# HW2-Q1

## Flights at ABIA

Agenda : Plot a visual story with respect to departure delays at Austin Airport Dataset : Only outbound flights from Austin

Cleaning the data and getting it in the required format:

```
library(ggplot2)

#Reading the file
flights <- read.csv("ABIA.csv")
#names(flights)
#unique(flights$UniqueCarrier)

#Creating variables
flights$DepHour <- round(flights$CRSDepTime/100,0)
flights$Season <- ifelse(flights$Month %in% c(12,1,2),"Winter",ifelse(flights
$Month %in% c(3,4,5),"Spring",ifelse(flights$Month %in% c(6,7,8),"Summer","Fa
ll")))
flights$dummy <- 1

#Converting numeric to factors
flights$Month <- as.factor(flights$Month)
flights$DayofMonth <- as.factor(flights$DayofMonth)
flights$DayOfWeek <- as.factor(flights$DayOfWeek)

#Subsetting all flights flying out of Austin
outbound_Aus <- subset(flights, flights$Origin == "AUS")

#Subsetting all flights flying out of Austin and have experienced delays
outbound_Aus_delay <- subset(flights, flights$Origin == "AUS" & flights$DepDe
lay > 0)
```

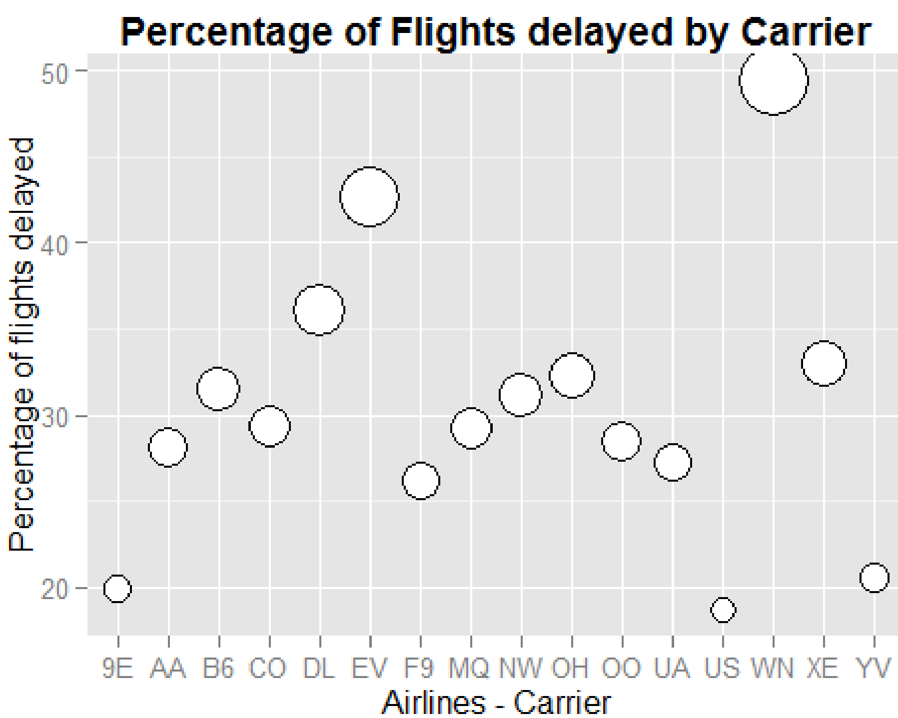
Which Airline carriers are notorious for delay in departure?

```
#Airlines delay count
carrier.delay <- aggregate(outbound_Aus_delay$dummy,by=list(outbound_Aus_dela
y$UniqueCarrier),FUN = sum, na.rm=TRUE)
names(carrier.delay)[1] <- "UniqueCarrier"
names(carrier.delay)[2] <- "AnnualDelays"

#Airlines total outbound count
carrier.total <- aggregate(outbound_Aus$dummy,by=list(outbound_Aus$UniqueCarr
ier),FUN = sum, na.rm=TRUE)
names(carrier.total)[1] <- "UniqueCarrier"
```

```
names(carrier.total)[2] <- "AnnualFlights"

carrier.delay.perc <- merge(carrier.total,carrier.delay, by = "UniqueCarrier"
)
carrier.delay.perc$per.delays <- round(carrier.delay.perc$AnnualDelays/carrie
r.delay.perc$AnnualFlights,3)*100
```

*Percentage of Flights delayed by Carrier:*

```
ggplot(data=carrier.delay.perc, aes(x= UniqueCarrier, y= per.delays)) + geom_
point( size= (carrier.delay.perc$per.delays)/4, shape=21, fill= "white") + sc
ale_size_area() + ggtitle("Percentage of Flights delayed by Carrier") + theme
(plot.title = element_text(lineheight=1.2, face="bold")) + xlab("Airlines - C
arrier") + ylab("Percentage of flights delayed")
```




```
#identify(carrier.delay.perc[,1], n=2)
```

Southwest Airlines (WN), EVA Air(EV) and Delta airlines(DL) had maximum percentage of flights delayed in 2008,while US Airways (US),Endeavor Air(9E) and Mesa Airlines (YV) had least percentage of flights delayed.

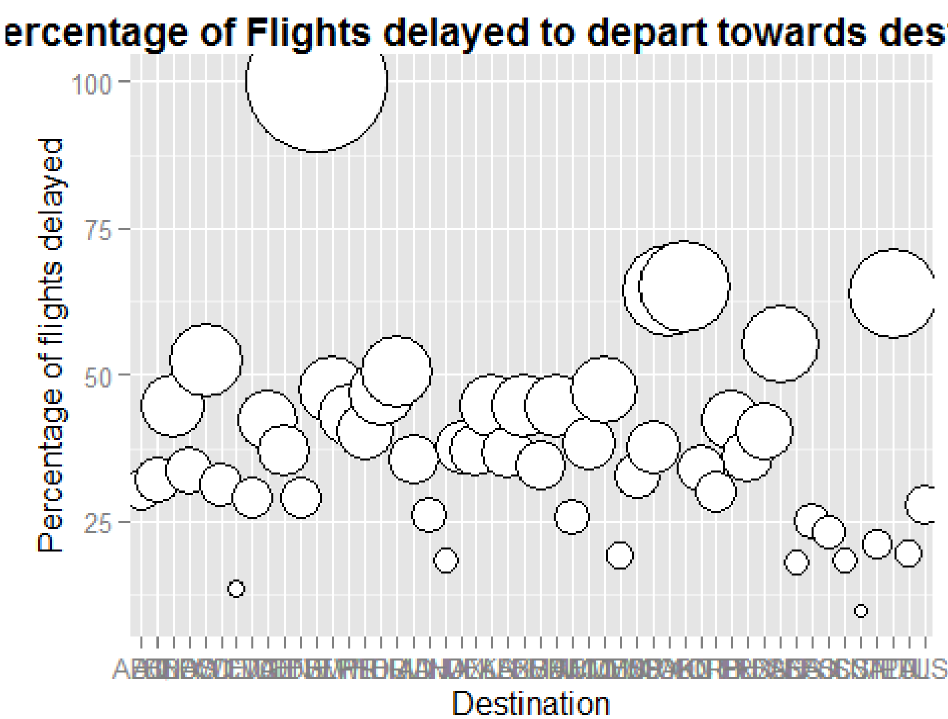
Now let's explore the destinations to which the carriers delay the departure time of the flight the most:

```
#Airlines delay count
dest.delay <- aggregate(outbound_Aus_delay$dummy,by=list(outbound_Aus_delay$D
est),FUN = sum, na.rm=TRUE)
names(dest.delay)[1] <- "Destination"
```

```
names(dest.delay)[2] <- "AnnualDelays"

dest.delay.time <- aggregate(outbound_Aus_delay$DepDelay,by=list(outbound_Aus_delay$Dest),FUN = mean, na.rm=TRUE)
names(dest.delay.time)[1] <- "Destination"
names(dest.delay.time)[2] <- "AverageDelayTime"

#Airlines total outbound count
dest.total <- aggregate(outbound_Aus$dummy,by=list(outbound_Aus$Dest),FUN = sum, na.rm=TRUE)
names(dest.total)[1] <- "Destination"
names(dest.total)[2] <- "AnnualFlights"

dest.delay.perc <- merge(dest.total,dest.delay, by = "Destination")
dest.delay.perc$per.delays <-round(dest.delay.perc$AnnualDelays/dest.delay.perc$AnnualFlights,3)*100

#par(mar=c(5.1,4.1,4.1,2.1))
```

*Percentage of Flights delayed to depart by destination:*

```
ggplot(data=dest.delay.perc, aes(x= Destination, y= per.delays)) + geom_point( size= dest.delay.perc$per.delays/4, shape=21, fill= "white") + scale_size_area() + ggtitle("Percentage of Flights delayed to depart towards destination") + theme(plot.title = element_text(lineheight=1.2, face="bold")) + xlab("Destination") + ylab("Percentage of flights delayed")
```



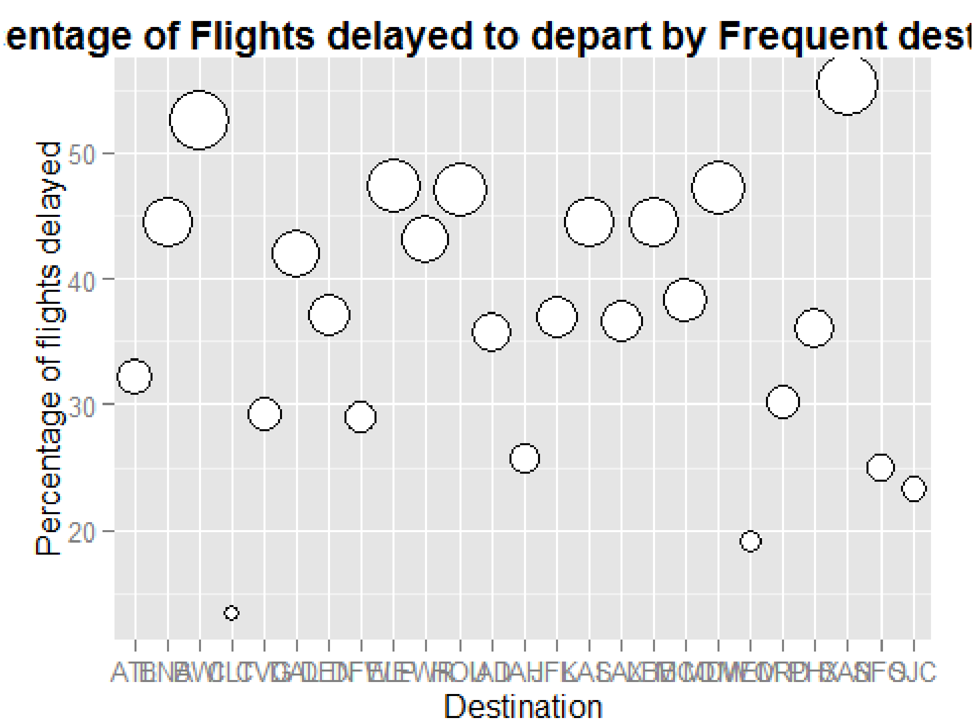
There was single flight scheduled to Des Moines International Airport, Iowa in 2008 which was delayed. Flight to Oakland International Airport,Calfornia (236 flights scheduled out of Austin in 2008) were delayed 64% of times, while flights to Will Rogers World Airport in Oklahoma (88 scheduled flights) were also delayed ~65% times. Likewise, we observe a lot of destinations with very fewer scheduled flights and longer departure delays.

Hence, to avoid this bias,let's have a look at the delay density with respect to flights to frequent destinations.

We shall look at destinations which had over 579 flights scheduled to departure in 2008. 579 is the median of scheduled flights to any destination out of Austin in that year.

*Percentage of Flights delayed to depart by Frequent destinations:*

```
#Only frequent destinations
ggplot(data=subset(dest.delay.perc,dest.delay.perc$AnnualFlights> median(dest.delay.perc$AnnualFlights)), aes(x= Destination, y= per.delays)) + geom_point( size= subset(dest.delay.perc,dest.delay.perc$AnnualFlights> median(dest.delay.perc$AnnualFlights))$per.delays/5 , shape=21, fill= "white") + scale_size_area() + ggtitle("Percentage of Flights delayed to depart by Frequent destinations") + theme(plot.title = element_text(lineheight=1.2, face="bold")) + xlab("Destination") + ylab("Percentage of flights delayed")
```



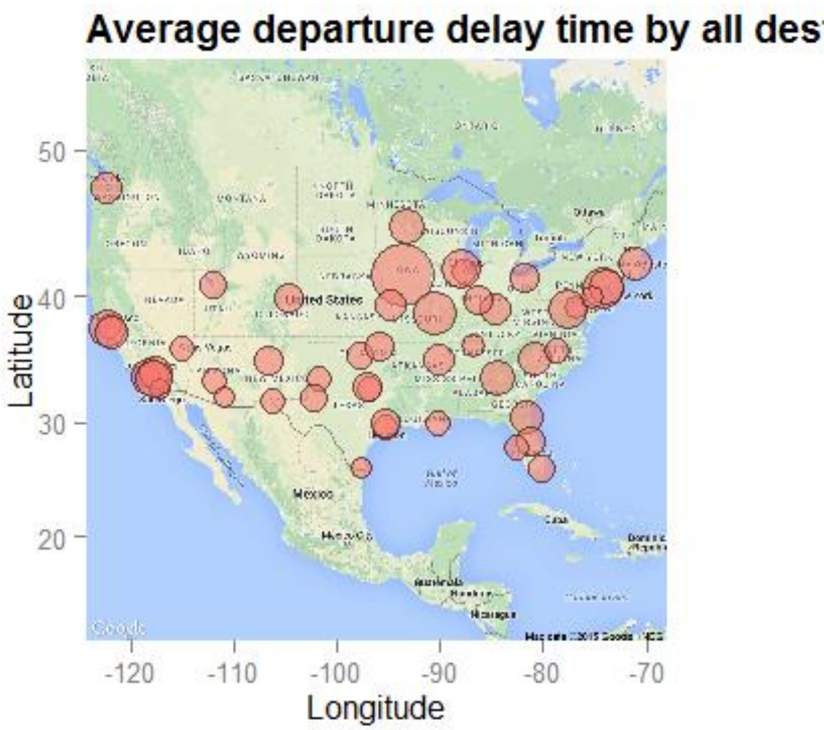

We observe that over 50% scheduled flights to Baltimore and San Diego (Frequent destinations) were delayed.

Now,let's observe the destinations affected the most by delay with respect to average departure delay time:

```
#install.packages("ggmap", dependencies = TRUE)
#Read airport codes
airport.codes <- read.csv("Airport_Codes_V1.csv")

#https://raw.githubusercontent.com/jpatokal/openflights/master/data/airports.
dat

#names(airport.codes)
names(airport.codes)[4] <- "Destination"
#names(dest.delay.perc)

#Merge it to our dataset
delay.map <- merge(dest.delay.perc,airport.codes,by = "Destination" ,x.all= T
RUE)
delay.map <- merge(delay.map,dest.delay.time, by = "Destination" ,x.all= TRUE
)

#sapply(delay.map,class)
#sum(is.na(delay.map))
```

*Average departure delay time by all destination Airports:*

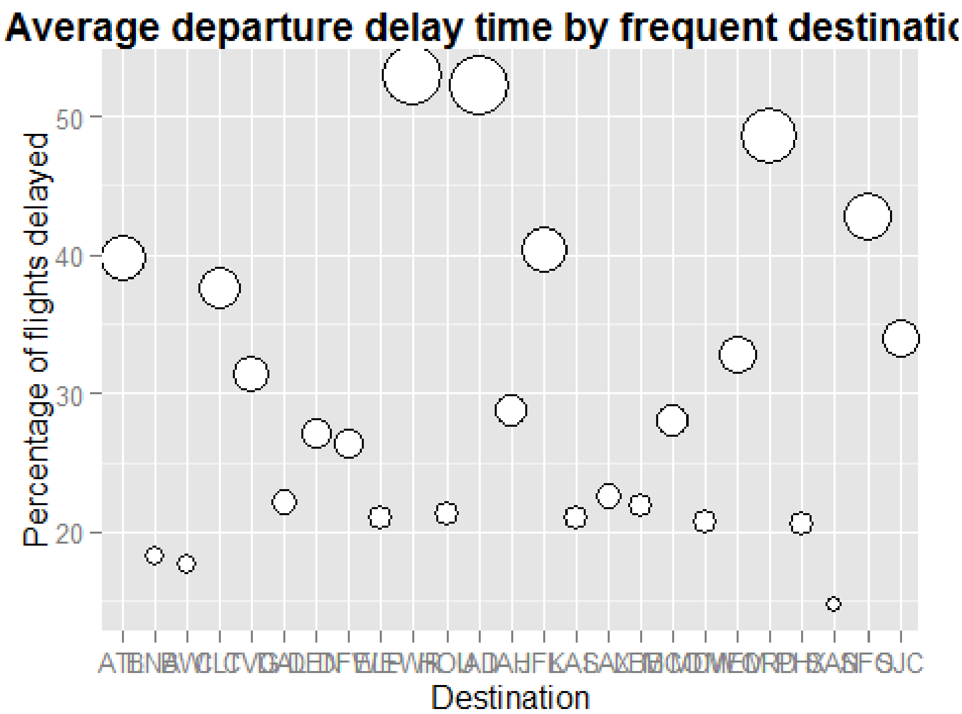
```
library(ggmap)

# getting the map
mapgilbert <- get_map(location = c(lon = mean(delay.map$Longitude), lat = mea
n(delay.map$Latitude)), zoom = 4,maptype = "roadmap", scale = 2)

## Map from URL : http://maps.googleapis.com/maps/api/staticmap?center=35.905
169,-96.219677&zoom=4&size=640x640&scale=2&maptype=roadmap&language=en-EN&sen
sor=false

# plotting the map with some points on it
ggmap(mapgilbert) + geom_point(data = delay.map, aes(x = Longitude, y = Latit
ude, fill = "red", alpha = 0.8), size = sqrt(delay.map$AverageDelayTime), sha
pe = 21) + guides(fill=FALSE, alpha=FALSE, size=FALSE) + labs(title="Average
departure delay time by all destination Airports", x="Longitude", y="Latitude
") + theme(plot.title = element_text(hjust = 0, vjust = 1, face = c("bold")))
```



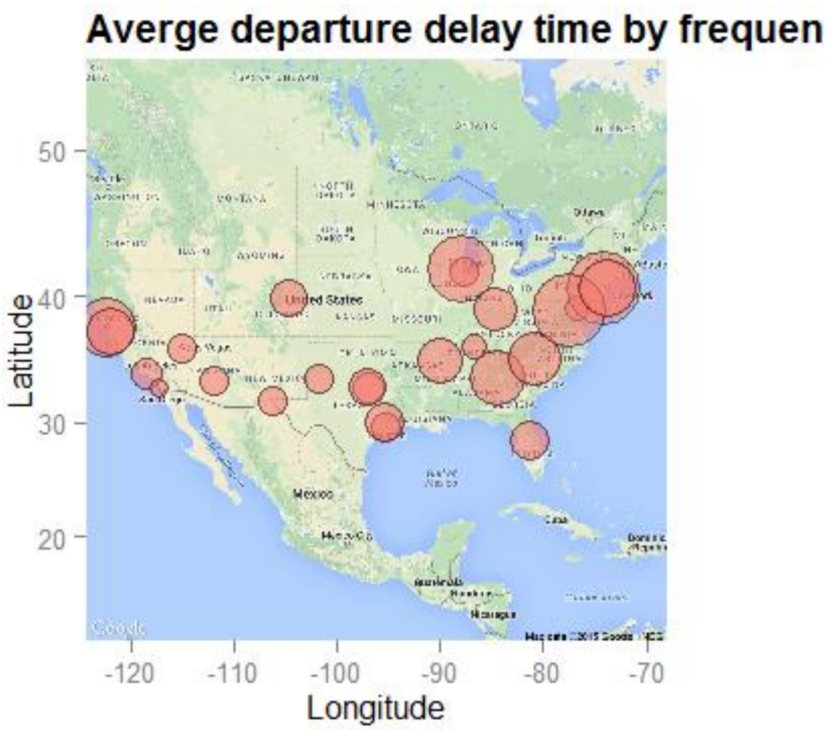

We notice that the flight to Iowa was delayed the most. But again, this is an outlier case due to a single scheduled flight. Hence,lets look at destinations with frequent flights out of Austin.

*Averge departure delay time by frequent destinations:*

```
#Only frequent destinations
ggplot(data=subset(delay.map,delay.map$AnnualFlights> median(delay.map$AnnualFlights)), aes(x= Destination, y= AverageDelayTime )) + geom_point( size= subset(delay.map,delay.map$AnnualFlights> median(delay.map$AnnualFlights))$AverageDelayTime/5 , shape=21, fill= "white") + scale_size_area() + ggtitle("Average departure delay time by frequent destinations") + theme(plot.title = element_text(lineheight=1.2, face="bold")) + xlab("Destination") + ylab("Percentage of flights delayed")
```




```
# getting the map
map <- get_map(location = c(lon = mean(delay.map$Longitude), lat = mean(delay
.map$Latitude)), zoom = 4,maptype = "roadmap", scale = 2)

## Map from URL : http://maps.googleapis.com/maps/api/staticmap?center=35.905
169,-96.219677&zoom=4&size=640x640&scale=2&maptype=roadmap&language=en-EN&sen
sor=false

# plotting the map with some points on it
ggmap(map) + geom_point(data = subset(delay.map,delay.map$AnnualFlights> medi
an(delay.map$AnnualFlights)), aes(x = Longitude, y = Latitude, fill = "red",
alpha = 0.8), size = subset(delay.map,delay.map$AnnualFlights> median(delay.m
ap$AnnualFlights))$AverageDelayTime/4, shape = 21) + guides(fill=FALSE, alpha
=FALSE, size=FALSE) + labs(title="Averge departure delay time by frequent des
tinations", x="Longitude", y="Latitude") + theme(plot.title = element_text(hj
ust = 0, vjust = 1, face = c("bold")))
```



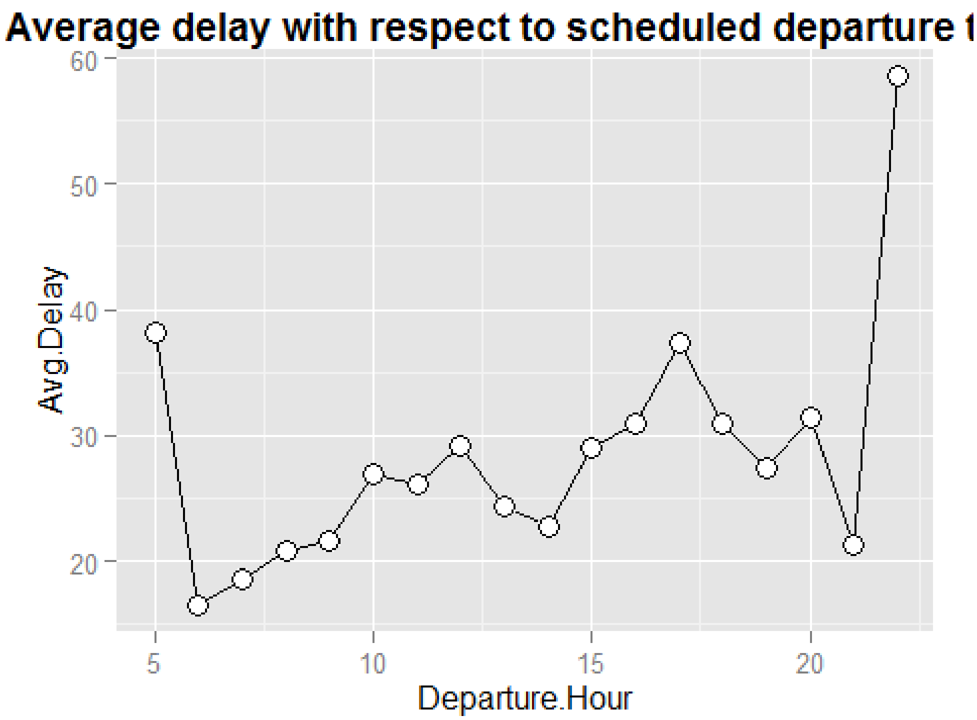

Flights to Washington Dulles International Airport, Washington DC and Newark Liberty International Airport, New Jersey were worst hit with respect to departure delay in 2008. They were closely followed by flights to O'Hare International Airport, Chicago.

Looking at time plots: Which is the worst time to fly out of Austin?

```
#Trends
by_hour <-aggregate(outbound_Aus_delay$DepDelay,by=list(outbound_Aus_delay$De
pHour), FUN=mean, na.rm=TRUE)
#names(by_DOW_hour)
names(by_hour)[1] <- "Departure.Hour"
names(by_hour)[2] <- "Avg.Delay"

by_hour <- by_hour[order(by_hour$Departure.Hour),]

#Remove departure hour 1
by_hour <- subset(by_hour,by_hour$Departure.Hour != 1)
```

*Average delay with respect to scheduled departure time:*

```
ggplot(data= by_hour, aes(x=Departure.Hour, y=Avg.Delay)) + geom_line() + geo
m_point( size=4, shape=21, fill="white") + ggtitle("Average delay with respec
t to scheduled departure time") + theme(plot.title = element_text(lineheight=
1.2, face="bold"))
```

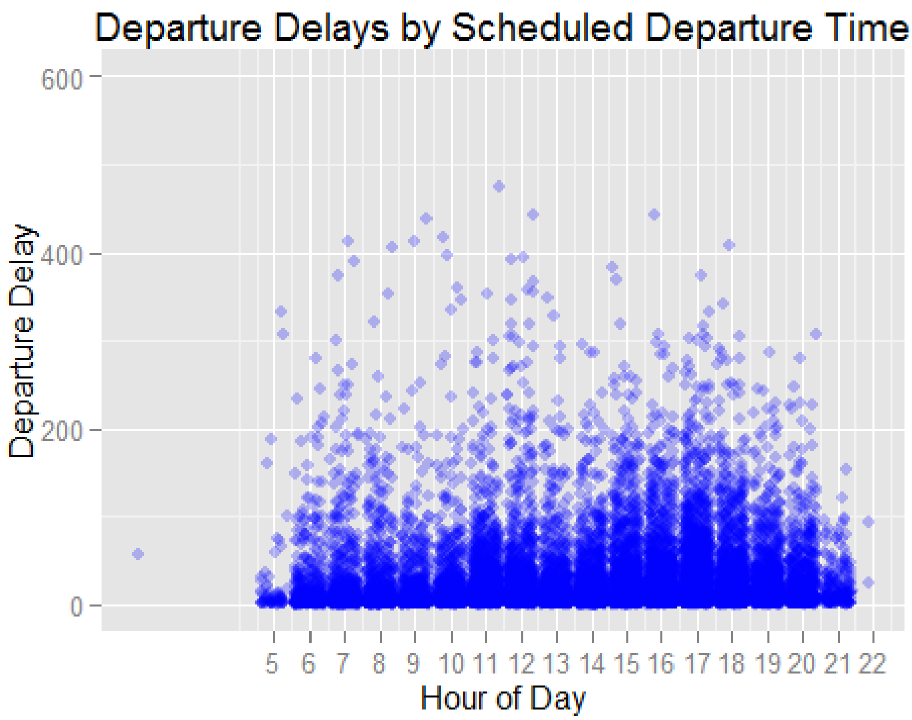



We observe that delay in departure gradually increases from 6 AM to 10 AM and drops till 2PM and increases it again util 4:30PM. Departure delay time starts dropping after 5PM till 10PM. Hence best time to fly would be early in the morning or late in the night.

Looking at an hourly density plot for departure delays:

*Departure Delays by Scheduled Departure Time - Density Plot:*

```
#Density plot for departure delay wrt scheduled departure
b1<-ggplot(outbound_Aus_delay, aes(DepHour, DepDelay)) + ylim(0, 600) +
  geom_jitter(alpha=I(1/4), col = "blue") +
  theme(legend.position = "none") +
  scale_x_continuous(breaks=seq(5,23)) +
  labs(x="Hour of Day",y="Departure Delay",title="Departure Delays by Schedul
ed Departure Time")
b1

## Warning: Removed 2 rows containing missing values (geom_point).
```

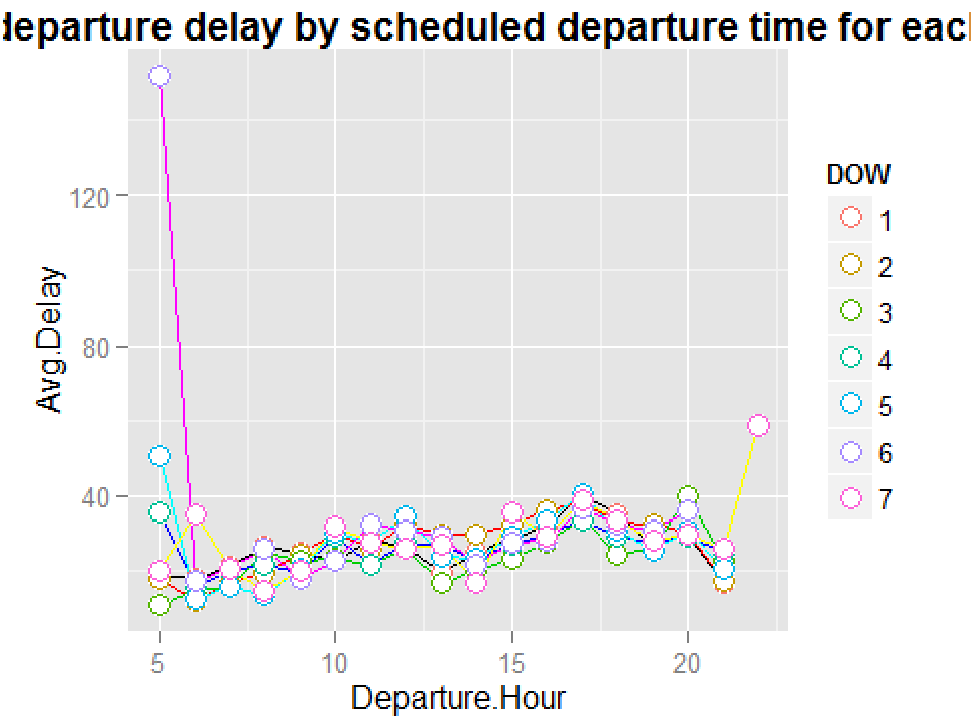



Diving into the data further to find out other time related trends:

```
#Trends by DOW
by_DOW_hour <-aggregate(outbound_Aus_delay$DepDelay,by=list(outbound_Aus_dela
y$DayOfWeek,outbound_Aus_delay$DepHour), FUN=mean, na.rm=TRUE)
#names(by_DOW_hour)
names(by_DOW_hour)[1] <- "DOW"
names(by_DOW_hour)[2] <- "Departure.Hour"
names(by_DOW_hour)[3] <- "Avg.Delay"

by_DOW_hour <- by_DOW_hour[order(by_DOW_hour$DOW,by_DOW_hour$Departure.Hour),
]

#Remove departure hour 1
by_DOW_hour <- subset(by_DOW_hour,by_DOW_hour$Departure.Hour != 1)
```

*Average departure delay by scheduled departure time for each DOW:*

```
ggplot(data= by_DOW_hour, aes(x=Departure.Hour, y=Avg.Delay, group = DOW, col
or = DOW)) + geom_line(colour = by_DOW_hour$DOW) + geom_point( size=4, shape=
21, fill="white") + ggtitle("Average departure delay by scheduled departure t
ime for each DOW") + theme(plot.title = element_text(lineheight=1.2, face="bo
ld"))
```



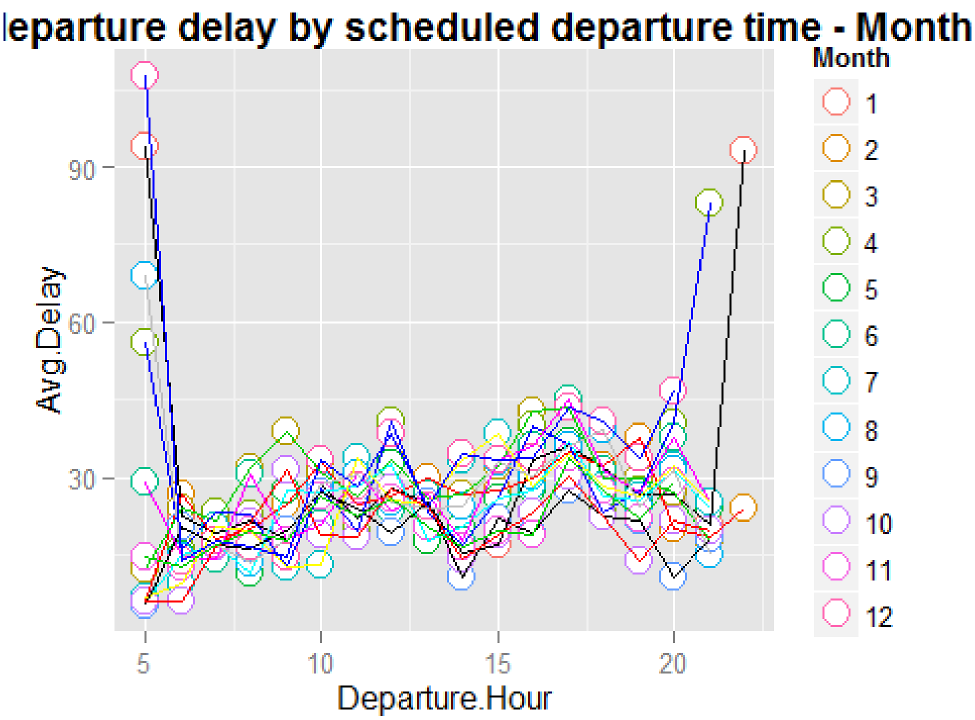

The day of the week trends show that throughout the year, daily delay patterns are fairly constant with delays increasing for flights scheduled to depart from 6 AM to 11 AM. The delay in flight plateaus till around 3 PM and starts increasing from 4 PM to 5:30 PM and drops back to an average delay of ~20 mins until 10 PM.

Hence, best time to fly would be either early in the morning before delays start peaking or late at night on any day of the week

```
#Month level
by_month_hour <-aggregate(outbound_Aus_delay$DepDelay,by=list(outbound_Aus_de
lay$Month,outbound_Aus_delay$DepHour), FUN=mean, na.rm=TRUE)
#names(by_month_hour)
names(by_month_hour)[1] <- "Month"
names(by_month_hour)[2] <- "Departure.Hour"
names(by_month_hour)[3] <- "Avg.Delay"

by_month_hour <- by_month_hour[order(by_month_hour$Month, by_month_hour$Depar
ture.Hour),]
by_month_hour <- subset(by_month_hour, by_month_hour$Departure.Hour != 1)
```

*Average departure delay by scheduled departure time - Month level:*

```
ggplot(data=by_month_hour, aes(x=Departure.Hour, y=Avg.Delay, group = Month,
color = Month)) + geom_point( size= 5, shape=21, fill="white") + geom_line(co
lour = by_month_hour$Month) + ggtitle("Average departure delay by scheduled d
eparture time - Month level") + theme(plot.title = element_text(lineheight=1.
2, face="bold"))
```

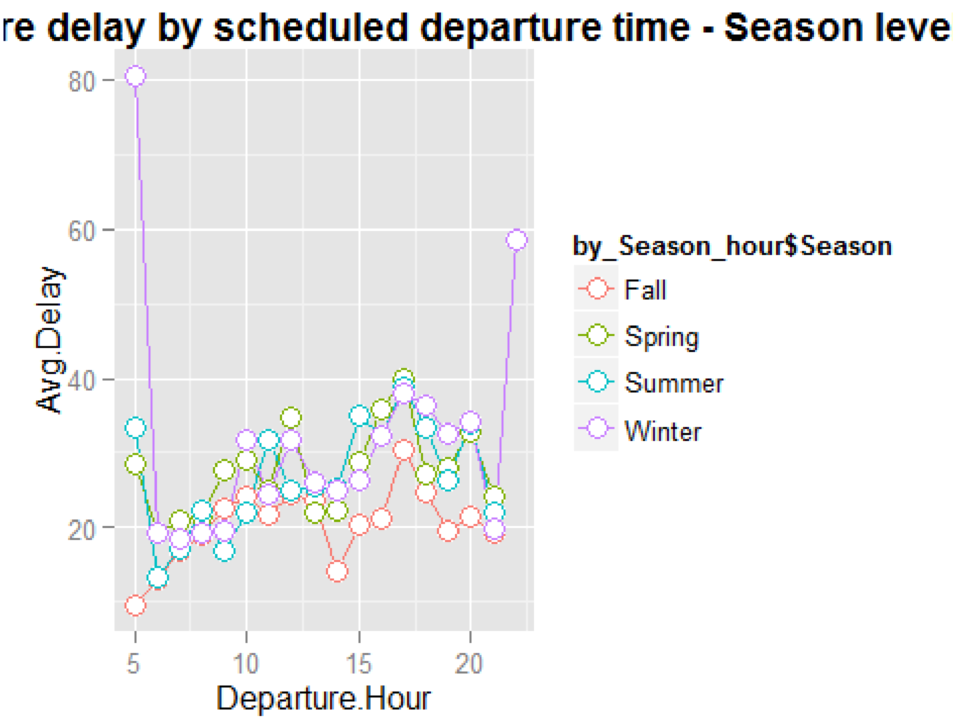



Departure by month show pretty much the same trends. Hence, let look at hourly departure delay by season:

```
#Season level
by_Season_hour <-aggregate(outbound_Aus_delay$DepDelay,by=list( outbound_Aus_
delay$Season,outbound_Aus_delay$DepHour), FUN=mean, na.rm=TRUE)
names(by_Season_hour)

## [1] "Group.1" "Group.2" "x"

names(by_Season_hour)[1] <- "Season"
names(by_Season_hour)[2] <- "Departure.Hour"
names(by_Season_hour)[3] <- "Avg.Delay"

by_Season_hour <- by_Season_hour[order(by_Season_hour$Season, by_Season_hour$
Departure.Hour),]
by_Season_hour <- subset(by_Season_hour, by_Season_hour$Departure.Hour != 1)
```

*Average departure delay by scheduled departure time - Season level:*

```
ggplot(data= by_Season_hour, aes(x=Departure.Hour, y=Avg.Delay, group = by_Se
ason_hour$Season, color = by_Season_hour$Season)) + geom_line(aes(colour = by
_Season_hour$Season)) + geom_point( size=4, shape=21, fill="white") + ggtitle
("Average departure delay by scheduled departure time - Season level") + them
e(plot.title = element_text(lineheight=1.2, face="bold"))
```




Flights are delayed less in Fall than in other seasons. Flights scheduled to departure from 6AM till 10AM experience delay gradually. The delay plateaus and then increases in he evening from 4PM to 6PM and then reduces to 20 mins average around 10 PM

The following plot tries to figure out if any particular airlines get affected the most by seasonal changes:

```
#Season and airlines level
by_Season_carrier_hour <-aggregate(outbound_Aus_delay$DepDelay,by=list( outbo
und_Aus_delay$Season,outbound_Aus_delay$DepHour,outbound_Aus_delay$UniqueCarr
ier), FUN=mean, na.rm=TRUE)
names(by_Season_carrier_hour)

## [1] "Group.1" "Group.2" "Group.3" "x"

names(by_Season_carrier_hour)[1] <- "Season"
names(by_Season_carrier_hour)[2] <- "Departure.Hour"
names(by_Season_carrier_hour)[3] <- "UniqueCarrier"
names(by_Season_carrier_hour)[4] <- "Avg.Delay"

by_Season_carrier_hour <- by_Season_hour[order(by_Season_carrier_hour$UniqueC
arrier,by_Season_carrier_hour$Season, by_Season_carrier_hour$Departure.Hour),
]
by_Season_carrier_hour <- subset(by_Season_carrier_hour, by_Season_carrier_ho
ur$Departure.Hour != 1)
```

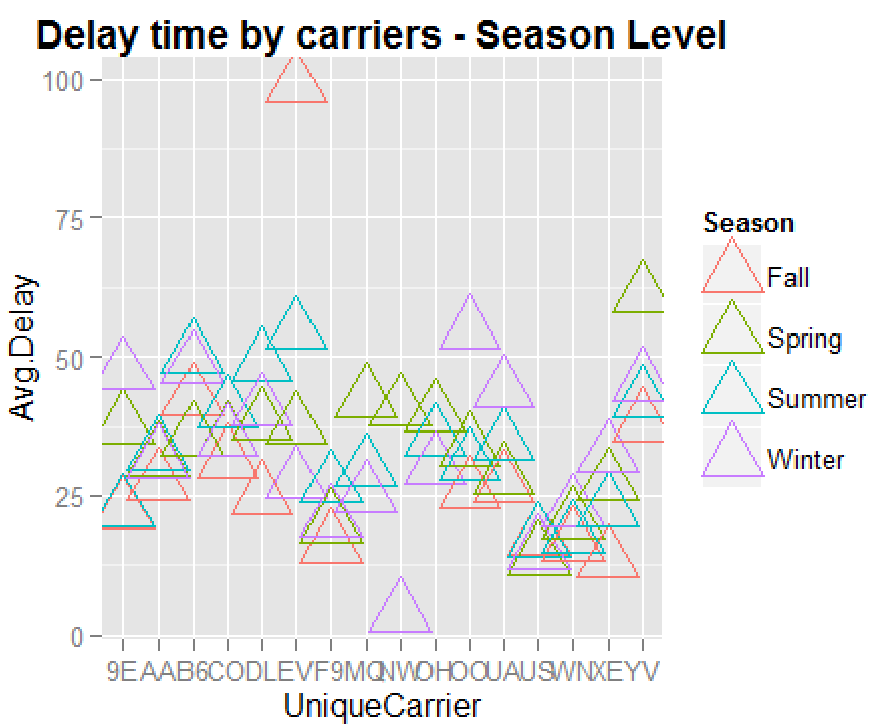
```
by_Season_carrier <-aggregate(outbound_Aus_delay$DepDelay,by=list( outbound_A
us_delay$Season,outbound_Aus_delay$UniqueCarrier), FUN=mean, na.rm=TRUE)
names(by_Season_carrier)
```

```
## [1] "Group.1" "Group.2" "x"
```

```
names(by_Season_carrier)[1] <- "Season"
names(by_Season_carrier)[2] <- "UniqueCarrier"
names(by_Season_carrier)[3] <- "Avg.Delay"

by_Season_carrier <- by_Season_carrier[order(by_Season_carrier$UniqueCarrier,
by_Season_carrier$Season),]
```

*Delay time by carriers - Season Level:*

```
ggplot(data= by_Season_carrier, aes(x=UniqueCarrier, y=Avg.Delay, group = Sea
son, color = Season)) + geom_point( size=8, shape=2, fill="white") + ggtitle(
"Delay time by carriers - Season Level") + theme(plot.title = element_text(li
neheight=1.2, face="bold"))
```




At a cursory glance, most of the airlines have a shorter departure delay in fall and longest in summer or winter.

## Summary :

The best time to fly out of Austin (to avoid departure delay) would be early in the morning or late in the night.Average departure delay is lesser in fall as compared to other seasons. Flights to Washington DC, New Jersey and Chicago have the maximum departure delay time as compared to flights flying to other places in United States. Maximum percentage of scheduled flights to Baltimore and San Diego are delayed. Southwest Airlines (WN), EVA Air(EV) and Delta airlines(DL) had maximum percentage of flights delayed in 2008,while US Airways (US),Endeavor Air(9E) and Mesa Airlines (YV) had least percentage of flights delayed.

# HW2-Q2

## Trying Naïve Bayes and Random Forest to classify text documents to authors

### Model 1: Naive Bayes

Creating the training dataset:

```
library(tm)

## Loading required package: NLP

readerPlain = function(fname){
  readPlain(elem=list(content=readLines(fname)),
            id=fname, language='en') }

#Training dataset

author_dirs = Sys.glob('C:/Users/Dhwani/Documents/Coursework/Summer - Predict
ive Analytics/STA380/STA380/data/ReutersC50/C50train/*')

file_list = NULL
labels = NULL

for(author in author_dirs) {
  author_name = substring(author, first=107)
  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)))
}

all_docs = lapply(file_list, readerPlain)
names(all_docs) = file_list
names(all_docs) = sub('.txt', '', names(all_docs))

train_corpus = Corpus(VectorSource(all_docs))
names(train_corpus) = file_list


train_corpus = tm_map(train_corpus, content_transformer(tolower))
train_corpus = tm_map(train_corpus, content_transformer(removeNumbers))
train_corpus = tm_map(train_corpus, content_transformer(removePunctuation))
```

```
train_corpus = tm_map(train_corpus, content_transformer(stripWhitespace))
train_corpus = tm_map(train_corpus, content_transformer(removeWords), stopwor
ds("en"))

DTM_train = DocumentTermMatrix(train_corpus)
DTM_train = removeSparseTerms(DTM_train, .99)
DTM_train
```

```
## <<DocumentTermMatrix (documents: 2500, terms: 3325)>>
## Non-/sparse entries: 376957/7935543
## Sparsity           : 95%
## Maximal term length: 20
## Weighting          : term frequency (tf)
```

```
X_train = as.matrix(DTM_train)
```

Running Naive Bayes on the training set:

```
smooth_count = 1/nrow(X_train)
w = rowsum(X_train + smooth_count, labels)
w = w/sum(w)
w = log(w)
```

Creating the test dataset:

```
# Test Dataset
author_dirs = Sys.glob('C:/Users/Dhwani/Documents/Coursework/Summer - Predict
ive Analytics/STA380/STA380/data/ReutersC50/C50test/*')
# author_dirs = author_dirs[1:2]
file_list = NULL
test_labels = NULL
author_names = NULL

for(author in author_dirs) {
  author_name = substring(author, first=106)
  author_names = append(author_names, author_name)
  files_to_add = Sys.glob(paste0(author, '/*.txt'))
  file_list = append(file_list, files_to_add)
  test_labels = append(test_labels, rep(author_name, length(files_to_add)))
}

all_docs = lapply(file_list, readerPlain)
names(all_docs) = file_list
names(all_docs) = sub('.txt', '', names(all_docs))

test_corpus = Corpus(VectorSource(all_docs))
names(test_corpus) = file_list

test_corpus = tm_map(test_corpus, content_transformer(tolower))
test_corpus = tm_map(test_corpus, content_transformer(removeNumbers))
test_corpus = tm_map(test_corpus, content_transformer(removePunctuation))
```

```
test_corpus = tm_map(test_corpus, content_transformer(stripWhitespace))
test_corpus = tm_map(test_corpus, content_transformer(removeWords), stopwords
("en"))

# make sure that we have all the words from training set
DTM_test = DocumentTermMatrix(test_corpus, list(dictionary=colnames(DTM_train
)))
DTM_test
```

```
## <<DocumentTermMatrix (documents: 2500, terms: 3325)>>
## Non-/sparse entries: 375822/7936678
## Sparsity           : 95%
## Maximal term length: 20
## Weighting          : term frequency (tf)
```

```
X_test = as.matrix(DTM_test)
```

Run prediction on test matrix:

```
predict = NULL
for (i in 1:nrow(X_test)) {
  # get maximum Naive Bayes log probabilities
  max = -(Inf)
  author = NULL
  for (j in 1:nrow(w)) {
    result = sum(w[j,]*X_test[i,])
    if(result > max) {
      max = result
      author = rownames(w)[j]
    }
  }
  predict = append(predict, author)
}

predict_results = table(test_labels,predict)
correct = NULL
for (i in 1:nrow(predict_results)) {
  correct = append(correct, predict_results[i, i])
}

author.predict.correct = data.frame(author_names, correct)
author.predict.correct <- author.predict.correct[order(-correct),]
author.predict.correct$per.correct <- author.predict.correct$correct/50

author.predict.correct
```

```
##         author_names correct per.correct
## 29   LynnleyBrowning      50        1.00
## 11    FumikoFujisaki      48        0.96
## 16      JimGilchrist      46        0.92
## 36         NickLouth      46        0.92
```

```
## 33       MatthewBunce      45          0.90
## 38      PeterHumphrey      44          0.88
## 21        KarlPenhaul      43          0.86
## 3       AlexanderSmith      42          0.84
## 22          KeithWeir      41          0.82
## 40         RobinSidel      41          0.82
## 47     TheresePoletti      41          0.82
## 28     LynneO'Donnell      40          0.80
## 34      MichaelConnor      40          0.80
## 41       RogerFillion      40          0.80
## 1       AaronPressman      39          0.78
## 6         BradDorfman      39          0.78
## 20       JonathanBirt      39          0.78
## 12     GrahamEarnshaw      36          0.72
## 14         JanLopatka      36          0.72
## 39         PierreTran      36          0.72
## 48         TimFarrand      36          0.72
## 24      KevinMorrison      35          0.70
## 25      KirstinRidley      33          0.66
## 43       SarahDavison      32          0.64
## 45        SimonCowell      32          0.64
## 26 KouroshKarimkhany      31          0.62
## 27          LydiaZajc      31          0.62
## 19       JohnMastrini      30          0.60
## 30    MarcelMichelson      30          0.60
## 42        SamuelPerry      30          0.60
## 18           JoeOrtiz      29          0.58
## 10        EricAuchard      28          0.56
## 32         MartinWolk      27          0.54
## 23     KevinDrawbaugh      26          0.52
## 37    PatriciaCommins      26          0.52
## 2          AlanCrosby      22          0.44
## 5       BernardHickey      21          0.42
## 15      JaneMacartney      21          0.42
## 49         ToddNissen      21          0.42
## 17     JoWinterbottom      19          0.38
## 35         MureDickie      19          0.38
## 13  HeatherScoffield      17          0.34
## 31       MarkBendeich      17          0.34
## 7   DarrenSchuettler      13          0.26
## 8         DavidLawder      11          0.22
## 50       WilliamKazer      11          0.22
## 4     BenjaminKangLim      10          0.20
## 9       EdnaFernandes      10          0.20
## 44        ScottHillis       5          0.10
## 46           TanEeLyn       1          0.02

#Accuracy
sum(author.predict.correct$correct)/nrow(X_test)
```

```
## [1] 0.6024
```

The Naive Bayes algorithm gives us 60.24% accuracy. The model is unable to predict properly for the following authors - They have prediction accuracy of less than 25%:

DavidLawder
WilliamKazer
BenjaminKangLim
EdnaFernandes
ScottHillis
TanEeLyn

### Model 2: Random Forest

```
#Adding words present in training but not in test, to our test dataframe for
Random Forest to run

DTM_test = as.matrix(DTM_test)
DTM_train = as.matrix(DTM_train)
DTM_train_df = as.data.frame(DTM_train)

common <- data.frame(DTM_test[,intersect(colnames(DTM_test), colnames(DTM_tra
in))])
all.train <- read.table(textConnection(""), col.names = colnames(DTM_train),
colClasses = "integer")

library(plyr)
DTM_test_clean = rbind.fill(common, all.train)
DTM_test_df = as.data.frame(DTM_test_clean)

library(randomForest)

## randomForest 4.6-10
## Type rfNews() to see new features/changes/bug fixes.

rf.model = randomForest(x=DTM_train_df, y=as.factor(labels), mtry=4, ntree=30
0)
rf.pred = predict(rf.model, data=DTM_test_clean)

rf.table = as.data.frame(table(rf.pred,labels))

#install.packages("caret", dependencies = TRUE)
library("caret")

## Warning: package 'caret' was built under R version 3.2.2

## Loading required package: lattice
## Loading required package: ggplot2
##
## Attaching package: 'ggplot2'
```

```
##
## The following object is masked from 'package:NLP':
##
##     annotate
```

```
rf.confusion.matrix = confusionMatrix(table(rf.pred,test_labels))
rf.confusion.matrix$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.7340000      0.7285714      0.7162117      0.7512385      0.0200000
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
```

```
rf.confusion.matrix.df = as.data.frame(rf.confusion.matrix$byClass)
rf.confusion.matrix.df[order(-rf.confusion.matrix.df$Sensitivity),1:2]
```

```
##                           Sensitivity Specificity
## Class: JimGilchrist              1.00   0.9824490
## Class: LynnleyBrowning           1.00   0.9963265
## Class: FumikoFujisaki            0.98   0.9967347
## Class: KarlPenhaul               0.98   0.9951020
## Class: LynneO'Donnell            0.98   0.9922449
## Class: KouroshKarimkhany         0.94   0.9783673
## Class: LydiaZajc                 0.94   0.9963265
## Class: RogerFillion              0.92   0.9951020
## Class: JoWinterbottom            0.90   0.9897959
## Class: MarcelMichelson           0.90   0.9963265
## Class: MarkBendeich              0.90   0.9955102
## Class: MatthewBunce              0.90   0.9987755
## Class: PeterHumphrey             0.90   0.9869388
## Class: TimFarrand                0.90   0.9930612
## Class: DavidLawder               0.88   0.9934694
## Class: GrahamEarnshaw            0.88   0.9942857
## Class: AlanCrosby                0.86   0.9942857
## Class: DarrenSchuettler          0.86   0.9987755
## Class: HeatherScoffield          0.86   0.9987755
## Class: RobinSidel                0.86   0.9967347
## Class: AaronPressman             0.82   0.9971429
## Class: NickLouth                 0.82   0.9959184
## Class: JanLopatka                0.80   0.9934694
## Class: PierreTran                0.80   0.9967347
## Class: BernardHickey             0.76   0.9963265
## Class: JonathanBirt              0.72   0.9955102
## Class: MartinWolk                0.72   0.9983673
## Class: MichaelConnor             0.72   0.9975510
## Class: BenjaminKangLim           0.70   0.9775510
## Class: KeithWeir                 0.70   0.9955102
## Class: ToddNissen                0.70   0.9942857
## Class: EdnaFernandes             0.68   0.9955102
## Class: KevinDrawbaugh            0.68   0.9914286
## Class: JoeOrtiz                  0.66   0.9971429
```

```
## Class: KirstinRidley          0.66   0.9971429
## Class: PatriciaCommins        0.66   0.9967347
## Class: SimonCowell            0.66   0.9963265
## Class: JohnMastrini           0.62   0.9963265
## Class: AlexanderSmith         0.60   0.9971429
## Class: SamuelPerry            0.58   0.9946939
## Class: TheresePoletti         0.56   0.9934694
## Class: EricAuchard            0.54   0.9967347
## Class: KevinMorrison          0.54   0.9967347
## Class: BradDorfman            0.48   0.9975510
## Class: SarahDavison           0.42   0.9979592
## Class: JaneMacartney          0.40   0.9971429
## Class: MureDickie             0.38   0.9967347
## Class: TanEeLyn               0.36   0.9955102
## Class: WilliamKazer           0.36   0.9934694
## Class: ScottHillis            0.26   0.9930612
```

Random Forest model shows an accuracy of 73.4% and has trouble predicting the following authors correctly:

JaneMacartney
MureDickie
TanEeLyn
WilliamKazer
ScottHillis

Random Forest shows better accuracy than Naïve Bayes. Also, Random Forest would be a better approach to classify authors because Random Forest takes care of correlated features, while Naïve Bayes assumes all the features (words) to be independent.

# HW2-Q3

## 3. Association Rule Mining

```r
library(arules)
```

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

```r
no_col <- max(count.fields("groceries.txt", sep = "\t"))
groceries <- read.table("groceries.txt",sep="\t",fill=TRUE,col.names=1:no_col
)
```

```r
groceries$basketid <- rownames(groceries)
#head(groceries)
#tail(groceries)
#class(groceries)

#install.packages("splitstackshape", dependencies = TRUE)
library(splitstackshape)
```

```
## Loading required package: data.table
```

```r
groceries.map <- cSplit(groceries, "X1", ",", direction = "long")

#groceries.list <- split(groceries, seq(nrow(groceries)))
groceries.list <- split(groceries.map$X1, f = groceries.map$basketid)
groceries.list <- lapply(groceries.list, unique)

## Cast this variable as a special arules "transactions" class.
groceries.trans <- as(groceries.list, "transactions")

# Now run the 'apriori' algorithm
# Look at rules with support > .01 & confidence >.3 & length (# grocery items
) <= 20
grocrules <- apriori(groceries.trans, parameter=list(support=.005, confidence
=.3, maxlen=8))
```

```
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport support minlen maxlen
##         0.3    0.1    1 none FALSE            TRUE   0.005      1      8
##  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 ... [120 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [482 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```r
# Look at the output
#inspect(grocrules)

## Choose a subset
inspect(subset(grocrules, subset=support > .05))
```

```
##   lhs                      rhs            support confidence     lift
## 1 {yogurt}              => {whole milk} 0.05602440  0.4016035 1.571735
## 2 {rolls/buns}          => {whole milk} 0.05663447  0.3079049 1.205032
## 3 {other vegetables} => {whole milk} 0.07483477  0.3867578 1.513634
```

Products like yogurt, rolls/buns and other vegetables have a high support - which means these products are purchased more often than the other products.

```
inspect(subset(grocrules, subset = confidence > 0.63))
```

```
##   lhs                        rhs                    support confidence     li
ft
## 1 {curd,
##    tropical fruit}     => {whole milk}       0.006507372  0.6336634 2.4799
36
## 2 {butter,
##    whipped/sour cream} => {whole milk}       0.006710727  0.6600000 2.5830
08
## 3 {butter,
##    root vegetables}    => {whole milk}       0.008235892  0.6377953 2.4961
07
## 4 {butter,
##    yogurt}             => {whole milk}       0.009354347  0.6388889 2.5003
87
## 5 {pip fruit,
##    whipped/sour cream} => {whole milk}       0.005998983  0.6483516 2.5374
21
## 6 {other vegetables,
##    pip fruit,
##    root vegetables}    => {whole milk}       0.005490595  0.6750000 2.6417
13
## 7 {citrus fruit,
##    root vegetables,
##    whole milk}         => {other vegetables} 0.005795628  0.6333333 3.2731
65
## 8 {root vegetables,
##    tropical fruit,
##    yogurt}             => {whole milk}       0.005693950  0.7000000 2.7395
54
```

Confidence is the ratio of the number of transactions that include all items in the consequent as well as the antecedent to the number of transactions that include all items in the antecedent.

Looking at products whose purchase likelihood increases by atleast 63% provided we purchased groceries in antecedent, we observe that a basket which contains fruits, root vegetables, yogurt, cream etc. is more likely to have whole milk as well. Hence, a store can work on its product placement strategy and place all the commonly purchased items like fruits, vegetables and dairy products together.

```
inspect(subset(grocrules, subset=lift > 3))
```

```
##    lhs                           rhs                      support confidence
lift
## 1  {herbs}                    => {root vegetables}  0.007015760  0.4312500 3.
956477
## 2  {beef}                     => {root vegetables}  0.017386884  0.3313953 3.
040367
## 3  {onions,
##     root vegetables}         => {other vegetables} 0.005693950  0.6021505 3.
112008
## 4  {onions,
##     other vegetables}        => {root vegetables}  0.005693950  0.4000000 3.
669776
## 5  {chicken,
##     whole milk}              => {root vegetables}  0.005998983  0.3410405 3.
128855
## 6  {frozen vegetables,
##     other vegetables}        => {root vegetables}  0.006100661  0.3428571 3.
145522
## 7  {beef,
##     other vegetables}        => {root vegetables}  0.007930859  0.4020619 3.
688692
## 8  {beef,
##     whole milk}              => {root vegetables}  0.008032537  0.3779904 3.
467851
## 9  {curd,
##     tropical fruit}          => {yogurt}           0.005287239  0.5148515 3.
690645
## 10 {butter,
##     other vegetables}        => {root vegetables}  0.006609049  0.3299492 3.
027100
## 11 {domestic eggs,
##     other vegetables}        => {root vegetables}  0.007320793  0.3287671 3.
016254
## 12 {pip fruit,
##     whipped/sour cream}      => {other vegetables} 0.005592272  0.6043956 3.
123610
## 13 {tropical fruit,
##     whipped/sour cream}      => {yogurt}           0.006202339  0.4485294 3.
215224
## 14 {citrus fruit,
##     pip fruit}               => {tropical fruit}   0.005592272  0.4044118 3.
854060
## 15 {pip fruit,
##     root vegetables}         => {tropical fruit}   0.005287239  0.3398693 3.
238967
## 16 {pip fruit,
##     yogurt}                  => {tropical fruit}   0.006405694  0.3559322 3.
392048
## 17 {other vegetables,
##     pip fruit}               => {tropical fruit}   0.009456024  0.3618677 3.
```

```
448613
## 18 {citrus fruit,
##     root vegetables}       => {tropical fruit}   0.005693950  0.3218391 3.
067139
## 19 {citrus fruit,
##     root vegetables}       => {other vegetables} 0.010371124  0.5862069 3.
029608
## 20 {citrus fruit,
##     other vegetables}      => {root vegetables}  0.010371124  0.3591549 3.
295045
## 21 {rolls/buns,
##     shopping bags}         => {sausage}          0.005998983  0.3072917 3.
270794
## 22 {root vegetables,
##     yogurt}                => {tropical fruit}   0.008134215  0.3149606 3.
001587
## 23 {root vegetables,
##     tropical fruit}        => {other vegetables} 0.012302999  0.5845411 3.
020999
## 24 {other vegetables,
##     tropical fruit}        => {root vegetables}  0.012302999  0.3427762 3.
144780
## 25 {fruit/vegetable juice,
##     other vegetables,
##     whole milk}            => {yogurt}           0.005083884  0.4854369 3.
479790
## 26 {other vegetables,
##     whipped/sour cream,
##     whole milk}            => {root vegetables}  0.005185562  0.3541667 3.
249281
## 27 {pip fruit,
##     root vegetables,
##     whole milk}            => {other vegetables} 0.005490595  0.6136364 3.
171368
## 28 {other vegetables,
##     pip fruit,
##     whole milk}            => {root vegetables}  0.005490595  0.4060150 3.
724961
## 29 {citrus fruit,
##     root vegetables,
##     whole milk}            => {other vegetables} 0.005795628  0.6333333 3.
273165
## 30 {citrus fruit,
##     other vegetables,
##     whole milk}            => {root vegetables}  0.005795628  0.4453125 4.
085493
## 31 {root vegetables,
##     tropical fruit,
##     whole milk}            => {yogurt}           0.005693950  0.4745763 3.
401937
```

```
## 32 {tropical fruit,
##     whole milk,
##     yogurt}                => {root vegetables}  0.005693950  0.3758389 3.
448112
## 33 {root vegetables,
##     whole milk,
##     yogurt}                => {tropical fruit}   0.005693950  0.3916084 3.
732043
## 34 {root vegetables,
##     tropical fruit,
##     whole milk}            => {other vegetables} 0.007015760  0.5847458 3.
022057
## 35 {other vegetables,
##     tropical fruit,
##     whole milk}            => {root vegetables}  0.007015760  0.4107143 3.
768074
## 36 {other vegetables,
##     tropical fruit,
##     whole milk}            => {yogurt}           0.007625826  0.4464286 3.
200164
## 37 {other vegetables,
##     whole milk,
##     yogurt}                => {tropical fruit}   0.007625826  0.3424658 3.
263712
## 38 {other vegetables,
##     whole milk,
##     yogurt}                => {root vegetables}  0.007829181  0.3515982 3.
225716
## 39 {other vegetables,
##     rolls/buns,
##     whole milk}            => {root vegetables}  0.006202339  0.3465909 3.
179778

#high lift and high support
inspect(subset(grocrules, subset=lift > 3 & support > .01))

##   lhs                      rhs                    support confidence     lift
## 1 {beef}                => {root vegetables}  0.01738688  0.3313953 3.040367
## 2 {citrus fruit,
##    root vegetables}     => {other vegetables} 0.01037112  0.5862069 3.029608
## 3 {citrus fruit,
##    other vegetables} => {root vegetables}  0.01037112  0.3591549 3.295045
## 4 {root vegetables,
##    tropical fruit}    => {other vegetables} 0.01230300  0.5845411 3.020999
## 5 {other vegetables,
##    tropical fruit}    => {root vegetables}  0.01230300  0.3427762 3.144780
```

Lift is the ratio of Confidence to Expected Confidence. In other words, Lift is a value that gives us information about the increase in probability of the consequent given the antecedent part.

High lift indicates that relationship between antecedent and consequent is more significant that what would be expected if two sets were independent. Larger the lift, higher is the association.

For products which are purchased frequently (high support), like beef goes with root vegetables. Root vegetables, tropical fruits, citrus fruits and other vegetables go hand in hand.

```
inspect(subset(grocrules, subset=support > .01 & confidence > 0.55))
```

```
##   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
```

The association exercise on grocery basket data can help a store manager place his products more accurately, or maybe prescribe coupons based on the association rules. Since, dairy products, citrus fruits, vegetables go well together with whole milk, a store manager can place all these products together so that it’s easy for a shopper to pick his groceries.

Incase of a new whole milk brand launch, the manager can target the consumer who are more likely to purchase other products with coupons or marketing campaign for the newer brand.