Introduction to Data Science

STAT240/08

Dr. Lloyd T. Elliott

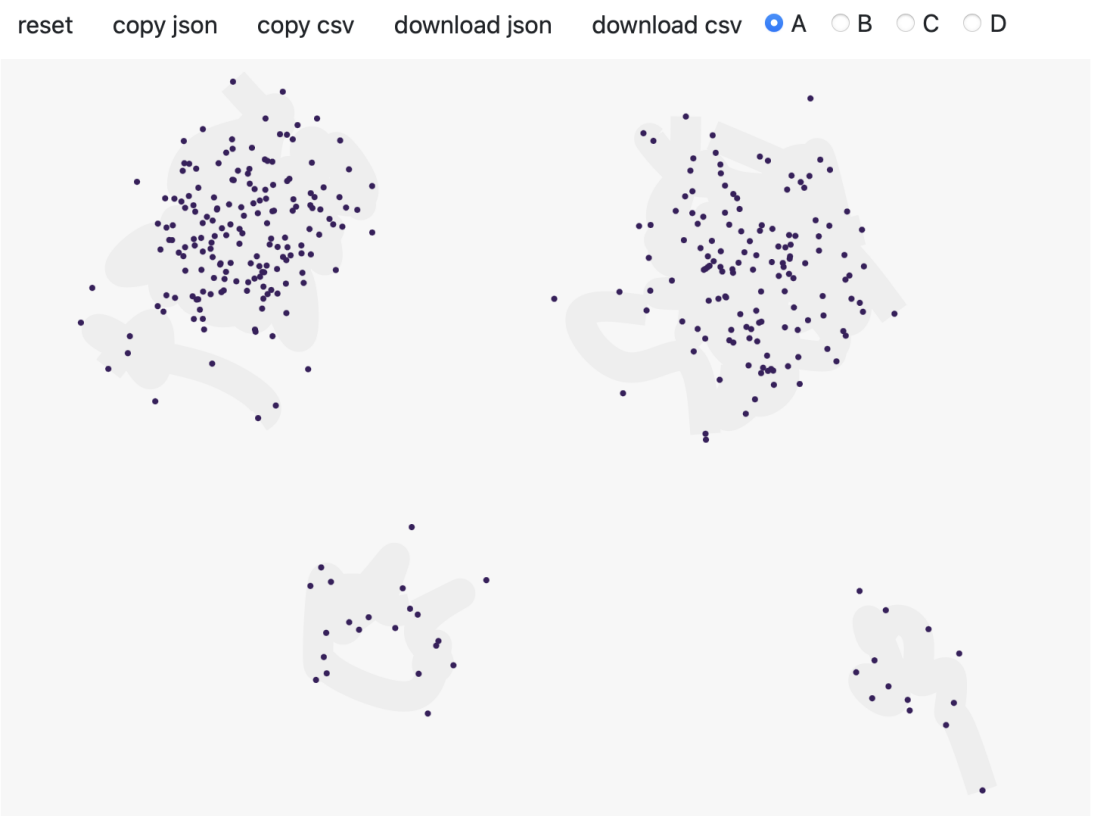
2023/03/05

Clustering

- Group similar data items together
- Exploratory data analysis / pattern recognition
- Example of *unsupervised learning* (a branch of ML)

