TO DESIGN A PROJECT FROM THE MNIST DATASET TO IDENTIFY DIGIT CLASSIFICATION.

PROBLEM STATEMENT:

The dataset is of 0-9 digits converted from image to input features, every row represents a specific digit image along with labels for every row. You can view the image using Matplotlib if you wish to.

Design a project from the MNIST dataset to identify digit classification using any of the classification algorithms.

What is Handwritten Digit Recognition?

The handwritten digit recognition is the ability of computers to recognize human handwritten digits. It is a hard task for the machine because handwritten digits are not perfect and can be made with many different flavors. The handwritten digit recognition is the solution to this problem which uses the image of a digit and recognizes the digit present in the image.

The MNIST dataset

The MNIST database (*Modified National Institute of Standards and Technology database*[1]) is a large database of handwritten digits that is commonly used for training various image processing systems.[2][3] The database is also widely used for training and testing in the field of machine learning.[4][5] It was created by "re-mixing" the samples from NIST's original datasets.[6] The creators felt that since NIST's training dataset was taken from American Census Bureau employees, while the testing dataset was taken from American high school students, it was not well-suited for machine learning experiments.[7] Furthermore, the black and white images from NIST were normalized to fit into a 28x28 pixel bounding box and anti-aliased, which introduced grayscale levels.[7]

The MNIST database contains 60,000 training images and 10,000 testing images.[8] Half of the training set and half of the test set were taken from NIST's training dataset, while the other half of the training set and the other half of the test set were taken from NIST's testing dataset.[9] The original creators of the database keep a list of some of the methods tested on

it.[7] In their original paper, they use a support-vector machine to get an error rate of 0.8%.[10] An extended dataset similar to MNIST called EMNIST has been published in 2017, which contains 240,000 training images, and 40,000 testing images of handwritten digits and characters.[11]

Building Python Machine Learning Project on Handwritten Digit Recognition

Below are the steps to implement the handwritten digit recognition project:

1. Import the libraries and load the dataset

```
df
```

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	pixel11	pixel12	pixel13	pixel14	pixel15	pixel16	pixel17	pixel18	pixel19	pixel20	pixel21	pixel22	pixel23	pixe
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
1	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	
2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
3	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
4	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	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
41995	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	
41996	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
41997	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
41998	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
41999	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

42000 rows × 785 columns

2. INPUT the data

```
x = df.iloc[:,1:].values
y = df.iloc[:,0].values
x.shape
```

(42000, 784)

```
y.shape
```

(42000,)

3. Splitting the data

The MNIST dataset contains 29,400 training images of handwritten digits from zero to nine and 12,600 images for testing. So, the MNIST dataset has 10 different classes. The handwritten digits images are represented as a 28×28 matrix where each cell contains grayscale pixel value

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3,random_state = 0)
```

4. Standardizing the data

```
x_train = sc.fit_transform(x_train)
x_test = sc.fit_transform(x_test)
```

5. Create the model

In a Support Vector Machine (SVM) model, the dataset is represented as points in space. The space is separated in clusters by several hyperplanes. Each hyperplan tries to maximize the margin between two classes (i.e. the distance to the closest points is maximized).

Scikit-learn provided multiple Support Vector Machine classifier implementations. SVC supports multiple kernel functions (used to split with non-linearly) but the training time complexity is quadradic with the number of samples. Multiclass classification is done with a one-vs-one scheme. On the other hand, LinearSVC only supports linear kernels but the training time is linear with the number of samples. The multiclass classification is done with a one-vs-others scheme.

We'll use LinearSVC here because it is fast enough. Feel free to experiment the standard SVC with a smaller training set.

```
from sklearn.svm import SVC
svc =  SVC(kernel="rbf", random_state=0)
svc.fit(x_train,y_train)
```

6. Predicting the model

```
y_pred = svc.predict(x_test)
```

7. Evaluating the model

```
y_test[2000]
```

```
7
```

```
y_pred[2000]
```

```
7
```

```
7
```

8. RESULT

```
accuracy = accuracy_score(y_test,y_pred)
print("Accuracy: {}%".format(int(round(accuracy*100))))
```

```
Accuracy: 96%
```

CONCLUSION

I tried using different approaches for Supervised Machine Learning classification. It's not possible to say which one is the best to classify this MNIST dataset because that depends on the many criteria and they can be fine-tuned to improve their performance (which I didn't here). The K-nearest neighbors algorithm is fast to train the data but is slow to compute the results. On the other hand, the Random Forest is faster to classify the data. The results obtained with LinearSVC were good,here I have used “RBF” parameter to show the accuracy ,where all other paremeter are working with the same accuracy level as well.Hence I have got accuracy of 96%

```
from sklearn.metrics import classification_report
prediction = svc.predict(x_test)
classification_report(y_test,prediction)
```

```
precision recall f1-score support\n\n 0 0.98 0.98 0.98 1242\n 1 0.98 0.97 0.98 1429\n 2 0.89 0.97 0.93 1276\n 3 0.96 0.94 0.95 1298\n 4 0.97 0.95 0.96 1236\n 5 0.95 0.96 0.96 1119\n 6 0.96 0.96 0.96 1243\n 7 0.96 0.94 0.95 1334\n 8 0.95 0.94 0.95 1204\n 9 0.95 0.94 0.94 1219\n\n accuracy 0.96 12600\n macro avg 0.96 0.96 0.96 12600\nweighted avg 0.96 0.96 0.96 12600\n
```