STATS 769 - Lab 02 - bole001

Bernard O'Leary

12 August 2019

Read in data and Linux commands

Data format is CSV (Comma Separated Values), reading into R using the read.csv function. Becuase there are a large number of files an alternative approach has been taken, to list all files in the Lab02 directory accroding to the file structure we are looking for ("trips*.csv") and then pick up each file and add to a dataframe (called"trips.df").

The machine that was used to create the report is a Microsoft Windows 10 machine that has an Ubuntu Linux Bash shell built into it as part of the WSL (Windows Subsystem for Linux) feature that is an optional feature of the Windows 10 OS (https://docs.microsoft.com/en-us/windows/wsl/install-win10). Once access was gained to the STATS 769 Linux machines, the files were copied to the Windows 10 machine used to create the report using the following scp command:

```
bernardo@PKS10198:/mnt/d/Study/UoA-STATS-769$ scp bole001@sc-cer00014-
04.its.auckland.ac.nz:/course/Labs/Lab02/*.csv .
```

From there RStudio was used on the local machine to develop the report. So that the RMD for the report will run anywhere on the STATS 769 Linux machines, a "directory" variable is used that holds the value of the directory that the trips files are in. The script uses this variable to locate the directory containing the data files. For the STATS 769 machine this variable is set as follows:

```
directory = '/course/Labs/Lab02/'
```

Whereas on the Windows 10 machine used to create the report, it was set to the local filesystem where the csv files for the lab were copied to. Code used to create the initial dataframe for the Trips data follows.

```
# When the RMD for the lab is submitted the following value will be
uncommented
#directory = '/course/Labs/Lab02/'
# When the RMD for the lab is submitted the following value will be commented
directory = 'D:/Study/UoA-STATS-769/Lab02/'
files.ls <- intersect(list.files(path=directory, pattern="trips"),
list.files(path=directory, pattern="*.csv"))
# First apply read.csv, then rbind
trips.df = do.call(rbind, lapply(files.ls, function(x) read.csv(x,
stringsAsFactors = FALSE)))
```

Add 'LongTrip' variable

Create variable named 'LongTrip' and add to the end of the dataframe.

```
trips.df$LongTrip <- trips.df$Trip.Distance > 1000
```

Add logged duration variable

Name the variable Trip.Duration.Logged and add to the end of the dataframe. Note that because the data are not cleansed by this point NaNs are produced here.

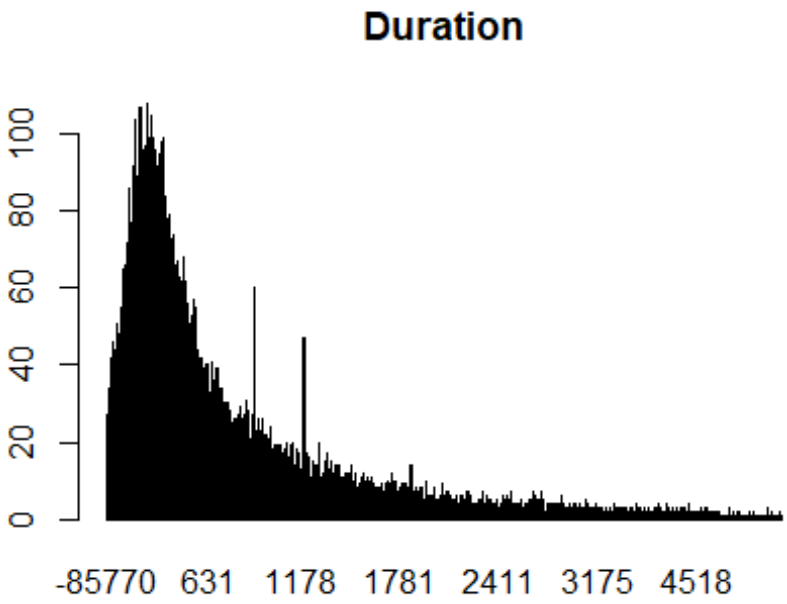
```
trips.df$Trip.Duration.Logged <- log(trips.df$Trip.Duration)

## Warning in log(trips.df$Trip.Duration): NaNs produced
```

