STATS 769 - Lab 06 - bole001

Bernard O'Leary

23 September 2019

Dataset description

The dataset provided "Dockless_Vehicle_Trips.csv" is a record of approximately 6.5 million trips taken by people in the Austin Texas area on various different types of "Dockless" vehicles. The dataset has 16 features, including trip Distance, trip Duration and the date the trip was taken. The data are not entirely cleansed and require some tidying to be used for modelling. The data are also significantly large and therefore need to be treated in subsets.

Number of rows in the file "Dockless_Vehicle_Trips.csv"

Get the files from a location on disk and use a "wc" system call to get the number of lines in the file. The command run is:

$ wc -l /course/data.austintexas.gov/Dockless_Vehicle_Trips.csv

There are 6532459 lines in the file, including the header.

```
# File location
# Local machine
#fileLocation <- "C:\\Files\\Dockless_Vehicle_Trips.csv"
fileLocation <- "/mnt/c/Files/Dockless_Vehicle_Trips.csv"
# Data Science VM
#file_location <- "/course/data.austintexas.gov/Dockless_Vehicle_Trips.csv"

# Print number of rows
system(paste("wc -l ", fileLocation))
```

Ingest the data in "Dockless_Vehicle_Trips.csv"

Show the size of each dataframe. The sizes increse by approximately an order of magnitide with each additional order of magnitide that is added to the number of rows. There is comparitively less data structure overhead as each additional order of magnitide is added, which presumably results in the decreasing file size (in proportion to number of rows) with each order of magnitide added to the number of rows in the dataframe.

With 6532459 rows in the file the estimated memory usage if it were all loaded into a dataframe would be approximately 1274067809.1 bytes, which is 1.27 gigabytes.

```
# Garbage collect; reset memory allocation
gc()
```

```
##          used (Mb) gc trigger (Mb) max used (Mb)
## Ncells 382411 20.5     592000 31.7   460000 24.6
## Vcells 583412  4.5    1308461 10.0   816011  6.3
```

```r
# Load 1000 rows
docklessDeliveryTrips1K <- read.csv(file=fileLocation,nrows=1000)
object.size(docklessDeliveryTrips1K)
```

```
## 275592 bytes
```

```r
# Load 10000 rows
docklessDeliveryTrips10K <- read.csv(file=fileLocation,nrows=10000)
object.size(docklessDeliveryTrips10K)
```

```
## 2466488 bytes
```

```r
# Load 100000 rows
docklessDeliveryTrips100K <- read.csv(file=fileLocation,nrows=100000)
object.size(docklessDeliveryTrips100K)
```

```
## 18482872 bytes
```

```r
# Estimate mem usage if we loaded the whole CSF into a dataframe
object.size(docklessDeliveryTrips100K) / 100000 * 6532459
```

```
## 1207386035.42248 bytes
```

Show the type of columns in one of the dataframes

Columns that are automatically set as type "factor" but have many factors (e.g. Device.ID which has 17414 levels) take up significantly more space thn columns that are set to integer however always take up the same amount of space. Factor columns that do not have a large number of factors in them take up the same space as an integer column.

```r
# Loop through column details
for(i in 1:ncol(docklessDeliveryTrips100K)) {
  print(paste("Column name: ", colnames(docklessDeliveryTrips100K)[i]))
  print("Column details: ")
  print(docklessDeliveryTrips100K[0,i])
  print(paste("Column class: ", class(docklessDeliveryTrips100K[,i])))
  print(paste("Column size: ", object.size(docklessDeliveryTrips100K[,i])))
  print("-------------------------------------------------")
}
```

```
## [1] "Column name:  ID"
## [1] "Column details: "
## factor(0)
## 100000 Levels: 00006563-5160-4bff-b467-6700318835ea ...
## [1] "Column class:  factor"
## [1] "Column size:  10000400"
## [1] "-------------------------------------------------"
## [1] "Column name:  Device.ID"
## [1] "Column details: "
```

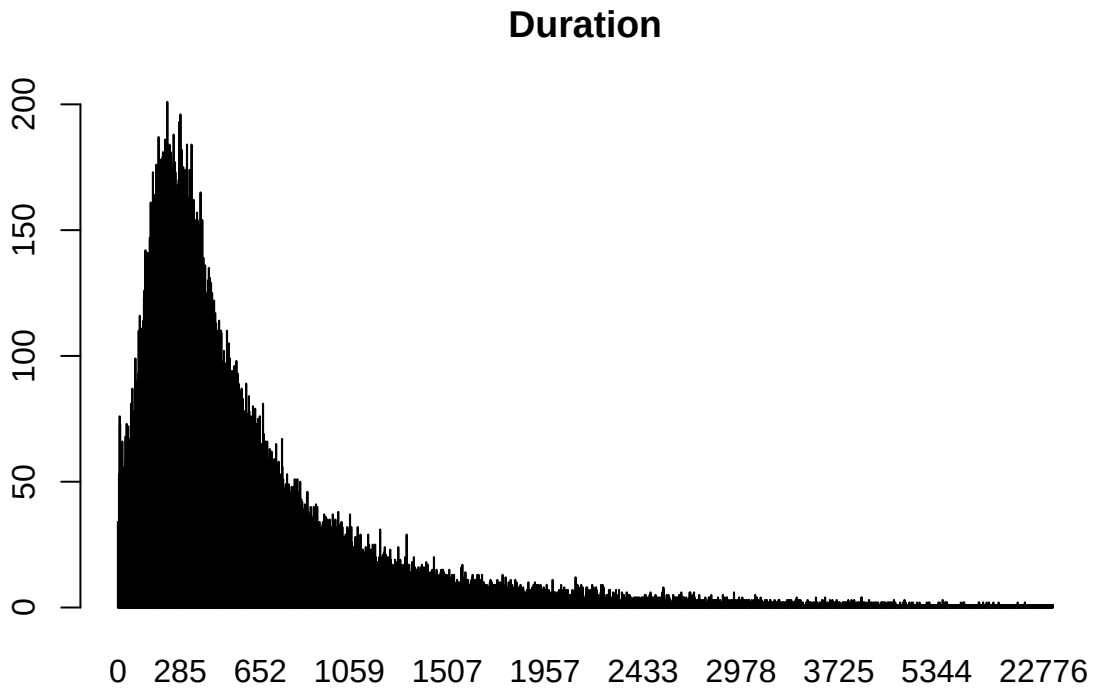
```
## factor(0)
## 17414 Levels: 0004ca6e-ed4d-4077-9628-64e9dd8bb2fd ...
## [1] "Column class:  factor"
## [1] "Column size:  2072144"
## [1] "-------------------------------------------------"
## [1] "Column name:  Vehicle.Type"
## [1] "Column details: "
## factor(0)
## Levels:  bicycle scooter
## [1] "Column class:  factor"
## [1] "Column size:  400576"
## [1] "-------------------------------------------------"
## [1] "Column name:  Trip.Duration"
## [1] "Column details: "
## integer(0)
## [1] "Column class:  integer"
## [1] "Column size:  400040"
## [1] "-------------------------------------------------"
## [1] "Column name:  Trip.Distance"
## [1] "Column details: "
## integer(0)
## [1] "Column class:  integer"
## [1] "Column size:  400040"
## [1] "-------------------------------------------------"
## [1] "Column name:  Start.Time"
## [1] "Column details: "
## factor(0)
## 2865 Levels:  04/02/2019 01:00:00 PM ... 11/30/2018 11:00:00 AM
## [1] "Column class:  factor"
## [1] "Column size:  629576"
## [1] "-------------------------------------------------"
## [1] "Column name:  End.Time"
## [1] "Column details: "
## factor(0)
## 2830 Levels:  04/02/2019 01:00:00 PM ... 07/23/2019 11:15:00 AM
## [1] "Column class:  factor"
## [1] "Column size:  626776"
## [1] "-------------------------------------------------"
## [1] "Column name:  Modified.Date"
## [1] "Column details: "
## factor(0)
## 4188 Levels: 04/16/2019 09:08:24 PM ... 12/21/2018 12:41:42 AM
## [1] "Column class:  factor"
## [1] "Column size:  735440"
## [1] "-------------------------------------------------"
## [1] "Column name:  Month"
## [1] "Column details: "
## integer(0)
## [1] "Column class:  integer"
## [1] "Column size:  400040"
## [1] "-------------------------------------------------"
## [1] "Column name:  Hour"
## [1] "Column details: "
## integer(0)
```

```
## [1] "Column class:  integer"
## [1] "Column size:  400040"
## [1] "-------------------------------------------------"
## [1] "Column name:  Day.of.Week"
## [1] "Column details: "
## integer(0)
## [1] "Column class:  integer"
## [1] "Column size:  400040"
## [1] "-------------------------------------------------"
## [1] "Column name:  Council.District..Start."
## [1] "Column details: "
## integer(0)
## [1] "Column class:  integer"
## [1] "Column size:  400040"
## [1] "-------------------------------------------------"
## [1] "Column name:  Council.District..End."
## [1] "Column details: "
## integer(0)
## [1] "Column class:  integer"
## [1] "Column size:  400040"
## [1] "-------------------------------------------------"
## [1] "Column name:  Year"
## [1] "Column details: "
## integer(0)
## [1] "Column class:  integer"
## [1] "Column size:  400040"
## [1] "-------------------------------------------------"
## [1] "Column name:  Census.Tract.Start"
## [1] "Column details: "
## factor(0)
## 108 Levels:  48055960101 48453000101 48453000102 ... OUT_OF_BOUNDS
## [1] "Column class:  factor"
## [1] "Column size:  407304"
## [1] "-------------------------------------------------"
## [1] "Column name:  Census.Tract.End"
## [1] "Column details: "
## factor(0)
## 128 Levels:  48453000101 48453000102 48453000203 ... OUT_OF_BOUNDS
## [1] "Column class:  factor"
## [1] "Column size:  408584"
## [1] "-------------------------------------------------"
```

