Homework 2

Qijing Zhang

August 14, 2015

ABIA airport

Look at how 2008 financial crisis affected airline industry, reflected in ABIA flights

Airline industry is known to be sensitive to business cycle, aka lower growth in economic downturn and contraction.

```
options(warn=-1)
library(doBy)
```

```
## Loading required package: survival
```

```
library(ggplot2)
library(plyr)
setwd('/Users/vickyzhang/Documents/MSBA/predictive2/hw2')
abia = read.csv("../STA380/data/ABIA.csv", header=TRUE)
summary(abia)
```

```
##       Year          Month         DayofMonth      DayOfWeek
##  Min.   :2008   Min.   : 1.00   Min.   : 1.00   Min.   :1.000
##  1st Qu.:2008   1st Qu.: 3.00   1st Qu.: 8.00   1st Qu.:2.000
##  Median :2008   Median : 6.00   Median :16.00   Median :4.000
##  Mean   :2008   Mean   : 6.29   Mean   :15.73   Mean   :3.902
##  3rd Qu.:2008   3rd Qu.: 9.00   3rd Qu.:23.00   3rd Qu.:6.000
##  Max.   :2008   Max.   :12.00   Max.   :31.00   Max.   :7.000
##
##     DepTime       CRSDepTime      ArrTime       CRSArrTime
##  Min.   :   1   Min.   :  55   Min.   :   1   Min.   :   5
##  1st Qu.: 917   1st Qu.: 915   1st Qu.:1107   1st Qu.:1115
##  Median :1329   Median :1320   Median :1531   Median :1535
##  Mean   :1329   Mean   :1320   Mean   :1487   Mean   :1505
##  3rd Qu.:1728   3rd Qu.:1720   3rd Qu.:1903   3rd Qu.:1902
##  Max.   :2400   Max.   :2346   Max.   :2400   Max.   :2400
##  NA's   :1413                  NA's   :1567
##  UniqueCarrier    FlightNum       TailNum      ActualElapsedTime
##  WN     :34876   Min.   :   1          : 1104   Min.   : 22.0
##  AA     :19995   1st Qu.: 640   N678CA :  195   1st Qu.: 57.0
##  CO     : 9230   Median :1465   N511SW :  180   Median :125.0
##  YV     : 4994   Mean   :1917   N526SW :  176   Mean   :120.2
##  B6     : 4798   3rd Qu.:2653   N528SW :  172   3rd Qu.:164.0
##  XE     : 4618   Max.   :9741   N520SW :  168   Max.   :506.0
##  (Other):20749                  (Other):97265   NA's   :1601
##  CRSElapsedTime     AirTime          ArrDelay          DepDelay
##  Min.   : 17.0   Min.   :  3.00   Min.   :-129.000   Min.   :-42.000
##  1st Qu.: 58.0   1st Qu.: 38.00   1st Qu.:  -9.000   1st Qu.: -4.000
```

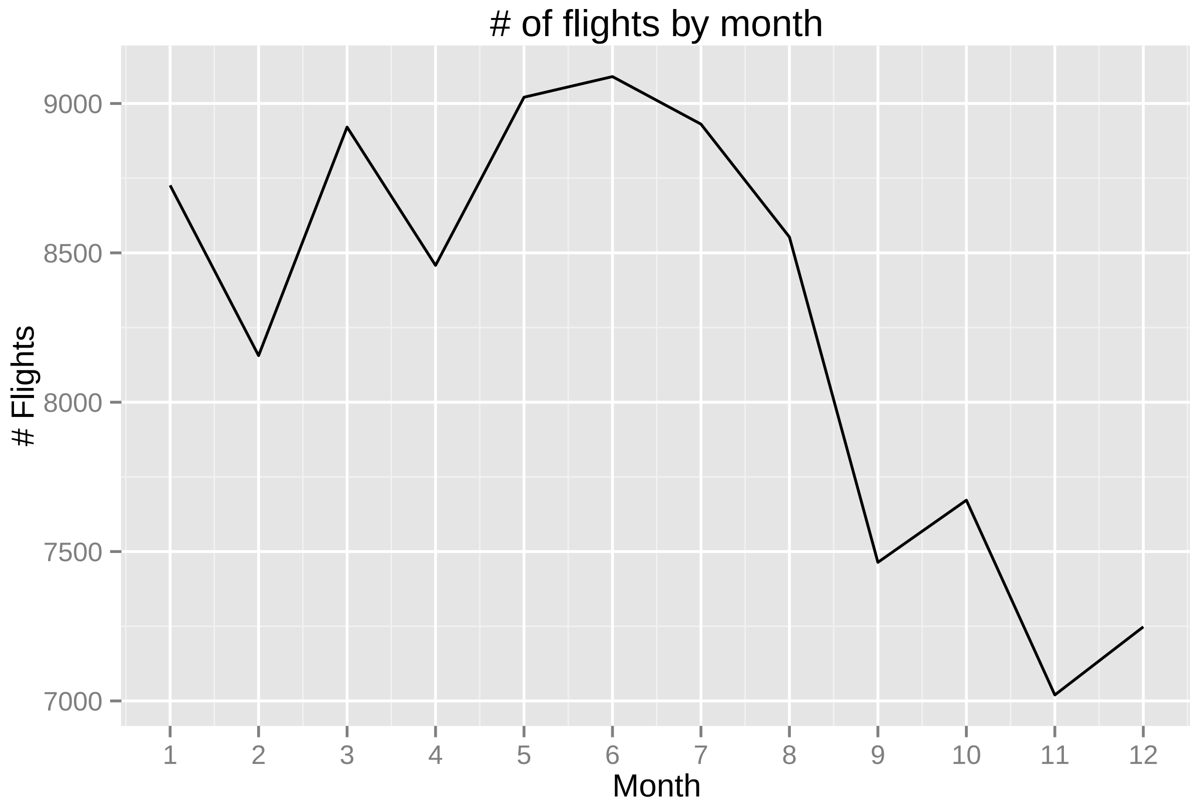
```
##   Median :130.0   Median :105.00   Median :  -2.000   Median :  0.000
##   Mean   :122.1   Mean   : 99.81   Mean   :   7.065   Mean   :  9.171
##   3rd Qu.:165.0   3rd Qu.:142.00   3rd Qu.:  10.000   3rd Qu.:  8.000
##   Max.   :320.0   Max.   :402.00   Max.   : 948.000   Max.   :875.000
##   NA's   :11      NA's   :1601     NA's   :1601       NA's   :1413
##       Origin           Dest          Distance         TaxiIn
##   AUS    :49623   AUS    :49637   Min.   :  66   Min.   :  0.000
##   DAL    : 5583   DAL    : 5573   1st Qu.: 190   1st Qu.:  4.000
##   DFW    : 5508   DFW    : 5506   Median : 775   Median :  5.000
##   IAH    : 3704   IAH    : 3691   Mean   : 705   Mean   :  6.413
##   PHX    : 2786   PHX    : 2783   3rd Qu.:1085   3rd Qu.:  7.000
##   DEN    : 2719   DEN    : 2673   Max.   :1770   Max.   :143.000
##   (Other):29337   (Other):29397                  NA's   :1567
##      TaxiOut         Cancelled       CancellationCode    Diverted
##   Min.   :  1.00   Min.   :0.00000    :97840          Min.   :0.000000
##   1st Qu.:  9.00   1st Qu.:0.00000   A:  719          1st Qu.:0.000000
##   Median : 12.00   Median :0.00000   B:  605          Median :0.000000
##   Mean   : 13.96   Mean   :0.01431   C:   96          Mean   :0.001824
##   3rd Qu.: 16.00   3rd Qu.:0.00000                    3rd Qu.:0.000000
##   Max.   :305.00   Max.   :1.00000                    Max.   :1.000000
##   NA's   :1419
##    CarrierDelay     WeatherDelay       NASDelay      SecurityDelay
##   Min.   :  0.00   Min.   :  0.00   Min.   :  0.00   Min.   :  0.00
##   1st Qu.:  0.00   1st Qu.:  0.00   1st Qu.:  0.00   1st Qu.:  0.00
##   Median :  0.00   Median :  0.00   Median :  2.00   Median :  0.00
##   Mean   : 15.39   Mean   :  2.24   Mean   : 12.47   Mean   :  0.07
##   3rd Qu.: 16.00   3rd Qu.:  0.00   3rd Qu.: 16.00   3rd Qu.:  0.00
##   Max.   :875.00   Max.   :412.00   Max.   :367.00   Max.   :199.00
##   NA's   :79513    NA's   :79513    NA's   :79513    NA's   :79513
##   LateAircraftDelay
##   Min.   :  0.00
##   1st Qu.:  0.00
##   Median :  6.00
##   Mean   : 22.97
##   3rd Qu.: 30.00
##   Max.   :458.00
##   NA's   :79513
```

```r
# get the count of flights, grouped by month
flight_by_month = summaryBy(FlightNum~Month, data=abia, FUN = length )
flight_by_month
```

```
##    Month FlightNum.length
## 1      1             8726
## 2      2             8156
## 3      3             8921
## 4      4             8458
## 5      5             9021
## 6      6             9090
## 7      7             8931
## 8      8             8553
## 9      9             7464
## 10    10             7672
## 11    11             7020
```

```
## 12    12              7248
```

```
ggplot(flight_by_month, aes(x=Month, y=FlightNum.length)) + geom_line(stat="identity") +
  labs(x="Month", y="# Flights") + labs(title = "# of flights by month") + scale_x_continuous(breaks=1:
```



There is an obvious decrease in number of flights since September, which is consistent with the time of financial crisis. It even affected the holiday season - usually the number of flights in December should be higher but it's lower than the other months.
How many flights did ABIA lose?

```
# on average, per month
mean(flight_by_month[c(1:8), 'FlightNum.length']) - mean(flight_by_month[c(9:12), 'FlightNum.length'])
```

```
## [1] 1381
```

```
# for all 4 months
(mean(flight_by_month[c(1:8), 'FlightNum.length']) - mean(flight_by_month[c(9:12), 'FlightNum.length']))
```

