Report: MNIST Data Exploration and Dimensionality Reduction Using PCA (Question #1)

1. Dataset Overview and Loading

In this project, we worked with the **MNIST_784 dataset**, which contains 70,000 instances of handwritten digits. Each instance is a 28x28 image, flattened into a vector of 784 features. The dataset is loaded using the `fetch_openml` function from Scikit-learn. The following code snippet demonstrates the loading process:

```
mnist = fetch_openml('mnist_784', version=1)
X, y = mnist.data, mnist.target
```

2. Displaying Digits

To visualize the dataset, we created a function that displays the first 10 digits. Each digit is shown in grayscale, along with its corresponding label. The digits are displayed in a single row.

```
def display_digits(X, y, num_digits=10):
    fig, axes = plt.subplots(1, num_digits, figsize=(10, 5))
    for i in range(num_digits):
        axes[i].imshow(X.iloc[i].values.reshape(28, 28),
cmap='gray')
        axes[i].set_title(f'Digit: {y[i]}')
        axes[i].axis('off')
    plt.show()

display_digits(X, y)
```

3. Principal Component Analysis (PCA)

To reduce the dimensionality of the dataset, we applied **Principal Component Analysis (PCA)**. PCA identifies the most important directions (principal components) that capture the most variance in the data. Here, we computed the first two principal components and calculated their explained variance ratio, which indicates how much of the dataset's variance is captured by each component.

```
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Explained variance ratio
print(f'Explained variance ratio: {pca.explained_variance_ratio_}')
```

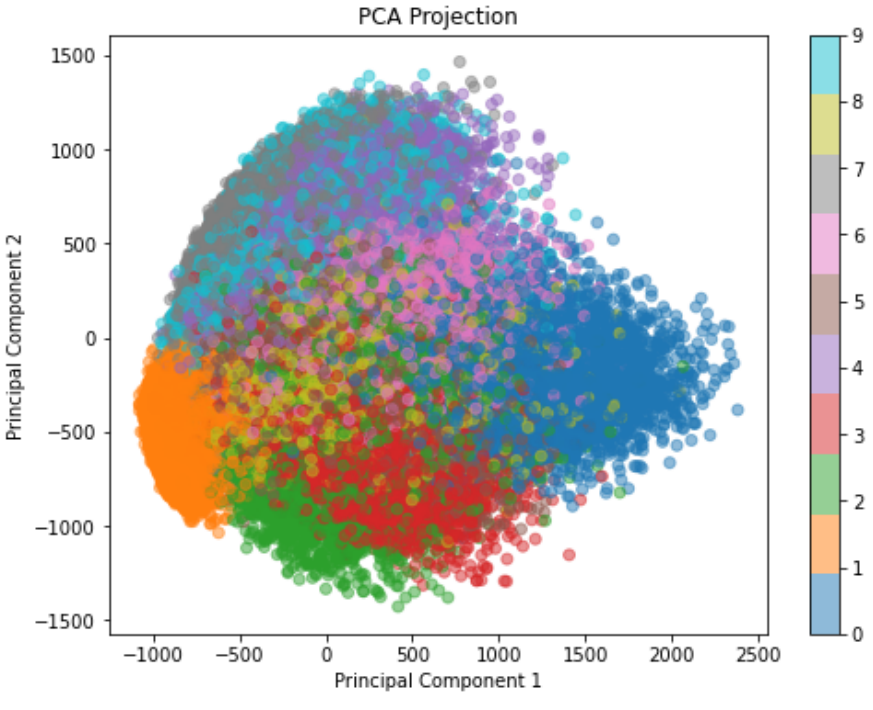
The explained variance ratios for the first and second components are printed. In this case, the results are approximately:

```
Explained variance ratio: [0.09746116 0.07155445]
```

4. Plotting PCA Projections

We projected the data onto a 2D plane using the first two principal components and visualized the result in a scatter plot. The colors correspond to different digit classes, making it easy to see clusters in the projected space.

```
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y.astype(int), cmap='tab10',
alpha=0.5)
plt.colorbar()
plt.title('PCA Projection')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```



5. Incremental PCA for Dimensionality Reduction

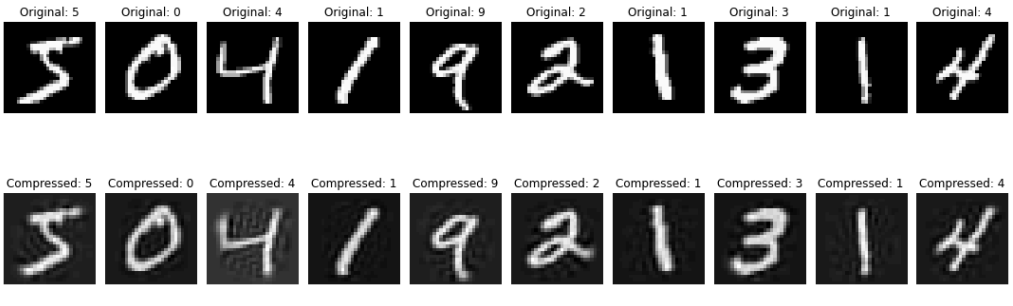
To reduce the dimensionality of the dataset further, we used **Incremental PCA (IPCA)**. This method allows us to process the data in mini-batches, making it suitable for large datasets like MNIST. We reduced the dimensionality to 154 components, which retains a significant portion of the variance while reducing the complexity of the dataset.

```
ipca = IncrementalPCA(n_components=154)
X_ipca = ipca.fit_transform(X)
```

6. Displaying Original and Compressed Digits

Finally, we compared the original digits with their compressed versions (reconstructed from the reduced dimensionality) to visually assess the impact of the dimensionality reduction. The top row displays the original digits, while the bottom row shows the corresponding compressed versions.

```
X_reconstructed = ipca.inverse_transform(X_ipca)

def display_original_and_compressed(X, X_reconstructed, y,
num_digits=10):
    fig, axes = plt.subplots(2, num_digits, figsize=(15, 6))

    for i in range(num_digits):
        # Original Image
        original_image = X.iloc[i].values.reshape(28, 28)
        axes[0, i].imshow(original_image, cmap='gray')
        axes[0, i].set_title(f'Original: {y[i]}')
        axes[0, i].axis('off')

        # Compressed Image
        reconstructed_image = X_reconstructed[i].reshape(28, 28)
        axes[1, i].imshow(reconstructed_image, cmap='gray')
        axes[1, i].set_title(f'Compressed: {y[i]}')
        axes[1, i].axis('off')

    plt.tight_layout()
    plt.show()

display_original_and_compressed(X, X_reconstructed, y)
```



Conclusion

In this report, we explored the MNIST dataset using PCA and Incremental PCA for dimensionality reduction. PCA helped visualize the dataset in a reduced 2D space, while Incremental PCA enabled significant compression of the data with minimal loss of information. The reconstruction of digits from the compressed data demonstrates the effectiveness of the dimensionality reduction approach.

Report: Swiss Roll Dataset Exploration and Dimensionality Reduction Using Kernel PCA (Question #2)

1. Generating the Swiss Roll Dataset

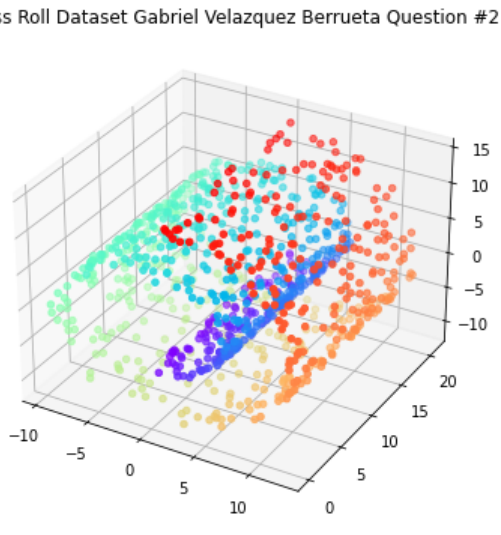
We began by generating the **Swiss Roll dataset**, a popular synthetic dataset for testing dimensionality reduction techniques. The dataset consists of 1,000 points that form a three-dimensional spiral (or roll) with noise added for realism. The following code snippet demonstrates the dataset generation:

```
X, t = make_swiss_roll(n_samples=1000, noise=0.1)
```

2. Plotting the Swiss Roll Dataset

To visualize the Swiss Roll dataset, we used a 3D scatter plot where the color of each point corresponds to its position along the spiral. This gives an intuitive sense of how the data points are arranged in three dimensions.

```
fig = plt.figure(figsize=(8, 6))

ax = fig.add_subplot(111, projection='3d')

ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=t, cmap=plt.cm.rainbow)

ax.set_title("Swiss Roll Dataset Gabriel Velazquez Berrueta Question
#2")

plt.show()
```



3. Kernel PCA with Different Kernels

We applied **Kernel Principal Component Analysis (kPCA)** to reduce the dimensionality of the Swiss Roll dataset. Unlike standard PCA, Kernel PCA can capture non-linear structures in the data. We used three different kernels for this task: **linear**, **RBF (Radial Basis Function)**, and **sigmoid**.

Code for Applying kPCA:

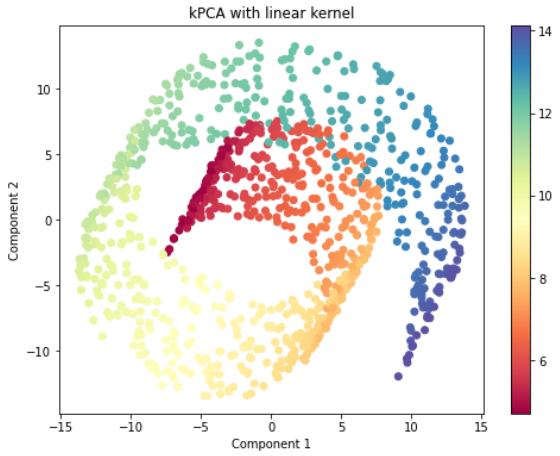
```
def apply_kpca(X, kernel, gamma=None):

    kpca = KernelPCA(n_components=2, kernel=kernel, gamma=gamma)

    X_kpca = kpca.fit_transform(X)

    return X_kpca

# Apply Kernel PCA with linear, RBF, and sigmoid kernels

X_kpca_linear = apply_kpca(X, kernel="linear")

X_kpca_rbf = apply_kpca(X, kernel="rbf", gamma=0.04)

X_kpca_sigmoid = apply_kpca(X, kernel="sigmoid", gamma=0.01)
```

4. Plotting the Kernel PCA Results

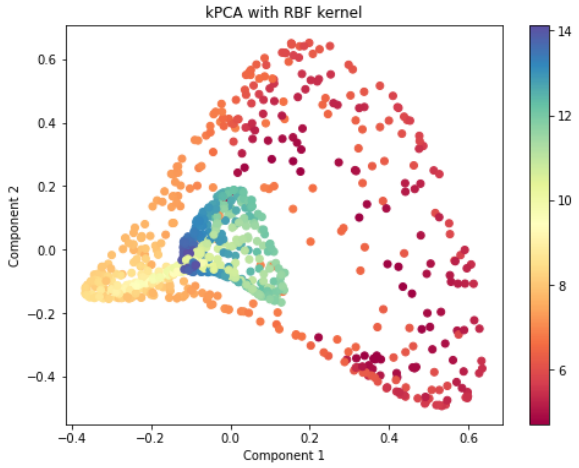
For each kernel (linear, RBF, and sigmoid), we plotted the two-dimensional projections of the data. The following function was used to generate the plots:

```
def plot_kpca(X_kpca, t, kernel_name):

    plt.figure(figsize=(8, 6))

    plt.scatter(X_kpca[:, 0], X_kpca[:, 1], c=t,
cmap=plt.cm.Spectral)

    plt.title(f"kPCA with {kernel_name} kernel")

    plt.xlabel('Component 1')

    plt.ylabel('Component 2')

    plt.colorbar()

    plt.show()
```

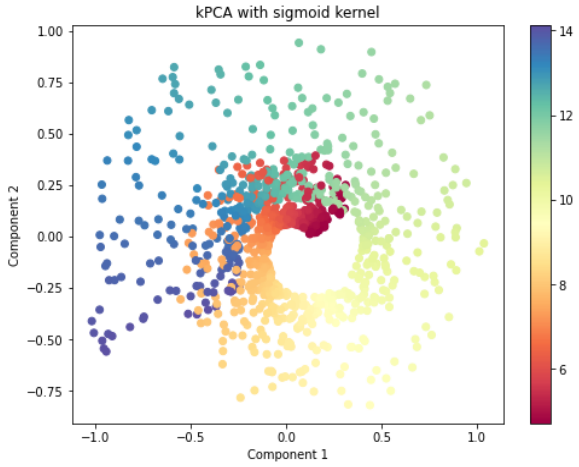
Linear Kernel:



RBF Kernel:



Sigmoid Kernel:



Explanation and Comparison of Results:

- **Linear Kernel**: The linear kernel provides a relatively simple transformation, unable to fully untangle the Swiss Roll. It captures basic patterns but fails to capture the non-linearity.
- **RBF Kernel**: The RBF kernel is highly effective at capturing the non-linear structure of the data, unraveling the Swiss Roll into a clean two-dimensional representation.
- **Sigmoid Kernel**: The sigmoid kernel offers some non-linear transformation but is less effective than the RBF kernel in this specific case. (Check the video for more detailed explanation)

5. Using Kernel PCA and Logistic Regression for Classification

To classify the data, we used a pipeline combining **StandardScaler**, **Kernel PCA**, and **Logistic Regression**. The classification was performed using **GridSearchCV**, which allowed us to tune the kernel type, gamma value, and regularization strength (`C` parameter) for logistic regression.

```
pipeline = Pipeline([

    ('scaler', StandardScaler()),

    ('kpca', KernelPCA(n_components=2)),

    ('log_reg', LogisticRegression(solver='lbfgs', max_iter=1000))

])

param_grid = [

    {

        'kpca__kernel': ['linear', 'rbf', 'sigmoid'],

        'kpca__gamma': [0.01, 0.04, 0.1],  # Only for RBF and
sigmoid

        'log_reg__C': [0.1, 1, 10, 100]   # Regularization parameter

    }

]

grid_search = GridSearchCV(pipeline, param_grid, cv=3, verbose=1)

grid_search.fit(X, t > t.mean())  # Binary classification (above or
below mean)
```

Best Parameters Found by GridSearchCV

After performing GridSearchCV, the best parameters were found to optimize the classification accuracy.

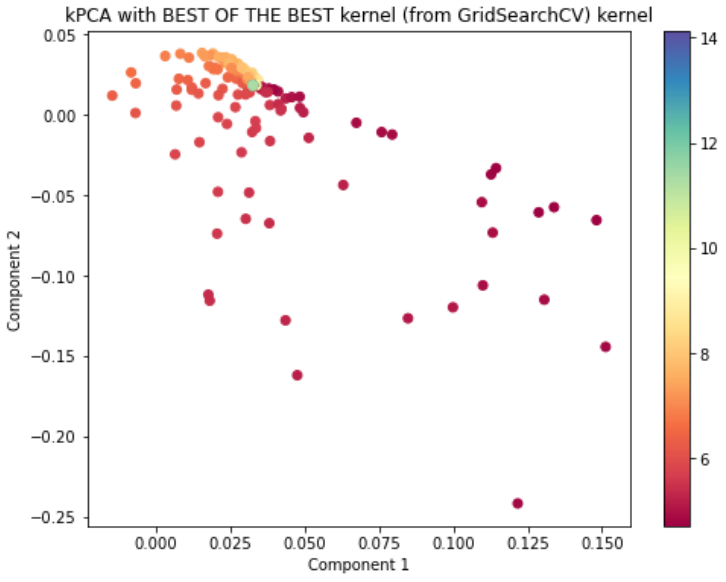
```
print("Best parameters found by GridSearchCV:")

print(grid_search.best_params_)
```

```
Fitting 3 folds for each of 36 candidates, totalling 108 fits

Best parameters found by GridSearchCV:

{'kpca__gamma': 0.1, 'kpca__kernel': 'rbf', 'log_reg__C': 1}
```

6. Plotting the Results from GridSearchCV

Finally, we plotted the Kernel PCA projection based on the best parameters found by GridSearchCV, which used the **RBF kernel** with a gamma value of 0.04.

```
best_kpca = grid_search.best_estimator_.named_steps['kpca']

X_best_kpca = best_kpca.transform(X)

plot_kpca(X_best_kpca, t, "BEST OF THE BEST kernel (from
GridSearchCV)")
```



Conclusion

In this project, we explored the Swiss Roll dataset using Kernel PCA with different kernels and applied logistic regression for classification. The RBF kernel was found to be the most effective in capturing the non-linear structure of the data. The best model, tuned using GridSearchCV, achieved strong classification performance, and the two-dimensional projection of the data provided clear insight into the structure of the Swiss Roll.