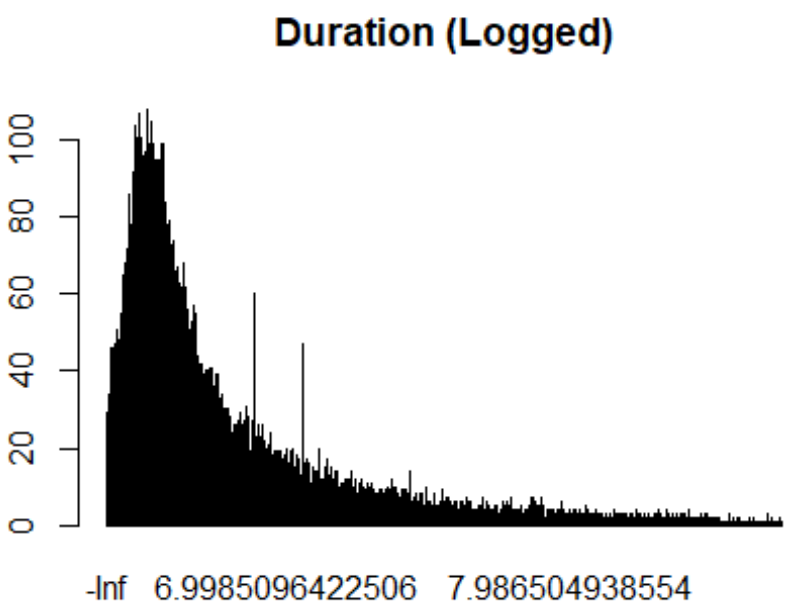
Explore Trip.Duration, Trip.Duration, Trip.Duration.Logged and LongTrip variables

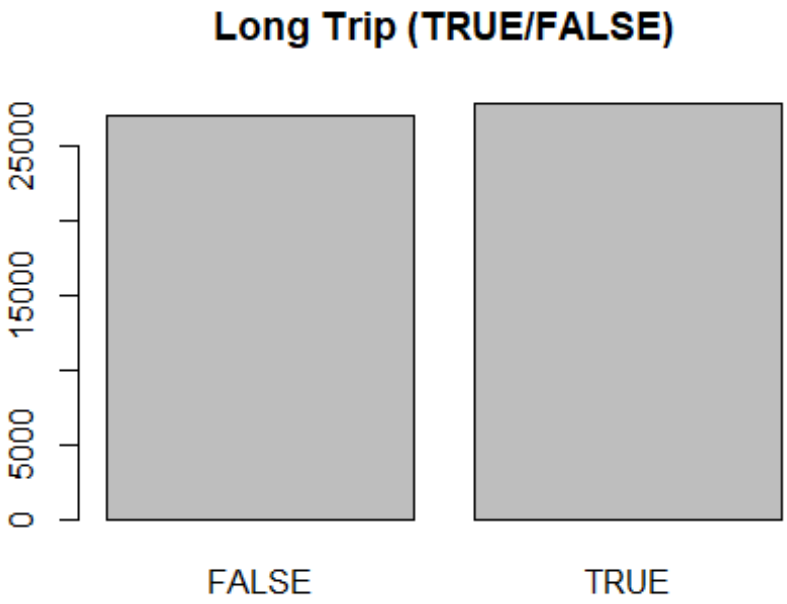
Visualise the Trip.Duration.Logged (what will be used as the predictor) and LongTrip (the response) features by generating a barplot of their respective values. The shape of the duration data are not changed much by logging it.

