Table of Contents

Applying K-means Clustering

K-means Clustering

Using the K-means clustering technique, we assign each observation in a dataset to one of K clusters.

The ultimate goal is to have K clusters in which the observations are quite diverse from one another across clusters, but fairly similar to one another inside each cluster.

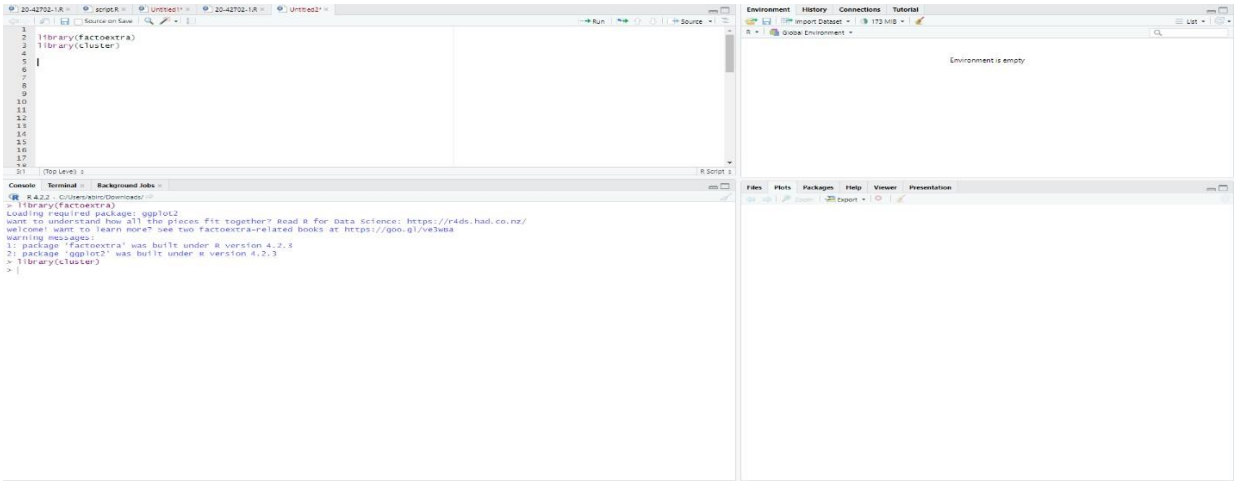
Dataset

For this project I have chosen the kidney-stone dataset. Age, gender, weight, height, family history, and numerous blood and urine parameters are just a few of the 12 characteristics in this kidney stone dataset that are linked to the likelihood of having kidney stones. The target variable, which is a continuous value expressing the chance of acquiring kidney stones, is part of a dataset of 180 entries. The dataset is helpful for developing predictive models that can help identify people who have a high chance of developing kidney stones, which can help with the creation of preventative and treatment plans. The dataset includes details on 90 patients who had kidney stones, including age, gender, and the type of stone each patient had as well as the outcomes of laboratory tests. The link for the dataset: https://www.kaggle.com/datasets/harshghadiya/kidneystone?resource=download.

The following shows all the necessary steps to perform K-means clustering on the chosen dataset in R.

Step 1: Loading the Necessary Packages

I have first loaded two packages that have a number of helpful R functions for k-means clustering.

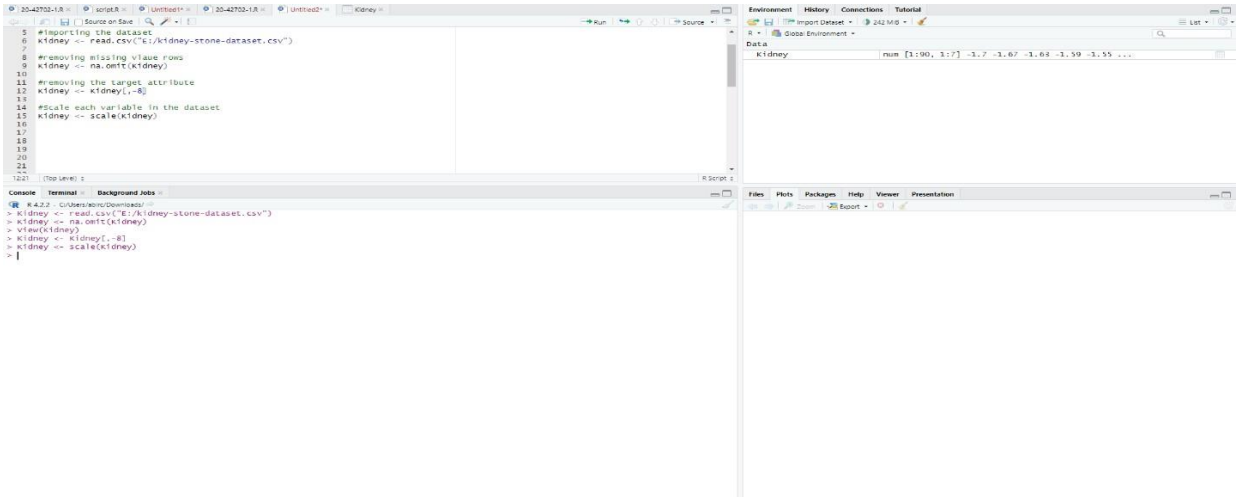


Step 2: Importing and Preparing the Data

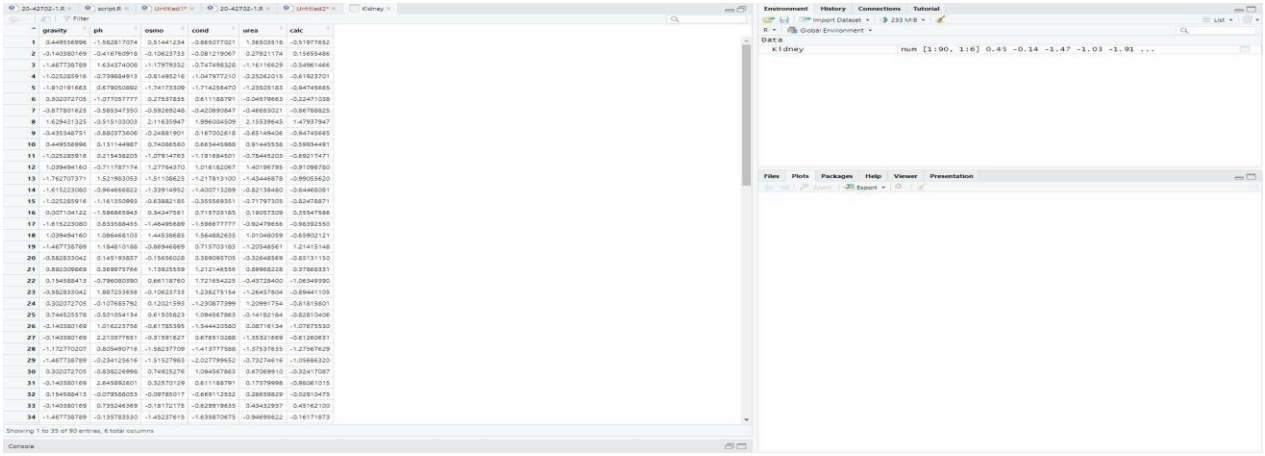
Here the dataset used is kidney-stone dataset. This dataset contains 12 features related to the risk of developing kidney stones, including age, gender, weight, height, family history, and various blood and urine measurements.

The code below demonstrates the following:

- Import the kidney-stone dataset
- Remove any rows with missing values
- Remove the target attribute
- Scale each variable in the dataset to have a mean of 0 and a standard deviation of 1



After step 2, the dataset:



Step 3: Finding the Optimal Number of Clusters

The built-in **kmeans()** function in R can be used to do k-means clustering and has the following syntax:

kmeans(data, centers, iter.max, nstart)

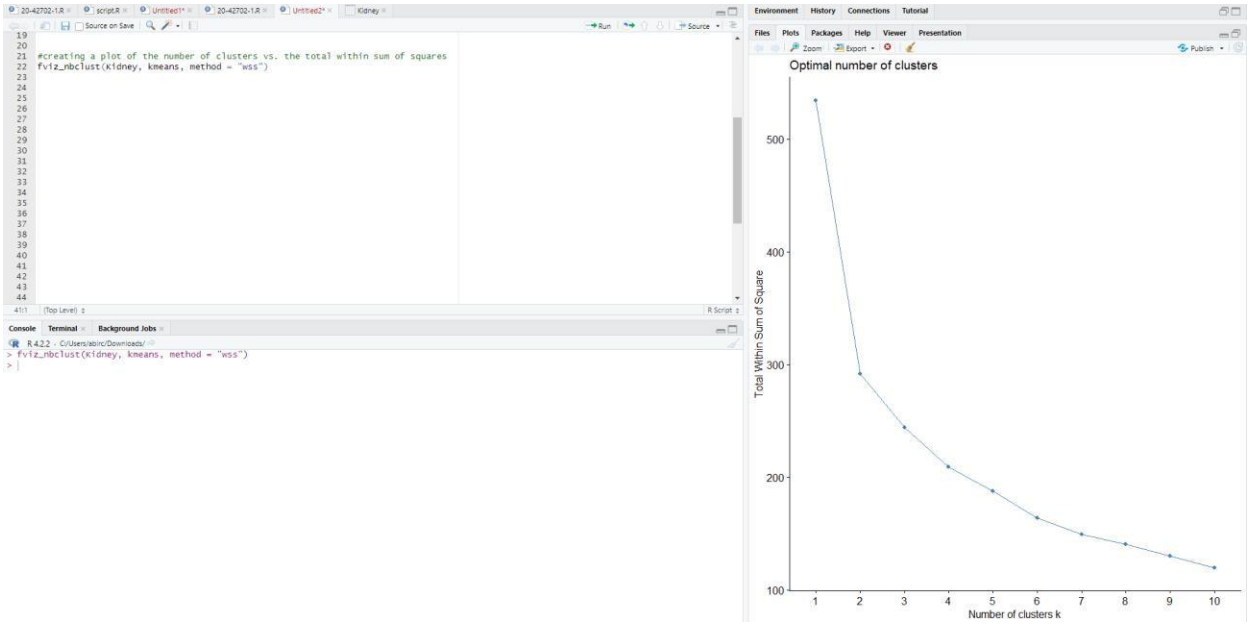
Here:

- **data:** Name of the dataset.
- **centers:** The number of centers denoted by k.
- **iter.max:** The maximum number of iterations allowed. The default value is 10.
- **nstart:** The number of random starting partitions.

Since how many clusters will be optimal is unknown, two different plots were chosen here for the help of deciding:

Step 3.1: Total Within Sum of Squares

At first the **fviz_nbclust()** function was used to plot the number of clusters vs the total within sum of squares.



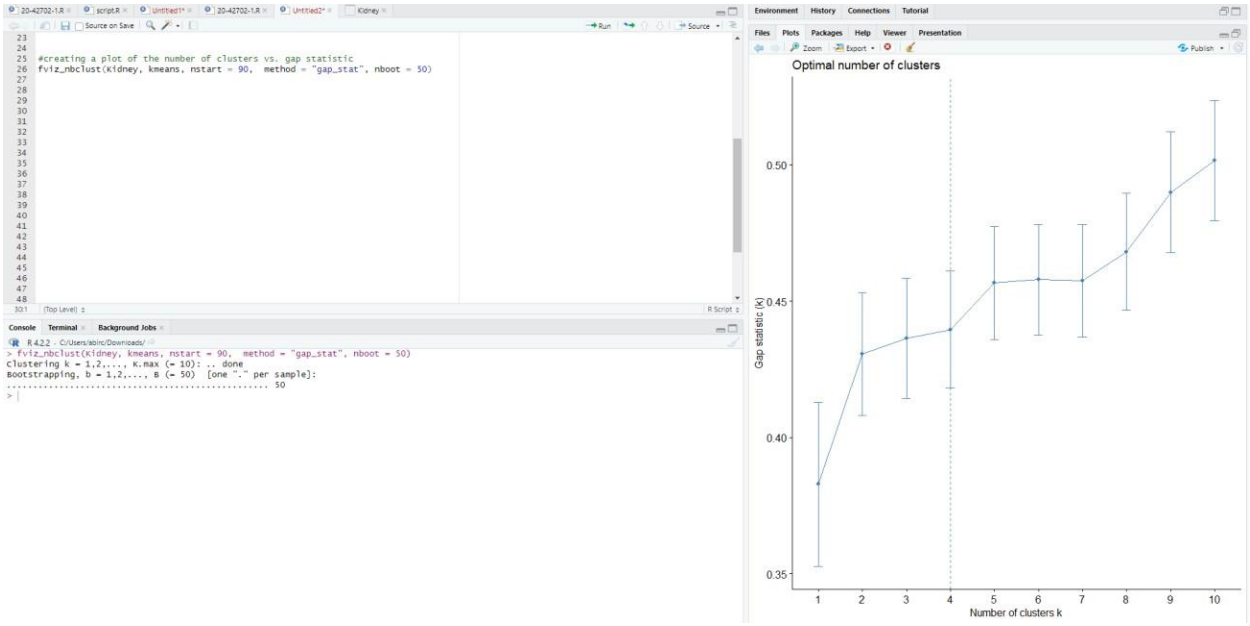
Usually, when making this kind of plot, we search for an "elbow" shape where the total number of squares starts to bend or level off. The ideal number of clusters is usually this.

There appears to be a slight elbow or bend in this figure at the k = 4 clusters.

Step 3.2: Gap Statistics

Utilizing the gap statistic, which contrasts the total intra-cluster variation for various values of k with their anticipated values for a distribution with no clustering, is another method for figuring out the ideal number of clusters.

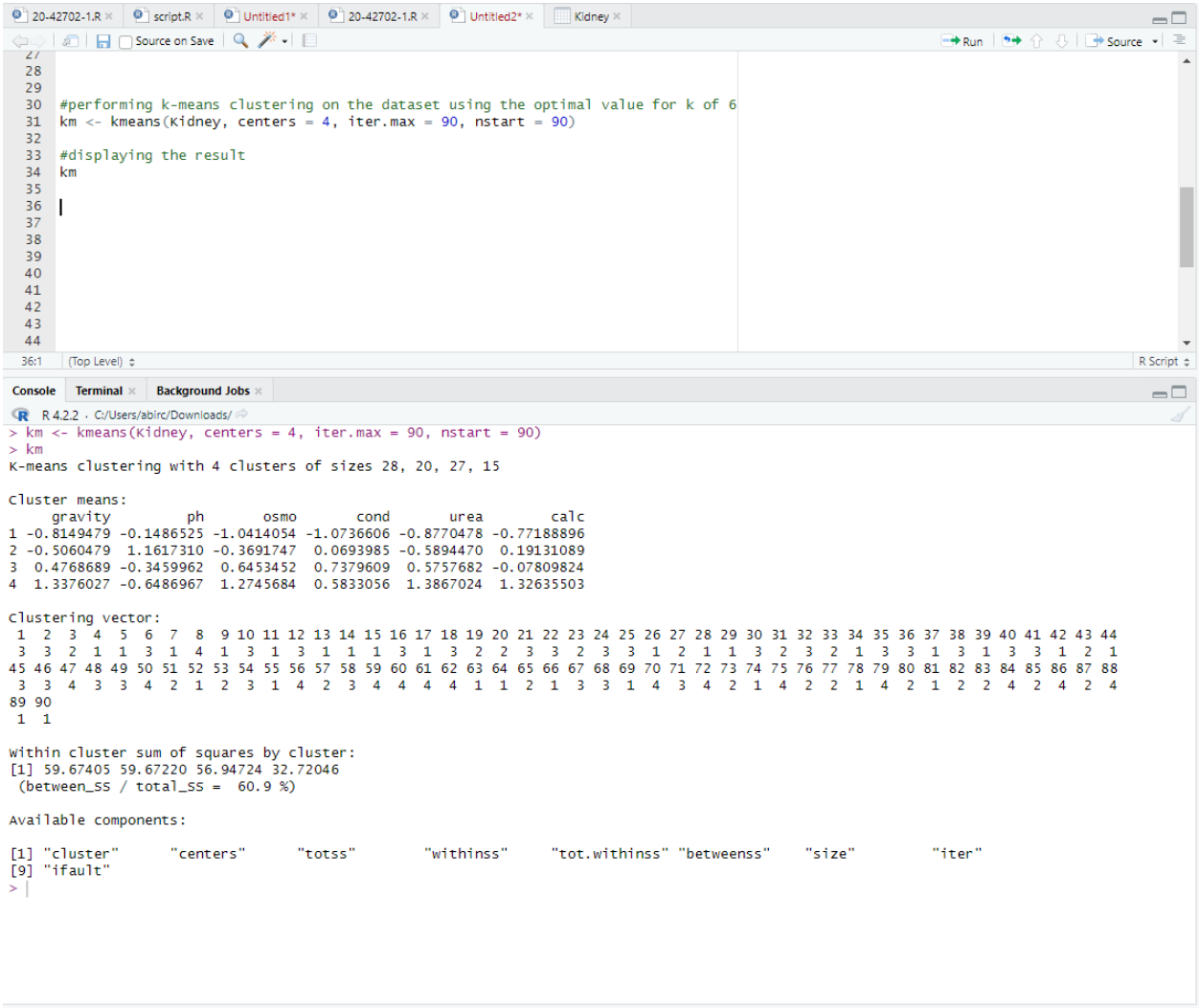
It can be calculated using the **fviz_nbclust()** function.



From the plot we can see that gap statistic suggests k = 4 clusters, which matches the Within Sum of Squares method.

Step 4: Performing K-Means Clustering with Optimal Clusters

Lastly, k-means clustering was performed on the dataset using the optimal value for *k* of 4:



```
#performing k-means clustering on the dataset using the optimal value for k of 6
km <- kmeans(Kidney, centers = 4, iter.max = 90, nstart = 90)

#displaying the result
km
```

```
> km <- kmeans(Kidney, centers = 4, iter.max = 90, nstart = 90)
> km
K-means clustering with 4 clusters of sizes 28, 20, 27, 15

Cluster means:
     gravity         ph       osmo       cond       urea        calc
1 -0.8149479 -0.1486525 -1.0414054 -1.0736606 -0.8770478 -0.77188896
2 -0.5060479  1.1617310 -0.3691747  0.0693985 -0.5894470  0.19131089
3  0.4768689 -0.3459962  0.6453452  0.7379609  0.5757682 -0.07809824
4  1.3376027 -0.6486967  1.2745684  0.5833056  1.3867024  1.32635503

Clustering vector:
 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 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44
 3  3  2  1  1  3  1  4  1  3  1  3  1  1  1  3  1  3  2  2  3  3  2  3  3  1  2  1  1  3  2  3  2  1  3  3  1  3  1  3  3  1  2  1
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 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88
 3  3  4  3  3  4  2  1  2  3  1  4  2  3  4  4  4  4  1  1  2  1  3  3  1  4  3  4  2  1  4  2  2  1  4  2  1  2  2  4  2  4  2  4
89 90
 1  1

Within cluster sum of squares by cluster:
[1] 59.67405 59.67220 56.94724 32.72046
 (between_SS / total_SS =  60.9 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"    "size"         "iter"
[9] "ifault"
>
```

The result showed the following:

- 28 instances in cluster 1
- 20 instances in cluster 2
- 27 instances in cluster 3
- 15 instances in cluster 4

Visualizing the Clusters

The **fivz_cluster()** function can be used to see the clusters on a scatterplot with the first two principal components shown on the axes: