WEEK – 1

ALY6000 – Introduction to Analytics

PROJECT - 1

NORTHEASTERN UNIVERSITY

College of Professional Studies, Northeastern University, Boston, MA 02215

Submitted By

Name: Rohit Kosamkar

NUID: 002478862

Dated: January 14, 2024

PROJECT 1

1. Write lines of code to compute all the following. Include the answers in your written report.
   123 * 453
   5^2 * 40
   TRUE & FALSE
   TRUE | FALSE
   75 %% 10
   75 / 10

Answer:

```
#1:
print(123 * 453)
print(5^2 * 40)
print(TRUE & FALSE)
print(TRUE | FALSE)
print(75 %% 10)
print(75/10)
```

```
[1] 55719
[1] 1000
[1] FALSE
[1] TRUE
[1] 5
[1] 7.5
```

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

Answer:

```
#2.
first_vector <- c(17, 12, -33, 5)
print(first_vector)
```

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

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

Answer:

```
#3.
counting_by_fives <- c(5,10,15,20,25,30,35)
print(counting_by_fives)
```

```
[1]  5 10 15 20 25 30 35
```

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.

Answer:

```
#4.
second_vector <- 20:1
```

```
print(second_vector)

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

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

Answer:

```
#5.
counting_vector <- 5:15
print(counting_vector)

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

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

Answer:

```
#6.
grades <- c(96,100,85,92,81,72)
print(grades)

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

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

Answer:

```
#7.
bonus_points_added <- grades + 3
print(bonus_points_added)

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

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.

Answer:

```
#8.
one_to_one_hundred <- 1:100
print(one_to_one_hundred)

  [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
```

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"
```

Answer:

```
#9.1 adding 20 to each element in second_vector
print (second_vector + 20)
[1] 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21

#9.2 multiplying by 20 to each element in second vector
print(second_vector * 20)
[1] 400 380 360 340 320 300 280 260 240 220 200 180 160 140 120 100  80  60
[19]  40  20

#9.3 This code gives a Boolean value output where elements in second_vector which are
greater than equal to 20 are 'TRUE' otherwise 'FALSE'
print(second_vector >= 20)
[1]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE
[13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE

#9.4 This code gives a Boolean value output where elements in second_vector which are not
equal to 20 are 'TRUE' otherwise 'FALSE'
print(second_vector != 20)
[1] FALSE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
TRUE
[13]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

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

Answer:

```
#10.
total <- sum(one_to_one_hundred)
print(total)
[1] 5050
```

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

Answer:

```
#11.
average_value <- mean(one_to_one_hundred)
print(average_value)
[1] 50.5
```

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

Answer:

```
#12.
median_value <- median(one_to_one_hundred)
print(median_value)
[1] 50.5
```

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

Answer:

```
#13.
max_value <- max(one_to_one_hundred)
print(max_value)
[1] 100
```

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

Answer:

```
#14.
min_value <- min(one_to_one_hundred)
print(min_value)
[1] 1
```

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

Answer:

```
#15.
first_value <- second_vector[1]
print(first_value)
[1] 20
```

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

Answer:

```
#16.
first_three_values <- second_vector[1:3]
print(first_three_values)
[1] 20 19 18
```

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

Answer:

```
#17.
vector_from_brackets <- second_vector[c(1,5,10,11)]
print(vector_from_brackets)
[1] 20 16 11 10
```

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.

Answer:

#18. In the vector first_vector, represented as [17, 12, -33, 5], applying a boolean vector for extraction results in retrieving only the values corresponding to the 'true' entries in the boolean vector

```
vector_from_boolean_brackets <- first_vector[c(FALSE,TRUE,FALSE,TRUE)]
print(vector_from_boolean_brackets)
```

[1] 12 5

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

Answer:

#19. This code evaluates each element in second_vector to determine if it is greater than 10, generating a logical vector with 'true' for elements greater than or equal to 10, and 'false' otherwise
print(second_vector >= 10)

[1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE FALSE
[13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE

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]

Answer:

#20.
print(one_to_one_hundred[one_to_one_hundred >= 20])
print(one_to_one_hundred)
#this code creates a new vector where all elements from one_to_one_hundred vector which are greater than or equal to 20 are included

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

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.

Answer:

#21.
lowest_grades_removed <- grades[grades >85]
print(lowest_grades_removed)
[1] 96 100 92

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.

Answer:

#22.
middle_grades_removed <- grades[-c(3,4)]
print(middle_grades_removed)
[1] 96 100 81 72

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

Answer:

```
#23.
fifth_vector <- second_vector[-c(5,10)]
print(fifth_vector)
[1] 20 19 18 17 15 14 13 12 10  9  8  7  6  5  4  3  2  1
```

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

Answer:

```
#24.
set.seed(5)
random_vector <- runif(n=10, min=0, max=1000)
print(random_vector)
[1] 200.2145 685.2186 916.8758 284.3995 104.6501 701.0575 527.9600 807.9352
 [9] 956.5001 110.4530
```

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

Answer:

```
#25.
sum_vector <- sum(random_vector)
print(sum_vector)
[1] 5295.264
```

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

Answer:

```
#26.
cumsum_vector <- cumsum(random_vector)
print(cumsum_vector)
[1]  200.2145  885.4330 1802.3088 2086.7083 2191.3584 2892.4159 3420.3759
 [8] 4228.3111 5184.8112 5295.2642
```

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

Answer:

```
#27.
mean_vector <- mean(random_vector)
print(mean_vector)
[1] 529.5264
```

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

Answer:

```
#28.
sd_vector <- sd(random_vector)
```

```
print(sd_vector)
[1] 331.3606
```

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

Answer:

```
#29.
round_vector <- round(random_vector)
print(round_vector)
[1] 200 685 917 284 105 701 528 808 957 110
```

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

Answer:

```
#30.
sort_vector <- sort(random_vector)
print(sort_vector)
[1] 104.6501 110.4530 200.2145 284.3995 527.9600 685.2186 701.0575 807.9352
 [9] 916.8758 956.5001
```

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.

Answer:

```
#31. file downloaded and saved at (../Kosamkar-Project1/ds_salaries.csv)
```

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.

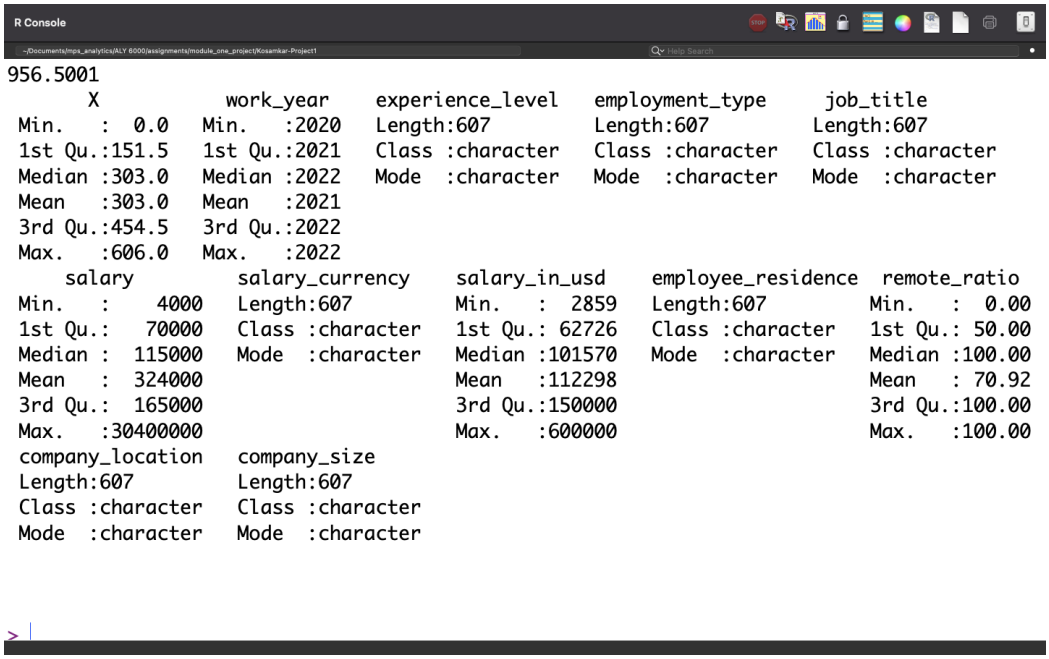
Answer:

```
first_dataframe <- read.csv("ds_salaries.csv")
print(head(first_dataframe))
```

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

Answer:

```
print(summary(first_dataframe)
```


R Console
956.5001
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
>

Thank You.