1. Simulate any random rectangular matrix A.

```
[[0.37454012 0.95071431 0.73199394]
 [0.59865848 0.15601864 0.15599452]
 [0.05808361 0.86617615 0.60111501]
 [0.70807258 0.02058449 0.96990985]
 [0.83244264 0.21233911 0.18182497]
 [0.18340451 0.30424224 0.52475643]]
```

1.1 What is the rank and trace of A?

- The Rank is 3, it means there are 3 columns that are linearly independent of each other.
- The trace is 1.13, it means that the sum of the elements on the main diagonal of is approximately 1.13

1.2 What is the determinant of A?

The matrix is rectangular, determinant couldn't be calculated

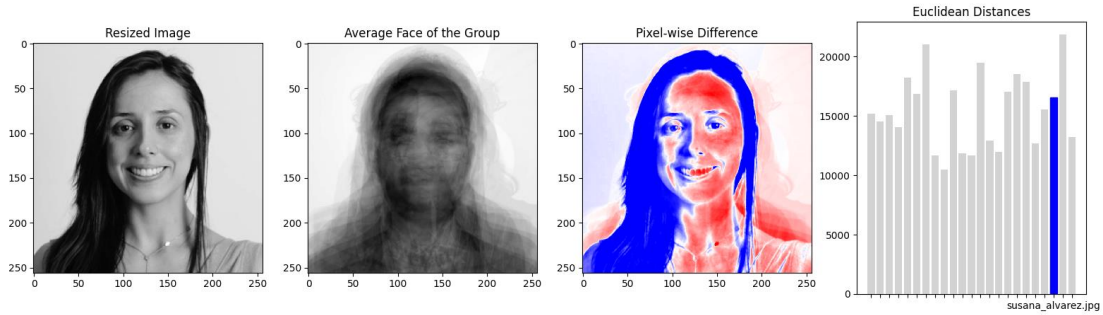
1.3 Can you invert A? How? We can use the pseudoinverse of a rectangular matrix, which is a generalization of the inverse for non-square matrices.

```
[[ 0.00796084  0.58223574 -0.25960078 -0.00443689  0.83810583 -0.16900818]
 [ 0.55029478  0.16692412  0.50583572 -0.75638671  0.25999659 -0.08873467]
 [-0.03759846 -0.39892527  0.09677574  0.95802259 -0.60717309  0.40548779]]
```

1.4 How are eigenvalues and eigenvectors of A'A and AA' related? What interesting differences can you notice between both?

The eigenvalues are the same for both A'A and AA' reflecting the symmetry of these matrices. The differences in the output are related to numerical precision and the presence of complex numbers in one of the computations.

2. How distant is your face from the average? How would you measure it?



The distance of susana_alvarez.jpg (approximately 16581.30341167026) is significantly greater than some other distances, such as those of jeison_arias.jpg (10486.15) and Jacobo Matteucci.jpg (11683.70), but less than others like tatiana_garcia.jpg (21640.57) and Huberth Hincapie.jpg (21053.33). In this sense, it can be said that the face in the image susana_alvarez.jpg has pixel characteristics that make it relatively distant from the average face in the group

The code calculates the Euclidean distance between the face in 'susana_alvarez.jpg' and the average face. The Euclidean distance is a measure of the straight-line distance between two points in a space. In this context, it quantifies the dissimilarity or distance between pixel values of the two faces.

Also, the code calculates the Euclidean distances for all other images in the specified directory. This step allows for a comprehensive analysis of how 'distant' each face is from the average face within the group.

By displaying these Euclidean distances in a bar chart, the code provides a visual representation of how each face in the group compares to the average face. The blue bar corresponds to the 'susana_alvarez.jpg' face, highlighting its Euclidean distance in relation to others.

Also, in the graph call “Pixel-wise difference” visually represents how each pixel in the 'susana_alvarez.jpg' face differs from the corresponding pixel in the average face. Blue regions indicate areas where the 'susana_alvarez.jpg' face has lower pixel values than the average, while red regions indicate higher pixel values. White regions suggest little to no difference between the faces in those areas.

3. The package is structured as follows:

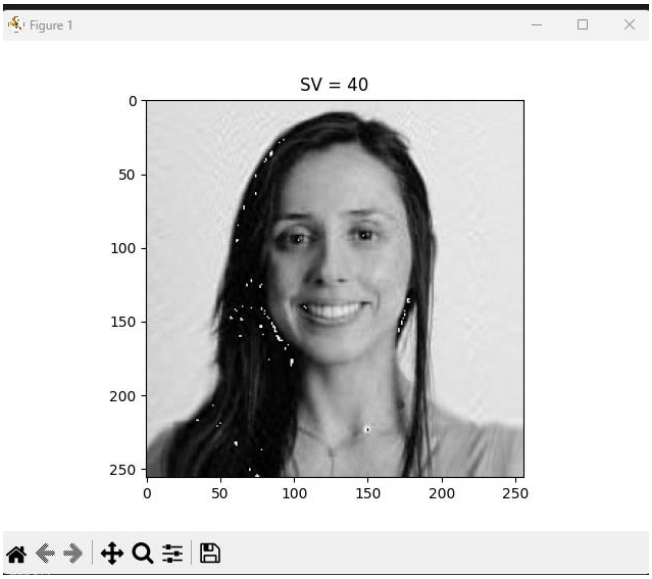
```
unsupervised/
|-- unsupervised/
|   |-- __init__.py
|   |-- pca.py
|   |-- svd.py
|   |-- tsne.py
|-- test_pca.py
|-- test_svd.py
|-- test_tsne.py
|-- setup.py
|-- Pipfile
|-- Pipfile.lock
```

- **unsupervised/:**
  - **unsupervised/:** This is the main directory containing the modules and packages of the implementation.
- **unsupervised/unsupervised/:**
  - **init.py:** This file indicates that the 'unsupervised' folder is a Python package.
  - **pca.py:** This file contains the implementation of the PCA algorithm.
  - **svd.py:** This file contains the implementation of the SVD algorithm.
  - **tsne.py:** This file contains the implementation of the t-SNE algorithm.
- **Test Files:**
  - **test_pca.py:** This file is a test script for the PCA implementation.
  - **test_svd.py:** This file is a test script for the SVD implementation.
  - **test_tsne.py:** This file is a test script for the t-SNE implementation.
- **setup.py:** This file is used to define the package distribution configuration. It contains information such as the package name, version, and dependencies.
- **Pipfile and Pipfile.lock:** These files are used by Pipenv to manage project dependencies. They contain information about the packages necessary for the proper execution of the modules.

The code was generated with Python version 3.12.2. To run a test of the different modules, follow these instructions:

- Ensure Python 3.12 is installed.
- Install Pipenv (if not already installed) by running pip install pipenv.
- Navigate to the project directory using the terminal.
- As Pipenv was used to manage dependencies, run pipenv install to install the dependencies specified in the Pipfile.
- Run the test files to test each of the modules.

4. After SVD is applied to the photo, It is observed that the optimal singular value representing my face corresponds to the 40th component



5. Train a naive logistic regression on raw MNIST images to distinguish between 0s and 8s. We are calling this our baseline. What can you tell about the baseline performance?

Results:

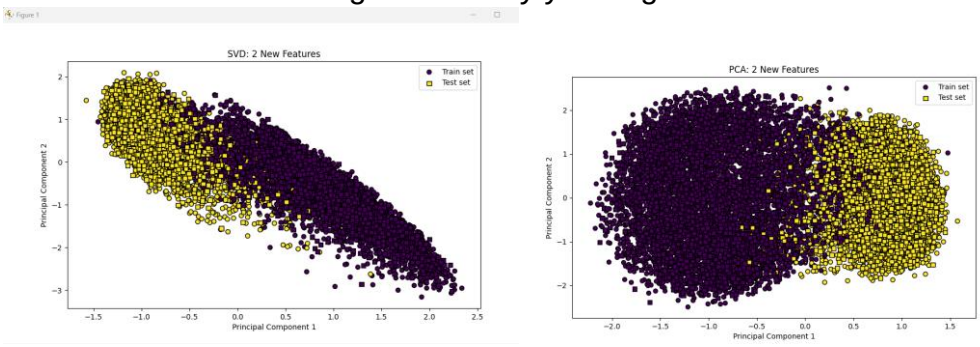
- Accuracy: 99.16%
- Precision: 99.12%
- F1 Score: 99.16%

The obtained results indicate excellent performance for the model in distinguishing between the digits '0' and '8' in the test set. Here's an interpretation of each metric:

- **Accuracy:** With an accuracy of 99.16%, the model correctly classified 99.16% of all instances in the test set. It provides a comprehensive measure of overall performance.
- **Precision:** A precision of 99.12% means that 99.12% of the instances predicted as positive (digit '8' in this case) were actually positive. It is a valuable metric when emphasizing the reliability of positive predictions.
- **F1 Score:** The F1 score, at 99.16%, represents the harmonic mean of precision and recall. It provides a balance between precision and recall, with a higher value indicating a better trade-off between the two.

6. Now, apply dimensionality reduction using all your algorithms to train the model with only 2 features per image.

   Plot the 2 new features generated by your algorithm:



   - Does this somehow impact the performance of your model?

   I only ran SVD and PCA because t-SNE took a long time and didn't finish. So, I decided to work only with SVD and PCA, and these are the conclusions:

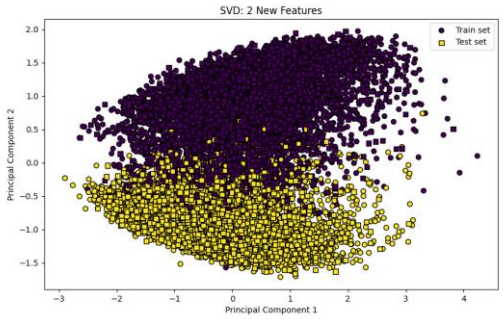
   Dimensionality reduction has allowed simplifying the representation of the data while maintaining generally high performance in classifying digits 0 and 8 in the MNIST dataset. Both techniques, SVD and PCA, have proven to be efficient in preserving relevant information:

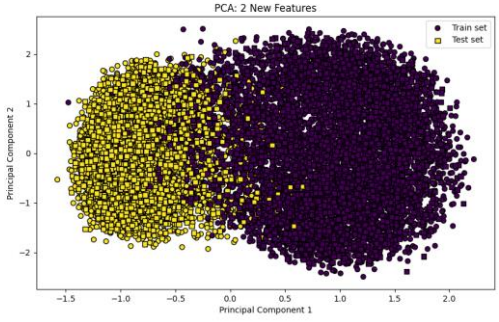
```
Without dimensionality reduction:
Accuracy: 99.16%
Precision: 99.12%
Recall: 99.19%
F1 Score: 99.16%

With dimensionality reduction (SVD):
Accuracy: 95.48%
Precision: 94.27%
Recall: 96.77%
F1 Score: 95.50%

With dimensionality reduction (PCA):
Accuracy: 95.48%
Precision: 94.59%
Recall: 96.40%
F1 Score: 95.49%
```

7. Repeat the process above but now using the built-in algorithms in the Scikit-Learn library.






```
NameError: name 'TruncatedSVD' is not defined
PS D:\Especializacion\ML2-Lab1> python 7.py

With dimensionality reduction (SVD):
Accuracy: 95.52%
Precision: 94.28%
Recall: 96.84%
F1 Score: 95.54%

With dimensionality reduction (PCA):
Accuracy: 95.48%
Precision: 94.59%
Recall: 96.40%
F1 Score: 95.49%
```

The results from the built-in Scikit-Learn algorithms and the custom implementation are very close, with only minor differences and can be attributed to variations in the underlying implementations and optimization techniques used by Scikit-Learn. In practice, these small discrepancies are normal and don't significantly impact the overall performance of the dimensionality reduction techniques. Scikit-Learn's implementation is likely optimized for efficiency and numerical stability.

The similarity in results suggests that both implementations are effectively reducing dimensionality while maintaining high performance in classifying digits 0 and 8 on the MNIST dataset.

8. What strategies do you know (or can think of) in order to make PCA more robust? (Bonus points for implementing them)

To make Principal Component Analysis (PCA) more robust, especially in the presence of noise or outliers, several strategies can be employed:

- **Robust PCA (rPCA):** Is a specific method designed to handle noise and outliers by decomposing a matrix into the sum of a low-rank matrix and a

sparse matrix. This technique can be particularly effective when dealing with data containing both structured patterns and sparse outliers.

- **Truncated Singular Value Decomposition (Truncated SVD):** Truncated SVD is a variant of PCA that involves selecting only the top k singular values and their corresponding vectors. This can help mitigate the impact of noise and focus on the dominant patterns in the data.
- **Data Preprocessing:** Standardize or normalize the data before applying PCA. This can be beneficial for handling data with varying scales and can improve the performance of PCA.
- **Outlier Detection:** Prior to applying PCA, we can remove or mitigate outliers in the dataset. Various outlier detection methods, such as z-scores, isolation forests, or robust statistical measures, can be employed.
- **Robust Covariance Estimation:** Instead of relying on the sample covariance matrix, consider using robust covariance estimation methods that are less sensitive to outliers, such as Minimum Covariance Determinant (MCD) or Huber's M-estimator.

9. Uniform Manifold Approximation and Projection (UMAP) is a modern machine learning algorithm designed for dimensionality reduction. It leverages concepts from Riemannian geometry to construct a graph in higher dimensions, forming connections between data points. This graph, known as a manifold, represents a probability distribution that reflects the weights or probabilities of connections between points and their neighbors. The goal of UMAP is to find this probability distribution in a higher dimension, revealing connections not visible in other dimensions. Maintaining these connections during dimensionality reduction involves finding a probability distribution in a lower dimension.

   UMAP excels in reducing dimensionality while preserving data structures, making it valuable for tasks such as visualization, feature extraction, exploratory data analysis, clustering, and classification. Its ability to handle large and complex datasets, capturing both local and global structures, sets it apart from traditional dimensionality reduction techniques.

10. Latent Dirichlet Allocation (LDA) is a generative probabilistic model for topic modeling, particularly in text data. It leverages the Dirichlet distribution to model the distribution of topics in a document and the distribution of words in a topic. LDA assumes a generative process for creating documents, where each document is a mixture of topics, and each topic is a mixture of words. The underlying mathematical principles involve probabilistic modeling, topic distributions, and a generative process that describes how documents are created based on latent topics.

LDA is useful for various natural language processing (NLP) tasks. It excels in topic modeling, helping to uncover the thematic structure of a collection of documents. The learned topics can be applied to tasks such as document classification, recommendation systems, and information retrieval. LDA provides a powerful framework for extracting meaningful information from unstructured text data by capturing latent topics and their distributions.