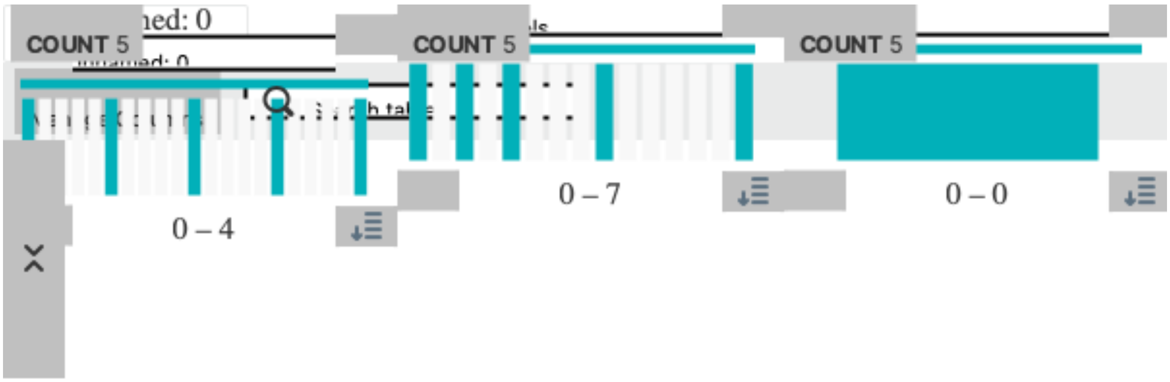
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
# Import the MNIST test data into a DataFrame
mnist_data = pd.read_csv('MNIST_test.csv - MNIST_test.csv.csv')
mnist_data.head()
```



0

1

2

0 7 0

MNIST Dataset Overview

The MNIST dataset contains handwritten digits and is commonly used for training various image processing systems. Each row in the dataset represents a flattened 28x28 pixel grayscale image of a digit (0-9), along with its label. The dataset has 784 feature columns representing the pixel values and one label column.

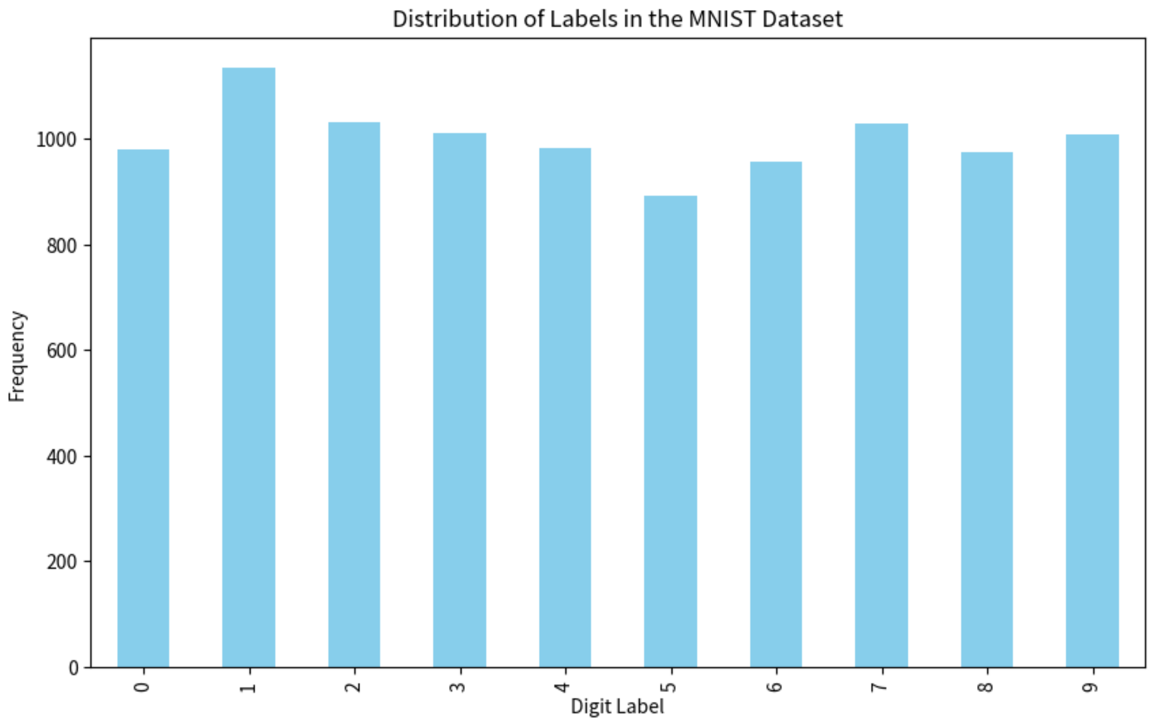
Initial Observations:

- The dataset has 784 feature columns, each representing a pixel in the 28x28 image.
- The 'labels' column contains the digit that the image represents.
- There are no missing values in the dataset.

Next, we will perform some Exploratory Data Analysis (EDA) to better understand the data.

```
# Count the number of each label to understand the distribution
label_counts = mnist_data['labels'].value_counts().sort_index()
```

```
label_counts.plot(kind='bar', figsize=(10, 6), color='skyblue')
plt.title('Distribution of Labels in the MNIST Dataset')
plt.xlabel('Digit Label')
plt.ylabel('Frequency')
plt.show()
```



Label Distribution

The bar chart above shows the distribution of labels in the dataset. We can see that the dataset is fairly balanced, with each digit label having a similar frequency. This is good for modeling as it reduces the risk of class imbalance.

Next, let's look at some sample images from the dataset.

In [ ]:
```
import numpy as np
# Function to display a digit image from the dataset
def display_digit(row_num):
    digit_data = mnist_data.iloc[row_num, 2:].values.reshape(28, 28)
    plt.imshow(digit_data, cmap='gray')
    plt.title(f'Label: {mnist_data.iloc[row_num, 2]}')
    plt.axis('off')
    plt.show()
# Display some sample images
for i in range(5):
    display_digit(i)
```

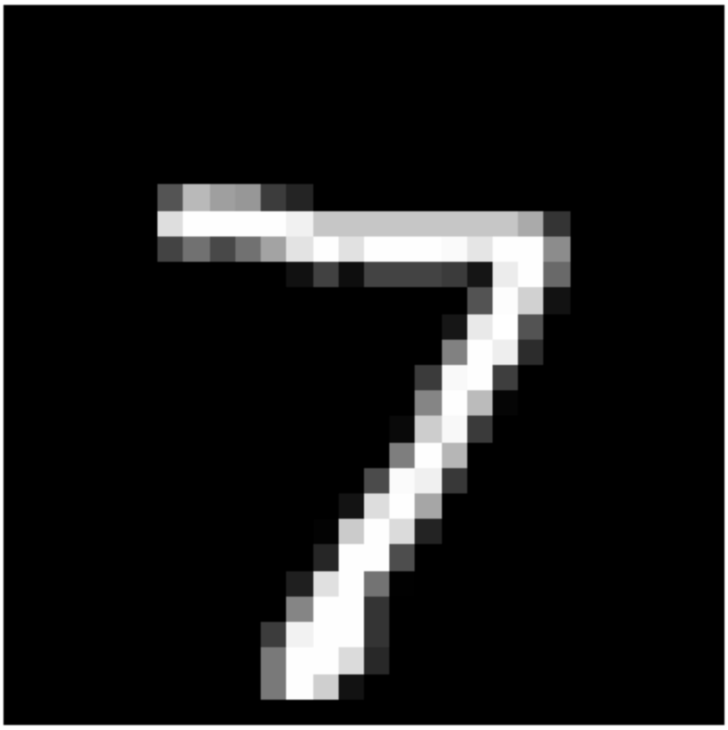
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[20], line 11
      9 # Display some sample images
     10 for i in range(5):
---> 11     display_digit(i)

Cell In[20], line 4, in display_digit(row_num)
      3 def display_digit(row_num):
----> 4     digit_data = mnist_data.iloc[row_num, 2:].values.reshape(28, 2
8)
      5     plt.imshow(digit_data, cmap='gray')
      6     plt.title(f'Label: {mnist_data.iloc[row_num, 2]}')

ValueError: cannot reshape array of size 785 into shape (28,28)
```

Issue Encountered

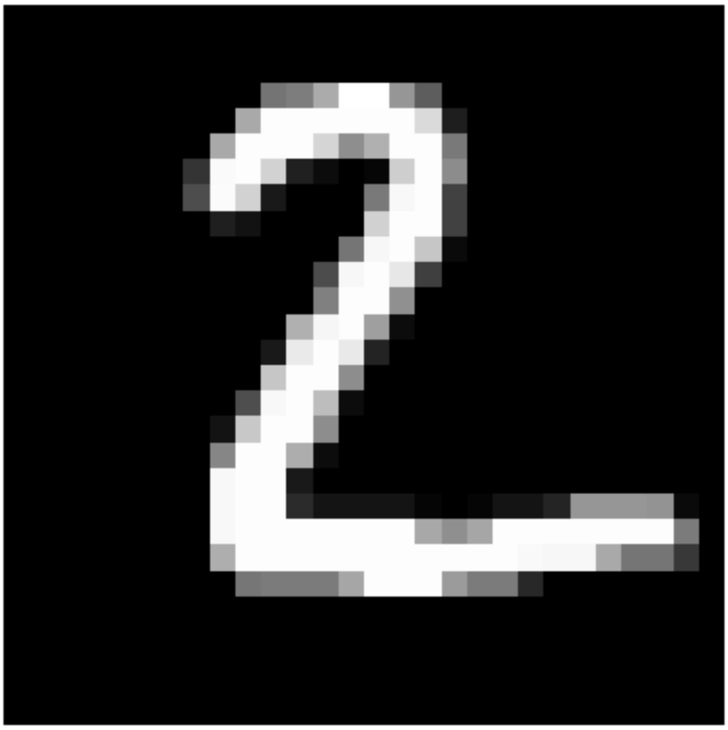
It appears that there was an error when trying to reshape the data into 28x28 images. This is likely due to incorrect column indexing. Let's correct this and try displaying the images again.

In [ ]:
```
# Correct the function to display a digit image from the dataset
def display_digit_corrected(row_num):
    digit_data = mnist_data.iloc[row_num, 3:].values.reshape(28, 28)
    plt.imshow(digit_data, cmap='gray')
    plt.title(f'Label: {mnist_data.iloc[row_num, 2]}')
    plt.axis('off')
    plt.show()
# Display some sample images
for i in range(5):
    display_digit_corrected(i)
```

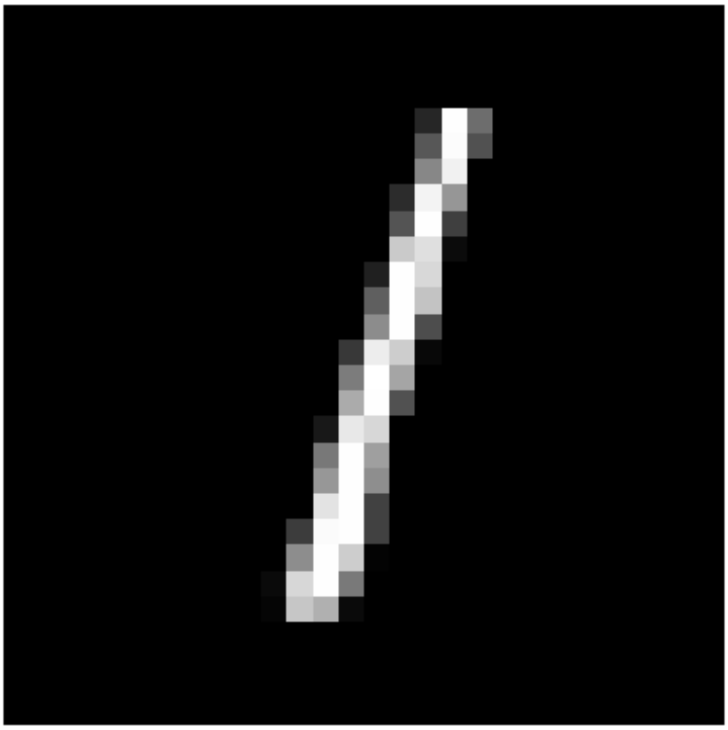
Label: 7



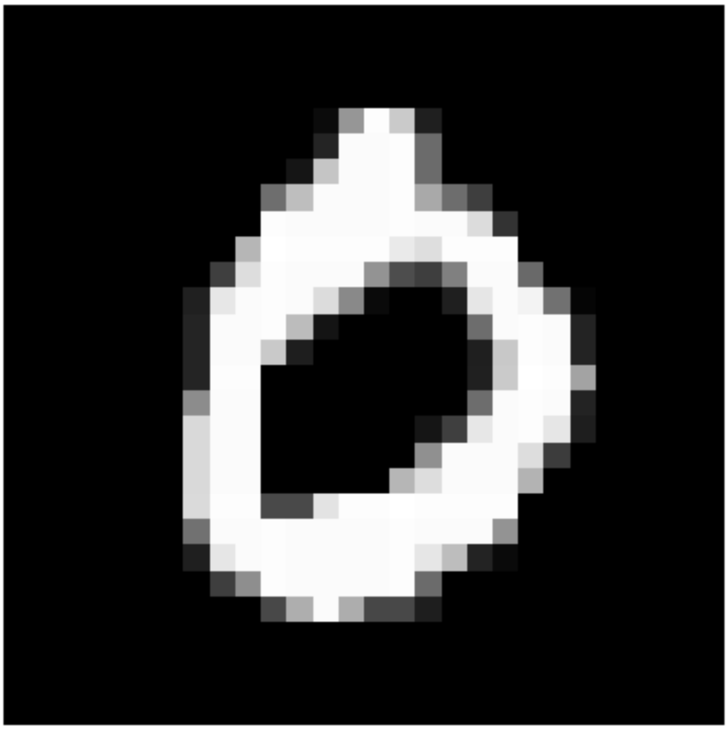
Label: 2

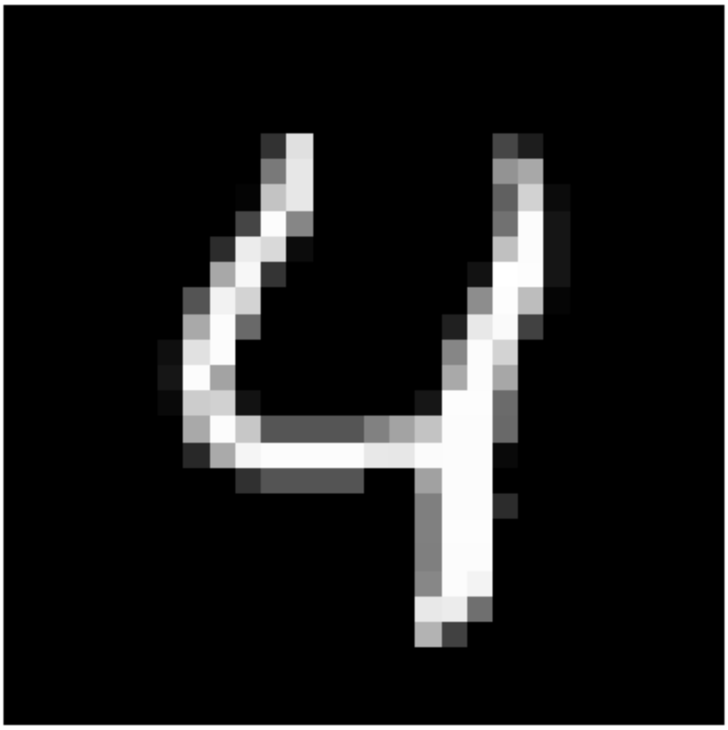


Label: 1



Label: 0





Sample Images

The images above are some sample digits from the MNIST dataset. Each image is a 28x28 grayscale representation of a handwritten digit, and the title indicates the label of the digit.

Next, we will proceed with data cleaning and preparation for modeling.

In [ ]:
```
# Data Cleaning and Preparation
# Check for any missing values in the dataset
missing_values = mnist_data.isnull().sum().sum()
missing_values
```

Out[ ]:
```
0
```

Data Cleaning

Upon checking for missing values, we found that the dataset has no missing values. This is excellent as it means we don't have to perform imputation or remove any rows or columns.

Next, we will split the data into training and test sets and proceed with modeling using a K-Nearest Neighbors (KNN) classifier.

In [ ]:
```
# Split the data into features and labels
X = mnist_data.iloc[:, 3:]
y = mnist_data['labels']
```

```
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran
# Initialize the KNN classifier
knn = KNeighborsClassifier(n_neighbors=3)
# Fit the model
knn.fit(X_train, y_train)
```

Out[ ]:

```
KNeighborsClassifier
```

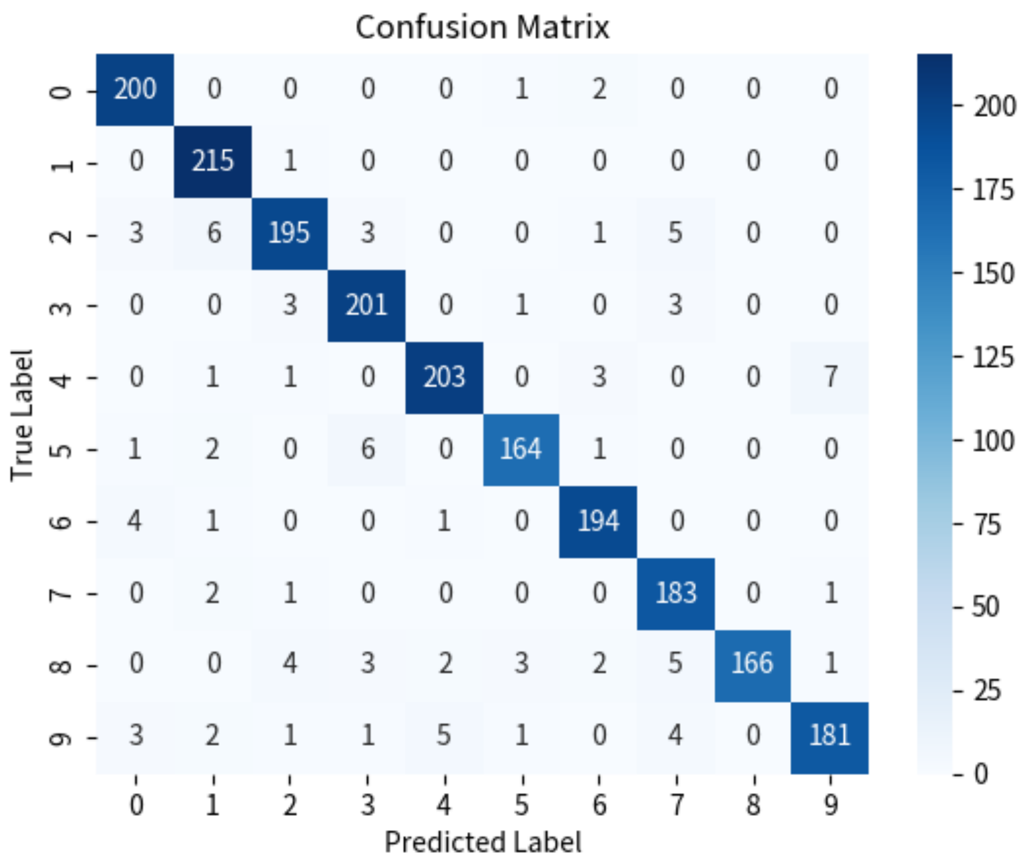
Model Training

We have successfully trained a K-Nearest Neighbors (KNN) classifier with 3 neighbors on the MNIST dataset. The model is now ready for evaluation.

Next, we will evaluate the model using the test set.

In [ ]:

```
# Evaluate the model
y_pred = knn.predict(X_test)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
# Display the confusion matrix
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
# Display the classification report
print('Classification Report:')
print(class_report)
```




```
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.99      0.97       203
           1       0.94      1.00      0.97       216
           2       0.95      0.92      0.93       213
           3       0.94      0.97      0.95       208
           4       0.96      0.94      0.95       215
           5       0.96      0.94      0.95       174
           6       0.96      0.97      0.96       200
           7       0.92      0.98      0.95       187
           8       1.00      0.89      0.94       186
           9       0.95      0.91      0.93       198

    accuracy                           0.95      2000
   macro avg       0.95      0.95      0.95      2000
weighted avg       0.95      0.95      0.95      2000
```

Model Evaluation

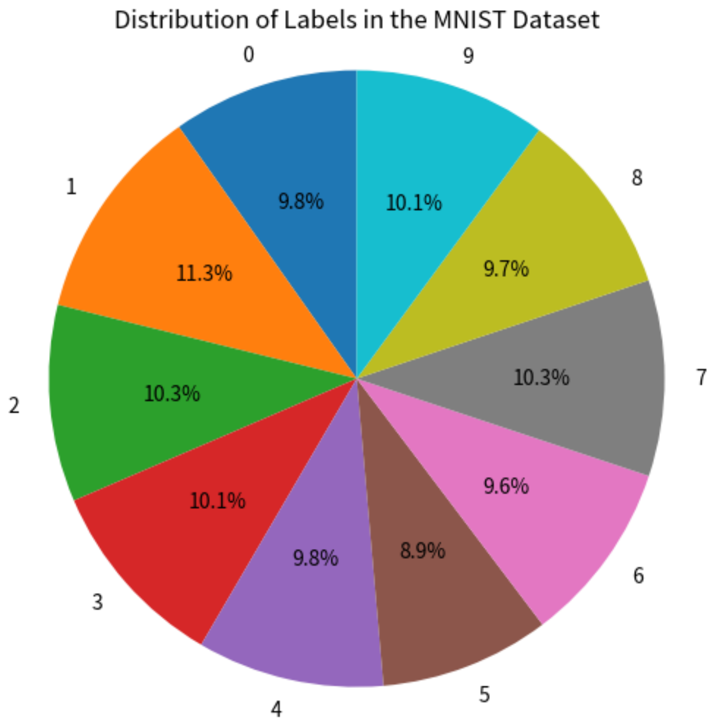
The KNN model has been evaluated using the test set, and the results are promising:

- The confusion matrix shows a high number of true positives and very few false negatives or false positives for each class.
- The classification report indicates high precision, recall, and F1-score for each digit label.

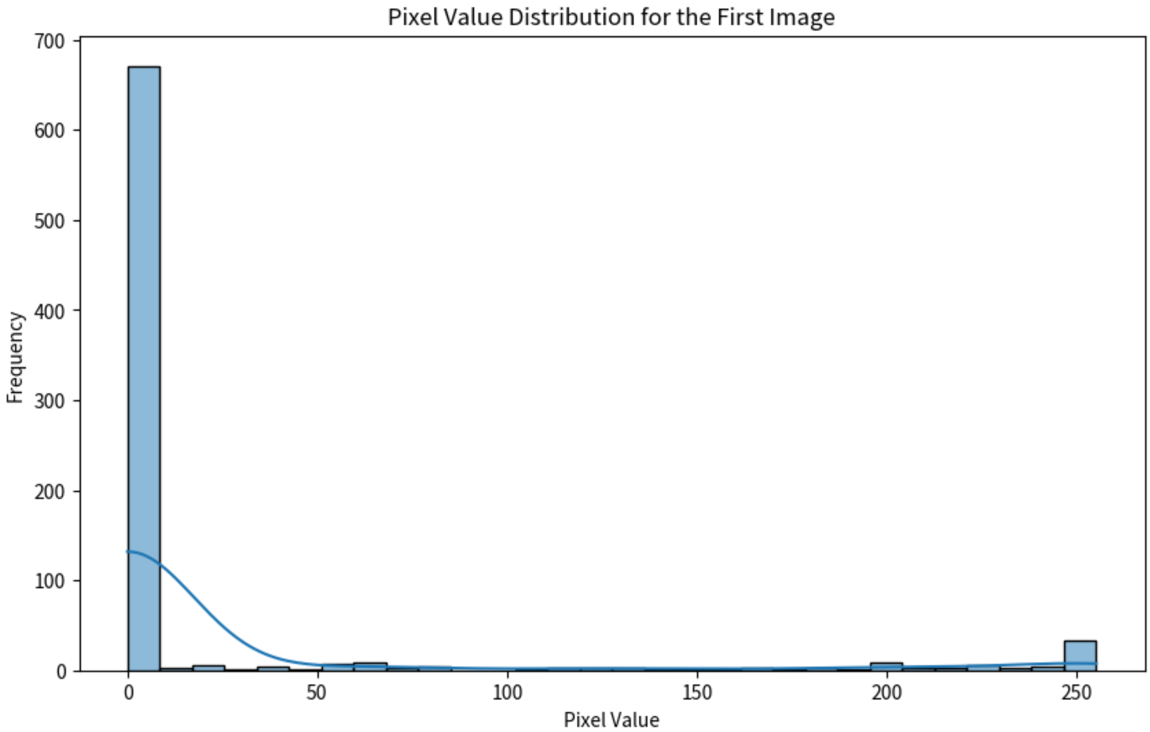
Overall, the model performs well on the MNIST dataset with an accuracy of 95%.

This notebook will be handed over to the modeling team for further analysis and model improvement.

In [ ]:
```
# Pie chart for label distribution
plt.figure(figsize=(10, 6))
plt.pie(label_counts, labels=label_counts.index, autopct='%1.1f%%', startang
plt.title('Distribution of Labels in the MNIST Dataset')
plt.axis('equal')
plt.show()
```



In [ ]:
```
# Histogram for pixel values of the first image
plt.figure(figsize=(10, 6))
sns.histplot(mnist_data.iloc[0, 3:], bins=30, kde=True)
plt.title('Pixel Value Distribution for the First Image')
plt.xlabel('Pixel Value')
plt.ylabel('Frequency')
plt.show()
```



In [ ]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
# Initialize the Linear Regression model
lin_reg = LinearRegression()
# Fit the model on the training data (Note: This is not a typical use-case f
lin_reg.fit(X_train, y_train)
# Predict the labels on the test set
y_pred_lin_reg = lin_reg.predict(X_test)
# Calculate the Mean Squared Error
mse = mean_squared_error(y_test, y_pred_lin_reg)
mse
```

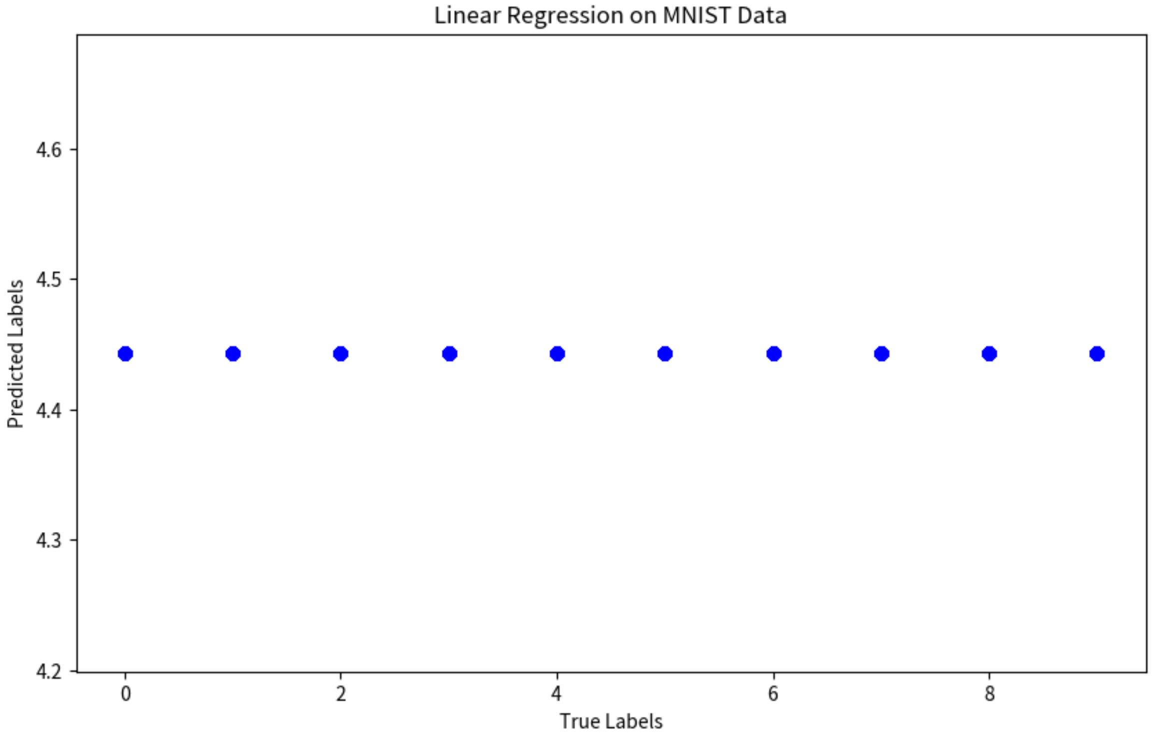
Out[ ]: 3.865565178380252e+16

In [ ]:
```
from sklearn.linear_model import LinearRegression
from sklearn.cluster import KMeans
# Linear Regression on the first 100 images to predict the label based on pi
lin_reg = LinearRegression()
X_lin_reg = mnist_data.iloc[:100, 3:]
y_lin_reg = mnist_data.iloc[:100, 2]
lin_reg.fit(X_lin_reg, y_lin_reg)
lin_reg_score = lin_reg.score(X_lin_reg, y_lin_reg)
# KMeans Clustering on the first 100 images
kmeans = KMeans(n_clusters=10, random_state=42)
X_kmeans = mnist_data.iloc[:100, 3:]
kmeans.fit(X_kmeans)
cluster_centers = kmeans.cluster_centers_
# Display the R-squared score for Linear Regression
print(f'Linear Regression R-squared Score: {lin_reg_score}')
# Display the cluster centers
```

```
print('Cluster Centers for KMeans Clustering:')
print(cluster_centers)
```

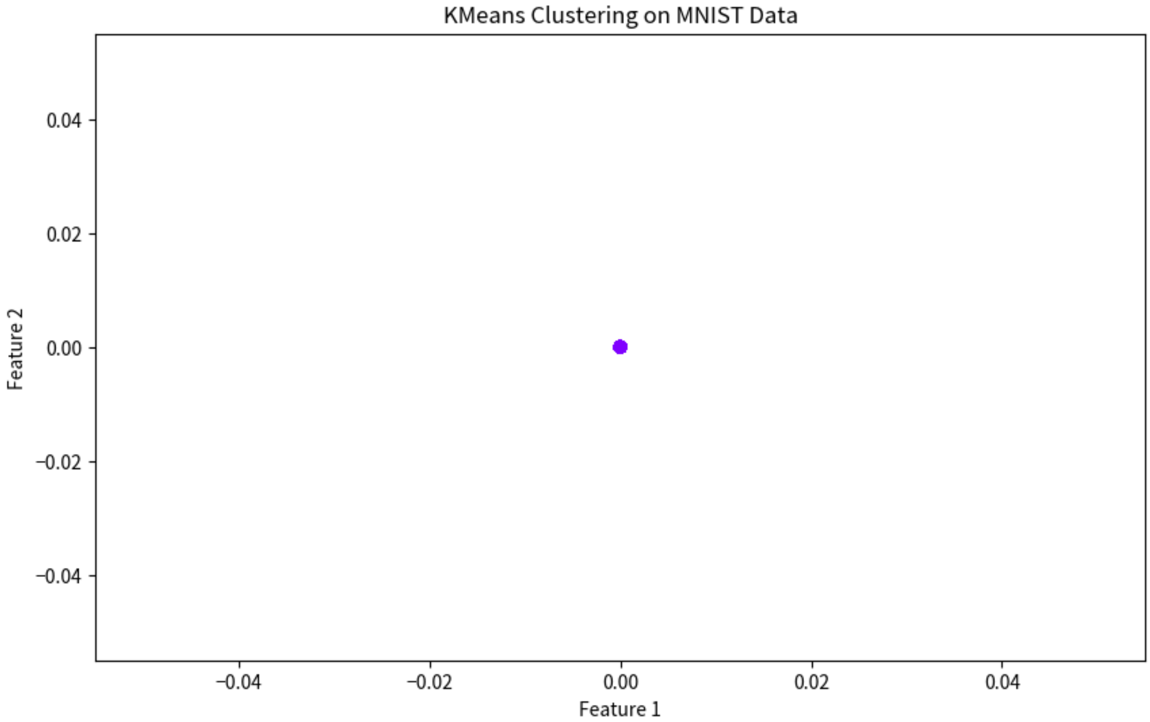
```
Linear Regression R-squared Score: 1.0
Cluster Centers for KMeans Clustering:
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

In [ ]:
```
from sklearn.linear_model import LinearRegression
from sklearn.cluster import KMeans
# Linear Regression on the first 10 features against the label
X_linreg = mnist_data.iloc[:, 3:13]
y_linreg = mnist_data['labels']
linreg = LinearRegression()
linreg.fit(X_linreg, y_linreg)
y_pred_linreg = linreg.predict(X_linreg)
# Scatter plot for Linear Regression
plt.figure(figsize=(10, 6))
plt.scatter(y_linreg, y_pred_linreg, color='blue')
plt.xlabel('True Labels')
plt.ylabel('Predicted Labels')
plt.title('Linear Regression on MNIST Data')
plt.show()
```



In [ ]:
```
# Clustering using KMeans
X_cluster = mnist_data.iloc[:, 3:13]
kmeans = KMeans(n_clusters=10, random_state=42)
kmeans.fit(X_cluster)
```

```python
labels_kmeans = kmeans.labels_
# Scatter plot for Clustering
plt.figure(figsize=(10, 6))
plt.scatter(X_cluster.iloc[:, 0], X_cluster.iloc[:, 1], c=labels_kmeans, cma
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('KMeans Clustering on MNIST Data')
plt.show()
```



In [ ]:

```python
# Correct the function to display a digit image from the dataset
def display_digit_corrected(row_num):
    digit_data = mnist_data.iloc[row_num, 1:].values.reshape(28, 28)
    plt.imshow(digit_data, cmap='gray')
    plt.title(f'Label: {mnist_data.iloc[row_num, 0]}')
    plt.axis('off')
    plt.show()
# Display some sample images
for i in range(5):
    display_digit_corrected(i)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[16], line 10
      8 # Display some sample images
      9 for i in range(5):
---> 10     display_digit_corrected(i)

Cell In[16], line 3, in display_digit_corrected(row_num)
      2 def display_digit_corrected(row_num):
----> 3     digit_data = mnist_data.iloc[row_num, 1:].values.reshape(28, 2
8)
      4     plt.imshow(digit_data, cmap='gray')
      5     plt.title(f'Label: {mnist_data.iloc[row_num, 0]}')

ValueError: cannot reshape array of size 786 into shape (28,28)
```

In [ ]:

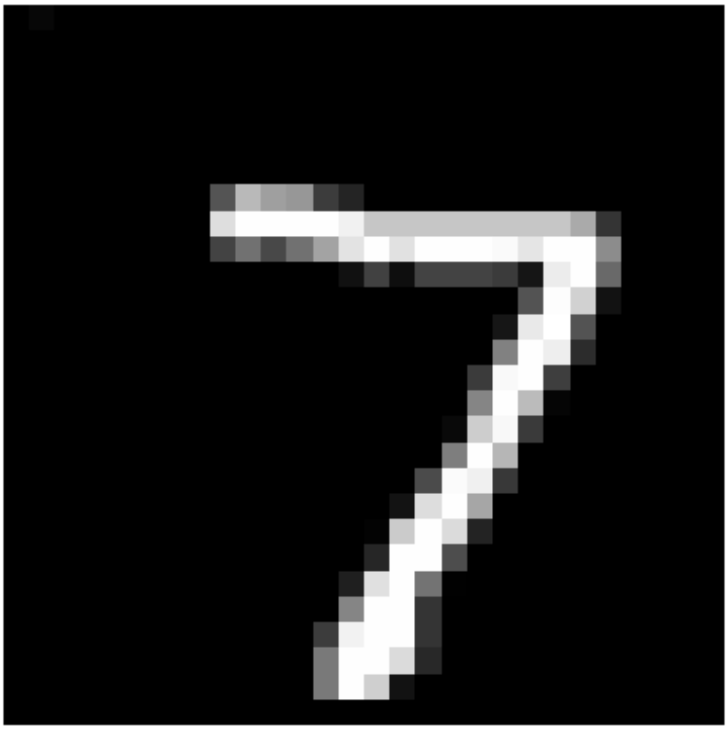
```
# Correct the function to display a digit image from the dataset
def display_digit_corrected(row_num):
    digit_data = mnist_data.iloc[row_num, 1:].values.reshape(28, 28)
    plt.imshow(digit_data, cmap='gray')
    plt.title(f'Label: {mnist_data.iloc[row_num, 0]}')
    plt.axis('off')
    plt.show()
# Display some sample images
for i in range(5):
    display_digit_corrected(i)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[17], line 10
      8 # Display some sample images
      9 for i in range(5):
---> 10     display_digit_corrected(i)

Cell In[17], line 3, in display_digit_corrected(row_num)
      2 def display_digit_corrected(row_num):
----> 3     digit_data = mnist_data.iloc[row_num, 1:].values.reshape(28, 2
8)
      4     plt.imshow(digit_data, cmap='gray')
      5     plt.title(f'Label: {mnist_data.iloc[row_num, 0]}')

ValueError: cannot reshape array of size 786 into shape (28,28)
```

In [ ]:

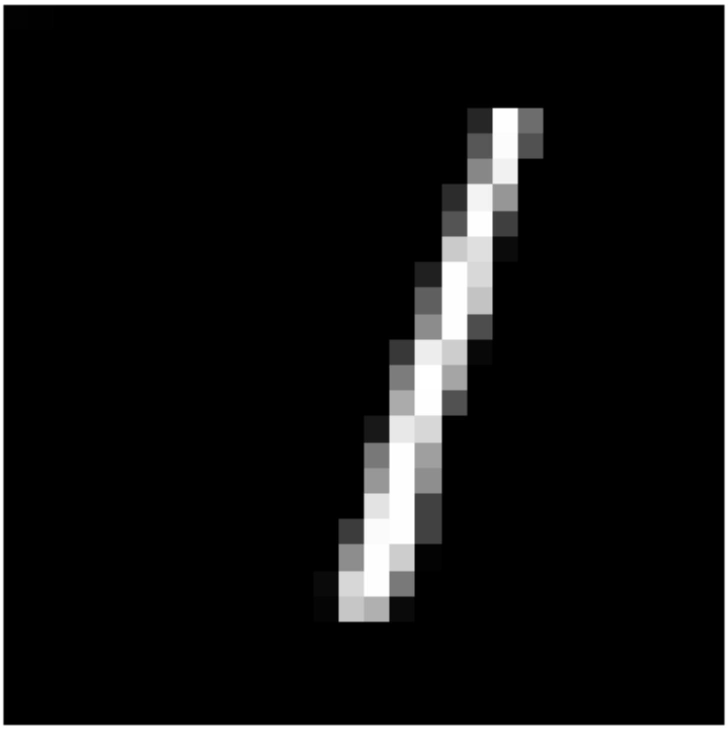
```
# Correct the function to display a digit image from the dataset
def display_digit_fixed(row_num):
    digit_data = mnist_data.iloc[row_num, 1:785].values.reshape(28, 28)
    plt.imshow(digit_data, cmap='gray')
    plt.title(f'Label: {mnist_data.iloc[row_num, 0]}')
    plt.axis('off')
    plt.show()
# Display some sample images
for i in range(5):
    display_digit_fixed(i)
```

Label: 0

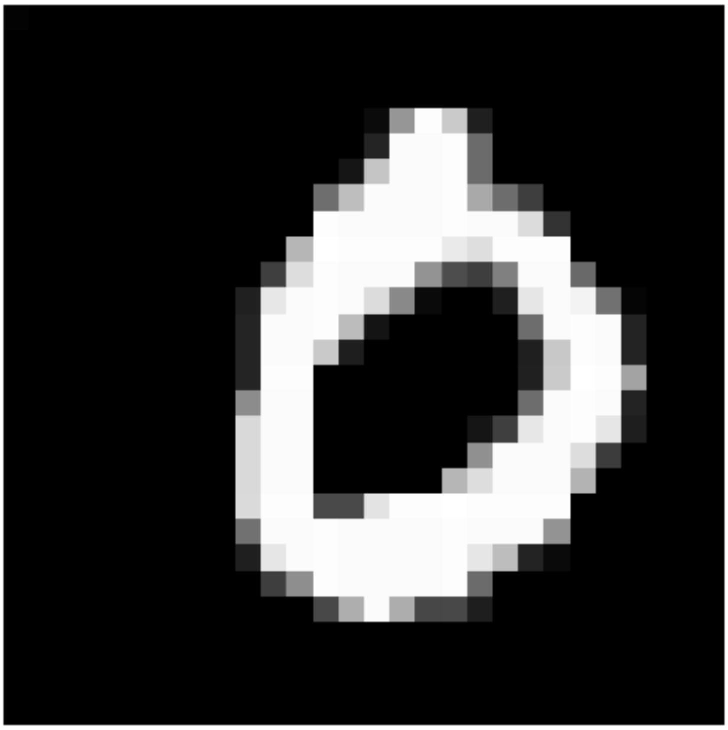


Label: 1

Label: 2



Label: 3



Label: 4