Goals

Stated Directives

- Overview description of what you are going to be doing (2 points)
- Organization (2 points)
- At least 1 different analysis per person in your group regardless of the division of labour (2 points) Note the following are applied once per different analysis. "The Markdown document should include results of at least 1 different analysis per person in your group"
- Data cleaning / preparation decisions should be justified (2 points)
- Describe limitations and clarifications of decisions made that might impact your results (2 points)
- Results explained. If you have good results, give some interpretation. If method(s) attempted won't work explain what you think the problem might be. (2 points)

Approach

At our briefing with the Canada Revenue Agency (CRA) they indicated an interest in the breadth and depth of the insights collected by their survey. Not only were there general themes/topic but there were some very specific concerns raised. We show in this document how we tried to capture both the overarching story as well as very targeted morsels of information and then extract from that emotional context.

Chapter 1: General Cleaning

Section 1-1: Getting Survey Text

First we took the raw file provided and seperated out the text based answers to the survey.

```r
# Read in initial data
rawData <- read.xlsx(paste0(storePath,"Redacted FAB_Project_raw_data_Clean EXCEL Dec.23.xlsx"),1)

# Process columns
columnNamesStore <- names(rawData)

# Instantiate a data frame for cleaning
cleanData <- rawData
names(cleanData) <- paste("c",1:ncol(cleanData),sep="_")

# If you want to see the data
# View(cleanData)

# Bring out the non-caegorical columns (on manual read through)
nonCatColumns <- c(14,
                   22,
                   30,
                   40,
                   41,
                   43,
                   45,
                   46,
                   48)
```

```r
# Scrape free text columns out of the original data with some identifiers
procData <- cleanData[,c(1,2,nonCatColumns)]
newNames <- names(procData)
selectNames  <- newNames[c(3:length(newNames))]
```

Section 1-2: Translating to English

After obtaining the free text we then implemented a google API to provide a rough translation that was later verified manually.

```r
# Prepare for Translation
## Pivot the data to identify french text
pivtData <- procData %>%

  ### Select only those columns with the ID and the phrases for further processing
  select(c("c_1","c_2",selectNames)) %>%

  ### Pivot the data to be cataloged by ID and question index
  pivot_longer(all_of(selectNames),names_to = "column",values_to = "response")
```

```r
## Note: Using an external vector in selections is ambiguous.
## i Use `all_of(selectNames)` instead of `selectNames` to silence this message.
## i See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This message is displayed once per session.
```

```r
# Pick out french responses
forTranslation <- pivtData %>%

  # Take out french
  filter(c_2 == "FR")

# Count Characters to estimate cost ... under the free limit, yay!
counterChar <- 0
for(currentStr in forTranslation$response) {
  counterChar <- counterChar + str_length(currentStr)
}

# Translation functional block
## Commented out as running it too many time could mean $$$
## control + shift + C to activate / deactivate lines
if(!file.exists(paste0(storePath,"frenchToEnglish.rds"))){

  # Prepare dense feed for translation
  translationFeed <- forTranslation$response[!(forTranslation$response == "")]

  # Standardize frame for API input and translate the first response
  translationFrame <- gl_translate(
    t_string,
    target = "en",
    format = "text",
    source = "fr",
    model = "nmt"
```

```r
  )

  # Translate other responses
  for (i in 2:length(translationFeed)){

    ## 8 was excluded because there was a copy/paste of a large text segment (i.e., repetition)
    if(i!=8){
      translationEnFr <- gl_translate(
        translationFeed[i],
        target = "en",
        format = "text",
        source = "fr",
        model = "nmt"
      )
    }
    translationFrame <- rbind(translationFrame, translationEnFr)
  }

  # Save the feed
  saveRDS(translationFrame,
          paste0(storePath,"frenchToEnglish.rds"))
  write.xlsx(translationFrame, paste0(storePath,"frenchToEnglish.xlsx"))

} else {

  # Read the file if available
  translationFrame <- readRDS(paste0(storePath,"frenchToEnglish.rds"))
}

# Read in improved matrix if it is available (i.e., the manually verified translation)
if(file.exists(paste0(storePath,"frenchToEnglishM.xlsx"))){
  translationFrame <-
    readxl::read_xlsx(paste0(storePath,"frenchToEnglishM.xlsx"),1)
}
```

```
## New names:
## * `` -> ...1
```

```r
# Create master translated tibble that merges french and english
if(!file.exists(paste0(storePath,"masterResponse.xlsx"))){

  FrenchSegmentResponse = forTranslation %>%
    filter(response != "") %>%
    mutate(english = translationFrame$translatedText)

  FrenchSegmentNonResponse = forTranslation %>%
    filter(response == "") %>%
    mutate(english = "")

  EnglishTibble = pivtData %>%
    # Take out french
    filter(c_2 == "EN") %>%
    mutate(english = response)
```

3

```r
  # Master Response
  MasterResponse = rbind(
    FrenchSegmentResponse,
    FrenchSegmentNonResponse,
    EnglishTibble
  ) %>%
    arrange(c_1,c_2,column)

  write.xlsx(MasterResponse, paste0(storePath,"masterResponse.xlsx"))
} else {
  MasterResponse = readxl::read_xlsx(paste0(storePath,"masterResponse.xlsx"),1)
}
```

```
## New names:
## * '' -> ...1
```

Section 1-3: Verifying English Responses

In order to maximize the value of the data, spelling errors and non-standard language must be analyzed. Note that how this data is used will be described in subsequent sections. All that is discussed here is how various facets of the answers to the CRA questions were elucidated.

```r
# Complete analysis versus load
if(!file.exists(paste0(storePath,"initialsVerification.xlsx"))){
  # Note that the French translation had significant work processed manually
  # The following discussion is with regards to the as yet unprocessed English data
  # Prepare data for assessment
  forProcessEng <- pivtData %>%

    # Take out french
    filter(c_2 == "EN") %>%

    # Get the responses into words
    unnest_tokens(word, response) %>%

    # Select the words column
    select(c("word")) %>%

    # Get the unique words
    unique()

  # load the dictionary (one source / source of choice)
  # wordVector <- qdapDictionaries::DICTIONARY$word

  # Comprehensive source of English words including slang and variants
  ## e.g., stopped is a variant of stop
  ## https://github.com/dwyl/english-words
  wordfile <- read.csv(paste0(storePath,"words.txt"),sep="\n")
  wordsList <- tolower(wordfile$X2)

  # Detect if the isolate word appears in the English language resource that is loaded
  lengthProc <- nrow(forProcessEng)
```

```r
isWord <- rep(FALSE,lengthProc)
for(currentIndex in 1:nrow(forProcessEng)) {
  isWord[currentIndex] <- as.character(forProcessEng[currentIndex,1]) %in%
    wordsList
}

# Create patterns to find initialisms (i.e., words that contain more than 1 capital letter)
initialismPattern <- c(
  # Natural strings
  "\\b\\w+[[:upper:]]\\w+[[:upper:]]\\w+\\b",
  "\\b[[:upper:]]\\w+[[:upper:]]\\w+\\b",
  "\\b\\w+[[:upper:]]\\w+[[:upper:]]\\b",
  "\\b\\w+[[:upper:]][[:upper:]]\\w+\\b",
  "\\b[[:upper:]][[:upper:]]\\w+\\b",
  "\\b\\w+[[:upper:]][[:upper:]]\\b",
  "\\b[[:upper:]]\\w+[[:upper:]]\\b",
  "\\b[[:upper:]][[:upper:]]\\b",

  # Possessive strings
  "\\b\\w+[[:upper:]]\\w+[[:upper:]]\\w+\'s\\b",
  "\\b[[:upper:]]\\w+[[:upper:]]\\w+\'s\\b",
  "\\b\\w+[[:upper:]]\\w+[[:upper:]]\'s\\b",
  "\\b\\w+[[:upper:]][[:upper:]]\\w+\'s\\b",
  "\\b[[:upper:]][[:upper:]]\\w+\'s\\b",
  "\\b\\w+[[:upper:]][[:upper:]]\'s\\b",
  "\\b[[:upper:]]\\w+[[:upper:]]\'s\\b",
  "\\b[[:upper:]][[:upper:]]\'s\\b"
)

# Collapse the responses into one searchable string
responsesTogether <- paste(pivtData$response, collapse = "\n")

# Get the initialisms from this searchable string
listInitialisms <- unlist(str_extract_all(responsesTogether,initialismPattern)) %>%
  unique() %>%
  sort()

# For each initialism find where it was discovered
indexHold <- c()
respondsHold <- c()
for(initialismIndex in 1:length(listInitialisms)){
  currentResponses <- grep(listInitialisms[initialismIndex],pivtData$response)
  respondsHold <- c(respondsHold, currentResponses)
  indexHold <- c(indexHold, rep(initialismIndex,length(currentResponses)))
}

# Decode verification frame for manual review
verificationFrame <- data.frame(listInitialisms[indexHold],
                                pivtData$response[respondsHold])

# Write the verification from to an excel file for ease of viewing
write.xlsx(verificationFrame,paste0(storePath,"initialsVerification.xlsx"))
```

```r
} else {
  verificationFrame <- read.xlsx(paste0(storePath,"initialsVerification.xlsx"),1)
}

## Misspellings
# Prepare to adjust names for misspelling data frame
adjColNames <- function(dataFrameIn,newNamesIn) {
  colnames(dataFrameIn) <- newNamesIn
  return(dataFrameIn)
}

# Show misspelling findings
misSpelled <- data.frame(rbind(
  c("adminsitrave","administrative"),
  c("assesment","assessment"),
  c("back and forths","redundant communication"),
  c("beuracracy","bureaucracy"),
  c("carreer","career"),
  c("clickets","clicks"),
  c("Clients's","client's"),
  c("Cluster's","clusters"),
  c("collegues","colleagues"),
  c("collugies","colleagues"),
  c("communicatiuons","communications"),
  c("consistetnly","consistently"),
  c("constent","constant"),
  c("containg","containing"),
  c("coporate","corporate"),
  c("costings","costing"),
  c("curretn","current"),
  c("desking","desk"),
  c("eceptional","exceptional"),
  c("effrective","effective"),
  c("emapathy","empathy"),
  c("emial","email"),
  c("empath ","empathetic"),
  c("enthusiactic","enthusiastic"),
  c("excellente","excellent"),
  c("explaination","explanation"),
  c("finanace","finance"),
  c("gliches","glitches"),
  c("inforamtion","information"),
  c("inperson","in person"),
  c("interfereing","interfering"),
  c("intrical","integral"),
  c("leavning","leaving"),
  c("managment","management"),
  c("nintey percent","90percent"),
  c("particualry","particularly"),
  c("perfer","prefer"),
  c("persay",""), #just eliminate / extraneous
  c("pletntiful","plentiful"),
  c("Plexi Glass","plexiglass"),
```

```r
  c("positve","positive"),
  c("postions","positions"),
  c("prompty","promptly"),
  c("puticular","particular"),
  c("questons","questions"),
  c("refferences","references"),
  c("ressource","resource"),
  c("ressources","resources"),
  c("serie","series"), #alert
  c("sifficient","sufficient"),
  c("strenghts","strengths"),
  c("stylis","stylus"),
  c("THe","The"),
  c("timefram","time frame"),
  c("timeframe","time frame"),
  c("timeframes","time frames"),
  c("timeline","time line"),
  c("timelines","time lines"),
  c("unintentially","unintentionally"),
  c("unprecedent","unprecedented")
)) %>%
  adjColNames(.,c("error","corrected"))

## Words that added no meaning to the sentences upon reading
wordsForElimination <- rbind(
  c("\\bbuilding\'s\\b",""),
  c("\\bhttps://",""),
  c("\\b\\(IAR\\)\\b","")
)

## Words that may reasonably be subbed
subWords <- rbind(
  c("actioned","addressed"),
  c("admin ","assistant "),
  c("admins","assistants"),
  c("cell phone","phone"),
  c("cell phones","phones"),
  c("cellphone","phone"),
  c("cellphones","phones"),
  c("covid 19","covid"),
  c("covid-19","covid"),
  c("cross boarding","transferring"),
  c("depts.","departments"),
  c("doctor\'s","doctor"),
  c("e.g","like"),
  c("Floorplan","floor plan"),
  c("googling","researching"),
  c("how-to\'s","procedures"),
  c("i.e.","like"),
  c("i.e","like"),
  c("I.T","information technology"),
  c("IDEA","excel protocol"),
  c("importants","important information"),
```

```r
c("inbox","mailbox"),
c("iphone","phone"),
c("iphones","phones"),
c("IT ServiceDesk",""),
c("JIRA","application 1"),
c("Kahoot","application 3"),
c("Kantech","application 4"),
c("kinda","somewhat"),
c("KnowHow","application 5"),
c("leaving them hanging",""),
c("mailroom","mail room"),
c("mastercard","credit"),
c("Microsoft Office",""),
c("Microsoft Outlook",""),
c("microsoft team",""),
c("Microsoft Teams",""),
c("Microsoft teams",""),
c("Microsoft Vista",""),
c("MobiliKey",""),
c("MS","Microsoft"),
c("msteams",""),
c("na",""),
c("onboarding","initiating"),
c("OneNote",""),
c("PowerPivot",""),
c("powerpoint",""),
c("PowerPoint",""),
c("PowerQuery",""),
c("PPE","Personal Protective Equipment"),
c("Samsung smartphone","phone"), #usual reference is desire for iphone
c("Samsung","phone"),
c("smart phone","phone"),
c("smart phones","phones"),
c("smartphone","phone"),
c("smartphones","phones"),
c("SnagIT",""),
c("SnipIt",""),
c("staff's","subordinate's"),
c("telecom","telephone companies"),
c("telework","telecommuting"),
c("teleworking","telecommuting"),
c("thank you's","commendations"),
c("touchpoints","interactions"),
c("transferees within the organization","transferees"),
c("unknows","uncertainty"),
c("USERID","name"),
c("USERID's","names"),
c("videoconferencing","teleconferencing"),
c("webform","electronic form"),
c("webinars","internet seminars"),
c("What is the product number for XXX","What is the product number for this"),
c("widescreen","wider"),
c("WiFI","wireless internet access"),
```

```r
  c("WiFi","wireless internet access"),
  c("wifi","wireless internet access"),
  c("WIKI","application 2"),
  c("Wiki","application 2"),
  c("workplaces","work space")
)

# Store discoveries (i.e., these were the collective discoveries)
## Initialisms found
trueWords <- rbind(
  c("ABSB",""),
  c("ABSC",""),
  c("ABW",""),
  c("ACO",""),
  c("AEP",""),
  c("ALASD",""),
  c("AMA",""), # mega alert
  c("ATIPs",""),
  c("BECC",""),
  c("BGIS",""),
  c("BI",""), # alert
  c("BIQA",""),
  c("BLO",""), # alert
  c("BMC",""),
  c("CAPS",""),
  c("CAS",""),
  c("CERB",""),
  c("CESB",""),
  c("client reorgs",""),
  c("CNAS",""),
  c("CO",""), # Alert - find with "CO,"
  c("CoEs",""),
  c("COMSEC",""),
  c("CPB",""),
  c("CPB\'s",""),
  c("CPI",""),
  c("CPIs",""),
  c("CPSP",""),
  c("CSMD",""),
  c("CVB",""),
  c("DG","Director General"),
  c("DGFA",""), # French
  c("DGO",""),
  c("DGRH",""), # French
  c("DGs","Directors General"),
  c("DMC",""),
  c("DoF","Department of Finance"), #department of finance
  c("DSFA","Delegation of Spending and Financial Authority"),
  c("DTA",""),
  c("EA requests",""),
  c("EAP",""), #Careful as  appears in many words
  c("EBus",""),
  c("ECOTSO",""),
```

```r
c("EEs",""), #Careful as EEs
c("EFM",""),
c("EFMS",""),
c("EPS project (Synergy replacement)",""),
c("EPS projects",""),
c("EUR",""),
c("F&A",""),
c("FAB\'s",""),
c("FAMF",""),
c("FAMF",""),
c("FandA",""),
c("FAQ","Frequently Asked Quesions"),
c("FIs",""), #alert
c("FM"),
c("FMA",""),
c("FMA\'s",""),
c("FMAs",""),
c("FMAS",""),
c("FMASD","Financial Management & Advisory Services Directorate"),
c("FMASD\'s","Financial Management & Advisory Services Directorate's"),
c("FMS","Financial Management System"),
c("FORD program","TBS program for the development of Financial Officers FIs"),
c("FORD","TBS program for the development of Financial Officers FIs"),
c("FRAD",""),
c("FRAD\'s",""),
c("FTEs",""),
c("GC Docs",""),
c("GCSurplus",""),
c("GCWCC",""),
c("GLs",""),
c("GoC","Government of Canada"),
c("GS",""),
c("GS\'s",""),
c("HQ","Headquarters"),
c("HR","Human Resources"),
c("HRB",""),
c("IAFCD",""),
c("IAFCD",""),
c("IAM","Identity and Access Management"),
c("IAR",""),
c("IBC",""),
c("IBC\'s",""),
c("ICD",""),
c("ID offices",""),
c("ID","identification"),
c("ID\'s",""), #alert
c("IO",""),
c("IPRT",""),
c("ISD",""), #alert
c("ISS",""), # Alert
c("ITB",""),
c("ITB",""),
c("ITSS",""),
```

```r
  c("ITSSP",""),
  c("JV",""),
  c("JVs",""),
  c("KRP",""),
  c("KRP\'s",""),
  c("LR",""),
  c("MERKS",""),
  c("MG1",""),
  c("MG1\'s",""),
  c("MG2",""),
  c("MG2\'s",""),
  c("MG3",""),
  c("MG4",""),
  c("MIFI",""),
  c("ML3",""),
  c("MP",""),
  c("MTS",""),
  c("MyAccount",""),
  c("NCR",""),
  c("NFDC",""),
  c("NPSW",""),
  c("NPSW",""),
  c("OAG",""),
  c("OGD",""),
  c("OGD\'s",""),
  c("OGDs",""),
  c("OHS",""),
  c("OPIs",""),
  c("P3",""),
  c("P6",""),
  c("P7",""),
  c("PAB",""), #alert
  c("PB",""),
  c("PBF",""),
  c("PCCE",""),
  c("PMA",""),
  c("PMBOK",""),
  c("PMI",""),
  c("PMP",""),
  c("PO",""),
  c("PO\'s",""),
  c("PPSL",""),
  c("PRINCE2",""),
  c("PSP",""),
  c("PSPC",""),
  c("PSPC",""),
  c("PSS",""),
  c("PSSDSG",""),
  c("RARAD",""),
  c("RBA",""), #Alert
  c("RC02",""),
  c("RC02",""),
  c("RFAS",""),
```

```r
c("RH",""), #French
c("RI",""), #French
c("RL Security helpdesk",""),
c("RMC bootcamps",""),
c("RMC",""),
c("RMD","Resource Management Directorate"),
c("RP",""),
c("RP1","Tenant Request for work"),
c("RPA",""), #Alert
c("RPRD",""),
c("RPRD\'s",""),
c("RPSA",""),
c("RPSID","Real Property & Service Integration Directorate"),
c("RR","respendable revenue"),
c("RR","RR section for FMASD-CVB"),
c("RSCAD",""),
c("RTA",""),
c("SACO",""),
c("SAE",""), # French
c("SD agents","Service Desk Agents"),
c("ServiceDesk","Service Desk"),
c("SIAD","Security and Internal Affairs Directorate"),
c("SIR\'s",""),
c("SLA",""),
c("SOP","Standard Operating Procedure"),
c("SOW",""),
c("SP 02",""),
c("SP",""),
c("SP02",""),
c("SP05",""),
c("SP07",""),
c("SP2",""),
c("SP3",""),
c("SP5",""),
c("SP5s",""),
c("SPC",""),
c("SPS+",""),
c("SRA",""),
c("SRC",""),
c("SSB",""),
c("SSC",""),
c("SW",""),
c("TB",""), #Like TB used to have
c("TBS",""),
c("TC",""),
c("TETSO",""),
c("TETSO",""),
c("TI","Information Technology"), #French
c("TL",""),
c("TN-TSO","Toronto North TSO"),
c("TNTSO",""),
c("TOC","Transformation Oversight Committee"),
c("TSO",""),
```

```r
  c("TSO\'s",""),
  c("TSOS",""),
  c("TWTSO","west?"),
  c("USF",""), #French
  c("WFH",""),
  c("ZDFA_RPT","")
)

## Words that were capitalized for emphasis
emphasisWords <- rbind(
  c("\\(TRUE\\)",""),
  c("a BAD client service example",""),
  c("AD HOC","not formally planned"), # in a french translation
  c("Admin staff have been present DAILY","consistently worked every day"),
  c("Advise them to STOP IT",""),
  c("ALL the slack","extreme amounts of slack"),
  c("better service MY clients",""),
  c("Doing the job correctly THE FIRST TIME",""),
  c("Doing the job correctly THE FIRST TIME","providing quality work initially"),
  c("if I sent a request to our admin to order that equipment that SHE WOULD",""),
  c("Merci BEAUCOUP!","Thank you very much!"),
  c("My team and I take client service VERY seriously",""),
  c("THANK YOU",""),
  c("THANKFUL",""),
  c("The system is NOT being utilized in an efficient way",""),
  c("visited EVERY site regularly","consistently visited sites"),
  c("WE want to be the best place for a client","")
)

## Words that truly are english but were not detected as such
otherWords <- rbind(
  c("CFO's","Cheif Financial Officer's"),
  c("checkin's",""),
  c("commissionaires",""),
  c("ebizz",""),
  c("efax",""),
  c("emails",""),
  c("false",""),
  c("group's",""),
  c("infozone",""),
  c("InfoZone",""),
  c("LAN","local area network"),
  c("lockdown",""),
  c("lockdowns",""),
  c("majorly",""),
  c("MSteam",""),
  c("MSTeams",""),
  c("NA",""),
  c("NOTE","note"),
  c("OK","okay"),
  c("ON","Ontario"),
  c("onsite","on-site"),
  c("PC","personal computer"),
```

```r
  c("PC’s","personal computers"),
  c("PDF",""),
  c("PEI","Prince Edward Island"),
  c("proactively",""),
  c("proactiveness","proactive"),
  c("PTSD","Post Traumatic Stress Disorder"),
  c("pushback","objections"),
  c("RAM","Random Access Memory"),
  c("RCMP","Royal Canadian Mounted Police"),
  c("resourced","supplied"),
  c("respendable",""),
  c("Screensharing",""),
  c("St Catharines",""),
  c("stakeholders",""),
  c("TEAM","Microsoft Teams"),
  c("team’s",""),
  c("TEAMS","Microsoft Teams"),
  c("teleconferencing",""),
  c("voicemail",""),
  c("voip",""),
  c("VPN","Virtual Protective Network"),
  c("webcam",""),
  c("webex",""),
  c("website",""),
  c("wiki",""),
  c("WIKI",""),
  c("Winfast",""),
  c("WinFAST",""),
  c("WINFAST",""),
  c("workaround",""),
  c("workflow",""),
  c("workflows",""),
  c("worksite","on-site"),
  c("www.deepl.com","")
)

## Words that are english but are being used in a different way in the text
dualWords <- rbind(
  c("AC",""),
  c("AD","Assistant Director"),
  c("AD’s","Assistant Directors"),
  c("ADs",""),
  c("CRA","Canada Revenue Agency"),
  c("CRA\’s","Canada Revenue Agency’s"),
  c("FAB",""),
  c("FAD",""),
  c("FAM",""),
  c("FI",""),
  c("IT","Information Techology"),
  c("ITS",""),
  c("OR","Operating Revenue"),
  c("SAP",""),
  c("SIP","")
```

```
)
```

Chapter 2: Latent Semantic Analysis Related Inference

Section 2-1: Applying Cleaning Information

For this segment, the main elements of cleaning that are relevant pertain to spelling errors. These were replace to avoid any dilution of the subsequent word embedding. Initiallisms were left unchanged as they should remain unimpacted by stopping and stemming.

```r
# File path
questionsInFile <- paste0(storePath,"masterResponse.xlsx")

# Read the file
questionsData <- readxl::read_xlsx(questionsInFile,1)
```

```
## New names:
## * `` -> ...1
```

```r
# Discovered misspellings are used from misSpelled
# Change misSpellings into substitution ready patterns
p1 = misSpelled %>%
  mutate(error = paste0("\\b",error,"\\b"))

# Take away NA
noNAAnswers = questionsData %>%
  filter(!is.na(response))

# Replace misSpellings
for(indexAnswer in 1:dim(misSpelled)[1]) {
  noNAAnswers = noNAAnswers %>%
    mutate(english = str_replace_all(english,p1[indexAnswer,1],p1[indexAnswer,2]))
}

# Persist the results
xlsx::write.xlsx2(noNAAnswers,paste0(storePath,"noNonsense.xlsx"))

# Once we have adjusted the entences properly
sentenceTokenize = noNAAnswers %>%
  unnest_tokens(output = sentences, token="sentences", input = english) %>%
  select(c_1, column, sentences)

dePunct = sentenceTokenize %>%
  mutate(sentences = str_replace_all(sentences,"[[:punct:]]"," "))

# Number of questions
questionColumns = unique(questionsData$column)

# Filter to run on python
for(indexQuestion in 1:length(questionColumns)){
  currentPull = dePunct %>%
```

```r
    filter(column == questionColumns[indexQuestion]) %>%
    select(c_1,sentences) %>%
    data.frame()

  # Need to 0 this so it will play nicely with sklearn
  rownames(currentPull) <- as.numeric(rownames(currentPull))-1
  xlsx::write.xlsx2(currentPull, paste0(
    storePath,
    questionColumns[indexQuestion],
    ".xlsx"))
}
```

Section 2-2: Latent Semantic Analysis (LSA) Directing Word2Vec Models (Python Chunk)

