Modeling Social Data: Homework 1

Anshul Doshi

Anshul Doshi (ad3222) Modeling Social Data Assignment: 1

Problem 1: Counting Scenarios

We are given a dataset that lists phonecalls between pairs of people, listing caller, callee, times of phone calls and duration. Our goal is to compute the total amount of time between each pair of people.

a.) Compute these total statistics for a small town of 100,000 people who call on avaerage 50 people.

Our data here is relatively small, containing at most 5 million data points. This can be handled by our normal computers. Here, we will use the split/apply/combine paradigm to compute the total duration of a call between all the pairs.

The relevant data for this exercise will be of the form of a pair of numbers that maps to a duration of call value.

In our tidyverse terms: we can group by both caller and callee and then summarise the data addding a total column which would be sum(duration of call). Essentially this could look like:

```
dataset %>%
  group_by(caller,callee) %>%
  summarise(total=sum(duration))
```

In more detail, with loops this is basically the simple split/apply/combine algorithm:

```
for each entry as (pair of callers, duration of call) do:
  place entry in bucket for corresponding pair of callers
end for
for each pair do:
  apply the sum function over the duration values
  output pair with respective total duration
end for
```

Note: this algorithm may not be optimal as we are doing two passes over the values. The streaming version outlined in the next part can work here and is faster

b.) Large city of 10,000,000 people, each calling 100 poeple.

Here we have about 10,000,000 people with 100 calls each -> leads to 1 billions observations. This size, although can be done on laptops, can lead to errors and be very slow.

Thus, we can need a streaming algorithm and can use a hash table. The algorithm could be something like:

1. Since there are 10,000,000 people, there can be at most n(n-1)/2 pairs. However, each person calls an average of 100 people so our size for the table can be (10,000,000)*100 =1,000,000,000. However, since a billion may be too much for memory, we can decrease the load factor and give up some perfromance if we need to rehash keys and initialize a smaller hash table.
2. Construct a hashing function that hashes two phone-numbers to an unique index into the table. For example, we can split each phone number in pairs of two 5-digit numbers and sum up these two numbers. For example, 436-555-4601 would be 43655+54601. We can multiply this result from the two phone numbers and then mod this by the size of our table.
3. Go through each observation in the input data. Hash the two numbers for our key and store the duration of the call as the value. If the key already exists, we add the duration of the call to the existing value, keeping a running total of the duration of call between a pair

```
for each entry in (pair, duration) do:
  hash pair to map caller/callee to duration of call
  if key exists
    update duration to reflect total duration
  else
    store the pair as the key mapping to that calls duration
  End if
End for
```

This only requires one pass with the data so this will work better for such a large dataset. Thus, at the end we will have a hash table containing each pair and their total duration on the phone

c.) Suppose the dataset is a call log of a nation of 300,000,000 people, each of whom calls 200 people on average. Please describe how you would compute the statistics.

This time, the dataset is way too big for our single computers to handle. One way to approach this is to use parrallel computing and the MapReduce paradigm to do the same type of split/apply/combine delineated in the above two parts.

It takes approx. 4 hours to read 1TB of data on a commodity hard disk (225GB/hr). Thus we can use this idea of a distributed solution through hadoop and MapReduce to approach this.

To approach this, we will break this large dataset into smaller parts, solve in parrallel, and combine the results.

As the programmer, we specify the map and reduce functions.

Mapper: Input is a single entry (caller, callee, duration of call)

```
def(mapper):
  combine caller and callee as a tuple
  output (pair, duration)
```

Shuffle: MapReduce will automatically select how many computers to use and how to split up the data. It will also shuffle/collect all intermediate pairs and hash them, then performing merge sort such that each reducer can end up all entries of the same pair.

Reducer: transform records of a single pair to output

```
def reducer(pair, records):
  total duration = 0
  for record in records:
    total duration += record.duration
  End for
  output( (pair, total duration) )
```

With this paradigm, we can evuluate this large amount of data.

Problem 2: Solutions Attached in .txt file but also here

Part 1: citibike.sh - bash commands

1.) count the number of unique stations

```bash
cut -d, -f8 201608-citibike-tripdata.csv | tail -n +2 | sort | uniq > station
cut -d, -f4 201608-citibike-tripdata.csv | tail -n +2 | sort | uniq >> station
cat station | sort | uniq | wc -l
```

```
##      582
```

2.) Count number of unique bikes

```
cut -d, -f12 201608-citibike-tripdata.csv | tail -n+2 | sort | uniq | wc -l
```

```
##     9284
```

3.) Count the number of trips per day

```
cut -d, -f2 201608-citibike-tripdata.csv | cut -d' ' -f1 | sort -t"/" -k2 -g | uniq -c | head
```

```
##    1 "starttime"
## 49401 "8/1/2016
## 56764 "8/2/2016
## 57003 "8/3/2016
## 56604 "8/4/2016
## 53297 "8/5/2016
## 44213 "8/6/2016
## 43631 "8/7/2016
## 52262 "8/8/2016
## 56112 "8/9/2016
```

