

**Documentation Of Handwritten Alphabets Classification(MNIST dataset)**

**Handwritten Alphabet Classification**

**Introduction**

**Contents**

[Handwritten Alphabet Recognition 1](#_Toc172736127)

[Introduction 2](#_Toc172736128)

[1.1 Project Overview 3](#_Toc172736129)

[1.2 Objectives 4](#_Toc172736130)

[2. Dataset Description 4](#_Toc172736131)

[2.1 MNIST Overview 4](#_Toc172736132)

[2.2 Dataframe Creation 4](#_Toc172736133)

[3 Data Engineering 5](#_Toc172736134)

[3.1 Unzipping and Directory Reading 5](#_Toc172736135)

[3.2 read\_image\_files(path) Function 5](#_Toc172736136)

[3.3 DataFrame Creation and Storage6](#_Toc172736137)

[4.Exploratory Data Analysis 7](#_Toc172736138)

[4.1 DataFrame Structure 7](#_Toc172736139)

[4.2 Label Column 8](#_Toc172736140)

[4.3 Distribution of Labels 8](#_Toc172736141)

[4.4 Sample Images Across the Alphabet 10](#_Toc172736142)

[4.5 Distribution of Pixel Intensities 11](#_Toc172736143)

[4.6 Distance from Mean Pixel Value by Label 12](#_Toc172736144)

[4.7 PCA of Image Data 13](#_Toc172736145)

[5. Data Preparation 14](#_Toc172736146)

[5.1 Data Segregation and Train-Test Split 14](#_Toc172736147)

[5.2 Data Preparation for Training Data 15](#_Toc172736148)

[5.3 Data Prepration for Test Data 15](#_Toc172736149)

[6. Model Development 16](#_Toc172736150)

[6.1 Model Selection 16](#_Toc172736151)

[6.2 Model Training 16](#_Toc172736152)

[**1. Naive Bayes Model 17**](#_Toc172736153)

[**2. Logistic Regression Model 17**](#_Toc172736154)

[**3. Decision Tree Model 18**](#_Toc172736157)

[**4. RandomForest Model Training 19**](#_Toc172736160)

[**5. XGBoost Model 20**](#_Toc172736163)

[6.3 Model Prediction and Evaluation on Unseen Data 20](#_Toc172736166)

[**1. Navie Bayes 21**](#_Toc172736167)

[**2. Logistic Regression 21**](#_Toc172736168)

[**3. Decision Tree 21**](#_Toc172736169)

[**4. Random Forest 22**](#_Toc172736170)

[**5. Xgboost 22**](#_Toc172736171)

[6.4 Comparison and Optimization 23](#_Toc172736172)

[7. Comparision Of Model’s Accuracy 24](#_Toc172736173)

[8. Model Training Time for Deployment 25](#_Toc172736174)

[9. Model Latency Comparison 25](#_Toc172736175)

[Conclusion 26](#_Toc172736176)

**1.1 Project Overview**

The MNIST dataset is a benchmark dataset in computer vision, widely used for training and testing image processing systems. It consists of handwritten digits and has been extended to include alphabets for this project. Recognizing handwritten characters accurately is crucial for various applications, including document digitization and automated data entry.



**1.2 Objectives**

The primary objectives of this project are:

* To preprocess the image files and create a pandas dataframe.
* To annotate each image into one of the 26 alphabet classes.
* To develop a robust machine learning model for character recognition.
* To ensure the model is lightweight and has low latency for deployment.

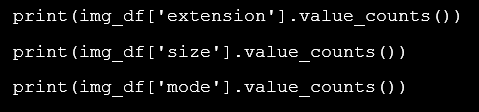
**2. Dataset Description**

**2.1 MNIST Overview**

The MNIST dataset is a collection of handwritten digits and alphabets, originally curated by the National Institute of Standards and Technology (NIST). This project uses a variation of the MNIST dataset that includes 26 classes representing the English alphabets.

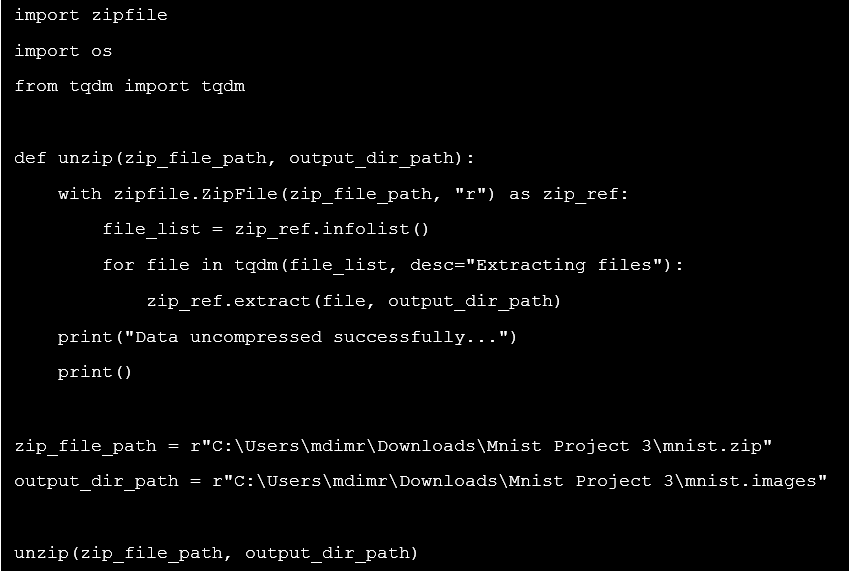
**2.2 Dataframe Creation**

The raw image files are converted into a structured pandas dataframe. Each row in the dataframe represents an image, with columns for the pixel values and the corresponding alphabet label. This preprocessing step is crucial for feeding the data into machine learning models.

