Mini Project: MNIST Digit Visualizer and Analyzer

This mini project demonstrates how to use **NumPy**, **Pandas**, and **Matplotlib** to:

1. Load and explore MNIST dataset (`mnist_train_small.csv`)
2. Perform basic statistics using Pandas
3. Visualize digits using NumPy reshaping
4. Plot digit distributions with Matplotlib

🔷 Step 1: Import Libraries

- What: bring in NumPy, Pandas, and Matplotlib.
- Why: each library has a clear role — Pandas for loading and quick tabular ops, NumPy for numeric arrays and reshaping images, Matplotlib for plotting/visual checks. Good habits here keep the rest simple.
- How:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

🔷 Step 2: Load Dataset

- What: read the CSV into a Pandas DataFrame. In MNIST CSVs the first column is label (0–9), remaining 784 columns are pixels (28×28 = 784).
- Why: Pandas gives easy preview, summary stats, and grouping; loading correctly avoids headaches later (e.g., wrong header, wrong shapes).
- How:

```
# Load dataset
file_path = "/content/sample_data/mnist_train_small.csv"
data = pd.read_csv(file_path, header=None)

# First column = label (digit 0–9), remaining = pixel values (0–255)
print("Dataset shape:", data.shape)
data.head()
```

```
Dataset shape: (20000, 785)
```

	0	1	2	3	4	5	6	7	8	9	...	775	776	777	778	779	780	781	782	7
0	6	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
1	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
2	7	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
3	9	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
4	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	

5 rows × 785 columns

Practical checks:

- data.shape[1] should be 785 (1 label + 784 pixels).
- If memory is a concern: pd.read_csv(..., dtype=np.uint8) or nrows=2000 for a quick preview.
- If you see weird values, open the raw CSV to check separators/headers.

🔷 Step 3: Separate Features & Labels

- What: split DataFrame into labels (target) and pixels (feature matrix).
- Why: ML & visualization workflows treat features and targets separately; pixels must be a numeric NumPy array to reshape into images.
- How:

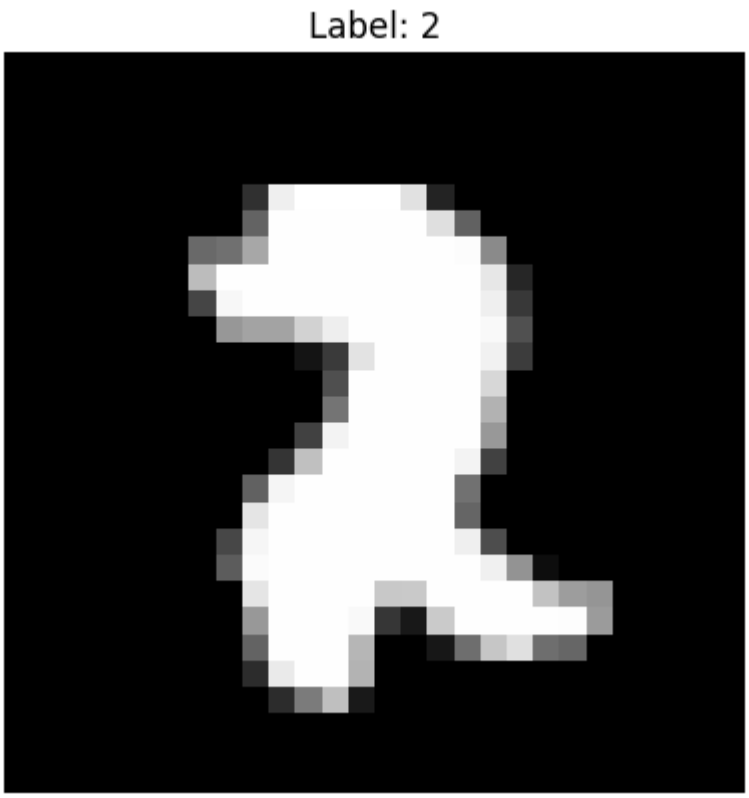
```
labels = data.iloc[:, 0]          # Digit labels
pixels = data.iloc[:, 1:].values  # Pixel values

print("Labels shape:", labels.shape)
print("Pixels shape:", pixels.shape)
```

```
Labels shape: (20000,)
Pixels shape: (20000, 784)
```

🔷 Step 4: Visualize a Digit

- What: reshape a row of 784 values into a 28×28 image and plot.
- Why: visual checks catch corrupted rows, orientation problems, or unexpected normalization. Also gives intuition about digit variability.
- How:

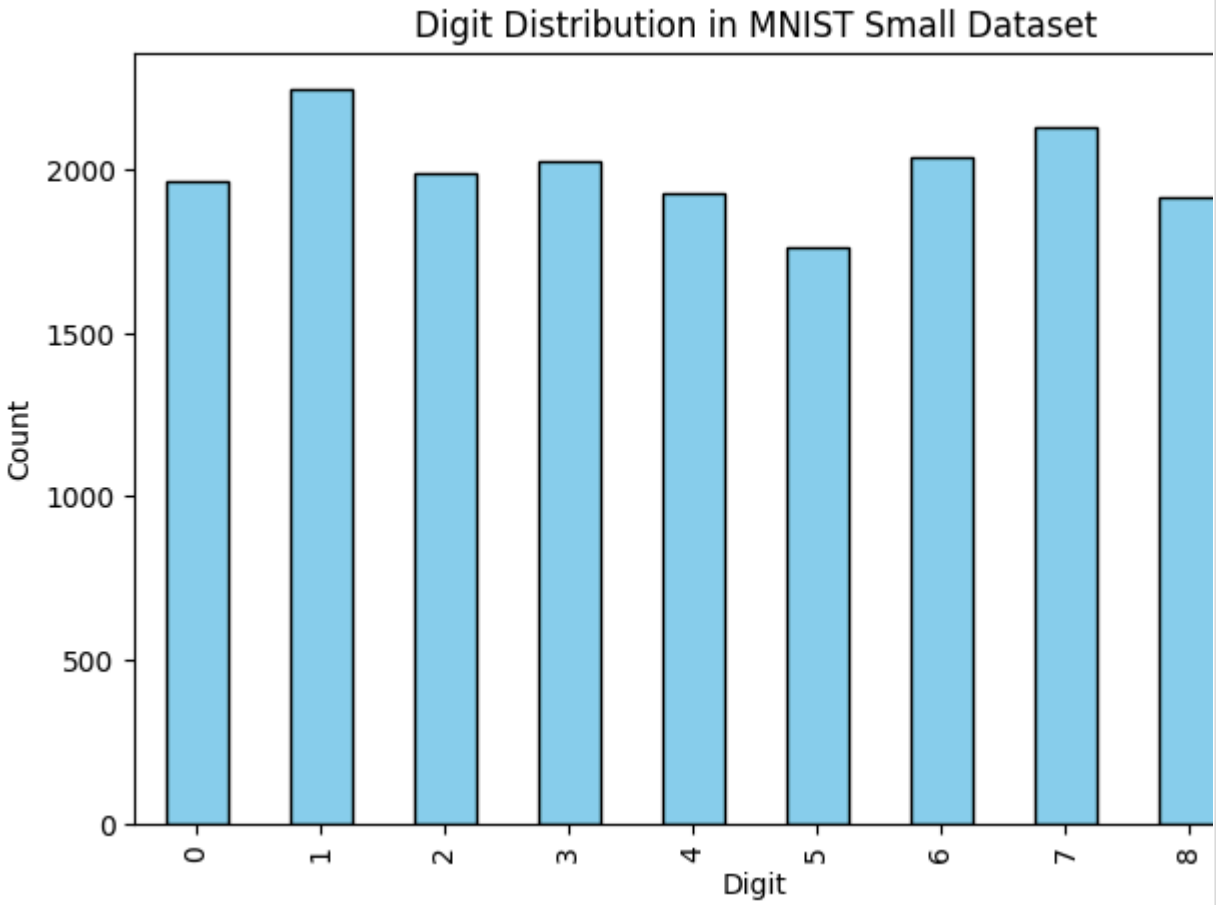
```
# Pick one sample
index = 10
digit = pixels[index].reshape(28, 28)  # MNIST is 28x28
plt.imshow(digit, cmap='gray')
plt.title(f"Label: {labels[index]}")
plt.axis("off")
plt.show()
```



🔷 Step 5: Plot Distribution of Digits

- What: count how many examples of each digit (0–9) and plot a bar chart.
- Why: class imbalance affects training. If some digits are rare, you may need augmentation, re-weighting, or stratified sampling.
- How:

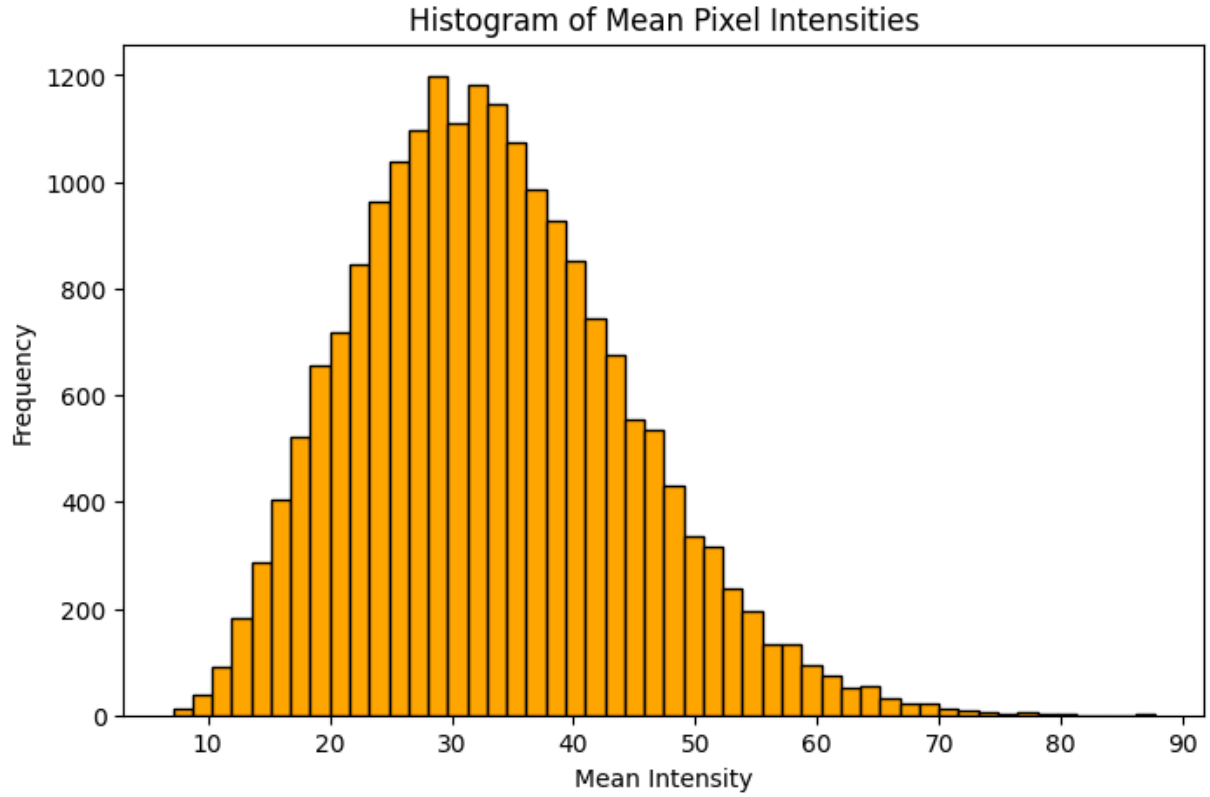
```
plt.figure(figsize=(8,5))
labels.value_counts().sort_index().plot(kind="bar", color="skyblue", edg
plt.title("Digit Distribution in MNIST Small Dataset")
plt.xlabel("Digit")
plt.ylabel("Count")
plt.show()
```



🔷 Step 6: Compute Statistics with Pandas + NumPy

- What: numeric summaries — mean pixel intensity per image, variance, and mean image across dataset.
- Why: these stats reveal dataset brightness, empty/near-empty images, and the "average stroke" shape (average image) which helps feature understanding.
- How:

```
# Mean pixel intensity per digit
mean_intensity = pd.DataFrame(pixels).mean(axis=1)

plt.figure(figsize=(8,5))
plt.hist(mean_intensity, bins=50, color="orange", edgecolor="black")
plt.title("Histogram of Mean Pixel Intensities")
plt.xlabel("Mean Intensity")
plt.ylabel("Frequency")
plt.show()
```



Interpretation:

- Very low mean intensity images might be blanks or near-black — inspect them.
- mean_image shows where pixels are typically dark/bright — useful for feature engineering.

```
## 🔷 Step 7: Compare Digits Side by Si

* What: pick a sample for each label (0
show them in a grid.
* Why: quick visual comparison helps yo
typical shapes and intra-class diversit
7s look like 9s, etc.). Good for spotti
labeling errors.
*How:
```

🔷 Step 7: Compare Digits Side by Side

- What: pick a sample for each label (0–9) and show them in a grid.
- Why: quick visual comparison helps you see typical shapes and intra-class diversity (some 7s look like 9s, etc.). Good for spotting labeling errors. *How:

```
    fig, axes = plt.subplots(2, 5, figsize=(10,5))
    for i in range(10):
```

```
    digit_img = pixels[labels[labels==i].index[0]].reshape(28,28)
    axes[i//5, i%5].imshow(digit_img, cmap="gray")
    axes[i//5, i%5].set_title(f"Digit {i}")
    axes[i//5, i%5].axis("off")

plt.tight_layout()
plt.show()
```