Geostatistics: Spatial Model Draft for The Holy of The Holies

David Graupere Villà

graupere@gmail.com

Statistician

2024-09-02

The dataset temples provides information about the temples

id: This is a unique identifier for each observation. In this case, it contains the names of different archaeologists/researchers/explorers (“Conder”, “Patrich”, etc.)

lat: Latitude values indicating the geographical north-south position of each entity on the Earth’s surface. The values are in decimal degrees and correspond to the approximate location of temples.

lon: Longitude values indicating the geographical east-west position of each entity on the Earth’s surface, also in decimal degrees.

alt: The altitude of each entity or temple in meters, likely representing the height above sea level.

alt_topo: The topographic altitude of each temple, which might have been adjusted based on local terrain. This could be different from the regular altitude (alt) due to topographic corrections or measurements.

area: The area in some fixed unit (in this case, all values are 251 for each temple). It represent the area of the temple.

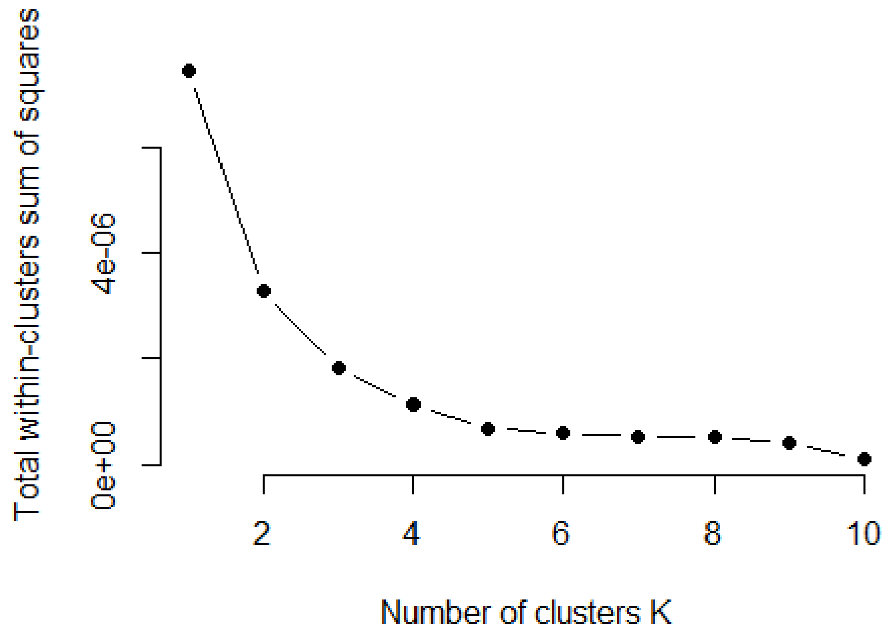
Type: This categorical variable indicates the type of the entity. In this dataset, it is set to “Temple” for all observations, meaning that each row represents a temple site.

```r
data0 <- data.frame(
  id = c("Conder", "Patrich", "Dalman", "De Vogue", "Ferguson", "Hollis", "Ka
ufman", "Mommert", "Ritmeyer", "Schick", "Sugiv", "Vincent", "Warren", "Watso
n", "Robertson"),
  lat = c(31.777881, 31.777910, 31.777896, 31.778269, 31.776902, 31.777995, 3
1.778745, 31.777925,
          31.777854, 31.777854, 31.776957, 31.777812, 31.777540, 31.777998, 3
1.776119),
  lon = c(35.235252, 35.235530, 35.234509, 35.234599, 35.234732, 35.235250, 3
5.234784, 35.234885,
          35.235389, 35.234629, 35.235491, 35.234514, 35.234772, 35.235410, 3
5.235012),
  alt = c(744,744 ,744 ,744 ,740 , 744, 744,744 , 744, 744, 739,744 , 743,744
,733 ),
  alt_topo = c(743,740, 728, 734, 707, 743, 737, 734, 743, 731, 731, 728, 725
, 743, 707),

  area = rep(251,15),
  Type = "Temple"
)
```

A1. First tool for determining the number of clusters in temples Elbow plot, which is commonly used to determine the optimal number of clusters (K) in k-means clustering.

Key Components of the Plot: X-axis (Number of clusters K): This represents the number of clusters, ranging from 2 to 10 in this case. Y-axis (Total within-clusters sum of squares): This shows the sum of squared distances between each data point and the centroid of its assigned cluster. The goal is to minimize this value to form tight clusters.

```
library(factoextra)

## Loading required package: ggplot2

## Welcome! Want to learn more? See two factoextra-related books at https://g
oo.gl/ve3WBa

wcss <- numeric()
for (k in 1:10) {
  set.seed(123)
  kmeans_model <- kmeans(data0[, c("lon", "lat")], centers = k)
  wcss[k] <- kmeans_model$tot.withinss
}

plot(1:10, wcss, type = "b", pch = 19, frame = FALSE,
     xlab = "Number of clusters K",
     ylab = "Total within-clusters sum of squares")
```



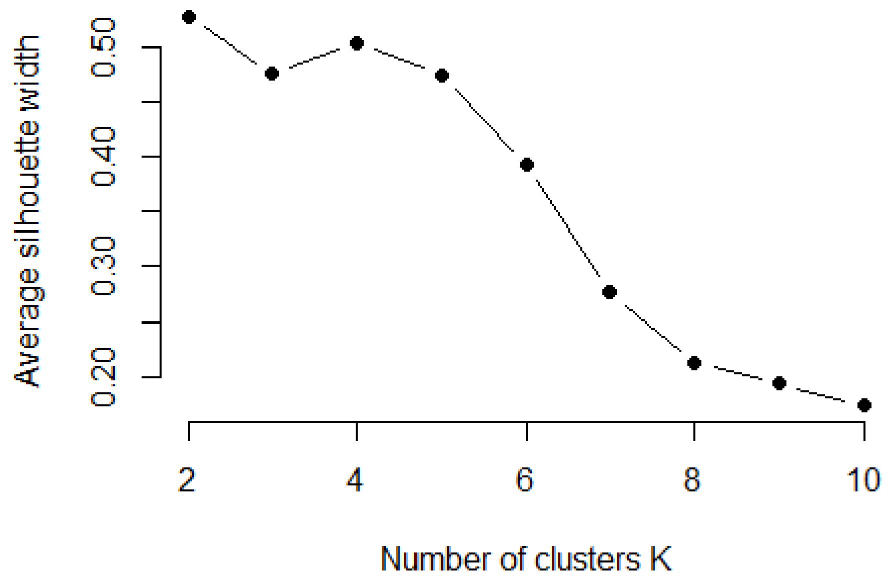
The plot shows a sharp decline in the total within-cluster sum of squares as the number of clusters increases, which is expected because as K increases, the data points are split into more clusters, reducing the distances within clusters. However, after a certain point (around K=3 or k=4), the rate of decrease slows down, forming an “elbow” shape. This suggests that adding more clusters beyond this point doesn’t significantly improve the clustering result.K=3 could also be a reasonable choice for the number of clusters if someone prioritizes a simpler model with fewer clusters.

A2. Second tool for determining the number of clusters in temples Calculate silhouette width for different numbers of clusters: this plot evaluates the quality of clustering for different numbers of clusters (K). The silhouette width measures how similar an object is to its own cluster compared to other clusters, where:

Values close to 1 indicate that the data point is well-clustered. Values close to 0 indicate that the data point lies between clusters. Negative values indicate that the data point is likely misclassified.

```r
library(cluster)
sil_width <- numeric()
for (k in 2:10) {
  set.seed(123)
  km <- kmeans(data0[, c("lon", "lat")], centers = k)
  sil <- silhouette(km$cluster, dist(data0[, c("lon", "lat")]))
  sil_width[k] <- mean(sil[, 3])
}
```

```r
plot(2:10, sil_width[-1], type = "b", pch = 19, frame = FALSE,
     xlab = "Number of clusters K",
     ylab = "Average silhouette width")
```

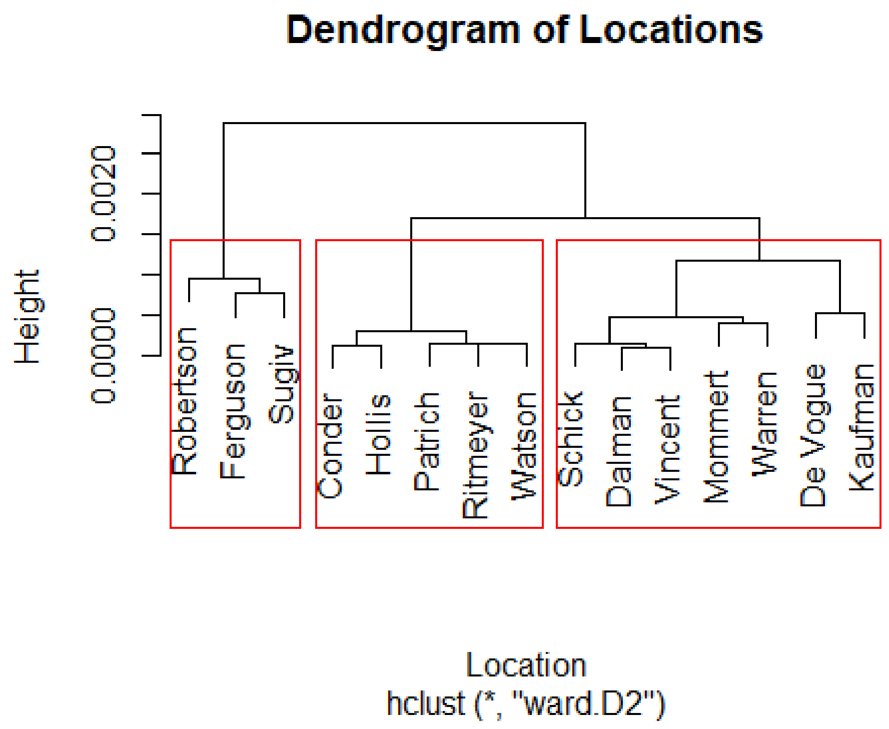


K=2 is likely the best choice in terms of having the highest average silhouette width, indicating well-separated clusters. If one is looking for more granularity and are willing to accept slightly less distinct clusters, K=3 could also be a valid option.

A3. Third tool for determining the number of clusters in temples The dendrogram is a visual representation of a hierarchical clustering analysis of locations based on their geographic coordinates (latitude and longitude). A dendrogram is a tree-like diagram that shows the arrangement of clusters formed by the hierarchical clustering algorithm. It helps us see how locations are progressively grouped together based on similarity (in this case, geographical proximity). ==>The y-axis represents height or distance (in this case, the geographical distance between locations) ==>The x-axis shows the locations involved in the clustering

```r
library(ggplot2)
library(dendextend)

##
## ---------------------
## Welcome to dendextend version 1.17.1
## Type citation('dendextend') for how to cite the package.
##
## Type browseVignettes(package = 'dendextend') for the package vignette.
## The github page is: https://github.com/talgalili/dendextend/
```

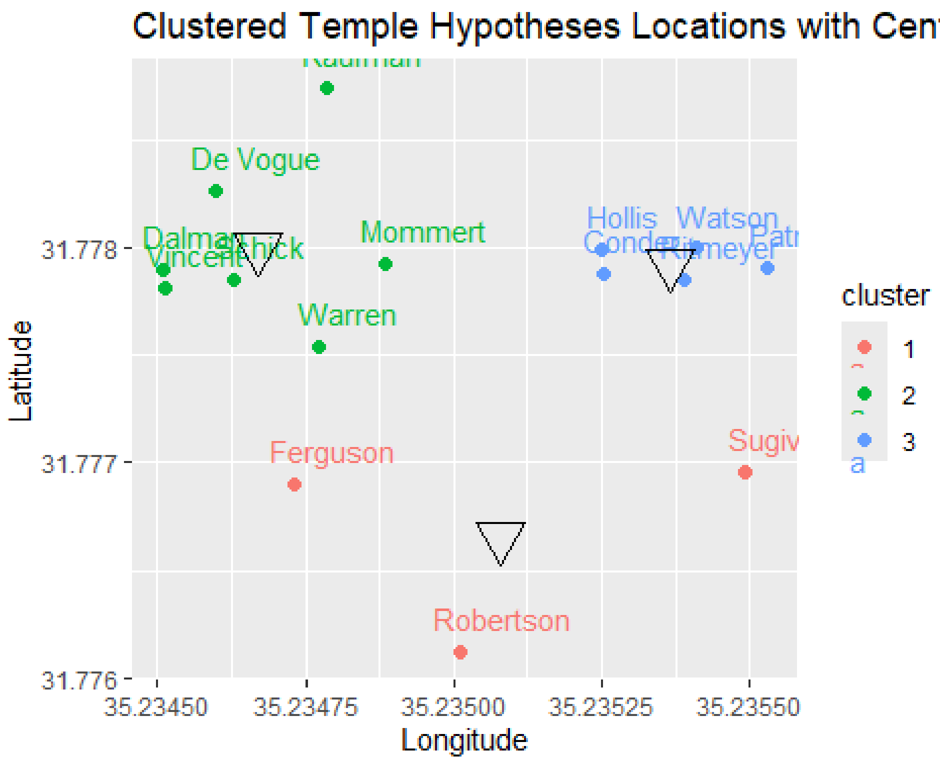
```
##
## Suggestions and bug-reports can be submitted at: https://github.com/talgal
ili/dendextend/issues
## You may ask questions at stackoverflow, use the r and dendextend tags:
##   https://stackoverflow.com/questions/tagged/dendextend
##
##  To suppress this message use:  suppressPackageStartupMessages(library(den
dextend))
## ---------------------

##
## Attaching package: 'dendextend'

## The following object is masked from 'package:stats':
##
##     cutree

distance_matrix <- dist(data0[, c("lat", "lon")])
hclust_result <- hclust(distance_matrix, method = "ward.D2")
plot(hclust_result, labels = data0$id, main = "Dendrogram of Locations", xlab
= "Location", ylab = "Height")

rect.hclust(hclust_result, k = 3, border = "red")
```



The number of clusters can be chosen based on the natural breaks in the dendrogram (where large vertical gaps exist between merges) or by domain-specific knowledge (such as the historical or geographical significance of certain locations). A significant vertical gap occurs between the

3 larger groups, making 3 clusters a reasonable choice: Cluster 1 (left group): Contains "Robertson," "Ferguson," and "Sugiv." Cluster 2 (middle group): Contains "Conder," "Hollis," "Patrich," "Ritmeyer," and "Watson." Cluster 3 (right group): Contains "Schick," "Dalman," "Vincent,", "Mommert", "Warren," "De Vogüe," and "Kaufman."

```
optimal_k <- 3

set.seed(123)
clusters <- kmeans(data0[, c("lon", "lat")], centers = optimal_k, nstart = 50
0)

data0$cluster <- as.factor(clusters$cluster)

cluster_centers <- data.frame(clusters$centers)
ggplot(data0, aes(x = lon, y = lat, color = cluster)) +
  geom_point(size = 2) +
  geom_text(aes(label = id), vjust = -0.9, hjust = 0.2) +
  geom_point(data = cluster_centers, aes(x = lon, y = lat), color = "black",
size = 6, shape = 6) +
  labs(title = "Clustered Temple Hypotheses Locations with Centers",
       x = "Longitude",
       y = "Latitude")
```



The clustering of locations might suggest that certain types of structures or artifacts are likely to be found in proximity to each other. The centroids (triangle markers) indicate the most central or representative location for each cluster, which could help archaeologists prioritize areas for excavation or further investigation. The distinct groups could represent different types of temples or structures, or even different periods of construction.

==>Archaeologists should focus on the centroids of each cluster, especially if these centroids represent potential temple sites based on prior hypotheses. ==>The spatial distribution of clusters (north vs. south) could hint at differences in temple layout or positioning based on environmental factors (like altitude, terrain, or proximity to water). ==>The central cluster (green) may be of particular interest as it contains a wide spread of locations, indicating possible diversity in structures or functions of sites within this area.

The dataset centroids provides information about the cisterns (water storage structures) in a specific geographic area. It consists of the following variables: ==>cistern: A numeric ID for each cistern or location, presumably representing different cisterns or relevant locations in the area (the numbers are from Ritmeyer's documents). ==>lat: The latitude coordinates of each cistern location, indicating the north-south position on the globe. ==>lon: The longitude coordinates of each cistern location, representing the east-west position on the globe.

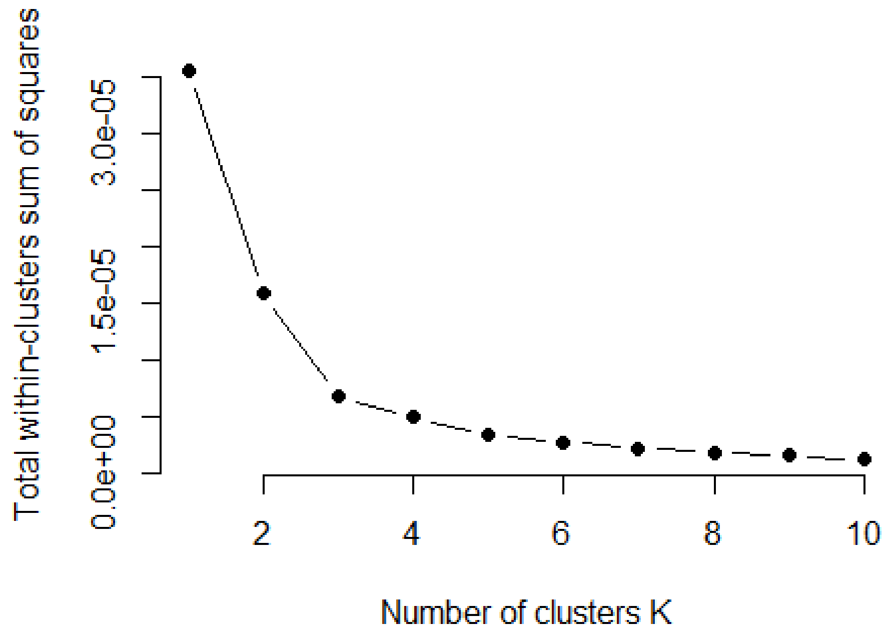
This dataset contains the geographic positions (latitude and longitude) of 24 cisterns, which can be used for spatial analysis.

```
centroids <- data.frame(

  cistern = c(23, 28, 15, 3, 2, 37, 34, 14, 10, 5, 7, 8, 31, 12, 25, 4, 5.1,
13, 11, 9, 32, 6, 36, 20),

  lat = c(
    31.7788857145598200, 31.7790639543329014, 31.7790102802945995, 31.7785680
613983992,
    31.7786333598254984, 31.7785697817048991, 31.7785759954760998, 31.7786486
569582003,
    31.7760007038991006, 31.7776242398734006, 31.7769479392513006, 31.7768120
092897988,
    31.7782008052429994, 31.7784174421062993, 31.7783203020720002, 31.7780654
357854004,
    31.7774787431775003,
    31.7785528878747989, 31.7765322104473000, 31.7764633536541012,
    31.7763211333066984, 31.7770055630779993, 31.7769595759471990, 31.7762468
320380016
  ),

  lon = c(
    35.2344290103842184, 35.2362438016556965, 35.2366602961786981, 35.2348996
294374004,
    35.2358879234683968, 35.2362771090801985, 35.2360127426784970, 35.2365811
685583026,
    35.2364301179070978, 35.2359717620989983, 35.2363416998961014, 35.2358669
316085980,
    35.2345107853488031, 35.2366948439786967, 35.2347202983246035, 35.2348491
515088966,
    35.2360544791140029,
    35.2366136251617021, 35.2363622442846989, 35.2357517684176997,
    35.2357938308960001, 35.2356618251995997, 35.2352593102747988, 35.2347345
864651018
  )
)
```

B1. First tool for determining the number of clusters Elbow plot, which is commonly used to determine the optimal number of clusters (K) in k-means clustering.

Key Components of the Plot: X-axis (Number of clusters K): This represents the number of clusters, ranging from 2 to 10 in this case. Y-axis (Total within-clusters sum of squares): This shows the sum of squared distances between each data point and the centroid of its assigned cluster. The goal is to minimize this value to form tight clusters.

```
library(spdep)

## Warning: package 'spdep' was built under R version 4.4.1

## Loading required package: spData

## Warning: package 'spData' was built under R version 4.4.1

## To access larger datasets in this package, install the spDataLarge
## package with: `install.packages('spDataLarge',
## repos='https://nowosad.github.io/drat/', type='source')`

## Loading required package: sf

## Warning: package 'sf' was built under R version 4.4.1

## Linking to GEOS 3.12.1, GDAL 3.8.4, PROJ 9.3.1; sf_use_s2() is TRUE

coords <- as.matrix(centroids[, c("lat", "lon")])

wss <- sapply(1:10, function(k){
  kmeans(coords, k, nstart=10)$tot.withinss
})

plot(1:10, wss, type="b", pch=19, frame=FALSE,
     xlab="Number of clusters K", ylab="Total within-clusters sum of squares"
)
```



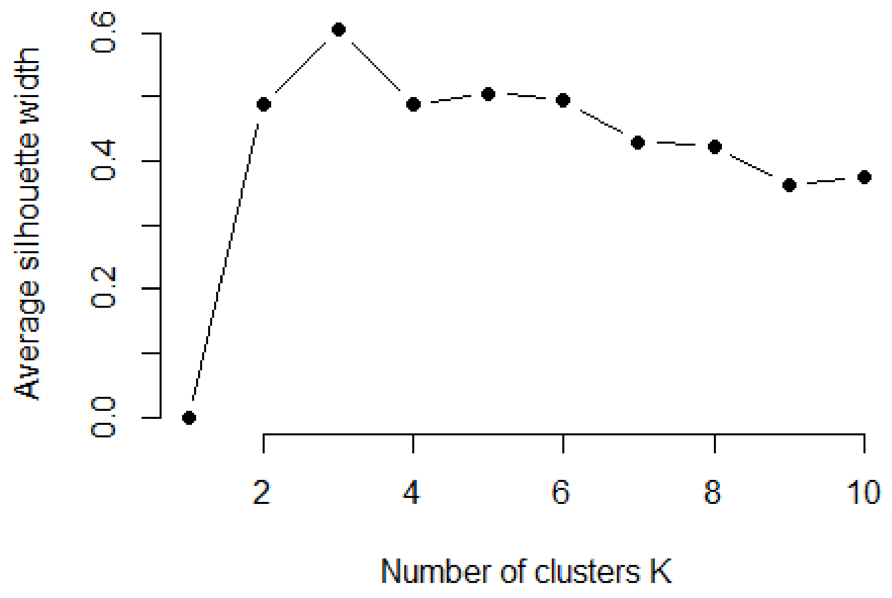
The plot shows a sharp decline in the total within-cluster sum of squares as the number of clusters increases, which is expected because as K increases, the data points are split into more clusters, reducing the distances within clusters. However, after a certain point (around K=3), the rate of decrease slows down, forming an "elbow" shape. This suggests that adding more clusters beyond this point doesn't significantly improve the clustering result.

B2. Second tool for determining the number of clusters Calculate silhouette width for different numbers of clusters: this plot evaluates the quality of clustering for different numbers of clusters (K). The silhouette width measures how similar an object is to its own cluster compared to other clusters, that is to say it's used to evaluate the quality of clustering by measuring how well-separated the clusters are for each value of K, where:

Values close to 1 indicate that the data point is well-clustered. Values close to 0 indicate that the data point lies between clusters. Negative values indicate that the data point is likely misclassified.

```r
library(cluster)

coords <- as.matrix(centroids[, c("lat", "lon")])

sil_width <- numeric(10)
for (k in 2:10) {
  set.seed(777)
  km_res <- kmeans(coords, centers = k, nstart = 25)
  sil <- silhouette(km_res$cluster, dist(coords))
  sil_width[k] <- mean(sil[, 3])
}
```

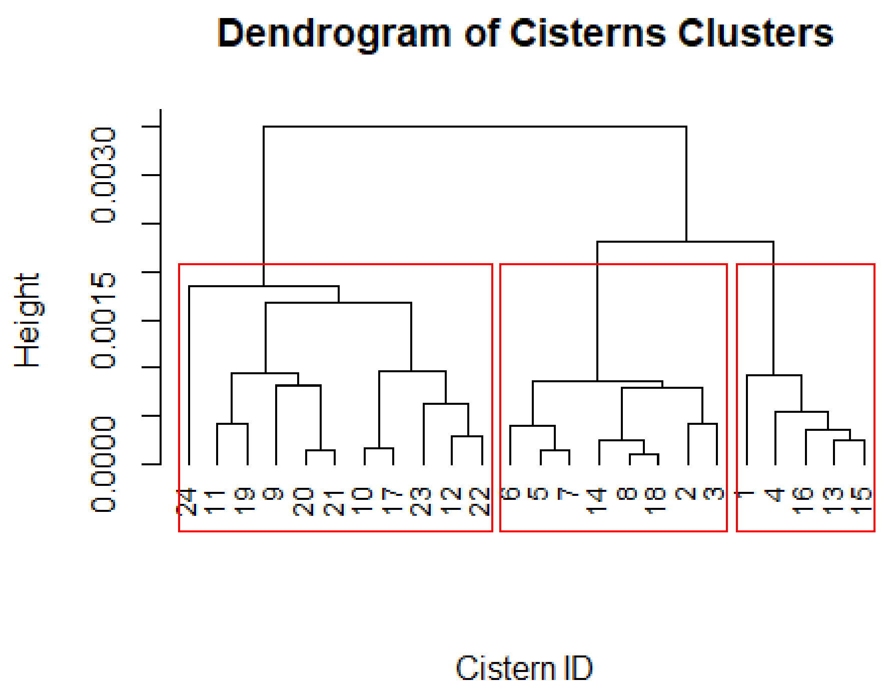
```
plot(1:10, sil_width, type = "b", pch = 19, frame = FALSE,
     xlab = "Number of clusters K", ylab = "Average silhouette width")
```



K = 3: The highest silhouette width occurs at K = 3, which suggests that splitting the data into three clusters results in the best-defined clusters, where data points are well-separated from points in other clusters

B3. Third tool for determining the number of clusters The dendrogram is a visual representation of a hierarchical clustering analysis of locations based on their geographic coordinates (latitude and longitude). A dendrogram is a tree-like diagram that shows the arrangement of clusters formed by the hierarchical clustering algorithm. It helps us see how locations are progressively grouped together based on similarity (in this case, geographical proximity). ==>The y-axis represents height or distance (in this case, the geographical distance between locations) ==>The x-axis shows the locations involved in the clustering

```
if (!require(cluster)) install.packages("cluster", dependencies=TRUE)

coords <- as.matrix(centroids[, c("lat", "lon")])

dist_matrix <- dist(coords)

hc <- hclust(dist_matrix, method = "complete")

plot(hc, main = "Dendrogram of Cisterns Clusters",
     xlab = "Cistern ID", sub = "", cex = 0.9, hang = -1)

rect.hclust(hc, k = 3, border = "red")
```



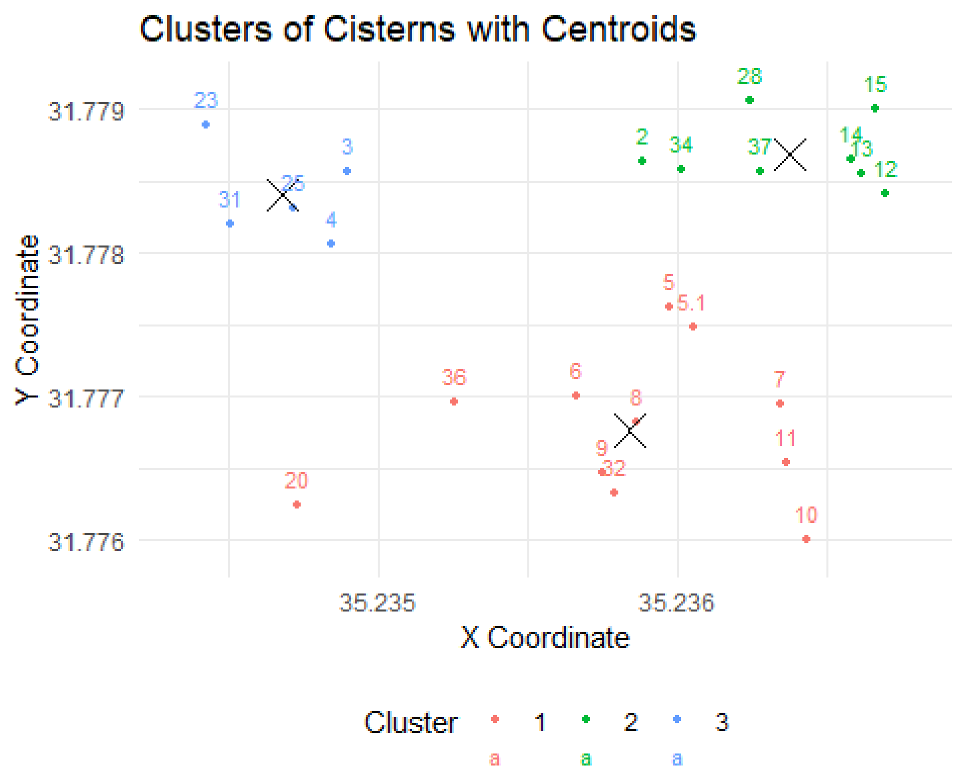
Visual Inspection of the Dendrogram: In the dendrogram, one can observe clear separation between branches at three distinct locations. These points represent natural divisions within the data, where cisterns in the same branch are more similar to each other than to objects in other branches.

Height Threshold: The height of the vertical lines in the dendrogram indicates the distance or dissimilarity between the clusters. By cutting the dendrogram at a height where there are three major branches, you capture a natural clustering pattern without grouping highly dissimilar points together.

Interpretation of Clustering: Three clusters often provide a balance between simplicity and meaningful differentiation. In this case, these clusters might represent different groups of cisterns based on their geographic location, allowing for better analysis and insights into their spatial distribution.

```
set.seed(42)
optimal_clusters <- 3
kmeans_result <- kmeans(coords, centers=optimal_clusters, nstart=50)

centroids$Cluster <- kmeans_result$cluster

library(ggplot2)
```

```r
lon_min <- min(centroids$lon) - 0.0001
lon_max <- max(centroids$lon) + 0.0001
lat_min <- min(centroids$lat) - 0.0001
lat_max <- max(centroids$lat) + 0.0001


ggplot(centroids, aes(x = lon, y = lat, color = as.factor(Cluster))) +
  geom_point(size = 1) +  # Plot data points
  geom_point(data = aggregate(. ~ Cluster, centroids, mean),
             aes(x = lon, y = lat), color = "black", size = 5, shape = 4) +
  geom_text(aes(label = cistern), vjust = -1, size = 3) +
  # Plot centroids
  labs(title = "Clusters of Cisterns with Centroids",
       x = "X Coordinate",
       y = "Y Coordinate",
       color = "Cluster") +
  xlim(lon_min, lon_max) +  # Set x-axis limits
  ylim(lat_min, lat_max) +  # Set y-axis limits
  theme_minimal() +
  theme(legend.position = "bottom")
```



The cisterns are grouped into three clusters, and their centroids (marked by "X") are shown for each cluster.The clustering helps to identify different regions where the cisterns are located, which can provide valuable insights for further geographic or archaeological analysis.

Cluster 1 (red points): Cisterns in this cluster tend to be located towards the southern part of the plot, with lower latitudes. This cluster is more dispersed horizontally (longitude), covering a wide area. The centroid of Cluster 1 is located slightly west and south of the main body of cisterns in this cluster. ==>Cluster 1 is the most widely dispersed cluster, suggesting that the cisterns in this group are spread across a larger geographical area.

Cluster 2 (green points): These cisterns are located towards the northeastern part of the plot, with higher longitudes and latitudes. The points in this cluster seem to be closely grouped together, indicating they are spatially closer. The centroid of Cluster 2 lies near the center of the group, reflecting the tight grouping of the cisterns in this cluster. ==>Cluster 2 is more densely packed, which might indicate a concentration of cisterns in this specific region.

Cluster 3 (blue points): This cluster is situated towards the northwestern part of the plot and is less dispersed compared to Cluster 1. The cisterns are grouped relatively close to each other. Centroids: The centroid of Cluster 3 is located centrally within the cluster, indicating that the cisterns in this group are similarly spaced out from the center. ==>Cluster 3 shows a relatively compact group, similar to Cluster 2 but located in the northwestern region.

1.1 Temple Data (data) Variables: ID: This is a unique identifier for each observation. In this case, it contains the names of different archaeologists/researchers/explorers ("Conder", "Patrich", etc.)

lat (Latitude): The geographic latitude of each site, expressed in decimal degrees. Latitude values describe the north-south position of a point on Earth's surface. In this dataset, all locations are close to latitude 31.777 to 31.778, which is likely in the region of Jerusalem, Israel. lon (Longitude):

The geographic longitude of each site, expressed in decimal degrees. Longitude values describe the east-west position of a point on Earth's surface. The longitude values in this dataset are around 35.234 to 35.236.

alt (Altitude): This represents the elevation or altitude of the site in meters above sea level. In this dataset, the altitude of these locations ranges from 733 to 744 meters.

alt_topo (Topographic Altitude): This refers to a specific topographic measurement of the altitude of the sites. It may be a more accurate or refined altitude measure adjusted for terrain features. This variable typically reflects the natural terrain or topographic elevation after considering surrounding landforms.

area: The area associated with the site in square meters (or another unit). For the temple data, all sites have the same area value of 251, which may represent a standardized or approximate area size for these temple locations.

Type: A categorical variable that identifies the type of the structure. In this case, all entries in the data dataframe are classified as "Temple", indicating that these are temple structures.

cluster: This variable assigns each temple location to a specific cluster.

1.2 Cistern Data (centroids) Variables:

ID: Identifiers for the cistern locations.

lat (Latitude): The geographic latitude of each cistern site, in decimal degrees. The latitude values are similar to the temple dataset but may have a wider spread, ranging from 31.776 to 31.779.

lon (Longitude): The geographic longitude of each cistern site, in decimal degrees. Like the temple data, the longitude values range from approximately 35.234 to 35.236.

alt (Altitude): The elevation of the cistern sites in meters above sea level. The values range from 725 to 744 meters, similar to the temple data, reflecting the variation in elevation of the landscape where the cisterns are located.

alt_topo (Topographic Altitude): Like in the temple data, this reflects the topographic altitude for each cistern site. It may reflect natural or topographical measurements of elevation, accounting for the surrounding landscape.

area: The area associated with each cistern location, expressed in square meters (or another unit). The cisterns have varying area values, unlike the temples, which all had the same area. These areas likely represent the surface area of the cisterns, ranging from 9.23 to over 1400.

Type: This identifies the structure type as "Cistern" for all the entries in this dataframe, indicating that these sites are cisterns (water storage structures).

cluster: This variable assigns cistern locations to clusters

```
data <- data.frame(ID = c("Conder", "Patrich", "Dalman", "De Vogue", "Ferguso
n", "Hollis", "Kaufman", "Mommert", "Ritmeyer", "Schick", "Sugiv", "Vincent",
"Warren", "Watson", "Robertson"),
  lat = c(31.777881, 31.777910, 31.777896, 31.778269, 31.776902, 31.777995, 3
1.778745, 31.777925, 31.777854, 31.777854, 31.776957, 31.777812, 31.777540, 3
1.777998, 31.776119),
  lon = c(35.235252, 35.235530, 35.234509, 35.234599, 35.234732, 35.235250, 3
5.234784, 35.234885, 35.235389, 35.234629, 35.235491, 35.234514, 35.234772, 3
5.235410, 35.235012),
  alt = c(744,744,744,744,740,744,744,744,744,744,739,744,743,744,733),
  alt_topo = c(743,740, 728,734,707,743,737,734,743,731,731, 728, 725,743,707
),
  area = rep(251, 15),
  Type = rep("Temple", 15)
)


temple_clusters <- c( 6 ,6 ,4 ,4 ,5 ,6 ,4 ,4 ,6 ,4 ,5 ,4 ,4 ,6 ,5)

data$cluster <- temple_clusters
```

```r
centroids <- data.frame(
  ID = c(23, 28, 15, 3, 2, 37, 34, 14, 10, 5, 7, 8, 31, 12, 25, 4, 5.1, 13, 1
1, 9, 32, 6, 36, 20),
  lat = c(
    31.7788857145598200, 31.7790639543329014, 31.7790102802945995, 31.7785680
613983992,
    31.7786333598254984, 31.7785697817048991, 31.7785759954760998, 31.7786486
569582003,
    31.7760007038991006, 31.7776242398734006, 31.7769479392513006, 31.7768120
092897988,
    31.7782008052429994, 31.7784174421062993, 31.7783203020720002, 31.7780654
357854004,
    31.7774787431775003, 31.7785528878747989, 31.7765322104473000, 31.7764633
536541012, 31.7763211333066984, 31.7770055630779993, 31.7769595759471990,
    31.7762468320380016
  ),
  lon = c(
    35.2344290103842184, 35.2362438016556965, 35.2366602961786981, 35.2348996
294374004,
    35.2358879234683968, 35.2362771090801985, 35.2360127426784970, 35.2365811
685583026,
    35.2364301179070978, 35.2359717620989983, 35.2363416998961014, 35.2358669
316085980,
    35.2345107853488031, 35.2366948439786967, 35.2347202983246035, 35.2348491
515088966,
    35.2360544791140029, 35.2366136251617021, 35.2363622442846989, 35.2357517
684176997, 35.2357938308960001, 35.2356618251995997, 35.2352593102747988,
    35.2347345864651018
  ),
  alt = c(744, 742, 737, 744, 741, 739, 741, 737, 725, 742, 737, 737, 744, 73
7, 744, 744, 742, 737, 732, 734, 733, 739, 740, 733),

  alt_topo = c(737, 734, 728, 737, 737, 737, 737, 734, 725, 737, 734, 734, 72
5, 731, 734, 734, 737, 734, 731, 728, 725, 734, 728, 704),

  area = c(16.912, 16.507, 47.364, 31.089, 166.316, 150.695, 126.706, 66.453,
547.808, 630.150, 1005.735, 1409.273, 18.308, 100.836, 16.928, 21.349, 9.230,
78.536, 739.177, 255.637, 11.289, 75.088, 181.703, 282.028),
  Type = "Cistern"
)
cistern_clusters <- c(3, 2, 2, 3, 2, 2, 2, 2, 1, 1, 1, 1, 3, 2, 3, 3, 1, 2, 1
, 1, 1, 1, 1, 1)
centroids$cluster <- cistern_clusters




# Combine data frames to match weights
overall_data <- rbind(data, centroids)
```

```
overall_data$Type_numeric <- ifelse(overall_data$Type == "Temple", 1, 0)
#overall_data$region_id <- 1:nrow(overall_data)
```

2.1. Load Necessary Libraries Matrix: This package provides efficient implementations for sparse and dense matrix operations. spatialreg: A package for spatial econometrics and regression modeling, especially for spatial autoregressive models (SAR) like the spatial lag model used here. spdep: A package for analyzing spatial data and constructing spatial weights matrices, which are essential for spatial regression models like the one being built. *2.2.Create Coordinates Matrix for Spatial Weighting* coordinates: This is a matrix created from the latitude (lat) and longitude (lon) columns of the overall_data dataset. The matrix has two columns: lat and lon, representing the spatial location of each observation (temple or cistern). This matrix is necessary for defining spatial relationships (i.e., which observations are neighbors to one another). *2.3 Create Spatial Weights Matrix* knearneigh(coordinates, k = 2):

This function calculates the k-nearest neighbors for each observation, based on their coordinates. k = 2 means each observation is connected to its 2 nearest neighbors in geographic space. This defines the spatial connectivity between observations. knn2nb(): Converts the nearest-neighbor object into a neighbor list (nb object), which is a standard format used in spatial econometrics to describe which locations are neighbors.

nb2listw(): This function converts the neighbor list into a spatial weights matrix (listw object).

style = “W” means the weights matrix will use row-standardized weights, which means the sum of weights for each observation is 1 (so that each observation’s neighbors equally contribute to its spatial influence). listw_alternate: This is the resulting spatial weights matrix, which is used to define the strength of spatial interactions in the spatial regression model.

2.4 Fit a Spatial Lag Model lagsarlm(): This function fits a spatial autoregressive lag model (SAR).

In this model, the value of the dependent variable at each location depends on a weighted average of the values at nearby locations (based on the spatial weights matrix). alt_topo ~ (lat+lon):Type:

This is the formula for the model. alt_topo is the dependent variable (the topographic altitude). The independent variables include an interaction term between lat and lon with the variable Type. The interaction allows the model to account for how the latitude and longitude affect the altitude differently depending on whether the site is a temple or a cistern. data = overall_data: The data being used in the model comes from the overall_data dataset, which contains information on both temple and cistern sites.

listw = listw_alternate: This tells the model to use the spatial weights matrix created earlier (listw_alternate) to capture the spatial dependencies between the observations.

2.5 Moran's I test Moran's I test is appropriate after fitting a spatial lag model. Moran's I is a measure of spatial autocorrelation in the residuals, and it helps assess whether there is remaining spatial dependence that hasn't been captured by the model.

In your code, you are already fitting a spatial lag model, which accounts for spatial autocorrelation in the response variable. However, it is still important to check if there is any remaining spatial autocorrelation in the residuals of your model. The Moran test can help identify whether your model has sufficiently captured the spatial structure.

Here's how you can conduct a Moran's I test on the residuals of your spatial lag model:

Steps to add the Moran's I test Extract the residuals from your fitted model. Run Moran's I test on the residuals to check for remaining spatial autocorrelation.

```
#2.1
library(Matrix)
library(spatialreg)

## Warning: package 'spatialreg' was built under R version 4.4.1

##
## Attaching package: 'spatialreg'

## The following objects are masked from 'package:spdep':
##
##     get.ClusterOption, get.coresOption, get.mcOption,
##     get.VerboseOption, get.ZeroPolicyOption, set.ClusterOption,
##     set.coresOption, set.mcOption, set.VerboseOption,
##     set.ZeroPolicyOption

library(spdep)

#2.2
coordinates <- as.matrix(overall_data[, c("lat", "lon")])

#2.3
listw_alternate <- nb2listw(knn2nb(knearneigh(coordinates, k = 2)), style = "
W")

#2.4
spatial_lag_model_alternate <- lagsarlm(alt_topo~(lat+lon):Type, data = overa
ll_data, listw = listw_alternate)

summary(spatial_lag_model_alternate)

##
## Call:lagsarlm(formula = alt_topo ~ (lat + lon):Type, data = overall_data,
##     listw = listw_alternate)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -15.37829  -1.92987   0.73955   2.83223   5.89945
##
## Type: lag
## Coefficients: (asymptotic standard errors)
##                 Estimate Std. Error z value  Pr(>|z|)
## (Intercept)    -137006.1    53141.6 -2.5781 0.0099337
## lat:TypeCistern   1963.0     1011.7  1.9402 0.0523577
## lat:TypeTemple    5559.8     1630.4  3.4100 0.0006497
## lon:TypeCistern   2123.6     1158.5  1.8331 0.0667924
## lon:TypeTemple   -1120.2     1682.6 -0.6658 0.5055503
##
## Rho: 0.72594, LR test value: 27.1, p-value: 1.9321e-07
## Asymptotic standard error: 0.071693
##     z-value: 10.126, p-value: < 2.22e-16
## Wald statistic: 102.53, p-value: < 2.22e-16
##
## Log likelihood: -116.3295 for lag model
## ML residual variance (sigma squared): 17.302, (sigma: 4.1596)
## Number of observations: 39
## Number of parameters estimated: 7
## AIC: 246.66, (AIC for lm: 271.76)
## LM test for residual autocorrelation
## test value: 11.015, p-value: 0.00090387
```

```
#2.5
residuals_alternate <- residuals(spatial_lag_model_alternate)
moran_test <- moran.test(residuals_alternate, listw_alternate)

print(moran_test)
```

```
##
##  Moran I test under randomisation
##
## data:  residuals_alternate
## weights: listw_alternate
##
## Moran I statistic standard deviate = 2.1355, p-value = 0.01636
## alternative hypothesis: greater
## sample estimates:
## Moran I statistic       Expectation          Variance
##        0.26540029       -0.02631579        0.01865980
```

Result model: Latitude has a strong positive relationship with the predicted altitude for both cisterns and temples, with temples having a much stronger dependence on latitude. Longitude has a moderate positive effect for cisterns but is insignificant for temples. The spatial dependence (captured by Rho = 0.726) is highly significant, meaning the altitude at one location is strongly influenced by its neighboring locations. The AIC value shows that the spatial lag model fits the data much better than a regular linear model. Some residual autocorrelation remains, indicating that there may be more complex spatial relationships at play that the current model doesn't fully capture. This model is useful for understanding the spatial relationships affecting the altitude of temples and cisterns based on their locations

Result test: Significant Spatial Autocorrelation: The Moran's I statistic is positive (0.2654), and the p-value is significant (0.016). This means that there is still spatial autocorrelation in the residuals, even after fitting the spatial lag model. In other words, the spatial lag model did not fully account for the spatial patterns in your data, and there may still be some spatial clustering in the errors.

4.Generate a Geographic Grid 4.1 Loading Required Libraries spdep: Provides functions for spatial data analysis, including spatial weights and spatial regression models. spData: A package for spatial data, which might contain useful geographic data sets. sf: Simple Features package, which provides tools for working with spatial vector data (points, lines, polygons) in R. This is important for handling geographic and spatial operations.

4.2 Creating the Grid lat_seq and lon_seq: These sequences create 35 equally spaced points between the minimum and maximum values of latitude (lat) and longitude (lon) from the overall_data dataset. expand.grid(): This function creates a grid of points by taking all combinations of the latitude (lat_seq) and longitude (lon_seq). The resulting grid dataframe will have 35 x 35 = 1,225 points, each defined by a latitude and longitude combination.

4.3 Calculating Cluster Centroids aggregate(): This function computes the mean latitude (lat) and mean longitude (lon) for each cluster in the overall_data. The result is a set of centroids for each cluster, which represent the average geographic location of each cluster. The ~ cluster formula indicates that we are grouping the data by cluster, and FUN = mean specifies that we want the mean value for lat and lon.

4.4 Assigning Clusters to Grid Points This function calculates the distance between each grid point and the centroids of the clusters, then assigns the grid point to the closest cluster.

==>sqrt((lat - cluster_centroids_lat)^2 + (lon - cluster_centroids_lon)^2): This calculates the Euclidean distance between a grid point (defined by lat, lon) and the centroids of the clusters. ==>which.min(distances): This finds the index of the minimum distance, i.e., the closest cluster centroid. ==>closest_cluster: The cluster number corresponding to the centroid with the minimum distance is returned.

4.5 Applying the Function to the Grid mapply(): This function applies assign_closest_cluster to each row in the grid dataset. It passes the lat and lon values of each grid point to the function, calculates the closest cluster, and assigns the result back to the cluster column of the grid dataset. The result is that each grid point is assigned to its nearest cluster.

4.6 Assigning Type Based on Cluster Number ifelse(): This conditional function assigns a "Temple" to grid points in clusters 1 to 3 (i.e., grid$cluster <= 3), and assigns "Cistern" to grid points in clusters 4 and higher. The result is stored in a new column Type.

4.7 Ensuring Type is a Factor factor(): This ensures that the Type column in the grid dataset is treated as a factor (categorical variable) and that it has the same levels (i.e., "Temple" and "Cistern") as in the original overall_data. This step is useful for consistency and when performing analyses that require categorical variables to be factors

```r
#4.1
library(spdep)
library(spData)
library(sf)

#4.2
lat_seq <- seq(min(overall_data$lat), max(overall_data$lat), length.out = 35)
lon_seq <- seq(min(overall_data$lon), max(overall_data$lon), length.out = 35)
grid <- expand.grid(lat = lat_seq, lon = lon_seq)

#4.3
cluster_centroids <- aggregate(cbind(lat, lon) ~ cluster, data = overall_data
, FUN = mean)

#4.4
assign_closest_cluster <- function(lat, lon, cluster_centroids) {
  distances <- sqrt((lat - cluster_centroids$lat)^2 + (lon - cluster_centroid
s$lon)^2)
  closest_cluster <- cluster_centroids$cluster[which.min(distances)]
  return(closest_cluster)
}

#4.5
grid$cluster <- mapply(assign_closest_cluster, grid$lat, grid$lon, MoreArgs =
list(cluster_centroids = cluster_centroids))

#4.6
grid$Type <- ifelse(grid$cluster >= 1 & grid$cluster <= 3, "Temple",
                    ifelse(grid$cluster >= 4 & grid$cluster <= 6, "Cistern",
NA))

#4.7
grid$Type <- factor(grid$Type)


head(grid)

##        lat      lon cluster    Type
## 1 31.77600 35.23443       5 Cistern
## 2 31.77609 35.23443       5 Cistern
## 3 31.77618 35.23443       5 Cistern
## 4 31.77627 35.23443       5 Cistern
## 5 31.77636 35.23443       5 Cistern
## 6 31.77645 35.23443       5 Cistern
```

5.1 Extracting Coordinates for the Grid This line extracts the latitude (lat) and longitude (lon) columns from the grid data frame and converts them into a matrix format. This matrix of coordinates will be used for calculating spatial relationships (i.e., the distance between grid points). It ensures that the spatial weights matrix will be based on the lat/lon positions of the grid.

5.2 Creating a Spatial Weights Matrix This line calculates the k-nearest neighbors for each point in the grid using the coordinates matrix. Specifically, the function knearneigh() finds the 2 nearest neighbors (k = 2) for each grid point. The nearest neighbors are determined based on geographic distance (latitude/longitude).

k = 2: You are looking for the 2 closest neighbors for each point. knearneigh(): This function generates an object that contains information on which points are the nearest neighbors to each other.

This second line converts the nearest-neighbors structure (knn_grid) into a spatial weights matrix. Here's a breakdown:

knn2nb(knn_grid): Converts the nearest neighbors object into a neighborhood object. It defines which points are neighbors of each other. nb2listw(): Converts the neighborhood object into a spatial weights list (listw), which defines how neighboring points are spatially related. style = "W": This indicates that a row-standardized weights matrix is used, where the weights of each point's neighbors sum to 1. The spatial weights matrix (listw_grid) will be used for spatial prediction, where each point's prediction is influenced by its neighboring points.

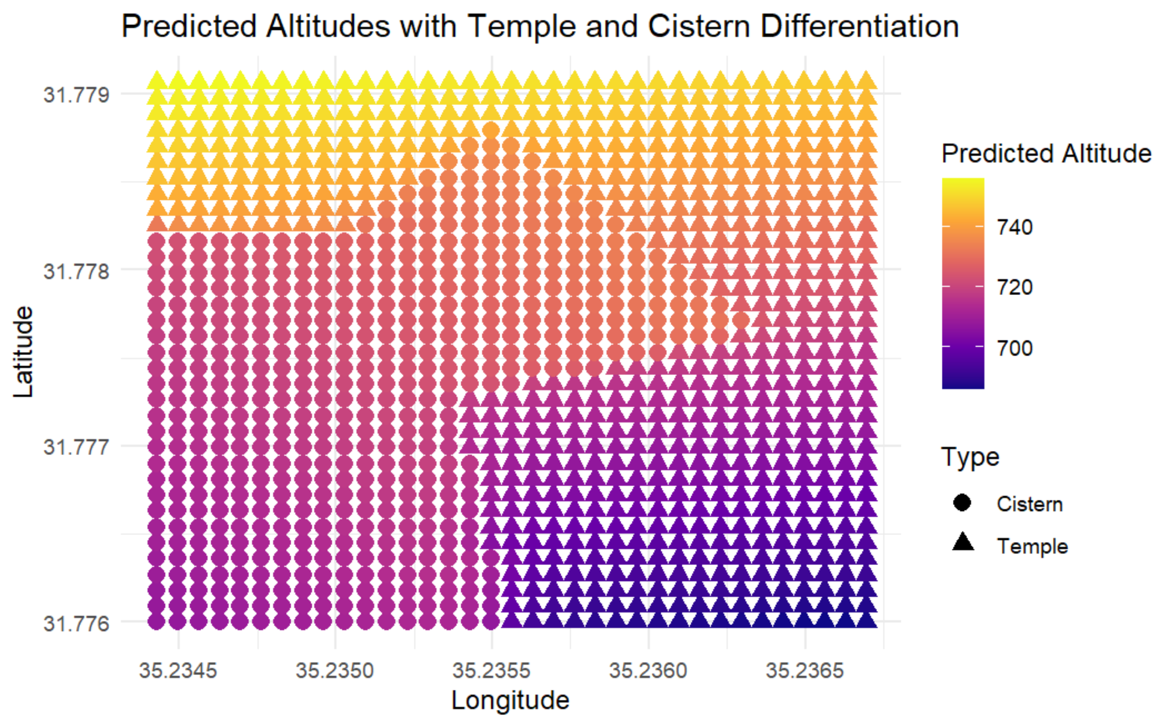
5.3 Making Predictions Using the Spatial Lag Model This line uses the fitted spatial lag model (spatial_lag_model_alternate) to predict altitudes (alt_topo) for each point in the grid based on the grid's spatial relationships.

spatial_lag_model_alternate: This is the pre-fitted spatial lag model that you have trained on the overall_data. It captures the relationships between altitude (alt_topo), latitude (lat), longitude (lon), and spatial effects. newdata = grid: The grid data (i.e., lat/lon points) is passed as new data for prediction. This is the data for which you want to predict altitudes. listw = listw_grid: The spatial weights matrix (listw_grid) is provided to account for spatial dependencies in the prediction. It ensures that each point's prediction considers its neighbors' values based on the spatial relationships defined earlier. The predicted altitudes are stored in the column predicted_alt1 of the grid data frame.

This code takes the predicted altitude data (predicted_alt) from the grid, and creates a spatial plot where:

Each point on the plot corresponds to a geographical location (latitude and longitude). The color of the point represents the predicted altitude at that location, using the Viridis color palette with the "plasma" option (ranging from purple for lower altitudes to yellow for

higher altitudes)



In this plot, we see a grid of points representing predicted altitudes based on the spatial model applied to the grid's latitude and longitude. The colors indicate the predicted altitudes, with the color scale ranging from purple (low altitudes, around 700) to yellow (higher altitudes, up to 745).

The final plot combines two key pieces of information that can help archaeologists identify the locations of temples and cisterns based on spatial modeling:

Predicted Altitudes: The color scale represents the predicted altitudes based on the spatial lag model. Yellow to orange colors represent higher altitudes (around 740 meters), while purple to blue colors represent lower altitudes (around 700 meters). This altitude range is important as it provides insights into the relative height of the landscape.

Temples vs. Cisterns: Circles represent cisterns, and triangles represent temples. The spatial distribution of these shapes across the grid shows how the model has assigned likely locations for temples and cisterns based on their spatial patterns and surrounding variables (latitude, longitude, and possibly other factors such as proximity to water sources). Key Insights for Archaeological Interpretation:

High-Altitude Temple Locations: In the northern section of the plot, there is a concentration of temples (triangles) located at higher altitudes (yellow-orange regions). This suggests that temples are more likely to be situated on elevated terrain. This pattern may reflect historical or cultural practices where temples were constructed on elevated ground for visibility, religious significance, or protection.

Low-Altitude Cistern Locations: In the southern and central regions of the plot, cisterns (circles) are located predominantly in lower-altitude regions (blue-purple areas). Cisterns are commonly built in lower areas to collect and store water, which flows down naturally due to gravity. The model's prediction aligns with this reasoning, as these circles are mostly in areas of lower elevation.

Transition Areas: There is a transition zone in the middle section where you see a mix of temples and cisterns, with altitudes ranging between 710 and 730 meters (red-orange). This transitional zone suggests there might be areas where both types of structures are located relatively close to one another, likely reflecting a balance between religious practices (temples) and functional water storage (cisterns).

Archaeological Recommendations:

Focus on Elevated Regions: The northern and northwestern areas, particularly those with triangles in the yellow-orange regions, are likely candidates for temple locations. Archaeologists should prioritize these high-altitude areas when searching for temples. In the plot, the maximum probability for temples corresponds to the triangles located in the northwestern region (upper-left) where the predicted altitudes are highest (yellow areas). These higher altitudes in the plot suggest that temples are more likely to be found in this region.

Thus, based on the model and the plot:

The upper-left area (northwest) where triangles are located in the yellow and orange zones represents the highest probability for temples. This region has the highest predicted altitudes, which align with the likelihood of temple locations according to the spatial model.

Investigate Lower Regions for Cisterns: The southern and central areas, where circles are predominantly located, should be investigated for cisterns. These lower areas are more likely to contain water storage facilities.

Mixed-Use Areas: Areas where both triangles and circles are found, especially around mid-range altitudes, may reflect historical multi-use zones where both religious and functional structures were located. These could be areas of interest for both temple and cistern research.