Scikit-Learn: Machine Learning Library

educative

Quickstart

Overview of Scikit-Learn (sklearn)

Scikit-learn is a powerful Python library for machine learning, providing simple and efficient data analysis and modeling tools.

Importance in Data Science and Machine Learning

• Widely used for its simplicity and consistency.

• A comprehensive range of algorithms and utilities for both supervised and unsupervised learning.

• Integrates well with other Python libraries such as NumPy, pandas, and Matplotlib.

Installation and the First Example

For installation of scikit-learn, use the following command:

```
pip install scikit-learn
```

The following example loads the Iris dataset, splits it into features (X) and target labels (y), and then trains a logistic regression model to predict the set variables based on the features.

```
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression

iris = load_iris()
X = iris.data   # Features
y = iris.target  # Target labels

model = LogisticRegression()
model.fit(X, y)
```

Data Loading

Loading Datasets into Scikit-Learn

• Scikit-learn provides built-in datasets like Iris, numbers, etc.

• Custom data can also be loaded from CSV, NumPy array, etc.

```
Import seaborn as sns
Iris = sns.load_dataset('iris')
```

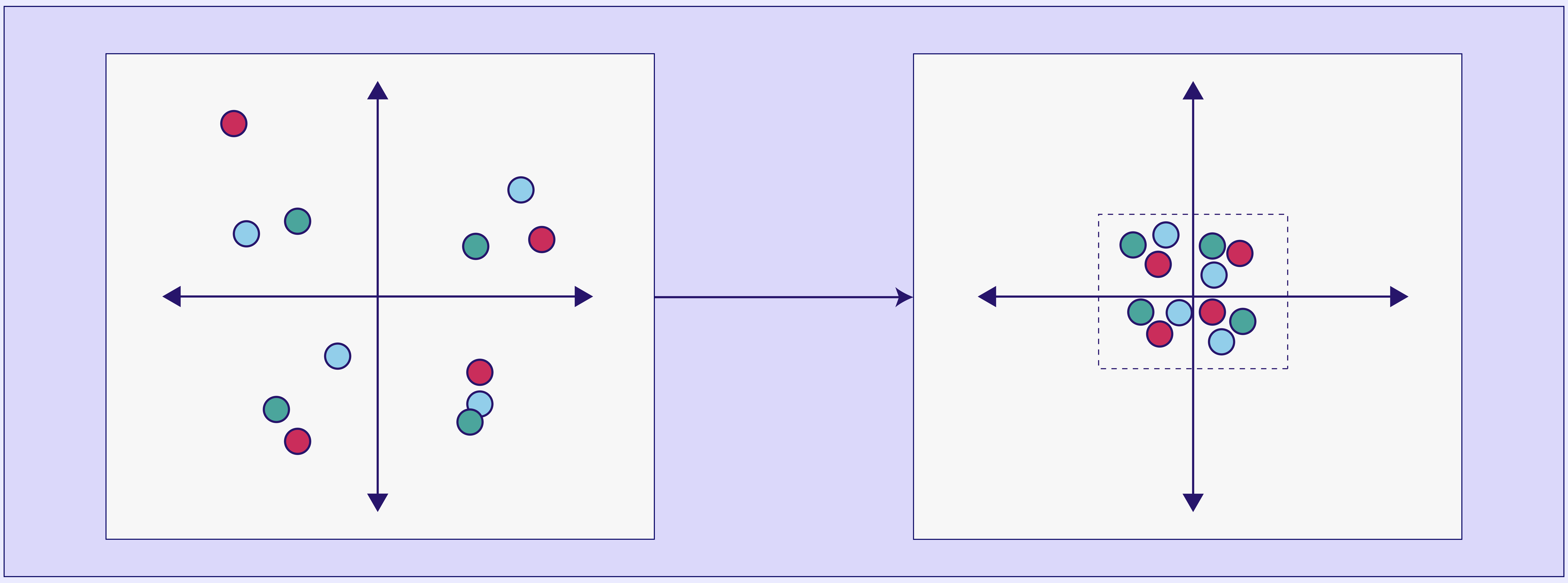
Preprocessing

Data Preprocessing Steps

• Handling missing values: Replace (fill) or remove missing values.

```
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy='mean')
X = imputer.fit_transform(X)
```

• Feature scaling (standardization, normalization): Normalize (mean=0, standard=1) or compare (0-1) the features.



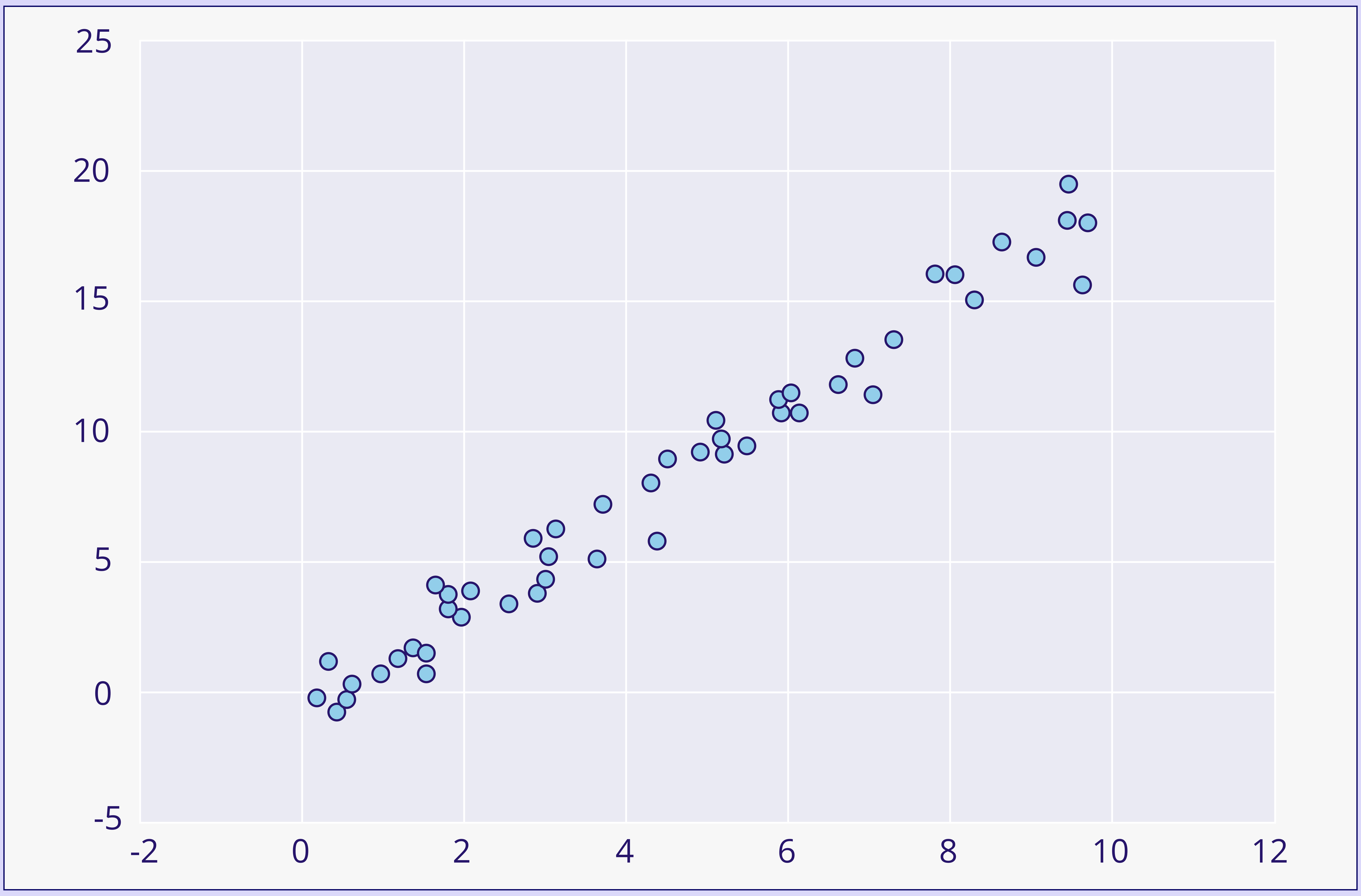
```
from sklearn.preprocessing import StandardScaler, MinMaxScaler

# Standardization
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Normalization
normalizer = MinMaxScaler()
X_normalized = normalizer.fit_transform(X)
```

• **Encoding categorical variables:** Converts a string variable to a numeric value.

```
from sklearn.preprocessing import OneHotEncoder

encoder = OneHotEncoder()
X_encoded = encoder.fit_transform(X_categorical).toarray()
```

• **Visualizations:** Use libraries like Matplotlib or Seaborn for data exploration and visualization.

```
Import matplotlib.pyplot as plt
Import numpy as np

Rng = np.random.RandomState(42)
X = 10 * rng.rand(50)
plt.scatter(x, y);
```

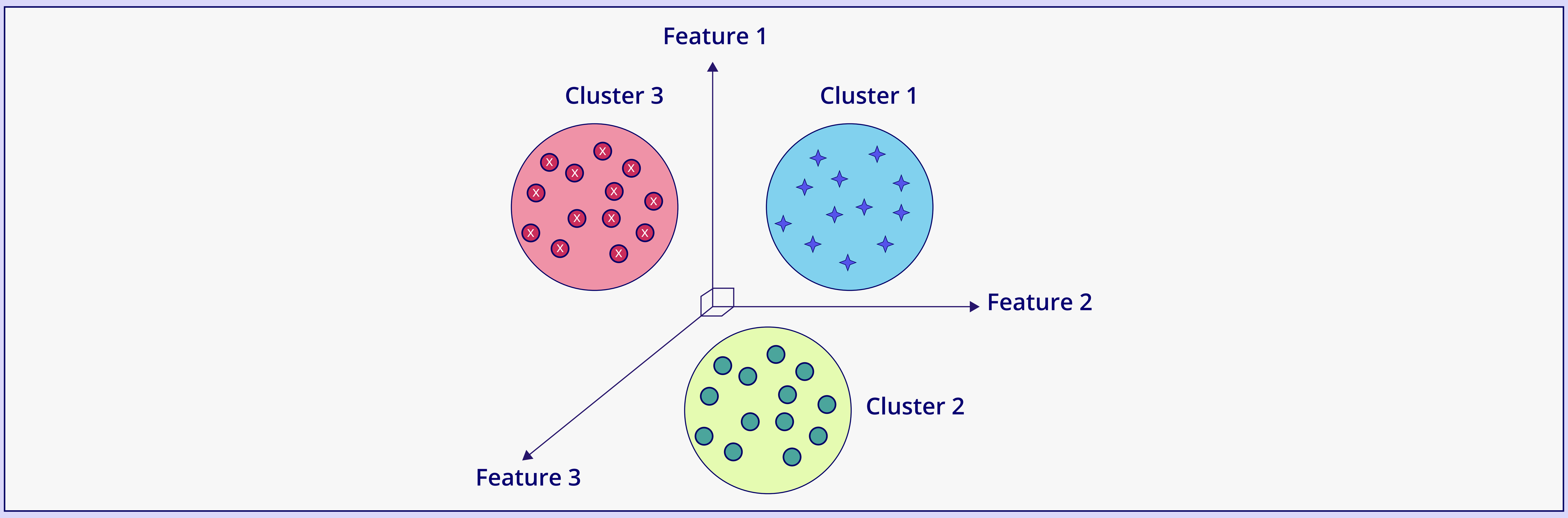


Machine Learning

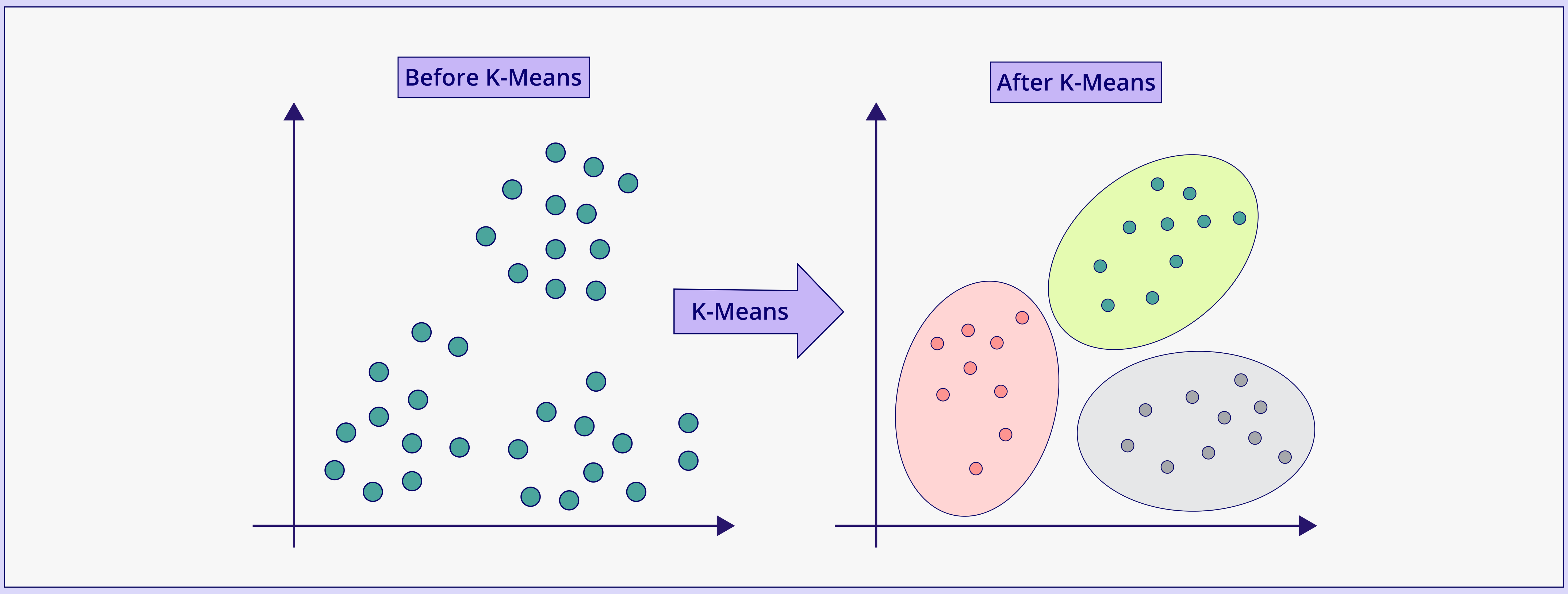
Unsupervised Learning: If the data is unlabeled and we use machines to label the data and discover unknown patterns, it's considered unsupervised learning.

Scikit-Learn: Machine Learning Library

- **Clustering:** Group similar data points together.

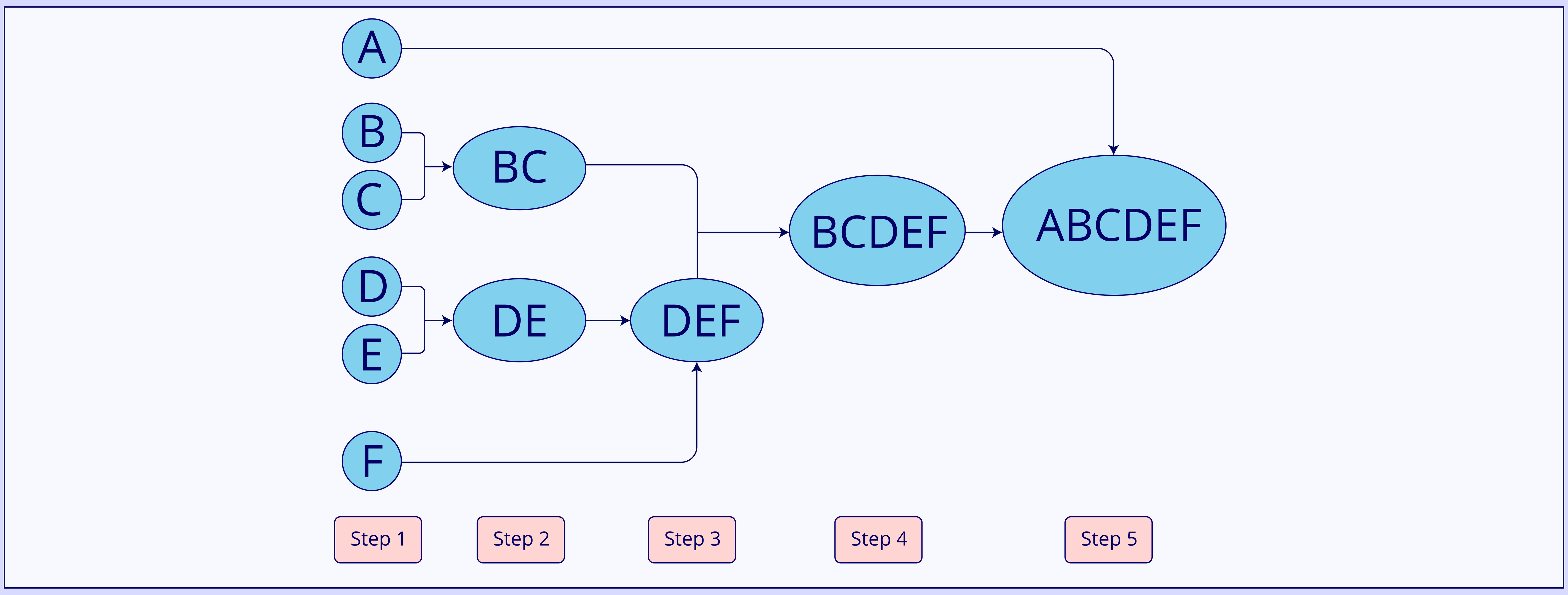


- **K-Means clustering:** This type of clustering is simple and efficient for spherical clusters.




```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
labels = kmeans.labels_
```

- **Hierarchical clustering:** This type of clustering builds a hierarchy of clusters for exploration

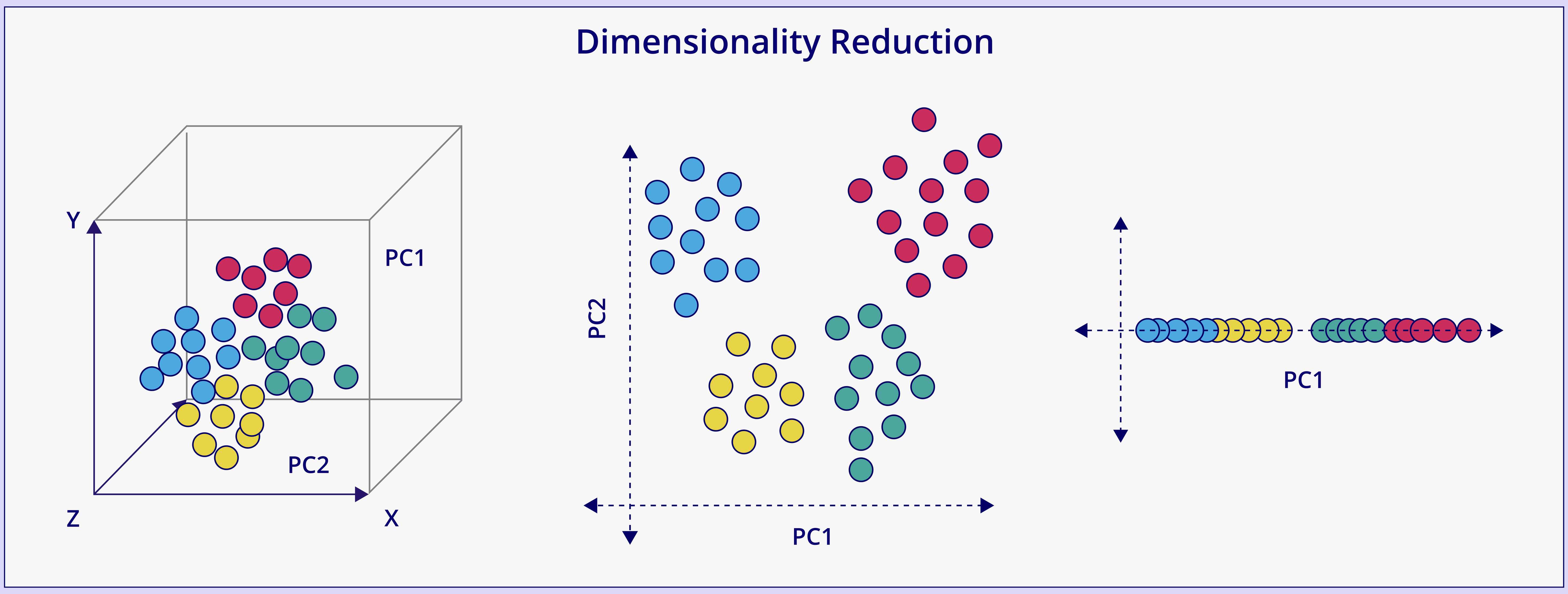



```
from sklearn.cluster import AgglomerativeClustering

agglo = AgglomerativeClustering(n_clusters=3)
labels = agglo.fit_predict(X)
```

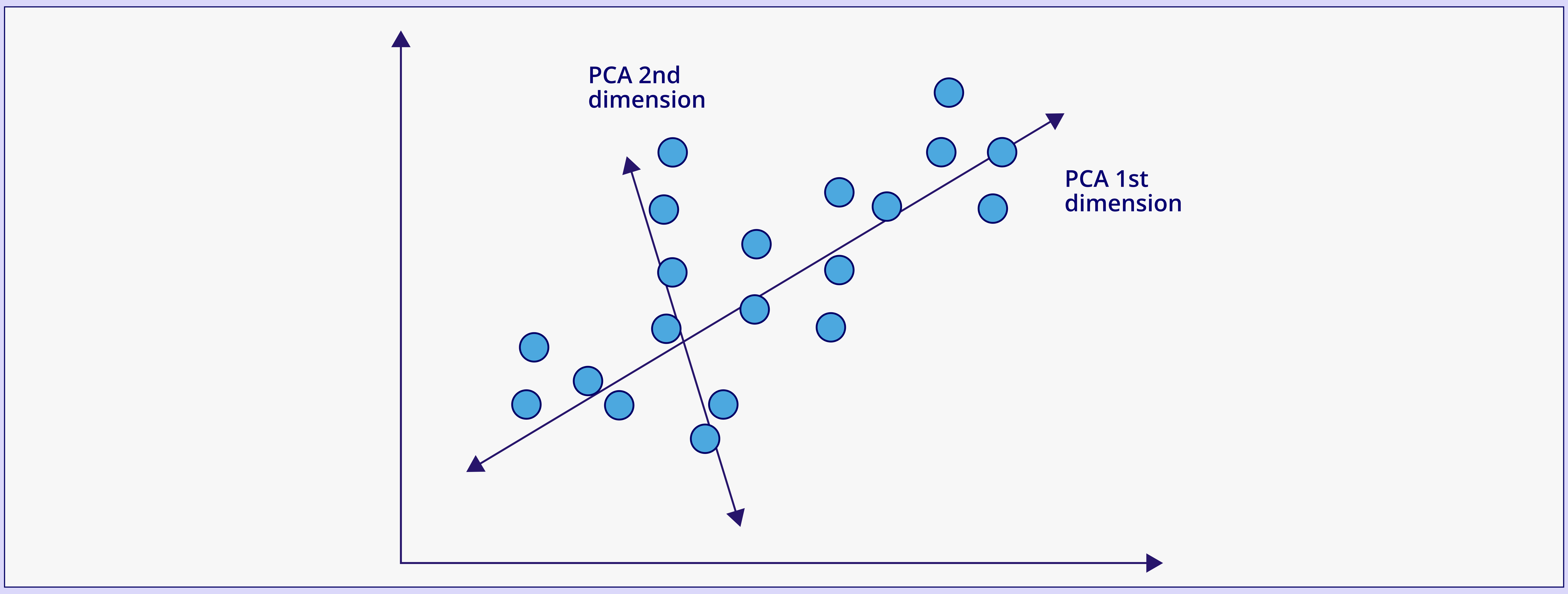
Scikit-Learn: Machine Learning Library

educative

Dimensionality reduction: Reduce data dimensionality while maintaining its information.

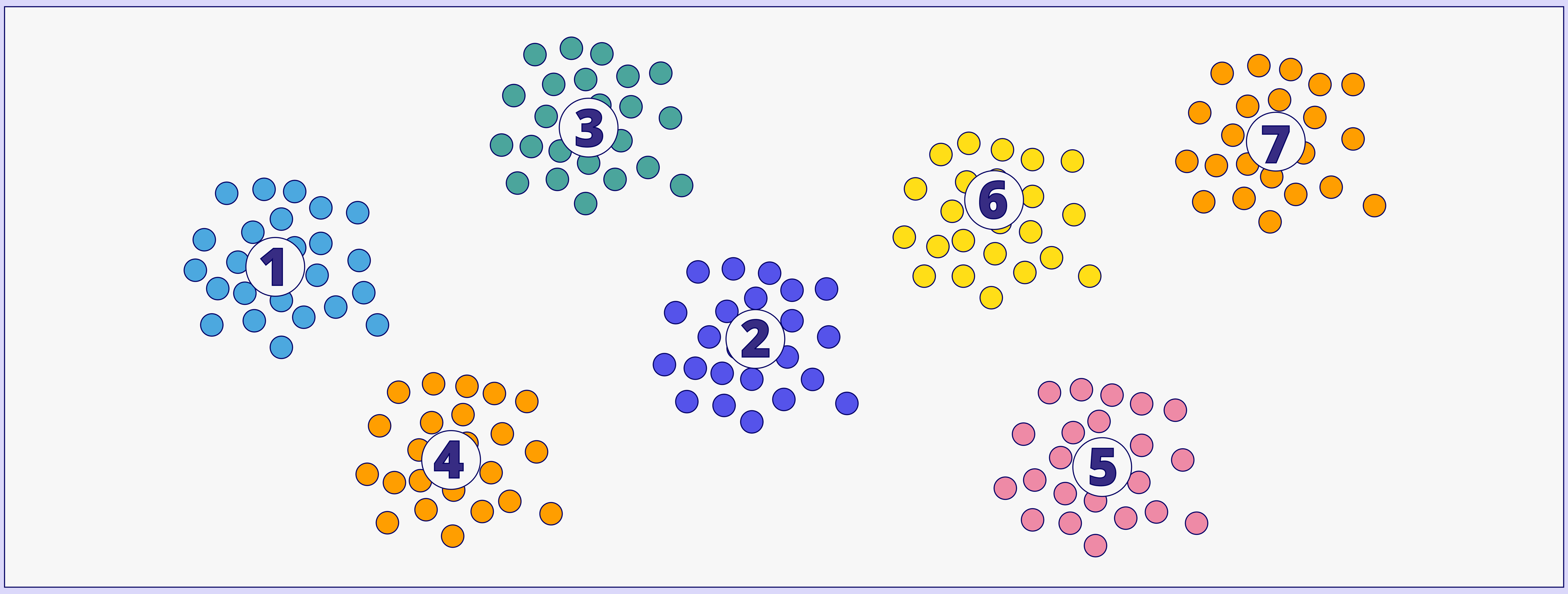


• **Principal component analysis (PCA):** Captures the most variance in reduced dimensions.




```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
```

• **TSNE (T-distributed Stochastic Neighbor Embedding):** Useful for visualizing high-dimensional data in lower dimensions.

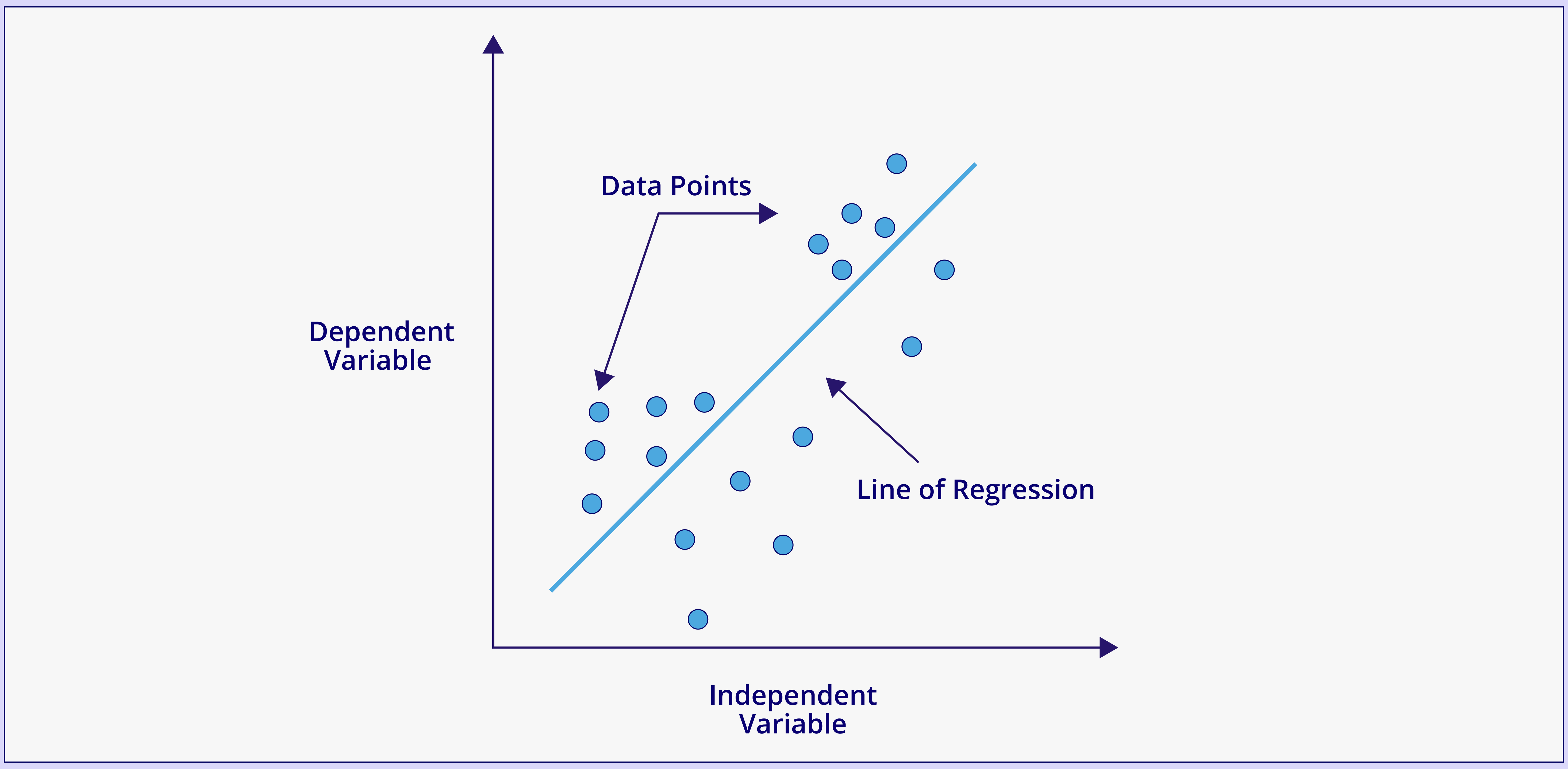


```python
from sklearn.manifold import TSNE

tsne = TSNE(n_components=2)
X_tsne = tsne.fit_transform(X)
```

Supervised Learning: If humans are labeling the data and the machine is then tasked with accurately labeling current or future data points, this is known as supervised learning.

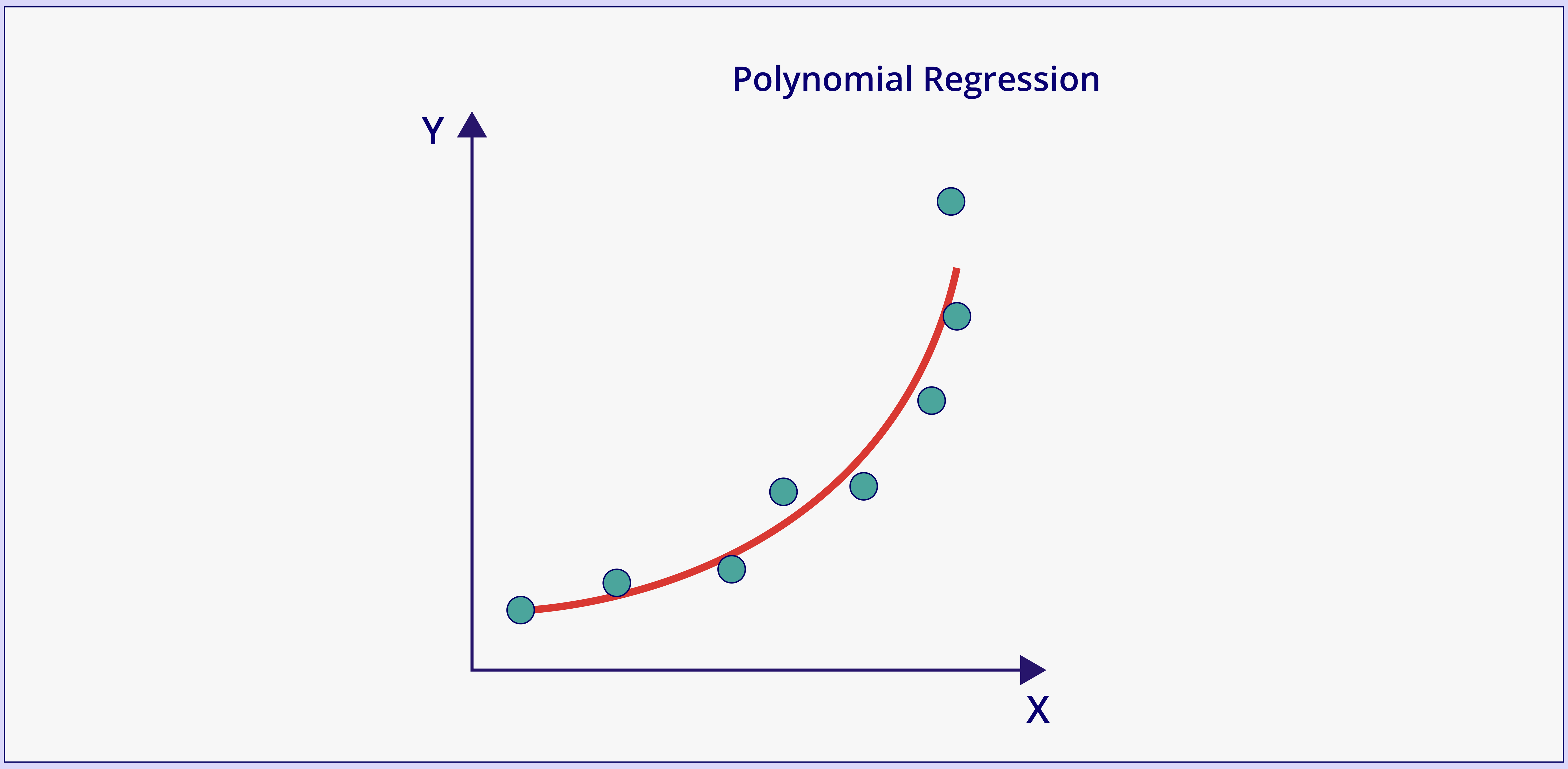
Regression: Involves predicting continuous target variables.

- **Linear regression:** Models linear relationships between features and target.



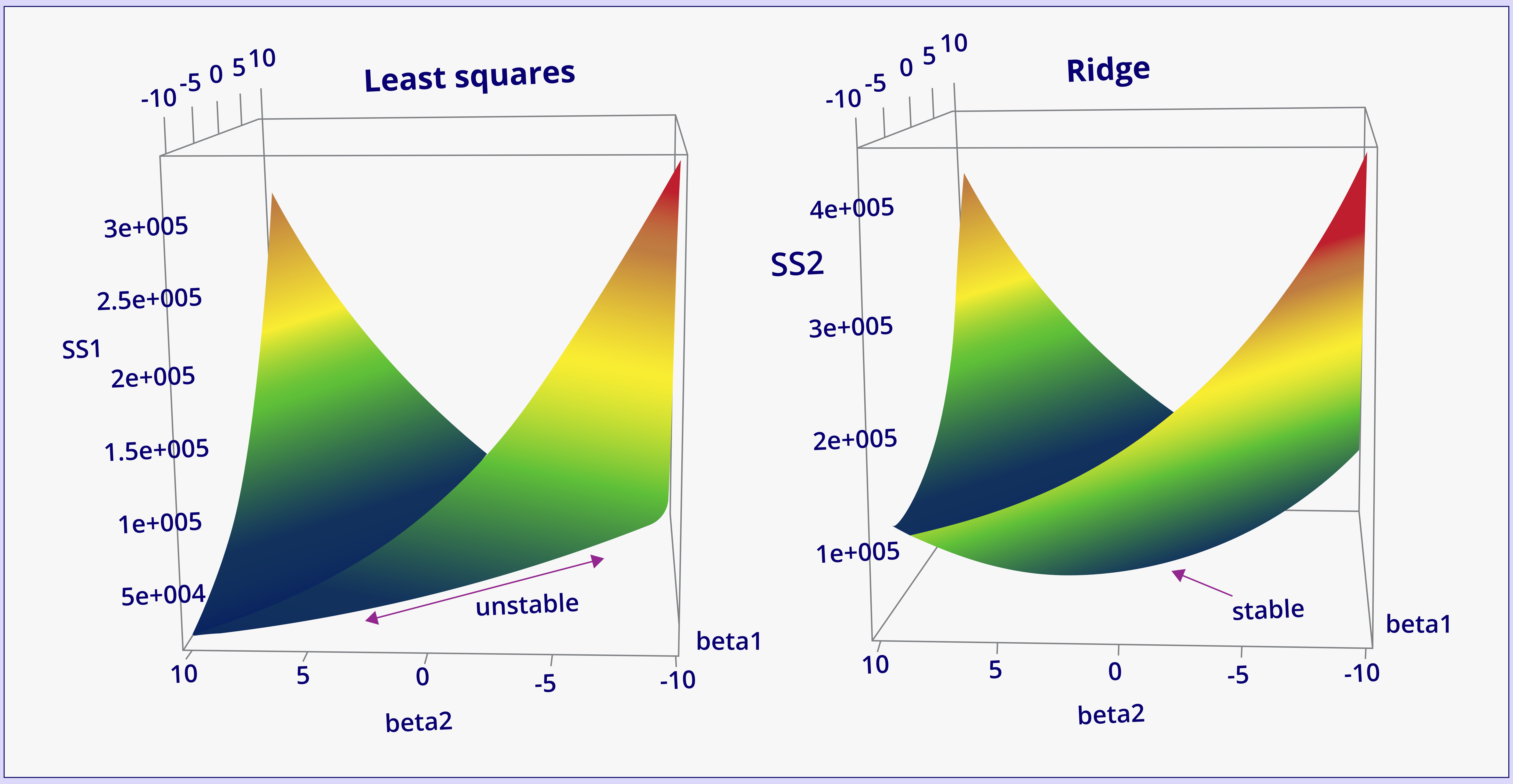

```python
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

- **Polynomial Regression:** Models nonlinear relationships.



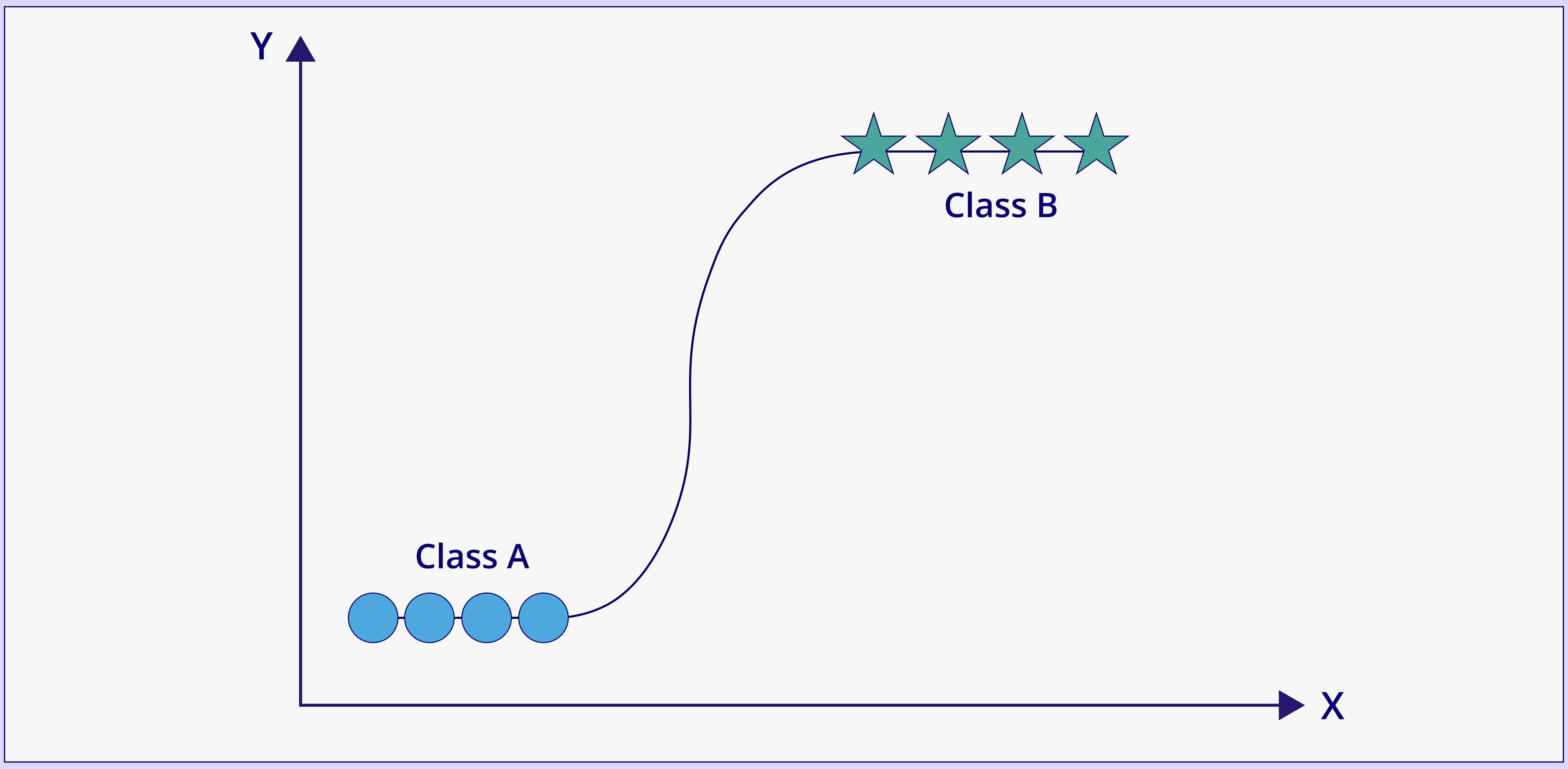
```
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X)
model = LinearRegression()
model.fit(X_poly, y)
```

- **Ridge and lasso regression:** Regularized regression techniques to prevent overfitting.




```
from sklearn.linear_model import Ridge, Lasso

ridge = Ridge(alpha=1.0)
lasso = Lasso(alpha=0.1)

ridge.fit(X_train, y_train)
lasso.fit(X_train, y_train)
```

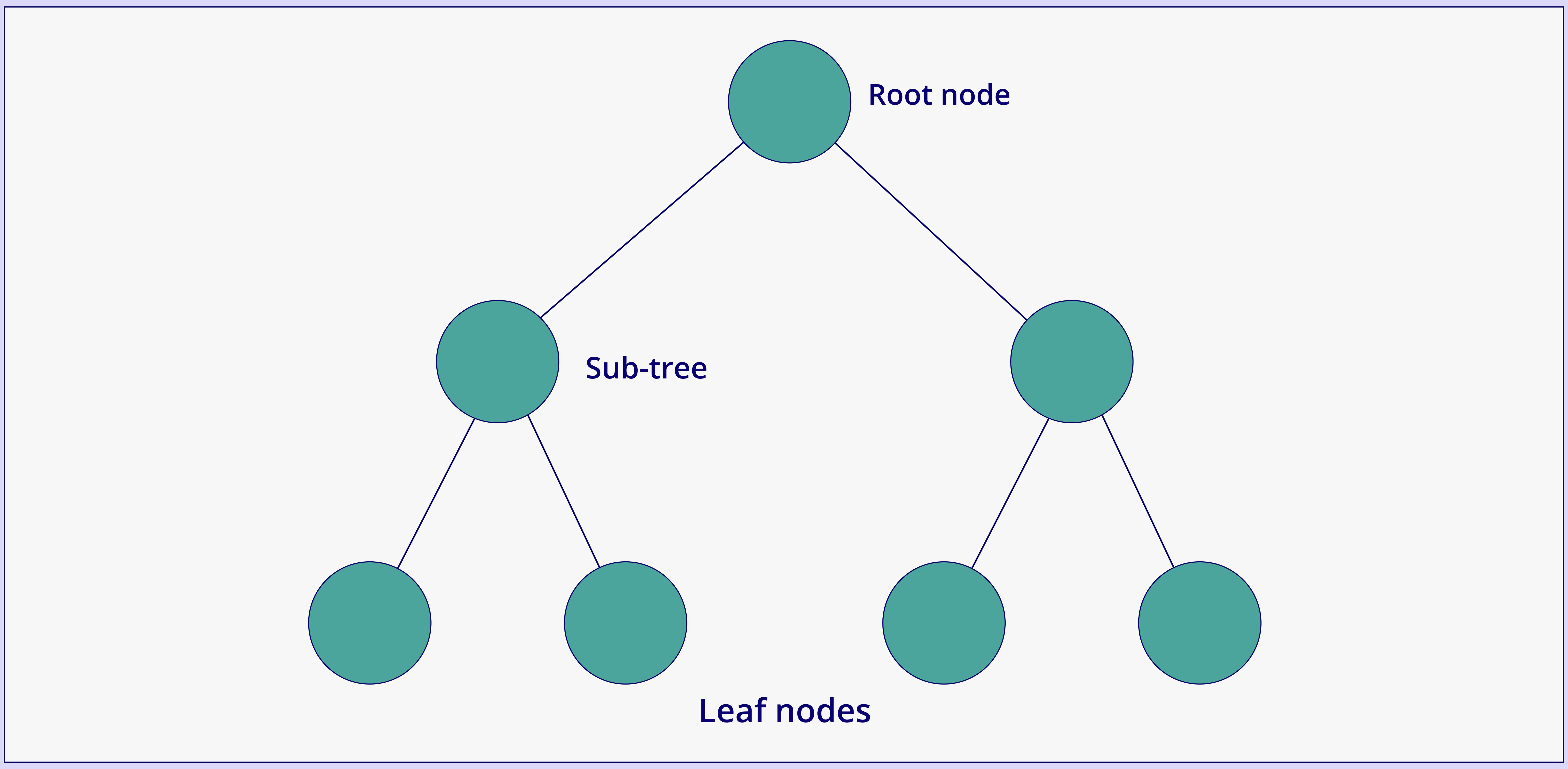
- **Classification:** Classifies data points into predefined categories.
  - **Logistic regression:** Predicts the probability of belonging to a class.



Scikit-Learn: Machine Learning Library

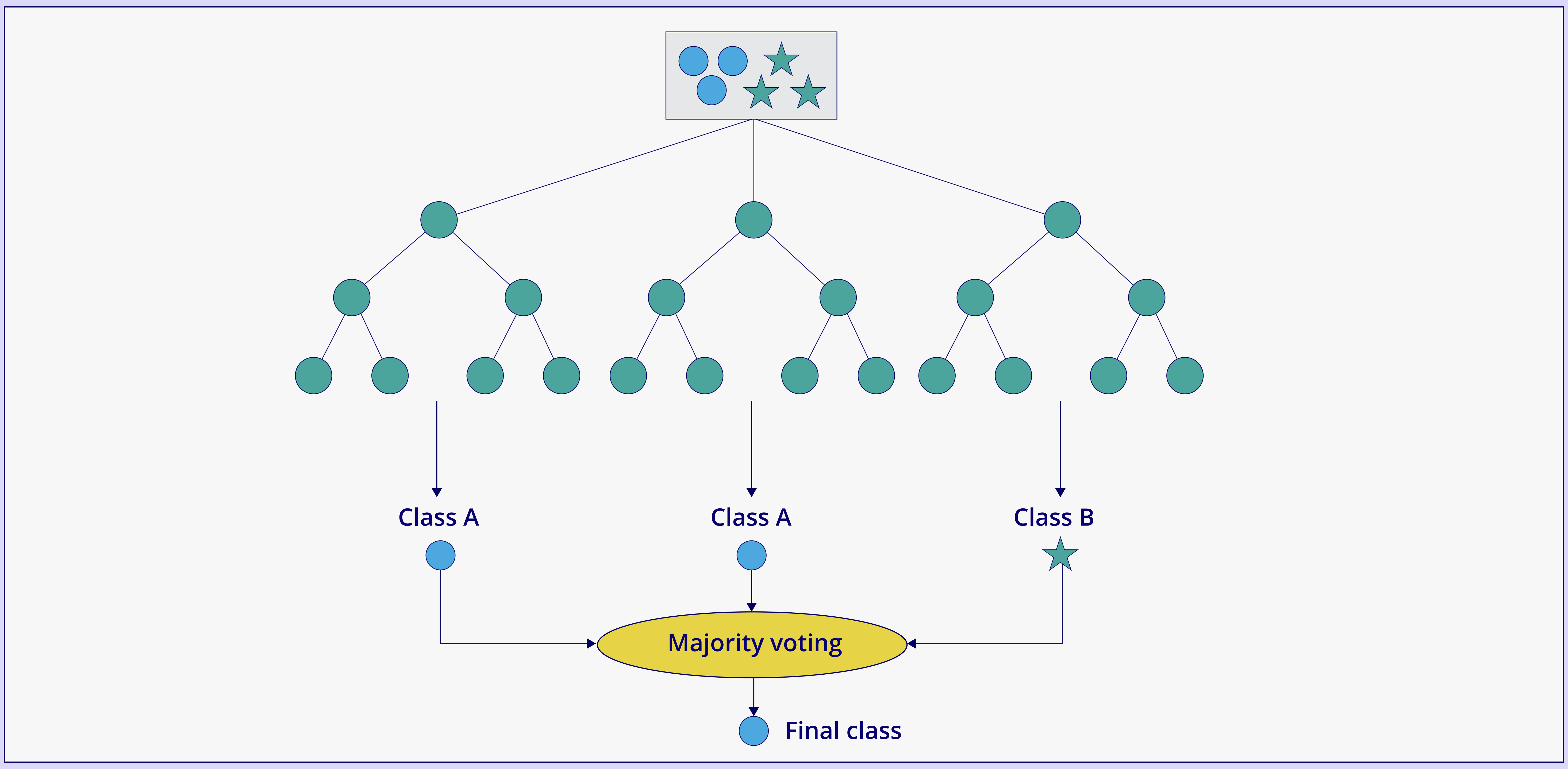

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

- **Decision trees:** Classifies data based on a tree-like structure.



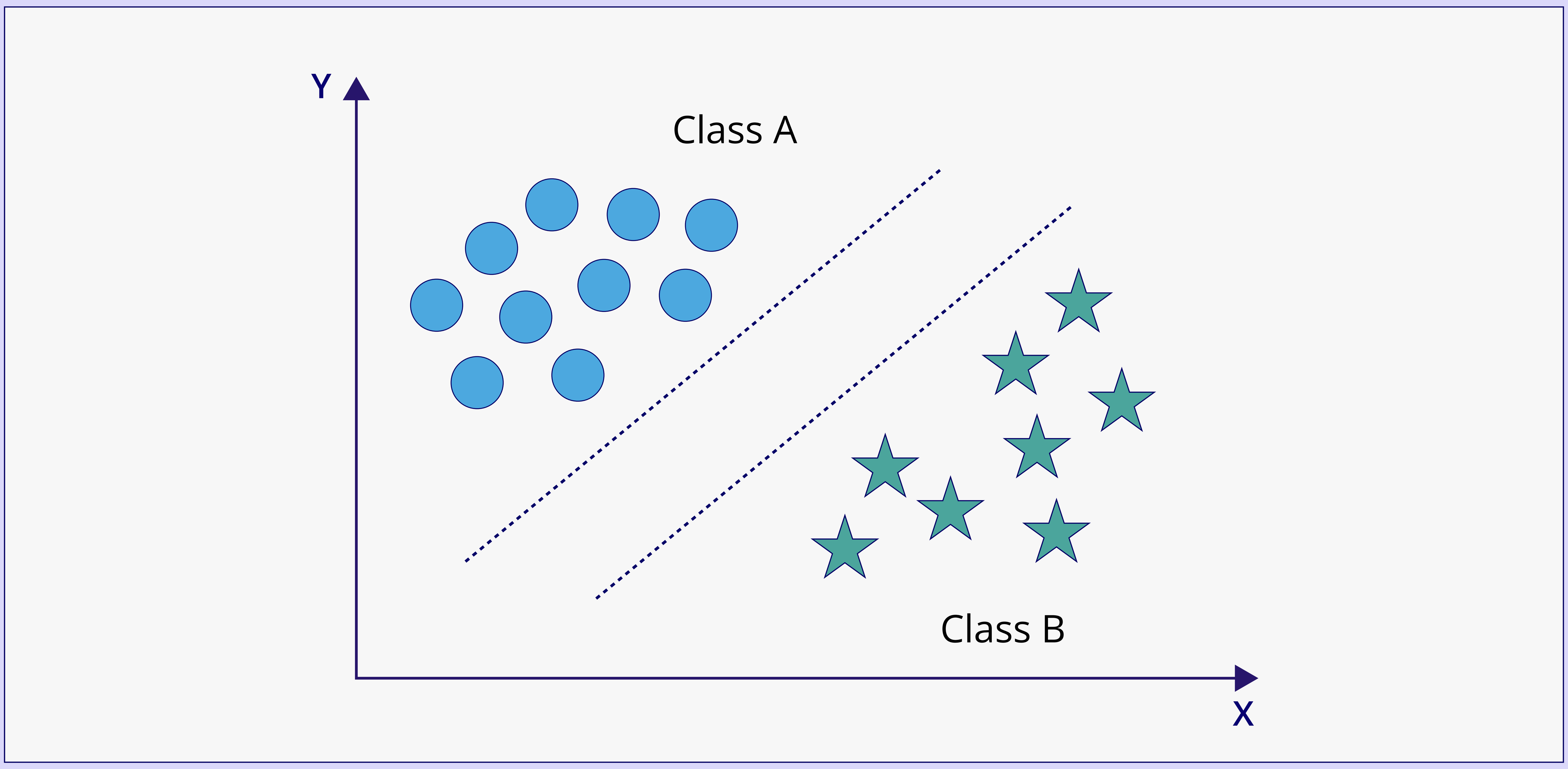

```
from sklearn.tree import DecisionTreeClassifier

tree = DecisionTreeClassifier()
tree.fit(X_train, y_train)
```

- **Random forests:** Ensemble method combining multiple decision trees for improved accuracy.



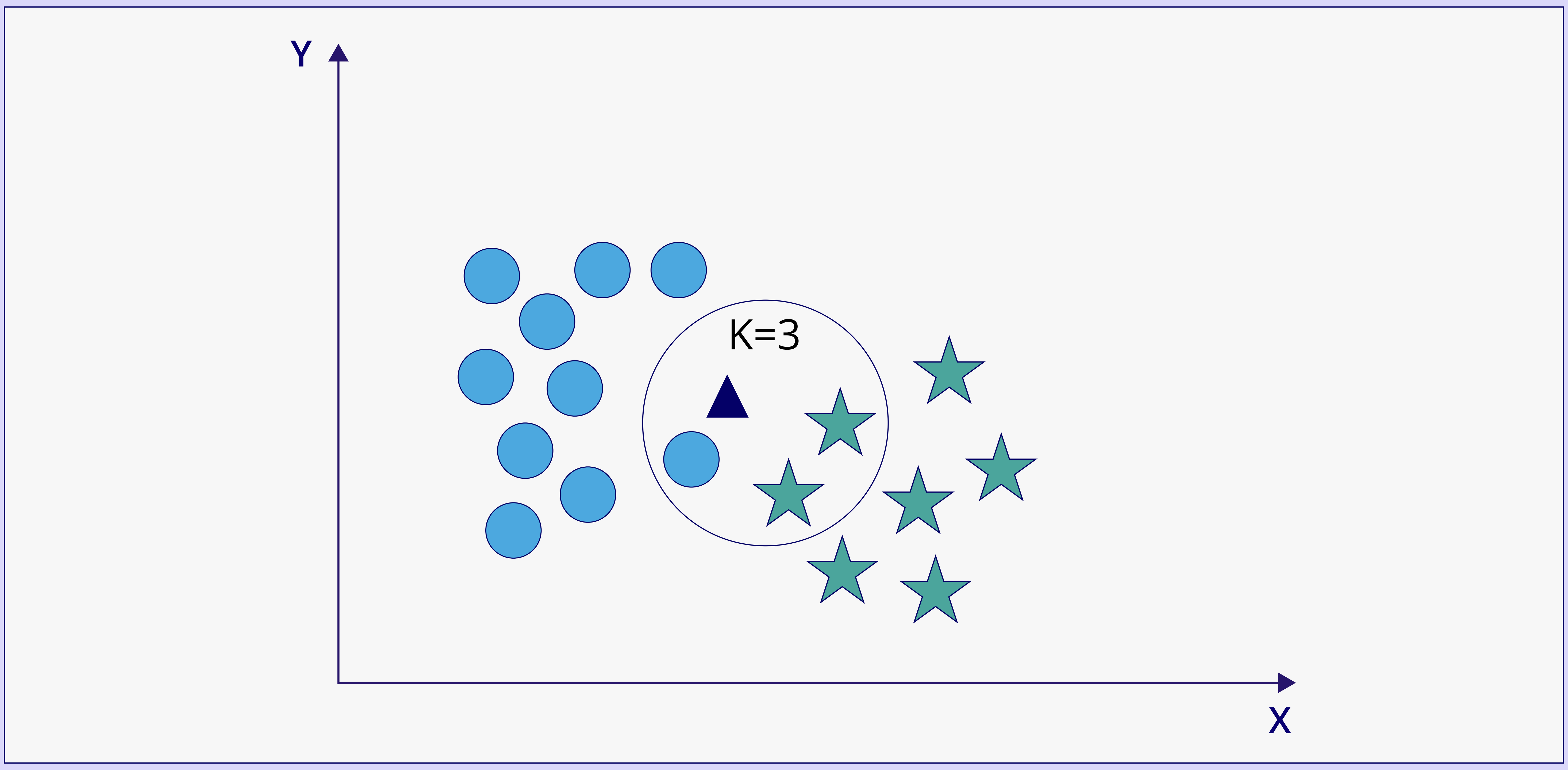

```
from sklearn.ensemble import RandomForestClassifier

forest = RandomForestClassifier()
forest.fit(X_train, y_train)
```

- **Support vector machines (SVM):** Finds hyperplanes to separate data points.




```
from sklearn.svm import SVC

svm = SVC()
svm.fit(X_train, y_train)
```

- **k-nearest neighbors (KNN):** Classifies data points based on the k nearest neighbors.



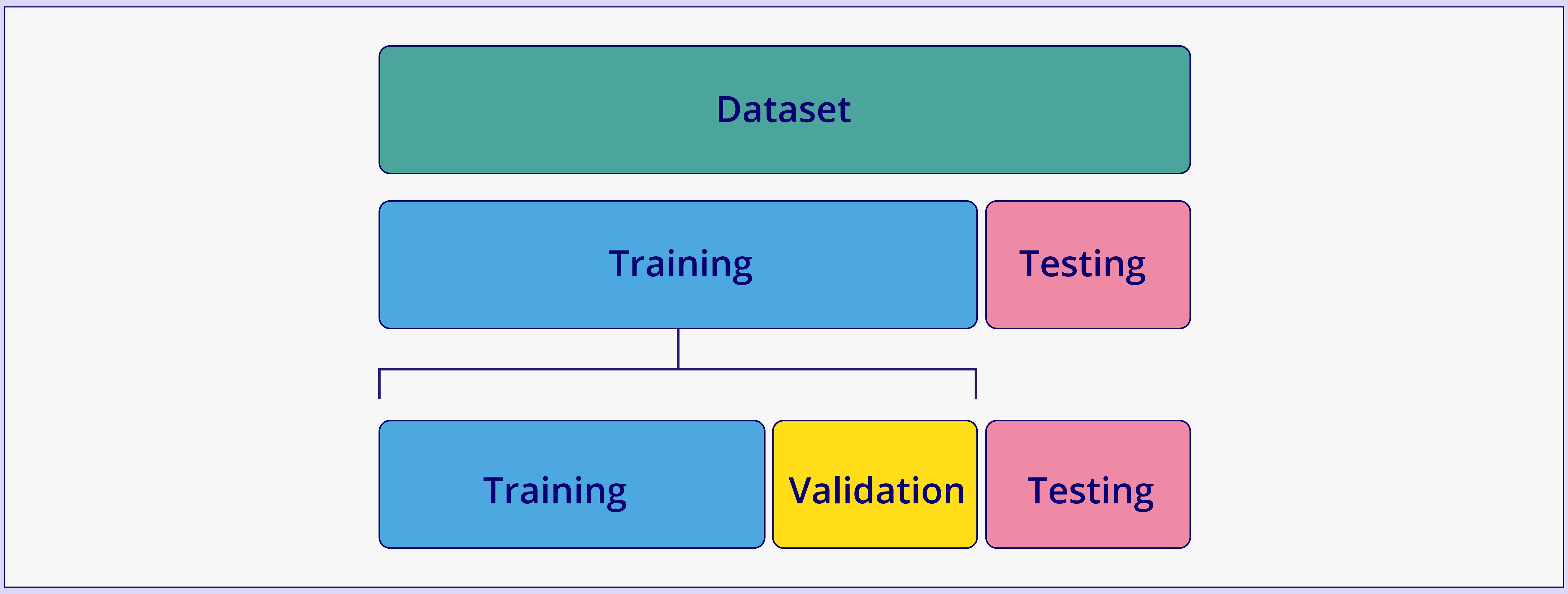

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
```

Scikit-Learn: Machine Learning Library

educative

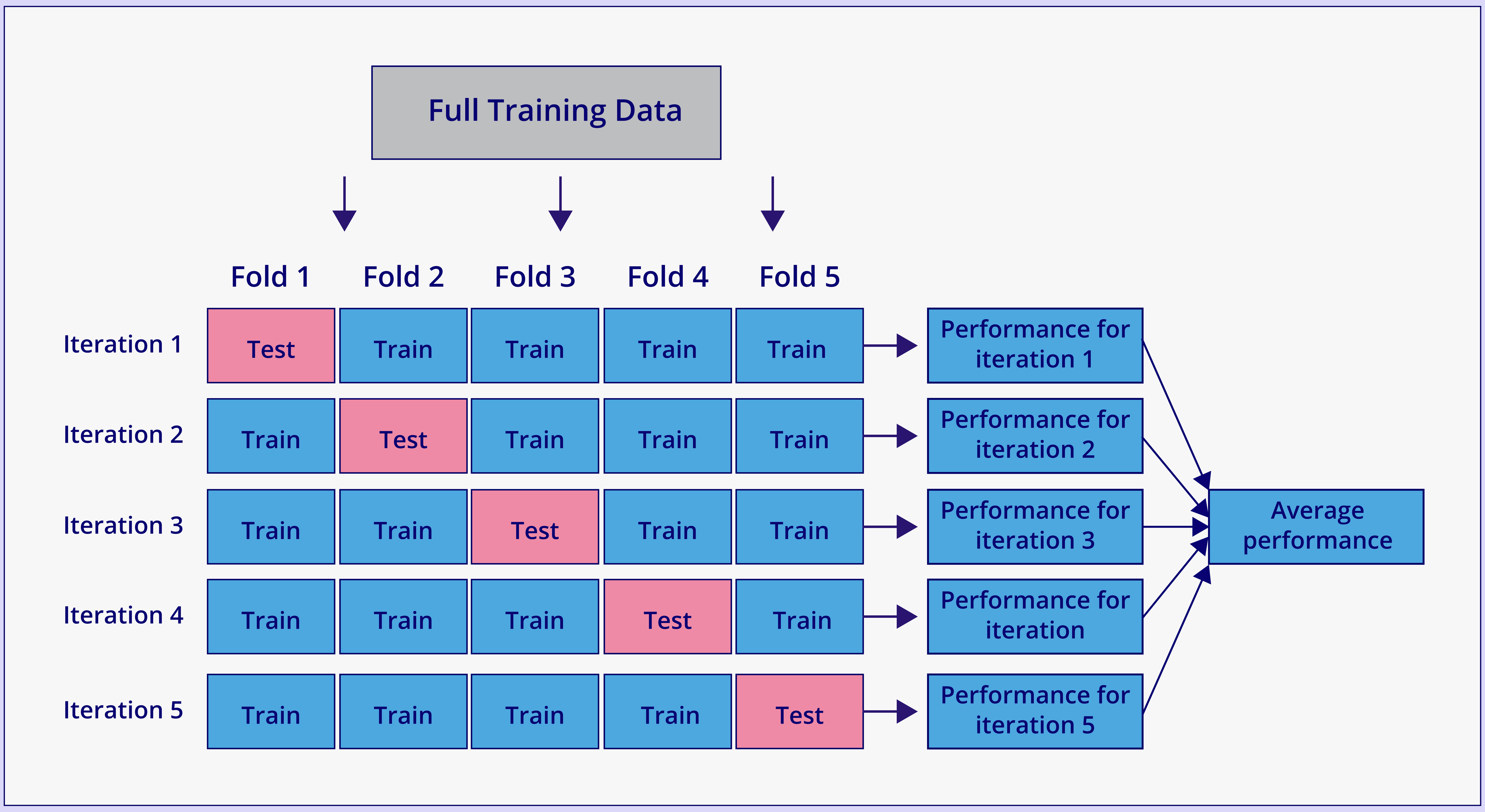
Model Evaluation and Validation

• **Train-test split:** Divide data into training and testing sets for model evaluation.




```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

• **Cross-validation techniques (k-fold and stratified k-fold):** Evaluate model performance using techniques like k-fold or stratified k-fold cross-validation.




```
from sklearn.model_selection import cross_val_score, StratifiedKFold

# k-Fold
scores = cross_val_score(model, X, y, cv=5)

# Stratified k-Fold
skf = StratifiedKFold(n_splits=5)
scores = cross_val_score(model, X, y, cv=skf)
```

Scikit-Learn: Machine Learning Library

educative

- **Metrics for Regression (e.g., Mean Absolute Error, R-squared)**
  - **Mean absolute error (MAE):** Average magnitude of the difference between predicted and actual values.
  - **R-squared:** Proportion of variance in the target variable explained by the model (higher is better for regression)

```
from sklearn.metrics import mean_absolute_error, r2_score

mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

- **Metrics for Classification (e.g., Accuracy, Precision, Recall, F1-Score):**
  - **Accuracy:** Proportion of correctly classified samples (all classes).
  - **Precision:** Ratio of true positives to all predicted positives (measures how well the model identifies actual positives).
  - **Recall:** Ratio of true positives to all actual positives (measures how good the model finds all positives).
  - **F1-score:** Harmonic mean of precision and recall, combining both metrics into a single score.

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
```

Typical Workflows and Use Cases

- **Common Scenarios Where Scikit-Learn Is Useful**
  - Predicting housing prices

```
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error

# Load dataset
boston = load_boston()
X, y = boston.data, boston.target

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)

# Evaluate
print("Mean Absolute Error:", mean_absolute_error(y_test, y_pred))
```

  - Classifying spam emails

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score

# Sample data
emails = ["Free money!!!", "Hi Bob, how are you?", "Win a new car now!", "Meeting tomorrow"]
labels = [1, 0, 1, 0]  # 1: Spam, 0: Not Spam

# Vectorize text data
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(emails)

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.2, random_state=42)

# Train model
model = MultinomialNB()
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)

# Evaluate
print("Accuracy:", accuracy_score(y_test, y_pred))
```

- Segmenting customers

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score

# Sample data
emails = ["Free money!!!", "Hi Bob, how are you?", "Win a new car now!", "Meeting tomorrow"]
labels = [1, 0, 1, 0]  # 1: Spam, 0: Not Spam

# Vectorize text data
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(emails)

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.2, random_state=42)

# Train model
model = MultinomialNB()
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)

# Evaluate
print("Accuracy:", accuracy_score(y_test, y_pred))
```

- Recognizing handwritten digits

```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Sample data
customers = [[20, 50000], [30, 60000], [40, 70000], [50, 80000], [60, 90000]]

# Standardize features
scaler = StandardScaler()
X = scaler.fit_transform(customers)

# Train model
kmeans = KMeans(n_clusters=2)
kmeans.fit(X)

# Get cluster labels
labels = kmeans.labels_
print("Cluster Labels:", labels)
```

Examples of Real-World Applications

- Customer churn prediction

```
from sklearn.ensemble import RandomForestClassifier

# Train a RandomForest classifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)
```

- Image recognition

```
from sklearn.svm import SVC

# Train a Support Vector Machine classifier
clf = SVC(kernel="linear", random_state=42)
clf.fit(X_train_pca, y_train)
```

- Fraud detection

```
from sklearn.ensemble import IsolationForest

# Train an Isolation Forest model
clf = IsolationForest(n_estimators=100, contamination=0.01, random_state=42)
clf.fit(X_train)
```

- Spam filtering

```
from sklearn.naive_bayes import MultinomialNB

# Train a Naive Bayes classifier
clf = MultinomialNB()
clf.fit(X_train, y_train)
```

Top 10 Rules of Thumb

Practical Guidelines for Effective Machine Learning with Scikit-Learn

1: Understand your data: Clean, explore, and understand the data before modeling.

```
import pandas as pd

# Load data
df = pd.read_csv('data.csv')

# Basic exploration
print(df.head())
print(df.describe())
print(df.info())
```

2: Choose the right algorithm: Select the appropriate algorithm based on the problem and data type.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC

# Example: RandomForest for classification
clf_rf = RandomForestClassifier()

# Example: SVC for classification
clf_svc = SVC(kernel='linear')
```

3: Preprocess your data: Handle missing values, scale features, and encode categorical variables.

```
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer

# Impute missing values and scale features
imputer = SimpleImputer(strategy='mean')
scaler = StandardScaler()

X_imputed = imputer.fit_transform(X)
X_scaled = scaler.fit_transform(X_imputed)
```

4: Train-test split: Separate data for training and testing to avoid overfitting.

```
from sklearn.model_selection import train_test_split

# Split data
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=42)
```

5: Cross-validate your model: Evaluate model performance using cross-validation to prevent overfitting.

```
from sklearn.model_selection import cross_val_score

# Cross-validation
scores = cross_val_score(clf, X_train, y_train, cv=5)
print(f'Cross-validation scores: {scores}')
```

6: Tune hyperparameters: Optimize model performance by tuning hyperparameters.

```
from sklearn.model_selection import GridSearchCV

# Define hyperparameter grid
param_grid = {
    'n_estimators': [50, 100],
    'max_depth': [10, 20]
}

# Grid search
grid_search = GridSearchCV(clf, param_grid, cv=5)
grid_search.fit(X_train, y_train)
print(f'Best parameters: {grid_search.best_params_}')
```

7: Evaluate different models: Compare different models and choose the best-performing one.

```
from sklearn.metrics import accuracy_score

# Train and evaluate model
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print(f'Accuracy: {accuracy_score(y_test, y_pred)}')
```

8: Handle class imbalance: Address class imbalance if present in your dataset.

```
from sklearn.utils.class_weight import compute_class_weight

# Compute class weights
class_weights = compute_class_weight('balanced', classes=np.unique(y), y=y)
print(f'Class weights: {class_weights}')
```

9: Feature selection: Choose the most relevant features to improve model efficiency.

```
from sklearn.feature_selection import SelectKBest, chi2

# Feature selection
selector = SelectKBest(chi2, k=10)
X_new = selector.fit_transform(X_train, y_train)
```

10: Understand model assumptions: Be aware of the assumptions behind your chosen algorithm.

```
# Example: Linear regression assumptions
from sklearn.linear_model import LinearRegression

# Train linear regression model
lr = LinearRegression()
lr.fit(X_train, y_train)
```