STA 380 Homework 2

Summer Wang, Yawen Ye, Wenduo Wang

August 15, 2016

Flights at ABIA

In this dataset, we want to answer serveral questions: 1: Which carrier has the worst punctuality in departuring time and arriving time?Which carrier has the best punctuality in departuring time and arriving time? 2: Which carrier has done the best job to minimize CarrierDelay? 3: What is the best time of day to fly to minimize delays? 4: What is the best time of year to fly to minimize delays? 5: How do patterns of flights to different destinations or parts of the country change over the course of the year?

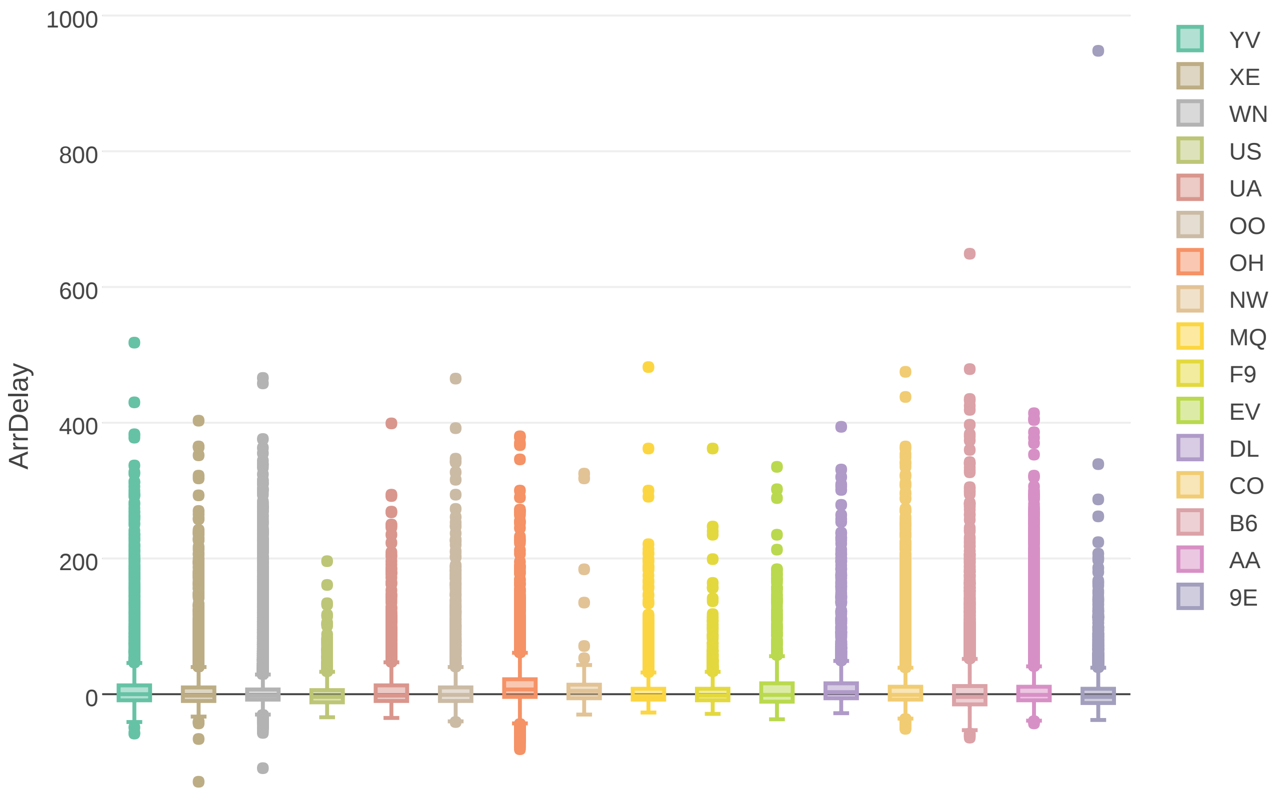
We will talk about each problem one by one.

1: Which carrier has the worst punctuality in departuring time and arriving time?

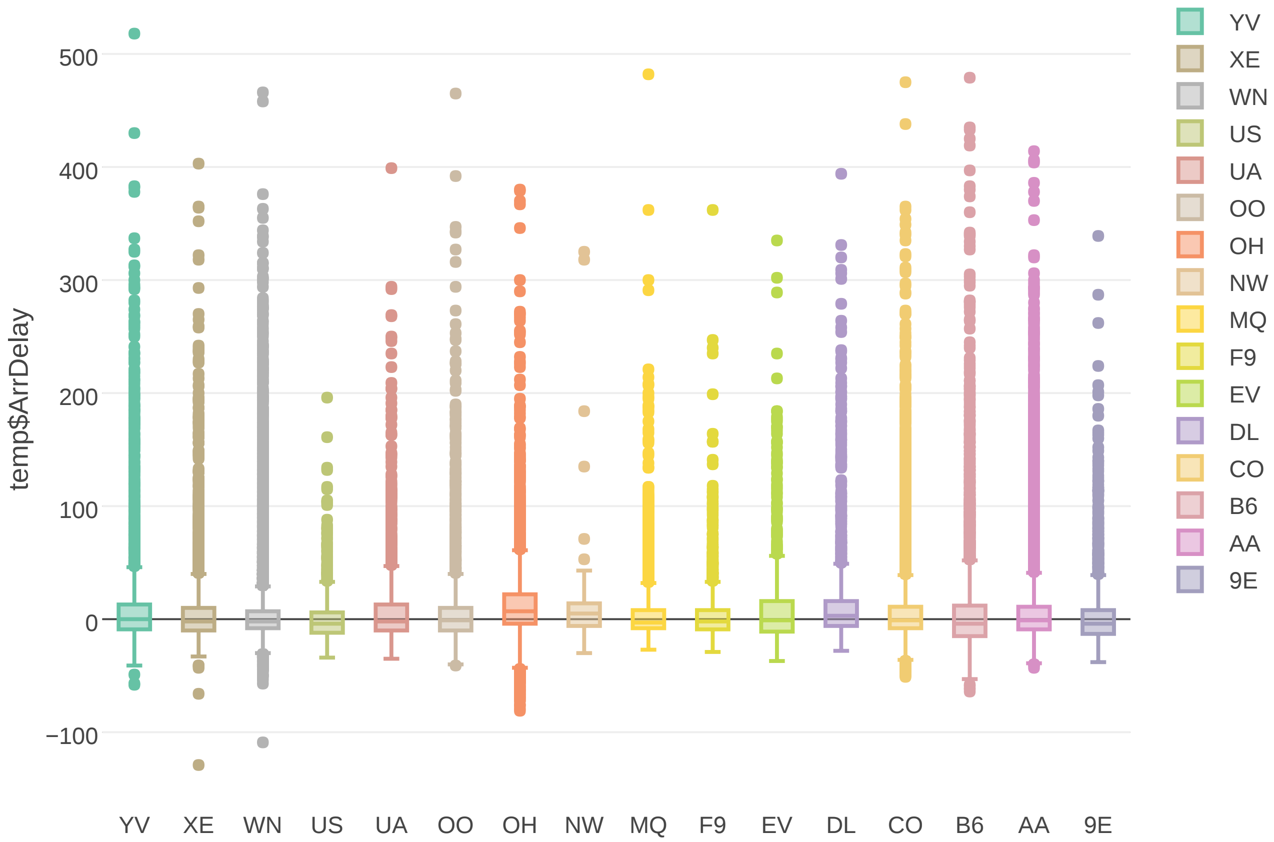
Which carrier has the best punctuality in departuring time and arriving time?

```
ABIA <-read.csv("data/ABIA.csv", header=T, na.strings=c("","NA"))
attach(ABIA)
library(dplyr)
library(plotly)
```

```
p1 <- plot_ly(ggplot2::diamonds, y = ArrDelay, color = UniqueCarrier, type = "box")
p1
```

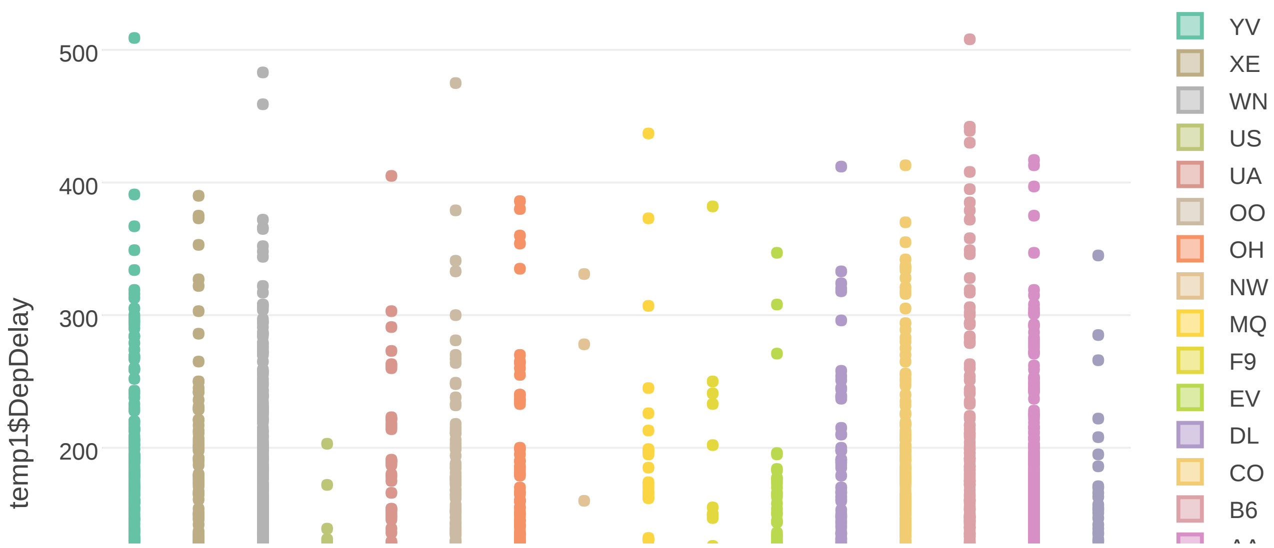


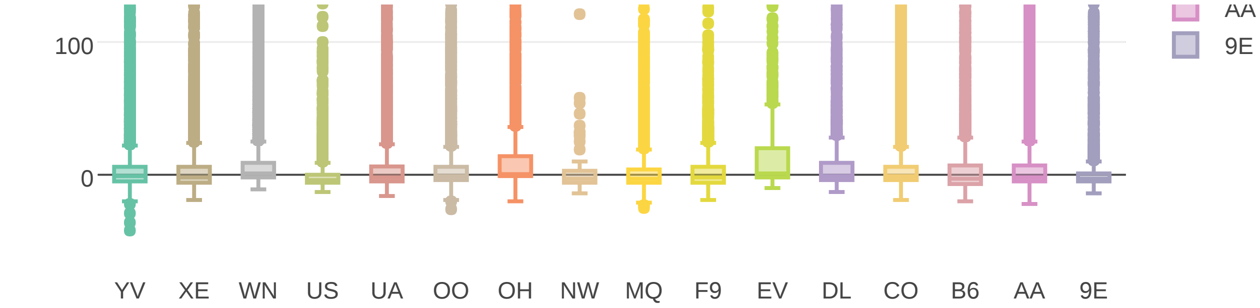

```
temp<- ABIA[which(ArrDelay!=649 & ArrDelay!= 948),]
p2 <- plot_ly(ggplot2::diamonds, y = temp$ArrDelay, color = temp$UniqueCarrier, type =
"box")
p2
```



We could actually see that OH flight is the worst when it comes to punctuality on arriving.

```
temp1<- ABIA[which(DepDelay!=665 & DepDelay!= 875),]
p3 <- plot_ly(ggplot2::diamonds, y = temp1$DepDelay, color = temp1$UniqueCarrier, type
= "box")
p3
```





We could actually see that OH and EV flight are the worst when it comes to punctuality on departing. US airline did a much better job on departure punctuality

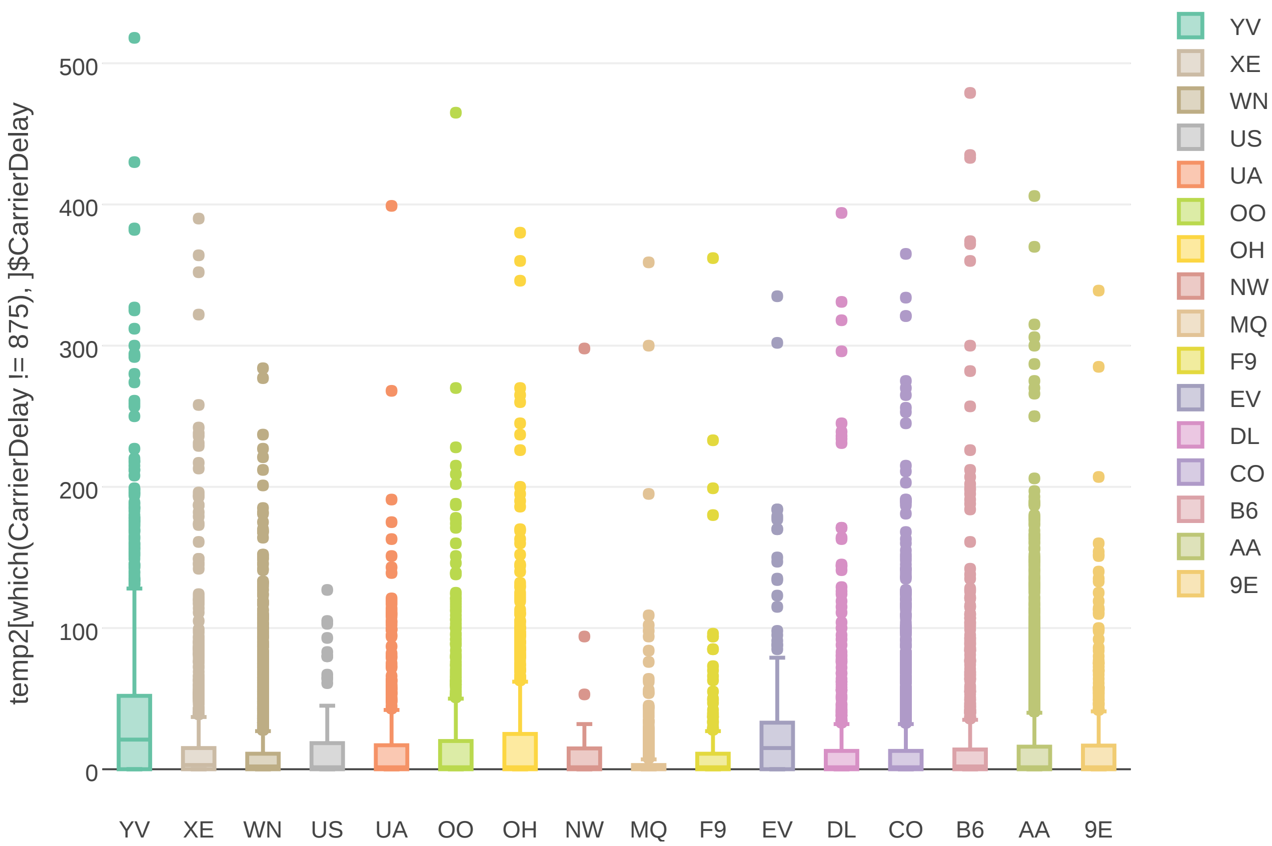
2: Which carrier has done the best job to minimize CarrierDelay?

```
temp2<-ABIA[!is.na(CarrierDelay),]
attach(temp2)
```

```
#we could see that ArrDelay consist of five parts:
sum(temp2[,'ArrDelay']!=temp2[,'WeatherDelay']+temp2[,'NASDelay']+temp2[,'SecurityDela
y']+ temp2[,'LateAircraftDelay']+temp2[,'CarrierDelay'])
```

```
## [1] 0
```

```
#let's get rid of the WeatherDelay, which people can not control
temp2[,'ArrDelay_control']=temp2[,'NASDelay']+temp2[,'SecurityDelay']+
                           temp2[,'LateAircraftDelay']+temp2[,'CarrierDelay']
p4 <- plot_ly(ggplot2::diamonds, y = temp2[which(CarrierDelay!=875),]$CarrierDelay,
        color = temp2[which(CarrierDelay!=875),]$UniqueCarrier, type = "box")
p4
```

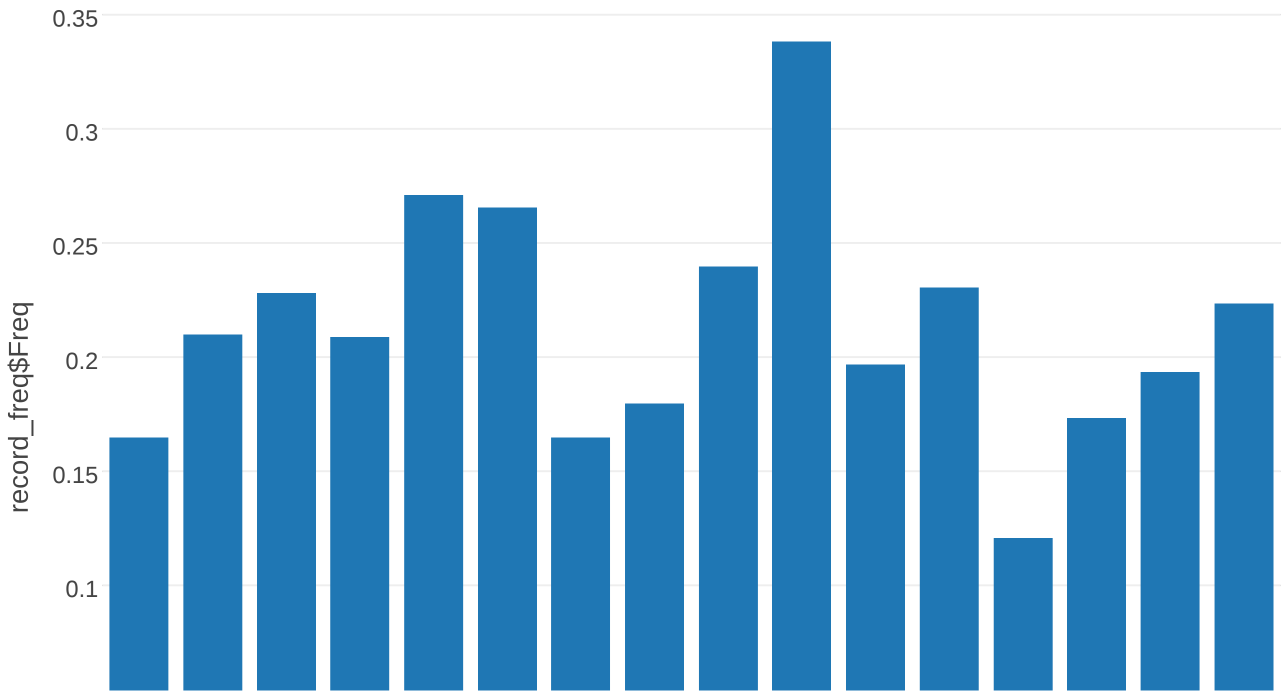


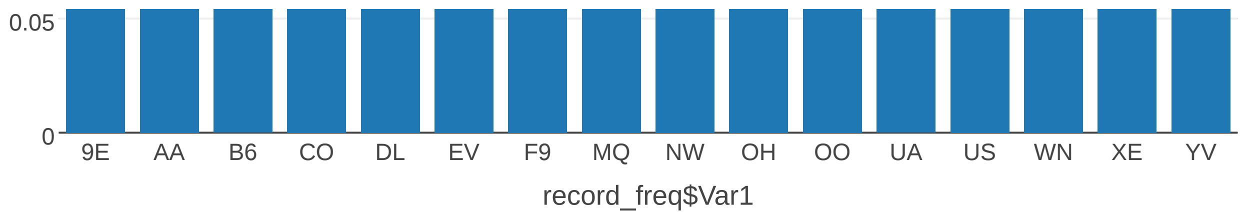
We could see that airline MQ did the best job in term of its own control of arriving time. Airline YV did a worest jobin term of its own control of arriving time. But this is actually part of the total list. So maybe, airline didn't accord many records. So let's take a look at the records of each airline.

```
temp2_freq=data.frame(table(temp2$UniqueCarrier))
ABIA_freq=data.frame(table(ABIA$UniqueCarrier))
record_freq <- cbind(temp2_freq[1],temp2_freq[-1]/ABIA_freq[-1])
record_freq
```

```
##    Var1      Freq
## 1    9E 0.1647705
## 2    AA 0.2099025
## 3    B6 0.2280117
## 4    CO 0.2087757
## 5    DL 0.2708529
## 6    EV 0.2654545
## 7    F9 0.1646341
## 8    MQ 0.1794968
## 9    NW 0.2396694
## 10   OH 0.3382451
## 11   OO 0.1967621
## 12   UA 0.2304394
## 13   US 0.1207133
## 14   WN 0.1731563
## 15   XE 0.1933738
## 16   YV 0.2234682
```

```
#record_freq
p <- plot_ly(
  x = record_freq$Var1,
  y = record_freq$Freq,
  name = "record freq",
  type = "bar")
p
```





We could see that actually MQ and YV give out the relatively good records on the five delay components. So we could say that, airline MQ did a good job on control its CarrierDelay time.YV did the worest. So for airline YV, it is important to improve its own control of arriving time

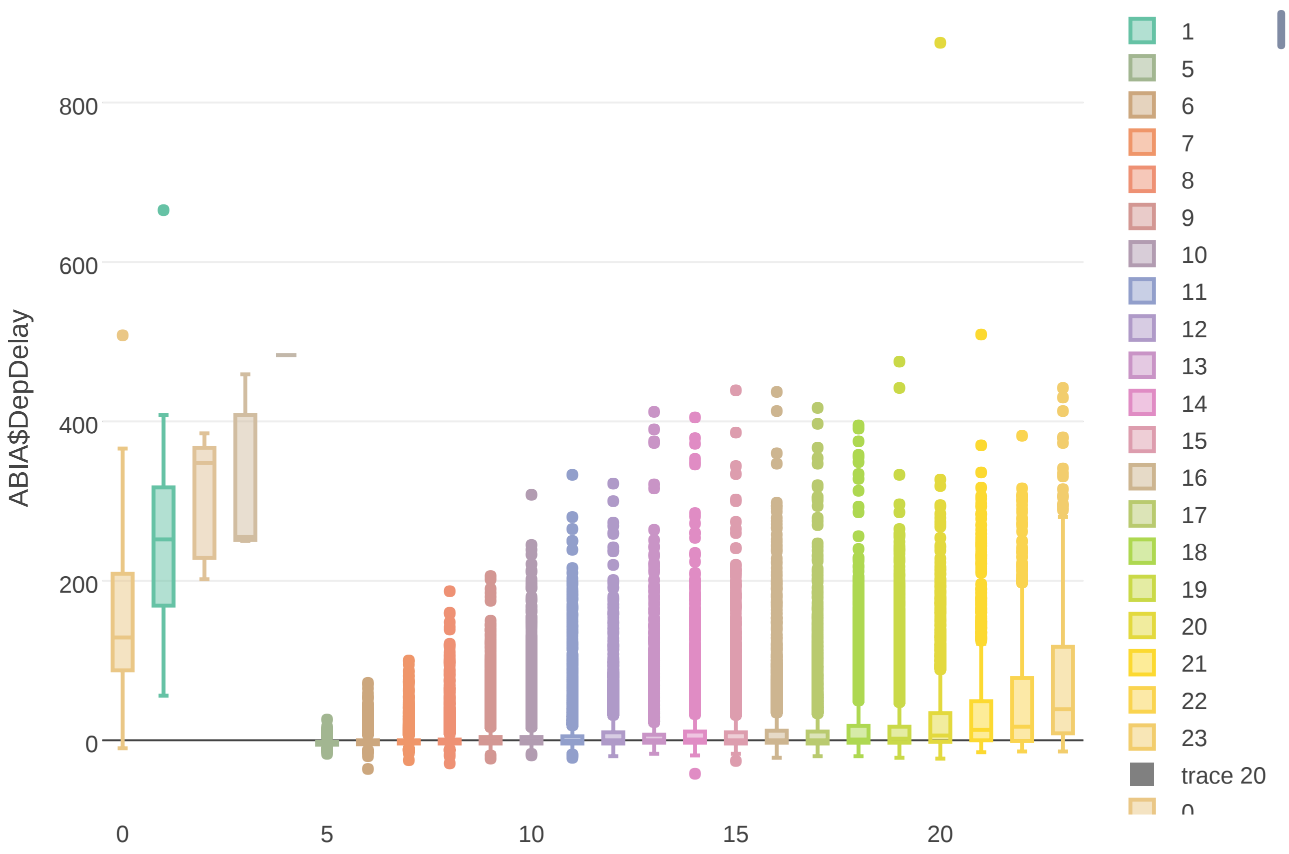
3: What is the best time of day to fly to minimize delays?

```
library(lubridate)
```

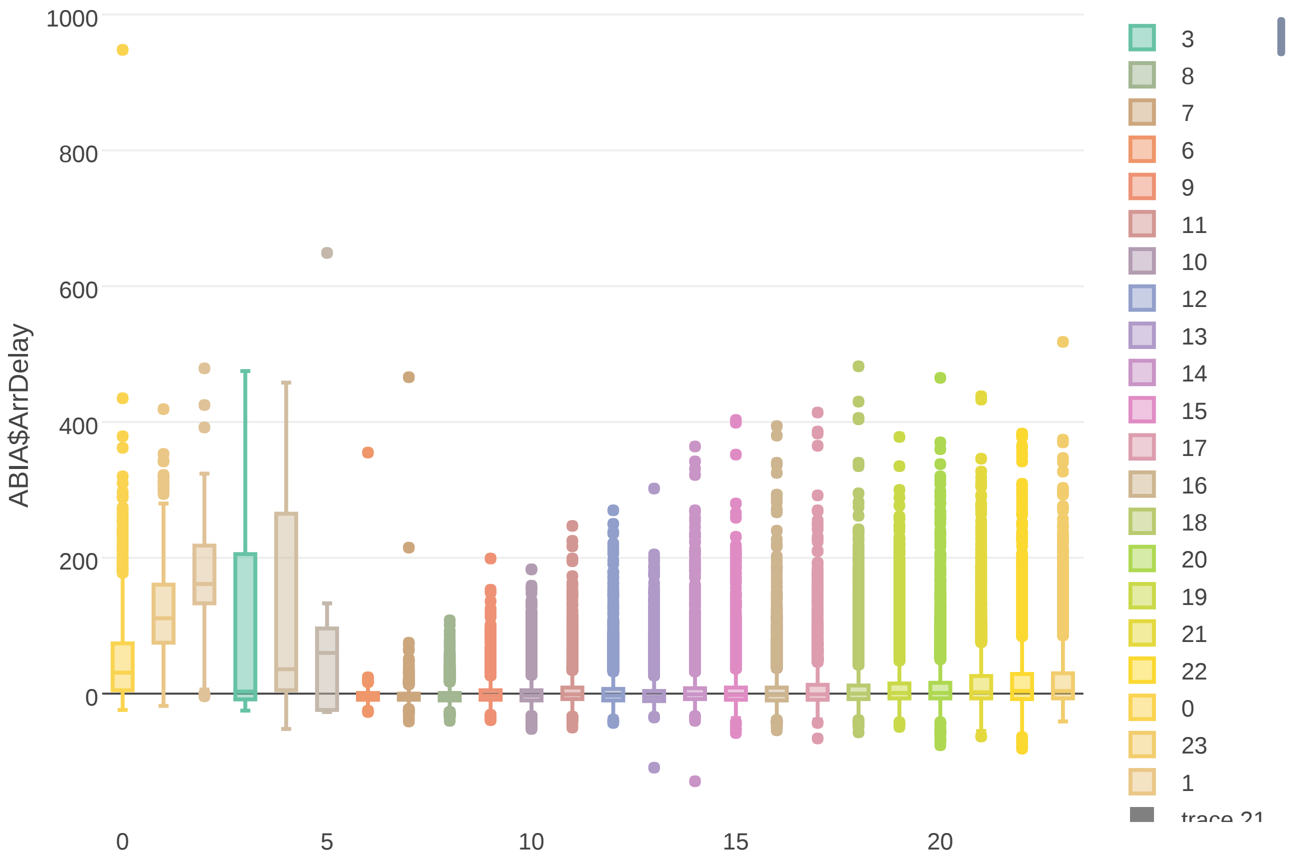
```
##
## Attaching package: 'lubridate'
```

```
## The following object is masked from 'package:base':
##
##     date
```

```
ABIA$DepTime<-format(strptime(sprintf("%04d", ABIA$DepTime), format="%H%M"), format =
"%H:%M")
ABIA$DepTime<-hm(ABIA$DepTime)
ABIA$DepTime_hour<-hour(ABIA$DepTime)
p5 <- plot_ly(ggplot2::diamonds, y = ABIA$DepDelay, color =as.character(ABIA$DepTime_ho
ur), type = "box")
p5
```



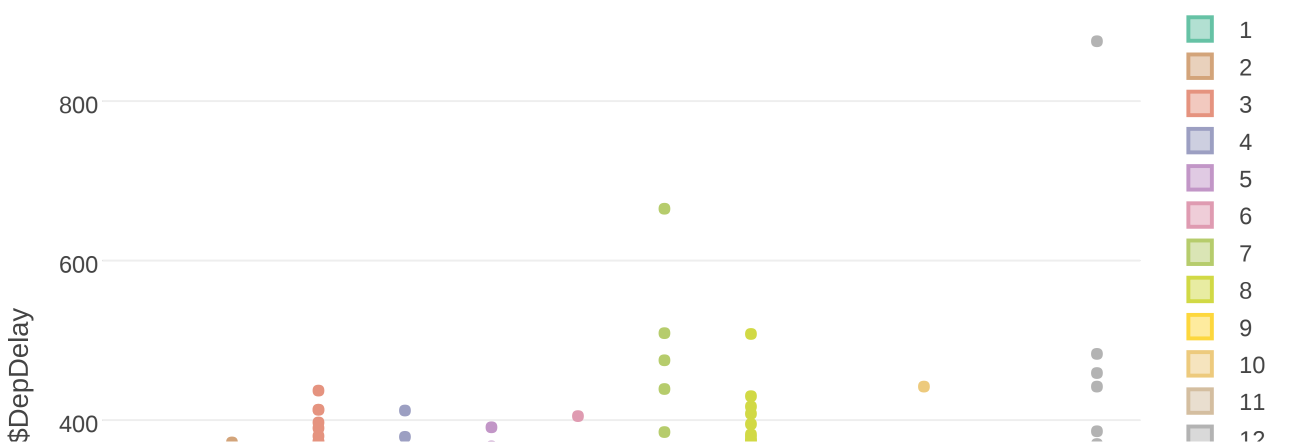
```
ABIA$ArrTime<-format(strptime(sprintf("%04d", ABIA$ArrTime), format="%H%M"), format =
"%H:%M")
ABIA$ArrTime<-hm(ABIA$ArrTime)
ABIA$ArrTime_hour<-hour(ABIA$ArrTime)
p6 <- plot_ly(ggplot2::diamonds, y = ABIA$ArrDelay, color =as.character(ABIA$ArrTime_ho
ur), type = "box")
p6
```

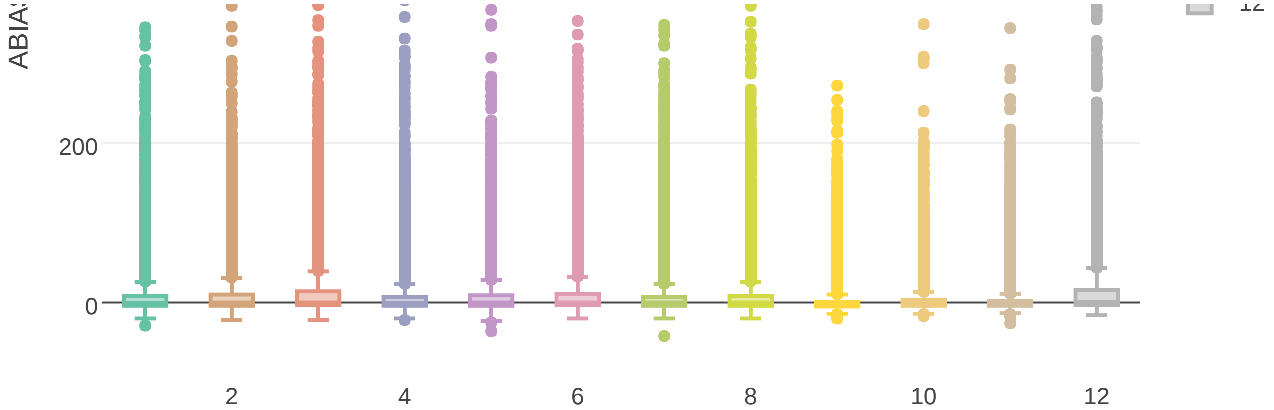


We can see from the plot that, 5:00am is the best time of the day to fly to minimize delays. In the early morning, 5:00am to 7:00am is a good period of time to fly to minimize delays.

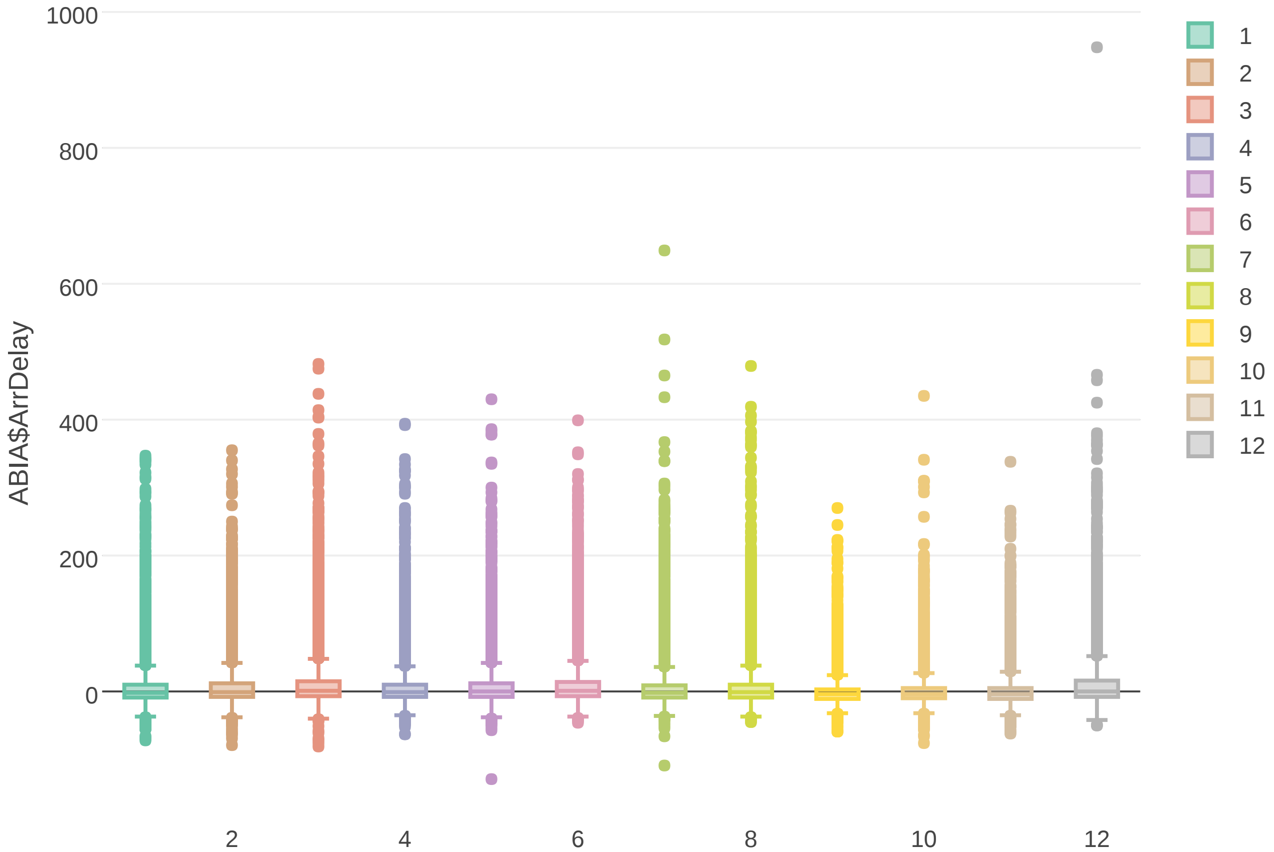
4: What is the best time of year to fly to minimize delays?

```
p7 <- plot_ly(ggplot2::diamonds, y = ABIA$DepDelay, color =as.character(ABIA$Month), ty
pe = "box")
p7
```





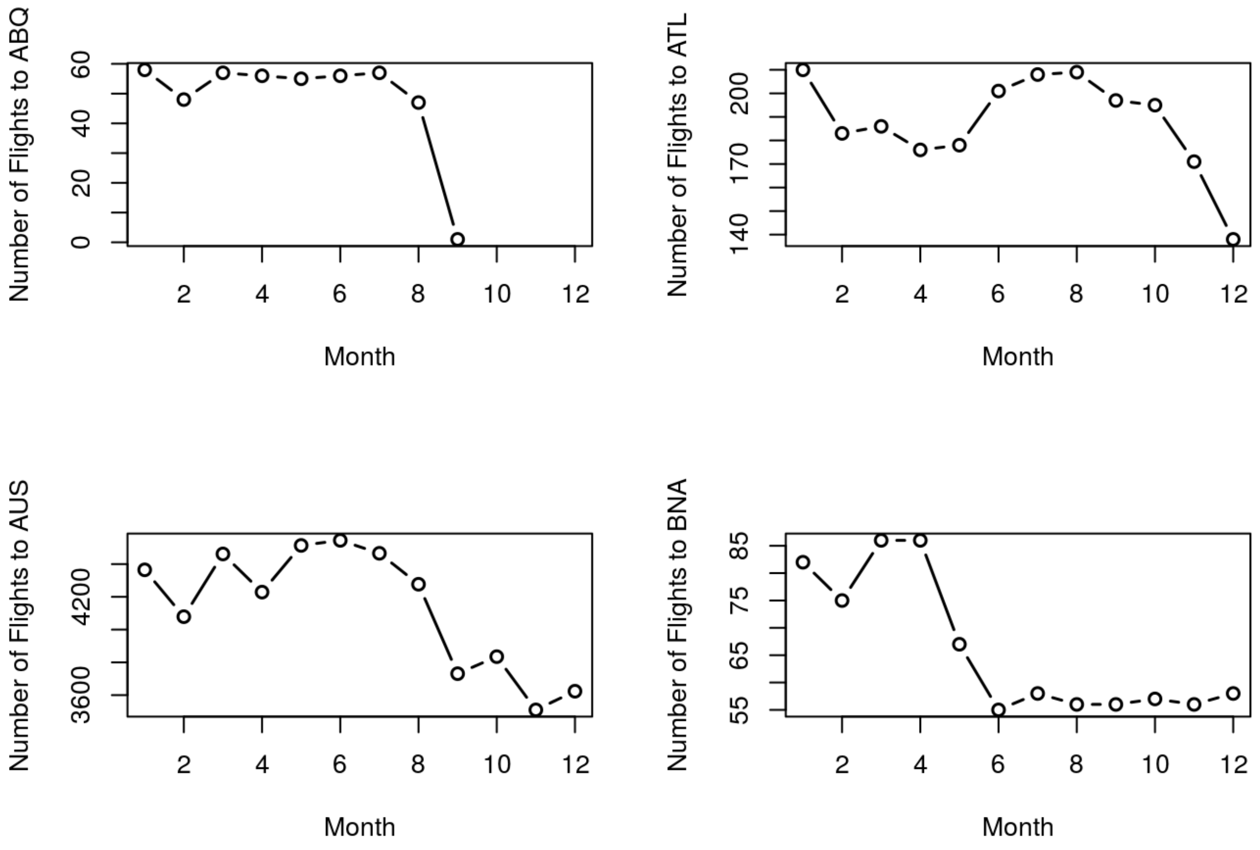

```
p8 <- plot_ly(ggplot2::diamonds, y = ABIA$ArrDelay, color =as.character(ABIA$Month), ty
pe = "box")
p8
```

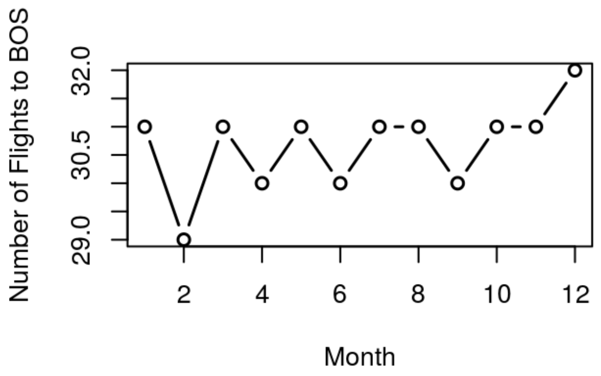


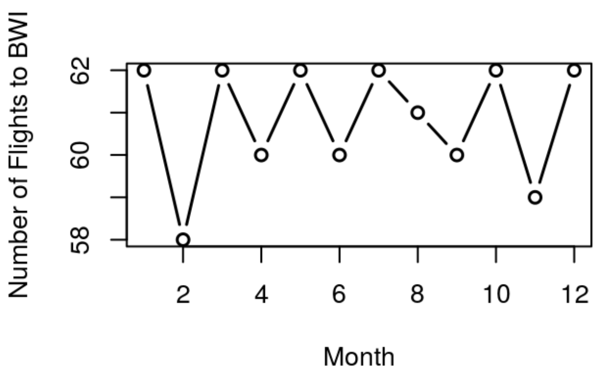
From the plots we could see that, September is the best month of the year to fly to minimize delays.

