Demo: Hashing and Anonymizing Datasets

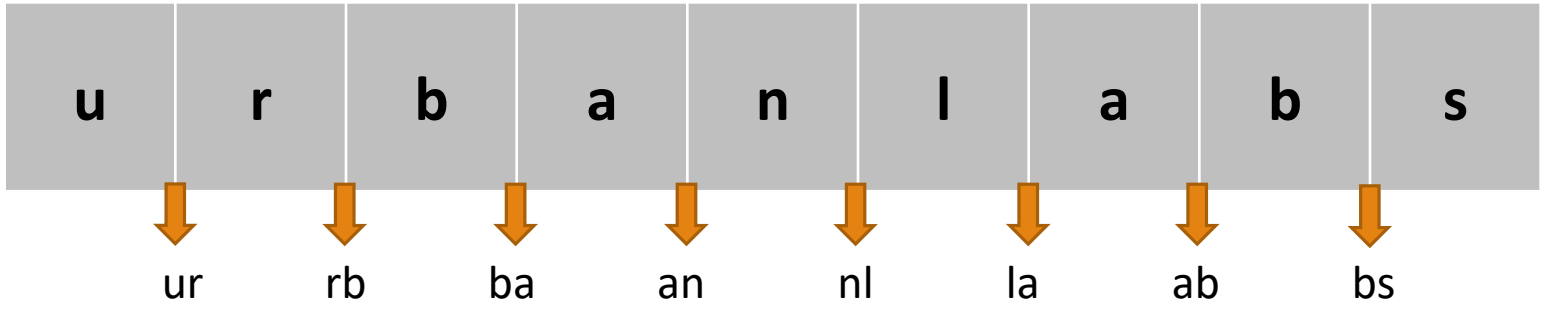
LEOSON HOAY

Contents

- Protocol
- Code Flow (Hashing)
- Action Points (Hashing)
- Code Flow (Merging)
- Action Points (Merging)

Protocol (from previous project)

- MD5, ‘Salted’, and Locality-Sensitive Hashing
- Locality-Sensitive Hashing (LSH)
  - **MinHash** (min-wise independent permutations)
  - Collect **minimum values** from hashing bigrams of the identifiable information (name), each time changing the hash value permutations by adding the iteration number to the
    - **150**



Protocol (from previous project)

- **Required Preparation**
  - Date-shifting and Dataset Truncation
    - Prevent identification/record linkage by date or ordering
    - **Shift all dates in the dataset by a random number of days** (to preserve temporal relationship)
    - Truncate random number of data points from front and back
  - The Salt
    - **Keyword, to be known only by analysts hashing the datasets**
    - Preferably read salt into script from another secure file
  - Password-Protected Folders
    - For data transfer between analysts
    - Use **randomly generated password**

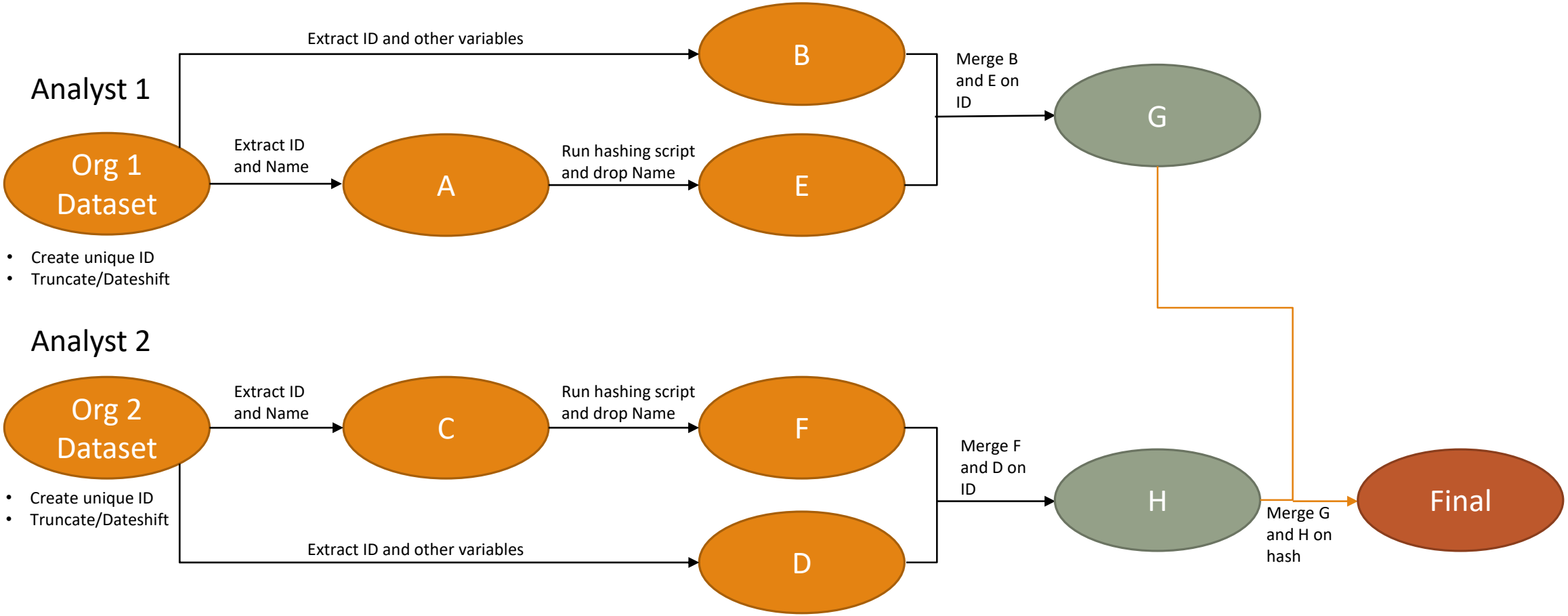
Protocol (from previous project)

- Analyst 1 handles sensitive dataset from Organization 1
  - Create a unique ID for each row
  - Truncate and Date-shift as necessary
  - Extract the name field and ID field into a separate dataset (Dataset A)
  - Extract the ID field and the other variables of interest into a separate dataset (Dataset B)
- Analyst 1 then communicates the salt and the value of the date-shift to Analyst 2 securely
- Analyst 2 handles sensitive Dataset from Organization
  - Perform the same steps as the dataset from Organization 1 to get Dataset C and Dataset D
- Analyst 1 runs the hashing script on Dataset A, drops the name column (Dataset E)
- Analyst 2 runs the hashing script on Dataset C, drops the name column (Dataset F)

Protocol (from previous project)

- Analyst 1 merges Dataset E with Dataset B using the unique ID (Dataset G)
- Analyst 2 merges Dataset F with Dataset D using the unique ID (Dataset H)
- Now, Dataset G is a dataset from Organization 1 with hashed names and Dataset H is a dataset from Organization 2 with hashed names
- Finally, Analyst 1 securely transfers Dataset G to Analyst 2, who can then perform the merging using the hashed names!

Protocol (from previous project)



Code Flow (Hashing)

- run script with **python gen_hash.py [csv filename] 150**
- code reads in the csv, stores name column and other variables into separate lists
  - cleans the names by lowercasing and removing all spaces
- code splits each name into **2-shingles**
- for each 2-shingle, the **salt is appended to the front and back, the iteration number, and then a hash value is obtained**
- in each iteration till 150, we take the **minimum of the these values**
- we end up with a series of **150 minhashes, which we call the signature**.
- code then outputs the names, cleaned names, the hash signature, and other relevant variables into a csv
- **remember to drop the names afterwards!** They are there just for checking purposes!

Action Points (Hashing)

```
38  class MinHashSignature(Signature):
39      """
40      Creates signatures for sets/tuples using minhash.
41      """
42
43      def hash_functions(self):
44          """
45          Return dim different hash functions
46          """
47          def hash_factory(n):
48              # load external text file as salt
49              with open('condiments.txt') as bottle:
50                  salt = bottle.readline()
51              return lambda x: int(hashlib.md5(salt.encode() + str(n).encode() + str(x).encode() + salt.encode()).hexdigest()[:12],16)
52          return [ hash_factory(_) for _ in range(self.dim) ]
53
```

Change filename for salt as necessary, and also the way the salt is read from the file – here the code assumes the salt is in a external text file that contains a single word on a single line.

Action Points (Hashing)

```
63  def hash_file(filename, width):
64      '''
65      Main function that hashes the names.
66      '''
67      hashes=[]
68      strings = []
69      strings_c= []
70      dobs = []
71      unids = []
72      fullhashes=[]
73
```

Change/add/remove lists as necessary to store relevant variables from dataset

Action Points (Hashing)

```
88      with open(filename) as input:
89          reader = csv.reader(input)
90          count = 0
91          blocks = 0
92          ceil = math.ceil(row_count/100)
93          next(reader)
94          for row in reader:
95              # keep original names, dobs, and unids for later merging
96              strings.append(row[0])
97              unids.append(row[1])
98              dobs.append(row[2])
99              # strip extraneous characters and clean to lower case
100             string_c = re.sub('[^a-zA-Z1-9]+', '', row[0]).strip().lower()
101             string_set = str(get_k_shingles(string_c, k=2))
102             hasher = MinHashSignature(int(width))
103             outie = hasher.sign(string_set)
104
```

Change row indexes as necessary and “append” the desired variable to the correct list: E.g. if gender is a desired variable and it is on the fourth column of the dataset, then add ‘genders.append(row[3])’. Don’t forget to create the ‘gender’ list first!

Action Points (Hashing)

```
123     # tabulate all required columns
124     fullcrosswalk=zip(strings,strings_c,hashes,dobs,unids)
125
126     # write csv
127     with open('hashed_' + filename ,'w', newline = '') as out:
128         csv_out=csv.writer(out)
129         csv_out.writerow(['Name','Cleaned Name','Hash', 'DOB', 'UNID'])
130         for row in fullcrosswalk:
131             csv_out.writerow(row)
```

Add all lists to be included in the output csv here, in the order you want to columns to be.

Add all the column names for the output csv here, in order, as well!

Code Flow (Merging)

- run R script after **changing path and filenames**
- the script reads **all the csvs in the directory (if data is subsetted)**, applies merging to each one in turn. The file suffixes are extracted for purposes of naming the output
- the distance threshold is defined in the beginning (currently 0.9)
- 2 dataframes are created for the separate methods (**remember to change variable names as necessary**)
- **Method 1**: Top match for every row, regardless of threshold.
- **Method 2**: Top match for every row, only if distance score meets threshold criteria >= 0.9
- Script applies modified Jaccard similarity measure (intersection divided by length), to compare each name in the first dataset to every name in the second dataset
- For each file, the script outputs “match_trial_[method][filesuffix].csv” as the matched file, along with some statistics regarding match numbers

Action Points (Merging)

```r
# Change working directory to where subset files are
setwd("")

library(dplyr)
library(lubridate)


############## Get all subsets in the directory (change path of files as necessary)
dat1_path <- ""
dat2_path <- ""
dat1_subsets <- list.files(dat1_path)
dat2_subsets <- list.files(dat2_path)
```

Change directory/paths to datasets

Action Points (Merging)

```
46  # Create dataframes for each of the methods
47  n_matched_M1 <- data.frame(hash_ccac = character(),
48                             hash_cps = character(),
49                             DOB_ccac = character(),
50                             DOB_cps = character(),
51                             Gender_ccac = character(),
52                             Gender_cps = character(),
53                             CCAC_Index = character(),
54                             CPS_ID = character(),
55                             similarity = numeric(),
56                             stringsAsFactors = FALSE)
57
58  n_matched_M2 <- data.frame(hash_ccac = character(),
59                               hash_cps = character(),
60                               DOB_ccac = character(),
61                               DOB_cps = character(),
62                               Gender_ccac = character(),
63                               Gender_cps = character(),
64                               CCAC_Index = character(),
65                               CPS_ID = character(),
66                               similarity = numeric(),
67                               stringsAsFactors = FALSE)
```

Change/add variable names to output dataframes as necessary.

Action Points (Merging)

```
90        # Method 1
91        if(nrow(sub1) != 0){
92          for(j in 1:length(sub1[,1])){
93            n_matched_M1[nrow(n_matched_M1) + 1, ] <- c(n_ccac[i, "Hash"], sub1[j,"Hash"], n_ccac[i, "DOB"],
94                                                       sub1[j, "DOB"], n_ccac[i, "G"], sub1[j, "gender"],
95                                                       n_ccac[i, "CCAC.Index"], sub1[j, "CPS.ID"], sub1[j, "jaccard"])
96          }
97        }
98
99        # Method 2
100       if(nrow(sub2) != 0){
101         sub3 <- sub2[which.max(sub2$jaccard),]
102         for(k in 1:length(sub3[,1])){
103           n_matched_M2[nrow(n_matched_M2) + 1, ] <- c(n_ccac[i, "Hash"], sub3[k,"Hash"], n_ccac[i, "DOB"],
104                                                      sub3[k, "DOB"], n_ccac[i, "G"], sub3[k, "gender"],
105                                                      n_ccac[i, "CCAC.Index"], sub3[k, "CPS.ID"], sub3[k, "jaccard"])
```

Align original dataframe variables to the new dataframe variables.