Introduction to regression

SUPERVISED LEARNING WITH SCIKIT-LEARN

Predicting blood glucose levels

```
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

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

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

Making predictions from a single feature

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

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

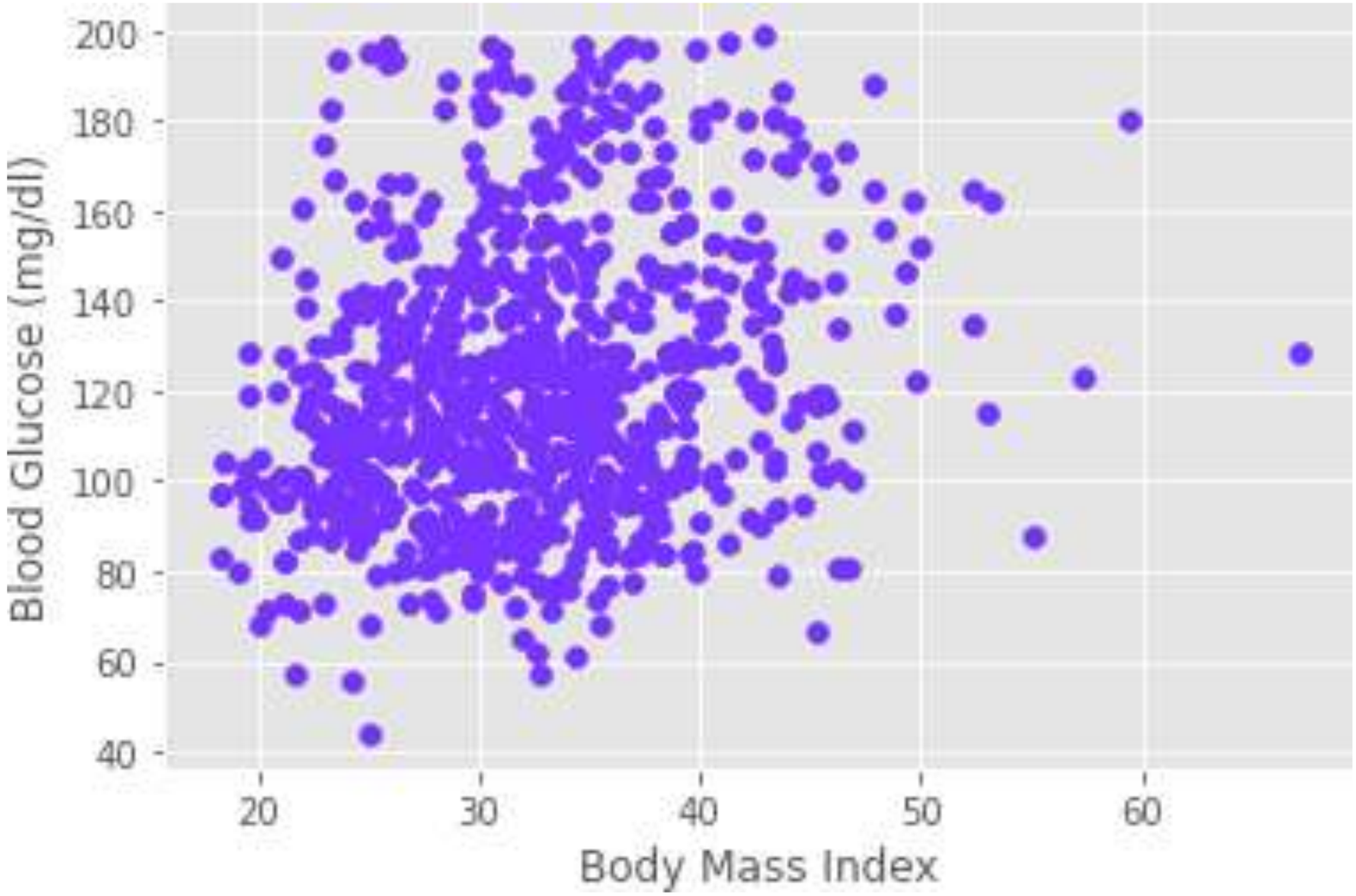
```
X_bmi = X_bmi.reshape(-1, 1)
print(X_bmi.shape)
```

```
(752, 1)
```

Plotting glucose vs. body mass index

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

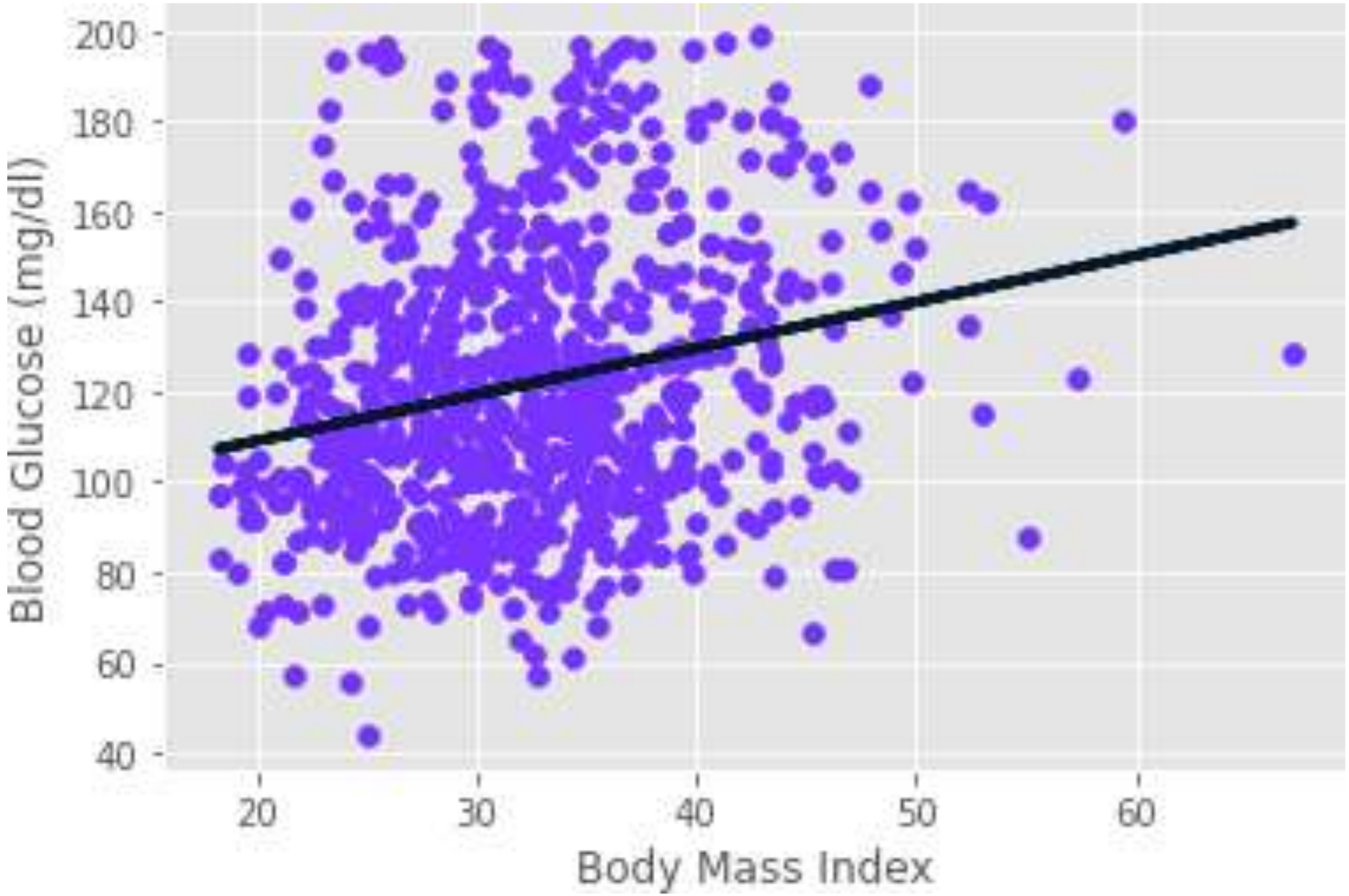
Plotting glucose vs. body mass index



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



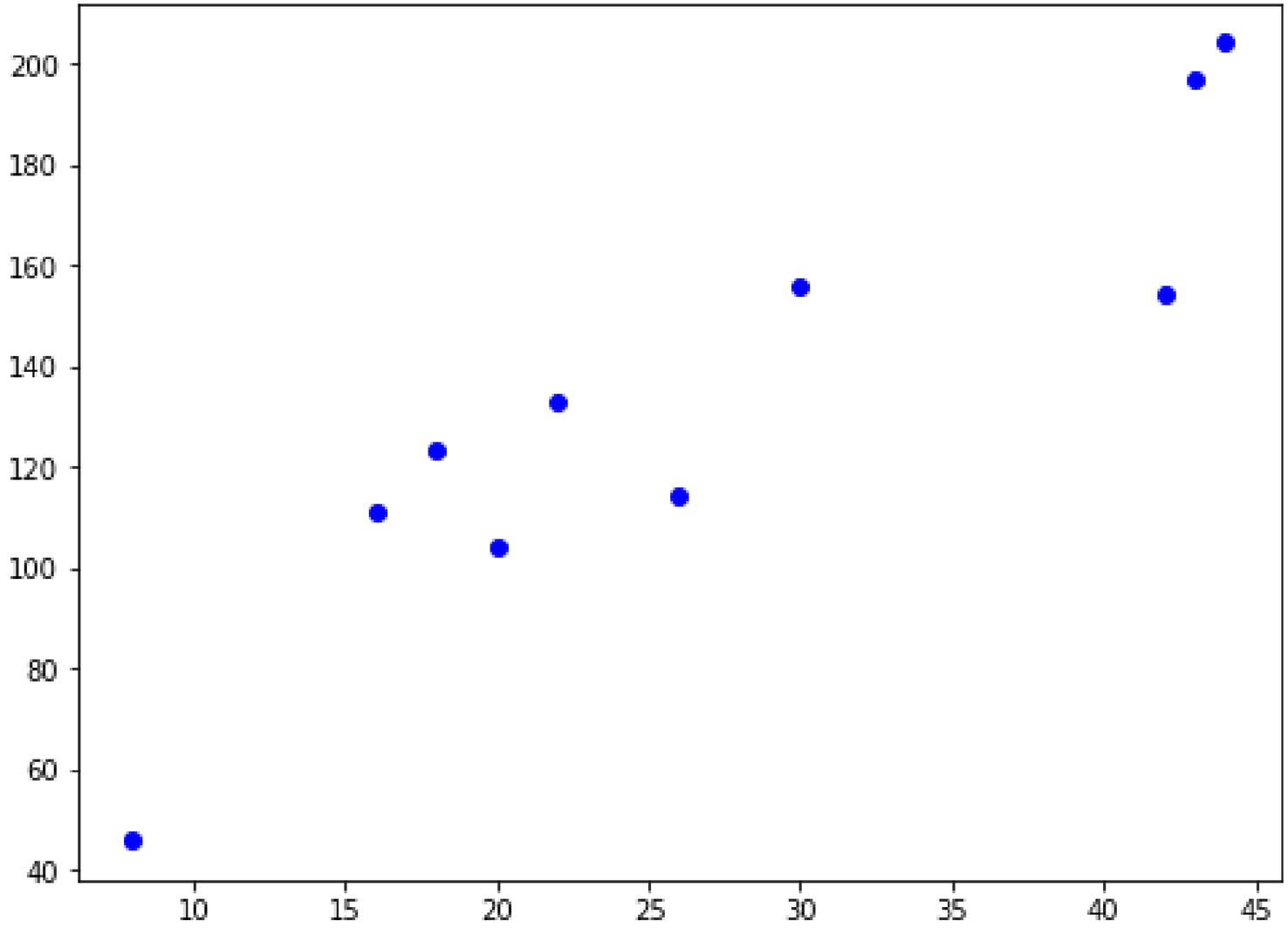
The basics of linear regression

SUPERVISED LEARNING WITH SCIKIT-LEARN

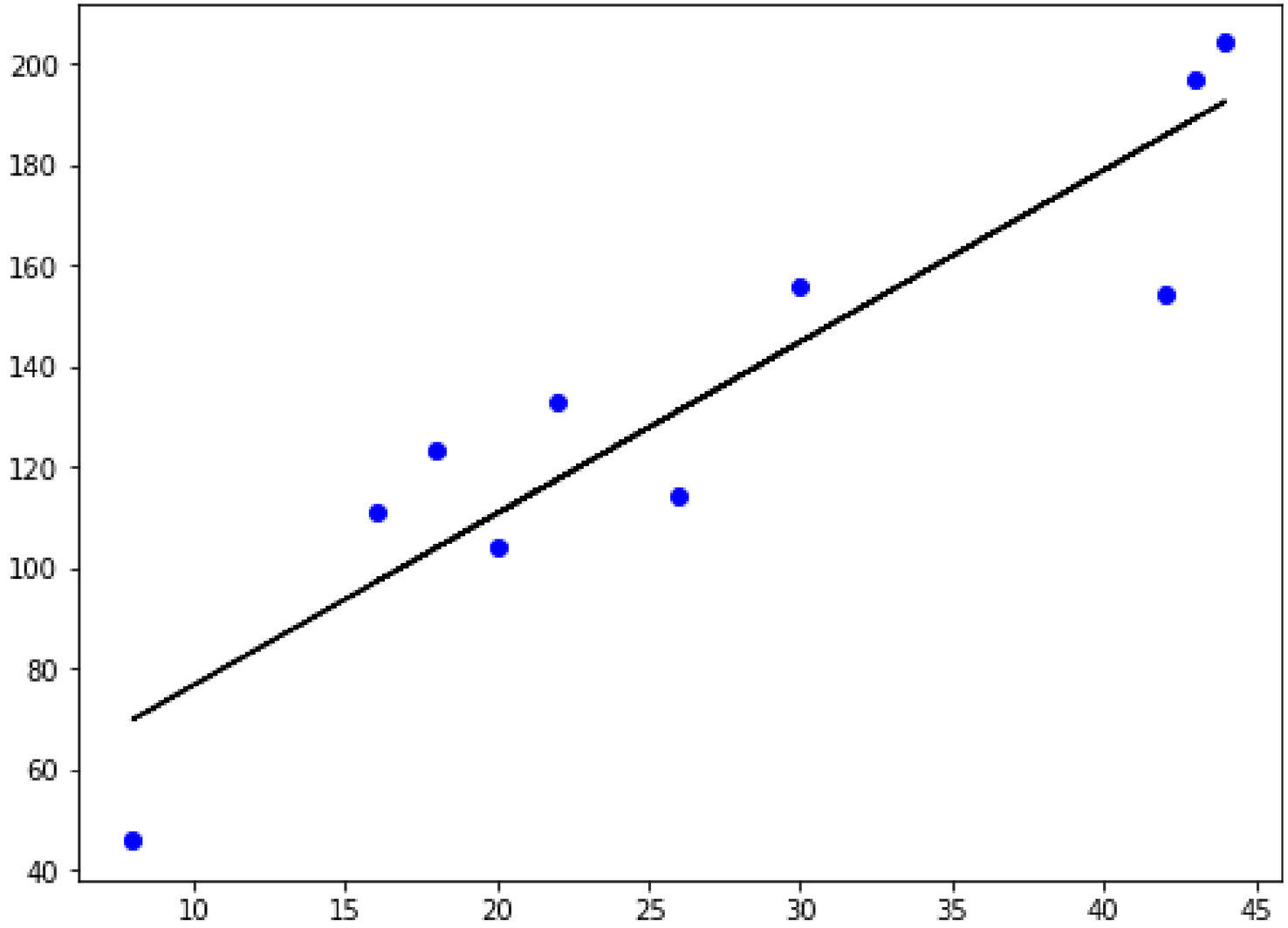
Regression mechanics

- $y = ax + b$
  - 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

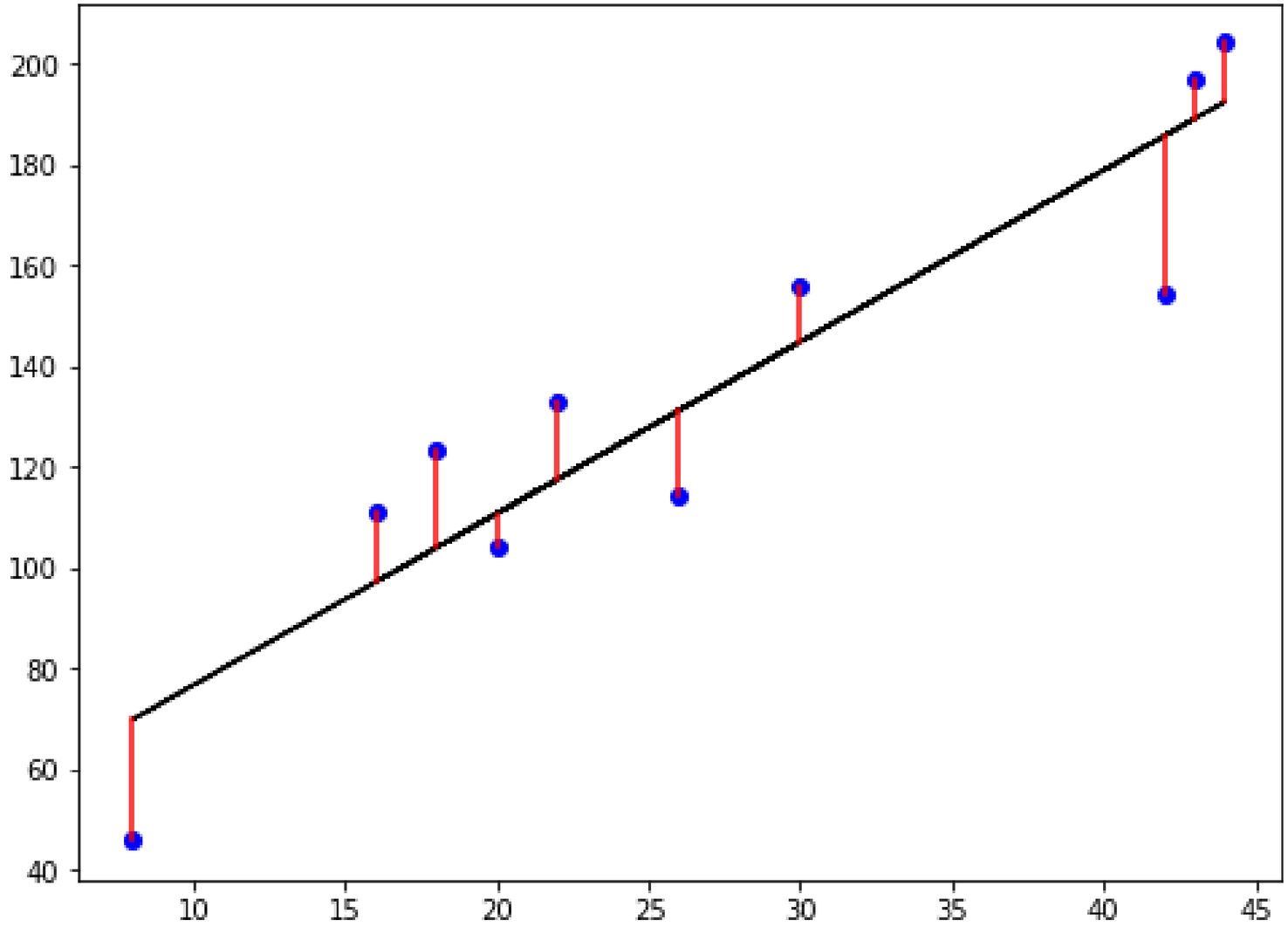
The loss function



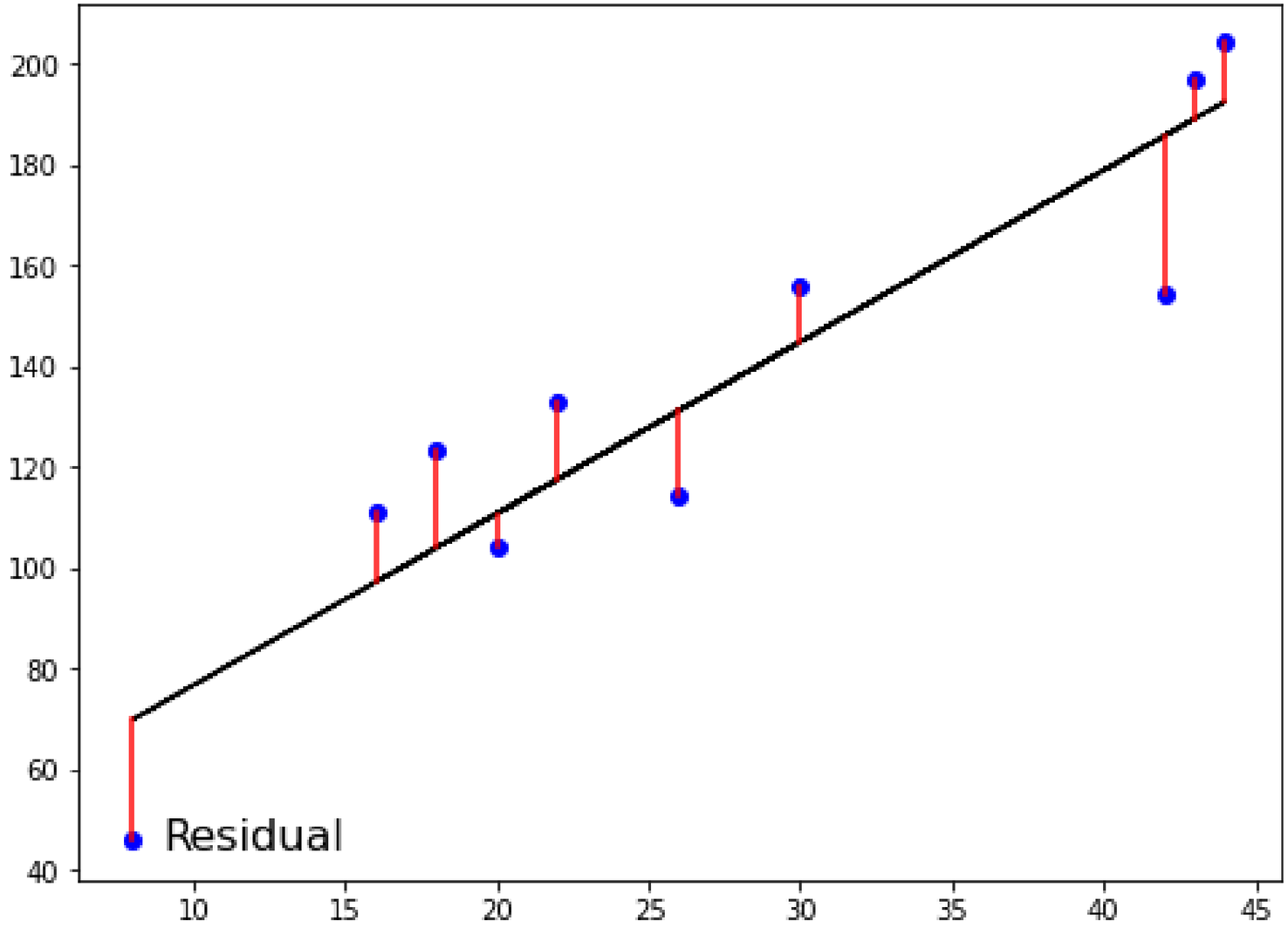
The loss function



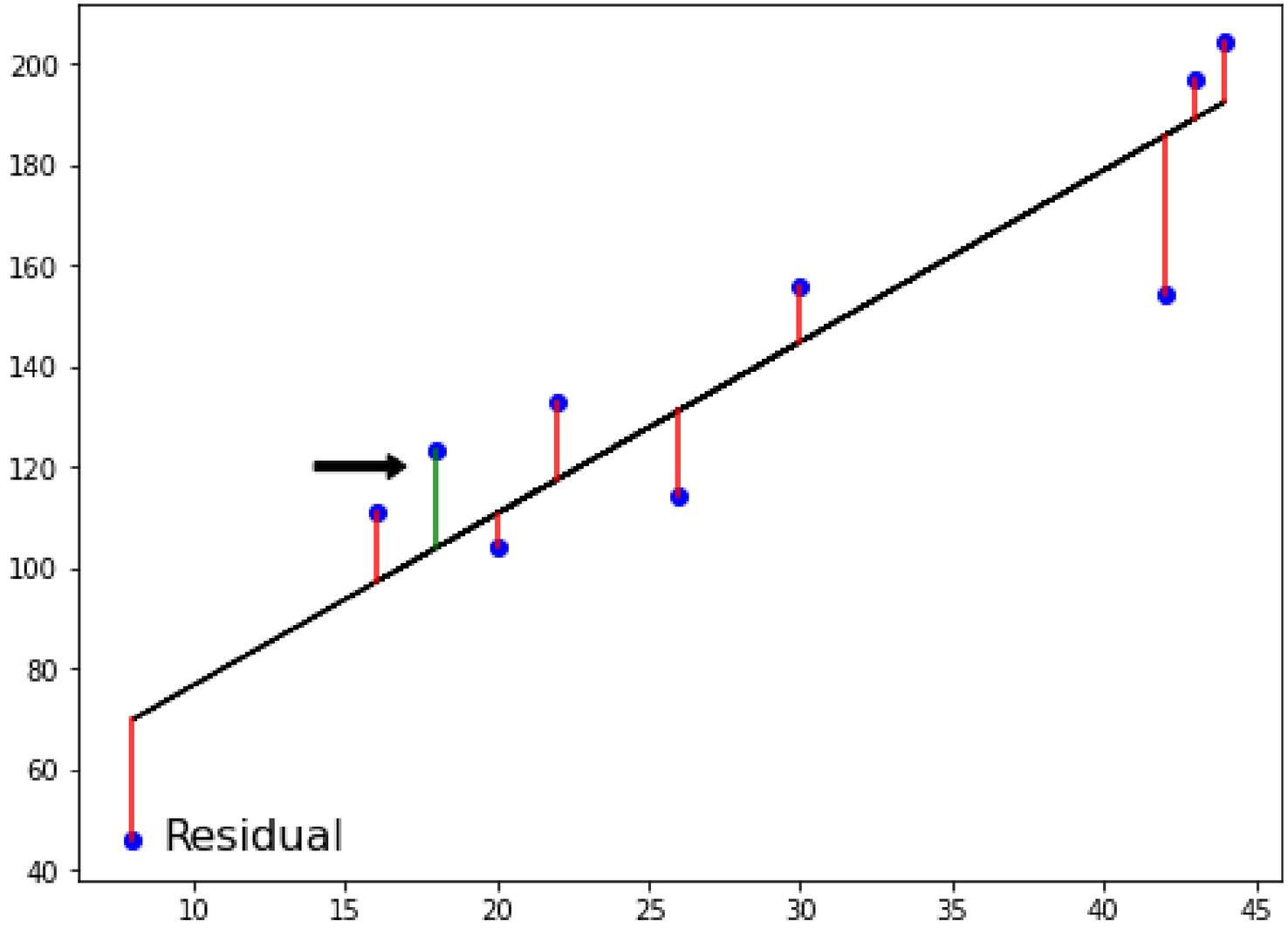
The loss function



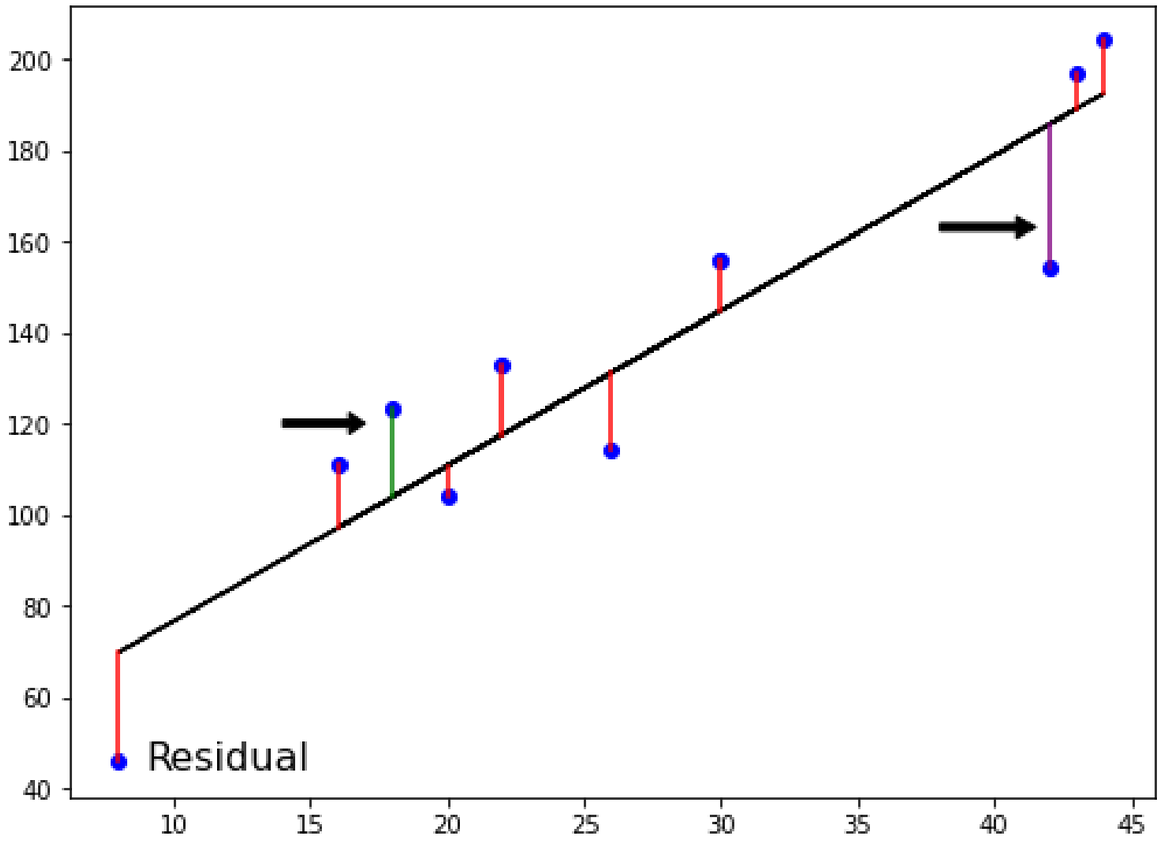
The loss function



The loss function



Ordinary Least Squares



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

Ordinary Least Squares (OLS): minimize RSS

RSS: Residual Sum of Squares

Linear regression in higher dimensions

$$y = a_1 x_1 + a_2 x_2 + b$$

- 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

$$y = a_1 x_1 + a_2 x_2 + a_3 x_3 + \ldots + a_n x_n + b$$

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

Linear regression using all features

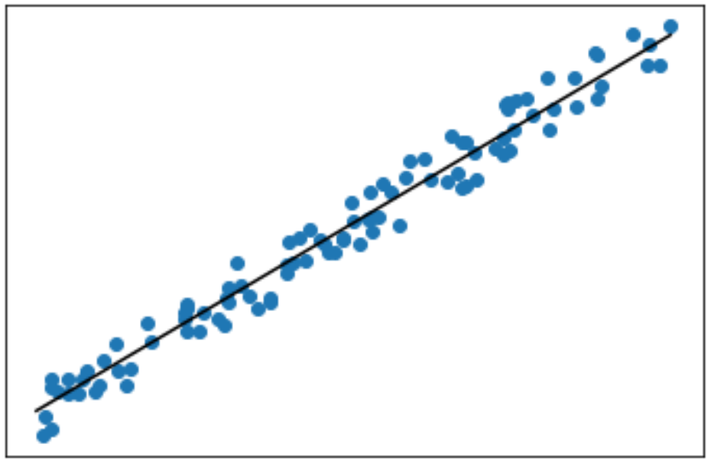
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
                                                    random_state=42)
reg_all = LinearRegression()
reg_all.fit(X_train, y_train)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

R-squared

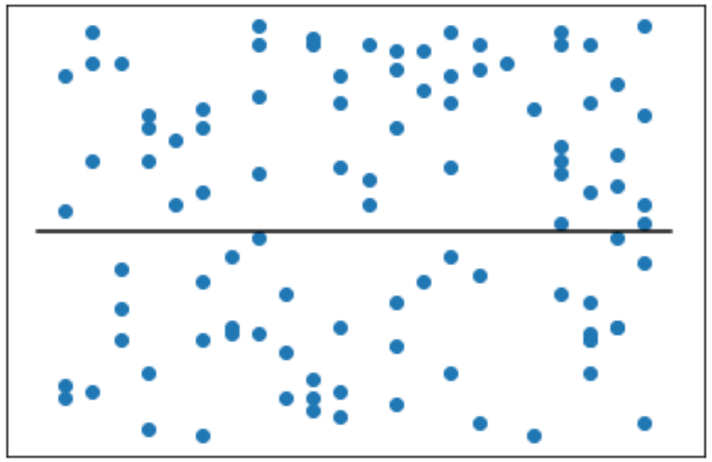
- R^2: quantifies the variance in target values explained by the features

  Values range from 0 to 1

- 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

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

- MSE is measured in target units, squared

$$RMSE = \sqrt{MSE}$$

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

RMSE in scikit-learn

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

```
24.028109426907236
```

squared=False: This option specifies that the output should be the **Root Mean Squared Error (RMSE)** instead of the MSE.
By default, squared=True, which returns the MSE

MSE is often used during optimization since it penalizes larger errors more heavily
RMSE is used for interpretation because it is in the same scale as the target variable,
making it easier to understand how far predictions deviate from actual values.

Cross-validation

SUPERVISED LEARNING WITH SCIKIT-LEARN

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	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

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)
```

shuffle=True:
Randomly shuffles the data before splitting it into folds.
This ensures that the data is not divided in the same order as it appears in the dataset,
which can improve the robustness of the cross-validation process.

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]))
```

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

Regularized regression

SUPERVISED LEARNING WITH SCIKIT-LEARN

Why regularize?

- 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

Ridge regression

Ridge regression is a type of linear regression that includes **L2 regularization**. It is used to prevent overfitting and handle multicollinearity in the data by introducing a penalty term to the loss function that shrinks the regression coefficients towards zero.

- Loss function = OLS loss function +

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

- Ridge penalizes large positive or negative coefficients
- α: parameter we need to choose
- Picking α is similar to picking `k` in KNN
- Hyperparameter: variable used to optimize model parameters
- α controls model complexity
  - $\alpha = 0$ = OLS (Can lead to overfitting)
  - Very high α: Can lead to underfitting

Multicollinearity refers to a situation in regression analysis where two or more independent variables (predictors) are highly correlated with each other. This means they provide redundant information about the dependent variable, making it difficult to isolate the individual effect of each predictor.

a_i Coefficients of the regression model
n Number of predictors

Ridge regression in scikit-learn

```
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)
```

```
[0.2828466623222221, 0.28320633574804777, 0.2853000732200006,
 0.26423984812668133, 0.19292424694100963]
```

Interpreting the Ridge Score:
High R2R^2R2 (close to 1): Indicates the model explains most of the variability in the data.
Low R2R^2R2 (close to 0): Indicates the model explains very little of the variability.
A better R2R^2R2 on test data compared to training data suggests that Ridge regression is effectively reducing overfitting.

Lasso regression

- Loss function = OLS loss function +

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

Lasso regression (short for **Least Absolute Shrinkage and Selection Operator**) is a linear regression method that includes **L1 regularization**. It is used for both preventing overfitting and performing feature selection.

L1 Regularization: The loss function in Lasso regression adds a penalty proportional to the absolute values of the regression coefficients

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)
```

```
[0.999916490711 23, 0.999617002842 23, 0.938822276710 69, 0.748553186762 32, -0.057410346400 16]
```

Interpreting Lasso Score:

1.High:

1. Indicates the model explains a significant portion of the variance in the target variable.
2. For Lasso, this means that the selected features (non-zero coefficients) are strong predictors of the target.

2.Low:

1. Suggests that the model does not explain much of the variance in the target variable.
2. For Lasso, a low R2R^2R2 may occur if the regularization parameter (α\alphaα) is too high, leading to excessive shrinkage of coefficients (or all coefficients being reduced to zero).

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 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()
```

Lasso for feature selection in scikit-learn