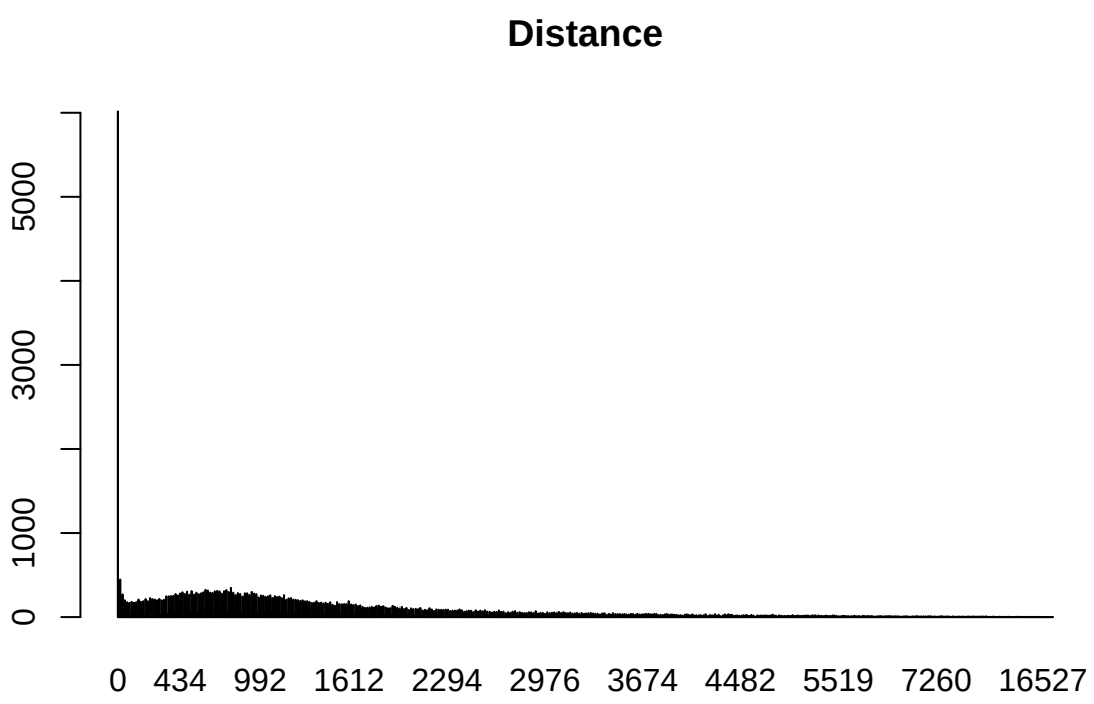
Remove large or null data values

Check for large values in Distance and Duration by plotting data and then removing outliers over a certain value. Trip.Duration has one exceptionally large value of 11491603 and Trip.Distance has a value of 15096088. Remove both of these. Visualise the data afterwards. We clearly still need to remove non-positive rows.

```
docklessDeliveryTrips100K <- docklessDeliveryTrips100K[docklessDeliveryTrips100K$Trip.Duration < 114916
docklessDeliveryTrips100K <- docklessDeliveryTrips100K[docklessDeliveryTrips100K$Trip.Distance < 150960
docklessDeliveryTrips100K <- na.exclude(docklessDeliveryTrips100K)
tripsKeep <- docklessDeliveryTrips100K
barplot(table(tripsKeep$Trip.Duration), main="Duration")
```




```
barplot(table(tripsKeep$Trip.Distance), main="Distance")
```



Explore maximum memory and larges individual objects of the dataframe

Looking at VCells (max used) for the first scenario we have used 93.1mb, for the second we have used 74.7mb. This suggests that using individual variables rather than adding columns to an existing dataframe is a less expensive approach. Adding up the size of the individual objects created by the second piece of code gives 1903432 bytes, compared to the trips dataframe created by the first piece of code at 26633560 - this seems to confirm the memory analysis results.

```
print("Explore memory usage")
```

```
## [1] "Explore memory usage"
```

```
# Reset memory
gc(reset=TRUE)
```

```
##           used (Mb) gc trigger (Mb) max used (Mb)
## Ncells  618361 33.1    1168576 62.5   618361 33.1
## Vcells 2878180 22.0    5187905 39.6  2878180 22.0
```

```
# Run code
trips <- subset(tripsKeep, Trip.Duration > 0 & Trip.Distance > 0)
```

```
trips$logDuration <- log(trips$Trip.Duration)
trips$logDistance <- log(trips$Trip.Distance)

# Explore memory usage
gc()
```

```
##           used (Mb) gc trigger (Mb) max used (Mb)
## Ncells  618634 33.1    1168576 62.5   641414 34.3
## Vcells 3912046 29.9    6305486 48.2  5168872 39.5
```

```
# Reset memory
gc(reset=TRUE)
```

```
##           used (Mb) gc trigger (Mb) max used (Mb)
## Ncells  618646 33.1    1168576 62.5   618646 33.1
## Vcells 3912064 29.9    6305486 48.2  3912064 29.9
```

```
# Run code
subset <- tripsKeep$Trip.Duration > 0 & tripsKeep$Trip.Distance > 0
logDuration <- log(tripsKeep$Trip.Duration[subset])
logDistance <- log(tripsKeep$Trip.Distance[subset])

# Explore memory usage
gc()
```

```
##           used (Mb) gc trigger (Mb) max used (Mb)
## Ncells  618710 33.1    1168576 62.5   649141 34.7
## Vcells 4150043 31.7    6305486 48.2  4781325 36.5
```

```
print("Explore largest objects created")
```

```
## [1] "Explore largest objects created"
```

```
object.size(trips)
```

```
## 24861104 bytes
```

```
object.size(logDuration)
```

```
## 751688 bytes
```

```
object.size(logDistance)
```

```
## 751688 bytes
```

```
object.size(subset)
```

```
## 400032 bytes
```

```
object.size(logDuration) + object.size(logDistance) + object.size(subset)
```

```
## 1903408 bytes
```

Create the model and the MSE function

Run the MSE function across the dataset using the k-fold cross-validation as per the provided MSE function. The system uses 138.2mb to run the cross-validation for 100K rows according the the VCell value. For all 6532459 rows this would be 138.2 / 100000 * 6532459 = 9027.858mb (estimated) - that is just under 1 gigabyte. If the whole class (say 40 people) were to run this, that would be approxaimtely 40 gigabytes of memory required. Because the "sc-cer00014-04.its.auckland.ac.nz" machine has 196 gigabytes of RAM, this would probably be OK.

```
# Reset memory
gc(reset=TRUE)
```

```
##           used (Mb) gc trigger (Mb) max used (Mb)
## Ncells  618576 33.1    1168576 62.5   618576 33.1
## Vcells 4149961 31.7    6305486 48.2  4149961 31.7
```

```
labels <- rep(1:10, length.out=length(logDuration))
groups <- sample(labels)

mse <- function(i, formula) {
  testSet <- groups == i
  trainSet <- groups != i
  fit <- lm(formula,
    data.frame(x=logDistance[trainSet],
    y=logDuration[trainSet]))
  pred <- predict(fit, data.frame(x=logDistance[testSet]))
  mean((pred - logDuration[testSet])^2, na.rm=TRUE)
}

mean(sapply(1:10, mse, y ~ x))
```

```
## [1] 0.3350149
```

```
# Explore memory usage
gc()
```

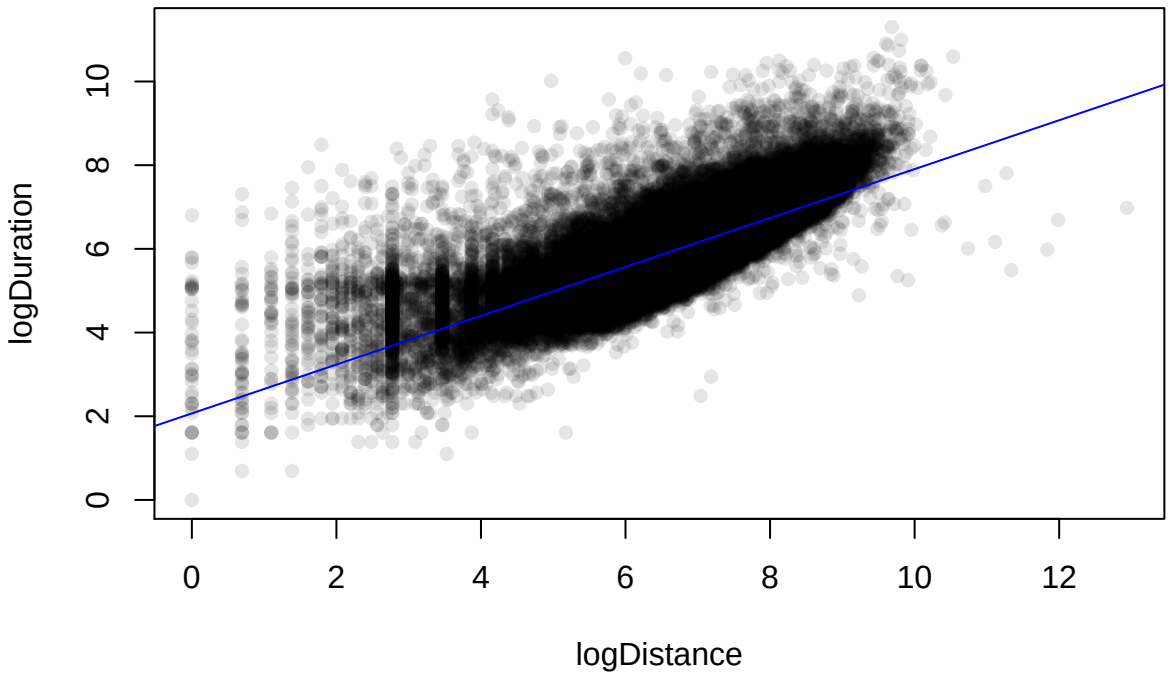
```
##           used (Mb) gc trigger (Mb) max used (Mb)
## Ncells  626284 33.5    1168576 62.5   656280 35.1
## Vcells 4257981 32.5    7646583 58.4  7645854 58.4
```

```
# Calculate mem usage for full dataset
138.2 / 100000 * 6532459
```

```
## [1] 9027.858
```

Plot the data and model

```
lmFit <- lm(y ~ x, data.frame(x=logDistance, y=logDuration))
plot(logDistance, logDuration, pch=16, col=rgb(0,0,0,.1))
abline(lmFit, col="blue")
```



Measure memory usage to run the RMD file

The command used to measure the memory usage of the RMD file was as follows:

/usr/bin/time -f "%M" Rscript -e "rmarkdown::render('STATS769_2019_S2_bole001_lab06.Rmd',output_file='STATS769_

This provided an output of 142048 KB (142 MB). This is not quite what I expected as the dataframe being used to hold the 100K rows that we have been using for the whole experiment is 195 MB in size.

Summary and Conclusion

R and the Bash Shell provide a range of ways to measure performance and memory use, in this labe some of those have been explored for the analysis of a large dataset. The machine that is used to run these experiments has significant memory capacity, however a normal computer would certainly struggle if the entire dataset were loaded, rather that only 100K. More investigation of the Bash "time"" command will be necessary for me to understand it properly.