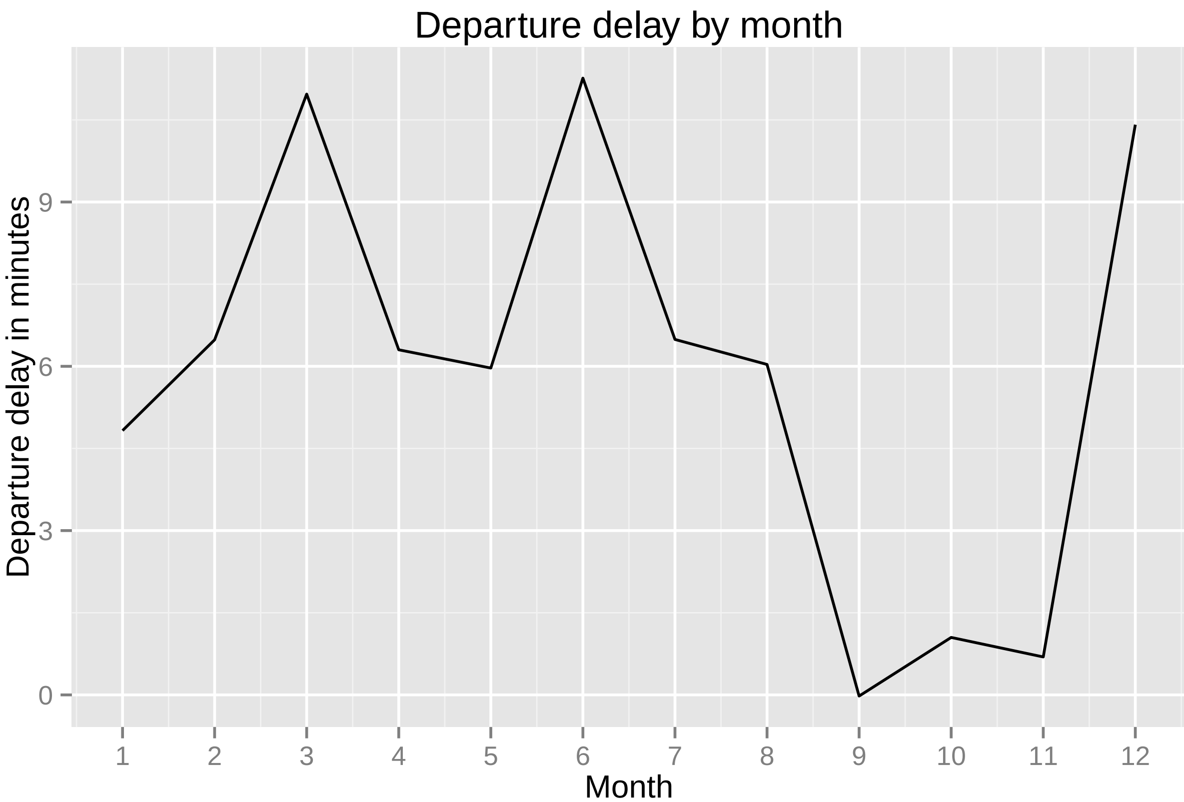
```
## [1] 5524
```

ABIA lost 5524 flights from Sept - Dec 2008, compared with the average level of the rest of the year. ### Does financial crisis affect on-time arrival?

```
# look at delay of departure flights
abia_dpt = abia[abia$Origin == 'AUS',]
# get arrival delay by month
dpt_delay_by_month = summaryBy(ArrDelay~Month, data=abia_dpt, FUN = function(x) c(m = mean(x, na.rm=TRU
dpt_delay_by_month
```

```
##    Month  ArrDelay.m
## 1      1  4.82597524
## 2      2  6.48366013
## 3      3 10.97075754
## 4      4  6.30176798
## 5      5  5.96803242
## 6      6 11.26261724
## 7      7  6.49058316
## 8      8  6.03372121
## 9      9 -0.02166164
## 10    10  1.04897852
## 11    11  0.69310049
## 12    12 10.41244046
```

```r
ggplot(dpt_delay_by_month, aes(x=Month, y=ArrDelay.m)) + geom_line(stat="identity") +
  labs(x="Month", y="Departure delay in minutes") + labs(title = "Departure delay by month") + scale_x_c
```

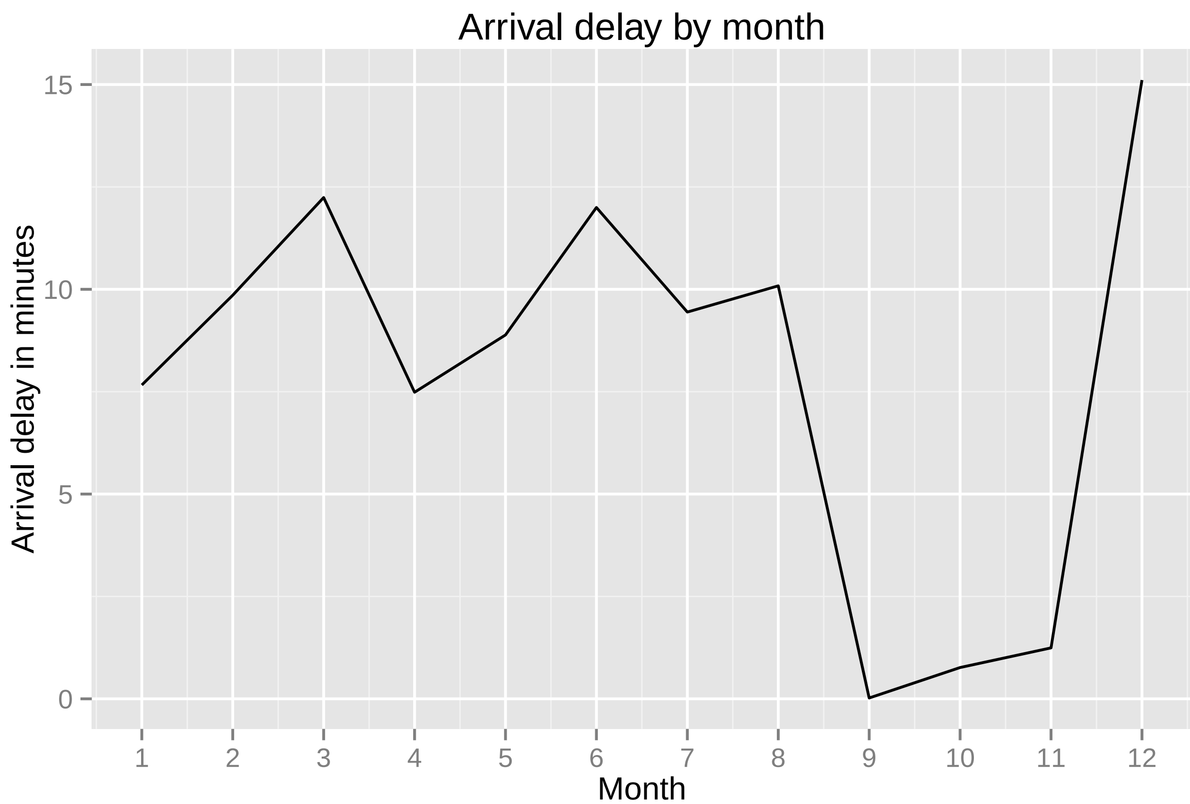


There are two peaks in this graph, March and June, which are both end of quarter, reflecting seasonality of airline industry.
In September, October and November, arrival delay actually dropped to a very low level. Maybe it's because there are fewer flights. In December, arrival delay jumped up again, probably related to holiday season. It's easy to draw correlations here, but hard to prove causality.

```r
# look at delay of arrival flights
abia_arr = abia[abia$Dest == 'AUS',]
# get arrival delay by month
arr_delay_by_month = summaryBy(ArrDelay~Month, data=abia_arr, FUN = function(x) c(m = mean(x, na.rm=TRUE
arr_delay_by_month
```

```
##    Month  ArrDelay.m
## 1      1  7.66340098
## 2      2  9.85571142
## 3      3 12.24066390
## 4      4  7.48632292
## 5      5  8.88537461
## 6      6 11.99799555
## 7      7  9.44416761
## 8      8 10.08457711
## 9      9  0.02025732
## 10    10  0.76450512
## 11    11  1.24420269
## 12    12 15.11045861
```

```
ggplot(arr_delay_by_month, aes(x=Month, y=ArrDelay.m)) + geom_line(stat="identity") +
  labs(x="Month", y="Arrival delay in minutes") + labs(title = "Arrival delay by month") + scale_x_cont
```

