Additional Exercises

Intro to Data Science for Public Policy, Spring 2016

by Jeff Chen & Dan Hammer, Georgetown University McCourt School of Public Policy

Contents

Additional Exercises! + Answers below

1. Write a function "unique2" to calculate the number of unique values in the following series. The series below is normally distributed ($n = 100$ with $\mu = 50$ and $\sigma = 10$), rounded to the nearest integer.

```
set.seed(123)
x <- round(rnorm(100,50,10))
```

2. Imputation of missing values is often times involves replacing unknown values with the mean, median, or mode. Mean can be calculated using `mean()`, median using `percentile()`, and mode using a combination of `table()`, `sort()`, `names()`. For categorical data, the mode is most appropriate. In the vectors below, replace all `NA` values with the most common string value in `y1` and the mean for the `y2`.

```
y1 <- c("a","a",NA,"c","d","e", NA,"f", NA,"g","a", NA, "a","c","z","g")
y2 <- c(1, 10, NA, 5, 6, 1, NA, NA, 9, NA, 15, 3, NA, NA, 3, 9, 2)
```

3. Convert your imputation code into a function named `plug.it()` that accepts a vector of any kind. The logic of the function should be as follows:

- function(vec)
- if class of vec is numeric, then impute with mean
- if class of vec is character, factor, or logical, then impute with mode
- return imputed set

Try out your function below.

```
y3 <- c("z","z",NA,"c","d","e", NA,"f", NA,"s","z", NA, "a","c","z","g")
y4 <- c(2, 11, NA, 3, NA, 5, 6, 1, NA, NA, 9, NA, 16, 3, NA, NA, 2, 4, 3)
```

4. Write a loop to impute each column of the following data frame using `plug.it()`. Since your function is flexible, all you'd need to do is loop column 1 through 3.

```
df <- data.frame( x1 = c(2, 11, NA, 3, NA, 5, 6, 1, NA, NA, 9, NA, 16, 3, NA, NA, 2, 4, 3),
                  x2 = c(20, NA, NA, 30, NA, 15, 6, 1, 10, 11, 9, 2, 16, 3, 400, 500, 2, 4, 3),
                  x3 = c(228, NA, NA, 39, NA, 2, 6, 1, 2, 5, 3, 2, NA, 3, NA, NA, 5, 2, 34))
```

Answers!

1. Answer. Steps:

- create a placeholder `p`
- loop through all values of vector `vec`
- if index `i` is not in placeholder `p`, then append index `i` to end of `p`
- when done, return `p`

```
  unique2 <- function(vec){
    #create placeholder
      p <- c()

    #loop
      for(i in vec){
        if(!(i %in% p)){
          p <- c(p, i)
        }
      }

    #return
      return(p)
  }

#test it
  set.seed(123)
  x <- round(rnorm(100,50,10))
  a <- unique2(x)
```

2. Answer!

- Use `table()` to return a list of unique values with their frequencies
- Sort the table descending, extract the first value (letter)
- Replace all blanks with the letter

```
#String
  y1 <- c("a","a",NA,"c","d","e", NA,"f", NA,"g","a", NA, "a","c","z","g")
  tab <- table(y1)
  letter <- names(sort(tab,decreasing=TRUE))[1]
  y1[is.na(y1)] <- letter
  print(y1)
```

```
##  [1] "a" "a" "a" "c" "d" "e" "a" "f" "a" "g" "a" "a" "a" "c" "z" "g"
```

```
#Numeric
  y2 <- c(1, 10, NA, 5, 6, 1, NA, NA, 9, NA, 15, 3, NA, NA, 3, 9, 2)
  mu <- mean(y2, na.rm = T)
  y2[is.na(y2)] <- mu
  print(y2)
```

```
##  [1]  1.000000 10.000000  5.818182  5.000000  6.000000  1.000000  5.818182
##  [8]  5.818182  9.000000  5.818182 15.000000  3.000000  5.818182  5.818182
## [15]  3.000000  9.000000  2.000000
```

3. Answer!

```
#Function
  plug.it <- function(vec){

    if(class(vec)=="numeric"){
      #numeric!
        mu <- mean(vec, na.rm = T)
        vec[is.na(vec)] <- mu

    } else if(class(vec) == "character" | class(vec) == "factor"){
```

```
      #string!
        tab <- table(vec)
        letter <- names(sort(tab,decreasing=TRUE))[1]
        vec[is.na(vec)] <- letter
    }
    return(vec)
  }

#Test
  y3 <- c("z","z",NA,"c","d","e", NA,"f", NA,"s","z", NA, "a","c","z","g")
  y4 <- c(2, 11, NA, 3, NA, 5, 6, 1, NA, NA, 9, NA, 16, 3, NA, NA, 2, 4, 3)

  plug.it(y3)
```

```
##  [1] "z" "z" "z" "c" "d" "e" "z" "f" "z" "s" "z" "z" "a" "c" "z" "g"
```

```
  plug.it(y4)
```

```
##  [1]  2.000000 11.000000  5.416667  3.000000  5.416667  5.000000  6.000000
##  [8]  1.000000  5.416667  5.416667  9.000000  5.416667 16.000000  3.000000
## [15]  5.416667  5.416667  2.000000  4.000000  3.000000
```

4. Answer!

```
  df <- data.frame( x1 = c(2, 11, NA, 3, NA, 5, 6, 1, NA, NA, 9, NA, 16, 3, NA, NA, 2, 4, 3),
                    x2 = c("a","a", NA,"c","d","e", NA,"f", NA,"g","a", NA, "a","c","z","g", "x", "g",
                    x3 = c(20, NA, NA, 30, NA, 15, 6, 1, 10, 11, 9, 2, 16, 3, 400, 500, 2, 4, 3),
                    x4 = c(228, NA, NA, 39, NA, 2, 6, 1, 2, 5, 3, 2, NA, 3, NA, NA, 5, 2, 34))
  #Loop it
  for(i in 1:ncol(df)){
    df[,i] <- plug.it(df[,i])
  }

  #view
  print(df)
```

```
##           x1 x2    x3        x4
## 1   2.000000  a  20.0 228.00000
## 2  11.000000  a  64.5  25.53846
## 3   5.416667  a  64.5  25.53846
## 4   3.000000  c  30.0  39.00000
## 5   5.416667  d  64.5  25.53846
## 6   5.000000  e  15.0   2.00000
## 7   6.000000  a   6.0   6.00000
## 8   1.000000  f   1.0   1.00000
## 9   5.416667  a  10.0   2.00000
## 10  5.416667  g  11.0   5.00000
## 11  9.000000  a   9.0   3.00000
## 12  5.416667  a   2.0   2.00000
## 13 16.000000  a  16.0  25.53846
## 14  3.000000  c   3.0   3.00000
## 15  5.416667  z 400.0  25.53846
## 16  5.416667  g 500.0  25.53846
## 17  2.000000  x   2.0   5.00000
## 18  4.000000  g   4.0   2.00000
## 19  3.000000  z   3.0  34.00000
```