Clustering

The data set used in this assignment can be accessed ***here (https://www.kaggle.com/datasets/aryashah2k/credit-card-customer-data)***

This data set includes information regarding customer credit card data. In this part of the assignment, I focused on the ‘Avg_Credit_Limit’ and ‘Total_Credit_Cards’ for the clustering, with ‘Total_visits_bank’ being the target.

KMeans Clustering

First, I performed KMeans Clustering on the data set. This section reads in the data from the csv file

```
df <- read.csv('/Users/kellytrinh/Desktop/school/Similarity and Ensemble/Credit Card Cus
tomer Data.csv', na.strings="NA", header=TRUE)
```

Data Cleaning and Exploration

I then did some data exploration, along with cleaning up the data by removing any NAs if they existed.

```
# Data exploration?
names(df)
```

```
## [1] "Sl_No"               "Customer.Key"        "Avg_Credit_Limit"
## [4] "Total_Credit_Cards"  "Total_visits_bank"   "Total_visits_online"
## [7] "Total_calls_made"
```

```
head(df)
```

	Sl_... <int>	Customer.Key <int>	Avg_Credit_Limit <int>	Total_Credit_Cards <int>	Total_visits_bank <int>	Total_
1	1	87073	100000	2	1	
2	2	38414	50000	3	0	
3	3	17341	50000	7	1	
4	4	40496	30000	5	1	
5	5	47437	100000	6	0	
6	6	58634	20000	3	0	

6 rows | 1-7 of 8 columns

```
summary(df)
```

```
##      Sl_No        Customer.Key   Avg_Credit_Limit Total_Credit_Cards
##  Min.   :  1.0   Min.   :11265   Min.   :  3000   Min.   : 1.000
##  1st Qu.:165.8   1st Qu.:33825   1st Qu.: 10000   1st Qu.: 3.000
##  Median :330.5   Median :53874   Median : 18000   Median : 5.000
##  Mean   :330.5   Mean   :55141   Mean   : 34574   Mean   : 4.706
##  3rd Qu.:495.2   3rd Qu.:77202   3rd Qu.: 48000   3rd Qu.: 6.000
##  Max.   :660.0   Max.   :99843   Max.   :200000   Max.   :10.000
##  Total_visits_bank Total_visits_online Total_calls_made
##  Min.   :0.000     Min.   : 0.000      Min.   : 0.000
##  1st Qu.:1.000     1st Qu.: 1.000      1st Qu.: 1.000
##  Median :2.000     Median : 2.000      Median : 3.000
##  Mean   :2.403     Mean   : 2.606      Mean   : 3.583
##  3rd Qu.:4.000     3rd Qu.: 4.000      3rd Qu.: 5.000
##  Max.   :5.000     Max.   :15.000      Max.   :10.000
```

```
sapply(df, function(x) sum(is.na(x)==TRUE))
```

```
##              Sl_No        Customer.Key    Avg_Credit_Limit  Total_Credit_Cards
##                  0                   0                   0                   0
##   Total_visits_bank Total_visits_online    Total_calls_made
##                  0                   0                   0
```

```
df <- df[!apply(is.na(df) | df == "", 1, all),]
```

Since the values of the data are on different scales (i.e. the scale of credit limit is extremely different from the scale of total credit cards), I scaled the data before performing the clustering. I displayed the head of the scaled dataset to see what the new values looked like as well.

```
df[,c(3:4)] <- scale(df[,c(3:4)])
head(df)
```

	Sl_... <int>	Customer.Key <int>	Avg_Credit_Limit <dbl>	Total_Credit_Cards <dbl>	Total_visits_bank <int>	Total_
1	1	87073	1.7388680	-1.2482780	1	
2	2	38414	0.4099816	-0.7869883	0	
3	3	17341	0.4099816	1.0581707	1	
4	4	40496	-0.1215730	0.1355912	1	
5	5	47437	1.7388680	0.5968810	0	
6	6	58634	-0.3873503	-0.7869883	0	

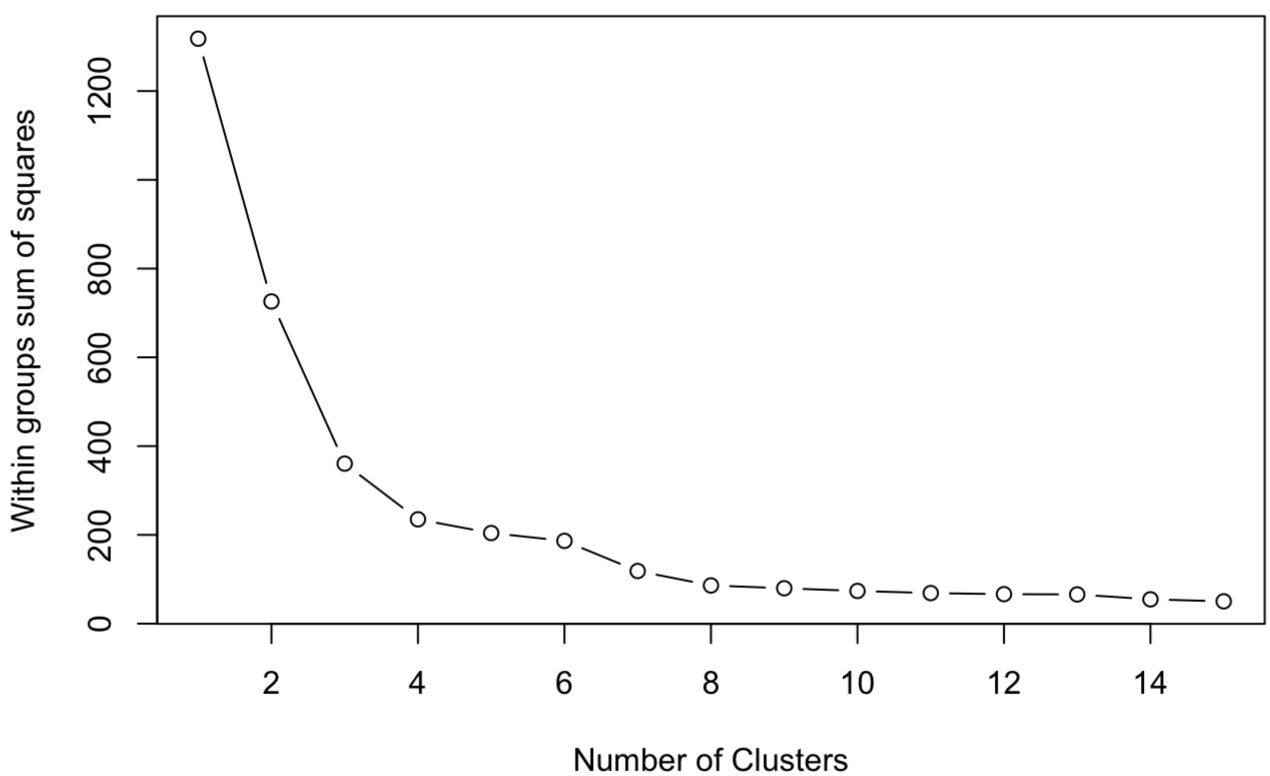
6 rows | 1-7 of 8 columns

Plot the within-groups sums of squares vs. the number of

clusters

In the section below, I used a function to plot the within-groups sums of squares vs. the number of clusters. I did this because I wanted to see where the plot would elbow. This would indicate the best amount of clusters that we would need in the KMeans clustering.

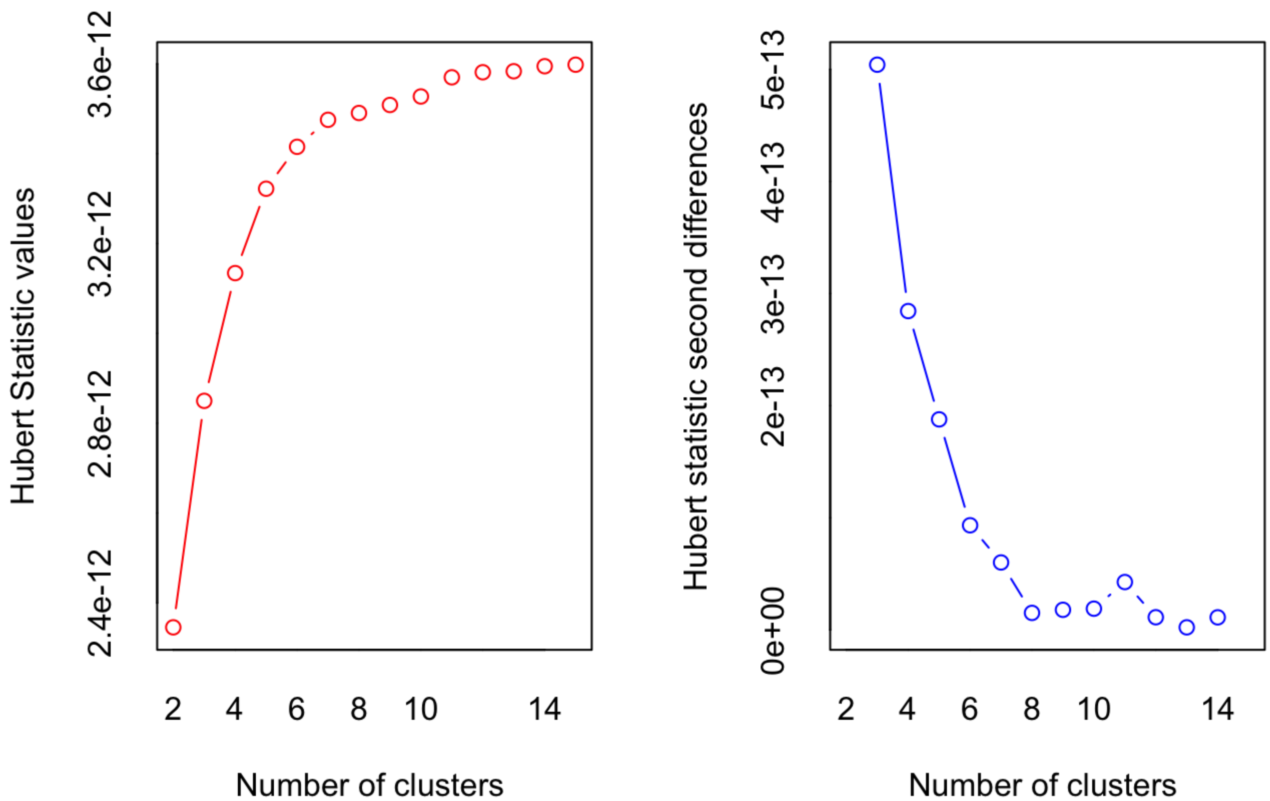
```
wsplot <- function(df, nc=15, seed=1234){
  withinss <- (nrow(df)-1)*sum(apply(df,2,var))
  for(i in 2:nc){
    set.seed(seed)
    withinss[i] <- sum(kmeans(df,centers=i)$withinss)
  }
  plot(1:nc, withinss, type="b", xlab="Number of Clusters", ylab="Within groups sum of s
quares")
}
wsplot(df[,c(3:4)])
```



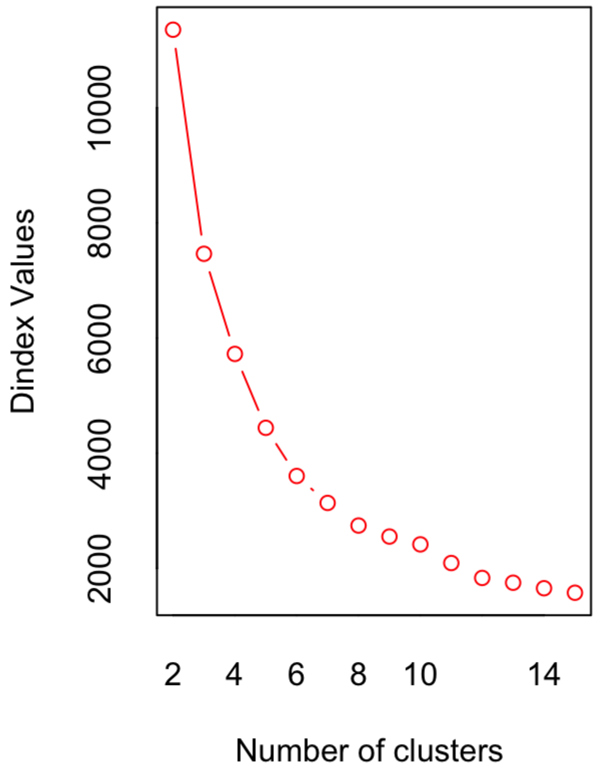
NbClust()

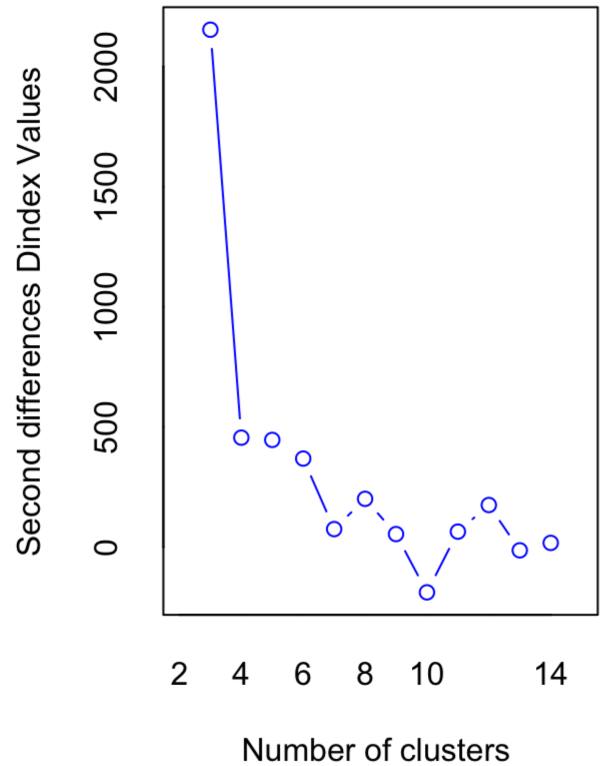
From the graph above, we can see that it elbows at about 3 clusters, so that indicates that using 3 clusters might be the best for this dataset. However, we can verify this estimate by using NbClust(), which is shown below.

```
library(NbClust)
set.seed(1234)
nc <- NbClust(df, min.nc=2, max.nc=15, method="kmeans")
```




```
## *** : The Hubert index is a graphical method of determining the number of clusters.
##                 In the plot of Hubert index, we seek a significant knee that correspo
nds to a
##                 significant increase of the value of the measure i.e the significant
peak in Hubert
##                 index second differences plot.
##
```



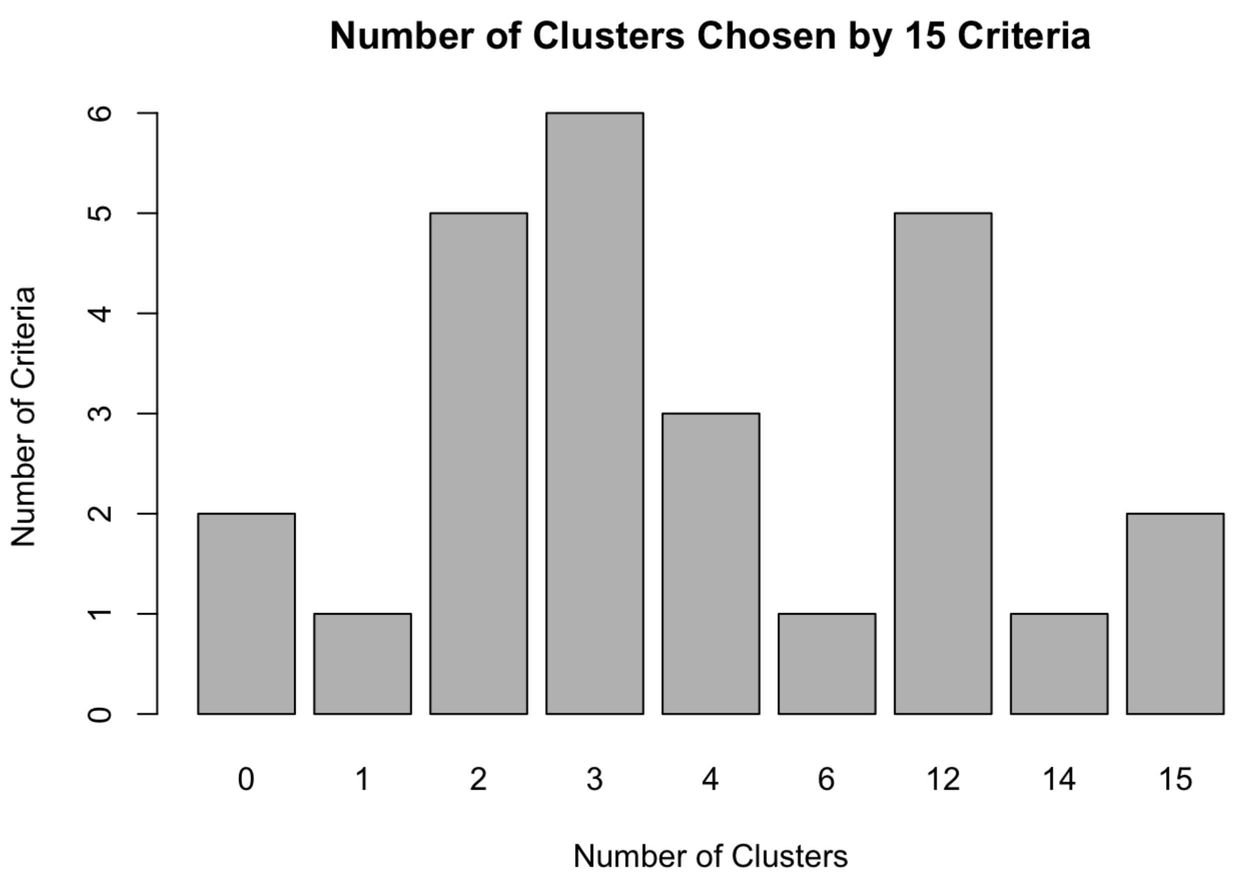



```
## *** : The D index is a graphical method of determining the number of clusters.
##                In the plot of D index, we seek a significant knee (the significant p
eak in Dindex
##                second differences plot) that corresponds to a significant increase o
f the value of
##                the measure.
##
## *******************************************************************
## * Among all indices:
## * 5 proposed 2 as the best number of clusters
## * 6 proposed 3 as the best number of clusters
## * 3 proposed 4 as the best number of clusters
## * 1 proposed 6 as the best number of clusters
## * 5 proposed 12 as the best number of clusters
## * 1 proposed 14 as the best number of clusters
## * 2 proposed 15 as the best number of clusters
##
##                   ***** Conclusion *****
##
## * According to the majority rule, the best number of clusters is  3
##
##
## *******************************************************************
```

```
table(nc$Best.n[1,])
```

```
##
##  0  1  2  3  4  6 12 14 15
##  2  1  5  6  3  1  5  1  2
```

```
barplot(table(nc$Best.n[1,]),
        xlab="Number of Clusters", ylab="Number of Criteria", main="Number of Clusters C
hosen by 15 Criteria")
```



The barplot above indicates that 2, 3, and 12 clusters would be good choices. Since 3 is the highest bar in the plot, I chose to go with 3 clusters for the KMeans clustering.

KMeans

Now, I'm fitting the model using the kmeans() function. I chose for the algorithm to have 20 random starts arbitrarily.

```
set.seed(1234)
bankCluster <- kmeans(df[,3:4], 3, nstart=20)
bankCluster
```

```
## K-means clustering with 3 clusters of sizes 48, 282, 330
##
## Cluster means:
##   Avg_Credit_Limit Total_Credit_Cards
## 1       2.87506592          1.9230890
## 2      -0.56981296         -0.9113075
## 3       0.06873967          0.4990316
##
## Clustering vector:
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
##   2   2   3   3   3   2   3   2   2   2   2   2   2   2   2   2   2   2   2   2
##  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40
##   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
##  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60
##   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
##  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80
##   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
##  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100
##   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
## 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120
##   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
## 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140
##   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
## 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160
##   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
## 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180
##   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
## 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200
##   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
## 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220
##   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
## 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240
##   2   2   2   2   2   2   2   2   3   3   2   3   3   3   3   3   3   3   3   3
## 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260
##   2   2   2   3   3   2   2   3   3   3   3   3   2   3   2   3   3   3   3   2
## 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280
##   2   3   3   2   2   3   3   3   3   3   2   3   3   3   3   3   2   2   3   3
## 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300
##   3   3   3   2   3   3   2   3   3   3   3   3   2   3   3   3   2   3   3   3
## 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320
##   3   3   2   2   2   3   3   3   2   3   2   3   3   2   3   2   3   3   3   3
## 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340
##   3   3   3   2   2   2   2   3   2   3   3   3   3   3   3   3   2   3   3   3
## 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360
##   3   2   3   3   3   3   3   3   2   3   3   3   3   3   3   3   3   2   3   2
## 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380
##   3   3   3   3   3   3   3   2   2   3   3   3   3   2   3   3   3   3   3   3
## 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400
##   3   2   3   3   2   3   3   3   3   2   2   2   2   2   3   2   3   3   3   3
## 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420
##   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3
## 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440
```

```
##   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3
## 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460
##   3   3   3   3   3   3   3   3   2   3   3   3   3   3   2   3   3   3   3   3
## 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480
##   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3
## 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500
##   3   3   3   3   3   3   3   3   3   3   3   3   3   3   2   3   3   3   2   3
## 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520
##   3   3   3   3   3   3   3   3   3   2   3   3   3   3   3   3   3   3   3   3
## 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540
##   3   3   3   3   3   3   2   3   2   3   3   2   3   3   3   2   3   3   3   3
## 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560
##   3   3   3   3   3   3   3   2   3   3   3   3   3   3   3   3   3   3   3   3
## 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580
##   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3
## 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600
##   3   3   3   3   3   3   3   3   3   2   3   3   3   3   3   3   3   3   3   3
## 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620
##   3   3   3   3   3   3   3   3   3   3   3   3   1   1   1   1   1   1   1   1
## 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640
##   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
## 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660
##   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
##
## Within cluster sum of squares by cluster:
## [1]  50.80039  94.72783 188.68910
##  (between_SS / total_SS =  74.6 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

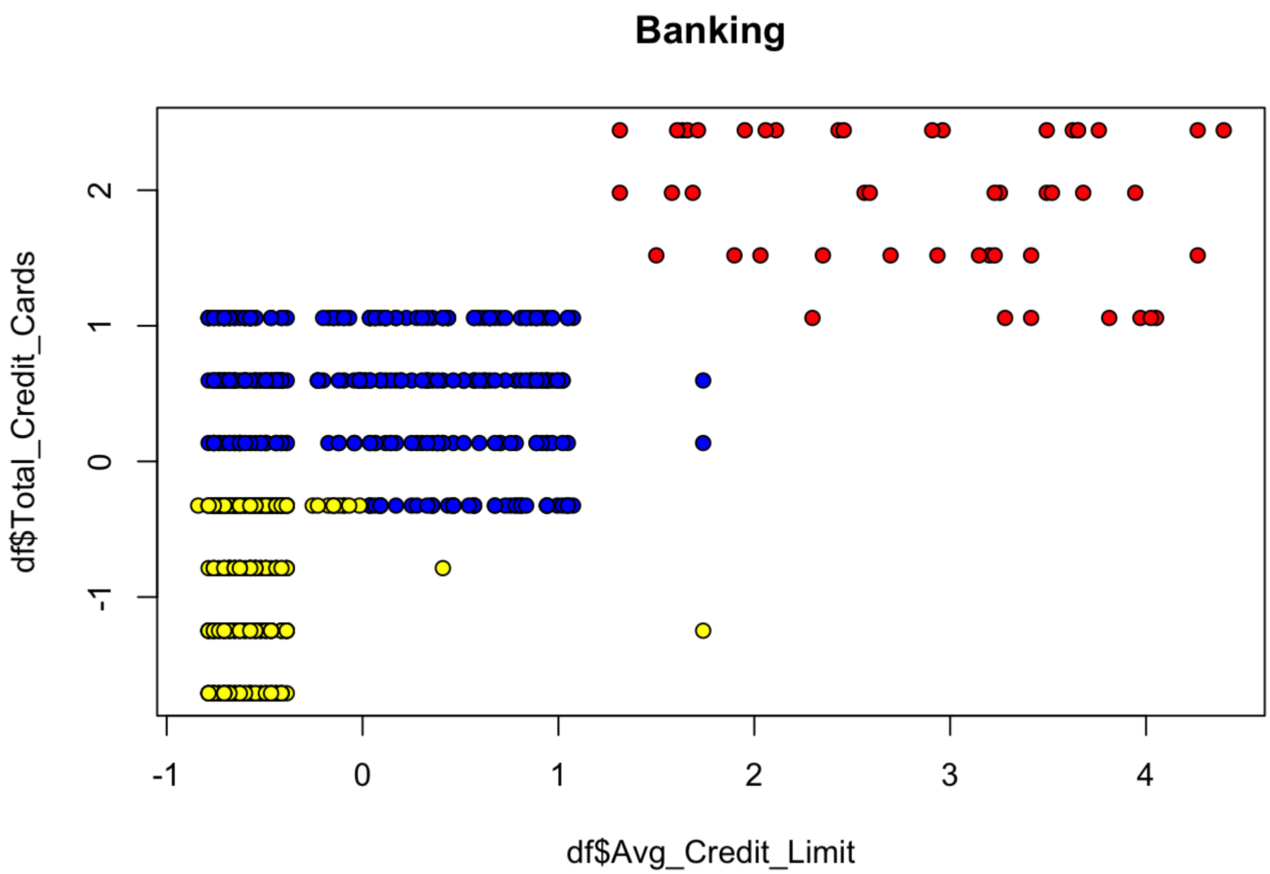
Although we ususally cluster blind, I chose my target for this assignment to be ‘Total_visits_bank’ in the dataset, so we compare the clusters with the total amount of visits that a customers has to the bank.

```
table(bankCluster$cluster, df$Total_visits_bank)
```

```
##
##      0  1  2  3  4  5
##   1 18 30  0  0  0  0
##   2 80 80 75 12 15 20
##   3  2  2 83 88 77 78
```

Now, I plot the clusters to see a visual representation of it. The plot is seen below.

```
plot(df$Avg_Credit_Limit, df$Total_Credit_Cards, pch=21, bg=c("red","yellow","blue","pur
ple")[unclass(bankCluster$cluster)], main="Banking")
```



The centroids are found in bankCluster$centers, so I wanted to display those as well. This is done below

```
bankCluster$size
```

```
## [1]  48 282 330
```

```
bankCluster$centers
```

```
##   Avg_Credit_Limit Total_Credit_Cards
## 1       2.87506592          1.9230890
## 2      -0.56981296         -0.9113075
## 3       0.06873967          0.4990316
```

Since we scaled the data in the beginning, the centroids were calculated based on that scaled data. Below, I used the aggregate() function to get the variable means for each cluster in units of the unscaled data.

```
aggregate(df[3:4], by=list(cluster=bankCluster$cluster), mean)
```

cluster <int>	Avg_Credit_Limit <dbl>	Total_Credit_Cards <dbl>
1	2.87506592	1.9230890

cluster	Avg_Credit_Limit	Total_Credit_Cards
<int>	<dbl>	<dbl>
2	-0.56981296	-0.9113075
3	0.06873967	0.4990316

3 rows

Model Analysis

We now analyze our data. I cross-tabulated the 'Total_visits_bank' with the clusters to see whether or not the clusters are strongly correlated with the amount of bank visits a customer performs.

```
ct.km <- table(df$Total_visits_bank, bankCluster$cluster)
head(ct.km)
```

```
##
##       1  2  3
##    0 18 80  2
##    1 30 80  2
##    2  0 75 83
##    3  0 12 88
##    4  0 15 77
##    5  0 20 78
```

Below, I quantified the agreement between the cluster and the target attribute. I used an adjusted Rand index, which provides a measure of agreement.

```
library(flexclust)
```

```
## Loading required package: grid
```

```
## Loading required package: lattice
```

```
## Loading required package: modeltools
```

```
## Loading required package: stats4
```

```
randIndex(ct.km)
```

```
##       ARI
## 0.1370758
```

As we can see, the result we got was around 0.137. Since the result could range from -1, being no agreement, and 1, being perfect agreement, we can say that our results had decent agreement. This means that there can be some relation between the clusters and the amount of visits that a customer performs at a bank.

Hierarchical Clustering

In contrast to KMeans Clustering, I'm now going to attempt to do Hierarchical clustering on the same data set. I'm going to be using the two same attributes for clustering, and I'm going to have the same target attribute to compare the clusters against.

Below, I re-read in the dataset from the csv file in order to get rid of the scaled data that might previously be there.

```
df <- read.csv('/Users/kellytrinh/Desktop/school/Similarity and Ensemble/Credit Card Cus
tomer Data.csv', na.strings="NA", header=TRUE)
```

Data exploration and cleaning

I re-did my data exploration and cleaning by removing the NAs. I have less functions to explore the data here because I previously completed that step.

```
head(df)
```

	Sl_...	Customer.Key	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_
	<int>	<int>	<int>	<int>	<int>	
1	1	87073	100000	2	1	
2	2	38414	50000	3	0	
3	3	17341	50000	7	1	
4	4	40496	30000	5	1	
5	5	47437	100000	6	0	
6	6	58634	20000	3	0	

6 rows | 1-7 of 8 columns

```
sapply(df, function(x) sum(is.na(x)==TRUE))
```

```
##                Sl_No         Customer.Key     Avg_Credit_Limit  Total_Credit_Cards
##                    0                    0                    0                   0
##    Total_visits_bank Total_visits_online     Total_calls_made
##                    0                    0                    0
```

```
df <- df[!apply(is.na(df) | df == "", 1, all),]
```

Scale the data for the clustering

I re-scaled the data of columns that I'm using, which is shown below.

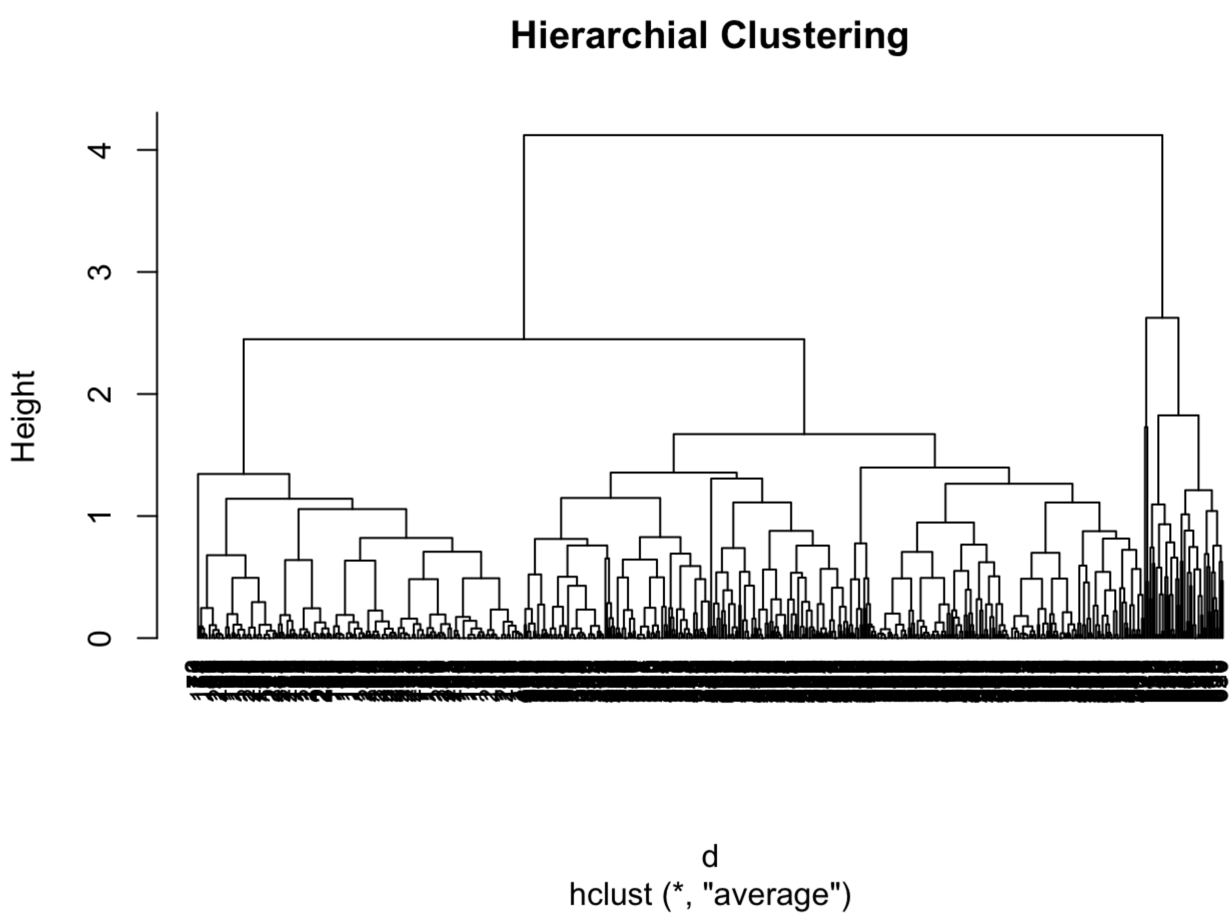
```
df.scaled <- scale(df[,c(3:5)])
head(df.scaled)
```

```
##   Avg_Credit_Limit Total_Credit_Cards Total_visits_bank
## 1        1.7388680         -1.2482780        -0.8597985
## 2        0.4099816         -0.7869883        -1.4726139
## 3        0.4099816          1.0581707        -0.8597985
## 4       -0.1215730          0.1355912        -0.8597985
## 5        1.7388680          0.5968810        -1.4726139
## 6       -0.3873503         -0.7869883        -1.4726139
```

Distance

Below, I calculated the Euclidean distances between each of the observations using average-linkage. Then, also shown below, I plotted the clustering performed in a dendogram.

```
d <- dist(df.scaled)
fit.average <- hclust(d, method="average")
plot(fit.average, hang=-1, cex=.8, main="Hierarchial Clustering")
```



As we can see the dendogram is extremely hard to interpret because there are so many observations in this data set.

Cut the dendogram

Below, I created a loop in order to cut the dendogram. I cut the dendogram in terms with ‘Total_bank_visits’. I chose to do up 15 cuts to stay consistent with the KMeans clustering so I could compare the two. The KMeans clustering used NbClust() chosen by 15 criteria.

```
for(c in 3:15) {
  cluster_cut <- cutree(fit.average,c)
  table_cut <- table(cluster_cut, df$Total_visits_bank)
  print(table_cut)
  ri <- randIndex(table_cut)
  print(paste("cut=", c, "Rand index = ", ri))
}
```

```
##
## cluster_cut   0   1   2   3   4   5
##           1   2   1   0   0   0   0
##           2  80  81 158 100  92  98
##           3  18  30   0   0   0   0
## [1] "cut= 3 Rand index =  0.0150107527388418"
##
## cluster_cut   0   1   2   3   4   5
##           1   2   1   0   0   0   0
##           2  80  79  51   0   0   0
##           3   0   2 107 100  92  98
##           4  18  30   0   0   0   0
## [1] "cut= 4 Rand index =  0.191429545062653"
##
## cluster_cut   0   1   2   3   4   5
##           1   2   1   0   0   0   0
##           2  80  79  51   0   0   0
##           3   0   2 107 100  92  98
##           4  13  15   0   0   0   0
##           5   5  15   0   0   0   0
## [1] "cut= 5 Rand index =  0.188391009915484"
##
## cluster_cut   0   1   2   3   4   5
##           1   0   1   0   0   0   0
##           2  80  79  51   0   0   0
##           3   0   2 107 100  92  98
##           4   2   0   0   0   0   0
##           5  13  15   0   0   0   0
##           6   5  15   0   0   0   0
## [1] "cut= 6 Rand index =  0.188399986732693"
##
## cluster_cut   0   1   2   3   4   5
##           1   0   1   0   0   0   0
##           2  80  79  51   0   0   0
##           3   0   2 107 100   0   0
##           4   2   0   0   0   0   0
##           5   0   0   0   0  92  98
##           6  13  15   0   0   0   0
##           7   5  15   0   0   0   0
## [1] "cut= 7 Rand index =  0.425940629391381"
##
## cluster_cut   0   1   2   3   4   5
##           1   0   1   0   0   0   0
##           2  80  79  51   0   0   0
##           3   0   2 107 100   0   0
##           4   2   0   0   0   0   0
##           5   0   0   0   0  83  89
##           6   0   0   0   0   9   9
##           7  13  15   0   0   0   0
##           8   5  15   0   0   0   0
## [1] "cut= 8 Rand index =  0.410652420978037"
##
```

```
## cluster_cut  0  1  2  3  4  5
##           1  0  1  0  0  0  0
##           2 80 79 51  0  0  0
##           3  0  1 38 50  0  0
##           4  0  1 69 50  0  0
##           5  2  0  0  0  0  0
##           6  0  0  0  0 83 89
##           7  0  0  0  0  9  9
##           8 13 15  0  0  0  0
##           9  5 15  0  0  0  0
## [1] "cut= 9 Rand index =  0.357303589280882"
##
## cluster_cut  0  1  2  3  4  5
##          1   0  1  0  0  0  0
##          2   1  0  0  0  0  0
##          3   0  1 38 50  0  0
##          4   0  1 69 50  0  0
##          5   2  0  0  0  0  0
##          6  79 79 51  0  0  0
##          7   0  0  0  0 83 89
##          8   0  0  0  0  9  9
##          9  13 15  0  0  0  0
##          10  5 15  0  0  0  0
## [1] "cut= 10 Rand index =  0.356767143291976"
##
## cluster_cut  0  1  2  3  4  5
##          1   0  1  0  0  0  0
##          2   1  0  0  0  0  0
##          3   0  1  0  0  0  0
##          4   0  1 69 50  0  0
##          5   2  0  0  0  0  0
##          6  79 79 51  0  0  0
##          7   0  0  0  0 83 89
##          8   0  0  0  0  9  9
##          9   0  0 38 50  0  0
##          10 13 15  0  0  0  0
##          11  5 15  0  0  0  0
## [1] "cut= 11 Rand index =  0.357504656988432"
##
## cluster_cut  0  1  2  3  4  5
##          1   0  1  0  0  0  0
##          2   1  0  0  0  0  0
##          3   0  1  0  0  0  0
##          4   0  1 69 50  0  0
##          5   2  0  0  0  0  0
##          6  79 79 51  0  0  0
##          7   0  0  0  0 45 41
##          8   0  0  0  0 38 48
##          9   0  0  0  0  9  9
##          10  0  0 38 50  0  0
##          11 13 15  0  0  0  0
##          12  5 15  0  0  0  0
```

```
## [1] "cut= 12 Rand index =  0.309579514270492"
##
## cluster_cut  0  1  2  3  4  5
##           1  0  1  0  0  0  0
##           2  1  0  0  0  0  0
##           3  0  1  0  0  0  0
##           4  0  1 69 50  0  0
##           5  2  0  0  0  0  0
##           6 79 79 51  0  0  0
##           7  0  0  0  0 45 41
##           8  0  0  0  0 38 48
##           9  0  0  0  0  9  9
##          10  0  0 38 50  0  0
##          11  7  8  0  0  0  0
##          12  5 15  0  0  0  0
##          13  6  7  0  0  0  0
## [1] "cut= 13 Rand index =  0.308185660131247"
##
## cluster_cut  0  1  2  3  4  5
##           1  0  1  0  0  0  0
##           2  1  0  0  0  0  0
##           3  0  1  0  0  0  0
##           4  0  1 42 16  0  0
##           5  2  0  0  0  0  0
##           6 79 79 51  0  0  0
##           7  0  0 27 34  0  0
##           8  0  0  0  0 45 41
##           9  0  0  0  0 38 48
##          10  0  0  0  0  9  9
##          11  0  0 38 50  0  0
##          12  7  8  0  0  0  0
##          13  5 15  0  0  0  0
##          14  6  7  0  0  0  0
## [1] "cut= 14 Rand index =  0.28566206681468"
##
## cluster_cut  0  1  2  3  4  5
##           1  0  1  0  0  0  0
##           2  1  0  0  0  0  0
##           3  0  1  0  0  0  0
##           4  0  1 42 16  0  0
##           5  2  0  0  0  0  0
##           6 79 79  0  0  0  0
##           7  0  0 51  0  0  0
##           8  0  0 27 34  0  0
##           9  0  0  0  0 45 41
##          10  0  0  0  0 38 48
##          11  0  0  0  0  9  9
##          12  0  0 38 50  0  0
##          13  7  8  0  0  0  0
##          14  5 15  0  0  0  0
##          15  6  7  0  0  0  0
## [1] "cut= 15 Rand index =  0.361644582942091"
```

The results from cutting the dendogram show that cuts at 7-11 and 15 gives the best correspondence with 'Total_bank_visits'. Specifically, 7 cuts gives the best correspondence out of all the Rand index values. This is interesting because, based on the KMeans clustering, having 7 clusters was not one of the best options.

Model Based Clustering

Now, I'm lastly going to try Model Based Clustering on the dataset, which is more of a statistical approach to clustering.

Below, I re-read in the data from the csv file in order to "reset" the data.

```
df <- read.csv('/Users/kellytrinh/Desktop/school/Similarity and Ensemble/Credit Card Cus
tomer Data.csv', na.strings="NA", header=TRUE)
```

Data cleaning and exploration

Since we previously did data exploration twice in this part of the assignment, I'm just going to clean the data again by removing NAs.

```
sapply(df, function(x) sum(is.na(x)==TRUE))
```

```
##               Sl_No        Customer.Key    Avg_Credit_Limit  Total_Credit_Cards
##                   0                   0                   0                   0
##   Total_visits_bank Total_visits_online    Total_calls_made
##                   0                   0                   0
```

```
df <- df[!apply(is.na(df) | df == "", 1, all),]
```

Model Clustering

The below code performs the model-based clustering based on the two attributes that I previously used, which are related to a customer's credit score and the customer's total amount of credit cards.

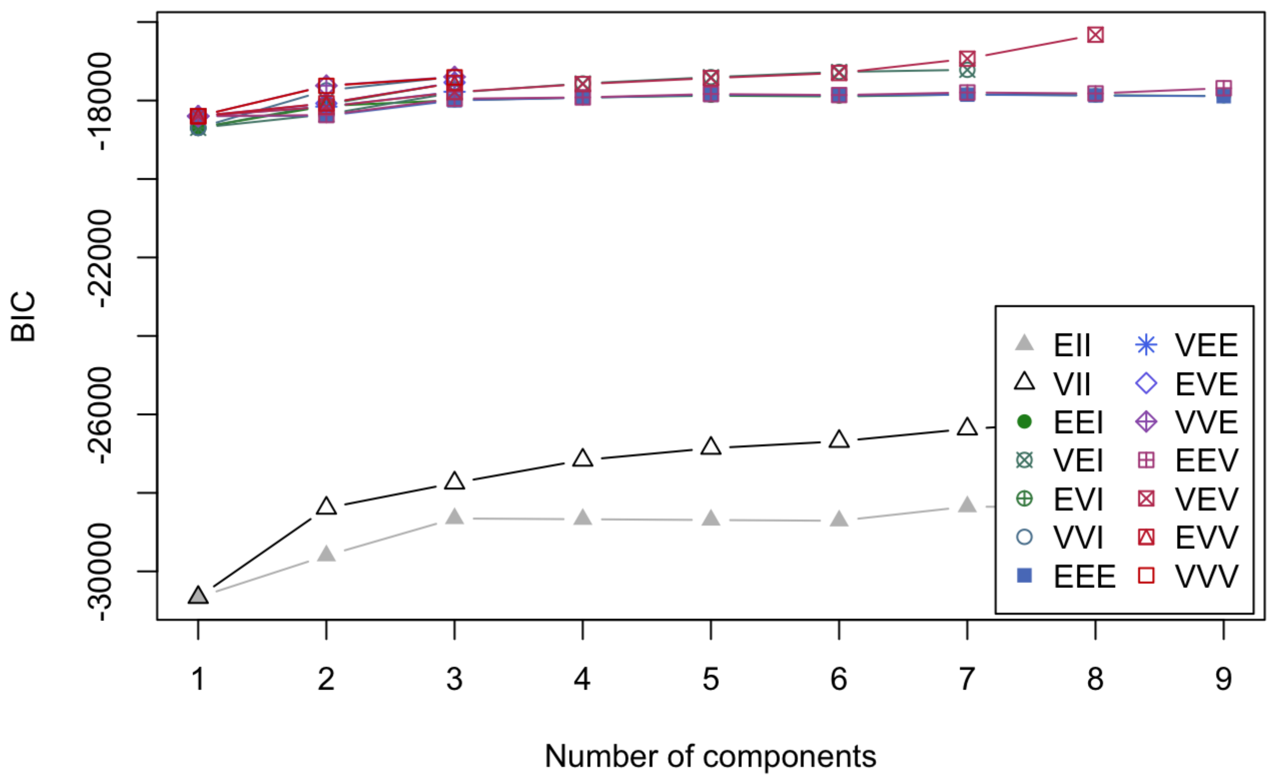
```
library(mclust)
```

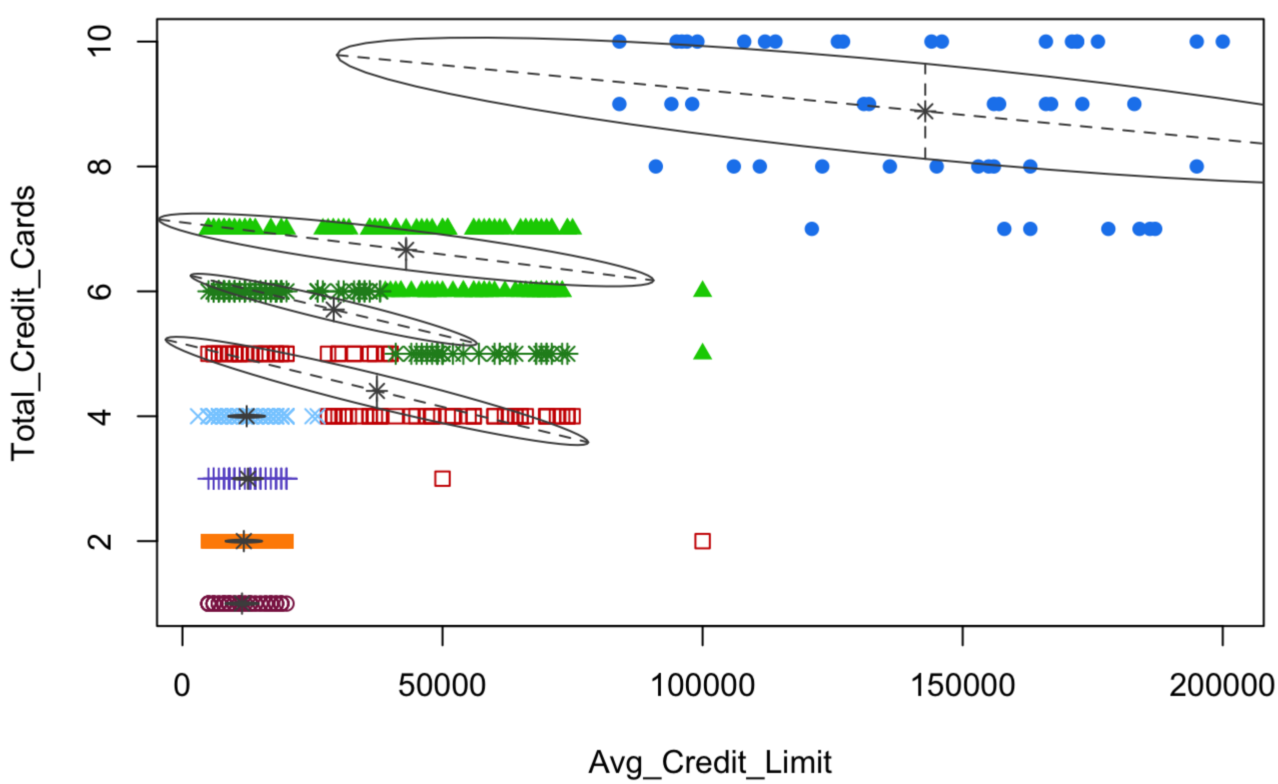
```
## Package 'mclust' version 6.0.0
## Type 'citation("mclust")' for citing this R package in publications.
```

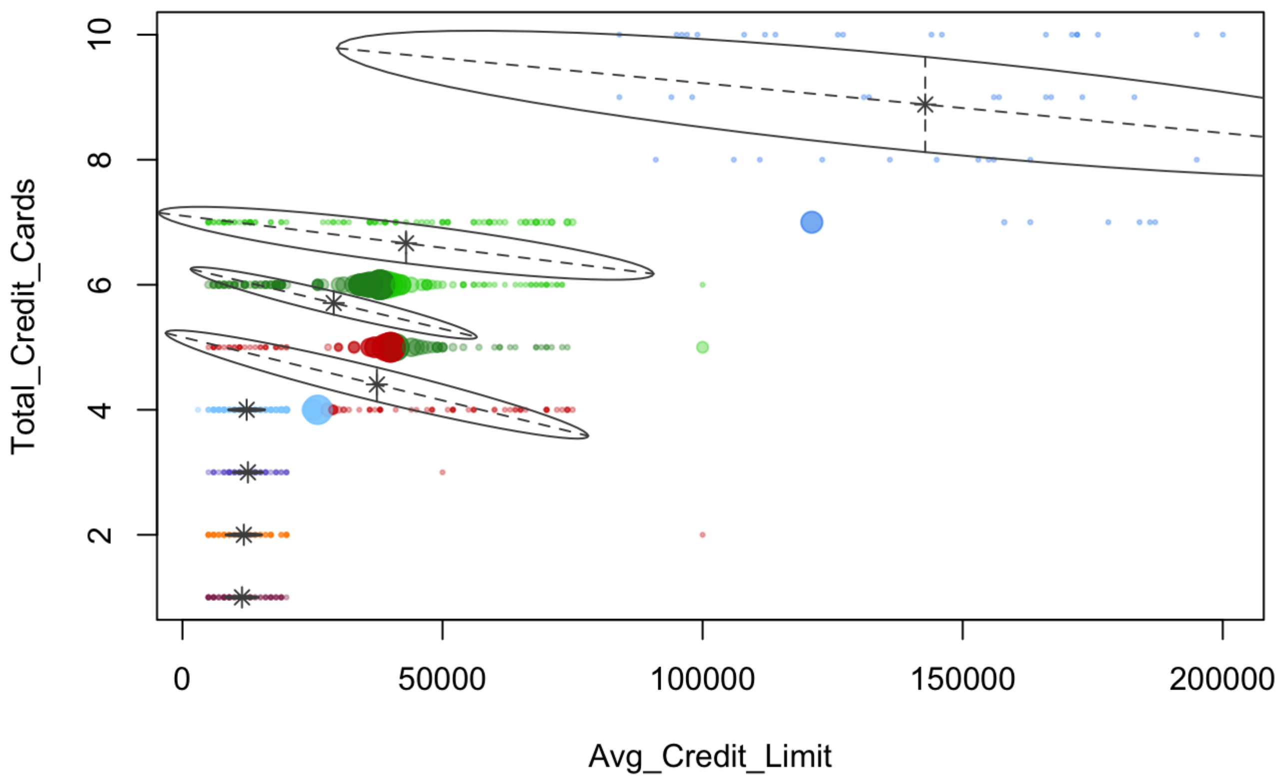
```
citation("mclust")
```

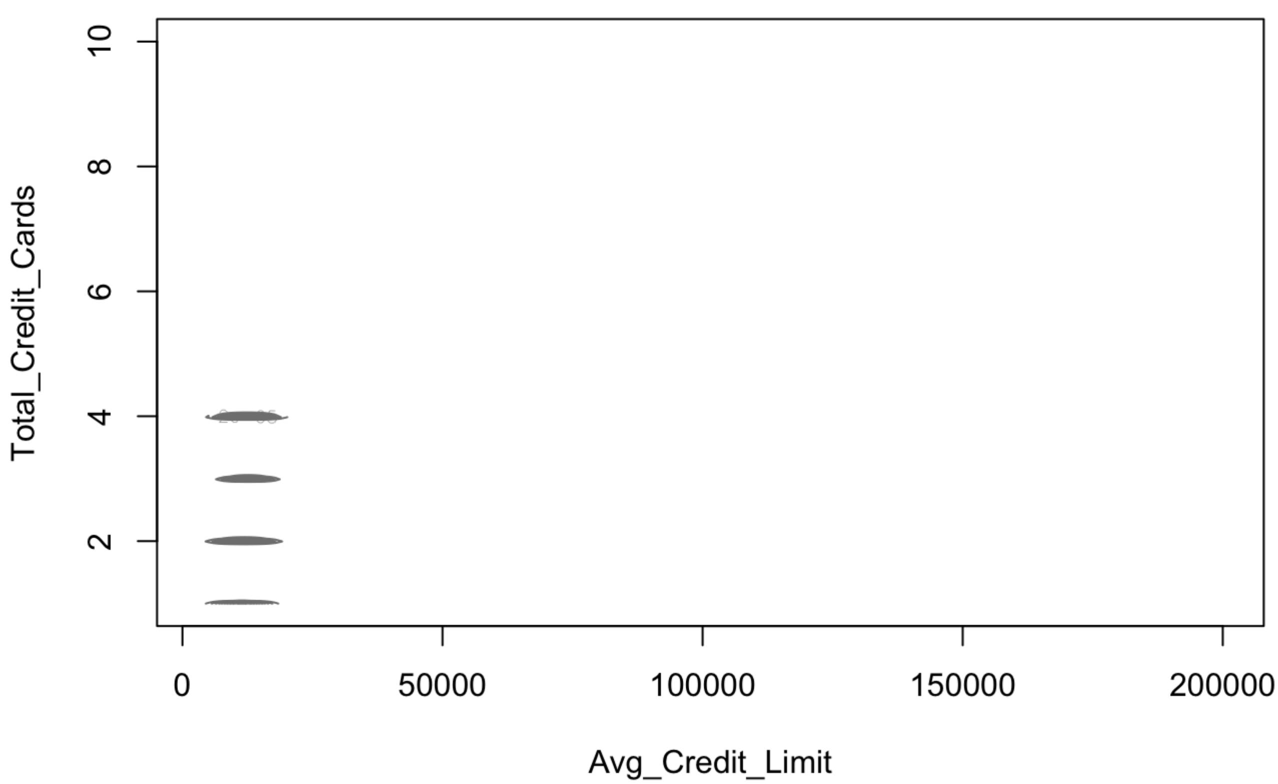
```
##
## To cite 'mclust' R package in publications, please use:
##
##   Scrucca L., Fop M., Murphy T. B. and Raftery A. E. (2016) mclust 5:
##   clustering, classification and density estimation using Gaussian
##   finite mixture models The R Journal 8/1, pp. 289-317
##
## A BibTeX entry for LaTeX users is
##
##   @Article{,
##     title = {{mclust} 5: clustering, classification and density estimation using {G}a
ussian finite mixture models},
##     author = {Luca Scrucca and Michael Fop and T. Brendan Murphy and Adrian E. Rafter
y},
##     journal = {The {R} Journal},
##     year = {2016},
##     volume = {8},
##     number = {1},
##     pages = {289--317},
##     url = {https://doi.org/10.32614/RJ-2016-021},
##   }
```

```
fit <- Mclust(df[,c(3:4)])
plot(fit)
```


BIC
-18000
-22000
-26000
-30000
Number of components
1
2
3
4
5
6
7
8
9
EII
VII
EEI
VEI
EVI
VVI
EEE
VEE
EVE
VVE
EEV
VEV
EVV
VVV


Total_Credit_Cards
2
4
6
8
10
Avg_Credit_Limit
0
50000
100000
150000
200000


Total_Credit_Cards
2
4
6
8
10
0
50000
100000
150000
200000
Avg_Credit_Limit


Total_Credit_Cards
2
4
6
8
10
0
50000
100000
150000
200000
Avg_Credit_Limit

Model-based clustering assumes many data models and uses statistical measures, such as maximum likelihood and Bayes Criteria to identify the most likely model and the most likely number of clusters. Above, I used the Mclust() function to perform the clustering, which selects the optimal model according to multiple statistical plots. I displayed all of the plots, but I'm going to be focusing on the Bayesian Information Criterion (BIC) plot.

The first plot, or the BIC plot, displays the models according to BIC for EM (Expectation-Maximization). The best model would be the one with the highest BIC, or the highest option in the plot. Using this, we can see that VEV with 8 clusters is the best model. VEV stands for varying volume, equal shape, varying orientation. This means that the shape of the clusters are ellipsoidal covariance.

Analysis of Clustering Models

After completing all of the three clustering methods, there were varying results. Using KMeans clustering, we were able to see that 3 clusters was the best option, however it did not give us a great agreement (ARI) score. This means that there may not have been a great amount of correspondence between the two attributes in terms of the amount of bank visits that a customer performs at a bank. For Hierarchial clustering, the best Rand index that we got was for 7 clusters using the same two attributes (Credit Limit and Total Credit Cards). This is different in comparison to using the KMeans clustering. However, the results of the Rand index for 7 clusters in this clustering method was still not extremely high, meaning that even though there was some correspondence, it still wasn't great. Lastly, with Model-based clustering, we were able to determine the best statistical model for clustering the dataset. This type of clustering wasn't able to tell us similar results in comparison to the first two, but it chose 9 clusters as the most optimal amount for the chosen model.

Through these observations, we are able to see that the dataset does not necessarily have a strong correspondence between average credit score and total credit cards in terms of total bank visits. The insights that we received were that the data does not necessarily have a hierarchical structure, and clustering the data using KMeans visually gave us three distinct clusters. The ARI score was also not negative, meaning that KMeans was an "okay" clustering method.

Each of the clustering methods had their own pros and cons, and there are reasons that the results may have not been completely consistent. For instance, model-based clustering assumes that the cluster group densities are Gaussian when they may not be. This may cause overfitting of the data.

References:

The references that I used in order to do the model-based clustering and analysis are: [1] https://www.statmethods.net/advstats/cluster.html (https://www.statmethods.net/advstats/cluster.html) [2] https://www.stat.cmu.edu/~rnugent/PUBLIC/teaching/CMU729/Lectures/MBCComments.pdf (https://www.stat.cmu.edu/~rnugent/PUBLIC/teaching/CMU729/Lectures/MBCComments.pdf)