What is a k-Means analysis?

A k-Means analysis is one of many *clustering* techniques for identifying structural features of a set of datapoints. The k-Means algorithm groups data into a pre-specified number of clusters, *k*, where the assignment of points to clusters minimizes the total sum-of-squares distance to the cluster's mean. We can then use the mean value of all the points in a given cluster as a prototypical point characterizing the cluster.

For example, we could have a dataset with the horsepower and fuel efficiency of various car models, and we want to use that data to classify the cars into natural groupings–sports cars, sedans, etc. Essentially, we want to find structure in a scatter plot of horsepower against fuel efficiency.

k-Means is easy to implement. In R, you can use the function `kmeans()` to quickly deploy an efficient k-Means algorithm. On datasets of reasonable size (thousands of rows), the `kmeans` function runs in fractions of a second.

k-Means is easy to interpret (in 2 dimensions). If you have two features of your k-Means analysis (*e.g.*, you are grouping by length and width), the result of the k-Means algorithm can be plotted on an xy-coordinate system to show the extent of each cluster. It's easy to visually inspect the assignment to see if the k-Means analysis returned a meaningful insight. In more dimensions (*e.g.*, length, width, and height) you will need to either create a 3D plot, summarize your features in a table, or find another alternative to describing your analysis. This loses the intuitive power that a 2D k-Means analysis has in convincing you or your audience that your analysis should be trusted. It's not to say that your analysis is wrong; it simply takes more mental focus to understand what your analysis says.

The k-Means analysis, however, is not always the best choice. k-Means does well on data that naturally falls into spherical clusters. If your data has a different shape (linear, spiral, etc.), k-Means will force clustering into circles, which can result in outputs that defy human expectations. The algorithm is not wrong; we have fed the algorithm data it was never intended to understand.

Every data analyst should be comfortable using and explaining the k-Means algorithm. It is easy to implement and interpret, and very few real-world datasets violate our spherical-clustering assumption.

How does the k-Means algorithm work?

1. Pick a number of clusters, k
2. Create k random points and call each of these the center of a cluster
3. For each point in your dataset, find the closest cluster center and assign the point to that cluster
4. Once each point has been assigned to a cluster, calculate the new center of that cluster
5. Repeat steps 3 and 4 until you reach a stage when no points need to be reassigned
6. Stop. You have found your k clusters and their centers!

If you want to learn more about k-Means, I would recommend this post on Medium, though be aware that the example code is all written in Python. If you are brave and want to go very deep in k-Means theory, take a look at the Wikipedia page. Or, if you would like to see one application of k-Means in R, see this blog's post about using k-Means to help assist in image classification with Keras. For a detailed illustration of how to implement k-Means in R, along with answers to some common questions, keep reading below.

How do we implement k-Means in R?

Let's begin by loading packages and functions, which we'll use later.

```
library(dplyr)
library(tidyr)
library(broom)
library(purrr)

library(ggplot2)
```

```
library(plotly)

# Define two functions for transforming a distribution of values
#  into the standard normal distribution (bell curve with mean = 0
#  and standard deviation (sd) = 1). More on this later.
normalize_values <- function(x, mean, sd) {
  (x-mean)/sd
}

unnormalize_values <- function(x, mean, sd) {
  (x*sd)+mean
}

set.seed(2021) # So you can reproduce this example
```

The data we will use for this example is from one of R's pre-loaded datasets, `quakes`. It is a data.frame with 1000 rows and five columns describing earthquakes near Fiji since 1964. The columns are latitude (degrees), longitude (degrees), depth (km), magnitude (Richter scale), and the number of stations reporting the quake. The only preprocessing we will do now is to remove `stations` and convert this to a tibble.

```
quakes_raw <- quakes %>%
  dplyr::select(-stations) %>%
  dplyr::as_tibble()

summary(quakes_raw)
##       lat              long           depth            mag
##  Min.   :-38.59   Min.   :165.7   Min.   : 40.0   Min.   :4.00
##  1st Qu.:-23.47   1st Qu.:179.6   1st Qu.: 99.0   1st Qu.:4.30
##  Median :-20.30   Median :181.4   Median :247.0   Median :4.60
##  Mean   :-20.64   Mean   :179.5   Mean   :311.4   Mean   :4.62
##  3rd Qu.:-17.64   3rd Qu.:183.2   3rd Qu.:543.0   3rd Qu.:4.90
##  Max.   :-10.72   Max.   :188.1   Max.   :680.0   Max.   :6.40
```

Now for the fun. For our first example, let's run a k-Means analysis on two variables: depth and magnitude. We reduce our raw data to *only* these two variables and pass it to the base R function, `kmeans()`. Before we hit "Run" on this function, though, let's talk through two principles of k-Means clustering.

Principle 1: Number of iterations

The k-Means algorithm, as mentioned above, iterates through a process of assigning points to a cluster based on the closest cluster center and recalulating cluster centers, not stopping until no more points are assigned to a new cluster during the assignement step. In some cases, the number of iterations can be very large, and the algorithm can consequentially become slow. For speed and convenience, we can cap the number of iterations at 10 (the default value), but for precision, more iterations are better than fewer.

Principle 2: Local vs. global minimum

The k-Means algorithm results in an assignment of points to clusters that minimizes the within-cluster sum of squares: in each iterative step, if we were to add up the total squared distances of points to the mean, we would find that the sum was less than the step before. In laymans terms, our algorithm stopped when it couldn't find a way to assign points into tighter groupings.

But, just because our algorithm couldn't find a better grouping doesn't mean that we found the best grouping. Based on our random set of starting points, we found the best solution, but a different set of starting points could have found a better solution.

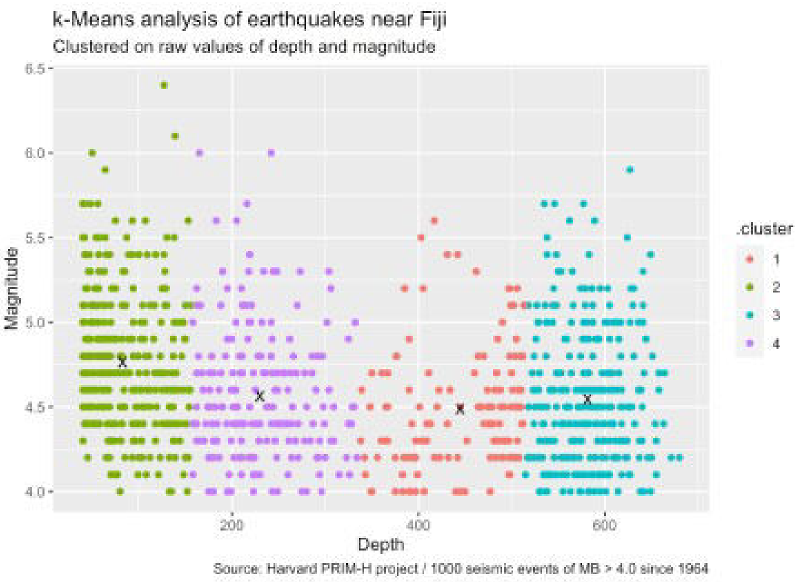
This is a problem common across machine learning algorithms. Finding the globally best solution is substantially harder than finding the best given particular starting point. The `kmeans` function can help us ensure our final product is at least a pretty good local minimum by running multiple times and showing only the best answer. By default, the number of tries `kmeans` takes is 1, but we can easily adjust this to, say, `nstart = 5`.

Now let's hit "Run" on our k-Means analysis and save the output as `kclust`. This is an object of class "kmeans," which is not the easiest to use for subsequent analysis. We'll use `augment`, `tidy`, and `glance` to extract useful summary information as tables. We'll also include a quick plot to see how our clustering did.

```
# 4-cluster k-Means analysis on depth and magnitude
kclust <- quakes_raw %>%
  dplyr::select(depth, mag) %>%
  kmeans(centers = 4, iter.max = 10, nstart = 5)

str(kclust)
## List of 9
##  $ cluster     : int [1:1000] 3 3 2 3 3 4 2 4 4 3 ...
##  $ centers     : num [1:4, 1:2] 444.84 82.52 581.64 229.99 4.49 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:4] "1" "2" "3" "4"
##   .. ..$ : chr [1:2] "depth" "mag"
##  $ totss       : num 46409257
##  $ withinss    : num [1:4] 379494 414571 412840 427964
##  $ tot.withinss: num 1634868
##  $ betweenss   : num 44774389
##  $ size        : int [1:4] 129 361 304 206
##  $ iter        : int 2
##  $ ifault      : int 0
##  - attr(*, "class")= chr "kmeans"
kclust
## K-means clustering with 4 clusters of sizes 129, 361, 304, 206
##
## Cluster means:
##       depth      mag
## 1 444.83721 4.488372
## 2  82.52355 4.763989
## 3 581.64145 4.544408
## 4 229.98544 4.563592
##
## Clustering vector:
##    [1] 3 3 2 3 3 4 2 4 4 3 3 4 3 3 2 4 2 3 3 3 3 4 1 2 1 3 2 1 3 4 3 2 4 1 2
3 4
##   [38] 3 3 4 4 2 1 2 4 2 4 2 1 2 3 4 2 3 1 3 1 3 3 3 3 4 3 2 3 4 3 3 3 2 2 4
2 1
##   [75] 3 3 4 4 4 4 2 1 3 4 1 4 2 3 2 2 2 4 1 4 4 1 1 4 2 1 2 3 3 2 3 4 2 2 2
2 4
##  [112] 1 1 3 3 3 2 2 4 4 2 2 3 3 4 2 3 3 2 4 4 3 2 1 3 2 2 4 2 4 3 3 4 4 3 4
2 2
##  [149] 1 3 2 2 4 2 2 3 2 1 2 2 3 1 4 2 2 2 4 4 3 4 3 1 3 2 1 4 3 4 3 1 3 2 4
1 3
##  [186] 2 4 3 3 4 4 4 3 3 1 3 4 4 3 3 1 1 1 4 3 4 3 1 1 3 1 1 4 2 3 3 4 3 4 1
2 2
##  [223] 2 1 1 2 4 2 4 2 3 1 1 2 3 3 2 3 2 1 4 2 2 2 4 3 3 2 3 2 4 2 3 2 4 3 4
4 3
##  [260] 3 4 3 4 3 4 2 2 2 1 3 2 3 1 1 3 3 3 1 3 3 1 3 2 1 4 3 3 2 3 3 3 2 3 1
4 2
##  [297] 3 3 4 4 3 4 2 3 3 1 3 3 1 2 3 3 1 1 4 1 1 4 1 4 2 4 3 2 2 2 1 2 2 2 2
3 2
##  [334] 4 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 3 2 1 2 2 3 2 3 1 4 2 1 2
```

```
1 4
##  [371] 3 4 3 3 4 2 3 2 2 2 4 4 2 4 3 4 3 1 4 2 2 2 3 1 3 3 3 2 3 3 2 2 2 3 2
3 1
##  [408] 2 1 2 2 2 2 1 2 4 2 4 4 2 1 2 2 3 2 4 3 1 2 3 1 3 3 3 4 2 4 1 1 4 4 2
2 3
##  [445] 2 2 2 1 3 1 4 3 2 4 4 2 3 2 3 3 2 3 3 3 4 3 2 4 3 4 4 2 2 2 2 4 2 2 1
2 3
##  [482] 4 4 2 3 2 4 3 3 3 2 4 3 1 4 2 2 2 1 4 2 2 4 2 3 3 2 2 2 1 2 2 2 3 3 2
2 2
##  [519] 2 1 2 3 2 2 2 3 2 4 1 2 2 2 4 2 2 2 3 2 2 3 2 2 2 2 2 2 2 3 2 3 1 2 2
2 2
##  [556] 2 2 2 3 2 3 2 2 4 1 4 4 3 4 2 4 2 1 2 3 2 3 3 1 2 2 3 2 3 3 2 2 1 3 3
3 1
##  [593] 4 2 2 2 2 1 4 2 2 2 3 3 3 4 2 2 1 4 2 2 4 3 2 1 2 2 3 2 2 4 3 3 2 3 2
1 2
##  [630] 1 3 2 2 3 4 1 4 4 2 3 3 4 2 2 3 4 2 2 2 3 3 3 2 3 3 2 3 2 3 3 4 1 3 3
3 3
##  [667] 3 4 1 4 3 4 3 3 2 2 2 1 3 3 4 4 1 3 2 4 3 1 3 3 3 1 3 3 1 3 3 4 4 3 4
3 3
##  [704] 1 4 2 3 2 4 1 2 4 2 4 3 3 3 2 4 1 2 3 3 3 4 4 1 1 3 1 3 1 4 1 4 2 2 3
3 1
##  [741] 3 2 2 4 4 2 4 2 3 1 3 2 3 2 1 4 2 2 2 2 4 3 4 2 2 2 4 4 4 2 2 1 4 3 3
3 4
##  [778] 3 4 3 2 2 2 2 2 2 2 3 2 1 3 3 3 4 3 4 4 1 2 3 2 1 2 3 3 4 3 3 1 1 3 2
2 2
##  [815] 3 3 4 2 3 3 1 2 3 1 2 1 4 2 2 4 4 2 4 1 4 2 4 1 2 4 3 2 2 2 1 3 1 1 3
1 2
##  [852] 2 2 2 3 4 3 4 4 4 3 3 3 3 2 4 1 4 2 4 3 2 2 3 3 3 1 3 3 3 4 3 2 2 4 3
4 2
##  [889] 2 2 4 3 2 4 3 3 2 3 4 3 4 2 2 3 3 3 1 2 4 2 3 2 4 3 2 2 4 2 2 3 4 2 3
3 4
##  [926] 3 2 2 4 2 4 3 3 2 2 2 3 2 3 3 3 4 3 2 2 2 1 2 2 2 3 2 4 3 3 1 2 2 3 4
2 2
##  [963] 2 4 2 3 1 3 2 2 4 2 2 4 2 4 4 2 3 2 3 2 2 2 2 4 2 2 3 2 3 1 4 3 2 1 4
4 2
## [1000] 4
##
## Within cluster sum of squares by cluster:
## [1] 379494.0 414570.5 412840.2 427963.8
##  (between_SS / total_SS =  96.5 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"
"tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
# Add the cluster number onto to our original data
point_assignments <- broom::augment(kclust, quakes_raw)

# Summarize each cluster
cluster_info <- broom::tidy(kclust)

# Summary stats about our model's fit
model_stats <- broom::glance(kclust)

head(point_assignments)
## # A tibble: 6 x 5
```

```
##      lat  long depth   mag .cluster
##
## 1 -20.4  182.   562   4.8 3
## 2 -20.6  181.   650   4.2 3
## 3 -26    184.    42   5.4 2
## 4 -18.0  182.   626   4.1 3
## 5 -20.4  182.   649   4   3
## 6 -19.7  184.   195   4   4
cluster_info
## # A tibble: 4 x 5
##   depth   mag  size withinss cluster
##
## 1 445.   4.49   129  379494. 1
## 2  82.5  4.76   361  414571. 2
## 3 582.   4.54   304  412840. 3
## 4 230.   4.56   206  427964. 4
model_stats
## # A tibble: 1 x 4
##       totss tot.withinss betweenss  iter
##
## 1 46409257.     1634868. 44774389.     2
# Visually inspect our clusters
ggplot2::ggplot() +
  ggplot2::geom_point(
    data = point_assignments, aes(x = depth, y = mag, color = .cluster)
  ) +
  ggplot2::geom_point(
    data = cluster_info, aes(x = depth, y = mag), size = 4, shape = "x"
  ) +
  ggplot2::labs(
    title = "k-Means analysis of earthquakes near Fiji",
    subtitle = "Clustered on raw values of depth and magnitude",
    caption = "Source: Harvard PRIM-H project / 1000 seismic events of MB > 4.0
since 1964",
    x = "Depth",
    y = "Magnitude"
  )
```



Reflect for a moment on what we just did. First, note that running kmeans is incredibly easy (one function!) and on 1000 data points was very fast. Second, note that our graph looks pretty strange. The algorithm has created clusters that seem only to care about depth. Does this mean magnitude is an irrelevant feature in our data? By no means! Time for Principle 3.

Principle 3: Feature scaling

k-Means calculates distance to the cluster center using Euclidian distance: the length of a line segment connecting the two points. In two dimensions, this is the Pythagorean Theorem. Aha, you say! I see the problem: we are comparing magnitudes (4.0-6.4) to depth (40-680). Depth has significantly more variation (standard deviation 0.4 for magnitude vs. 215 for depth) and therefore gets overweighted when calculating distance to the mean.

We need to employ feature scaling. As a general rule, if we are comparing unlike units (meters and kilograms) or independent measurements (height in meters and circumference in meters), we should normalize values, but if units are related (petal length and petal width), we should leave them as is.

Unfortunately, many cases require judgment both on whether to scale and how to scale. This is where your expert opinion as a data analyst becomes important. For the purposes of this blog post, we will normalize all of our features, including latitude and longitude, by transforming them to standard normal distributions. The geologists might object to this methodology for normalizing (magnitude is a log scale!!), but please forgive some imprecision for the sake of illustration.

```
# Create a tibble to store the information we need to normalize
#  Tibble with row 1 = mean and row 2 = standard deviation
transformations <- dplyr::tibble(
  lat   = c(mean(quakes_raw$lat),   sd(quakes_raw$lat)),
  long  = c(mean(quakes_raw$long),  sd(quakes_raw$long)),
  depth = c(mean(quakes_raw$depth), sd(quakes_raw$depth)),
  mag   = c(mean(quakes_raw$mag),   sd(quakes_raw$mag))
)

# Use the convenient function we wrote earlier
quakes_normalized <- quakes_raw %>%
  dplyr::mutate(
    lat = normalize_values(
      lat, transformations$lat[1], transformations$lat[2]
    ),
    long = normalize_values(
      long, transformations$long[1], transformations$long[2]
    ),
    depth = normalize_values(
      depth, transformations$depth[1], transformations$depth[2]
    ),
    mag = normalize_values(
      mag, transformations$mag[1], transformations$mag[2]
    )
  )

summary(quakes_normalized)
##       lat                long              depth              mag
##  Min.   :-3.56890   Min.   :-2.27235   Min.   :-1.2591   Min.   :-1.54032
##  1st Qu.:-0.56221   1st Qu.: 0.02603   1st Qu.:-0.9853   1st Qu.:-0.79548
##  Median : 0.06816   Median : 0.32095   Median :-0.2987   Median :-0.05065
##  Mean   : 0.00000   Mean   : 0.00000   Mean   : 0.0000   Mean   : 0.00000
##  3rd Qu.: 0.59761   3rd Qu.: 0.61586   3rd Qu.: 1.0747   3rd Qu.: 0.69419
##  Max.   : 1.97319   Max.   : 1.42812   Max.   : 1.7103   Max.   : 4.41837
```

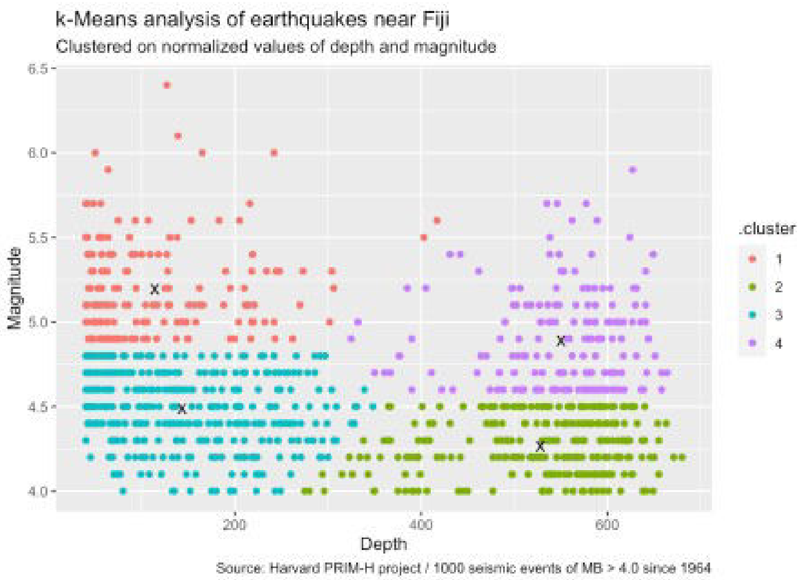
With our fully-preprocessed data, let's re-run our k-Means analysis.

```
kclust <- quakes_normalized %>%
```

```
  dplyr::select(depth, mag) %>%
  kmeans(centers = 4, iter.max = 10, nstart = 5)

str(kclust)
## List of 9
##  $ cluster     : int [1:1000] 4 2 1 2 2 3 3 3 3 2 ...
##  $ centers     : num [1:4, 1:2] -0.915 1.006 -0.779 1.108 1.43 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:4] "1" "2" "3" "4"
##   .. ..$ : chr [1:2] "depth" "mag"
##  $ totss       : num 1998
##  $ withinss    : num [1:4] 103.7 87.8 170.3 107
##  $ tot.withinss: num 469
##  $ betweenss   : num 1529
##  $ size        : int [1:4] 168 262 393 177
##  $ iter        : int 3
##  $ ifault      : int 0
##  - attr(*, "class")= chr "kmeans"
kclust
## K-means clustering with 4 clusters of sizes 168, 262, 393, 177
##
## Cluster means:
##        depth        mag
## 1 -0.9151439  1.4301570
## 2  1.0063484 -0.8817196
## 3 -0.7787787 -0.3241978
## 4  1.1081408  0.6675362
##
## Clustering vector:
##    [1] 4 2 1 2 2 3 3 3 3 2 2 3 2 2 1 3 1 2 2 2 2 3 2 3 4 2 3 4 2 3 4 3 3 2 3
2 3
##   [38] 4 2 3 3 3 2 3 3 1 3 3 2 1 2 3 3 2 2 2 2 2 2 2 4 3 4 3 4 3 2 4 2 1 3 3
3 4
##   [75] 2 2 3 3 2 1 1 2 4 3 2 3 3 2 3 3 1 3 4 3 3 2 2 1 1 4 3 4 4 3 4 3 3 3 1
1 3
##  [112] 2 2 4 2 2 1 3 3 3 3 3 4 2 3 1 2 4 3 3 3 2 3 4 2 3 1 3 1 3 4 2 1 3 2 3
3 1
##  [149] 4 2 1 1 3 3 3 4 3 2 1 3 2 4 3 3 3 1 1 1 4 3 2 4 2 3 2 1 4 3 2 3 2 3 3
2 2
##  [186] 1 3 4 4 3 1 3 4 2 4 2 3 3 2 4 2 2 4 3 2 3 4 2 4 2 4 2 3 1 2 2 3 4 3 2
3 3
##  [223] 1 2 2 1 3 3 1 1 4 2 2 1 2 2 3 4 3 4 3 3 1 3 1 4 2 3 4 3 3 3 4 1 3 2 3
3 2
##  [260] 4 1 4 3 2 3 1 3 3 4 2 3 4 2 2 4 4 4 2 2 4 2 2 3 2 3 4 2 3 2 4 4 3 4 4
1 1
##  [297] 4 2 3 3 2 3 3 4 2 4 2 4 4 3 4 4 4 2 1 2 2 1 2 4 3 1 2 3 1 3 2 3 3 1 1
2 3
##  [334] 1 4 3 3 1 3 3 3 3 3 3 3 3 3 3 1 3 3 1 2 1 3 4 1 1 3 3 2 3 4 2 3 3 4 3
2 3
##  [371] 4 1 4 4 3 1 2 1 3 1 1 3 1 1 4 1 2 2 3 1 3 3 2 2 4 2 4 1 4 4 3 3 3 4 3
2 2
##  [408] 3 2 3 3 3 3 2 3 1 3 3 3 3 2 3 3 4 3 3 4 4 3 2 2 4 2 4 3 3 3 2 2 3 3 3
3 4
##  [445] 1 3 3 4 4 2 3 2 3 3 3 3 2 3 4 4 3 4 4 2 1 2 3 3 2 3 3 3 3 1 3 3 1 3 2
3 2
##  [482] 2 3 1 2 1 3 2 4 4 3 3 2 2 3 1 3 3 2 3 1 3 3 3 4 2 3 3 3 4 3 1 3 4 2 3
3 3
```

```
##  [519] 3 4 3 2 3 3 1 2 3 1 2 3 1 3 2 3 3 3 4 3 1 4 1 3 3 3 3 3 1 4 1 2 2 3 3
3 3
##  [556] 3 3 1 4 3 4 3 1 1 2 3 3 4 3 1 1 3 2 1 2 3 4 2 4 1 3 2 1 4 4 3 3 2 4 4
2 2
##  [593] 3 3 3 3 1 2 3 3 1 3 4 4 4 3 3 1 4 3 3 3 3 2 1 4 1 1 2 3 3 3 4 4 1 2 3
2 1
##  [630] 4 2 3 3 2 3 1 2 4 3 4 4 3 1 3 4 3 3 3 1 2 4 4 1 2 4 3 4 3 2 2 3 4 4 4
2 4
##  [667] 4 3 4 3 2 1 4 2 1 3 3 4 4 4 1 3 4 2 3 3 2 2 4 2 2 4 2 2 2 2 4 3 3 4 3
4 4
##  [704] 2 3 3 4 1 3 2 3 1 1 1 4 2 4 3 3 2 3 2 4 4 3 3 2 3 4 2 4 4 2 2 3 3 3 2
4 2
##  [741] 4 1 3 3 1 1 3 3 2 2 2 1 4 1 2 3 1 1 1 3 3 2 3 1 1 3 3 3 3 3 1 2 3 2 2
2 3
##  [778] 2 3 2 3 3 1 3 1 3 1 4 3 4 2 4 2 3 2 3 3 2 3 2 1 2 3 2 2 3 4 4 2 4 4 1
3 3
##  [815] 4 2 3 3 4 2 4 3 2 2 3 2 3 3 3 3 2 3 3 2 2 3 3 4 1 1 4 3 1 1 2 2 4 2 4
4 3
##  [852] 3 1 3 2 3 4 2 3 3 2 4 2 2 3 3 2 3 1 1 2 3 1 2 2 2 2 2 4 2 3 2 1 3 1 2
3 1
##  [889] 1 1 3 2 1 3 2 2 1 2 1 2 3 1 1 2 2 2 2 1 3 1 4 3 3 2 1 1 3 3 3 4 1 1 2
2 1
##  [926] 2 1 1 3 3 3 2 4 3 1 1 2 1 2 4 2 3 4 1 3 3 2 1 3 3 2 1 3 2 2 2 3 1 2 3
3 3
##  [963] 3 3 1 2 4 2 3 1 3 1 3 1 1 3 3 3 2 1 4 3 1 3 3 3 1 3 2 3 2 2 1 2 3 2 3
3 3
## [1000] 1
## 
## Within cluster sum of squares by cluster:
## [1] 103.74605  87.77469 170.27538 106.98076
##  (between_SS / total_SS =  76.5 %)
## 
## Available components:
## 
## [1] "cluster"      "centers"      "totss"        "withinss"     
"tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
# Unnormalize values for more intuitive summary stats and plots
point_assignments <- broom::augment(kclust, quakes_normalized) %>%
  dplyr::select(-lat, -long) %>%
  dplyr::mutate(
    depth = unnormalize_values(
      depth, transformations$depth[1], transformations$depth[2]
    ),
    mag = unnormalize_values(
      mag, transformations$mag[1], transformations$mag[2]
    )
  )

cluster_info <- broom::tidy(kclust) %>%
  dplyr::mutate(
    depth = unnormalize_values(
      depth, transformations$depth[1], transformations$depth[2]
    ),
    mag = unnormalize_values(
      mag, transformations$mag[1], transformations$mag[2]
    )
```

```
  )

model_stats <- broom::glance(kclust)

head(point_assignments)
## # A tibble: 6 x 3
##   depth   mag .cluster
##
## 1   562   4.8 4
## 2   650   4.2 2
## 3    42   5.4 1
## 4   626   4.1 2
## 5   649   4   2
## 6   195   4   3
cluster_info
## # A tibble: 4 x 5
##   depth   mag  size withinss cluster
##
## 1  114.  5.20   168    104.  1
## 2  528.  4.27   262     87.8 2
## 3  144.  4.49   393    170.  3
## 4  550.  4.89   177    107.  4
model_stats
## # A tibble: 1 x 4
##   totss tot.withinss betweenss  iter
##
## 1 1998.         469.     1529.     3
ggplot2::ggplot() +
  ggplot2::geom_point(
    data = point_assignments, aes(x = depth, y = mag, color = .cluster)
  ) +
  ggplot2::geom_point(
    data = cluster_info, aes(x = depth, y = mag), size = 4, shape = "x"
  ) +
  ggplot2::labs(
    title = "k-Means analysis of earthquakes near Fiji",
    subtitle = "Clustered on normalized values of depth and magnitude",
    caption = "Source: Harvard PRIM-H project / 1000 seismic events of MB > 4.0
since 1964",
    x = "Depth",
    y = "Magnitude"
  )
```



Now for a little more fun. k-Means can be extended beyond 2 feature dimensions. Let's re-run our k-Means analysis again, but this time include all four variables: latitude, longitude, depth, and magnitude. I'll forego the 2D `ggplot2` graph and show you instead a 3D `plotly` graph, with magnitude described by each bubble's size.

```
kclust <- kmeans(quakes_normalized, centers = 4, iter.max = 10, nstart = 5)

str(kclust)
## List of 9
##  $ cluster     : int [1:1000] 3 3 2 3 3 1 4 2 2 3 ...
##  $ centers     : num [1:4, 1:4] 0.2945 -1.7356 -0.0138 0.9338 0.8927 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:4] "1" "2" "3" "4"
##   .. ..$ : chr [1:4] "lat" "long" "depth" "mag"
##  $ totss       : num 3996
##  $ withinss    : num [1:4] 344 253 591 358
##  $ tot.withinss: num 1546
##  $ betweenss   : num 2450
##  $ size        : int [1:4] 244 143 418 195
##  $ iter        : int 4
##  $ ifault      : int 0
##  - attr(*, "class")= chr "kmeans"
kclust
## K-means clustering with 4 clusters of sizes 244, 143, 418, 195
##
## Cluster means:
##           lat       long      depth         mag
## 1  0.29451805  0.8926707 -0.7392667 -0.08422757
## 2 -1.73556818  0.3865266 -0.8379212  0.32263747
## 3 -0.01380115  0.2206365  1.0758663 -0.25081626
## 4  0.93380886 -1.8733897 -0.7667092  0.40643880
##
## Clustering vector:
##    [1] 3 3 2 3 3 1 4 2 2 3 3 4 3 3 4 1 4 3 3 3 3 4 3 2 3 3 4 3 3 1 3 4 1 3 1
3 4
##   [38] 3 3 4 2 1 3 1 4 2 2 4 3 1 3 1 4 3 3 3 3 3 3 3 3 1 3 4 3 1 3 3 3 1 1 1
4 3
##   [75] 3 3 1 4 3 2 2 3 3 1 3 1 4 3 1 1 4 4 3 4 1 3 2 1 4 3 1 3 3 2 3 1 2 4 2
2 1
##  [112] 3 3 3 3 3 4 4 4 4 4 1 3 3 2 4 3 3 2 1 1 3 4 3 3 4 1 1 2 1 4 3 4 2 3 1
```

```
1 4
##  [149] 3 3 2 4 1 4 4 3 4 3 4 4 3 3 4 2 2 2 2 1 3 4 3 3 3 1 3 2 3 1 3 1 3 1 1
3 3
##  [186] 1 3 3 3 1 1 4 3 3 2 3 1 1 3 3 3 3 1 2 4 1 3 3 2 3 2 1 2 1 3 3 2 3 1 3
1 4
##  [223] 1 3 3 4 1 1 2 4 3 3 3 1 3 3 2 3 4 3 1 1 4 1 1 3 3 1 3 4 4 4 3 4 2 3 1
4 3
##  [260] 3 1 3 4 3 1 2 4 4 3 3 1 3 3 3 3 3 3 3 3 3 2 3 1 3 1 3 3 1 3 3 3 4 3 3
1 1
##  [297] 3 3 2 4 3 1 1 3 3 3 3 3 3 2 4 4 3 3 1 3 3 4 3 4 4 1 3 2 4 2 3 4 1 4 1
3 2
##  [334] 4 3 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 4 3 2 2 3 4 1 2 4 3 1 3 2 4 2 3 1
3 1
##  [371] 3 1 3 3 1 1 3 1 1 1 4 4 2 4 3 2 3 3 4 2 1 2 3 3 3 3 3 1 3 4 2 4 2 3 1
3 3
##  [408] 4 3 2 1 1 4 3 1 4 4 2 2 1 3 2 1 3 2 2 3 3 4 3 1 3 3 3 1 2 4 3 3 1 4 1
4 3
##  [445] 1 1 1 3 3 3 2 3 4 1 1 4 3 1 3 3 1 3 3 3 2 3 4 1 3 2 1 1 1 4 1 2 2 2 3
1 3
##  [482] 1 1 2 3 2 2 3 3 3 2 4 3 3 1 4 1 2 3 1 1 1 1 4 3 3 4 4 4 3 2 1 1 3 3 1
4 1
##  [519] 1 3 1 3 2 4 2 3 4 4 3 2 4 4 1 4 1 4 3 4 4 3 4 4 4 4 4 4 4 3 1 3 3 4 4
1 1
##  [556] 1 1 2 3 4 3 1 2 1 3 1 2 3 1 2 4 1 3 1 3 1 3 3 3 2 4 3 4 3 3 1 4 3 3 3
3 3
##  [593] 4 4 1 4 4 3 1 1 2 2 3 3 3 2 1 1 3 2 2 4 4 3 1 3 1 4 3 4 2 2 3 3 4 3 2
3 4
##  [630] 3 3 2 2 3 1 3 1 4 1 3 3 4 2 1 3 1 2 1 2 3 3 3 4 3 3 4 3 2 3 3 2 3 3 3
3 3
##  [667] 3 1 3 1 3 4 3 3 4 2 1 3 3 3 4 1 3 3 1 4 3 3 3 3 3 3 3 3 3 3 3 1 4 3 4
3 3
##  [704] 3 1 1 3 1 1 3 1 4 1 4 3 3 3 2 1 3 4 3 3 3 4 4 3 1 3 3 3 2 2 3 4 1 1 3
3 3
##  [741] 3 1 1 2 1 2 2 1 3 3 3 1 3 4 3 4 1 1 4 1 1 3 1 1 4 4 1 1 4 1 1 3 1 3 3
3 1
##  [778] 3 4 3 4 4 2 1 1 2 4 3 1 3 3 3 3 2 3 2 4 3 1 3 2 3 1 3 3 1 3 3 3 3 3 1
4 1
##  [815] 3 3 1 1 3 3 3 1 3 3 4 3 4 1 1 1 2 1 1 3 3 1 4 3 1 4 3 1 2 4 3 3 3 3 3
3 1
##  [852] 4 4 1 3 1 3 1 2 4 3 3 3 3 4 4 3 1 4 4 3 4 4 3 3 3 3 3 3 3 1 3 4 4 2 3
4 4
##  [889] 1 2 4 3 4 1 3 3 1 3 1 3 2 1 2 3 3 3 3 4 4 2 3 4 1 3 4 4 1 1 1 3 4 1 3
3 2
##  [926] 3 4 2 2 4 1 3 3 1 1 1 3 1 3 3 3 2 3 2 1 2 3 2 1 1 3 2 1 3 3 3 2 1 3 1
1 4
##  [963] 4 1 1 3 3 3 4 2 2 2 1 4 1 1 1 2 3 4 3 4 2 2 1 4 4 4 3 1 3 3 1 3 1 3 4
1 1
## [1000] 4
##
## Within cluster sum of squares by cluster:
## [1] 343.7673 252.9391 590.9476 358.4645
##  (between_SS / total_SS =  61.3 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"
"tot.withinss"
```

```
## [6] "betweenss"    "size"         "iter"         "ifault"
point_assignments <- broom::augment(kclust, quakes_normalized) %>%
  dplyr::mutate(
    lat = unnormalize_values(
      lat, transformations$lat[1], transformations$lat[2]
    ),
    long = unnormalize_values(
      long, transformations$long[1], transformations$long[2]
    ),
    depth = unnormalize_values(
      depth, transformations$depth[1], transformations$depth[2]
    ),
    mag = unnormalize_values(
      mag, transformations$mag[1], transformations$mag[2]
    )
  )

cluster_info <- broom::tidy(kclust) %>%
  dplyr::mutate(
    lat = unnormalize_values(
      lat, transformations$lat[1], transformations$lat[2]
    ),
    long = unnormalize_values(
      long, transformations$long[1], transformations$long[2]
    ),
    depth = unnormalize_values(
      depth, transformations$depth[1], transformations$depth[2]
    ),
    mag = unnormalize_values(
      mag, transformations$mag[1], transformations$mag[2]
    )
  )

model_stats <- broom::glance(kclust)

head(point_assignments)
## # A tibble: 6 x 5
##     lat  long depth   mag .cluster
##
## 1 -20.4  182.   562   4.8 3
## 2 -20.6  181.   650   4.2 3
## 3 -26    184.    42   5.4 2
## 4 -18.0  182.   626   4.1 3
## 5 -20.4  182.   649   4   3
## 6 -19.7  184.   195   4   1
cluster_info
## # A tibble: 4 x 7
##     lat  long depth   mag  size withinss cluster
##
## 1 -19.2  185.  152.  4.59   244     344. 1
## 2 -29.4  182.  131.  4.75   143     253. 2
## 3 -20.7  181.  543.  4.52   418     591. 3
## 4 -15.9  168.  146.  4.78   195     358. 4
model_stats
## # A tibble: 1 x 4
##   totss tot.withinss betweenss  iter
##
```

```
## 1  3996        1546.     2450.     4
plotly::plot_ly() %>%
  plotly::add_trace(
    data = point_assignments,
    x = ~long, y = ~lat, z = ~depth*-1, size = ~mag,
    color = ~.cluster,
    type = "scatter3d", mode = "markers",
    marker = list(symbol = "circle", sizemode = "diameter"),
    sizes = c(5, 30)
  ) %>%
  plotly::layout(scene = list(
    xaxis = list(title = "Longitude"),
    yaxis = list(title = "Latitude"),
    zaxis = list(title = "Depth")
  ))
```

If you've made it this far, and if you are thinking about analyses that you would like to run on your own data, you may be asking yourself why we have been running these analyses with four clusters. It feels like a good number, but is it the right number?

Principle 4: Choose the right number of clusters

The k-Means algorithm cannot tell us what the ideal number of clusters is. The "right" number of clusters is subjective and depends on both the structure of your data and what your intended purpose is. Frequently, we want to find the number of clusters that most efficiently clusters points; we learned a lot by increasing from `k-1` to `k` clusters, but increasing to `k+1` clusters only reduces our sum of squares by a little bit more.

Let's begin by using `purrr::map` to run `kmeans` using 1 through 12 clusters.

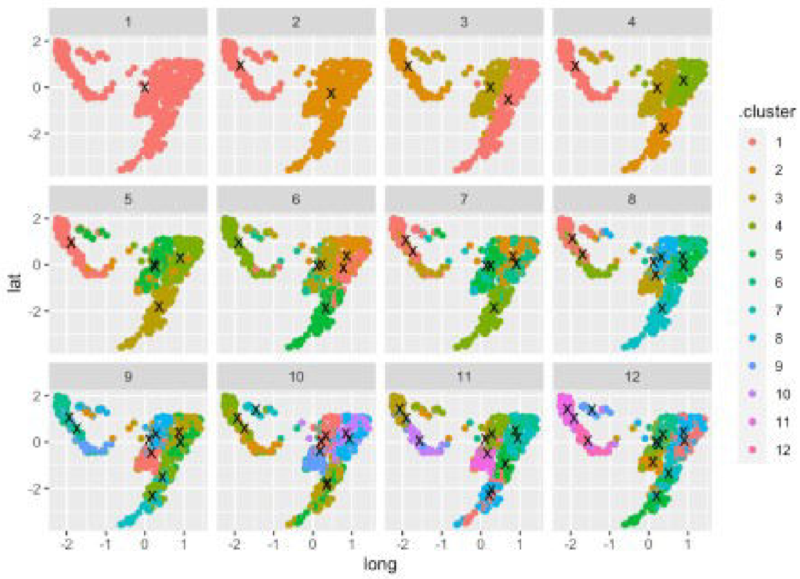
```
# Run analysis with multiple cluster options (can be slow!)
kclusts <-
  dplyr::tibble(n_clusts = 1:12) %>%
  dplyr::mutate(
```

```
    kclust = purrr::map(
      n_clusts,
      ~kmeans(quakes_normalized, centers = .x, iter.max = 10, nstart = 5)
    ),
    augmented = purrr::map(kclust, broom::augment, quakes_normalized),
    tidied = purrr::map(kclust, broom::tidy),
    glanced = purrr::map(kclust, broom::glance)
  ) %>%
  dplyr::select(-kclust)

str(kclusts, max.level = 3)
## tibble [12 × 4] (S3: tbl_df/tbl/data.frame)
##  $ n_clusts : int [1:12] 1 2 3 4 5 6 7 8 9 10 ...
##  $ augmented:List of 12
##   ..$ : tibble [1,000 × 5] (S3: tbl_df/tbl/data.frame)
##   ..$ : tibble [1,000 × 5] (S3: tbl_df/tbl/data.frame)
##   ..$ : tibble [1,000 × 5] (S3: tbl_df/tbl/data.frame)
##   ..$ : tibble [1,000 × 5] (S3: tbl_df/tbl/data.frame)
##   ..$ : tibble [1,000 × 5] (S3: tbl_df/tbl/data.frame)
##   ..$ : tibble [1,000 × 5] (S3: tbl_df/tbl/data.frame)
##   ..$ : tibble [1,000 × 5] (S3: tbl_df/tbl/data.frame)
##   ..$ : tibble [1,000 × 5] (S3: tbl_df/tbl/data.frame)
##   ..$ : tibble [1,000 × 5] (S3: tbl_df/tbl/data.frame)
##   ..$ : tibble [1,000 × 5] (S3: tbl_df/tbl/data.frame)
##   ..$ : tibble [1,000 × 5] (S3: tbl_df/tbl/data.frame)
##   ..$ : tibble [1,000 × 5] (S3: tbl_df/tbl/data.frame)
##  $ tidied   :List of 12
##   ..$ : tibble [1 × 7] (S3: tbl_df/tbl/data.frame)
##   ..$ : tibble [2 × 7] (S3: tbl_df/tbl/data.frame)
##   ..$ : tibble [3 × 7] (S3: tbl_df/tbl/data.frame)
##   ..$ : tibble [4 × 7] (S3: tbl_df/tbl/data.frame)
##   ..$ : tibble [5 × 7] (S3: tbl_df/tbl/data.frame)
##   ..$ : tibble [6 × 7] (S3: tbl_df/tbl/data.frame)
##   ..$ : tibble [7 × 7] (S3: tbl_df/tbl/data.frame)
##   ..$ : tibble [8 × 7] (S3: tbl_df/tbl/data.frame)
##   ..$ : tibble [9 × 7] (S3: tbl_df/tbl/data.frame)
##   ..$ : tibble [10 × 7] (S3: tbl_df/tbl/data.frame)
##   ..$ : tibble [11 × 7] (S3: tbl_df/tbl/data.frame)
##   ..$ : tibble [12 × 7] (S3: tbl_df/tbl/data.frame)
##  $ glanced  :List of 12
##   ..$ : tibble [1 × 4] (S3: tbl_df/tbl/data.frame)
##   ..$ : tibble [1 × 4] (S3: tbl_df/tbl/data.frame)
##   ..$ : tibble [1 × 4] (S3: tbl_df/tbl/data.frame)
##   ..$ : tibble [1 × 4] (S3: tbl_df/tbl/data.frame)
##   ..$ : tibble [1 × 4] (S3: tbl_df/tbl/data.frame)
##   ..$ : tibble [1 × 4] (S3: tbl_df/tbl/data.frame)
##   ..$ : tibble [1 × 4] (S3: tbl_df/tbl/data.frame)
##   ..$ : tibble [1 × 4] (S3: tbl_df/tbl/data.frame)
##   ..$ : tibble [1 × 4] (S3: tbl_df/tbl/data.frame)
##   ..$ : tibble [1 × 4] (S3: tbl_df/tbl/data.frame)
##   ..$ : tibble [1 × 4] (S3: tbl_df/tbl/data.frame)
##   ..$ : tibble [1 × 4] (S3: tbl_df/tbl/data.frame)
kclusts
## # A tibble: 12 x 4
##    n_clusts augmented            tidied            glanced
##
##  1        1
```

```
## 2         2
## 3         3
## 4         4
## 5         5
## 6         6
## 7         7
## 8         8
## 9         9
## 10       10
## 11       11
## 12       12
point_assignments <- kclusts %>%
  dplyr::select(n_clusts, augmented) %>%
  tidyr::unnest(augmented)

cluster_info <- kclusts %>%
  dplyr::select(n_clusts, tidied) %>%
  tidyr::unnest(tidied)

model_stats <- kclusts %>%
  dplyr::select(n_clusts, glanced) %>%
  tidyr::unnest(glanced)

head(point_assignments)
## # A tibble: 6 x 6
##   n_clusts      lat  long  depth    mag .cluster
##
## 1        1  0.0443  0.356  1.16   0.446 1
## 2        1  0.00452 0.258  1.57  -1.04  1
## 3        1 -1.07    0.764 -1.25   1.94  1
## 4        1  0.531   0.362  1.46  -1.29  1
## 5        1  0.0443  0.412  1.57  -1.54  1
## 6        1  0.191   0.799 -0.540 -1.54  1
head(cluster_info)
## # A tibble: 6 x 8
##   n_clusts       lat      long     depth       mag  size withinss cluster
##
## 1        1 -8.27e-17  7.74e-16  1.43e-16 -1.07e-16  1000    3996. 1
## 2        2  9.46e- 1 -1.85e+ 0 -6.68e- 1  3.96e- 1   204     417. 1
## 3        2 -2.42e- 1  4.74e- 1  1.71e- 1 -1.01e- 1   796    2318. 2
## 4        3 -5.16e- 1  7.10e- 1 -8.32e- 1  1.14e- 1   361     922. 1
## 5        3  9.33e- 1 -1.87e+ 0 -7.67e- 1  4.04e- 1   196     361. 2
## 6        3  7.50e- 3  2.47e- 1  1.02e+ 0 -2.71e- 1   443     662. 3
head(model_stats)
## # A tibble: 6 x 5
##   n_clusts totss tot.withinss betweenss  iter
##
## 1        1  3996        3996.  5.46e-12     1
## 2        2  3996        2735.  1.26e+ 3     1
## 3        3  3996        1945.  2.05e+ 3     4
## 4        4  3996        1546.  2.45e+ 3     3
## 5        5  3996        1269.  2.73e+ 3     5
## 6        6  3996        1095.  2.90e+ 3     4
```

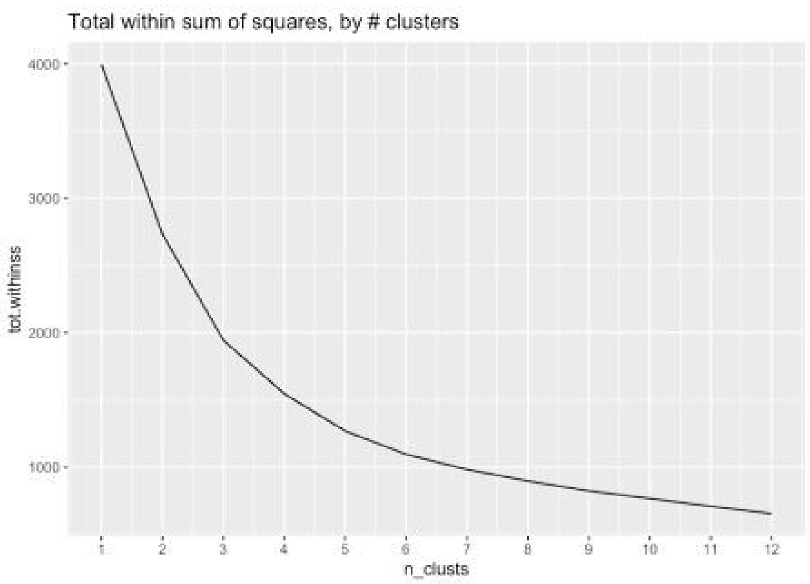
We can plot our 12 k-Means analyses using `ggplot2::facet_wrap`. Since we clustered on four features and our graphs only show two dimensions (latitude and longitude), it's a little hard to get a full picture of how well each k clustered our data. But, it looks like two clusters is reasonable but not very insightful, four seems pretty good, and by eight everything becomes a mess.

```
# Show clusters
ggplot2::ggplot() +
  ggplot2::geom_point(
    data = point_assignments, aes(x = long, y = lat, color = .cluster)
  ) +
  ggplot2::geom_point(
    data = cluster_info, aes(x = long, y = lat), size = 4, shape = "x"
  ) +
  ggplot2::facet_wrap(~n_clusts)
```



We'll build an elbow chart to help us understand how our residual error terms–the sum across all clusters of the within-cluster sum of squares–change with the number of clusters. We plot k against total within sum of squares.

```
# Elbow chart
ggplot2::ggplot(data = model_stats, aes(n_clusts, tot.withinss)) +
  ggplot2::geom_line() +
  ggplot2::scale_x_continuous(limits = c(1, 12), breaks = seq(1, 12, 1)) +
  ggplot2::ggtitle("Total within sum of squares, by # clusters")
```



Ideally, we would see a sharp bend in this curve, where a single cluster number is a turning point from the graph having a steep negative slope to having a shallow negative slope. In this example, however, the rate of change of the slope is gradual, with no clear elbow. This is where the art comes in, and my interpretation is that 3-5 clusters balances a small sum of squares while not losing too much interpretability.

Principle 5: The human side of data science

Principles 3 (Feature scaling) and 4 (Choose the right number of clusters) highlight the importance of human judgment on machine learning algorithms, and data science in general. Algorithms are very good at doing exactly as they are told. But if we feed them poor instructions or poor quality data, they will accurately calculate a useless result. The value of the data analyst, data scientist, or statistician is not that they *can* run complicated analyses, but that they have skill at knowing how *best* to run those analyses.

Concluding thoughts

In this post we have had a brief introduction to the k-Means clustering algorithm, gained an understanding of how it works and where its weaknesses lie, and seen it demonstrated in R using `kmeans()`. The algorithm is easy to run and visualize, and it is a helpful tool for anyone looking to cluster and characterize a large set of datapoints.

```
devtools::session_info()
## ─ Session info ───────────────────────────────────────────────────────────
───
##  setting  value
##  version  R version 4.0.4 (2021-02-15)
##  os       macOS Catalina 10.15.7
##  system   x86_64, darwin17.0
##  ui       X11
##  language (EN)
##  collate  en_US.UTF-8
##  ctype    en_US.UTF-8
##  tz       Europe/Berlin
##  date     2021-03-14
##
## ─ Packages ───────────────────────────────────────────────────────────
───────
##  package      * version date       lib source
##  assertthat     0.2.1   2019-03-21 [2] CRAN (R 4.0.0)
##  backports      1.2.1   2020-12-09 [2] CRAN (R 4.0.2)
##  blogdown       1.2     2021-03-04 [2] CRAN (R 4.0.2)
##  bookdown       0.21    2020-10-13 [2] CRAN (R 4.0.2)
##  broom        * 0.7.5   2021-02-19 [2] CRAN (R 4.0.4)
##  bslib          0.2.4   2021-01-25 [2] CRAN (R 4.0.2)
##  cachem         1.0.4   2021-02-13 [2] CRAN (R 4.0.2)
##  callr          3.5.1   2020-10-13 [2] CRAN (R 4.0.2)
##  cli            2.3.1   2021-02-23 [2] CRAN (R 4.0.4)
##  colorspace     2.0-0   2020-11-11 [2] CRAN (R 4.0.2)
##  crayon         1.4.1   2021-02-08 [2] CRAN (R 4.0.2)
##  crosstalk      1.1.1   2021-01-12 [2] CRAN (R 4.0.2)
##  data.table     1.14.0  2021-02-21 [2] CRAN (R 4.0.4)
##  DBI            1.1.1   2021-01-15 [2] CRAN (R 4.0.2)
##  desc           1.3.0   2021-03-05 [2] CRAN (R 4.0.2)
##  devtools       2.3.2   2020-09-18 [2] CRAN (R 4.0.2)
##  digest         0.6.27  2020-10-24 [2] CRAN (R 4.0.2)
##  dplyr        * 1.0.5   2021-03-05 [2] CRAN (R 4.0.2)
##  ellipsis       0.3.1   2020-05-15 [2] CRAN (R 4.0.0)
##  evaluate       0.14    2019-05-28 [2] CRAN (R 4.0.1)
##  fansi          0.4.2   2021-01-15 [2] CRAN (R 4.0.2)
##  farver         2.1.0   2021-02-28 [2] CRAN (R 4.0.2)
##  fastmap        1.1.0   2021-01-25 [2] CRAN (R 4.0.2)
```

```
##  fs             1.5.0   2020-07-31 [2] CRAN (R 4.0.2)
##  generics       0.1.0   2020-10-31 [2] CRAN (R 4.0.2)
##  ggplot2      * 3.3.3   2020-12-30 [2] CRAN (R 4.0.2)
##  glue           1.4.2   2020-08-27 [2] CRAN (R 4.0.2)
##  gtable         0.3.0   2019-03-25 [2] CRAN (R 4.0.0)
##  highr          0.8     2019-03-20 [2] CRAN (R 4.0.0)
##  htmltools      0.5.1.1 2021-01-22 [2] CRAN (R 4.0.2)
##  htmlwidgets    1.5.3   2020-12-10 [2] CRAN (R 4.0.2)
##  httr           1.4.2   2020-07-20 [2] CRAN (R 4.0.2)
##  jquerylib      0.1.3   2020-12-17 [2] CRAN (R 4.0.2)
##  jsonlite       1.7.2   2020-12-09 [2] CRAN (R 4.0.2)
##  knitr          1.31    2021-01-27 [2] CRAN (R 4.0.2)
##  labeling       0.4.2   2020-10-20 [2] CRAN (R 4.0.2)
##  lazyeval       0.2.2   2019-03-15 [2] CRAN (R 4.0.0)
##  lifecycle      1.0.0   2021-02-15 [2] CRAN (R 4.0.2)
##  magrittr       2.0.1   2020-11-17 [2] CRAN (R 4.0.2)
##  memoise        2.0.0   2021-01-26 [2] CRAN (R 4.0.2)
##  munsell        0.5.0   2018-06-12 [2] CRAN (R 4.0.0)
##  pillar         1.5.1   2021-03-05 [2] CRAN (R 4.0.2)
##  pkgbuild       1.2.0   2020-12-15 [2] CRAN (R 4.0.2)
##  pkgconfig      2.0.3   2019-09-22 [2] CRAN (R 4.0.0)
##  pkgload        1.2.0   2021-02-23 [2] CRAN (R 4.0.4)
##  plotly       * 4.9.3   2021-01-10 [2] CRAN (R 4.0.2)
##  prettyunits    1.1.1   2020-01-24 [2] CRAN (R 4.0.0)
##  processx       3.4.5   2020-11-30 [2] CRAN (R 4.0.2)
##  ps             1.6.0   2021-02-28 [2] CRAN (R 4.0.2)
##  purrr        * 0.3.4   2020-04-17 [2] CRAN (R 4.0.0)
##  R6             2.5.0   2020-10-28 [2] CRAN (R 4.0.2)
##  RColorBrewer   1.1-2   2014-12-07 [2] CRAN (R 4.0.0)
##  remotes        2.2.0   2020-07-21 [2] CRAN (R 4.0.2)
##  rlang          0.4.10  2020-12-30 [2] CRAN (R 4.0.2)
##  rmarkdown      2.7     2021-02-19 [2] CRAN (R 4.0.4)
##  rprojroot      2.0.2   2020-11-15 [2] CRAN (R 4.0.2)
##  rstudioapi     0.13    2020-11-12 [2] CRAN (R 4.0.2)
##  sass           0.3.1   2021-01-24 [2] CRAN (R 4.0.2)
##  scales         1.1.1   2020-05-11 [2] CRAN (R 4.0.0)
##  sessioninfo    1.1.1   2018-11-05 [2] CRAN (R 4.0.0)
##  stringi        1.5.3   2020-09-09 [2] CRAN (R 4.0.2)
##  stringr        1.4.0   2019-02-10 [2] CRAN (R 4.0.0)
##  testthat       3.0.2   2021-02-14 [2] CRAN (R 4.0.2)
##  tibble         3.1.0   2021-02-25 [2] CRAN (R 4.0.2)
##  tidyr        * 1.1.3   2021-03-03 [2] CRAN (R 4.0.2)
##  tidyselect     1.1.0   2020-05-11 [2] CRAN (R 4.0.0)
##  usethis        2.0.1   2021-02-10 [2] CRAN (R 4.0.2)
##  utf8           1.2.1   2021-03-12 [2] CRAN (R 4.0.4)
##  vctrs          0.3.6   2020-12-17 [2] CRAN (R 4.0.2)
##  viridisLite    0.3.0   2018-02-01 [2] CRAN (R 4.0.0)
##  withr          2.4.1   2021-01-26 [2] CRAN (R 4.0.2)
##  xfun           0.22    2021-03-11 [2] CRAN (R 4.0.2)
##  yaml           2.2.1   2020-02-01 [2] CRAN (R 4.0.0)
##
## [1] /Users/shiringlander/Library/R/4.0/library
## [2] /Library/Frameworks/R.framework/Versions/4.0/Resources/library
```