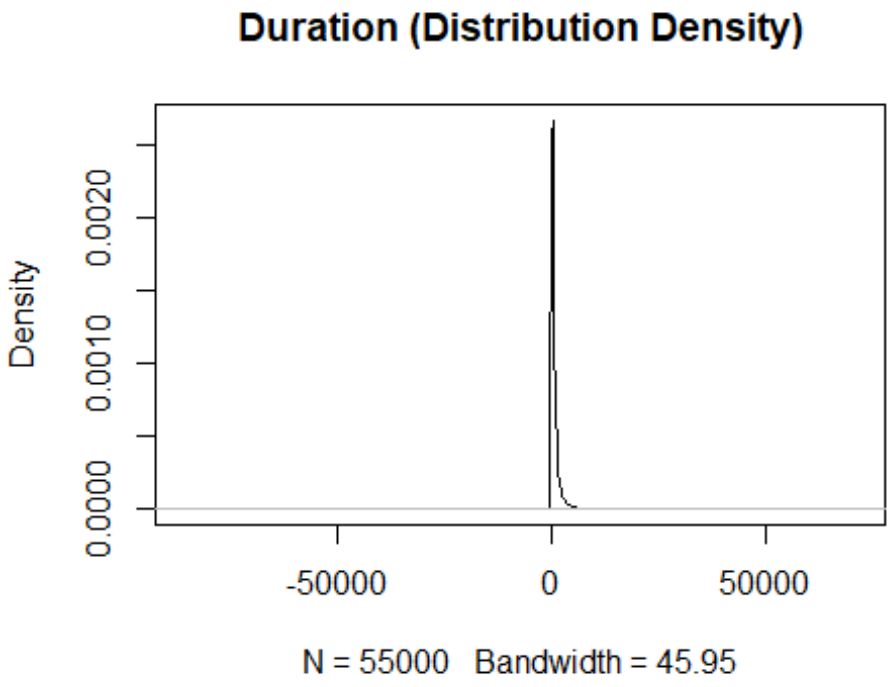
There are about the same volume of LongTrips as there are not LongTrips - the distance of 1000 is a good place to split the data in this regard.

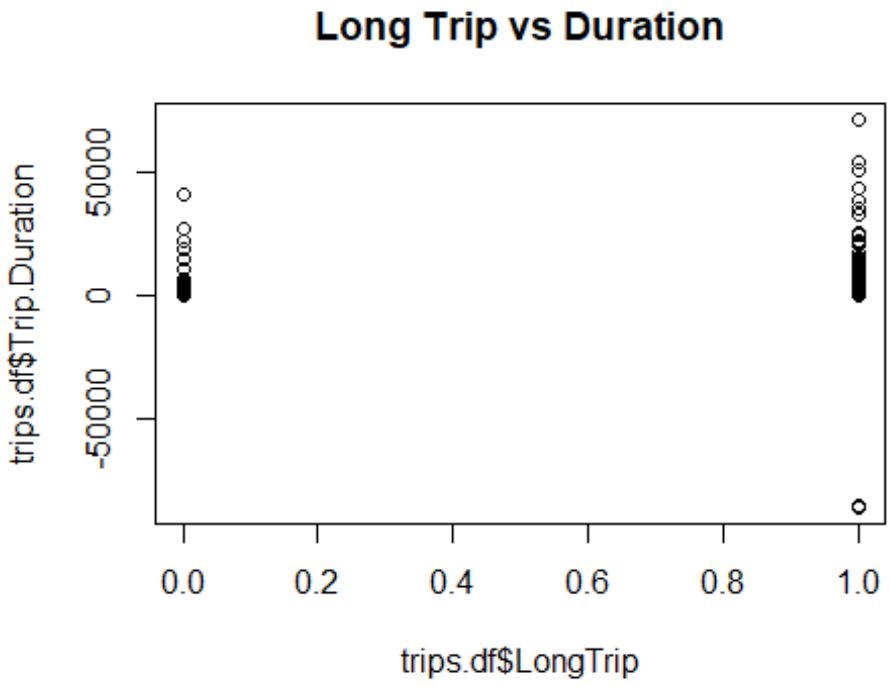
THe Long Trip vs Duration plots reveal that as would be expected, trips with longer distance also have a longer duration in general.

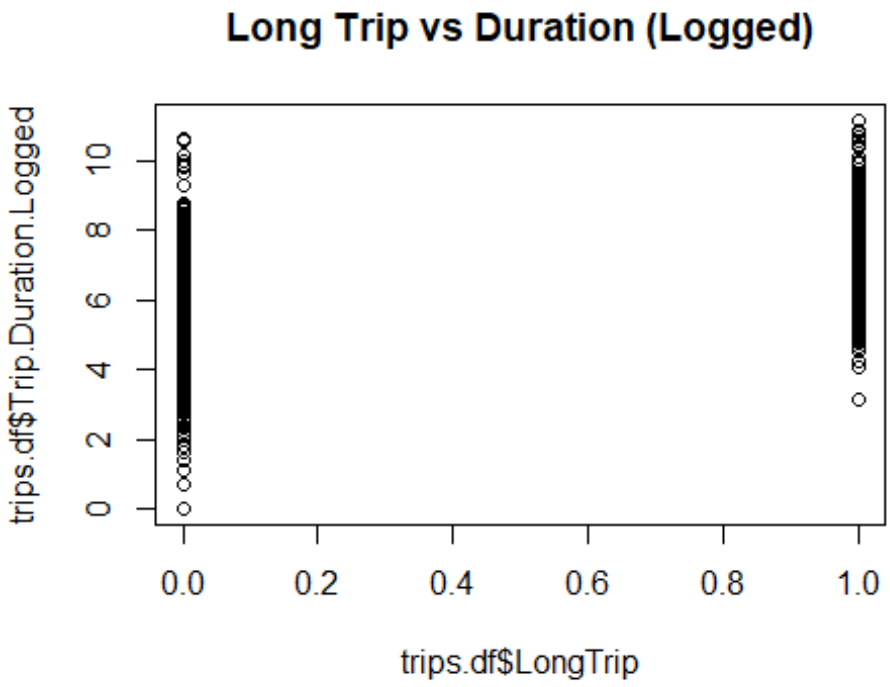

Duration
0
20
40
60
80
100
-85770
631
1178
1781
2411
3175
4518


Duration (Logged)
0
20
40
60
80
100
-Inf
6.9985096422506
7.986504938554


Long Trip (TRUE/FALSE)
0
5000
15000
25000
FALSE
TRUE


Duration (Distribution Density)
Density
0.0000
0.0010
0.0020
-50000
0
50000
N = 55000 Bandwidth = 45.95


Long Trip vs Duration
trips.df$Trip.Duration
50000
0
-50000
0.0
0.2
0.4
0.6
0.8
1.0
trips.df$LongTrip


Long Trip vs Duration (Logged)
trips.df$Trip.Duration.Logged
0
2
4
6
8
10
0.0
0.2
0.4
0.6
0.8
1.0
trips.df$LongTrip

Train/test split

Use the plyr package to split the records evenly by month (1000 of each) and take samples. Use month as divider and take a random sample of 0.8 of the dataset split evenly across the months for the training set. Capture the remainder (0.2) of the split in the second part of the output for the test set.

The plyr package can be found here:
https://www.rdocumentation.org/packages/plyr/versions/1.8.4/topics/dlply

The sample() function enables a random selection of data from an R dataframe.

```
# use the plyr library to split the records evenly and take samples
# use month as divider and take a random sample of 0.8 of the dataset split
evenly across the months
# capture the remainder (0.2) of the split in the second part of the output
# https://www.rdocumentation.org/packages/plyr/versions/1.8.4/topics/dlply
# https://stackoverflow.com/questions/18258690/take-randomly-sample-based-on-
groups
# https://stackoverflow.com/questions/18942792/training-and-test-set-with-
respect-to-group-affiliation
library(plyr)

## Warning: package 'plyr' was built under R version 3.6.1

split_set = dlply(trips.df, .(Month), function(.) { s = sample(1:nrow(.),
trunc(nrow(.) * 0.8)); list(.[s, ], .[-s,]) } )

# train/test split
# https://www.rdocumentation.org/packages/plyr/versions/1.8.4/topics/ldply
training_set = ldply(split_set, function(.) .[[1]])
test_set = ldply(split_set, function(.) .[[2]])
```

Get rid of non-positive and non-finite values in duration and distance

Remove from the training set only

```
training_set <- subset(training_set, training_set$Trip.Duration > 0)
training_set <- subset(training_set, training_set$Trip.Distance > 0)
training_set <- subset(training_set, is.finite(training_set$Trip.Duration))
training_set <- subset(training_set, is.finite(training_set$Trip.Distance))
```

Make the model

Use logged duration (rather than raw duration) to avoid warning “glm.fit: fitted probabilities numerically 0 or 1 occurred”.

```
y_train = training_set$LongTrip
#x_train = training_set$Trip.Duration
x_train = training_set$Trip.Duration.Logged
fit_mean = mean(y_train)
```

```
fit_glm <- glm(formula=y~x, data.frame(y=y_train, x=x_train),
family=binomial(link=logit))
```

Get rid of non-positive and non-finite values in duration and distance

Cleanse the test_set dataframe now.

```
test_set <- subset(test_set, test_set$Trip.Duration > 0)
test_set <- subset(test_set, test_set$Trip.Distance > 0)
test_set <- subset(test_set, is.finite(test_set$Trip.Duration))
test_set <- subset(test_set, is.finite(test_set$Trip.Distance))
```

Test results

Generate two vectors; the predicted sample mean and the prediction generated by the GLM.

```
y_test <- test_set$LongTrip
x_test <- test_set$Trip.Duration.Logged

# make a vector that is populated with the fit_mean
pred_mean <- rep(fit_mean, length(y_test))

# get a vector of predictions based on test data
pred_glm <- predict(fit_glm, data.frame(x=x_test))
```

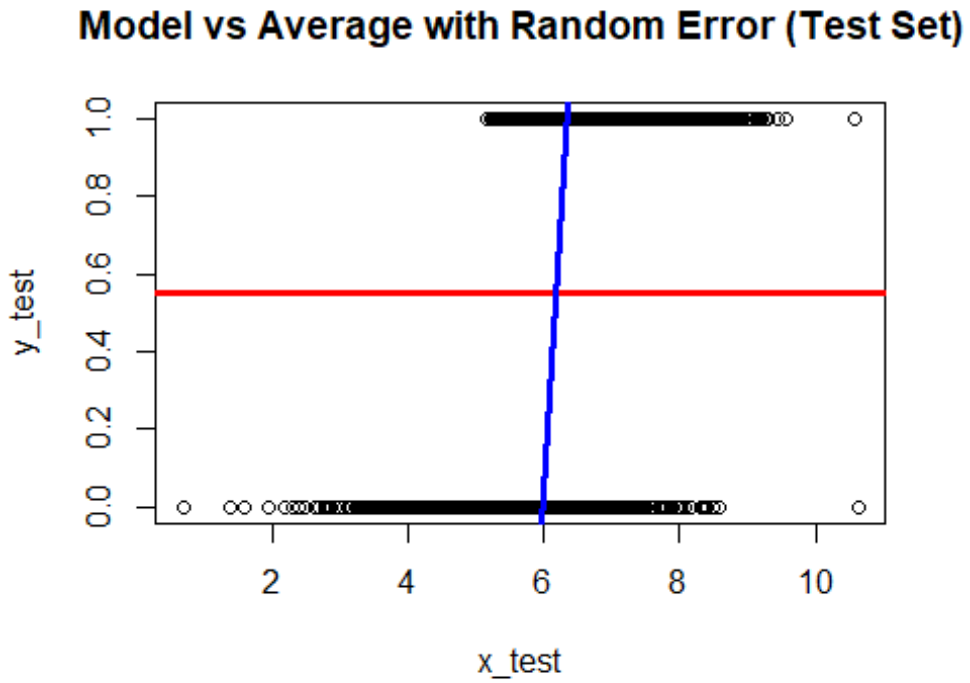
Define RMSE function and evaluate using test set

Take the RMSE (root mean square error) of the predicted mean and the predicted logistic regression model. The model gives us a far higher RMSE than the predicted mean (the predicted mean is an average model with random noise).

```
# define RMSE function
RMSE <- function(m, o) {
    sqrt(mean((m - o)^2))
}

# get the RMSE
RMSE(pred_mean, y_test)

## [1] 0.4970554

RMSE(pred_glm, y_test)

## [1] 2.390155
```

Visualise the model

Plot y_test vs x_test and overlay the average with random error model and our derived logistic regression model. This is to help us visualise the data and the derived model as it applies to the test dataset.



Summary and analysis

Our model has not reduced the variability that we see by simply taking the average model with random error - in-fact it slightly increases the variablity. This is not a good model as it explains less of the variability in the data than the average model with random error. The model could be improved to better fit the data.