Customer Segmentation

using

Machine Learning in R

- Suryansh Singh [1828079]
- Sourav Gupta [1828112]
- Soumya Shekhar Maurya [1828111]
- Yash Vardhan Sinha [1828128]
- Satyajit Biswal [1828098]
- Ranadeb Chaudhuri [1828090]
- Manav Arora [1828079]

Introduction

Customer Segmentation is the process of division of customer base into several groups of individuals that share a similarity in different ways that are relevant to marketing such as gender, age, interests, and miscellaneous spending habits.

Companies aim to gain a deeper approach of the customer they are targeting. Therefore, their aim has to be specific and should be tailored to address the requirements of each and every individual customer. Furthermore, through the data collected, companies can gain a deeper understanding of customer preferences as well as the requirements for discovering valuable segments that would reap them maximum profit. This way, they can strategize their marketing techniques more efficiently and minimize the possibility of risk to their investment.

The technique of customer segmentation is dependent on several key differentiators that divide customers into groups to be targeted. Data related to demographics, geography, economic status as well as behavioural patterns play a crucial role in determining the company direction towards addressing the various segments.

Customer Segmentation is one the most important applications of unsupervised learning. Using clustering techniques, companies can identify the several segments of customers allowing them to target the potential user base. In this machine learning project, we will make use of K-means clustering which is the essential algorithm for clustering unlabelled dataset.

Link for Data set -
:https://drive.google.com/file/d/19BOhwz52NUY3dg8XErVYglctpr5sjTy4/view

Design and Working

Procedure to Implement Customer Segmentation in R

In the first step of this data science project, we will perform data exploration. We will import the essential packages required for this role and then read our data. Finally, we will go through the input data to gain necessary insights about it.

```
customer_data=read.csv("/home/dataflair/Mall_Customers.csv")
str(customer_data)
```

```
## 'data.frame':    200 obs. of  5 variables:
##  $ CustomerID            : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Gender                : Factor w/ 2 levels "Female","Male": 2 2 1 1 1 1 1 1 2 1
...
##  $ Age                   : int  19 21 20 23 31 22 35 23 64 30 ...
##  $ Annual.Income..k..    : int  15 15 16 16 17 17 18 18 19 19 ...
##  $ Spending.Score..1.100.: int  39 81 6 77 40 76 6 94 3 72 ...
```

```
names(customer_data)
```

```
## [1] "CustomerID"             "Gender"
## [3] "Age"                    "Annual.Income..k.."
## [5] "Spending.Score..1.100."
```

We will now display the first six rows of our dataset using the head() function and use the summary() function to output summary of it.

```
head(customer_data)
```

```
##   CustomerID Gender Age Annual.Income..k.. Spending.Score..1.100.
## 1          1   Male  19                  15                     39
## 2          2   Male  21                  15                     81
## 3          3 Female  20                  16                      6
## 4          4 Female  23                  16                     77
## 5          5 Female  31                  17                     40
## 6          6 Female  22                  17                     76
```

```
summary(customer_data$Age)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   18.00   28.75   36.00   38.85   49.00   70.00
```

```
sd(customer_data$Age)
```

```
## [1] 13.96901
```

```
summary(customer_data$Annual.Income..k..)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   15.00   41.50   61.50   60.56   78.00  137.00
```

```
sd(customer_data$Annual.Income..k..)
```

```
## [1] 26.26472
```

```
summary(customer_data$Age)
```

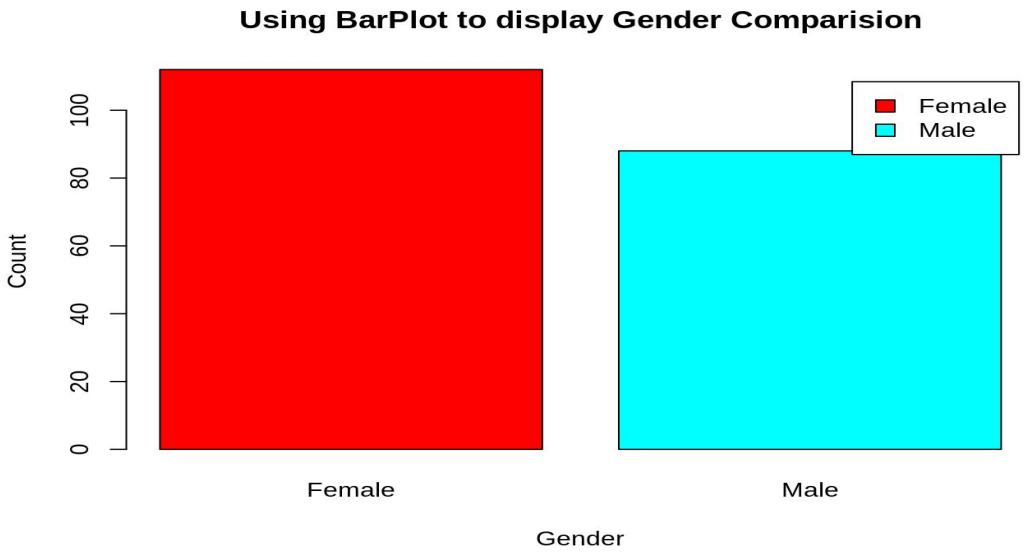
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   18.00   28.75   36.00   38.85   49.00   70.00
```

```
sd(customer_data$Spending.Score..1.100.)
```

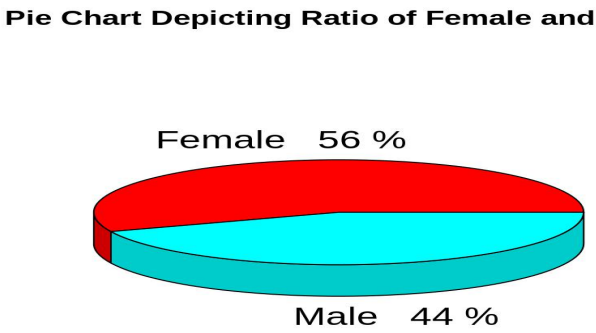
```
## [1] 25.82352
```

Customer Gender Visualization

In this, we will create a barplot and a piechart to show the gender distribution across our customer_data dataset.



From the above barplot, we observe that the number of females is higher than the males. Now, let us visualize a pie chart to observe the ratio of male and female distribution.



From the above graph, we conclude that the percentage of females is **56%**, whereas the percentage of male in the customer dataset is **44%**.

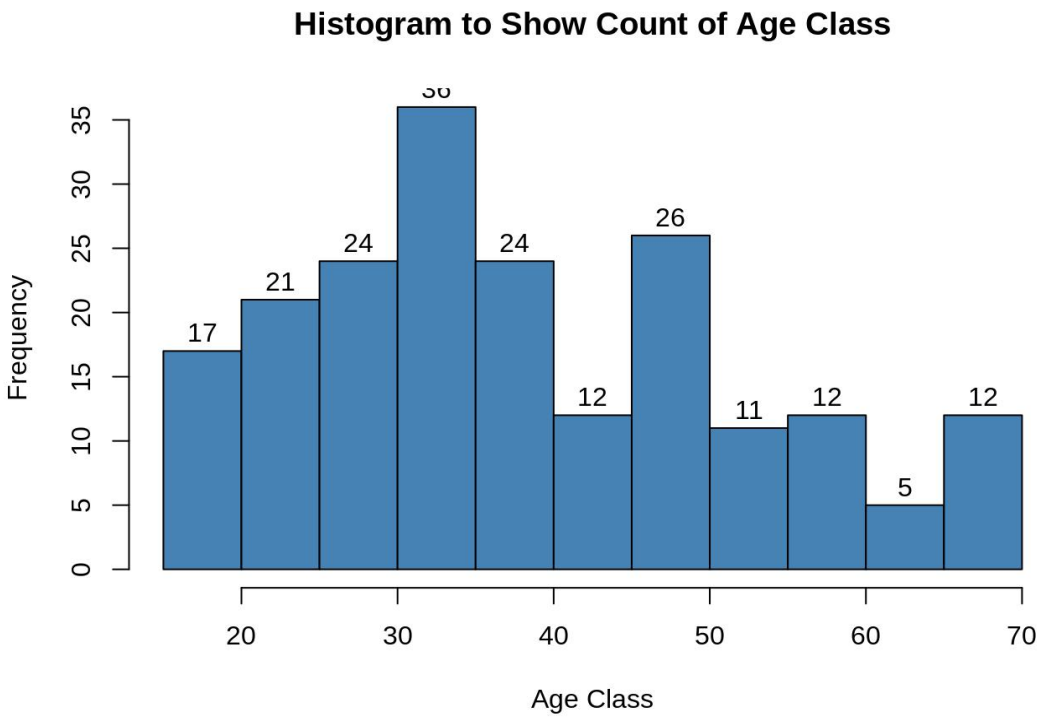
Visualization of Age Distribution

Let us plot a histogram to view the distribution to plot the frequency of customer ages. We will first proceed by taking summary of the Age variable.

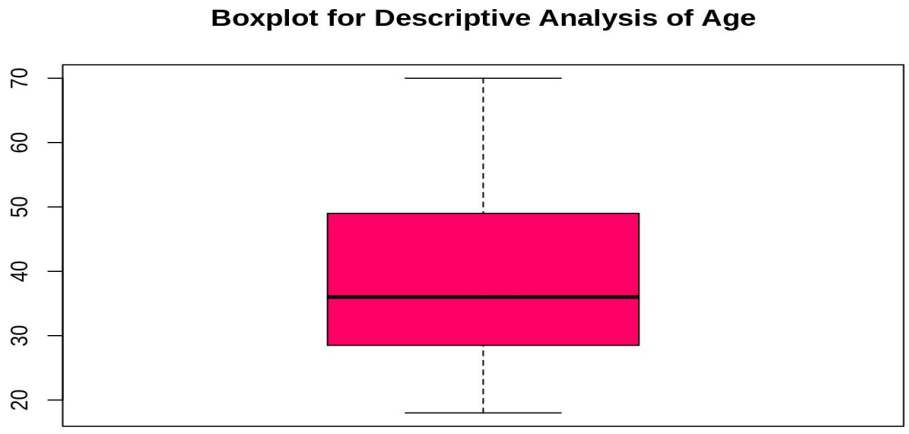
```
summary(customer_data$Age)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   18.00   28.75   36.00   38.85   49.00   70.00
```

```
hist(customer_data$Age,
     col="blue",
     main="Histogram to Show Count of Age Class",
     xlab="Age Class",
     ylab="Frequency",
     labels=TRUE)
```




```
boxplot(customer_data$Age,
        col="#ff0066",
        main="Boxplot for Descriptive Analysis of Age")
```



From the above two visualizations, we conclude that the maximum customer ages are between 30 and 35. The minimum age of customers is 18, whereas, the maximum age is 70.

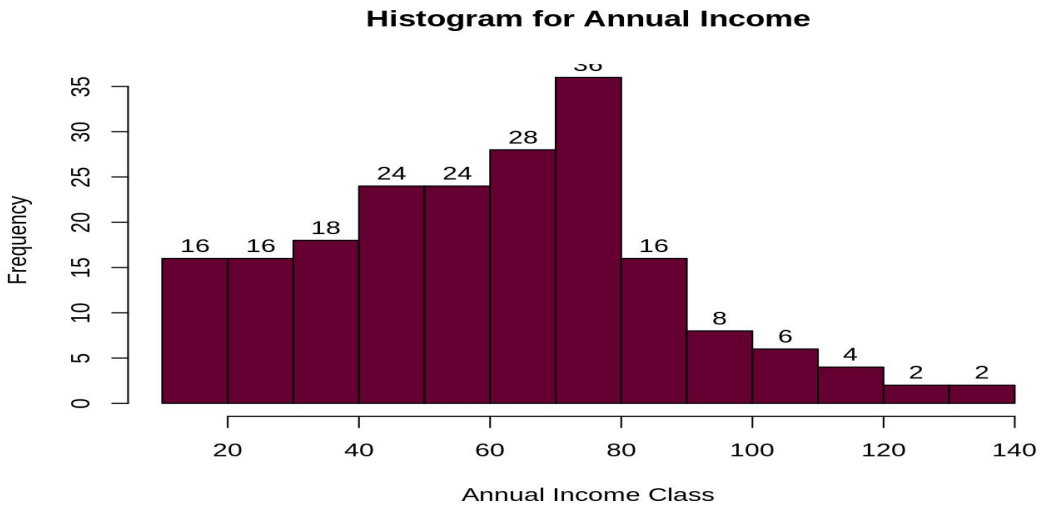
Analysis of the Annual Income of the Customers

In this section of the R project, we will create visualizations to analyze the annual income of the customers. We will plot a histogram and then we will proceed to examine this data using a density plot.

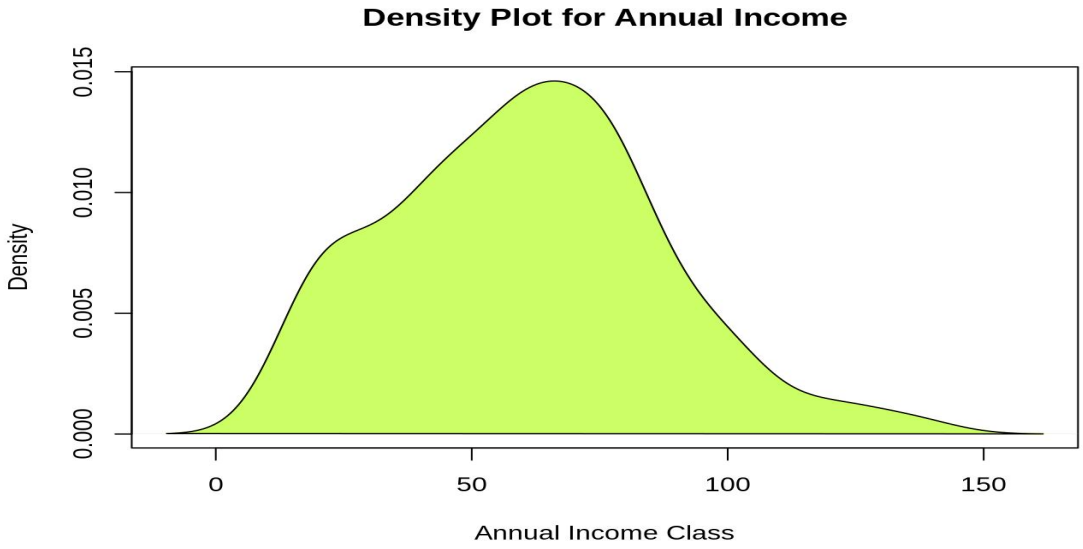
```
summary(customer_data$Annual.Income..k..)
```

```
##     Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    15.00   41.50   61.50   60.56   78.00  137.00
```

```
hist(customer_data$Annual.Income..k..,
     col="#660033",
     main="Histogram for Annual Income",
     xlab="Annual Income Class",
     ylab="Frequency",
     labels=TRUE)
```




```
plot(density(customer_data$Annual.Income..k..),
     col="yellow",
     main="Density Plot for Annual Income",
     xlab="Annual Income Class",
     ylab="Density")
polygon(density(customer_data$Annual.Income..k..),
        col="#ccff66")
```



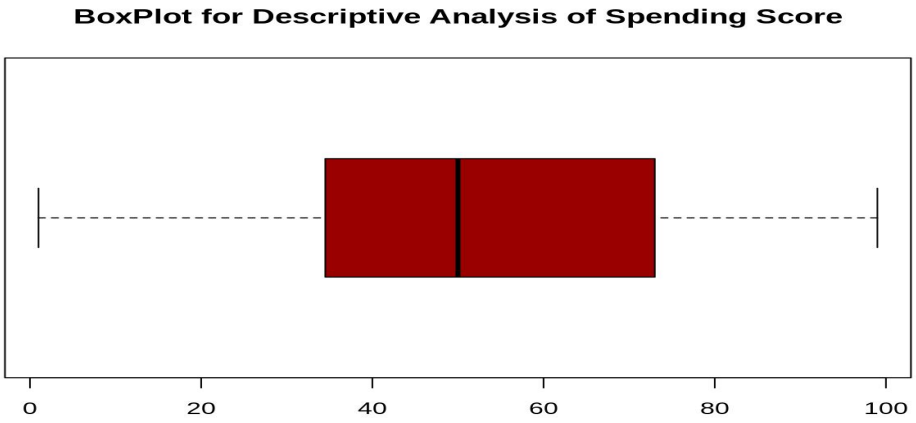
From the above descriptive analysis, we conclude that the minimum annual income of the customers is 15 and the maximum income is 137. People earning an average income of 70 have the highest frequency count in our histogram distribution. The average salary of all the customers is 60.56. In the Kernel Density Plot that we displayed above, we observe that the annual income has a *normal distribution*.

Analyzing Spending Score of the Customers

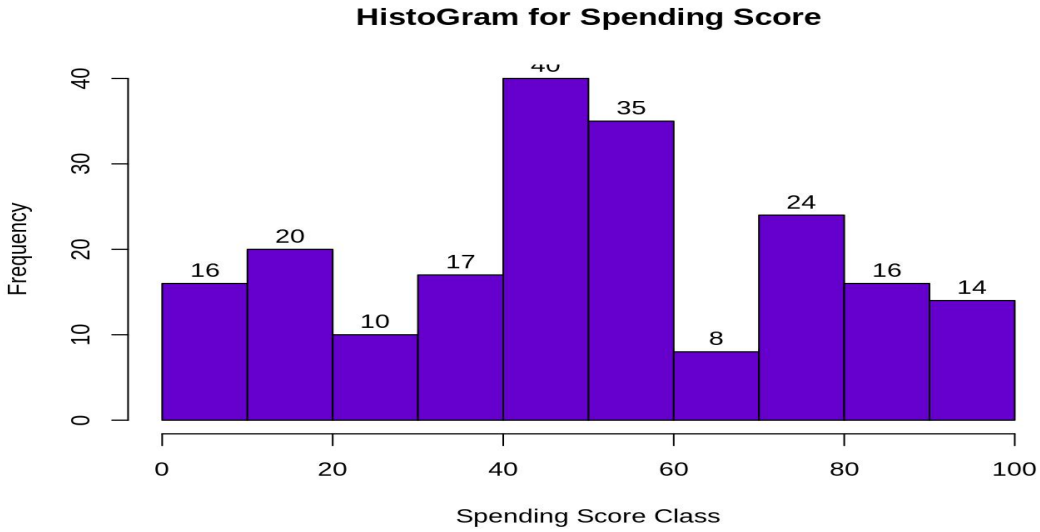
```
summary(customer_data$Spending.Score..1.100.)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00   34.75   50.00   50.20   73.00   99.00
```

```
boxplot(customer_data$Spending.Score..1.100.,
        horizontal=TRUE,
        col="#990000",
        main="BoxPlot for Descriptive Analysis of Spending Score")
```




```
hist(customer_data$Spending.Score..1.100.,
     main="HistoGram for Spending Score",
     xlab="Spending Score Class",
     ylab="Frequency",
     col="#6600cc",
     labels=TRUE)
```



The minimum spending score is 1, maximum is 99 and the average is 50.20. We can see Descriptive Analysis of Spending Score is that Min is 1, Max is 99 and avg. is 50.20. From

the histogram, we conclude that customers between class 40 and 50 have the highest spending score among all the classes.

Algorithm

Usage of K-Means Algorithm in Customer Segmentation

We'll be using K-Means Clustering in R to segment customers into distinct groups based on purchasing habits. K-Means Clustering is an unsupervised learning technique, which means we don't need to have a target for clustering. All we need is to format the data in a way the algorithm can process, and we'll let it determine the customer segments or clusters

While using the k-means clustering algorithm, the first step is to indicate the number of clusters (k) that we wish to produce in the final output. The algorithm starts by selecting k objects from dataset randomly that will serve as the initial centers for our clusters. These selected objects are the cluster means, also known as centroids. Then, the remaining objects have an assignment of the closest centroid. This centroid is defined by the Euclidean Distance present between the object and the cluster mean. We refer to this step as "cluster assignment". When the assignment is complete, the algorithm proceeds to calculate new mean value of each cluster present in the data. After the recalculation of the centers, the observations are checked if they are closer to a different cluster. Using the updated cluster mean, the objects undergo reassignment. This goes on repeatedly through several iterations until the cluster assignments stop altering. The clusters that are present in the current iteration are the same as the ones obtained in the previous iteration.

Summing up the K-means clustering –

- We specify the number of clusters that we need to create.
- The algorithm selects k objects at random from the dataset. This object is the initial cluster or mean.
- The closest centroid obtains the assignment of a new observation. We base this assignment on the Euclidean Distance between object and the centroid.
- k clusters in the data points update the centroid through calculation of the new mean values present in all the data points of the cluster. The kth cluster's centroid has a length of p that contains means of all variables for observations in the k-th cluster. We denote the number of variables with p.
- Iterative minimization of the total within the sum of squares. Then through the iterative minimization of the total sum of the square, the assignment stop wavering when we achieve maximum iteration. The default value is 10 that the R software uses for the maximum iterations.

Determining Optimal Clusters

While working with clusters, we need to specify the number of clusters to use. We would like to utilize the optimal number of clusters. To help us in determining the optimal clusters, there are three popular methods –

- Elbow method
- Silhouette method
- Gap statistic

Elbow Method

The main goal behind cluster partitioning methods like k-means is to define the clusters such that the intra-cluster variation stays minimum.

minimize(sum W(Ck)), k=1...k

Where Ck represents the kth cluster and W(Ck) denotes the intra-cluster variation. With the measurement of the total intra-cluster variation, one can evaluate the compactness of the clustering boundary. We can then proceed to define the optimal clusters as follows –

First, we calculate the clustering algorithm for several values of k. This can be done by creating a variation within k from 1 to 10 clusters. We then calculate the total intra-cluster sum of square (iss). Then, we proceed to plot iss based on the number of k clusters. This plot denotes the appropriate number of clusters required in our model. In the plot, the location of a bend or a knee is the indication of the optimum number of clusters. Let us implement this in R as follows –

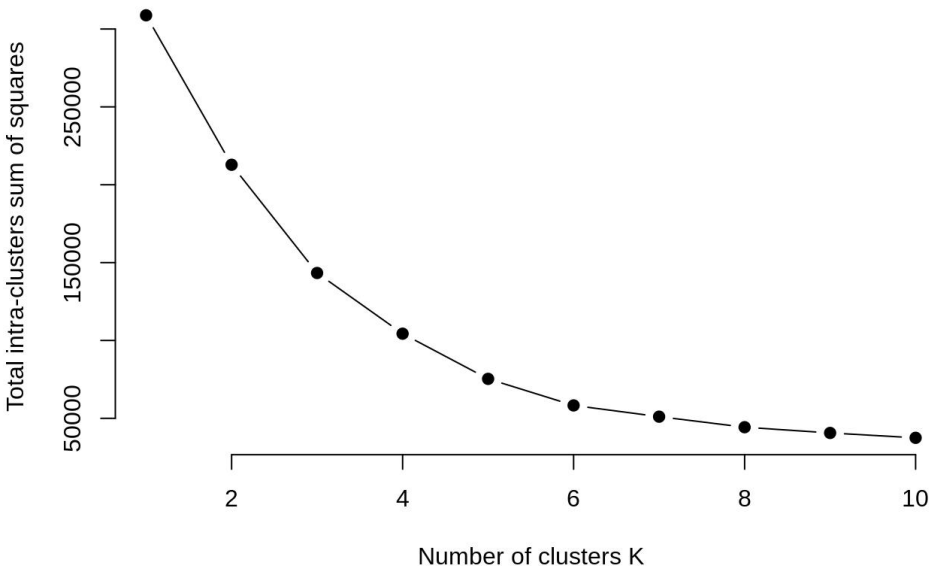
Code:

```
library(purrr)

set.seed(123)

# function to calculate total intra-cluster sum of square

iss <- function(k) {

kmeans(customer_data[,3:5],k,iter.max=100,nstart=100,algorithm="Lloyd" )$tot.within
ss

}

k.values <- 1:10

iss_values <- map_dbl(k.values, iss)

plot(k.values, iss_values,

type="b", pch = 19, frame = FALSE,

xlab="Number of clusters K",

ylab="Total intra-clusters sum of squares")
```

Screenshot:

```
library(purrr)
set.seed(123)
# function to calculate total intra-cluster sum of square
iss <- function(k) {
  kmeans(customer_data[,3:5],k,iter.max=100,nstart=100,algorithm="Lloyd" )$tot.withins
s
}

k.values <- 1:10


iss_values <- map_dbl(k.values, iss)

plot(k.values, iss_values,
     type="b", pch = 19, frame = FALSE,
     xlab="Number of clusters K",
     ylab="Total intra-clusters sum of squares")
```

Output:



But the issue with this method is that the elbow is never prominent in real world situations as in the above figure. And for us as well it got confusing. Is it 2, 4, 5, or 6, that we should take as the optimal number of clusters here?

Average Silhouette Method

With the help of the average silhouette method, we can measure the quality of our clustering operation. With this, we can determine how well within the cluster is the data object. If we obtain a high average silhouette width, it means that we have good clustering. The average silhouette method calculates the mean of silhouette observations for different k values. With the optimal number of k clusters, one can maximize the average silhouette over significant values for k clusters.

Using the silhouette function in the cluster package, we can compute the average silhouette width using the kmean function. Here, the optimal cluster will possess highest average.

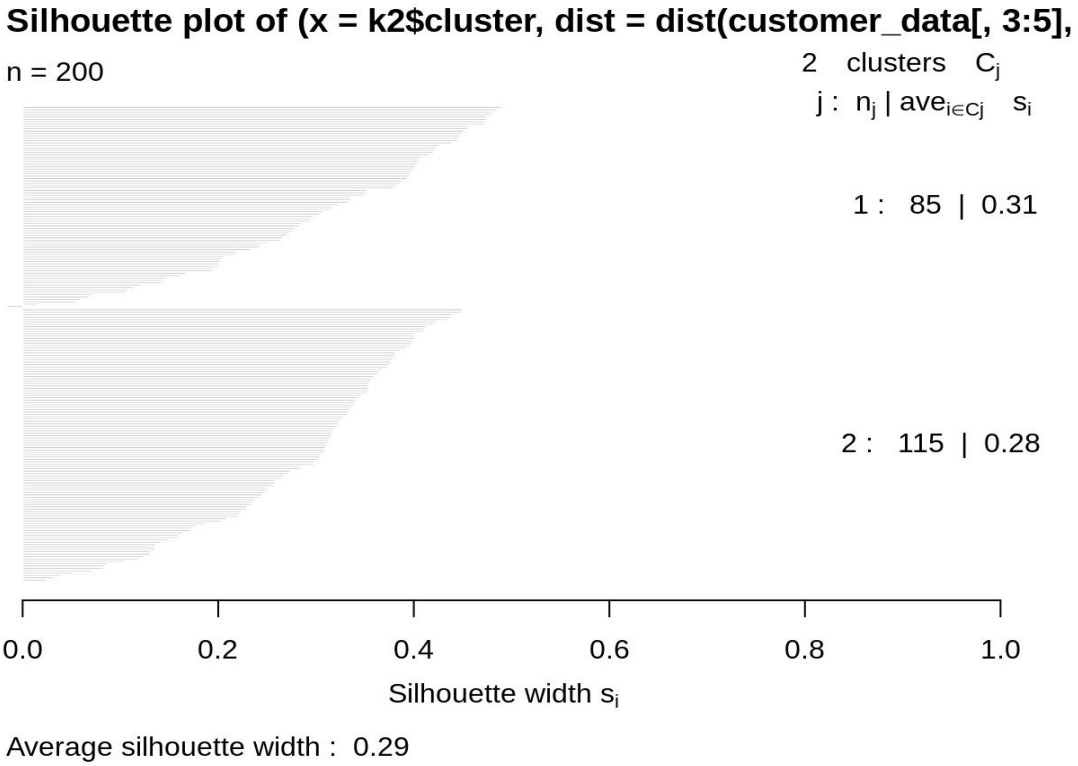
Code:

```
1.library(cluster)

2.library(gridExtra)

3.library(grid)

4.k2<-kmeans(customer_data[,3:5],2,iter.max=100,nstart=50,algorithm="Lloyd")

5.s2<-plot(silhouette(k2$cluster,dist(customer_data[,3:5],"euclidean")))
```

Screenshot:

```
library(cluster)
library(gridExtra)
library(grid)


k2<-kmeans(customer_data[,3:5],2,iter.max=100,nstart=50,algorithm="Lloyd")
s2<-plot(silhouette(k2$cluster,dist(customer_data[,3:5],"euclidean")))
```

Output:



Code:

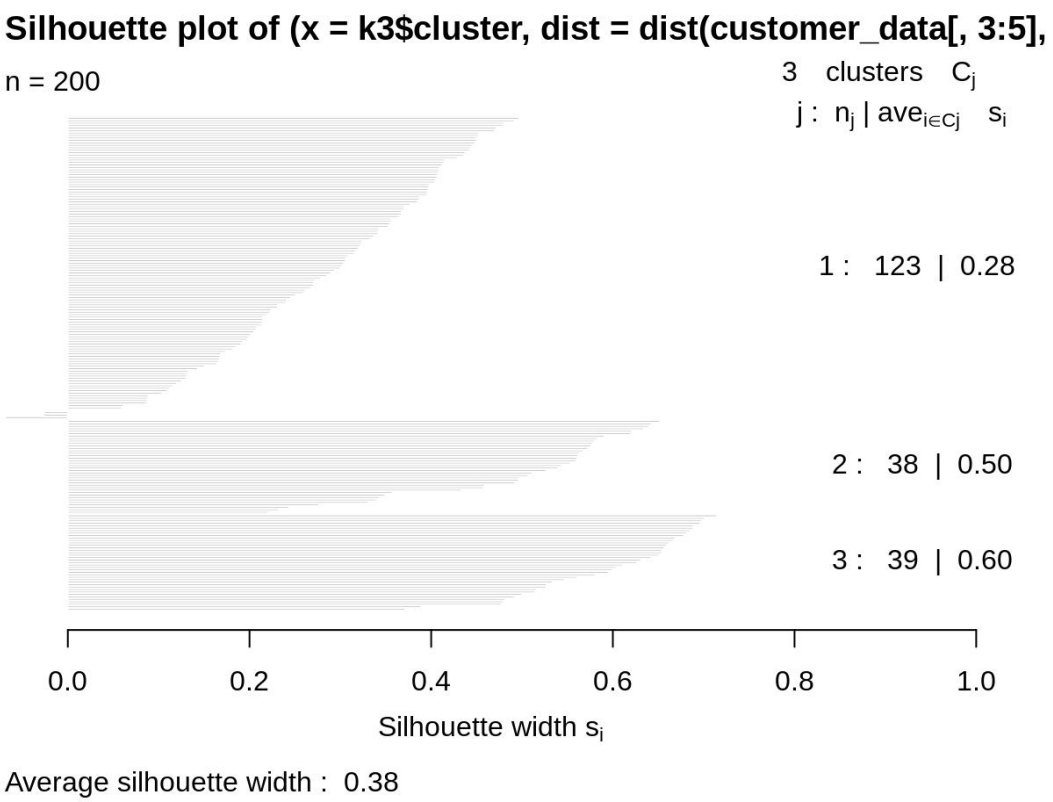
```
1.k3<-kmeans(customer_data[,3:5],3,iter.max=100,nstart=50,algorithm="Lloyd")
```

```
2.s3<-plot(silhouette(k3$cluster,dist(customer_data[,3:5],"euclidean")))
```

Screenshot:

```
k3<-kmeans(customer_data[,3:5],3,iter.max=100,nstart=50,algorithm="Lloyd")
s3<-plot(silhouette(k3$cluster,dist(customer_data[,3:5],"euclidean")))
```

Output:



Code:

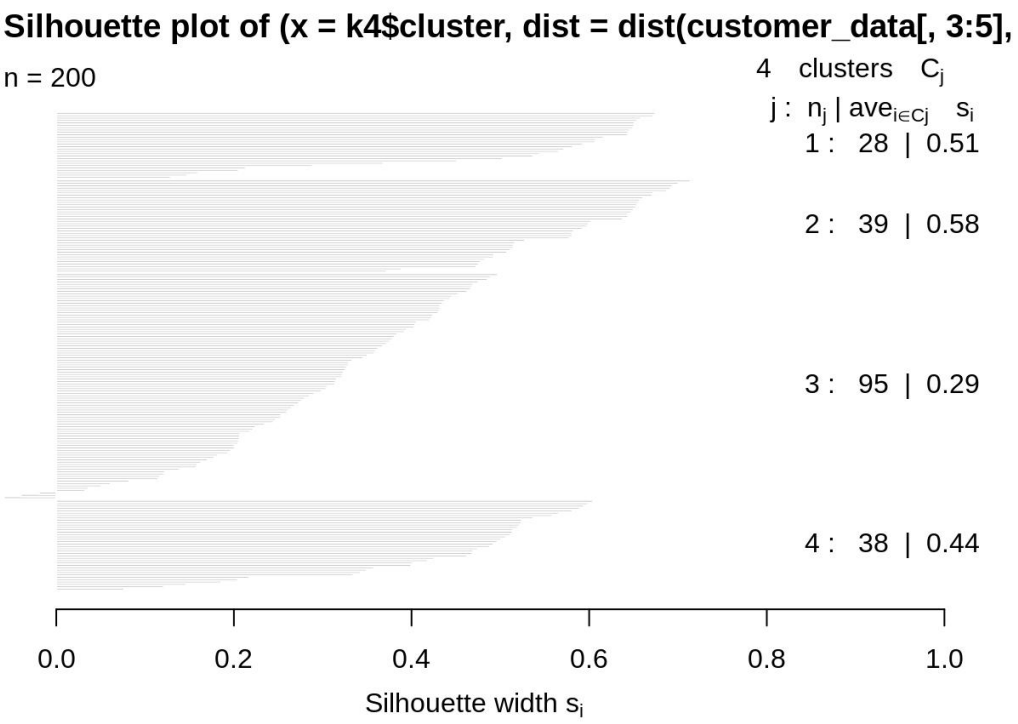
```
k4<-kmeans(customer_data[,3:5],4,iter.max=100,nstart=50,algorithm="Lloyd")
```

```
s4<-plot(silhouette(k4$cluster,dist(customer_data[,3:5],"euclidean"))
```

Screenshot:

```
k4<-kmeans(customer_data[,3:5],4,iter.max=100,nstart=50,algorithm="Lloyd")
s4<-plot(silhouette(k4$cluster,dist(customer_data[,3:5],"euclidean")))
```

Output:



Average silhouette width : 0.41

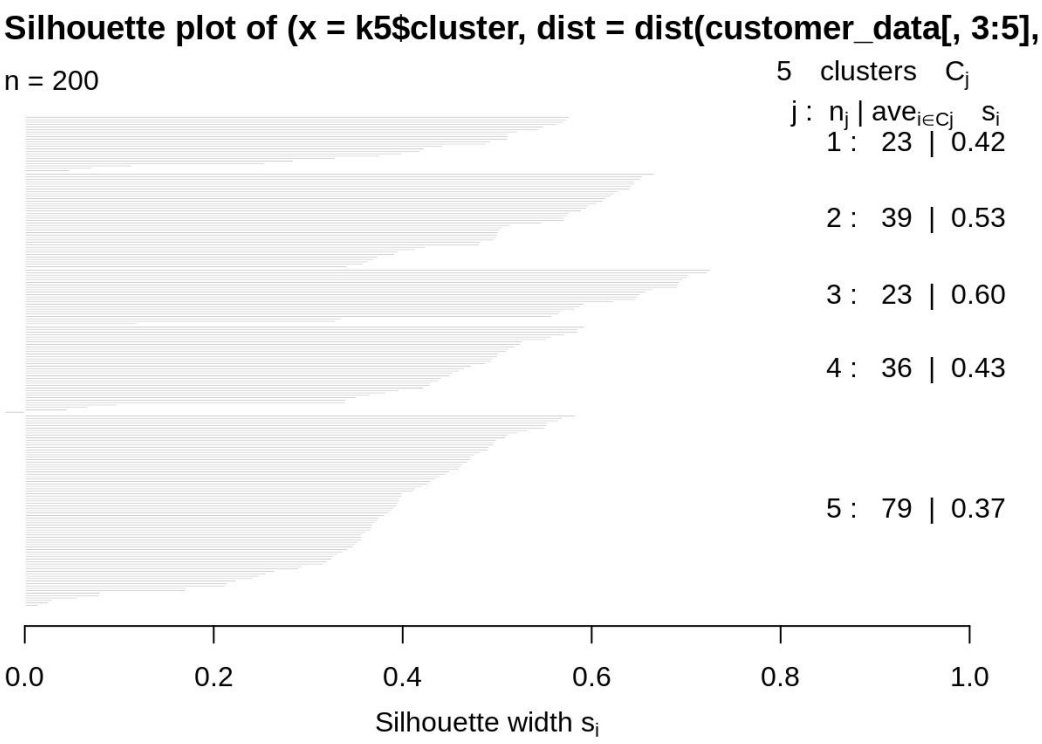
Code:

```
k5<-kmeans(customer_data[,3:5],5,iter.max=100,nstart=50,algorithm="Lloyd")

s5<-plot(silhouette(k5$cluster,dist(customer_data[,3:5],"euclidean")))
```

Screenshot:

```
k5<-kmeans(customer_data[,3:5],5,iter.max=100,nstart=50,algorithm="Lloyd")
s5<-plot(silhouette(k5$cluster,dist(customer_data[,3:5],"euclidean")))
```

Output:



Code:

```
k6<-kmeans(customer_data[,3:5],6,iter.max=100,nstart=50,algorithm="Lloyd")
s6<-plot(silhouette(k6$cluster,dist(customer_data[,3:5],"euclidean")))
```

Screenshot:

```
k6<-kmeans(customer_data[,3:5],6,iter.max=100,nstart=50,algorithm="Lloyd")
s6<-plot(silhouette(k6$cluster,dist(customer_data[,3:5],"euclidean")))
```

Output:

Code:

```
k7<-kmeans(customer_data[,3:5],7,iter.max=100,nstart=50,algorithm="Lloyd")
s7<-plot(silhouette(k7$cluster,dist(customer_data[,3:5],"euclidean")))
```

Screenshot:

```
k7<-kmeans(customer_data[,3:5],7,iter.max=100,nstart=50,algorithm="Lloyd")
s7<-plot(silhouette(k7$cluster,dist(customer_data[,3:5],"euclidean")))
```

Output:

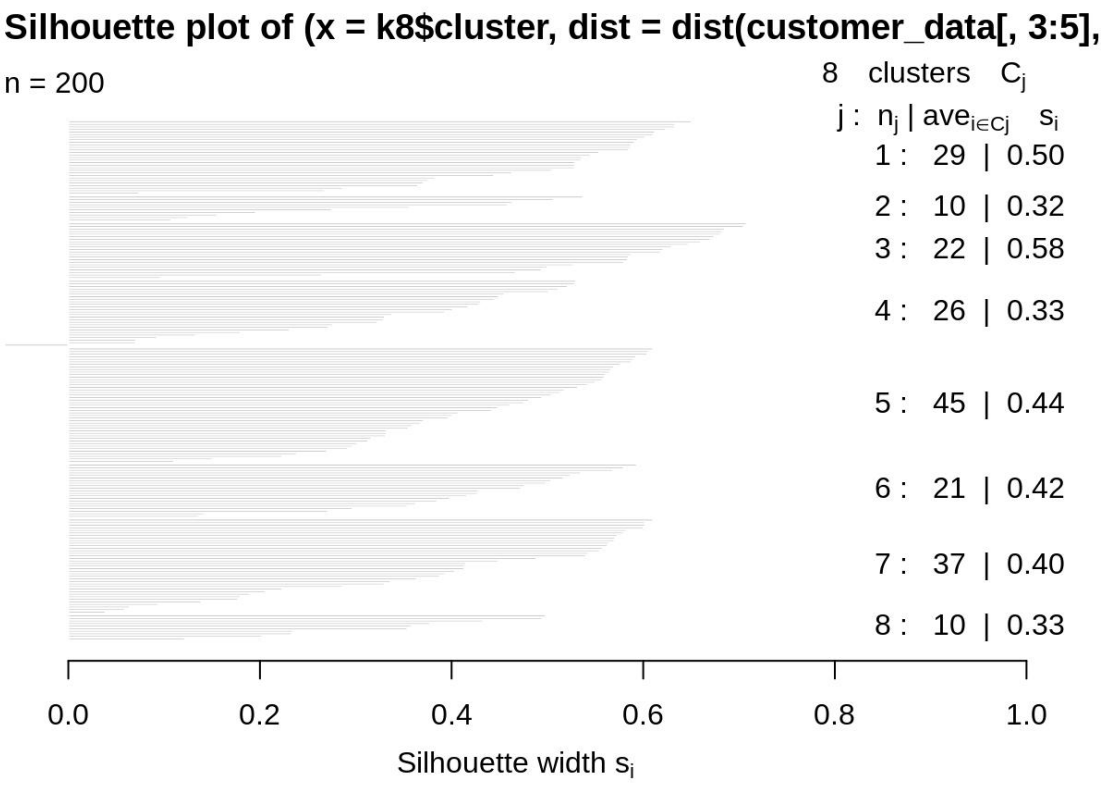
Code:

```
1.k8<-kmeans(customer_data[,3:5],8,iter.max=100,nstart=50,algorithm="Lloyd")

2.s8<-plot(silhouette(k8$cluster,dist(customer_data[,3:5],"euclidean")))
```

Screenshot:

```
k8<-kmeans(customer_data[,3:5],8,iter.max=100,nstart=50,algorithm="Lloyd")
s8<-plot(silhouette(k8$cluster,dist(customer_data[,3:5],"euclidean")))
```

Output:



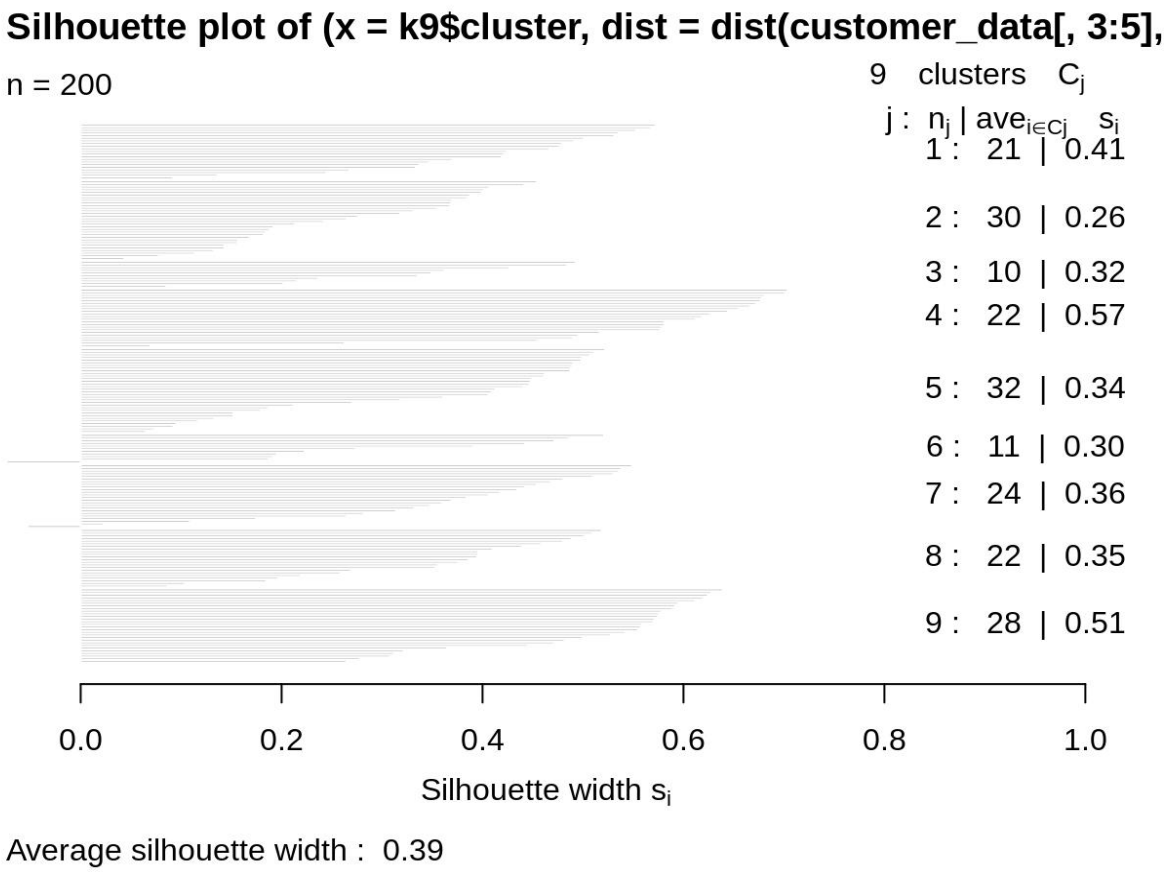
Code:

```
1.k9<-kmeans(customer_data[,3:5],9,iter.max=100,nstart=50,algorithm="Lloyd")
2.s9<-plot(silhouette(k9$cluster,dist(customer_data[,3:5],"euclidean")))
```

Screenshot:

```
k9<-kmeans(customer_data[,3:5],9,iter.max=100,nstart=50,algorithm="Lloyd")
s9<-plot(silhouette(k9$cluster,dist(customer_data[,3:5],"euclidean")))
```

Output:



Code:

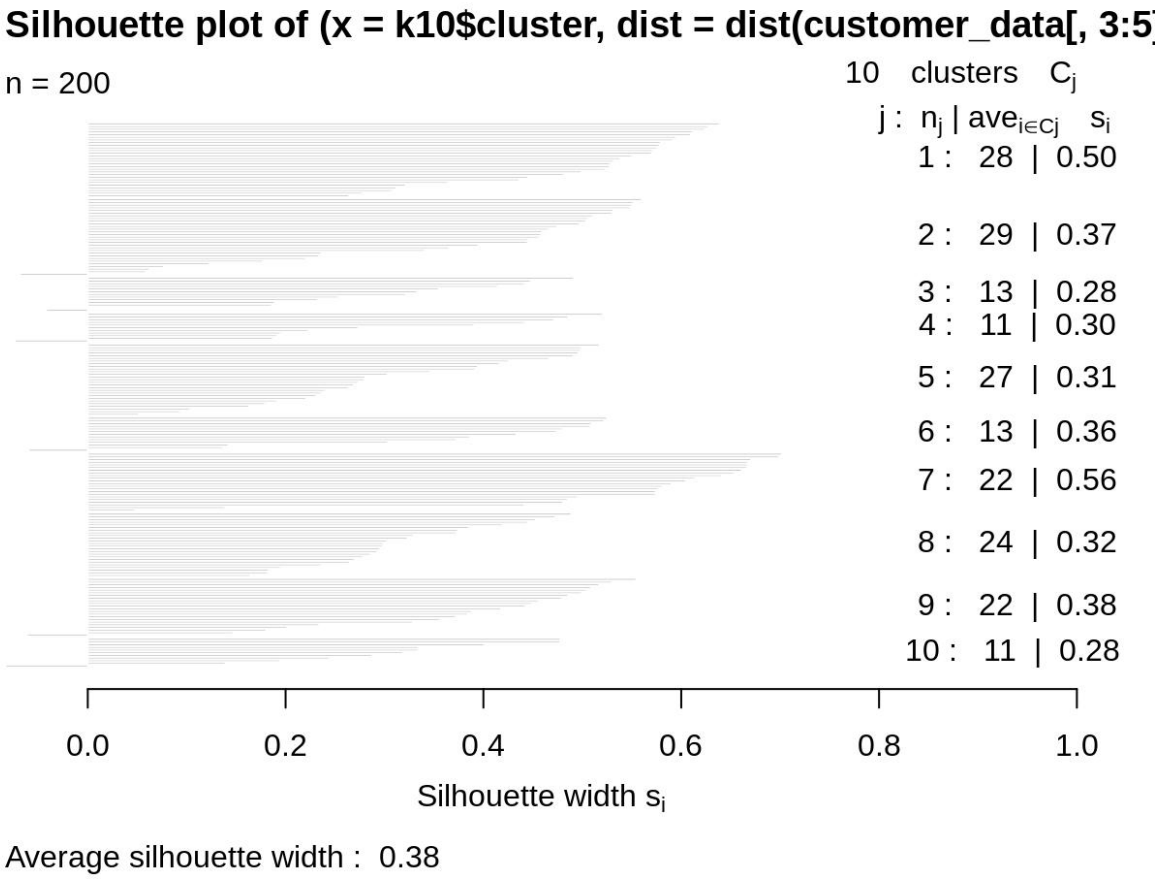
```
k10<-kmeans(customer_data[,3:5],10,iter.max=100,nstart=50,algorithm="Lloyd")
```

```
s10<-plot(silhouette(k10$cluster,dist(customer_data[,3:5],"euclidean")))
```

Screenshot:

```
k10<-kmeans(customer_data[,3:5],10,iter.max=100,nstart=50,algorithm="Lloyd")
s10<-plot(silhouette(k10$cluster,dist(customer_data[,3:5],"euclidean")))
```

Output:



Now, we make use of the fviz_nbclust() function to determine and visualize the optimal number of clusters as follows –

Code:

```
1.library(NbClust)
```

```
2.library(factoextra)
```

```
3.fviz_nbclust(customer_data[,3:5], kmeans, method = "silhouette")
```

Screenshot:

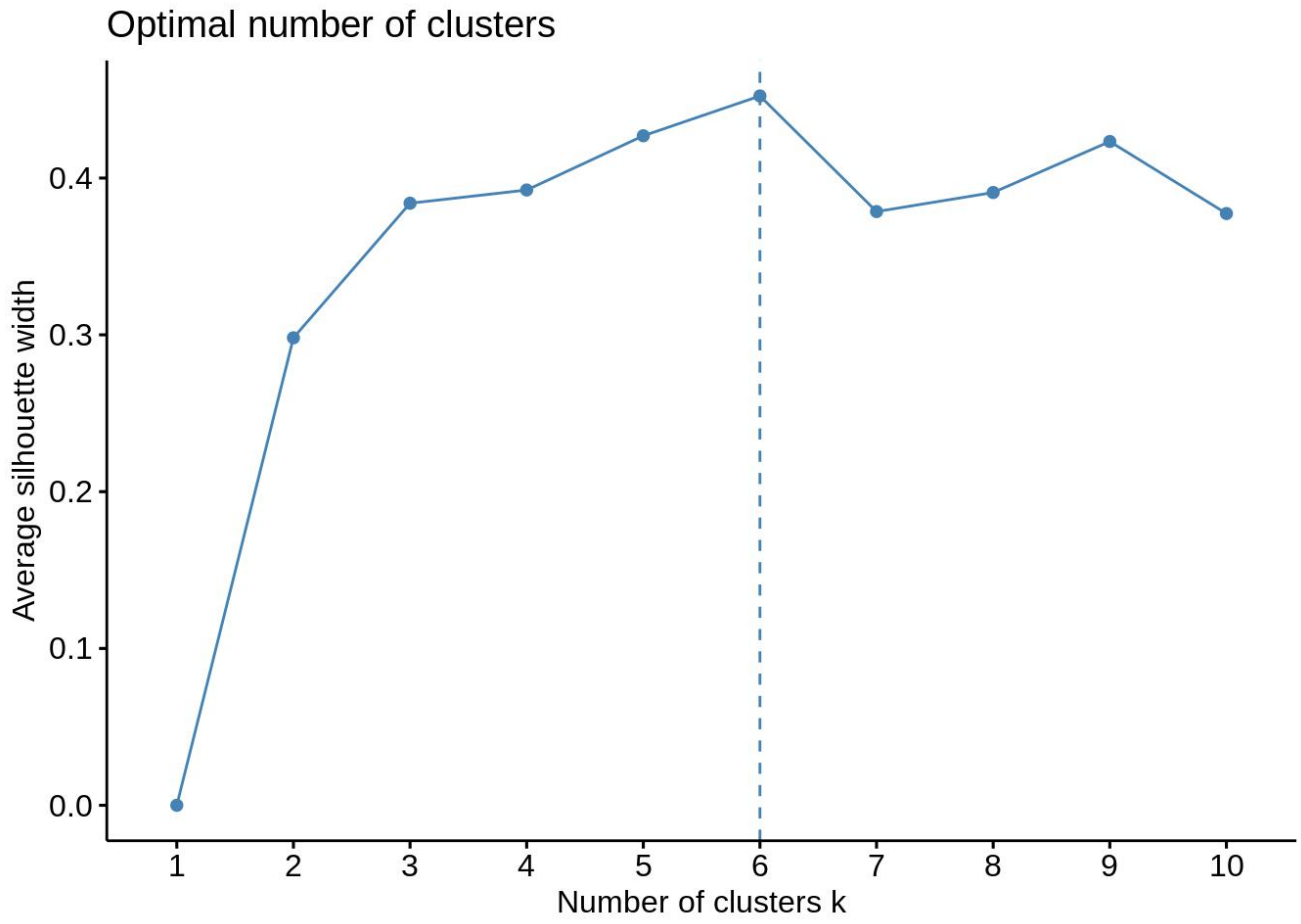
```
library(NbClust)
library(factoextra)
```

```
## Loading required package: ggplot2
```

```
## Welcome! Related Books: `Practical Guide To Cluster Analysis in R` at https://goo.g
l/13EFCZ
```

```
fviz_nbclust(customer_data[,3:5], kmeans, method = "silhouette")
```

Output:



Gap Statistic Method

In 2001, researchers at Stanford University – **R. Tibshirani, G.Walther and T. Hastie** published the Gap Statistic Method. We can use this method to any of the clustering method like K-means, hierarchical clustering etc. Using the gap statistic, one can compare the total intracluster variation for different values of k along with their expected values under the null reference distribution of data. With the help of **Monte Carlo simulations**, one can produce the sample dataset. For each variable in the dataset, we can calculate the range between min(xi) and max (xj) through which we can produce values uniformly from interval lower bound to upper bound.

For computing the gap statistics method we can utilize the clusGap function for providing gap statistic as well as standard error for a given output.

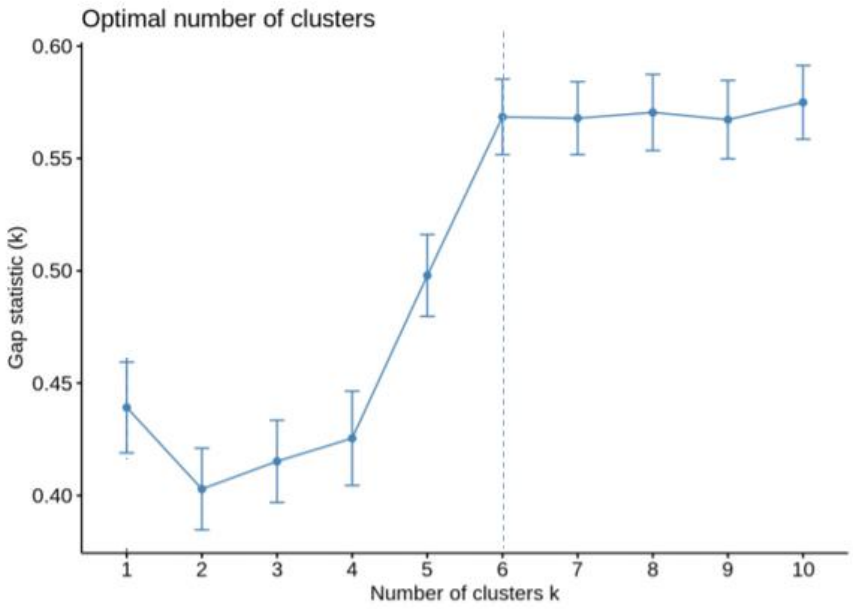
Code:

```
1.set.seed(125)
2.stat_gap <- clusGap(customer_data[,3:5], FUN = kmeans, nstart = 25,
3.K.max = 10, B = 50)
4.fviz_gap_stat(stat_gap)
```

Screenshot:

```
# compute gap statistic
set.seed(123)
gap_stat <- clusGap(customer_data[,3:5], FUN = kmeans, nstart = 25,
                    K.max = 10, B = 50)
fviz_gap_stat(gap_stat)
```

Output:



Now, let us take k = 6 as our optimal cluster –

Code:

```
1.k6<-kmeans(customer_data[,3:5],6,iter.max=100,nstart=50,algorithm="Lloyd")
2.k6
```

Output Screenshot

```
# compute gap statistic
k6<-kmeans(customer_data[,3:5],6,iter.max=100,nstart=50,algorithm="Lloyd")
k6
```

```
## K-means clustering with 6 clusters of sizes 45, 22, 21, 38, 35, 39
##
## Cluster means:
##        Age Annual.Income..k.. Spending.Score..1.100.
## 1 56.15556           53.37778                49.08889
## 2 25.27273           25.72727                79.36364
## 3 44.14286           25.14286                19.52381
## 4 27.00000           56.65789                49.13158
## 5 41.68571           88.22857                17.28571
## 6 32.69231           86.53846                82.12821
##
## Clustering vector:
##   [1] 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3
##  [36] 2 3 2 3 2 1 2 1 4 3 2 1 4 4 4 1 4 4 1 1 1 1 1 4 1 1 4 1 1 1 4 1 1 4 4
##  [71] 1 1 1 1 1 4 1 4 4 1 1 4 1 1 4 1 1 4 4 1 1 4 1 4 4 4 1 4 1 4 4 1 1 4 1
```

In the output of our kmeans operation, we observe a list with several key information. From this, we conclude the useful information being –

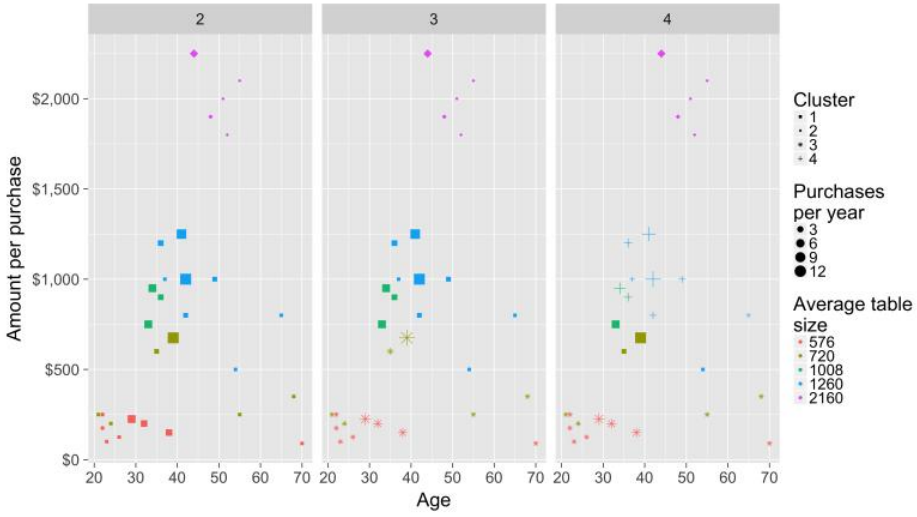
- **cluster –** This is a vector of several integers that denote the cluster which has an allocation of each point.

- **totss –** This represents the total sum of squares.
- **centers –** Matrix comprising of several cluster centers
- **withinss –** This is a vector representing the intra-cluster sum of squares having one component per cluster.
- **tot.withinss –** This denotes the total intra-cluster sum of squares.
- **betweenss –** This is the sum of between-cluster squares.
- **size –** The total number of points that each cluster holds.

Results and Analysis

K means clustering

The algorithm starts by choosing “k” points as the initial central values (often called centroids) [1]. Next, every point in the data is assigned to the central value it is closest to. Now every point is assigned a cluster, but we need to check if the initial guesses of central values are the best ones (very unlikely!). The way to check that is to compute the new central value of each cluster — if all of the recomputed central values are the same as the original ones, then you are at the best solution and the algorithm can stop. Otherwise, the algorithm tries again by reassigning points to the newly computed central values. This will continue until the recomputed central values don’t change.



In all cases, the buyers of the 2160 cm² tables are in their own cluster, but the rest of the customers are a little more co-mingled depending on their characteristics. When k is equal to 2, the clusters look reasonable, but there is likely some more granularity that could be differentiated for the customers buying smaller tables. When k is equal to 3 and 4, these customers get split up into smaller segments.

Elbow and Silhouette are direct methods ;

While using the k-means clustering algorithm, the first step is to indicate the number of clusters (k) that we wish to produce in the final output.

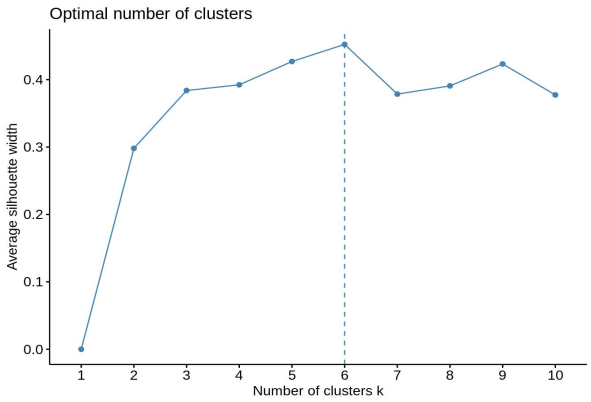
OUR EXAMPLE:

```
88
89  # Determining Optimal Clusters
90
91  # Using Elbow Method
92  library(purrr)
93  set.seed(123)
94  # function to calculate total intra-cluster sum of square
95  iss <- function(k) {
96    kmeans(customer_data[,3:5],k,iter.max=100,nstart=100,algorithm="Lloyd" )
97  }
98  k.values <- 1:10
99  iss_values <- map_dbl(k.values, iss)
100 plot(k.values, iss_values,
101      type="b", pch = 19, frame = FALSE,
102      xlab="Number of clusters K",
103      ylab="Total intra-clusters sum of squares")
104
```

The elbow method looks at the percentage of variance explained as a function of the number of clusters: One should choose a number of clusters so that adding another cluster doesn't give much better modeling of the data. More precisely, if one plots the percentage of variance explained by the clusters against the number of clusters, the first clusters will add much information (explain a lot of variance), but at some point the marginal gain will drop, giving an angle in the graph. The number of clusters is chosen at this point, hence the "elbow criterion". This "elbow" cannot always be unambiguously identified.

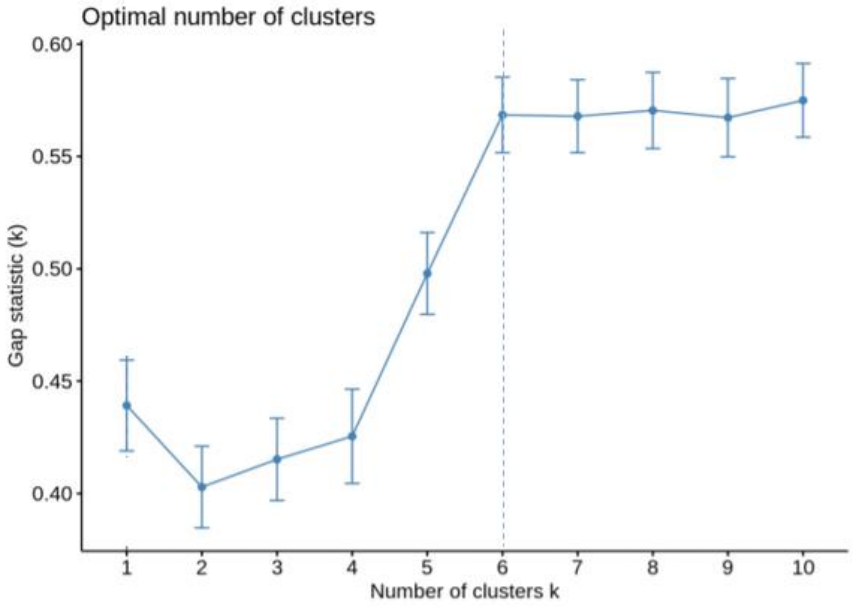
Average Silhouette Method:

```
104
105 # Using Average Silhouette Method
106 library(cluster)
107 library(gridExtra)
108 library(grid)
109 k2<-kmeans(customer_data[,3:5],2,iter.max=100,nstart=50,algorithm="Lloyd")
110 s2<-plot(silhouette(k2$cluster,dist(customer_data[,3:5],"euclidean")))
111 k3<-kmeans(customer_data[,3:5],3,iter.max=100,nstart=50,algorithm="Lloyd")
112 s3<-plot(silhouette(k3$cluster,dist(customer_data[,3:5],"euclidean")))
113 k4<-kmeans(customer_data[,3:5],4,iter.max=100,nstart=50,algorithm="Lloyd")
114 s4<-plot(silhouette(k4$cluster,dist(customer_data[,3:5],"euclidean")))
115 k5<-kmeans(customer_data[,3:5],5,iter.max=100,nstart=50,algorithm="Lloyd")
116 s5<-plot(silhouette(k5$cluster,dist(customer_data[,3:5],"euclidean")))
117 k6<-kmeans(customer_data[,3:5],6,iter.max=100,nstart=50,algorithm="Lloyd")
118 s6<-plot(silhouette(k6$cluster,dist(customer_data[,3:5],"euclidean")))
119 k7<-kmeans(customer_data[,3:5],7,iter.max=100,nstart=50,algorithm="Lloyd")
120 s7<-plot(silhouette(k7$cluster,dist(customer_data[,3:5],"euclidean")))
121 k8<-kmeans(customer_data[,3:5],8,iter.max=100,nstart=50,algorithm="Lloyd")
122 s8<-plot(silhouette(k8$cluster,dist(customer_data[,3:5],"euclidean")))
123 k9<-kmeans(customer_data[,3:5],9,iter.max=100,nstart=50,algorithm="Lloyd")
124 s9<-plot(silhouette(k9$cluster,dist(customer_data[,3:5],"euclidean")))
125 k10<-kmeans(customer_data[,3:5],10,iter.max=100,nstart=50,algorithm="Lloyd
126 s10<-plot(silhouette(k10$cluster,dist(customer_data[,3:5],"euclidean")))
127
128 library(NbClust)
129 library(factoextra)
130 fviz_nbclust(customer_data[,3:5], kmeans, method = "silhouette")
131
```



The above method of calculating silhouette score using `silhouette()` and plotting the results states that optimal number of clusters as 6

Gap Statistics:



The **Gap statistic** is a standard method for determining the number of clusters in a set of data. The **Gap statistic** standardizes the graph of log(Wk), where Wk is the within-cluster dispersion, by comparing it to its expectation under an appropriate null reference distribution of the data.

This information is contained in the following formula for the gap statistic:

$$/displaystyle + /mathrmGap_n(k) + = + E_n^* / / log + W_k / + - + / log + W_k$$

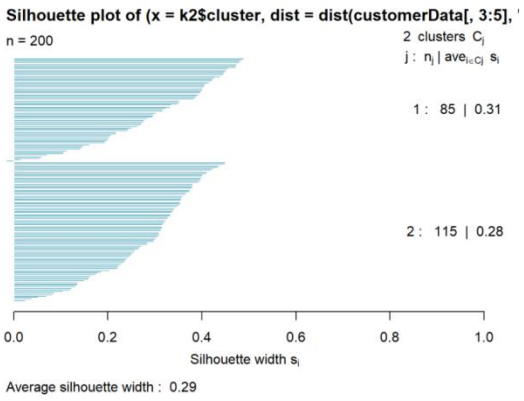
The reference datasets are in our case generated by sampling uniformly from the original dataset's bounding box (see green box in the upper right plot of the figures below). To obtain the estimate $E_n^* / / log + W_k /$ we compute the average of B copies $/log + W_k^*$ for $B = 10$, each of which is generated with a Monte Carlo sample from the reference distribution. Those $/log + W_k^*$ from the B Monte Carlo replicates exhibit a standard deviation $/mathrmsd(k)$ which, accounting for the simulation error, is turned into the quantity

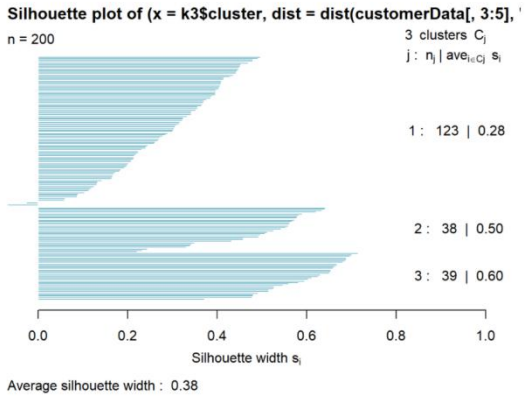
$$/displaystyle + s_k + = + / sqrt 1 + 1 / B / . / mathrmsd(k)$$

Finally, the optimal number of clusters K is the smallest k such that $/mathrmGap(k) + / geq + / mathrmGap(k+1) + - + s_{k+1}$.

Our results:

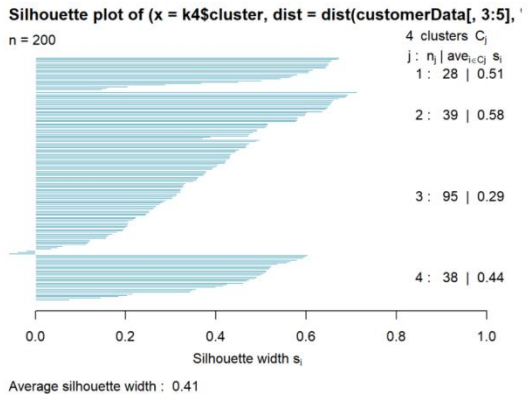
```
library(grid)
k2 <- kmeans(customerData[,3:5],2,iter.max = 100,nstart = 50,algorithm = "Lloyd")
s2 <- plot(silhouette(k2$cluster,dist(customerData[,3:5],"euclidean")),col = "#1287A5")
```






```
k4 <- kmeans(customerData[,3:5],4,iter.max = 100,nstart = 50,algorithm = "Lloyd")
s4 <- plot(silhouette(k4$cluster,dist(customerData[,3:5],"euclidean")),col="#1287A5")
```


```
k4 <- kmeans(customerData[,3:5],4,iter.max = 100,nstart = 50,algorithm = "Lloyd")
s4 <- plot(silhouette(k4$cluster,dist(customerData[,3:5],"euclidean")),col="#1287A5")
```

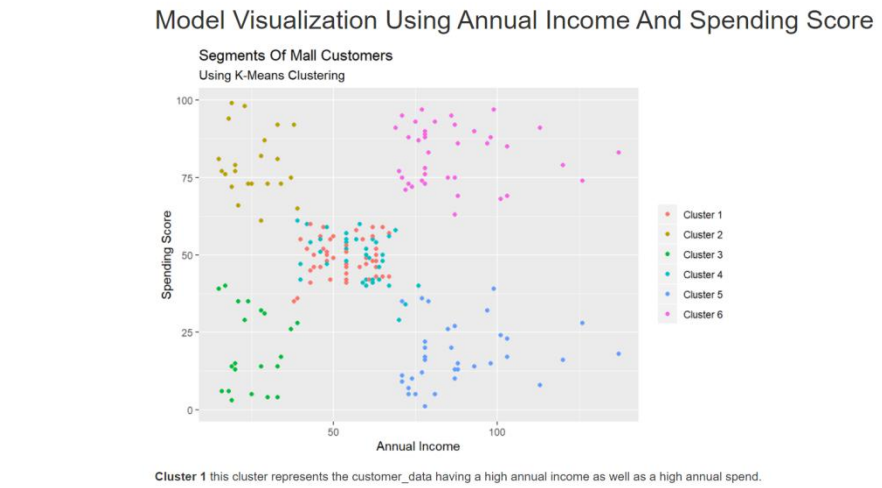



And this continues till silhouette width is 0.38.

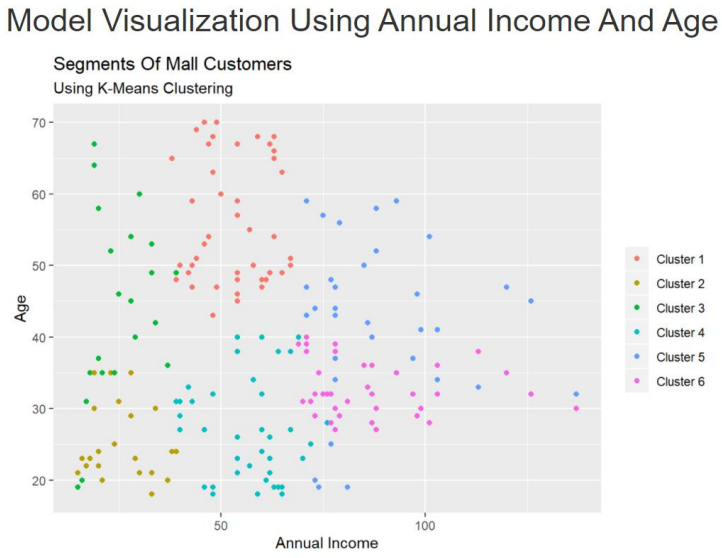
Visualizing the Clustering Results using the First Two Principle Components

Our result:

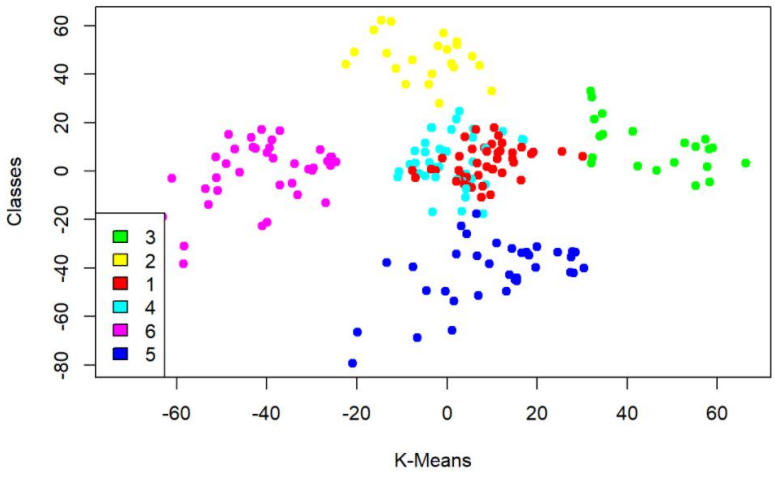
```
140
141 # Visualizing the Clustering Results using the First Two Principle Compone
142 pcclust=prcomp(customer_data[,3:5],scale=FALSE) #principal component analy
143 summary(pcclust)
144 pcclust$rotation[,1:2]
145
146 set.seed(1)
147 ggplot(customer_data, aes(x =Annual.Income..k.., y = Spending.Score..1.100
148   geom_point(stat = "identity", aes(color = as.factor(k6$cluster))) +
149   scale_color_discrete(name=" ",
150                        breaks=c("1", "2", "3", "4", "5","6"),
151                        labels=c("Cluster 1", "Cluster 2", "Cluster 3", "Cl
152   ggtitle("Segments of Mall Customers", subtitle = "Using K-means Clusteri
153
154 ggplot(customer_data, aes(x =Spending.Score..1.100., y =Age)) +
155   geom_point(stat = "identity", aes(color = as.factor(k6$cluster))) +
156   scale_color_discrete(name=" ",
157                        breaks=c("1", "2", "3", "4", "5","6"),
158                        labels=c("Cluster 1", "Cluster 2", "Cluster 3", "Cl
159   ggtitle("Segments of Mall Customers", subtitle = "Using K-means Clusteri
160 kCols=function(vec){cols=rainbow (length (unique (vec)))
161 return (cols[as.numeric(as.factor(vec))])}
162 digCluster<-k6$cluster; dignm<-as.character(digCluster); # K-means cluster
163 plot(pcclust$x[,1:2], col =kCols(digCluster),pch =19,xlab ="K-means",ylab=
164 legend("bottomleft",unique(dignm),fill=unique(kCols(digCluster)))
165
```



Cluster 1 this cluster represents the customer_data having a high annual income as well as a high annual spend.



COCLUDING CLUSTERAL DIVISION:



- **Cluster 4 and 1** These two clusters consist of customers with medium PCA1 and medium PCA2 score.
- **Cluster 6** this cluster represents customers having a high PCA2 and a low PCA1.
- **Cluster 5** In this cluster, there are customers with a medium PCA1 and a low PCA2 score.
- **Cluster 3** This cluster comprises of customers with a high PCA1 income and a high PCA2.
- **Cluster 2** This comprises of customers with a high PCA2 and a medium annual spend of income.

CONCLUSION

With the help of clustering, we can understand the variables much better, prompting us to take careful decisions. With the identification of customers, companies can release products and services that target customers based on several parameters like income, age, spending patterns, etc. Furthermore, more complex patterns like product reviews are taken into consideration for better segmentation.

In this data science project, we went through the customer segmentation model. We developed this using a class of machine learning known as unsupervised learning. Specifically, we made use of a clustering algorithm called K-means clustering. We analysed and visualized the data and then proceeded to implement our algorithm.