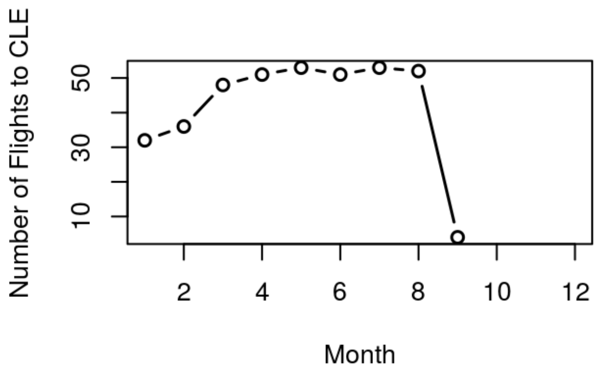
5: How do patterns of flights to different destinations or parts of the country change over the course of the year?

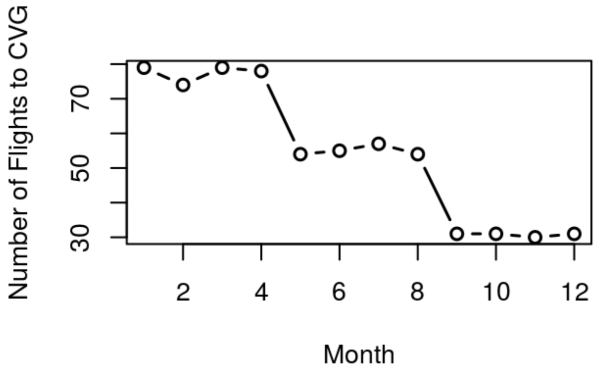
```r
library(plyr)

group<- ABIA %>%
  dplyr::group_by(Month,Dest) %>%
  dplyr::summarise(length(Dest))

par(mfrow=c(2,2))
colors <- rainbow(length(unique(group$Dest)))
linetype <- c(1:length(unique(group$Dest)))
plotchar <- seq(18,18+length(unique(group$Dest)),1)

for (i in (1:length(unique(group$Dest)))) {
  temp3 <- subset(group,Dest==unique(group$Dest)[i])
  xrange <- range(1:12)
  yrange<- range(temp3$`length(Dest)`)
  plot(xrange, yrange, type="n", xlab="Month",ylab=paste("Number of Flights to ", as.ch
aracter(unique(group$Dest)[i]), sep = ""))
  lines(temp3$Month, temp3$`length(Dest)`, type="b", lwd=1.5)
  line <- readline()
}
```

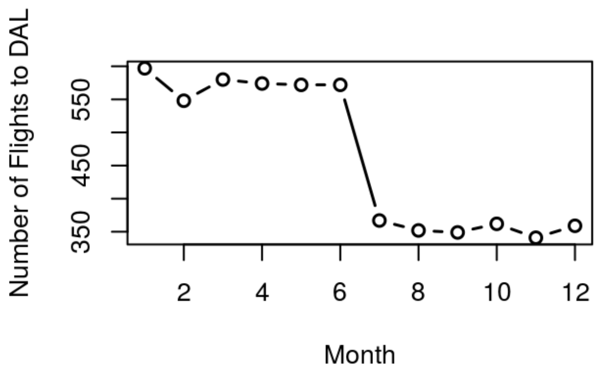


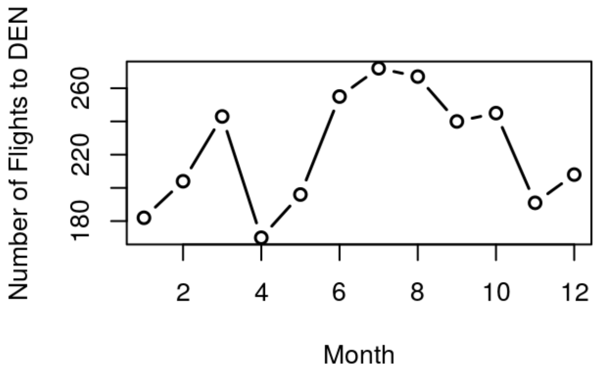

Number of Flights to BOS
29.0
30.5
32.0
2
4
6
8
10
12
Month

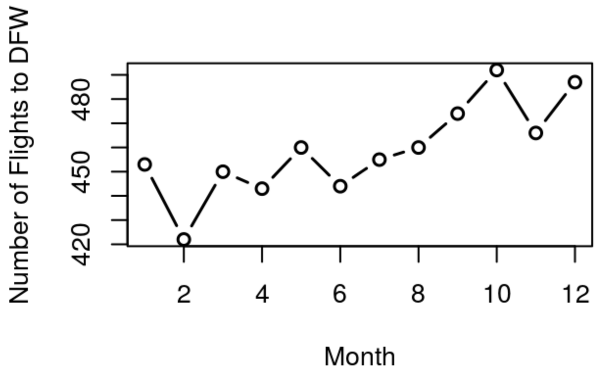

Number of Flights to BWI
58
60
62
2
4
6
8
10
12
Month

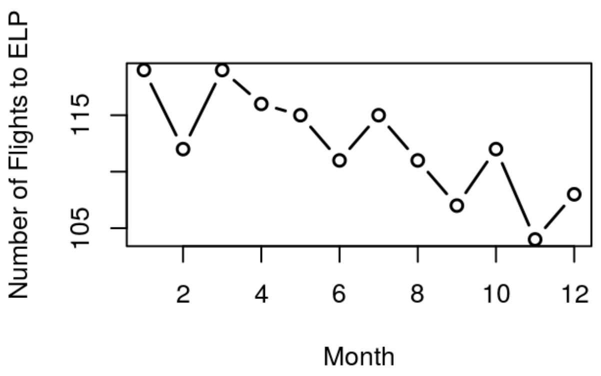

Number of Flights to CLE
10
30
50
2
4
6
8
10
12
Month

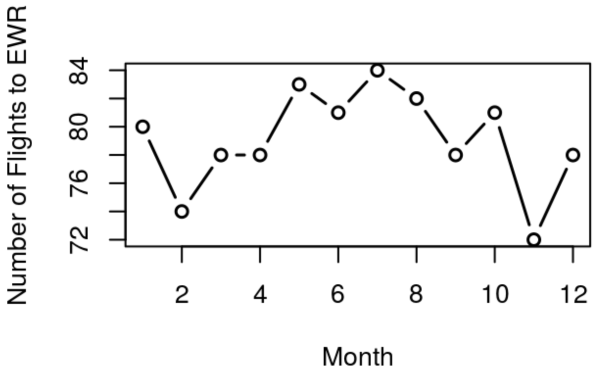

Number of Flights to CVG
30
50
70
2
4
6
8
10
12
Month

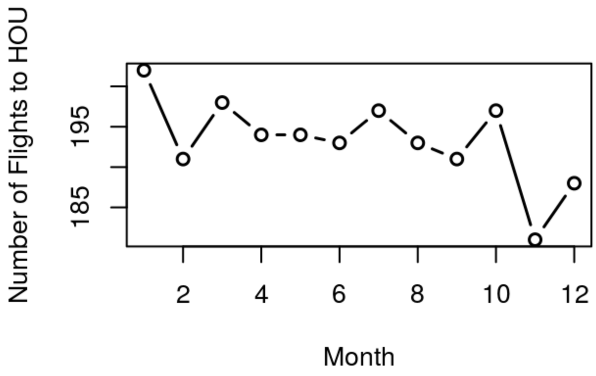

Number of Flights to DAL
350
450
550
2
4
6
8
10
12
Month

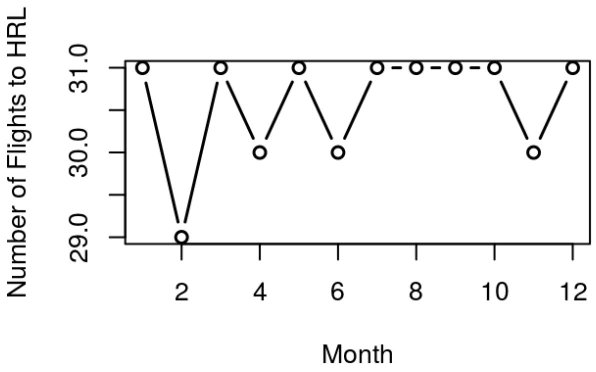

Number of Flights to DEN
180
220
260
2
4
6
8
10
12
Month

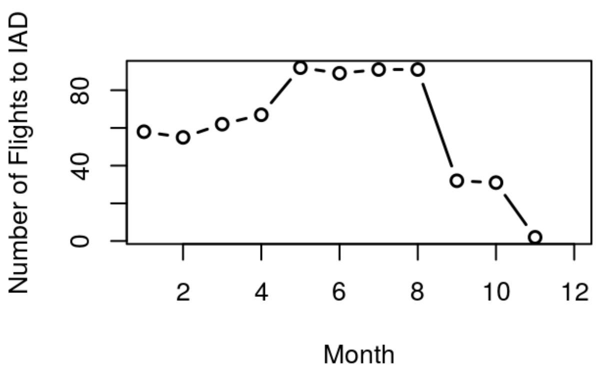

Number of Flights to DFW
420
450
480
2
4
6
8
10
12
Month

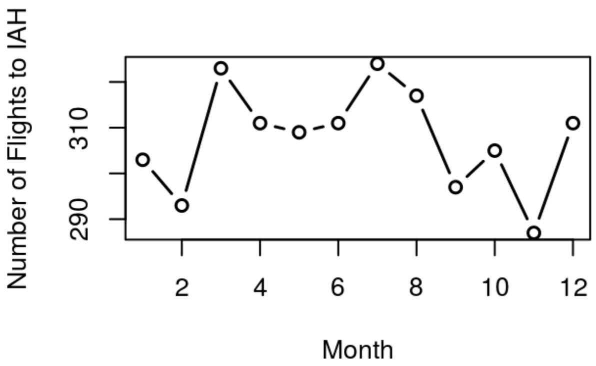

Number of Flights to ELP
105
115
2
4
6
8
10
12
Month

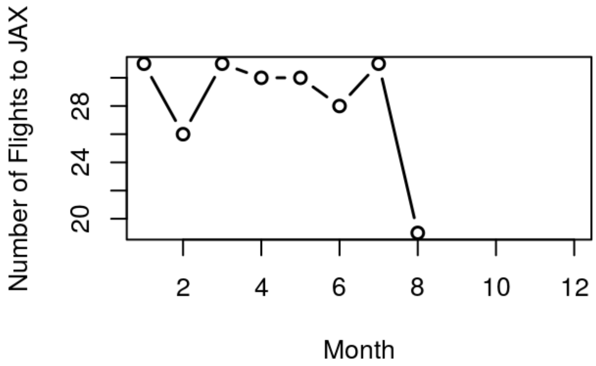

Number of Flights to EWR
72
76
80
84
2
4
6
8
10
12
Month

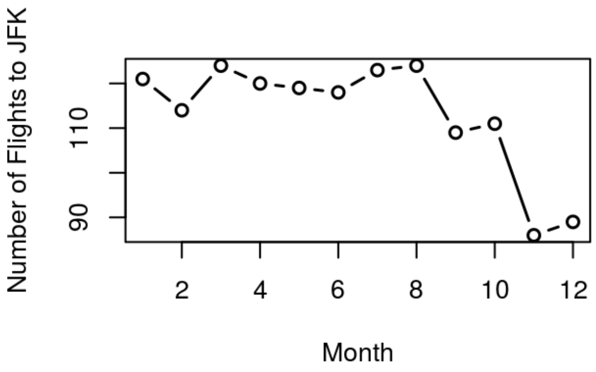

Number of Flights to HOU
185
195
2
4
6
8
10
12
Month

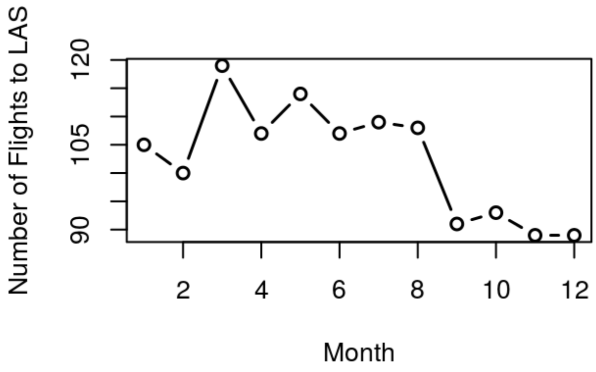

Number of Flights to HRL
29.0
30.0
31.0
2
4
6
8
10
12
Month

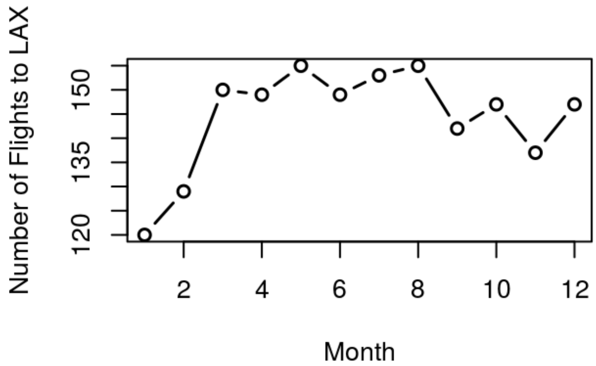

Number of Flights to IAD
0
40
80
2
4
6
8
10
12
Month

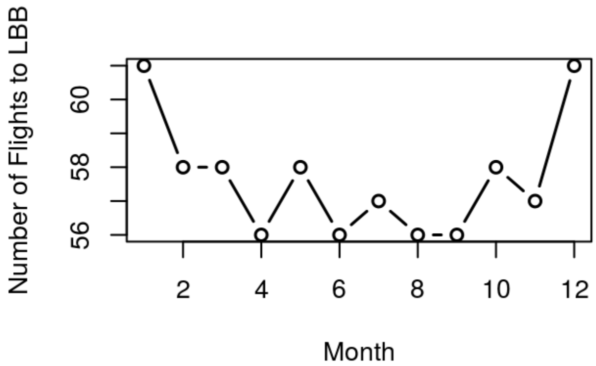

Number of Flights to IAH
290
310
2
4
6
8
10
12
Month

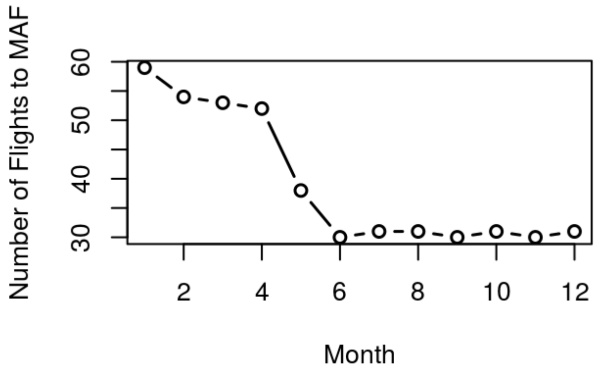

Number of Flights to JAX
20
24
28
2
4
6
8
10
12
Month

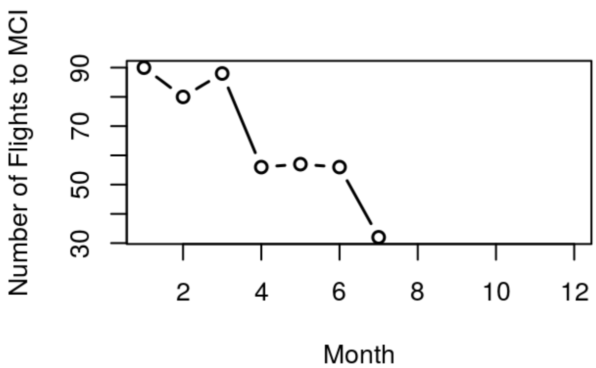

Number of Flights to JFK
90
110
2
4
6
8
10
12
Month

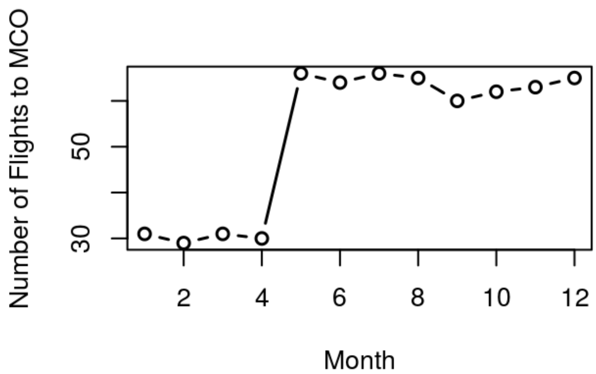

Number of Flights to LAS
90
105
120
2
4
6
8
10
12
Month

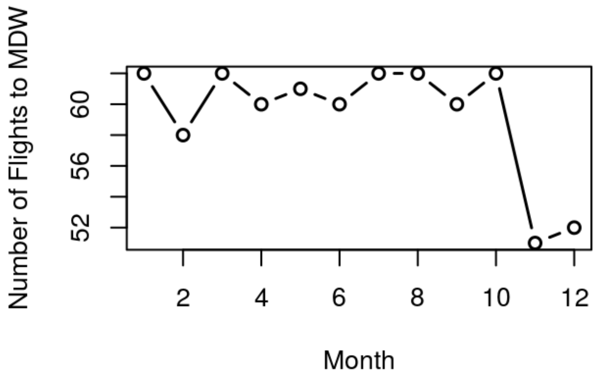

Number of Flights to LAX
120
135
150
2
4
6
8
10
12
Month

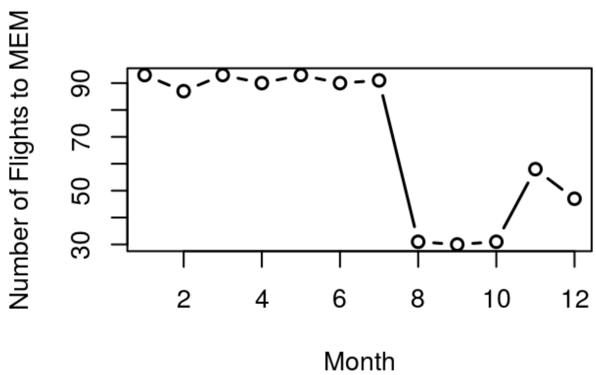

Number of Flights to LBB
56
58
60
2
4
6
8
10
12
Month

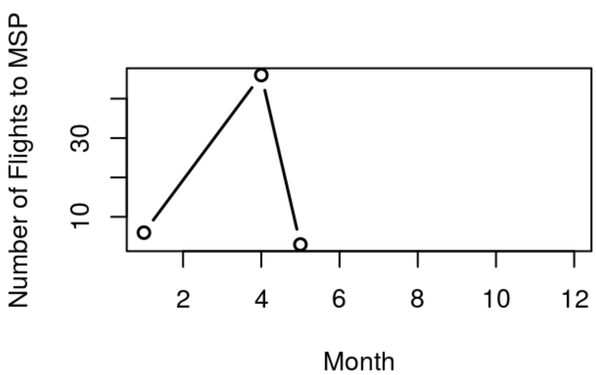

Number of Flights to MAF
30
40
50
60
2
4
6
8
10
12
Month

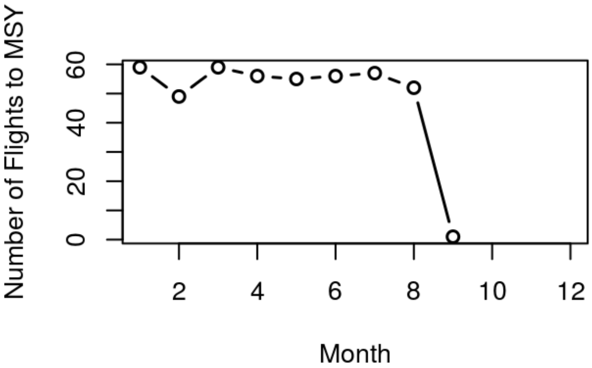

Number of Flights to MCI
30
50
70
90
2
4
6
8
10
12
Month

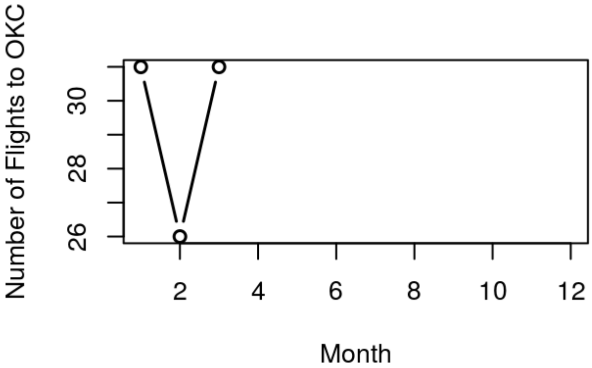

Number of Flights to MCO
30
50
2
4
6
8
10
12
Month

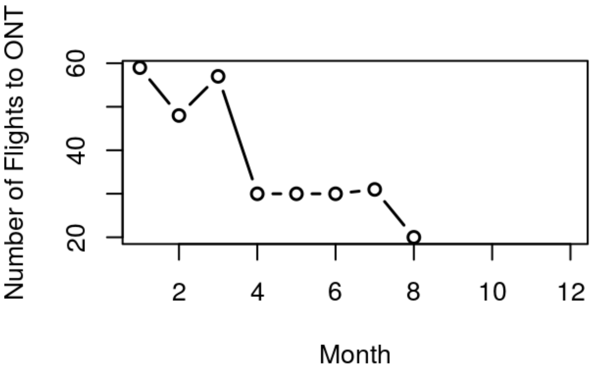

Number of Flights to MDW
52
56
60
2
4
6
8
10
12
Month

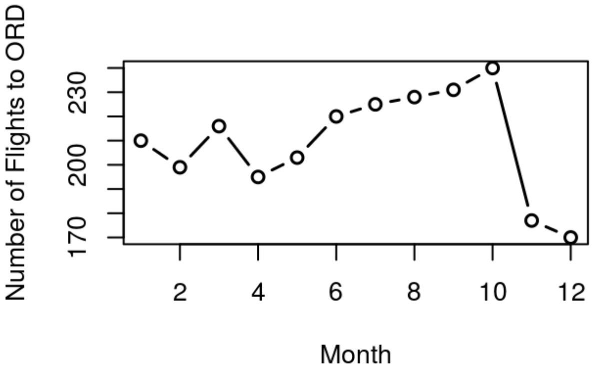

Number of Flights to MEM
30
50
70
90
2
4
6
8
10
12
Month

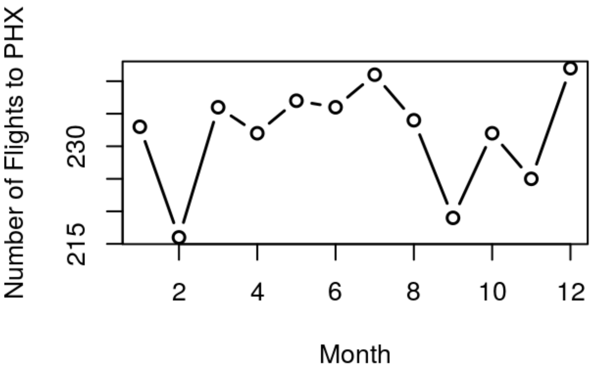

Number of Flights to MSP
10
30
2
4
6
8
10
12
Month

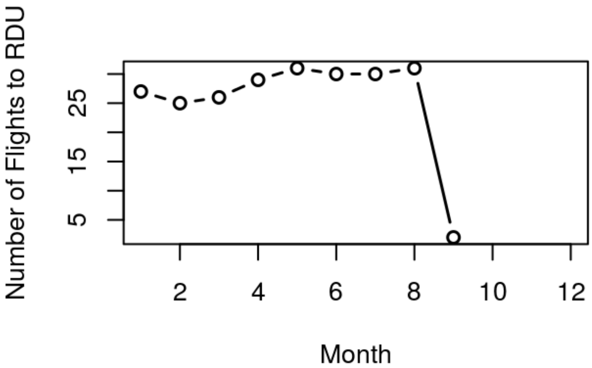

Number of Flights to MSY
0
20
40
60
2 4 6 8 10 12
Month

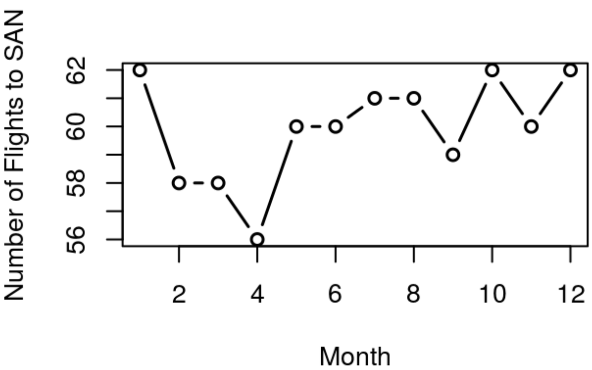

Number of Flights to OKC
26
28
30
2 4 6 8 10 12
Month

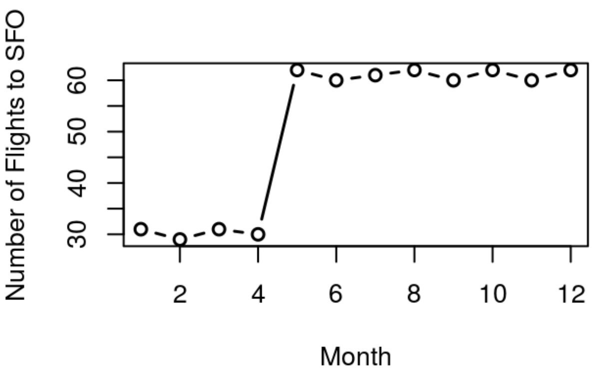

Number of Flights to ONT
20
40
60
2 4 6 8 10 12
Month

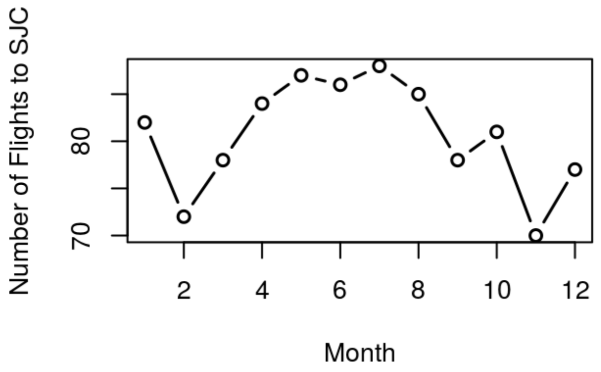

Number of Flights to ORD
170
200
230
2 4 6 8 10 12
Month

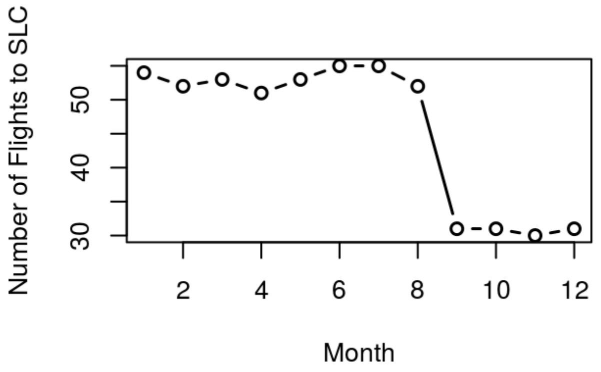

Number of Flights to PHX
215
230
2 4 6 8 10 12
Month

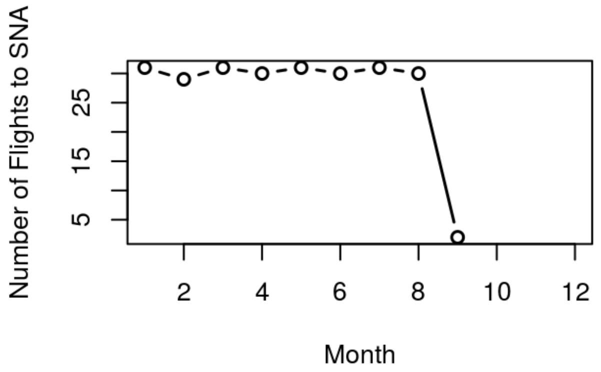

Number of Flights to RDU
5
15
25
2 4 6 8 10 12
Month

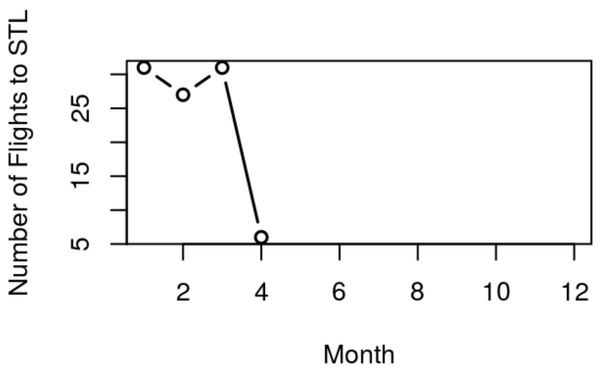

Number of Flights to SAN
56
58
60
62
2 4 6 8 10 12
Month

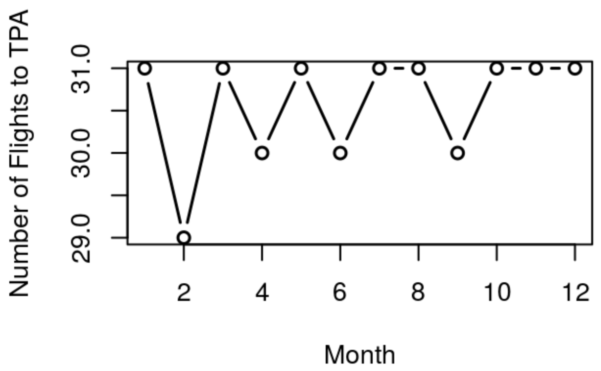

Number of Flights to SFO
30
40
50
60
2 4 6 8 10 12
Month

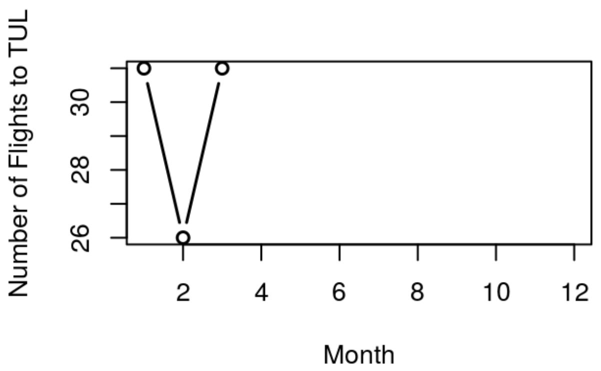

Number of Flights to SJC
70
80
2
4
6
8
10
12
Month

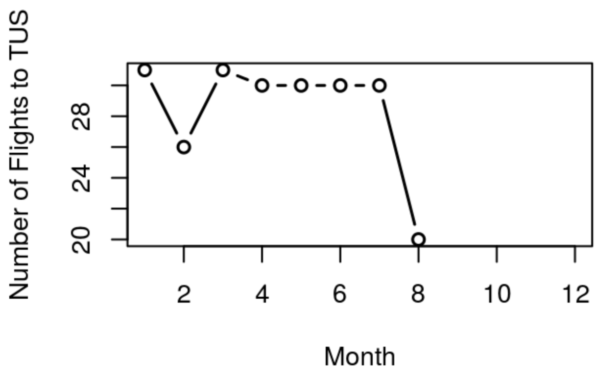

Number of Flights to SLC
30
40
50
2
4
6
8
10
12
Month

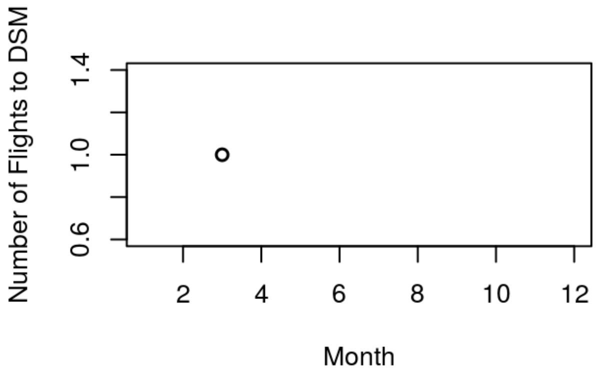

Number of Flights to SNA
5
15
25
2
4
6
8
10
12
Month

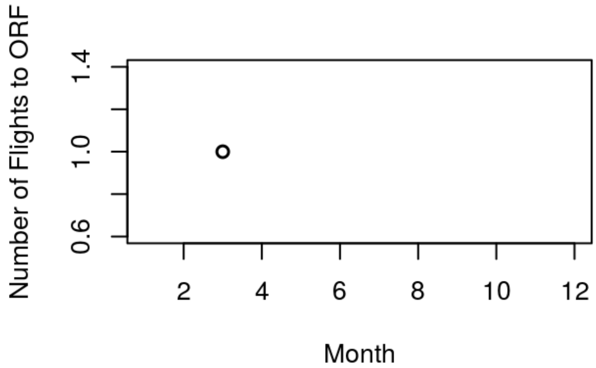

Number of Flights to STL
5
15
25
2
4
6
8
10
12
Month

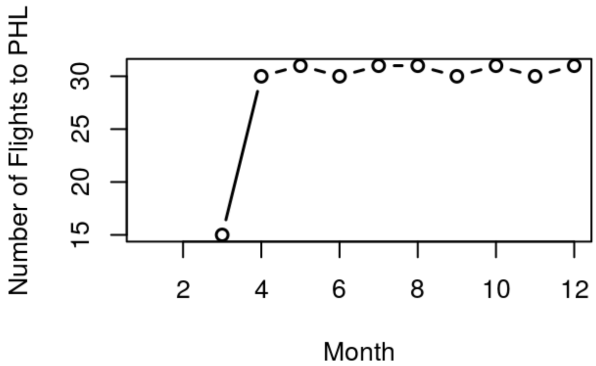

Number of Flights to TPA
29.0
30.0
31.0
2
4
6
8
10
12
Month

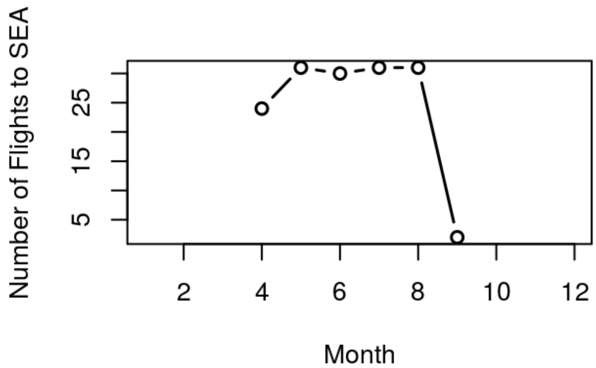

Number of Flights to TUL
26
28
30
2
4
6
8
10
12
Month

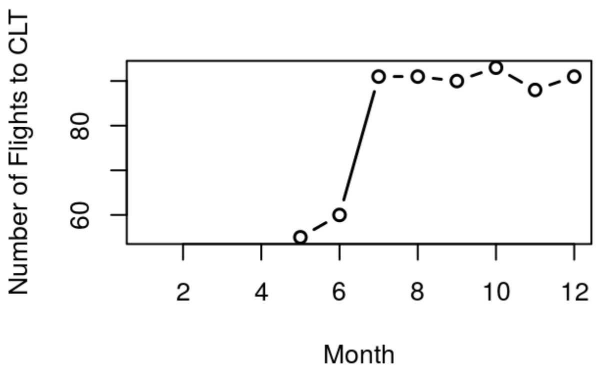

Number of Flights to TUS
20
24
28
2
4
6
8
10
12
Month

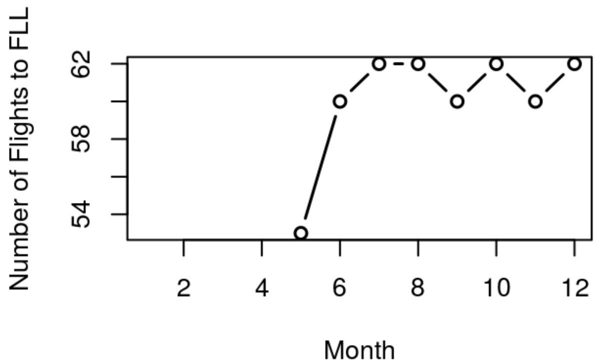

Number of Flights to DSM
0.6
1.0
1.4
2
4
6
8
10
12
Month

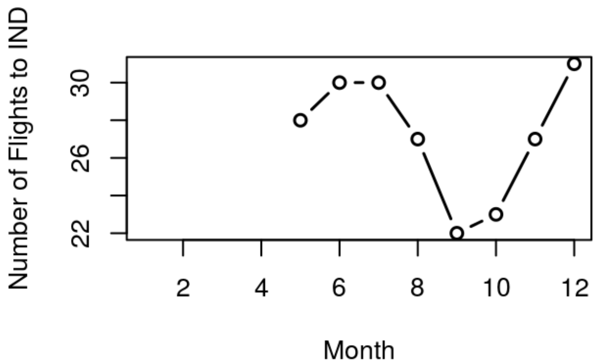

Number of Flights to ORF
0.6
1.0
1.4
2 4 6 8 10 12
Month

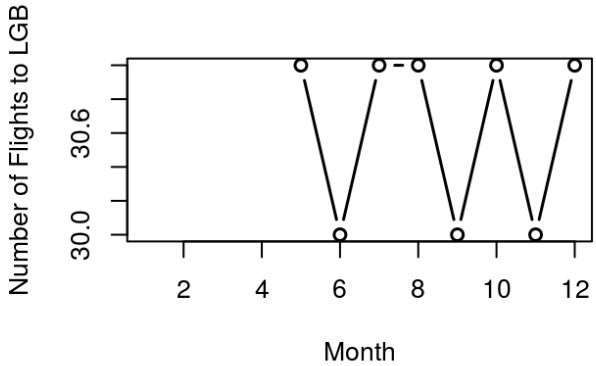

Number of Flights to PHL
15
20
25
30
2 4 6 8 10 12
Month

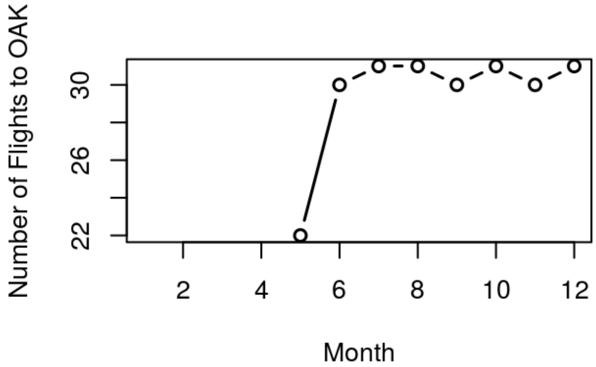

Number of Flights to SEA
5
15
25
2 4 6 8 10 12
Month

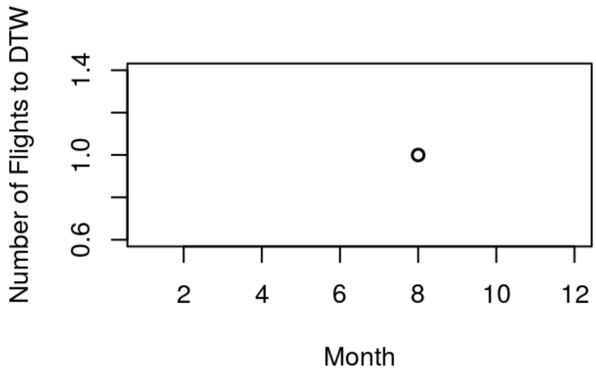

Number of Flights to CLT
60
80
2 4 6 8 10 12
Month


Number of Flights to FLL
54
58
62
2 4 6 8 10 12
Month


Number of Flights to IND
22
26
30
2 4 6 8 10 12
Month


Number of Flights to LGB
30.0
30.6
2 4 6 8 10 12
Month


Number of Flights to OAK
22
26
30
2 4 6 8 10 12
Month



There is clearly a drop in the frequency of the flights to some destinations starting around September. These destinations include ABQ, AUS, CLE, CVG, IAD, LAS, MEM, MSY, ONT, RDU, SLC, SNA, TUS, SEA. The frequency of the flights to some destinations even drop around June, which include BNA, DAL, MAF, MCI.

Author attribution

Import `tm` and `dplyr` libraries. The former is the text mining library, and the latter provides tools for matrix operations.

```
library(compiler)
enableJIT(3)
```

```
## [1] 0
```

```
rm(list=ls())
gc()
```

```
##           used (Mb) gc trigger (Mb) max used (Mb)
## Ncells  655161 35.0    1168576 62.5  1168576 62.5
## Vcells 3284682 25.1   10405454 79.4 13004239 99.3
```

```
library(tm)
library(dplyr)
```

The first thing to do with the documents is to convert them to `Corpus` objects defined in `tm` library. While the texts are read in, a series of operations are performed, including enforcing lowercase, removing numbers & punctuations, and stripping white spaces. It is sometimes problematic to remove *stopwords* as such could change the meaning of the text. But given the abundance of data in this case, removing *stopwords* provides more benefits in terms of simplifying computation.

The training set and test set are converted into `Corpus` objects in the above way, and then the object is transformed into a `DocumentTermMatrix` and later normal matrix. No sparse terms are teased out at this stage.

First let’s define a set of helper functions.

```
# a function to read articles inside a folder
readArticle <- function(article_url){
    article_list <- lapply(article_url,
                           function(article_url) readPlain(
                               elem=list(content=readLines(article_url)),
                               language="en",
                               id=article_url
                           ))
    return(article_list)
}

# convert an article into Corpus
get_corpus <- function(article){
    corp <- Corpus(VectorSource(article))
    return(corp)
}

# clean up the corpus object, i.e. standardize corpus format
clean_corpus <- function(corp, tfidf=0){
    corp <- tm_map(corp, content_transformer(tolower))
    corp <- tm_map(corp, content_transformer(removeNumbers))
    corp <- tm_map(corp, content_transformer(removePunctuation))
    corp <- tm_map(corp, content_transformer(stripWhitespace))
    corp <- tm_map(corp, content_transformer(removeWords), stopwords("en"))

    # define an optional "control" parameter
    # to weight the terms according to their tfidf index
    if (tfidf) {
        control <- list(weighting=function(x) weightTfIdf(x, normalize=FALSE))
        corp_DTM <- DocumentTermMatrix(corp, control=control)
    } else {
        corp_DTM <- DocumentTermMatrix(corp)
    }

    corp_mat <- as.matrix(corp_DTM)

    return(corp_mat)
}

# a function to convert all the folders and articles inside the folders
# into a list of corpus.
# every element in the corpus list is a terms vector of a document
# inside the same folder.
get_mat <- function(url){
    author_list <- Sys.glob(url)
    folder_list <- Sys.glob(paste(author_list, "/*", sep=""))
    articles <- lapply(folder_list, readArticle)
    corpus_list <- lapply(articles, get_corpus)
    corpus_list <- lapply(corpus_list, clean_corpus)
    return(corpus_list)
}

# a function to calculate the angle between two given vecotrs
get_cosine <- function(v1, v2){
    # first measure the original length of each vector
    len1 <- sqrt(v1%*%t(v1))
    len2 <- sqrt(v2%*%t(v2))
    # only the common columns between the two matrices are kept.
```

```
    v1_conform <- v1[colnames(v1) %in% colnames(v2)]
    v2_conform <- v2[colnames(v2) %in% colnames(v1)]
    # if the two vectors are orthogonal, then return 0
    if (length(v1_conform)*length(v2_conform)==0){
        return(0)
    } else {
        # vector angle formula
        cosine <- t(v1_conform) %*% v2_conform / (len1*len2)
        return(cosine)
    }
}

# translate the prediction matrix to author indices
pred_author <- function(pred){
    article_index <- lapply(pred, which.max)
    author_index <- lapply(article_index, function(ind) ceiling(ind/50))
    return(author_index)
}

# calculate prediction accuracy
pred_accuracy <- function(author_list){
    target <- rep(1:50, each=50)
    len <- length(author_list)
    accuracy <- mean(author_list==target[1:len])
    return(accuracy)
}

# get term frequencies within each article for Naive Bayes
get_frequency <- function(mat){
    frequency_mat <- log(mat) / as.vector(rowSums(mat))
}

# calculate the cumulative term probability for each article
# only the common columns between the two matrices are kept
get_product <- function(v1, v2){
    v1_conform <- v1[colnames(v1) %in% colnames(v2)]
    v2_conform <- v2[colnames(v2) %in% colnames(v1)]
    if (length(v1_conform)*length(v2_conform)==0){
        return(0)
    } else {
        product <- t(v1_conform) %*% v2_conform
        return(product)
    }
}
```

Read in training and test data.

> Due to high resource consumption, the following chunks are disabled.

```
train_mat <- get_mat("data/ReutersC50/C50train/*")

test_mat <- get_mat("data/ReutersC50/C50test/*")
```

Model 1: Vector angle

The first model is based on the angle between the train article terms vector and the test article terms vector. In this model, each new article from the test set is compared with all articles in the train set, by calculating the cosine of their vector angle. The train article showing the smallest angle is taken as the output, and therefore the test article is deemed to belong to the same author as the former.

```
cosine_mat <- lapply(test_mat[1:100],
                     function(v) lapply(train_mat, function(x) get_cosine(x, v)))

author_list_1 <- pred_author(cosine_mat)
accuracy_1 <- pred_accuracy(author_list_1)
accuracy_1
```

> The accuracy is ~54%.

Let’s have a look at the authors whose articles are most difficult to guess.

```
authors <- Sys.glob("data/ReutersC50/C50train/*")
authors <- gsub(".+/", "", authors)
author_list_1 <- as.matrix((read.csv("author_list_vec.csv")[-1]))
author_list_1 <- data.frame(t(author_list_1))
author_df <- data.frame(author=rep(authors, each=50))
author_df <- cbind(author_df, author_list_1[,1], rep(1:50, each=50))
colnames(author_df) <- c("author", "prediction", "actual")
author_df <- cbind(author_df, (author_df$prediction==author_df$actual)*1)
colnames(author_df)[ncol(author_df)] <- "mistake"
```

```
author_agg <- aggregate(mistake~author, data=author_df, FUN="sum")
author_agg[order(author_agg$mistake, decreasing = TRUE),]
```

```
>              author mistake
> 29   LynnleyBrowning      48
> 11    FumikoFujisaki      47
> 16      JimGilchrist      47
> 21       KarlPenhaul      45
> 33      MatthewBunce      43
> 6        BradDorfman      39
> 17          JoeOrtiz      39
> 12    GrahamEarnshaw      38
> 28    LynneO'Donnell      38
> 47    TheresePoletti      37
> 1      AaronPressman      36
> 34     MichaelConnor      36
> 36         NickLouth      35
> 22         KeithWeir      34
> 14     JaneMacartney      32
> 26 KouroshKarimkhany      32
> 27         LydiaZajc      32
> 48        TimFarrand      32
> 32        MartinWolk      30
> 40        RobinSidel      30
> 41      RogerFillion      30
> 45       SimonCowell      30
> 24     KevinMorrison      29
> 37   PatriciaCommins      29
> 39        PierreTran      29
> 5      BernardHickey      28
> 19      JonathanBirt      28
> 46          TanEeLyn      27
> 38     PeterHumphrey      24
> 23    KevinDrawbaugh      23
> 31      MarkBendeich      23
> 43      SarahDavison      23
> 18      JohnMastrini      22
> 30   MarcelMichelson      22
> 49        ToddNissen      22
> 25     KirstinRidley      21
> 2         AlanCrosby      20
> 13  HeatherScoffield      20
> 8        DavidLawder      19
> 10       EricAuchard      19
> 9      EdnaFernandes      18
> 20    JoWinterbottom      18
> 44       ScottHillis      16
> 50      WilliamKazer      16
> 3     AlexanderSmith      15
> 35        MureDickie      15
> 7   DarrenSchuettler      11
> 42       SamuelPerry       9
> 4     BenjaminKangLim       8
> 15        JanLopatka       6
```

Model 2: Naive Bayes

We can also apply Naive Bayes method to determine the authors. Specifically, in this case our problem is to predict the probability

$$P(the\ author\ is\ x|given\ the\ terms\ matrix\ of\ an\ author)$$

Where *x* can be any of the known authors from the training set. According to the general Bayes theorem, the above probability is equal to

$$\frac{P(the\ author's\ new\ document\ terms\ matrix\ is\ like\ the\ test\ set|the\ author\ is\ x)\cdot P(author\ is\ x)}{P(a\ new\ document\ terms\ matrix\ is\ like\ the\ test\ set)}$$

Since $\frac{P(author\ is\ x)}{P(a\ new\ document\ terms\ matrix\ is\ like\ the\ test\ set)}$ is the same for all articles, we consider it as a constant, and therefore we can only calculate and compare $P(the\ author's\ new\ document\ terms\ matrix\ is\ like\ the\ test\ set|the\ author\ is\ x)$.

Take logarithm of the expression and we will get the log probability terms as matrix expression

$$test * \log\frac{train}{sum(train)}$$

```
# get term frequencies for each terms vector in the training set
train_freq <- lapply(train_mat, get_frequency)
# multiply the test set article vector by that in the training set
prob_mat <- lapply(test_mat[1:100],
                   function (v) lapply(train_freq, function(x) get_product(x, v)))
# select the author with the highest conditional probability for each article
author_list_2 <- pred_author(prob_mat)
accuracy_2 <- pred_accuracy(author_list_2)
accuracy_2
author_agg <- aggregate(mistake~author, data=author_df, FUN="sum")
author_agg[order(author_agg$mistake, decreasing = TRUE),]
```

> The accuracy is ~47%.

Let's see the authors that were guesses wrong most.

```
author_list_2 <- as.matrix((read.csv("author_list_NB.csv")[-1]))
author_list_2 <- data.frame(t(author_list_2))
author_df <- data.frame(author=rep(authors, each=50))
author_df <- cbind(author_df, author_list_2[,1], rep(1:50, each=50))
colnames(author_df) <- c("author", "prediction", "actual")
author_df <- cbind(author_df, (author_df$prediction==author_df$actual)*1)
colnames(author_df)[ncol(author_df)] <- "mistake"
```

```
author_agg <- aggregate(mistake~author, data=author_df, FUN="sum")
author_agg[order(author_agg$mistake, decreasing = TRUE),]
```

```
>               author mistake
> 16      JimGilchrist      49
> 21       KarlPenhaul      43
> 29   LynnleyBrowning      43
> 33      MatthewBunce      42
> 11    FumikoFujisaki      40
> 1      AaronPressman      36
> 26 KouroshKarimkhany      36
> 12    GrahamEarnshaw      34
> 17          JoeOrtiz      34
> 27         LydiaZajc      34
> 14     JaneMacartney      33
> 28    LynneO'Donnell      32
> 34     MichaelConnor      32
> 47    TheresePoletti      32
> 5      BernardHickey      30
> 9      EdnaFernandes      29
> 40        RobinSidel      29
> 41      RogerFillion      29
> 6        BradDorfman      28
> 45       SimonCowell      28
> 19      JonathanBirt      27
> 30   MarcelMichelson      27
> 39        PierreTran      27
> 37   PatriciaCommins      25
> 22         KeithWeir      24
> 23    KevinDrawbaugh      23
> 24     KevinMorrison      23
> 46          TanEeLyn      23
> 48        TimFarrand      21
> 32        MartinWolk      20
> 13  HeatherScoffield      19
> 18      JohnMastrini      19
> 31      MarkBendeich      19
> 36         NickLouth      19
> 38     PeterHumphrey      19
> 20    JoWinterbottom      15
> 44       ScottHillis      15
> 49        ToddNissen      14
> 50      WilliamKazer      14
> 2         AlanCrosby      12
> 10       EricAuchard      12
> 25     KirstinRidley      12
> 8        DavidLawder      11
> 35        MureDickie      11
> 7   DarrenSchuettler       9
> 42       SamuelPerry       9
> 4    BenjaminKangLim       6
> 3    AlexanderSmith        3
> 15        JanLopatka       3
> 43      SarahDavison       3
```

It is seen that the two models give similar prediction accuracies, and the vanilla vector angle method is even a little bit more accurate than Naive Bayes, possibly suggesting dependency between terms (which is likely true).

Let's see which authors' articles are hard to predict.

Practice with association rule mining

```
library(compiler)
enableJIT(3)
```

```
## [1] 3
```

```
rm(list=ls())
gc()
```

```
##           used (Mb) gc trigger (Mb) max used (Mb)
## Ncells  679899 36.4    1168576 62.5  1168576 62.5
## Vcells 3331504 25.5   10405454 79.4 13004239 99.3
```

Import `arules` library for association rule mining.

```
library(arules)
```

Import data with `read.transactions()` function from `arules`, which will automatically convert each row into a list of items separated by commas, and the returned object is a `transactions` object. This function will also drop duplicate items from each basket if `rm.duplicates = TRUE`.

```
groceries <- read.transactions("data/groceries.txt", sep=",", rm.duplicates = TRUE)
```

We can assign each user an id by converting the `transactions` to a list with `as(from="transactions", to="list")`, and define `names()` of the list, and then convert the list back to `transactions`.

```
groceries_list <- as(groceries, "list")
names(groceries_list) <- as.character(1:length(groceries_list))
groceries <- as(groceries_list, "transactions")
```

At this point, the *groceries* object is suitable for a priori analysis. Before applying the `apriori` function, we need to determine what the *support* and *confidence* thresholds, and *maxlen* value. This is essentially a heuristic process, so here let's first try a higher *support* level 0.01 and *confidence* threshold 0.55 (just a little bit more than 0.5), and see what we get.

```
params <- list(support=.01, confidence=.55, maxlen=4)
grocery_rules <- apriori(groceries, parameter = params)
```

```
> Apriori
>
> Parameter specification:
>  confidence minval smax arem  aval originalSupport support minlen maxlen
>        0.55    0.1    1 none FALSE            TRUE    0.01      1      4
>  target   ext
>   rules FALSE
>
> Algorithmic control:
>  filter tree heap memopt load sort verbose
>     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
>
> Absolute minimum support count: 98
>
> set item appearances ...[0 item(s)] done [0.00s].
> set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
> sorting and recoding items ... [88 item(s)] done [0.00s].
> creating transaction tree ... done [0.00s].
> checking subsets of size 1 2 3 4 done [0.00s].
> writing ... [7 rule(s)] done [0.00s].
> creating S4 object  ... done [0.00s].
```

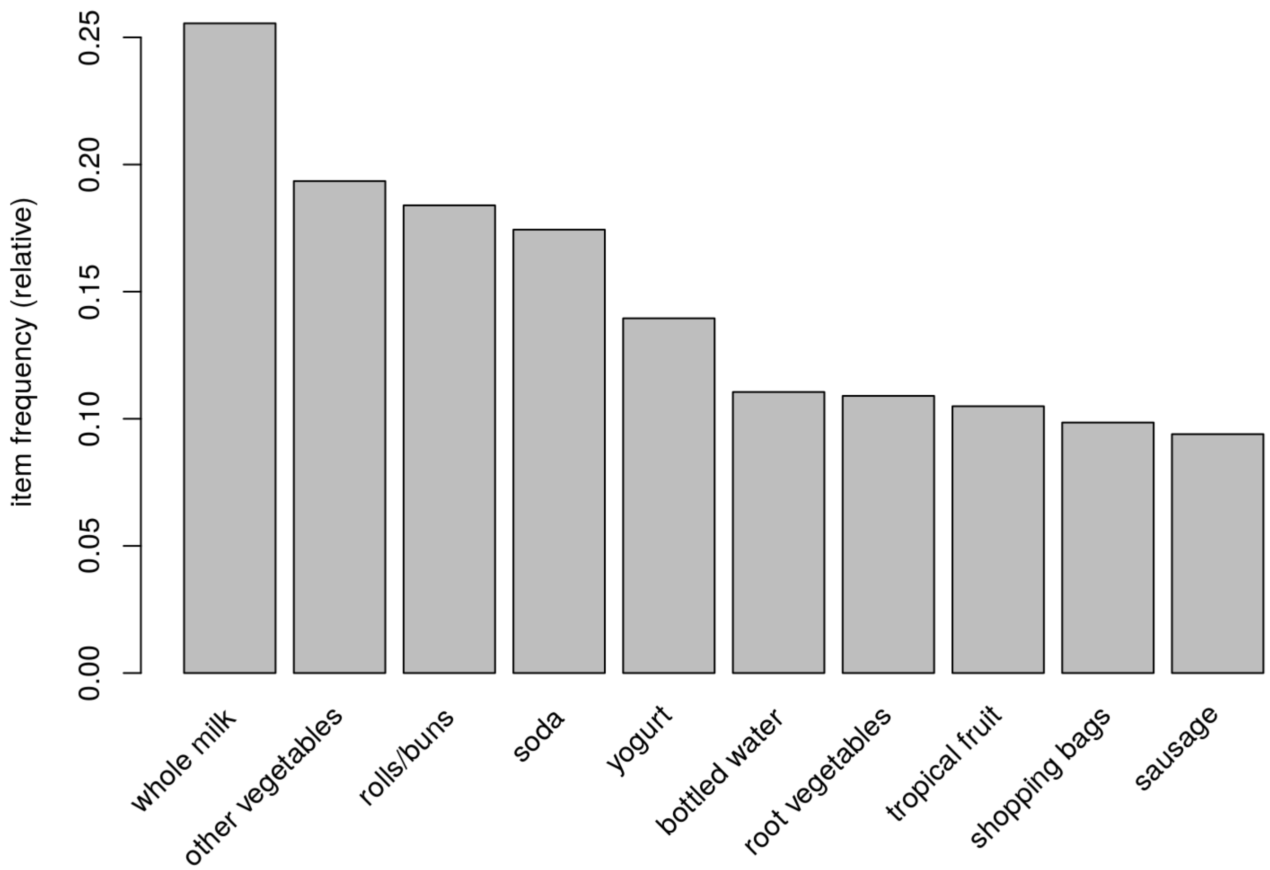
```
inspect(subset(grocery_rules, subset=lift>=2))
```

```
>   lhs                   rhs                     support confidence     lift
> 1 {curd,
>    yogurt}           => {whole milk}       0.01006609  0.5823529 2.279125
> 2 {butter,
>    other vegetables} => {whole milk}       0.01148958  0.5736041 2.244885
> 3 {domestic eggs,
>    other vegetables} => {whole milk}       0.01230300  0.5525114 2.162336
> 4 {citrus fruit,
>    root vegetables}  => {other vegetables} 0.01037112  0.5862069 3.029608
> 5 {root vegetables,
>    tropical fruit}   => {other vegetables} 0.01230300  0.5845411 3.020999
> 6 {root vegetables,
>    tropical fruit}   => {whole milk}       0.01199797  0.5700483 2.230969
> 7 {root vegetables,
>    yogurt}           => {whole milk}       0.01453991  0.5629921 2.203354
```

Here we only selected the associations with *lift* greater than 2, which gives 7 in total. And among them are items such as *other vegetables* and *whole milk*, which are themselves frequent terms across all baskets. Such results provide limited information, so we need to look closer into more interesting and less ubiquitous items.

So a natural question to be asked here is, which are the most frequent items? Let's make a plot to show the top 10 frequent terms.

```
itemFrequencyPlot(groceries, topN=10)
```



So here we can see clearly that *whole milk* and *other vegetables* are indeed frequent terms containing relatively less information.

Therefore, let’s lower *support* level to 0.001 to include less often items. Also, when printing out the associations, we raise the *lift* threshold to 10, which indicates highly correlated and dependent items.

```
params <- list(support=.001, confidence=.55, maxlen=4)
grocery_rules <- apriori(groceries, parameter = params)
```

```
> Apriori
>
> Parameter specification:
>  confidence minval smax arem  aval originalSupport support minlen maxlen
>        0.55    0.1    1 none FALSE            TRUE   0.001      1      4
>  target   ext
>   rules FALSE
>
> Algorithmic control:
>  filter tree heap memopt load sort verbose
>     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
>
> Absolute minimum support count: 9
>
> set item appearances ...[0 item(s)] done [0.00s].
> set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
> sorting and recoding items ... [157 item(s)] done [0.00s].
> creating transaction tree ... done [0.00s].
> checking subsets of size 1 2 3 4 done [0.01s].
> writing ... [3314 rule(s)] done [0.00s].
> creating S4 object  ... done [0.00s].
```

```
inspect(subset(grocery_rules, subset=lift>=10))
```

```
>   lhs                        rhs                  support confidence     lift
> 1 {liquor,
>    red/blush wine}        => {bottled beer}   0.001931876  0.9047619 11.23527
> 2 {popcorn,
>    soda}                  => {salty snack}    0.001220132  0.6315789 16.69779
> 3 {Instant food products,
>    soda}                  => {hamburger meat} 0.001220132  0.6315789 18.99565
> 4 {ham,
>    processed cheese}      => {white bread}    0.001931876  0.6333333 15.04549
> 5 {baking powder,
>    flour}                 => {sugar}          0.001016777  0.5555556 16.40807
> 6 {hard cheese,
>    whipped/sour cream,
>    yogurt}                => {butter}         0.001016777  0.5882353 10.61522
> 7 {hamburger meat,
>    whipped/sour cream,
>    yogurt}                => {butter}         0.001016777  0.6250000 11.27867
```

Here we see some intriguing associations which are less frequent among all baskets but exhibits huge correlation in terms of *lift*, which is a measure of dependence. While in the previous case there were only 7 associations even with *lift* level higher than 2, here there are 7 associations with *lift* higher than 10.

Let's look at the association {liquor, red/blush wine} => {bottled beer}, it is intuitively this is some combination appealing to an alcohol lover. This intuition also holds for other association groups, such as {baking powder, flour} => {sugar} which is probably a part of common baking recipe.

And what about other associations?

```
params <- list(support=.001, confidence=.55, maxlen=4)
grocery_rules <- apriori(groceries, parameter = params)
```

```
> Apriori
>
> Parameter specification:
>  confidence minval smax arem  aval originalSupport support minlen maxlen
>        0.55    0.1    1 none FALSE            TRUE   0.001      1      4
>  target   ext
>   rules FALSE
>
> Algorithmic control:
>  filter tree heap memopt load sort verbose
>     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
>
> Absolute minimum support count: 9
>
> set item appearances ...[0 item(s)] done [0.00s].
> set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
> sorting and recoding items ... [157 item(s)] done [0.00s].
> creating transaction tree ... done [0.00s].
> checking subsets of size 1 2 3 4 done [0.01s].
> writing ... [3314 rule(s)] done [0.00s].
> creating S4 object  ... done [0.00s].
```

```
inspect(subset(grocery_rules, subset=(lift<=10 & lift>8)))
```

```
>    lhs                          rhs                          support confidence     lift
> 1  {frozen vegetables,
>     specialty chocolate}   => {fruit/vegetable juice} 0.001016777  0.6250000 8.645394
> 2  {frozen fish,
>     other vegetables,
>     tropical fruit}        => {pip fruit}             0.001016777  0.6666667 8.812724
> 3  {flour,
>     root vegetables,
>     whole milk}            => {whipped/sour cream}    0.001728521  0.5862069 8.177794
> 4  {misc. beverages,
>     other vegetables,
>     tropical fruit}        => {fruit/vegetable juice} 0.001016777  0.5882353 8.136841
> 5  {citrus fruit,
>     fruit/vegetable juice,
>     grapes}                => {tropical fruit}        0.001118454  0.8461538 8.063879
> 6  {fruit/vegetable juice,
>     grapes,
>     tropical fruit}        => {citrus fruit}          0.001118454  0.6875000 8.306588
> 7  {citrus fruit,
>     grapes,
>     tropical fruit}        => {fruit/vegetable juice} 0.001118454  0.6111111 8.453274
> 8  {butter,
>     hard cheese,
>     yogurt}                => {whipped/sour cream}    0.001016777  0.6250000 8.718972
> 9  {butter,
>     hard cheese,
>     other vegetables}      => {whipped/sour cream}    0.001220132  0.6000000 8.370213
> 10 {butter,
>     hard cheese,
>     whole milk}            => {whipped/sour cream}    0.001423488  0.6666667 9.300236
> 11 {ham,
>     other vegetables,
>     tropical fruit}        => {pip fruit}             0.001626843  0.6153846 8.134822
> 12 {butter,
>     sliced cheese,
>     whole milk}            => {whipped/sour cream}    0.001220132  0.6000000 8.370213
> 13 {cream cheese,
>     sugar,
>     whole milk}            => {domestic eggs}         0.001118454  0.5500000 8.668670
> 14 {curd,
>     sugar,
>     yogurt}                => {whipped/sour cream}    0.001016777  0.6250000 8.718972
> 15 {butter,
>     other vegetables,
>     sugar}                 => {whipped/sour cream}    0.001016777  0.7142857 9.964539
> 16 {citrus fruit,
>     cream cheese,
>     whole milk}            => {domestic eggs}         0.001626843  0.5714286 9.006410
> 17 {domestic eggs,
>     frankfurter,
>     tropical fruit}        => {pip fruit}             0.001016777  0.6250000 8.261929
> 18 {shopping bags,
>     tropical fruit,
>     whipped/sour cream}    => {pip fruit}             0.001118454  0.6470588 8.553526
```

Above are associations with *lift* between 8 and 10. And here we can see some interesting combinations such as {butter, hard cheese, milk} => {whipped/sour cream}. Why is the customer buying such protein and fat heavy foots altogether? Probably it is simply because of the way these products are placed in the store. If some products are placed together, then they are more likely to be sold in a bundle. This can also be seen in {citrus fruit, grapes, tropical fruit} => {fruit/vegetable juice}.