IMPORTING REQUIRED LIBRARIES

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
print("Successfully Imported!!")
```

```
Successfully Imported!!
```

IMPORTING DATASET

In [2]:
```
##Importing Training data set
data_train=pd.read_csv('fashion-mnist_train.csv')
training_data=pd.DataFrame(data_train)
training_data.head(10)
```

Out[2]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel
0	2	0	0	0	0	0	0	0	0	0	...	0	
1	9	0	0	0	0	0	0	0	0	0	...	0	
2	6	0	0	0	0	0	0	0	5	0	...	0	
3	0	0	0	0	1	2	0	0	0	0	...	3	
4	3	0	0	0	0	0	0	0	0	0	...	0	
5	4	0	0	0	5	4	5	5	3	5	...	7	
6	4	0	0	0	0	0	0	0	0	0	...	14	
7	5	0	0	0	0	0	0	0	0	0	...	0	
8	4	0	0	0	0	0	0	3	2	0	...	1	
9	8	0	0	0	0	0	0	0	0	0	...	203	

10 rows × 785 columns

In [3]:
```
training_data.tail(10)
```

Out[3]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775
59990	0	0	0	0	0	0	0	0	0	0	...	154
59991	5	0	0	0	0	0	0	0	0	0	...	0
59992	5	0	0	0	0	0	0	0	0	0	...	0
59993	2	0	0	0	0	0	0	1	0	0	...	0
59994	9	0	0	0	0	0	0	0	0	0	...	0
59995	9	0	0	0	0	0	0	0	0	0	...	0
59996	1	0	0	0	0	0	0	0	0	0	...	73
59997	8	0	0	0	0	0	0	0	0	0	...	160
59998	8	0	0	0	0	0	0	0	0	0	...	0
59999	7	0	0	0	0	0	0	0	0	0	...	0

10 rows × 785 columns

In [4]:

```
##Importing Test data set
data_test=pd.read_csv('fashion-mnist_test.csv')
testing_data=pd.DataFrame(data_test)
testing_data.head(10)
```

Out[4]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel
0	0	0	0	0	0	0	0	0	9	8	...	103	
1	1	0	0	0	0	0	0	0	0	0	...	34	
2	2	0	0	0	0	0	0	14	53	99	...	0	
3	2	0	0	0	0	0	0	0	0	0	...	137	
4	3	0	0	0	0	0	0	0	0	0	...	0	
5	2	0	0	0	0	0	44	105	44	10	...	105	
6	8	0	0	0	0	0	0	0	0	0	...	0	
7	6	0	0	0	0	0	0	0	1	0	...	174	
8	5	0	0	0	0	0	0	0	0	0	...	0	
9	0	0	0	0	0	0	0	0	0	0	...	57	

10 rows × 785 columns

In [5]:

