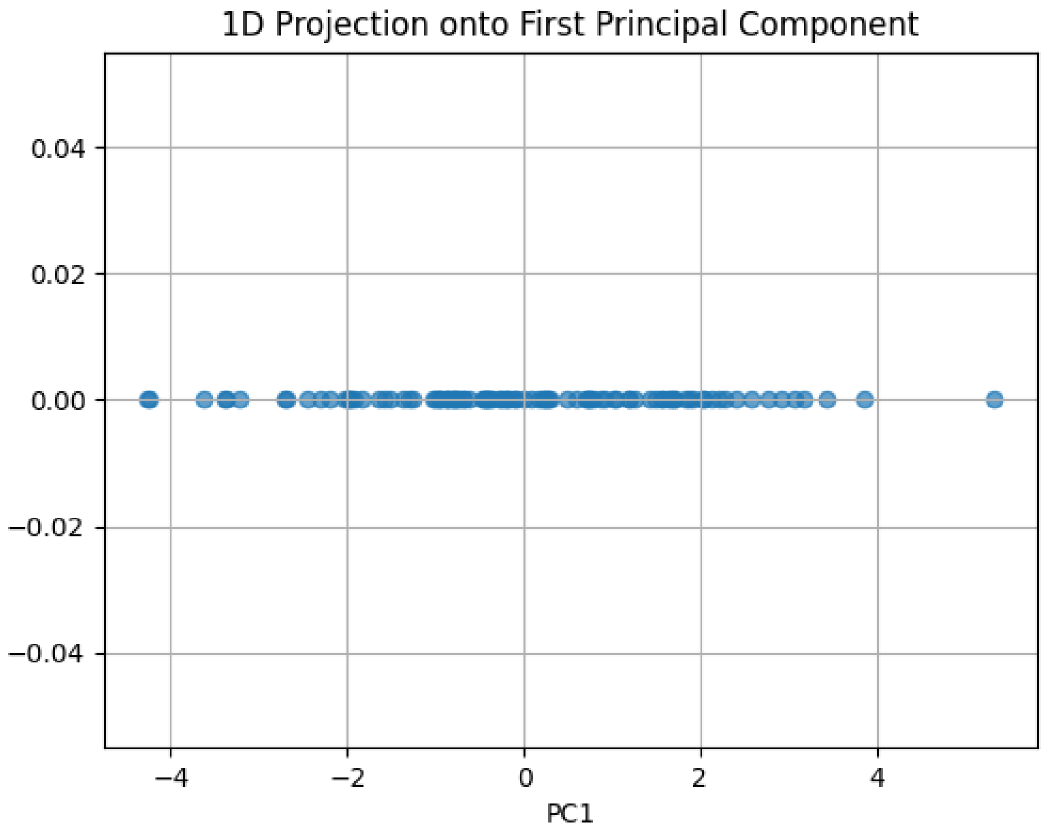
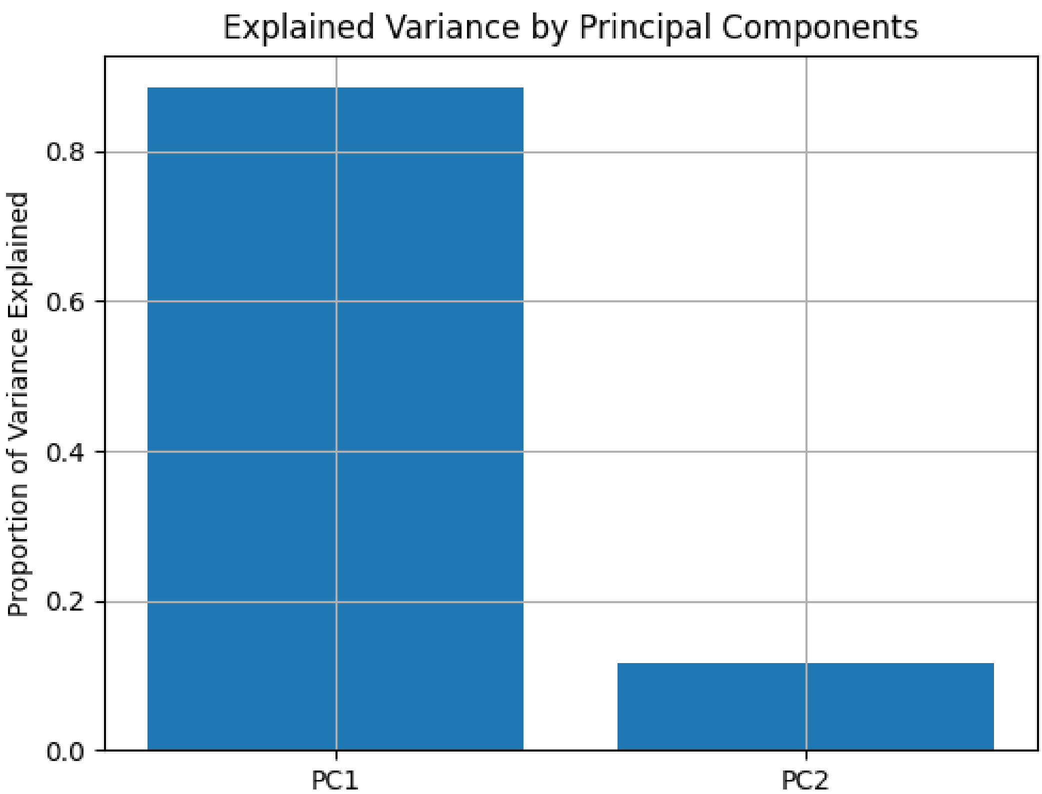
Part 1:

```python
import numpy as np

import matplotlib.pyplot as plt


# Step 1: Generate a 2D dataset with specified mean and covariance

np.random.seed(42)

X = np.random.multivariate_normal(mean=[2, 5], cov=[[3, 2], [2, 2]], size=100)


# Step 2: Plot the original 2D data

plt.scatter(X[:, 0], X[:, 1], alpha=0.7)

plt.title("Original 2D Data")

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

plt.axis("equal")

plt.grid(True)

plt.savefig("pca_original_data.png")

plt.show()


# Step 3: Center the data (subtract mean from each feature)

X_centered = X - np.mean(X, axis=0)


# Step 4: Compute the covariance matrix

cov_matrix = np.cov(X_centered.T)

print("Covariance Matrix:\n", cov_matrix)


# Step 5: Compute eigenvalues and eigenvectors of the covariance matrix

eig_vals, eig_vecs = np.linalg.eig(cov_matrix)

print("Eigenvalues:", eig_vals)
```

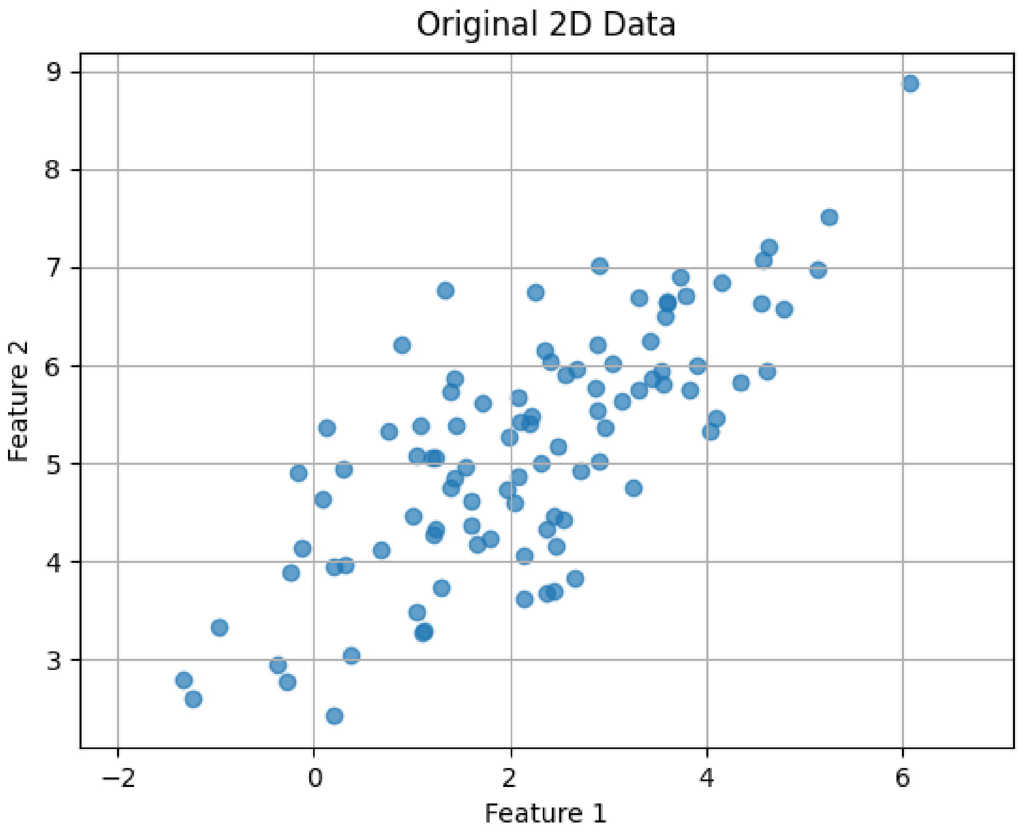
```
print("Eigenvectors:\n", eig_vecs)

# Step 6: Project the data onto the first principal component (PC1)
pc1 = eig_vecs[:, 0]
X_pca_1d = X_centered @ pc1  # Matrix multiplication to get 1D projection

# Step 7: Plot the 1D projection
plt.scatter(X_pca_1d, np.zeros_like(X_pca_1d), alpha=0.7)
plt.title("1D Projection onto First Principal Component")
plt.xlabel("PC1")
plt.grid(True)
plt.savefig("pca_1d_projection.png")
plt.show()

# Step 8: Plot the variance explained by each component
explained_variance = eig_vals / np.sum(eig_vals)
plt.bar(["PC1", "PC2"], explained_variance)
plt.title("Explained Variance by Principal Components")
plt.ylabel("Proportion of Variance Explained")
plt.grid(True)
plt.savefig("pca_explained_variance.png")
plt.show()
```



1D Projection onto First Principal Component
0.04
0.02
0.00
−0.02
−0.04
−4
−2
0
2
4
PC1



Explained Variance by Principal Components
Proportion of Variance Explained
0.8
0.6
0.4
0.2
0.0
PC1
PC2

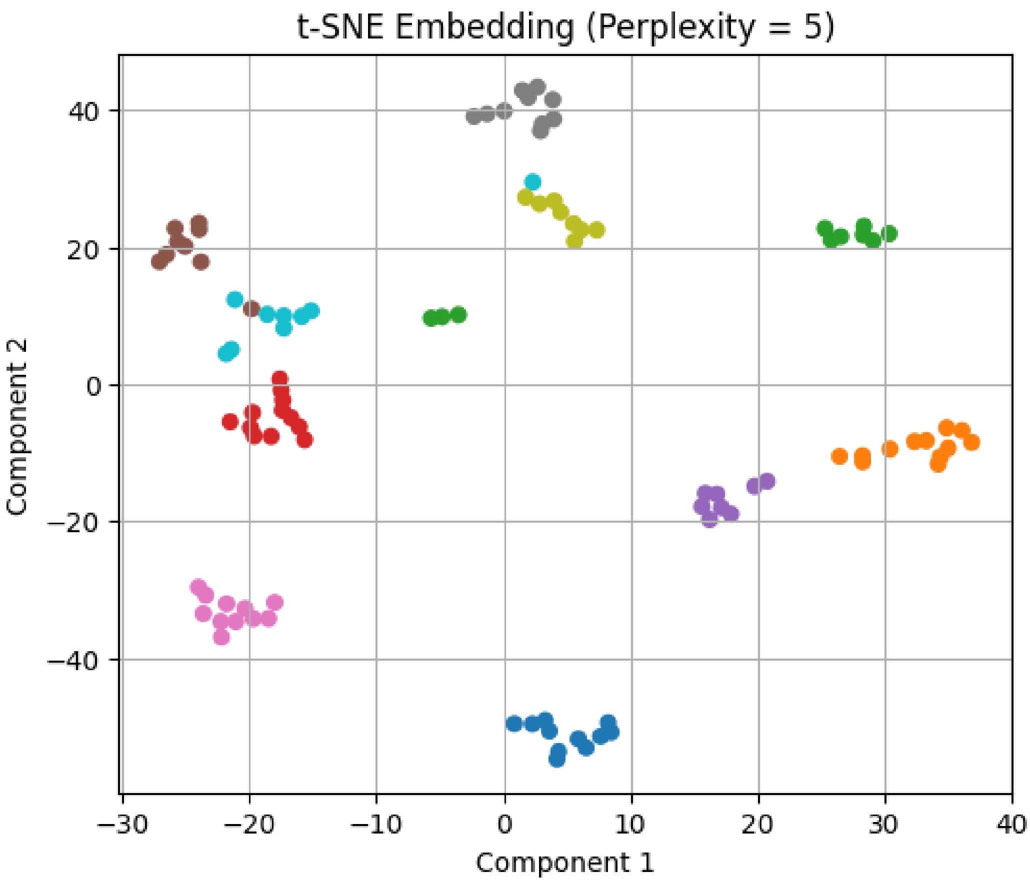


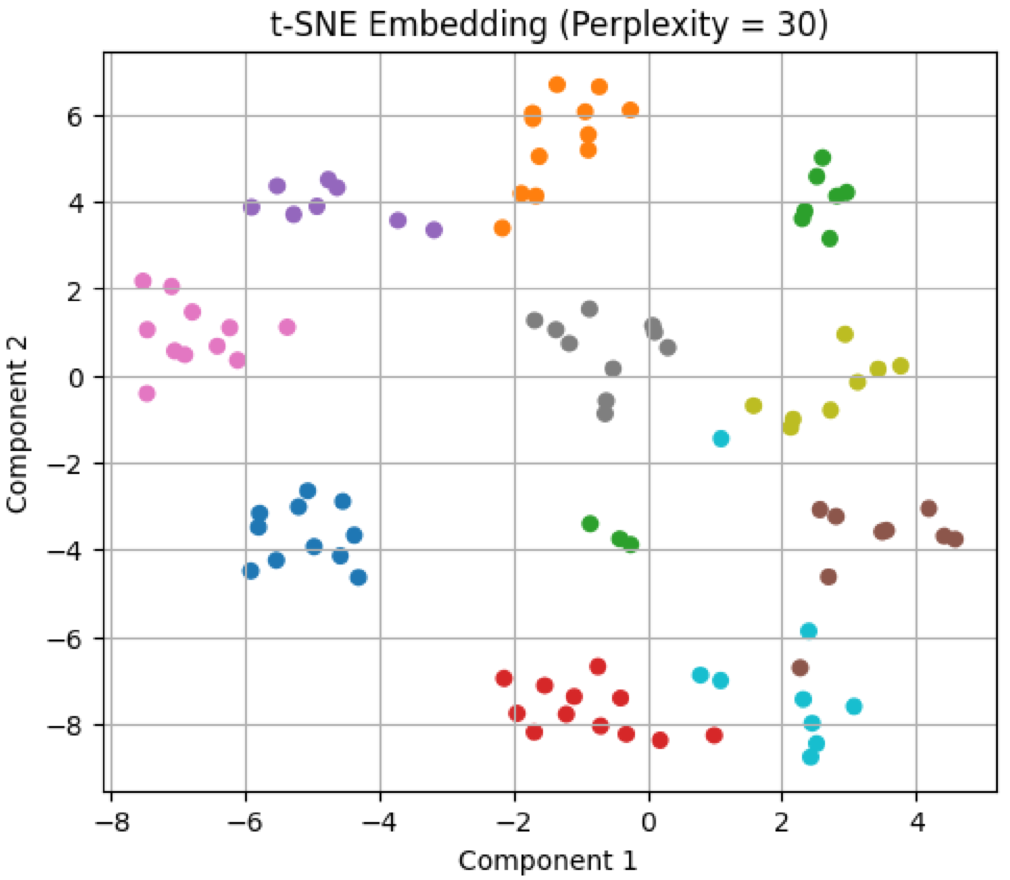
Original 2D Data
Feature 2
Feature 1
9
8
7
6
5
4
3
−2
0
2
4
6

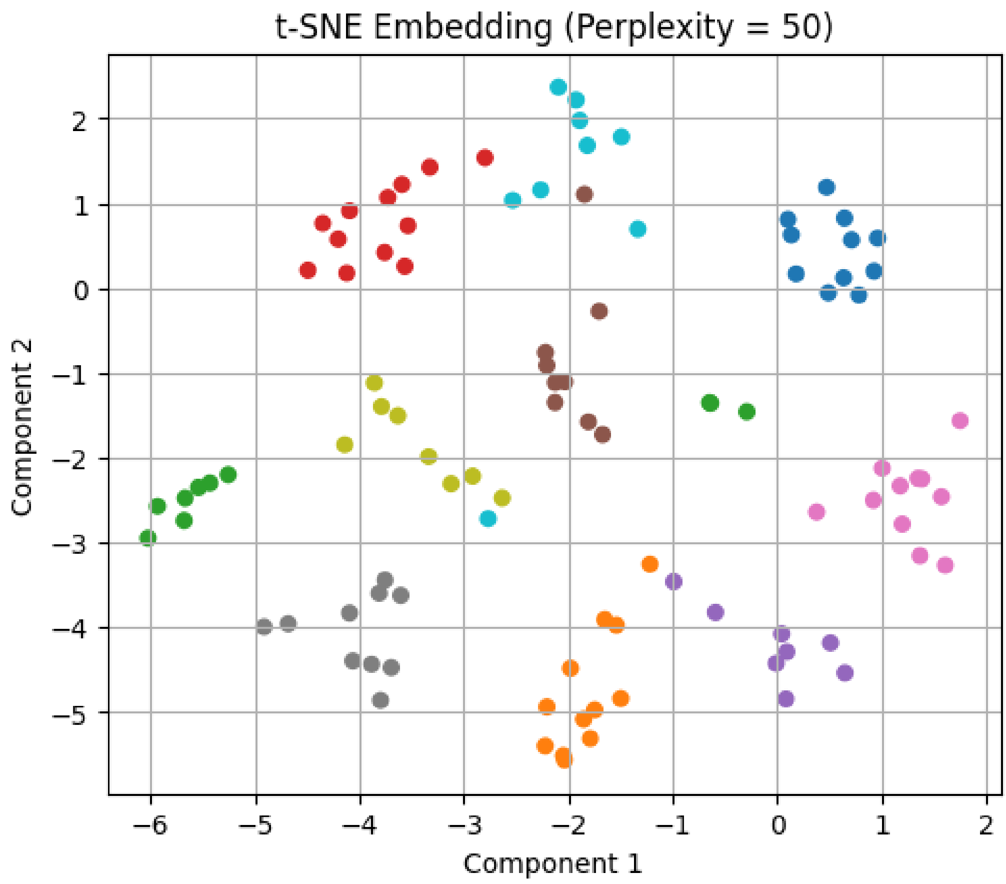
Part 2:

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
from sklearn.manifold import TSNE

# Step 1: Load the digits dataset
digits = load_digits()
X = digits.data[:100]   # Use only 100 samples for faster t-SNE execution
y = digits.target[:100]

print("Shape of X:", X.shape)  # (100, 64)
print("Each sample has 64 features representing an 8x8 image of a handwritten digit.")

# Step 2: Run t-SNE with different perplexity values
perplexities = [5, 30, 50]

for perplexity in perplexities:
    print(f"Running t-SNE with perplexity = {perplexity}")

    tsne = TSNE(n_components=2, perplexity=perplexity, random_state=42)
    X_embedded = tsne.fit_transform(X)

    # Step 3: Visualize the 2D t-SNE embedding
    plt.figure(figsize=(6, 5))
    scatter = plt.scatter(X_embedded[:, 0], X_embedded[:, 1], c=y, cmap='tab10', s=30)
    plt.title(f"t-SNE Embedding (Perplexity = {perplexity})")
    plt.xlabel("Component 1")
    plt.ylabel("Component 2")
```

```
plt.grid(True)


# Step 4: Save each figure
plt.savefig(f"tsne_digits_perplexity_{perplexity}.png")
plt.show()
```


t-SNE Embedding (Perplexity = 5)
Component 2
Component 1
40
20
0
−20
−40
−30
−20
−10
0
10
20
30
40


t-SNE Embedding (Perplexity = 30)
Component 2
Component 1
6
4
2
0
−2
−4
−6
−8
−8
−6
−4
−2
0
2
4



t-SNE Embedding (Perplexity = 50)
Component 1
Component 2
−6
−5
−4
−3
−2
−1
0
1
2
−5
−4
−3
−2
−1
0
1
2