Stat 243: Problem Set 1

Eugene Yedvabny

09/12/2014

Question 1

I've filled out the class survey, but for the sake of completeness, here's a brief summary about me:

- *Name*: Eugene Yedvabny
- *Email*: eyedvabny@berkeley.edu
- *Department*: Chemistry
- *Program*: 3rd-year PhD, but leaving the program
- *Stats Experience*: Brief, unfortunately. A 'baby stats' course in undergrad on probablity theory and statistics. Two graduate-level courses at Berkeley on Statistical Mechanics. Currently co-enrolled in Stat 215, Statistical Models. So I know my way around but am still very much a beginner.
- *Programming Experience*: Fairly extensive. I have a minor in CS, mostly in systems programming. For research here I program in C, C++ and do a lot of analysis in Python (SciPy stack). I took the Johns Hopkins Coursera track on Data Science, and that's my level of exposure to R. I used to know Java but it's been a few years since I programmed in that.
- *Reasons for taking the class*: I've decided to leave my research program in computational chemistry because I realized I really don't like it. So for my last semester at Berkeley I want to study up on mathematical/statistical topics that both interest me and might help me land a compelling job in industry.

Question 2

I am not going to be using the SCF VM since I already have a Fedora VM I use for classes. But I've downloaded the SCF OVA and poked around: by default it's set to **1 processor** with **1 GB of RAM**.

Question 3

a)

Let's download the FEC campaign expenditure data in CSV format. The link was manually obtained from the FEC website

```
wget http://www.fec.gov/data/CandidateSummary.do?format=csv -O CandidateSummary.csv
```

I understand that we're not required to use Sed or Awk, but frankly, they are so much nicer than piped cut. So pardon the liberty of using both tools from the get-go. First priority is to do some cleanup (from 3-a-ii but might as well do them here). Thank you Chris for the Sed command. Also most of this came from Googling; I don't remember most of Sed/Awk/Grep syntax off the top of my head.

```
sed -i -e '2,$s/\([^",]\),/\1/g' -e 's/["]//g' CandidateSummary.csv
```

I know that **field 4 is the office designation** and **field 7 is the party affiliation** from looking at the metadata. If that's not available the following code will return which column I would need to look at (still need to know the column names though).

```
head -n 1 CandidateSummary.csv | sed 's/,/\n/g' | grep -n "^can_par_aff$\|^can_off$"
```

```
4:can_off
7:can_par_aff
```

Looks like the columns are aligned as designed in the metadata. We can proceed with processing of candidates. The code below will separate the downloaded CSV by `,` and then populate the files with Senate Democrats and Republicans.

```
awk '
  BEGIN {
    FS = ","
  }
  {
    if ($4 == "S") {
      if ($7 == "DEM")
        print $0 > "dem_sens.csv";
      else if ($7 == "REP")
        print $0 > "rep_sens.csv";
    }
  }' CandidateSummary.csv
```

Let's check how many Dems and Reps ran for Senate in 2014:

```
NUM_DEMS=$(head -n -1 dem_sens.csv | wc -l)
NUM_REPS=$(head -n -1 rep_sens.csv | wc -l)
echo "There are" $NUM_DEMS "democrats and" $NUM_REPS "republicans"
```

```
There are 92 democrats and 195 republicans
```

These are consistent with the numbers reported by the online tool, so it looks like the above command worked as intended. Now let's extract the total *individual* contributions. The column we want is **#16**, `ind_con`.

```
for party in {dem,rep}
do
  if [ $party = "dem" ]
  then
    echo "Democratic Contributions"
  else
    echo "Republican Contributions"
  fi

  cut -d "," -f 3,5,16 ${party}_sens.csv | sort -t "," -k 3 -n -r |
  awk '
    BEGIN {
```

```
      FS = ",";
      printf("%25s %5s %12s\n","Name","State","Contribution");
    }
    NR < 6 {
      printf("%25s %5s %12.0f\n",$1,$2,$3);
    }'
  echo ""
done
```

```
Democratic Contributions
                     Name State Contribution
    MARKEY EDWARD JOHN MR    MA     14668778
            BOOKER CORY A    NJ     14128197
              HAGAN KAY R    NC     10456106
 GRIMES ALISON  LUNDERGAN    KY     10290296
       NUNN MARY MICHELLE    GA      8016437

Republican Contributions
                     Name State Contribution
          MCCONNELL MITCH    KY      8991643
              CORNYN JOHN    TX      6903804
            COTTON THOMAS    AR      5558956
          CASSIDY WILLIAM    LA      5186351
      GRAHAM LINDSEY OLIN    SC      5104768
```

Bash code chunks break on in-line comments, so here's a brief description of above code:

- For each party file
  - Print out an appropriate header
  - Use cut to extract only the name, state, and contribution
  - Pipe into sort to sort by descending order on contribution
  - Pipe into awk for pretty-printing into a table
  - Wrap up with an empty line to separate the two parties

b)

The first step here is to download the two zip files and extract the necessary text files.

```
wget ftp://ftp.fec.gov/FEC/2014/cn14.zip
unzip cn14.zip
wget ftp://ftp.fec.gov/FEC/2014/indiv14.zip
unzip indiv14.zip
```

As it turns out, Bash functions do not carry over from chunk to chunk, so as a work-around I am going to write the functions to a file and then source them in later chunks when needed. A bit hacky but works.

```
cat << EOF > get_committee_id.sh
get_committee_id() {

  if [ "\$#" -lt 1 ]
  then
```

```
    echo "Please enter the candidate's last name and optional first name"
    exit 1
  fi

  awk -v name="\$1" '
    BEGIN {
      FS = "|";
      IGNORECASE = 1;
    }
    {
      if (\$2 ~ name)
        print \$10;
    }' cn.txt | uniq
}
EOF
```

The function takes last name and optional first name of a candidate and returns their Committee ID from the downloaded cn.txt. If the first name is provided, it has to come in quotations with the last name so it is read as a single argument. The function is saved into a file and can be sourced from future bash instances. The shell variables have to be escaped so cat doesn't try to fill them from the local environment. Knitr actually has a cat engine that can do the text saving, but it's not as transparent since the reader won't see which file is being written. For illustrative purposes, let's get the comittee ID for Mitch McConnell.

```
source get_committee_id.sh

echo "Without first name:" $(get_committee_id "mcconnell")
echo "With first name:" $(get_committee_id "mcconnell, mitch")
```

```
Without first name: C00426130 C00193342
With first name: C00193342
```

Now we can use the committee IDs to count the number of contributions. The FEC Committee ID can appear in Column 1 - Filer Identification Number - or Column 16 - Other Identification Number. We should only look at the former. Let's write a function to count the individual contributions.

```
cat << EOF > get_num_contributions.sh
get_num_contributions() {

  if [ "\$#" -lt 1 ]
  then
    echo "Please enter the candidate's last name, optional first name, and optional state"
    exit 1
  fi

  CMID=\$(get_committee_id "\$1")
  cut -d "|" -f 1,10 itcont.txt | egrep -c "\$CMID.*\$2"
}
EOF
```

The above function is again saved to a file. It will crash if the required internal function, get_committee_id, is not present. Not very safe, but I'd rather source the two files together than source one from the other. Let's see how many times Mitch McConnell got paid in 2013-2014

```
source get_committee_id.sh
source get_num_contributions.sh

MC_TOT=$(get_num_contributions "mcconnell, mitch")
MC_CA=$(get_num_contributions "mcconnell, mitch" "CA")
GR_TOT=$(get_num_contributions "grimes, alison")
GR_CA=$(get_num_contributions "grimes, alison" "CA")

echo "In 2013-2014 election year"
echo "Mitch McConnell received $MC_TOT contributions, of them $MC_CA in California"
echo "Alison Grimes received $GR_TOT contributions, of them $GR_CA in California"
```

```
In 2013-2014 election year
Mitch McConnell received 6751 contributions, of them 288 in California
Alison Grimes received 6452 contributions, of them 1026 in California
```

Hmm, it seems California prefers Alison Grimes over Mitch McConnell, though the latter has more contributions in total.

c)

The bulk of the work is already complete as part of **b** so we just need to loop through a list of names. I'm from Boston, so let's look at candidates for MA senate.

```
source get_committee_id.sh
source get_num_contributions.sh

grep "|MA|S|" cn.txt | cut -d "|" -f 2 > mass_sen_names.txt

echo "In 2013-2014 election year, Senate candidates from Massachusetts received:"
while read name
do
  printf "%25s: %7s donations\n" "$name" $(get_num_contributions "$name")
done < mass_sen_names.txt
```

```
In 2013-2014 election year, Senate candidates from Massachusetts received:
      SULLIVAN, MICHAEL J:     319 donations
         ROBINSON, JACK E:       4 donations
          COAKLEY, MARTHA:       2 donations
         LYNCH, STEPHEN F:    2499 donations
             KHAZEI, ALAN:      12 donations
           BROWN, SCOTT P:     359 donations
       KENNEDY, JOSEPH L:       0 donations
   MASSIE, ROBERT KINLOCH:      0 donations
            WARREN, SETTI:      75 donations
       WARREN, ELIZABETH:      808 donations
            MONDESIR, ENO:       0 donations
   MARKEY, EDWARD JOHN MR:   11017 donations
            KERRY, JOHN F:     175 donations
          GOMEZ, GABRIEL:     2965 donations
            BIELAT, SEAN:       3 donations
            WINSLOW, DAN:     377 donations
```

```
           INMAN, J MARK:       1 donations
     HEOS, RICHARD ALFRED:      1 donations
             HERR, BRIAN:      61 donations
   ADDIVINOLA, FRANK J JR:     51 donations
           SKARIN, BRUCE:       1 donations
 UNDERWOOD, ROBERT JOSEPH:      0 donations
```

Question 4

The easiest solution for scraping the site would be to leverage wget in recursive mode:

```
wget -r -nd -A *.txt http://www1.ncdc.noaa.gov/pub/data/ghcn/daily/
```

For some reason it keeps getting stuck on network requests, so parsing HTML was the next best solution.

```
curl -s http://www1.ncdc.noaa.gov/pub/data/ghcn/daily/ |
  grep -o -E 'href=.*\.txt"' |
  cut -d '"' -f2 |
  wget -B http://www1.ncdc.noaa.gov/pub/data/ghcn/daily/ -P noaa -nv -i -
```

```
ls noaa
```

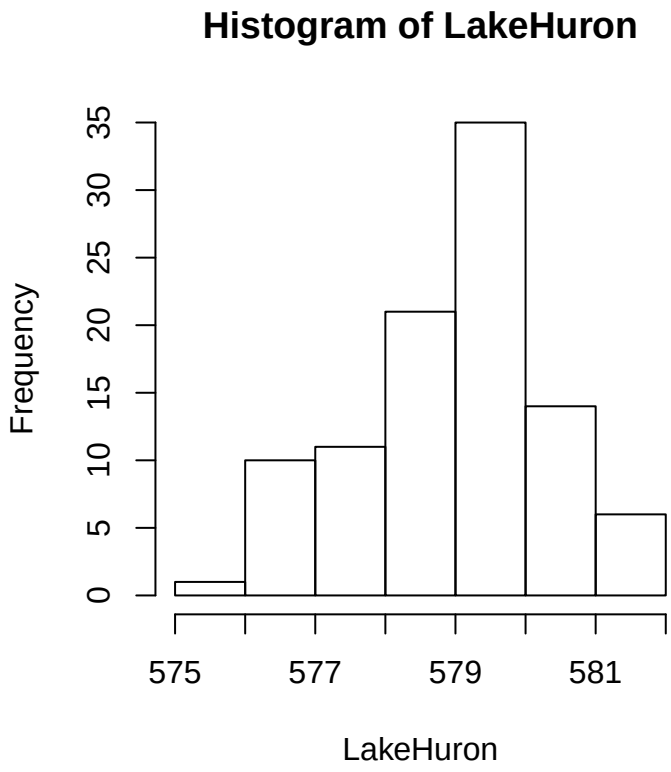
```
ghcnd-countries.txt
ghcnd-inventory.txt
ghcnd-states.txt
ghcnd-stations.txt
ghcnd-version.txt
readme.txt
status.txt
```

The above code gets the index html, parses out all .txt files, and then pipes that file list into wget for downloading. Wget is partially silenced to only report the file name. I am not actually certain why nothing is printed to the final PDF; the filenames do appear in my shell. Output of ls is provided for verification.

Question 5

I've written this whole document in RMarkdown, but for the sake of completness:

```
hist(LakeHuron)
```




```
lowHi <- c(which.min(LakeHuron), which.max(LakeHuron))
yearExtrema <- attributes(LakeHuron)$tsp[1] - 1 + lowHi
```

Above you can see the histogram of water level in Lake Huron between 1876 and 1964