0.376	0.518
M2 - Decision tree on yes/no	Imbalanced	0.251	0.726
M2 - Decision tree on yes/no	Balanced	0.369	0.293
M3 - Random forest on yes/no	Imbalanced	NA	0.786
M3 - Random forest on yes/no	Balanced	0.352	0.214
M4 - k-Nearest neighbors on yes/no	Imbalanced	0.385	0.574
M4 - k-Nearest neighbors on yes/no	Balanced	0.433	0.639
M5 - PDA Regression on category (imbalanced)	Imbalanced	NA	0.786
M5 - PDA Regression on category (balanced)	Balanced	NA	0.783
M6 - HDDA Regression on category (imbalanced)	Imbalanced	NA	0.733
M6 - HDDA Regression on category (balanced)	Balanced	NA	0.510
M7 - k-Nearest neighbors on category (imbalanced)	Imbalanced	NA	0.785
M7 - k-Nearest neighbors on category (balanced)	Balanced	NA	0.725

Based on this overview, we can conclude that the k-Nearest neighbors model on the balanced dataset performed best in terms of F1 score (0.433). Looking at Accuracy (more relevant for models M5 through M7), models M3, M5, M6 and M7 stand out - all with an accuracy of around 78%. However - the way that most of these models achieve this result is by simply predicting most or all flights to be not-delayed, thereby being wrong only 21% of the times. This does not add anything to the knowledge we already had from the exploratory data analysis.

In practical application, even the best performing models we trained would do rather poorly. Let's say we incorporate these models into a flight booking system, advising users about whether a specific flight will be delayed. In that case, it makes no sense to use this model to predict a flight is not delayed and then being wrong in about 21% of predictions. Then we might just as well ignore the prediction model and just state that the overall probability of a delay is 21%.

That is why a different way of using the models might work. Let's say we use M1 to predict either delay or no-delay. This was the confusion matrix for M1 on the balanced dataset:

```
cm <- table(prediction = m1_predict_class, reference = test$Delayed)
cm
```

```
##           reference
## prediction     0     1
##          0  9171  1664
##          1 10492  3686
```

Next we divide flights by prediction class: 0 for no-delay and 1 for delay. For the 0-group, the confusion matrix shows that prediction was correct in 85% of cases. In other words: of flights predicted to be on time, actually 15% was still delayed. That means this group of flights has a slightly lower chance of being delayed than a random flight (which has a 21% chance of delay). Now let's look at the second group. Flights in the 1-group were actually mostly not delayed. Looking at the confusion matrix, the predictions for the 1-group were wrong in 74% of cases. In other words: of flights predicted to be delayed, actually 26% was delayed. These are the probabilities we just described:

```
# False negative, ie. probability of delay when no-delay
# predicted
p_fn <- cm[3]/(cm[1] + cm[3])
p_fn
```

```
## [1] 0.154
```

```
# True positive, ie. probability of delay when delay
# predicted
p_tp <- cm[4]/(cm[2] + cm[4])
p_tp
```

```
## [1] 0.26
```

Using this approach, we can actually present the user with a slightly more specific delay probability for the chosen flight than just the overall delay probability of 21%. If the flight was predicted a 0, then we present a delay probability of 15%; if the flight was predicted a 1, then we present a delay probability of 26%. Both groups give the user a roughly 5%-point more specific delay probability than the overall delay probability. The more distant from the overall average (ie. the larger the spread between both probabilities), the more specific the prediction is and the better it serves our user. We will call this metric 'probability specificity' or 'ProbSpec'.

```r
# Calculate 'ProbSpec' or spread between true positive and
# false negative probabilities
m1_probspec <- p_tp - p_fn
m1_probspec
```

```
## [1] 0.106
```

With the help of a simple custom function, we can calculate the ProbSpec for each of the models we ran (see below code). If the function is applied to a model trained to predict delay categories rather than simply delay/no-delay, then the function automatically converts the predictions to a 2x2 confusion matrix. That enables us to assess the model performance on this metric the same way for every model.

```r
# Define function to calculate ProbSpec or spread between
# true positive and false negative The function takes two
# arguments: the predicted values and the reference (true)
# values
calc_probspec <- function(pred, ref) {
    # Create confusion matrix
    cm <- table(pred, ref)
    # Check if is a 5x5 matrix or 2x2
    if (dim(cm)[1] > 2) {
        # If it is a 5x5 matrix, add categories b to e into
        # one category
        cm_1 <- cm[1]
        cm_2 <- sum(cm[2:5])
        cm_3 <- sum(cm[6], cm[11], cm[16], cm[21])
        cm_4 <- sum(cm[6:25]) - cm_3
    } else {
        # If it is a 2x2 matrix, then just use the four
        # cells
        cm_1 <- cm[1]
        cm_2 <- cm[2]
        cm_3 <- cm[3]
        cm_4 <- cm[4]
    }
    # Calculate positive and negative prediction rates
    p_p <- (cm_2 + cm_4)  # Total cases predicted 1
    p_n <- (cm_1 + cm_3)  # Total cases predicted 0
    p_t <- p_p + p_n  # Total cases predicted overall
    # Calculate false negative probability
    p_fn <- cm_3/p_n
    # Calculate true positive probability
    p_tp <- cm_4/p_p
    # Calculate spread
    probsec <- p_tp - p_fn
    # Return spread, probabilities and distribution over
    # classes
    return(c(probsec, p_fn, p_tp, p_n/p_t, p_p/p_t))
}
```

The table below summarizes the results, with the models yielding the largest spread between both probability predictions (ie. the highest ProbSpec) first. Columns FN and TP specify both probability predictions (lower

Table 3: Model results (ProbSpec metric)

Model	Dataset	ProbSpec	FN	TP	Negatives	Positives
M7 - k-Nearest neighbors on category	Imbalanced	0.321	0.213	0.534	0.996	0.004
M7 - k-Nearest neighbors on category	Balanced	0.232	0.179	0.411	0.849	0.151
M4 - k-Nearest neighbors on yes/no	Balanced	0.195	0.131	0.326	0.577	0.423
M2 - Decision tree on yes/no	Balanced	0.151	0.077	0.228	0.093	0.907
M4 - k-Nearest neighbors on yes/no	Imbalanced	0.124	0.154	0.279	0.521	0.479
M1 - Logistic Regression on yes/no	Imbalanced	0.106	0.154	0.260	0.433	0.567
M2 - Decision tree on yes/no	Imbalanced	0.104	0.198	0.302	0.848	0.152
M1 - Logistic Regression on yes/no	Balanced	0.104	0.156	0.260	0.442	0.558
M5 - PDA Regression on category	Imbalanced	0.094	0.214	0.308	0.999	0.001
M5 - PDA Regression on category	Balanced	0.074	0.213	0.288	0.994	0.006
M6 - HDDA Regression on category	Balanced	0.071	0.185	0.256	0.592	0.408
M6 - HDDA Regression on category	Imbalanced	0.054	0.210	0.264	0.922	0.078
M3 - Random forest on yes/no	Imbalanced	NaN	0.214	NaN	1.000	0.000
M3 - Random forest on yes/no	Balanced	NaN	0.214	NaN	1.000	0.000

and upper) and columns Negatives and Positives show the share of flights predicted to be on-time (negative) or delayed (positive).

As we can see from the values, the models simply predicting delay or no-delay perform better on this metric than the models that predict a delay category. The first M7 (a k-Nearest neighbors model) yields the best ProbSpec, but this model predicts nearly all flights to be on-time (see column Negatives), thereby not adding any value. The second M7 model comes closer to our desired result. For 85% of flights, it predicts a delay probability of 18% (slightly below the overall average) and for the remaining 15% of flights, it predicts a much higher delay probability of 41%. That is information a user looking to choose a flight would appreciate!

Going down the list a bit, the third model (M4, a k-Nearest neighbors model on the balanced dataset) might actually be slightly more useful to our users. For 58% of flights, it predicts a delay probability of 13% - well below the overall average. For the remaining 42% of flights, it predicts a 33% delay probability - substantially higher than the overall average. Because lower and upper probabilities both differ substantially from the overall delay probability of 21%, this model is the most informative from a users' perspective. That is no coincidence: this is also the model that resulted in the highest F1 score.

Given the above assessment, the best performing model in terms of practical application is the M4 model using k-nearest neighbors algorithm on the adjusted, balanced dataset.

3 Results

For each of the seven algorithms that were trained, the performance was evaluated using the F1 score and, for multiclass models, Accuracy score. The table below summarizes performance scores, distinguishing between models trained on the imbalanced and balanced datasets.

Table 4: Model results (F1 and Accuracy)

Model	Dataset	F1	Accuracy
M1 - Logistic Regression on yes/no	Imbalanced	0.378	0.514
M1 - Logistic Regression on yes/no	Balanced	0.376	0.518
M2 - Decision tree on yes/no	Imbalanced	0.251	0.726
M2 - Decision tree on yes/no	Balanced	0.369	0.293
M3 - Random forest on yes/no	Imbalanced	NA	0.786
M3 - Random forest on yes/no	Balanced	0.352	0.214
M4 - k-Nearest neighbors on yes/no	Imbalanced	0.385	0.574
M4 - k-Nearest neighbors on yes/no	Balanced	0.433	0.639
M5 - PDA Regression on category (imbalanced)	Imbalanced	NA	0.786
M5 - PDA Regression on category (balanced)	Balanced	NA	0.783
M6 - HDDA Regression on category (imbalanced)	Imbalanced	NA	0.733
M6 - HDDA Regression on category (balanced)	Balanced	NA	0.510
M7 - k-Nearest neighbors on category (imbalanced)	Imbalanced	NA	0.785
M7 - k-Nearest neighbors on category (balanced)	Balanced	NA	0.725

All models performed rather poorly in predicting if a flight is delayed or not. The model that showed the best performance is M4, a k-Nearest neighbors model trained on the balanced dataset. But given the accuracy of 64% even this model was wrong in 36% of cases

Given the poor prediction performance, none of the models is found to be suitable to advise users of a flight booking system on which flights will be delayed or not. That is why another approach was developed, presenting users with a more specific delay probability for a particular flight. To evaluate each model's performance for this application, a new metric 'ProbSpec' (short for probability specificity) was developed, indicating how specific (ie. different from the overall delay probability of 21%) the presented probabilities are. The table below summarizes the ProbSpec scores for each of the models.

Table 5: Model results (ProbSpec metric)

Model	Dataset	ProbSpec	FN	TP	Negatives	Positives
M1 - Logistic Regression on yes/no	Imbalanced	0.106	0.154	0.260	0.433	0.567
M1 - Logistic Regression on yes/no	Balanced	0.104	0.156	0.260	0.442	0.558
M2 - Decision tree on yes/no	Imbalanced	0.104	0.198	0.302	0.848	0.152
M2 - Decision tree on yes/no	Balanced	0.151	0.077	0.228	0.093	0.907
M3 - Random forest on yes/no	Imbalanced	NaN	0.214	NaN	1.000	0.000
M3 - Random forest on yes/no	Balanced	NaN	0.214	NaN	1.000	0.000
M4 - k-Nearest neighbors on yes/no	Imbalanced	0.124	0.154	0.279	0.521	0.479
M4 - k-Nearest neighbors on yes/no	Balanced	0.195	0.131	0.326	0.577	0.423
M5 - PDA Regression on category	Imbalanced	0.094	0.214	0.308	0.999	0.001
M5 - PDA Regression on category	Balanced	0.074	0.213	0.288	0.994	0.006
M6 - HDDA Regression on category	Imbalanced	0.054	0.210	0.264	0.922	0.078
M6 - HDDA Regression on category	Balanced	0.071	0.185	0.256	0.592	0.408
M7 - k-Nearest neighbors on category	Imbalanced	0.321	0.213	0.534	0.996	0.004
M7 - k-Nearest neighbors on category	Balanced	0.232	0.179	0.411	0.849	0.151

Of the models used for this approach, model M7 (also a k-Nearest neighbors model), trained on both the imbalanced and balanced datasets, showed the highest ProbSpec score. However, since these models group most flights into one group, they are actually not suitable to present users with a more specific delay probability. Instead, the third-ranking model is selected as the winner: the same M4 model, trained on the balanced dataset. Using this model would enable us to present a user with differentiated delay probabilities of either 13% or 33% instead of the overall 21%.

4 Conclusion

This final chapter summarizes the results of this data science project and puts these in a perspective of limitations and recommendations for further analysis.

4.1 Summary of results

In this data science project, several machine learning models were developed to predict if a particular flight is delayed or not, based on predictors such as the planned departure time, airline, and destination. The models were trained on dataset containing all flights departing from San Francisco in 2019, containing 166,750 flights.

Of the seven models trained and tested, four were ran to simply predict delay or no-delay for each flight in the test set. The other three models were trained to predict delay time classification (0-15 min, 16-30 min, 31-60 min, 61-120 min, >120 min). Of all models, M4 (k-Nearest neighbors) performed best. But with an F1 score of 0.433 and accuracy of 64%, predictions are still quite inaccurate and of little help in a flight booking system, the hypothetical application for this machine learning project.

That is why another approach was developed, giving a user, who is looking to choose the best flight, a more specific probability of each flight's delay. Using the machine learning models we can do better than either inaccurately predicting a delay or no-delay or giving the overall delay probability of 21%. Instead, using the M4 model we can inform the user that a particular flight either has a 13% or a 33% chance of delay. This should allow the user to make a better informed decision for which flight to book.

4.2 Limitations of the analysis

Given the nature of this project, the analysis and results presented have some limitations. The first is that the predictors present in the dataset have proved to have too little predictive power to actually develop a well-performing prediction model. Even considering the information in the dataset (such as departure time, airline and destination), there is in practice still too much (random) variation between delayed and not-delayed flights. In order to build a better performing model, other indicators should be taken into account, such as operational disruptions at the time of departure, weather conditions, aircraft malfunctions etc.

A second limitation is that only basic machine learning techniques were used in this analysis. More advanced techniques, such as neural networks, can improve the predictive power of a given model, but are also more complex and time-consuming to implement. Also, more thorough tuning of models such as random forests might yield better results, but was not performed in this project due to time constraints.

A third limitation is that only a subset of the full dataset with flights was used (only departing from San Francisco), due to limited computing power available. Models might perform better if they have more cases of delay (and from other departure airports) available in the training set.

4.3 Recommendations for future analysis

If one would want to build on or improve this analysis in the future, the limitations from the previous paragraph provide some guidance into how to do this. There are three recommendations for future analysis:

1. Enrich the dataset with exogenous factors, such as operational disruptions and weather conditions at the time and location of departure.
2. Apply more advanced techniques, such as neural networks to improve the predictive power of the model.
3. Increase the dataset size to include other airports of departure, and even other years.

Notes

1. 'Multi-Class Metrics Made Simple, Part II: the F1-score', Boaz Smueli (2019), published on https://towardsdatascience.com/multi-class-metrics-made-simple-part-ii-the-f1-score-ebe8b2c2ca1
2. 'Introduction to Data Science: Data Analysis and Prediction Algorithms with R', Rafael A. Irrizary (2021). Published on https://rafalab.github.io/dsbook/.
3. '8 Tactics to Combat Imbalanced Classes in Your Machine Learning Dataset', Jason Brownlee (2020), published on https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/
4. '10 Techniques to deal with Imbalanced Classes in Machine Learning', Benai Kumar (2020), published on https://www.analyticsvidhya.com/blog/2020/07/10-techniques-to-deal-with-class-imbalance-in-machine-learning/