Data Mining practice 6

Task 1: 강의노트 20 페이지 k-means clustering을 구현하시오. (시연 가능한 형태로 준비)

```
# k-means clust
set.seed(2018)
synth.data <- data.frame(
  x1 = c(rnorm(20, 3, 1.5), rnorm(20, 0, 1), rnorm(20, 5,
1)),
  x2 = c(rnorm(20, 0, 1), rnorm(20, 4, 1), rnorm(20, 5, 1))
)
ndata <- nrow(synth.data)

# 유클리드 거리 계산 함수
u_dist <- function(u, v) {
  sqrt(sum((u - v)^2))
}

# 클러스터 수 설정
k <- 3
```

가장 먼저 데이터 프레임을 생성함(예시를 참고함) 랜덤하게 데이터프레임을 생성한 후, 유클리드 거리를 계산하는 함수를 지정, 클러스터 수를 설정함 3개의 군집을 찾고자

```
# 초기 중심점 설정
set.seed(123)  # 재현성을 위해 시드 설정
cents <- data.frame(cl = 1:k)
cents <- cbind(cents, synth.data[sample(1:ndata, k),])

# 각 데이터 포인트를 가장 가까운 중심점에 할당하는 함수
assign_clusters <- function(data, cents) {
  apply(data[, c("x1", "x2")], 1, function(point) {
    dists <- apply(cents[, c("x1", "x2")], 1, function(cent
```

```
roid) {
      u_dist(point, centroid)
    })
    which.min(dists)
  })
}
```

랜덤한 값으로 초기 중심정을 설정함 무작위 3개를 데이터프레임의 센터로 설정하고, 중심점으로 잡음

이제 x1과 x2에 대해 가까운 거리의 클러스터를 찾는 과정을 반복하는 함수를 만듬

계산된 거리중 가장 가까운 값을 가지는 중심점을 찾는 which.min(dists)사용

```
# 중심점을 업데이트하는 함수
update_centroids <- function(data, k) {
  cents_updated <- data %>%
    group_by(cl) %>%
    summarise(x1 = mean(x1), x2 = mean(x2)) %>%
    ungroup()

  # 클러스터 수가 k와 동일하지 않으면, NA로 처리된 값을 삭제하고 새로운 중심점을 추가
  while (nrow(cents_updated) < k) {
    new_cents <- synth.data[sample(1:ndata, 1), ]
    new_cent <- data.frame(cl = max(cents_updated$cl) + 1, x1 = new_cents$x1, x2 = new_cents$x2)
    cents_updated <- rbind(cents_updated, new_cent)
  }

  return(cents_updated)
}
```

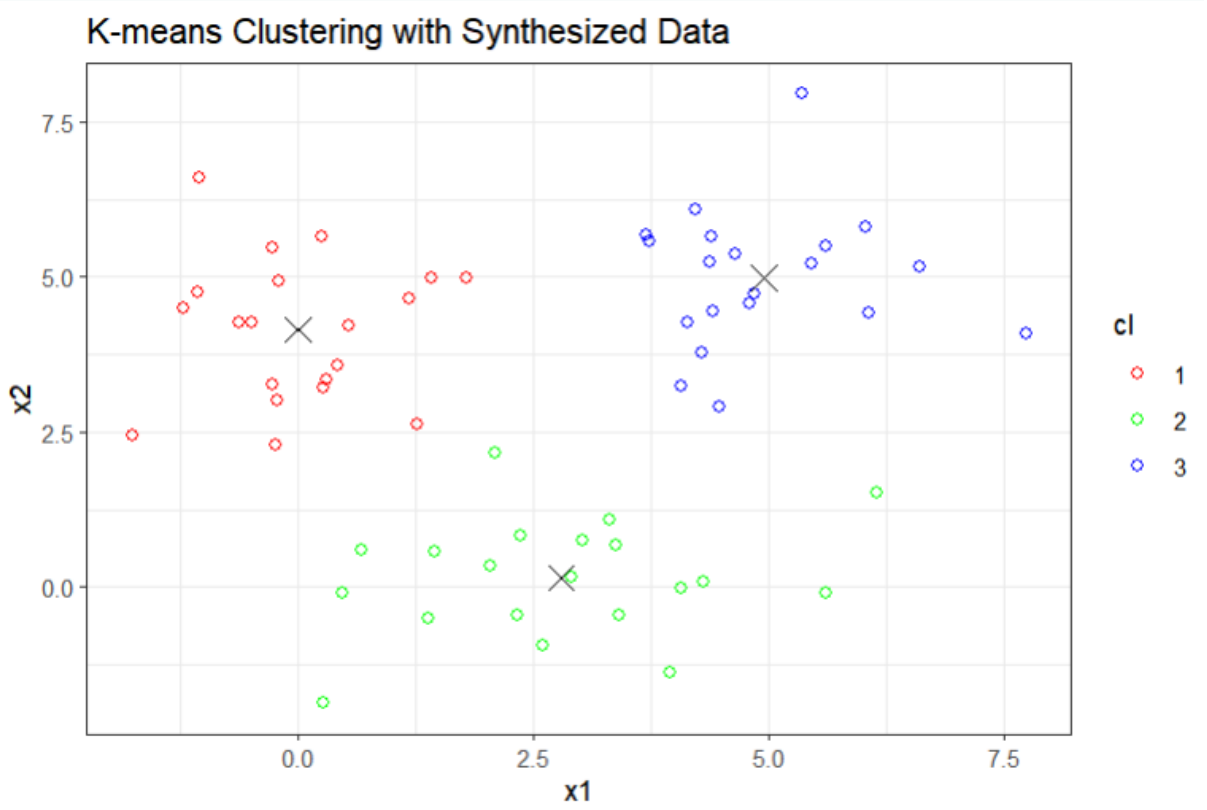
찾은 중심점을 업데이트하는 함수, 센터 업데이트를 통해 데이터를 클러스터별로 그룹화를 한 후, x1클러스터 내의 평균과 x2클러스터 내의 평균을 반환함

이후 업데이트된 클러스터들을 찾음

```r
# K-means 클러스터링 실행
for (i in 1:10) {  # 10번 반복
  # 1. 각 데이터 포인트를 가장 가까운 중심점에 할당
  synth.data$cl <- factor(assign_clusters(synth.data, cent
s), levels = 1:k)

  # 2. 중심점을 할당된 데이터의 평균으로 업데이트
  cents <- update_centroids(synth.data, k)
}
```

이 과정을 총 10번 반복하며 각 데이터포인트를 가장 가까운 중심으로 이동시키고, 중심점을 데이터의 평균으로 업데이트함

```r
# 결과 시각화 (세 가지 다른 색상을 지정)
synth.data %>%
  ggplot(aes(x = x1, y = x2, col = cl)) +
  geom_point(shape = 1, size = 2) +  # 데이터 포인트
  theme_bw() +
  geom_point(data = cents, aes(x = x1, y = x2), shape = 4,
col = 'black', size = 4) +  # 중심점
  scale_color_manual(values = c("red", "green", "blue")) +
# 세 가지 색상 지정
  labs(title = "K-means Clustering with Synthesized Data",
x = "x1", y = "x2") +
  theme(legend.position = "right")
```



결과 시각화 중심값을 X로 표시하며 가장 근처에 있는 클러스터 군집을 각기 다른 색으로 표시함

Task2: 강의노트 38페이지의 k 값에 따라 달라지는 CH index와 WSS를 계산하여 그래프로 표현하여라표현하여라.

```
protein <- read.table("C:/Users/silkj/Desktop/한동대학교/5학기/데이터 마이닝 실습/Data-Mining-Practicum/myR/protein.txt",
                      sep = "\t", header = TRUE)
vars.to.use <- colnames(protein)[-1]
pmatrix <- scale(protein[,vars.to.use])
pcenter <- attr(pmatrix, "scaled:center")
pscale <- attr(pmatrix, "scaled:scale")
```

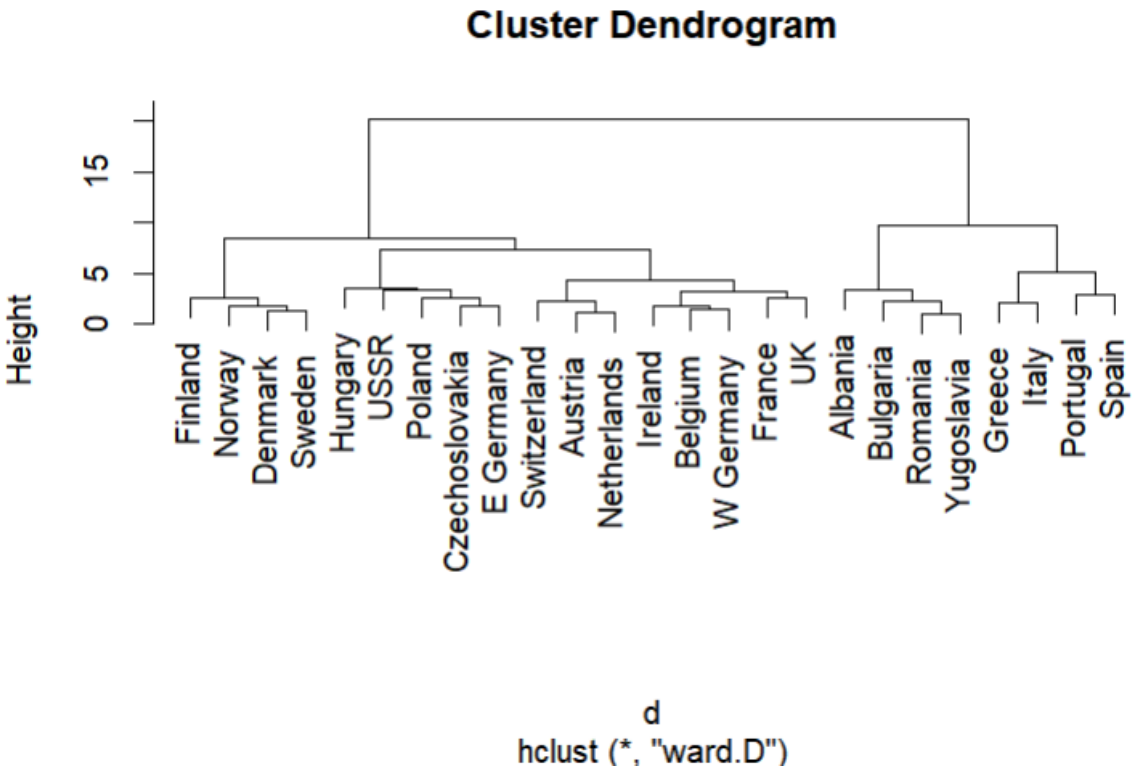
프로틴 데이터를 가지고 온 후, 데이터 정규화 과정을 거침 이제 변수간 크기 차이를 최소화해 스케일의 영향을 줄임

```
d <- dist(pmatrix, method = "euclidean") #유클리드 거리를 계산하는 함수
pfit <- hclust(d, method = "ward.D") # 계층적 클러스터링을 시행하는 함수
plot(pfit, labels=protein$Country)
```

이제 계층적 클러스터링과 K-means클러스터링을 통해 데이터를 군집화함

데이터간의 거리는 d ← dist(pmatrix, method = "euclidean") 를 통해 유클리드 거리를 통해 계산함

이제 계층적 클러스터링을 시행, hclust함수를 사용하고 이전에 구한 유클리드 거리에 대해, Ward.D방법을 통해 각 데이터를 클러스터로 묶을 때, 각 클러스터 내부 제곱합을 최소화함. 즉 유사한 데이터끼리 묶어서 군집 내 변동성을 줄임



덴드로그램 시각화, 각 국가들이 어떤 군집에 속하는지 시각화함

```
groups <- cutree(pfit, k = 5)
print_clusters<- function(labels, k){
  for(i in 1:k){
```

```
    print(paste("cluster",i))
    print(protein[labels == i, c("Country","RedMeat","Fis
h","Fr.Veg")])
  }
}
print_clusters(groups,5)
pclusters <- kmeans(pmatrix, 5, nstart = 100, iter.max = 10
0)
groups <- pclusters$cluster
```

덴드로그램을 5개의 클러스터로 나눔

이제 각 클러스터별로 어디에 속하는지 출력하고 kmeans클러스터 수행. 총 100회 시행하며 결과를 groups로 저장 → LMS 예시

이제 WSS와 CH index 계산 시행

```
k_values <- 2:10  # 클러스터 수 K를 2부터 10까지 실험
wss_values <- numeric(length(k_values))  # WSS 저장 리스트
ch_values <- numeric(length(k_values))   # CH index 저장 리스
트
```

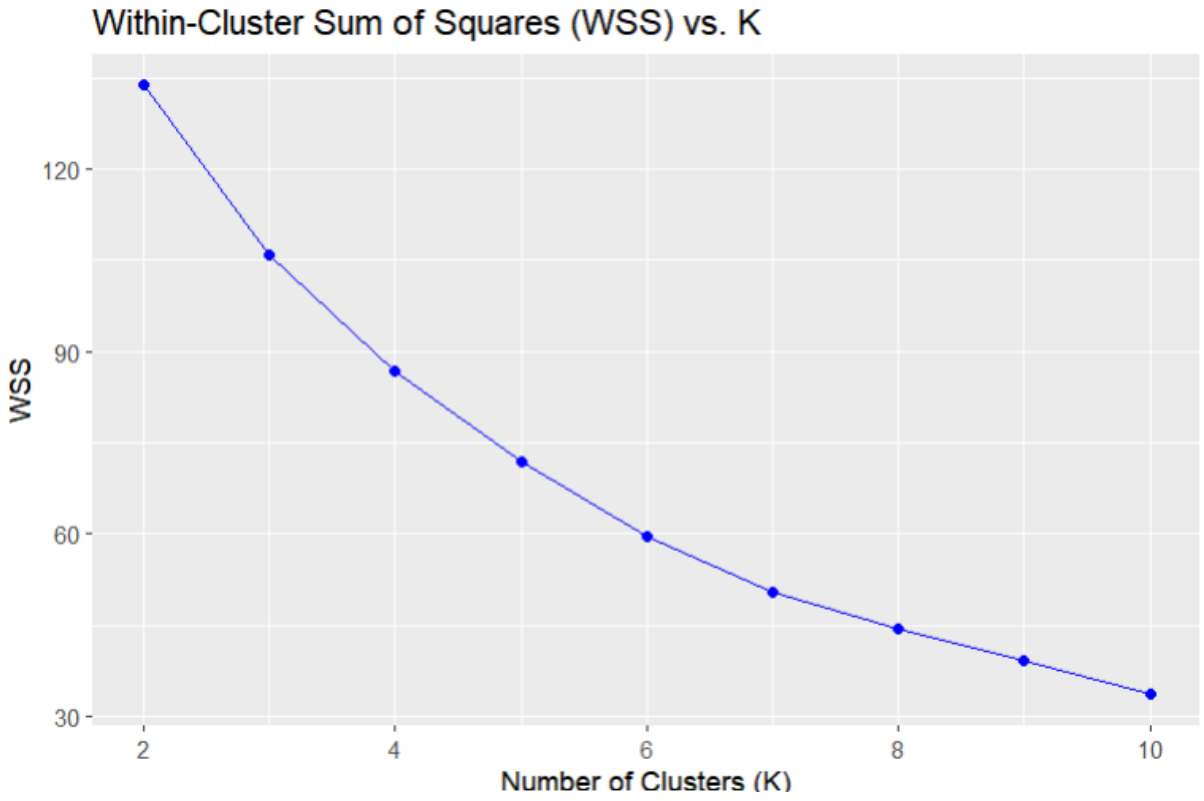
k값의 범위를 2부터 10까지로, 이제 Wss와 CH index를 저장할 리스트를 생성한 후,

```
# K값별로 K-means 클러스터링을 실행하고 WSS와 CH index 계산
for (i in 1:length(k_values)) {
  k <- k_values[i]
  kmeans_result <- kmeans(pmatrix, centers = k, nstart = 2
5)

  # WSS 값 저장
  wss_values[i] <- kmeans_result$tot.withinss

  # CH Index 계산 (fpc 패키지 사용)
  stats <- cluster.stats(d = dist(pmatrix), kmeans_result$c
luster)
```

```
  ch_values[i] <- stats$ch  # Calinski-Harabasz Index
}
```

주어진 값에 따라 Kmeans클러스터링을 수행하며, 클러스터 내부 데이터 포인트간의 거리 총합인 WSS와 클러스터 결과에 대한 Ch index를 계산하고 저장함 이때 cluster.stats는 `fpc` **패키지**의 함수로, 다양한 클러스터링 평가 지표를 계산하는 함수임

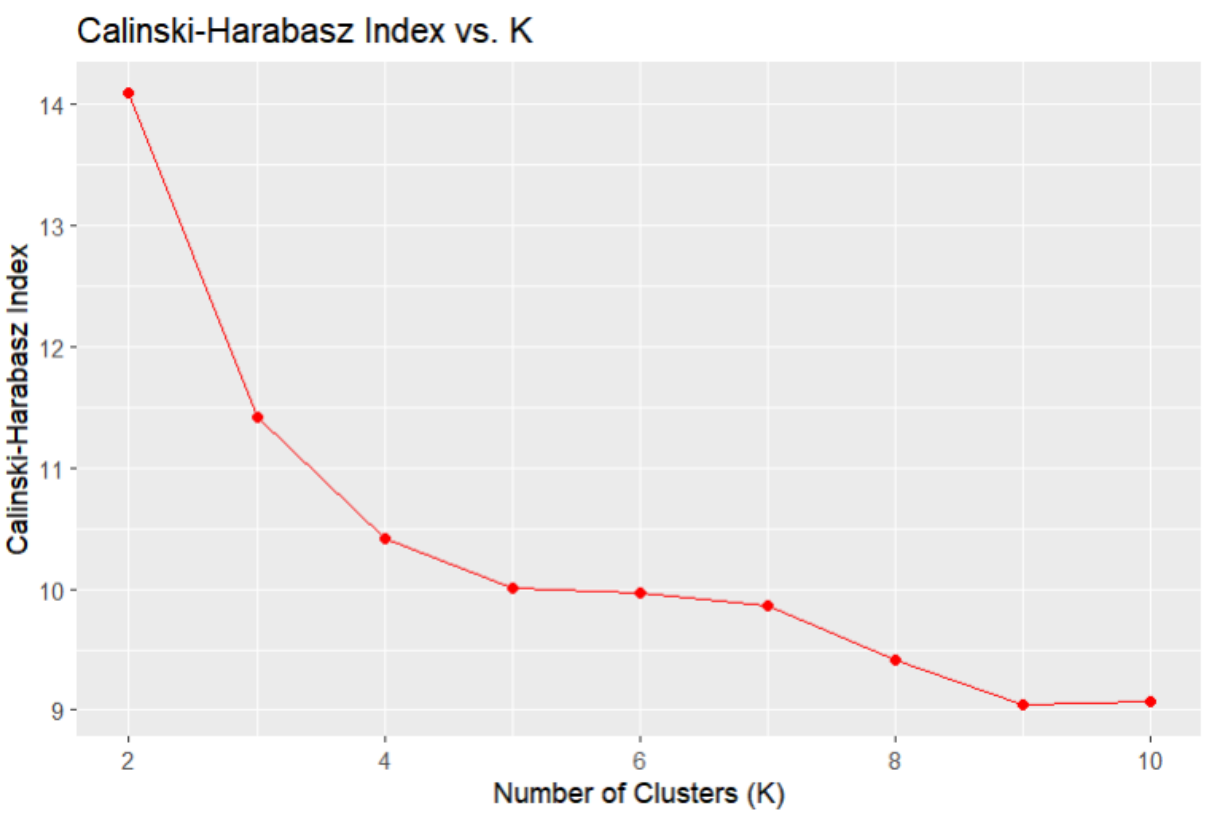
```
wss_df <- data.frame(K = k_values, WSS = wss_values)
ggplot(wss_df, aes(x = K, y = WSS)) +
  geom_line(color = "blue") +
  geom_point(color = "blue") +
  ggtitle("Within-Cluster Sum of Squares (WSS) vs. K") +
  xlab("Number of Clusters (K)") +
  ylab("WSS")
```



