Final Exam STA380

MacAlistair Husted(MWH2359)

8/17/2020

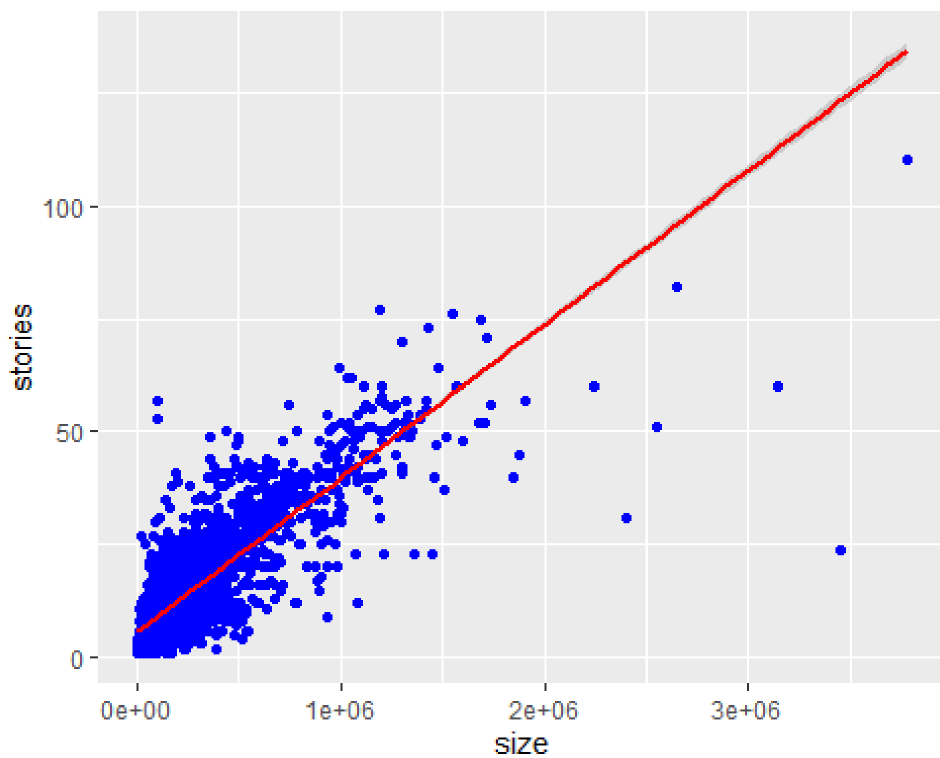
Table of Contents

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)
```

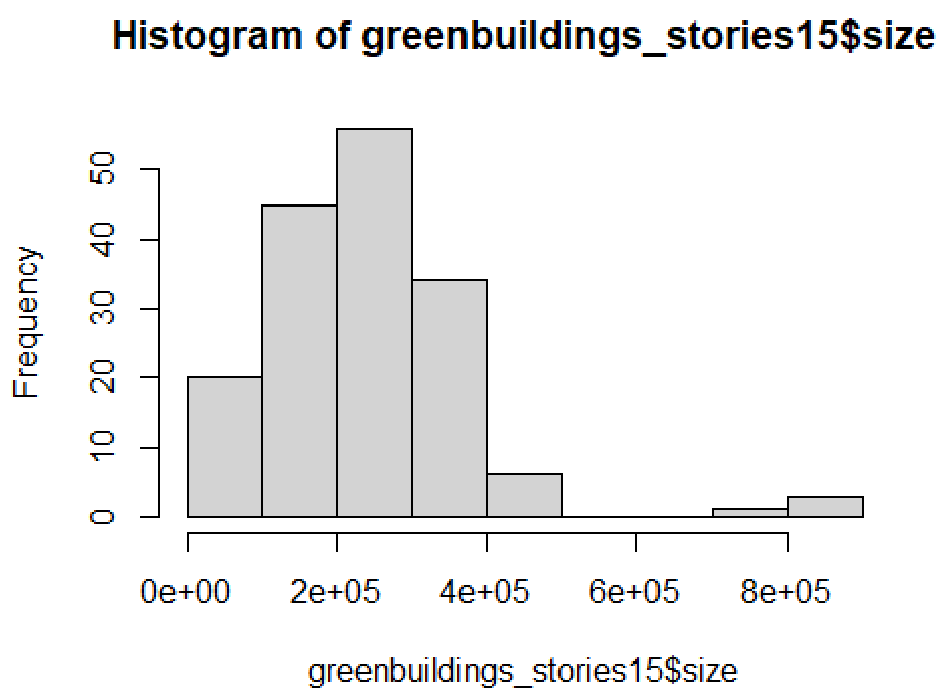
```r
greenbuildings$renovated <- as.factor(greenbuildings$renovated)
greenbuildings$Energystar <- as.factor(greenbuildings$Energystar)
greenbuildings$LEED <- as.factor(greenbuildings$LEED)
greenbuildings <- subset(greenbuildings,greenbuildings$leasing_rate > 0.1)
greenbuildings <- na.omit(greenbuildings)
#correlation of stories and size of building
cor(greenbuildings$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.

```r
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)
```

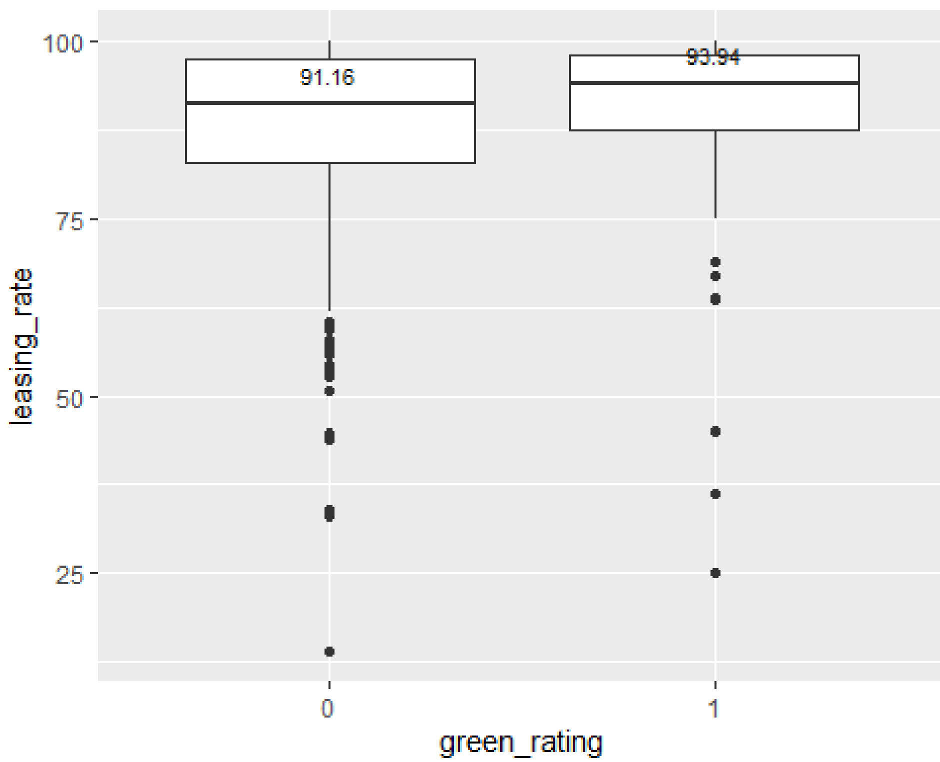
```
greenbuildings <- na.omit(greenbuildings)
greenbuildings_stories15 <- subset(greenbuildings, greenbuildings$stories ==
15)
hist(greenbuildings_stories15$size)
```



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)
```

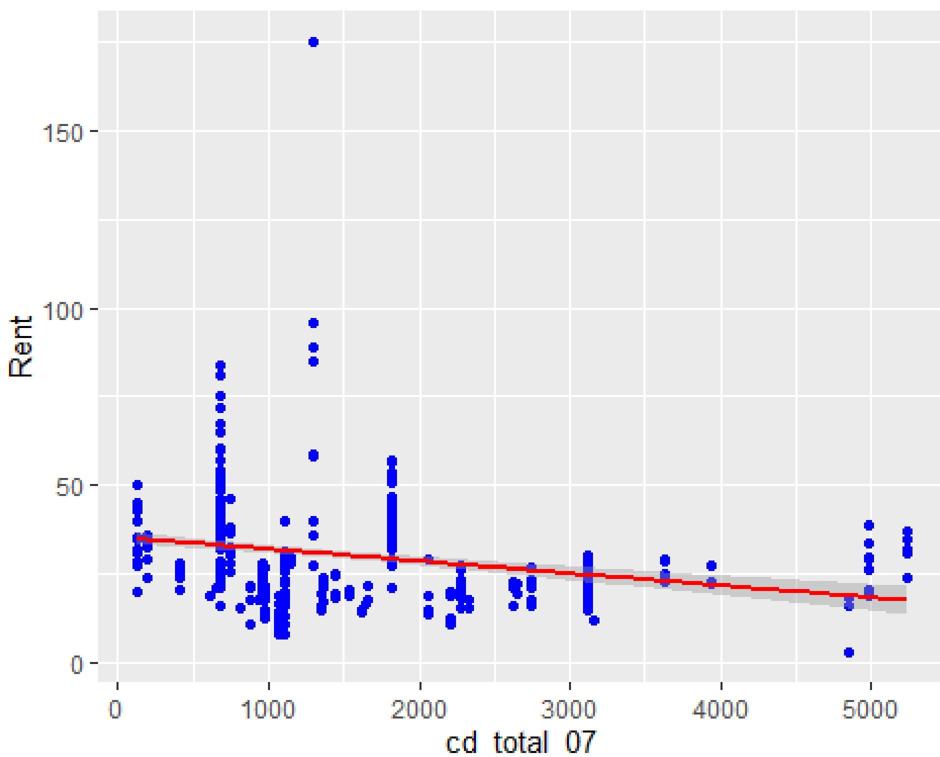
```r
greenbuildings <- subset(greenbuildings,greenbuildings$leasing_rate > 0.1)
greenbuildings <- na.omit(greenbuildings)
subset_greenbuildings <- subset(greenbuildings, greenbuildings$size > 250000
& greenbuildings$size < 350000)
green_subset_medians <- ddply(subset_greenbuildings, .(green_rating),
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)
```



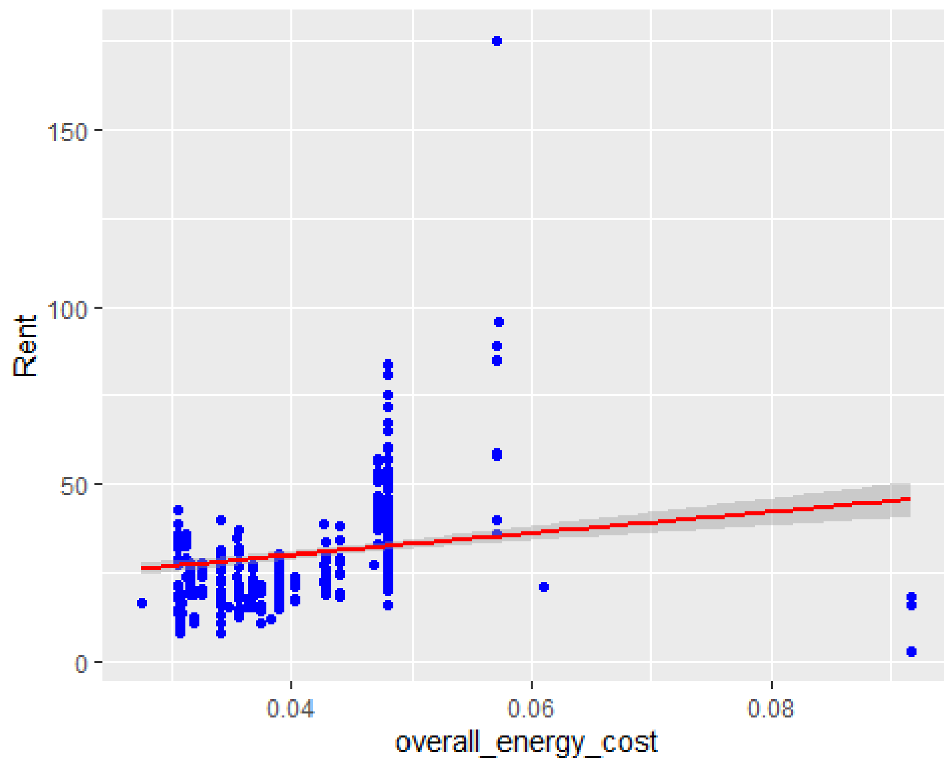
Now, I analyzed the data further to find possible confounding variables that the stat guru missed in her analysis. The first variable is cd_total_07 which is the number of cooling degree days in the region. This is an especially important variable as Austin is on average quite hot. This variable was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

```r
library(mosaic)
library(tidyverse)
library(ggplot2)
library(boot)
library(caret)
library(psych)
require(gridExtra)
library(reshape2)
library(plyr)
greenbuildings <-
```

```r
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(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

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")
```

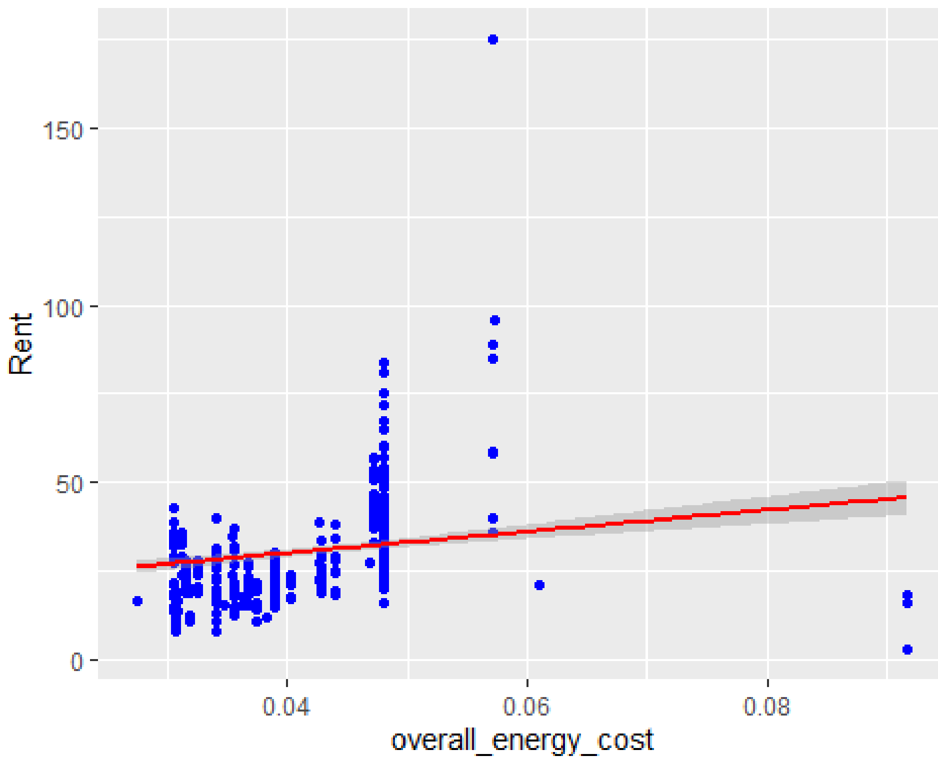



```
cor(subset_greenbuildings$Rent, subset_greenbuildings$overall_energy_cost)

## [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.

```
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.

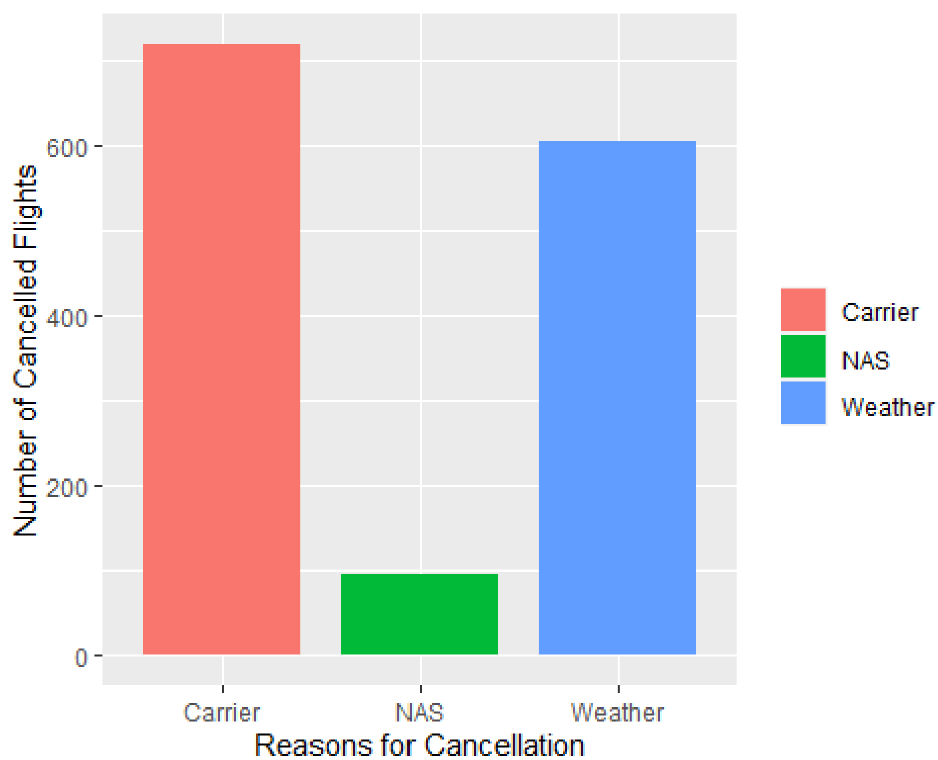
```r
library(mosaic)
library(tidyverse)
library(ggplot2)
library(boot)
library(caret)
library(psych)
require(gridExtra)
library(reshape2)
library(plyr)
greenbuildings <-
read.csv("C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/greenbuildings
.csv")
greenbuildings$green_rating <- as.factor(greenbuildings$green_rating)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

```r
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

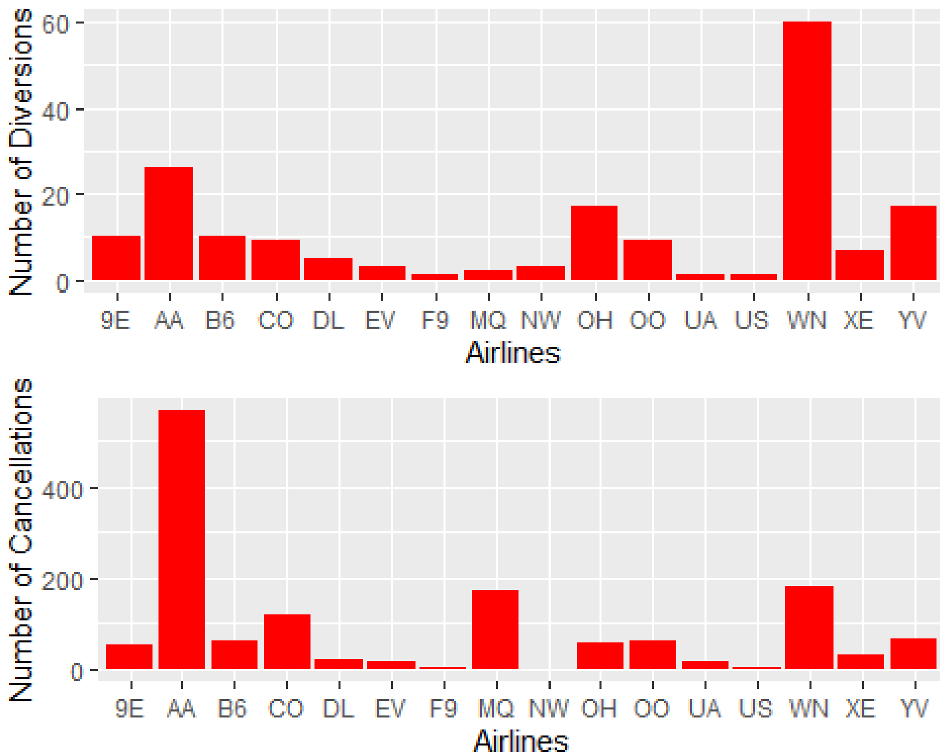
```r
library(ggplot2)


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.

```r
library(ggplot2)

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

```
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)
```

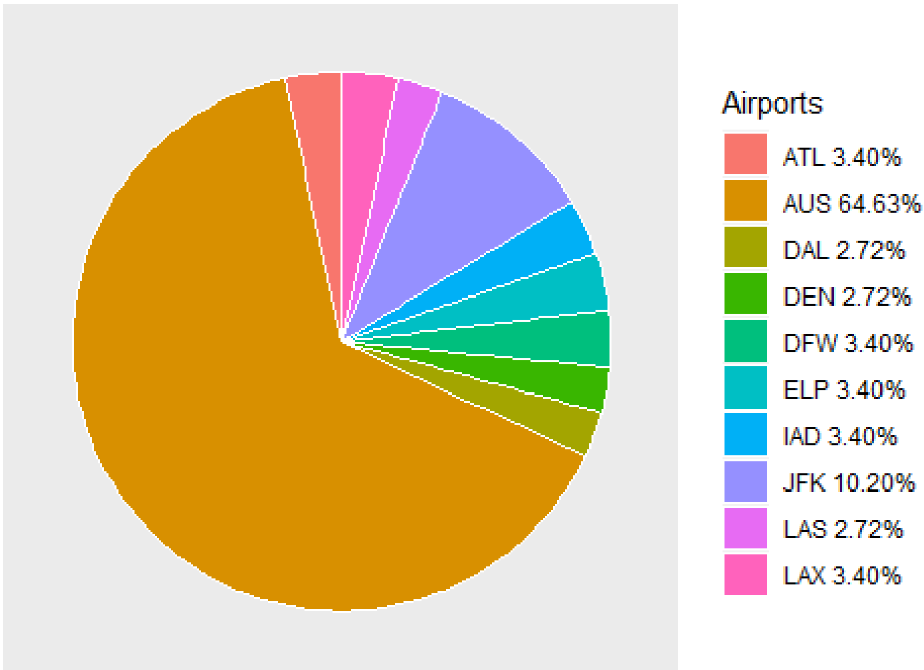


Distribution of cancellations and diversions by Airline. WN (Southwest) has highest number of cancellations going to and from Austin Airport. While, AA(American Airlines) has the highest number of diversions.

```
library(dplyr)
library(ggplot2)

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

colnames(airportcanceled) = c("Airport", "Cancellations")
```

```
airportcanceled <-airportcanceled[airportcanceled$Cancellations != 0,]
top10 = head(sort(airportcanceled$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")
```

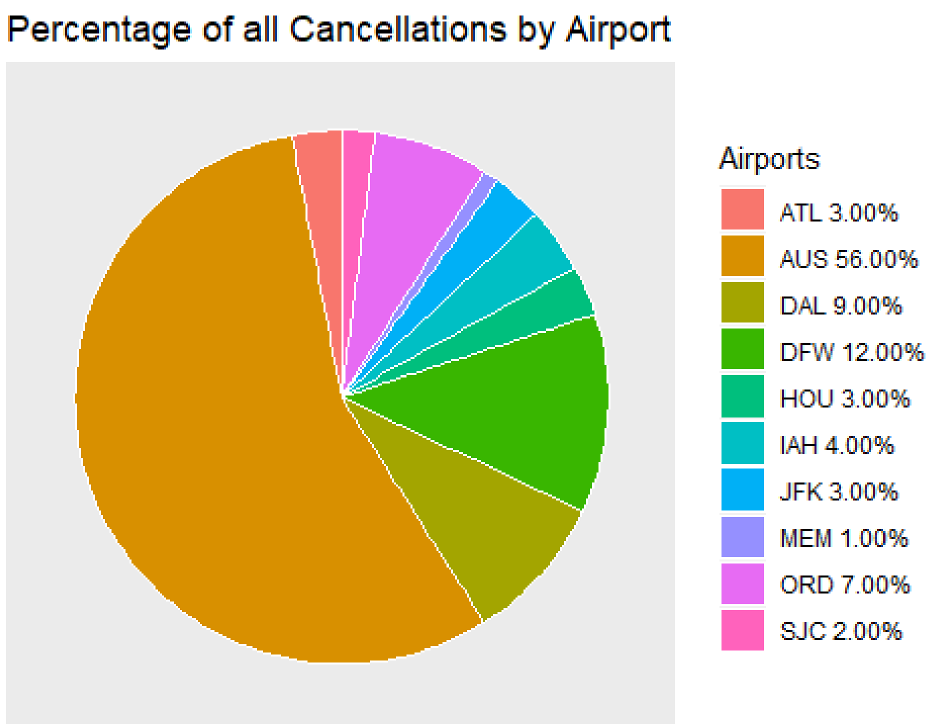
Percentage of all Diversions by Airport



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

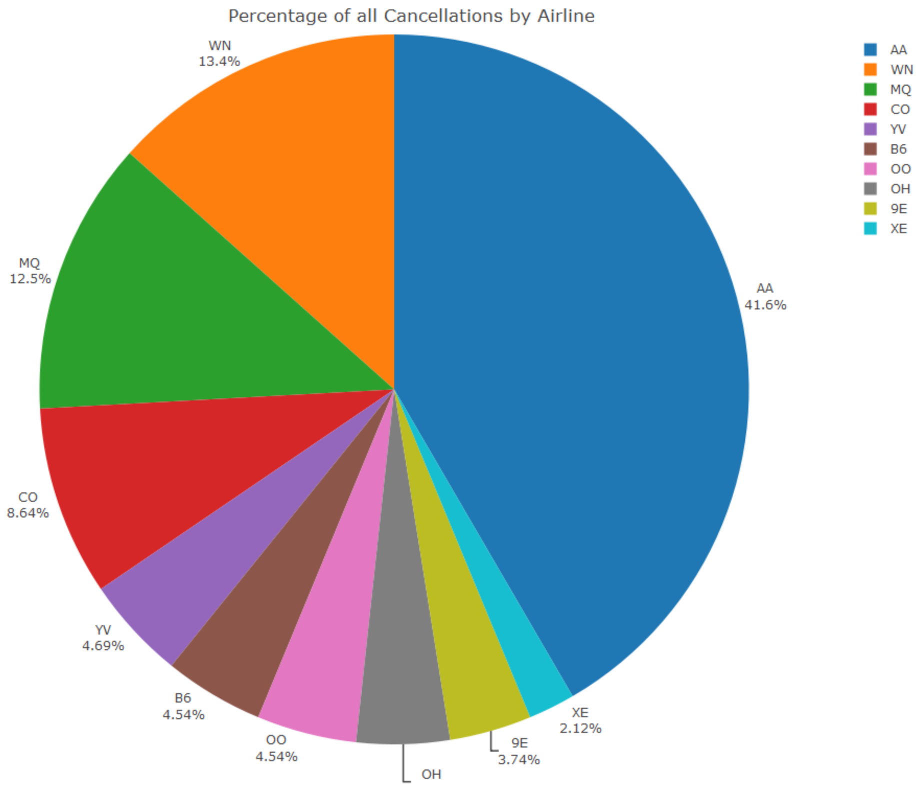
```
library(dplyr)
library(ggplot2)

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])
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.

```
airlinescanceled3<- aggregate(x=ABIA$Cancelled,
by=list(Airport=ABIA$UniqueCarrier), FUN=sum)
airlinesdiverted3<- aggregate(x=ABIA$Diverted,
by=list(Airport=ABIA$UniqueCarrier), FUN=sum)
```

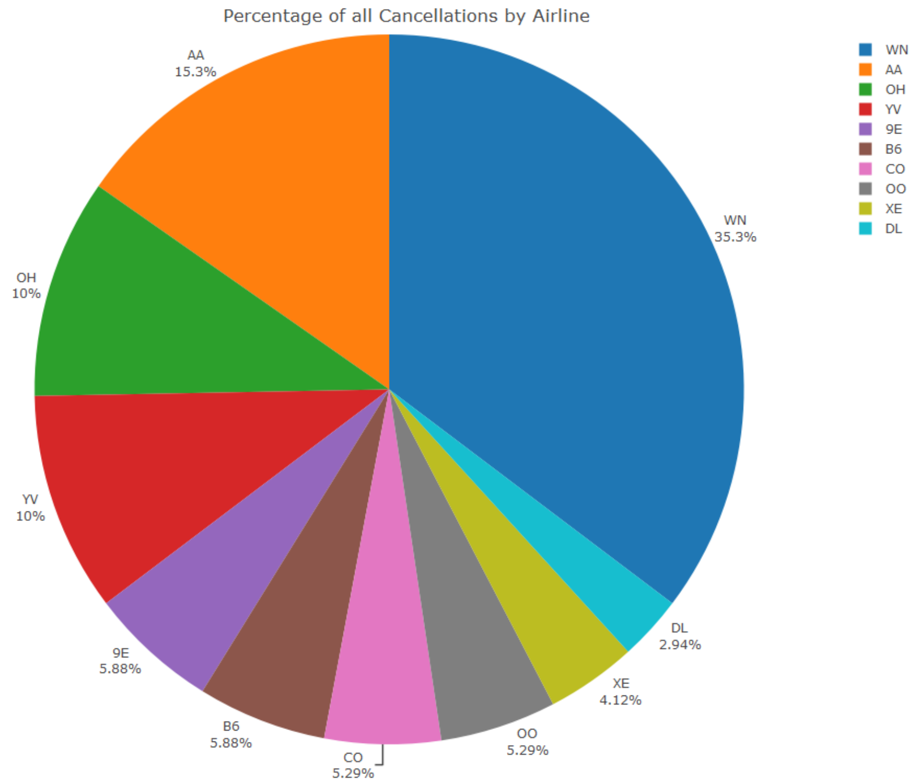
```
colnames(airlinescanceled3) = c("Airlines", "Cancellations")
colnames(airlinesdiverted3) = c("Airlines", "Diversions")

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 /
```

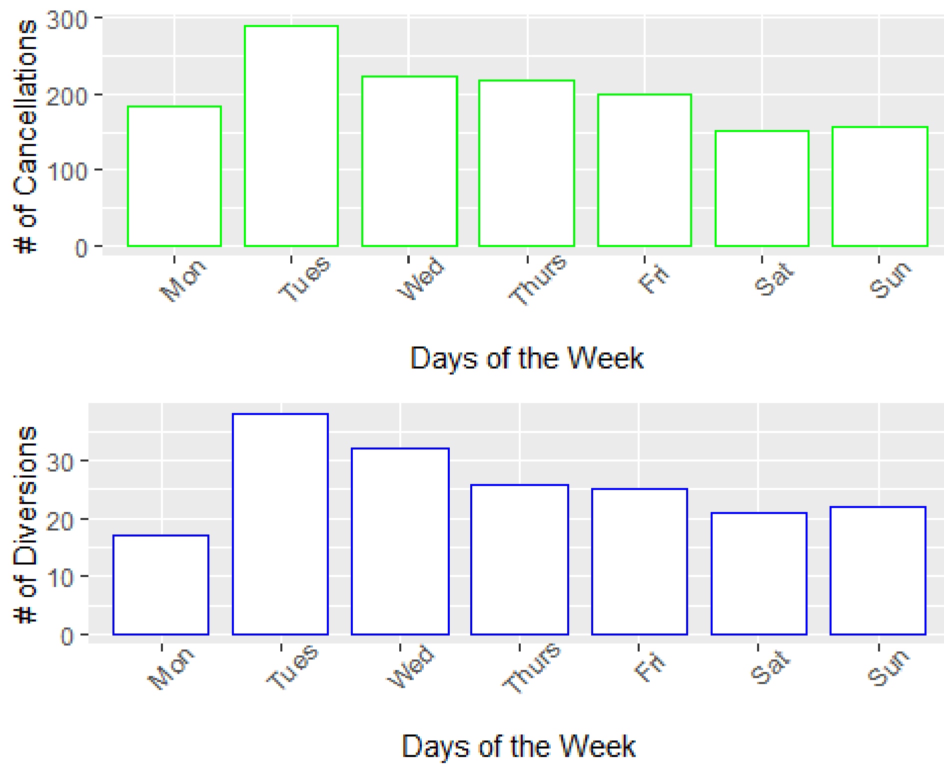
```r
sum(Cancellations))

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))
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') +
```

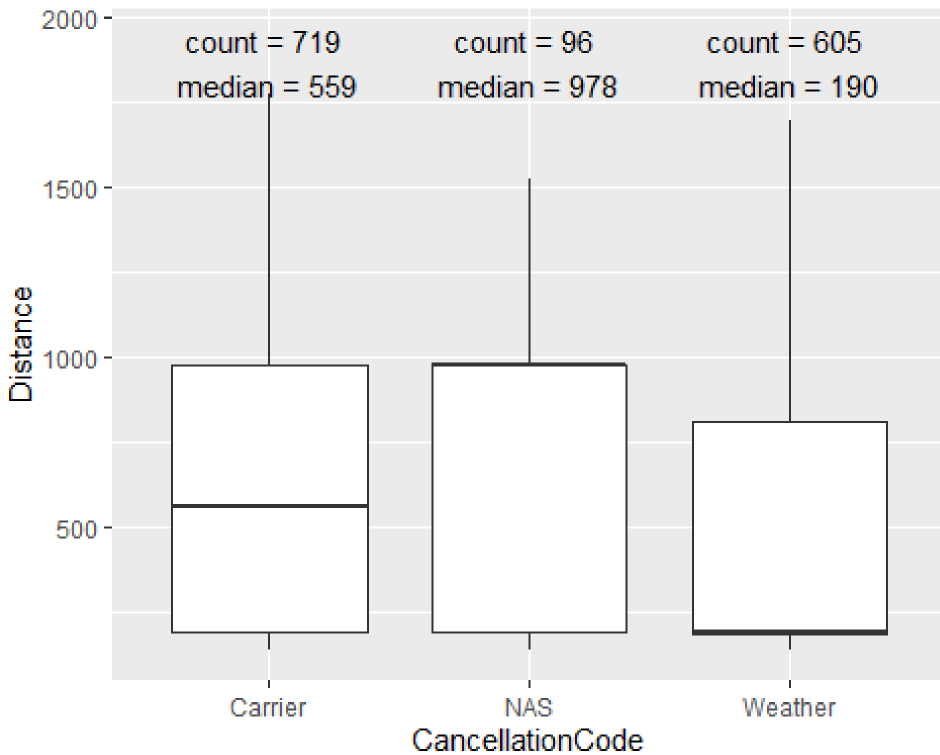
```
  xlab("Days of the Week") + ylab("# of Diversions") + theme(axis.text.x =
element_text(size=10, angle=45))
gridExtra::grid.arrange(weekcanceled1,weekdiverted1)
```



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))


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')
    )
```

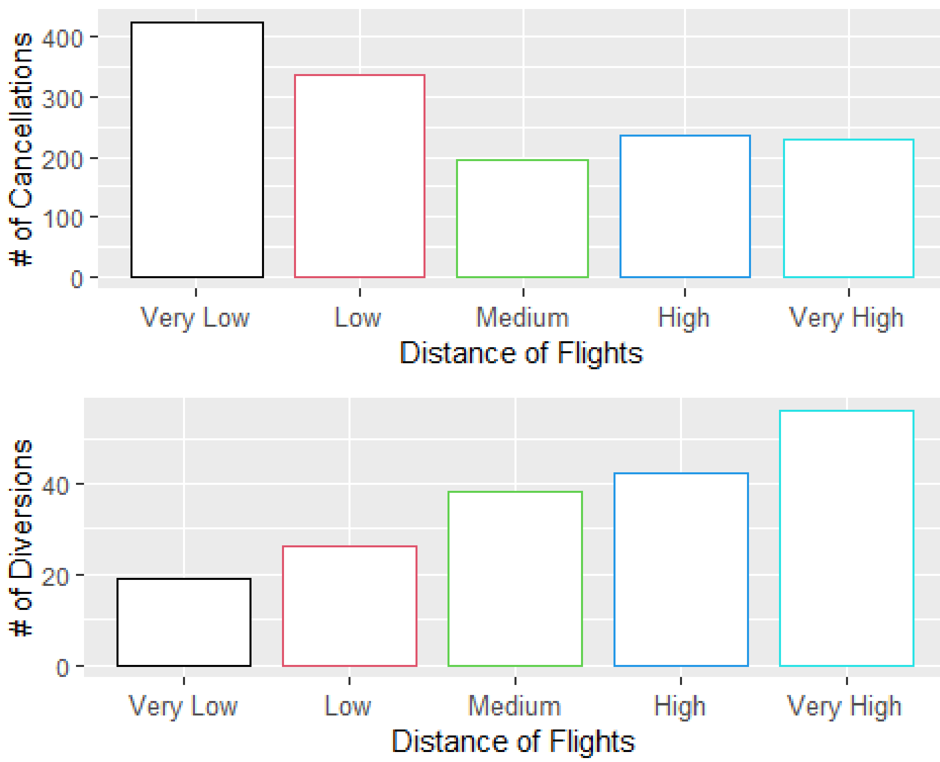
```
  )
}
ggplot(data=ABIA_sub, aes(x=CancellationCode, y=Distance)) +
  geom_boxplot() +
  stat_summary(
    fun.data = stat_box_data,
    geom = "text",
    hjust = 0.5,
    vjust = 0.9
  )
```



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")
```

```r
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.

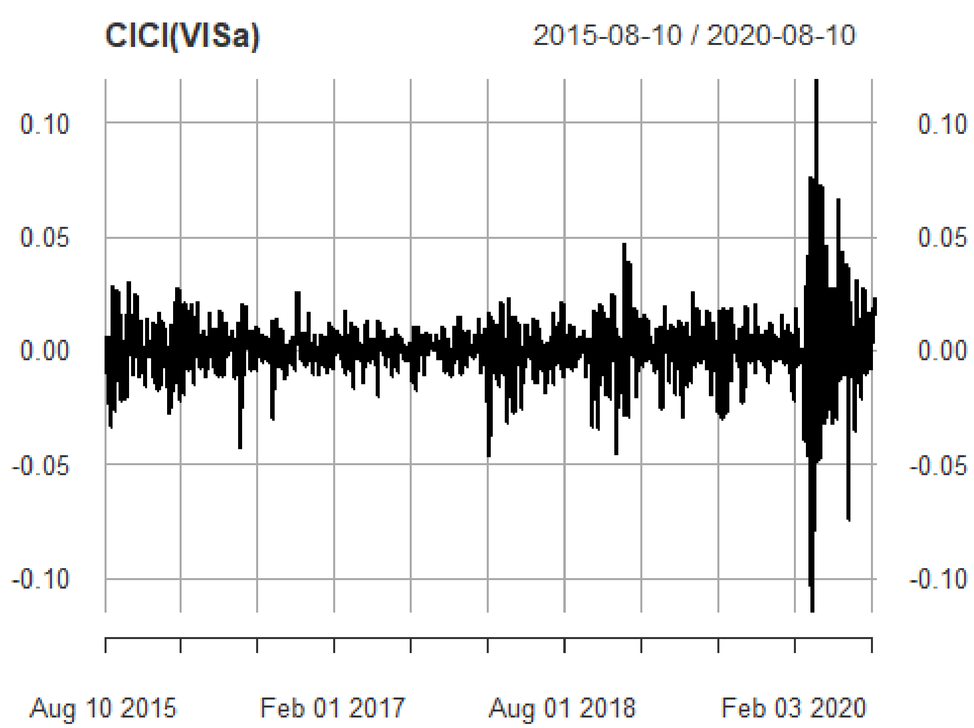
In my third portfolio, the underlying goal was to pick ETFs that are a bit more volatile with the goal of making more money, but with more risk overall. The ETF that were chosen are the following: SOXL(Direxion Daily Semiconductor Bull 3X Shares), TQQQ(ROSHARES TR/ULTRAPRO QQQ), ROM(ProShares Ultra Technology), 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
```

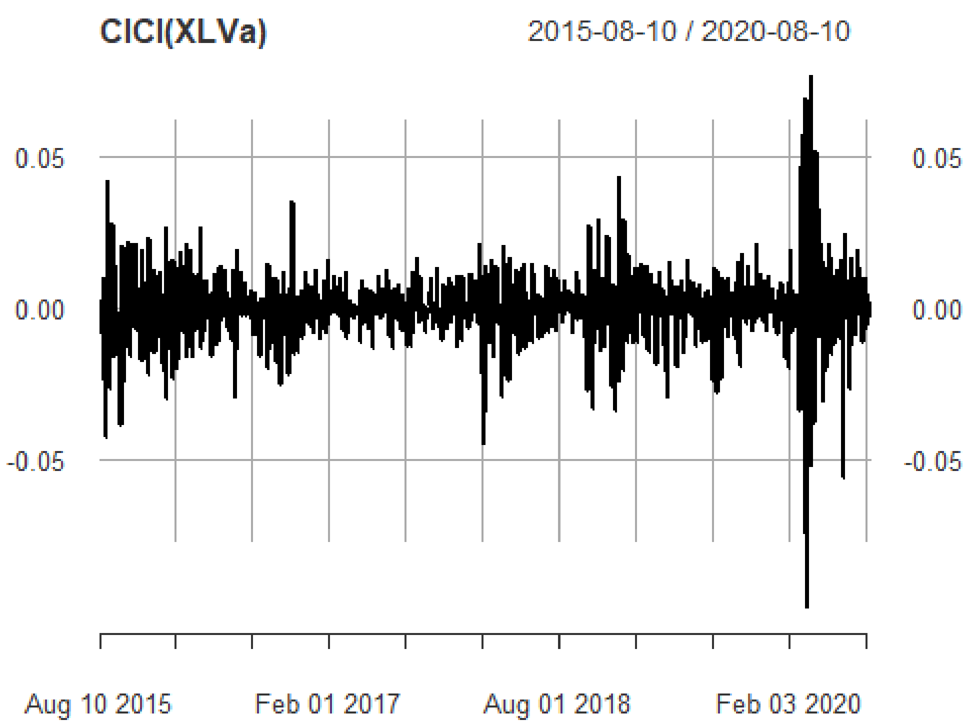
```r
mystocks = c("VIS", "XLV", "IXC","XNTK","VWO")
getSymbols(mystocks)
```

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

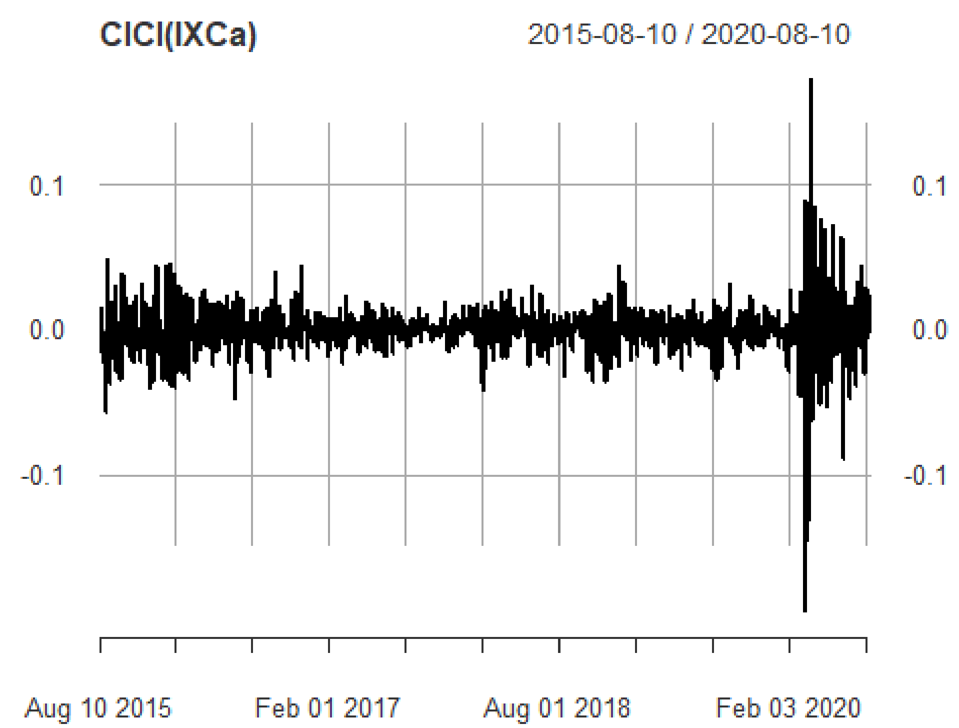
```r
VIS <- VIS["2015-08-10::2020-08-10"]
XLV <- XLV["2015-08-10::2020-08-10"]
IXC <- IXC["2015-08-10::2020-08-10"]
XNTK <- XNTK["2015-08-10::2020-08-10"]
VWO <- VWO["2015-08-10::2020-08-10"]

# Adjust for splits and dividends
VISa = adjustOHLC(VIS)
XLVa = adjustOHLC(XLV)
IXCa = adjustOHLC(IXC)
XNTKa = adjustOHLC(XNTK)
VWOa = adjustOHLC(VWO)


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



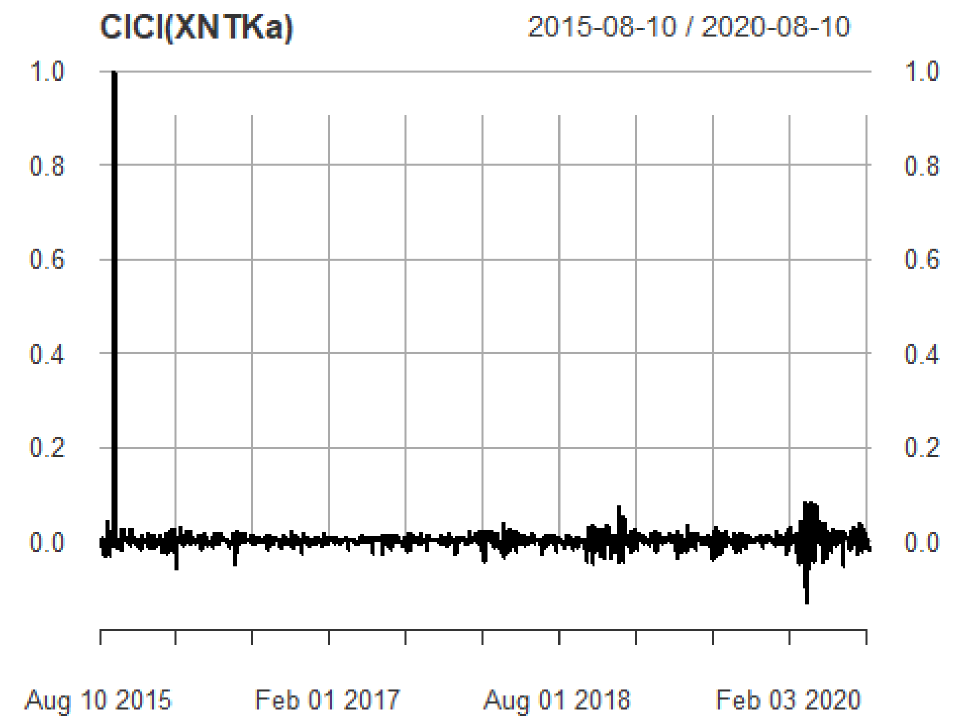

```r
plot(ClCl(XLVa))
```



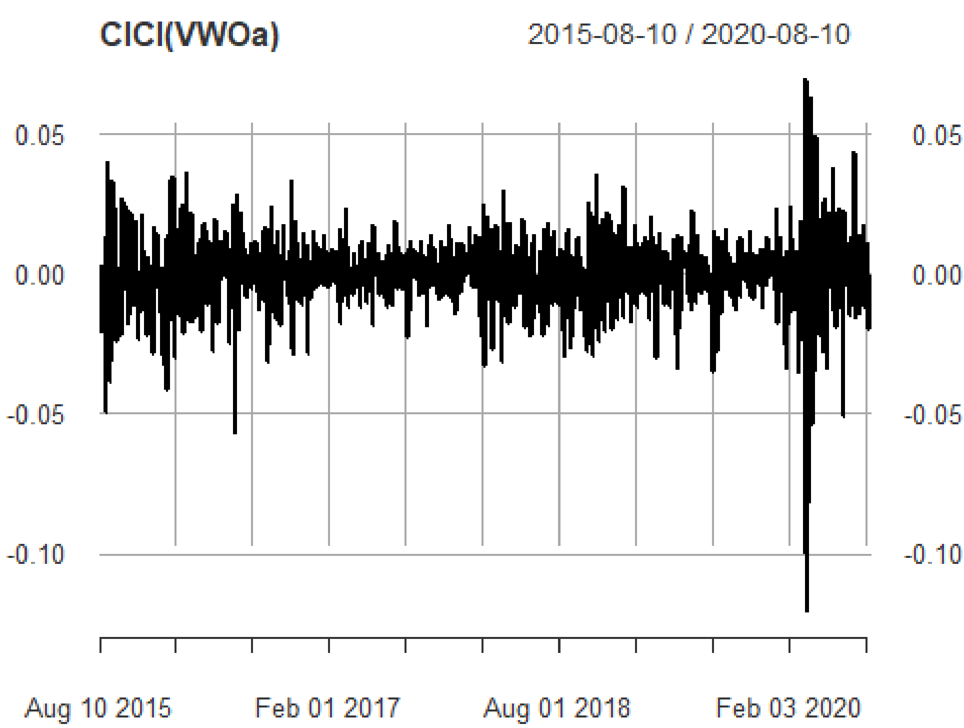

```
plot(ClCl(IXCa))
```



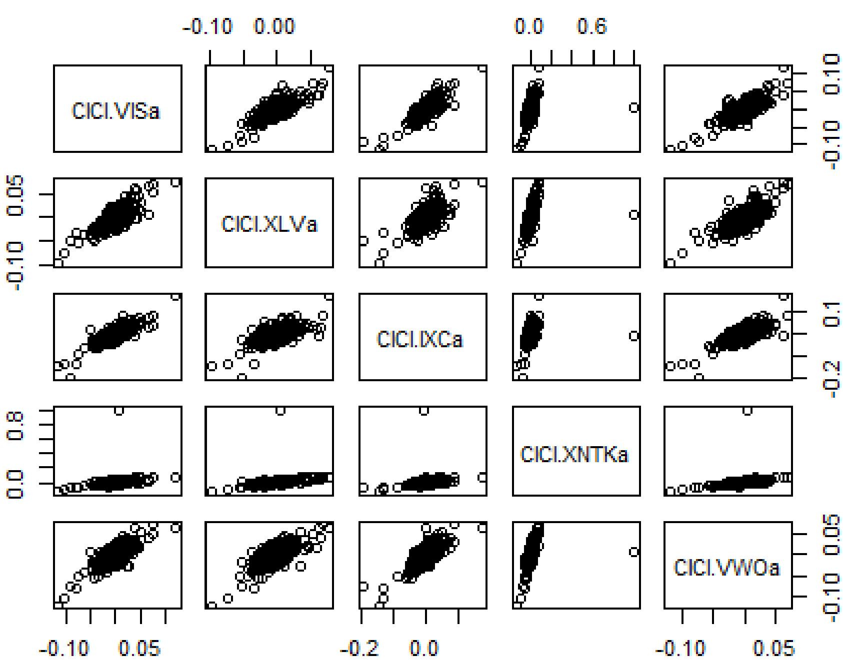

```
plot(ClCl(XNTKa))
```



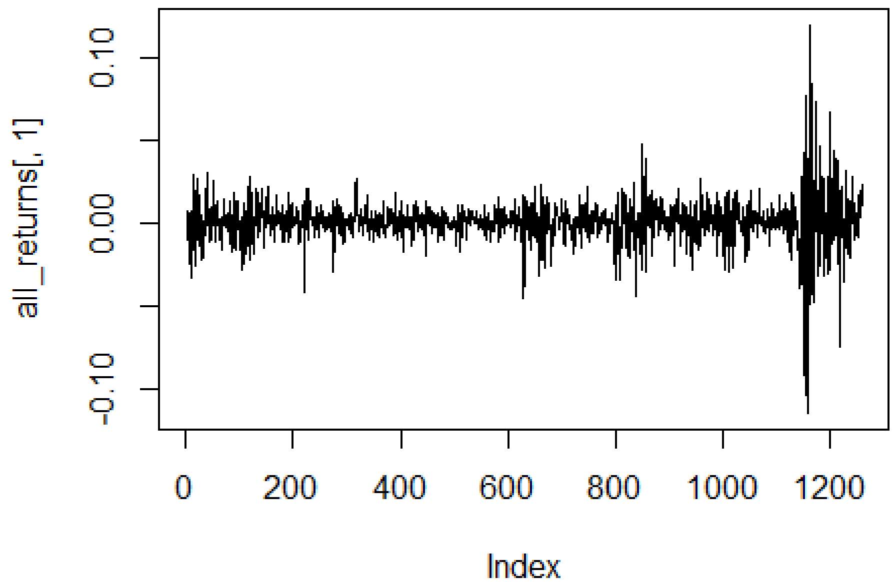

```
plot(ClCl(VWOa))
```



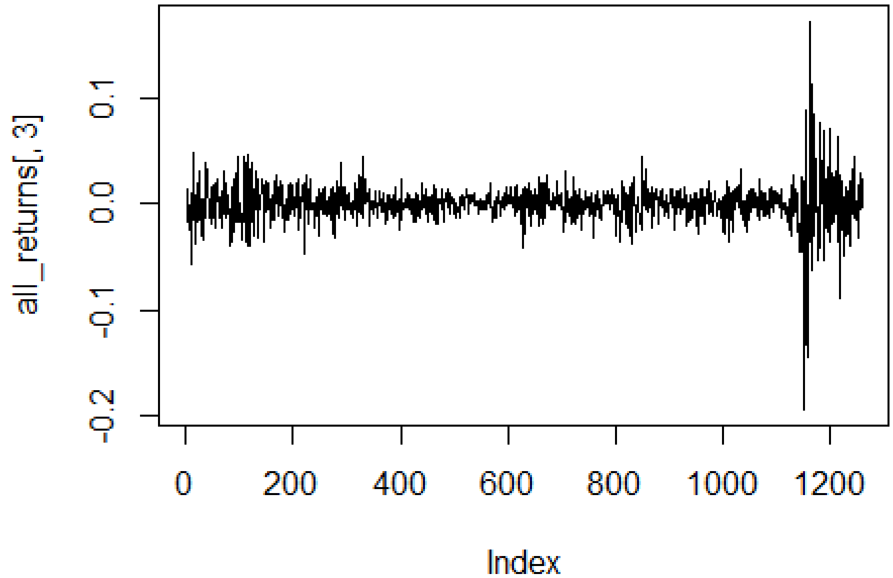
```r
# Combine close to close changes in a single matrix
all_returns = cbind(ClCl(VISa),ClCl(XLVa),ClCl(IXCa),ClCl(XNTKa),ClCl(VWOa))
head(all_returns)

##                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)
```



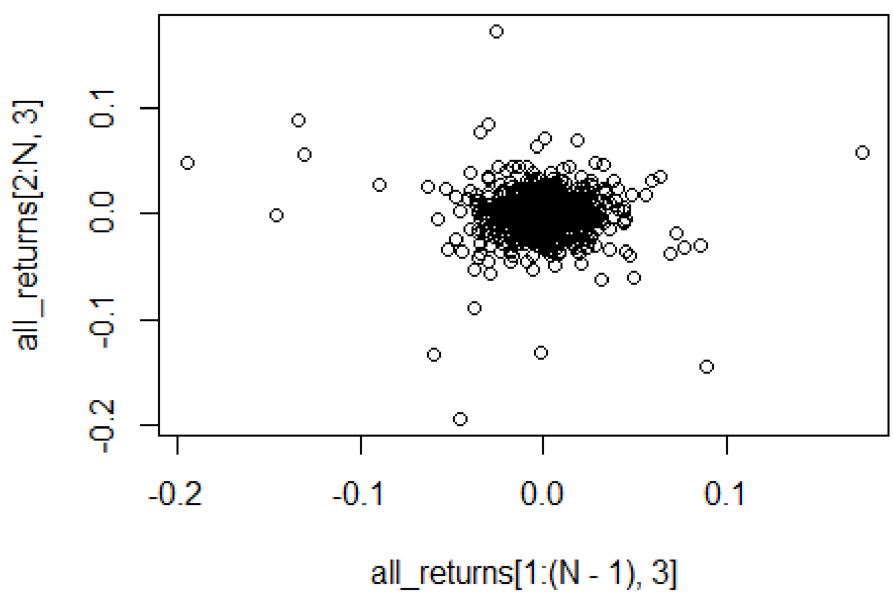
```r
plot(all_returns[,1], type='l')
```



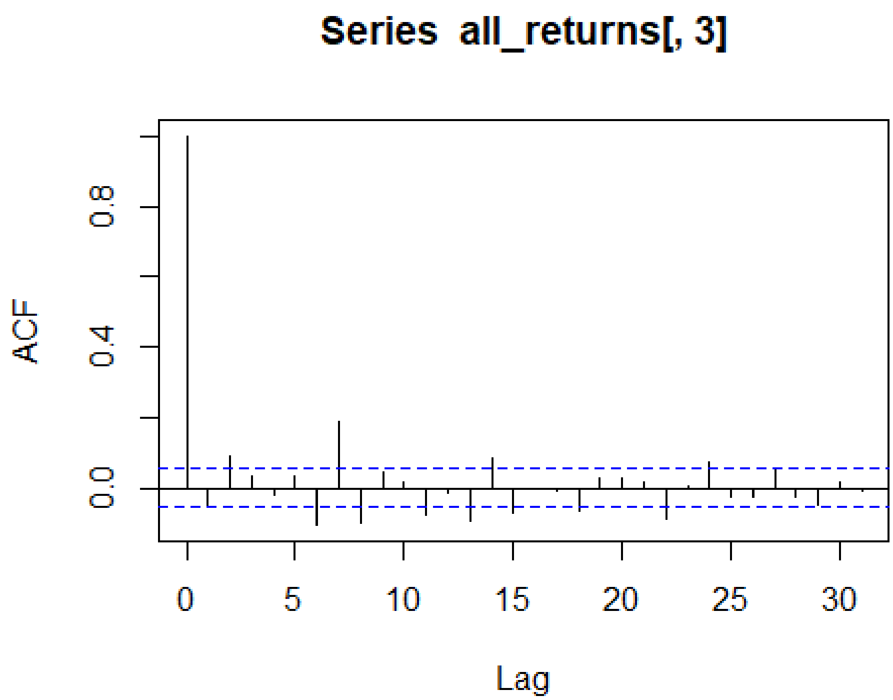

```r
# Look at the market returns over time
plot(all_returns[,3], type='l')
```




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

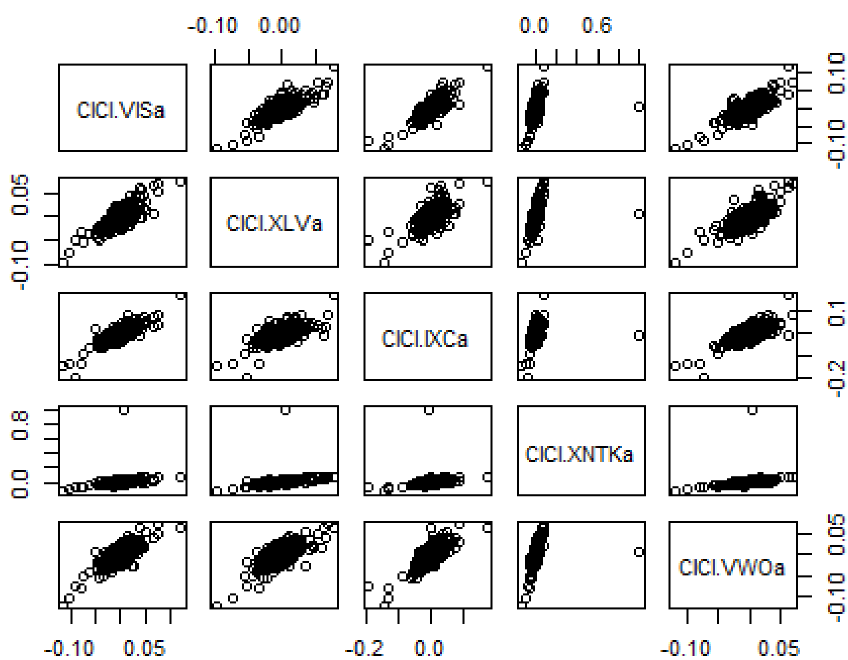


```
# An autocorrelation plot: nothing there
acf(all_returns[,3])
```



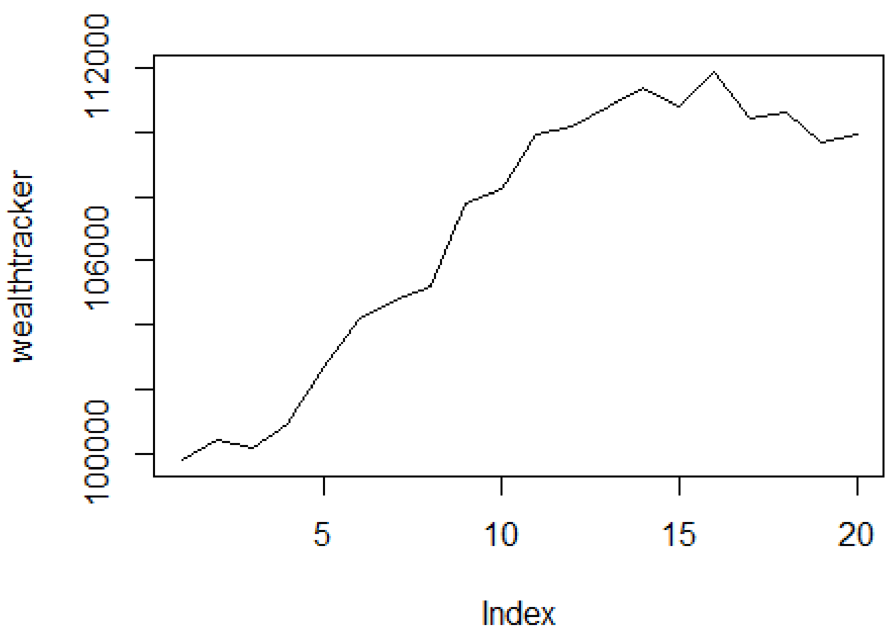

```
# conclusion: returns uncorrelated from one day to the next
# (makes sense, otherwise it'd be an easy inefficiency to exploit,
# and market inefficiencies that are exploited tend to disappear as a result)

#### 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(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] 99966.59

# 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] 109905

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



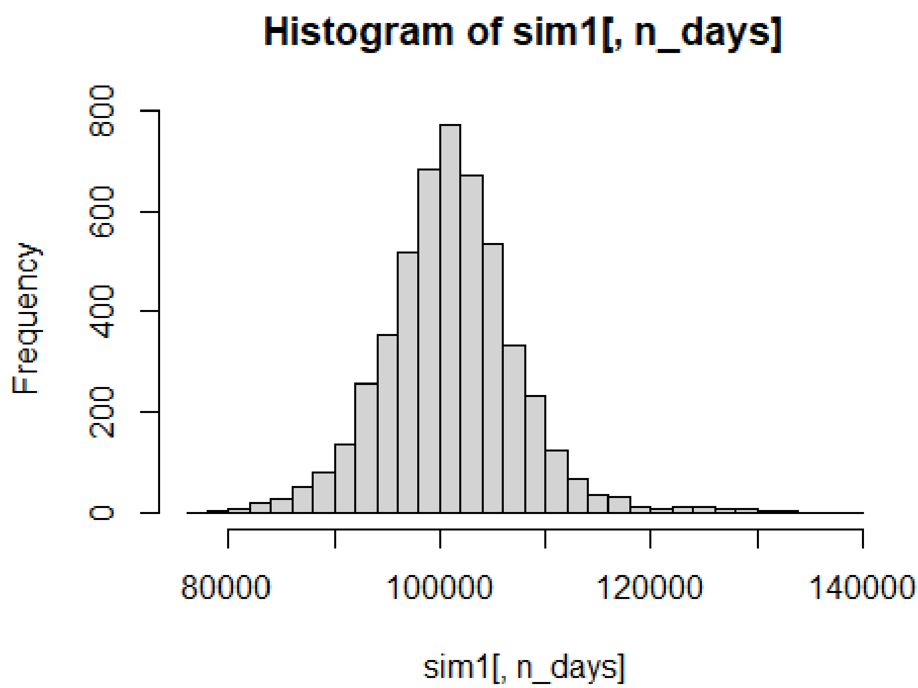

```r
## end block
```

```
# 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  98109.31  98474.83  98909.83  99511.03  97959.32  95412.58
96448.30
## result.2  99039.16  99812.27  98871.82  99484.43 100930.15 100597.65
100790.26
## result.3 100006.17  99397.43 100371.72 100686.04 100533.57 100228.20
99648.89
## result.4 101389.83 101773.20 101858.23 102211.41 102704.20 101017.97
101443.72
## result.5  99528.09 100093.67  99718.28  99503.02 100570.00 101640.08
102500.60
## result.6 100028.11 100867.67 100339.22  99442.19  99681.34  99918.28
100564.31
##               [,8]      [,9]     [,10]     [,11]     [,12]     [,13]
[,14]
## result.1  95951.12  96683.17  97774.24  95914.46  95475.87  95323.47
94729.70
## result.2 100720.31 101361.48 101404.42  99205.44  98980.12  99802.66
99931.89
## result.3  97243.40  98417.93  98202.64  99472.06  99979.37 102247.42
102017.03
## result.4 100531.39  99168.86 119195.42 119738.07 118376.14 114879.38
114973.17
## result.5 103391.93 100631.01 100507.12  99258.36  98880.23 100119.03
100568.31
## result.6 101081.81 103534.41 104208.05 100524.15  99868.51  99904.84
99679.70
```

```
##              [,15]     [,16]     [,17]     [,18]     [,19]     [,20]
## result.1  95053.18  96356.83  97241.93  96603.74  98439.69  98151.01
## result.2 100994.41 101273.88 101752.77 102289.51 103082.40 103516.03
## result.3 101832.65 102810.45  92257.61  92839.05  93955.27  96097.57
## result.4 117430.70 117507.52 119018.56 119127.34 120686.75 121269.33
## result.5 100375.61 100798.35 100094.01 100195.85 100173.07 100201.04
## result.6 100122.35 100687.87 100846.89 101114.09 100572.52 100070.28
```

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



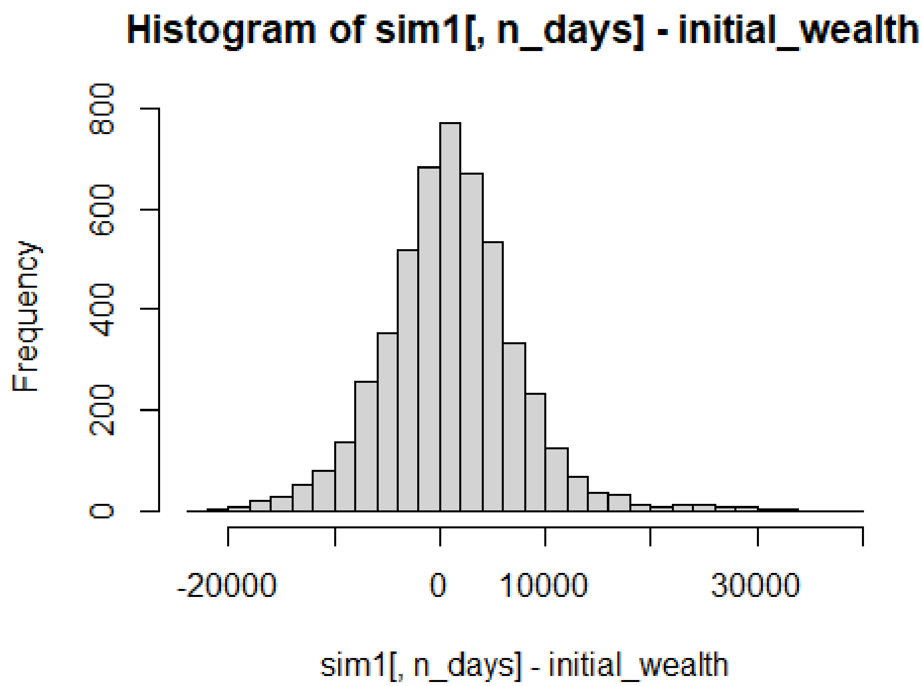

```r
# Profit/loss
mean(sim1[,n_days])
```

```
## [1] 101028.4
```

```r
mean(sim1[,n_days] - initial_wealth)
```

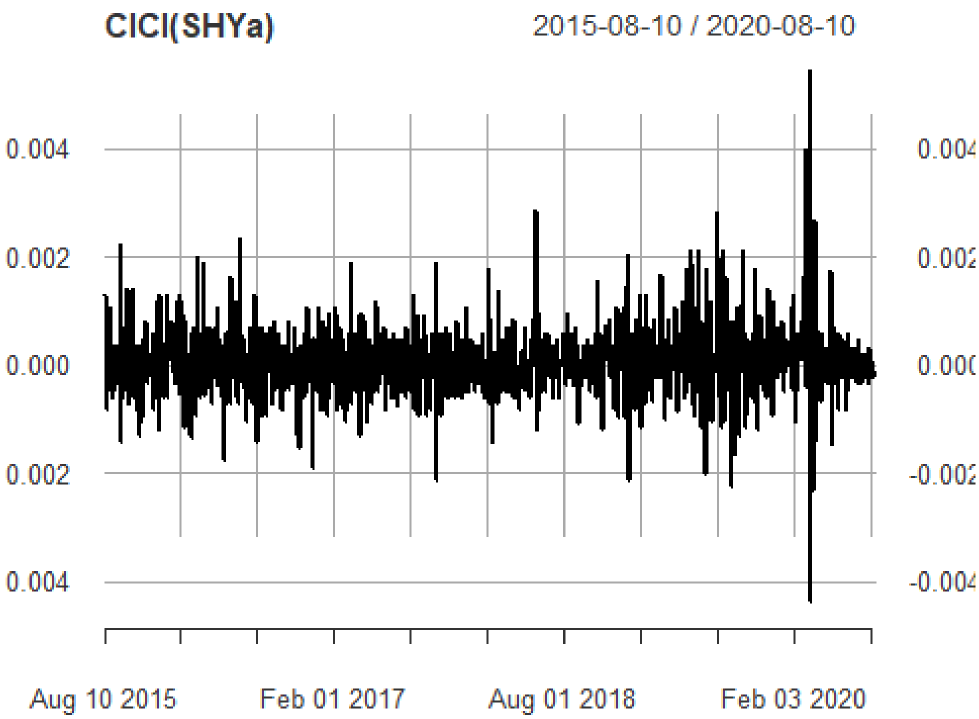
```
## [1] 1028.39
```

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

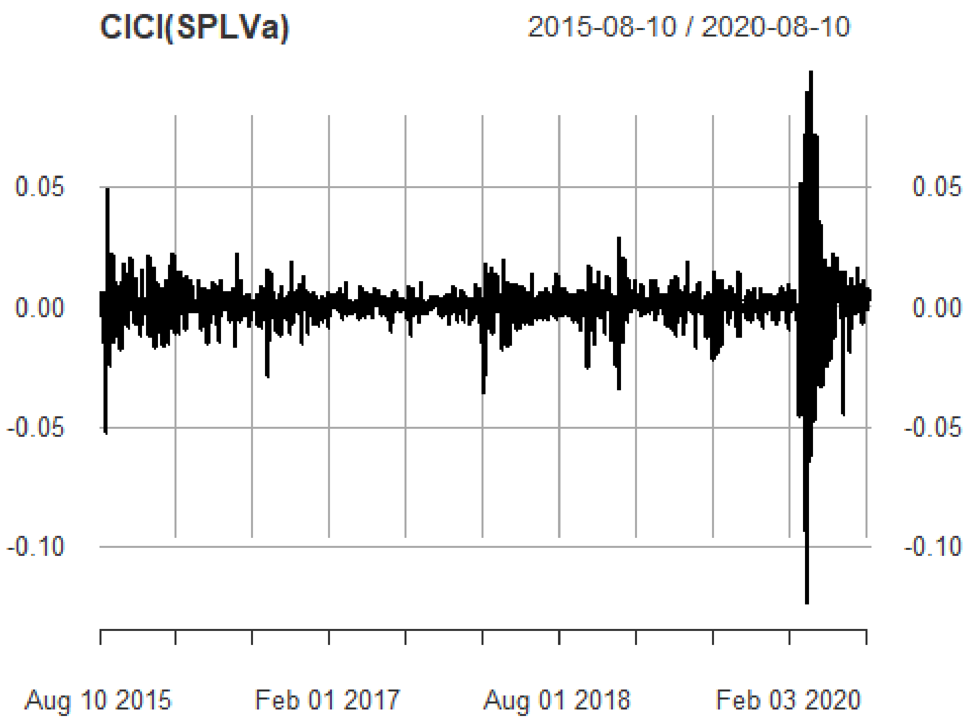



```r
# 5% value at risk:
quantile(sim1[,n_days]- initial_wealth, prob=0.05)

##        5%
## -8888.738

# 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)
```

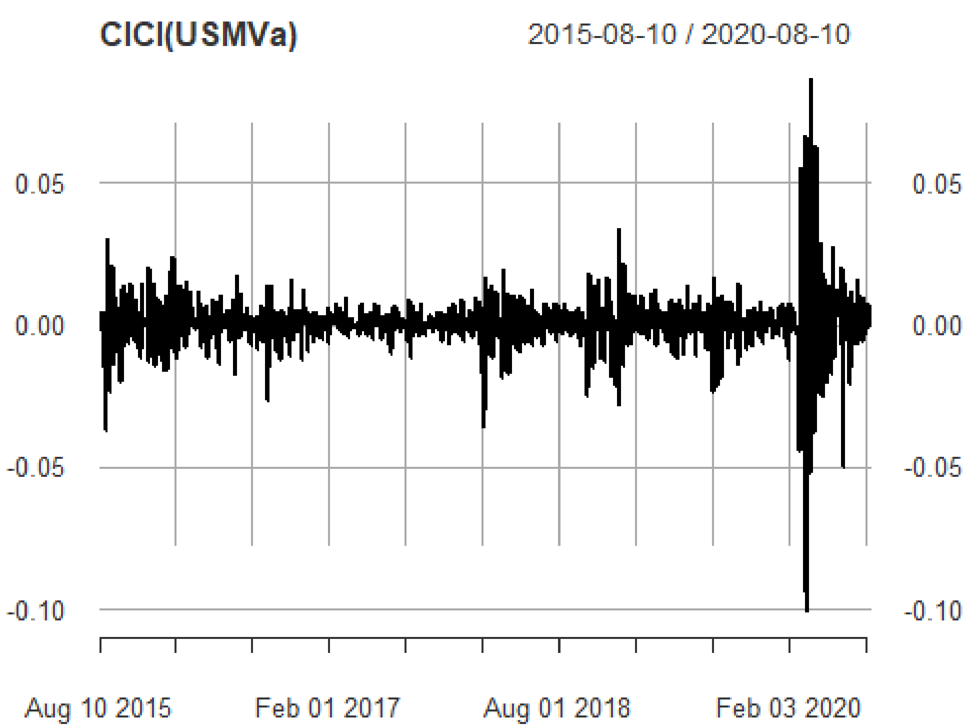
```
EFAVa = adjustOHLC(EFAV)

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



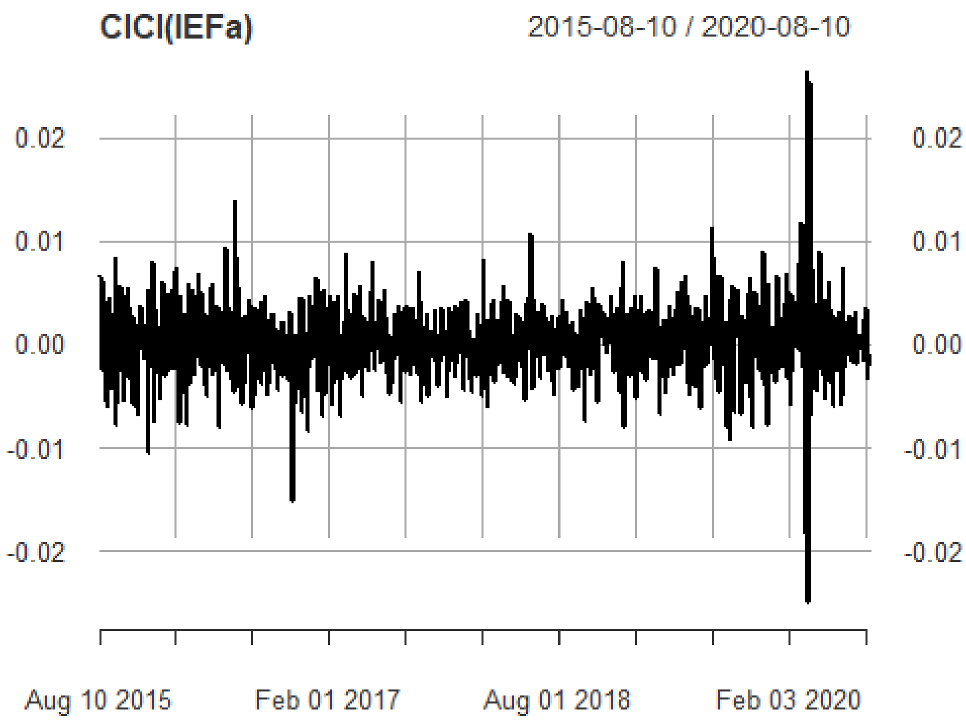

```
plot(ClCl(SPLVa))
```



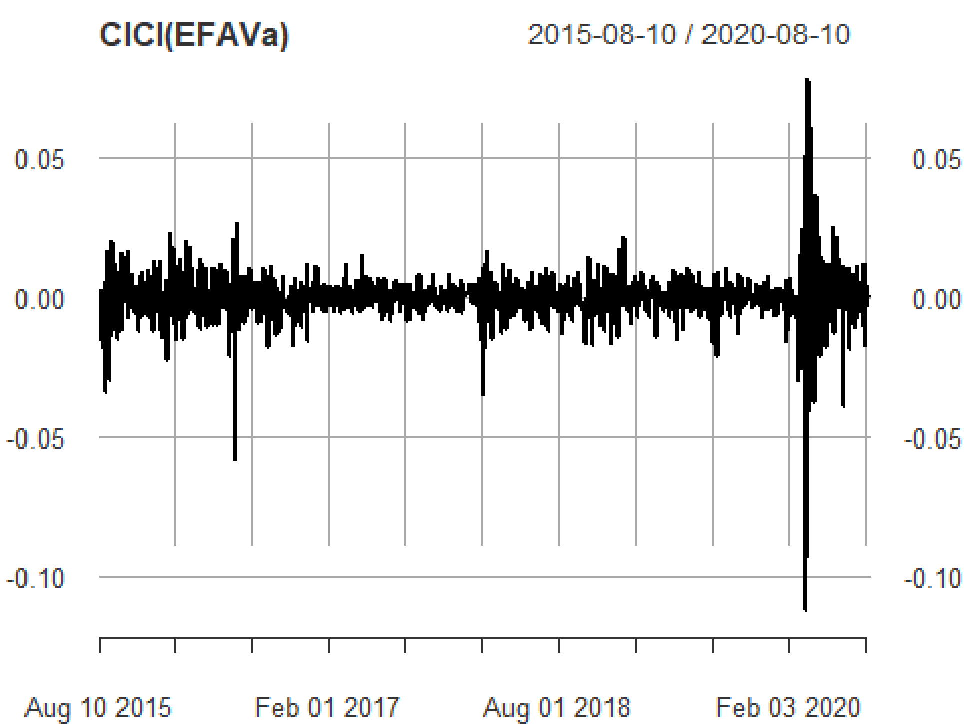

```
plot(ClCl(USMVa))
```



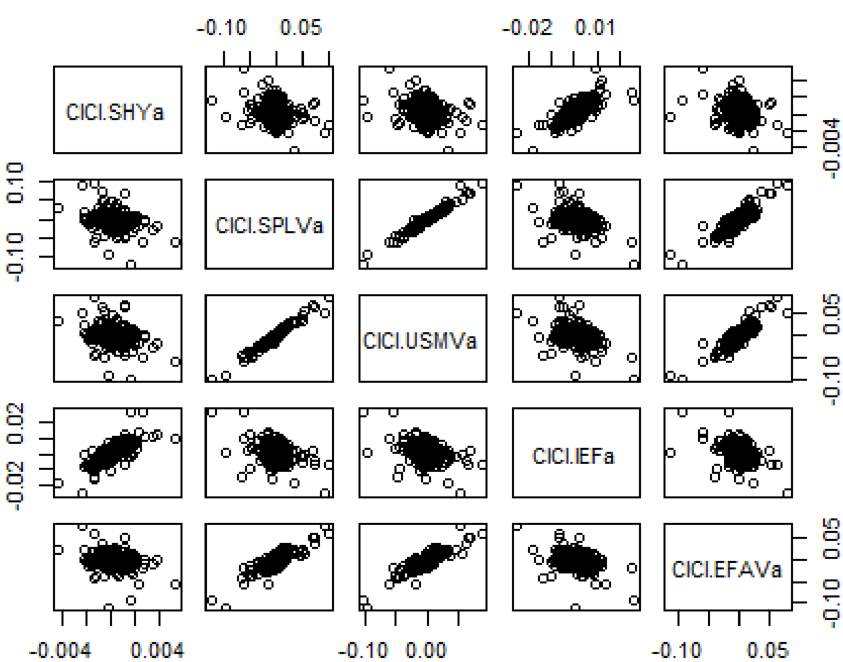

```
plot(ClCl(IEFa))
```



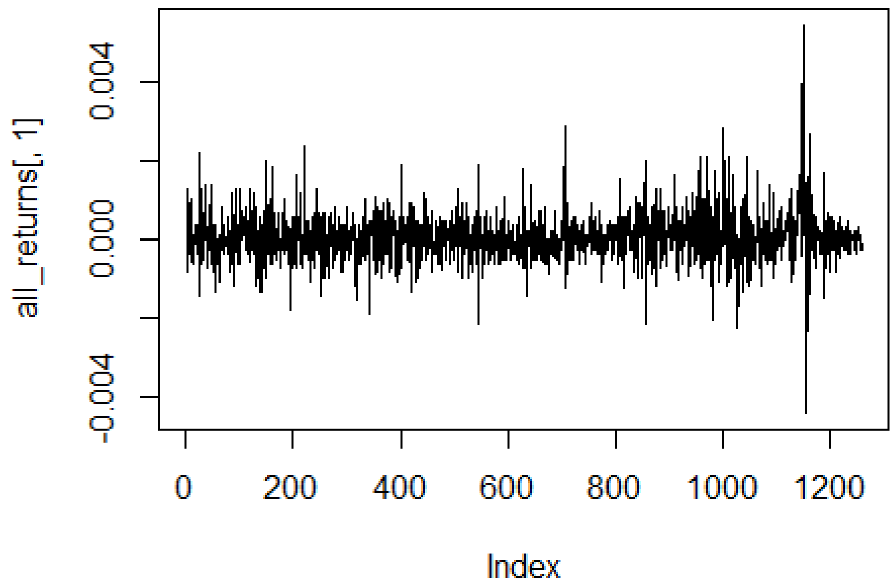

```r
plot(ClCl(EFAVa))
```



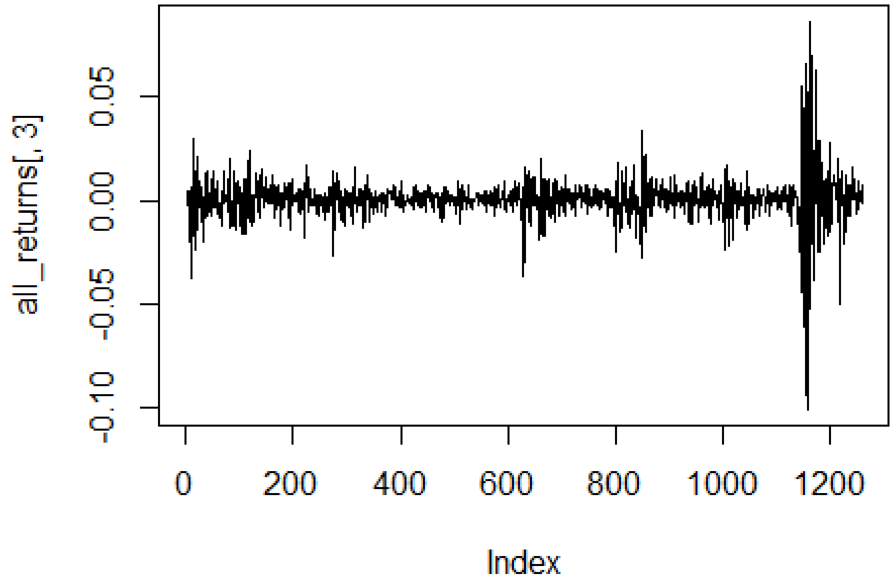
```r
# Combine close to close changes in a single matrix
all_returns =
cbind(ClCl(SHYa),ClCl(SPLVa),ClCl(USMVa),ClCl(IEFa),ClCl(EFAVa))
head(all_returns)

##                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)
```



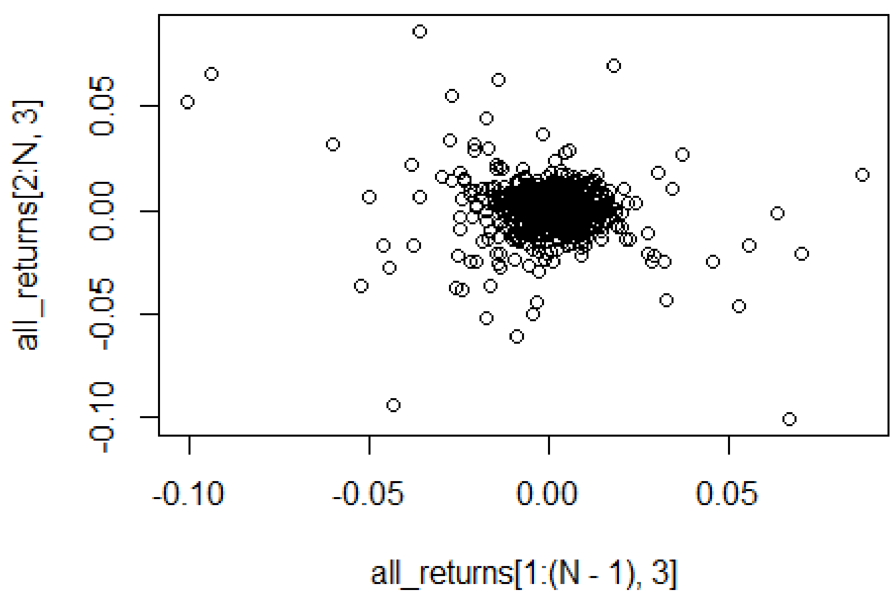
```r
plot(all_returns[,1], type='l')
```



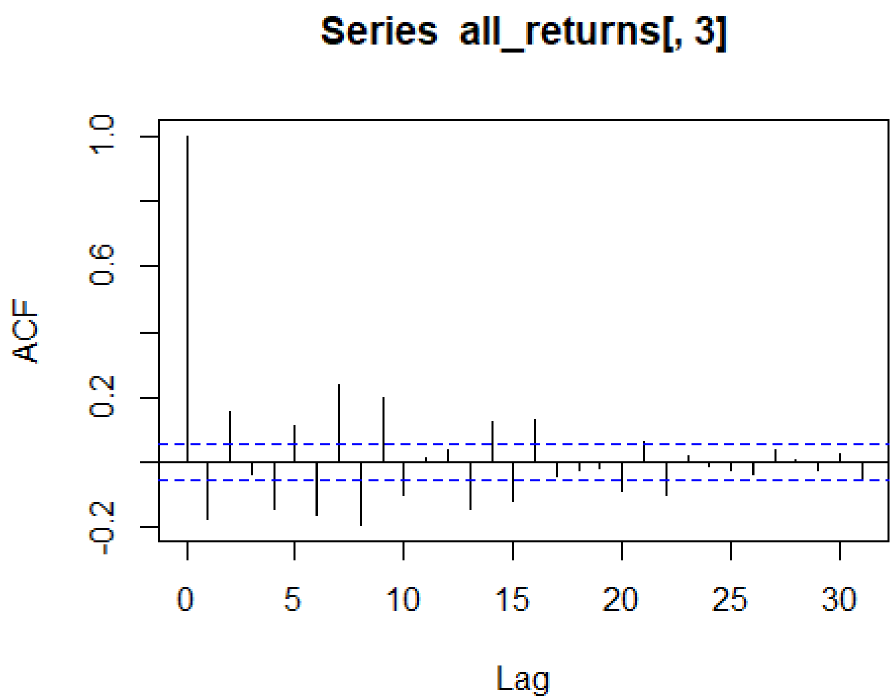

```r
# Look at the market returns over time
plot(all_returns[,3], type='l')
```




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

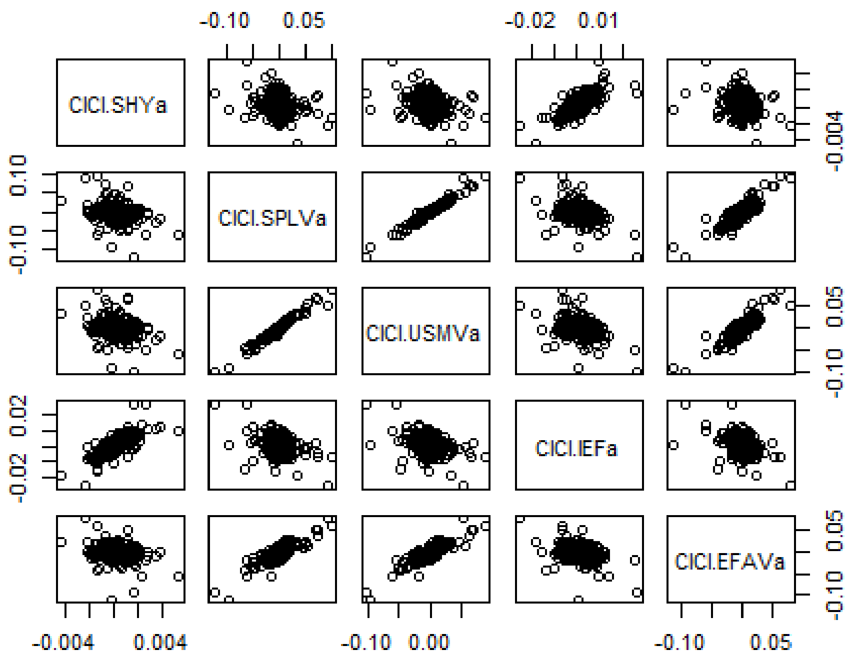


```r
# An autocorrelation plot: nothing there
acf(all_returns[,3])
```



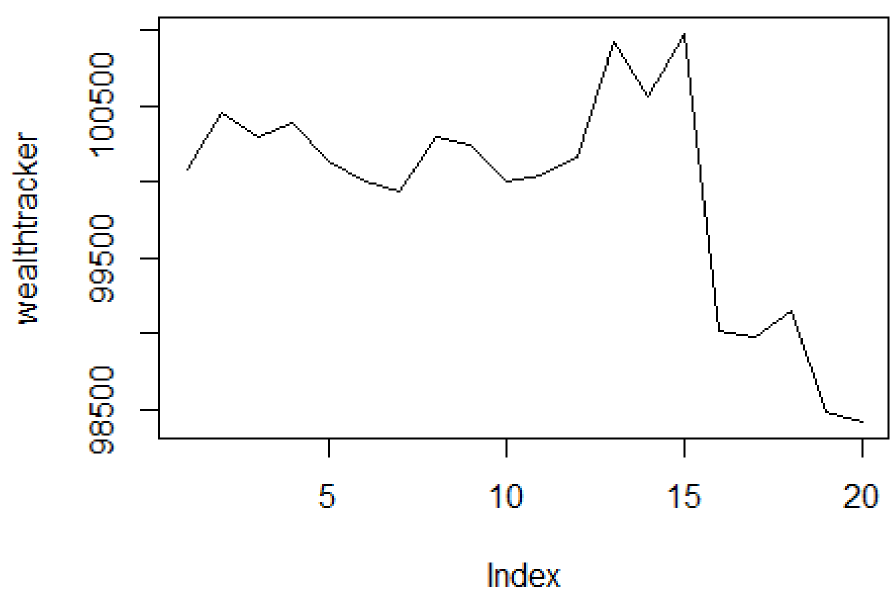

```r
# conclusion: returns uncorrelated from one day to the next
# (makes sense, otherwise it'd be an easy inefficiency to exploit,
# and market inefficiencies that are exploited tend to disappear as a result)

#### 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(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
```

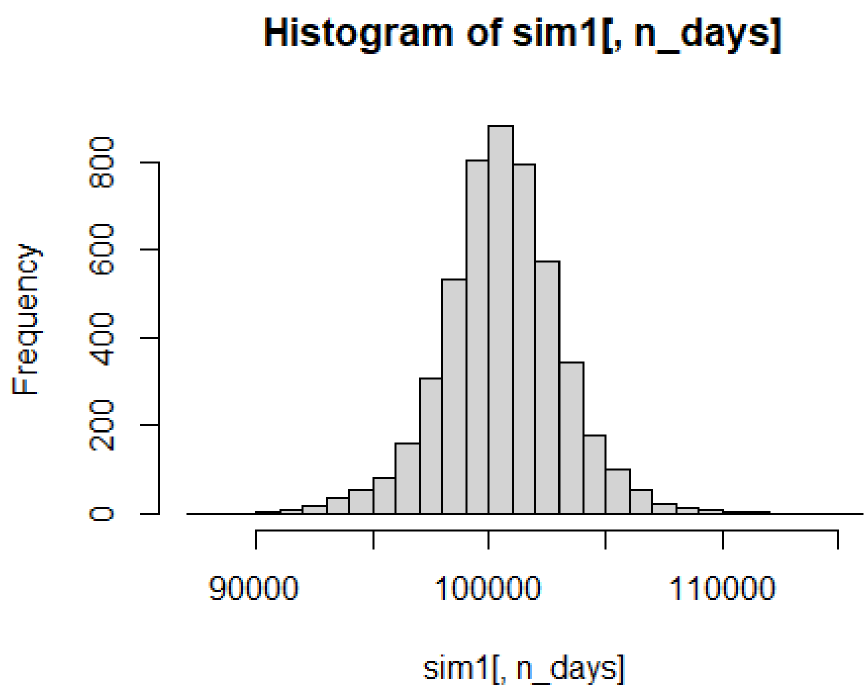
```
holdings = holdings*(1 + return.today)

holdings

##            ClCl.SHYa ClCl.SPLVa ClCl.USMVa ClCl.IEFa ClCl.EFAVa
## 2015-11-19  19995.27   20057.08   20004.78  20043.45   20137.24

# 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] 98419.47

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



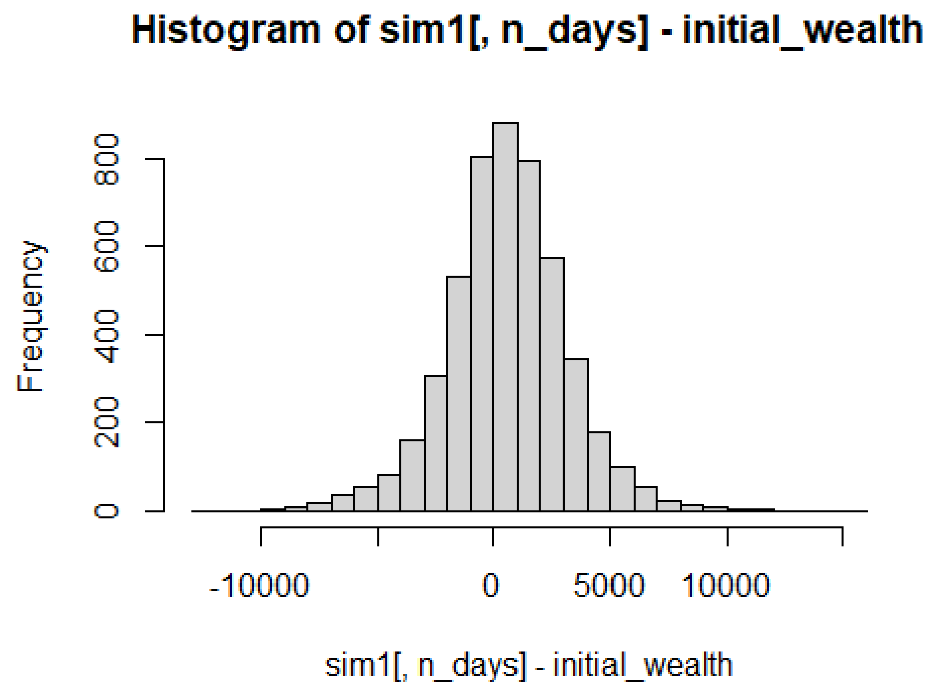
```
## end block

# 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 100030.81 100456.24 100632.86 100669.19 100455.42 100578.83
101398.91
## result.2  99127.52  99188.83  98897.97  98888.65  99238.72  99164.55
99442.87
## result.3  99611.96  99929.97 100272.07 101700.49 101600.53 101601.37
101641.74
## result.4  99946.00 100789.21 100990.69  98515.90  98234.65  98053.73
97984.52
## result.5 100147.41 100236.07 100413.96  99212.02  98490.58  98310.75
98560.58
## result.6 100296.63 100747.59 100416.60 100276.61  99867.09  99726.98
99953.15
##                [,8]      [,9]     [,10]     [,11]     [,12]     [,13]
[,14]
## result.1 100501.71 100453.74 101276.17 101454.89 101151.73 101039.98
101485.23
## result.2  99601.41  99783.32  99541.53 100279.34 100562.79 100966.70
100929.63
## result.3 101798.30 101885.37 101937.10 101636.89 101942.43 101766.66
103164.66
## result.4  98091.31  98130.16  97964.67  98209.97  98357.73  98269.14
98258.10
## result.5  98672.05  98392.02  98769.93  98358.63  98044.90  97913.08
97820.26
```

```
## result.6  99984.31 100112.97 100139.48  99245.05  99543.00  99226.42
99332.24
##                [,15]     [,16]     [,17]     [,18]     [,19]     [,20]
## result.1 101700.24 101733.14 106586.10 106640.28 106104.05 105932.45
## result.2 100887.15 100871.07 100738.88 100614.24 100245.60 101392.46
## result.3 103122.05 103293.24 103775.98 103648.69 103556.34 103788.79
## result.4  97004.39  97138.61  96867.37  97257.10  97526.37  97569.79
## result.5  97837.95  97991.98  99194.53  98777.98  98368.97  98344.24
## result.6  99230.87  99257.05  98468.07  98738.01  99116.51  98854.42

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

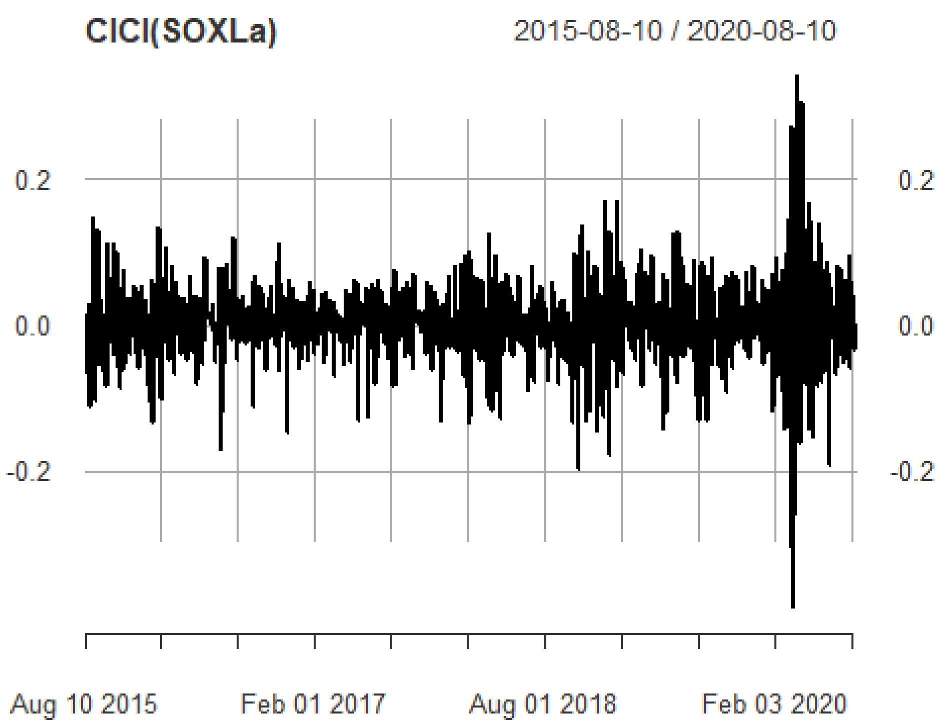



```
# Profit/loss
mean(sim1[,n_days])

## [1] 100561.7

mean(sim1[,n_days] - initial_wealth)

## [1] 561.7033

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

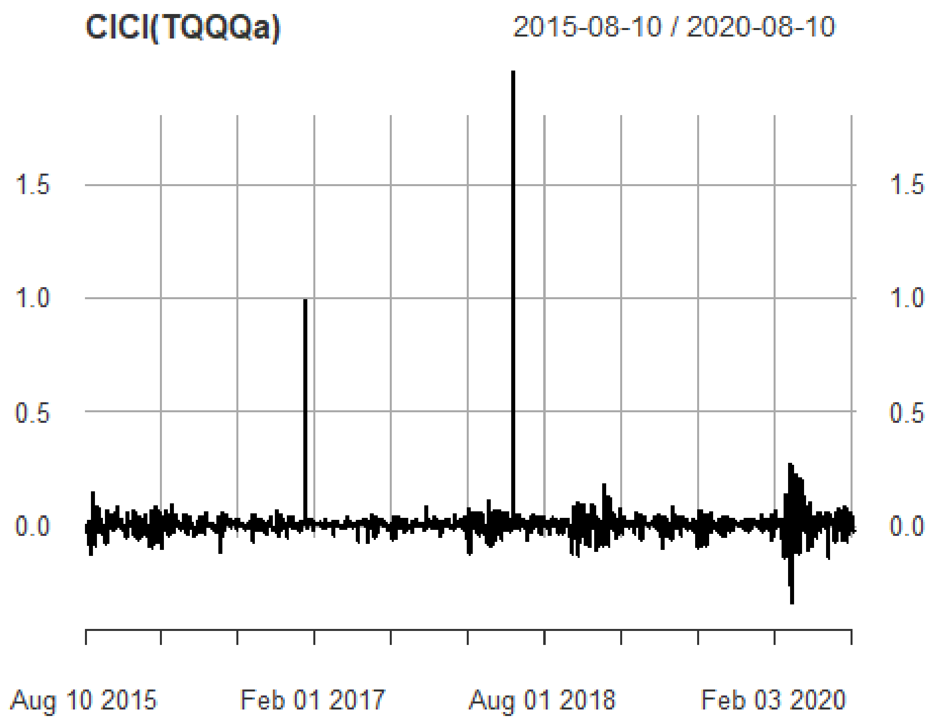



```
# 5% value at risk:
quantile(sim1[,n_days]- initial_wealth, prob=0.05)

##        5%
## -3692.513

# 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)
```

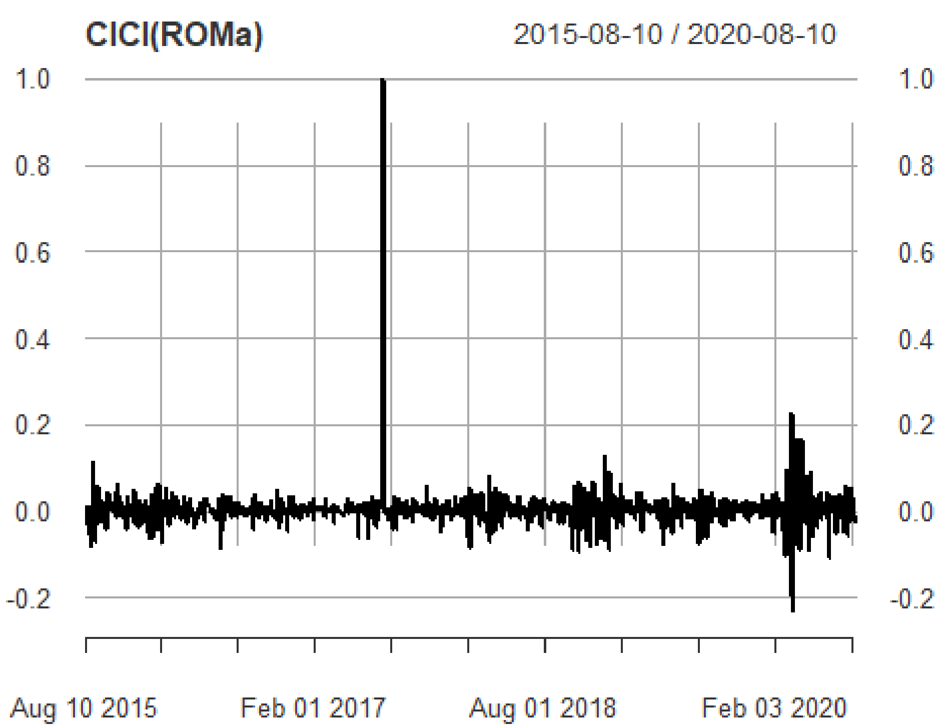
```
# Look at close-to-close changes
plot(ClCl(SOXLa))
```



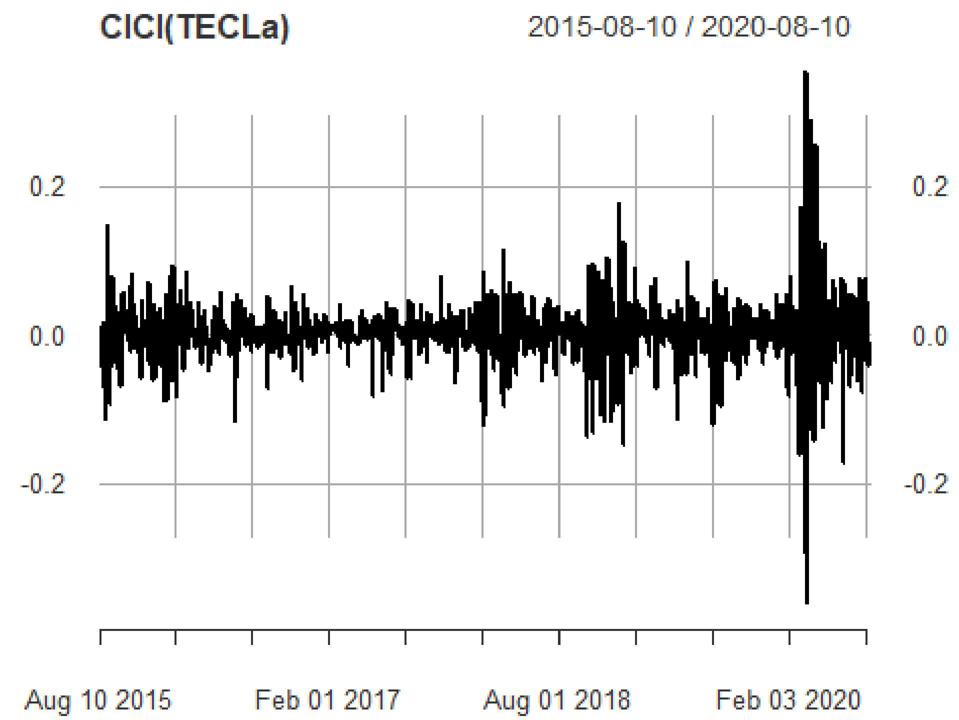

```
plot(ClCl(TQQQa))
```



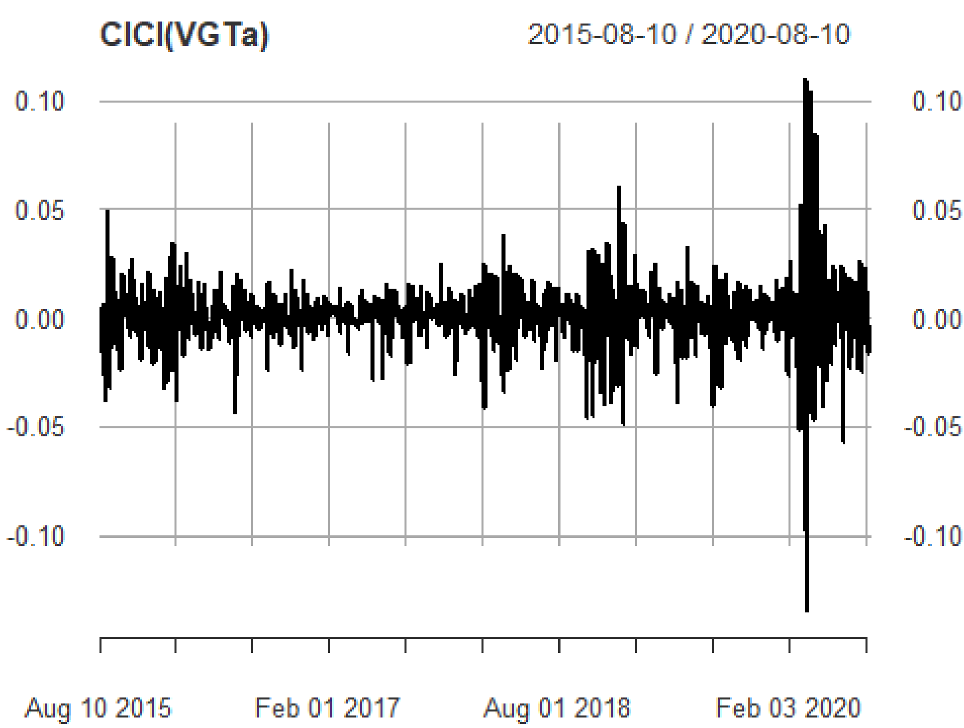

```
plot(ClCl(ROMa))
```




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



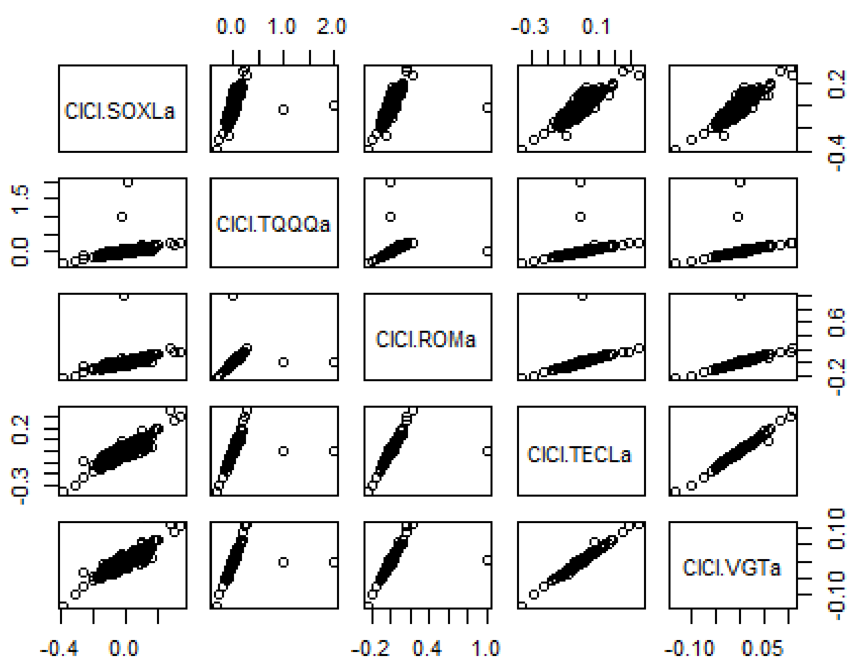

```
plot(ClCl(VGTa))
```



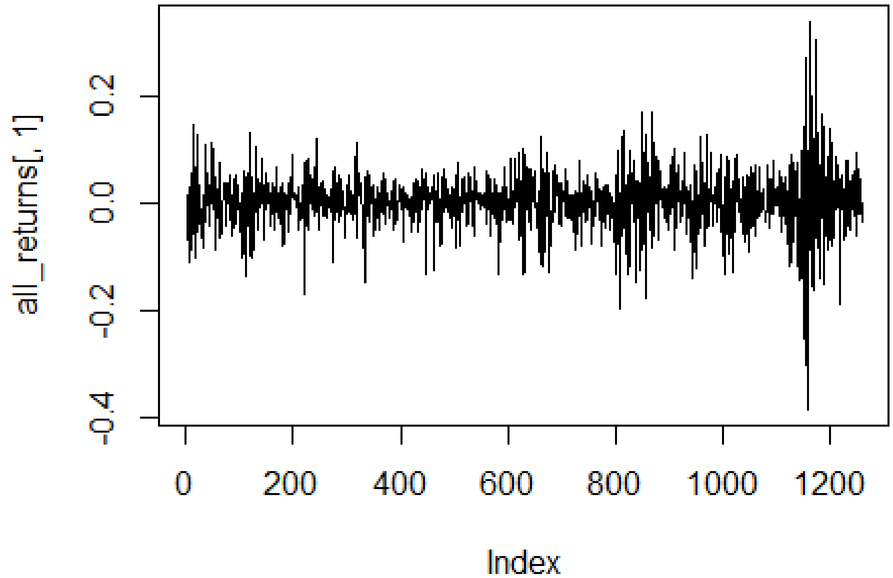
```r
# Combine close to close changes in a single matrix
all_returns =
cbind(ClCl(SOXLa),ClCl(TQQQa),ClCl(ROMa),ClCl(TECLa),ClCl(VGTa))
head(all_returns)
```

```
##              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
```

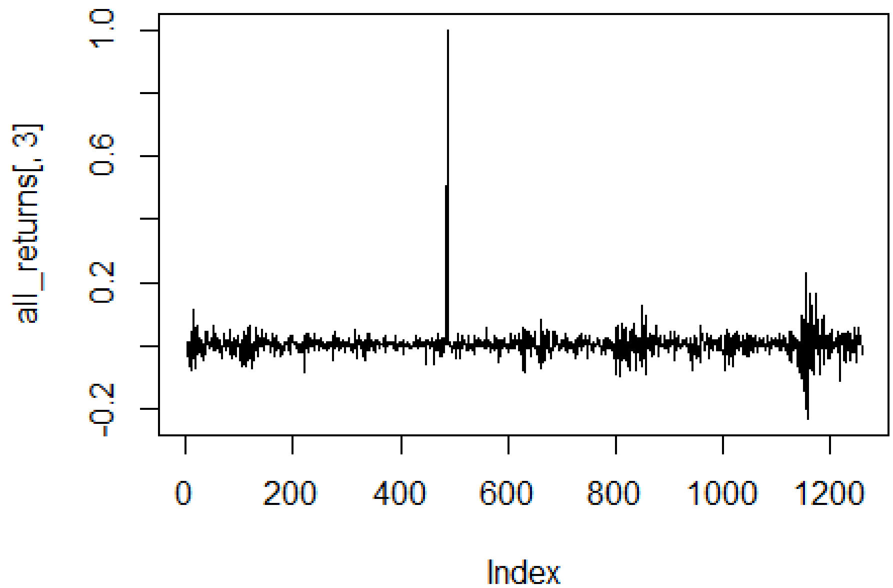
```r
# first row is NA because we didn't have a "before" in our data
all_returns = as.matrix(na.omit(all_returns))
N = nrow(all_returns)

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



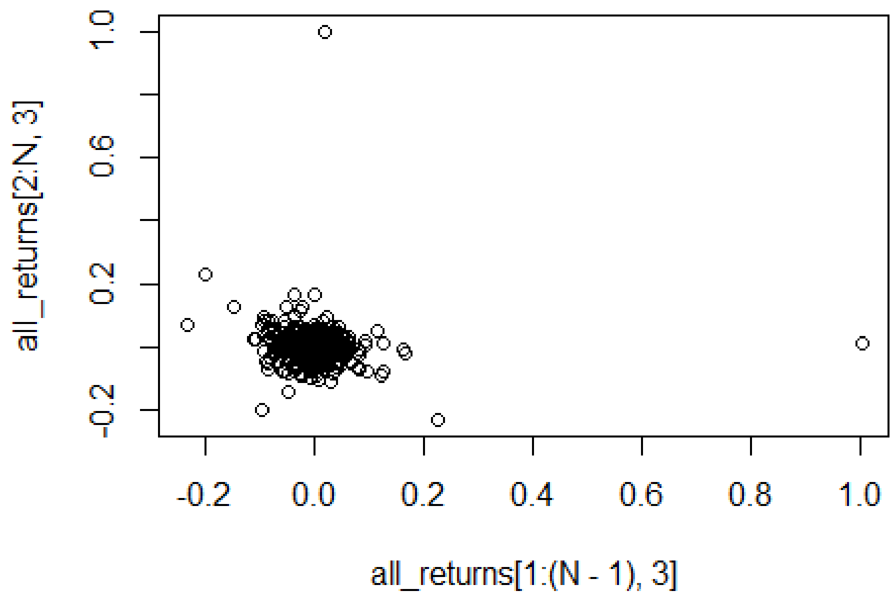

```r
plot(all_returns[,1], type='l')
```



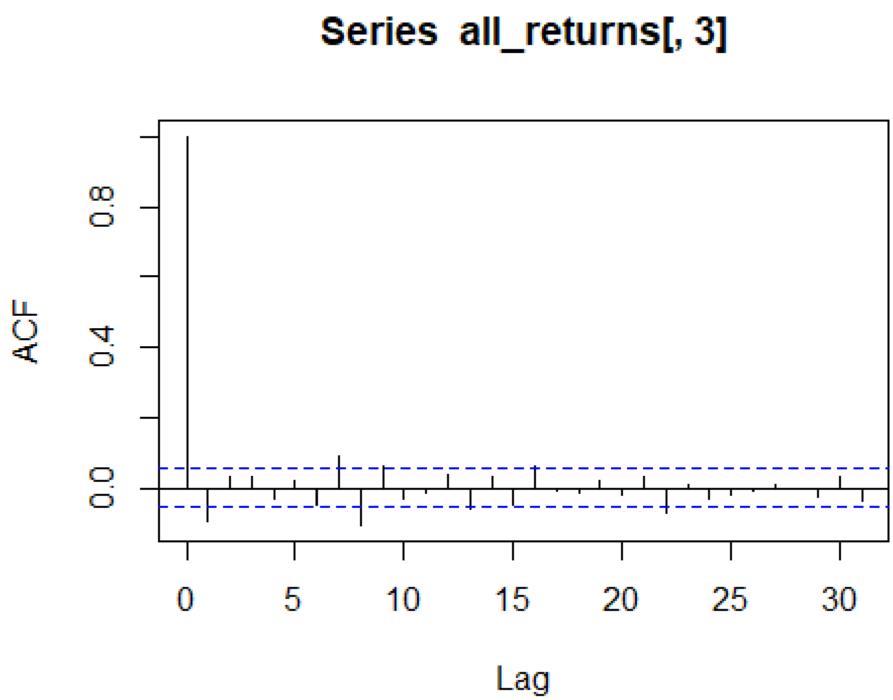

```
# Look at the market returns over time
plot(all_returns[,3], type='l')
```



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

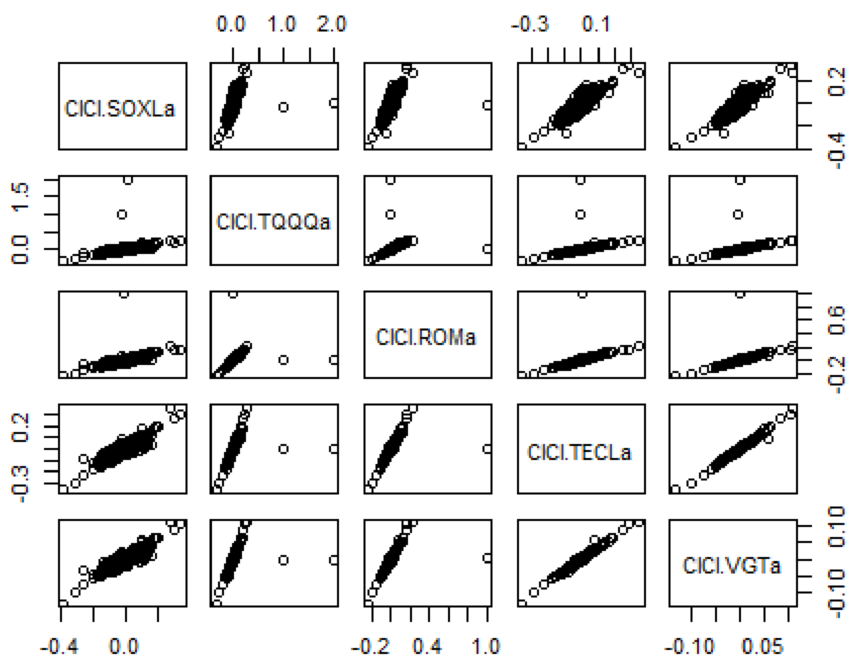



```r
# An autocorrelation plot: nothing there
acf(all_returns[,3])
```



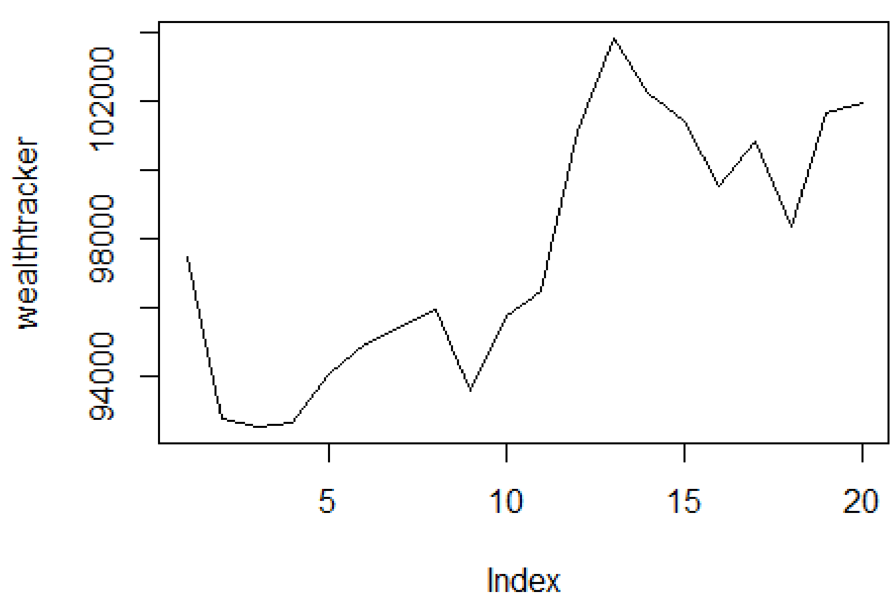

```
# conclusion: returns uncorrelated from one day to the next
# (makes sense, otherwise it'd be an easy inefficiency to exploit,
# and market inefficiencies that are exploited tend to disappear as a result)

#### 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)
```



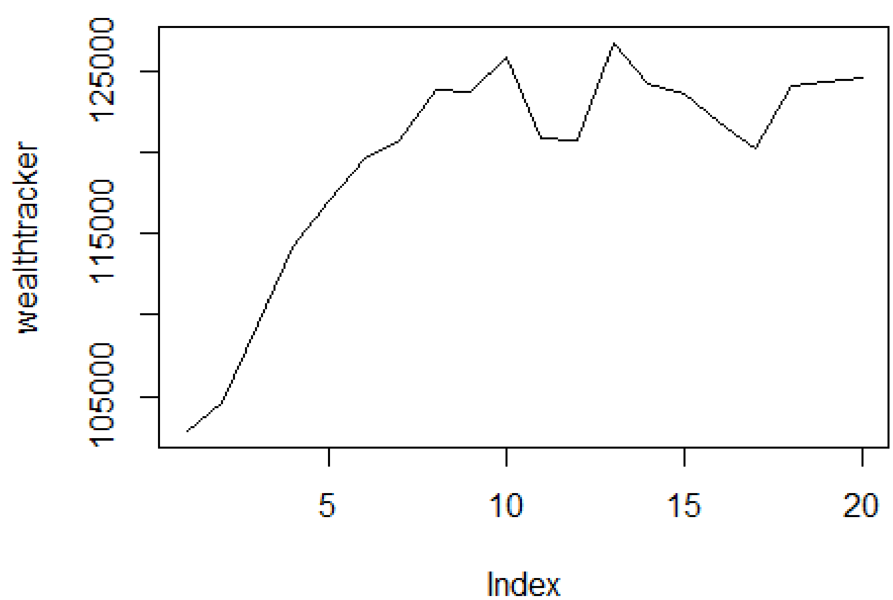

```
# Sample a random return from the empirical joint distribution
# This simulates a random day
return.today = resample(all_returns, 1, orig.ids=FALSE)

# 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
## 2016-10-31   20315.13   19886.51  20067.88   20008.46  20001.67

# 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] 101900.8

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



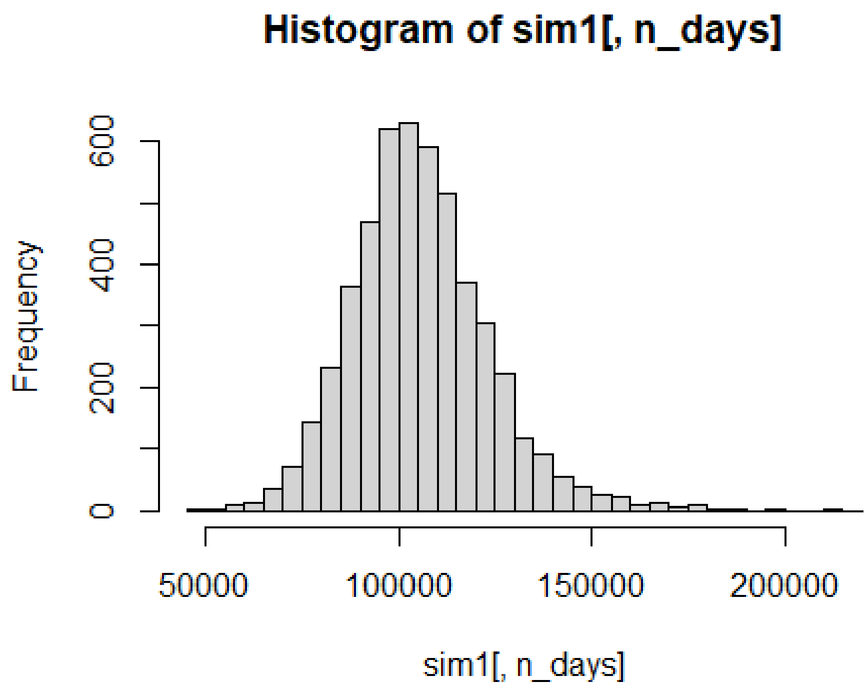

```
## end block
```

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

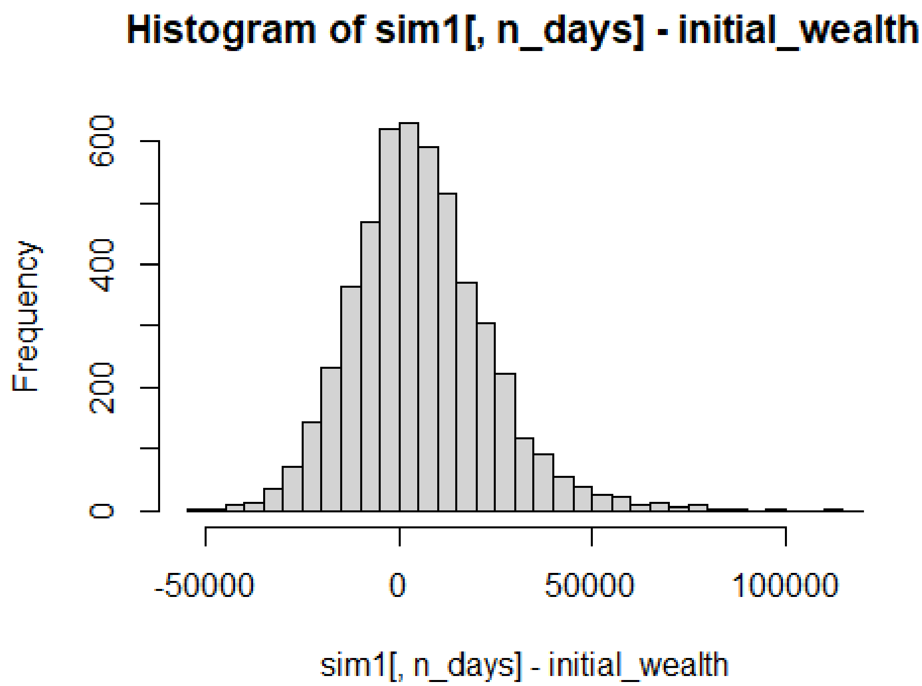



```r
# each row is a simulated trajectory
# each column is a data
head(sim1)

##                [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
[,7]
## result.1 100126.14  99167.61  99838.30  99724.30  99023.53  98262.74
103028.72
```

```
## result.2 101255.02 111230.78 111798.30 118741.47 123445.08 122628.47
120273.76
## result.3  97961.33  96538.96  95905.94  97880.74 102385.46 105962.89
105729.48
## result.4 103401.58 104573.75 104816.58 108811.61 109202.92 110462.73
109861.76
## result.5 100874.60  91850.08  93127.30  91064.53  91832.58  91365.84
92524.68
## result.6  98623.59  96231.10 106194.50 100625.53  97461.01  97935.14
100241.34
##               [,8]      [,9]     [,10]     [,11]     [,12]     [,13]
[,14]
## result.1  92937.15  94594.53  95981.64  96959.79 102721.21 103812.98
103574.49
## result.2 120560.42 122163.95 118464.61 119511.24 120544.22 119512.28
121759.58
## result.3 107128.98 104951.10  96184.31  96431.13  97175.29  98934.66
96132.22
## result.4 110843.25 108893.89 102128.56 104640.54 101740.50  99518.02
87681.69
## result.5  99581.20  97596.87  99249.98  97399.08  97251.61  94602.72
95122.60
## result.6 101480.40 105940.91 104389.45  97552.56  96632.26  98988.00
103130.06
##              [,15]     [,16]     [,17]     [,18]     [,19]     [,20]
## result.1 105179.60 116935.71 116886.16  89202.66  87720.57  85547.29
## result.2 125210.18 124534.04 121816.49 122338.21 127136.51 127776.46
## result.3  96121.42  99187.27  99689.84  95581.38  95576.41  98048.99
## result.4  90407.26  91700.76  95511.67 101378.10  95059.75 100605.39
## result.5  96592.71  96957.41 100214.03  99270.41  99652.48  99155.12
## result.6 102430.23 107412.81 108428.01 104571.65 101678.10 102354.61

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




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

## [1] 105734.6

mean(sim1[,n_days] - initial_wealth)

## [1] 5734.63

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




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

##        5%
## -20855.58
```

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
- 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

Low…

- Chatter
- Shopping

Market Segment 7:

High…

- Automotive
- Online Gaming
- Sports playing

Low…

- Arts
- TV Film

Market Segment 8:

High…

- Automotive
- News
- Tv film

Low…

- 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
```

```
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
```

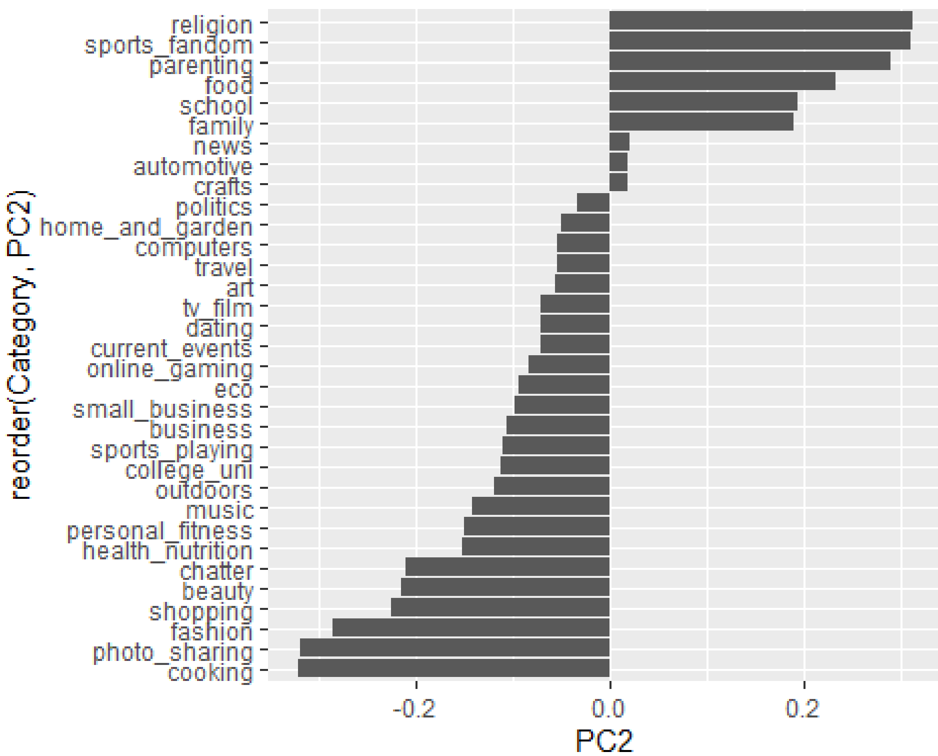
```
## 19           music -0.015176251
## 20   photo_sharing -0.023874179
## 21             eco -0.032698589
## 22   sports_fandom -0.046091360
## 23          family -0.047131188
## 24          school -0.078432899
## 25       parenting -0.084044430
## 26        religion -0.086883035
## 27            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))
```

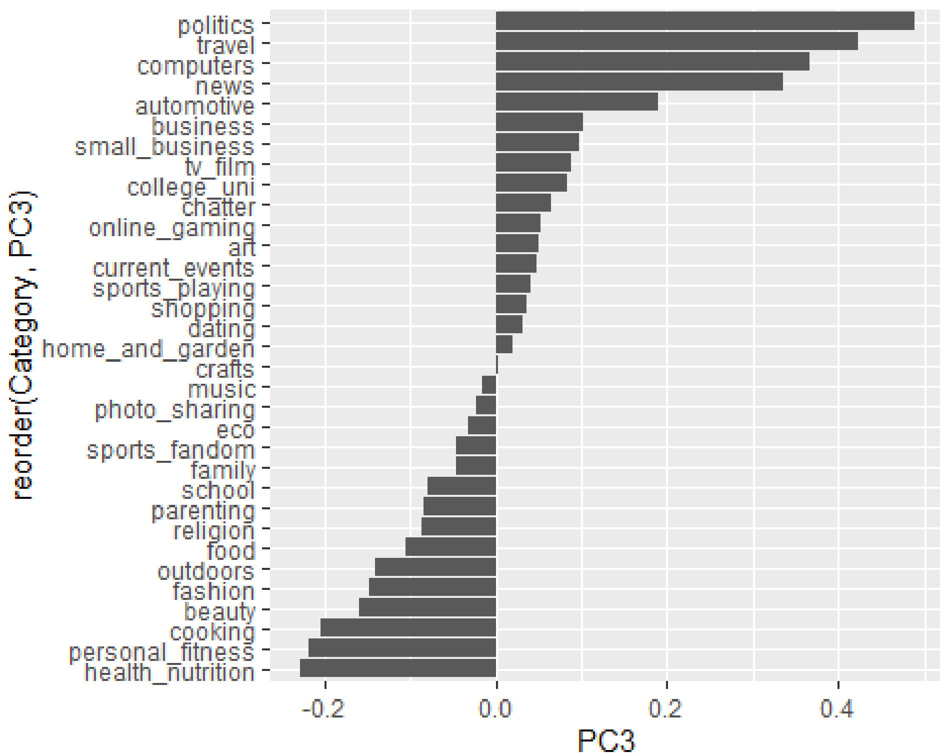
```
##            Category          PC4
## 1  health_nutrition  0.463218592
## 2  personal_fitness  0.443782383
## 3          outdoors  0.413458799
## 4          politics  0.194231190
## 5              news  0.176097051
## 6            travel  0.142861467
## 7         computers  0.135653450
## 8               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
```

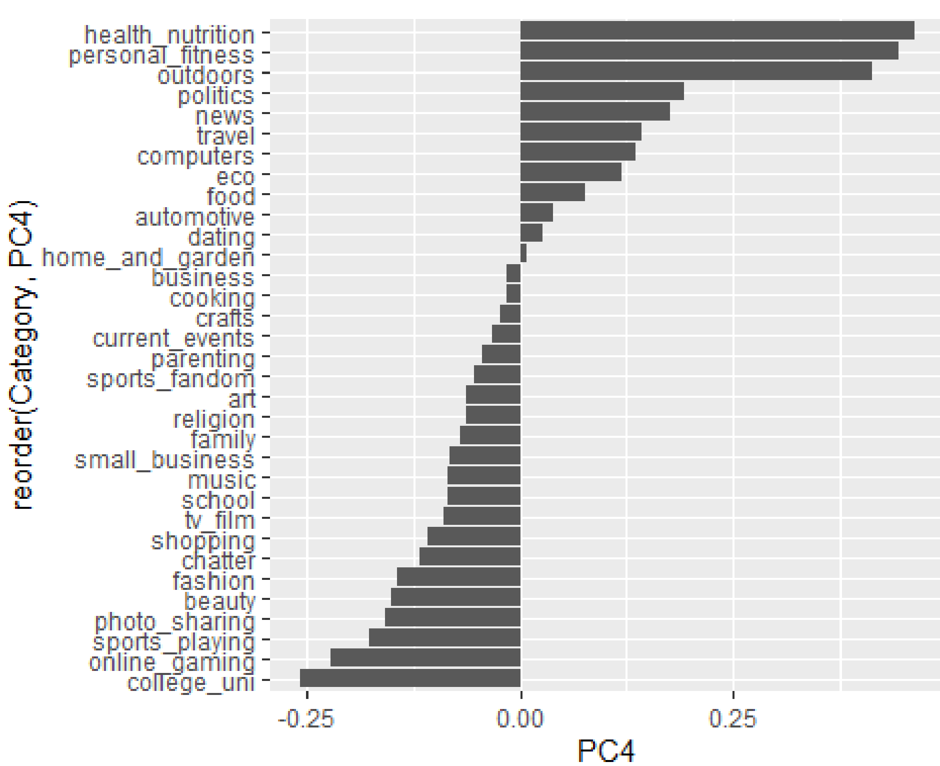
```r
ggplot(loadings) +
  geom_col(aes(x=reorder(Category, PC2), y=PC2)) +
  coord_flip()
```



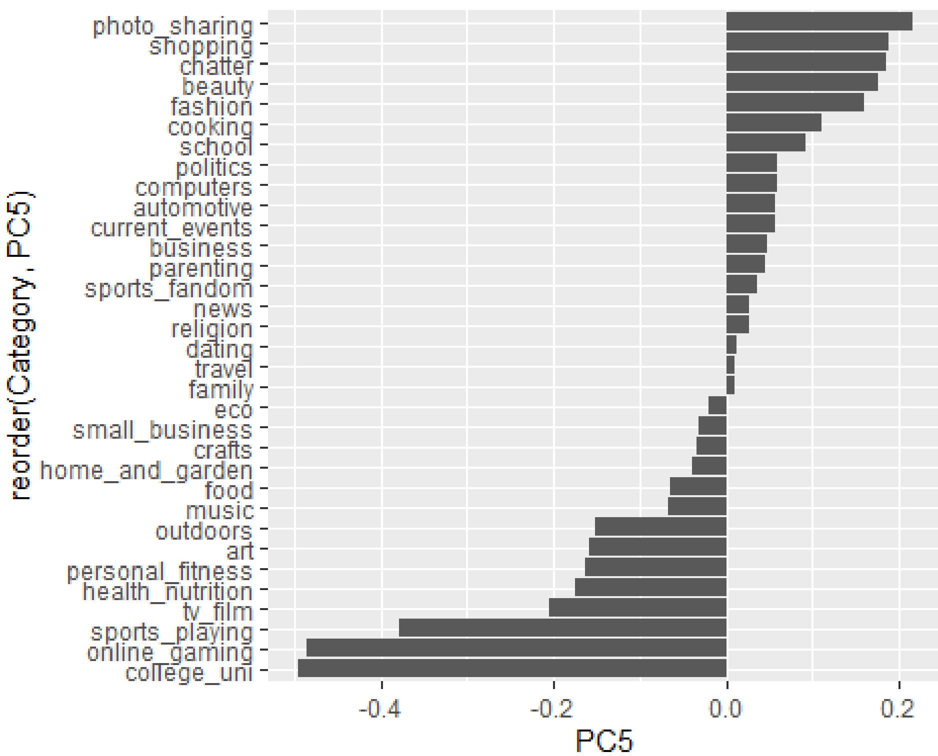

```r
ggplot(loadings) +
  geom_col(aes(x=reorder(Category, PC3), y=PC3)) +
  coord_flip()
```



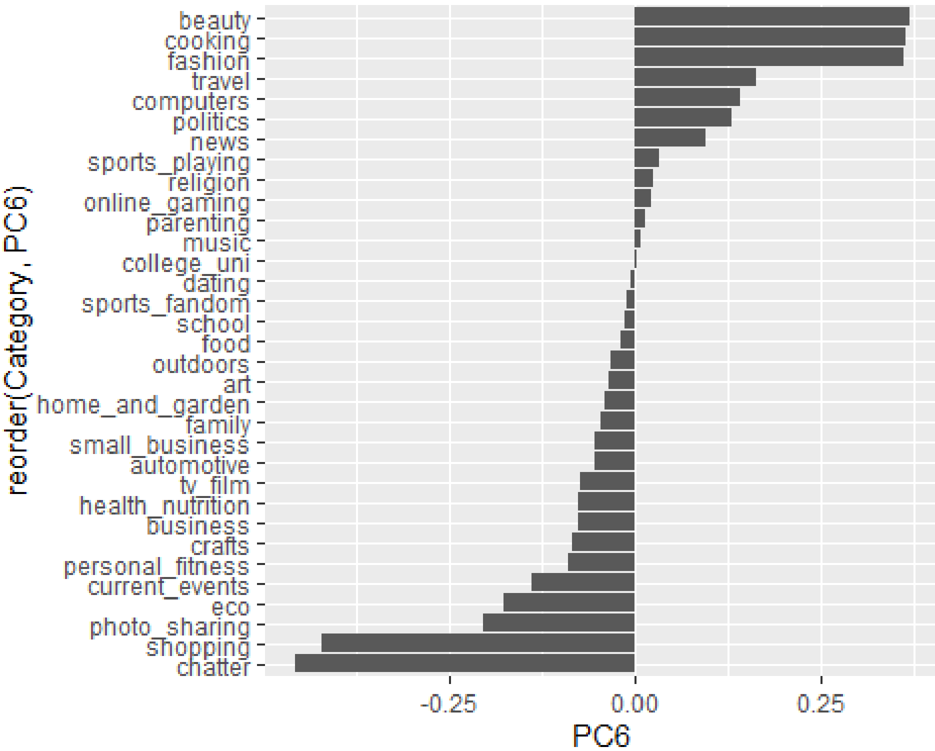

```r
ggplot(loadings) +
  geom_col(aes(x=reorder(Category, PC4), y=PC4)) +
  coord_flip()
```



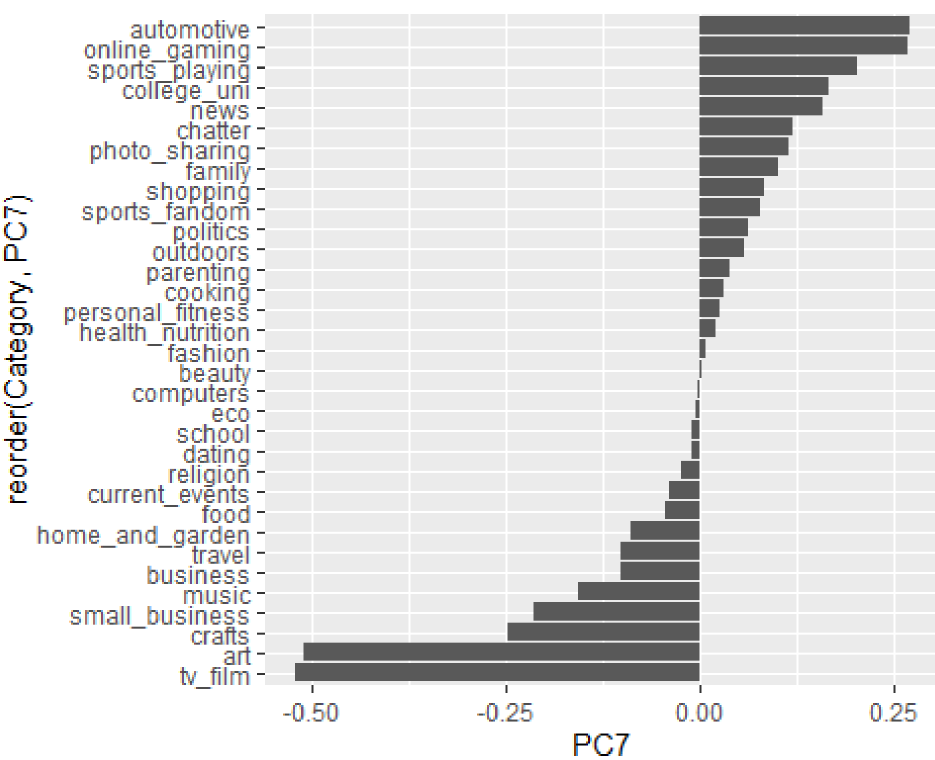
```r
ggplot(loadings) +
  geom_col(aes(x=reorder(Category, PC5), y=PC5)) +
  coord_flip()
```



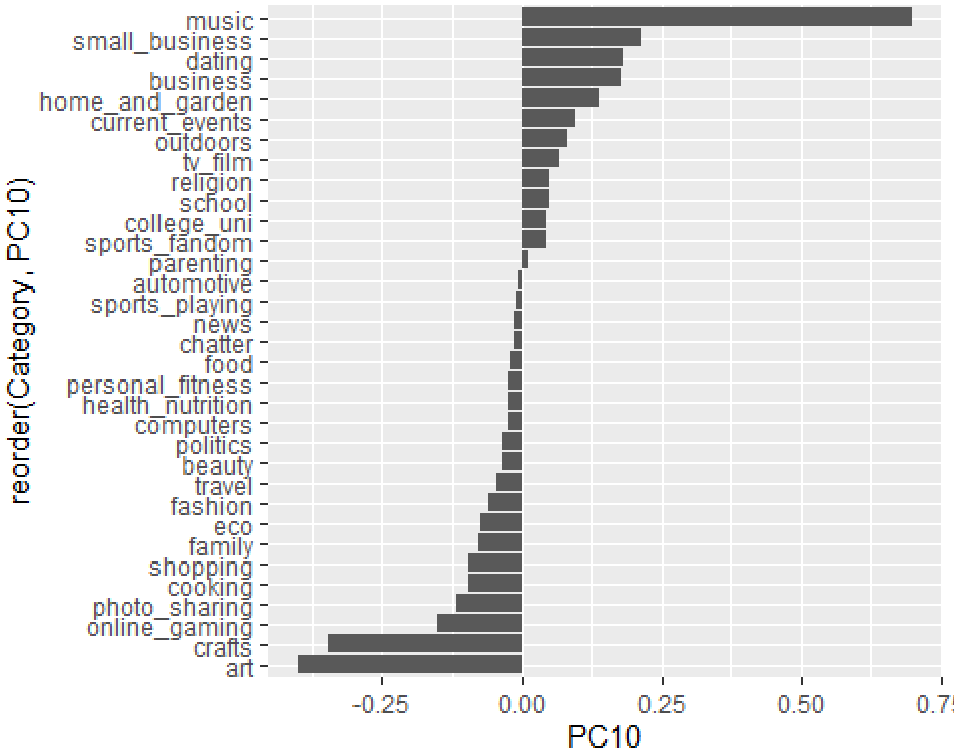

```r
ggplot(loadings) +
  geom_col(aes(x=reorder(Category, PC6), y=PC6)) +
  coord_flip()
```



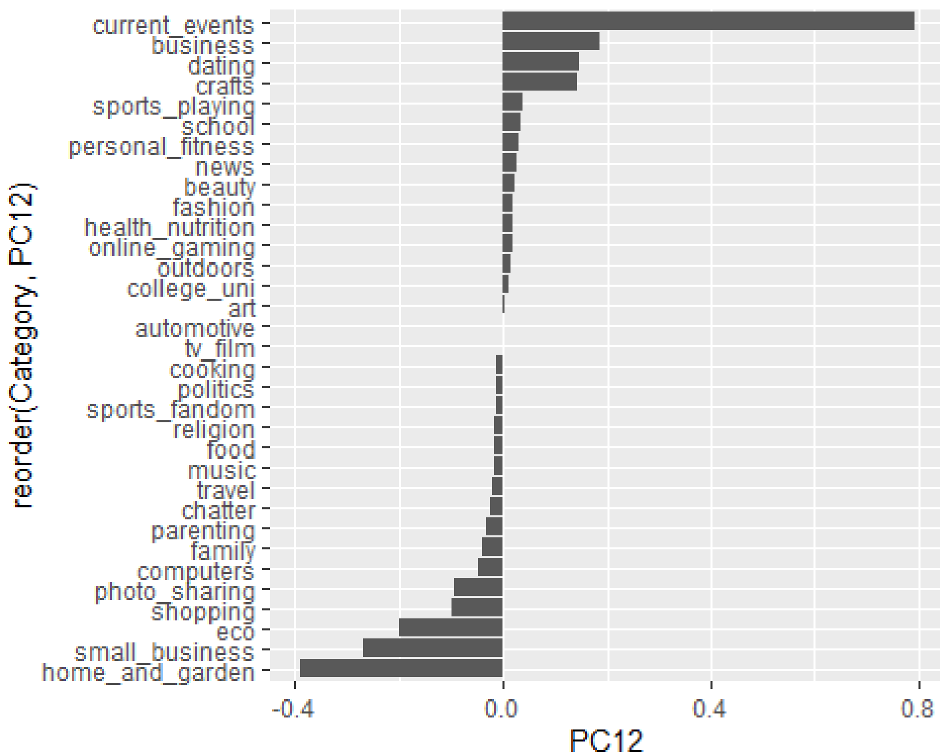

```
ggplot(loadings) +
  geom_col(aes(x=reorder(Category, PC7), y=PC7)) +
  coord_flip()
```



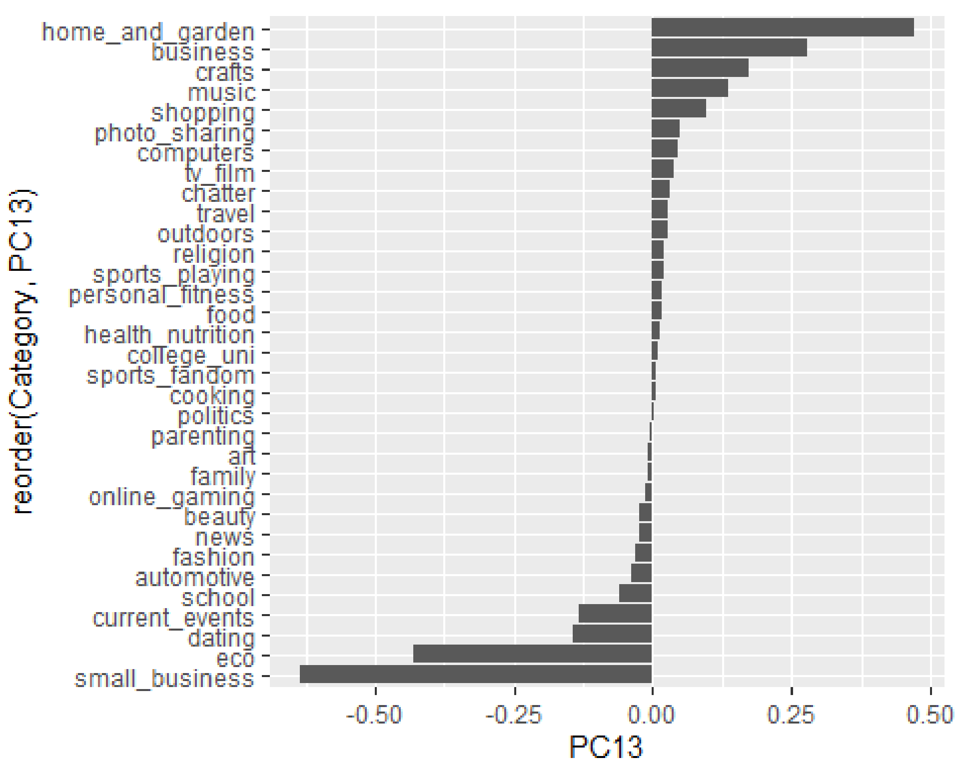
```r
ggplot(loadings) +
  geom_col(aes(x=reorder(Category, PC10), y=PC10)) +
  coord_flip()
```



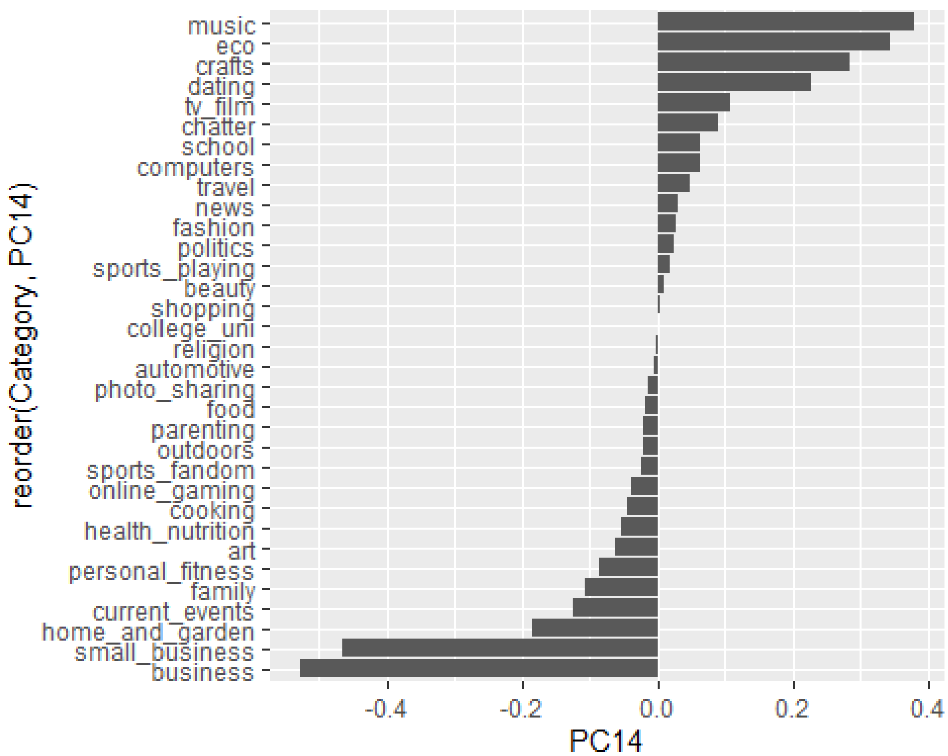

```r
ggplot(loadings) +
  geom_col(aes(x=reorder(Category, PC12), y=PC12)) +
  coord_flip()
```



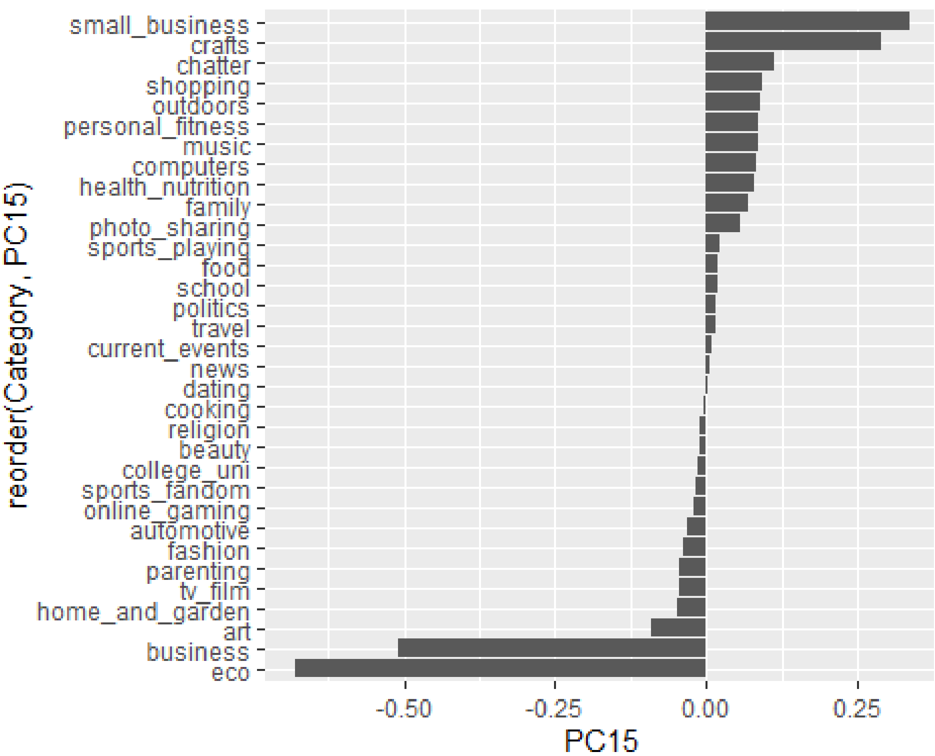

```r
ggplot(loadings) +
  geom_col(aes(x=reorder(Category, PC13), y=PC13)) +
  coord_flip()
```



```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 then transposed the TF IDF matrices for each author so the matrix had the words as rows and the documents as columns. 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.

```r
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)
```

```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 %>%
  { 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
```

```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("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
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))
```

```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("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
```

```r
## 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)
```

```r
# 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,
```

```r
c("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
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("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
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("usersmachuonedrivedocumentsgithubstadatareutersсctrainsimoncowellnewsmltx
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 = '') } %>%
```

```r
  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("cusersmachuonedrivedocumentsgithubstadatareutersctrainsimoncowellnewsmltx
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)
```

```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(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("cusersmachuonedrivedocumentsgithubstadatareutersctrainsimoncowellnewsmltx
t"))

## create a doc-term-matrix from the corpus
```

```r
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
```

```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("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
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 ))
```

```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_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("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
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 =
```

```r
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
```

```r
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
# 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),
```

```r
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
```

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

```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("cusersmachuonedrivedocumentsgithubstadatareutersсctrainsimoncowellnewsmltx
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
```

```r
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("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
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'))
```

```r
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("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
t"))


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

DTM_kirstin  = removeSparseTerms(DTM_kirstin , 0.95)
```

```r
# 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("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("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltxt"))


## 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/C50train/')

#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/',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 =
```

```r
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("cusersmachuonedrivedocumentsgithubstadatareutersc
ctrainsimoncowellnewsmltx
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("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
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("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


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

```r
## 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("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("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
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("cusersmachuonedrivedocumentsgithubstadatareutersctrainsimoncowellnewsmltx
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
```

```r
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("cusersmachuonedrivedocumentsgithubstadatareutersccttrainsimoncowellnewsmltx
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("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
```

```r
  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_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
```

```r
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("cusersmachuonedrivedocumentsgithubstadatareutersctrainsimoncowellnewsmltx
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"))
```

```r
## 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("usersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
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("usersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
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("cusersmachuonedrivedocumentsgithubstadatareutersсctrainsimoncowellnewsmltx
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

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] =
```

```r
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("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)

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("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
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'))
```

```r
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)
```

```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/',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("cusersmachuonedrivedocumentsgithubstadatareutersc ctrainsimoncowellnewsmltx
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
```

```r
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("cusersmachuonedrivedocumentsgithubstadatareutersc50trainsimoncowellnewsmltx
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
```

```r
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("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
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'))
```

```r
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("usersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
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("cusersmachuonedrivedocumentsgithubstadatareutersctrainsimoncowellnewsmltx
```

```
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("usersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
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
```

```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(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("cusersmachuonedrivedocumentsgithubstadatareutersсctrainsimoncowellnewsmltx
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("cusersmachuonedrivedocumentsgithubstadatareutersctrainsimoncowellnewsmltx
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
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) } %>%
```

```r
  { 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)
```

```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(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
```

```r
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
```

```r
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("cusersmachuonedrivedocumentsgithubstadatareutersctrainsimoncowellnewsmltx
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
```

```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_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("cusersmachuonedrivedocumentsgithubstadatareutersccttrainsimoncowellnewsmltx
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("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
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
####
```

```r
#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"))
```

```r
#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)



### 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("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


## 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/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("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
#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/',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("cusersmachuonedrivedocumentsgithubstadatareutersctrainsimoncowellnewsmltx
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'))

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
```

```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_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("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
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))

## 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] =
```

```r
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("cusersmachuonedrivedocumentsgithubstadatareutersсctrainsimoncowellnewsmltx
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
# 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 %>%
  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_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("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
mynamesAl = file_listAl %>%
  { strsplit(., '/', fixed=TRUE) } %>%
  { lapply(., tail, n=2) } %>%
```

```r
  { 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 =
```

```r
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("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
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
```

```r
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 )
```

```r
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("cusersmachuonedrivedocumentsgithubstadatareutersctrainsimoncowellnewsmltx
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
```

```r
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...
#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)
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/C50test/',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 = '') } %>%
```

```r
  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

## 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)
```

```r
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("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]




### 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)


# 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.
# 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("cusersmachuonedrivedocumentsgithubstadatareutersccctraindavidcowellnewsmltx
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
  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) } %>%
```

```r
  { 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)
```

```r
# 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("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)]
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

# 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(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)]
```

```r
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("cusersmachuonedrivedocumentsgithubstadatareutersccttrainjancowellnewsmltxt"
))

## 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)
```

```r
# 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
```

```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("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")
#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 %>%
  { 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)


########################################################################
```

```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'))
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("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
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
```

```r
  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 = '') } %>%
```

```r
  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("cusersmachuonedrivedocumentsgithubstadatareutersctrainsimoncowellnewsmltx
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("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
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("usersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
t"))


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

```r
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
```

```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_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("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
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:
```

```r
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)

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("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
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("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
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
```

```r
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
```

```r
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
# 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("cusersmachuonedrivedocumentsgithubstadatareutersctrainsimoncowellnewsmltx
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 =
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)

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("cusersmachuonedrivedocumentsgithubstadatareutersc ctrainsimoncowellnewsmltx
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("cusersmachuonedrivedocumentsgithubstadatareutersсctrainsimoncowellnewsmltx
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"))
```

```r
#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("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
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("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))

## 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_mark  = DocumentTermMatrix(my_documents)
```

```r
## 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("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("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
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

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

```r
## 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("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
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'))

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)
```

```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/',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),
```

```r
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("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
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))

## Some pre-processing/tokenization steps.
```

```r
## 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

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("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
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
# 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("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
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 )
```

```r
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("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
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'))

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
```

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

```r
## 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("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsimoncowellnewsmltx
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("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsimoncowellnewsmltx
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/')
```

```r
#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
```

```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(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)
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 %>%
  { 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

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

```r
## 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("cusersmachuonedrivedocumentsgithubstadatareutersc ctrainbencowellnewsmltxt"
))


## 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
mynames = file_list_test %>%
```

```r
  { 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("cusersmachuonedrivedocumentsgithubstadatareutersccttrainbradcowellnewsmltxt
"))


## 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)


# 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.
```

```r
## 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.
# this is just what I hacked together
```

```r
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("cusersmachuonedrivedocumentsgithubstadatareutersccctrainednacowellnewsmltxt
"))


## 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
```

```r
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'))

fumiko_test = lapply(file_list_test, readerPlain)
```

```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

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)]
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("cusersmachuonedrivedocumentsgithubstadatareutersccttraingrahamcowellnewsmlt
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))
```

```r
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")
#stopwords("SMART")
```

```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("cusersmachuonedrivedocumentsgithubstadatareutersccctrainjimcowellnewsmltxt"
))


## 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
## create a text mining 'corpus' with:
```

```r
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'))

joe_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(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
```

```r
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("cusersmachuonedrivedocumentsgithubstadatareuterscctrainjohncowellnewsmltxt
"))
```

```r
## 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("cusersmachuonedrivedocumentsgithubstadatareuterscctrainjonathancowellnewsmltxt"))


## 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/C50test/',KarlPenhaul,'/*.txt'))

karl_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(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)]
```

```r
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"))
```

```
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareuterscctrainkeithcowellnewsmltx
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:
```

```r
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("cusersmachuonedrivedocumentsgithubstadatareutersccctrainkevindcowellnewsmlt
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
```

```
# 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("cusersmachuonedrivedocumentsgithubstadatareutersccctrainkirstincowellnewsml
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
```

```r
## 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("cusersmachuonedrivedocumentsgithubstadatareuterscctrainkouroshcowellnewsml
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("cusersmachuonedrivedocumentsgithubstadatareutersccctrainlynnecowellnewsmltx
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
```

```
# 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("cusersmachuonedrivedocumentsgithubstadatareutersctrainmarkcowellnewsmltxt
"))
```

```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("cusersmachuonedrivedocumentsgithubstadatareutersccctrainmartincowellnewsmlt
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
```

```
# 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("cusersmachuonedrivedocumentsgithubstadatareutersccctrainmurecowellnewsmltxt
"))


## 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
```

```r
# 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
```

```
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("cusersmachuonedrivedocumentsgithubstadatareuterscctrainpetercowellnewsmltx
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)
```

```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(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("cusersmachuonedrivedocumentsgithubstadatareutersctrainpierrecowellnewsmlt
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("cusersmachuonedrivedocumentsgithubstadatareutersccctrainrobincowellnewsmltx
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("cusersmachuonedrivedocumentsgithubstadatareutersccctrainsamuelcowellnewsmlt
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)
```

```
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("cusersmachuonedrivedocumentsgithubstadatareuterscctrainsarahcowellnewsmltx
t"))
```

```
## 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("cusersmachuonedrivedocumentsgithubstadatareutersccraintancowellnewsmltxt"
))


## 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'))
```

```r
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("cusersmachuonedrivedocumentsgithubstadatareuterscctraintheresecowellnewsml
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
```

```r
white-space

my_documents = tm_map(my_documents, content_transformer(removeWords),
stopwords("en"))
my_documents <- tm_map(my_documents, removeWords,
c("cusersmachuonedrivedocumentsgithubstadatareutersccctraintimcowellnewsmltxt"
))


## 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("cusersmachuonedrivedocumentsgithubstadatareutersccctraintoddcowellnewsmltxt
"))


## 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]
```

```
transpose_simon <- as.data.frame(t(simon_DF))
transpose_simon$Author <- SimonCowell

transpose_aaron <- as.data.frame(t(aaron_DF))
transpose_aaron$Author <- AaronPressman

transpose_alan <- as.data.frame(t(alan_DF))
transpose_alan$Author <- AlanCrosby
transpose_alex <- as.data.frame(t(alex_DF))
transpose_alex$Author <- AlexanderSmith

transpose_ben <- as.data.frame(t(ben_DF))
transpose_ben$Author <- BenjaminKangLim

transpose_brad <- as.data.frame(t(brad_DF))
transpose_brad$Author <- BradDorfman

tranpose_bernard <- as.data.frame(t(bernard_DF))
tranpose_bernard$Author <- BernardHickey

transpose_darren <- as.data.frame(t(darren_DF))
transpose_darren$Author <- DarrenSchuettler

transpose_david <- as.data.frame(t(david_DF))
transpose_david$Author <- DavidLawder

transpose_edna <- as.data.frame(t(edna_DF))
transpose_edna$Author <- EdnaFernandes
```

```
transpose_eric <- as.data.frame(t(eric_DF))
transpose_eric$Author <- EricAuchard

transpose_fumiko <- as.data.frame(t(fumiko_DF))
transpose_fumiko$Author <- FumikoFujisaki

transpose_graham <- as.data.frame(t(graham_DF))
transpose_graham$Author <- GrahamEarnshaw

transpose_heather <- as.data.frame(t(heather_DF))
transpose_heather$Author <- HeatherScoffield

transpose_jan <-as.data.frame(t(jan_DF))
transpose_jan$Author <- JanLopatka

transpose_jane <- as.data.frame(t(jane_DF))
transpose_jane$Author <- JaneMacartney

transpose_jim <- as.data.frame(t(jim_DF))
transpose_jim$Author <- JimGilchrist

transpose_joe <- as.data.frame(t(joe_DF))
transpose_joe$Author <- JoeOrtiz

transpose_jo <- as.data.frame(t(jo_DF))
transpose_jo$Author <- JoWinterbottom
```

```
transpose_jonathan <- as.data.frame(t(jonathan_DF))
transpose_jonathan$Author <- JonathanBirt
transpose_john <- as.data.frame(t(john_DF))
transpose_john$Author <- JohnMastrini

transpose_karl <- as.data.frame(t(karl_DF))
transpose_karl$Author <- KarlPenhaul

transpose_keith <- as.data.frame(t(keith_DF))
transpose_keith$Author <- KeithWeir

transpose_kevind <- as.data.frame(t(kevind_DF))
transpose_kevind$Author <- KevinDrawbaugh

transpose_kevinm <- as.data.frame(t(kevinm_DF))
transpose_kevinm$Author <- KevinMorrison

transpose_kirstin <- as.data.frame(t(kirstin_DF))
transpose_kirstin$Author <- KirstinRidley

transpose_kourosh <- as.data.frame(t(kourosh_DF))
transpose_kourosh$Author <- KouroshKarimkhany

transpose_lynne <- as.data.frame(t(lynne_DF))
transpose_lynne$Author <- `LynneO'Donnell`

transpose_lydia <- as.data.frame(t(lydia_DF))
transpose_lydia$Author <- LydiaZajc
```

```
transpose_lynnley <- as.data.frame(t(lynnley_DF))

transpose_lynnley$Author <- LynnleyBrowning


transpose_marcel <- as.data.frame(t(marcel_DF))

transpose_marcel$Author <- MarcelMichelson


transpose_mark <- as.data.frame( t(mark_DF))

transpose_mark$Author <- MarkBendeich


transpose_martin <- as.data.frame(t(martin_DF))

transpose_martin$Author <- MartinWolk


transpose_michael <- as.data.frame(t(michael_DF))

transpose_michael$Author <- MichaelConnor


transpose_matthew <- as.data.frame( t(matthew_DF))

transpose_matthew$Author <- MatthewBunce


transpose_mure <- as.data.frame(t(mure_DF))

transpose_mure$Author <- MureDickie


transpose_nick <- as.data.frame(t(nick_DF))

transpose_nick$Author <- NickLouth

transpose_patricia <- as.data.frame(t(patricia_DF))

transpose_patricia$Author <- PatriciaCommins


transpose_peter <- as.data.frame(t(peter_DF))
```

```
transpose_peter$Author <- PeterHumphrey


transpose_pierre <-as.data.frame(t(pierre_DF))

transpose_pierre$Author <- PierreTran


transpose_robin <- as.data.frame(t(robin_DF))

transpose_robin$Author <- RobinSidel


transpose_roger <-as.data.frame(t(roger_DF))

transpose_roger$Author <- RogerFillion

transpose_samuel<-as.data.frame(t(samuel_DF))

transpose_samuel$Author <- SamuelPerry

transpose_sarah <- as.data.frame(t(sarah_DF))

transpose_sarah$Author <- SarahDavison

transpose_scott <- as.data.frame(t(scott_DF))

transpose_scott$Author <- ScottHillis

transpose_tan <- as.data.frame(t(tan_DF))

transpose_tan$Author <- TanEeLyn

transpose_therese <- as.data.frame(t(therese_DF))

transpose_therese$Author <- TheresePoletti

transpose_todd <- as.data.frame(t(todd_DF))

transpose_todd$Author <- ToddNissen

transpose_tim <- as.data.frame(t(tim_DF))

transpose_tim$Author <- TimFarrand


transpose_will <- as.data.frame(t(will_DF))

transpose_will$Author <- WilliamKazer

train3 <- bind_rows(transpose_alan, transpose_alex,transpose_darren,
transpose_david,
```

```
                    transpose_edna, transpose_eric,
transpose_fumiko,transpose_graham,

transpose_heather,transpose_jan,transpose_jim,transpose_jane,

transpose_jo,transpose_joe,transpose_jonathan,transpose_john,

transpose_karl,transpose_keith,transpose_kevind,transpose_kevinm,

transpose_kirstin,transpose_kourosh,transpose_lydia,transpose_lynne,

transpose_lynnley,transpose_marcel,transpose_mark,transpose_martin,
                    transpose_matthew, transpose_michael,transpose_nick,
transpose_mure,

transpose_patricia,transpose_peter,transpose_pierre,transpose_robin,
                    transpose_roger,transpose_sarah, transpose_samuel)
transpose_simon_test <- as.data.frame(t(simon_DF_test))
transpose_simon_test$Author <- SimonCowell


transpose_aaron_test <- as.data.frame(t(aaron_DF_test))
transpose_aaron_test$Author <- AaronPressman


transpose_alan_test <- as.data.frame(t(alan_DF_test))
transpose_alan_test$Author <- AlanCrosby
transpose_alex_test <- as.data.frame(t(alex_DF_test))
transpose_alex_test$Author <- AlexanderSmith


transpose_ben_test <- as.data.frame(t(ben_DF_test))
transpose_ben_test$Author <- BenjaminKangLim


transpose_brad_test <- as.data.frame(t(brad_DF_test))
```

```
transpose_brad_test$Author <- BradDorfman


tranpose_bernard_test <- as.data.frame(t(bernard_DF_test))

tranpose_bernard_test$Author <- BernardHickey


transpose_darren_test <- as.data.frame(t(darren_DF_test))

transpose_darren_test$Author <- DarrenSchuettler


transpose_david_test <- as.data.frame(t(david_DF_test))

transpose_david_test$Author <- DavidLawder


transpose_edna_test <- as.data.frame(t(edna_DF_test))

transpose_edna$Author <- EdnaFernandes


transpose_eric_test <- as.data.frame(t(eric_DF_test))

transpose_eric_test$Author <- EricAuchard


transpose_fumiko_test <- as.data.frame(t(fumiko_DF_test))

transpose_fumiko_test$Author <- FumikoFujisaki


transpose_graham_test <- as.data.frame(t(graham_DF_test))

transpose_graham_test$Author <- GrahamEarnshaw


transpose_heather_test <- as.data.frame(t(heather_DF_test))

transpose_heather_test$Author <- HeatherScoffield


transpose_jan_test <-as.data.frame(t(jan_DF_test))

transpose_jan_test$Author <- JanLopatka
```

```
transpose_jane_test <- as.data.frame(t(jane_DF_test))
transpose_jane_test$Author <- JaneMacartney

transpose_jim_test <- as.data.frame(t(jim_DF_test))
transpose_jim_test$Author <- JimGilchrist

transpose_joe_test <- as.data.frame(t(joe_DF_test))
transpose_joe_test$Author <- JoeOrtiz

transpose_jo_test <- as.data.frame(t(jo_DF_test))
transpose_jo_test$Author <- JoWinterbottom

transpose_jonathan_test <- as.data.frame(t(jonathan_DF_test))
transpose_jonathan_test$Author <- JonathanBirt
transpose_john_test <- as.data.frame(t(john_DF_test))
transpose_john_test$Author <- JohnMastrini

transpose_karl_test <- as.data.frame(t(karl_DF_test))
transpose_karl_test$Author <- KarlPenhaul

transpose_keith_test <- as.data.frame(t(keith_DF_test))
transpose_keith_test$Author <- KeithWeir

transpose_kevind_test <- as.data.frame(t(kevind_DF_test))
transpose_kevind_test$Author <- KevinDrawbaugh

transpose_kevinm_test <- as.data.frame(t(kevinm_DF_test))
```

```
transpose_kevinm_test$Author <- KevinMorrison


transpose_kirstin_test <- as.data.frame(t(kirstin_DF_test))

transpose_kirstin_test$Author <- KirstinRidley


transpose_kourosh_test <- as.data.frame(t(kourosh_DF_test))

transpose_kourosh_test$Author <- KouroshKarimkhany


transpose_lynne_test <- as.data.frame(t(lynne_DF_test))

transpose_lynne_test$Author <- `LynneO'Donnell`


transpose_lydia_test <- as.data.frame(t(lydia_DF_test))

transpose_lydia_test$Author <- LydiaZajc


transpose_lynnley_test <- as.data.frame(t(lynnley_DF_test))

transpose_lynnley_test$Author <- LynnleyBrowning


transpose_marcel_test <- as.data.frame(t(marcel_DF_test))

transpose_marcel_test$Author <- MarcelMichelson


transpose_mark_test <- as.data.frame( t(mark_DF_test))

transpose_mark$Author <- MarkBendeich


transpose_martin_test <- as.data.frame(t(martin_DF_test))

transpose_martin_test$Author <- MartinWolk


transpose_michael_test <- as.data.frame(t(michael_DF_test))

transpose_michael_test$Author <- MichaelConnor
```

```
transpose_matthew_test <- as.data.frame( t(matthew_DF_test))
transpose_matthew_test$Author <- MatthewBunce

transpose_mure_test <- as.data.frame(t(mure_DF_test))
transpose_mure_test$Author <- MureDickie

transpose_nick_test <- as.data.frame(t(nick_DF_test))
transpose_nick_test$Author <- NickLouth
transpose_patricia_test <- as.data.frame(t(patricia_DF_test))
transpose_patricia_test$Author <- PatriciaCommins

transpose_peter_test <- as.data.frame(t(peter_DF_test))
transpose_peter_test$Author <- PeterHumphrey

transpose_pierre_test <-as.data.frame(t(pierre_DF_test))
transpose_pierre_test$Author <- PierreTran

transpose_robin_test <- as.data.frame(t(robin_DF_test))
transpose_robin_test$Author <- RobinSidel

transpose_roger_test <-as.data.frame(t(roger_DF_test))
transpose_roger_test$Author <- RogerFillion
transpose_samuel_test<-as.data.frame(t(samuel_DF_test))
transpose_samuel_test$Author <- SamuelPerry
transpose_sarah_test <- as.data.frame(t(sarah_DF_test))
transpose_sarah_test$Author <- SarahDavison
transpose_scott_test <- as.data.frame(t(scott_DF_test))
```

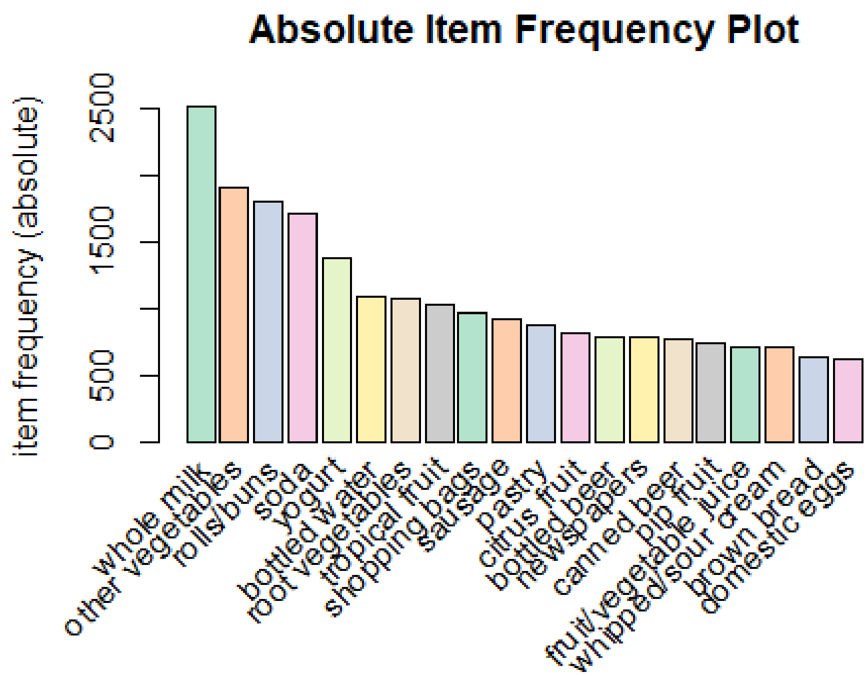
```
transpose_scott_test$Author <- ScottHillis

transpose_tan_test<- as.data.frame(t(tan_DF_test))

transpose_tan_test$Author <- TanEeLyn

transpose_therese_test <- as.data.frame(t(therese_DF_test))

transpose_therese_test$Author <- TheresePoletti

transpose_todd_test <- as.data.frame(t(todd_DF_test))

transpose_todd_test$Author <- ToddNissen

transpose_tim_test <- as.data.frame(t(tim_DF_test))

transpose_tim_test$Author <- TimFarrand

transpose_will_test <- as.data.frame(t(will_DF_test))

transpose_will_test$Author <- WilliamKazer


test3 <- bind_rows(transpose_alan_test,
transpose_alex_test,transpose_darren_test, transpose_david_test,

                   transpose_edna_test, transpose_eric_test,
transpose_fumiko_test,transpose_graham_test,


transpose_heather_test,transpose_jan_test,transpose_jim_test,transpose_jane_t
est,


transpose_jo_test,transpose_joe_test,transpose_jonathan_test,transpose_john_t
est,


transpose_karl_test,transpose_keith_test,transpose_kevind_test,transpose_kevi
nm_test,


transpose_kirstin_test,transpose_kourosh_test,transpose_lydia_test,transpose_
lynne_test,


transpose_lynnley_test,transpose_marcel_test,transpose_mark_test,transpose_ma
rtin_test,

                   transpose_matthew_test,
transpose_michael_test,transpose_nick_test, transpose_mure_test,
```

```
transpose_patricia_test,transpose_peter,transpose_pierre_test,transpose_robin_test,

                   transpose_roger_test,transpose_sarah_test,
transpose_samuel_test,transpose_sarah_test,transpose_scott_test,transpose_tan_test,


transpose_therese_test,transpose_todd_test,transpose_tim_test)






library(rpart)

control<-
rpart.control(minsplit=30,minbucket=10,cp=0.01,maxcomplete=6,maxsurrogate=8,usersurrogate=2,xval=15,surrogatestyle=0,maxdepth=15)

train.rpart<-rpart(Author~.,data=train3[1:51],control=control)

testpred.rpart<-predict(train.rpart,test3[,1:51],type="class")

trainpred.rpart <- predict(train.rpart,train3[,1:51],type = "class")

MisClassTest<-table("Predict"=testpred.rpart,"Actual"=test3$Author)  ## Test Data Prediction

MisClassTrain<-table("Predict"=trainpred.rpart,"Actual"=train3$Author)  ## Test Data Prediction



accuracyCART<-(100-mean(c((nrow(test3)-
sum(diag(MisClassTest)))/nrow(test3)),(nrow(train3)-
sum(diag(MisClassTrain)))/nrow(train3)))

accuracyCART

[1] 99.01905
```

Question 6

First, I looked 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.

```r
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")
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")
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].
```

```
# 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].
```

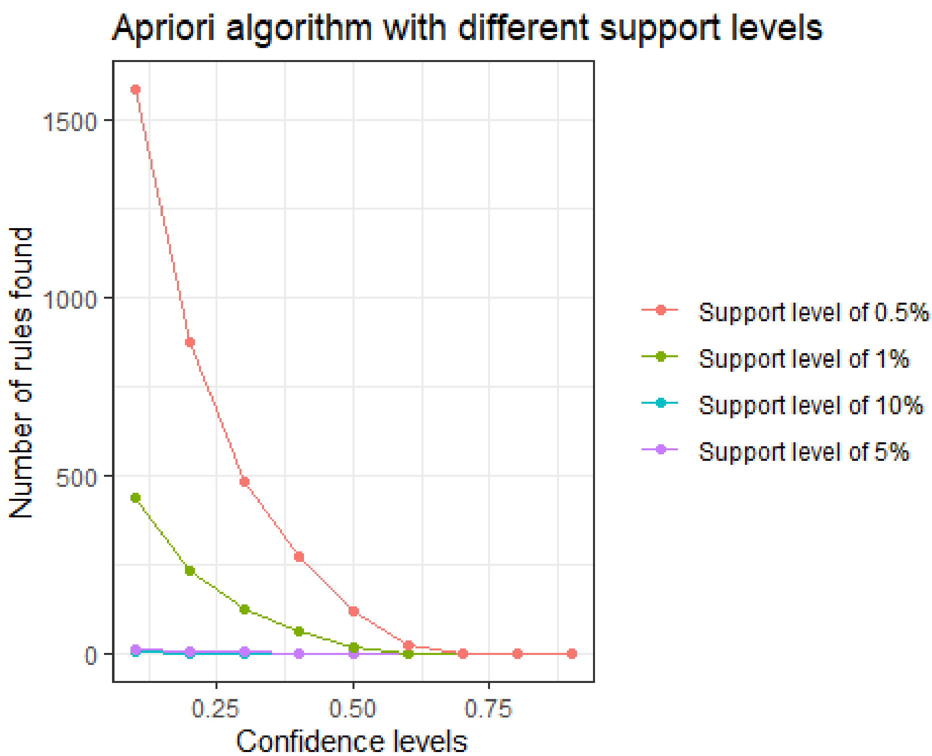
```
## writing ... [0 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.8    0.1    1 none FALSE            TRUE       5    0.01      1
##  maxlen target  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 [0.01s].
## 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].
```

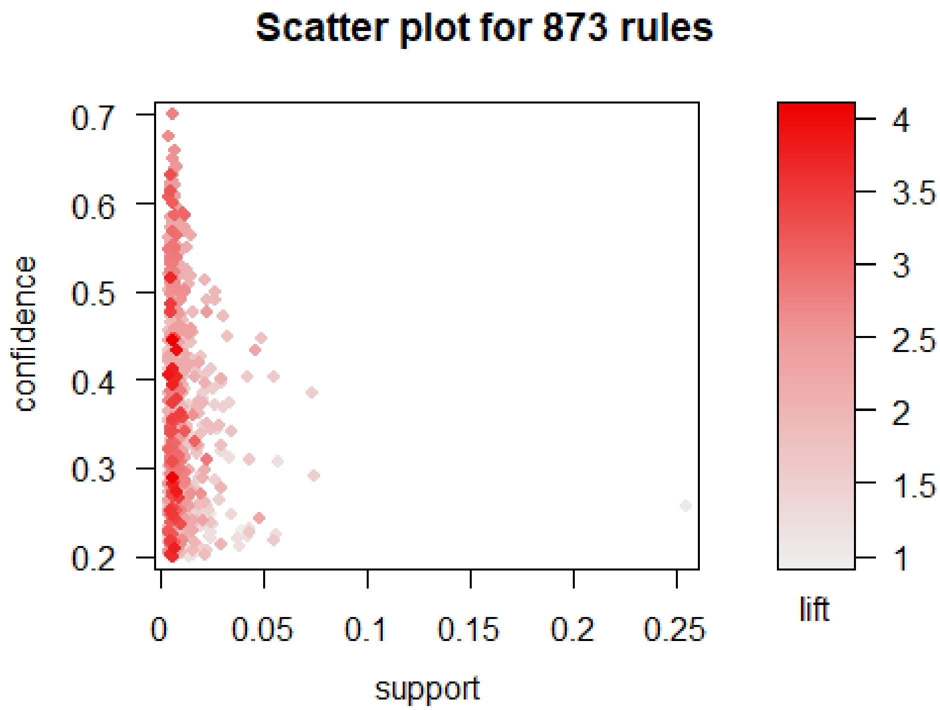
```
# Data frame
num_rules <- data.frame(rules_sup10, rules_sup5, rules_sup1, rules_sup0.5,
confidenceLevels)
library(ggplot2)
# Number of rules found with a support level of 10%, 5%, 1% and 0.5%
ggplot(data=num_rules, aes(x=confidenceLevels)) +

  # 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())
```



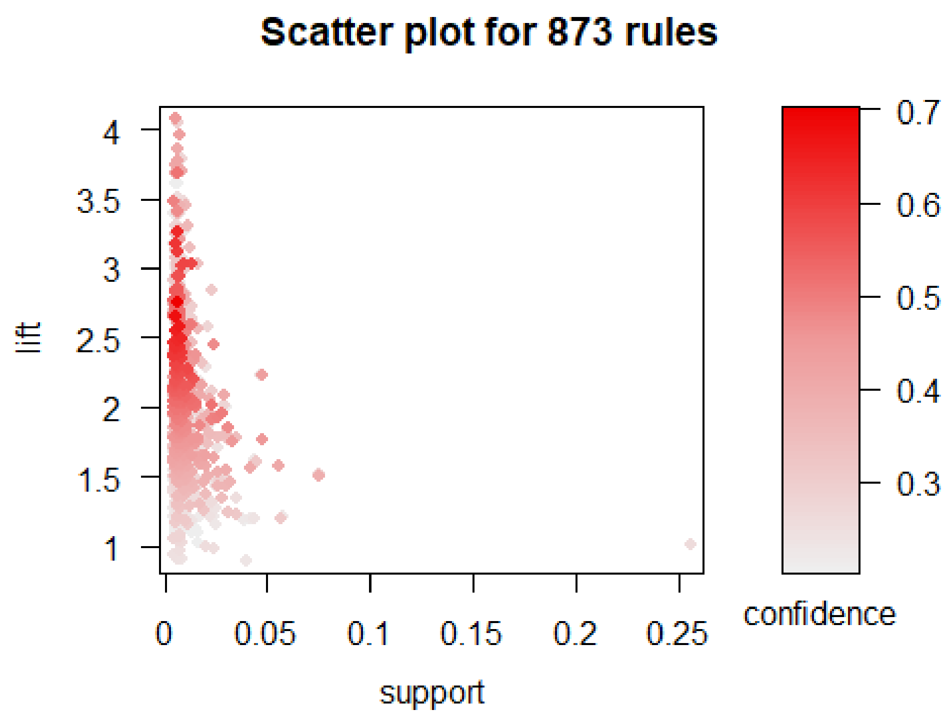
Using the thresholds specified, I got 873 rules. This quite high, but this is the number of rules I wanted in order to find the associate rules with higher lifts.

```
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"))
mbaskets <- mbaskets[,-2]
```

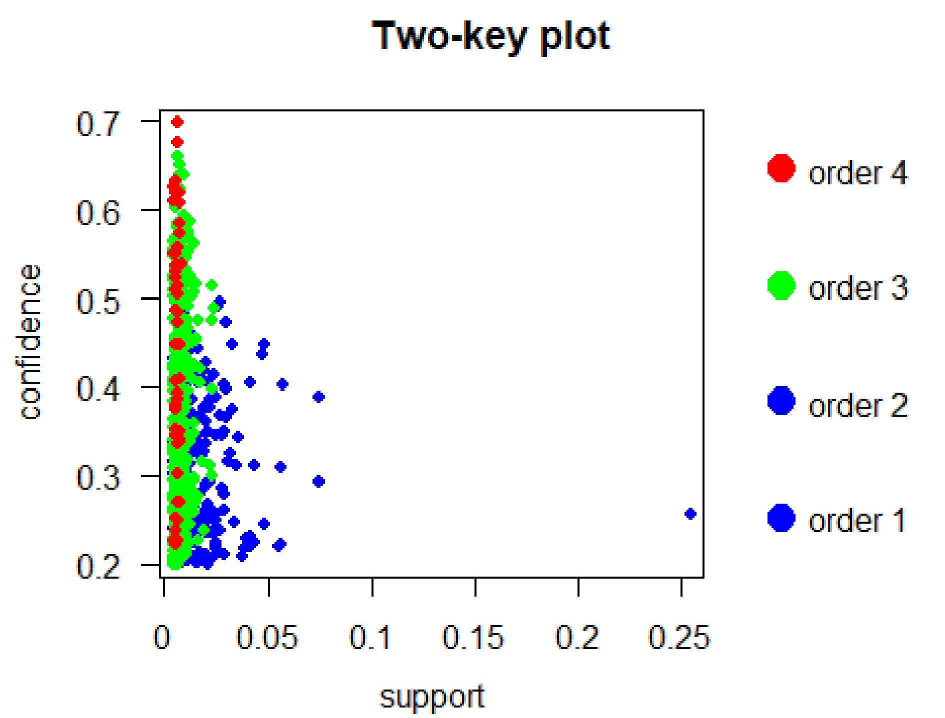
```r
mbaskets$Baskets = factor(mbaskets$Baskets )
mbaskets <- na.omit(mbaskets)


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)
plot(basketrules)
```




```r
## Choose a subset
#inspect(subset(basketrules, subset=lift > 3))
#inspect(subset(basketrules, subset=confidence > 0.5))
#inspect(subset(basketrules, subset=lift > 3 & confidence > 0.55))

# plot all the rules in (support, confidence) space
# notice that high lift rules tend to have low support
#plot(basketrules)

# can swap the axes and color scales
plot(basketrules, measure = c("support", "lift"), shading = "confidence")
```




```
# "two key" plot: coloring is by size (order) of item set
plot(basketrules, method='two-key plot')
```



```
# can now look at subsets driven by the plot
#inspect(subset(basketrules, support > 0.01))
#inspect(subset(basketrules, confidence > 0.2))
```

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.

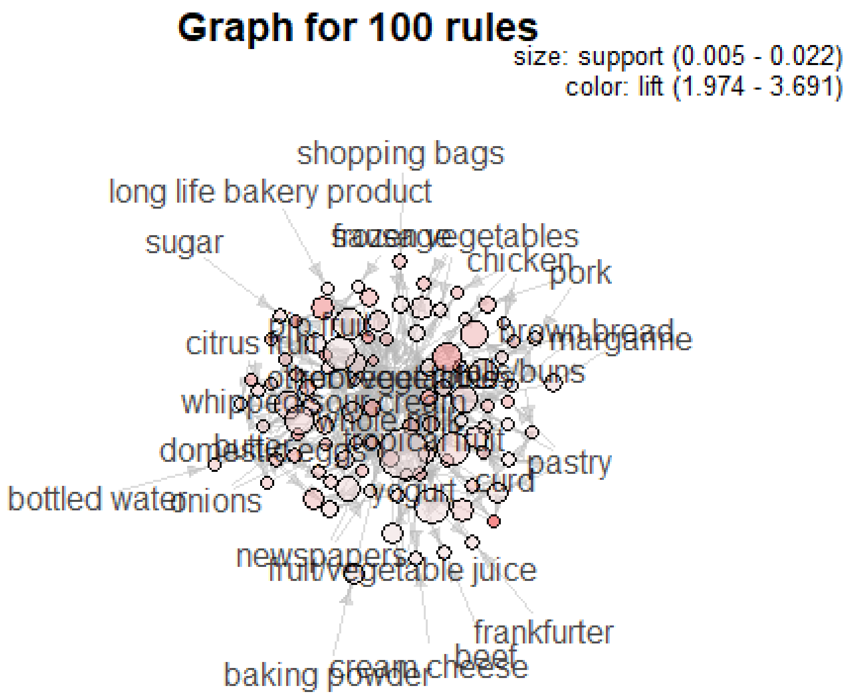
```
library(dplyr)
library(arules)
library(reshape)
library(tidyverse)
library(arules)  # has a big ecosystem of packages built around it
library(arulesViz)
baskets <-
read.delim("C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/groceries.tx
t",header = FALSE)

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)




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
```

```
plot(sub1, method='graph')
```




```
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
```

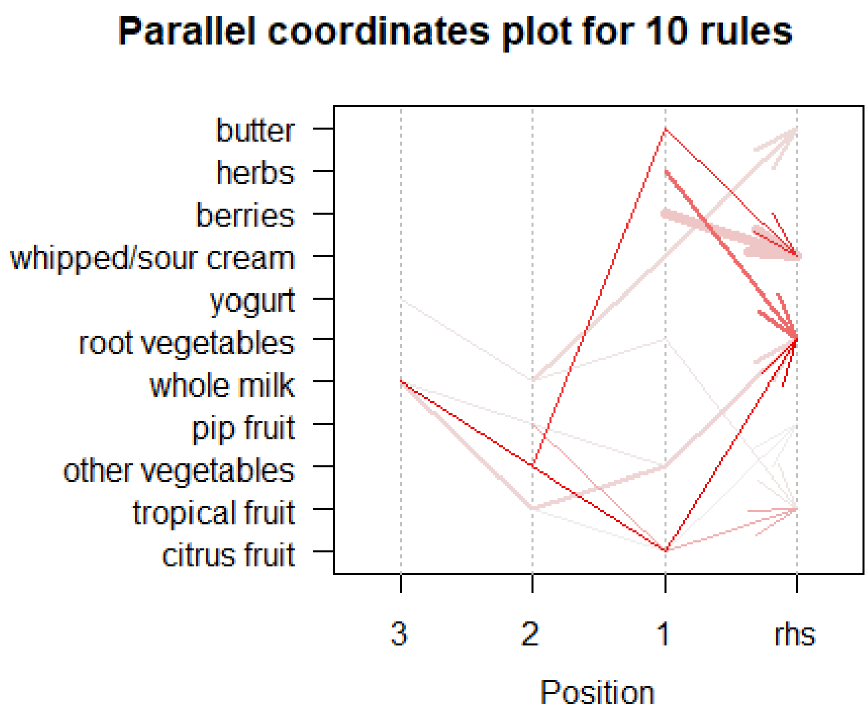
```
## [105] {root vegetables,
##        tropical fruit,
##        yogurt}                    => {whole milk}       0.005693950
0.7000000 0.008134215 2.739554    56
## [106] {other vegetables,
##        root vegetables,
##        tropical fruit}            => {whole milk}       0.007015760
0.5702479 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.

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

```r
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")
```

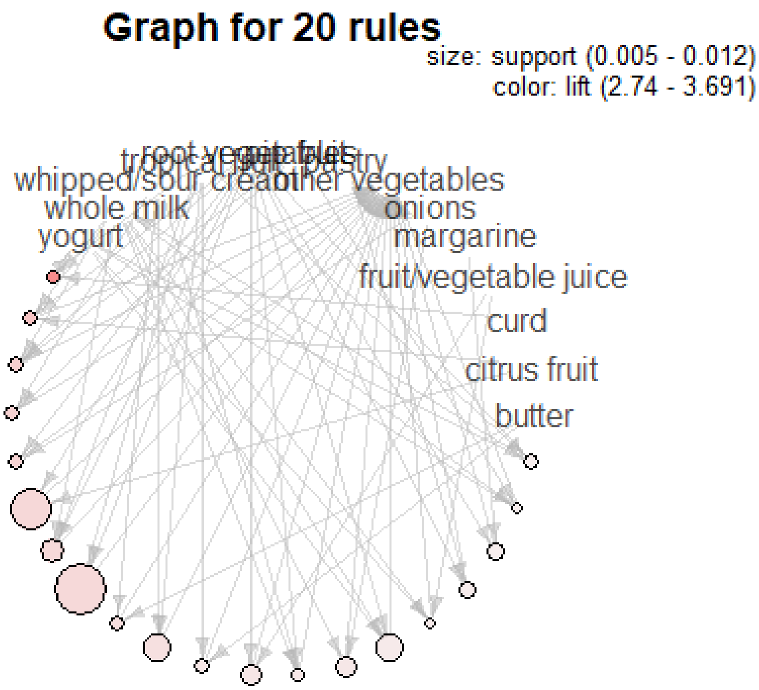
```r
subRules2 <- head(sort(basketrules, by="lift"), 10)
plot(subRules2, method="paracoord",reorder=TRUE)
```



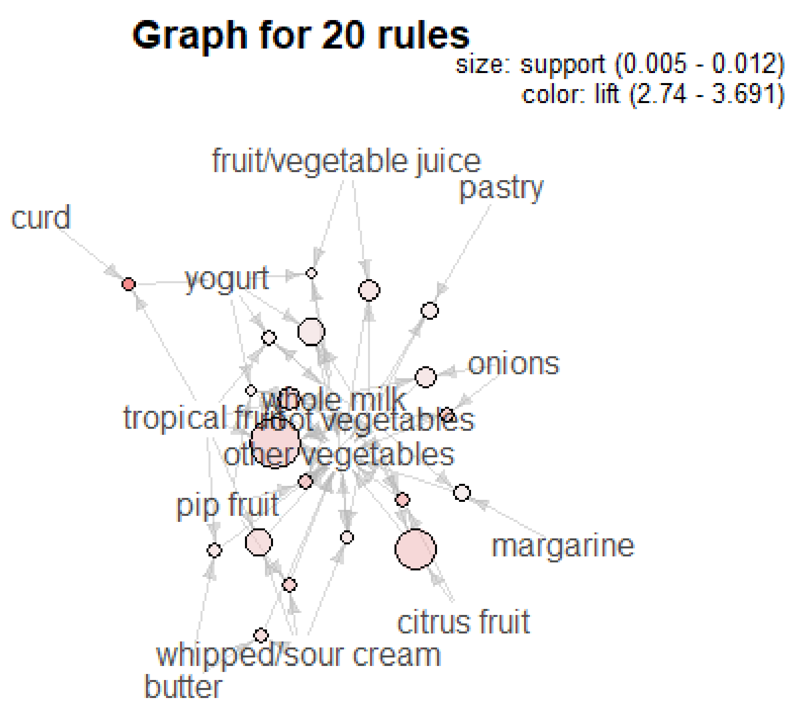
To get a better idea about the association rules, I plotted the top 20 rules based on lift in circular format. In addition, I also plotted the top 20 rules unformatted too as I believe that the different views provided an interesting perspective.

```r
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]
```

```r
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(head(sub1, 20, by='lift'), method='graph',
control=list(layout=igraph::in_circle()))
```




```
plot(head(sub1, 20, by='lift'), method='graph')
```



I also plotted the basket items that would usually not be purchased together along with an inspection of these association rules.

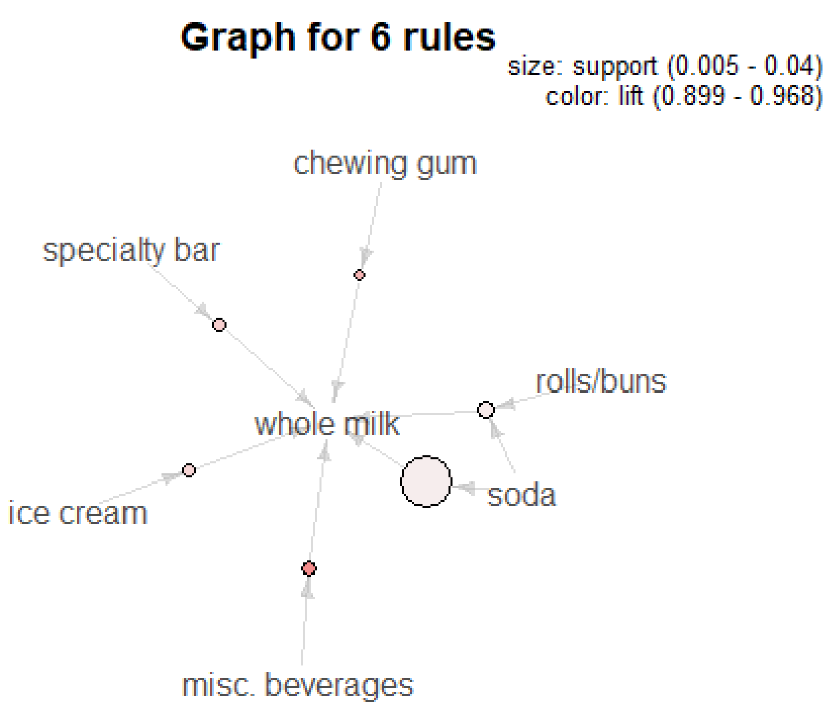
```
#lower end of the lift
library(dplyr)
library(arules)
library(reshape)
library(tidyverse)
library(arules)  # has a big ecosystem of packages built around it
library(arulesViz)
baskets <-
read.delim("C:/Users/machu/OneDrive/Documents/GitHub/STA380/data/groceries.tx
t",header = FALSE)

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"))
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(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
```

```
plot(tail(sort(basketrules, by = "lift")),method='graph')
```



Lastly, I plotted the larger, filtered network of basket items.