In [1]:
```python
import numpy as np
from sklearn.datasets import fetch_openml
```

In [2]:
```python
mnist = fetch_openml('mnist_784', version = 1, cache = True, as_frame = False)
print(mnist.DESCR)
```

```
C:\Users\armen\anaconda3\Lib\site-packages\sklearn\datasets\_openml.py:968: Futur
eWarning: The default value of `parser` will change from `'liac-arff'` to `'aut
o'` in 1.4. You can set `parser='auto'` to silence this warning. Therefore, an `I
mportError` will be raised from 1.4 if the dataset is dense and pandas is not ins
talled. Note that the pandas parser may return different data types. See the Note
s Section in fetch_openml's API doc for details.
  warn(
```

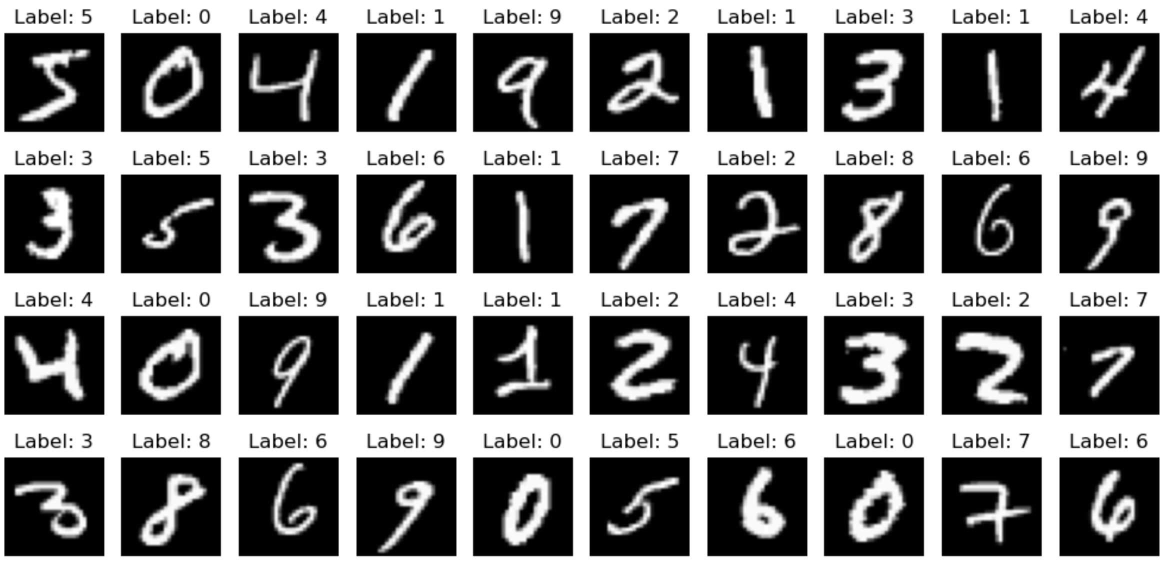
```
**Author**: Yann LeCun, Corinna Cortes, Christopher J.C. Burges
**Source**: [MNIST Website](http://yann.lecun.com/exdb/mnist/) - Date unknown
**Please cite**:

The MNIST database of handwritten digits with 784 features, raw data available a
t: http://yann.lecun.com/exdb/mnist/. It can be split in a training set of the fi
rst 60,000 examples, and a test set of 10,000 examples

It is a subset of a larger set available from NIST. The digits have been size-nor
malized and centered in a fixed-size image. It is a good database for people who
want to try learning techniques and pattern recognition methods on real-world dat
a while spending minimal efforts on preprocessing and formatting. The original bl
ack and white (bilevel) images from NIST were size normalized to fit in a 20x20 p
ixel box while preserving their aspect ratio. The resulting images contain grey l
evels as a result of the anti-aliasing technique used by the normalization algori
thm. the images were centered in a 28x28 image by computing the center of mass of
the pixels, and translating the image so as to position this point at the center
of the 28x28 field.

With some classification methods (particularly template-based methods, such as SV
M and K-nearest neighbors), the error rate improves when the digits are centered
by bounding box rather than center of mass. If you do this kind of pre-processin
g, you should report it in your publications. The MNIST database was constructed
from NIST's NIST originally designated SD-3 as their training set and SD-1 as the
ir test set. However, SD-3 is much cleaner and easier to recognize than SD-1. The
reason for this can be found on the fact that SD-3 was collected among Census Bur
eau employees, while SD-1 was collected among high-school students. Drawing sensi
ble conclusions from learning experiments requires that the result be independent
of the choice of training set and test among the complete set of samples. Therefo
re it was necessary to build a new database by mixing NIST's datasets.

The MNIST training set is composed of 30,000 patterns from SD-3 and 30,000 patter
ns from SD-1. Our test set was composed of 5,000 patterns from SD-3 and 5,000 pat
terns from SD-1. The 60,000 pattern training set contained examples from approxim
ately 250 writers. We made sure that the sets of writers of the training set and
test set were disjoint. SD-1 contains 58,527 digit images written by 500 differen
t writers. In contrast to SD-3, where blocks of data from each writer appeared in
sequence, the data in SD-1 is scrambled. Writer identities for SD-1 is available
and we used this information to unscramble the writers. We then split SD-1 in tw
o: characters written by the first 250 writers went into our new training set. Th
e remaining 250 writers were placed in our test set. Thus we had two sets with ne
arly 30,000 examples each. The new training set was completed with enough example
s from SD-3, starting at pattern # 0, to make a full set of 60,000 training patte
rns. Similarly, the new test set was completed with SD-3 examples starting at pat
tern # 35,000 to make a full set with 60,000 test patterns. Only a subset of 10,0
00 test images (5,000 from SD-1 and 5,000 from SD-3) is available on this site. T
he full 60,000 sample training set is available.

Downloaded from openml.org.
```

In [3]:

```python
#To check data structure first
import matplotlib.pyplot as plt
import pandas as pd

X, y = mnist["data"], mnist["target"]
print("Shape of X:", X.shape)
print("Shape of y:", y.shape)
```

```
Shape of X: (70000, 784)
Shape of y: (70000,)
```

In [4]:
```python
plt.figure(figsize=(10, 5))

for i in range(40):
    plt.subplot(4, 10, i+1)
    plt.imshow(X[i].reshape(28, 28), cmap='gray')
    plt.title('Label: ' + str(y[i]))
    plt.axis('off')

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



In [5]:
```python
print("Data type:", X.dtype)
```

```
Data type: float64
```

In [6]:
```python
print(pd.DataFrame(X).describe())
```

```
             0        1        2        3        4        5        6        7  \
count  70000.0  70000.0  70000.0  70000.0  70000.0  70000.0  70000.0  70000.0
mean       0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
std        0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
min        0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
25%        0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
50%        0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
75%        0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
max        0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0

             8        9  ...           774           775           776  \
count  70000.0  70000.0  ...  70000.000000  70000.000000  70000.000000
mean       0.0      0.0  ...      0.197414      0.099543      0.046629
std        0.0      0.0  ...      5.991206      4.256304      2.783732
min        0.0      0.0  ...      0.000000      0.000000      0.000000
25%        0.0      0.0  ...      0.000000      0.000000      0.000000
50%        0.0      0.0  ...      0.000000      0.000000      0.000000
75%        0.0      0.0  ...      0.000000      0.000000      0.000000
max        0.0      0.0  ...    254.000000    254.000000    253.000000

                777           778           779      780      781      782  \
count  70000.000000  70000.000000  70000.000000  70000.0  70000.0  70000.0
mean       0.016614      0.012957      0.001714      0.0      0.0      0.0
std        1.561822      1.553796      0.320889      0.0      0.0      0.0
min        0.000000      0.000000      0.000000      0.0      0.0      0.0
25%        0.000000      0.000000      0.000000      0.0      0.0      0.0
50%        0.000000      0.000000      0.000000      0.0      0.0      0.0
75%        0.000000      0.000000      0.000000      0.0      0.0      0.0
max      253.000000    254.000000     62.000000      0.0      0.0      0.0

           783
count  70000.0
mean       0.0
std        0.0
min        0.0
25%        0.0
50%        0.0
75%        0.0
max        0.0

[8 rows x 784 columns]
```

In [7]:

```python
# 导入必要的库
import numpy as np
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

In [8]:

```python
# Pre-Preprocess data
print("Preprocessing data...")
# 将像素值缩放到0到1之间
X = X / 255.0
# test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

print("Number of samples in training set:", X_train.shape[0])
```

```
print("Number of samples in test set:", X_test.shape[0])
print("Number of features:", X_train.shape[1])
```

```
Preprocessing data...
Number of samples in training set: 56000
Number of samples in test set: 14000
Number of features: 784
```

In [9]:
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# skape regression mode
print("Training logistic regression model...")
logistic_regression = LogisticRegression(max_iter=1000, random_state=42)
logistic_regression.fit(X_train_scaled, y_train)

train_accuracy_lr = logistic_regression.score(X_train_scaled, y_train)
test_accuracy_lr = logistic_regression.score(X_test_scaled, y_test)
print("Accuracy of logistic regression on training set:", train_accuracy_lr)
print("Accuracy of logistic regression on test set:", test_accuracy_lr)
```

```
Training logistic regression model...
Accuracy of logistic regression on training set: 0.945625
Accuracy of logistic regression on test set: 0.9164285714285715
```

In [10]:
```
# skapa och träning random forest model
print("Training random forest model...")
random_forest = RandomForestClassifier(n_estimators=100, random_state=42)
random_forest.fit(X_train, y_train)

train_accuracy_rf = random_forest.score(X_train, y_train)
test_accuracy_rf = random_forest.score(X_test, y_test)
print("Accuracy of random forest on training set:", train_accuracy_rf)
print("Accuracy of random forest on test set:", test_accuracy_rf)
```

```
Training random forest model...
Accuracy of random forest on training set: 1.0
Accuracy of random forest on test set: 0.9675
```

In [11]:
```
import joblib

joblib.dump(random_forest, 'C:/Users/armen/Downloads/EC-utbildning/2024-V.7-Mach

joblib.dump(logistic_regression, 'C:/Users/armen/Downloads/EC-utbildning/2024-V.
```

Out[11]:
```
['C:/Users/armen/Downloads/EC-utbildning/2024-V.7-Machine learning/Xiaowen_Chen
inlämningsuppgifter/logistic_regression_model(2).pkl']
```

In [ ]: