Introduction to regression

SUPERVISED LEARNING WITH SCIKIT-LEARN

George Boorman
Core Curriculum Manager, DataCamp

Predicting blood glucose levels

```python
import pandas as pd
diabetes_df = pd.read_csv("diabetes.csv")
print(diabetes_df.head())
```

```
    pregnancies    glucose    triceps    insulin    bmi     age    diabetes
0   6              148        35         0          33.6    50     1
1   1              85         29         0          26.6    31     0
2   8              183        0          0          23.3    32     1
3   1              89         23         94         28.1    21     0
4   0              137        35         168        43.1    33     1
```

Creating feature and target arrays

```python
X = diabetes_df.drop("glucose", axis=1).values
y = diabetes_df["glucose"].values
print(type(X), type(y))
```

```
<class 'numpy.ndarray'> <class 'numpy.ndarray'>
```

To use all the features in our dataset, we drop our target, blood glucose levels, and store the values attribute as X. For y, we take the target columns value attribute.

Making predictions from a single feature

```python
X_bmi = X[:, 3]
print(y.shape, X_bmi.shape)
```

Here we slice out the BMI column of X which is the 4th column.

```
(752,) (752,)
```

When you check the shape you can see that both are one dimensional arrays. This is fine for y, but our features must be formatted as a 2 dimensional array to be accepted by scikit-learn.

```python
X_bmi = X_bmi.reshape(-1, 1)
print(X_bmi.shape)
```

-1 in reshape function is used when you don't know or want to explicitly tell the dimension of that axis.

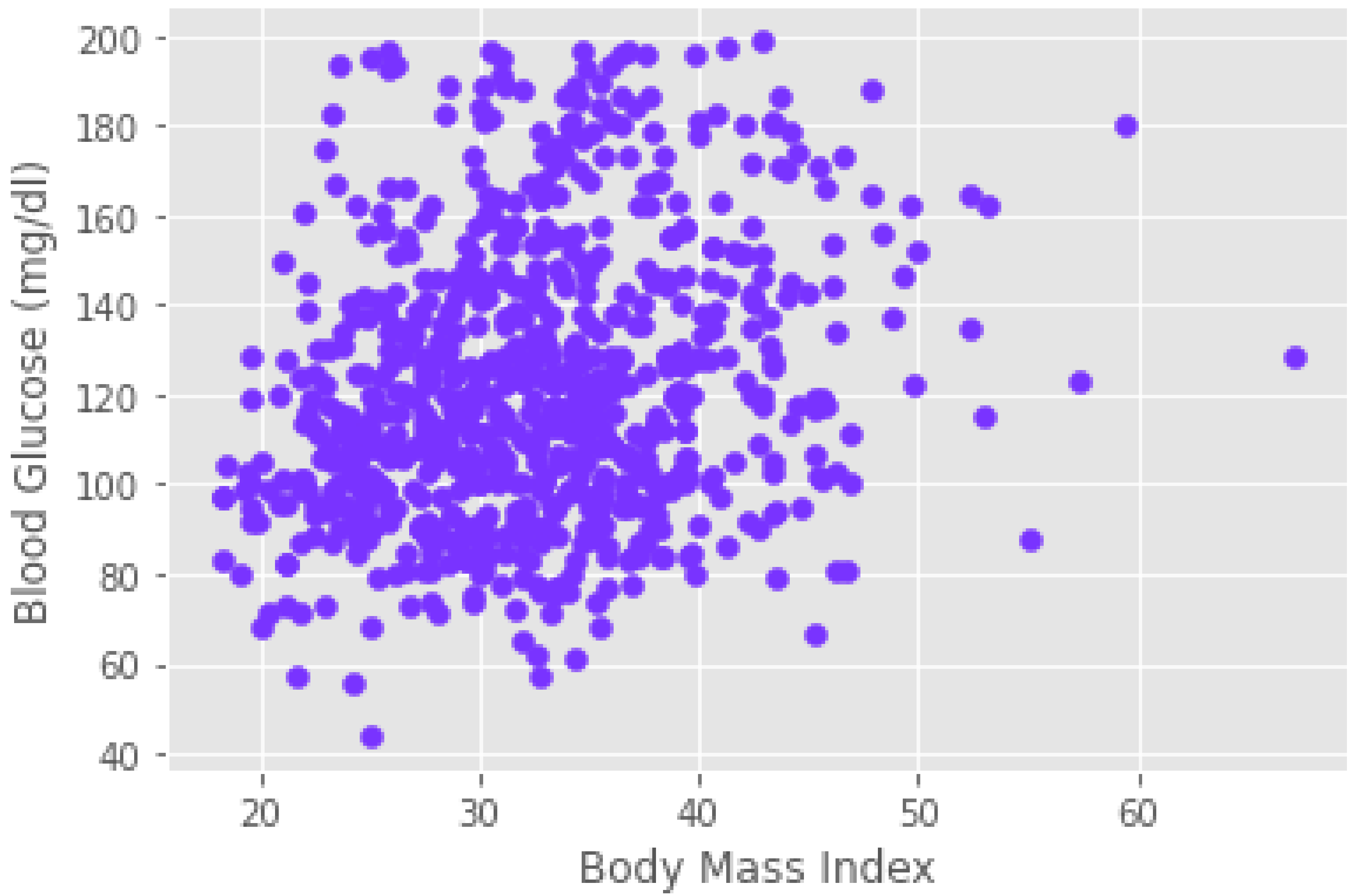
```
(752, 1)
```

E.g,
If you have an array of shape (2,4) then reshaping it with (-1, 1), then the array will get reshaped in such a way that the resulting array has only 1 column and this is only possible by having 8 rows, hence, (8,1).

Plotting glucose vs. body mass index

```python
import matplotlib.pyplot as plt
plt.scatter(X_bmi, y)
plt.ylabel("Blood Glucose (mg/dl)")
plt.xlabel("Body Mass Index")
plt.show()
```

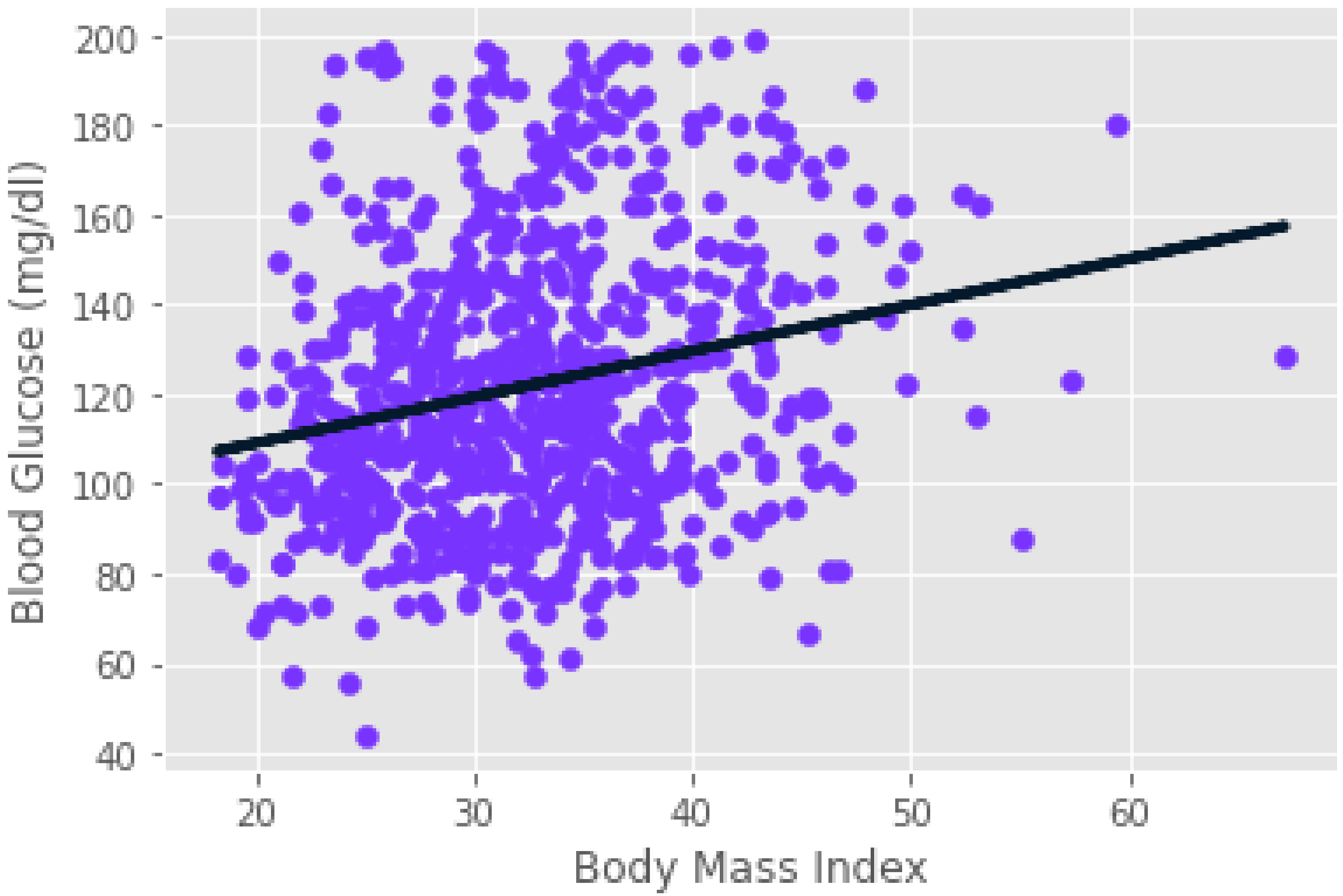
Plotting glucose vs. body mass index



Fitting a regression model

```
from sklearn.linear_model import LinearRegression
reg = LinearRegression()
reg.fit(X_bmi, y)
predictions = reg.predict(X_bmi)
plt.scatter(X_bmi, y)
plt.plot(X_bmi, predictions)
plt.ylabel("Blood Glucose (mg/dl)")
plt.xlabel("Body Mass Index")
plt.show()
```

Fitting a regression model



Let's practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

The basics of linear regression

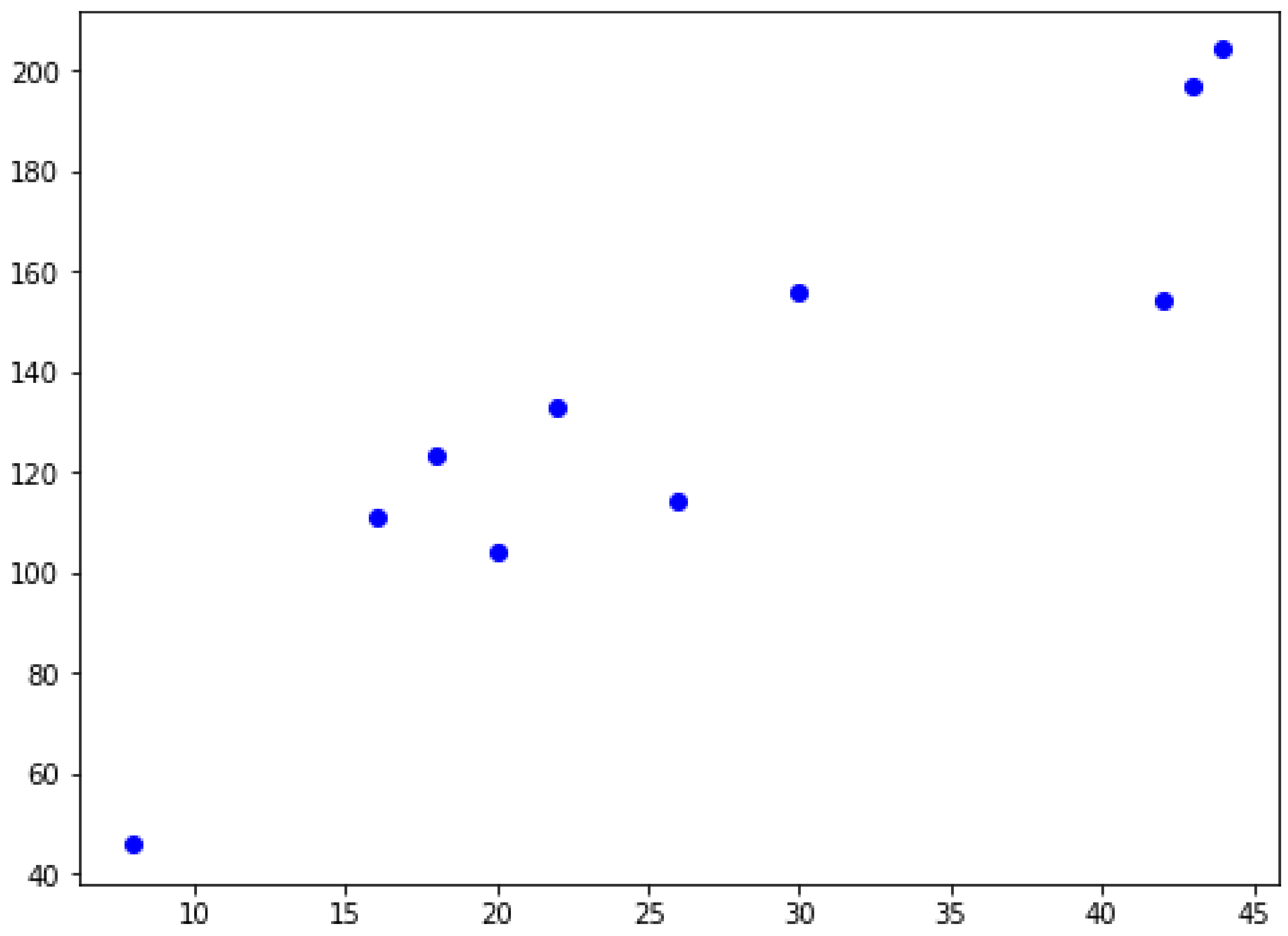
SUPERVISED LEARNING WITH SCIKIT-LEARN

George Boorman
Core Curriculum Manager, DataCamp

Regression mechanics

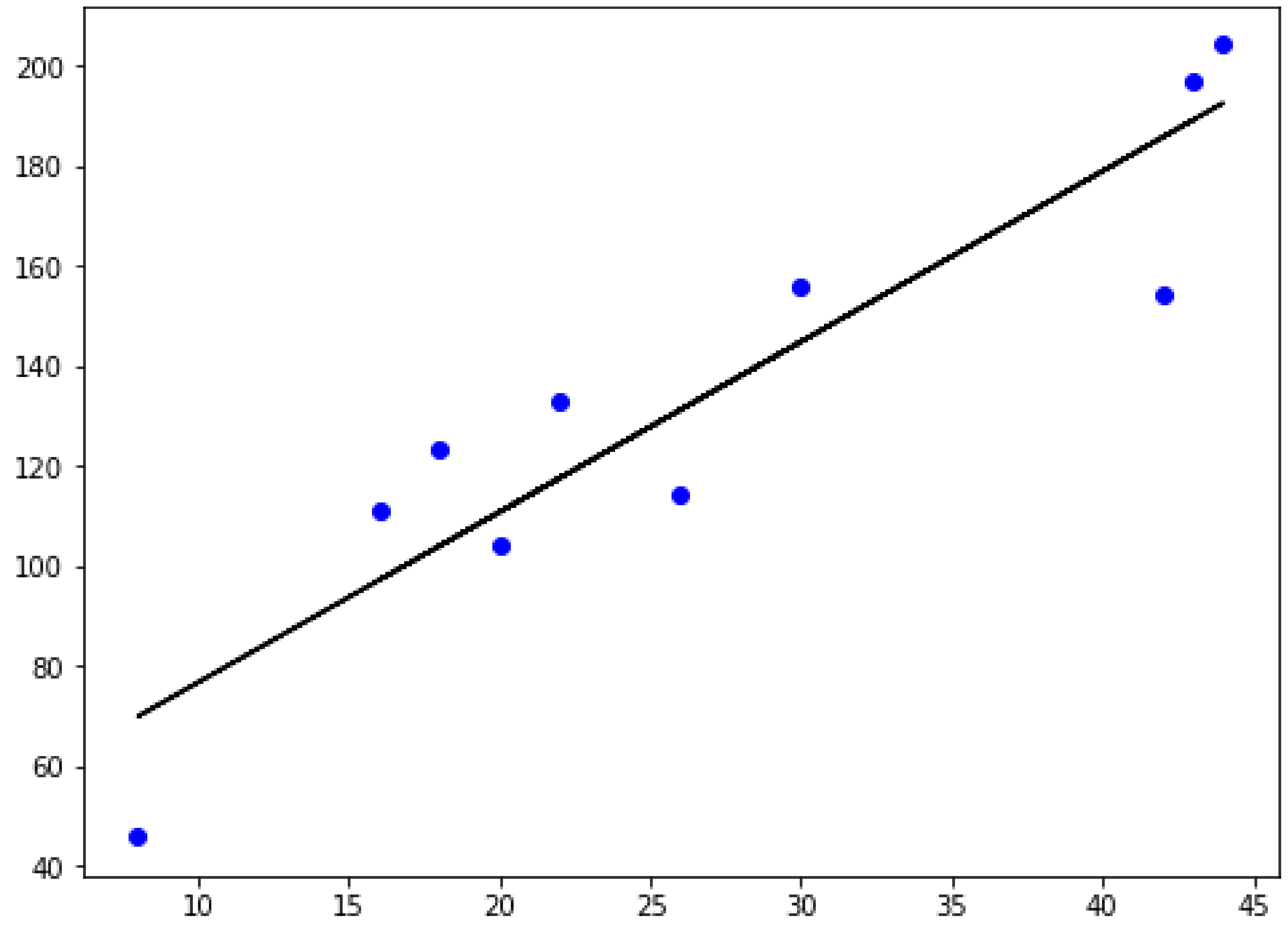
- $y = ax + b$
  - Simple linear regression uses one feature
    - y = target
    - x = single feature
    - a, b = parameters/coefficients of the model - slope, intercept
- How do we choose a and b?
  - Define an error function for any given line
  - Choose the line that minimizes the error function
- Error function = loss function = cost function

The loss function



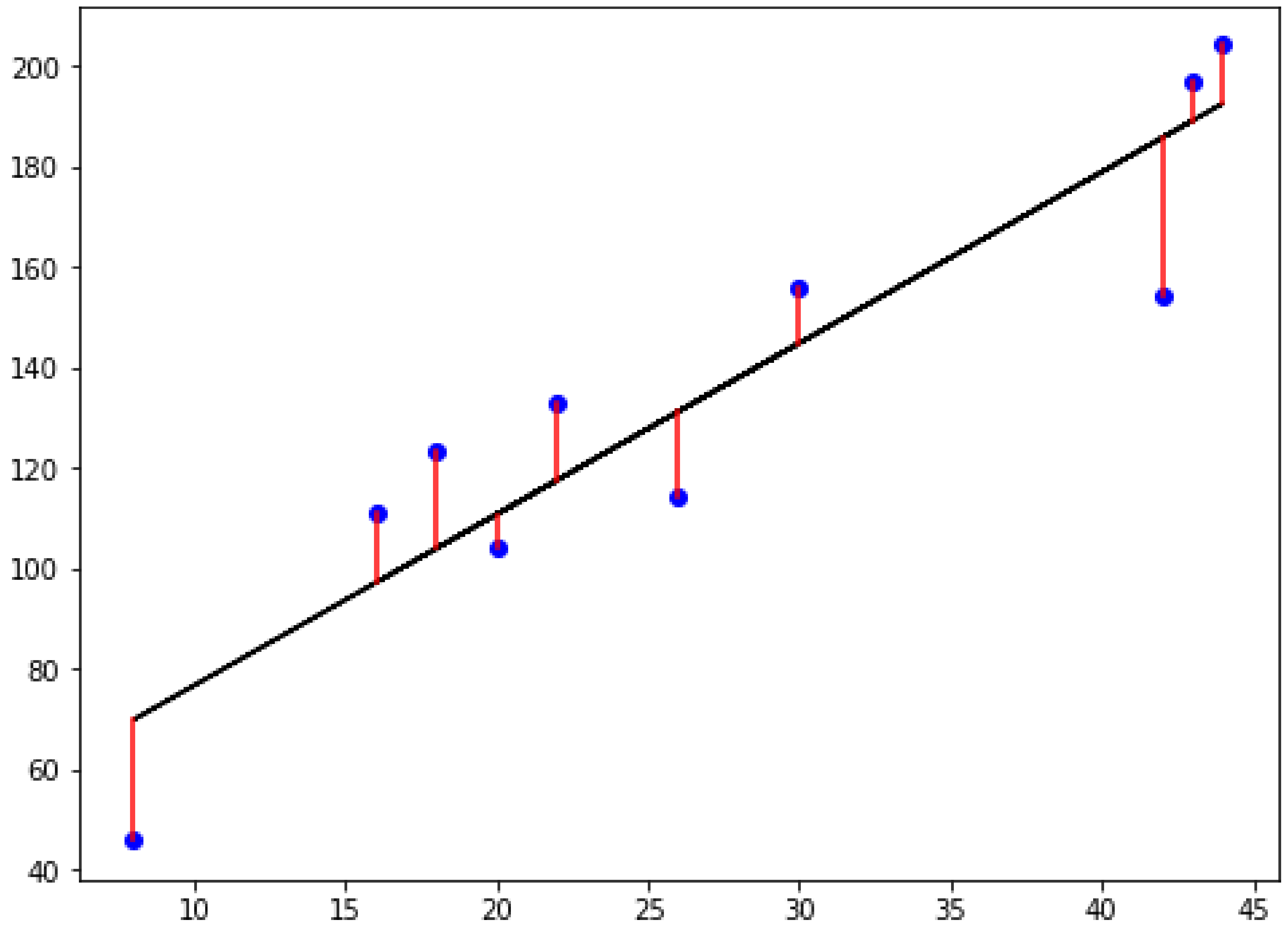
The loss function

Let's visualize a loss function using this scatter plot. We want the line to be as close to the observations as possible. Therefore, we want to minimize the vertical distance between the fit and the data.

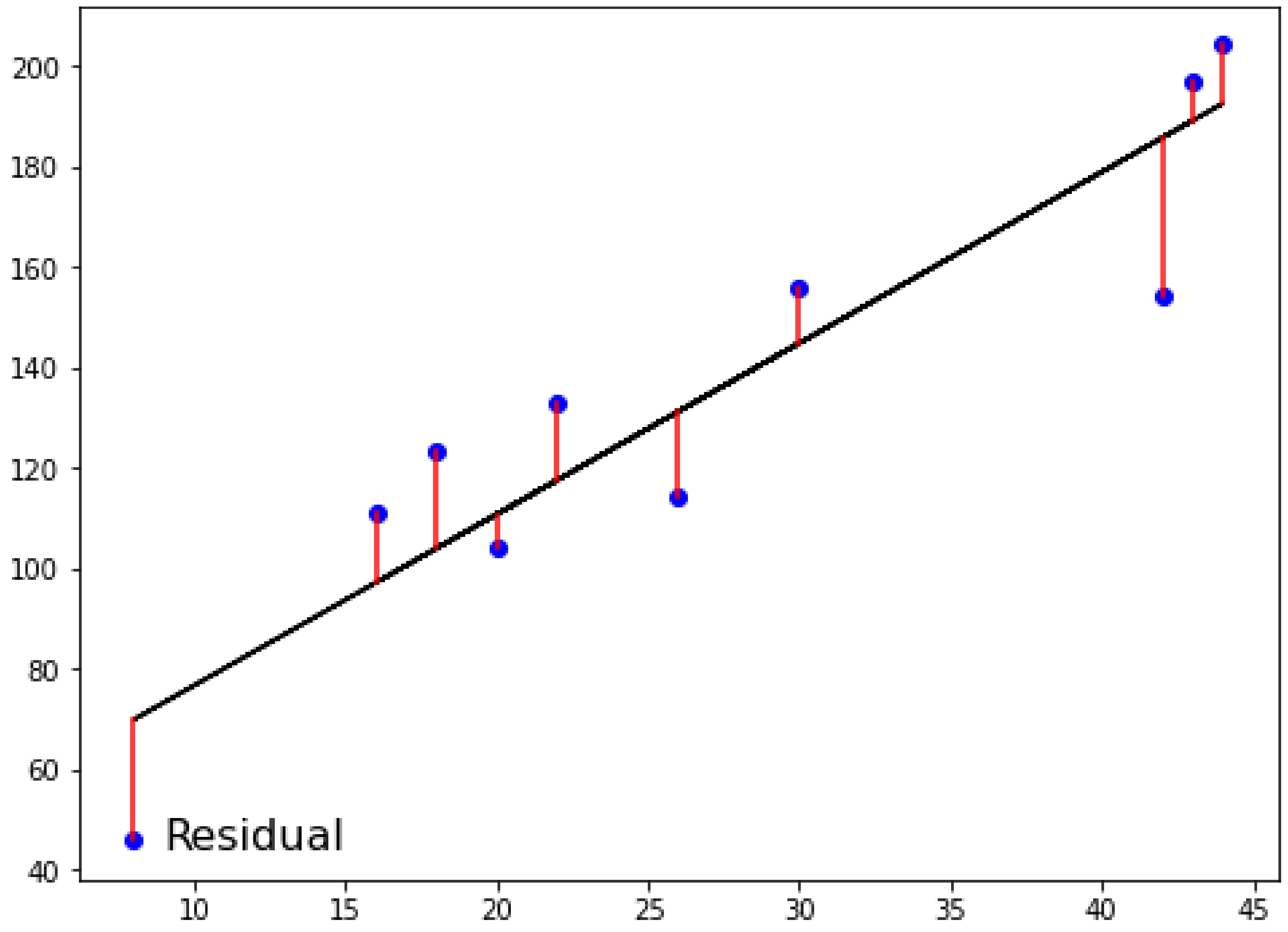


The loss function

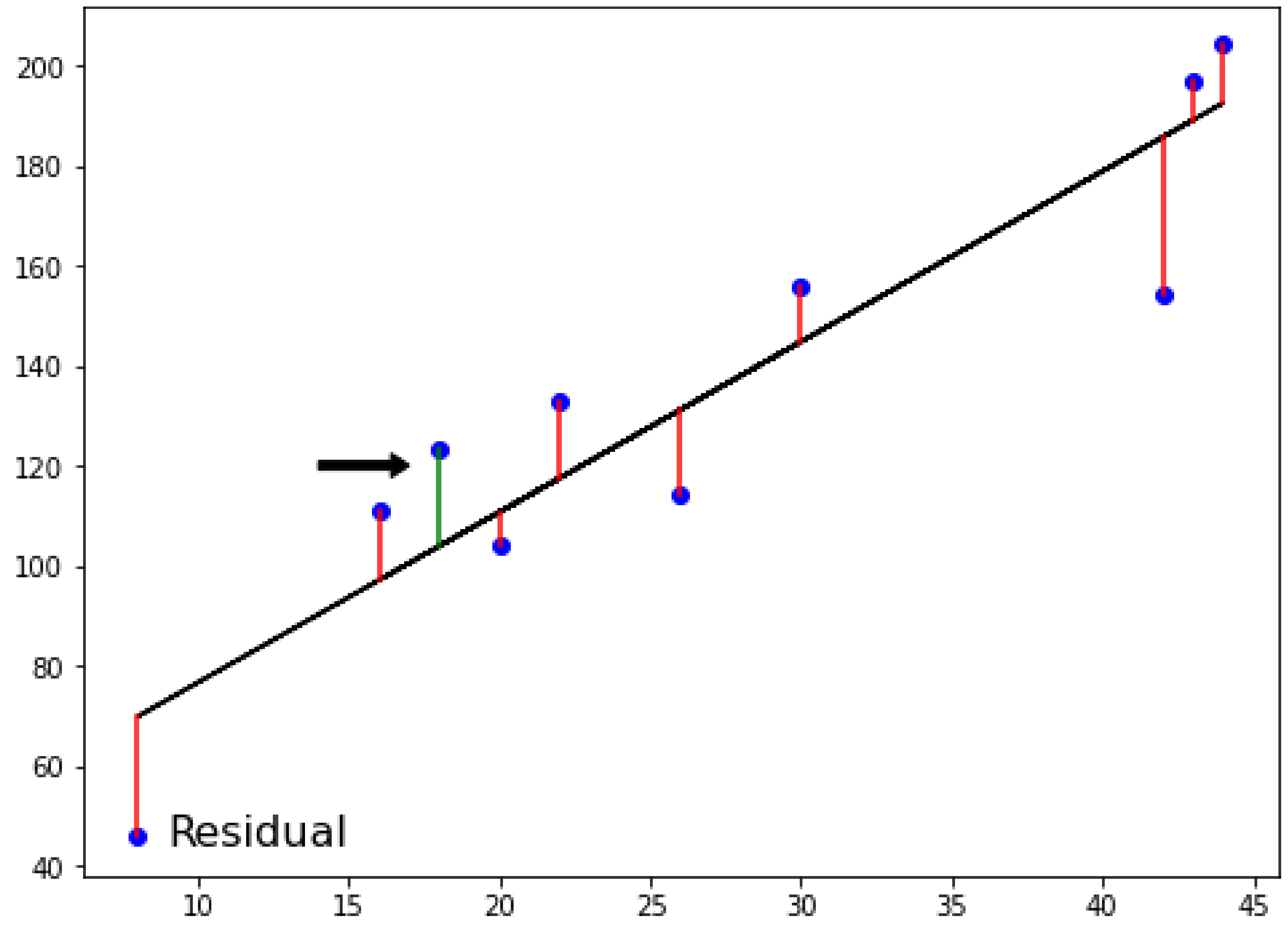
So for each observation we calculate the vertical distance between it and the line. We call this distance the residual. We could try to minimize the sum of the residuals. But then each positive residual would cancel out each negative residual. So we square the residuals.



The loss function

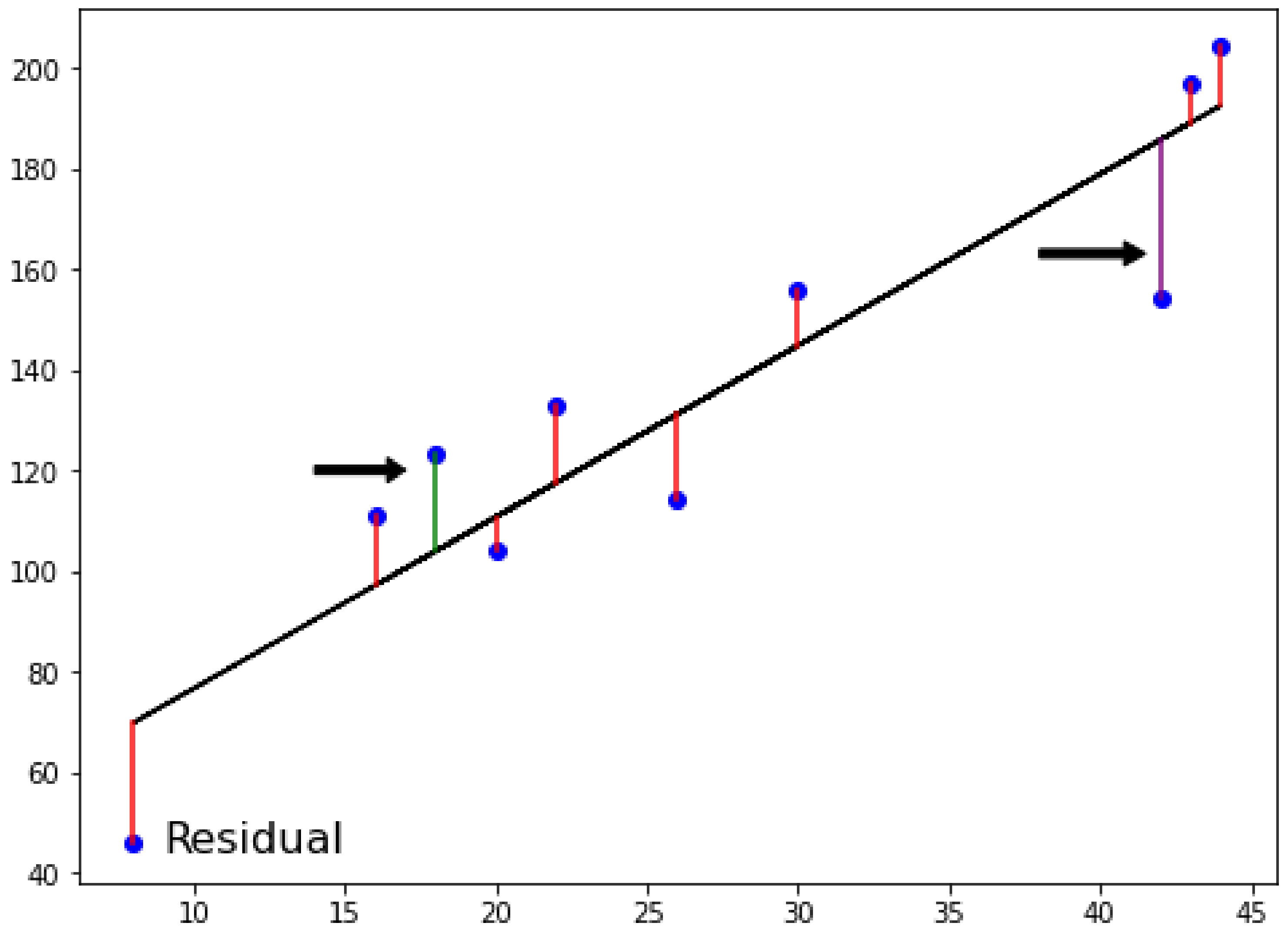


The loss function



Ordinary Least Squares

By adding all the squared residuals we calculate this.



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

Ordinary Least Squares (OLS): minimize RSS

Linear regression in higher dimensions

$$y = a_1x_1 + a_2x_2 + b$$

- To fit a linear regression model here:
  - Need to specify 3 variables: $a_1,\ a_2,\ b$
- In higher dimensions:
  - Known as multiple regression
  - Must specify coefficients for each feature and the variable b

$$y = a_1x_1 + a_2x_2 + a_3x_3 + ... + a_nx_n + b$$

- scikit-learn works exactly the same way:
  - Pass two arrays: features and target

Linear regression using all features

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
                                                    random_state=42)
reg_all = LinearRegression()
reg_all.fit(X_train, y_train)
y_pred = reg_all.predict(X_test)
```

Note that linear regression in scikit-learn performs OLS (Ordinary Least Squares) under the hood.

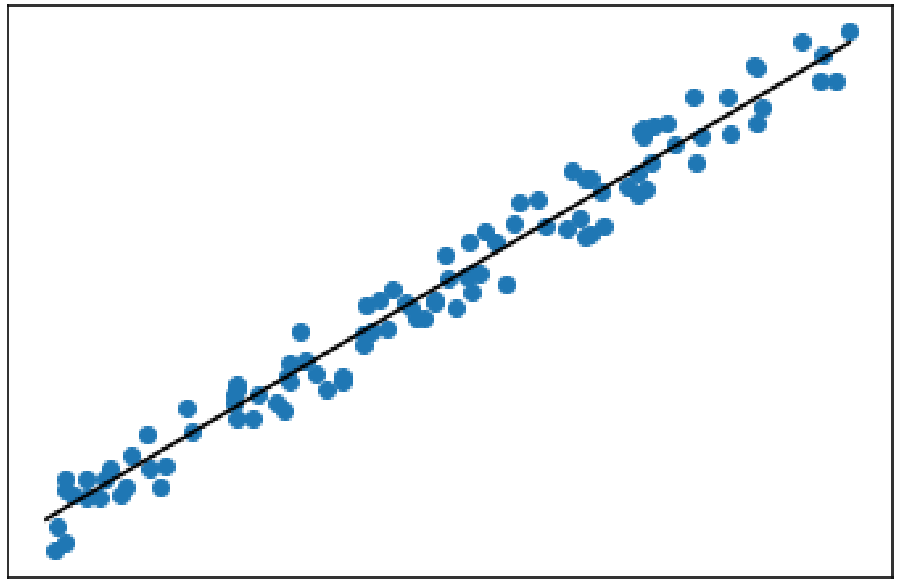
R-squared

The default metric for linear regression is R squared.

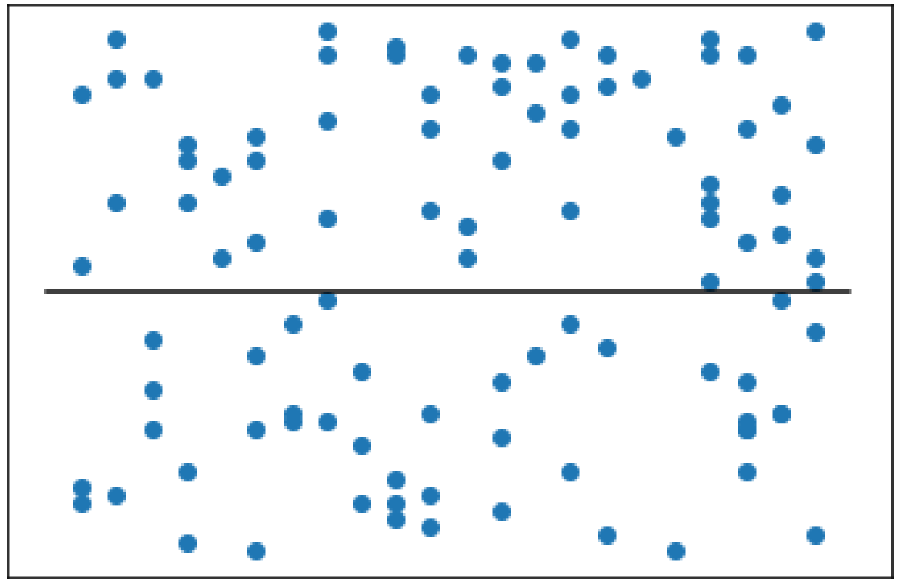
- R^2: quantifies the variance in target values explained by the features
  - Values range from 0 to 1

1 means the features completely explain the target variance.

- High R^2:



- Low R^2:



R-squared in scikit-learn

```
reg_all.score(X_test, y_test)
```

```
0.356302876407827
```

Mean squared error and root mean squared error

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Here the mean of the residual sum of squares is taken. Known as the Mean Sqaures Error.

- MSE is measured in target units, squared

$$RMSE = \sqrt{MSE}$$

- Measure $RMSE$ in the same units at the target variable

RMSE in scikit-learn

```python
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, y_pred, squared=False)
```

```
24.028109426907236
```

Let's practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

Cross-validation

SUPERVISED LEARNING WITH SCIKIT-LEARN

George Boorman
Core Curriculum Manager, DataCamp

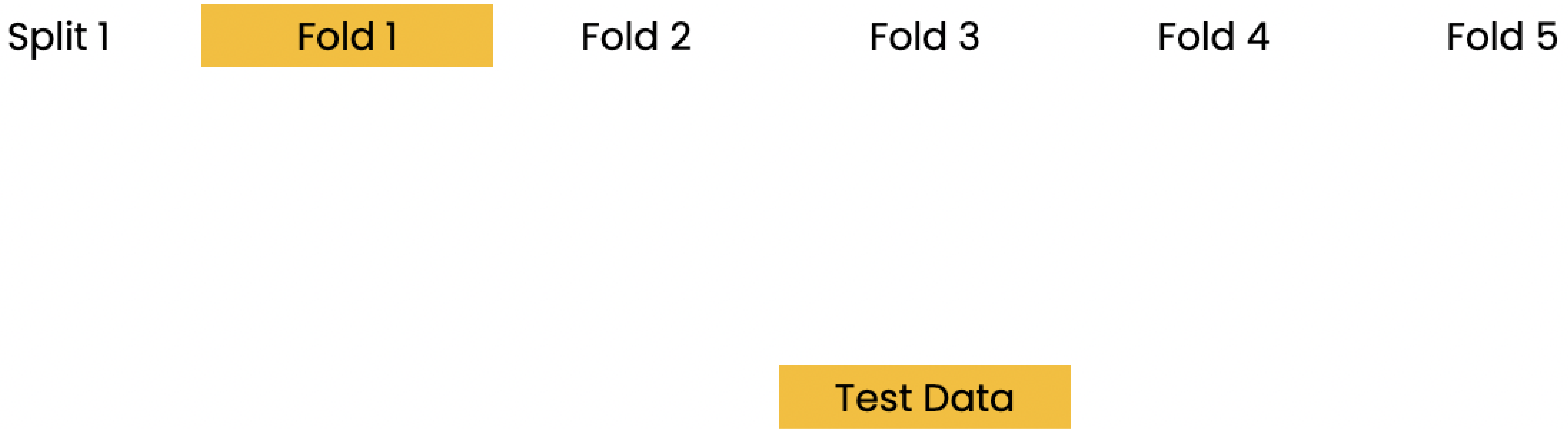
Cross-validation motivation

- Model performance is dependent on the way we split up the data
- Not representative of the model's ability to generalize to unseen data
- Solution: Cross-validation!

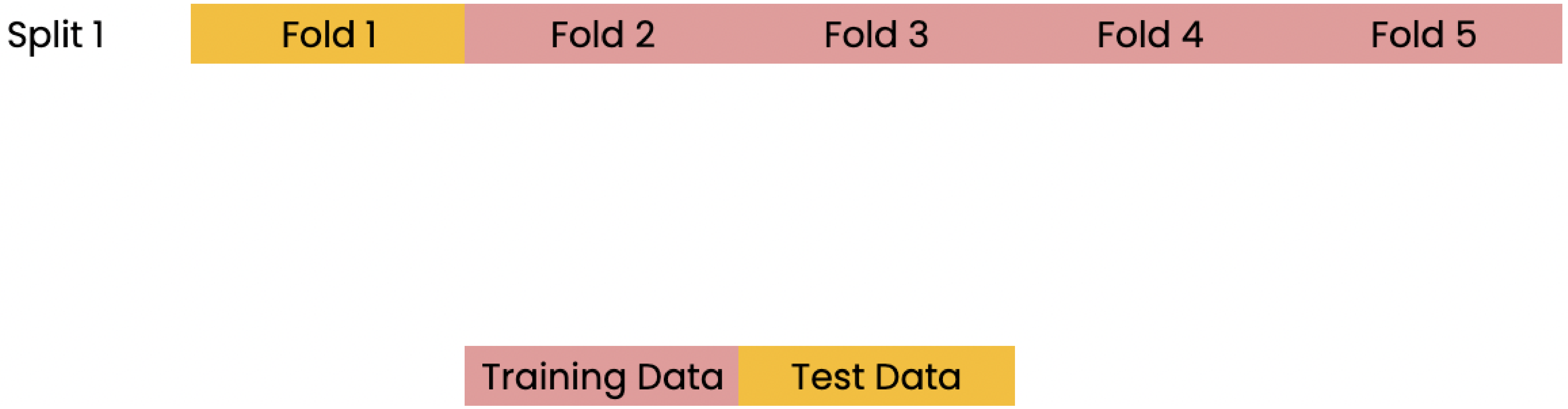
Cross-validation basics

Split 1 Fold 1 Fold 2 Fold 3 Fold 4 Fold 5

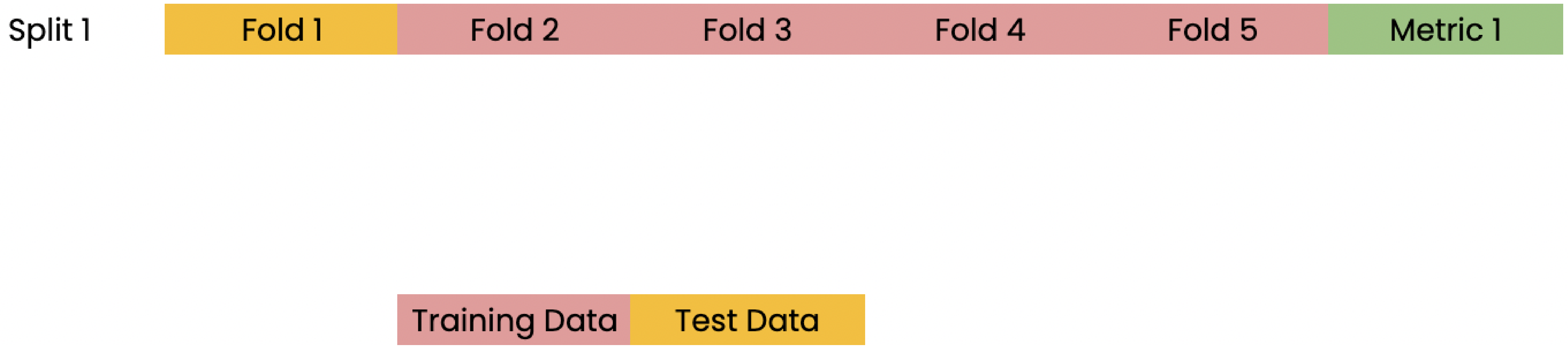
Cross-validation basics



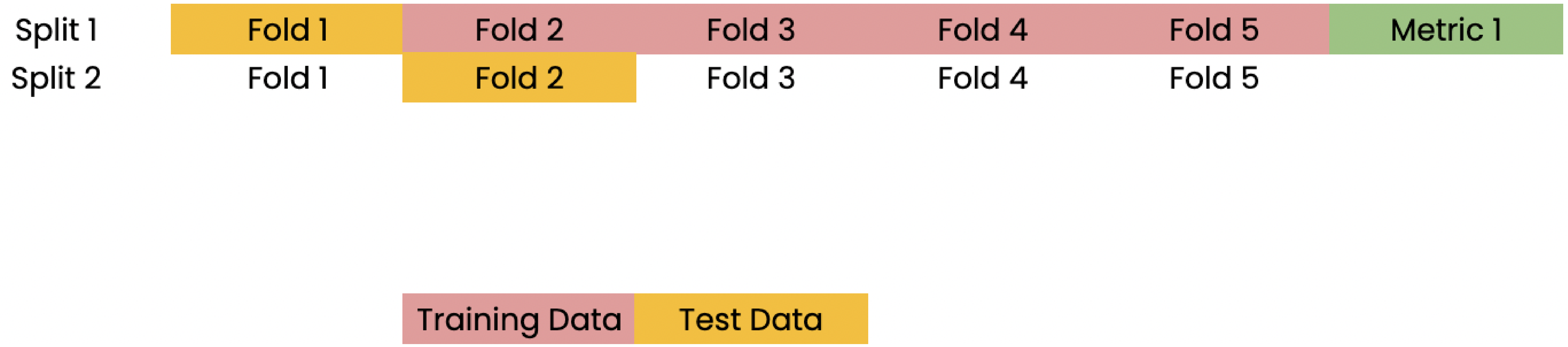
Cross-validation basics



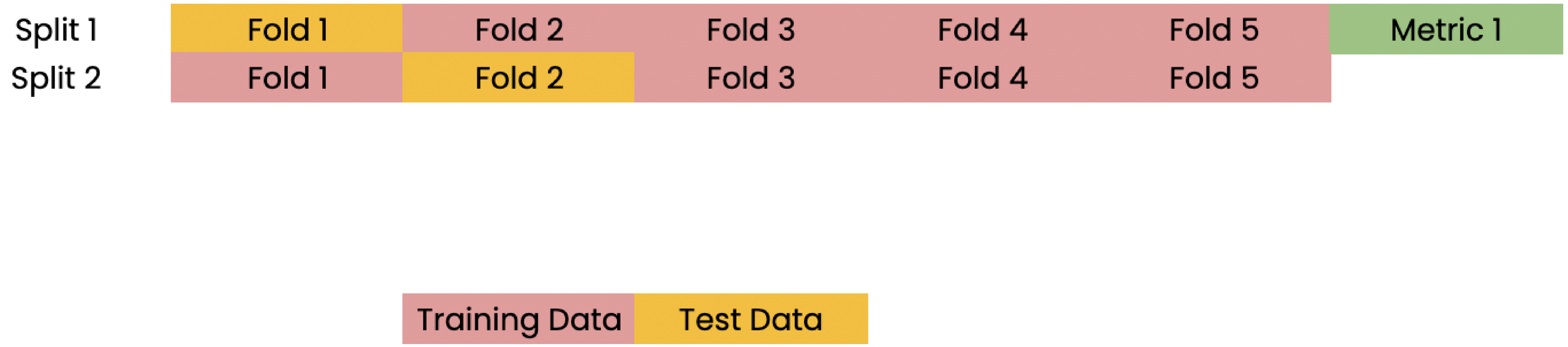
Cross-validation basics



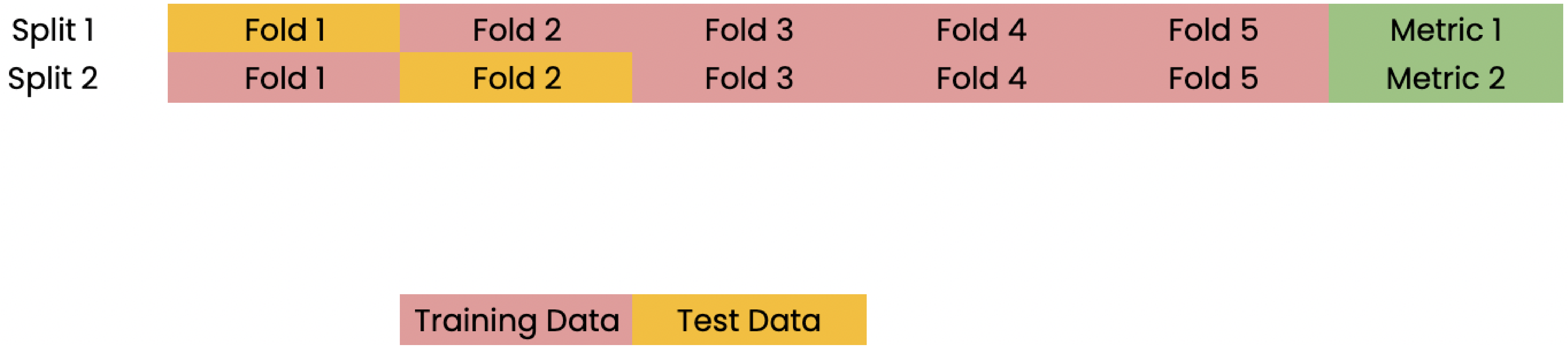
Cross-validation basics



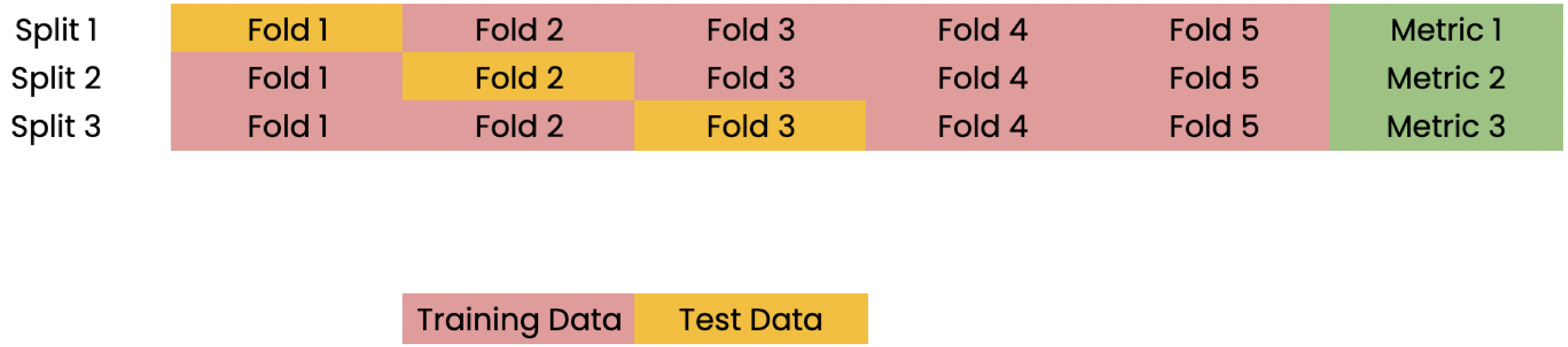
Cross-validation basics



Cross-validation basics



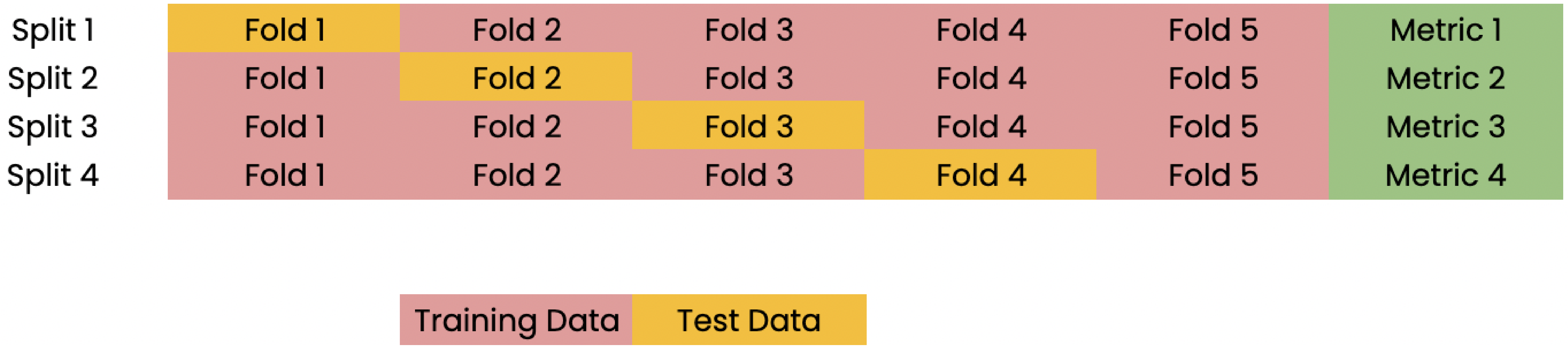
Cross-validation basics



Split 1	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Metric 1
Split 2	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Metric 2
Split 3	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Metric 3

Training Data	Test Data

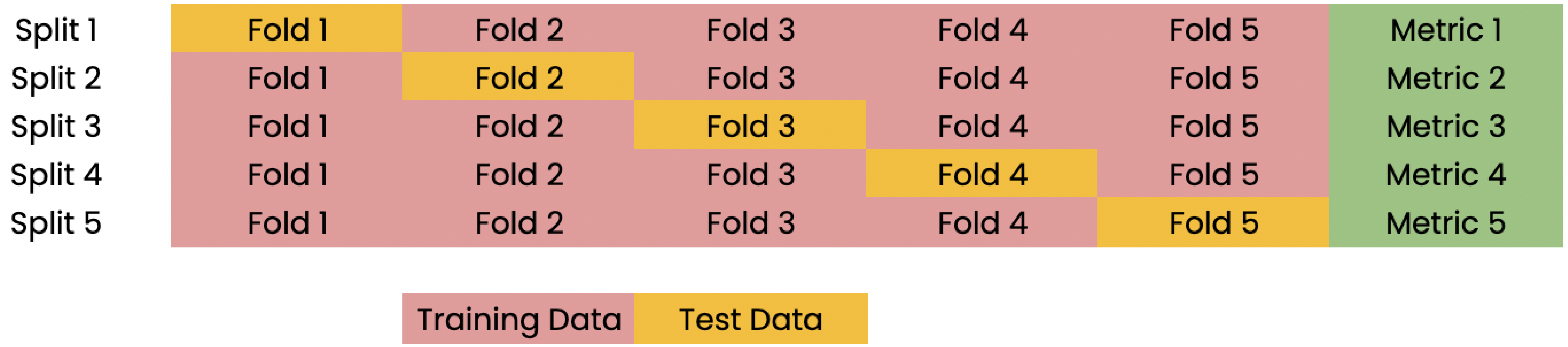
Cross-validation basics



Split 1	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Metric 1
Split 2	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Metric 2
Split 3	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Metric 3
Split 4	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Metric 4

🟥 Training Data	🟨 Test Data

Cross-validation basics



Split 1	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Metric 1
Split 2	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Metric 2
Split 3	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Metric 3
Split 4	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Metric 4
Split 5	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Metric 5

🟥 Training Data 🟨 Test Data

Cross-validation and model performance

- 5 folds = 5-fold CV
- 10 folds = 10-fold CV
- k folds = k-fold CV
- More folds = More computationally expensive

This is a trade-off. This is because we are fitting ans predicting more times.

Cross-validation in scikit-learn

```
from sklearn.model_selection import cross_val_score, KFold
kf = KFold(n_splits=6, shuffle=True, random_state=42)
reg = LinearRegression()
cv_results = cross_val_score(reg, X, y, cv=kf)
```

This allows to set a seed and shuffle our data making our results repeatable downstream.

n_splits argument is in default 5. shuffle shuffles our dataset before splitting into folds. We also assign a to the random_state keyword argument ensuring our data will split in the same way if we repeat the process.

This will make results repeatable downstream.
cv_results returns an array of cross validation scores which we assign to cv_results. The length of the array is the number of folds utilized. Note that the score reported is r-squared as this is the default score of linear regression.

Evaluating cross-validation peformance

```
print(cv_results)
```

```
[0.70262578, 0.7659624, 0.75188205, 0.76914482, 0.72551151, 0.73608277]
```

```
print(np.mean(cv_results), np.std(cv_results))
```

```
0.7418682216666667 0.023330243960652888
```

```
print(np.quantile(cv_results, [0.025, 0.975]))
```

Here we find the 95% confidence interval by passing np.quantile passing our results as well as the upper and lower limits of our intervals as decimals.

```
array([0.7054865, 0.76874702])
```

Let's practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

Regularized regression

SUPERVISED LEARNING WITH SCIKIT-LEARN

George Boorman
Core Curriculum Manager, DataCamp

Why regularize?

Regularization is a technique that avoids overfitting.

- Recall: Linear regression minimizes a loss function
- It chooses a coefficient, a, for each feature variable, plus b
- Large coefficients can lead to overfitting
- Regularization: Penalize large coefficients

so it is a common practise to alter the loss function so it penailzes large coeffitients.

Ridge regression

Ordinary Leased Squared loss function

- Loss function = OLS loss function +

$$\alpha * \sum_{i=1}^{n} {a_i}^2$$

This is the squared value of each coefficient multiplied by a coefficient alpha.

When minimizing the loss function,

- Ridge penalizes large positive or negative coefficients
- α: parameter we need to choose

When using ridge we need to chose the alpha value in order to fit and predict. We can select an alpha in which our model performs best.

- Picking α is similar to picking `k` in KNN

Alpha in ridge is known as a hyperparameter.

- Hyperparameter: variable used to optimize model parameters
- α controls model complexity
  - α = 0 = OLS (Can lead to overfitting)
  - Very high α: Can lead to underfitting

When alpha is zero we are performing OLS where large coefficients are not penalized and overfitting might occur.
A high alpha means large coefficients are significantly penalized which can lead to underfitting.

Ridge regression in scikit-learn

```python
from sklearn.linear_model import Ridge
scores = []
for alpha in [0.1, 1.0, 10.0, 100.0, 1000.0]:
    ridge = Ridge(alpha=alpha)
    ridge.fit(X_train, y_train)
    y_pred = ridge.predict(X_test)
    scores.append(ridge.score(X_test, y_test))
print(scores)
```

To highlight the impact of different alpha values we create an empty list for our scores, then look through a list of different alpha values.

Inside the for loop we instantiate ridge setting the alpha keyword argument equals to the iterator also called alpha.

After that we fit on the training data and predict on the test data.

We save the model's R squared value to the scores list.

Finally outside the loop we print scores for the models with 5 different alpha values.

```
[0.2828466623222221, 0.28320633574804777, 0.2853000732200006,
 0.2642398481266813, 0.19292424694100963]
```

We see that the performance gets worst as alpha increases.

Lasso regression

This is another type of regularized regression.

Here the loss function is the OLS loss function plus the absolute value of each coefficient multiplied by some constant alpha.

- Loss function = OLS loss function +

$$\alpha * \sum_{i=1}^{n} |a_i|$$

Lasso regression in scikit-learn

```
from sklearn.linear_model import Lasso
scores = []
for alpha in [0.01, 1.0, 10.0, 20.0, 50.0]:
  lasso = Lasso(alpha=alpha)
  lasso.fit(X_train, y_train)
  lasso_pred = lasso.predict(X_test)
  scores.append(lasso.score(X_test, y_test))
print(scores)
```

We import this from sklearn.linear_model

The method of performing lasso regression is similar to the way that we performed ridge regression.
The performance drops substantially as the alpha drops below 20.

```
[0.99991649071123, 0.99961700284223, 0.93882227671069, 0.74855318676232, -0.05741034640016]
```

Lasso regression for feature selection

- Lasso can select important features of a dataset
- Shrinks the coefficients of less important features to zero
- Features not shrunk to zero are selected by lasso

Lasso regression can actually be used to assess feature importance.

Because it tends to shrink the coefficients of less important features to zero.

Lets check this out in practise in the next example.

Lasso for feature selection in scikit-learn

```
from sklearn.linear_model import Lasso
X = diabetes_df.drop("glucose", axis=1).values
y = diabetes_df["glucose"].values
names = diabetes_df.drop("glucose", axis=1).columns
lasso = Lasso(alpha=0.1)
lasso_coef = lasso.fit(X, y).coef_
plt.bar(names, lasso_coef)
plt.xticks(rotation=45)
plt.show()
```

Here we access the names of the features using the dataset's .columns attribute and store as the variable "names".

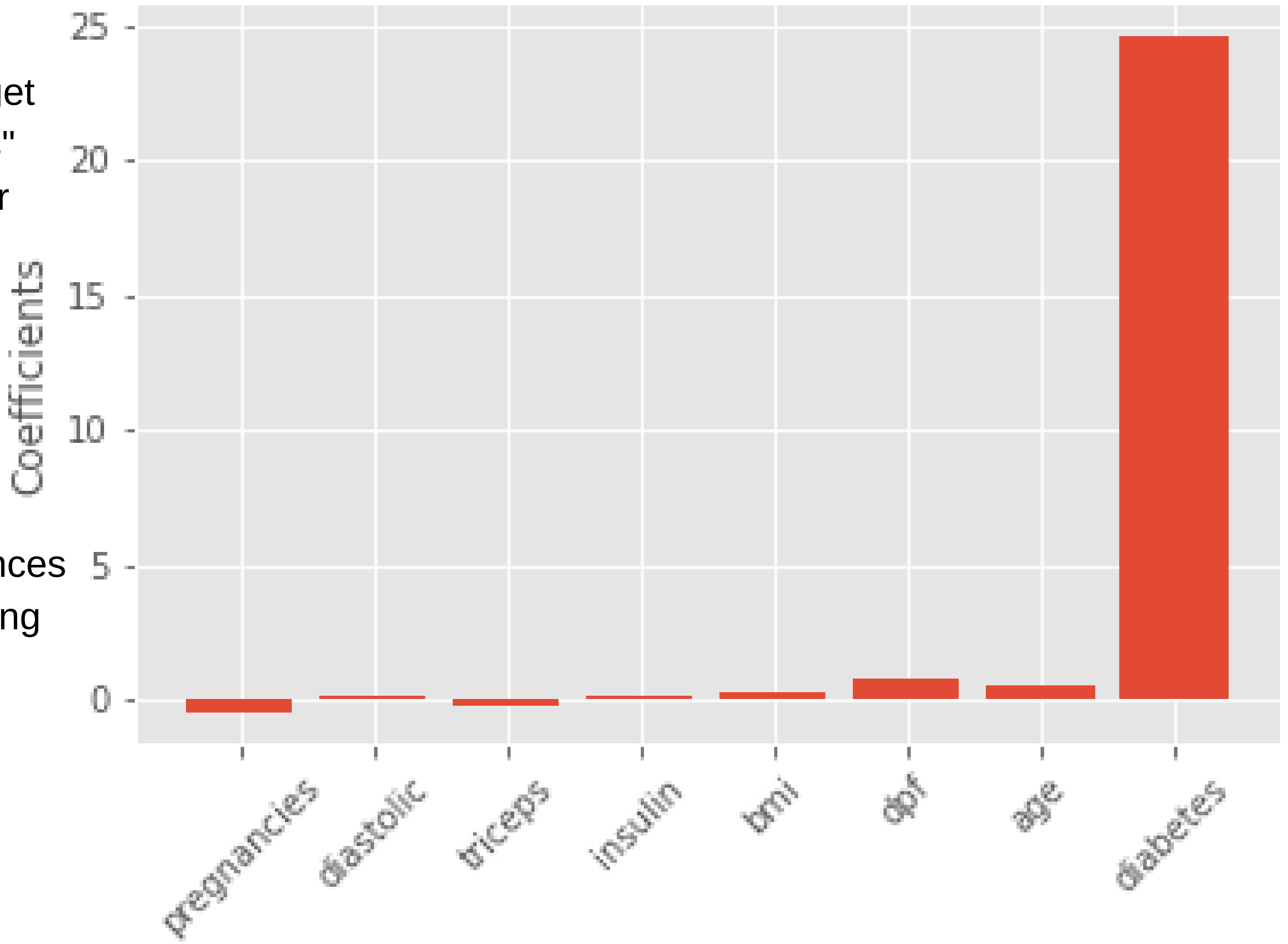
As we are calculating feature importance we use the entire dataset rather than splitting it. We then instantiate lasso setting alpha to 0.1. We fit the model to the data and extract the coefficients using the .coef_ attribute, storing as the lasso_coef.

We then plot the coefficients for each feature.

Lasso for feature selection in scikit-learn

We can see that the most important predictor for our taget variable."blood glucose levels" is the binary value for whether an individual has diabetes or not.

This type of feature selection is very important because it allows us to communicate results to non technical audiences and it is also useful to identifying which factors are important predictors for various physical phenomena.



Let's practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN