worksheet_10

December 9, 2021

1 Worksheet 10 - Clustering

1.0.1 Lecture and Tutorial Learning Goals:

After completing this week's lecture and tutorial work, you will be able to:

- Describe a case where clustering would be an appropriate tool, and what insight it would bring from the data.
- Explain the k-means clustering algorithm.
- Interpret the output of a k-means cluster analysis.
- Perform k-means clustering in R using k-means
- Visualize the output of k-means clustering in R using a coloured scatter plot
- Identify when it is necessary to scale variables before clustering and do this using R
- Use the elbow method to choose the number of clusters for k-means
- Describe advantages, limitations and assumptions of the k-means clustering algorithm.

[1]:
```
### Run this cell before continuing.
library(tidyverse)
library(forcats)
library(repr)
library(broom)
options(repr.matrix.max.rows = 6)
source('tests_worksheet_10.R')
source("cleanup_worksheet_10.R")
```

```
  Attaching packages                                        tidyverse
1.3.0

 ggplot2 3.3.2      purrr   0.3.4
 tibble  3.0.3      dplyr   1.0.2
 tidyr   1.1.2      stringr 1.4.0
 readr   1.3.1      forcats 0.5.0

Warning message:
"package ‘ggplot2’ was built under R version 4.0.1"
Warning message:
"package ‘tibble’ was built under R version 4.0.2"
Warning message:
"package ‘tidyr’ was built under R version 4.0.2"
```

```
Warning message:
"package ‘dplyr’ was built under R version 4.0.2"
  Conflicts
tidyverse_conflicts()
 dplyr::filter() masks stats::filter()
 dplyr::lag()    masks stats::lag()

Warning message:
"package ‘broom’ was built under R version 4.0.2"

Attaching package: ‘testthat’


The following object is masked from ‘package:dplyr’:

    matches


The following object is masked from ‘package:purrr’:

    is_null


The following object is masked from ‘package:tidyr’:

    matches
```

Question 0.0 Multiple Choice: {points: 1}

In which of the following scenarios would clustering methods likely be appropriate?

A. Identifying sub-groups of houses according to their house type, value, and geographical location

B. Predicting whether a given user will click on an ad on a website

C. Segmenting customers based on their preferences to target advertising

D. Both A. and B.

E. Both A. and C.

Assign your answer to an object called `answer0.0`. Make sure your answer is an uppercase letter and is surrounded by quotation marks (e.g. `"F"`).

[2]:
```
# your code here
answer0.0 <- "E"
```

[3]:
```
test_0.0()
```

```
[1] "Success!"
```

Question 0.1 Multiple Choice: {points: 1}

Which step in the description of the k-means algorithm below is *incorrect*?

0. Choose the number of clusters
1. Randomly assign each of the points to one of the clusters
2. Calculate the position for the cluster centre (centroid) for each of the clusters (this is the middle of the points in the cluster, as measured by straight-line distance)
3. Re-assign each of the points to the cluster who's centroid is furthest from that point
4. Repeat steps 1 - 2 until the cluster centroids don't change very much between iterations

Assign your answer to an object called `answer0.1`. Your answer should be a single numerical character surrounded by quotes.

[4]:
```
# your code here
answer0.1 <- "3"
```

[5]:
```
test_0.1()
```

```
[1] "Success!"
```

1.1 Hoppy Craft Beer

Craft beer is a strong market in Canada and the US, and is expanding to other countries as well. If you wanted to get into the craft beer brewing market, you might want to better understand the product landscape. One popular craft beer product is hopped craft beer. Breweries create/label many different kinds of hopped craft beer, but how many different kinds of hopped craft beer are there really when you look at the chemical properties instead of the human labels?

We will start to look at the question by looking at a craft beer data set from Kaggle. In this data set, we will use the alcoholic content by volume (`abv` column) and the International bittering units (`ibu` column) as variables to try to cluster the beers. The `abv` variable has values 0 (indicating no alcohol) up to 1 (pure alcohol) and the `ibu` variable quantifies the bitterness of the beer (higher values indicate higher bitterness).

Question 1.0 {points: 1}

Read in the `beers.csv` data using `read_csv()` and assign it to an object called `beer`. The data is located within the `worksheet_10/data/` folder.

Assign your dataframe answer to an object called `beer`.

[6]:
```
# your code here
beer <- read_csv("data/beers.csv")
beer
```

```
Warning message:
"Missing column names filled in: 'X1' [1]"
Parsed with column specification:
cols(
  X1 = col_double(),
  abv = col_double(),
  ibu = col_double(),
  id = col_double(),
  name = col_character(),
  style = col_character(),
  brewery_id = col_double(),
  ounces = col_double()
)
```

A spec_tbl_df: 2410 × 8

X1 <dbl>	abv <dbl>	ibu <dbl>	id <dbl>	name <chr>	style <chr>
0	0.050	NA	1436	Pub Beer	American Pale Lager
1	0.066	NA	2265	Devil's Cup	American Pale Ale (APA)
2	0.071	NA	2264	Rise of the Phoenix	American IPA
2407	0.055	NA	620	B3K Black Lager	Schwarzbier
2408	0.055	40	145	Silverback Pale Ale	American Pale Ale (APA)
2409	0.052	NA	84	Rail Yard Ale (2009)	American Amber / Red A

```
[7]: test_1.0()
```

```
[1] "Success!"
```

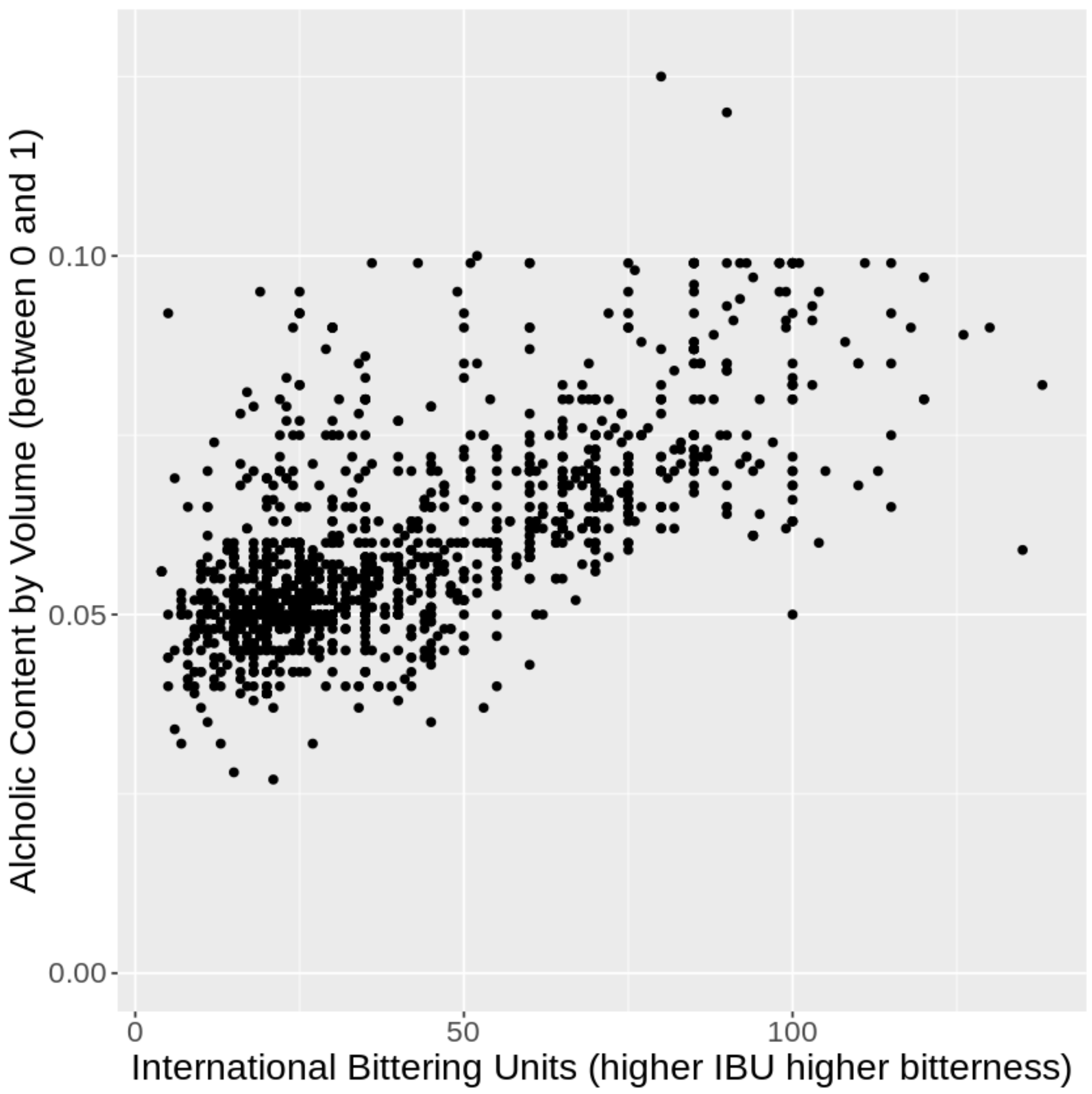
Question 1.1 {points: 1}

Let's start by visualizing the variables we are going to use in our cluster analysis as a scatter plot. Put `ibu` on the horizontal axis, and `abv` on the vertical axis. Name the plot object `beer_eda`.

Remember to follow the best visualization practices, including adding human-readable labels to your plot.

```
[8]: # your code here
beer_eda <- ggplot(beer, aes(x = ibu, y = abv)) +
  geom_point() +
  xlab("International Bittering Units (higher IBU higher bitterness)") +
  ylab("Alcholic Content by Volume (between 0 and 1)")+
  theme(text = element_text(size = 17))
beer_eda
```

```
Warning message:
"Removed 1005 rows containing missing values (geom_point)."
```



```
[9]: test_1.1()
```

```
[1] "Success!"
```

Question 1.2 {points: 1}

We need to clean this data a bit. Specifically, we need to remove the rows where `ibu` is `NA`, and select only the columns we are interested in clustering, which are `ibu` and `abv`.

Assign your answer to an object named `clean_beer`.

```
[10]: # your code here
clean_beer <- filter(beer, ibu !="NA") %>% select(ibu, abv)#drop_na(beer) %>%
           #select(ibu, abv)
clean_beer
```

A tibble: 1405 × 2

ibu <dbl>	abv <dbl>
60	0.061
92	0.099
45	0.079
50	0.060
45	0.067
40	0.055

```
[11]: test_1.2()
```

```
[1] "Success!"
```

Question 1.3.1 Multiple Choice: {points: 1}

Why do we need to scale the variables when using k-means clustering?

A. k-means uses the Euclidean distance to compute how similar data points are to each cluster center

B. k-means is an iterative algorithm

C. Some variables might be more important for prediction than others

D. To make sure their mean is 0

Assign your answer to an object named `answer1.3.1`. Make sure your answer is an uppercase letter and is surrounded by quotation marks (e.g. `"F"`).

```
[12]: # your code here
      answer1.3.1 <- "A"
```

```
[13]: test_1.3.1()
```

```
[1] "Success!"
```

Question 1.3.2 {points: 1}

Let's do that scaling now. Recall that we used a `recipe` for scaling when doing classification and regression. This is because we needed to be able to split train and test data, compute a standardization on *just* training data, and apply the standardization to *both* train and test data.

But in clustering, there is no train/test split. So let's use the much simpler `scale` function in R. `scale` takes in a column of a dataframe and outputs the standardized version of it. We can therefore apply `scale` to all variables in the cleaned data frame using the `map_df` function.

Note: you could still use a recipe to do this, using `prep`/`bake` appropriately. But `scale` is much simpler.

Assign your answer to an object named `scaled_beer`. Use the scaffolding provided.

```
[14]: # ... <- ... %>%
      #    map_df(...)
```

```r
# your code here
#scale
scaled_beer <- clean_beer %>%
                map_df(scale)
scaled_beer
```

A tibble: 1405 × 2

ibu	abv
<dbl[,1]>	<dbl[,1]>
0.66605490	0.08000109
1.89900237	2.87899086
0.08811077	1.40583835
0.28075881	0.006343468
0.08811077	0.521946846
-0.10453727	-0.361944659

[15]:
```r
test_1.3.2()
```

```
[1] "Success!"
```

Question 1.4 {points: 1}

From our exploratory data visualization, 2 seems like a reasonable number of clusters. Use the `kmeans` function with `centers = 2` to perform clustering with this choice of k.

Assign your model to an object named `beer_cluster_k2`. *Note that since k-means uses a random initialization, we need to set the seed again; don't change the value!*

[16]:
```r
# DON'T CHANGE THE SEED VALUE!
set.seed(1234)

# ... <- kmeans(..., centers = 2)
# your code here
beer_cluster_k2 <- kmeans(scaled_beer, centers = 2)
beer_cluster_k2
```

```
K-means clustering with 2 clusters of sizes 917, 488

Cluster means:
         ibu        abv
1 -0.5830271 -0.5506271
2  1.0955653  1.0346824

Clustering vector:
   [1] 2 2 2 1 1 1 1 2 2 2 2 2 1 2 2 2 2 1 2 1 1 1 1 2 2 1 1 2 1 1 2 1 1 2 1 1 2
  [38] 1 2 2 2 1 1 1 1 1 2 1 2 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1 1 1 2
  [75] 1 1 1 1 2 1 1 2 1 1 2 1 2 1 2 1 1 2 1 1 1 1 1 1 2 1 1 1 1 2 1 1 2 1 1 1 2
 [112] 2 1 1 2 2 1 2 2 1 1 1 2 1 1 2 1 1 2 2 1 1 2 1 1 1 2 1 2 1 1 1 1 1 2 2 2 2
 [149] 1 1 2 1 1 1 2 1 1 2 1 1 1 2 1 1 2 2 1 2 1 2 1 2 1 1 2 2 2 1 1 1 1 1 2 1 1
 [186] 1 1 2 2 1 1 2 1 1 1 1 1 1 1 2 1 1 1 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 2
 [223] 1 1 1 1 2 1 2 1 1 1 1 2 2 2 2 1 1 1 1 1 2 1 2 1 2 1 1 1 1 1 2 1 1 2 1 1 1
 [260] 2 2 1 1 1 1 1 1 2 2 2 1 1 1 1 1 2 1 1 1 1 2 1 2 1 2 1 1 1 2 2 1 1 1 1 1 1
 [297] 1 1 2 2 2 1 1 2 1 1 2 1 1 1 2 1 1 2 2 2 2 2 2 2 2 2 2 2 1 1 2 2 1 2 2 1 1 2
 [334] 1 2 1 1 2 1 1 2 2 1 1 1 1 2 1 1 1 1 1 2 2 1 2 2 1 1 2 1 1 1 1 2 1 1 1 1 2
 [371] 1 2 1 1 1 1 2 1 2 2 1 1 1 2 1 2 2 1 1 2 1 2 1 1 1 2 2 2 2 1 1 1 2 2 2 2 1
 [408] 1 1 2 1 2 1 2 1 1 2 2 1 2 1 2 2 1 1 1 1 2 2 1 2 1 1 2 1 1 1 2 1 1 1 2 2 1
 [445] 1 1 1 1 2 1 1 1 1 2 1 1 1 2 2 1 1 2 2 1 2 2 2 2 1 2 2 1 2 2 2 1 1 2 1 1 2
 [482] 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 2 2 1 1 1 2 1 2 1 1
 [519] 1 1 1 1 1 1 1 1 2 1 2 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1
 [556] 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 1 1 2 2 1 2 1 1 1 1 1 1 1 1 1 1 2 2 1
 [593] 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 2 2 2 1 2 1 2 1 1 1 2 1 1 1
 [630] 1 1 2 1 1 2 1 1 1 2 2 1 1 1 1 2 1 1 1 1 1 1 1 1 2 1 1 2 1 2 1 1 1 2 2 1 2
 [667] 1 2 1 2 1 1 1 1 1 1 1 2 1 1 1 1 2 2 2 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 2
 [704] 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1 2 2 2 1 2 1 1 2 1 2 2 1 1 1 1 2 1
 [741] 1 1 2 1 1 1 2 2 1 1 1 1 1 1 2 1 2 2 1 1 2 2 2 1 2 1 1 1 2 1 2 1 1 1 2 1 2
 [778] 2 2 1 2 1 1 1 2 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 1 1 1 1 2 2 1 2 1 1 1
 [815] 1 2 1 1 1 1 1 1 2 1 2 1 1 2 2 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1 2 1 1 2 2 1 1
 [852] 1 2 2 1 2 1 1 1 1 1 2 1 1 2 2 1 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2
 [889] 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 2 2 1 1 2 1 1 1 1
 [926] 1 2 2 2 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 2 1 1 1 1 2 2 1 1
 [963] 1 1 1 1 1 1 1 2 2 2 1 2 1 1 1 1 1 1 2 1 1 1 2 1 1 1 2 1 1 1 2 2 2 2 1 1 1 2
[1000] 1 2 1 1 2 1 1 1 2 2 1 1 1 2 1 1 1 1 2 1 1 1 1 2 2 1 2 1 1 1 1 1 1 2 2 2 2
[1037] 1 2 1 2 2 2 1 1 1 2 1 2 1 1 2 1 2 2 2 1 1 2 2 2 1 1 1 2 2 1 1 1 2 2 2 1 1
[1074] 1 2 1 2 2 1 1 1 1 2 1 2 1 1 1 1 2 1 1 1 1 1 1 1 2 2 2 2 1 1 2 2 1 1 1 1 2
[1111] 2 2 1 2 1 2 1 1 2 1 1 2 2 2 2 1 2 1 2 1 1 1 1 1 1 2 1 2 2 1 1 2 1 1 1 1 1
[1148] 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 2 2 2 1 2 2 2 2
[1185] 2 2 2 1 2 1 1 1 2 2 1 1 1 2 2 2 1 2 1 1 1 2 2 2 1 1 1 1 2 1 1 2 2 1 1 1 1
[1222] 1 2 1 1 1 1 1 1 2 1 1 1 1 2 1 2 1 1 1 1 2 1 1 2 2 1 2 1 1 1 1 2 2 2 1 2 1
[1259] 1 1 1 2 1 1 2 1 1 1 1 1 2 1 1 1 1 1 1 1 1 2 2 1 1 1 1 2 1 1 1 1 1 1 2 1 2 1
[1296] 2 1 2 2 1 2 1 2 1 1 1 1 2 2 1 1 1 2 1 2 1 1 1 1 1 1 2 1 1 1 2 1 1 2 1 1 2 1
[1333] 2 1 1 1 1 2 1 2 2 2 1 1 1 2 2 2 2 2 2 2 2 2 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1
[1370] 1 2 1 1 1 1 2 1 2 2 1 1 1 1 1 1 1 2 2 2 1 2 1 2 2 1 2 1 2 2 1 2 2 1 2 1

Within cluster sum of squares by cluster:
[1] 465.5831 644.5188
 (between_SS / total_SS =  60.5 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
[6] "betweenss"    "size"         "iter"         "ifault"
```

[17]:
```
test_1.4()
```

```
[1] "Success!"
```

Question 1.5 {points: 1}

Use the `augment` function from the `broom` package to get the cluster assignment for each point in the `scaled_beer` data frame.

Assign your answer to an object named `tidy_beer_cluster_k2`*.*

[18]:
```
# ... <- augment(..., ...)
# your code here
tidy_beer_cluster_k2 <- augment(beer_cluster_k2, scaled_beer)
tidy_beer_cluster_k2
```

A tibble: 1405 × 3

ibu <dbl[,1]>	abv <dbl[,1]>	.cluster <fct>
0.66605490	0.08000109	2
1.89900237	2.87899086	2
0.08811077	1.40583835	2
0.28075881	0.006343468	1
0.08811077	0.521946846	2
-0.10453727	-0.361944659	1

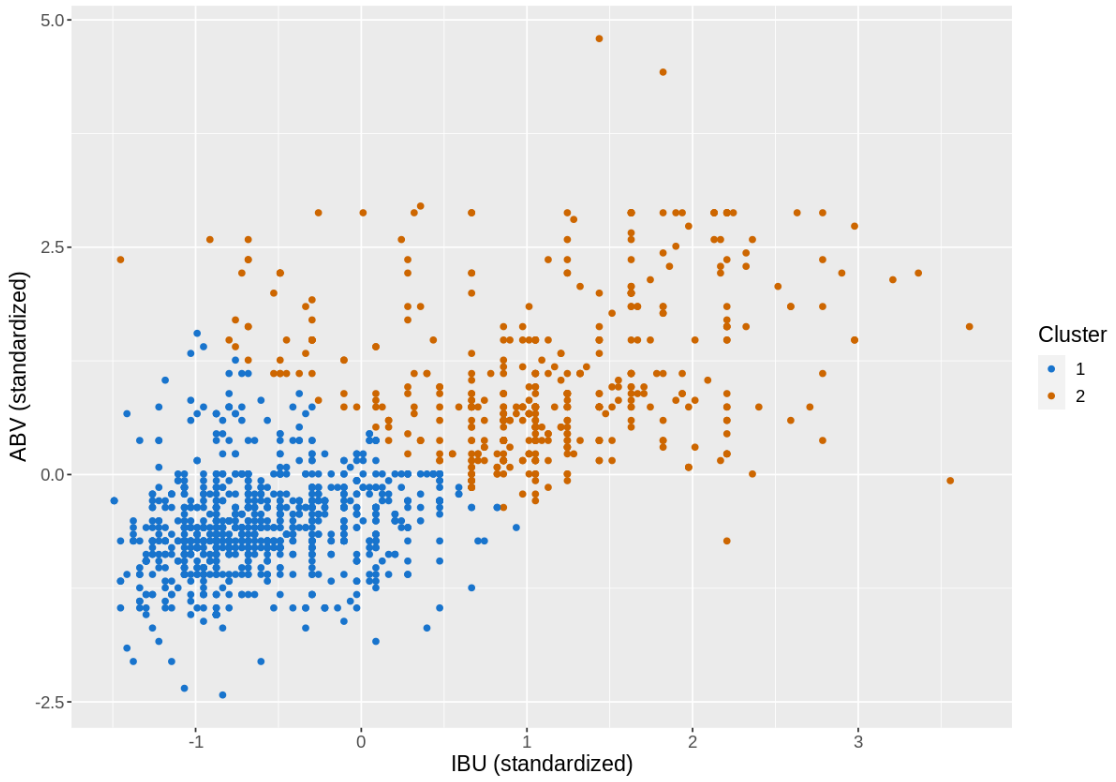
[19]:
```
test_1.5()
```

```
[1] "Success!"
```

Question 1.6 {points: 1}

Create a scatter plot of `abv` on the y-axis versus `ibu` on the x-axis (using the data in `tidy_beer_cluster_k2`) where the points are labelled by their cluster assignment. Name the plot object `tidy_beer_cluster_k2_plot`.

[20]:
```
# your code here
options(repr.plot.width = 10, repr.plot.height = 7)
tidy_beer_cluster_k2_plot <- ggplot(tidy_beer_cluster_k2,
  aes(x = ibu,
      y = abv,
      color = .cluster),
  size = 2) +
  geom_point() +
  labs(x = "IBU (standardized)",
       y = "ABV (standardized)",
       color = "Cluster") +
  scale_color_manual(values = c("dodgerblue3",
                                "darkorange3"))+
   theme(text = element_text(size = 14))
tidy_beer_cluster_k2_plot
```



[21]:
```
test_1.6()
```

```
[1] "Success!"
```

Question 1.7.1 Multiple Choice: {points: 1}

We do not know, however, that two clusters ($k = 2$) is the best choice for this data set. What can we do to choose the best K?

A. Perform *cross-validation* for a variety of possible k's. Choose the one where within-cluster sum of squares distance starts to *decrease less*.

B. Perform *cross-validation* for a variety of possible k's. Choose the one where the within-cluster sum of squares distance starts to *decrease more*.

C. Perform *clustering* for a variety of possible k's. Choose the one where within-cluster sum of squares distance starts to *decrease less*.

D. Perform *clustering* for a variety of possible k's. Choose the one where the within-cluster sum of squares distance starts to *decrease more*.

Assign your answer to an object called `answer1.7.1`. *Make sure your answer is an uppercase letter and is surrounded by quotation marks (e.g.* `"F"`*).*

[22]:
```
# your code here
answer1.7.1 <- "C"
```

```
[23]: test_1.7.1()
```

```
[1] "Success!"
```

Question 1.7.2 {points: 1}

Use the `glance` function from the `broom` library to get the model-level statistics for the clustering we just performed, including total within-cluster sum of squares.

Assign your answer to an object named `beer_cluster_k2_model_stats`.

```
[24]: # your code here
beer_cluster_k2_model_stats <- glance(beer_cluster_k2)
beer_cluster_k2_model_stats
```

A tibble: 1 × 4

totss	tot.withinss	betweenss	iter
<dbl>	<dbl>	<dbl>	<int>
2808	1110.102	1697.898	1

```
[25]: test_1.7.2()
```

```
[1] "Success!"
```

Question 1.8 {points: 1}

Let's now choose the best K for this clustering problem. To do this we need to create a tibble with a column named `k`, where k has values 1 to 10.

Assign your answer to an object named `beer_ks`.

```
[26]: # your code here
beer_ks <- tibble(k = 1:10)
beer_ks
```

A tibble: 10 × 1

k
<int>
1
2
3
8
9
10

```
[27]: test_1.8()
```

```
[1] "Success!"
```

Question 1.9 {points: 1}

Next we use `mutate` to create a new column named `models` in `beer_ks`, where we use `map` to apply the `kmeans` function to our `scaled_beer` data set for each of the k's .

> This is a more complicated use of the `map` function than we have seen previously in the course. This is because we need to iterate over the different values of k, which is the second argument to the `kmeans` function. In the past, we have used `map` only to iterate over values of the first argument of a function. Since that is the default, we could simply write `map(data_frame, function_name)`. This won't work here; we need to provide our data frame as the first argument to the `kmeans` function. You might want to refer back to the section of the textbook that explains before completing this question: K-means in R

This will give us a data frame with two columns, the first being `k`, which holds the values of the k's we mapped (i.e, iterated) over. The second will be `models`, which holds the k-means model fits for each of the k's we mapped over.

> This second column is a new type of column, that we have not yet encountered in this course. It is called a list column. It can contain more complex objects, like models and even data frames (as we will see in a later question). In Jupyter it is easier to preview and understand this more complex data frame using the `print` function as opposed to calling the data frame itself as we usually do. This is a current limitation of Jupyter's rendering of R's output and will hopefully be fixed in the future.

Assign your answer to an object named `beer_clustering`*.*

[28]:
```r
set.seed(1234) # DO NOT REMOVE
# ... <- ... %>%
    # mutate(models = map(..., ~kmeans(scaled_beer, .x)))

# your code here
beer_clustering <- beer_ks %>%
    mutate(models = map(k, ~kmeans(scaled_beer, .x)))
print(beer_clustering)
```

```
# A tibble: 10 x 2
       k models
   <int> <list>
 1     1 <kmeans>
 2     2 <kmeans>
 3     3 <kmeans>
 4     4 <kmeans>
 5     5 <kmeans>
 6     6 <kmeans>
 7     7 <kmeans>
 8     8 <kmeans>
 9     9 <kmeans>
10    10 <kmeans>
```

[29]:
```r
test_1.9()
```

```
[1] "Success!"
```

Question 2.0 {points: 1}

Next we use `mutate` again to create a new column called `model_statistics` where we use `map` to apply the `glance` function to each of our models (in the `models` column) to get the model-level statistics (this is where we can get the value for total within sum of squares that we use to choose K).

> Here, because we are interating over the first argument to the `glance` function (which is the `models` column), we can use the simpler syntax for `map` as we did earlier in the course.

Assign your answer to an object named `beer_model_stats`*.*

[30]:
```
# ... <- ... %>%
    # mutate(... = map(models, ...))

# your code here
beer_model_stats <- beer_clustering %>%
     mutate(model_statistics = map(models, glance))
print(beer_model_stats)
```

```
# A tibble: 10 x 3
       k models   model_statistics
   <int> <list>
<list>
 1     1 <kmeans> <tibble [1 × 4]>
 2     2 <kmeans> <tibble [1 × 4]>
 3     3 <kmeans> <tibble [1 × 4]>
 4     4 <kmeans> <tibble [1 × 4]>
 5     5 <kmeans> <tibble [1 × 4]>
 6     6 <kmeans> <tibble [1 × 4]>
 7     7 <kmeans> <tibble [1 × 4]>
 8     8 <kmeans> <tibble [1 × 4]>
 9     9 <kmeans> <tibble [1 × 4]>
10    10 <kmeans> <tibble [1 × 4]>
```

[31]:
```
test_2.0()
```

```
[1] "Success!"
```

> Here when we create our third column, called `model_statistics`, we can see it is another list column! This time it contains data frames instead of models! Run the cell below to see how you can look at the data frame that is stored as the first element of the `model_statistics` column (model where we used $k = 1$):

[32]:
```
beer_model_stats %>%
    slice(1) %>%
    pull(model_statistics)
```

1. A tibble: 1 × 4

totss	tot.withinss	betweenss	iter
<dbl>	<dbl>	<dbl>	<int>
2808	2808	-2.273737e-12	1

Question 2.1 {points: 1}

Now we use the `unnest` function to expand the data frames in the `model_statistics` column so that we can access the values for total within sum of squares as a column.

Assign your answer to an object named `beer_clustering_unnested`*.*

[33]:
```
# ... <- ... %>% unnest(model_statistics)
# your code here
beer_clustering_unnested <- beer_model_stats %>% unnest(model_statistics)
print(beer_clustering_unnested)
```

```
# A tibble: 10 x 6
       k models   totss tot.withinss betweenss  iter
   <int> <list>   <dbl>
<dbl>     <dbl> <int>
 1     1 <kmeans> 2808.         2808.
-2.27e-12      1
 2     2 <kmeans> 2808.         1110.
1.70e+ 3     1
 3     3 <kmeans> 2808.          803.
2.00e+ 3     3
 4     4 <kmeans> 2808.          624.
2.18e+ 3     3
 5     5 <kmeans> 2808.          567.
2.24e+ 3     3
 6     6 <kmeans> 2808.          417.
2.39e+ 3     6
 7     7 <kmeans> 2808.          361.
2.45e+ 3     6
 8     8 <kmeans> 2808.          318.
2.49e+ 3     5
 9     9 <kmeans> 2808.          294.
2.51e+ 3     4
10    10 <kmeans> 2808.          264.
2.54e+ 3     6
```

[34]:
```
test_2.1()
```

```
[1] "Success!"
```

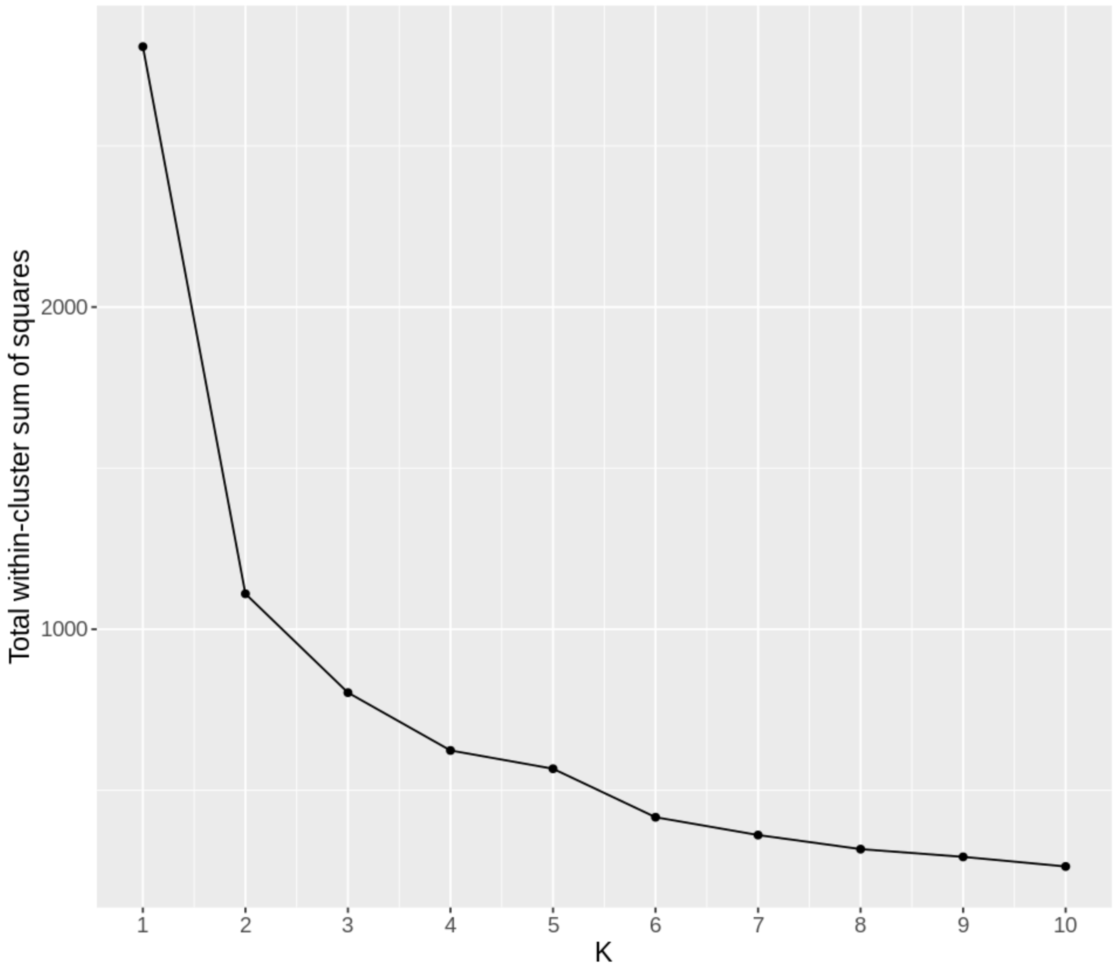
Question 2.2 {points: 1}

We now have the the values for total within-cluster sum of squares for each model in a column (`tot.withinss`). Let's use it to create a line plot with points of total within-cluster sum of squares versus k, so that we can choose the best number of clusters to use.

Assign your plot to an object called `choose_beer_k`*. Total within-cluster sum of squares should be on the y-axis and K should be on the x-axis. Remember to follow the best visualization practices, including adding human-readable labels to your plot.*

```
[35]: options(repr.plot.width = 8, repr.plot.height = 7)

      # your code here
      choose_beer_k <- ggplot(beer_clustering_unnested, aes(x = k, y = tot.withinss))␣
       ↪+
        geom_point() +
        geom_line() +
        xlab("K") +
        ylab("Total within-cluster sum of squares") +
        scale_x_continuous(breaks = 1:10)+
        theme(text = element_text(size = 14))
      choose_beer_k
```



```
[36]: test_2.2()
```

```
[1] "Success!"
```

Question 2.3 {points: 1}

From the plot above, which k should we choose?

Assign your answer to an object called `answer2.3`*. Make sure your answer is a single numerical character surrounded by quotation marks.*

[37]:
```
# your code here
answer2.3 <- "2"
```

[38]:
```
test_2.3()
```

```
[1] "Success!"
```

Question 2.4 {points: 1}

Why did we choose the k we chose above?

A. It had the greatest total within-cluster sum of squares

B. It had the smallest total within-cluster sum of squares

C. Increasing k further than this only decreased the total within-cluster sum of squares a small amount

D. Increasing k further than this only increased the total within-cluster sum of squares a small amount

Assign your answer to an object called `answer2.4`*. Make sure your answer is an uppercase letter and is surrounded by quotation marks (e.g.* `"F"`*).*

[39]:
```
# your code here
answer2.4 <- "C"
```

[40]:
```
test_2.4()
```

```
[1] "Success!"
```

Question 2.5 Multiple Choice: {points: 1}

What can we conclude from our analysis? How many different types of hoppy craft beer are there in this data set using the two variables we have?

A. 1

B. 2

C. 3

D. 4

Assign your answer to an object called `answer2.5`*. Make sure your answer is an uppercase letter and is surrounded by quotation marks (e.g.* `"F"`*).*

[41]:
```
# your code here
answer2.5 <- "B"
```

```
[42]: test_2.5()
```

```
[1] "Success!"
```

Question 2.6 True or false: {points: 1}

Our analysis might change if we added additional variables, true or false?

Assign your answer to an object called `answer2.6`*. Make sure your answer is written in lowercase and is surrounded by quotation marks (e.g.* `"true"` *or* `"false"`*).*

```
[43]: # your code here
      answer2.6 <- "true"
```

```
[44]: test_2.6()
```

```
[1] "Success!"
```

```
[45]: source("cleanup_worksheet_10.R")
```