# Aufgabe 1: Iris-Klassifikation

**Ziel:** Klassifikation der Iris-Blumen in die drei Arten *setosa*, *versicolor* und *virginica*.

**Vorgehen:**

1. **Datenaufbereitung**
   - Datensatz laden und auf Vollständigkeit prüfen.
   - Aufteilen in Trainings- (80 %) und Testdaten (20 %) mit festem Seed:
     - R: `set.seed(42)`
     - Python: `random_state=42`
     - RapidMiner: `random_seed=42`
2. **Modellbildung**
   - R:

```r
model_rf <- randomForest(Species ~ ., data = train)
```

   - Python:

```python
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(random_state=42)
clf.fit(X_train, y_train)
```

   - Orange: File → Data Sampler (80/20) → Random Forest → Test Score
   - RapidMiner: Read CSV → Split Data → Random Forest → Apply Model → Performance
3. **Evaluation**
   - Metriken: Accuracy, Precision, Recall, F1-Score, Confusion Matrix.

**Ergebnisse (Beispiel):**

- Accuracy: 0.97
- Precision: Beste Klasse *virginica* (0.98)

**Interpretation:** Hohe Trennbarkeit, besonders zwischen *setosa* und den anderen beiden Arten.

# Aufgabe 2: Algorithmusvergleich (Decision Tree, Naive Bayes, SVM)

**Ziel:** Vergleich der Klassifikationsleistung dreier Algorithmen auf demselben Datensatz und Splits.

**Vorgehen:**

1. Datenbasis: Iris oder anderer Benchmark-Datensatz.
2. Splits: Wie in Aufgabe 1.
3. Modelle:
   - **Decision Tree**
     - R: `rpart()`
     - Python: `DecisionTreeClassifier()`
     - Orange: Decision Tree-Widget
     - RapidMiner: Decision Tree → Apply Model
   - **Naive Bayes**
     - R: `e1071::naiveBayes()`
     - Python: `GaussianNB()`
     - Orange: Naive Bayes-Widget
     - RapidMiner: Naive Bayes → Apply Model
   - **SVM**
     - R: `e1071::svm()`
     - Python: `SVC()`
     - Orange: SVM-Widget
     - RapidMiner: SVM → Apply Model
4. Evaluation & Vergleich:
   - Accuracy, Precision, Recall, F1
   - ROC-AUC (binär) bzw. Micro-/Macro-AUC (multiklassig)

**Ergebnisse (Beispiel):**

| Algorithmus | Accuracy | ROC-AUC |
|---|---|---|
| Decision Tree | 0.93 | 0.95 |
| Naive Bayes | 0.95 | 0.96 |
| SVM | 0.96 | 0.98 |

**Interpretation:** SVM zeigt insgesamt höchste Genauigkeit und AUC.

# Aufgabe 3: Unüberwachtes Clustering (Rotwein-Daten)

**Ziel:** Strukturen in den Rotwein-Daten entdecken mittels K-Means und hierarchischem Clustering.

**Vorgehen:**

1. Preprocessing:
   - Fehlende Werte behandeln
   - Z-Score-Skalierung aller Merkmale
2. Bestimmung k:
   - Elbow-Plot (Within-Cluster-Sum-of-Squares)
   - Silhouette-Score
3. Clustering:
   - K-Means (R: `kmeans()`, Python: `KMeans()`, Orange: K-Means-Widget, RapidMiner: K-Means-Operator)
   - Hierarchical (Ward)
4. Auswertung:
   - Silhouette-Score, Cluster-Profile (Mittelwerte)
   - Visualisierung (Dendrogramm, PCA-Plot)

**Ergebnisse (Beispiel):**

- Optimaler k = 3
- Cluster unterscheiden sich vor allem im Phenol-Gehalt

**Interpretation:** Zwei Cluster mit hohem bzw. niedrigem Phenolgehalt; ein drittes Cluster intermediär.

# Aufgabe 4: Google Trends Clustering

**Ziel:** Regionale Suchmuster in Google Trends-Zeitreihen clustern.

**Vorgehen:**

1. Datenbeschaffung: CSV-Export aus Google Trends
2. Preprocessing:
   - Fehlende Werte imputieren oder entfernen
   - Normalisierung/Standardisierung
3. Feature-Matrix: Regionen als Beobachtungen, Suchbegriffe/Zeiträume als Merkmale
4. Clustering: Wie in Aufgabe 3 (K-Means, hierarchisch)
5. Visualisierung:
   - PCA-Scatterplots
   - Kartenplot (z.B. mit Python `geopandas` oder Orange-Geo-Widget)

**Ergebnisse (Beispiel):**

- Drei Cluster: saisonale Peaks, stabile Volumina, volatile Trends

**Interpretation:** Saisonale Urlaubsregionen vs. ganzjährig beliebte Destinationen vs. gering frequentierte Gebiete.

*Hinweis: Alle Arbeitsschritte wurden in R, Python (scikit-learn), Orange und RapidMiner (Repdiminer) implementiert, um Tool-typische Unterschiede in Usability und Konfigurationsmöglichkeiten zu vergleichen.*