R programming Assignment-4

1.How can you produce co-relations and covariances?\

ANS.

Gathering data on another variable:

Over the course of your time on the islands, you notice that the teacup giraffes seem to have an affinity for celery, which you have already used to entice so they come close enough for a height measurement. Suprisingly, one of the small ones had eaten so much of it! You decide to quantify how much celery each of the giraffes consumed to see if there is any relationship to height.

You systematically measure the amount of celery eaten and add it to your log of data, which is stored as a data frame called giraffe_data.

We can check out the first entries of the data frame by using the head( ) command:

```
head(giraffe_data)
##     Heights Celery_Eaten
## 1  7.038865     16.36129
## 2 13.154339     10.72354
## 3  8.086511     17.16798
## 4  8.159990     22.52115
## 5  6.004716     13.77993
## 6  9.455408     16.92304
```

Make a scatter plot:

It’s difficult to get an idea if there’s any relationship by just looking at the data frame. We learn so much more from creating a plot. Let’s revisit our newly acquired ggplot skills to make a scatter plot.

A lot of the code used previously can be re-used for the scatter plot. Two main differences are the following:

1. Because we now have an additional variable, we need to **assign a y = for the aes( ) command within ggplot( )**. Create the plot so that Celery_Eaten will be on the y-axis.

2. **Add (+) a geom_point( )** element instead of geom_hist( )

Also, we will add lines to our plot, representing the mean of each variable. Here’s how we’ll do that:

1. **Add a horizontal line** by adding (+) a geom_hline( ) component. This takes the argument yintercept =, which equals the value for where the horizontal line should cross the y-intercept.
   - Since we want to place this line at the mean of y variable (y¯y¯), we can use the mean( ) function instead of specifying a numeric value directly.
2. **Add a vertical line** by following the same structure as above but using geom_vline( ) and xintercept = instead. This vertical line will represent the mean (x¯x¯) of the heights.

Crossproduct:

After obtaining the deviation scores, we need to combine them into a single measure. We do not simply combine the deviation scores themselves by adding (please see the page about the variance for a discussion of why this is). Instead, we take both deviation scores from the same observation and multiply them together to create a two-dimensional “shape” (analagous to when we multiplied a single deviation score by itself to create a square in the variance calculation).

NOTE: The resulting value is now on the order of a “squared” term. This will become important later.

1. **Multiply the two deviation scores.** This is called the **crossproduct**. The shapes created by the crossproduct will serve as the “squared” terms that we can then use in the next step to sum and summarize the deviations into a single value.

The equation is shown below for the *sample* crossproduct of the deviation scores.

$$(xi-\bar{x})(yi-\bar{y})(1)(1)(xi-\bar{x})(yi-\bar{y})$$

Let’s explore the attributes of the crossproduct:

- First, we should note that the *overall sign* of the crossproduct will depend on whether or not the two x- and y- values from the same observation move in the same directions relative to their means. Look at the annotated plot below. The crossproducts will either create “negative” shapes (shown in blue) or “positive” shapes (in red). A third outcome is that the crossproduct could also be 0 – this will occur when an observation falls on the mean.

- Second, the **magnitude of the crossproduct** will scale with the absolute value of the deviation scores. In other words, the further away both deviation scores are from their means, the larger the area of their shapes will be.

- **Sum the crossproducts.** The sum of the crossproduct gives us a single number.

$$\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y}) \qquad (2)$$

Covariance:

- We then **divide the sum of the crossproduct by $n-1$ (or N in the population equation)** so that we have taken into account how many observations contributed to this quantity. (Why $n-1$? See here.) This final number is called the **covariance**, and its value tells us how much our two variables fluctuate together. The higher the absolute value, the stronger the relationship.

The equation for the covariance (abbreviated "cov") of the variables x and y is shown below. As a preference of style, we multiply by $\frac{1}{n-1}$ instead of dividing the entire term by $n-1$.

$$\text{cov}(x,y)=\frac{1}{n-1}\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y}) \qquad (3)$$

Problem of interpretation:

However, the covariance is not an intuitive value. Remember that we have been working with terms that are on the "squared" scale, which is not only difficult to interpret (just like the variance is) but it is also the product of two variables on possibly different scales. How could we interpret a covariance with units of millimeters*grams mean?

Another point to make is that value of the covariance will be vastly different if we had decided to change the units (millimeters vs centimeters, or grams vs kilograms).

As a result, the covariance is not an easy metric to work with or to compare with other covariances. So we need to standardize it!

correlation coefficient, r:

How do we standardize the covariance?

The solution is to (1) take the standard deviations of each variable, (2) multiply them together, and (3) divide the covariance by this product – the resulting value is called the **Pearson correlation coefficient**. When referring to the population correlation coefficient, the symbol ρρ (pronounced "rho") is used. When referring to the sample correlation coefficient, a lowercase rr is used (often called "Pearson's r").

Here is the equation for the **population correlation**:

ρ(x,y)=1N∑Ni=1(xi−μx)(yi−μy)σxσy(4)(4)ρ(x,y)=1N∑i=1N(xi−μx)(yi−μy)σxσy

This equation is used for the **sample correlation**:

r(x,y)=1n−1∑ni=1(xi−x¯)(yi−y¯)sxsy(5)

2.What is t-tests in R?

ANS: T-tests in R is one of the most common tests in statistics. So, we use it to determine whether the means of two groups are equal to each other. The assumption for the test is that both groups are sampled from normal distributions with equal variances. The null hypothesis is that the two means are equal, and the alternative is that they are not equal. It is known that under the null hypothesis, we can calculate a t-statistic that will follow a t-distribution with **n1 + n2 – 2 degrees of freedom**.

Welch’s T-test is a user modification of the T-test that adjusts the number of degrees of freedom when the variances are thought not to be equal to each other.

We use t.test() which provides a variety of T-tests:

# independent 2-group T-test
t.test(y~x) # where y is numeric and x is a binary factor

# independent 2-group T-test
t.test(y1,y2) # where y1 and y2 are numeric

# paired T-test
t.test(y1,y2,paired=TRUE) # where y1 & y2 are numeric

# one sample T-test
t.test(y,mu=3) # Ho: mu=3

3.What data structures in R are responsible for Statistical graphs and creating graphs?

ANS:

The data structures those are useful to perform statistical analysis and to create graphs are: vectors, arrays, data frames and matrices.

4.Write a program to combine three arrays so that the first row of the first array is followed by the first row of the second array and then the first row of the third array?

ANS:

R Programming Code :

```
num1 = rbind(rep("A",3), rep("B",3), rep("C",3))

print("num1")

print(num1)

num2 = rbind(rep("P",3), rep("Q",3), rep("R",3))

print("num2")

print(num2)

num3 = rbind(rep("X",3), rep("Y",3), rep("Z",3))

print("num3")

print(num3)

a = matrix(t(cbind(num1,num2,num3)),ncol=3, byrow=T)

print("Combine three arrays, taking one row from each one by one:")

print(a)
```

OUTPUT:

```
[1] "num1"
     [,1] [,2] [,3]
```

```
[1,] "A"  "A"  "A"
[2,] "B"  "B"  "B"
[3,] "C"  "C"  "C"
[1] "num2"
     [,1] [,2] [,3]
[1,] "P"  "P"  "P"
[2,] "Q"  "Q"  "Q"
[3,] "R"  "R"  "R"
[1] "num3"
     [,1] [,2] [,3]
[1,] "X"  "X"  "X"
[2,] "Y"  "Y"  "Y"
[3,] "Z"  "Z"  "Z"
[1] "Combine three arrays, taking one row from each one by one:"
      [,1] [,2] [,3]
 [1,] "A"  "A"  "A"
 [2,] "P"  "P"  "P"
 [3,] "X"  "X"  "X"
 [4,] "B"  "B"  "B"
 [5,] "Q"  "Q"  "Q"
 [6,] "Y"  "Y"  "Y"
 [7,] "C"  "C"  "C"
 [8,] "R"  "R"  "R"
 [9,] "Z"  "Z"  "Z"
```

5.Write a program to create a two-dimensional 6x4 array of sequences of even integers greater than 60?

ANS:

R Programming Code :

```
a <- array(seq(from = 60, length.out = 24, by = 2), c(6, 4))

print("Content of the array:")

print("5×3 array of sequence of even integers greater than 60:")

print(a)
```

Output:

```
[1] "Content of the array:"

[1] "6×4 array of sequence of even integers greater than 60:"

     [,1]    [,2]    [,3]     [,4]

[1,] 60     70     80     90

[2,] 62      72    82     92

[3,] 64      74    84     94

[4,]  66     76     86    96

[5,]  68     78      88   98

[6 ]  60      70      80   90
```

6.What is the general form of matrices in R?

ANS:

General form :

In R, a two-dimensional rectangular data set is known as a matrix. A matrix is created with the help of the vector input to the matrix function. On R matrices, we can perform addition, subtraction, multiplication, and division operation. In the R matrix, elements are arranged in a fixed number of rows and columns.

Defnition:

Matrices are the R objects in which the elements are arranged in a two-dimensional rectangular layout. They contain elements of the same atomic types. Though we can create a matrix containing only characters or only logical values, they are not of much use. We use matrices containing numeric elements to be used in mathematical calculations.

A Matrix is created using the **matrix()** function.

Syntax:

The basic syntax for creating a matrix in R is −

```
matrix(data, nrow, ncol, byrow, dimnames)
```

Following is the description of the parameters used −

- **data** is the input vector which becomes the data elements of the matrix.
- **nrow** is the number of rows to be created.
- **ncol** is the number of columns to be created.

- **byrow** is a logical clue. If TRUE then the input vector elements are arranged by row.
- **dimname** is the names assigned to the rows and columns.

7.Write a program to create an empty Data Frame?

ANS:

R Programming Code :

```
df = data.frame(Ints=integer(),
                Doubles=double(),
                Characters=character(),
                Logicals=logical(),
                Factors=factor(),
                stringsAsFactors=FALSE)
print("Structure of the empty dataframe:")
print(str(df))
```

Output:

```
[1] "Structure of the empty dataframe:"
'data.frame':    0 obs. of  5 variables:
 $ Ints      : int
 $ Doubles   : num
 $ Characters: chr
 $ Logicals  : logi
 $ Factors   : Factor w/ 0 levels:
NULL
```