Goals

- 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)

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.

```
# 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)

# 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")
```

```
## 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"
  )

  # 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
```

```
# 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)

  # 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.

```
# 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)
  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"))

} 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"),
  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"),
  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"),
  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",""),
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",""),
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("RCO2",""),
c("RCO2",""),
c("RFAS",""),
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",""),
  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"),
  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 %>%
    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)

```
# importing all necessary modules
## Data managment
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
### 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.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()

        ### 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
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())

    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

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, ran

    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

    return topic_encoded_df, encoding_matrix

# Word2Vec
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=1000

    ## 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

# 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
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
```

```
## 0
```

```
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
```

```
## 0
```

```
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
```

```
## 0
```

```
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
```

```
## 0
```

```
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
```

```
## 0
```

```
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
```

```
## 0
```

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

```
## 0
```

```
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
```

```
## 0
```

```
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

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

## Use single variable decomposition for the number of topics elucidated in the prior segment
```

```
## 0
```

```
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')
    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')

# 2. Word2Vec Segment
## Model the topic to find synonyms
model14 = create_sg_model(c14df)
model14.wv.most_similar('work')[:10]
```

```
## [('quickli', 0.5907912254333496), ('longer', 0.582170844078064), ('slow', 0.5799008011817932), ('poi
```

```
model14.wv.most_similar('train')[:10]
```

```
## [('advanc', 0.6264476180076599), ('self', 0.6156746745109558), ('mentor', 0.6150597333908081), ('bas
```

```
model14.wv.most_similar('tool')[:10]
```

```
## [('obsolet', 0.667883574962616), ('winfast', 0.6522048711776733), ('forecast', 0.5873702168464661),
```

```
model22 = create_sg_model(c22df)
model22.wv.most_similar('email')[:10]
```

```
## [('pick', 0.9696599841117859), ('contact', 0.968529462814331), ('prefer', 0.963080644607544), ('go',
```

```
model22.wv.most_similar('team')[:10]
```

```
## [('level', 0.9594218134880066), ('winfast', 0.9548113942146301), ('contact', 0.9533220529556274), ('
```

```
model30 = create_sg_model(c30df)
model30.wv.most_similar('servic')[:10]
```

```
## [('except', 0.9672766923904419), ('provid', 0.962245762348175), ('unabl', 0.9045405387878418), ('give
```

```
model30.wv.most_similar('burden')[:10]
```

```
## [('case', 0.9962979555130005), ('administr', 0.9791812896728516), ('us', 0.9722337126731873), ('staf
```

```
model40 = create_sg_model(c40df)
model40.wv.most_similar('project')[:10]
```

```
## [('gap', 0.7306121587753296), ('rc02', 0.7271509766578674), ('handoff', 0.6887916326522827), ('eur',
```

```
model40.wv.most_similar('procur')[:10]
```

```
## [('asset', 0.7347708940505981), ('25', 0.7248533368110657), ('safeti', 0.6834445595741272), ('stream
```

```
model40.wv.most_similar('fund')[:10]
```

```
## [('agreement', 0.8035979270935059), ('earlier', 0.8010596036911011), ('fiscal', 0.7480176687240601),
```

```
model41 = create_sg_model(c41df)
model41.wv.most_similar('time')[:10]
```

```
## [('relay', 0.5462765693664551), ('accur', 0.5457806587219238), ('relev', 0.5347835421562195), ('road
```

```
model41.wv.most_similar('servic')[:10]
```

```
## [('target', 0.6411448121070862), ('fist', 0.6173452138900757), ('tailor', 0.6170654892921448), ('ris
```

```
model43 = create_sg_model(c43df)
model43.wv.most_similar('time')[:10]
```

```
## [('rapid', 0.7020166516304016), ('audit', 0.6932877898216248), ('fashion', 0.6799972653388977), ('ac
```

```
model43.wv.most_similar('respons')[:10]
```

```
## [('resolut', 0.8028725981712341), ('written', 0.7772082090377808), ('ye', 0.7747296094894409), ('sta
```

```
model45 = create_sg_model(c45df)
model45.wv.most_similar('time')[:10]
```

```
## [('achiev', 0.6582099199295044), ('comfort', 0.6525663137435913), ('6990', 0.6282333731651306), ('ne
```

```
model45.wv.most_similar('manag')[:10]
```

```
## [('ye', 0.771929144859314), ('want', 0.7471611499786377), ('frad', 0.7238593101501465), ('inquir', 0
```

```
model46 = create_sg_model(c46df)
model46.wv.most_similar('home')[:10]
```

```
## [('big', 0.6613602042198181), ('plu', 0.6117099523544312), ('frame', 0.5763698220252991), ('advantag
```

```
model46.wv.most_similar('provid')[:10]
```

```
## [('excel', 0.7533884048461914), ('influenc', 0.6707832217216492), ('profession', 0.6652481555938721)
```

```
model48 = create_sg_model(c48df)
model48.wv.most_similar('listen')[:10]
```

```
## [('merci', 0.9528046250343323), ('reach', 0.9495986104011536), ('consider', 0.9203194975852966), ('t
```

```
model48.wv.most_similar('feedback')[:10]
```

```
## [('check', 0.9832060933113098), ('gather', 0.9011625051498413), ('consider', 0.8640397787094116), ('
```

```
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)

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

```
## [('judgement', 0.7956973910331726), ('matter', 0.7656961679458618), ('manner', 0.7331850528717041),
```