STAT 206 Lab 2

Due Monday, October 16, 5:00 PM

General instructions for labs: You are encouraged to work in pairs to complete the lab. Labs must be completed as an R Markdown file. Be sure to include your lab partner (if you have one) and your own name in the file. Give the commands to answer each question in its own code block, which will also produce plots that will be automatically embedded in the output file. Each answer must be supported by written statements as well as any code used.

Agenda: Manipulating data frames; practicing iteration; practicing re-writing code; checking how reliable random methods are.

Part I – Data Frames

R includes a number of pre-specified data objects as part of its default installation. We will load and manipulate one of these, a data frame of 93 cars with model year 1993. Begin by ensuring that you can load this data with the commands

```
library(MASS)
data(Cars93)
```

Begin by examining the data frame with the command `View(Cars93)` to understand the underlying object. You will need to use functions and other commands to extract elements for this assignment.

```
# Class displays what type a given argument is

class(Cars93)
```

```
## [1] "data.frame"
```

```
#View puts the data frame into its own table with header

View(Cars93)
```

1. Obtain a `summary()` of the full data structure. Can you tell from this how many rows are in the data? If so, say how; if not, use another method to obtain the number of rows.

```
#the summary function returns some descriptve statistics on from the data frame. It includes the "5 num
#the 5 number summary includes the min value, q1, median, q3 and max value
#From this data we can get an idea of how many rows and columns. We can count the number of attrributes
#summary returns and we can know the amount of columns, and for the rows we can find an attribute that
#an idex to find the number of rows. in this case we see both Model and Make display 6 values and shows
#My guess would be that there is 93 rows
#It is probably much better to use a couple funcions to accomplish this and display the number of colum

summary(Cars93)
```

```
##     Manufacturer     Model         Type      Min.Price         Price
##  Chevrolet: 8    100    : 1   Compact:16   Min.   : 6.70   Min.   : 7.40
##  Ford     : 8    190E   : 1   Large  :11   1st Qu.:10.80   1st Qu.:12.20
##  Dodge    : 6    240    : 1   Midsize:22   Median :14.70   Median :17.70
##  Mazda    : 5    300E   : 1   Small  :21   Mean   :17.13   Mean   :19.51
##  Pontiac  : 5    323    : 1   Sporty :14   3rd Qu.:20.30   3rd Qu.:23.30
##  Buick    : 4    535i   : 1   Van    : 9   Max.   :45.40   Max.   :61.90
```

```
##  (Other)  :57    (Other):87
##    Max.Price       MPG.city      MPG.highway                  AirBags
##  Min.   : 7.9   Min.   :15.00   Min.   :20.00   Driver & Passenger:16
##  1st Qu.:14.7   1st Qu.:18.00   1st Qu.:26.00   Driver only       :43
##  Median :19.6   Median :21.00   Median :28.00   None              :34
##  Mean   :21.9   Mean   :22.37   Mean   :29.09
##  3rd Qu.:25.3   3rd Qu.:25.00   3rd Qu.:31.00
##  Max.   :80.0   Max.   :46.00   Max.   :50.00
##
##  DriveTrain  Cylinders    EngineSize      Horsepower          RPM
##  4WD  :10   3     : 3   Min.   :1.000   Min.   : 55.0   Min.   :3800
##  Front:67   4     :49   1st Qu.:1.800   1st Qu.:103.0   1st Qu.:4800
##  Rear :16   5     : 2   Median :2.400   Median :140.0   Median :5200
##             6     :31   Mean   :2.668   Mean   :143.8   Mean   :5281
##             8     : 7   3rd Qu.:3.300   3rd Qu.:170.0   3rd Qu.:5750
##             rotary: 1   Max.   :5.700   Max.   :300.0   Max.   :6500
##
##   Rev.per.mile  Man.trans.avail Fuel.tank.capacity   Passengers
##  Min.   :1320   No :32          Min.   : 9.20      Min.   :2.000
##  1st Qu.:1985   Yes:61          1st Qu.:14.50      1st Qu.:4.000
##  Median :2340                   Median :16.40      Median :5.000
##  Mean   :2332                   Mean   :16.66      Mean   :5.086
##  3rd Qu.:2565                   3rd Qu.:18.80      3rd Qu.:6.000
##  Max.   :3755                   Max.   :27.00      Max.   :8.000
##
##      Length        Wheelbase         Width        Turn.circle
##  Min.   :141.0   Min.   : 90.0   Min.   :60.00   Min.   :32.00
##  1st Qu.:174.0   1st Qu.: 98.0   1st Qu.:67.00   1st Qu.:37.00
##  Median :183.0   Median :103.0   Median :69.00   Median :39.00
##  Mean   :183.2   Mean   :103.9   Mean   :69.38   Mean   :38.96
##  3rd Qu.:192.0   3rd Qu.:110.0   3rd Qu.:72.00   3rd Qu.:41.00
##  Max.   :219.0   Max.   :119.0   Max.   :78.00   Max.   :45.00
##
##  Rear.seat.room   Luggage.room       Weight         Origin
##  Min.   :19.00   Min.   : 6.00   Min.   :1695   USA    :48
##  1st Qu.:26.00   1st Qu.:12.00   1st Qu.:2620   non-USA:45
##  Median :27.50   Median :14.00   Median :3040
##  Mean   :27.83   Mean   :13.89   Mean   :3073
##  3rd Qu.:30.00   3rd Qu.:15.00   3rd Qu.:3525
##  Max.   :36.00   Max.   :22.00   Max.   :4105
##  NA's   :2       NA's   :11
##             Make
##  Acura Integra: 1
##  Acura Legend : 1
##  Audi 100     : 1
##  Audi 90      : 1
##  BMW 535i     : 1
##  Buick Century: 1
##  (Other)      :87
```

```
nrow(Cars93)
```

```
## [1] 93
```

```
ncol(Cars93)
```

```
## [1] 27
```

```
# nrow and ncol will display the number of rows and columns in  a dataframe, respectively. This is a mu
```

2. What is the mean price of a car with a rear-wheel drive train?

```
#First step is to get a slice of this data frame to extract only rows that have 'Rear' for attribute Dr
#We also want to get the price values assocaited with the above values
#we can use the subset function for anintuitive way to obtain this slice

rwd_avg_p = subset(Cars93, DriveTrain == 'Rear', select = c('Price', 'DriveTrain'))

#Finally, we use the mean function but only on the price values in our subset

mean(rwd_avg_p[,'Price'])
```

```
## [1] 28.95
```

3. What is the minimum horsepower of all cars with capacity for 7 passengers? With a capacity of at least 6 passengers?

```
#Again the subset function will be used to grab a slice of the data frame that consists of horse power
#first will be a slice for when passengers = 7

cap_hp = subset(Cars93, Passengers == 7, select = c('Passengers', 'Horsepower'))

#the min function is called on the attribute Horsepower for the subset data

min(cap_hp[,'Horsepower'])
```

```
## [1] 109
```

```
#Next we will get a slice for when there are atleast 6 passengers

cap_hp2 = subset(Cars93, Passengers >= 6, select = c('Passengers', 'Horsepower'))

#now we will use the min function again to get the new minimum of this slice

min(cap_hp2[,'Horsepower'])
```

```
## [1] 100
```

4. Assuming that these cars are exactly as fuel efficient as this table indicates, find the cars that have the maximum, minimum and median distance travellable for highway driving. You will need at least two columns to work this out; why those two?

```
# To obtain which cars have the best, worst and middle hwy mpg we need to first know what these values
#we can do this using the max, min, and median functions on the Cars93 data, on the hwy mpg column
# note this info is available in the summary above

maxmpg = max(Cars93[,'MPG.highway'])
maxmpg
```

```
## [1] 50
```

```
medianmpg = median(Cars93[,'MPG.highway'])
medianmpg
```

```
## [1] 28
```

```
minmpg = min(Cars93[,'MPG.highway'])
minmpg
```

```
## [1] 20
```

```
# Now we can create slices of the data frame for Cars that have the above values

maxhighway = subset(Cars93, MPG.highway == maxmpg, select = c('MPG.highway', 'Make'))
maxhighway
```

```
##    MPG.highway      Make
## 39          50 Geo Metro
```

```
medianhighway = subset(Cars93, MPG.highway == medianmpg, select = c('Make', 'MPG.highway'))
medianhighway
```

```
##                       Make MPG.highway
## 7             Buick LeSabre          28
## 14         Chevrolet Camaro          28
## 20        Chrylser Concorde          28
## 21         Chrysler LeBaron          28
## 30             Eagle Vision          28
## 71 Oldsmobile Eighty-Eight          28
## 75         Pontiac Firebird          28
## 77       Pontiac Bonneville          28
## 92                Volvo 240          28
## 93                Volvo 850          28
```

```
minhighway = subset(Cars93, MPG.highway == minmpg, select = c('Make', 'MPG.highway'))
minhighway
```

```
##                Make MPG.highway
## 17 Chevrolet Astro          20
## 36   Ford Aerostar          20
```

Part II – Reproducibility and Functions

Some of the lectures have included examples of planning production for a factory that turns steel and labor into cars and trucks. Below is a piece of code that optimizes the factory's output (roughly) given the available resources, using a `repeat` loop. It's embedded in a function to make it easier for you to run.

```
factory.function <- function (cars.output=1, trucks.output=1) {
  factory <- matrix(c(40,1,60,3),nrow=2,
    dimnames=list(c("labor","steel"),c("cars","trucks")))
  available <- c(1600,70); names(available) <- rownames(factory)
  slack <- c(8,1); names(slack) <- rownames(factory)
  output <- c(cars.output, trucks.output); names(output) <- colnames(factory)

  passes <- 0 # How many times have we  been around the loop?
  repeat {
     passes <- passes + 1
     needed <- factory %*% output # What do we need for that output level?
     # If we're not using too much, and are within the slack, we're done
     if (all(needed <= available) &&
```

```
        all((available - needed) <= slack)) {
      break()
    }
    # If we're using too much of everything, cut back by 10%
    if (all(needed > available)) {
      output <- output * 0.9
      next()
    }
    # If we're using too little of everything, increase by 10%
    if (all(needed < available)) {
      output <- output * 1.1
      next()
    }
    # If we're using too much of some resources but not others, randomly
    # tweak the plan by up to 10%
     # runif == Random number, UNIFormly distributed, not "run if"
    output <- output * (1+runif(length(output),min=-0.1,max=0.1))
  }

  return(output)
}
```

5. Run the function above with the command

```
factory.function()
```

```
##     cars   trucks
## 10.28648 19.69488
```

to obtain a default output value, starting from a very low initial planned output. What is the final output capacity obtained?

6. Repeat this four more times to obtain new output values. Do these answers differ from each other? If so why? If not, why not?

```
factory.function()
```

```
##      cars    trucks
##  9.949649 19.959082
```

```
factory.function()
```

```
##     cars   trucks
## 10.08165 19.91860
```

```
factory.function()
```

```
##     cars   trucks
## 10.47071 19.64083
```

```
factory.function()
```

```
##     cars   trucks
## 10.25063 19.75981
```

```
#All of the answers are relatively close to one another however, all of them are different. This is due
#function itself. It even says that it will "randomly tweak the plan by up to %10" This means that each
#the function will return a different output
```

7. Right now, the number of `passes` is a value held within the function itself and not shared. Change the code so that the number of `passes` will be returned at the end of the function, as well as the final `output`.

```
#The return(output) at the end was commented out so I could use a print statement to display the output

factory.function <- function (cars.output=1, trucks.output=1) {
  factory <- matrix(c(40,1,60,3),nrow=2,
    dimnames=list(c("labor","steel"),c("cars","trucks")))
  available <- c(1600,70); names(available) <- rownames(factory)
  slack <- c(8,1); names(slack) <- rownames(factory)
  output <- c(cars.output, trucks.output); names(output) <- colnames(factory)

  passes <- 0 # How many times have we  been around the loop?
  repeat {
     passes <- passes + 1
     needed <- factory %*% output # What do we need for that output level?
     # If we're not using too much, and are within the slack, we're done
     if (all(needed <= available) &&
         all((available - needed) <= slack)) {
       break()
     }
     # If we're using too much of everything, cut back by 10%
     if (all(needed > available)) {
       output <- output * 0.9
       next()
     }
     # If we're using too little of everything, increase by 10%
     if (all(needed < available)) {
       output <- output * 1.1
       next()
     }
     # If we're using too much of some resources but not others, randomly
     # tweak the plan by up to 10%
      # runif == Random number, UNIFormly distributed, not "run if"
     output <- output * (1+runif(length(output),min=-0.1,max=0.1))


  }

  #return(output)
  print(output) + print(passes)
}

factory.function()
```

```
##     cars   trucks
## 10.54919 19.61909
## [1] 2239
```

8. Now, set the initial output levels to 30 cars and 20 trucks and run the code. What is the final output plan (`output`)? What is the final demand for resources (`needed`)? Is the plan within budget and within the slack? How many iterations did it take to converge (`passes`)? For all but `output` you will need to either print this message out deliberately, or return an object that contains all the quantities you want.

```r
# the code was copied from aove but in the print statement we are printing out 'needed' to see how much
#material was consumed for the output. In this case the output was almost 10 cars and 20 trucks

factory.function <- function (cars.output=1, trucks.output=1) {
  factory <- matrix(c(40,1,60,3),nrow=2,
    dimnames=list(c("labor","steel"),c("cars","trucks")))
  available <- c(1600,70); names(available) <- rownames(factory)
  slack <- c(8,1); names(slack) <- rownames(factory)
  output <- c(cars.output, trucks.output); names(output) <- colnames(factory)

  passes <- 0 # How many times have we  been around the loop?
  repeat {
     passes <- passes + 1
     needed <- factory %*% output # What do we need for that output level?
     # If we're not using too much, and are within the slack, we're done
     if (all(needed <= available) &&
         all((available - needed) <= slack)) {
       break()
     }
     # If we're using too much of everything, cut back by 10%
     if (all(needed > available)) {
       output <- output * 0.9
       next()
     }
     # If we're using too little of everything, increase by 10%
     if (all(needed < available)) {
       output <- output * 1.1
       next()
     }
     # If we're using too much of some resources but not others, randomly
     # tweak the plan by up to 10%
      # runif == Random number, UNIFormly distributed, not "run if"
     output <- output * (1+runif(length(output),min=-0.1,max=0.1))


  }

  #return(output)
  print(output) + print(passes) + print(needed)
}

factory.function(cars.output = 1, trucks.output = 20)
```

```
##     cars   trucks
## 10.26458 19.79964
## [1] 1146
##             [,1]
## labor 1598.56186
## steel   69.66351
```