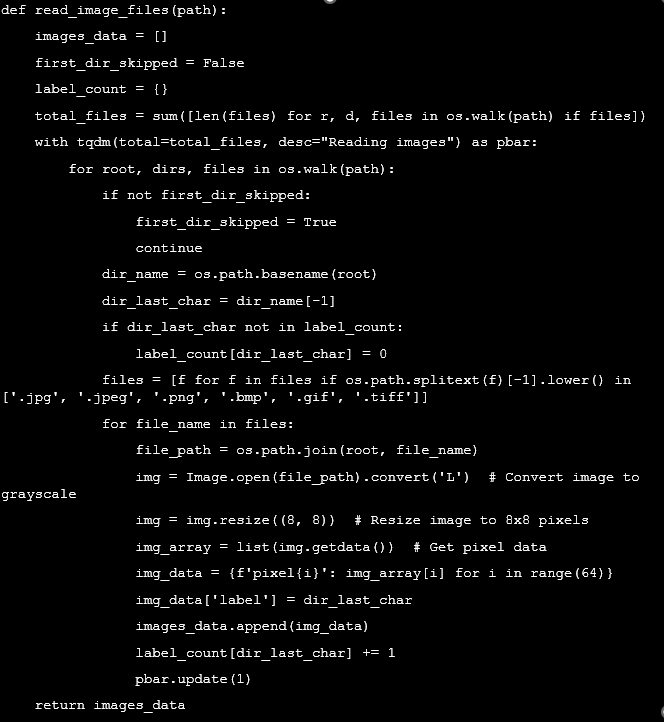
* **Total Number of Files:** 372,451
* **File Extensions:** {'.png'}
* **Mode:** Grayscale ('L') with 372,451 images
* **Size:** (28, 28) pixels with 372,451 images

## 3 Data Engineering

### ****3.1 Unzipping and Directory Reading****

The .zip file is successfully extracted, and images are organized into subdirectories labeled with alphabet letters. The dataset contains 372,451 .png images, each with dimensions 28x28 pixels and in grayscale mode.

### **3.2 read\_image\_files(path) Function**

* This function reads images from the directory, converts them to grayscale, and resizes them to 8x8 pixels. The image data is flattened (using .ravel()) and combined with labels into a list of dictionaries, with progress tracked by tqdm.combined with labels into a list of dictionaries, with progress tracked by tqdm.

### ****DataFrame Creation and Storage****

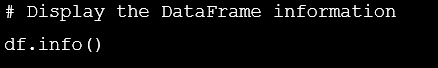
* The list of image data is converted into a pandas DataFrame using load\_into\_df(). The DataFrame is displayed and saved as images\_data.csv for further analysis.

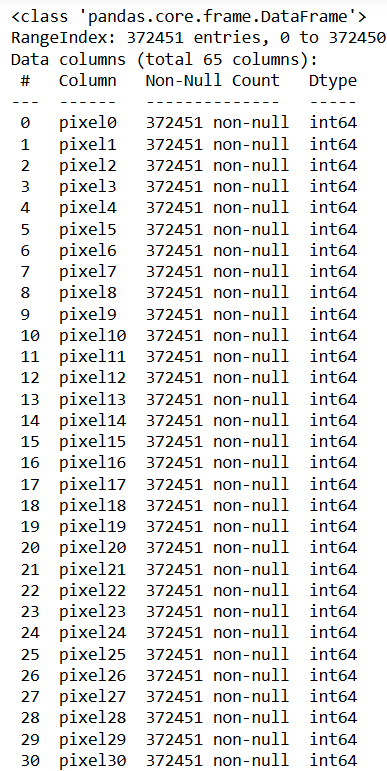
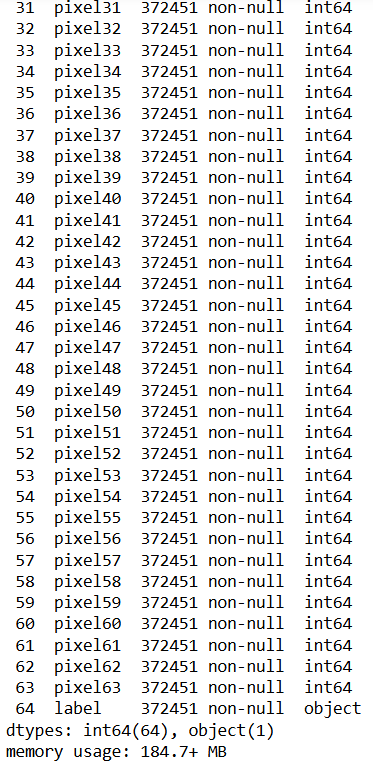
## 4.Exploratory Data Analysis

### 4.1 DataFrame Structure

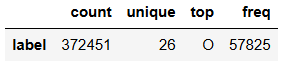
The DataFrame contains 372,451 entries and 65 columns. The columns include 64 numerical columns representing pixel values (pixel0 to pixel63) and 1 categorical column for the labels (label), which is the target variable.

* **Numerical Columns**: 64 columns (pixel0 to pixel63), all of which are of type int64. These columns contain pixel values of the images, and no missing values are present in these columns.
* **Categorical Column**: 1 column (label), of type object, representing the class label of the image. This column is the target variable and also has no missing values.

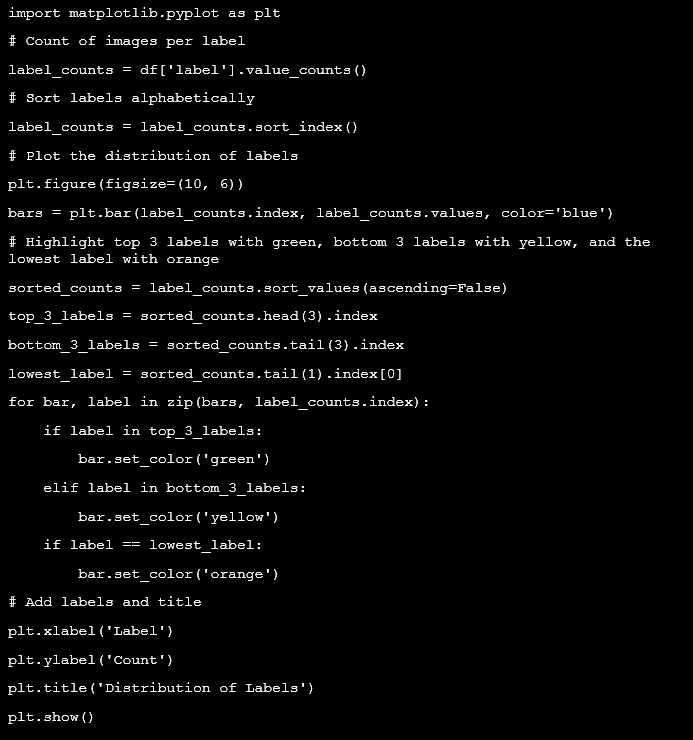
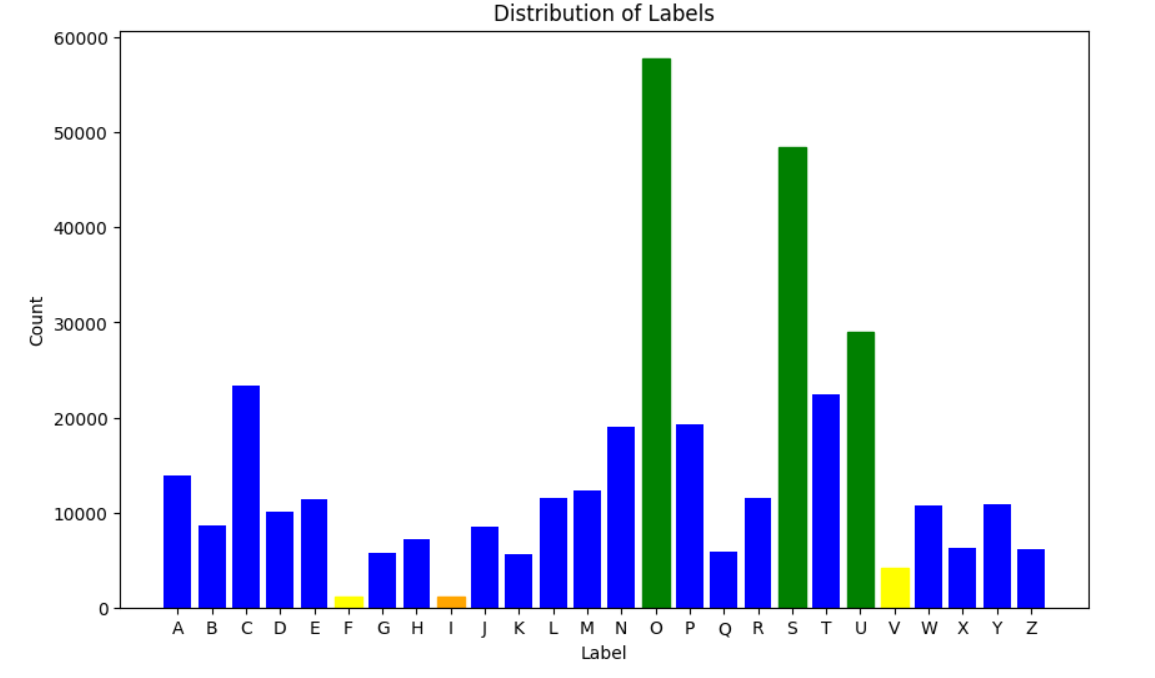
The DataFrame does not have any missing values in any of the columns.



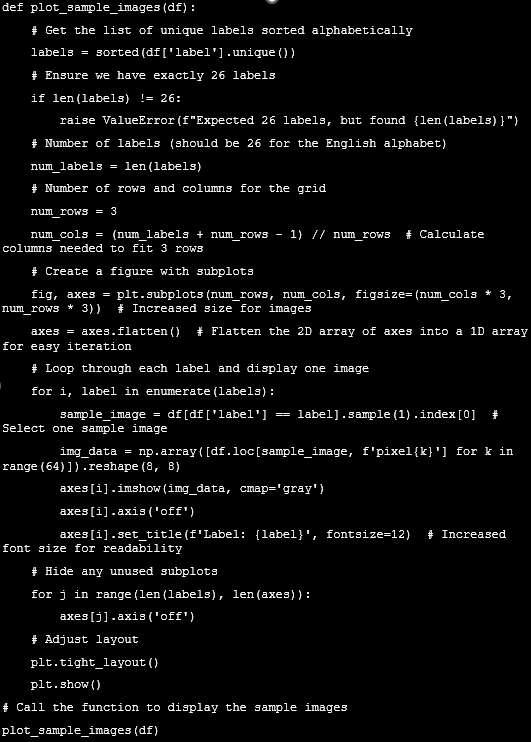
### 4.2 Label Column

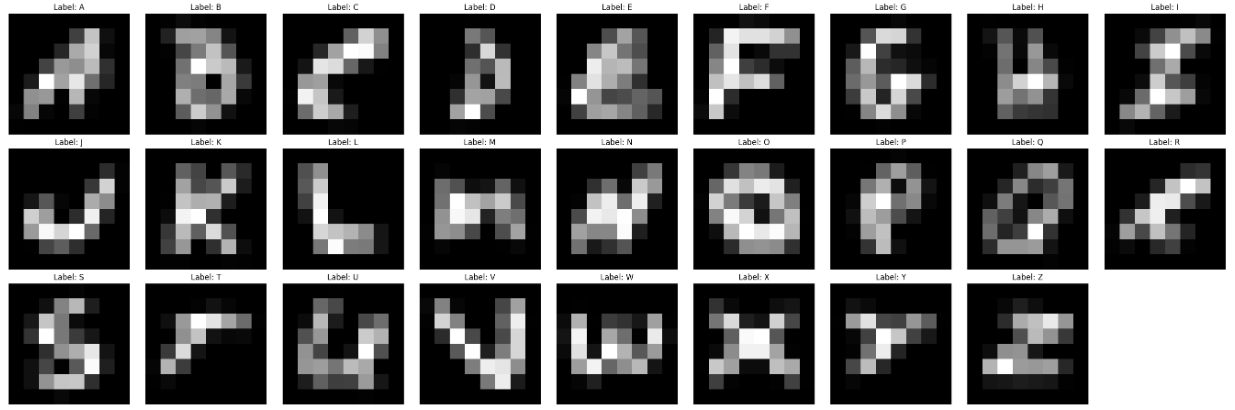
Label column has 372,451 entries with 26 unique classes. The most common class is O, appearing 57,825 times.

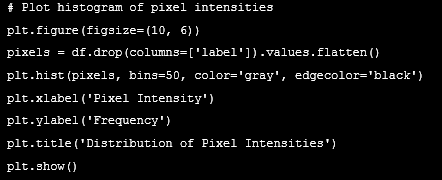
### 4.3 Distribution of Labels

The bar chart shows the distribution of images per label, highlighting the top 3 most frequent labels in green (s, o, u), the bottom 3 in yellow (f, i, v), and the lowest label in orange (f). This visualization reveals label imbalances, with f being significantly under-represented.

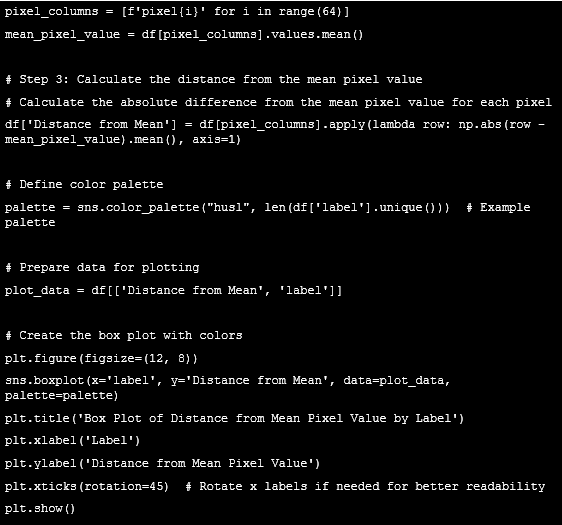
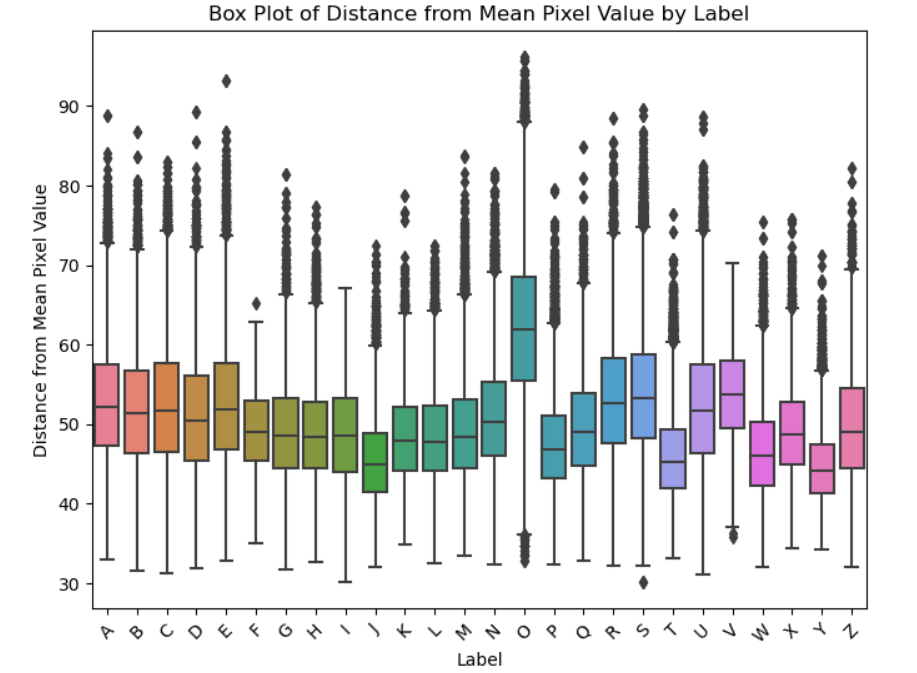
### 4.4 Sample Images Across the Alphabet

****The visualization displays one sample image for each letter of the alphabet, revealing diverse styles and varying levels of clarity. While most letters are clearly represented, some images are blurred, indicating inconsistency in the quality of the handwritten samples across the dataset.

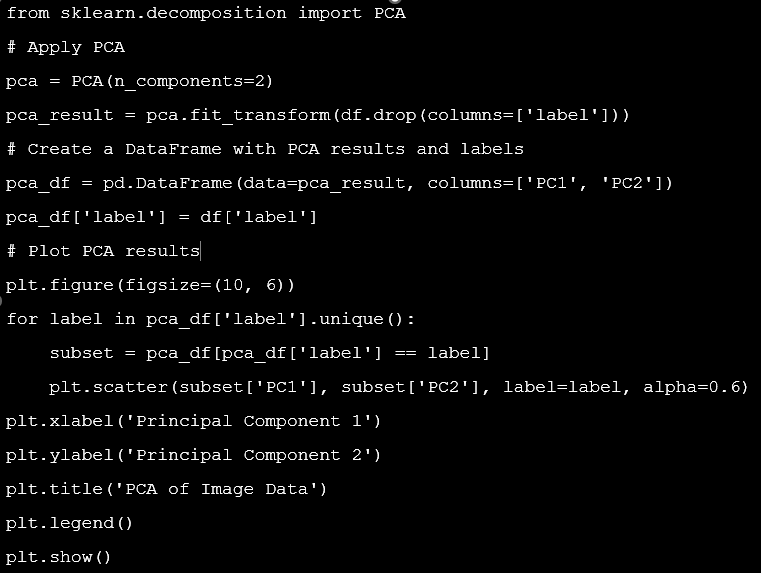
**4.5 Distribution of Pixel Intensities**

The histogram of pixel intensities shows that the most frequent pixel value is near 0, indicating that the majority of the image pixels are dark. Pixels with higher values, closer to 250, are less common, suggesting that lighter pixels are relatively rare. This distribution highlights that the dataset predominantly consists of darker pixels, with lighter pixels being infrequent.

### ****4.6 Distance from Mean Pixel Value by Label****

The box plot illustrates the variability of pixel values for each label, showing that different labels exhibit varying degrees of deviation from the mean pixel value. Notably, some labels have extreme outliers, indicating considerable differences in pixel intensities compared to the mean. The label 'O' stands out as having distinct pixel intensity patterns, with a noticeable deviation from the average compared to other labels.****

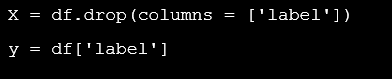
### 4.7 PCA of Image Data

The PCA plot shows distinct clusters for different letters, highlighting effective dimensionality reduction. However, some overlap indicates feature similarities between classes.

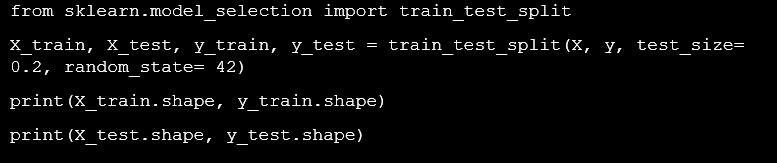
## 5. Data Preparation

### ****5.1 Data Segregation and Train-Test Split****

The data is segregated into inputs (X) and outputs (y), with X containing pixel values and y containing labels.

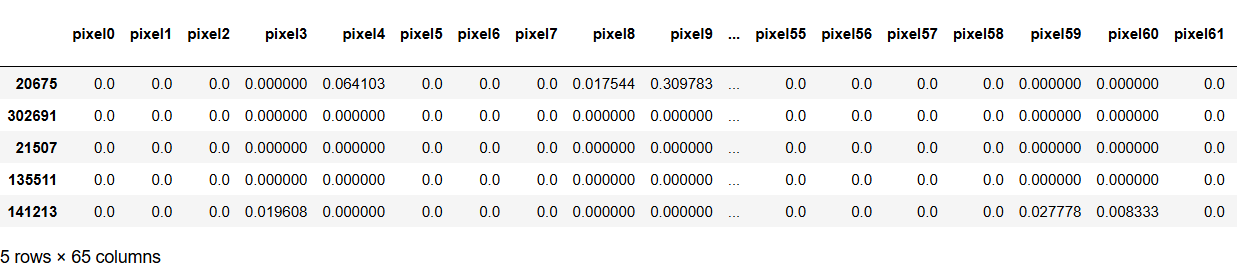
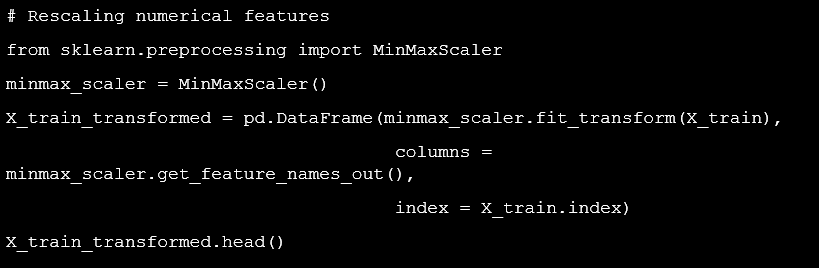


The dataset is then split into training (80%) and testing (20%) subsets, resulting in 297,960 samples for training and 74,491 samples for testing, ensuring a balanced evaluation of model performance.



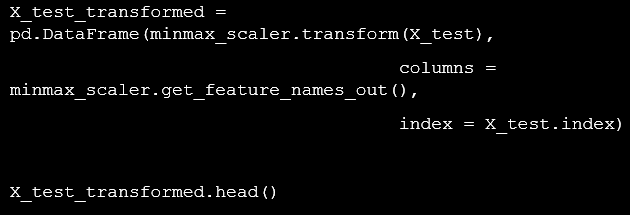


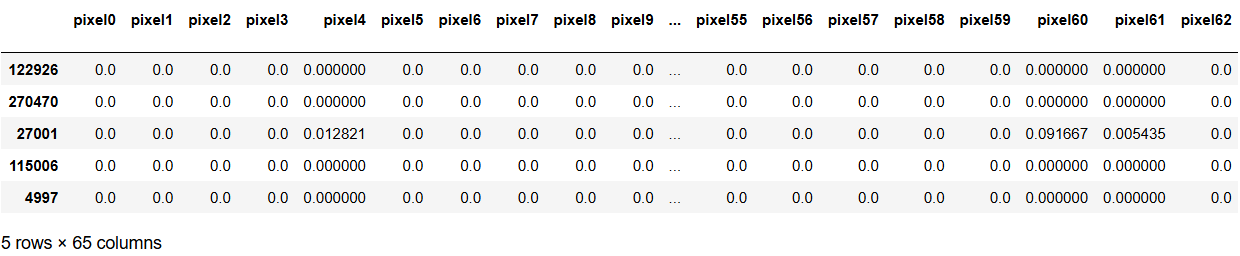
### ****5.2 Data Preparation for Training Data****

****The training data's numerical features are rescaled using MinMaxScaler, which normalizes values to a range between 0 and 1.The transformed features are stored in X\_train\_transformed, ensuring consistent scaling across the dataset for improved model performance.

### ****5.3 Data Prepration for Test Data****

The test data is rescaled using the same MinMaxScaler fitted on the training data, ensuring consistent feature scaling. The transformed test data is stored in X\_test\_transformed, ready for evaluation with the same scaling applied to the training data.





**6. Model Development**

### 6.1 Model Selection

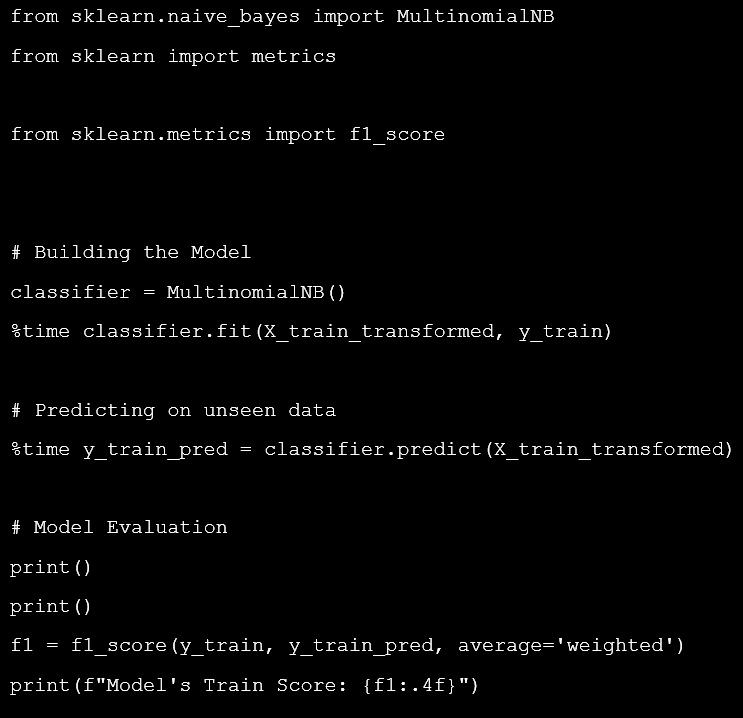
In this task, we evaluate several machine learning classifiers, including Logistic Regression, Support Vector Machines (SVM), Random Forest, Convolutional Neural Networks (CNN), Decision Tree, Naive Bayes, and XGBoost, to determine their suitability for recognizing handwritten alphabets.

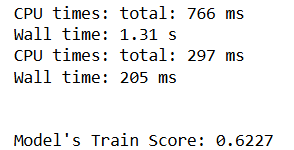
### 6.2 Model Training

The selected models are trained using the preprocessed training dataset. The training process involves multiple iterations to improve the model's performance. While techniques such as cross-validation and hyperparameter tuning can be used to optimize the models, in this particular task, they are not necessary because the train and test scores indicate good performance without additional tuning. However, if the training and testing scores were not satisfactory, we would consider the optimization techniques for each model

#### **Naive Bayes Model**

The Naive Bayes model is initialized and trained on the preprocessed training data. The training time is approximately 1.31 seconds . Predictions are made on the training data, with an F1 score of 0.6227.





#### **Logistic Regression Model**

The Logistic Regression model is initialized and trained on the preprocessed training data. The training time is approximately and 38.1 seconds . Predictions are made on the training data, with an F1 score of 0.8626.

### 

### 

#### **Decision Tree Model**

The Decision Tree model is trained on the preprocessed training data, with a training time of approximately 23.7 seconds. Predictions are made on the training data, resulting in a perfect F1 score of 1.0000.

### 

### 

### 

#### **RandomForest Model Training**

The Random Forest model is trained on the preprocessed data, with a training time of approximately 3 minutes and 54 seconds . Predictions are made on the training data, achieving an F1 score of 1.0000.

### 

### 

#### **XGBoost Model**

The XGBoost model is trained on the preprocessed data. The training time is approximately 141.8559 seconds. Predictions on the training data result in an F1 score of 0.9998.

### 

### 

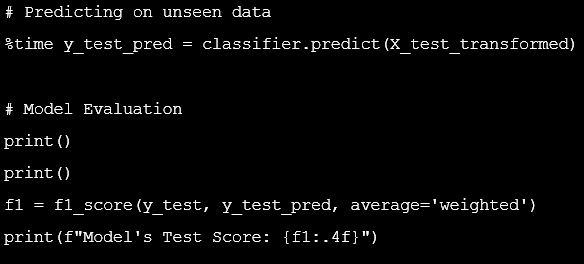
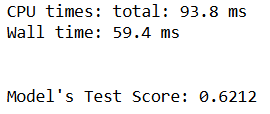
### 

### 6.3 Model Prediction and Evaluation on Unseen Data

In this section, we will use the trained models to predict the labels on the unseen test data. We will then compute the F1 score to evaluate the performance of each model on this test data. The F1 score is a suitable metric for this task due to the imbalanced nature of the dataset.

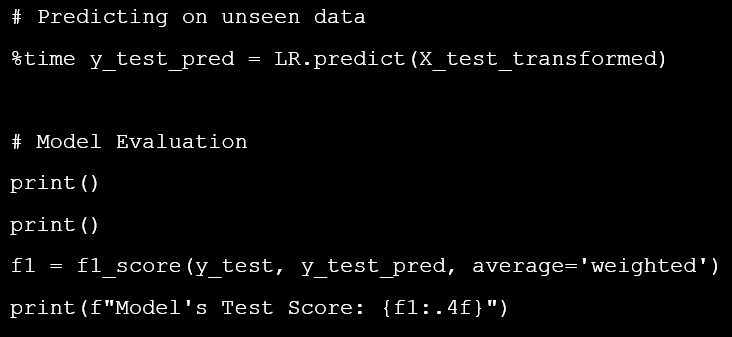
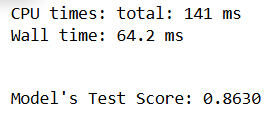
#### **Navie Bayes**

The Naive Bayes model predicts labels on the test data with a time of 0.0594 seconds. The F1 score on the test set is 0.6212, indicating its performance.



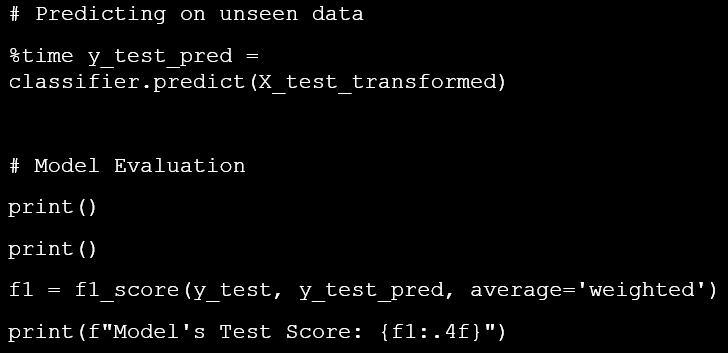
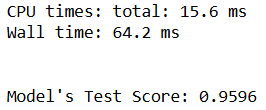
#### **Logistic Regression**

The Logistic Regression model is used to predict the labels on the unseen test data, with a time of 0.0642 seconds. The F1 score on the test set is 0.8630, reflecting its performance.



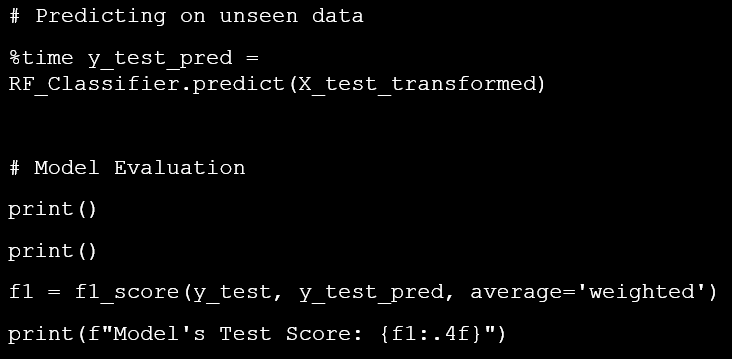
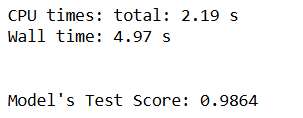
#### **Decision Tree**

The Decision Tree model is used to predict the labels on the unseen test data, taking a time of 0.0642 seconds. The F1 score on the test set is 0.9596, indicating strong performance.



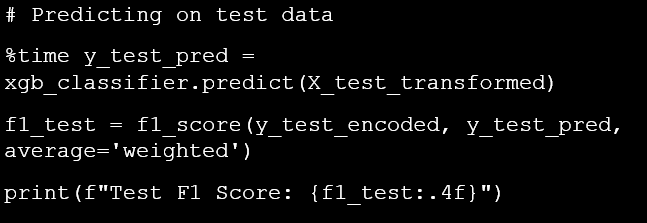
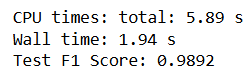
#### **Random Forest**

The Random Forest model is used to predict the labels on the unseen test data, taking a total time of 4.97 seconds. The F1 score on the test set is 0.9864, demonstrating high performance.



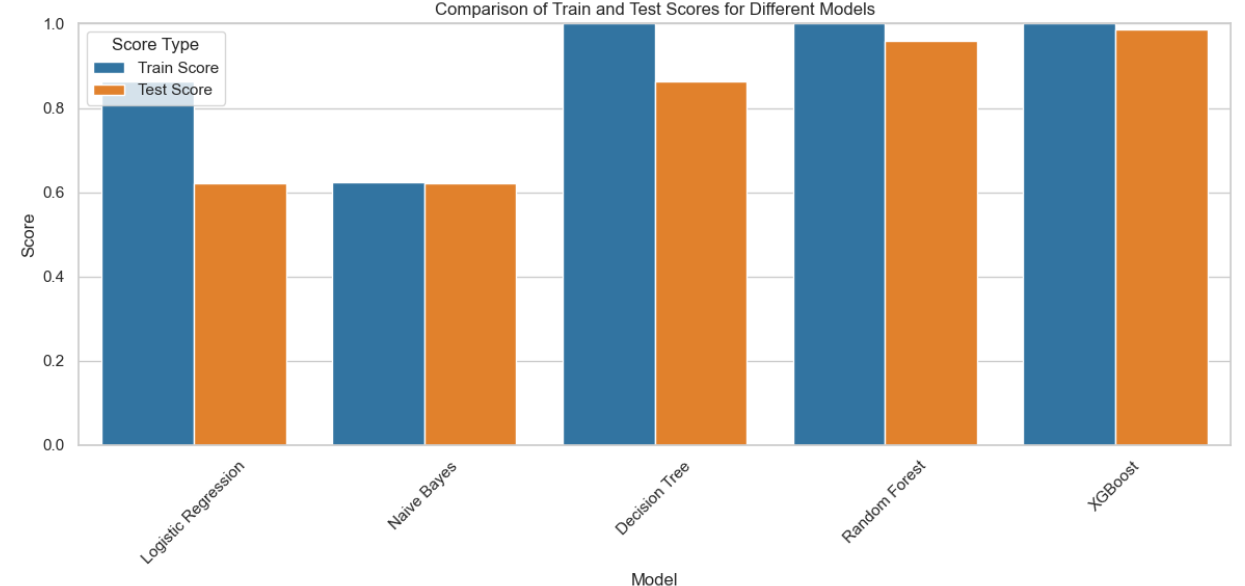
#### **Xgboost**

The XGBoost model is used to predict the labels on the unseen test data, taking a total time of 1.94 seconds. The F1 score on the test set is 0.9892, indicating strong performance



### 6.4 Comparison and Optimization

To resolve issues of underfitting and overfitting, we compare the training and test scores of different models. By analyzing discrepancies, we identify if a model is overfitting (high train score but low test score) or underfitting (low scores on both). The best model is selected if its train and test scores are equal or if the test score is higher, indicating balanced performance and good generalization to unseen data.

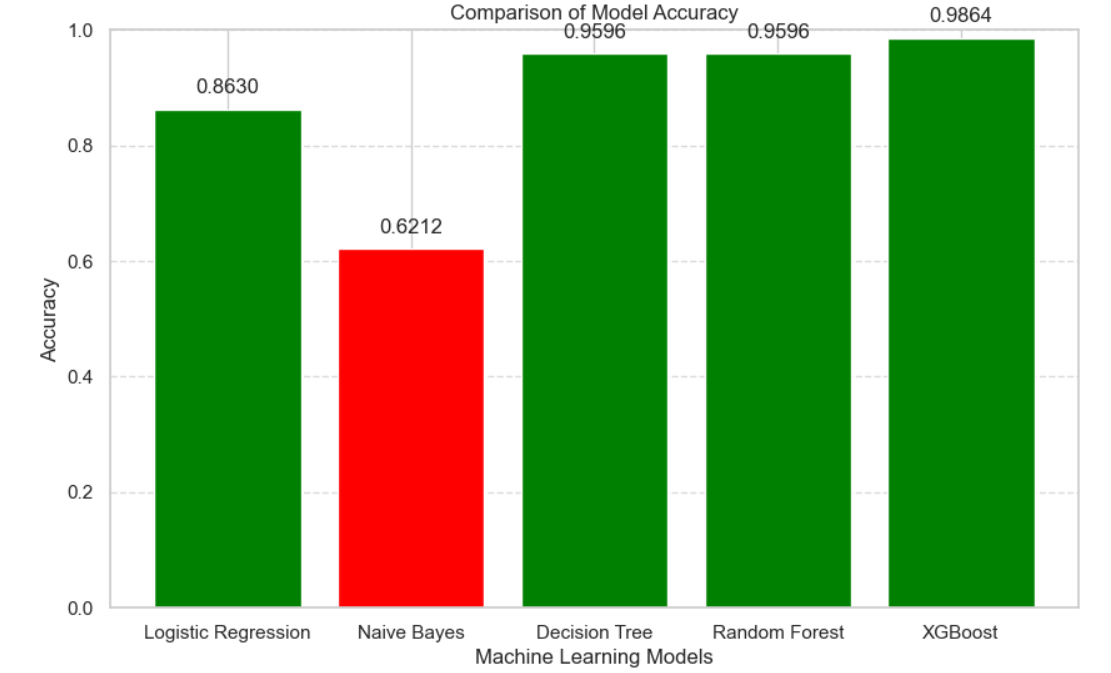


* **Logistic Regression**: This model shows a significant drop from train to test score, indicating potential overfitting or issues with generalization.
* **Naive Bayes**: The scores are very close, suggesting the model performs consistently across both datasets and is a good fit.
* **Decision Tree:** The high train score and slightly lower test score indicate overfitting, as the model performs exceptionally well on training data but slightly less on unseen data.
* **Random Forest**: Similar to the Decision Tree, this model also shows some overfitting, but it performs well on both training and test data.
* **XGBoost**: This model has high scores on both train and test datasets with minimal difference, indicating strong generalization and balanced performance.

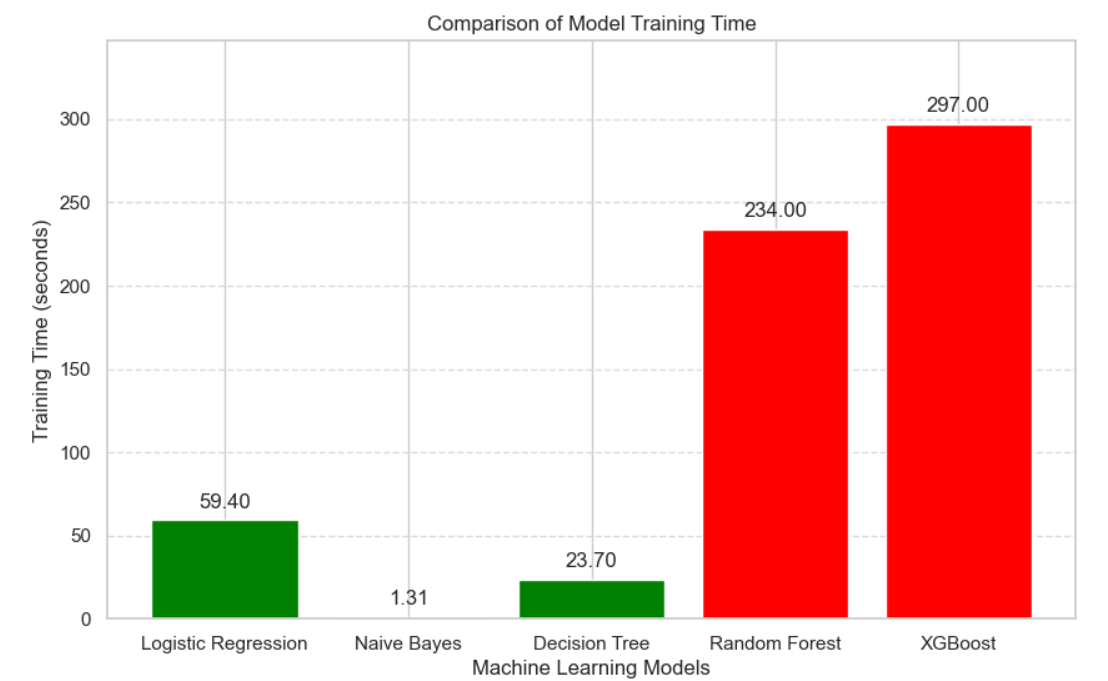
**Random Forest** and **Decision Tree** display overfitting, as their training scores are much higher than their test scores. **Logistic Regression**, **Naive Bayes**, and **XGBoost** demonstrate a better fit with balanced performance across both datasets.

## 7. Comparision Of Model’s Accuracy

The bar plot compares the accuracy of different machine learning models, with **Naive Bayes excluded** due to its **lower accuracy**. Models with accuracies **below** **0.80 are shown in red**, while those at or **above 0.80 are in green**.

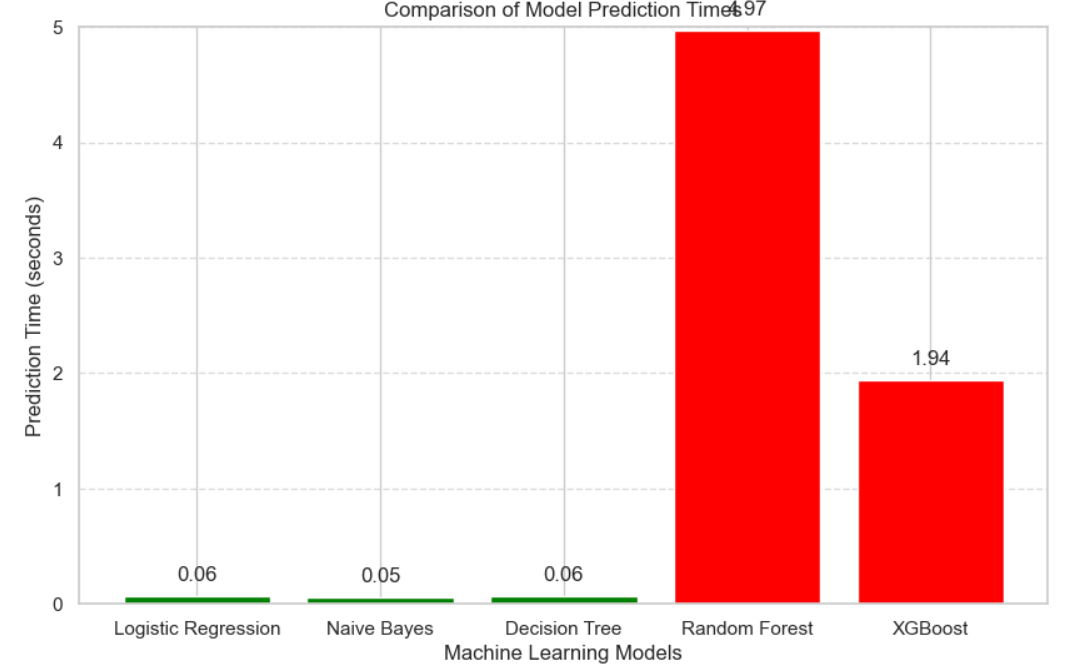


## 8. Model Training Time for Deployment



Random Forest and XGBoost demonstrate high accuracy but have longer training times, which might not be ideal for deployment if quick training is required. For clients needing models with shorter training times, it's better to choose models with lower training times, such as **Logistic Regression and Decision Tree**

## 9. Model Latency Comparison

**Naive Bayes** and **Logistic Regression** have the shortest prediction times (around 0.05 to 0.06 seconds), making them ideal for applications requiring quick responses. **Decision Tree** also has a low prediction time (0.0642 seconds). In contrast, **XGBoost** and **Random Forest** have significantly higher prediction times (1.94 and 4.97 seconds, respectively). For deployment scenarios where minimizing latency is crucial, models with lower prediction times, such as Naive Bayes and Logistic Regression, are more suitable.

## Conclusion

In conclusion, models like Decision Tree and Random Forest suffer from overfitting, and both Random Forest and XGBoost have high training and prediction times. Naive Bayes is not suitable due to its low accuracy**. Logistic Regression** stands out as the best fit, offering a balanced combination of high accuracy and minimal latency, making it the optimal choice for deployment.