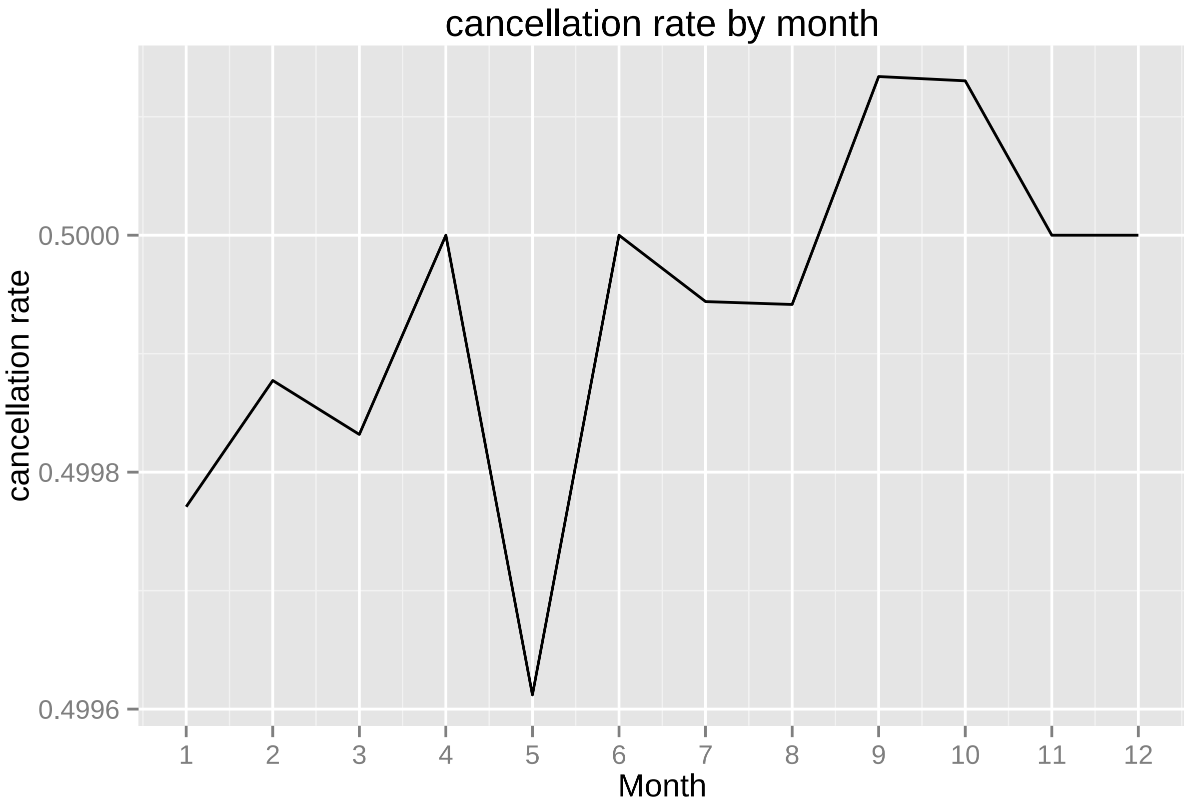


The pattern is similar to departure flights; peak in March and June, affected by financial crisis in Sept, Oct and Nov, picks up in Dec.

Does financial crisis affect flight cancellation rate?

```
# get arrival cancellation rate by month
cancel_by_month = summaryBy(Cancelled~Month, data=abia, FUN = length)
dpt_cancel_by_month = summaryBy(Cancelled~Month, data=abia_dpt, FUN = length)
# set up the new df from old df, in order to preserve 'month' column
dpt_cancel_rate = dpt_cancel_by_month
```

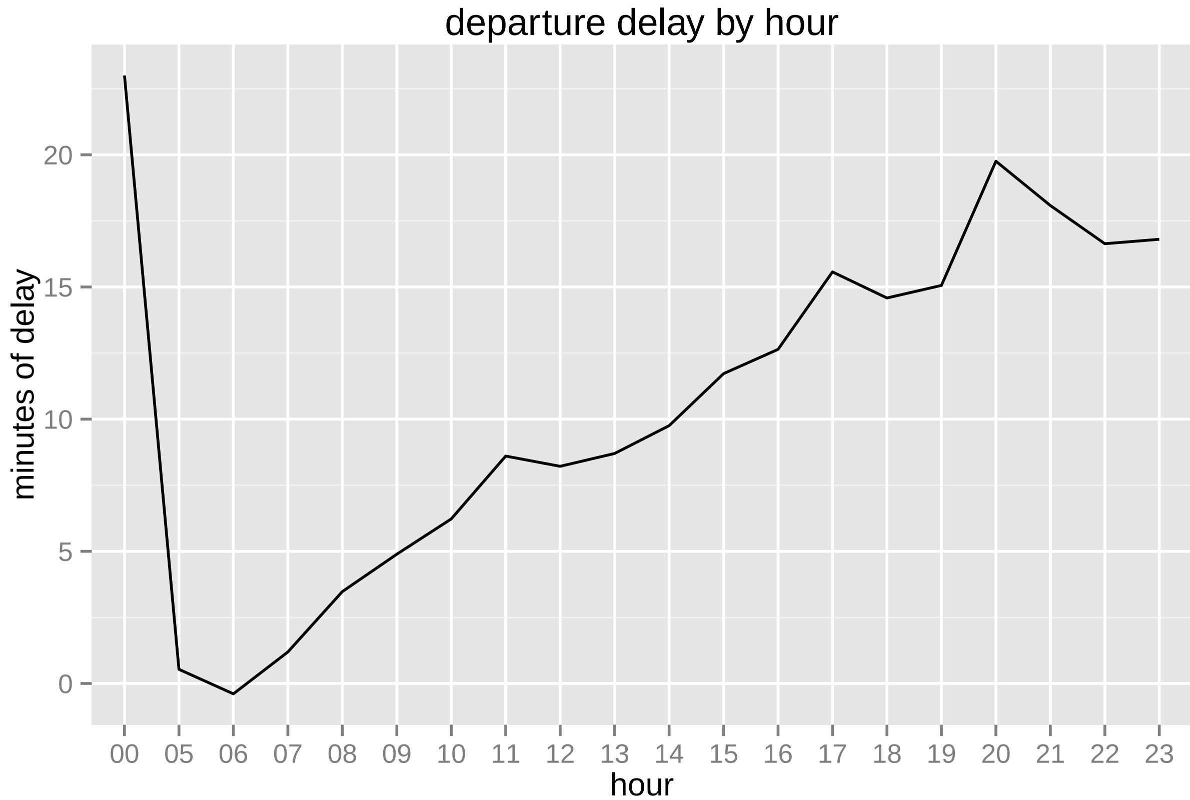
```
dpt_cancel_rate$pct= dpt_cancel_by_month[,2] / cancel_by_month[,2]
ggplot(dpt_cancel_rate, aes(x=Month, y=pct)) + geom_line(stat="identity") +
  labs(x="Month", y="cancellation rate") + labs(title = "cancellation rate by month") + scale_x_continuo
```



In terms of cancellation rate, post-crisis rate is higher than pre-crisis rate, especially in September and October when it just hit.

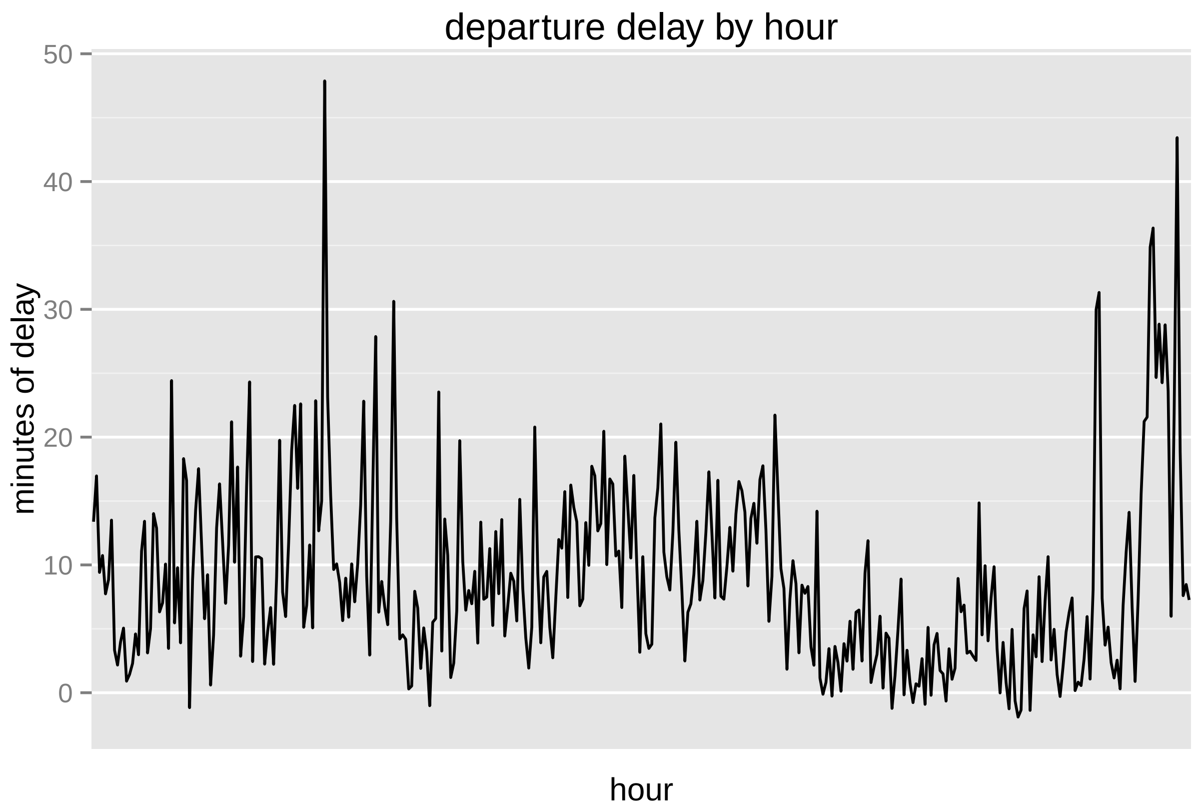
What is the best time of day to fly to minimize delays?

```
# get average minutes of departure delay by scheduled departure time (by hour)
library(stringr)
# get the hour first, doing some string manipulation on CRSDepTime
abia$hour = lapply(abia$CRSDepTime, FUN = function(x) substr(str_pad(x, width = 4, pad = 0), 1, 2))
abia <- as.data.frame(lapply(abia, unlist))
dpt_delay_by_hr = summaryBy(DepDelay~hour, data=abia, FUN = function(x) c(m = mean(x, na.rm=TRUE)))
ggplot(dpt_delay_by_hr, aes(x=hour, y=DepDelay.m, group = 1)) + geom_line(stat="identity") +
  labs(x="hour", y="minutes of delay") + labs(title = "departure delay by hour")
```



The best time to depart is 6 am and 5 am. They have around 0 minutes of departure delay.Worst time is around 12 am, which has more than 20 min of delay. ### What is the best time of year to fly to minimize delays?

```
# set up a new column having month and day of month
attach(abia)
abia$date = paste(str_pad(Month, width = 2, pad = 0), str_pad(DayofMonth, width = 2, pad = 0), sep='')
dpt_delay_by_date = summaryBy(DepDelay~date, data=abia, FUN = function(x) c(m = mean(x, na.rm=TRUE)))
abia <- within(abia, date <- factor(date))
# bug here. did get to display x axis tick labels. How to do this??????????
ggplot(dpt_delay_by_date, aes(x=date, y=DepDelay.m, group = 1)) + geom_line(stat="identity") +
  labs(x="hour", y="minutes of delay") + labs(title = "departure delay by hour") + scale_x_discrete(brea
```

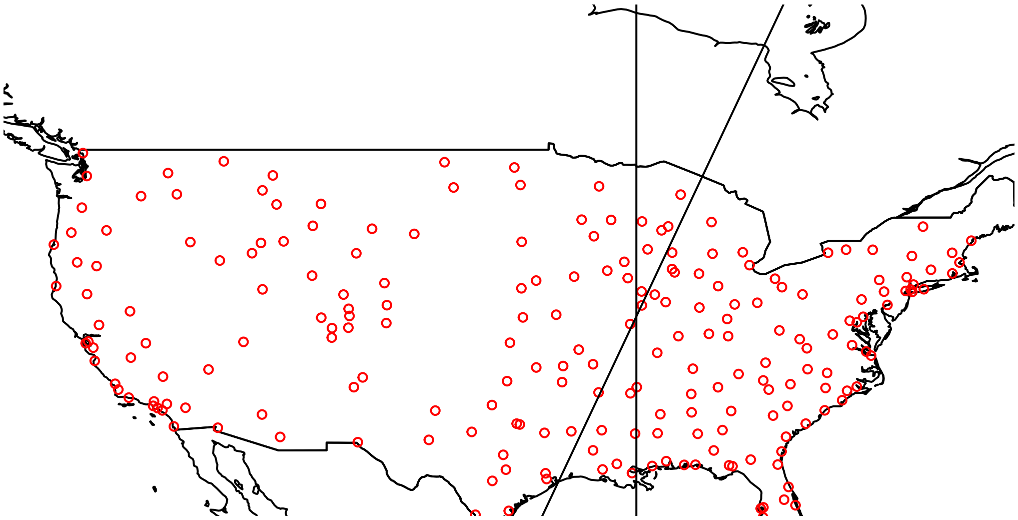


How do patterns of flights to different destinations or parts of the country change over the course of the year?

```
airports <- read.csv('https://raw.githubusercontent.com/plotly/datasets/master/2011_february_us_airport_
library(rworldmap)
```

```
## Loading required package: sp
## ### Welcome to rworldmap ###
## For a short introduction type :   vignette('rworldmap')
```

```
newmap <- getMap(resolution = "low")
plot(newmap, xlim = c(-125, -70), ylim = c(40, 45), asp = 1)
points(airports$long, airports$lat, col = "red", cex = .6)
x0=40.65236
y0=-75.44040
x1= 30.19453
y1= -97.66987
b1 = (y1-y0)/(x1-x0)
a1 = y1 - b1 * x1
abline(v=-90)
lines(x=c(x1, x0), y=c(y1, y0), col='red')
abline(a=230, b=2.12, untf = FALSE)
abline(h=c(-20,20),lty=2,col='grey')
segments(x0=40.65236, y0=-75.44040, x1= 30.19453, y1= -97.66987, col='red')
```



```r
library(ggmap)
library(mapproj)
```

```
## Loading required package: maps
##
## Attaching package: 'maps'
##
## The following object is masked from 'package:plyr':
##
##     ozone
```

```r
map <- qmap(location = 'USA', zoom = 4)
```

```
## Map from URL : http://maps.googleapis.com/maps/api/staticmap?center=USA&zoom=4&size=640x640&scale=2&m
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=USA&sensor=false
```

```r
# map + geom_point(aes(x = long, y = lat), data = airports, size=merged$Month.length)
# colnames(airports)[1] = 'Dest'
# merged = merge(airport_by_popular, airports, by='Dest', all.y=TRUE)
# plot_ly(df, lat = lat, lon = long, text = hover, color = cnt,
#         type = 'scattergeo', locationmode = 'USA-states', mode = 'markers',
#         marker = m, filename="r-docs/us-airports") %>%
#   layout(title = 'Most trafficked US airports<br>(Hover for airport)', geo = g)
```

Got stuck with plotting. start over with just numerical analysis.

```r
# the most popular destination airports
airport_by_popular = summaryBy(Month~Month+Dest, data=abia, FUN = length)
# get the top 10 most popular airports by month
library(foreach)
results = foreach(i = 1:12, .combine='c') %do% {
  rank_of_month = airport_by_popular[airport_by_popular$Month == i,]
  rank_of_month = rank_of_month[order(-rank_of_month$Month.length),]
  rank_of_month = rank_of_month[1:10,]
}
results = data.frame(results)
```

author attribution

```r
library(knitr)
library(doBy)
library(ggplot2)
library(plyr)
library(XML)
library(foreach)
# Some helper functions
library(tm)
```

```
## Loading required package: NLP
##
## Attaching package: 'NLP'
##
## The following object is masked from 'package:ggplot2':
##
##     annotate
```

```r
readerPlain = function(fname){
                readPlain(elem=list(content=readLines(fname)),
                            id=fname, language='en') }
# # get list of authors
author_dirs = Sys.glob('../data/ReutersC50/C50train/*')
author_dirs = lapply(author_dirs, function(x){substring(x, first=29)})


### get test articles from both test and training directory, do all the pre processing steps
setwd('/Users/vickyzhang/Documents/MSBA/predictive2/STA380/R') ## spend lots of time fixing a bug here
test_dirs = Sys.glob(c('../data/ReutersC50/C50tests/*', '../data/ReutersC50/C50train/*'))
file_list = NULL
labels = NULL
for(author in test_dirs) {
    author_name = substring(author, first=29)
    files_to_add = Sys.glob(paste0(author, '/*.txt'))
    file_list = append(file_list, files_to_add)
    labels = append(labels, rep(author_name, length(files_to_add)))
}
head(labels)
```

```
## [1] "AaronPressman" "AaronPressman" "AaronPressman" "AaronPressman"
## [5] "AaronPressman" "AaronPressman"
```

```r
# Need a more clever regex to get better names here
all_docs = lapply(file_list, readerPlain)
names(all_docs) = file_list
names(all_docs) = sub('.txt', '', names(all_docs))

my_corpus = Corpus(VectorSource(all_docs))
names(my_corpus) = file_list
```

```r
# Preprocessing, tokenization, data cleaning
my_corpus = tm_map(my_corpus, content_transformer(tolower)) # make everything lowercase
my_corpus = tm_map(my_corpus, content_transformer(removeNumbers)) # remove numbers
my_corpus = tm_map(my_corpus, content_transformer(removePunctuation)) # remove punctuation
my_corpus = tm_map(my_corpus, content_transformer(stripWhitespace)) ## remove excess white-space
my_corpus = tm_map(my_corpus, content_transformer(removeWords), stopwords("SMART"))

DTM = DocumentTermMatrix(my_corpus)
DTM # some basic summary statistics
```

```
## <<DocumentTermMatrix (documents: 5000, terms: 44235)>>
## Non-/sparse entries: 858721/220316279
## Sparsity           : 100%
## Maximal term length: 45
## Weighting          : term frequency (tf)
```

```r
class(DTM)  # a special kind of sparse matrix format
```

```
## [1] "DocumentTermMatrix"    "simple_triplet_matrix"
```

```r
## You can inspect its entries...
#inspect(DTM[1:10,1:20])
DTM = removeSparseTerms(DTM, 0.975) # remove those that are 0 in 97.5% of the docs or more
DTM
```

```
## <<DocumentTermMatrix (documents: 5000, terms: 1386)>>
## Non-/sparse entries: 494509/6435491
## Sparsity           : 93%
## Maximal term length: 18
## Weighting          : term frequency (tf)
```

```r
# Now a dense matrix
Z = as.matrix(DTM) # Y is test data matrix
#print(dim(Z))

# get prob for training
smooth_count = 1/2500
X = Z[1:2500,]
prob = foreach(i = 1:50, .combine='rbind') %do% {
  AP_train = X[(1+50*(i-1)):(50*i),] # be careful about dimensions, always specify both rows and col!
  w_AP = colSums(AP_train + smooth_count)
  w_AP = w_AP/sum(w_AP)
}
dim(prob)
```

```
## [1]   50 1386
```

```r
smooth_count = 1/2500
```

```r
# have to remember the dot before combine argument, otherwise... it won't run
cols = colnames(prob)
Y = Z[2501:5000,]
predictions = foreach(j=1:2500, .combine='rbind') %do% {
  y_test = Y[j,]

  logprob = foreach(i = 1:50, .combine='rbind') %do% {
    sum(y_test*log(prob[i,]))
  }

  # set the list of authors as row names of logprob
  rownames(logprob) = author_dirs
  logprob = t(logprob)
  # get the predicted author
  y_predict = names(logprob[,logprob == max(logprob)])
}
```

```r
good = 0
head(labels)
```

```
## [1] "AaronPressman" "AaronPressman" "AaronPressman" "AaronPressman"
## [5] "AaronPressman" "AaronPressman"
```

```r
labels_test = labels[2501:5000]
labels_test[1]
```

```
## [1] "AaronPressman"
```

```r
### NOTE TO PROFESSOR: For whatever reason, this code block can be run in
#console, but breaks every time I try to knit it. I will just put results from
#the console in comments. I'm not sure why it's behaving like that. I tried my
#best to diagnose it with no luck. But I believe if you run it in your R studio
#console it should work. Appreciate any input you might have on this issue.
#########
# for (i in seq(1, 2500)) {
#   if (labels_test[i] == predictions[i]) {
#   good = good + 1
#    }
#  }
good # 1465
```

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

```r
good/2500 # 0.586
```

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

So it doesn't work very well. Try PCA.

```
# each row is the PCA of an author
X = Z[1:2500,]
Y = Z[2501:4000,]
pca_all_author = foreach(j=1:50, .combine='rbind') %do% {

  corpus = X[j:(j+49),] # don't forget the bracket around j+49
  corpus = corpus/rowSums(corpus)
  # all prepared. run PCA!
  pca_author = prcomp(corpus, scale=FALSE)
  pca_author = pca_author$rotation[order(abs(pca_author$rotation[,1]),decreasing=TRUE),1]
}
pca_all_author[,pca_all_author=0] = 0.00000001
rownames(pca_all_author) = author_dirs

prediction_pca = foreach(j=1:1000, .combine='rbind') %do% {
  y_test = Y[j,]

  # for each article in test set, get the inner products, get the predicted author, put into a list
  logprob = foreach(i = 1:50, .combine='rbind') %do% {
    product = y_test*log(pca_all_author[i,])
    product[product == -Inf] = 0.0000001
    sum(product, na.rm = TRUE)
  }

  # set the list of authors as row names of logprob
  rownames(logprob) = author_dirs
  logprob = t(logprob)
  # get the predicted author
  y_predict = names(logprob[,logprob == max(logprob)])
}
```

```
### NOTE TO PROFESSOR: For whatever reason, this code block can be run in
#console, but breaks every time I try to knit it. I will just put results from
#the console in comments. I'm not sure why it's behaving like that. I tried my
#best to diagnose it with no luck. But I believe if you run it in your R studio
#console it should work. Appreciate any input you might have on this issue.
#########
# labels_test = labels[2501:5000]
# labels_test[1]
# prediction_pca[1]
# accuracy_pca = foreach(j=1:1000, .combine='c') %do% {
#   if (prediction_pca[j] == labels_test[j]) {
#     result = TRUE
#   }
#   else {
#     result = FALSE
#   }
#   result
# }
# accuracy[accuracy == TRUE] = 1
# accuracy[accuracy == FALSE] = 0
#
# sum(accuracy_pca) # 2
```

The conclusion is that Naive Bayes works much better than PCA in this scenario.

Grocery basket

```r
library(arules)  # has a big ecosystem of packages built around it
```

```
## Loading required package: Matrix
##
## Attaching package: 'arules'
##
## The following object is masked from 'package:tm':
##
##     inspect
##
## The following objects are masked from 'package:base':
##
##     %in%, write
```

```r
# Read in grocery from users
setwd('/Users/vickyzhang/Documents/MSBA/predictive2/hw2')
grocery <- read.transactions("../STA380/data/groceries.txt", rm.duplicates = TRUE, sep = ',', format =

# Now run the 'apriori' algorithm Look at rules with support > .01 & confidence
# >.5 & length (# artists) <= 4. I chose confidence=.5 because 0.5 is the
# highest level of confidence achieved; if confidence is raised 0.6, the result
# will be null. Also, even if I lower the confidence to 0.3, the rhs more or
# less look the same as confidence = 0.5
groceryrules <- apriori(grocery,
    parameter=list(support=0.01, confidence=0.5, maxlen=4))
```

```
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport support minlen maxlen
##         0.5    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
##
## apriori - find association rules with the apriori algorithm
## version 4.21 (2004.05.09)        (c) 1996-2004   Christian Borgelt
## 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 ... [15 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```r
# Look at the output
inspect(groceryrules) #### what's up with so many empty stuff?????
```

```
##    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  {whipped/sour cream,
##     yogurt}                => {whole milk}       0.01087951  0.5245098 2.052747
## 5  {other vegetables,
##     whipped/sour cream}    => {whole milk}       0.01464159  0.5070423 1.984385
## 6  {other vegetables,
##     pip fruit}             => {whole milk}       0.01352313  0.5175097 2.025351
## 7  {citrus fruit,
##     root vegetables}       => {other vegetables} 0.01037112  0.5862069 3.029608
## 8  {root vegetables,
##     tropical fruit}        => {other vegetables} 0.01230300  0.5845411 3.020999
## 9  {root vegetables,
##     tropical fruit}        => {whole milk}       0.01199797  0.5700483 2.230969
## 10 {tropical fruit,
##     yogurt}                => {whole milk}       0.01514997  0.5173611 2.024770
## 11 {root vegetables,
##     yogurt}                => {other vegetables} 0.01291307  0.5000000 2.584078
## 12 {root vegetables,
##     yogurt}                => {whole milk}       0.01453991  0.5629921 2.203354
## 13 {rolls/buns,
##     root vegetables}       => {other vegetables} 0.01220132  0.5020921 2.594890
## 14 {rolls/buns,
##     root vegetables}       => {whole milk}       0.01270971  0.5230126 2.046888
## 15 {other vegetables,
##     yogurt}                => {whole milk}       0.02226741  0.5128806 2.007235
```

```r
# it's not very informative other than 'this customer didn't buy more than 4 items'

## Choose a subset
inspect(subset(groceryrules, 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  {whipped/sour cream,
##     yogurt}                => {whole milk}       0.01087951  0.5245098 2.052747
## 5  {other vegetables,
##     pip fruit}             => {whole milk}       0.01352313  0.5175097 2.025351
## 6  {citrus fruit,
##     root vegetables}       => {other vegetables} 0.01037112  0.5862069 3.029608
```

```
## 7  {root vegetables,
##      tropical fruit}     => {other vegetables} 0.01230300  0.5845411 3.020999
## 8  {root vegetables,
##      tropical fruit}     => {whole milk}       0.01199797  0.5700483 2.230969
## 9  {tropical fruit,
##      yogurt}             => {whole milk}       0.01514997  0.5173611 2.024770
## 10 {root vegetables,
##      yogurt}             => {other vegetables} 0.01291307  0.5000000 2.584078
## 11 {root vegetables,
##      yogurt}             => {whole milk}       0.01453991  0.5629921 2.203354
## 12 {rolls/buns,
##      root vegetables}    => {other vegetables} 0.01220132  0.5020921 2.594890
## 13 {rolls/buns,
##      root vegetables}    => {whole milk}       0.01270971  0.5230126 2.046888
## 14 {other vegetables,
##      yogurt}             => {whole milk}       0.02226741  0.5128806 2.007235
```

```r
inspect(subset(groceryrules, subset=confidence > 0.5))
```

```
##    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  {whipped/sour cream,
##      yogurt}             => {whole milk}       0.01087951  0.5245098 2.052747
## 5  {other vegetables,
##      whipped/sour cream} => {whole milk}       0.01464159  0.5070423 1.984385
## 6  {other vegetables,
##      pip fruit}          => {whole milk}       0.01352313  0.5175097 2.025351
## 7  {citrus fruit,
##      root vegetables}    => {other vegetables} 0.01037112  0.5862069 3.029608
## 8  {root vegetables,
##      tropical fruit}     => {other vegetables} 0.01230300  0.5845411 3.020999
## 9  {root vegetables,
##      tropical fruit}     => {whole milk}       0.01199797  0.5700483 2.230969
## 10 {tropical fruit,
##      yogurt}             => {whole milk}       0.01514997  0.5173611 2.024770
## 11 {root vegetables,
##      yogurt}             => {whole milk}       0.01453991  0.5629921 2.203354
## 12 {rolls/buns,
##      root vegetables}    => {other vegetables} 0.01220132  0.5020921 2.594890
## 13 {rolls/buns,
##      root vegetables}    => {whole milk}       0.01270971  0.5230126 2.046888
## 14 {other vegetables,
##      yogurt}             => {whole milk}       0.02226741  0.5128806 2.007235
```

```r
inspect(subset(groceryrules, subset=support > .02 & confidence > 0.5))
```

```
##   lhs                   rhs               support confidence     lift
## 1 {other vegetables,
##    yogurt}           => {whole milk} 0.02226741  0.5128806 2.007235
```

Generally, those who buy groceries buy whole milk and other vegetables. These two items have the biggest support and confidence. The one with highest support and confidence is {other vegetables,yogurt} => {whole milk}. So if people buy both yogurt and vegetables, they tend to buy whole milk too.