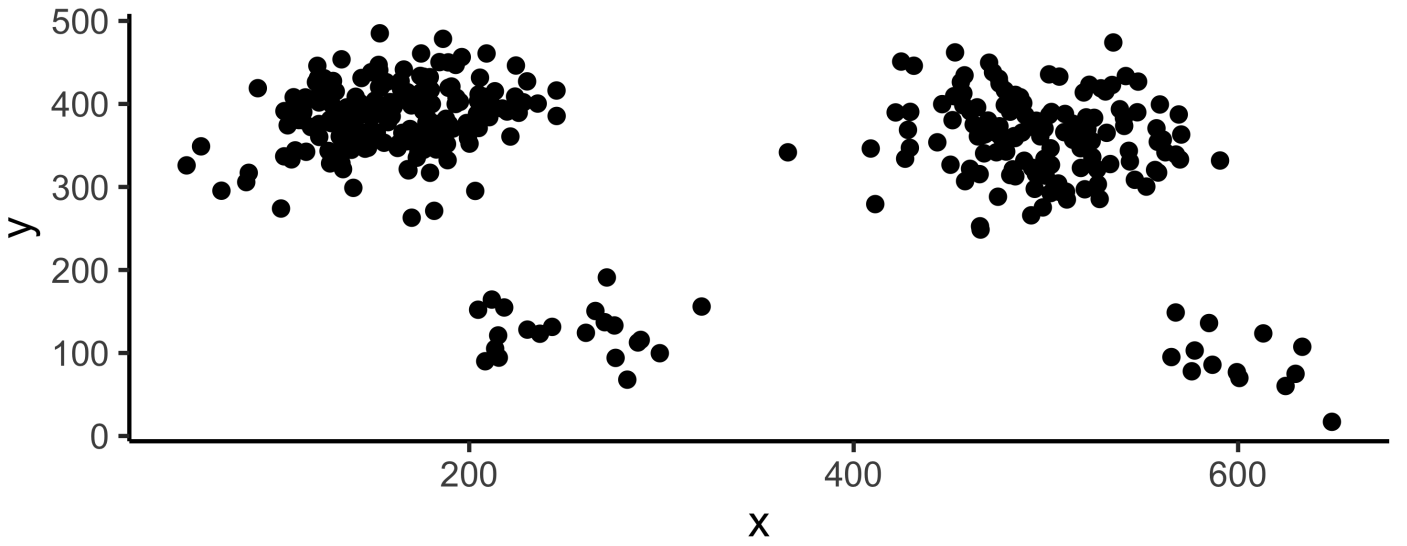
Simulated dataset

- I used Calm Code to simulate a dataset demonstrating clustering



Simulated dataset

```
data = read.csv('data.csv')
library(ggplot2)
ggplot(data, aes(x = x, y = y)) +
  theme_classic() +
  geom_point()
```



k-means clustering

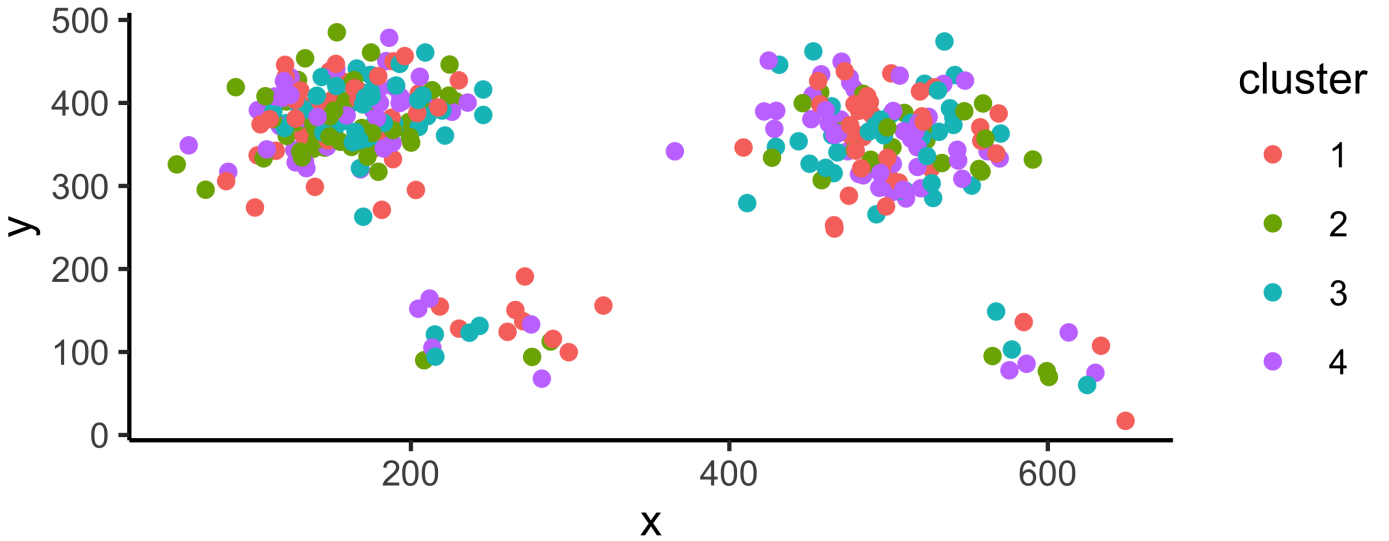
The simplest clustering method is k-means clustering

- Iterative method with random initialization
- The parameter is fixed: the number of clusters *k*

k-means clustering in 2D

- Step 1: Randomly assign each data item to one of the *k* clusters

```
N = dim(data)[1]; K = 4; set.seed(240)
cluster = sample(K, size = N, replace = TRUE)
data$cluster = cluster
p = ggplot(data, aes(x = x, y = y, color = as.factor(cluster)))
+ labs(color = "cluster") + theme_classic() + geom_point()
print(p)
```



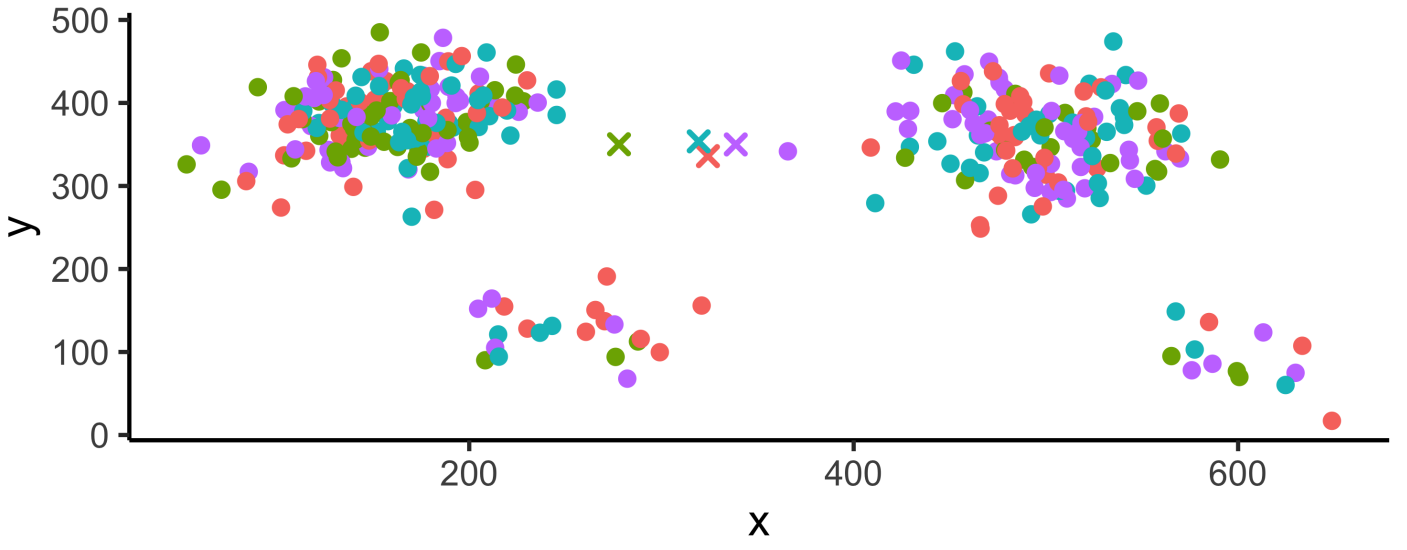
k-means clustering in 2D

- Step 2: Compute the mean *x* and *y* coordinate of each cluster (this is called a centroid)
- If a cluster is empty, assign the corresponding centroid to a randomly selected data item

```
mus = matrix(NA, K, 3)
colnames(mus) = c("x", "y", "cluster")
for (k in 1:K) {
  mus[k, 1] = mean(data$x[data$cluster == k])
  mus[k, 2] = mean(data$y[data$cluster == k])
  mus[k, 3] = k
}
q = p + geom_point(as.data.frame(mus), mapping = aes(x = x, y =
y, color = as.factor(cluster)), shape=4, stroke = 1) +
theme(legend.position = "none")
print(q)
```

k-means clustering in 2D

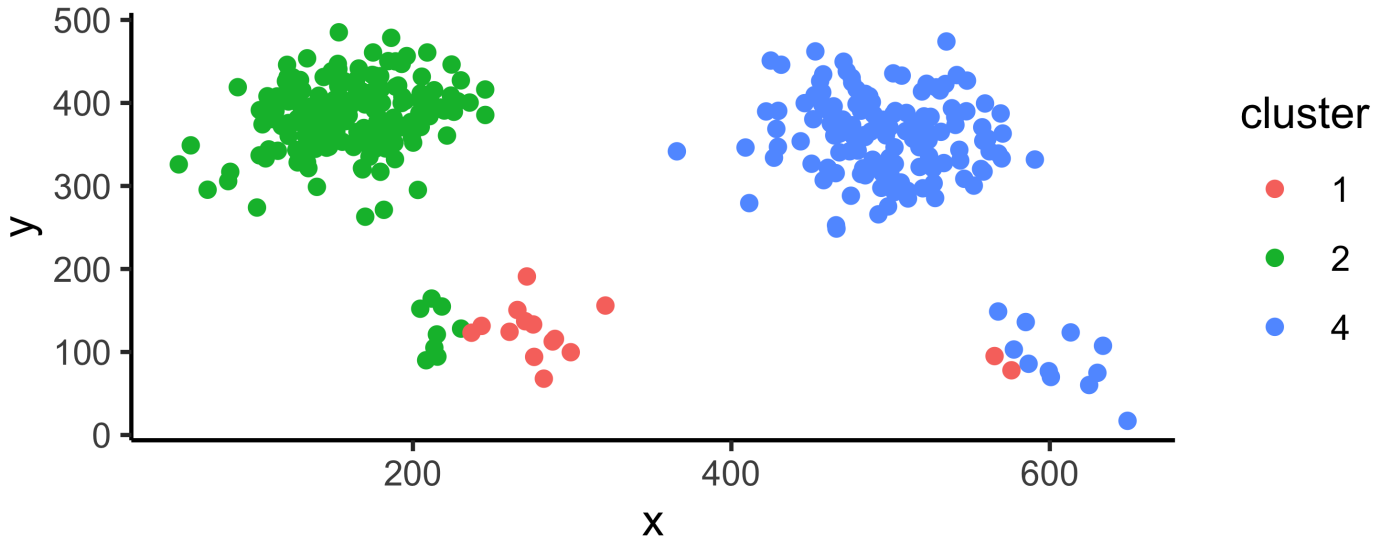
- Step 2:



k-means clustering in 2D

- Step 3: Reassign each data item to the cluster with the closest centroid

```
library(flexclust)
d = dist2(data[, 1:2], mus[, 1:2])
for (n in 1:N) { data$cluster[n] = which.min(d[n, ]) }
p = ggplot(data, aes(x = x, y = y, color = as.factor(cluster)))
+ labs(color = "cluster") + theme_classic() + geom_point()
print(p)
```



k-means clustering in 2D

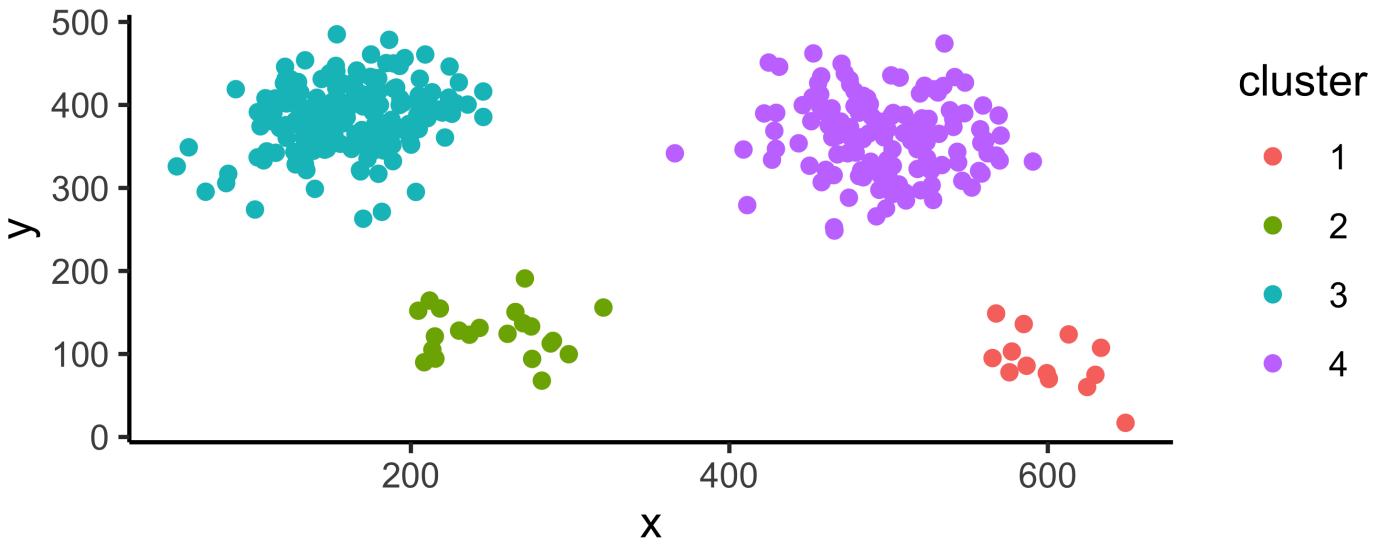
- Iterate by repeating steps 2 and 3 until a stopping condition is met
- Examples of stopping conditions:

1. A maximum number of iterations is reached
2. The cluster assignment doesn't change
3. The centroids move by only a small amount

k-means clustering in 2D

- The result may look something like this:

```
set.seed(240)
result = kmeans(data[, 1:2], 4, iter.max = 100)
data$cluster = result$cluster
p = ggplot(data, aes(x = x, y = y, color = as.factor(cluster)))
+ labs(color = "cluster") + theme_classic() + geom_point()
print(p)
```



k-means theory

- The k-means algorithm is one of the simplest clustering algorithms
- A big drawback is that you have to already know how many clusters you want
- Another drawback is that the "variance" of each cluster is the same: A cluster that is very compact might have a k-means solution that includes aspects of other clusters

k-means theory

- Another drawback is that k-means clustering doesn't work for "convex" clusters

String manipulation

- Natural language processing (NLP)
  - Sentiment analysis
  - Chatbots, translation
  - Electronic healthcare records ...
- Preprocessing data sources
  - Normalization of factors
  - Parsing websites or documents ...
- Data analysis
  - Understanding DNA ...

String manipulation

We'll review string manipulation in base R. First, we will read the text of the Great Gatsby (Fitzgerald 1925) into a single string variable

```
fn = "gatsby.txt"
s = readChar(fn, file.info(fn)$size)
nchar(s) # Print number of characters in text
```

```
[1] 296673
```

String manipulation

- We want a character vector with one word per element

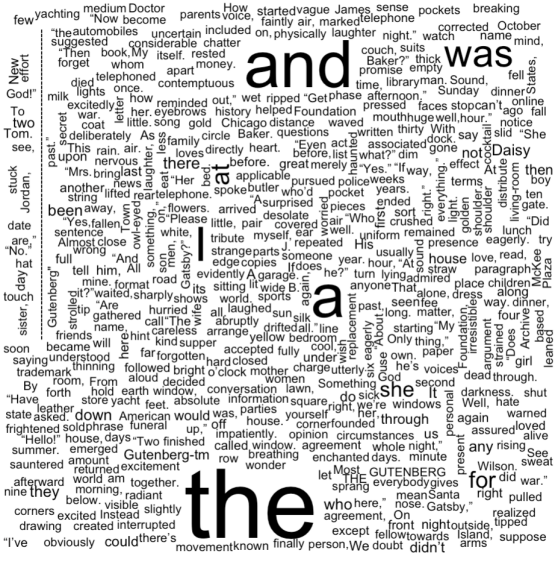
```
x = strsplit(s, '\\s+')
x = unlist(x)
print(x[507:517])
```

```
 [1] "I"       "felt"    "that"    "I"       "wanted"  "the"
"world"
 [8] "to"      "be"      "in"      "uniform"
```

Wordclouds

- We can create a wordcloud of the document (visualization)

```
library(wordcloud)
t = table(x)
wordcloud(names(t), t)
```



Wordclouds

- In NLP, we often ignore "stopwords" such as "a" and "the". Here, we also ignore infrequent words

```
library(stopwords)
x = tolower(x)
x = x[!(x %in% stopwords("en"))]
t = table(x)
t = t[t >= 20]
```

Low level string manipulation

Easiest R package: *stringr*

- Match a string and extract it *str_extract*
- Detect a matching string *str_detect*
- Replace one string with another *str_replace*
- Split a string on a substring *str_split*

Strings: extracting

```
a = c("apples x4", "bag of flour", "bag of sugar", "milk x2")
str_extract(a, "\\d")
```

```
[1] "4" NA  NA  "2"
```

```
str_extract(a, "[a-z]+")
```

```
[1] "apples" "bag"    "bag"    "milk"
```

Strings: detecting

```
a = c("apple", "banana", "pear", "pineapple")
str_detect(a, "a")
```

```
[1] TRUE TRUE TRUE TRUE
```

```
str_detect(a, "^a")
```

```
[1]  TRUE FALSE FALSE FALSE
```

Strings: replacing

```
a = c("one apple", "two pears", "three bananas")
str_replace(a, "[aeiou]", "-")
```

```
[1] "-ne apple"     "tw- pears"     "thr-e bananas"
```

```
str_replace_all(a, "[aeiou]", "-")
```

```
[1] "-n- -ppl-"     "tw- p--rs"     "thr-- b-n-n-s"
```

Strings: splitting

```
a = c("apples and oranges and pears and bananas",
  "pineapples and mangos and guavas")
str_split(a, " and ")
```

```
[[1]]
[1] "apples"  "oranges" "pears"   "bananas"

[[2]]
[1] "pineapples" "mangos"     "guavas"
```

```
str_split(a, "\\s+")
```

```
[[1]]
[1] "apples"  "and"     "oranges" "and"     "pears"   "and"
"bananas"

[[2]]
[1] "pineapples" "and"        "mangos"     "and"        "guavas"
```

Reading

- Munzert Section 8.2