```
testing_data.tail(10)
```

Out[5]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	p
9990	7	0	0	0	0	0	0	0	0	0	...	0	
9991	9	0	0	0	0	0	0	0	0	0	...	0	
9992	4	0	0	0	0	0	0	0	0	0	...	120	
9993	8	0	0	0	0	0	0	0	0	0	...	0	
9994	0	0	0	0	0	0	0	0	1	0	...	85	
9995	0	0	0	0	0	0	0	0	0	0	...	32	
9996	6	0	0	0	0	0	0	0	0	0	...	0	
9997	8	0	0	0	0	0	0	0	0	0	...	175	
9998	8	0	1	3	0	0	0	0	0	0	...	0	
9999	1	0	0	0	0	0	0	0	140	119	...	111	

10 rows × 785 columns

In [6]:
```
training_data.shape
```
Out[6]:
```
(60000, 785)
```

In [7]:
```
testing_data.shape
```
Out[7]:
```
(10000, 785)
```

CONVERTING THE DATA SET BOTH TRAINING AND TESTING INTO NUMPY ARRAY

In [8]:
```
training=np.array(training_data,dtype='float')
testing=np.array(testing_data,dtype='float')
```

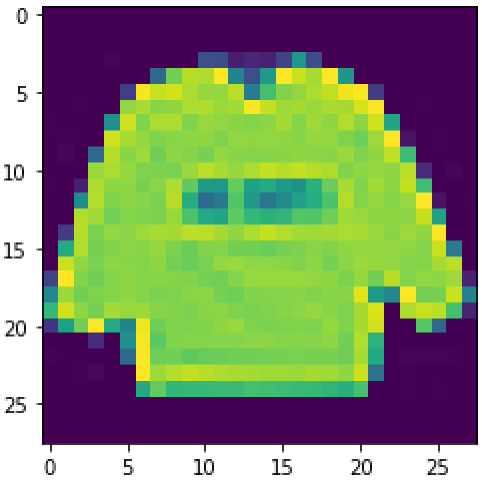
In [9]:
```
training
```
Out[9]:
```
array([[2., 0., 0., ..., 0., 0., 0.],
       [9., 0., 0., ..., 0., 0., 0.],
       [6., 0., 0., ..., 0., 0., 0.],
       ...,
       [8., 0., 0., ..., 0., 0., 0.],
       [8., 0., 0., ..., 0., 0., 0.],
       [7., 0., 0., ..., 0., 0., 0.]])
```

In [10]:
```
testing
```
Out[10]:
```
array([[0., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       [2., 0., 0., ..., 0., 0., 0.],
       ...,
       [8., 0., 0., ..., 0., 1., 0.],
       [8., 0., 1., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.]])
```

PLOTTING ALL THE LABELS THAT IS PRESENT IN DATASET

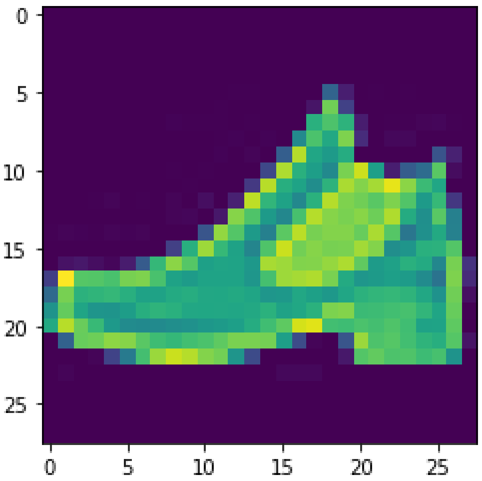
In [11]:
```
plt.imshow(training[0,1:].reshape(28,28))
```

Out[11]: <matplotlib.image.AxesImage at 0x1f4c528f7c0>



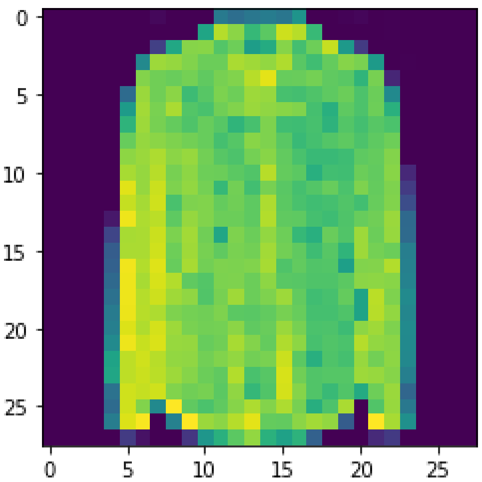
In [12]:
```
plt.imshow(training[1,1:].reshape(28,28))
```

Out[12]: <matplotlib.image.AxesImage at 0x1f4c539b940>



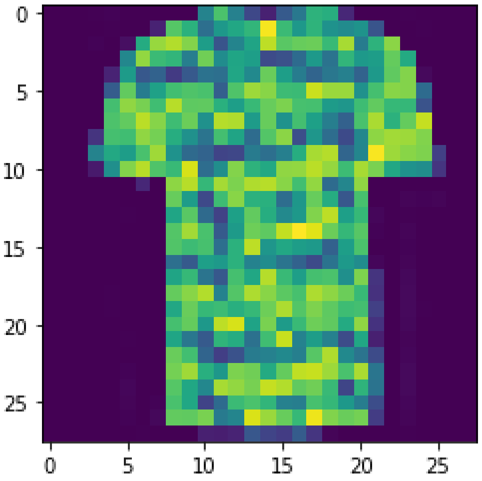
In [13]:
```
plt.imshow(training[2,1:].reshape(28,28))
```

Out[13]: <matplotlib.image.AxesImage at 0x1f4c542b430>



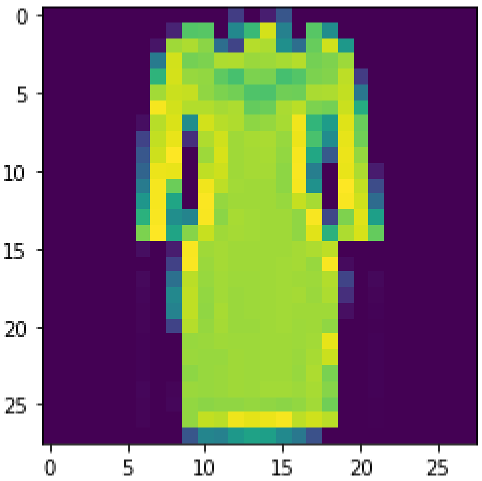
In [14]:
```
plt.imshow(training[3,1:].reshape(28,28))
```

Out[14]: <matplotlib.image.AxesImage at 0x1f4c5b3a760>



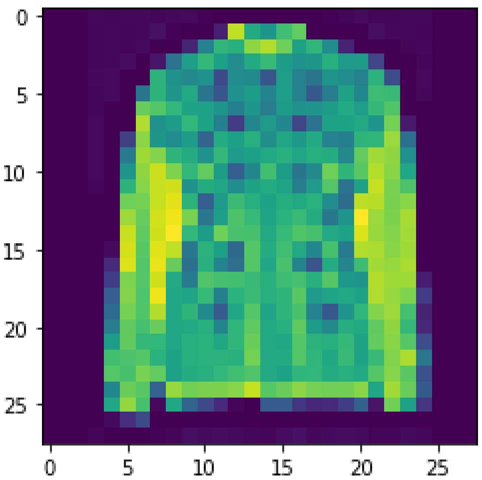
In [15]:
```python
plt.imshow(training[4,1:].reshape(28,28))
```

Out[15]: <matplotlib.image.AxesImage at 0x1f4c5330970>



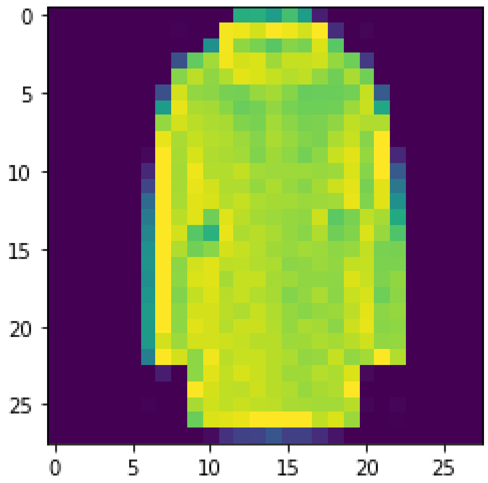
In [16]:
```python
plt.imshow(training[5,1:].reshape(28,28))
```

Out[16]: <matplotlib.image.AxesImage at 0x1f4c509da30>



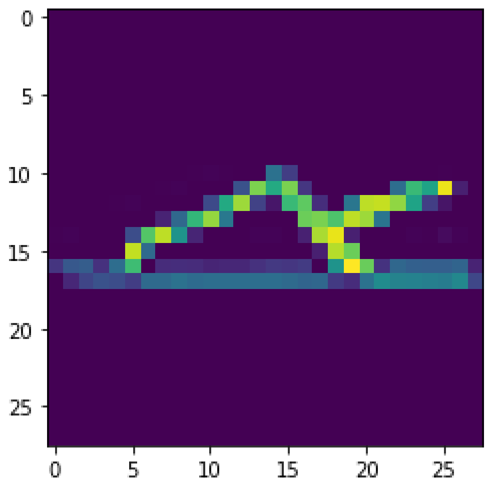
In [17]:
```python
plt.imshow(training[6,1:].reshape(28,28))
```

Out[17]: <matplotlib.image.AxesImage at 0x1f4c522ed00>



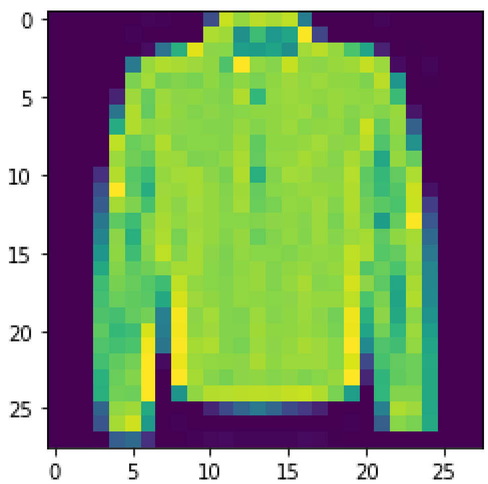
In [18]:
```
plt.imshow(training[7,1:].reshape(28,28))
```

Out[18]: <matplotlib.image.AxesImage at 0x1f4806fb0d0>



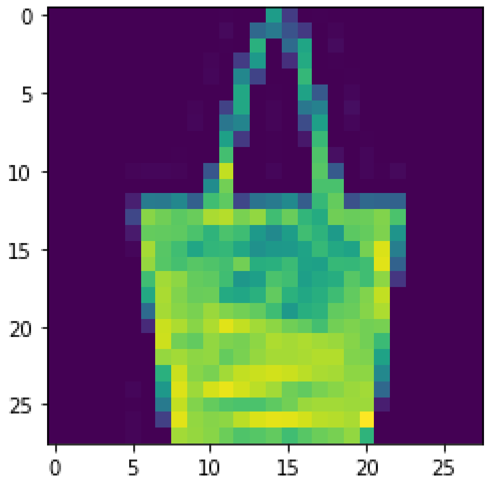
In [19]:
```
plt.imshow(training[8,1:].reshape(28,28))
```

Out[19]: <matplotlib.image.AxesImage at 0x1f4807593d0>



In [20]:
```
plt.imshow(training[9,1:].reshape(28,28))
```

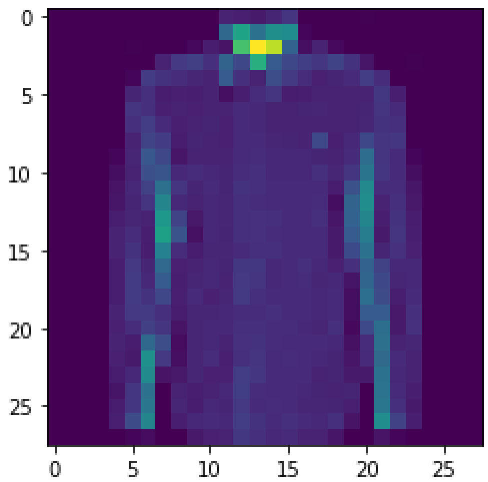
Out[20]: <matplotlib.image.AxesImage at 0x1f4807bb610>



In [21]:
```
class_names = ['T_shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

In [22]:
```
import random
i=random.randint(1,60000)
plt.imshow(training[i,1:].reshape(28,28))
label=training[i,0]
x=int(label)
print(class_names[x])
```

```
Shirt
```



Labels Each training and test example is assigned to one of the following labels as shown below:

0 T-shirt/top

1 Trouser

2 Pullover

3 Dress

4 Coat

5 Sandal

6 Shirt

7 Sneaker

8 Bag

9 Ankle boot

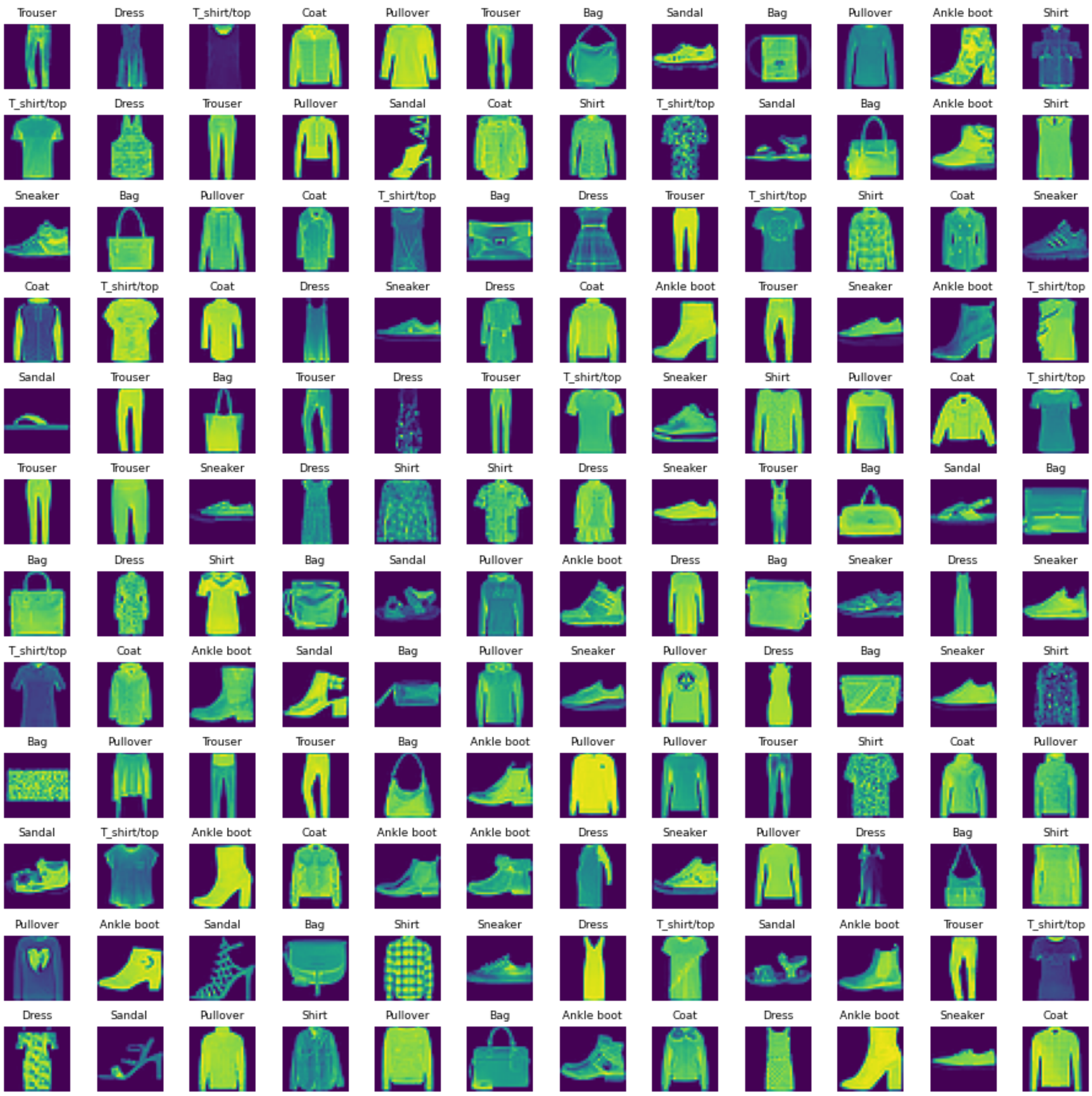
In [23]:
```python
W_grid = 12
L_grid = 12

fig, axes = plt.subplots(L_grid, W_grid, figsize = (16,16))
axes = axes.ravel() # flaten the 12 x 12 matrix into 225 array
n_train = len(training) # get the length of the train dataset

# Select a random number from 0 to n_train
for i in np.arange(0, W_grid * L_grid): # create evenly spaces variables

    # Select a random number
    index = np.random.randint(0, n_train)
    # read and display an image with the selected index
    axes[i].imshow( training[index,1:].reshape((28,28)) )
    labelindex = int(training[index,0])
    axes[i].set_title(class_names[labelindex], fontsize = 9)
    axes[i].axis('off')

plt.subplots_adjust(hspace=0.4)
```



TRAINING DATA FOR MODEL

In [24]:
```
X_train=training[:,1:]/255
y_train=training[:,0]
```

In [25]:
```
X_test=testing[:,1:]/255
y_test=testing[:,0]
```

In [26]:
```
from sklearn.model_selection import train_test_split
```

In [27]:
```
X_train,X_validate,y_train,y_validate=train_test_split(X_train,y_train,test_size=0
```

In [28]:
```
X_train.shape
```

Out[28]:
```
(48000, 784)
```

In [29]:
```
X_validate.shape
```

Out[29]:
```
(12000, 784)
```

In [30]:
```
X_train
```

Out[30]:
```
array([[0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       ...,
       [0.        , 0.        , 0.        , ..., 0.08235294, 0.03921569,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ]])
```

In [31]:
```
X_train=X_train.reshape(X_train.shape[0],*(28,28,1))
X_test=X_test.reshape(X_test.shape[0],*(28,28,1))
X_validate=X_validate.reshape(X_validate.shape[0],*(28,28,1))
```

