SciencesPo
DEPARTMENT OF ECONOMICS

ScPoEconometrics: Advanced

Statistical Learning 2

Bluebery Planterose
SciencesPo Paris
2023-04-18

Resampling Methods

- We already encountered the **bootstrap**: we resample repeatedly *with replacement* from our analysis data.

- We'll also learn about **cross validation** today, which is a related idea.

- The bootstratp is useful assess model uncertainty.

- Cross Validation is used assess model accuracy.

Resampling Methods

- We already encountered the **bootstrap**: we resample repeatedly *with replacement* from our analysis data.
- We'll also learn about **cross validation** today, which is a related idea.
- The bootstratp is useful assess model uncertainty.
- Cross Validation is used assess model accuracy.

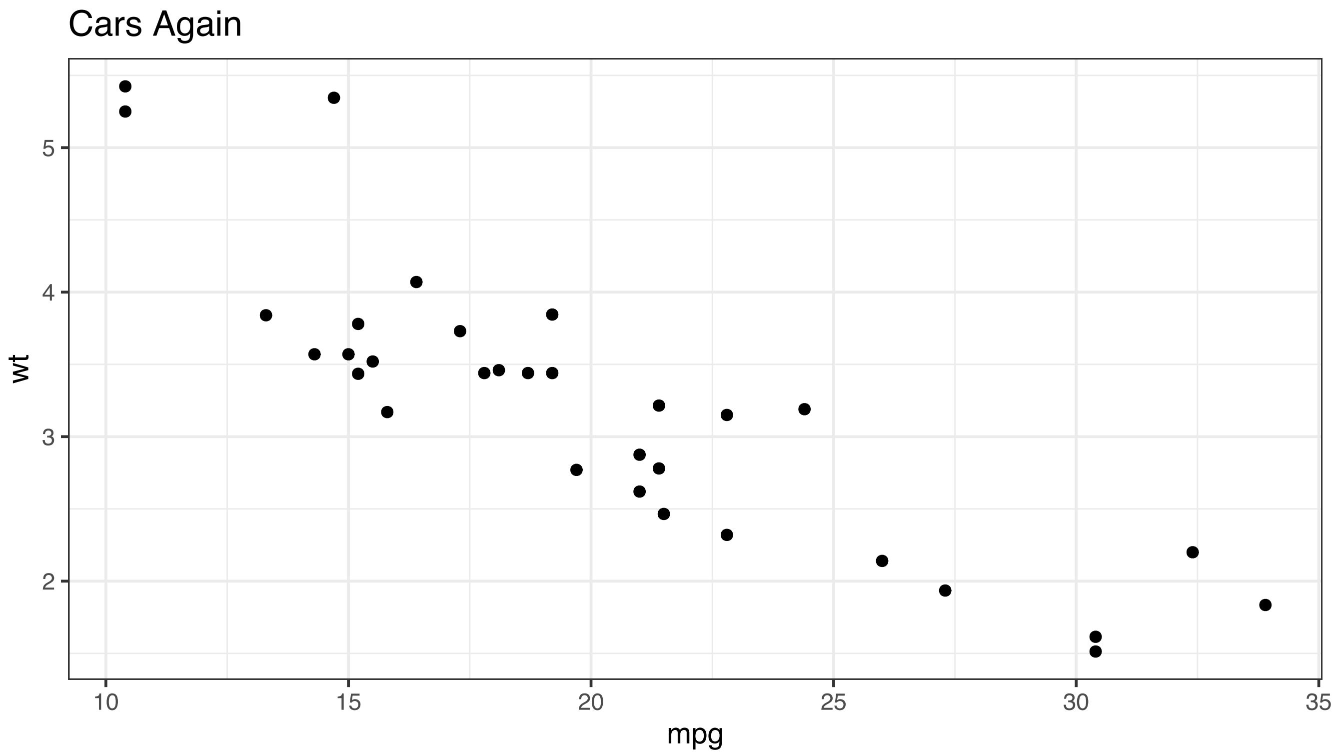
- Remember how **bootstrapping** works: We just pretend that our sample *is* the full population.
- And we repeatedly draw from this randomly, with replacement.
- This will create a sampling distribution, which *closely* approximates the true sampling distribution!
- We can use this to compute confidence intervals when no closed form exists or illustrate uncertainty.

Do The Bootstrap!

- The `tidymodels` suite of packages is amazing here. I copied most of the code from them.

- Let's look at fitting a *nonlinear* least squares model to this data:

```
library(tidyverse)
ggplot(mtcars, aes(mpg,wt)) +
  geom_point()
```



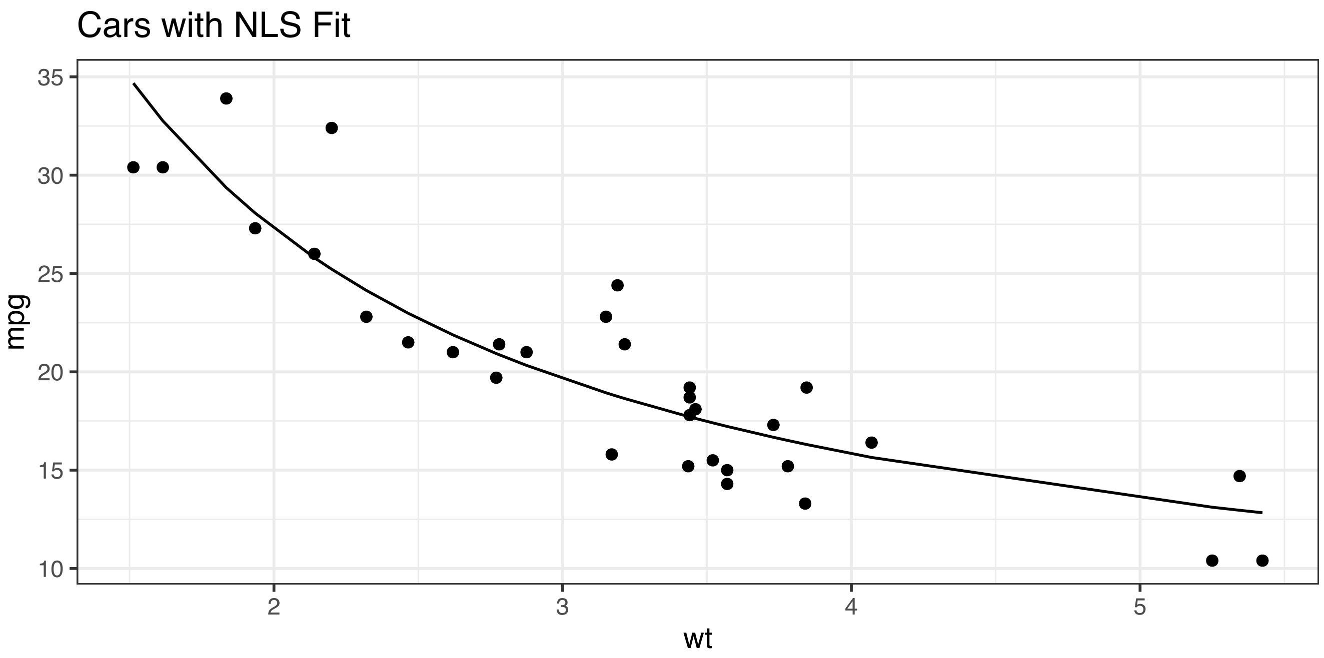
Non-linear Least Squares (NLS) for Cars

- Remember: OLS required *linear* parameters.
- NLS relaxes that:

$$y_i = f(x_i, \beta) + e_i$$

- Again want the β's.
- f is known!

```
nlsfit <- nls(mpg ~ k / wt + b,
              mtcars,
              start = list(k = 1, b = 0))

ggplot(mtcars, aes(wt, mpg)) +
    geom_point() +
    geom_line(aes(y = predict(nlsfit))) + theme_bw() + ggtitle("Cars with NLS 
```



Bootstrapping the NLS models

1. Let's create 200 bootstrap samples.

2. Estimate our NLS model on each.

3. Get coefficients from each.

4. Assess their variation.

```r
# 1.
library(rsample)
library(stats)
boots <- bootstraps(mtcars, times = N, apparent = TRUE)

# 2. a) create a wrapper for nls
fit_nls_on_bootstrap <- function(split) {
    nls(mpg ~ k / wt + b, analysis(split), start = list(k = 1, b = 0))
}

# 2. b) map wrapper on to each bootstrap sample
boot_models <-
  boots %>%
  mutate(model = map(splits, fit_nls_on_bootstrap),
         coef_info = map(model, tidy))
# 3.
boot_coefs <-
  boot_models %>%
  unnest(coef_info)
```

Bootstrapping the NLS models: Using the `rsample` package

- `rsample` functions *split* datasets. `bootstrap` draws total number of observations for `analysis` (i.e. for *training*)
- `boot_coefs` has estimates for each bootstrap sample.

```
head(boots)
```

```
## # A tibble: 6 × 2
##   splits          id
##   <list>          <chr>
## 1 <split [32/12]> Bootstrap001
## 2 <split [32/14]> Bootstrap002
## 3 <split [32/11]> Bootstrap003
## 4 <split [32/8]>  Bootstrap004
## 5 <split [32/10]> Bootstrap005
## 6 <split [32/10]> Bootstrap006
```

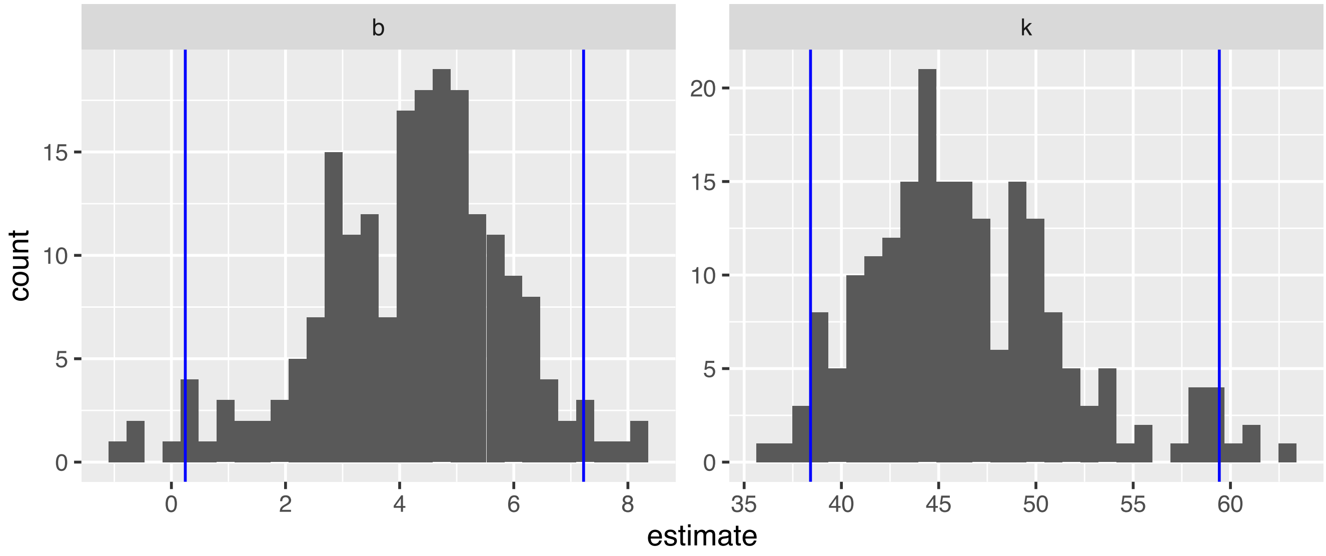
```
head(boot_coefs)
```

```
## # A tibble: 6 × 8
##   splits          id           model  term  estimate std.error statis…¹  p.value
##   <list>          <chr>        <list> <chr>    <dbl>     <dbl>    <dbl>    <dbl>
## 1 <split [32/12]> Bootstrap001 <nls>  k        48.0       4.61    10.4  1.76e-11
## 2 <split [32/12]> Bootstrap001 <nls>  b         4.22      1.70     2.48 1.90e- 2
## 3 <split [32/14]> Bootstrap002 <nls>  k        43.2       3.37    12.8  1.04e-13
## 4 <split [32/14]> Bootstrap002 <nls>  b         4.61      1.14     4.04 3.40e- 4
## 5 <split [32/11]> Bootstrap003 <nls>  k        45.9       4.50    10.2  2.85e-11
## 6 <split [32/11]> Bootstrap003 <nls>  b         5.05      1.67     3.02 5.06e- 3
## # … with abbreviated variable name ¹statistic
```

Confidence Intervals

- We can now easily compute and plot bootstrap CIs!
- Remember: *percentile method* just takes 2.5 and 97.5 quantiles of bootstrap sampling distribution as bounds of CI.

```
percentile_intervals <- int_pctl(boot_models, coef_info)
ggplot(boot_coefs, aes(estimate)) +
  geom_histogram(bins = 30) +
  facet_wrap( ~ term, scales = "free") +
  geom_vline(aes(xintercept = .lower), data = percentile_intervals, col = "blu
  geom_vline(aes(xintercept = .upper), data = percentile_intervals, col = "blu
```



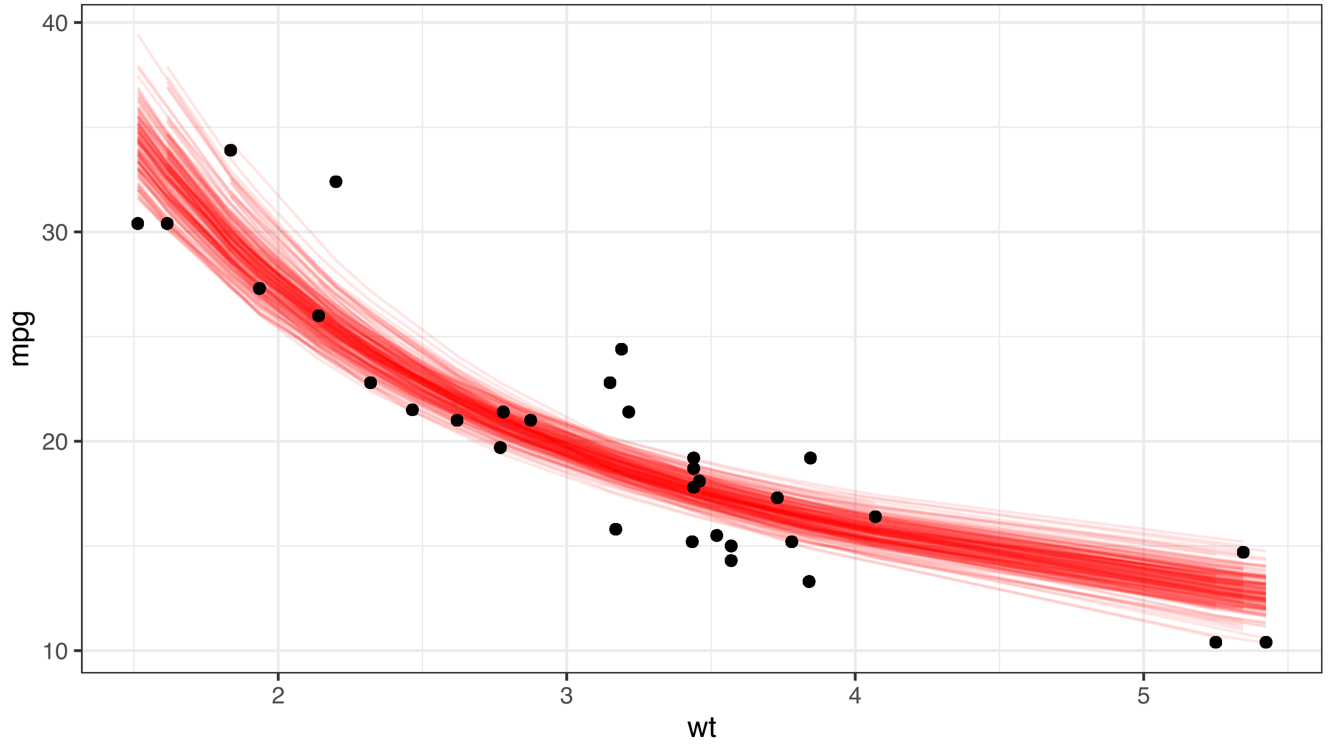
Illustrate More Uncertainty

- It's also easy to illustrate uncertainty in fit with this.

- Let's get predicted values with `augment` from our models.

```
boot_aug <-
  boot_models %>%
  sample_n(200) %>%
  mutate(augmented =
           map(model, augment)) %
  unnest(augmented)
```

```
ggplot(boot_aug, aes(wt, mpg)) +
  geom_line(aes(y = .fitted, group = id), alpha = .1, col = "red") +
  geom_point() + theme_bw()
```



Cross Validation

- Last week we encountered the test MSE.
- In simulation studies, we can compute it, but in real life? It's much harder to obtain a true test data set.
- What we can do in practice, however, is to **hold out** part of our data for testing purposes.
- We just set it aside at the beginning and don't use it for training.

Several Approaches:

1. Validation Set
2. Leave-one-out cross validation (LOOCV)
3. k-fold Cross Validation (k-CV)

K-fold Cross Validation (k-CV)

- Randomly divide your data into k groups of equal size.
- train model on all but last groups (*folds*), compute MSE on last fold.
- train model on all but penultimat fold, compute MSE there, etc
- The *k-fold CV* is then

$$CV_{(k)} = \frac{1}{k} \sum_{i=1}^{k} \mathrm{MSE}_i$$

K-fold Cross Validation (k-CV)

- Randomly divide your data into k groups of equal size.
- train model on all but last groups (*folds*), compute MSE on last fold.
- train model on all but penultimat fold, compute MSE there, etc
- The *k-fold CV* is then

$$CV_{(k)} = \frac{1}{k} \sum_{i=1}^{k} \text{MSE}_i$$

- We have to fit the model k times here.
- Previous methods (LOOCV) are much more costly in terms of computing time.
- In practice one often chooses $k = 5$ or $k = 10$.
- Let's look again at the `rsample` package as to how to set this up!

`rsample` package again

Splits for Bootstrap Samples

```
library(rsample) # already loaded...
bcars <- bootstraps(mtcars, times = 3)
head(bcars,n=3)
```

```
## # Bootstrap sampling
## # A tibble: 3 × 2
##   splits          id
##   <list>          <chr>
## 1 <split [32/15]> Bootstrap1
## 2 <split [32/10]> Bootstrap2
## 3 <split [32/14]> Bootstrap3
```

```
nrow(analysis(bcars$splits[[1]]))
```

```
## [1] 32
```

Splits for Testing/Training

```
set.seed(1221)
cvcars <- vfold_cv(mtcars, v = 10, repeats = 10)
head(cvcars,n=3)
```

```
## # A tibble: 3 × 3
##   splits         id       id2
##   <list>         <chr>    <chr>
## 1 <split [28/4]> Repeat01 Fold01
## 2 <split [28/4]> Repeat01 Fold02
## 3 <split [29/3]> Repeat01 Fold03
```

```
nrow(analysis(cvcars$splits[[1]]))
```

```
## [1] 28
```

```
nrow(assessment(cvcars$splits[[1]]))
```

```
## [1] 4
```

Regularized Regression and Variable Selection

What to do when you have 307 potential predictors?

The Linear Model is Great

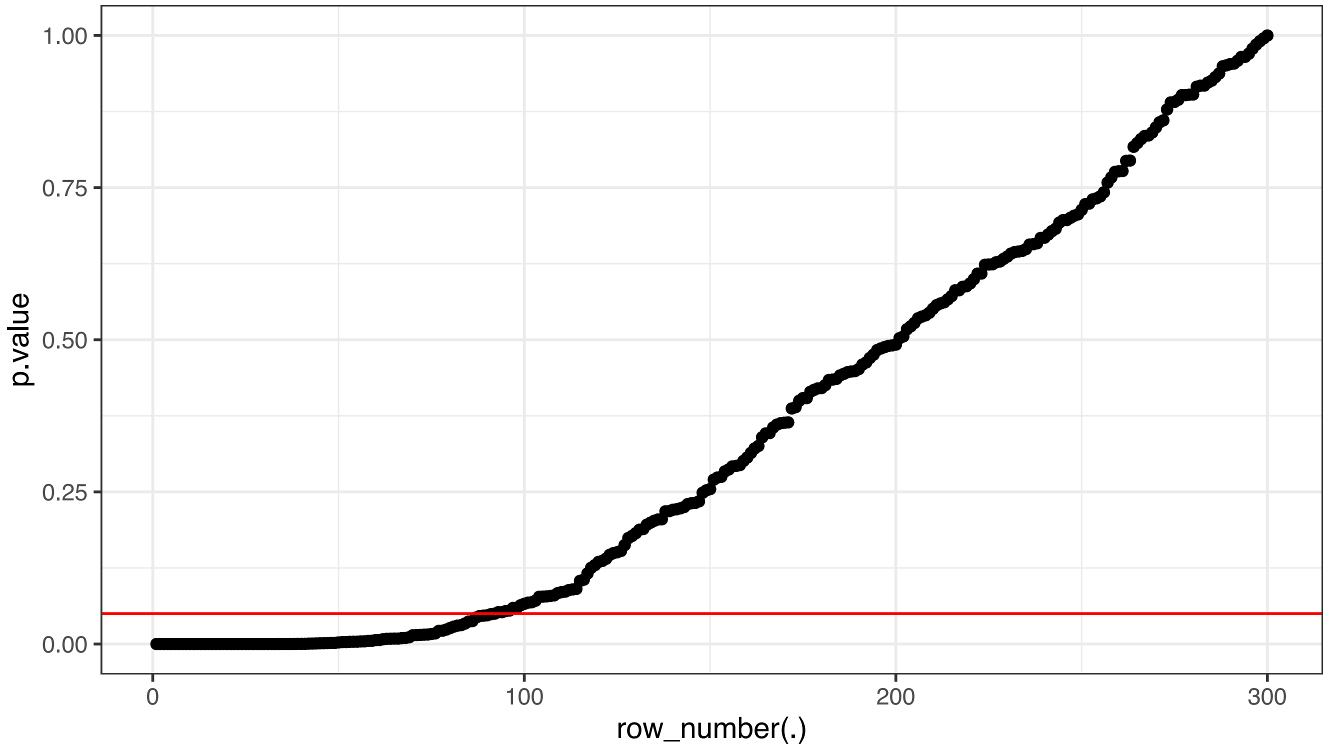
$$Y = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$$

- If we have relatively few parameters $p << n$ we typically have low variance with the linear model.
- This deteriorates with larger p. With $p > n$ (more parameters than data points) we cannot even compute our β with OLS.
- We often look for stars `***` when deciding which variable should be part of a model. Correct only under certain assumptions.
- Despite `***` we don't discover whether variable x_j is an important predictor of the outcome.
- OLS will never deliver an estimate β_j *exactly* zero.
- Example?

307 predictors for `Sale_Price`

```
a = AmesHousing::make_ames()  # house price sales
lma = lm(Sale_Price ~ . , data = a) # include all variables
broom::tidy(lma) %>% select(p.value) %>% arrange(desc(p.value)) %>%
  ggplot(aes(x = row_number(.),y = p.value)) + geom_point() + theme_bw() + geom_hline(yintercept = 0.05, color =
```

- 307 predictors! Which ones to include?
- Wait, we still have **p-values**!
- Can't we just take all predictors with `p < 0.05`?
- why not `p < 0.06`?
- why not `p < 0.07`?



AmesHousing

```
# from: https://uc-r.github.io/regularized_regression
# Create training (70%) and
# test (30%) sets for the AmesHousing::make_ames() data.

set.seed(123)
ames_split <- initial_split(a, prop = .7, strata = "Sale_Price")
ames_train <- training(ames_split)
ames_test  <- testing(ames_split)


# extract model matrix from both: code each factor level as a dummy
# don't take the intercept ([,-1])
ames_train_x <- model.matrix(Sale_Price ~ ., ames_train)[, -1]
ames_train_y <- log(ames_train$Sale_Price)

ames_test_x <- model.matrix(Sale_Price ~ ., ames_test)[, -1]
ames_test_y <- log(ames_test$Sale_Price)


# What is the dimension of of your feature matrix?
dim(ames_train_x)
```

```
## [1] 2049  308
```

House Price Data: `AmesHousing` Package

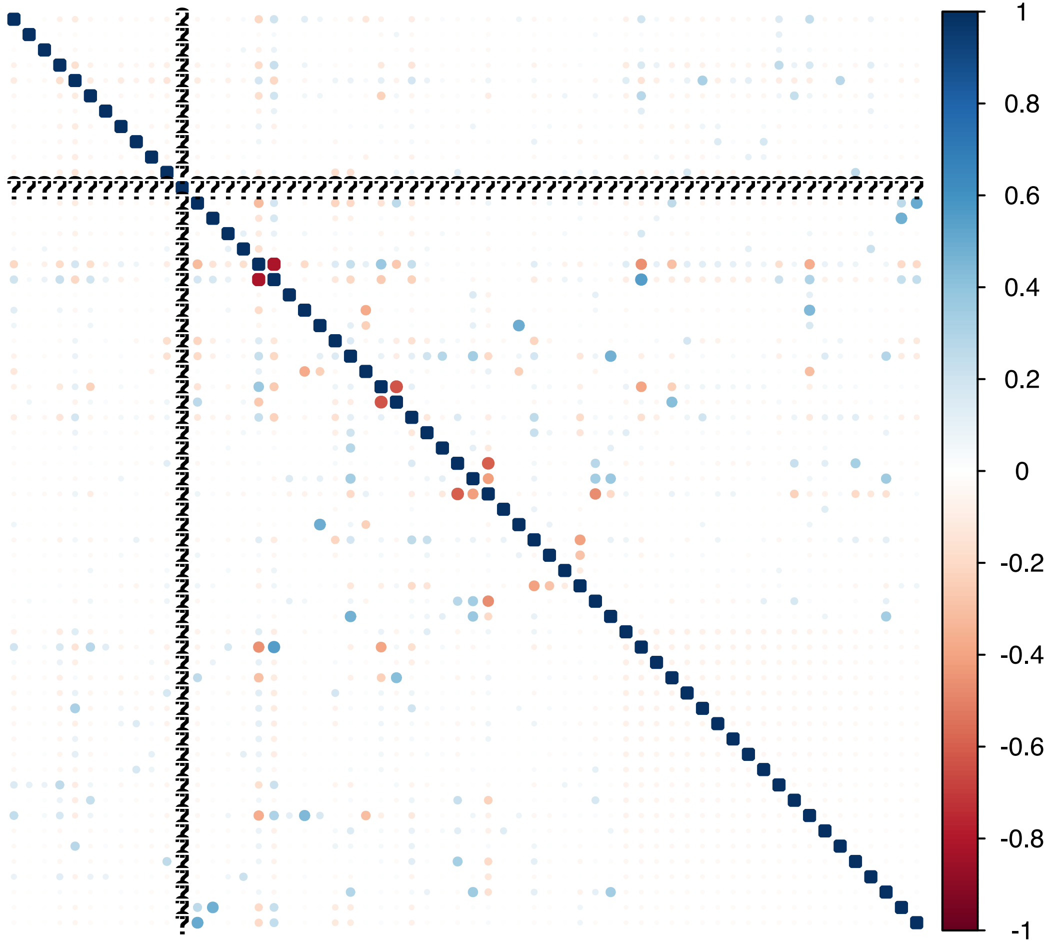
- Lots of multicollinearity among our predictors.

- This will inflate variance of our estimates.

- Here is the correlation matrix of the first 60 predictors:

```
ca = cor(ames_train_x[,1:60])
corrplot::corrplot(ca,
                   tl.pos = "n")
```

- Darker colours spell trouble!



Regularization: Add a *Penalty*

- We can add a *penalty* P to the OLS objective:

$$\min SSE + P$$

- P will *punish* the algorithm for choosing *too large* parameter values
- Looking closely at P is beyond our scope here.
- But we will show how to use two popular methods.

Regularization: Add a *Penalty*

- We can add a *penalty* P to the OLS objective:

$$\min SSE + P$$

- P will *punish* the algorithm for choosing *too large* parameter values
- Looking closely at P is beyond our scope here.
- But we will show how to use two popular methods.

- Ridge Objective: L_2 penalty

$$\min SSE + \lambda \sum_{j=1}^{P} \beta_j^2$$

- λ is a *tuning parameter*: $\lambda = 0$ is no penalty.
- Lasso Objective: L_1 penalty

$$\min SSE + \lambda \sum_{j=1}^{P} |\beta_j|$$

Ridge Regression with the `glmnet` package

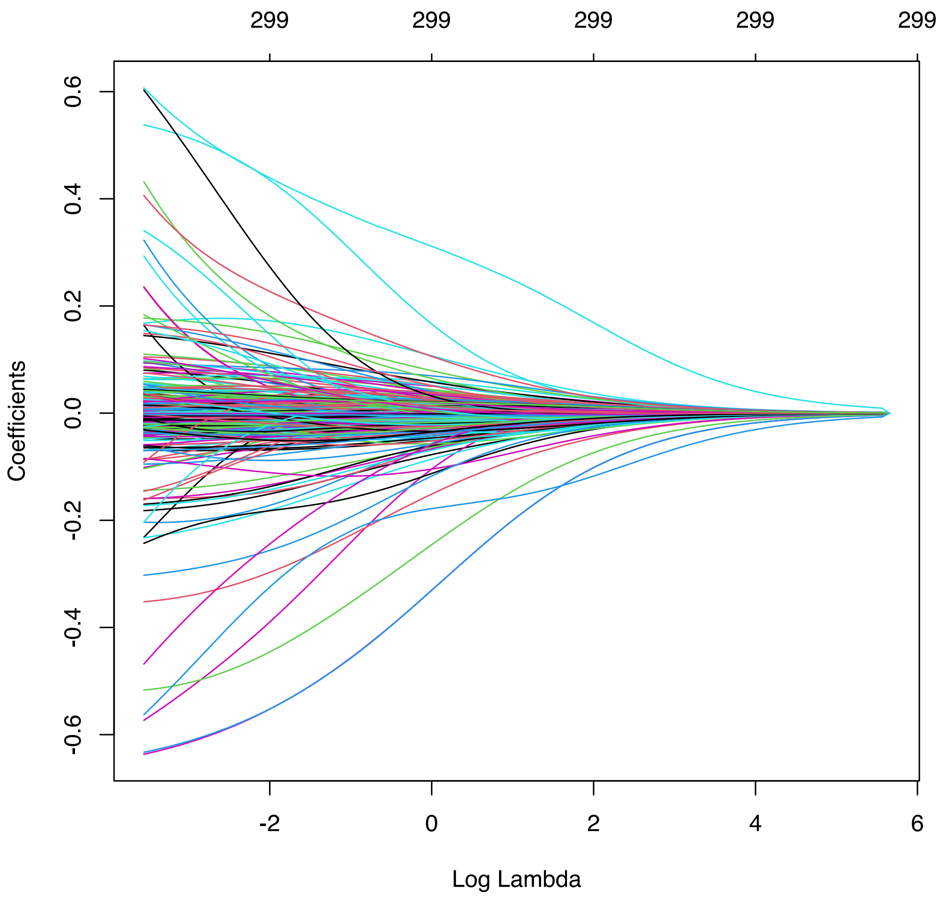
Ridge in `AmesHousing`

- Parameter `alpha` $\in [0, 1]$ governs whether we do *Ridge* or *Lasso*. Ridge with `alpha = 0`.

- Using the `glmnet::glmnet` function by default *standardizes* all regressors

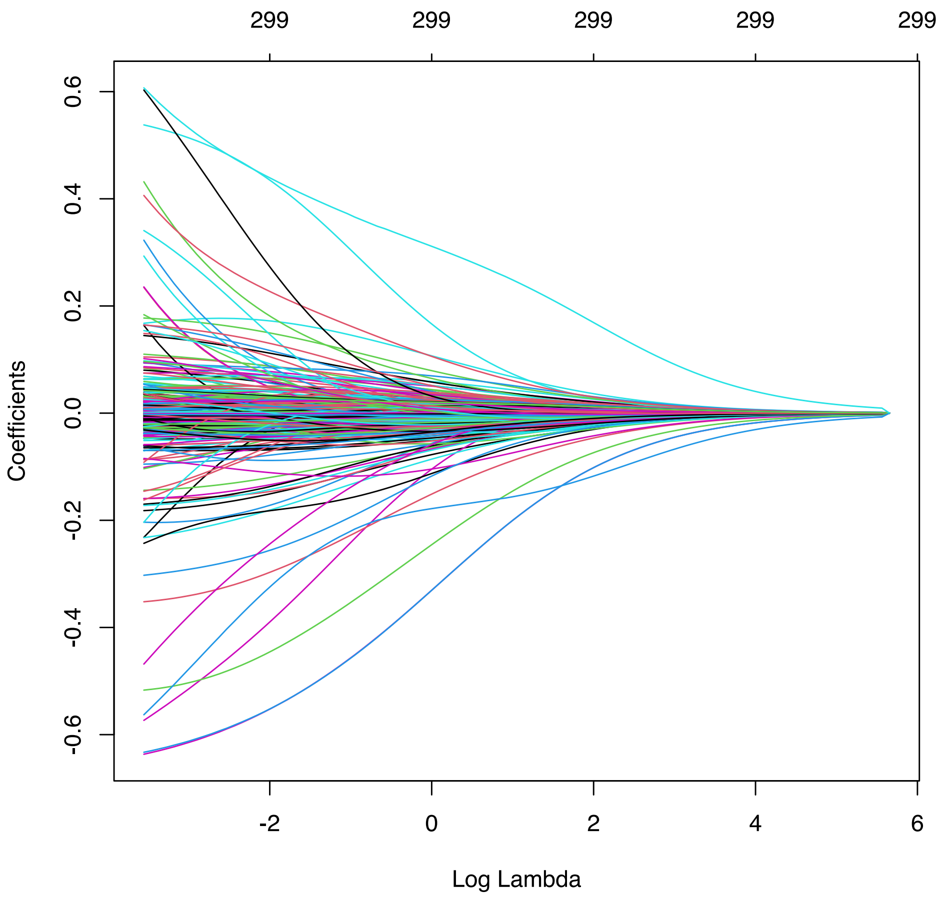
- `glmnet::glmnet` will run for many values of λ.

```
# Apply Ridge regression to ames data
library(glmnet)
ames_ridge <- glmnet(
  x = ames_train_x,
  y = ames_train_y,
  alpha = 0
)

plot(ames_ridge, xvar = "lambda")
```



Ridge in `AmesHousing`

- Each line is the point estimate for one regressor at a given λ
- All regressors are non-zero, but get arbitrarily small at high λ. We compress considerable variation in estimates (remember those are all standardized!)
- So, what's the right λ then?
- λ is a tuning parameter.
- Let's do CV to find out the best λ.

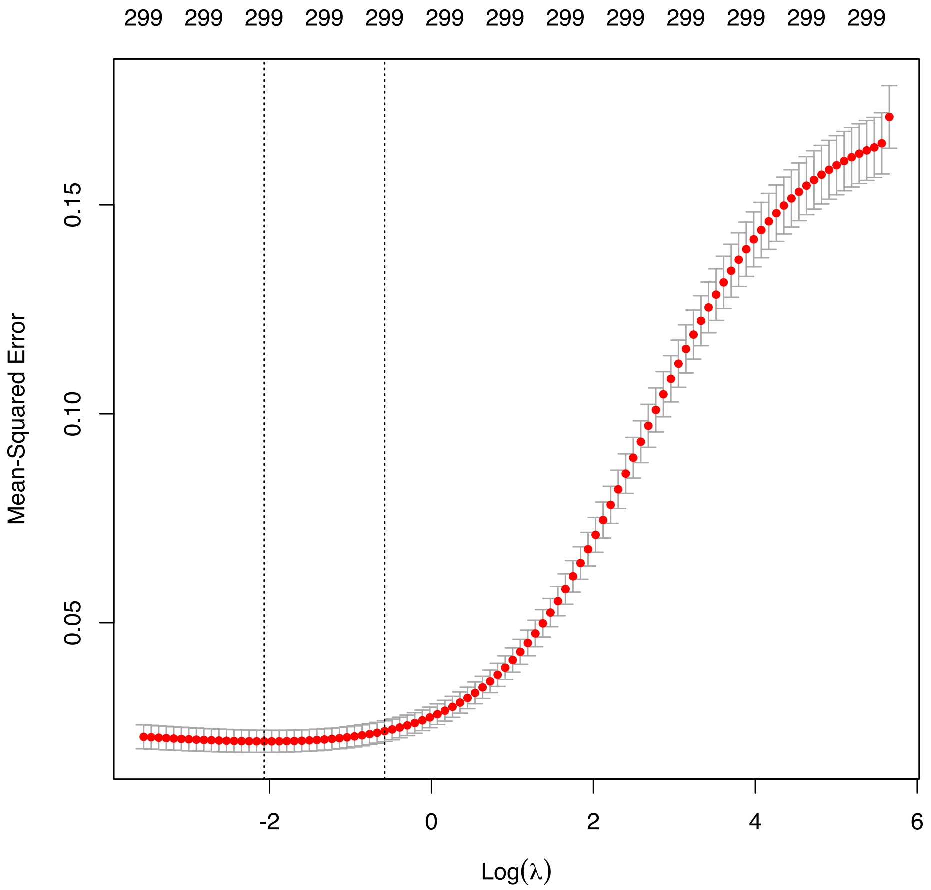


Tuning Ridge

- Remember what we said about **Overfitting**: there is a sweet spot that balances flexibility (here: many regressors) and interpretability (here: few regressors).

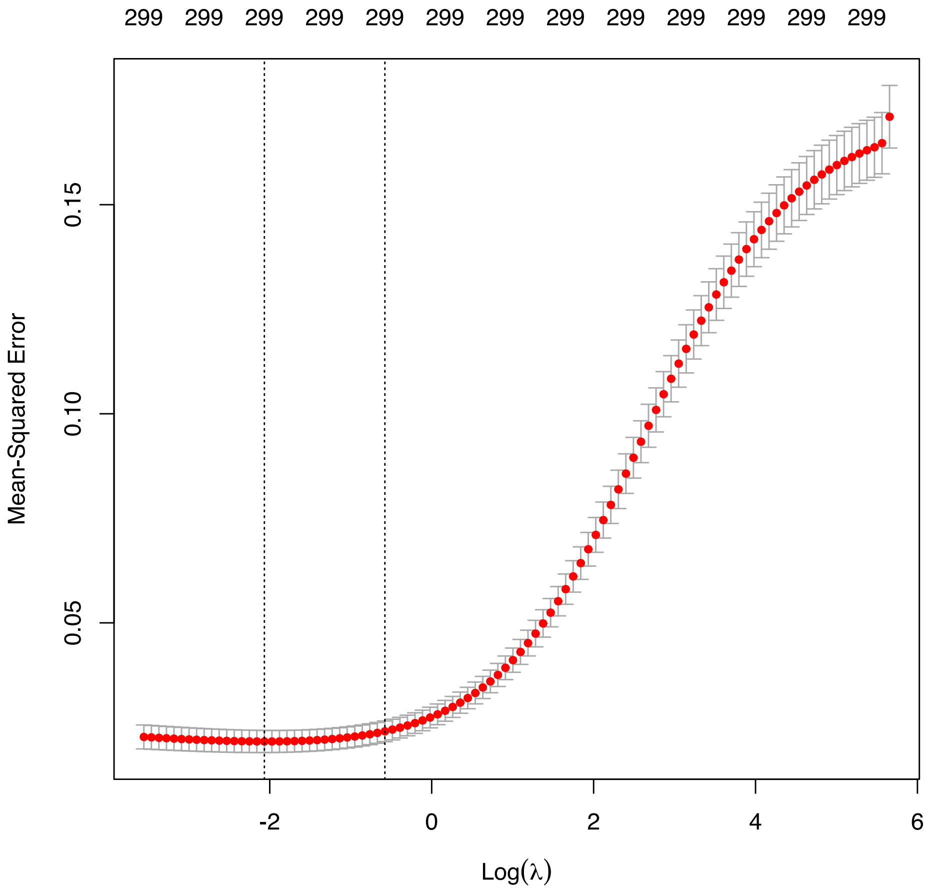
- Let's do k-fold CV to compute our test MSE, built in with `glmnet::cv.glmnet`:

```
# Apply CV Ridge regression to ames data
ames_ridge <- cv.glmnet(
  x = ames_train_x,
  y = ames_train_y,
  alpha = 0
)

# plot results
plot(ames_ridge)
```



Tuning Ridge

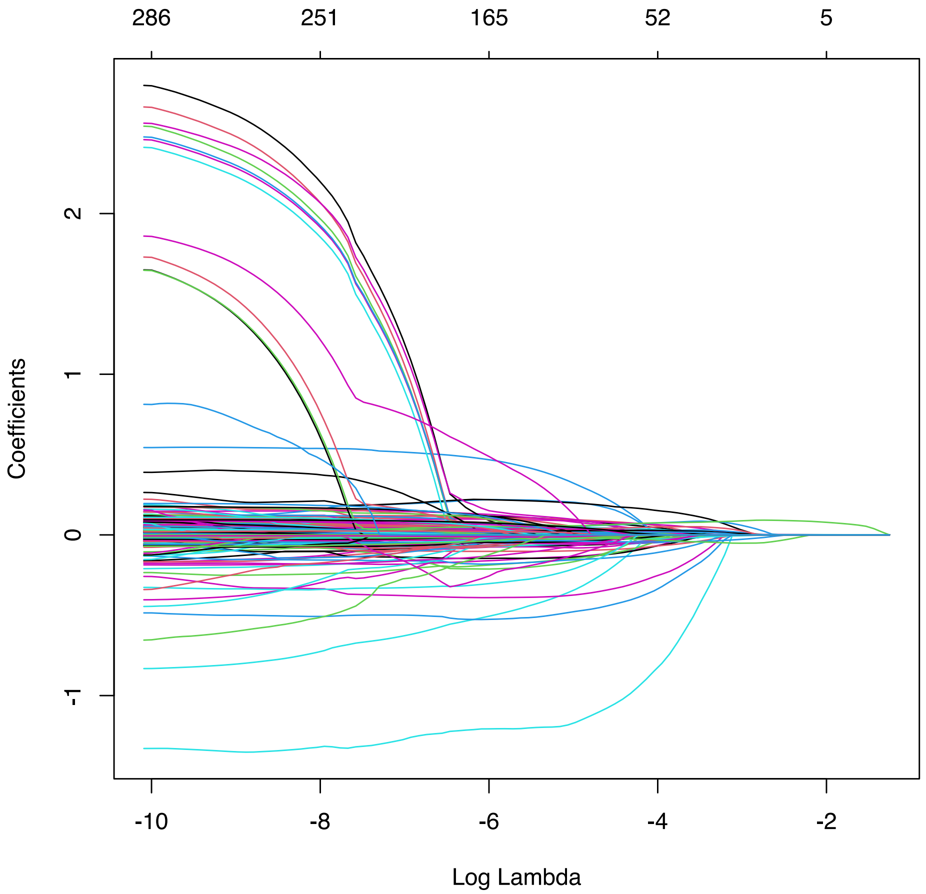
- The dashed vertical lines mark the minimum MSE and the largest λ within one std error of this minimum (to the right of the first lines).
- We would choose a lambda withing those two dashed lines.
- Remember that this keeps all variables.



lasso (*least absolute shrinkage and selection operator*)

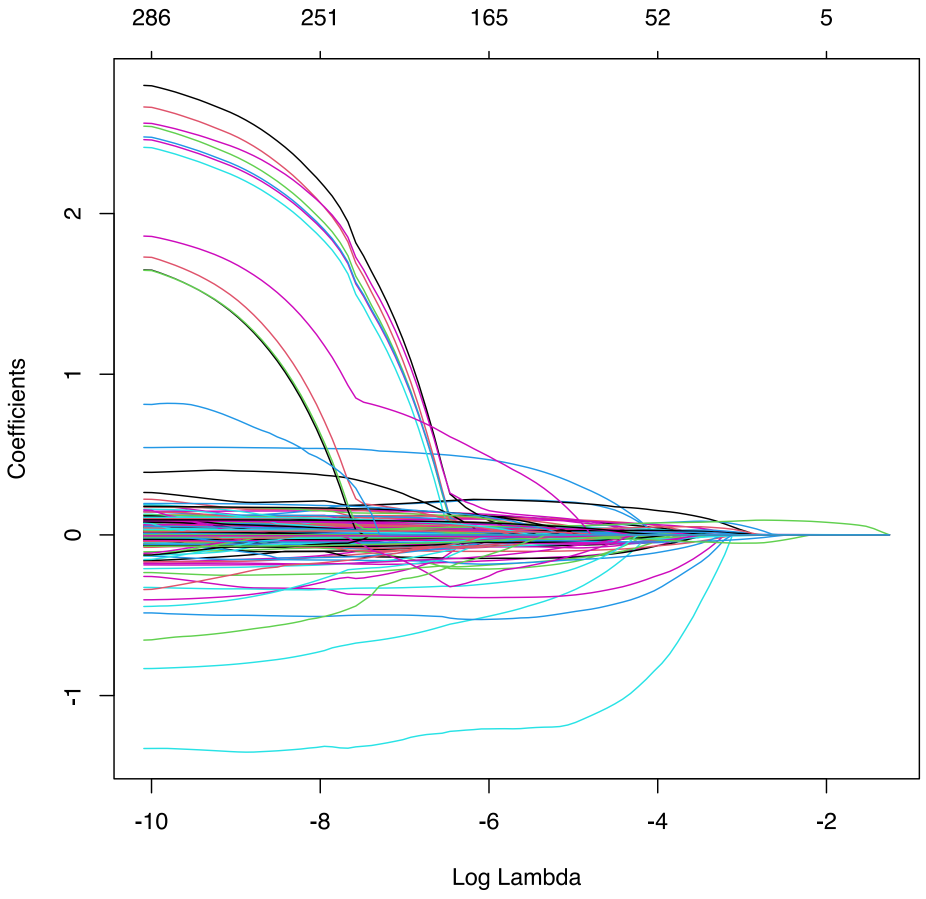
- Lasso with `alpha = 1`.
- You will see that this forces some estimates to zero.
- Hence it reduces the number of variables in the model

```
ames_lasso <- glmnet(
  x = ames_train_x,
  y = ames_train_y,
  alpha = 1
)
# plot results
plot(ames_lasso, xvar = "lambda")
```



lasso (*least absolute shrinkage and selection operator*)

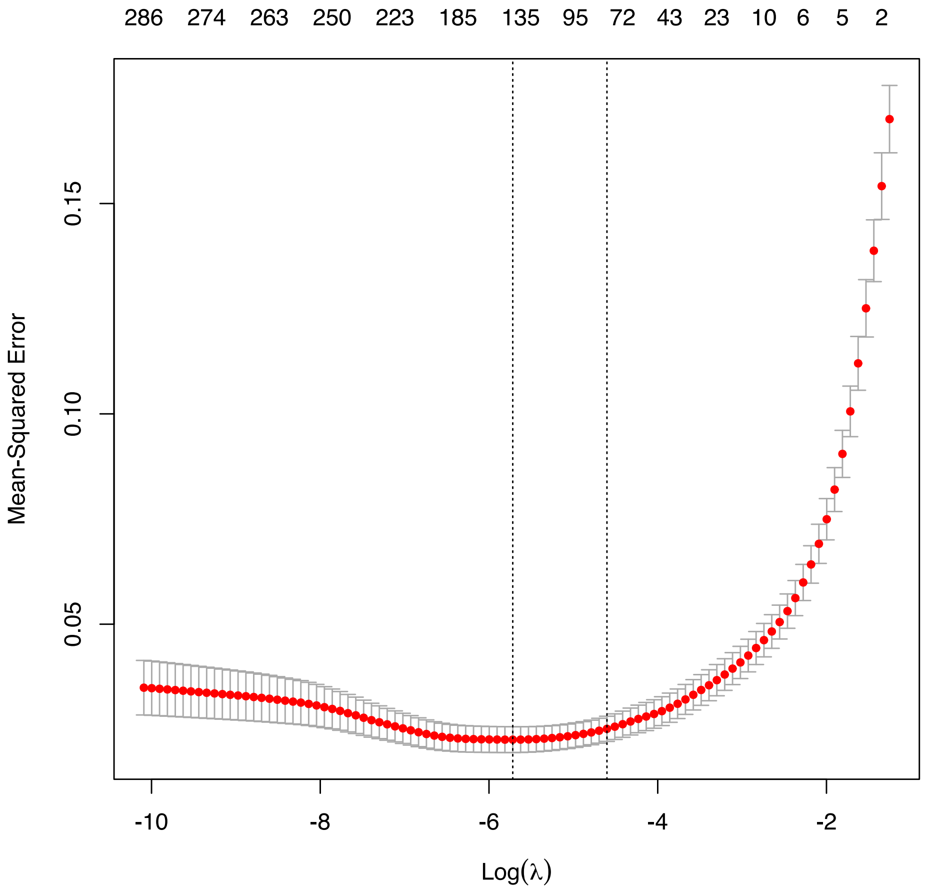
- Huge variation in estimates gets shrunken.
- The top bar of the graph shows number of active variables for each λ.
- Again: What's the right λ then?
- Again: let's look at the test MSE!



Tuning Lasso

- Let's use the same function as before.
- Let's do k-fold CV to compute our test MSE, built in with `glmnet::cv.glmnet`:

```
ames_lasso <- cv.glmnet(
  x = ames_train_x,
  y = ames_train_y,
  alpha = 1
)
# plot results
plot(ames_lasso)
```



Tuning Lasso

```r
min(ames_lasso$cvm)          # minimum MSE
```

```
## [1] 0.02255555
```

```r
ames_lasso$lambda.min       # lambda for this min MSE
```

```
## [1] 0.00328574
```

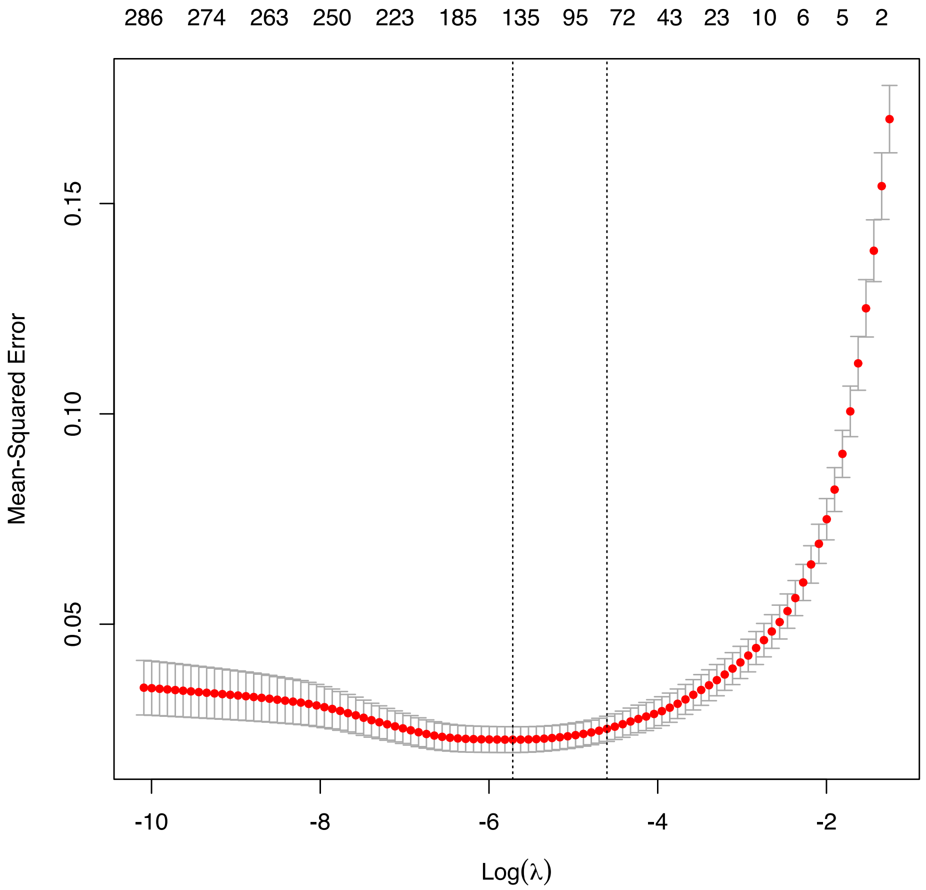
```r
# 1 st.error of min MSE
ames_lasso$cvm[ames_lasso$lambda == ames_lasso$lambda
```

```
## [1] 0.02512657
```

```r
ames_lasso$lambda.1se   # lambda for this MSE
```

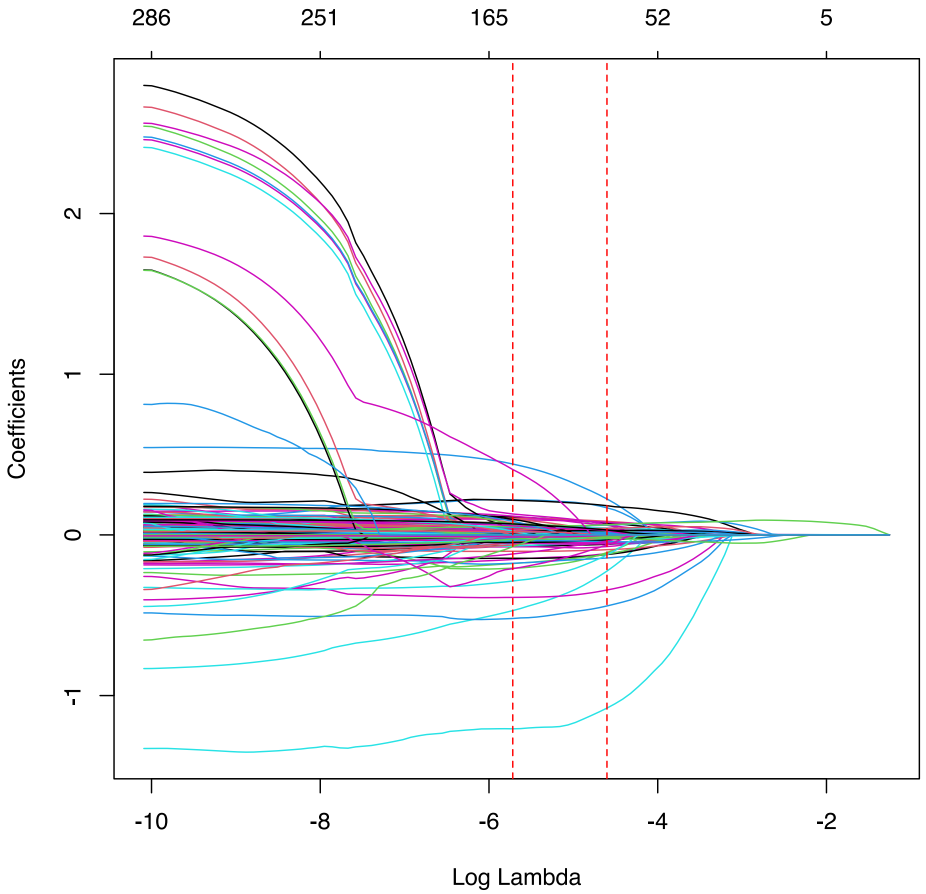
```
## [1] 0.01003418
```

- So: at MSE-minimizing λ, we went down to < 139 variables.
- Going 1 SE to the right incurs slightly higher MSE, but important reduction in variables!



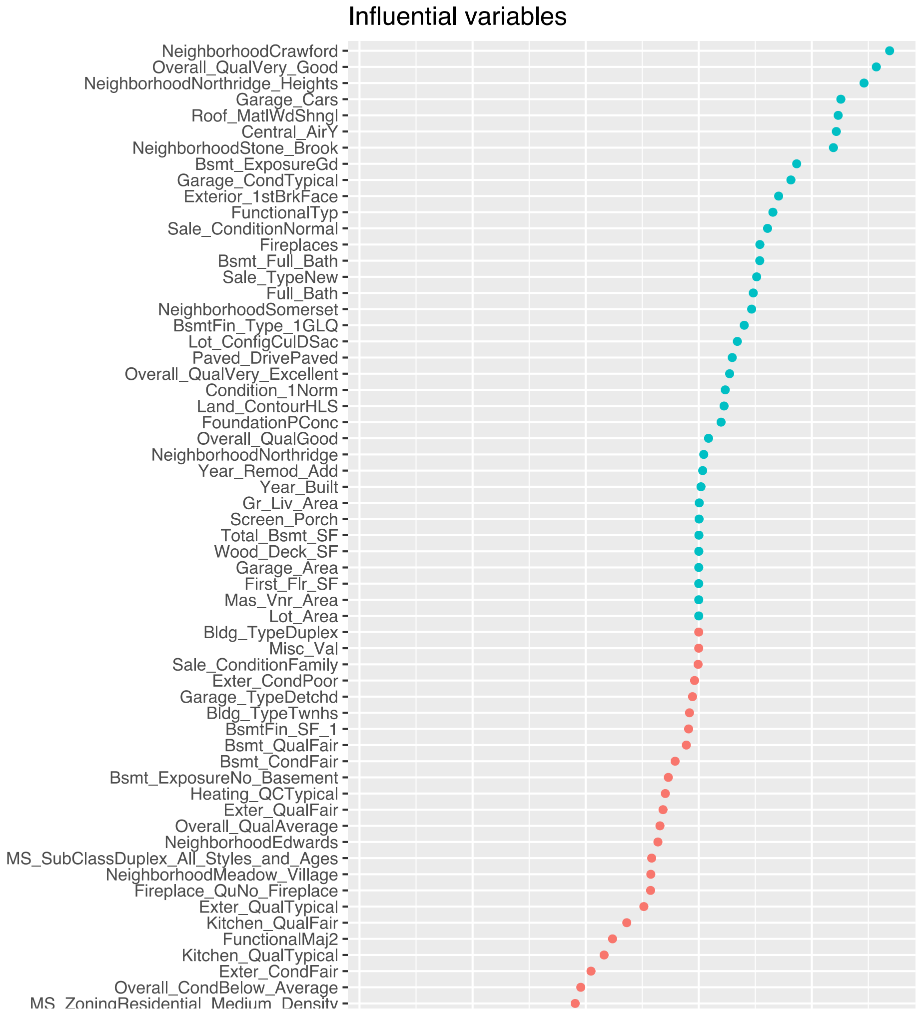
Lasso predictors at optimal MSEs

- Let's look again at coef estimates
- The red dashed lines are minimal λ and `lambda.1se`
- Depending on your task, the second line may be acceptable.



lasso vars

- So, the lasso really *selects* variables.
- Which ones are the most influential variables then?
- Remember, this is about finding the best *predictive* model.

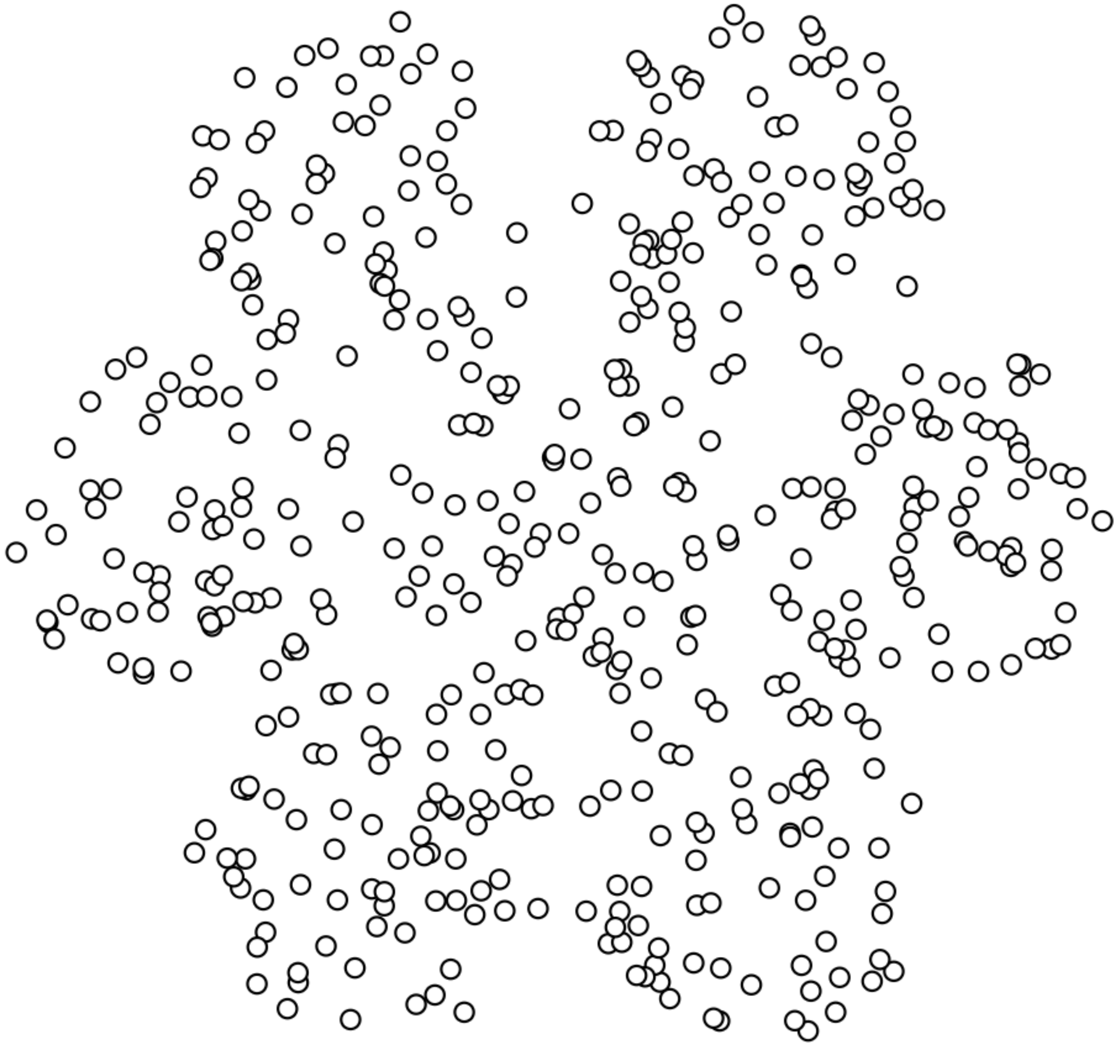


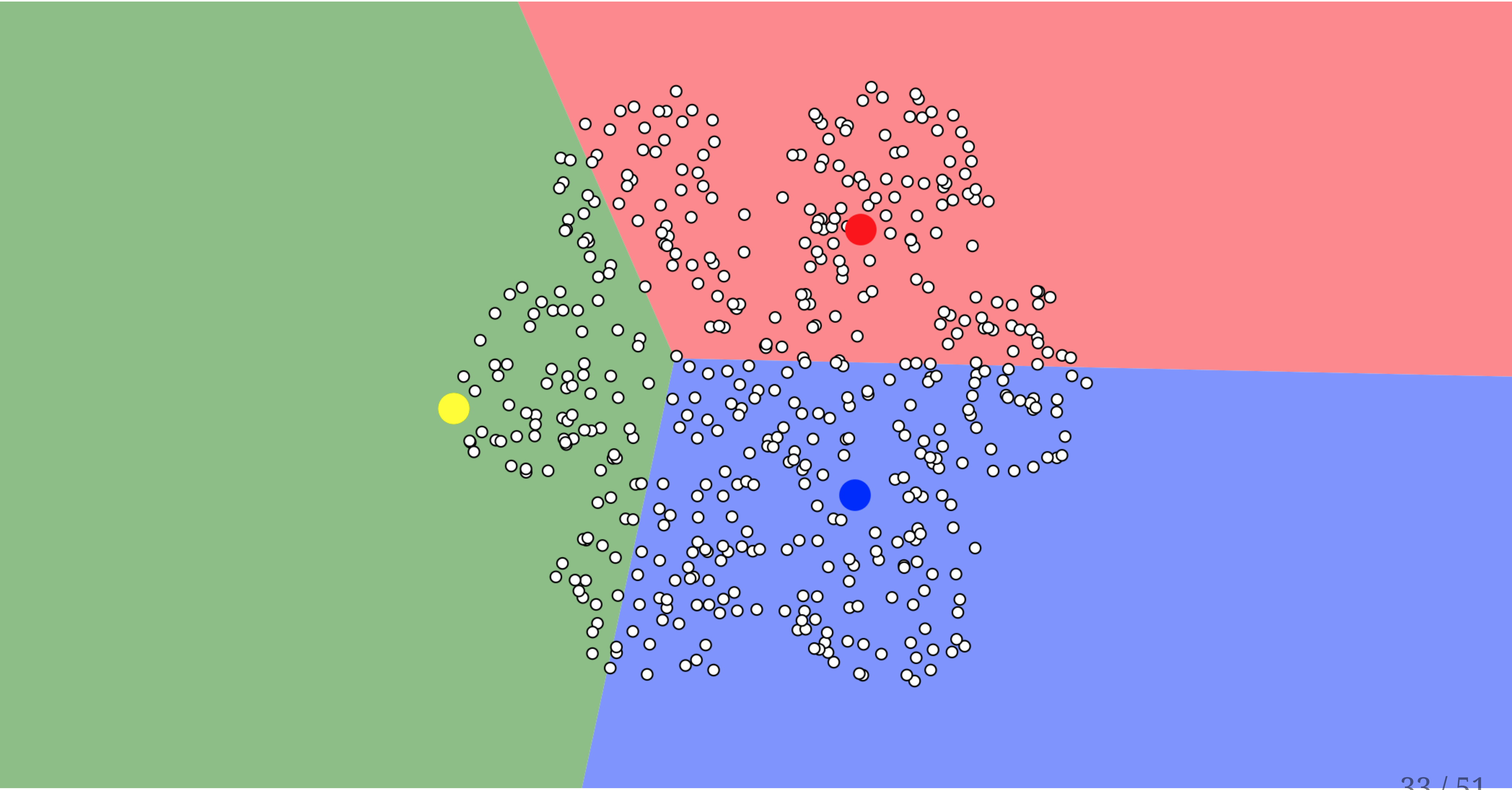
Unsupervised Methods

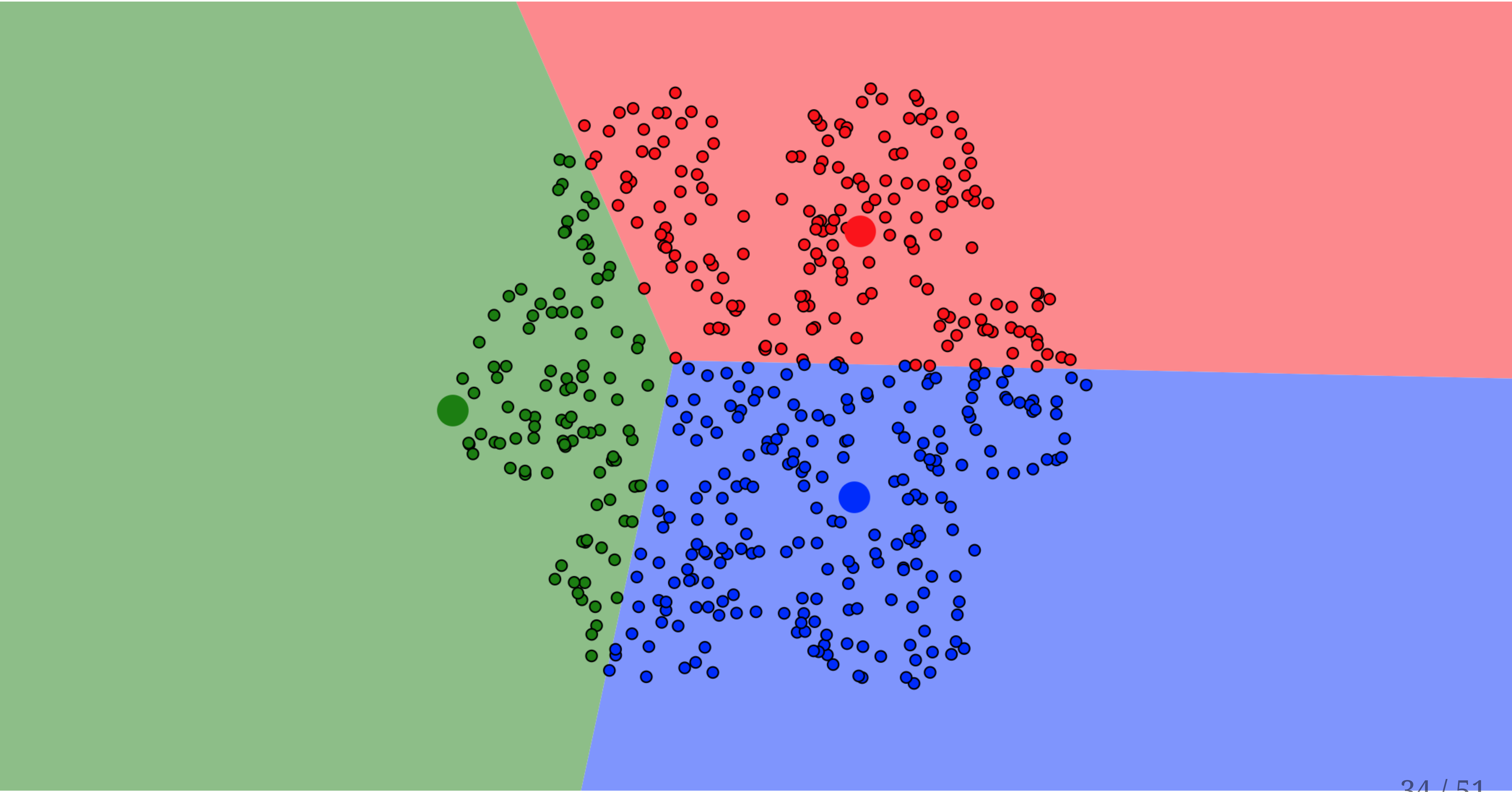
Unsupervised Methods

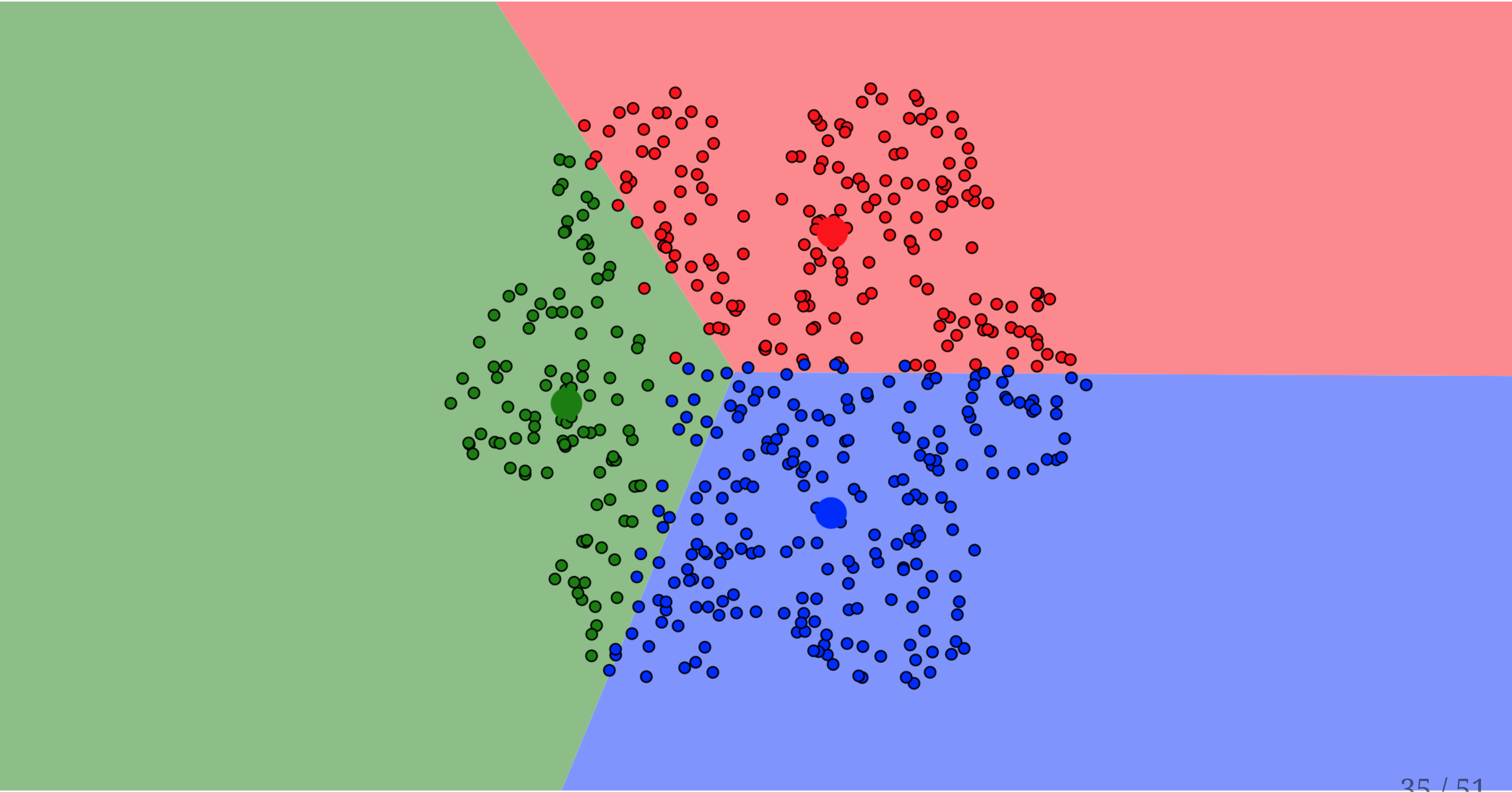
- Remember: in this class of methods we don't have a designated *output* y for our *input* variables x.
- We will talk about about Clustering methods
- We won't have time for Principal Component Analysis (PCA).
- Both of those are useful to *summarise* high-dimensional datasets.

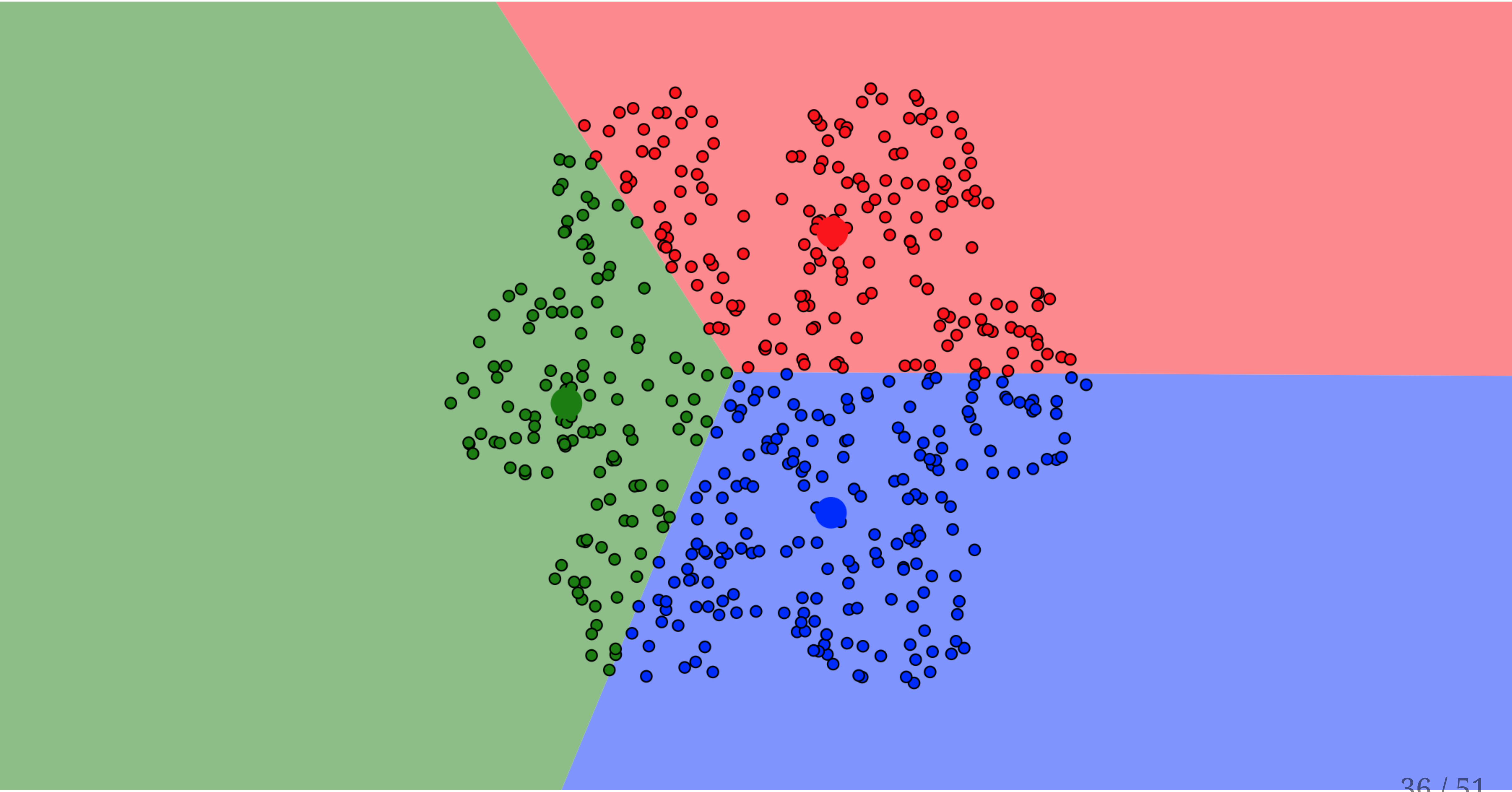
K-means Clustering

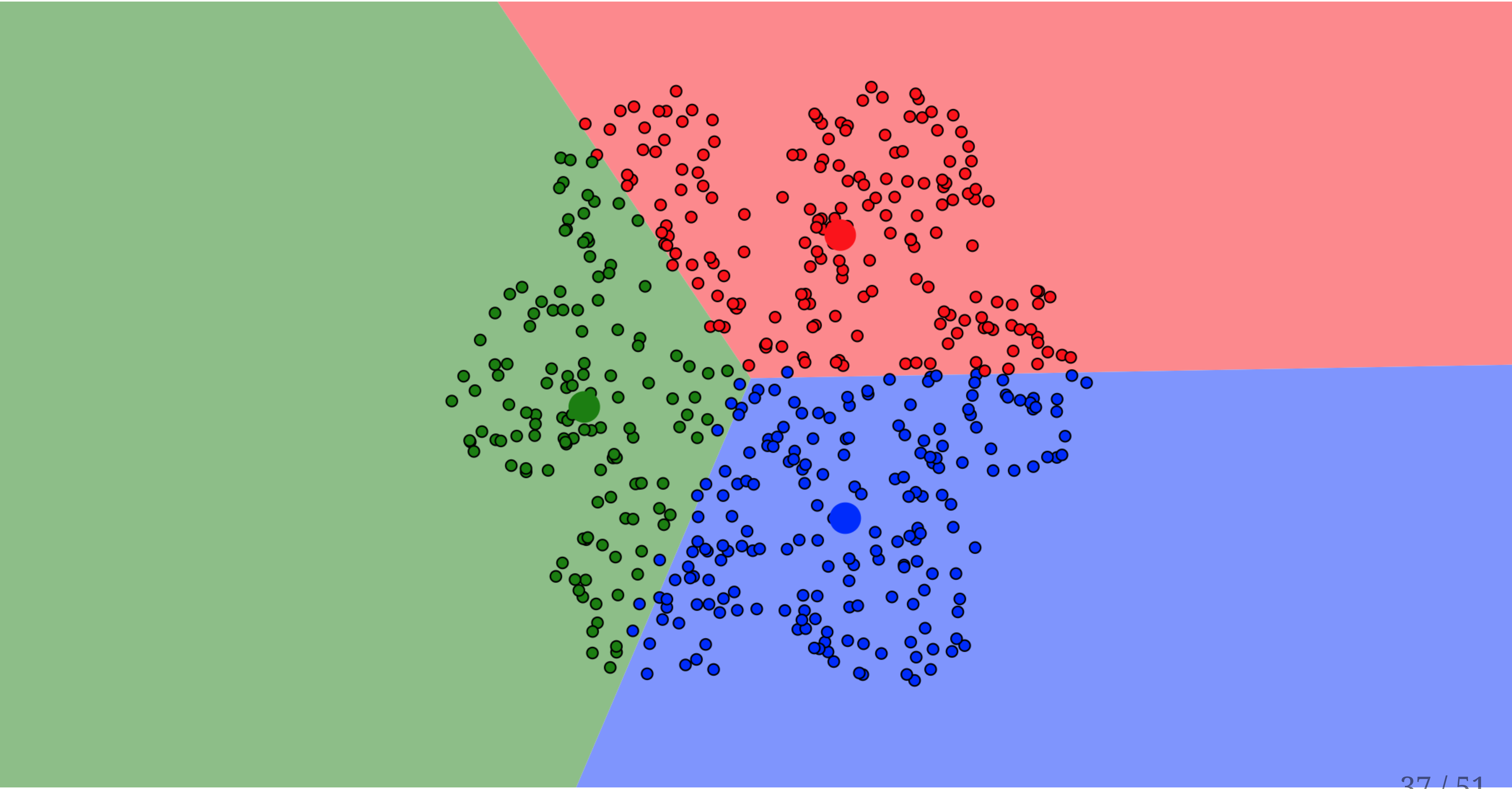


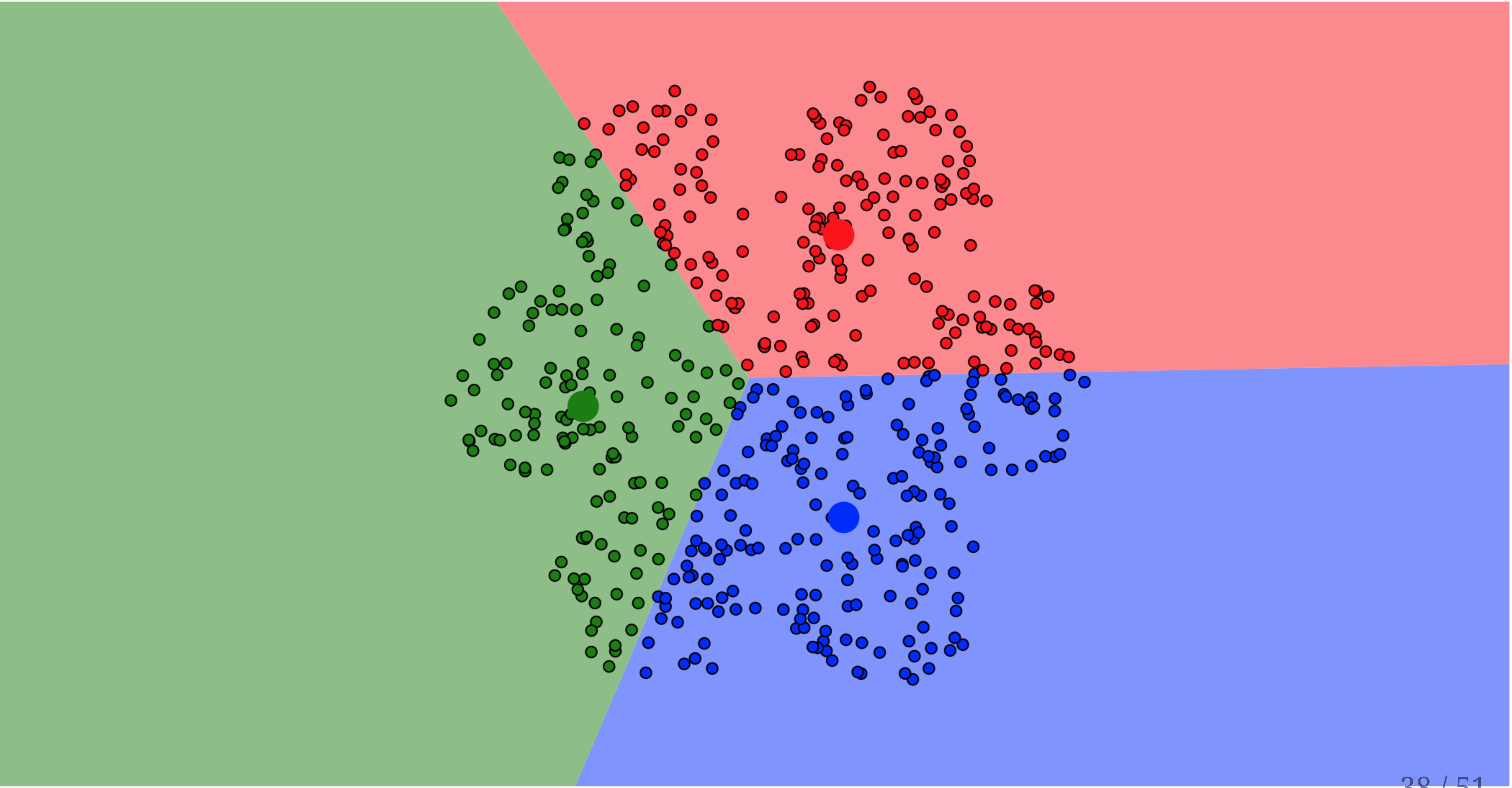


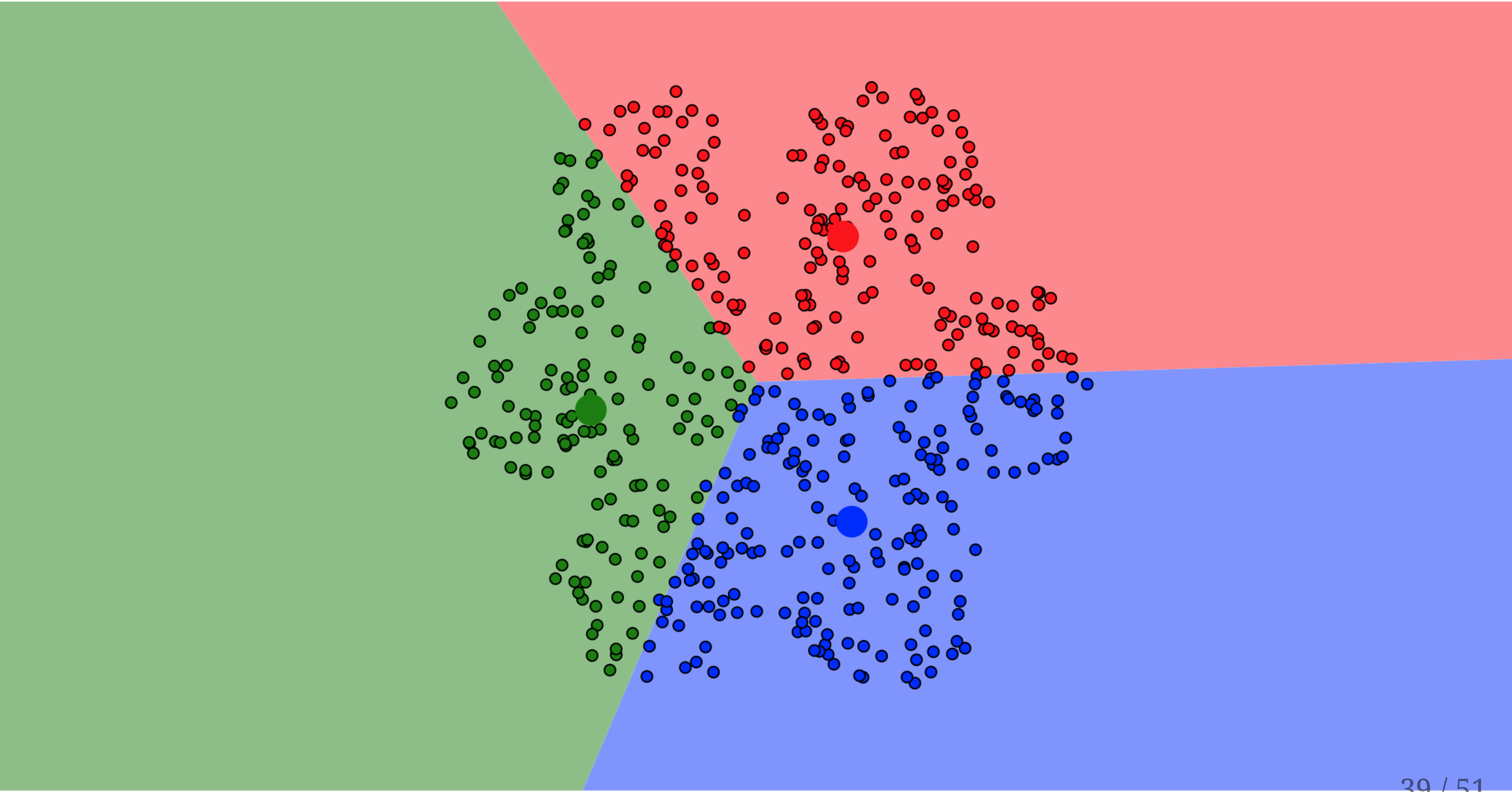


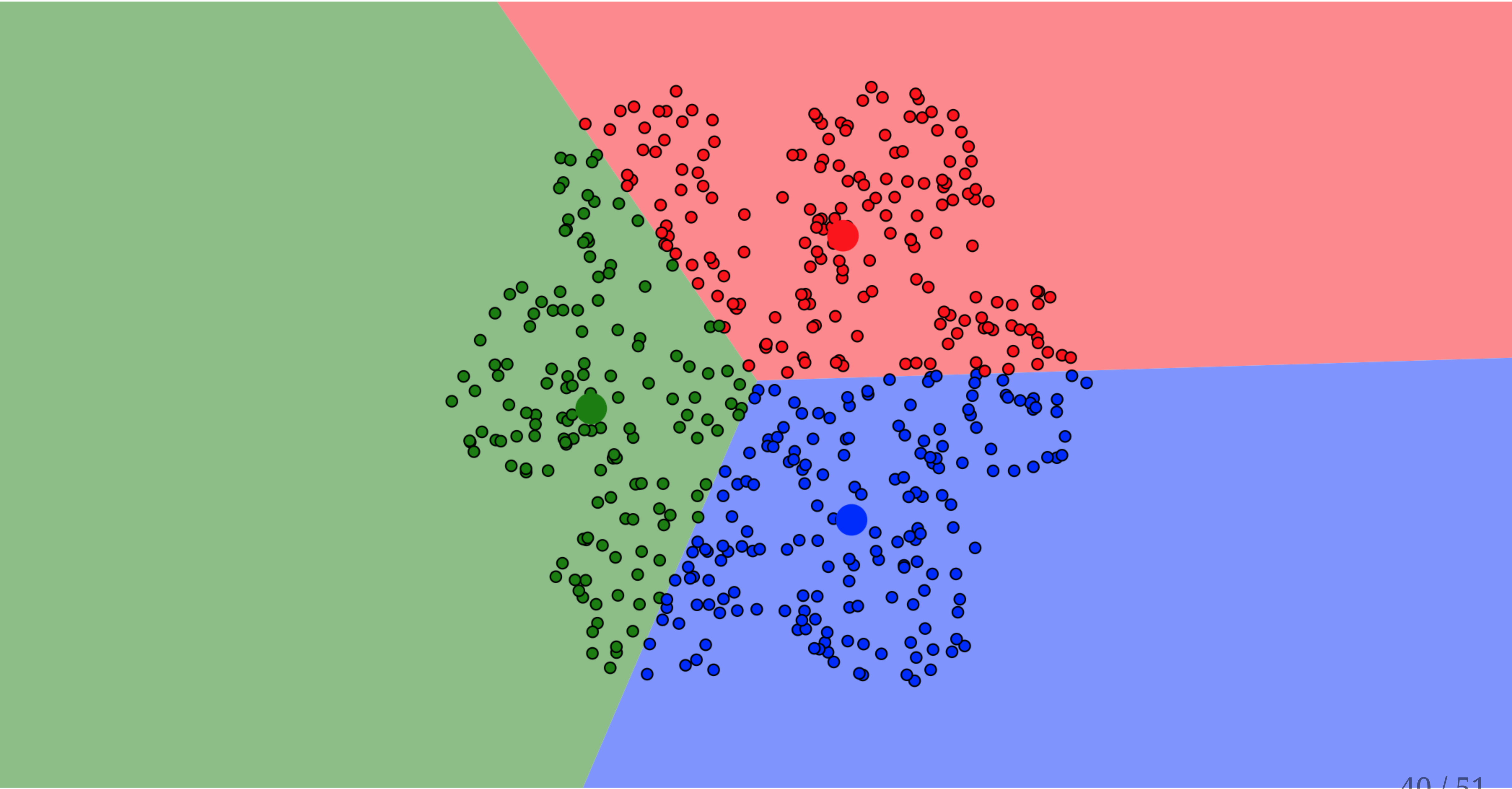


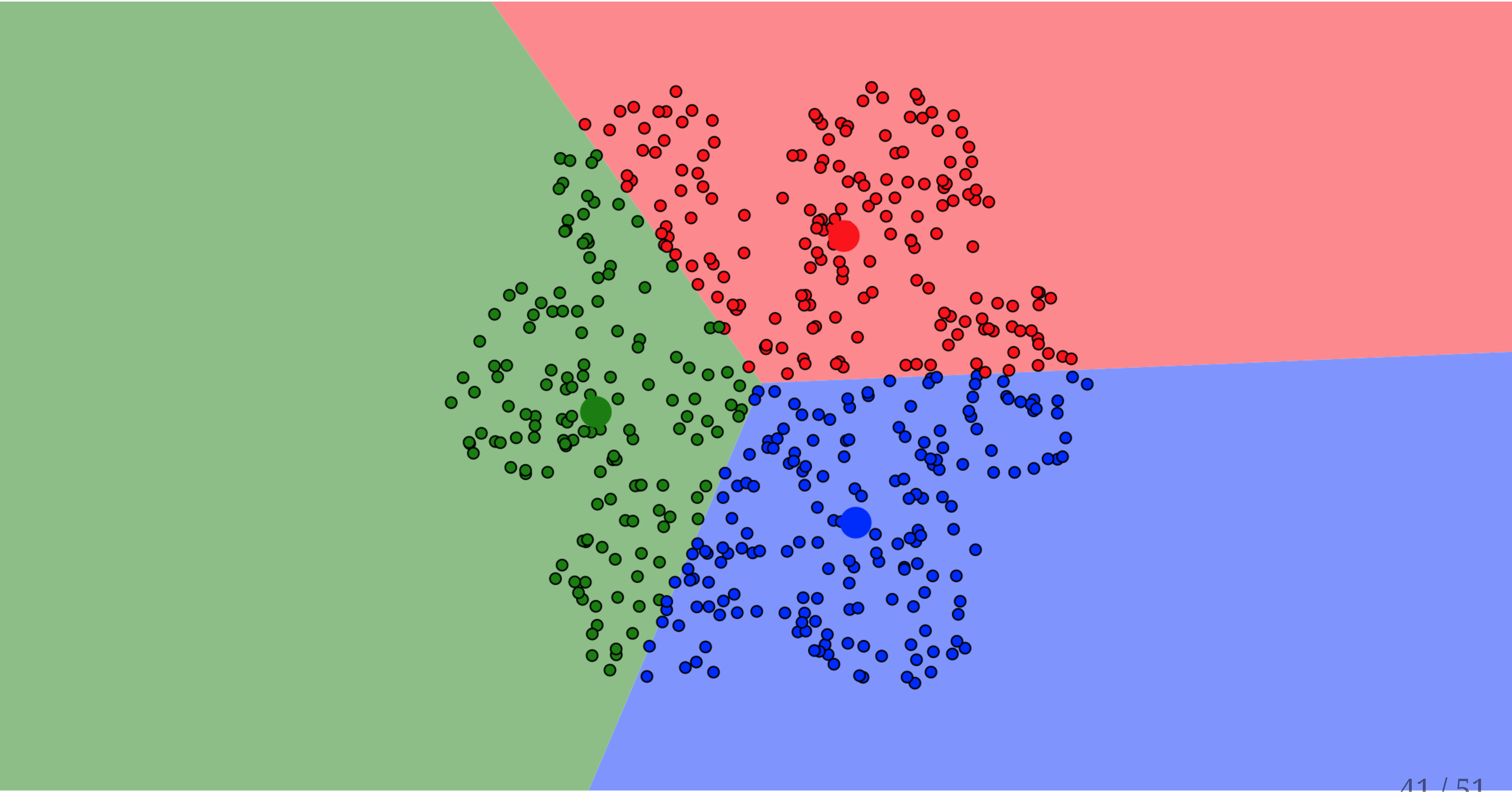












Now Try Yourself!

https://www.naftaliharris.com/blog/visualizing-k-means-clustering/

What is k-Means Clustering Doing?

- Denote C_k the k-th cluster.
- Each observation is assigned to exactly one cluster.
- Clusters are non-overlapping.
- A **good** clustering is one where *within-cluster variation* is as small as possible.
- Let's write $W(C_k)$ as some measure of **within cluster variation**.
- K-means tries to solve the problem of how to setup the clusters (i.e. how to assign observations to clusters), in order to...

What is k-Means Clustering Doing?

- Denote C_k the k-th cluster.
- Each observation is assigned to exactly one cluster.
- Clusters are non-overlapping.
- A **good** clustering is one where *within-cluster variation* is as small as possible.
- Let's write $W(C_k)$ as some measure of **within cluster variation**.
- K-means tries to solve the problem of how to setup the clusters (i.e. how to assign observations to clusters), in order to...
- ...minimize the total sum of $W(C_k)$:

$$\min_{C_1,\ldots,C_K} \left\{ \sum_{k=1}^{K} W(C_k) \right\}$$

- A common choice for $W(C_k)$ is the squared *Euclidean Distance*:

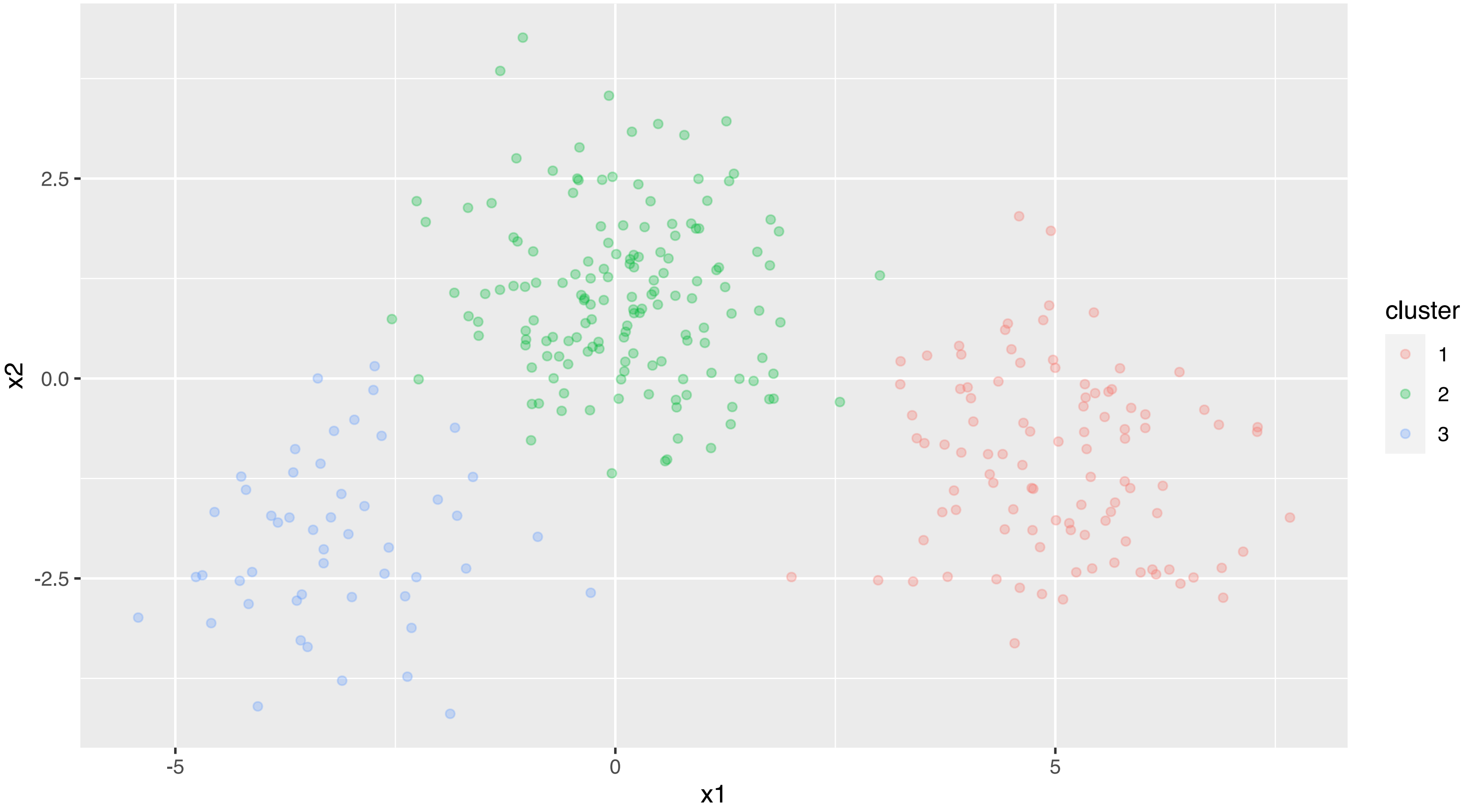
$$W(C_k) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2$$

where $|C_k|$ is the number of elements of C_k.

tidymodels k-means clustering

```r
library(tidymodels)

set.seed(27)

centers <- tibble(
  cluster = factor(1:3),
  num_points = c(100, 150, 50),  # number points in each cluster
  x1 = c(5, 0, -3),              # x1 coordinate of cluster center
  x2 = c(-1, 1, -2)              # x2 coordinate of cluster center
)

labelled_points <-
  centers %>%
  mutate(
    x1 = map2(num_points, x1, rnorm),
    x2 = map2(num_points, x2, rnorm)
  ) %>%
  select(-num_points) %>%
  unnest(cols = c(x1, x2))

ggplot(labelled_points, aes(x1, x2, color = cluster)) +
  geom_point(alpha = 0.3)
```

`tidymodels` k-means clustering



base R: kmeans

```
points <-
  labelled_points %>%
  select(-cluster)

kclust <- kmeans(points, centers = 3)
kclust
```

```
## K-means clustering with 3 clusters of sizes 148, 51, 101
##
## Cluster means:
##            x1        x2
## 1  0.08853475  1.045461
## 2 -3.14292460 -2.000043
## 3  5.00401249 -1.045811
##
## Clustering vector:
##   [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [38] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [75] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 1 1 1 1 1 1 1 1 1 1
## [112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [149] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [186] 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [223] 1 1 1 1 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 2 2 2 2 2 2 2 2
## [260] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [297] 2 2 2 2
##
## Within cluster sum of squares by cluster:
## [1] 298.9415 108.8112 243.2092
##  (between SS / total SS =  82.5 %)
```

How many clusters `k` to choose?

```
kclusts <-
  tibble(k = 1:9) %>%
  mutate(
    kclust = map(k, ~kmeans(points, .x)),
    tidied = map(kclust, tidy),
    glanced = map(kclust, glance),
    augmented = map(kclust, augment, points)
  )
kclusts
```

```
## # A tibble: 9 × 5
##       k kclust   tidied           glanced          augmented
##   <int> <list>   <list>           <list>           <list>
## 1     1 <kmeans> <tibble [1 × 5]> <tibble [1 × 4]> <tibble [300 × 3]>
## 2     2 <kmeans> <tibble [2 × 5]> <tibble [1 × 4]> <tibble [300 × 3]>
## 3     3 <kmeans> <tibble [3 × 5]> <tibble [1 × 4]> <tibble [300 × 3]>
## 4     4 <kmeans> <tibble [4 × 5]> <tibble [1 × 4]> <tibble [300 × 3]>
## 5     5 <kmeans> <tibble [5 × 5]> <tibble [1 × 4]> <tibble [300 × 3]>
## 6     6 <kmeans> <tibble [6 × 5]> <tibble [1 × 4]> <tibble [300 × 3]>
## 7     7 <kmeans> <tibble [7 × 5]> <tibble [1 × 4]> <tibble [300 × 3]>
## 8     8 <kmeans> <tibble [8 × 5]> <tibble [1 × 4]> <tibble [300 × 3]>
## 9     9 <kmeans> <tibble [9 × 5]> <tibble [1 × 4]> <tibble [300 × 3]>
```

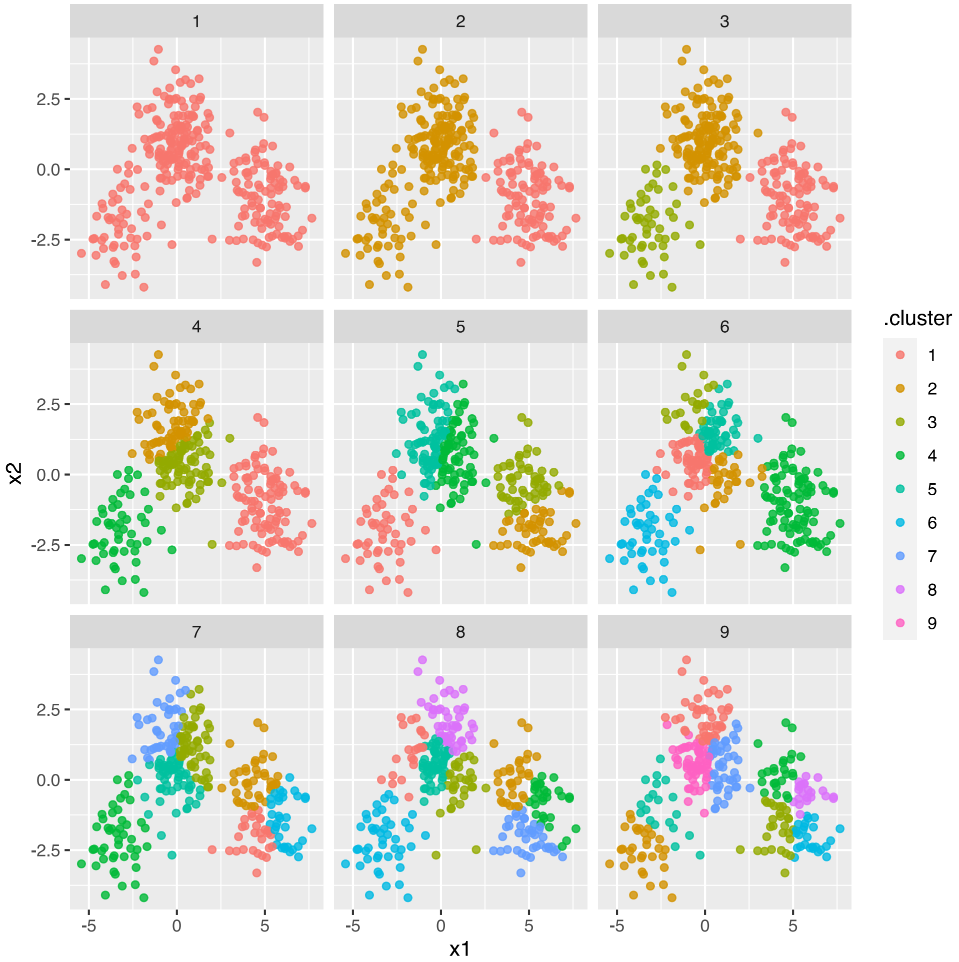
How many clusters `k` to choose?

- Teasing out different datasets for plotting
- notice the `unnest` calls are useful for `list` columns

```
clusters <-
  kclusts %>%
  unnest(cols = c(tidied))

assignments <-
  kclusts %>%
  unnest(cols = c(augmented))

clusterings <-
  kclusts %>%
  unnest(cols = c(glanced))
```

How many clusters `k` to choose?

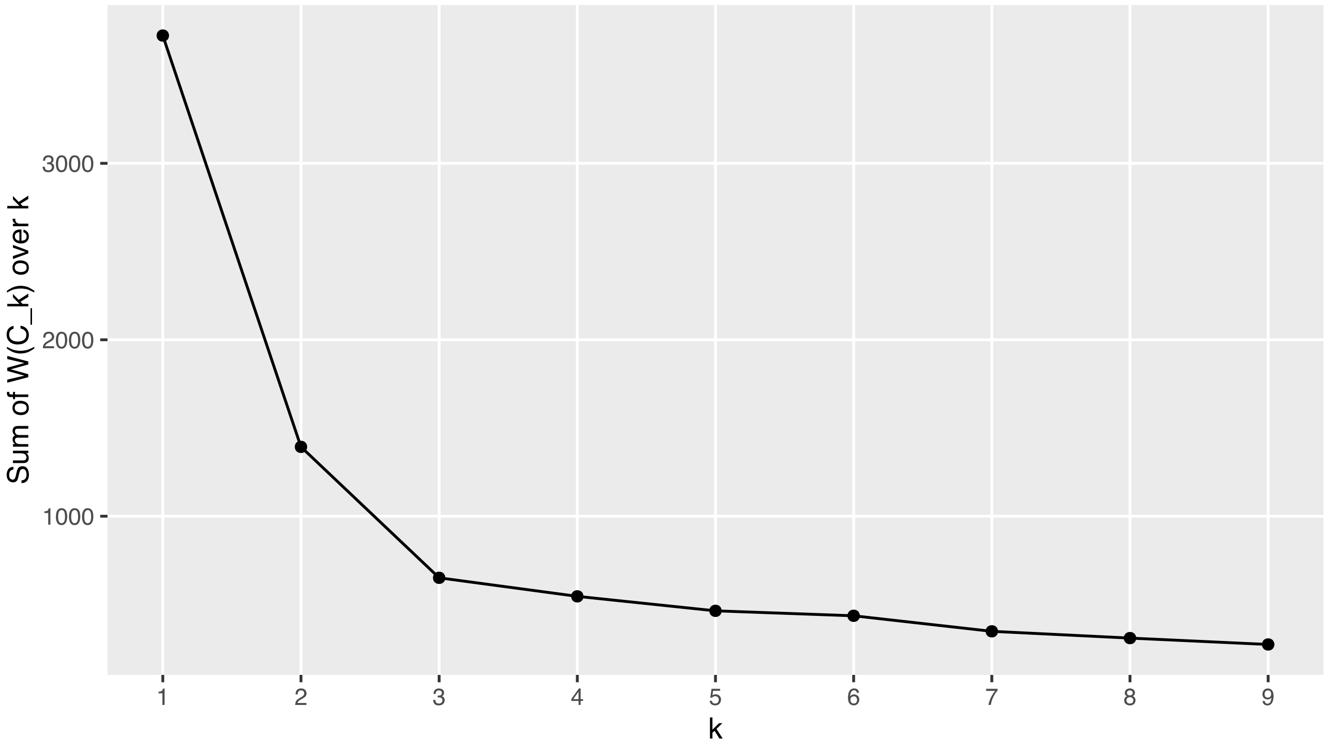
```r
p1 <-
  ggplot(assignments, aes(x = x1, y = x2)) +
  geom_point(aes(color = .cluster), alpha = 0.8) +
  facet_wrap(~ k)
```



How many clusters `k` to choose? The *Elbow* Method

```r
# the Elbow plot
ggplot(clusterings, aes(k, tot.withinss)) +
  geom_line() + ylab("Sum of W(C_k) over k") +
  geom_point()
```

- Look for the **Elbow**!
- Here at `k = 3` the reduction in $\sum_k W(C_k)$ slows down a lot.
- More flexibility (more clusters) **overfits** the data beyond a certain point (the *elbow*)



SciencesPo
DEPARTMENT OF ECONOMICS

END

	bluebery.planterose@sciencespo.fr
	Original Slides from Florian Oswald
	Book
	@ScPoEcon
	@ScPoEcon