```
#%% md
# **INTRO**

1. [Daten](#1)
2. [Modelle](#2):
   - [Logistic Regression](#3)
   - [Naive Bayes](#4)
   - [Support Vector Machine (SVM)](#5)
   - [KNN - K NEAREST NEIGHBOUR](#6)
   - [Decision Tree](#7)
   - [Random Forest](#8)
3. [Vergleich](#9)


# Beschreibung des Datensatzes

Der Datensatz enthält **7 Merkmale (Features)** und **eine Zielspalte (Label)**.
Er basiert auf synthetischen Gesundheits- und Lifestyle-Daten und eignet sich für Klassifikationsaufgaben.

---

## Merkmale (Features)

- **bmi** – Körpergewicht im Verhältnis zur Körpergröße (Body-Mass-Index)
- **daily_steps** – Durchschnittliche Anzahl an Schritten pro Tag, zeigt das Aktivitätsniveau
- **sleep_hours** – Durchschnittliche Schlafdauer pro Nacht, steht für den Erholungsfaktor
- **calories_consumed** – Durchschnittliche Kalorienaufnahme pro Tag, zeigt den Ernährungszustand
- **cholesterol** – Cholesterinwert, wichtiger Indikator für das Herz-Kreislauf-System
- **systolic_bp** / **diastolic_bp** – Blutdruckwerte (systolisch und diastolisch), geben Auskunft über den Gefäßdruck
- **family_history** – Gibt an, ob in der Familie gesundheitliche Vorbelastungen bestehen (0 = nein, 1 = ja)

---

## Zielspalte (Label)

- **disease_risk** – Klassifizierungslabel:
  - `0` = gesund
  - `1` = erhöhtes Krankheitsrisiko


#%%
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.simplefilter("ignore")

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report


#%% md
<a id="1"></a>
# **1. Daten**

#%%
# === Daten laden ===
df = pd.read_csv("health_lifestyle_dataset.csv")
df.head()

#%%
df.info()

#%%
# === Verteilung der Zielvariable ===
sns.countplot(x='disease_risk', data=df)
plt.title("Verteilung der Zielvariable (0 = gesund, 1 = Risiko)")
plt.show()

#%%
# === Beispiel-Visualisierung: BMI vs. Cholesterin (alle Daten) ===
plt.figure(figsize=(6, 4))
sns.scatterplot(
    x='bmi',
    y='cholesterol',
    hue='disease_risk',
    data=df,
    palette='coolwarm',
    alpha=0.4,
    s=10
```

```
)
plt.title("Zusammenhang zwischen BMI und Cholesterin (alle Daten)")
plt.xlabel("BMI")
plt.ylabel("Cholesterinwert (mg/dL)")
plt.show()

# === Beispiel-Visualisierung: BMI vs. Cholesterin (5 000er Stichprobe) ===
sample = df.sample(5000)

plt.figure(figsize=(6, 4))
sns.scatterplot(
    x='bmi',
    y='cholesterol',
    hue='disease_risk',
    data=sample,
    palette='coolwarm',
    alpha=0.6,
    s=30
)
plt.title("Zusammenhang zwischen BMI und Cholesterin (5 000 Stichprobe)")
plt.xlabel("BMI")
plt.ylabel("Cholesterinwert (mg/dL)")
plt.show()

#%%
# === Features (X) und Zielvariable (y) ===
X = df[[
    "bmi",
    "daily_steps",
    "sleep_hours",
    "calories_consumed",
    "cholesterol",
    "systolic_bp", "diastolic_bp",
    "family_history"
]]
y = df["disease_risk"]

print("X-Shape:", X.shape)
print("y-Shape:", y.shape)

#%% md
# **2. Modelle**

#%% md
<a id = "3"></a><br>
## **Logistic Regression Classifaction**

#%%
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report

# === Daten splitten (30 % Test) ===
x_train, x_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

# === Modell trainieren ===
logistic_regression = LogisticRegression(max_iter=1000, random_state=42)
logistic_regression.fit(x_train, y_train)

# === Accuracy ausgeben ===
print("Logistic Regression – Test Accuracy:", logistic_regression.score(x_test, y_test))

#%%
# === Logistic Regression – Vergleich über verschiedene Testanteile ===
test_sizes = [0.1, 0.3, 0.5, 0.7, 0.9]
rows = []

for t in test_sizes:
    x_train, x_test, y_train, y_test = train_test_split(
        X, y, test_size=t, random_state=42, stratify=y
    )
    lr = LogisticRegression(max_iter=1000, random_state=42)
    lr.fit(x_train, y_train)
    acc = lr.score(x_test, y_test)
    rows.append({"Testanteil": t, "Trainanteil": 1 - t, "Accuracy": acc})
    print(f"Testanteil {int(t*100)}% – Accuracy: {acc:.4f}")

# === Tabelle & Plot ===
lr_acc = pd.DataFrame(rows)
display(lr_acc.style.format({"Testanteil": "{:.0%}", "Trainanteil": "{:.0%}", "Accuracy": "{:.3f}"}))
```

```
plt.figure(figsize=(7, 4))
sns.lineplot(data=lr_acc, x="Testanteil", y="Accuracy", marker="o", color="teal")
plt.title("Logistic Regression – Accuracy über verschiedene Testanteile")
plt.xlabel("Testanteil")
plt.ylabel("Accuracy")
plt.ylim(lr_acc["Accuracy"].min() - 0.02, lr_acc["Accuracy"].max() + 0.02)
plt.grid(True)
plt.tight_layout()
plt.show()

lr_acc.to_csv("logreg_accuracy_by_split.csv", index=False)
print("Gespeichert: logreg_accuracy_by_split.csv")

#%%
# === Logistic Regression – Confusion Matrix & Report für verschiedene Testanteile ===
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt

test_sizes = [0.1, 0.3, 0.5, 0.7, 0.9]

for split_p in test_sizes:
    print(f"\n=== Testanteil: {int(split_p * 100)}% ===")

    # Daten splitten
    x_train, x_test, y_train, y_test = train_test_split(
        X, y, test_size=split_p, random_state=42, stratify=y
    )

    # Modell trainieren
    lr = LogisticRegression(max_iter=1000, random_state=42)
    lr.fit(x_train, y_train)

    # Vorhersage
    y_pred = lr.predict(x_test)
    acc = lr.score(x_test, y_test)

    # Ergebnisse ausgeben
    print(f"Accuracy: {acc:.4f}")
    print("Classification Report:\n", classification_report(y_test, y_pred, digits=3))

    # Confusion Matrix visualisieren
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(4, 4))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)
    plt.title(f"Logistic Regression – Confusion Matrix ({int(split_p * 100)}% Test)")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.tight_layout()
    plt.show()

#%%
# === Logistic Regression – Übersichtstabelle & Plot (Accuracy je Testanteil) ===
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import pandas as pd
import numpy as np

test_sizes = [0.1, 0.3, 0.5, 0.7, 0.9]
rows = []

for t in test_sizes:
    x_train, x_test, y_train, y_test = train_test_split(
        X, y, test_size=t, random_state=42, stratify=y
    )
    lr = LogisticRegression(max_iter=1000, random_state=42).fit(x_train, y_train)
    acc = lr.score(x_test, y_test)
    rows.append({"Testanteil": t, "Trainanteil": 1 - t, "Accuracy": acc})

lr_acc = pd.DataFrame(rows)
display(lr_acc.style.format({"Testanteil": "{:.0%}", "Trainanteil": "{:.0%}", "Accuracy": "{:.3f}"}))

plt.figure(figsize=(7, 4))
sns.lineplot(data=lr_acc, x="Testanteil", y="Accuracy", marker="o")
plt.xticks(test_sizes, [f"{int(t*100)}%" for t in test_sizes])
plt.ylim(lr_acc["Accuracy"].min() - 0.02, lr_acc["Accuracy"].max() + 0.02)
plt.yticks(np.arange(round(lr_acc["Accuracy"].min(), 2),
                     round(lr_acc["Accuracy"].max() + 0.02, 2), 0.01))
plt.ticklabel_format(style="plain", axis="y")
plt.title("Logistic Regression – Accuracy über verschiedene Testanteile")
plt.xlabel("Testanteil")
```

```
plt.ylabel("Accuracy")
plt.grid(True)
plt.tight_layout()
plt.show()

# Optional: für Bericht speichern
lr_acc.to_csv("lr_accuracy_by_split.csv", index=False)
print("Gespeichert: lr_accuracy_by_split.csv")

#%% md
<a id = "3"></a><br>

## **Naive Bayes Classifaction**


#%%
# === Naive Bayes: Einzellauf (30 % Test) ===
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB

x_train, x_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

naive_bayes = GaussianNB()
naive_bayes.fit(x_train, y_train)

print("Test accuracy: ", naive_bayes.score(x_test, y_test))

#%%
# === Naive Bayes: 10/30/50/70/90 % Testanteil ===
test_sizes = [0.1, 0.3, 0.5, 0.7, 0.9]
nb_rows = []

for t in test_sizes:
    x_train, x_test, y_train, y_test = train_test_split(
        X, y, test_size=t, random_state=42, stratify=y
    )
    naive_bayes = GaussianNB().fit(x_train, y_train)
    acc = naive_bayes.score(x_test, y_test)
    nb_rows.append({"Testanteil": t, "Trainanteil": 1 - t, "Accuracy": acc})
    print(f"Testanteil {int(t*100)}% - Accuracy: {acc:.4f}")

import pandas as pd, seaborn as sns, matplotlib.pyplot as plt
nb_df = pd.DataFrame(nb_rows)
display(nb_df)

plt.figure(figsize=(7, 4))
sns.lineplot(data=nb_df, x="Testanteil", y="Accuracy", marker="o", color="steelblue")

ax = plt.gca()
# Y-Achse normal darstellen (keine 1e-5 Schreibweise)
ax.ticklabel_format(style="plain", axis="y")
ax.yaxis.set_major_formatter(plt.FormatStrFormatter("%.3f"))

# Achsen-Layout vereinheitlichen
plt.ylim(nb_df["Accuracy"].min() - 0.005, nb_df["Accuracy"].max() + 0.005)
plt.yticks(np.arange(round(nb_df["Accuracy"].min(), 3) - 0.005,
                     round(nb_df["Accuracy"].max(), 3) + 0.005, 0.002))

plt.title("Naive Bayes – Accuracy über verschiedene Testanteile")
plt.xlabel("Testanteil")
plt.ylabel("Accuracy")
plt.grid(True)
plt.tight_layout()
plt.show()



#%%
# === Naive Bayes – Alle Splits (10, 30, 50, 70, 90 %) ===
from sklearn.metrics import confusion_matrix, classification_report

test_sizes = [0.1, 0.3, 0.5, 0.7, 0.9]

for split_p in test_sizes:
    print(f"\n=== Testanteil: {int(split_p * 100)}% ===")

    x_train, x_test, y_train, y_test = train_test_split(
        X, y, test_size=split_p, random_state=42, stratify=y
    )

    naive_bayes = GaussianNB().fit(x_train, y_train)
```

```
    y_pred = naive_bayes.predict(x_test)

    acc = naive_bayes.score(x_test, y_test)
    print(f"Accuracy: {acc:.4f}")
    print("Classification Report:\n", classification_report(y_test, y_pred))

    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(4, 4))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)
    plt.title(f"Confusion Matrix ({int(split_p * 100)}% Test)")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()

#%%
# === Naive Bayes – Übersichtstabelle & Plot (Accuracy je Testanteil) ===
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
import pandas as pd

test_sizes = [0.1, 0.3, 0.5, 0.7, 0.9]
rows = []

for t in test_sizes:
    x_train, x_test, y_train, y_test = train_test_split(
        X, y, test_size=t, random_state=42, stratify=y
    )
    nb = GaussianNB().fit(x_train, y_train)
    acc = nb.score(x_test, y_test)
    rows.append({"Testanteil": t, "Trainanteil": 1 - t, "Accuracy": acc})

nb_acc = pd.DataFrame(rows)
display(nb_acc.style.format({"Testanteil": "{:.0%}", "Trainanteil": "{:.0%}", "Accuracy": "{:.3f}"}))

# Plot
plt.figure(figsize=(7, 4))
sns.lineplot(data=nb_acc, x="Testanteil", y="Accuracy", marker="o")
plt.xticks(test_sizes, [f"{int(t*100)}%" for t in test_sizes])
plt.ylim(0, 1)
plt.title("Naive Bayes – Accuracy über verschiedene Testanteile")

plt.xlabel("Testanteil")
plt.ylabel("Accuracy")
plt.grid(True)
plt.tight_layout()
plt.show()

nb_acc.to_csv("nb_accuracy_by_split.csv", index=False)
print("Gespeichert: nb_accuracy_by_split.csv")

#%% md
<a id = "4"></a><br>

## **Support Vector Machine (SVM) Classifaction**



#%%
# === SVM: Einzellauf (30 % Test) ===
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC

x_train, x_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

svm = SVC()  # Template-nah (ohne Pipeline)
svm.fit(x_train, y_train)

print("SVM Test accuracy:", svm.score(x_test, y_test))

#%%
# === SVM: 10/30/50/70/90 % Testanteil ===
test_sizes = [0.1, 0.3, 0.5, 0.7, 0.9]
svm_rows = []

for t in test_sizes:
    x_train, x_test, y_train, y_test = train_test_split(
        X, y, test_size=t, random_state=42, stratify=y
    )
    svm = SVC().fit(x_train, y_train)
    acc = svm.score(x_test, y_test)
    svm_rows.append({"Testanteil": t, "Trainanteil": 1 - t, "Accuracy": acc})
```

```
    print(f"Testanteil {int(t*100)}% – Accuracy: {acc:.4f}")

svm_df = pd.DataFrame(svm_rows)
display(svm_df)

plt.figure(figsize=(7, 4))
sns.lineplot(data=svm_df, x="Testanteil", y="Accuracy", marker="o", color="darkred")

ax = plt.gca()
ax.ticklabel_format(style="plain", axis="y")
ax.yaxis.set_major_formatter(plt.FormatStrFormatter("%.3f"))

plt.ylim(svm_df["Accuracy"].min() - 0.005, svm_df["Accuracy"].max() + 0.005)
plt.yticks(np.arange(round(svm_df["Accuracy"].min(), 3) - 0.005,
                     round(svm_df["Accuracy"].max(), 3) + 0.005, 0.002))

plt.title("SVM – Accuracy über verschiedene Testanteile")
plt.xlabel("Testanteil")
plt.ylabel("Accuracy")
plt.grid(True)
plt.tight_layout()
plt.show()

#%%
# === SVM: Confusion Matrices für alle Testanteile ===
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split

test_sizes = [0.1, 0.3, 0.5, 0.7, 0.9]

for split_p in test_sizes:
    print(f"\n=== Testanteil: {int(split_p * 100)}% ===")

    x_train, x_test, y_train, y_test = train_test_split(
        X, y, test_size=split_p, random_state=42, stratify=y
    )

    svm = SVC().fit(x_train, y_train)
    y_pred = svm.predict(x_test)

    acc = svm.score(x_test, y_test)
    print(f"Accuracy: {acc:.4f}")
    print("Classification Report:\n", classification_report(y_test, y_pred))

    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(4, 4))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)
    plt.title(f"SVM – Confusion Matrix ({int(split_p * 100)}% Test)")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()

#%%
# === SVM – Übersichtstabelle & Plot (Accuracy je Testanteil) ===
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
import pandas as pd

test_sizes = [0.1, 0.3, 0.5, 0.7, 0.9]
rows = []

for t in test_sizes:
    x_train, x_test, y_train, y_test = train_test_split(
        X, y, test_size=t, random_state=42, stratify=y
    )
    svm = SVC().fit(x_train, y_train)
    acc = svm.score(x_test, y_test)
    rows.append({"Testanteil": t, "Trainanteil": 1 - t, "Accuracy": acc})

svm_acc = pd.DataFrame(rows)
display(svm_acc.style.format({"Testanteil": "{:.0%}", "Trainanteil": "{:.0%}", "Accuracy": "{:.3f}"}))

plt.figure(figsize=(7, 4))
sns.lineplot(data=svm_acc, x="Testanteil", y="Accuracy", marker="o")
plt.xticks(test_sizes, [f"{int(t*100)}%" for t in test_sizes])
plt.ylim(0, 1)
plt.title("SVM – Accuracy über verschiedene Testanteile")
plt.xlabel("Testanteil")
plt.ylabel("Accuracy")
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
svm_acc.to_csv("svm_accuracy_by_split.csv", index=False)
print("Gespeichert: svm_accuracy_by_split.csv")

#%% md
<a id = "5"></a><br>

## **KNN - K NEAREST NEIGHBOUR**



#%%
# === KNN: Einzellauf (30 % Test) ===
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

x_train, x_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

knn = KNeighborsClassifier(n_neighbors=5)  # Standardwert, wie im Template
knn.fit(x_train, y_train)

print("KNN Test accuracy:", knn.score(x_test, y_test))

#%%
# === KNN: 10/30/50/70/90 % Testanteil (k=5) – Tabelle + Plot ===
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
import pandas as pd

test_sizes = [0.1, 0.3, 0.5, 0.7, 0.9]
rows = []

for t in test_sizes:
    x_train, x_test, y_train, y_test = train_test_split(
        X, y, test_size=t, random_state=42, stratify=y
    )
    knn = KNeighborsClassifier(n_neighbors=5).fit(x_train, y_train)
    acc = knn.score(x_test, y_test)
    rows.append({"Testanteil": t, "Trainanteil": 1 - t, "Accuracy": acc})
    print(f"Testanteil {int(t*100)}% – Accuracy: {acc:.4f}")

knn_acc = pd.DataFrame(rows)
display(knn_acc.style.format({"Testanteil": "{:.0%}", "Trainanteil": "{:.0%}", "Accuracy": "{:.3f}"}))

plt.figure(figsize=(7, 4))
sns.lineplot(data=knn_acc, x="Testanteil", y="Accuracy", marker="o")
plt.xticks(test_sizes, [f"{int(t*100)}%" for t in test_sizes])
plt.ylim(0, 1)
plt.title("KNN (k=5) – Accuracy über verschiedene Testanteile")
plt.ylim(knn_acc["Accuracy"].min() - 0.02, knn_acc["Accuracy"].max() + 0.02)
plt.yticks(np.arange(round(knn_acc["Accuracy"].min(), 2),
                     round(knn_acc["Accuracy"].max() + 0.02, 2), 0.01))

plt.xlabel("Testanteil")
plt.ylabel("Accuracy")
plt.grid(True)
plt.tight_layout()
plt.show()

knn_acc.to_csv("knn_accuracy_by_split.csv", index=False)
print("Gespeichert: knn_accuracy_by_split.csv")

#%%
# === KNN: Confusion Matrices für alle Testanteile ===
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split

test_sizes = [0.1, 0.3, 0.5, 0.7, 0.9]

for split_p in test_sizes:
    print(f"\n=== Testanteil: {int(split_p * 100)}% ===")

    x_train, x_test, y_train, y_test = train_test_split(
        X, y, test_size=split_p, random_state=42, stratify=y
    )

    # Modelltraining mit k = 5
    knn = KNeighborsClassifier(n_neighbors=5).fit(x_train, y_train)
    y_pred = knn.predict(x_test)
```

```
    # Genauigkeit & Report
    acc = knn.score(x_test, y_test)
    print(f"Accuracy: {acc:.4f}")
    print("Classification Report:\n", classification_report(y_test, y_pred))

    # Confusion Matrix visualisieren
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(4, 4))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)
    plt.title(f"KNN (k=5) – Confusion Matrix ({int(split_p * 100)}% Test)")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.tight_layout()
    plt.show()

#%%
# === KNN: k-Werte bei verschiedenen Testanteilen (10/30/50/70/90 %) ===
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report
import pandas as pd
import numpy as np

test_sizes = [0.1, 0.3, 0.5, 0.7, 0.9]
k_list = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21]

rows = []

# --- Sweep über Testanteile und k ---
for t in test_sizes:
    x_train, x_test, y_train, y_test = train_test_split(
        X, y, test_size=t, random_state=42, stratify=y
    )
    for k in k_list:
        knn = KNeighborsClassifier(n_neighbors=k).fit(x_train, y_train)
        acc = knn.score(x_test, y_test)
        rows.append({"Testanteil": t, "k": k, "Accuracy": acc})

# === Tabelle: jede Zeile = k, jede Spalte = Testanteil ===
knn_grid = pd.DataFrame(rows)
knn_pivot = knn_grid.pivot(index="k", columns="Testanteil", values="Accuracy").sort_index()
display(knn_pivot.style.format("{:.3f}"))

# === Linienplot: Accuracy vs. k, farblich je Testanteil ===
plt.figure(figsize=(8, 5))
sns.lineplot(data=knn_grid, x="k", y="Accuracy", hue="Testanteil", marker="o")
plt.title("KNN – Accuracy über k für verschiedene Testanteile")
plt.xlabel("Anzahl der Nachbarn (k)")
plt.ylabel("Accuracy")
plt.ylim(knn_grid["Accuracy"].min() - 0.02, knn_grid["Accuracy"].max() + 0.02)
plt.grid(True)
plt.tight_layout()
plt.show()

# (Optional) Heatmap: k × Testanteil
plt.figure(figsize=(7, 4.5))
sns.heatmap(knn_pivot, annot=True, fmt=".3f", cmap="Blues")
plt.title("KNN – Accuracy (Heatmap) für k und Testanteil")
plt.xlabel("Testanteil"); plt.ylabel("k")
plt.tight_layout()
plt.show()

# === Bestes k je Testanteil bestimmen + Confusion-Matrix ausgeben ===
from sklearn.metrics import classification_report

for t in test_sizes:
    sub = knn_grid[knn_grid["Testanteil"] == t]
    best_row = sub.loc[sub["Accuracy"].idxmax()]
    best_k = int(best_row["k"])
    best_acc = float(best_row["Accuracy"])
    print(f"\n=== Testanteil {int(t*100)}% → bestes k = {best_k} (Accuracy {best_acc:.4f}) ===")

    x_train, x_test, y_train, y_test = train_test_split(
        X, y, test_size=t, random_state=42, stratify=y
    )
    best_knn = KNeighborsClassifier(n_neighbors=best_k).fit(x_train, y_train)
    y_pred = best_knn.predict(x_test)

    print("Classification Report:\n", classification_report(y_test, y_pred, digits=3))
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(4, 4))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)
    plt.title(f"KNN – Confusion Matrix (Test {int(t*100)}%, k={best_k})")
```

```
    plt.xlabel("Predicted"); plt.ylabel("Actual")
    plt.tight_layout()
    plt.show()

knn_grid.to_csv("knn_accuracy_grid_over_k_and_splits.csv", index=False)
print("Gespeichert: knn_accuracy_grid_over_k_and_splits.csv")

#%% md
<a id = "6"></a><br>

## **Decision Tree Classifaction**



#%%
# === Decision Tree: Einzellauf (30 % Test) ===
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

x_train, x_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

dt = DecisionTreeClassifier(random_state=42)
dt.fit(x_train, y_train)

print("Decision Tree Test accuracy:", dt.score(x_test, y_test))

#%%
# === Decision Tree: 10/30/50/70/90 % Testanteil (Tabelle + Plot) ===
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
import pandas as pd

test_sizes = [0.1, 0.3, 0.5, 0.7, 0.9]
dt_rows = []

for t in test_sizes:
    x_train, x_test, y_train, y_test = train_test_split(
        X, y, test_size=t, random_state=42, stratify=y
    )
    dt_tmp = DecisionTreeClassifier(random_state=42).fit(x_train, y_train)
    acc = dt_tmp.score(x_test, y_test)
    dt_rows.append({"Testanteil": t, "Trainanteil": 1 - t, "Accuracy": acc})
    print(f"Testanteil {int(t*100)}% – Accuracy: {acc:.4f}")

# Tabelle + Plot
_dt_df = pd.DataFrame(dt_rows)
display(_dt_df)

plt.figure(figsize=(7, 4))
sns.lineplot(data=_dt_df, x="Testanteil", y="Accuracy", marker="o")
plt.title("Decision Tree – Accuracy über verschiedene Testanteile")
plt.xlabel("Testanteil"); plt.ylabel("Accuracy"); plt.grid(True); plt.show()

#%%
# === Decision Tree: Confusion Matrices für alle Testanteile ===
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split

test_sizes = [0.1, 0.3, 0.5, 0.7, 0.9]

for split_p in test_sizes:
    print(f"\n=== Testanteil: {int(split_p * 100)}% ===")
    x_train, x_test, y_train, y_test = train_test_split(
        X, y, test_size=split_p, random_state=42, stratify=y
    )

    dt = DecisionTreeClassifier(random_state=42).fit(x_train, y_train)
    y_pred = dt.predict(x_test)

    acc = dt.score(x_test, y_test)
    print(f"Accuracy: {acc:.4f}")
    print("Classification Report:\n", classification_report(y_test, y_pred))

    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(4, 4))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)
    plt.title(f"Decision Tree – Confusion Matrix ({int(split_p * 100)}% Test)")
    plt.xlabel("Predicted"); plt.ylabel("Actual"); plt.tight_layout(); plt.show()
```

```
#%%
# === Decision Tree – Übersichtstabelle & Plot (Accuracy je Testanteil) ===
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
import pandas as pd
import numpy as np

test_sizes = [0.1, 0.3, 0.5, 0.7, 0.9]
rows = []

for t in test_sizes:
    x_train, x_test, y_train, y_test = train_test_split(
        X, y, test_size=t, random_state=42, stratify=y
    )
    dt = DecisionTreeClassifier(random_state=42).fit(x_train, y_train)
    acc = dt.score(x_test, y_test)
    rows.append({"Testanteil": t, "Trainanteil": 1 - t, "Accuracy": acc})
    print(f"Testanteil {int(t*100)}% – Accuracy: {acc:.4f}")

dt_acc = pd.DataFrame(rows)
display(dt_acc.style.format({"Testanteil": "{:.0%}", "Trainanteil": "{:.0%}", "Accuracy": "{:.3f}"}))

plt.figure(figsize=(7, 4))
sns.lineplot(data=dt_acc, x="Testanteil", y="Accuracy", marker="o")
plt.xticks(test_sizes, [f"{int(t*100)}%" for t in test_sizes])
plt.ylim(dt_acc["Accuracy"].min() - 0.02, dt_acc["Accuracy"].max() + 0.02)
plt.yticks(np.arange(round(dt_acc["Accuracy"].min(), 2),
                     round(dt_acc["Accuracy"].max() + 0.02, 2), 0.01))
plt.title("Decision Tree – Accuracy über verschiedene Testanteile")
plt.xlabel("Testanteil"); plt.ylabel("Accuracy"); plt.grid(True); plt.tight_layout(); plt.show()

dt_acc.to_csv("dt_accuracy_by_split.csv", index=False)
print("Gespeichert: dt_accuracy_by_split.csv")


#%% md
<a id = "7"></a><br>

## **Random Forest Classifaction**


#%%
# === Random Forest: Einzellauf (30 % Test) ===
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

x_train, x_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(x_train, y_train)

print("Random Forest Test accuracy:", rf.score(x_test, y_test))

#%%
# === Random Forest: 10/30/50/70/90 % Testanteil (Tabelle + Plot) ===
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

test_sizes = [0.1, 0.3, 0.5, 0.7, 0.9]
rf_rows = []

for t in test_sizes:
    x_train, x_test, y_train, y_test = train_test_split(
        X, y, test_size=t, random_state=42, stratify=y
    )
    rf_tmp = RandomForestClassifier(n_estimators=100, random_state=42).fit(x_train, y_train)
    acc = rf_tmp.score(x_test, y_test)
    rf_rows.append({"Testanteil": t, "Trainanteil": 1 - t, "Accuracy": acc})
    print(f"Testanteil {int(t*100)}% – Accuracy: {acc:.4f}")

# Tabelle + Plot
rf_df = pd.DataFrame(rf_rows)
display(rf_df.style.format({"Testanteil": "{:.0%}", "Trainanteil": "{:.0%}", "Accuracy": "{:.3f}"}))

plt.figure(figsize=(7, 4))
sns.lineplot(data=rf_df, x="Testanteil", y="Accuracy", marker="o", color="forestgreen")
plt.title("Random Forest – Accuracy über verschiedene Testanteile")
plt.xlabel("Testanteil")
```

```
plt.ylabel("Accuracy")
plt.ylim(rf_df["Accuracy"].min() - 0.02, rf_df["Accuracy"].max() + 0.02)
plt.grid(True)
plt.tight_layout()
plt.show()

rf_df.to_csv("rf_accuracy_by_split.csv", index=False)
print("Gespeichert: rf_accuracy_by_split.csv")

#%%
# === Random Forest: Confusion Matrices für alle Testanteile ===
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

test_sizes = [0.1, 0.3, 0.5, 0.7, 0.9]

for split_p in test_sizes:
    print(f"\n=== Testanteil: {int(split_p * 100)}% ===")
    x_train, x_test, y_train, y_test = train_test_split(
        X, y, test_size=split_p, random_state=42, stratify=y
    )

    rf = RandomForestClassifier(n_estimators=100, random_state=42).fit(x_train, y_train)
    y_pred = rf.predict(x_test)

    acc = rf.score(x_test, y_test)
    print(f"Accuracy: {acc:.4f}")
    print("Classification Report:\n", classification_report(y_test, y_pred))

    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(4, 4))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)
    plt.title(f"Random Forest – Confusion Matrix ({int(split_p * 100)}% Test)")
    plt.xlabel("Predicted"); plt.ylabel("Actual"); plt.tight_layout(); plt.show()

#%%
# === Random Forest – Übersichtstabelle & Plot (Accuracy je Testanteil) ===
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import pandas as pd
import numpy as np

test_sizes = [0.1, 0.3, 0.5, 0.7, 0.9]
rows = []

for t in test_sizes:
    x_train, x_test, y_train, y_test = train_test_split(
        X, y, test_size=t, random_state=42, stratify=y
    )
    rf = RandomForestClassifier(n_estimators=100, random_state=42).fit(x_train, y_train)
    acc = rf.score(x_test, y_test)
    rows.append({"Testanteil": t, "Trainanteil": 1 - t, "Accuracy": acc})
    print(f"Testanteil {int(t*100)}% – Accuracy: {acc:.4f}")

rf_acc = pd.DataFrame(rows)
display(rf_acc.style.format({"Testanteil": "{:.0%}", "Trainanteil": "{:.0%}", "Accuracy": "{:.3f}"}))

plt.figure(figsize=(7, 4))
sns.lineplot(data=rf_acc, x="Testanteil", y="Accuracy", marker="o")
plt.xticks(test_sizes, [f"{int(t*100)}%" for t in test_sizes])
plt.ylim(rf_acc["Accuracy"].min() - 0.02, rf_acc["Accuracy"].max() + 0.02)
plt.yticks(np.arange(round(rf_acc["Accuracy"].min(), 2),
                     round(rf_acc["Accuracy"].max() + 0.02, 2), 0.01))
plt.title("Random Forest – Accuracy über verschiedene Testanteile")
plt.xlabel("Testanteil"); plt.ylabel("Accuracy"); plt.grid(True); plt.tight_layout(); plt.show()

rf_acc.to_csv("rf_accuracy_by_split.csv", index=False)
print("Gespeichert: rf_accuracy_by_split.csv")

#%% md
# **3. Vergleich**

#%%
# === Abschließender Vergleich – alle Modelle über verschiedene Testanteile
#     (KNN nimmt je Split automatisch das beste k; inkl. Logistic Regression) ===
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
```

```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

test_sizes = [0.1, 0.3, 0.5, 0.7, 0.9]
# KNN: Kandidaten für k (ungerade vermeiden Ties)
k_list = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21]

# Alle Basismodelle (KNN separat, weil k-Sweep)
base_models = {
    "Logistic Regression": LogisticRegression(max_iter=1000, random_state=42),
    "Naive Bayes": GaussianNB(),
    "SVM": SVC(),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(n_estimators=100, random_state=42),
}

results = []
best_k_rows = []  # Übersicht: bestes k je Split

for t in test_sizes:
    # Split je Testanteil
    x_train, x_test, y_train, y_test = train_test_split(
        X, y, test_size=t, random_state=42, stratify=y
    )

    # 1) Basismodelle trainieren & messen
    for name, model in base_models.items():
        model.fit(x_train, y_train)
        acc = model.score(x_test, y_test)
        results.append({
            "Modell": name,
            "Testanteil": t,
            "Trainanteil": 1 - t,
            "Accuracy": acc
        })
        print(f"{name:19} | Test {int(t*100)}% | Acc: {acc:.4f}")

    # 2) KNN: bestes k für diesen Split suchen
    best_k, best_acc = None, -1
    for k in k_list:
        knn = KNeighborsClassifier(n_neighbors=k).fit(x_train, y_train)
        acc = knn.score(x_test, y_test)
        if acc > best_acc:
            best_acc, best_k = acc, k

    # KNN (best k) aufnehmen
    results.append({
        "Modell": f"KNN (best k={best_k})",
        "Testanteil": t,
        "Trainanteil": 1 - t,
        "Accuracy": best_acc
    })
    best_k_rows.append({"Testanteil": t, "Bestes k": best_k, "Accuracy": best_acc})
    print(f"KNN (best k={best_k:>2})      | Test {int(t*100)}% | Acc: {best_acc:.4f}")

# === Ergebnisse: Tabellen ===
results_df = pd.DataFrame(results)
best_k_df = pd.DataFrame(best_k_rows)

display(results_df.style.format({
    "Testanteil": "{:.0%}",
    "Trainanteil": "{:.0%}",
    "Accuracy": "{:.3f}"
}))

print("\nBestes k je Testanteil (KNN):")
display(best_k_df.style.format({
    "Testanteil": "{:.0%}",
    "Accuracy": "{:.3f}"
}))

# === Plot – Accuracy pro Modell und Testanteil (KNN mit best k) ===
plt.figure(figsize=(9, 5))
sns.lineplot(data=results_df, x="Testanteil", y="Accuracy", hue="Modell", marker="o")

ax = plt.gca()
ax.ticklabel_format(style="plain", axis="y")
ax.yaxis.set_major_formatter(plt.FormatStrFormatter("%.3f"))

xticks = sorted(results_df["Testanteil"].unique())
plt.xticks(xticks, [f"{int(t*100)}%" for t in xticks])
```

```
# dynamische, „gezoomte“ Y-Achse
ymin = results_df["Accuracy"].min()
ymax = results_df["Accuracy"].max()
plt.ylim(ymin - 0.02, ymax + 0.02)
plt.yticks(np.arange(round(ymin, 2), round(ymax + 0.02, 2), 0.01))

plt.title("Vergleich der Modelle über verschiedene Testanteile\n(KNN je Split mit best k)")
plt.xlabel("Testanteil")
plt.ylabel("Accuracy")
plt.grid(True)
plt.tight_layout()
plt.show()

# === Optional: speichern für Bericht ===
results_df.to_csv("model_accuracy_over_splits_knn_bestk.csv", index=False)
best_k_df.to_csv("knn_bestk_per_split.csv", index=False)
print("Gespeichert: model_accuracy_over_splits_knn_bestk.csv, knn_bestk_per_split.csv")

#%% md
```