Homework 3

BBIC: Bovee, Bichay, Iseneker, Coeman

2/8/2018

Problem 1

OLS function

First we simulate data for our LM

```
### simulate data

library(MASS)
set.seed(6886)
data = mvrnorm(
    n=100, mu=c(-2, 3),
    Sigma=matrix(c(8,3,3,2),nrow=2,ncol=2)
)
colnames(data) = c('y','x')
```

Here we create the function that calculates an OLS model manually. Comments within explain each step.

```
### create function

my_lm <- function(q, p){


# load data

x = as.matrix(cbind(1,p))
y = as.matrix(q)

# calculate b

beta = solve( t(x) %*% x) %*%t (x) %*% y

# calculate vcv and se

yhat <- x %*% beta    # predicted values

ssr <- sum((yhat - y)^2)    # sum of squared residuals

# set N = number of observations; k = number of variables (incl. intercept)

N <- nrow(y)
k <- ncol(x)

sigma2 <- (ssr/(N-k))    # variance

vcv <- (sigma2)*(solve( t(x) %*% x ))    # vcv
se <- sqrt(diag(vcv))

# output results

model_summary <- matrix(c(beta, se), nrow=2, ncol=2,
                        dimnames=list( c("b0", "b1"),
                            c("Coef", "S.E."))
                        )

return(model_summary)

}

my_lm(data[,"y"], data[,"x"])
```

```
##         Coef      S.E.
## b0 -6.390654 0.4681754
## b1  1.499807 0.1309778
```

Problem 2

"Create this" matrix, from slide 37

Use the rep function to fill the data in the data frame.

```
clever_df <- data.frame(Patient = c(1:16),
                          Gender = rep(c("Male","Female"), each = 8),
                          Treatment1 = rep(c("Yes", "No"), each = 4),
                          Treatment2 = rep(c("Yes", "No"), each = 2),
                          Treatment3 = rep(c("Yes", "No"))
                          )
clever_df
```

```
##    Patient Gender Treatment1 Treatment2 Treatment3
## 1        1   Male        Yes        Yes        Yes
## 2        2   Male        Yes        Yes         No
## 3        3   Male        Yes         No        Yes
## 4        4   Male        Yes         No         No
## 5        5   Male         No        Yes        Yes
## 6        6   Male         No        Yes         No
## 7        7   Male         No         No        Yes
## 8        8   Male         No         No         No
## 9        9 Female        Yes        Yes        Yes
## 10      10 Female        Yes        Yes         No
## 11      11 Female        Yes         No        Yes
## 12      12 Female        Yes         No         No
## 13      13 Female         No        Yes        Yes
## 14      14 Female         No        Yes         No
## 15      15 Female         No         No        Yes
## 16      16 Female         No         No         No
```

Problem 3

Polity matrix and summary stats

We first load the data, and then store the various statistics in resepctive vectors

```
load("/Users/zacharycoeman/Desktop/900/polity_dataframe.rda")

mean <- c(mean(polity$democ, na.rm=TRUE), mean(polity$autoc, na.rm=TRUE),
             mean(polity$polity2, na.rm=TRUE), mean(polity$xconst, na.rm=TRUE))

median <- c(median(polity$democ, na.rm=TRUE), median(polity$autoc, na.rm=TRUE),
             median(polity$polity2, na.rm=TRUE), median(polity$xconst, na.rm=TRUE))

stdev <- c(sd(polity$democ, na.rm=TRUE), sd(polity$autoc, na.rm=TRUE),
             sd(polity$polity2, na.rm=TRUE), sd(polity$xconst, na.rm=TRUE))

max <- c(max(polity$democ, na.rm=TRUE), max(polity$autoc, na.rm=TRUE),
             max(polity$polity2, na.rm=TRUE), max(polity$xconst, na.rm=TRUE))

min <- c(min(polity$democ, na.rm=TRUE), min(polity$autoc, na.rm=TRUE),
             min(polity$polity2, na.rm=TRUE), min(polity$xconst, na.rm=TRUE))

n <- c(NROW(na.omit(polity$democ)), NROW(na.omit(polity$autoc)),
             NROW(na.omit(polity$polity2)), NROW(na.omit(polity$xconst)))

nNA <- c(sum(is.na(polity$democ)), sum(is.na(polity$autoc)),
             sum(is.na(polity$polity2)), sum(is.na(polity$xconst)))
```

Next, we create a matrix from the 7 vectors

```
summary  <- matrix(c(n, nNA, mean, median, max, min, stdev), nrow=4, ncol=7,
                    dimnames=list(names(polity[5:8]),
                            c("N", "nNA", "Mean", "Median", "Max", "Min", "StDev"))
                )
```

And finally, we round the matrix to two decimal places, and transpose it so it matches the matrix in the slide

```
summary <- round(t(summary), 2)

summary
```

```
##          democ   autoc polity2  xconst
## N      8122.00 8122.00 8385.00 8122.00
## nNA     357.00  357.00   94.00  357.00
## Mean      4.27    3.38    0.85    4.19
## Median    3.00    2.00    1.00    4.00
## Max      10.00   10.00   10.00    7.00
## Min       0.00    0.00  -10.00    1.00
## StDev     4.17    3.57    7.43    2.33
```

Problem 4

Merging the other variables

We first load in the two datasets

```
worldBank <- read.csv( file='/Users/zacharycoeman/Desktop/900/worldBank.csv', stringsAsFactors=FALSE
)

load("/Users/zacharycoeman/Desktop/900/polity_dataframe.rda")



colnames(worldBank)[4:7] = c( 'gdpGrowth', 'gdppcConstant', 'popGrowth', 'tradeOpen'
)
```

Next, we use the package countrycode to make a matchable country name variable, and then account for the time-series nature of the data by creating a country-year indicator

```
library(countrycode)
polity$cname <- countrycode(
    sourcevar=polity$country,
    origin='country.name', destination='country.name')
```

```
## Warning in countrycode(sourcevar = polity$country, origin = "country.name", : Some values were not ma

## Warning in countrycode(sourcevar = polity$country, origin = "country.name", : Some strings were matcl
```

```
worldBank$cname <- countrycode(
    worldBank$country,
    'country.name', 'country.name')
```

```
## Warning in countrycode(worldBank$country, "country.name", "country.name"): Some values were not matcl
```

```
head(unique(polity[,c('country','cname')]))
```

```
##         country       cname
## 161 Afghanistan Afghanistan
## 264     Albania     Albania
## 321     Algeria     Algeria
## 376      Angola      Angola
## 553   Argentina   Argentina
## 610     Armenia     Armenia
```

```
polity$cyear <- paste(
    polity$cname, polity$year,
    sep='_'
    )
worldBank$cyear <- paste(
    worldBank$cname, worldBank$year,
    sep='_'
    )
```

This matches the GDP growth variable, as we did in class

```
polity$gdpGrowth <- worldBank$gdpGrowth[
    match(
        polity$cyear, worldBank$cyear
        )
]

matched <- head(match(polity$cyear, worldBank$cyear))

polity$cyear[1:6]
```

```
## [1] "Afghanistan_1960" "Afghanistan_1961" "Afghanistan_1962"
## [4] "Afghanistan_1963" "Afghanistan_1964" "Afghanistan_1965"
```

```
worldBank[matched,'cyear']
```

```
## [1] "Afghanistan_1960" "Afghanistan_1961" "Afghanistan_1962"
## [4] "Afghanistan_1963" "Afghanistan_1964" "Afghanistan_1965"
```

And finally, we use a loop to match the remaining variables

```
worldBank <- worldBank[!is.na(worldBank$cname),]    # remove NAs (from addtional regions that were in w

to_match <- list("gdppcConstant", "popGrowth", "tradeOpen")

for(i in to_match){

    polity[,i] <- worldBank[,i] [
    match(
        polity$cyear, worldBank$cyear
        )
]
}

head(polity)
```

```
##     ccode scode     country year democ autoc polity2 xconst       cname
## 161   700   AFG Afghanistan 1960     0    10     -10      1 Afghanistan
## 162   700   AFG Afghanistan 1961     0    10     -10      1 Afghanistan
## 163   700   AFG Afghanistan 1962     0    10     -10      1 Afghanistan
## 164   700   AFG Afghanistan 1963     0    10     -10      1 Afghanistan
## 165   700   AFG Afghanistan 1964     0     7      -7      3 Afghanistan
## 166   700   AFG Afghanistan 1965     0     7      -7      3 Afghanistan
##                cyear gdpGrowth gdppcConstant popGrowth tradeOpen
## 161 Afghanistan_1960        NA            NA  1.816077  11.15703
## 162 Afghanistan_1961        NA            NA  1.876528  12.55061
## 163 Afghanistan_1962        NA            NA  1.934999  14.22764
## 164 Afghanistan_1963        NA            NA  1.992521  26.03551
## 165 Afghanistan_1964        NA            NA  2.049423  26.94445
## 166 Afghanistan_1965        NA            NA  2.105369  32.67108
```

Problem 5

Hadley problems

Data structures:

####Vectors: 1. What are the six types of atomic vector?
logical, integer, double, character, complex and raw How does a list differ from an atomic vector? lists can include any element type, including other lists

2. What makes is.vector() and is.numeric() fundamentally different to is.list() and is.character()? is.vector and is.numeric do not test for a specific type of element
3. Test your knowledge of vector coercion rules by predicting the output of the following uses of c(): c(1, FALSE) <- double c("a", 1) <- character c(list(1), "a") <- list c(TRUE, 1L) <- integer
4. Why do you need to use unlist() to convert a list to an atomic vector? Why doesn't as.vector() work? Lists are vectors, but not atomic vectors, so as.vector doesn't do anything as it is coercing a vector to a vector.
5. Why is 1 == "1" true? Why is -1 < FALSE true? Why is "one" < 2 false? 1 == "1": the numeric is coerced into a character - moving up from a less flexible element to more -1 < F: FALSE is coerced into an integer, become 0, making the statement -1 < 0 "one" < 2: the numeric is coerced into a character and "one" < "2" is then FALSE
6. Why is the default missing value, NA, a logical vector? What's special about logical vectors? (Hint: think about c(FALSE, NA_character_).) Logical is the least flexible type, so NA are logical so they don't lose any information when coerced.

####Attributes: 1. An early draft used this code to illustrate structure(): structure(1:5, comment = "my attribute") [1] 1 2 3 4 5 But when you print that object you don't see the comment attribute. Why? Is the attribute missing, or is there something else special about it? (Hint: try using help.) the "comment" attribute is a special kind that does not print, unlike the others

2. What happens to a factor when you modify its levels? f1 <- factor(letters) levels(f1) <- rev(levels(f1)) It reversed both the levels and the factor
3. What does this code do? How do f2 and f3 differ from f1? f2 reverses the factor, but not the levels f3 reverses the levels, but not the factor

####Matrices and Arrays 1. What does dim() return when applied to a vector? NULL

2. If is.matrix(x) is TRUE, what will is.array(x) return? TRUE, an array is a matrix
3. How would you describe the following three objects? What makes them different to 1:5? x1 <- array(1:5, c(1, 1, 5)) a 1x1x5 array x2 <- array(1:5, c(1, 5, 1)) a 1x5x1 array x3 <- array(1:5, c(5, 1, 1)) a 5x1x1 array They differ from 1:5 in that they have dimensions (3)

####Data Frames: 1. What attributes does a data frame possess? names, row.names, class

2. What does as.matrix() do when applied to a data frame with columns of different types? Coerces everything into the same type.
3. Can you have a data frame with 0 rows? What about 0 columns? Yes, an empty data frame

Subsetting:

####Data Types:

1. Fix each of the following common data frame subsetting errors: mtcars[mtcars$cyl = 4,] -> mtcars[mtcars$cyl == 4,] mtcars[-1:4,] -> mtcars[-(1:4),] mtcars[mtcars$cyl <= 5] -> tcars[mtcars$cyl <= 5,] mtcars[mtcars$cyl == 4 | 6,] -> mtcars[mtcars$cyl == 4|mtcars$cyl == 6,]

2. Why does x <- 1:5; x[NA] yield five missing values? (Hint: why is it different from x[NA_real_]?) X takes NA and expands it to a length of 5

3. What does upper.tri() return? How does subsetting a matrix with it work? Do we need any additional subsetting rules to describe its behaviour? x <- outer(1:5, 1:5, FUN = "*") x[upper.tri(x)] upper.tri() returns a logical matrix the size of the upper triangle of the matrix. Using it to subset the matrix, is in x[upper.tri(x)], returns a vector of the subsetted part of the matrix

4. Why does mtcars[1:20] return an error? How does it differ from the similar mtcars[1:20,]? mtcars[1:20] tells R to look at 20 columns, but there are not that many columns in mtcars. mtcars[1:20,] tells R to look at rows 1:20 and all the columns

5. Implement your own function that extracts the diagonal entries from a matrix (it should behave like diag(x) where x is a matrix).

my_diag <- function(x){ n <- nrow(x) return(x[matrix(seq_len(n), nrow = n, ncol = 2)]) }

6. What does df[is.na(df)] <- 0 do? How does it work? subsets a dataframe to the values that are NA and turns NAs into 0s.

####Subsetting Operators

1. Given a linear model, e.g., mod <- lm(mpg ~ wt, data = mtcars), extract the residual degrees of freedom. Extract the R squared from the model summary (summary(mod)) mod$df.residual summary(mod)$r.squared

####Applications

1.How would you randomly permute the columns of a data frame? (This is an important technique in random forests.) Can you simultaneously permute the rows and columns in one step? df[,sample(ncol(df))] df[sample(nrow(df)),sample(ncol(df))]

2. How would you select a random sample of m rows from a data frame? What if the sample had to be contiguous (i.e., with an initial row, a final row, and every row in between)? m <- some number testdf[sample(nrow(testdf), size = m),] testdf[sample(nrow(testdf) - m + 1, size = 1) + (0:(m - 1)),]

3. How could you put the columns in a data frame in alphabetical order? testdf[order(names(testdf))]