The Iris Dataset: PCA vs t-SNE

t-distributed Stochastic Neighbor Embedding (t-SNE) is a powerful technique for dimensionality reduction widely regarded for its ability to preserve local structures in high-dimensional data. Unlike linear methods such as Principal Component Analysis (PCA), t-SNE excels at capturing complex relationships and clustering tendencies, making it particularly effective for visualising high-dimensional datasets. By transforming data into a lower-dimensional space while maintaining meaningful patterns, t-SNE reveals insights that are often obscured in higher dimensions.

1. Importing necessary libraries

First, we need to import the libraries required for our analysis: `pandas` and `numpy` for data manipulation, `sklearn` for loading the dataset and applying Principal Component Analysis (PCA) and t-SNE on it, `matplotlib` and `seaborn` for data visualisation, and `time` for measuring execution time.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
import time
```

2. Loading the dataset

Here, we load the Iris dataset, which contains features of three different species of Iris flowers. The features are stored in `X`, while the target labels are in `y`.

```
iris = load_iris()
X = iris.data
y = iris.target
target_names = iris.target_names
```

We also create a DataFrame for easier data manipulation and plotting, and display its first few rows for verification.

```
df_iris = pd.DataFrame(X, columns=iris.feature_names)
df_iris['target'] = y

# Displaying the first few rows of the dataset
print(df_iris.head())
```

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

   target
0       0
1       0
2       0
3       0
4       0
```

3. Applying PCA and t-SNE for dimensionality reduction

Now, we apply PCA to reduce the dataset to two dimensions, with timing measurements added to evaluate performance.

```
start_time_pca = time.time()    # Starting timer for PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
end_time_pca = time.time()    # Ending timer for PCA
pca_time = end_time_pca - start_time_pca    # Calculating time taken for PCA
```

Similarly, we apply t-SNE, and record its computation time.

```
start_time_tsne = time.time()    # Starting timer for t-SNE
tsne = TSNE(n_components=2, random_state=0)
X_tsne = tsne.fit_transform(X)
end_time_tsne = time.time()    # Ending timer for t-SNE
tsne_time = end_time_tsne - start_time_tsne    # Calculating time taken for t-SNE
```

4. Visualising the results

In this section, we define a function to creaate scatterplots of the dimensionality-reduced data for PCA and t-SNE, allowing for direct comparison.

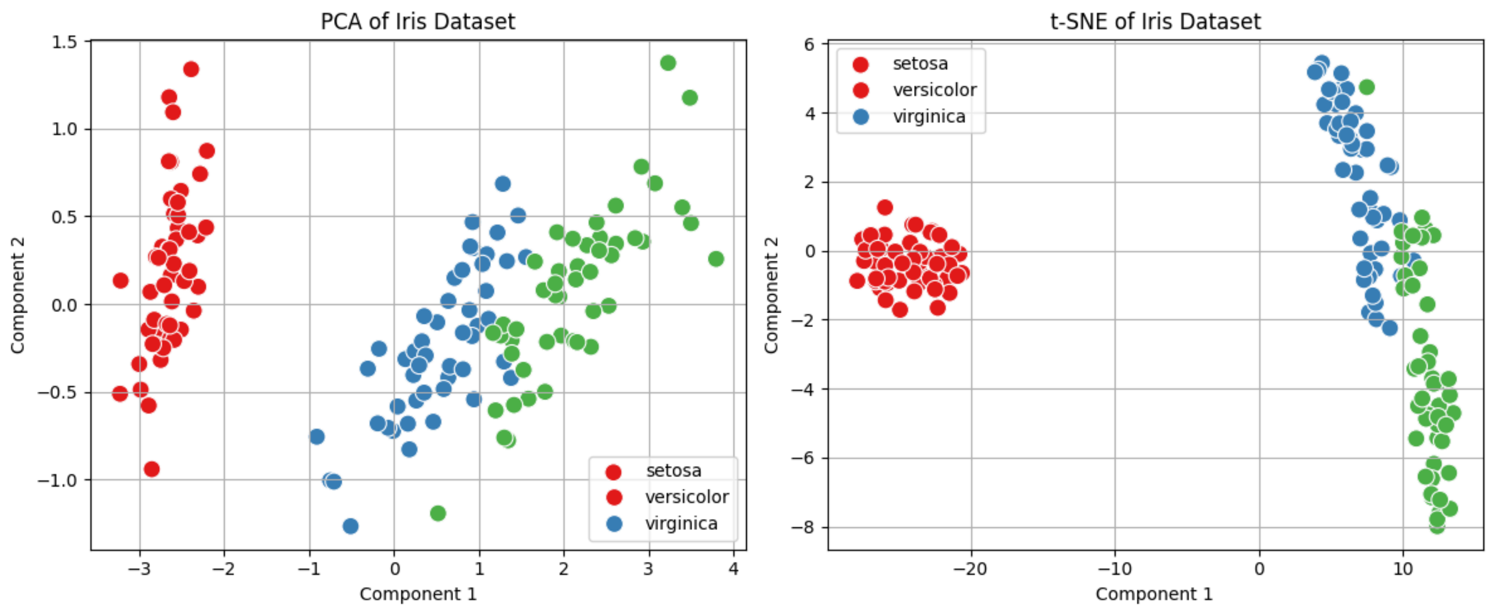
```
def plot_reduction(X1, X2, y, title1, title2):
    plt.figure(figsize=(12,5))

    plt.subplot(1, 2, 1)
    sns.scatterplot(x=X1[:, 0], y=X1[:, 1], hue=y, palette='Set1', s=100)
    plt.title(title1)
    plt.xlabel('Component 1')
    plt.ylabel('Component 2')
    plt.legend(target_names)
    plt.grid()

    plt.subplot(1, 2, 2)
    sns.scatterplot(x=X2[:, 0], y=X2[:, 1], hue=y, palette='Set1', s=100)
    plt.title(title2)
    plt.xlabel('Component 1')
    plt.ylabel('Component 2')
    plt.legend(target_names)
    plt.grid()

    plt.tight_layout()
    plt.show()

plot_reduction(X_pca, X_tsne, y, 'PCA of Iris Dataset', 't-SNE of Iris Dataset')
```



5. Comparing the performance of PCA and t-SNE

Finally, we print the explained variance ratios for PCA components. We also create a DataFrame to summarise the performance comparison based on computation time, and print it.

```
# Explained variance ratio
explained_variance = pca.explained_variance_ratio_
print("PCA Explained Variance Ratio:", explained_variance)

# Performance comparison
performance_comparison = pd.DataFrame({
    'Method': ['PCA', 't-SNE'],
    'Time (seconds)': [pca_time, tsne_time]
})
print("\nPerformance Comparison:")
print(performance_comparison)
```

```
PCA Explained Variance Ratio: [0.92461872 0.05306648]

Performance Comparison:
  Method  Time (seconds)
0    PCA        0.003218
1  t-SNE        0.789717
```