Data Science Capstone Milestone Report

Geoff Freedman

Overview and Introduction:

This milestone report is for the Johns Hopkins Data Science Captstone (https://www.coursera.org/course/dsscapstone) focusing on natural language processing. This report does the following:

* Downloads the english text data sets provided from news feeds, blogs and twitter.

* Proceseses the data and creates a sample set to analyze.

* Runs some summary statistics on the full English data set.

* Creates a text corpus that natural langugage processing code can use to mine the text data

* Runs some basic exploratory plots using word count and word pairing frequencies

* Talks about next steps

01 Data Processing

Once we've configured R to load in necessary software packages, the first step is to load in our text data:

The packages have been successfully loaded and we can begin:

```
# Load in the english versions of our text files:
englishBlogs <- readLines("../final/en_US/en_US.blogs.txt", encoding = "UTF-8", sk
ipNul=TRUE)
englishNews <- readLines("../final/en_US/en_US.news.txt", encoding = "UTF-8", skip
Nul=TRUE)
englishTwitter <- readLines("../final/en_US/en_US.twitter.txt", encoding = "UTF-8"
, skipNul=TRUE)

# Create an aggregated sample of all of our text data:
SAMPLE_SIZE = 10000
sampleTwitter <- englishTwitter[sample(1:length(englishTwitter),SAMPLE_SIZE)]
sampleNews <- englishNews[sample(1:length(englishNews),SAMPLE_SIZE)]
sampleBlogs <- englishBlogs[sample(1:length(englishBlogs),SAMPLE_SIZE)]
textSample <- c(sampleTwitter,sampleNews,sampleBlogs)

# Write the aggregated sample to a text file:
writeLines(textSample, "./sample/textSample.txt")
theSampleCon <- file("./sample/textSample.txt")
theSample <- readLines(theSampleCon)
close(theSampleCon)
```

Now that we have the text data safely loaded in we can start to generate some basic summary statistics such as word and line counts just to get a sense of what we're dealing with.

02 Basic Summmary Statistics:

```
# File Sizes:
englishTwitterSize <- round(file.info("../final/en_US/en_US.twitter.txt")$size / (
1024*1024),0)
englishNewsSize <- round(file.info("../final/en_US/en_US.news.txt")$size / (1024*1
024),0)
englishBlogsSize <- round(file.info("../final/en_US/en_US.blogs.txt")$size / (1024
*1024),0)
englishSampleFileSize <- round(file.info("./sample/textSample.txt")$size / (1024*1
024),0)

# Line Counts:
numEnglishTwitterLines <- countLines("../final/en_US/en_US.twitter.txt")[1]
numEnglishNewsLines <- countLines("../final/en_US/en_US.news.txt")[1]
numEnglishBlogsLines <- countLines("../final/en_US/en_US.blogs.txt")[1]
numEnglishSampleLines <- countLines("./sample/textSample.txt")[1]

# Word Counts:
numWordsEnglishTwitter <- as.numeric(system2("wc", args = "-w < ../final/en_US/en_
US.twitter.txt", stdout=TRUE))
numWordsEnglishNews <- as.numeric(system2("wc", args = "-w < ../final/en_US/en_US.
news.txt", stdout=TRUE))
numWordsEnglishBlog <- as.numeric(system2("wc", args = "-w < ../final/en_US/en_US.
blogs.txt", stdout=TRUE))
numWordsEnglishSample <- as.numeric(system2("wc", args = "-w < ./sample/textSampl
e.txt", stdout=TRUE))

# Aggregate the above into a data frame:
fileSummary <- data.frame(
  fileName = c("Blogs","News","Twitter", "Aggregated Sample"),
  fileSize = c(round(englishBlogsSize, digits = 2),
               round(englishNewsSize,digits = 2),
               round(englishTwitterSize, digits = 2),
               round(englishSampleFileSize, digits = 2)),
  lineCount = c(numEnglishBlogsLines, numEnglishNewsLines, numEnglishTwitterLines,
 numEnglishSampleLines),
  wordCount = c(numWordsEnglishBlog, numWordsEnglishNews, numWordsEnglishTwitter,
numWordsEnglishSample)
)
colnames(fileSummary) <- c("Name", "Size", "Num Lines", "Num Words")
fileSummary
```

```
##                Name Size Num Lines Num Words
## 1             Blogs  200    899288  37334690
## 2              News  196   1010242  34372720
## 3           Twitter  159   2360148  30374206
## 4 Aggregated Sample    5     30000    878639
```

You can see from the table above what the basic characteristics of our English data sets are.

03 Create An English Corpus From Our Sample:

Now we can take our sample text file and create a text corpus that our natural language processing code can analyze:

```
# Setup The Text Mining Class:
cname <- file.path(".", "sample")
finalCorpus <- Corpus(DirSource(cname))

# Convert corpus to lowercase:
finalCorpus <- tm_map(finalCorpus, content_transformer(tolower))

# Remove more transforms:
toSpace <- content_transformer(function(x, pattern) gsub(pattern, " ", x))
finalCorpus <- tm_map(finalCorpus, toSpace, "/|@|\\|")

# Remove punctuation:
finalCorpus <- tm_map(finalCorpus, removePunctuation)

# Remove numbers:
finalCorpus <- tm_map(finalCorpus, removeNumbers)

# Strip whitespace:
finalCorpus <- tm_map(finalCorpus, stripWhitespace)

# Remove english stop words:
finalCorpus <- tm_map(finalCorpus, removeWords, stopwords("english"))

# Initiate stemming:
finalCorpus <- tm_map(finalCorpus, stemDocument)
```

04 Create Our 'N-Grams' for Exploratory Data Analysis:

Next we create 'N-Grams'. An n-gram is a contiguous sequence of n items from a given sequence of text. You can read more about N-Grams here –> https://en.wikipedia.org/wiki/N-gram (https://en.wikipedia.org/wiki/N-gram).

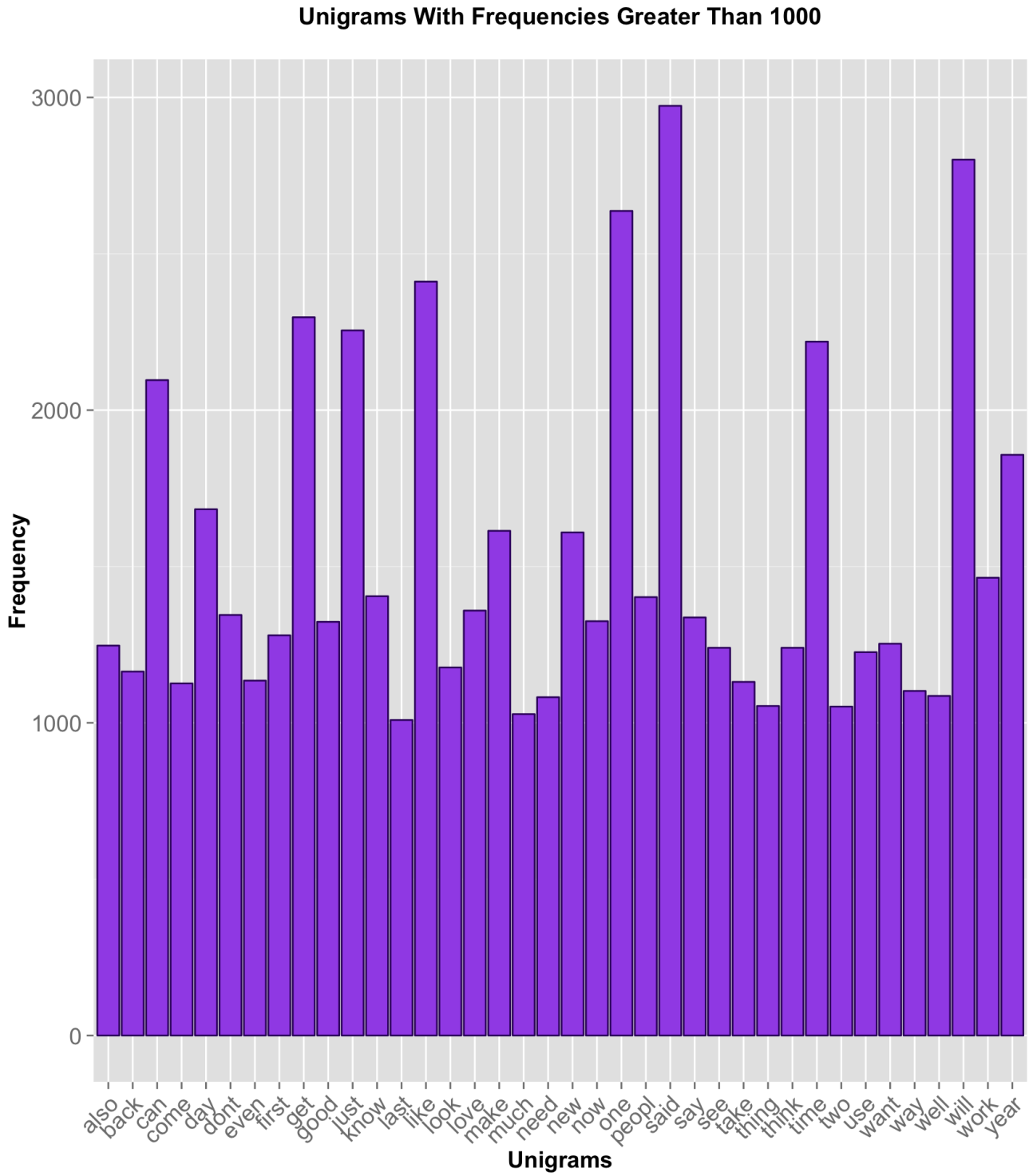
```
# Create a unigram:
unigramTokenizer <- function(x) NGramTokenizer(x, Weka_control(min = 1, max = 1))
unigram <- DocumentTermMatrix(finalCorpus, control = list(tokenize = unigramTokeni
zer))
unigramFreq <- sort(colSums(as.matrix(unigram)), decreasing=TRUE)
unigramWordFreq <- data.frame(word=names(unigramFreq), freq=unigramFreq)

# Create a bigram:
bigramTokenizer <- function(x) NGramTokenizer(x, Weka_control(min = 2, max = 2))
bigram <- DocumentTermMatrix(finalCorpus, control = list(tokenize = bigramTokenize
r))
bigramFreq <- sort(colSums(as.matrix(bigram)), decreasing=TRUE)
bigramWordFreq <- data.frame(word=names(bigramFreq), freq=bigramFreq)

# Create a trigram:
trigramTokenizer <- function(x) NGramTokenizer(x, Weka_control(min = 3, max = 3))
trigram <- DocumentTermMatrix(finalCorpus, control = list(tokenize = trigramTokeni
zer))
trigramFreq <- sort(colSums(as.matrix(trigram)), decreasing=TRUE)
trigramWordFreq <- data.frame(word=names(trigramFreq), freq=trigramFreq)
```

05 Exploratory Data Analysis:

Now we can create exploratory plots to get a sense of what our data looks like:

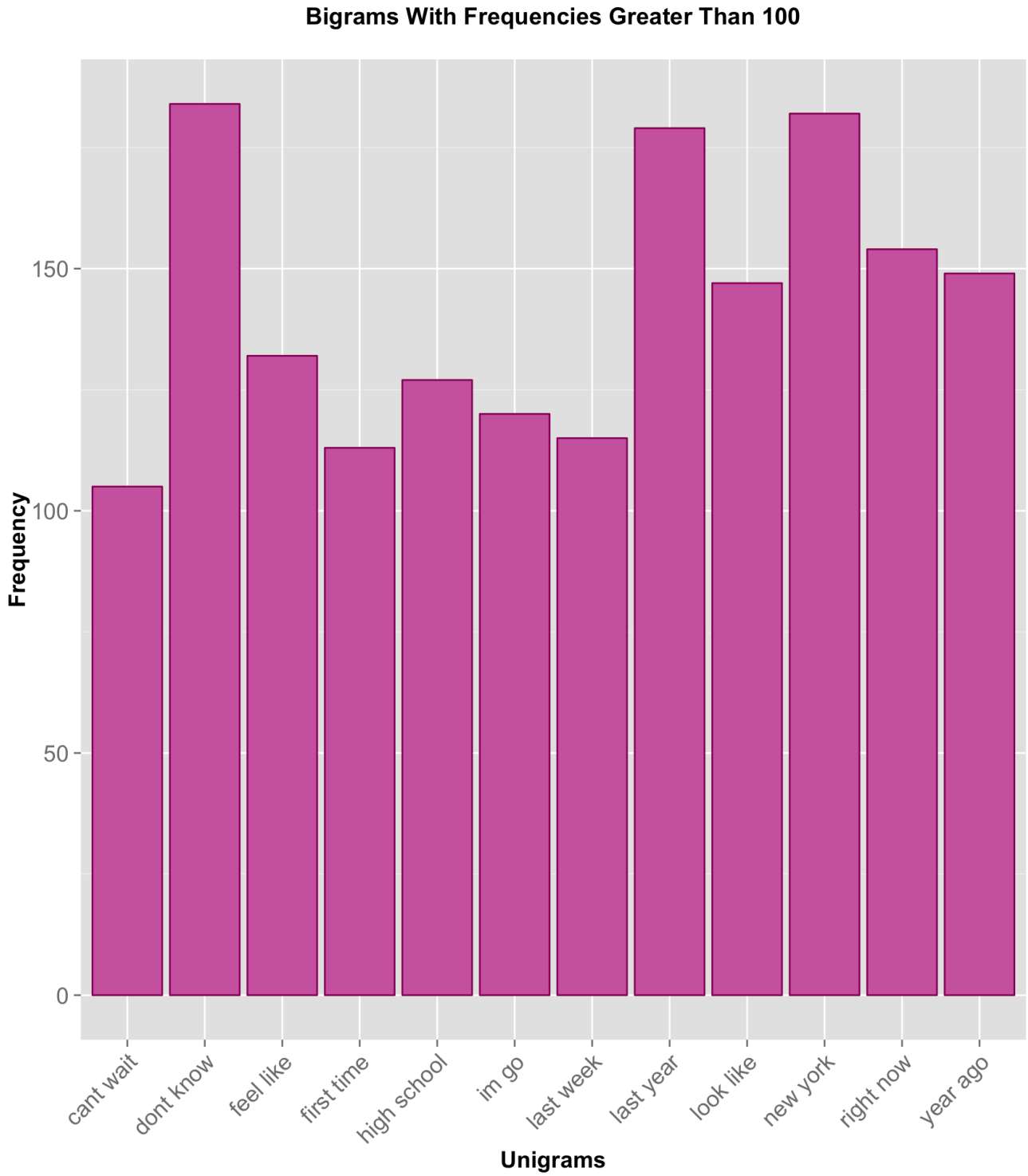
First we'll look at unigrams, which is just how many times do we see one word repeated in the text corpus:

```
# First Create A Plot of Our Unigrams:
unigramWordFreq %>% filter(freq > 1000) %>% ggplot(aes(word,freq)) +
  geom_bar(stat="identity", colour="#37006b", fill="#a257e9") +
  ggtitle("Unigrams With Frequencies Greater Than 1000") +
  xlab("Unigrams") + ylab("Frequency") +
  theme(axis.text.x=element_text(angle=45, hjust=1)) +
  theme(axis.text=element_text(size=14), axis.title=element_text(size=14,face="bol
d")) +
  theme(plot.title = element_text(lineheight=1.8, face="bold", vjust=3))
```



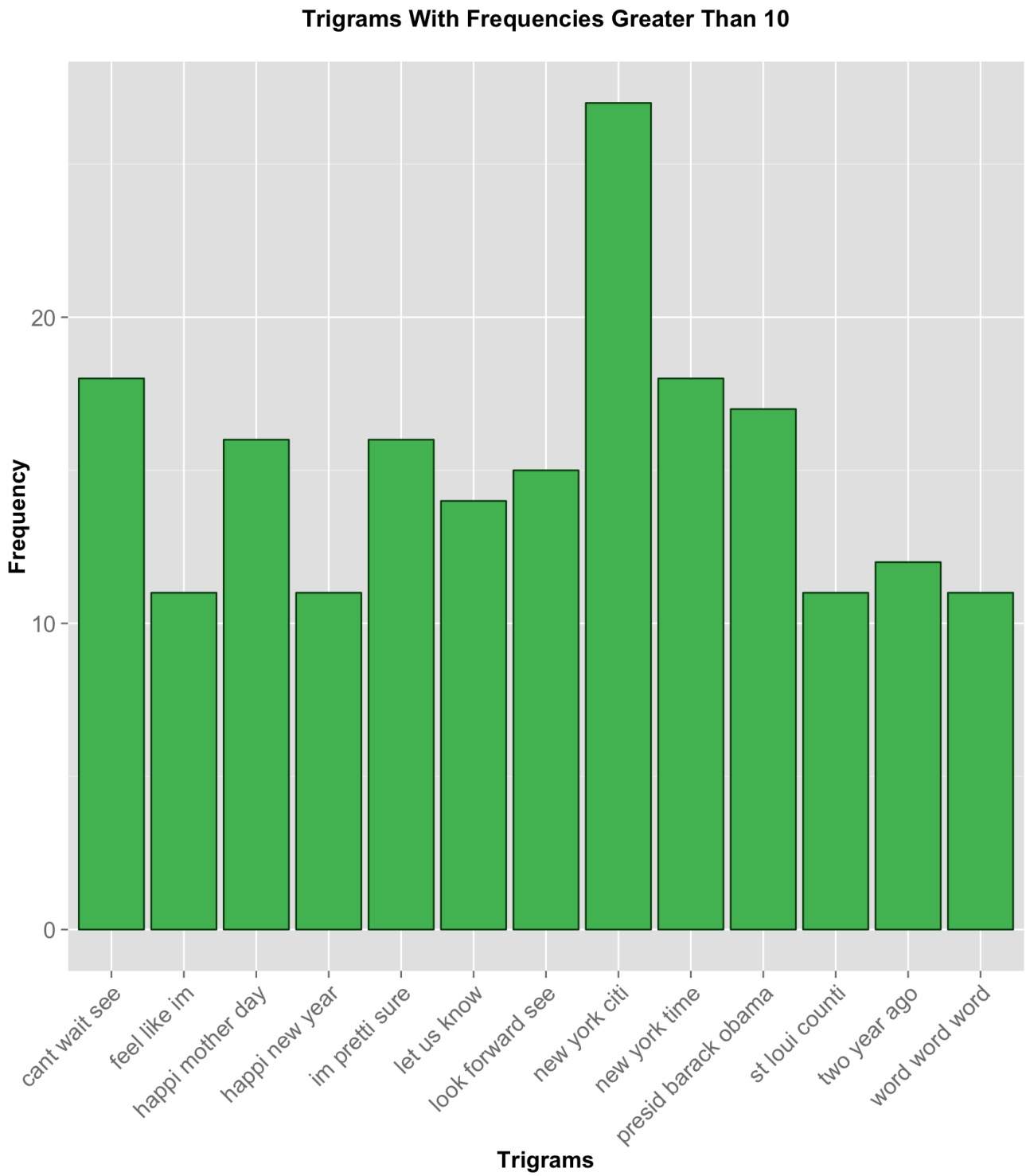
Second, we'll look at bigrams, which is a sequence of two words… this can be useful when predicting what a user will tap into their mobile phone next when writing something.

```
# Plot bigrams:
bigramWordFreq %>% filter(freq > 100) %>% ggplot(aes(word,freq)) +
  geom_bar(stat="identity", colour="#990068", fill="#cf6aaf") +
  ggtitle("Bigrams With Frequencies Greater Than 100") +
  xlab("Unigrams") + ylab("Frequency") +
  theme(axis.text.x=element_text(angle=45, hjust=1)) +
  theme(axis.text=element_text(size=14), axis.title=element_text(size=14,face="bol
d")) +
  theme(plot.title = element_text(lineheight=1.8, face="bold", vjust=3))
```



Third, we'll look at trigrams, which is a sequence of three words

```
# Plot trigrams:
trigramWordFreq %>% filter(freq > 10) %>% ggplot(aes(word,freq)) +
  geom_bar(stat="identity", colour="#00470d", fill="#4ebc63") +
  ggtitle("Trigrams With Frequencies Greater Than 10") +
  xlab("Trigrams") + ylab("Frequency") +
  theme(axis.text.x=element_text(angle=45, hjust=1)) +
  theme(axis.text=element_text(size=14), axis.title=element_text(size=14,face="bol
d")) +
  theme(plot.title = element_text(lineheight=1.8, face="bold", vjust=3))
```

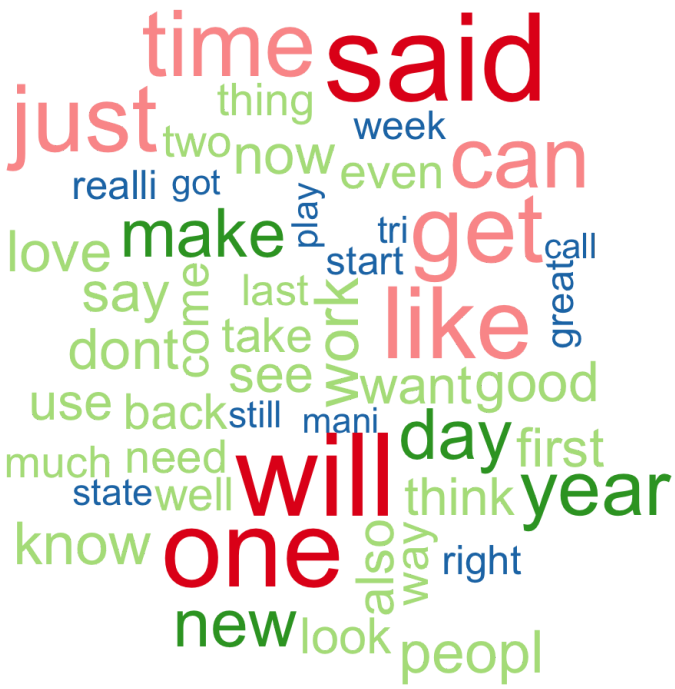


06 Exploratory Data Analysis Part 2:

Let's look at a word cloud plots of our n-grams.. you can read about word clouds here –> https://en.wikipedia.org/wiki/Tag_cloud (https://en.wikipedia.org/wiki/Tag_cloud)

Unigram Word Cloud Plot:

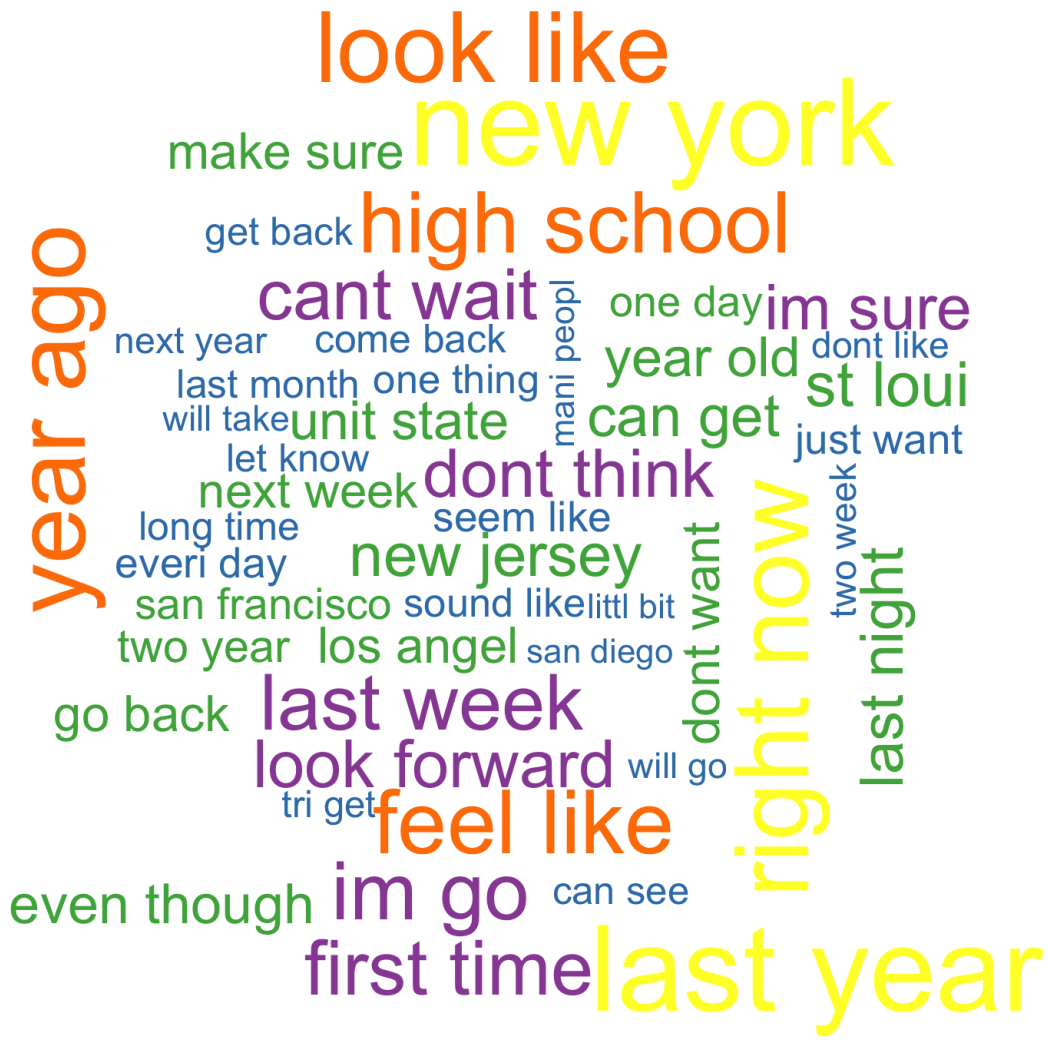
```
set.seed(1991)
wordcloud(names(unigramFreq), unigramFreq, max.words=50, scale=c(5, .1), colors=br
ewer.pal(6, "Paired"))
```



Bigram Word Cloud Plot:

```
wordcloud(names(bigramFreq), bigramFreq, max.words=50, scale=c(5, .1), colors=brew
er.pal(6, "Set1"))
```

```
## Warning in wordcloud(names(bigramFreq), bigramFreq, max.words = 50, scale =
## c(5, : dont know could not be fit on page. It will not be plotted.
```



Trigram Word Cloud Plot:

```
wordcloud(names(trigramFreq), trigramFreq, max.words=50, scale=c(5, .1), colors=br
ewer.pal(6, "Dark2"))
```

```
## Warning in wordcloud(names(trigramFreq), trigramFreq, max.words = 50, scale
## = c(5, : happi mother day could not be fit on page. It will not be plotted.
```

07 Conclusions and Next Steps:

The next step of is the creation a prediction application. I'll have to crate a proper prediction algorithm and ensure that it runs quickly for acceptable use as a web product. Also I might have to find a better way than just using N-Gram tokenization to predict the next word in a sequence of words. The upcoming Shiny app should ideally satisfy all of the above requirements.