4.) find the day with most rides

```
cut -d, -f2 201608-citibike-tripdata.csv | cut -d' ' -f1 | sort -t"/" -k2 -g | uniq -c | sort | tail -1
```

```
## 58674 "8/23/2016
```

5.) find the day with the least rides

```
cut -d, -f2 201608-citibike-tripdata.csv | cut -d' ' -f1 | sort -t"/" -k2 -g | uniq -c | sort | head -2
```

```
##    1 "starttime"
## 32961 "8/14/2016
```

6.) find the id of the bike with the most rides

```
cut -d, -f12 201608-citibike-tripdata.csv | sort | uniq -c | sort | tail -1
```

```
##  438 "25384"
```

7.) count the number of rides by gender and birth year

```
cut -d, -f14,15 201608-citibike-tripdata.csv | sort -k1 | uniq -c | head
```

```
## 214534 "","0"
##   47 "","1"
##   43 "","2"
##   90 "1885","0"
##   36 "1888","1"
##    2 "1893","2"
##    5 "1894","2"
##    8 "1896","1"
##   68 "1899","1"
##    1 "1900","0"
```

8.) count the number of trips that start on cross streets that both contain numbers (e.g., "1 Ave & E 15 St", "E 39 St & 2 Ave", ...)

```
cut -d, -f5 201608-citibike-tripdata.csv | grep '.*\d\+.*&.*\d\+.*' | sort| uniq | wc -l
```

```
##      147
```

9.) Compute average trip duration

```
cut -d, -f1 201608-citibike-tripdata.csv | sed -e 's/^"//' -e 's/"$//' | awk '{sum += $1} END {print sum
```

```
## 980.285
```

Part b: citibike.R

```r
library(tidyverse)
```

```
## Loading tidyverse: ggplot2
## Loading tidyverse: tibble
## Loading tidyverse: tidyr
## Loading tidyverse: readr
## Loading tidyverse: purrr
## Loading tidyverse: dplyr

## Conflicts with tidy packages ----------------------------------------------

## filter(): dplyr, stats
## lag():    dplyr, stats
```

```r
library(lubridate)
```

```
##
## Attaching package: 'lubridate'

## The following object is masked from 'package:base':
##
##     date
```

```r
########################################
# READ AND TRANSFORM THE DATA
########################################

# read one month of data
trips <- read_csv('201608-citibike-tripdata.csv')
```

```
## Parsed with column specification:
## cols(
##   tripduration = col_integer(),
##   starttime = col_character(),
##   stoptime = col_character(),
##   `start station id` = col_integer(),
##   `start station name` = col_character(),
##   `start station latitude` = col_double(),
##   `start station longitude` = col_double(),
##   `end station id` = col_integer(),
##   `end station name` = col_character(),
##   `end station latitude` = col_double(),
##   `end station longitude` = col_double(),
##   bikeid = col_integer(),
##   usertype = col_character(),
##   `birth year` = col_integer(),
##   gender = col_integer()
## )
```

```r
# replace spaces in column names with underscores
names(trips) <- gsub(' ', '_', names(trips))

# convert dates strings to dates
trips <- mutate(trips, starttime = mdy_hms(starttime), stoptime = mdy_hms(stoptime))

# recode gender as a factor 0->"Unknown", 1->"Male", 2->"Female"
trips <- mutate(trips, gender = factor(gender, levels=c(0,1,2), labels = c("Unknown","Male","Female")))


########################################
# YOUR SOLUTIONS BELOW
########################################

# count the number of trips (= rows in the data frame)
nrow(trips)
```

```
## [1] 1557663
```

```r
# find the earliest and latest birth years (see help for max and min to deal with NAs)
max(trips$birth_year, na.rm = TRUE)
```

```
## [1] 2000
```

```r
min(trips$birth_year, na.rm = TRUE)
```

```
## [1] 1885
```

```r
# use filter and grepl to find all trips that either start or end on broadway
filter(trips, grepl("Broadway", start_station_name, ignore.case = TRUE)
       | grepl("Broadway", end_station_name, ignore.case = TRUE)) %>%
  select(start_station_name, end_station_name)
```

```
## # A tibble: 285,606 × 2
##          start_station_name          end_station_name
##                       <chr>                     <chr>
## 1        Broadway & E 14 St         E 7 St & Avenue A
## 2           5 Ave & E 29 St        Broadway & W 49 St
## 3            Great Jones St        E 17 St & Broadway
## 4  Franklin St & W Broadway            W 18 St & 6 Ave
## 5        Broadway & E 14 St            E 19 St & 3 Ave
## 6  Vesey Pl & River Terrace      Fulton St & Broadway
## 7        Broadway & E 14 St         E 2 St & Avenue B
## 8        Broadway & W 55 St            9 Ave & W 22 St
## 9     Broadway & Whipple St Putnam Ave & Nostrand Ave
## 10       Broadway & W 29 St     Allen St & Stanton St
## # ... with 285,596 more rows
```

```r
# do the same, but find all trips that both start and end on broadway
filter(trips, grepl("Broadway", start_station_name, ignore.case = TRUE)
       & grepl("Broadway", end_station_name, ignore.case = TRUE)) %>%
  select(start_station_name, end_station_name)
```

```
## # A tibble: 21,664 × 2
##       start_station_name         end_station_name
##                    <chr>                    <chr>
## 1     Broadway & W 49 St       Broadway & W 49 St
```

```
## 2     Broadway & W 49 St       Broadway & W 49 St
## 3  Broadway & Battery Pl       Broadway & W 32 St
## 4  Broadway & Battery Pl    Broadway & Battery Pl
## 5  Liberty St & Broadway     Fulton St & Broadway
## 6     Broadway & E 14 St       Broadway & E 22 St
## 7     E 17 St & Broadway       E 11 St & Broadway
## 8     Broadway & W 51 St       Broadway & W 53 St
## 9     Broadway & W 29 St Washington Pl & Broadway
## 10    Broadway & W 29 St       Broadway & W 24 St
## # ... with 21,654 more rows
```

```r
# find all unique station names
stations <- trips$start_station_name
stations <- append(stations, trips$end_station_name, after=length(stations))
head(unique(stations))
```

```
## [1] "Avenue D & E 3 St"              "Broadway & E 14 St"
## [3] "Metropolitan Ave & Bedford Ave" "E 10 St & 5 Ave"
## [5] "LaGuardia Pl & W 3 St"          "Grand St & Havemeyer St"
```

```r
# count the number of trips by gender
select(trips, gender) %>%
  summary()
```

```
##      gender
##  Unknown:218533
##  Male   :999541
##  Female :339589
```

```r
# compute the average trip time by gender
trips %>%
  group_by(gender) %>%
  summarize(count=n(),
            avg_trip_duration=mean(tripduration),
            sd_trip_duration=sd(tripduration))
```

```
## # A tibble: 3 × 4
##    gender  count avg_trip_duration sd_trip_duration
##    <fctr>  <int>             <dbl>            <dbl>
## 1 Unknown 218533         1846.8789        18880.278
## 2    Male 999541          799.7071         6891.169
## 3  Female 339589          954.1263         9505.789
```

```r
# comment on whether there's a (statistically) significant difference
# By the rule of thumb, since the error bars are well separated, and they do
# overlap, the difference between the two means is not statistically significant.

# find the 10 most frequent station-to-station trips
ungroup(trips)
```

```
## # A tibble: 1,557,663 × 15
##    tripduration           starttime            stoptime start_station_id
##           <int>              <dttm>              <dttm>            <int>
## 1           288 2016-08-01 00:01:22 2016-08-01 00:06:11              302
## 2           457 2016-08-01 00:01:43 2016-08-01 00:09:21              285
## 3           278 2016-08-01 00:02:10 2016-08-01 00:06:49              539
## 4           862 2016-08-01 00:02:13 2016-08-01 00:16:36              280
```

```
## 5          407 2016-08-01 00:02:21 2016-08-01 00:09:09               161
## 6          321 2016-08-01 00:03:33 2016-08-01 00:08:55               471
## 7          122 2016-08-01 00:04:16 2016-08-01 00:06:19              3101
## 8          770 2016-08-01 00:04:21 2016-08-01 00:17:11               434
## 9          230 2016-08-01 00:04:31 2016-08-01 00:08:21              3265
## 10         422 2016-08-01 00:05:27 2016-08-01 00:12:30               252
## # ... with 1,557,653 more rows, and 11 more variables:
## #   start_station_name <chr>, start_station_latitude <dbl>,
## #   start_station_longitude <dbl>, end_station_id <int>,
## #   end_station_name <chr>, end_station_latitude <dbl>,
## #   end_station_longitude <dbl>, bikeid <int>, usertype <chr>,
## #   birth_year <int>, gender <fctr>
```

```
trips %>%
  group_by(start_station_name,end_station_name) %>%
    summarize(count=n()) %>%
    arrange(desc(count)) %>%
    head(n=10)
```

```
## Source: local data frame [10 x 3]
## Groups: start_station_name [6]
##
##         start_station_name        end_station_name count
##                      <chr>                   <chr> <int>
## 1   Central Park S & 6 Ave  Central Park S & 6 Ave  1273
## 2   Central Park S & 6 Ave         5 Ave & E 88 St   914
## 3    Yankee Ferry Terminal   Yankee Ferry Terminal   757
## 4         Soissons Landing   Yankee Ferry Terminal   747
## 5    Yankee Ferry Terminal        Soissons Landing   742
## 6        E 7 St & Avenue A  Cooper Square & E 7 St   710
## 7   Central Park S & 6 Ave         5 Ave & E 73 St   659
## 8  Centre St & Chambers St Centre St & Chambers St   647
## 9         Soissons Landing        Soissons Landing   636
## 10         12 Ave & W 40 St   West St & Chambers St   620
```

```
# find the top 3 end stations for trips starting from each start station
ungroup(trips)
```

```
## # A tibble: 1,557,663 × 15
##    tripduration           starttime            stoptime start_station_id
##           <int>              <dttm>              <dttm>            <int>
## 1           288 2016-08-01 00:01:22 2016-08-01 00:06:11               302
## 2           457 2016-08-01 00:01:43 2016-08-01 00:09:21               285
## 3           278 2016-08-01 00:02:10 2016-08-01 00:06:49               539
## 4           862 2016-08-01 00:02:13 2016-08-01 00:16:36               280
## 5           407 2016-08-01 00:02:21 2016-08-01 00:09:09               161
## 6           321 2016-08-01 00:03:33 2016-08-01 00:08:55               471
## 7           122 2016-08-01 00:04:16 2016-08-01 00:06:19              3101
## 8           770 2016-08-01 00:04:21 2016-08-01 00:17:11               434
## 9           230 2016-08-01 00:04:31 2016-08-01 00:08:21              3265
## 10          422 2016-08-01 00:05:27 2016-08-01 00:12:30               252
## # ... with 1,557,653 more rows, and 11 more variables:
## #   start_station_name <chr>, start_station_latitude <dbl>,
## #   start_station_longitude <dbl>, end_station_id <int>,
## #   end_station_name <chr>, end_station_latitude <dbl>,
## #   end_station_longitude <dbl>, bikeid <int>, usertype <chr>,
```

```
## #   birth_year <int>, gender <fctr>
```

```r
trips %>%
  group_by(start_station_name, end_station_name) %>%
  summarize(count=n()) %>%
  arrange(desc(count)) %>%
  mutate(station_rank=row_number()) %>%
  filter(station_rank<=3) %>%
  arrange(start_station_name)
```

```
## Source: local data frame [1,714 x 4]
## Groups: start_station_name [574]
##
##    start_station_name      end_station_name count station_rank
##                 <chr>                 <chr> <int>        <int>
## 1      1 Ave & E 16 St       E 23 St & 1 Ave   262            1
## 2      1 Ave & E 16 St       E 15 St & 3 Ave   250            2
## 3      1 Ave & E 16 St   E 20 St & FDR Drive   197            3
## 4      1 Ave & E 18 St    E 17 St & Broadway   149            1
## 5      1 Ave & E 18 St       E 15 St & 3 Ave   127            2
## 6      1 Ave & E 18 St    E 20 St & Park Ave   113            3
## 7      1 Ave & E 30 St Pershing Square North   142            1
## 8      1 Ave & E 30 St       W 31 St & 7 Ave   142            2
## 9      1 Ave & E 30 St       W 41 St & 8 Ave   122            3
## 10     1 Ave & E 44 St       1 Ave & E 68 St   107            1
## # ... with 1,704 more rows
```

```r
# find the top 3 most common station-to-station trips by gender
ungroup(trips)
```

```
## # A tibble: 1,557,663 × 15
##    tripduration           starttime            stoptime start_station_id
##           <int>              <dttm>              <dttm>            <int>
## 1           288 2016-08-01 00:01:22 2016-08-01 00:06:11              302
## 2           457 2016-08-01 00:01:43 2016-08-01 00:09:21              285
## 3           278 2016-08-01 00:02:10 2016-08-01 00:06:49              539
## 4           862 2016-08-01 00:02:13 2016-08-01 00:16:36              280
## 5           407 2016-08-01 00:02:21 2016-08-01 00:09:09              161
## 6           321 2016-08-01 00:03:33 2016-08-01 00:08:55              471
## 7           122 2016-08-01 00:04:16 2016-08-01 00:06:19             3101
## 8           770 2016-08-01 00:04:21 2016-08-01 00:17:11              434
## 9           230 2016-08-01 00:04:31 2016-08-01 00:08:21             3265
## 10          422 2016-08-01 00:05:27 2016-08-01 00:12:30              252
## # ... with 1,557,653 more rows, and 11 more variables:
## #   start_station_name <chr>, start_station_latitude <dbl>,
## #   start_station_longitude <dbl>, end_station_id <int>,
## #   end_station_name <chr>, end_station_latitude <dbl>,
## #   end_station_longitude <dbl>, bikeid <int>, usertype <chr>,
## #   birth_year <int>, gender <fctr>
```

```r
trips %>%
  group_by(gender, start_station_name, end_station_name) %>%
  summarize(count=n()) %>%
  arrange(desc(count)) %>%
  ungroup() %>%
  group_by(gender) %>%
```

```
  mutate(station_rank=row_number()) %>%
  filter(station_rank<=3) %>%
  arrange(gender)
```

```
## Source: local data frame [9 x 5]
## Groups: gender [3]
##
##    gender      start_station_name        end_station_name count
##    <fctr>                   <chr>                   <chr> <int>
## 1 Unknown  Central Park S & 6 Ave  Central Park S & 6 Ave   994
## 2 Unknown  Central Park S & 6 Ave          5 Ave & E 88 St   810
## 3 Unknown Centre St & Chambers St Centre St & Chambers St   582
## 4    Male        E 7 St & Avenue A  Cooper Square & E 7 St   521
## 5    Male   Pershing Square North          W 41 St & 8 Ave   498
## 6    Male   Pershing Square North    E 24 St & Park Ave S   443
## 7  Female        E 7 St & Avenue A  Cooper Square & E 7 St   180
## 8  Female         Soissons Landing   Yankee Ferry Terminal   134
## 9  Female   Yankee Ferry Terminal        Soissons Landing   132
## # ... with 1 more variables: station_rank <int>
```

```
# find the day with the most trips
# tip: first add a column for year/month/day without time of day (use as.Date or floor_date from the lu
trips %>%
  mutate(ymd=as.Date(starttime)) %>%
  group_by(ymd) %>%
  summarise(count=n()) %>%
  arrange(desc(count))
```

```
## # A tibble: 31 × 2
##           ymd count
##        <date> <int>
## 1  2016-08-23 58674
## 2  2016-08-24 58154
## 3  2016-08-30 57405
## 4  2016-08-31 57214
## 5  2016-08-03 57003
## 6  2016-08-02 56764
## 7  2016-08-04 56604
## 8  2016-08-09 56112
## 9  2016-08-22 54404
## 10 2016-08-17 53989
## # ... with 21 more rows
```

```
# compute the average number of trips taken during each of the 24 hours of the day across the entire mo
ungroup(trips)
```

```
## # A tibble: 1,557,663 × 15
##    tripduration           starttime            stoptime start_station_id
##           <int>              <dttm>              <dttm>            <int>
## 1           288 2016-08-01 00:01:22 2016-08-01 00:06:11              302
## 2           457 2016-08-01 00:01:43 2016-08-01 00:09:21              285
## 3           278 2016-08-01 00:02:10 2016-08-01 00:06:49              539
## 4           862 2016-08-01 00:02:13 2016-08-01 00:16:36              280
## 5           407 2016-08-01 00:02:21 2016-08-01 00:09:09              161
## 6           321 2016-08-01 00:03:33 2016-08-01 00:08:55              471
```

```
## 7           122 2016-08-01 00:04:16 2016-08-01 00:06:19             3101
## 8           770 2016-08-01 00:04:21 2016-08-01 00:17:11              434
## 9           230 2016-08-01 00:04:31 2016-08-01 00:08:21             3265
## 10          422 2016-08-01 00:05:27 2016-08-01 00:12:30              252
## # ... with 1,557,653 more rows, and 11 more variables:
## #   start_station_name <chr>, start_station_latitude <dbl>,
## #   start_station_longitude <dbl>, end_station_id <int>,
## #   end_station_name <chr>, end_station_latitude <dbl>,
## #   end_station_longitude <dbl>, bikeid <int>, usertype <chr>,
## #   birth_year <int>, gender <fctr>
```

```r
trips %>%
  mutate(ymd=as.Date(starttime)) %>%
  mutate(hour=hour(starttime)) %>%
  group_by(ymd,hour) %>%
  summarise(count=n()) %>%
  ungroup() %>%
  group_by(hour) %>%
  summarise(avg_trips=mean(count))
```

```
## # A tibble: 24 × 2
##     hour  avg_trips
##    <int>      <dbl>
## 1      0  472.58065
## 2      1  259.70968
## 3      2  151.06452
## 4      3   93.64516
## 5      4   92.77419
## 6      5  285.90323
## 7      6 1135.45161
## 8      7 2317.32258
## 9      8 4006.16129
## 10     9 3233.77419
## # ... with 14 more rows
```

```r
# what time(s) of day tend to be peak hour(s)?
ungroup(trips)
```

```
## # A tibble: 1,557,663 × 15
##    tripduration           starttime            stoptime start_station_id
##           <int>              <dttm>              <dttm>            <int>
## 1           288 2016-08-01 00:01:22 2016-08-01 00:06:11              302
## 2           457 2016-08-01 00:01:43 2016-08-01 00:09:21              285
## 3           278 2016-08-01 00:02:10 2016-08-01 00:06:49              539
## 4           862 2016-08-01 00:02:13 2016-08-01 00:16:36              280
## 5           407 2016-08-01 00:02:21 2016-08-01 00:09:09              161
## 6           321 2016-08-01 00:03:33 2016-08-01 00:08:55              471
## 7           122 2016-08-01 00:04:16 2016-08-01 00:06:19             3101
## 8           770 2016-08-01 00:04:21 2016-08-01 00:17:11              434
## 9           230 2016-08-01 00:04:31 2016-08-01 00:08:21             3265
## 10          422 2016-08-01 00:05:27 2016-08-01 00:12:30              252
## # ... with 1,557,653 more rows, and 11 more variables:
## #   start_station_name <chr>, start_station_latitude <dbl>,
## #   start_station_longitude <dbl>, end_station_id <int>,
## #   end_station_name <chr>, end_station_latitude <dbl>,
```

```
## #   end_station_longitude <dbl>, bikeid <int>, usertype <chr>,
## #   birth_year <int>, gender <fctr>
```

```
trips %>%
  mutate(ymd=as.Date(starttime)) %>%
  mutate(hour=hour(starttime)) %>%
  group_by(ymd,hour) %>%
  summarise(count=n()) %>%
  ungroup() %>%
  group_by(hour) %>%
  summarise(avg_trips=mean(count)) %>%
  arrange(desc(avg_trips))
```

```
## # A tibble: 24 × 2
##     hour avg_trips
##    <int>     <dbl>
## 1     17  4983.452
## 2     18  4914.677
## 3      8  4006.161
## 4     19  3592.258
## 5     16  3407.032
## 6      9  3233.774
## 7     15  2816.194
## 8     14  2669.839
## 9     13  2614.290
## 10    12  2532.355
## # ... with 14 more rows
```

```
#Peak hour is 17 => 5pm which makes sense as people are getting out of work and going home.
```

Problem 3: Eccentricity

The goal of this excercise is to graph the fraction of user satisfied vs. inventory size. Here, we say a user is p% satisfies if p% of the movies he/she rated is contained within an inventory size of the k most popular movies.

We are given a data set that lists all users and their movies they rated. We will be manipulating two data frames, one telling use the ranking of each movie and another telling how many users ranked that movie.

Algorithm idea:

1. A person is 100% satisfied if from all the movies they ranked, the least popular movie is at or less than inventory size k
2. Create a data frame ranking each movie by popularity
3. left_join these rankings into the data frame with all users and their ranked movies
4. summarise each user and the their worst ranked movie they rated
5. for each ranking, find amount of users who have a worst ranked movie as that ranking
6. bring that data back into the movie by popularity data frame and plot

Same idea for 90% but rather than worst ranked movie rated, we take count of all movies a user rated, multiply by 90% and round up. This is the index for their 90th percent worst ranked movie. Then algorithm is the same.

```
library(tidyverse)
library(scales)
```

```
##
## Attaching package: 'scales'
```

```
## The following object is masked from 'package:purrr':
##
##     discard
## The following objects are masked from 'package:readr':
##
##     col_factor, col_numeric
```

```r
########################################
# YOUR SOLUTION BELOW
########################################
setwd("~/Desktop/Spring2017/Modeling Social Data")

# Load the data file and add headers
header <- c('UserID', 'MovieID', 'Rating', 'Timestamp')
ratings <- read_csv('ratings.csv', col_names=header)
```

```
## Parsed with column specification:
## cols(
##   UserID = col_integer(),
##   MovieID = col_integer(),
##   Rating = col_double(),
##   Timestamp = col_integer()
## )
```

```r
head(ratings)
```

```
## # A tibble: 6 × 4
##   UserID MovieID Rating Timestamp
##    <int>   <int>  <dbl>     <int>
## 1      1     122      5 838985046
## 2      1     185      5 838983525
## 3      1     231      5 838983392
## 4      1     292      5 838983421
## 5      1     316      5 838983392
## 6      1     329      5 838983392
```

```r
# Show how many movies each user rated
usr_ratings <- ratings %>%
  group_by(UserID) %>%
  summarise(count=n()) %>%
  filter(count>=10)

# Rank movies in terms of popularity
ungroup(ratings)
```

```
## # A tibble: 10,000,054 × 4
##    UserID MovieID Rating Timestamp
##     <int>   <int>  <dbl>     <int>
## 1       1     122      5 838985046
## 2       1     185      5 838983525
## 3       1     231      5 838983392
## 4       1     292      5 838983421
## 5       1     316      5 838983392
## 6       1     329      5 838983392
## 7       1     355      5 838984474
## 8       1     356      5 838983653
```

```
## 9       1     362      5 838984885
## 10      1     364      5 838983707
## # ... with 10,000,044 more rows
```

```
movie_popularity <- ratings %>%
  group_by(MovieID) %>%
  summarise(count=n()) %>%
  arrange(desc(count)) %>%
  mutate(rank=row_number())

# Use left_join to see the ranking of each movie a user rated
usr_movies <- ratings %>%
  select(UserID, MovieID) %>%
  left_join(movie_popularity) %>%
  select(UserID, MovieID, rank) %>%
  arrange(UserID)
```

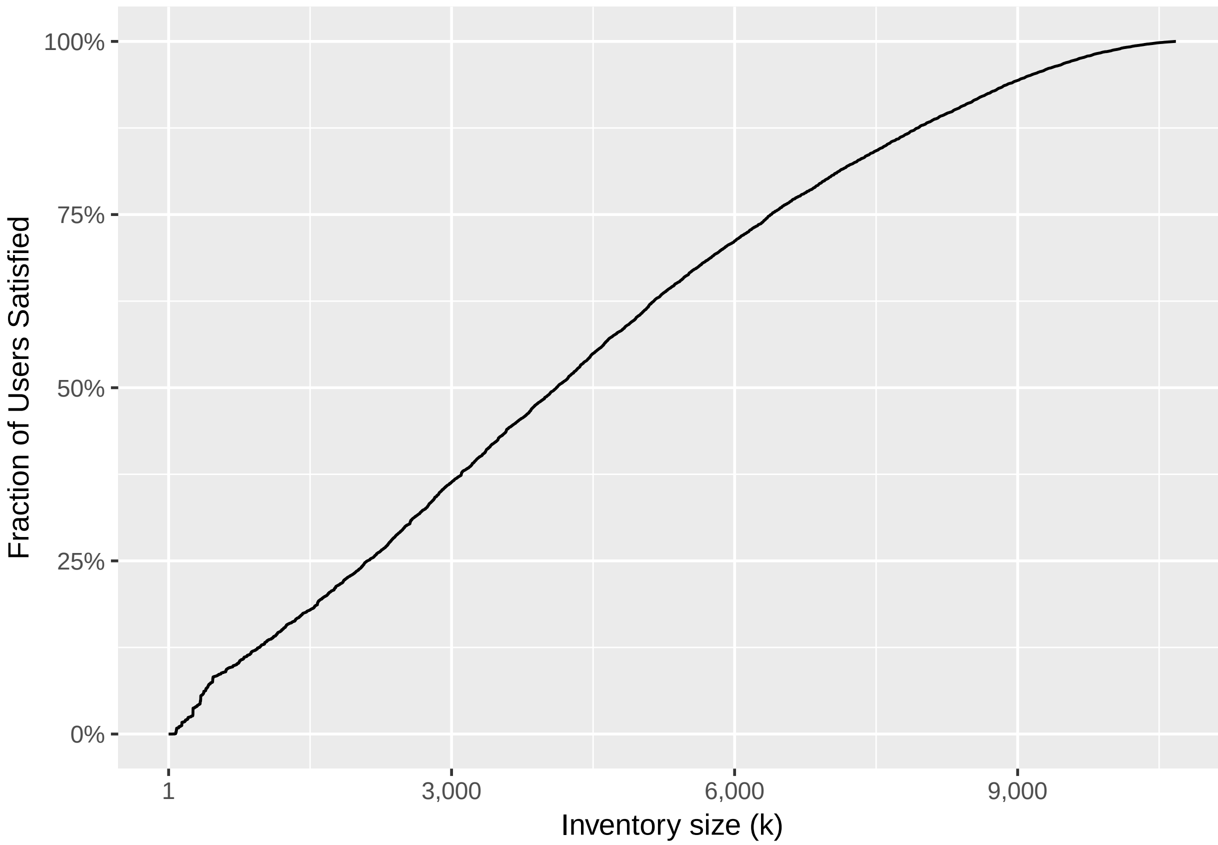
```
## Joining, by = "MovieID"
```

```
# Take the worst ranked movie for a user and summarise each movie rank
# with how many users have that ranking as their worst movie
usr_max_rank <- usr_movies %>%
  group_by(UserID) %>%
  summarise(max_ranking=max(rank)) %>%
  ungroup() %>%
  group_by(max_ranking) %>%
  summarise(num_users=n())

# bring in the num_user data into our movie rankings
movie_max_rank <- movie_popularity %>%
  left_join(usr_max_rank, by=c("rank" = "max_ranking"))

# convert NA's to 0
movie_max_rank[is.na(movie_max_rank)] <- 0

# plot fraction satisfied
movie_max_rank %>%
  mutate(frac_satisfied = cumsum(num_users) / sum(num_users) ) %>%
  ggplot(aes(x = rank, y = frac_satisfied)) +
  geom_line() +
  scale_x_continuous(label = comma, breaks = c(1, 3e3, 6e3, 9e3)) +
  scale_y_continuous(label = percent) +
  xlab('Inventory size (k)') +
  ylab('Fraction of Users Satisfied')
```




```r
# 90% satsfaction

# Instead of worst movie ranked, we take 90% of how many
# movies the user rated and find that kth ranked movie for them
ungroup(usr_movies)
```

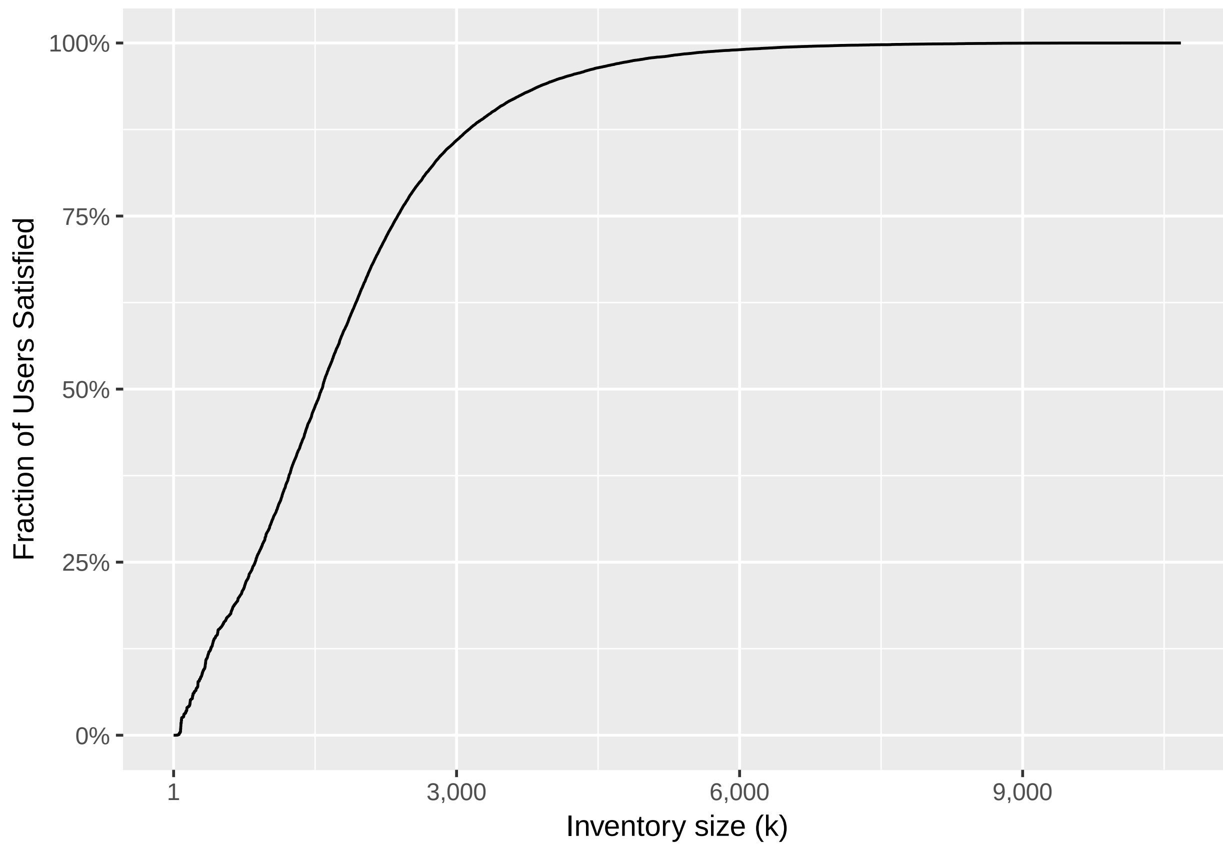
```
## # A tibble: 10,000,054 × 3
##    UserID MovieID  rank
##     <int>   <int> <int>
## 1       1     122  1058
## 2       1     185    77
## 3       1     231    47
## 4       1     292    61
## 5       1     316    40
## 6       1     329    59
## 7       1     355   427
## 8       1     356     2
## 9       1     362   620
## 10      1     364    31
## # ... with 10,000,044 more rows
```

```r
usr_nine_perc <- usr_movies %>%
  group_by(UserID) %>%
  arrange(UserID, rank) %>%
  summarise(my_count=n(), nine_perc_rank=rank[ceiling(.9*my_count)]) %>%
  ungroup() %>%
  group_by(nine_perc_rank) %>%
  summarise(num_users=n())
```

```r
# left_join to bring users into movie ranking data
movie_nine_rank <- movie_popularity %>%
  left_join(usr_nine_perc, by=c("rank" = "nine_perc_rank"))

movie_nine_rank[is.na(movie_nine_rank)] <- 0

# plot fraction satisfied
movie_nine_rank %>%
  mutate(frac_satisfied = cumsum(num_users) / sum(num_users) ) %>%
  ggplot(aes(x = rank, y = frac_satisfied)) +
  geom_line() +
  scale_x_continuous(label = comma, breaks = c(1, 3e3, 6e3, 9e3)) +
  scale_y_continuous(label = percent) +
  xlab('Inventory size (k)') +
  ylab('Fraction of Users Satisfied')
```




```r
# Plot both lines on same plot
full_satisfied <- movie_nine_rank %>%
  mutate(frac_satisfied = cumsum(num_users) / sum(num_users) )

ninety_satisfied <- movie_max_rank %>%
  mutate(frac_satisfied = cumsum(num_users) / sum(num_users) )

ggplot() +
  geom_line(data=full_satisfied, mapping=aes(x=rank, y=frac_satisfied)) +
  geom_line(data=ninety_satisfied, mapping=aes(x=rank, y=frac_satisfied)) +
```

```r
  scale_x_continuous(label = comma, breaks = c(1, 3e3, 6e3, 9e3)) +
  scale_y_continuous(label = percent) +
  xlab('Inventory size (k)') +
  ylab('Fraction of Users Satisfied')
```