Su_X_HW5

Xueqi Su

MATH 510 HW5

1.Print to the console all methods and attributes associates with a dataframe. Write code to determine the number of columns in a dataframe

```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.1.3
```

```
data("diamonds")
#use diamonds data to develop my code
mydata = diamonds
#get the general information of mydata
str(mydata)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    53940 obs. of  10 variables:
##  $ carat  : num  0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...
##  $ cut    : Ord.factor w/ 5 levels "Fair"<"Good"<..: 5 4 2 4 2 3 3 3 1 3 ...
##  $ color  : Ord.factor w/ 7 levels "D"<"E"<"F"<"G"<..: 2 2 2 6 7 7 6 5 2 5 ...
##  $ clarity: Ord.factor w/ 8 levels "I1"<"SI2"<"SI1"<..: 2 3 5 4 2 6 7 3 4 5 ...
##  $ depth  : num  61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9 65.1 59.4 ...
##  $ table  : num  55 61 65 58 58 57 57 55 61 61 ...
##  $ price  : int  326 326 327 334 335 336 336 337 337 338 ...
##  $ x      : num  3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 4 ...
##  $ y      : num  3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78 4.05 ...
##  $ z      : num  2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49 2.39 ...
```

```
summary(mydata)
```

```
##      carat               cut        color        clarity     
##  Min.   :0.2000   Fair      : 1610   D: 6775   SI1    :13065  
##  1st Qu.:0.4000   Good      : 4906   E: 9797   VS2    :12258  
##  Median :0.7000   Very Good:12082   F: 9542   SI2    : 9194  
##  Mean   :0.7979   Premium   :13791   G:11292   VS1    : 8171  
##  3rd Qu.:1.0400   Ideal     :21551   H: 8304   VVS2   : 5066  
##  Max.   :5.0100                     I: 5422   VVS1   : 3655  
##                                     J: 2808   (Other): 2531  
##      depth           table           price             x         
##  Min.   :43.00   Min.   :43.00   Min.   :  326   Min.   : 0.000  
##  1st Qu.:61.00   1st Qu.:56.00   1st Qu.:  950   1st Qu.: 4.710  
##  Median :61.80   Median :57.00   Median : 2401   Median : 5.700  
##  Mean   :61.75   Mean   :57.46   Mean   : 3933   Mean   : 5.731  
##  3rd Qu.:62.50   3rd Qu.:59.00   3rd Qu.: 5324   3rd Qu.: 6.540  
##  Max.   :79.00   Max.   :95.00   Max.   :18823   Max.   :10.740  
##                                                                  
##        y                z        
##  Min.   : 0.000   Min.   : 0.000  
```

```
##  1st Qu.: 4.720   1st Qu.: 2.910
##  Median : 5.710   Median : 3.530
##  Mean   : 5.735   Mean   : 3.539
##  3rd Qu.: 6.540   3rd Qu.: 4.040
##  Max.   :58.900   Max.   :31.800
##
```

```
#print all methods and attributes that are associated with mydata
methods(class=data.frame)
```

```
##  [1] $.data.frame             $<-.data.frame
##  [3] Math.data.frame          Ops.data.frame
##  [5] Summary.data.frame       [.data.frame
##  [7] [<-.data.frame           [[.data.frame
##  [9] [[<-.data.frame          aggregate.data.frame
## [11] anyDuplicated.data.frame as.data.frame.data.frame
## [13] as.list.data.frame       as.matrix.data.frame
## [15] by.data.frame            cbind.data.frame
## [17] dim.data.frame           dimnames.data.frame
## [19] dimnames<-.data.frame    droplevels.data.frame
## [21] duplicated.data.frame    edit.data.frame*
## [23] format.data.frame        formula.data.frame*
## [25] fortify.data.frame*      ggplot.data.frame*
## [27] head.data.frame*         is.na.data.frame
## [29] merge.data.frame         na.exclude.data.frame*
## [31] na.omit.data.frame*      plot.data.frame*
## [33] print.data.frame         prompt.data.frame*
## [35] rbind.data.frame         row.names.data.frame
## [37] row.names<-.data.frame   rowsum.data.frame
## [39] split.data.frame         split<-.data.frame
## [41] stack.data.frame*        str.data.frame*
## [43] subset.data.frame        summary.data.frame
## [45] t.data.frame             tail.data.frame*
## [47] transform.data.frame     unique.data.frame
## [49] unstack.data.frame*      within.data.frame
##
##    Non-visible functions are asterisked
```

```
attributes(mydata)$names
```

```
##  [1] "carat"   "cut"     "color"   "clarity" "depth"   "table"   "price"
##  [8] "x"       "y"       "z"
```

```
#print the number of columns
ncol(mydata)
```

```
## [1] 10
```

2.Write code to determine how many rows are in a dataframe

```r
#print the number of rows
nrow(mydata)
```

```
## [1] 53940
```

3.Write code to extract the column names from the dataframe and print the names of the columns(one per line) to the console

```r
#because it asks the names should be displayed one per line
#I use "\n" to seperate each names.
cat(names(mydata), sep = "\n")
```

```
## carat
## cut
## color
## clarity
## depth
## table
## price
## x
## y
## z
```

4.Write code to determine the type of each column (numeric, factor, logical, etc.). Print the type of each column to the console

```r
#use function lapply to get the type of each column and print them out
(types = lapply(mydata, class))
```

```
## $carat
## [1] "numeric"
##
## $cut
## [1] "ordered" "factor"
##
## $color
## [1] "ordered" "factor"
##
## $clarity
## [1] "ordered" "factor"
##
## $depth
## [1] "numeric"
##
## $table
## [1] "numeric"
##
## $price
## [1] "integer"
##
## $x
```

```
## [1] "numeric"
##
## $y
## [1] "numeric"
##
## $z
## [1] "numeric"
```

5.Write code that will loop through any dataframe and calculate the mean of every numeric column. Label the output with the name of the column.

```
#Method 1
colmean <- function(data.frame){
  #define a function that accepts dataframe as parameter
  #define a term indexnum to store the index of the columns that are
  #numeric
  indexnum = which(types == "numeric")
  #use function colMean to get the mean of every numeric column
  #and also label the output with the name of the column
  colMeans(data.frame[indexnum])
}
#check
colmean(mydata)
```

```
##      carat      depth      table          x          y          z
##  0.7979397 61.7494049 57.4571839  5.7311572  5.7345260  3.5387338
```

```
#Method 2
#apply sapply and colMeans
#is.numeric here indicates the type of columns we want
colMeans(mydata[sapply(mydata,is.numeric)])
```

```
##        carat        depth        table        price            x
##    0.7979397   61.7494049   57.4571839 3932.7997219    5.7311572
##            y            z
##    5.7345260    3.5387338
```

```
#the difference between these two methods is that the first method
#would not include the result of price because the type of price was
#marked as "integer".
#while the Method 2 would recognize the column price as
#numeric column and then return the mean of the column price.
```

6.Write code that will loop through any dataframe and create a frequency table for every factor column. Label the output with the name of the column.

```
#use sapply function here
#is.factor indicates the type of columns we want
#summary produces the frequency table
summary(mydata[sapply(mydata,is.factor)])
```

```
##         cut        color        clarity
##  Fair     : 1610   D: 6775   SI1    :13065
##  Good     : 4906   E: 9797   VS2    :12258
##  Very Good:12082   F: 9542   SI2    : 9194
##  Premium  :13791   G:11292   VS1    : 8171
##  Ideal    :21551   H: 8304   VVS2   : 5066
##                    I: 5422   VVS1   : 3655
##                    J: 2808   (Other): 2531
```

7.Write code that will loop through any dataframe and determine the number of rows containing NA (missing value) in each column and the percentage of rows containing an NA in any of the columns. HINT: In a single row, zero or more columns may contain an NA. For the percentage of rows containing NA in any column, do not double count NA in rows that contain more than one column with an NA. Print the results to the console.

```
#use apply and sapply function here
#use is.na to check if there exists NAs in columns
#2 indicates columns
#use sum to get the number of rows containing NA in each column.
apply(sapply(mydata,is.na),2,sum)
```

```
##   carat     cut   color clarity   depth   table   price       x       y
##       0       0       0       0       0       0       0       0       0
##       z
##       0
```

```
#use rowSums here to get the if there exists NAs.
#the result consists of T and F. T means the row has NA; F means not. #divided by the number of rows to
sum(rowSums(is.na(mydata))>0)/nrow(mydata)
```

```
## [1] 0
```

8.Create an R function that can accept any dataframe as a parameter and returns a dataframe that contains each pair of column names in the first column in a single string separated by a -, e.g. for the variables x and y, you should form the string ???x-y??? (HINT: Look at the help provided for the paste function) and their corresponding Pearson correlation coefficient in the second column. (HINT: There is a function that calculates correlation coefficients ??? look carefully at what is returned and optimize how you extract the correlation coefficients). Do not repeat any pairs.

```
pairname = NULL
numcor = NULL
#define two terms for futher use

paircor = function(dataframe){
  #define a function that accepts dataframe as parameter
  #get the numeric date subsets
  num = dataframe[sapply(dataframe,is.numeric)]
  #get the column names of the subset above
  numname = colnames(num)
  #use two for loops here to pair each column
  for (i in 1:(length(numname)-1)){
    for (j in (i+1):length(numname)){
      #use paste function to connect each two column names with "-"
```

```
      pairname = c(pairname, paste(numname[i], numname[j], sep = "-"))
      #calculate the correlation of each pair of the subset num

      numcor = c(numcor, cor(num[i], num[j], method = "pearson"))

    }
  }
  #return a dataframe that consists of the names of each pair
  #and their correlation coefficients

  return(data.frame(pairname, numcor))
}

#check
paircor(mydata)
```

```
##       pairname      numcor
## 1  carat-depth  0.02822431
## 2  carat-table  0.18161755
## 3  carat-price  0.92159130
## 4      carat-x  0.97509423
## 5      carat-y  0.95172220
## 6      carat-z  0.95338738
## 7  depth-table -0.29577852
## 8  depth-price -0.01064740
## 9      depth-x -0.02528925
## 10     depth-y -0.02934067
## 11     depth-z  0.09492388
## 12 table-price  0.12713390
## 13     table-x  0.19534428
## 14     table-y  0.18376015
## 15     table-z  0.15092869
## 16     price-x  0.88443516
## 17     price-y  0.86542090
## 18     price-z  0.86124944
## 19         x-y  0.97470148
## 20         x-z  0.97077180
## 21         y-z  0.95200572
```