Harvardx Capstone
Domestic Flight Arrivals Delays into LAX Airport

Laura Butgereit

08/05/2021

Contents

List of Figures

List of Tables

1 Introduction

In a 2017, report, the International Air Transport Association (IATA) valued the airline industry in the United States at 779 billion US Dollars (IATA, 2017). The recent 2020 effects of locksdowns around the world brought on by governmental responses to the novel Coronavirus have decimated the airline industry (IATA, 2020). However, with vaccination rollouts happening worldwide, there are recent indications that a slight recovery in the airline industry has started.

This paper looks back to a time when the airline industry was robust. It attempts to use historical average arrival delays of airflights and to classify a flight as being either on-time or late. It is acknowledged that using pre-lockdown data may or may not be applicable to predictions in a post-lockdown situation or mid-lockdown situation.

This project was part of a Capstone project in Harvardx's *Professional Certificate in Data Science*. As such, some accomodation needs to be given to the academic nature of this research. This paper assumes a pre-lockdown world.

Airlines and airports pride themselves in their on-time arrival statistics. Awards are often given for airlines with best on-time statistics (OAG, 2020).

This project looks at domestic arrivals at one specific airport (LAX - Los Angeles International) during the period of time 1995 to 2020. Using this data, this project attempts to answer the question

Can predictions be made as to whether or not a flight will be late?

In order to answer that question, however, investigations must be made into the average arrival delay time.

Section 2 describes the computational environment where this research was executed. This section also itemizes any modifications that might have needed to be done in order to process the large data files.

Section 3 describes where the data was sourced and issues which were encountered in downloading the required data files. This section provides various strategies which were utilized to handle these issues.

Section 4 describes how the original data was shaped into datasets for this specific project. This section describes the columns which were used. It also describes how the training set, test set, and validation set

were created.

Section 5 of this paper describes the exploration into the dataset and the analysis thereof. This analysis specificially looked at how various data fields such as airline, departure airport, and time of day might affect the average arrival delays of an airflight. The assumption is that arrival delay values are indicative of whether or not a flight is late.

Section 6 describes the various investigations into possible models to be able to predict whether the flight was on-time or not. There were a number of different strategies which were investigated. For each model which attempted to predict whether a flight was on-time or late, the accuracy, sensitivity, and specificity are provided.

A summary of the results for the final model including the confusion matrix can be found in Section 6.5.

Conclusions can be found in Section 7.

Before continuing with this document, the author wishes to share the wise words of Donald Knuth (1984). In his journal paper entitled *Literate Programming* he states:

> Let us change our traditional attitude to the construction of [computer] programs: Instead of imagining that our main task is to instruct a *computer* what to do, let us concentrate rather on explaining to *human beings* what we want a computer to do.
>
> The practitioner of literate programming can be regarded as an essayist, whose main concern is with exposition and excellence of style. Such an author, with thesaurus in hand, chooses the names of variables carefully and explains what each variable means. He or she strives for a program that is comprehensible because its concepts have been introduced in an order that is best for human understanding...

The author eschews variables such as *mu* and *y_hat* (which may be familiar to data science practitioners and statisticians), in favour of more descriptive variables such as *flights_by_origin_airport* and *flights_by_departure_delay*. Although this practice may be in conflict with code examples in (Irizarry, 2021), the author is employed in the IT industry and is attending this course with the view of integrating R (running in an automated fashion started by Java programs) into a Spring Boot environment. As such the practices of good naming conventions in source code are well ingrained.

2 Compute Environment and Memory issues

The analyses presented in the paper were done on a Dell XPS 13 9380 13.3" Core i7-8565U Notebook. The notebook has 8GB memory and a Core i7-8565U processor. The notebook was running Ubuntu 20.04.2 LTS with R 4.0.3 installed.

2.1 Adding more swap space

There were memory issues and an additional 16GB swap space was configured on the notebook with the following Linux commands executed as superuser:

```
fallocate -l 16G /harvardx_swapspace
chmod 600 /harvardx_swapspace
mkswap /harvardx_swapspace
swapon /harvardx_swapspace
```

In order to have this swapspace available automatically when the notebook boots up an entry needed to be made in the file

```
/etc/fstab
```

which looks like

```
/harvardx_swapfile none swap sw 0 0
```

NB: In retrospect, even with 16GB of swapspace, some of the models which the author wished to run aborted for lack of memory. For the models that did run, 8GB of swapspace was sufficient

2.2 Modifying *swappiness*

The *swappiness* value defines how aggressively the Linux kernel swaps memory pages to swap devices. The value ranges between zero and one hundred (Ljubuncic, 2015). A higher *swappiness* value implies a stronger preference toward swapping more readily. A lower value implies not swapping (Love, 2013). The value can be configured on this file

```
/etc/sysctl.conf
```

The entry looks like

```
vm.swappiness=10
```

This change was made to stop Linux swapping excessively.

2.3 Request extra memory when executing R

In order to successfully build and execute, the scripts were run in R and not Rstudio with the following command

```
R -max-mem-size=7000M --vanilla < Butgereit-script.R  > output.txt
```

The report was generated with the following command

```
Rscript -e "rmarkdown::render('Butgereit-report.Rmd')"
```

3 Data Sources - Downloading, unzipping, and validating data

The original data source for this project was compiled by the US Department of Transport and is published on the Bureau of Transportation Statistics website (BTS, 2021). The dataset also appears on many machine learning websites. For the scope of the project described in this paper, the data was downloaded from IBM's developer website for artificial intelligence (IBM, 2021). For the scope of this project, the dataset identified as *2 Million Row Sample Dataset* was used.

The author lives in a rural area of Africa where download speeds are slow and connections often break. For that reason, a *timeout* parameter needs to be used for all downloads. This is setup as a pseudo constant at the top of the R script. *The code reviewer may need to increase the parameter depending on his or her local circumstances.* The code then attempted to download the required zip file, store the file locally, and then to check the size of the downloaded file to verify that the entire data file was downloaded. If the download and size check did not succeed, error messages are printed and the script is stopped. If the download and size check did succeed, only then is the data unzipped.

3.1 Set up pseudo-constants

First a handful of pseudo-constants were set up

```
#
# download time out.  if you have slow internet, you may need to
# increase this value.  it is measured in seconds
#
download_timeout <- 600 # seconds == 10 minutes

#
# url of data file.  important stats about data file etc
#
on_time_performance_url <-
    "https://dax-cdn.cdn.appdomain.cloud/dax-airline/1.0.1/airline_2m.tar.gz"
on_time_performance_gz <- "airline_2m.tar.gz"
on_time_performance_gz_size <- 151681776
on_time_performance_csv <- "airline_2m.csv"
on_time_performance_csv_size <- 882162600
on_time_performance_rda <- "on_time_performance.rda"

#
# intermediate objects
#
validation_rda <- "validation.rda"
non_validation_rda <- "non_validation.rda"
test_rda <- "test.rda"
train_rda <- "train.rda"

#
# airport to investigate arrivals
#
destination_airport <- "LAX"
#
# percentage for validation set and then test set
#
percent <- 0.1

#
```

```
# which iteration to run
#
build_late_models_iteration_1 <- TRUE
build_late_models_iteration_2 <- TRUE
build_late_models_iteration_3 <- TRUE
build_late_models_iteration_4 <- TRUE
```

3.2 Download zip file and check sizes

The script was broken up into sections where the sizes of all the downloaded files were tested before the next step was executed.

```
#
# if the gz file has not been downloaded or if it is
# the wrong size, download it
#
if ( (!file.exists(on_time_performance_gz)) ||
     (file.info(on_time_performance_gz)$size != on_time_performance_gz_size) ) {

        #
        # Download gz file
        #
        print(paste("file", on_time_performance_gz,
        "does not exist or is the wrong size", sep=" "))
        print("Downloading it")
        options(timeout = download_timeout)
        download.file(on_time_performance_url, on_time_performance_gz)
        print(paste("Downloaded", file.info(on_time_performance_gz)$size,
        "bytes"))

        #
        # check to see if the entire file got downloaded
        #
        if ( file.info(on_time_performance_gz)$size == on_time_performance_gz_size) {
                print(paste(on_time_performance_gz, "downloaded OK", sep=" "))
        } else {
                print(paste(on_time_performance_gz,
            "was not completely downloaded", sep=" "))
                print(paste("You can download it through your browser ",
            "by pointing your browser to"), sep="")
                print(on_time_performance-url)
                stop()
        }
}
```

3.3 Untar file and check sizes

The downloaded file is in gzipped tar format. The R function *untar* does both a gzip decompress and an untar of the files

```
#
# if the csv file does not exist or is the wrong size
# or if it is older than the gz file, then untar the
# gz file
#
```

```
if ( (!file.exists(on_time_performance_csv)) |
     (file.info(on_time_performance_csv)$size != on_time_performance_csv_size) ||
     (file.info(on_time_performance_gz)$ctime >
        file.info(on_time_performance_csv)$ctime) ) {

        #
        # untar (inclues gz decompress)
        #
        print(paste("Untaring (will also do a gz decompress)",
        on_time_performance_gz, sep=" "))
        untar(on_time_performance_gz)

        #
        # check to see if the files sizes are correct
        #
        if ( file.info(on_time_performance_csv)$size == on_time_performance_csv_size) {
                print(paste(on_time_performance_csv, "untared OK", sep=" "))
        } else {
                print(paste(on_time_performance_csv,
            "was not untarred correctly", sep=" "))
                print(paste("You can try decrompressing and untaring ",
            "it your self and rerunning this script"), sep="")
                stop()
        }
}
```

4 Extracting and Creating datasets

The original downloaded file contained numerous pieces of information.

There are 109 columns in the original data file.

4.1 Filter data and select columns

Many of these columns were empty and many were not relevant to the issue at hand. The author followed an interative approach into determining which columns were important. An additional column was added which indicated whether or not the flight was late in arriving.

```
#
# if the on_time_performance_rda file does not exist,
# or if the csv file is younger than it, then rebuild it
#
if ( (!file.exists(on_time_performance_rda)) ||
     (file.info(on_time_performance_csv)$ctime >
        file.info(on_time_performance_rda)$ctime) ) {

        #
        # read in the csv
        #
        print("Reading data file")
        csv <- read.csv(on_time_performance_csv)
        print("read csv")

        #
        # filter out only flights landing at destination_airport
        # select the columns which might interest us (human choice)
        # only keep complete cases
        # add a flag for Late
        #
        on_time_performance <- csv %>%
                filter(Dest == destination_airport) %>%
                select(Year, Month, DayofMonth, DayOfWeek,
                        DOT_ID_Reporting_Airline,
                        Flight_Number_Reporting_Airline,
                        OriginAirportID,
                        Tail_Number,
                        Distance,
                        ArrDel15,
                        AirTime,
                        ArrDel15,
                        DepTime, ArrTime,
                        DepDelay, ArrDelay) %>%
                filter(complete.cases(.)) %>%           # NB the full stop
                mutate(late = factor(ifelse(ArrDelay > 0, 1, 0)),
                       arrival_delay = factor(ArrDelay),
                       departure_slot = previous_half_hour(DepTime),
                       arrival_slot = previous_half_hour(ArrTime),
                       arrival_slot_number = factor(arrival_slot),
                       airline = factor(DOT_ID_Reporting_Airline),
                       flight_number = factor(Flight_Number_Reporting_Airline),
                       tail_number = factor(Tail_Number),
```

```
                        origin_airport = factor(OriginAirportID))
        print("filtered out NAs and selected columns")

        save(on_time_performance, file=on_time_performance_rda)
        print(paste("Saved", on_time_performance_rda))
}
```

It is important to note that some fields are essentially in the dataset twice. For example, there is a field *Tail_Number* and a field *tail_number*. The first field, *Tail_Number*, is the original data and the second field, *tail_number* is recreated as a factor.

4.2 Creating *validation*, *train*, and *test* datasets

From this newly created dataset, four subsets were created. The first subset was to extract a validation set which would be put aside and not accessed until the final model was going to be evaluated. The second subset was the remaining elements which were not in the validation set. This second subset (the non validation dataset), was subdivided into a training set and a testing set. All four of these subsets were stored on intermediate R objects.

```
#
# if validation, train, and test do not exist or if
# one of them is younger than the big rda
#
# make sure that the Late flag is equitably distributed
#
if ( !file.exists(test_rda) || !file.exists(train_rda) || !file.exists(validation_rda) ||
     !file.exists(non_validation_rda) ||
     (file.info(on_time_performance_rda)$ctime > file.info(validation_rda)) ||
     (file.info(on_time_performance_rda)$ctime > file.info(non_validation_rda)) ||
     (file.info(on_time_performance_rda)$ctime > file.info(train_rda)) ||
     (file.info(on_time_performance_rda)$ctime > file.info(test_rda))) {

    #
    # supress the warning about sample.kind
    #
    warnLevel <- getOption("warn")
    options(warn = -1)
    set.seed(1, sample.kind="Rounding")
    options(warn = warnLevel)
    load(on_time_performance_rda)

    #
    # make validation and non-validation datasets
    #
    validation_index <- createDataPartition(y=on_time_performance$late,
                    times=1, p=percent, list=FALSE)
    validation <- on_time_performance[validation_index,]
    non_validation <- on_time_performance[-validation_index,]


    #
    # divide the non-validation dataset into test and train
    #
    test_index <- createDataPartition(y=non_validation$late,
```

```
                    times=1, p=percent, list=FALSE)
    test <- non_validation[test_index,]
    train <- non_validation[-test_index,]

        #
        # save on intermediate data objects
        #
        save(test, file=test_rda)
        save(train, file=train_rda)
        save(validation, file=validation_rda)
        save(non_validation, file=non_validation_rda)

        #
        # free up memory
        #
        rm(test, train, validation, non_validation, on_time_performance)
}
```

The number of rows in each dataset can be seen in Table 1

Table 1: Sizes of various datasets

datasets	numRows
Non-validation	48516
Validation	5391
Train	43663
Test	4853

5 Exploratory Analysis

All values and graphs in this exploratory section refer only to the *non_validation* set. The *non_validation* dataset includes the training set and test set combined, but specifically excludes the validation set which was extracted and saved for future use. This section explores the data and reports on interesting findings.

The section first looks at the *ArrDelay* field and how it is affected by other fields. The section then looks specifically at the *late* field which indicates whether or not the flight arrived late.

5.1 Arrival delay

Although the term is called *ArrDelay*, the value can be positive (indicating an actual delay) or negative (indicating that the flight arrived early). Statistics about the *ArrDelay* field can be seen in Table 2

Table 2: Statistics about arrival delay times

Measurement	Value
Minimum	-90.000000
Maximum	1295.000000
Mean	5.768654
Std Dev	36.907533

5.2 Flights through the years

The dataset included records which spanned a number of years from 1995 to 2020. A distribution of the flights throughout the years can be seen in Figure 1

```
flights_per_year <- non_validation %>% group_by(Year) %>%
            summarize(number_of_flights = n(), Year = first(Year))
bar_graph(x = flights_per_year$Year, y = flights_per_year$number_of_flights,
    data=flights_per_year) +
    xlab("Year") +
    ylab("Number of flights") +
    scale_x_continuous(
        limits=c(min(non_validation$Year)-1, max(non_validation$Year)+1),
        breaks=seq(min(non_validation$Year), max(non_validation$Year), 5))
```

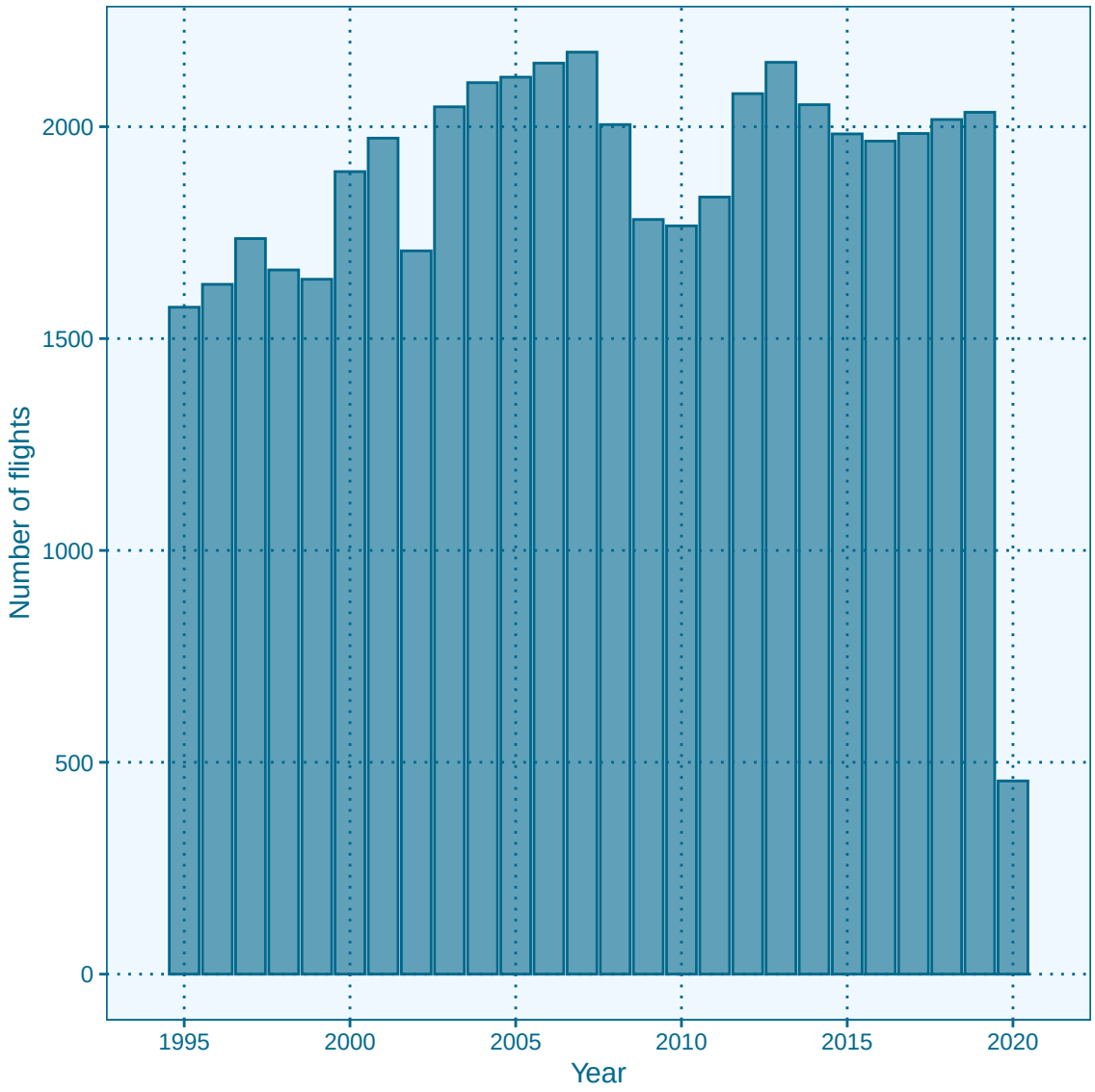

Figure 1: Distribution of flights by year

5.3 Arrival delay by year

The average arrival delay by year can be seen in Figure 2

```
flights_per_year <- non_validation %>% group_by(Year) %>%
    summarize(average_delay = mean(ArrDelay), Year = first(Year))
arrival_delay_graph(x = flights_per_year$Year, y = flights_per_year$average_delay,
    data=flights_per_year) +
    xlab("Year")
```

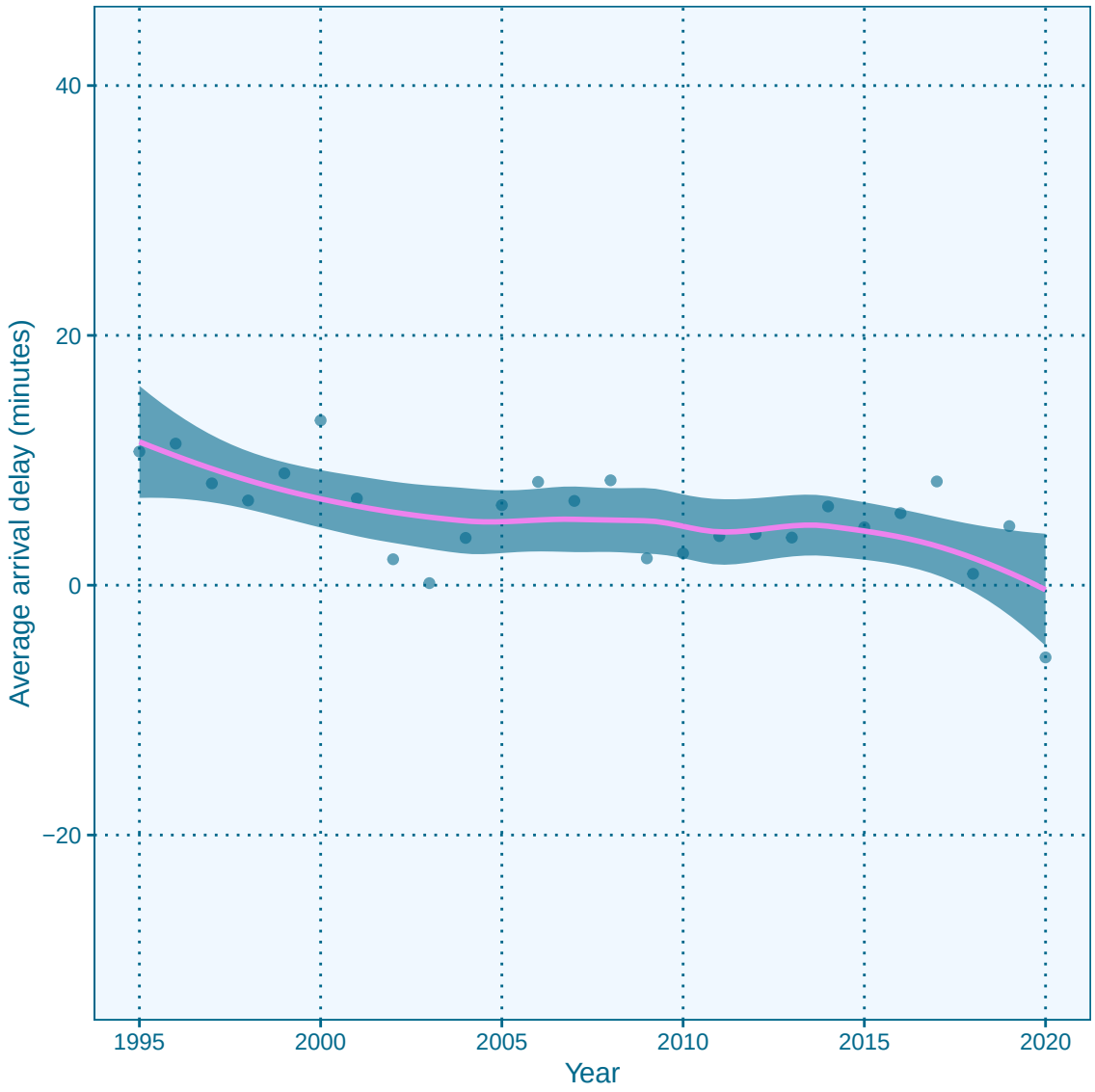

Figure 2: Average arrival delay by year

One may argue that Figure 2 may show an over all trend of decreasing average arrival delays over the years which may be due to the overall increase performance of the airline industry as a whole. The author, however, takes the view that this slight decrease is not significant.

5.4 Arrival delay by month

The month of the year (indicating the season and possible weather effects) could influence arrival delays. This relationship can be seen in Figure 3.

```
flights_per_month <- non_validation %>% group_by(Month) %>%
    summarize(average_delay = mean(ArrDelay), Month = first(Month))
arrival_delay_graph(x = flights_per_month$Month, y = flights_per_month$average_delay,
    data=flights_per_month) +
    xlab("Month") +
    ylab("Average delay (minutes)") +
    scale_x_continuous(limits=c(1, 12), breaks=1:12,
```

```
    labels=c("Jan", "Feb", "Mar", "Apr", "May", "Jun",
        "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"))
```

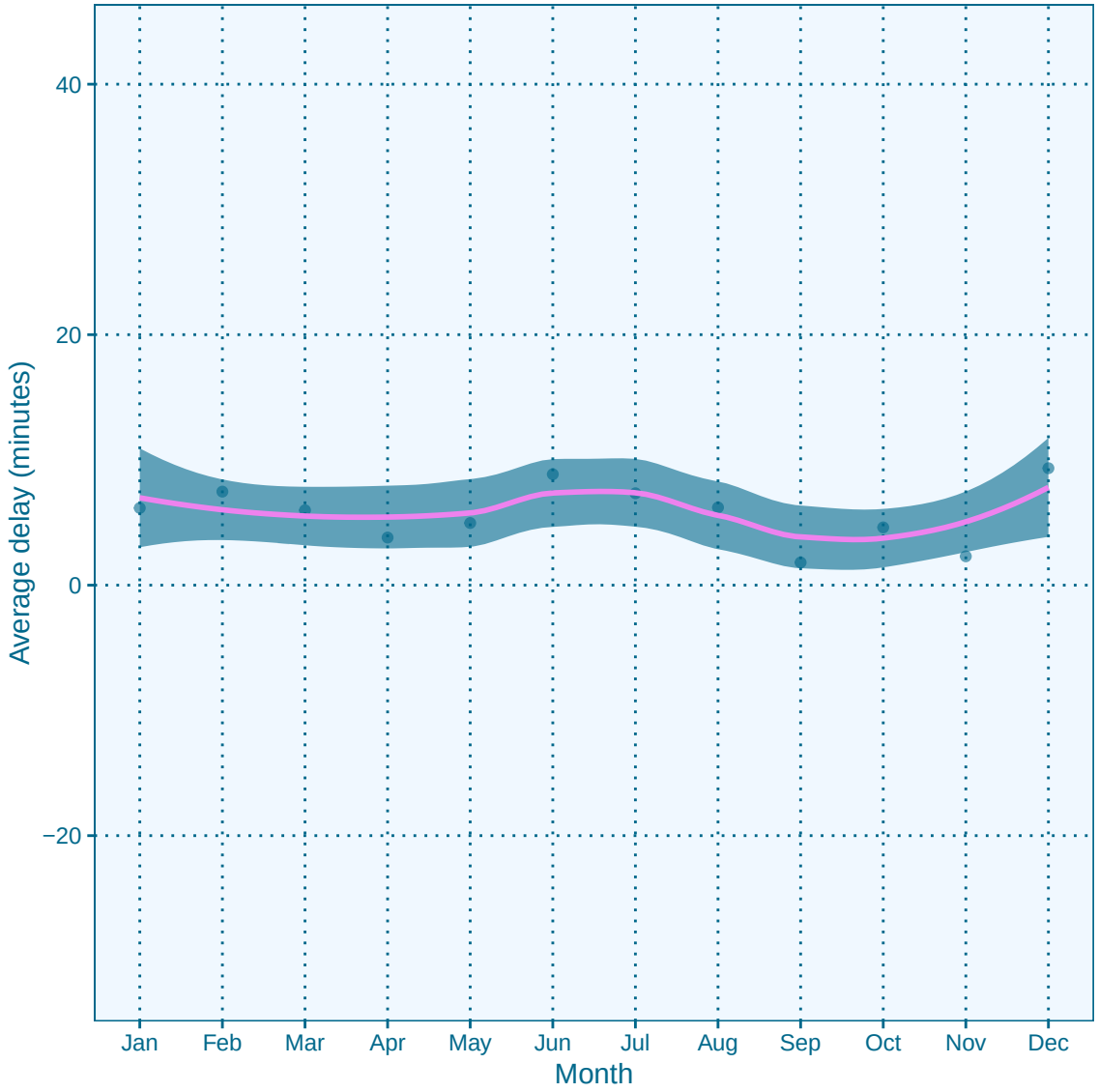

Figure 3: Average arrival delay by month

From Figure 3, one might argue that there is a very slight increase in average arrival delays in the summer months (June and July) and at the Christmas season (December). But the change is not significant.

5.5 Arrival delay by day of the month

The day of the month could possibly influence arrival times with end of month traffic being different than mid-month traffic. This relationship can be seen in Figure 4.

```
flights_per_day <- non_validation %>% group_by(DayofMonth) %>%
    summarize(average_delay = mean(ArrDelay), DayofMonth = first(DayofMonth))
arrival_delay_graph(x = flights_per_day$DayofMonth, y = flights_per_day$average_delay,
    data=flights_per_day) +
```

```
    xlab("Day of the Month") +
    scale_x_continuous(limits=c(1, 31), breaks=1:31)
```

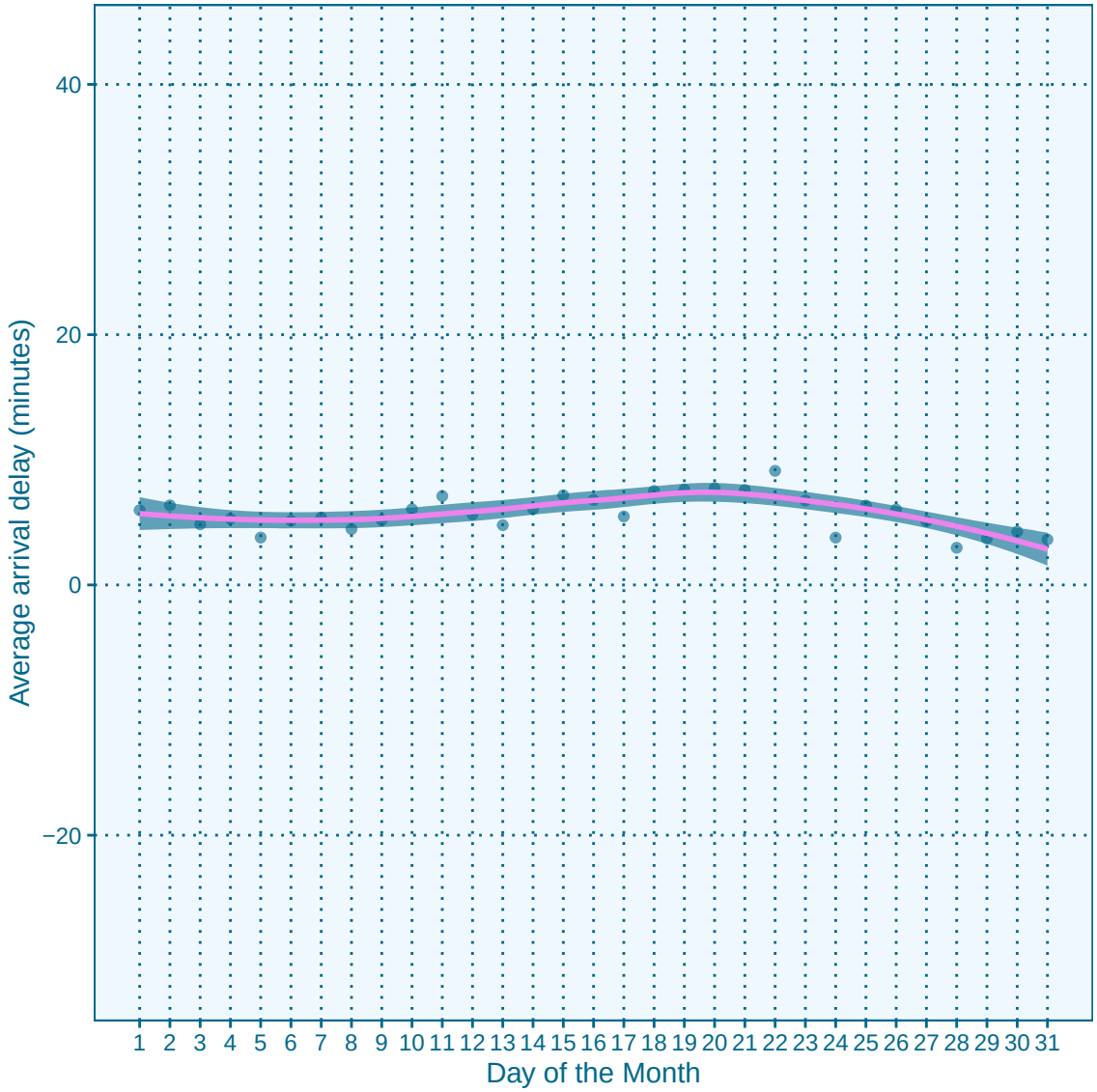

Figure 4: Average arrival delay by day of the month

Although one may argue that Figure 4 shows a very slight decrease in average delays at the end of the month, that decrease is not significant.

5.6 Arrival delay by day of the week

The day of the week could possibly influence arrival times with weekend traffic being different than weekday traffic. This relationship can be seen in Figure 5.

```
flights_per_day <- non_validation %>% group_by(DayOfWeek) %>%
    summarize(average_delay = mean(ArrDelay), DayOfWeek = first(DayOfWeek))
arrival_delay_graph(x = flights_per_day$DayOfWeek, y = flights_per_day$average_delay,
    data=flights_per_day) +
```

```
    xlab("Day of the Week") +
    ylab("Average delay (minutes)") +
    scale_x_continuous(limits=c(1, 7), breaks=1:7,
        labels=c("Sun", "Mon", "Tue", "Wed", "Thu",
        "Fri", "Sat"))
```

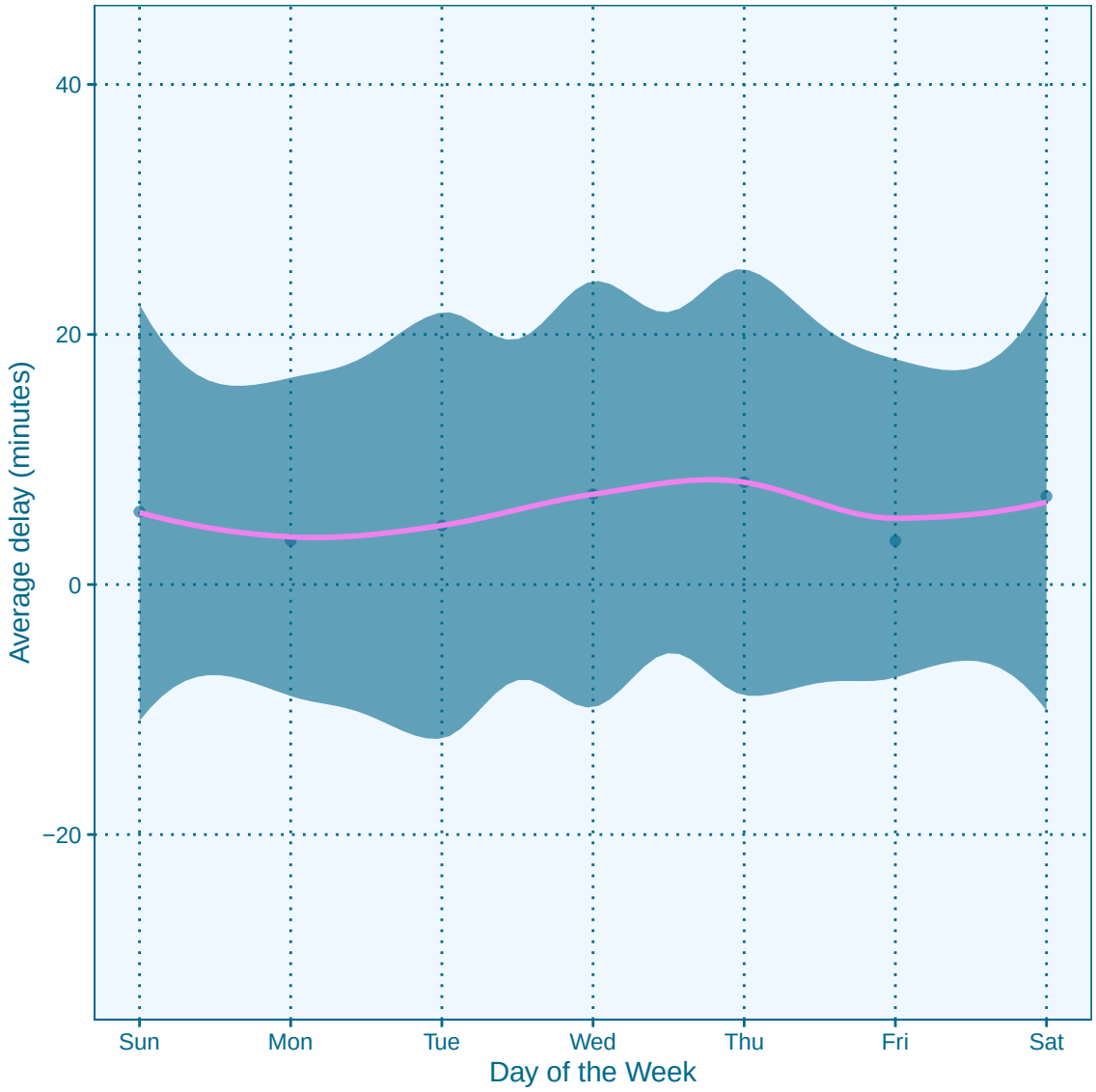

Figure 5: Average arrival delay by day of the week

Figure 5 clearly shows that day of the week is not a predictor of arrival delays.

5.7 Arrival delay by origin airport

The airport where the flight originated may influence whether or not the flight arrives late at the destination airport. This can be seen in Figure 6

```
flights_by_origin_airport <- non_validation %>% group_by(OriginAirportID) %>%
    summarize(average_delay = mean(ArrDelay), OriginAirportID = first(OriginAirportID))
```

```
arrival_delay_graph(
    x = reorder(flights_by_origin_airport$OriginAirportID,
            -flights_by_origin_airport$average_delay),
    y = flights_by_origin_airport$average_delay,
    data=flights_by_origin_airport) +
    xlab("Departing airport (arranged in descending order of delay)") +
    theme(axis.text.x=element_blank(),
    axis.ticks.x=element_blank(),
    panel.grid.major.x =  element_line(colour = "white", linetype = 3, size = 0.5),
    panel.grid.minor.x = element_blank())
```

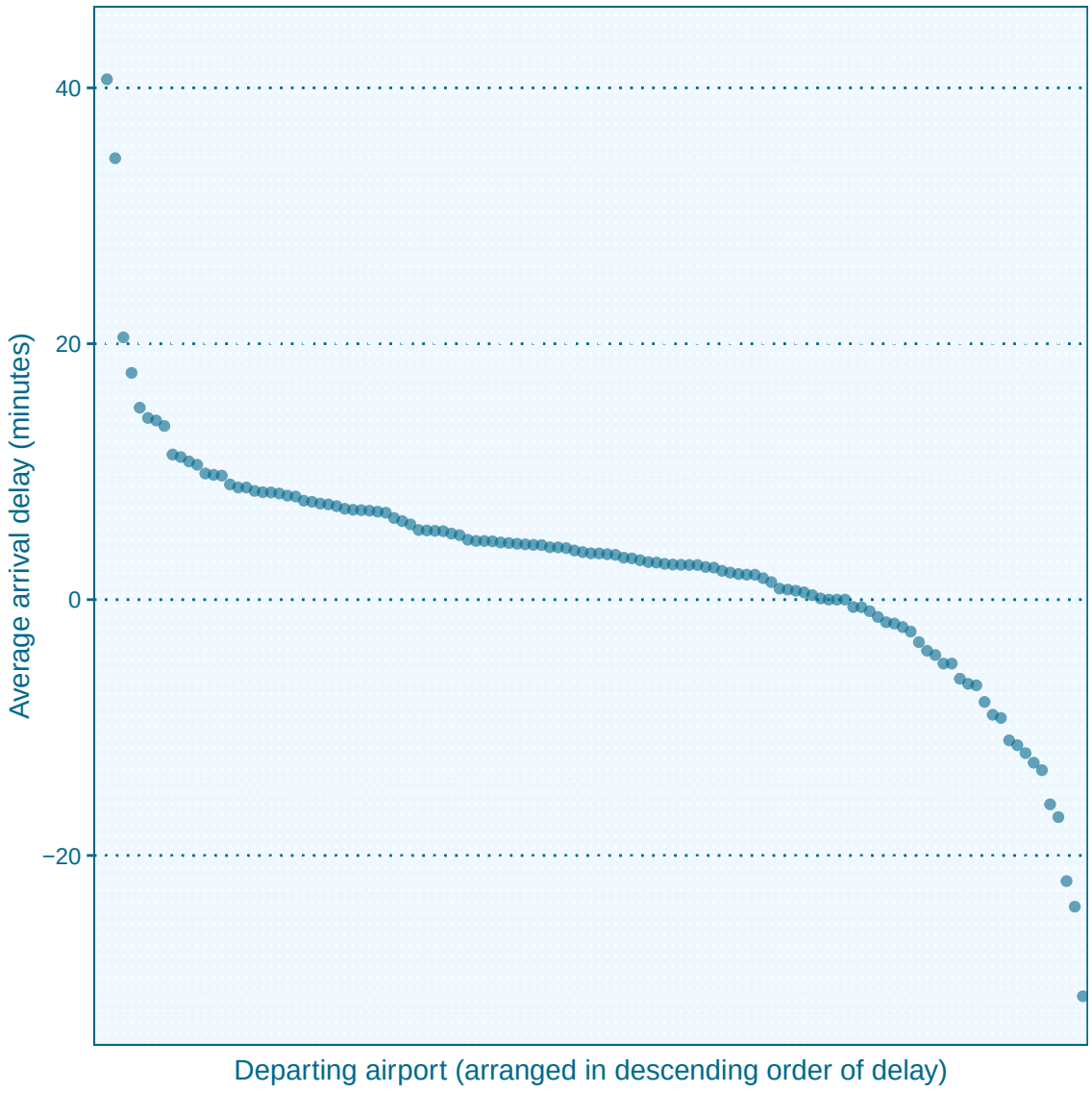

Figure 6: Average arrival delay by origin airport

As can be seen in Figure 6, the airport from which a flight departs or originates could be a predictor of the arrival delay. That means the airport could be influential in determining whether or not the flight arrives on time at its destination. This could be because the airport of origin may consistently delay planes from

departing on time or perhaps the airport of origin may consistently have foggy weather.

5.8 Arrival delay by airline

The airline itself may be an influencing factor on whether or not a flight arrives on time. This relationship can be seen in Figure 7.

```
flights_by_airline <- non_validation %>% group_by(DOT_ID_Reporting_Airline) %>%
    summarize(average_delay = mean(ArrDelay),
        DOT_ID_Reporting_Airline = first(DOT_ID_Reporting_Airline))
arrival_delay_graph(
    x = reorder(flights_by_airline$DOT_ID_Reporting_Airline,
        -flights_by_airline$average_delay),
    y = flights_by_airline$average_delay,
    data=flights_by_airline) +
    xlab("Airline (arranged in descending order of delay)")  +
    theme(axis.text.x=element_blank(),
        axis.ticks.x=element_blank(),
        panel.grid.major.x =  element_line(colour = "white", linetype = 3, size = 0.5),
        panel.grid.minor.x = element_blank())
```

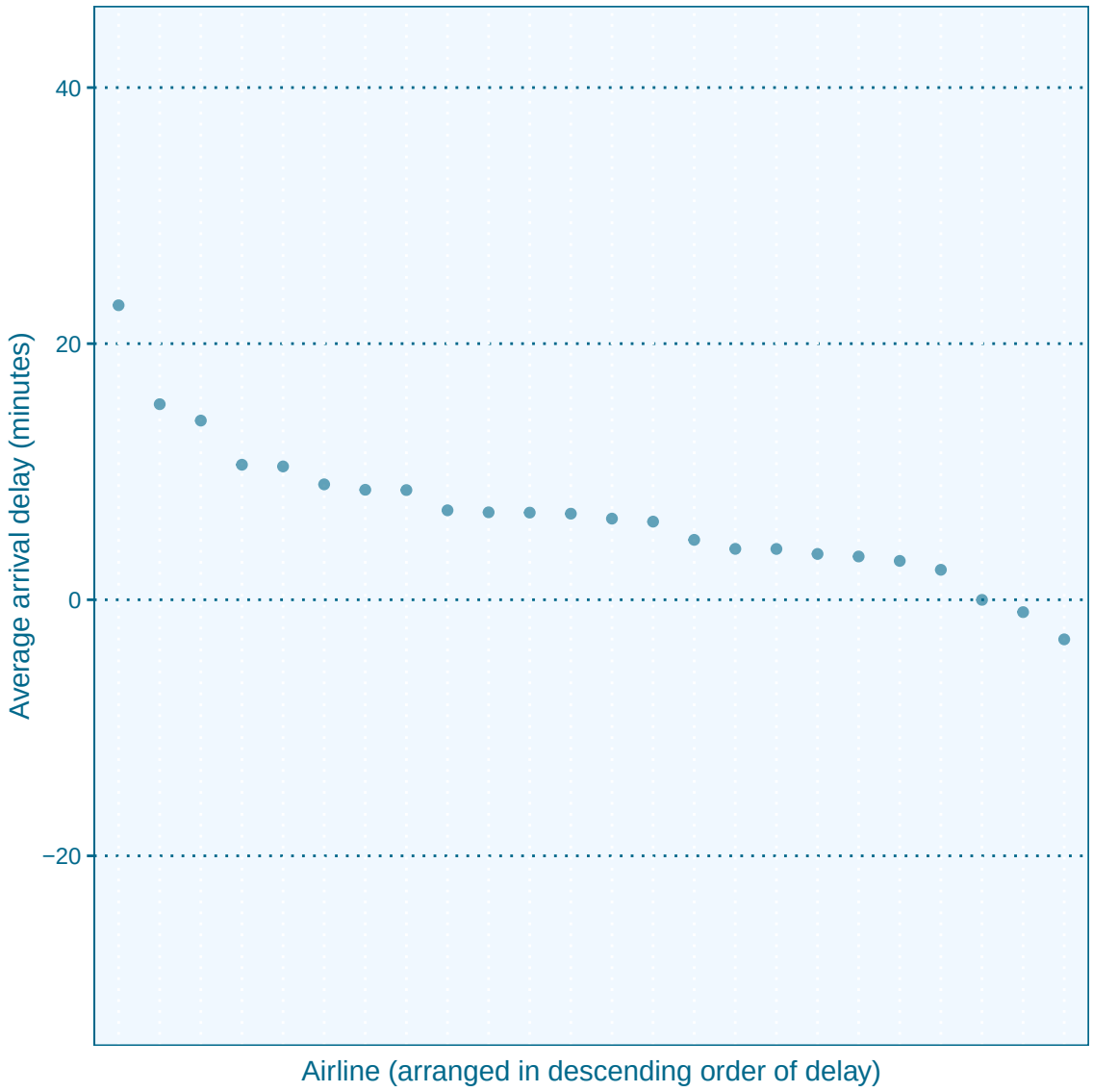

Figure 7: Average arrival delay by airline

It is clear from Figure 7 that the airline itself could be a predictor of the arrival delay. That means that the airline could be influential in determining whether or not its own flights arrive on time. This is intuitive. Some airlines pride themselves on their *on-time arrival* statistics.

5.9 Arrival delay by flight number

The flight itself may determine whether or not it is late in arriving. This could be due to the departing airport (which would be the same for all flights with the same flight number) and also to the airline itself. It could possibly be because of the crew which is assigned to this flight. This relationship can be seen in Figure 8.

```
flights_by_flight_number <- non_validation %>%
    group_by(Flight_Number_Reporting_Airline) %>%
    summarize(average_delay = mean(ArrDelay),
        Flight_Number_Reporting_Airline = first(Flight_Number_Reporting_Airline))
arrival_delay_graph(
    x = reorder(flights_by_flight_number$Flight_Number_Reporting_Airline,
```

```
    -flights_by_flight_number$average_delay),
y = flights_by_flight_number$average_delay,
data=flights_by_flight_number) +
xlab("Flight number (arranged in descending order of delay)") +
theme(axis.text.x=element_blank(),
    axis.ticks.x=element_blank(),
    panel.grid.major.x =  element_line(colour = "white", linetype = 3, size = 0.5),
    panel.grid.minor.x = element_blank())
```

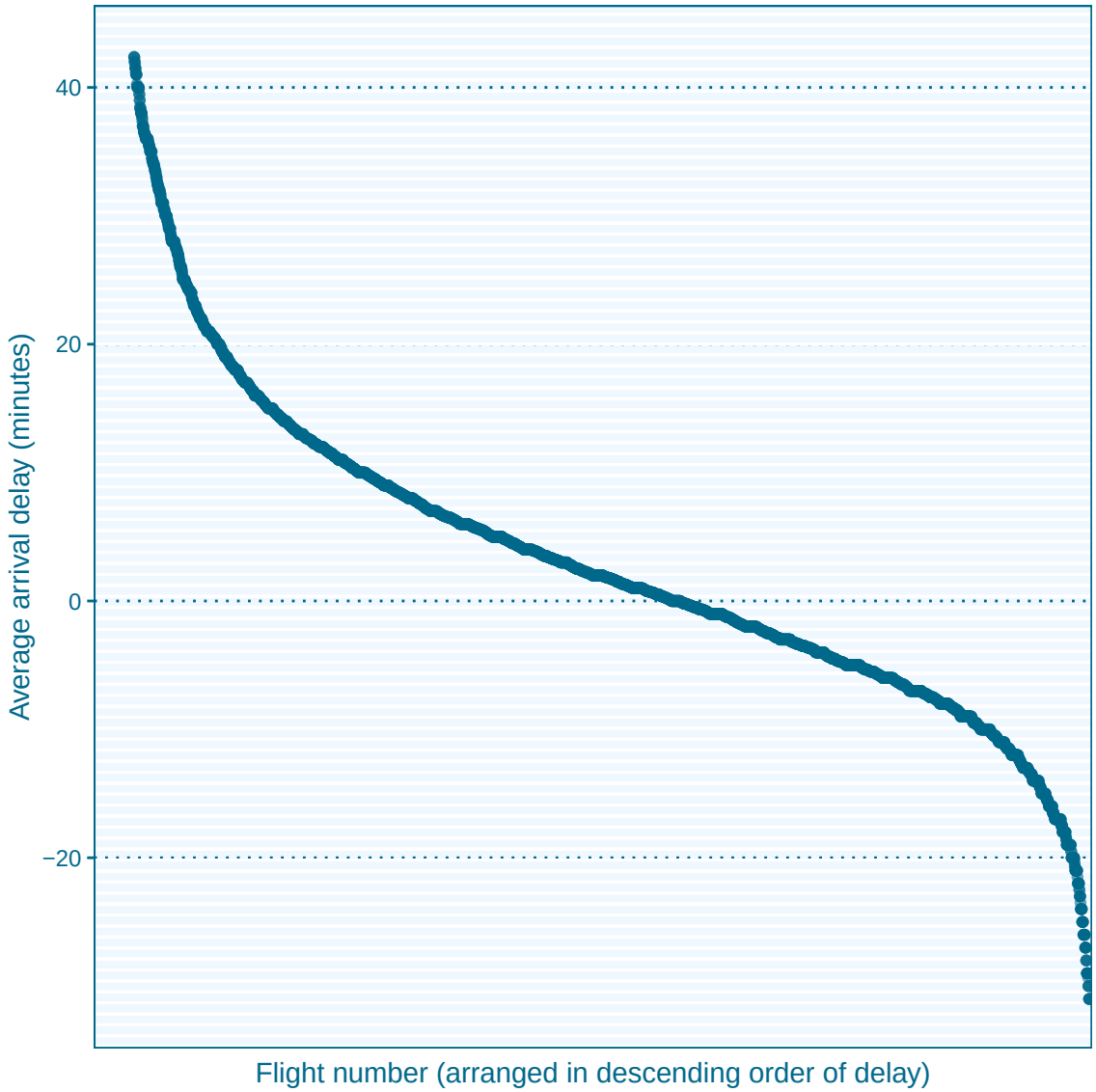

Figure 8: Average arrival delay by flight number

It is clear from Figure 8 that the scheduled flight number itself is a predictor of possible arrival delays for that flight. Care must also be taken to not allow airport of origin, airline, and flight number to over emphasize the same common fact.

5.10 Arrival delay by tail number (specific airplane)

Physical airplanes have unique tail numbers. The specific physical airplane itself may determine whether or not it is late in arriving. This may indicate that this specific physical airplane flies faster or slower than other physical airplanes. This relationship can be seen in Figure 9.

```
flights_by_tail_number <- non_validation %>% group_by(Tail_Number) %>%
    summarize(average_delay = mean(ArrDelay),
        Tail_Number = first(Tail_Number))
arrival_delay_graph(
    x = reorder(flights_by_tail_number$Tail_Number,
        -flights_by_tail_number$average_delay),
    y = flights_by_tail_number$average_delay,
    data=flights_by_tail_number) +
    xlab("Tail number (arranged in descending order of delay)") +
    theme(axis.text.x=element_blank(),
    axis.ticks.x=element_blank(),
    panel.grid.major.x =  element_line(colour = "white", linetype = 3, size = 0.5),
    panel.grid.minor.x = element_blank())
```

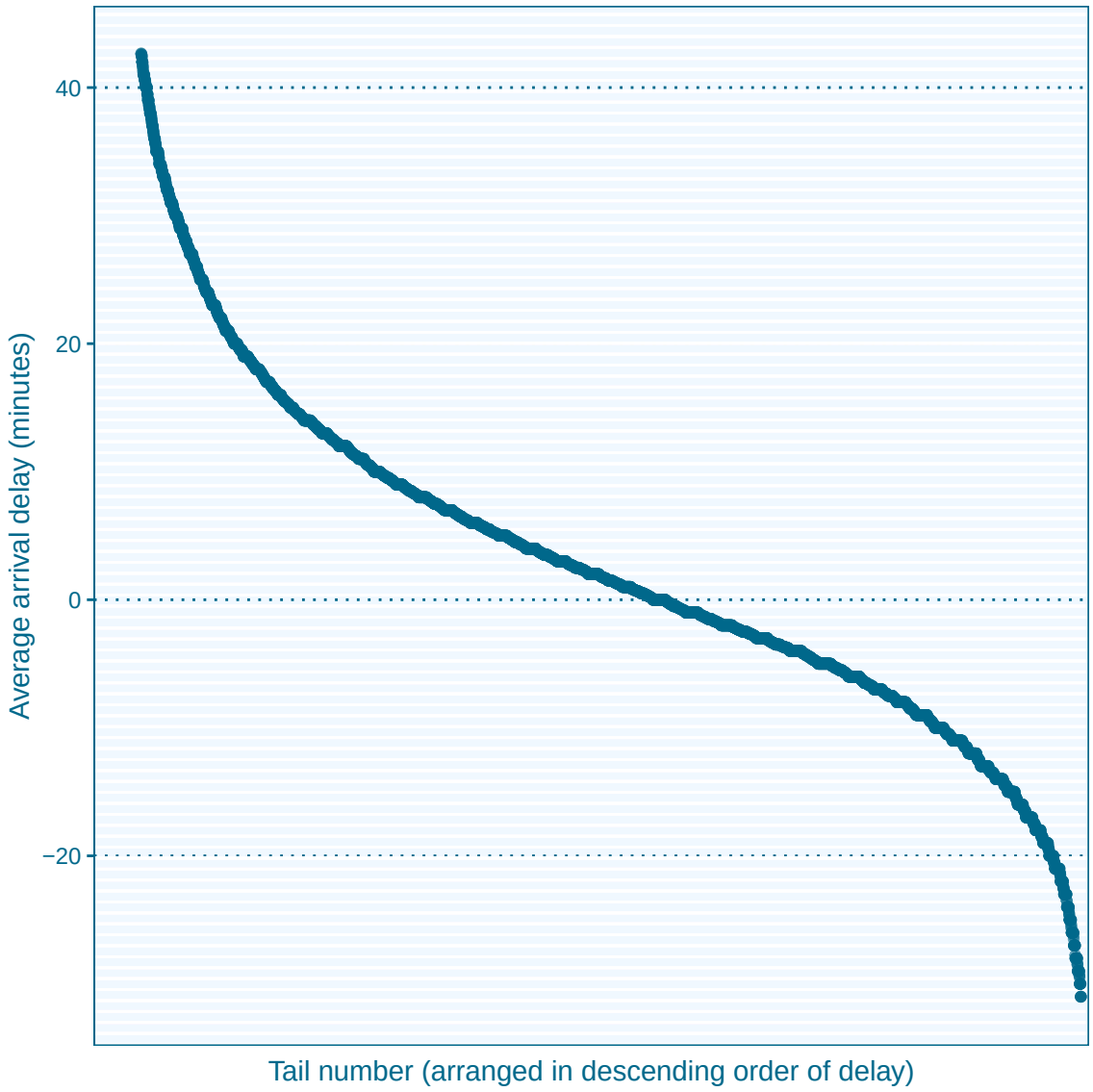

Figure 9: Average arrival delay by tail number

From Figure 9, it is clear that the tail number is a predictor of arrival delays. But care must be taken to not allow the airport of origin, airline, flight number, and tail number to over emphasize some common fact.

5.11 Arrival delay by distance flown

The distance of a flight may influence whether or not there is a delay in arrival. This relationship can be seen in Figure 10.

```
flights_by_distance <- non_validation %>% group_by(Distance) %>%
    summarize(average_delay = mean(ArrDelay),
    Distance = first(Distance))
arrival_delay_graph(
    x = flights_by_distance$Distance,
    y = flights_by_distance$average_delay,
    data=flights_by_distance) +
    xlab("Distance (miles)")
```

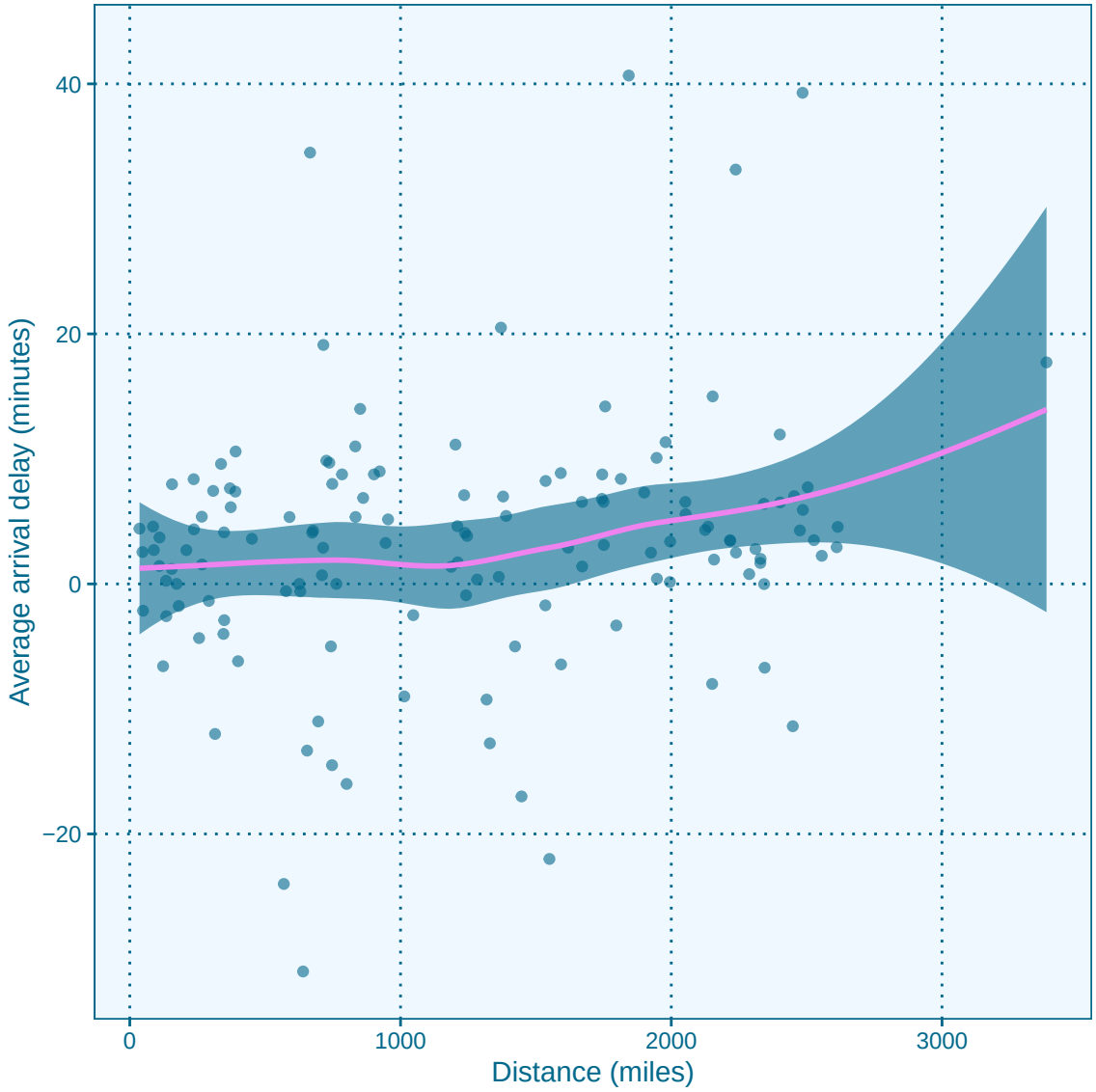

Figure 10: Average arrival delay by distance flown

From Figure 10, it can be seen that the distance flown (distance between the origin airport and the destination airport) is not a predictor of arrival delays.

5.12 Arrival delay by departure delay

It is intuitive to assume that if the flight departs late, then it will probably arrive late. This relationship can be seen in Figure 11.

```
flights_by_departure_delay <- non_validation %>% group_by(DepDelay) %>%
    summarize(average_delay = mean(ArrDelay),
    DepDelay = first(DepDelay))
arrival_delay_graph(
    x = flights_by_departure_delay$DepDelay,
    y = flights_by_departure_delay$average_delay,
```

```
    data=flights_by_departure_delay) +
    xlab("Departure delay (minutes)") +
    scale_x_continuous(
        limits=c(mean(non_validation$ArrDelay) - sd(non_validation$ArrDelay),
            mean(non_validation$ArrDelay) + sd(non_validation$ArrDelay)))
```

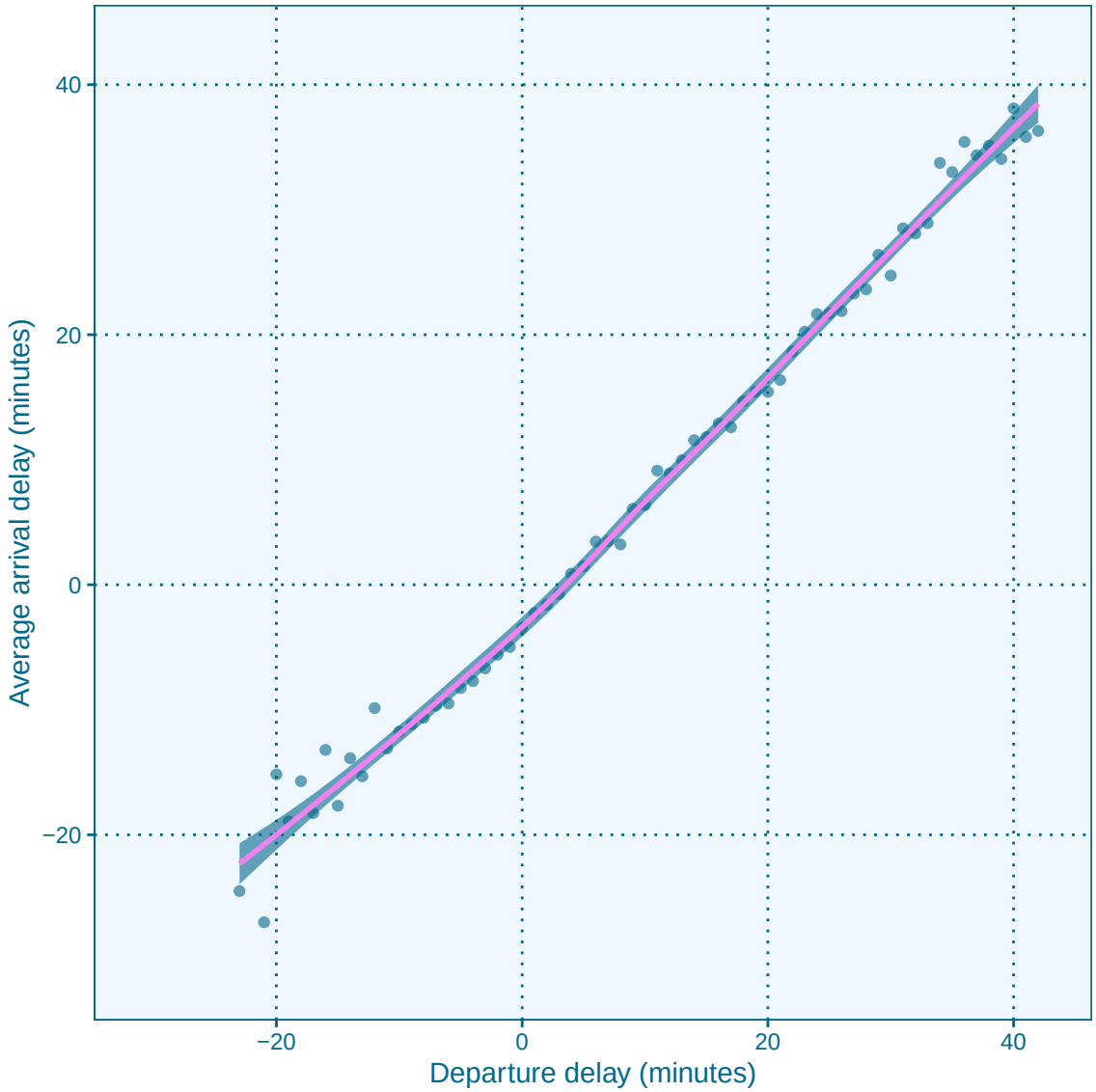

Figure 11: Average arrival delay by departure delay

The horizontal and vertical scales on Figure 11 are the same showing a near linear relationship between departure delay and arrival delay. Although beyond the scope of this research, this leads the author to believe that perhaps two models could be created: 1) one that attempts make predictions before the flight departs and 2) and one that attempts to make predictions immediately after the flight departs taking into account any departure delays.

The limits of the horizontal and vertical axes can be changed to include all arrival delays and departure delays. This can be seen in Figure 12.

```
arrival_delay_graph(
    x = flights_by_departure_delay$DepDelay,
    y = flights_by_departure_delay$average_delay,
    data=flights_by_departure_delay) +
    xlab("Departure delay (minutes)") +
    scale_y_continuous(
        limits=c(min(non_validation$ArrDelay),
            max(non_validation$ArrDelay)))
```

```
## Scale for 'y' is already present. Adding another scale for 'y', which will
## replace the existing scale.
```

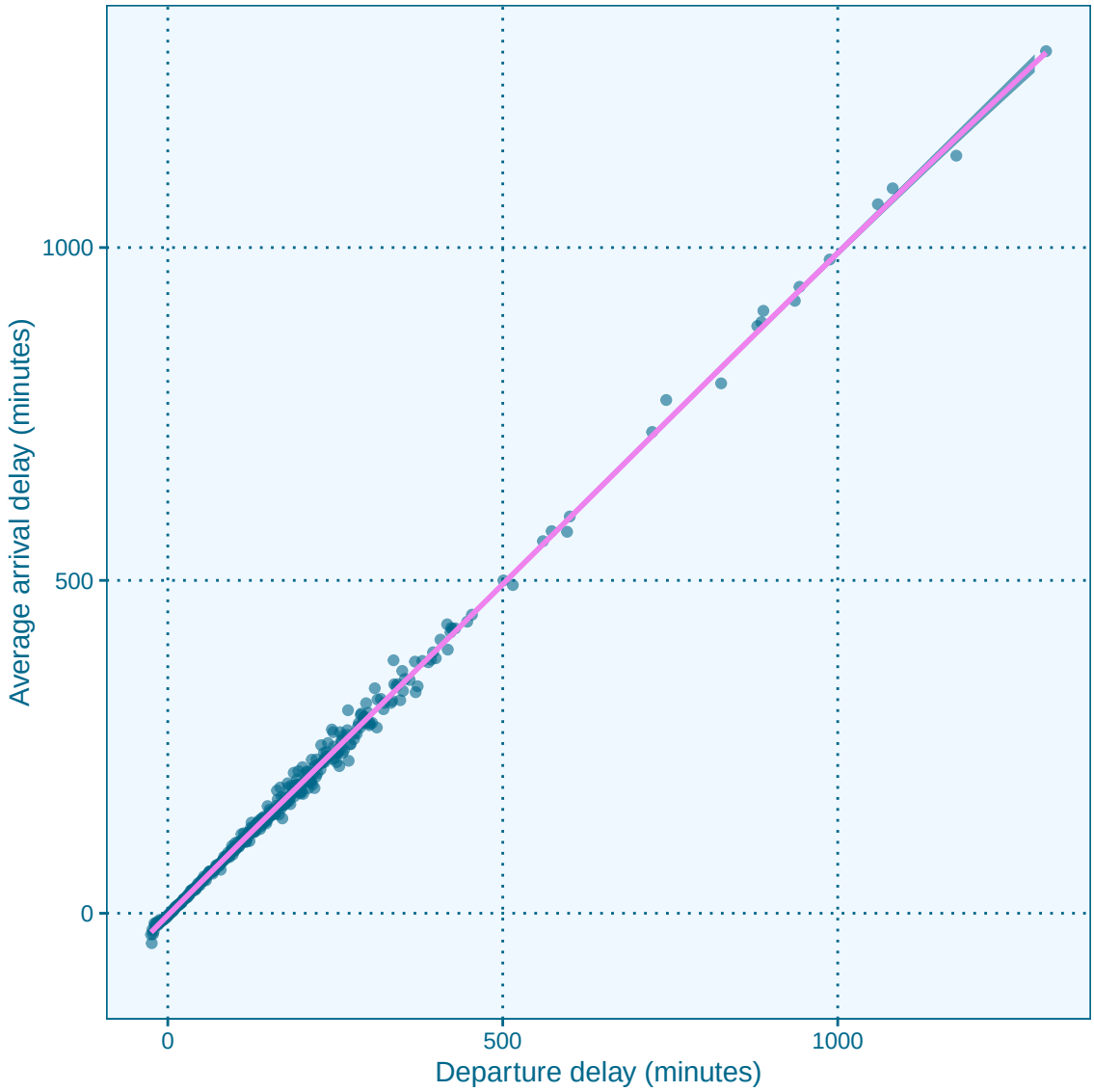

Figure 12: Average arrival delay by departure delay extended times

```
    scale_x_continuous(
        limits=c(min(non_validation$DepDelay),
            max(non_validation$DepDelay)))
```

```
## <ScaleContinuousPosition>
##  Range:
##  Limits:  -25 -- 1.31e+03
```

Figure 12 shows that even with flights which had the maximum arrival delays (1295 minutes), the linear relationship exists.

5.13 Arrival delay by time flying

The duration of a flight (in minutes) may influence whether or not there is a delay in arrival. This relationship can be seen in Figure 13.

```
flights_by_airtime <- non_validation %>% group_by(AirTime) %>%
    summarize(average_delay = mean(ArrDelay),
AirTime = first(Distance))
    arrival_delay_graph(
    x = flights_by_airtime$AirTime,
    y = flights_by_airtime$average_delay,
    data=flights_by_airtime) +
    xlab("Duration (minutes)")
```

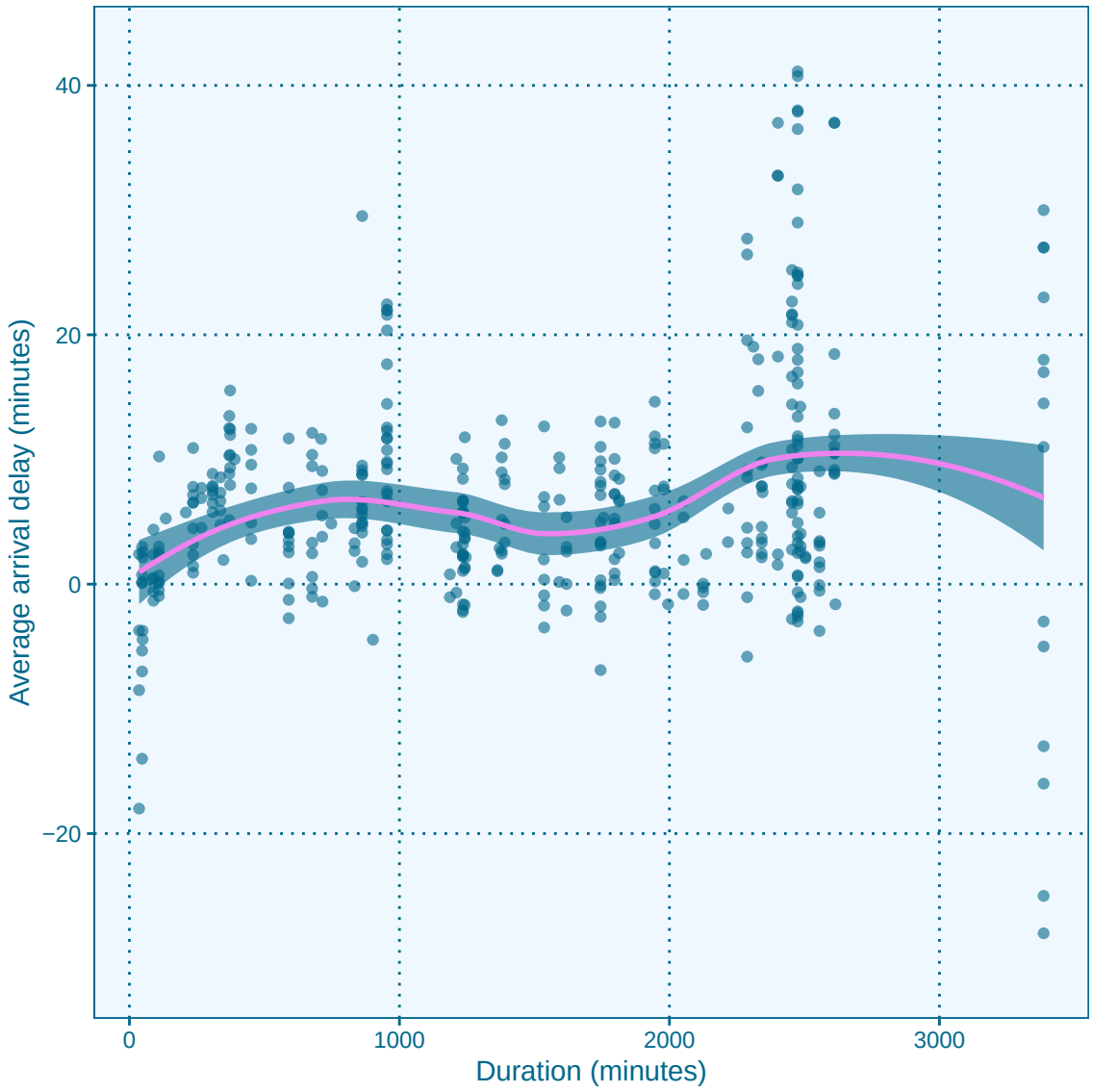

Figure 13: Average arrival delay by airtime flown

Figure 13 shows that the time flying is not a predictor of arrival delays.

5.14 Arrival delay by arrival time

The arrival time of the flight (when it should have arrived at the destination airport) could be influential in predicting arrival delays. This relationship can be seen in Figure 14

```
flights_by_arrival_time <- non_validation %>% group_by(arrival_slot) %>%
    summarize(average_delay = mean(ArrDelay),
    arrival_slot = first(arrival_slot))
arrival_delay_graph(
    x = flights_by_arrival_time$arrival_slot,
    y = flights_by_arrival_time$average_delay,
    data=flights_by_arrival_time) +
    xlab("Arrival Time") +
```

```
    scale_x_continuous(limits=c(0, 2500), breaks=seq(0, 2400, 100))+
    theme(axis.text.x = element_text(angle = 270, vjust = 0.5, hjust=1))
```

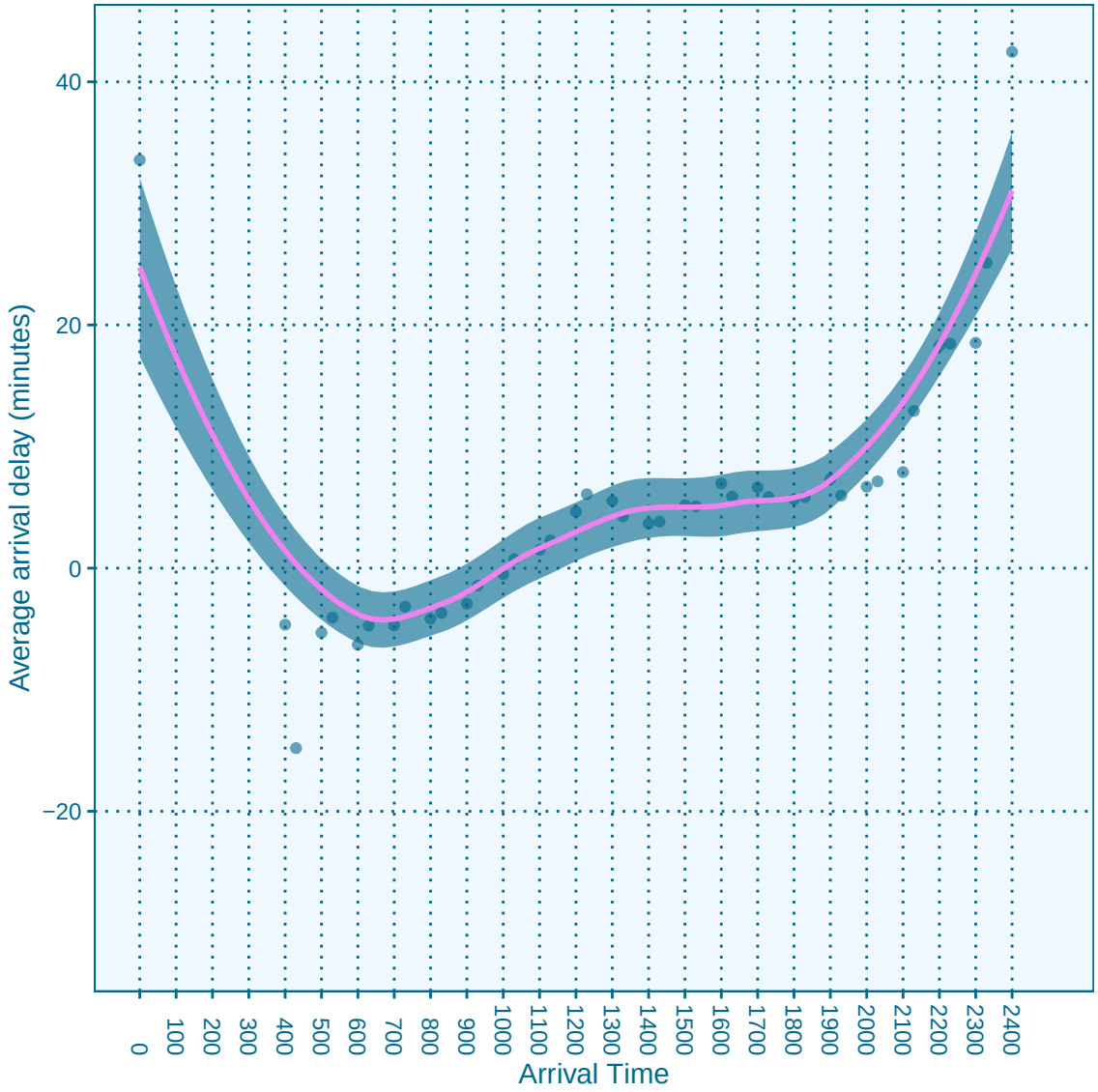

Figure 14: Average arrival delay by arrival time

Figure 14 appears to show an increase in arrival delays between approximately 21:00 and 4:00. This could be important. However, in analysing arrival times, the author could not find any documentation on whether the arrival time was in local time (time at the destination airport - LAX in this case) or whether the time was in GMT which is typical of aviation flight plans.

To determine this, the author needed to refer back to the original data sources. Combining the origin airport, destination airport along with the departure time, arrival time, and airtime (flight time), one could find, for example, a flight from Chicago, Illinois, to Los Angeles, California, with information as can be seen in Table 3

Typical flying time from Chicago to Los Angeles is 4.5 hours or 270 minutes. In the example displayed in Table 3, the total *AirTime* is in the vicinity of 270 minutes. But if you subtract the *DepTime* from the

Table 3: Example Flight Chicago to Los Angeles

Field	Value
Origin	ORD
OriginCityName	Chicago, IL
DepTime	1902
Dest	LAX
DestCityName	Los Angeles, CA
ArrTime	2048
AirTime	208

ArrTime, there is only approximately 3 hours. Therefore, it is appropriate to assume that the fields *DepTime* and *ArrTime* refer to the local time at the airport in question.

Therefore Figure 14 shows an increase in arrival delays during the evening, night, and early morning hours.

5.15 Investigating *late*

This sub-section specifically looks at the field *late* which is one if the flight was late (arrived later than scheduled) and was zero otherwise. This field did not cater for earlier arrivals. Statistics about the *late* field can be seen in Table 4

Table 4: Statistics about late arrivals

Measurement	Value
Total Rows	48516.00
Late	21104.00
% Late	0.43
Not Late	27412.00
% Not Late	0.57

5.16 Summary

From this exploration, it is assumed that data fields which predict a delay in arrival times also predict whether or not a flight will be late.

Table 5: Important columns in dataset

Fields
Airport of Origin
Airline
Flight Number
Tail Number
Arrival Time
Departure Delay

From the aforegoing, it is clear that the fields indicated in Table 5 are predictors of arrival delays and could be used to predict if a flight is late or not. From these fields, two were omitted from the models which were investigated. The flight number provides the same type of information as the airport of origin with the

possible loss of information about the actual flight crew involved in the flight. The tail number identifies the physical plane being used.

The reason these two fields were omitted is explained in more detail in Sub-Section 6.1.

6 Model Investigations and Results

This section discusses the various models which were developed and the results which were obtained. This section deals primarily with the *train* dataset and the *test* dataset. The *validation* dataset is not used in this section except at the end to test the final model. That is described in Sub-Section 6.4.

There were four iterations of model development.

6.1 Iteration 1 - Superficial loop through a number of models

The first attempt to find a model to predict whether or not a flight will be late involved a superficial run through a handful of different models using default tuning parameters. Four different models were chosen because they were *structurally* different. The four different models attempted to solve the prediction problem in completely different ways.

The first attempt at Iteration 1 was to include all the fields itemized in Table 5. For the first attempt, the notebook used for this research had 4GB of swapspace. The building of the *rpart* model crashed. The author increased swapspace to 8GB. The *rpart* model built for a substantially longer amount of additional time but still crashed. The author increased the swapspace again to 16GB and the model still crashed after an even longer period of time. On investigation the tail number included 8121 unique values and the flight number included 4855 unique values. The author believes that the number of factors was too high for the compute environment available to the author. Those two fields were removed.

The four models chosen for Iteration 1 were *glm* (generalized linear model), *rpart* (a decision tree), *mlp* (a multilayer perceptron), and *knn* (K nearest neighbour). These were run using default tuning parameters using two iterations each. This gave the author an indication of which model might be the most effective.

This investigation took some time. This section of code to execute the investigation in *this* document is not *executable* but is presented for reading purposes only. The code is, of course, executable on the R script file presented with this document.

```
#
# determine if we must rebuild the model for each iteration
# and associated confusion matrix
must_rebuild <- function(model_rda_filename, cm_rda_filename ) {
        (!file.exists(model_rda_filename)) ||
        (!file.exists(cm_rda_filename)) ||
        (file.info(model_rda_filename)$ctime < file.info(train_rda)$ctime) ||
        (file.info(cm_rda_filename)$ctime < file.info(train_rda)$ctime)
}

if ( build_late_models_iteration_1 ) {
        load(train_rda)
        load(test_rda)
        models <- c("rpart", "glm", "mlp")              #  knn never returned
        messages <- c(
                "rpart took just over 2 minutes to build in author's notebook",
                "glm took just over 5 minutes to build on author's notebook",
                "mlp took nearly 14 minutes to build on author's notebook")
        for(i in 1:length(models) ) {
                model <- models[i]
                model_rda_filename <- paste(model, "_1.rda", sep="")
                cm_rda_filename <- paste(model, "_1_cm.rda", sep="")
                time_txt_filename <- paste(model, "_1.txt", sep="")
                if ( must_rebuild(model_rda_filename, cm_rda_filename ) ) {
                        #
                        # supress the warning about sample.kind
```

```
                        #
                        warnLevel <- getOption("warn")
                        options(warn = -1)
                        set.seed(1, sample.kind="Rounding")
                        options(warn = warnLevel)
                        print(model)
                        print(messages[i])
                        print("Please be patient")
                        begTime <- Sys.time()
                        print(begTime)
                        fit <- train(
                                late ~ airline + origin_airport + ArrTime + DepDelay,
                                method=model, data=train,
                                tuneLength = 2)
                        endTime <- Sys.time()
                        print(endTime)
                        cat(difftime(endTime, begTime, units="mins"),
                file = time_txt_filename, sep="\n")
                        print(fit)
                        save(fit, file=model_rda_filename)
                        predictions <- predict(fit, test)
                        print(paste("length predictions", length(predictions)))
                        print(paste("length test$late", length(test$late)))
                        cm <- confusionMatrix(data = predictions, reference = test$late)
                        save(cm, file=cm_rda_filename)
                        print(cm)
                }
        }
        rm(test, train)
}
```

The *knn* model did not complete after 12 hours. The results for the other three models are presented in Table 6.

Table 6: Iteration 1 model results

Model	BestParameter	Accuracy	Sensitivity	Specificity
rpart	cp = 0.000895066603485495	0.7589120	0.9328957	0.5329228
glm	NA	0.7622089	0.9124726	0.5670298
mlp	size = 1	0.5650113	1.0000000	0.0000000

This information showed the author that the *rpart* model and *glm* model performed similarly. Better results might be obtained if actual tuning parameters were given - especially in the case of the *mlp* model.

6.2 Iteration 2 - Loop through models with tuning parameters

The second iteration on attempting to predict whether or not a flight was late used tuning parameters for the four models. In the case of the *rpart* model and the *mlp* model, parameters in the vicinity of the parameters generated in the first iteration were used. In the case of the *knn* model, the author chose a range of values to try. Again, the *knn* model did not return after 12 hours.

Since the *glm* model does not have tuning parameters, that model was not rebuilt but the results of Iteration 1 will be used in the comparisons with Iteration 2.

This investigation took some time and the code to execute this is not *executable* in this document but is

presented for reading purposes only. The code is, of course, executable on the R script file presented with this document.

```
if ( build_late_models_iteration_2 ) {
        load(train_rda)
        load(test_rda)
        #
        # models and tuning_parameters must be in same order.
        # NAs for models with no tuning parameters
        #
        models <- c("rpart", "mlp")
        tuning_parameters <- c(
                        data.frame(cp = seq(.0005, .00100, 0.0001)),    # rpart
                        data.frame(size = seq(1, 5, 2)))                # mlp
        messages <- c("rpart took just under 2 minutes to build on the author's notebook",
                      "mlp took up to 25 minutes to build on the author's notebook")
        for(i in 1:length(models) ) {
                model <- models[i]
                model_rda_filename <- paste(model, "_2.rda", sep="")
                cm_rda_filename <- paste(model, "_2_cm.rda", sep="")
                time_txt_filename <- paste(model, "_2.txt", sep="")
                if ( must_rebuild(model_rda_filename, cm_rda_filename ) ) {
                        #
                        # supress the warning about sample.kind
                        #
                        warnLevel <- getOption("warn")
                        options(warn = -1)
                        set.seed(1, sample.kind="Rounding")
                        options(warn = warnLevel)
                        print(model)
                        print(messages[i])
                        print("Please be patient")
                        begTime <- Sys.time()
                        print(begTime)
                        fit <- NA                       # scoping
                        if ( is.na(tuning_parameters[i]  ) ) {
                                print(paste("model=", model,
                    " no tuning parameters",
                    sep=" "))
                                fit <- train(
                                        late ~ airline + origin_airport + ArrTime +
                    DepDelay,
                                        method=model, data=train)
                        } else {
                                print(paste("model=", model, "tuning parameters",
                    tuning_parameters[i], sep = " "))
                                fit <- train(
                                        late ~ airline + origin_airport + ArrTime +
                    DepDelay,
                                        method=model, data=train,
                                        tuneGrid = expand.grid(tuning_parameters[i]))
                        }
                        endTime <- Sys.time()
                        print(endTime)
```

```
                        cat(difftime(endTime, begTime, units="mins"),
                file = time_txt_filename, sep="\n")
                        print(fit)
                        save(fit, file=model_rda_filename)
                        predictions <- predict(fit, test)
                        print(paste("length predictions", length(predictions)))
                        print(paste("length test$late", length(test$late)))
                        cm <- confusionMatrix(data = predictions, reference = test$late)
                        save(cm, file=cm_rda_filename)
                        print(cm)
                }
        }
        rm(test, train)
}
```

The results of Iteration 2 can be seen in Table 7.

Table 7: Iteration 2 model results

Model	BestParameter	Accuracy	Sensitivity	Specificity
rpart	cp = 6e-04	0.7626211	0.9091904	0.5722406
glm	NA	0.7622089	0.9124726	0.5670298
mlp	size = 1	0.5650113	1.0000000	0.0000000

These results informed the author that the *rpart* model and *glm* model again performed similarly. Those two models will be investigated in more detail in Iteration 3 and the *mlp* model will be abandoned.

6.3 Iteration 3 - Selection of best model and further tuning

The tuning parameters for *rpart* were expanded and compared to the one configuration for *glm*.

```
if ( build_late_models_iteration_3 ) {
        load(train_rda)
        load(test_rda)
        #
        # models and tuning_parameters must be in same order.
        # NAs for models with no tuning parameters
        #
        models <- c("rpart")
        tuning_parameters <- c(
                        data.frame(cp = seq(.0001, .00200, 0.0001)))    # rpart
        messages <- c("rpart model took two minutes to build on the author's notebook")
        for(i in 1:length(models) ) {
                model <- models[i]
                model_rda_filename <- paste(model, "_3.rda", sep="")
                cm_rda_filename <- paste(model, "_3_cm.rda", sep="")
                time_txt_filename <- paste(model, "_3.txt", sep="")
                if ( must_rebuild(model_rda_filename, cm_rda_filename ) ) {
                        #
                        # supress the warning about sample.kind
                        #
                        warnLevel <- getOption("warn")
                        options(warn = -1)
                        set.seed(1, sample.kind="Rounding")
```

```
                        options(warn = warnLevel)
                        print(model)
                        print(messages[i])
                        print("Please be patient")
                        begTime <- Sys.time()
                        print(begTime)
                        fit <- NA                       # scoping
                        if ( is.na(tuning_parameters[i]  ) ) {
                                print(paste("model=", model, " no tuning parameters",
                    sep=" "))
                                fit <- train(
                                        late ~ airline + origin_airport + ArrTime +
                        DepDelay,
                                        method=model, data=train)
                        } else {
                                print(paste("model=", model, "tuning parameters",
                    tuning_parameters[i], sep = " "))
                                fit <- train(
                                        late ~ airline + origin_airport + ArrTime +
                        DepDelay,
                                        method=model, data=train,
                                        tuneGrid = expand.grid(tuning_parameters[i]))
                        }
                        endTime <- Sys.time()
                        print(endTime)
                        cat(difftime(endTime, begTime, units="mins"),
                file = time_txt_filename, sep="\n")
                        print(fit)
                        save(fit, file=model_rda_filename)
                        predictions <- predict(fit, test)
                        print(paste("length predictions", length(predictions)))
                        print(paste("length test$late", length(test$late)))
                        cm <- confusionMatrix(data = predictions, reference = test$late)
                        save(cm, file=cm_rda_filename)
                        print(cm)
                }
        }
        rm(test, train)
}
```

The results of Iteration 3 can be seen in Table 8.

Table 8: Iteration 3 model results

Model	BestParameter	Accuracy	Sensitivity	Specificity
rpart	cp = 6e-04	0.7626211	0.9091904	0.5722406
glm	NA	0.7622089	0.9124726	0.5670298

Table 8 shows that the *rpart* model with the appropriate parameters outperforms the *glm* model.

6.4 Iteration 4 - Final model with *validation* dataset

Iteration 3 shows that the *rpart* algorithm with a *cp* value of 6e-04 provided the best accuracy values. That selection of model and tuning parameters are used in Iteration 4 with the *non_validation* set (combination of

the *train* set and *test* set) to create the final model which will be tested against the *validation* set.

```
if ( build_late_models_iteration_4 ) {
        #
        # now the entire non_validation data set (which comprises train and test)
        # will be used to train the rpart model and the validation set will be
        # used to validate it
        #
        load(non_validation_rda)
        load(validation_rda)
        #
        # need to get the best complexity value
        #
        load("rpart_3.rda")
        rpart_best_cp <- fit$bestTune
        model_rda_filename <- paste(model, "_4.rda", sep="")
        cm_rda_filename <- paste(model, "_4_cm.rda", sep="")
        time_txt_filename <- paste(model, "_4.txt", sep="")
        if ( must_rebuild(model_rda_filename, cm_rda_filename ) ) {
                #
                # supress the warning about sample.kind
                #
                warnLevel <- getOption("warn")
                options(warn = -1)
                set.seed(1, sample.kind="Rounding")
                options(warn = warnLevel)
                print("rpart model took two minutes to build on the author's notebook")
                print("Please be patient")
                begTime <- Sys.time()
                print(begTime)
                fit <- train(
                        late ~ airline + origin_airport + ArrTime + DepDelay,
                        method="rpart", data=non_validation,
                        tuneGrid = data.frame(cp = rpart_best_cp))
                endTime <- Sys.time()
                print(endTime)
                cat(difftime(endTime, begTime, units="mins"),
            file = time_txt_filename, sep="\n")
                print(fit)
                save(fit, file=model_rda_filename)
                predictions <- predict(fit, validation)
                print(paste("length predictions", length(predictions)))
                print(paste("length validation$late", length(validation$late)))
                cm <- confusionMatrix(data = predictions, reference = validation$late)
                save(cm, file=cm_rda_filename)
                print(cm)
        }
}
```

The results of Iteration 4 can be seen in Table 9.

6.5 Summary of Results

Four iterations were done in an attempt to find the best model for predicting whether or not a domestic flight would arrive late into LAX airport.

Table 9: Iteration 4 model results

	Model	BestParameter	Accuracy	Sensitivity	Specificity
Accuracy	rpart	cp = 6e-04	0.7597848	0.8962574	0.582516

The best model was *rpart* with a *cp* parameter with the value 6e-04. This final model achieved an accuracy of 0.7597848 with a Sensitivity of 0.8962574 and a Specificity of 0.582516.

The confusion matrix can be seen in Table 10.

Table 10: Confusion matrix Iteration 4 model results

	0	1
0	2730	979
1	316	1366

This confusion matrix could be used to guide an Iteration 5 in the future.

7 Conclusion

This project asked the question

Can predictions be made as to whether or not a flight will be late?

and the project answers the question with the answer

Yes, it is possible to predict whether or not a flight will be late.

Data which was originally supplied by the US Department of Transportation was used for this project. A subset of records dealing only with domestic arrivals into LAX (Los Angeles airport) were extracted.

The project experienced memory problems in the compute environment which was used. Even with increasing the swap space to 16GB, some of the models could not be built.

However, despite the memory limitations, the final model achieved an accuracy of 0.7597848 with a Sensitivity of 0.8962574 and a Specificity of 0.582516. This accuracy value was substantially higher than the percentge of late flights which are in the *validation* set which was 0.4349842 which could be achieved by just guessing all flights would be late or all flights would be on-time.

It is firmly believed that if a server or virtual machine were sourced for this project (and not a notebook), better results could be obtained.

Acknowledgements and credits

The author would like to thank Professor Irizarry and his team for creating a compelling and effective course. The exercises drew a person into the problem and enticed them to investigate the data. Thank you.

The image of LAX airport is from WikiMedia and is released with a Creative Commons Attribution-Share Alike 3.0 license.

References

BTS, 2021, Bureau of Transportation Statistics, Accessed: https://www.transtats.bts.gov/DL_SelectFields.asp?gnoyr_VQ=FGJ&QO_fu146_anzr=b0-gvzr

IATA, 2017, Value of Air Transport in the United States, Accessed: https://www.iata.org/contentassets/cf15de08044e49b6b6e1ebf8a3513ed4/economic-studies-usa-eng-final.pdf

IATA, 2020, Economic Performance of the Airline Industry, Accessed: https://www.iata.org/en/iata-repository/publications/economic-reports/airline-industry-economic-performance---november-2020---report/

IBM, 2021, Airline Reporting Carrier On-Time Performance Dataset, Accessed: https://developer.ibm.com/technologies/artificial-intelligence/data/airline/

Irizarry, R.A., 2021, Introduction to Data Science.

Knuth, D.E., 1984. Literate programming. The Computer Journal, 27(2), pp.97-111.

Ljubuncic, I., 2015. Problem-solving in High Performance Computing: A Situational Awareness Approach with Linux. Morgan Kaufmann.

Love, R., 2013. Linux system programming: talking directly to the kernel and C library. O'Reilly Media, Inc.

OAG, 2020, Official Aviation Guide, Accessed: https://www.oag.com/monthly-on-time-performance-reports