Final

Fan Yang (fy2232)

Section 03#

Instructions (Read this completely first)

You should complete the exam by editting this file directly. Please knit the file often, so that if you make a mistake you catch it before the end of the exam. You will have exactly 160 minutes from your start time to complete the exam. **At the end you must turn in your knitted .pdf file and raw .Rmd file on Courseworks.**

When the time is up, you must shut your computer immediately. We will take off points from anyone whose computer is still open after time is up.

You may use your class notes for the exam, but not the internet. You absolutely may not communicate with anyone else during the exam. Doing so will result in an F in this class and likely result in termination from the MA program.

Question 0 (5 points)

a. (0.5 points) Place your section number as the date of the document. If you don't know your section number, you can determine it below based on when your lab meets.

- Section 002 Lab meets TR 7:40pm-8:55pm
- Section 003 Lab meets TR 11:40am-12:55pm
- Section 004 Lab meets MW 8:40am-9:55am
- Section 005 Lab meets TR 8:40am-9:55am

b. (0.5 points) Write your name and UNI as the author of the document.

c. (4 points) Please present your answers in a readable format. This includes things like indenting your code and generally presenting easy-to-read code. Presentation of the overall Markdown document will be considered as well.

Question 1: Fitting Data (49 points)

(a) (4 points) You have an urn with 30 balls – 10 are red, 10 are blue, and 10 are green. Write a single line of code to simulate randomly picking 400 balls from the urn with replacement. Create a variable `num_green` that records the number of green balls selected in the 400 draws.

```
set.seed(1)

# Your answer to question 1.a here.  Don't remove the set.seed(1) command.
num_total <- sample(c(rep('r',10),rep('b',10),rep('g',10)), 400,
                    replace = TRUE)
num_green <- sum(num_total == 'g')
num_green
```

```
## [1] 123
```

(b) (4 points) Now repeat the above experiment 1000 times. Create a vector `data`, such that each element in `data` is the result (counting the number of green balls) from an independent trial like that described in 1.a.

```
set.seed(2)

# Your answer to question 1.b here.  Don't remove the set.seed(2) command.
data <- c()
for (i in 1:1000){
  num_total_b <- sample(c(rep('r',10),rep('b',10),rep('g',10)), 400,
                    replace = TRUE)
  num_green_b <- sum(num_total_b == 'g')
  data <- c(data, num_green_b)
}
data
```

```
##    [1] 136 144 135 130 127 135 144 127 110 143 146 135 150 136 136 147 127
##   [18] 127 138 122 144 138 143 135 121 139 133 115 128 133 123 139 136 156
##   [35] 124 140 141 126 141 145 119 130 135 146 121 148 131 146 144 126 147
##   [52] 121 118 143 123 124 150 130 133 135 130 139 153 137 117 126 135 127
##   [69] 123 114 114 139 145 134 120 133 139 147 137 130 140 132 142 131 130
##   [86] 143 126 139 144 141 141 147 142 123 121 140 134 139 134 126 124 167
##  [103] 120 130 122 142 130 153 136 141 145 142 102 141 137 140 124 138 134
##  [120] 137 127 144 120 143 129 137 153 137 133 121 146 116 157 130 130 116
##  [137] 130 134 141 137 138 128 132 126 130 123 121 132 132 136 141 129 127
##  [154] 134 126 120 133 132 140 128 136 130 141 132 150 146 130 135 135 142
##  [171] 123 139 120 135 137 118 136 143 138 150 139 136 134 129 120 140 125
##  [188] 120 154 130 130 130 132 131 147 116 138 138 136 139 153 126 142 132
##  [205] 126 149 142 144 143 141 151 129 125 147 142 134 119 139 138 140 130
##  [222] 133 115 127 136 144 149 139 137 140 142 137 137 140 137 132 126 152
##  [239] 137 144 121 129 129 134 129 142 128 125 139 125 139 145 135 124 118
##  [256] 144 138 117 133 138 118 129 154 146 118 148 146 135 143 146 112 149
##  [273] 131 133 123 129 140 119 128 135 129 129 123 134 134 120 140 142 149
##  [290] 142 145 120 133 134 129 125 125 131 146 126 135 125 130 143 136 145
##  [307] 139 127 138 146 124 137 135 136 152 123 136 142 143 140 133 136 136
##  [324] 127 139 141 140 132 127 142 139 124 150 126 126 128 133 136 142 117
##  [341] 135 110 147 139 148 124 119 136 113 134 139 150 128 138 124 130 125
##  [358] 126 123 135 140 121 149 123 114 144 127 103 143 144 118 133 132 150
##  [375] 133 138 142 138 127 156 146 130 143 142 135 126 139 130 132 139 136
##  [392] 129 138 141 123 129 107 132 129 151 143 138 126 127 137 140 135 129
```

```
##  [409] 133 114 126 135 148 140 162 129 134 132 147 115 130 123 133 127 136
##  [426] 140 152 141 139 125 133 141 134 148 133 137 127 132 137 140 146 147
##  [443] 131 133 136 135 141 144 123 122 141 130 139 138 144 122 142 147 128
##  [460] 142 137 133 137 117 136 131 132 134 115 131 129 158 140 162 129 119
##  [477] 123 128 145 125 136 131 143 135 138 126 129 132 129 123 109 139 151
##  [494] 124 113 136 143 136 132 150 129 128 137 112 141 117 133 132 129 120
##  [511] 144 126 131 125 143 148 149 139 139 130 134 133 133 128 123 151 138
##  [528] 134 128 127 123 134 130 132 147 131 142 149 145 135 121 139 144 147
##  [545] 141 126 138 120 139 124 117 135 122 117 143 152 140 124 112 139 135
##  [562] 119 125 128 129 129 135 135 122 118 125 150 132 125 130 142 133 134
##  [579] 118 143 137 150 134 143 138 136 131 133 128 126 132 117 139 132 116
##  [596] 131 124 143 139 136 149 127 135 122 136 157 137 129 133 133 135 120
##  [613] 133 142 151 130 123 140 138 126 128 117 122 119 147 123 131 136 142
##  [630] 123 154 122 138 140 133 146 161 147 138 139 129 140 142 137 142 138
##  [647] 139 141 131 134 139 132 141 139 132 135 127 132 119 133 132 134 131
##  [664] 134 139 143 121 141 120 133 131 134 146 139 134 112 145 136 140 126
##  [681] 139 144 155 117 128 131 127 116 145 144 153 134 128 125 136 147 138
##  [698] 136 135 126 137 131 127 134 127 129 121 144 129 130 136 134 130 137
##  [715] 144 139 134 120 127 123 139 135 133 130 134 126 132 135 136 129 133
##  [732] 124 154 133 135 118 122 150 122 142 127 142 136 146 122 133 149 144
##  [749] 137 124 136 137 145 127 132 130 120 129 124 132 135 139 145 141 134
##  [766] 142 138 141 126 118 124 143 131 132 120 137 145 128 134 126 125 141
##  [783] 129 143 131 135 127 138 123 149 116 134 132 150 133 157 137 144 130
##  [800] 145 145 125 127 133 133 137 128 131 138 125 134 134 141 142 148 136
##  [817] 129 139 128 116 139 114 148 129 133 149 145 140 142 125 139 128 140
##  [834] 135 149 146 128 140 134 130 165 126 118 139 138 138 150 146 144 137
##  [851] 128 125 117 148 143 133 131 123 124 138 128 149 120 146 130 142 147
##  [868] 144 139 137 123 126 147 122 128 120 144 143 113 110 127 148 135 113
##  [885] 143 142 123 126 141 128 133 138 144 142 135 134 124 123 128 119 139
##  [902] 143 129 143 121 141 126 124 147 137 154 118 130 153 114 123 130 122
##  [919] 131 136 115 134 134 124 131 125 140 124 140 133 121 139 140 143 141
##  [936] 126 131 127 128 138 128 131 132 133 143 140 132 125 148 125 134 141
##  [953] 137 148 138 132 124 125 128 129 121 113 142 123 153 120 140 127 136
##  [970] 138 135 129 138 136 119 132 133 128 108 136 135 119 142 127 139 122
##  [987] 128 128 140 135 134 123 131 139 128 133 134 120 137 117
```

```
# If you can't produce a data vector, uncomment the following line of code
# and use it for the rest of the questions:

# data <- rnorm(1000, 133, 9)
```

(c) (6 points) Note that if a random variable X is the number of green balls selected in 400 draws with replacement from the urn, then X follows a binomial distribution, namely $X \sim bin(n, p)$ where p is the probability of selecting a green ball from the urn in a single draw, n is the total number of draws, and

$$Pr(X = x) = \binom{n}{x} p^x (1-p)^{n-x} \quad \text{for} \quad x = 0, 1, \ldots, 400,$$

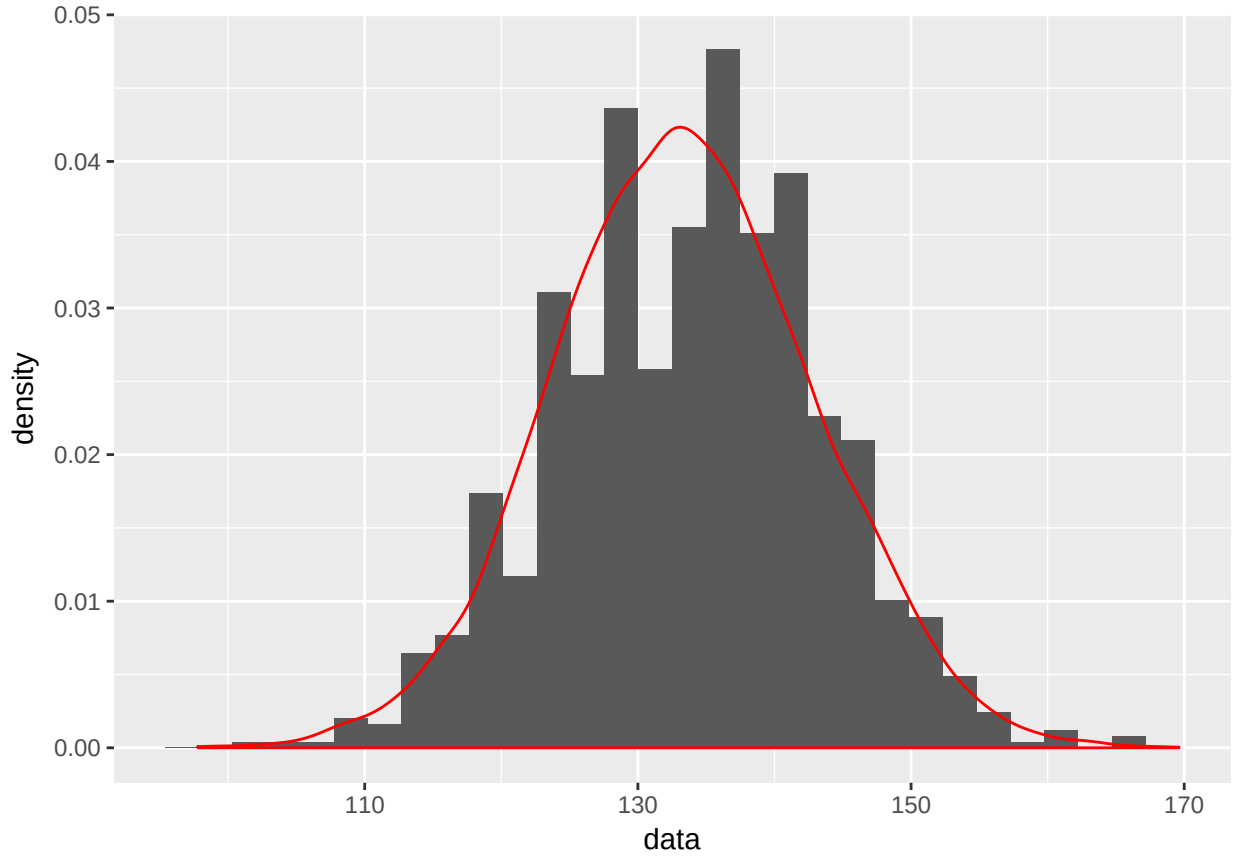
with $\mathbb{E}[X] = np$. Recall that the binomial distribution is well-approximated by the normal distribution. To see that this is a good approximation, plot a histogram of your data from 1.b along with a normal density curve colored red having mean $np = 400 * (1/3)$ and variance $np(1-p) = 400 * (1/3) * (2/3)$.

```
# Your answer to question 1.c here.
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.4.2
```

```
samp_norm <- rnorm(10000,400*(1/3), sqrt(400*(1/3)*(2/3)) )
ggplot() +
  geom_histogram(aes(x = data, y = ..density..)) +
  geom_density(aes(x = samp_norm), col = "red")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



(d) (5 points) Give the proportion of values in your data vector that are less than or equal to 100 or greater than or equal to 150. Using R functions for probability distributions, compare this proportion to the probability that a normal random variable having mean $np = 400 * (1/3)$ and variance $np(1-p) = 400 * (1/3) * (2/3)$ is less than 100 or greater than 150.

```
# Your answer to question 1.d here.
mean(data<=100 | data >=150)
```

```
## [1] 0.046
```

```
mu = 400*(1/3)  #mean of normal
sigma = sqrt(400*(1/3)*(2/3))  #sd of normal
pnorm(100,mu,sigma) + 1 - pnorm(150,mu,sigma)
```

```
## [1] 0.03875341
```

(e) (5 points) Write a function `MomentEstimator` that takes two input: `data`, a vector containing the number of green balls selected in each experiment, and `n` the total number of balls selected in each experiment (in 1.a, $n = 400$ but we write the function where this could change) and returns a single output value `phat` that is the method of moments estimate of of the probability p. After the function is written run the code `MomentEstimator(data, 400)` to see the method of moment estimator from your

simulated data in 1.b and the code `MomentEstimator(80, 100)` to check the functionality.

```
# Your answer to question 1.e here.
MomentEstimator <- function(data, n){
  phat <- mean(data) / n
  return(phat)
}
MomentEstimator(data,400)
```

```
## [1] 0.3344825
```

```
MomentEstimator(80,100)
```

```
## [1] 0.8
```

(f) (7 points) If num is the number of experiments run (i.e. in 1.b num is 1000) and x_i is the number of green balls selected in experiment $i = 1, 2, \ldots, num$, then the log-likelihood for the binomial distribution is given by the following:

$$\ell(p) = \sum_{i=1}^{num} \left(\log \binom{n}{x_i} + x_i \log p + (n - x_i) \log(1 - p) \right).$$

Find the MLE estimate by writing a function that calculates the negative log-likelihood and then using `nlm()` to minimize it. Find the MLE estimate in this way on your data from part 1.b. Use an initial guess of $p = 0.5$.

```
# Your answer to question 1.f here.
Neg_log_lik <- function(data, p){
  n <- length(data)
  log_lik <- sum(log(choose(n,data)) + data*log(p) + (n-data)*log(1-p) )
  return (-log_lik)
}
nlm(Neg_log_lik,  p = 0.5, data = data)
```

```
## $minimum
## [1] 3692.69
##
## $estimate
## [1] 0.133793
##
## $gradient
## [1] 0.001180031
##
## $code
## [1] 1
##
## $iterations
## [1] 7
```

```
nlm(Neg_log_lik,  p = 0.5, data = data)$estimate
```

```
## [1] 0.133793
```

(g) (10 points) Use the bootstrap procedure to estimate the variance of your method of moments estimator. Use 5000 bootstrap resamples of the data (stored in vector `data`) you calculated in 1.b. The actual variance of the method of moments estimator is $5.56e-07$.

```
set.seed(3)
```

```r
# Your answer to question 1.g here.  Don't remove the set.seed(3) command.
B <- 5000
n <- length(data)
param_ests <- rep(NA, B)
for (b in 1:B) {
  resamp <- sample(1:n, n, replace = TRUE)
  param_ests[b] <- MomentEstimator(data[resamp], 400)
}
var(param_ests)
```

```
## [1] 5.983345e-07
```

(h) (8 points) Use simulation to provide evidence that the method of moments estimate is consistent (meaning that as the sample size increases *num*, the estimator converges to the population value).

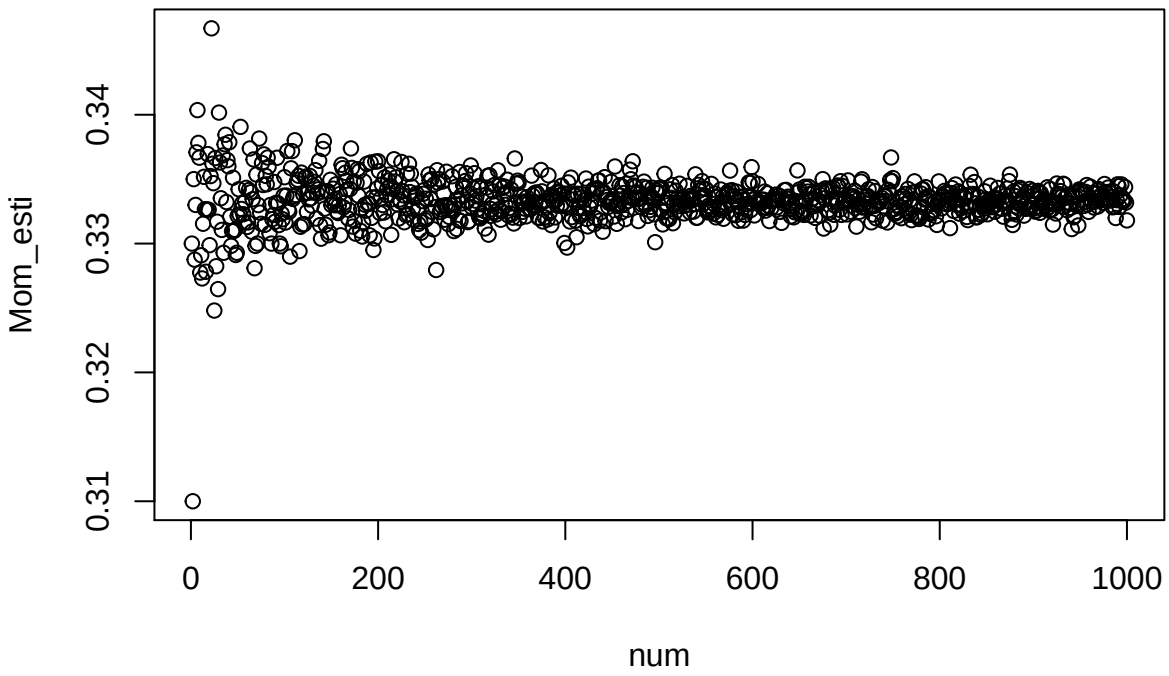
```r
set.seed(4)

