**Prac these with your own data**

Clustering

# Generate synthetic 2D data for clustering

def generate\_data():

    cluster\_1 = np.random.randn(100, 2) + np.array([2, 2])

    cluster\_2 = np.random.randn(100, 2) + np.array([-2, -2])

    data = np.vstack([cluster\_1, cluster\_2])

    np.random.shuffle(data)

    return data

# Neural Network Autoencoder for clustering

class Autoencoder(nn.Module):

    def \_\_init\_\_(self):

        super(Autoencoder, self).\_\_init\_\_()

        self.encoder = nn.Sequential(

            nn.Linear(2, 4),

            nn.ReLU(),

            nn.Linear(4, 2)

        )

        self.decoder = nn.Sequential(

            nn.Linear(2, 4),

            nn.ReLU(),

            nn.Linear(4, 2)

        )

    def forward(self, x):

        encoded = self.encoder(x)

        decoded = self.decoder(encoded)

        return encoded, decoded

# Perform simple clustering based on encoded values

cluster\_labels = (encoded\_data[:, 0] > 0).astype(int)

# Generate 2D data

np.random.seed(0)

X = np.random.randn(100, 2)

# Compute the principal direction

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

X\_centered = X - mean

u, s, vh = np.linalg.svd(X\_centered)

principal\_axis = vh[0]

# Project data onto the principal axis (1D reduction)

X\_projected = X\_centered @ principal\_axis[:, np.newaxis] @ principal\_axis[np.newaxis, :]

# Generate 2D data with an obvious principal component

np.random.seed(42)

x = np.linspace(0, 10, 100)

y = 2 \* x + np.random.normal(0, 2, size=x.shape)  # Clear linear relationship with some noise

X = np.column\_stack((x, y))

# Compute the principal direction

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

X\_centered = X - mean

u, s, vh = np.linalg.svd(X\_centered)

principal\_axis = vh[0]

# Project data onto the principal axis (1D reduction)

X\_projected = X\_centered @ principal\_axis[:, np.newaxis] @ principal\_axis[np.newaxis, :]

# Generate random 3D data

np.random.seed(1)

X = np.random.rand(100, 3)

# Apply PCA to reduce dimensions to 2D

pca = PCA(n\_components=2)

X\_reduced = pca.fit\_transform(X)

# Visualizing variance explained by PCA components

explained\_variance = pca.explained\_variance\_ratio\_

cumulative\_variance = np.cumsum(explained\_variance)

# Reconstruct data using top principal components

X\_reconstructed = pca.inverse\_transform(X\_reduced)

# Load dataset

data = load\_iris()

X, y = data.data, data.target

# Split data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Apply PCA to reduce dimensions

pca = PCA(n\_components=2)

X\_train\_reduced = pca.fit\_transform(X\_train)

X\_test\_reduced = pca.transform(X\_test)

# Train and evaluate logistic regression on reduced data

clf = LogisticRegression()

clf.fit(X\_train\_reduced, y\_train)

y\_pred = clf.predict(X\_test\_reduced)

print("Accuracy after PCA:", accuracy\_score(y\_test, y\_pred))