Final Exam STA380

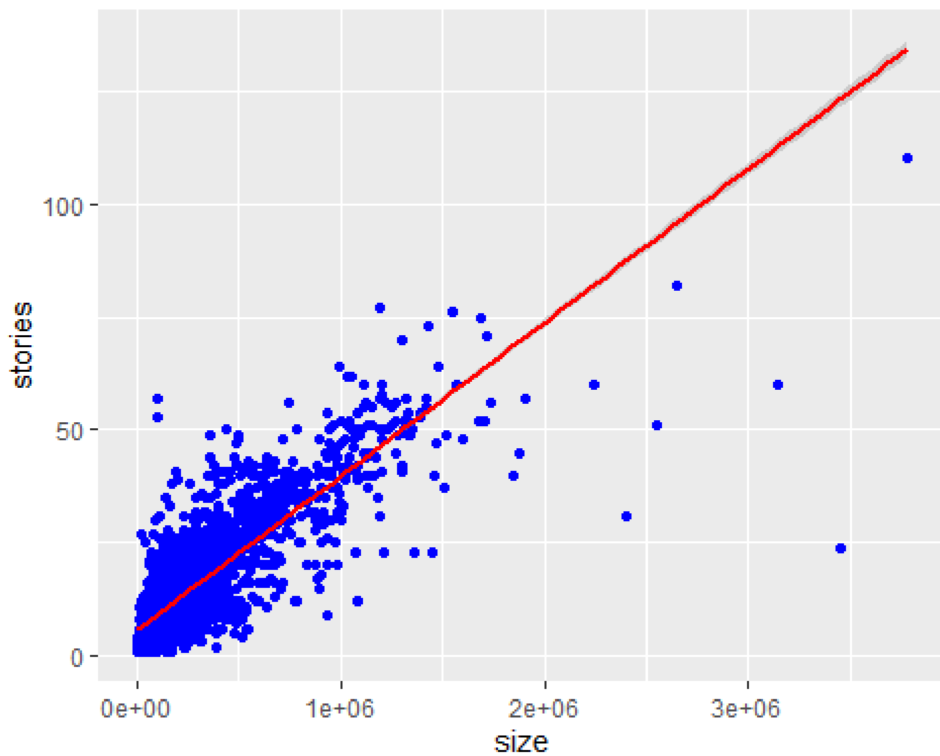
MacAlistair Husted(MWH2359)

8/17/2020

Question 1

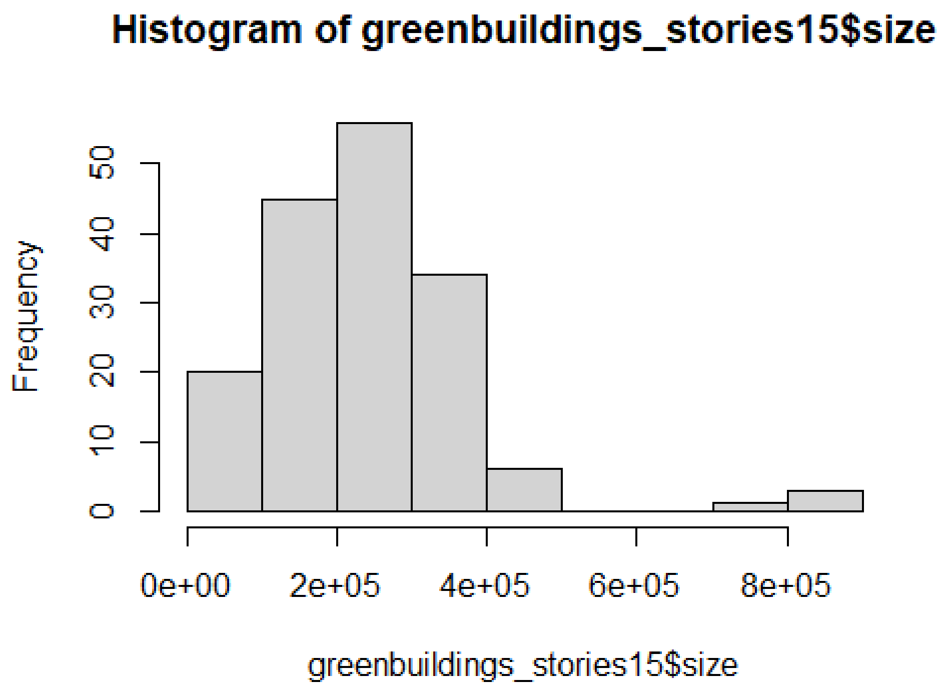
Before conducting any analysis, I first cleaned the data. The procedure was quite similar to the stat guru's besides ensuring that there were no missing data in any of the columns by omitting NAs. Since the stat guru did not analyze green buildings vs. non-green buildings based on the number of stories of a building, I first looked into whether size and buildings are highly correlated. And thus, size could be considered a proxy for stories. Based on the results of the correlation, the size of the building and the number of stories are highly correlated. This is visualized the scatter plot created.

```
library(mosaic)
library(tidyverse)
library(ggplot2)
library(boot)
library(caret)
library(psych)
require(gridExtra)
library(reshape2)
library(plyr)
greenbuildings <-
read.csv("C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/greenbuildings
.csv")
greenbuildings$green_rating <- as.factor(greenbuildings$green_rating)
greenbuildings$amenities <- as.factor(greenbuildings$amenities)
greenbuildings$renovated <- as.factor(greenbuildings$renovated)
greenbuildings$Energystar <- as.factor(greenbuildings$Energystar)
greenbuildings$LEED <- as.factor(greenbuildings$LEED)
greenbuildings <- subset(greenbuildings,greenbuildings$leasing_rate > 0.1)
greenbuildings <- na.omit(greenbuildings)
#correlation of stories and size of building
cor(greenbuildings$size,greenbuildings$stories)

## [1] 0.8261416

ggplot(greenbuildings,aes(size, stories))+ geom_point(colour = "blue")+
  geom_smooth(method="lm", se=TRUE, fullrange=FALSE, level=0.95, color="red")
```



Now to determine if the square feet of 250000 that the stat guru used is correct. Based on the histogram of only 15 story buildings. The size of 15 story buildings is anywhere between 250000 and 35000. This is why we use only buildings that are between 250000 and 350000 to analyze further.

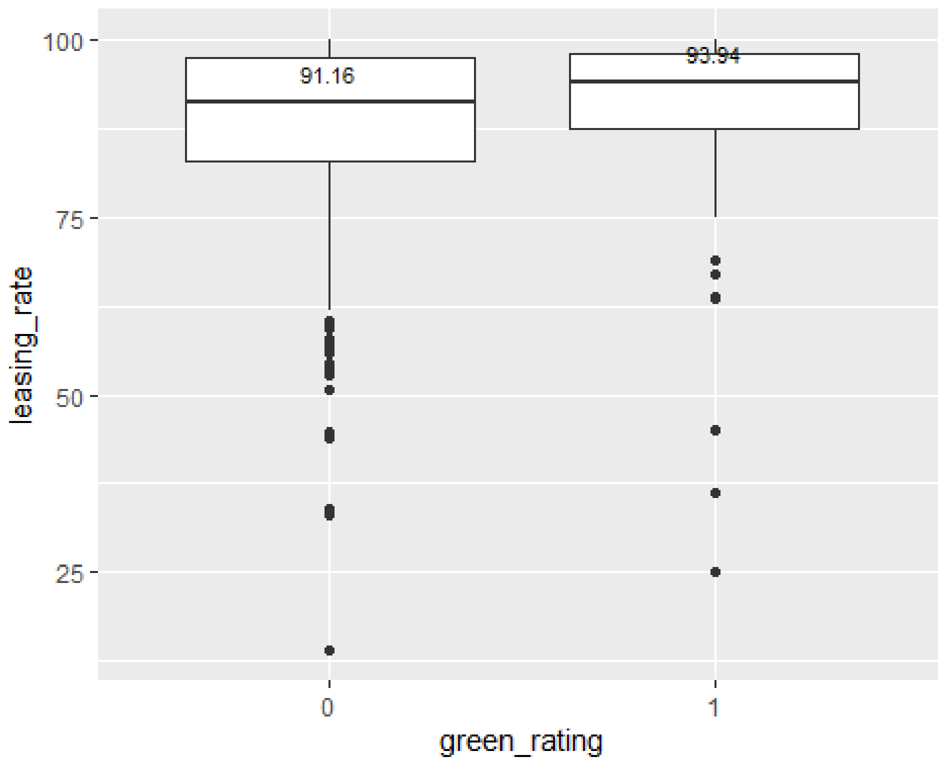
```
greenbuildings <-
read.csv("C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/greenbuildings
.csv")
greenbuildings$green_rating <- as.factor(greenbuildings$green_rating)
greenbuildings$amenities <- as.factor(greenbuildings$amenities)
greenbuildings$renovated <- as.factor(greenbuildings$renovated)
greenbuildings$Energystar <- as.factor(greenbuildings$Energystar)
greenbuildings$LEED <- as.factor(greenbuildings$LEED)
greenbuildings <- subset(greenbuildings,greenbuildings$leasing_rate > 0.1)
greenbuildings <- na.omit(greenbuildings)
greenbuildings_stories15 <- subset(greenbuildings, greenbuildings$stories ==
15)
hist(greenbuildings_stories15$size)
```



To determine whether her assumption that the green buildings would have 90% leasing rate was grounded in the data. I looked at leasing rate between green and non-green buildings via a boxplot. Based on the boxplot, it does seem that her assumption is grounded in the data as the median lease is very close to 90%, even slightly larger.

```
library(mosaic)
library(tidyverse)
library(ggplot2)
library(boot)
library(caret)
library(psych)
require(gridExtra)
library(reshape2)
library(plyr)
greenbuildings <-
read.csv("C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/greenbuildings
.csv")
greenbuildings$green_rating <- as.factor(greenbuildings$green_rating)
greenbuildings$amenities <- as.factor(greenbuildings$amenities)
greenbuildings$renovated <- as.factor(greenbuildings$renovated)
greenbuildings$Energystar <- as.factor(greenbuildings$Energystar)
greenbuildings$LEED <- as.factor(greenbuildings$LEED)
greenbuildings <- subset(greenbuildings,greenbuildings$leasing_rate > 0.1)
greenbuildings <- na.omit(greenbuildings)
subset_greenbuildings <- subset(greenbuildings, greenbuildings$size > 250000
& greenbuildings$size < 350000)
```

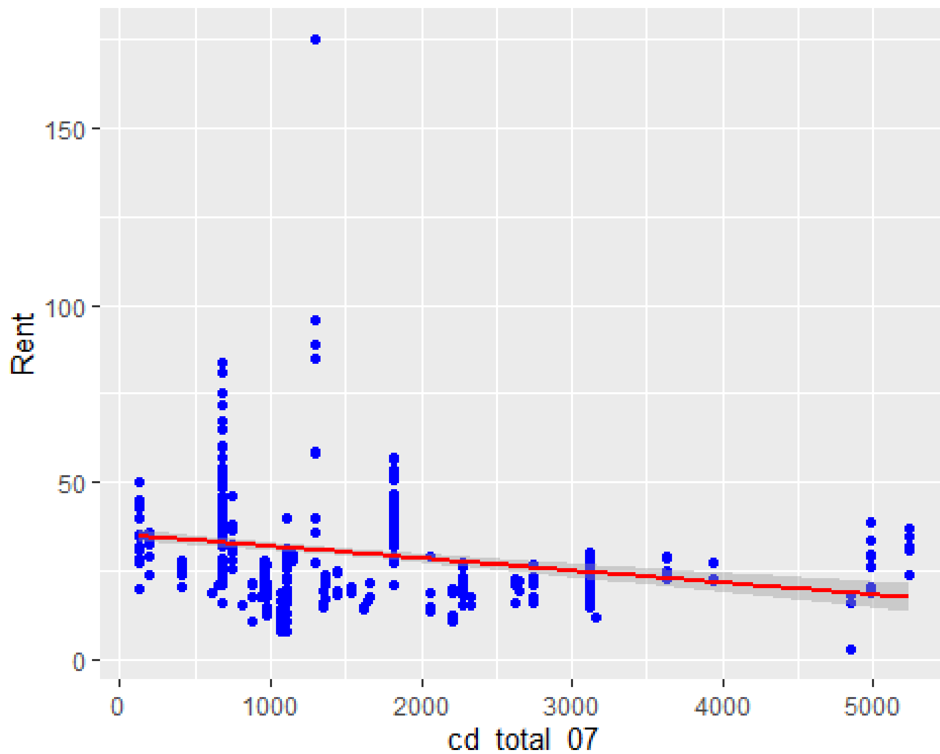
```
green_subset_medians <- ddply(subset_greenbuildings, .(green_rating),
summarise, median = round(median(leasing_rate),4))

ggplot(data=subset_greenbuildings,aes(x = green_rating, y = leasing_rate)) +
  geom_boxplot()+
  geom_text(data =green_subset_medians,aes(x = green_rating, y = median,
label = median), size = 3, vjust = -1.1)
```



Now, I analyzed the data further to find possible confounding variables that the stat guru missed in her analysis. The first variable is cd_total_07 which is the number of cooling degree days in the region. This is an especially important variable as Austin is on average quite hot. This variable was negatively correlated with Rent.

```
library(mosaic)
library(tidyverse)
library(ggplot2)
library(boot)
library(caret)
library(psych)
require(gridExtra)
library(reshape2)
library(plyr)
greenbuildings <-
read.csv("C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/greenbuildings
.csv")
greenbuildings$green_rating <- as.factor(greenbuildings$green_rating)
greenbuildings$amenities <- as.factor(greenbuildings$amenities)
```

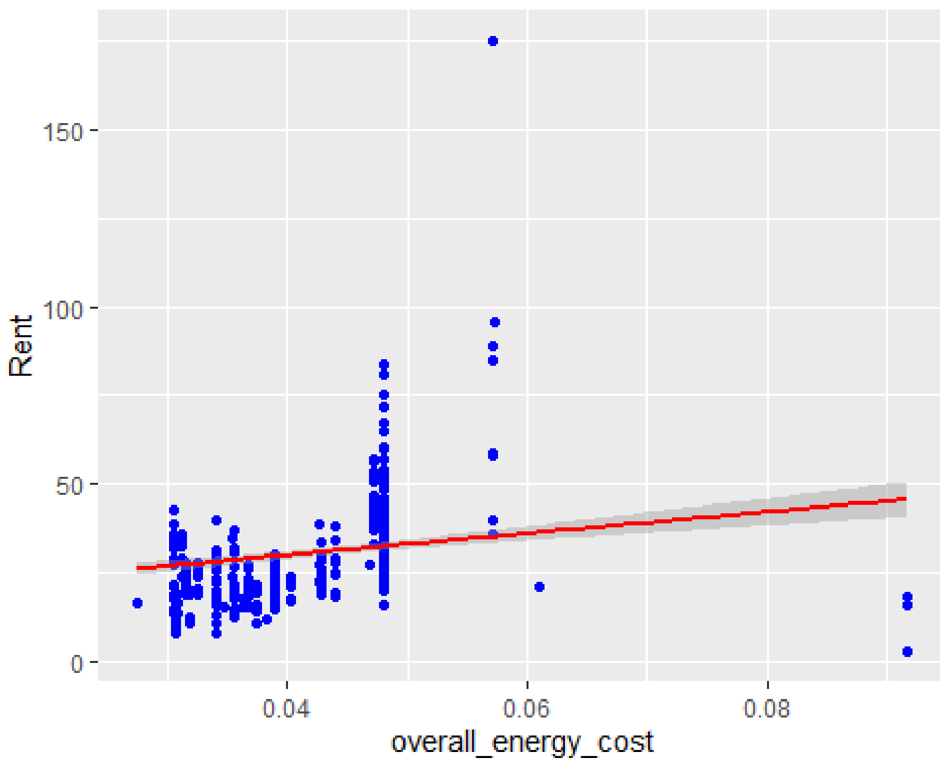
```r
greenbuildings$renovated <- as.factor(greenbuildings$renovated)
greenbuildings$Energystar <- as.factor(greenbuildings$Energystar)
greenbuildings$LEED <- as.factor(greenbuildings$LEED)
greenbuildings <- subset(greenbuildings,greenbuildings$leasing_rate > 0.1)
greenbuildings <- na.omit(greenbuildings)
subset_greenbuildings <- subset(greenbuildings, greenbuildings$size > 250000
& greenbuildings$size < 350000)
green_subset_medians <- ddply(subset_greenbuildings, .(green_rating),
summarise, median = round(median(leasing_rate),4))
ggplot(subset_greenbuildings,aes(cd_total_07, Rent))+ geom_point(colour =
"blue")+
  geom_smooth(method="lm", se=TRUE, fullrange=FALSE, level=0.95, color="red")
```




```r
subset_greenbuildings$overall_energy_cost <- subset_greenbuildings$Gas_Costs
+ subset_greenbuildings$Electricity_Costs
cor(subset_greenbuildings$Rent,subset_greenbuildings$cd_total_07)
```

```
## [1] -0.2579879
```

```r
ggplot(subset_greenbuildings,aes(overall_energy_cost, Rent))+
geom_point(colour = "blue")+
  geom_smooth(method="lm", se=TRUE, fullrange=FALSE, level=0.95, color="red")
```

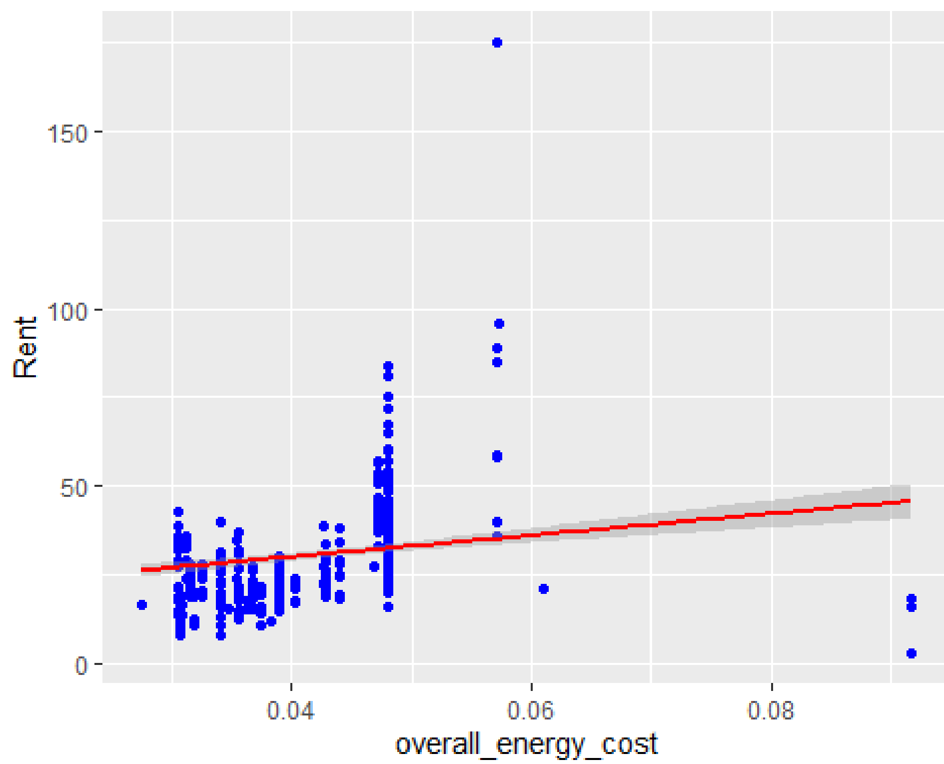



```r
cor(subset_greenbuildings$Rent, subset_greenbuildings$overall_energy_cost)

## [1] 0.2291294
```

This is also illustrated by the scatter plot. The second confounding variable was Gas and Electricity Cost. Although, I combined these two variables into one in my analysis. This variable had a positive correlation with Rent of buildings. This is also illustrated by the scatter plot.

```r
library(mosaic)
library(tidyverse)
library(ggplot2)
library(boot)
library(caret)
library(psych)
require(gridExtra)
library(reshape2)
library(plyr)
greenbuildings <-
read.csv("C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/greenbuildings
.csv")
greenbuildings$green_rating <- as.factor(greenbuildings$green_rating)
greenbuildings$amenities <- as.factor(greenbuildings$amenities)
greenbuildings$renovated <- as.factor(greenbuildings$renovated)
greenbuildings$Energystar <- as.factor(greenbuildings$Energystar)
greenbuildings$LEED <- as.factor(greenbuildings$LEED)
greenbuildings <- subset(greenbuildings,greenbuildings$leasing_rate > 0.1)
```

```r
greenbuildings <- na.omit(greenbuildings)
subset_greenbuildings <- subset(greenbuildings, greenbuildings$size > 250000
& greenbuildings$size < 350000)
subset_greenbuildings$overall_energy_cost <- subset_greenbuildings$Gas_Costs
+ subset_greenbuildings$Electricity_Costs
cor(subset_greenbuildings$Rent,subset_greenbuildings$cd_total_07)

## [1] -0.2579879

ggplot(subset_greenbuildings,aes(overall_energy_cost, Rent))+
geom_point(colour = "blue")+
  geom_smooth(method="lm", se=TRUE, fullrange=FALSE, level=0.95, color="red")
```




```r
cor(subset_greenbuildings$Rent, subset_greenbuildings$overall_energy_cost)

## [1] 0.2291294
```

I used these two confounding variables in my analysis of Rent between non-green and green buildings. To calculate the Rent by square feet for green and non-green while using the two confounding variables, I conducted a linear regression with Rent as the response variable and cd_total_07 and total energy cost as the predictors. Prior to conducting the linear regression model, I created a subset based on whether the buildings were green or not green. The indicator used was green_rating. The two data sets would be used in two linear regression looking at green buildings and another looking at non-green buildings for comparison. After conducting this regression, I input the mean rent for green and non-green into their corresponding linear regression equations. To get a comparison of Rent per square.

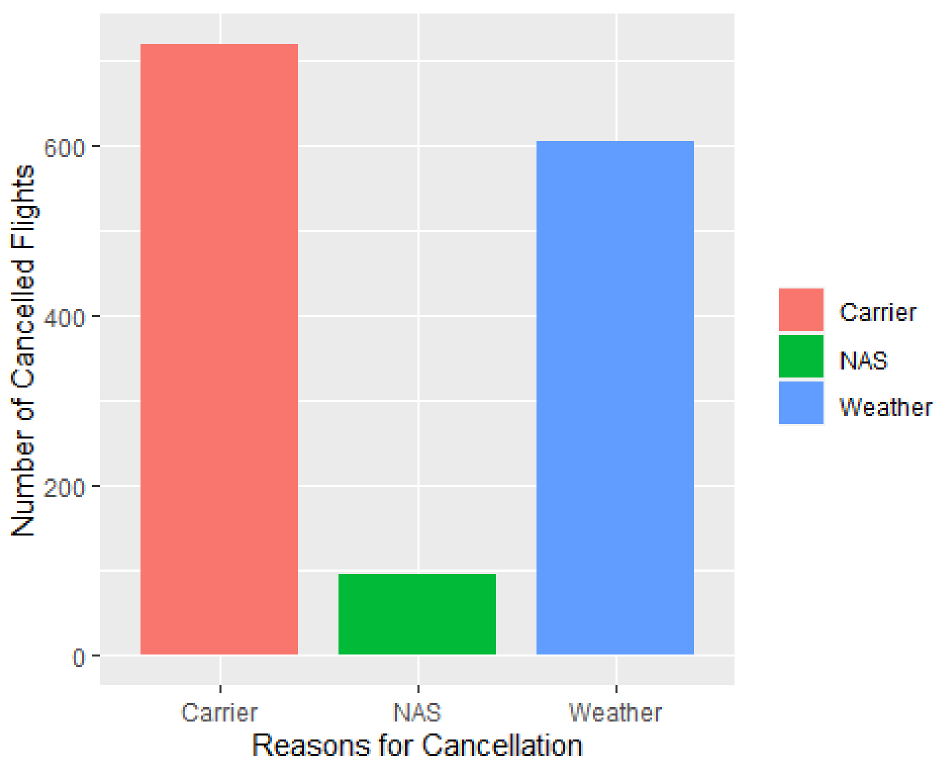
```r
library(mosaic)
library(tidyverse)
library(ggplot2)
library(boot)
library(caret)
library(psych)
require(gridExtra)
library(reshape2)
library(plyr)
greenbuildings <-
read.csv("C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/greenbuildings
.csv")
greenbuildings$green_rating <- as.factor(greenbuildings$green_rating)
greenbuildings$amenities <- as.factor(greenbuildings$amenities)
greenbuildings$renovated <- as.factor(greenbuildings$renovated)
greenbuildings$Energystar <- as.factor(greenbuildings$Energystar)
greenbuildings$LEED <- as.factor(greenbuildings$LEED)
greenbuildings <- subset(greenbuildings,greenbuildings$leasing_rate > 0.1)
greenbuildings <- na.omit(greenbuildings)
subset_greenbuildings <- subset(greenbuildings, greenbuildings$size > 250000
& greenbuildings$size < 350000)
subset_greenbuildings$overall_energy_cost <- subset_greenbuildings$Gas_Costs
+ subset_greenbuildings$Electricity_Costs
subset_greenbuildings_green <- subset(subset_greenbuildings, green_rating ==
1)

regression <- lm(log(subset_greenbuildings_green$Rent) ~ overall_energy_cost
+cd_total_07 , data = subset_greenbuildings_green)
summary(regression)

##
## Call:
## lm(formula = log(subset_greenbuildings_green$Rent) ~ overall_energy_cost +
##     cd_total_07, data = subset_greenbuildings_green)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.97827 -0.22816  0.00924  0.23320  0.88719
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)          2.939e+00  1.995e-01  14.732  < 2e-16 ***
## overall_energy_cost  1.483e+01  4.351e+00   3.410 0.000949 ***
## cd_total_07         -8.772e-05  2.987e-05  -2.936 0.004148 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3721 on 97 degrees of freedom
## Multiple R-squared:  0.1951, Adjusted R-squared:  0.1785
## F-statistic: 11.76 on 2 and 97 DF,  p-value: 2.681e-05
```

```
mean_green_reg <- mean(log(subset_greenbuildings_green$Rent))
result = mean_green_reg *regression$coefficients[2] +
mean_green_reg*regression$coefficients[3] +  regression$coefficients[1]
print(paste("Overall Rent:", as.numeric(result)))

## [1] "Overall Rent: 53.9518358222461"

subset_greenbuildings_nongreen <- subset(subset_greenbuildings, green_rating
== 0)
regression <- lm(log(subset_greenbuildings_nongreen$Rent) ~
overall_energy_cost +cd_total_07+leasing_rate , data =
subset_greenbuildings_nongreen)
summary(regression)

##
## Call:
## lm(formula = log(subset_greenbuildings_nongreen$Rent) ~
overall_energy_cost +
##     cd_total_07 + leasing_rate, data = subset_greenbuildings_nongreen)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.00638 -0.22812  0.03007  0.34330  1.70124
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)          2.932e+00  1.333e-01  22.004  < 2e-16 ***
## overall_energy_cost  9.335e+00  1.924e+00   4.853 1.58e-06 ***
## cd_total_07         -1.937e-04  1.991e-05  -9.728  < 2e-16 ***
## leasing_rate         2.507e-03  1.412e-03   1.775   0.0764 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4959 on 562 degrees of freedom
## Multiple R-squared:  0.1558, Adjusted R-squared:  0.1513
## F-statistic: 34.57 on 3 and 562 DF,  p-value: < 2.2e-16

mean_nongreen_reg <- mean(log(subset_greenbuildings_nongreen$Rent))
result = mean_nongreen_reg *regression$coefficients[2] +
mean_nongreen_reg*regression$coefficients[3] + regression$coefficients[1]
print(paste("Overall Rent:", as.numeric(result)))

## [1] "Overall Rent: 33.5314134003712"
```

As you can see, the rent per square feet for green buildings is much higher than non-green buildings. Thus, the stat's guru's conclusion is correct that green buildings will be more profitable for the real estate developer over time.

Question 2

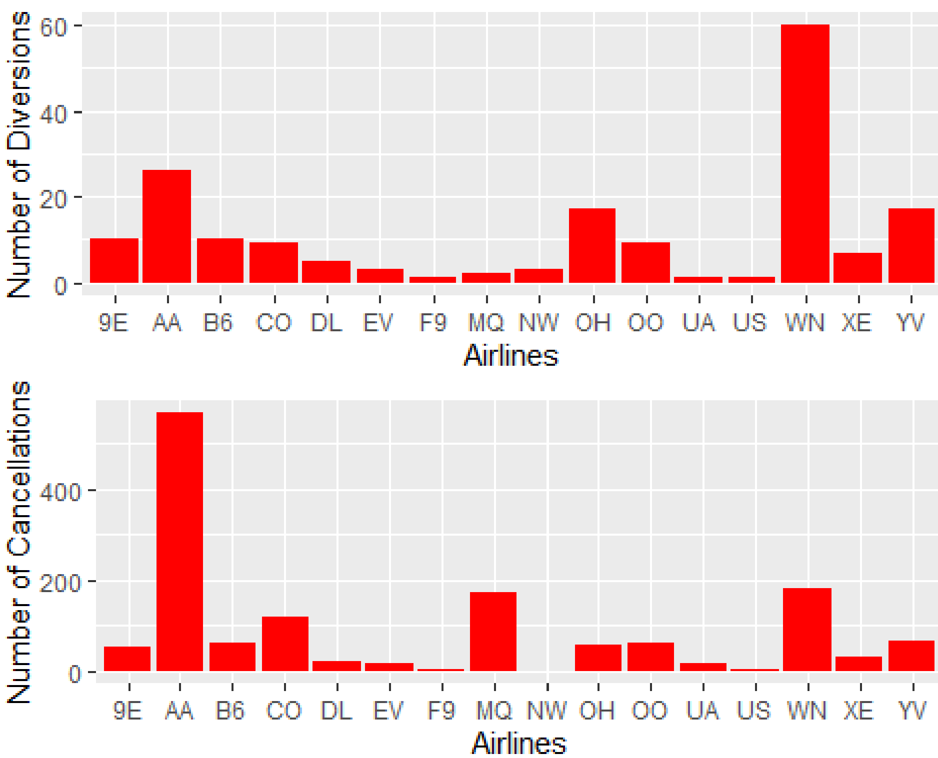
```
library(ggplot2)


ABIA <-
read.csv("C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ABIA.csv")


library(dplyr)
cancelled <- aggregate(x=ABIA$Cancelled,
by=list(TypeCancelled=ABIA$CancellationCode), FUN=sum)
cancelled$TypeCancelled <- ifelse(cancelled$TypeCancelled ==
'A',"Carrier",ifelse(cancelled$TypeCancelled == 'B', 'Weather', 'NAS' ))
cancelled <-cancelled[-1,]
ggplot(cancelled, aes(x = TypeCancelled, y = x, fill = TypeCancelled))+
  geom_bar(stat = "identity", width = 0.8) +
  xlab("Reasons for Cancellation") + ylab("Number of Cancelled Flights") +
  theme(legend.title = element_blank())
```



Distribution of Cancellation Code. It seems that flights get cancelled to and from Austin due to Weather or due to the airline.

```
library(ggplot2)

df_cancelled <- aggregate(x=ABIA$Cancelled,
by=list(Carriers=ABIA$UniqueCarrier), FUN=sum)
```

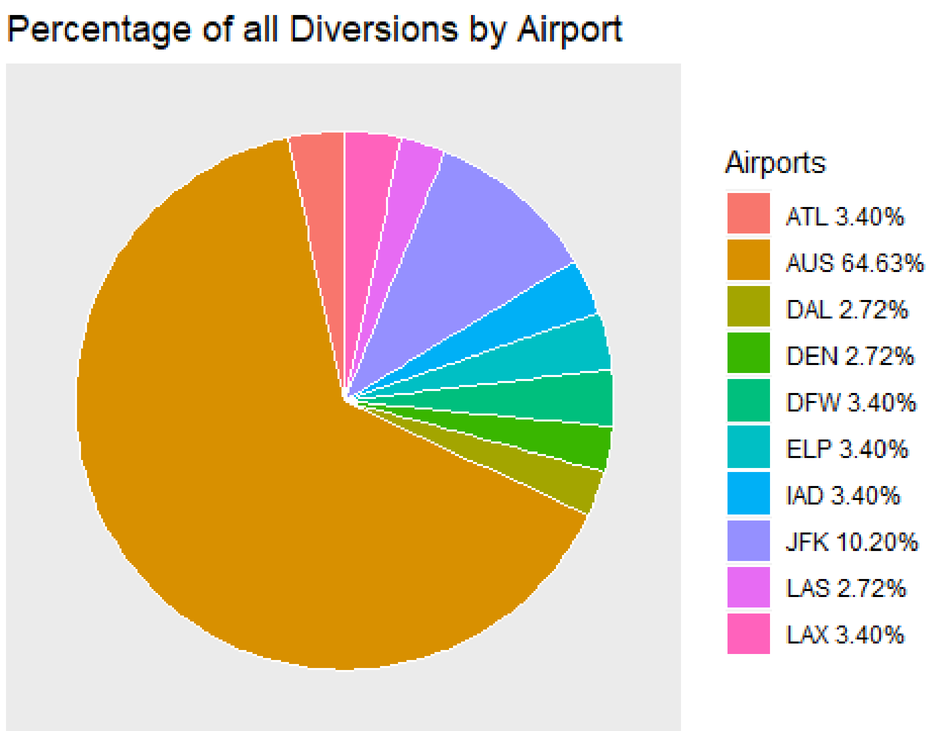
```r
df_diverted <- aggregate(x=ABIA$Diverted,
by=list(Carriers=ABIA$UniqueCarrier), FUN=sum)
colnames(df_cancelled) = c("Carrier", "Cancellations")

colnames(df_diverted) = c("Carrier", "Diversions")


divert= ggplot(df_diverted,aes(x= Carrier,y=Diversions))+ geom_bar(stat =
'identity',fill = 'red') + labs(y = 'Number of Diversions', x =
'Airlines',color =df_diverted$Carrier)
cancel = ggplot(df_cancelled,aes(x= Carrier,y=Cancellations))+ geom_bar(stat
= 'identity',fill = 'red') + labs(y = 'Number of Cancellations', x =
'Airlines', color = df_cancelled$Carrier)
gridExtra::grid.arrange(divert,cancel)
```



Distribution of cancellations and diversions by Airline. WN (Southwest) has highest number of cancellations going to and from Austin Airport. While, AA(American Airlines) has the highest number of diversions.

```r
library(dplyr)
library(ggplot2)

airportcanceled<- aggregate(x=ABIA$Cancelled, by=list(Airport=ABIA$Origin),
FUN=sum)

colnames(airportcanceled) = c("Airport", "Cancellations")
airportcanceled <-airportcanceled[airportcanceled$Cancellations != 0,]
```

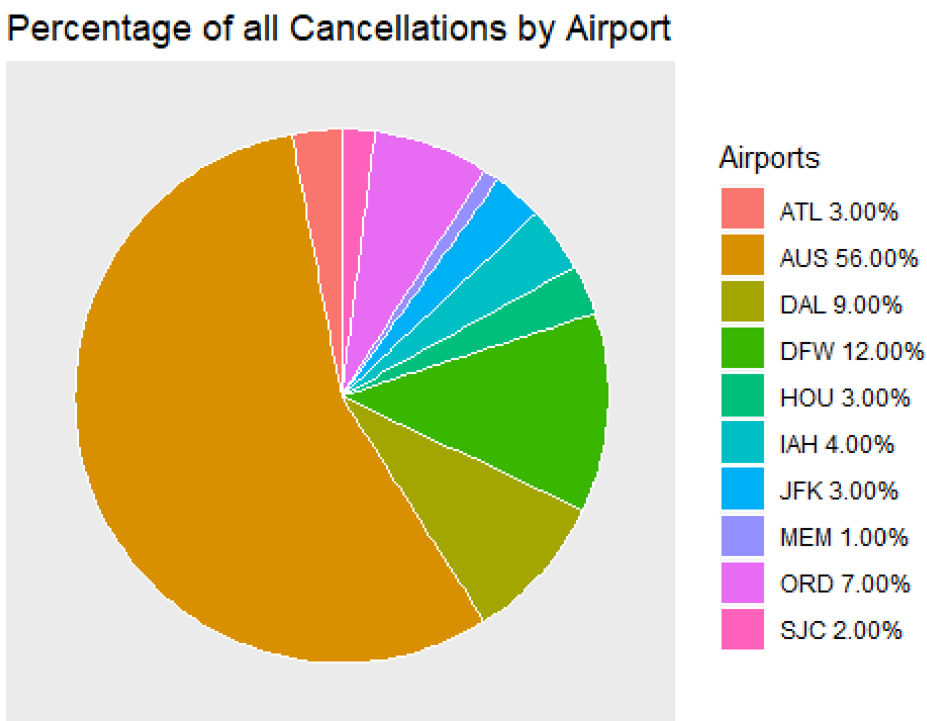
```r
top10 = head(sort(airportcanceled$Cancellations,decreasing=TRUE), n = 10)
top10airports <- airportcanceled[which(airportcanceled$Cancellations %in%
top10),]

top10airports_per = mutate(top10airports,
                cancel_pct = top10airports$Cancellations /
sum(Cancellations))

top10airports_per$cancel_pct <- round(top10airports_per$cancel_pct,2)
percent <- function(x, digits = 2, format = "f", ...) {      # Create user-
defined function
  paste0(formatC(x * 100, format = format, digits = digits, ...), "%")
}
top10airports_per$labels <- percent(top10airports_per$cancel_pct)
top10airports_per$str_pct <-  as.character(top10airports_per$labels)
top10airports_per$Airports <- paste(top10airports_per$Airport,
top10airports_per$str_pct)

airportdiverted<- aggregate(x=ABIA$Diverted, by=list(Airport=ABIA$Origin),
FUN=sum)
colnames(airportdiverted) = c("Airport", "Diversions")
top10divert = head(sort(airportdiverted$Diversions,decreasing=TRUE), n = 10)
top10airports_divert <- airportdiverted[which(airportdiverted$Diversions %in%
top10divert),]

top10airports_divert_per = mutate(top10airports_divert,
                                  divert_pct =
top10airports_divert$Diversions / sum(Diversions))
top10airports_divert_per$labels <-
percent(top10airports_divert_per$divert_pct)
top10airports_divert_per$str_pct <-
as.character(top10airports_divert_per$labels)
top10airports_divert_per$Airports <- paste(top10airports_divert_per$Airport,
top10airports_divert_per$str_pct)


require(ggrepel)

## Loading required package: ggrepel

ggplot(top10airports_divert_per, aes(x ='', y=Diversions, fill=Airports)) +
  geom_bar(stat="identity", width=1, color="white") + coord_polar(theta ="y",
start=0) +
  theme(axis.title.x = element_blank(), axis.title.y =
element_blank(),axis.text = element_blank(),
        axis.ticks = element_blank(),
        panel.grid  = element_blank()) + labs(title = "Percentage of all
Diversions by Airport")
```



The percentage of the top 10 Airports with the most diversions for flights going into Austin.

```
library(dplyr)

airportdiverted<- aggregate(x=ABIA$Diverted, by=list(Airport=ABIA$Origin),
FUN=sum)
colnames(airportdiverted) = c("Airport", "Diversions")
top10divert = head(sort(airportdiverted$Diversions,decreasing=TRUE), n = 10)
top10airports_divert <- airportdiverted[which(airportdiverted$Diversions %in%
top10divert),]

top10airports_divert_per = mutate(top10airports_divert,
                                  divert_pct =
top10airports_divert$Diversions / sum(Diversions))
top10airports_divert_per$labels <-
percent(top10airports_divert_per$divert_pct)
top10airports_divert_per$str_pct <-
as.character(top10airports_divert_per$labels)
top10airports_divert_per$Airports <- paste(top10airports_divert_per$Airport,
top10airports_divert_per$str_pct)
require(ggrepel)

ggplot(top10airports_per, aes(x ='', y=cancel_pct, fill=Airports)) +
  geom_bar(stat="identity", width=1, color="white") + coord_polar(theta ="y",
start=0) +
  theme(axis.title.x = element_blank(), axis.title.y =
element_blank(),axis.text = element_blank(),
```

```
        axis.ticks = element_blank(),
        panel.grid  = element_blank()) +labs(title = "Percentage of all
Cancellations by Airport")
```



The percentage of the top 10 Airports with the most cancellations for flights going into Austin

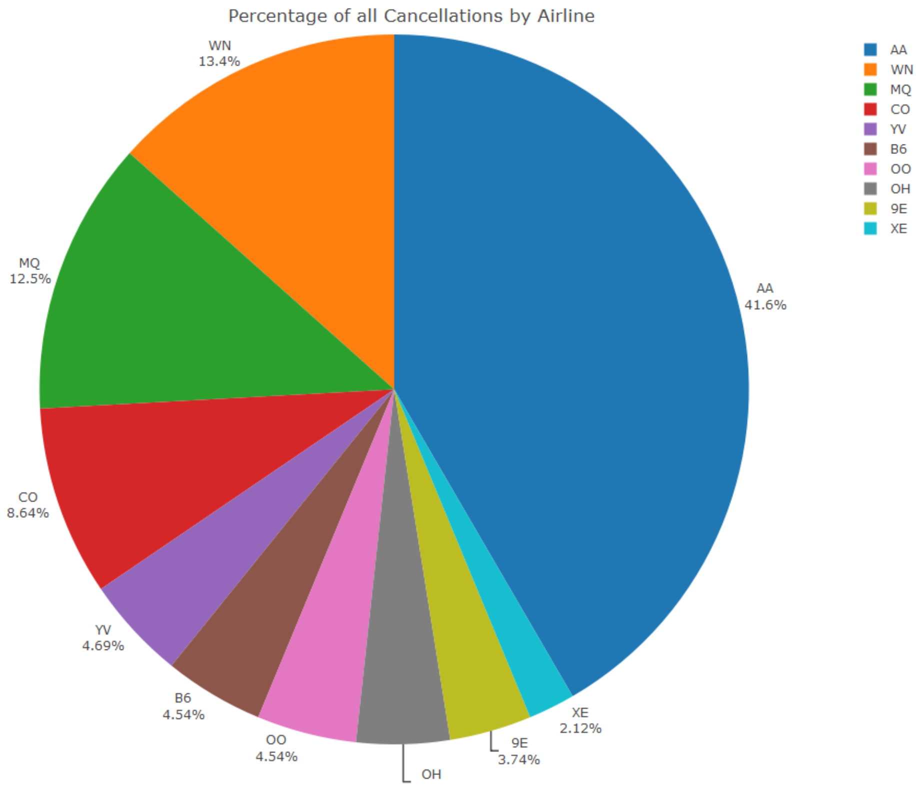
```
library(dplyr)
library(ggplot2)

airlinescanceled2<- aggregate(x=ABIA$Cancelled,
by=list(Airport=ABIA$UniqueCarrier), FUN=sum)
colnames(airlinescanceled2) = c("Airlines", "Cancellations")
airlinesdiverted2 <- aggregate(x=ABIA$Diverted,
by=list(Airport=ABIA$UniqueCarrier), FUN=sum)
colnames(airlinesdiverted2) = c("Airlines", "Diversions")

airlines <- merge(airlinesdiverted2, airlinescanceled2,by=c("Airlines"))
sortedairlines= airlines[with(airlines, order(-Cancellations, Diversions)), ]
top10airlines <-  head(sortedairlines, n=10)

cancelled_month <- aggregate(x=ABIA$Cancelled, by=list(ABIA$Month), FUN=sum)
divert_month <- aggregate(x=ABIA$Diverted, by =list(ABIA$Month), FUN = sum)

colnames(cancelled_month) = c("Months", "Cancellations")
colnames(divert_month) = c("Months", "Diversions")

cancelled_month<- cancelled_month %>% mutate(MonthName = month.name[Months])
```

```r
divert_month<- divert_month %>% mutate(MonthName = month.name[Months])


cancelled_month1<-ggplot(cancelled_month, aes(x =
factor(MonthName,levels=month.name), y = Cancellations))+
  geom_bar(stat = "identity", width = 0.8, fill = 'red') +
  xlab("Months") + ylab("# of Cancellations") + theme(axis.text.x =
element_text(size=10, angle=45))

divert_month1<- ggplot(divert_month, aes(x =
factor(MonthName,levels=month.name), y = Diversions))+
  geom_bar(stat = "identity", width = 0.8,fill = 'yellow') +
  xlab("Months") + ylab("# of Diversions") + theme(axis.text.x =
element_text(size=10, angle=45))
gridExtra::grid.arrange(cancelled_month1,divert_month1)
```

Number of cancellations and diversions by month in 2008. February had the highest number of cancellations while July had the highest number diversions.

```r
airlinescanceled3<- aggregate(x=ABIA$Cancelled,
by=list(Airport=ABIA$UniqueCarrier), FUN=sum)
airlinesdiverted3<- aggregate(x=ABIA$Diverted,
by=list(Airport=ABIA$UniqueCarrier), FUN=sum)
colnames(airlinescanceled3) = c("Airlines", "Cancellations")
colnames(airlinesdiverted3) = c("Airlines", "Diversions")

top10canceled = head(sort(airlinescanceled3$Cancellations,decreasing=TRUE), n
```

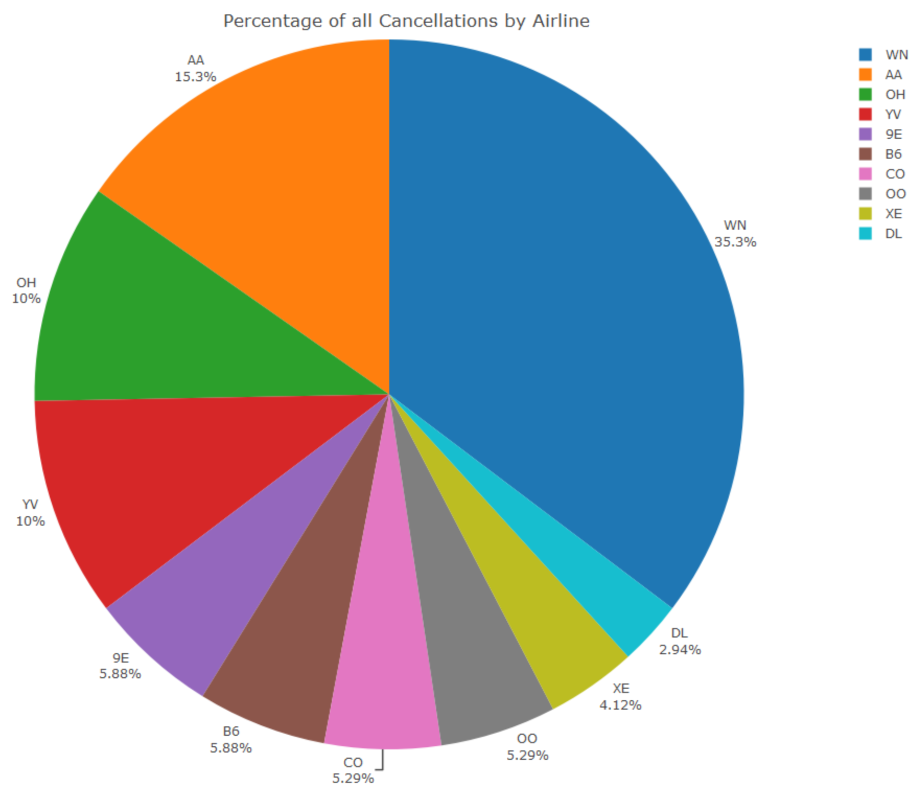
```
= 10)
top10airline_canceled <-
airlinescanceled3[which(airlinescanceled3$Cancellations %in% top10canceled),]

top10diverted = head(sort(airlinesdiverted3$Diversions,decreasing=TRUE), n =
10)
top10airline_diverted <- airlinesdiverted3[which(airlinesdiverted3$Diversions
%in% top10diverted),]

top10airlines_per_cancelled = mutate(top10airline_canceled,
                           cancel_pct = top10airline_canceled$Cancellations /
sum(Cancellations))

top10airlines_per_cancelled$cancel_pct <-
percent(top10airlines_per_cancelled$cancel_pct)

top10airlines_per_cancelled = mutate(top10airline_canceled,
                                    cancel_pct =
top10airline_canceled$Cancellations /
sum(top10airline_canceled$Cancellations))



library(plotly)
library(webshot)
p1<- plot_ly(top10airlines_per_cancelled, labels = ~Airlines, values =
~cancel_pct, type = 'pie',textposition = 'outside',textinfo =
'label+percent') %>%
  layout(title = 'Percentage of all Cancellations by Airline',
         xaxis = list(showgrid = FALSE, zeroline = FALSE, showticklabels =
FALSE),
         yaxis = list(showgrid = FALSE, zeroline = FALSE, showticklabels =
FALSE))

top10airlines_per_diverted = mutate(top10airline_diverted,
                                     diverted_pct =
top10airline_diverted$Diversions / sum(Diversions))
tmpFile <- tempfile(fileext = ".png")
export(p1, file = tmpFile)
```



The percentage of the top 10 Airports with the most cancellations for flights going into Austin.

```
library(dplyr)
library(ggplot2)
airlinescanceled3<- aggregate(x=ABIA$Cancelled,
by=list(Airport=ABIA$UniqueCarrier), FUN=sum)
airlinesdiverted3<- aggregate(x=ABIA$Diverted,
by=list(Airport=ABIA$UniqueCarrier), FUN=sum)
colnames(airlinescanceled3) = c("Airlines", "Cancellations")
colnames(airlinesdiverted3) = c("Airlines", "Diversions")

top10canceled = head(sort(airlinescanceled3$Cancellations,decreasing=TRUE), n
= 10)
top10airline_canceled <-
airlinescanceled3[which(airlinescanceled3$Cancellations %in% top10canceled),]

top10diverted = head(sort(airlinesdiverted3$Diversions,decreasing=TRUE), n =
10)
top10airline_diverted <- airlinesdiverted3[which(airlinesdiverted3$Diversions
%in% top10diverted),]

top10airlines_per_cancelled = mutate(top10airline_canceled,
                              cancel_pct = top10airline_canceled$Cancellations /
sum(Cancellations))
```

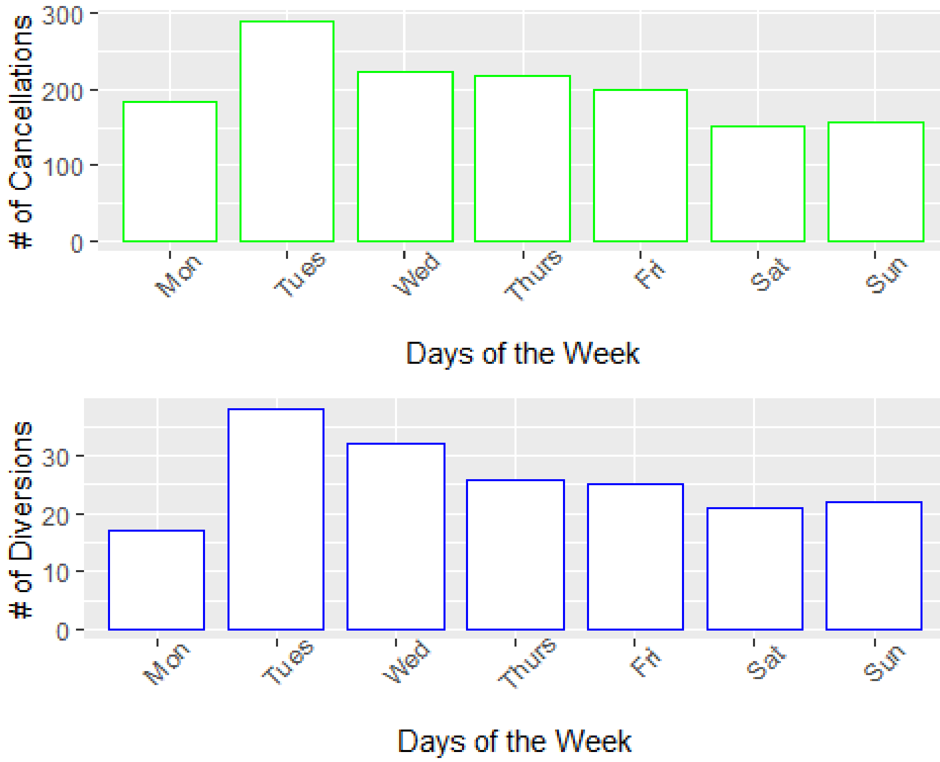
```r
top10airlines_per_cancelled$cancel_pct <-
percent(top10airlines_per_cancelled$cancel_pct)

top10airlines_per_cancelled = mutate(top10airline_canceled,
                                    cancel_pct =
top10airline_canceled$Cancellations /
sum(top10airline_canceled$Cancellations))
top10airlines_per_diverted = mutate(top10airline_diverted,
                                     diverted_pct =
top10airline_diverted$Diversions / sum(Diversions))

library(plotly)
library(webshot)
p2 <- plot_ly(top10airlines_per_diverted, labels = ~Airlines, values =
~diverted_pct, type = 'pie',textposition = 'outside',textinfo =
'label+percent') %>%
  layout(title = 'Percentage of all Cancellations by Airline',
         xaxis = list(showgrid = FALSE, zeroline = FALSE, showticklabels =
FALSE),
         yaxis = list(showgrid = FALSE, zeroline = FALSE, showticklabels =
FALSE))

tmpFile <- tempfile(fileext = ".png")
export(p2, file = tmpFile)
```



The percentage of the top 10 Airports with the most diversions for flights going into Austin.

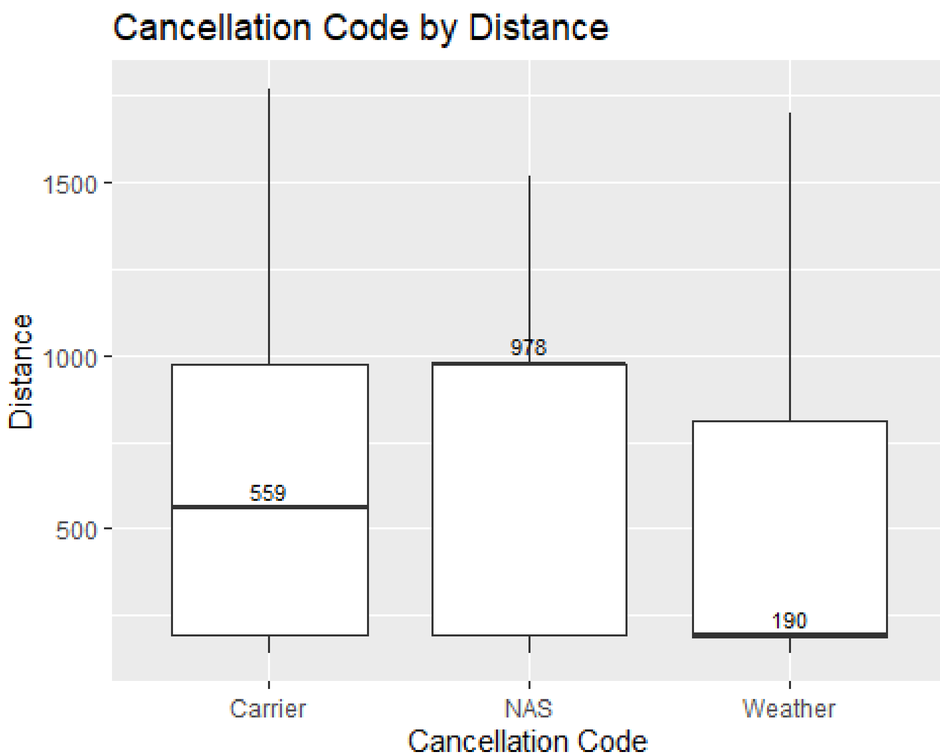
```
library(dplyr)
library(ggplot2)
ABIA$DayOfWeeknames <- ifelse(ABIA$DayOfWeek == 1,
"Mon",ifelse(ABIA$DayOfWeek ==2, "Tues",ifelse(ABIA$DayOfWeek == 3,
"Wed",ifelse(ABIA$DayOfWeek == 4,"Thurs",ifelse(ABIA$DayOfWeek == 5,
"Fri",ifelse(ABIA$DayOfWeek == 6, "Sat", ifelse(ABIA$DayOfWeek == 7,
"Sun",0)))))))
weekdiverted<- aggregate(x=ABIA$Diverted,
by=list(DayofWeek=ABIA$DayOfWeeknames), FUN=sum)
weekcanceled<- aggregate(x=ABIA$Cancelled,
by=list(DayofWeek=ABIA$DayOfWeeknames), FUN=sum)


weekcanceled1<-ggplot(weekcanceled, aes(x = factor(DayofWeek, levels =
c("Mon", "Tues", "Wed", "Thurs", "Fri", "Sat", "Sun")), y = x))+
  geom_bar(stat = "identity", width = 0.8,color = 'green',fill ='white') +
  xlab("Days of the Week") + ylab("# of Cancellations") + theme(axis.text.x =
element_text(size=10, angle=45))

weekdiverted1<- ggplot(weekdiverted, aes(x = factor(DayofWeek, levels =
c("Mon", "Tues", "Wed", "Thurs", "Fri", "Sat", "Sun")), y = x))+
  geom_bar(stat = "identity", width = 0.8, color = 'blue',fill ='white') +
```

```
  xlab("Days of the Week") + ylab("# of Diversions") + theme(axis.text.x =
element_text(size=10, angle=45))
gridExtra::grid.arrange(weekcanceled1,weekdiverted1)
```

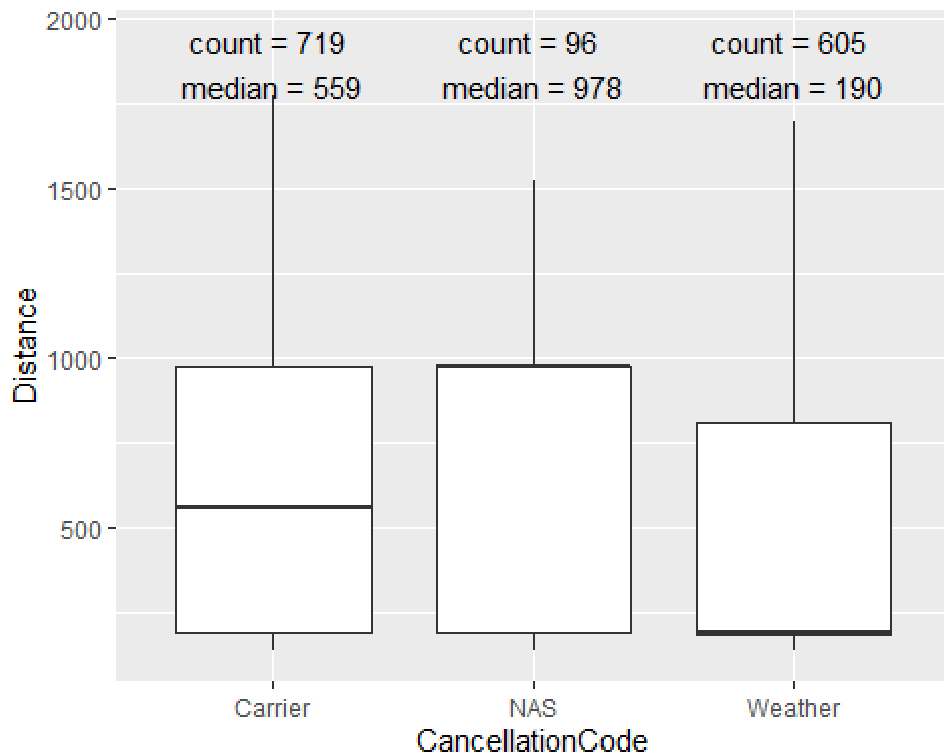


Number of cancellations and diversions by day of the week in 2008. Tuesday had the largest number of cancellations and diversions.

```
library(dplyr)
library(ggplot2)
ABIA$CancellationCode <- ifelse(ABIA$CancellationCode ==
'A',"Carrier",ifelse(ABIA$CancellationCode== 'B',
'Weather',ifelse(ABIA$CancellationCode == 'C','NAS', 'None')))


ABIA_sub <- subset(ABIA, ABIA$CancellationCode != 'None')
options(scipen = 999)
dataMedian <- summarise(group_by(ABIA_sub, CancellationCode), MD =
median(Distance))

ggplot(data=ABIA_sub, aes(x=as.factor(ABIA_sub$CancellationCode),
y=ABIA_sub$Distance)) +
  geom_boxplot() + geom_text(data = dataMedian, aes(x=CancellationCode, y =
MD, label= MD),
                              position = position_dodge(width = 0.8), size =
3, vjust = -0.5) +
  ggtitle("Cancellation Code by Distance")+ xlab("Cancellation Code") +
ylab("Distance")
```



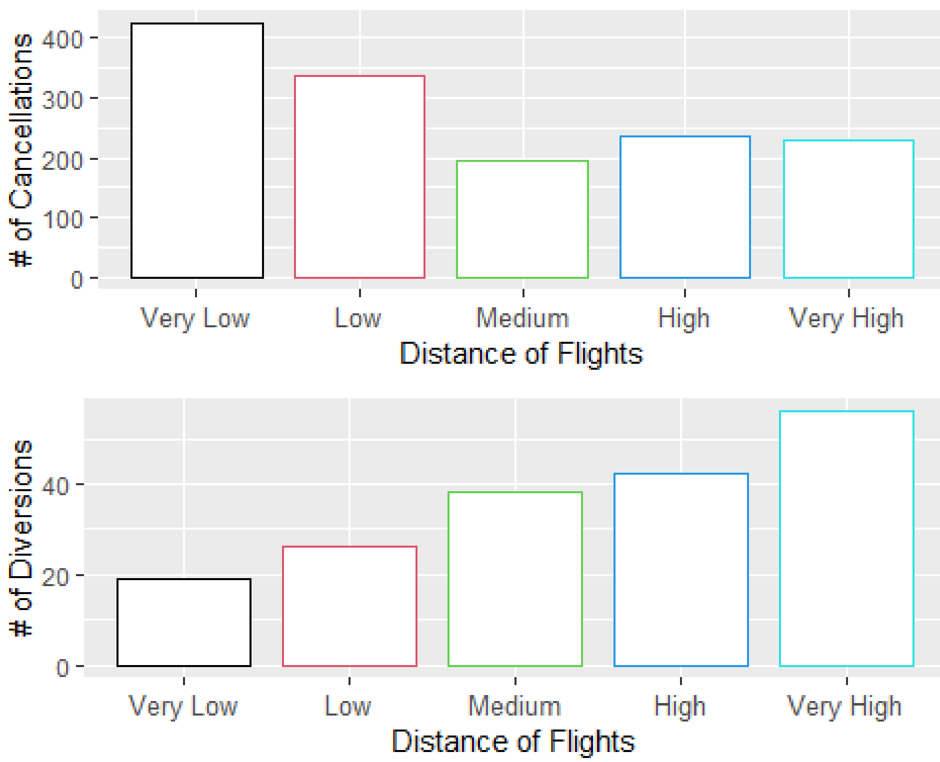

```
stat_box_data <- function(y, upper_limit = max(ABIA_sub$Distance) * 1.15) {
  return(
    data.frame(
      y = 0.95 * upper_limit,
      label = paste('count =', length(y), '\n',
                    'median =', round(median(y), 1), '\n')
    )
  )
}
ggplot(data=ABIA_sub, aes(x=CancellationCode, y=Distance)) +
  geom_boxplot() +
  stat_summary(
    fun.data = stat_box_data,
    geom = "text",
    hjust = 0.5,
    vjust = 0.9
  )
```



Boxplot displaying the number of different types of cancellations by distance travelled.

```
library(gtools)
library(dplyr)
library(ggplot2)
ABIA$DistanceFactor <- quantcut(ABIA$Distance, q=5, na.rm = TRUE,c('Very 
Low', 'Low','Medium','High', 'Very High'))

Distancecanceled<- aggregate(x=ABIA$Cancelled, 
by=list(DistanceQuantile=ABIA$DistanceFactor), FUN=sum)
Distancediverted<- aggregate(x=ABIA$Diverted, 
by=list(DistanceQuantile=ABIA$DistanceFactor), FUN=sum)

colnames(Distancecanceled) = c("Distance Quantiles", "Cancellations")
colnames(Distancediverted) = c("Distance Quantiles", "Diversions")

Distancecanceled1<-ggplot(Distancecanceled, aes(x = 
Distancecanceled$`Distance Quantiles`, y = Cancellations))+
  geom_bar(stat = "identity", width = 0.8, color = Distancecanceled$`Distance 
Quantiles`, fill = 'white' ) +
  xlab("Distance of Flights") + ylab("# of Cancellations") + 
theme(axis.text.x = element_text(size=10))

Distancediverted1<- ggplot(Distancediverted, aes(x = 
Distancediverted$`Distance Quantiles`, y = Diversions))+
  geom_bar(stat = "identity", width = 0.8, color = Distancecanceled$`Distance 
Quantiles`, fill = 'white') +
```

```
  xlab("Distance of Flights") + ylab("# of Diversions") + theme(axis.text.x =
element_text(size=10))
gridExtra::grid.arrange(Distancecanceled1,Distancediverted1)
```



Distribution of the number of cancellations and diversion by distance. It seems flights at very low distances have more cancellations while flights with very high distances have more diversions.

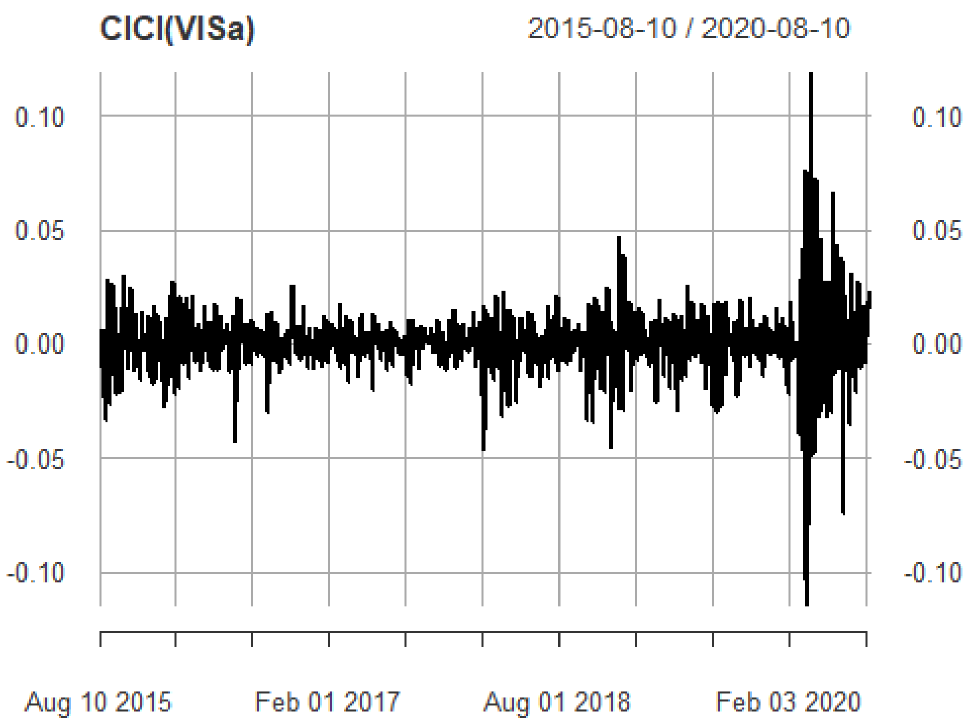
Question 3

In my first portfolio, the underlying goal was to pick ETFs that were from multiple different industries and markets. This is what I would like to consider my diverse ETF portfolio. This portfolio had the following five ETFs: VIS(Vanguard Industrial ETF), XLV (Health Care Select Sector SPDR Fund), XNTK(SPDR Nyse Technology ETF), and VWO(Vanguard Emerging Markets Stock Index). From the plots of the close to close changes there is a lot of volatility after February 2020 which makes sense because this was when there was more volatility in the market due to COVID-19. When looking at the correlation between ETFs, it does look like there is a strong correlation between ETFs. All the correlations seem to also be positive. While, the correlation seems to be much more positive for XNTK and all other ETFs. This is quite interesting and unexpected as I would've thought the correlation would be only slight positive as these ETFs are focused on diverse markets and industries. In terms of the entire portfolios correlation between today's returns and tomorrow's returns, there was no correlation. There also only seem to be a slight autocorrelation between lags. Possibly, a slight correlation between lags. After going through a simulation of 25 days, it seems that I could make a profit of $1200.67. The 5% VAR for this portfolio after running my simulation was 9083.38. Thus, my worst 5% of outcomes have me losing $9083.38 during 4 week period. In my second portfolio, the underlying goal was to pick ETFs that are considered safe. My criteria is that the volatility of the ETF has to be considered minimal or low. This is to minimize my loses. The ETFs that were chosen are the following: SHY (iShares 1-3 Year Treasury Bond), SPLV(Invesco S&P 500 Low Volatility ETF), USMV(iShares Edge MSCI Min Vol USA ETF), IEF(iShares 7-10 Year Treasury Bond), and EFAV(iShares MSCI EAFE Min Vol Factor). Overall, the ETFs do have relative low volatility based on their plots of close to close changes. Although, there is still the largely volatile period starting at February 2020 which is expected. Unlike the diverse portfolio, the correlations between the ETFs for the safe portfolio are not very correlated overall. Although, there is a positive correlation between SPLV and USMV. Overall, this expected since many of these ETFs are not based on a particular industry. When looking at the correlation between all the returns for the portfolio, there is no correlation. In terms of autocorrelation, there only seems to be a slight correlation between lags. After going through a simulation of 25 days, it seems that I could make a profit of $528.33. The 5% VAR for this portfolio is $3766.34. This makes sense given that I was trying to be safe so my worst 5% of the outcomes would have me losing less than my diverse portfolio.

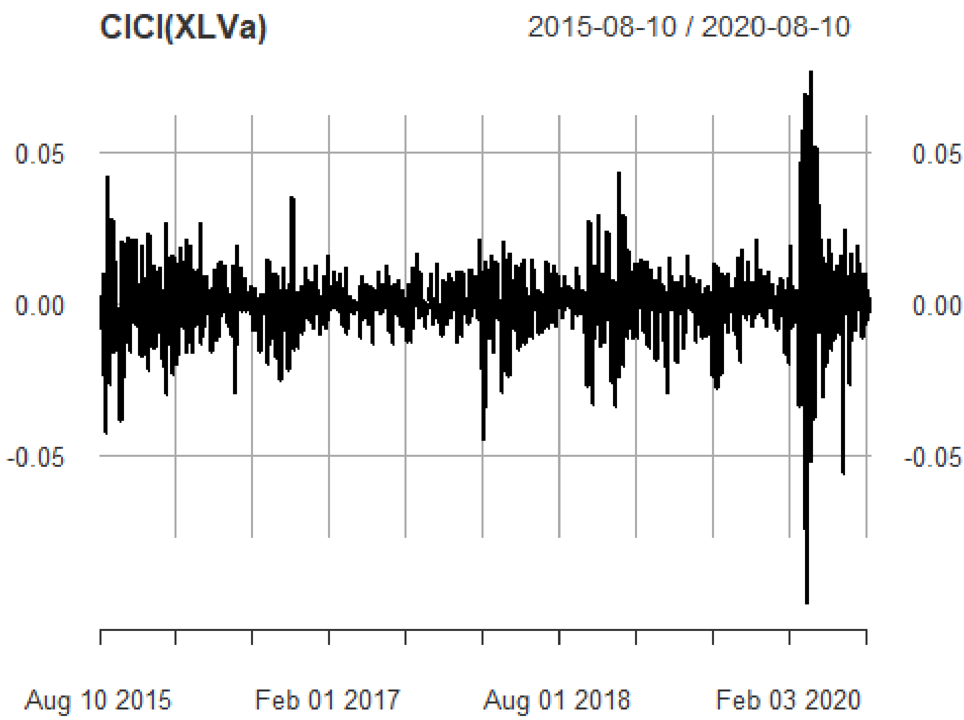
In my third portfolio, the underlying goal was to pick ETFs that are a bit more volatile with the goal of making more money, but with more risk overall. The ETF that were chosen are the following: SOXL(Direxion Daily Semiconductor Bull 3X Shares), TQQQ(ROSHARES TR/ULTRAPRO QQQ), ROM(ProShares Ultra Technology), TECL(Direxion Daily Technology Bull 3X Shares), and VGT(Vanguard Information Technology). Overall, the ETFs do have quite volatilities based on their close to close changes. There is still a relative increase in volatility after February 2020 for reason described previously. The correlations between ETFs is positive, but not as positive as the safe portfolio. There is no correlation in the portfolio's returns when looking at all the ETFs in this portfolio together. There is also no autocorrelation either. After going through a simulation of 25 days, it seems that I could make a profit of $6200.00. Although, my 5% VAR is 20078 which is much higher than my other portfolio which is expected.

```
library(mosaic)
library(quantmod)
library(foreach)
#Diverse------
# Import a few stocks
mystocks = c("VIS", "XLV", "IXC","XNTK","VWO")
getSymbols(mystocks)

## [1] "VIS"  "XLV"  "IXC"  "XNTK" "VWO"

VIS <- VIS["2015-08-10::2020-08-10"]
XLV <- XLV["2015-08-10::2020-08-10"]
IXC <- IXC["2015-08-10::2020-08-10"]
XNTK <- XNTK["2015-08-10::2020-08-10"]
VWO <- VWO["2015-08-10::2020-08-10"]

# Adjust for splits and dividends
```

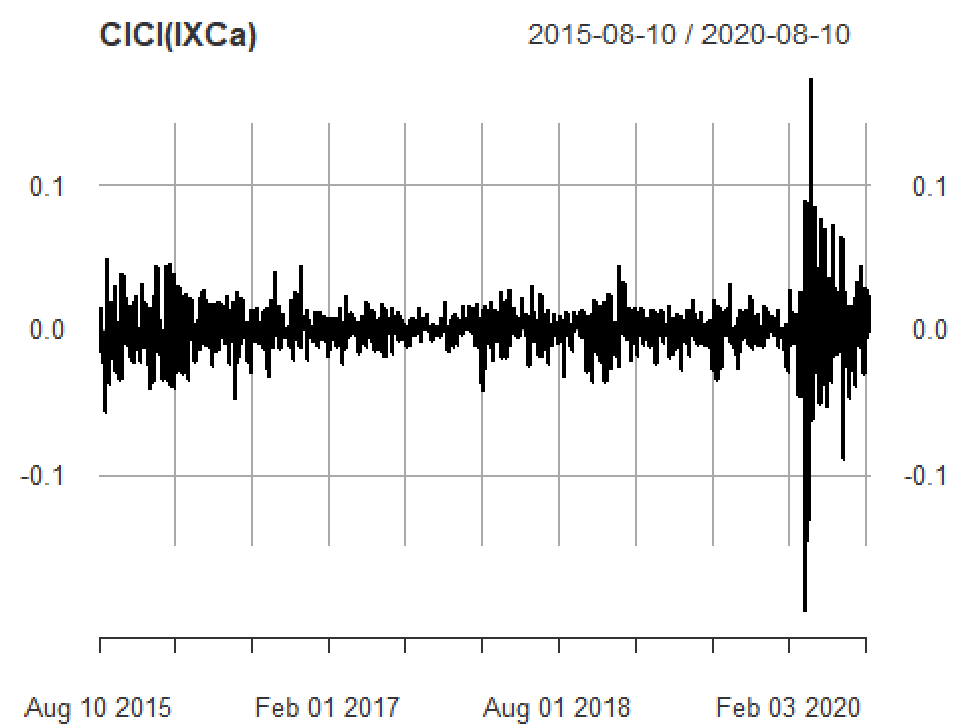
```r
VISa = adjustOHLC(VIS)
XLVa = adjustOHLC(XLV)
IXCa = adjustOHLC(IXC)
XNTKa = adjustOHLC(XNTK)
VWOa = adjustOHLC(VWO)


# Look at close-to-close changes
plot(ClCl(VISa))
```



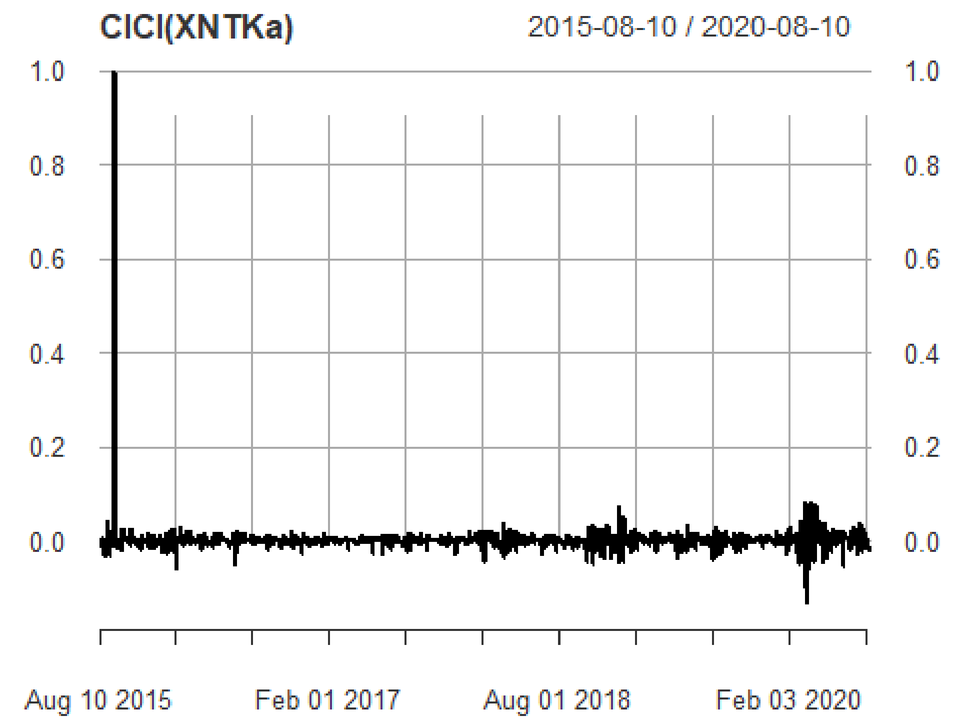

```r
plot(ClCl(XLVa))
```



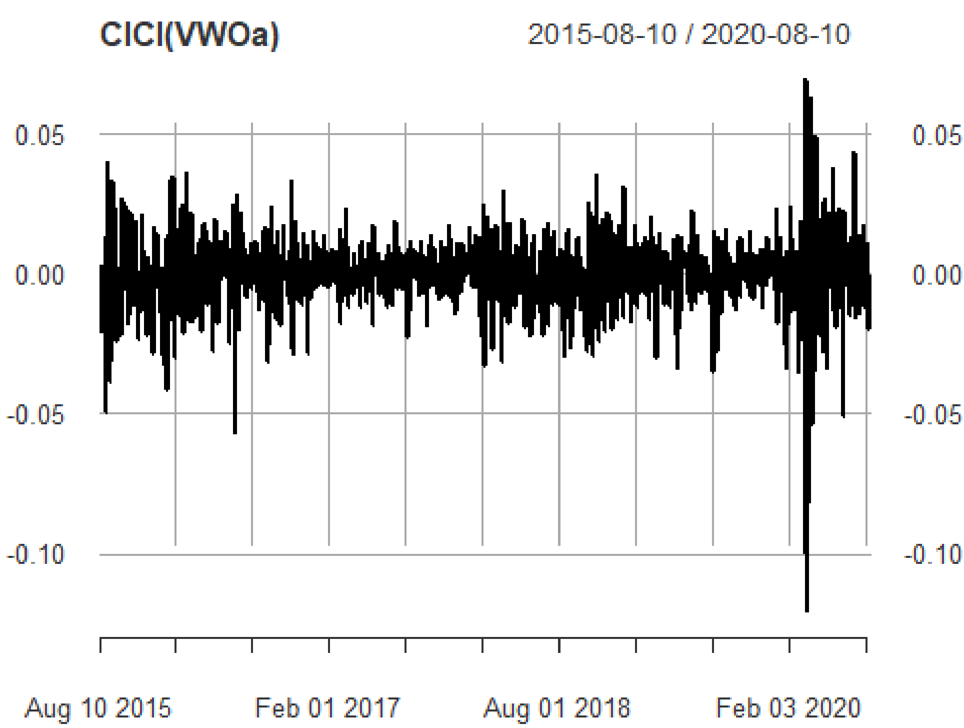

```
plot(ClCl(IXCa))
```




```
plot(ClCl(XNTKa))
```



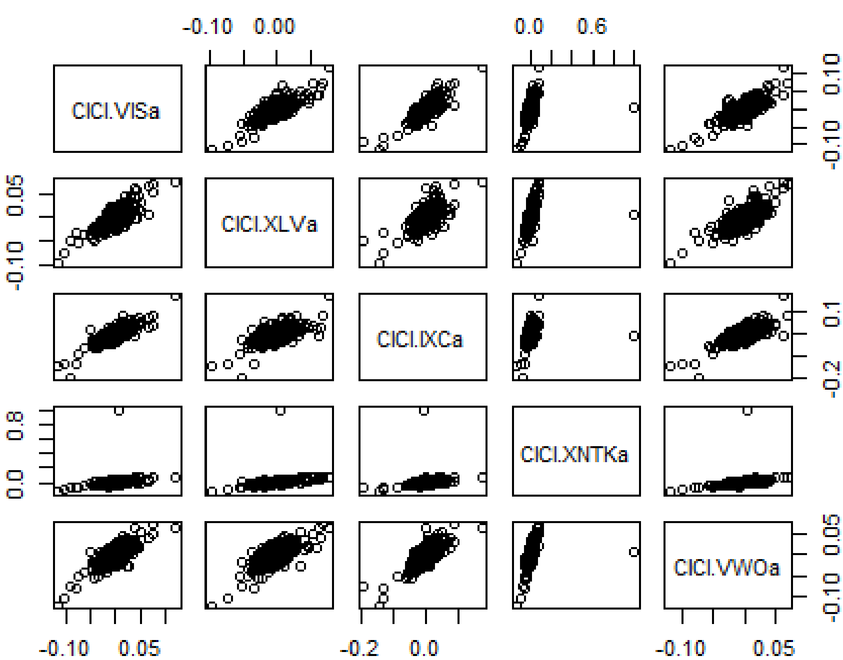

```
plot(ClCl(VWOa))
```



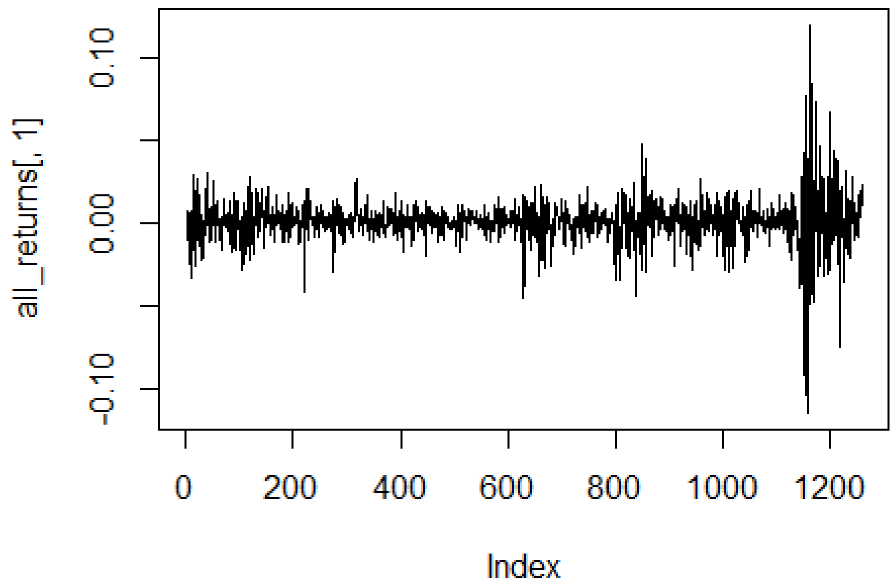
```
# Combine close to close changes in a single matrix
all_returns = cbind(ClCl(VISa),ClCl(XLVa),ClCl(IXCa),ClCl(XNTKa),ClCl(VWOa))
head(all_returns)
```

```
##                ClCl.VISa     ClCl.XLVa    ClCl.IXCa    ClCl.XNTKa
ClCl.VWOa
## 2015-08-10            NA            NA           NA            NA
NA
## 2015-08-11 -0.0106181453 -0.0084288028 -0.003054459 -0.0107836386 -
0.021477187
## 2015-08-12 -0.0006707551  0.0006640324  0.014705883  0.0011276571 -
0.015524678
## 2015-08-13  0.0000000000 -0.0022564111 -0.016606251  0.0002816108 -
0.001903208
## 2015-08-14  0.0065202799  0.0027936545 -0.004912496  0.0045984047
0.002996486
## 2015-08-17  0.0052395541  0.0100822769 -0.001234218  0.0037364968 -
0.010592070
```

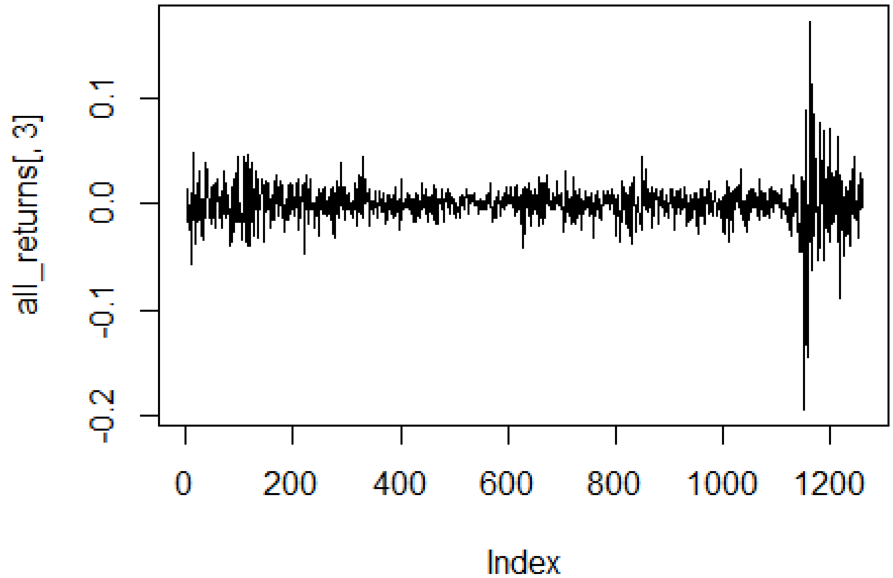
```
# first row is NA because we didn't have a "before" in our data
all_returns = as.matrix(na.omit(all_returns))
N = nrow(all_returns)

# These returns can be viewed as draws from the joint distribution
# strong correlation, but certainly not Gaussian!
pairs(all_returns)
```



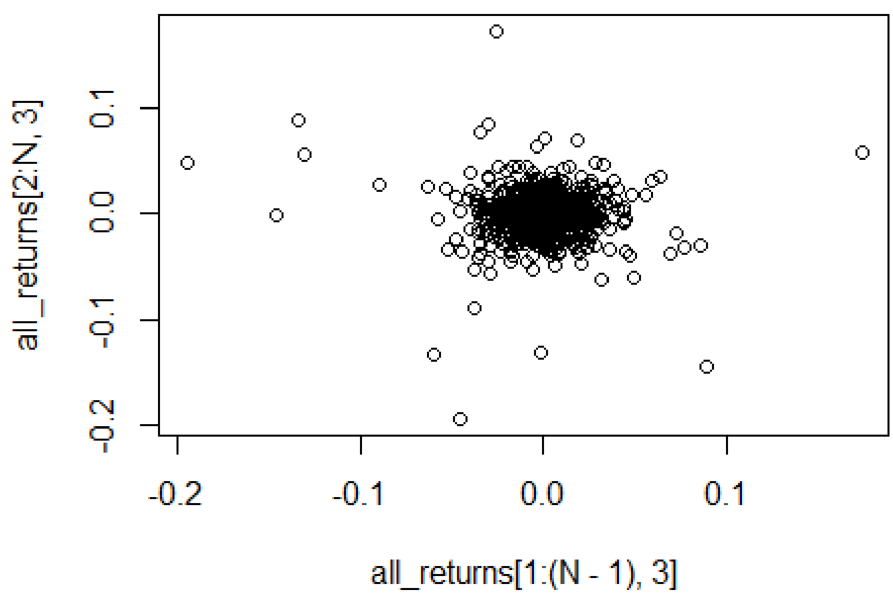
```r
plot(all_returns[,1], type='l')
```



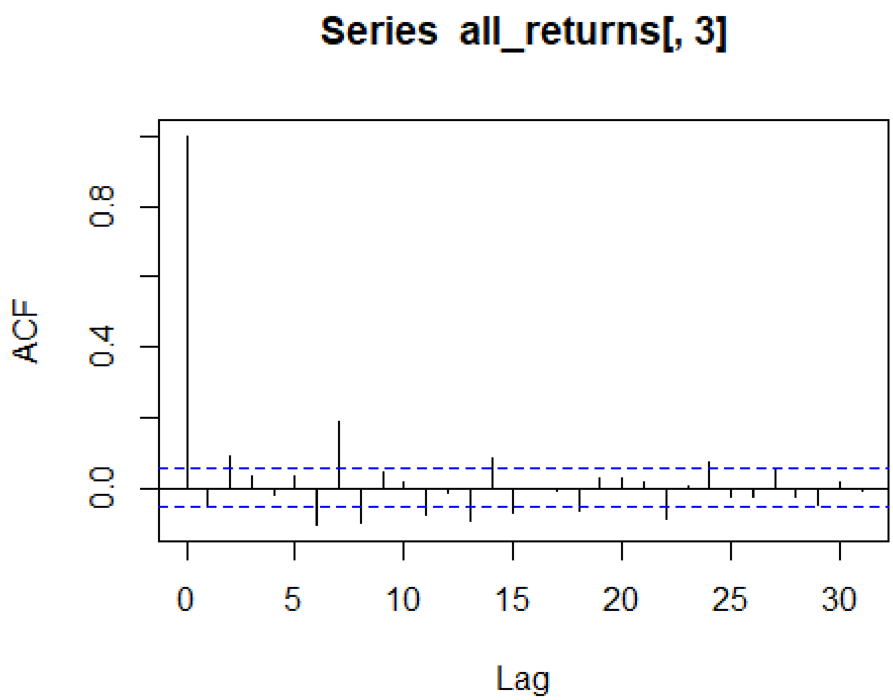

```r
# Look at the market returns over time
plot(all_returns[,3], type='l')
```




```
# are today's returns correlated with tomorrow's?
# not really!
plot(all_returns[1:(N-1),3], all_returns[2:N,3])
```

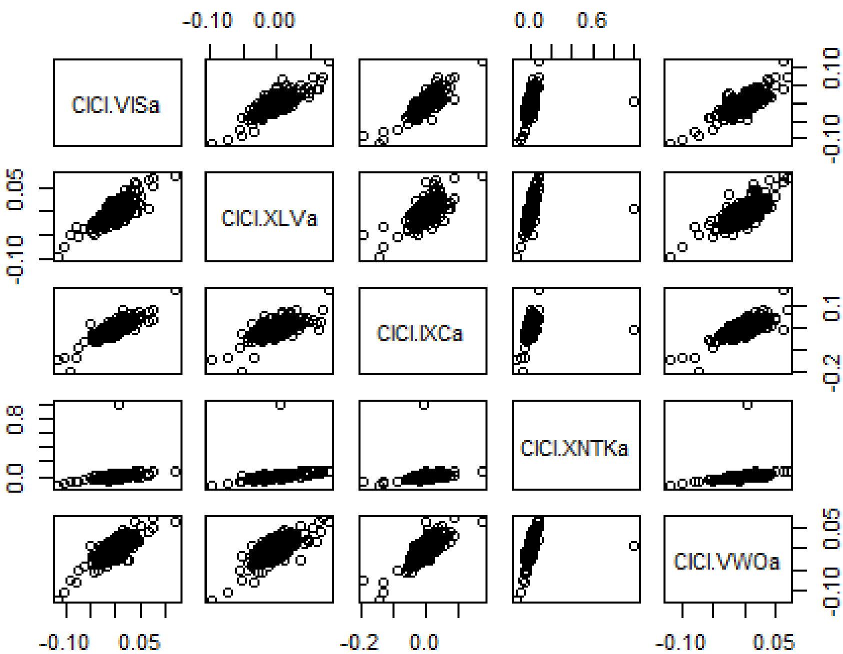


```r
# An autocorrelation plot: nothing there
acf(all_returns[,3])
```



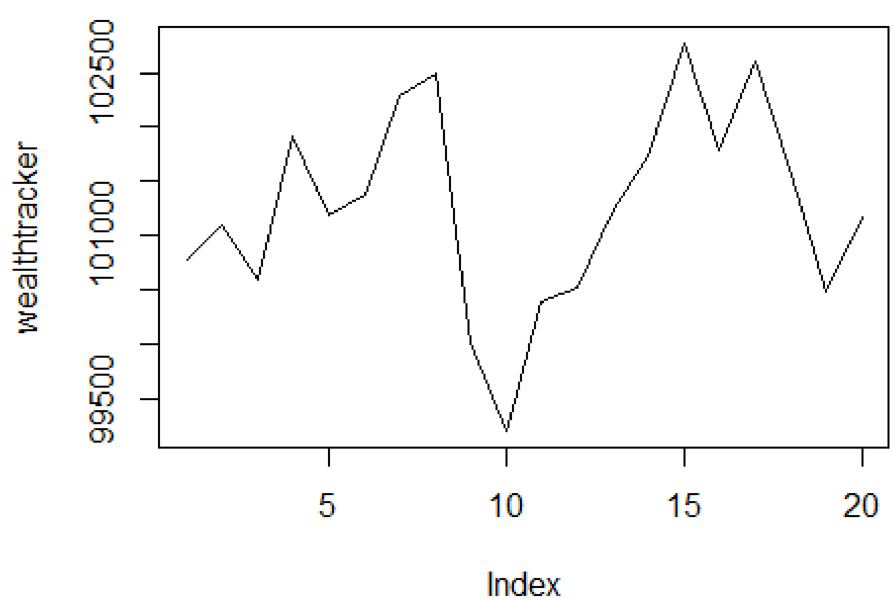

```r
# conclusion: returns uncorrelated from one day to the next
# (makes sense, otherwise it'd be an easy inefficiency to exploit,
# and market inefficiencies that are exploited tend to disappear as a result)

#### Now use a bootstrap approach
#### With more stocks
#mystocks = c("VIS", "PBE", "IXC","XNTK","VWO")

# myprices = getSymbols(mystocks, from = "2015-01-01")


# A chunk of code for adjusting all stocks
# creates a new object adding 'a' to the end
# For example, WMT becomes WMTa, etc
for(ticker in mystocks) {
  expr = paste0(ticker, "a = adjustOHLC(", ticker, ")")
  eval(parse(text=expr))
}

head(XNTKa)

##            XNTK.Open XNTK.High XNTK.Low XNTK.Close XNTK.Volume
XNTK.Adjusted
## 2015-08-10  19.73367  19.84604 19.72998   19.81657       10000
```

```
39.63312
## 2015-08-11  19.73919  19.80367 19.51076   19.60287       23000
39.20573
## 2015-08-12  19.47207  19.62498 19.32101   19.62498        9000
39.24995
## 2015-08-13  19.63787  19.74656 19.59182   19.63050        9600
39.26100
## 2015-08-14  19.59734  19.72077 19.57708   19.72077       17600
39.44154
## 2015-08-17  19.59919  19.82946 19.59919   19.79446       14400
39.58890

# Combine all the returns in a matrix


# Compute the returns from the closing prices
pairs(all_returns)
```




```
# Sample a random return from the empirical joint distribution
# This simulates a random day
return.today = resample(all_returns, 1, orig.ids=FALSE)

# Update the value of your holdings
# Assumes an equal allocation to each asset
total_wealth = 100000
my_weights = c(0.2,0.2,0.2, 0.2, 0.2)
holdings = total_wealth*my_weights
```

```r
holdings = holdings*(1 + return.today)

sum(holdings)

## [1] 100687.5

# Now loop over two trading weeks
# let's run the following block of code 5 or 6 times
# to eyeball the variability in performance trajectories

## begin block
n_days = 20
wealthtracker = rep(0, n_days) # Set up a placeholder to track total wealth
for(today in 1:n_days) {
  return.today = resample(all_returns, 1, orig.ids=FALSE)
  holdings = holdings + holdings*return.today
  total_wealth = sum(holdings)
  wealthtracker[today] = total_wealth
}
total_wealth

## [1] 101166

plot(wealthtracker, type='l')
```



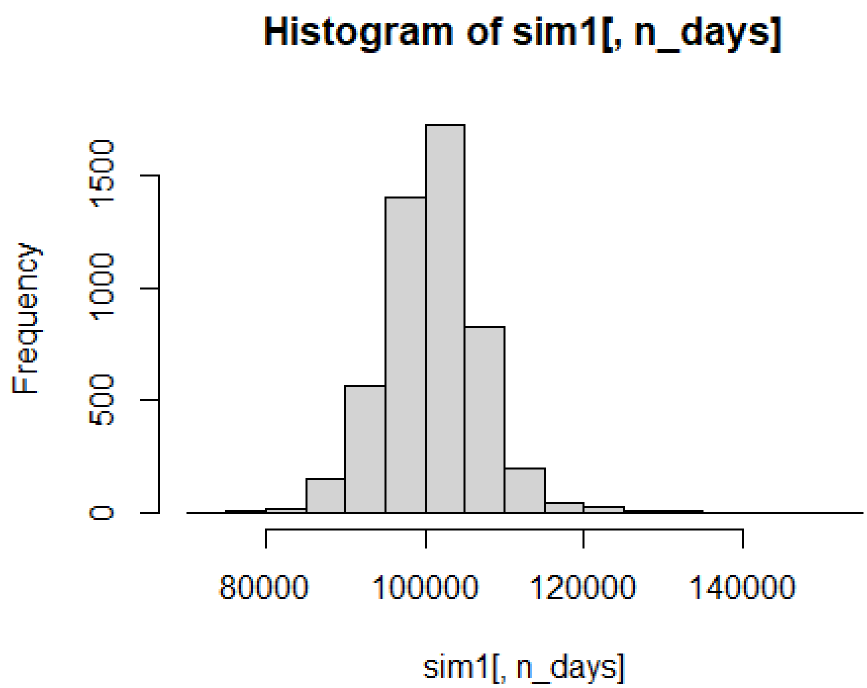

```r
## end block
```

```r
# Now simulate many different possible futures
# just repeating the above block thousands of times
initial_wealth = 100000
sim1 = foreach(i=1:5000, .combine='rbind') %do% {
  total_wealth = initial_wealth
  weights = c(0.2, 0.2, 0.2, 0.2, 0.2)
  holdings = weights * total_wealth
  n_days = 20
  wealthtracker = rep(0, n_days)
  for(today in 1:n_days) {
    return.today = resample(all_returns, 1, orig.ids=FALSE)
    holdings = holdings + holdings*return.today
    total_wealth = sum(holdings)
    wealthtracker[today] = total_wealth
  }
  wealthtracker
}

# each row is a simulated trajectory
# each column is a data
head(sim1)
```

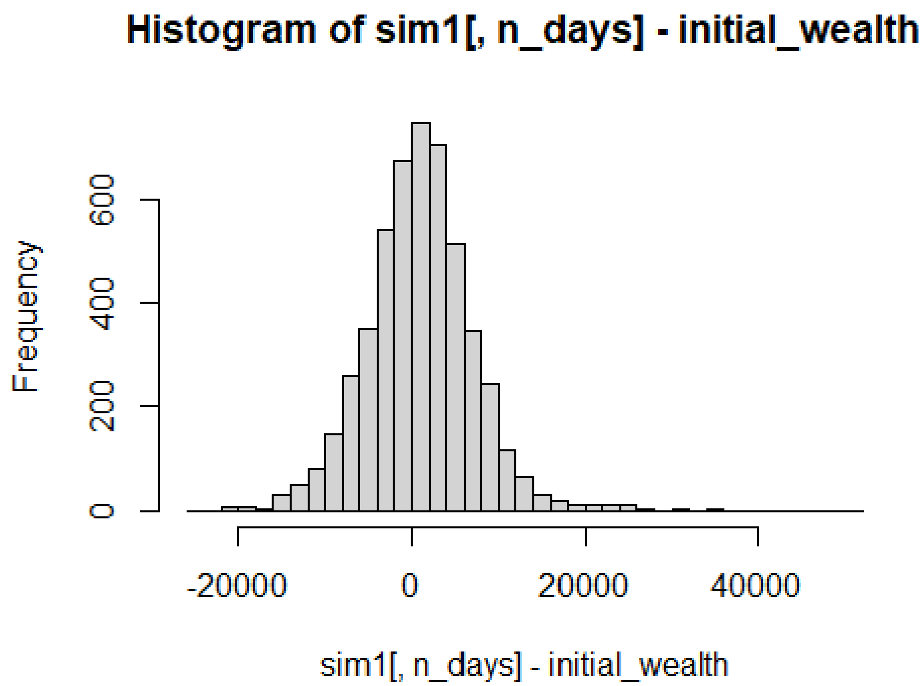
```
##               [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
[,7]
## result.1  99507.33  99536.61  98691.01  98245.15 100398.21  99122.24
99060.38
## result.2  98783.72  98269.19  98333.28  99269.39  98918.60  99309.78
98727.93
## result.3 100467.93 102147.34 102687.63 103305.83 104736.55 103719.02
103222.54
## result.4  99756.90 101421.42 102221.83 100034.76 101073.71 101781.55
101572.75
## result.5  96739.88  98061.26  98449.51  97766.99  98723.05  98917.01
98795.43
## result.6  98936.90  99468.64  99474.59 100050.04  99955.20 100929.14
99705.91
##               [,8]      [,9]     [,10]     [,11]    [,12]     [,13]
[,14]
## result.1  99912.86  98332.94  98884.68  99147.24 100022.4 100436.36
98640.40
## result.2  99352.52  98911.83  99256.21 100106.31 100687.5  99356.20
99490.70
## result.3 103505.35 104404.36 104151.69 103767.12 102370.4 102459.67
102259.75
## result.4 101950.13 100720.99 100621.22 100868.72 102519.7 102598.60
102258.86
## result.5  98655.58  98542.57  98722.72  98825.67 100288.9  99342.82
99302.75
## result.6 100505.69 101174.03 101465.12 101561.49 100854.4 100437.22
101662.50
```

```
##               [,15]     [,16]     [,17]     [,18]     [,19]     [,20]
## result.1  99467.12  99888.93  98322.29  99006.57 100161.09 100172.49
## result.2  99468.74  98016.06  97315.75 100574.24 101373.40 102846.43
## result.3  99773.15  99268.43  99050.00  99366.87  99766.55  99821.82
## result.4 102443.44 102852.94 102590.44 102356.98 103017.11 103474.14
## result.5 100633.56 102778.61 103642.56 104374.37 104215.95 104265.72
## result.6 101506.93 101273.42 102679.21 103334.79 103984.78 104168.84
```

```r
hist(sim1[,n_days], 25)
```

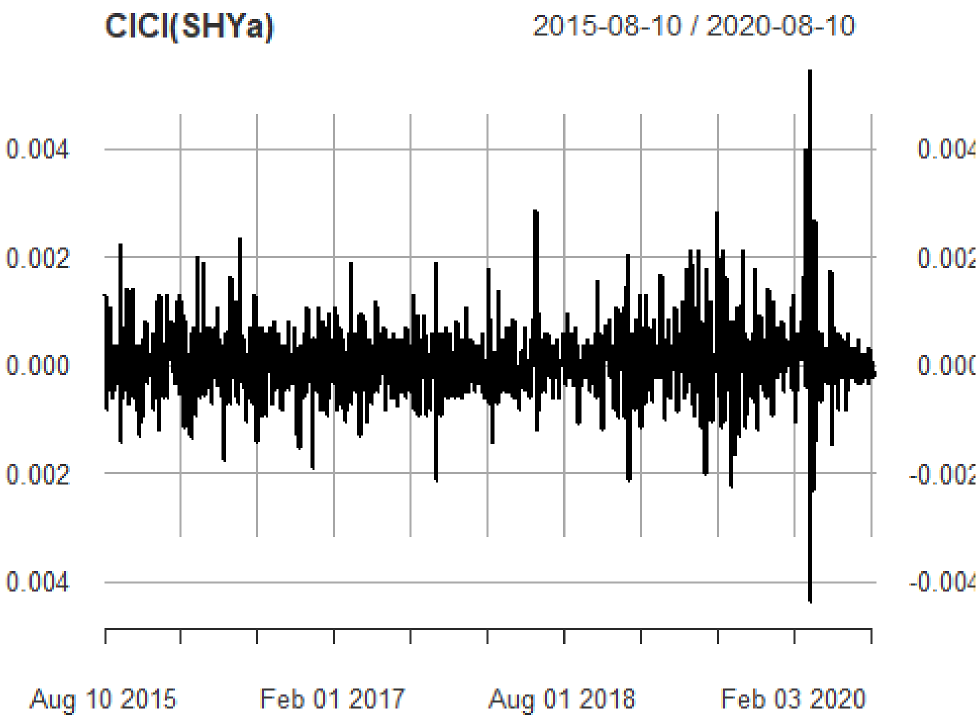



```r
# Profit/loss
mean(sim1[,n_days])

## [1] 100993.7

mean(sim1[,n_days] - initial_wealth)

## [1] 993.6644

hist(sim1[,n_days]- initial_wealth, breaks=30)
```

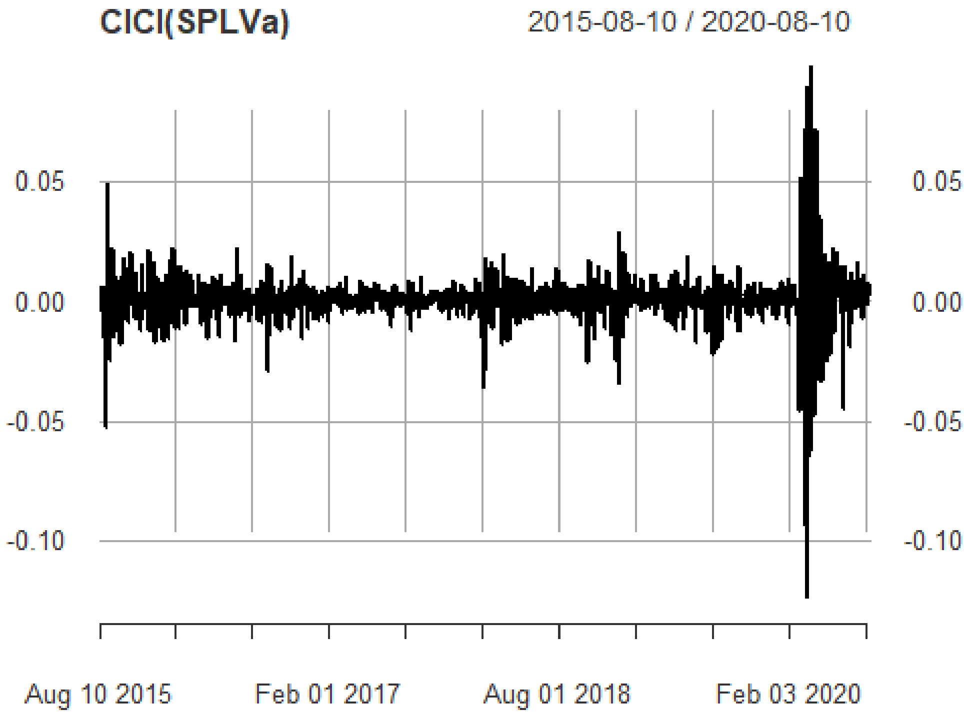



```
# 5% value at risk:
quantile(sim1[,n_days]- initial_wealth, prob=0.05)

##        5%
## -9137.798

# note: this is  a negative number (a loss, e.g. -500), but we conventionally
# express VaR as a positive number (e.g. 500)

#Safe/Defensive Approach
# Import a few stocks
mystocks = c("SHY", "SPLV", "USMV","IEF","EFAV")
getSymbols(mystocks)

## [1] "SHY"  "SPLV" "USMV" "IEF"  "EFAV"

SHY <- SHY["2015-08-10::2020-08-10"]
SPLV <- SPLV["2015-08-10::2020-08-10"]
USMV <- USMV["2015-08-10::2020-08-10"]
IEF <- IEF["2015-08-10::2020-08-10"]
EFAV <- EFAV["2015-08-10::2020-08-10"]

# Adjust for splits and dividends
SHYa = adjustOHLC(SHY)
SPLVa = adjustOHLC(SPLV)
USMVa = adjustOHLC(USMV)
IEFa = adjustOHLC(IEF)
```

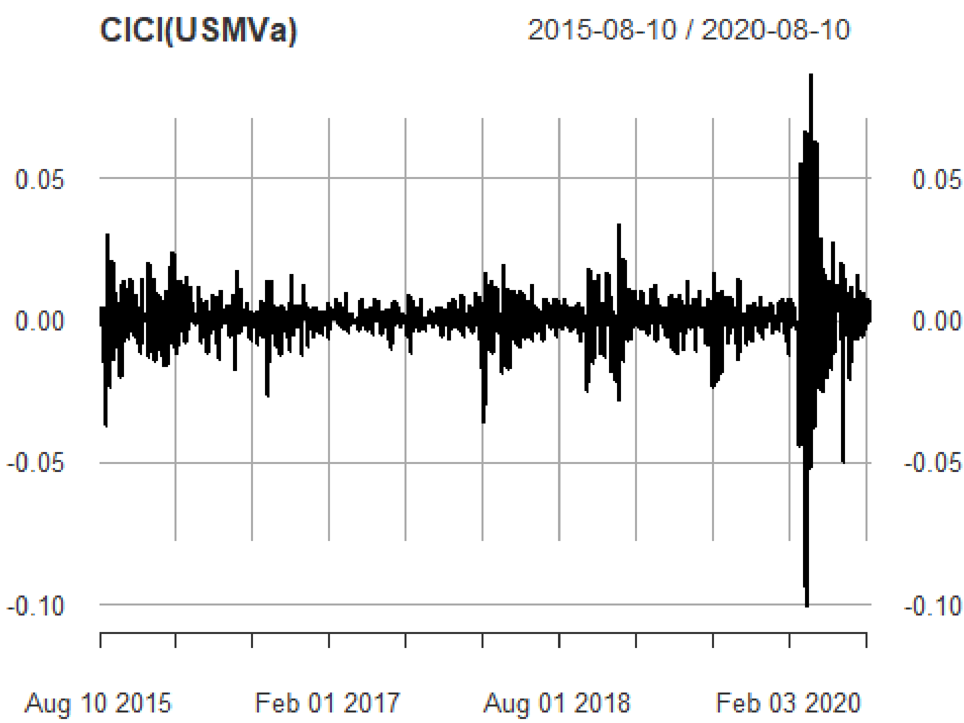
```r
EFAVa = adjustOHLC(EFAV)

# Look at close-to-close changes
plot(ClCl(SHYa))
```



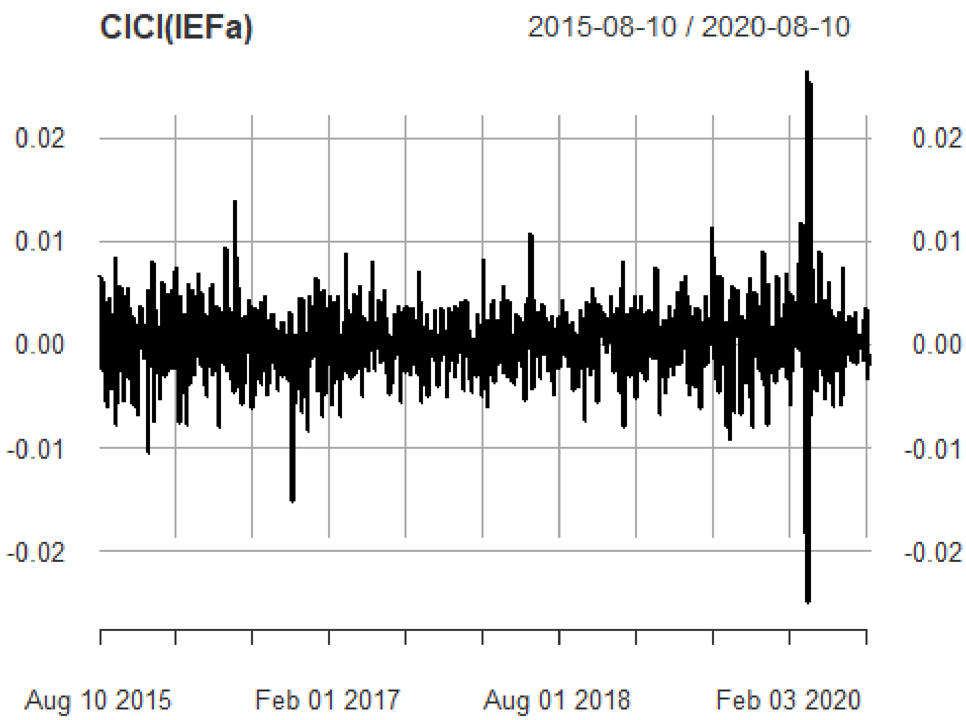

```r
plot(ClCl(SPLVa))
```



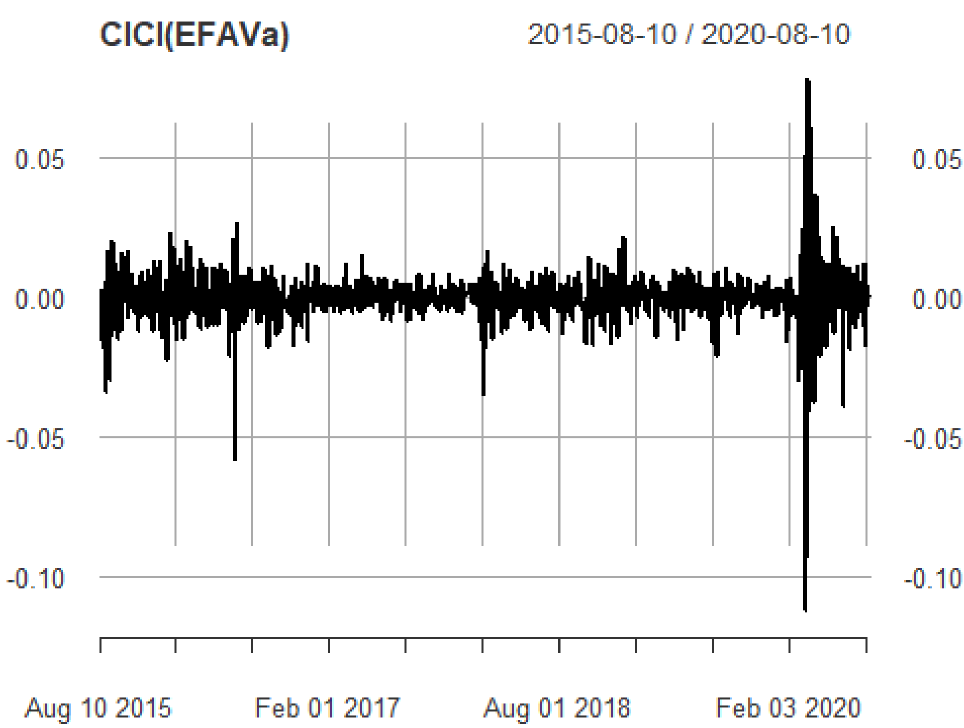

```
plot(ClCl(USMVa))
```



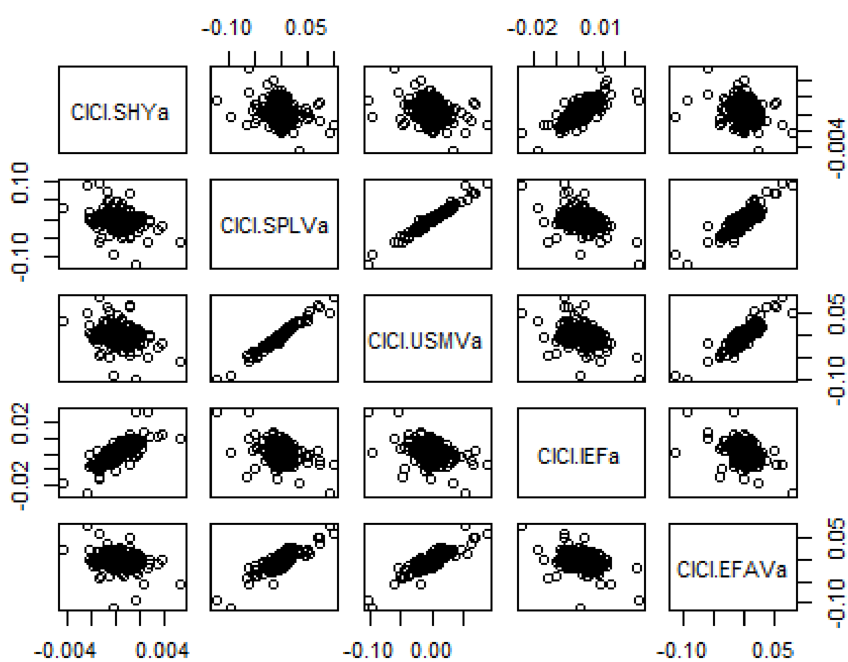

```
plot(ClCl(IEFa))
```



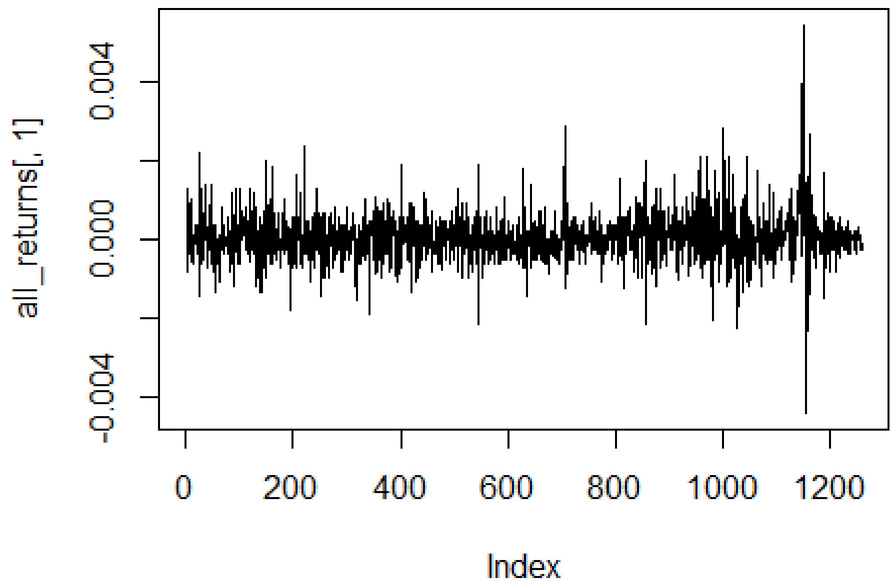

```r
plot(ClCl(EFAVa))
```



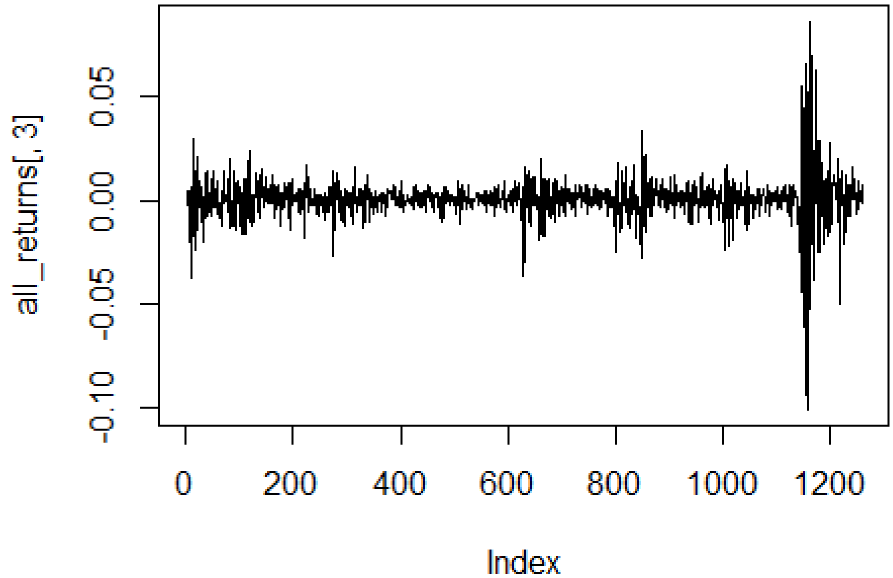
```
# Combine close to close changes in a single matrix
all_returns =
cbind(ClCl(SHYa),ClCl(SPLVa),ClCl(USMVa),ClCl(IEFa),ClCl(EFAVa))
head(all_returns)

##                ClCl.SHYa    ClCl.SPLVa    ClCl.USMVa    ClCl.IEFa
ClCl.EFAVa
## 2015-08-10            NA            NA            NA           NA
NA
## 2015-08-11  0.0012985598 -0.0036374121 -0.0021321962  0.006604435 -
0.0157964305
## 2015-08-12  0.0000000000 -0.0007822165  0.0028490741 -0.000374918 -
0.0017833705
## 2015-08-13 -0.0008252771  0.0018266962  0.0002366951 -0.002625401 -
0.0002976924
## 2015-08-14 -0.0004719882  0.0058199234  0.0047337515 -0.001034136
0.0031273119
## 2015-08-17  0.0005902845  0.0033730409  0.0042402826  0.001788020
0.0001484857

# first row is NA because we didn't have a "before" in our data
all_returns = as.matrix(na.omit(all_returns))
N = nrow(all_returns)

# These returns can be viewed as draws from the joint distribution
# strong correlation, but certainly not Gaussian!
pairs(all_returns)
```



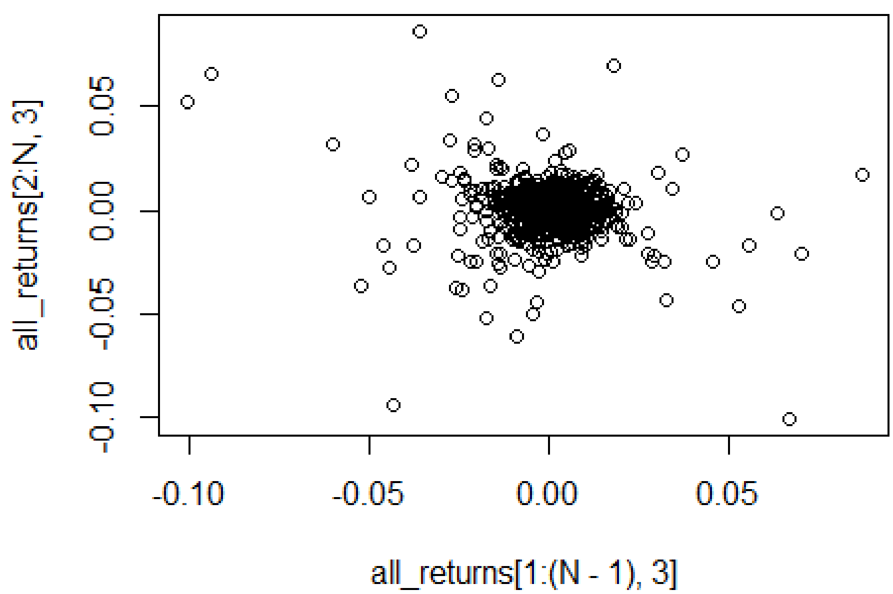
```r
plot(all_returns[,1], type='l')
```



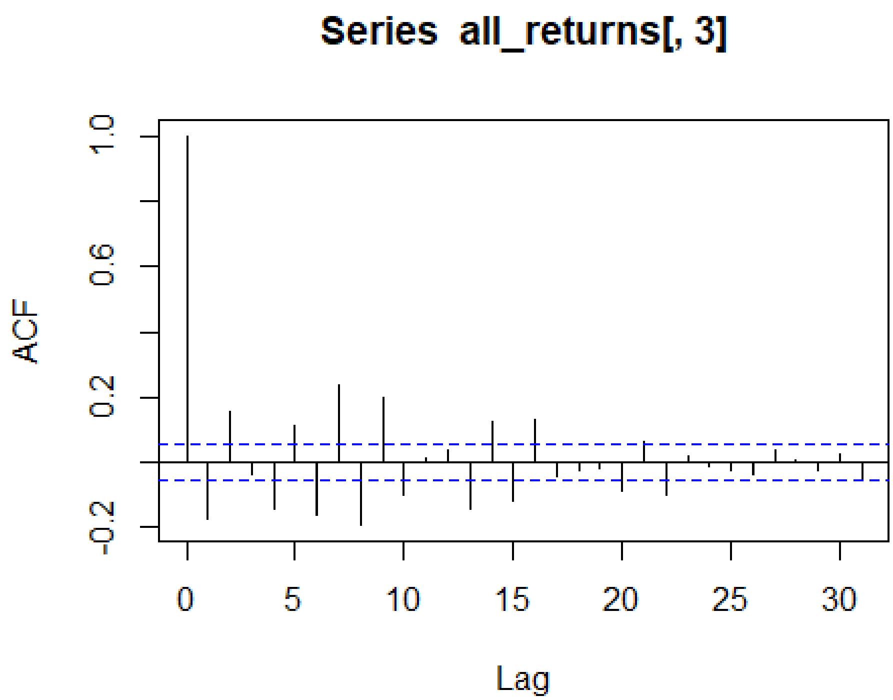

```r
# Look at the market returns over time
plot(all_returns[,3], type='l')
```




```
# are today's returns correlated with tomorrow's?
# not really!
plot(all_returns[1:(N-1),3], all_returns[2:N,3])
```

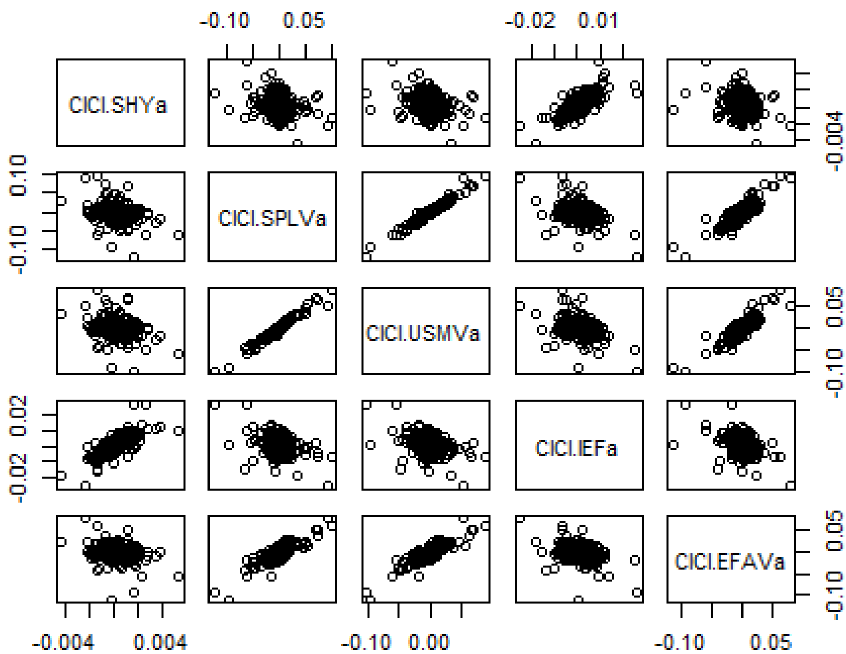


```
# An autocorrelation plot: nothing there
acf(all_returns[,3])
```



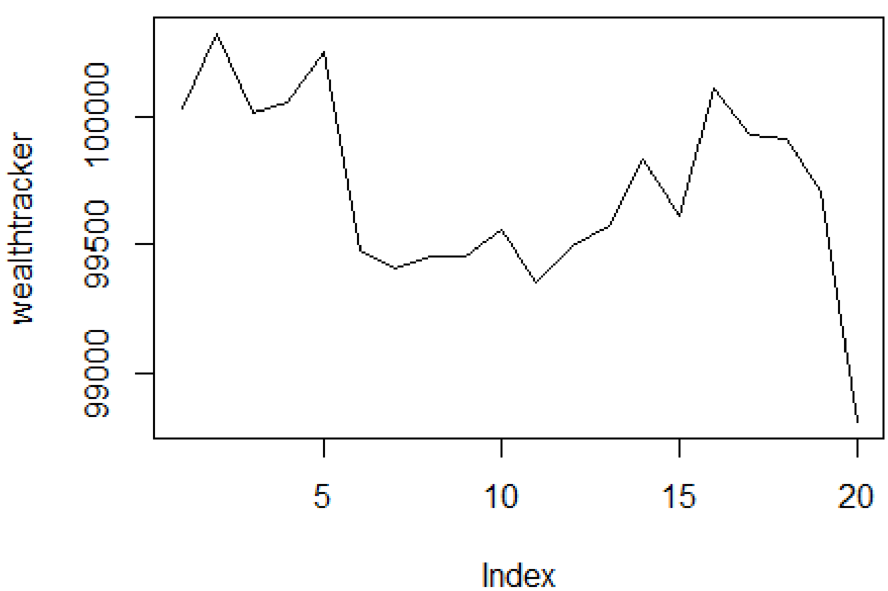

```
# conclusion: returns uncorrelated from one day to the next
# (makes sense, otherwise it'd be an easy inefficiency to exploit,
# and market inefficiencies that are exploited tend to disappear as a result)

#### Now use a bootstrap approach
#### With more stocks
#mystocks = c("VIS", "PBE", "IXC","XNTK","VWO")

#myprices = getSymbols(mystocks, from = "2015-01-01")


# A chunk of code for adjusting all stocks
# creates a new object adding 'a' to the end
# For example, WMT becomes WMTa, etc
for(ticker in mystocks) {
  expr = paste0(ticker, "a = adjustOHLC(", ticker, ")")
  eval(parse(text=expr))
}

head(USMVa)

##            USMV.Open USMV.High USMV.Low USMV.Close USMV.Volume
USMV.Adjusted
## 2015-08-10  37.94760  38.08284 37.94760   38.05579      414000
```

```
38.05579
## 2015-08-11  37.95662  38.03776 37.84843   37.97465     2229400
37.97464
## 2015-08-12  37.84843  38.10087 37.61402   38.08284     1055200
38.08285
## 2015-08-13  38.10087  38.23611 37.96564   38.09186      741700
38.09185
## 2015-08-14  38.04678  38.28119 38.03776   38.27217      949000
38.27217
## 2015-08-17  38.25414  38.45249 38.06481   38.43446      915300
38.43446

# Combine all the returns in a matrix


# Compute the returns from the closing prices
pairs(all_returns)
```




```
# Sample a random return from the empirical joint distribution
# This simulates a random day
return.today = resample(all_returns, 1, orig.ids=FALSE)

# Update the value of your holdings
# Assumes an equal allocation to each asset
total_wealth = 100000
my_weights = c(0.2,0.2,0.2, 0.2, 0.2)
holdings = total_wealth*my_weights
```

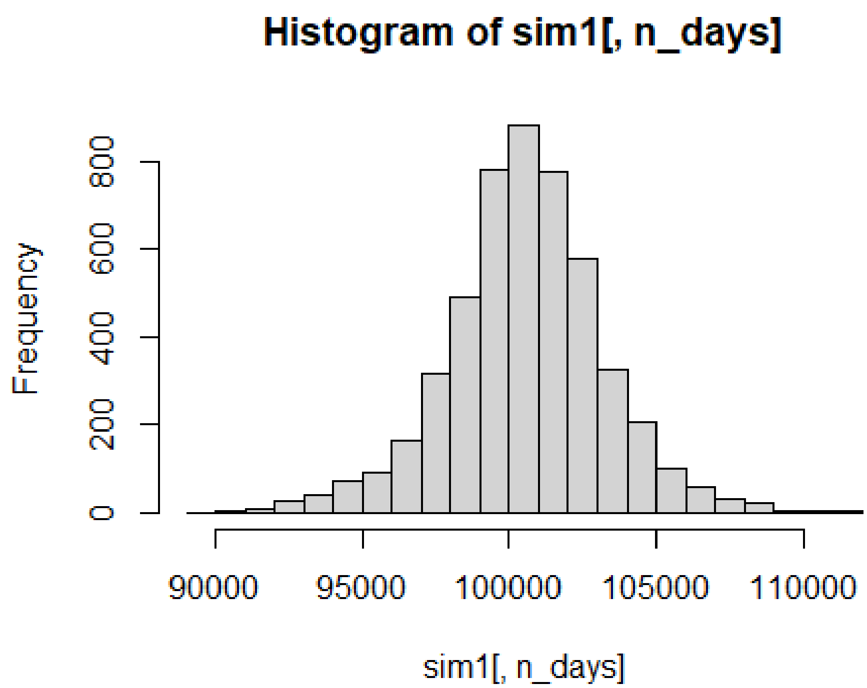
```r
holdings = holdings*(1 + return.today)

holdings

##            ClCl.SHYa ClCl.SPLVa ClCl.USMVa ClCl.IEFa ClCl.EFAVa
## 2017-04-18  20016.54   20013.78   20004.18  20114.26   19969.95

# Now loop over two trading weeks
# let's run the following block of code 5 or 6 times
# to eyeball the variability in performance trajectories

## begin block
n_days = 20
wealthtracker = rep(0, n_days) # Set up a placeholder to track total wealth
for(today in 1:n_days) {
  return.today = resample(all_returns, 1, orig.ids=FALSE)
  holdings = holdings + holdings*return.today
  total_wealth = sum(holdings)
  wealthtracker[today] = total_wealth
}
total_wealth

## [1] 98808.36

plot(wealthtracker, type='l')
```



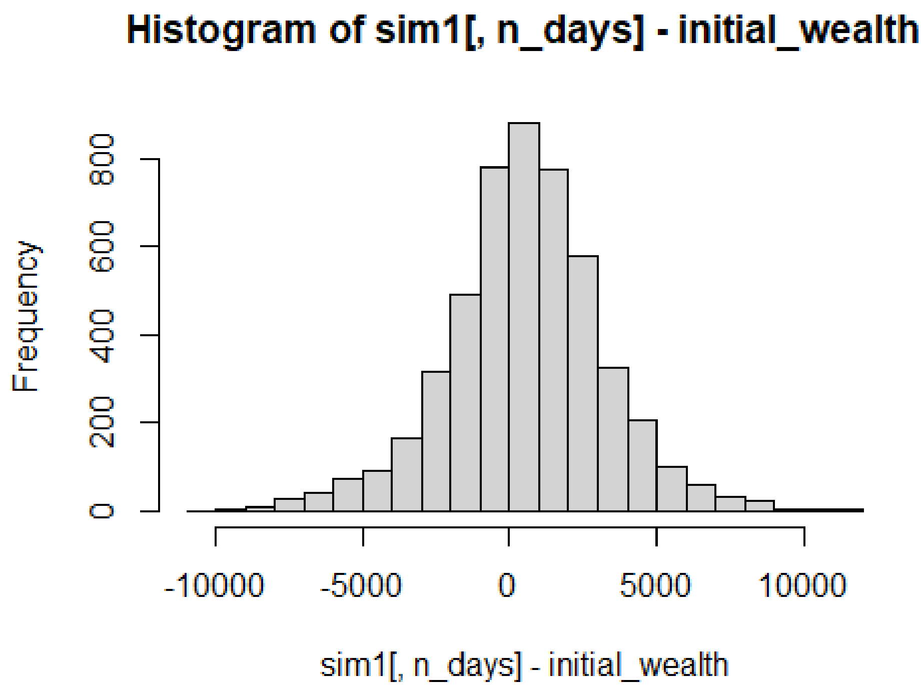
```
## end block

# Now simulate many different possible futures
# just repeating the above block thousands of times
initial_wealth = 100000
sim1 = foreach(i=1:5000, .combine='rbind') %do% {
  total_wealth = initial_wealth
  weights = c(0.2, 0.2, 0.2, 0.2, 0.2)
  holdings = weights * total_wealth
  n_days = 20
  wealthtracker = rep(0, n_days)
  for(today in 1:n_days) {
    return.today = resample(all_returns, 1, orig.ids=FALSE)
    holdings = holdings + holdings*return.today
    total_wealth = sum(holdings)
    wealthtracker[today] = total_wealth
  }
  wealthtracker
}

# each row is a simulated trajectory
# each column is a data
head(sim1)
```

```
##               [,1]      [,2]      [,3]     [,4]     [,5]      [,6]
[,7]
## result.1 100026.19 100845.87 101235.98 101214.4 101759.9 102072.64
102388.33
## result.2 100124.60 100437.82 100511.48 100646.7 100921.2 100970.07
100904.41
## result.3 100289.18 100244.73 100270.27 100377.7 100586.8 100912.78
101135.11
## result.4 100197.45 100374.03 100435.98 100644.4 100665.7 100623.51
101811.33
## result.5 100099.63 100264.52 100344.61 100380.3 100548.6 100856.94
101018.88
## result.6  99917.44  99208.28  98770.66  98951.9  98393.4  98538.38
98323.86
##               [,8]     [,9]    [,10]    [,11]    [,12]    [,13]    [,14]
## result.1 102215.70 102200.2 101781.9 101998.8 101658.6 101822.7 101991.1
## result.2 101839.74 101328.3 101484.9 101745.4 101817.1 101599.5 102034.0
## result.3 101847.59 101639.6 101126.3 100604.9 100165.5 100167.0  99951.8
## result.4 102312.15 102645.6 102502.9 102411.0 102460.3 101799.9 101676.7
## result.5 101295.43 101091.2 101026.0 101797.5 101737.6 101744.0 103290.8
## result.6  97879.01 101535.3 101218.4 100878.4 100724.6 100929.5 100846.5
##             [,15]    [,16]    [,17]    [,18]    [,19]     [,20]
## result.1 101831.6 101477.1 101793.8 100614.4 100013.7  99806.57
## result.2 101784.2 101685.9 101810.1 102035.5 102086.8 102111.01
## result.3 100765.4 100806.9 100486.2 100154.6 104048.9 104131.21
## result.4 101615.3 102505.4 102533.1 102078.4 101993.9 102075.27
```

```
## result.5 103514.6 103771.5 103981.0 104308.0 104331.4 104809.16
## result.6 100723.5 100482.0 100557.5 100670.1 101000.7 100915.28

hist(sim1[,n_days], 25)
```

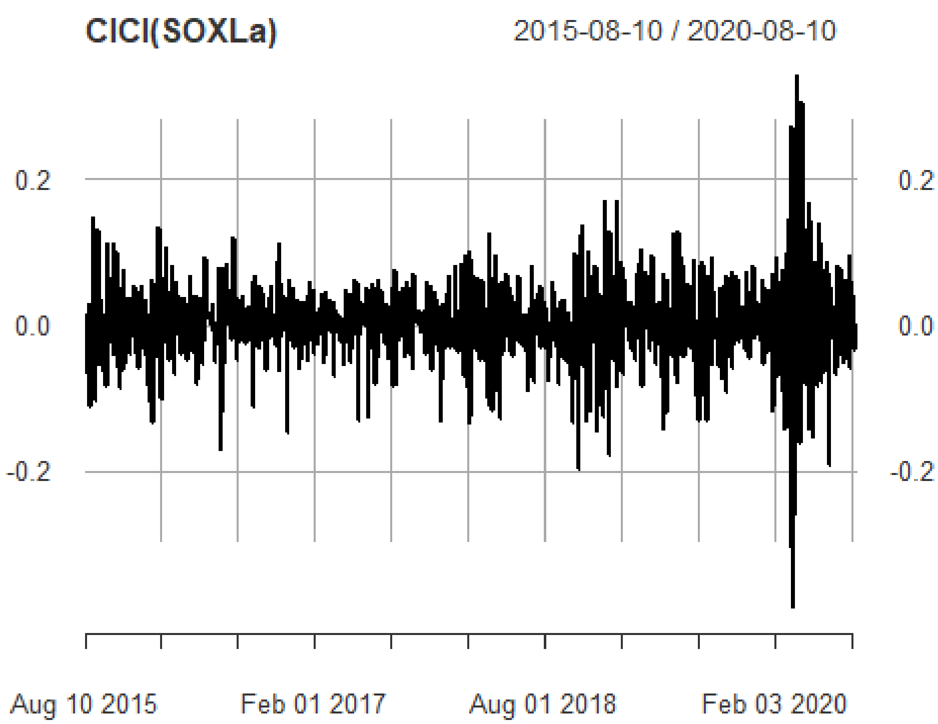



```
# Profit/loss
mean(sim1[,n_days])

## [1] 100543.3

mean(sim1[,n_days] - initial_wealth)

## [1] 543.2602

hist(sim1[,n_days]- initial_wealth, breaks=30)
```

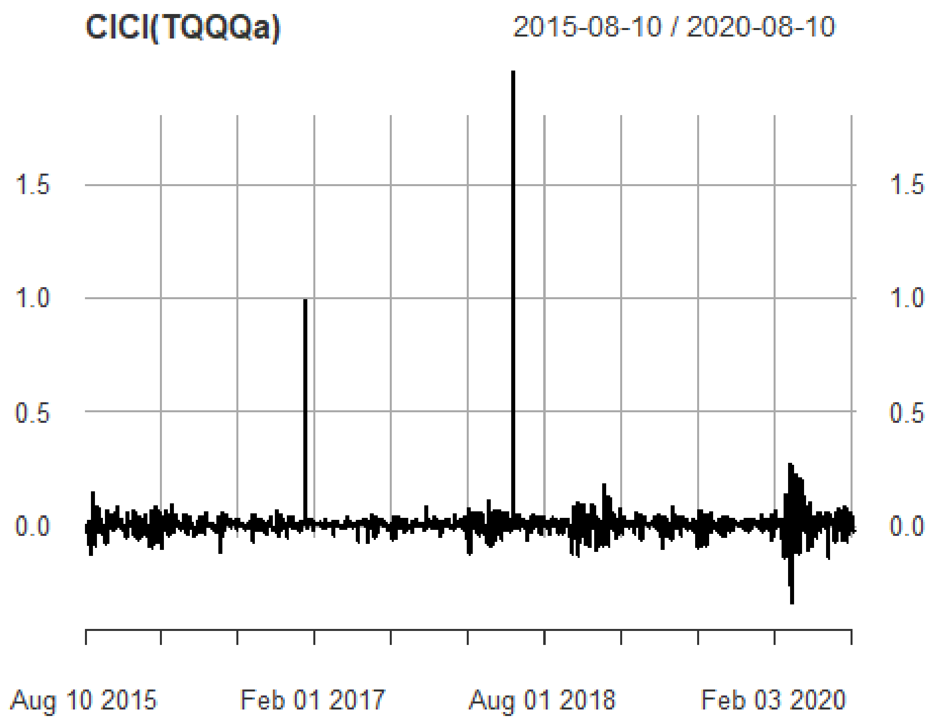



```r
# 5% value at risk:
quantile(sim1[,n_days]- initial_wealth, prob=0.05)

##        5%
## -3937.856

# note: this is  a negative number (a loss, e.g. -500), but we conventionally
# express VaR as a positive number (e.g. 500)

mystocks = c("SOXL", "TQQQ", "ROM","TECL","VGT")
getSymbols(mystocks)

## [1] "SOXL" "TQQQ" "ROM"  "TECL" "VGT"

SOXL <- SOXL["2015-08-10::2020-08-10"]
TQQQ <- TQQQ["2015-08-10::2020-08-10"]
ROM <- ROM["2015-08-10::2020-08-10"]
TECL <- TECL["2015-08-10::2020-08-10"]
VGT <- VGT["2015-08-10::2020-08-10"]

# Adjust for splits and dividends
SOXLa = adjustOHLC(SOXL)
TQQQa = adjustOHLC(TQQQ)
ROMa = adjustOHLC(ROM)
TECLa = adjustOHLC(TECL)
VGTa = adjustOHLC(VGT)
```

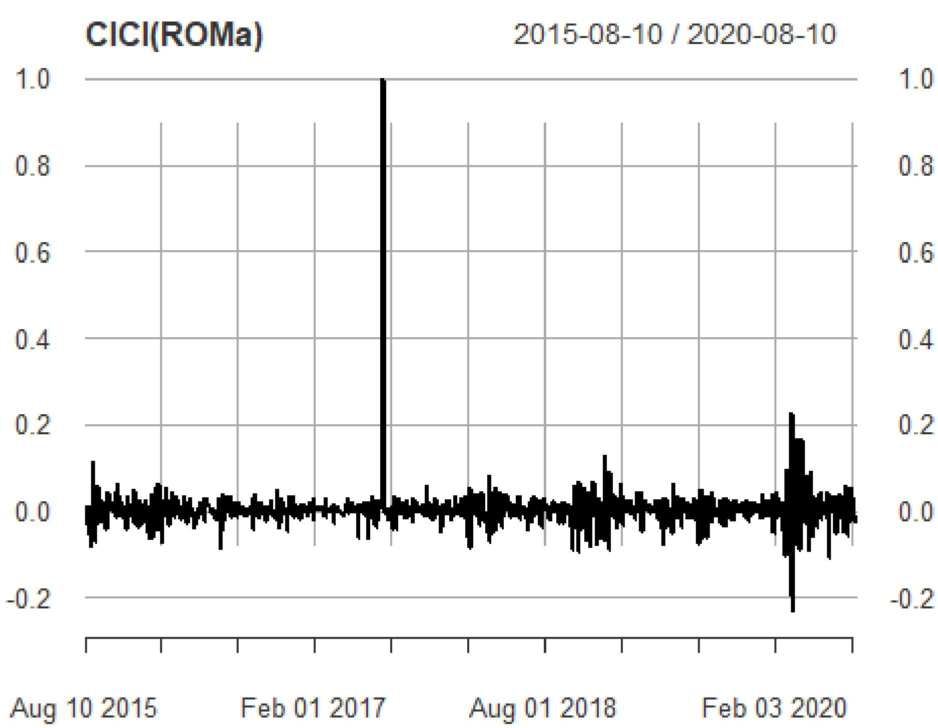
```r
# Look at close-to-close changes
plot(ClCl(SOXLa))
```



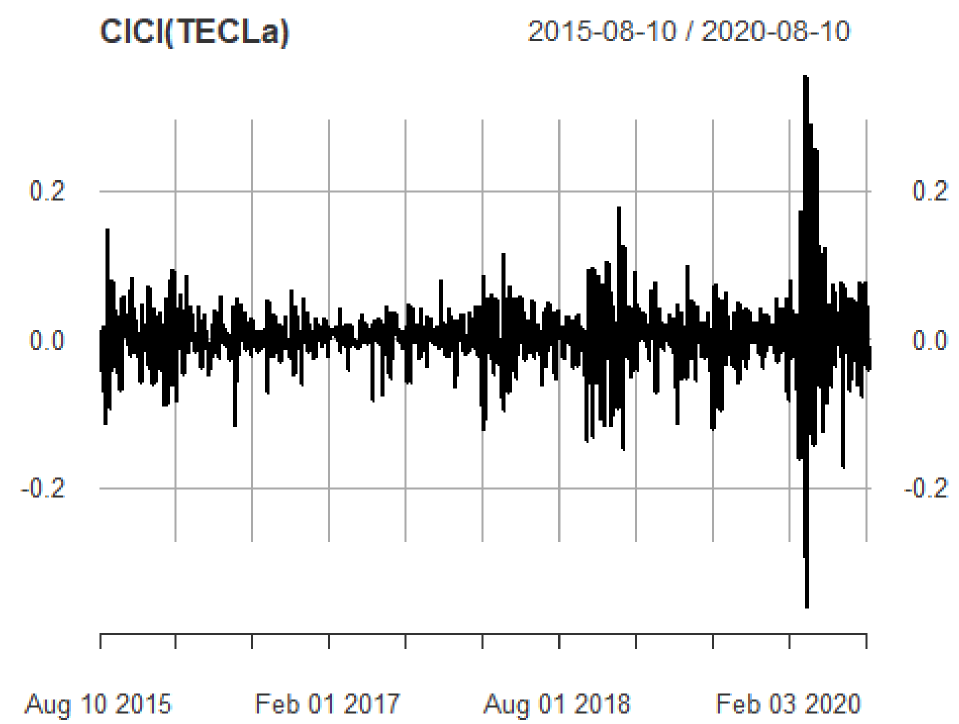

```r
plot(ClCl(TQQQa))
```



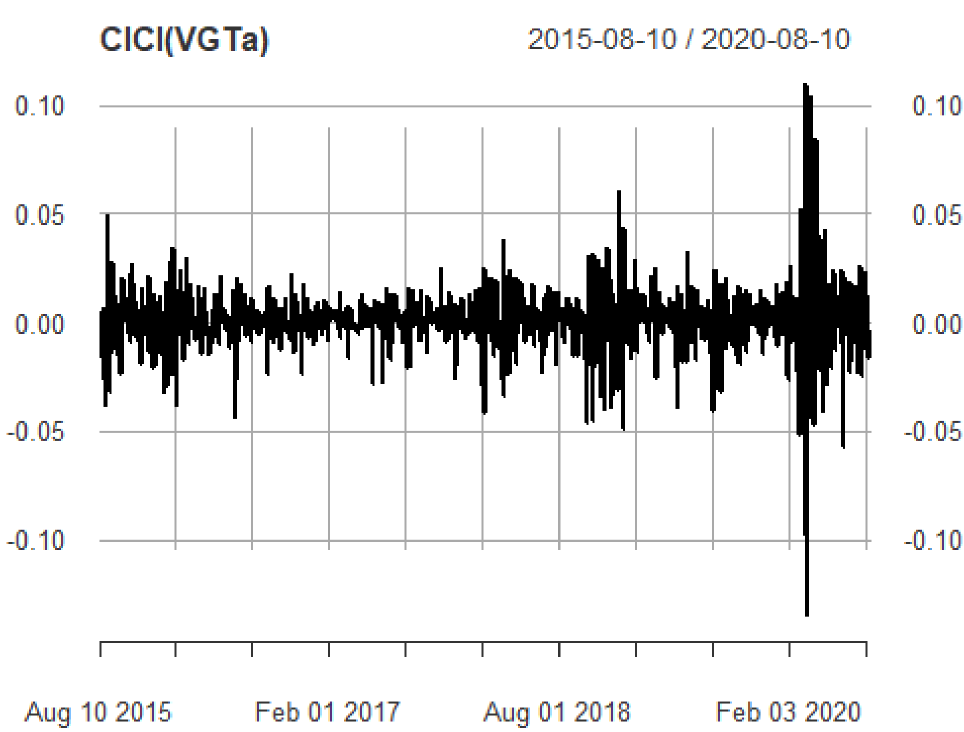

```r
plot(ClCl(ROMa))
```




```r
plot(ClCl(TECLa))
```



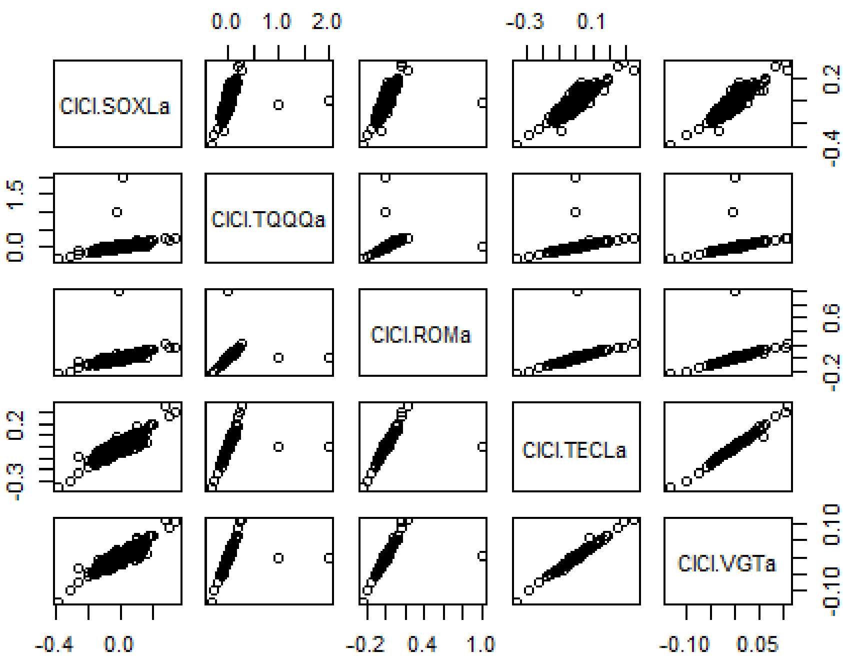

```
plot(ClCl(VGTa))
```



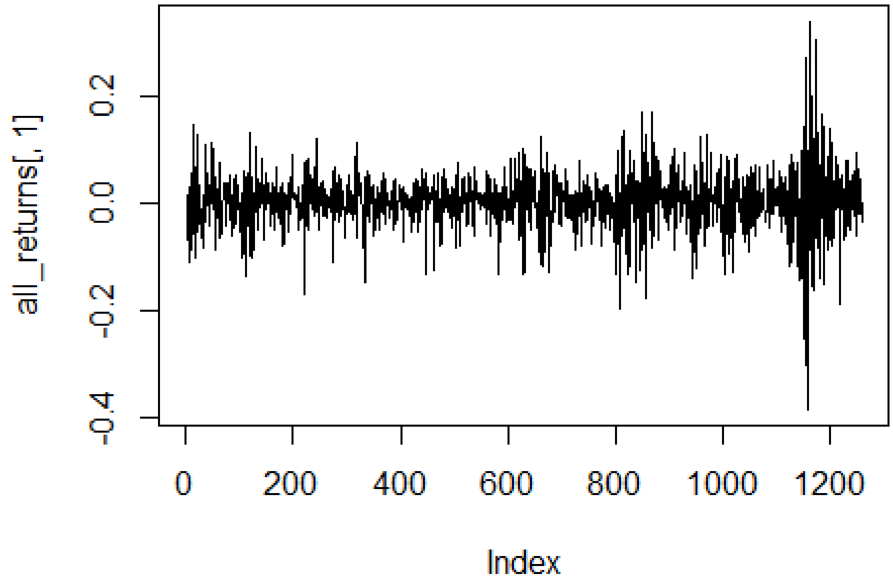
```
# Combine close to close changes in a single matrix
all_returns =
cbind(ClCl(SOXLa),ClCl(TQQQa),ClCl(ROMa),ClCl(TECLa),ClCl(VGTa))
head(all_returns)
```

```
##              ClCl.SOXLa   ClCl.TQQQa    ClCl.ROMa   ClCl.TECLa    ClCl.VGTa
## 2015-08-10           NA           NA           NA           NA           NA
## 2015-08-11 -0.06798249 -0.038128800 -0.032714362 -0.044438582 -0.016334762
## 2015-08-12  0.01686275  0.008818633  0.010838365  0.011974380  0.004851171
## 2015-08-13 -0.03046668 -0.004500571 -0.002971244 -0.006053963 -0.002506703
## 2015-08-14 -0.01750191  0.005390350  0.006867064  0.012458500  0.005305287
## 2015-08-17  0.02995951  0.024558959  0.012482306  0.017500656  0.006480854
```

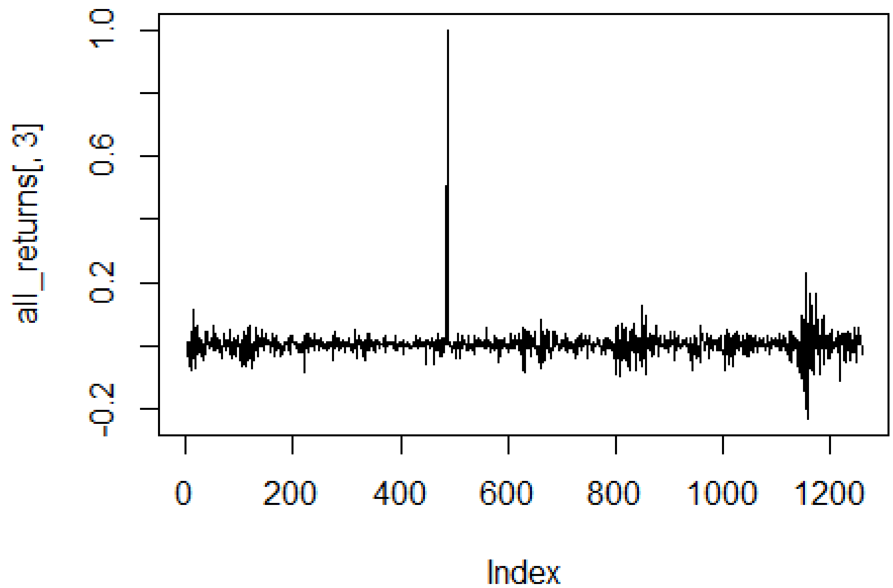
```
# first row is NA because we didn't have a "before" in our data
all_returns = as.matrix(na.omit(all_returns))
N = nrow(all_returns)

# These returns can be viewed as draws from the joint distribution
# strong correlation, but certainly not Gaussian!
pairs(all_returns)
```



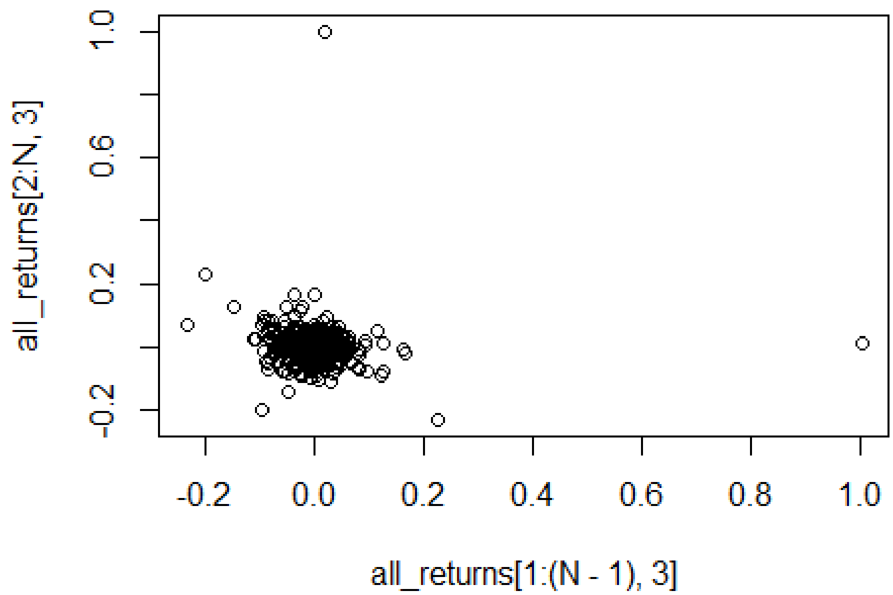

```
plot(all_returns[,1], type='l')
```



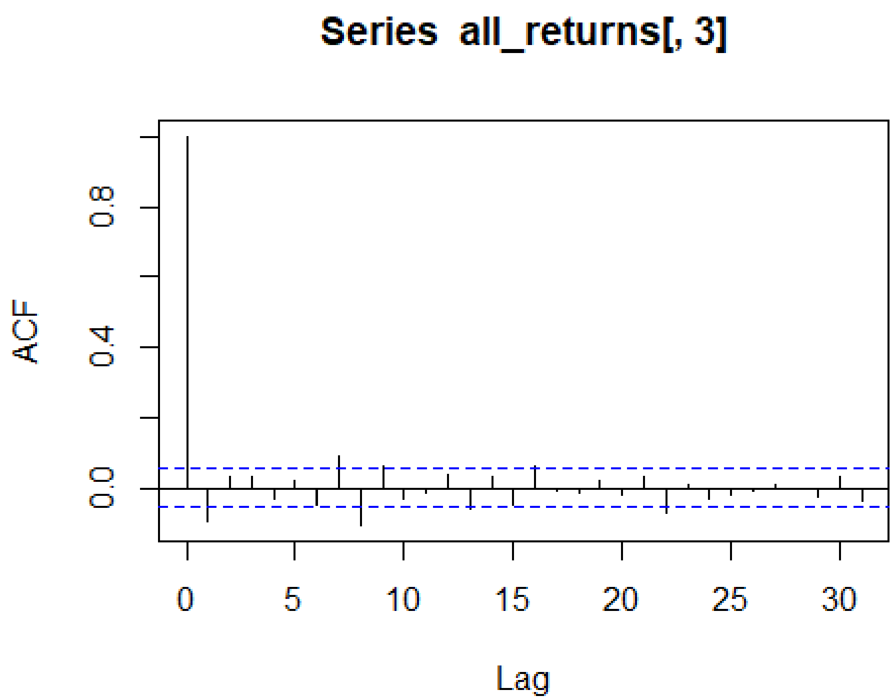

```r
# Look at the market returns over time
plot(all_returns[,3], type='l')
```



```r
# are today's returns correlated with tomorrow's?
# not really!
plot(all_returns[1:(N-1),3], all_returns[2:N,3])
```

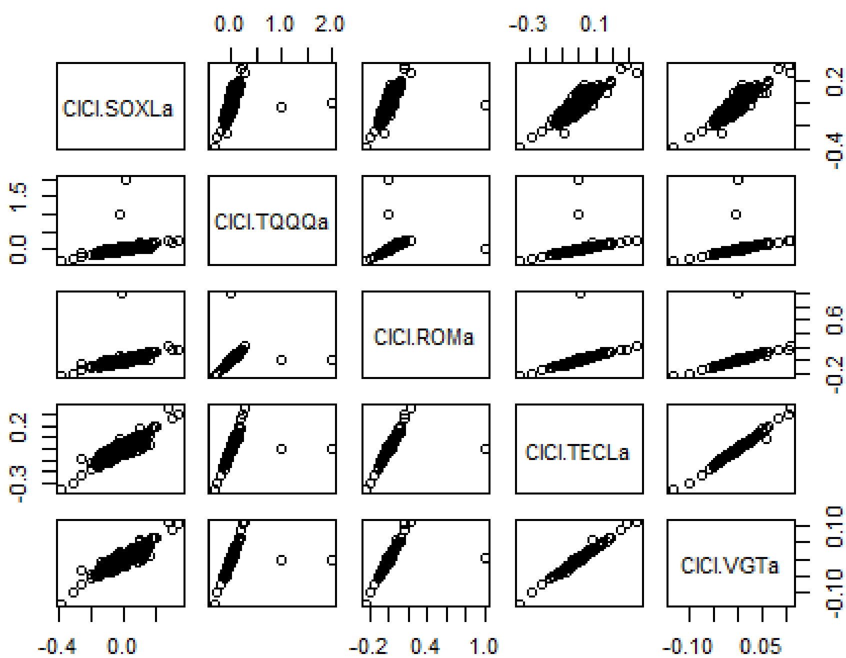



```r
# An autocorrelation plot: nothing there
acf(all_returns[,3])
```



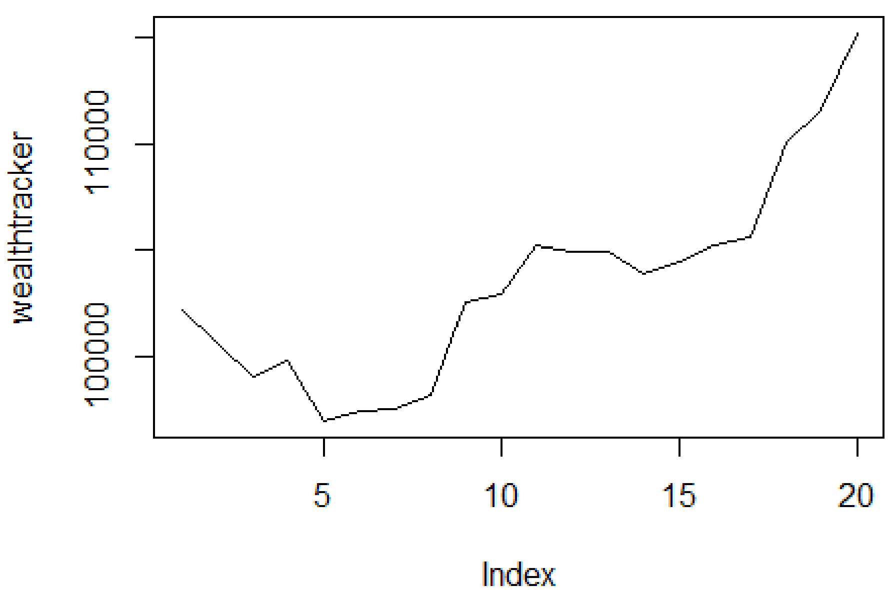

```
# conclusion: returns uncorrelated from one day to the next
# (makes sense, otherwise it'd be an easy inefficiency to exploit,
# and market inefficiencies that are exploited tend to disappear as a result)

#### Now use a bootstrap approach
#### With more stocks
#mystocks = c("VIS", "PBE", "IXC","XNTK","VWO")

#myprices = getSymbols(mystocks, from = "2015-01-01")


# A chunk of code for adjusting all stocks
# creates a new object adding 'a' to the end
# For example, WMT becomes WMTa, etc
for(ticker in mystocks) {
  expr = paste0(ticker, "a = adjustOHLC(", ticker, ")")
  eval(parse(text=expr))
}

head(USMVa)

##            USMV.Open USMV.High USMV.Low USMV.Close USMV.Volume
USMV.Adjusted
## 2015-08-10  37.94760  38.08284 37.94760   38.05579      414000
38.05579
## 2015-08-11  37.95662  38.03776 37.84843   37.97465     2229400
```

```
37.97464
## 2015-08-12  37.84843  38.10087 37.61402   38.08284     1055200
38.08285
## 2015-08-13  38.10087  38.23611 37.96564   38.09186      741700
38.09185
## 2015-08-14  38.04678  38.28119 38.03776   38.27217      949000
38.27217
## 2015-08-17  38.25414  38.45249 38.06481   38.43446      915300
38.43446

# Combine all the returns in a matrix


# Compute the returns from the closing prices
pairs(all_returns)
```



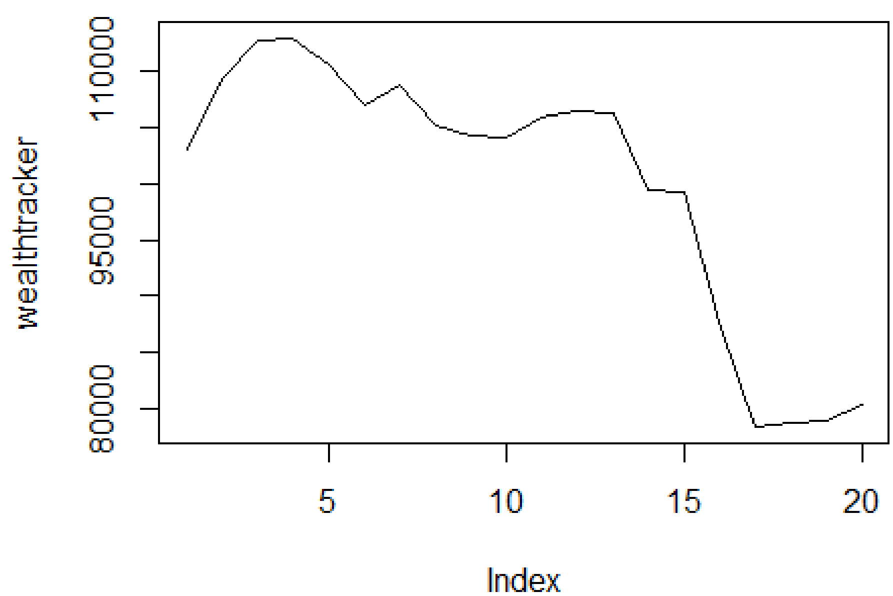

```
# Sample a random return from the empirical joint distribution
# This simulates a random day
return.today = resample(all_returns, 1, orig.ids=FALSE)

# Update the value of your holdings
# Assumes an equal allocation to each asset
total_wealth = 100000
my_weights = c(0.2,0.2,0.2, 0.2, 0.2)
holdings = total_wealth*my_weights
holdings = holdings*(1 + return.today)
```

```
holdings

##             ClCl.SOXLa ClCl.TQQQa ClCl.ROMa ClCl.TECLa ClCl.VGTa
## 2020-08-04    20859.2   20260.03  20044.85   20179.95  20065.53

# Now loop over two trading weeks
# let's run the following block of code 5 or 6 times
# to eyeball the variability in performance trajectories

## begin block
n_days = 20
wealthtracker = rep(0, n_days) # Set up a placeholder to track total wealth
for(today in 1:n_days) {
  return.today = resample(all_returns, 1, orig.ids=FALSE)
  holdings = holdings + holdings*return.today
  total_wealth = sum(holdings)
  wealthtracker[today] = total_wealth
}
total_wealth

## [1] 115212.3

plot(wealthtracker, type='l')
```




```
## end block
```

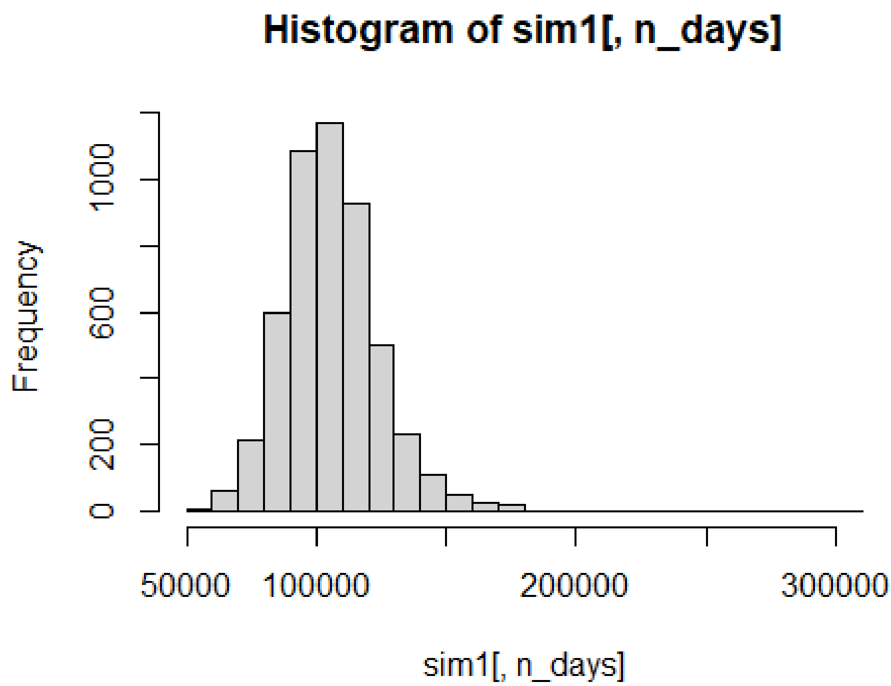
```
# Now simulate many different possible futures
# just repeating the above block thousands of times
initial_wealth = 100000
sim1 = foreach(i=1:5000, .combine='rbind') %do% {
  total_wealth = initial_wealth
  weights = c(0.2, 0.2, 0.2, 0.2, 0.2)
  holdings = weights * total_wealth
  n_days = 20
  wealthtracker = rep(0, n_days)
  for(today in 1:n_days) {
    return.today = resample(all_returns, 1, orig.ids=FALSE)
    holdings = holdings + holdings*return.today
    total_wealth = sum(holdings)
    wealthtracker[today] = total_wealth
  }
  wealthtracker
}
plot(wealthtracker, type='l')
```



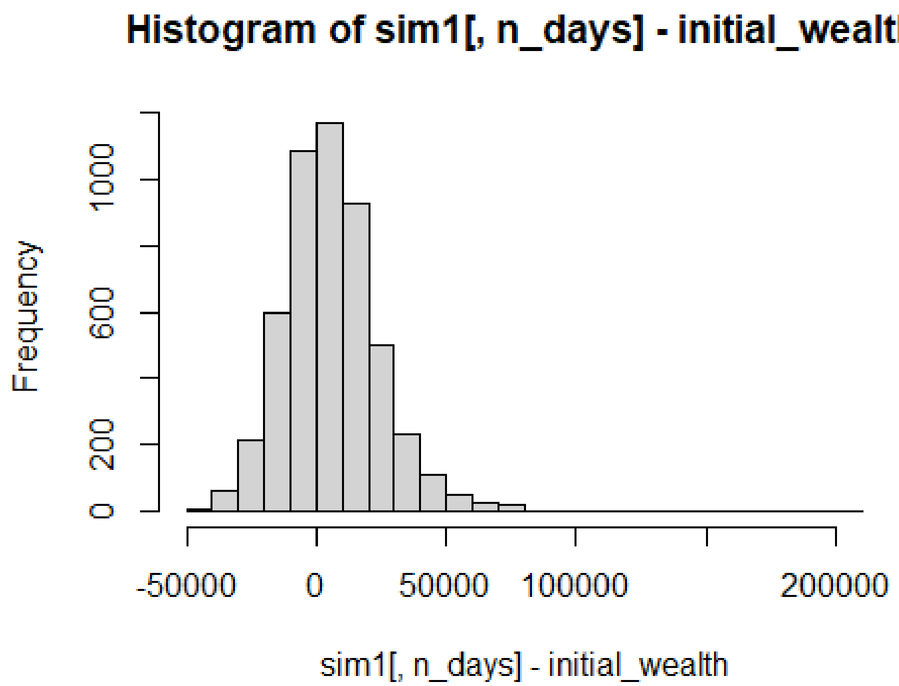

```
# each row is a simulated trajectory
# each column is a data
head(sim1)

##                [,1]      [,2]     [,3]     [,4]      [,5]      [,6]
[,7]
## result.1 101327.18 101789.37 102554.4  92477.9  95087.88 100619.57
101913.5
```

```
## result.2 103228.81 102194.19 101850.2 102743.6 105290.34 108204.78
107597.0
## result.3  99318.68 104084.23 102581.5 104338.5 101056.91 100612.02
102163.4
## result.4 100355.25 101420.57 100465.9 100891.8 103378.98 103593.88
103401.0
## result.5 100125.29  99800.53 100257.4 101644.0 100473.52  97946.73
97784.7
## result.6 105874.65 106113.05 106626.8 109886.2 111073.59 117445.29
119286.1
##               [,8]      [,9]     [,10]     [,11]     [,12]     [,13]
[,14]
## result.1  98554.36  91816.08  91828.47  94456.16  90941.33  91725.34
93067.62
## result.2 108522.04 112088.08 112653.85 114884.93 118865.65 120001.36
117473.69
## result.3 100574.62 111761.99 112421.69 107911.51 112707.25 125378.24
128781.98
## result.4  97822.16  98213.58  96557.53 100984.74 100944.33  98726.67
98846.70
## result.5  96036.30  96320.29  88170.57  72595.53  72971.27  73733.73
76594.78
## result.6 122542.09 123396.69 118814.98 119602.37 120782.31 120500.61
123275.44
##              [,15]     [,16]     [,17]     [,18]     [,19]     [,20]
## result.1  93512.05  95712.57 132118.94 134009.69 129163.83 130898.62
## result.2 118016.21 119089.72 112694.54 112504.04 110823.62 112862.37
## result.3 135380.75 135240.54 131071.02 131135.31 135628.53 135854.92
## result.4 102113.74 102532.65 106592.03 100475.12  99821.33  93521.49
## result.5  77404.99  76767.18  75664.82  69199.46  69509.62  70657.60
## result.6 124189.91 126359.46 126846.16 129497.81 128860.63 134598.47
```

```
hist(sim1[,n_days], 25)
```




```
# Profit/loss
mean(sim1[,n_days])

## [1] 105786.6

mean(sim1[,n_days] - initial_wealth)

## [1] 5786.605

hist(sim1[,n_days]- initial_wealth, breaks=30)
```




```r
# 5% value at risk:
quantile(sim1[,n_days]- initial_wealth, prob=0.05)

##        5%
## -20791.28
```

Question 4

First, the data was preprocessed to ensure that the results were as meaningful of as possible. This meant removing all social media users who were indicated as writing spam, anything categorized as adult, and uncategorized. These users were not the main audience for the product which is why they were removed from the analysis. After conducting all the data preprocessing, a PCA was conducted on the cleaned data. The first result of the PCA had all loadings included, but after reviewing the result it was determined that 16 loadings would be enough as it accounted for 80% of the variance. This did not mean that all the marketing segments would be used for this analysis because after closely reviewing the loads only about 12 of the loads were of interest as some of them did not really provide a lot of insight.

Market segment 1: High… * Religion * Food * Parenting * School * Family * Beauty * Crafts

Market Segment 2: High… * Religion * Food * Parenting * Sports fandom * Food * School * Family Low… * Cooking * Photo sharing * Fashion

Market Segment 3: High… * Politics * Travel * Computers * News * Automotive Low… * Health and nutrient * Personal fitness

Market Segment 4: High… * Health and Nutrient * Personal fitness * Outdoors Low… * College and University * Online gaming

Market Segment 5: High… * Photo sharing Low… * College and university * Online gaming

Market Segment 6: High… * Beauty * Cooking * Fashion * Chatter * Shopping

Market Segment 7: High… * Automotive * Online Gaming * Sports playing Low… * Arts * TV Film

Market Segment 8: High… * Automotive * News * Tv film * Computers * Travel

Market Segment 9: High… * Dating * Home and gardening * School Low… * Music

Market Segment 10: High… * Music * Small business Low… * Arts * Crafts

Market Segment 11: High… * Home and gardening * Current events * Eco Low… * Business * Craft

Market Segment 12: High… * Music * Eco Low… * Small Business * Business

```
library(tidyverse)
library(randomForest)
library(splines)

socialmarketing <-
read.csv("C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/social_marketi
ng.csv")
#data cleaning
socialmarketing <- subset(socialmarketing, socialmarketing$spam != 1 |
socialmarketing$adult != 1)
socialmarketing <- socialmarketing[,c(-36,-37,-6)]
socialmarketing_PCA = prcomp(socialmarketing[,-1], rank=34, scale=TRUE)
head(socialmarketing_PCA$rotation)

##                       PC1         PC2         PC3         PC4         PC5
## chatter        0.12434039 -0.20982909  0.06524036 -0.11722123  0.18510271
## current_events 0.09699472 -0.07067962  0.04916933 -0.03246847  0.05611240
## travel         0.11730148 -0.05389072  0.42369620  0.14286147  0.01119277
## photo_sharing  0.17781387 -0.31768609 -0.02387418 -0.15742650  0.21634222
## tv_film        0.09362971 -0.07034052  0.08803030 -0.08908232 -0.20446274
## sports_fandom  0.29469739  0.31163967 -0.04609136 -0.05311160  0.03602160
##                         PC6         PC7         PC8         PC9
PC10
## chatter        -0.457377313  0.11985776 -0.07745263  0.09628736 -
0.01404138
## current_events -0.138583473 -0.03977839  0.05924758 -0.08925141
0.09406931
## travel          0.162193765 -0.10163928 -0.30191648 -0.10922170 -
```

```
0.04346621
## photo_sharing  -0.204901452  0.11564007 -0.02184555 -0.12432176 -
0.11712987
## tv_film        -0.073695612 -0.52145628  0.23925268 -0.07782077
0.06701168
## sports_fandom  -0.009831838  0.07903071  0.10531665 -0.04751409
0.04410351
##                       PC11          PC12         PC13        PC14
PC15
## chatter        -0.06861798 -0.0256648479  0.031322782  0.08911691
0.111154628
## current_events  0.52102984  0.7931842845 -0.132203431 -0.12535225
0.009037982
## travel          0.05303062 -0.0195845537  0.029047436  0.04980604
0.015196973
## photo_sharing  -0.01045177 -0.0931551471  0.049490001 -0.01435000
0.054631909
## tv_film        -0.05937961 -0.0007197019  0.036967552  0.10786090 -
0.046027340
## sports_fandom  -0.02542150 -0.0131149773  0.005355396 -0.02308185 -
0.018847501
##                       PC16          PC17         PC18         PC19
PC20
## chatter        -0.13617038 -0.0072142691 -0.007999102  0.129032479
0.04408413
## current_events  0.03263011 -0.0001772328 -0.006482393 -0.018767323 -
0.01739717
## travel         -0.09194504  0.0227190818 -0.001256605 -0.004429809 -
0.04492248
## photo_sharing  -0.07076666 -0.0423932573  0.003985895 -0.053313244 -
0.06331564
## tv_film        -0.26802068  0.0658057771  0.094225788  0.071502228 -
0.21113410
## sports_fandom  -0.01769400 -0.1280913874  0.018335674  0.097364976 -
0.14613414
##                       PC21        PC22         PC23         PC24
PC25
## chatter         0.11389699 -0.09291684 -0.135977662 -0.212028466 -
0.0001442715
## current_events  0.01353753  0.01723101 -0.009880252  0.001587184 -
0.0001560642
## travel         -0.07638891  0.03136240  0.207438671 -0.224706600 -
0.3397394845
## photo_sharing   0.16065752 -0.26896243 -0.221546022 -0.389312849
0.1152693914
## tv_film        -0.21396447 -0.55890816  0.087153576  0.104145524 -
0.0454526295
## sports_fandom   0.07011800 -0.05424472  0.122561944  0.079348635
0.5827530313
##                       PC26         PC27         PC28        PC29
```

```
PC30
## chatter        -0.31130188 -0.135684225  0.600203219  0.12745889 -
0.0045253373
## current_events  0.02731926 -0.008395394 -0.003171978 -0.02466891
0.0005984978
## travel         -0.05967286  0.359759745  0.111862737 -0.24280762 -
0.4479775621
## photo_sharing   0.29673070  0.230420480 -0.468741794  0.01806054 -
0.0313608728
## tv_film        -0.03435094 -0.122190254 -0.022116674 -0.03257722
0.0582150174
## sports_fandom  -0.13874080  0.416525005  0.156277640 -0.38532200
0.0614328419
##                       PC31         PC32         PC33
## chatter        -0.09076177  0.010922486  0.009432833
## current_events -0.01161154  0.005481428 -0.005872545
## travel         -0.04918875  0.015489535 -0.018634315
## photo_sharing   0.07931095 -0.021265310 -0.034530550
## tv_film        -0.01689476  0.166292214 -0.051227993
## sports_fandom  -0.00597990 -0.008890838 -0.014946669
```

```r
summary(socialmarketing_PCA)
```

```
## Importance of components:
##                          PC1     PC2     PC3     PC4    PC5     PC6
PC7
## Standard deviation     2.110 1.68608 1.59286 1.53355 1.4792 1.36842
1.27377
## Proportion of Variance 0.135 0.08615 0.07689 0.07127 0.0663 0.05674
0.04917
## Cumulative Proportion  0.135 0.22112 0.29800 0.36927 0.4356 0.49232
0.54148
##                            PC8     PC9   PC10   PC11    PC12    PC13
PC14
## Standard deviation     1.19216 1.06095 0.9933 0.9681 0.96178 0.93977
0.92323
## Proportion of Variance 0.04307 0.03411 0.0299 0.0284 0.02803 0.02676
0.02583
## Cumulative Proportion  0.58455 0.61866 0.6486 0.6770 0.70499 0.73176
0.75758
##                           PC15    PC16    PC17    PC18    PC19    PC20
PC21
## Standard deviation     0.91593 0.85450 0.80886 0.75371 0.69818 0.68810
0.65473
## Proportion of Variance 0.02542 0.02213 0.01983 0.01721 0.01477 0.01435
0.01299
## Cumulative Proportion  0.78301 0.80513 0.82496 0.84217 0.85694 0.87129
0.88428
##                           PC22    PC23    PC24    PC25    PC26    PC27
PC28
```

```
## Standard deviation     0.65059 0.63989 0.63728 0.61730 0.60211 0.59482
0.58787
## Proportion of Variance 0.01283 0.01241 0.01231 0.01155 0.01099 0.01072
0.01047
## Cumulative Proportion  0.89711 0.90952 0.92182 0.93337 0.94436 0.95508
0.96555
##                           PC29    PC30    PC31    PC32   PC33
## Standard deviation     0.55071 0.48575 0.47615 0.43880 0.4223
## Proportion of Variance 0.00919 0.00715 0.00687 0.00583 0.0054
## Cumulative Proportion  0.97474 0.98189 0.98876 0.99460 1.0000
```

```
socialmarketing_PCA2 = prcomp(socialmarketing[,-1], rank=16, scale=TRUE)
summary(socialmarketing_PCA2)
```

```
## Importance of first k=16 (out of 33) components:
##                          PC1     PC2     PC3     PC4    PC5     PC6
PC7
## Standard deviation     2.110 1.68608 1.59286 1.53355 1.4792 1.36842
1.27377
## Proportion of Variance 0.135 0.08615 0.07689 0.07127 0.0663 0.05674
0.04917
## Cumulative Proportion  0.135 0.22112 0.29800 0.36927 0.4356 0.49232
0.54148
##                           PC8     PC9   PC10   PC11    PC12    PC13
PC14
## Standard deviation     1.19216 1.06095 0.9933 0.9681 0.96178 0.93977
0.92323
## Proportion of Variance 0.04307 0.03411 0.0299 0.0284 0.02803 0.02676
0.02583
## Cumulative Proportion  0.58455 0.61866 0.6486 0.6770 0.70499 0.73176
0.75758
##                           PC15    PC16
## Standard deviation     0.91593 0.85450
## Proportion of Variance 0.02542 0.02213
## Cumulative Proportion  0.78301 0.80513
```

```
loadings = socialmarketing_PCA2$rotation %>%
  as.data.frame %>% rownames_to_column('Category')

loadings %>%
  select(Category, PC2) %>%
  arrange(desc(PC2))
```

```
##             Category         PC2
## 1           religion  0.31318929
## 2      sports_fandom  0.31163967
## 3          parenting  0.29012449
## 4               food  0.23377946
## 5             school  0.19415706
## 6             family  0.18932264
## 7               news  0.02229173
```

```
## 8        automotive  0.02031965
## 9            crafts  0.02008688
## 10         politics -0.03307619
## 11  home_and_garden -0.04839369
## 12        computers -0.05309321
## 13           travel -0.05389072
## 14              art -0.05491716
## 15          tv_film -0.07034052
## 16           dating -0.07061193
## 17   current_events -0.07067962
## 18    online_gaming -0.08343524
## 19              eco -0.09374260
## 20   small_business -0.09714565
## 21         business -0.10636760
## 22   sports_playing -0.10903269
## 23      college_uni -0.11264856
## 24         outdoors -0.11839813
## 25            music -0.14191969
## 26 personal_fitness -0.14935045
## 27 health_nutrition -0.15063628
## 28          chatter -0.20982909
## 29           beauty -0.21299485
## 30         shopping -0.22347106
## 31          fashion -0.28506359
## 32    photo_sharing -0.31768609
## 33          cooking -0.31977474
```

```
loadings %>%
  select(Category, PC3) %>%
  arrange(desc(PC3))
```

```
##             Category          PC3
## 1           politics  0.489199287
## 2             travel  0.423696202
## 3          computers  0.365368765
## 4               news  0.336846723
## 5         automotive  0.189918066
## 6           business  0.101837280
## 7     small_business  0.098296766
## 8            tv_film  0.088030302
## 9        college_uni  0.083677564
## 10           chatter  0.065240356
## 11     online_gaming  0.053276980
## 12               art  0.050728140
## 13    current_events  0.049169326
## 14    sports_playing  0.040492264
## 15          shopping  0.036967301
## 16            dating  0.030807397
## 17   home_and_garden  0.019968845
## 18            crafts  0.003101066
```

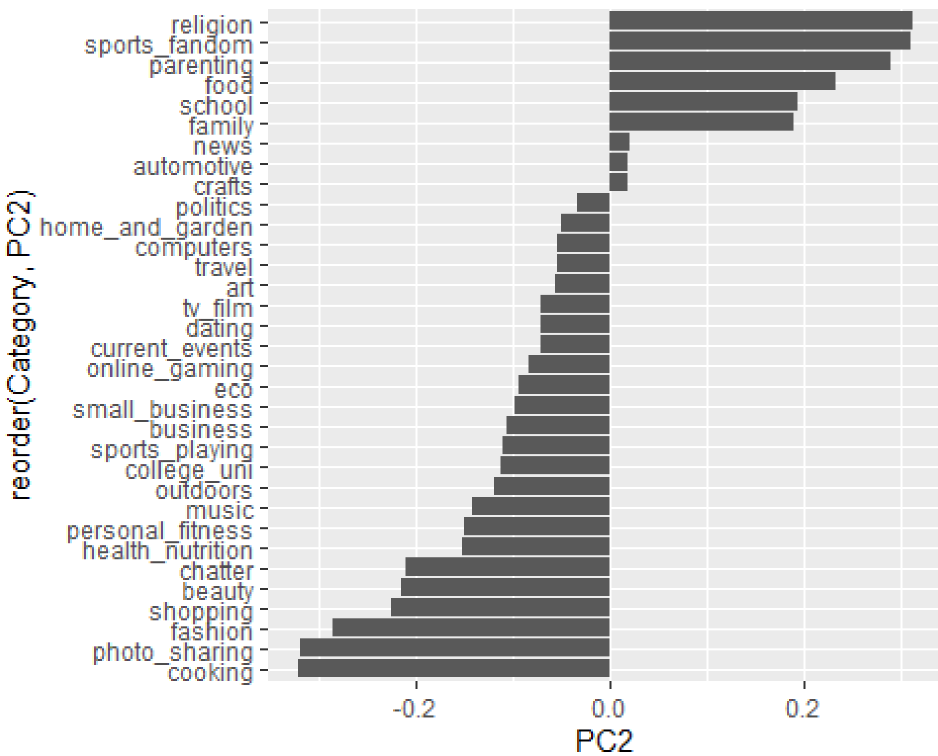
```
## 19            music -0.015176251
## 20    photo_sharing -0.023874179
## 21              eco -0.032698589
## 22    sports_fandom -0.046091360
## 23           family -0.047131188
## 24           school -0.078432899
## 25        parenting -0.084044430
## 26         religion -0.086883035
## 27             food -0.105701916
## 28         outdoors -0.141075535
## 29          fashion -0.148632764
## 30           beauty -0.158714742
## 31          cooking -0.204235400
## 32 personal_fitness -0.219407444
## 33 health_nutrition -0.227408956
```

```r
loadings %>%
  select(Category, PC4) %>%
  arrange(desc(PC4))
```

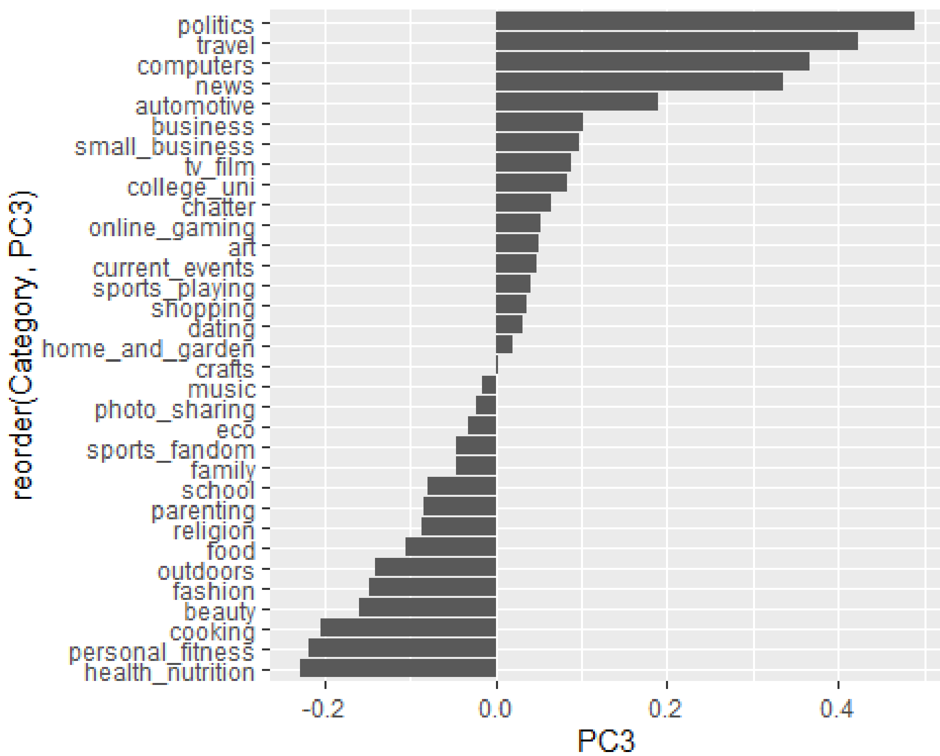
```
##            Category          PC4
## 1  health_nutrition  0.463218592
## 2  personal_fitness  0.443782383
## 3          outdoors  0.413458799
## 4          politics  0.194231190
## 5              news  0.176097051
## 6            travel  0.142861467
## 7         computers  0.135653450
## 8               eco  0.120197612
## 9              food  0.076474116
## 10       automotive  0.038028567
## 11           dating  0.027507581
## 12  home_and_garden  0.009065924
## 13         business -0.014359814
## 14          cooking -0.015092706
## 15           crafts -0.023027670
## 16   current_events -0.032468473
## 17        parenting -0.043698338
## 18    sports_fandom -0.053111605
## 19              art -0.061674210
## 20         religion -0.061955703
## 21           family -0.070429227
## 22   small_business -0.081253975
## 23            music -0.083573201
## 24           school -0.083715710
## 25          tv_film -0.089082322
## 26         shopping -0.107490730
## 27          chatter -0.117221234
## 28          fashion -0.142549602
## 29           beauty -0.150386888
```

```
## 30    photo_sharing -0.157426501
## 31   sports_playing -0.177019154
## 32    online_gaming -0.222104495
## 33      college_uni -0.256605911
```

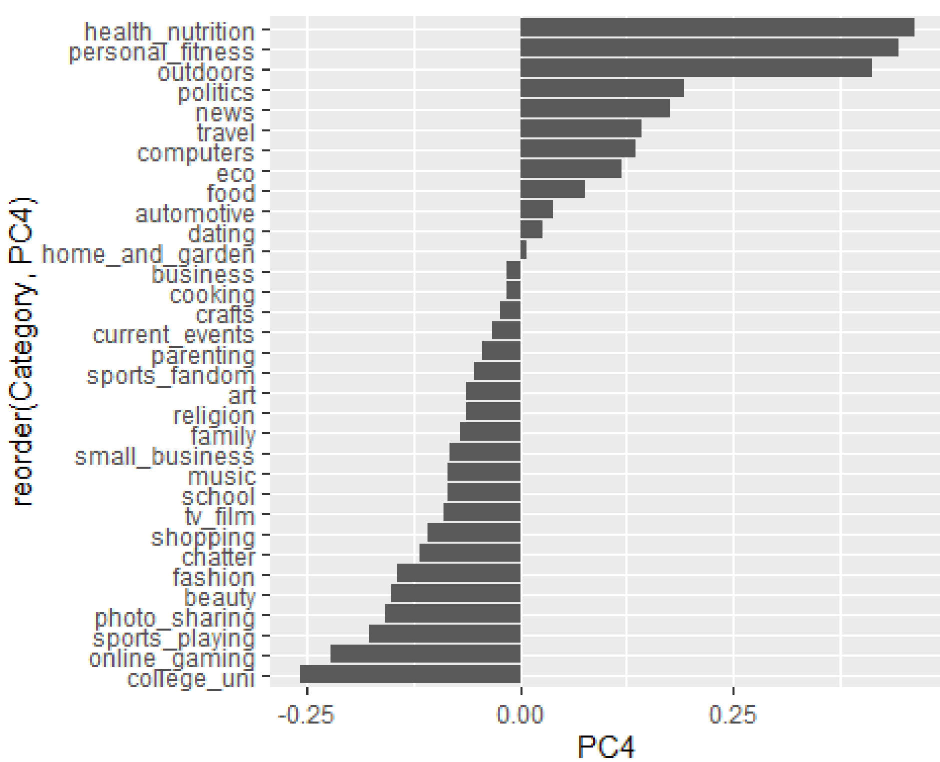
```r
ggplot(loadings) +
  geom_col(aes(x=reorder(Category, PC2), y=PC2)) +
  coord_flip()
```



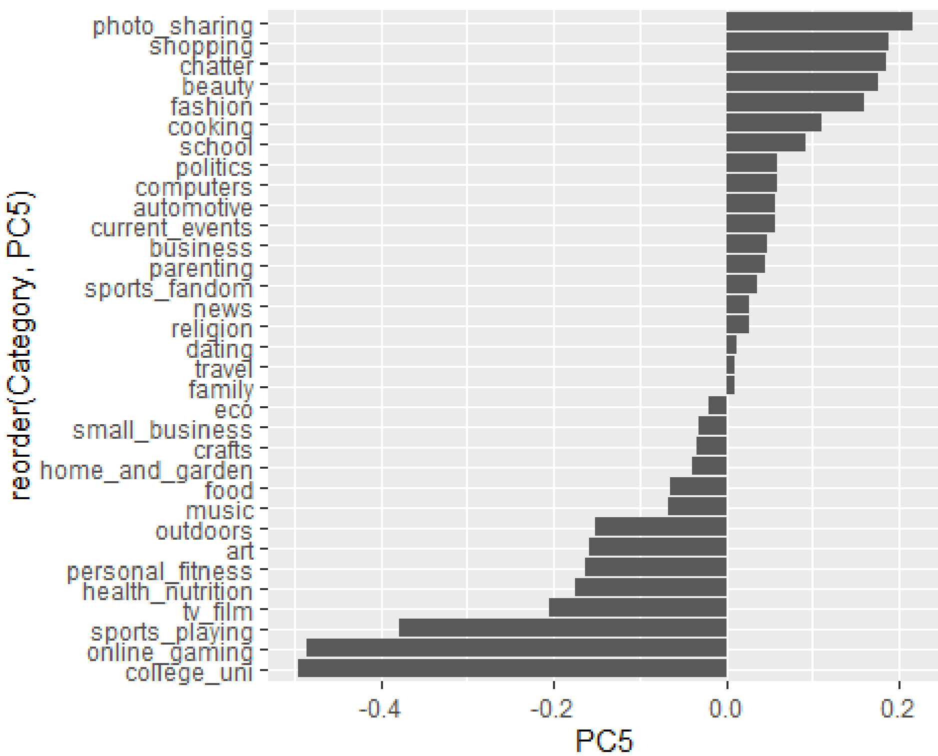

```r
ggplot(loadings) +
  geom_col(aes(x=reorder(Category, PC3), y=PC3)) +
  coord_flip()
```



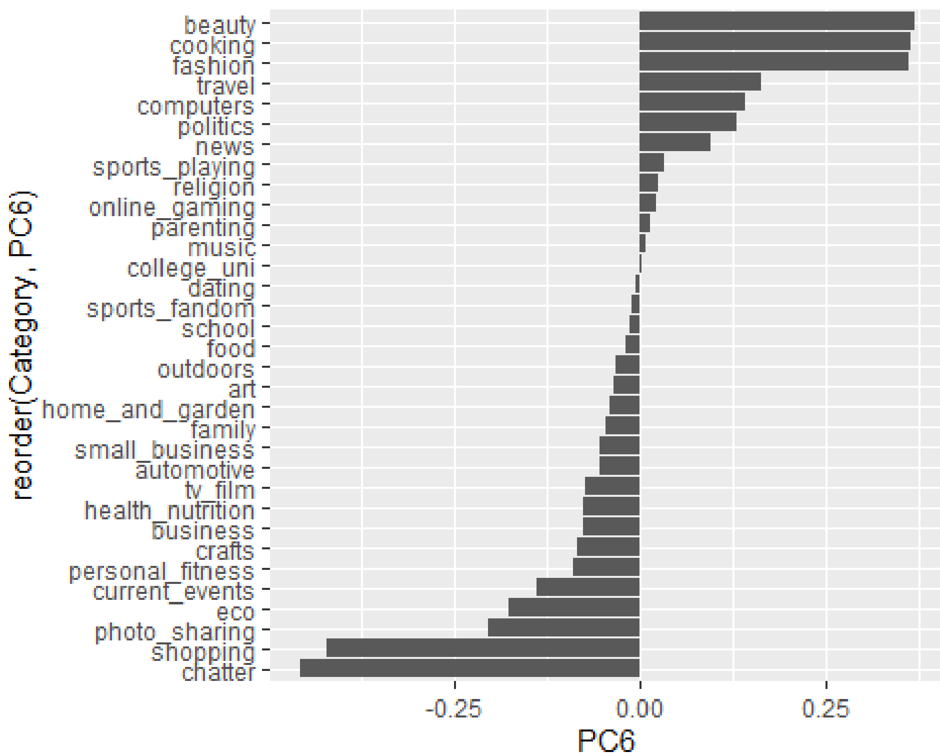

```r
ggplot(loadings) +
  geom_col(aes(x=reorder(Category, PC4), y=PC4)) +
  coord_flip()
```



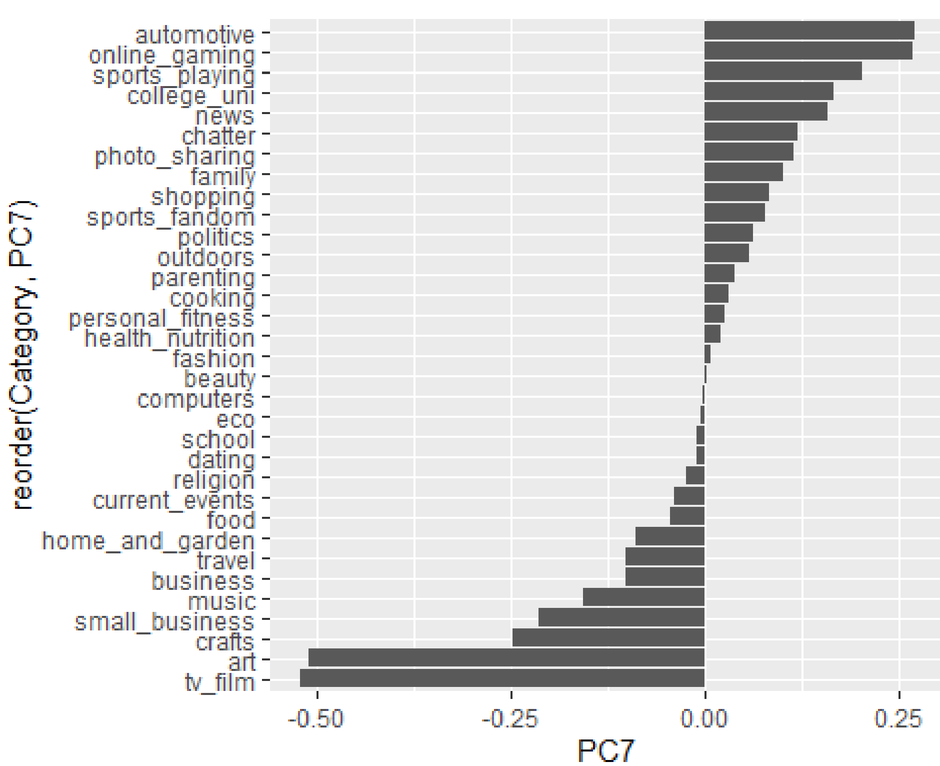
```
ggplot(loadings) +
  geom_col(aes(x=reorder(Category, PC5), y=PC5)) +
  coord_flip()
```



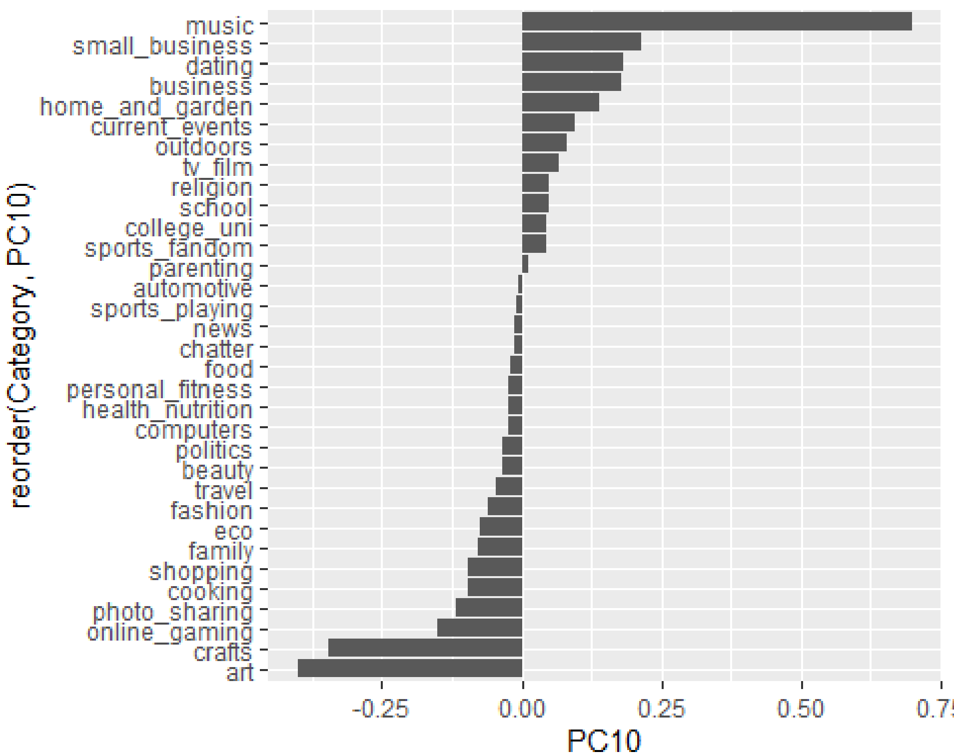

```
ggplot(loadings) +
  geom_col(aes(x=reorder(Category, PC6), y=PC6)) +
  coord_flip()
```



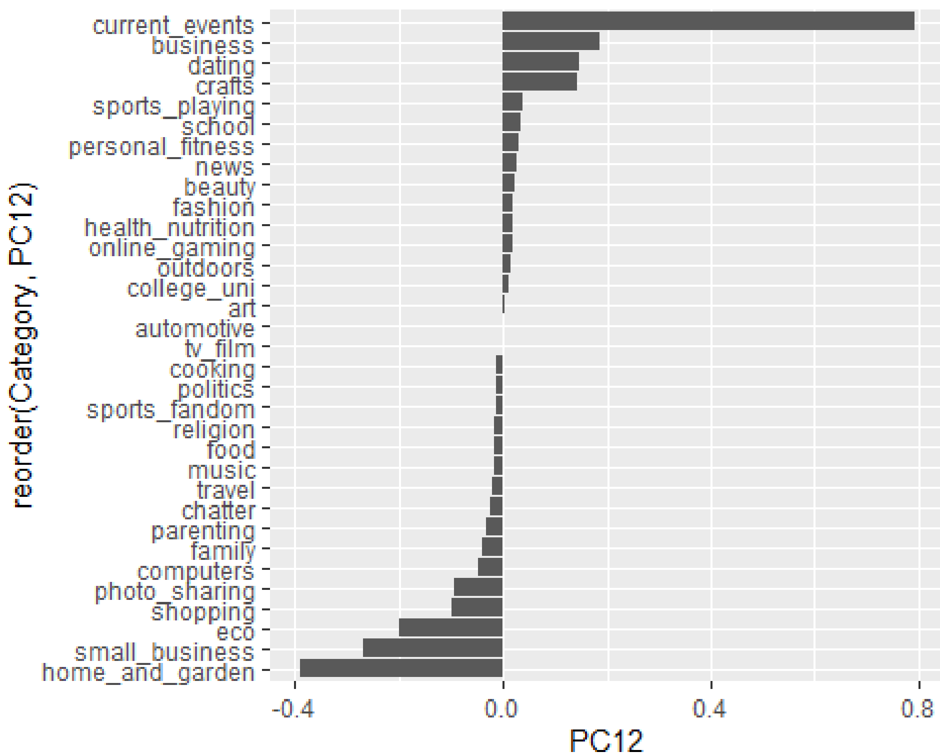

```
ggplot(loadings) +
  geom_col(aes(x=reorder(Category, PC7), y=PC7)) +
  coord_flip()
```



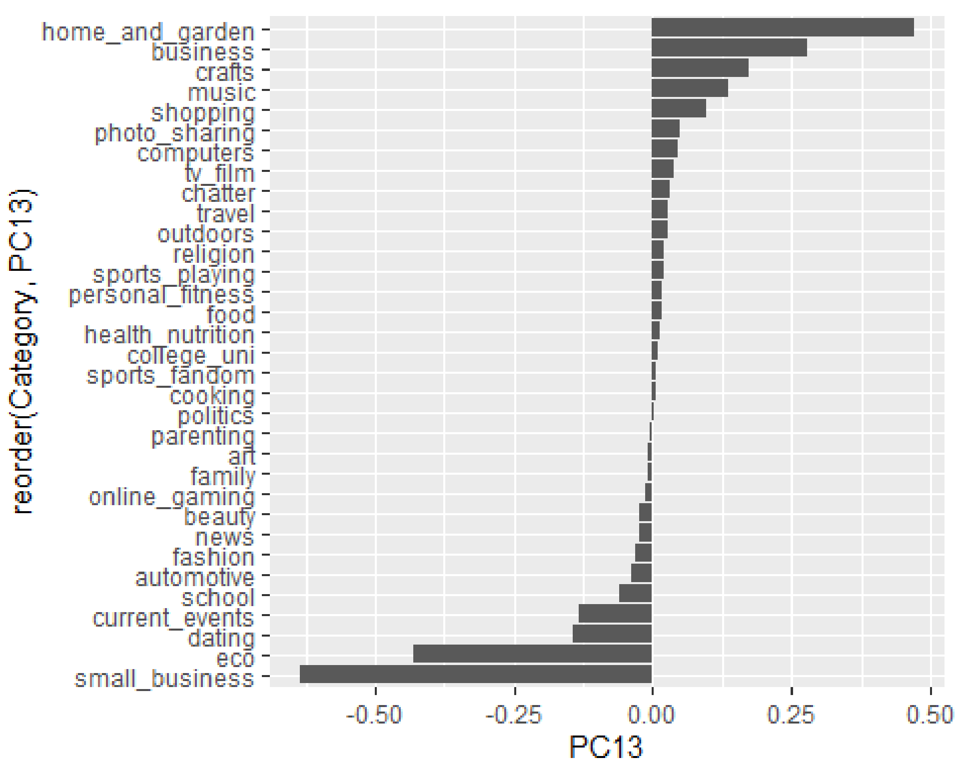
```r
ggplot(loadings) +
  geom_col(aes(x=reorder(Category, PC10), y=PC10)) +
  coord_flip()
```



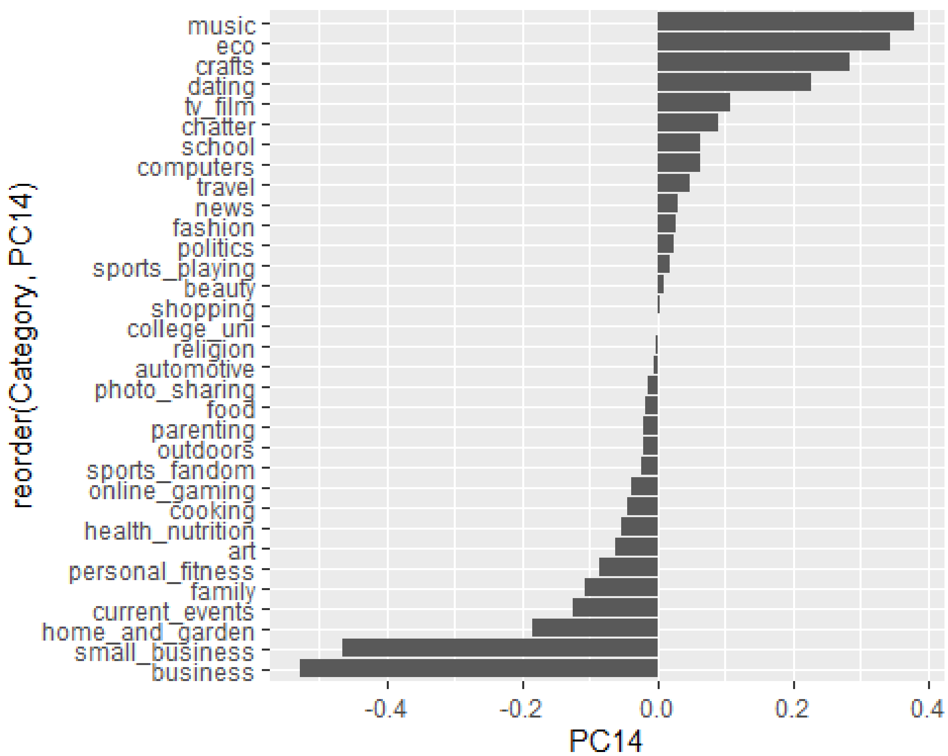

```r
ggplot(loadings) +
  geom_col(aes(x=reorder(Category, PC12), y=PC12)) +
  coord_flip()
```



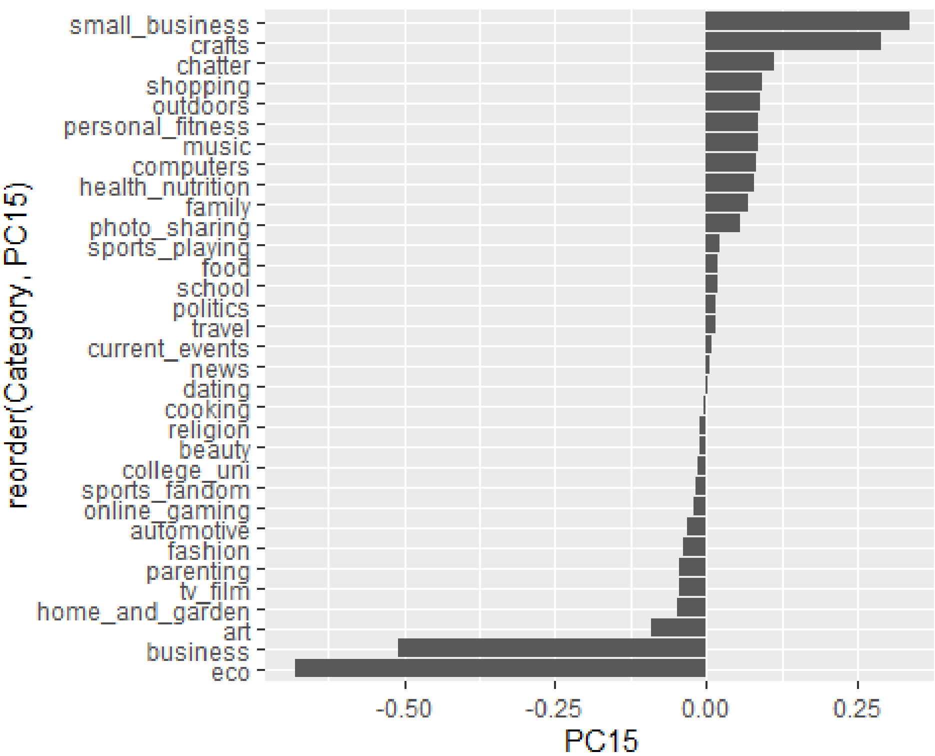

```r
ggplot(loadings) +
  geom_col(aes(x=reorder(Category, PC13), y=PC13)) +
  coord_flip()
```



```r
ggplot(loadings) +
  geom_col(aes(x=reorder(Category, PC14), y=PC14)) +
  coord_flip()
```




```r
ggplot(loadings) +
  geom_col(aes(x=reorder(Category, PC15), y=PC15)) +
  coord_flip()
```



Question 5

The dataset used was the training and test datasets from C50 Reuters. Prior to converting the documents into a corpus, I had to read it in using a readerPlain function. Once this task was completed, I converted all the documents from all the authors into corpus then cleaned the corpus to remove extra white space, punctuation, and numbers. Along with that I also removed any English stop words that were used in the document. I then converted my corpus for each of the authors into a document term matrix. Once I had all 50 document matrices, I removed any words that didn't appear in 95% of all 50 documents which improved the overall sparsity for each of my 50 document term matrices. I then calculated the TF IDF for each of the author's document term matrices. I did this for both the training dataset and the test dataset training. The only slight difference between the procedure for the test dataset was that I removed the words from the test dataset that were not present on the training set before training my model.
Once most of my preprocessing was completed, I continued with training my model. To train my model, I used the TF IDF of every author's document term matrix from the training dataset. I used a decision tree model using the rpart algorithm to train my model. Once I had the model, I then predicted my results using model and the TF IDF from the test dataset. Lastly, I calculated the accuracy. I did not continue to optimize my model since my

out of sample accuracy was 99% which is extremely close to what I would've gotten for the in-sample accuracy.

```
library(tm)
library(tidyverse)
library(slam)
library(proxy)
library(tidytext)
library(dplyr)

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




## apply to all of Simon Cowell's articles
## (probably not THE Simon Cowell: https://twitter.com/simoncowell)
## "globbing" = expanding wild cards in filename paths

list.dirs <- function(path=".", pattern=NULL, all.dirs=FALSE,
                      full.names=FALSE, ignore.case=FALSE) {
  # use full.names=TRUE to pass to file.info
  all <- list.files(path, pattern, all.dirs,
                    full.names=TRUE, recursive=FALSE, ignore.case)
  dirs <- all[file.info(all)$isdir]
  # determine whether to return full names or just dir names
  if(isTRUE(full.names))
    return(dirs)
  else
    return(basename(dirs))
}

list_folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0train/')
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
p <- function(..., sep='') {
  paste(..., sep=sep, collapse=sep)
}
for (i in list_folders){
  folder = i
  assign(i,i)
}
file_list =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',SimonCowell,'/*.txt'))
```

```r
simon = lapply(file_list, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(simon) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(simon))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_simon = DocumentTermMatrix(my_documents)



## Finally, let's drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
```

```
DTM_simon = removeSparseTerms(DTM_simon, 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_simon = weightTfIdf(DTM_simon)
simon_DF <- as.matrix(tfidf_simon)
DTM_simon2 <- as.data.frame(as.matrix(DTM_simon), stringsAsFactors=False)


#########################################################################
#Next Author

#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
filename =
p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C50train/'
,AaronPressman,'/*.txt')
file_listA = Sys.glob(filename)

aaron = lapply(file_listA, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesA = file_listA %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist


names(aaron) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(aaron))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
```

```
another one's treasure.
# 2 example built-in sets of stop words

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_aaron = DocumentTermMatrix(my_documents)
## You can inspect its entries...

DTM_aaron = removeSparseTerms(DTM_aaron, 0.95)
tfidf_aaron = weightTfIdf(DTM_aaron)
aaron_DF <- as.matrix(tfidf_aaron)
DTM_aaron2 <- as.data.frame(as.matrix(DTM_aaron), stringsAsFactors=False)


###next Author
list_folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0train/')

#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',AlexanderSmith,'/*.txt'))

alex = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(alex) = mynames

## once you have documents in a vector, you
```

```r
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(alex))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccttrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_alex = DocumentTermMatrix(my_documents)


DTM_alex = removeSparseTerms(DTM_alex, 0.95)

tfidf_alex = weightTfIdf(DTM_alex)
alex_DF <- as.matrix(tfidf_alex)

###next Author
list_folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0train/')

#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',AlanCrosby,'/*.txt'))

alan = lapply(file_listAl, readerPlain)
```

```r
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(alan) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(alan))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_alan = DocumentTermMatrix(my_documents)


DTM_alan = removeSparseTerms(DTM_alan, 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_alan = weightTfIdf(DTM_alan)
```

```
alan_DF <- as.matrix(tfidf_alan)
DTM_alan2 <- as.data.frame(as.matrix(DTM_alan), stringsAsFactors=False)

# the proxy library has a built-in function to calculate cosine distance
# define the cosine distance matrix for our DTM using this function

####
# Dimensionality reduction
####

# Now PCA on term frequencies
###Next Author
list_folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0train/')

#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',BenjaminKangLim,'/*.txt'))

ben = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
names(ben) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(ben))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
```

```
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersсctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_ben = DocumentTermMatrix(my_documents)

DTM_ben = removeSparseTerms(DTM_ben, 0.95)


tfidf_ben = weightTfIdf(DTM_ben)
ben_DF <- as.matrix(tfidf_ben)
DTM_ben2 <- as.data.frame(as.matrix(DTM_ben), stringsAsFactors=False)

###Next Author
list_folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0train/')

#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',BenjaminKangLim,'/*.txt'))

ben = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
```

```r
names(ben) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(ben))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_ben = DocumentTermMatrix(my_documents)

DTM_ben = removeSparseTerms(DTM_ben, 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_ben = weightTfIdf(DTM_ben)
ben_DF <- as.matrix(tfidf_ben)
DTM_ben2 <- as.data.frame(as.matrix(DTM_ben), stringsAsFactors=False)


##Next Author

#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',BradDorfman,'/*.txt'))
```

```
brad = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles

names(brad) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(brad))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_brad = DocumentTermMatrix(my_documents)

## Finally, let's drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to learn if a term occured once.
```

```r
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
DTM_brad = removeSparseTerms(DTM_brad, 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_brad = weightTfIdf(DTM_brad)
brad_DF <- as.matrix(tfidf_brad)
DTM_brad2 <- as.data.frame(as.matrix(DTM_brad), stringsAsFactors=False)



### Next Author

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',DarrenSchuettler,'/*.txt'))

darren = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(darren) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(darren))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words

# let's just use the "basic English" stop words
```

```r
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_darren = DocumentTermMatrix(my_documents)

DTM_darren = removeSparseTerms(DTM_darren, 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_darren = weightTfIdf(DTM_darren)
darren_DF <- as.matrix(tfidf_darren)
DTM_darren2 <- as.data.frame(as.matrix(DTM_darren), stringsAsFactors=False)

##Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',DavidLawder,'/*.txt'))

david = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(david) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(david))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
```

```
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_david = DocumentTermMatrix(my_documents)

DTM_david = removeSparseTerms(DTM_david, 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_david = weightTfIdf(DTM_david)
david_DF <- as.matrix(tfidf_david)
DTM_david2 <- as.data.frame(as.matrix(DTM_david), stringsAsFactors=False)

### Next Author
file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',EdnaFernandes,'/*.txt'))

edna  = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(edna ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(edna ))
```

```r
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))

## create a doc-term-matrix from the corpus
DTM_edna  = DocumentTermMatrix(my_documents)

DTM_edna  = removeSparseTerms(DTM_edna , 0.95)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_edna  = weightTfIdf(DTM_edna )
edna_DF <- as.matrix(tfidf_edna )
DTM_edna2 <- as.data.frame(as.matrix(DTM_edna ), stringsAsFactors=False)

###Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',EricAuchard,'/*.txt'))

eric  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
```

```
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(eric ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(eric ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_eric  = DocumentTermMatrix(my_documents)

DTM_eric  = removeSparseTerms(DTM_eric , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_eric  = weightTfIdf(DTM_eric )
eric_DF <- as.matrix(tfidf_eric )
DTM_eric2 <- as.data.frame(as.matrix(DTM_eric ), stringsAsFactors=False)

###Next Author

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',FumikoFujisaki,'/*.txt'))
```

```r
fumiko  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist


names(fumiko ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(fumiko ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))

## create a doc-term-matrix from the corpus
DTM_fumiko  = DocumentTermMatrix(my_documents)

DTM_fumiko  = removeSparseTerms(DTM_fumiko , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_fumiko  = weightTfIdf(DTM_fumiko )
fumiko_DF <- as.matrix(tfidf_fumiko )
```

```
DTM_fumiko2 <- as.data.frame(as.matrix(DTM_fumiko ), stringsAsFactors=False)
### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',GrahamEarnshaw,'/*.txt'))

graham  = lapply(file_listAl, readerPlain)



# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist


names(graham ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(graham ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
```

```
t"))

## create a doc-term-matrix from the corpus
DTM_graham  = DocumentTermMatrix(my_documents)

DTM_graham  = removeSparseTerms(DTM_graham , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_graham  = weightTfIdf(DTM_graham )
graham_DF <- as.matrix(tfidf_graham )
DTM_graham2 <- as.data.frame(as.matrix(DTM_graham ), stringsAsFactors=False)

### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',HeatherScoffield,'/*.txt'))

heather  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(heather ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(heather ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space
```

```
## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_heather  = DocumentTermMatrix(my_documents)

DTM_heather  = removeSparseTerms(DTM_heather , 0.95)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_heather  = weightTfIdf(DTM_heather )
heather_DF <- as.matrix(tfidf_heather )
DTM_heather2 <- as.data.frame(as.matrix(DTM_heather ),
stringsAsFactors=False)


###Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JanLopatka,'/*.txt'))

jan  = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(jan ) = mynames

## once you have documents in a vector, you
```

```r
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jan ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_jan  = DocumentTermMatrix(my_documents)

DTM_jan  = removeSparseTerms(DTM_jan , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_jan  = weightTfIdf(DTM_jan )
jan_DF <- as.matrix(tfidf_jan )
DTM_jan2 <- as.data.frame(as.matrix(DTM_jan ), stringsAsFactors=False)

### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JaneMacartney,'/*.txt'))

jane  = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
```

```
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(jane ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jane ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_jane  = DocumentTermMatrix(my_documents)

DTM_jane  = removeSparseTerms(DTM_jane , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_jane  = weightTfIdf(DTM_jane )
jane_DF <- as.matrix(tfidf_jane )
DTM_jane2 <- as.data.frame(as.matrix(DTM_jane ), stringsAsFactors=False)

###Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JimGilchrist,'/*.txt'))
```

```r
jim  = lapply(file_listAl, readerPlain)



# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist


names(jim ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jim ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_jim  = DocumentTermMatrix(my_documents)

DTM_jim  = removeSparseTerms(DTM_jim , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_jim  = weightTfIdf(DTM_jim )
jim_DF <- as.matrix(tfidf_jim )
DTM_jim2 <- as.data.frame(as.matrix(DTM_jim ), stringsAsFactors=False)
```

```r
### Next Author

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JoWinterbottom,'/*.txt'))

jo  = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(jo ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jo ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space


my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))


## create a doc-term-matrix from the corpus
DTM_jo  = DocumentTermMatrix(my_documents)

DTM_jo  = removeSparseTerms(DTM_jo , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_jo  = weightTfIdf(DTM_jo )
jo_DF <- as.matrix(tfidf_jo )
DTM_jo2 <- as.data.frame(as.matrix(DTM_jo ), stringsAsFactors=False)
```

```
## Next Author

#file_list + list_folders[i] = 
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl = 
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JoeOrtiz,'/*.txt'))

joe  = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
names(joe ) = mynames

## once you have documents in a vector, you 
## create a text mining 'corpus' with: 
documents_raw = Corpus(VectorSource(joe ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything 
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess 
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords), 
stopwords("en"))


## create a doc-term-matrix from the corpus
DTM_joe  = DocumentTermMatrix(my_documents)

DTM_joe  = removeSparseTerms(DTM_joe , 0.95)
```

```r
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_joe  = weightTfIdf(DTM_joe )
joe_DF <- as.matrix(tfidf_joe )
DTM_joe2 <- as.data.frame(as.matrix(DTM_joe ), stringsAsFactors=False)

### Next Author

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JohnMastrini,'/*.txt'))

john  = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(john ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(john ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersсctrainsimoncowellnewsmltx
t"))
```

```r
## create a doc-term-matrix from the corpus
DTM_john  = DocumentTermMatrix(my_documents)

DTM_john  = removeSparseTerms(DTM_john , 0.95)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_john  = weightTfIdf(DTM_john )
john_DF <- as.matrix(tfidf_john )
DTM_john2 <- as.data.frame(as.matrix(DTM_john ), stringsAsFactors=False)

## Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JonathanBirt,'/*.txt'))

jonathan  = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(jonathan ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jonathan))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space
```

```
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_jonathan  = DocumentTermMatrix(my_documents)

DTM_jonathan  = removeSparseTerms(DTM_jonathan , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_jonathan  = weightTfIdf(DTM_jonathan )
jonathan_DF <- as.matrix(tfidf_jonathan )
DTM_jonathan2 <- as.data.frame(as.matrix(DTM_jonathan ),
stringsAsFactors=False)

### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KarlPenhaul,'/*.txt'))

karl  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(karl ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(karl))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
```

```
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))

## create a doc-term-matrix from the corpus
DTM_karl  = DocumentTermMatrix(my_documents)

DTM_karl  = removeSparseTerms(DTM_karl , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_karl  = weightTfIdf(DTM_karl )
karl_DF <- as.matrix(tfidf_karl )
DTM_karl2 <- as.data.frame(as.matrix(DTM_karl ), stringsAsFactors=False)


## Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KeithWeir,'/*.txt'))

keith  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(keith ) = mynames
```

```
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(keith))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_keith  = DocumentTermMatrix(my_documents)

DTM_keith  = removeSparseTerms(DTM_keith , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_keith  = weightTfIdf(DTM_keith )
keith_DF <- as.matrix(tfidf_keith )
DTM_keith2 <- as.data.frame(as.matrix(DTM_keith ), stringsAsFactors=False)


### Next Author

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KevinDrawbaugh,'/*.txt'))

kevind  = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
```

```
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(kevind ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(kevind))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_kevind  = DocumentTermMatrix(my_documents)

## ...find words with greater than a min count...
## ...or fiDTM_kevind ds whose count correlates with a specified word.
# the top entries here look like they go with "genetic"
DTM_kevind  = removeSparseTerms(DTM_kevind , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_kevind  = weightTfIdf(DTM_kevind )
kevind_DF <- as.matrix(tfidf_kevind )
DTM_kevind2 <- as.data.frame(as.matrix(DTM_kevind ), stringsAsFactors=False)


### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
```

```r
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KevinMorrison,'/*.txt'))

kevinm  = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(kevinm ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(kevinm))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_kevinm  = DocumentTermMatrix(my_documents)

DTM_kevinm  = removeSparseTerms(DTM_kevinm , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_kevinm  = weightTfIdf(DTM_kevinm )
kevinm_DF <- as.matrix(tfidf_kevinm )
DTM_kevinm2 <- as.data.frame(as.matrix(DTM_kevinm ), stringsAsFactors=False)
```

```
####
# Compare /cluster documents
####

### Next Author


file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KirstinRidley,'/*.txt'))

kirstin  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(kirstin ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(kirstin))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_kirstin  = DocumentTermMatrix(my_documents)
```

```r
DTM_kirstin  = removeSparseTerms(DTM_kirstin , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_kirstin  = weightTfIdf(DTM_kirstin )
kirstin_DF <- as.matrix(tfidf_kirstin )
DTM_kirstin2 <- as.data.frame(as.matrix(DTM_kirstin ),
stringsAsFactors=False)

library(tm)
library(tidyverse)
library(slam)
library(proxy)
library(tidytext)
library(dplyr)

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



## apply to all of Simon Cowell's articles
## (probably not THE Simon Cowell: https://twitter.com/simoncowell)
## "globbing" = expanding wild cards in filename paths

list.dirs <- function(path=".", pattern=NULL, all.dirs=FALSE,
                      full.names=FALSE, ignore.case=FALSE) {
  # use full.names=TRUE to pass to file.info
  all <- list.files(path, pattern, all.dirs,
                    full.names=TRUE, recursive=FALSE, ignore.case)
  dirs <- all[file.info(all)$isdir]
  # determine whether to return full names or just dir names
  if(isTRUE(full.names))
    return(dirs)
  else
    return(basename(dirs))
}

list_folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0train/')
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
p <- function(..., sep='') {
  paste(..., sep=sep, collapse=sep)
}
```

```r
for (i in list_folders){
  folder = i
  assign(i,i)
}
file_list =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',SimonCowell,'/*.txt'))

simon = lapply(file_list, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(simon) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(simon))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_simon = DocumentTermMatrix(my_documents)
```

```
## Finally, let's drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
DTM_simon = removeSparseTerms(DTM_simon, 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_simon = weightTfIdf(DTM_simon)
simon_DF <- as.matrix(tfidf_simon)
DTM_simon2 <- as.data.frame(as.matrix(DTM_simon), stringsAsFactors=False)


##########################################################################
#Next Author

#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
filename =
p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C50train/'
,AaronPressman,'/*.txt')
file_listA = Sys.glob(filename)

aaron = lapply(file_listA, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesA = file_listA %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist


names(aaron) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(aaron))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
```

```
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_aaron = DocumentTermMatrix(my_documents)
## You can inspect its entries...

DTM_aaron = removeSparseTerms(DTM_aaron, 0.95)
tfidf_aaron = weightTfIdf(DTM_aaron)
aaron_DF <- as.matrix(tfidf_aaron)
DTM_aaron2 <- as.data.frame(as.matrix(DTM_aaron), stringsAsFactors=False)


###next Author
list_folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0train/')

#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',AlexanderSmith,'/*.txt'))

alex = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
```

```
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(alex) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(alex))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_alex = DocumentTermMatrix(my_documents)


DTM_alex = removeSparseTerms(DTM_alex, 0.95)

tfidf_alex = weightTfIdf(DTM_alex)
alex_DF <- as.matrix(tfidf_alex)

###next Author
list_folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0train/')

#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
```

```
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',AlanCrosby,'/*.txt'))

alan = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(alan) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(alan))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccttrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_alan = DocumentTermMatrix(my_documents)
```

```
DTM_alan = removeSparseTerms(DTM_alan, 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_alan = weightTfIdf(DTM_alan)
alan_DF <- as.matrix(tfidf_alan)
DTM_alan2 <- as.data.frame(as.matrix(DTM_alan), stringsAsFactors=False)

# the proxy library has a built-in function to calculate cosine distance
# define the cosine distance matrix for our DTM using this function

####
# Dimensionality reduction
####

# Now PCA on term frequencies
###Next Author
list_folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0train/')

#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',BenjaminKangLim,'/*.txt'))

ben = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
names(ben) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(ben))
```

```
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_ben = DocumentTermMatrix(my_documents)

DTM_ben = removeSparseTerms(DTM_ben, 0.95)


tfidf_ben = weightTfIdf(DTM_ben)
ben_DF <- as.matrix(tfidf_ben)
DTM_ben2 <- as.data.frame(as.matrix(DTM_ben), stringsAsFactors=False)

###Next Author
list_folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0train/')

#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',BenjaminKangLim,'/*.txt'))

ben = lapply(file_listAl, readerPlain)


# Clean up the file names
```

```r
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(ben) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(ben))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_ben = DocumentTermMatrix(my_documents)

DTM_ben = removeSparseTerms(DTM_ben, 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_ben = weightTfIdf(DTM_ben)
ben_DF <- as.matrix(tfidf_ben)
DTM_ben2 <- as.data.frame(as.matrix(DTM_ben), stringsAsFactors=False)


##Next Author
```

```
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',BradDorfman,'/*.txt'))

brad = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles

names(brad) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(brad))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersсctrainsimoncowellnewsmltx
t"))
```

```
## create a doc-term-matrix from the corpus
DTM_brad = DocumentTermMatrix(my_documents)

## Finally, let's drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
DTM_brad = removeSparseTerms(DTM_brad, 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_brad = weightTfIdf(DTM_brad)
brad_DF <- as.matrix(tfidf_brad)
DTM_brad2 <- as.data.frame(as.matrix(DTM_brad), stringsAsFactors=False)



### Next Author

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',DarrenSchuettler,'/*.txt'))

darren = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(darren) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(darren))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
```

```
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccttrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_darren = DocumentTermMatrix(my_documents)

DTM_darren = removeSparseTerms(DTM_darren, 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_darren = weightTfIdf(DTM_darren)
darren_DF <- as.matrix(tfidf_darren)
DTM_darren2 <- as.data.frame(as.matrix(DTM_darren), stringsAsFactors=False)

##Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',DavidLawder,'/*.txt'))

david = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(david) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
```

```
documents_raw = Corpus(VectorSource(david))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersсctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_david = DocumentTermMatrix(my_documents)

DTM_david = removeSparseTerms(DTM_david, 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_david = weightTfIdf(DTM_david)
david_DF <- as.matrix(tfidf_david)
DTM_david2 <- as.data.frame(as.matrix(DTM_david), stringsAsFactors=False)

### Next Author
file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',EdnaFernandes,'/*.txt'))

edna  = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
```

```
  unlist

names(edna ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(edna ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))

## create a doc-term-matrix from the corpus
DTM_edna  = DocumentTermMatrix(my_documents)

DTM_edna  = removeSparseTerms(DTM_edna , 0.95)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_edna  = weightTfIdf(DTM_edna )
edna_DF <- as.matrix(tfidf_edna )
DTM_edna2 <- as.data.frame(as.matrix(DTM_edna ), stringsAsFactors=False)

###Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',EricAuchard,'/*.txt'))
```

```r
eric  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(eric ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(eric ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_eric  = DocumentTermMatrix(my_documents)

DTM_eric  = removeSparseTerms(DTM_eric , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_eric  = weightTfIdf(DTM_eric )
eric_DF <- as.matrix(tfidf_eric )
```

```
DTM_eric2 <- as.data.frame(as.matrix(DTM_eric ), stringsAsFactors=False)

###Next Author

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',FumikoFujisaki,'/*.txt'))

fumiko  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist


names(fumiko ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(fumiko ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))

## create a doc-term-matrix from the corpus
DTM_fumiko  = DocumentTermMatrix(my_documents)
```

```r
DTM_fumiko  = removeSparseTerms(DTM_fumiko , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_fumiko  = weightTfIdf(DTM_fumiko )
fumiko_DF <- as.matrix(tfidf_fumiko )
DTM_fumiko2 <- as.data.frame(as.matrix(DTM_fumiko ), stringsAsFactors=False)
### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',GrahamEarnshaw,'/*.txt'))

graham  = lapply(file_listAl, readerPlain)



# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist


names(graham ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(graham ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
```

```
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))

## create a doc-term-matrix from the corpus
DTM_graham  = DocumentTermMatrix(my_documents)

DTM_graham  = removeSparseTerms(DTM_graham , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_graham  = weightTfIdf(DTM_graham )
graham_DF <- as.matrix(tfidf_graham )
DTM_graham2 <- as.data.frame(as.matrix(DTM_graham ), stringsAsFactors=False)

### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',HeatherScoffield,'/*.txt'))

heather  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(heather ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(heather ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
```

```
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_heather  = DocumentTermMatrix(my_documents)

DTM_heather  = removeSparseTerms(DTM_heather , 0.95)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_heather  = weightTfIdf(DTM_heather )
heather_DF <- as.matrix(tfidf_heather )
DTM_heather2 <- as.data.frame(as.matrix(DTM_heather ),
stringsAsFactors=False)


###Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JanLopatka,'/*.txt'))

jan  = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
```

```
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(jan ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jan ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccTrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_jan  = DocumentTermMatrix(my_documents)

DTM_jan  = removeSparseTerms(DTM_jan , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_jan  = weightTfIdf(DTM_jan )
jan_DF <- as.matrix(tfidf_jan )
DTM_jan2 <- as.data.frame(as.matrix(DTM_jan ), stringsAsFactors=False)

### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JaneMacartney,'/*.txt'))
```

```
jane  = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(jane ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jane ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_jane  = DocumentTermMatrix(my_documents)

DTM_jane  = removeSparseTerms(DTM_jane , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_jane  = weightTfIdf(DTM_jane )
jane_DF <- as.matrix(tfidf_jane )
DTM_jane2 <- as.data.frame(as.matrix(DTM_jane ), stringsAsFactors=False)

###Next Author
#file_list + list_folders[i] =
```

```
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JimGilchrist,'/*.txt'))

jim  = lapply(file_listAl, readerPlain)



# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist


names(jim ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jim ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_jim  = DocumentTermMatrix(my_documents)

DTM_jim  = removeSparseTerms(DTM_jim , 0.95)
```

```
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_jim  = weightTfIdf(DTM_jim )
jim_DF <- as.matrix(tfidf_jim )
DTM_jim2 <- as.data.frame(as.matrix(DTM_jim ), stringsAsFactors=False)

### Next Author

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JoWinterbottom,'/*.txt'))

jo  = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(jo ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jo ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space


my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))


## create a doc-term-matrix from the corpus
DTM_jo  = DocumentTermMatrix(my_documents)
```

```
DTM_jo  = removeSparseTerms(DTM_jo , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_jo  = weightTfIdf(DTM_jo )
jo_DF <- as.matrix(tfidf_jo )
DTM_jo2 <- as.data.frame(as.matrix(DTM_jo ), stringsAsFactors=False)

## Next Author

#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JoeOrtiz,'/*.txt'))

joe  = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
names(joe ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(joe ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
```

```
stopwords("en"))


## create a doc-term-matrix from the corpus
DTM_joe  = DocumentTermMatrix(my_documents)

DTM_joe  = removeSparseTerms(DTM_joe , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_joe  = weightTfIdf(DTM_joe )
joe_DF <- as.matrix(tfidf_joe )
DTM_joe2 <- as.data.frame(as.matrix(DTM_joe ), stringsAsFactors=False)

### Next Author

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JohnMastrini,'/*.txt'))

john  = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(john ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(john ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%            # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%       # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%   # remove punctuation
  tm_map(content_transformer(stripWhitespace))         # remove excess
white-space

# let's just use the "basic English" stop words
```

```r
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_john  = DocumentTermMatrix(my_documents)

DTM_john  = removeSparseTerms(DTM_john , 0.95)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_john  = weightTfIdf(DTM_john )
john_DF <- as.matrix(tfidf_john )
DTM_john2 <- as.data.frame(as.matrix(DTM_john ), stringsAsFactors=False)

## Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JonathanBirt,'/*.txt'))

jonathan  = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(jonathan ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jonathan))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
```

```
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_jonathan  = DocumentTermMatrix(my_documents)

DTM_jonathan  = removeSparseTerms(DTM_jonathan , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_jonathan  = weightTfIdf(DTM_jonathan )
jonathan_DF <- as.matrix(tfidf_jonathan )
DTM_jonathan2 <- as.data.frame(as.matrix(DTM_jonathan ),
stringsAsFactors=False)

### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KarlPenhaul,'/*.txt'))

karl  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(karl ) = mynames
```

```r
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(karl))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything 
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess 
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords), 
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords, 
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))

## create a doc-term-matrix from the corpus
DTM_karl  = DocumentTermMatrix(my_documents)

DTM_karl  = removeSparseTerms(DTM_karl , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_karl  = weightTfIdf(DTM_karl )
karl_DF <- as.matrix(tfidf_karl )
DTM_karl2 <- as.data.frame(as.matrix(DTM_karl ), stringsAsFactors=False)


## Next Author
#file_list + list_folders[i] = 
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl = 
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KeithWeir,'/*.txt'))

keith  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
```

```
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(keith ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(keith))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_keith  = DocumentTermMatrix(my_documents)

DTM_keith  = removeSparseTerms(DTM_keith , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_keith  = weightTfIdf(DTM_keith )
keith_DF <- as.matrix(tfidf_keith )
DTM_keith2 <- as.data.frame(as.matrix(DTM_keith ), stringsAsFactors=False)


### Next Author

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KevinDrawbaugh,'/*.txt'))

kevind  = lapply(file_listAl, readerPlain)
```

```r
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(kevind ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(kevind))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_kevind  = DocumentTermMatrix(my_documents)

## ...find words with greater than a min count...
## ...or fiDTM_kevind ds whose count correlates with a specified word.
# the top entries here look like they go with "genetic"
DTM_kevind  = removeSparseTerms(DTM_kevind , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_kevind  = weightTfIdf(DTM_kevind )
kevind_DF <- as.matrix(tfidf_kevind )
DTM_kevind2 <- as.data.frame(as.matrix(DTM_kevind ), stringsAsFactors=False)
```

```r
### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KevinMorrison,'/*.txt'))

kevinm  = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(kevinm ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(kevinm))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_kevinm  = DocumentTermMatrix(my_documents)
```

```r
DTM_kevinm  = removeSparseTerms(DTM_kevinm , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_kevinm  = weightTfIdf(DTM_kevinm )
kevinm_DF <- as.matrix(tfidf_kevinm )
DTM_kevinm2 <- as.data.frame(as.matrix(DTM_kevinm ), stringsAsFactors=False)
####
# Compare /cluster documents
####

### Next Author


file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KirstinRidley,'/*.txt'))

kirstin  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(kirstin ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(kirstin))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
```

```
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_kirstin  = DocumentTermMatrix(my_documents)

DTM_kirstin  = removeSparseTerms(DTM_kirstin , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_kirstin  = weightTfIdf(DTM_kirstin )
kirstin_DF <- as.matrix(tfidf_kirstin )
DTM_kirstin2 <- as.data.frame(as.matrix(DTM_kirstin ),
stringsAsFactors=False)

library(tm)
library(tidyverse)
library(slam)
library(proxy)
library(tidytext)
library(dplyr)

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



## apply to all of Simon Cowell's articles
## (probably not THE Simon Cowell: https://twitter.com/simoncowell)
## "globbing" = expanding wild cards in filename paths

list.dirs <- function(path=".", pattern=NULL, all.dirs=FALSE,
                      full.names=FALSE, ignore.case=FALSE) {
  # use full.names=TRUE to pass to file.info
  all <- list.files(path, pattern, all.dirs,
                    full.names=TRUE, recursive=FALSE, ignore.case)
  dirs <- all[file.info(all)$isdir]
  # determine whether to return full names or just dir names
  if(isTRUE(full.names))
    return(dirs)
  else
    return(basename(dirs))
}

list_folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
```

```
0train/')
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
p <- function(..., sep='') {
  paste(..., sep=sep, collapse=sep)
}
for (i in list_folders){
  folder = i
  assign(i,i)
}
file_list =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',SimonCowell,'/*.txt'))

simon = lapply(file_list, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(simon) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(simon))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
```

```
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_simon = DocumentTermMatrix(my_documents)



## Finally, let's drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
DTM_simon = removeSparseTerms(DTM_simon, 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_simon = weightTfIdf(DTM_simon)
simon_DF <- as.matrix(tfidf_simon)
DTM_simon2 <- as.data.frame(as.matrix(DTM_simon), stringsAsFactors=False)


######################################################################
#Next Author

#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
filename =
p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C50train/'
,AaronPressman,'/*.txt')
file_listA = Sys.glob(filename)

aaron = lapply(file_listA, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesA = file_listA %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist


names(aaron) = mynames

## once you have documents in a vector, you
```

```
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(aaron))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_aaron = DocumentTermMatrix(my_documents)
## You can inspect its entries...

DTM_aaron = removeSparseTerms(DTM_aaron, 0.95)
tfidf_aaron = weightTfIdf(DTM_aaron)
aaron_DF <- as.matrix(tfidf_aaron)
DTM_aaron2 <- as.data.frame(as.matrix(DTM_aaron), stringsAsFactors=False)


###next Author
list_folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0train/')

#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',AlexanderSmith,'/*.txt'))

alex = lapply(file_listAl, readerPlain)
```

```
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(alex) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(alex))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_alex = DocumentTermMatrix(my_documents)


DTM_alex = removeSparseTerms(DTM_alex, 0.95)

tfidf_alex = weightTfIdf(DTM_alex)
alex_DF <- as.matrix(tfidf_alex)
```

```
###next Author
list_folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0train/')

#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',AlanCrosby,'/*.txt'))

alan = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(alan) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(alan))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
```

```
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_alan = DocumentTermMatrix(my_documents)


DTM_alan = removeSparseTerms(DTM_alan, 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_alan = weightTfIdf(DTM_alan)
alan_DF <- as.matrix(tfidf_alan)
DTM_alan2 <- as.data.frame(as.matrix(DTM_alan), stringsAsFactors=False)

# the proxy library has a built-in function to calculate cosine distance
# define the cosine distance matrix for our DTM using this function

####
# Dimensionality reduction
####

# Now PCA on term frequencies
###Next Author
list_folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0train/')

#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',BenjaminKangLim,'/*.txt'))

ben = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
```

```
# Rename the articles
names(ben) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(ben))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_ben = DocumentTermMatrix(my_documents)

DTM_ben = removeSparseTerms(DTM_ben, 0.95)


tfidf_ben = weightTfIdf(DTM_ben)
ben_DF <- as.matrix(tfidf_ben)
DTM_ben2 <- as.data.frame(as.matrix(DTM_ben), stringsAsFactors=False)

###Next Author
list_folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0train/')

#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
```

```
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',BenjaminKangLim,'/*.txt'))

ben = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(ben) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(ben))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_ben = DocumentTermMatrix(my_documents)

DTM_ben = removeSparseTerms(DTM_ben, 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
```

```r
tfidf_ben = weightTfIdf(DTM_ben)
ben_DF <- as.matrix(tfidf_ben)
DTM_ben2 <- as.data.frame(as.matrix(DTM_ben), stringsAsFactors=False)


##Next Author

#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',BradDorfman,'/*.txt'))

brad = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles

names(brad) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(brad))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%            # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
```

```r
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_brad = DocumentTermMatrix(my_documents)

## Finally, let's drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
DTM_brad = removeSparseTerms(DTM_brad, 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_brad = weightTfIdf(DTM_brad)
brad_DF <- as.matrix(tfidf_brad)
DTM_brad2 <- as.data.frame(as.matrix(DTM_brad), stringsAsFactors=False)


### Next Author

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',DarrenSchuettler,'/*.txt'))

darren = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(darren) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(darren))

## Some pre-processing/tokenization steps.
```

```
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_darren = DocumentTermMatrix(my_documents)

DTM_darren = removeSparseTerms(DTM_darren, 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_darren = weightTfIdf(DTM_darren)
darren_DF <- as.matrix(tfidf_darren)
DTM_darren2 <- as.data.frame(as.matrix(DTM_darren), stringsAsFactors=False)

##Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',DavidLawder,'/*.txt'))

david = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
```

```
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(david) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(david))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_david = DocumentTermMatrix(my_documents)

DTM_david = removeSparseTerms(DTM_david, 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_david = weightTfIdf(DTM_david)
david_DF <- as.matrix(tfidf_david)
DTM_david2 <- as.data.frame(as.matrix(DTM_david), stringsAsFactors=False)

### Next Author
file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',EdnaFernandes,'/*.txt'))

edna  = lapply(file_listAl, readerPlain)
```

```r
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(edna ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(edna ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))

## create a doc-term-matrix from the corpus
DTM_edna  = DocumentTermMatrix(my_documents)

DTM_edna  = removeSparseTerms(DTM_edna , 0.95)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_edna  = weightTfIdf(DTM_edna )
edna_DF <- as.matrix(tfidf_edna )
DTM_edna2 <- as.data.frame(as.matrix(DTM_edna ), stringsAsFactors=False)

###Next Author
```

```r
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',EricAuchard,'/*.txt'))

eric  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(eric ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(eric ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersсctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_eric  = DocumentTermMatrix(my_documents)
```

```
DTM_eric  = removeSparseTerms(DTM_eric , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_eric  = weightTfIdf(DTM_eric )
eric_DF <- as.matrix(tfidf_eric )
DTM_eric2 <- as.data.frame(as.matrix(DTM_eric ), stringsAsFactors=False)

###Next Author

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',FumikoFujisaki,'/*.txt'))

fumiko  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist


names(fumiko ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(fumiko ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
```

```
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersсctrainsimoncowellnewsmltx
t"))

## create a doc-term-matrix from the corpus
DTM_fumiko  = DocumentTermMatrix(my_documents)

DTM_fumiko  = removeSparseTerms(DTM_fumiko , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_fumiko  = weightTfIdf(DTM_fumiko )
fumiko_DF <- as.matrix(tfidf_fumiko )
DTM_fumiko2 <- as.data.frame(as.matrix(DTM_fumiko ), stringsAsFactors=False)
### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',GrahamEarnshaw,'/*.txt'))

graham  = lapply(file_listAl, readerPlain)



# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist


names(graham ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(graham ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
```

```
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("usersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))

## create a doc-term-matrix from the corpus
DTM_graham  = DocumentTermMatrix(my_documents)

DTM_graham  = removeSparseTerms(DTM_graham , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_graham  = weightTfIdf(DTM_graham )
graham_DF <- as.matrix(tfidf_graham )
DTM_graham2 <- as.data.frame(as.matrix(DTM_graham ), stringsAsFactors=False)

### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',HeatherScoffield,'/*.txt'))

heather  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(heather ) = mynames
```

```r
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(heather ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccttrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_heather  = DocumentTermMatrix(my_documents)

DTM_heather  = removeSparseTerms(DTM_heather , 0.95)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_heather  = weightTfIdf(DTM_heather )
heather_DF <- as.matrix(tfidf_heather )
DTM_heather2 <- as.data.frame(as.matrix(DTM_heather ),
stringsAsFactors=False)


###Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JanLopatka,'/*.txt'))

jan  = lapply(file_listAl, readerPlain)
```

```r
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(jan ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jan ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_jan  = DocumentTermMatrix(my_documents)

DTM_jan  = removeSparseTerms(DTM_jan , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_jan  = weightTfIdf(DTM_jan )
jan_DF <- as.matrix(tfidf_jan )
DTM_jan2 <- as.data.frame(as.matrix(DTM_jan ), stringsAsFactors=False)

### Next Author
#file_list + list_folders[i] =
```

```
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JaneMacartney,'/*.txt'))

jane  = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(jane ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jane ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccttrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_jane  = DocumentTermMatrix(my_documents)

DTM_jane  = removeSparseTerms(DTM_jane , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
```

```r
# as features in a predictive model
tfidf_jane  = weightTfIdf(DTM_jane )
jane_DF <- as.matrix(tfidf_jane )
DTM_jane2 <- as.data.frame(as.matrix(DTM_jane ), stringsAsFactors=False)

###Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JimGilchrist,'/*.txt'))

jim  = lapply(file_listAl, readerPlain)



# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist


names(jim ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jim ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersctrainsimoncowellnewsmltx
```

```r
t"))


## create a doc-term-matrix from the corpus
DTM_jim  = DocumentTermMatrix(my_documents)

DTM_jim  = removeSparseTerms(DTM_jim , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_jim  = weightTfIdf(DTM_jim )
jim_DF <- as.matrix(tfidf_jim )
DTM_jim2 <- as.data.frame(as.matrix(DTM_jim ), stringsAsFactors=False)

### Next Author

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JoWinterbottom,'/*.txt'))

jo  = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(jo ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jo ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space
```

```r
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))


## create a doc-term-matrix from the corpus
DTM_jo  = DocumentTermMatrix(my_documents)

DTM_jo  = removeSparseTerms(DTM_jo , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_jo  = weightTfIdf(DTM_jo )
jo_DF <- as.matrix(tfidf_jo )
DTM_jo2 <- as.data.frame(as.matrix(DTM_jo ), stringsAsFactors=False)

## Next Author

#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JoeOrtiz,'/*.txt'))

joe  = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
names(joe ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(joe ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
```

```r
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))


## create a doc-term-matrix from the corpus
DTM_joe  = DocumentTermMatrix(my_documents)

DTM_joe  = removeSparseTerms(DTM_joe , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_joe  = weightTfIdf(DTM_joe )
joe_DF <- as.matrix(tfidf_joe )
DTM_joe2 <- as.data.frame(as.matrix(DTM_joe ), stringsAsFactors=False)

### Next Author

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JohnMastrini,'/*.txt'))

john  = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(john ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(john ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
```

```r
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_john  = DocumentTermMatrix(my_documents)

DTM_john  = removeSparseTerms(DTM_john , 0.95)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_john  = weightTfIdf(DTM_john )
john_DF <- as.matrix(tfidf_john )
DTM_john2 <- as.data.frame(as.matrix(DTM_john ), stringsAsFactors=False)

## Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JonathanBirt,'/*.txt'))

jonathan  = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(jonathan ) = mynames
```

```
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jonathan))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersсctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_jonathan  = DocumentTermMatrix(my_documents)

DTM_jonathan  = removeSparseTerms(DTM_jonathan , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_jonathan  = weightTfIdf(DTM_jonathan )
jonathan_DF <- as.matrix(tfidf_jonathan )
DTM_jonathan2 <- as.data.frame(as.matrix(DTM_jonathan ),
stringsAsFactors=False)

### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KarlPenhaul,'/*.txt'))

karl  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
```

```
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(karl ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(karl))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))

## create a doc-term-matrix from the corpus
DTM_karl  = DocumentTermMatrix(my_documents)

DTM_karl  = removeSparseTerms(DTM_karl , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_karl  = weightTfIdf(DTM_karl )
karl_DF <- as.matrix(tfidf_karl )
DTM_karl2 <- as.data.frame(as.matrix(DTM_karl ), stringsAsFactors=False)



## Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KeithWeir,'/*.txt'))
```

```
keith  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(keith ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(keith))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_keith  = DocumentTermMatrix(my_documents)

DTM_keith  = removeSparseTerms(DTM_keith , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_keith  = weightTfIdf(DTM_keith )
keith_DF <- as.matrix(tfidf_keith )
DTM_keith2 <- as.data.frame(as.matrix(DTM_keith ), stringsAsFactors=False)


### Next Author
```

```r
file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KevinDrawbaugh,'/*.txt'))

kevind  = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(kevind ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(kevind))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_kevind  = DocumentTermMatrix(my_documents)

## ...find words with greater than a min count...
## ...or fiDTM_kevind ds whose count correlates with a specified word.
# the top entries here look like they go with "genetic"
DTM_kevind  = removeSparseTerms(DTM_kevind , 0.95)
```

```
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_kevind  = weightTfIdf(DTM_kevind )
kevind_DF <- as.matrix(tfidf_kevind )
DTM_kevind2 <- as.data.frame(as.matrix(DTM_kevind ), stringsAsFactors=False)


### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KevinMorrison,'/*.txt'))

kevinm  = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(kevinm ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(kevinm))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
```

```
c("cusersmachuonedrivedocumentsgithubstadatareutersc50ctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_kevinm  = DocumentTermMatrix(my_documents)

DTM_kevinm  = removeSparseTerms(DTM_kevinm , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_kevinm  = weightTfIdf(DTM_kevinm )
kevinm_DF <- as.matrix(tfidf_kevinm )
DTM_kevinm2 <- as.data.frame(as.matrix(DTM_kevinm ), stringsAsFactors=False)
####
# Compare /cluster documents
####

### Next Author


file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KirstinRidley,'/*.txt'))

kirstin  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(kirstin ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(kirstin))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
```

```
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_kirstin  = DocumentTermMatrix(my_documents)

DTM_kirstin  = removeSparseTerms(DTM_kirstin , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_kirstin  = weightTfIdf(DTM_kirstin )
kirstin_DF <- as.matrix(tfidf_kirstin )
DTM_kirstin2 <- as.data.frame(as.matrix(DTM_kirstin ),
stringsAsFactors=False)


### Next Author

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KouroshKarimkhany,'/*.txt'))

kourosh  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(kourosh ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(kourosh))

## Some pre-processing/tokenization steps.
```

```
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))

## create a doc-term-matrix from the corpus
DTM_kourosh  = DocumentTermMatrix(my_documents)

DTM_kourosh  = removeSparseTerms(DTM_kourosh , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_kourosh  = weightTfIdf(DTM_kourosh )
kourosh_DF <- as.matrix(tfidf_kourosh )
DTM_kourosh2 <- as.data.frame(as.matrix(DTM_kourosh ),
stringsAsFactors=False)
####
# Compare /cluster documents



### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',LydiaZajc,'/*.txt'))

lydia  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
```

```
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(lydia ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(lydia))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_lydia  = DocumentTermMatrix(my_documents)

DTM_lydia  = removeSparseTerms(DTM_lydia , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_lydia  = weightTfIdf(DTM_lydia )
lydia_DF <- as.matrix(tfidf_lydia )
DTM_lydia2 <- as.data.frame(as.matrix(DTM_lydia ), stringsAsFactors=False)
####
# Compare /cluster documents
####



## Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
```

```
file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',`LynneO'Donnell`,'/*.txt'))

lynne  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(lynne ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(lynne))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccttrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_lynne  = DocumentTermMatrix(my_documents)

DTM_lynne  = removeSparseTerms(DTM_lynne , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_lynne  = weightTfIdf(DTM_lynne )
lynne_DF <- as.matrix(tfidf_lynne )
DTM_lynne2 <- as.data.frame(as.matrix(DTM_lynne ), stringsAsFactors=False)
####
```

```r
# Compare /cluster documents
####

#Next Author

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',LynnleyBrowning,'/*.txt'))

lynnley  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(lynnley ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(lynnley))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_lynnley  = DocumentTermMatrix(my_documents)

DTM_lynnley  = removeSparseTerms(DTM_lynnley , 0.95)
```

```r
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_lynnley  = weightTfIdf(DTM_lynnley )
lynnley_DF <- as.matrix(tfidf_lynnley )
DTM_lynnley2 <- as.data.frame(as.matrix(DTM_lynnley ),
stringsAsFactors=False)
####
# Compare /cluster documents
####



### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',MarcelMichelson,'/*.txt'))

marcel  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(marcel ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(marcel))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space
```

```
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))

## create a doc-term-matrix from the corpus
DTM_marcel  = DocumentTermMatrix(my_documents)

## Finally, let's drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
DTM_marcel  = removeSparseTerms(DTM_marcel , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_marcel  = weightTfIdf(DTM_marcel )
marcel_DF <- as.matrix(tfidf_marcel )
DTM_marcel2 <- as.data.frame(as.matrix(DTM_marcel ), stringsAsFactors=False)



### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',MarkBendeich,'/*.txt'))

mark  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(mark ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(mark))
```

```r
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess white-space


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords), stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords, c("cusersmachuonedrivedocumentsgithubstadatareutersctrainsimoncowellnewsmltxt"))


## create a doc-term-matrix from the corpus
DTM_mark  = DocumentTermMatrix(my_documents)

## Finally, let's drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
DTM_mark  = removeSparseTerms(DTM_mark , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_mark  = weightTfIdf(DTM_mark )
mark_DF <- as.matrix(tfidf_mark )
DTM_mark2 <- as.data.frame(as.matrix(DTM_mark ), stringsAsFactors=False)


### Next Question
#file_list + list_folders[i] = Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl = Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C50train/',MartinWolk,'/*.txt'))

martin  = lapply(file_listAl, readerPlain)

# Clean up the file names
```

```
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(martin ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(martin))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_martin  = DocumentTermMatrix(my_documents)
DTM_martin  = removeSparseTerms(DTM_martin , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_martin  = weightTfIdf(DTM_martin )
martin_DF <- as.matrix(tfidf_martin )
DTM_martin2 <- as.data.frame(as.matrix(DTM_martin ), stringsAsFactors=False)

### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
```

```
50train/',MichaelConnor,'/*.txt'))

michael  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(michael ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(michael))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_michael  = DocumentTermMatrix(my_documents)

DTM_michael  = removeSparseTerms(DTM_michael , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_michael  = weightTfIdf(DTM_michael )
michael_DF <- as.matrix(tfidf_michael )
DTM_michael2 <- as.data.frame(as.matrix(DTM_michael ),
stringsAsFactors=False)

### Next Author
```

```r
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',MureDickie,'/*.txt'))

mure  = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(mure ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(mure))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_mure  = DocumentTermMatrix(my_documents)

DTM_mure  = removeSparseTerms(DTM_mure , 0.95)
```

```
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_mure  = weightTfIdf(DTM_mure )
mure_DF <- as.matrix(tfidf_mure )
DTM_mure2 <- as.data.frame(as.matrix(DTM_mure ), stringsAsFactors=False)

### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',NickLouth,'/*.txt'))

nick  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(nick ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(nick))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersctrainsimoncowellnewsmltx
t"))
```

```r
## create a doc-term-matrix from the corpus
DTM_nick  = DocumentTermMatrix(my_documents)

## Finally, let's drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
DTM_nick  = removeSparseTerms(DTM_nick , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_nick  = weightTfIdf(DTM_nick )
nick_DF <- as.matrix(tfidf_nick )
DTM_nick2 <- as.data.frame(as.matrix(DTM_nick ), stringsAsFactors=False)

### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',PatriciaCommins,'/*.txt'))

patricia  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(patricia ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(patricia))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
```

```
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))

## create a doc-term-matrix from the corpus
DTM_patricia  = DocumentTermMatrix(my_documents)

DTM_patricia  = removeSparseTerms(DTM_patricia , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_patricia  = weightTfIdf(DTM_patricia )
patricia_DF <- as.matrix(tfidf_patricia )
DTM_patricia2 <- as.data.frame(as.matrix(DTM_patricia ),
stringsAsFactors=False)

### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',PeterHumphrey,'/*.txt'))

peter  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(peter ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(peter))
```

```
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything 
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess 
white-space


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords), 
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords, 
c("usersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_peter  = DocumentTermMatrix(my_documents)

## Finally, let's drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.  
## Probably a bit stringent here... but only 50 docs!
DTM_peter  = removeSparseTerms(DTM_peter , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_peter  = weightTfIdf(DTM_peter )
peter_DF <- as.matrix(tfidf_peter )
DTM_peter2 <- as.data.frame(as.matrix(DTM_peter ), stringsAsFactors=False)

### Next Author
#file_list + list_folders[i] = 
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl = 
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',PierreTran,'/*.txt'))

pierre  = lapply(file_listAl, readerPlain) 
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
```

```
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(pierre ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(pierre))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_pierre  = DocumentTermMatrix(my_documents)

DTM_pierre  = removeSparseTerms(DTM_pierre , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_pierre  = weightTfIdf(DTM_pierre )
pierre_DF <- as.matrix(tfidf_pierre )
DTM_pierre2 <- as.data.frame(as.matrix(DTM_pierre ), stringsAsFactors=False)

### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',RobinSidel,'/*.txt'))
```

```r
robin  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(robin ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(robin))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccttrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_robin  = DocumentTermMatrix(my_documents)

DTM_robin  = removeSparseTerms(DTM_robin , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_robin  = weightTfIdf(DTM_robin )
robin_DF <- as.matrix(tfidf_robin )
DTM_robin2 <- as.data.frame(as.matrix(DTM_robin ), stringsAsFactors=False)

### Next Author
```

```r
file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',RogerFillion,'/*.txt'))

roger  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(roger ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(roger))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_roger  = DocumentTermMatrix(my_documents)
DTM_roger  = removeSparseTerms(DTM_roger , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_roger  = weightTfIdf(DTM_roger )
roger_DF <- as.matrix(tfidf_roger )
DTM_roger2 <- as.data.frame(as.matrix(DTM_roger ), stringsAsFactors=False)
```

```r
### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',SarahDavison,'/*.txt'))

sarah  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(sarah ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(sarah))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_sarah  = DocumentTermMatrix(my_documents)
```

```r
DTM_sarah  = removeSparseTerms(DTM_sarah , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_sarah  = weightTfIdf(DTM_sarah )
sarah_DF <- as.matrix(tfidf_sarah )
DTM_sarah2 <- as.data.frame(as.matrix(DTM_sarah ), stringsAsFactors=False)


### Next Question
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',ScottHillis,'/*.txt'))

scott  = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(scott ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(scott))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
```

```
c("cusersmachuonedrivedocumentsgithubstadatareutersctrainsimoncowellnewsmltx
t"))

DTM_scott  = DocumentTermMatrix(my_documents)


## Finally, let's drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
DTM_scott  = removeSparseTerms(DTM_scott , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_scott  = weightTfIdf(DTM_scott )
scott_DF <- as.matrix(tfidf_scott )
DTM_scott2 <- as.data.frame(as.matrix(DTM_scott ), stringsAsFactors=False)




### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',TheresePoletti,'/*.txt'))

therese  = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(therese ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(therese))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
```

```r
  tm_map(content_transformer(tolower))  %>%             # make everything 
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess 
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords), 
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords, 
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_therese  = DocumentTermMatrix(my_documents)

DTM_therese  = removeSparseTerms(DTM_therese , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_therese  = weightTfIdf(DTM_therese )
therese_DF <- as.matrix(tfidf_therese )
DTM_therese2 <- as.data.frame(as.matrix(DTM_therese ), 
stringsAsFactors=False)


### Next Author

file_listAl = 
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',TimFarrand,'/*.txt'))

tim  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(tim ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
```

```r
documents_raw = Corpus(VectorSource(tim))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))



## create a doc-term-matrix from the corpus
DTM_tim  = DocumentTermMatrix(my_documents)



DTM_tim  = removeSparseTerms(DTM_tim , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_tim  = weightTfIdf(DTM_tim )
tim_DF <- as.matrix(tfidf_tim )
DTM_tim2 <- as.data.frame(as.matrix(DTM_tim ), stringsAsFactors=False)
####
# Compare /cluster documents
####


### Next Author

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',WilliamKazer,'/*.txt'))

will  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
```

```r
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(will ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(will))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_will  = DocumentTermMatrix(my_documents)

DTM_will  = removeSparseTerms(DTM_will , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_will  = weightTfIdf(DTM_will )
will_DF <- as.matrix(tfidf_will )
DTM_will2 <- as.data.frame(as.matrix(DTM_will ), stringsAsFactors=False)

#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',BernardHickey,'/*.txt'))

bernard  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
```

```r
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(bernard ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(bernard))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_bernard  = DocumentTermMatrix(my_documents)
## ...find words with greater than a min count...
DTM_bernard  = removeSparseTerms(DTM_bernard , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_bernard  = weightTfIdf(DTM_bernard )
bernard_DF <- as.matrix(tfidf_bernard )
DTM_bernard2 <- as.data.frame(as.matrix(DTM_bernard ),
stringsAsFactors=False)


### Next Author

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',MatthewBunce,'/*.txt'))
```

```r
matthew  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(matthew ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(matthew))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_matthew  = DocumentTermMatrix(my_documents)


DTM_matthew  = removeSparseTerms(DTM_matthew , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_matthew  = weightTfIdf(DTM_matthew )
bernard_DF <- as.matrix(tfidf_matthew )
DTM_matthew2 <- as.data.frame(as.matrix(DTM_matthew ),
stringsAsFactors=False)
```

```
### Next Author

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',SamuelPerry,'/*.txt'))

samuel  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(samuel ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(samuel))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_samuel  = DocumentTermMatrix(my_documents)
DTM_samuel  = removeSparseTerms(DTM_samuel , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_samuel  = weightTfIdf(DTM_samuel )
bernard_DF <- as.matrix(tfidf_samuel )
DTM_samuel2 <- as.data.frame(as.matrix(DTM_samuel ), stringsAsFactors=False)
```

```
### Next Author

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',TanEeLyn,'/*.txt'))

tan  = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(tan ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(tan))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccttrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_tan  = DocumentTermMatrix(my_documents)


DTM_tan  = removeSparseTerms(DTM_tan , 0.95)
```

```r
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_tan  = weightTfIdf(DTM_tan )
bernard_DF <- as.matrix(tfidf_tan )
DTM_tan2 <- as.data.frame(as.matrix(DTM_tan ), stringsAsFactors=False)


### Next Author


file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',ToddNissen,'/*.txt'))

todd  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(todd ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(todd))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
```

```r
DTM_todd  = DocumentTermMatrix(my_documents)
DTM_todd  = removeSparseTerms(DTM_todd , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_todd  = weightTfIdf(DTM_todd )
bernard_DF <- as.matrix(tfidf_todd )
DTM_todd2 <- as.data.frame(as.matrix(DTM_todd ), stringsAsFactors=False)

library(e1071)

### Creating Tests
#Author1
readerPlain = function(fname){
  readPlain(elem=list(content=readLines(fname)),
            id=fname, language='en') }



list_folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0test/')
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50test/'+list_folders[i]+'/*.txt'))
p <- function(..., sep='') {
  paste(..., sep=sep, collapse=sep)
}
for (i in list_folders){
  folder = i
  assign(i,i)
}
file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',SimonCowell,'/*.txt'))

simon_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
```

```
# Rename the articles
#mynames
names(simon_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(simon_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
#stopwords("en")
#stopwords("SMART")
#?stopwords
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_simon_test = DocumentTermMatrix(my_documents)

DTM_simon_test = removeSparseTerms(DTM_simon_test, 0.95)
#DTM_simon_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_simon_test = weightTfIdf(DTM_simon_test)
simon_DF <- as.data.frame(as.matrix(tfidf_simon))
simon_DF_test <- as.data.frame(as.matrix(tfidf_simon_test))
simon_words  <- names(simon_DF)
simon_words_test  <- names(simon_DF_test)
remove_simon <- simon_words_test[!(simon_words_test %in% simon_words)]
simon_DF_test <- simon_DF_test[, !colnames(simon_DF_test) %in% remove_simon]
```

```
### Next Author


list_folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0test/')
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50test/'+list_folders[i]+'/*.txt'))

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',AaronPressman,'/*.txt'))

aaron_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(aaron_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(aaron_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess

my_documents = tm_map(my_documents, content_transformer(removeWords),
```

```r
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainaaroncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_aaron_test = DocumentTermMatrix(my_documents)

DTM_aaron_test = removeSparseTerms(DTM_aaron_test, 0.95)
#DTM_aaron_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_aaron_test = weightTfIdf(DTM_aaron_test)
aaron_DF <- as.data.frame(as.matrix(tfidf_aaron))
aaron_DF_test <- as.data.frame(as.matrix(tfidf_aaron_test))
aaron_words  <- names(aaron_DF)
aaron_words_test  <- names(aaron_DF_test)
remove_aaron <- aaron_words_test[!(aaron_words_test %in% aaron_words)]
aaron_DF_test <- aaron_DF_test[, !colnames(aaron_DF_test) %in% remove_aaron]


### Next Author


file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',AlanCrosby,'/*.txt'))

alan_test = lapply(file_list_test, readerPlain)



mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(alan_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(alan_test))
```

```
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainalancowellnewsmltxt
"))


## create a doc-term-matrix from the corpus
DTM_alan_test = DocumentTermMatrix(my_documents)

DTM_alan_test = removeSparseTerms(DTM_alan_test, 0.95)

tfidf_alan_test = weightTfIdf(DTM_alan_test)
alan_DF <- as.data.frame(as.matrix(tfidf_alan))
alan_DF_test <- as.data.frame(as.matrix(tfidf_alan_test))
alan_words  <- names(alan_DF)
alan_words_test  <- names(alan_DF_test)
remove_alan <- alan_words_test[!(alan_words_test %in% alan_words)]
alan_DF_test <- alan_DF_test[, !colnames(alan_DF_test) %in% remove_alan]


### Next Author


file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',AlexanderSmith,'/*.txt'))

alex_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
```

```r
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(alex_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(alex_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainalexcowellnewsmltxt
"))


## create a doc-term-matrix from the corpus
DTM_alex_test = DocumentTermMatrix(my_documents)

DTM_alex_test = removeSparseTerms(DTM_alex_test, 0.95)
#DTM_alex_test # now ~ 1000 terms (versus ~3000 before)



# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_alex_test = weightTfIdf(DTM_alex_test)
alex_DF <- as.data.frame(as.matrix(tfidf_alex))
alex_DF_test <- as.data.frame(as.matrix(tfidf_alex_test))
alex_words  <- names(alex_DF)
alex_words_test  <- names(alex_DF_test)
remove_alex <- alex_words_test[!(alex_words_test %in% alex_words)]
alex_DF_test <- alex_DF_test[, !colnames(alex_DF_test) %in% remove_alex]



### Next Author
```

```r
file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',BenjaminKangLim,'/*.txt'))

ben_test = lapply(file_list_test, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(ben_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(ben_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersсctrainbencowellnewsmltxt"
))


## create a doc-term-matrix from the corpus
DTM_ben_test = DocumentTermMatrix(my_documents)



DTM_ben_test = removeSparseTerms(DTM_ben_test, 0.95)
```

```r
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_ben_test = weightTfIdf(DTM_ben_test)
ben_DF <- as.data.frame(as.matrix(tfidf_ben))
ben_DF_test <- as.data.frame(as.matrix(tfidf_ben_test))
ben_words  <- names(ben_DF)
ben_words_test  <- names(ben_DF_test)
remove_ben <- ben_words_test[!(ben_words_test %in% ben_words)]
ben_DF_test <- ben_DF_test[, !colnames(ben_DF_test) %in% remove_ben]

### Next Author

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',BernardHickey,'/*.txt'))

bernard_test = lapply(file_list_test, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(bernard_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(bernard_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersctrainbernardcowellnewsml
txt"))
```

```r
## create a doc-term-matrix from the corpus
DTM_bernard_test = DocumentTermMatrix(my_documents)

DTM_bernard_test = removeSparseTerms(DTM_bernard_test, 0.95)
#DTM_bernard_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_bernard_test = weightTfIdf(DTM_bernard_test)
bernard_DF <- as.data.frame(as.matrix(tfidf_bernard))
bernard_DF_test <- as.data.frame(as.matrix(tfidf_bernard_test))
bernard_words  <- names(bernard_DF)
bernard_words_test  <- names(bernard_DF_test)
remove_bernard <- bernard_words_test[!(bernard_words_test %in%
bernard_words)]
bernard_DF_test <- bernard_DF_test[, !colnames(bernard_DF_test) %in%
remove_bernard]



### Next Author

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',BradDorfman,'/*.txt'))

brad_test = lapply(file_list_test, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(brad_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(brad_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
```

```
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainbradcowellnewsmltxt
"))


## create a doc-term-matrix from the corpus
DTM_brad_test = DocumentTermMatrix(my_documents)


## Finally, let's drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
DTM_brad_test = removeSparseTerms(DTM_brad_test, 0.95)
#DTM_brad_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_brad_test = weightTfIdf(DTM_brad_test)
brad_DF <- as.data.frame(as.matrix(tfidf_brad))
brad_DF_test <- as.data.frame(as.matrix(tfidf_brad_test))
brad_words  <- names(brad_DF)
brad_words_test  <- names(brad_DF_test)
remove_brad <- brad_words_test[!(brad_words_test %in% brad_words)]
brad_DF_test <- brad_DF_test[, !colnames(brad_DF_test) %in% remove_brad]


### Next Author


file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',DarrenSchuettler,'/*.txt'))

darren_test = lapply(file_list_test, readerPlain)
```

```r
# The file names are ugly...
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(darren_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(darren_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctraindarrencowellnewsmlt
xt"))


## create a doc-term-matrix from the corpus
DTM_darren_test = DocumentTermMatrix(my_documents)


DTM_darren_test = removeSparseTerms(DTM_darren_test, 0.95)
#DTM_darren_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_darren_test = weightTfIdf(DTM_darren_test)
darren_DF <- as.data.frame(as.matrix(tfidf_darren))
```

```r
darren_DF_test <- as.data.frame(as.matrix(tfidf_darren_test))
darren_words  <- names(darren_DF)
darren_words_test  <- names(darren_DF_test)
remove_darren <- darren_words_test[!(darren_words_test %in% darren_words)]
darren_DF_test <- darren_DF_test[, !colnames(darren_DF_test) %in%
remove_darren]




### Next Author

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',DavidLawder,'/*.txt'))

david_test = lapply(file_list_test, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(david_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(david_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccttraindavidcowellnewsmltx
t"))
```

```r
## create a doc-term-matrix from the corpus
DTM_david_test = DocumentTermMatrix(my_documents)

DTM_david_test = removeSparseTerms(DTM_david_test, 0.95)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_david_test = weightTfIdf(DTM_david_test)
david_DF <- as.data.frame(as.matrix(tfidf_david))
david_DF_test <- as.data.frame(as.matrix(tfidf_david_test))
david_words  <- names(david_DF)
david_words_test  <- names(david_DF_test)
remove_david <- david_words_test[!(david_words_test %in% david_words)]
david_DF_test <- david_DF_test[, !colnames(david_DF_test) %in% remove_david]

### Next Author


file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',EdnaFernandes,'/*.txt'))

edna_test = lapply(file_list_test, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(edna_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(edna_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
```

```r
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainednacowellnewsmltxt
"))


## create a doc-term-matrix from the corpus
DTM_edna_test = DocumentTermMatrix(my_documents)

DTM_edna_test = removeSparseTerms(DTM_edna_test, 0.95)
#DTM_edna_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_edna_test = weightTfIdf(DTM_edna_test)
edna_DF <- as.data.frame(as.matrix(tfidf_edna))
edna_DF_test <- as.data.frame(as.matrix(tfidf_edna_test))
edna_words  <- names(edna_DF)
edna_words_test  <- names(edna_DF_test)
remove_edna <- edna_words_test[!(edna_words_test %in% edna_words)]
edna_DF_test <- edna_DF_test[, !colnames(edna_DF_test) %in% remove_edna]


### Next Author



file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',EricAuchard,'/*.txt'))

eric_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
```

```r
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(eric_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(eric_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainericcowellnewsmltxt
"))


## create a doc-term-matrix from the corpus
DTM_eric_test = DocumentTermMatrix(my_documents)
#DTM_eric_test # some basic summary statistics
## You can inspect its entries...
#inspect(DTM_eric_test[1:10,1:20])

## ...find words with greater than a min count...
#findFreqTerms(DTM_eric_test, 50)

## ...or find words whose count correlates with a specified word.
# the top entries here look like they go with "genetic"
#findAssocs(DTM_eric_test, "genetic", .5)

## Finally, let's drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
DTM_eric_test = removeSparseTerms(DTM_eric_test, 0.95)
```

```
#DTM_eric_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_eric_test = weightTfIdf(DTM_eric_test)
eric_DF <- as.data.frame(as.matrix(tfidf_eric))
eric_DF_test <- as.data.frame(as.matrix(tfidf_eric_test))
eric_words  <- names(eric_DF)
eric_words_test  <- names(eric_DF_test)
remove_eric <- eric_words_test[!(eric_words_test %in% eric_words)]
eric_DF_test <- eric_DF_test[, !colnames(eric_DF_test) %in% remove_eric]


### Next Author


file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',FumikoFujisaki,'/*.txt'))

fumiko_test = lapply(file_list_test, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(fumiko_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(fumiko_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

my_documents = tm_map(my_documents, content_transformer(removeWords),
```

```
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainfumikocowellnewsmlt
xt"))


## create a doc-term-matrix from the corpus
DTM_fumiko_test = DocumentTermMatrix(my_documents)

DTM_fumiko_test = removeSparseTerms(DTM_fumiko_test, 0.95)
#DTM_fumiko_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_fumiko_test = weightTfIdf(DTM_fumiko_test)
fumiko_DF <- as.data.frame(as.matrix(tfidf_fumiko))
fumiko_DF_test <- as.data.frame(as.matrix(tfidf_fumiko_test))
fumiko_words  <- names(fumiko_DF)
fumiko_words_test  <- names(fumiko_DF_test)
remove_fumiko <- fumiko_words_test[!(fumiko_words_test %in% fumiko_words)]
fumiko_DF_test <- fumiko_DF_test[, !colnames(fumiko_DF_test) %in%
remove_fumiko]


### Next Author

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',GrahamEarnshaw,'/*.txt'))

graham_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(graham_test) = mynames
```

```
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(graham_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctraingrahamcowellnewsmlt
xt"))


## create a doc-term-matrix from the corpus
DTM_graham_test = DocumentTermMatrix(my_documents)

DTM_graham_test = removeSparseTerms(DTM_graham_test, 0.95)
#DTM_graham_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_graham_test = weightTfIdf(DTM_graham_test)
graham_DF <- as.data.frame(as.matrix(tfidf_graham))
graham_DF_test <- as.data.frame(as.matrix(tfidf_graham_test))
graham_words  <- names(graham_DF)
graham_words_test  <- names(graham_DF_test)
remove_graham <- graham_words_test[!(graham_words_test %in% graham_words)]
graham_DF_test <- graham_DF_test[, !colnames(graham_DF_test) %in%
remove_graham]


### Next Author
file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',HeatherScoffield,'/*.txt'))

heather_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
#file_list_test
```

```r
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(heather_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(heather_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainheathercowellnewsml
txt"))


## create a doc-term-matrix from the corpus
DTM_heather_test = DocumentTermMatrix(my_documents)

DTM_heather_test = removeSparseTerms(DTM_heather_test, 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_heather_test = weightTfIdf(DTM_heather_test)
heather_DF <- as.data.frame(as.matrix(tfidf_heather))
heather_DF_test <- as.data.frame(as.matrix(tfidf_heather_test))
heather_words  <- names(heather_DF)
heather_words_test  <- names(heather_DF_test)
```

```r
remove_heather <- heather_words_test[!(heather_words_test %in%
heather_words)]
heather_DF_test <- heather_DF_test[, !colnames(heather_DF_test) %in%
remove_heather]


## Next Author


file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',JanLopatka,'/*.txt'))

jan_test = lapply(file_list_test, readerPlain)

mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(jan_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jan_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainjancowellnewsmltxt"
))

## create a doc-term-matrix from the corpus
DTM_jan_test = DocumentTermMatrix(my_documents)

DTM_jan_test = removeSparseTerms(DTM_jan_test, 0.95)
```

```r
#DTM_jan_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_jan_test = weightTfIdf(DTM_jan_test)
jan_DF <- as.data.frame(as.matrix(tfidf_jan))
jan_DF_test <- as.data.frame(as.matrix(tfidf_jan_test))
jan_words  <- names(jan_DF)
jan_words_test  <- names(jan_DF_test)
remove_jan <- jan_words_test[!(jan_words_test %in% jan_words)]
jan_DF_test <- jan_DF_test[, !colnames(jan_DF_test) %in% remove_jan]

### Next Author

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',JaneMacartney,'/*.txt'))

jane_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(jane_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jane_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
```

```
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainjanecowellnewsmltxt
"))


## create a doc-term-matrix from the corpus
DTM_jane_test = DocumentTermMatrix(my_documents)
DTM_jane_test = removeSparseTerms(DTM_jane_test, 0.95)
#DTM_jane_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_jane_test = weightTfIdf(DTM_jane_test)
jane_DF <- as.data.frame(as.matrix(tfidf_jane))
jane_DF_test <- as.data.frame(as.matrix(tfidf_jane_test))
jane_words  <- names(jane_DF)
jane_words_test  <- names(jane_DF_test)
remove_jane <- jane_words_test[!(jane_words_test %in% jane_words)]
jane_DF_test <- jane_DF_test[, !colnames(jane_DF_test) %in% remove_jane]

### Next Author

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',JimGilchrist,'/*.txt'))

jim_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
```

```r
#mynames
names(jim_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jim_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
#stopwords("en")
#stopwords("SMART")
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainjimcowellnewsmltxt"
))


## create a doc-term-matrix from the corpus
DTM_jim_test = DocumentTermMatrix(my_documents)

DTM_jim_test = removeSparseTerms(DTM_jim_test, 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_jim_test = weightTfIdf(DTM_jim_test)
jim_DF <- as.data.frame(as.matrix(tfidf_jim))
jim_DF_test <- as.data.frame(as.matrix(tfidf_jim_test))
jim_words  <- names(jim_DF)
jim_words_test  <- names(jim_DF_test)
remove_jim <- jim_words_test[!(jim_words_test %in% jim_words)]
jim_DF_test <- jim_DF_test[, !colnames(jim_DF_test) %in% remove_jim]

library(tm)
library(tidyverse)
library(slam)
library(proxy)
```

```
library(tidytext)
library(dplyr)

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



## apply to all of Simon Cowell's articles
## (probably not THE Simon Cowell: https://twitter.com/simoncowell)
## "globbing" = expanding wild cards in filename paths

list.dirs <- function(path=".", pattern=NULL, all.dirs=FALSE,
                      full.names=FALSE, ignore.case=FALSE) {
  # use full.names=TRUE to pass to file.info
  all <- list.files(path, pattern, all.dirs,
                    full.names=TRUE, recursive=FALSE, ignore.case)
  dirs <- all[file.info(all)$isdir]
  # determine whether to return full names or just dir names
  if(isTRUE(full.names))
    return(dirs)
  else
    return(basename(dirs))
}

list_folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0train/')
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
p <- function(..., sep='') {
  paste(..., sep=sep, collapse=sep)
}
for (i in list_folders){
  folder = i
  assign(i,i)
}
file_list =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',SimonCowell,'/*.txt'))

simon = lapply(file_list, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list %>%
```

```r
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(simon) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(simon))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_simon = DocumentTermMatrix(my_documents)




## Finally, let's drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
DTM_simon = removeSparseTerms(DTM_simon, 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_simon = weightTfIdf(DTM_simon)
simon_DF <- as.matrix(tfidf_simon)
DTM_simon2 <- as.data.frame(as.matrix(DTM_simon), stringsAsFactors=False)
```

```
##########################################################################
#Next Author

#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
filename =
p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C50train/'
,AaronPressman,'/*.txt')
file_listA = Sys.glob(filename)

aaron = lapply(file_listA, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesA = file_listA %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist


names(aaron) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(aaron))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
```

```
c("cusersmachuonedrivedocumentsgithubstadatareutersccttrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_aaron = DocumentTermMatrix(my_documents)
## You can inspect its entries...

DTM_aaron = removeSparseTerms(DTM_aaron, 0.95)
tfidf_aaron = weightTfIdf(DTM_aaron)
aaron_DF <- as.matrix(tfidf_aaron)
DTM_aaron2 <- as.data.frame(as.matrix(DTM_aaron), stringsAsFactors=False)


###next Author
list_folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0train/')

#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',AlexanderSmith,'/*.txt'))

alex = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(alex) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(alex))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
```

```
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_alex = DocumentTermMatrix(my_documents)


DTM_alex = removeSparseTerms(DTM_alex, 0.95)

tfidf_alex = weightTfIdf(DTM_alex)
alex_DF <- as.matrix(tfidf_alex)

###next Author
list_folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0train/')

#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',AlanCrosby,'/*.txt'))

alan = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
```

```r
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(alan) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(alan))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_alan = DocumentTermMatrix(my_documents)


DTM_alan = removeSparseTerms(DTM_alan, 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_alan = weightTfIdf(DTM_alan)
alan_DF <- as.matrix(tfidf_alan)
DTM_alan2 <- as.data.frame(as.matrix(DTM_alan), stringsAsFactors=False)

# the proxy library has a built-in function to calculate cosine distance
# define the cosine distance matrix for our DTM using this function

####
```

```
# Dimensionality reduction
####

# Now PCA on term frequencies
###Next Author
list_folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0train/')

#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',BenjaminKangLim,'/*.txt'))

ben = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
names(ben) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(ben))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
```

```
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_ben = DocumentTermMatrix(my_documents)

DTM_ben = removeSparseTerms(DTM_ben, 0.95)


tfidf_ben = weightTfIdf(DTM_ben)
ben_DF <- as.matrix(tfidf_ben)
DTM_ben2 <- as.data.frame(as.matrix(DTM_ben), stringsAsFactors=False)

###Next Author
list_folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0train/')

#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',BenjaminKangLim,'/*.txt'))

ben = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(ben) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(ben))
```

```
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything 
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess 
white-space

## Remove stopwords.  Always be careful with this: one person's trash is 
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords), 
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords, 
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_ben = DocumentTermMatrix(my_documents)

DTM_ben = removeSparseTerms(DTM_ben, 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_ben = weightTfIdf(DTM_ben)
ben_DF <- as.matrix(tfidf_ben)
DTM_ben2 <- as.data.frame(as.matrix(DTM_ben), stringsAsFactors=False)


##Next Author

#file_list + list_folders[i] = 
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl = 
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',BradDorfman,'/*.txt'))

brad = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
```

```
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles

names(brad) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(brad))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_brad = DocumentTermMatrix(my_documents)

## Finally, let's drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
DTM_brad = removeSparseTerms(DTM_brad, 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_brad = weightTfIdf(DTM_brad)
```

```
brad_DF <- as.matrix(tfidf_brad)
DTM_brad2 <- as.data.frame(as.matrix(DTM_brad), stringsAsFactors=False)


### Next Author

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',DarrenSchuettler,'/*.txt'))

darren = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(darren) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(darren))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersctrainsimoncowellnewsmltx
t"))
```

```
## create a doc-term-matrix from the corpus
DTM_darren = DocumentTermMatrix(my_documents)

DTM_darren = removeSparseTerms(DTM_darren, 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_darren = weightTfIdf(DTM_darren)
darren_DF <- as.matrix(tfidf_darren)
DTM_darren2 <- as.data.frame(as.matrix(DTM_darren), stringsAsFactors=False)

##Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',DavidLawder,'/*.txt'))

david = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(david) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(david))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
```

```r
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_david = DocumentTermMatrix(my_documents)

DTM_david = removeSparseTerms(DTM_david, 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_david = weightTfIdf(DTM_david)
david_DF <- as.matrix(tfidf_david)
DTM_david2 <- as.data.frame(as.matrix(DTM_david), stringsAsFactors=False)

### Next Author
file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',EdnaFernandes,'/*.txt'))

edna  = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(edna ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(edna ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
```

```
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersctrainsimoncowellnewsmltx
t"))

## create a doc-term-matrix from the corpus
DTM_edna  = DocumentTermMatrix(my_documents)

DTM_edna  = removeSparseTerms(DTM_edna , 0.95)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_edna  = weightTfIdf(DTM_edna )
edna_DF <- as.matrix(tfidf_edna )
DTM_edna2 <- as.data.frame(as.matrix(DTM_edna ), stringsAsFactors=False)

###Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',EricAuchard,'/*.txt'))

eric  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(eric ) = mynames
```

```r
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(eric ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_eric  = DocumentTermMatrix(my_documents)

DTM_eric  = removeSparseTerms(DTM_eric , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_eric  = weightTfIdf(DTM_eric )
eric_DF <- as.matrix(tfidf_eric )
DTM_eric2 <- as.data.frame(as.matrix(DTM_eric ), stringsAsFactors=False)

###Next Author

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',FumikoFujisaki,'/*.txt'))

fumiko  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
```

```
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist


names(fumiko ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(fumiko ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))

## create a doc-term-matrix from the corpus
DTM_fumiko  = DocumentTermMatrix(my_documents)

DTM_fumiko  = removeSparseTerms(DTM_fumiko , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_fumiko  = weightTfIdf(DTM_fumiko )
fumiko_DF <- as.matrix(tfidf_fumiko )
DTM_fumiko2 <- as.data.frame(as.matrix(DTM_fumiko ), stringsAsFactors=False)
### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
```

```r
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',GrahamEarnshaw,'/*.txt'))

graham  = lapply(file_listAl, readerPlain)



# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist


names(graham ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(graham ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersсctrainsimoncowellnewsmltx
t"))

## create a doc-term-matrix from the corpus
DTM_graham  = DocumentTermMatrix(my_documents)

DTM_graham  = removeSparseTerms(DTM_graham , 0.95)
```

```
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_graham  = weightTfIdf(DTM_graham )
graham_DF <- as.matrix(tfidf_graham )
DTM_graham2 <- as.data.frame(as.matrix(DTM_graham ), stringsAsFactors=False)

### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',HeatherScoffield,'/*.txt'))

heather  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(heather ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(heather ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
```

```
c("cusersmachuonedrivedocumentsgithubstadatareutersc ctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_heather  = DocumentTermMatrix(my_documents)

DTM_heather  = removeSparseTerms(DTM_heather , 0.95)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_heather  = weightTfIdf(DTM_heather )
heather_DF <- as.matrix(tfidf_heather )
DTM_heather2 <- as.data.frame(as.matrix(DTM_heather ),
stringsAsFactors=False)


###Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JanLopatka,'/*.txt'))

jan  = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(jan ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jan ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
```

```
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_jan  = DocumentTermMatrix(my_documents)

DTM_jan  = removeSparseTerms(DTM_jan , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_jan  = weightTfIdf(DTM_jan )
jan_DF <- as.matrix(tfidf_jan )
DTM_jan2 <- as.data.frame(as.matrix(DTM_jan ), stringsAsFactors=False)

### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JaneMacartney,'/*.txt'))

jane  = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(jane ) = mynames
```

```r
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jane ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("usersmachuonedrivedocumentsgithubstadatareutersc ctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_jane  = DocumentTermMatrix(my_documents)

DTM_jane  = removeSparseTerms(DTM_jane , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_jane  = weightTfIdf(DTM_jane )
jane_DF <- as.matrix(tfidf_jane )
DTM_jane2 <- as.data.frame(as.matrix(DTM_jane ), stringsAsFactors=False)

###Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JimGilchrist,'/*.txt'))

jim  = lapply(file_listAl, readerPlain)



# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
```

```
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist


names(jim ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jim ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_jim  = DocumentTermMatrix(my_documents)

DTM_jim  = removeSparseTerms(DTM_jim , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_jim  = weightTfIdf(DTM_jim )
jim_DF <- as.matrix(tfidf_jim )
DTM_jim2 <- as.data.frame(as.matrix(DTM_jim ), stringsAsFactors=False)

### Next Author

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JoWinterbottom,'/*.txt'))

jo  = lapply(file_listAl, readerPlain)
```

```r
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(jo ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jo ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%              # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%         # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%     # remove punctuation
  tm_map(content_transformer(stripWhitespace))           # remove excess
white-space


my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))


## create a doc-term-matrix from the corpus
DTM_jo  = DocumentTermMatrix(my_documents)

DTM_jo  = removeSparseTerms(DTM_jo , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_jo  = weightTfIdf(DTM_jo )
jo_DF <- as.matrix(tfidf_jo )
DTM_jo2 <- as.data.frame(as.matrix(DTM_jo ), stringsAsFactors=False)

## Next Author

#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
```

```
file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JoeOrtiz,'/*.txt'))

joe  = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
names(joe ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(joe ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))


## create a doc-term-matrix from the corpus
DTM_joe  = DocumentTermMatrix(my_documents)

DTM_joe  = removeSparseTerms(DTM_joe , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_joe  = weightTfIdf(DTM_joe )
joe_DF <- as.matrix(tfidf_joe )
DTM_joe2 <- as.data.frame(as.matrix(DTM_joe ), stringsAsFactors=False)
```

```
### Next Author

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JohnMastrini,'/*.txt'))

john  = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(john ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(john ))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_john  = DocumentTermMatrix(my_documents)

DTM_john  = removeSparseTerms(DTM_john , 0.95)


# construct TF IDF weights -- might be useful if we wanted to use these
```

```r
# as features in a predictive model
tfidf_john  = weightTfIdf(DTM_john )
john_DF <- as.matrix(tfidf_john )
DTM_john2 <- as.data.frame(as.matrix(DTM_john ), stringsAsFactors=False)

## Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',JonathanBirt,'/*.txt'))

jonathan  = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(jonathan ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jonathan))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))
```

```r
## create a doc-term-matrix from the corpus
DTM_jonathan  = DocumentTermMatrix(my_documents)

DTM_jonathan  = removeSparseTerms(DTM_jonathan , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_jonathan  = weightTfIdf(DTM_jonathan )
jonathan_DF <- as.matrix(tfidf_jonathan )
DTM_jonathan2 <- as.data.frame(as.matrix(DTM_jonathan ),
stringsAsFactors=False)

### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KarlPenhaul,'/*.txt'))

karl  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(karl ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(karl))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space
```

```
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersctrainsimoncowellnewsmltx
t"))

## create a doc-term-matrix from the corpus
DTM_karl  = DocumentTermMatrix(my_documents)

DTM_karl  = removeSparseTerms(DTM_karl , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_karl  = weightTfIdf(DTM_karl )
karl_DF <- as.matrix(tfidf_karl )
DTM_karl2 <- as.data.frame(as.matrix(DTM_karl ), stringsAsFactors=False)


## Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KeithWeir,'/*.txt'))

keith  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(keith ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(keith))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
```

```
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_keith  = DocumentTermMatrix(my_documents)

DTM_keith  = removeSparseTerms(DTM_keith , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_keith  = weightTfIdf(DTM_keith )
keith_DF <- as.matrix(tfidf_keith )
DTM_keith2 <- as.data.frame(as.matrix(DTM_keith ), stringsAsFactors=False)


### Next Author

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KevinDrawbaugh,'/*.txt'))

kevind  = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(kevind ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
```

```
documents_raw = Corpus(VectorSource(kevind))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersсctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_kevind  = DocumentTermMatrix(my_documents)

## ...find words with greater than a min count...
## ...or fiDTM_kevind ds whose count correlates with a specified word.
# the top entries here look like they go with "genetic"
DTM_kevind  = removeSparseTerms(DTM_kevind , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_kevind  = weightTfIdf(DTM_kevind )
kevind_DF <- as.matrix(tfidf_kevind )
DTM_kevind2 <- as.data.frame(as.matrix(DTM_kevind ), stringsAsFactors=False)


### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KevinMorrison,'/*.txt'))

kevinm  = lapply(file_listAl, readerPlain)


# Clean up the file names
```

```
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(kevinm ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(kevinm))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_kevinm  = DocumentTermMatrix(my_documents)

DTM_kevinm  = removeSparseTerms(DTM_kevinm , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_kevinm  = weightTfIdf(DTM_kevinm )
kevinm_DF <- as.matrix(tfidf_kevinm )
DTM_kevinm2 <- as.data.frame(as.matrix(DTM_kevinm ), stringsAsFactors=False)
####
# Compare /cluster documents
####

### Next Author
```

```
file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KirstinRidley,'/*.txt'))

kirstin  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(kirstin ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(kirstin))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersсctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_kirstin  = DocumentTermMatrix(my_documents)

DTM_kirstin  = removeSparseTerms(DTM_kirstin , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_kirstin  = weightTfIdf(DTM_kirstin )
kirstin_DF <- as.matrix(tfidf_kirstin )
```

```
DTM_kirstin2 <- as.data.frame(as.matrix(DTM_kirstin ),
stringsAsFactors=False)


### Next Author

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',KouroshKarimkhany,'/*.txt'))

kourosh  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(kourosh ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(kourosh))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))

## create a doc-term-matrix from the corpus
DTM_kourosh  = DocumentTermMatrix(my_documents)

DTM_kourosh  = removeSparseTerms(DTM_kourosh , 0.95)
```

```
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_kourosh  = weightTfIdf(DTM_kourosh )
kourosh_DF <- as.matrix(tfidf_kourosh )
DTM_kourosh2 <- as.data.frame(as.matrix(DTM_kourosh ),
stringsAsFactors=False)
####
# Compare /cluster documents




### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',LydiaZajc,'/*.txt'))

lydia  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(lydia ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(lydia))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space



# let's just use the "basic English" stop words
```

```r
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_lydia  = DocumentTermMatrix(my_documents)

DTM_lydia  = removeSparseTerms(DTM_lydia , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_lydia  = weightTfIdf(DTM_lydia )
lydia_DF <- as.matrix(tfidf_lydia )
DTM_lydia2 <- as.data.frame(as.matrix(DTM_lydia ), stringsAsFactors=False)
####
# Compare /cluster documents
####




## Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',`LynneO'Donnell`,'/*.txt'))

lynne  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(lynne ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(lynne))
```

```
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_lynne  = DocumentTermMatrix(my_documents)

DTM_lynne  = removeSparseTerms(DTM_lynne , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_lynne  = weightTfIdf(DTM_lynne )
lynne_DF <- as.matrix(tfidf_lynne )
DTM_lynne2 <- as.data.frame(as.matrix(DTM_lynne ), stringsAsFactors=False)
####
# Compare /cluster documents
####

#Next Author

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',LynnleyBrowning,'/*.txt'))

lynnley  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(lynnley ) = mynames
```

```
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(lynnley))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_lynnley  = DocumentTermMatrix(my_documents)

DTM_lynnley  = removeSparseTerms(DTM_lynnley , 0.95)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_lynnley  = weightTfIdf(DTM_lynnley )
lynnley_DF <- as.matrix(tfidf_lynnley )
DTM_lynnley2 <- as.data.frame(as.matrix(DTM_lynnley ),
stringsAsFactors=False)
####
# Compare /cluster documents
####




### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',MarcelMichelson,'/*.txt'))
```

```
marcel  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(marcel ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(marcel))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))

## create a doc-term-matrix from the corpus
DTM_marcel  = DocumentTermMatrix(my_documents)

## Finally, let's drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
DTM_marcel  = removeSparseTerms(DTM_marcel , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_marcel  = weightTfIdf(DTM_marcel )
marcel_DF <- as.matrix(tfidf_marcel )
DTM_marcel2 <- as.data.frame(as.matrix(DTM_marcel ), stringsAsFactors=False)
```

```r
### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',MarkBendeich,'/*.txt'))

mark  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(mark ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(mark))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
```

```r
DTM_mark  = DocumentTermMatrix(my_documents)

## Finally, let's drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
DTM_mark  = removeSparseTerms(DTM_mark , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_mark  = weightTfIdf(DTM_mark )
mark_DF <- as.matrix(tfidf_mark )
DTM_mark2 <- as.data.frame(as.matrix(DTM_mark ), stringsAsFactors=False)


### Next Question
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',MartinWolk,'/*.txt'))

martin  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(martin ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(martin))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
```

```
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("usersmachuonedrivedocumentsgithubstadatareutersctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_martin  = DocumentTermMatrix(my_documents)
DTM_martin  = removeSparseTerms(DTM_martin , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_martin  = weightTfIdf(DTM_martin )
martin_DF <- as.matrix(tfidf_martin )
DTM_martin2 <- as.data.frame(as.matrix(DTM_martin ), stringsAsFactors=False)

### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',MichaelConnor,'/*.txt'))

michael  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(michael ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(michael))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
```

```
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_michael  = DocumentTermMatrix(my_documents)

DTM_michael  = removeSparseTerms(DTM_michael , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_michael  = weightTfIdf(DTM_michael )
michael_DF <- as.matrix(tfidf_michael )
DTM_michael2 <- as.data.frame(as.matrix(DTM_michael ),
stringsAsFactors=False)

### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',MureDickie,'/*.txt'))

mure  = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(mure ) = mynames
```

```r
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(mure))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_mure  = DocumentTermMatrix(my_documents)

DTM_mure  = removeSparseTerms(DTM_mure , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_mure  = weightTfIdf(DTM_mure )
mure_DF <- as.matrix(tfidf_mure )
DTM_mure2 <- as.data.frame(as.matrix(DTM_mure ), stringsAsFactors=False)

### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',NickLouth,'/*.txt'))

nick  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
```

```
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(nick ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(nick))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_nick  = DocumentTermMatrix(my_documents)

## Finally, let's drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
DTM_nick  = removeSparseTerms(DTM_nick , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_nick  = weightTfIdf(DTM_nick )
nick_DF <- as.matrix(tfidf_nick )
DTM_nick2 <- as.data.frame(as.matrix(DTM_nick ), stringsAsFactors=False)

### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))
```

```r
file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',PatriciaCommins,'/*.txt'))

patricia  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(patricia ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(patricia))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))

## create a doc-term-matrix from the corpus
DTM_patricia  = DocumentTermMatrix(my_documents)

DTM_patricia  = removeSparseTerms(DTM_patricia , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_patricia  = weightTfIdf(DTM_patricia )
patricia_DF <- as.matrix(tfidf_patricia )
```

```
DTM_patricia2 <- as.data.frame(as.matrix(DTM_patricia ),
stringsAsFactors=False)

### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',PeterHumphrey,'/*.txt'))

peter  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(peter ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(peter))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%            # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))



## create a doc-term-matrix from the corpus
```

```
DTM_peter  = DocumentTermMatrix(my_documents)

## Finally, let's drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
DTM_peter  = removeSparseTerms(DTM_peter , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_peter  = weightTfIdf(DTM_peter )
peter_DF <- as.matrix(tfidf_peter )
DTM_peter2 <- as.data.frame(as.matrix(DTM_peter ), stringsAsFactors=False)

### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',PierreTran,'/*.txt'))

pierre  = lapply(file_listAl, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(pierre ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(pierre))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space
```

```
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_pierre  = DocumentTermMatrix(my_documents)

DTM_pierre  = removeSparseTerms(DTM_pierre , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_pierre  = weightTfIdf(DTM_pierre )
pierre_DF <- as.matrix(tfidf_pierre )
DTM_pierre2 <- as.data.frame(as.matrix(DTM_pierre ), stringsAsFactors=False)

### Next Author
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',RobinSidel,'/*.txt'))

robin  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(robin ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(robin))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
```

```
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_robin  = DocumentTermMatrix(my_documents)

DTM_robin  = removeSparseTerms(DTM_robin , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_robin  = weightTfIdf(DTM_robin )
robin_DF <- as.matrix(tfidf_robin )
DTM_robin2 <- as.data.frame(as.matrix(DTM_robin ), stringsAsFactors=False)

### Next Author

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',RogerFillion,'/*.txt'))

roger  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(roger ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(roger))
```

```r
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything 
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess 
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords), 
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords, 
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_roger  = DocumentTermMatrix(my_documents)
DTM_roger  = removeSparseTerms(DTM_roger , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_roger  = weightTfIdf(DTM_roger )
roger_DF <- as.matrix(tfidf_roger )
DTM_roger2 <- as.data.frame(as.matrix(DTM_roger ), stringsAsFactors=False)

### Next Author
#file_list + list_folders[i] = 
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl = 
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',SarahDavison,'/*.txt'))

sarah  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
```

```r
names(sarah ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(sarah))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_sarah  = DocumentTermMatrix(my_documents)


DTM_sarah  = removeSparseTerms(DTM_sarah , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_sarah  = weightTfIdf(DTM_sarah )
sarah_DF <- as.matrix(tfidf_sarah )
DTM_sarah2 <- as.data.frame(as.matrix(DTM_sarah ), stringsAsFactors=False)


### Next Question
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',ScottHillis,'/*.txt'))

scott  = lapply(file_listAl, readerPlain)
# Clean up the file names
```

```r
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(scott ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(scott))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))

DTM_scott  = DocumentTermMatrix(my_documents)


## Finally, let's drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
DTM_scott  = removeSparseTerms(DTM_scott , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_scott  = weightTfIdf(DTM_scott )
scott_DF <- as.matrix(tfidf_scott )
DTM_scott2 <- as.data.frame(as.matrix(DTM_scott ), stringsAsFactors=False)



### Next Author
```

```
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',TheresePoletti,'/*.txt'))

therese  = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(therese ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(therese))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_therese  = DocumentTermMatrix(my_documents)

DTM_therese  = removeSparseTerms(DTM_therese , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
```

```
# as features in a predictive model
tfidf_therese  = weightTfIdf(DTM_therese )
therese_DF <- as.matrix(tfidf_therese )
DTM_therese2 <- as.data.frame(as.matrix(DTM_therese ),
stringsAsFactors=False)


### Next Author

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',TimFarrand,'/*.txt'))

tim  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(tim ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(tim))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))



## create a doc-term-matrix from the corpus
DTM_tim  = DocumentTermMatrix(my_documents)
```

```
DTM_tim  = removeSparseTerms(DTM_tim , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_tim  = weightTfIdf(DTM_tim )
tim_DF <- as.matrix(tfidf_tim )
DTM_tim2 <- as.data.frame(as.matrix(DTM_tim ), stringsAsFactors=False)
####
# Compare /cluster documents
####



### Next Author

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',WilliamKazer,'/*.txt'))

will  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(will ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(will))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
```

```
#my_documents <- tm_map(my_documents, removeWords, 
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_will  = DocumentTermMatrix(my_documents)

DTM_will  = removeSparseTerms(DTM_will , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_will  = weightTfIdf(DTM_will )
will_DF <- as.matrix(tfidf_will )
DTM_will2 <- as.data.frame(as.matrix(DTM_will ), stringsAsFactors=False)

#file_list + list_folders[i] = 
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50train/'+list_folders[i]+'/*.txt'))

file_listAl = 
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',BernardHickey,'/*.txt'))

bernard  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(bernard ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(bernard))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything 
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess 
white-space
```

```
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_bernard  = DocumentTermMatrix(my_documents)
## ...find words with greater than a min count...
DTM_bernard  = removeSparseTerms(DTM_bernard , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_bernard  = weightTfIdf(DTM_bernard )
bernard_DF <- as.matrix(tfidf_bernard )
DTM_bernard2 <- as.data.frame(as.matrix(DTM_bernard ),
stringsAsFactors=False)


### Next Author

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',MatthewBunce,'/*.txt'))

matthew  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(matthew ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(matthew))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
```

```
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_matthew  = DocumentTermMatrix(my_documents)


DTM_matthew  = removeSparseTerms(DTM_matthew , 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_matthew  = weightTfIdf(DTM_matthew )
bernard_DF <- as.matrix(tfidf_matthew )
DTM_matthew2 <- as.data.frame(as.matrix(DTM_matthew ),
stringsAsFactors=False)

### Next Author

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',SamuelPerry,'/*.txt'))

samuel  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(samuel ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(samuel))
```

```
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_samuel  = DocumentTermMatrix(my_documents)
DTM_samuel  = removeSparseTerms(DTM_samuel , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_samuel  = weightTfIdf(DTM_samuel )
bernard_DF <- as.matrix(tfidf_samuel )
DTM_samuel2 <- as.data.frame(as.matrix(DTM_samuel ), stringsAsFactors=False)


### Next Author

file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',TanEeLyn,'/*.txt'))

tan  = lapply(file_listAl, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(tan ) = mynames
```

```
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(tan))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_tan  = DocumentTermMatrix(my_documents)


DTM_tan  = removeSparseTerms(DTM_tan , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_tan  = weightTfIdf(DTM_tan )
bernard_DF <- as.matrix(tfidf_tan )
DTM_tan2 <- as.data.frame(as.matrix(DTM_tan ), stringsAsFactors=False)


### Next Author


file_listAl =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50train/',ToddNissen,'/*.txt'))

todd  = lapply(file_listAl, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
```

```
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
names(todd ) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(todd))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
#my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersсctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_todd  = DocumentTermMatrix(my_documents)
DTM_todd  = removeSparseTerms(DTM_todd , 0.95)
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_todd  = weightTfIdf(DTM_todd )
bernard_DF <- as.matrix(tfidf_todd )
DTM_todd2 <- as.data.frame(as.matrix(DTM_todd ), stringsAsFactors=False)


library(e1071)

### Creating Tests
#Author1
readerPlain = function(fname){
  readPlain(elem=list(content=readLines(fname)),
            id=fname, language='en') }




list_folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
```

```
0test/')
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50test/'+list_folders[i]+'/*.txt'))
p <- function(..., sep='') {
  paste(..., sep=sep, collapse=sep)
}
for (i in list_folders){
  folder = i
  assign(i,i)
}
file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',SimonCowell,'/*.txt'))

simon_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(simon_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(simon_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
```

```
another one's treasure.
# 2 example built-in sets of stop words
#stopwords("en")
#stopwords("SMART")
#?stopwords
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_simon_test = DocumentTermMatrix(my_documents)

DTM_simon_test = removeSparseTerms(DTM_simon_test, 0.95)
#DTM_simon_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_simon_test = weightTfIdf(DTM_simon_test)
simon_DF <- as.data.frame(as.matrix(tfidf_simon))
simon_DF_test <- as.data.frame(as.matrix(tfidf_simon_test))
simon_words  <- names(simon_DF)
simon_words_test  <- names(simon_DF_test)
remove_simon <- simon_words_test[!(simon_words_test %in% simon_words)]
simon_DF_test <- simon_DF_test[, !colnames(simon_DF_test) %in% remove_simon]


### Next Author


list_folders <-
list.dirs('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C5
0test/')
#file_list + list_folders[i] =
Sys.glob(paste('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC
50/C50test/'+list_folders[i]+'/*.txt'))

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',AaronPressman,'/*.txt'))

aaron_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
```

```r
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(aaron_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(aaron_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess

my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainaaroncowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_aaron_test = DocumentTermMatrix(my_documents)

DTM_aaron_test = removeSparseTerms(DTM_aaron_test, 0.95)
#DTM_aaron_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_aaron_test = weightTfIdf(DTM_aaron_test)
aaron_DF <- as.data.frame(as.matrix(tfidf_aaron))
aaron_DF_test <- as.data.frame(as.matrix(tfidf_aaron_test))
aaron_words  <- names(aaron_DF)
aaron_words_test  <- names(aaron_DF_test)
```

```r
remove_aaron <- aaron_words_test[!(aaron_words_test %in% aaron_words)]
aaron_DF_test <- aaron_DF_test[, !colnames(aaron_DF_test) %in% remove_aaron]


### Next Author


file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',AlanCrosby,'/*.txt'))

alan_test = lapply(file_list_test, readerPlain)



mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(alan_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(alan_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersctrainalancowellnewsmltxt
"))


## create a doc-term-matrix from the corpus
DTM_alan_test = DocumentTermMatrix(my_documents)
```

```r
DTM_alan_test = removeSparseTerms(DTM_alan_test, 0.95)

tfidf_alan_test = weightTfIdf(DTM_alan_test)
alan_DF <- as.data.frame(as.matrix(tfidf_alan))
alan_DF_test <- as.data.frame(as.matrix(tfidf_alan_test))
alan_words  <- names(alan_DF)
alan_words_test  <- names(alan_DF_test)
remove_alan <- alan_words_test[!(alan_words_test %in% alan_words)]
alan_DF_test <- alan_DF_test[, !colnames(alan_DF_test) %in% remove_alan]


### Next Author


file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',AlexanderSmith,'/*.txt'))

alex_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(alex_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(alex_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
```

```
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainalexcowellnewsmltxt
"))


## create a doc-term-matrix from the corpus
DTM_alex_test = DocumentTermMatrix(my_documents)

DTM_alex_test = removeSparseTerms(DTM_alex_test, 0.95)
#DTM_alex_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_alex_test = weightTfIdf(DTM_alex_test)
alex_DF <- as.data.frame(as.matrix(tfidf_alex))
alex_DF_test <- as.data.frame(as.matrix(tfidf_alex_test))
alex_words  <- names(alex_DF)
alex_words_test  <- names(alex_DF_test)
remove_alex <- alex_words_test[!(alex_words_test %in% alex_words)]
alex_DF_test <- alex_DF_test[, !colnames(alex_DF_test) %in% remove_alex]


### Next Author

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',BenjaminKangLim,'/*.txt'))

ben_test = lapply(file_list_test, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(ben_test) = mynames
```

```r
## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(ben_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainbencowellnewsmltxt"
))


## create a doc-term-matrix from the corpus
DTM_ben_test = DocumentTermMatrix(my_documents)



DTM_ben_test = removeSparseTerms(DTM_ben_test, 0.95)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_ben_test = weightTfIdf(DTM_ben_test)
ben_DF <- as.data.frame(as.matrix(tfidf_ben))
ben_DF_test <- as.data.frame(as.matrix(tfidf_ben_test))
ben_words  <- names(ben_DF)
ben_words_test  <- names(ben_DF_test)
remove_ben <- ben_words_test[!(ben_words_test %in% ben_words)]
ben_DF_test <- ben_DF_test[, !colnames(ben_DF_test) %in% remove_ben]

### Next Author

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',BernardHickey,'/*.txt'))

bernard_test = lapply(file_list_test, readerPlain)
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
```

```r
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(bernard_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(bernard_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainbernardcowellnewsml
txt"))


## create a doc-term-matrix from the corpus
DTM_bernard_test = DocumentTermMatrix(my_documents)

DTM_bernard_test = removeSparseTerms(DTM_bernard_test, 0.95)
#DTM_bernard_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_bernard_test = weightTfIdf(DTM_bernard_test)
bernard_DF <- as.data.frame(as.matrix(tfidf_bernard))
bernard_DF_test <- as.data.frame(as.matrix(tfidf_bernard_test))
bernard_words  <- names(bernard_DF)
bernard_words_test  <- names(bernard_DF_test)
remove_bernard <- bernard_words_test[!(bernard_words_test %in%
bernard_words)]
bernard_DF_test <- bernard_DF_test[, !colnames(bernard_DF_test) %in%
remove_bernard]
```

```r
### Next Author

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',BradDorfman,'/*.txt'))

brad_test = lapply(file_list_test, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(brad_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(brad_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainbradcowellnewsmltxt
"))


## create a doc-term-matrix from the corpus
DTM_brad_test = DocumentTermMatrix(my_documents)


## Finally, let's drop those terms that only occur in one or two documents
```

```r
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
DTM_brad_test = removeSparseTerms(DTM_brad_test, 0.95)
#DTM_brad_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_brad_test = weightTfIdf(DTM_brad_test)
brad_DF <- as.data.frame(as.matrix(tfidf_brad))
brad_DF_test <- as.data.frame(as.matrix(tfidf_brad_test))
brad_words  <- names(brad_DF)
brad_words_test  <- names(brad_DF_test)
remove_brad <- brad_words_test[!(brad_words_test %in% brad_words)]
brad_DF_test <- brad_DF_test[, !colnames(brad_DF_test) %in% remove_brad]


### Next Author


file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',DarrenSchuettler,'/*.txt'))

darren_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(darren_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(darren_test))
```

```
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctraindarrencowellnewsmlt
xt"))


## create a doc-term-matrix from the corpus
DTM_darren_test = DocumentTermMatrix(my_documents)


DTM_darren_test = removeSparseTerms(DTM_darren_test, 0.95)
#DTM_darren_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_darren_test = weightTfIdf(DTM_darren_test)
darren_DF <- as.data.frame(as.matrix(tfidf_darren))
darren_DF_test <- as.data.frame(as.matrix(tfidf_darren_test))
darren_words  <- names(darren_DF)
darren_words_test  <- names(darren_DF_test)
remove_darren <- darren_words_test[!(darren_words_test %in% darren_words)]
darren_DF_test <- darren_DF_test[, !colnames(darren_DF_test) %in%
remove_darren]




### Next Author

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',DavidLawder,'/*.txt'))

david_test = lapply(file_list_test, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
```

```r
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(david_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(david_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctraindavidcowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_david_test = DocumentTermMatrix(my_documents)

DTM_david_test = removeSparseTerms(DTM_david_test, 0.95)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_david_test = weightTfIdf(DTM_david_test)
david_DF <- as.data.frame(as.matrix(tfidf_david))
david_DF_test <- as.data.frame(as.matrix(tfidf_david_test))
david_words  <- names(david_DF)
david_words_test  <- names(david_DF_test)
remove_david <- david_words_test[!(david_words_test %in% david_words)]
david_DF_test <- david_DF_test[, !colnames(david_DF_test) %in% remove_david]

### Next Author
```

```
file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',EdnaFernandes,'/*.txt'))

edna_test = lapply(file_list_test, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(edna_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(edna_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainednacowellnewsmltxt
"))


## create a doc-term-matrix from the corpus
DTM_edna_test = DocumentTermMatrix(my_documents)

DTM_edna_test = removeSparseTerms(DTM_edna_test, 0.95)
#DTM_edna_test # now ~ 1000 terms (versus ~3000 before)



# construct TF IDF weights -- might be useful if we wanted to use these
```

```
# as features in a predictive model
tfidf_edna_test = weightTfIdf(DTM_edna_test)
edna_DF <- as.data.frame(as.matrix(tfidf_edna))
edna_DF_test <- as.data.frame(as.matrix(tfidf_edna_test))
edna_words  <- names(edna_DF)
edna_words_test  <- names(edna_DF_test)
remove_edna <- edna_words_test[!(edna_words_test %in% edna_words)]
edna_DF_test <- edna_DF_test[, !colnames(edna_DF_test) %in% remove_edna]


### Next Author



file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',EricAuchard,'/*.txt'))

eric_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(eric_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(eric_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
```

```
white-space

my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainericcowellnewsmltxt
"))


## create a doc-term-matrix from the corpus
DTM_eric_test = DocumentTermMatrix(my_documents)
#DTM_eric_test # some basic summary statistics
## You can inspect its entries...
#inspect(DTM_eric_test[1:10,1:20])

## ...find words with greater than a min count...
#findFreqTerms(DTM_eric_test, 50)

## ...or find words whose count correlates with a specified word.
# the top entries here look like they go with "genetic"
#findAssocs(DTM_eric_test, "genetic", .5)

## Finally, let's drop those terms that only occur in one or two documents
## This is a common step: the noise of the "long tail" (rare terms)
## can be huge, and there is nothing to learn if a term occured once.
## Below removes those terms that have count 0 in >95% of docs.
## Probably a bit stringent here... but only 50 docs!
DTM_eric_test = removeSparseTerms(DTM_eric_test, 0.95)
#DTM_eric_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_eric_test = weightTfIdf(DTM_eric_test)
eric_DF <- as.data.frame(as.matrix(tfidf_eric))
eric_DF_test <- as.data.frame(as.matrix(tfidf_eric_test))
eric_words  <- names(eric_DF)
eric_words_test  <- names(eric_DF_test)
remove_eric <- eric_words_test[!(eric_words_test %in% eric_words)]
eric_DF_test <- eric_DF_test[, !colnames(eric_DF_test) %in% remove_eric]


### Next Author


file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',FumikoFujisaki,'/*.txt'))
```

```r
fumiko_test = lapply(file_list_test, readerPlain)

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(fumiko_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(fumiko_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainfumikocowellnewsmlt
xt"))


## create a doc-term-matrix from the corpus
DTM_fumiko_test = DocumentTermMatrix(my_documents)

DTM_fumiko_test = removeSparseTerms(DTM_fumiko_test, 0.95)
#DTM_fumiko_test # now ~ 1000 terms (versus ~3000 before)



# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_fumiko_test = weightTfIdf(DTM_fumiko_test)
fumiko_DF <- as.data.frame(as.matrix(tfidf_fumiko))
fumiko_DF_test <- as.data.frame(as.matrix(tfidf_fumiko_test))
fumiko_words  <- names(fumiko_DF)
fumiko_words_test  <- names(fumiko_DF_test)
remove_fumiko <- fumiko_words_test[!(fumiko_words_test %in% fumiko_words)]
```

```r
fumiko_DF_test <- fumiko_DF_test[, !colnames(fumiko_DF_test) %in%
remove_fumiko]


### Next Author

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',GrahamEarnshaw,'/*.txt'))

graham_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(graham_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(graham_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctraingrahamcowellnewsmlt
xt"))


## create a doc-term-matrix from the corpus
```

```
DTM_graham_test = DocumentTermMatrix(my_documents)

DTM_graham_test = removeSparseTerms(DTM_graham_test, 0.95)
#DTM_graham_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_graham_test = weightTfIdf(DTM_graham_test)
graham_DF <- as.data.frame(as.matrix(tfidf_graham))
graham_DF_test <- as.data.frame(as.matrix(tfidf_graham_test))
graham_words  <- names(graham_DF)
graham_words_test  <- names(graham_DF_test)
remove_graham <- graham_words_test[!(graham_words_test %in% graham_words)]
graham_DF_test <- graham_DF_test[, !colnames(graham_DF_test) %in%
remove_graham]


### Next Author
file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',HeatherScoffield,'/*.txt'))

heather_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(heather_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(heather_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
```

```
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainheathercowellnewsml
txt"))


## create a doc-term-matrix from the corpus
DTM_heather_test = DocumentTermMatrix(my_documents)

DTM_heather_test = removeSparseTerms(DTM_heather_test, 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_heather_test = weightTfIdf(DTM_heather_test)
heather_DF <- as.data.frame(as.matrix(tfidf_heather))
heather_DF_test <- as.data.frame(as.matrix(tfidf_heather_test))
heather_words  <- names(heather_DF)
heather_words_test  <- names(heather_DF_test)
remove_heather <- heather_words_test[!(heather_words_test %in%
heather_words)]
heather_DF_test <- heather_DF_test[, !colnames(heather_DF_test) %in%
remove_heather]


## Next Author


file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',JanLopatka,'/*.txt'))

jan_test = lapply(file_list_test, readerPlain)

mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
```

```
# Rename the articles
#mynames
names(jan_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jan_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainjancowellnewsmltxt"
))

## create a doc-term-matrix from the corpus
DTM_jan_test = DocumentTermMatrix(my_documents)

DTM_jan_test = removeSparseTerms(DTM_jan_test, 0.95)
#DTM_jan_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_jan_test = weightTfIdf(DTM_jan_test)
jan_DF <- as.data.frame(as.matrix(tfidf_jan))
jan_DF_test <- as.data.frame(as.matrix(tfidf_jan_test))
jan_words  <- names(jan_DF)
jan_words_test  <- names(jan_DF_test)
remove_jan <- jan_words_test[!(jan_words_test %in% jan_words)]
jan_DF_test <- jan_DF_test[, !colnames(jan_DF_test) %in% remove_jan]

### Next Author

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',JaneMacartney,'/*.txt'))

jane_test = lapply(file_list_test, readerPlain)
```

```r
# The file names are ugly...
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(jane_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jane_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords), stopwords("en"))
my_documents <- tm_map(my_documents, removeWords, c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainjanecowellnewsmltxt"))


## create a doc-term-matrix from the corpus
DTM_jane_test = DocumentTermMatrix(my_documents)
DTM_jane_test = removeSparseTerms(DTM_jane_test, 0.95)
#DTM_jane_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_jane_test = weightTfIdf(DTM_jane_test)
jane_DF <- as.data.frame(as.matrix(tfidf_jane))
```

```
jane_DF_test <- as.data.frame(as.matrix(tfidf_jane_test))
jane_words  <- names(jane_DF)
jane_words_test  <- names(jane_DF_test)
remove_jane <- jane_words_test[!(jane_words_test %in% jane_words)]
jane_DF_test <- jane_DF_test[, !colnames(jane_DF_test) %in% remove_jane]

### Next Author

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',JimGilchrist,'/*.txt'))

jim_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(jim_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jim_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
#stopwords("en")
```

```r
#stopwords("SMART")
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainjimcowellnewsmltxt"
))


## create a doc-term-matrix from the corpus
DTM_jim_test = DocumentTermMatrix(my_documents)

DTM_jim_test = removeSparseTerms(DTM_jim_test, 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_jim_test = weightTfIdf(DTM_jim_test)
jim_DF <- as.data.frame(as.matrix(tfidf_jim))
jim_DF_test <- as.data.frame(as.matrix(tfidf_jim_test))
jim_words  <- names(jim_DF)
jim_words_test  <- names(jim_DF_test)
remove_jim <- jim_words_test[!(jim_words_test %in% jim_words)]
jim_DF_test <- jim_DF_test[, !colnames(jim_DF_test) %in% remove_jim]
### Next Author

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',JoWinterbottom,'/*.txt'))

jo_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(jo_test) = mynames

## once you have documents in a vector, you
```

```r
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jo_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
#stopwords("en")
#stopwords("SMART")
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccTrainjocowellnewsmltxt")
)


## create a doc-term-matrix from the corpus
DTM_jo_test = DocumentTermMatrix(my_documents)

DTM_jo_test = removeSparseTerms(DTM_jo_test, 0.95)
#DTM_jo_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_jo_test = weightTfIdf(DTM_jo_test)
jo_DF <- as.data.frame(as.matrix(tfidf_jo))
jo_DF_test <- as.data.frame(as.matrix(tfidf_jo_test))
jo_words  <- names(jo_DF)
jo_words_test  <- names(jo_DF_test)
remove_jo <- jo_words_test[!(jo_words_test %in% jo_words)]
jo_DF_test <- jo_DF_test[, !colnames(jo_DF_test) %in% remove_jo]

### Next Author

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',JoeOrtiz,'/*.txt'))
```

```r
joe_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(joe_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(joe_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainjoecowellnewsmltxt"
))


## create a doc-term-matrix from the corpus
DTM_joe_test = DocumentTermMatrix(my_documents)

DTM_joe_test = removeSparseTerms(DTM_joe_test, 0.95)
#DTM_joe_test # now ~ 1000 terms (versus ~3000 before)



# construct TF IDF weights -- might be useful if we wanted to use these
```

```
# as features in a predictive model
tfidf_joe_test = weightTfIdf(DTM_joe_test)
joe_DF <- as.data.frame(as.matrix(tfidf_joe))
joe_DF_test <- as.data.frame(as.matrix(tfidf_joe_test))
joe_words  <- names(joe_DF)
joe_words_test  <- names(joe_DF_test)
remove_joe <- joe_words_test[!(joe_words_test %in% joe_words)]
joe_DF_test <- joe_DF_test[, !colnames(joe_DF_test) %in% remove_joe]

### Next Author

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',JohnMastrini,'/*.txt'))

john_test = lapply(file_list_test, readerPlain)


# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(john_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(john_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainjohncowellnewsmltxt
```

```
"))


## create a doc-term-matrix from the corpus
DTM_john_test = DocumentTermMatrix(my_documents)

DTM_john_test = removeSparseTerms(DTM_john_test, 0.95)
#DTM_john_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_john_test = weightTfIdf(DTM_john_test)
john_DF <- as.data.frame(as.matrix(tfidf_john))
john_DF_test <- as.data.frame(as.matrix(tfidf_john_test))
john_words  <- names(john_DF)
john_words_test  <- names(john_DF_test)
remove_john <- john_words_test[!(john_words_test %in% john_words)]
john_DF_test <- john_DF_test[, !colnames(john_DF_test) %in% remove_john]


### Next Author

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',JonathanBirt,'/*.txt'))

jonathan_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(jonathan_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(jonathan_test))
```

```
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything 
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess 
white-space

my_documents = tm_map(my_documents, content_transformer(removeWords), 
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords, 
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainjonathancowellnewsm
ltxt"))


## create a doc-term-matrix from the corpus
DTM_jonathan_test = DocumentTermMatrix(my_documents)

DTM_jonathan_test = removeSparseTerms(DTM_jonathan_test, 0.95)
#DTM_jonathan_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_jonathan_test = weightTfIdf(DTM_jonathan_test)
jonathan_DF <- as.data.frame(as.matrix(tfidf_jonathan))
jonathan_DF_test <- as.data.frame(as.matrix(tfidf_jonathan_test))
jonathan_words  <- names(jonathan_DF)
jonathan_words_test  <- names(jonathan_DF_test)
remove_jonathan <- jonathan_words_test[!(jonathan_words_test %in% 
jonathan_words)]
jonathan_DF_test <- jonathan_DF_test[, !colnames(jonathan_DF_test) %in% 
remove_jonathan]

### Next Author

file_list_test = 
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',KarlPenhaul,'/*.txt'))

karl_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
#file_list_test
```

```
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(karl_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(karl_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainkarlcowellnewsmltxt
"))


## create a doc-term-matrix from the corpus
DTM_karl_test = DocumentTermMatrix(my_documents)

DTM_karl_test = removeSparseTerms(DTM_karl_test, 0.95)
#DTM_karl_test # now ~ 1000 terms (versus ~3000 before)



# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_karl_test = weightTfIdf(DTM_karl_test)
karl_DF <- as.data.frame(as.matrix(tfidf_karl))
karl_DF_test <- as.data.frame(as.matrix(tfidf_karl_test))
karl_words  <- names(karl_DF)
karl_words_test  <- names(karl_DF_test)
remove_karl <- karl_words_test[!(karl_words_test %in% karl_words)]
```

```
karl_DF_test <- karl_DF_test[, !colnames(karl_DF_test) %in% remove_karl]

### Next Author

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',KeithWeir,'/*.txt'))

keith_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(keith_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(keith_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
#stopwords("en")
#stopwords("SMART")
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
```

```r
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainkeithcowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_keith_test = DocumentTermMatrix(my_documents)

DTM_keith_test = removeSparseTerms(DTM_keith_test, 0.95)
#DTM_keith_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_keith_test = weightTfIdf(DTM_keith_test)
keith_DF <- as.data.frame(as.matrix(tfidf_keith))
keith_DF_test <- as.data.frame(as.matrix(tfidf_keith_test))
keith_words  <- names(keith_DF)
keith_words_test  <- names(keith_DF_test)
remove_keith <- keith_words_test[!(keith_words_test %in% keith_words)]
keith_DF_test <- keith_DF_test[, !colnames(keith_DF_test) %in% remove_keith]

### Next Author

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',KevinDrawbaugh,'/*.txt'))

kevind_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(kevind_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
```

```
documents_raw = Corpus(VectorSource(kevind_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainkevindcowellnewsmlt
xt"))


## create a doc-term-matrix from the corpus
DTM_kevind_test = DocumentTermMatrix(my_documents)

DTM_kevind_test = removeSparseTerms(DTM_kevind_test, 0.95)
#DTM_kevind_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_kevind_test = weightTfIdf(DTM_kevind_test)
kevind_DF <- as.data.frame(as.matrix(tfidf_kevind))
kevind_DF_test <- as.data.frame(as.matrix(tfidf_kevind_test))
kevind_words  <- names(kevind_DF)
kevind_words_test  <- names(kevind_DF_test)
remove_kevind <- kevind_words_test[!(kevind_words_test %in% kevind_words)]
kevind_DF_test <- kevind_DF_test[, !colnames(kevind_DF_test) %in%
remove_kevind]


### Next Author

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',KevinMorrison,'/*.txt'))

kevinm_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
#file_list_test
```

```r
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(kevinm_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(kevinm_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainkevinmcowellnewsmlt
xt"))


## create a doc-term-matrix from the corpus
DTM_kevinm_test = DocumentTermMatrix(my_documents)

DTM_kevinm_test = removeSparseTerms(DTM_kevinm_test, 0.95)
#DTM_kevinm_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_kevinm_test = weightTfIdf(DTM_kevinm_test)
kevinm_DF <- as.data.frame(as.matrix(tfidf_kevinm))
kevinm_DF_test <- as.data.frame(as.matrix(tfidf_kevinm_test))
kevinm_words  <- names(kevinm_DF)
```

```
kevinm_words_test  <- names(kevinm_DF_test)
remove_kevinm <- kevinm_words_test[!(kevinm_words_test %in% kevinm_words)]
kevinm_DF_test <- kevinm_DF_test[, !colnames(kevinm_DF_test) %in%
remove_kevinm]


### Next Author

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',KirstinRidley,'/*.txt'))

kirstin_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(kirstin_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(kirstin_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
```

```r
c("cusersmachuonedrivedocumentsgithubstadatareutersccttrainkirstincowellnewsml
txt"))


## create a doc-term-matrix from the corpus
DTM_kirstin_test = DocumentTermMatrix(my_documents)

DTM_kirstin_test = removeSparseTerms(DTM_kirstin_test, 0.95)
#DTM_kirstin_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_kirstin_test = weightTfIdf(DTM_kirstin_test)
kirstin_DF <- as.data.frame(as.matrix(tfidf_kirstin))
kirstin_DF_test <- as.data.frame(as.matrix(tfidf_kirstin_test))
kirstin_words  <- names(kirstin_DF)
kirstin_words_test  <- names(kirstin_DF_test)
remove_kirstin <- kirstin_words_test[!(kirstin_words_test %in%
kirstin_words)]
kirstin_DF_test <- kirstin_DF_test[, !colnames(kirstin_DF_test) %in%
remove_kirstin]

### Next Author

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',KouroshKarimkhany,'/*.txt'))

kourosh_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(kourosh_test) = mynames

## once you have documents in a vector, you
```

```
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(kourosh_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainkouroshcowellnewsml
txt"))


## create a doc-term-matrix from the corpus
DTM_kourosh_test = DocumentTermMatrix(my_documents)

## Probably a bit stringent here... but only 50 docs!
DTM_kourosh_test = removeSparseTerms(DTM_kourosh_test, 0.95)
#DTM_kourosh_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_kourosh_test = weightTfIdf(DTM_kourosh_test)
kourosh_DF <- as.data.frame(as.matrix(tfidf_kourosh))
kourosh_DF_test <- as.data.frame(as.matrix(tfidf_kourosh_test))
kourosh_words  <- names(kourosh_DF)
kourosh_words_test  <- names(kourosh_DF_test)
remove_kourosh <- kourosh_words_test[!(kourosh_words_test %in%
kourosh_words)]
kourosh_DF_test <- kourosh_DF_test[, !colnames(kourosh_DF_test) %in%
remove_kourosh]

### Next Author


file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',LydiaZajc,'/*.txt'))

lydia_test = lapply(file_list_test, readerPlain)
```

```r
# The file names are ugly...
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(lydia_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(lydia_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space


# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainlydiacowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_lydia_test = DocumentTermMatrix(my_documents)

DTM_lydia_test = removeSparseTerms(DTM_lydia_test, 0.95)
#DTM_lydia_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
```

```
tfidf_lydia_test = weightTfIdf(DTM_lydia_test)
lydia_DF <- as.data.frame(as.matrix(tfidf_lydia))
lydia_DF_test <- as.data.frame(as.matrix(tfidf_lydia_test))
lydia_words  <- names(lydia_DF)
lydia_words_test  <- names(lydia_DF_test)
remove_lydia <- lydia_words_test[!(lydia_words_test %in% lydia_words)]
lydiah_DF_test <- lydia_DF_test[, !colnames(lydia_DF_test) %in% remove_lydia]


### Next Author

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',`LynneO'Donnell`,'/*.txt'))

lynne_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(lynne_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(lynne_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
```

```r
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainlynnecowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_lynne_test = DocumentTermMatrix(my_documents)

DTM_lynne_test = removeSparseTerms(DTM_lynne_test, 0.95)
#DTM_lynne_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_lynne_test = weightTfIdf(DTM_lynne_test)
lynne_DF <- as.data.frame(as.matrix(tfidf_lynne))
lynne_DF_test <- as.data.frame(as.matrix(tfidf_lynne_test))
lynne_words  <- names(lynne_DF)
lynne_words_test  <- names(lynne_DF_test)
remove_lynne <- lynne_words_test[!(lynne_words_test %in% lynne_words)]
lynne_DF_test <- lynne_DF_test[, !colnames(lynne_DF_test) %in% remove_lynne]


### Next Author

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',LynnleyBrowning,'/*.txt'))

lynnley_test = lapply(file_list_test, readerPlain)

mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(lynnley_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(lynnley_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
```

```
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainlynnleycowellnewsml
txt"))


## create a doc-term-matrix from the corpus
DTM_lynnley_test = DocumentTermMatrix(my_documents)

DTM_lynnley_test = removeSparseTerms(DTM_lynnley_test, 0.95)
#DTM_lynnley_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_lynnley_test = weightTfIdf(DTM_lynnley_test)
lynnley_DF <- as.data.frame(as.matrix(tfidf_lynnley))
lynnley_DF_test <- as.data.frame(as.matrix(tfidf_lynnley_test))
lynnley_words  <- names(lynnley_DF)
lynnley_words_test  <- names(lynnley_DF_test)
remove_lynnley <- lynnley_words_test[!(lynnley_words_test %in%
lynnley_words)]
lynnley_DF_test <- lynnley_DF_test[, !colnames(lynnley_DF_test) %in%
remove_lynnley]



### Next Author

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',MarcelMichelson,'/*.txt'))

marcel_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
#file_list_test

# Clean up the file names
```

```r
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(marcel_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(marcel_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainmarcelcowellnewsmlt
xt"))


## create a doc-term-matrix from the corpus
DTM_marcel_test = DocumentTermMatrix(my_documents)

DTM_marcel_test = removeSparseTerms(DTM_marcel_test, 0.95)
#DTM_marcel_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_marcel_test = weightTfIdf(DTM_marcel_test)
marcel_DF <- as.data.frame(as.matrix(tfidf_marcel))
marcel_DF_test <- as.data.frame(as.matrix(tfidf_marcel_test))
marcel_words  <- names(marcel_DF)
marcel_words_test  <- names(marcel_DF_test)
remove_marcel <- marcel_words_test[!(marcel_words_test %in% marcel_words)]
marcel_DF_test <- marcel_DF_test[, !colnames(marcel_DF_test) %in%
```

```
remove_marcel]

### Next Author


file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',MarkBendeich,'/*.txt'))

mark_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(mark_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(mark_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainmarkcowellnewsmltxt
"))
```

```r
## create a doc-term-matrix from the corpus
DTM_mark_test = DocumentTermMatrix(my_documents)

DTM_mark_test = removeSparseTerms(DTM_mark_test, 0.95)
#DTM_mark_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_mark_test = weightTfIdf(DTM_mark_test)
mark_DF <- as.data.frame(as.matrix(tfidf_mark))
mark_DF_test <- as.data.frame(as.matrix(tfidf_mark_test))
mark_words  <- names(mark_DF)
mark_words_test  <- names(mark_DF_test)
remove_mark <- mark_words_test[!(mark_words_test %in% mark_words)]
mark_DF_test <- mark_DF_test[, !colnames(mark_DF_test) %in% remove_mark]


### Next Author

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',MartinWolk,'/*.txt'))

martin_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(martin_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(martin_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
```

```r
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
#stopwords("en")
#stopwords("SMART")
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainmartincowellnewsmlt
xt"))


## create a doc-term-matrix from the corpus
DTM_martin_test = DocumentTermMatrix(my_documents)

DTM_martin_test = removeSparseTerms(DTM_martin_test, 0.95)
#DTM_martin_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_martin_test = weightTfIdf(DTM_martin_test)
martin_DF <- as.data.frame(as.matrix(tfidf_martin))
martin_DF_test <- as.data.frame(as.matrix(tfidf_martin_test))
martin_words  <- names(martin_DF)
martin_words_test  <- names(martin_DF_test)
remove_martin <- martin_words_test[!(martin_words_test %in% martin_words)]
martin_DF_test <- martin_DF_test[, !colnames(martin_DF_test) %in%
remove_martin]

### Next Author

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',MatthewBunce,'/*.txt'))

matthew_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
```

```r
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(matthew_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(matthew_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

## Remove stopwords.  Always be careful with this: one person's trash is
another one's treasure.
# 2 example built-in sets of stop words
#stopwords("en")
#stopwords("SMART")
# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainmatthewcowellnewsml
txt"))


## create a doc-term-matrix from the corpus
DTM_matthew_test = DocumentTermMatrix(my_documents)

DTM_matthew_test = removeSparseTerms(DTM_matthew_test, 0.95)
#DTM_matthew_test # now ~ 1000 terms (versus ~3000 before)
```

```r
# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_matthew_test = weightTfIdf(DTM_matthew_test)
matthew_DF <- as.data.frame(as.matrix(tfidf_matthew))
matthew_DF_test <- as.data.frame(as.matrix(tfidf_matthew_test))
matthew_words  <- names(matthew_DF)
matthew_words_test  <- names(matthew_DF_test)
remove_matthew <- matthew_words_test[!(matthew_words_test %in%
matthew_words)]
matthew_DF_test <- matthew_DF_test[, !colnames(matthew_DF_test) %in%
remove_matthew]

### Next Author

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',MichaelConnor,'/*.txt'))

michael_test = lapply(file_list_test, readerPlain)



# The file names are ugly...
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(michael_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(michael_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
```

```
white-space

my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainmichaelcowellnewsml
txt"))


## create a doc-term-matrix from the corpus
DTM_michael_test = DocumentTermMatrix(my_documents)

DTM_michael_test = removeSparseTerms(DTM_michael_test, 0.95)
#DTM_michael_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_michael_test = weightTfIdf(DTM_michael_test)
michael_DF <- as.data.frame(as.matrix(tfidf_michael))
michael_DF_test <- as.data.frame(as.matrix(tfidf_michael_test))
michael_words  <- names(michael_DF)
michael_words_test  <- names(michael_DF_test)
remove_michael <- michael_words_test[!(michael_words_test %in%
michael_words)]
michael_DF_test <- michael_DF_test[, !colnames(michael_DF_test) %in%
remove_michael]


### Next Author

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',MureDickie,'/*.txt'))

mure_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist
```

```r
# Rename the articles
#mynames
names(mure_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(mure_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainmurecowellnewsmltxt
"))


## create a doc-term-matrix from the corpus
DTM_mure_test = DocumentTermMatrix(my_documents)

DTM_mure_test = removeSparseTerms(DTM_mure_test, 0.95)
#DTM_mure_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_mure_test = weightTfIdf(DTM_mure_test)
mure_DF <- as.data.frame(as.matrix(tfidf_mure))
mure_DF_test <- as.data.frame(as.matrix(tfidf_mure_test))
mure_words  <- names(mure_DF)
mure_words_test  <- names(mure_DF_test)
remove_mure <- mure_words_test[!(mure_words_test %in% mure_words)]
mure_DF_test <- mure_DF_test[, !colnames(mure_DF_test) %in% remove_mure]


### Next Author

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',NickLouth,'/*.txt'))
```

```r
nick_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(nick_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(nick_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainnickcowellnewsmltxt
"))


## create a doc-term-matrix from the corpus
DTM_nick_test = DocumentTermMatrix(my_documents)

DTM_nick_test = removeSparseTerms(DTM_nick_test, 0.95)
#DTM_nick_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
```

```
# as features in a predictive model
tfidf_nick_test = weightTfIdf(DTM_nick_test)
nick_DF <- as.data.frame(as.matrix(tfidf_nick))
nick_DF_test <- as.data.frame(as.matrix(tfidf_nick_test))
nick_words  <- names(nick_DF)
nick_words_test  <- names(nick_DF_test)
remove_nick <- nick_words_test[!(nick_words_test %in% nick_words)]
nick_DF_test <- nick_DF_test[, !colnames(nick_DF_test) %in% remove_nick]


### Next Author

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',PatriciaCommins,'/*.txt'))

patricia_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

names(patricia_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(patricia_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
```

```r
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainpatriciacowellnewsm
ltxt"))


## create a doc-term-matrix from the corpus
DTM_patricia_test = DocumentTermMatrix(my_documents)

DTM_patricia_test = removeSparseTerms(DTM_patricia_test, 0.95)
#DTM_patricia_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_patricia_test = weightTfIdf(DTM_patricia_test)
patricia_DF <- as.data.frame(as.matrix(tfidf_patricia))
patricia_DF_test <- as.data.frame(as.matrix(tfidf_patricia_test))
patricia_words  <- names(patricia_DF)
patricia_words_test  <- names(patricia_DF_test)
remove_patricia <- patricia_words_test[!(patricia_words_test %in%
patricia_words)]
patricia_DF_test <- patricia_DF_test[, !colnames(patricia_DF_test) %in%
remove_patricia]




### Next Author

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',PeterHumphrey,'/*.txt'))

peter_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
```

```r
names(peter_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(peter_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainpetercowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_peter_test = DocumentTermMatrix(my_documents)

DTM_peter_test = removeSparseTerms(DTM_peter_test, 0.95)
#DTM_peter_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_peter_test = weightTfIdf(DTM_peter_test)
peter_DF <- as.data.frame(as.matrix(tfidf_peter))
peter_DF_test <- as.data.frame(as.matrix(tfidf_peter_test))
peter_words  <- names(peter_DF)
peter_words_test  <- names(peter_DF_test)
remove_peter <- peter_words_test[!(peter_words_test %in% peter_words)]
peter_DF_test <- peter_DF_test[, !colnames(peter_DF_test) %in% remove_peter]

### Next Author

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',PierreTran,'/*.txt'))

pierre_test = lapply(file_list_test, readerPlain)
```

```
# The file names are ugly...
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(pierre_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(pierre_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainpierrecowellnewsmlt
xt"))


## create a doc-term-matrix from the corpus
DTM_pierre_test = DocumentTermMatrix(my_documents)

DTM_pierre_test = removeSparseTerms(DTM_pierre_test, 0.95)
#DTM_pierre_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_pierre_test = weightTfIdf(DTM_pierre_test)
pierre_DF <- as.data.frame(as.matrix(tfidf_pierre))
pierre_DF_test <- as.data.frame(as.matrix(tfidf_pierre_test))
```

```
pierre_words  <- names(pierre_DF)
pierre_words_test  <- names(pierre_DF_test)
remove_pierre <- pierre_words_test[!(pierre_words_test %in% pierre_words)]
pierre_DF_test <- pierre_DF_test[, !colnames(pierre_DF_test) %in%
remove_pierre]




### Next Author

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',RobinSidel,'/*.txt'))

robin_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(robin_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(robin_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
```

```
my_documents <- tm_map(my_documents, removeWords,
c("usersmachuonedrivedocumentsgithubstadatareuterscctrainrobincowellnewsmltx
t"))


## create a doc-term-matrix from the corpus
DTM_robin_test = DocumentTermMatrix(my_documents)

DTM_robin_test = removeSparseTerms(DTM_robin_test, 0.95)
#DTM_robin_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_robin_test = weightTfIdf(DTM_robin_test)
robin_DF <- as.data.frame(as.matrix(tfidf_robin))
robin_DF_test <- as.data.frame(as.matrix(tfidf_robin_test))
robin_words  <- names(robin_DF)
robin_words_test  <- names(robin_DF_test)
remove_robin <- robin_words_test[!(robin_words_test %in% robin_words)]
robin_DF_test <- robin_DF_test[, !colnames(robin_DF_test) %in% remove_robin]


### Next Author

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',RogerFillion,'/*.txt'))

roger_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(roger_test) = mynames

## once you have documents in a vector, you
```

```r
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(roger_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything 
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess 
white-space

my_documents = tm_map(my_documents, content_transformer(removeWords), 
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords, 
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainrogercowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM_roger_test = DocumentTermMatrix(my_documents)


DTM_roger_test = removeSparseTerms(DTM_roger_test, 0.95)
#DTM_roger_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_roger_test = weightTfIdf(DTM_roger_test)
roger_DF <- as.data.frame(as.matrix(tfidf_roger))
roger_DF_test <- as.data.frame(as.matrix(tfidf_roger_test))
roger_words  <- names(roger_DF)
roger_words_test  <- names(roger_DF_test)
remove_roger <- roger_words_test[!(roger_words_test %in% roger_words)]





### Next Author

file_list_test = 
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',SamuelPerry,'/*.txt'))

samuel_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
#file_list_test
```

```r
# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(samuel_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(samuel_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space


my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersc50trainsamuelcowellnewsmlt
xt"))


## create a doc-term-matrix from the corpus
DTM_samuel_test = DocumentTermMatrix(my_documents)

DTM_samuel_test = removeSparseTerms(DTM_samuel_test, 0.95)
#DTM_samuel_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_samuel_test = weightTfIdf(DTM_samuel_test)
samuel_DF <- as.data.frame(as.matrix(tfidf_samuel))
samuel_DF_test <- as.data.frame(as.matrix(tfidf_samuel_test))
samuel_words  <- names(samuel_DF)
```

```r
samuel_words_test  <- names(samuel_DF_test)
remove_samuel <- samuel_words_test[!(samuel_words_test %in% samuel_words)]
samuel_DF_test <- samuel_DF_test[, !colnames(samuel_DF_test) %in%
remove_samuel]

### Next Author

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',SarahDavison,'/*.txt'))

sarah_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(sarah_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(sarah_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("usersmachuonedrivedocumentsgithubstadatareuterscctrainsarahcowellnewsmltx
t"))
```

```r
## create a doc-term-matrix from the corpus
DTM_sarah_test = DocumentTermMatrix(my_documents)

DTM_sarah_test = removeSparseTerms(DTM_sarah_test, 0.95)
#DTM_sarah_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_sarah_test = weightTfIdf(DTM_sarah_test)
sarah_DF <- as.data.frame(as.matrix(tfidf_sarah))
sarah_DF_test <- as.data.frame(as.matrix(tfidf_sarah_test))
sarah_words  <- names(sarah_DF)
sarah_words_test  <- names(sarah_DF_test)
remove_sarah <- sarah_words_test[!(sarah_words_test %in% sarah_words)]
sarah_DF_test <- sarah_DF_test[, !colnames(sarah_DF_test) %in% remove_sarah]



### Next Author
file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',ScottHillis,'/*.txt'))

scott_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(scott_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(scott_test))
```

```
## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainscottcowellnewsmltx
t"))
## create a doc-term-matrix from the corpus
DTM_scott_test = DocumentTermMatrix(my_documents)

DTM_scott_test = removeSparseTerms(DTM_scott_test, 0.95)
#DTM_scott_test # now ~ 1000 terms (versus ~3000 before)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_scott_test = weightTfIdf(DTM_scott_test)
scott_DF <- as.data.frame(as.matrix(tfidf_scott))
scott_DF_test <- as.data.frame(as.matrix(tfidf_scott_test))
scott_words  <- names(scott_DF)
scott_words_test  <- names(scott_DF_test)
remove_scott <- scott_words_test[!(scott_words_test %in% scott_words)]
scott_DF_test <- scott_DF_test[, !colnames(scott_DF_test) %in% remove_scott]

### Next Author

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',TanEeLyn,'/*.txt'))

tan_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
```

```
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(tan_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(tan_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctraintancowellnewsmltxt"
))


## create a doc-term-matrix from the corpus
DTM_tan_test = DocumentTermMatrix(my_documents)

DTM_tan_test = removeSparseTerms(DTM_tan_test, 0.95)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_tan_test = weightTfIdf(DTM_tan_test)
tan_DF <- as.data.frame(as.matrix(tfidf_tan))
tan_DF_test <- as.data.frame(as.matrix(tfidf_tan_test))
tan_words  <- names(tan_DF)
tan_words_test  <- names(tan_DF_test)
remove_tan <- tan_words_test[!(tan_words_test %in% tan_words)]
tan_DF_test <- tan_DF_test[, !colnames(tan_DF_test) %in% remove_tan]

### Next Author

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',TheresePoletti,'/*.txt'))
```

```
therese_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(therese_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(therese_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space


my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctraintheresecowellnewsml
txt"))


## create a doc-term-matrix from the corpus
DTM_therese_test = DocumentTermMatrix(my_documents)

DTM_therese_test = removeSparseTerms(DTM_therese_test, 0.95)


# construct TF IDF weights -- might be useful if we wanted to use these
```

```r
# as features in a predictive model
tfidf_therese_test = weightTfIdf(DTM_therese_test)
therese_DF <- as.data.frame(as.matrix(tfidf_therese))
therese_DF_test <- as.data.frame(as.matrix(tfidf_therese_test))
therese_words  <- names(therese_DF)
therese_words_test  <- names(therese_DF_test)
remove_therese <- therese_words_test[!(therese_words_test %in%
therese_words)]
therese_DF_test <- therese_DF_test[, !colnames(therese_DF_test) %in%
remove_therese]


### Next Author

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',TimFarrand,'/*.txt'))

tim_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(tim_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(tim_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
```

```
white-space

my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctraintimcowellnewsmltxt"
))


## create a doc-term-matrix from the corpus
DTM_tim_test = DocumentTermMatrix(my_documents)

DTM_tim_test = removeSparseTerms(DTM_tim_test, 0.95)
#DTM_tim_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_tim_test = weightTfIdf(DTM_tim_test)
tim_DF <- as.data.frame(as.matrix(tfidf_tim))
tim_DF_test <- as.data.frame(as.matrix(tfidf_tim_test))
tim_words  <- names(tim_DF)
tim_words_test  <- names(tim_DF_test)
remove_tim <- tim_words_test[!(tim_words_test %in% tim_words)]
tim_DF_test <- tim_DF_test[, !colnames(tim_DF_test) %in% remove_tim]


### Next Author

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',ToddNissen,'/*.txt'))

todd_test = lapply(file_list_test, readerPlain)


# The file names are ugly...
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
```

```
#mynames
names(todd_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(todd_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

# let's just use the "basic English" stop words
my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctraintoddcowellnewsmltxt
"))


## create a doc-term-matrix from the corpus
DTM_todd_test = DocumentTermMatrix(my_documents)
## Probably a bit stringent here... but only 50 docs!
DTM_todd_test = removeSparseTerms(DTM_todd_test, 0.95)
#DTM_todd_test # now ~ 1000 terms (versus ~3000 before)


# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_todd_test = weightTfIdf(DTM_todd_test)
todd_DF <- as.data.frame(as.matrix(tfidf_todd))
todd_DF_test <- as.data.frame(as.matrix(tfidf_todd_test))
todd_words  <- names(todd_DF)
todd_words_test  <- names(todd_DF_test)
remove_todd <- todd_words_test[!(todd_words_test %in% todd_words)]
todd_DF_test <- todd_DF_test[, !colnames(todd_DF_test) %in% remove_todd]

### Next Author

file_list_test =
Sys.glob(p('C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/ReutersC50/C
50test/',WilliamKazer,'/*.txt'))

will_test = lapply(file_list_test, readerPlain)
```

```r
# The file names are ugly...
#file_list_test

# Clean up the file names
# no doubt the stringr library would be nicer here.
# this is just what I hacked together
mynames = file_list_test %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
  { lapply(., paste0, collapse = '') } %>%
  unlist

# Rename the articles
#mynames
names(will_test) = mynames

## once you have documents in a vector, you
## create a text mining 'corpus' with:
documents_raw = Corpus(VectorSource(will_test))

## Some pre-processing/tokenization steps.
## tm_map just maps some function to every document in the corpus
my_documents = documents_raw %>%
  tm_map(content_transformer(tolower))  %>%             # make everything
lowercase
  tm_map(content_transformer(removeNumbers)) %>%        # remove numbers
  tm_map(content_transformer(removePunctuation)) %>%    # remove punctuation
  tm_map(content_transformer(stripWhitespace))          # remove excess
white-space

my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainwillcowellnewsmltxt
"))
## create a doc-term-matrix from the corpus
DTM_will_test = DocumentTermMatrix(my_documents)

# construct TF IDF weights -- might be useful if we wanted to use these
# as features in a predictive model
tfidf_will_test = weightTfIdf(DTM_will_test)
will_DF <- as.data.frame(as.matrix(tfidf_will))
will_DF_test <- as.data.frame(as.matrix(tfidf_will_test))
will_words  <- names(will_DF)
will_words_test  <- names(will_DF_test)
remove_will <- will_words_test[!(will_words_test %in% will_words)]
will_DF_test <- will_DF_test[, !colnames(will_DF_test) %in% remove_will]
```

Question 6

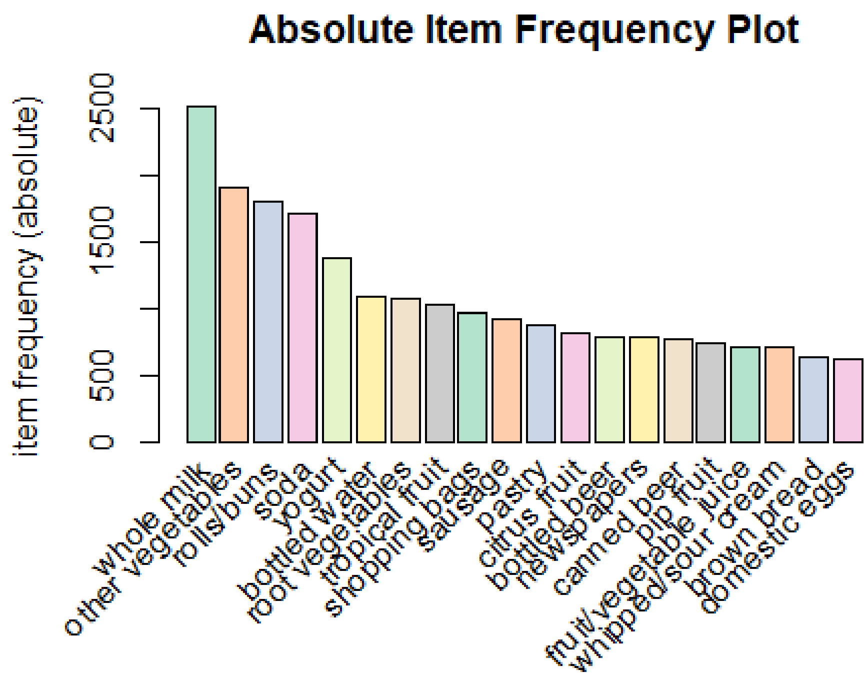
First, I took a look at the most frequent items to ground myself in the data. It also helped in downstream analysis when analyzing the association rules of basket items. As you can see, whole milk is predominately purchased the most in the dataset of basket items.

```
library(dplyr)
library(arules)
library(reshape)
library(tidyverse)
library(arules)  # has a big ecosystem of packages built around it
library(arulesViz)
baskets <-
read.delim("C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/groceries.tx
t",header = FALSE)

baskets <- cbind(Baskets = rownames(baskets), baskets)
rownames(baskets) <- 1:nrow(baskets)

library(splitstackshape)
nbaskets <- concat.split(baskets, "V1", ",")
# baskets1= split(x=baskets_1[,-1], f=baskets$Baskets)





mbaskets <- nbaskets[,-2]

mbaskets <- melt(mbaskets, id=c("Baskets"))
mbaskets <- mbaskets[,-2]

mbaskets$Baskets = factor(mbaskets$Baskets )
mbaskets <- na.omit(mbaskets)


baskets1= split(x=mbaskets$value, f=mbaskets$Baskets)
baskets = lapply(baskets1, unique)
baskettrans = as(baskets, "transactions")
summary(baskettrans)
```

```
## transactions as itemMatrix in sparse format with
##  9835 rows (elements/itemsets/transactions) and
##  169 columns (items) and a density of 0.02609146
##
## most frequent items:
##       whole milk other vegetables       rolls/buns             soda
```

```
##             2513             1903             1809             1715
##           yogurt          (Other)
##             1372            34055
##
## element (itemset/transaction) length distribution:
## sizes
##    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
16
## 2159 1643 1299 1005  855  645  545  438  350  246  182  117   78   77   55
46
##   17   18   19   20   21   22   23   24   26   27   28   29   32
##   29   14   14    9   11    4    6    1    1    1    1    3    1
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   2.000   3.000   4.409   6.000  32.000
##
## includes extended item information - examples:
##             labels
## 1 abrasive cleaner
## 2 artif. sweetener
## 3   baby cosmetics
##
## includes extended transaction information - examples:
##   transactionID
## 1             1
## 2            10
## 3           100
```

```
require("RColorBrewer")
itemFrequencyPlot(baskettrans,topN=20,type="absolute",col=brewer.pal(8,'Paste
l2'), main="Absolute Item Frequency Plot")
```



Prior to running the apriori algorithm, I wanted some way to determine which thresholds I should as parameters in the algorithm. I figured out away to plot the number of rules by confidence level at different support levels. Based on the plot, I determine that the thresholds should be .005 and .2 as this would provide me with enough rules to analyze. In addition, I also found that that the lift is negatively effected as you increase the support and confidence level.

```
library(dplyr)
library(arules)
library(reshape)
library(tidyverse)
library(arules)  # has a big ecosystem of packages built around it
library(arulesViz)
baskets <-
read.delim("C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/groceries.tx
t",header = FALSE)

baskets <- cbind(Baskets = rownames(baskets), baskets)
rownames(baskets) <- 1:nrow(baskets)

library(splitstackshape)
nbaskets <- concat.split(baskets, "V1", ",")
# baskets1= split(x=baskets_1[,-1], f=baskets$Baskets)
```

```
mbaskets <- nbaskets[,-2]

mbaskets <- melt(mbaskets, id=c("Baskets"))
mbaskets <- mbaskets[,-2]

mbaskets$Baskets = factor(mbaskets$Baskets )
mbaskets <- na.omit(mbaskets)


baskets1= split(x=mbaskets$value, f=mbaskets$Baskets)
baskets = lapply(baskets1, unique)
baskettrans = as(baskets, "transactions")
summary(baskettrans)

## transactions as itemMatrix in sparse format with
##  9835 rows (elements/itemsets/transactions) and
##  169 columns (items) and a density of 0.02609146
##
## most frequent items:
##       whole milk other vegetables       rolls/buns             soda
##             2513             1903             1809             1715
##           yogurt          (Other)
##             1372            34055
##
## element (itemset/transaction) length distribution:
## sizes
##    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
16
## 2159 1643 1299 1005  855  645  545  438  350  246  182  117   78   77   55
46
##   17   18   19   20   21   22   23   24   26   27   28   29   32
##   29   14   14    9   11    4    6    1    1    1    1    3    1
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   2.000   3.000   4.409   6.000  32.000
##
## includes extended item information - examples:
##            labels
## 1 abrasive cleaner
## 2 artif. sweetener
## 3   baby cosmetics
##
## includes extended transaction information - examples:
##   transactionID
## 1             1
## 2            10
## 3           100
```

```
supportLevels <- c(0.1, 0.05, 0.01, 0.005)
confidenceLevels <- c(0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1)

# Empty integers
rules_sup10 <- integer(length=9)
rules_sup5 <- integer(length=9)
rules_sup1 <- integer(length=9)
rules_sup0.5 <- integer(length=9)

# Apriori algorithm with a support level of 10%
for (i in 1:length(confidenceLevels)) {

  rules_sup10[i] <- length(apriori(baskettrans,
parameter=list(sup=supportLevels[1],

conf=confidenceLevels[i], target="rules")))

}

## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.9    0.1    1 none FALSE            TRUE       5     0.1      1
##  maxlen target   ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 983
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [8 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [0 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.8    0.1    1 none FALSE            TRUE       5     0.1      1
##  maxlen target   ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
```

```
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 983
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [8 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [0 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.7    0.1    1 none FALSE            TRUE       5     0.1      1
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 983
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [8 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [0 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.6    0.1    1 none FALSE            TRUE       5     0.1      1
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 983
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [8 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
```

```
## checking subsets of size 1 2 done [0.00s].
## writing ... [0 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.5    0.1    1 none FALSE            TRUE       5     0.1      1
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 983
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [8 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [0 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.4    0.1    1 none FALSE            TRUE       5     0.1      1
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 983
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [8 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [0 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.3    0.1    1 none FALSE            TRUE       5     0.1      1
```

```
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 983
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [8 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [0 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.2    0.1    1 none FALSE            TRUE       5     0.1      1
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 983
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [8 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [1 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.1    0.1    1 none FALSE            TRUE       5     0.1      1
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 983
```

```
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [8 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [8 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
# Apriori algorithm with a support level of 5%
for (i in 1:length(confidenceLevels)){

  rules_sup5[i] <- length(apriori(baskettrans,
parameter=list(sup=supportLevels[2],

conf=confidenceLevels[i], target="rules")))

}
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.9    0.1    1 none FALSE            TRUE       5    0.05      1
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 491
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [28 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [0 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.8    0.1    1 none FALSE            TRUE       5    0.05      1
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
```

```
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 491
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [28 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [0 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.7    0.1    1 none FALSE            TRUE       5    0.05      1
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 491
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [28 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [0 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.6    0.1    1 none FALSE            TRUE       5    0.05      1
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 491
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [28 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
```

```
## checking subsets of size 1 2 done [0.00s].
## writing ... [0 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.5    0.1    1 none FALSE            TRUE       5    0.05      1
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 491
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [28 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [0 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.4    0.1    1 none FALSE            TRUE       5    0.05      1
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 491
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [28 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [1 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.3    0.1    1 none FALSE            TRUE       5    0.05      1
```

```
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 491
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [28 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [3 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.2    0.1    1 none FALSE            TRUE       5    0.05      1
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 491
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [28 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [7 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.1    0.1    1 none FALSE            TRUE       5    0.05      1
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 491
```

```
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [28 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [14 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
for (i in 1:length(confidenceLevels)){

  rules_sup0.5[i] <- length(apriori(baskettrans,
parameter=list(sup=supportLevels[4],

conf=confidenceLevels[i], target="rules")))

}
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.9    0.1    1 none FALSE            TRUE       5   0.005      1
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 49
##
## 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 ... [0 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.8    0.1    1 none FALSE            TRUE       5   0.005      1
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
```

```
##
## Absolute minimum support count: 49
##
## 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 ... [0 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.7    0.1    1 none FALSE            TRUE       5   0.005      1
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 49
##
## 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 ... [1 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.6    0.1    1 none FALSE            TRUE       5   0.005      1
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 49
##
## 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 ... [22 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.5    0.1    1 none FALSE            TRUE       5   0.005      1
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 49
##
## 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 ... [120 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.4    0.1    1 none FALSE            TRUE       5   0.005      1
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 49
##
## 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 ... [270 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.3    0.1    1 none FALSE            TRUE       5   0.005      1
##  maxlen target  ext
```

```
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 49
##
## 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].
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.2    0.1    1 none FALSE            TRUE       5   0.005      1
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 49
##
## 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 ... [873 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.1    0.1    1 none FALSE            TRUE       5   0.005      1
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 49
##
```

```
## 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 ... [1582 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```r
# Apriori algorithm with a support level of 1%
for (i in 1:length(confidenceLevels)){

  rules_sup1[i] <- length(apriori(baskettrans,
parameter=list(sup=supportLevels[3],

conf=confidenceLevels[i], target="rules")))
}
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.9    0.1    1 none FALSE            TRUE       5    0.01      1
##  maxlen target  ext
##      10  rules TRUE
##
## 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 ... [0 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.8    0.1    1 none FALSE            TRUE       5    0.01      1
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
```

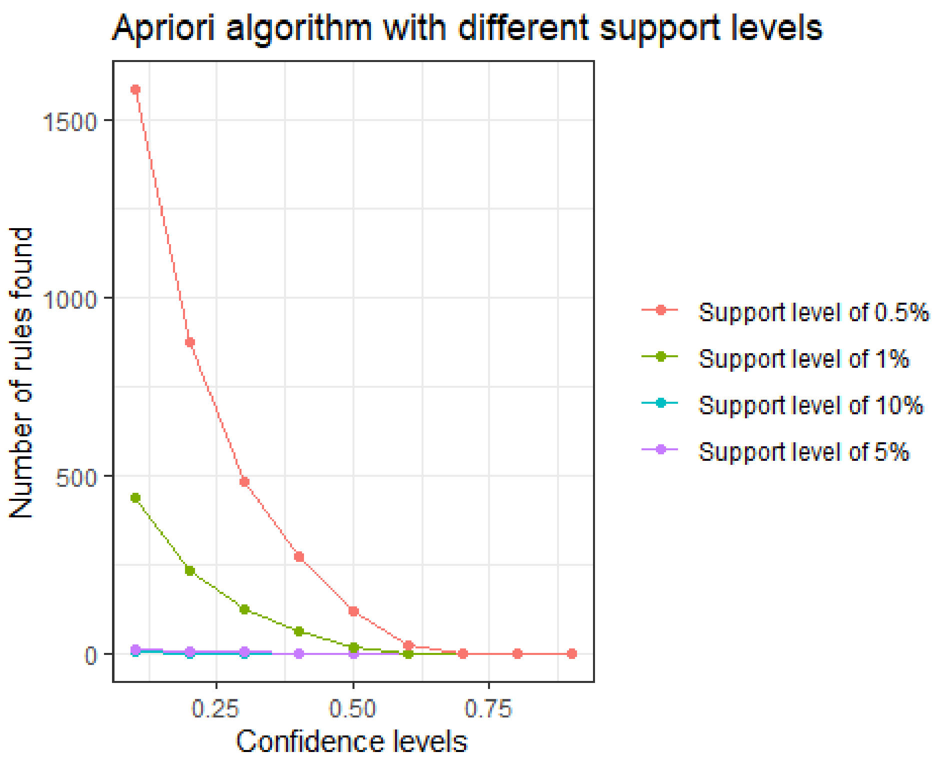
```
## Absolute minimum support count: 98
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [88 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [0 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.7    0.1    1 none FALSE            TRUE       5    0.01      1
##  maxlen target  ext
##      10  rules TRUE
##
## 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 ... [0 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.6    0.1    1 none FALSE            TRUE       5    0.01      1
##  maxlen target  ext
##      10  rules TRUE
##
## 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 ... [0 rule(s)] done [0.00s].
```

```
## creating S4 object  ... done [0.00s].
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.5    0.1    1 none FALSE            TRUE       5    0.01      1
##  maxlen target  ext
##      10  rules TRUE
##
## 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 ... [15 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.4    0.1    1 none FALSE            TRUE       5    0.01      1
##  maxlen target  ext
##      10  rules TRUE
##
## 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 ... [62 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.3    0.1    1 none FALSE            TRUE       5    0.01      1
##  maxlen target  ext
##      10  rules TRUE
```

```
##
## 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 ... [125 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.2    0.1    1 none FALSE            TRUE       5    0.01      1
##  maxlen target  ext
##      10  rules TRUE
##
## 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 ... [232 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.1    0.1    1 none FALSE            TRUE       5    0.01      1
##  maxlen target  ext
##      10  rules TRUE
##
## 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 ... [435 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
# Data frame
num_rules <- data.frame(rules_sup10, rules_sup5, rules_sup1, rules_sup0.5,
confidenceLevels)
library(ggplot2)
# Number of rules found with a support level of 10%, 5%, 1% and 0.5%
ggplot(data=num_rules, aes(x=confidenceLevels)) +

  # Plot line and points (support level of 10%)
  geom_line(aes(y=rules_sup10, colour="Support level of 10%")) +
  geom_point(aes(y=rules_sup10, colour="Support level of 10%")) +

  # Plot line and points (support level of 5%)
  geom_line(aes(y=rules_sup5, colour="Support level of 5%")) +
  geom_point(aes(y=rules_sup5, colour="Support level of 5%")) +

  # Plot line and points (support level of 1%)
  geom_line(aes(y=rules_sup1, colour="Support level of 1%")) +
  geom_point(aes(y=rules_sup1, colour="Support level of 1%")) +

  # Plot line and points (support level of 0.5%)
  geom_line(aes(y=rules_sup0.5, colour="Support level of 0.5%")) +
  geom_point(aes(y=rules_sup0.5, colour="Support level of 0.5%")) +

  # Labs and theme
  labs(x="Confidence levels", y="Number of rules found",
       title="Apriori algorithm with different support levels") +
  theme_bw() +
  theme(legend.title=element_blank())
```



Using the thresholds specified, I got 873 rules. This quite high, but this is the number of rules I wanted in order to find the associate rules with higher lifts.

```
library(dplyr)
library(arules)
library(reshape)
library(tidyverse)
library(arules)  # has a big ecosystem of packages built around it
library(arulesViz)
baskets <-
read.delim("C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/groceries.tx
t",header = FALSE)

baskets <- cbind(Baskets = rownames(baskets), baskets)
rownames(baskets) <- 1:nrow(baskets)

library(splitstackshape)
nbaskets <- concat.split(baskets, "V1", ",")
# baskets1= split(x=baskets_1[,-1], f=baskets$Baskets)




mbaskets <- nbaskets[,-2]

mbaskets <- melt(mbaskets, id=c("Baskets"))
```

```r
mbaskets <- mbaskets[,-2]

mbaskets$Baskets = factor(mbaskets$Baskets )
mbaskets <- na.omit(mbaskets)


baskets1= split(x=mbaskets$value, f=mbaskets$Baskets)
baskets = lapply(baskets1, unique)
baskettrans = as(baskets, "transactions")
basketrules = apriori(baskettrans,
                      parameter=list(support=.005, confidence=.2, maxlen=5))

## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.2    0.1    1 none FALSE            TRUE       5   0.005      1
##  maxlen target  ext
##       5  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 49
##
## 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 ... [873 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].

inspect(basketrules)

##       lhs                          rhs                          support
confidence    coverage      lift count
## [1]   {}                        => {whole milk}             0.255516014
0.2555160 1.000000000 1.0000000  2513
## [2]   {cake bar}                => {whole milk}             0.005592272
0.4230769 0.013218099 1.6557746    55
## [3]   {dishes}                  => {other vegetables}       0.005998983
0.3410405 0.017590239 1.7625502    59
## [4]   {dishes}                  => {whole milk}             0.005287239
0.3005780 0.017590239 1.1763569    52
## [5]   {mustard}                 => {whole milk}             0.005185562
0.4322034 0.011997966 1.6914924    51
## [6]   {pot plants}              => {whole milk}             0.006914082
0.4000000 0.017285206 1.5654596    68
```

```
## [7]   {chewing gum}             => {soda}                 0.005388917
0.2560386 0.021047280 1.4683033    53
## [8]   {chewing gum}             => {whole milk}           0.005083884
0.2415459 0.021047280 0.9453259    50
## [9]   {canned fish}             => {other vegetables}     0.005083884
0.3378378 0.015048297 1.7459985    50
## [10]  {pasta}                   => {whole milk}           0.006100661
0.4054054 0.015048297 1.5866145    60
## [11]  {herbs}                   => {root vegetables}      0.007015760
0.4312500 0.016268429 3.9564774    69
## [12]  {herbs}                   => {other vegetables}     0.007727504
0.4750000 0.016268429 2.4548739    76
## [13]  {herbs}                   => {whole milk}           0.007727504
0.4750000 0.016268429 1.8589833    76
## [14]  {processed cheese}        => {soda}                 0.005287239
0.3190184 0.016573462 1.8294729    52
## [15]  {processed cheese}        => {other vegetables}     0.005490595
0.3312883 0.016573462 1.7121497    54
## [16]  {processed cheese}        => {whole milk}           0.007015760
0.4233129 0.016573462 1.6566981    69
## [17]  {semi-finished bread}     => {other vegetables}     0.005185562
0.2931034 0.017691917 1.5148042    51
## [18]  {semi-finished bread}     => {whole milk}           0.007117438
0.4022989 0.017691917 1.5744565    70
## [19]  {beverages}               => {yogurt}               0.005490595
0.2109375 0.026029487 1.5120775    54
## [20]  {beverages}               => {rolls/buns}           0.005388917
0.2070312 0.026029487 1.1255679    53
## [21]  {beverages}               => {whole milk}           0.006812405
0.2617188 0.026029487 1.0242753    67
## [22]  {ice cream}               => {soda}                 0.006100661
0.2439024 0.025012710 1.3987058    60
## [23]  {ice cream}               => {other vegetables}     0.005083884
0.2032520 0.025012710 1.0504381    50
## [24]  {ice cream}               => {whole milk}           0.005897306
0.2357724 0.025012710 0.9227303    58
## [25]  {detergent}               => {other vegetables}     0.006405694
0.3333333 0.019217082 1.7227185    63
## [26]  {detergent}               => {whole milk}           0.008947636
0.4656085 0.019217082 1.8222281    88
## [27]  {pickled vegetables}      => {other vegetables}     0.006405694
0.3579545 0.017895272 1.8499648    63
## [28]  {pickled vegetables}      => {whole milk}           0.007117438
0.3977273 0.017895272 1.5565650    70
## [29]  {baking powder}           => {other vegetables}     0.007320793
0.4137931 0.017691917 2.1385471    72
## [30]  {baking powder}           => {whole milk}           0.009252669
0.5229885 0.017691917 2.0467935    91
## [31]  {flour}                   => {other vegetables}     0.006304016
0.3625731 0.017386884 1.8738342    62
```

```
## [32]  {flour}                          => {whole milk}           0.008439248
0.4853801 0.017386884 1.8996074    83
## [33]  {soft cheese}                    => {yogurt}               0.005998983
0.3511905 0.017081851 2.5174623    59
## [34]  {soft cheese}                    => {rolls/buns}           0.005388917
0.3154762 0.017081851 1.7151511    53
## [35]  {soft cheese}                    => {other vegetables}     0.007117438
0.4166667 0.017081851 2.1533981    70
## [36]  {soft cheese}                    => {whole milk}           0.007524148
0.4404762 0.017081851 1.7238692    74
## [37]  {specialty bar}                  => {soda}                 0.007219115
0.2639405 0.027351296 1.5136181    71
## [38]  {specialty bar}                  => {rolls/buns}           0.005592272
0.2044610 0.027351296 1.1115940    55
## [39]  {specialty bar}                  => {other vegetables}     0.005592272
0.2044610 0.027351296 1.0566861    55
## [40]  {specialty bar}                  => {whole milk}           0.006507372
0.2379182 0.027351296 0.9311284    64
## [41]  {misc. beverages}                => {soda}                 0.007320793
0.2580645 0.028368073 1.4799210    72
## [42]  {misc. beverages}                => {whole milk}           0.007015760
0.2473118 0.028368073 0.9678917    69
## [43]  {grapes}                         => {tropical fruit}       0.006100661
0.2727273 0.022369090 2.5991015    60
## [44]  {grapes}                         => {other vegetables}     0.009049314
0.4045455 0.022369090 2.0907538    89
## [45]  {grapes}                         => {whole milk}           0.007320793
0.3272727 0.022369090 1.2808306    72
## [46]  {cat food}                       => {yogurt}               0.006202339
0.2663755 0.023284189 1.9094778    61
## [47]  {cat food}                       => {other vegetables}     0.006507372
0.2794760 0.023284189 1.4443753    64
## [48]  {cat food}                       => {whole milk}           0.008845958
0.3799127 0.023284189 1.4868448    87
## [49]  {specialty chocolate}            => {soda}                 0.006304016
0.2073579 0.030401627 1.1891338    62
## [50]  {specialty chocolate}            => {other vegetables}     0.006100661
0.2006689 0.030401627 1.0370881    60
## [51]  {specialty chocolate}            => {whole milk}           0.008032537
0.2642140 0.030401627 1.0340410    79
## [52]  {meat}                           => {sausage}              0.005287239
0.2047244 0.025826131 2.1790742    52
## [53]  {meat}                           => {soda}                 0.005490595
0.2125984 0.025826131 1.2191869    54
## [54]  {meat}                           => {yogurt}               0.005287239
0.2047244 0.025826131 1.4675398    52
## [55]  {meat}                           => {rolls/buns}           0.006914082
0.2677165 0.025826131 1.4554959    68
## [56]  {meat}                           => {other vegetables}     0.009964413
0.3858268 0.025826131 1.9940128    98
```

```
## [57]  {meat}                          => {whole milk}          0.009964413
0.3858268 0.025826131 1.5099906    98
## [58]  {frozen meals}                  => {soda}                0.006202339
0.2186380 0.028368073 1.2538220    61
## [59]  {frozen meals}                  => {yogurt}              0.006202339
0.2186380 0.028368073 1.5672774    61
## [60]  {frozen meals}                  => {other vegetables}    0.007524148
0.2652330 0.028368073 1.3707653    74
## [61]  {frozen meals}                  => {whole milk}          0.009862735
0.3476703 0.028368073 1.3606593    97
## [62]  {hard cheese}                   => {sausage}             0.005185562
0.2116183 0.024504321 2.2524519    51
## [63]  {hard cheese}                   => {root vegetables}     0.005592272
0.2282158 0.024504321 2.0937519    55
## [64]  {hard cheese}                   => {yogurt}              0.006405694
0.2614108 0.024504321 1.8738886    63
## [65]  {hard cheese}                   => {rolls/buns}          0.005897306
0.2406639 0.024504321 1.3084187    58
## [66]  {hard cheese}                   => {other vegetables}    0.009456024
0.3858921 0.024504321 1.9943505    93
## [67]  {hard cheese}                   => {whole milk}          0.010066090
0.4107884 0.024504321 1.6076815    99
## [68]  {butter milk}                   => {yogurt}              0.008540925
0.3054545 0.027961362 2.1896104    84
## [69]  {butter milk}                   => {rolls/buns}          0.007625826
0.2727273 0.027961362 1.4827378    75
## [70]  {butter milk}                   => {other vegetables}    0.010371124
0.3709091 0.027961362 1.9169159   102
## [71]  {butter milk}                   => {whole milk}          0.011591256
0.4145455 0.027961362 1.6223854   114
## [72]  {candy}                         => {soda}                0.008642603
0.2891156 0.029893238 1.6579897    85
## [73]  {candy}                         => {rolls/buns}          0.007117438
0.2380952 0.029893238 1.2944537    70
## [74]  {candy}                         => {other vegetables}    0.006914082
0.2312925 0.029893238 1.1953557    68
## [75]  {candy}                         => {whole milk}          0.008235892
0.2755102 0.029893238 1.0782502    81
## [76]  {ham}                           => {tropical fruit}      0.005388917
0.2070312 0.026029487 1.9730158    53
## [77]  {ham}                           => {yogurt}              0.006710727
0.2578125 0.026029487 1.8480947    66
## [78]  {ham}                           => {rolls/buns}          0.006914082
0.2656250 0.026029487 1.4441249    68
## [79]  {ham}                           => {other vegetables}    0.009150991
0.3515625 0.026029487 1.8169297    90
## [80]  {ham}                           => {whole milk}          0.011489578
0.4414062 0.026029487 1.7275091   113
## [81]  {sliced cheese}                 => {sausage}             0.007015760
0.2863071 0.024504321 3.0474349    69
```

```
## [82]  {sliced cheese}           => {tropical fruit}        0.005287239
0.2157676 0.024504321 2.0562739    52
## [83]  {sliced cheese}           => {root vegetables}       0.005592272
0.2282158 0.024504321 2.0937519    55
## [84]  {sliced cheese}           => {soda}                  0.005083884
0.2074689 0.024504321 1.1897705    50
## [85]  {sliced cheese}           => {yogurt}                0.008032537
0.3278008 0.024504321 2.3497968    79
## [86]  {sliced cheese}           => {rolls/buns}            0.007625826
0.3112033 0.024504321 1.6919208    75
## [87]  {sliced cheese}           => {other vegetables}      0.009049314
0.3692946 0.024504321 1.9085720    89
## [88]  {sliced cheese}           => {whole milk}            0.010777834
0.4398340 0.024504321 1.7213560   106
## [89]  {UHT-milk}                => {bottled water}         0.007320793
0.2188450 0.033451957 1.9800740    72
## [90]  {UHT-milk}                => {soda}                  0.007625826
0.2279635 0.033451957 1.3073010    75
## [91]  {UHT-milk}                => {yogurt}                0.007422471
0.2218845 0.033451957 1.5905496    73
## [92]  {UHT-milk}                => {other vegetables}      0.008134215
0.2431611 0.033451957 1.2566944    80
## [93]  {oil}                     => {root vegetables}       0.007015760
0.2500000 0.028063040 2.2936101    69
## [94]  {oil}                     => {other vegetables}      0.009964413
0.3550725 0.028063040 1.8350697    98
## [95]  {oil}                     => {whole milk}            0.011286223
0.4021739 0.028063040 1.5739675   111
## [96]  {onions}                  => {root vegetables}       0.009456024
0.3049180 0.031011693 2.7974523    93
## [97]  {onions}                  => {yogurt}                0.007219115
0.2327869 0.031011693 1.6687019    71
## [98]  {onions}                  => {rolls/buns}            0.006812405
0.2196721 0.031011693 1.1942927    67
## [99]  {onions}                  => {other vegetables}      0.014234875
0.4590164 0.031011693 2.3722681   140
## [100] {onions}                  => {whole milk}            0.012099644
0.3901639 0.031011693 1.5269647   119
## [101] {berries}                 => {whipped/sour cream}    0.009049314
0.2721713 0.033248602 3.7968855    89
## [102] {berries}                 => {tropical fruit}        0.006710727
0.2018349 0.033248602 1.9234941    66
## [103] {berries}                 => {soda}                  0.007320793
0.2201835 0.033248602 1.2626849    72
## [104] {berries}                 => {yogurt}                0.010574479
0.3180428 0.033248602 2.2798477   104
## [105] {berries}                 => {other vegetables}      0.010269446
0.3088685 0.033248602 1.5962805   101
## [106] {berries}                 => {whole milk}            0.011794611
0.3547401 0.033248602 1.3883281   116
```

```
## [107] {hamburger meat}          => {rolls/buns}          0.008642603
0.2599388 0.033248602 1.4132109    85
## [108] {hamburger meat}          => {other vegetables}    0.013828165
0.4159021 0.033248602 2.1494470   136
## [109] {hamburger meat}          => {whole milk}          0.014743264
0.4434251 0.033248602 1.7354101   145
## [110] {hygiene articles}        => {tropical fruit}      0.006710727
0.2037037 0.032943569 1.9413042    66
## [111] {hygiene articles}        => {soda}                0.007015760
0.2129630 0.032943569 1.2212774    69
## [112] {hygiene articles}        => {yogurt}              0.007320793
0.2222222 0.032943569 1.5929705    72
## [113] {hygiene articles}        => {other vegetables}    0.009557702
0.2901235 0.032943569 1.4994032    94
## [114] {hygiene articles}        => {whole milk}          0.012811388
0.3888889 0.032943569 1.5219746   126
## [115] {salty snack}             => {soda}                0.009354347
0.2473118 0.037824098 1.4182576    92
## [116] {salty snack}             => {other vegetables}    0.010777834
0.2849462 0.037824098 1.4726465   106
## [117] {salty snack}             => {whole milk}          0.011184545
0.2956989 0.037824098 1.1572618   110
## [118] {sugar}                   => {soda}                0.007320793
0.2162162 0.033858668 1.2399338    72
## [119] {sugar}                   => {yogurt}              0.006914082
0.2042042 0.033858668 1.4638107    68
## [120] {sugar}                   => {rolls/buns}          0.007015760
0.2072072 0.033858668 1.1265245    69
## [121] {sugar}                   => {other vegetables}    0.010777834
0.3183183 0.033858668 1.6451186   106
## [122] {sugar}                   => {whole milk}          0.015048297
0.4444444 0.033858668 1.7393996   148
## [123] {waffles}                 => {soda}                0.009557702
0.2486772 0.038434164 1.4260879    94
## [124] {waffles}                 => {rolls/buns}          0.009150991
0.2380952 0.038434164 1.2944537    90
## [125] {waffles}                 => {other vegetables}    0.010066090
0.2619048 0.038434164 1.3535645    99
## [126] {waffles}                 => {whole milk}          0.012709710
0.3306878 0.038434164 1.2941961   125
## [127] {long life bakery product} => {soda}               0.007625826
0.2038043 0.037417387 1.1687555    75
## [128] {long life bakery product} => {yogurt}             0.008744281
0.2336957 0.037417387 1.6752163    86
## [129] {long life bakery product} => {rolls/buns}         0.007930859
0.2119565 0.037417387 1.1523452    78
## [130] {long life bakery product} => {other vegetables}   0.010676157
0.2853261 0.037417387 1.4746096   105
## [131] {long life bakery product} => {whole milk}         0.013523132
0.3614130 0.037417387 1.4144438   133
```

```
## [132] {dessert}                    => {soda}               0.009862735
0.2657534 0.037112354 1.5240145    97
## [133] {dessert}                    => {yogurt}             0.009862735
0.2657534 0.037112354 1.9050182    97
## [134] {dessert}                    => {other vegetables}   0.011591256
0.3123288 0.037112354 1.6141636   114
## [135] {dessert}                    => {whole milk}         0.013726487
0.3698630 0.037112354 1.4475140   135
## [136] {cream cheese}               => {yogurt}             0.012404677
0.3128205 0.039654296 2.2424123   122
## [137] {cream cheese}               => {rolls/buns}         0.009964413
0.2512821 0.039654296 1.3661465    98
## [138] {cream cheese}               => {other vegetables}   0.013726487
0.3461538 0.039654296 1.7889769   135
## [139] {cream cheese}               => {whole milk}         0.016471784
0.4153846 0.039654296 1.6256696   162
## [140] {chicken}                    => {root vegetables}    0.010879512
0.2535545 0.042907982 2.3262206   107
## [141] {chicken}                    => {rolls/buns}         0.009659380
0.2251185 0.042907982 1.2239029    95
## [142] {chicken}                    => {other vegetables}   0.017895272
0.4170616 0.042907982 2.1554393   176
## [143] {chicken}                    => {whole milk}         0.017590239
0.4099526 0.042907982 1.6044106   173
## [144] {white bread}                => {tropical fruit}     0.008744281
0.2077295 0.042094560 1.9796699    86
## [145] {white bread}                => {soda}               0.010269446
0.2439614 0.042094560 1.3990437   101
## [146] {white bread}                => {yogurt}             0.009049314
0.2149758 0.042094560 1.5410258    89
## [147] {white bread}                => {other vegetables}   0.013726487
0.3260870 0.042094560 1.6852681   135
## [148] {white bread}                => {whole milk}         0.017081851
0.4057971 0.042094560 1.5881474   168
## [149] {chocolate}                  => {soda}               0.013523132
0.2725410 0.049618709 1.5629391   133
## [150] {chocolate}                  => {rolls/buns}         0.011794611
0.2377049 0.049618709 1.2923316   116
## [151] {chocolate}                  => {other vegetables}   0.012709710
0.2561475 0.049618709 1.3238103   125
## [152] {chocolate}                  => {whole milk}         0.016675140
0.3360656 0.049618709 1.3152427   164
## [153] {coffee}                     => {other vegetables}   0.013421454
0.2311734 0.058057956 1.1947400   132
## [154] {coffee}                     => {whole milk}         0.018708693
0.3222417 0.058057956 1.2611408   184
## [155] {frozen vegetables}          => {root vegetables}    0.011591256
0.2410148 0.048093543 2.2111759   114
## [156] {frozen vegetables}          => {yogurt}             0.012404677
0.2579281 0.048093543 1.8489235   122
```

```
## [157] {frozen vegetables}        => {rolls/buns}           0.010167768
0.2114165 0.048093543 1.1494092   100
## [158] {frozen vegetables}        => {other vegetables}     0.017793594
0.3699789 0.048093543 1.9121083   175
## [159] {frozen vegetables}        => {whole milk}           0.020437214
0.4249471 0.048093543 1.6630940   201
## [160] {beef}                     => {root vegetables}      0.017386884
0.3313953 0.052465684 3.0403668   171
## [161] {beef}                     => {yogurt}               0.011692933
0.2228682 0.052465684 1.5976012   115
## [162] {beef}                     => {rolls/buns}           0.013624809
0.2596899 0.052465684 1.4118576   134
## [163] {beef}                     => {other vegetables}     0.019725470
0.3759690 0.052465684 1.9430662   194
## [164] {beef}                     => {whole milk}           0.021250635
0.4050388 0.052465684 1.5851795   209
## [165] {curd}                     => {root vegetables}      0.010879512
0.2041985 0.053279105 1.8734067   107
## [166] {curd}                     => {yogurt}               0.017285206
0.3244275 0.053279105 2.3256154   170
## [167] {curd}                     => {other vegetables}     0.017183528
0.3225191 0.053279105 1.6668288   169
## [168] {curd}                     => {whole milk}           0.026131164
0.4904580 0.053279105 1.9194805   257
## [169] {napkins}                  => {soda}                 0.011997966
0.2291262 0.052364006 1.3139687   118
## [170] {napkins}                  => {yogurt}               0.012302999
0.2349515 0.052364006 1.6842183   121
## [171] {napkins}                  => {rolls/buns}           0.011692933
0.2233010 0.052364006 1.2140216   115
## [172] {napkins}                  => {other vegetables}     0.014438231
0.2757282 0.052364006 1.4250060   142
## [173] {napkins}                  => {whole milk}           0.019725470
0.3766990 0.052364006 1.4742678   194
## [174] {pork}                     => {root vegetables}      0.013624809
0.2363316 0.057651246 2.1682099   134
## [175] {pork}                     => {soda}                 0.011896289
0.2063492 0.057651246 1.1833495   117
## [176] {pork}                     => {other vegetables}     0.021657346
0.3756614 0.057651246 1.9414764   213
## [177] {pork}                     => {whole milk}           0.022165735
0.3844797 0.057651246 1.5047187   218
## [178] {frankfurter}              => {rolls/buns}           0.019217082
0.3258621 0.058973055 1.7716161   189
## [179] {frankfurter}              => {other vegetables}     0.016471784
0.2793103 0.058973055 1.4435193   162
## [180] {frankfurter}              => {whole milk}           0.020538892
0.3482759 0.058973055 1.3630295   202
## [181] {bottled beer}             => {soda}                 0.016980173
0.2108586 0.080528724 1.2092094   167
```

```
## [182] {bottled beer}             => {other vegetables}      0.016166751
0.2007576 0.080528724 1.0375464   159
## [183] {bottled beer}             => {whole milk}            0.020437214
0.2537879 0.080528724 0.9932367   201
## [184] {brown bread}              => {yogurt}                0.014539908
0.2241379 0.064870361 1.6067030   143
## [185] {brown bread}              => {other vegetables}      0.018708693
0.2884013 0.064870361 1.4905025   184
## [186] {brown bread}              => {whole milk}            0.025216065
0.3887147 0.064870361 1.5212930   248
## [187] {margarine}                => {yogurt}                0.014234875
0.2430556 0.058566345 1.7423115   140
## [188] {margarine}                => {rolls/buns}            0.014743264
0.2517361 0.058566345 1.3686151   145
## [189] {margarine}                => {other vegetables}      0.019725470
0.3368056 0.058566345 1.7406635   194
## [190] {margarine}                => {whole milk}            0.024199288
0.4131944 0.058566345 1.6170980   238
## [191] {butter}                   => {root vegetables}       0.012913066
0.2330275 0.055414337 2.1378971   127
## [192] {butter}                   => {yogurt}                0.014641586
0.2642202 0.055414337 1.8940273   144
## [193] {butter}                   => {rolls/buns}            0.013421454
0.2422018 0.055414337 1.3167800   132
## [194] {butter}                   => {other vegetables}      0.020030503
0.3614679 0.055414337 1.8681223   197
## [195] {butter}                   => {whole milk}            0.027554652
0.4972477 0.055414337 1.9460530   271
## [196] {newspapers}               => {rolls/buns}            0.019725470
0.2471338 0.079816980 1.3435934   194
## [197] {newspapers}               => {other vegetables}      0.019318760
0.2420382 0.079816980 1.2508912   190
## [198] {newspapers}               => {whole milk}            0.027351296
0.3426752 0.079816980 1.3411103   269
## [199] {domestic eggs}            => {root vegetables}       0.014336553
0.2259615 0.063446873 2.0730706   141
## [200] {domestic eggs}            => {yogurt}                0.014336553
0.2259615 0.063446873 1.6197753   141
## [201] {domestic eggs}            => {rolls/buns}            0.015658363
0.2467949 0.063446873 1.3417510   154
## [202] {domestic eggs}            => {other vegetables}      0.022267412
0.3509615 0.063446873 1.8138238   219
## [203] {domestic eggs}            => {whole milk}            0.029994916
0.4727564 0.063446873 1.8502027   295
## [204] {fruit/vegetable juice}    => {soda}                  0.018403660
0.2545710 0.072292832 1.4598869   181
## [205] {fruit/vegetable juice}    => {yogurt}                0.018708693
0.2587904 0.072292832 1.8551049   184
## [206] {fruit/vegetable juice}    => {rolls/buns}            0.014539908
0.2011252 0.072292832 1.0934583   143
```

```
## [207] {fruit/vegetable juice}    => {other vegetables}      0.021047280
0.2911392 0.072292832 1.5046529   207
## [208] {fruit/vegetable juice}    => {whole milk}            0.026639553
0.3684951 0.072292832 1.4421604   262
## [209] {whipped/sour cream}       => {root vegetables}       0.017081851
0.2382979 0.071682766 2.1862496   168
## [210] {whipped/sour cream}       => {yogurt}                0.020742247
0.2893617 0.071682766 2.0742510   204
## [211] {whipped/sour cream}       => {rolls/buns}            0.014641586
0.2042553 0.071682766 1.1104760   144
## [212] {whipped/sour cream}       => {other vegetables}      0.028876462
0.4028369 0.071682766 2.0819237   284
## [213] {whipped/sour cream}       => {whole milk}            0.032231825
0.4496454 0.071682766 1.7597542   317
## [214] {pip fruit}                => {tropical fruit}        0.020437214
0.2701613 0.075648195 2.5746476   201
## [215] {pip fruit}                => {root vegetables}       0.015556685
0.2056452 0.075648195 1.8866793   153
## [216] {pip fruit}                => {yogurt}                0.017996950
0.2379032 0.075648195 1.7053777   177
## [217] {pip fruit}                => {other vegetables}      0.026131164
0.3454301 0.075648195 1.7852365   257
## [218] {pip fruit}                => {whole milk}            0.030096594
0.3978495 0.075648195 1.5570432   296
## [219] {pastry}                   => {soda}                  0.021047280
0.2365714 0.088967972 1.3566647   207
## [220] {pastry}                   => {rolls/buns}            0.020945602
0.2354286 0.088967972 1.2799558   206
## [221] {pastry}                   => {other vegetables}      0.022572445
0.2537143 0.088967972 1.3112349   222
## [222] {pastry}                   => {whole milk}            0.033248602
0.3737143 0.088967972 1.4625865   327
## [223] {citrus fruit}             => {tropical fruit}        0.019928826
0.2407862 0.082765633 2.2947022   196
## [224] {citrus fruit}             => {root vegetables}       0.017691917
0.2137592 0.082765633 1.9611211   174
## [225] {citrus fruit}             => {yogurt}                0.021657346
0.2616708 0.082765633 1.8757521   213
## [226] {citrus fruit}             => {rolls/buns}            0.016776817
0.2027027 0.082765633 1.1020349   165
## [227] {citrus fruit}             => {other vegetables}      0.028876462
0.3488943 0.082765633 1.8031403   284
## [228] {citrus fruit}             => {whole milk}            0.030503305
0.3685504 0.082765633 1.4423768   300
## [229] {shopping bags}            => {soda}                  0.024605999
0.2497420 0.098525674 1.4321939   242
## [230] {shopping bags}            => {other vegetables}      0.023182511
0.2352941 0.098525674 1.2160366   228
## [231] {shopping bags}            => {whole milk}            0.024504321
0.2487100 0.098525674 0.9733637   241
```

```
## [232] {sausage}                    => {soda}                 0.024300966
0.2586580 0.093950178 1.4833245   239
## [233] {sausage}                    => {yogurt}               0.019623793
0.2088745 0.093950178 1.4972889   193
## [234] {sausage}                    => {rolls/buns}           0.030604982
0.3257576 0.093950178 1.7710480   301
## [235] {sausage}                    => {other vegetables}     0.026944586
0.2867965 0.093950178 1.4822091   265
## [236] {sausage}                    => {whole milk}           0.029893238
0.3181818 0.093950178 1.2452520   294
## [237] {bottled water}              => {soda}                 0.028978139
0.2621895 0.110523640 1.5035766   285
## [238] {bottled water}              => {yogurt}               0.022979156
0.2079117 0.110523640 1.4903873   226
## [239] {bottled water}              => {rolls/buns}           0.024199288
0.2189512 0.110523640 1.1903734   238
## [240] {bottled water}              => {other vegetables}     0.024809354
0.2244710 0.110523640 1.1601012   244
## [241] {bottled water}              => {whole milk}           0.034367056
0.3109476 0.110523640 1.2169396   338
## [242] {tropical fruit}             => {root vegetables}      0.021047280
0.2005814 0.104931368 1.8402220   207
## [243] {tropical fruit}             => {yogurt}               0.029283172
0.2790698 0.104931368 2.0004746   288
## [244] {yogurt}                     => {tropical fruit}       0.029283172
0.2099125 0.139501779 2.0004746   288
## [245] {tropical fruit}             => {rolls/buns}           0.024605999
0.2344961 0.104931368 1.2748863   242
## [246] {tropical fruit}             => {other vegetables}     0.035892222
0.3420543 0.104931368 1.7677896   353
## [247] {tropical fruit}             => {whole milk}           0.042297916
0.4031008 0.104931368 1.5775950   416
## [248] {root vegetables}            => {yogurt}               0.025826131
0.2369403 0.108998475 1.6984751   254
## [249] {root vegetables}            => {rolls/buns}           0.024300966
0.2229478 0.108998475 1.2121013   239
## [250] {root vegetables}            => {other vegetables}     0.047381800
0.4347015 0.108998475 2.2466049   466
## [251] {other vegetables}           => {root vegetables}      0.047381800
0.2448765 0.193492628 2.2466049   466
## [252] {root vegetables}            => {whole milk}           0.048906965
0.4486940 0.108998475 1.7560310   481
## [253] {soda}                       => {rolls/buns}           0.038332486
0.2198251 0.174377224 1.1951242   377
## [254] {rolls/buns}                 => {soda}                 0.038332486
0.2084024 0.183934926 1.1951242   377
## [255] {soda}                       => {whole milk}           0.040061007
0.2297376 0.174377224 0.8991124   394
## [256] {yogurt}                     => {rolls/buns}           0.034367056
0.2463557 0.139501779 1.3393633   338
```

```
## [257] {yogurt}                    => {other vegetables}      0.043416370
0.3112245 0.139501779 1.6084566   427
## [258] {other vegetables}          => {yogurt}                0.043416370
0.2243826 0.193492628 1.6084566   427
## [259] {yogurt}                    => {whole milk}            0.056024403
0.4016035 0.139501779 1.5717351   551
## [260] {whole milk}                => {yogurt}                0.056024403
0.2192598 0.255516014 1.5717351   551
## [261] {rolls/buns}                => {other vegetables}      0.042602949
0.2316197 0.183934926 1.1970465   419
## [262] {other vegetables}          => {rolls/buns}            0.042602949
0.2201787 0.193492628 1.1970465   419
## [263] {rolls/buns}                => {whole milk}            0.056634469
0.3079049 0.183934926 1.2050318   557
## [264] {whole milk}                => {rolls/buns}            0.056634469
0.2216474 0.255516014 1.2050318   557
## [265] {other vegetables}          => {whole milk}            0.074834774
0.3867578 0.193492628 1.5136341   736
## [266] {whole milk}                => {other vegetables}      0.074834774
0.2928770 0.255516014 1.5136341   736
## [267] {oil,
##        other vegetables}          => {whole milk}            0.005083884
0.5102041 0.009964413 1.9967597    50
## [268] {oil,
##        whole milk}                => {other vegetables}      0.005083884
0.4504505 0.011286223 2.3279980    50
## [269] {onions,
##        root vegetables}           => {other vegetables}      0.005693950
0.6021505 0.009456024 3.1120076    56
## [270] {onions,
##        other vegetables}          => {root vegetables}       0.005693950
0.4000000 0.014234875 3.6697761    56
## [271] {onions,
##        other vegetables}          => {whole milk}            0.006609049
0.4642857 0.014234875 1.8170513    65
## [272] {onions,
##        whole milk}                => {other vegetables}      0.006609049
0.5462185 0.012099644 2.8229421    65
## [273] {hamburger meat,
##        other vegetables}          => {whole milk}            0.006304016
0.4558824 0.013828165 1.7841635    62
## [274] {hamburger meat,
##        whole milk}                => {other vegetables}      0.006304016
0.4275862 0.014743264 2.2098320    62
## [275] {hygiene articles,
##        other vegetables}          => {whole milk}            0.005185562
0.5425532 0.009557702 2.1233628    51
## [276] {hygiene articles,
##        whole milk}                => {other vegetables}      0.005185562
0.4047619 0.012811388 2.0918725    51
```

```
## [277] {other vegetables,
##        sugar}                    => {whole milk}            0.006304016
0.5849057 0.010777834 2.2891155    62
## [278] {sugar,
##        whole milk}               => {other vegetables}      0.006304016
0.4189189 0.015048297 2.1650381    62
## [279] {long life bakery product,
##        other vegetables}         => {whole milk}            0.005693950
0.5333333 0.010676157 2.0872795    56
## [280] {long life bakery product,
##        whole milk}               => {other vegetables}      0.005693950
0.4210526 0.013523132 2.1760655    56
## [281] {cream cheese,
##        yogurt}                   => {other vegetables}      0.005287239
0.4262295 0.012404677 2.2028204    52
## [282] {cream cheese,
##        other vegetables}         => {yogurt}                0.005287239
0.3851852 0.013726487 2.7611489    52
## [283] {cream cheese,
##        yogurt}                   => {whole milk}            0.006609049
0.5327869 0.012404677 2.0851409    65
## [284] {cream cheese,
##        whole milk}               => {yogurt}                0.006609049
0.4012346 0.016471784 2.8761968    65
## [285] {cream cheese,
##        other vegetables}         => {whole milk}            0.006710727
0.4888889 0.013726487 1.9133395    66
## [286] {cream cheese,
##        whole milk}               => {other vegetables}      0.006710727
0.4074074 0.016471784 2.1055449    66
## [287] {chicken,
##        root vegetables}          => {other vegetables}      0.005693950
0.5233645 0.010879512 2.7048291    56
## [288] {chicken,
##        other vegetables}         => {root vegetables}       0.005693950
0.3181818 0.017895272 2.9191401    56
## [289] {chicken,
##        root vegetables}          => {whole milk}            0.005998983
0.5514019 0.010879512 2.1579934    59
## [290] {chicken,
##        whole milk}               => {root vegetables}       0.005998983
0.3410405 0.017590239 3.1288554    59
## [291] {chicken,
##        rolls/buns}               => {whole milk}            0.005287239
0.5473684 0.009659380 2.1422079    52
## [292] {chicken,
##        whole milk}               => {rolls/buns}            0.005287239
0.3005780 0.017590239 1.6341542    52
## [293] {chicken,
##        other vegetables}         => {whole milk}            0.008439248
```

```
0.4715909 0.017895272 1.8456413    83
## [294] {chicken,
##        whole milk}               => {other vegetables}      0.008439248
0.4797688 0.017590239 2.4795197    83
## [295] {other vegetables,
##        white bread}              => {whole milk}            0.005897306
0.4296296 0.013726487 1.6814196    58
## [296] {white bread,
##        whole milk}               => {other vegetables}      0.005897306
0.3452381 0.017081851 1.7842442    58
## [297] {chocolate,
##        soda}                     => {whole milk}            0.005083884
0.3759398 0.013523132 1.4712966    50
## [298] {chocolate,
##        whole milk}               => {soda}                  0.005083884
0.3048780 0.016675140 1.7483823    50
## [299] {chocolate,
##        other vegetables}         => {whole milk}            0.005490595
0.4320000 0.012709710 1.6906964    54
## [300] {chocolate,
##        whole milk}               => {other vegetables}      0.005490595
0.3292683 0.016675140 1.7017098    54
## [301] {coffee,
##        yogurt}                   => {whole milk}            0.005083884
0.5208333 0.009761057 2.0383589    50
## [302] {coffee,
##        whole milk}               => {yogurt}                0.005083884
0.2717391 0.018708693 1.9479259    50
## [303] {coffee,
##        other vegetables}         => {whole milk}            0.006405694
0.4772727 0.013421454 1.8678779    63
## [304] {coffee,
##        whole milk}               => {other vegetables}      0.006405694
0.3423913 0.018708693 1.7695315    63
## [305] {frozen vegetables,
##        root vegetables}          => {other vegetables}      0.006100661
0.5263158 0.011591256 2.7200819    60
## [306] {frozen vegetables,
##        other vegetables}         => {root vegetables}       0.006100661
0.3428571 0.017793594 3.1455224    60
## [307] {frozen vegetables,
##        root vegetables}          => {whole milk}            0.006202339
0.5350877 0.011591256 2.0941455    61
## [308] {frozen vegetables,
##        whole milk}               => {root vegetables}       0.006202339
0.3034826 0.020437214 2.7842829    61
## [309] {frozen vegetables,
##        yogurt}                   => {other vegetables}      0.005287239
0.4262295 0.012404677 2.2028204    52
## [310] {frozen vegetables,
```

```
##        other vegetables}         => {yogurt}                0.005287239
0.2971429 0.017793594 2.1300292    52
## [311] {frozen vegetables,
##        yogurt}                   => {whole milk}            0.006100661
0.4918033 0.012404677 1.9247454    60
## [312] {frozen vegetables,
##        whole milk}               => {yogurt}                0.006100661
0.2985075 0.020437214 2.1398111    60
## [313] {frozen vegetables,
##        rolls/buns}               => {whole milk}            0.005083884
0.5000000 0.010167768 1.9568245    50
## [314] {frozen vegetables,
##        whole milk}               => {rolls/buns}            0.005083884
0.2487562 0.020437214 1.3524143    50
## [315] {frozen vegetables,
##        other vegetables}         => {whole milk}            0.009659380
0.5428571 0.017793594 2.1245523    95
## [316] {frozen vegetables,
##        whole milk}               => {other vegetables}      0.009659380
0.4726368 0.020437214 2.4426606    95
## [317] {beef,
##        root vegetables}          => {other vegetables}      0.007930859
0.4561404 0.017386884 2.3574043    78
## [318] {beef,
##        other vegetables}         => {root vegetables}       0.007930859
0.4020619 0.019725470 3.6886925    78
## [319] {beef,
##        root vegetables}          => {whole milk}            0.008032537
0.4619883 0.017386884 1.8080601    79
## [320] {beef,
##        whole milk}               => {root vegetables}       0.008032537
0.3779904 0.021250635 3.4678506    79
## [321] {beef,
##        yogurt}                   => {other vegetables}      0.005185562
0.4434783 0.011692933 2.2919646    51
## [322] {beef,
##        other vegetables}         => {yogurt}                0.005185562
0.2628866 0.019725470 1.8844677    51
## [323] {beef,
##        yogurt}                   => {whole milk}            0.006100661
0.5217391 0.011692933 2.0419038    60
## [324] {beef,
##        whole milk}               => {yogurt}                0.006100661
0.2870813 0.021250635 2.0579045    60
## [325] {beef,
##        rolls/buns}               => {other vegetables}      0.005795628
0.4253731 0.013624809 2.1983945    57
## [326] {beef,
##        other vegetables}         => {rolls/buns}            0.005795628
0.2938144 0.019725470 1.5973825    57
```

```
## [327] {beef,
##        rolls/buns}             => {whole milk}           0.006812405
0.5000000 0.013624809 1.9568245    67
## [328] {beef,
##        whole milk}             => {rolls/buns}           0.006812405
0.3205742 0.021250635 1.7428673    67
## [329] {beef,
##        other vegetables}       => {whole milk}           0.009252669
0.4690722 0.019725470 1.8357838    91
## [330] {beef,
##        whole milk}             => {other vegetables}     0.009252669
0.4354067 0.021250635 2.2502495    91
## [331] {curd,
##        whipped/sour cream}     => {whole milk}           0.005897306
0.5631068 0.010472801 2.2038024    58
## [332] {curd,
##        whole milk}             => {whipped/sour cream}   0.005897306
0.2256809 0.026131164 3.1483291    58
## [333] {curd,
##        tropical fruit}         => {yogurt}               0.005287239
0.5148515 0.010269446 3.6906446    52
## [334] {curd,
##        yogurt}                 => {tropical fruit}       0.005287239
0.3058824 0.017285206 2.9150707    52
## [335] {curd,
##        tropical fruit}         => {other vegetables}     0.005287239
0.5148515 0.010269446 2.6608326    52
## [336] {curd,
##        other vegetables}       => {tropical fruit}       0.005287239
0.3076923 0.017183528 2.9323196    52
## [337] {curd,
##        tropical fruit}         => {whole milk}           0.006507372
0.6336634 0.010269446 2.4799360    64
## [338] {curd,
##        whole milk}             => {tropical fruit}       0.006507372
0.2490272 0.026131164 2.3732392    64
## [339] {curd,
##        root vegetables}        => {other vegetables}     0.005490595
0.5046729 0.010879512 2.6082280    54
## [340] {curd,
##        other vegetables}       => {root vegetables}      0.005490595
0.3195266 0.017183528 2.9314780    54
## [341] {curd,
##        root vegetables}        => {whole milk}           0.006202339
0.5700935 0.010879512 2.2311457    61
## [342] {curd,
##        whole milk}             => {root vegetables}      0.006202339
0.2373541 0.026131164 2.1775909    61
## [343] {curd,
##        yogurt}                 => {other vegetables}     0.006100661
```

```
0.3529412 0.017285206 1.8240549    60
## [344] {curd,
##        other vegetables}         => {yogurt}                0.006100661
0.3550296 0.017183528 2.5449825    60
## [345] {curd,
##        yogurt}                   => {whole milk}            0.010066090
0.5823529 0.017285206 2.2791250    99
## [346] {curd,
##        whole milk}               => {yogurt}                0.010066090
0.3852140 0.026131164 2.7613555    99
## [347] {curd,
##        rolls/buns}               => {whole milk}            0.005897306
0.5858586 0.010066090 2.2928449    58
## [348] {curd,
##        whole milk}               => {rolls/buns}            0.005897306
0.2256809 0.026131164 1.2269607    58
## [349] {curd,
##        other vegetables}         => {whole milk}            0.009862735
0.5739645 0.017183528 2.2462956    97
## [350] {curd,
##        whole milk}               => {other vegetables}      0.009862735
0.3774319 0.026131164 1.9506268    97
## [351] {napkins,
##        yogurt}                   => {whole milk}            0.006100661
0.4958678 0.012302999 1.9406524    60
## [352] {napkins,
##        whole milk}               => {yogurt}                0.006100661
0.3092784 0.019725470 2.2170208    60
## [353] {napkins,
##        rolls/buns}               => {whole milk}            0.005287239
0.4521739 0.011692933 1.7696500    52
## [354] {napkins,
##        whole milk}               => {rolls/buns}            0.005287239
0.2680412 0.019725470 1.4572612    52
## [355] {napkins,
##        other vegetables}         => {whole milk}            0.006812405
0.4718310 0.014438231 1.8465809    67
## [356] {napkins,
##        whole milk}               => {other vegetables}      0.006812405
0.3453608 0.019725470 1.7848785    67
## [357] {pork,
##        root vegetables}          => {other vegetables}      0.007015760
0.5149254 0.013624809 2.6612144    69
## [358] {other vegetables,
##        pork}                     => {root vegetables}       0.007015760
0.3239437 0.021657346 2.9720018    69
## [359] {pork,
##        root vegetables}          => {whole milk}            0.006812405
0.5000000 0.013624809 1.9568245    67
## [360] {pork,
```

```
##         whole milk}                => {root vegetables}       0.006812405
0.3073394 0.022165735 2.8196674    67
## [361] {pork,
##         rolls/buns}                => {other vegetables}      0.005592272
0.4954955 0.011286223 2.5607978    55
## [362] {other vegetables,
##         pork}                      => {rolls/buns}            0.005592272
0.2582160 0.021657346 1.4038441    55
## [363] {pork,
##         rolls/buns}                => {whole milk}            0.006202339
0.5495495 0.011286223 2.1507441    61
## [364] {pork,
##         whole milk}                => {rolls/buns}            0.006202339
0.2798165 0.022165735 1.5212799    61
## [365] {other vegetables,
##         pork}                      => {whole milk}            0.010167768
0.4694836 0.021657346 1.8373939   100
## [366] {pork,
##         whole milk}                => {other vegetables}      0.010167768
0.4587156 0.022165735 2.3707136   100
## [367] {frankfurter,
##         tropical fruit}            => {whole milk}            0.005185562
0.5483871 0.009456024 2.1461946    51
## [368] {frankfurter,
##         whole milk}                => {tropical fruit}        0.005185562
0.2524752 0.020538892 2.4060989    51
## [369] {frankfurter,
##         root vegetables}           => {whole milk}            0.005083884
0.5000000 0.010167768 1.9568245    50
## [370] {frankfurter,
##         whole milk}                => {root vegetables}       0.005083884
0.2475248 0.020538892 2.2709011    50
## [371] {frankfurter,
##         yogurt}                    => {whole milk}            0.006202339
0.5545455 0.011184545 2.1702963    61
## [372] {frankfurter,
##         whole milk}                => {yogurt}                0.006202339
0.3019802 0.020538892 2.1647050    61
## [373] {frankfurter,
##         rolls/buns}                => {other vegetables}      0.005592272
0.2910053 0.019217082 1.5039606    55
## [374] {frankfurter,
##         other vegetables}          => {rolls/buns}            0.005592272
0.3395062 0.016471784 1.8457950    55
## [375] {frankfurter,
##         rolls/buns}                => {whole milk}            0.005998983
0.3121693 0.019217082 1.2217211    59
## [376] {frankfurter,
##         whole milk}                => {rolls/buns}            0.005998983
0.2920792 0.020538892 1.5879486    59
```

```
## [377] {frankfurter,
##        other vegetables}          => {whole milk}            0.007625826
0.4629630 0.016471784 1.8118745    75
## [378] {frankfurter,
##        whole milk}                => {other vegetables}      0.007625826
0.3712871 0.020538892 1.9188696    75
## [379] {bottled beer,
##        bottled water}             => {soda}                  0.005083884
0.3225806 0.015760041 1.8499013    50
## [380] {bottled beer,
##        soda}                      => {bottled water}         0.005083884
0.2994012 0.016980173 2.7089336    50
## [381] {bottled beer,
##        bottled water}             => {whole milk}            0.006100661
0.3870968 0.015760041 1.5149609    60
## [382] {bottled beer,
##        whole milk}                => {bottled water}         0.006100661
0.2985075 0.020437214 2.7008472    60
## [383] {bottled beer,
##        yogurt}                    => {whole milk}            0.005185562
0.5604396 0.009252669 2.1933637    51
## [384] {bottled beer,
##        whole milk}                => {yogurt}                0.005185562
0.2537313 0.020437214 1.8188395    51
## [385] {bottled beer,
##        rolls/buns}                => {whole milk}            0.005388917
0.3955224 0.013624809 1.5479358    53
## [386] {bottled beer,
##        whole milk}                => {rolls/buns}            0.005388917
0.2636816 0.020437214 1.4335591    53
## [387] {bottled beer,
##        other vegetables}          => {whole milk}            0.007625826
0.4716981 0.016166751 1.8460609    75
## [388] {bottled beer,
##        whole milk}                => {other vegetables}      0.007625826
0.3731343 0.020437214 1.9284162    75
## [389] {brown bread,
##        tropical fruit}            => {whole milk}            0.005693950
0.5333333 0.010676157 2.0872795    56
## [390] {brown bread,
##        whole milk}                => {tropical fruit}        0.005693950
0.2258065 0.025216065 2.1519442    56
## [391] {brown bread,
##        root vegetables}           => {whole milk}            0.005693950
0.5600000 0.010167768 2.1916435    56
## [392] {brown bread,
##        whole milk}                => {root vegetables}       0.005693950
0.2258065 0.025216065 2.0716478    56
## [393] {brown bread,
##        soda}                      => {whole milk}            0.005083884
```

```
0.4032258 0.012608033 1.5780843    50
## [394] {brown bread,
##        whole milk}                 => {soda}                 0.005083884
0.2016129 0.025216065 1.1561883    50
## [395] {brown bread,
##        yogurt}                     => {other vegetables}     0.005185562
0.3566434 0.014539908 1.8431883    51
## [396] {brown bread,
##        other vegetables}           => {yogurt}               0.005185562
0.2771739 0.018708693 1.9868844    51
## [397] {brown bread,
##        yogurt}                     => {whole milk}           0.007117438
0.4895105 0.014539908 1.9157723    70
## [398] {brown bread,
##        whole milk}                 => {yogurt}               0.007117438
0.2822581 0.025216065 2.0233295    70
## [399] {brown bread,
##        rolls/buns}                 => {whole milk}           0.005287239
0.4193548 0.012608033 1.6412077    52
## [400] {brown bread,
##        whole milk}                 => {rolls/buns}           0.005287239
0.2096774 0.025216065 1.1399544    52
## [401] {brown bread,
##        other vegetables}           => {whole milk}           0.009354347
0.5000000 0.018708693 1.9568245    92
## [402] {brown bread,
##        whole milk}                 => {other vegetables}     0.009354347
0.3709677 0.025216065 1.9172190    92
## [403] {domestic eggs,
##        margarine}                  => {whole milk}           0.005185562
0.6219512 0.008337570 2.4340988    51
## [404] {margarine,
##        whole milk}                 => {domestic eggs}        0.005185562
0.2142857 0.024199288 3.3774038    51
## [405] {margarine,
##        root vegetables}            => {other vegetables}     0.005897306
0.5321101 0.011082867 2.7500277    58
## [406] {margarine,
##        other vegetables}           => {root vegetables}      0.005897306
0.2989691 0.019725470 2.7428739    58
## [407] {margarine,
##        yogurt}                     => {other vegetables}     0.005693950
0.4000000 0.014234875 2.0672622    56
## [408] {margarine,
##        other vegetables}           => {yogurt}               0.005693950
0.2886598 0.019725470 2.0692194    56
## [409] {margarine,
##        yogurt}                     => {whole milk}           0.007015760
0.4928571 0.014234875 1.9288699    69
## [410] {margarine,
```

```
##        whole milk}                => {yogurt}                 0.007015760
0.2899160 0.024199288 2.0782241    69
## [411] {margarine,
##        rolls/buns}                => {other vegetables}       0.005185562
0.3517241 0.014743264 1.8177651    51
## [412] {margarine,
##        other vegetables}          => {rolls/buns}             0.005185562
0.2628866 0.019725470 1.4292370    51
## [413] {margarine,
##        rolls/buns}                => {whole milk}             0.007930859
0.5379310 0.014743264 2.1052733    78
## [414] {margarine,
##        whole milk}                => {rolls/buns}             0.007930859
0.3277311 0.024199288 1.7817774    78
## [415] {margarine,
##        other vegetables}          => {whole milk}             0.009252669
0.4690722 0.019725470 1.8357838    91
## [416] {margarine,
##        whole milk}                => {other vegetables}       0.009252669
0.3823529 0.024199288 1.9760595    91
## [417] {butter,
##        domestic eggs}             => {whole milk}             0.005998983
0.6210526 0.009659380 2.4305820    59
## [418] {butter,
##        whole milk}                => {domestic eggs}          0.005998983
0.2177122 0.027554652 3.4314091    59
## [419] {domestic eggs,
##        whole milk}                => {butter}                 0.005998983
0.2000000 0.029994916 3.6091743    59
## [420] {butter,
##        whipped/sour cream}        => {other vegetables}       0.005795628
0.5700000 0.010167768 2.9458487    57
## [421] {butter,
##        other vegetables}          => {whipped/sour cream}     0.005795628
0.2893401 0.020030503 4.0363970    57
## [422] {other vegetables,
##        whipped/sour cream}        => {butter}                 0.005795628
0.2007042 0.028876462 3.6218827    57
## [423] {butter,
##        whipped/sour cream}        => {whole milk}             0.006710727
0.6600000 0.010167768 2.5830084    66
## [424] {butter,
##        whole milk}                => {whipped/sour cream}     0.006710727
0.2435424 0.027554652 3.3975033    66
## [425] {whipped/sour cream,
##        whole milk}                => {butter}                 0.006710727
0.2082019 0.032231825 3.7571846    66
## [426] {butter,
##        citrus fruit}              => {whole milk}             0.005083884
0.5555556 0.009150991 2.1742495    50
```

```
## [427] {bottled water,
##        butter}                  => {whole milk}            0.005388917
0.6022727 0.008947636 2.3570841    53
## [428] {butter,
##        tropical fruit}          => {other vegetables}      0.005490595
0.5510204 0.009964413 2.8477592    54
## [429] {butter,
##        other vegetables}        => {tropical fruit}        0.005490595
0.2741117 0.020030503 2.6122949    54
## [430] {butter,
##        tropical fruit}          => {whole milk}            0.006202339
0.6224490 0.009964413 2.4360468    61
## [431] {butter,
##        whole milk}              => {tropical fruit}        0.006202339
0.2250923 0.027554652 2.1451379    61
## [432] {butter,
##        root vegetables}         => {other vegetables}      0.006609049
0.5118110 0.012913066 2.6451190    65
## [433] {butter,
##        other vegetables}        => {root vegetables}       0.006609049
0.3299492 0.020030503 3.0270996    65
## [434] {butter,
##        root vegetables}         => {whole milk}            0.008235892
0.6377953 0.012913066 2.4961069    81
## [435] {butter,
##        whole milk}              => {root vegetables}       0.008235892
0.2988930 0.027554652 2.7421759    81
## [436] {butter,
##        yogurt}                  => {other vegetables}      0.006405694
0.4375000 0.014641586 2.2610681    63
## [437] {butter,
##        other vegetables}        => {yogurt}                0.006405694
0.3197970 0.020030503 2.2924220    63
## [438] {butter,
##        yogurt}                  => {whole milk}            0.009354347
0.6388889 0.014641586 2.5003869    92
## [439] {butter,
##        whole milk}              => {yogurt}                0.009354347
0.3394834 0.027554652 2.4335417    92
## [440] {butter,
##        rolls/buns}              => {other vegetables}      0.005693950
0.4242424 0.013421454 2.1925508    56
## [441] {butter,
##        other vegetables}        => {rolls/buns}            0.005693950
0.2842640 0.020030503 1.5454594    56
## [442] {butter,
##        rolls/buns}              => {whole milk}            0.006609049
0.4924242 0.013421454 1.9271757    65
## [443] {butter,
##        whole milk}              => {rolls/buns}            0.006609049
```

```
0.2398524 0.027554652 1.3040068    65
## [444] {butter,
##        other vegetables}         => {whole milk}           0.011489578
0.5736041 0.020030503 2.2448850   113
## [445] {butter,
##        whole milk}               => {other vegetables}     0.011489578
0.4169742 0.027554652 2.1549874   113
## [446] {newspapers,
##        tropical fruit}           => {whole milk}           0.005083884
0.4310345 0.011794611 1.6869177    50
## [447] {newspapers,
##        root vegetables}          => {other vegetables}     0.005998983
0.5221239 0.011489578 2.6984175    59
## [448] {newspapers,
##        other vegetables}         => {root vegetables}      0.005998983
0.3105263 0.019318760 2.8489051    59
## [449] {newspapers,
##        root vegetables}          => {whole milk}           0.005795628
0.5044248 0.011489578 1.9741415    57
## [450] {newspapers,
##        whole milk}               => {root vegetables}      0.005795628
0.2118959 0.027351296 1.9440264    57
## [451] {newspapers,
##        yogurt}                   => {rolls/buns}           0.005083884
0.3311258 0.015353330 1.8002336    50
## [452] {newspapers,
##        rolls/buns}               => {yogurt}               0.005083884
0.2577320 0.019725470 1.8475174    50
## [453] {newspapers,
##        yogurt}                   => {other vegetables}     0.005592272
0.3642384 0.015353330 1.8824408    55
## [454] {newspapers,
##        other vegetables}         => {yogurt}               0.005592272
0.2894737 0.019318760 2.0750537    55
## [455] {newspapers,
##        yogurt}                   => {whole milk}           0.006609049
0.4304636 0.015353330 1.6846834    65
## [456] {newspapers,
##        whole milk}               => {yogurt}               0.006609049
0.2416357 0.027351296 1.7321334    65
## [457] {newspapers,
##        rolls/buns}               => {other vegetables}     0.005490595
0.2783505 0.019725470 1.4385588    54
## [458] {newspapers,
##        other vegetables}         => {rolls/buns}           0.005490595
0.2842105 0.019318760 1.5451689    54
## [459] {newspapers,
##        rolls/buns}               => {whole milk}           0.007625826
0.3865979 0.019725470 1.5130086    75
## [460] {newspapers,
```

```
##        whole milk}                  => {rolls/buns}           0.007625826
0.2788104 0.027351296 1.5158100    75
## [461] {newspapers,
##        other vegetables}            => {whole milk}           0.008337570
0.4315789 0.019318760 1.6890485    82
## [462] {newspapers,
##        whole milk}                  => {other vegetables}     0.008337570
0.3048327 0.027351296 1.5754229    82
## [463] {domestic eggs,
##        whipped/sour cream}          => {other vegetables}     0.005083884
0.5102041 0.009964413 2.6368141    50
## [464] {domestic eggs,
##        other vegetables}            => {whipped/sour cream}   0.005083884
0.2283105 0.022267412 3.1850125    50
## [465] {domestic eggs,
##        whipped/sour cream}          => {whole milk}           0.005693950
0.5714286 0.009964413 2.2363709    56
## [466] {domestic eggs,
##        pip fruit}                   => {whole milk}           0.005388917
0.6235294 0.008642603 2.4402753    53
## [467] {citrus fruit,
##        domestic eggs}               => {whole milk}           0.005693950
0.5490196 0.010371124 2.1486701    56
## [468] {domestic eggs,
##        tropical fruit}              => {whole milk}           0.006914082
0.6071429 0.011387900 2.3761441    68
## [469] {domestic eggs,
##        whole milk}                  => {tropical fruit}       0.006914082
0.2305085 0.029994916 2.1967547    68
## [470] {domestic eggs,
##        root vegetables}             => {other vegetables}     0.007320793
0.5106383 0.014336553 2.6390582    72
## [471] {domestic eggs,
##        other vegetables}            => {root vegetables}      0.007320793
0.3287671 0.022267412 3.0162543    72
## [472] {domestic eggs,
##        root vegetables}             => {whole milk}           0.008540925
0.5957447 0.014336553 2.3315356    84
## [473] {domestic eggs,
##        whole milk}                  => {root vegetables}      0.008540925
0.2847458 0.029994916 2.6123830    84
## [474] {domestic eggs,
##        soda}                        => {other vegetables}     0.005083884
0.4098361 0.012404677 2.1180965    50
## [475] {domestic eggs,
##        other vegetables}            => {soda}                 0.005083884
0.2283105 0.022267412 1.3092908    50
## [476] {domestic eggs,
##        soda}                        => {whole milk}           0.005185562
0.4180328 0.012404677 1.6360336    51
```

```
## [477] {domestic eggs,
##        yogurt}                   => {other vegetables}      0.005795628
0.4042553 0.014336553 2.0892544    57
## [478] {domestic eggs,
##        other vegetables}         => {yogurt}                0.005795628
0.2602740 0.022267412 1.8657394    57
## [479] {domestic eggs,
##        yogurt}                   => {whole milk}            0.007727504
0.5390071 0.014336553 2.1094846    76
## [480] {domestic eggs,
##        whole milk}               => {yogurt}                0.007727504
0.2576271 0.029994916 1.8467658    76
## [481] {domestic eggs,
##        rolls/buns}               => {other vegetables}      0.005897306
0.3766234 0.015658363 1.9464482    58
## [482] {domestic eggs,
##        other vegetables}         => {rolls/buns}            0.005897306
0.2648402 0.022267412 1.4398580    58
## [483] {domestic eggs,
##        rolls/buns}               => {whole milk}            0.006609049
0.4220779 0.015658363 1.6518648    65
## [484] {domestic eggs,
##        whole milk}               => {rolls/buns}            0.006609049
0.2203390 0.029994916 1.1979181    65
## [485] {domestic eggs,
##        other vegetables}         => {whole milk}            0.012302999
0.5525114 0.022267412 2.1623358   121
## [486] {domestic eggs,
##        whole milk}               => {other vegetables}      0.012302999
0.4101695 0.029994916 2.1198197   121
## [487] {bottled water,
##        fruit/vegetable juice}    => {soda}                  0.005185562
0.3642857 0.014234875 2.0890671    51
## [488] {fruit/vegetable juice,
##        soda}                     => {bottled water}         0.005185562
0.2817680 0.018403660 2.5493908    51
## [489] {bottled water,
##        fruit/vegetable juice}    => {whole milk}            0.005795628
0.4071429 0.014234875 1.5934142    57
## [490] {fruit/vegetable juice,
##        whole milk}               => {bottled water}         0.005795628
0.2175573 0.026639553 1.9684228    57
## [491] {fruit/vegetable juice,
##        tropical fruit}           => {other vegetables}      0.006609049
0.4814815 0.013726487 2.4883712    65
## [492] {fruit/vegetable juice,
##        other vegetables}         => {tropical fruit}        0.006609049
0.3140097 0.021047280 2.9925242    65
## [493] {fruit/vegetable juice,
##        tropical fruit}           => {whole milk}            0.005998983
```

```
0.4370370 0.013726487 1.7104096    59
## [494] {fruit/vegetable juice,
##        whole milk}                => {tropical fruit}        0.005998983
0.2251908 0.026639553 2.1460774    59
## [495] {fruit/vegetable juice,
##        root vegetables}           => {other vegetables}      0.006609049
0.5508475 0.011997966 2.8468653    65
## [496] {fruit/vegetable juice,
##        other vegetables}          => {root vegetables}       0.006609049
0.3140097 0.021047280 2.8808629    65
## [497] {fruit/vegetable juice,
##        root vegetables}           => {whole milk}            0.006507372
0.5423729 0.011997966 2.1226571    64
## [498] {fruit/vegetable juice,
##        whole milk}                => {root vegetables}       0.006507372
0.2442748 0.026639553 2.2410847    64
## [499] {fruit/vegetable juice,
##        soda}                      => {yogurt}                0.005083884
0.2762431 0.018403660 1.9802120    50
## [500] {fruit/vegetable juice,
##        yogurt}                    => {soda}                  0.005083884
0.2717391 0.018708693 1.5583407    50
## [501] {fruit/vegetable juice,
##        soda}                      => {whole milk}            0.006100661
0.3314917 0.018403660 1.2973422    60
## [502] {fruit/vegetable juice,
##        whole milk}                => {soda}                  0.006100661
0.2290076 0.026639553 1.3132887    60
## [503] {fruit/vegetable juice,
##        yogurt}                    => {other vegetables}      0.008235892
0.4402174 0.018708693 2.2751120    81
## [504] {fruit/vegetable juice,
##        other vegetables}          => {yogurt}                0.008235892
0.3913043 0.021047280 2.8050133    81
## [505] {fruit/vegetable juice,
##        yogurt}                    => {whole milk}            0.009456024
0.5054348 0.018708693 1.9780943    93
## [506] {fruit/vegetable juice,
##        whole milk}                => {yogurt}                0.009456024
0.3549618 0.026639553 2.5444968    93
## [507] {fruit/vegetable juice,
##        rolls/buns}                => {whole milk}            0.005592272
0.3846154 0.014539908 1.5052496    55
## [508] {fruit/vegetable juice,
##        whole milk}                => {rolls/buns}            0.005592272
0.2099237 0.026639553 1.1412931    55
## [509] {fruit/vegetable juice,
##        other vegetables}          => {whole milk}            0.010472801
0.4975845 0.021047280 1.9473713   103
## [510] {fruit/vegetable juice,
```

```
##        whole milk}                 => {other vegetables}      0.010472801
0.3931298 0.026639553 2.0317558   103
## [511] {pip fruit,
##        whipped/sour cream}        => {other vegetables}      0.005592272
0.6043956 0.009252669 3.1236105    55
## [512] {other vegetables,
##        pip fruit}                 => {whipped/sour cream}    0.005592272
0.2140078 0.026131164 2.9854844    55
## [513] {pip fruit,
##        whipped/sour cream}        => {whole milk}            0.005998983
0.6483516 0.009252669 2.5374208    59
## [514] {citrus fruit,
##        whipped/sour cream}        => {other vegetables}      0.005693950
0.5233645 0.010879512 2.7048291    56
## [515] {citrus fruit,
##        whipped/sour cream}        => {whole milk}            0.006304016
0.5794393 0.010879512 2.2677219    62
## [516] {citrus fruit,
##        whole milk}                => {whipped/sour cream}    0.006304016
0.2066667 0.030503305 2.8830733    62
## [517] {sausage,
##        whipped/sour cream}        => {whole milk}            0.005083884
0.5617978 0.009049314 2.1986792    50
## [518] {tropical fruit,
##        whipped/sour cream}        => {yogurt}                0.006202339
0.4485294 0.013828165 3.2152236    61
## [519] {whipped/sour cream,
##        yogurt}                    => {tropical fruit}        0.006202339
0.2990196 0.020742247 2.8496685    61
## [520] {tropical fruit,
##        yogurt}                    => {whipped/sour cream}    0.006202339
0.2118056 0.029283172 2.9547626    61
## [521] {tropical fruit,
##        whipped/sour cream}        => {other vegetables}      0.007829181
0.5661765 0.013828165 2.9260881    77
## [522] {other vegetables,
##        whipped/sour cream}        => {tropical fruit}        0.007829181
0.2711268 0.028876462 2.5838485    77
## [523] {other vegetables,
##        tropical fruit}            => {whipped/sour cream}    0.007829181
0.2181303 0.035892222 3.0429952    77
## [524] {tropical fruit,
##        whipped/sour cream}        => {whole milk}            0.007930859
0.5735294 0.013828165 2.2445928    78
## [525] {whipped/sour cream,
##        whole milk}                => {tropical fruit}        0.007930859
0.2460568 0.032231825 2.3449307    78
## [526] {root vegetables,
##        whipped/sour cream}        => {yogurt}                0.006405694
0.3750000 0.017081851 2.6881378    63
```

```
## [527] {whipped/sour cream,
##        yogurt}                    => {root vegetables}       0.006405694
0.3088235 0.020742247 2.8332830    63
## [528] {root vegetables,
##        yogurt}                    => {whipped/sour cream}    0.006405694
0.2480315 0.025826131 3.4601273    63
## [529] {root vegetables,
##        whipped/sour cream}        => {other vegetables}      0.008540925
0.5000000 0.017081851 2.5840778    84
## [530] {other vegetables,
##        whipped/sour cream}        => {root vegetables}       0.008540925
0.2957746 0.028876462 2.7135668    84
## [531] {root vegetables,
##        whipped/sour cream}        => {whole milk}            0.009456024
0.5535714 0.017081851 2.1664843    93
## [532] {whipped/sour cream,
##        whole milk}                => {root vegetables}       0.009456024
0.2933754 0.032231825 2.6915550    93
## [533] {soda,
##        whipped/sour cream}        => {whole milk}            0.005490595
0.4736842 0.011591256 1.8538337    54
## [534] {whipped/sour cream,
##        yogurt}                    => {other vegetables}      0.010167768
0.4901961 0.020742247 2.5334096   100
## [535] {other vegetables,
##        whipped/sour cream}        => {yogurt}                0.010167768
0.3521127 0.028876462 2.5240730   100
## [536] {other vegetables,
##        yogurt}                    => {whipped/sour cream}    0.010167768
0.2341920 0.043416370 3.2670620   100
## [537] {whipped/sour cream,
##        yogurt}                    => {whole milk}            0.010879512
0.5245098 0.020742247 2.0527473   107
## [538] {whipped/sour cream,
##        whole milk}                => {yogurt}                0.010879512
0.3375394 0.032231825 2.4196066   107
## [539] {rolls/buns,
##        whipped/sour cream}        => {other vegetables}      0.006710727
0.4583333 0.014641586 2.3687380    66
## [540] {other vegetables,
##        whipped/sour cream}        => {rolls/buns}            0.006710727
0.2323944 0.028876462 1.2634597    66
## [541] {rolls/buns,
##        whipped/sour cream}        => {whole milk}            0.007829181
0.5347222 0.014641586 2.0927151    77
## [542] {whipped/sour cream,
##        whole milk}                => {rolls/buns}            0.007829181
0.2429022 0.032231825 1.3205877    77
## [543] {other vegetables,
##        whipped/sour cream}        => {whole milk}            0.014641586
```

```
0.5070423 0.028876462 1.9843854   144
## [544] {whipped/sour cream,
##        whole milk}              => {other vegetables}      0.014641586
0.4542587 0.032231825 2.3476795   144
## [545] {pastry,
##        pip fruit}               => {whole milk}            0.005083884
0.4761905 0.010676157 1.8636424    50
## [546] {citrus fruit,
##        pip fruit}               => {tropical fruit}        0.005592272
0.4044118 0.013828165 3.8540598    55
## [547] {pip fruit,
##        tropical fruit}          => {citrus fruit}          0.005592272
0.2736318 0.020437214 3.3061046    55
## [548] {citrus fruit,
##        tropical fruit}          => {pip fruit}             0.005592272
0.2806122 0.019928826 3.7094374    55
## [549] {citrus fruit,
##        pip fruit}               => {other vegetables}      0.005897306
0.4264706 0.013828165 2.2040663    58
## [550] {other vegetables,
##        pip fruit}               => {citrus fruit}          0.005897306
0.2256809 0.026131164 2.7267469    58
## [551] {citrus fruit,
##        other vegetables}        => {pip fruit}             0.005897306
0.2042254 0.028876462 2.6996725    58
## [552] {citrus fruit,
##        pip fruit}               => {whole milk}            0.005185562
0.3750000 0.013828165 1.4676184    51
## [553] {pip fruit,
##        sausage}                 => {whole milk}            0.005592272
0.5188679 0.010777834 2.0306669    55
## [554] {pip fruit,
##        tropical fruit}          => {root vegetables}       0.005287239
0.2587065 0.020437214 2.3734870    52
## [555] {pip fruit,
##        root vegetables}         => {tropical fruit}        0.005287239
0.3398693 0.015556685 3.2389674    52
## [556] {root vegetables,
##        tropical fruit}          => {pip fruit}             0.005287239
0.2512077 0.021047280 3.3207366    52
## [557] {pip fruit,
##        tropical fruit}          => {yogurt}                0.006405694
0.3134328 0.020437214 2.2468017    63
## [558] {pip fruit,
##        yogurt}                  => {tropical fruit}        0.006405694
0.3559322 0.017996950 3.3920477    63
## [559] {tropical fruit,
##        yogurt}                  => {pip fruit}             0.006405694
0.2187500 0.029283172 2.8916751    63
## [560] {pip fruit,
```

```
##         tropical fruit}            => {other vegetables}      0.009456024
0.4626866 0.020437214 2.3912361    93
## [561] {other vegetables,
##         pip fruit}                 => {tropical fruit}        0.009456024
0.3618677 0.026131164 3.4486132    93
## [562] {other vegetables,
##         tropical fruit}            => {pip fruit}             0.009456024
0.2634561 0.035892222 3.4826487    93
## [563] {pip fruit,
##         tropical fruit}            => {whole milk}            0.008439248
0.4129353 0.020437214 1.6160839    83
## [564] {pip fruit,
##         whole milk}                => {tropical fruit}        0.008439248
0.2804054 0.030096594 2.6722744    83
## [565] {pip fruit,
##         root vegetables}           => {yogurt}                0.005287239
0.3398693 0.015556685 2.4363079    52
## [566] {pip fruit,
##         yogurt}                    => {root vegetables}       0.005287239
0.2937853 0.017996950 2.6953158    52
## [567] {root vegetables,
##         yogurt}                    => {pip fruit}             0.005287239
0.2047244 0.025826131 2.7062696    52
## [568] {pip fruit,
##         root vegetables}           => {other vegetables}      0.008134215
0.5228758 0.015556685 2.7023036    80
## [569] {other vegetables,
##         pip fruit}                 => {root vegetables}       0.008134215
0.3112840 0.026131164 2.8558569    80
## [570] {pip fruit,
##         root vegetables}           => {whole milk}            0.008947636
0.5751634 0.015556685 2.2509877    88
## [571] {pip fruit,
##         whole milk}                => {root vegetables}       0.008947636
0.2972973 0.030096594 2.7275363    88
## [572] {pip fruit,
##         yogurt}                    => {other vegetables}      0.008134215
0.4519774 0.017996950 2.3358895    80
## [573] {other vegetables,
##         pip fruit}                 => {yogurt}                0.008134215
0.3112840 0.026131164 2.2313984    80
## [574] {pip fruit,
##         yogurt}                    => {whole milk}            0.009557702
0.5310734 0.017996950 2.0784351    94
## [575] {pip fruit,
##         whole milk}                => {yogurt}                0.009557702
0.3175676 0.030096594 2.2764410    94
## [576] {pip fruit,
##         rolls/buns}                => {other vegetables}      0.005083884
0.3649635 0.013929842 1.8861882    50
```

```
## [577] {pip fruit,
##        rolls/buns}               => {whole milk}            0.006202339
0.4452555 0.013929842 1.7425737    61
## [578] {pip fruit,
##        whole milk}               => {rolls/buns}            0.006202339
0.2060811 0.030096594 1.1204021    61
## [579] {other vegetables,
##        pip fruit}                => {whole milk}            0.013523132
0.5175097 0.026131164 2.0253514   133
## [580] {pip fruit,
##        whole milk}               => {other vegetables}      0.013523132
0.4493243 0.030096594 2.3221780   133
## [581] {pastry,
##        sausage}                  => {whole milk}            0.005693950
0.4552846 0.012506355 1.7818239    56
## [582] {pastry,
##        tropical fruit}           => {other vegetables}      0.005083884
0.3846154 0.013218099 1.9877521    50
## [583] {other vegetables,
##        pastry}                   => {tropical fruit}        0.005083884
0.2252252 0.022572445 2.1464051    50
## [584] {pastry,
##        tropical fruit}           => {whole milk}            0.006710727
0.5076923 0.013218099 1.9869295    66
## [585] {pastry,
##        whole milk}               => {tropical fruit}        0.006710727
0.2018349 0.033248602 1.9234941    66
## [586] {pastry,
##        root vegetables}          => {other vegetables}      0.005897306
0.5370370 0.010981190 2.7754909    58
## [587] {other vegetables,
##        pastry}                   => {root vegetables}       0.005897306
0.2612613 0.022572445 2.3969258    58
## [588] {pastry,
##        root vegetables}          => {whole milk}            0.005693950
0.5185185 0.010981190 2.0292995    56
## [589] {pastry,
##        soda}                     => {rolls/buns}            0.005388917
0.2560386 0.021047280 1.3920067    53
## [590] {pastry,
##        rolls/buns}               => {soda}                  0.005388917
0.2572816 0.020945602 1.4754309    53
## [591] {pastry,
##        soda}                     => {other vegetables}      0.005490595
0.2608696 0.021047280 1.3482145    54
## [592] {other vegetables,
##        pastry}                   => {soda}                  0.005490595
0.2432432 0.022572445 1.3949255    54
## [593] {pastry,
##        soda}                     => {whole milk}            0.008235892
```

```
0.3913043 0.021047280 1.5314279    81
## [594] {pastry,
##        whole milk}               => {soda}                  0.008235892
0.2477064 0.033248602 1.4205205    81
## [595] {soda,
##        whole milk}               => {pastry}                0.008235892
0.2055838 0.040061007 2.3107614    81
## [596] {pastry,
##        yogurt}                   => {rolls/buns}            0.005795628
0.3275862 0.017691917 1.7809897    57
## [597] {pastry,
##        rolls/buns}               => {yogurt}                0.005795628
0.2766990 0.020945602 1.9834803    57
## [598] {pastry,
##        yogurt}                   => {other vegetables}      0.006609049
0.3735632 0.017691917 1.9306328    65
## [599] {other vegetables,
##        pastry}                   => {yogurt}                0.006609049
0.2927928 0.022572445 2.0988463    65
## [600] {pastry,
##        yogurt}                   => {whole milk}            0.009150991
0.5172414 0.017691917 2.0243012    90
## [601] {pastry,
##        whole milk}               => {yogurt}                0.009150991
0.2752294 0.033248602 1.9729451    90
## [602] {pastry,
##        rolls/buns}               => {other vegetables}      0.006100661
0.2912621 0.020945602 1.5052880    60
## [603] {other vegetables,
##        pastry}                   => {rolls/buns}            0.006100661
0.2702703 0.022572445 1.4693798    60
## [604] {pastry,
##        rolls/buns}               => {whole milk}            0.008540925
0.4077670 0.020945602 1.5958569    84
## [605] {pastry,
##        whole milk}               => {rolls/buns}            0.008540925
0.2568807 0.033248602 1.3965849    84
## [606] {other vegetables,
##        pastry}                   => {whole milk}            0.010574479
0.4684685 0.022572445 1.8334212   104
## [607] {pastry,
##        whole milk}               => {other vegetables}      0.010574479
0.3180428 0.033248602 1.6436947   104
## [608] {bottled water,
##        citrus fruit}             => {other vegetables}      0.005083884
0.3759398 0.013523132 1.9429156    50
## [609] {bottled water,
##        other vegetables}         => {citrus fruit}          0.005083884
0.2049180 0.024809354 2.4758831    50
## [610] {bottled water,
```

```
##        citrus fruit}              => {whole milk}            0.005897306
0.4360902 0.013523132 1.7067041    58
## [611] {citrus fruit,
##        tropical fruit}            => {root vegetables}       0.005693950
0.2857143 0.019928826 2.6212687    56
## [612] {citrus fruit,
##        root vegetables}           => {tropical fruit}        0.005693950
0.3218391 0.017691917 3.0671389    56
## [613] {root vegetables,
##        tropical fruit}            => {citrus fruit}          0.005693950
0.2705314 0.021047280 3.2686441    56
## [614] {citrus fruit,
##        tropical fruit}            => {yogurt}                0.006304016
0.3163265 0.019928826 2.2675448    62
## [615] {citrus fruit,
##        yogurt}                    => {tropical fruit}        0.006304016
0.2910798 0.021657346 2.7740019    62
## [616] {tropical fruit,
##        yogurt}                    => {citrus fruit}          0.006304016
0.2152778 0.029283172 2.6010528    62
## [617] {citrus fruit,
##        tropical fruit}            => {other vegetables}      0.009049314
0.4540816 0.019928826 2.3467645    89
## [618] {citrus fruit,
##        other vegetables}          => {tropical fruit}        0.009049314
0.3133803 0.028876462 2.9865262    89
## [619] {other vegetables,
##        tropical fruit}            => {citrus fruit}          0.009049314
0.2521246 0.035892222 3.0462480    89
## [620] {citrus fruit,
##        tropical fruit}            => {whole milk}            0.009049314
0.4540816 0.019928826 1.7771161    89
## [621] {citrus fruit,
##        whole milk}                => {tropical fruit}        0.009049314
0.2966667 0.030503305 2.8272448    89
## [622] {tropical fruit,
##        whole milk}                => {citrus fruit}          0.009049314
0.2139423 0.042297916 2.5849172    89
## [623] {citrus fruit,
##        root vegetables}           => {other vegetables}      0.010371124
0.5862069 0.017691917 3.0296084   102
## [624] {citrus fruit,
##        other vegetables}          => {root vegetables}       0.010371124
0.3591549 0.028876462 3.2950455   102
## [625] {other vegetables,
##        root vegetables}           => {citrus fruit}          0.010371124
0.2188841 0.047381800 2.6446257   102
## [626] {citrus fruit,
##        root vegetables}           => {whole milk}            0.009150991
0.5172414 0.017691917 2.0243012    90
```

```
## [627] {citrus fruit,
##        whole milk}              => {root vegetables}       0.009150991
0.3000000 0.030503305 2.7523321    90
## [628] {citrus fruit,
##        yogurt}                  => {rolls/buns}            0.005795628
0.2676056 0.021657346 1.4548930    57
## [629] {citrus fruit,
##        rolls/buns}              => {yogurt}                0.005795628
0.3454545 0.016776817 2.4763451    57
## [630] {citrus fruit,
##        yogurt}                  => {other vegetables}      0.007625826
0.3521127 0.021657346 1.8197731    75
## [631] {citrus fruit,
##        other vegetables}        => {yogurt}                0.007625826
0.2640845 0.028876462 1.8930548    75
## [632] {citrus fruit,
##        yogurt}                  => {whole milk}            0.010269446
0.4741784 0.021657346 1.8557678   101
## [633] {citrus fruit,
##        whole milk}              => {yogurt}                0.010269446
0.3366667 0.030503305 2.4133503   101
## [634] {citrus fruit,
##        rolls/buns}              => {other vegetables}      0.005998983
0.3575758 0.016776817 1.8480071    59
## [635] {citrus fruit,
##        other vegetables}        => {rolls/buns}            0.005998983
0.2077465 0.028876462 1.1294564    59
## [636] {citrus fruit,
##        rolls/buns}              => {whole milk}            0.007219115
0.4303030 0.016776817 1.6840550    71
## [637] {citrus fruit,
##        whole milk}              => {rolls/buns}            0.007219115
0.2366667 0.030503305 1.2866869    71
## [638] {citrus fruit,
##        other vegetables}        => {whole milk}            0.013014743
0.4507042 0.028876462 1.7638982   128
## [639] {citrus fruit,
##        whole milk}              => {other vegetables}      0.013014743
0.4266667 0.030503305 2.2050797   128
## [640] {sausage,
##        shopping bags}           => {soda}                  0.005693950
0.3636364 0.015658363 2.0853432    56
## [641] {shopping bags,
##        soda}                    => {sausage}               0.005693950
0.2314050 0.024605999 2.4630604    56
## [642] {sausage,
##        soda}                    => {shopping bags}         0.005693950
0.2343096 0.024300966 2.3781580    56
## [643] {sausage,
##        shopping bags}           => {rolls/buns}            0.005998983
```

```
0.3831169 0.015658363 2.0828936    59
## [644] {rolls/buns,
##        shopping bags}           => {sausage}               0.005998983
0.3072917 0.019522115 3.2707939    59
## [645] {sausage,
##        shopping bags}           => {other vegetables}      0.005388917
0.3441558 0.015658363 1.7786509    53
## [646] {other vegetables,
##        shopping bags}           => {sausage}               0.005388917
0.2324561 0.023182511 2.4742491    53
## [647] {other vegetables,
##        sausage}                 => {shopping bags}         0.005388917
0.2000000 0.026944586 2.0299278    53
## [648] {root vegetables,
##        shopping bags}           => {other vegetables}      0.006609049
0.5158730 0.012811388 2.6661120    65
## [649] {other vegetables,
##        shopping bags}           => {root vegetables}       0.006609049
0.2850877 0.023182511 2.6155203    65
## [650] {root vegetables,
##        shopping bags}           => {whole milk}            0.005287239
0.4126984 0.012811388 1.6151567    52
## [651] {shopping bags,
##        whole milk}              => {root vegetables}       0.005287239
0.2157676 0.024504321 1.9795473    52
## [652] {shopping bags,
##        soda}                    => {rolls/buns}            0.006304016
0.2561983 0.024605999 1.3928749    62
## [653] {rolls/buns,
##        shopping bags}           => {soda}                  0.006304016
0.3229167 0.019522115 1.8518282    62
## [654] {shopping bags,
##        soda}                    => {other vegetables}      0.005388917
0.2190083 0.024605999 1.1318688    53
## [655] {other vegetables,
##        shopping bags}           => {soda}                  0.005388917
0.2324561 0.023182511 1.3330648    53
## [656] {shopping bags,
##        soda}                    => {whole milk}            0.006812405
0.2768595 0.024605999 1.0835309    67
## [657] {shopping bags,
##        whole milk}              => {soda}                  0.006812405
0.2780083 0.024504321 1.5942925    67
## [658] {shopping bags,
##        yogurt}                  => {other vegetables}      0.005388917
0.3533333 0.015251652 1.8260816    53
## [659] {other vegetables,
##        shopping bags}           => {yogurt}                0.005388917
0.2324561 0.023182511 1.6663310    53
## [660] {shopping bags,
```

```
##         yogurt}                        => {whole milk}          0.005287239
0.3466667 0.015251652 1.3567317    52
## [661] {shopping bags,
##         whole milk}                    => {yogurt}              0.005287239
0.2157676 0.024504321 1.5467017    52
## [662] {rolls/buns,
##         shopping bags}                 => {other vegetables}    0.005287239
0.2708333 0.019522115 1.3997088    52
## [663] {other vegetables,
##         shopping bags}                 => {rolls/buns}          0.005287239
0.2280702 0.023182511 1.2399503    52
## [664] {rolls/buns,
##         shopping bags}                 => {whole milk}          0.005287239
0.2708333 0.019522115 1.0599466    52
## [665] {shopping bags,
##         whole milk}                    => {rolls/buns}          0.005287239
0.2157676 0.024504321 1.1730651    52
## [666] {other vegetables,
##         shopping bags}                 => {whole milk}          0.007625826
0.3289474 0.023182511 1.2873845    75
## [667] {shopping bags,
##         whole milk}                    => {other vegetables}    0.007625826
0.3112033 0.024504321 1.6083472    75
## [668] {bottled water,
##         sausage}                       => {other vegetables}    0.005083884
0.4237288 0.011997966 2.1898964    50
## [669] {bottled water,
##         other vegetables}              => {sausage}             0.005083884
0.2049180 0.024809354 2.1811351    50
## [670] {sausage,
##         tropical fruit}                => {other vegetables}    0.005998983
0.4306569 0.013929842 2.2257020    59
## [671] {other vegetables,
##         sausage}                       => {tropical fruit}      0.005998983
0.2226415 0.026944586 2.1217822    59
## [672] {sausage,
##         tropical fruit}                => {whole milk}          0.007219115
0.5182482 0.013929842 2.0282415    71
## [673] {sausage,
##         whole milk}                    => {tropical fruit}      0.007219115
0.2414966 0.029893238 2.3014719    71
## [674] {root vegetables,
##         sausage}                       => {yogurt}              0.005185562
0.3469388 0.014946619 2.4869846    51
## [675] {sausage,
##         yogurt}                        => {root vegetables}     0.005185562
0.2642487 0.019623793 2.4243340    51
## [676] {root vegetables,
##         yogurt}                        => {sausage}             0.005185562
0.2007874 0.025826131 2.1371689    51
```

```
## [677] {root vegetables,
##        sausage}                    => {other vegetables}      0.006812405
0.4557823 0.014946619 2.3555539    67
## [678] {other vegetables,
##        sausage}                    => {root vegetables}       0.006812405
0.2528302 0.026944586 2.3195755    67
## [679] {root vegetables,
##        sausage}                    => {whole milk}            0.007727504
0.5170068 0.014946619 2.0233832    76
## [680] {sausage,
##        whole milk}                 => {root vegetables}       0.007727504
0.2585034 0.029893238 2.3716240    76
## [681] {sausage,
##        soda}                       => {yogurt}                0.005592272
0.2301255 0.024300966 1.6496243    55
## [682] {sausage,
##        yogurt}                     => {soda}                  0.005592272
0.2849741 0.019623793 1.6342392    55
## [683] {soda,
##        yogurt}                     => {sausage}               0.005592272
0.2044610 0.027351296 2.1762701    55
## [684] {sausage,
##        soda}                       => {rolls/buns}            0.009659380
0.3974895 0.024300966 2.1610335    95
## [685] {rolls/buns,
##        sausage}                    => {soda}                  0.009659380
0.3156146 0.030604982 1.8099532    95
## [686] {rolls/buns,
##        soda}                       => {sausage}               0.009659380
0.2519894 0.038332486 2.6821598    95
## [687] {sausage,
##        soda}                       => {other vegetables}      0.007219115
0.2970711 0.024300966 1.5353098    71
## [688] {other vegetables,
##        sausage}                    => {soda}                  0.007219115
0.2679245 0.026944586 1.5364652    71
## [689] {other vegetables,
##        soda}                       => {sausage}               0.007219115
0.2204969 0.032740214 2.3469556    71
## [690] {sausage,
##        soda}                       => {whole milk}            0.006710727
0.2761506 0.024300966 1.0807566    66
## [691] {sausage,
##        whole milk}                 => {soda}                  0.006710727
0.2244898 0.029893238 1.2873803    66
## [692] {sausage,
##        yogurt}                     => {rolls/buns}            0.005998983
0.3056995 0.019623793 1.6619980    59
## [693] {sausage,
##        yogurt}                     => {other vegetables}      0.008134215
```

```
0.4145078 0.019623793 2.1422406    80
## [694] {other vegetables,
##        sausage}                  => {yogurt}                0.008134215
0.3018868 0.026944586 2.1640354    80
## [695] {sausage,
##        yogurt}                   => {whole milk}            0.008744281
0.4455959 0.019623793 1.7439058    86
## [696] {sausage,
##        whole milk}               => {yogurt}                0.008744281
0.2925170 0.029893238 2.0968694    86
## [697] {rolls/buns,
##        sausage}                  => {other vegetables}      0.008845958
0.2890365 0.030604982 1.4937858    87
## [698] {other vegetables,
##        sausage}                  => {rolls/buns}            0.008845958
0.3283019 0.026944586 1.7848806    87
## [699] {other vegetables,
##        rolls/buns}               => {sausage}               0.008845958
0.2076372 0.042602949 2.2100781    87
## [700] {rolls/buns,
##        sausage}                  => {whole milk}            0.009354347
0.3056478 0.030604982 1.1961984    92
## [701] {sausage,
##        whole milk}               => {rolls/buns}            0.009354347
0.3129252 0.029893238 1.7012820    92
## [702] {other vegetables,
##        sausage}                  => {whole milk}            0.010167768
0.3773585 0.026944586 1.4768487   100
## [703] {sausage,
##        whole milk}               => {other vegetables}      0.010167768
0.3401361 0.029893238 1.7578760   100
## [704] {bottled water,
##        tropical fruit}           => {soda}                  0.005185562
0.2802198 0.018505338 1.6069747    51
## [705] {soda,
##        tropical fruit}           => {bottled water}         0.005185562
0.2487805 0.020843925 2.2509256    51
## [706] {bottled water,
##        tropical fruit}           => {yogurt}                0.007117438
0.3846154 0.018505338 2.7570644    70
## [707] {bottled water,
##        yogurt}                   => {tropical fruit}        0.007117438
0.3097345 0.022979156 2.9517819    70
## [708] {tropical fruit,
##        yogurt}                   => {bottled water}         0.007117438
0.2430556 0.029283172 2.1991273    70
## [709] {bottled water,
##        tropical fruit}           => {rolls/buns}            0.005388917
0.2912088 0.018505338 1.5832164    53
## [710] {bottled water,
```

```
##        rolls/buns}                => {tropical fruit}        0.005388917
0.2226891 0.024199288 2.1222355    53
## [711] {rolls/buns,
##        tropical fruit}            => {bottled water}         0.005388917
0.2190083 0.024605999 1.9815513    53
## [712] {bottled water,
##        tropical fruit}            => {other vegetables}      0.006202339
0.3351648 0.018505338 1.7321840    61
## [713] {bottled water,
##        other vegetables}          => {tropical fruit}        0.006202339
0.2500000 0.024809354 2.3825097    61
## [714] {bottled water,
##        tropical fruit}            => {whole milk}            0.008032537
0.4340659 0.018505338 1.6987817    79
## [715] {bottled water,
##        whole milk}                => {tropical fruit}        0.008032537
0.2337278 0.034367056 2.2274351    79
## [716] {bottled water,
##        root vegetables}           => {other vegetables}      0.007015760
0.4480519 0.015658363 2.3156022    69
## [717] {bottled water,
##        other vegetables}          => {root vegetables}       0.007015760
0.2827869 0.024809354 2.5944114    69
## [718] {bottled water,
##        root vegetables}           => {whole milk}            0.007320793
0.4675325 0.015658363 1.8297580    72
## [719] {bottled water,
##        whole milk}                => {root vegetables}       0.007320793
0.2130178 0.034367056 1.9543186    72
## [720] {bottled water,
##        soda}                      => {yogurt}                0.007422471
0.2561404 0.028978139 1.8361081    73
## [721] {bottled water,
##        yogurt}                    => {soda}                  0.007422471
0.3230088 0.022979156 1.8523569    73
## [722] {soda,
##        yogurt}                    => {bottled water}         0.007422471
0.2713755 0.027351296 2.4553613    73
## [723] {bottled water,
##        soda}                      => {rolls/buns}            0.006812405
0.2350877 0.028978139 1.2781027    67
## [724] {bottled water,
##        rolls/buns}                => {soda}                  0.006812405
0.2815126 0.024199288 1.6143886    67
## [725] {bottled water,
##        other vegetables}          => {soda}                  0.005693950
0.2295082 0.024809354 1.3161593    56
## [726] {bottled water,
##        soda}                      => {whole milk}            0.007524148
0.2596491 0.028978139 1.0161755    74
```

```
## [727] {bottled water,
##        whole milk}              => {soda}                  0.007524148
0.2189349 0.034367056 1.2555247    74
## [728] {bottled water,
##        yogurt}                  => {rolls/buns}            0.007117438
0.3097345 0.022979156 1.6839353    70
## [729] {bottled water,
##        rolls/buns}              => {yogurt}                0.007117438
0.2941176 0.024199288 2.1083433    70
## [730] {rolls/buns,
##        yogurt}                  => {bottled water}         0.007117438
0.2071006 0.034367056 1.8738126    70
## [731] {bottled water,
##        yogurt}                  => {other vegetables}      0.008134215
0.3539823 0.022979156 1.8294356    80
## [732] {bottled water,
##        other vegetables}        => {yogurt}                0.008134215
0.3278689 0.024809354 2.3502844    80
## [733] {bottled water,
##        yogurt}                  => {whole milk}            0.009659380
0.4203540 0.022979156 1.6451180    95
## [734] {bottled water,
##        whole milk}              => {yogurt}                0.009659380
0.2810651 0.034367056 2.0147778    95
## [735] {bottled water,
##        rolls/buns}              => {other vegetables}      0.007320793
0.3025210 0.024199288 1.5634756    72
## [736] {bottled water,
##        other vegetables}        => {rolls/buns}            0.007320793
0.2950820 0.024809354 1.6042737    72
## [737] {bottled water,
##        rolls/buns}              => {whole milk}            0.008744281
0.3613445 0.024199288 1.4141757    86
## [738] {bottled water,
##        whole milk}              => {rolls/buns}            0.008744281
0.2544379 0.034367056 1.3833037    86
## [739] {bottled water,
##        other vegetables}        => {whole milk}            0.010777834
0.4344262 0.024809354 1.7001918   106
## [740] {bottled water,
##        whole milk}              => {other vegetables}      0.010777834
0.3136095 0.034367056 1.6207825   106
## [741] {root vegetables,
##        tropical fruit}          => {yogurt}                0.008134215
0.3864734 0.021047280 2.7703835    80
## [742] {tropical fruit,
##        yogurt}                  => {root vegetables}       0.008134215
0.2777778 0.029283172 2.5484556    80
## [743] {root vegetables,
##        yogurt}                  => {tropical fruit}        0.008134215
```

```
0.3149606 0.025826131 3.0015870    80
## [744] {root vegetables,
##        tropical fruit}           => {rolls/buns}           0.005897306
0.2801932 0.021047280 1.5233281    58
## [745] {rolls/buns,
##        tropical fruit}           => {root vegetables}      0.005897306
0.2396694 0.024605999 2.1988328    58
## [746] {rolls/buns,
##        root vegetables}          => {tropical fruit}       0.005897306
0.2426778 0.024300966 2.3127291    58
## [747] {root vegetables,
##        tropical fruit}           => {other vegetables}     0.012302999
0.5845411 0.021047280 3.0209991   121
## [748] {other vegetables,
##        tropical fruit}           => {root vegetables}      0.012302999
0.3427762 0.035892222 3.1447798   121
## [749] {other vegetables,
##        root vegetables}          => {tropical fruit}       0.012302999
0.2596567 0.047381800 2.4745380   121
## [750] {root vegetables,
##        tropical fruit}           => {whole milk}           0.011997966
0.5700483 0.021047280 2.2309690   118
## [751] {tropical fruit,
##        whole milk}               => {root vegetables}      0.011997966
0.2836538 0.042297916 2.6023653   118
## [752] {root vegetables,
##        whole milk}               => {tropical fruit}       0.011997966
0.2453222 0.048906965 2.3379305   118
## [753] {soda,
##        tropical fruit}           => {yogurt}               0.006609049
0.3170732 0.020843925 2.2728970    65
## [754] {tropical fruit,
##        yogurt}                   => {soda}                 0.006609049
0.2256944 0.029283172 1.2942885    65
## [755] {soda,
##        yogurt}                   => {tropical fruit}       0.006609049
0.2416357 0.027351296 2.3027975    65
## [756] {soda,
##        tropical fruit}           => {rolls/buns}           0.005388917
0.2585366 0.020843925 1.4055872    53
## [757] {rolls/buns,
##        tropical fruit}           => {soda}                 0.005388917
0.2190083 0.024605999 1.2559454    53
## [758] {soda,
##        tropical fruit}           => {other vegetables}     0.007219115
0.3463415 0.020843925 1.7899466    71
## [759] {other vegetables,
##        tropical fruit}           => {soda}                 0.007219115
0.2011331 0.035892222 1.1534370    71
## [760] {other vegetables,
```

```
##         soda}                        => {tropical fruit}        0.007219115
0.2204969 0.032740214 2.1013440    71
## [761] {soda,
##         tropical fruit}              => {whole milk}            0.007829181
0.3756098 0.020843925 1.4700048    77
## [762] {tropical fruit,
##         yogurt}                      => {rolls/buns}            0.008744281
0.2986111 0.029283172 1.6234606    86
## [763] {rolls/buns,
##         tropical fruit}              => {yogurt}                0.008744281
0.3553719 0.024605999 2.5474363    86
## [764] {rolls/buns,
##         yogurt}                      => {tropical fruit}        0.008744281
0.2544379 0.034367056 2.4248028    86
## [765] {tropical fruit,
##         yogurt}                      => {other vegetables}      0.012302999
0.4201389 0.029283172 2.1713431   121
## [766] {other vegetables,
##         tropical fruit}              => {yogurt}                0.012302999
0.3427762 0.035892222 2.4571457   121
## [767] {other vegetables,
##         yogurt}                      => {tropical fruit}        0.012302999
0.2833724 0.043416370 2.7005496   121
## [768] {tropical fruit,
##         yogurt}                      => {whole milk}            0.015149975
0.5173611 0.029283172 2.0247698   149
## [769] {tropical fruit,
##         whole milk}                  => {yogurt}                0.015149975
0.3581731 0.042297916 2.5675162   149
## [770] {whole milk,
##         yogurt}                      => {tropical fruit}        0.015149975
0.2704174 0.056024403 2.5770885   149
## [771] {rolls/buns,
##         tropical fruit}              => {other vegetables}      0.007829181
0.3181818 0.024605999 1.6444131    77
## [772] {other vegetables,
##         tropical fruit}              => {rolls/buns}            0.007829181
0.2181303 0.035892222 1.1859102    77
## [773] {rolls/buns,
##         tropical fruit}              => {whole milk}            0.010981190
0.4462810 0.024605999 1.7465872   108
## [774] {tropical fruit,
##         whole milk}                  => {rolls/buns}            0.010981190
0.2596154 0.042297916 1.4114524   108
## [775] {other vegetables,
##         tropical fruit}              => {whole milk}            0.017081851
0.4759207 0.035892222 1.8625865   168
## [776] {tropical fruit,
##         whole milk}                  => {other vegetables}      0.017081851
0.4038462 0.042297916 2.0871397   168
```

```
## [777] {other vegetables,
##        whole milk}             => {tropical fruit}       0.017081851
0.2282609 0.074834774 2.1753349   168
## [778] {root vegetables,
##        soda}                   => {other vegetables}     0.008235892
0.4426230 0.018607016 2.2875443    81
## [779] {other vegetables,
##        soda}                   => {root vegetables}      0.008235892
0.2515528 0.032740214 2.3078561    81
## [780] {root vegetables,
##        soda}                   => {whole milk}           0.008134215
0.4371585 0.018607016 1.7108848    80
## [781] {soda,
##        whole milk}             => {root vegetables}      0.008134215
0.2030457 0.040061007 1.8628305    80
## [782] {root vegetables,
##        yogurt}                 => {rolls/buns}           0.007219115
0.2795276 0.025826131 1.5197090    71
## [783] {rolls/buns,
##        root vegetables}        => {yogurt}               0.007219115
0.2970711 0.024300966 2.1295150    71
## [784] {rolls/buns,
##        yogurt}                 => {root vegetables}      0.007219115
0.2100592 0.034367056 1.9271753    71
## [785] {root vegetables,
##        yogurt}                 => {other vegetables}     0.012913066
0.5000000 0.025826131 2.5840778   127
## [786] {other vegetables,
##        root vegetables}        => {yogurt}               0.012913066
0.2725322 0.047381800 1.9536108   127
## [787] {other vegetables,
##        yogurt}                 => {root vegetables}      0.012913066
0.2974239 0.043416370 2.7286977   127
## [788] {root vegetables,
##        yogurt}                 => {whole milk}           0.014539908
0.5629921 0.025826131 2.2033536   143
## [789] {root vegetables,
##        whole milk}             => {yogurt}               0.014539908
0.2972973 0.048906965 2.1311362   143
## [790] {whole milk,
##        yogurt}                 => {root vegetables}      0.014539908
0.2595281 0.056024403 2.3810253   143
## [791] {rolls/buns,
##        root vegetables}        => {other vegetables}     0.012201322
0.5020921 0.024300966 2.5948898   120
## [792] {other vegetables,
##        root vegetables}        => {rolls/buns}           0.012201322
0.2575107 0.047381800 1.4000100   120
## [793] {other vegetables,
##        rolls/buns}             => {root vegetables}      0.012201322
```

```
0.2863962 0.042602949 2.6275247   120
## [794] {rolls/buns,
##        root vegetables}          => {whole milk}           0.012709710
0.5230126 0.024300966 2.0468876   125
## [795] {root vegetables,
##        whole milk}               => {rolls/buns}           0.012709710
0.2598753 0.048906965 1.4128652   125
## [796] {rolls/buns,
##        whole milk}               => {root vegetables}      0.012709710
0.2244165 0.056634469 2.0588959   125
## [797] {other vegetables,
##        root vegetables}          => {whole milk}           0.023182511
0.4892704 0.047381800 1.9148326   228
## [798] {root vegetables,
##        whole milk}               => {other vegetables}     0.023182511
0.4740125 0.048906965 2.4497702   228
## [799] {other vegetables,
##        whole milk}               => {root vegetables}      0.023182511
0.3097826 0.074834774 2.8420820   228
## [800] {soda,
##        yogurt}                   => {rolls/buns}           0.008642603
0.3159851 0.027351296 1.7179181    85
## [801] {rolls/buns,
##        soda}                     => {yogurt}               0.008642603
0.2254642 0.038332486 1.6162101    85
## [802] {rolls/buns,
##        yogurt}                   => {soda}                 0.008642603
0.2514793 0.034367056 1.4421567    85
## [803] {soda,
##        yogurt}                   => {other vegetables}     0.008337570
0.3048327 0.027351296 1.5754229    82
## [804] {other vegetables,
##        soda}                     => {yogurt}               0.008337570
0.2546584 0.032740214 1.8254849    82
## [805] {soda,
##        yogurt}                   => {whole milk}           0.010472801
0.3828996 0.027351296 1.4985348   103
## [806] {soda,
##        whole milk}               => {yogurt}               0.010472801
0.2614213 0.040061007 1.8739641   103
## [807] {rolls/buns,
##        soda}                     => {other vegetables}     0.009862735
0.2572944 0.038332486 1.3297376    97
## [808] {other vegetables,
##        soda}                     => {rolls/buns}           0.009862735
0.3012422 0.032740214 1.6377653    97
## [809] {other vegetables,
##        rolls/buns}               => {soda}                 0.009862735
0.2315036 0.042602949 1.3276022    97
## [810] {rolls/buns,
```

```
##        soda}                        => {whole milk}            0.008845958
0.2307692 0.038332486 0.9031498    87
## [811] {soda,
##        whole milk}                  => {rolls/buns}            0.008845958
0.2208122 0.040061007 1.2004908    87
## [812] {other vegetables,
##        soda}                        => {whole milk}            0.013929842
0.4254658 0.032740214 1.6651240   137
## [813] {soda,
##        whole milk}                  => {other vegetables}      0.013929842
0.3477157 0.040061007 1.7970490   137
## [814] {rolls/buns,
##        yogurt}                      => {other vegetables}      0.011489578
0.3343195 0.034367056 1.7278153   113
## [815] {other vegetables,
##        yogurt}                      => {rolls/buns}            0.011489578
0.2646370 0.043416370 1.4387534   113
## [816] {other vegetables,
##        rolls/buns}                  => {yogurt}                0.011489578
0.2696897 0.042602949 1.9332351   113
## [817] {rolls/buns,
##        yogurt}                      => {whole milk}            0.015556685
0.4526627 0.034367056 1.7715630   153
## [818] {whole milk,
##        yogurt}                      => {rolls/buns}            0.015556685
0.2776770 0.056024403 1.5096478   153
## [819] {rolls/buns,
##        whole milk}                  => {yogurt}                0.015556685
0.2746858 0.056634469 1.9690488   153
## [820] {other vegetables,
##        yogurt}                      => {whole milk}            0.022267412
0.5128806 0.043416370 2.0072345   219
## [821] {whole milk,
##        yogurt}                      => {other vegetables}      0.022267412
0.3974592 0.056024403 2.0541308   219
## [822] {other vegetables,
##        whole milk}                  => {yogurt}                0.022267412
0.2975543 0.074834774 2.1329789   219
## [823] {other vegetables,
##        rolls/buns}                  => {whole milk}            0.017895272
0.4200477 0.042602949 1.6439194   176
## [824] {rolls/buns,
##        whole milk}                  => {other vegetables}      0.017895272
0.3159785 0.056634469 1.6330258   176
## [825] {other vegetables,
##        whole milk}                  => {rolls/buns}            0.017895272
0.2391304 0.074834774 1.3000817   176
## [826] {fruit/vegetable juice,
##        other vegetables,
##        yogurt}                      => {whole milk}            0.005083884
```

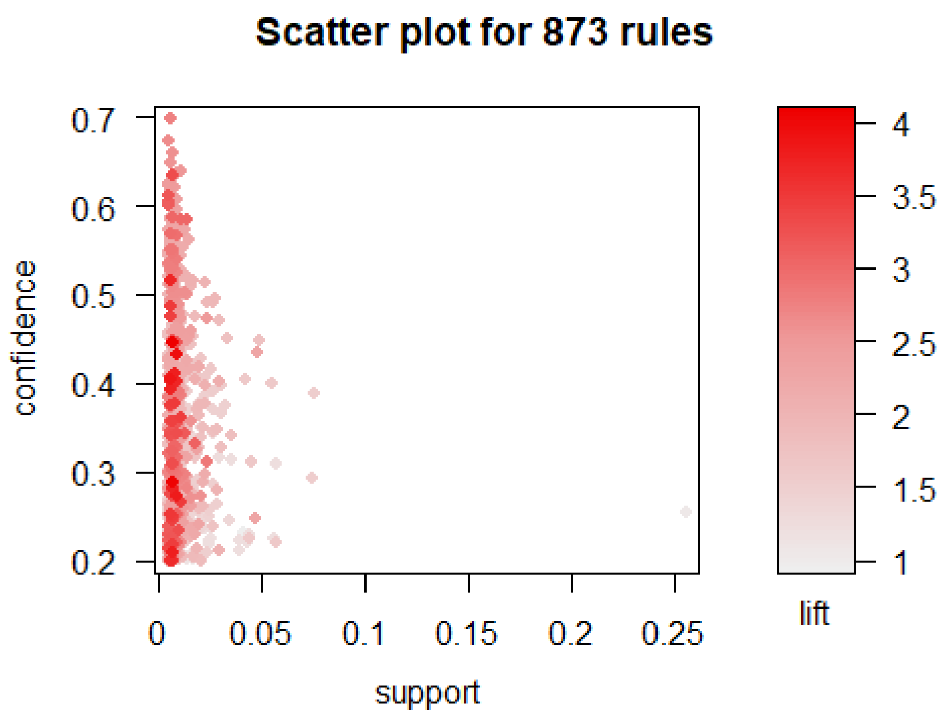
```
0.6172840 0.008235892 2.4158327    50
## [827] {fruit/vegetable juice,
##        whole milk,
##        yogurt}                   => {other vegetables}      0.005083884
0.5376344 0.009456024 2.7785782    50
## [828] {fruit/vegetable juice,
##        other vegetables,
##        whole milk}               => {yogurt}                0.005083884
0.4854369 0.010472801 3.4797900    50
## [829] {other vegetables,
##        whole milk,
##        yogurt}                   => {fruit/vegetable juice} 0.005083884
0.2283105 0.022267412 3.1581347    50
## [830] {other vegetables,
##        root vegetables,
##        whipped/sour cream}       => {whole milk}            0.005185562
0.6071429 0.008540925 2.3761441    51
## [831] {root vegetables,
##        whipped/sour cream,
##        whole milk}               => {other vegetables}      0.005185562
0.5483871 0.009456024 2.8341498    51
## [832] {other vegetables,
##        whipped/sour cream,
##        whole milk}               => {root vegetables}       0.005185562
0.3541667 0.014641586 3.2492809    51
## [833] {other vegetables,
##        root vegetables,
##        whole milk}               => {whipped/sour cream}    0.005185562
0.2236842 0.023182511 3.1204741    51
## [834] {other vegetables,
##        whipped/sour cream,
##        yogurt}                   => {whole milk}            0.005592272
0.5500000 0.010167768 2.1525070    55
## [835] {whipped/sour cream,
##        whole milk,
##        yogurt}                   => {other vegetables}      0.005592272
0.5140187 0.010879512 2.6565286    55
## [836] {other vegetables,
##        whipped/sour cream,
##        whole milk}               => {yogurt}                0.005592272
0.3819444 0.014641586 2.7379181    55
## [837] {other vegetables,
##        whole milk,
##        yogurt}                   => {whipped/sour cream}    0.005592272
0.2511416 0.022267412 3.5035137    55
## [838] {other vegetables,
##        pip fruit,
##        root vegetables}          => {whole milk}            0.005490595
0.6750000 0.008134215 2.6417131    54
## [839] {pip fruit,
```

```
##         root vegetables,
##         whole milk}                => {other vegetables}      0.005490595
0.6136364 0.008947636 3.1713682    54
## [840] {other vegetables,
##         pip fruit,
##         whole milk}                => {root vegetables}       0.005490595
0.4060150 0.013523132 3.7249607    54
## [841] {other vegetables,
##         root vegetables,
##         whole milk}                => {pip fruit}             0.005490595
0.2368421 0.023182511 3.1308362    54
## [842] {other vegetables,
##         pip fruit,
##         yogurt}                    => {whole milk}            0.005083884
0.6250000 0.008134215 2.4460306    50
## [843] {pip fruit,
##         whole milk,
##         yogurt}                    => {other vegetables}      0.005083884
0.5319149 0.009557702 2.7490189    50
## [844] {other vegetables,
##         pip fruit,
##         whole milk}                => {yogurt}                0.005083884
0.3759398 0.013523132 2.6948749    50
## [845] {other vegetables,
##         whole milk,
##         yogurt}                    => {pip fruit}             0.005083884
0.2283105 0.022267412 3.0180562    50
## [846] {citrus fruit,
##         other vegetables,
##         root vegetables}           => {whole milk}            0.005795628
0.5588235 0.010371124 2.1870392    57
## [847] {citrus fruit,
##         root vegetables,
##         whole milk}                => {other vegetables}      0.005795628
0.6333333 0.009150991 3.2731652    57
## [848] {citrus fruit,
##         other vegetables,
##         whole milk}                => {root vegetables}       0.005795628
0.4453125 0.013014743 4.0854929    57
## [849] {other vegetables,
##         root vegetables,
##         whole milk}                => {citrus fruit}          0.005795628
0.2500000 0.023182511 3.0205774    57
## [850] {root vegetables,
##         tropical fruit,
##         yogurt}                    => {whole milk}            0.005693950
0.7000000 0.008134215 2.7395543    56
## [851] {root vegetables,
##         tropical fruit,
##         whole milk}                => {yogurt}                0.005693950
```

```
0.4745763 0.011997966 3.4019370    56
## [852] {tropical fruit,
##        whole milk,
##        yogurt}                     => {root vegetables}       0.005693950
0.3758389 0.015149975 3.4481118    56
## [853] {root vegetables,
##        whole milk,
##        yogurt}                     => {tropical fruit}        0.005693950
0.3916084 0.014539908 3.7320432    56
## [854] {other vegetables,
##        root vegetables,
##        tropical fruit}             => {whole milk}            0.007015760
0.5702479 0.012302999 2.2317503    69
## [855] {root vegetables,
##        tropical fruit,
##        whole milk}                 => {other vegetables}      0.007015760
0.5847458 0.011997966 3.0220571    69
## [856] {other vegetables,
##        tropical fruit,
##        whole milk}                 => {root vegetables}       0.007015760
0.4107143 0.017081851 3.7680737    69
## [857] {other vegetables,
##        root vegetables,
##        whole milk}                 => {tropical fruit}        0.007015760
0.3026316 0.023182511 2.8840907    69
## [858] {other vegetables,
##        tropical fruit,
##        yogurt}                     => {whole milk}            0.007625826
0.6198347 0.012302999 2.4258155    75
## [859] {tropical fruit,
##        whole milk,
##        yogurt}                     => {other vegetables}      0.007625826
0.5033557 0.015149975 2.6014206    75
## [860] {other vegetables,
##        tropical fruit,
##        whole milk}                 => {yogurt}                0.007625826
0.4464286 0.017081851 3.2001640    75
## [861] {other vegetables,
##        whole milk,
##        yogurt}                     => {tropical fruit}        0.007625826
0.3424658 0.022267412 3.2637119    75
## [862] {other vegetables,
##        root vegetables,
##        yogurt}                     => {whole milk}            0.007829181
0.6062992 0.012913066 2.3728423    77
## [863] {root vegetables,
##        whole milk,
##        yogurt}                     => {other vegetables}      0.007829181
0.5384615 0.014539908 2.7828530    77
## [864] {other vegetables,
```

```
##         root vegetables,
##         whole milk}              => {yogurt}              0.007829181
0.3377193 0.023182511 2.4208960    77
## [865] {other vegetables,
##         whole milk,
##         yogurt}                  => {root vegetables}     0.007829181
0.3515982 0.022267412 3.2257165    77
## [866] {other vegetables,
##         rolls/buns,
##         root vegetables}         => {whole milk}          0.006202339
0.5083333 0.012201322 1.9894383    61
## [867] {rolls/buns,
##         root vegetables,
##         whole milk}              => {other vegetables}    0.006202339
0.4880000 0.012709710 2.5220599    61
## [868] {other vegetables,
##         root vegetables,
##         whole milk}              => {rolls/buns}          0.006202339
0.2675439 0.023182511 1.4545571    61
## [869] {other vegetables,
##         rolls/buns,
##         whole milk}              => {root vegetables}     0.006202339
0.3465909 0.017895272 3.1797776    61
## [870] {other vegetables,
##         rolls/buns,
##         yogurt}                  => {whole milk}          0.005998983
0.5221239 0.011489578 2.0434097    59
## [871] {rolls/buns,
##         whole milk,
##         yogurt}                  => {other vegetables}    0.005998983
0.3856209 0.015556685 1.9929489    59
## [872] {other vegetables,
##         whole milk,
##         yogurt}                  => {rolls/buns}          0.005998983
0.2694064 0.022267412 1.4646832    59
## [873] {other vegetables,
##         rolls/buns,
##         whole milk}              => {yogurt}              0.005998983
0.3352273 0.017895272 2.4030322    59
```

```
plot(basketrules)
```




```
## Choose a subset
inspect(subset(basketrules, subset=lift > 3))

##      lhs                           rhs                      support
confidence    coverage     lift count
## [1]  {herbs}                    => {root vegetables}      0.007015760
0.4312500 0.016268429 3.956477   69
## [2]  {sliced cheese}            => {sausage}              0.007015760
0.2863071 0.024504321 3.047435   69
## [3]  {berries}                  => {whipped/sour cream}   0.009049314
0.2721713 0.033248602 3.796886   89
## [4]  {beef}                     => {root vegetables}      0.017386884
0.3313953 0.052465684 3.040367  171
## [5]  {onions,
##       root vegetables}          => {other vegetables}     0.005693950
0.6021505 0.009456024 3.112008   56
## [6]  {onions,
##       other vegetables}         => {root vegetables}      0.005693950
0.4000000 0.014234875 3.669776   56
## [7]  {chicken,
##       whole milk}               => {root vegetables}      0.005998983
0.3410405 0.017590239 3.128855   59
## [8]  {frozen vegetables,
##       other vegetables}         => {root vegetables}      0.006100661
0.3428571 0.017793594 3.145522   60
## [9]  {beef,
##       other vegetables}         => {root vegetables}      0.007930859
```

```
0.4020619 0.019725470 3.688692    78
## [10] {beef,
##       whole milk}            => {root vegetables}       0.008032537
0.3779904 0.021250635 3.467851    79
## [11] {curd,
##       whole milk}            => {whipped/sour cream}    0.005897306
0.2256809 0.026131164 3.148329    58
## [12] {curd,
##       tropical fruit}        => {yogurt}                0.005287239
0.5148515 0.010269446 3.690645    52
## [13] {margarine,
##       whole milk}            => {domestic eggs}         0.005185562
0.2142857 0.024199288 3.377404    51
## [14] {butter,
##       whole milk}            => {domestic eggs}         0.005998983
0.2177122 0.027554652 3.431409    59
## [15] {domestic eggs,
##       whole milk}            => {butter}                0.005998983
0.2000000 0.029994916 3.609174    59
## [16] {butter,
##       other vegetables}      => {whipped/sour cream}    0.005795628
0.2893401 0.020030503 4.036397    57
## [17] {other vegetables,
##       whipped/sour cream}    => {butter}                0.005795628
0.2007042 0.028876462 3.621883    57
## [18] {butter,
##       whole milk}            => {whipped/sour cream}    0.006710727
0.2435424 0.027554652 3.397503    66
## [19] {whipped/sour cream,
##       whole milk}            => {butter}                0.006710727
0.2082019 0.032231825 3.757185    66
## [20] {butter,
##       other vegetables}      => {root vegetables}       0.006609049
0.3299492 0.020030503 3.027100    65
## [21] {domestic eggs,
##       other vegetables}      => {whipped/sour cream}    0.005083884
0.2283105 0.022267412 3.185012    50
## [22] {domestic eggs,
##       other vegetables}      => {root vegetables}       0.007320793
0.3287671 0.022267412 3.016254    72
## [23] {pip fruit,
##       whipped/sour cream}    => {other vegetables}      0.005592272
0.6043956 0.009252669 3.123610    55
## [24] {tropical fruit,
##       whipped/sour cream}    => {yogurt}                0.006202339
0.4485294 0.013828165 3.215224    61
## [25] {other vegetables,
##       tropical fruit}        => {whipped/sour cream}    0.007829181
0.2181303 0.035892222 3.042995    77
## [26] {root vegetables,
```

```
##       yogurt}                 => {whipped/sour cream}    0.006405694
0.2480315 0.025826131 3.460127    63
## [27] {other vegetables,
##       yogurt}                 => {whipped/sour cream}    0.010167768
0.2341920 0.043416370 3.267062   100
## [28] {citrus fruit,
##       pip fruit}              => {tropical fruit}        0.005592272
0.4044118 0.013828165 3.854060    55
## [29] {pip fruit,
##       tropical fruit}         => {citrus fruit}          0.005592272
0.2736318 0.020437214 3.306105    55
## [30] {citrus fruit,
##       tropical fruit}         => {pip fruit}             0.005592272
0.2806122 0.019928826 3.709437    55
## [31] {pip fruit,
##       root vegetables}        => {tropical fruit}        0.005287239
0.3398693 0.015556685 3.238967    52
## [32] {root vegetables,
##       tropical fruit}         => {pip fruit}             0.005287239
0.2512077 0.021047280 3.320737    52
## [33] {pip fruit,
##       yogurt}                 => {tropical fruit}        0.006405694
0.3559322 0.017996950 3.392048    63
## [34] {other vegetables,
##       pip fruit}              => {tropical fruit}        0.009456024
0.3618677 0.026131164 3.448613    93
## [35] {other vegetables,
##       tropical fruit}         => {pip fruit}             0.009456024
0.2634561 0.035892222 3.482649    93
## [36] {citrus fruit,
##       root vegetables}        => {tropical fruit}        0.005693950
0.3218391 0.017691917 3.067139    56
## [37] {root vegetables,
##       tropical fruit}         => {citrus fruit}          0.005693950
0.2705314 0.021047280 3.268644    56
## [38] {other vegetables,
##       tropical fruit}         => {citrus fruit}          0.009049314
0.2521246 0.035892222 3.046248    89
## [39] {citrus fruit,
##       root vegetables}        => {other vegetables}      0.010371124
0.5862069 0.017691917 3.029608   102
## [40] {citrus fruit,
##       other vegetables}       => {root vegetables}       0.010371124
0.3591549 0.028876462 3.295045   102
## [41] {rolls/buns,
##       shopping bags}          => {sausage}               0.005998983
0.3072917 0.019522115 3.270794    59
## [42] {root vegetables,
##       yogurt}                 => {tropical fruit}        0.008134215
0.3149606 0.025826131 3.001587    80
```

```
## [43] {root vegetables,
##       tropical fruit}        => {other vegetables}      0.012302999
0.5845411 0.021047280 3.020999   121
## [44] {other vegetables,
##       tropical fruit}        => {root vegetables}       0.012302999
0.3427762 0.035892222 3.144780   121
## [45] {fruit/vegetable juice,
##       other vegetables,
##       whole milk}            => {yogurt}                0.005083884
0.4854369 0.010472801 3.479790    50
## [46] {other vegetables,
##       whole milk,
##       yogurt}                => {fruit/vegetable juice} 0.005083884
0.2283105 0.022267412 3.158135    50
## [47] {other vegetables,
##       whipped/sour cream,
##       whole milk}            => {root vegetables}       0.005185562
0.3541667 0.014641586 3.249281    51
## [48] {other vegetables,
##       root vegetables,
##       whole milk}            => {whipped/sour cream}    0.005185562
0.2236842 0.023182511 3.120474    51
## [49] {other vegetables,
##       whole milk,
##       yogurt}                => {whipped/sour cream}    0.005592272
0.2511416 0.022267412 3.503514    55
## [50] {pip fruit,
##       root vegetables,
##       whole milk}            => {other vegetables}      0.005490595
0.6136364 0.008947636 3.171368    54
## [51] {other vegetables,
##       pip fruit,
##       whole milk}            => {root vegetables}       0.005490595
0.4060150 0.013523132 3.724961    54
## [52] {other vegetables,
##       root vegetables,
##       whole milk}            => {pip fruit}             0.005490595
0.2368421 0.023182511 3.130836    54
## [53] {other vegetables,
##       whole milk,
##       yogurt}                => {pip fruit}             0.005083884
0.2283105 0.022267412 3.018056    50
## [54] {citrus fruit,
##       root vegetables,
##       whole milk}            => {other vegetables}      0.005795628
0.6333333 0.009150991 3.273165    57
## [55] {citrus fruit,
##       other vegetables,
##       whole milk}            => {root vegetables}       0.005795628
0.4453125 0.013014743 4.085493    57
```

```
## [56] {other vegetables,
##       root vegetables,
##       whole milk}            => {citrus fruit}          0.005795628
0.2500000 0.023182511 3.020577    57
## [57] {root vegetables,
##       tropical fruit,
##       whole milk}            => {yogurt}                0.005693950
0.4745763 0.011997966 3.401937    56
## [58] {tropical fruit,
##       whole milk,
##       yogurt}                => {root vegetables}       0.005693950
0.3758389 0.015149975 3.448112    56
## [59] {root vegetables,
##       whole milk,
##       yogurt}                => {tropical fruit}        0.005693950
0.3916084 0.014539908 3.732043    56
## [60] {root vegetables,
##       tropical fruit,
##       whole milk}            => {other vegetables}      0.007015760
0.5847458 0.011997966 3.022057    69
## [61] {other vegetables,
##       tropical fruit,
##       whole milk}            => {root vegetables}       0.007015760
0.4107143 0.017081851 3.768074    69
## [62] {other vegetables,
##       tropical fruit,
##       whole milk}            => {yogurt}                0.007625826
0.4464286 0.017081851 3.200164    75
## [63] {other vegetables,
##       whole milk,
##       yogurt}                => {tropical fruit}        0.007625826
0.3424658 0.022267412 3.263712    75
## [64] {other vegetables,
##       whole milk,
##       yogurt}                => {root vegetables}       0.007829181
0.3515982 0.022267412 3.225716    77
## [65] {other vegetables,
##       rolls/buns,
##       whole milk}            => {root vegetables}       0.006202339
0.3465909 0.017895272 3.179778    61
```

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

```
##       lhs                           rhs                     support
confidence    coverage     lift count
## [1]   {baking powder}            => {whole milk}       0.009252669
0.5229885 0.017691917 2.046793    91
## [2]   {oil,
##        other vegetables}         => {whole milk}       0.005083884
0.5102041 0.009964413 1.996760    50
```

```
## [3]   {onions,
##        root vegetables}          => {other vegetables} 0.005693950
0.6021505 0.009456024 3.112008    56
## [4]   {onions,
##        whole milk}               => {other vegetables} 0.006609049
0.5462185 0.012099644 2.822942    65
## [5]   {hygiene articles,
##        other vegetables}         => {whole milk}       0.005185562
0.5425532 0.009557702 2.123363    51
## [6]   {other vegetables,
##        sugar}                    => {whole milk}       0.006304016
0.5849057 0.010777834 2.289115    62
## [7]   {long life bakery product,
##        other vegetables}         => {whole milk}       0.005693950
0.5333333 0.010676157 2.087279    56
## [8]   {cream cheese,
##        yogurt}                   => {whole milk}       0.006609049
0.5327869 0.012404677 2.085141    65
## [9]   {chicken,
##        root vegetables}          => {other vegetables} 0.005693950
0.5233645 0.010879512 2.704829    56
## [10]  {chicken,
##        root vegetables}          => {whole milk}       0.005998983
0.5514019 0.010879512 2.157993    59
## [11]  {chicken,
##        rolls/buns}               => {whole milk}       0.005287239
0.5473684 0.009659380 2.142208    52
## [12]  {coffee,
##        yogurt}                   => {whole milk}       0.005083884
0.5208333 0.009761057 2.038359    50
## [13]  {frozen vegetables,
##        root vegetables}          => {other vegetables} 0.006100661
0.5263158 0.011591256 2.720082    60
## [14]  {frozen vegetables,
##        root vegetables}          => {whole milk}       0.006202339
0.5350877 0.011591256 2.094146    61
## [15]  {frozen vegetables,
##        other vegetables}         => {whole milk}       0.009659380
0.5428571 0.017793594 2.124552    95
## [16]  {beef,
##        yogurt}                   => {whole milk}       0.006100661
0.5217391 0.011692933 2.041904    60
## [17]  {curd,
##        whipped/sour cream}       => {whole milk}       0.005897306
0.5631068 0.010472801 2.203802    58
## [18]  {curd,
##        tropical fruit}           => {yogurt}           0.005287239
0.5148515 0.010269446 3.690645    52
## [19]  {curd,
##        tropical fruit}           => {other vegetables} 0.005287239
```

```
0.5148515 0.010269446 2.660833    52
## [20]  {curd,
##        tropical fruit}           => {whole milk}       0.006507372
0.6336634 0.010269446 2.479936    64
## [21]  {curd,
##        root vegetables}          => {other vegetables} 0.005490595
0.5046729 0.010879512 2.608228    54
## [22]  {curd,
##        root vegetables}          => {whole milk}       0.006202339
0.5700935 0.010879512 2.231146    61
## [23]  {curd,
##        yogurt}                   => {whole milk}       0.010066090
0.5823529 0.017285206 2.279125    99
## [24]  {curd,
##        rolls/buns}               => {whole milk}       0.005897306
0.5858586 0.010066090 2.292845    58
## [25]  {curd,
##        other vegetables}         => {whole milk}       0.009862735
0.5739645 0.017183528 2.246296    97
## [26]  {pork,
##        root vegetables}          => {other vegetables} 0.007015760
0.5149254 0.013624809 2.661214    69
## [27]  {pork,
##        rolls/buns}               => {whole milk}       0.006202339
0.5495495 0.011286223 2.150744    61
## [28]  {frankfurter,
##        tropical fruit}           => {whole milk}       0.005185562
0.5483871 0.009456024 2.146195    51
## [29]  {frankfurter,
##        yogurt}                   => {whole milk}       0.006202339
0.5545455 0.011184545 2.170296    61
## [30]  {bottled beer,
##        yogurt}                   => {whole milk}       0.005185562
0.5604396 0.009252669 2.193364    51
## [31]  {brown bread,
##        tropical fruit}           => {whole milk}       0.005693950
0.5333333 0.010676157 2.087279    56
## [32]  {brown bread,
##        root vegetables}          => {whole milk}       0.005693950
0.5600000 0.010167768 2.191643    56
## [33]  {domestic eggs,
##        margarine}                => {whole milk}       0.005185562
0.6219512 0.008337570 2.434099    51
## [34]  {margarine,
##        root vegetables}          => {other vegetables} 0.005897306
0.5321101 0.011082867 2.750028    58
## [35]  {margarine,
##        rolls/buns}               => {whole milk}       0.007930859
0.5379310 0.014743264 2.105273    78
## [36]  {butter,
```

```
##        domestic eggs}            => {whole milk}       0.005998983
0.6210526 0.009659380 2.430582    59
## [37]  {butter,
##        whipped/sour cream}       => {other vegetables} 0.005795628
0.5700000 0.010167768 2.945849    57
## [38]  {butter,
##        whipped/sour cream}       => {whole milk}       0.006710727
0.6600000 0.010167768 2.583008    66
## [39]  {butter,
##        citrus fruit}             => {whole milk}       0.005083884
0.5555556 0.009150991 2.174249    50
## [40]  {bottled water,
##        butter}                   => {whole milk}       0.005388917
0.6022727 0.008947636 2.357084    53
## [41]  {butter,
##        tropical fruit}           => {other vegetables} 0.005490595
0.5510204 0.009964413 2.847759    54
## [42]  {butter,
##        tropical fruit}           => {whole milk}       0.006202339
0.6224490 0.009964413 2.436047    61
## [43]  {butter,
##        root vegetables}          => {other vegetables} 0.006609049
0.5118110 0.012913066 2.645119    65
## [44]  {butter,
##        root vegetables}          => {whole milk}       0.008235892
0.6377953 0.012913066 2.496107    81
## [45]  {butter,
##        yogurt}                   => {whole milk}       0.009354347
0.6388889 0.014641586 2.500387    92
## [46]  {butter,
##        other vegetables}         => {whole milk}       0.011489578
0.5736041 0.020030503 2.244885   113
## [47]  {newspapers,
##        root vegetables}          => {other vegetables} 0.005998983
0.5221239 0.011489578 2.698417    59
## [48]  {newspapers,
##        root vegetables}          => {whole milk}       0.005795628
0.5044248 0.011489578 1.974142    57
## [49]  {domestic eggs,
##        whipped/sour cream}       => {other vegetables} 0.005083884
0.5102041 0.009964413 2.636814    50
## [50]  {domestic eggs,
##        whipped/sour cream}       => {whole milk}       0.005693950
0.5714286 0.009964413 2.236371    56
## [51]  {domestic eggs,
##        pip fruit}                => {whole milk}       0.005388917
0.6235294 0.008642603 2.440275    53
## [52]  {citrus fruit,
##        domestic eggs}            => {whole milk}       0.005693950
0.5490196 0.010371124 2.148670    56
```

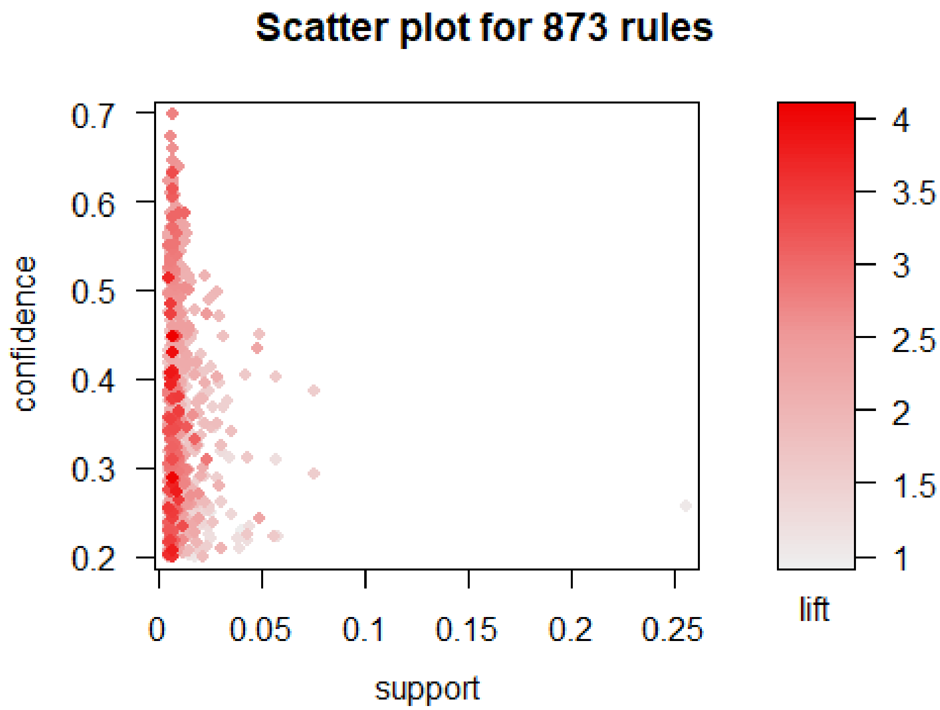
```
## [53]  {domestic eggs,
##        tropical fruit}          => {whole milk}       0.006914082
0.6071429 0.011387900 2.376144    68
## [54]  {domestic eggs,
##        root vegetables}         => {other vegetables} 0.007320793
0.5106383 0.014336553 2.639058    72
## [55]  {domestic eggs,
##        root vegetables}         => {whole milk}       0.008540925
0.5957447 0.014336553 2.331536    84
## [56]  {domestic eggs,
##        yogurt}                  => {whole milk}       0.007727504
0.5390071 0.014336553 2.109485    76
## [57]  {domestic eggs,
##        other vegetables}        => {whole milk}       0.012302999
0.5525114 0.022267412 2.162336   121
## [58]  {fruit/vegetable juice,
##        root vegetables}         => {other vegetables} 0.006609049
0.5508475 0.011997966 2.846865    65
## [59]  {fruit/vegetable juice,
##        root vegetables}         => {whole milk}       0.006507372
0.5423729 0.011997966 2.122657    64
## [60]  {fruit/vegetable juice,
##        yogurt}                  => {whole milk}       0.009456024
0.5054348 0.018708693 1.978094    93
## [61]  {pip fruit,
##        whipped/sour cream}      => {other vegetables} 0.005592272
0.6043956 0.009252669 3.123610    55
## [62]  {pip fruit,
##        whipped/sour cream}      => {whole milk}       0.005998983
0.6483516 0.009252669 2.537421    59
## [63]  {citrus fruit,
##        whipped/sour cream}      => {other vegetables} 0.005693950
0.5233645 0.010879512 2.704829    56
## [64]  {citrus fruit,
##        whipped/sour cream}      => {whole milk}       0.006304016
0.5794393 0.010879512 2.267722    62
## [65]  {sausage,
##        whipped/sour cream}      => {whole milk}       0.005083884
0.5617978 0.009049314 2.198679    50
## [66]  {tropical fruit,
##        whipped/sour cream}      => {other vegetables} 0.007829181
0.5661765 0.013828165 2.926088    77
## [67]  {tropical fruit,
##        whipped/sour cream}      => {whole milk}       0.007930859
0.5735294 0.013828165 2.244593    78
## [68]  {root vegetables,
##        whipped/sour cream}      => {whole milk}       0.009456024
0.5535714 0.017081851 2.166484    93
## [69]  {whipped/sour cream,
##        yogurt}                  => {whole milk}       0.010879512
```

```
0.5245098 0.020742247 2.052747   107
## [70]  {rolls/buns,
##        whipped/sour cream}       => {whole milk}       0.007829181
0.5347222 0.014641586 2.092715    77
## [71]  {other vegetables,
##        whipped/sour cream}       => {whole milk}       0.014641586
0.5070423 0.028876462 1.984385   144
## [72]  {pip fruit,
##        sausage}                  => {whole milk}       0.005592272
0.5188679 0.010777834 2.030667    55
## [73]  {pip fruit,
##        root vegetables}          => {other vegetables} 0.008134215
0.5228758 0.015556685 2.702304    80
## [74]  {pip fruit,
##        root vegetables}          => {whole milk}       0.008947636
0.5751634 0.015556685 2.250988    88
## [75]  {pip fruit,
##        yogurt}                   => {whole milk}       0.009557702
0.5310734 0.017996950 2.078435    94
## [76]  {other vegetables,
##        pip fruit}                => {whole milk}       0.013523132
0.5175097 0.026131164 2.025351   133
## [77]  {pastry,
##        tropical fruit}           => {whole milk}       0.006710727
0.5076923 0.013218099 1.986930    66
## [78]  {pastry,
##        root vegetables}          => {other vegetables} 0.005897306
0.5370370 0.010981190 2.775491    58
## [79]  {pastry,
##        root vegetables}          => {whole milk}       0.005693950
0.5185185 0.010981190 2.029299    56
## [80]  {pastry,
##        yogurt}                   => {whole milk}       0.009150991
0.5172414 0.017691917 2.024301    90
## [81]  {citrus fruit,
##        root vegetables}          => {other vegetables} 0.010371124
0.5862069 0.017691917 3.029608   102
## [82]  {citrus fruit,
##        root vegetables}          => {whole milk}       0.009150991
0.5172414 0.017691917 2.024301    90
## [83]  {root vegetables,
##        shopping bags}            => {other vegetables} 0.006609049
0.5158730 0.012811388 2.666112    65
## [84]  {sausage,
##        tropical fruit}           => {whole milk}       0.007219115
0.5182482 0.013929842 2.028241    71
## [85]  {root vegetables,
##        sausage}                  => {whole milk}       0.007727504
0.5170068 0.014946619 2.023383    76
## [86]  {root vegetables,
```

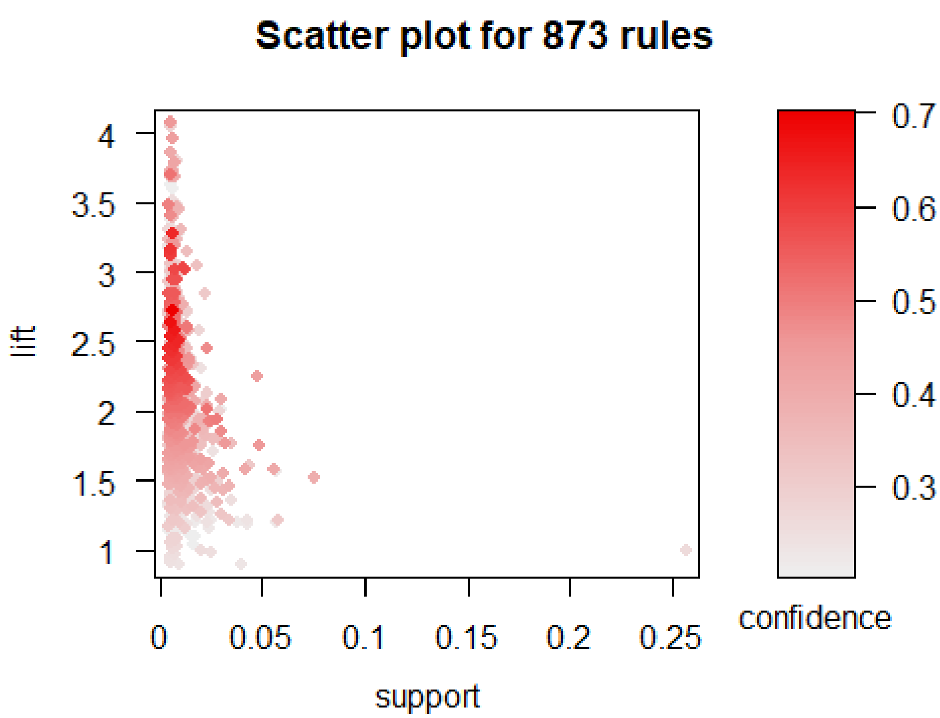
```
##        tropical fruit}           => {other vegetables} 0.012302999
0.5845411 0.021047280 3.020999   121
## [87]  {root vegetables,
##        tropical fruit}           => {whole milk}       0.011997966
0.5700483 0.021047280 2.230969   118
## [88]  {tropical fruit,
##        yogurt}                   => {whole milk}       0.015149975
0.5173611 0.029283172 2.024770   149
## [89]  {root vegetables,
##        yogurt}                   => {whole milk}       0.014539908
0.5629921 0.025826131 2.203354   143
## [90]  {rolls/buns,
##        root vegetables}          => {other vegetables} 0.012201322
0.5020921 0.024300966 2.594890   120
## [91]  {rolls/buns,
##        root vegetables}          => {whole milk}       0.012709710
0.5230126 0.024300966 2.046888   125
## [92]  {other vegetables,
##        yogurt}                   => {whole milk}       0.022267412
0.5128806 0.043416370 2.007235   219
## [93]  {fruit/vegetable juice,
##        other vegetables,
##        yogurt}                   => {whole milk}       0.005083884
0.6172840 0.008235892 2.415833    50
## [94]  {fruit/vegetable juice,
##        whole milk,
##        yogurt}                   => {other vegetables} 0.005083884
0.5376344 0.009456024 2.778578    50
## [95]  {other vegetables,
##        root vegetables,
##        whipped/sour cream}       => {whole milk}       0.005185562
0.6071429 0.008540925 2.376144    51
## [96]  {root vegetables,
##        whipped/sour cream,
##        whole milk}               => {other vegetables} 0.005185562
0.5483871 0.009456024 2.834150    51
## [97]  {other vegetables,
##        whipped/sour cream,
##        yogurt}                   => {whole milk}       0.005592272
0.5500000 0.010167768 2.152507    55
## [98]  {whipped/sour cream,
##        whole milk,
##        yogurt}                   => {other vegetables} 0.005592272
0.5140187 0.010879512 2.656529    55
## [99]  {other vegetables,
##        pip fruit,
##        root vegetables}          => {whole milk}       0.005490595
0.6750000 0.008134215 2.641713    54
## [100] {pip fruit,
##        root vegetables,
```

```
##        whole milk}                => {other vegetables} 0.005490595
0.6136364 0.008947636 3.171368    54
## [101] {other vegetables,
##        pip fruit,
##        yogurt}                    => {whole milk}       0.005083884
0.6250000 0.008134215 2.446031    50
## [102] {pip fruit,
##        whole milk,
##        yogurt}                    => {other vegetables} 0.005083884
0.5319149 0.009557702 2.749019    50
## [103] {citrus fruit,
##        other vegetables,
##        root vegetables}           => {whole milk}       0.005795628
0.5588235 0.010371124 2.187039    57
## [104] {citrus fruit,
##        root vegetables,
##        whole milk}                => {other vegetables} 0.005795628
0.6333333 0.009150991 3.273165    57
## [105] {root vegetables,
##        tropical fruit,
##        yogurt}                    => {whole milk}       0.005693950
0.7000000 0.008134215 2.739554    56
## [106] {other vegetables,
##        root vegetables,
##        tropical fruit}            => {whole milk}       0.007015760
0.5702479 0.012302999 2.231750    69
## [107] {root vegetables,
##        tropical fruit,
##        whole milk}                => {other vegetables} 0.007015760
0.5847458 0.011997966 3.022057    69
## [108] {other vegetables,
##        tropical fruit,
##        yogurt}                    => {whole milk}       0.007625826
0.6198347 0.012302999 2.425816    75
## [109] {tropical fruit,
##        whole milk,
##        yogurt}                    => {other vegetables} 0.007625826
0.5033557 0.015149975 2.601421    75
## [110] {other vegetables,
##        root vegetables,
##        yogurt}                    => {whole milk}       0.007829181
0.6062992 0.012913066 2.372842    77
## [111] {root vegetables,
##        whole milk,
##        yogurt}                    => {other vegetables} 0.007829181
0.5384615 0.014539908 2.782853    77
## [112] {other vegetables,
##        rolls/buns,
##        root vegetables}           => {whole milk}       0.006202339
0.5083333 0.012201322 1.989438    61
```

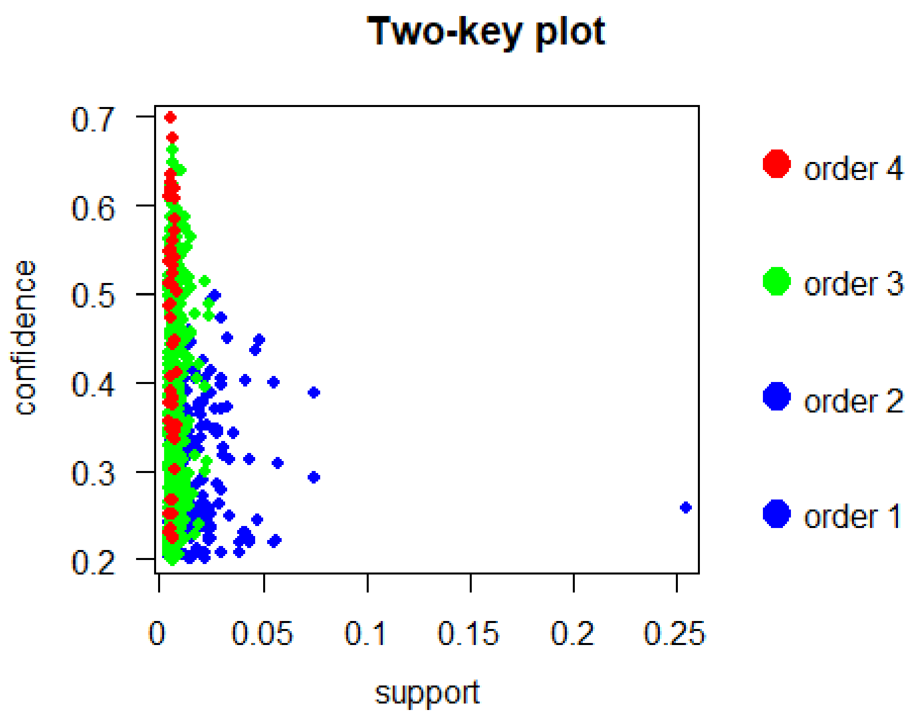
```
## [113] {other vegetables,
##        rolls/buns,
##        yogurt}                  => {whole milk}       0.005998983
0.5221239 0.011489578 2.043410    59

inspect(subset(basketrules, subset=lift > 3 & confidence > 0.55))

##     lhs                        rhs                     support confidence
coverage     lift count
## [1] {onions,
##      root vegetables}    => {other vegetables} 0.005693950  0.6021505
0.009456024 3.112008    56
## [2] {pip fruit,
##      whipped/sour cream} => {other vegetables} 0.005592272  0.6043956
0.009252669 3.123610    55
## [3] {citrus fruit,
##      root vegetables}    => {other vegetables} 0.010371124  0.5862069
0.017691917 3.029608   102
## [4] {root vegetables,
##      tropical fruit}     => {other vegetables} 0.012302999  0.5845411
0.021047280 3.020999   121
## [5] {pip fruit,
##      root vegetables,
##      whole milk}         => {other vegetables} 0.005490595  0.6136364
0.008947636 3.171368    54
## [6] {citrus fruit,
##      root vegetables,
##      whole milk}         => {other vegetables} 0.005795628  0.6333333
0.009150991 3.273165    57
## [7] {root vegetables,
##      tropical fruit,
##      whole milk}         => {other vegetables} 0.007015760  0.5847458
0.011997966 3.022057    69

# plot all the rules in (support, confidence) space
# notice that high lift rules tend to have low support
plot(basketrules)
```




```r
# can swap the axes and color scales
plot(basketrules, measure = c("support", "lift"), shading = "confidence")
```



```r
# "two key" plot: coloring is by size (order) of item set
plot(basketrules, method='two-key plot')
```




```r
# can now look at subsets driven by the plot
inspect(subset(basketrules, support > 0.01))
```

```
##       lhs                           rhs                    support
confidence   coverage      lift count
## [1]   {}                         => {whole milk}        0.25551601
0.2555160 1.00000000 1.0000000  2513
## [2]   {hard cheese}              => {whole milk}        0.01006609
0.4107884 0.02450432 1.6076815    99
## [3]   {butter milk}              => {other vegetables}  0.01037112
0.3709091 0.02796136 1.9169159   102
## [4]   {butter milk}              => {whole milk}        0.01159126
0.4145455 0.02796136 1.6223854   114
## [5]   {ham}                      => {whole milk}        0.01148958
0.4414062 0.02602949 1.7275091   113
## [6]   {sliced cheese}            => {whole milk}        0.01077783
0.4398340 0.02450432 1.7213560   106
## [7]   {oil}                      => {whole milk}        0.01128622
0.4021739 0.02806304 1.5739675   111
## [8]   {onions}                   => {other vegetables}  0.01423488
0.4590164 0.03101169 2.3722681   140
## [9]   {onions}                   => {whole milk}        0.01209964
0.3901639 0.03101169 1.5269647   119
## [10]  {berries}                  => {yogurt}            0.01057448
```

```
0.3180428 0.03324860 2.2798477   104
## [11]  {berries}                     => {other vegetables}   0.01026945
0.3088685 0.03324860 1.5962805   101
## [12]  {berries}                     => {whole milk}         0.01179461
0.3547401 0.03324860 1.3883281   116
## [13]  {hamburger meat}              => {other vegetables}   0.01382816
0.4159021 0.03324860 2.1494470   136
## [14]  {hamburger meat}              => {whole milk}         0.01474326
0.4434251 0.03324860 1.7354101   145
## [15]  {hygiene articles}            => {whole milk}         0.01281139
0.3888889 0.03294357 1.5219746   126
## [16]  {salty snack}                 => {other vegetables}   0.01077783
0.2849462 0.03782410 1.4726465   106
## [17]  {salty snack}                 => {whole milk}         0.01118454
0.2956989 0.03782410 1.1572618   110
## [18]  {sugar}                       => {other vegetables}   0.01077783
0.3183183 0.03385867 1.6451186   106
## [19]  {sugar}                       => {whole milk}         0.01504830
0.4444444 0.03385867 1.7393996   148
## [20]  {waffles}                     => {other vegetables}   0.01006609
0.2619048 0.03843416 1.3535645    99
## [21]  {waffles}                     => {whole milk}         0.01270971
0.3306878 0.03843416 1.2941961   125
## [22]  {long life bakery product} => {other vegetables}   0.01067616
0.2853261 0.03741739 1.4746096   105
## [23]  {long life bakery product} => {whole milk}         0.01352313
0.3614130 0.03741739 1.4144438   133
## [24]  {dessert}                     => {other vegetables}   0.01159126
0.3123288 0.03711235 1.6141636   114
## [25]  {dessert}                     => {whole milk}         0.01372649
0.3698630 0.03711235 1.4475140   135
## [26]  {cream cheese}                => {yogurt}             0.01240468
0.3128205 0.03965430 2.2424123   122
## [27]  {cream cheese}                => {other vegetables}   0.01372649
0.3461538 0.03965430 1.7889769   135
## [28]  {cream cheese}                => {whole milk}         0.01647178
0.4153846 0.03965430 1.6256696   162
## [29]  {chicken}                     => {root vegetables}    0.01087951
0.2535545 0.04290798 2.3262206   107
## [30]  {chicken}                     => {other vegetables}   0.01789527
0.4170616 0.04290798 2.1554393   176
## [31]  {chicken}                     => {whole milk}         0.01759024
0.4099526 0.04290798 1.6044106   173
## [32]  {white bread}                 => {soda}               0.01026945
0.2439614 0.04209456 1.3990437   101
## [33]  {white bread}                 => {other vegetables}   0.01372649
0.3260870 0.04209456 1.6852681   135
## [34]  {white bread}                 => {whole milk}         0.01708185
0.4057971 0.04209456 1.5881474   168
## [35]  {chocolate}                   => {soda}               0.01352313
```

```
0.2725410 0.04961871 1.5629391   133
## [36]  {chocolate}                 => {rolls/buns}         0.01179461
0.2377049 0.04961871 1.2923316   116
## [37]  {chocolate}                 => {other vegetables}   0.01270971
0.2561475 0.04961871 1.3238103   125
## [38]  {chocolate}                 => {whole milk}         0.01667514
0.3360656 0.04961871 1.3152427   164
## [39]  {coffee}                    => {other vegetables}   0.01342145
0.2311734 0.05805796 1.1947400   132
## [40]  {coffee}                    => {whole milk}         0.01870869
0.3222417 0.05805796 1.2611408   184
## [41]  {frozen vegetables}         => {root vegetables}    0.01159126
0.2410148 0.04809354 2.2111759   114
## [42]  {frozen vegetables}         => {yogurt}             0.01240468
0.2579281 0.04809354 1.8489235   122
## [43]  {frozen vegetables}         => {rolls/buns}         0.01016777
0.2114165 0.04809354 1.1494092   100
## [44]  {frozen vegetables}         => {other vegetables}   0.01779359
0.3699789 0.04809354 1.9121083   175
## [45]  {frozen vegetables}         => {whole milk}         0.02043721
0.4249471 0.04809354 1.6630940   201
## [46]  {beef}                      => {root vegetables}    0.01738688
0.3313953 0.05246568 3.0403668   171
## [47]  {beef}                      => {yogurt}             0.01169293
0.2228682 0.05246568 1.5976012   115
## [48]  {beef}                      => {rolls/buns}         0.01362481
0.2596899 0.05246568 1.4118576   134
## [49]  {beef}                      => {other vegetables}   0.01972547
0.3759690 0.05246568 1.9430662   194
## [50]  {beef}                      => {whole milk}         0.02125064
0.4050388 0.05246568 1.5851795   209
## [51]  {curd}                      => {root vegetables}    0.01087951
0.2041985 0.05327911 1.8734067   107
## [52]  {curd}                      => {yogurt}             0.01728521
0.3244275 0.05327911 2.3256154   170
## [53]  {curd}                      => {other vegetables}   0.01718353
0.3225191 0.05327911 1.6668288   169
## [54]  {curd}                      => {whole milk}         0.02613116
0.4904580 0.05327911 1.9194805   257
## [55]  {napkins}                   => {soda}               0.01199797
0.2291262 0.05236401 1.3139687   118
## [56]  {napkins}                   => {yogurt}             0.01230300
0.2349515 0.05236401 1.6842183   121
## [57]  {napkins}                   => {rolls/buns}         0.01169293
0.2233010 0.05236401 1.2140216   115
## [58]  {napkins}                   => {other vegetables}   0.01443823
0.2757282 0.05236401 1.4250060   142
## [59]  {napkins}                   => {whole milk}         0.01972547
0.3766990 0.05236401 1.4742678   194
## [60]  {pork}                      => {root vegetables}    0.01362481
```

```
0.2363316 0.05765125 2.1682099   134
## [61]  {pork}                     => {soda}               0.01189629
0.2063492 0.05765125 1.1833495   117
## [62]  {pork}                     => {other vegetables}   0.02165735
0.3756614 0.05765125 1.9414764   213
## [63]  {pork}                     => {whole milk}         0.02216573
0.3844797 0.05765125 1.5047187   218
## [64]  {frankfurter}              => {rolls/buns}         0.01921708
0.3258621 0.05897306 1.7716161   189
## [65]  {frankfurter}              => {other vegetables}   0.01647178
0.2793103 0.05897306 1.4435193   162
## [66]  {frankfurter}              => {whole milk}         0.02053889
0.3482759 0.05897306 1.3630295   202
## [67]  {bottled beer}             => {soda}               0.01698017
0.2108586 0.08052872 1.2092094   167
## [68]  {bottled beer}             => {other vegetables}   0.01616675
0.2007576 0.08052872 1.0375464   159
## [69]  {bottled beer}             => {whole milk}         0.02043721
0.2537879 0.08052872 0.9932367   201
## [70]  {brown bread}              => {yogurt}             0.01453991
0.2241379 0.06487036 1.6067030   143
## [71]  {brown bread}              => {other vegetables}   0.01870869
0.2884013 0.06487036 1.4905025   184
## [72]  {brown bread}              => {whole milk}         0.02521607
0.3887147 0.06487036 1.5212930   248
## [73]  {margarine}                => {yogurt}             0.01423488
0.2430556 0.05856634 1.7423115   140
## [74]  {margarine}                => {rolls/buns}         0.01474326
0.2517361 0.05856634 1.3686151   145
## [75]  {margarine}                => {other vegetables}   0.01972547
0.3368056 0.05856634 1.7406635   194
## [76]  {margarine}                => {whole milk}         0.02419929
0.4131944 0.05856634 1.6170980   238
## [77]  {butter}                   => {root vegetables}    0.01291307
0.2330275 0.05541434 2.1378971   127
## [78]  {butter}                   => {yogurt}             0.01464159
0.2642202 0.05541434 1.8940273   144
## [79]  {butter}                   => {rolls/buns}         0.01342145
0.2422018 0.05541434 1.3167800   132
## [80]  {butter}                   => {other vegetables}   0.02003050
0.3614679 0.05541434 1.8681223   197
## [81]  {butter}                   => {whole milk}         0.02755465
0.4972477 0.05541434 1.9460530   271
## [82]  {newspapers}               => {rolls/buns}         0.01972547
0.2471338 0.07981698 1.3435934   194
## [83]  {newspapers}               => {other vegetables}   0.01931876
0.2420382 0.07981698 1.2508912   190
## [84]  {newspapers}               => {whole milk}         0.02735130
0.3426752 0.07981698 1.3411103   269
## [85]  {domestic eggs}            => {root vegetables}    0.01433655
```

```
0.2259615 0.06344687 2.0730706   141
## [86]  {domestic eggs}            => {yogurt}            0.01433655
0.2259615 0.06344687 1.6197753   141
## [87]  {domestic eggs}            => {rolls/buns}        0.01565836
0.2467949 0.06344687 1.3417510   154
## [88]  {domestic eggs}            => {other vegetables}  0.02226741
0.3509615 0.06344687 1.8138238   219
## [89]  {domestic eggs}            => {whole milk}        0.02999492
0.4727564 0.06344687 1.8502027   295
## [90]  {fruit/vegetable juice}    => {soda}              0.01840366
0.2545710 0.07229283 1.4598869   181
## [91]  {fruit/vegetable juice}    => {yogurt}            0.01870869
0.2587904 0.07229283 1.8551049   184
## [92]  {fruit/vegetable juice}    => {rolls/buns}        0.01453991
0.2011252 0.07229283 1.0934583   143
## [93]  {fruit/vegetable juice}    => {other vegetables}  0.02104728
0.2911392 0.07229283 1.5046529   207
## [94]  {fruit/vegetable juice}    => {whole milk}        0.02663955
0.3684951 0.07229283 1.4421604   262
## [95]  {whipped/sour cream}       => {root vegetables}   0.01708185
0.2382979 0.07168277 2.1862496   168
## [96]  {whipped/sour cream}       => {yogurt}            0.02074225
0.2893617 0.07168277 2.0742510   204
## [97]  {whipped/sour cream}       => {rolls/buns}        0.01464159
0.2042553 0.07168277 1.1104760   144
## [98]  {whipped/sour cream}       => {other vegetables}  0.02887646
0.4028369 0.07168277 2.0819237   284
## [99]  {whipped/sour cream}       => {whole milk}        0.03223183
0.4496454 0.07168277 1.7597542   317
## [100] {pip fruit}                => {tropical fruit}    0.02043721
0.2701613 0.07564820 2.5746476   201
## [101] {pip fruit}                => {root vegetables}   0.01555669
0.2056452 0.07564820 1.8866793   153
## [102] {pip fruit}                => {yogurt}            0.01799695
0.2379032 0.07564820 1.7053777   177
## [103] {pip fruit}                => {other vegetables}  0.02613116
0.3454301 0.07564820 1.7852365   257
## [104] {pip fruit}                => {whole milk}        0.03009659
0.3978495 0.07564820 1.5570432   296
## [105] {pastry}                   => {soda}              0.02104728
0.2365714 0.08896797 1.3566647   207
## [106] {pastry}                   => {rolls/buns}        0.02094560
0.2354286 0.08896797 1.2799558   206
## [107] {pastry}                   => {other vegetables}  0.02257245
0.2537143 0.08896797 1.3112349   222
## [108] {pastry}                   => {whole milk}        0.03324860
0.3737143 0.08896797 1.4625865   327
## [109] {citrus fruit}             => {tropical fruit}    0.01992883
0.2407862 0.08276563 2.2947022   196
## [110] {citrus fruit}             => {root vegetables}   0.01769192
```

```
0.2137592 0.08276563 1.9611211   174
## [111] {citrus fruit}             => {yogurt}            0.02165735
0.2616708 0.08276563 1.8757521   213
## [112] {citrus fruit}             => {rolls/buns}        0.01677682
0.2027027 0.08276563 1.1020349   165
## [113] {citrus fruit}             => {other vegetables}  0.02887646
0.3488943 0.08276563 1.8031403   284
## [114] {citrus fruit}             => {whole milk}        0.03050330
0.3685504 0.08276563 1.4423768   300
## [115] {shopping bags}            => {soda}              0.02460600
0.2497420 0.09852567 1.4321939   242
## [116] {shopping bags}            => {other vegetables}  0.02318251
0.2352941 0.09852567 1.2160366   228
## [117] {shopping bags}            => {whole milk}        0.02450432
0.2487100 0.09852567 0.9733637   241
## [118] {sausage}                  => {soda}              0.02430097
0.2586580 0.09395018 1.4833245   239
## [119] {sausage}                  => {yogurt}            0.01962379
0.2088745 0.09395018 1.4972889   193
## [120] {sausage}                  => {rolls/buns}        0.03060498
0.3257576 0.09395018 1.7710480   301
## [121] {sausage}                  => {other vegetables}  0.02694459
0.2867965 0.09395018 1.4822091   265
## [122] {sausage}                  => {whole milk}        0.02989324
0.3181818 0.09395018 1.2452520   294
## [123] {bottled water}            => {soda}              0.02897814
0.2621895 0.11052364 1.5035766   285
## [124] {bottled water}            => {yogurt}            0.02297916
0.2079117 0.11052364 1.4903873   226
## [125] {bottled water}            => {rolls/buns}        0.02419929
0.2189512 0.11052364 1.1903734   238
## [126] {bottled water}            => {other vegetables}  0.02480935
0.2244710 0.11052364 1.1601012   244
## [127] {bottled water}            => {whole milk}        0.03436706
0.3109476 0.11052364 1.2169396   338
## [128] {tropical fruit}           => {root vegetables}   0.02104728
0.2005814 0.10493137 1.8402220   207
## [129] {tropical fruit}           => {yogurt}            0.02928317
0.2790698 0.10493137 2.0004746   288
## [130] {yogurt}                   => {tropical fruit}    0.02928317
0.2099125 0.13950178 2.0004746   288
## [131] {tropical fruit}           => {rolls/buns}        0.02460600
0.2344961 0.10493137 1.2748863   242
## [132] {tropical fruit}           => {other vegetables}  0.03589222
0.3420543 0.10493137 1.7677896   353
## [133] {tropical fruit}           => {whole milk}        0.04229792
0.4031008 0.10493137 1.5775950   416
## [134] {root vegetables}          => {yogurt}            0.02582613
0.2369403 0.10899847 1.6984751   254
## [135] {root vegetables}          => {rolls/buns}        0.02430097
```

```
0.2229478 0.10899847 1.2121013   239
## [136] {root vegetables}            => {other vegetables}   0.04738180
0.4347015 0.10899847 2.2466049   466
## [137] {other vegetables}           => {root vegetables}    0.04738180
0.2448765 0.19349263 2.2466049   466
## [138] {root vegetables}            => {whole milk}         0.04890696
0.4486940 0.10899847 1.7560310   481
## [139] {soda}                       => {rolls/buns}         0.03833249
0.2198251 0.17437722 1.1951242   377
## [140] {rolls/buns}                 => {soda}               0.03833249
0.2084024 0.18393493 1.1951242   377
## [141] {soda}                       => {whole milk}         0.04006101
0.2297376 0.17437722 0.8991124   394
## [142] {yogurt}                     => {rolls/buns}         0.03436706
0.2463557 0.13950178 1.3393633   338
## [143] {yogurt}                     => {other vegetables}   0.04341637
0.3112245 0.13950178 1.6084566   427
## [144] {other vegetables}           => {yogurt}             0.04341637
0.2243826 0.19349263 1.6084566   427
## [145] {yogurt}                     => {whole milk}         0.05602440
0.4016035 0.13950178 1.5717351   551
## [146] {whole milk}                 => {yogurt}             0.05602440
0.2192598 0.25551601 1.5717351   551
## [147] {rolls/buns}                 => {other vegetables}   0.04260295
0.2316197 0.18393493 1.1970465   419
## [148] {other vegetables}           => {rolls/buns}         0.04260295
0.2201787 0.19349263 1.1970465   419
## [149] {rolls/buns}                 => {whole milk}         0.05663447
0.3079049 0.18393493 1.2050318   557
## [150] {whole milk}                 => {rolls/buns}         0.05663447
0.2216474 0.25551601 1.2050318   557
## [151] {other vegetables}           => {whole milk}         0.07483477
0.3867578 0.19349263 1.5136341   736
## [152] {whole milk}                 => {other vegetables}   0.07483477
0.2928770 0.25551601 1.5136341   736
## [153] {curd,
##        yogurt}                     => {whole milk}         0.01006609
0.5823529 0.01728521 2.2791250    99
## [154] {curd,
##        whole milk}                 => {yogurt}             0.01006609
0.3852140 0.02613116 2.7613555    99
## [155] {other vegetables,
##        pork}                       => {whole milk}         0.01016777
0.4694836 0.02165735 1.8373939   100
## [156] {pork,
##        whole milk}                 => {other vegetables}   0.01016777
0.4587156 0.02216573 2.3707136   100
## [157] {butter,
##        other vegetables}           => {whole milk}         0.01148958
0.5736041 0.02003050 2.2448850   113
```

```
## [158] {butter,
##        whole milk}              => {other vegetables}   0.01148958
0.4169742 0.02755465 2.1549874   113
## [159] {domestic eggs,
##        other vegetables}        => {whole milk}         0.01230300
0.5525114 0.02226741 2.1623358   121
## [160] {domestic eggs,
##        whole milk}              => {other vegetables}   0.01230300
0.4101695 0.02999492 2.1198197   121
## [161] {fruit/vegetable juice,
##        other vegetables}        => {whole milk}         0.01047280
0.4975845 0.02104728 1.9473713   103
## [162] {fruit/vegetable juice,
##        whole milk}              => {other vegetables}   0.01047280
0.3931298 0.02663955 2.0317558   103
## [163] {whipped/sour cream,
##        yogurt}                  => {other vegetables}   0.01016777
0.4901961 0.02074225 2.5334096   100
## [164] {other vegetables,
##        whipped/sour cream}      => {yogurt}             0.01016777
0.3521127 0.02887646 2.5240730   100
## [165] {other vegetables,
##        yogurt}                  => {whipped/sour cream} 0.01016777
0.2341920 0.04341637 3.2670620   100
## [166] {whipped/sour cream,
##        yogurt}                  => {whole milk}         0.01087951
0.5245098 0.02074225 2.0527473   107
## [167] {whipped/sour cream,
##        whole milk}              => {yogurt}             0.01087951
0.3375394 0.03223183 2.4196066   107
## [168] {other vegetables,
##        whipped/sour cream}      => {whole milk}         0.01464159
0.5070423 0.02887646 1.9843854   144
## [169] {whipped/sour cream,
##        whole milk}              => {other vegetables}   0.01464159
0.4542587 0.03223183 2.3476795   144
## [170] {other vegetables,
##        pip fruit}               => {whole milk}         0.01352313
0.5175097 0.02613116 2.0253514   133
## [171] {pip fruit,
##        whole milk}              => {other vegetables}   0.01352313
0.4493243 0.03009659 2.3221780   133
## [172] {other vegetables,
##        pastry}                  => {whole milk}         0.01057448
0.4684685 0.02257245 1.8334212   104
## [173] {pastry,
##        whole milk}              => {other vegetables}   0.01057448
0.3180428 0.03324860 1.6436947   104
## [174] {citrus fruit,
##        root vegetables}         => {other vegetables}   0.01037112
```

```
0.5862069 0.01769192 3.0296084   102
## [175] {citrus fruit,
##        other vegetables}         => {root vegetables}    0.01037112
0.3591549 0.02887646 3.2950455   102
## [176] {other vegetables,
##        root vegetables}          => {citrus fruit}       0.01037112
0.2188841 0.04738180 2.6446257   102
## [177] {citrus fruit,
##        yogurt}                   => {whole milk}         0.01026945
0.4741784 0.02165735 1.8557678   101
## [178] {citrus fruit,
##        whole milk}               => {yogurt}             0.01026945
0.3366667 0.03050330 2.4133503   101
## [179] {citrus fruit,
##        other vegetables}         => {whole milk}         0.01301474
0.4507042 0.02887646 1.7638982   128
## [180] {citrus fruit,
##        whole milk}               => {other vegetables}   0.01301474
0.4266667 0.03050330 2.2050797   128
## [181] {other vegetables,
##        sausage}                  => {whole milk}         0.01016777
0.3773585 0.02694459 1.4768487   100
## [182] {sausage,
##        whole milk}               => {other vegetables}   0.01016777
0.3401361 0.02989324 1.7578760   100
## [183] {bottled water,
##        other vegetables}         => {whole milk}         0.01077783
0.4344262 0.02480935 1.7001918   106
## [184] {bottled water,
##        whole milk}               => {other vegetables}   0.01077783
0.3136095 0.03436706 1.6207825   106
## [185] {root vegetables,
##        tropical fruit}           => {other vegetables}   0.01230300
0.5845411 0.02104728 3.0209991   121
## [186] {other vegetables,
##        tropical fruit}           => {root vegetables}    0.01230300
0.3427762 0.03589222 3.1447798   121
## [187] {other vegetables,
##        root vegetables}          => {tropical fruit}     0.01230300
0.2596567 0.04738180 2.4745380   121
## [188] {root vegetables,
##        tropical fruit}           => {whole milk}         0.01199797
0.5700483 0.02104728 2.2309690   118
## [189] {tropical fruit,
##        whole milk}               => {root vegetables}    0.01199797
0.2836538 0.04229792 2.6023653   118
## [190] {root vegetables,
##        whole milk}               => {tropical fruit}     0.01199797
0.2453222 0.04890696 2.3379305   118
## [191] {tropical fruit,
```

```
##        yogurt}                    => {other vegetables}   0.01230300
0.4201389 0.02928317 2.1713431   121
## [192] {other vegetables,
##        tropical fruit}            => {yogurt}             0.01230300
0.3427762 0.03589222 2.4571457   121
## [193] {other vegetables,
##        yogurt}                    => {tropical fruit}     0.01230300
0.2833724 0.04341637 2.7005496   121
## [194] {tropical fruit,
##        yogurt}                    => {whole milk}         0.01514997
0.5173611 0.02928317 2.0247698   149
## [195] {tropical fruit,
##        whole milk}                => {yogurt}             0.01514997
0.3581731 0.04229792 2.5675162   149
## [196] {whole milk,
##        yogurt}                    => {tropical fruit}     0.01514997
0.2704174 0.05602440 2.5770885   149
## [197] {rolls/buns,
##        tropical fruit}            => {whole milk}         0.01098119
0.4462810 0.02460600 1.7465872   108
## [198] {tropical fruit,
##        whole milk}                => {rolls/buns}         0.01098119
0.2596154 0.04229792 1.4114524   108
## [199] {other vegetables,
##        tropical fruit}            => {whole milk}         0.01708185
0.4759207 0.03589222 1.8625865   168
## [200] {tropical fruit,
##        whole milk}                => {other vegetables}   0.01708185
0.4038462 0.04229792 2.0871397   168
## [201] {other vegetables,
##        whole milk}                => {tropical fruit}     0.01708185
0.2282609 0.07483477 2.1753349   168
## [202] {root vegetables,
##        yogurt}                    => {other vegetables}   0.01291307
0.5000000 0.02582613 2.5840778   127
## [203] {other vegetables,
##        root vegetables}           => {yogurt}             0.01291307
0.2725322 0.04738180 1.9536108   127
## [204] {other vegetables,
##        yogurt}                    => {root vegetables}    0.01291307
0.2974239 0.04341637 2.7286977   127
## [205] {root vegetables,
##        yogurt}                    => {whole milk}         0.01453991
0.5629921 0.02582613 2.2033536   143
## [206] {root vegetables,
##        whole milk}                => {yogurt}             0.01453991
0.2972973 0.04890696 2.1311362   143
## [207] {whole milk,
##        yogurt}                    => {root vegetables}    0.01453991
0.2595281 0.05602440 2.3810253   143
```

```
## [208] {rolls/buns,
##        root vegetables}          => {other vegetables}   0.01220132
0.5020921 0.02430097 2.5948898   120
## [209] {other vegetables,
##        root vegetables}          => {rolls/buns}         0.01220132
0.2575107 0.04738180 1.4000100   120
## [210] {other vegetables,
##        rolls/buns}               => {root vegetables}    0.01220132
0.2863962 0.04260295 2.6275247   120
## [211] {rolls/buns,
##        root vegetables}          => {whole milk}         0.01270971
0.5230126 0.02430097 2.0468876   125
## [212] {root vegetables,
##        whole milk}               => {rolls/buns}         0.01270971
0.2598753 0.04890696 1.4128652   125
## [213] {rolls/buns,
##        whole milk}               => {root vegetables}    0.01270971
0.2244165 0.05663447 2.0588959   125
## [214] {other vegetables,
##        root vegetables}          => {whole milk}         0.02318251
0.4892704 0.04738180 1.9148326   228
## [215] {root vegetables,
##        whole milk}               => {other vegetables}   0.02318251
0.4740125 0.04890696 2.4497702   228
## [216] {other vegetables,
##        whole milk}               => {root vegetables}    0.02318251
0.3097826 0.07483477 2.8420820   228
## [217] {soda,
##        yogurt}                   => {whole milk}         0.01047280
0.3828996 0.02735130 1.4985348   103
## [218] {soda,
##        whole milk}               => {yogurt}             0.01047280
0.2614213 0.04006101 1.8739641   103
## [219] {other vegetables,
##        soda}                     => {whole milk}         0.01392984
0.4254658 0.03274021 1.6651240   137
## [220] {soda,
##        whole milk}               => {other vegetables}   0.01392984
0.3477157 0.04006101 1.7970490   137
## [221] {rolls/buns,
##        yogurt}                   => {other vegetables}   0.01148958
0.3343195 0.03436706 1.7278153   113
## [222] {other vegetables,
##        yogurt}                   => {rolls/buns}         0.01148958
0.2646370 0.04341637 1.4387534   113
## [223] {other vegetables,
##        rolls/buns}               => {yogurt}             0.01148958
0.2696897 0.04260295 1.9332351   113
## [224] {rolls/buns,
##        yogurt}                   => {whole milk}         0.01555669
```

```
0.4526627 0.03436706 1.7715630   153
## [225] {whole milk,
##        yogurt}                    => {rolls/buns}         0.01555669
0.2776770 0.05602440 1.5096478   153
## [226] {rolls/buns,
##        whole milk}                => {yogurt}             0.01555669
0.2746858 0.05663447 1.9690488   153
## [227] {other vegetables,
##        yogurt}                    => {whole milk}         0.02226741
0.5128806 0.04341637 2.0072345   219
## [228] {whole milk,
##        yogurt}                    => {other vegetables}   0.02226741
0.3974592 0.05602440 2.0541308   219
## [229] {other vegetables,
##        whole milk}                => {yogurt}             0.02226741
0.2975543 0.07483477 2.1329789   219
## [230] {other vegetables,
##        rolls/buns}                => {whole milk}         0.01789527
0.4200477 0.04260295 1.6439194   176
## [231] {rolls/buns,
##        whole milk}                => {other vegetables}   0.01789527
0.3159785 0.05663447 1.6330258   176
## [232] {other vegetables,
##        whole milk}                => {rolls/buns}         0.01789527
0.2391304 0.07483477 1.3000817   176
```

```
inspect(subset(basketrules, confidence > 0.2))
```

```
##       lhs                            rhs                      support
confidence    coverage      lift count
## [1]   {}                          => {whole milk}          0.255516014
0.2555160 1.000000000 1.0000000  2513
## [2]   {cake bar}                  => {whole milk}          0.005592272
0.4230769 0.013218099 1.6557746    55
## [3]   {dishes}                    => {other vegetables}    0.005998983
0.3410405 0.017590239 1.7625502    59
## [4]   {dishes}                    => {whole milk}          0.005287239
0.3005780 0.017590239 1.1763569    52
## [5]   {mustard}                   => {whole milk}          0.005185562
0.4322034 0.011997966 1.6914924    51
## [6]   {pot plants}                => {whole milk}          0.006914082
0.4000000 0.017285206 1.5654596    68
## [7]   {chewing gum}               => {soda}                0.005388917
0.2560386 0.021047280 1.4683033    53
## [8]   {chewing gum}               => {whole milk}          0.005083884
0.2415459 0.021047280 0.9453259    50
## [9]   {canned fish}               => {other vegetables}    0.005083884
0.3378378 0.015048297 1.7459985    50
## [10]  {pasta}                     => {whole milk}          0.006100661
0.4054054 0.015048297 1.5866145    60
```

```
## [11]  {herbs}                        => {root vegetables}       0.007015760
0.4312500 0.016268429 3.9564774    69
## [12]  {herbs}                        => {other vegetables}      0.007727504
0.4750000 0.016268429 2.4548739    76
## [13]  {herbs}                        => {whole milk}            0.007727504
0.4750000 0.016268429 1.8589833    76
## [14]  {processed cheese}             => {soda}                  0.005287239
0.3190184 0.016573462 1.8294729    52
## [15]  {processed cheese}             => {other vegetables}      0.005490595
0.3312883 0.016573462 1.7121497    54
## [16]  {processed cheese}             => {whole milk}            0.007015760
0.4233129 0.016573462 1.6566981    69
## [17]  {semi-finished bread}          => {other vegetables}      0.005185562
0.2931034 0.017691917 1.5148042    51
## [18]  {semi-finished bread}          => {whole milk}            0.007117438
0.4022989 0.017691917 1.5744565    70
## [19]  {beverages}                    => {yogurt}                0.005490595
0.2109375 0.026029487 1.5120775    54
## [20]  {beverages}                    => {rolls/buns}            0.005388917
0.2070312 0.026029487 1.1255679    53
## [21]  {beverages}                    => {whole milk}            0.006812405
0.2617188 0.026029487 1.0242753    67
## [22]  {ice cream}                    => {soda}                  0.006100661
0.2439024 0.025012710 1.3987058    60
## [23]  {ice cream}                    => {other vegetables}      0.005083884
0.2032520 0.025012710 1.0504381    50
## [24]  {ice cream}                    => {whole milk}            0.005897306
0.2357724 0.025012710 0.9227303    58
## [25]  {detergent}                    => {other vegetables}      0.006405694
0.3333333 0.019217082 1.7227185    63
## [26]  {detergent}                    => {whole milk}            0.008947636
0.4656085 0.019217082 1.8222281    88
## [27]  {pickled vegetables}           => {other vegetables}      0.006405694
0.3579545 0.017895272 1.8499648    63
## [28]  {pickled vegetables}           => {whole milk}            0.007117438
0.3977273 0.017895272 1.5565650    70
## [29]  {baking powder}                => {other vegetables}      0.007320793
0.4137931 0.017691917 2.1385471    72
## [30]  {baking powder}                => {whole milk}            0.009252669
0.5229885 0.017691917 2.0467935    91
## [31]  {flour}                        => {other vegetables}      0.006304016
0.3625731 0.017386884 1.8738342    62
## [32]  {flour}                        => {whole milk}            0.008439248
0.4853801 0.017386884 1.8996074    83
## [33]  {soft cheese}                  => {yogurt}                0.005998983
0.3511905 0.017081851 2.5174623    59
## [34]  {soft cheese}                  => {rolls/buns}            0.005388917
0.3154762 0.017081851 1.7151511    53
## [35]  {soft cheese}                  => {other vegetables}      0.007117438
0.4166667 0.017081851 2.1533981    70
```

```
## [36]  {soft cheese}              => {whole milk}          0.007524148
0.4404762 0.017081851 1.7238692    74
## [37]  {specialty bar}            => {soda}                0.007219115
0.2639405 0.027351296 1.5136181    71
## [38]  {specialty bar}            => {rolls/buns}          0.005592272
0.2044610 0.027351296 1.1115940    55
## [39]  {specialty bar}            => {other vegetables}    0.005592272
0.2044610 0.027351296 1.0566861    55
## [40]  {specialty bar}            => {whole milk}          0.006507372
0.2379182 0.027351296 0.9311284    64
## [41]  {misc. beverages}          => {soda}                0.007320793
0.2580645 0.028368073 1.4799210    72
## [42]  {misc. beverages}          => {whole milk}          0.007015760
0.2473118 0.028368073 0.9678917    69
## [43]  {grapes}                   => {tropical fruit}      0.006100661
0.2727273 0.022369090 2.5991015    60
## [44]  {grapes}                   => {other vegetables}    0.009049314
0.4045455 0.022369090 2.0907538    89
## [45]  {grapes}                   => {whole milk}          0.007320793
0.3272727 0.022369090 1.2808306    72
## [46]  {cat food}                 => {yogurt}              0.006202339
0.2663755 0.023284189 1.9094778    61
## [47]  {cat food}                 => {other vegetables}    0.006507372
0.2794760 0.023284189 1.4443753    64
## [48]  {cat food}                 => {whole milk}          0.008845958
0.3799127 0.023284189 1.4868448    87
## [49]  {specialty chocolate}      => {soda}                0.006304016
0.2073579 0.030401627 1.1891338    62
## [50]  {specialty chocolate}      => {other vegetables}    0.006100661
0.2006689 0.030401627 1.0370881    60
## [51]  {specialty chocolate}      => {whole milk}          0.008032537
0.2642140 0.030401627 1.0340410    79
## [52]  {meat}                     => {sausage}             0.005287239
0.2047244 0.025826131 2.1790742    52
## [53]  {meat}                     => {soda}                0.005490595
0.2125984 0.025826131 1.2191869    54
## [54]  {meat}                     => {yogurt}              0.005287239
0.2047244 0.025826131 1.4675398    52
## [55]  {meat}                     => {rolls/buns}          0.006914082
0.2677165 0.025826131 1.4554959    68
## [56]  {meat}                     => {other vegetables}    0.009964413
0.3858268 0.025826131 1.9940128    98
## [57]  {meat}                     => {whole milk}          0.009964413
0.3858268 0.025826131 1.5099906    98
## [58]  {frozen meals}             => {soda}                0.006202339
0.2186380 0.028368073 1.2538220    61
## [59]  {frozen meals}             => {yogurt}              0.006202339
0.2186380 0.028368073 1.5672774    61
## [60]  {frozen meals}             => {other vegetables}    0.007524148
0.2652330 0.028368073 1.3707653    74
```

```
## [61]  {frozen meals}              => {whole milk}          0.009862735
0.3476703 0.028368073 1.3606593    97
## [62]  {hard cheese}               => {sausage}             0.005185562
0.2116183 0.024504321 2.2524519    51
## [63]  {hard cheese}               => {root vegetables}     0.005592272
0.2282158 0.024504321 2.0937519    55
## [64]  {hard cheese}               => {yogurt}              0.006405694
0.2614108 0.024504321 1.8738886    63
## [65]  {hard cheese}               => {rolls/buns}          0.005897306
0.2406639 0.024504321 1.3084187    58
## [66]  {hard cheese}               => {other vegetables}    0.009456024
0.3858921 0.024504321 1.9943505    93
## [67]  {hard cheese}               => {whole milk}          0.010066090
0.4107884 0.024504321 1.6076815    99
## [68]  {butter milk}               => {yogurt}              0.008540925
0.3054545 0.027961362 2.1896104    84
## [69]  {butter milk}               => {rolls/buns}          0.007625826
0.2727273 0.027961362 1.4827378    75
## [70]  {butter milk}               => {other vegetables}    0.010371124
0.3709091 0.027961362 1.9169159   102
## [71]  {butter milk}               => {whole milk}          0.011591256
0.4145455 0.027961362 1.6223854   114
## [72]  {candy}                     => {soda}                0.008642603
0.2891156 0.029893238 1.6579897    85
## [73]  {candy}                     => {rolls/buns}          0.007117438
0.2380952 0.029893238 1.2944537    70
## [74]  {candy}                     => {other vegetables}    0.006914082
0.2312925 0.029893238 1.1953557    68
## [75]  {candy}                     => {whole milk}          0.008235892
0.2755102 0.029893238 1.0782502    81
## [76]  {ham}                       => {tropical fruit}      0.005388917
0.2070312 0.026029487 1.9730158    53
## [77]  {ham}                       => {yogurt}              0.006710727
0.2578125 0.026029487 1.8480947    66
## [78]  {ham}                       => {rolls/buns}          0.006914082
0.2656250 0.026029487 1.4441249    68
## [79]  {ham}                       => {other vegetables}    0.009150991
0.3515625 0.026029487 1.8169297    90
## [80]  {ham}                       => {whole milk}          0.011489578
0.4414062 0.026029487 1.7275091   113
## [81]  {sliced cheese}             => {sausage}             0.007015760
0.2863071 0.024504321 3.0474349    69
## [82]  {sliced cheese}             => {tropical fruit}      0.005287239
0.2157676 0.024504321 2.0562739    52
## [83]  {sliced cheese}             => {root vegetables}     0.005592272
0.2282158 0.024504321 2.0937519    55
## [84]  {sliced cheese}             => {soda}                0.005083884
0.2074689 0.024504321 1.1897705    50
## [85]  {sliced cheese}             => {yogurt}              0.008032537
0.3278008 0.024504321 2.3497968    79
```

```
## [86]  {sliced cheese}           => {rolls/buns}          0.007625826
0.3112033 0.024504321 1.6919208    75
## [87]  {sliced cheese}           => {other vegetables}    0.009049314
0.3692946 0.024504321 1.9085720    89
## [88]  {sliced cheese}           => {whole milk}          0.010777834
0.4398340 0.024504321 1.7213560   106
## [89]  {UHT-milk}                => {bottled water}       0.007320793
0.2188450 0.033451957 1.9800740    72
## [90]  {UHT-milk}                => {soda}                0.007625826
0.2279635 0.033451957 1.3073010    75
## [91]  {UHT-milk}                => {yogurt}              0.007422471
0.2218845 0.033451957 1.5905496    73
## [92]  {UHT-milk}                => {other vegetables}    0.008134215
0.2431611 0.033451957 1.2566944    80
## [93]  {oil}                     => {root vegetables}     0.007015760
0.2500000 0.028063040 2.2936101    69
## [94]  {oil}                     => {other vegetables}    0.009964413
0.3550725 0.028063040 1.8350697    98
## [95]  {oil}                     => {whole milk}          0.011286223
0.4021739 0.028063040 1.5739675   111
## [96]  {onions}                  => {root vegetables}     0.009456024
0.3049180 0.031011693 2.7974523    93
## [97]  {onions}                  => {yogurt}              0.007219115
0.2327869 0.031011693 1.6687019    71
## [98]  {onions}                  => {rolls/buns}          0.006812405
0.2196721 0.031011693 1.1942927    67
## [99]  {onions}                  => {other vegetables}    0.014234875
0.4590164 0.031011693 2.3722681   140
## [100] {onions}                  => {whole milk}          0.012099644
0.3901639 0.031011693 1.5269647   119
## [101] {berries}                 => {whipped/sour cream}  0.009049314
0.2721713 0.033248602 3.7968855    89
## [102] {berries}                 => {tropical fruit}      0.006710727
0.2018349 0.033248602 1.9234941    66
## [103] {berries}                 => {soda}                0.007320793
0.2201835 0.033248602 1.2626849    72
## [104] {berries}                 => {yogurt}              0.010574479
0.3180428 0.033248602 2.2798477   104
## [105] {berries}                 => {other vegetables}    0.010269446
0.3088685 0.033248602 1.5962805   101
## [106] {berries}                 => {whole milk}          0.011794611
0.3547401 0.033248602 1.3883281   116
## [107] {hamburger meat}          => {rolls/buns}          0.008642603
0.2599388 0.033248602 1.4132109    85
## [108] {hamburger meat}          => {other vegetables}    0.013828165
0.4159021 0.033248602 2.1494470   136
## [109] {hamburger meat}          => {whole milk}          0.014743264
0.4434251 0.033248602 1.7354101   145
## [110] {hygiene articles}        => {tropical fruit}      0.006710727
0.2037037 0.032943569 1.9413042    66
```

```
## [111] {hygiene articles}         => {soda}                  0.007015760
0.2129630 0.032943569 1.2212774    69
## [112] {hygiene articles}         => {yogurt}                0.007320793
0.2222222 0.032943569 1.5929705    72
## [113] {hygiene articles}         => {other vegetables}      0.009557702
0.2901235 0.032943569 1.4994032    94
## [114] {hygiene articles}         => {whole milk}            0.012811388
0.3888889 0.032943569 1.5219746   126
## [115] {salty snack}              => {soda}                  0.009354347
0.2473118 0.037824098 1.4182576    92
## [116] {salty snack}              => {other vegetables}      0.010777834
0.2849462 0.037824098 1.4726465   106
## [117] {salty snack}              => {whole milk}            0.011184545
0.2956989 0.037824098 1.1572618   110
## [118] {sugar}                    => {soda}                  0.007320793
0.2162162 0.033858668 1.2399338    72
## [119] {sugar}                    => {yogurt}                0.006914082
0.2042042 0.033858668 1.4638107    68
## [120] {sugar}                    => {rolls/buns}            0.007015760
0.2072072 0.033858668 1.1265245    69
## [121] {sugar}                    => {other vegetables}      0.010777834
0.3183183 0.033858668 1.6451186   106
## [122] {sugar}                    => {whole milk}            0.015048297
0.4444444 0.033858668 1.7393996   148
## [123] {waffles}                  => {soda}                  0.009557702
0.2486772 0.038434164 1.4260879    94
## [124] {waffles}                  => {rolls/buns}            0.009150991
0.2380952 0.038434164 1.2944537    90
## [125] {waffles}                  => {other vegetables}      0.010066090
0.2619048 0.038434164 1.3535645    99
## [126] {waffles}                  => {whole milk}            0.012709710
0.3306878 0.038434164 1.2941961   125
## [127] {long life bakery product} => {soda}                  0.007625826
0.2038043 0.037417387 1.1687555    75
## [128] {long life bakery product} => {yogurt}                0.008744281
0.2336957 0.037417387 1.6752163    86
## [129] {long life bakery product} => {rolls/buns}            0.007930859
0.2119565 0.037417387 1.1523452    78
## [130] {long life bakery product} => {other vegetables}      0.010676157
0.2853261 0.037417387 1.4746096   105
## [131] {long life bakery product} => {whole milk}            0.013523132
0.3614130 0.037417387 1.4144438   133
## [132] {dessert}                  => {soda}                  0.009862735
0.2657534 0.037112354 1.5240145    97
## [133] {dessert}                  => {yogurt}                0.009862735
0.2657534 0.037112354 1.9050182    97
## [134] {dessert}                  => {other vegetables}      0.011591256
0.3123288 0.037112354 1.6141636   114
## [135] {dessert}                  => {whole milk}            0.013726487
0.3698630 0.037112354 1.4475140   135
```

```
## [136] {cream cheese}            => {yogurt}              0.012404677
0.3128205 0.039654296 2.2424123   122
## [137] {cream cheese}            => {rolls/buns}          0.009964413
0.2512821 0.039654296 1.3661465    98
## [138] {cream cheese}            => {other vegetables}    0.013726487
0.3461538 0.039654296 1.7889769   135
## [139] {cream cheese}            => {whole milk}          0.016471784
0.4153846 0.039654296 1.6256696   162
## [140] {chicken}                 => {root vegetables}     0.010879512
0.2535545 0.042907982 2.3262206   107
## [141] {chicken}                 => {rolls/buns}          0.009659380
0.2251185 0.042907982 1.2239029    95
## [142] {chicken}                 => {other vegetables}    0.017895272
0.4170616 0.042907982 2.1554393   176
## [143] {chicken}                 => {whole milk}          0.017590239
0.4099526 0.042907982 1.6044106   173
## [144] {white bread}             => {tropical fruit}      0.008744281
0.2077295 0.042094560 1.9796699    86
## [145] {white bread}             => {soda}                0.010269446
0.2439614 0.042094560 1.3990437   101
## [146] {white bread}             => {yogurt}              0.009049314
0.2149758 0.042094560 1.5410258    89
## [147] {white bread}             => {other vegetables}    0.013726487
0.3260870 0.042094560 1.6852681   135
## [148] {white bread}             => {whole milk}          0.017081851
0.4057971 0.042094560 1.5881474   168
## [149] {chocolate}               => {soda}                0.013523132
0.2725410 0.049618709 1.5629391   133
## [150] {chocolate}               => {rolls/buns}          0.011794611
0.2377049 0.049618709 1.2923316   116
## [151] {chocolate}               => {other vegetables}    0.012709710
0.2561475 0.049618709 1.3238103   125
## [152] {chocolate}               => {whole milk}          0.016675140
0.3360656 0.049618709 1.3152427   164
## [153] {coffee}                  => {other vegetables}    0.013421454
0.2311734 0.058057956 1.1947400   132
## [154] {coffee}                  => {whole milk}          0.018708693
0.3222417 0.058057956 1.2611408   184
## [155] {frozen vegetables}       => {root vegetables}     0.011591256
0.2410148 0.048093543 2.2111759   114
## [156] {frozen vegetables}       => {yogurt}              0.012404677
0.2579281 0.048093543 1.8489235   122
## [157] {frozen vegetables}       => {rolls/buns}          0.010167768
0.2114165 0.048093543 1.1494092   100
## [158] {frozen vegetables}       => {other vegetables}    0.017793594
0.3699789 0.048093543 1.9121083   175
## [159] {frozen vegetables}       => {whole milk}          0.020437214
0.4249471 0.048093543 1.6630940   201
## [160] {beef}                    => {root vegetables}     0.017386884
0.3313953 0.052465684 3.0403668   171
```

```
## [161] {beef}                        => {yogurt}                0.011692933
0.2228682 0.052465684 1.5976012   115
## [162] {beef}                        => {rolls/buns}            0.013624809
0.2596899 0.052465684 1.4118576   134
## [163] {beef}                        => {other vegetables}      0.019725470
0.3759690 0.052465684 1.9430662   194
## [164] {beef}                        => {whole milk}            0.021250635
0.4050388 0.052465684 1.5851795   209
## [165] {curd}                        => {root vegetables}       0.010879512
0.2041985 0.053279105 1.8734067   107
## [166] {curd}                        => {yogurt}                0.017285206
0.3244275 0.053279105 2.3256154   170
## [167] {curd}                        => {other vegetables}      0.017183528
0.3225191 0.053279105 1.6668288   169
## [168] {curd}                        => {whole milk}            0.026131164
0.4904580 0.053279105 1.9194805   257
## [169] {napkins}                     => {soda}                  0.011997966
0.2291262 0.052364006 1.3139687   118
## [170] {napkins}                     => {yogurt}                0.012302999
0.2349515 0.052364006 1.6842183   121
## [171] {napkins}                     => {rolls/buns}            0.011692933
0.2233010 0.052364006 1.2140216   115
## [172] {napkins}                     => {other vegetables}      0.014438231
0.2757282 0.052364006 1.4250060   142
## [173] {napkins}                     => {whole milk}            0.019725470
0.3766990 0.052364006 1.4742678   194
## [174] {pork}                        => {root vegetables}       0.013624809
0.2363316 0.057651246 2.1682099   134
## [175] {pork}                        => {soda}                  0.011896289
0.2063492 0.057651246 1.1833495   117
## [176] {pork}                        => {other vegetables}      0.021657346
0.3756614 0.057651246 1.9414764   213
## [177] {pork}                        => {whole milk}            0.022165735
0.3844797 0.057651246 1.5047187   218
## [178] {frankfurter}                 => {rolls/buns}            0.019217082
0.3258621 0.058973055 1.7716161   189
## [179] {frankfurter}                 => {other vegetables}      0.016471784
0.2793103 0.058973055 1.4435193   162
## [180] {frankfurter}                 => {whole milk}            0.020538892
0.3482759 0.058973055 1.3630295   202
## [181] {bottled beer}                => {soda}                  0.016980173
0.2108586 0.080528724 1.2092094   167
## [182] {bottled beer}                => {other vegetables}      0.016166751
0.2007576 0.080528724 1.0375464   159
## [183] {bottled beer}                => {whole milk}            0.020437214
0.2537879 0.080528724 0.9932367   201
## [184] {brown bread}                 => {yogurt}                0.014539908
0.2241379 0.064870361 1.6067030   143
## [185] {brown bread}                 => {other vegetables}      0.018708693
0.2884013 0.064870361 1.4905025   184
```

```
## [186] {brown bread}              => {whole milk}          0.025216065
0.3887147 0.064870361 1.5212930   248
## [187] {margarine}                => {yogurt}              0.014234875
0.2430556 0.058566345 1.7423115   140
## [188] {margarine}                => {rolls/buns}          0.014743264
0.2517361 0.058566345 1.3686151   145
## [189] {margarine}                => {other vegetables}    0.019725470
0.3368056 0.058566345 1.7406635   194
## [190] {margarine}                => {whole milk}          0.024199288
0.4131944 0.058566345 1.6170980   238
## [191] {butter}                   => {root vegetables}     0.012913066
0.2330275 0.055414337 2.1378971   127
## [192] {butter}                   => {yogurt}              0.014641586
0.2642202 0.055414337 1.8940273   144
## [193] {butter}                   => {rolls/buns}          0.013421454
0.2422018 0.055414337 1.3167800   132
## [194] {butter}                   => {other vegetables}    0.020030503
0.3614679 0.055414337 1.8681223   197
## [195] {butter}                   => {whole milk}          0.027554652
0.4972477 0.055414337 1.9460530   271
## [196] {newspapers}               => {rolls/buns}          0.019725470
0.2471338 0.079816980 1.3435934   194
## [197] {newspapers}               => {other vegetables}    0.019318760
0.2420382 0.079816980 1.2508912   190
## [198] {newspapers}               => {whole milk}          0.027351296
0.3426752 0.079816980 1.3411103   269
## [199] {domestic eggs}            => {root vegetables}     0.014336553
0.2259615 0.063446873 2.0730706   141
## [200] {domestic eggs}            => {yogurt}              0.014336553
0.2259615 0.063446873 1.6197753   141
## [201] {domestic eggs}            => {rolls/buns}          0.015658363
0.2467949 0.063446873 1.3417510   154
## [202] {domestic eggs}            => {other vegetables}    0.022267412
0.3509615 0.063446873 1.8138238   219
## [203] {domestic eggs}            => {whole milk}          0.029994916
0.4727564 0.063446873 1.8502027   295
## [204] {fruit/vegetable juice}    => {soda}                0.018403660
0.2545710 0.072292832 1.4598869   181
## [205] {fruit/vegetable juice}    => {yogurt}              0.018708693
0.2587904 0.072292832 1.8551049   184
## [206] {fruit/vegetable juice}    => {rolls/buns}          0.014539908
0.2011252 0.072292832 1.0934583   143
## [207] {fruit/vegetable juice}    => {other vegetables}    0.021047280
0.2911392 0.072292832 1.5046529   207
## [208] {fruit/vegetable juice}    => {whole milk}          0.026639553
0.3684951 0.072292832 1.4421604   262
## [209] {whipped/sour cream}       => {root vegetables}     0.017081851
0.2382979 0.071682766 2.1862496   168
## [210] {whipped/sour cream}       => {yogurt}              0.020742247
0.2893617 0.071682766 2.0742510   204
```

```
## [211] {whipped/sour cream}       => {rolls/buns}            0.014641586
0.2042553 0.071682766 1.1104760   144
## [212] {whipped/sour cream}       => {other vegetables}      0.028876462
0.4028369 0.071682766 2.0819237   284
## [213] {whipped/sour cream}       => {whole milk}            0.032231825
0.4496454 0.071682766 1.7597542   317
## [214] {pip fruit}                => {tropical fruit}        0.020437214
0.2701613 0.075648195 2.5746476   201
## [215] {pip fruit}                => {root vegetables}       0.015556685
0.2056452 0.075648195 1.8866793   153
## [216] {pip fruit}                => {yogurt}                0.017996950
0.2379032 0.075648195 1.7053777   177
## [217] {pip fruit}                => {other vegetables}      0.026131164
0.3454301 0.075648195 1.7852365   257
## [218] {pip fruit}                => {whole milk}            0.030096594
0.3978495 0.075648195 1.5570432   296
## [219] {pastry}                   => {soda}                  0.021047280
0.2365714 0.088967972 1.3566647   207
## [220] {pastry}                   => {rolls/buns}            0.020945602
0.2354286 0.088967972 1.2799558   206
## [221] {pastry}                   => {other vegetables}      0.022572445
0.2537143 0.088967972 1.3112349   222
## [222] {pastry}                   => {whole milk}            0.033248602
0.3737143 0.088967972 1.4625865   327
## [223] {citrus fruit}             => {tropical fruit}        0.019928826
0.2407862 0.082765633 2.2947022   196
## [224] {citrus fruit}             => {root vegetables}       0.017691917
0.2137592 0.082765633 1.9611211   174
## [225] {citrus fruit}             => {yogurt}                0.021657346
0.2616708 0.082765633 1.8757521   213
## [226] {citrus fruit}             => {rolls/buns}            0.016776817
0.2027027 0.082765633 1.1020349   165
## [227] {citrus fruit}             => {other vegetables}      0.028876462
0.3488943 0.082765633 1.8031403   284
## [228] {citrus fruit}             => {whole milk}            0.030503305
0.3685504 0.082765633 1.4423768   300
## [229] {shopping bags}            => {soda}                  0.024605999
0.2497420 0.098525674 1.4321939   242
## [230] {shopping bags}            => {other vegetables}      0.023182511
0.2352941 0.098525674 1.2160366   228
## [231] {shopping bags}            => {whole milk}            0.024504321
0.2487100 0.098525674 0.9733637   241
## [232] {sausage}                  => {soda}                  0.024300966
0.2586580 0.093950178 1.4833245   239
## [233] {sausage}                  => {yogurt}                0.019623793
0.2088745 0.093950178 1.4972889   193
## [234] {sausage}                  => {rolls/buns}            0.030604982
0.3257576 0.093950178 1.7710480   301
## [235] {sausage}                  => {other vegetables}      0.026944586
0.2867965 0.093950178 1.4822091   265
```

```
## [236] {sausage}                  => {whole milk}          0.029893238
0.3181818 0.093950178 1.2452520   294
## [237] {bottled water}            => {soda}                0.028978139
0.2621895 0.110523640 1.5035766   285
## [238] {bottled water}            => {yogurt}              0.022979156
0.2079117 0.110523640 1.4903873   226
## [239] {bottled water}            => {rolls/buns}          0.024199288
0.2189512 0.110523640 1.1903734   238
## [240] {bottled water}            => {other vegetables}    0.024809354
0.2244710 0.110523640 1.1601012   244
## [241] {bottled water}            => {whole milk}          0.034367056
0.3109476 0.110523640 1.2169396   338
## [242] {tropical fruit}           => {root vegetables}     0.021047280
0.2005814 0.104931368 1.8402220   207
## [243] {tropical fruit}           => {yogurt}              0.029283172
0.2790698 0.104931368 2.0004746   288
## [244] {yogurt}                   => {tropical fruit}      0.029283172
0.2099125 0.139501779 2.0004746   288
## [245] {tropical fruit}           => {rolls/buns}          0.024605999
0.2344961 0.104931368 1.2748863   242
## [246] {tropical fruit}           => {other vegetables}    0.035892222
0.3420543 0.104931368 1.7677896   353
## [247] {tropical fruit}           => {whole milk}          0.042297916
0.4031008 0.104931368 1.5775950   416
## [248] {root vegetables}          => {yogurt}              0.025826131
0.2369403 0.108998475 1.6984751   254
## [249] {root vegetables}          => {rolls/buns}          0.024300966
0.2229478 0.108998475 1.2121013   239
## [250] {root vegetables}          => {other vegetables}    0.047381800
0.4347015 0.108998475 2.2466049   466
## [251] {other vegetables}         => {root vegetables}     0.047381800
0.2448765 0.193492628 2.2466049   466
## [252] {root vegetables}          => {whole milk}          0.048906965
0.4486940 0.108998475 1.7560310   481
## [253] {soda}                     => {rolls/buns}          0.038332486
0.2198251 0.174377224 1.1951242   377
## [254] {rolls/buns}               => {soda}                0.038332486
0.2084024 0.183934926 1.1951242   377
## [255] {soda}                     => {whole milk}          0.040061007
0.2297376 0.174377224 0.8991124   394
## [256] {yogurt}                   => {rolls/buns}          0.034367056
0.2463557 0.139501779 1.3393633   338
## [257] {yogurt}                   => {other vegetables}    0.043416370
0.3112245 0.139501779 1.6084566   427
## [258] {other vegetables}         => {yogurt}              0.043416370
0.2243826 0.193492628 1.6084566   427
## [259] {yogurt}                   => {whole milk}          0.056024403
0.4016035 0.139501779 1.5717351   551
## [260] {whole milk}               => {yogurt}              0.056024403
0.2192598 0.255516014 1.5717351   551
```

```
## [261] {rolls/buns}              => {other vegetables}      0.042602949
0.2316197 0.183934926 1.1970465   419
## [262] {other vegetables}        => {rolls/buns}            0.042602949
0.2201787 0.193492628 1.1970465   419
## [263] {rolls/buns}              => {whole milk}            0.056634469
0.3079049 0.183934926 1.2050318   557
## [264] {whole milk}              => {rolls/buns}            0.056634469
0.2216474 0.255516014 1.2050318   557
## [265] {other vegetables}        => {whole milk}            0.074834774
0.3867578 0.193492628 1.5136341   736
## [266] {whole milk}              => {other vegetables}      0.074834774
0.2928770 0.255516014 1.5136341   736
## [267] {oil,
##        other vegetables}        => {whole milk}            0.005083884
0.5102041 0.009964413 1.9967597    50
## [268] {oil,
##        whole milk}              => {other vegetables}      0.005083884
0.4504505 0.011286223 2.3279980    50
## [269] {onions,
##        root vegetables}         => {other vegetables}      0.005693950
0.6021505 0.009456024 3.1120076    56
## [270] {onions,
##        other vegetables}        => {root vegetables}       0.005693950
0.4000000 0.014234875 3.6697761    56
## [271] {onions,
##        other vegetables}        => {whole milk}            0.006609049
0.4642857 0.014234875 1.8170513    65
## [272] {onions,
##        whole milk}              => {other vegetables}      0.006609049
0.5462185 0.012099644 2.8229421    65
## [273] {hamburger meat,
##        other vegetables}        => {whole milk}            0.006304016
0.4558824 0.013828165 1.7841635    62
## [274] {hamburger meat,
##        whole milk}              => {other vegetables}      0.006304016
0.4275862 0.014743264 2.2098320    62
## [275] {hygiene articles,
##        other vegetables}        => {whole milk}            0.005185562
0.5425532 0.009557702 2.1233628    51
## [276] {hygiene articles,
##        whole milk}              => {other vegetables}      0.005185562
0.4047619 0.012811388 2.0918725    51
## [277] {other vegetables,
##        sugar}                   => {whole milk}            0.006304016
0.5849057 0.010777834 2.2891155    62
## [278] {sugar,
##        whole milk}              => {other vegetables}      0.006304016
0.4189189 0.015048297 2.1650381    62
## [279] {long life bakery product,
##        other vegetables}        => {whole milk}            0.005693950
```

```
0.5333333 0.010676157 2.0872795    56
## [280] {long life bakery product,
##        whole milk}                => {other vegetables}      0.005693950
0.4210526 0.013523132 2.1760655    56
## [281] {cream cheese,
##        yogurt}                    => {other vegetables}      0.005287239
0.4262295 0.012404677 2.2028204    52
## [282] {cream cheese,
##        other vegetables}          => {yogurt}                0.005287239
0.3851852 0.013726487 2.7611489    52
## [283] {cream cheese,
##        yogurt}                    => {whole milk}            0.006609049
0.5327869 0.012404677 2.0851409    65
## [284] {cream cheese,
##        whole milk}                => {yogurt}                0.006609049
0.4012346 0.016471784 2.8761968    65
## [285] {cream cheese,
##        other vegetables}          => {whole milk}            0.006710727
0.4888889 0.013726487 1.9133395    66
## [286] {cream cheese,
##        whole milk}                => {other vegetables}      0.006710727
0.4074074 0.016471784 2.1055449    66
## [287] {chicken,
##        root vegetables}           => {other vegetables}      0.005693950
0.5233645 0.010879512 2.7048291    56
## [288] {chicken,
##        other vegetables}          => {root vegetables}       0.005693950
0.3181818 0.017895272 2.9191401    56
## [289] {chicken,
##        root vegetables}           => {whole milk}            0.005998983
0.5514019 0.010879512 2.1579934    59
## [290] {chicken,
##        whole milk}                => {root vegetables}       0.005998983
0.3410405 0.017590239 3.1288554    59
## [291] {chicken,
##        rolls/buns}                => {whole milk}            0.005287239
0.5473684 0.009659380 2.1422079    52
## [292] {chicken,
##        whole milk}                => {rolls/buns}            0.005287239
0.3005780 0.017590239 1.6341542    52
## [293] {chicken,
##        other vegetables}          => {whole milk}            0.008439248
0.4715909 0.017895272 1.8456413    83
## [294] {chicken,
##        whole milk}                => {other vegetables}      0.008439248
0.4797688 0.017590239 2.4795197    83
## [295] {other vegetables,
##        white bread}               => {whole milk}            0.005897306
0.4296296 0.013726487 1.6814196    58
## [296] {white bread,
```

```
##        whole milk}              => {other vegetables}      0.005897306
0.3452381 0.017081851 1.7842442    58
## [297] {chocolate,
##        soda}                    => {whole milk}            0.005083884
0.3759398 0.013523132 1.4712966    50
## [298] {chocolate,
##        whole milk}              => {soda}                  0.005083884
0.3048780 0.016675140 1.7483823    50
## [299] {chocolate,
##        other vegetables}        => {whole milk}            0.005490595
0.4320000 0.012709710 1.6906964    54
## [300] {chocolate,
##        whole milk}              => {other vegetables}      0.005490595
0.3292683 0.016675140 1.7017098    54
## [301] {coffee,
##        yogurt}                  => {whole milk}            0.005083884
0.5208333 0.009761057 2.0383589    50
## [302] {coffee,
##        whole milk}              => {yogurt}                0.005083884
0.2717391 0.018708693 1.9479259    50
## [303] {coffee,
##        other vegetables}        => {whole milk}            0.006405694
0.4772727 0.013421454 1.8678779    63
## [304] {coffee,
##        whole milk}              => {other vegetables}      0.006405694
0.3423913 0.018708693 1.7695315    63
## [305] {frozen vegetables,
##        root vegetables}         => {other vegetables}      0.006100661
0.5263158 0.011591256 2.7200819    60
## [306] {frozen vegetables,
##        other vegetables}        => {root vegetables}       0.006100661
0.3428571 0.017793594 3.1455224    60
## [307] {frozen vegetables,
##        root vegetables}         => {whole milk}            0.006202339
0.5350877 0.011591256 2.0941455    61
## [308] {frozen vegetables,
##        whole milk}              => {root vegetables}       0.006202339
0.3034826 0.020437214 2.7842829    61
## [309] {frozen vegetables,
##        yogurt}                  => {other vegetables}      0.005287239
0.4262295 0.012404677 2.2028204    52
## [310] {frozen vegetables,
##        other vegetables}        => {yogurt}                0.005287239
0.2971429 0.017793594 2.1300292    52
## [311] {frozen vegetables,
##        yogurt}                  => {whole milk}            0.006100661
0.4918033 0.012404677 1.9247454    60
## [312] {frozen vegetables,
##        whole milk}              => {yogurt}                0.006100661
0.2985075 0.020437214 2.1398111    60
```

```
## [313] {frozen vegetables,
##        rolls/buns}              => {whole milk}           0.005083884
0.5000000 0.010167768 1.9568245    50
## [314] {frozen vegetables,
##        whole milk}              => {rolls/buns}           0.005083884
0.2487562 0.020437214 1.3524143    50
## [315] {frozen vegetables,
##        other vegetables}        => {whole milk}           0.009659380
0.5428571 0.017793594 2.1245523    95
## [316] {frozen vegetables,
##        whole milk}              => {other vegetables}     0.009659380
0.4726368 0.020437214 2.4426606    95
## [317] {beef,
##        root vegetables}         => {other vegetables}     0.007930859
0.4561404 0.017386884 2.3574043    78
## [318] {beef,
##        other vegetables}        => {root vegetables}      0.007930859
0.4020619 0.019725470 3.6886925    78
## [319] {beef,
##        root vegetables}         => {whole milk}           0.008032537
0.4619883 0.017386884 1.8080601    79
## [320] {beef,
##        whole milk}              => {root vegetables}      0.008032537
0.3779904 0.021250635 3.4678506    79
## [321] {beef,
##        yogurt}                  => {other vegetables}     0.005185562
0.4434783 0.011692933 2.2919646    51
## [322] {beef,
##        other vegetables}        => {yogurt}               0.005185562
0.2628866 0.019725470 1.8844677    51
## [323] {beef,
##        yogurt}                  => {whole milk}           0.006100661
0.5217391 0.011692933 2.0419038    60
## [324] {beef,
##        whole milk}              => {yogurt}               0.006100661
0.2870813 0.021250635 2.0579045    60
## [325] {beef,
##        rolls/buns}              => {other vegetables}     0.005795628
0.4253731 0.013624809 2.1983945    57
## [326] {beef,
##        other vegetables}        => {rolls/buns}           0.005795628
0.2938144 0.019725470 1.5973825    57
## [327] {beef,
##        rolls/buns}              => {whole milk}           0.006812405
0.5000000 0.013624809 1.9568245    67
## [328] {beef,
##        whole milk}              => {rolls/buns}           0.006812405
0.3205742 0.021250635 1.7428673    67
## [329] {beef,
##        other vegetables}        => {whole milk}           0.009252669
```

```
0.4690722 0.019725470 1.8357838    91
## [330] {beef,
##        whole milk}                => {other vegetables}      0.009252669
0.4354067 0.021250635 2.2502495    91
## [331] {curd,
##        whipped/sour cream}        => {whole milk}            0.005897306
0.5631068 0.010472801 2.2038024    58
## [332] {curd,
##        whole milk}                => {whipped/sour cream}    0.005897306
0.2256809 0.026131164 3.1483291    58
## [333] {curd,
##        tropical fruit}            => {yogurt}                0.005287239
0.5148515 0.010269446 3.6906446    52
## [334] {curd,
##        yogurt}                    => {tropical fruit}        0.005287239
0.3058824 0.017285206 2.9150707    52
## [335] {curd,
##        tropical fruit}            => {other vegetables}      0.005287239
0.5148515 0.010269446 2.6608326    52
## [336] {curd,
##        other vegetables}          => {tropical fruit}        0.005287239
0.3076923 0.017183528 2.9323196    52
## [337] {curd,
##        tropical fruit}            => {whole milk}            0.006507372
0.6336634 0.010269446 2.4799360    64
## [338] {curd,
##        whole milk}                => {tropical fruit}        0.006507372
0.2490272 0.026131164 2.3732392    64
## [339] {curd,
##        root vegetables}           => {other vegetables}      0.005490595
0.5046729 0.010879512 2.6082280    54
## [340] {curd,
##        other vegetables}          => {root vegetables}       0.005490595
0.3195266 0.017183528 2.9314780    54
## [341] {curd,
##        root vegetables}           => {whole milk}            0.006202339
0.5700935 0.010879512 2.2311457    61
## [342] {curd,
##        whole milk}                => {root vegetables}       0.006202339
0.2373541 0.026131164 2.1775909    61
## [343] {curd,
##        yogurt}                    => {other vegetables}      0.006100661
0.3529412 0.017285206 1.8240549    60
## [344] {curd,
##        other vegetables}          => {yogurt}                0.006100661
0.3550296 0.017183528 2.5449825    60
## [345] {curd,
##        yogurt}                    => {whole milk}            0.010066090
0.5823529 0.017285206 2.2791250    99
## [346] {curd,
```

```
##        whole milk}              => {yogurt}              0.010066090
0.3852140 0.026131164 2.7613555    99
## [347] {curd,
##        rolls/buns}              => {whole milk}          0.005897306
0.5858586 0.010066090 2.2928449    58
## [348] {curd,
##        whole milk}              => {rolls/buns}          0.005897306
0.2256809 0.026131164 1.2269607    58
## [349] {curd,
##        other vegetables}        => {whole milk}          0.009862735
0.5739645 0.017183528 2.2462956    97
## [350] {curd,
##        whole milk}              => {other vegetables}    0.009862735
0.3774319 0.026131164 1.9506268    97
## [351] {napkins,
##        yogurt}                  => {whole milk}          0.006100661
0.4958678 0.012302999 1.9406524    60
## [352] {napkins,
##        whole milk}              => {yogurt}              0.006100661
0.3092784 0.019725470 2.2170208    60
## [353] {napkins,
##        rolls/buns}              => {whole milk}          0.005287239
0.4521739 0.011692933 1.7696500    52
## [354] {napkins,
##        whole milk}              => {rolls/buns}          0.005287239
0.2680412 0.019725470 1.4572612    52
## [355] {napkins,
##        other vegetables}        => {whole milk}          0.006812405
0.4718310 0.014438231 1.8465809    67
## [356] {napkins,
##        whole milk}              => {other vegetables}    0.006812405
0.3453608 0.019725470 1.7848785    67
## [357] {pork,
##        root vegetables}         => {other vegetables}    0.007015760
0.5149254 0.013624809 2.6612144    69
## [358] {other vegetables,
##        pork}                    => {root vegetables}     0.007015760
0.3239437 0.021657346 2.9720018    69
## [359] {pork,
##        root vegetables}         => {whole milk}          0.006812405
0.5000000 0.013624809 1.9568245    67
## [360] {pork,
##        whole milk}              => {root vegetables}     0.006812405
0.3073394 0.022165735 2.8196674    67
## [361] {pork,
##        rolls/buns}              => {other vegetables}    0.005592272
0.4954955 0.011286223 2.5607978    55
## [362] {other vegetables,
##        pork}                    => {rolls/buns}          0.005592272
0.2582160 0.021657346 1.4038441    55
```

```
## [363] {pork,
##        rolls/buns}               => {whole milk}            0.006202339
0.5495495 0.011286223 2.1507441    61
## [364] {pork,
##        whole milk}               => {rolls/buns}            0.006202339
0.2798165 0.022165735 1.5212799    61
## [365] {other vegetables,
##        pork}                     => {whole milk}            0.010167768
0.4694836 0.021657346 1.8373939   100
## [366] {pork,
##        whole milk}               => {other vegetables}      0.010167768
0.4587156 0.022165735 2.3707136   100
## [367] {frankfurter,
##        tropical fruit}           => {whole milk}            0.005185562
0.5483871 0.009456024 2.1461946    51
## [368] {frankfurter,
##        whole milk}               => {tropical fruit}        0.005185562
0.2524752 0.020538892 2.4060989    51
## [369] {frankfurter,
##        root vegetables}          => {whole milk}            0.005083884
0.5000000 0.010167768 1.9568245    50
## [370] {frankfurter,
##        whole milk}               => {root vegetables}       0.005083884
0.2475248 0.020538892 2.2709011    50
## [371] {frankfurter,
##        yogurt}                   => {whole milk}            0.006202339
0.5545455 0.011184545 2.1702963    61
## [372] {frankfurter,
##        whole milk}               => {yogurt}                0.006202339
0.3019802 0.020538892 2.1647050    61
## [373] {frankfurter,
##        rolls/buns}               => {other vegetables}      0.005592272
0.2910053 0.019217082 1.5039606    55
## [374] {frankfurter,
##        other vegetables}         => {rolls/buns}            0.005592272
0.3395062 0.016471784 1.8457950    55
## [375] {frankfurter,
##        rolls/buns}               => {whole milk}            0.005998983
0.3121693 0.019217082 1.2217211    59
## [376] {frankfurter,
##        whole milk}               => {rolls/buns}            0.005998983
0.2920792 0.020538892 1.5879486    59
## [377] {frankfurter,
##        other vegetables}         => {whole milk}            0.007625826
0.4629630 0.016471784 1.8118745    75
## [378] {frankfurter,
##        whole milk}               => {other vegetables}      0.007625826
0.3712871 0.020538892 1.9188696    75
## [379] {bottled beer,
##        bottled water}            => {soda}                  0.005083884
```

```
0.3225806 0.015760041 1.8499013    50
## [380] {bottled beer,
##        soda}                     => {bottled water}         0.005083884
0.2994012 0.016980173 2.7089336    50
## [381] {bottled beer,
##        bottled water}            => {whole milk}            0.006100661
0.3870968 0.015760041 1.5149609    60
## [382] {bottled beer,
##        whole milk}               => {bottled water}         0.006100661
0.2985075 0.020437214 2.7008472    60
## [383] {bottled beer,
##        yogurt}                   => {whole milk}            0.005185562
0.5604396 0.009252669 2.1933637    51
## [384] {bottled beer,
##        whole milk}               => {yogurt}                0.005185562
0.2537313 0.020437214 1.8188395    51
## [385] {bottled beer,
##        rolls/buns}               => {whole milk}            0.005388917
0.3955224 0.013624809 1.5479358    53
## [386] {bottled beer,
##        whole milk}               => {rolls/buns}            0.005388917
0.2636816 0.020437214 1.4335591    53
## [387] {bottled beer,
##        other vegetables}         => {whole milk}            0.007625826
0.4716981 0.016166751 1.8460609    75
## [388] {bottled beer,
##        whole milk}               => {other vegetables}      0.007625826
0.3731343 0.020437214 1.9284162    75
## [389] {brown bread,
##        tropical fruit}           => {whole milk}            0.005693950
0.5333333 0.010676157 2.0872795    56
## [390] {brown bread,
##        whole milk}               => {tropical fruit}        0.005693950
0.2258065 0.025216065 2.1519442    56
## [391] {brown bread,
##        root vegetables}          => {whole milk}            0.005693950
0.5600000 0.010167768 2.1916435    56
## [392] {brown bread,
##        whole milk}               => {root vegetables}       0.005693950
0.2258065 0.025216065 2.0716478    56
## [393] {brown bread,
##        soda}                     => {whole milk}            0.005083884
0.4032258 0.012608033 1.5780843    50
## [394] {brown bread,
##        whole milk}               => {soda}                  0.005083884
0.2016129 0.025216065 1.1561883    50
## [395] {brown bread,
##        yogurt}                   => {other vegetables}      0.005185562
0.3566434 0.014539908 1.8431883    51
## [396] {brown bread,
```

```
##         other vegetables}          => {yogurt}                 0.005185562
0.2771739 0.018708693 1.9868844    51
## [397] {brown bread,
##         yogurt}                    => {whole milk}             0.007117438
0.4895105 0.014539908 1.9157723    70
## [398] {brown bread,
##         whole milk}                => {yogurt}                 0.007117438
0.2822581 0.025216065 2.0233295    70
## [399] {brown bread,
##         rolls/buns}                => {whole milk}             0.005287239
0.4193548 0.012608033 1.6412077    52
## [400] {brown bread,
##         whole milk}                => {rolls/buns}             0.005287239
0.2096774 0.025216065 1.1399544    52
## [401] {brown bread,
##         other vegetables}          => {whole milk}             0.009354347
0.5000000 0.018708693 1.9568245    92
## [402] {brown bread,
##         whole milk}                => {other vegetables}       0.009354347
0.3709677 0.025216065 1.9172190    92
## [403] {domestic eggs,
##         margarine}                 => {whole milk}             0.005185562
0.6219512 0.008337570 2.4340988    51
## [404] {margarine,
##         whole milk}                => {domestic eggs}          0.005185562
0.2142857 0.024199288 3.3774038    51
## [405] {margarine,
##         root vegetables}           => {other vegetables}       0.005897306
0.5321101 0.011082867 2.7500277    58
## [406] {margarine,
##         other vegetables}          => {root vegetables}        0.005897306
0.2989691 0.019725470 2.7428739    58
## [407] {margarine,
##         yogurt}                    => {other vegetables}       0.005693950
0.4000000 0.014234875 2.0672622    56
## [408] {margarine,
##         other vegetables}          => {yogurt}                 0.005693950
0.2886598 0.019725470 2.0692194    56
## [409] {margarine,
##         yogurt}                    => {whole milk}             0.007015760
0.4928571 0.014234875 1.9288699    69
## [410] {margarine,
##         whole milk}                => {yogurt}                 0.007015760
0.2899160 0.024199288 2.0782241    69
## [411] {margarine,
##         rolls/buns}                => {other vegetables}       0.005185562
0.3517241 0.014743264 1.8177651    51
## [412] {margarine,
##         other vegetables}          => {rolls/buns}             0.005185562
0.2628866 0.019725470 1.4292370    51
```

```
## [413] {margarine,
##        rolls/buns}              => {whole milk}           0.007930859
0.5379310 0.014743264 2.1052733    78
## [414] {margarine,
##        whole milk}              => {rolls/buns}           0.007930859
0.3277311 0.024199288 1.7817774    78
## [415] {margarine,
##        other vegetables}        => {whole milk}           0.009252669
0.4690722 0.019725470 1.8357838    91
## [416] {margarine,
##        whole milk}              => {other vegetables}     0.009252669
0.3823529 0.024199288 1.9760595    91
## [417] {butter,
##        domestic eggs}           => {whole milk}           0.005998983
0.6210526 0.009659380 2.4305820    59
## [418] {butter,
##        whole milk}              => {domestic eggs}        0.005998983
0.2177122 0.027554652 3.4314091    59
## [419] {butter,
##        whipped/sour cream}      => {other vegetables}     0.005795628
0.5700000 0.010167768 2.9458487    57
## [420] {butter,
##        other vegetables}        => {whipped/sour cream}   0.005795628
0.2893401 0.020030503 4.0363970    57
## [421] {other vegetables,
##        whipped/sour cream}      => {butter}               0.005795628
0.2007042 0.028876462 3.6218827    57
## [422] {butter,
##        whipped/sour cream}      => {whole milk}           0.006710727
0.6600000 0.010167768 2.5830084    66
## [423] {butter,
##        whole milk}              => {whipped/sour cream}   0.006710727
0.2435424 0.027554652 3.3975033    66
## [424] {whipped/sour cream,
##        whole milk}              => {butter}               0.006710727
0.2082019 0.032231825 3.7571846    66
## [425] {butter,
##        citrus fruit}            => {whole milk}           0.005083884
0.5555556 0.009150991 2.1742495    50
## [426] {bottled water,
##        butter}                  => {whole milk}           0.005388917
0.6022727 0.008947636 2.3570841    53
## [427] {butter,
##        tropical fruit}          => {other vegetables}     0.005490595
0.5510204 0.009964413 2.8477592    54
## [428] {butter,
##        other vegetables}        => {tropical fruit}       0.005490595
0.2741117 0.020030503 2.6122949    54
## [429] {butter,
##        tropical fruit}          => {whole milk}           0.006202339
```

```
0.6224490 0.009964413 2.4360468    61
## [430] {butter,
##        whole milk}              => {tropical fruit}        0.006202339
0.2250923 0.027554652 2.1451379    61
## [431] {butter,
##        root vegetables}         => {other vegetables}      0.006609049
0.5118110 0.012913066 2.6451190    65
## [432] {butter,
##        other vegetables}        => {root vegetables}       0.006609049
0.3299492 0.020030503 3.0270996    65
## [433] {butter,
##        root vegetables}         => {whole milk}            0.008235892
0.6377953 0.012913066 2.4961069    81
## [434] {butter,
##        whole milk}              => {root vegetables}       0.008235892
0.2988930 0.027554652 2.7421759    81
## [435] {butter,
##        yogurt}                  => {other vegetables}      0.006405694
0.4375000 0.014641586 2.2610681    63
## [436] {butter,
##        other vegetables}        => {yogurt}                0.006405694
0.3197970 0.020030503 2.2924220    63
## [437] {butter,
##        yogurt}                  => {whole milk}            0.009354347
0.6388889 0.014641586 2.5003869    92
## [438] {butter,
##        whole milk}              => {yogurt}                0.009354347
0.3394834 0.027554652 2.4335417    92
## [439] {butter,
##        rolls/buns}              => {other vegetables}      0.005693950
0.4242424 0.013421454 2.1925508    56
## [440] {butter,
##        other vegetables}        => {rolls/buns}            0.005693950
0.2842640 0.020030503 1.5454594    56
## [441] {butter,
##        rolls/buns}              => {whole milk}            0.006609049
0.4924242 0.013421454 1.9271757    65
## [442] {butter,
##        whole milk}              => {rolls/buns}            0.006609049
0.2398524 0.027554652 1.3040068    65
## [443] {butter,
##        other vegetables}        => {whole milk}            0.011489578
0.5736041 0.020030503 2.2448850   113
## [444] {butter,
##        whole milk}              => {other vegetables}      0.011489578
0.4169742 0.027554652 2.1549874   113
## [445] {newspapers,
##        tropical fruit}          => {whole milk}            0.005083884
0.4310345 0.011794611 1.6869177    50
## [446] {newspapers,
```

```
##        root vegetables}          => {other vegetables}      0.005998983
0.5221239 0.011489578 2.6984175    59
## [447] {newspapers,
##        other vegetables}         => {root vegetables}       0.005998983
0.3105263 0.019318760 2.8489051    59
## [448] {newspapers,
##        root vegetables}          => {whole milk}            0.005795628
0.5044248 0.011489578 1.9741415    57
## [449] {newspapers,
##        whole milk}               => {root vegetables}       0.005795628
0.2118959 0.027351296 1.9440264    57
## [450] {newspapers,
##        yogurt}                   => {rolls/buns}            0.005083884
0.3311258 0.015353330 1.8002336    50
## [451] {newspapers,
##        rolls/buns}               => {yogurt}                0.005083884
0.2577320 0.019725470 1.8475174    50
## [452] {newspapers,
##        yogurt}                   => {other vegetables}      0.005592272
0.3642384 0.015353330 1.8824408    55
## [453] {newspapers,
##        other vegetables}         => {yogurt}                0.005592272
0.2894737 0.019318760 2.0750537    55
## [454] {newspapers,
##        yogurt}                   => {whole milk}            0.006609049
0.4304636 0.015353330 1.6846834    65
## [455] {newspapers,
##        whole milk}               => {yogurt}                0.006609049
0.2416357 0.027351296 1.7321334    65
## [456] {newspapers,
##        rolls/buns}               => {other vegetables}      0.005490595
0.2783505 0.019725470 1.4385588    54
## [457] {newspapers,
##        other vegetables}         => {rolls/buns}            0.005490595
0.2842105 0.019318760 1.5451689    54
## [458] {newspapers,
##        rolls/buns}               => {whole milk}            0.007625826
0.3865979 0.019725470 1.5130086    75
## [459] {newspapers,
##        whole milk}               => {rolls/buns}            0.007625826
0.2788104 0.027351296 1.5158100    75
## [460] {newspapers,
##        other vegetables}         => {whole milk}            0.008337570
0.4315789 0.019318760 1.6890485    82
## [461] {newspapers,
##        whole milk}               => {other vegetables}      0.008337570
0.3048327 0.027351296 1.5754229    82
## [462] {domestic eggs,
##        whipped/sour cream}       => {other vegetables}      0.005083884
0.5102041 0.009964413 2.6368141    50
```

```
## [463] {domestic eggs,
##        other vegetables}         => {whipped/sour cream}    0.005083884
0.2283105 0.022267412 3.1850125    50
## [464] {domestic eggs,
##        whipped/sour cream}       => {whole milk}            0.005693950
0.5714286 0.009964413 2.2363709    56
## [465] {domestic eggs,
##        pip fruit}                => {whole milk}            0.005388917
0.6235294 0.008642603 2.4402753    53
## [466] {citrus fruit,
##        domestic eggs}            => {whole milk}            0.005693950
0.5490196 0.010371124 2.1486701    56
## [467] {domestic eggs,
##        tropical fruit}           => {whole milk}            0.006914082
0.6071429 0.011387900 2.3761441    68
## [468] {domestic eggs,
##        whole milk}               => {tropical fruit}        0.006914082
0.2305085 0.029994916 2.1967547    68
## [469] {domestic eggs,
##        root vegetables}          => {other vegetables}      0.007320793
0.5106383 0.014336553 2.6390582    72
## [470] {domestic eggs,
##        other vegetables}         => {root vegetables}       0.007320793
0.3287671 0.022267412 3.0162543    72
## [471] {domestic eggs,
##        root vegetables}          => {whole milk}            0.008540925
0.5957447 0.014336553 2.3315356    84
## [472] {domestic eggs,
##        whole milk}               => {root vegetables}       0.008540925
0.2847458 0.029994916 2.6123830    84
## [473] {domestic eggs,
##        soda}                     => {other vegetables}      0.005083884
0.4098361 0.012404677 2.1180965    50
## [474] {domestic eggs,
##        other vegetables}         => {soda}                  0.005083884
0.2283105 0.022267412 1.3092908    50
## [475] {domestic eggs,
##        soda}                     => {whole milk}            0.005185562
0.4180328 0.012404677 1.6360336    51
## [476] {domestic eggs,
##        yogurt}                   => {other vegetables}      0.005795628
0.4042553 0.014336553 2.0892544    57
## [477] {domestic eggs,
##        other vegetables}         => {yogurt}                0.005795628
0.2602740 0.022267412 1.8657394    57
## [478] {domestic eggs,
##        yogurt}                   => {whole milk}            0.007727504
0.5390071 0.014336553 2.1094846    76
## [479] {domestic eggs,
##        whole milk}               => {yogurt}                0.007727504
```

```
0.2576271 0.029994916 1.8467658    76
## [480] {domestic eggs,
##        rolls/buns}               => {other vegetables}      0.005897306
0.3766234 0.015658363 1.9464482    58
## [481] {domestic eggs,
##        other vegetables}         => {rolls/buns}            0.005897306
0.2648402 0.022267412 1.4398580    58
## [482] {domestic eggs,
##        rolls/buns}               => {whole milk}            0.006609049
0.4220779 0.015658363 1.6518648    65
## [483] {domestic eggs,
##        whole milk}               => {rolls/buns}            0.006609049
0.2203390 0.029994916 1.1979181    65
## [484] {domestic eggs,
##        other vegetables}         => {whole milk}            0.012302999
0.5525114 0.022267412 2.1623358   121
## [485] {domestic eggs,
##        whole milk}               => {other vegetables}      0.012302999
0.4101695 0.029994916 2.1198197   121
## [486] {bottled water,
##        fruit/vegetable juice}    => {soda}                  0.005185562
0.3642857 0.014234875 2.0890671    51
## [487] {fruit/vegetable juice,
##        soda}                     => {bottled water}         0.005185562
0.2817680 0.018403660 2.5493908    51
## [488] {bottled water,
##        fruit/vegetable juice}    => {whole milk}            0.005795628
0.4071429 0.014234875 1.5934142    57
## [489] {fruit/vegetable juice,
##        whole milk}               => {bottled water}         0.005795628
0.2175573 0.026639553 1.9684228    57
## [490] {fruit/vegetable juice,
##        tropical fruit}           => {other vegetables}      0.006609049
0.4814815 0.013726487 2.4883712    65
## [491] {fruit/vegetable juice,
##        other vegetables}         => {tropical fruit}        0.006609049
0.3140097 0.021047280 2.9925242    65
## [492] {fruit/vegetable juice,
##        tropical fruit}           => {whole milk}            0.005998983
0.4370370 0.013726487 1.7104096    59
## [493] {fruit/vegetable juice,
##        whole milk}               => {tropical fruit}        0.005998983
0.2251908 0.026639553 2.1460774    59
## [494] {fruit/vegetable juice,
##        root vegetables}          => {other vegetables}      0.006609049
0.5508475 0.011997966 2.8468653    65
## [495] {fruit/vegetable juice,
##        other vegetables}         => {root vegetables}       0.006609049
0.3140097 0.021047280 2.8808629    65
## [496] {fruit/vegetable juice,
```

```
##        root vegetables}          => {whole milk}           0.006507372
0.5423729 0.011997966 2.1226571    64
## [497] {fruit/vegetable juice,
##        whole milk}               => {root vegetables}      0.006507372
0.2442748 0.026639553 2.2410847    64
## [498] {fruit/vegetable juice,
##        soda}                     => {yogurt}               0.005083884
0.2762431 0.018403660 1.9802120    50
## [499] {fruit/vegetable juice,
##        yogurt}                   => {soda}                 0.005083884
0.2717391 0.018708693 1.5583407    50
## [500] {fruit/vegetable juice,
##        soda}                     => {whole milk}           0.006100661
0.3314917 0.018403660 1.2973422    60
## [501] {fruit/vegetable juice,
##        whole milk}               => {soda}                 0.006100661
0.2290076 0.026639553 1.3132887    60
## [502] {fruit/vegetable juice,
##        yogurt}                   => {other vegetables}     0.008235892
0.4402174 0.018708693 2.2751120    81
## [503] {fruit/vegetable juice,
##        other vegetables}         => {yogurt}               0.008235892
0.3913043 0.021047280 2.8050133    81
## [504] {fruit/vegetable juice,
##        yogurt}                   => {whole milk}           0.009456024
0.5054348 0.018708693 1.9780943    93
## [505] {fruit/vegetable juice,
##        whole milk}               => {yogurt}               0.009456024
0.3549618 0.026639553 2.5444968    93
## [506] {fruit/vegetable juice,
##        rolls/buns}               => {whole milk}           0.005592272
0.3846154 0.014539908 1.5052496    55
## [507] {fruit/vegetable juice,
##        whole milk}               => {rolls/buns}           0.005592272
0.2099237 0.026639553 1.1412931    55
## [508] {fruit/vegetable juice,
##        other vegetables}         => {whole milk}           0.010472801
0.4975845 0.021047280 1.9473713   103
## [509] {fruit/vegetable juice,
##        whole milk}               => {other vegetables}     0.010472801
0.3931298 0.026639553 2.0317558   103
## [510] {pip fruit,
##        whipped/sour cream}       => {other vegetables}     0.005592272
0.6043956 0.009252669 3.1236105    55
## [511] {other vegetables,
##        pip fruit}                => {whipped/sour cream}   0.005592272
0.2140078 0.026131164 2.9854844    55
## [512] {pip fruit,
##        whipped/sour cream}       => {whole milk}           0.005998983
0.6483516 0.009252669 2.5374208    59
```

```
## [513] {citrus fruit,
##        whipped/sour cream}       => {other vegetables}      0.005693950
0.5233645 0.010879512 2.7048291    56
## [514] {citrus fruit,
##        whipped/sour cream}       => {whole milk}            0.006304016
0.5794393 0.010879512 2.2677219    62
## [515] {citrus fruit,
##        whole milk}               => {whipped/sour cream}    0.006304016
0.2066667 0.030503305 2.8830733    62
## [516] {sausage,
##        whipped/sour cream}       => {whole milk}            0.005083884
0.5617978 0.009049314 2.1986792    50
## [517] {tropical fruit,
##        whipped/sour cream}       => {yogurt}                0.006202339
0.4485294 0.013828165 3.2152236    61
## [518] {whipped/sour cream,
##        yogurt}                   => {tropical fruit}        0.006202339
0.2990196 0.020742247 2.8496685    61
## [519] {tropical fruit,
##        yogurt}                   => {whipped/sour cream}    0.006202339
0.2118056 0.029283172 2.9547626    61
## [520] {tropical fruit,
##        whipped/sour cream}       => {other vegetables}      0.007829181
0.5661765 0.013828165 2.9260881    77
## [521] {other vegetables,
##        whipped/sour cream}       => {tropical fruit}        0.007829181
0.2711268 0.028876462 2.5838485    77
## [522] {other vegetables,
##        tropical fruit}           => {whipped/sour cream}    0.007829181
0.2181303 0.035892222 3.0429952    77
## [523] {tropical fruit,
##        whipped/sour cream}       => {whole milk}            0.007930859
0.5735294 0.013828165 2.2445928    78
## [524] {whipped/sour cream,
##        whole milk}               => {tropical fruit}        0.007930859
0.2460568 0.032231825 2.3449307    78
## [525] {root vegetables,
##        whipped/sour cream}       => {yogurt}                0.006405694
0.3750000 0.017081851 2.6881378    63
## [526] {whipped/sour cream,
##        yogurt}                   => {root vegetables}       0.006405694
0.3088235 0.020742247 2.8332830    63
## [527] {root vegetables,
##        yogurt}                   => {whipped/sour cream}    0.006405694
0.2480315 0.025826131 3.4601273    63
## [528] {root vegetables,
##        whipped/sour cream}       => {other vegetables}      0.008540925
0.5000000 0.017081851 2.5840778    84
## [529] {other vegetables,
##        whipped/sour cream}       => {root vegetables}       0.008540925
```

```
0.2957746 0.028876462 2.7135668    84
## [530] {root vegetables,
##        whipped/sour cream}       => {whole milk}            0.009456024
0.5535714 0.017081851 2.1664843    93
## [531] {whipped/sour cream,
##        whole milk}               => {root vegetables}       0.009456024
0.2933754 0.032231825 2.6915550    93
## [532] {soda,
##        whipped/sour cream}       => {whole milk}            0.005490595
0.4736842 0.011591256 1.8538337    54
## [533] {whipped/sour cream,
##        yogurt}                   => {other vegetables}      0.010167768
0.4901961 0.020742247 2.5334096   100
## [534] {other vegetables,
##        whipped/sour cream}       => {yogurt}                0.010167768
0.3521127 0.028876462 2.5240730   100
## [535] {other vegetables,
##        yogurt}                   => {whipped/sour cream}    0.010167768
0.2341920 0.043416370 3.2670620   100
## [536] {whipped/sour cream,
##        yogurt}                   => {whole milk}            0.010879512
0.5245098 0.020742247 2.0527473   107
## [537] {whipped/sour cream,
##        whole milk}               => {yogurt}                0.010879512
0.3375394 0.032231825 2.4196066   107
## [538] {rolls/buns,
##        whipped/sour cream}       => {other vegetables}      0.006710727
0.4583333 0.014641586 2.3687380    66
## [539] {other vegetables,
##        whipped/sour cream}       => {rolls/buns}            0.006710727
0.2323944 0.028876462 1.2634597    66
## [540] {rolls/buns,
##        whipped/sour cream}       => {whole milk}            0.007829181
0.5347222 0.014641586 2.0927151    77
## [541] {whipped/sour cream,
##        whole milk}               => {rolls/buns}            0.007829181
0.2429022 0.032231825 1.3205877    77
## [542] {other vegetables,
##        whipped/sour cream}       => {whole milk}            0.014641586
0.5070423 0.028876462 1.9843854   144
## [543] {whipped/sour cream,
##        whole milk}               => {other vegetables}      0.014641586
0.4542587 0.032231825 2.3476795   144
## [544] {pastry,
##        pip fruit}                => {whole milk}            0.005083884
0.4761905 0.010676157 1.8636424    50
## [545] {citrus fruit,
##        pip fruit}                => {tropical fruit}        0.005592272
0.4044118 0.013828165 3.8540598    55
## [546] {pip fruit,
```

```
##        tropical fruit}           => {citrus fruit}          0.005592272
0.2736318 0.020437214 3.3061046    55
## [547] {citrus fruit,
##        tropical fruit}           => {pip fruit}             0.005592272
0.2806122 0.019928826 3.7094374    55
## [548] {citrus fruit,
##        pip fruit}                => {other vegetables}      0.005897306
0.4264706 0.013828165 2.2040663    58
## [549] {other vegetables,
##        pip fruit}                => {citrus fruit}          0.005897306
0.2256809 0.026131164 2.7267469    58
## [550] {citrus fruit,
##        other vegetables}         => {pip fruit}             0.005897306
0.2042254 0.028876462 2.6996725    58
## [551] {citrus fruit,
##        pip fruit}                => {whole milk}            0.005185562
0.3750000 0.013828165 1.4676184    51
## [552] {pip fruit,
##        sausage}                  => {whole milk}            0.005592272
0.5188679 0.010777834 2.0306669    55
## [553] {pip fruit,
##        tropical fruit}           => {root vegetables}       0.005287239
0.2587065 0.020437214 2.3734870    52
## [554] {pip fruit,
##        root vegetables}          => {tropical fruit}        0.005287239
0.3398693 0.015556685 3.2389674    52
## [555] {root vegetables,
##        tropical fruit}           => {pip fruit}             0.005287239
0.2512077 0.021047280 3.3207366    52
## [556] {pip fruit,
##        tropical fruit}           => {yogurt}                0.006405694
0.3134328 0.020437214 2.2468017    63
## [557] {pip fruit,
##        yogurt}                   => {tropical fruit}        0.006405694
0.3559322 0.017996950 3.3920477    63
## [558] {tropical fruit,
##        yogurt}                   => {pip fruit}             0.006405694
0.2187500 0.029283172 2.8916751    63
## [559] {pip fruit,
##        tropical fruit}           => {other vegetables}      0.009456024
0.4626866 0.020437214 2.3912361    93
## [560] {other vegetables,
##        pip fruit}                => {tropical fruit}        0.009456024
0.3618677 0.026131164 3.4486132    93
## [561] {other vegetables,
##        tropical fruit}           => {pip fruit}             0.009456024
0.2634561 0.035892222 3.4826487    93
## [562] {pip fruit,
##        tropical fruit}           => {whole milk}            0.008439248
0.4129353 0.020437214 1.6160839    83
```

```
## [563] {pip fruit,
##        whole milk}              => {tropical fruit}        0.008439248
0.2804054 0.030096594 2.6722744    83
## [564] {pip fruit,
##        root vegetables}         => {yogurt}                0.005287239
0.3398693 0.015556685 2.4363079    52
## [565] {pip fruit,
##        yogurt}                  => {root vegetables}       0.005287239
0.2937853 0.017996950 2.6953158    52
## [566] {root vegetables,
##        yogurt}                  => {pip fruit}             0.005287239
0.2047244 0.025826131 2.7062696    52
## [567] {pip fruit,
##        root vegetables}         => {other vegetables}      0.008134215
0.5228758 0.015556685 2.7023036    80
## [568] {other vegetables,
##        pip fruit}               => {root vegetables}       0.008134215
0.3112840 0.026131164 2.8558569    80
## [569] {pip fruit,
##        root vegetables}         => {whole milk}            0.008947636
0.5751634 0.015556685 2.2509877    88
## [570] {pip fruit,
##        whole milk}              => {root vegetables}       0.008947636
0.2972973 0.030096594 2.7275363    88
## [571] {pip fruit,
##        yogurt}                  => {other vegetables}      0.008134215
0.4519774 0.017996950 2.3358895    80
## [572] {other vegetables,
##        pip fruit}               => {yogurt}                0.008134215
0.3112840 0.026131164 2.2313984    80
## [573] {pip fruit,
##        yogurt}                  => {whole milk}            0.009557702
0.5310734 0.017996950 2.0784351    94
## [574] {pip fruit,
##        whole milk}              => {yogurt}                0.009557702
0.3175676 0.030096594 2.2764410    94
## [575] {pip fruit,
##        rolls/buns}              => {other vegetables}      0.005083884
0.3649635 0.013929842 1.8861882    50
## [576] {pip fruit,
##        rolls/buns}              => {whole milk}            0.006202339
0.4452555 0.013929842 1.7425737    61
## [577] {pip fruit,
##        whole milk}              => {rolls/buns}            0.006202339
0.2060811 0.030096594 1.1204021    61
## [578] {other vegetables,
##        pip fruit}               => {whole milk}            0.013523132
0.5175097 0.026131164 2.0253514   133
## [579] {pip fruit,
##        whole milk}              => {other vegetables}      0.013523132
```

```
0.4493243 0.030096594 2.3221780   133
## [580] {pastry,
##        sausage}                 => {whole milk}           0.005693950
0.4552846 0.012506355 1.7818239    56
## [581] {pastry,
##        tropical fruit}          => {other vegetables}     0.005083884
0.3846154 0.013218099 1.9877521    50
## [582] {other vegetables,
##        pastry}                  => {tropical fruit}       0.005083884
0.2252252 0.022572445 2.1464051    50
## [583] {pastry,
##        tropical fruit}          => {whole milk}           0.006710727
0.5076923 0.013218099 1.9869295    66
## [584] {pastry,
##        whole milk}              => {tropical fruit}       0.006710727
0.2018349 0.033248602 1.9234941    66
## [585] {pastry,
##        root vegetables}         => {other vegetables}     0.005897306
0.5370370 0.010981190 2.7754909    58
## [586] {other vegetables,
##        pastry}                  => {root vegetables}      0.005897306
0.2612613 0.022572445 2.3969258    58
## [587] {pastry,
##        root vegetables}         => {whole milk}           0.005693950
0.5185185 0.010981190 2.0292995    56
## [588] {pastry,
##        soda}                    => {rolls/buns}           0.005388917
0.2560386 0.021047280 1.3920067    53
## [589] {pastry,
##        rolls/buns}              => {soda}                 0.005388917
0.2572816 0.020945602 1.4754309    53
## [590] {pastry,
##        soda}                    => {other vegetables}     0.005490595
0.2608696 0.021047280 1.3482145    54
## [591] {other vegetables,
##        pastry}                  => {soda}                 0.005490595
0.2432432 0.022572445 1.3949255    54
## [592] {pastry,
##        soda}                    => {whole milk}           0.008235892
0.3913043 0.021047280 1.5314279    81
## [593] {pastry,
##        whole milk}              => {soda}                 0.008235892
0.2477064 0.033248602 1.4205205    81
## [594] {soda,
##        whole milk}              => {pastry}               0.008235892
0.2055838 0.040061007 2.3107614    81
## [595] {pastry,
##        yogurt}                  => {rolls/buns}           0.005795628
0.3275862 0.017691917 1.7809897    57
## [596] {pastry,
```

```
##        rolls/buns}                 => {yogurt}                0.005795628
0.2766990 0.020945602 1.9834803    57
## [597] {pastry,
##        yogurt}                     => {other vegetables}      0.006609049
0.3735632 0.017691917 1.9306328    65
## [598] {other vegetables,
##        pastry}                     => {yogurt}                0.006609049
0.2927928 0.022572445 2.0988463    65
## [599] {pastry,
##        yogurt}                     => {whole milk}            0.009150991
0.5172414 0.017691917 2.0243012    90
## [600] {pastry,
##        whole milk}                 => {yogurt}                0.009150991
0.2752294 0.033248602 1.9729451    90
## [601] {pastry,
##        rolls/buns}                 => {other vegetables}      0.006100661
0.2912621 0.020945602 1.5052880    60
## [602] {other vegetables,
##        pastry}                     => {rolls/buns}            0.006100661
0.2702703 0.022572445 1.4693798    60
## [603] {pastry,
##        rolls/buns}                 => {whole milk}            0.008540925
0.4077670 0.020945602 1.5958569    84
## [604] {pastry,
##        whole milk}                 => {rolls/buns}            0.008540925
0.2568807 0.033248602 1.3965849    84
## [605] {other vegetables,
##        pastry}                     => {whole milk}            0.010574479
0.4684685 0.022572445 1.8334212   104
## [606] {pastry,
##        whole milk}                 => {other vegetables}      0.010574479
0.3180428 0.033248602 1.6436947   104
## [607] {bottled water,
##        citrus fruit}               => {other vegetables}      0.005083884
0.3759398 0.013523132 1.9429156    50
## [608] {bottled water,
##        other vegetables}           => {citrus fruit}          0.005083884
0.2049180 0.024809354 2.4758831    50
## [609] {bottled water,
##        citrus fruit}               => {whole milk}            0.005897306
0.4360902 0.013523132 1.7067041    58
## [610] {citrus fruit,
##        tropical fruit}             => {root vegetables}       0.005693950
0.2857143 0.019928826 2.6212687    56
## [611] {citrus fruit,
##        root vegetables}            => {tropical fruit}        0.005693950
0.3218391 0.017691917 3.0671389    56
## [612] {root vegetables,
##        tropical fruit}             => {citrus fruit}          0.005693950
0.2705314 0.021047280 3.2686441    56
```

```
## [613] {citrus fruit,
##        tropical fruit}          => {yogurt}               0.006304016
0.3163265 0.019928826 2.2675448    62
## [614] {citrus fruit,
##        yogurt}                  => {tropical fruit}       0.006304016
0.2910798 0.021657346 2.7740019    62
## [615] {tropical fruit,
##        yogurt}                  => {citrus fruit}         0.006304016
0.2152778 0.029283172 2.6010528    62
## [616] {citrus fruit,
##        tropical fruit}          => {other vegetables}     0.009049314
0.4540816 0.019928826 2.3467645    89
## [617] {citrus fruit,
##        other vegetables}        => {tropical fruit}       0.009049314
0.3133803 0.028876462 2.9865262    89
## [618] {other vegetables,
##        tropical fruit}          => {citrus fruit}         0.009049314
0.2521246 0.035892222 3.0462480    89
## [619] {citrus fruit,
##        tropical fruit}          => {whole milk}           0.009049314
0.4540816 0.019928826 1.7771161    89
## [620] {citrus fruit,
##        whole milk}              => {tropical fruit}       0.009049314
0.2966667 0.030503305 2.8272448    89
## [621] {tropical fruit,
##        whole milk}              => {citrus fruit}         0.009049314
0.2139423 0.042297916 2.5849172    89
## [622] {citrus fruit,
##        root vegetables}         => {other vegetables}     0.010371124
0.5862069 0.017691917 3.0296084   102
## [623] {citrus fruit,
##        other vegetables}        => {root vegetables}      0.010371124
0.3591549 0.028876462 3.2950455   102
## [624] {other vegetables,
##        root vegetables}         => {citrus fruit}         0.010371124
0.2188841 0.047381800 2.6446257   102
## [625] {citrus fruit,
##        root vegetables}         => {whole milk}           0.009150991
0.5172414 0.017691917 2.0243012    90
## [626] {citrus fruit,
##        whole milk}              => {root vegetables}      0.009150991
0.3000000 0.030503305 2.7523321    90
## [627] {citrus fruit,
##        yogurt}                  => {rolls/buns}           0.005795628
0.2676056 0.021657346 1.4548930    57
## [628] {citrus fruit,
##        rolls/buns}              => {yogurt}               0.005795628
0.3454545 0.016776817 2.4763451    57
## [629] {citrus fruit,
##        yogurt}                  => {other vegetables}     0.007625826
```

```
0.3521127 0.021657346 1.8197731    75
## [630] {citrus fruit,
##        other vegetables}         => {yogurt}               0.007625826
0.2640845 0.028876462 1.8930548    75
## [631] {citrus fruit,
##        yogurt}                   => {whole milk}           0.010269446
0.4741784 0.021657346 1.8557678   101
## [632] {citrus fruit,
##        whole milk}               => {yogurt}               0.010269446
0.3366667 0.030503305 2.4133503   101
## [633] {citrus fruit,
##        rolls/buns}               => {other vegetables}     0.005998983
0.3575758 0.016776817 1.8480071    59
## [634] {citrus fruit,
##        other vegetables}         => {rolls/buns}           0.005998983
0.2077465 0.028876462 1.1294564    59
## [635] {citrus fruit,
##        rolls/buns}               => {whole milk}           0.007219115
0.4303030 0.016776817 1.6840550    71
## [636] {citrus fruit,
##        whole milk}               => {rolls/buns}           0.007219115
0.2366667 0.030503305 1.2866869    71
## [637] {citrus fruit,
##        other vegetables}         => {whole milk}           0.013014743
0.4507042 0.028876462 1.7638982   128
## [638] {citrus fruit,
##        whole milk}               => {other vegetables}     0.013014743
0.4266667 0.030503305 2.2050797   128
## [639] {sausage,
##        shopping bags}            => {soda}                 0.005693950
0.3636364 0.015658363 2.0853432    56
## [640] {shopping bags,
##        soda}                     => {sausage}              0.005693950
0.2314050 0.024605999 2.4630604    56
## [641] {sausage,
##        soda}                     => {shopping bags}        0.005693950
0.2343096 0.024300966 2.3781580    56
## [642] {sausage,
##        shopping bags}            => {rolls/buns}           0.005998983
0.3831169 0.015658363 2.0828936    59
## [643] {rolls/buns,
##        shopping bags}            => {sausage}              0.005998983
0.3072917 0.019522115 3.2707939    59
## [644] {sausage,
##        shopping bags}            => {other vegetables}     0.005388917
0.3441558 0.015658363 1.7786509    53
## [645] {other vegetables,
##        shopping bags}            => {sausage}              0.005388917
0.2324561 0.023182511 2.4742491    53
## [646] {root vegetables,
```

```
##        shopping bags}            => {other vegetables}      0.006609049
0.5158730 0.012811388 2.6661120    65
## [647] {other vegetables,
##        shopping bags}            => {root vegetables}       0.006609049
0.2850877 0.023182511 2.6155203    65
## [648] {root vegetables,
##        shopping bags}            => {whole milk}            0.005287239
0.4126984 0.012811388 1.6151567    52
## [649] {shopping bags,
##        whole milk}               => {root vegetables}       0.005287239
0.2157676 0.024504321 1.9795473    52
## [650] {shopping bags,
##        soda}                     => {rolls/buns}            0.006304016
0.2561983 0.024605999 1.3928749    62
## [651] {rolls/buns,
##        shopping bags}            => {soda}                  0.006304016
0.3229167 0.019522115 1.8518282    62
## [652] {shopping bags,
##        soda}                     => {other vegetables}      0.005388917
0.2190083 0.024605999 1.1318688    53
## [653] {other vegetables,
##        shopping bags}            => {soda}                  0.005388917
0.2324561 0.023182511 1.3330648    53
## [654] {shopping bags,
##        soda}                     => {whole milk}            0.006812405
0.2768595 0.024605999 1.0835309    67
## [655] {shopping bags,
##        whole milk}               => {soda}                  0.006812405
0.2780083 0.024504321 1.5942925    67
## [656] {shopping bags,
##        yogurt}                   => {other vegetables}      0.005388917
0.3533333 0.015251652 1.8260816    53
## [657] {other vegetables,
##        shopping bags}            => {yogurt}                0.005388917
0.2324561 0.023182511 1.6663310    53
## [658] {shopping bags,
##        yogurt}                   => {whole milk}            0.005287239
0.3466667 0.015251652 1.3567317    52
## [659] {shopping bags,
##        whole milk}               => {yogurt}                0.005287239
0.2157676 0.024504321 1.5467017    52
## [660] {rolls/buns,
##        shopping bags}            => {other vegetables}      0.005287239
0.2708333 0.019522115 1.3997088    52
## [661] {other vegetables,
##        shopping bags}            => {rolls/buns}            0.005287239
0.2280702 0.023182511 1.2399503    52
## [662] {rolls/buns,
##        shopping bags}            => {whole milk}            0.005287239
0.2708333 0.019522115 1.0599466    52
```

```
## [663] {shopping bags,
##        whole milk}              => {rolls/buns}             0.005287239
0.2157676 0.024504321 1.1730651    52
## [664] {other vegetables,
##        shopping bags}           => {whole milk}             0.007625826
0.3289474 0.023182511 1.2873845    75
## [665] {shopping bags,
##        whole milk}              => {other vegetables}       0.007625826
0.3112033 0.024504321 1.6083472    75
## [666] {bottled water,
##        sausage}                 => {other vegetables}       0.005083884
0.4237288 0.011997966 2.1898964    50
## [667] {bottled water,
##        other vegetables}        => {sausage}                0.005083884
0.2049180 0.024809354 2.1811351    50
## [668] {sausage,
##        tropical fruit}          => {other vegetables}       0.005998983
0.4306569 0.013929842 2.2257020    59
## [669] {other vegetables,
##        sausage}                 => {tropical fruit}         0.005998983
0.2226415 0.026944586 2.1217822    59
## [670] {sausage,
##        tropical fruit}          => {whole milk}             0.007219115
0.5182482 0.013929842 2.0282415    71
## [671] {sausage,
##        whole milk}              => {tropical fruit}         0.007219115
0.2414966 0.029893238 2.3014719    71
## [672] {root vegetables,
##        sausage}                 => {yogurt}                 0.005185562
0.3469388 0.014946619 2.4869846    51
## [673] {sausage,
##        yogurt}                  => {root vegetables}        0.005185562
0.2642487 0.019623793 2.4243340    51
## [674] {root vegetables,
##        yogurt}                  => {sausage}                0.005185562
0.2007874 0.025826131 2.1371689    51
## [675] {root vegetables,
##        sausage}                 => {other vegetables}       0.006812405
0.4557823 0.014946619 2.3555539    67
## [676] {other vegetables,
##        sausage}                 => {root vegetables}        0.006812405
0.2528302 0.026944586 2.3195755    67
## [677] {root vegetables,
##        sausage}                 => {whole milk}             0.007727504
0.5170068 0.014946619 2.0233832    76
## [678] {sausage,
##        whole milk}              => {root vegetables}        0.007727504
0.2585034 0.029893238 2.3716240    76
## [679] {sausage,
##        soda}                    => {yogurt}                 0.005592272
```

```
0.2301255 0.024300966 1.6496243    55
## [680] {sausage,
##        yogurt}                   => {soda}                  0.005592272
0.2849741 0.019623793 1.6342392    55
## [681] {soda,
##        yogurt}                   => {sausage}               0.005592272
0.2044610 0.027351296 2.1762701    55
## [682] {sausage,
##        soda}                     => {rolls/buns}            0.009659380
0.3974895 0.024300966 2.1610335    95
## [683] {rolls/buns,
##        sausage}                  => {soda}                  0.009659380
0.3156146 0.030604982 1.8099532    95
## [684] {rolls/buns,
##        soda}                     => {sausage}               0.009659380
0.2519894 0.038332486 2.6821598    95
## [685] {sausage,
##        soda}                     => {other vegetables}      0.007219115
0.2970711 0.024300966 1.5353098    71
## [686] {other vegetables,
##        sausage}                  => {soda}                  0.007219115
0.2679245 0.026944586 1.5364652    71
## [687] {other vegetables,
##        soda}                     => {sausage}               0.007219115
0.2204969 0.032740214 2.3469556    71
## [688] {sausage,
##        soda}                     => {whole milk}            0.006710727
0.2761506 0.024300966 1.0807566    66
## [689] {sausage,
##        whole milk}               => {soda}                  0.006710727
0.2244898 0.029893238 1.2873803    66
## [690] {sausage,
##        yogurt}                   => {rolls/buns}            0.005998983
0.3056995 0.019623793 1.6619980    59
## [691] {sausage,
##        yogurt}                   => {other vegetables}      0.008134215
0.4145078 0.019623793 2.1422406    80
## [692] {other vegetables,
##        sausage}                  => {yogurt}                0.008134215
0.3018868 0.026944586 2.1640354    80
## [693] {sausage,
##        yogurt}                   => {whole milk}            0.008744281
0.4455959 0.019623793 1.7439058    86
## [694] {sausage,
##        whole milk}               => {yogurt}                0.008744281
0.2925170 0.029893238 2.0968694    86
## [695] {rolls/buns,
##        sausage}                  => {other vegetables}      0.008845958
0.2890365 0.030604982 1.4937858    87
## [696] {other vegetables,
```

```
##        sausage}                    => {rolls/buns}           0.008845958
0.3283019 0.026944586 1.7848806    87
## [697] {other vegetables,
##        rolls/buns}                 => {sausage}              0.008845958
0.2076372 0.042602949 2.2100781    87
## [698] {rolls/buns,
##        sausage}                    => {whole milk}           0.009354347
0.3056478 0.030604982 1.1961984    92
## [699] {sausage,
##        whole milk}                 => {rolls/buns}           0.009354347
0.3129252 0.029893238 1.7012820    92
## [700] {other vegetables,
##        sausage}                    => {whole milk}           0.010167768
0.3773585 0.026944586 1.4768487   100
## [701] {sausage,
##        whole milk}                 => {other vegetables}     0.010167768
0.3401361 0.029893238 1.7578760   100
## [702] {bottled water,
##        tropical fruit}             => {soda}                 0.005185562
0.2802198 0.018505338 1.6069747    51
## [703] {soda,
##        tropical fruit}             => {bottled water}        0.005185562
0.2487805 0.020843925 2.2509256    51
## [704] {bottled water,
##        tropical fruit}             => {yogurt}               0.007117438
0.3846154 0.018505338 2.7570644    70
## [705] {bottled water,
##        yogurt}                     => {tropical fruit}       0.007117438
0.3097345 0.022979156 2.9517819    70
## [706] {tropical fruit,
##        yogurt}                     => {bottled water}        0.007117438
0.2430556 0.029283172 2.1991273    70
## [707] {bottled water,
##        tropical fruit}             => {rolls/buns}           0.005388917
0.2912088 0.018505338 1.5832164    53
## [708] {bottled water,
##        rolls/buns}                 => {tropical fruit}       0.005388917
0.2226891 0.024199288 2.1222355    53
## [709] {rolls/buns,
##        tropical fruit}             => {bottled water}        0.005388917
0.2190083 0.024605999 1.9815513    53
## [710] {bottled water,
##        tropical fruit}             => {other vegetables}     0.006202339
0.3351648 0.018505338 1.7321840    61
## [711] {bottled water,
##        other vegetables}           => {tropical fruit}       0.006202339
0.2500000 0.024809354 2.3825097    61
## [712] {bottled water,
##        tropical fruit}             => {whole milk}           0.008032537
0.4340659 0.018505338 1.6987817    79
```

```
## [713] {bottled water,
##        whole milk}                => {tropical fruit}        0.008032537
0.2337278 0.034367056 2.2274351    79
## [714] {bottled water,
##        root vegetables}           => {other vegetables}      0.007015760
0.4480519 0.015658363 2.3156022    69
## [715] {bottled water,
##        other vegetables}          => {root vegetables}       0.007015760
0.2827869 0.024809354 2.5944114    69
## [716] {bottled water,
##        root vegetables}           => {whole milk}            0.007320793
0.4675325 0.015658363 1.8297580    72
## [717] {bottled water,
##        whole milk}                => {root vegetables}       0.007320793
0.2130178 0.034367056 1.9543186    72
## [718] {bottled water,
##        soda}                      => {yogurt}                0.007422471
0.2561404 0.028978139 1.8361081    73
## [719] {bottled water,
##        yogurt}                    => {soda}                  0.007422471
0.3230088 0.022979156 1.8523569    73
## [720] {soda,
##        yogurt}                    => {bottled water}         0.007422471
0.2713755 0.027351296 2.4553613    73
## [721] {bottled water,
##        soda}                      => {rolls/buns}            0.006812405
0.2350877 0.028978139 1.2781027    67
## [722] {bottled water,
##        rolls/buns}                => {soda}                  0.006812405
0.2815126 0.024199288 1.6143886    67
## [723] {bottled water,
##        other vegetables}          => {soda}                  0.005693950
0.2295082 0.024809354 1.3161593    56
## [724] {bottled water,
##        soda}                      => {whole milk}            0.007524148
0.2596491 0.028978139 1.0161755    74
## [725] {bottled water,
##        whole milk}                => {soda}                  0.007524148
0.2189349 0.034367056 1.2555247    74
## [726] {bottled water,
##        yogurt}                    => {rolls/buns}            0.007117438
0.3097345 0.022979156 1.6839353    70
## [727] {bottled water,
##        rolls/buns}                => {yogurt}                0.007117438
0.2941176 0.024199288 2.1083433    70
## [728] {rolls/buns,
##        yogurt}                    => {bottled water}         0.007117438
0.2071006 0.034367056 1.8738126    70
## [729] {bottled water,
##        yogurt}                    => {other vegetables}      0.008134215
```

```
0.3539823 0.022979156 1.8294356    80
## [730] {bottled water,
##        other vegetables}         => {yogurt}                0.008134215
0.3278689 0.024809354 2.3502844    80
## [731] {bottled water,
##        yogurt}                   => {whole milk}            0.009659380
0.4203540 0.022979156 1.6451180    95
## [732] {bottled water,
##        whole milk}               => {yogurt}                0.009659380
0.2810651 0.034367056 2.0147778    95
## [733] {bottled water,
##        rolls/buns}               => {other vegetables}      0.007320793
0.3025210 0.024199288 1.5634756    72
## [734] {bottled water,
##        other vegetables}         => {rolls/buns}            0.007320793
0.2950820 0.024809354 1.6042737    72
## [735] {bottled water,
##        rolls/buns}               => {whole milk}            0.008744281
0.3613445 0.024199288 1.4141757    86
## [736] {bottled water,
##        whole milk}               => {rolls/buns}            0.008744281
0.2544379 0.034367056 1.3833037    86
## [737] {bottled water,
##        other vegetables}         => {whole milk}            0.010777834
0.4344262 0.024809354 1.7001918   106
## [738] {bottled water,
##        whole milk}               => {other vegetables}      0.010777834
0.3136095 0.034367056 1.6207825   106
## [739] {root vegetables,
##        tropical fruit}           => {yogurt}                0.008134215
0.3864734 0.021047280 2.7703835    80
## [740] {tropical fruit,
##        yogurt}                   => {root vegetables}       0.008134215
0.2777778 0.029283172 2.5484556    80
## [741] {root vegetables,
##        yogurt}                   => {tropical fruit}        0.008134215
0.3149606 0.025826131 3.0015870    80
## [742] {root vegetables,
##        tropical fruit}           => {rolls/buns}            0.005897306
0.2801932 0.021047280 1.5233281    58
## [743] {rolls/buns,
##        tropical fruit}           => {root vegetables}       0.005897306
0.2396694 0.024605999 2.1988328    58
## [744] {rolls/buns,
##        root vegetables}          => {tropical fruit}        0.005897306
0.2426778 0.024300966 2.3127291    58
## [745] {root vegetables,
##        tropical fruit}           => {other vegetables}      0.012302999
0.5845411 0.021047280 3.0209991   121
## [746] {other vegetables,
```

```
##        tropical fruit}            => {root vegetables}       0.012302999
0.3427762 0.035892222 3.1447798   121
## [747] {other vegetables,
##        root vegetables}           => {tropical fruit}        0.012302999
0.2596567 0.047381800 2.4745380   121
## [748] {root vegetables,
##        tropical fruit}            => {whole milk}            0.011997966
0.5700483 0.021047280 2.2309690   118
## [749] {tropical fruit,
##        whole milk}                => {root vegetables}       0.011997966
0.2836538 0.042297916 2.6023653   118
## [750] {root vegetables,
##        whole milk}                => {tropical fruit}        0.011997966
0.2453222 0.048906965 2.3379305   118
## [751] {soda,
##        tropical fruit}            => {yogurt}                0.006609049
0.3170732 0.020843925 2.2728970    65
## [752] {tropical fruit,
##        yogurt}                    => {soda}                  0.006609049
0.2256944 0.029283172 1.2942885    65
## [753] {soda,
##        yogurt}                    => {tropical fruit}        0.006609049
0.2416357 0.027351296 2.3027975    65
## [754] {soda,
##        tropical fruit}            => {rolls/buns}            0.005388917
0.2585366 0.020843925 1.4055872    53
## [755] {rolls/buns,
##        tropical fruit}            => {soda}                  0.005388917
0.2190083 0.024605999 1.2559454    53
## [756] {soda,
##        tropical fruit}            => {other vegetables}      0.007219115
0.3463415 0.020843925 1.7899466    71
## [757] {other vegetables,
##        tropical fruit}            => {soda}                  0.007219115
0.2011331 0.035892222 1.1534370    71
## [758] {other vegetables,
##        soda}                      => {tropical fruit}        0.007219115
0.2204969 0.032740214 2.1013440    71
## [759] {soda,
##        tropical fruit}            => {whole milk}            0.007829181
0.3756098 0.020843925 1.4700048    77
## [760] {tropical fruit,
##        yogurt}                    => {rolls/buns}            0.008744281
0.2986111 0.029283172 1.6234606    86
## [761] {rolls/buns,
##        tropical fruit}            => {yogurt}                0.008744281
0.3553719 0.024605999 2.5474363    86
## [762] {rolls/buns,
##        yogurt}                    => {tropical fruit}        0.008744281
0.2544379 0.034367056 2.4248028    86
```

```
## [763] {tropical fruit,
##        yogurt}                    => {other vegetables}      0.012302999
0.4201389 0.029283172 2.1713431   121
## [764] {other vegetables,
##        tropical fruit}            => {yogurt}                0.012302999
0.3427762 0.035892222 2.4571457   121
## [765] {other vegetables,
##        yogurt}                    => {tropical fruit}        0.012302999
0.2833724 0.043416370 2.7005496   121
## [766] {tropical fruit,
##        yogurt}                    => {whole milk}            0.015149975
0.5173611 0.029283172 2.0247698   149
## [767] {tropical fruit,
##        whole milk}                => {yogurt}                0.015149975
0.3581731 0.042297916 2.5675162   149
## [768] {whole milk,
##        yogurt}                    => {tropical fruit}        0.015149975
0.2704174 0.056024403 2.5770885   149
## [769] {rolls/buns,
##        tropical fruit}            => {other vegetables}      0.007829181
0.3181818 0.024605999 1.6444131    77
## [770] {other vegetables,
##        tropical fruit}            => {rolls/buns}            0.007829181
0.2181303 0.035892222 1.1859102    77
## [771] {rolls/buns,
##        tropical fruit}            => {whole milk}            0.010981190
0.4462810 0.024605999 1.7465872   108
## [772] {tropical fruit,
##        whole milk}                => {rolls/buns}            0.010981190
0.2596154 0.042297916 1.4114524   108
## [773] {other vegetables,
##        tropical fruit}            => {whole milk}            0.017081851
0.4759207 0.035892222 1.8625865   168
## [774] {tropical fruit,
##        whole milk}                => {other vegetables}      0.017081851
0.4038462 0.042297916 2.0871397   168
## [775] {other vegetables,
##        whole milk}                => {tropical fruit}        0.017081851
0.2282609 0.074834774 2.1753349   168
## [776] {root vegetables,
##        soda}                      => {other vegetables}      0.008235892
0.4426230 0.018607016 2.2875443    81
## [777] {other vegetables,
##        soda}                      => {root vegetables}       0.008235892
0.2515528 0.032740214 2.3078561    81
## [778] {root vegetables,
##        soda}                      => {whole milk}            0.008134215
0.4371585 0.018607016 1.7108848    80
## [779] {soda,
##        whole milk}                => {root vegetables}       0.008134215
```

```
0.2030457 0.040061007 1.8628305    80
## [780] {root vegetables,
##        yogurt}                    => {rolls/buns}            0.007219115
0.2795276 0.025826131 1.5197090    71
## [781] {rolls/buns,
##        root vegetables}           => {yogurt}                0.007219115
0.2970711 0.024300966 2.1295150    71
## [782] {rolls/buns,
##        yogurt}                    => {root vegetables}       0.007219115
0.2100592 0.034367056 1.9271753    71
## [783] {root vegetables,
##        yogurt}                    => {other vegetables}      0.012913066
0.5000000 0.025826131 2.5840778   127
## [784] {other vegetables,
##        root vegetables}           => {yogurt}                0.012913066
0.2725322 0.047381800 1.9536108   127
## [785] {other vegetables,
##        yogurt}                    => {root vegetables}       0.012913066
0.2974239 0.043416370 2.7286977   127
## [786] {root vegetables,
##        yogurt}                    => {whole milk}            0.014539908
0.5629921 0.025826131 2.2033536   143
## [787] {root vegetables,
##        whole milk}                => {yogurt}                0.014539908
0.2972973 0.048906965 2.1311362   143
## [788] {whole milk,
##        yogurt}                    => {root vegetables}       0.014539908
0.2595281 0.056024403 2.3810253   143
## [789] {rolls/buns,
##        root vegetables}           => {other vegetables}      0.012201322
0.5020921 0.024300966 2.5948898   120
## [790] {other vegetables,
##        root vegetables}           => {rolls/buns}            0.012201322
0.2575107 0.047381800 1.4000100   120
## [791] {other vegetables,
##        rolls/buns}                => {root vegetables}       0.012201322
0.2863962 0.042602949 2.6275247   120
## [792] {rolls/buns,
##        root vegetables}           => {whole milk}            0.012709710
0.5230126 0.024300966 2.0468876   125
## [793] {root vegetables,
##        whole milk}                => {rolls/buns}            0.012709710
0.2598753 0.048906965 1.4128652   125
## [794] {rolls/buns,
##        whole milk}                => {root vegetables}       0.012709710
0.2244165 0.056634469 2.0588959   125
## [795] {other vegetables,
##        root vegetables}           => {whole milk}            0.023182511
0.4892704 0.047381800 1.9148326   228
## [796] {root vegetables,
```

```
##        whole milk}              => {other vegetables}      0.023182511
0.4740125 0.048906965 2.4497702   228
## [797] {other vegetables,
##        whole milk}              => {root vegetables}       0.023182511
0.3097826 0.074834774 2.8420820   228
## [798] {soda,
##        yogurt}                  => {rolls/buns}            0.008642603
0.3159851 0.027351296 1.7179181    85
## [799] {rolls/buns,
##        soda}                    => {yogurt}                0.008642603
0.2254642 0.038332486 1.6162101    85
## [800] {rolls/buns,
##        yogurt}                  => {soda}                  0.008642603
0.2514793 0.034367056 1.4421567    85
## [801] {soda,
##        yogurt}                  => {other vegetables}      0.008337570
0.3048327 0.027351296 1.5754229    82
## [802] {other vegetables,
##        soda}                    => {yogurt}                0.008337570
0.2546584 0.032740214 1.8254849    82
## [803] {soda,
##        yogurt}                  => {whole milk}            0.010472801
0.3828996 0.027351296 1.4985348   103
## [804] {soda,
##        whole milk}              => {yogurt}                0.010472801
0.2614213 0.040061007 1.8739641   103
## [805] {rolls/buns,
##        soda}                    => {other vegetables}      0.009862735
0.2572944 0.038332486 1.3297376    97
## [806] {other vegetables,
##        soda}                    => {rolls/buns}            0.009862735
0.3012422 0.032740214 1.6377653    97
## [807] {other vegetables,
##        rolls/buns}              => {soda}                  0.009862735
0.2315036 0.042602949 1.3276022    97
## [808] {rolls/buns,
##        soda}                    => {whole milk}            0.008845958
0.2307692 0.038332486 0.9031498    87
## [809] {soda,
##        whole milk}              => {rolls/buns}            0.008845958
0.2208122 0.040061007 1.2004908    87
## [810] {other vegetables,
##        soda}                    => {whole milk}            0.013929842
0.4254658 0.032740214 1.6651240   137
## [811] {soda,
##        whole milk}              => {other vegetables}      0.013929842
0.3477157 0.040061007 1.7970490   137
## [812] {rolls/buns,
##        yogurt}                  => {other vegetables}      0.011489578
0.3343195 0.034367056 1.7278153   113
```

```
## [813] {other vegetables,
##        yogurt}                  => {rolls/buns}            0.011489578
0.2646370 0.043416370 1.4387534   113
## [814] {other vegetables,
##        rolls/buns}              => {yogurt}                0.011489578
0.2696897 0.042602949 1.9332351   113
## [815] {rolls/buns,
##        yogurt}                  => {whole milk}            0.015556685
0.4526627 0.034367056 1.7715630   153
## [816] {whole milk,
##        yogurt}                  => {rolls/buns}            0.015556685
0.2776770 0.056024403 1.5096478   153
## [817] {rolls/buns,
##        whole milk}              => {yogurt}                0.015556685
0.2746858 0.056634469 1.9690488   153
## [818] {other vegetables,
##        yogurt}                  => {whole milk}            0.022267412
0.5128806 0.043416370 2.0072345   219
## [819] {whole milk,
##        yogurt}                  => {other vegetables}      0.022267412
0.3974592 0.056024403 2.0541308   219
## [820] {other vegetables,
##        whole milk}              => {yogurt}                0.022267412
0.2975543 0.074834774 2.1329789   219
## [821] {other vegetables,
##        rolls/buns}              => {whole milk}            0.017895272
0.4200477 0.042602949 1.6439194   176
## [822] {rolls/buns,
##        whole milk}              => {other vegetables}      0.017895272
0.3159785 0.056634469 1.6330258   176
## [823] {other vegetables,
##        whole milk}              => {rolls/buns}            0.017895272
0.2391304 0.074834774 1.3000817   176
## [824] {fruit/vegetable juice,
##        other vegetables,
##        yogurt}                  => {whole milk}            0.005083884
0.6172840 0.008235892 2.4158327    50
## [825] {fruit/vegetable juice,
##        whole milk,
##        yogurt}                  => {other vegetables}      0.005083884
0.5376344 0.009456024 2.7785782    50
## [826] {fruit/vegetable juice,
##        other vegetables,
##        whole milk}              => {yogurt}                0.005083884
0.4854369 0.010472801 3.4797900    50
## [827] {other vegetables,
##        whole milk,
##        yogurt}                  => {fruit/vegetable juice} 0.005083884
0.2283105 0.022267412 3.1581347    50
## [828] {other vegetables,
```

```
##         root vegetables,
##         whipped/sour cream}        => {whole milk}            0.005185562
0.6071429 0.008540925 2.3761441    51
## [829] {root vegetables,
##         whipped/sour cream,
##         whole milk}                => {other vegetables}      0.005185562
0.5483871 0.009456024 2.8341498    51
## [830] {other vegetables,
##         whipped/sour cream,
##         whole milk}                => {root vegetables}       0.005185562
0.3541667 0.014641586 3.2492809    51
## [831] {other vegetables,
##         root vegetables,
##         whole milk}                => {whipped/sour cream}    0.005185562
0.2236842 0.023182511 3.1204741    51
## [832] {other vegetables,
##         whipped/sour cream,
##         yogurt}                    => {whole milk}            0.005592272
0.5500000 0.010167768 2.1525070    55
## [833] {whipped/sour cream,
##         whole milk,
##         yogurt}                    => {other vegetables}      0.005592272
0.5140187 0.010879512 2.6565286    55
## [834] {other vegetables,
##         whipped/sour cream,
##         whole milk}                => {yogurt}                0.005592272
0.3819444 0.014641586 2.7379181    55
## [835] {other vegetables,
##         whole milk,
##         yogurt}                    => {whipped/sour cream}    0.005592272
0.2511416 0.022267412 3.5035137    55
## [836] {other vegetables,
##         pip fruit,
##         root vegetables}           => {whole milk}            0.005490595
0.6750000 0.008134215 2.6417131    54
## [837] {pip fruit,
##         root vegetables,
##         whole milk}                => {other vegetables}      0.005490595
0.6136364 0.008947636 3.1713682    54
## [838] {other vegetables,
##         pip fruit,
##         whole milk}                => {root vegetables}       0.005490595
0.4060150 0.013523132 3.7249607    54
## [839] {other vegetables,
##         root vegetables,
##         whole milk}                => {pip fruit}             0.005490595
0.2368421 0.023182511 3.1308362    54
## [840] {other vegetables,
##         pip fruit,
##         yogurt}                    => {whole milk}            0.005083884
```

```
0.6250000 0.008134215 2.4460306    50
## [841] {pip fruit,
##        whole milk,
##        yogurt}                  => {other vegetables}      0.005083884
0.5319149 0.009557702 2.7490189    50
## [842] {other vegetables,
##        pip fruit,
##        whole milk}              => {yogurt}                0.005083884
0.3759398 0.013523132 2.6948749    50
## [843] {other vegetables,
##        whole milk,
##        yogurt}                  => {pip fruit}             0.005083884
0.2283105 0.022267412 3.0180562    50
## [844] {citrus fruit,
##        other vegetables,
##        root vegetables}         => {whole milk}            0.005795628
0.5588235 0.010371124 2.1870392    57
## [845] {citrus fruit,
##        root vegetables,
##        whole milk}              => {other vegetables}      0.005795628
0.6333333 0.009150991 3.2731652    57
## [846] {citrus fruit,
##        other vegetables,
##        whole milk}              => {root vegetables}       0.005795628
0.4453125 0.013014743 4.0854929    57
## [847] {other vegetables,
##        root vegetables,
##        whole milk}              => {citrus fruit}          0.005795628
0.2500000 0.023182511 3.0205774    57
## [848] {root vegetables,
##        tropical fruit,
##        yogurt}                  => {whole milk}            0.005693950
0.7000000 0.008134215 2.7395543    56
## [849] {root vegetables,
##        tropical fruit,
##        whole milk}              => {yogurt}                0.005693950
0.4745763 0.011997966 3.4019370    56
## [850] {tropical fruit,
##        whole milk,
##        yogurt}                  => {root vegetables}       0.005693950
0.3758389 0.015149975 3.4481118    56
## [851] {root vegetables,
##        whole milk,
##        yogurt}                  => {tropical fruit}        0.005693950
0.3916084 0.014539908 3.7320432    56
## [852] {other vegetables,
##        root vegetables,
##        tropical fruit}          => {whole milk}            0.007015760
0.5702479 0.012302999 2.2317503    69
## [853] {root vegetables,
```

```
##         tropical fruit,
##         whole milk}                => {other vegetables}      0.007015760
0.5847458 0.011997966 3.0220571    69
## [854] {other vegetables,
##         tropical fruit,
##         whole milk}                => {root vegetables}       0.007015760
0.4107143 0.017081851 3.7680737    69
## [855] {other vegetables,
##         root vegetables,
##         whole milk}                => {tropical fruit}        0.007015760
0.3026316 0.023182511 2.8840907    69
## [856] {other vegetables,
##         tropical fruit,
##         yogurt}                    => {whole milk}            0.007625826
0.6198347 0.012302999 2.4258155    75
## [857] {tropical fruit,
##         whole milk,
##         yogurt}                    => {other vegetables}      0.007625826
0.5033557 0.015149975 2.6014206    75
## [858] {other vegetables,
##         tropical fruit,
##         whole milk}                => {yogurt}                0.007625826
0.4464286 0.017081851 3.2001640    75
## [859] {other vegetables,
##         whole milk,
##         yogurt}                    => {tropical fruit}        0.007625826
0.3424658 0.022267412 3.2637119    75
## [860] {other vegetables,
##         root vegetables,
##         yogurt}                    => {whole milk}            0.007829181
0.6062992 0.012913066 2.3728423    77
## [861] {root vegetables,
##         whole milk,
##         yogurt}                    => {other vegetables}      0.007829181
0.5384615 0.014539908 2.7828530    77
## [862] {other vegetables,
##         root vegetables,
##         whole milk}                => {yogurt}                0.007829181
0.3377193 0.023182511 2.4208960    77
## [863] {other vegetables,
##         whole milk,
##         yogurt}                    => {root vegetables}       0.007829181
0.3515982 0.022267412 3.2257165    77
## [864] {other vegetables,
##         rolls/buns,
##         root vegetables}           => {whole milk}            0.006202339
0.5083333 0.012201322 1.9894383    61
## [865] {rolls/buns,
##         root vegetables,
##         whole milk}                => {other vegetables}      0.006202339
```

```
0.4880000 0.012709710 2.5220599    61
## [866] {other vegetables,
##        root vegetables,
##        whole milk}                 => {rolls/buns}           0.006202339
0.2675439 0.023182511 1.4545571    61
## [867] {other vegetables,
##        rolls/buns,
##        whole milk}                 => {root vegetables}      0.006202339
0.3465909 0.017895272 3.1797776    61
## [868] {other vegetables,
##        rolls/buns,
##        yogurt}                     => {whole milk}           0.005998983
0.5221239 0.011489578 2.0434097    59
## [869] {rolls/buns,
##        whole milk,
##        yogurt}                     => {other vegetables}     0.005998983
0.3856209 0.015556685 1.9929489    59
## [870] {other vegetables,
##        whole milk,
##        yogurt}                     => {rolls/buns}           0.005998983
0.2694064 0.022267412 1.4646832    59
## [871] {other vegetables,
##        rolls/buns,
##        whole milk}                 => {yogurt}               0.005998983
0.3352273 0.017895272 2.4030322    59
```

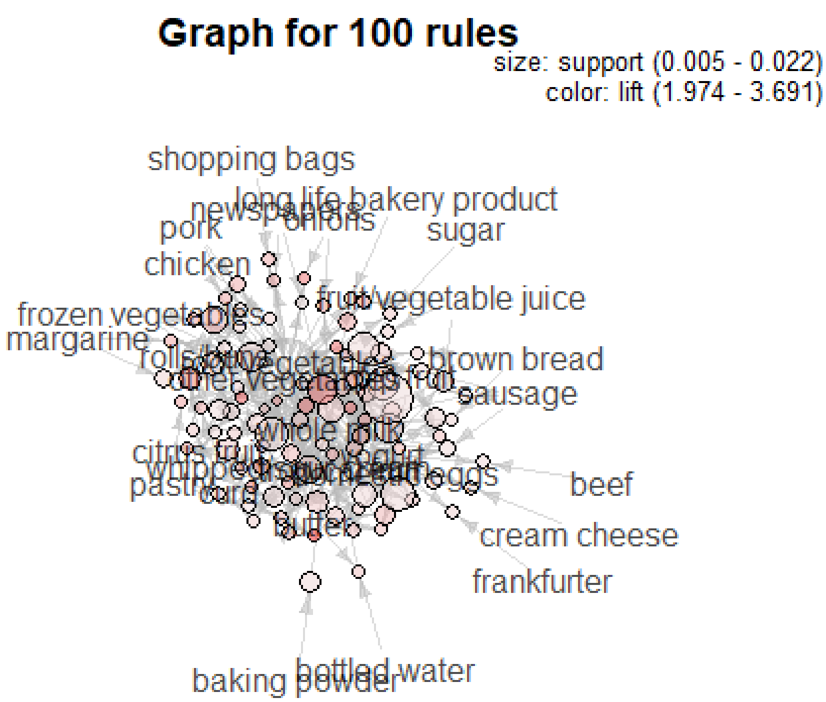
I then determined that I wanted only associate rules that were higher than 0.5 confidence level and around .005 support. I created a subset based on this criteria. Again, you can see that the number of rules is still high.

```
library(dplyr)
library(arules)
library(reshape)
library(tidyverse)
library(arules)  # has a big ecosystem of packages built around it
library(arulesViz)
baskets <-
read.delim("C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/groceries.tx
t",header = FALSE)

baskets <- cbind(Baskets = rownames(baskets), baskets)
rownames(baskets) <- 1:nrow(baskets)

library(splitstackshape)
nbaskets <- concat.split(baskets, "V1", ",")
# baskets1= split(x=baskets_1[,-1], f=baskets$Baskets)
```

```r
mbaskets <- nbaskets[,-2]

mbaskets <- melt(mbaskets, id=c("Baskets"))
mbaskets <- mbaskets[,-2]

mbaskets$Baskets = factor(mbaskets$Baskets )
mbaskets <- na.omit(mbaskets)


baskets1= split(x=mbaskets$value, f=mbaskets$Baskets)
baskets = lapply(baskets1, unique)
baskettrans = as(baskets, "transactions")
basketrules = apriori(baskettrans,
                      parameter=list(support=.005, confidence=.2, maxlen=5))

## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.2    0.1    1 none FALSE            TRUE       5   0.005      1
##  maxlen target  ext
##       5  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 49
##
## 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 ... [873 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].

sub1 = subset(basketrules, subset=confidence > 0.5 & support > 0.005)
summary(sub1)

## set of 113 rules
##
## rule length distribution (lhs + rhs):sizes
##  2  3  4
##  1 91 21
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   2.000   3.000   3.000   3.177   3.000   4.000
##
## summary of quality measures:
```

```
##     support           confidence        coverage             lift       
##  Min.   :0.005084   Min.   :0.5021   Min.   :0.008134   Min.   :1.974  
##  1st Qu.:0.005694   1st Qu.:0.5221   1st Qu.:0.009964   1st Qu.:2.109  
##  Median :0.006202   Median :0.5484   Median :0.011388   Median :2.268  
##  Mean   :0.007315   Mean   :0.5570   Mean   :0.013268   Mean   :2.394  
##  3rd Qu.:0.007931   3rd Qu.:0.5824   3rd Qu.:0.014642   3rd Qu.:2.657  
##  Max.   :0.022267   Max.   :0.7000   Max.   :0.043416   Max.   :3.691  
##      count       
##  Min.   : 50.00  
##  1st Qu.: 56.00  
##  Median : 61.00  
##  Mean   : 71.95  
##  3rd Qu.: 78.00  
##  Max.   :219.00  
## 
## mining info:
##         data ntransactions support confidence
##  baskettrans          9835   0.005        0.2
```

```r
plot(sub1, method='graph')
```




```r
inspect(sub1)
```

```
##       lhs                                rhs                        support
confidence    coverage      lift count
## [1]   {baking powder}                 => {whole milk}        0.009252669
0.5229885 0.017691917 2.046793    91
```

```
## [2]   {oil,
##        other vegetables}         => {whole milk}       0.005083884
0.5102041 0.009964413 1.996760    50
## [3]   {onions,
##        root vegetables}          => {other vegetables} 0.005693950
0.6021505 0.009456024 3.112008    56
## [4]   {onions,
##        whole milk}               => {other vegetables} 0.006609049
0.5462185 0.012099644 2.822942    65
## [5]   {hygiene articles,
##        other vegetables}         => {whole milk}       0.005185562
0.5425532 0.009557702 2.123363    51
## [6]   {other vegetables,
##        sugar}                    => {whole milk}       0.006304016
0.5849057 0.010777834 2.289115    62
## [7]   {long life bakery product,
##        other vegetables}         => {whole milk}       0.005693950
0.5333333 0.010676157 2.087279    56
## [8]   {cream cheese,
##        yogurt}                   => {whole milk}       0.006609049
0.5327869 0.012404677 2.085141    65
## [9]   {chicken,
##        root vegetables}          => {other vegetables} 0.005693950
0.5233645 0.010879512 2.704829    56
## [10]  {chicken,
##        root vegetables}          => {whole milk}       0.005998983
0.5514019 0.010879512 2.157993    59
## [11]  {chicken,
##        rolls/buns}               => {whole milk}       0.005287239
0.5473684 0.009659380 2.142208    52
## [12]  {coffee,
##        yogurt}                   => {whole milk}       0.005083884
0.5208333 0.009761057 2.038359    50
## [13]  {frozen vegetables,
##        root vegetables}          => {other vegetables} 0.006100661
0.5263158 0.011591256 2.720082    60
## [14]  {frozen vegetables,
##        root vegetables}          => {whole milk}       0.006202339
0.5350877 0.011591256 2.094146    61
## [15]  {frozen vegetables,
##        other vegetables}         => {whole milk}       0.009659380
0.5428571 0.017793594 2.124552    95
## [16]  {beef,
##        yogurt}                   => {whole milk}       0.006100661
0.5217391 0.011692933 2.041904    60
## [17]  {curd,
##        whipped/sour cream}       => {whole milk}       0.005897306
0.5631068 0.010472801 2.203802    58
## [18]  {curd,
##        tropical fruit}           => {yogurt}           0.005287239
```

```
0.5148515 0.010269446 3.690645    52
## [19]  {curd,
##        tropical fruit}           => {other vegetables} 0.005287239
0.5148515 0.010269446 2.660833    52
## [20]  {curd,
##        tropical fruit}           => {whole milk}       0.006507372
0.6336634 0.010269446 2.479936    64
## [21]  {curd,
##        root vegetables}          => {other vegetables} 0.005490595
0.5046729 0.010879512 2.608228    54
## [22]  {curd,
##        root vegetables}          => {whole milk}       0.006202339
0.5700935 0.010879512 2.231146    61
## [23]  {curd,
##        yogurt}                   => {whole milk}       0.010066090
0.5823529 0.017285206 2.279125    99
## [24]  {curd,
##        rolls/buns}               => {whole milk}       0.005897306
0.5858586 0.010066090 2.292845    58
## [25]  {curd,
##        other vegetables}         => {whole milk}       0.009862735
0.5739645 0.017183528 2.246296    97
## [26]  {pork,
##        root vegetables}          => {other vegetables} 0.007015760
0.5149254 0.013624809 2.661214    69
## [27]  {pork,
##        rolls/buns}               => {whole milk}       0.006202339
0.5495495 0.011286223 2.150744    61
## [28]  {frankfurter,
##        tropical fruit}           => {whole milk}       0.005185562
0.5483871 0.009456024 2.146195    51
## [29]  {frankfurter,
##        yogurt}                   => {whole milk}       0.006202339
0.5545455 0.011184545 2.170296    61
## [30]  {bottled beer,
##        yogurt}                   => {whole milk}       0.005185562
0.5604396 0.009252669 2.193364    51
## [31]  {brown bread,
##        tropical fruit}           => {whole milk}       0.005693950
0.5333333 0.010676157 2.087279    56
## [32]  {brown bread,
##        root vegetables}          => {whole milk}       0.005693950
0.5600000 0.010167768 2.191643    56
## [33]  {domestic eggs,
##        margarine}                => {whole milk}       0.005185562
0.6219512 0.008337570 2.434099    51
## [34]  {margarine,
##        root vegetables}          => {other vegetables} 0.005897306
0.5321101 0.011082867 2.750028    58
## [35]  {margarine,
```

```
##        rolls/buns}                 => {whole milk}       0.007930859
0.5379310 0.014743264 2.105273    78
## [36]  {butter,
##        domestic eggs}              => {whole milk}       0.005998983
0.6210526 0.009659380 2.430582    59
## [37]  {butter,
##        whipped/sour cream}         => {other vegetables} 0.005795628
0.5700000 0.010167768 2.945849    57
## [38]  {butter,
##        whipped/sour cream}         => {whole milk}       0.006710727
0.6600000 0.010167768 2.583008    66
## [39]  {butter,
##        citrus fruit}               => {whole milk}       0.005083884
0.5555556 0.009150991 2.174249    50
## [40]  {bottled water,
##        butter}                     => {whole milk}       0.005388917
0.6022727 0.008947636 2.357084    53
## [41]  {butter,
##        tropical fruit}             => {other vegetables} 0.005490595
0.5510204 0.009964413 2.847759    54
## [42]  {butter,
##        tropical fruit}             => {whole milk}       0.006202339
0.6224490 0.009964413 2.436047    61
## [43]  {butter,
##        root vegetables}            => {other vegetables} 0.006609049
0.5118110 0.012913066 2.645119    65
## [44]  {butter,
##        root vegetables}            => {whole milk}       0.008235892
0.6377953 0.012913066 2.496107    81
## [45]  {butter,
##        yogurt}                     => {whole milk}       0.009354347
0.6388889 0.014641586 2.500387    92
## [46]  {butter,
##        other vegetables}           => {whole milk}       0.011489578
0.5736041 0.020030503 2.244885   113
## [47]  {newspapers,
##        root vegetables}            => {other vegetables} 0.005998983
0.5221239 0.011489578 2.698417    59
## [48]  {newspapers,
##        root vegetables}            => {whole milk}       0.005795628
0.5044248 0.011489578 1.974142    57
## [49]  {domestic eggs,
##        whipped/sour cream}         => {other vegetables} 0.005083884
0.5102041 0.009964413 2.636814    50
## [50]  {domestic eggs,
##        whipped/sour cream}         => {whole milk}       0.005693950
0.5714286 0.009964413 2.236371    56
## [51]  {domestic eggs,
##        pip fruit}                  => {whole milk}       0.005388917
0.6235294 0.008642603 2.440275    53
```

```
## [52]  {citrus fruit,
##        domestic eggs}            => {whole milk}       0.005693950
0.5490196 0.010371124 2.148670    56
## [53]  {domestic eggs,
##        tropical fruit}           => {whole milk}       0.006914082
0.6071429 0.011387900 2.376144    68
## [54]  {domestic eggs,
##        root vegetables}          => {other vegetables} 0.007320793
0.5106383 0.014336553 2.639058    72
## [55]  {domestic eggs,
##        root vegetables}          => {whole milk}       0.008540925
0.5957447 0.014336553 2.331536    84
## [56]  {domestic eggs,
##        yogurt}                   => {whole milk}       0.007727504
0.5390071 0.014336553 2.109485    76
## [57]  {domestic eggs,
##        other vegetables}         => {whole milk}       0.012302999
0.5525114 0.022267412 2.162336   121
## [58]  {fruit/vegetable juice,
##        root vegetables}          => {other vegetables} 0.006609049
0.5508475 0.011997966 2.846865    65
## [59]  {fruit/vegetable juice,
##        root vegetables}          => {whole milk}       0.006507372
0.5423729 0.011997966 2.122657    64
## [60]  {fruit/vegetable juice,
##        yogurt}                   => {whole milk}       0.009456024
0.5054348 0.018708693 1.978094    93
## [61]  {pip fruit,
##        whipped/sour cream}       => {other vegetables} 0.005592272
0.6043956 0.009252669 3.123610    55
## [62]  {pip fruit,
##        whipped/sour cream}       => {whole milk}       0.005998983
0.6483516 0.009252669 2.537421    59
## [63]  {citrus fruit,
##        whipped/sour cream}       => {other vegetables} 0.005693950
0.5233645 0.010879512 2.704829    56
## [64]  {citrus fruit,
##        whipped/sour cream}       => {whole milk}       0.006304016
0.5794393 0.010879512 2.267722    62
## [65]  {sausage,
##        whipped/sour cream}       => {whole milk}       0.005083884
0.5617978 0.009049314 2.198679    50
## [66]  {tropical fruit,
##        whipped/sour cream}       => {other vegetables} 0.007829181
0.5661765 0.013828165 2.926088    77
## [67]  {tropical fruit,
##        whipped/sour cream}       => {whole milk}       0.007930859
0.5735294 0.013828165 2.244593    78
## [68]  {root vegetables,
##        whipped/sour cream}       => {whole milk}       0.009456024
```

```
0.5535714 0.017081851 2.166484    93
## [69]  {whipped/sour cream,
##        yogurt}                     => {whole milk}       0.010879512
0.5245098 0.020742247 2.052747   107
## [70]  {rolls/buns,
##        whipped/sour cream}         => {whole milk}       0.007829181
0.5347222 0.014641586 2.092715    77
## [71]  {other vegetables,
##        whipped/sour cream}         => {whole milk}       0.014641586
0.5070423 0.028876462 1.984385   144
## [72]  {pip fruit,
##        sausage}                    => {whole milk}       0.005592272
0.5188679 0.010777834 2.030667    55
## [73]  {pip fruit,
##        root vegetables}            => {other vegetables} 0.008134215
0.5228758 0.015556685 2.702304    80
## [74]  {pip fruit,
##        root vegetables}            => {whole milk}       0.008947636
0.5751634 0.015556685 2.250988    88
## [75]  {pip fruit,
##        yogurt}                     => {whole milk}       0.009557702
0.5310734 0.017996950 2.078435    94
## [76]  {other vegetables,
##        pip fruit}                  => {whole milk}       0.013523132
0.5175097 0.026131164 2.025351   133
## [77]  {pastry,
##        tropical fruit}             => {whole milk}       0.006710727
0.5076923 0.013218099 1.986930    66
## [78]  {pastry,
##        root vegetables}            => {other vegetables} 0.005897306
0.5370370 0.010981190 2.775491    58
## [79]  {pastry,
##        root vegetables}            => {whole milk}       0.005693950
0.5185185 0.010981190 2.029299    56
## [80]  {pastry,
##        yogurt}                     => {whole milk}       0.009150991
0.5172414 0.017691917 2.024301    90
## [81]  {citrus fruit,
##        root vegetables}            => {other vegetables} 0.010371124
0.5862069 0.017691917 3.029608   102
## [82]  {citrus fruit,
##        root vegetables}            => {whole milk}       0.009150991
0.5172414 0.017691917 2.024301    90
## [83]  {root vegetables,
##        shopping bags}              => {other vegetables} 0.006609049
0.5158730 0.012811388 2.666112    65
## [84]  {sausage,
##        tropical fruit}             => {whole milk}       0.007219115
0.5182482 0.013929842 2.028241    71
## [85]  {root vegetables,
```

```
##        sausage}                     => {whole milk}       0.007727504
0.5170068 0.014946619 2.023383    76
## [86]  {root vegetables,
##        tropical fruit}              => {other vegetables} 0.012302999
0.5845411 0.021047280 3.020999   121
## [87]  {root vegetables,
##        tropical fruit}              => {whole milk}       0.011997966
0.5700483 0.021047280 2.230969   118
## [88]  {tropical fruit,
##        yogurt}                      => {whole milk}       0.015149975
0.5173611 0.029283172 2.024770   149
## [89]  {root vegetables,
##        yogurt}                      => {whole milk}       0.014539908
0.5629921 0.025826131 2.203354   143
## [90]  {rolls/buns,
##        root vegetables}             => {other vegetables} 0.012201322
0.5020921 0.024300966 2.594890   120
## [91]  {rolls/buns,
##        root vegetables}             => {whole milk}       0.012709710
0.5230126 0.024300966 2.046888   125
## [92]  {other vegetables,
##        yogurt}                      => {whole milk}       0.022267412
0.5128806 0.043416370 2.007235   219
## [93]  {fruit/vegetable juice,
##        other vegetables,
##        yogurt}                      => {whole milk}       0.005083884
0.6172840 0.008235892 2.415833    50
## [94]  {fruit/vegetable juice,
##        whole milk,
##        yogurt}                      => {other vegetables} 0.005083884
0.5376344 0.009456024 2.778578    50
## [95]  {other vegetables,
##        root vegetables,
##        whipped/sour cream}          => {whole milk}       0.005185562
0.6071429 0.008540925 2.376144    51
## [96]  {root vegetables,
##        whipped/sour cream,
##        whole milk}                  => {other vegetables} 0.005185562
0.5483871 0.009456024 2.834150    51
## [97]  {other vegetables,
##        whipped/sour cream,
##        yogurt}                      => {whole milk}       0.005592272
0.5500000 0.010167768 2.152507    55
## [98]  {whipped/sour cream,
##        whole milk,
##        yogurt}                      => {other vegetables} 0.005592272
0.5140187 0.010879512 2.656529    55
## [99]  {other vegetables,
##        pip fruit,
##        root vegetables}             => {whole milk}       0.005490595
```

```
0.6750000 0.008134215 2.641713    54
## [100] {pip fruit,
##        root vegetables,
##        whole milk}                 => {other vegetables} 0.005490595
0.6136364 0.008947636 3.171368    54
## [101] {other vegetables,
##        pip fruit,
##        yogurt}                     => {whole milk}       0.005083884
0.6250000 0.008134215 2.446031    50
## [102] {pip fruit,
##        whole milk,
##        yogurt}                     => {other vegetables} 0.005083884
0.5319149 0.009557702 2.749019    50
## [103] {citrus fruit,
##        other vegetables,
##        root vegetables}            => {whole milk}       0.005795628
0.5588235 0.010371124 2.187039    57
## [104] {citrus fruit,
##        root vegetables,
##        whole milk}                 => {other vegetables} 0.005795628
0.6333333 0.009150991 3.273165    57
## [105] {root vegetables,
##        tropical fruit,
##        yogurt}                     => {whole milk}       0.005693950
0.7000000 0.008134215 2.739554    56
## [106] {other vegetables,
##        root vegetables,
##        tropical fruit}             => {whole milk}       0.007015760
0.5702479 0.012302999 2.231750    69
## [107] {root vegetables,
##        tropical fruit,
##        whole milk}                 => {other vegetables} 0.007015760
0.5847458 0.011997966 3.022057    69
## [108] {other vegetables,
##        tropical fruit,
##        yogurt}                     => {whole milk}       0.007625826
0.6198347 0.012302999 2.425816    75
## [109] {tropical fruit,
##        whole milk,
##        yogurt}                     => {other vegetables} 0.007625826
0.5033557 0.015149975 2.601421    75
## [110] {other vegetables,
##        root vegetables,
##        yogurt}                     => {whole milk}       0.007829181
0.6062992 0.012913066 2.372842    77
## [111] {root vegetables,
##        whole milk,
##        yogurt}                     => {other vegetables} 0.007829181
0.5384615 0.014539908 2.782853    77
## [112] {other vegetables,
```

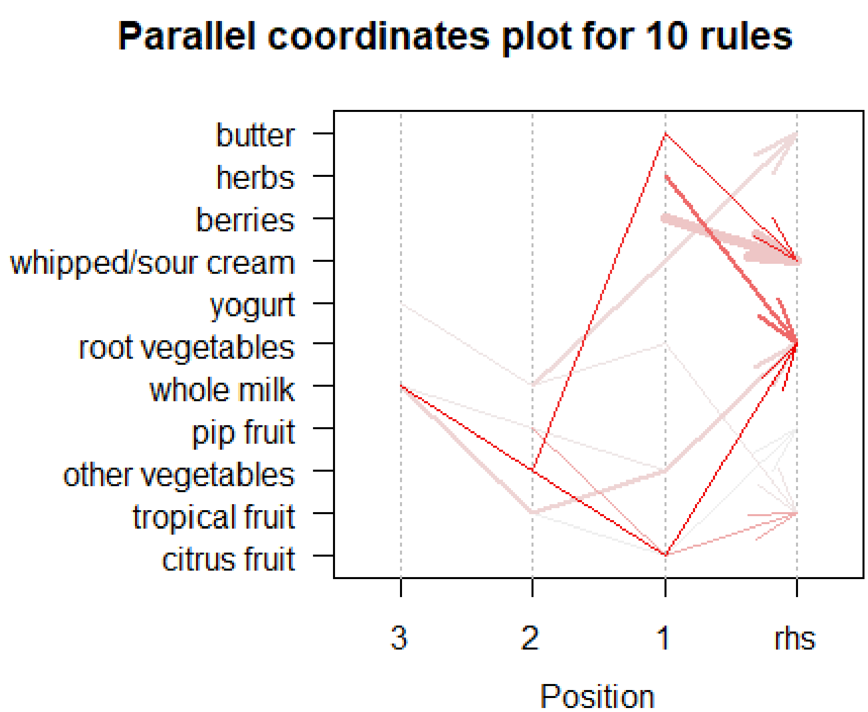
```
##        rolls/buns,
##        root vegetables}          => {whole milk}       0.006202339
0.5083333 0.012201322 1.989438    61
## [113] {other vegetables,
##        rolls/buns,
##        yogurt}                   => {whole milk}       0.005998983
0.5221239 0.011489578 2.043410    59
```

To get a better idea about the association rules, I plotted the top ten association rules based on lift in a parallel coordinate plot. The y value represents the basket item or the rule itself while the x axis is the position in the association rule. As you can see, curd seems to be what people purchase with other items. Almost all of the top 10 rules have curd as the RHS. It is also interesting that if you bought curd and then tropical fruit. You are likely to buy whole milk.

```
library(dplyr)
library(arules)
library(reshape)
library(tidyverse)
library(arules)  # has a big ecosystem of packages built around it
library(arulesViz)
baskets <-
read.delim("C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/groceries.tx
t",header = FALSE)

baskets <- cbind(Baskets = rownames(baskets), baskets)
rownames(baskets) <- 1:nrow(baskets)

library(splitstackshape)
nbaskets <- concat.split(baskets, "V1", ",")
# baskets1= split(x=baskets_1[,-1], f=baskets$Baskets)

mbaskets <- nbaskets[,-2]

mbaskets <- melt(mbaskets, id=c("Baskets"))
mbaskets <- mbaskets[,-2]

mbaskets$Baskets = factor(mbaskets$Baskets )
mbaskets <- na.omit(mbaskets)


baskets1= split(x=mbaskets$value, f=mbaskets$Baskets)
baskets = lapply(baskets1, unique)
baskettrans = as(baskets, "transactions")
basketrules = apriori(baskettrans,
                      parameter=list(support=.005, confidence=.2, maxlen=5))

## Apriori
##
```

```
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.2    0.1    1 none FALSE            TRUE       5   0.005      1
##  maxlen target  ext
##       5  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 49
##
## 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 ... [873 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
sub1 = subset(basketrules, subset=confidence > 0.5 & support > 0.005)
subRules2<-head(sub1, n=20, by="lift")
subRules2 <- head(sort(basketrules, by="lift"), 10)
plot(subRules2, method="paracoord",reorder=TRUE)
```

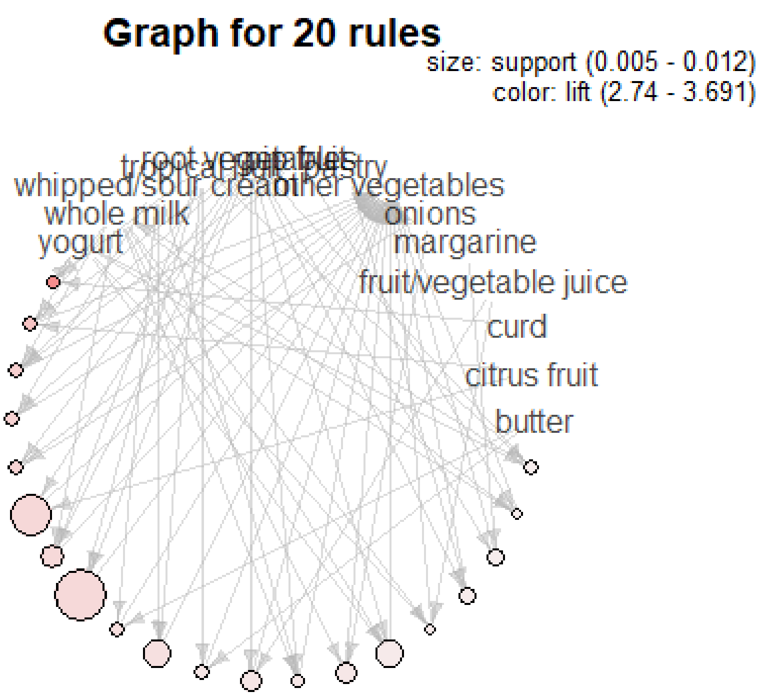


To get a better idea about the association rules, I plotted the top 20 rules based on lift in circular format. In addition, I also plotted the top 20 rules unformatted too as I believe that the different views provided an interesting perspective.

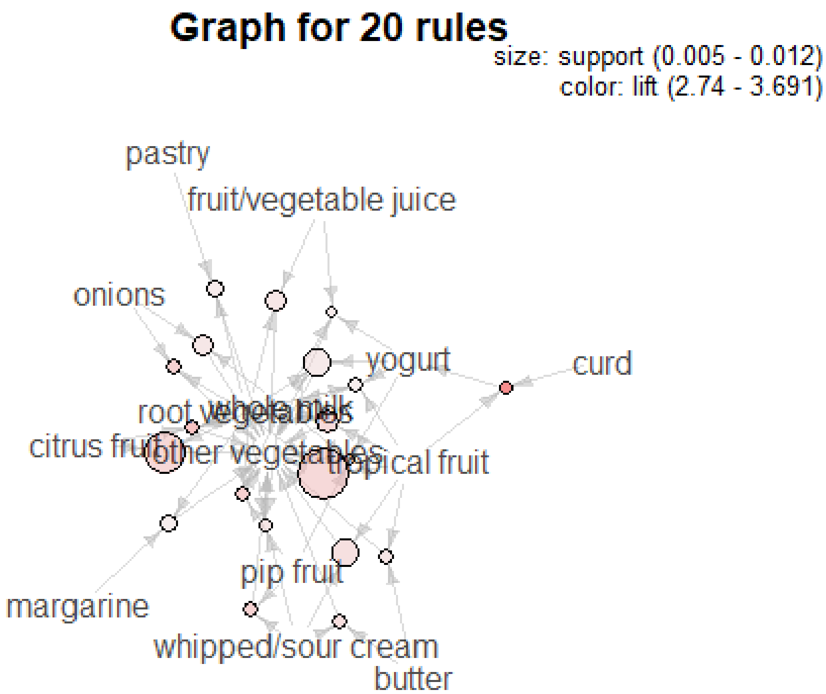
```
library(dplyr)
library(arules)
library(reshape)
library(tidyverse)
library(arules)  # has a big ecosystem of packages built around it
library(arulesViz)
baskets <-
read.delim("C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/groceries.tx
t",header = FALSE)

baskets <- cbind(Baskets = rownames(baskets), baskets)
rownames(baskets) <- 1:nrow(baskets)

library(splitstackshape)
nbaskets <- concat.split(baskets, "V1", ",")
# baskets1= split(x=baskets_1[,-1], f=baskets$Baskets)

mbaskets <- nbaskets[,-2]

mbaskets <- melt(mbaskets, id=c("Baskets"))
mbaskets <- mbaskets[,-2]

mbaskets$Baskets = factor(mbaskets$Baskets )
mbaskets <- na.omit(mbaskets)


baskets1= split(x=mbaskets$value, f=mbaskets$Baskets)
baskets = lapply(baskets1, unique)
baskettrans = as(baskets, "transactions")
basketrules = apriori(baskettrans,
                      parameter=list(support=.005, confidence=.2, maxlen=5))

## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.2    0.1    1 none FALSE            TRUE       5   0.005      1
##  maxlen target  ext
##       5  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 49
```

```
##
## 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 ... [873 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```r
sub1 = subset(basketrules, subset=confidence > 0.5 & support > 0.005)
subRules2<-head(sub1, n=20, by="lift")
subRules2 <- head(sort(basketrules, by="lift"), 10)
plot(head(sub1, 20, by='lift'), method='graph',
control=list(layout=igraph::in_circle()))
```



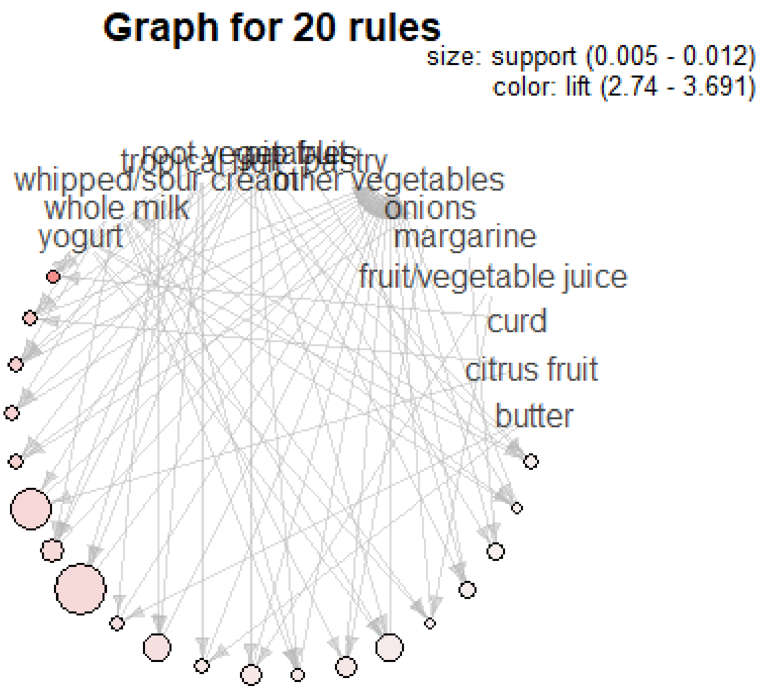

```r
plot(head(sub1, 20, by='lift'), method='graph')
```



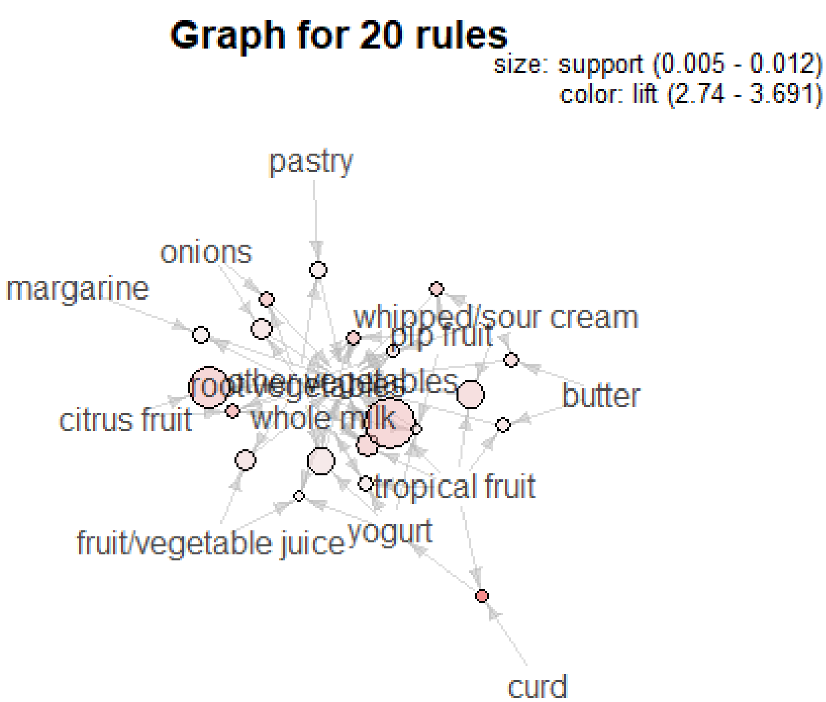
I also included grouped matrix that provided a different view of the top 20 rules. It shows very similar information as the parallel coordinate plot. I also plotted the basket items that would usually not be purchased together along with an inspection of these association rules.

```
#lower end of the lift
library(dplyr)
library(arules)
library(reshape)
library(tidyverse)
library(arules)  # has a big ecosystem of packages built around it
library(arulesViz)
baskets <-
read.delim("C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/groceries.tx
t",header = FALSE)

baskets <- cbind(Baskets = rownames(baskets), baskets)
rownames(baskets) <- 1:nrow(baskets)

library(splitstackshape)
nbaskets <- concat.split(baskets, "V1", ",")
# baskets1= split(x=baskets_1[,-1], f=baskets$Baskets)

mbaskets <- nbaskets[,-2]

mbaskets <- melt(mbaskets, id=c("Baskets"))
```

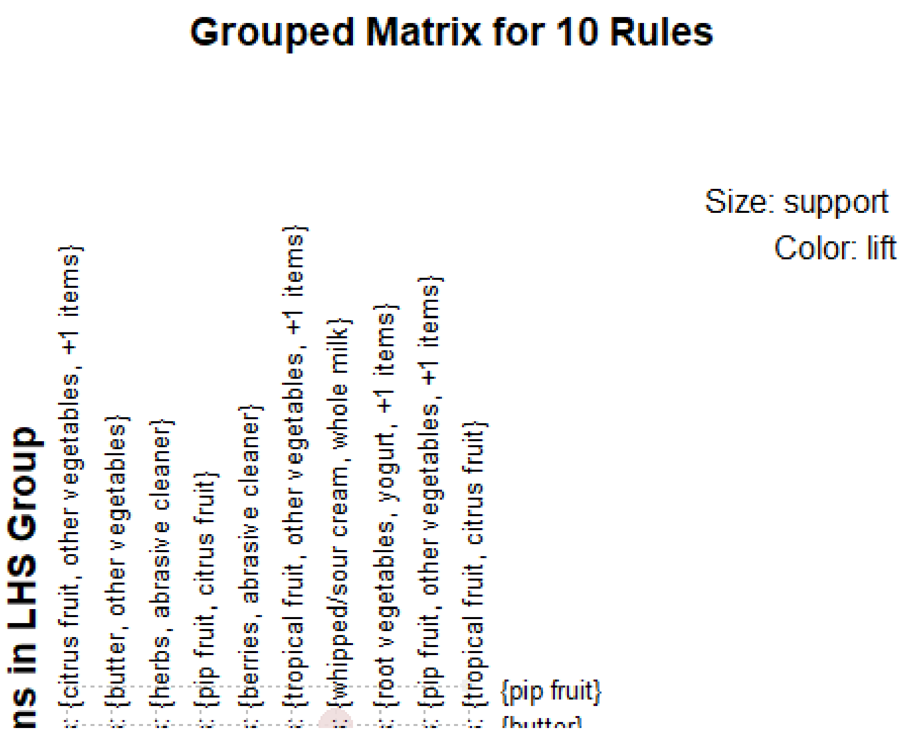
```r
mbaskets <- mbaskets[,-2]

mbaskets$Baskets = factor(mbaskets$Baskets )
mbaskets <- na.omit(mbaskets)


baskets1= split(x=mbaskets$value, f=mbaskets$Baskets)
baskets = lapply(baskets1, unique)
baskettrans = as(baskets, "transactions")
basketrules = apriori(baskettrans,
                      parameter=list(support=.005, confidence=.2, maxlen=5))

## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.2    0.1    1 none FALSE            TRUE       5   0.005      1
##  maxlen target  ext
##       5  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 49
##
## 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 ... [873 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].

sub1 = subset(basketrules, subset=confidence > 0.5 & support > 0.005)
subRules2<-head(sub1, n=20, by="lift")
subRules2 <- head(sort(basketrules, by="lift"), 10)
plot(head(sub1, 20, by='lift'), method='graph',
control=list(layout=igraph::in_circle()))
```




```r
plot(head(sub1, 20, by='lift'), method='graph')
```




```r
plot(subRules2, method = 'grouped')
```




```
inspect(tail(sort(basketrules, by = "lift")))

##     lhs                    rhs          support     confidence coverage
## [1] {misc. beverages} => {whole milk} 0.007015760 0.2473118  0.02836807
## [2] {chewing gum}     => {whole milk} 0.005083884 0.2415459  0.02104728
## [3] {specialty bar}   => {whole milk} 0.006507372 0.2379182  0.02735130
## [4] {ice cream}       => {whole milk} 0.005897306 0.2357724  0.02501271
## [5] {rolls/buns,soda} => {whole milk} 0.008845958 0.2307692  0.03833249
## [6] {soda}            => {whole milk} 0.040061007 0.2297376  0.17437722
##     lift      count
## [1] 0.9678917  69
## [2] 0.9453259  50
## [3] 0.9311284  64
## [4] 0.9227303  58
## [5] 0.9031498  87
## [6] 0.8991124 394
```

```
inspect(head(sort(basketrules, by = "lift")))

##     lhs                     rhs                        support confidence
coverage     lift count
## [1] {citrus fruit,
##      other vegetables,
##      whole milk}        => {root vegetables}    0.005795628  0.4453125
0.01301474 4.085493    57
## [2] {butter,
##      other vegetables} => {whipped/sour cream} 0.005795628  0.2893401
```

```
0.02003050 4.036397    57
## [3] {herbs}              => {root vegetables}    0.007015760  0.4312500
0.01626843 3.956477    69
## [4] {citrus fruit,
##      pip fruit}          => {tropical fruit}     0.005592272  0.4044118
0.01382816 3.854060    55
## [5] {berries}            => {whipped/sour cream} 0.009049314  0.2721713
0.03324860 3.796886    89
## [6] {other vegetables,
##      tropical fruit,
##      whole milk}         => {root vegetables}    0.007015760  0.4107143
0.01708185 3.768074    69
```

Lastly, I plotted the larger, filtered network of basket items.