# Your answer to question 1.h here.  Don't remove the set.seed(4) command.
##first generate sample of size num
generate_samp <- function(num){
  data <- rep(NA, num)
  for (i in 1:num){
    num_total_b <- sample(c(rep('r',10),rep('b',10),rep('g',10)), 400,
                          replace = TRUE)
    data[i] <- sum(num_total_b == 'g')
  }
  return(data)
}
##as num range from 1:5000, compute the moment estimator
Mom_esti <- c()
for (i in 1:1000){
  esti <- MomentEstimator(generate_samp(i),400)
  Mom_esti <- c(Mom_esti, esti)
}
mean(Mom_esti)
```

```
## [1] 0.333313
```

```r
var(Mom_esti)
```

```
## [1] 2.997921e-06
```

```r
plot(Mom_esti, xlab = "num")
```



We can see the results above, as num goes larger, the moment estimator converge to the population value with variation becomes smaller and smaller.

Question 2: Transforming Data (46 points)

Gross domestic product (GDP) is a measure of the total market value of all goods and services produced in a given country in a given year. The percentage growth rate of GDP in year t is

$$100 \times \left(\frac{GDP_{t+1} - GDP_t}{GDP_t} \right) - 100$$

An important claim in economics is that the rate of GDP growth is closely related to the level of government debt, specifically with the ratio of the government's debt to the GDP. The file `debt.csv` contains measurements of GDP growth and of the debt-to-GDP ratio for twenty countries around the world, from the 1940s to 2010. Note that not every country has data for the same years, and some years in the middle of the period are missing data for some countries but not others.

```
debt <- read.csv("debt.csv", as.is = TRUE)
dim(debt)
```

```
## [1] 1171    4
```

```
head(debt)
```

```
##     Country Year    growth     ratio
## 1 Australia 1946 -3.557951 190.41908
## 2 Australia 1947  2.459475 177.32137
## 3 Australia 1948  6.437534 148.92981
## 4 Australia 1949  6.611994 125.82870
## 5 Australia 1950  6.920201 109.80940
## 6 Australia 1951  4.272612  87.09448
```

(a) (5 points) Calculate the average GDP growth rate for each year (averaging over countries). This is a classic split/apply/combine problem, and you should use `split()` and a function form the apply family of functions to solve it. You should not need to use a loop to do this. (The average growth rates for 1972 and 1989 should be 5.63 and 3.19, respectively. Print these values in your output.)

```
# Your answer to question 2.a here.
ratio.by.year <- split(debt,f = debt$Year)

mean.fun <- function(df) {
  return(mean(df$growth))
}
aveg_growth <- lapply(ratio.by.year, mean.fun)
aveg_growth$`1972`
```

```
## [1] 5.629986
```

```
aveg_growth$`1989`
```

```
## [1] 3.186842
```

(b) (5 points) Calculate the average GDP growth rate for each year (averaging over countries). This is a classic split/apply/combine problem, and you should use `ddply()` to solve it. Save your output as `year.avgs` and change the column names to be `Year` and `AverageGrowth`. You should not need to use a loop to do this. (The average growth rates for 1972 and 1989 should be 5.63 and 3.19, respectively. Print these values in your output.)