In [32]:
```
X_train.shape
```

Out[32]:
```
(48000, 28, 28, 1)
```

In [33]:
```
X_test.shape
```

Out[33]:
```
(10000, 28, 28, 1)
```

In [34]:
```
X_validate.shape
```

Out[34]:
```
(12000, 28, 28, 1)
```

In [35]:
```
import tensorflow as tf
import keras
from keras.models import Sequential
from keras.layers import Conv2D,Conv3D,MaxPool2D,Flatten,Dense,Dropout
from keras.optimizers import Adam
```

In [36]:
```
our_model=Sequential()
our_model
```

Out[36]: <keras.engine.sequential.Sequential at 0x1f4c5b763d0>

In [37]:
```
our_model.add(Conv2D(32,3,3, input_shape=(28,28,1),activation='relu'))
```

In [38]:
```
our_model.add(MaxPool2D(pool_size=(2,2)))
```

In [39]:
```
our_model.add(Flatten())
```

In [40]:
```
our_model.add(Dense(units=32,activation='relu'))
```

In [41]:
```
our_model.add(Dense(units=10,activation='sigmoid'))
```

In [42]:
```
#importing sparse_categorical_crossentropy from tensorflow
tf.keras.losses.SparseCategoricalCrossentropy(
    from_logits=False,
    name='sparse_categorical_crossentropy'
)
our_model.compile(loss='sparse_categorical_crossentropy',optimizer=Adam(learning_ra
```

In [43]:
```
epochs=50
```

In [44]:
```
our_model.fit(X_train,
    y_train,
    batch_size=None,
    epochs=epochs,
    verbose=1,
    validation_data=(X_validate,y_validate))
```

```
Epoch 1/50
1500/1500 [==============================] - 6s 4ms/step - loss: 0.7151 - accurac
y: 0.7435 - val_loss: 0.5243 - val_accuracy: 0.8040
Epoch 2/50
1500/1500 [==============================] - 6s 4ms/step - loss: 0.4815 - accurac
y: 0.8232 - val_loss: 0.4705 - val_accuracy: 0.8284
Epoch 3/50
1500/1500 [==============================] - 6s 4ms/step - loss: 0.4413 - accurac
y: 0.8381 - val_loss: 0.4301 - val_accuracy: 0.8447
Epoch 4/50
1500/1500 [==============================] - 6s 4ms/step - loss: 0.4196 - accurac
y: 0.8459 - val_loss: 0.4341 - val_accuracy: 0.8422
Epoch 5/50
1500/1500 [==============================] - 8s 5ms/step - loss: 0.4063 - accurac
y: 0.8509 - val_loss: 0.4065 - val_accuracy: 0.8518
Epoch 6/50
1500/1500 [==============================] - 8s 5ms/step - loss: 0.3938 - accurac
y: 0.8557 - val_loss: 0.4006 - val_accuracy: 0.8528
Epoch 7/50
1500/1500 [==============================] - 6s 4ms/step - loss: 0.3837 - accurac
y: 0.8607 - val_loss: 0.3886 - val_accuracy: 0.8599
Epoch 8/50
1500/1500 [==============================] - 7s 4ms/step - loss: 0.3770 - accurac
y: 0.8616 - val_loss: 0.4192 - val_accuracy: 0.8451
Epoch 9/50
1500/1500 [==============================] - 7s 4ms/step - loss: 0.3683 - accurac
y: 0.8658 - val_loss: 0.3953 - val_accuracy: 0.8561
Epoch 10/50
1500/1500 [==============================] - 5s 3ms/step - loss: 0.3619 - accurac
y: 0.8675 - val_loss: 0.4010 - val_accuracy: 0.8510
Epoch 11/50
1500/1500 [==============================] - 6s 4ms/step - loss: 0.3539 - accurac
y: 0.8700 - val_loss: 0.3857 - val_accuracy: 0.8585
Epoch 12/50
1500/1500 [==============================] - 7s 5ms/step - loss: 0.3475 - accurac
y: 0.8723 - val_loss: 0.3741 - val_accuracy: 0.8649
Epoch 13/50
1500/1500 [==============================] - 6s 4ms/step - loss: 0.3433 - accurac
y: 0.8759 - val_loss: 0.3725 - val_accuracy: 0.8620
Epoch 14/50
1500/1500 [==============================] - 7s 5ms/step - loss: 0.3358 - accurac
y: 0.8771 - val_loss: 0.3736 - val_accuracy: 0.8622
Epoch 15/50
1500/1500 [==============================] - 5s 3ms/step - loss: 0.3321 - accurac
y: 0.8775 - val_loss: 0.3641 - val_accuracy: 0.8677
Epoch 16/50
1500/1500 [==============================] - 9s 6ms/step - loss: 0.3274 - accurac
y: 0.8806 - val_loss: 0.3616 - val_accuracy: 0.8690
Epoch 17/50
1500/1500 [==============================] - 6s 4ms/step - loss: 0.3221 - accurac
y: 0.8817 - val_loss: 0.3656 - val_accuracy: 0.8686
Epoch 18/50
1500/1500 [==============================] - 7s 5ms/step - loss: 0.3174 - accurac
y: 0.8834 - val_loss: 0.3623 - val_accuracy: 0.8694
Epoch 19/50
1500/1500 [==============================] - 5s 3ms/step - loss: 0.3128 - accurac
y: 0.8842 - val_loss: 0.3605 - val_accuracy: 0.8696
Epoch 20/50
1500/1500 [==============================] - 8s 5ms/step - loss: 0.3098 - accurac
y: 0.8859 - val_loss: 0.3580 - val_accuracy: 0.8704
Epoch 21/50
1500/1500 [==============================] - 6s 4ms/step - loss: 0.3072 - accurac
y: 0.8871 - val_loss: 0.3619 - val_accuracy: 0.8679
Epoch 22/50
```

```
1500/1500 [==============================] - 7s 5ms/step - loss: 0.3029 - accurac
y: 0.8886 - val_loss: 0.3595 - val_accuracy: 0.8684
Epoch 23/50
1500/1500 [==============================] - 9s 6ms/step - loss: 0.2996 - accurac
y: 0.8890 - val_loss: 0.3578 - val_accuracy: 0.8708
Epoch 24/50
1500/1500 [==============================] - 8s 5ms/step - loss: 0.2972 - accurac
y: 0.8896 - val_loss: 0.3540 - val_accuracy: 0.8717
Epoch 25/50
1500/1500 [==============================] - 8s 5ms/step - loss: 0.2946 - accurac
y: 0.8901 - val_loss: 0.3629 - val_accuracy: 0.8683
Epoch 26/50
1500/1500 [==============================] - 7s 5ms/step - loss: 0.2908 - accurac
y: 0.8923 - val_loss: 0.3610 - val_accuracy: 0.8716
Epoch 27/50
1500/1500 [==============================] - 5s 4ms/step - loss: 0.2881 - accurac
y: 0.8927 - val_loss: 0.3705 - val_accuracy: 0.8698
Epoch 28/50
1500/1500 [==============================] - 5s 3ms/step - loss: 0.2866 - accurac
y: 0.8937 - val_loss: 0.3671 - val_accuracy: 0.8692
Epoch 29/50
1500/1500 [==============================] - 6s 4ms/step - loss: 0.2829 - accurac
y: 0.8941 - val_loss: 0.3658 - val_accuracy: 0.8697
Epoch 30/50
1500/1500 [==============================] - 8s 5ms/step - loss: 0.2811 - accurac
y: 0.8955 - val_loss: 0.3662 - val_accuracy: 0.8712
Epoch 31/50
1500/1500 [==============================] - 5s 3ms/step - loss: 0.2772 - accurac
y: 0.8971 - val_loss: 0.3622 - val_accuracy: 0.8703
Epoch 32/50
1500/1500 [==============================] - 6s 4ms/step - loss: 0.2757 - accurac
y: 0.8972 - val_loss: 0.3538 - val_accuracy: 0.8745
Epoch 33/50
1500/1500 [==============================] - 7s 4ms/step - loss: 0.2723 - accurac
y: 0.8981 - val_loss: 0.3544 - val_accuracy: 0.8742
Epoch 34/50
1500/1500 [==============================] - 5s 4ms/step - loss: 0.2710 - accurac
y: 0.8997 - val_loss: 0.3569 - val_accuracy: 0.8729
Epoch 35/50
1500/1500 [==============================] - 8s 6ms/step - loss: 0.2685 - accurac
y: 0.8996 - val_loss: 0.3614 - val_accuracy: 0.8692
Epoch 36/50
1500/1500 [==============================] - 6s 4ms/step - loss: 0.2664 - accurac
y: 0.9010 - val_loss: 0.3519 - val_accuracy: 0.8767
Epoch 37/50
1500/1500 [==============================] - 7s 5ms/step - loss: 0.2647 - accurac
y: 0.9017 - val_loss: 0.3705 - val_accuracy: 0.8685
Epoch 38/50
1500/1500 [==============================] - 6s 4ms/step - loss: 0.2626 - accurac
y: 0.9029 - val_loss: 0.3591 - val_accuracy: 0.8726
Epoch 39/50
1500/1500 [==============================] - 5s 3ms/step - loss: 0.2605 - accurac
y: 0.9034 - val_loss: 0.3580 - val_accuracy: 0.8726
Epoch 40/50
1500/1500 [==============================] - 6s 4ms/step - loss: 0.2575 - accurac
y: 0.9031 - val_loss: 0.3660 - val_accuracy: 0.8711
Epoch 41/50
1500/1500 [==============================] - 5s 3ms/step - loss: 0.2567 - accurac
y: 0.9056 - val_loss: 0.3649 - val_accuracy: 0.8730
Epoch 42/50
1500/1500 [==============================] - 5s 4ms/step - loss: 0.2552 - accurac
y: 0.9052 - val_loss: 0.3655 - val_accuracy: 0.8710
Epoch 43/50
1500/1500 [==============================] - 6s 4ms/step - loss: 0.2526 - accurac
```

```
y: 0.9065 - val_loss: 0.3704 - val_accuracy: 0.8748
Epoch 44/50
1500/1500 [==============================] - 5s 4ms/step - loss: 0.2522 - accurac
y: 0.9063 - val_loss: 0.3725 - val_accuracy: 0.8692
Epoch 45/50
1500/1500 [==============================] - 5s 3ms/step - loss: 0.2504 - accurac
y: 0.9071 - val_loss: 0.3604 - val_accuracy: 0.8748
Epoch 46/50
1500/1500 [==============================] - 6s 4ms/step - loss: 0.2489 - accurac
y: 0.9072 - val_loss: 0.3762 - val_accuracy: 0.8711
Epoch 47/50
1500/1500 [==============================] - 5s 3ms/step - loss: 0.2468 - accurac
y: 0.9083 - val_loss: 0.3710 - val_accuracy: 0.8755
Epoch 48/50
1500/1500 [==============================] - 5s 3ms/step - loss: 0.2454 - accurac
y: 0.9070 - val_loss: 0.3824 - val_accuracy: 0.8726
Epoch 49/50
1500/1500 [==============================] - 6s 4ms/step - loss: 0.2438 - accurac
y: 0.9095 - val_loss: 0.3837 - val_accuracy: 0.8644
Epoch 50/50
1500/1500 [==============================] - 6s 4ms/step - loss: 0.2423 - accurac
y: 0.9106 - val_loss: 0.3843 - val_accuracy: 0.8734
```

Out[44]:
```
<keras.callbacks.History at 0x1f494ed2fd0>
```

In [45]:
```python
evaluate_m=our_model.evaluate(X_test,y_test)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.3722 - accuracy:
0.8749
```

In [46]:
```python
print(f'Test Accuracy {round(evaluate_m[1],2)}')
```

```
Test Accuracy 0.87
```

In [47]:
```python
predict_x=our_model.predict(X_test)
classes_x=np.argmax(predict_x,axis=1)
```

```
313/313 [==============================] - 1s 2ms/step
```

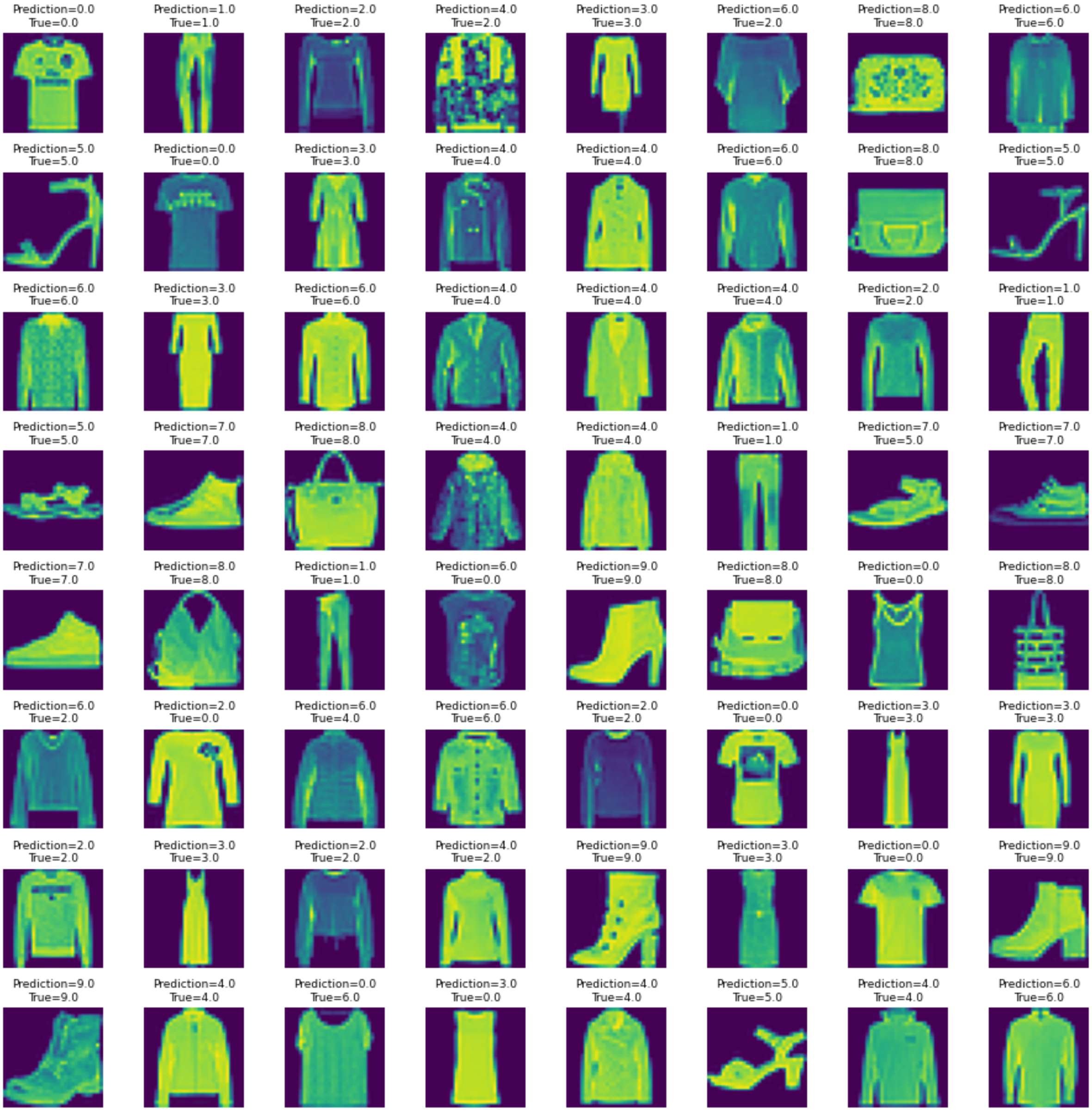
In [48]:
```python
classes_x
```

Out[48]:
```
array([0, 1, 2, ..., 8, 8, 1], dtype=int64)
```

In [49]:
```python
W_grid = 8
L_grid = 8

fig, axes = plt.subplots(L_grid, W_grid, figsize = (16,16))
axes = axes.ravel() # flaten the 12 x 12 matrix into 225 array
# Select a random number from 0 to n_train
for i in np.arange(0, W_grid * L_grid): # create evenly spaces variables
    # read and display an image with the selected index
    axes[i].imshow(X_test[i].reshape((28,28)) )
    axes[i].set_title('Prediction={:0.1f}\nTrue={:0.1f}'.format(classes_x[i],y_tes
    axes[i].axis('off')

plt.subplots_adjust(hspace=0.4)
```

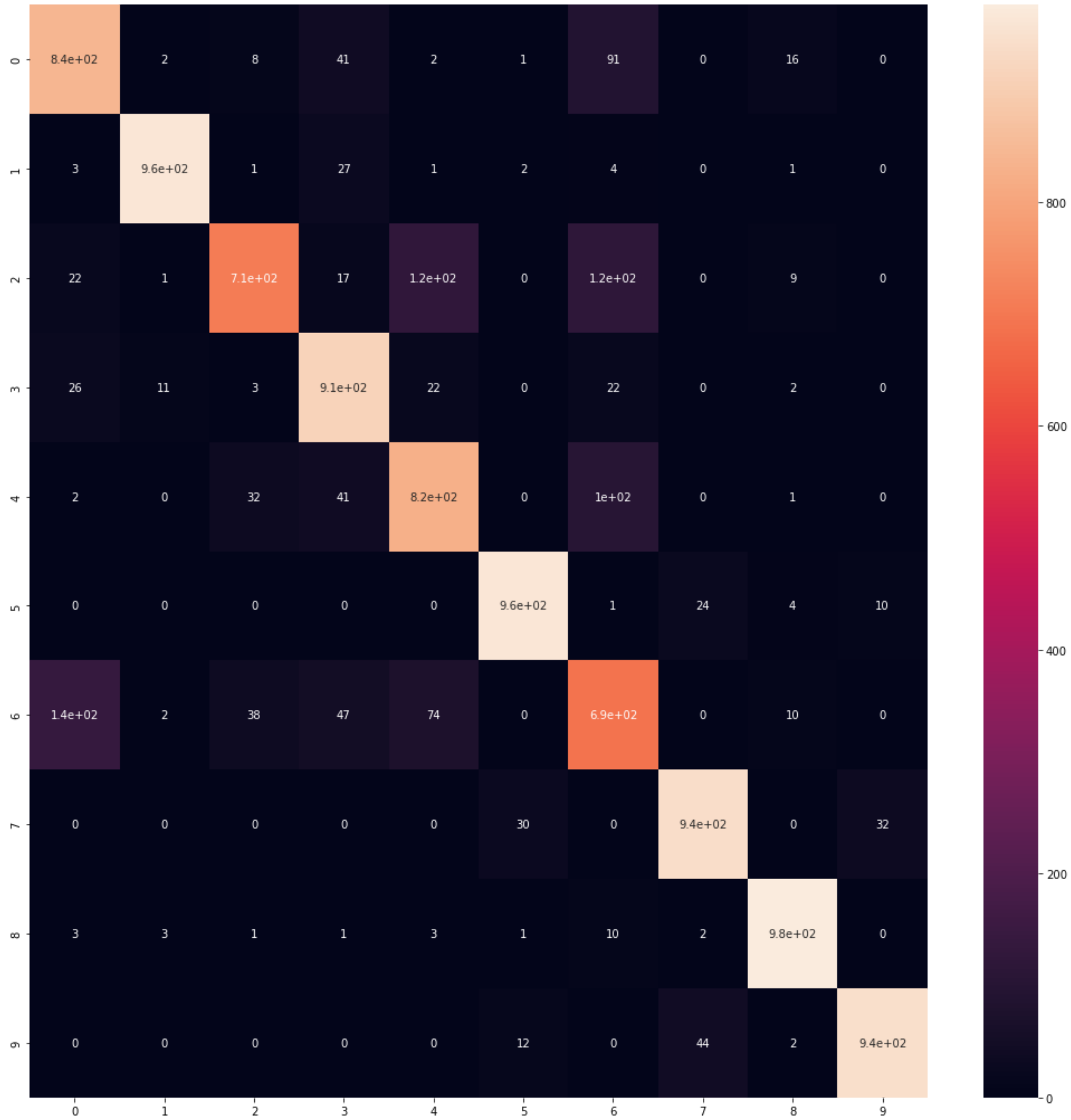


In [50]:
```
from sklearn.metrics import confusion_matrix
```

In [51]:
```
cn=confusion_matrix(y_test,classes_x)
```

In [52]:
```
plt.figure(figsize=(18,18))
sns.heatmap(cn,annot=True)
```

Out[52]: <AxesSubplot:>



In [53]:

```
from sklearn.metrics import classification_report

print(classification_report(y_test,classes_x))
```

```
              precision    recall  f1-score   support

         0.0       0.81      0.84      0.82      1000
         1.0       0.98      0.96      0.97      1000
         2.0       0.89      0.71      0.79      1000
         3.0       0.84      0.91      0.88      1000
         4.0       0.78      0.82      0.80      1000
         5.0       0.95      0.96      0.96      1000
         6.0       0.66      0.69      0.68      1000
         7.0       0.93      0.94      0.93      1000
         8.0       0.96      0.98      0.97      1000
         9.0       0.96      0.94      0.95      1000

    accuracy                           0.87     10000
   macro avg       0.88      0.87      0.87     10000
weighted avg       0.88      0.87      0.87     10000
```

In [ ]: