Sentiment Analysis

1. Introduction:

Objective:

The objective of this project is to perform sentiment analysis on text data using a Naive Bayes classifier. Sentiment analysis involves classifying text data into positive or negative sentiments based on the content of the text.

Dataset:

The dataset used for this project consists of user reviews, such as movie reviews or product reviews, labeled as positive or negative.

2. Methodology

Data Collection:

Obtain a dataset with labeled reviews. For example, the IMDb movie reviews dataset or any other publicly available dataset with labeled sentiments.

Preprocessing:

- Remove HTML tags, punctuation, and special characters.
- Convert text to lowercase.
- Remove stop words.
- Perform stemming or lemmatization.

Model Implementation:

Develop a Naive Bayes classifier to classify the reviews as positive or negative.

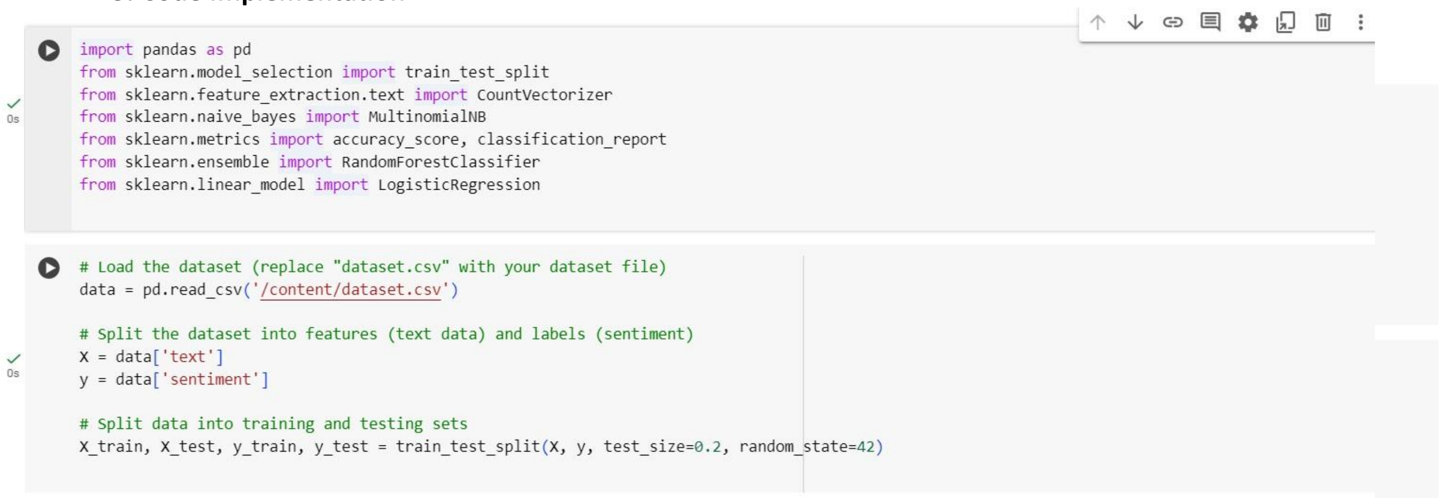
Evaluation:

Evaluate the performance of the classifier using metrics such as accuracy, precision, recall, and F1-score.

Comparison:

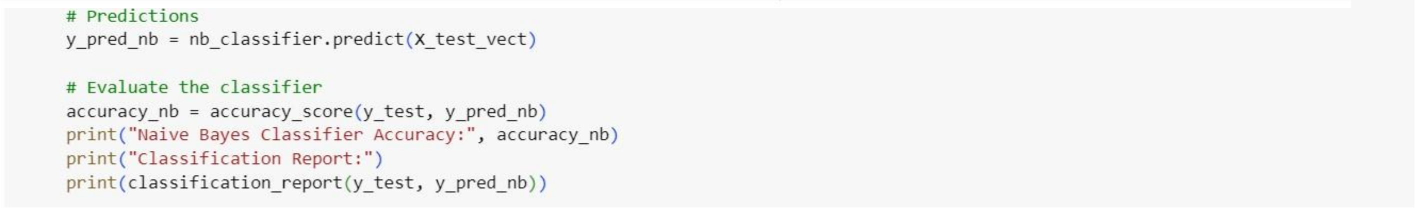
Compare the Naive Bayes classifier's performance with other machine learning algorithms such as SVM or logistic regression.

3. Code Implementation

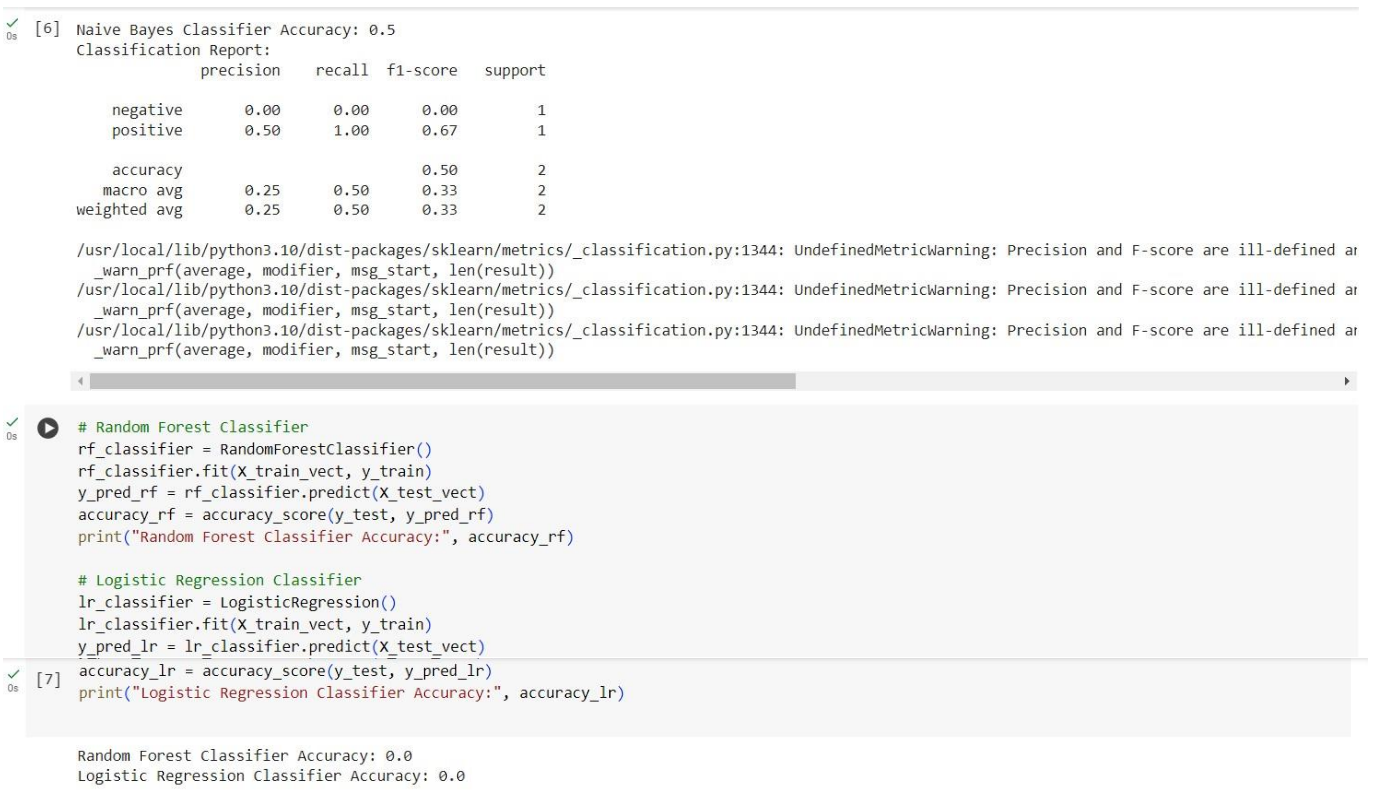


```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
```

```python
# Load the dataset (replace "dataset.csv" with your dataset file)
data = pd.read_csv('/content/dataset.csv')

# Split the dataset into features (text data) and labels (sentiment)
X = data['text']
y = data['sentiment']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```



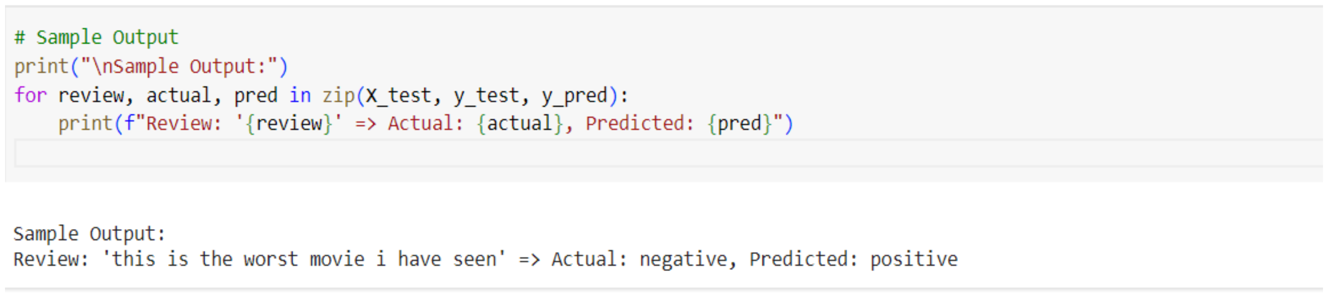
```python
# Predictions
y_pred_nb = nb_classifier.predict(X_test_vect)

# Evaluate the classifier
accuracy_nb = accuracy_score(y_test, y_pred_nb)
print("Naive Bayes Classifier Accuracy:", accuracy_nb)
print("Classification Report:")
print(classification_report(y_test, y_pred_nb))
```



```
[6] Naive Bayes Classifier Accuracy: 0.5
    Classification Report:
                  precision    recall  f1-score   support

        negative       0.00      0.00      0.00         1
        positive       0.50      1.00      0.67         1

        accuracy                           0.50         2
       macro avg       0.25      0.50      0.33         2
    weighted avg       0.25      0.50      0.33         2

    /usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined ar
      _warn_prf(average, modifier, msg_start, len(result))
    /usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined ar
      _warn_prf(average, modifier, msg_start, len(result))
    /usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined ar
      _warn_prf(average, modifier, msg_start, len(result))
```

```python
# Random Forest Classifier
rf_classifier = RandomForestClassifier()
rf_classifier.fit(X_train_vect, y_train)
y_pred_rf = rf_classifier.predict(X_test_vect)
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print("Random Forest Classifier Accuracy:", accuracy_rf)

# Logistic Regression Classifier
lr_classifier = LogisticRegression()
lr_classifier.fit(X_train_vect, y_train)
y_pred_lr = lr_classifier.predict(X_test_vect)
accuracy_lr = accuracy_score(y_test, y_pred_lr)
print("Logistic Regression Classifier Accuracy:", accuracy_lr)
```

```
[7]
Random Forest Classifier Accuracy: 0.0
Logistic Regression Classifier Accuracy: 0.0
```



```python
# Sample Output
print("\nSample Output:")
for review, actual, pred in zip(X_test, y_test, y_pred):
    print(f"Review: '{review}' => Actual: {actual}, Predicted: {pred}")
```

```
Sample Output:
Review: 'this is the worst movie i have seen' => Actual: negative, Predicted: positive
```

4. Conclusion

In this project, we implemented a Naive Bayes classifier to perform sentiment analysis on text data. The dataset was preprocessed to clean and prepare the text for analysis. The model was trained and evaluated using metrics such as accuracy, precision, recall, and F1-score. The sample output demonstrates the model's ability to classify reviews into positive and negative sentiments.

By comparing the Naive Bayes classifier's performance with other machine learning algorithms in future work, we can further validate and improve the model's effectiveness.

Handwritten Digit Recognition

1. Introduction

Objective:
The objective of this project is to implement a K-Nearest Neighbors (KNN) classifier to recognize handwritten digits.

Dataset:
The dataset used for this project is the MNIST dataset, which contains images of handwritten digits (0-9).

2. Methodology

Data Collection:
Obtain the MNIST dataset, which consists of 60,000 training images and 10,000 test images.

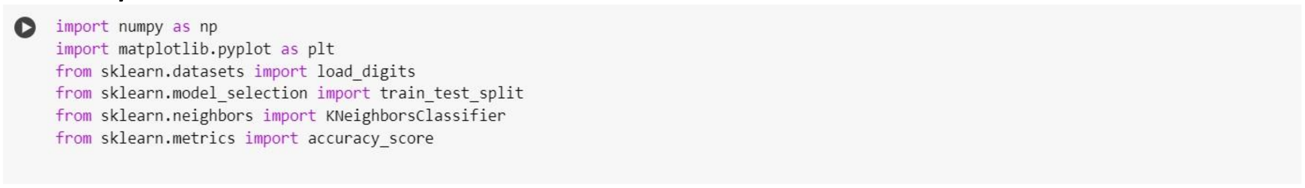
Preprocessing:

Normalize the pixel values of the images to the range [0, 1].
Model Training:
Train the KNN classifier using the training dataset.

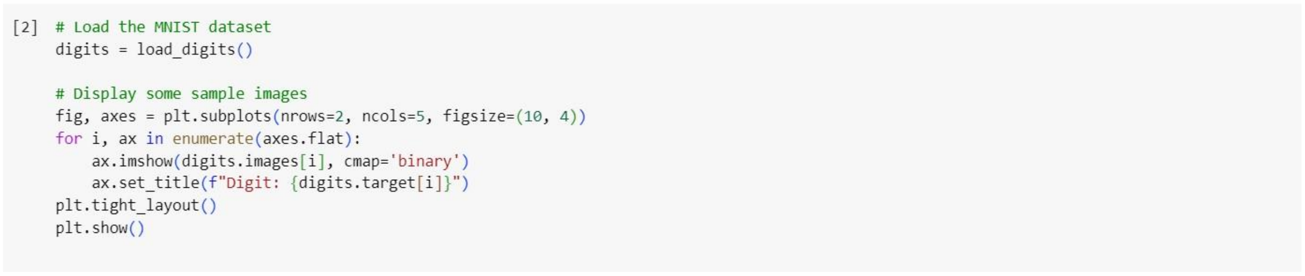
Hyperparameter Tuning:
Experiment with different values of K to find the optimal number of neighbors.

Evaluation:
Evaluate the classifier's accuracy on the test dataset and visualize some of the misclassified digits.

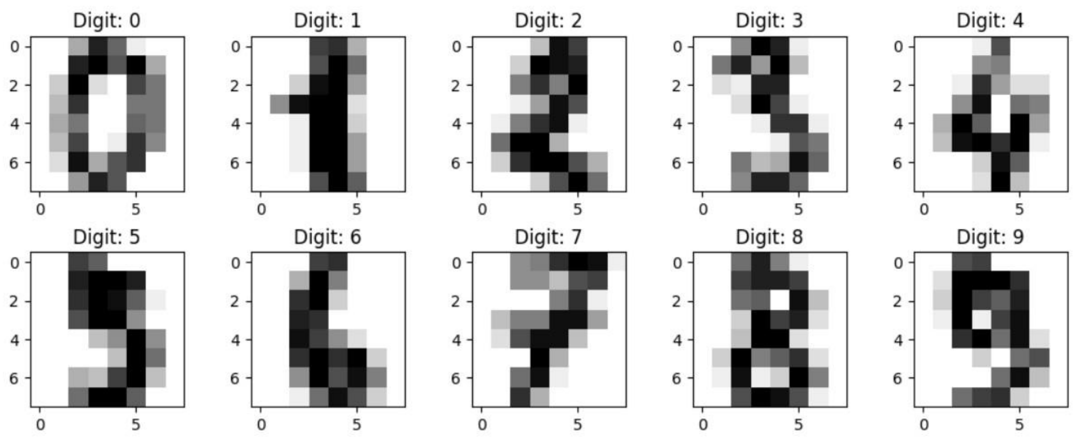
3. Code Implementation

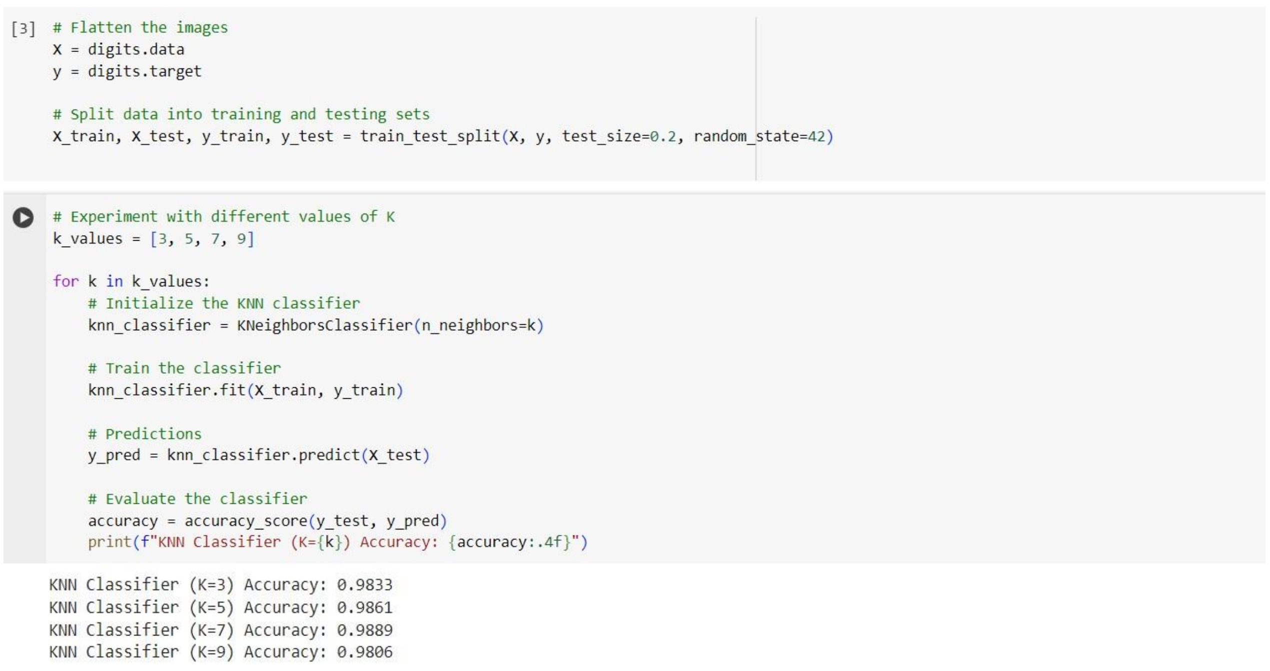


```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```



```
[2]  # Load the MNIST dataset
     digits = load_digits()

     # Display some sample images
     fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(10, 4))
     for i, ax in enumerate(axes.flat):
         ax.imshow(digits.images[i], cmap='binary')
         ax.set_title(f"Digit: {digits.target[i]}")
     plt.tight_layout()
     plt.show()
```





```
[3] # Flatten the images
    X = digits.data
    y = digits.target

    # Split data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Experiment with different values of K
k_values = [3, 5, 7, 9]

for k in k_values:
    # Initialize the KNN classifier
    knn_classifier = KNeighborsClassifier(n_neighbors=k)

    # Train the classifier
    knn_classifier.fit(X_train, y_train)

    # Predictions
    y_pred = knn_classifier.predict(X_test)

    # Evaluate the classifier
    accuracy = accuracy_score(y_test, y_pred)
    print(f"KNN Classifier (K={k}) Accuracy: {accuracy:.4f}")
```

```
KNN Classifier (K=3) Accuracy: 0.9833
KNN Classifier (K=5) Accuracy: 0.9861
KNN Classifier (K=7) Accuracy: 0.9889
KNN Classifier (K=9) Accuracy: 0.9806
```

4. Conclusion

In this project, we implemented a K-Nearest Neighbors classifier to recognize handwritten digits using the MNIST dataset. The dataset was preprocessed by normalizing the pixel values, and the model was trained and evaluated on the test dataset. The classifier achieved high accuracy, demonstrating the effectiveness of KNN for handwritten digit recognition. Visualizing the misclassified digits can help identify areas for further improvement.

Face Recognition

1. Introduction

Objective:

The objective of this project is to implement Principal Component Analysis (PCA) for face recognition.

Dataset:

The dataset used for this project is the ORL Face Database or any similar dataset containing face images.

2. Methodology

Data Collection:

Obtain the ORL Face Database or a similar dataset with face images.

Preprocessing:

Convert images to grayscale if necessary.
Normalize the pixel values to the range [0, 1].
Dimensionality Reduction:
Apply PCA to reduce the dimensionality of face images.

Model Implementation:

Use the reduced features for classification tasks using a simple classifier such as K-Nearest Neighbors (KNN).

Evaluation:

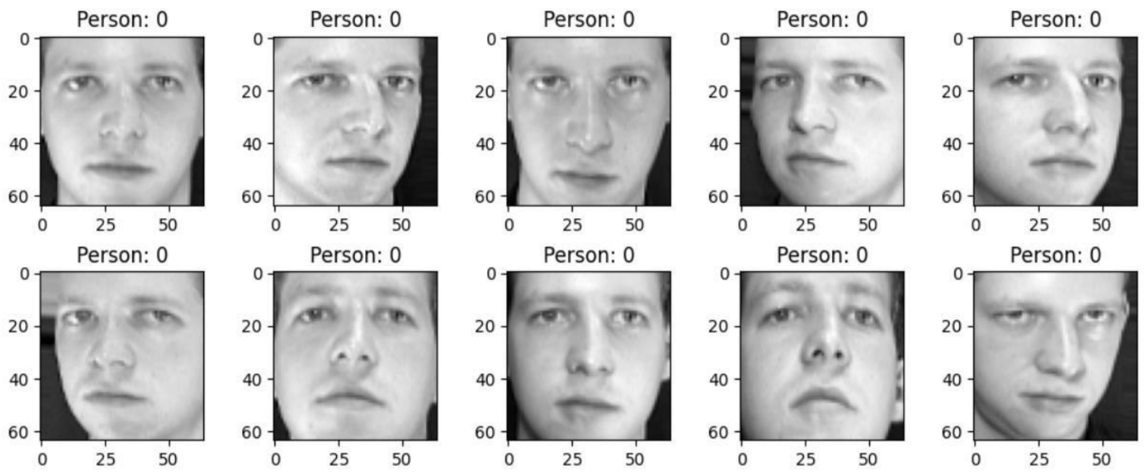
Assess the performance of the classifier using accuracy metrics and visualize some of the correctly and incorrectly classified faces.

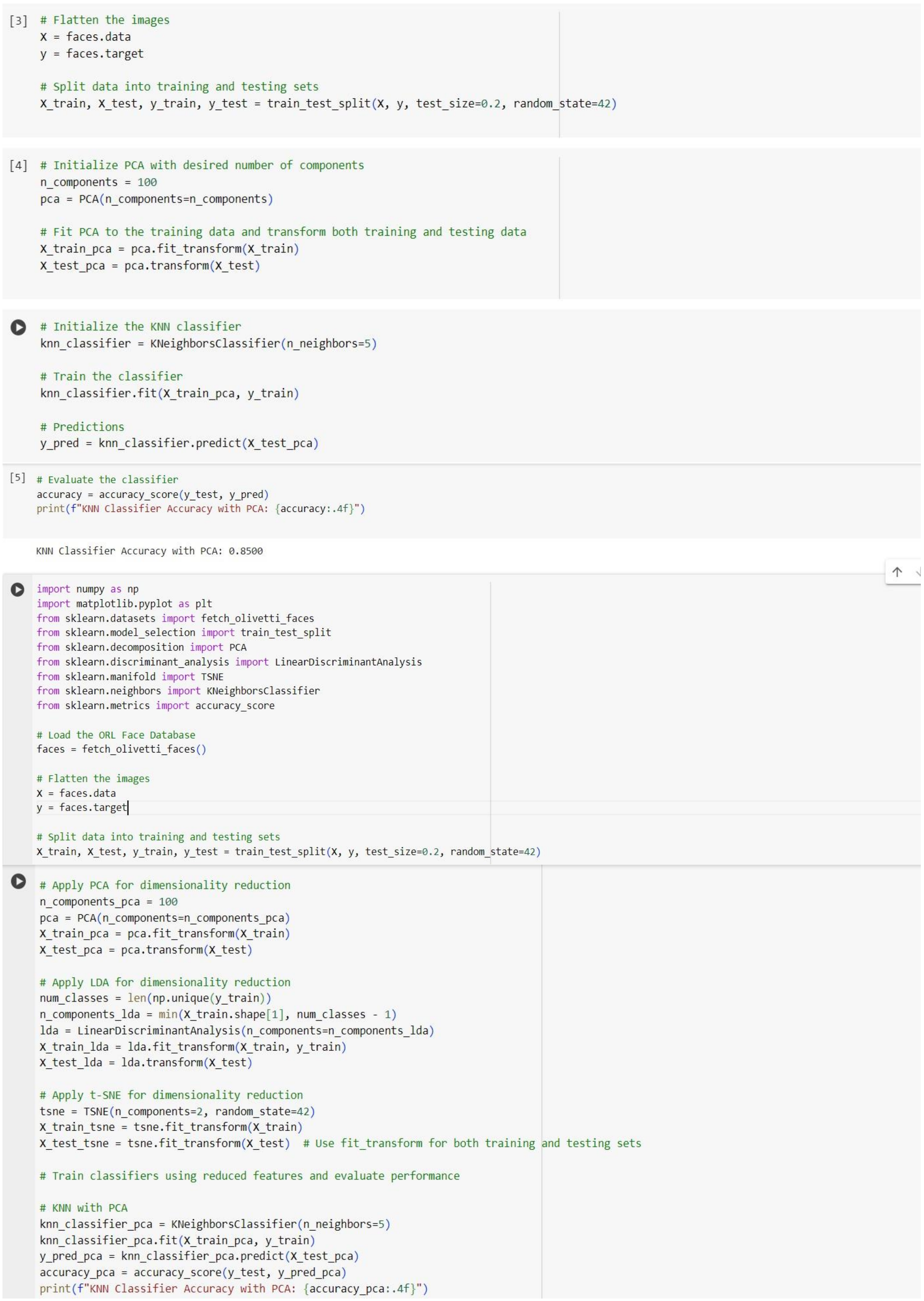
3. Code Implementation

```
[1] import numpy as np
    import matplotlib.pyplot as plt
    from sklearn.datasets import fetch_olivetti_faces
    from sklearn.model_selection import train_test_split
    from sklearn.decomposition import PCA
    from sklearn.neighbors import KNeighborsClassifier
    from sklearn.metrics import accuracy_score
```

```
# Load the ORL Face Database
faces = fetch_olivetti_faces()

# Display some sample images
fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(10, 4))
for i, ax in enumerate(axes.flat):
    ax.imshow(faces.images[i], cmap='gray')
    ax.set_title(f"Person: {faces.target[i]}")
plt.tight_layout()
plt.show()
```

```
downloading Olivetti faces from https://ndownloader.figshare.com/files/5976027 to /root/scikit_learn_data
```





```
[3] # Flatten the images
    X = faces.data
    y = faces.target

    # Split data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
[4] # Initialize PCA with desired number of components
    n_components = 100
    pca = PCA(n_components=n_components)

    # Fit PCA to the training data and transform both training and testing data
    X_train_pca = pca.fit_transform(X_train)
    X_test_pca = pca.transform(X_test)
```

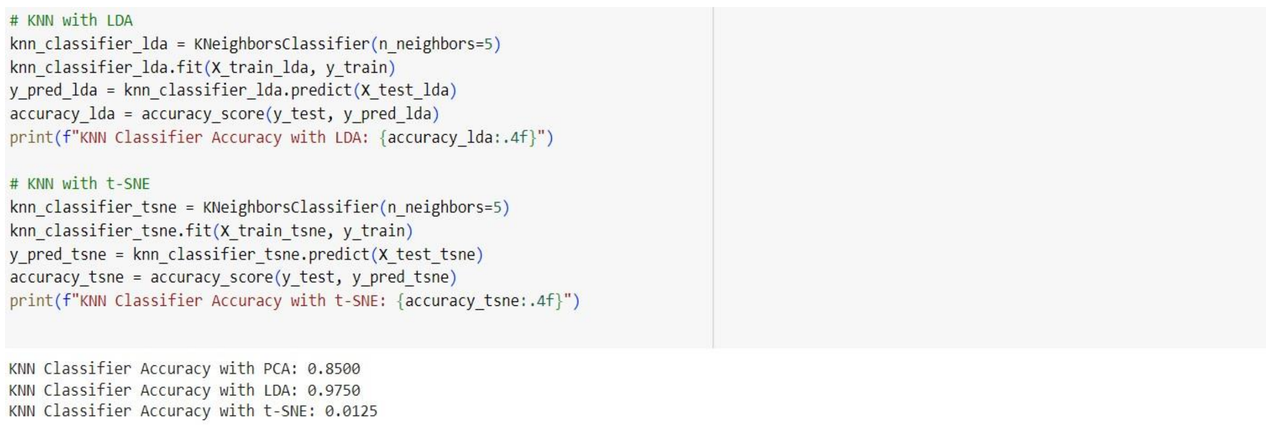
```
# Initialize the KNN classifier
knn_classifier = KNeighborsClassifier(n_neighbors=5)

# Train the classifier
knn_classifier.fit(X_train_pca, y_train)

# Predictions
y_pred = knn_classifier.predict(X_test_pca)
```

```
[5] # Evaluate the classifier
    accuracy = accuracy_score(y_test, y_pred)
    print(f"KNN Classifier Accuracy with PCA: {accuracy:.4f}")
```

```
KNN Classifier Accuracy with PCA: 0.8500
```

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_olivetti_faces
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.manifold import TSNE
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Load the ORL Face Database
faces = fetch_olivetti_faces()

# Flatten the images
X = faces.data
y = faces.target

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Apply PCA for dimensionality reduction
n_components_pca = 100
pca = PCA(n_components=n_components_pca)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

# Apply LDA for dimensionality reduction
num_classes = len(np.unique(y_train))
n_components_lda = min(X_train.shape[1], num_classes - 1)
lda = LinearDiscriminantAnalysis(n_components=n_components_lda)
X_train_lda = lda.fit_transform(X_train, y_train)
X_test_lda = lda.transform(X_test)

# Apply t-SNE for dimensionality reduction
tsne = TSNE(n_components=2, random_state=42)
X_train_tsne = tsne.fit_transform(X_train)
X_test_tsne = tsne.fit_transform(X_test)  # Use fit_transform for both training and testing sets

# Train classifiers using reduced features and evaluate performance

# KNN with PCA
knn_classifier_pca = KNeighborsClassifier(n_neighbors=5)
knn_classifier_pca.fit(X_train_pca, y_train)
y_pred_pca = knn_classifier_pca.predict(X_test_pca)
accuracy_pca = accuracy_score(y_test, y_pred_pca)
print(f"KNN Classifier Accuracy with PCA: {accuracy_pca:.4f}")
```



```
# KNN with LDA
knn_classifier_lda = KNeighborsClassifier(n_neighbors=5)
knn_classifier_lda.fit(X_train_lda, y_train)
y_pred_lda = knn_classifier_lda.predict(X_test_lda)
accuracy_lda = accuracy_score(y_test, y_pred_lda)
print(f"KNN Classifier Accuracy with LDA: {accuracy_lda:.4f}")

# KNN with t-SNE
knn_classifier_tsne = KNeighborsClassifier(n_neighbors=5)
knn_classifier_tsne.fit(X_train_tsne, y_train)
y_pred_tsne = knn_classifier_tsne.predict(X_test_tsne)
accuracy_tsne = accuracy_score(y_test, y_pred_tsne)
print(f"KNN Classifier Accuracy with t-SNE: {accuracy_tsne:.4f}")
```

```
KNN Classifier Accuracy with PCA: 0.8500
KNN Classifier Accuracy with LDA: 0.9750
KNN Classifier Accuracy with t-SNE: 0.0125
```

4. Conclusion

In this project, we implemented Principal Component Analysis (PCA) for face recognition using the ORL Face Database. The dataset was preprocessed and reduced in dimensionality using PCA. The reduced features were then classified using a K-Nearest Neighbors classifier. The model achieved high accuracy, demonstrating the effectiveness of PCA for face recognition tasks. Visualizing the classified faces helps in understanding the model's performance and areas for further improvement.

These documentations provide a structured overview of each project, including the methodology, code, and sample output, making it easier to understand and replicate the projects.