```
# Your answer to question 2.b here.
library(plyr)
```

```
## Warning: package 'plyr' was built under R version 3.4.1
```

```
my.mean.func <- function(data) {
  return(mean(data$growth))
}
year.avgs <- ddply(debt, .(... = Year), my.mean.func)
names(year.avgs) <- c("Year", "AverageGrowth")
year.avgs$AverageGrowth[year.avgs$Year==1972]
```

```
## [1] 5.629986
```

```
year.avgs$AverageGrowth[year.avgs$Year==1989]
```

```
## [1] 3.186842
```

(c) (4 points) The `year.avgs` dataframe from 2.b will be sorted by Year, meaning row 1 corresponds to 1946, row 2 to 1947, and so on with row 64 corresponding to 2009. Produce a dataframe that instead has row 1 corresponding to the year with the largest average growth, row 2 to the year with the second largest average growth, and so on with row 64 corresponding to the year with the smallest average growth.

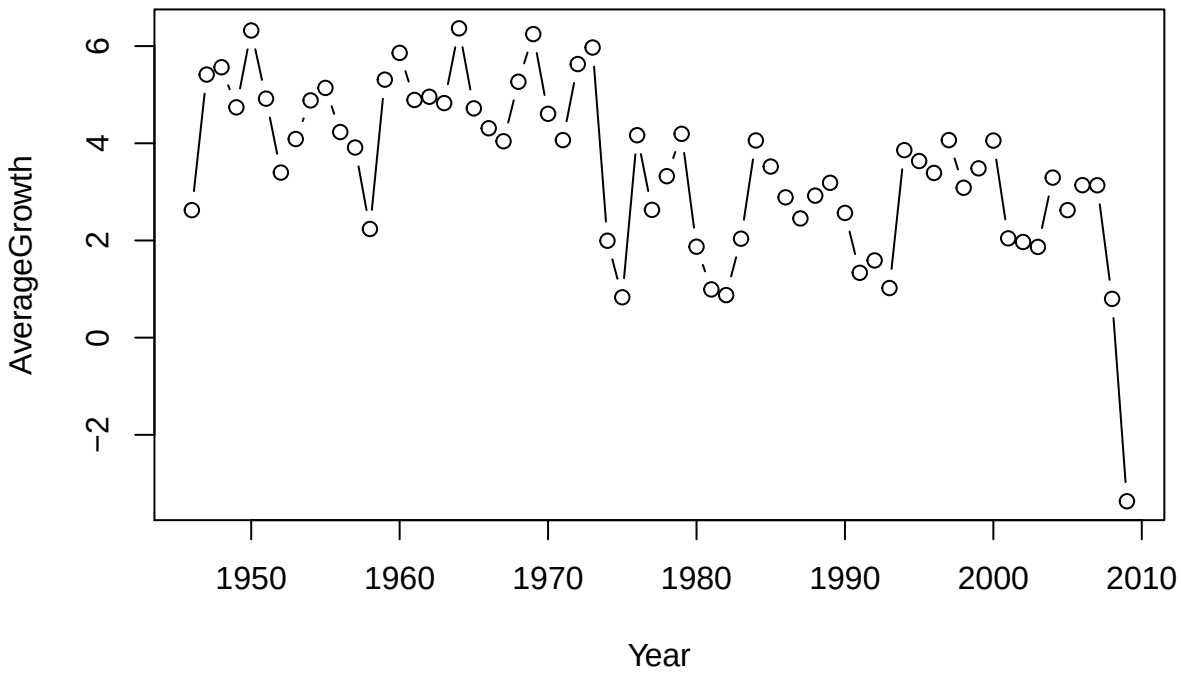
```
# Your answer to question 2.c here.
year.avgs[order(year.avgs$AverageGrowth, decreasing = T), ]
```

```
##    Year AverageGrowth
## 19 1964     6.3654718
## 5  1950     6.3214896
## 24 1969     6.2470505
## 28 1973     5.9712432
## 15 1960     5.8604385
## 27 1972     5.6299862
## 3  1948     5.5648414
## 2  1947     5.4147299
## 14 1959     5.3098167
## 23 1968     5.2665878
## 10 1955     5.1396220
## 17 1962     4.9571904
## 6  1951     4.9184456
## 16 1961     4.8915229
## 9  1954     4.8828652
## 18 1963     4.8275013
## 4  1949     4.7396296
## 20 1965     4.7188763
## 25 1970     4.6064498
## 21 1966     4.3093773
## 11 1956     4.2313542
## 34 1979     4.1939645
## 31 1976     4.1659118
## 8  1953     4.0873110
## 26 1971     4.0655311
## 52 1997     4.0654455
## 39 1984     4.0582113
## 55 2000     4.0559841
## 22 1967     4.0422048
## 12 1957     3.9128688
## 49 1994     3.8585838
## 50 1995     3.6340300
## 40 1985     3.5210599
```