wss시각화, 클러스터의 갯수가 많아질 때 마다 WSS(클러스터 내부 포인트간 거리 총합)이 점점 줄어드는 것으로 관측된다. 이는 클러스터가 많아지고, 그 군집 내부의 갯수가 줄어들면서 내부 포인트의 거리가 줄어드는데, 지금 그래프를 보면 완만해지는 구간이 있다. Wss

값이 급격히 줄어들다가 완만한 경사를 가지는 부분이 클러스터의 갯수를 늘려도 WSS값이 크게 변하지 않게되는 지점으로 **엘보우**라고 부른다. k값이 2에서 4까지는 급격히 줄어들고 5 이상으로 올라가면서 점점 기울기가 줄어드는 것이 보인다. k = 4에서 엘보우 포인트가 보인다. 따라서 k = 4로 k-means 클러스터링을 수행하면 최적의 결과를 얻을 수 있다.

```
# CH Index 그래프 그리기
ch_df <- data.frame(K = k_values, CH = ch_values)
ggplot(ch_df, aes(x = K, y = CH)) +
  geom_line(color = "red") +
  geom_point(color = "red") +
  ggtitle("Calinski-Harabasz Index vs. K") +
  xlab("Number of Clusters (K)") +
  ylab("Calinski-Harabasz Index")
```



Ch index 시각화 Calinski-arabaseZ란 클러스터간 분산과, 클러스터 내 분산의 비율을 사용해 클러스터의 품질을 평가하는 지표로, 이 CH index값이 클수록 클러스터 내 데이터들이 모여있고, 각 클러스터 간 거리가 떨어져 있어 클러스터링 품질이 좋다는 것을 의미함

그래프를 확인해보면 클러스터의 갯수가 많아질수록 점점 값이 감소하게 되며 그 정도는 각 클러스터 마다 다르지만 k값이 2일때 가장 높은 점수를 보인다.

Task3 Euclidean Distance 외에 다른 종류의 다양한 distance measure (4개 이상이상)을 조사하시오

Manhattan Distance (맨하탄 거리)

Manhattan Distance는 각 차원의 차이의 절대값을 더하여 계산

```
# Manhattan distance 계산
manhattan_dist <- dist(pmatrix, method = "manhattan")
print(manhattan_dist)
```

장점: 유클리드 거리에 비해 **이상치**에 덜 민감함, **고차원 거리**에서 유클리드 거리보다 더 정확한 결과를 제공함

단점: 직선 거리가 아니라서, **연속적인 공간**에서의 이동을 정확히 측정하는 데는 부적합할 수 있음.

적용 예시: 도시 내 경로찾기

Cosine Similarity(코사인 유사도)
두 벡터 사이의
코사인 각도를 계산하여, 벡터의 방향이 얼마나 유사한지를 측정하는 방법

```
cosine_similarity <- proxy::simil(pmatrix, method = "cosin
e")
cosine_dist <- 1 - cosine_similarity
print(cosine_dist)
```

장점: 벡터의 방향을 비교하기 때문에, 크기보다는 **패턴의 유사성**을 측정할 수 있음

단점: 벡터 간의 크기 차이를 무시하기 때문에, 패턴이 아닌 크기가 중요한 경우에는 부적합할 수 있음

적용 예시: 텍스트 마이닝에서 문서간의 거리를 통해 비슷한 문서나 문장을 찾는데 용이함

Minkowski Distance (민코프스키 거리)

Euclidean Distance와 Manhattan Distance를 일반화한 거리 계산법

```
d_minkowski <- dist(pmatrix, method = "minkowski", p = 3)
d_minkowski[1:5]  # 상위 5개 거리 출력
```

장점: `p` 값을 유연하게 조정 가능 여기서 p값은 맨해튼, 유클리드 중 하나

단점: 선택한 p값에 따라 결과가 달라짐

예시: **유연한 거리 계산이 필요한 경우**에 사용됩니다. 다양한 거리 계산 방식 중에서 최적의 `p` 값을 찾을 수 있는 경우 유리

Jaccard Distance (자카드 거리)
두 데이터셋 간에
교집합과 합집합의 비율을 계산하여 거리로 변환

```
# Jaccard distance 계산
d_jaccard <- jaccard.dist(pmatrix)
d_jaccard[1:5, 1:5]  # 상위 5개의 거리 출력
```

장점: 이진 또는 범주형 데이터에 적합하며, 두 데이터셋 간의 유사성을 효과적으로 측정할 수 있음

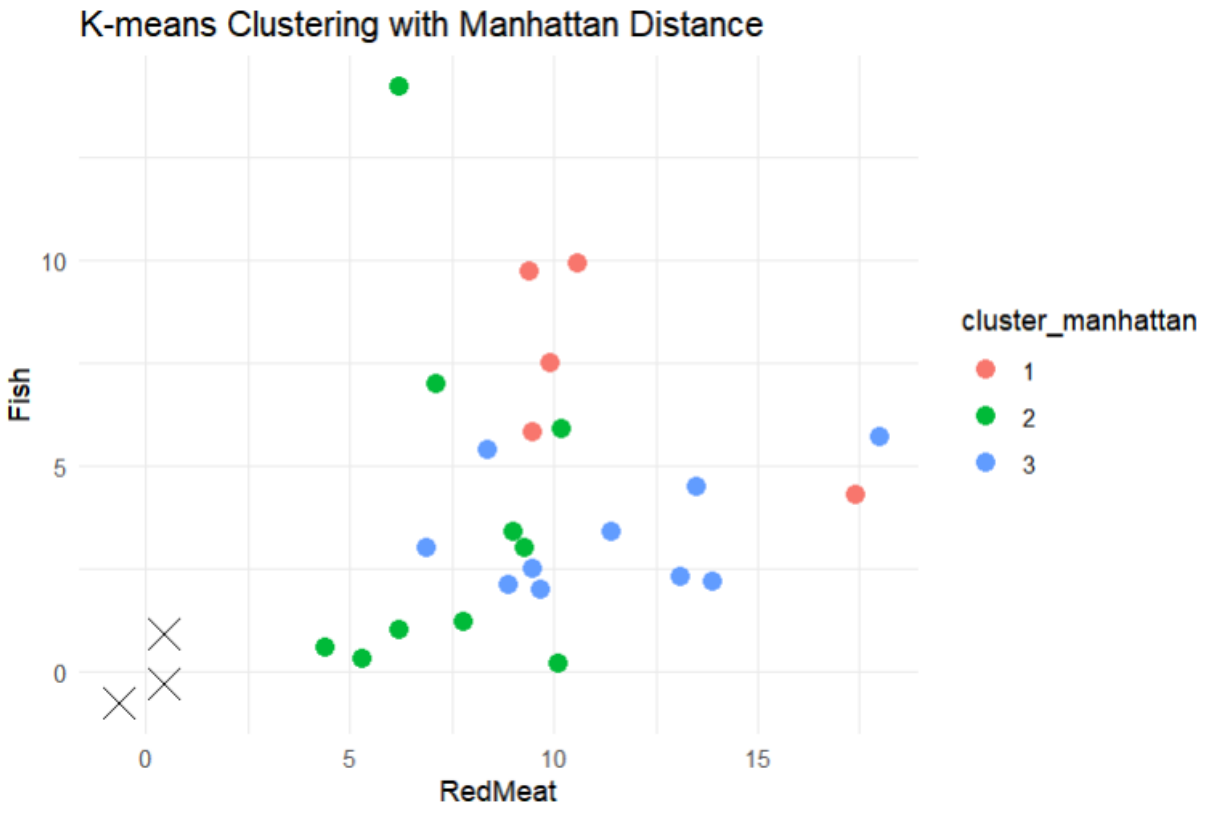
단점: 연속형 데이터에서는 사용하기 어려움

적용예시: 소셜 네트워크 분석, 사용자 관심사나 이벤트 간의 유사도 계산

맨해튼 거리 kmeans

```
manhattan_dist <- function(u, v) {
  sum(abs(u - v))
}

# 각 데이터 포인트를 가장 가까운 중심점에 할당하는 함수 (Manhattan Di
stance 사용)
assign_clusters_manhattan <- function(data, centers) {
  apply(data, 1, function(point) {
    distances <- apply(centers, 1, function(center) {
      manhattan_dist(point, center)
    })
    which.min(distances)  # 가장 가까운 중심점의 클러스터 번호 반
환
  })
}

# 중심점을 업데이트하는 함수
update_centers <- function(data, clusters, k) {
  centers <- matrix(0, nrow = k, ncol = ncol(data))
  for (i in 1:k) {
    centers[i, ] <- colMeans(data[clusters == i, , drop = F
ALSE])  # 각 클러스터 내의 데이터 평균 계산
  }
  centers
}

# K-means 클러스터링 실행 (Manhattan Distance 기반)
kmeans_manhattan <- function(data, k, max_iter = 100) {
  set.seed(123)
  centers <- data[sample(1:nrow(data), k), ]  # 초기 중심점 무
```

```
작위 선택
  clusters <- assign_clusters_manhattan(data, centers)  # 
초기 클러스터 할당

  for (i in 1:max_iter) {
    centers <- update_centers(data, clusters, k)  # 중심점 업
데이트
    new_clusters <- assign_clusters_manhattan(data, center
s)  # 클러스터 다시 할당

    if (all(new_clusters == clusters)) {
      break  # 클러스터가 변하지 않으면 종료
    }
    clusters <- new_clusters
  }

  list(clusters = clusters, centers = centers)
}

# 클러스터링 실행 (K = 3으로 설정)
k <- 3
kmeans_result_manhattan <- kmeans_manhattan(pmatrix, k)

# 클러스터링 결과 시각화
clusters <- kmeans_result_manhattan$clusters
centers <- kmeans_result_manhattan$centers

# 시각화 (첫 번째 두 개의 차원으로만 시각화)
protein$cluster_manhattan <- as.factor(clusters)
ggplot(protein, aes(x = RedMeat, y = Fish, color = cluster_
manhattan)) +
  geom_point(size = 3) +
  geom_point(data = as.data.frame(centers), aes(x = V1, y = 
V2), color = 'black', size = 5, shape = 4) +
  ggtitle("K-means Clustering with Manhattan Distance") +
  theme_minimal()
```



맨해튼 거리를 사용하여 K-means 클러스터링을 수행해 보았다.

세 개의 포인트에 대해 각각 절대값에 따라 계산이 되었으며, 1번인 붉은색은 RedMeat가 8 이상인 값들에 대해 Fish수가 상대적으로 높은 국가들이 포함되었고,

2번 녹색은 RedMeat가 4~11로 상대적으로 적으며 Fish지수가 비교적 아래에 있는 국가들이 포함되었다.

3번 파란색은 RedMeat지수가6이상이며 Fish지수가 중간에 위치한 그룹으로 확인이 되었다. 그러나 각 그룹별로 정확한 분류가 되었는지는 잘 모르겠다.

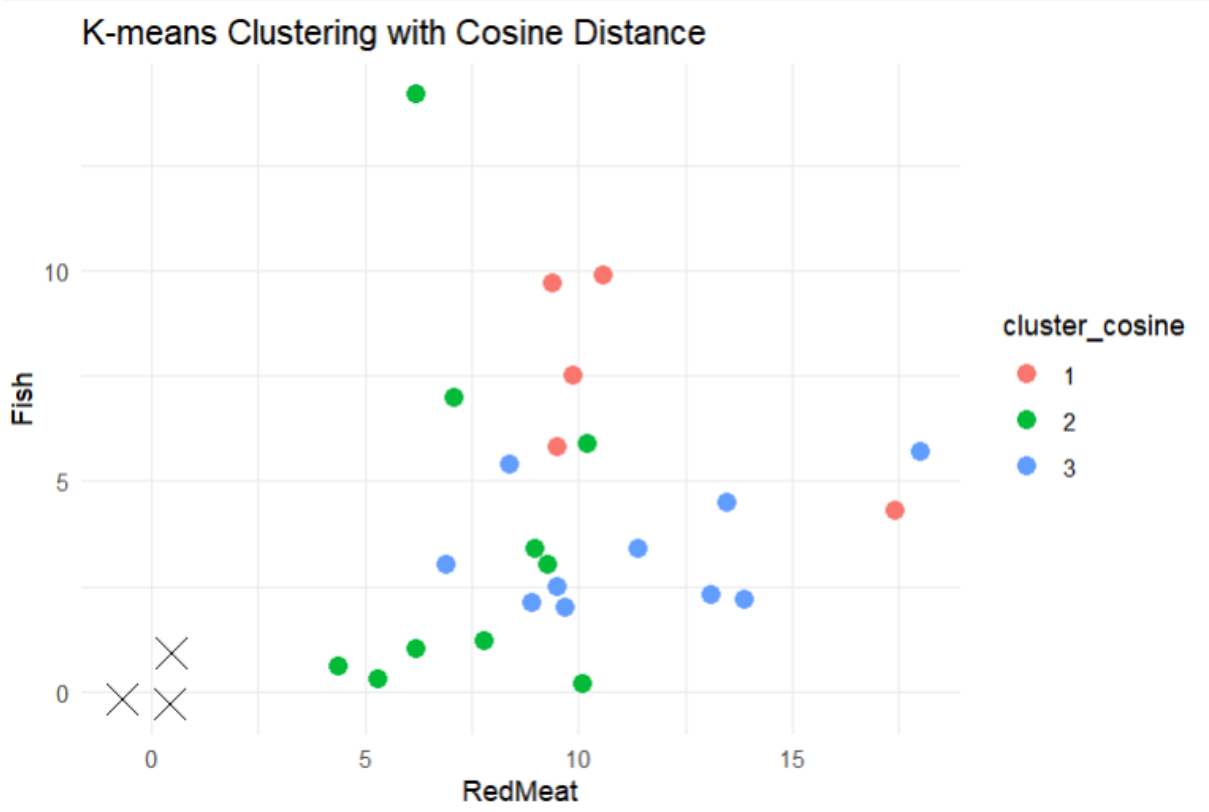
cosine_similarity 기반 K-means 클러스터링

```
# 데이터 준비 (표준화된 pmatrix 사용)
pmatrix <- scale(protein[, -1])  # 첫 번째 열 제외하고 데이터 표준화
```

```r
# 코사인 유사도 계산 함수 정의
cosine_similarity <- function(x, y) {
  sum(x * y) / (sqrt(sum(x^2)) * sqrt(sum(y^2)))
}

# 코사인 거리를 계산하는 함수 (1 - 코사인 유사도)
cosine_distance <- function(data) {
  n <- nrow(data)
  dist_matrix <- matrix(0, n, n)  # 거리 행렬 초기화

  for (i in 1:n) {
    for (j in 1:n) {
      if (i != j) {
        dist_matrix[i, j] <- 1 - cosine_similarity(data[i,
], data[j, ])  # 1 - 유사도 = 거리
      }
    }
  }

  as.dist(dist_matrix)  # 거리 행렬 반환
}

# 코사인 거리를 사용한 K-means 클러스터링 실행 함수
kmeans_cosine <- function(data, k, max_iter = 100) {
  set.seed(123)
  centers <- data[sample(1:nrow(data), k), ]  # 초기 중심점 무
작위 선택
  clusters <- rep(0, nrow(data))  # 클러스터 할당 초기화

  for (i in 1:max_iter) {
    # 각 데이터 포인트를 가장 가까운 중심점에 할당
    dist_matrix <- cosine_distance(rbind(centers, data))
    dist_matrix <- as.matrix(dist_matrix)[(k+1):(k+nrow(dat
a)), 1:k]
    new_clusters <- apply(dist_matrix, 1, which.min)  # 가장
가까운 중심점 찾기
```

```
    if (all(new_clusters == clusters)) {
      break  # 클러스터가 변하지 않으면 종료
    }

    clusters <- new_clusters
    for (j in 1:k) {
      centers[j, ] <- colMeans(data[clusters == j, , drop =
FALSE])  # 중심점 업데이트
    }
  }

  list(clusters = clusters, centers = centers)
}

# 클러스터링 실행 (K = 3으로 설정)
k <- 3
kmeans_result_cosine <- kmeans_cosine(pmatrix, k)

# 클러스터링 결과 시각화 준비
# 클러스터링 결과 시각화 준비
centers_cosine <- kmeans_result_cosine$centers  # 중심점 가져
오기
clusters_cosine <- kmeans_result_cosine$clusters  # 클러스터
할당 가져오기

# 필요한 열만 추출 (RedMeat와 Fish 열을 선택)
centers_df <- as.data.frame(centers_cosine)  # 행렬을 데이터프
레임으로 변환
colnames(centers_df) <- colnames(pmatrix)  # pmatrix의 열 이
름으로 설정
centers_df <- centers_df[, c("RedMeat", "Fish")]  # RedMeat
와 Fish 열만 선택

protein$cluster_cosine <- as.factor(clusters_cosine)

# ggplot을 사용한 시각화
ggplot(protein, aes(x = RedMeat, y = Fish, color = cluster_
cosine)) +
```

```
  geom_point(size = 3) +
  geom_point(data = centers_df, aes(x = RedMeat, y = Fish),
color = 'black', size = 5, shape = 4) +
  ggtitle("K-means Clustering with Cosine Distance") +
  theme_minimal()
```

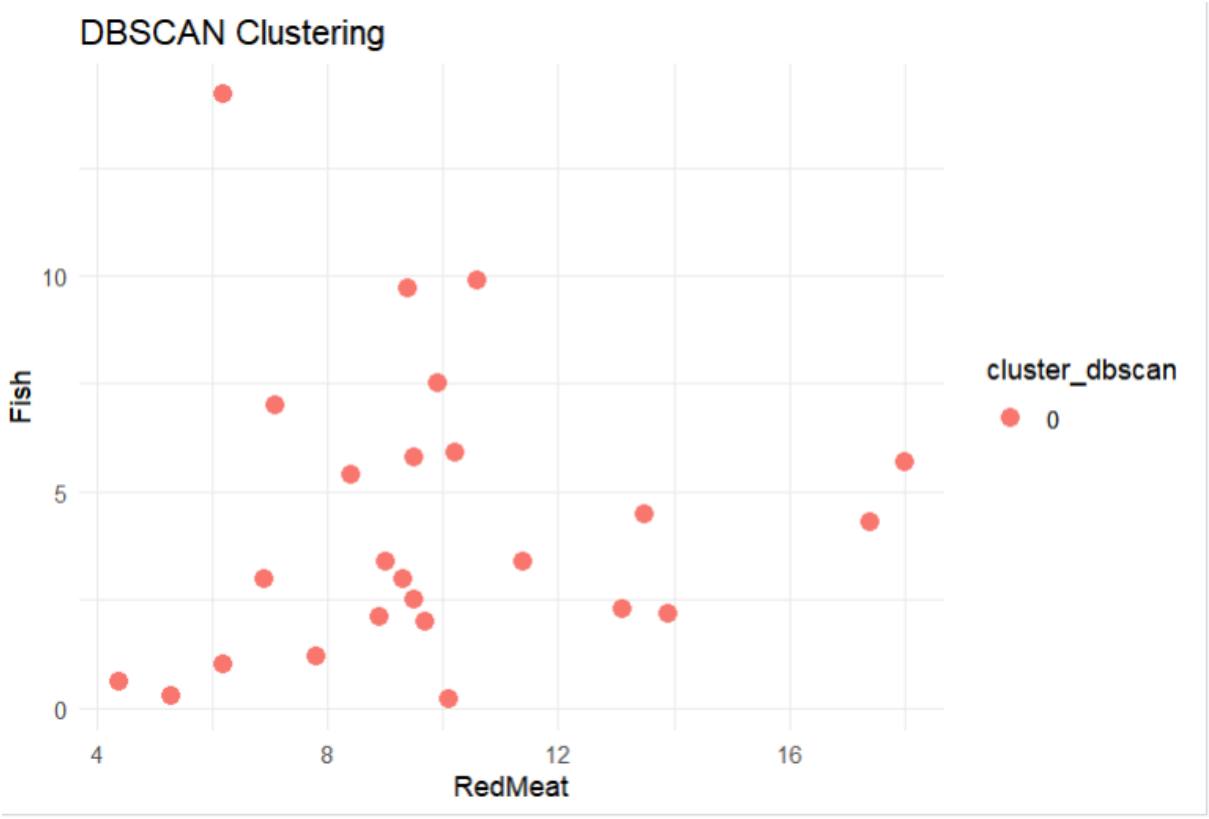


맨해탄 거리를 사용한 것과, 코사인 유사도에 따른 k-means 두가지 다 동일한 결과를 보인다.

Task 4 DB scan algorithm을 구현하여라 (Optional – 시연가능한 형태 extra credit)

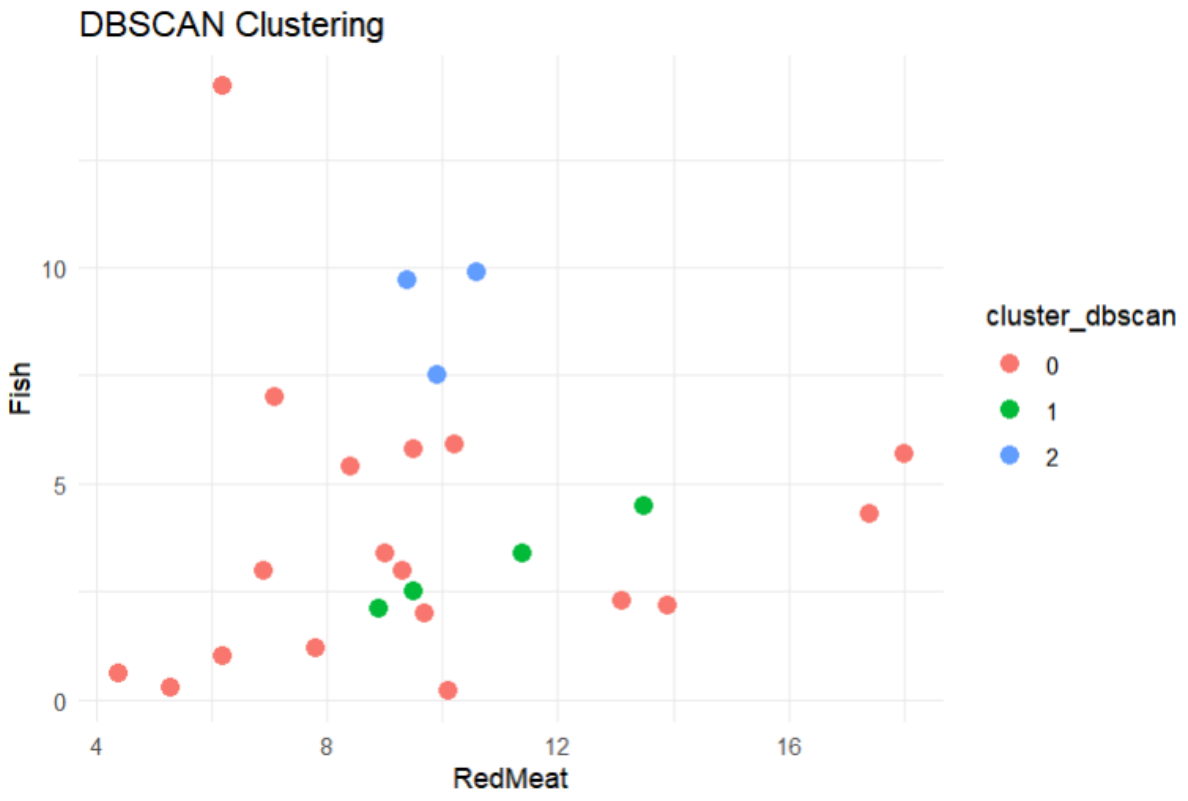
```
#DBScan
install.packages("dbscan")
library(dbscan)

# 데이터 준비 (protein 데이터를 사용한다고 가정)
```

```r
pmatrix <- scale(protein[, -1])  # 첫 번째 열 제외하고 데이터 표준화

# DBSCAN 클러스터링 수행
# eps: 거리 기준, minPts: 최소 이웃 포인트 수
dbscan_result <- dbscan(pmatrix, eps = 0.5, minPts = 5)

# 클러스터 결과 확인
dbscan_result$cluster

# DBSCAN 결과를 protein 데이터에 추가
protein$cluster_dbscan <- as.factor(dbscan_result$cluster)

# 클러스터링 결과 시각화 (RedMeat와 Fish 변수를 사용하여)
ggplot(protein, aes(x = RedMeat, y = Fish, color = cluster_dbscan)) +
  geom_point(size = 3) +
  ggtitle("DBSCAN Clustering") +
  theme_minimal()
```



DBScan은 밀도를 위주로 군집화하여 밀도가 낮은 부분은 노이즈로 처리하는 방법으로, 밀도에 따라 군집화를 하는데, 이 그래프상에서는 모든 값이 하나의 군집으로 묶인 것 같다.

```
pmatrix <- scale(protein[, 2:9])  # 수치형 열만 선택

# DBSCAN 클러스터링 수행
# eps: 거리 기준, minPts: 최소 이웃 포인트 수

dbscan_result <- dbscan(pmatrix, eps = 1.5, minPts = 3)

# 클러스터 결과 확인
dbscan_result$cluster

# DBSCAN 결과를 protein 데이터에 추가
protein$cluster_dbscan <- as.factor(dbscan_result$cluster)

# 클러스터링 결과 시각화 (RedMeat와 Fish 변수를 사용하여)
ggplot(protein, aes(x = RedMeat, y = Fish, color = cluster_
dbscan)) +
  geom_point(size = 3) +
  ggtitle("DBSCAN Clustering") +
  theme_minimal()
```



이번에는 밀도를 결정하는 요소인 eps와 minpts값을 바꾸었다.

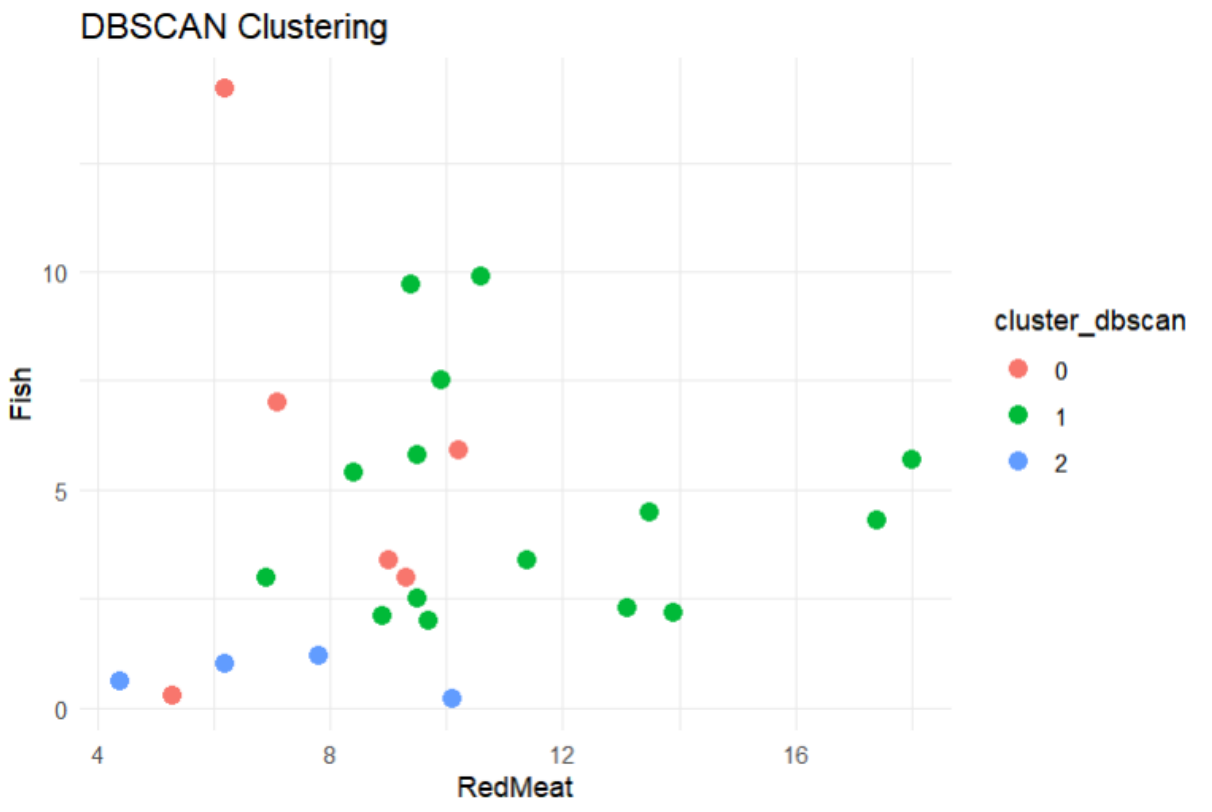
eps는 두 데이터 사이의 최대 거리를 나타내며, 같은 군집으로 묶이기 위해서는 각 거리가 eps값보다 작아야 한다

minpts는 하나의 클러스터가 되기 위한 최소 갯수를 의미하며, 최소 갯수를 3로 설정해서 현재와 같은 클러스터가 나온다.

이번에는 정확히 거리를 유클리드 거리로 계산하도록 통제한 상태에서 dbscan을 진행하겠다

```
pmatrix <- scale(protein[, 2:9])  # 수치형 열만 선택

# DBSCAN 클러스터링 수행
distance_matrix <- dist(pmatrix, method = "euclidean")
# eps: 거리 기준, minPts: 최소 이웃 포인트 수
dbscan_result <- dbscan(as.matrix(distance_matrix), eps =
5.0, minPts = 3)

# 클러스터 결과 확인
dbscan_result$cluster
```

```r
# DBSCAN 결과를 protein 데이터에 추가
protein$cluster_dbscan <- as.factor(dbscan_result$cluster)

# 클러스터링 결과 시각화 (RedMeat와 Fish 변수를 사용하여)
ggplot(protein, aes(x = RedMeat, y = Fish, color = cluster_
dbscan)) +
  geom_point(size = 3) +
  ggtitle("DBSCAN Clustering") +
  theme_minimal()
```



그래도 그나마 조금 비슷한 군집으로 묶인 것 같다.