Data Science and Data Analytics

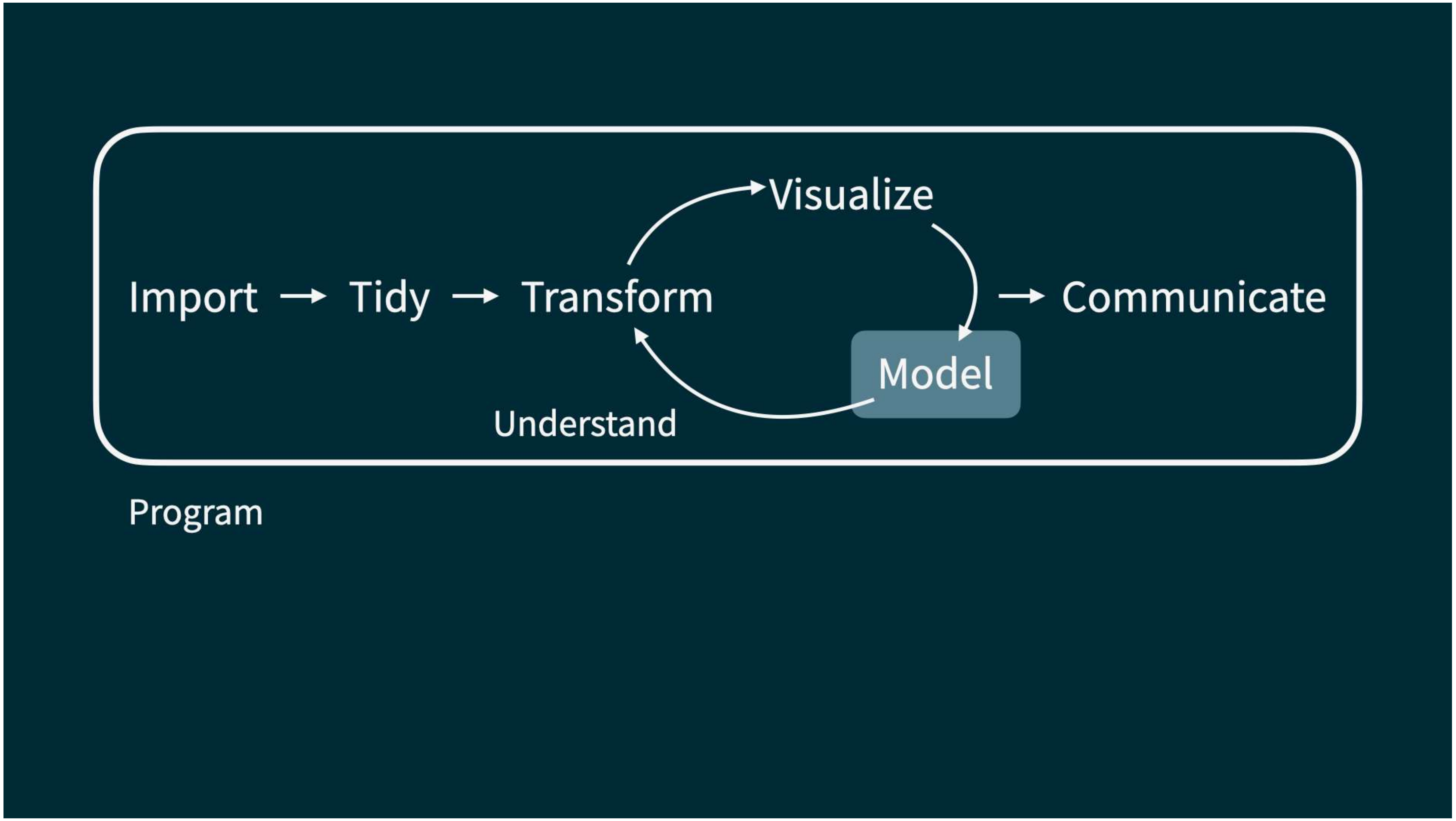
The Data Science Workflow III – Model

Julian Amon, PhD
Charlotte Fresenius Privatuniversität

April 30, 2025

Introduction to Machine Learning

The Data Science workflow – Model

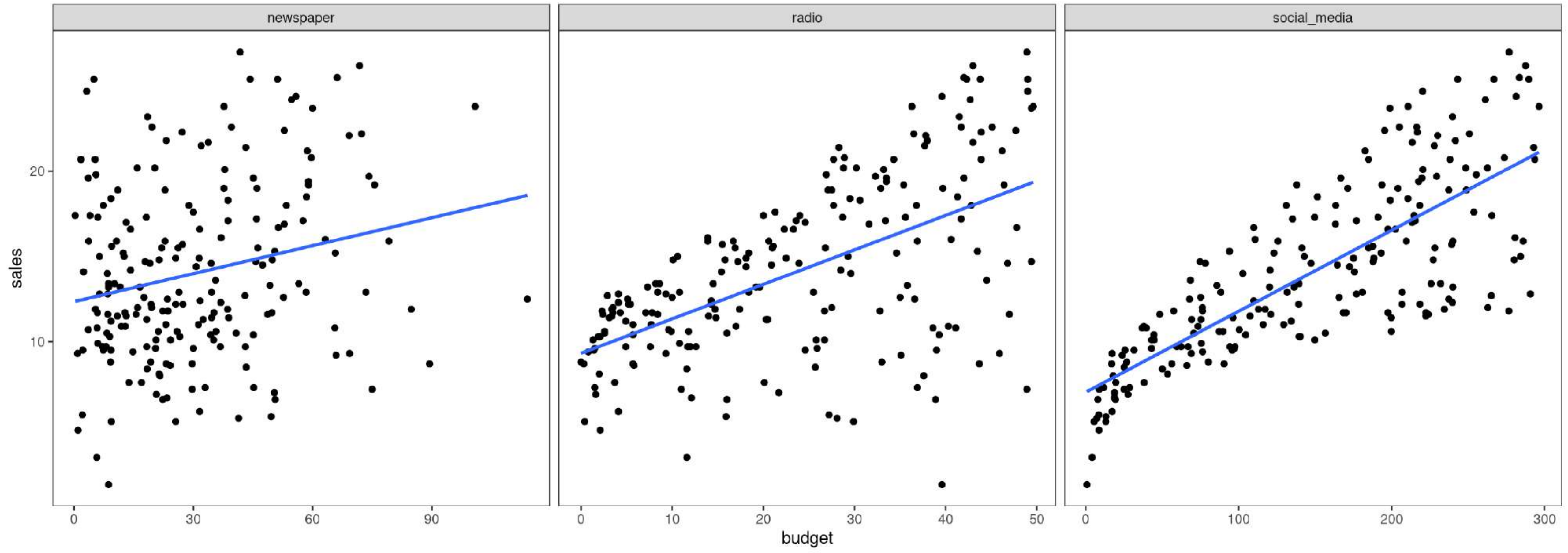


Time to put the data to use...

- Having imported, tidied, transformed and visualized our data, now it is time to actually put this data to good use...
- The goal of the **modeling** stage of the data science workflow is therefore to **detect patterns** and **extract information** from data.
- The tools and processes we use for this purpose stem from a field called **machine learning** (or **statistical learning**).
- To motivate our study of this field, let's begin with a simple example:
  - We are statistical consultants hired to investigate the association between **advertising** and **sales** of a particular product.
  - We are given a data set consisting of the `sales` of that product in 200 different markets, along with advertising budgets for three different media: `social_media`, `radio` and `newspaper`.

Machine learning – A motivating example

- Our client cannot directly increase sales of course... But, they can control advertising expenses in each of the three media.
- Our goal is therefore to develop an accurate **model** that can be used to predict sales on the basis of the three media budgets.



Machine learning – A motivating example

- In this setting, the advertising budgets are **input variables**:
  - Input variables are also called **predictors** or **features**.
  - We denote them by X with a subscript to distinguish them.
  - In the example, X_1 is the `social_media` budget, X_2 the `radio` budget and X_3 the `newspaper` budget.
- On the other hand, `sales` is an **output variable**:
  - The output variable is also called the **response (variable)** or **label**.
  - We denote it by Y.

Machine learning – A motivating example

- Using this notation more generally, suppose that we observe a **quantitative** response Y and p different predictors $X_1, X_2, \ldots, X_p$.
- We assume that there is some relationship between Y and $X = (X_1, X_2, \ldots, X_p)$, which can be written in the very general form

$$Y = f(X) + \epsilon,$$

  where f is some fixed but **unknown** function of $X_1, \ldots, X_p$, and ϵ is a random **error term**, which is independent of X and has mean zero.
- We can think of f as the **systematic information** that X provides about Y.
- A large part of the **machine learning** literature revolves around different approaches for estimating f.

Why estimate f? – Prediction

- Estimate f for **prediction** purposes: For inputs X, we can predict Y using

$$\hat{Y} = \hat{f}(X),$$

  where $\hat{f}$ represents our estimate for f and $\hat{Y}$ represents the resulting prediction for Y.

- The accuracy of $\hat{Y}$ as a prediction for Y depends on two quantities:
  - **Reducible error**: $\hat{f}$ will not be a perfect estimate for f and this inaccuracy will introduce some error. However, it is **reducible** by using more appropriate learning techniques or collecting more data.
  - **Irreducible error**: Even if we knew f, our prediction would have some error in it. This is because of the random error term ϵ affecting Y.

Why estimate f? – Inference

- We might also estimate f for **inference** purposes, i.e. to understand the association between Y and $X_1, \ldots, X_p$.
- In this context, one may be interested in answering the following questions:
  - *Which predictors are associated with the response?* Often only a small subset of the available predictors are important in explaining Y.
  - *What is the relationship between the response and each predictor?* Is it positive, negative, linear, non-linear, …?
- In the **advertising** example, specific inference-related questions are:
  - Which media are associated with sales?
  - Which media generate the biggest boost in sales?
  - How large of an increase in sales is associated with a given increase in social media advertising?

How do we estimate f?

- We assume that we have observed a set of n different data points. Let x_{ij} represent the value of the jth predictor for observation i, where $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, p$. Correspondingly, let y_i represent the response variable for observation i.
- With this, our data set consists of

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\},$$

where $x_i = (x_{i1}, \ldots, x_{ip})$. These observations are called the **training data** because we will use them to **train** our method on how to estimate f.

- Our goal with this is to estimate the unknown function f, i.e. find a function $\hat{f}$ such that $Y \approx \hat{f}(X)$ for any observation (X, Y).

Parametric methods for estimating f

- Broadly speaking, most statistical learning methods for this task can be characterized as either **parametric** or **non-parametric**.
- **Parametric methods** involve a two-step model-based approach:
  1. Make an assumption about the function form of f. For example, one very simple assumption is that f is linear in X:

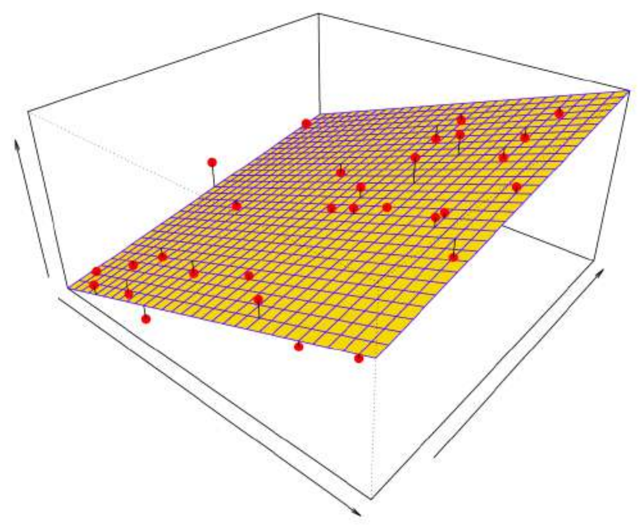
$$f(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p$$

  2. Find and apply a procedure that uses training data to **fit** or **train** the model. In case of the linear model, that means finding values of parameters $\beta_0, \beta_1, \ldots, \beta_p$ such that:

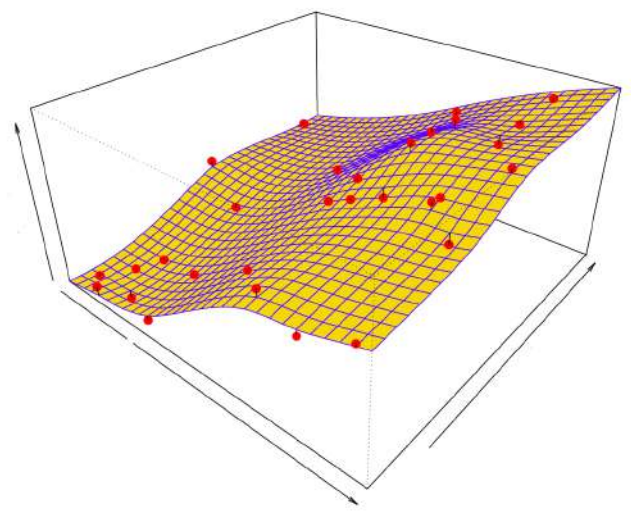
$$Y \approx \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p$$

Non-parametric methods for estimating f

- **Non-parametric** methods do not make explicit assumptions about the functional form of f. Instead, they seek an estimate of f that gets as close to the training data points as possible without being too wiggly.
- **Advantage**: can accommodate a much wider range of possible shapes for f compared to parametric approaches.
- **Disadvantage**: do not reduce the problem of estimating f to a small number of parameters $\longrightarrow$ a very large number of observations is needed.



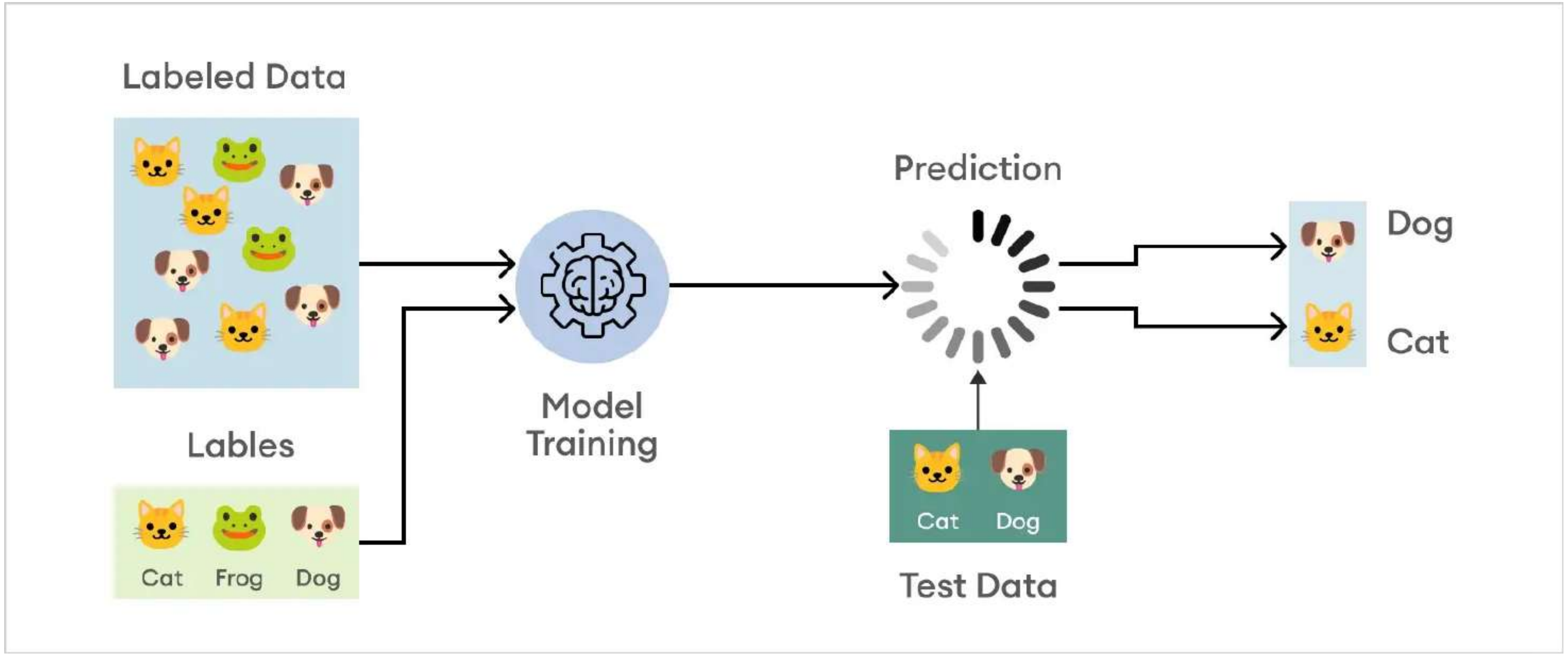
Parametric fit



Non-parametric fit

Estimating f as part of supervised learning

- Fitting a model for the purposes of prediction or inference based on labeled training data (x_i, y_i) characterizes a class of machine learning problems called **supervised learning**.
- For these problems, there is an associated output y_i for each input x_i. These **labels** act as a “supervisor” guiding the learning process, similar to a student learning under a teacher’s guidance.



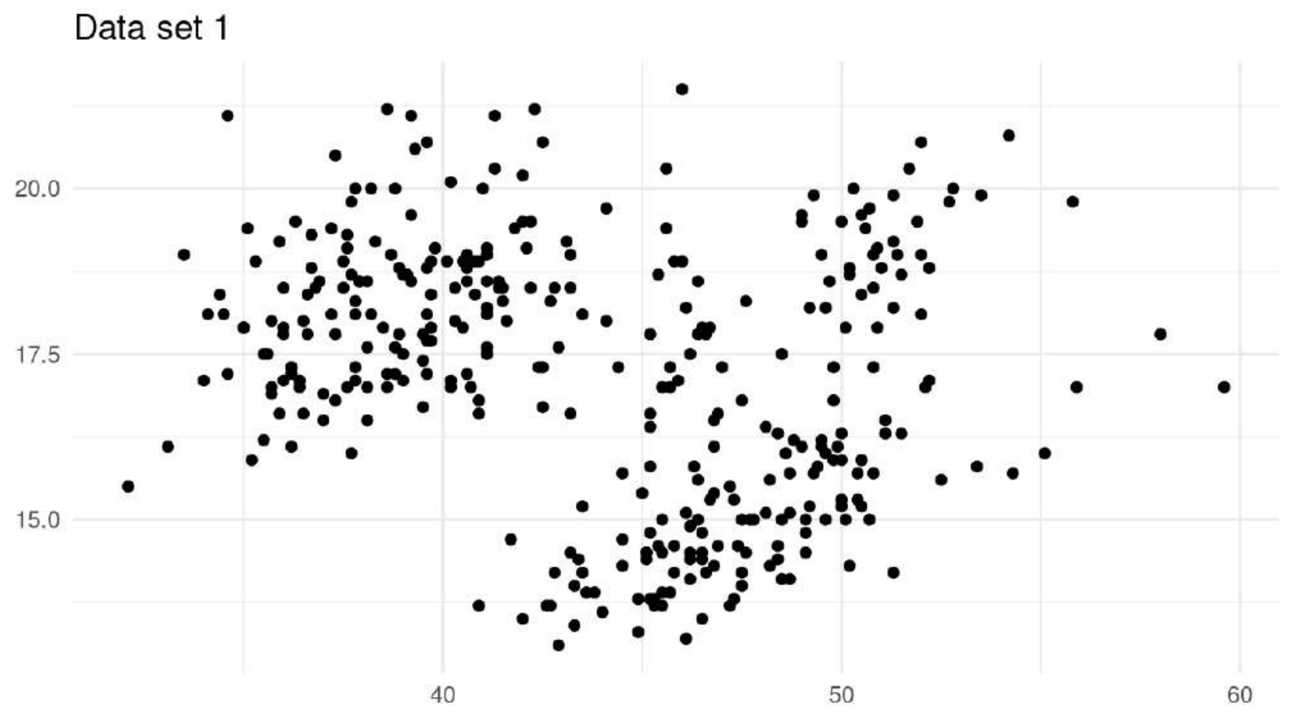
Regression vs. classification problems

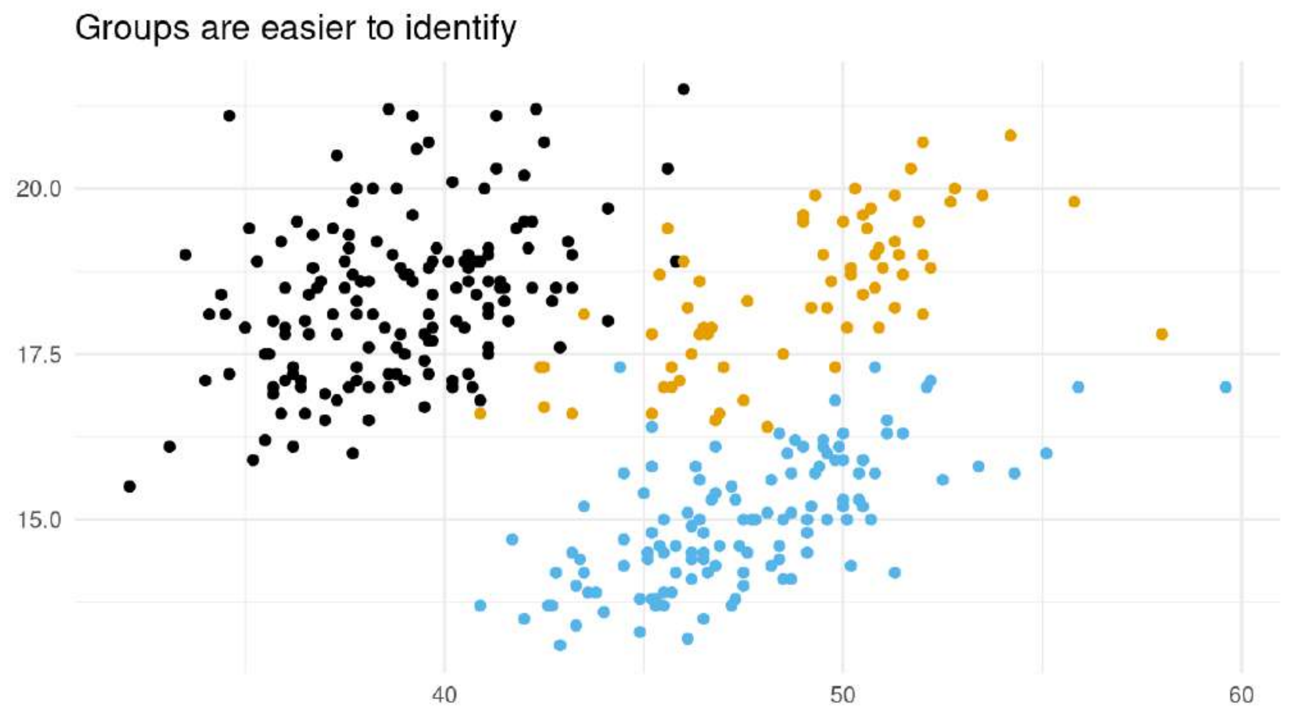
- Supervised learning problems can be further sub-categorized into **regression** and **classification** problems depending on the type of response variable:
  - Problems with a **quantitative response** are called **regression problems**, e.g. modeling sales on the basis of the three media budgets.
  - Problems with a **categorical response** are called **classification problems**, e.g. modeling consumer credit default based on economic variables. Depending on the number of categories, these problems are either **binary** (e.g. yes/no) or **multiclass** (e.g. Adelie/Chinstrap/Gentoo).
- While we select supervised learning methods on the basis of whether the response is quantitative or categorical, the type of the **predictors** is less important. Typically, methods can be applied with both quantitative and categorical predictors.

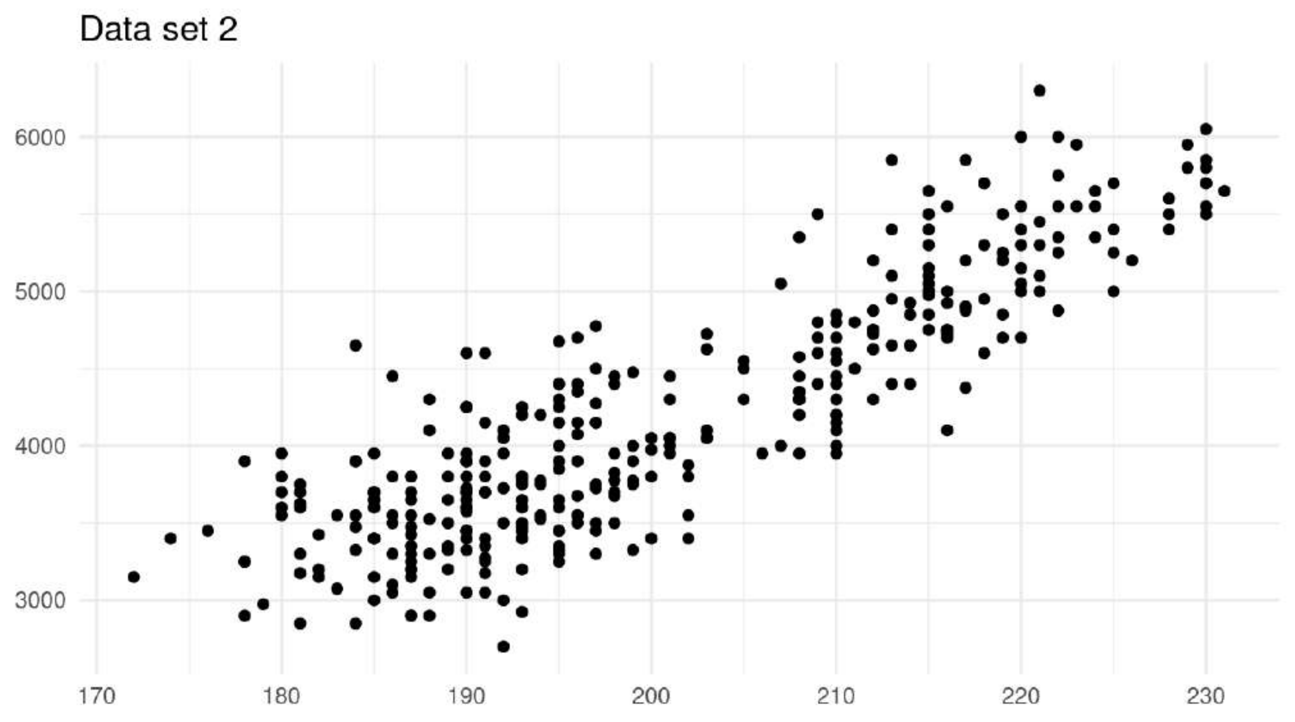
Unsupervised learning

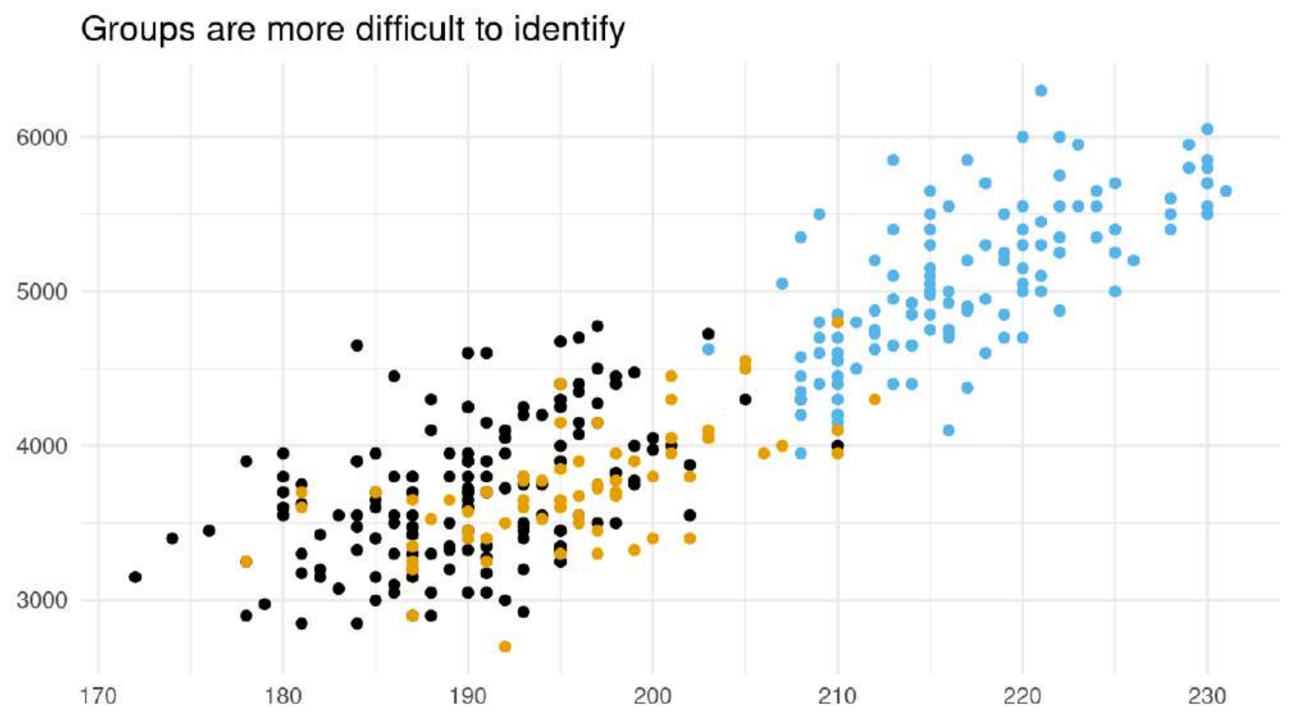
- By contrast, **unsupervised learning** describes the somewhat more challenging situation in which we only observe a vector of measurements x_i, but no associated response y_i.
- We cannot fit a model because there is no response variable to predict.
- What sort of analysis can we do then?
  - **Clustering**: Assigning observations into relatively distinct groups, e.g. market segmentation: grouping customers into clusters based on socio-demographic characteristics.
  - **Outlier detection**: Identifying data points that significantly deviate from the norm within a data set, potentially revealing errors, unusual events, or important discoveries.
  - …

Unsupervised learning – Clustering

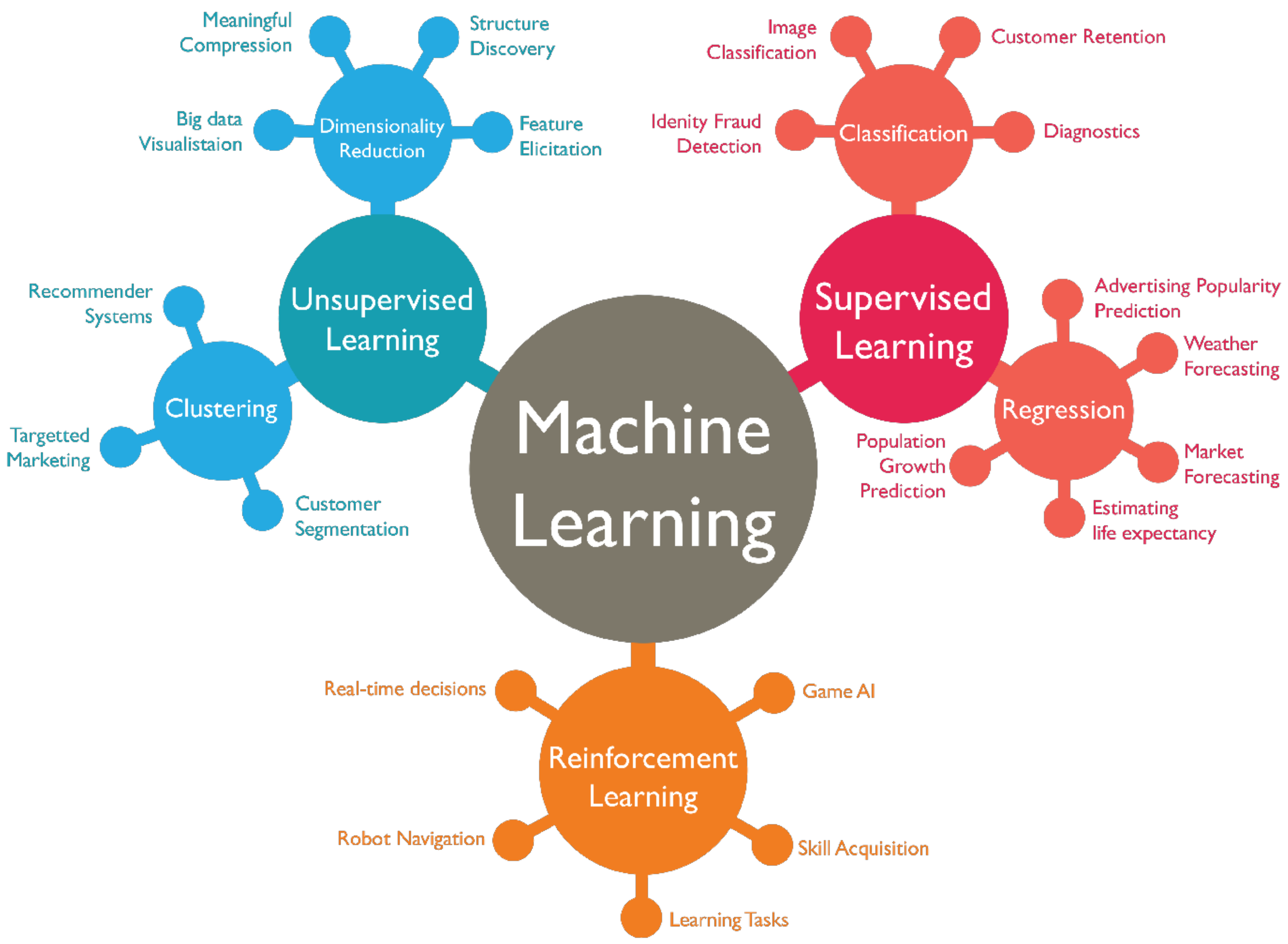








Machine learning – Overview with examples



Assessing model accuracy

Regression – Measuring the quality of fit

- In order to evaluate the performance of a statistical learning method on a given data set, we need some way to quantify how far the **predicted** response values are from the **true** response values.
- For **regression** problems, the most commonly-used measure is the **mean squared error (MSE)**:

$$\mathrm{MSE}_{\mathrm{train}} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}\,(x_i))^2$$

  - The MSE will be **small** if the predicted responses are very close to the true responses.
  - The MSE will be **large** if the predicted responses and the true responses differ substantially.

Training vs. test MSE

- As we use our n training observations to compute the MSE on the previous slide, we call this the **training MSE**.
- However, in general, we do not really care about how well the method works on the training data. Rather, we are interested in the prediction accuracy on previously unseen **test data**.
- Consider the following examples:
  - When fitting a model to predict a stock price, we do not really care about how well it does on historical stock prices. Instead, we care about how well it predicts **tomorrow's** stock price.
  - When estimating a model to predict whether someone has diabetes based on clinical measurements (e.g. weight, blood pressure, age, …), we want to use this model to predict diabetes risk for **future** patients, not for the ones used for training.

Training vs. test MSE

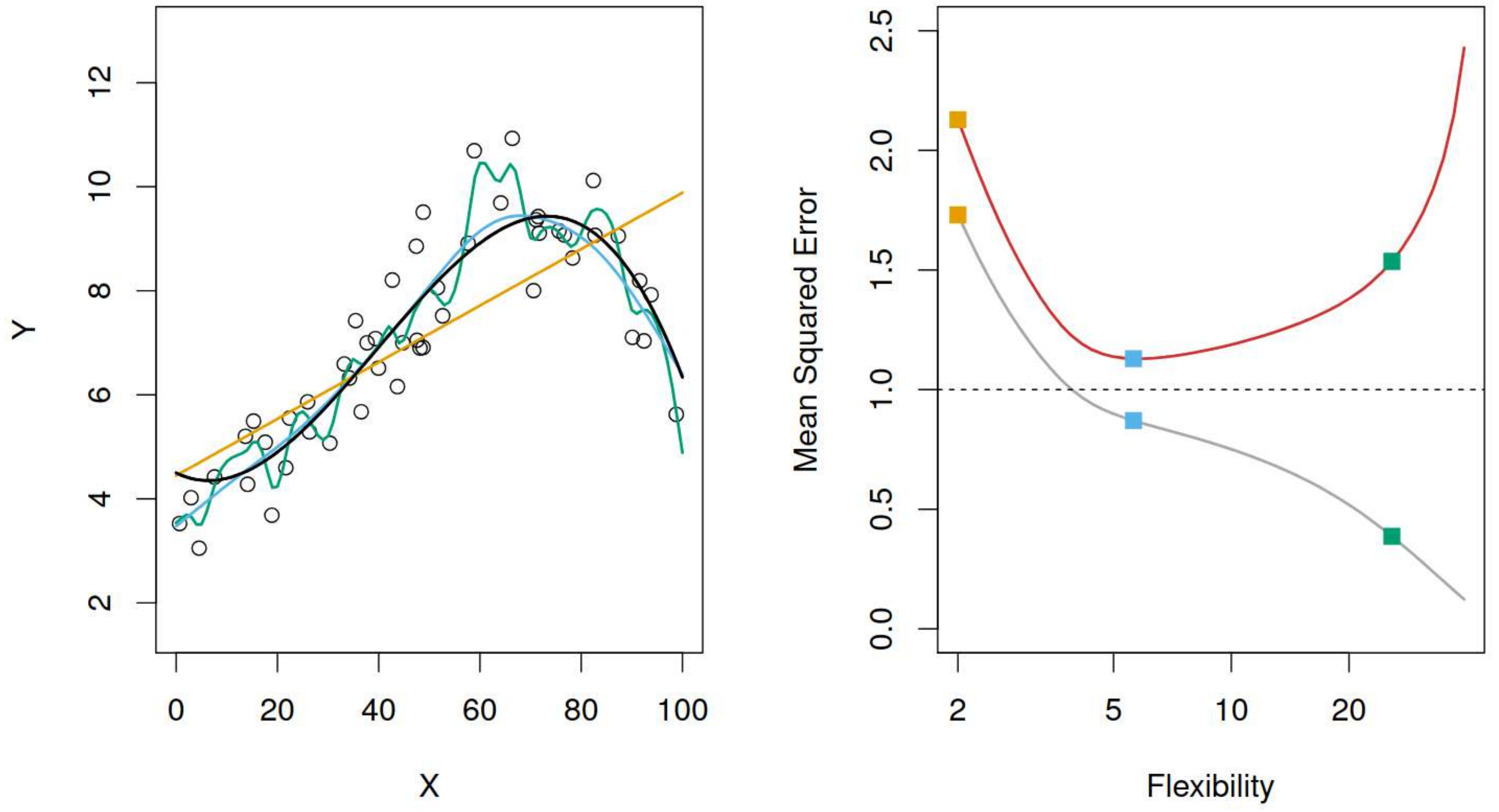
- Mathematically speaking, suppose we have a set of n' previously unseen test observations (x_i', y_i') that were **not used to train the machine learning method**. Then we can compute the **test MSE** as:

$$\mathrm{MSE}_{\mathrm{test}} = \frac{1}{n'} \sum_{i=1}^{n'} (y_i' - \hat{f}(x_i'))^2$$

- We can use this quantity to compare different methods: we want to select the method for which it is **as small as possible**.
- Now, how do we go about minimizing **test MSE** on unseen data? Can't we just select the method that minimizes **training MSE**? Turns out this is not a good idea at all…

Training vs. test MSE

- Suppose we observe the data points in the left panel, which come from the true f shown in black. We estimate it using three different methods with increasing levels of flexibility (low: orange, medium: blue, high: green).
- The right panel shows the corresponding training and test MSEs.



Training vs. test MSE

- Clearly, the best fit is achieved by the blue curve, the orange (linear) one is too inflexible, the green one is too flexible (too wiggly).
- The comparison of train and test MSE as a function of flexibility reveals an interesting pattern:
  - The **training MSE** is monotonically decreasing: the more flexible the method, the better we fit our training data.
  - However, the all-important **test MSE** exhibits a U-shape: it initially decreases with increasing flexibility, but then levels off and eventually increases again.
- This is a fundamental property of statistical learning that holds regardless of data set and statistical method:

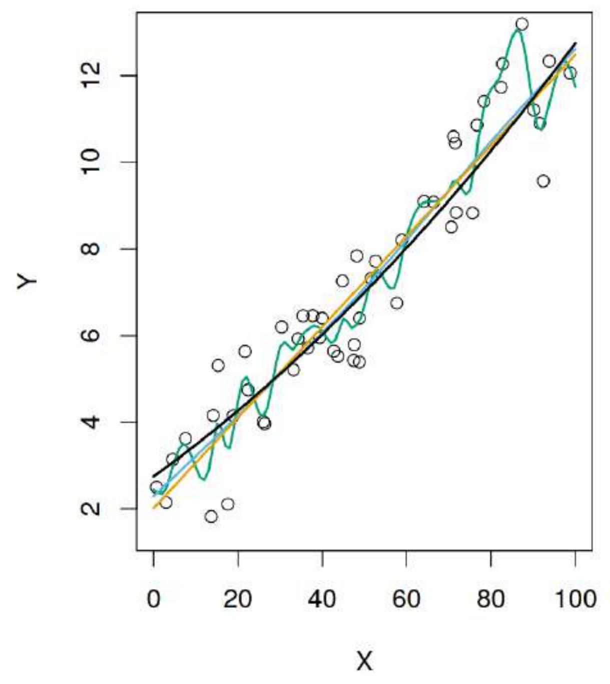
  As model flexibility increases, the training MSE will decrease, but the test MSE may not.

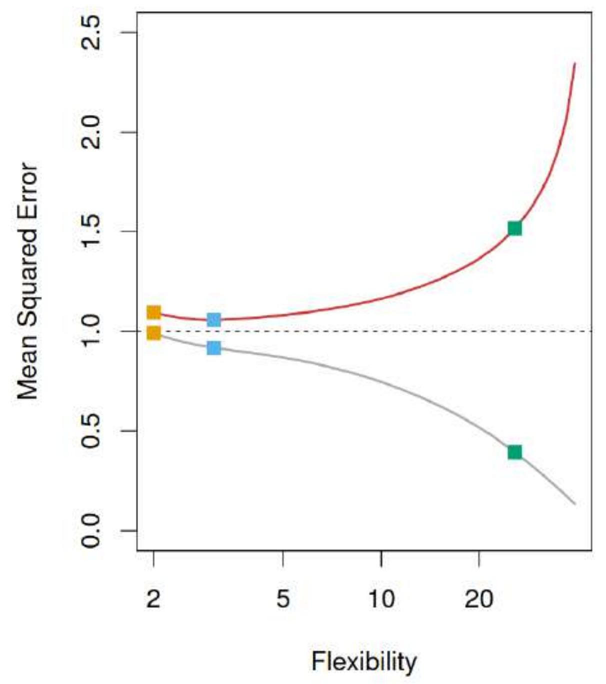
Over- and underfitting

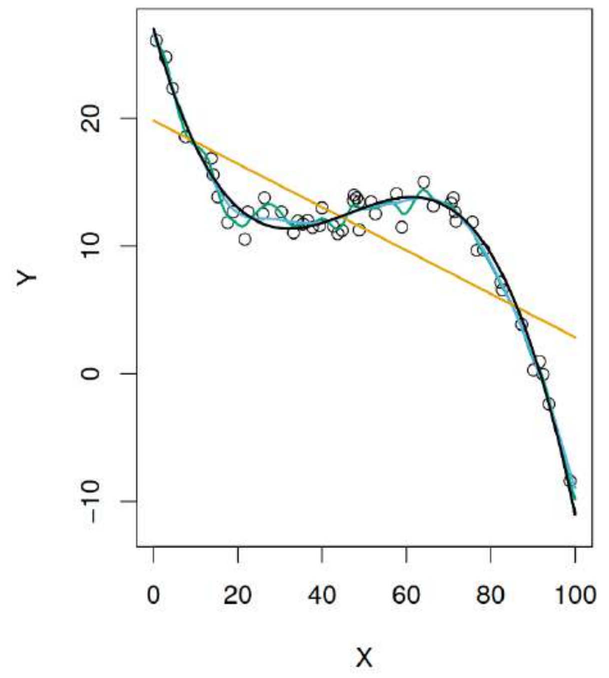
- The green fit has small training MSE, but high test MSE. We call this **overfitting**:
  - With its high flexibility, this method is fitting patterns in the training data that are just caused by **random chance** rather than by true properties of f.
  - The **test MSE** will then be very large because the supposed patterns that the method found in the training data simply do not exist in the test data.
- Conversely, the orange (linear) fit has large training MSE and large test MSE. We call this **underfitting**:
  - With its low flexibility, namely its assumption of linearity, this method is not able to fit the non-linearity of the true f well, even on the training data.
  - By increasing flexibility from here, we are able to use more information contained in X to improve the fit.
- Consequently, the blue fit with medium-level flexibility is close to optimal.

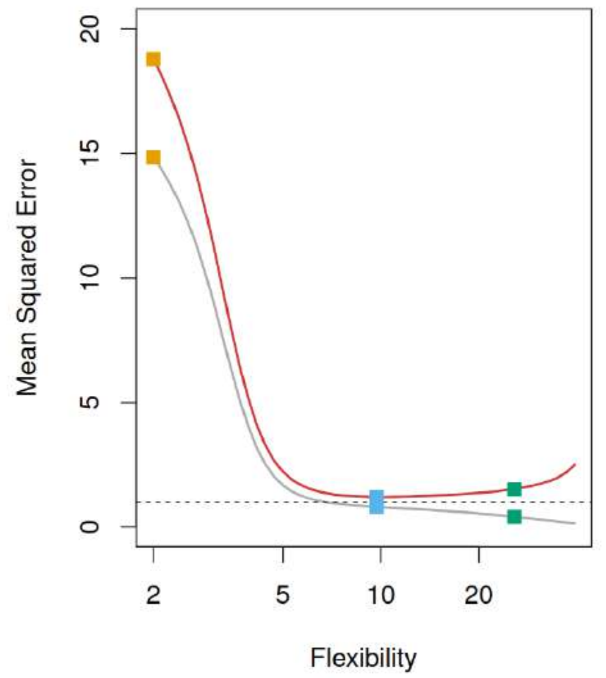
Over- and underfitting

Let’s see two more examples.









- Here, the true f is approximately linear.
- The U-shape in test MSE is still there, but because the truth is close to linear, the test MSE only decreases slightly before increasing again.

- Here, the true f is highly non-linear.
- The MSE curves still exhibit the same general patterns, but now there is a rapid decrease in both curves before the test MSE starts to increase slowly.

The bias-variance trade-off

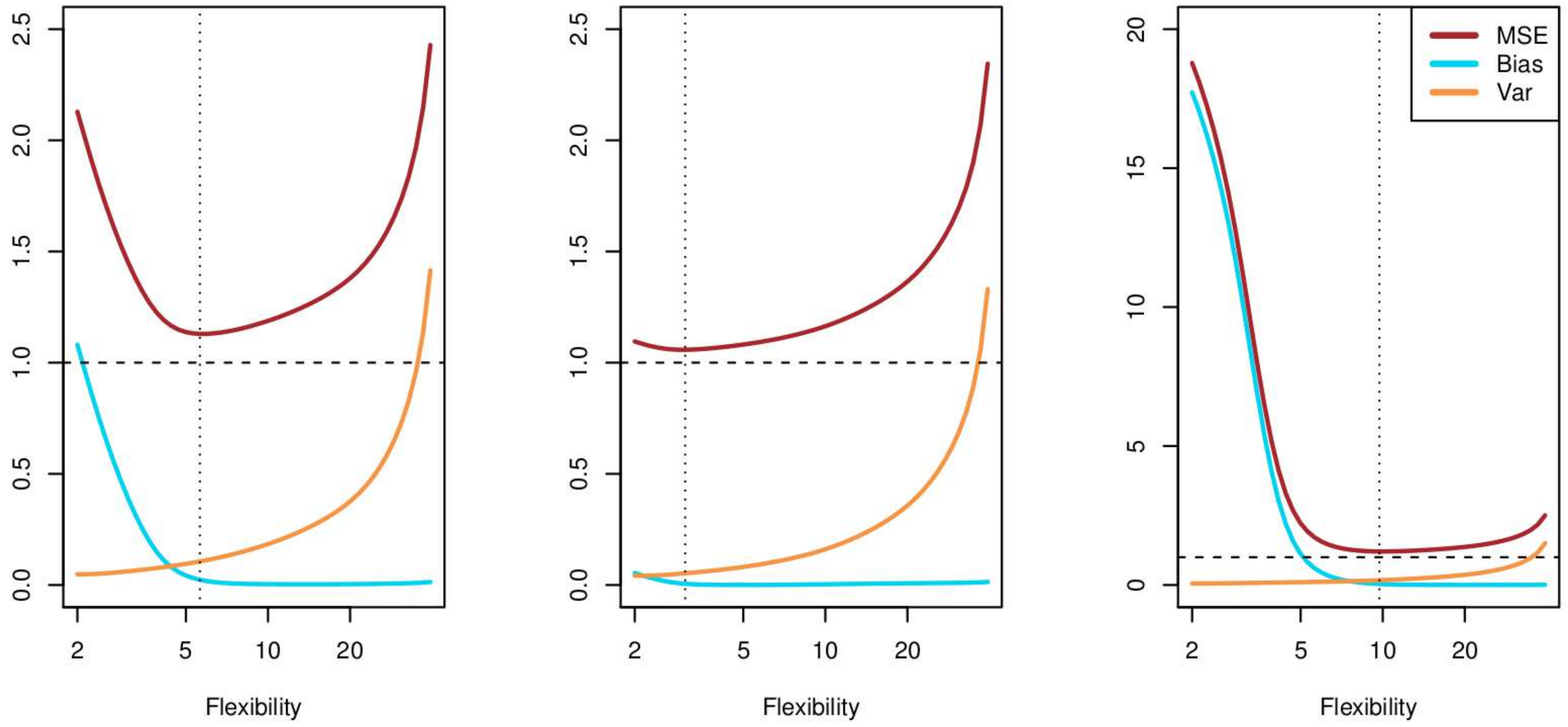
- The U-shape observed in test MSE curves turns out to be the result of two competing properties of statistical learning methods:

$$\text{Expected test MSE} = \text{Var}(\hat{f}\,) + \text{Bias}(\hat{f}\,)^2 + \text{Var}(\epsilon)$$

- So we want methods that simultaneously achieve **low variance** and **low bias**. What is meant by these terms in this context?
  - **Variance** refers to the amount by which $\hat{f}$ would change if we estimated it using a different training data set. In general, more flexible statistical methods have higher variance.
  - **Bias** refers to the error that is introduced by approximating a complicated real-life problem by a much simpler model. In general, more flexible statistical methods result in less bias.

The bias-variance trade-off

Below, we decompose the three MSE curves from earlier into their three constituent parts: the variance of $\hat{f}$, its squared bias and $\mathrm{Var}(\epsilon)$, the irreducible error (represented by the dashed line):



The bias-variance trade-off

- These plots reveal the following insights:
  - As the method's flexibility increases, the **variance increases** and the **bias decreases**.
  - The relative rate of change of these two quantities determine whether the test MSE (as the sum of the two plus $\mathrm{Var}(\epsilon)$) increases or decreases.
  - This is what we call the **bias-variance trade-off**: bias and variance need to be balanced optimally to minimize test MSE.
  - This trade-off is different for every data set and method applied to it.
- Due to the particularities of different data sets, there is **no free lunch** in statistical learning: no one method optimally trades off bias and variance on all possible data sets.

Classification – Measuring the quality of fit

- With MSE, our discussion of model performance has so far focused on the **regression** setting. But many concepts we encountered easily transfer over to the classification setting.
- One thing that does change is the metric for measuring the quality of fit. With $y_1, \ldots, y_n$ now **categorical**, the most common approach for quantifying model accuracy is the training **error rate** ER:

$$ER_{\text{train}} = \frac{1}{n} \sum_{i=1}^{n} I(y_i \neq \hat{y}_i),$$

where $\hat{y}_i$ is the predicted class label for the ith observation using $\hat{f}$ and $I(y_i \neq \hat{y}_i)$ is an indicator variable that equals 1 if $\hat{y}_i \neq y_i$ and 0 if $\hat{y}_i = y_i$.

Training vs. test error rate

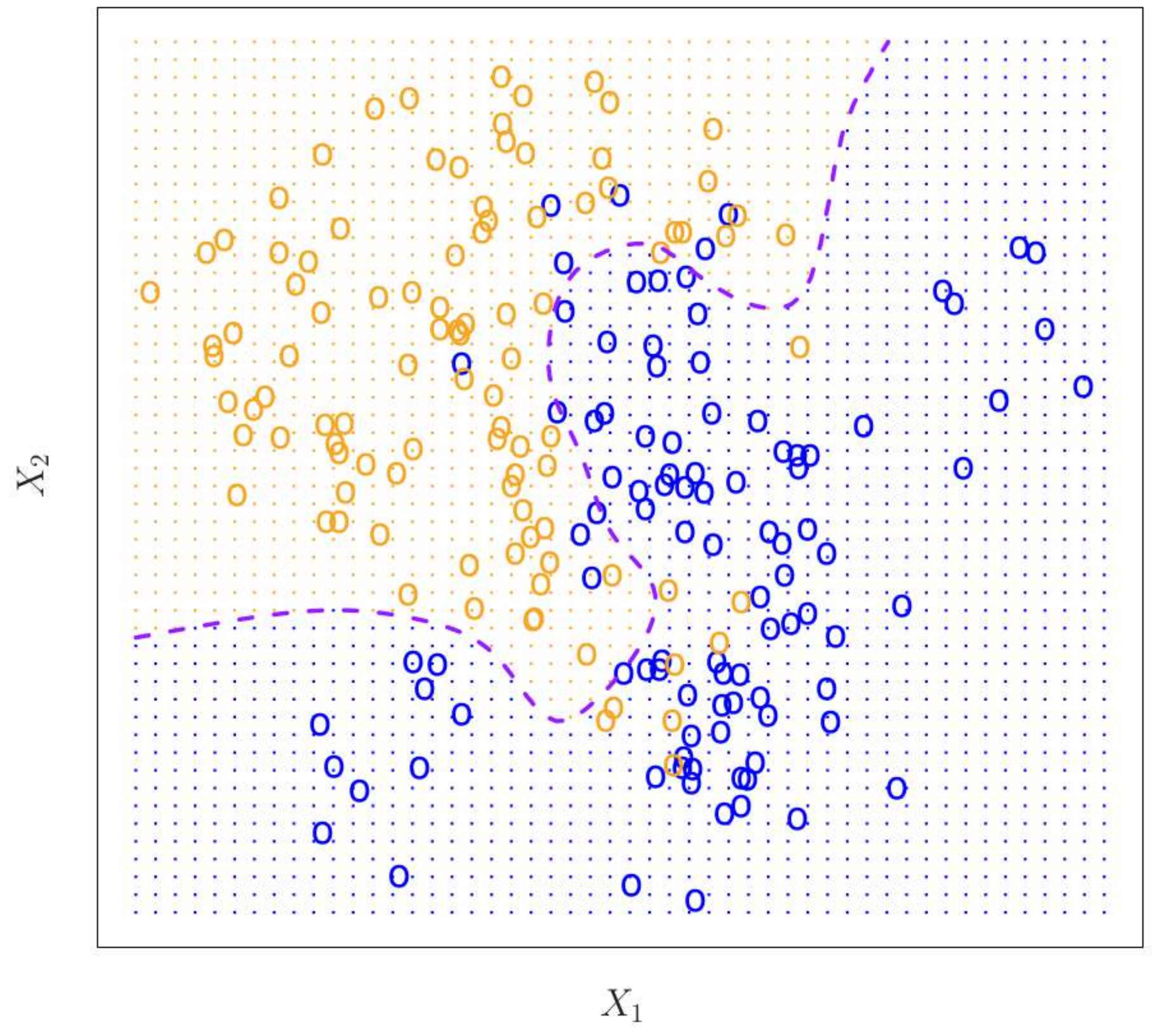
- As in the regression context, we are mostly interested in the prediction accuracy on previously unseen **test data**.
- Therefore, if we have a test set of n' observations (x_i', y_i') that were not used to train our method, we can compute the **test error rate** as:

$$ER_{\text{test}} = \frac{1}{n'} \sum_{i=1}^{n} I(y_i' \neq \hat{y}_i')$$

- Again, we can use this quantity to compare different **classification** methods: we want to select the method for which the test error rate is **as small as possible**.
- Let’s consider an example…

Classification example

Suppose we wanted to predict whether an individual will default on his or her credit card payment on the basis of annual income (X_1) and monthly credit card balance (X_2). Then we might use the x and the y axis to display the two predictors and colour to indicate default (orange) or no default (blue), like so:

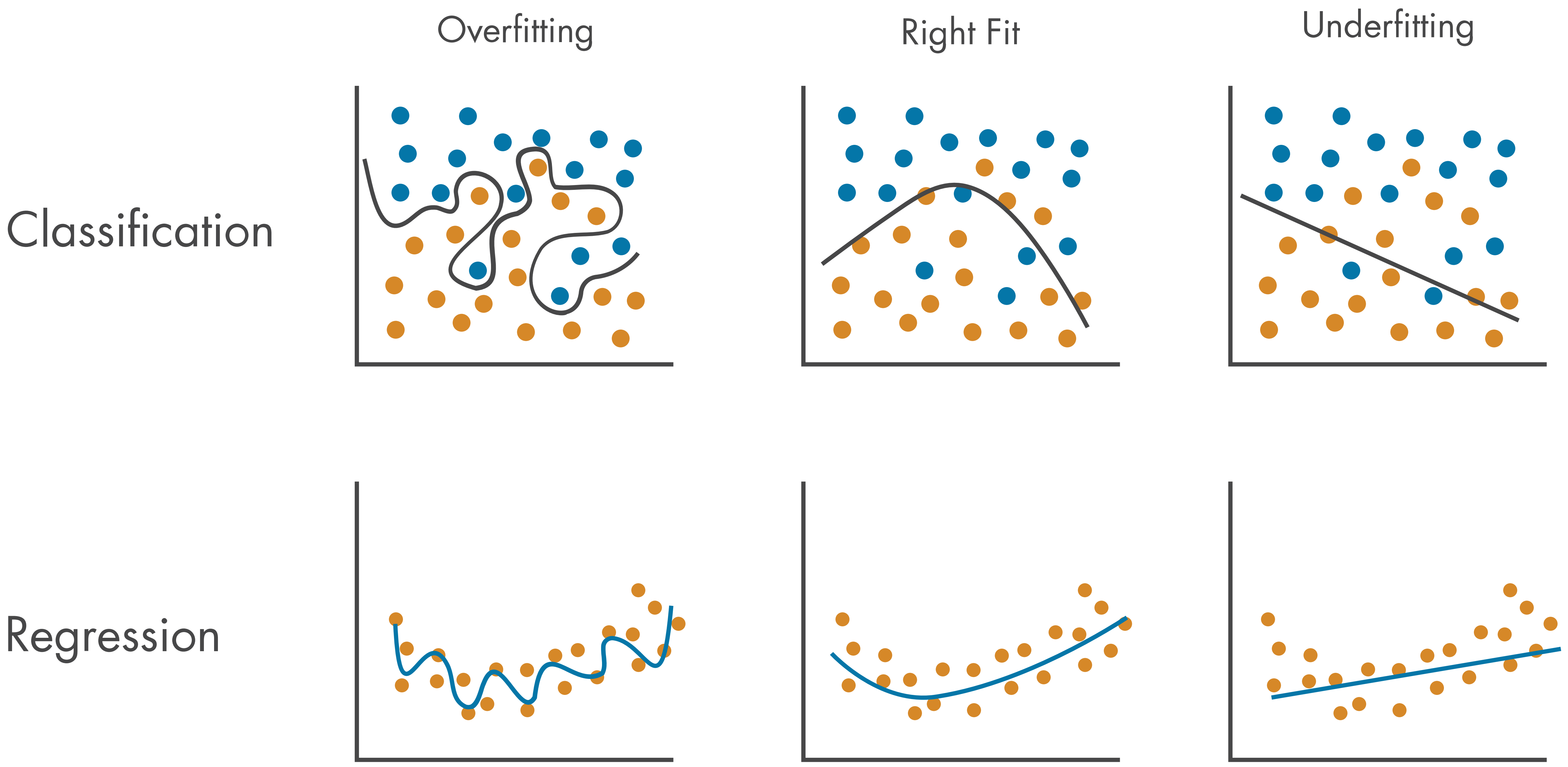


- Orange and blue circles indicate training observations.
- For each value of X_1 and X_2, there is a different probability of the response being orange or blue.
- The orange shaded region reflects the set of points for which $\Pr(Y = \text{orange}|X) > 0.5$.
- The purple line indicates the true **decision boundary**.

Classification example

- The displayed data set is based on simulation, so that we know the **true** decision boundary.
- Typically, this is not the case of course and we only observe training data points. The goal then is to come as close as possible to the (unknown) true decision boundary.
- Note two parallels to the regression setting:
  - Even if we knew the true decision boundary, we would commit errors. This is because there is some **irreducible** error.
  - In choosing a classification method, we also have to **trade off bias and variance** to avoid over-/underfitting.

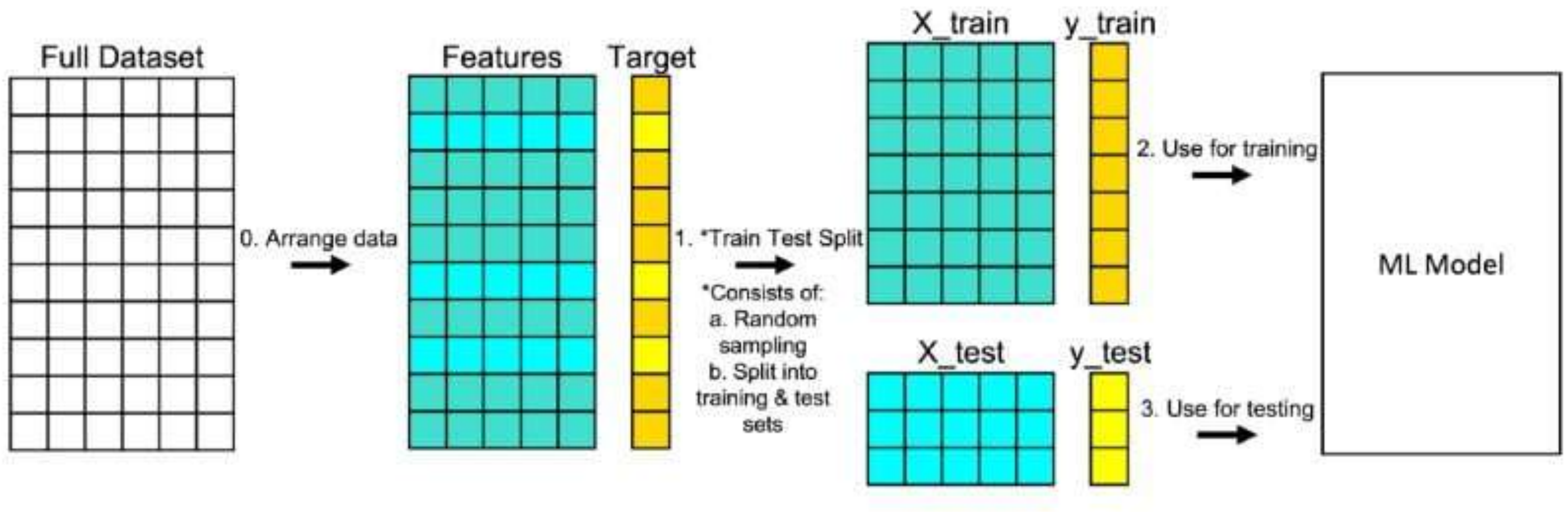
Over- and underfitting – Regression vs. classification



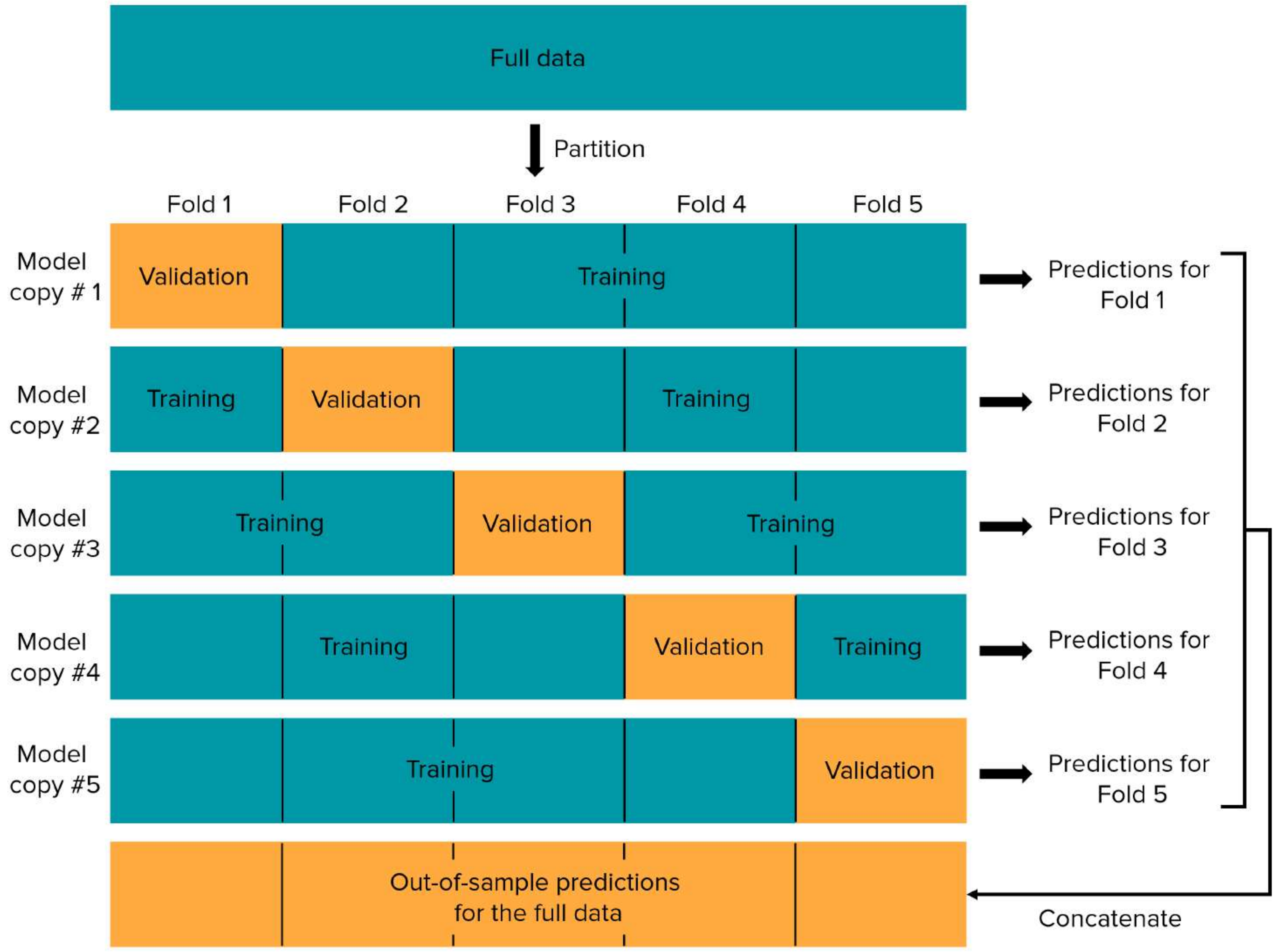
How to get test data

- To assess when we are likely to overfit, we need some way of measuring the MSE (regression) or ER (classification) on **unseen test data**. How do we do that?
- **Answer**: use the training data in a smart way!
- There are many different approaches, two prominent ones are:
    - **Train/test split**: split the data randomly into a train and test sample and compute error on the test sample.
    - **K-fold cross-validation**: split the entire data randomly into K folds. Fit the model using the $K-1$ folds and evaluate the model using the remaining fold. Repeat this process until every fold has served as the test set.

How to get test data – Train/test split



How to get test data – K-fold cross validation



Linear Regression

Introduction to linear regression

- Let's finally get to building our first machine learning model!
- One of the most widely used tools for predicting a quantitative variable is **linear regression**.
- As the name suggests, linear regression assumes that the dependence of a response variable Y on predictors $X_1, \ldots, X_p$ is linear.
- Even if true relationships are rarely linear, this simple approach is extremely useful both conceptually and practically.
- Let's start with the simplest case: predicting a quantitative variable Y on the basis of a single predictor X. We assume therefore:

$$Y = \beta_0 + \beta_1 \cdot X + \epsilon,$$

where β_0 and β_1 are unknown **parameters** or **coefficients**.

Introduction to linear regression

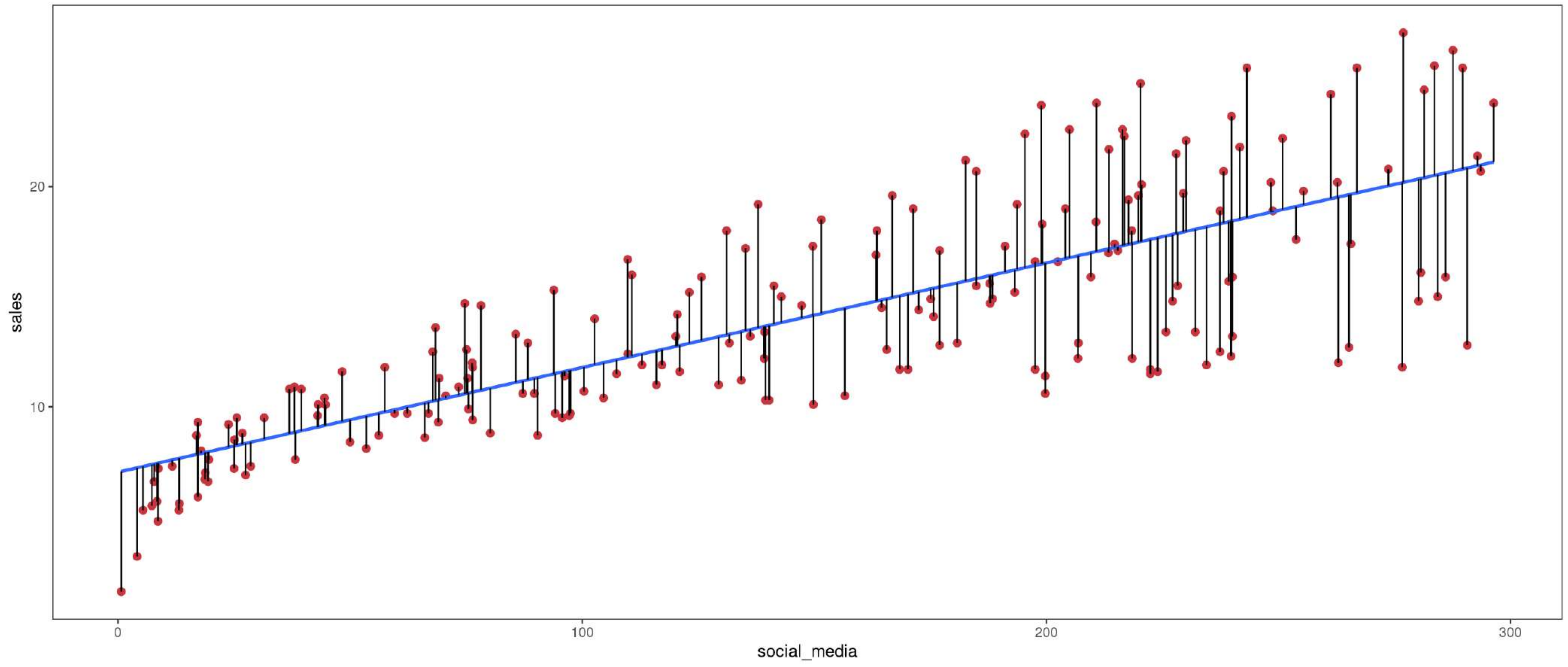
- In the case of **simple** linear regression, there are only two coefficients:
  - The slope coefficient β_1. It specifies how much Y changes on average for a given change in X.
  - The intercept β_0. It specifies the value that Y takes on average, if $X = 0$.
- Given some estimates $\hat{\beta}_0$ and $\hat{\beta}_1$, we can predict future values of Y by using the regression line:

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 \cdot x,$$

  where $\hat{y}$ indicates a prediction of Y on the basis of X being equal to x.
- For the purposes of illustration, let's come back to the example with sales as a function of advertising budgets.

Introduction to linear regression

Suppose we are only interested in the relationship between the `social_media` budget and `sales`. Our goal is to obtain estimates $\hat{\beta}_0$ and $\hat{\beta}_1$, such that $y_i \approx \hat{\beta}_0 + \hat{\beta}_1 x_i$ for $i = 1, \ldots, n$. There are multiple ways to do this.



Estimating coefficients through OLS

- Let $\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i$ be the **fitted value** for Y based on the ith observation for X. Then $e_i = y_i - \hat{y}_i$ represents the ith **residual.**
- One possibility to choose the estimates for the coefficients is to minimize the **residual sum of squares (RSS)**:

$$RSS = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} (y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i))^2.$$

- The values of $\hat{\beta}_0$ and $\hat{\beta}_1$ that minimize this expression are called **ordinary least squares (OLS)** estimators and are given by:

$$\hat{\beta}_1 = \frac{\frac{1}{n}\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\frac{1}{n}\sum_i (x_i - \bar{x})^2}, \qquad \hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x},$$

where $\bar{x}$ and $\bar{y}$ denote the means of the variables.

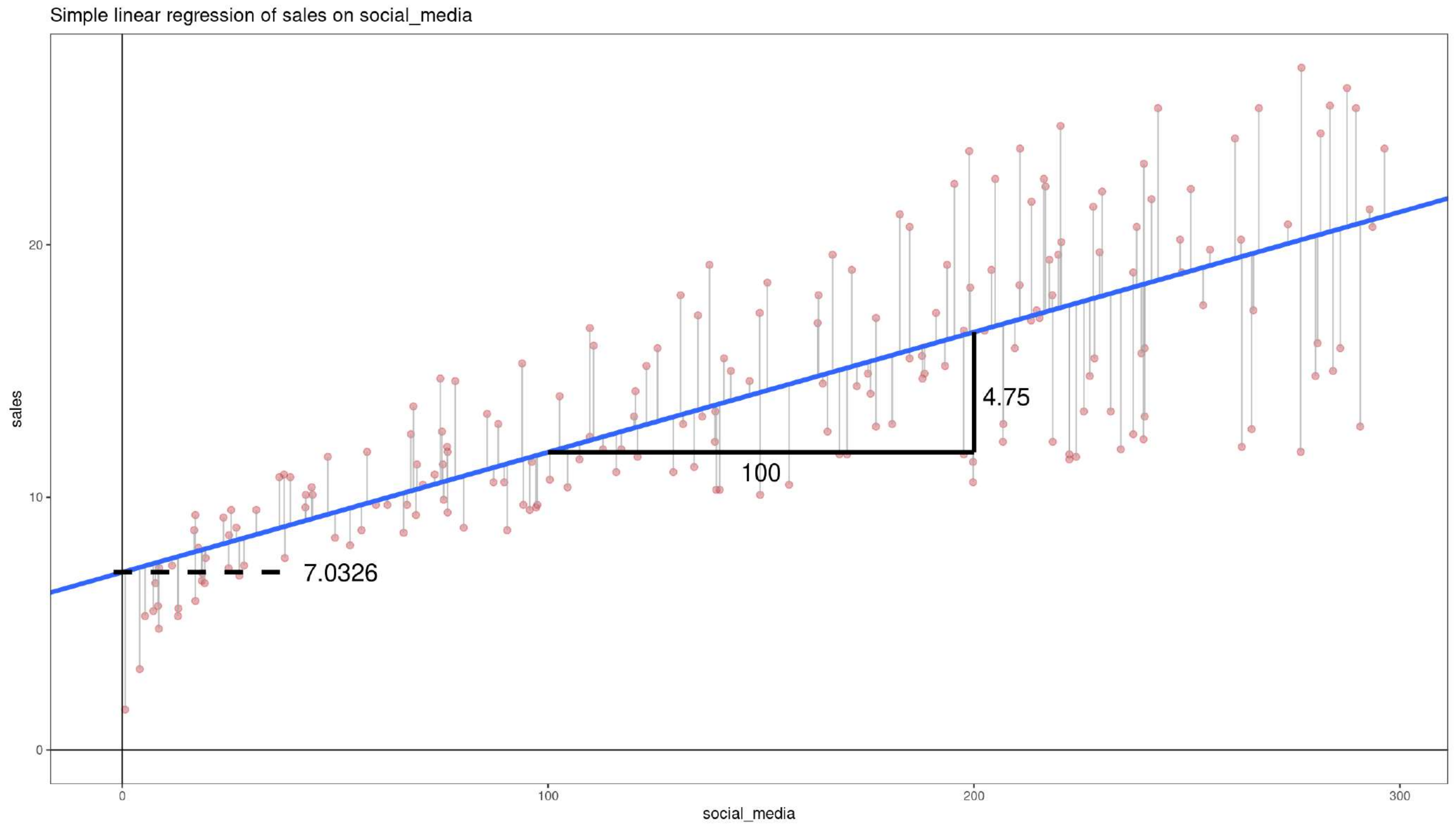
Estimating coefficients – Example

- Estimating the parameters for the simple linear regression where `social_media` is X and `sales` is Y, we get:

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 \cdot X = 7.0326 + 0.0475 \cdot X$$

- Interpretation of the coefficients:
  - $\hat{\beta}_0 = 7.0326$: when no money is invested in social media advertising, the model expects average sales of 7.0326 units.
  - $\hat{\beta}_1 = 0.0475$: for every additional 100 dollars spent on social media advertising, the expected sales increase by around $100 \cdot 0.0475 \approx 4.75$ units on average.

Estimating coefficients – Example



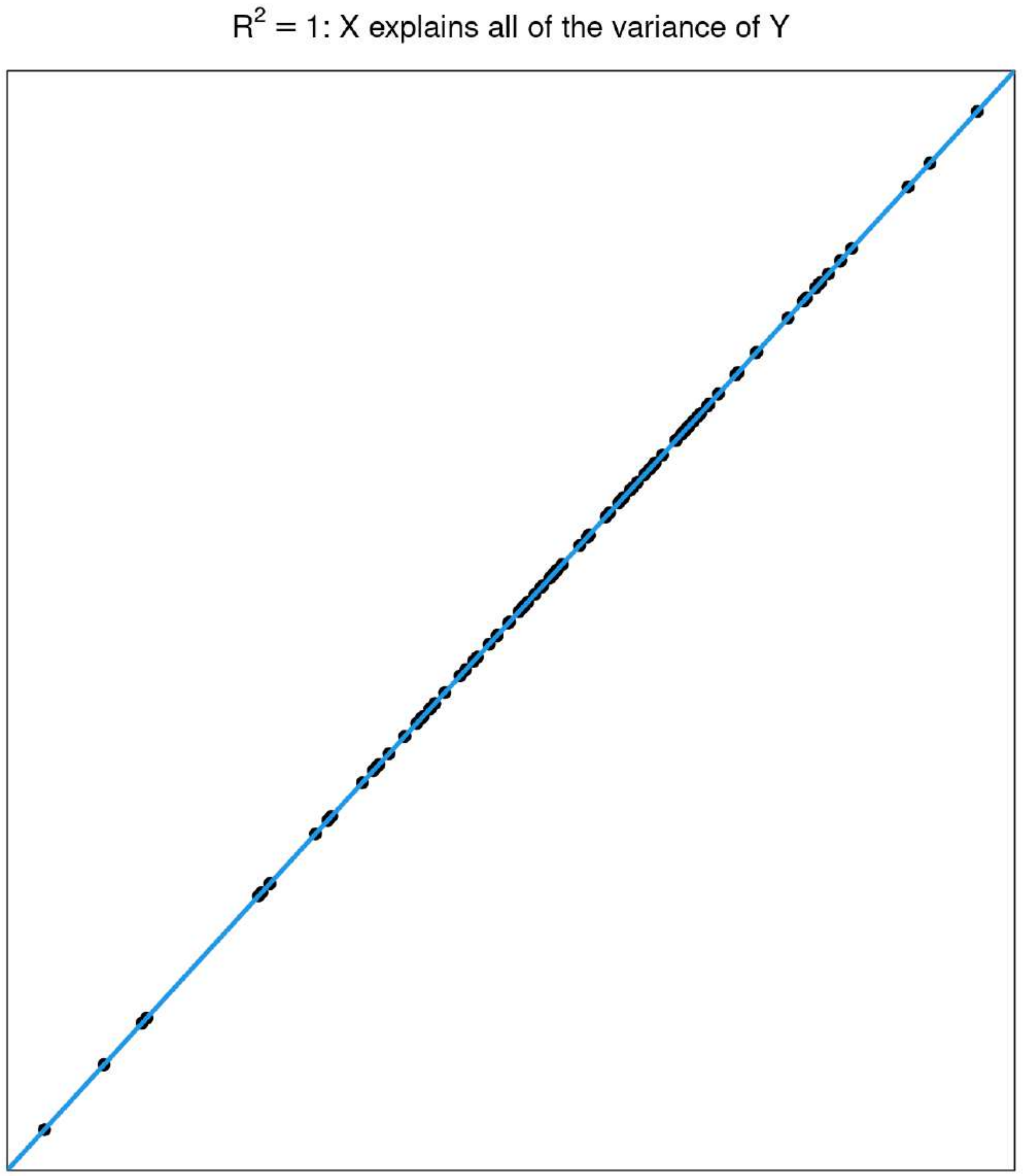
Assessing the goodness-of-fit

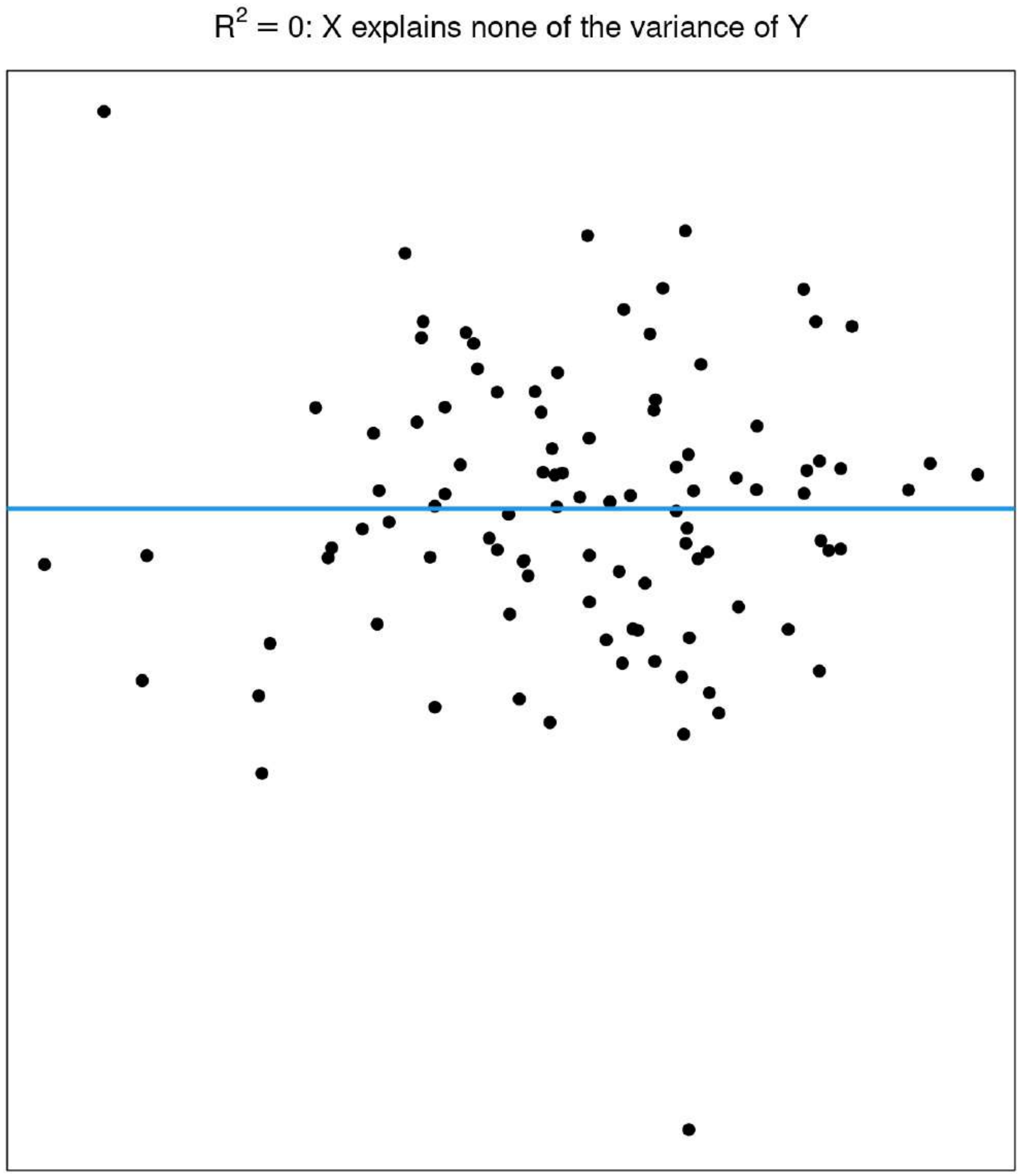
- How well a regression line fits the data is described by its **goodness-of-fit**. It is typically assessed by the **coefficient of determination** R^2.
- The R^2 measures the proportion of explained variance:

$$R^2 = \frac{TSS - RSS}{TSS} = 1 - \frac{RSS}{TSS} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2},$$

  where TSS is the total sum of squares as defined by the expression in the denominator.
- The value of the R^2 lies between 0 and 1:
  - $R^2 = 1$: 100% of the variance of Y can be explained by X.
  - $R^2 = 0$: 0% of the variance of Y can be explained by X.

Assessing the goodness-of-fit





Statistical properties of OLS estimators

- Under certain assumptions on the underlying true regression model, we can derive **sampling distributions** of the OLS estimators $\hat{\beta}_0$ and $\hat{\beta}_1$, i.e. how these estimators will be distributed when computing them over several data sets.
- We can use these sampling distributions use for **testing hypotheses** like

$$H_0 : \beta_1 = 0, \qquad H_1 : \beta_1 \neq 0$$

- … or to compute two-sided **confidence intervals** like

$$[\hat{\beta}_1 - t_{1-\alpha/2;n-2} \cdot \hat{\sigma}_{\hat{\beta}_1}; \hat{\beta}_1 + t_{1-\alpha/2;n-2} \cdot \hat{\sigma}_{\hat{\beta}_1}],$$

where $t_{1-\alpha/2;n-2}$ is the $(1-\alpha/2)$-quantile of a t-distribution with $n-2$ degrees of freedom and $\hat{\sigma}_{\hat{\beta}_1}$ is the standard error of the estimator of the slope coefficient $\hat{\beta}_1$.

Linear Regression in R

- However, this is a data science course and not a statistics course, so we will not compute these quantities by hand. Instead, we want to use R for all this.
- To estimate a linear regression model in R, we need two things to pass into the function `lm` (which stands for "linear model"):
  - A `data.frame` holding the variables we want included in the model.
  - A `formula` that specifies the model. In simple linear regression, this will take the form of `response ~ predictor`.
- So, in our advertising example, we would do the following:

```r
advertising <- read.csv("data/advertising.csv")
advertising <- advertising[, -c(1, 2, 7)]

simple_reg <- lm(sales ~ social_media, data = advertising)
```

Linear Regression in R

- We can then have a look at the result using the `summary` function:

```r
simple_reg_summary <- summary(simple_reg)
simple_reg_summary
```

```
Call:
lm(formula = sales ~ social_media, data = advertising)

Residuals:
    Min      1Q  Median      3Q     Max 
-8.3860 -1.9545 -0.1913  2.0671  7.2124 

Coefficients:
             Estimate Std. Error t value Pr(>|t|)    
(Intercept) 7.032594   0.457843   15.36   <2e-16 ***
social_media 0.047537   0.002691   17.67   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

- This output contains a lot of useful information. First of all:
  - `Call`: summary of the estimated regression model
  - `Residuals`: a five-point-summary of the residuals e_i

Linear Regression in R

- Then, it displays a table called `Coefficients` with six columns:
  - 1st column: name of the variable pertaining to the corresponding coefficient (here only `(Intercept)` and `social_media`).
  - 2nd column (`Estimate`): OLS estimate of the coefficient, so $\hat{\beta}_i$ for $i = 0, 1$.
  - 3rd column (`Std. Error`): Estimate of the coefficient standard errors, so $\hat{\sigma}_{\hat{\beta}_i}$.
  - 4th column (`t value`): Test statistic t for the two-sided hypothesis test against 0, i.e. to test the null hypothesis $H_0 : \beta_i = 0$ against the alternative $H_1 : \beta_i \neq 0$.
  - 5th column (`Pr(>|t|)`): p-value of the corresponding hypothesis test.
  - 6th column: “stars of significance”
- Finally, additional information is displayed:
  - `Residual standard error`: Estimate of the square root of $\mathrm{Var}(\epsilon)$.
  - `Multiple R-squared`: R^2
  - `Adjusted R-squared` / `F-statistic`: see example with multiple predictors.

Linear Regression in R – Accessor functions

From the created model object `simple_reg`, we can extract:

- The coefficient estimates $\hat{\beta}_0$ and $\hat{\beta}_1$ using the function `coef`:

```r
coef(simple_reg)
```

```
(Intercept) social_media 
 7.03259355   0.04753664 
```

- The fitted values $\hat{y}_i$ using the function `fitted`:

```r
head(fitted(simple_reg))
```

```
        1         2         3         4         5         6 
17.970775  9.147974  7.850224 14.234395 15.627218  7.446162 
```

- The residuals $e_i = y_i - \hat{y}_i$ using the function `residuals`:

```r
head(residuals(simple_reg))
```

```
         1          2          3          4          5          6 
 4.1292255  1.2520260  1.4497762  4.2656054 -2.7272181 -0.2461623 
```

- The residual standard error using the function `sigma`:

```r
sigma(simple_reg)
```

```
[1] 3.258656
```

Linear Regression in R – Accessor functions

- Confidence intervals for all regression coefficients at level $1 - \alpha$ using the function `confint` and its `level` argument:

```r
confint(simple_reg, level = 0.95)
```

```
                  2.5 %     97.5 %
(Intercept)  6.12971927 7.93546783
social_media 0.04223072 0.05284256
```

Additionally, we can extract the following quantities from the created model summary object `simple_reg_summary`:

- The coefficient estimates, their standard errors, t and p values using the function `coef`:

```r
coef(simple_reg_summary)
```

```
               Estimate  Std. Error  t value     Pr(>|t|)
(Intercept)  7.03259355 0.457842940 15.36028 1.40630e-35
social_media 0.04753664 0.002690607 17.66763 1.46739e-42
```

- The R^2 by accessing the `r.squared` field of the corresponding list:

```r
simple_reg_summary$r.squared
```

```
[1] 0.6118751
```

Linear Regression in R – Prediction

- If we are interested in **prediction** rather than **inference**, we need a way to predict new data points. In R, this functionality is provided by the `predict` function.
- It requires the estimated model we want to use for prediction and the values of the features for which predictions should be generated in a `data.frame`.
- Suppose, we want to predict `sales` for `social_media` budgets of 100 and 200:

```r
new_data <- data.frame(social_media = c(100, 200))
predict(simple_reg, newdata = new_data)
```

```
       1        2 
11.78626 16.53992 
```

- Note that the names of the features (here only `social_media`) in `newdata` have to be identical to their names in the training data:

```r
new_data <- data.frame(social_med = c(100, 200))
predict(simple_reg, newdata = new_data)
```

```
Error in eval(predvars, data, env): Objekt 'social_media' nicht gefunden
```

K-fold cross-validation in R

- Now, we have all we need to estimate the test MSE of our model using **K-fold cross validation**! Let's say for $K = 5$.
- Let's go step by step: first, we randomly assign each observation to one of the 5 folds. The randomness is once again introduced via the function `sample`:

```r
n <- nrow(advertising)
K <- 5

advertising$fold <- sample(rep(1:K, length.out = n))
head(advertising, 10)
```

```
   social_media radio newspaper sales fold
1         230.1  37.8      69.2  22.1    1
2          44.5  39.3      45.1  10.4    1
3          17.2  45.9      69.3   9.3    5
4         151.5  41.3      58.5  18.5    5
5         180.8  10.8      58.4  12.9    3
6           8.7  48.9      75.0   7.2    4
7          57.5  32.8      23.5  11.8    1
8         120.2  19.6      11.6  13.2    3
9           8.6   2.1       1.0   4.8    3
10        199.8   2.6      21.2  10.6    2
```

K-fold cross-validation in R

- Next, we write a loop, in which for every fold $k = 1, \ldots, 5$:
  - We fit our simple linear regression model on all data except the kth fold.
  - Using this model, we predict `sales` on the kth fold.
  - Since we know the true `sales` for all observations, we can then compute the MSE on the kth fold.
  - Save the MSE of the kth fold in a vector.

```r
mses_linear <- numeric(K)

for(k in 1:K){
  m <- lm(sales ~ social_media, data = subset(advertising, fold != k))
  preds <- predict(m, newdata = subset(advertising, fold == k))
  mse <- mean((preds - advertising$sales[advertising$fold == k])^2)
  mses_linear[k] <- mse
}
mses_linear
```

```
[1]  7.03702 10.98168 12.21465 10.85549 12.54239
```

K-fold cross-validation in R

- Finally, we can compute the **cross-validated MSE** by calculating the mean of the MSEs across the 5 folds:

```r
mean(mses_linear)
```

```
[1] 10.72624
```

- Since the MSE lives on a squared scale (due to the squaring in its computation), we can bring the scale back to the original linear scale by taking the square root. The result is called the **root mean squared error (RMSE)**:

```r
sqrt(mean(mses_linear))
```

```
[1] 3.275095
```

- In and of itself, these numbers do not tell us too much yet. However, we will be comparing them against the cross-validated (R)MSE of other methods, to see which one is the best.

Incorporating non-linearities

- The term **linear** in “linear regression” actually refers only to linearity in parameters β_0 and β_1.
- We can therefore easily incorporate **non-linearities** by appropriately defining the dependent variable Y and the independent variable X. For example:
  - $\log(Y) = \beta_0 + \beta_1 \cdot X + \epsilon$
  - $\log(Y) = \beta_0 + \beta_1 \cdot \log(X) + \epsilon$
  - $Y = \beta_0 + \beta_1 \cdot \sqrt{X} + \epsilon$
- Let’s estimate each of these models in our advertising example and assess their test MSEs using 5-fold cross validation again. First, we add the log and square root of the relevant variable to our data set:

```r
advertising$log_sales <- log(advertising$sales)
advertising$log_social_media <- log(advertising$social_media)
advertising$sqrt_social_media <- sqrt(advertising$social_media)
```

Incorporating non-linearities

- Now, let's assess the test MSE of each of the three above models using 5-fold cross validation:

```r
mses_nonlinear <- matrix(0, K, 3)

for(k in 1:K){
  m1 <- lm(log_sales ~ social_media, data = subset(advertising, fold != k))
  m2 <- lm(log_sales ~ log_social_media, data = subset(advertising, fold != k))
  m3 <- lm(sales ~ sqrt_social_media, data = subset(advertising, fold != k))

  preds1 <- exp(predict(m1, newdata = subset(advertising, fold == k)))
  preds2 <- exp(predict(m2, newdata = subset(advertising, fold == k)))
  preds3 <- predict(m3, newdata = subset(advertising, fold == k))

  mses_nonlinear[k, 1] <- mean((preds1 - advertising$sales[advertising$fold == k])^2)
  mses_nonlinear[k, 2] <- mean((preds2 - advertising$sales[advertising$fold == k])^2)
  mses_nonlinear[k, 3] <- mean((preds3 - advertising$sales[advertising$fold == k])^2)
}
colMeans(mses_nonlinear)
```

```
[1] 11.83003 10.71977 10.39436
```

- The model $Y = \beta_0 + \beta_1 \cdot \sqrt{X}$ gives the lowest test MSE out of the three and lower than the linear model as well, albeit not by much.

Multiple linear regression

- To improve our model even further, we probably have to include additional features into it. For instance, `sales` will probably also depend on the `radio` and `newspaper` advertising budget.
- In the context of the linear model, we can include additional variables to create a **multiple linear regression** of the form

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p + \epsilon$$

- We interpret β_j as the average effect on Y of a one unit increase in X_j, **holding all other predictors fixed** (i.e. ceteris paribus).
- As before, the regression coefficients $\beta_0, \beta_1, \ldots, \beta_p$ are unknown and need to be estimated from the training data.

Multiple linear regression – OLS estimation

- As in the simple linear regression case, we choose the parameters to minimize the **residual sum of squares (RSS)**:

$$RSS = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} (y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_{1i} + \ldots + \hat{\beta}_p x_{pi}))$$

- The values of $\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_p$ that minimize this expression are complicated formulas of the data that are most easily represented using matrix algebra. These formulas do not really matter to us, as R computes them for us anyway.
- Estimating parameters for the multiple linear regression where `sales` is Y, `social_media` is X_1, `radio` is X_2 and `newspaper` is X_3, we get:

$$\hat{Y} = 2.9389 + 0.0458 \cdot X_1 + 0.1885 \cdot X_2 - 0.0010 \cdot X_3$$

Multiple linear regression – OLS estimation

- Let's again go through the interpretation of each of those coefficients:
  - $\hat{\beta}_0 = 2.9389$: when no money is invested in advertising through any of the three channels, the model expects average sales of 2.9389 units.
  - $\hat{\beta}_1 = 0.0458$: for every additional 100 dollars spent on social media advertising **ceteris paribus**, the expected sales increase by around $100 \cdot 0.0458 \approx 4.58$ units on average.
  - $\hat{\beta}_2 = 0.1885$: for every additional 100 dollars spent on radio advertising **ceteris paribus**, the expected sales increase by around $100 \cdot 0.1885 \approx 18.85$ units on average.
  - $\hat{\beta}_3 = -0.0010$: for every additional 100 dollars spent on newspaper advertising **ceteris paribus**, the expected sales **decrease** by around $100 \cdot 0.0010 \approx 0.10$ units on average.

Assessing the goodness-of-fit

- The R^2 is still computed and interpreted in the same way as before:

$$R^2 = \frac{TSS - RSS}{TSS} = 1 - \frac{RSS}{TSS} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

- It now measures the proportion of the variance of Y explained **jointly** by the predictors $X_1, \dots, X_p$. Alternatively, one can say that it measures the proportion of the variance of Y explained by the model.
- When adding additional variables to the model, the R^2 **never decreases** and usually increases. This makes it a poor tool for deciding whether an additional variable should be added to the model.

Assessing the goodness-of-fit

- An alternative to the R^2 that is the so-called **adjusted** R^2:

$$R^2_{\text{adj}} = 1 - \frac{(1 - R^2)(n - 1)}{n - p - 1}$$

- It no longer has the interpretation of the R^2, but it is more useful for **comparing models**. Consider the following example:
  - The simple linear regression of `sales` on `social_media` had an adjusted R^2 of 0.6099 (see corresponding R output from earlier).
  - The multiple linear regression of `sales` on all three advertising budgets has an adjusted R^2 of 0.8956 (see upcoming R output).
  - Therefore, the adjusted R^2 strongly suggests that the bigger model is superior to the smaller model.

Statistical properties of OLS estimators

- Using the standard assumptions of the multiple linear regression model, we can again derive **sampling distributions** of the OLS estimators $\hat{\beta}_0, \ldots, \hat{\beta}_p$ and thus construct t-tests for hypotheses like

$$H_0 : \beta_j = 0, \qquad H_1 : \beta_j \neq 0$$

- Additionally, we can test the null hypothesis that none of the coefficients are useful in predicting the response, i.e.

$$H_0 : \beta_1 = \beta_2 = \ldots = \beta_p = 0, \qquad H_1 : \beta_j \neq 0 \text{ für min. ein } j,$$

using the F-statistic

$$F = \frac{n-p-1}{p} \cdot \frac{TSS - RSS}{RSS} \sim F_{p,n-p-1}$$

Multiple linear regression in R

- However, again, this course focuses on how to use R to estimate these types of models and compute these types of statistics.
- To estimate a **multiple** linear regression in R, we simply combine the predictors on the right side of the `~` using a `+` sign, i.e. `response ~ predictor1 + predictor2 + ... + predictorp`
- So, to estimate the model with all three predictors in our advertising example, we would run the following line of code:

```r
multiple_reg <- lm(sales ~ social_media + radio + newspaper, data = advertising)
```

- Fortunately, many of the other functionalities we have seen in simple linear regression neatly generalize to the multiple linear regression case!

Multiple linear regression in R

- Again, we inspect the result using the `summary` function:

```r
summary(multiple_reg)
```

```

Call:
lm(formula = sales ~ social_media + radio + newspaper, data = advertising)

Residuals:
    Min      1Q  Median      3Q     Max 
-8.8277 -0.8908  0.2418  1.1893  2.8292 

Coefficients:
              Estimate Std. Error t value Pr(>|t|)    
(Intercept)   2.938889   0.311908   9.422   <2e-16 ***
social_media  0.045765   0.001395  32.809   <2e-16 ***
radio         0.188530   0.008611  21.893   <2e-16 ***
newspaper    -0.001037   0.005871  -0.177     0.86    
```

  - All predictors except for `newspaper` have a significant influence on `sales` on the 0.1% significance level.
  - Around 89.7% of the variance of `sales` can be explained with this model.

Comparing model performance

But how does its test MSE hold up to the models estimated so far? Let's investigate this question using our cross-validation routine again:

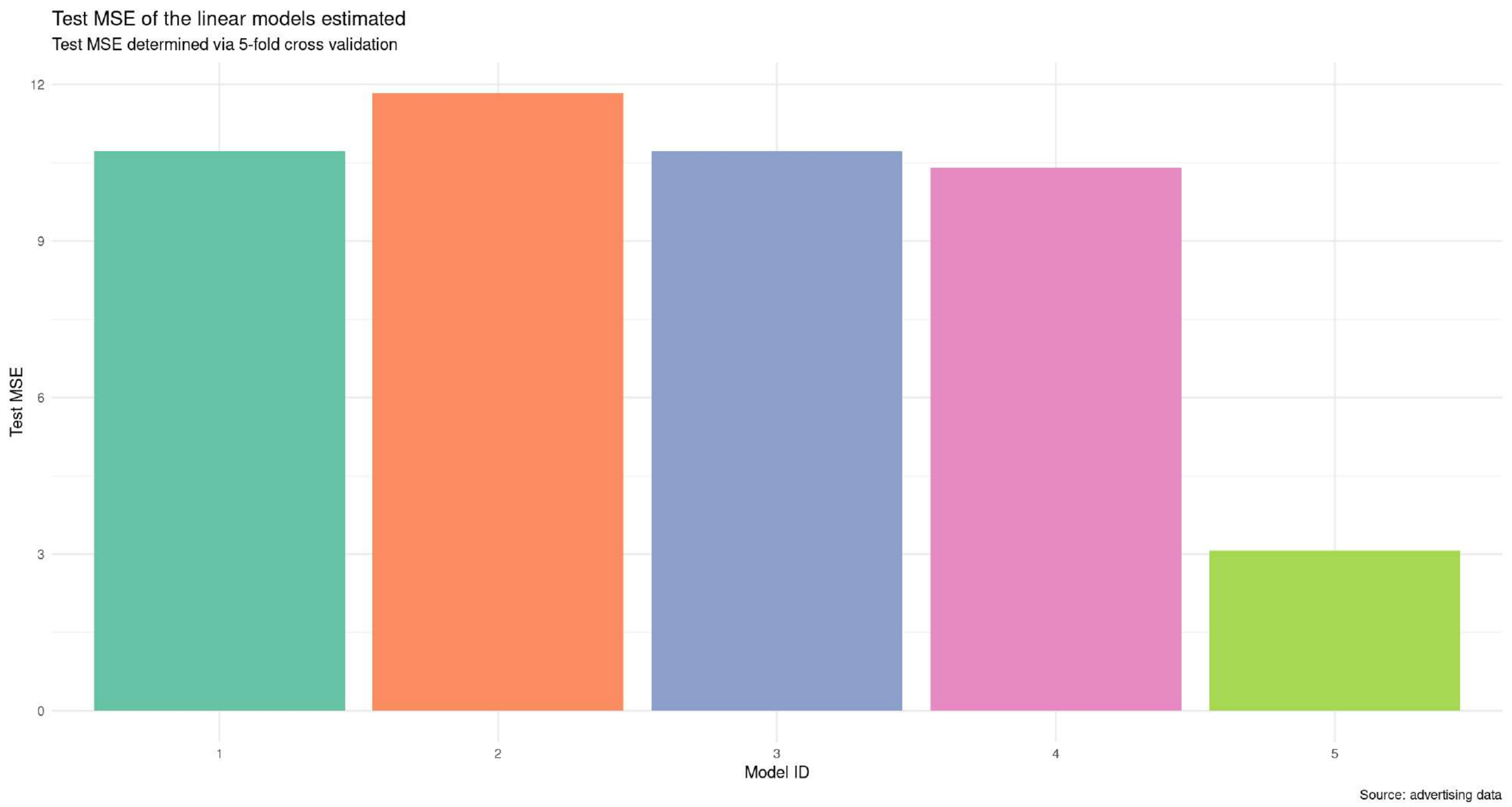
```r
mses_big <- numeric(K)

for(k in 1:K){
  m <- lm(sales ~ social_media + radio + newspaper,
          data = subset(advertising, fold != k))
  preds <- predict(m, newdata = subset(advertising,
  mse <- mean((preds - advertising$sales[advertisin
  mses_big[k] <- mse
}

results <- data.frame(model_id = factor(1:5),
                      model_names = model_names,
                      test_mse = c(mean(mses_linear
                                   colMeans(mses_no
                                   mean(mses_big)))
knitr::kable(results[,-1], col.names = c("Model", "
             digits = 2, row.names = TRUE)
```

	Model	Test MSE
1	$Y \sim X_1$	10.73
2	$\log(Y) \sim X_1$	11.83
3	$\log(Y) \sim \log(X_1)$	10.72
4	$Y \sim \sqrt{X_1}$	10.39
5	$Y \sim X_1 + X_2 + X_3$	3.06

Unsurprisingly, the model that uses all three advertising budgets as predictors is **much** better than any of the models that just uses the social media budget!

Comparing model performance

Plot Code



Categorical predictors

- So far, all the predictors we have looked at have been **quantitative** variables. In practice, we often have to deal with **categorical predictors** as well.
- To see an example, let's consider the `Credit` data set from the `ISLR` package. It contains socio-economic information on 400 credit card customers.
- The data set contains seven quantitative variables:

```r
library(ISLR)
head(Credit[, c("Income", "Limit", "Rating", "Cards", "Age", "Education", "Balance")])
```

```
   Income Limit Rating Cards Age Education Balance
1  14.891  3606    283     2  34        11     333
2 106.025  6645    483     3  82        15     903
3 104.593  7075    514     4  71        11     580
4 148.924  9504    681     3  36        11     964
5  55.882  4897    357     2  68        16     331
6  80.180  8047    569     4  77        10    1151
```

- The goal here is to build a predictive model for `balance`, the average credit card balance of a given customer.

Categorical predictors

- However, aside from the quantitative variables, the data set also contains four **qualitative (categorical)** variables:

```r
head(Credit[, c("Gender", "Student", "Married", "Ethnicity")])
```

```
  Gender Student Married Ethnicity
1   Male      No     Yes Caucasian
2 Female     Yes     Yes     Asian
3   Male      No      No     Asian
4 Female      No      No     Asian
5   Male      No     Yes Caucasian
6   Male      No      No Caucasian
```

- So how can we include the information in these qualitative predictors to benefit our linear model for `balance`?
- The answer lies in **dummy variables**, i.e. variables that only take on the values 0 or 1. For a categorical variable with K levels, we create $K-1$ dummy variables. Let's see an example of how this works.

Dummy variables

- The variable `Ethnicity` takes on $K = 3$ different values:

```r
table(Credit$Ethnicity)
```

```
African American            Asian        Caucasian 
              99              102              199 
```

- So, we can **encode** the information in the `Ethnicity` variable in $K - 1 = 2$ dummy variables, called d_1 and d_2. For example, one could be used to indicate Asian ethnicity and the other to indicate Caucasian:

$$d_{1i} = \begin{cases} 1 & \text{if person } i \text{ is of Asian ethnicity} \\ 0 & \text{otherwise} \end{cases}$$

$$d_{2i} = \begin{cases} 1 & \text{if person } i \text{ is of Caucasian ethnicity} \\ 0 & \text{otherwise} \end{cases}$$

Note that a third dummy variable is not necessary.

Dummy variables

- We could manually add these dummy variables to the data set:

```r
Credit$Asian <- ifelse(Credit$Ethnicity == "Asian", 1, 0)
Credit$Caucasian <- ifelse(Credit$Ethnicity == "Caucasian", 1, 0)
head(Credit[,c("Ethnicity", "Asian", "Caucasian")], 8)
```

```
         Ethnicity Asian Caucasian
1        Caucasian     0         1
2            Asian     1         0
3            Asian     1         0
4            Asian     1         0
5        Caucasian     0         1
6        Caucasian     0         1
7 African American     0         0
8            Asian     1         0
```

- With the categorical data now numerically encoded, we can estimate the following model:

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 d_{1i} + \hat{\beta}_2 d_{2i},$$

where y_i is the ith observation of the `balance` variable.

Dummy variables

- Note that since d_{1i} and d_{2i} can only take two possible values (0 or 1), this model can only predict three distinct values:

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 d_{1i} + \hat{\beta}_2 d_{2i} = \begin{cases} \hat{\beta}_0 + \hat{\beta}_1 & \text{if } i \text{ is Asian} \\ \hat{\beta}_0 + \hat{\beta}_2 & \text{if } i \text{ is Caucasian} \\ \hat{\beta}_0 & \text{if } i \text{ is African American} \end{cases}$$

- Therefore, we can interpret the regression coefficients as follows:
  - $\hat{\beta}_0$ is the average credit card balance of people of African American ethnicity. This category constitutes the so-called **reference category**.
  - $\hat{\beta}_1$ is the difference in average credit card balance between people of Asian ethnicity and the reference category (African American).
  - $\hat{\beta}_2$ is the difference in average credit card balance between people of Caucasian ethnicity and the reference category (African American).

Dummy variables in R

- Let’s estimate this model with R. Fortunately, as long as the categorical variables are properly encoded as `factor`s, we do not have to create the dummy variables ourselves. Let’s check:

```r
class(Credit$Ethnicity)
```

```
[1] "factor"
```

```r
levels(Credit$Ethnicity)
```

```
[1] "African American" "Asian"            "Caucasian"
```

- This looks good! Note that R will always automatically use the first factor level as the **reference category** when estimating a linear model with factors:

```r
credit_model1 <- lm(Balance ~ Ethnicity, data = Credit)
```

- Note that with the proper `factor` encoding of `Ethnicity`, we do not need to make use of the dummy variables `Asian` and `Caucasian` we manually added to the data set.

Dummy variables in R

- Let's have a look at the created model object:

```r
summary(credit_model1)
```

```
Call:
lm(formula = Balance ~ Ethnicity, data = Credit)

Residuals:
    Min      1Q  Median      3Q     Max 
-531.00 -457.08  -63.25  339.25 1480.50 

Coefficients:
                   Estimate Std. Error t value Pr(>|t|)    
(Intercept)          531.00      46.32  11.464   <2e-16 ***
EthnicityAsian       -18.69      65.02  -0.287    0.774    
EthnicityCaucasian   -12.50      56.68  -0.221    0.826    
---
```

- This shows that on average…
  - people of African American ethnicity have a balance of 531.
  - people of Asian ethnicity have a balance of $531 - 18.69 = 512.31$.
  - people of Caucasian ethnicity have a balance of $531 - 12.50 = 518.50$.

Dummy variables in R

- However, as the regression output shows, these differences in average credit card balances by ethnicity are clearly not significant.
- Note that the group averages inferred from the regression coefficients genuinely correspond to the group averages:

```r
round(tapply(Credit$Balance, Credit$Ethnicity, mean), 2)
```

```
African American            Asian        Caucasian 
          531.00           512.31           518.50 
```

- Based on this approach of **dummy variable encoding**, we can include any number of categorical variables into the regression equation.
- As an example, the next slide displays the regression output from using **all available predictors** (quantitative and qualitative) to model credit card balance in the usual additive fashion.

Quantitative and qualitative predictors

```r
credit_model2 <- lm(Balance ~ Income + Limit + Rating + Cards + Age + Education +
                              Gender + Student + Married + Ethnicity,
                    data = Credit)
summary(credit_model2)
```

```

Call:
lm(formula = Balance ~ Income + Limit + Rating + Cards + Age + 
    Education + Gender + Student + Married + Ethnicity, data = Credit)

Residuals:
    Min      1Q  Median      3Q     Max 
-161.64  -77.70  -13.49   53.98  318.20 

Coefficients:
                     Estimate Std. Error t value Pr(>|t|)    
(Intercept)        -479.20787   35.77394 -13.395  < 2e-16 ***
Income               -7.80310    0.23423 -33.314  < 2e-16 ***
Limit                 0.19091    0.03278   5.824 1.21e-08 ***
```

With over 95% variance explained, this is a very good model for credit card balance. However, is it reasonable to leave all of the predictors in the model? Or are we better off removing some of them?

Variable selection

- The task of determining which predictors are associated with the response and are thus to be included in the model, is referred to as **variable selection**.
- We can decide which variables to select on the basis of statistics like
  - cross-validated MSE (the lower the better).
  - adjusted R^2 (the higher the better).
  - Akaike information criterion (AIC, the lower the better)
- Most direct approach: **best subset regression**: compute OLS fit for every subset of variables and choose the best according to some metric:
  - Unfortunately, there are a total of 2^p models to consider.
  - For $p = 2$, that's 4 models, but for $p = 30$ that is 1,073,741,824 models to estimate! This is not practical.
- Instead, we need a more efficient approach to select a smaller set of models.

Forward selection

- One approach to this task is **forward** selection (using AIC):
  - Begin with a **null model**, i.e. a model that contains no predictors.
  - Fit p simple linear regressions (one for each predictor individually) and add to the null model the variable that results in the lowest AIC.
  - Add to that model the variable that results in the lowest AIC for the new two-variable model.
  - Continue until some stopping rule is satisfied, e.g. no model with a smaller AIC can be fitted.
- In R, forward selection can be achieved with the help of the `step` function. It requires the null model (as a lower bound) and the full model with all variables (as an upper bound) as input.

Forward selection

```r
m0 <- lm(Balance ~ 1, data = Credit)
credit_model_fw <- step(m0, scope = Balance ~ Income + Limit + Rating + Cards + Age +
                                    Education + Gender + Student + Married + Ethnicity,
                              direction = "forward")
```

```
Start:  AIC=4905.56
Balance ~ 1

            Df Sum of Sq      RSS    AIC
+ Rating     1  62904790 21435122 4359.6
+ Limit      1  62624255 21715657 4364.8
+ Income     1  18131167 66208745 4810.7
+ Student    1   5658372 78681540 4879.8
+ Cards      1    630416 83709496 4904.6
<none>                   84339912 4905.6
+ Gender     1     38892 84301020 4907.4
+ Education  1      5481 84334431 4907.5
+ Married    1      2715 84337197 4907.5
+ Age        1       284 84339628 4907.6
```

The final model found with forward selection contains predictors `Rating`, `Income`, `Student`, `Limit`, `Cards` and `Age`. It has an AIC of 3679.9. The object `credit_model_fw` can be used like any other `lm` object.

Backward selection

- Another approach to the task of variable selection is **backward** selection:
  - Begin with a **full model**, i.e. a model that contains all available predictors.
  - Fit p individual linear regressions, removing one variable from the full model at a time. Choose the model that has the lowest AIC.
  - Repeat with the resulting model and continue until some stopping rule is satisfied, e.g. no model with a smaller AIC can be fitted.
- In R, backward selection can also be achieved with the help of the `step` function. It only requires the full model (as an upper bound) and the direction `backward` as input.

Backward selection

```r
credit_model_bw <- step(credit_model2, direction = "backward")
```

```
Start:  AIC=3686.22
Balance ~ Income + Limit + Rating + Cards + Age + Education + 
    Gender + Student + Married + Ethnicity

            Df Sum of Sq     RSS    AIC
- Ethnicity  2     14084 3800814 3683.7
- Education  1      4615 3791345 3684.7
- Married    1      6619 3793349 3684.9
- Gender     1     11269 3798000 3685.4
<none>                   3786730 3686.2
- Age        1     42558 3829288 3688.7
- Rating     1     52314 3839044 3689.7
- Cards      1    162702 3949432 3701.0
- Limit      1    331050 4117780 3717.7
```

The final model found with backward selection also contains predictors `Rating`, `Income`, `Student`, `Limit`, `Cards` and `Age`. With an AIC of 3679.9, it is the same model that was found with forward selection. The object `credit_model_bw` can be used like any other `lm` object.

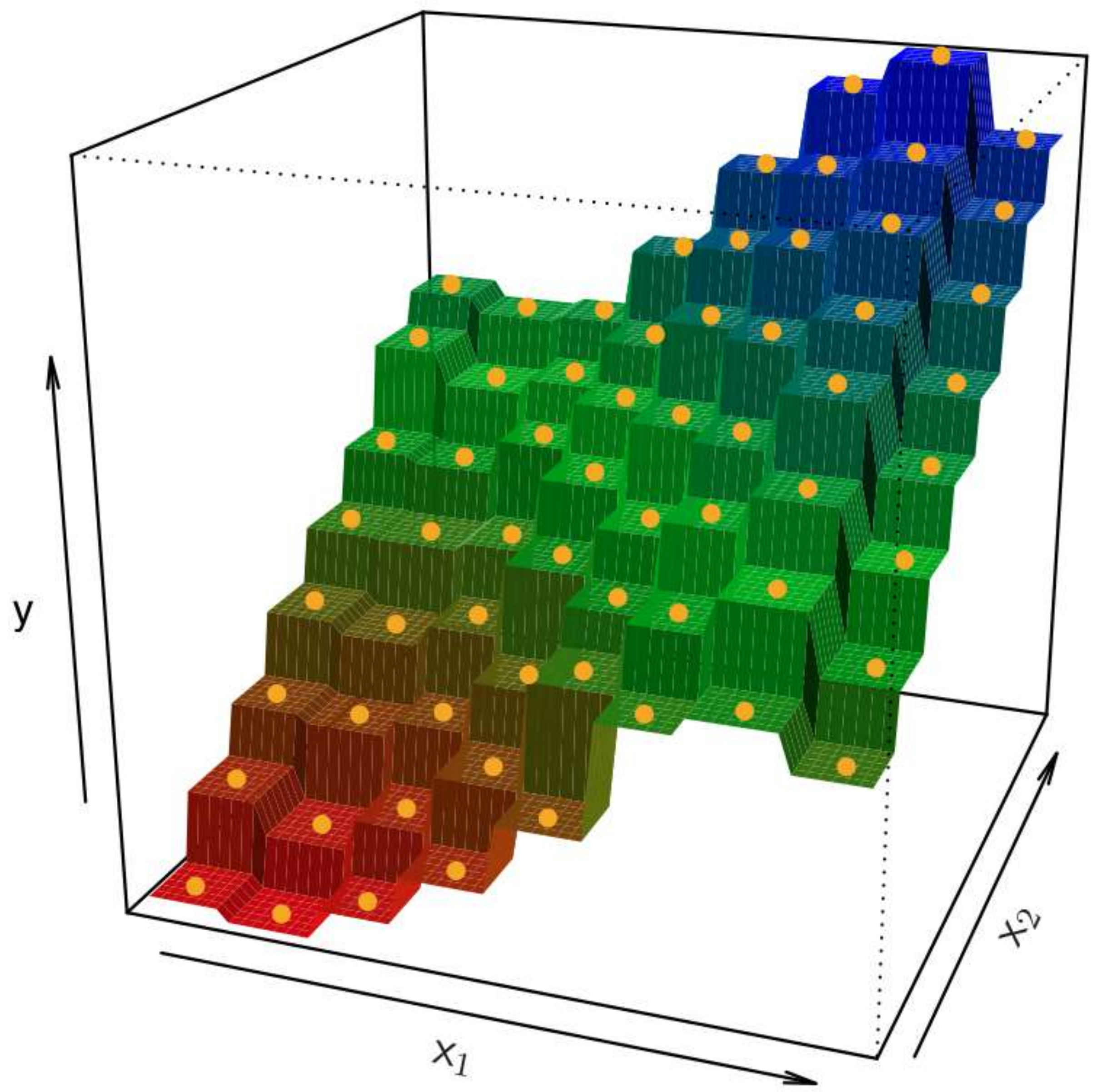
K-nearest neighbours (KNN) regression

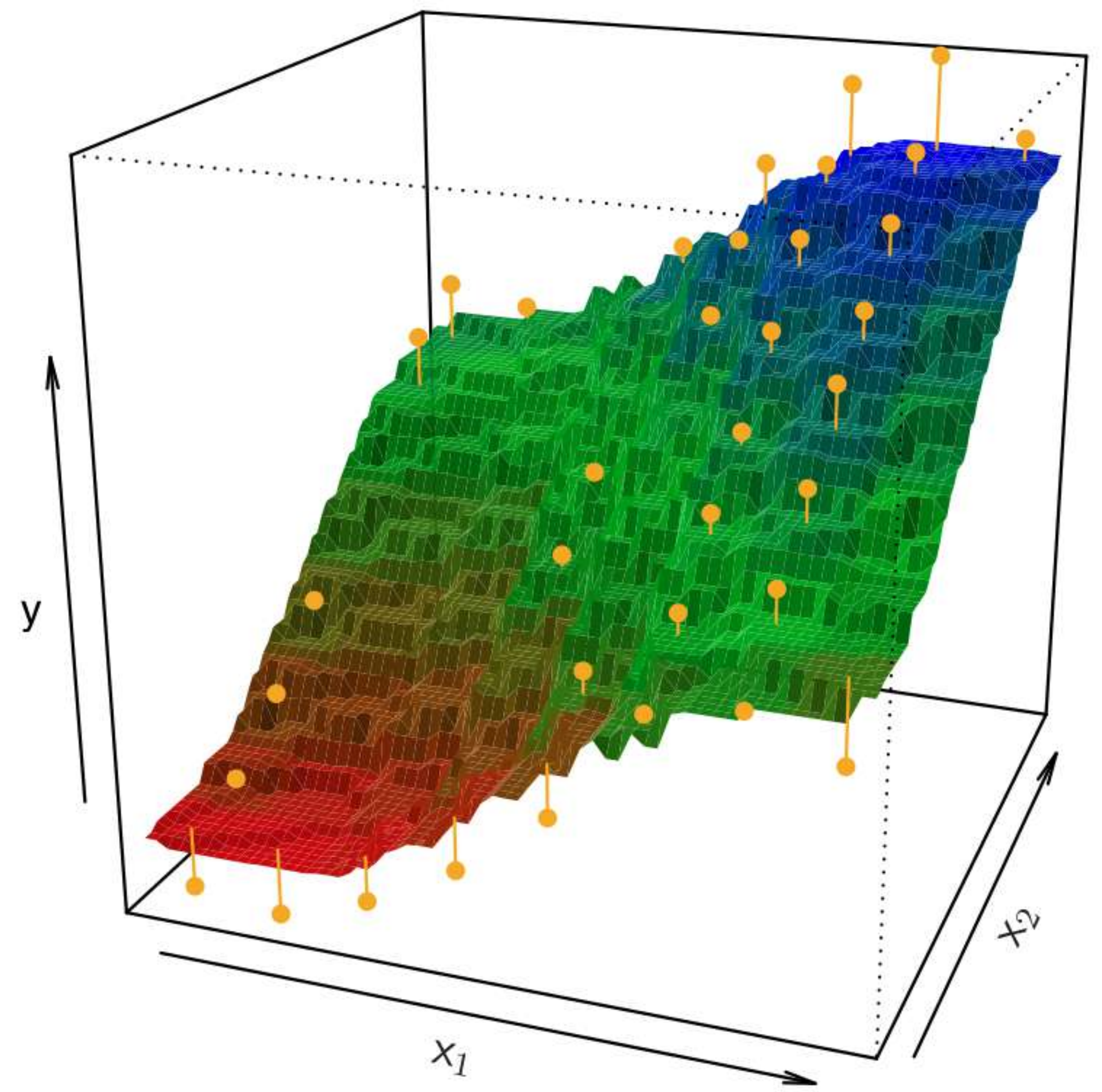
K-nearest neighbours (KNN) regression

- Parametric methods like linear regression make strong assumptions about the form of $f(X)$ (namely linearity in all predictors).
- By contrast, non-parametric methods do not assume a parametric form for $f(X)$, thereby providing a more flexible alternative for regression tasks.
- One of the simplest non-parametric regression methods is **K-nearest neighbours (KNN) regression**, which works as follows:
  - Given a value for K and a prediction point x_0, we first identify the K training observations that are closest to x_0, represented by $\mathcal{N}_0$.
  - We then estimate $f(x_0)$ using the average of all the training responses in $\mathcal{N}_0$, i.e. formally we have

$$\hat{f}(x_0) = \frac{1}{K} \sum_{x_i \in \mathcal{N}_0} y_i$$

K-nearest neighbours (KNN) regression



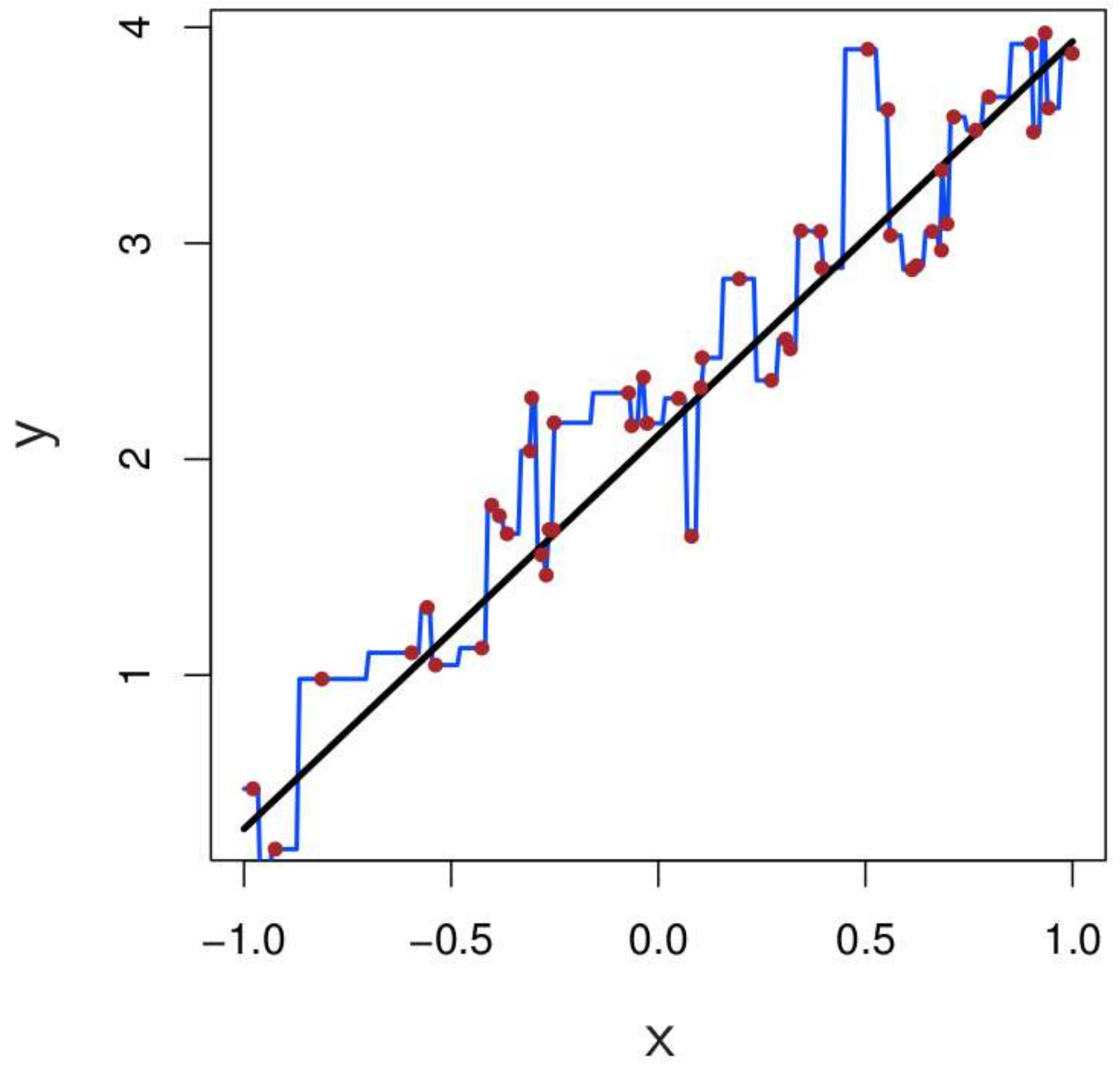


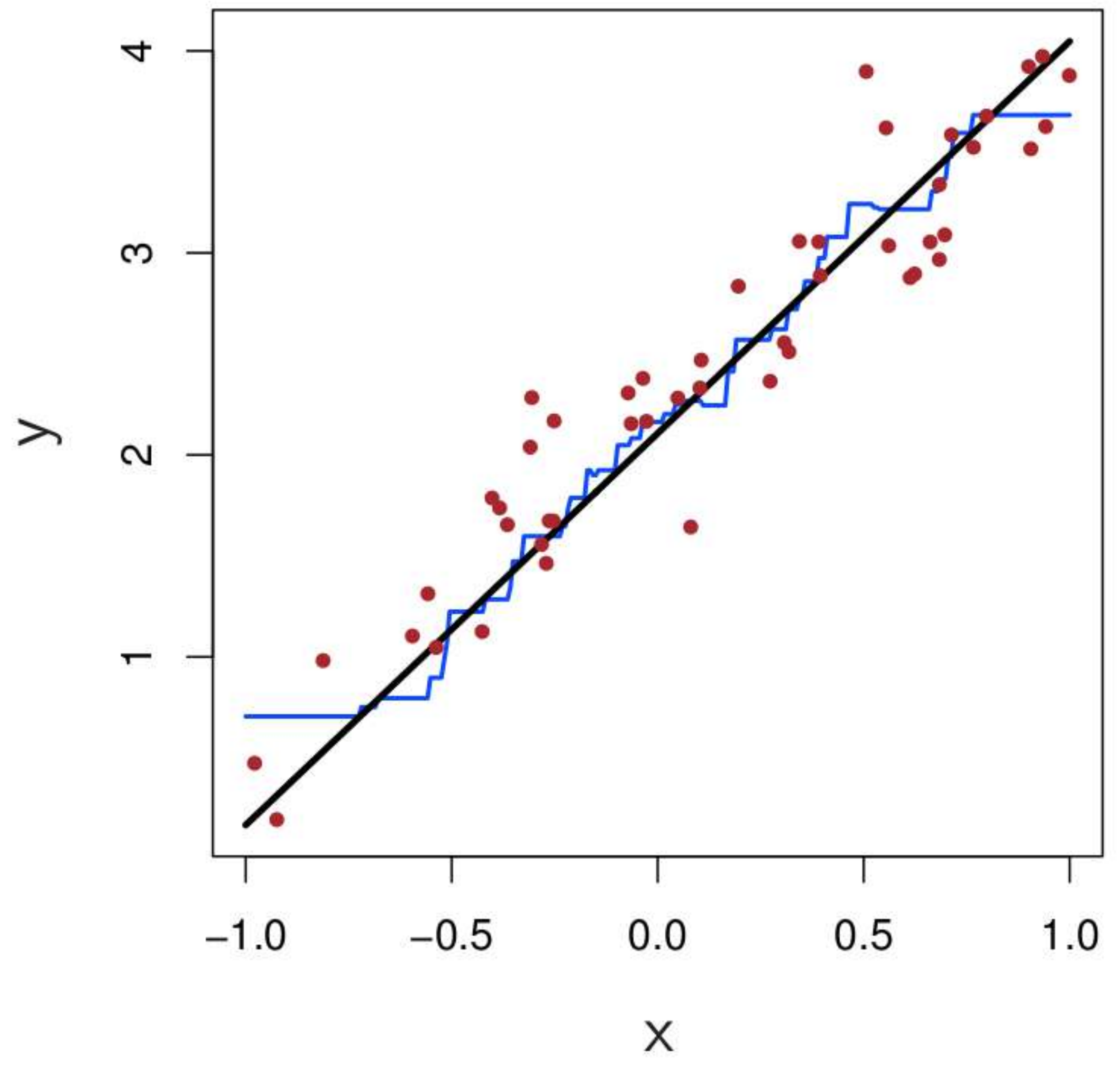
Plots of $\hat{f}(X)$ using KNN regression on a two-dimensional data set with 64 observations (orange dots). Left: $K = 1$ results in a rough step function fit. Right: $K = 9$ produces a much smoother fit.

K-nearest neighbours (KNN) regression

- The number of neighbours K is what we call a **hyperparameter**. It changes the exact behaviour of the algorithm: a small value for K provides lots of flexibility (low bias, high variance), while a large value for K provide a smoother, less variable fit (high bias, low variance).
- The optimal value for K will be different in every data set and can be determined via cross-validation, for instance.
- So, how will the performance of KNN and linear regression compare? That depends on the true form of f:
  - If f is close to linear, then linear regression will generally be better.
  - If f is highly non-linear, then KNN regression will generally be better.
  - As we do not know the true f, we have to judge which one is better based on estimates of test MSE, individually for every data set under investigation.

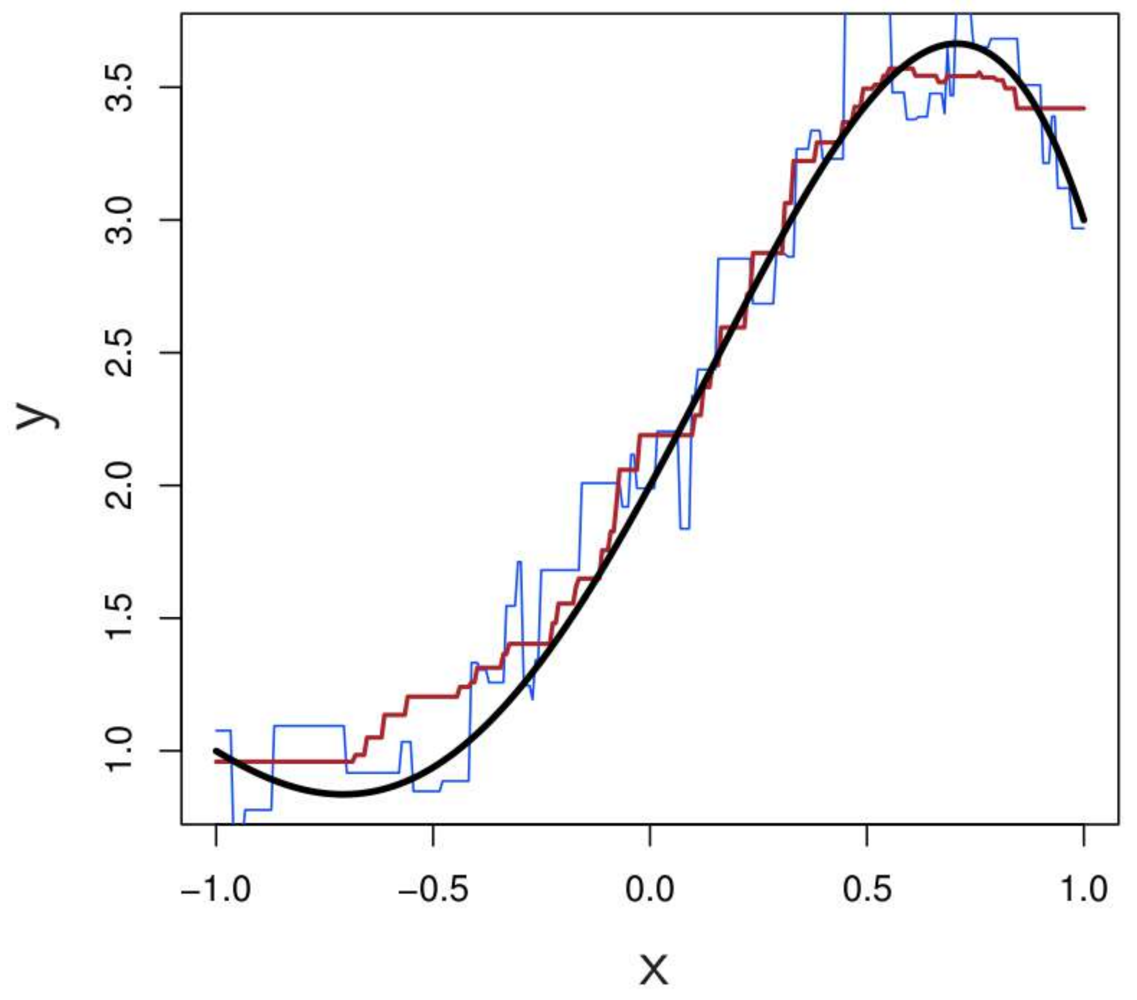
KNN vs. linear regression

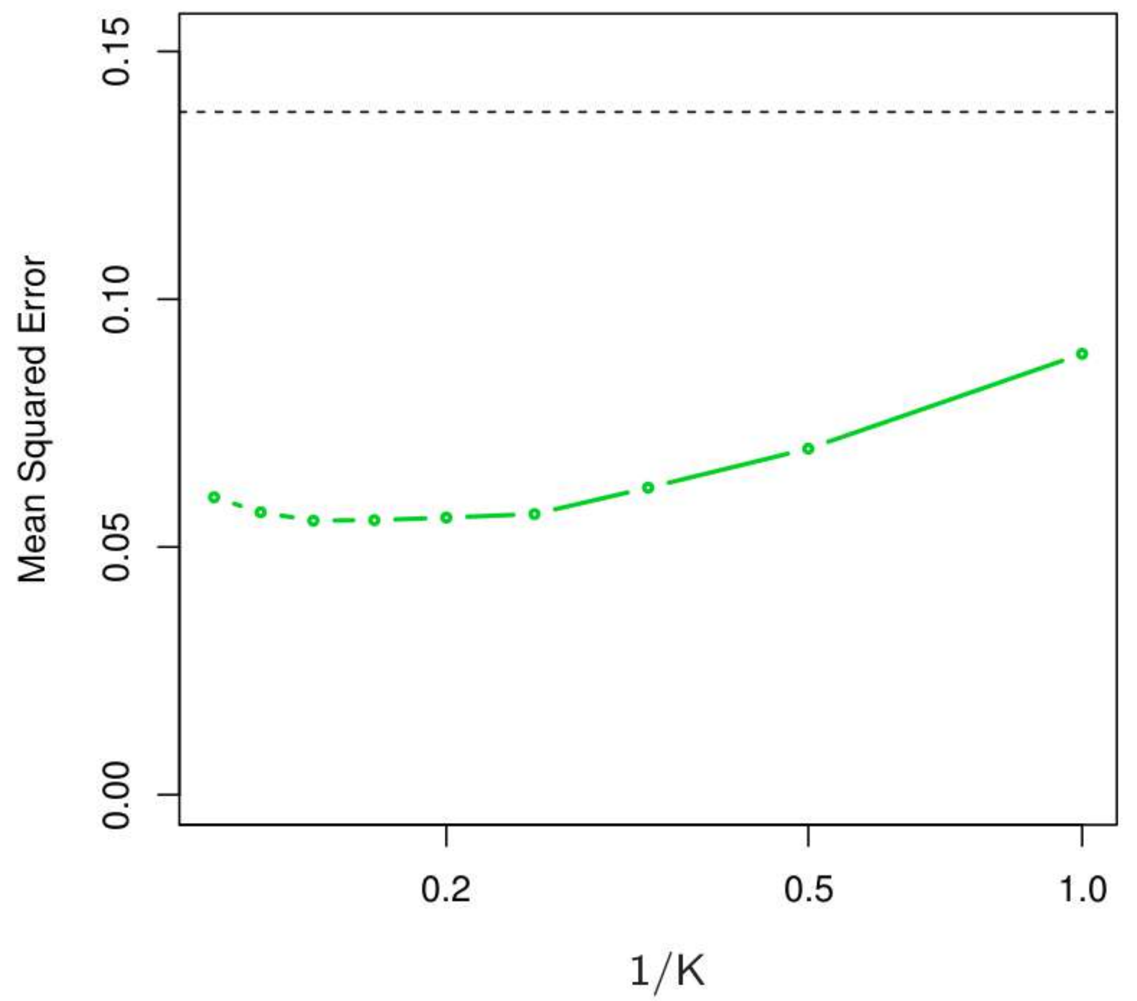




Plots of $\hat{f}(X)$ using KNN regression on a one-dimensional data set with 50 observations. Clearly, when the true relationship (black line) is linear, KNN regression is sub-optimal, regardless of whether $K = 1$ (left) or $K = 9$ (right).

KNN vs. linear regression





For a non-linear relationship between X and Y, however, KNN outperforms linear regression. On the left, we see KNN fits for $K = 1$ (blue) and $K = 9$ (red) compared to the true relationship (black). On the right, we see that the test set MSE for linear regression (horizontal black line) is significantly higher than the test set MSE for KNN with different values of K (displayed as $1/K$).

Caveats of KNN regression

- **First caveat of KNN regression**: The **curse of dimensionality**:
  - As we saw, with small p, KNN typically performs slightly worse than linear regression when f is linear, but much better if f is non-linear.
  - However, when p becomes larger (with n constant), KNN performance typically **deteriorates**.
  - This is because, in high dimensions (e.g. $p = 50$), the K observations that are nearest to x_0 may still be very far away from x_0, leading to a very poor prediction of $f(x_0)$.
- As a general rule therefore: parametric methods will tend to outperform non-parametric approaches when the number of observations per predictor (i.e. n/p) is small.

Caveats of KNN regression

- **Second caveat of KNN regression: feature scaling:**
  - Since KNN predicts a test observation by identifying the training observations that are **nearest** to it, the scale of the variables matter.
  - If all features are measured on the same scale (like in the advertising example), there is nothing to worry about.
  - However, consider the features `salary` (in USD) and `age` (in years): for the KNN, a difference of 1000 USD in salary is enormous compared to a difference of 50 years in age.
  - Consequently, all the predictions will be driven by `salary` rather than age.
- As a general rule therefore: when feature scales differ, perform **variable standardization** before doing KNN regression.

KNN regression in R

In R, KNN regression can be achieved with the help of the `FNN` package. It provides a function called `knn.reg`, which requires a data frame of features, a vector of responses and a value for K to estimate a KNN model. Let's apply that to our advertising example, say for $K = 3$:

```r
library(FNN)

knn_ad <- knn.reg(train = advertising[, 1:3], y = advertising$sales, k = 3)
knn_ad
```

```
PRESS =  373.3867
R2-Predict =  0.9310732
```

- As there are no coefficients to interpret or inferential statistics to analyse, the pure model fit is not that interesting.
- Instead, the value of KNN regression lies mostly in prediction.
- Let's therefore see how KNN regression with different values of K performs in cross-validation compared to the best-performing linear regression model from earlier.

KNN regression in R

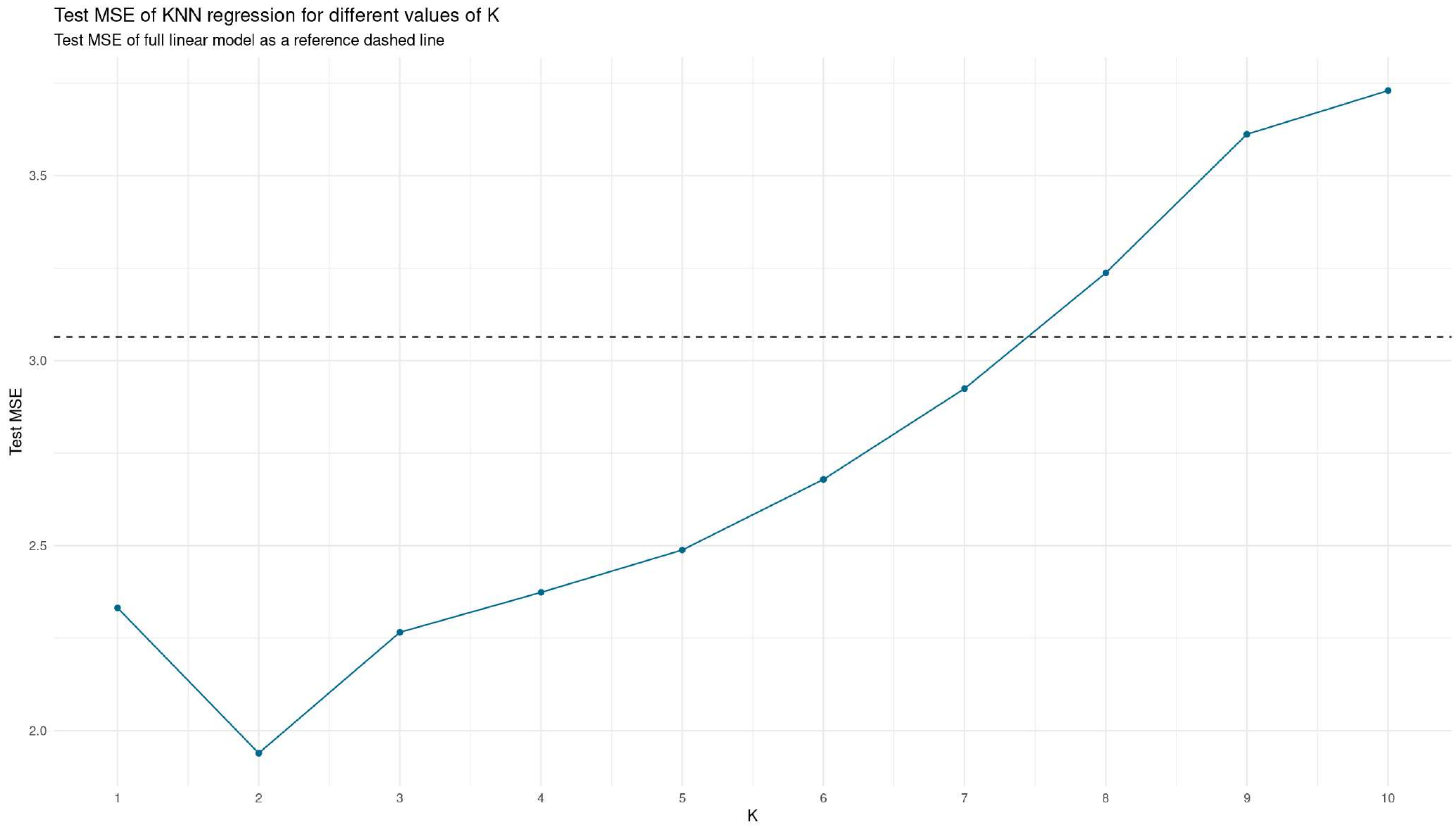
With the `knn.reg` function, we can do fitting and predicting in one function call. This requires passing the prediction features as the `test` argument to the function. We use this functionality in our existing cross-validation scheme:

```r
n_neighbours <- 1:10 # number of neighbours considered
mses_knn <- matrix(NA, K, length(n_neighbours))
colnames(mses_knn) <- sprintf("K = %d", n_neighbours)

for(k in 1:K){
  preds <- lapply(n_neighbours, function(x) knn.reg(train = subset(advertising, fold != k, 1:3),
                                                    test = subset(advertising, fold == k, 1:3),
                                                    y = advertising$sales[advertising$fold != k],
                                                    k = x)$pred)
  mses_knn[k, ] <- colMeans((do.call(cbind, preds) - advertising$sales[advertising$fold == k])^2)
}
round(colMeans(mses_knn), 2)
```

```
 K = 1  K = 2  K = 3  K = 4  K = 5  K = 6  K = 7  K = 8  K = 9 K = 10
  2.33   1.94   2.27   2.37   2.49   2.68   2.92   3.24   3.61   3.73
```

The best test MSE is achieved by setting $K = 2$. The resulting model has a test MSE of 1.94, which is also considerably better than the best-performing linear model, which had test MSE 3.06.

KNN regression in R



Classification

An overview of classification

- So far, we have dealt with situations in which the response variable Y was **quantitative**. However, often it is **qualitative / categorical**.
- Predicting a qualitative response is also referred to as **classifying** that observation, as it involves assigning the observation to a class / category.
- A **classifier** is an algorithm that maps input data (i.e. the predictors) to a category. Some of them first predict a probability for each of the categories. In this sense, they behave like regression methods.
- In this first part, we will discuss the following two simple classifiers:
  - Logistic regression
  - KNN classification

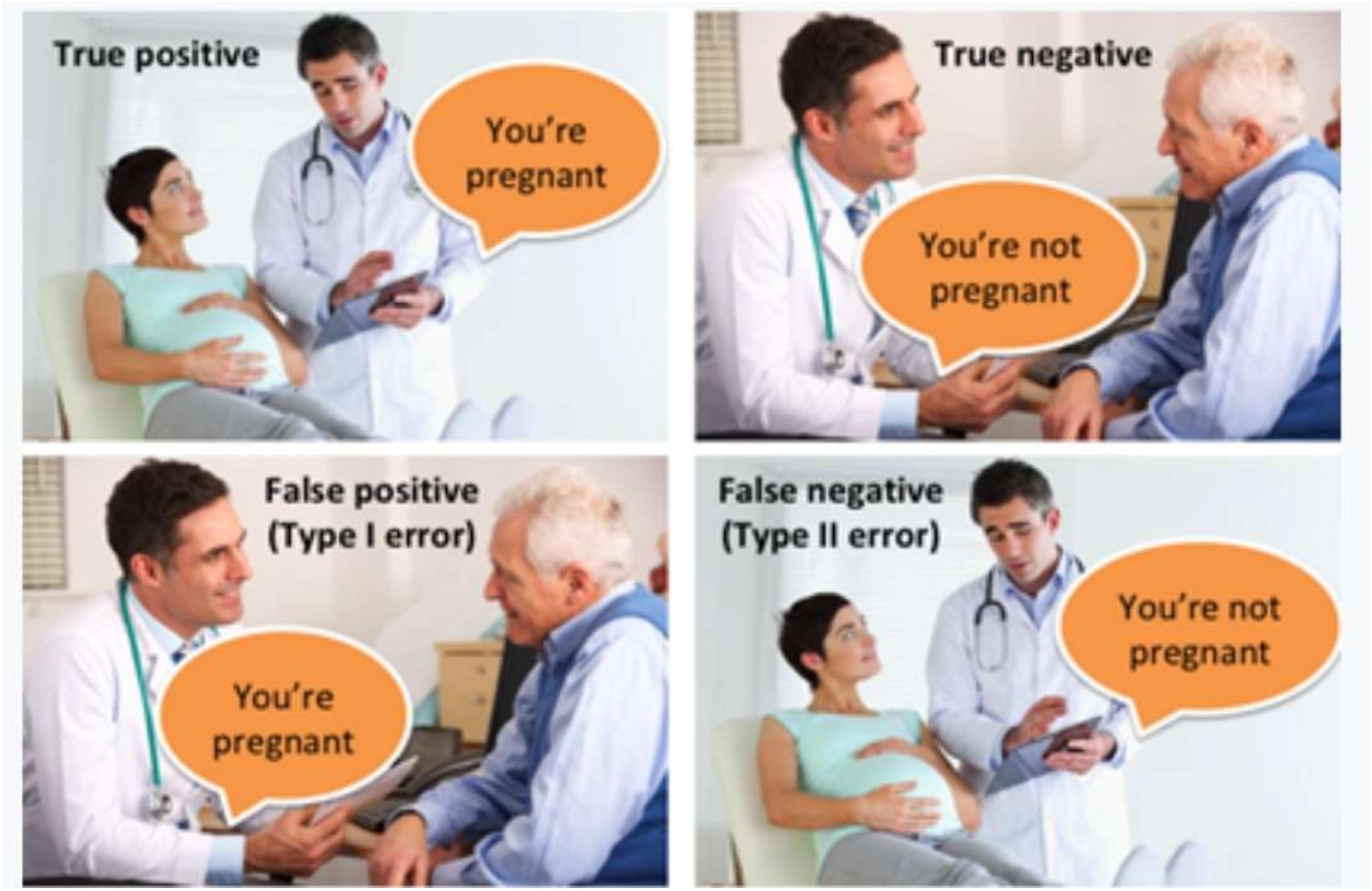
Evaluation of classifiers

- In the introduction, we already saw that a common way of evaluating classifiers is the **error rate**, i.e. the proportion of misclassified observations.
- However, there are many more ways to evaluate classifiers. One typically looks at the joint distribution of observed response and predicted response.
- For binary classification, we can illustrate this distribution with the help of a so-called **confusion matrix**:

	observed/true = 0	**observed/true = 1**
predicted = 0	true negatives (TN)	false negatives (FN)
predicted = 1	false positives (FP)	true positives (TP)

- For an extended version of this table, see here.

Evaluation of classifiers



Evaluation of classifiers

- From this confusion matrix, we can compute several metrics:
  - **Accuracy**: percentage of correctly classified observations:

$$accuracy = (TN + TP)/(TN + TP + FN + FP)$$

  - **Recall**: percentage of correctly classified positives:

$$recall = TP/(FN + TP)$$

  - **Precision**: percentage of correctly classified predicted positives:

$$precision = TP/(FP + TP)$$

  - **F1 score**: harmonic mean of precision and recall

$$F1 = 2 \cdot recall \cdot precision/(recall + precision)$$

Evaluation of classifiers

- The choice of the evaluation measure is application- and data-dependent. One typically evaluates the relative costs of **false negatives** and **false positives**.
- Consider the following example:
  - Say, we have built a classifier that predicts whether someone has cancer based on an MRI image.
  - **False positive**: telling someone they have cancer even though they do not.
  - **False negative**: telling someone they don’t have cancer, although they do.
  - If the first is considered more problematic, optimize for **precision**.
  - If the second is considered more problematic, optimize for **recall**.
  - If both are important, optimize for **F1** or **accuracy**.
- However, the sole use of accuracy can also be misleading…

Using accuracy for evaluation

- Suppose we have a data set of 100 credit card customers, of which 5 have defaulted on their credit card debt. We want to build a classifier that tells us whether someone will default or not.
- After training, the classifier produced the following confusion matrix:

	observed/true = 0	**observed/true = 1**	**Total**
predicted = 0	94	4	98
predicted = 1	1	1	2
Total	95	5	100

  - Our classifier achieves **95% accuracy (5% error rate)** on the training data!
  - But recall is only 1/5, only one default was correctly identified.
  - Precision is only 1/2, only one of the two default predictions was correct.
- **Accuracy can be misleading for unbalanced class distributions!**

Logistic Regression

Click data set

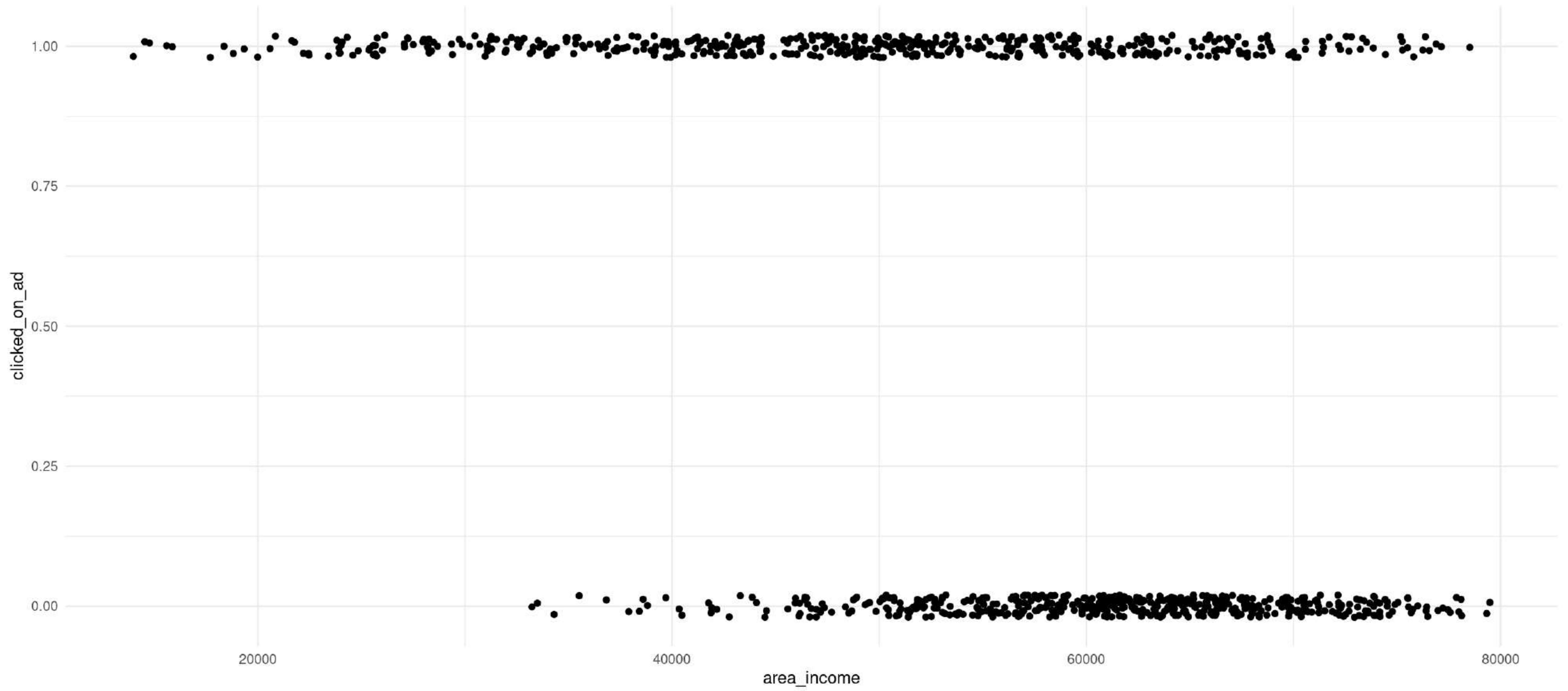
- Advertising companies want to target advertisements to users based on a user's **likelihood to click**.
- For this purpose, we are using a data set from Kaggle, which contains information about the behaviour of 1000 users.
- A total of 9 variables are available for each user:
  - The binary variable **clicked_on_ad** indicates whether the user clicked the ad or not. This will be our response variable.
  - Personal information such as age, sex, time spent on the website.
  - Other information, such as average annual income of the area where the user resides, city and country.

```r
click <- read.csv("data/click.csv")
```

Simple logistic regression model

Let’s visualize the relationship between `area_income` and the response `clicked_on_ad` in a scatter plot (with a slight jitter):

Plot | Code



Simple logistic regression model

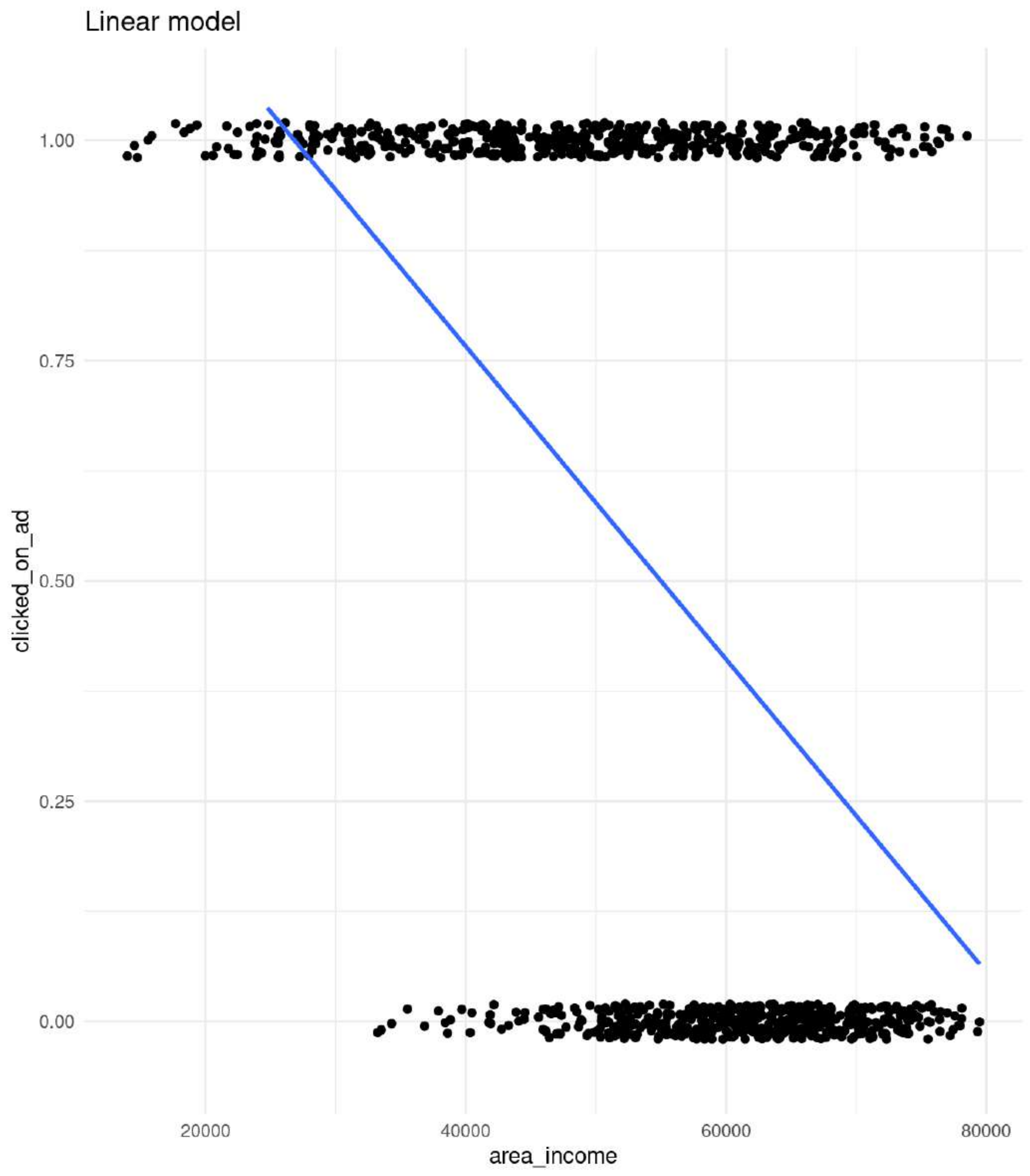
- Now, of course, we could simply lay a linear regression line into this scatter plot. This would amount to estimating a regression for the probability of a click ($Y = 1$) using area income as the explanatory variable X:

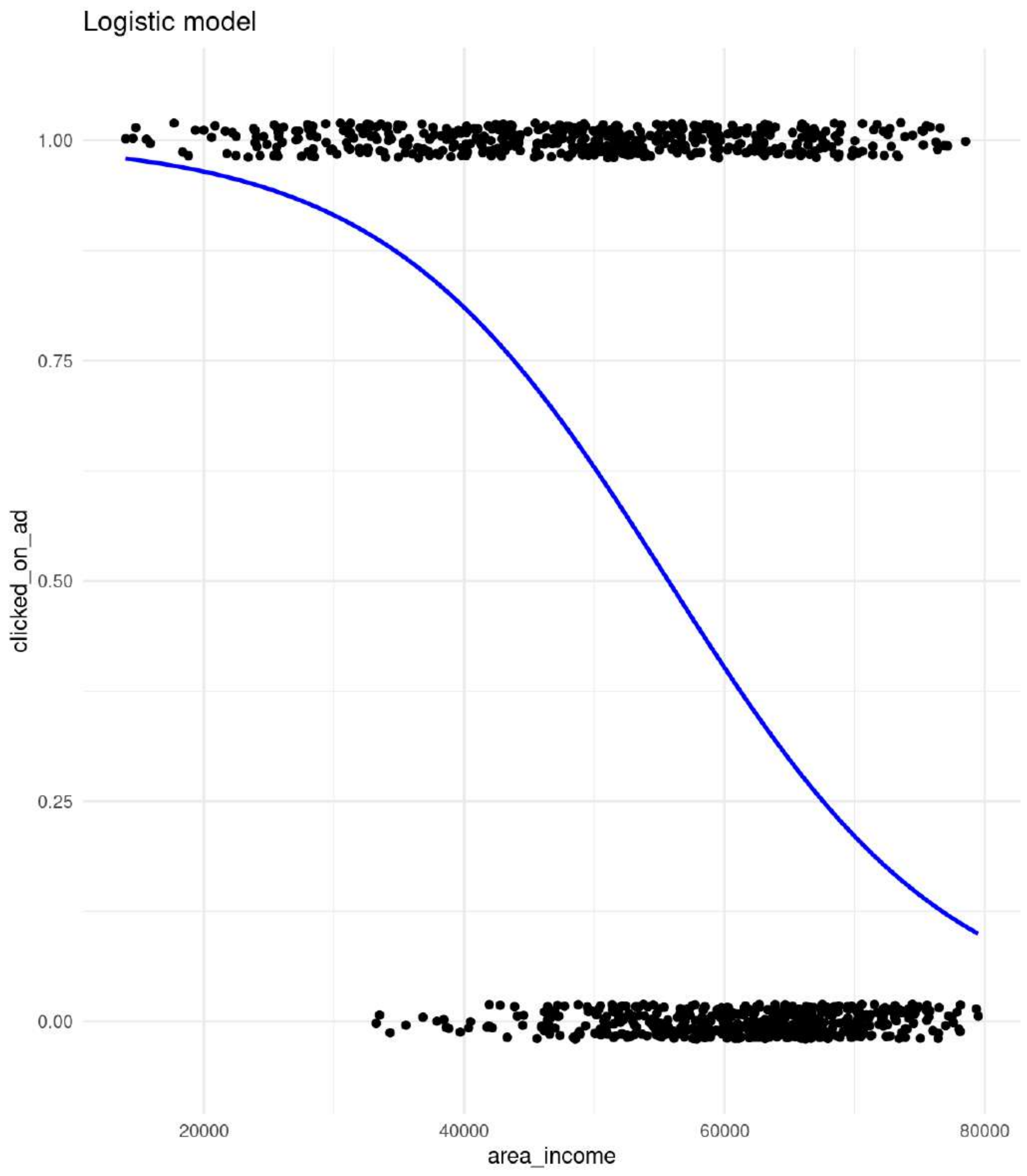
$$p_{\text{click}} = P(Y = 1|X) = \beta_0 + \beta_1 X + \epsilon$$

- However, this would not be appropriate as the possible values lie between $-\infty$ and ∞, but a probability should be between 0 and 1.
- So, we need to transform the right-hand side in order to ensure that it lies between 0 and 1. In **logistic regression**, this transformation is given by the so-called **logistic function**:

$$p_{\text{click}} = P(Y = 1|X) = \frac{e^{\beta_0 + \beta_1 X}}{1 + e^{\beta_0 + \beta_1 X}}$$

Linear vs. logistic regression model





Interpretation of simple logistic regression

- Note that with some re-arrangement of the terms, the simple logistic regression model can also be written as

$$\log\left(\frac{p_{\text{click}}}{1-p_{\text{click}}}\right) = \beta_0 + \beta_1 X$$

- The left-hand side of this equation are the so-called **log odds**. In logistic regression, we model the log odds of the success probability as a **linear function** of the predictors. This implies:
  - Increasing X by one unit changes the log odds by β_1.
  - Equivalently, it **multiplies** the odds by e^{β_1}.
  - However, because the relationship between p_{click} and X is not a straight line, β_1 does **not** correspond to the change in p_{click} associated with a one-unit increase in X. This is a difference to linear regression.

Multiple logistic regression

- Given multiple covariates $X_1, X_2, \ldots, X_p$, we can easily generalize the simple logistic regression model to the following:

$$p = P(Y = 1|X_1, \ldots, X_p) = \frac{e^{\beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p}}{1 + e^{\beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p}}$$

- Again, we model log odds as a linear function of the predictors:

$$\log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p$$

  - β_0 gives the baseline log odds for $Y = 1$ when $X_1 = X_2 = \ldots = X_p = 0.$
  - A one unit increase in covariate X_j changes the log odds for $Y = 1$ by β_j, keeping all other covariates constant (**ceteris paribus**).

Multiple logistic regression

- In logistic regression, the linear combination of the covariates $\beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p$ is called the **linear predictor**.
- It has the following interpretation:
  - If $\beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p = 0$, then $P(Y = 1|X_1, \ldots, X_p) = 0.5$, i.e. $Y = 1$ and $Y = 0$ are equally likely.
  - If $\beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p > 0$, then $P(Y = 1|X_1, \ldots, X_p) > 0.5$, i.e. $Y = 1$ is more likely.
  - If $\beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p < 0$, then $P(Y = 1|X_1, \ldots, X_p) < 0.5$, i.e. $Y = 0$ is more likely.
- In practice, the coefficients $\beta_0, \beta_1, \ldots, \beta_p$ are **unknown** and must be estimated on the basis of available training data.
- Estimation no longer happens with OLS, but with **maximum likelihood (ML)**, a very common procedure in statistics.

Statistical inference in logistic regression

- Broadly speaking, the idea of **ML estimation** is to choose parameters $\beta_0, \beta_1, \ldots, \beta_p$ such that the probability of observing the data we observed is **maximized**. The details are beyond the scope of this course.
- Fortunately, R has efficient algorithms for ML estimation of logistic regression models, which we will soon see how to use.
- Besides estimating the regression coefficients, ML estimation allows us to perform inferential statistics akin to the linear regression case:
  - Hypothesis tests for $H_0 : \beta_j = 0$ vs. $H_1 : \beta_j \neq 0$.
  - Confidence intervals for regression coefficients.
  - Goodness-of-fit measures similar to the R^2.
- However, when using logistic regression as a classification algorithm in machine learning, we are mostly interested in **prediction**, not **inference**.

Making predictions

- Once the coefficients have been estimated, we can compute the probability for $Y = 1$ for every combination of values of the predictors.
- We get $\hat{p}(x_1, \ldots, x_p) = \hat{P}(Y = 1 | X_1 = x_1, \ldots, X_p = x_p)$ by simply plugging in the coefficient estimates $\hat{\beta}_0, \ldots, \hat{\beta}_p$ into the logistic regression formula:

$$\hat{p}(x_1, \ldots, x_p) = \frac{e^{\hat{\beta}_0 + \hat{\beta}_1 x_1 + \ldots + \hat{\beta}_p x_p}}{1 + e^{\hat{\beta}_0 + \hat{\beta}_1 x_1 + \ldots + \hat{\beta}_p x_p}}$$

- The prediction of the conditional response is straightforward:

$$\hat{Y} | (X_1 = x_1, \ldots, X_p = x_p) = \begin{cases} 0 & \text{if } \hat{p}(x_1, \ldots, x_p) < 0.5 \\ 1 & \text{if } \hat{p}(x_1, \ldots, x_p) \geq 0.5 \end{cases}$$

Logistic regression in R

- Let's finally see an example of how to do all this in R!
- To estimate a logistic regression in R, we need three things to pass into the function `glm` (which stands for "generalized linear model"):
  - A `data.frame` holding the variables we want included in the model, especially the binary response as a `factor` or `numeric`.
  - A `formula` that specifies the model, similar to the `lm` case.
  - A `family` that specifies the type of generalized linear model. For logistic regression, this is always equal to `binomial()`.
- So, if we wanted to use `area_income` and `age` as predictors for the response `clicked_on_ad`, we would do the following:

```r
logreg <- glm(clicked_on_ad ~ area_income + age, data = click, family = binomial())
```

Logistic regression in R

- We can then have a look at the result using the `summary` function:

```r
summary(logreg)
```

```

Call:
glm(formula = clicked_on_ad ~ area_income + age, family = binomial(),
    data = click)

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept)  9.160e-02  5.399e-01    0.17    0.865
area_income -1.033e-04  8.207e-06  -12.59   <2e-16 ***
age          1.626e-01  1.261e-02   12.90   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)
```

- While the output looks somewhat different to the linear regression case, the all-important coefficients matrix with `Estimate`, `Std. Error`, `z value` and `Pr(>|z|)` looks virtually identical.

Logistic regression in R

- Using X_1 for `area_income` (in thousand USD) and X_2 for `age`, the estimated model for the **click probability** p thus has the following form:

$$\log\left(\frac{p}{1-p}\right) = 0.0916 - 0.1033X_1 + 0.1626X_2$$

- Interpretation:
  - Ceteris paribus, increasing the `area_income` by 1000 USD **decreases** the log odds for clicking on the ad by 0.1033 on average.
  - Ceteris paribus, increasing the `age` by one year **increases** the log odds for clicking on the ad by 0.1626 on average.
- The regression output also shows that both of these variables are **highly significant** with p-values very close to zero. Both `area_income` and `age` seem to be important for predicting whether or not a user will click on the ad.

Logistic regression in R

- In general, there is a high degree of consistency in how we can operate on `glm` objects compared to `lm` objects. For example, many **accessor functions** work in the same way as before:

```r
head(residuals(logreg))
```

```
         1          2          3          4          5          6 
-0.9342594 -0.5191394 -0.5391085 -0.8425931 -0.5408458 -0.4287287 
```

```r
head(fitted(logreg))
```

```
        1         2         3         4         5         6 
0.3536540 0.1260681 0.1352536 0.2988136 0.1360644 0.0878074 
```

```r
coef(logreg)
```

```
  (Intercept)   area_income           age 
 0.0915995255 -0.0001032967  0.1626462670 
```

```r
confint(logreg)
```

```
                    2.5 %        97.5 %
(Intercept) -0.9649226490  1.154141e+00
area_income -0.0001198858 -8.768127e-05
age          0.1386819426  1.881543e-01
```

- Due to the different scales involved (probabilities vs. log odds), some of these functions come with additional options, as we shall now see for **prediction**.

Logistic regression in R – Prediction

- Using our model, what do we estimate the probability that a 29-year old living in an area with an average annual income of 50,000 USD clicks on the ad to be?
- Let's simply plug in to our equation for the log odds:

$$\log\left(\frac{p}{1-p}\right) = 0.0916 - 0.1033 \cdot 50 + 0.1626 \cdot 29 = -0.358$$

- Clearly, that is a log odds ratio and **not a probability**. To turn this into a probability, we need to plug into the other formula for logistic regression:

$$p = \frac{e^{0.0916-0.1033\cdot 50+0.1626\cdot 29}}{1+e^{0.0916-0.1033\cdot 50+0.1626\cdot 29}} = \frac{e^{-0.358}}{1+e^{-0.358}} \approx 0.41$$

- We predict that this person will click on the ad with a probability of 41%!

Logistic regression in R – Prediction

- Ignoring the rounding differences, both of these types of predictions are available through the `predict` function in R:
  - We get the linear predictor, i.e. the log odds for the click, by setting `type` to `link` (which is the default):

```r
new_click <- data.frame(area_income = 50000, age = 29)
predict(logreg, newdata = new_click, type = "link")
```

```
         1 
-0.3564913 
```

  - In the machine learning context, we are typically only interested in the probability. This we get by setting `type` to `response`:

```r
predict(logreg, newdata = new_click, type = "response")
```

```
        1 
0.4118092 
```

- With the ability to predict new observations secured, let's evaluate the performance of our model as a classifier using 5-fold cross validation!

Logistic regression in R – Evaluation

- Again, we randomly assign each observation to one of 5 folds:

```r
n <- nrow(click)
K <- 5
click$fold <- sample(rep(1:K, length.out = n))
```

- This time, we save the class predictions on each hold-out fold together into one variable called `class_preds_lr`. This will then allow us to evaluate the out-of-sample performance of the classifier:

```r
click$class_preds_lr <- NA

for(k in 1:K){
  m <- glm(clicked_on_ad ~ area_income + age, data = subset(click, fold != k),
           family = binomial())
  prob_preds <- predict(m, newdata = subset(click, fold == k), type = "response")
  click$class_preds_lr[click$fold == k] <- ifelse(prob_preds >= 0.5, 1, 0)
}
```

  - We classify an observation as a **click** if its probability is at least 0.5.
  - We classify an observation as **no click** if its probability is smaller than 0.5.

Logistic regression in R – Evaluation

- Now, we can compute the out-of-sample **confusion matrix** of our classifier:

```r
table(click$class_preds, click$clicked_on_ad,
      dnn = c("click_prediction", "click_truth"))
```

```
                click_truth
click_prediction   0   1
               0 431 123
               1  69 377
```

- From this, we can now calculate the aforementioned performance metrics:
  - **Accuracy:** $(431 + 377)/1000 = 0.808$. Our logistic regression model correctly predicts the click decision in 80.8% of the observations.
  - **Recall:** $377/(123 + 377) = 0.754$. Our logistic regression model correctly identifies 75.4% of all clicks on an ad.
  - **Precision:** $377/(69 + 377) = 0.845$. When our logistic regression model predicts a click, it is correct 84.5% of the time.
  - **F1 score:** $2 \cdot 0.754 \cdot 0.845/(0.754 + 0.845) = 0.797$.

Further topics on logistic regression

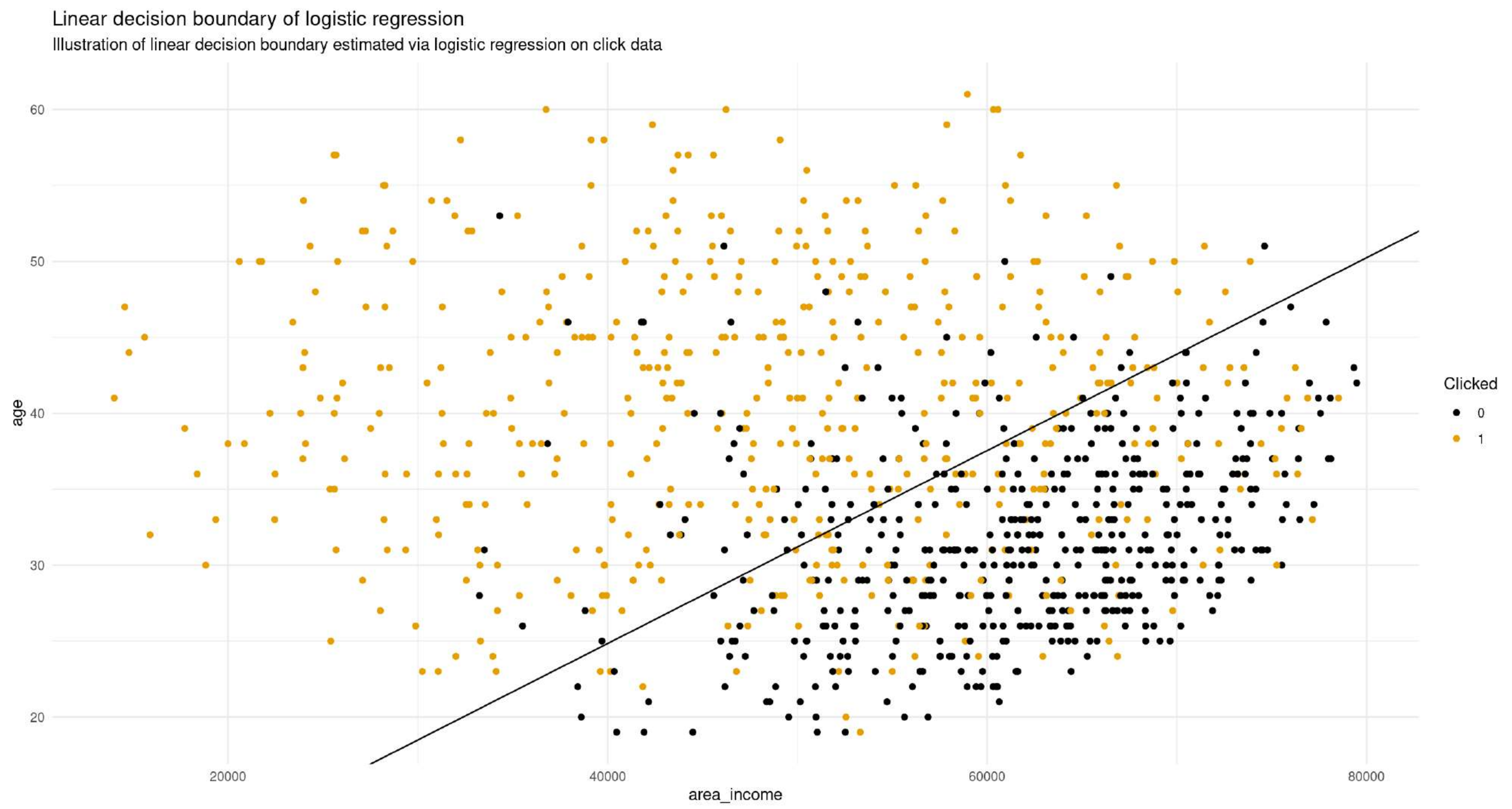
- As we saw, many concepts from linear regression generalize to logistic regression. Further examples of this include:
  - **Categorical predictors**: we can use dummy variables to encode categorical variables as predictors also in logistic regression.
  - **Variable selection**: the `step` function for forward and backward selection works equally on `glm` objects.
  - **Linearity**: logistic regression is actually linear in the sense that it models a **linear decision boundary**, i.e. the curve separating observations of different classes is a straight line. It satisfies the following equation (see slide 107):

$$\beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p = 0$$

- In our example with two predictors, we can nicely visualize the estimated decision boundary.

Linear decision boundary in logistic regression

Plot Code



K-nearest neighbours (KNN) classification

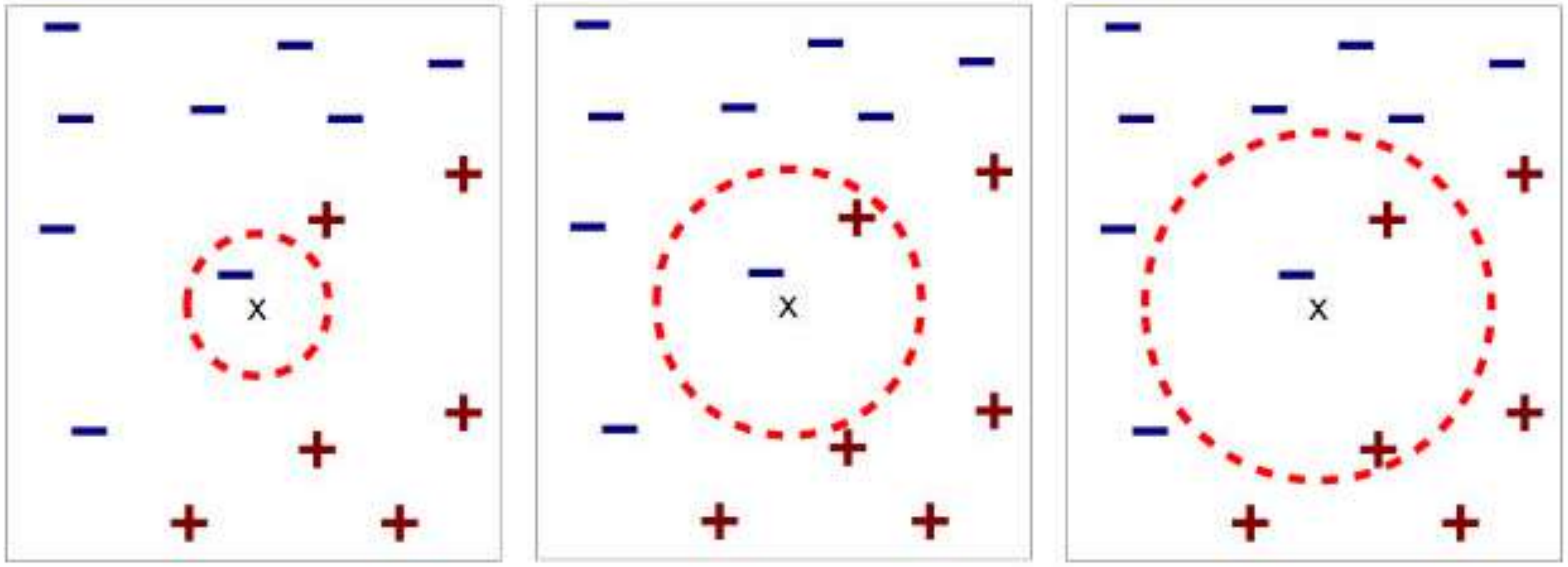
K-nearest neighbours (KNN) classification

- Similar to the regression context, a non-parametric alternative for performing classification is given by **K-nearest neighbours (KNN) classification**.
- Unlike logistic regression, KNN classification can also be used for **multi-class classification**, i.e. for when there are more than two classes.
- In principle, the procedure is straightforward:
  - Given a value for K and a prediction point x_0, we again start by identifying the K training observations that are closest to x_0, represented by $\mathcal{N}_0$.
  - We then estimate the conditional probability for class j as the fraction of points in $\mathcal{N}_0$ whose response values equal j:

$$\hat{P}(Y = j | X = x_0) = \frac{1}{K} \sum_{i \in \mathcal{N}_0} I(y_i = j)$$

K-nearest neighbours (KNN) classification

Suppose we have two classes ($+$ and $-$) and two features X_1 and X_2. The figure below illustrates how conditional probability would be estimated for a point x and $K = 1, 2, 3$ in a toy example data set:



$K = 1, \hat{P}(+|x) = 0/1 \quad K = 2, \hat{P}(+|x) = 1/2 \quad K = 3, \hat{P}(+|x) = 2/3$

KNN classification in R

- In R, KNN classification can also be achieved with the help of the `FNN` package:
  - It provides a function called `knn`, whose arguments are very similar to the `knn.reg` function we saw earlier.
  - With this function, we **have to** perform fitting and prediction in one function call. So let's run KNN classification for different values of K directly in our existing cross-validation scheme.
  - First though, we need to standardize our variables, since `area_income` is measured in USD and `age` is measured in years. To **standardize** a variable, we deduct the mean and divide by the standard deviation:

```r
click$ai_scaled <- (click$area_income - mean(click$area_income))/sd(click$area_income)
click$age_scaled <- (click$age - mean(click$age))/sd(click$age)
```

KNN classification in R

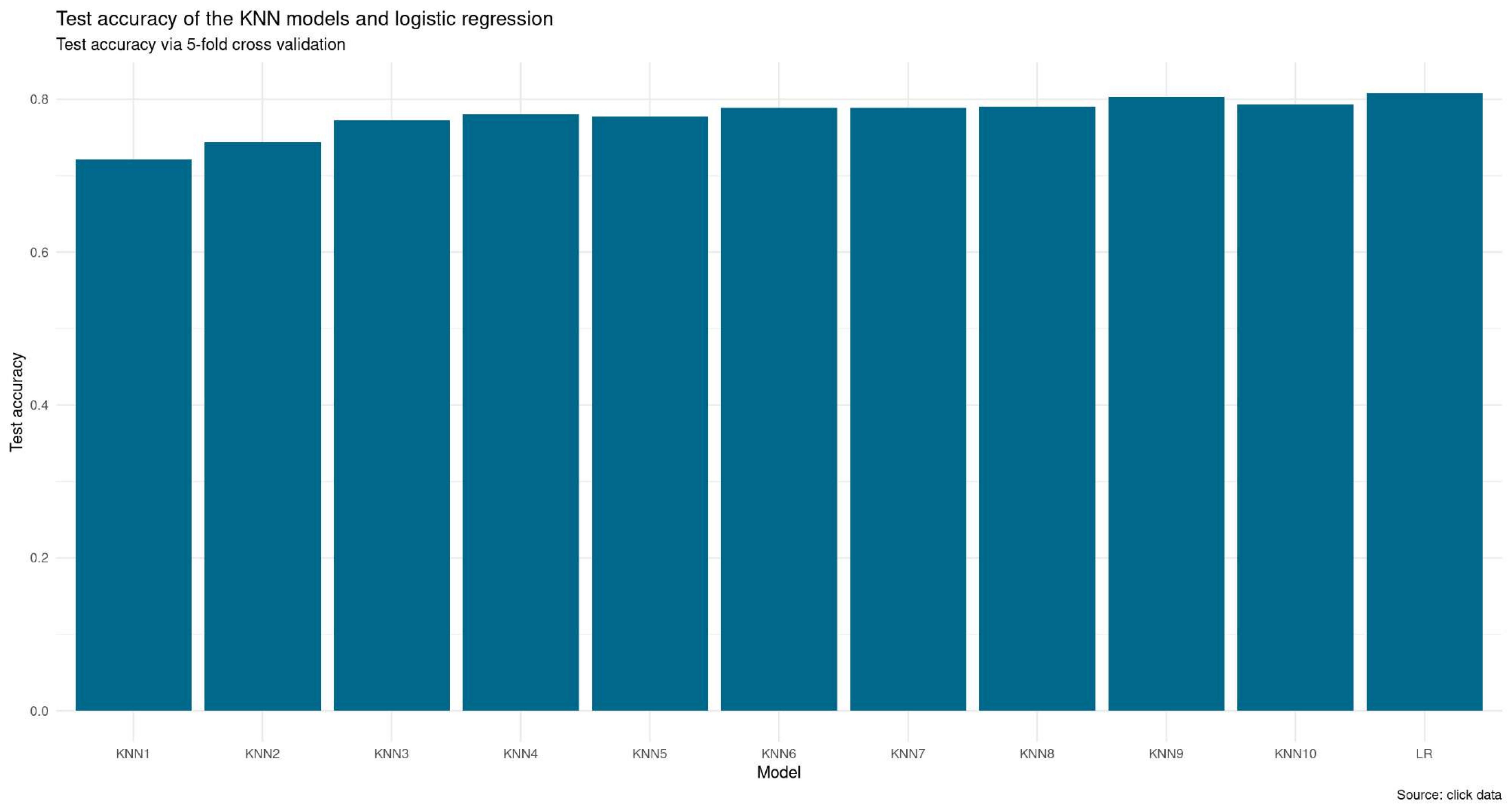
- Now, we can actually run the KNN classification for $K = 1, \ldots, 10$ and predict each fold in the 5-fold cross validation scheme from earlier:

```r
n_neighbours <- 1:10

for(nn in n_neighbours){
  knn_colname <- sprintf("class_preds_KNN%d", nn)
  click[[knn_colname]] <- factor(NA, levels = c(0, 1))
  for(k in 1:K){
    preds <- knn(train = click[click$fold != k, c("ai_scaled", "age_scaled")],
                 test = click[click$fold == k, c("ai_scaled", "age_scaled")],
                 cl = click$clicked_on_ad[click$fold != k],
                 k = nn)
    click[click$fold == k, knn_colname] <- preds
  }
}
```

- Let's compare the performance (in terms of classification accuracy on the test set) of these KNN models to the logistic regression model from earlier!

KNN classification vs. logistic regression

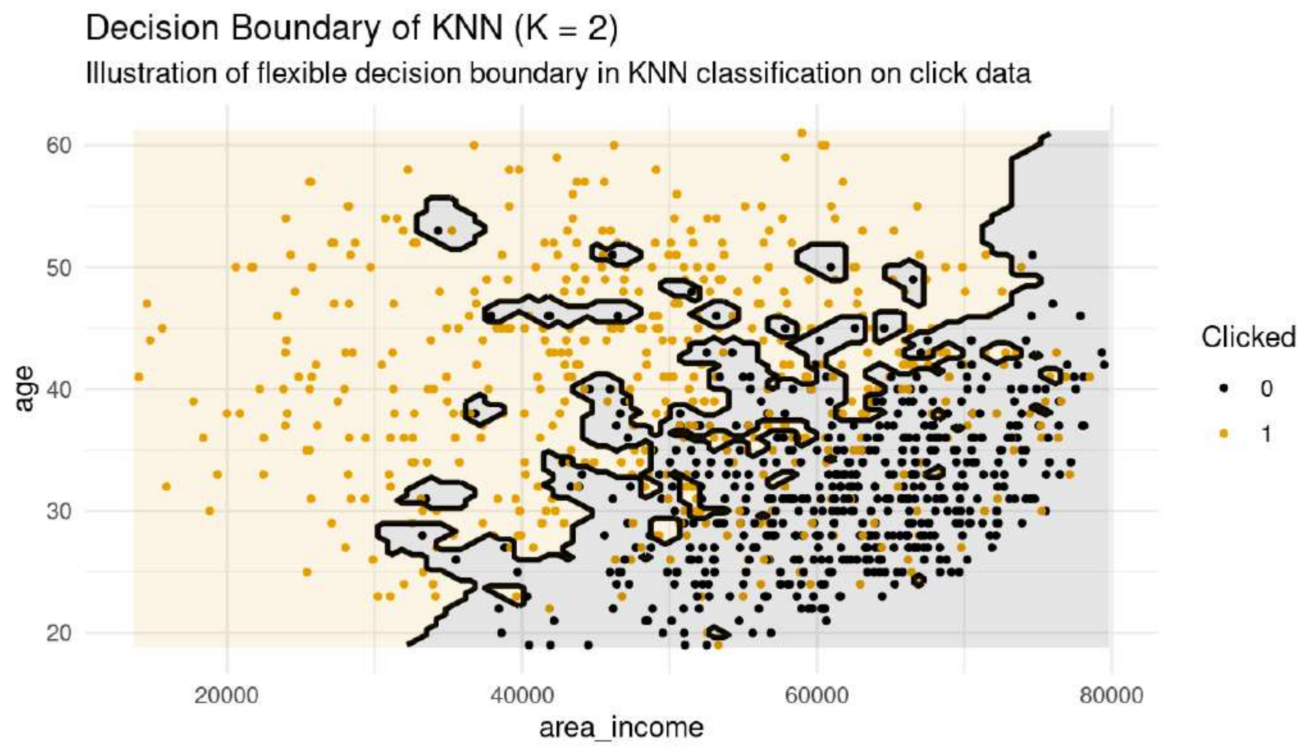
Plot Code

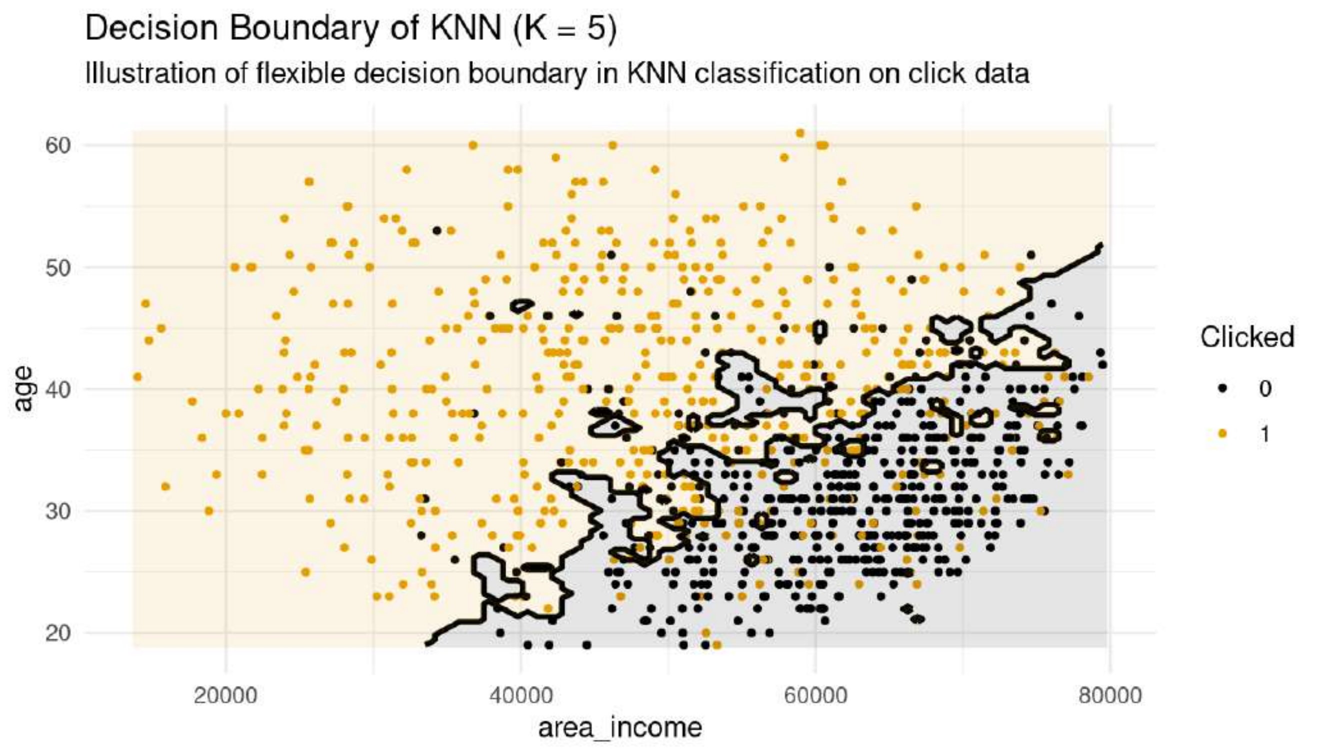


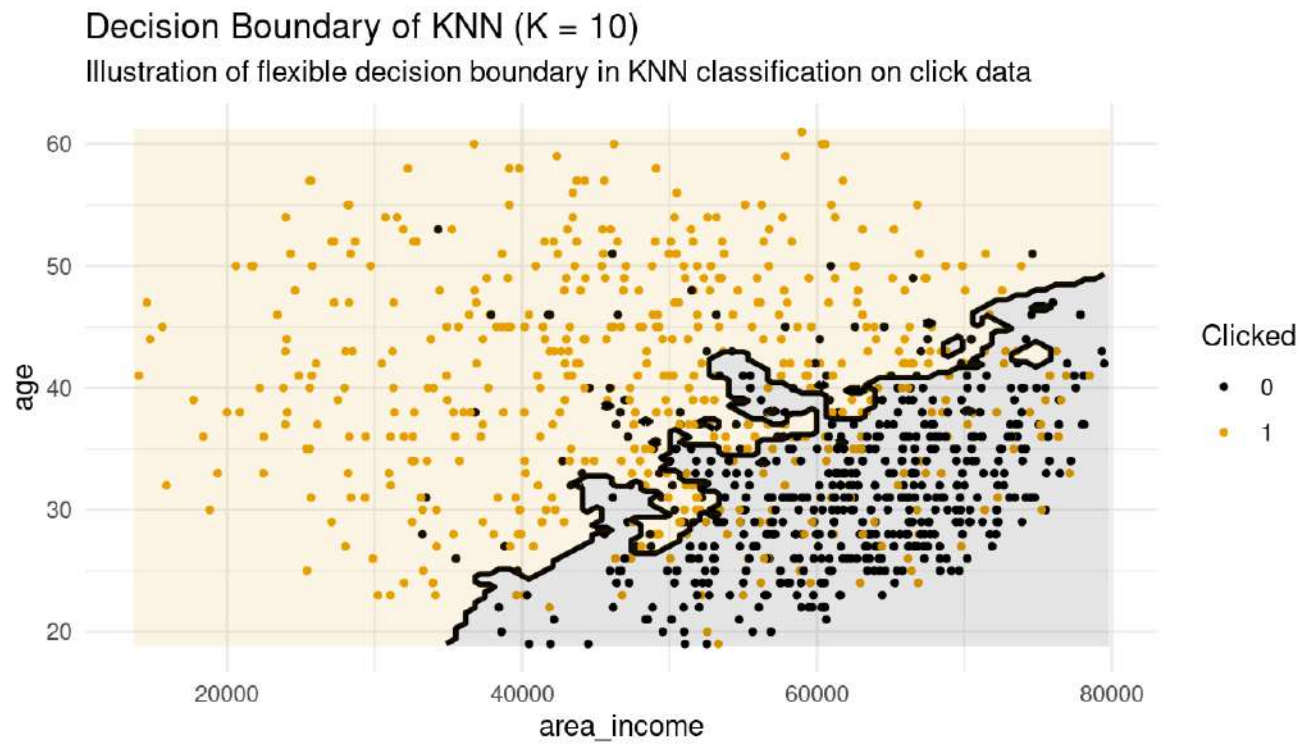
KNN classification vs. logistic regression

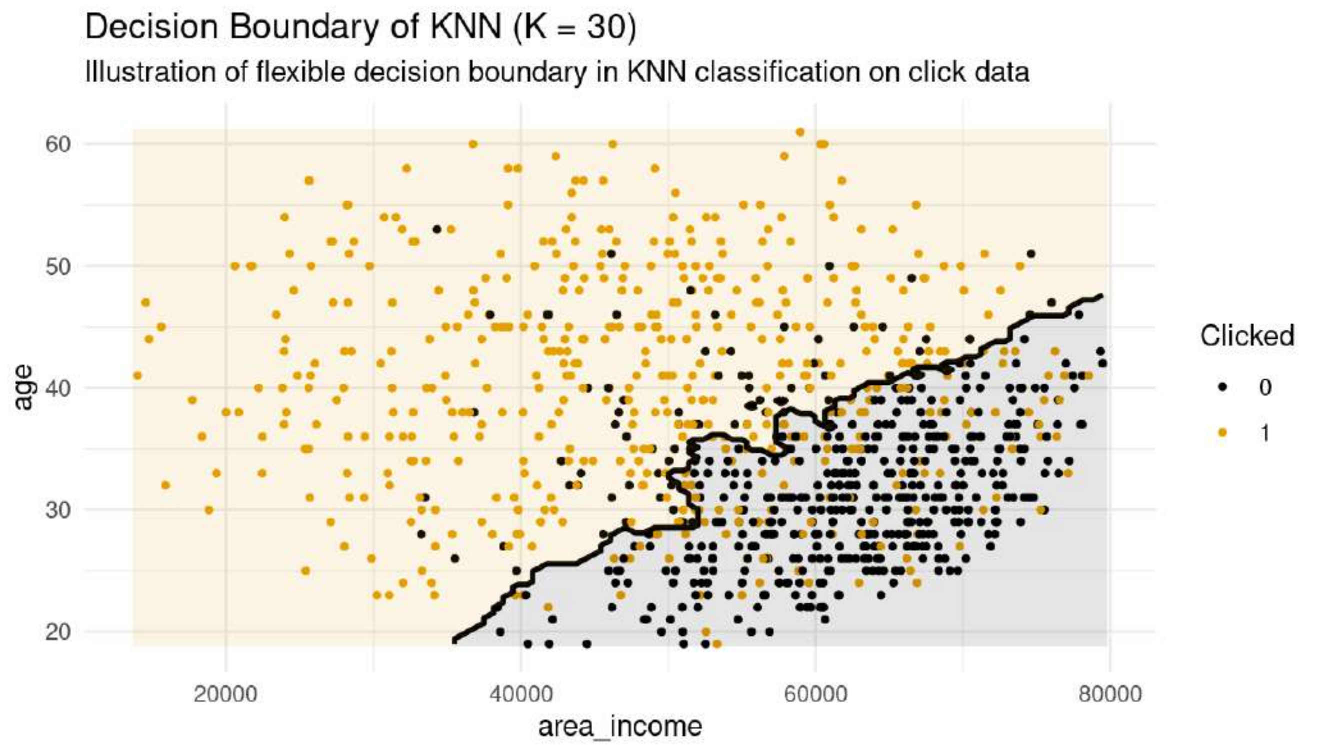
- It appears that the logistic regression model (narrowly) outperforms all KNN models in terms of **accuracy**.
- To get an even more detailed picture of the relative performances, one would look at **precision**, **recall** and **F1 score** of all presented classification approaches.
- With its non-parametric flexibility, KNN classification can in principle model any shape of decision boundary. However, in the specific case of our `click` data set, it seems that the **true decision boundary is close to linear**, so the additional flexibility does not provide any benefit.
- To illustrate this, consider the estimated decision boundaries for some of our KNN models on the next slide.

Decision boundaries in KNN classification









Tree-based methods

Tree-based methods

- So far, we know about the following **regression methods**:
  - Simple and multiple linear regression
  - Linear regression with forward and backward variable selection
  - KNN regression
- … and the following **classification methods**:
  - Simple and multiple logistic regression
  - Logistic regression with forward and backward variable selection
  - KNN classification
- Now, we will look into another class of machine learning methods called **tree-based methods**. In particular, we will discuss so-called **decision trees** that can be used for both **regression** and **classification**.

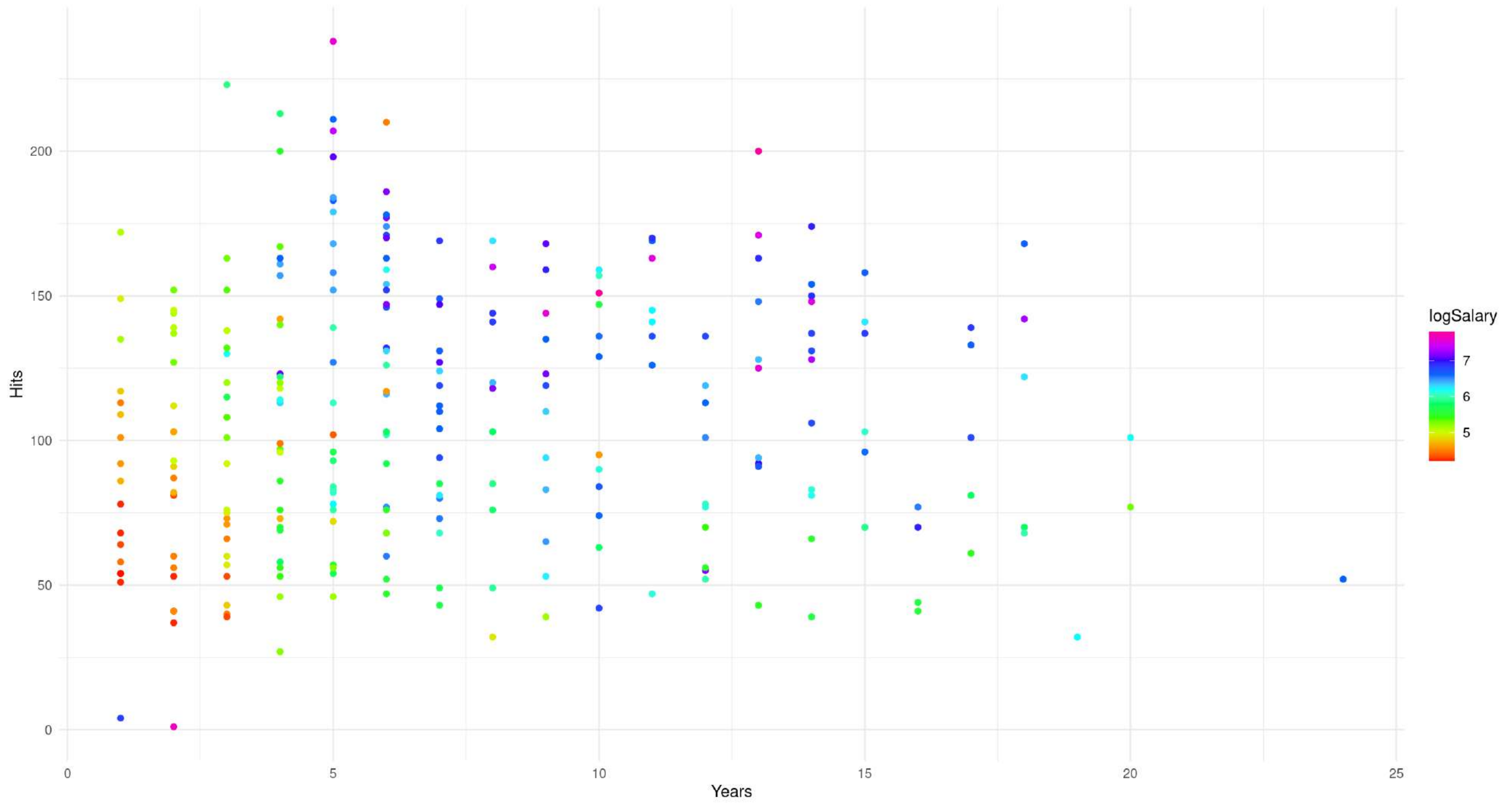
Introduction to regression trees

- When decision trees are used for regression tasks, they are referred to as **regression trees**.
- In order to motivate regression trees, we begin with a **simple example**:
  - The `Hitters` data set in the `ISLR` package contains statistics of 322 baseball players in the American MLB from the 1986 and 1987 seasons.
  - Our goal is to predict a player's `Salary` based on `Years` (number of years in the MLB) and `Hits` (number of hits he made in the previous year)
  - First, we remove observations that are missing `Salary` values and log transform the remaining salaries.

```
Hitters <- Hitters[!is.na(Hitters$Salary), ]
Hitters$logSalary <- log(Hitters$Salary)
```

Baseball salary data

Plot Code

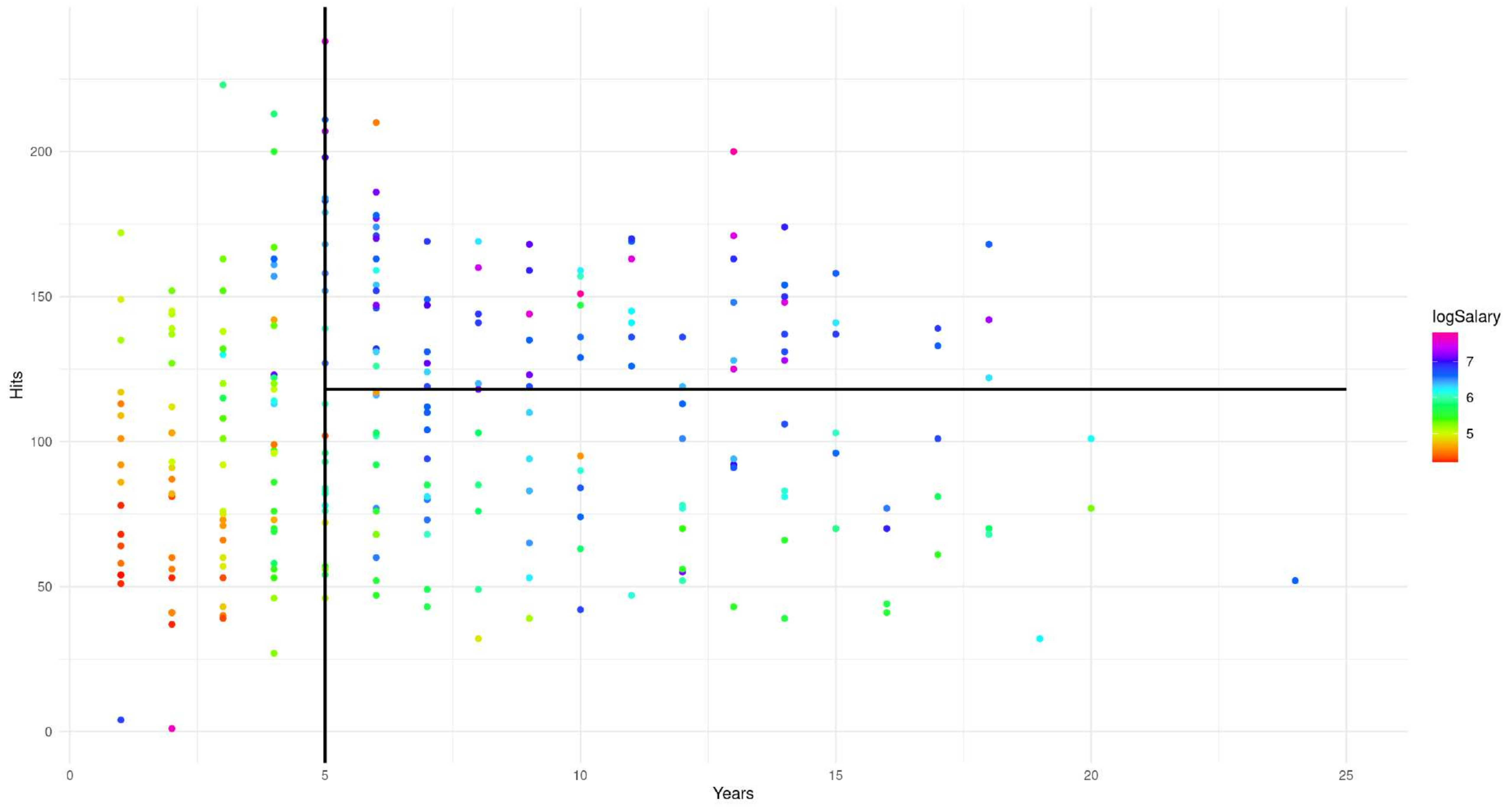


A simple regression tree for baseball data

- Using this data set, let’s say, we wanted to find players who, based on their `Hits` and `Years` in the MLB, had similar salaries:
  - We could, for instance, group together all players with less than five years of experience.
  - Among those with more than five years of experience, we could again distinguish between those with less than 118 hits and those with more.
- With these **decision rules**, we would thus have three groups in total:
  - Players with `Years < 5`
  - Players with `Years >= 5` and `Hits < 118`
  - Players with `Years >= 5` and `Hits >= 118`
- Let’s draw these three regions into the plot from earlier!

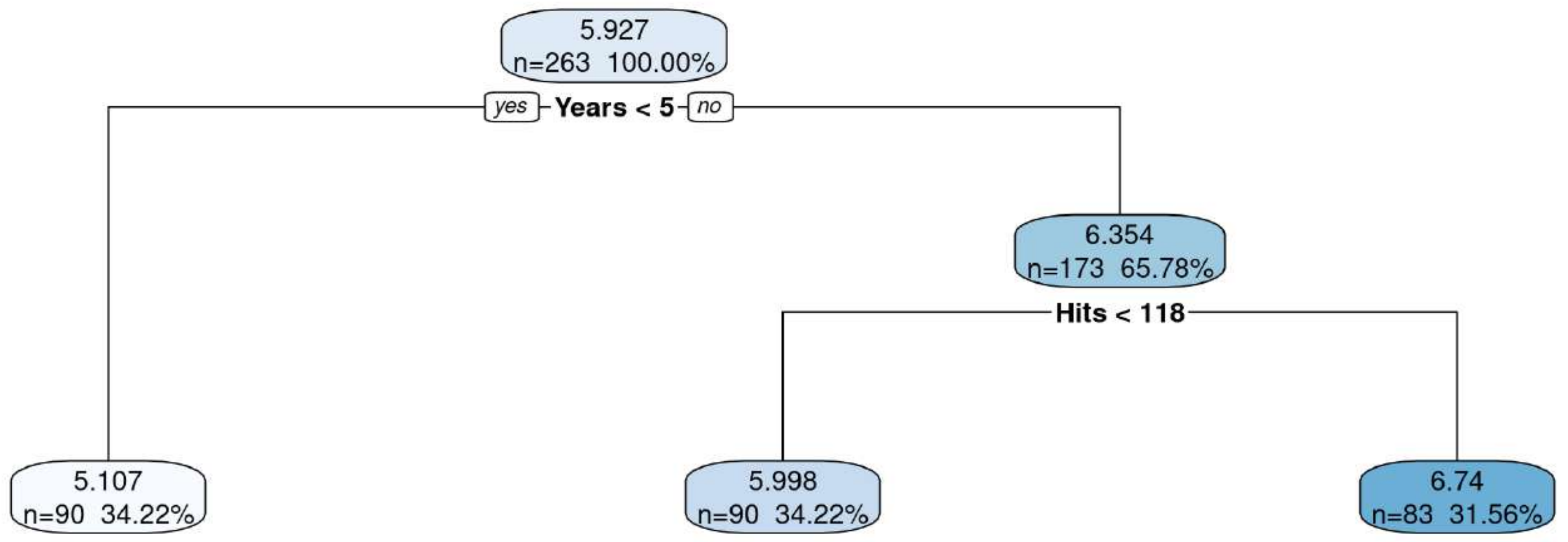
A simple regression tree for baseball data

Plot Code



Regression tree illustration

We have already built our first **regression tree**! The reason for the name is that we can illustrate our decisions on the groups in a tree structure:



- The final three regions are known as **terminal nodes** or **leaves** of the tree.
- The points along the tree where the predictor space is split are referred to as **internal nodes**.

Prediction with regression trees

- But how do we use a regression tree for **prediction**?
- Very simple: our prediction for the log salary of a new player is the **average** log salary of all the players in his region!
- As the plot shows, our regression tree can thus only predict 3 different values:
  - A log salary of 5.107 for players with `Years < 5`
  - A log salary of 5.998 for players with `Years >= 5` and `Hits < 118`
  - A log salary of 6.740 for players with `Years >= 5` and `Hits >= 118`
- **Interpretation**:
  - For a player with less than 5 years of experience, the number of hits that he made in the previous year seems to play little role in his salary.
  - But among more experienced players, the number of hits made in the previous year does (positively) affect salary.

Regression trees – The general idea

- While this is likely an oversimplification of the true relationship between experience, hits and salary, the regression tree has the advantage of being **easy to interpret** and **to visualize**.
- We have now seen a basic example of a regression tree. Now, we will discuss how to **build** them. Roughly speaking, there are two steps:
    1. We divide the **predictor space**, i.e. the set of possible values for $X_1, X_2, \ldots, X_p$, into J distinct and non-overlapping regions, $R_1, R_2, \ldots, R_J$. In the previous example, we had $J = 3$.
    2. For every observation that falls into the region R_j, we make the same prediction, which is simply the mean of the response values for the training observations in R_j.
- As we saw in the example, once we know the regions, step 2 is easy. But how do we get these regions $R_1, R_2, \ldots, R_J$?

Regression trees – Constructing regions

- In theory, the regions could have any shape. However, in decision trees, we typically divide the predictor space into high-dimensional **rectangles or boxes**.
- The goal is to find boxes $R_1, \ldots, R_J$ that minimize the RSS given by

$$\sum_{j=1}^{J} \sum_{i: x_i \in R_j} (y_i - \hat{y}_{R_j})^2,$$

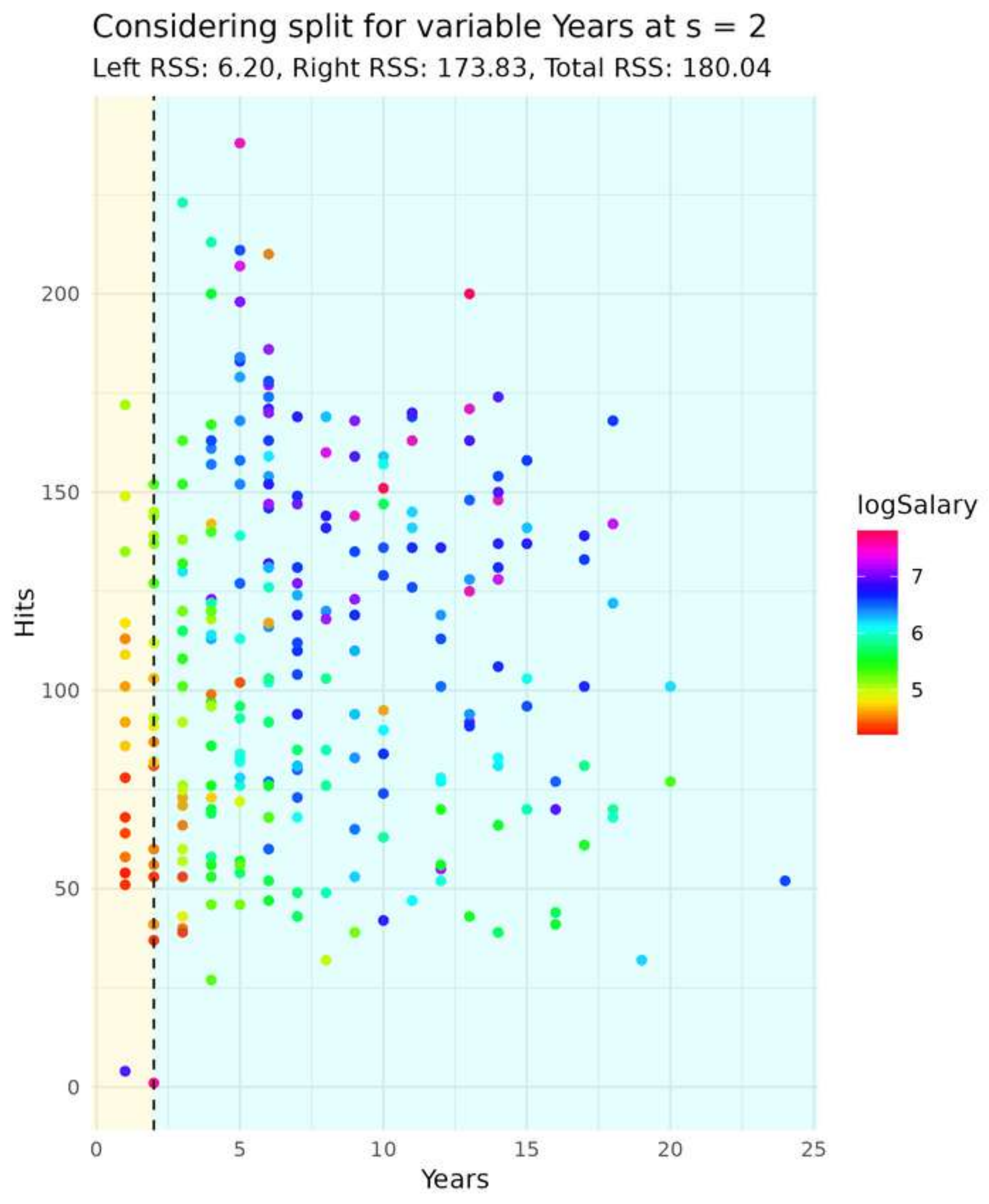
  where $\hat{y}_{R_j}$ is the mean response for the training observations within the jth box. Unfortunately, it is computationally infeasible to consider every possible partition of the feature space into J boxes.
- Instead, an algorithm known as **recursive binary splitting** is used for the purpose of achieving this goal.

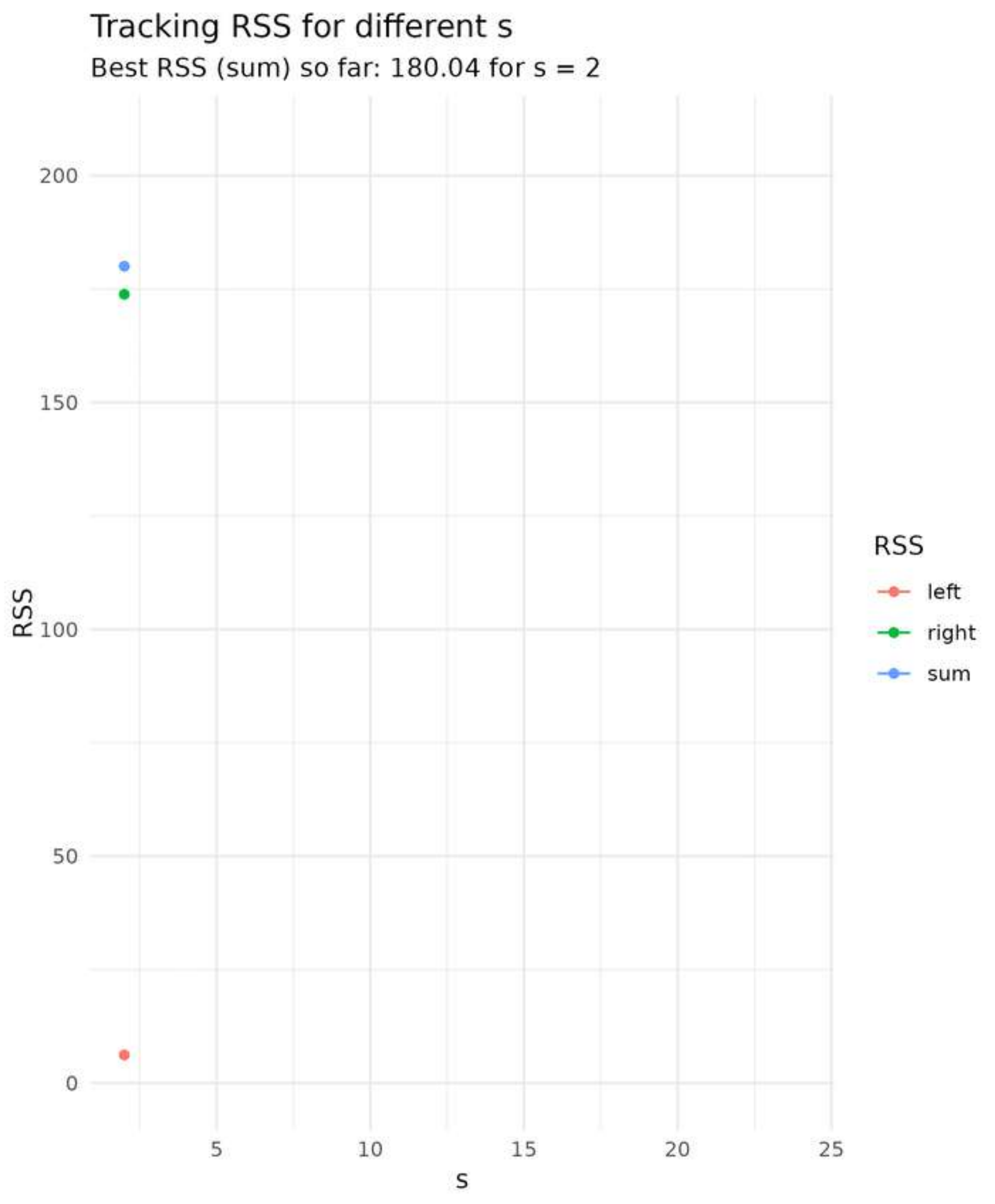
Regression trees – Recursive binary splitting

- **Recursive binary splitting** is based on the idea of, at every step, using a **single predictor** X_j at a node as a splitting variable.
- This algorithm roughly works as follows:
  - First, select the predictor X_j and the cutpoint s such that splitting the predictor space into the regions $\{X|X_j < s\}$ and $\{X|X_j \geq s\}$ leads to the greatest possible reduction in RSS.
  - Next, we repeat the process, choosing predictor and cutpoint so as to minimize the RSS within each of the resulting regions. However, this time, instead of splitting the entire predictor space, we split one of the two previously identified regions.
  - Continue splitting optimally in this way until some stopping criterion is reached, e.g. until no region contains more than five observations.

Regression trees – Recursive binary splitting

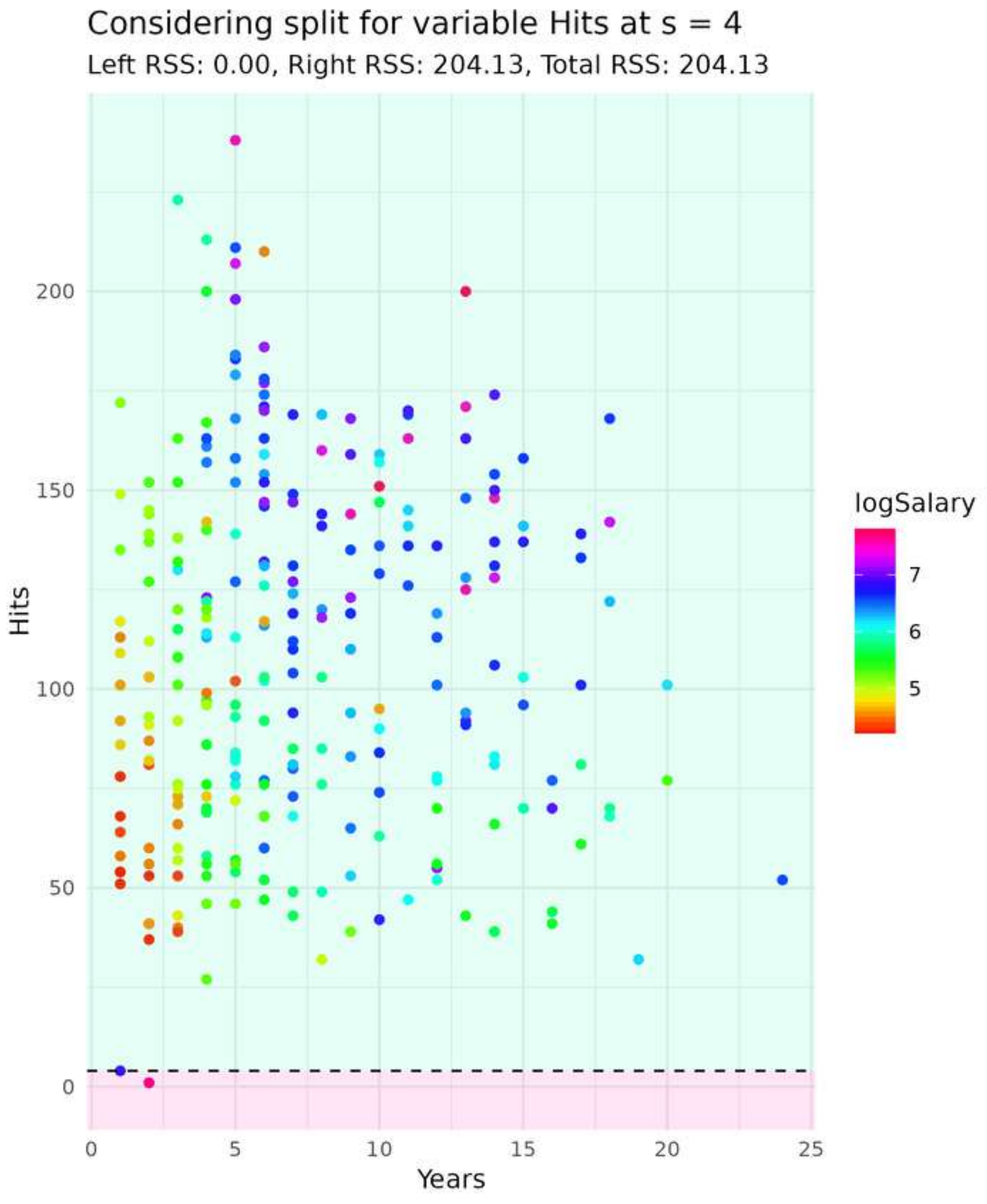
In the baseball example, we first consider all possible splits on `Years`:

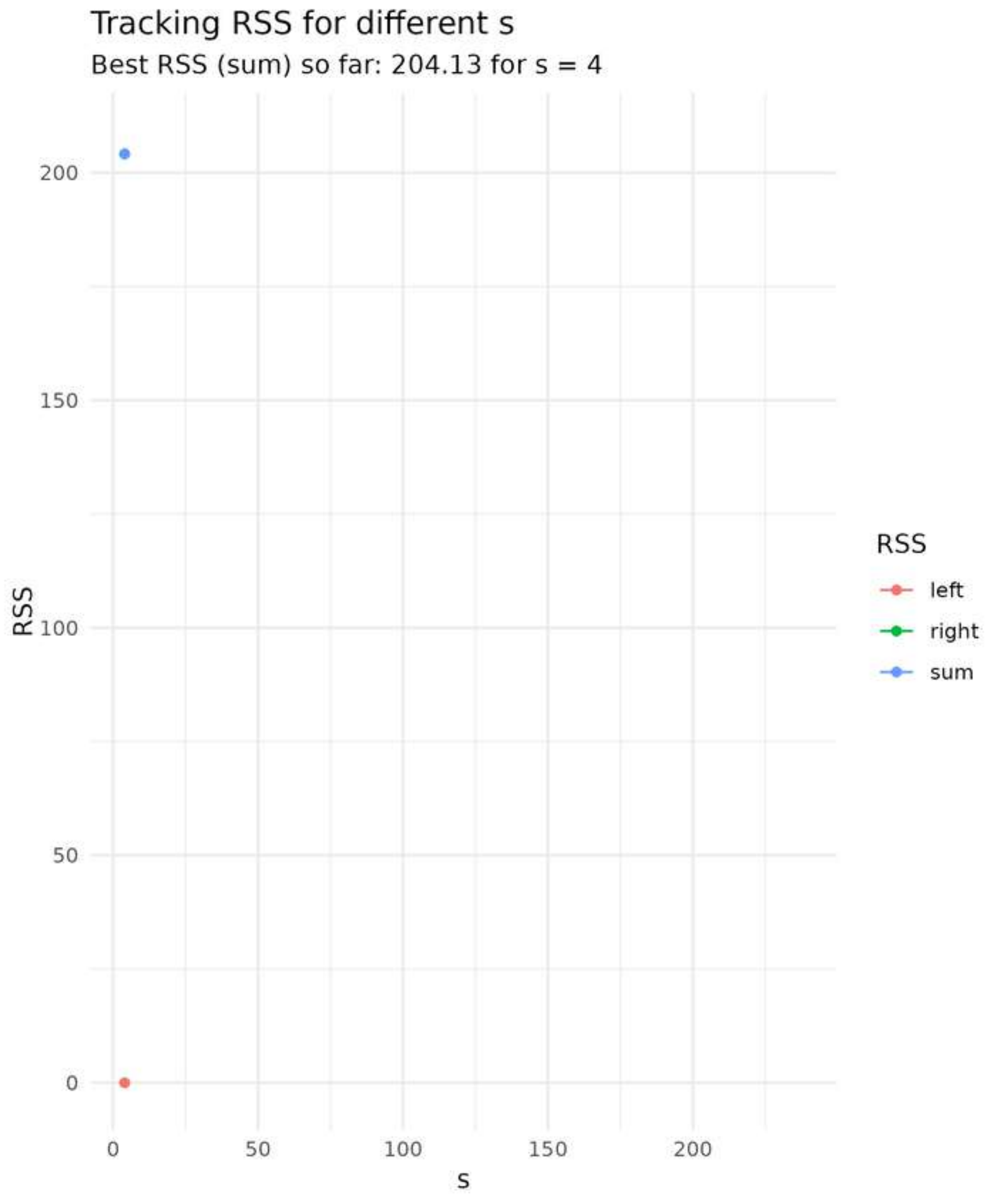




Regression trees – Recursive binary splitting

Then, we run the same procedure for `Hits` to check if we get any lower RSS:

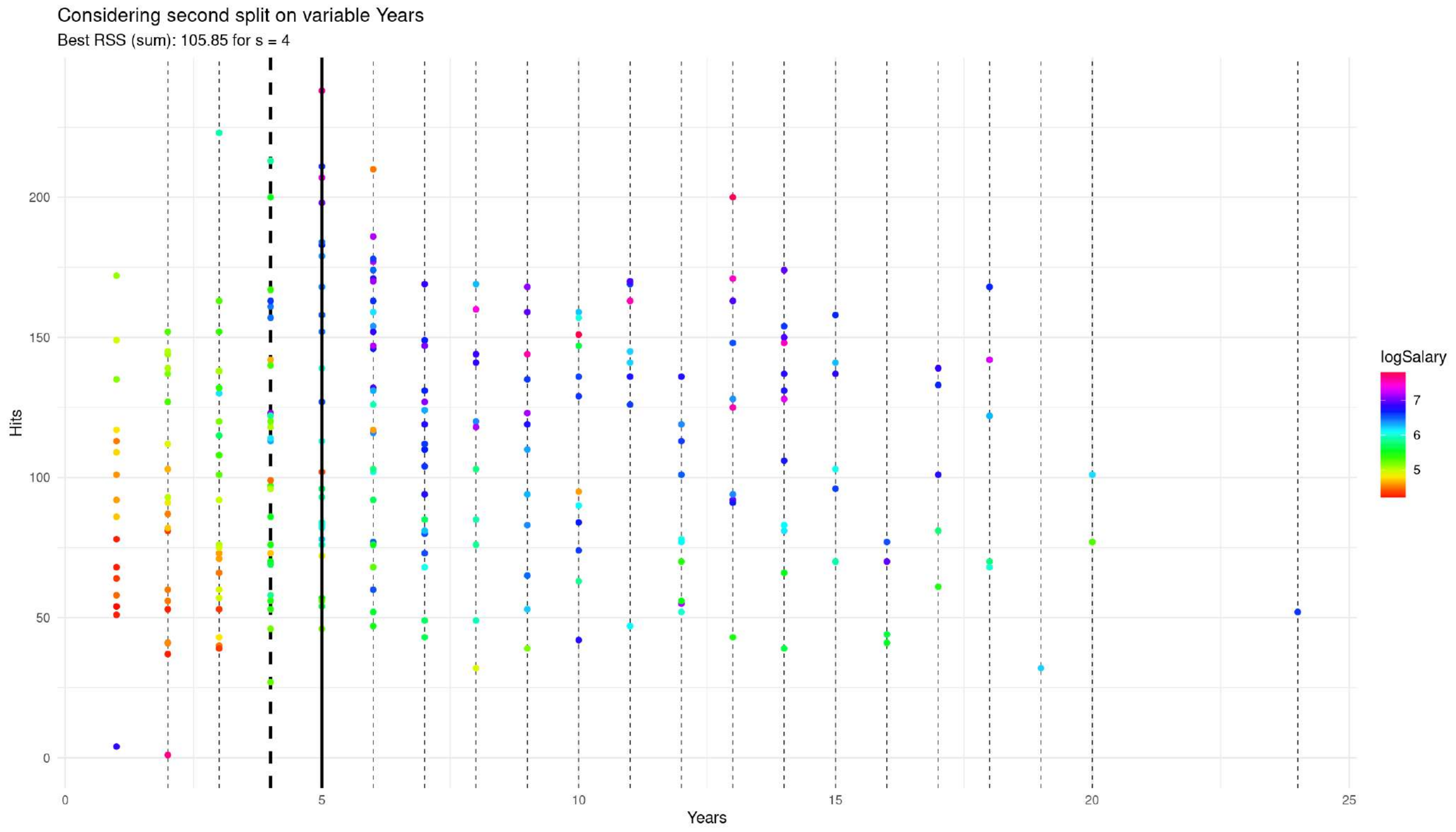




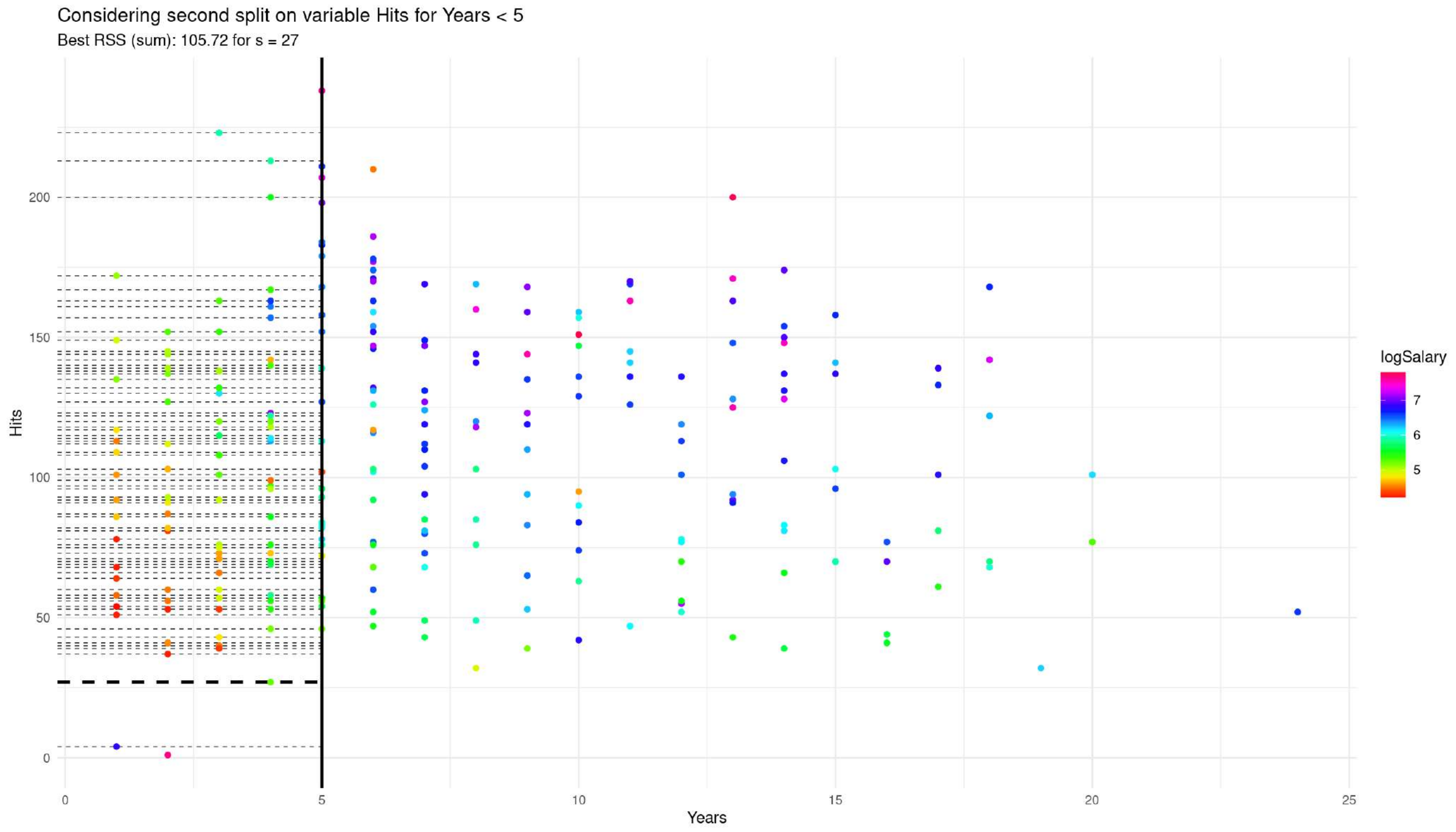
Regression trees – Recursive binary splitting

- Based on these results, we can determine the first split:
  - For the variable `Years`, the lowest RSS was achieved for $s = 5$. The lowest achievable RSS was 115.06.
  - For the variable `Hits`, the lowest RSS was achieved for $s = 118$. The lowest achievable RSS was 160.97.
  - Therefore, the first split will be based on `Years` for $s = 5$. We split the predictor space into two rectangles: one where `Years < 5` and one where `Years >= 5`, both times irrespective of the number of `Hits`.
- Next, we again check which split (across either of those two rectangles) gives the lowest RSS. Remember that we only split **one rectangle at a time**.

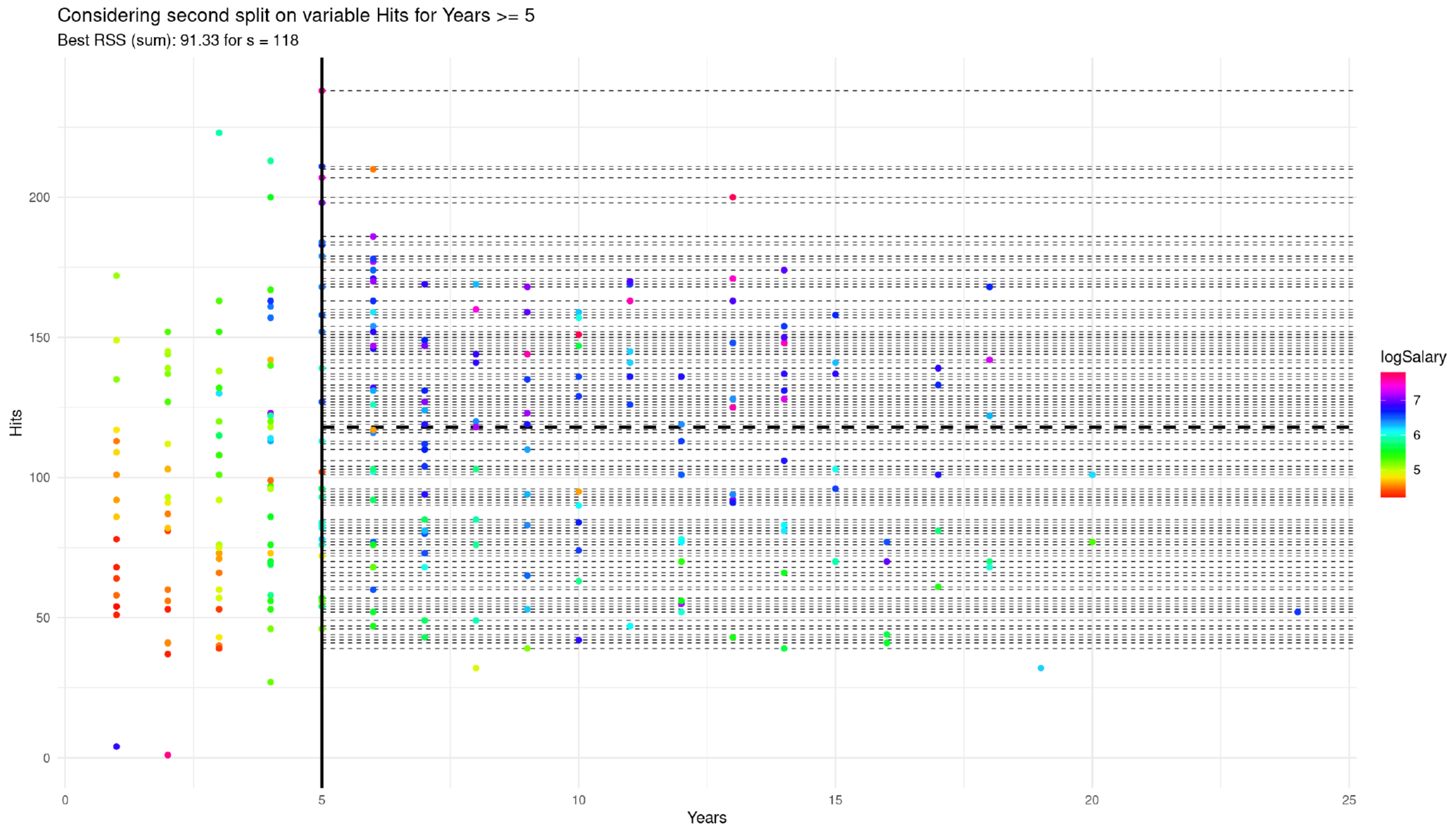
Recursive binary splitting – Second split



Recursive binary splitting – Second split



Recursive binary splitting – Second split

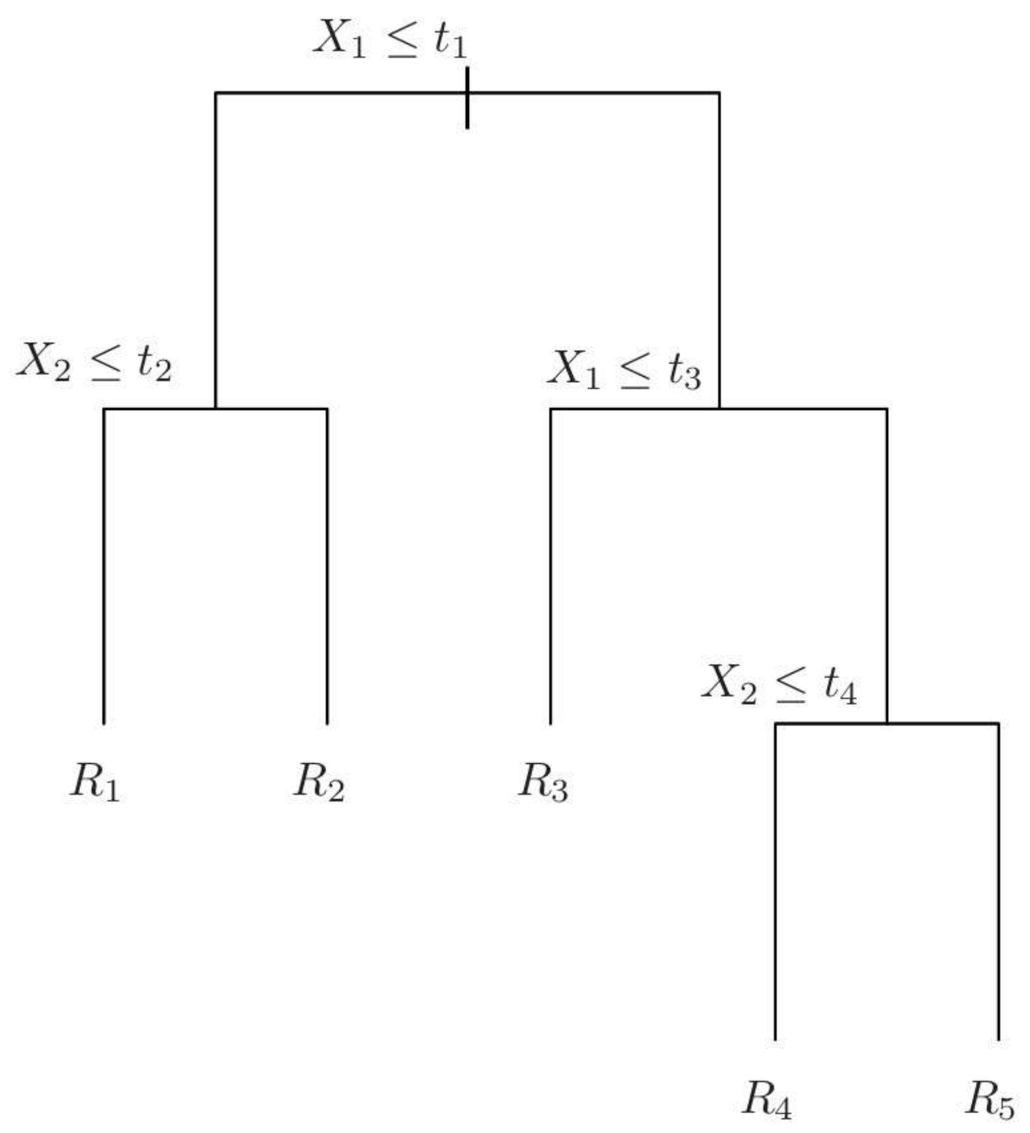


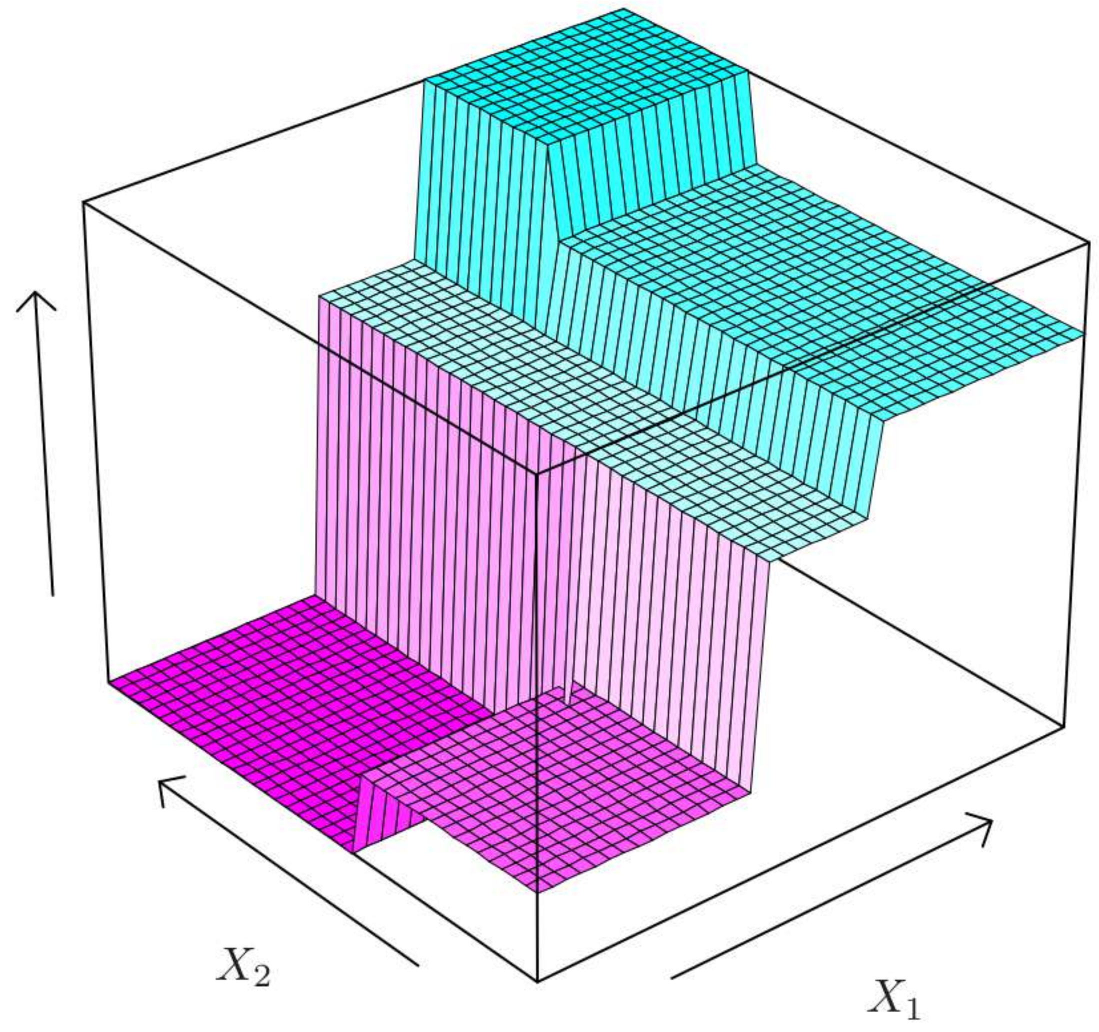
Recursive binary splitting – Second split

- The next best split is splitting the rectangle where `Years >= 5` into two smaller rectangles: one where `Hits < 118` and one where `Hits >= 118`. This gives $RSS = 91.33$.
- Stopping after these two steps gave us the regression tree we saw in the introduction. However, we could continue until some stopping criterion is reached, e.g. no region contains more than five observations.
- Recursive binary splitting is known as a **top-down**, **greedy** algorithm:
  - It is **top-down** because it begins at the top of the tree (all observations in a single region) and then successively splits the predictor space, each split indicated as two new branches further down on the tree.
  - It is **greedy** because at each step, the best split is made at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step.

Prediction surface in regression trees

With recursive binary splitting, the prediction surface of regression trees looks like a (irregular) **stair case**. Below is an example with five terminal nodes. Notice the difference to linear regression which would be a plane.





Pruning a tree

- Now, performing recursive binary splitting, until only a few observations are left in each region is likely to **overfit** the data:
  - A high number of splits produces a very complex tree, which will have **low training error**, but (probably) **high test error**.
  - Thus, a smaller tree might lead to **lower variance** and better interpretation at the **cost of only a little bias**.
- One way to mitigate this is to grow a very large tree T_0 and then to **prune it back** (cut branches of the tree) in order to obtain a subtree.
- The way we do this in practice is through an approach called **cost complexity pruning** (aka **weakest link pruning**).

Cost complexity pruning

- The idea of **cost complexity pruning** is to choose a subtree T of T_0 that optimally trades off a subtree's complexity against its fit to the training data.
- For this purpose, we set a tuning parameter $\alpha > 0$ and then find T so that

$$\sum_{m=1}^{|T|} \sum_{i:x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha \cdot |T|$$

  is minimized. $|T|$ is the size of the tree, i.e. the number of terminal nodes.
- The tuning parameter α controls the aforementioned trade-off:
  - When $\alpha = 0$, then $T = T_0$ because the expression just measures the training error.
  - However, when $\alpha > 0$, there is a price to pay for having a tree with many terminal nodes, so the expression will "prefer" a smaller subtree.

Cost complexity pruning – Choosing α

- In general, the higher the value of α, the more the initial fully grown tree T_0 will be pruned. Thus, as we increase α from 0, the optimal subtree will become smaller and smaller.
- So how do we choose the optimal α?
  - We perform K-fold cross-validation!
  - More specifically, we assess the predictive performance of the subtrees on each of the K hold-out folds as a function of α and then pick the α that minimizes the cross-validated test MSE.
- Having determined the optimal α in this way, we can then apply it to obtain the optimally pruned tree on the entire training data set.
- Fortunately, this process is implemented in a highly optimized R package called `rpart` that does most of this work for us. Let's have a look at it!

Regression trees in R – Fitting and visualization

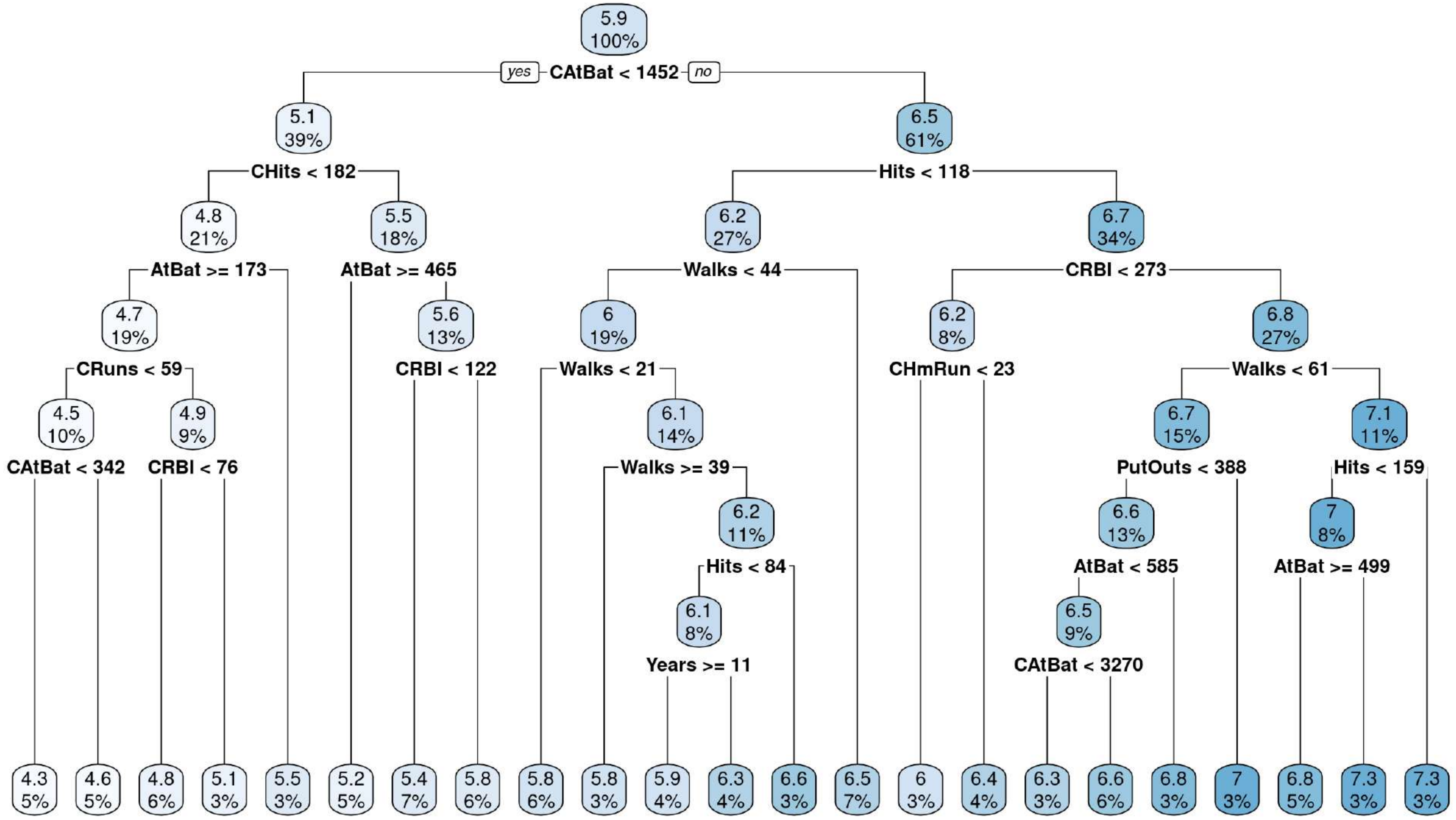
- The main function in the `rpart` package is also called `rpart`. Besides fitting a regression tree, it performs the cross-validation analysis for α automatically in the background.
- To use it, we need to pass it a model **formula** and **data** (like in `lm`) as well as the minimum α value, which is referred to as `cp` (complexity parameter) in the package (0.01 by default).
- Using all the variables in the baseball data, we can thus fit a “full” tree like this:

```r
library(rpart)

rt_full <- rpart(logSalary ~ . - Salary, data = Hitters,
                 control = list(cp = 0))
```

- The output from `print(rt)` is not very user-friendly so we will visualize the tree using the `rpart.plot` package (result on the next slide):

```r
library(rpart.plot)
rpart.plot(rt_full)
```

Regression trees in R – Visualization



Regression trees in R – Pruning

- As expected, this is quite a large tree! So how do we go about optimally **pruning** it?
- As indicated before, `rpart` has done the necessary cross-validation (10-fold by default) exercise during fitting. We can look at the result by calling

```
printcp(rt_full)
```

```
Regression tree:
rpart(formula = logSalary ~ . - Salary, data = Hitters, control = list(cp = 0))

Variables actually used in tree construction:
 [1] AtBat   CAtBat  CHits   CHmRun  CRBI    CRuns   Hits    PutOuts Walks
[10] Years

Root node error: 207.15/263 = 0.78766

n= 263

          CP nsplit rel error  xerror     xstd
1  0.5689379      0   1.00000 1.00594 0.065303
```

Regression trees in R – Pruning

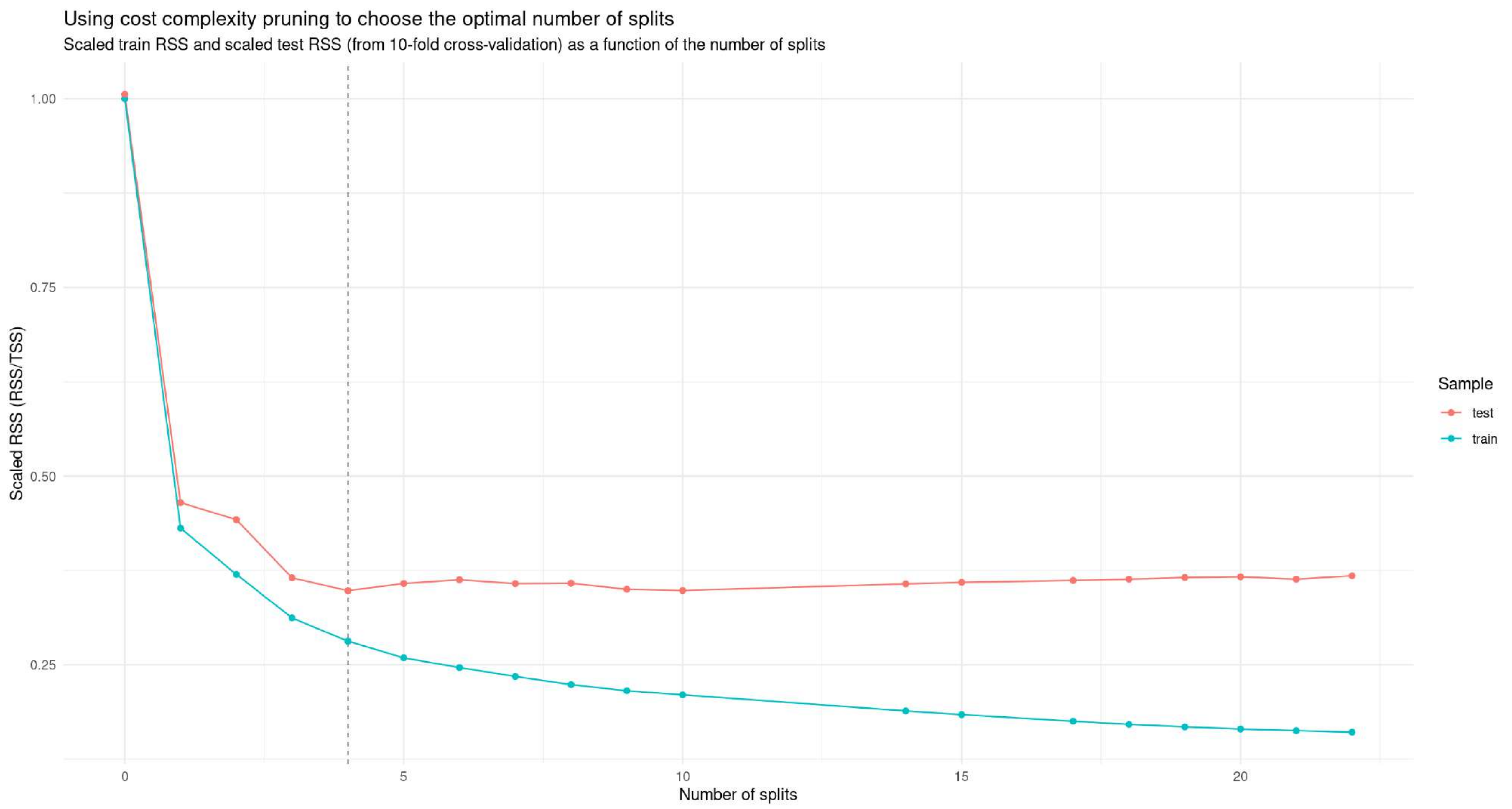
- The main output of that function is a table with **five columns**:
  - `CP`: The **complexity parameter**, a scaled version of α. In each row, it is the smallest penalty parameter, for which the number of splits is still equal to `nsplit`, e.g. 0.06129 is the smallest value for which there is only one split.
  - `nsplit`: The number of splits induced by the value of `CP`.
  - `rel error`: Scaled training RSS. More precisely, it is RSS/TSS in-sample. Will always decrease as the number of splits (i.e. the size of the tree) increases.
  - `xerror`: Scaled test RSS. More precisely, it is RSS/TSS out-of-sample (i.e. averaged across the hold-out folds). Primary metric for assessing optimal number of splits.
  - `xstd`: Standard deviation of `xerror` over the different folds.

Regression trees in R – Pruning

- With this table, we can now plot the **scaled RSS** (in train and test sample) on the y-axis against the **number of splits** on the x-axis and then choose the number of splits optimally.
- In choosing the number of splits, we want to balance to conflicting goals:
  - Choose as large a tree as necessary to deliver a good fit.
  - Among those with similar goodness-of-fit, choose the smallest tree possible.
- A good, yet inevitably somewhat subjective heuristic is therefore: **Choose the number of splits where the scaled test RSS starts to flatten out.**
- Let's create the plot to see where that point would be in our example (**Note**: we do not have to use `ggplot2` to create this plot, a quick way to do it is to simply run the function `rsq.rpart` on our tree)!

Regression trees in R – Pruning

Plot Code



Regression trees in R – Pruning

- Seems like a good value for the number of splits in the baseball example is 4. Increasing the number of splits to any point beyond that does not deliver any (meaningful) reduction in the scaled test RSS.
- Now, looking back to the table, we have to find the complexity parameter `cp` corresponding to `nsplit = 4`:

```r
rt_full$cptable[rt_full$cptable[,"nsplit"] == 4, ]
```

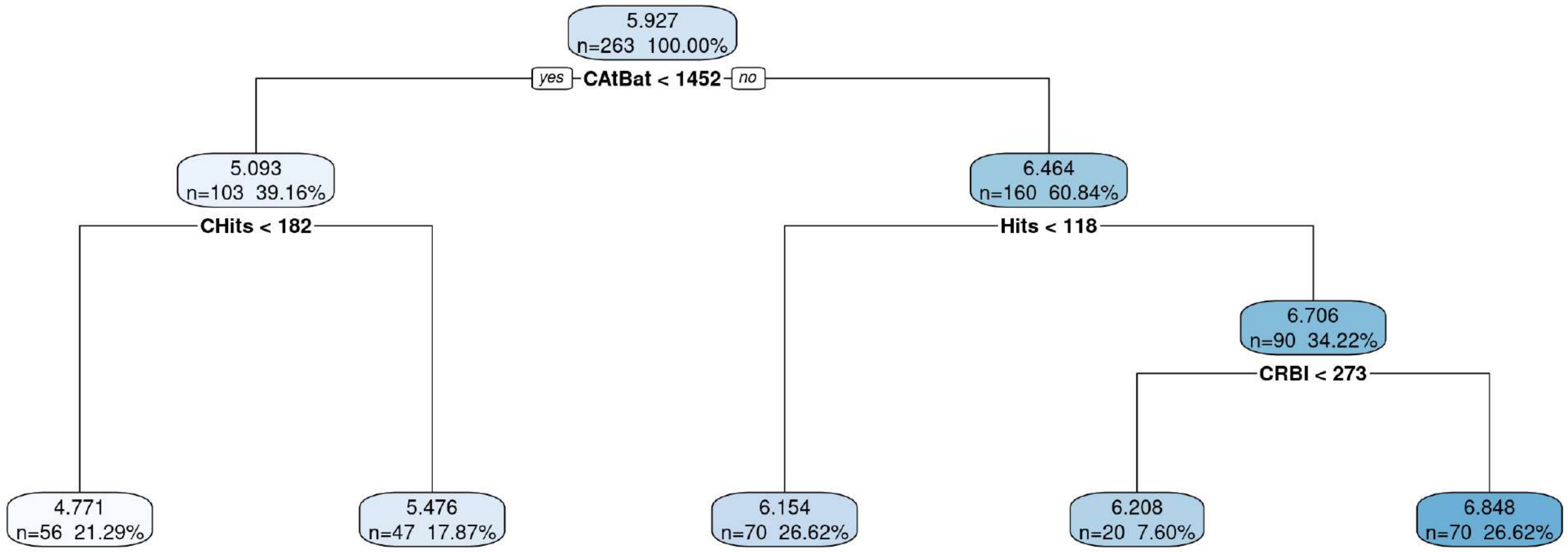
```
        CP     nsplit  rel error     xerror       xstd
0.02194488 4.00000000 0.28120373 0.34820532 0.05901179
```

- It is 0.02194488. To finally optimally prune the tree, we now have to apply the `prune` function to our full tree `rt_full` together with the optimal `cp` parameter:

```r
rt_pruned <- prune(rt_full, cp = 0.02194488)
```

Regression trees in R – Optimally pruned tree

```r
# Plotting pruned tree with some extra information:
rpart.plot(rt_pruned, extra = 101, digits = 4)
```

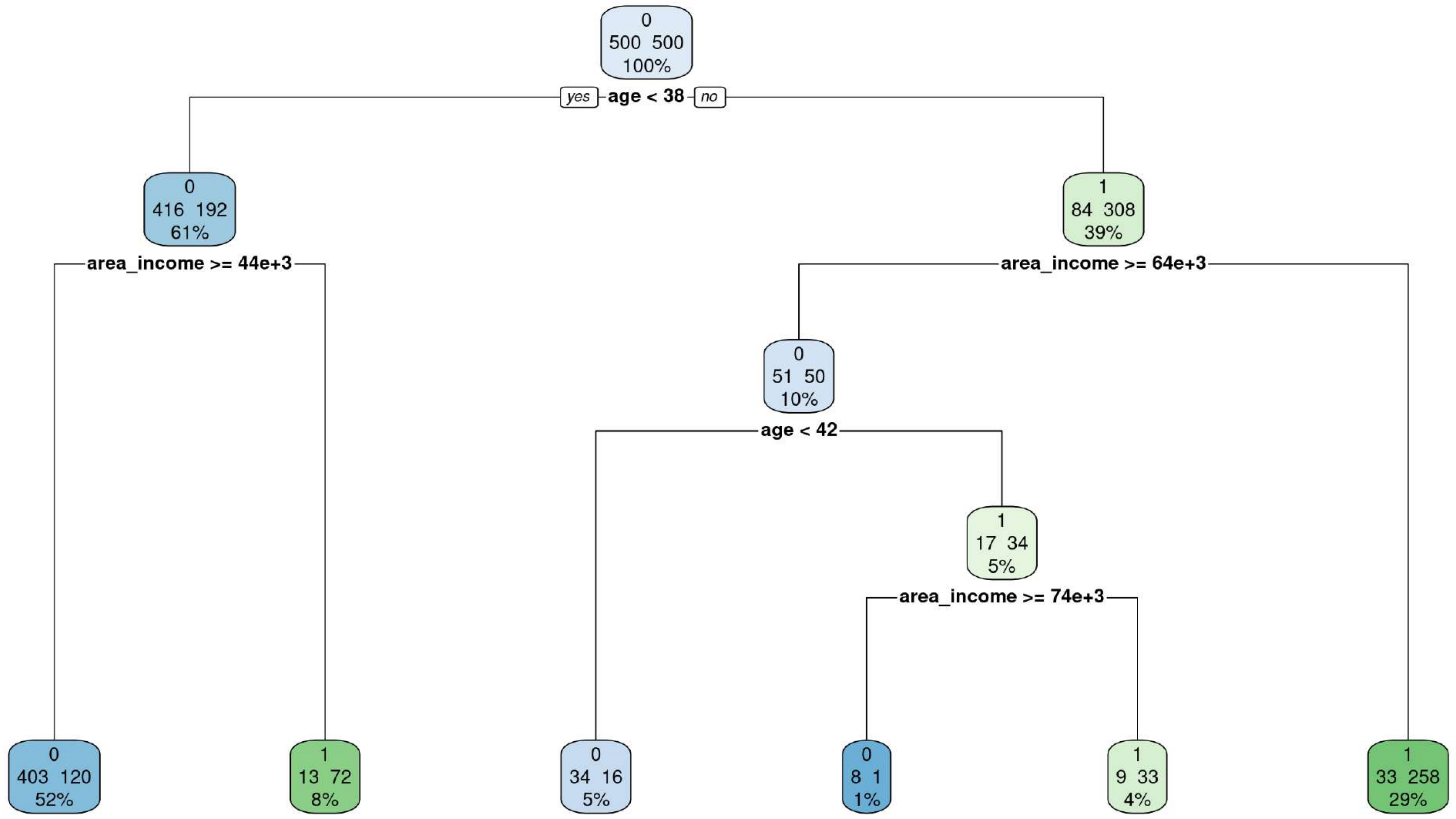


This is now a much more parsimonious model!

Classification trees

- When decision trees are used for classification tasks, they are referred to as **classification trees**.
- Almost all of the logic regarding recursive binary splitting, tree growing and pruning carries over from **regression trees**, but there are two crucial differences:
  - In regression trees, the predicted response is the **mean response** of the training observations belonging to the same terminal node. In classification trees, we use the **most commonly occurring class**.
  - In regression trees, we use **RSS** as a criterion for making the binary splits. In classification trees, we use the so-called **Gini index** instead.
- Let's grow a tree on the `click` data set from earlier (with predictors `area_income` and `age`) to dive deeper into these differences.

Classification trees – Example



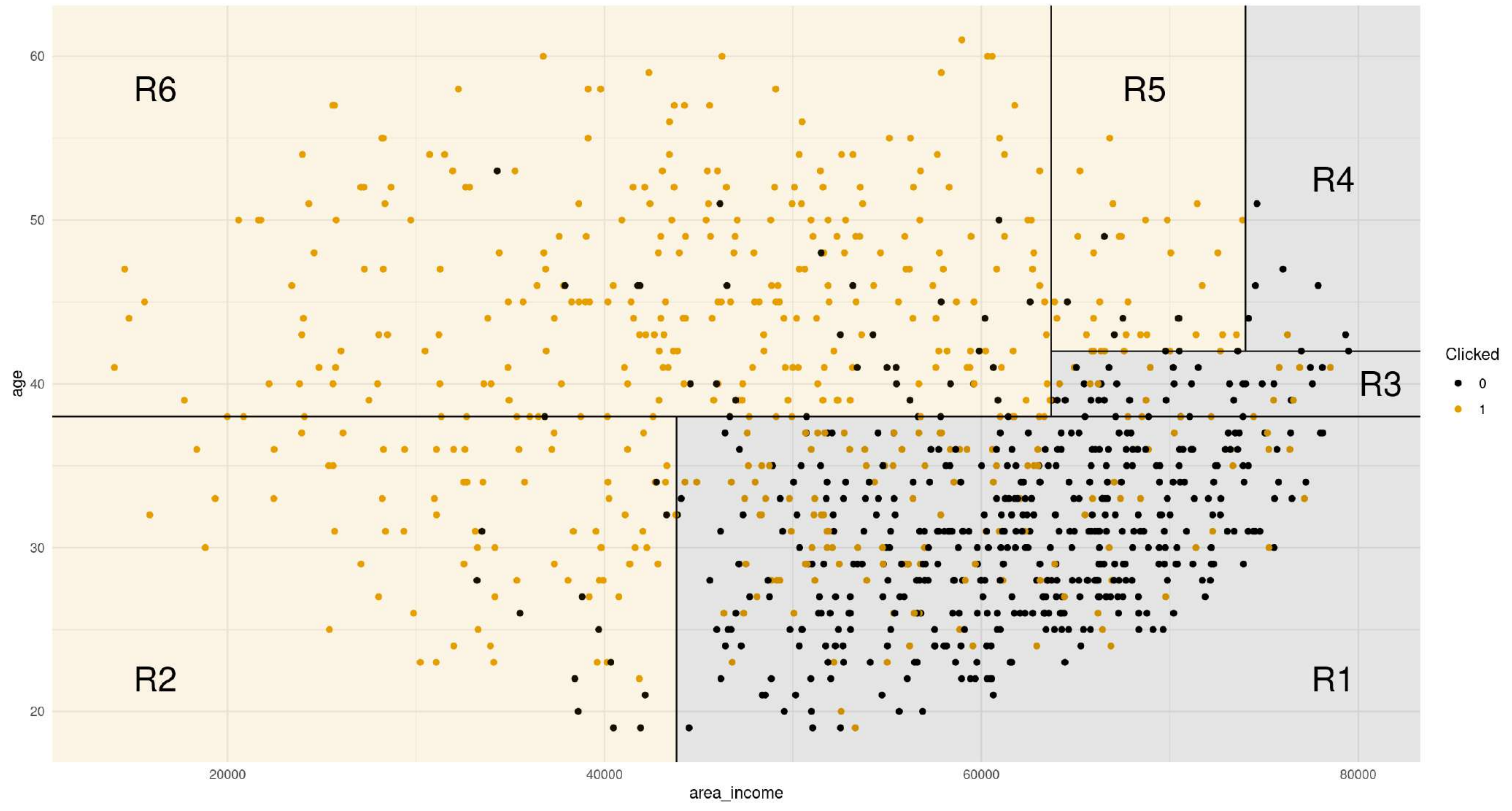
Classification trees – Example

- The grown **classification tree** has five internal nodes and **six terminal nodes (leaves)**. From left to right, we label the six resulting rectangular regions by R_1, R_2, R_3, R_4, R_5 and R_6.
- The predicted class in each leaf is the one having the majority among the observations in there: green for click and blue for no click with the darkness indicating the **purity** of the node.
- Each node displays the following pieces of information:
  - The majority class at the top (0 or 1 for no click or click).
  - The number of observations falling into each of these classes in that node.
  - The percentage of all observations in the node.
- The rectangular regions and resulting decision boundary can be highlighted in the two-dimensional scatter plot from earlier.

Classification trees – Example

Decision Boundary of classification tree built with recursive binary splitting with Gini index

Background colour indicates majority class in each region



Classification trees – Gini index

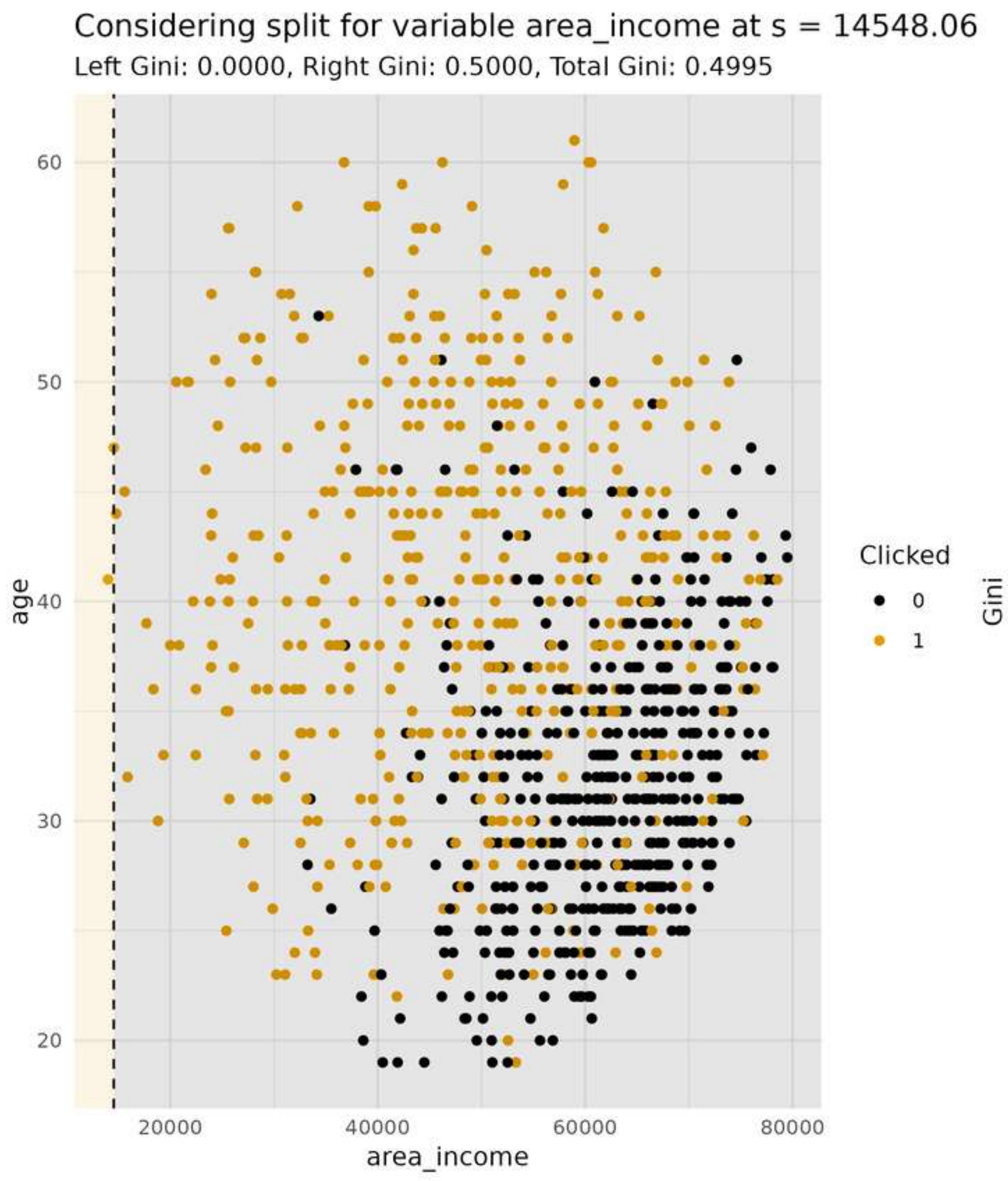
- Classification trees are still built with **recursive binary splitting**, only the criterion changes as RSS cannot be used for classification.
- Instead, at each step, we choose the split that minimizes the **Gini index**

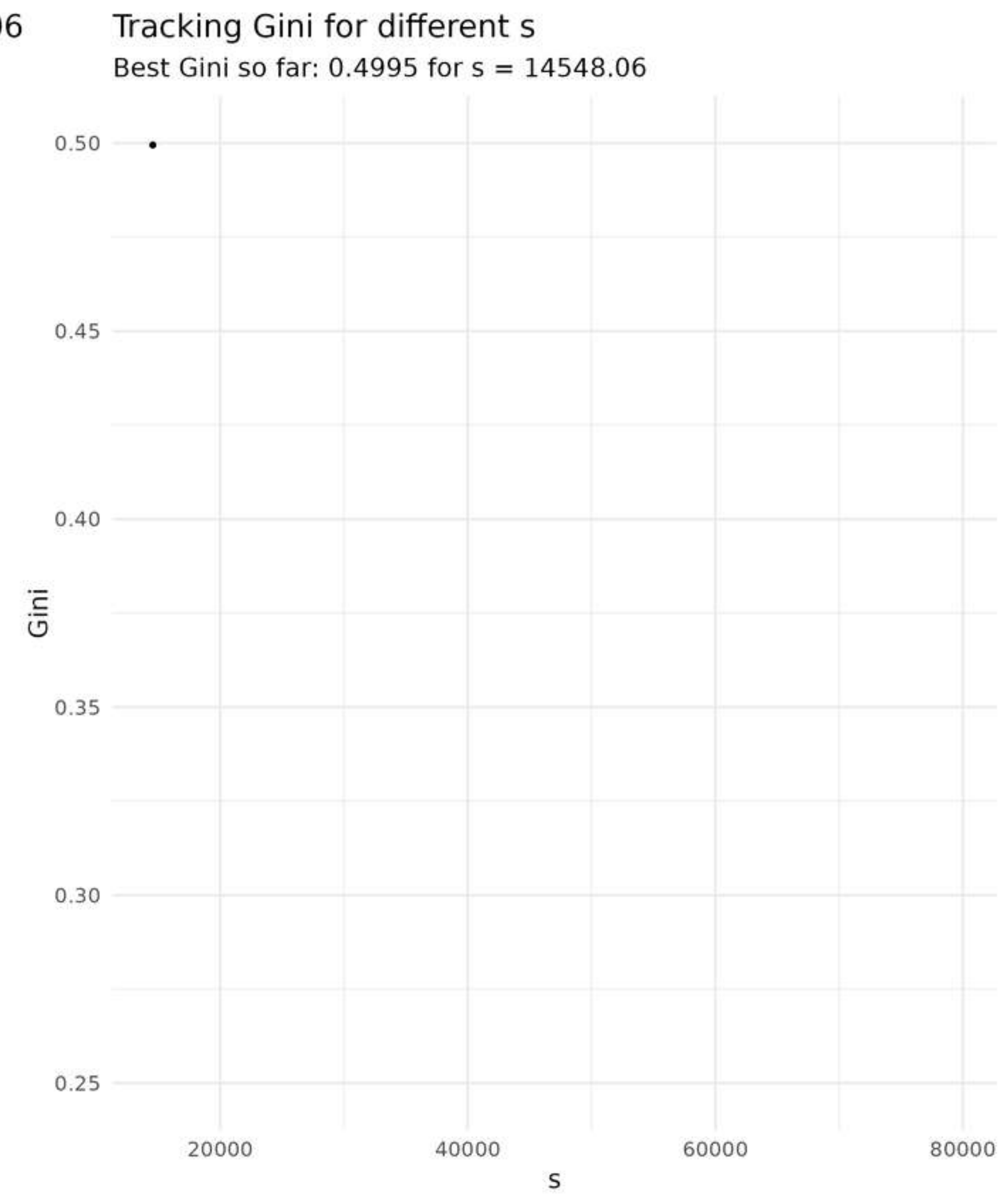
$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}),$$

  where K is the number of classes and $\hat{p}_{mk}$ is the proportion of training observations in the *m*th region that are from the *k*th class.
- The Gini index is a measure of node **purity**:
  - It is low when most observations belong to the same class.
  - It is high when they are evenly distributed among classes.

Classification trees – Recursive binary splitting

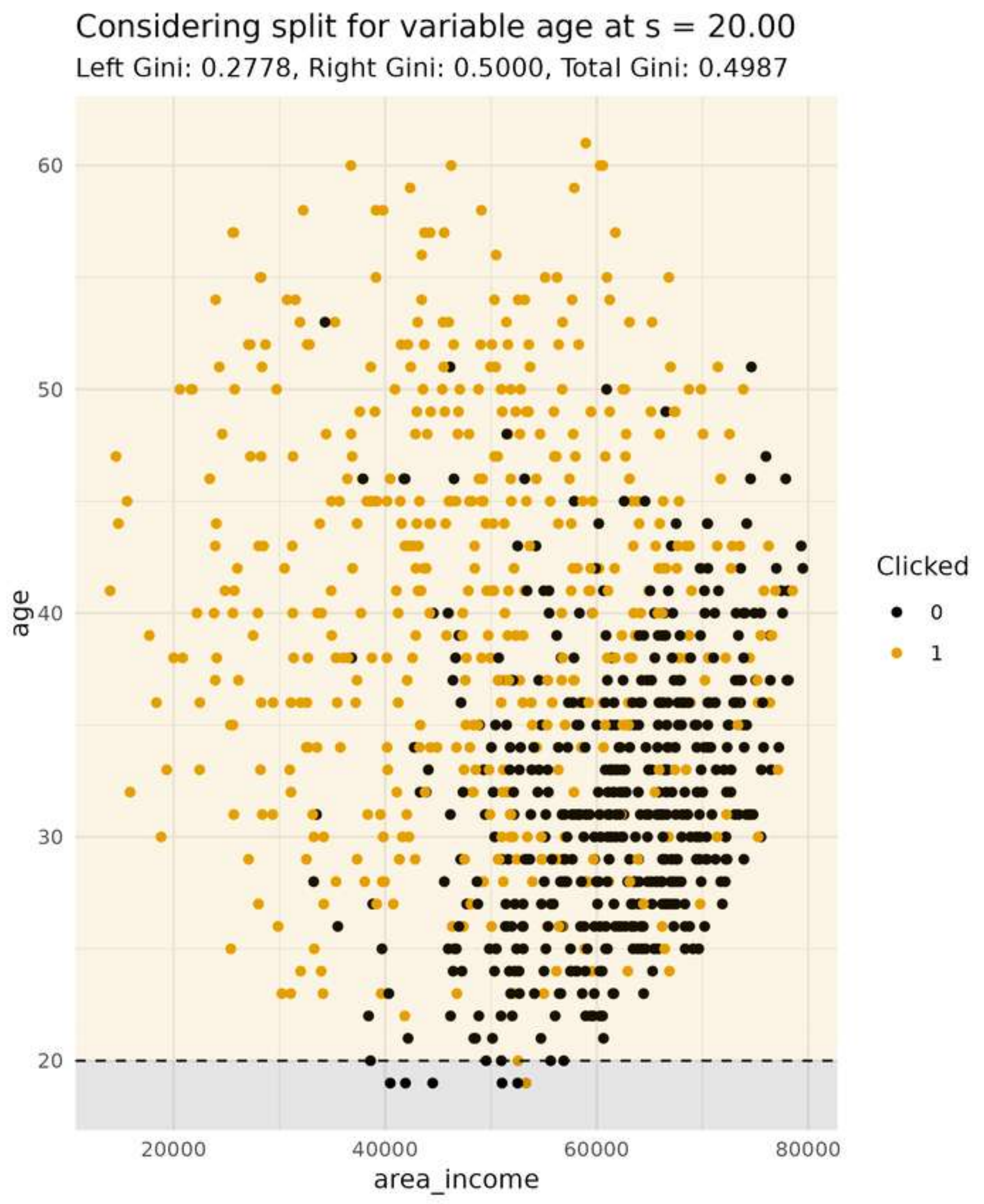
Let’s consider all possible splits on the variable `area_income`:

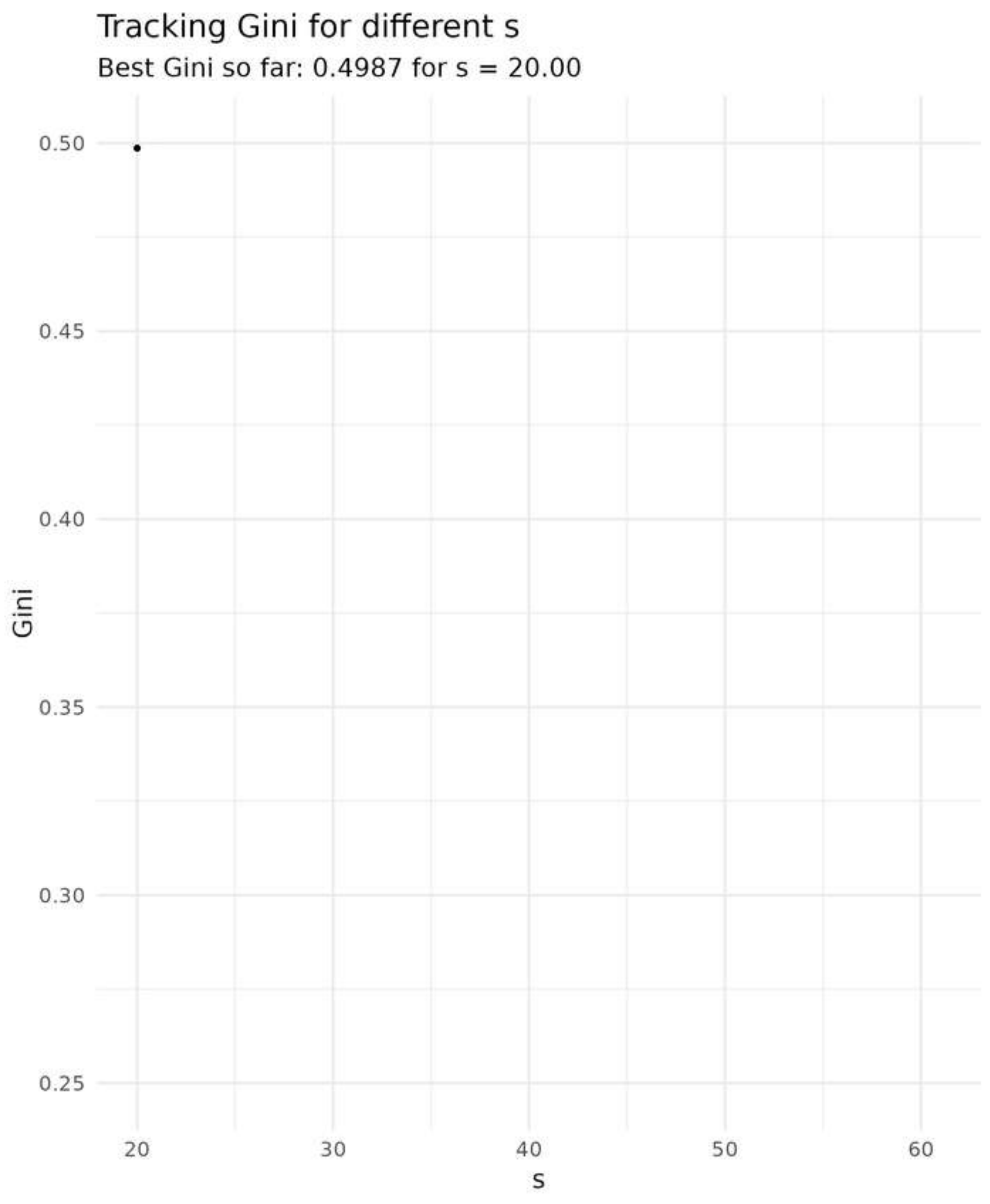




Classification trees – Recursive binary splitting

Then, we run the same procedure for age to check if we get any lower Gini:





Classification trees – Recursive binary splitting

- Based on these results, we can determine the first split:
  - For the variable `area_income`, the lowest Gini was achieved for $s = 50337.93$. The lowest achievable Gini index was 0.3983.
  - For the variable `age`, the lowest Gini was achieved for $s = 38$. The lowest achievable Gini index was 0.3947.
  - Therefore, the first split will be based on `age` for $s = 38$. We split the predictor space into two rectangles: one where `age < 38` and one where `age >= 38`, both times irrespective of the `area_income`.
- This of course again corresponds to the result we already saw. In recursive binary splitting, we would now go on splitting like this until some stopping criterion is reached.

Classification trees in R – Fitting

- In R, not much changes when we want to fit a classification tree compared to a regression tree. As long as the response is converted to a factor, the `rpart` function will automatically fit a classification tree.
- So, let's make sure the response in the `click` data set is a factor:

```r
click$clicked_on_ad <- factor(click$clicked_on_ad)
```

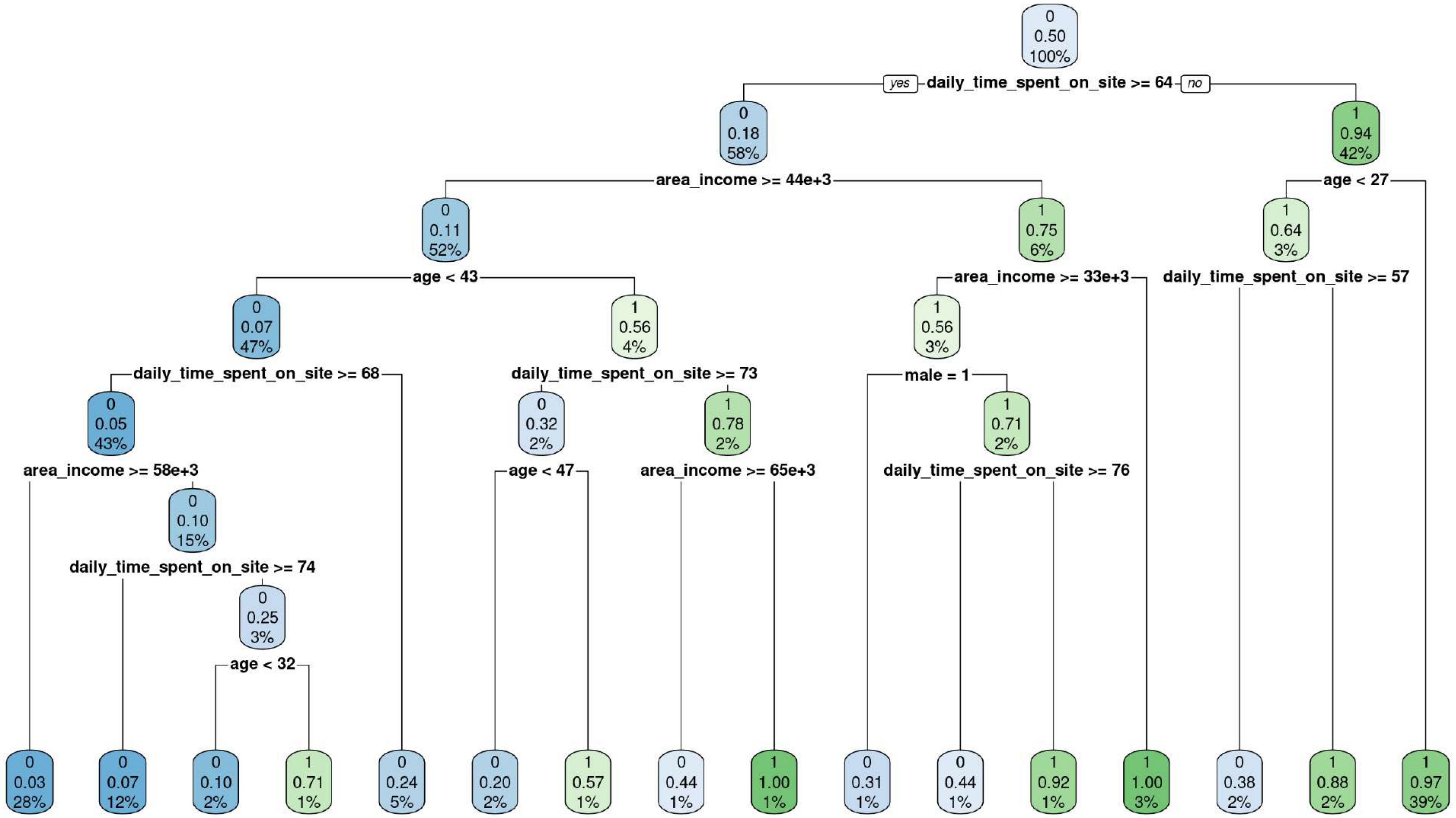
- Now, we can grow a "full" tree again in the same way as before in the `click` data. We want to use predictors `age`, `area_income`, `male` and `daily_time_spent_on_site`:

```r
ct_full <- rpart(clicked_on_ad ~ age + area_income + male + daily_time_spent_on_site,
                 data = click, control = list(cp = 0))
```

- We can again visualize the resulting classification tree using `rpart.plot` (result on the next slide):

```r
rpart.plot(ct_full)
```

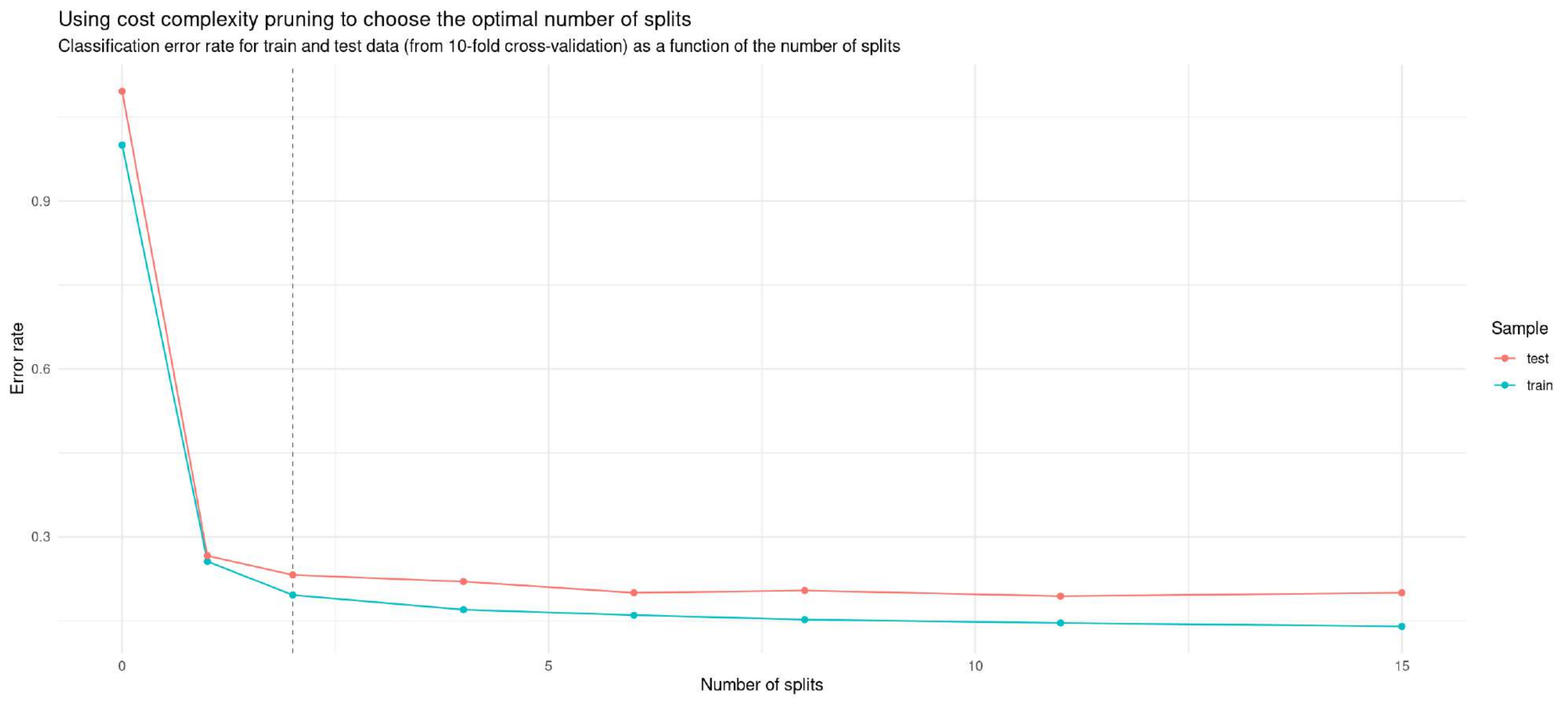
Classification trees in R – Visualization



Classification trees in R – Pruning

All functionality for pruning also carries over from regression trees. The metric used in the `cptable` is classification **error rate**:

Plot Code



Classification trees in R – Pruning

- Seems like a good value for the number of splits in the `click` example is 2. Increasing the number of splits to any point beyond that does not deliver any (meaningful) reduction in error rate.
- Now, looking at the `cptable`, we find the corresponding value of `cp`

```r
ct_full$cptable[ct_full$cptable[,"nsplit"] == 2, ]
```

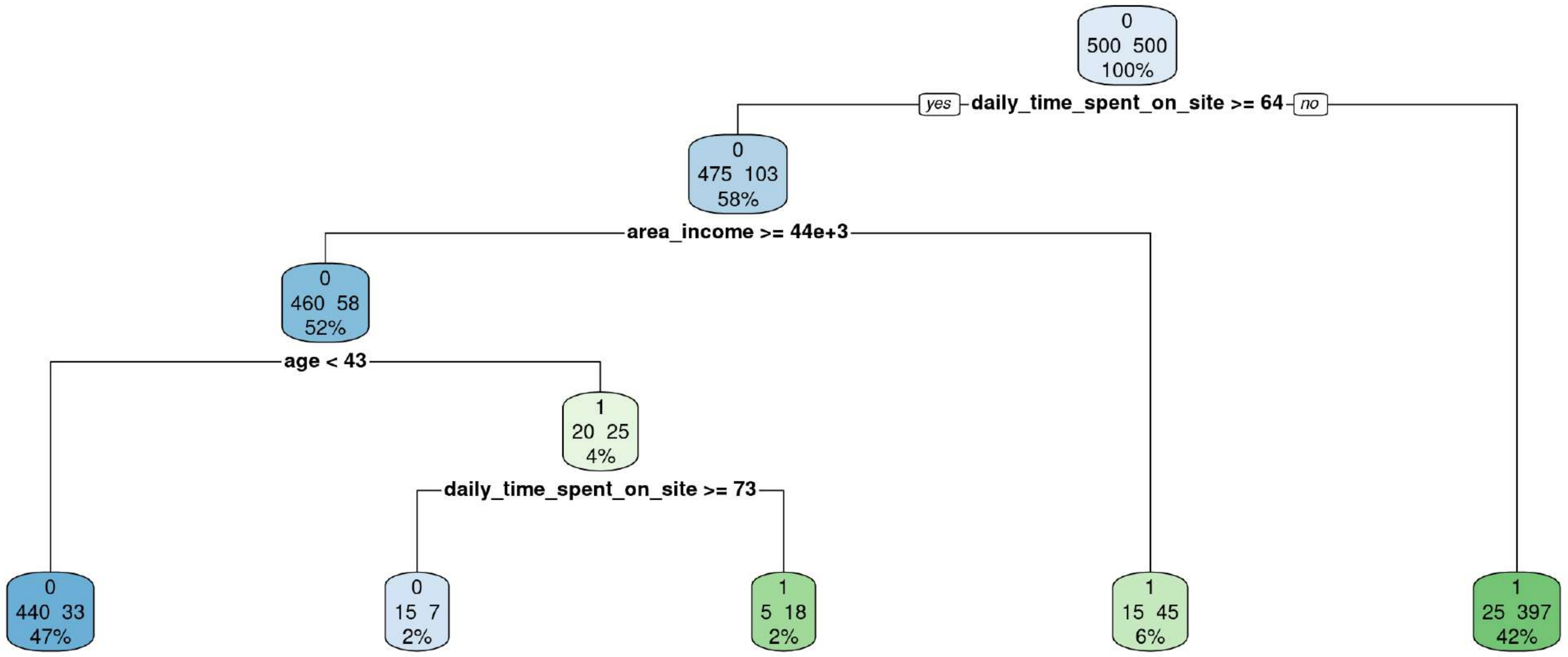
```
       CP    nsplit rel error    xerror      xstd
0.0130000 2.0000000 0.1960000 0.2320000 0.0202528
```

- It is 0.013. To finally optimally prune the tree, we again simply apply the `prune` function to the full tree `ct_full` together with the optimal `cp` parameter:

```r
ct_pruned <- prune(ct_full, cp = 0.013)
```

Classification trees in R – Optimal pruning

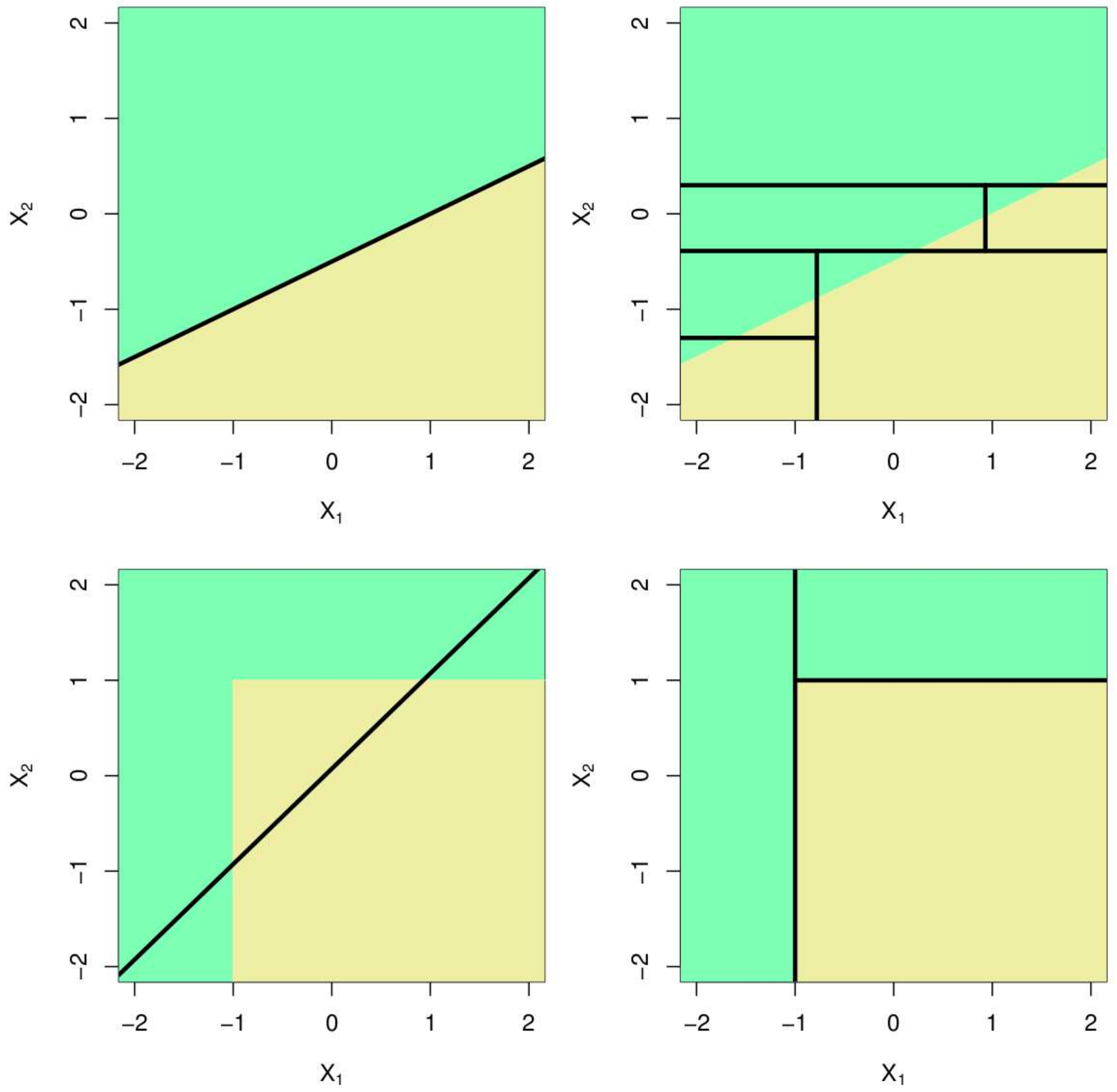
```r
# Plotting pruned tree with some extra information:
rpart.plot(ct_pruned, extra = 101)
```



Final assessment of decision trees

- Whether used for regression or classification, decision trees have many **advantages**:
  - Very easy to explain intuitively. Even easier than linear or logistic regression.
  - Can be nicely visualized and easily interpreted as long as they are only moderately large, even by non-experts.
  - Can handle qualitative predictors without the need for dummy variables.
- However, there are some **downsides** of course:
  - Due to their simplistic nature, decision trees typically do not have the same level of predictive accuracy as other methods.
  - Trees struggle with linearity, e.g. linear decision boundaries (see next slide).

Final assessment of decision trees



Unsupervised learning – Clustering

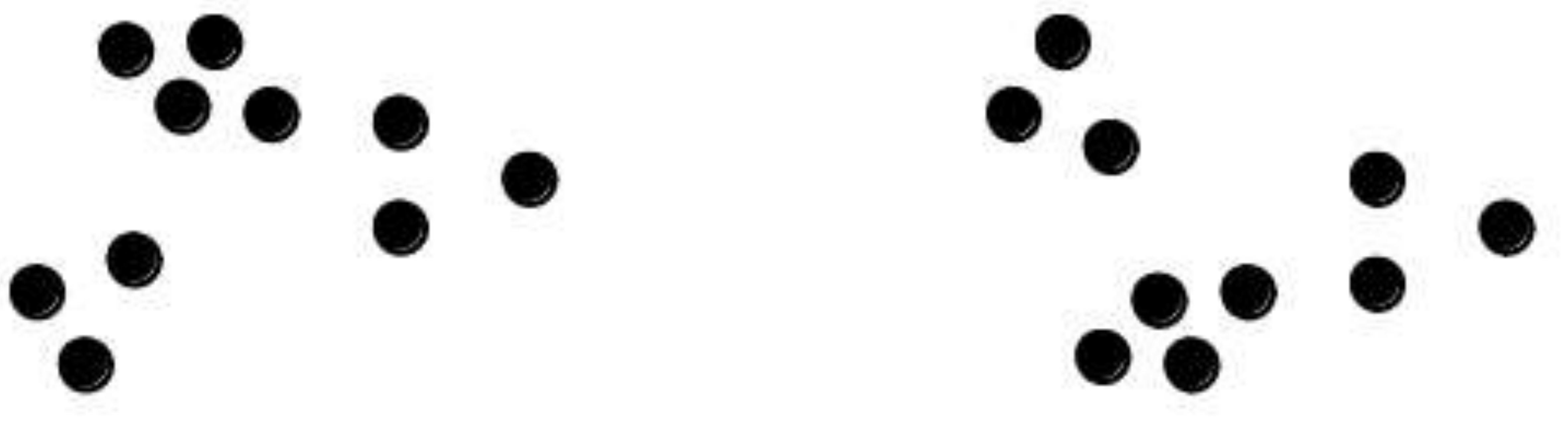
Unsupervised learning

- So far, this course has focused on **supervised learning** methods, such as regression and classification.
- In that setting, we observe both a set of features $X_1, \ldots, X_p$ for each observation and a response variable Y. The goal then is to **predict** Y using $X_1, \ldots, X_p$.
- Now, we will instead focus on **unsupervised learning**, where we **only** have set a of features $X_1, \ldots, X_p$ measured on n observations.
- We are not interested in prediction because we do not have an associated response variable Y. Rather, the goal is to discover **interesting things** about the measurements on $X_1, \ldots, X_p$.
- We will focus on one particular such interesting thing, namely: can we find **subgroups / clusters** in the data?

Unsupervised learning – Clustering

- **Clustering** refers to a broad set of techniques for finding **subgroups** or **clusters** in a data set. Using clustering, we ideally group the observations such that…
  - observations within each group are quite similar to each other and
  - observations in different groups are quite different from each other.
- To make this concrete, we must define what it means for two or more observations to be “similar” or “different”. This is a domain-specific consideration that depends on the data being studied.
- Consider the following application of clustering for **market segmentation**:
  - Say we have access to a large number of measurements of (potential) customers, e.g. income, occupation, past purchase behaviour, etc.
  - We can **perform clustering** to identify subgroups of people who may be more receptive to a particular form of advertising.

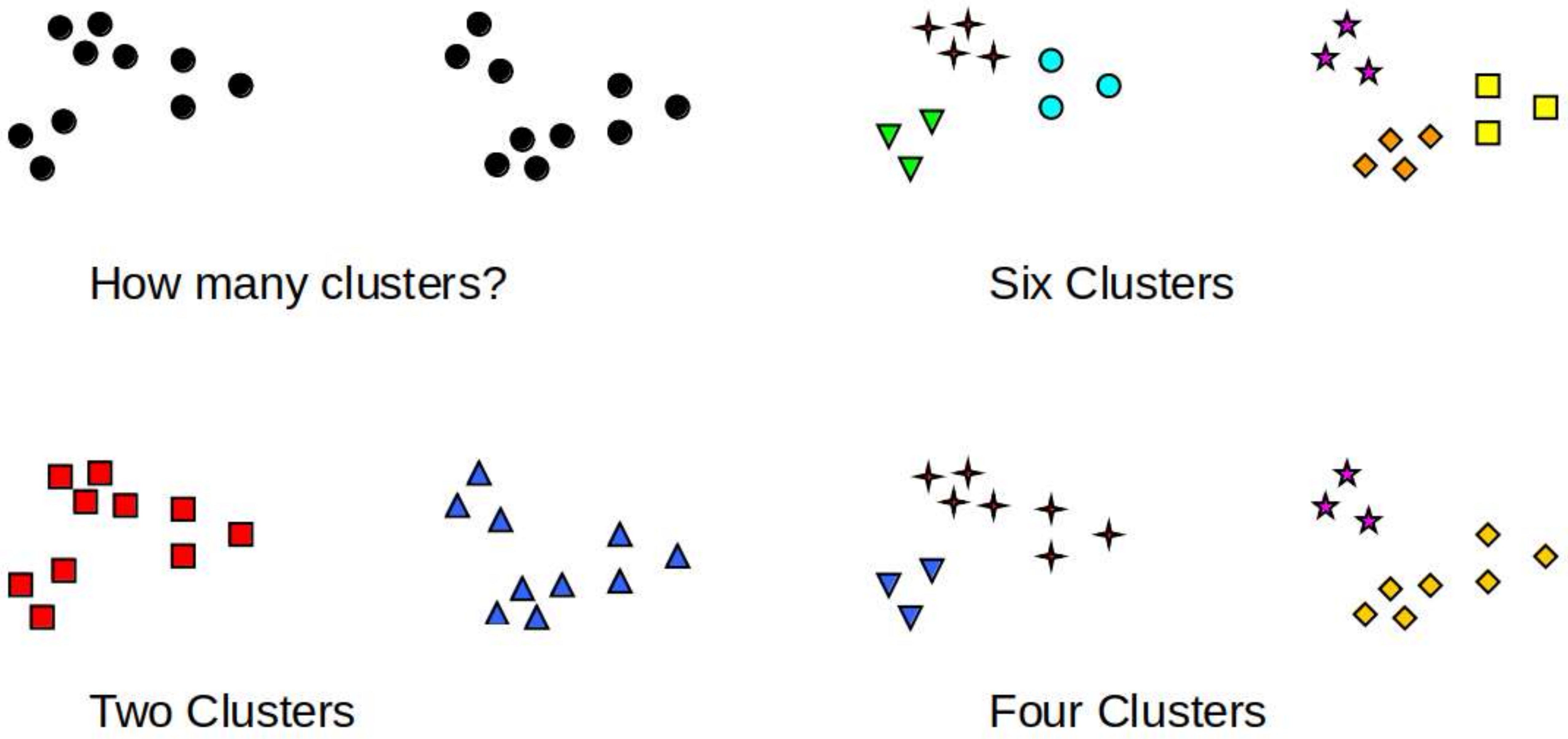
What is a cluster?



How many clusters?

What is a cluster?

Clearly, the notion of a cluster can be **ambiguous**. There is subjective judgement required in deciding what constitutes a cluster:



Clustering methods

- Since clustering is popular in many fields, there exist a great number of clustering methods. Some of the most popular ones are:
  - K-means clustering
  - Hierarchical clustering
  - DBSCAN
  - Model-based clustering
  - …
- As with supervised learning, there is **no free lunch**: clustering methods perform differently on different data sets.
- As an introduction to clustering, we will only be looking more deeply into the first of these methods: **K-means clustering**.

K-means clustering

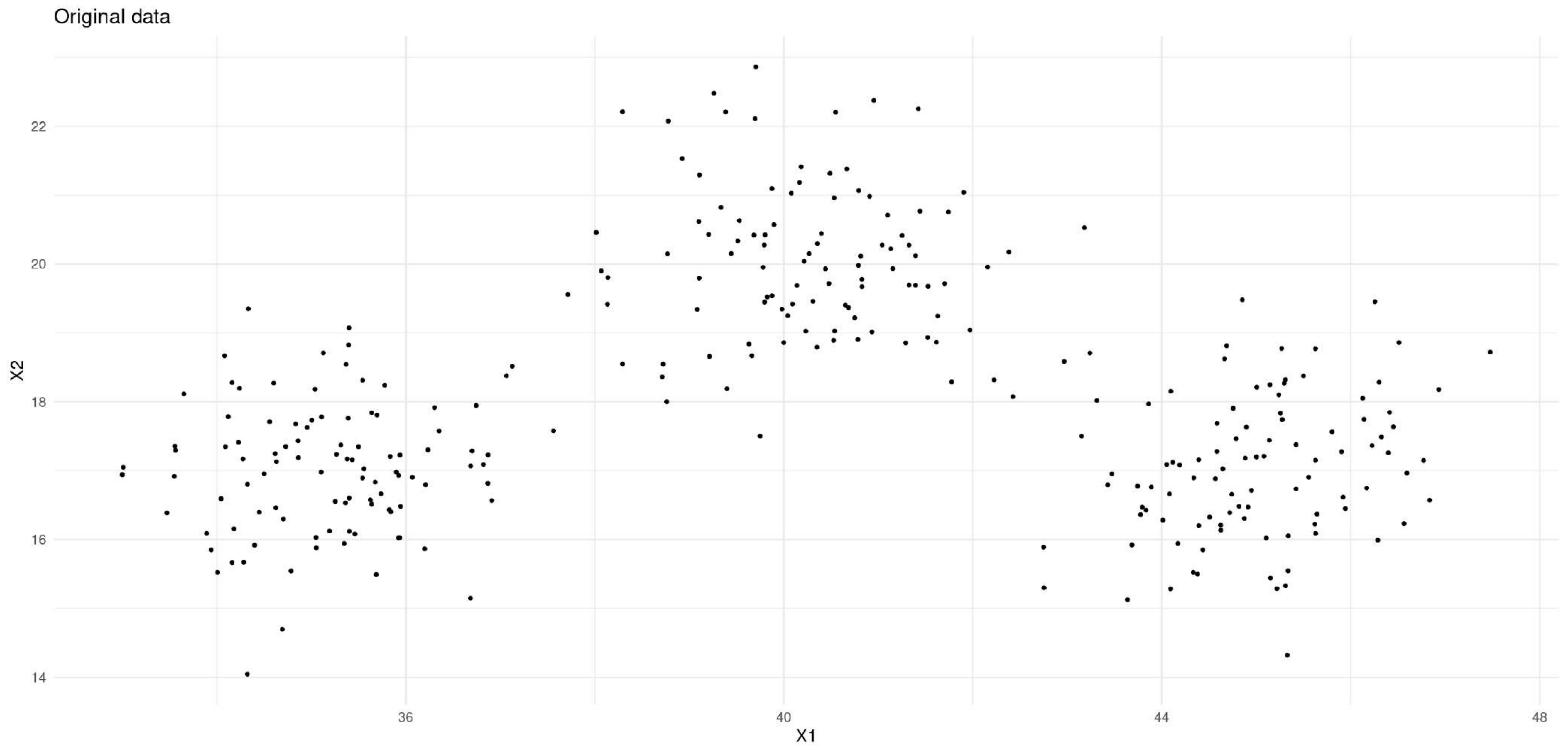
- Given a number of clusters K, **K-means clustering** divides the observations into K distinct, non-overlapping clusters, i.e. each data point is in **exactly one** subset.
- The idea is pretty simple:
  - Each cluster is associated with a so-called **centroid**. It is essentially a cluster's **midpoint**, calculated as the vector of feature means for all the observations in that cluster.
  - Each data point is assigned to the cluster with the closest centroid, where "close" is usually defined based on Euclidean distance, i.e.

$$d(x, y) = \sqrt{(x_1 - y_1)^2 + \ldots + (x_p - y_p)^2}$$

- The algorithm is best understood when illustrating it with an example.

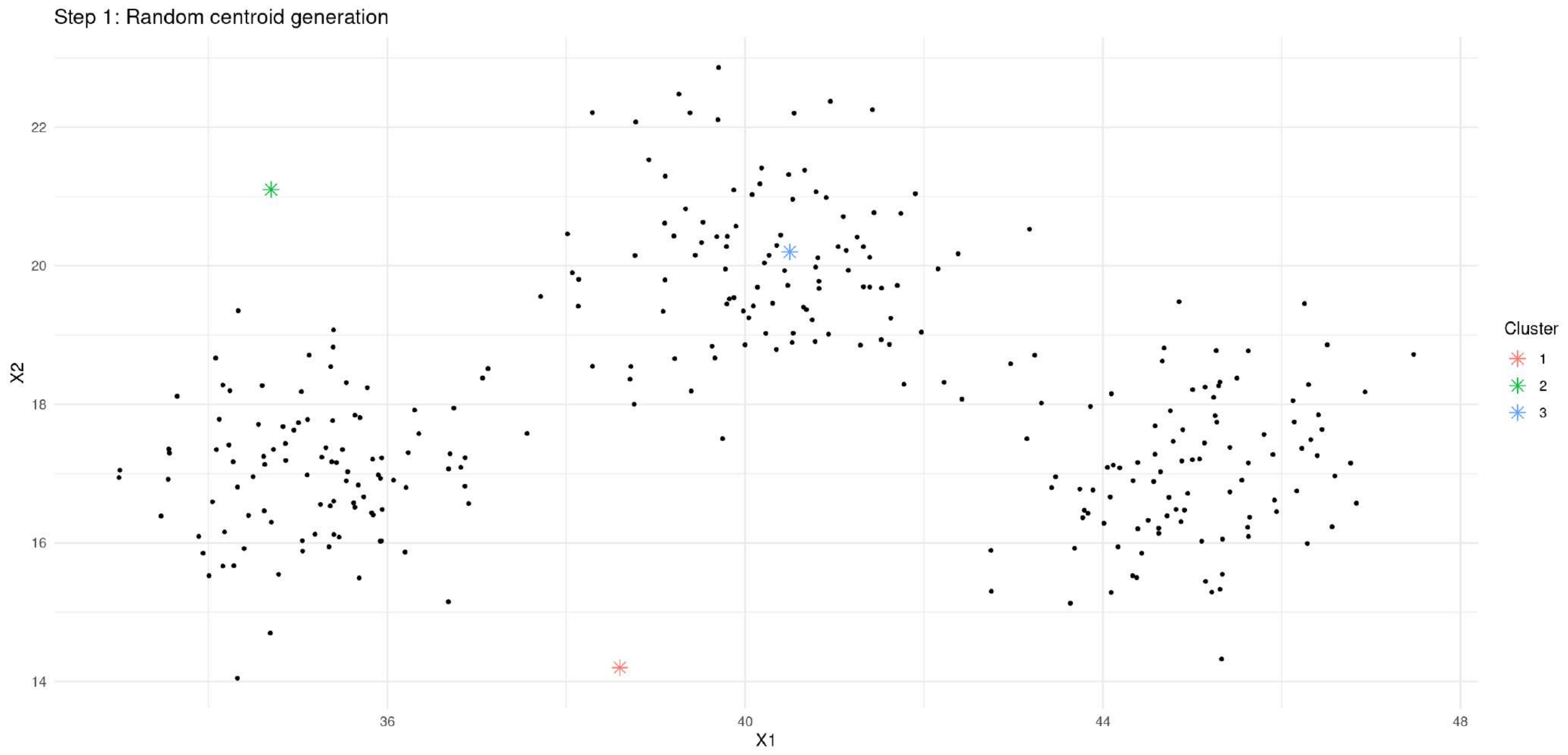
K-means clustering – Example

In the following data set, we have $p = 2$ features named `X1` and `X2`. Based on visual inspection, choosing $K = 3$ for clustering seems appropriate.



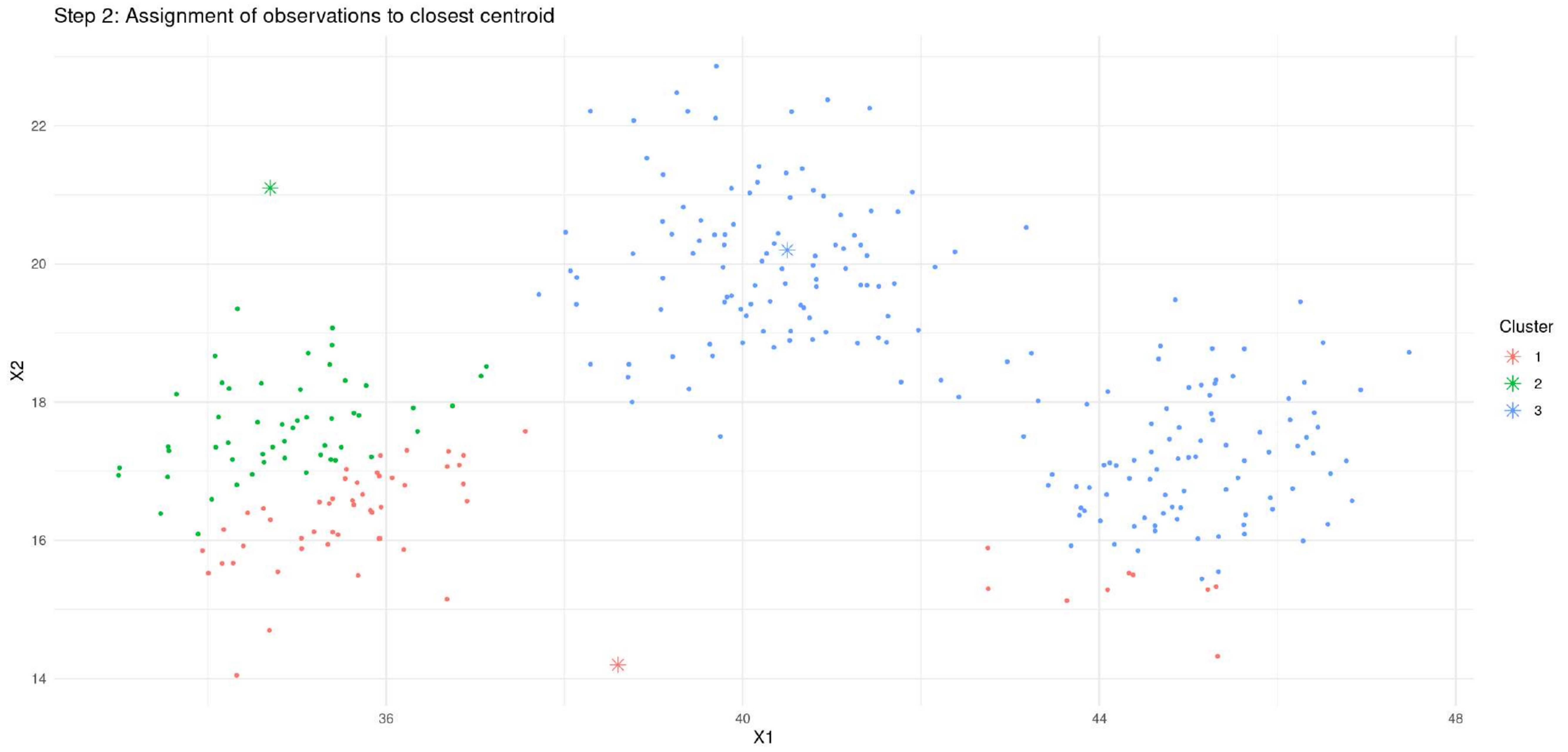
K-means clustering – Example

Step 1: generate K random centroids in the range of the data. We plot the three generated **random initial cluster centroids** as stars into the plot below:



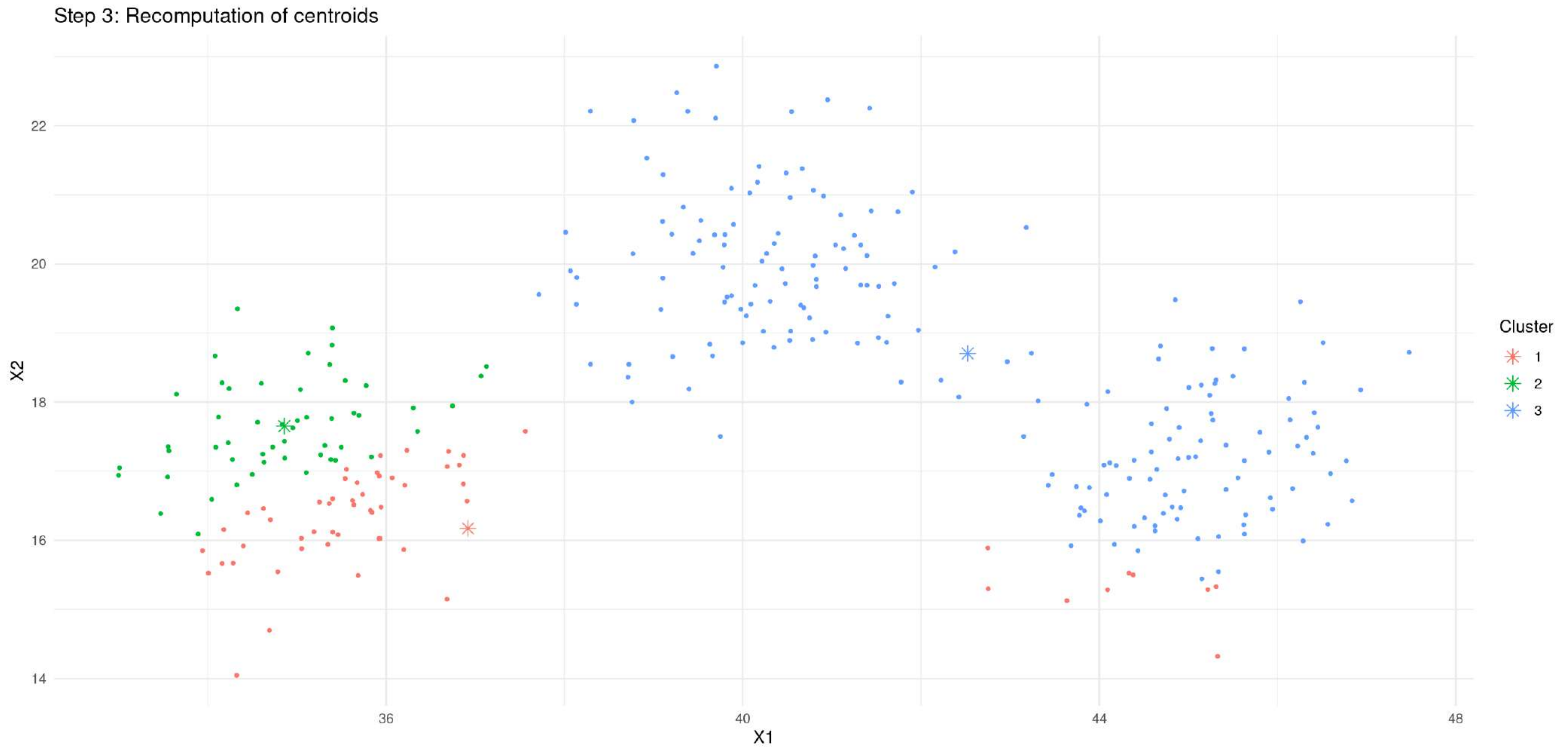
K-means clustering – Example

Step 2: Assign each observation to the centroid that it is closest to. Below, we colour each point in accordance with the colour of its assigned centroid:



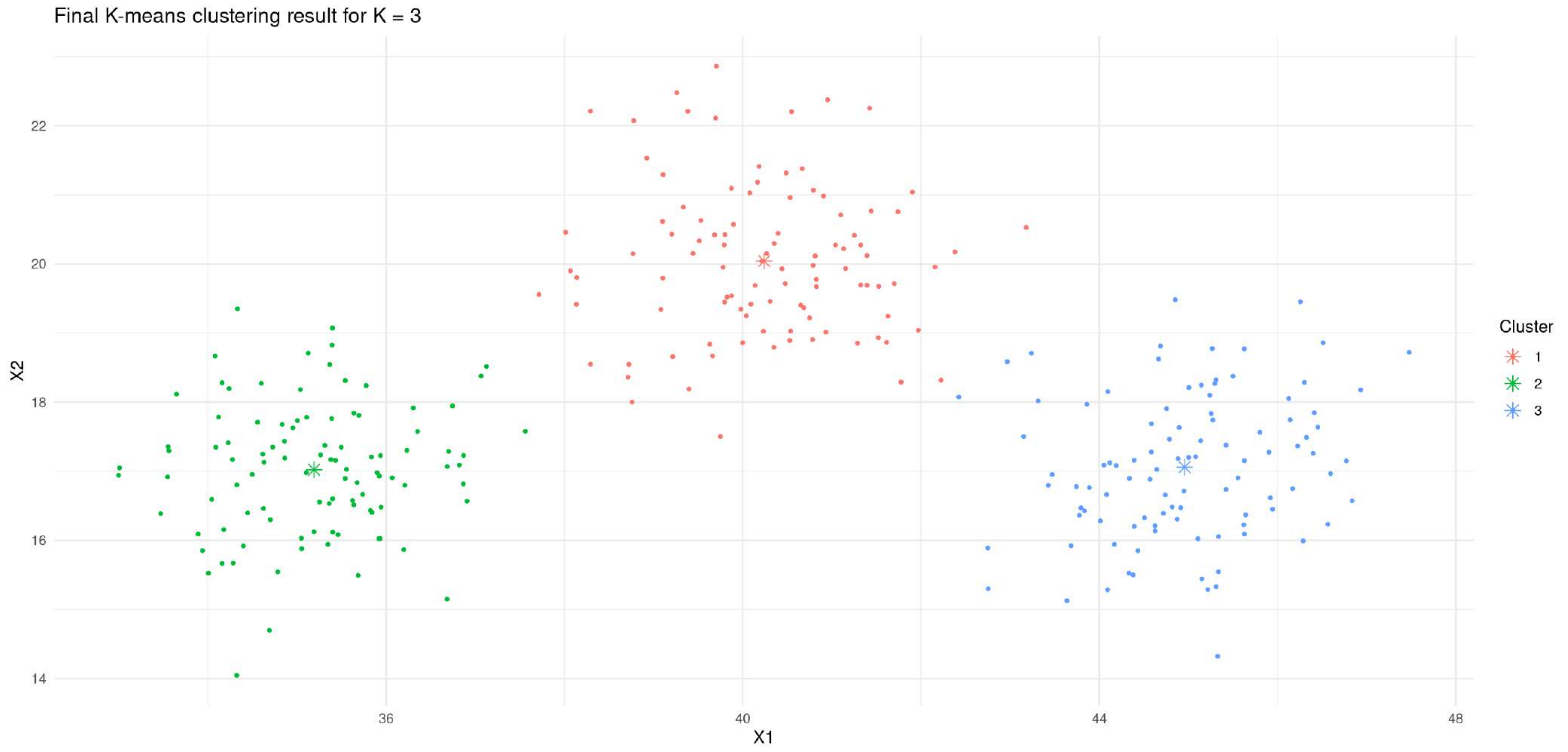
K-means clustering – Example

Step 3: Now, re-compute the centroid of each cluster as the mean (in `X1` and `X2`, respectively) of all observations in a given cluster:



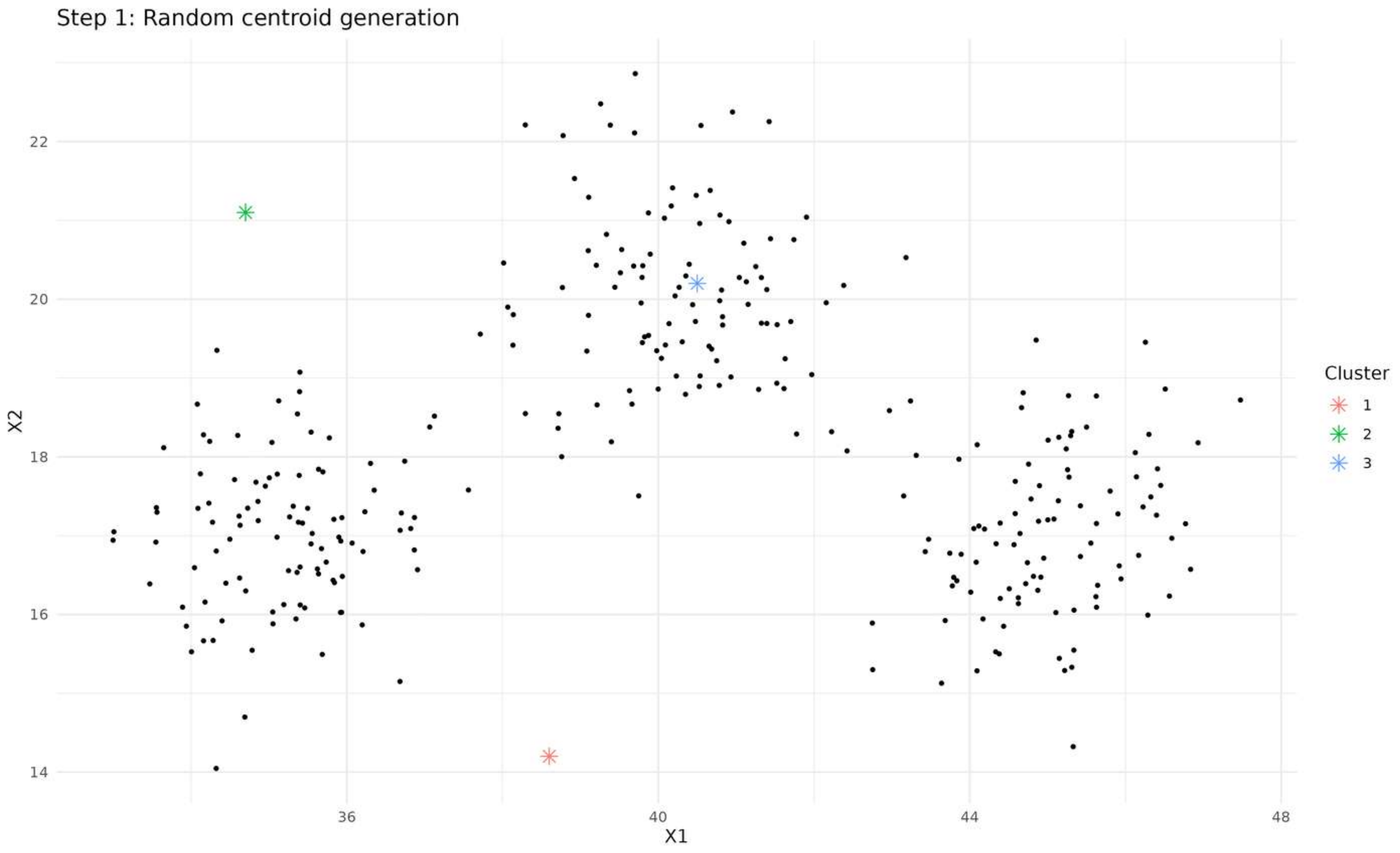
K-means clustering – Example

Step 4: Finally, iterate the assignment of observations to the cluster with the closest centroid and the re-computation of centroids until the clusters are stable:



UNIVERSITY OF SUSTAINABILITY
CHARLOTTE FRESENIUS PRIVATUNIVERSITÄT

K-means clustering – Example

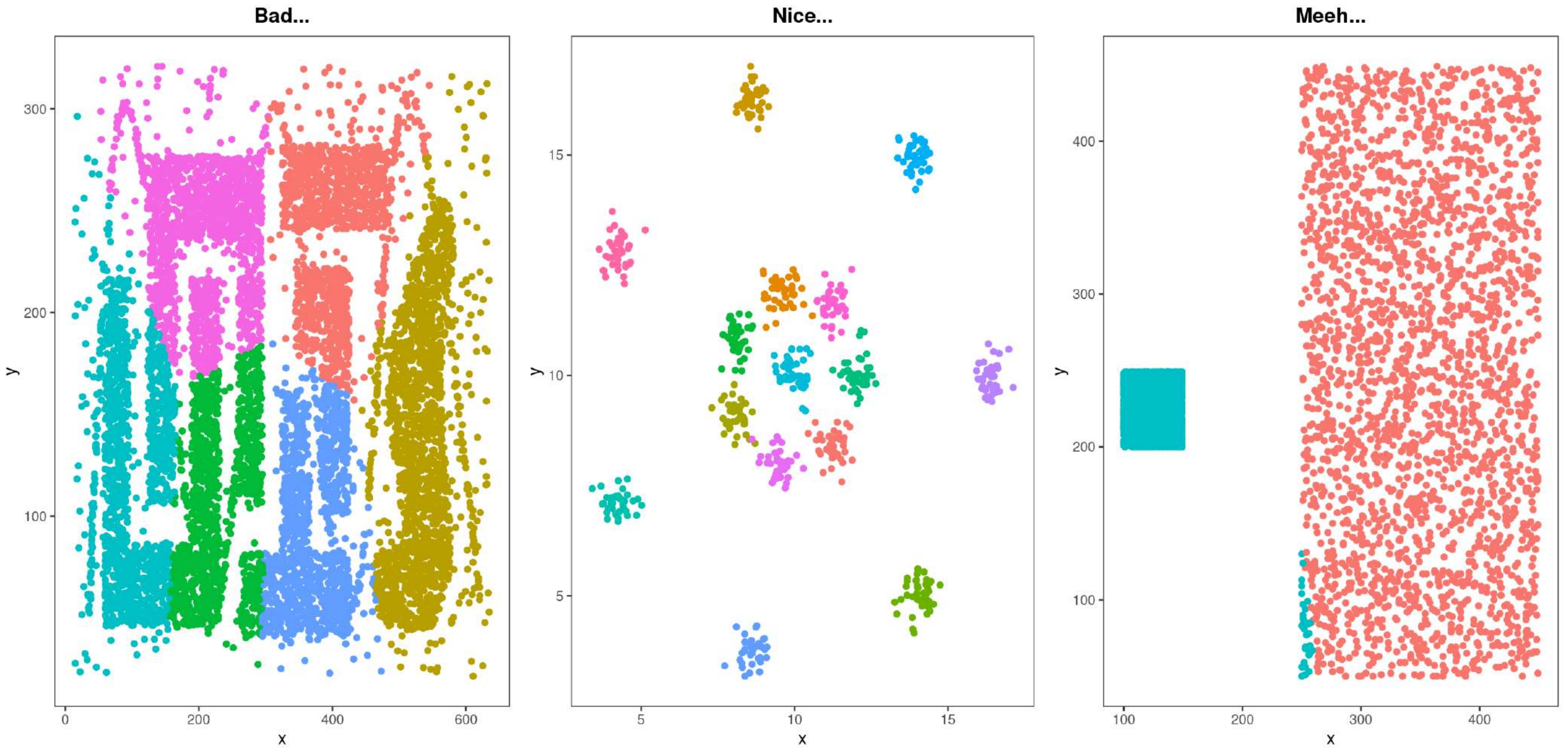


Notes on K-means clustering

- Due to the **randomness** of the initial cluster centroids, K-means clustering is **non-deterministic**, i.e. the final clustering outcome can be different every time the algorithm is run:
  - To mitigate this, we typically run the algorithm on **multiple (random) starting values** and see which run gives the best result.
  - The clusters labels (1, 2, 3) are also random, so they can be different for different runs even when the same observations are clustered together.
- In general, K-means clustering…
  - is a **simple**, easy-to-understand algorithm.
  - works well also in **higher dimensions**.
  - tends to build **spherical** clusters of equal size.

Notes on K-means clustering

With these properties, **K-means clustering** can be expected to perform well on some data sets, whilst performing badly on others:



K-means clustering in R

- Let's see how to run this algorithm on an actual real-world data set in R!
- We use the `mall_customers` data set from Kaggle. This data set contains information on 200 customers of a large mall.
- Besides information on the age, sex and annual income (in thousand USD) of these customers, the data also contains a **spending score** variable based on customer behaviour and purchasing data:

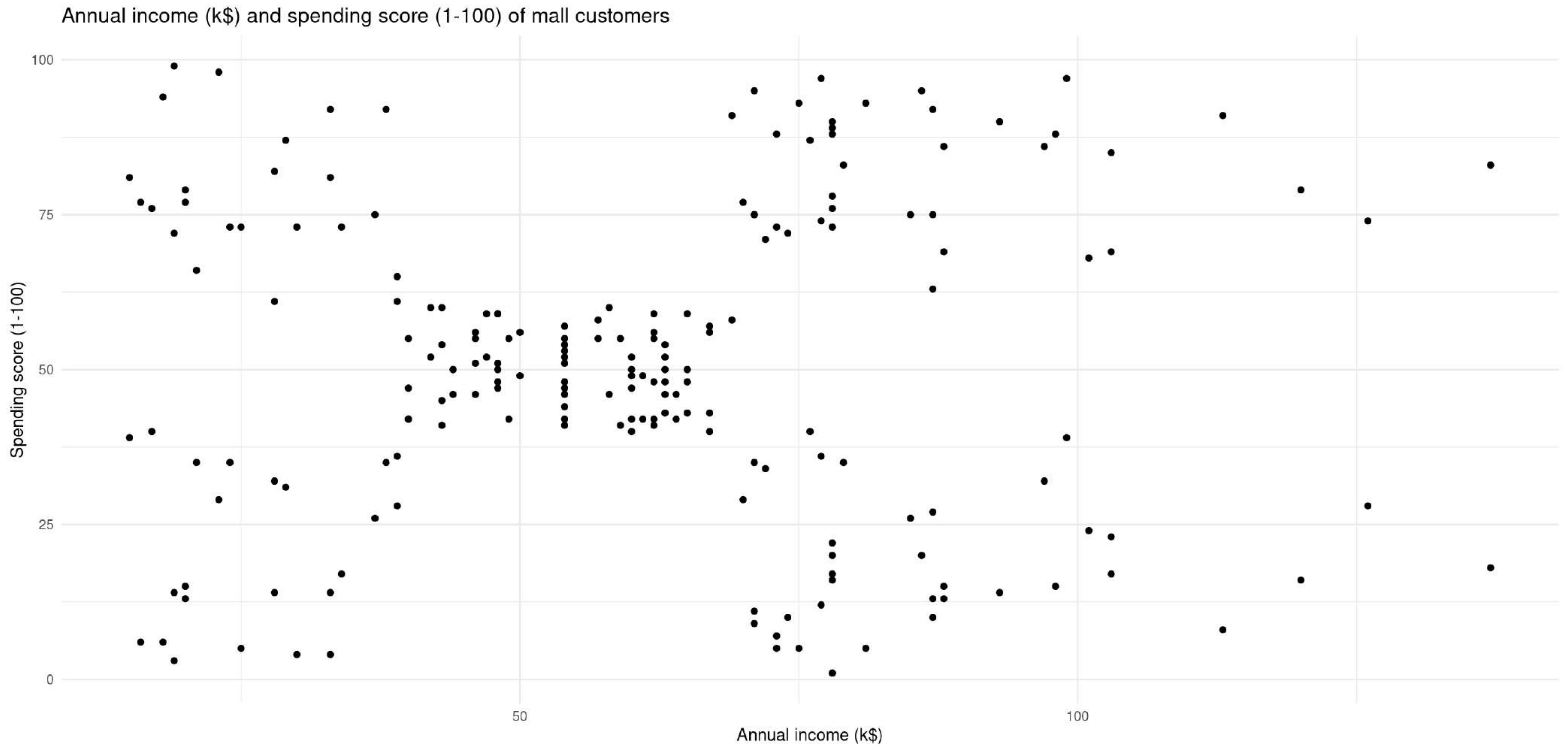
```r
mall <- read.csv("data/mall_customers.csv")
names(mall) <- c("customer_id", "sex", "age", "annual_income", "spending_score")
head(mall)
```

```
  customer_id    sex age annual_income spending_score
1           1   Male  19            15             39
2           2   Male  21            15             81
3           3 Female  20            16              6
4           4 Female  23            16             77
5           5 Female  31            17             40
6           6 Female  22            17             76
```

- Using K-means clustering, we want to perform **market segmentation** based on the variables `annual_income` and `spending_score`.

K-means clustering in R

Let’s start by plotting these two variables in a scatter plot. Based on this plot, a choice of $K = 5$ seems reasonable (more on how to choose K later).



K-means clustering in R

- **Important**: before we apply K-means clustering, we should always **scale / standardize our variables**!
- This is because – as with KNN regression or classification – K-means clustering revolves around computing **distances** between observations, which is highly driven by the scale on which the features are measured.
- To scale our observations, we can either manually subtract the mean and divide by the standard deviation, or use the R function `scale`:

```r
mall_scaled <- scale(mall[, c("annual_income", "spending_score")])
```

- Now, to apply **K-means clustering in R**, we use the function `kmeans`. Besides the (scaled) data, it requires the number of clusters K (argument `centers`) and the number of random starting points (argument `nstart`):

```r
km_mall <- kmeans(mall_scaled, centers = 5, nstart = 10)
```

K-means clustering in R

- Among other things, the resulting `kmeans` object holds the coordinates of the **centroids** (`centers`) and the **clustering vector** (`cluster`), which gives the cluster to which each of the observations was assigned.
- Let's start by looking at the sizes of the clusters:

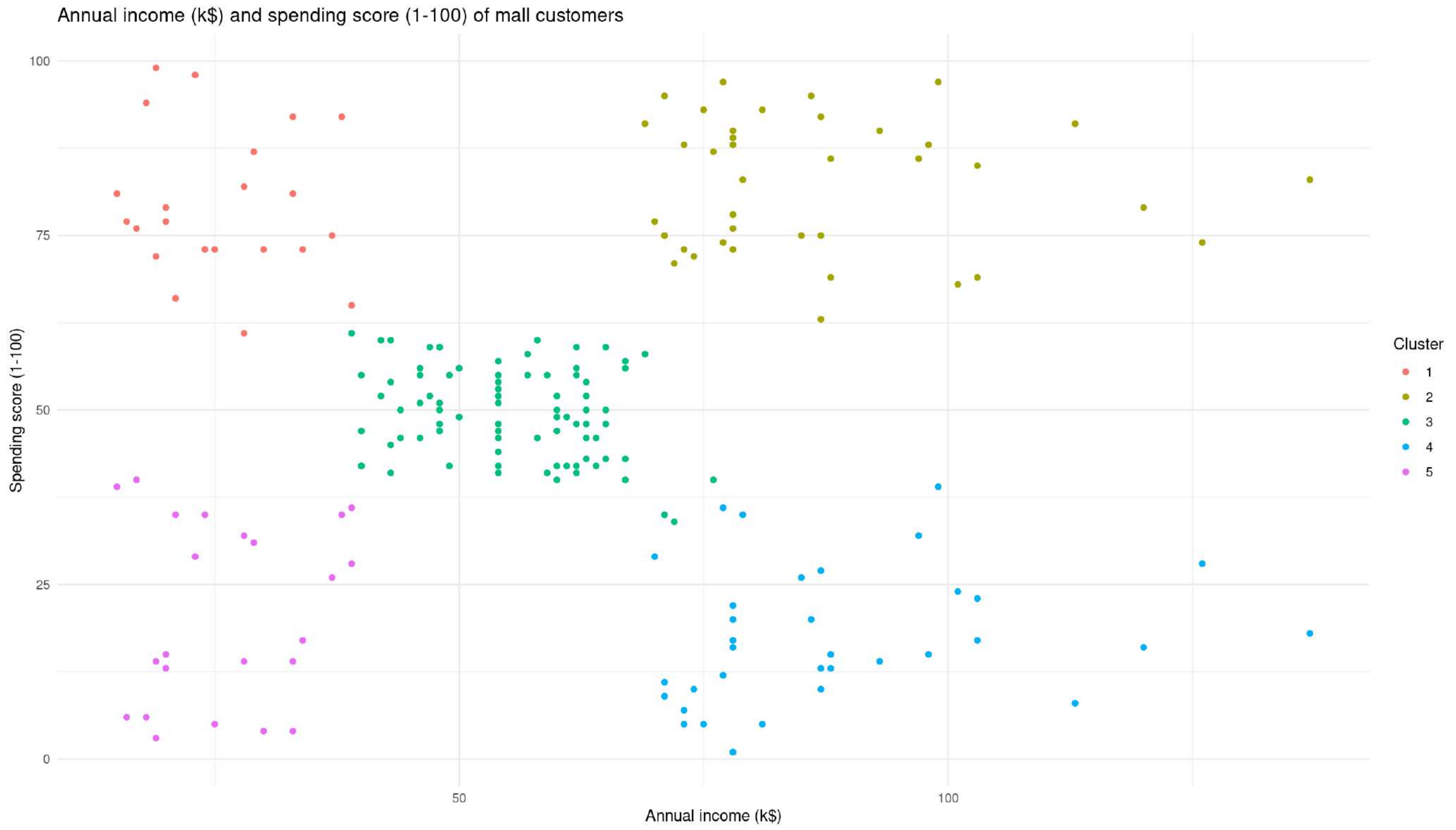
```r
table(km_mall$cluster)
```

```
 1  2  3  4  5 
22 39 81 35 23 
```

- Next, we want to highlight the detected clusters in the original scatter plot using colour (result on the next slide):

```r
mall$Cluster <- factor(km_mall$cluster)

ggplot(mall, aes(x = annual_income, y = spending_score, color = Cluster)) +
  geom_point() +
  labs(title = "Annual income (k$) and spending score (1-100) of mall customers",
       x = "Annual income (k$)", y = "Spending score (1-100)") +
  theme_minimal()
```

K-means clustering in R



Choosing the number of clusters K

- So far, we have determined a suitable number of clusters K by inspecting the data. While this may work when there are 2 or 3 features, it certainly no longer works in higher dimensions.
- Therefore, we need some other metric to assess **clustering quality**. One commonly used metric is the **within-cluster sum of squares (WCSS)**:

$$WCSS = \sum_{k=1}^{K} \sum_{i \in C_k} \sum_{j=1}^{p} (x_{ij} - \bar{x}_{kj})^2$$

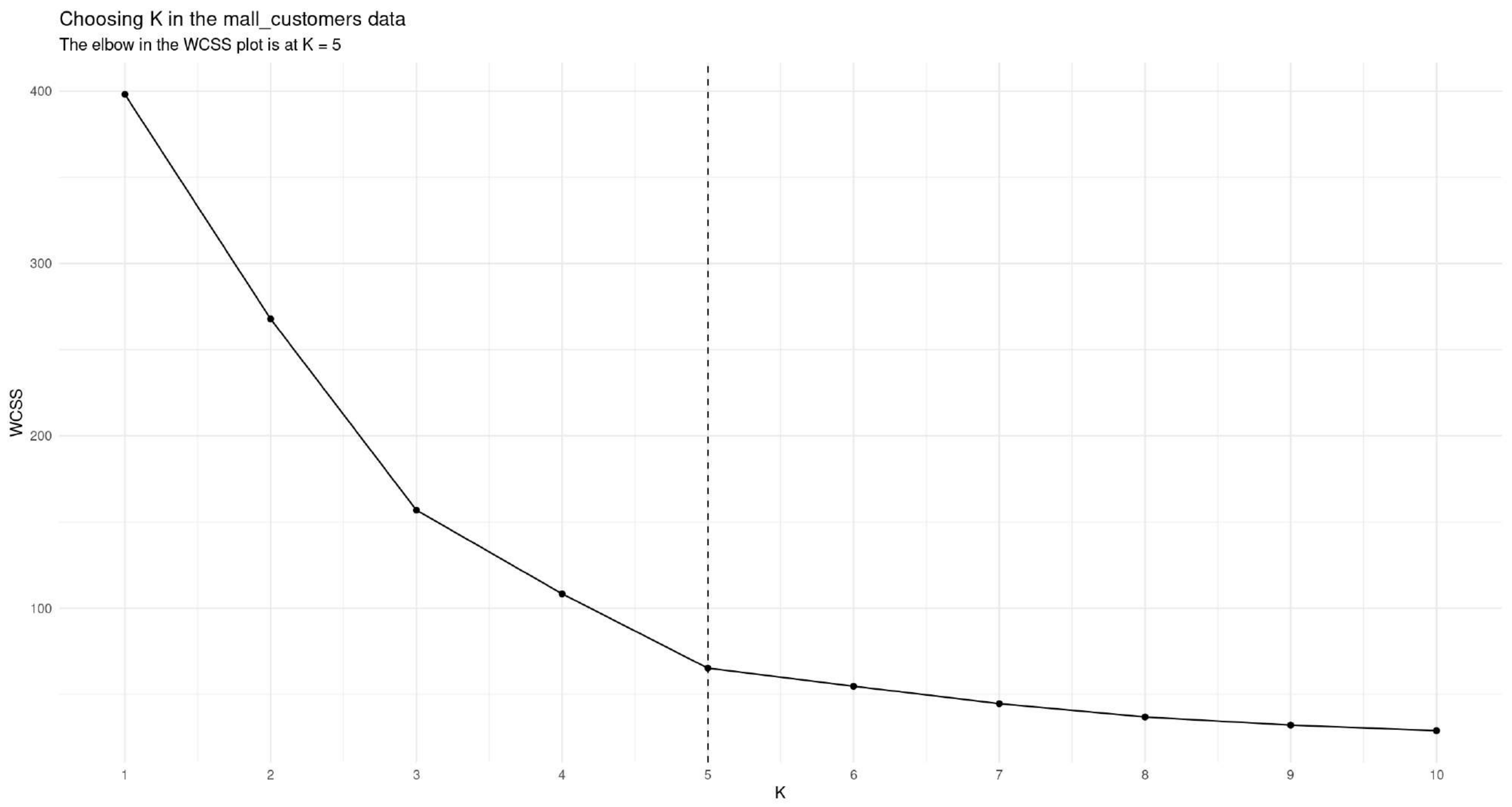
- The WCSS measures how much the observations **within a given cluster** differ from each other:
  - It is small when the observations a very **tightly packed** around the centroid.
  - It is large when they are **widely spread** around the centroid.

Choosing the number of clusters K

- Therefore, we want a clustering with as small a $WCSS$ as possible. In fact, K-means clustering is designed to minimize $WCSS$.
- Typically (although not necessarily), the $WCSS$ decreases as we increase the number of clusters.
- Consequently, one way of choosing K is a heuristic called the **elbow method**:
  - Run K-means with different values of K and record the $WCSS$ for each.
  - Plot the number of clusters against the $WCSS$ and find the “elbow” in the plot, i.e. the point where the reduction in $WCSS$ for an increase in K becomes very small. Choose K accordingly.
- While this heuristic is easy to understand, it is not always unambiguous and does require some **subjective judgement**.

Choosing the number of clusters K

Plot Code


Choosing K in the mall_customers data
The elbow in the WCSS plot is at K = 5
WCSS
K