First we considered LSA as a rapid alternative to Latent Direchlet Allocation (LDA). LSA provided a single value decomposition alternative to LDA. The concept of coherence was also useful in this case. Data analysis in this section began with removal of common English stop words. Sentences were then manually verified to ensure important terms were not removed. Stemming was then used to reduce words to their roots to reduce the effects of tense and voice on creating unnecessary divisions between effectively equivalent words. Given the restricted amount of data, use of a UMass algorithm was considered potentially superior to an arbitrary spit and training (i.e., test and training sets) and so this was implemented. We later look at performance of LDA. Note that to maximize the capacity for accurate extraction of topic number, a median of boot straps of topic number estimates was computed and used to develop the later models.

```python
# importing all necessary modules
## Data management
import pandas as pd

## Gensim Main
import gensim
from gensim.models import Word2Vec, KeyedVectors
from gensim.test.utils import common_texts

## Tokenizing
import nltk
from nltk.tokenize import sent_tokenize, word_tokenize, regexp_tokenize
import warnings

## Other
from nltk.corpus import stopwords
from gensim.test.utils import datapath
import re
import unicodedata
from tqdm import tqdm
import multiprocessing
import random
import xlrd
import openpyxl
from statistics import median

## More for LSA
```

16

```python
### Gensim
import os.path
from gensim import corpora
from gensim.models import LsiModel
from nltk.tokenize import RegexpTokenizer
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from gensim.models.coherencemodel import CoherenceModel

### sklearn
from sklearn import cluster, metrics
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD

### For plotting if required
## import matplotlib.pyplot as plt

## The following resource was used to direct further analysis
## https://www.datacamp.com/community/tutorials/discovering-hidden-topics-python

# Create functions
## For loading excel files
def load_excel(path,file_name):
    """
    Bring in an excel file
    """
    return pd.read_excel(os.path.join(path, file_name), index_col=0)

## For taking word units like paragraphs or sentences into word tokens
def process_tokens(input_text_units, target_column='sentences'):
    """
    Input: What ever division of data is desired, paragraphs or sentences
    Output: processed tokens for analysis
    """

    ## Tokenize
    ### https://medium.com/0xcode/tokenizing-words-and-sentences-using-nltk-in-python-a11e5d33c312
    processed_tokens = []

    ### Create a tokenizer unless word_tokenize is used
    #### tokenizer = RegexpTokenizer(r'\w+')

    ### Get usual english stop words
    eng_stop = set(stopwords.words('english'))

    ### Create a Stemmer if desired
    ## Stemming: https://tartarus.org/martin/PorterStemmer/
    p_stemmer = PorterStemmer()

    for i in input_text_units[target_column]:
        ### clean and tokenize document string
        ### lower case attribute required for stemmer
        raw = i.lower()
```

```python
        ### tokenizer
        tokens = word_tokenize(raw)

        ### remove stop words from tokens if desired
        stopped_tokens = [i for i in tokens if not i in eng_stop]

        ### stem tokens
        stemmed_tokens = [p_stemmer.stem(i) for i in stopped_tokens]

        ### add tokens to list
        processed_tokens.append(stemmed_tokens)

    return processed_tokens

## For taking word units like paragraphs or sentences into word tokens without frills
def reprocess_tokens(input_text_units, target_column='sentences'):
    """
    make just simple token lists
    """

    ## Tokenize
    processed_tokens = []

    for i in input_text_units[target_column]:
        ### clean and tokenize document string
        ### lower case attribute required for stemmer
        raw = i.lower()

        ### tokenizer
        tokens = word_tokenize(raw)

        ### put the tokens together
        ##linked_tokens = [i for i in tokens]

        ### add tokens to list
        processed_tokens.append(tokens)

    return processed_tokens

## detokenize for sklearn
### https://towardsdatascience.com/latent-semantic-analysis-deduce-the-hidden-topic-from-the-document-f
### https://scikit-learn.org/
def detokenize_for_sk(input_tokens):
    """
    takes the tokens back to mutated sentences
    """
    detokenized_text = []
    for i in range(len(input_tokens)):
        t = ' '.join(input_tokens[i])
        detokenized_text.append(t)
    return detokenized_text

## Create A Document Term Matrix
```

```python
def dictionary_DTM(clean_list):
    """
    Create the dictionary and Document Term Matrix (DTM)
    """
    # Create dictionary for courpus
    dictionary = corpora.Dictionary(clean_list)

    # Create Document Term Matrix using dictionary
    doc_term_matrix = [dictionary.doc2bow(doc) for doc in clean_list]

    # generate LDA model
    return dictionary,doc_term_matrix

## Create Latent Semantic Analysis Models
def create_lsa_model(clean_list,number_of_topics):
    """
    Create LSA from the input text given a number of topics and number of words associated with a topic
    """
    dictionary,DTM=dictionary_DTM(clean_list)

    # generate LSA model
    lsamodel = LsiModel(DTM, num_topics=number_of_topics, id2word = dictionary)
    #print(lsamodel.print_topics(num_topics=number_of_topics, num_words=words))
    return lsamodel

## Find Coherence
def get_coherence_for_set_DTM(dictionary, DTM, clean_list, stop, step=1, start=2):
    """
    find topic coherence and output models for use
    """

    # Initialize
    coherence_values = []
    model_list = []
    for num_topics in range(start, stop, step):

        # generate LSA model
        model = LsiModel(DTM, num_topics=num_topics, id2word = dictionary)

        # store the model
        model_list.append(model)

        # compute coherence
        ## Multiple coherence techniques to choose from:
        ### 'u_mass', 'c_v', 'c_uci', 'c_npmi'
        ## https://radimrehurek.com/gensim/models/coherencemodel.html
        ## https://mimno.infosci.cornell.edu/papers/mimno-semantic-emnlp.pdf
        ## https://www.aclweb.org/anthology/D12-1087.pdf
        ## Selected Umass because it is rapid and
        coherencemodel = CoherenceModel(model=model, texts=clean_list, dictionary=dictionary, coherence=

        # append coherence values
        coherence_values.append(coherencemodel.get_coherence())
```

```python
    return model_list, coherence_values


## Rep Modelling
def rep_coherence(dictionaryIn,DTMIn,tokensIn, num_iter = 10000):
    """
    find the average topic selection
    """
    coherence_lists = []
    for iter_num in range(num_iter):
        # print(iter_num)
        modelList, cohere = get_coherence_for_set_DTM(dictionaryIn,
                                                      DTMIn,
                                                      tokensIn,
                                                      10)
        max_value = max(cohere)
        max_index = cohere.index(max_value)
        coherence_lists.append(max_index)

    return median(coherence_lists)


# SK learn
## Reference
### https://towardsdatascience.com/latent-semantic-analysis-deduce-the-hidden-topic-from-the-document-f

## Obtain topics
def SVD_topic(dfInIt, numTopicsIn = 2):
    """
    return words and topics
    """
    ## Create topic vector / list
    topicHeadings = []
    for num_topics_ind in range(1, numTopicsIn + 1):
        topicHeadings.append("topic_" + str(num_topics_ind))

    ## Instantiate Vectorizer
    vectorizer = TfidfVectorizer(smooth_idf=True)

    ## Instantiate Single Value Decomposition Model
    svd_model_topic = TruncatedSVD(n_components=num_topics_ind, algorithm='randomized', n_iter=100, rand

    vectX = vectorizer.fit_transform(dfInIt['prep_sentences'])
    lsaX = svd_model_topic.fit_transform(vectX)

    topic_encoded_df = pd.DataFrame(lsaX, columns = topicHeadings)
    topic_encoded_df["documents"] = dfInIt['prep_sentences']
    topic_encoded_df["documents_raw"] = dfInIt['sentences']
    topic_encoded_df["identifier"] = dfInIt['c_1']
    dictionary = vectorizer.get_feature_names()

    # Note the transpose
    encoding_matrix = pd.DataFrame(svd_model_topic.components_, index = topicHeadings, columns = (dicti
    encoding_matrix["word"] = dictionary
```

```python
        return topic_encoded_df, encoding_matrix

# Word2Vec
## Create the general model
def create_sg_model(sentsIn, columnFocus = 'prep_sentences', num_iter = 100):
    """
    create skip gram models to find words commonly in the vacinity
    """
    # initiate model
    ## use skip gram model as we wish to take a focal word and predict its context
    modelX = Word2Vec(min_count=1, vector_size=50, workers=cores-1, window=5, sg=1, max_vocab_size=10000

    ## get the tokens / words
    tokIn = reprocess_tokens(sentsIn,columnFocus)

    ## build the vocabulary with the tokens
    modelX.build_vocab(tokIn, update = False)

    ## train the model
    modelX.train(tokIn,total_examples=modelX.corpus_count,epochs=num_iter)

    return modelX

## Word2Vec Clustering Metrics
def find_silhouettes(w2vModel):
    """
    uses skip gram models to find words commonly in the vacinity
    """
    ## Create Silhouettes Hold
    silhouettesOut = []
    clustersOut = []
    for num_clusters_ind in range(2, 11):

        # Set up kmeans model
        kmeansOut = cluster.KMeans(n_clusters=num_clusters_ind,
                                    random_state=12345)

        # Capture dictionary
        dictOut = w2vModel.wv.key_to_index.keys()

        # Cluster
        kmeansOut.fit(w2vModel.wv[dictOut])

        # Get the Silhouette
        silhouetteAveOut = metrics.silhouette_score(w2vModel.wv[dictOut],
                                        kmeansOut.labels_,
                                        metric='euclidean')

        # Store the data
        silhouettesOut.append(silhouetteAveOut)
        clustersOut.append(num_clusters_ind)

    # find the maximum silhouette
```

```python
        max_value = max(silhouettesOut)
        max_index = silhouettesOut.index(max_value)

        return clustersOut[max_index]

# Word2Vec Clustering Model
def create_Clusters(w2vModel,kClusters):
    """
    create cluster model
    """

    # Set up kmeans model
    kmeansOut = cluster.KMeans(n_clusters=kClusters,
                               random_state=12345)

    # Capture dictionary
    dictOut = w2vModel.wv.key_to_index.keys()

    # Get Embeddings
    embeddingsOut = w2vModel.wv[dictOut]

    # Cluster
    kmeansOut.fit(embeddingsOut)

    return kmeansOut.labels_, embeddingsOut, dictOut

# Suppress warnings
warnings.filterwarnings(action = 'ignore')

# General variables
## data path
data_path = "/Users/johnbrooks/Dropbox/R_files/Users/johnbrooks/Dropbox/Synced/R/STAT 5702/Store/"

## Use multiprocessing package to find the number of cores
cores= multiprocessing.cpu_count()

# Read in our data
c14df = load_excel(data_path,"c_14.xlsx")
c22df = load_excel(data_path,"c_22.xlsx")
c30df = load_excel(data_path,"c_30.xlsx")
c40df = load_excel(data_path,"c_40.xlsx")
c41df = load_excel(data_path,"c_41.xlsx")
c43df = load_excel(data_path,"c_43.xlsx")
c45df = load_excel(data_path,"c_45.xlsx")
c46df = load_excel(data_path,"c_46.xlsx")
c48df = load_excel(data_path,"c_48.xlsx")

# 1. Gensim Segment
## Segment variables
number_Iterations = 1

## First run
varInIt = c14df
```

```python
xOut = process_tokens(varInIt)
dOut,DTMOut = dictionary_DTM(xOut)
ml14, c14 = get_coherence_for_set_DTM(dOut,DTMOut,xOut,10)
c14df['prep_sentences'] = detokenize_for_sk(xOut)

## Bootstrap number of topics by recalculating coherence and taking median of bootstraps
### We add 2 because the index is returned
#### The indicies indicate the number of topic where index 0 = 2 topics, 1 = 3...
top14 = rep_coherence(dOut,DTMOut,xOut,number_Iterations) + 2
#print(top14)

varInIt = c22df
xOut = process_tokens(varInIt)
dOut,DTMOut = dictionary_DTM(xOut)
ml22, c22 = get_coherence_for_set_DTM(dOut,DTMOut,xOut,10)
c22df['prep_sentences'] = detokenize_for_sk(xOut)
top22 = rep_coherence(dOut,DTMOut,xOut,number_Iterations) + 2
#print(top22)

varInIt = c30df
xOut = process_tokens(varInIt)
dOut,DTMOut = dictionary_DTM(xOut)
ml30, c30 = get_coherence_for_set_DTM(dOut,DTMOut,xOut,10)
c30df['prep_sentences'] = detokenize_for_sk(xOut)
top30 = rep_coherence(dOut,DTMOut,xOut,number_Iterations) + 2
#print(top30)

varInIt = c40df
xOut = process_tokens(varInIt)
dOut,DTMOut = dictionary_DTM(xOut)
ml40, c40 = get_coherence_for_set_DTM(dOut,DTMOut,xOut,10)
c40df['prep_sentences'] = detokenize_for_sk(xOut)
top40 = rep_coherence(dOut,DTMOut,xOut,number_Iterations) + 2
#print(top40)

varInIt = c41df
xOut = process_tokens(varInIt)
dOut,DTMOut = dictionary_DTM(xOut)
ml41, c41 = get_coherence_for_set_DTM(dOut,DTMOut,xOut,10)
c41df['prep_sentences'] = detokenize_for_sk(xOut)
top41 = rep_coherence(dOut,DTMOut,xOut,number_Iterations) + 2
#print(top41)

varInIt = c43df
xOut = process_tokens(varInIt)
dOut,DTMOut = dictionary_DTM(xOut)
ml43, c43 = get_coherence_for_set_DTM(dOut,DTMOut,xOut,10)
c43df['prep_sentences'] = detokenize_for_sk(xOut)
top43 = rep_coherence(dOut,DTMOut,xOut,number_Iterations) + 2
#print(top43)

varInIt = c45df
xOut = process_tokens(varInIt)
```

```python
dOut,DTMOut = dictionary_DTM(xOut)
ml45, c45 = get_coherence_for_set_DTM(dOut,DTMOut,xOut,10)
c45df['prep_sentences'] = detokenize_for_sk(xOut)
top45 = rep_coherence(dOut,DTMOut,xOut,number_Iterations) + 2
#print(top45)

varInIt = c46df
xOut = process_tokens(varInIt)
dOut,DTMOut = dictionary_DTM(xOut)
ml46, c46 = get_coherence_for_set_DTM(dOut,DTMOut,xOut,10)
c46df['prep_sentences'] = detokenize_for_sk(xOut)
top46 = rep_coherence(dOut,DTMOut,xOut,number_Iterations) + 2
#print(top46)

varInIt = c48df
xOut = process_tokens(varInIt)
dOut,DTMOut = dictionary_DTM(xOut)
ml48, c48 = get_coherence_for_set_DTM(dOut,DTMOut,xOut,10)
c48df['prep_sentences'] = detokenize_for_sk(xOut)
top48 = rep_coherence(dOut,DTMOut,xOut,number_Iterations) + 2
#print(top48)

# 2. SK learn Segment
## Reference
### https://towardsdatascience.com/latent-semantic-analysis-deduce-the-hidden-topic-from-the-document-f

# After 10000 simulations we found that number of topics were 3 in question 10, 19, 22 and 30 but other

## Use single variable decomposition for the number of topics elucidated in the prior segment
te14, em14 = SVD_topic(c14df,3)
te22, em22 = SVD_topic(c22df)
te30, em30 = SVD_topic(c30df,3)
te40, em40 = SVD_topic(c40df,3)
te41, em41 = SVD_topic(c41df)
te43, em43 = SVD_topic(c43df)
te45, em45 = SVD_topic(c45df)
te46, em46 = SVD_topic(c46df)
te48, em48 = SVD_topic(c48df,3)

## Write out results
with pd.ExcelWriter(os.path.join(data_path, "wordsOut.xlsx")) as writer:
    em14.to_excel(writer, sheet_name='c_14')
    em22.to_excel(writer, sheet_name='c_22')
    em30.to_excel(writer, sheet_name='c_30')
    em40.to_excel(writer, sheet_name='c_40')
    em41.to_excel(writer, sheet_name='c_41')
    em43.to_excel(writer, sheet_name='c_43')
    em45.to_excel(writer, sheet_name='c_45')
    em46.to_excel(writer, sheet_name='c_46')
    em48.to_excel(writer, sheet_name='c_48')

with pd.ExcelWriter(os.path.join(data_path, "topicOut.xlsx")) as writer:
    te14.to_excel(writer, sheet_name='c_14')
```

```python
        te22.to_excel(writer, sheet_name='c_22')
        te30.to_excel(writer, sheet_name='c_30')
        te40.to_excel(writer, sheet_name='c_40')
        te41.to_excel(writer, sheet_name='c_41')
        te43.to_excel(writer, sheet_name='c_43')
        te45.to_excel(writer, sheet_name='c_45')
        te46.to_excel(writer, sheet_name='c_46')
        te48.to_excel(writer, sheet_name='c_48')

# 3. Word2Vec Segment
## Model the topic to find synonyms using LSA model insights
model14 = create_sg_model(c14df)
t14t1 = model14.wv.most_similar('work')[:10]
t14t2 = model14.wv.most_similar('train')[:10]
t14t3 = model14.wv.most_similar('tool')[:10]

model22 = create_sg_model(c22df)
t22t1 = model22.wv.most_similar('email')[:10]
t22t2 = model22.wv.most_similar('team')[:10]

model30 = create_sg_model(c30df)
t30t1 = model30.wv.most_similar('servic')[:10]
t30t2 = model30.wv.most_similar('burden')[:10]

model40 = create_sg_model(c40df)
t40t1 = model40.wv.most_similar('project')[:10]
t40t2 = model40.wv.most_similar('procur')[:10]
t40t3 = model40.wv.most_similar('fund')[:10]

model41 = create_sg_model(c41df)
t41t1 = model41.wv.most_similar('time')[:10]
t41t2 = model41.wv.most_similar('servic')[:10]

model43 = create_sg_model(c43df)
t43t1 = model43.wv.most_similar('time')[:10]
t43t2 = model43.wv.most_similar('respons')[:10]

model45 = create_sg_model(c45df)
t45t1 = model45.wv.most_similar('manag')[:10]
t45t2 = model45.wv.most_similar('time')[:10]

model46 = create_sg_model(c46df)
t46t1 = model46.wv.most_similar('home')[:10]
t46t2 = model46.wv.most_similar('provid')[:10]

model48 = create_sg_model(c48df)
t48t1 = model48.wv.most_similar('listen')[:10]
t48t2 = model48.wv.most_similar('feedback')[:10]
t48t3 = model48.wv.most_similar('client')[:10]

# note that as soon as the vocab is updated the corpus is updated
# model.build_vocab(tokenized_sents, update = False)
```

```python
## Clustering of Word 2 Vec Models
### https://ai.intelligentonlinetools.com/ml/k-means-clustering-example-word2vec/
### https://www.datanovia.com/en/lessons/determining-the-optimal-number-of-clusters-3-must-know-methods,
### https://scikit-learn.org/stable/modules/clustering.html

# Find the number of topics with a limit of 10 topics
topW14 = find_silhouettes(model14)
topW22 = find_silhouettes(model22)
topW30 = find_silhouettes(model30)
topW40 = find_silhouettes(model40)
topW41 = find_silhouettes(model41)
topW43 = find_silhouettes(model43)
topW45 = find_silhouettes(model45)
topW46 = find_silhouettes(model46)
topW48 = find_silhouettes(model48)


# Create the clustering labels
kO14, emb14, Dic14 = create_Clusters(model14,topW14)
kO22, emb22, Dic22 = create_Clusters(model22,topW22)
kO30, emb30, Dic30 = create_Clusters(model30,topW30)
kO40, emb40, Dic40 = create_Clusters(model40,topW40)
kO41, emb41, Dic41 = create_Clusters(model41,topW41)
kO43, emb43, Dic43 = create_Clusters(model43,topW43)
kO45, emb45, Dic45 = create_Clusters(model45,topW45)
kO46, emb46, Dic46 = create_Clusters(model46,topW46)
kO48, emb48, Dic48 = create_Clusters(model48,topW48)


# Make the embedings into frames
dfemb14 = pd.DataFrame(emb14)
dfemb14['dict'] = Dic14
dfemb14['labs'] = kO14

dfemb22 = pd.DataFrame(emb22)
dfemb22['dict'] = Dic22
dfemb22['labs'] = kO22

dfemb30 = pd.DataFrame(emb30)
dfemb30['dict'] = Dic30
dfemb30['labs'] = kO30

dfemb40 = pd.DataFrame(emb40)
dfemb40['dict'] = Dic40
dfemb40['labs'] = kO40

dfemb41 = pd.DataFrame(emb41)
dfemb41['dict'] = Dic41
dfemb41['labs'] = kO41

dfemb43 = pd.DataFrame(emb43)
dfemb43['dict'] = Dic43
dfemb43['labs'] = kO43

dfemb45 = pd.DataFrame(emb45)
```

```
dfemb45['dict'] = Dic45
dfemb45['labs'] = k045

dfemb46 = pd.DataFrame(emb46)
dfemb46['dict'] = Dic46
dfemb46['labs'] = k046

dfemb48 = pd.DataFrame(emb48)
dfemb48['dict'] = Dic48
dfemb48['labs'] = k048

# Integrate into r with reticulate
## https://rstudio.github.io/reticulate/articles/r_markdown.html
```

Here the topics are presented including their interpretation and top 10 associated words. This is followed by the use of a keyword that feeds into a word to vector model. The idea for this was that one could visualize the topic but then use the context surrounding a word to see what other words are similar in usage. Even if the number of words is small in the corpus overall, if the particular word / concept was consistently attached to the key word, sometimes even exclusively, this with would be highlighted. In essence the topic analysis gave themes and the word embedding analysis gave insights into particular concepts related to those topic. A sum of vectors to express a topic was also considered but not implemented presently.

```
# Read out python list variables for better presentation
feedTwoByTwo <- function(pythonList){
  newVector <- unlist(pythonList)
  indiciesVectO <- seq(1,length(newVector),2)
  indiciesVectE <- seq(2,length(newVector),2)
  return(data.frame(word = newVector[indiciesVectO],
                    similarity = newVector[indiciesVectE]))
}

# Draw the top 10 words associated with a select topic
drawTop10 <- function(dfInD,topicOfInterest = 1){
  orderVect <- order(dfInD[,topicOfInterest],decreasing = TRUE)
  return(dfInD$word[orderVect[1:10]])
}

# Question 10. Training / Tools Desired
## Topic Interpretation: working from home
drawTop10(py$em14,1)
```

```
##  [1] "train"   "monitor" "would"   "work"    "home"    "laptop"  "use"
##  [8] "second"  "tool"    "phone"
```

```
## Word tested: "work"
(feedTwoByTwo(py$t14t1))
```

```
##        word        similarity
## 1   quickli   0.59079122543335
## 2    longer   0.582170844078064
## 3      slow   0.579900801181793
## 4     point   0.577888190746307
```

27

```
## 5    struggl 0.573284387588501
## 6       lag 0.571965992450714
## 7    financ 0.571338415145874
## 8    almost 0.570468604564667
## 9      late 0.564323008060455
## 10 properli 0.560805022716522
```

```
## Topic Interpretation: training concerns - want formalized / online / at home
drawTop10(py$em14,2)
```

```
##  [1] "train"   "onlin"   "team"    "learn"   "tool"    "excel"   "new"
##  [8] "time"    "applic"  "softwar"
```

```
## Word tested: "train"
(feedTwoByTwo(py$t14t2))
```

```
##         word        similarity
## 1    advanc  0.62644761800766
## 2      self 0.615674674510956
## 3    mentor 0.615059733390808
## 4      basi 0.613614022731781
## 5    acrobat 0.602330386638641
## 6      redo 0.600631237030029
## 7   uniform 0.595509052276611
## 8     refer 0.594849169254303
## 9      path 0.593330085277557
## 10     adob 0.590692639350891
```

```
## Topic Interpretation: tools available / printer and scanner more than software
drawTop10(py$em14,3)
```

```
##  [1] "printer" "scanner" "train"   "offic"   "tool"    "onlin"   "import"
##  [8] "inform"  "portabl" "effici"
```

```
## Word tested: "tool"
(feedTwoByTwo(py$t14t3))
```

```
##          word        similarity
## 1    obsolet 0.667883574962616
## 2    winfast 0.652204871177673
## 3   forecast 0.587370216846466
## 4       none 0.581343472003937
## 5     compar 0.577169954776764
## 6    bringin 0.573075830936432
## 7       read 0.551555931568146
## 8      digit 0.539543867111206
## 9     welcom 0.535694718360901
## 10 classifi 0.530349433422089
```

```
# Question 17. When communicating with my clients
## Topic Interpretation: email as most important tool
drawTop10(py$em22,1)
```

```
##  [1] "email"    "phone"    "team"     "ms"       "combin"   "person"
##  [7] "winfast"  "depend"   "situat"   "telephon"
```

```
## Word tested: "email"
(feedTwoByTwo(py$t22t1))
```

```
##          word         similarity
## 1        pick 0.969659984111786
## 2     contact 0.968529462814331
## 3      prefer 0.963080644607544
## 4          go 0.963042676448822
## 5   afterward 0.961083173751831
## 6       extens  0.96104097366333
## 7       first   0.96052747964859
## 8      client 0.960240364074707
## 9        type 0.960174322128296
## 10      phone 0.960074782371521
```

```
## Topic Interpretation: team work (ms teams)
drawTop10(py$em22,2)
```

```
##  [1] "depend" "situat" "ms"     "team"   "method" "need"   "best"   "time"
##  [9] "sever"  "type"
```

```
## Word tested: "team"
(feedTwoByTwo(py$t22t2))
```

```
##          word         similarity
## 1        level 0.959421813488007
## 2      winfast  0.95481139421463
## 3      contact 0.953322052955627
## 4       extens 0.953035712242126
## 5         pick 0.952727496623993
## 6           go 0.950397908687592
## 7       person 0.949970722198486
## 8            e 0.949288427829742
## 9       servic 0.946495711803436
## 10       phone 0.946228444576263
```

```
#(feedTwoByTwo(py$t22t3))
```

```
# Question 19. Prevents client service
## Note that a negative result category (i.e., no response) was skipped
## Topic Interpretation: service quality
drawTop10(py$em30,2)
```

```
## [1] "servic"  "provid"  "except"  "client"  "noth"     "time"      "prevent"
## [8] "issu"    "custom"  "expect"
```

Word tested: "servic"
```
(feedTwoByTwo(py$t30t1))
```

```
##         word          similarity
## 1     except 0.967276692390442
## 2     provid 0.962245762348175
## 3      unabl 0.904540538787842
## 4       give 0.892934858798981
## 5       noth 0.892545938491821
## 6     client 0.885583579540253
## 7      think 0.875845670700073
## 8     burden 0.870618641376495
## 9       case 0.865028619766235
## 10   prevent 0.847844481468201
```

Topic Interpretation: impediments / burden of administration
```
drawTop10(py$em30,3)
```

```
## [1] "noth"     "prevent"  "except"   "custom"   "servic"    "give"
## [7] "burden"   "case"     "administr" "ext"
```

Word tested: "burden"
```
(feedTwoByTwo(py$t30t2))
```

```
##            word          similarity
## 1          case    0.996297955513
## 2      administr 0.979181289672852
## 3            us  0.972233712673187
## 4         staff 0.946190595626831
## 5          like 0.943098664283752
## 6         would 0.942600131034851
## 7    bureaucraci 0.937678694725037
## 8         anyth 0.936509490013123
## 9          give 0.934337615966797
## 10          say 0.934216201305389
```

```
#(feedTwoByTwo(py$t30t3))

# Question 22. Policy improvement
## Topic Interpretation: project management / projection process
drawTop10(py$em40,1)
```

```
## [1] "process" "project" "polici"  "manag"   "improv"  "need"     "time"
## [8] "work"    "client"  "could"
```

Word tested: "project"
```
(feedTwoByTwo(py$t40t1))
```

```
##          word           similarity
## 1        gap  0.73061215877533
## 2       rc02  0.727150976657867
## 3    handoff  0.688791632652283
## 4        eur  0.685973703861237
## 5    conduct  0.683545112609863
## 6      minor  0.679595112800598
## 7        40+  0.674368023872375
## 8   somewhat  0.671094000339508
## 9        rmc  0.669441342353821
## 10     oppos  0.668806970119476
```

```
## Topic Interpretation: procurement
drawTop10(py$em40,2)
```

```
##   [1] "process"   "procur"    "furnitur"  "project"    "cumbersom" "approv"
##   [7] "streamlin" "purchas"   "lengthi"   "hire"
```

```
## Word tested: "procur"
(feedTwoByTwo(py$t40t2))
```

```
##            word           similarity
## 1        asset  0.734770894050598
## 2           25  0.724853336811066
## 3        safeti  0.683444559574127
## 4        stream  0.681004166603088
## 5          dta  0.672903001308441
## 6        panel  0.661972343921661
## 7     telework  0.655548572540283
## 8           mt  0.650134325027466
## 9      useless  0.624921560287476
## 10   cumbersom  0.621451616287231
```

```
## Topic Interpretation: funding distribution
drawTop10(py$em40,3)
```

```
##   [1] "project" "fund"     "could"    "improv"  "report"  "branch"   "system"
##   [8] "financi" "manag"    "rmc"
```

```
## Word tested: "fund"
(feedTwoByTwo(py$t40t3))
```

```
##            word           similarity
## 1    agreement  0.803597927093506
## 2      earlier  0.801059603691101
## 3        fiscal   0.74801766872406
## 4       offset  0.746970057487488
## 5      strategi  0.736086964607239
## 6         solv  0.730539321899414
## 7      earmark  0.724034607410431
## 8        alloc  0.722932100296021
## 9      concurr  0.722701966762543
## 10         sip  0.718764543533325
```

```r
# Question 23. Successes
## Topic Interpretation: service / client responsibility excellence
drawTop10(py$em41,1)
```

```
##  [1] "client"  "servic"  "provid"  "team"    "work"    "need"    "time"
##  [8] "alway"   "excel"   "request"
```

```r
## Word tested: "time"
(feedTwoByTwo(py$t41t1))
```

```
##              word        similarity
## 1          relay 0.546276569366455
## 2          accur 0.545780658721924
## 3          relev 0.534783542156219
## 4           road 0.532711148262024
## 5          concis 0.531237483024597
## 6      constraint 0.521571278572083
## 7    unexpectedli 0.518473327159882
## 8         someon 0.516756296157837
## 9           away 0.514239132404327
## 10          save 0.510424733161926
```

```r
## Topic Interpretation: services provided
drawTop10(py$em41,2)
```

```
##  [1] "servic"  "provid"  "excel"   "team"    "secur"   "custom"  "support"
##  [8] "proud"   "work"    "employe"
```

```r
## Word tested: "servic"
(feedTwoByTwo(py$t41t2))
```

```
##            word        similarity
## 1        target 0.641144812107086
## 2          fist 0.617345213890076
## 3         tailor 0.617065489292145
## 4          rise 0.596190452575684
## 5         impecc 0.591826438903809
## 6         known 0.581665754318237
## 7        mindset  0.57266229391098
## 8        backbon 0.572591364383698
## 9       strongli 0.569387793540955
## 10           rp 0.569231033325195
```

```r
#(feedTwoByTwo(py$t41t3))

# Question 25. Compliments to respondant
## Topic Interpretation: time / timeliness is good
drawTop10(py$em43,1)
```

```
##  [1] "client"  "provid"  "time"    "servic"  "respons" "request" "need"
##  [8] "inform"  "quick"   "help"
```

```r
## Word tested: "time"
(feedTwoByTwo(py$t43t1))
```

```
##              word        similarity
## 1           rapid 0.702016651630402
## 2           audit 0.693287789821625
## 3         fashion 0.679997265338898
## 4           accur  0.6777423620224
## 5          speedi 0.671153604984283
## 6          extern 0.660579085350037
## 7    particularli 0.659950375556946
## 8           worri 0.653675496578217
## 9          manner 0.648744106292725
## 10           play 0.638804972171783
```

```r
## Topic Interpretation: fast and accurate communication / response
drawTop10(py$em43,2)
```

```
##  [1] "respons" "quick"   "accur"   "reliabl" "detail"  "simpl"   "time"
##  [8] "clear"   "prompt"  "written"
```

```r
## Word tested: "respons"
(feedTwoByTwo(py$t43t2))
```

```
##         word        similarity
## 1    resolut 0.802872598171234
## 2    written 0.777208209037781
## 3         ye 0.774729609489441
## 4   standard 0.769759595394135
## 5   although 0.757893919944763
## 6    concern 0.748212575912476
## 7     fulsom 0.742151975631714
## 8      rapid 0.730399608612061
## 9      quick 0.707601726055145
## 10     realm 0.700079619884491
```

```r
#(feedTwoByTwo(py$t43t3))

# Question 27. Services received by respondant
## Topic Interpretation: managers qualities / support
drawTop10(py$em45,1)
```

```
##  [1] "servic"  "time"    "respons" "provid"  "work"    "help"    "team"
##  [8] "manag"   "quick"   "need"
```

```r
## Word tested: "manag"
(feedTwoByTwo(py$t45t1))
```

```
##         word        similarity
## 1         ye 0.771929144859314
```

```
## 2        want 0.747161149978638
## 3        frad 0.723859310150146
## 4      inquir 0.717276155948639
## 5     insight 0.690407037734985
## 6          ad 0.677298367023468
## 7          co 0.671559810638428
## 8       rarad 0.669999480247498
## 9     honestli 0.661668121814728
## 10   knowledg 0.655730187892914
```

```
## Topic Interpretation: time / timeliness is good
drawTop10(py$em45,2)
```

```
##  [1] "respons" "quick"   "servic"  "easi"    "speed"   "fast"
##  [7] "mutual"  "assist"  "request" "reaction"
```

```
## Word tested: "time"
(feedTwoByTwo(py$t45t2))
```

```
##        word      similarity
## 1    achiev 0.658209919929504
## 2   comfort 0.652566313743591
## 3      6990 0.628233373165131
## 4    negoti 0.624539971351624
## 5    fulsom 0.621523320674896
## 6      wait 0.616708517074585
## 7      code  0.61565488576889
## 8   concern 0.612578809261322
## 9   fashion 0.603159129619598
## 10    queri 0.593327045440674
```

```
#(feedTwoByTwo(py$t45t3))
```

```
# Question 28. Impact of restrictions
## Topic Interpretation: working from home not impactful
drawTop10(py$em46,1)
```

```
##  [1] "servic" "provid" "client" "work"   "excel"  "home"   "covid"  "impact"
##  [9] "abil"   "offic"
```

```
## Word tested: "home"
(feedTwoByTwo(py$t46t1))
```

```
##        word       similarity
## 1       big 0.661360204219818
## 2       plu 0.611709952354431
## 3     frame 0.576369822025299
## 4   advantag  0.56815493106842
## 5      work 0.567934036254883
## 6     lucki 0.561885952949524
## 7   disposit 0.561749756336212
```

```
## 8   properli 0.561677694320679
## 9      separ  0.55406790971756
## 10      ride 0.546838939189911
```

```
## Topic Interpretation: what can be offered has not changed
drawTop10(py$em46,2)
```

```
##  [1] "servic"  "provid"  "excel"   "impact" "still"  "abil"     "client"
##  [8] "best"    "chang"   "continu"
```

```
## Word tested: "provid"
(feedTwoByTwo(py$t46t2))
```

```
##              word        similarity
## 1          excel 0.753388404846191
## 2       influenc 0.670783221721649
## 3    profession 0.665248155593872
## 4        percept 0.646396279335022
## 5        consequ 0.643276512622833
## 6      compromis 0.634297847747803
## 7            bad 0.633751153945923
## 8          servic 0.627605020999908
## 9           nice   0.6199911236763
## 10        impact 0.612680196762085
```

```
#(feedTwoByTwo(py$t46t3))
```

```
# Question 30. General Comments
## Topic Interpretation: appreciation / being listened to
drawTop10(py$em48,1)
```

```
##  [1] "thank"    "time"     "work"     "listen"  "much"     "comment"
##  [7] "merci"    "would"    "servic"   "consider"
```

```
## Word tested: "listen"
(feedTwoByTwo(py$t48t1))
```

```
##              word        similarity
## 1          merci 0.952804625034332
## 2          reach 0.949598610401154
## 3       consider 0.920319497585297
## 4          thank 0.896722733974457
## 5          check 0.866260528564453
## 6       feedback 0.859009027481079
## 7       beaucoup 0.826710045337677
## 8         produc   0.79055243730545
## 9         effort 0.742974519729614
## 10        formul 0.737739384174347
```

```
## Topic Interpretation: appreciation / comments of service and needs
drawTop10(py$em48,2)
```

```
##  [1] "work"    "servic"  "client"  "time"    "fab"     "employe" "home"
##  [8] "like"    "would"   "provid"
```

```
## Word tested: "feedback"
(feedTwoByTwo(py$t48t2))
```

```
##          word        similarity
## 1       check  0.98320609331131
## 2      gather 0.901162505149841
## 3    consider 0.864039778709412
## 4      listen 0.859008967876434
## 5       merci 0.857014656066895
## 6       reach 0.842521965503693
## 7         ask 0.820696949958801
## 8          us 0.785100519657135
## 9       choos 0.784457445144653
## 10   beaucoup 0.784049928188324
```

```
## Topic Interpretation: client experience unimpaired
drawTop10(py$em48,3)
```

```
##  [1] "time"     "noth"     "consider" "allevi"   "oversight" "unrealist"
##  [7] "take"     "gather"   "line"     "best"
```

```
## Word tested: "client"
(feedTwoByTwo(py$t48t3))
```

```
##          word        similarity
## 1    judgement 0.795697391033173
## 2       matter 0.765696167945862
## 3       manner 0.733185052871704
## 4      respond 0.728854119777679
## 5          guy 0.707869589328766
## 6          eye 0.707795858383179
## 7          120 0.697684109210968
## 8      certifi 0.686562955379486
## 9       financ 0.686520040035248
## 10      strive 0.684995293617249
```

Some of the word embedding projections were quite reasonable while others are hard to interpret. For example, in question 10 the "train" topic did identify the desire for formal online learning. The "tool" topic did capture that printer and scanner were among the most desirable additions for tools. Question 19 did indeed also have a clear link between "burden" of work and bureaucracy. Question 22 strongly identified projection processes as being unpleasant. Where it was not successful it may be that the select key term is not close enough to the focus of a topic / mentioned often making co-occurrence very difficult to ascertain with certainty. I had considered creating a combination vector to assess distance between term as an alternate topic representation although that is not presented in the current analysis.

```r
# emb14fd

#emb14fd[,-c(ncol(emb14fd)-1,ncol(emb14fd))]
# emb22fd
# emb30fd
# emb40fd
# emb41fd
# emb43fd
# emb45fd
# emb46fd
# emb48fd
#mnist_raw <- read_csv("https://pjreddie.com/media/files/mnist_train.csv", col_names = FALSE) #60,000 I
#TEST = sample(1:60000, size = 6000)

# Each row is a word
# mnist2use <- mnist_raw[TEST,-1] # randomly sample 6000 , remove the labels
# Mnistlabels = mnist_raw[TEST,1] # just the labels

tsneTheData <- function(embeddingFrame) {

  # Make the TSNE
  get.a.tsne = Rtsne(embeddingFrame[,-c(ncol(embeddingFrame)-1,ncol(embeddingFrame))],
                     dims = 2,
                     perplexity = 40)

  # Initialize frame
  outTsne <- data.frame(Dim1 = get.a.tsne$Y[,1],
                        Dim2 = get.a.tsne$Y[,2],
                        words = as.matrix(embeddingFrame[,ncol(embeddingFrame)-1]),
                        topics = as.matrix(embeddingFrame[,ncol(embeddingFrame)])

  )

  return(outTsne)
}

# Question 10. Training / Tools Desired
dfOfIntC <- emb14fd
currentTSNE <- tsneTheData(dfOfIntC)
indexMaster <- 1:nrow(dfOfIntC)

# Master plot
ggplot(currentTSNE, aes(x=Dim1, y=Dim2, colour=topics)) +
  geom_point() +
  theme(legend.title = element_blank(),
        plot.title = element_blank(),
        axis.title.x = element_blank(),
        axis.title.y = element_blank())
```
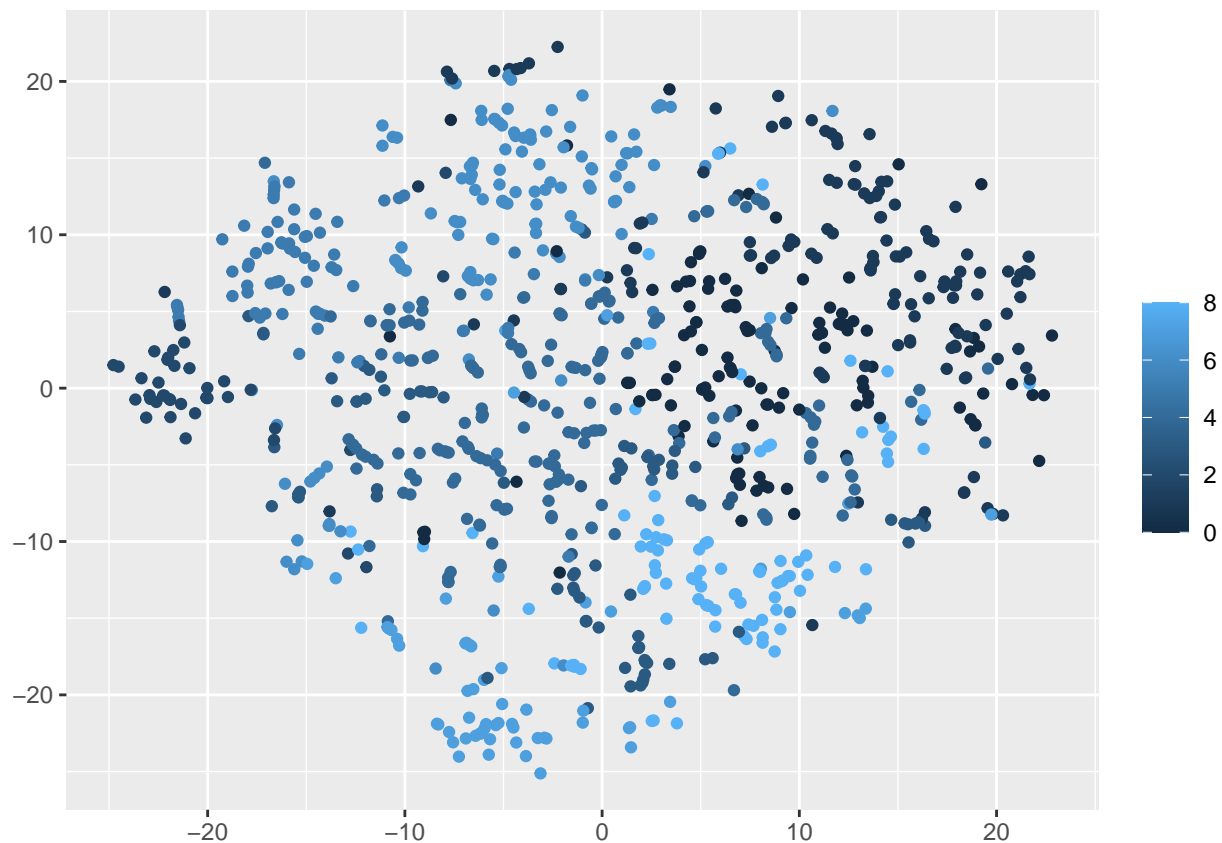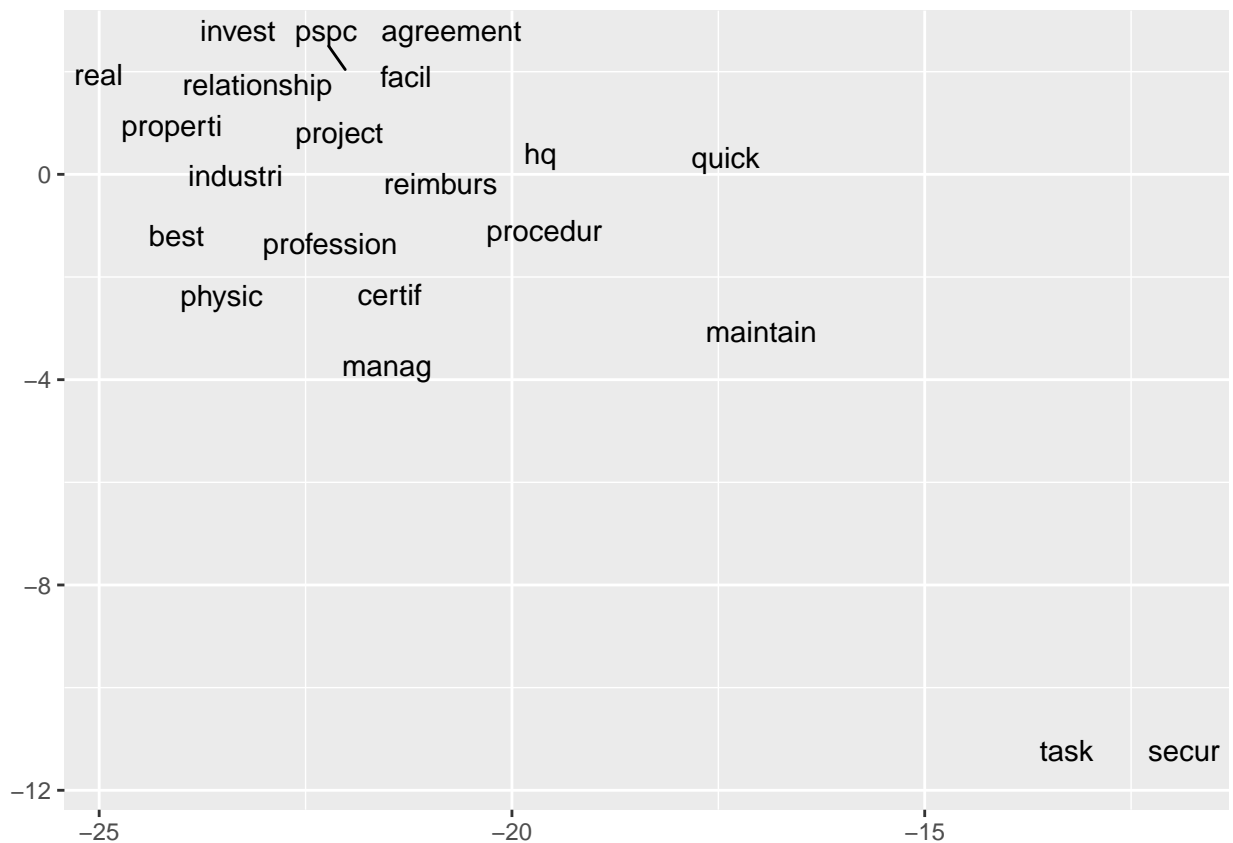
```
subsamplePlot <- unique(sample(indexMaster[dfOfIntC$labs == 0],40, replace = TRUE))
ggplot(currentTSNE[subsamplePlot,], aes(x=Dim1, y=Dim2, label=words)) +
  geom_text_repel() +
  theme(legend.title = element_blank(),
        plot.title = element_blank(),
        axis.title.x = element_blank(),
        axis.title.y = element_blank())
```
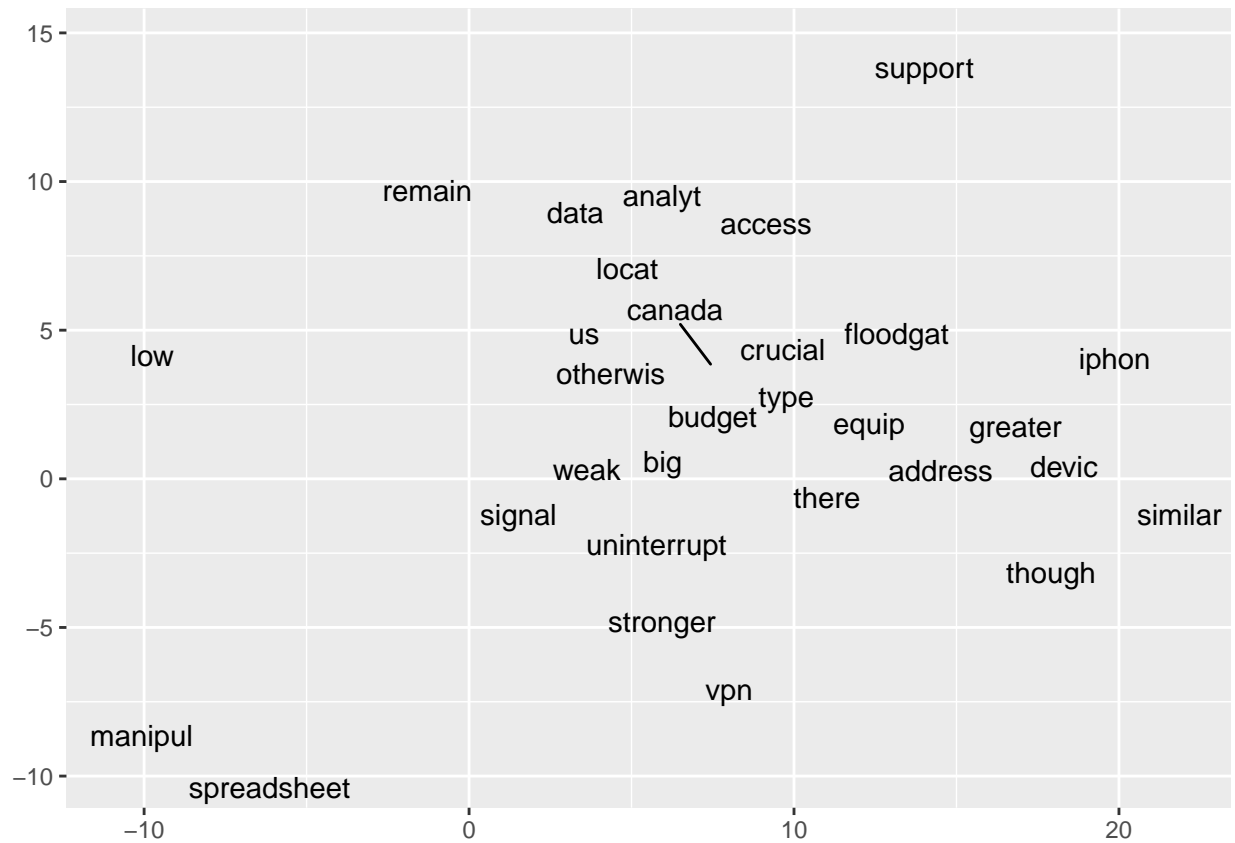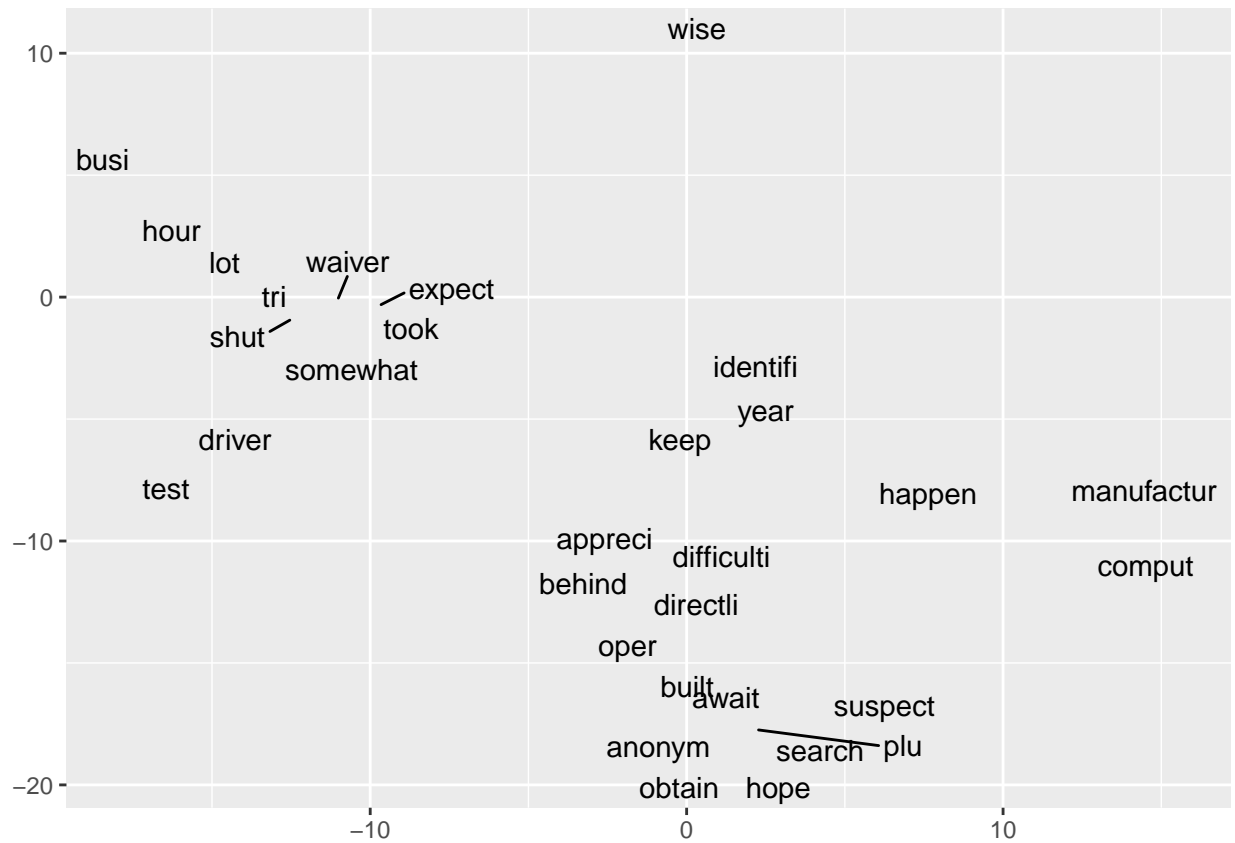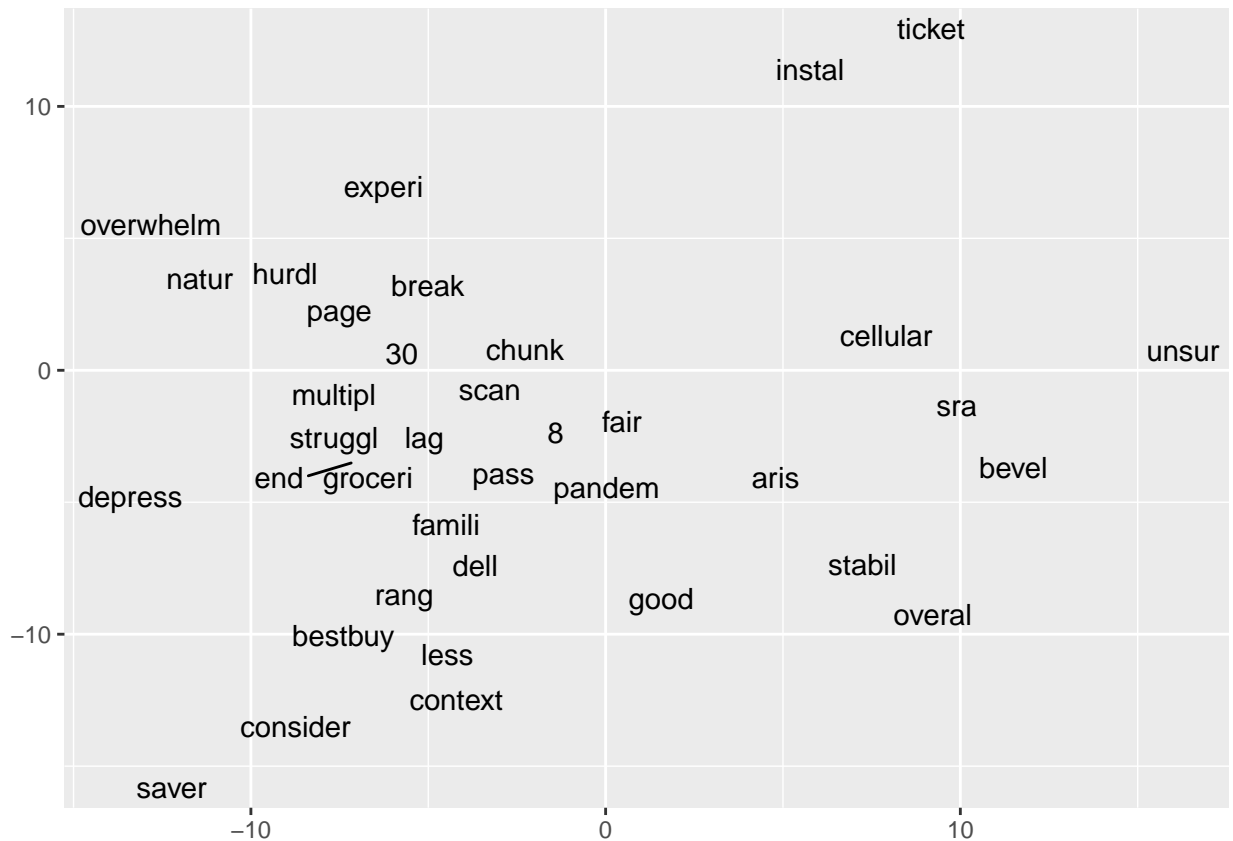
```
subsamplePlot <- unique(sample(indexMaster[dfOfIntC$labs == 1],40, replace = TRUE))
ggplot(currentTSNE[subsamplePlot,], aes(x=Dim1, y=Dim2, label=words)) +
  geom_text_repel() +
  theme(legend.title = element_blank(),
        plot.title = element_blank(),
        axis.title.x = element_blank(),
        axis.title.y = element_blank())
```
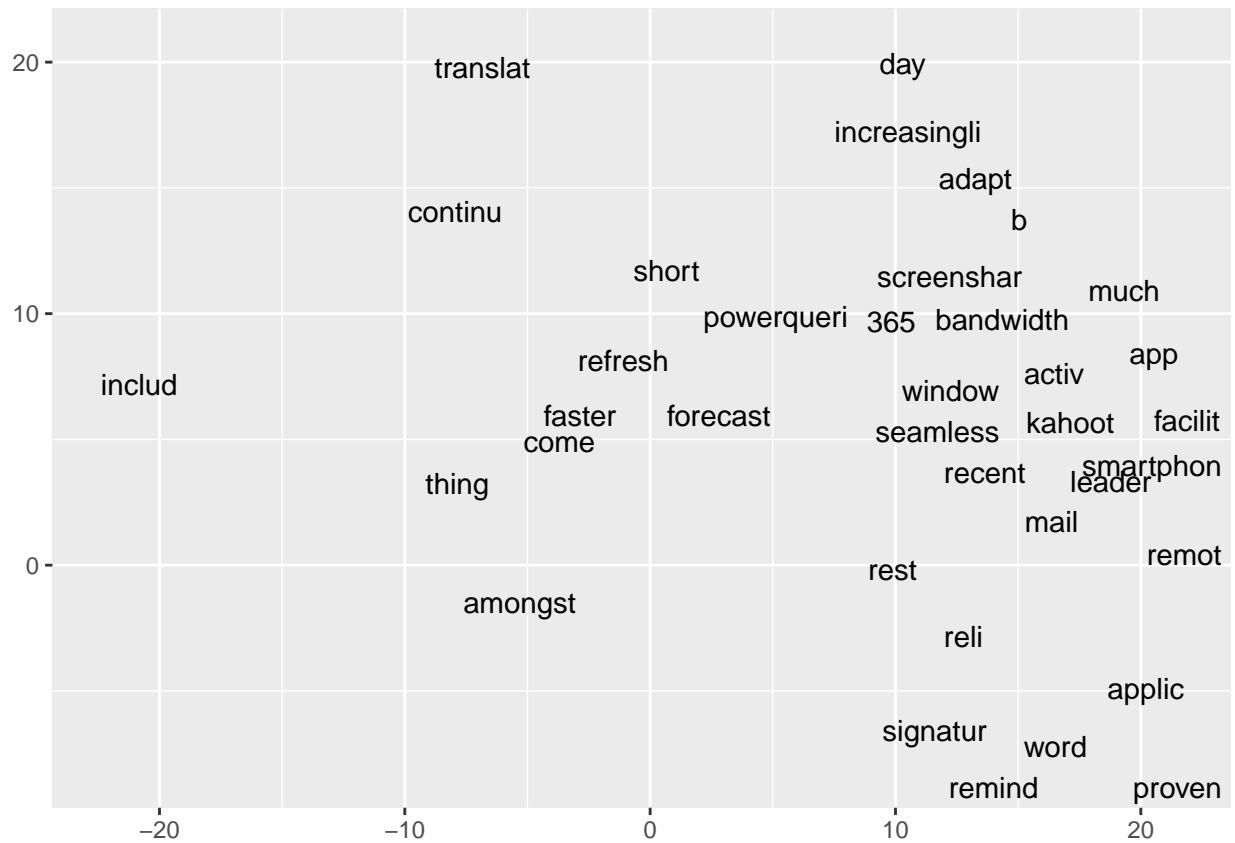
```
subsamplePlot <- unique(sample(indexMaster[dfOfIntC$labs == 2],40, replace = TRUE))
ggplot(currentTSNE[subsamplePlot,], aes(x=Dim1, y=Dim2, label=words)) +
  geom_text_repel() +
  theme(legend.title = element_blank(),
        plot.title = element_blank(),
        axis.title.x = element_blank(),
        axis.title.y = element_blank())
```

```
subsamplePlot <- unique(sample(indexMaster[dfOfIntC$labs == 3],40, replace = TRUE))
ggplot(currentTSNE[subsamplePlot,], aes(x=Dim1, y=Dim2, label=words)) +
  geom_text_repel() +
  theme(legend.title = element_blank(),
        plot.title = element_blank(),
        axis.title.x = element_blank(),
        axis.title.y = element_blank())
```

```
## Warning: ggrepel: 2 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```
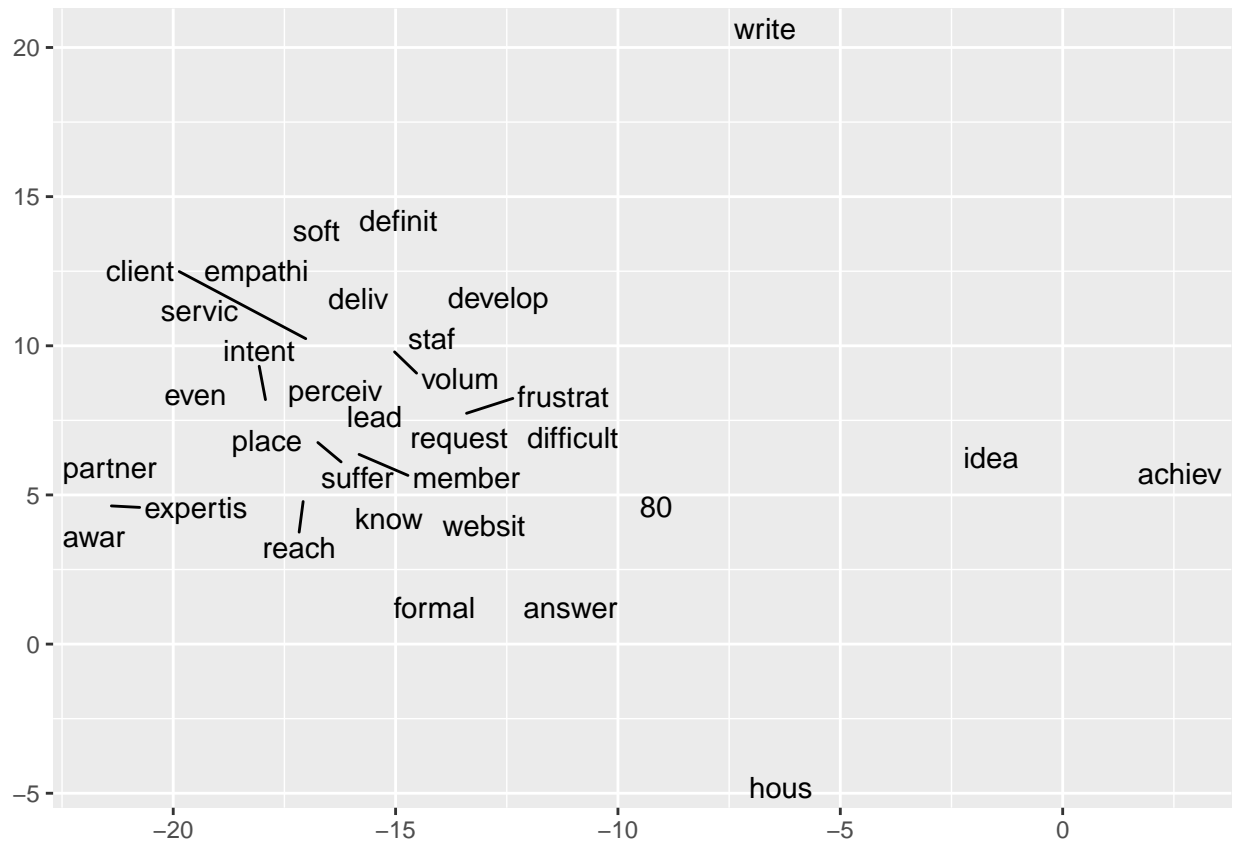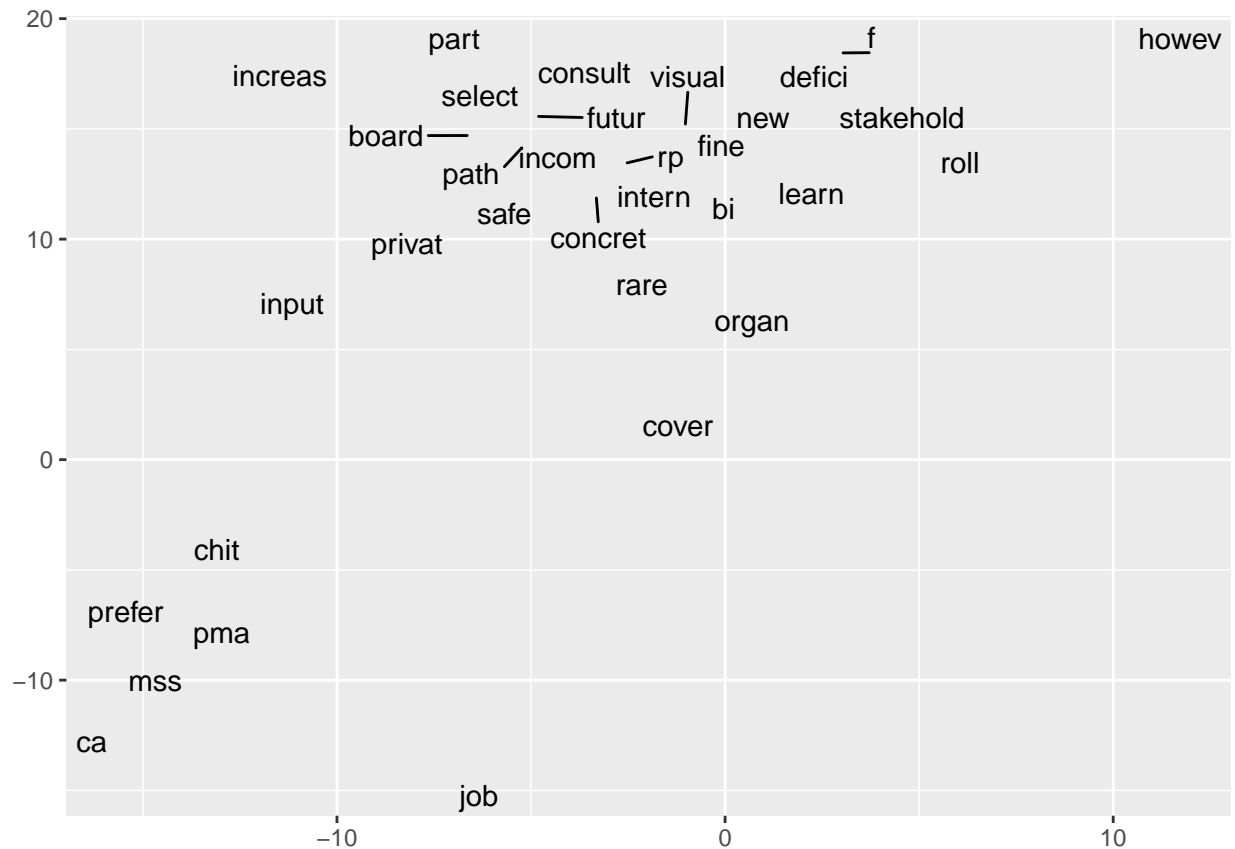
```
subsamplePlot <- unique(sample(indexMaster[dfOfIntC$labs == 4],40, replace = TRUE))
ggplot(currentTSNE[subsamplePlot,], aes(x=Dim1, y=Dim2, label=words)) +
  geom_text_repel() +
  theme(legend.title = element_blank(),
        plot.title = element_blank(),
        axis.title.x = element_blank(),
        axis.title.y = element_blank())
```

```
subsamplePlot <- unique(sample(indexMaster[dfOfIntC$labs == 5],40, replace = TRUE))
ggplot(currentTSNE[subsamplePlot,], aes(x=Dim1, y=Dim2, label=words)) +
  geom_text_repel() +
  theme(legend.title = element_blank(),
        plot.title = element_blank(),
        axis.title.x = element_blank(),
        axis.title.y = element_blank())
```

```
## Warning: ggrepel: 1 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```
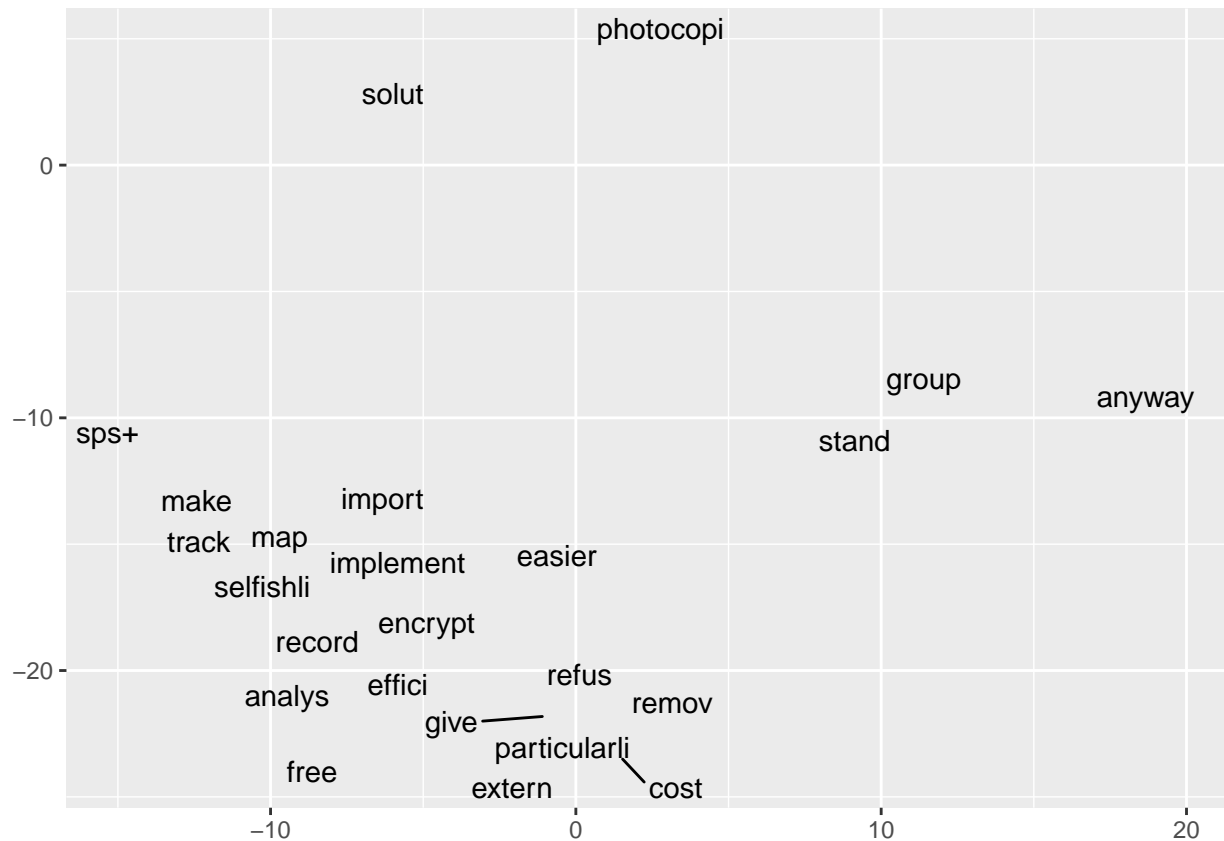
```
subsamplePlot <- unique(sample(indexMaster[dfOfIntC$labs == 6],40, replace = TRUE))
ggplot(currentTSNE[subsamplePlot,], aes(x=Dim1, y=Dim2, label=words)) +
  geom_text_repel() +
  theme(legend.title = element_blank(),
        plot.title = element_blank(),
        axis.title.x = element_blank(),
        axis.title.y = element_blank())
```

```
## Warning: ggrepel: 2 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```

```
subsamplePlot <- unique(sample(indexMaster[dfOfIntC$labs == 7],40, replace = TRUE))
ggplot(currentTSNE[subsamplePlot,], aes(x=Dim1, y=Dim2, label=words)) +
  geom_text_repel() +
  theme(legend.title = element_blank(),
        plot.title = element_blank(),
        axis.title.x = element_blank(),
        axis.title.y = element_blank())
```

```
## Warning: ggrepel: 9 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```

```
subsamplePlot <- unique(sample(indexMaster[dfOfIntC$labs == 8],40, replace = TRUE))
ggplot(currentTSNE[subsamplePlot,], aes(x=Dim1, y=Dim2, label=words)) +
  geom_text_repel() +
  theme(legend.title = element_blank(),
        plot.title = element_blank(),
        axis.title.x = element_blank(),
        axis.title.y = element_blank())
```

```
## Warning: ggrepel: 1 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```
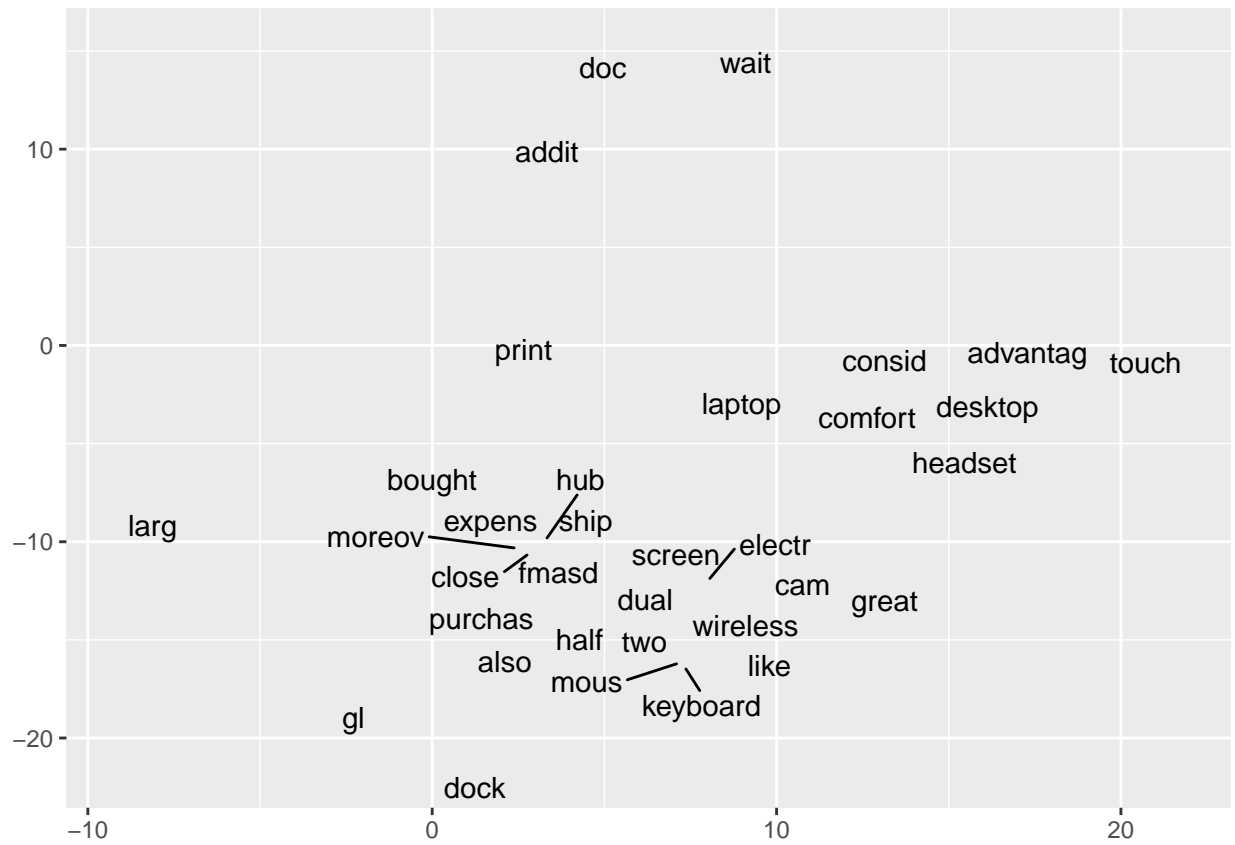
```
#plot(mnist2use.tsne$Y[,1], mnist2use.tsne$Y[,2], col = as.matrix(emb14fd[,ncol(emb14fd)]))
```