Thesis & Dissertation Data

Clarke Iakovakis

July, 2018

Contents

1 Getting the data **1**
- 1.1 Sierra 1
- 1.2 Exporting the data 3

2 Reading the data **4**
- 2.1 Cleaning author (100) field for ETDs 4
- 2.2 Data Cleaning 6

3 Finished **11**

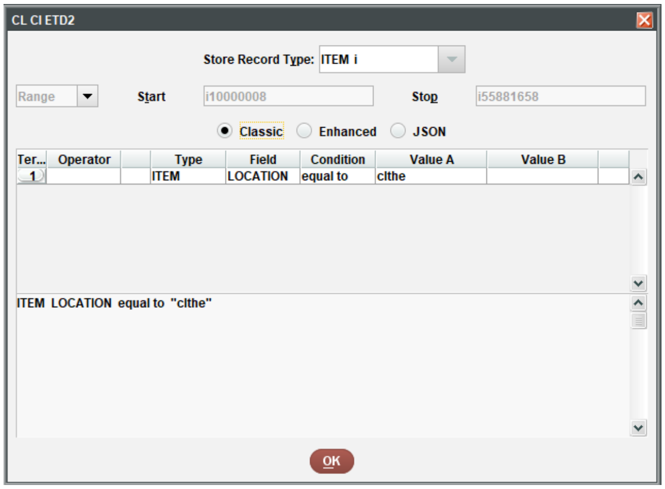
1 Getting the data

1.1 Sierra

Start by downloading the data from Sierra.

1.1.1 General Collection

Start in the general collection. This one is easy: **Record type** is ITEM. **ITEM LOCATION** equal to *clthe*.



CL CI ETD2

Store Record Type: ITEM i

Range Start i10000008 Stop i55881658

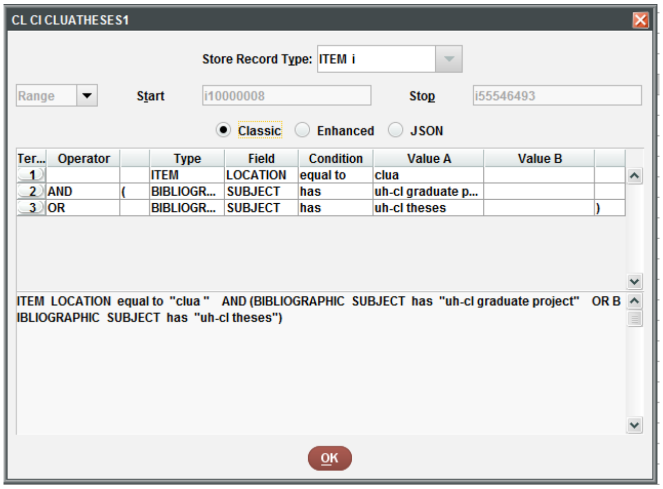
● Classic ○ Enhanced ○ JSON

Ter...	Operator		Type	Field	Condition	Value A	Value B
1			ITEM	LOCATION	equal to	clthe	

ITEM LOCATION equal to "clthe"

1.1.2 Archives

There are also some theses and dissertations in archives that are not in the general collection. **Record type** is ITEM. (**ITEM LOCATION** equal to *clua*) AND (**BIBLIOGRAPHIC SUBJECT** has *uh-cl graduate project* OR **BIBLIOGRAPHIC SUBJECT** has *uh-cl theses*)



Store Record Type: ITEM i

Range Start i10000008 Stop i55546493

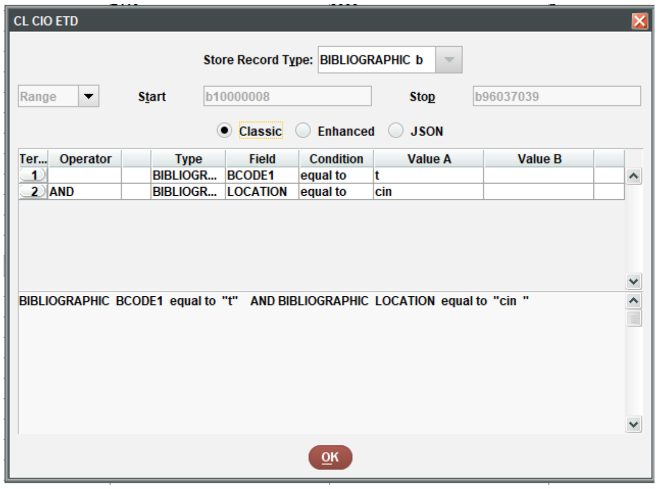
● Classic ○ Enhanced ○ JSON

Ter...	Operator		Type	Field	Condition	Value A	Value B	
1			ITEM	LOCATION	equal to	clua		
2	AND	(	BIBLIOGR...	SUBJECT	has	uh-cl graduate p...		
3	OR		BIBLIOGR...	SUBJECT	has	uh-cl theses		)

```
ITEM  LOCATION  equal to  "clua "   AND (BIBLIOGRAPHIC  SUBJECT  has  "uh-cl graduate project"   OR B
IBLIOGRAPHIC  SUBJECT  has  "uh-cl theses")
```

1.1.3 ETDs

Finally we get the ETDs. Note that for this one, **Record type** is BIBLIOGRAPHIC. (**BCODE1** = *t*) AND (**BIBLIOGRAPHIC LOCATION** equal to *cin.*)



Store Record Type: BIBLIOGRAPHIC b

Range Start b10000008 Stop b96037039

● Classic ○ Enhanced ○ JSON

Ter...	Operator		Type	Field	Condition	Value A	Value B	
1			BIBLIOGR...	BCODE1	equal to	t		
2	AND		BIBLIOGR...	LOCATION	equal to	cin		

```
BIBLIOGRAPHIC  BCODE1  equal to  "t"   AND BIBLIOGRAPHIC  LOCATION  equal to  "cin "
```

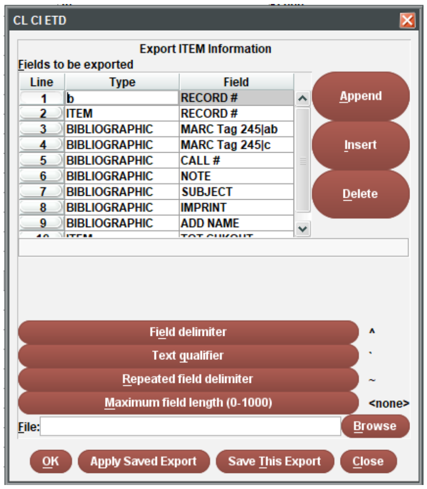
1.2 Exporting the data

The export fields include:

- BIBLIOGRAPHIC RECORD #
- BIBLIOGRAPPHIC ITEM #
- BIBLIOGRAPHIC MARC Tag 245|ab
- BIBLIOGRAPHIC MARC Tag 245|c
- BIBLIOGRAPHIC CALL #
- BIBLIOGRAPHIC NOTE
- BIBLIOGRAPHIC SUBJECT
- BIBLIOGRAPHIC IMPRINT
- BIBLIOGRAPHIC ADD NAME (exports from Sierra as ADD AUTHOR)
- ITEM TOTCHKOUT
- BIBLIOGRAPHIC MARC Tag 100 [ETDs only]
- BIBLIOGRAPHIC MARC Tag 856|u [ETDs only]

Important: export with the following parameters:

- Field delimiter: ˆ (carat)
- Text qualifier: ‘ (**not a quotation mark**. It’s a backquote, or shift + tilde, next to the 1 key)
- Repeated field delimiter: ∼ (tilde)



Export ITEM Information

Fields to be exported

Line	Type	Field
1	b	RECORD #
2	ITEM	RECORD #
3	BIBLIOGRAPHIC	MARC Tag 245|ab
4	BIBLIOGRAPHIC	MARC Tag 245|c
5	BIBLIOGRAPHIC	CALL #
6	BIBLIOGRAPHIC	NOTE
7	BIBLIOGRAPHIC	SUBJECT
8	BIBLIOGRAPHIC	IMPRINT
9	BIBLIOGRAPHIC	ADD NAME

Field delimiter ^

Text qualifier `

Repeated field delimiter ~

Maximum field length (0-1000) <none>

File:

2 Reading the data

Data is downloaded from Sierra in three files: from archives (tds.ua), from the general print collection (tds.gen), and ETDs (tds.etd). Notice when we `read.csv()` it in that the `sep` argument set to carat, and the `quote` argument set to backquote. Also, `encoding` is UTF-8.

```
#setwd("C:/Users/iakovakis/Documents/Projects/ETDs/Summer18/")
tds.ua <- read.csv(file="./data/raw/tds.ua.txt"
                   , sep = "^"
                   , quote = "`"
                   , na.strings = ""
                   , stringsAsFactors = F
                   , encoding= "UTF-8")
tds.gen <- read.csv("./data/raw/tds.clthe.txt"
                    , sep = "^"
                    , na.strings = ""
                    , quote = "`"
                    , stringsAsFactors = F
                    , encoding= "UTF-8")
tds.etd <- read.csv(file="./data/raw/tds.etds.txt"
                   , sep = "^"
                   , quote = "`"
                   , na.strings = ""
                   , stringsAsFactors = F
                   , encoding= "UTF-8")
tds.etd <- rename(tds.etd, link = X856.u)
```

2.1 Cleaning author (100) field for ETDs

The 100 field in the ETDs (`tds.etd`) metadata has some name issues that need to be resolved right off the bat so we can merge it with the archives and general collection. Specifically, since ETDs were exported from a Bibliographic rather than an Item level list, the name of the NOTE field must be changed with `rename()`. Then, the word "author" appears in a couple different configurations and need to be removed, as do periods.

Then, ETDs were cataloged with last name, first name. However, last name needs to be swapped with first name so it matches the rest of the items (use `extract()` and a regular expression `([^ ]+) (.*)` putting each name into a different variable). Then get rid of the comma, and `paste()` the names together, replace the NA values with the results, and remove the extra variables so the columns match the other two datasets exactly.

```
clean.etdAuthor <- function(df) {
  df <- rename(df, NOTE = NOTE.BIBLIO.)
  df$X100 <- str_replace_all(df$X100, ", author", "")
  df$X100 <- str_replace_all(df$X100, " author", "")
  df$X100 <- str_replace_all(df$X100, "\\.", "")
  df <- df %>%
    # pull last name and first name and paste them together
    extract(X100, c("LastName", "FirstName"), "([^ ]+) (.*)") %>%
    mutate(LastName = str_replace_all(LastName, ",", "")) %>%
    mutate(author = paste(FirstName, LastName))
  df$X245.c[which(is.na(df$X245.c))] <- df$author[which(is.na(df$X245.c))]
  df <- select(df, -FirstName, -LastName, -author)
}
tds.etd <- clean.etdAuthor(tds.etd)
```

2.1.1 Bind it together

Of course, we don't want to work with three different datasets, so we `rbind()` them together since all their variables match. Then `rename()` the columns into something more comprehensible.

The next two expressions are to deal with items that have more than one copy with possibly more than one checkout per copy. We don't want duplicate records, but we do want to account for all checkouts. So we `group_by()` bibRecord, then sum the total checkouts and return that to `bibGroup`. Then we get rid of duplicate bibliographic records in the primary dataset (`tds`), deselect the existing total checkouts column, and join it with `bibGroup`, which has the summed checkouts per bibliographic record.

```
tds.gen$format <- rep("print", nrow(tds.gen))
tds.ua$format <- rep("print", nrow(tds.ua))
tds.etd$format <- rep("electronic", nrow(tds.etd))
tds.gen$link <- rep(NA, nrow(tds.gen))
tds.ua$link <- rep(NA, nrow(tds.ua))


tds <- rbind(tds.ua, tds.gen, tds.etd)
tds <- rename(tds
              , bibRecord = RECORD...BIBLIO.
              , itemRecord = RECORD...ITEM.
              , title = X245.ab
              , author = X245.c
              , callNumber = CALL...BIBLIO.)
bibGroup <- tds %>%
  group_by(bibRecord) %>%
  summarise(TOT.CHKOUT = sum(TOT.CHKOUT))
tds <- tds %>%
  filter(!duplicated(bibRecord)) %>%
  select(-TOT.CHKOUT) %>%
  left_join(bibGroup, by = "bibRecord") %>%
  arrange(desc(TOT.CHKOUT))
```

2.2 Data Cleaning

This data needs a lot of cleaning.

2.2.1 Create Type

The `create.type` function creates a variable called type in the main theses & dissertations dataframe (`tds`) which includes three values: thesis, graduate project, and dissertation.

It works by first creating an empty vector, `type`, of `mode = character` that has as many elements as whatever dataframe is fed into it (`length = nrow(df)`). It then contains four expressions that build the vector piece by piece. This is called "growing a vector."

The first expression looks through the SUBJECT column for mention of the word "theses"–if it detects the word (`str_detect(df$SUBJECT, "theses")`), it creates a new value in the `type` vector in the same place that the item was detected, with the word "thesis". This is done by subsetting type using `[]` brackets and assigning the new value to `which` element the detection was found in. It does the same thing looking for the word "Thesis" in the NOTE variable, the word "graduate project" in SUBJECT, and the word "dissertation" in SUBJECT.

```
create.type <- function(df){
  # cleans and makes consistent the different types of graduate works in
  # the type variable.
  #
  # Arg:
  #   A dataframe of theses, dissertations, and graduate projects exported
  # from Sierra including the BIBLIOGRAPHIC SUBJECT
  #   and BIBLIOGRAPHIC TYPE variables
  #
  # Returns:
  #   The same dataframe with a clean and consistent type variable
  type <- vector(mode="character", length=nrow(df))  # create empty vector
  type[which(str_detect(df$SUBJECT, "theses"))] <- "thesis"
  type[which(str_detect(df$NOTE, "Thesis"))] <- "thesis"
  type[which(str_detect(df$SUBJECT, "graduate project"))] <- "graduate project"
  type[which(str_detect(df$SUBJECT, "dissertation"))] <- "dissertation"
  df$type <- type
  return(df)
}
```

```
# run the function
tds <- create.type(tds)
```

2.2.2 Create College

The `create.college` function creates a variable called `College` in the main theses & dissertations dataframe which includes four values: College of Science and Computer Engineering, College of Human Sciences & Humanities, College of Education, and College of Business.

It works by growing a vector–two actually. The first scans through the ADD.AUTHOR variable looking for unique portions of the college name from past and present in the ADD.AUTHOR field. When those portions are detected, the full, current college name is added to the College variable. For any items in which nothing was found, it adds an NA. The second takes those NA values and looks through the NOTE field for the same information. Those are then integrated into the final College variable, which is returned.

```
create.college <- function(df) {
  # Creates a clean College variable to include which Colleges the items come from
  #
  # Arg: A dataframe of theses, dissertations, and graduate projects exported from Sie
  #   and NOTE variables
  #
  # Returns:
  #   The same dataframe with a clean and consistent College variable
  #
  # create empty vector
  College <- vector(mode = "character", length = nrow(df))
  #
  # Scan through the ADD.AUTHOR variable for key unique portions of a college name, an
  College[which(str_detect(df$ADD.AUTHOR, "of Natural"))] <- "College of Science and Eng
  College[which(str_detect(df$ADD.AUTHOR, "of Human Sciences"))] <- "College of Human Sc
  College[which(str_detect(df$ADD.AUTHOR, "of Human Science"))] <- "College of Human Sci
  College[which(str_detect(df$ADD.AUTHOR, "of Education"))] <- "College of Education"
  College[which(str_detect(df$ADD.AUTHOR, "and Computer"))] <- "College of Science and E
  College[which(str_detect(df$ADD.AUTHOR, "and Engineering"))] <- "College of Science an
  College[which(str_detect(df$ADD.AUTHOR, "of Business"))] <- "College of Business"
  College[which(str_detect(df$ADD.AUTHOR, "Sciences and Tech"))] <- "College of Science
  #
  # Attach it to the main data frame
  df$College <- College
  #
  # Replace the blank spaces in the College variable with NA)
  df$College[which(df$College == "")] <- NA
  # Since a College was not found for all items, it is now necessary to
  # check in the NOTE field to see if we can detect one there
  #
  # create a subset based on the number of elements that are NA in the
  # just-created College variable
  College.na <- df[is.na(df$College), ]
  # a new empty vector to grow
  College2 <- vector(mode = "character", length = nrow(College.na))
  #
  College2[which(str_detect(df$NOTE, "of Natural"))] <- "College of Science and Engineer
  College2[which(str_detect(df$NOTE, "of Human Sciences"))] <- "College of Human Science
  College2[which(str_detect(df$NOTE, "of Human Science"))] <- "College of Human Sciences
  College2[which(str_detect(df$NOTE, "of Education"))] <- "College of Education"
  College2[which(str_detect(df$NOTE, "and Computer"))] <- "College of Science and Engine
  College2[which(str_detect(df$NOTE, "and Engineering"))] <- "College of Science and Eng
  College2[which(str_detect(df$NOTE, "of Business"))] <- "College of Business"
  College2[which(str_detect(df$NOTE, "Sciences and Tech"))] <- "College of Science and E
  #
  College2[which(College2 == "")] <- NA
  #
  # Create an index of NA values in the College variable
  College.repl <- which(is.na(df$College))
  # Replace that index with the same items from the College 2 variable
  df$College[College.repl] <- College2[College.repl]
  return(df)
}
# run the function
tds <- create.college(tds)

# View(filter(tds, is.na(TOT.CHKOUT)))
```

2.2.3 Create Date

We create a date column with the `create.date` function. This function takes the IMPRINT variable, removes extraneous information such as the university name, periods, and quotation marks.

```
clean.date <- function(df) {
  # Remove quotation marks and periods from IMPRINT field & convert it to integer
  #
  # get that junk out
  df$IMPRINT <- str_trim(df$IMPRINT)
  df$IMPRINT <- str_replace_all(df$IMPRINT, fixed("University of Houston-Clear Lake "),
  df$IMPRINT <- str_replace_all(df$IMPRINT, fixed("University of Houston--Clear Lake, ")
  df$IMPRINT <- str_replace_all(df$IMPRINT, '\\"|\\.', "")
  df$IMPRINT <- str_replace_all(df$IMPRINT, fixed("[Houston, Tex] : University of Housto
  df$IMPRINT <- str_replace_all(df$IMPRINT, "University of Houston-Clear Lake, ", "")
  # coerce to integer
  df$IMPRINT <- as.integer(df$IMPRINT)
  return(df)
}
# run the function
tds <- clean.date(tds)
```

2.2.4 Clean author

We next clean the author column. First, the words “by” and “prepared” are removed. Then whitespace is trimmed off with `str_trim()`. The next is a bit sloppy, but works for the moment. It creates three subsets and then binds them together. The first subset `dfx` gets all items in the author column with a period at the end, and removes the period. The next `dfy` simply gathers those items without a period at the end from the outset, and the last `dfz` are the NA values. The three are then bound together with `rbind()`.

```
clean.author <- function(df) {
  df <- mutate(df, author = str_replace_all(author, "by |prepared ", ""))
  df <- mutate(df, author = str_trim(author))
  # table(is.na(df$author))  # 70 NA values
  # sloppy way to do this. get a period at the end, then take it away,
  # then bind the three together
  dfx <- df %>%
    filter(str_sub(author, nchar(author), nchar(author)) == ".") %>%
    mutate(author = str_sub(author, 1, nchar(author) - 1))
  dfy <- df %>%
    filter(str_sub(author, nchar(author), nchar(author)) != ".")
  dfz <- df[is.na(df$author), ]
  df <- rbind(dfx, dfy, dfz)
  return(df)
}
tds <- clean.author(tds)
```

2.2.5 Create hyperlink

Next a hyperlink is created by adding the base URL to the bibliographic record, minus the last character.

```
create.hyperlink <- function(df) {
  dfx <- filter(df, is.na(df$link))
  dfy <- filter(df, !is.na(df$link))
  record <- str_sub(dfx$bibRecord, 1, nchar(dfx$bibRecord) - 1)
  dfx$link <- paste0("https://library.uh.edu/record=", record)
  df <- rbind(dfx, dfy)
  return(df)
}
tds <- create.hyperlink(tds)
```

2.2.6 Clean chair

This is the most difficult field to clean because consistent name authority control was not applied across time. The `create.chair` function starts by extracting any and all text after the words chair, chairs,advisor, director, and supervisor. It then extracts all text before the appearance of a tilde (remember this was the Repeated Fields delimiter selected in the Sierra export). It then attempts to remove all titles such as Dr, Drs, PhD, EdD, MSW, JD, and so on. It finally removes all punctuation.

```
create.chair <- function(df) {
  faculty <- str_extract(df$NOTE, "(?<=chair\\: |advisor\\: |chairs\\: |director\\: |co-
  faculty[which(str_detect(faculty, "\\~"))] <- str_extract(faculty[which(str_detect(fa
  faculty[which(str_detect(faculty, "\\,"))] <- str_extract(faculty[which(str_detect(fa
  faculty <- str_replace_all(faculty, "Dr\\. |Drs\\. |Ph\\.D\\.| PhD|Ed\\.D\\.|M\\.S\\.
  faculty <- str_trim(faculty)
  faculty <- str_replace_all(faculty, "[[:punct:]]?$", "")
  faculty <- str_trim(faculty)
  faculty <- str_replace(faculty, "[[:punct:]]?$", "")
  faculty <- str_trim(faculty)
    df$chair <- faculty
  df <- arrange(df, chair)
  return(df)
}
tds <- create.chair(tds)
```

I messed up and didn't pull in the latest batch of ETDs from Spring 2018 which don't have records in Sierra yet. I did some cleanup in Excel for this one, then ran a function to clean up the names, as I did with the above `cleanETDAuthor` function.

```
# this is the latest batch from Spring 2018
tds.vireo <- read.csv(file="./data/raw/vireoExport.csv"
                      , na.strings = ""
                      , stringsAsFactors = F)

cleantds.vireo <- function(df) {
  df$author <- str_remove_all(df$author, "[[:digit:]]")
  df$author <- str_remove_all(df$author, "\\-")
  df$author <- str_trim(df$author)
  df <- df %>%
    # pull last name and first name and paste them together
    extract(author, c("LastName", "FirstName"), "([^ ]+) (.*)") %>%
    mutate(LastName = str_replace_all(LastName, ",", "")) %>%
    mutate(author = paste(FirstName, LastName)) %>%
    select(-FirstName, -LastName) %>%
    extract(chair, c("LastName", "FirstName"), "([^ ]+) (.*)") %>%
    mutate(LastName = str_replace_all(LastName, ",", "")) %>%
    mutate(chair = paste(FirstName, LastName)) %>%
    select(-FirstName, -LastName)
}
tds.vireo <- cleantds.vireo(tds.vireo)
```

2.2.6.1 Clean chair in OpenRefine

Some of the cleanup is just too extensive to do in R, and I had to use OpenRefine. See the below video for how I did that. I can't remember what the point of the below code is, and if you've followed me this far, hopefully you can parse it out :-)

```
# write.csv(tds
#           , "./data/processed/tdsV1.csv"
#           , row.names = F)

tdsOR <- read.csv("./data/processed/tdsV1-OR.csv"
                  , stringsAsFactors = F)
tdsMerge <- select(tds, bibRecord, format)
tdsOR2 <- left_join(tdsOR, tdsMerge, by = "bibRecord")
names(tdsOR2) <- tolower(names(tdsOR2))
names(tds.vireo) <- tolower(names(tds.vireo))
tdsx <- rbind(tdsOR2, tds.vireo)

# write.csv(tdsx, "./data/results/tdsFinal.csv", row.names = F)
```

3 Finished

Now we have a dataset with relatively clean faculty names. Here's how to read it in. All work at this point was finished in Tableau, and that story won't be told here. But you can view it at http://uhcl.libguides.com/thesesdissertations/visualization.

```
all <- read.csv("./data/results/tdsFinal.csv"
                , stringsAsFactors = F
                , encoding = "ASCII")  # got converted to ASCII in Excel
```

I didn't use this below bit of code, but you can do some Open Refine work in R, so I'm keeping it here for future reference.