1. Import Libraries

In [ ]:

```
import pandas as pd
import matplotlib.pyplot as plt
import keras
import tensorflow as tf
from  tensorflow.keras.utils import to_categorical
```

2. Import DataSet

In [14]:

```
(x_train,y_train),(x_test,y_test)=tf.keras.datasets.mnist.load_data(path='mnist.npz')
```

In [15]:

```
x_train.shape,y_train.shape
```

Out[15]:

```
((60000, 28, 28), (60000,))
```

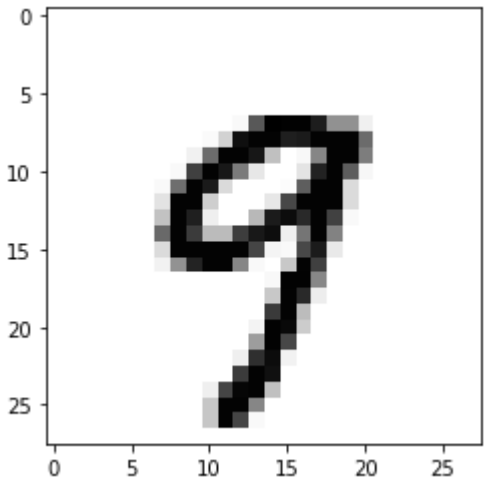
3.Data Undestanding

In [16]:

```
plt.imshow(x_train[45],cmap="Greys")
```

Out[16]:

```
<matplotlib.image.AxesImage at 0x248e792af40>
```



In [17]:

```
y_train[45]
```

Out[17]:

```
9
```

4.Standrization Data

In [18]:

```
x_train=x_train/255
x_test=x_test/255
```

5.Reshape input data

In [21]:

```
x_train_reshape=x_train.reshape(60000,28,28,1)
x_train_reshape.shape
```

Out[21]:

```
(60000, 28, 28, 1)
```

In [20]:

```
x_test_reshape=x_test.reshape(10000,28,28,1)
```

6. output we have to add softmax so add one hot encoding in end of output function

In [28]:

```
y_train_encoder=to_categorical(y_train)
y_test_encoder=to_categorical(y_test)
y_train_encoder.shape
```

Out[28]:

```
(60000, 10)
```

7. Model Building

In [29]:

```
from keras.layers import Flatten,Dense
from keras.models import Sequential
```

In [43]:

```
model=Sequential()
model.add(Flatten())
model.add(Dense(units=150,activation='tanh'))
# model.add(Dense(units=150,activation='tanh'))
model.add(Dense(units=10,activation="softmax"))
```

8.Model Training

In [47]:

```
model.compile(optimizer='rmsprop',loss='categorical_crossentropy',metrics='categorical_accu
```

9.Model Testing

In [48]:

```
model.fit(x=x_train_reshape,y=y_train_encoder,batch_size=100,epochs=15,validation_data=(x_t
```

```
Epoch 1/15
600/600 [==============================] - 4s 5ms/step - loss: 0.0078 - ca
tegorical_accuracy: 0.9990 - val_loss: 0.0688 - val_categorical_accuracy:
0.9796
Epoch 2/15
600/600 [==============================] - 3s 5ms/step - loss: 0.0065 - ca
tegorical_accuracy: 0.9990 - val_loss: 0.0726 - val_categorical_accuracy:
0.9795
Epoch 3/15
600/600 [==============================] - 3s 5ms/step - loss: 0.0052 - ca
tegorical_accuracy: 0.9995 - val_loss: 0.0716 - val_categorical_accuracy:
0.9790
Epoch 4/15
600/600 [==============================] - 3s 5ms/step - loss: 0.0045 - ca
tegorical_accuracy: 0.9994 - val_loss: 0.0718 - val_categorical_accuracy:
0.9804
Epoch 5/15
600/600 [==============================] - 3s 5ms/step - loss: 0.0038 - ca
tegorical_accuracy: 0.9995 - val_loss: 0.0774 - val_categorical_accuracy:
0.9796
Epoch 6/15
600/600 [==============================] - 3s 4ms/step - loss: 0.0031 - ca
tegorical_accuracy: 0.9996 - val_loss: 0.0742 - val_categorical_accuracy:
0.9803
Epoch 7/15
600/600 [==============================] - 3s 5ms/step - loss: 0.0026 - ca
tegorical_accuracy: 0.9996 - val_loss: 0.0786 - val_categorical_accuracy:
0.9791
Epoch 8/15
600/600 [==============================] - 3s 5ms/step - loss: 0.0021 - ca
tegorical_accuracy: 0.9998 - val_loss: 0.0777 - val_categorical_accuracy:
0.9794
Epoch 9/15
600/600 [==============================] - 3s 5ms/step - loss: 0.0019 - ca
tegorical_accuracy: 0.9998 - val_loss: 0.0816 - val_categorical_accuracy:
0.9797
Epoch 10/15
600/600 [==============================] - 3s 5ms/step - loss: 0.0014 - ca
tegorical_accuracy: 0.9998 - val_loss: 0.0797 - val_categorical_accuracy:
0.9805
Epoch 11/15
600/600 [==============================] - 3s 5ms/step - loss: 0.0013 - ca
tegorical_accuracy: 0.9998 - val_loss: 0.0865 - val_categorical_accuracy:
0.9792
Epoch 12/15
600/600 [==============================] - 3s 5ms/step - loss: 0.0010 - ca
tegorical_accuracy: 0.9998 - val_loss: 0.0853 - val_categorical_accuracy:
0.9803
Epoch 13/15
600/600 [==============================] - 3s 5ms/step - loss: 8.8990e-04
- categorical_accuracy: 0.9999 - val_loss: 0.0868 - val_categorical_accura
cy: 0.9803
Epoch 14/15
600/600 [==============================] - 3s 5ms/step - loss: 6.9647e-04
- categorical_accuracy: 0.9999 - val_loss: 0.0921 - val_categorical_accura
cy: 0.9799
```

```
Epoch 15/15
600/600 [==============================] - 3s 5ms/step - loss: 6.2005e-04
- categorical_accuracy: 0.9999 - val_loss: 0.0884 - val_categorical_accura
cy: 0.9798
```

Out[48]:

```
<keras.callbacks.History at 0x248897c44c0>
```

In [49]:

```
model.evaluate(x_test_reshape,y_test_encoder)
```

```
313/313 [==============================] - 1s 4ms/step - loss: 0.0884 - cate
gorical_accuracy: 0.9798
```

Out[49]:

```
[0.08840543031692505, 0.9797999858856201]
```

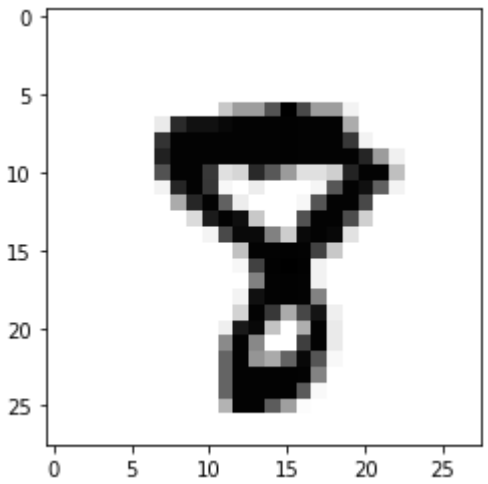
10.Model Testing

In [38]:

```
plt.imshow(x_test_reshape[233],cmap="Greys")
```

Out[38]:

```
<matplotlib.image.AxesImage at 0x2488716eaf0>
```



In [39]:

```
import numpy as np
```

In [40]:

```
np.argmax(y_test_encoder[233])
```

Out[40]:

```
8
```