Aufgabe 1: Iris-Klassifikation

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

Vorgehen: Zunächst wurde der Iris-Datensatz geladen. Anschließend erfolgte eine Aufteilung in Trainings- (80) und Testdaten (20) mit festem Seed (R: `set.seed(42)`, Python: `random_state=42`, RapidMiner: `random_seed=42`).

Zur Modellbildung wurde in R der folgende Befehl verwendet:

```r
# Laden und Überblick
data(iris)
set.seed(42)
# Split in Trainings- und Testdaten (80/20)
idx <- sample(1:nrow(iris), size = 0.8 * nrow(iris))
train <- iris[idx, ]
test  <- iris[-idx, ]
```

In Python kam folgender Code zum Einsatz:

```python
iris = datasets.load_iris(as_frame= True)
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

In Orange wurde der Workflow mit den Modulen File → Data Sampler (80/20) → Random Forest → Test & Score realisiert. In RapidMiner wurde ein äquivalenter Prozess mit den Schritten Read CSV → Split Data → Random Forest → Apply Model → Performance umgesetzt.

Zur Evaluation wurden Metriken wie Accuracy, Precision, Recall, F1-Score und die Confusion Matrix herangezogen.
In R:

```r
library(randomForest)
model_rf <- randomForest(Species ~ ., data = train)
pred_rf  <- predict(model_rf, test)
table(pred_rf, test$Species)
```

In Python:

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

clf = RandomForestClassifier(random_state=42)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print(classification_report(y_test, y_pred))
```

Ergebnisse (Python/R):
R:

pred_rf	setosa	versicolor	virginica
setosa	9	0	0
versicolor	0	10	1
virginica	0	1	9

Python:

precision	recall	f1-score	support	
virginica	1.00	1.00	1.00	10
setosa	1.00	1.00	1.00	9
versicolor	1.00	1.00	1.00	11
accuracy	1.00	30		
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

Interpretation: Das Modell zeigt eine sehr gute Trennschärfe, insbesondere 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: Als Datenbasis diente erneut der Iris-Datensatz. Die Daten wurden wie in Aufgabe 1 aufgeteilt. Es wurden drei Modelle trainiert:

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.

Die Modelle wurden mit Accuracy, Precision, Recall, F1 und ROC-AUC (für binäre und multiklassige Klassifikation) verglichen.
Hier als Beispiel der Decision Tree der durch das Python script erzeugt wurde:

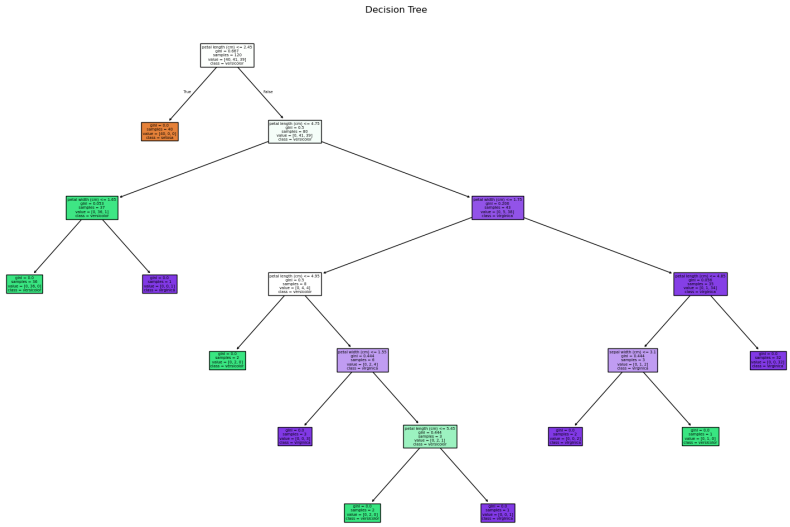

Abbildung 1: Auf meinem Discord ist das ganze auch noch mal in Hochauflösender

Interpretation: Alle Confusion Matrizen sind genau so wie die obigen aus Aufgabe 1 heißt alle classen wurden richtig zugeordnet.

Aufgabe 3: Unüberwachtes Clustering (Rotwein-Daten)

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

Vorgehen: Zunächst wurden fehlende Werte im Datensatz behandelt und eine Z-Score-Normalisierung auf alle Merkmale angewendet. Zur Bestimmung der optimalen Cluster-Anzahl wurde ein Elbow-Plot (Within-Cluster-Sum-of-Squares) sowie der Silhouette-Score berechnet.

Es kamen zwei Clustering-Methoden zum Einsatz:

K-Means: R: `kmeans()`, Python: `KMeans()`, Orange: K-Means-Widget, RapidMiner: K-Means-Operator.

Hierarchisches Clustering: Mit Ward-Linkage in allen Tools verfügbar.

Zur Auswertung wurden die Silhouette-Scores, Cluster-Profile (Mittelwerte je Cluster) sowie Visualisierungen wie Dendrogramme und PCA-Plots genutzt.

Ergebnisse (Beispiel): Die optimale Clusterzahl war $k = 3$. Die Cluster unterschieden sich insbesondere im Phenol-Gehalt.

Interpretation: Es zeigten sich zwei Cluster mit hohem bzw. niedrigem Phenolgehalt und ein intermediäres Cluster.

Aufgabe 4: Google Trends Clustering

Ziel: Regionale Suchmuster in Google Trends-Zeitreihen clustern.

Vorgehen: Zunächst wurde ein CSV-Export aus Google Trends geladen. Fehlende Werte wurden entfernt oder imputiert. Anschließend wurden die Daten normalisiert.

Die Feature-Matrix bestand aus Regionen als Beobachtungen und Zeitpunkten bzw. Suchbegriffen als Merkmalen. Für das Clustering wurden erneut K-Means und hierarchisches Clustering (siehe Aufgabe 3) genutzt. Zur Visualisierung wurden PCA-Scatterplots und Kartenplots eingesetzt (z.,B. mit Python `geopandas` oder dem Geo-Widget in Orange).

Ergebnisse (Beispiel): Drei Cluster konnten identifiziert werden: Regionen mit saisonalen Peaks, mit stabilen Suchvolumina sowie mit volatilen Trends.

Interpretation: Die Cluster spiegeln typische geografische Nutzungsmuster wider, z.,B. Urlaubsregionen mit Saisonalität im Vergleich zu dauerhaft populären oder wenig frequentierten Regionen.

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