Gerne, um die Schritte der Hauptkomponentenanalyse (PCA) auf einem zweidimensionalen Datensatz zu veranschaulichen, werden wir die ersten beiden Merkmale des Iris-Datensatzes – Kelchblattlänge (sepal length) und Kelchblattbreite (sepal width) – auswählen und die Transformationen von den Originaldaten bis zur Darstellung im Hauptkomponenten-Koordinatensystem Schritt für Schritt visualisieren.

Hier ist die Implementierung der geforderten Schritte in Python, begleitet von Erklärungen, die auf den bereitgestellten Quellen basieren:

```
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
import seaborn as sns
import pandas as pd
import numpy as np

# Iris-Datensatz laden
iris = load_iris(as_frame=True)
iris.frame["target"] = iris.target_names[iris.target]

# Für diese Demonstration wählen wir nur die ersten beiden Merkmale des Iris-Datensatzes:
# Kelchblattlänge (sepal length) und Kelchblattbreite (sepal width).
# Die Originaldaten des Iris-Datensatzes bestehen aus 4 Merkmalen.
# Um die Funktionsweise der PCA Schritt für Schritt in 2D zu zeigen,
# verwenden wir hier explizit nur 2 dieser Originalmerkmale.
X_2d_original = iris.data[['sepal length (cm)', 'sepal width (cm)']]
y_target = iris.target
target_names = iris.target_names

print(f"Gewählte Merkmale für die PCA: {X_2d_original.columns.tolist()}")

# --- 1. Plotte diese zweidimensionalen Daten zunächst in einem klassischen x-y Koordinatensy
plt.figure(figsize=(7,7))
sns.scatterplot(
    x=X_2d_original['sepal length (cm)'],
    y=X_2d_original['sepal width (cm)'],
    hue=iris.frame['target'], # Farben nach Iris-Typ
    palette='viridis',
    s=70,
    alpha=0.8
)
plt.title("1. Originale Iris-Daten (Kelchblattlänge vs. Kelchblattbreite)")
```

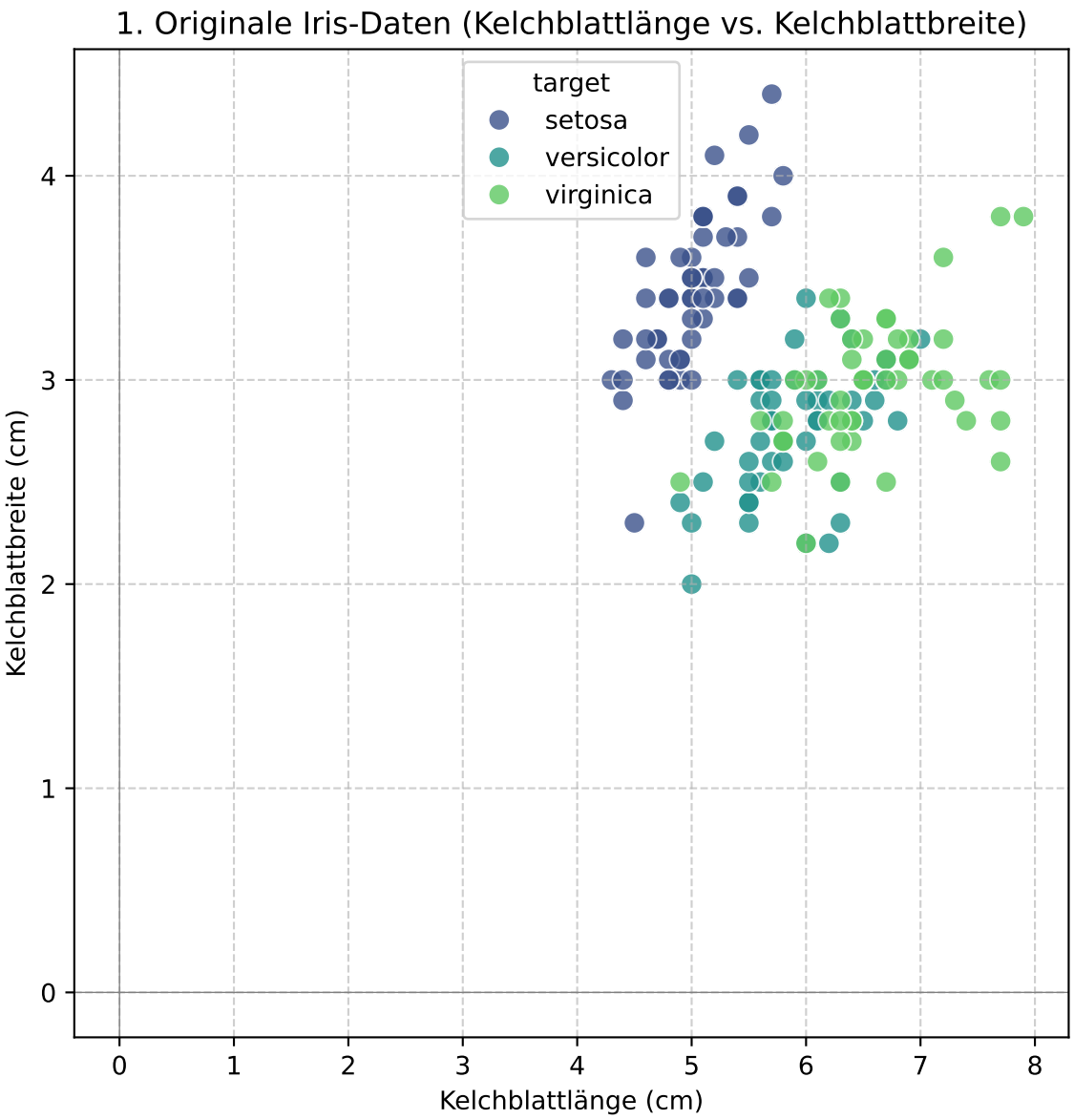
```
plt.xlabel("Kelchblattlänge (cm)")
plt.ylabel("Kelchblattbreite (cm)")
plt.grid(True, linestyle='--', alpha=0.6)
plt.axhline(0, color='gray', linewidth=0.5) # Ursprungslinien zur Orientierung
plt.axvline(0, color='gray', linewidth=0.5)
plt.show()

# --- 2. Skaliere die Daten und zeichne diese ebenfalls. ---
# --- 3. Verschiebe die Daten in den Ursprung und zeichne die Daten. ---
# Bevor PCA durchgeführt wird, ist es entscheidend, die Daten zu skalieren und zu zentrieren
# Ungleich skalierte Variablen können die PCA verzerren.
# Der StandardScaler zentriert die Daten (d.h., der Mittelwert jeder Spalte wird 0)
# und skaliert sie (d.h., die Standardabweichung jeder Spalte wird 1).
# Das Zentrieren bedeutet effektiv, dass der Mittelpunkt der Daten zum Ursprung (0,0) versch
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_2d_original)

# Konvertiere skalierte Daten zurück in einen DataFrame für einfachere Beschriftung und Dars
X_scaled_df = pd.DataFrame(X_scaled, columns=X_2d_original.columns)

# After scaling the data, determine the axis limits
x_min = X_scaled_df['sepal length (cm)'].min() * 1.1
x_max = X_scaled_df['sepal length (cm)'].max() * 1.1
y_min = X_scaled_df['sepal width (cm)'].min() * 1.1
y_max = X_scaled_df['sepal width (cm)'].max() * 1.1

# Function to set consistent axis limits and styling
def set_consistent_axes(ax):
    ax.set_xlim(x_min, x_max)
    ax.set_ylim(y_min, y_max)
    ax.axhline(0, color='gray', linewidth=0.8, linestyle='-')
    ax.axvline(0, color='gray', linewidth=0.8, linestyle='-')
    ax.grid(True, linestyle='--', alpha=0.6)

# Plot 2/3: Scaled and centered data
plt.figure(figsize=(7,7))
ax = plt.gca()  # get current axes
sns.scatterplot(
    x=X_scaled_df['sepal length (cm)'],
    y=X_scaled_df['sepal width (cm)'],
    hue=iris.frame['target'],
    palette='viridis',
```

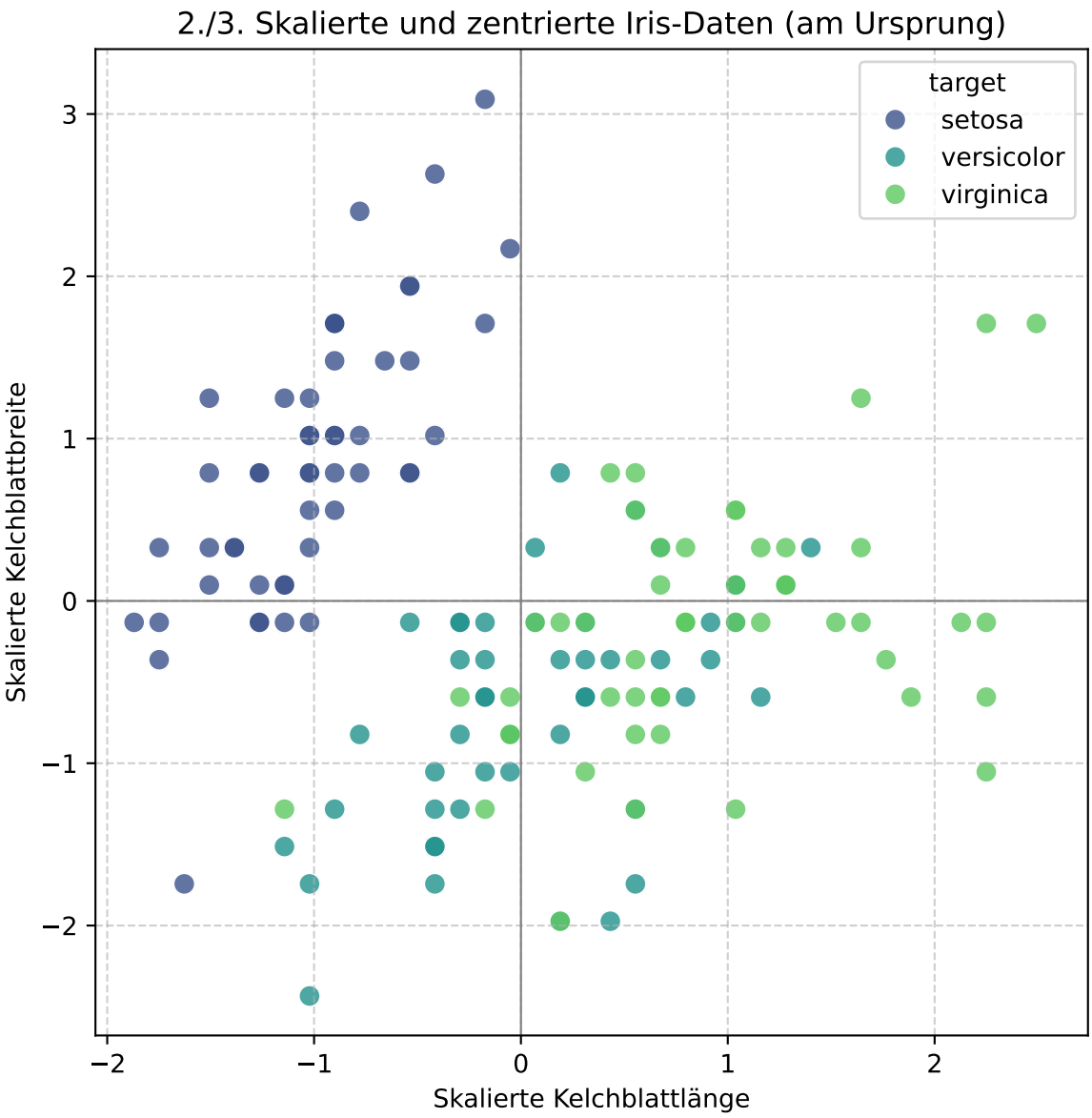
```
    s=70,
    alpha=0.8
)
plt.title("2./3. Skalierte und zentrierte Iris-Daten (am Ursprung)")
plt.xlabel("Skalierte Kelchblattlänge")
plt.ylabel("Skalierte Kelchblattbreite")
set_consistent_axes(ax)
plt.show()

# --- 4. Bestimme die erste Hauptkomponente und zeichne diese. ---
pca_1_component = PCA(n_components=1)
pca_1_component.fit(X_scaled)

# Get the direction vector of PC1
pc1_vector = pca_1_component.components_[0]  # Extract first component vector

# Create line coordinates for plotting
scale_factor = np.max(np.abs(X_scaled)) * 1.5
line_x = np.array([-scale_factor * pc1_vector[0], scale_factor * pc1_vector[0]])
line_y = np.array([-scale_factor * pc1_vector[1], scale_factor * pc1_vector[1]])

# Plot the scaled data and PC1
# Plot 4: With PC1
plt.figure(figsize=(7,7))
ax = plt.gca()
sns.scatterplot(
    x=X_scaled_df['sepal length (cm)'],
    y=X_scaled_df['sepal width (cm)'],
    hue=iris.frame['target'],
    palette='viridis',
    s=70,
    alpha=0.8
)
plt.plot(line_x, line_y, color='red', linestyle='-', linewidth=2, label='Hauptkomponente 1 (
plt.title("4. Skalierte Daten mit der ersten Hauptkomponente (PC1)")
plt.xlabel("Skalierte Kelchblattlänge")
plt.ylabel("Skalierte Kelchblattbreite")
set_consistent_axes(ax)
plt.legend()
plt.show()

# --- 5. Visualisiere den weiteren Verlauf der PCA, so dass am Ende die transformierten Date
```

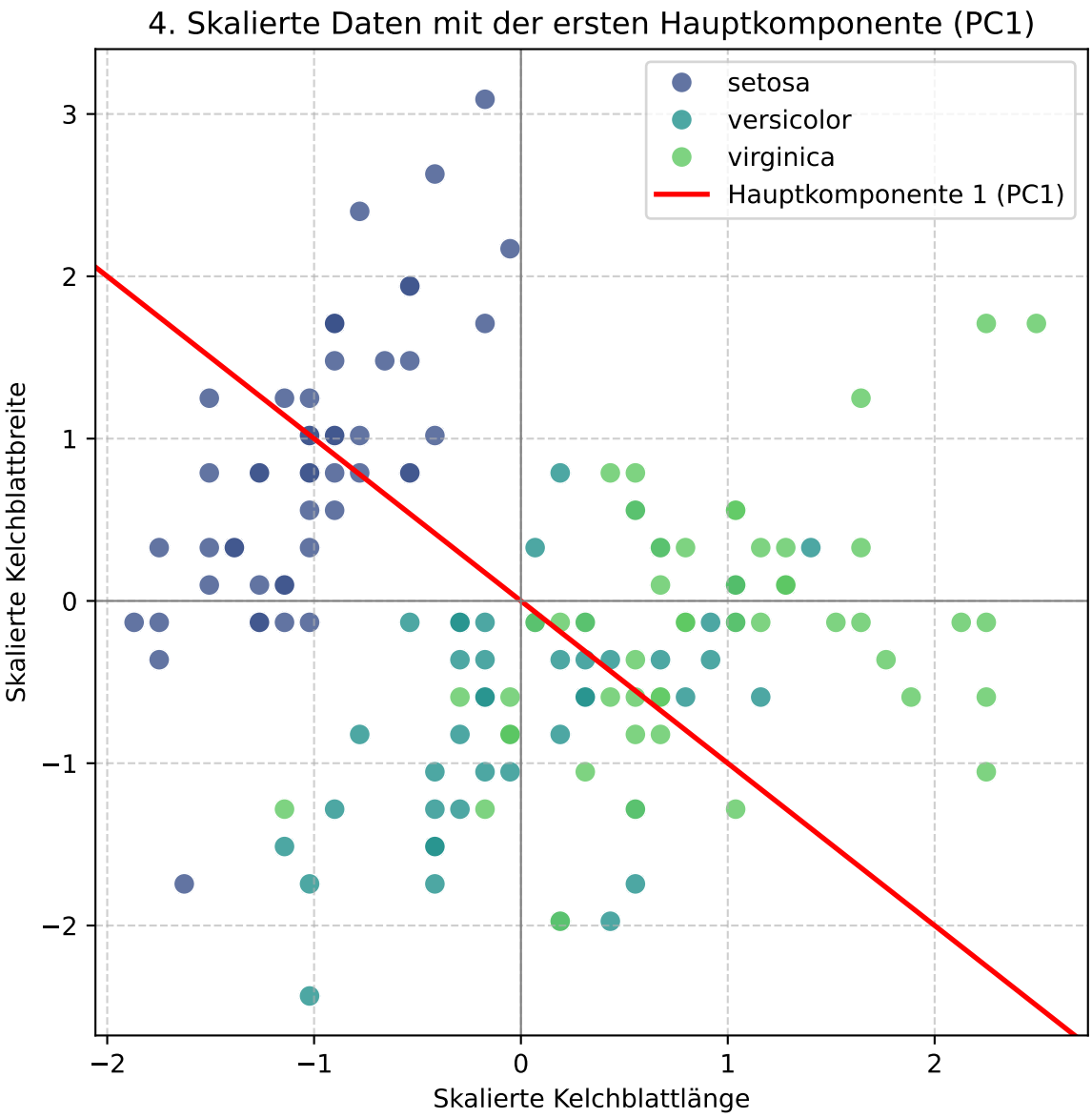
```
# Da es sich um zweidimensionale Daten handelt, ist die zweite Hauptkomponente (PC2)
# einfach die Linie durch den Ursprung, die senkrecht zu PC1 steht.
# Die transformierten Daten sind die Projektionen der Originaldaten auf diese neuen Achsen.
# Die endgültige PCA-Grafik wird erstellt, indem alles so gedreht wird,
# dass PC1 horizontal (als neue X-Achse) und PC2 vertikal (als neue Y-Achse) liegt.
pca_2_components = PCA(n_components=2)
X_pca_2d = pca_2_components.fit_transform(X_scaled) # Transformiert die Daten in das PC-Koord

# Optional: Visualisierung von PC1 und PC2 auf den skalierten Daten
# Optional: Visualisierung von PC1 und PC2 auf den skalierten Daten
pc1_vector = pca_2_components.components_[0]  # First principal component
pc2_vector = pca_2_components.components_[1]  # Second principal component

# Scale factor for visualization
scale_factor = np.max(np.abs(X_scaled)) * 1.5

# Create line coordinates for PC1
line1_x = np.array([-scale_factor * pc1_vector[0], scale_factor * pc1_vector[0]])
line1_y = np.array([-scale_factor * pc1_vector[1], scale_factor * pc1_vector[1]])

# Create line coordinates for PC2
line2_x = np.array([-scale_factor * pc2_vector[0], scale_factor * pc2_vector[0]])
line2_y = np.array([-scale_factor * pc2_vector[1], scale_factor * pc2_vector[1]])

# Plot the scaled data with both principal components
# Plot 5a: With PC1 and PC2
plt.figure(figsize=(7,7))
ax = plt.gca()
sns.scatterplot(
    x=X_scaled_df['sepal length (cm)'],
    y=X_scaled_df['sepal width (cm)'],
    hue=iris.frame['target'],
    palette='viridis',
    s=70,
    alpha=0.8
)
plt.plot(line1_x, line1_y, color='red', linestyle='-', linewidth=2, label='Hauptkomponente 1
plt.plot(line2_x, line2_y, color='blue', linestyle='--', linewidth=2, label='Hauptkomponente
plt.title("Skalierte Daten mit PC1 und PC2 (vor der Drehung)")
plt.xlabel("Skalierte Kelchblattlänge")
plt.ylabel("Skalierte Kelchblattbreite")
set_consistent_axes(ax)
```

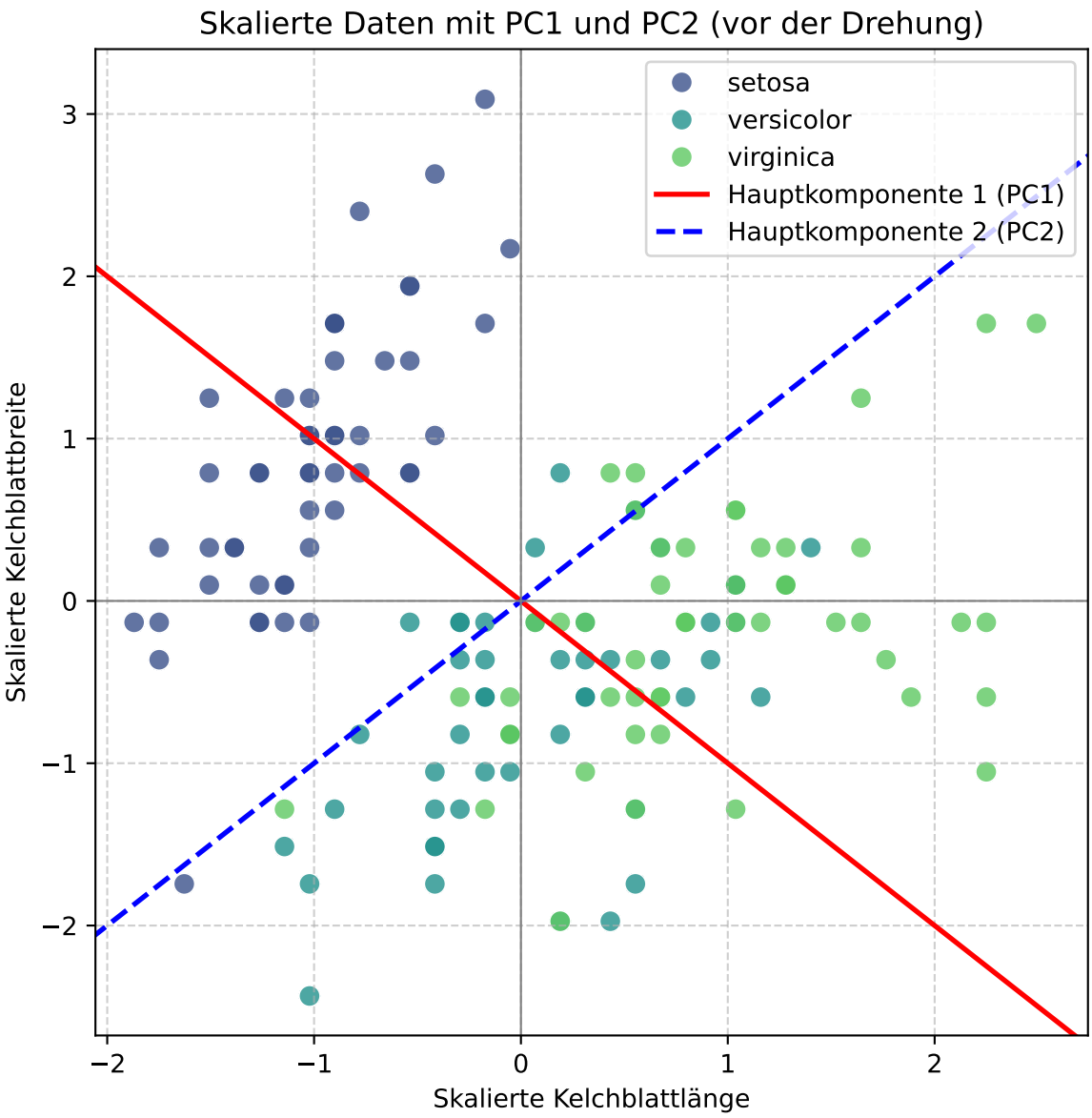
```
plt.legend()
plt.show()

# Plot 5b: Transformed data
# # Plot der transformierten Daten im neuen Koordinatensystem
plt.figure(figsize=(7,7))
ax = plt.gca()
sns.scatterplot(
    x=X_pca_2d[:, 0],
    y=X_pca_2d[:, 1],
    hue=iris.frame['target'],
    palette='viridis',
    s=70,
    alpha=0.8
)
plt.title("Transformierte Iris-Daten im PC1-PC2-Koordinatensystem")
plt.xlabel(f"Hauptkomponente 1 (PC1) - {pca_2_components.explained_variance_ratio_[0]:.1%} Va
plt.ylabel(f"Hauptkomponente 2 (PC2) - {pca_2_components.explained_variance_ratio_[1]:.1%} Va
# Gitter hinzufügen
plt.grid(True, linestyle='--', alpha=0.6)
# Koordinatenachsen hinzufügen
plt.axhline(0, color='red', linewidth=2, linestyle='-')
plt.axvline(0, color='blue', linewidth=2, linestyle='--')
set_consistent_axes(ax)
plt.show()
```

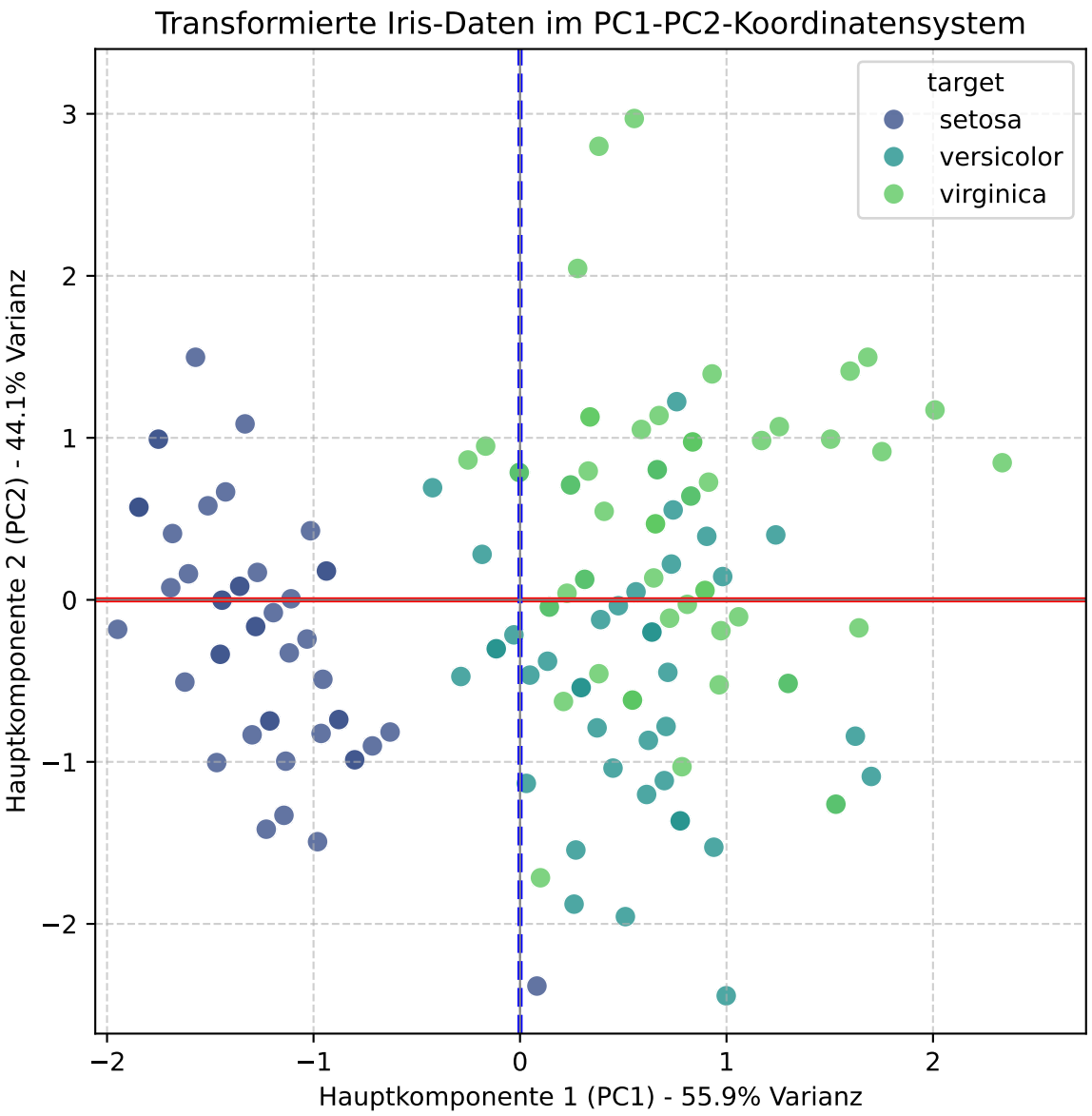
```
Gewählte Merkmale für die PCA: ['sepal length (cm)', 'sepal width (cm)']
```


1. Originale Iris-Daten (Kelchblattlänge vs. Kelchblattbreite)
target
setosa
versicolor
virginica
Kelchblattbreite (cm)
Kelchblattlänge (cm)
0
1
2
3
4
5
6
7
8


2./3. Skalierte und zentrierte Iris-Daten (am Ursprung)
target
setosa
versicolor
virginica
Skalierte Kelchblattbreite
Skalierte Kelchblattlänge
3
2
1
0
−1
−2
−2
−1
0
1
2


4. Skalierte Daten mit der ersten Hauptkomponente (PC1)
setosa
versicolor
virginica
Hauptkomponente 1 (PC1)
Skalierte Kelchblattbreite
Skalierte Kelchblattlänge
3
2
1
0
−1
−2
−2
−1
0
1
2


Skalierte Daten mit PC1 und PC2 (vor der Drehung)
setosa
versicolor
virginica
Hauptkomponente 1 (PC1)
Hauptkomponente 2 (PC2)
Skalierte Kelchblattbreite
Skalierte Kelchblattlänge
3
2
1
0
−1
−2
−2
−1
0
1
2


Transformierte Iris-Daten im PC1-PC2-Koordinatensystem
target
setosa
versicolor
virginica
Hauptkomponente 2 (PC2) - 44.1% Varianz
Hauptkomponente 1 (PC1) - 55.9% Varianz
3
2
1
0
−1
−2
−2
−1
0
1
2