```
## 54 1999     3.4843512
## 7  1952     3.3976694
## 51 1996     3.3896732
## 33 1978     3.3230568
## 59 2004     3.2936823
## 44 1989     3.1868422
## 61 2006     3.1381842
## 62 2007     3.1359031
## 53 1998     3.0850886
## 43 1988     2.9223717
## 41 1986     2.8879720
## 32 1977     2.6299752
## 1  1946     2.6239890
## 60 2005     2.6239322
## 45 1990     2.5665909
## 42 1987     2.4530780
## 13 1958     2.2362356
## 56 2001     2.0436501
## 38 1983     2.0365803
## 29 1974     1.9944636
## 57 2002     1.9685731
## 35 1980     1.8711923
## 58 2003     1.8670089
## 47 1992     1.5891679
## 46 1991     1.3348964
## 48 1993     1.0208583
## 36 1981     0.9920489
## 37 1982     0.8758437
## 30 1975     0.8301904
## 63 2008     0.7980262
## 64 2009    -3.3668270
```

(d) (3 points) Make a plot of the growth rates (y-axis) versus the year (x-axis) using your results from either 2.a or 2.b (they should be the same). Make sure the axes are labeled appropriately.

```
# Your answer to question 2.d here.
plot(year.avgs$Year, year.avgs$AverageGrowth, type = 'b',
     xlab = "Year", ylab = "AverageGrowth")
```

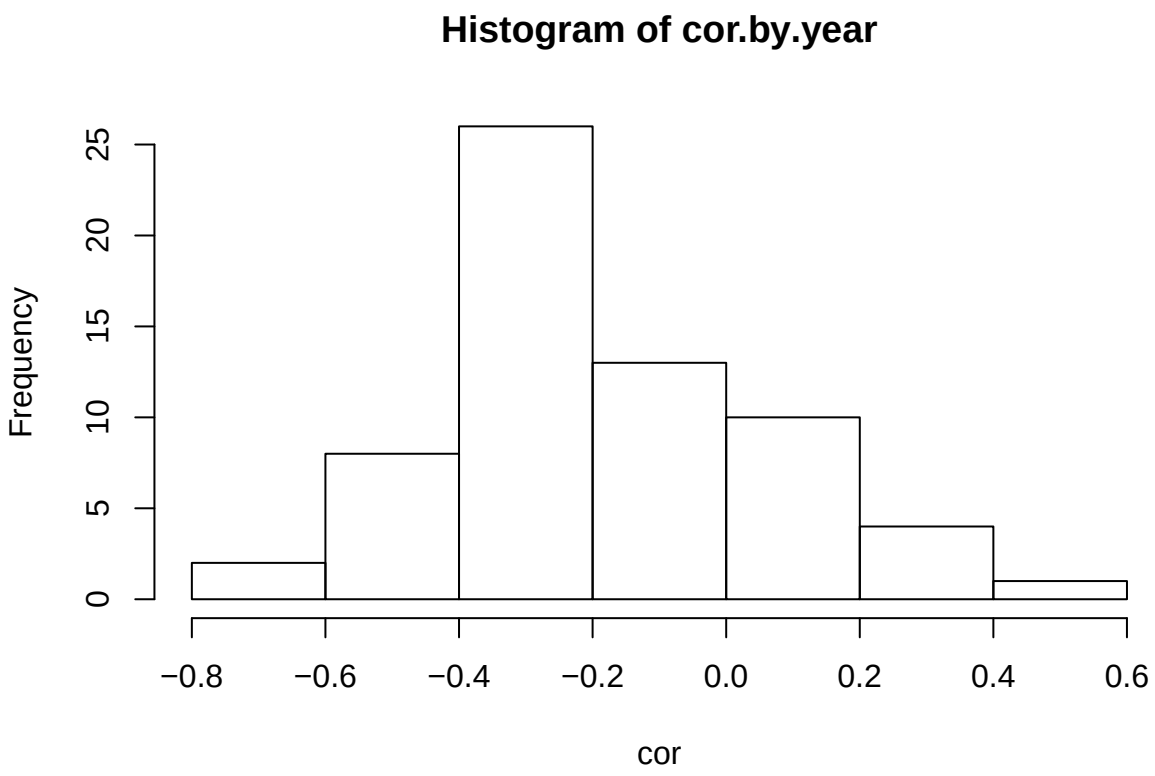


(e) (6 points) The function `cor(x,y)` calculates the correlation coefficient between two vectors `x` and `y`. First calculate the correlation coefficient between GDP growth and the debt ratio over the whole data set (all countries, all years). Your answer should be −0.1995. Second, compute the correlation coefficient separately for each year, and plot a histogram of these coefficients. The mean of these correlations should be −0.1906. Do not use a loop.

```
# Your answer to question 2.e here.
cor(debt$growth,debt$ratio)
```

```
## [1] -0.199468
```

```
my.cor <- function(data){
  return(cor(data$growth,data$ratio))
}
cor.by.year <- sapply(split(debt, debt$Year), my.cor)
hist(cor.by.year,xlab = "cor")
```




```
mean(cor.by.year)
```

```
## [1] -0.1905526
```

(f) (3 points) Some economists claim that high levels of government debt cause slower growth. Other economists claim that low economic growth leads to higher levels of government debt. The data file, as given, lets us relate this year's debt to this year's growth rate; to check these claims, we need to relate current debt to future growth. Create a new dataframe that contains all the rows of `debt` for France. It should have 54 rows and 4 columns. Note that some years are missing from the middle of this data set.

```
# Your answer to question 2.f here.
debt.france <- debt[debt$Country=="France",]
dim(debt.france)
```

```
## [1] 54  4
```

(g) Create a new column in your dataframe for France created in 2.f, labeled `next.growth`, which gives next year's growth *if* the next year is in the data frame, or `NA` if the next year is missing. Do this in two steps.

1. (7 points) First write a function `n.growth()` that takes in two arguments, a year and a dataframe (that has the same columns as `debt` but rows only corresponding to a single country), and outputs the proper next growth value for that year and that dataframe (i.e. it gives next year's growth if the next year is in the input dataframe, or `NA` if the next year is missing).

```
# Your answer to question 2.g.1 here.
n.growth <- function(year, data){
  logic.stat <- sum(data$Year==year+1)>0
  if(logic.stat){
```

```
    return(data$growth[which(data$Year==year+1)])
  }else{
    return(NA)
  }
}
```

2. (5 points) Next use `n.growth()` and one of the functions from the apply family to create the `next.growth` column in the dataframe. (`next.growth` for 1971 should be 5.886, but for 1972 it should be `NA`.)

```
# Your answer to question 2.g.2 here.
yearrange <- debt.france$Year
next.growth <- sapply(yearrange, n.growth, data = debt.france)
next.growth[which(yearrange==1971)]
```

```
## [1] 5.885827
```

```
next.growth[which(yearrange==1972)]
```

```
## [1] NA
```

(h) (8 points) Add a `next.growth` column, as in 2.g, to the whole of the `debt` data frame. Make sure that you do not accidentally put the first growth value for one country as the `next.growth` value for another. (The `next.growth` for France in 2009 should be `NA`, not 9.167 .) Hints: Write a function to encapsulate what you did in 2.f, and apply it using `ddply()`.

```
# Your answer to question 2.g here.
my.next.growth <- function(data.country){
  year <- unique(debt$Year)
  next.growth <- sapply(year, n.growth, data = data.country)
  return(next.growth)
}
next.growth1 <- ddply(debt, .(Country), my.next.growth)
names(next.growth1) <- c("Country",unique(debt$Year))
next.growth <- rep(NA, length(debt$Year))
for (i in 1:nrow(debt)){
  xloc <- which(next.growth1$Country==debt$Country[i])
  yloc <- which(names(next.growth1)==as.character(debt$Year[i]))
  value = next.growth1[xloc,yloc]
  next.growth[i] <- value
}
debt <- cbind(debt,next.growth)
head(debt)
```

```
##     Country Year    growth     ratio next.growth
## 1 Australia 1946 -3.557951 190.41908   2.4594746
## 2 Australia 1947  2.459475 177.32137   6.4375341
## 3 Australia 1948  6.437534 148.92981   6.6119938
## 4 Australia 1949  6.611994 125.82870   6.9202012
## 5 Australia 1950  6.920201 109.80940   4.2726115
## 6 Australia 1951  4.272612  87.09448   0.9046516
```