# 1.3 Meet R - R Practice (Answers)

*Ryan Safner*

*ECON 480 - Fall 2019*

## Getting Set Up

Before we begin, start a new file with `File` → `New File` → `R Script`. As you work through this sheet in the console in `R`, also add (copy/paste) your commands that work into this new file. At the end, save it, and run to execute all of your commands at once.

## Creating Objects

### 1. Work on the following parts:

a. Create a vector called `me` with two objects, your first name, and your last name.

```
me <- c("Ryan", "Safner")
```

b. Call the vector to inspect it.

```
me
```

```
## [1] "Ryan"   "Safner"
```

c. Confirm it is a character class vector.

```
class(me)
```

```
## [1] "character"
```

### 2. Use R's help functions to determine what the `paste()` function does. Then paste together your first name and last name.

```
?paste() # or help(paste)
 # paste is a function that combines (concatenates) multiple string objects into a single object
paste("Ryan", "Safner")
```

```
## [1] "Ryan Safner"
```

```
# note you can choose how to separate string objects with the "sep" argument
# for example
paste("Ryan", "Safner", sep="") # no separation
```

```
## [1] "RyanSafner"
```

```
paste("Ryan", "Safner", sep=" ") # separate with a space " " (the default)
```

```
## [1] "Ryan Safner"
```

```
paste("Ryan", "Safner", sep="_") # separate with underscore
```

```
## [1] "Ryan_Safner"
```

### 3. Create a vector called `my_vector` with all the even integers from 2 to 10.

```
my_vector <- c(2,4,6,8,10)

# verify it worked
my_vector
```

```
## [1]  2  4  6  8 10
```

```
# alternatively, you can use the sequence function, seq()
# see the Class page for more about this function
my_vector <- seq(from = 2, # starting integer
                 to = 10, # ending integer
                 by = 2) # by 2's

# you can shorten it by not including the names of the arguments:
my_vector <- seq(2,10,2)

# verify it worked
my_vector
```

```
## [1]  2  4  6  8 10
```

### 4. Find the mean of `my_vector` with `mean()`.

```
mean(my_vector)
```

```
## [1] 6
```

### 5. Take all the integers from 18 to 763,[1] then get the mean.

```
# create a sequence of integers by 1 with starting_number:ending_number
# see Class 3 page for more

# you can do this all at once without making an object
mean(18:763)
```

```
## [1] 390.5
```

```
# alternatively you can save this as a vector and run the mean on it
vec1 <- 18:763

mean(vec1)
```

```
## [1] 390.5
```

[1] Hint: use the : operator to create a sequence from a starting number to an ending number

## Playing with Data

For the following questions, we will use the `diamonds` dataset, included as part of `ggplot2`.

**6. Install `ggplot2`.**

```
install.packages("ggplot2") # note the s and the quotes
```

**7. Load `ggplot2` with the `library()` command.**

```
library("ggplot2") # quotes not necessary, but can be used
```

**8. Get the `structure` of the diamonds data frame. What are the different variables and what kind of data does each contain?**

```
str(diamonds)
```

```
## 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 ...
```

```
# We have
# - carat: a number
# - cut: an ordered factor
# - color: an ordered factor
# - clarity: an ordered factor
# - depth: a number
# - table: a number
# - price: an integer
# - x: a number
# - y: a number
# - z: a number
```

**9. Get summary statistics separately for `carat`, `depth`, `table`, and `price`.**

```
summary(diamonds$carat)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.2000  0.4000  0.7000  0.7979  1.0400  5.0100
```

```
summary(diamonds$depth)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   43.00   61.00   61.80   61.75   62.50   79.00
```

```
summary(diamonds$table)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   43.00   56.00   57.00   57.46   59.00   95.00
```

```
summary(diamonds$price)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     326     950    2401    3933    5324   18823
```

**10. `color`, `cut`, and `clarity` are categorical variables (factors). Use the `table()` command to generate frequency tables for each.**

```
table(diamonds$color)
```

```
##
##     D     E     F     G     H     I     J
##  6775  9797  9542 11292  8304  5422  2808
```

```
table(diamonds$cut)
```

```
##
##      Fair      Good Very Good   Premium     Ideal
##      1610      4906     12082     13791     21551
```

```
table(diamonds$clarity)
```

```
##
##    I1   SI2   SI1   VS2   VS1  VVS2  VVS1    IF
##   741  9194 13065 12258  8171  5066  3655  1790
```

**11. Now rerun the `summary()` command on the entire data frame.**

```
summary(diamonds)
```

```
##      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
##
```

**12. Now look only at (subset) the first 4 diamonds in the dataset.**

```
# remember, dataframes are indexed by: df[row#s, column#s]
diamonds[1:4,] # select first through fourth rows, all columns
```

```
## # A tibble: 4 x 10
##   carat cut     color clarity depth table price     x     y     z
##   <dbl> <ord>   <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1 0.23  Ideal   E     SI2      61.5    55   326  3.95  3.98  2.43
## 2 0.21  Premium E     SI1      59.8    61   326  3.89  3.84  2.31
## 3 0.23  Good    E     VS1      56.9    65   327  4.05  4.07  2.31
## 4 0.290 Premium I     VS2      62.4    58   334  4.2   4.23  2.63
```

```
# alternatively
diamonds[c(1,2,3,4),] # using a vector-approach
```

```
## # A tibble: 4 x 10
##   carat cut     color clarity depth table price     x     y     z
##   <dbl> <ord>   <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1 0.23  Ideal   E     SI2      61.5    55   326  3.95  3.98  2.43
## 2 0.21  Premium E     SI1      59.8    61   326  3.89  3.84  2.31
## 3 0.23  Good    E     VS1      56.9    65   327  4.05  4.07  2.31
## 4 0.290 Premium I     VS2      62.4    58   334  4.2   4.23  2.63
```

**13. Now look only at (subset) the third and seventh diamond in the dataset.**

```
diamonds[c(3,7),] # select 3rd and 7th row, all columns
```

```
## # A tibble: 2 x 10
##   carat cut       color clarity depth table price     x     y     z
##   <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31
## 2  0.24 Very Good I     VVS1     62.3    57   336  3.95  3.98  2.47
```

**14. Now look only at (subset) the second column of the dataset.**

```
diamonds[,2] # select all rows, 2nd column
```

```
## # A tibble: 53,940 x 1
##    cut
##    <ord>
##  1 Ideal
##  2 Premium
##  3 Good
```

```
##  4 Premium
##  5 Good
##  6 Very Good
##  7 Very Good
##  8 Very Good
##  9 Fair
## 10 Very Good
## # ... with 53,930 more rows
```

**15. Do this again, but look using the $ to pull up the second column by name.**

```
# second column is called "cut"
diamonds$cut # dont' run this, it'll print 53,000 rows!
```

**16. Now look only at diamonds that have a carat greater than or equal to 1.**

```
# use the [square brackets] to subset,
# first argument (rows) are chosen by conditional:
# - choose diamonds based on their carat, and only carats >= 1
diamonds[diamonds$carat >= 1,] # select rows on condition, and all columns
```

```
## # A tibble: 19,060 x 10
##    carat cut       color clarity depth table price     x     y     z
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  1.17 Very Good J     I1       60.2    61  2774  6.83  6.9   4.13
##  2  1.01 Premium   F     I1       61.8    60  2781  6.39  6.36  3.94
##  3  1.01 Fair      E     I1       64.5    58  2788  6.29  6.21  4.03
##  4  1.01 Premium   H     SI2      62.7    59  2788  6.31  6.22  3.93
##  5  1.05 Very Good J     SI2      63.2    56  2789  6.49  6.45  4.09
##  6  1.05 Fair      J     SI2      65.8    59  2789  6.41  6.27  4.18
##  7  1    Premium   I     SI2      58.2    60  2795  6.61  6.55  3.83
##  8  1.01 Fair      E     SI2      67.4    60  2797  6.19  6.05  4.13
##  9  1.04 Premium   G     I1       62.2    58  2801  6.46  6.41  4
## 10  1    Premium   J     SI2      62.3    58  2801  6.45  6.34  3.98
## # ... with 19,050 more rows
```

**17. Now look only at diamonds that have a VVS1 clarity.**

```
# we are testing for equality, so we need two ==
# we are selecting based on clarity, a character/factor, so we need quotes
diamonds[diamonds$clarity=="VVS1",] # select rows on condition, and all columns
```

```
## # A tibble: 3,655 x 10
##    carat cut       color clarity depth table price     x     y     z
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  0.24 Very Good I     VVS1     62.3  57     336  3.95  3.98  2.47
##  2  0.32 Ideal     I     VVS1     62    55.3   553  4.39  4.42  2.73
##  3  0.24 Premium   E     VVS1     60.7  58     553  4.01  4.03  2.44
##  4  0.24 Very Good D     VVS1     61.5  60     553  3.97  4     2.45
##  5  0.26 Very Good E     VVS1     62.6  59     554  4.06  4.09  2.55
##  6  0.26 Very Good E     VVS1     63.4  59     554  4     4.04  2.55
```

```
##  7  0.26 Very Good D     VVS1     62.1  60     554  4.03  4.12  2.53
##  8  0.26 Good      E     VVS1     57.9  60     554  4.22  4.25  2.45
##  9  0.24 Premium   G     VVS1     62.3  59     554  3.95  3.92  2.45
## 10  0.24 Premium   H     VVS1     61.2  58     554  4.01  3.96  2.44
## # ... with 3,645 more rows
```

**18. Now look only at dimaonds that have a color of E, F, I, and J.**

```
# same idea as last problem, except now we want one of any of these 4 colors

# first (tedious) way, a series of checking equality and using "OR"s (|)
diamonds[diamonds$color=="E" | diamonds$color=="F" | diamonds$color=="I" | diamonds$color=="J",] # sele
```

```
## # A tibble: 27,569 x 10
##    carat cut       color clarity depth table price     x     y     z
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1 0.23  Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
##  2 0.21  Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
##  3 0.23  Good      E     VS1      56.9    65   327  4.05  4.07  2.31
##  4 0.290 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
##  5 0.31  Good      J     SI2      63.3    58   335  4.34  4.35  2.75
##  6 0.24  Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
##  7 0.24  Very Good I     VVS1     62.3    57   336  3.95  3.98  2.47
##  8 0.22  Fair      E     VS2      65.1    61   337  3.87  3.78  2.49
##  9 0.3   Good      J     SI1      64      55   339  4.25  4.28  2.73
## 10 0.23  Ideal     J     VS1      62.8    56   340  3.93  3.9   2.46
## # ... with 27,559 more rows
```

```
# second (better) way, using group membership operator (%in%) and list the elements as a vector
diamonds[diamonds$color %in% c("E","F","I","J"),] # select rows on condition, and all columns
```

```
## # A tibble: 27,569 x 10
##    carat cut       color clarity depth table price     x     y     z
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1 0.23  Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
##  2 0.21  Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
##  3 0.23  Good      E     VS1      56.9    65   327  4.05  4.07  2.31
##  4 0.290 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
##  5 0.31  Good      J     SI2      63.3    58   335  4.34  4.35  2.75
##  6 0.24  Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
##  7 0.24  Very Good I     VVS1     62.3    57   336  3.95  3.98  2.47
##  8 0.22  Fair      E     VS2      65.1    61   337  3.87  3.78  2.49
##  9 0.3   Good      J     SI1      64      55   339  4.25  4.28  2.73
## 10 0.23  Ideal     J     VS1      62.8    56   340  3.93  3.9   2.46
## # ... with 27,559 more rows
```

**19. Now look only at diamonds that have a `carat` greater than or equal to 1 and a VVS1 `clarity`.**

```
# testing for two conditions (AND)
diamonds[diamonds$carat>=1 & diamonds$clarity=="VVS1",] # select rows on condition, and all columns
```

```
## # A tibble: 435 x 10
```

```
##    carat cut       color clarity depth table price     x     y     z
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  1    Good      I     VVS1     56.5    62  4445  6.58  6.55  3.71
##  2  1    Good      J     VVS1     63.5    59  4633  6.29  6.34  4.01
##  3  1    Very Good J     VVS1     63.5    59  4717  6.34  6.29  4.01
##  4  1.01 Premium   H     VVS1     60.5    57  4955  6.55  6.48  3.94
##  5  1.01 Premium   I     VVS1     62      59  4989  6.39  6.32  3.94
##  6  1.04 Premium   H     VVS1     60.4    58  5102  6.58  6.53  3.96
##  7  1.01 Ideal     I     VVS1     61.7    56  5478  6.42  6.47  3.98
##  8  1.09 Very Good J     VVS1     63.9    56  5588  6.47  6.51  4.15
##  9  1.27 Premium   J     VVS1     60.1    58  5761  7.06  6.99  4.22
## 10  1.21 Premium   J     VVS1     61.3    59  5893  6.81  6.86  4.19
## # ... with 425 more rows
```

**20. Get the average price of diamonds in question 18.**[2]

```
# use command from last question as the argument to the mean function,
## but be sure that you look at the price, specifically

mean(diamonds$price[diamonds$carat>=1 & diamonds$color=="D" & diamonds$clarity=="VVS1"])
```

```
## [1] 13935.48
```

**21. What is the highest price for a diamond with a 1.0 carat, D color, and VVS1 clarity?**

```
max(diamonds$price[diamonds$carat>=1 & diamonds$color=="D" & diamonds$clarity=="VVS1"])
```

```
## [1] 17932
```

# Execute your R Script

Save the R Script you created at the beginning and (hopefully) have been pasting all of your valid commands to. This creates a *.R* file wherever you choose to save it to. Now looking at the file in the upper left pane of *R Studio* look for the button in the upper right corner that says **Run**. Sit back and watch `R` redo everything you've carefully worked on, all at once.

Your `.R` file should look something like this:

```
# 1 --------------------

## a
me <- c("Ryan", "Safner")

## b
me

## c
class(me)
```

[2]Hints: use your subset command as an argument to the mean function. You will not need a comma here because you are looking for a single row.

```r
# 2 ----------------

?paste() # or help(paste)
 # paste is a function that combines (concatenates) multiple string objects into a single object
paste("Ryan", "Safner")

# note you can choose how to separate string objects with the "sep" argument
# for example
paste("Ryan", "Safner", sep="") # no separation
paste("Ryan", "Safner", sep=" ") # separate with a space " " (the default)
paste("Ryan", "Safner", sep="_") # separate with underscore

# 3 ---------------

my_vector <- c(2,4,6,8,10)

# verify it worked
my_vector
# alternatively, you can use the sequence function, seq()
# see the Class page for more about this function
my_vector <- seq(from = 2, # starting integer
                 to = 10, # ending integer
                 by = 2) # by 2's

# you can shorten it by not including the names of the arguments:
my_vector <- seq(2,10,2)

# verify it worked
my_vector

# 4 -------------------

mean(my_vector)


# 5 -------------------

# create a sequence of integers by 1 with starting_number:ending_number
# see Class 3 page for more

# you can do this all at once without making an object
mean(18:763)

# alternatively you can save this as a vector and run the mean on it
vec1 <- 18:763

mean(vec1)

# 6 ------------------

install.packages("ggplot2") # note the s and the quotes

# 7 ------------------
```

```r
library("ggplot2") # quotes not necessary, but can be used

# 8 ------------------

str(diamonds)

# We have
# - carat: a number
# - cut: an ordered factor
# - color: an ordered factor
# - clarity: an ordered factor
# - depth: a number
# - table: a number
# - price: an integer
# - x: a number
# - y: a number
# - z: a number

# 9 ------------------

summary(diamonds$carat)
summary(diamonds$depth)
summary(diamonds$table)
summary(diamonds$price)

# 10 ------------------

table(diamonds$color)
table(diamonds$cut)
table(diamonds$clarity)

# 11 ------------------

summary(diamonds)

# 12 ------------------

# remember, dataframes are indexed by: df[row#s, column#s]
diamonds[1:4,] # select first through fourth rows, all columns

# alternatively
diamonds[c(1,2,3,4),] # using a vector-approach

# 13 ------------------

diamonds[c(3,7),] # select 3rd and 7th row, all columns

# 14 ------------------
diamonds[,2] # select all rows, 2nd column

# 15 ------------------

# second column is called "cut"
```

```r
# diamonds$cut dont' run this, it'll print 53,000 rows!

# 16 -------------------

# use the [square brackets] to subset,
# first argument (rows) are chosen by conditional:
# - choose diamonds based on their carat, and only carats >= 1
diamonds[diamonds$carat >= 1,] # select rows on condition, and all columns

# 17 -------------------

# we are testing for equality, so we need two ==
# we are selecting based on clarity, a character/factor, so we need quotes
diamonds[diamonds$clarity=="VVS1",] # select rows on condition, and all columns

# 18 -------------------

# same idea as last problem, except now we want one of any of these 4 colors

# first (tedious) way, a series of checking equality and using "OR"s (|)
diamonds[diamonds$color=="E" | diamonds$color=="F" | diamonds$color=="I" | diamonds$color=="J",] # sele

# second (better) way, using group membership operator (%in%) and list the elements as a vector
diamonds[diamonds$color %in% c("E","F","I","J"),] # select rows on condition, and all columns

# 19 -------------------

# testing for two conditions (AND)
diamonds[diamonds$carat>=1 & diamonds$clarity=="VVS1",] # select rows on condition, and all columns

# 20 -------------------

# use command from last question as the argument to the mean function
## but be sure that you look at the price, specifically
mean(diamonds$price[diamonds$carat>=1 & diamonds$color=="D" & diamonds$clarity=="VVS1"])

# 21 -------------------

max(diamonds$price[diamonds$carat>=1 & diamonds$color=="D" & diamonds$clarity=="VVS1"])
```