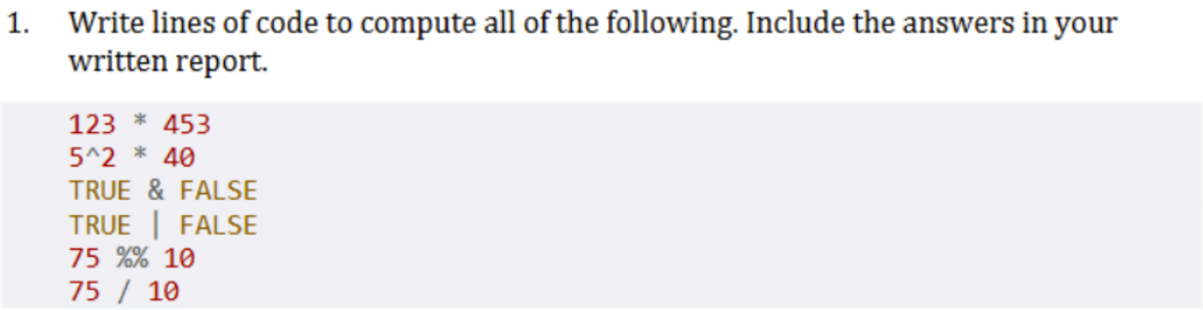
Project 1 - R Practice Report

Elenchezhian_Project1.R
Name : Sri Ram Prabu
Date: [01/17/2025]
Class: ALY_6000

# Problem 1



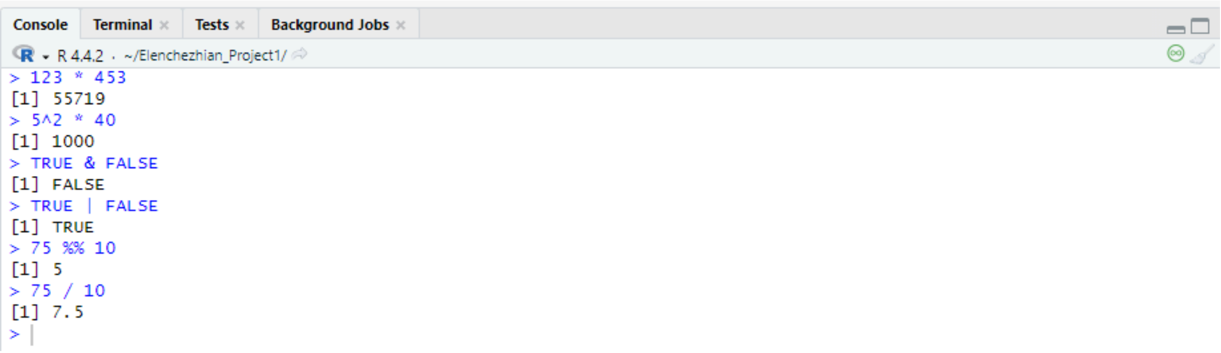
1. Write lines of code to compute all of the following. Include the answers in your written report.

```
123 * 453
5^2 * 40
TRUE & FALSE
TRUE | FALSE
75 %% 10
75 / 10
```

Rcode :

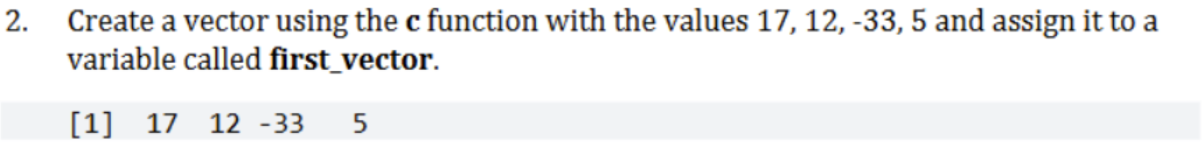
```
123 * 453
5^2 * 40
TRUE & FALSE
TRUE | FALSE
75 %% 10
75 / 10
```

Output:



```
> 123 * 453
[1] 55719
> 5^2 * 40
[1] 1000
> TRUE & FALSE
[1] FALSE
> TRUE | FALSE
[1] TRUE
> 75 %% 10
[1] 5
> 75 / 10
[1] 7.5
> 
```

# Problem 2



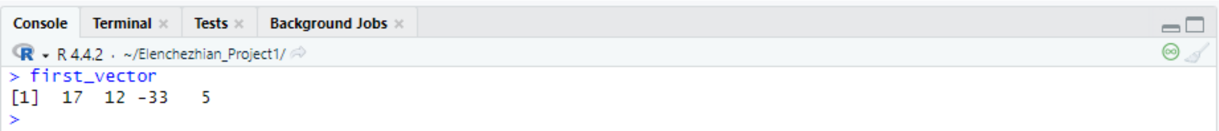
2. Create a vector using the **c** function with the values 17, 12, -33, 5 and assign it to a variable called **first_vector**.

```
[1]  17  12 -33   5
```

Rcode :

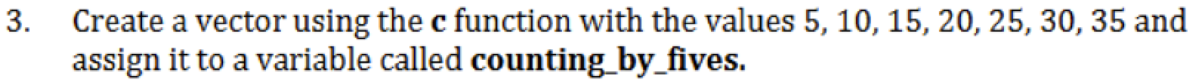
first_vector <- c(17, 12, -33, 5)

Output:



```
> first_vector
[1]  17  12 -33   5
>
```

# Problem 3

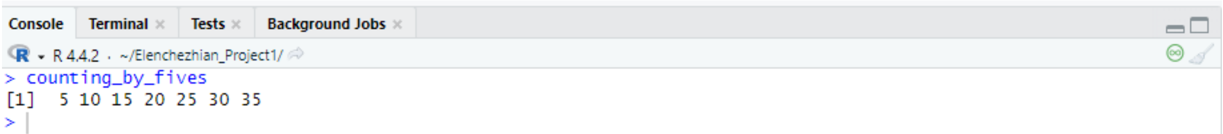


3. Create a vector using the **c** function with the values 5, 10, 15, 20, 25, 30, 35 and assign it to a variable called **counting_by_fives.**

Rcode :

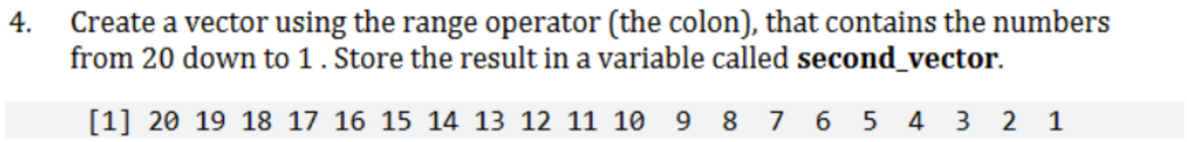
counting_by_fives <- c(5, 10, 15, 20, 25, 30, 35)

Output:



```
> counting_by_fives
[1]  5 10 15 20 25 30 35
> |
```

# Problem 4



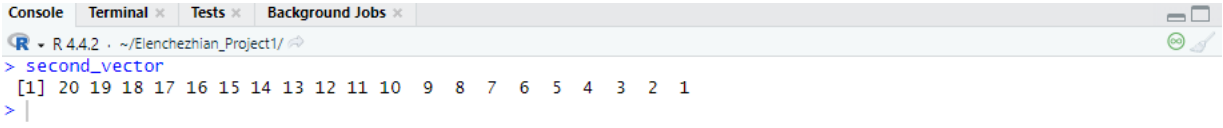
4. Create a vector using the range operator (the colon), that contains the numbers from 20 down to 1 . Store the result in a variable called **second_vector**.

```
[1] 20 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
```

Rcode :

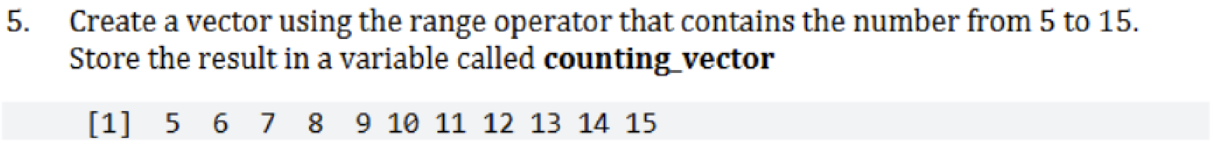
second_vector <- 20:1

Output:



```
> second_vector
 [1] 20 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
> 
```

# Problem 5



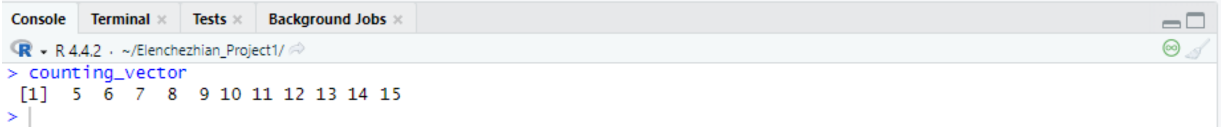
5. Create a vector using the range operator that contains the number from 5 to 15. Store the result in a variable called **counting_vector**

```
[1]  5  6  7  8  9 10 11 12 13 14 15
```

Rcode :

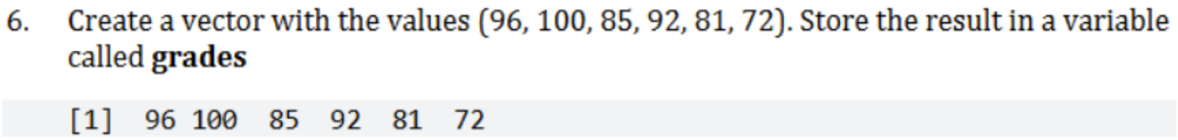
counting_vector <- 5:15

Output:



```
> counting_vector
 [1]  5  6  7  8  9 10 11 12 13 14 15
> 
```

# Problem 6



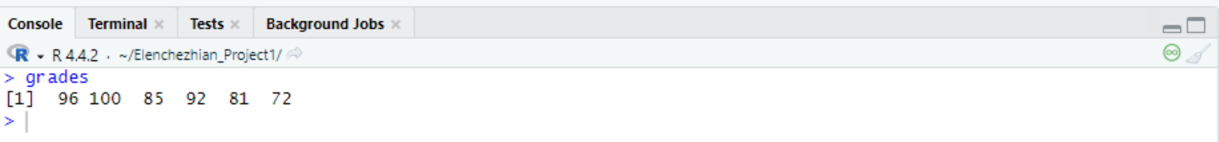
6. Create a vector with the values (96, 100, 85, 92, 81, 72). Store the result in a variable called **grades**

```
[1]  96 100  85  92  81  72
```

Rcode :

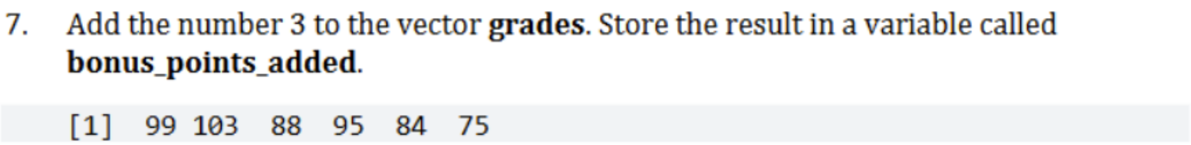
grades <- c(96, 100, 85, 92, 81, 72)

Output:



```
> grades
[1]  96 100  85  92  81  72
> 
```

# Problem 7



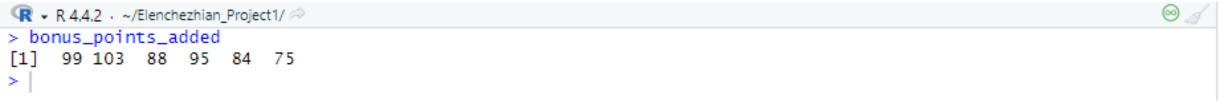
7. Add the number 3 to the vector **grades**. Store the result in a variable called **bonus_points_added**.

```
[1]  99 103  88  95  84  75
```

Rcode :

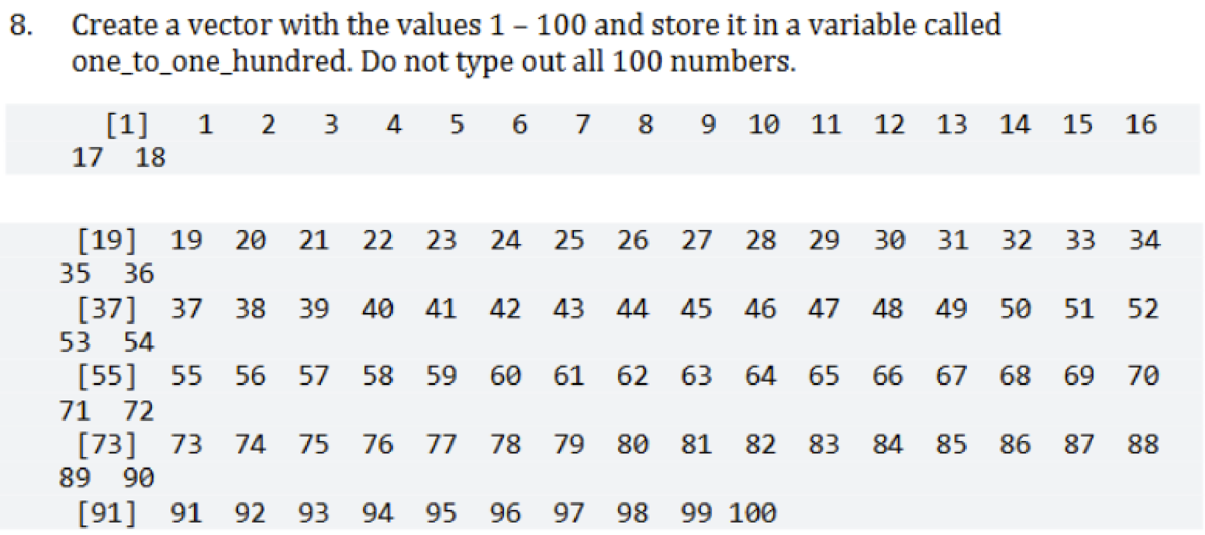
bonus_points_added <- grades + 3

Output:



```
> bonus_points_added
[1]  99 103  88  95  84  75
> 
```

# Problem 8



8. Create a vector with the values 1 – 100 and store it in a variable called one_to_one_hundred. Do not type out all 100 numbers.

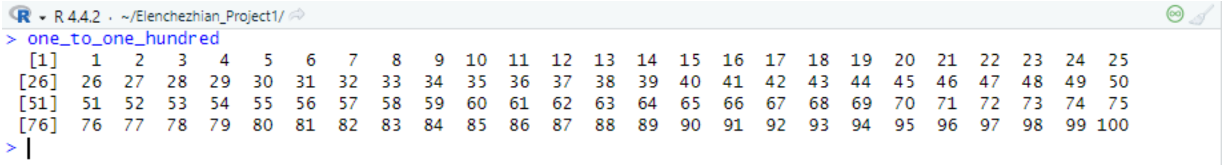
```
   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16
 17  18

  [19]  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34
 35  36
  [37]  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52
 53  54
  [55]  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70
 71  72
  [73]  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88
 89  90
  [91]  91  92  93  94  95  96  97  98  99 100
```

Rcode :

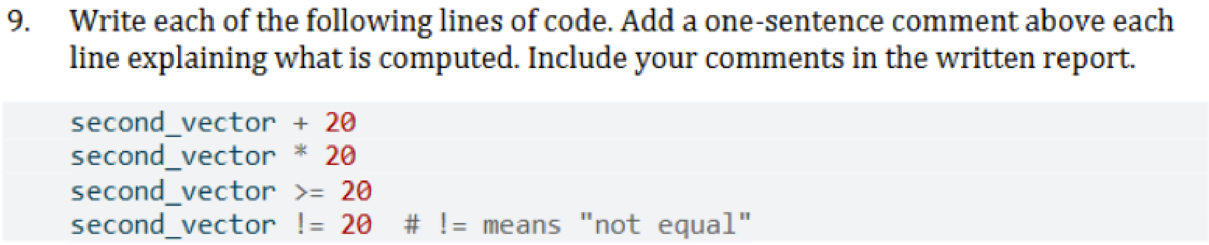
one_to_one_hundred <- 1:100

Output:



```
> one_to_one_hundred
  [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25
 [26]  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50
 [51]  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75
 [76]  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100
> |
```

# Problem 9



9. Write each of the following lines of code. Add a one-sentence comment above each line explaining what is computed. Include your comments in the written report.

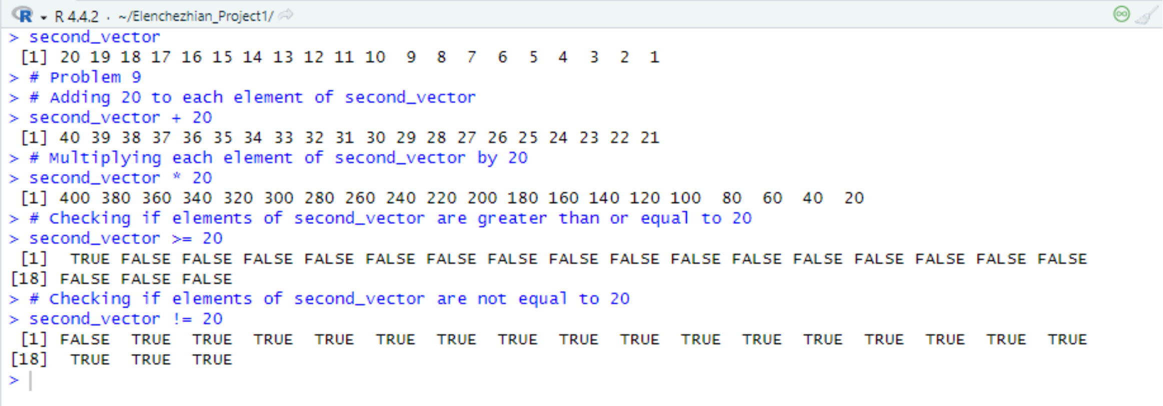
```
second_vector + 20
second_vector * 20
second_vector >= 20
second_vector != 20  # != means "not equal"
```

Rcode :

# Adding 20 to each element of second_vector
second_vector + 20
# Multiplying each element of second_vector by 20
second_vector * 20

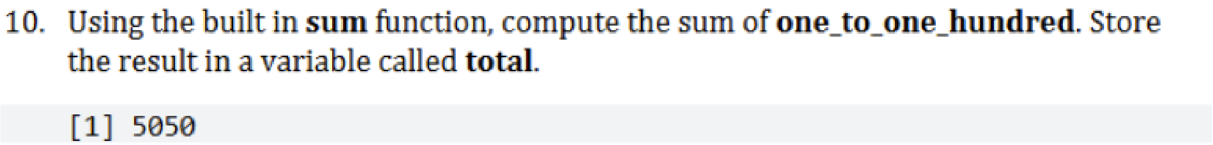
Checking if elements of second_vector are greater than or equal to 20
second_vector >= 20
Checking if elements of second_vector are not equal to 20
second_vector != 20

Output:



```
> second_vector
 [1] 20 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
> # Problem 9
> # Adding 20 to each element of second_vector
> second_vector + 20
 [1] 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21
> # Multiplying each element of second_vector by 20
> second_vector * 20
 [1] 400 380 360 340 320 300 280 260 240 220 200 180 160 140 120 100  80  60  40  20
> # Checking if elements of second_vector are greater than or equal to 20
> second_vector >= 20
 [1]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[18] FALSE FALSE FALSE
> # Checking if elements of second_vector are not equal to 20
> second_vector != 20
 [1] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[18]  TRUE  TRUE  TRUE
> |
```

Problem 10



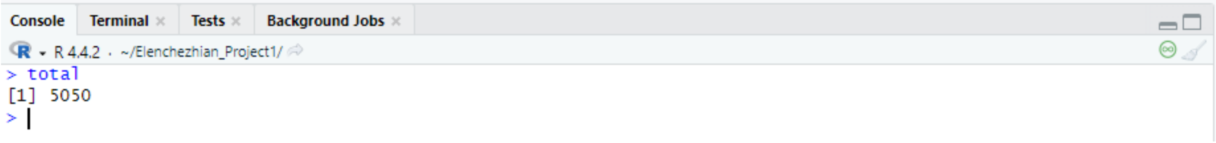
10. Using the built in **sum** function, compute the sum of **one_to_one_hundred**. Store the result in a variable called **total**.

```
[1] 5050
```

Rcode :

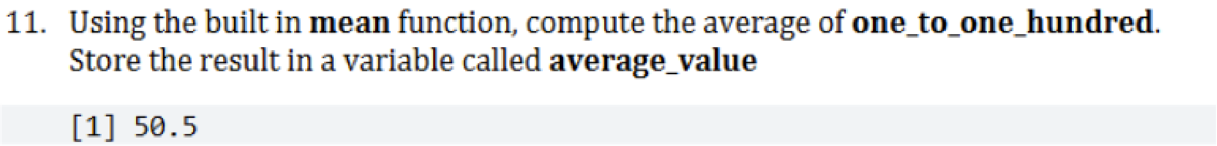
total <- sum(one_to_one_hundred)

Output:



```
> total
[1] 5050
> |
```

Problem 11



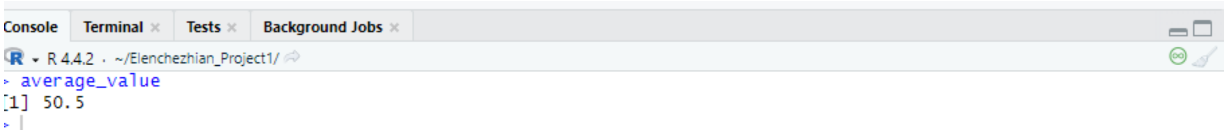
11. Using the built in **mean** function, compute the average of **one_to_one_hundred**. Store the result in a variable called **average_value**

```
[1] 50.5
```

Rcode :

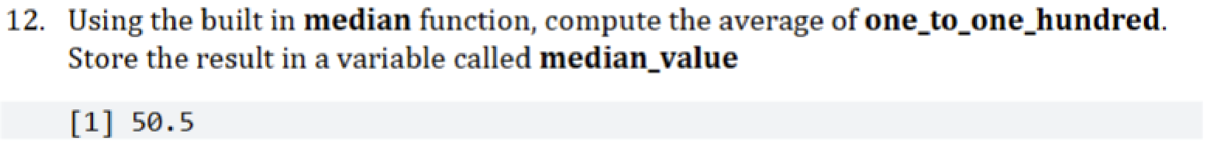
average_value <- mean(one_to_one_hundred)

Output:



```
> average_value
[1] 50.5
>
```

# Problem 12



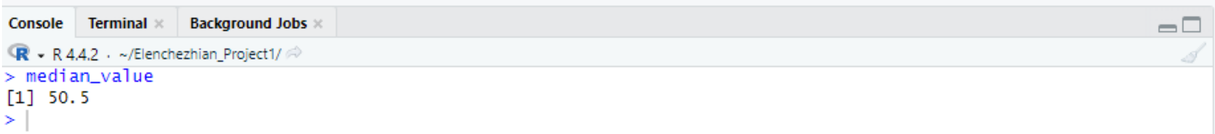
12. Using the built in **median** function, compute the average of **one_to_one_hundred**. Store the result in a variable called **median_value**

```
[1] 50.5
```

Rcode :

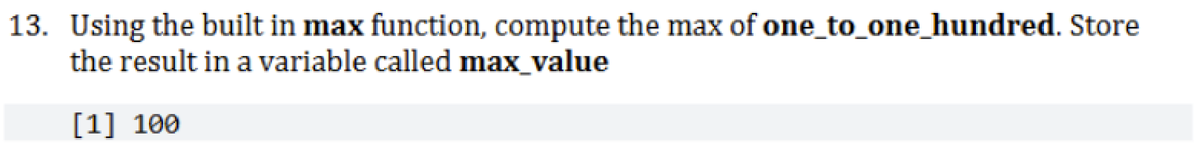
median_value <- median(one_to_one_hundred)

Output:



```
> median_value
[1] 50.5
>
```

# Problem 13



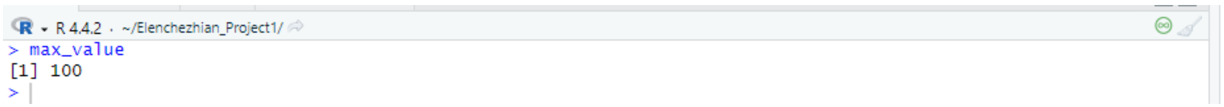
13. Using the built in **max** function, compute the max of **one_to_one_hundred**. Store the result in a variable called **max_value**

```
[1] 100
```

Rcode :

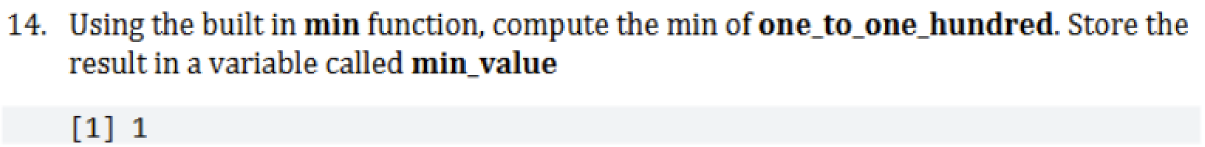
max_value <- max(one_to_one_hundred)

Output:



```
> max_value
[1] 100
> 
```

# Problem 14



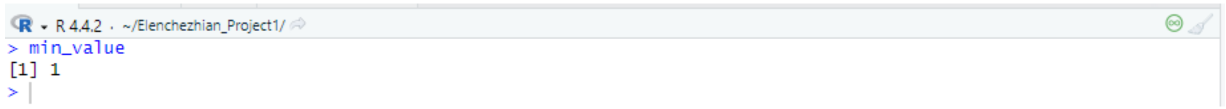
14. Using the built in **min** function, compute the min of **one_to_one_hundred**. Store the result in a variable called **min_value**

```
[1] 1
```

Rcode :

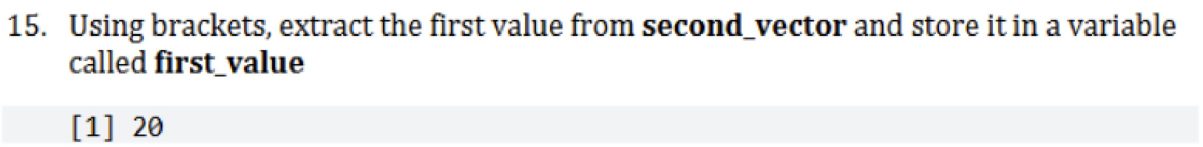
min_value <- min(one_to_one_hundred)

Output:



```
> min_value
[1] 1
> 
```

# Problem 15



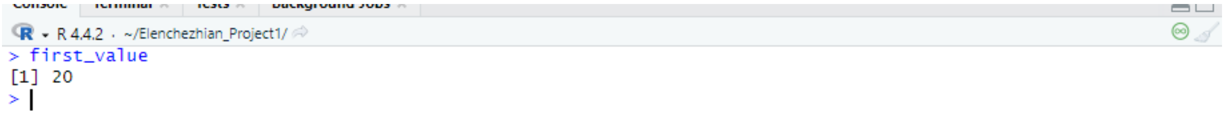
15. Using brackets, extract the first value from **second_vector** and store it in a variable called **first_value**

```
[1] 20
```

Rcode :

first_value <- second_vector[1]

Output:



```
> first_value
[1] 20
> 
```

# Problem 16

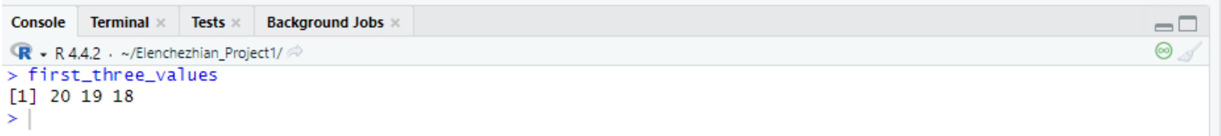
16. Using brackets, extract the first, second and third values from **second_vector**. Store the result in a variable called **first_three_values.**

```
[1] 20 19 18
```

Rcode :

first_three_values <- second_vector[1:3]

Output:



```
> first_three_values
[1] 20 19 18
> 
```

# Problem 17

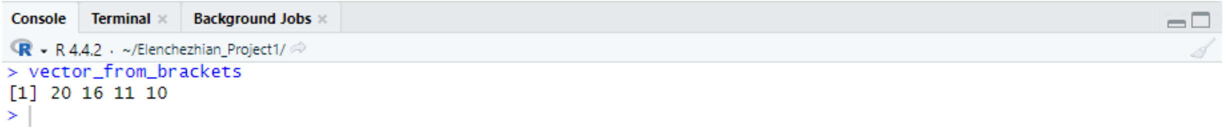
17. Using brackets, extract the 1st, 5th, 10th, and 11th elements of **second_vector**. Store the resulting vector in a variable called **vector_from_brackets**.

```
[1] 20 16 11 10
```

Rcode :

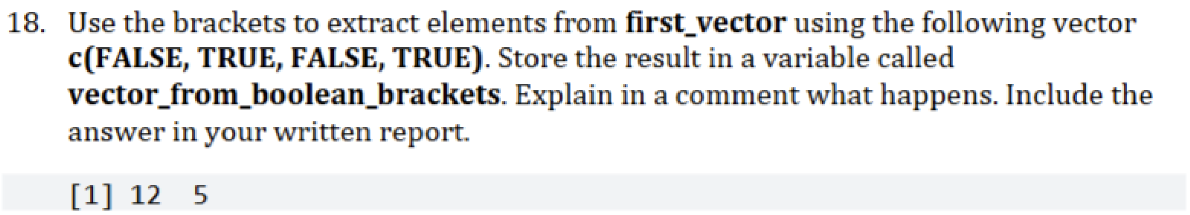
vector_from_brackets <- second_vector[c(1, 5, 10, 11)]

Output:



```
> vector_from_brackets
[1] 20 16 11 10
> 
```

# Problem 18



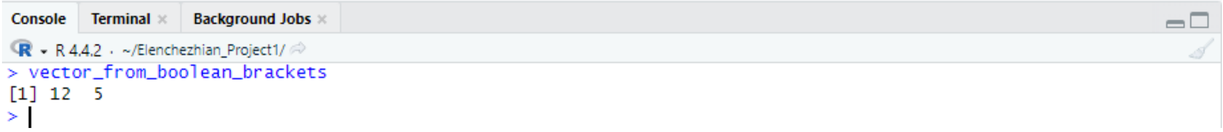
18. Use the brackets to extract elements from **first_vector** using the following vector **c(FALSE, TRUE, FALSE, TRUE)**. Store the result in a variable called **vector_from_boolean_brackets**. Explain in a comment what happens. Include the answer in your written report.

```
[1] 12  5
```

Rcode :

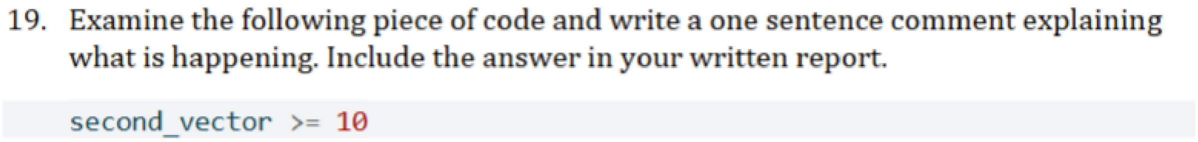
vector_from_boolean_brackets <- first_vector[c(FALSE, TRUE, FALSE, TRUE)]
# Explanation: Extracts elements where the corresponding boolean value is TRUE.

Output:



```
> vector_from_boolean_brackets
[1] 12  5
> 
```

# Problem 19



19. Examine the following piece of code and write a one sentence comment explaining what is happening. Include the answer in your written report.

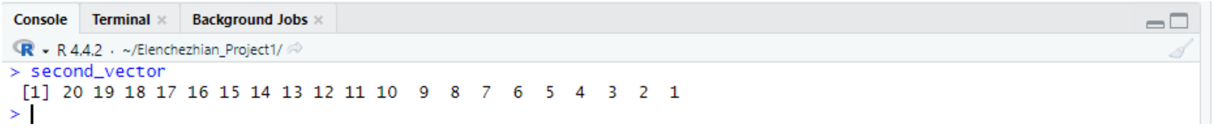
```
second_vector >= 10
```

Rcode :

# Comparing each element of second_vector to see if it is greater than or equal to 10.

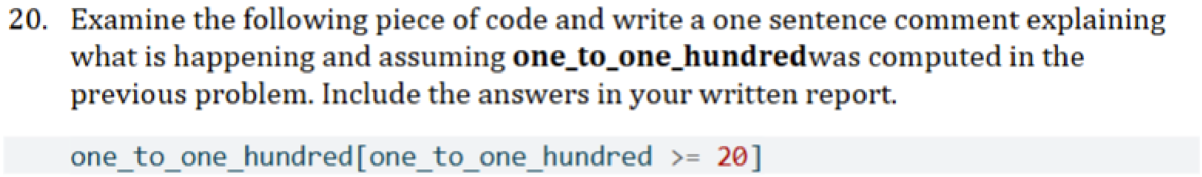
second_vector >= 10

Output:



```
> second_vector
 [1] 20 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
> |
```

Problem 20



20. Examine the following piece of code and write a one sentence comment explaining what is happening and assuming **one_to_one_hundred**was computed in the previous problem. Include the answers in your written report.

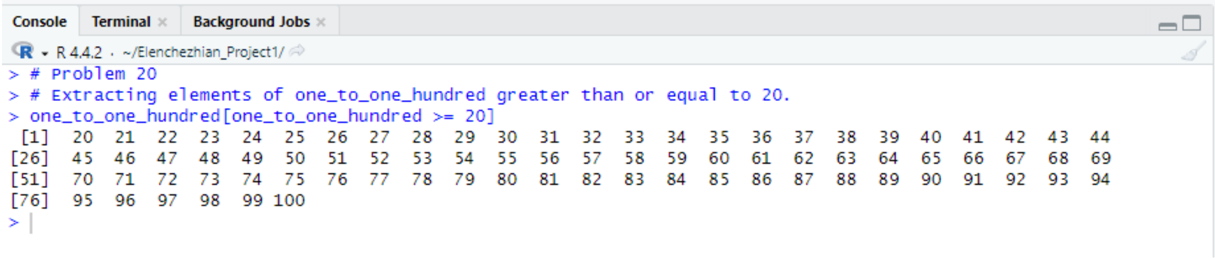
```
one_to_one_hundred[one_to_one_hundred >= 20]
```

Rcode :

Extracting elements of one_to_one_hundred greater than or equal to 20.

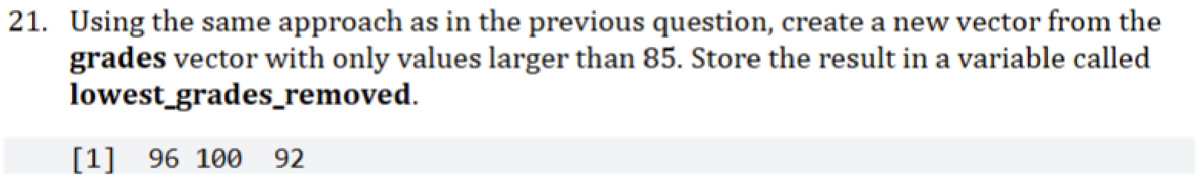
one_to_one_hundred[one_to_one_hundred >= 20]

Output:



```
> # Problem 20
> # Extracting elements of one_to_one_hundred greater than or equal to 20.
> one_to_one_hundred[one_to_one_hundred >= 20]
 [1]  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44
[26]  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69
[51]  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94
[76]  95  96  97  98  99 100
> |
```

# Problem 21



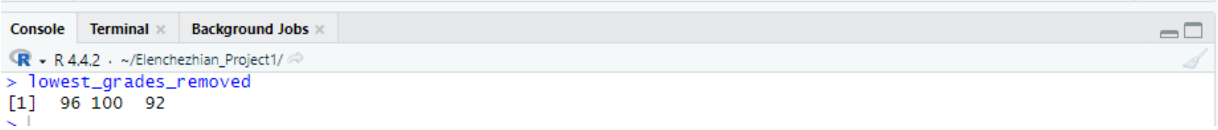
21. Using the same approach as in the previous question, create a new vector from the **grades** vector with only values larger than 85. Store the result in a variable called **lowest_grades_removed**.

```
[1]  96 100  92
```

Rcode :

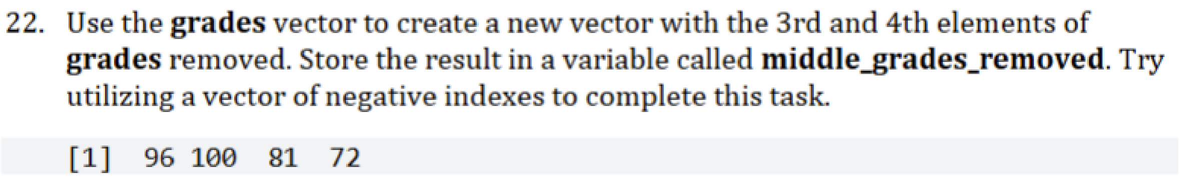
lowest_grades_removed <- grades[grades > 85]

Output:



```
> lowest_grades_removed
[1]  96 100  92
>
```

# Problem 22



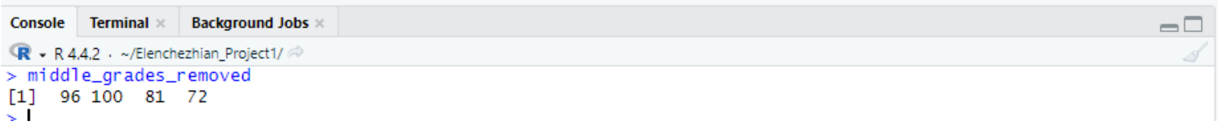
22. Use the **grades** vector to create a new vector with the 3rd and 4th elements of **grades** removed. Store the result in a variable called **middle_grades_removed**. Try utilizing a vector of negative indexes to complete this task.

```
[1]  96 100  81  72
```

Rcode :

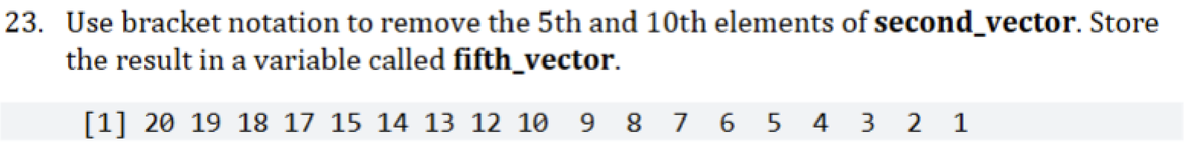
middle_grades_removed <- grades[-c(3, 4)]

Output:



```
> middle_grades_removed
[1]  96 100  81  72
>
```

# Problem 23



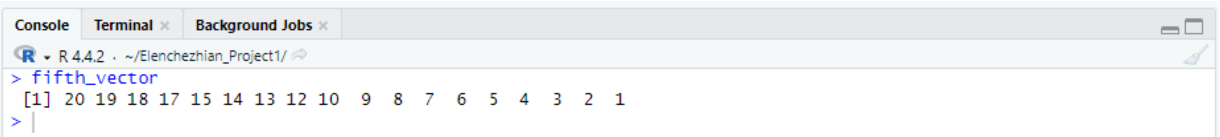
23. Use bracket notation to remove the 5th and 10th elements of **second_vector**. Store the result in a variable called **fifth_vector**.

```
[1] 20 19 18 17 15 14 13 12 10  9  8  7  6  5  4  3  2  1
```

Rcode :

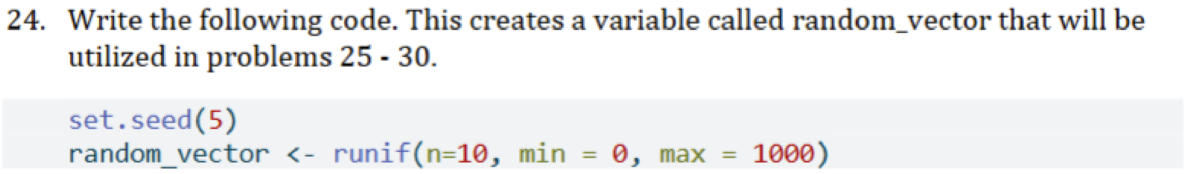
fifth_vector <- second_vector[-c(5, 10)]

Output:



```
> fifth_vector
 [1] 20 19 18 17 15 14 13 12 10  9  8  7  6  5  4  3  2  1
> 
```

# Problem 24



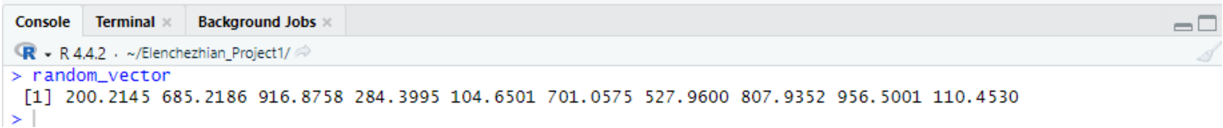
24. Write the following code. This creates a variable called random_vector that will be utilized in problems 25 - 30.

```
set.seed(5)
random_vector <- runif(n=10, min = 0, max = 1000)
```

Rcode :

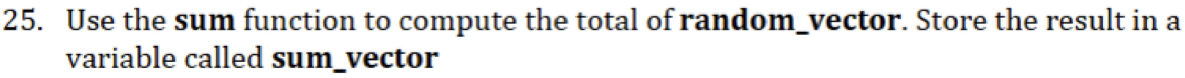
set.seed(5)
random_vector <- runif(n=10, min=0, max=1000)

Output:

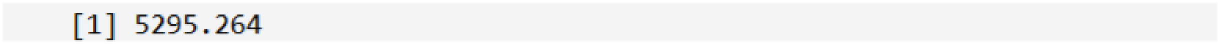


```
> random_vector
 [1] 200.2145 685.2186 916.8758 284.3995 104.6501 701.0575 527.9600 807.9352 956.5001 110.4530
> 
```

# Problem 25



25. Use the **sum** function to compute the total of **random_vector**. Store the result in a variable called **sum_vector**

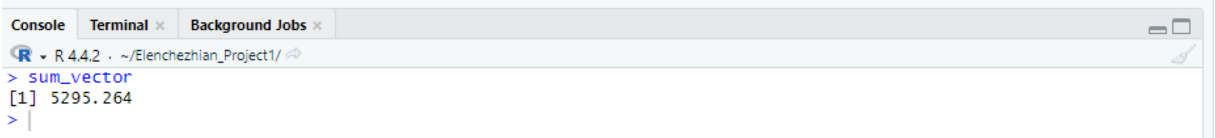


```
[1] 5295.264
```

Rcode :

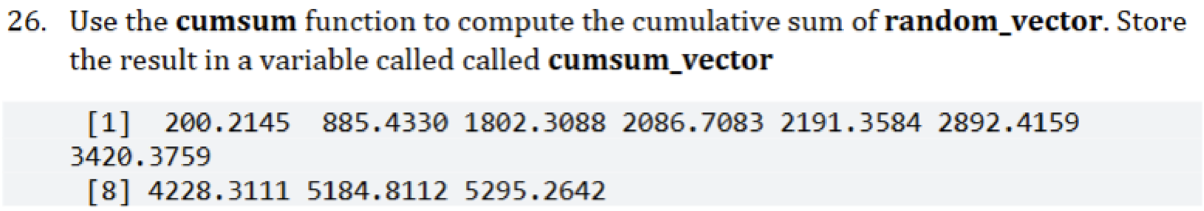
sum_vector <- sum(random_vector)

Output:



```
> sum_vector
[1] 5295.264
> 
```

# Problem 26



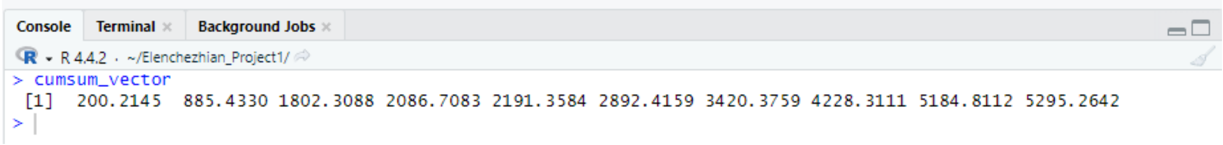
26. Use the **cumsum** function to compute the cumulative sum of **random_vector**. Store the result in a variable called called **cumsum_vector**

```
  [1]  200.2145  885.4330 1802.3088 2086.7083 2191.3584 2892.4159
3420.3759
  [8] 4228.3111 5184.8112 5295.2642
```

Rcode :

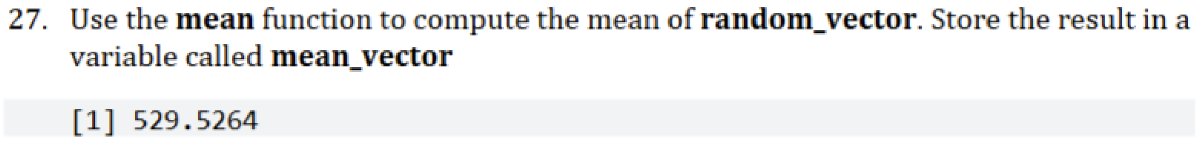
cumsum_vector <- cumsum(random_vector)

Output:



```
> cumsum_vector
 [1]  200.2145  885.4330 1802.3088 2086.7083 2191.3584 2892.4159 3420.3759 4228.3111 5184.8112 5295.2642
> 
```

# Problem 27



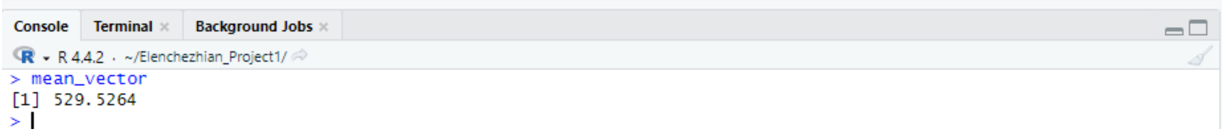
27. Use the **mean** function to compute the mean of **random_vector**. Store the result in a variable called **mean_vector**

```
 [1] 529.5264
```

Rcode :

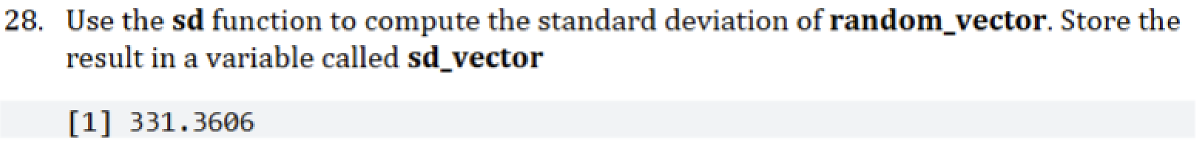
mean_vector <- mean(random_vector)

Output:



```
> mean_vector
[1] 529.5264
> 
```

# Problem 28



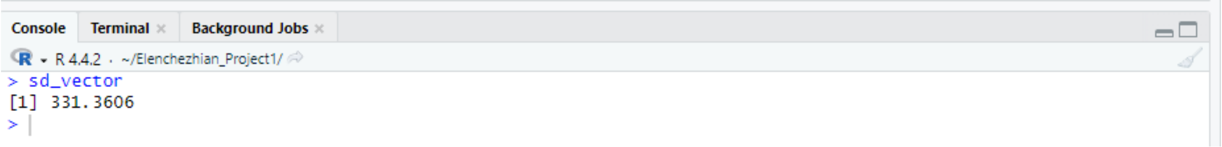
28. Use the **sd** function to compute the standard deviation of **random_vector**. Store the result in a variable called **sd_vector**

```
[1] 331.3606
```

Rcode :

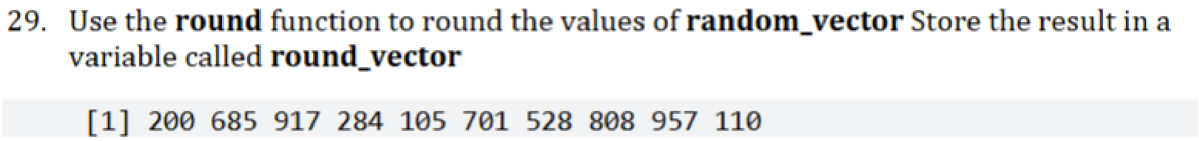
sd_vector <- sd(random_vector)

Output:



```
> sd_vector
[1] 331.3606
> 
```

# Problem 29



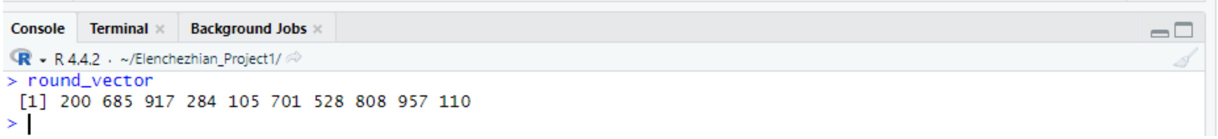
29. Use the **round** function to round the values of **random_vector** Store the result in a variable called **round_vector**

```
[1] 200 685 917 284 105 701 528 808 957 110
```

Rcode :

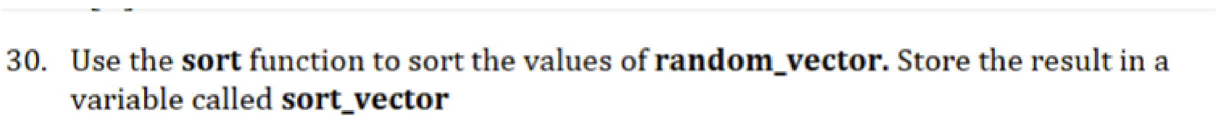
round_vector <- round(random_vector)

Output:



```
> round_vector
 [1] 200 685 917 284 105 701 528 808 957 110
> 
```

# Problem 30

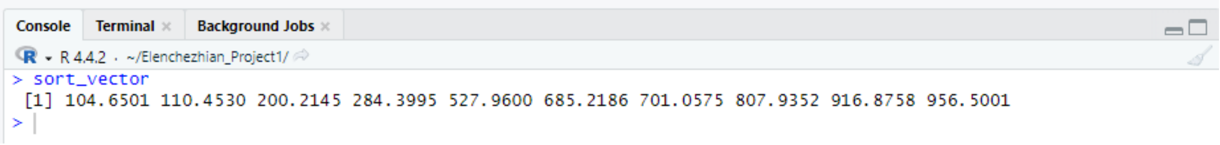


30. Use the **sort** function to sort the values of **random_vector.** Store the result in a variable called **sort_vector**

Rcode :

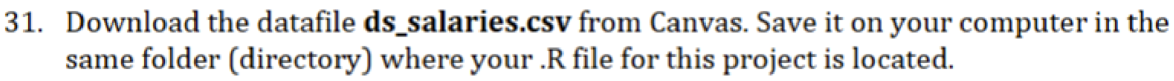
sort_vector <- sort(random_vector)

Output:

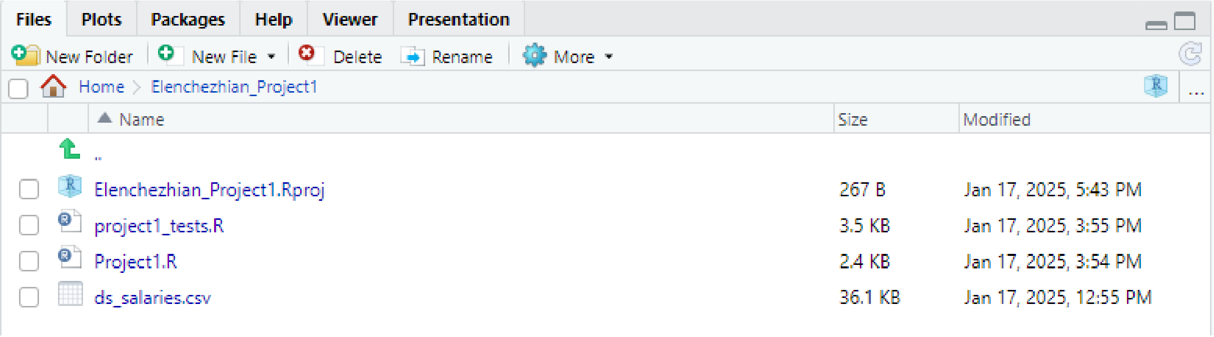


```
> sort_vector
 [1] 104.6501 110.4530 200.2145 284.3995 527.9600 685.2186 701.0575 807.9352 916.8758 956.5001
> |
```

# Problem 31

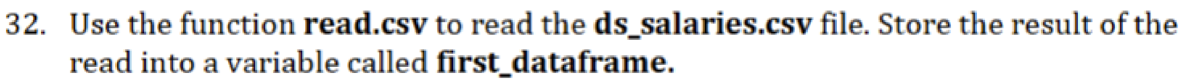


31. Download the datafile **ds_salaries.csv** from Canvas. Save it on your computer in the same folder (directory) where your .R file for this project is located.



Name	Size	Modified
..		
Elenchezhian_Project1.Rproj	267 B	Jan 17, 2025, 5:43 PM
project1_tests.R	3.5 KB	Jan 17, 2025, 3:55 PM
Project1.R	2.4 KB	Jan 17, 2025, 3:54 PM
ds_salaries.csv	36.1 KB	Jan 17, 2025, 12:55 PM

# Problem 32

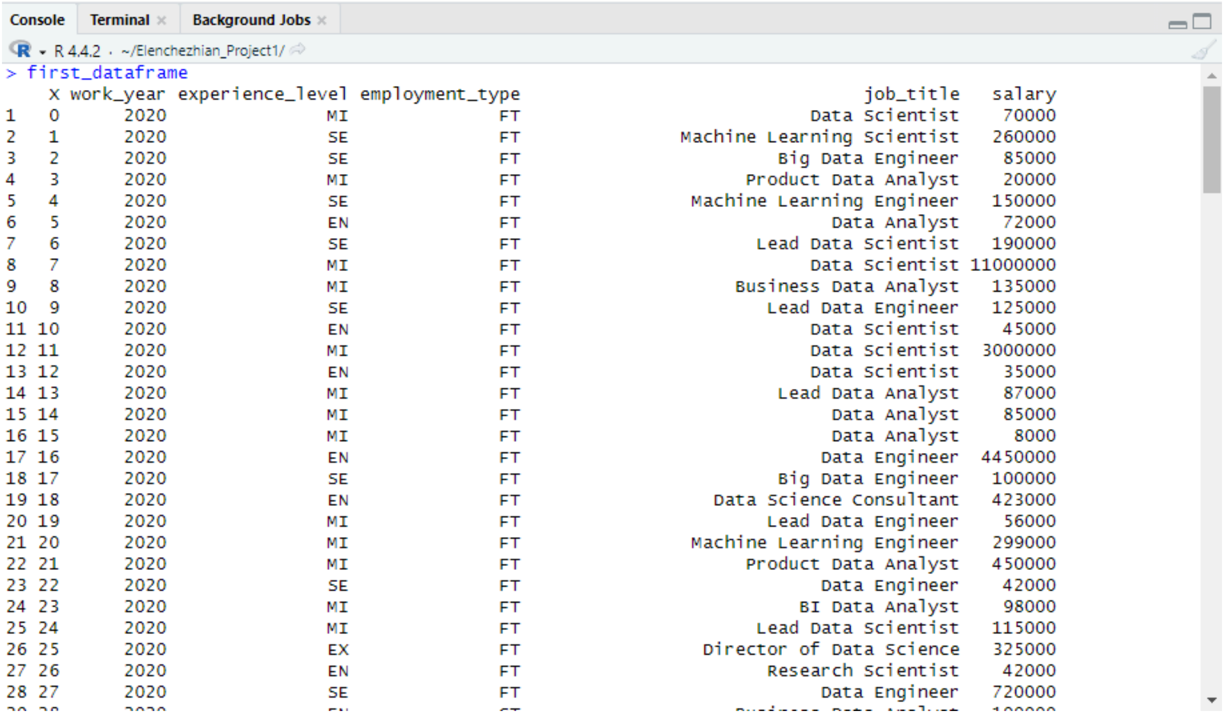


32. Use the function **read.csv** to read the **ds_salaries.csv** file. Store the result of the read into a variable called **first_dataframe.**

Rcode :

first_dataframe <- read.csv("ds_salaries.csv")

Output:



```
> first_dataframe
    X work_year experience_level employment_type                  job_title   salary
1   0      2020               MI              FT             Data Scientist    70000
2   1      2020               SE              FT Machine Learning Scientist   260000
3   2      2020               SE              FT          Big Data Engineer    85000
4   3      2020               MI              FT       Product Data Analyst    20000
5   4      2020               SE              FT  Machine Learning Engineer   150000
6   5      2020               EN              FT               Data Analyst    72000
7   6      2020               SE              FT        Lead Data Scientist   190000
8   7      2020               MI              FT             Data Scientist 11000000
9   8      2020               MI              FT      Business Data Analyst   135000
10  9      2020               SE              FT         Lead Data Engineer   125000
11 10      2020               EN              FT             Data Scientist    45000
12 11      2020               MI              FT             Data Scientist  3000000
13 12      2020               EN              FT             Data Scientist    35000
14 13      2020               MI              FT          Lead Data Analyst    87000
15 14      2020               MI              FT               Data Analyst    85000
16 15      2020               MI              FT               Data Analyst     8000
17 16      2020               EN              FT              Data Engineer  4450000
18 17      2020               SE              FT          Big Data Engineer   100000
19 18      2020               EN              FT    Data Science Consultant   423000
20 19      2020               MI              FT         Lead Data Engineer    56000
21 20      2020               MI              FT  Machine Learning Engineer   299000
22 21      2020               MI              FT       Product Data Analyst   450000
23 22      2020               SE              FT              Data Engineer    42000
24 23      2020               MI              FT            BI Data Analyst    98000
25 24      2020               MI              FT        Lead Data Scientist   115000
26 25      2020               EX              FT   Director of Data Science   325000
27 26      2020               EN              FT         Research Scientist    42000
28 27      2020               SE              FT              Data Engineer   720000
```

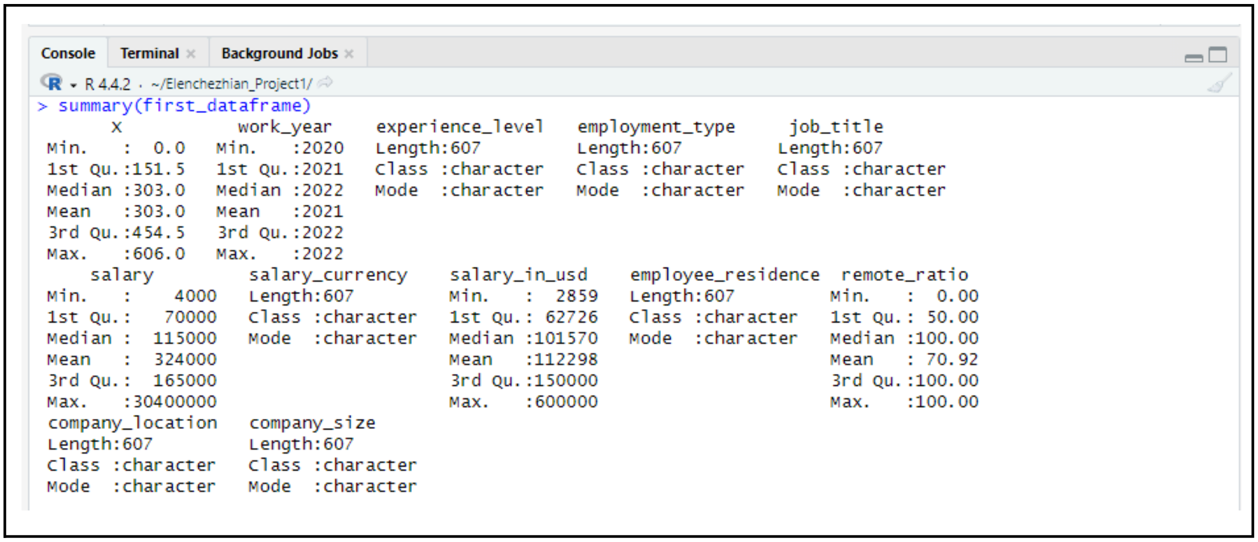
# Problem 33

33. Use the **summary** function with **first_dataframe** to produce summary statistics based on each column of the data frame.

Rcode :

summary(first_dataframe)

Output:



```
> summary(first_dataframe)
       X            work_year    experience_level   employment_type     job_title
 Min.   :  0.0   Min.   :2020   Length:607         Length:607         Length:607
 1st Qu.:151.5   1st Qu.:2021   Class :character   Class :character   Class :character
 Median :303.0   Median :2022   Mode  :character   Mode  :character   Mode  :character
 Mean   :303.0   Mean   :2021
 3rd Qu.:454.5   3rd Qu.:2022
 Max.   :606.0   Max.   :2022
     salary         salary_currency    salary_in_usd    employee_residence  remote_ratio
 Min.   :    4000   Length:607         Min.   :  2859   Length:607         Min.   :  0.00
 1st Qu.:   70000   Class :character   1st Qu.: 62726   Class :character   1st Qu.: 50.00
 Median :  115000   Mode  :character   Median :101570   Mode  :character   Median :100.00
 Mean   :  324000                      Mean   :112298                      Mean   : 70.92
 3rd Qu.:  165000                      3rd Qu.:150000                      3rd Qu.:100.00
 Max.   :30400000                      Max.   :600000                      Max.   :100.00
 company_location   company_size
 Length:607         Length:607
 Class :character   Class :character
 Mode  :character   Mode  :character
```