In [1]:

```
import os
import numpy as np
import pandas as pd
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
```

In [ ]:

```
from sklearn.datasets import fetch_openml
X,y = fetch_openml('mnist_784', version = 1, return_X_y=True)
```

```
C:\Users\User\anaconda3\Lib\site-packages\sklearn\datasets\_openml.py:1002: F
utureWarning: The default value of `parser` will change from `'liac-arff'` to
`'auto'` in 1.4. You can set `parser='auto'` to silence this warning. Therefo
re, an `ImportError` will be raised from 1.4 if the dataset is dense and pand
as is not installed. Note that the pandas parser may return different data ty
pes. See the Notes Section in fetch_openml's API doc for details.
  warn(
```

In [ ]:

```
X.shape
```

In [ ]:

```
X.head()
```

In [ ]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=1/7,
                                                    random_state=0)
```

In [ ]:

```
X_train.shape
```

In [ ]:

```
X_test.shape
```

In [ ]:

```
y_train.shape
```

In [ ]:

```
y_test.shape
```

In [ ]:

```
X_train = X_train.to_numpy()
X_test = X_test.to_numpy()
```

In [ ]:
```
# reshape and scale to be in [0,1] that is normalisation
X_train = X_train.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 225
X_test /= 225
```

In [ ]:
```
plt.figure(figsize=(20,4))
for index in range(5):
    plt.subplot(1,5, index+1)
    plt.imshow(X_train[index].reshape((28,28)), cmap=plt.cm.gray)
    plt.title('Training:%i\n'%int(y_train.to_numpy()[index]), fontsize=20)
```

In [ ]:
```
#Model Development and Prediction
from sklearn.neural_network import MLPClassifier
```

In [ ]:
```
#make an instance of the Model
mlp = MLPClassifier()
mlp = MLPClassifier(hidden_layer_sizes=(100,50), activation="relu", solver='ada
```

In [ ]:
```
#Training the model on the data, storing the information learned from the data
#Training
mlp.fit(X_train, y_train)
```

In [ ]:
```
#Prediction
predictions = mlp.predict(X_test)
```

In [ ]:
```
#Evaluation
score = mlp.score(X_test, y_test)
print(score)
```

In [ ]:
```
from sklearn.metrics import classification_report
print (classification_report(y_test, predictions, target_names=mlp.classes_toli
```

In [ ]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, predictions)
```

In [ ]:
```
plt.figure(figsize=(9.9))
sns.heatmap(cm, annot=True, fmt=",2f", linewidth=0.5, square=True, cmap="Blues_
plt.ylabel("Actual label")
plt.xlabel("Predicted label")
plt.title('Accuracy Score:{0}'.format(score), size=15)
plt.show()
```

In [ ]:

```
plt.plot(mlp.loss_curve_)
plt.title("Loss Curve", fontsize=14)
plt.xlabel('Iterations')
plt.ylabel('Cost')
plt.show()
```

In [ ]: