Project Code

Jared Yu

March 22, 2019

```
library(dplyr); library(ggplot2); library(astsa); library(rvest) # Load libraries
library(htmlwidgets); library(RColorBrewer); library(fredr)
library(lubridate); library(tidyverse); library(plotly); library(scales)
library(beepr); library(tm); library(SnowballC); library(NLP)

# Load/Clean Data
transaction <- read.csv('transaction.csv', stringsAsFactors = FALSE)
data_sub <- subset(transaction, select = c('recipient_location_state_code',
  'total_obligation', 'award_id', 'fiscal_year'),
  transaction$recipient_location_state_code %in% state.abb &
  transaction$total_obligation > 0 &
  transaction$fiscal_year > 2010 &
  transaction$fiscal_year < 2019)

data_list <- split(data_sub, (seq(nrow(data_sub))-1) %/% 10000)

unique_id_func <- function(list_element) {
  # This function works to get the unique total_obligation per
  # award_id. It does so using the following methods:
  # Using the list of dataframes which have been split by a certain
  # length, split again each dataframe by award_id. It then coalesces
  # each of these award_id's such that all the repeated information
  # is removed and NA's are ignored if possible.
  id_split <- split(list_element, list_element$award_id)
  # head(id_split)
  # id_split <- id_split[1:2]
  df_list <- lapply(id_split, function(x) lapply(1:nrow(x), function(y) x[y,]))
  coalesce_list <- lapply(df_list, function(x) coalesce(!!!x))
  do.call('rbind', coalesce_list)
}

# Merge data together and repeat for final data frame
data_list_merge <- lapply(data_list, unique_id_func)
data_list_comb <- do.call('rbind', data_list_merge)
data_list_award_table <- table(data_list_comb$award_id)
data_list_award_table_sub <- data_list_award_table[data_list_award_table > 1]
remain_dupl <- names(data_list_award_table_sub)
remain_dupl_df <- data_list_comb[data_list_comb$award_id %in% remain_dupl, ]
not_dupl_df <- data_list_comb[!data_list_comb$award_id %in% remain_dupl, ]
last_coalesce <- unique_id_func(remain_dupl_df)
data_comb <- rbind(last_coalesce, not_dupl_df)
data_comb$recipient_location_state_code <- as.character(data_comb$recipient_location_state_code)

# spending by state
ann_state_spending <- data_comb %>%
  select(recipient_location_state_code, total_obligation, fiscal_year) %>%
  group_by(fiscal_year, recipient_location_state_code) %>%
```

```
  summarise(state_spending = sum(total_obligation))

ann_state_spending <- as.data.frame(ann_state_spending)

### Plotting Spending
make_lineplot = function(folder_path, state_index, mydata) {
  # Create barplot of state spending
  png(paste(folder_path, state.abb[state_index], ".png", sep = '')) # Save to file
  myplot <- ggplot(data = subset(mydata, recipient_location_state_code ==
                                   state.abb[state_index]), aes(x = fiscal_year, y = state_spending)) +
    geom_line(aes(y = state_spending, colour = 'State Spending')) +
    geom_line(aes(y = ann_spending$ann_sum/50, colour = 'Country Spending')) +
    scale_x_continuous(breaks = 0:2100) +
    xlab(paste('Year')) + ylab('Annual Spending') +
    ggtitle(paste(state.name[state_index], ' Annual Spending')) +
    theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
    scale_y_continuous(sec.axis = sec_axis(~., name = 'Country'))
  print(myplot)
  dev.off()
}
sapply(1:length(state.abb), function(x) # Create barplot for all states
  make_lineplot(folder_path = "./state_spending/",
                state_index = x,
                mydata = ann_state_spending))

state_national_spending_func <- function(annual_state_spending = ann_state_spending,
                                         annual_national_spending = ann_spending,
                                         state_index = 1, folder_path) {
  # Plot the GDP for the whole country With local state GDP.
  png(paste(folder_path, state.abb[state_index], ".png", sep = ''),
      width = 1920, height = 1000) # Save to file
  time <- seq(2011, 2018, 1)
  left_y_axis <- annual_state_spending[annual_state_spending$recipient_location_state_code ==
                                         state.abb[state_index],]
  right_y_axis <- annual_national_spending$ann_sum
  par(mar=c(5, 4, 4, 6) + 0.1)

  plot(time, left_y_axis$state_spending, pch=16, axes=FALSE, xlab="", ylab="",
       type="b",col="black", main = paste0(state.name[state_index], ' Annual Spending'))
  axis(2, ylim=c(range(left_y_axis$state_spending)[1],range(left_y_axis$state_spending)[2]),
       col="black",las=1)  ## las=1 makes horizontal labels
  mtext(paste0(state.abb[state_index], ' Spending'),side=2,line=2.5)
  box()
  par(new=TRUE)

  ## Plot the second plot and put axis scale on right
  plot(time, right_y_axis, pch=15,  xlab="", ylab="",
       axes=FALSE, type="b", col="red")
  ## a little farther out (line=4) to make room for labels
  mtext("National Spending",side=4,col="red",line=4)
  axis(4, col="red",col.axis="red",las=1)

  ## Draw the time axis
```

```
  axis(1,pretty(range(time)))
  mtext("Years",side=1,col="black",line=2.5)

  ## Add Legend
  legend("topright",legend=c(paste0(state.abb[state_index], ' Spending'),
                             "National Spending"), text.col=c("black","red"),pch=c(16,15),col=c("black"
  dev.off()
}

sapply(1:length(state.abb), function(x)
  state_national_spending_func(folder_path = "./state_spending/",
                               state_index = x))

### Time Series Forecasting
norm_diff_plots <- function(state_spending = ann_state_spending, curr_state = state.abb,
                            state_index = 1, folder_path) {
  png(paste(folder_path, curr_state[state_index], ".png", sep = ''),
      width = 1920, height = 1000) # Save to file
  par(mfrow=c(2,2))
  curr_state_spending <- state_spending[state_spending$recipient_location_state_code ==
                                          curr_state[state_index], ]
  plot(curr_state_spending$fiscal_year, curr_state_spending$state_spending, type = 'l',
       main = paste(curr_state[state_index],' Total Obligation'),
       xlab = 'Year', ylab = 'Annual Total Obligation')
  points(curr_state_spending$fiscal_year, curr_state_spending$state_spending)

  # Differenced data
  plot(curr_state_spending$fiscal_year[-1], diff(curr_state_spending$state_spending),
       type = 'l', main = paste(curr_state[state_index],' Differenced Total Obligation'),
       xlab = 'Year', ylab = 'Differenced Annual Total Obligation')
  points(curr_state_spending$fiscal_year[-1], diff(curr_state_spending$state_spending))

  # ACF/PACF
  acf(diff(curr_state_spending$state_spending), main = 'ACF (differenced)')
  pacf(diff(curr_state_spending$state_spending), main = 'PACF (differenced)')
  dev.off()
}

sapply(1:length(state.abb), function(x)
  norm_diff_plots(state_spending = ann_state_spending, state_index = x,
                  folder_path = './forecast/'))

# Time Series modeling
ann_spending_scaled <- ann_state_spending
ann_spending_scaled$state_spending <- log(ann_spending_scaled$state_spending)

fitting_sarima = function(i, j, state_forecast = curr_forecast){
  sarima(state_forecast$state_spending,
         p = i, d = 0, q = j, Model = FALSE, details = FALSE)
}

ann_spending_forecast <- function(ann_spending = ann_spending_scaled, folder_path,
                                  curr_state = state.abb, state_index = 1) {
```

```
  # This function will plot the observed and fitted values for an ARIMA model according to AICc.
  # Additionally, it will create a forecast for the next 5 years in annual total obligation.

  png(paste(folder_path, state.abb[state_index], ".png", sep = ''))

  # Load current state data, and find the p, q for ARIMA(p,1,q)
  curr_forecast <- ann_spending[ann_spending$recipient_location_state_code ==
                                  curr_state[state_index],]
  pq_choices <- data.frame(p = rep(0:4, each = 5), q = rep(0:4, 5))
  all_sarima <- mapply(fitting_sarima, pq_choices$p, pq_choices$q)

  npair = dim(all_sarima)[2]
  AICC_result = sapply(1:npair, function(x) all_sarima[,x]$AICc)
  pq <- pq_choices[which.min(AICC_result),]

  arima_p1q <- sarima(curr_forecast$state_spending, # Fit ARIMA(p,1,q) model
                      p = pq[,1], d = 1, q = pq[,2],
                      details = FALSE, Model = FALSE)

  sarima.for(curr_forecast$state_spending, n.ahead = 5,
             p = pq[,1], d = 1, q = pq[,2])
  title(paste(state.name[state_index], ' Annual Spending Forecast'))
  dev.off()
}

sapply(1:length(state.abb), function(x)
  ann_spending_forecast(folder_path = './arima_forecast/', state_index = x))

### Sector Analysis
### Web scrape NAICS sector information
url <- 'https://www.missourieconomy.org/about_us/naics_sect.stm'
webpage <- read_html(url)
html_nodes(webpage,'table')
web_data <- webpage %>%
  html_nodes("table") %>%
  html_nodes("td") %>%
  html_text("td")

# Sort html table data
code_posi <- grep("[0-9]", web_data)
x <- code_posi[1]:length(web_data)
pos <- code_posi - 3
pat <- rep(seq_along(pos), times = diff(c(pos, length(x) + 1)))
row_num_posi_list <- split(x, pat)
naics_sec <- do.call("rbind", lapply(row_num_posi_list, function(x) web_data[x]))
wrong_sec_info <- grepl("[0-9]", naics_sec[,3])
naics_sec_corrected <- naics_sec[,3]
for (i in 1:length(naics_sec[,3])){ # Correct sector information
  if (wrong_sec_info[i]) {
    naics_sec_corrected[i] <- naics_sec_corrected[i-1]
  }
}
naics_sec[,3] <- naics_sec_corrected
```

```r
naics_sec <- data.frame(naics_sec)

# Clean NAICS data frame
names(naics_sec) <- c("naics_code", "description", "sector")
naics_sec$description <- gsub("[\r\n\t]", "", naics_sec$description)
naics_sec$description <- gsub("\\s+", " ", naics_sec$description)
naics_sec$naics_code <- as.character(naics_sec$naics_code)
f1 <- rbind(naics_sec[4,], naics_sec[4,])
f1$naics_code <- c(31,32)
f2 <- rbind(naics_sec[6,], naics_sec[6,])
f2$naics_code <- c(48,49)
f3 <- rbind(naics_sec[8,], naics_sec[8,])
f3$naics_code <- c(44,45)
naics_sec <- rbind(naics_sec, f1, f2, f3)
naics_sec <- naics_sec[c(-4,-6,-8), ]
naics_sec_sub <- naics_sec[,-2]
rownames(naics_sec_sub) <- NULL

# Subset data to important sections
eda_states = subset(transaction, select = c('recipient_location_state_code',
  'fiscal_year', 'naics_code'),
  !is.na(transaction$naics_code) &
  transaction$recipient_location_state_code %in% state.abb &
  transaction$fiscal_year != "")

eda_states$naics_code_sub <- substr(eda_states$naics_code, start = 1, stop = 2)
eda_states$sectors <- naics_sec_sub$sector[match(eda_states$naics_code_sub,
  naics_sec_sub$naics_code)] # Add sector information

state_sec_func <- function(state_data = eda_states, state_index = 1) {
  # Find proportion of transactions by sector, returns percentage
  # of each sector per state.
  curr_state <- state_data[state_data$recipient_location_state_code ==
                             state.abb[state_index], ]
  table(curr_state$sectors) / sum(table(curr_state$sectors))
}

# Plotly
geo_usa <- list( # Plotly formatting for geoplot
  scope = 'usa',
  showland = TRUE,
  landcolor = toRGB("gray95"),
  countrycolor = toRGB("gray80")
)

sector_geo <- function(geo_plot = geo_usa, sector_df = services_df, color_scheme = 'Reds',
                       legend_title = "Percentage (%)", sec_name) {
  # Saves a plotly geo heatmap for a specific sector. Outlines the different
  # percentages that each state has for a given sector. Saves to interactive html.
  file_path <- file.path(getwd(), "sector_geo", paste0(sec_name, '.html'))
  sector_plot <-
    plot_geo(sector_df, locationmode = 'USA-states') %>%
    add_trace(
```

```
      z = sector_df$percentage, locations = sector_df$states,
      colors = color_scheme
    ) %>%
    colorbar(title = legend_title) %>%
    layout(
      title = paste('Prevalence of ', sec_name, ' Sector by State (by %)'),
      geo = geo_plot
    )
  saveWidget(as_widget(sector_plot), file = file_path)
  sector_plot
}
sector_geo(sector_df = services_df, sec_name = 'Services')

### Economic Data
api.key <- '65e973606fb36ebc54acd85cfe427fa9' # Load API
fredr_set_key(api.key)

# Load data from Economic Data from Fred St. Louis
fred_api <- function(series_id_start, series_id_end) {
  # The function uses the API to gather all the data by state from the
  # economic website data Fred St. Louis. The function will first loop
  # through the states to apply the correct series_id to each query.
  # Then, the function will turn clean the data structure, as well as
  # doing some data cleaning to make the data more simple.
  # input: series_id_start: the begining of the series_id, series_id_end:
  # the end of the series_id
  # output: dataframe of state data for each API query
  fred_list <- lapply(state.abb, function(state) fredr(series_id = paste(series_id_start,
                                                                         state, series_id_end, sep = ''
  fred_df <- as.data.frame(do.call("rbind", fred_list))
  fred_df$series_id <- substr(fred_df$series_id, start = nchar(series_id_start) + 1,
                              stop = nchar(fred_df$series_id) - nchar(series_id_end))
  fred_df$date <- as.numeric(substr(fred_df$date, start = 1, stop = 4))
  return(fred_df)
}

NGSP <- fred_api(series_id_start = '', series_id_end = 'NGSP')

gdp_barplot <- function(folder_dir, state_index, gdp_data) {
  # Plot the GDP data
  png(paste(folder_dir, state.abb[state_index], ".png", sep = ''))
  myplot <- ggplot(data = subset(gdp_data, series_id == state.abb[state_index]),
                   aes(x = date, y = value)) + geom_bar(stat = 'identity') +
    scale_x_continuous(breaks = 0:2100) +
    xlab(paste('Year')) + ylab('Annual GDP') +
    ggtitle(paste(state.name[state_index], ' Annual GDP')) +
    geom_vline(xintercept = 2007, color = 'red') +
    theme(axis.text.x = element_text(angle = 45, hjust = 1))
  print(myplot)
  dev.off()
}

sapply(1:length(state.abb), function(x)
```

```
  gdp_barplot(folder_dir = "./state_gdp/", state_index = x, gdp_data = NGSP))

year_breaks_func <- function(year_vec, num_breaks) {
  # This function is for efficiently finding the number
  # of breaks to place inbetween the years on the x-axis.
  unique_years <- sort(unique(year_vec))
  unique_years[seq(1, length(unique_years), num_breaks)]
}

econ_ggplot <- function(econ_data, x_label = 'Years', y_label,
                        main_title, breaks_vec) {
  # Plot the economic data
  ggplot(econ_data) + geom_line(mapping = aes(x = date, y = value)) +
    facet_wrap(~ series_id, scales = 'free') + ggtitle(main_title) +
    xlab(x_label) + ylab(y_label) + geom_vline(xintercept = 2007) +
    theme(axis.text.x = element_text(angle = 30, hjust = 1)) +
    scale_x_continuous(breaks = breaks_vec)
}

econ_ggplot2 <- function(econ_data, x_label = 'Years', y_label,
                         main_title) {
  # Plot the economic data
  ggplot(econ_data) + geom_line(mapping = aes(x = date, y = value)) +
    ggtitle(main_title) + xlab(x_label) + ylab(y_label) +
    geom_vline(aes(xintercept = as.numeric(ymd('2007-02-20')))) +
    theme(axis.text.x = element_text(angle = 30, hjust = 1)) #+
}

econ_filter1 <- function(series_id_start = '', series_id_end = '', start_date = 0,
                         num_breaks, y_label, main_title) {
  # Filter the economic data so it can be plotted.
  econ_data <- fred_api(series_id_start = series_id_start,
                        series_id_end = series_id_end)
  econ_data <- econ_data[econ_data$date > start_date, ]
  econ_years <- year_breaks_func(year_vec = econ_data$date, num_breaks = num_breaks)

  econ_ggplot(econ_data = econ_data, y_label = y_label,
              main_title = main_title,
              breaks_vec = econ_years)
}

econ_filter2 <- function(series_id, start_date = 0,
                         y_label, main_title) {
  # Filter the economic data so it can be plotted.
  api_query <- fredr(series_id = series_id)
  econ_data <- api_query[, c('date', 'value')]
  econ_data <- econ_data[year(econ_data$date) > start_date, ]
  econ_ggplot2(econ_data = econ_data, y_label = y_label,
               main_title = main_title)
}

# pop size
state_pop <- econ_filter1(series_id_end = 'POP', start_date = 2000,
```

```
                          num_breaks = 3, y_label = 'Population',
                          main_title = 'State Population 2000-2018')

# med income
state_med_inc <- econ_filter1(series_id_start = 'MEHOINUS',
  series_id_end = 'A672N', start_date = 1995, num_breaks = 4,
  y_label = 'Median Income', main_title = 'State Median Income')

# education level
state_edu <- econ_filter1(series_id_start = 'GCT1502', num_breaks = 2,
  y_label = 'Percentage with Bachelor\'s Degree or Higher',
  main_title = 'State Education Level')

# unemployment
unemp_plot <- econ_filter2(series_id = 'UNRATE', start_date = 2000,
  y_label = 'Unemployment (%)', main_title = 'Unemployment Rate')

# interest rates
ir_plot <- econ_filter2(series_id = 'FEDFUNDS', start_date = 1980,
  y_label = 'Interest Rate (%)', main_title = 'Interest Rates')

# debt
debt_plot <- econ_filter2(series_id = 'GFDEBTN', start_date = 1970,
  y_label = 'Millions ($)', main_title = 'Total Public Debt (1970-2018)')

### Text Mining
bus_cat <- colnames(transaction)[c(5, 6, 8, 36, 61)]
award_agency <- colnames(transaction)[c(5, 6, 8, 36, 53)]

# Initial subset
tran_sub <- subset(transaction, select = bus_cat,
  transaction$fiscal_year > 2010 & transaction$fiscal_year < 2019 &
  transaction$total_obligation > 0 & transaction$business_categories != '' &
  transaction$recipient_location_state_code %in% state.abb &
  transaction$award_id != '')

tran_sub2 <- subset(transaction, select = award_agency,
  transaction$fiscal_year > 2010 & transaction$fiscal_year < 2019 &
  transaction$total_obligation > 0 &
  transaction$awarding_toptier_agency_name != '' &
  transaction$recipient_location_state_code %in% state.abb &
  transaction$award_id != '')

### text_mine() functions
tran_dupl_removal <- function(tran_data) {
  # Combine duplicate award_id's
  id_table <- table(tran_sub$award_id)
  id_table_sub <- id_table[id_table > 1]
  dupl_id <- names(id_table_sub)
  tran_dupl <- tran_sub[tran_sub$award_id %in% dupl_id, ]
  tran_not_dupl <- tran_sub[!tran_sub$award_id %in% dupl_id, ]
  tran_dupl_unique <- unique_id_func(tran_dupl)
  tran_comb <- rbind(tran_not_dupl, tran_dupl_unique)
```

```
  return(tran_comb)
}

year_to_state <- function(yearly_df = tran_data_year[[1]], desc = 'business_categories') {
  # Returns two lists, where the description and spending for a specific
  # year for all states is returned.
  yearly_df <- data.frame(yearly_df, stringsAsFactors = FALSE)
  state_split <- split(yearly_df, yearly_df$recipient_location_state_code)
  state_desc <- lapply(state_split, function(x) x[,desc])
  state_spend <- lapply(state_split, function(x) x$total_obligation)
  return(list(state_desc, state_spend))
}

annual_desc_spend <- function(year_list_element = year_to_state_desc_spend[[1]]) {
  # Takes the yearly list element and breaks down into list of 50 descriptions
  # and 50 spending amounts. Then filters the descriptions into a weighted
  # character vector. Returns top 25.
  year_desc <- year_list_element[[1]]
  year_spend <- year_list_element[[2]]

  # Go by state to find the weighted character vectors.
  # Creates a list of 50 states each with a list of an index and string.
  state_desc_word_filter <- lapply(year_desc,
                                   function(x) word_filter(year_desc_element = x))
  # Add the spending to the list of 50.
  for(i in 1:length(state_desc_word_filter)) {
    state_desc_word_filter[[i]][length(state_desc_word_filter[[i]]) + 1] <-
      year_spend[i]
  }

  # Sort through each 50 states in list and use the word filter.
  state_weighted_top25 <- lapply(state_desc_word_filter, function(x)
    state_list_to_char_vec(state_string_ind_spend = x))
  return(state_weighted_top25)
}

word_filter <- function(year_desc_element = year_desc[[1]]) {
  # Filters words to stem words.
  # Process of doing initial text mining of description.
  removed_punc = gsub('[[:punct:]]', ' ', year_desc_element)
  lower_case = tolower(removed_punc)

  # Remove stopwords, create stem words
  stop_words1 = removeWords(lower_case, stopwords('en'))
  stop_words2 = removeWords(stop_words1, stopwords('SMART'))
  stem_words = stemDocument(stop_words2)

  # Deal with any white space and create removal index
  clean_whitespace = stripWhitespace(stem_words)
  remove_na = clean_whitespace[clean_whitespace != 'NA']
  rm_ind = which(remove_na != '')
  lead_trail = gsub("^\\s+|\\s+$", "", clean_whitespace[rm_ind])
  rm_ind = which(lead_trail != '')
```

```
  return(list(lead_trail, rm_ind))
}

state_list_to_char_vec <- function(state_string_ind_spend =
                                     state_desc_word_filter[[1]]) {
  # Creates a data frame from filtered descriptions and then returns
  # the final top 25 weighted words.
  lead_trail <- state_string_ind_spend[[1]]
  rm_ind <- state_string_ind_spend[[2]]
  price <- state_string_ind_spend[[3]]

  character_vector <- data.frame(keywords = lead_trail[rm_ind], # df of keywords and prices
                                 weighted_price = price[rm_ind] /
                                   sapply(strsplit(lead_trail[rm_ind], " "), length),
                                 stringsAsFactors = FALSE)
  weighted_char_vectors(price = price[rm_ind], character_vector = character_vector)
}

weighted_char_vectors <- function(price, character_vector) {
  # The weighted_char_vectors function will take in the price and character
  # vector which have previously been created. It will then split the
  # character vector into the individual keywords of each expense, and then
  # assign the average price to each keyword. These keywords are then combined
  # so that keywords which are matching will also have their weighted prices
  # from across different expenses summed together.

  # Split the keywords from each exepense, and assign them to their average
  # prices from each character vector.
  keywords_list = strsplit(character_vector$keywords, ' ', fixed = TRUE)
  keywords_list_len = sapply(keywords_list, length)
  avg_price = character_vector$weighted_price / keywords_list_len
  weighted_words = Map(cbind, keywords_list, avg_price)
  weighted_words = do.call("rbind", weighted_words)

  # Turn the keywords into a dataframe, and then group the dataframe
  # by keyword to sum the total weighted average.
  weighted_df = as.data.frame(weighted_words, stringsAsFactors = FALSE)
  names(weighted_df) = c("keywords", "char_weights")
  weighted_df$char_weights = as.numeric(weighted_df$char_weights)
  sum_df = tapply(weighted_df$char_weights, weighted_df$keywords,
                  function(x) sum(x, na.rm = TRUE))

  # Return the top 25 keywords, along with their weighted prices in terms
  # of their proportion.
  top_25 = head(sort(sum_df, decreasing = TRUE), n = 25)
  round(top_25 / sum(top_25), 3)
}

text_mine <- function(data_sub = tran_sub, curr_desc = 'business_categories') {
  # Collects the top 25 stem words from each state by year/
  data_sub$recipient_location_state_code <- # Fix datatype
    as.character(data_sub$recipient_location_state_code)
  data_sub <- data.frame(data_sub, stringsAsFactors = FALSE)
```

```
  tran_comb <- tran_dupl_removal(tran_data = data_sub) # Remove duplicates

  tran_data_year <- split(tran_comb, tran_comb$fiscal_year) # Split by year

  year_to_state_desc_spend <- lapply(tran_data_year,
                                     function(x) year_to_state(yearly_df = x, desc = curr_desc))

  ann_state_desc <- lapply(year_to_state_desc_spend, function(x)
    annual_desc_spend(year_list_element = x))
  ann_state_desc
}

# Create weighted character vectors
bus_text_mine <- text_mine(data_sub = tran_sub,
                           curr_desc = business_categories)
agency_name_text_mine <- text_mine(data_sub = tran_sub2,
                                   curr_desc = awarding_toptier_agency_name)
```