AI

MACHINE LEARNING ENVIRONMENT SETUP

Dennis A. N. Gookyi

CONTENTS

- Machine Learning Environment Setup

TENSORFLOW ON A LOCAL MACHINE - TOOLS

❖ TensorFlow on a local machine

Anaconda Jupyter

- Anaconda – Python Distribution.
- Jupyter Notebooks – Web-based program to code Python

TENSORFLOW ON A LOCAL MACHINE - TOOLS

❖ TensorFlow on a local machine

Numpy – Multidimensional arrays and matrices

Keras – Python interface for ANN

TensorFlow – Training and Inference of DNN

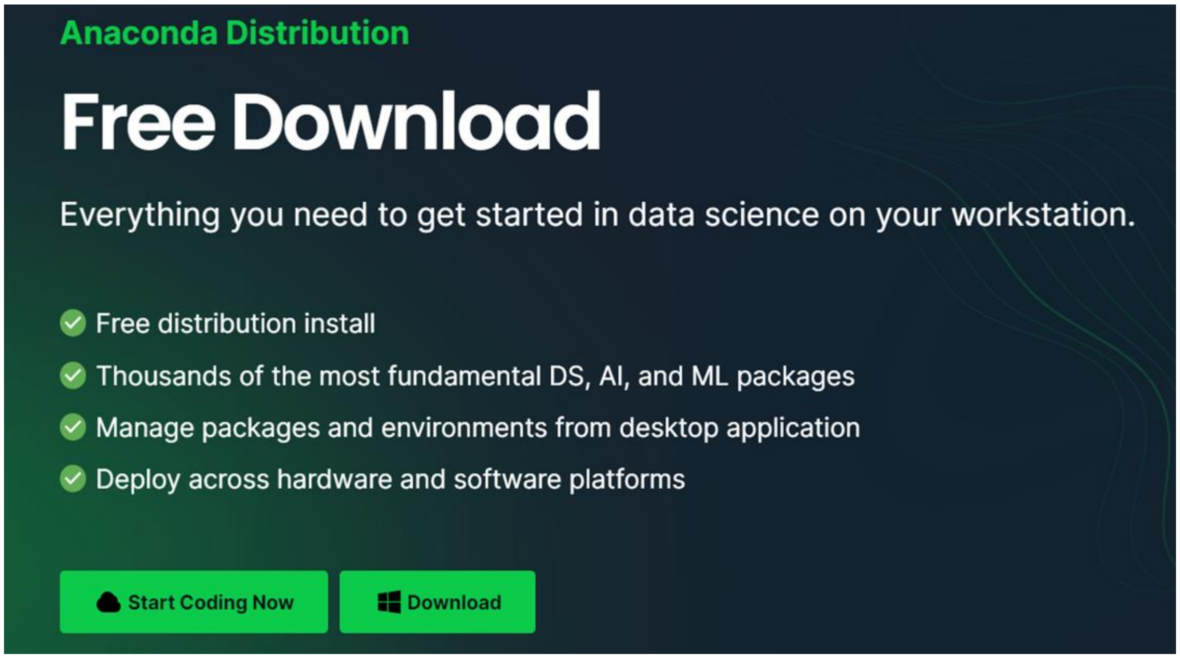
Pandas – data manipulation and analysis

OpenCV – Real-time computer vision

Matplotlib – Plottig library

INSTALLATION OF TOOLS

❖ Installation of tools

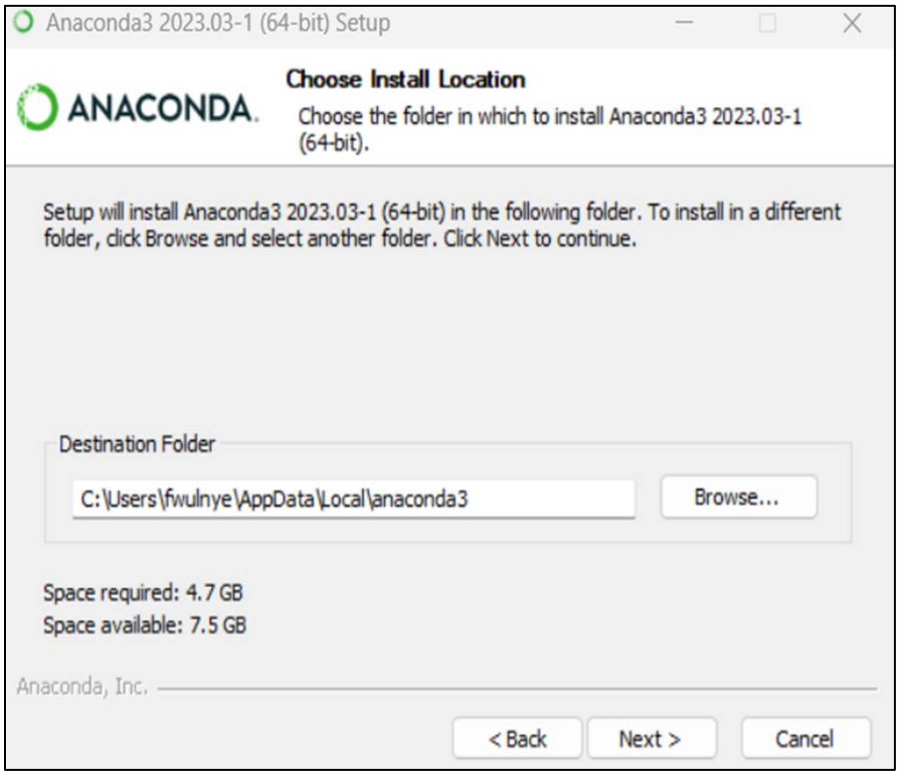


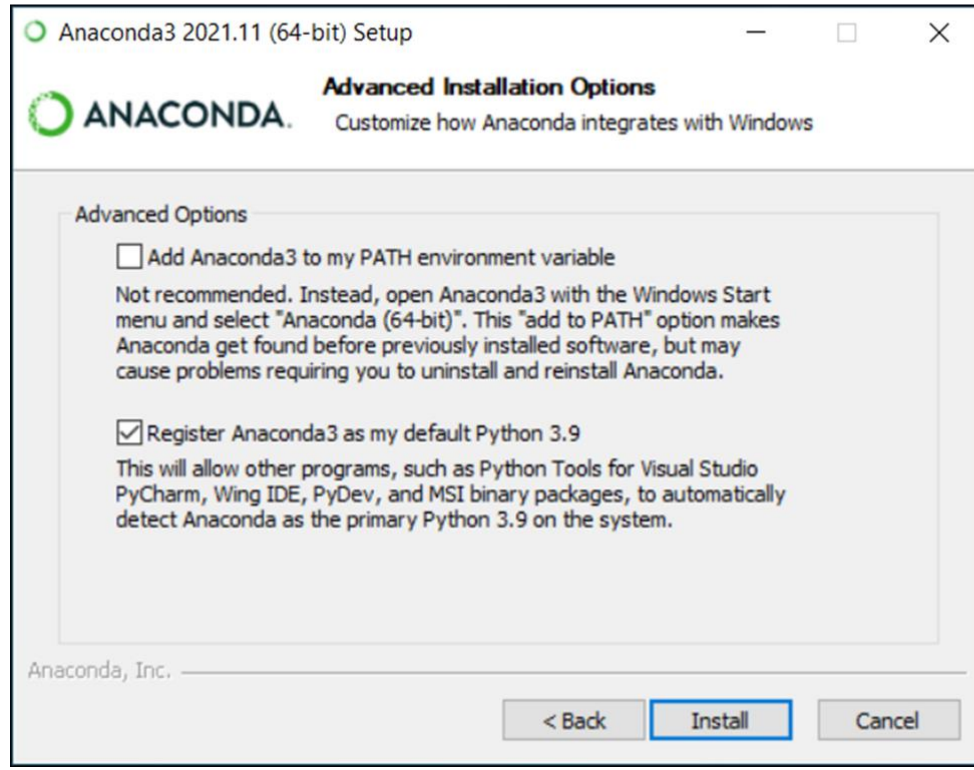
Visit – The above link to download

[Anaconda Download Link]

INSTALLATION OF TOOLS

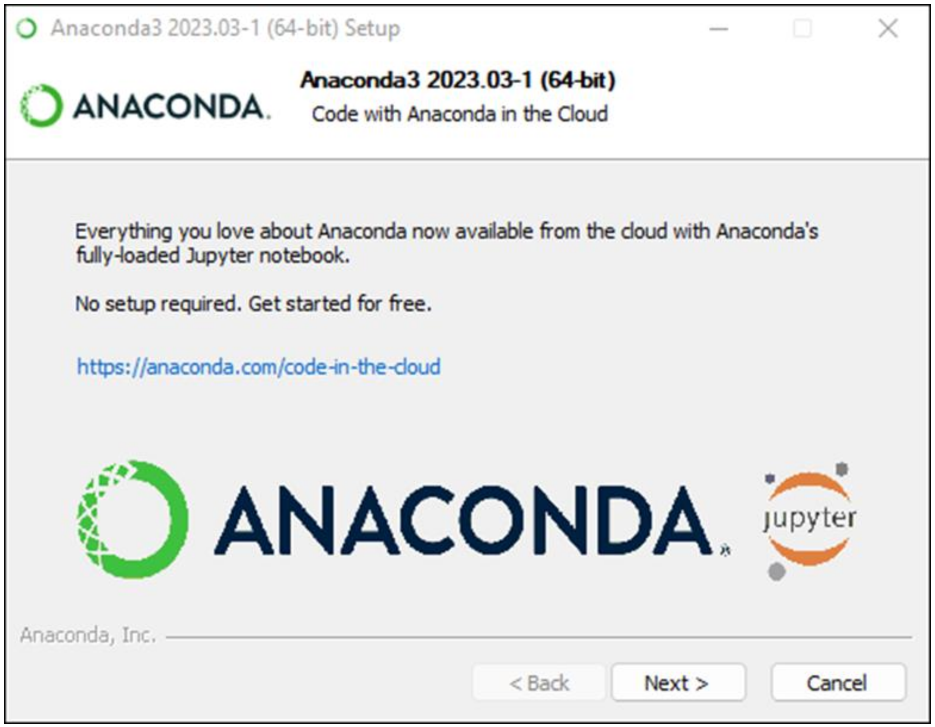
❖ Installation of tools

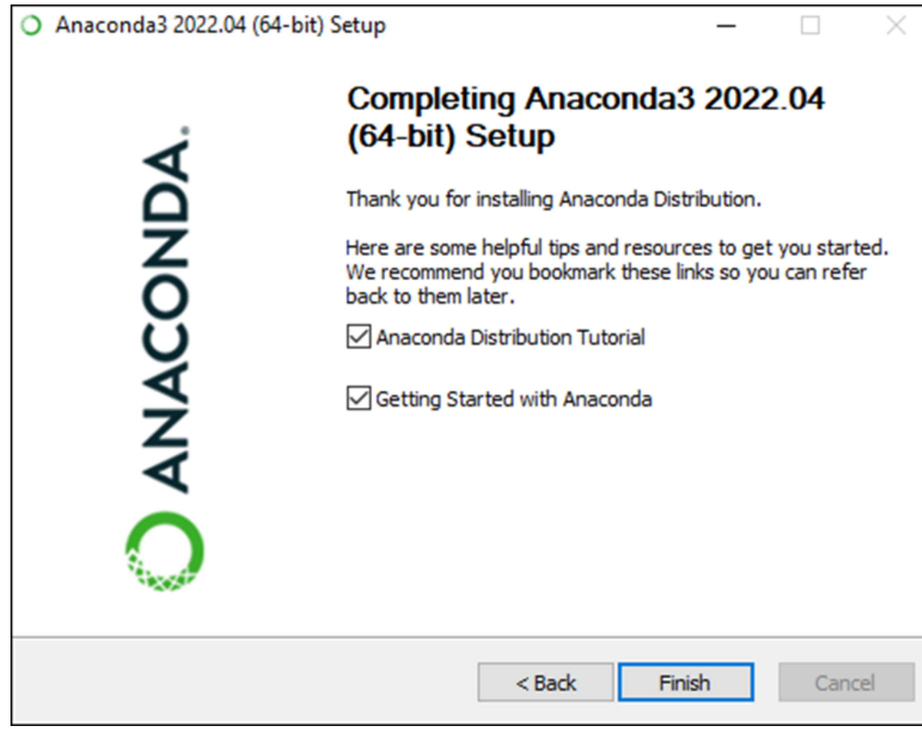




INSTALLATION OF TOOLS

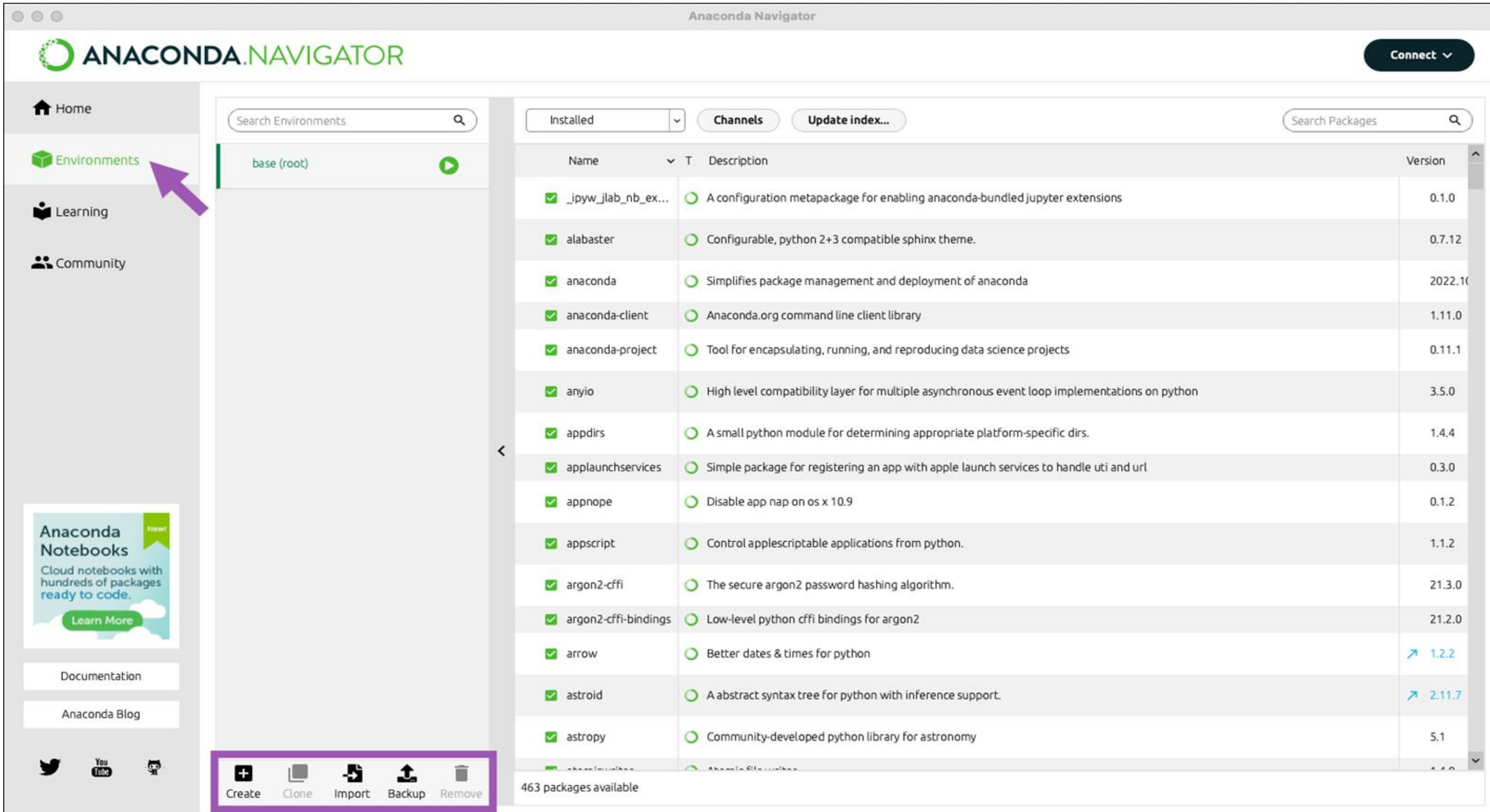
❖ Installation of tools





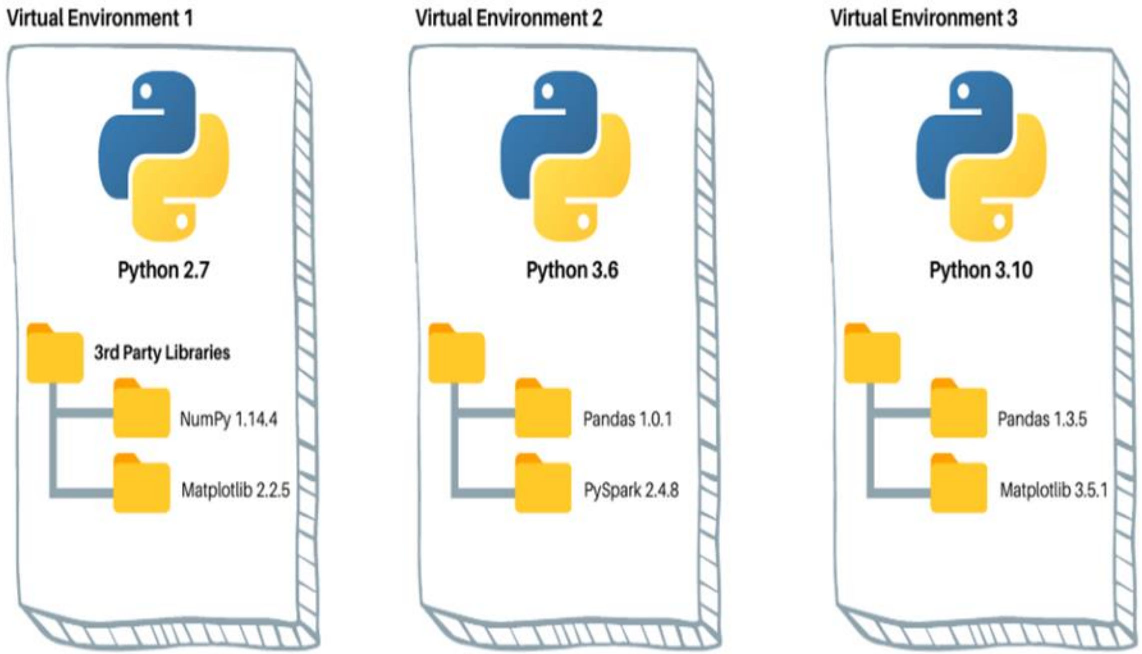
INSTALLATION OF TOOLS

- Installation of tools



INSTALLATION OF TOOLS

❖ Installation of tools



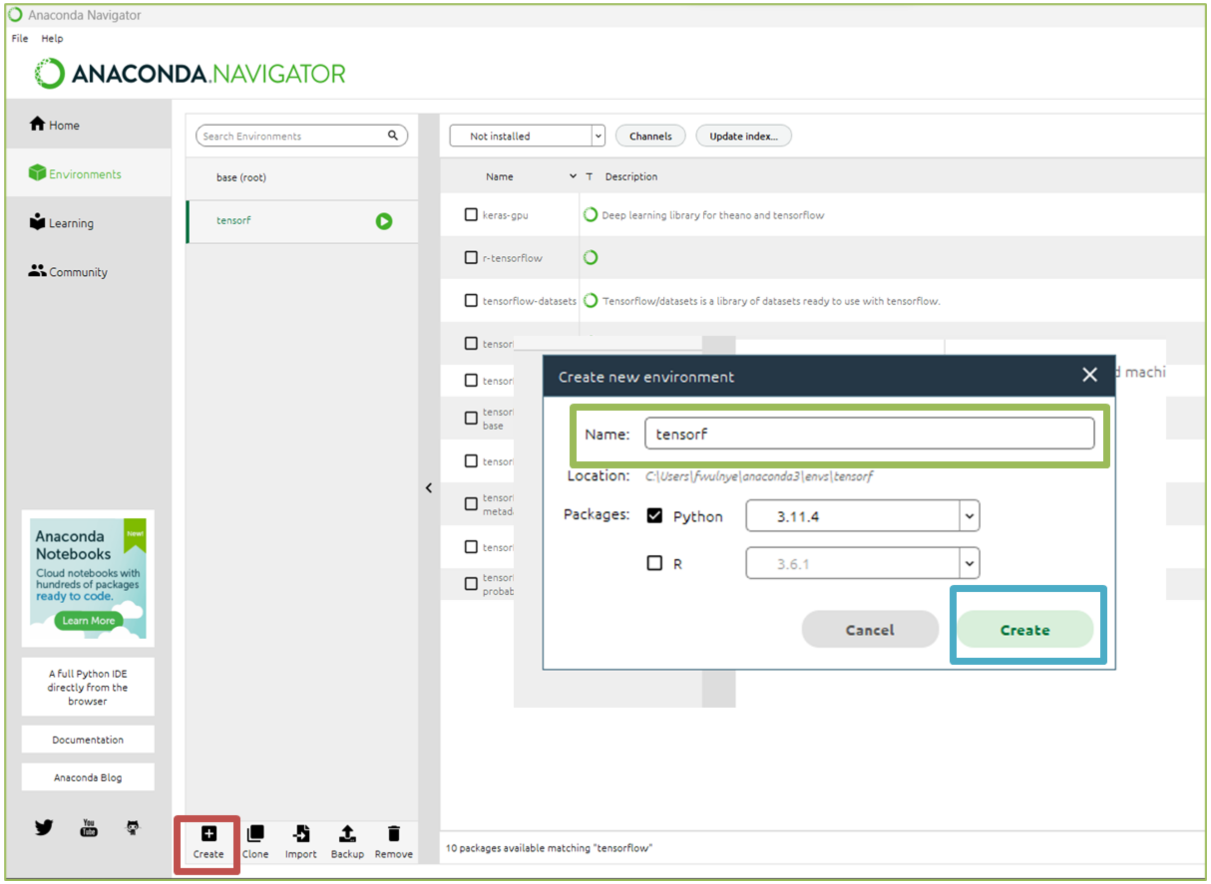
Virtual env name – tensorf

A tool to create isolated Python environments. virtualenv creates a folder which contains all the necessary executables to use the packages that a Python project would need.

GUI – Anaconda Navigator

INSTALLATION OF TOOLS

❖ Installation of tools



Click on Create

Name – Type in the name of your env

Select Python package and version – 3.11.4

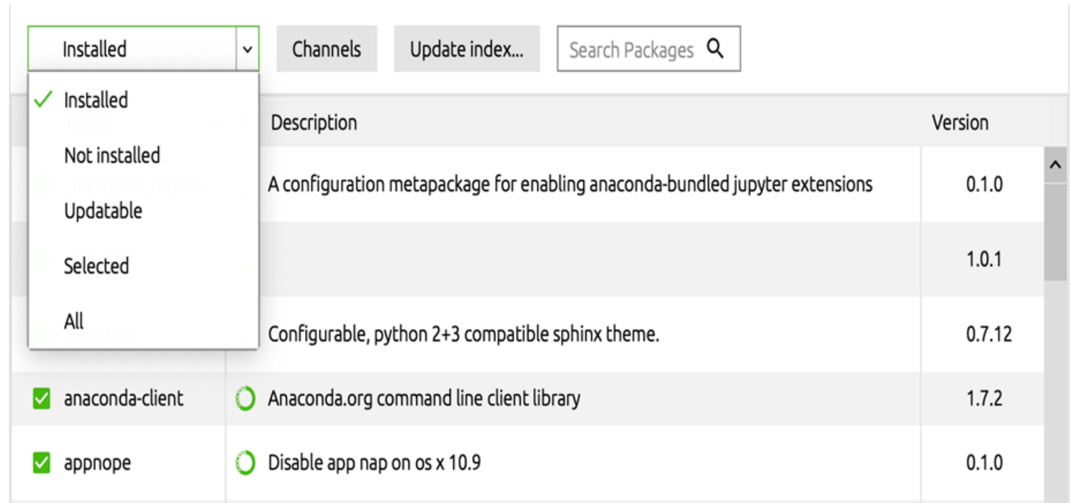
Click Create

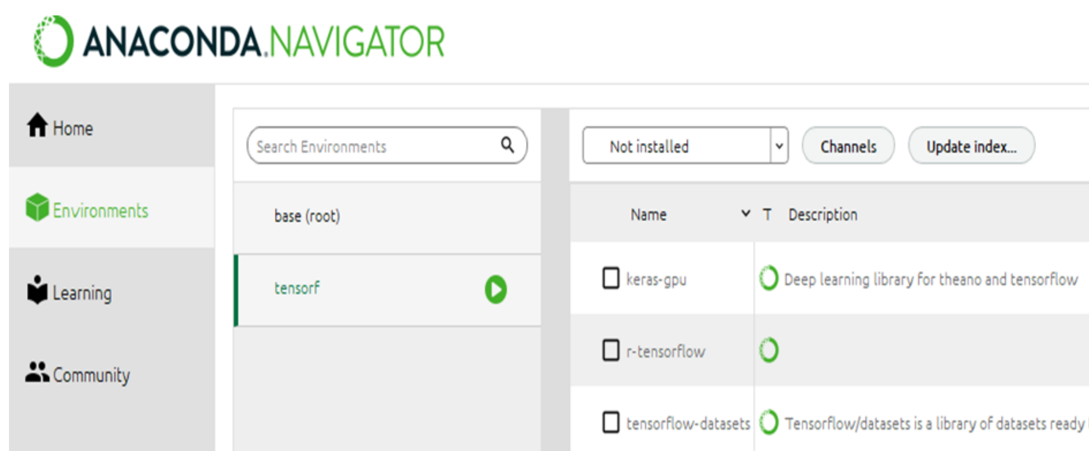
❖ Installation of tools

tensorfl_backup.yaml | libraries.txt ×

libraries.txt

```
name: tensorf
channels:
  - defaults
dependencies:
  - numpy=1.23.5=py310h60c9a35_0
  - numpy-base=1.23.5=py310h04254f7_0
  - pandas=1.5.3=py310h4ed8f06_0
  - scikit-image=0.19.3=py310hd77b12b_1
  - scikit-learn=1.2.1=py310hd77b12b_0
  - seaborn=0.12.2=py310haa95532_0
  - tensorflow=2.10.0=mkl_py310hd99672f_0
  - tensorflow-base=2.10.0=mkl_py310h6a7f48e_0
  - pandas=1.5.3=py310h4ed8f06_0
  - matplotlib=3.7.0=py310haa95532_0
  - matplotlib-base=3.7.0=py310h4ed8f06_0
  - opencv=4.6.0=py310h4ed8f06_3
  - keras==2.12.0
```



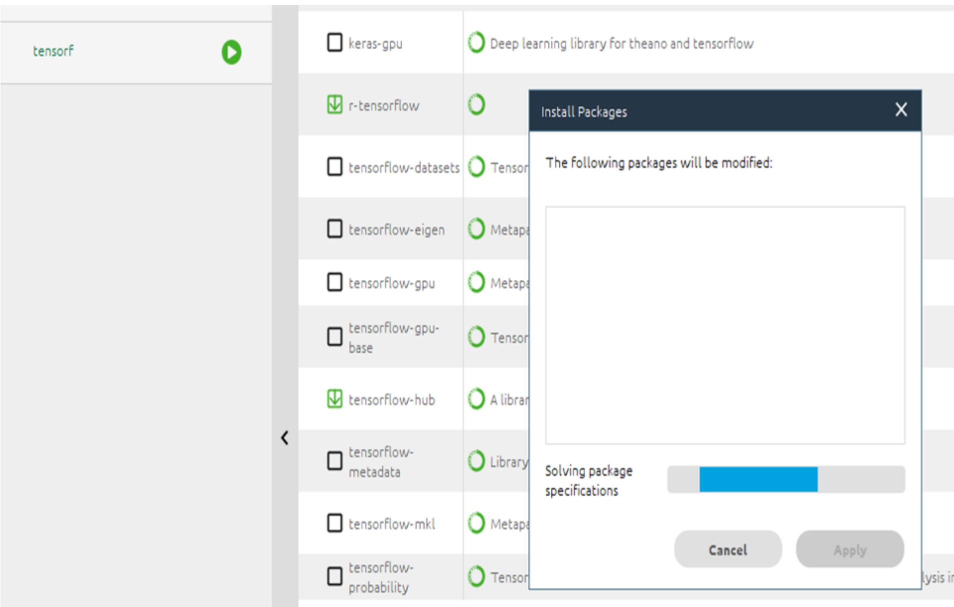


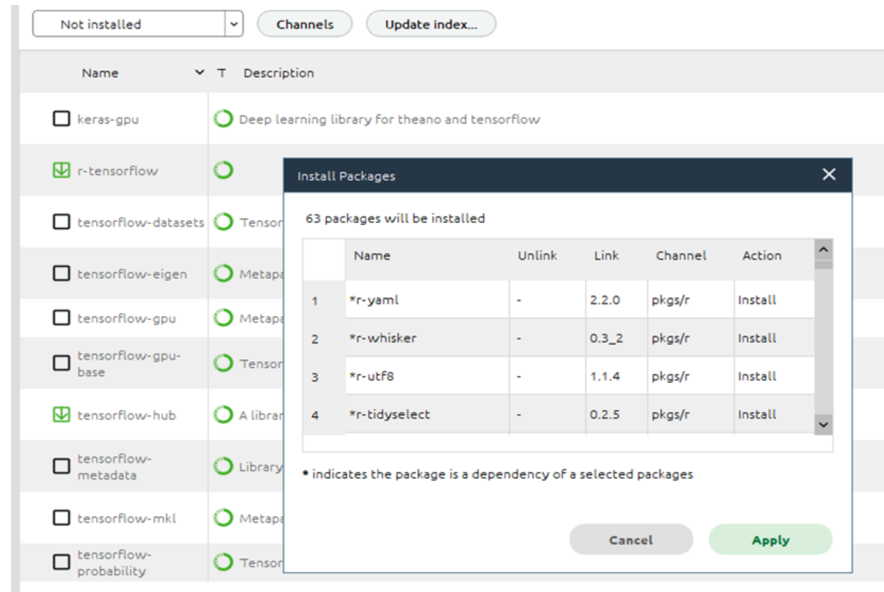
Navigate to the tensorf environment

Select not installed. And search and select for the library names in the library.txt .

INSTALLATION OF TOOLS

❖ Installation of tools



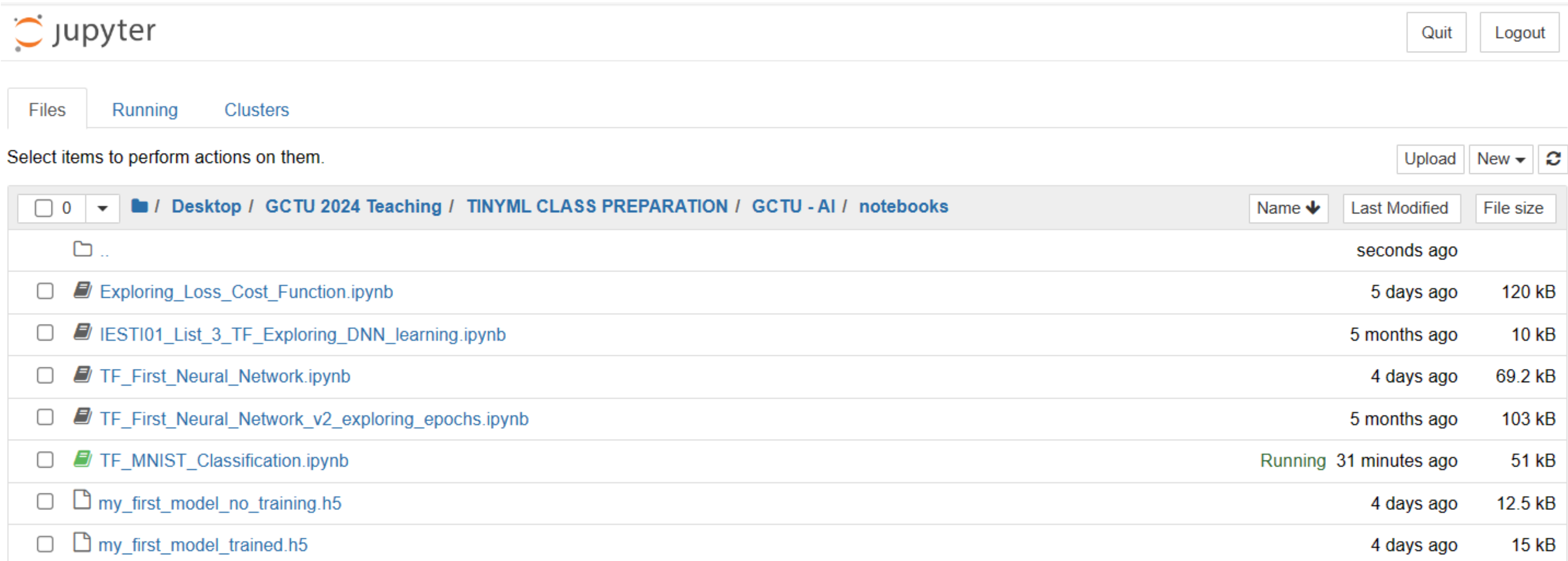


The package specification will be solved

Click Apply to install

NOTEBOOK EXAMPLE

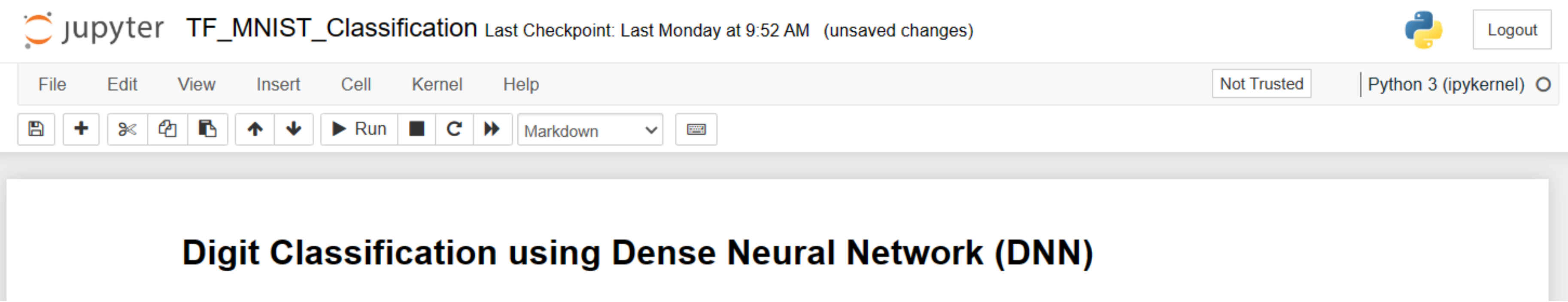
- Notebook Example: MNIST Classification



Note book interface

NOTEBOOK EXAMPLE

- Notebook Example: MNIST Classification



NOTEBOOK EXAMPLE

❖ Notebook Example: MNIST Classification

Import Libraries

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
```

Notebook Example: MNIST Classification

Upload and Explore Dataset

MNIST handwritten digits dataset

The MNIST database of handwritten digits, available from this page, has a training set of 60,000 28x28 grayscale images of the 10 digits along a test set of 10,000 images. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.

In [2]:
```
data = tf.keras.datasets.mnist

(training_images, training_labels), (val_images, val_labels) = data.load_data()
```

In [3]:
```
print(training_images.shape)
print(training_labels.shape)
```
```
(60000, 28, 28)
(60000,)
```

In [4]:
```
print(val_images.shape)
print(val_labels.shape)
```
```
(10000, 28, 28)
(10000,)
```

NOTEBOOK EXAMPLE

- Notebook Example: MNIST Classification

Exploring Labels

In [5]:
```
training_labels
```
Out[5]:
```
array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)
```

It is possible to keep training labels as "numbers", but in this case when compiling the model, you should use: `loss="sparse_categorical_crossentropy"`.

And how about changing labels to categorical?

In [7]:
```
training_labels[:10]
```
Out[7]:
```
array([5, 0, 4, 1, 9, 2, 1, 3, 1, 4], dtype=uint8)
```

In [8]:
```
from tensorflow.keras.utils import to_categorical
to_categorical(training_labels[:10])
```
Out[8]:
```
array([[0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0., 0., 0.]], dtype=float32)
```

When labels are defined as categories, when compiling the model you should use: `loss="categorical_crossentropy"`.

NOTEBOOK EXAMPLE

❖ Notebook Example: MNIST Classification

Exploring images

In [8]:
```
np.set_printoptions(linewidth=200)
print(training_images[2])
```

```
[[  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   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   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   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   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   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   0   0   0   0   0   0   0   0   0   0  67 232  39   0   0   0   0   0]
 [  0   0   0   0  62  81   0   0   0   0   0   0   0   0   0   0   0   0   0   0 120 180  39   0   0   0   0   0]
 [  0   0   0   0 126 163   0   0   0   0   0   0   0   0   0   0   0   0   0   2 153 210  40   0   0   0   0   0]
 [  0   0   0   0 220 163   0   0   0   0   0   0   0   0   0   0   0   0   0  27 254 162   0   0   0   0   0   0]
 [  0   0   0   0 222 163   0   0   0   0   0   0   0   0   0   0   0   0   0 183 254 125   0   0   0   0   0   0]
 [  0   0   0  46 245 163   0   0   0   0   0   0   0   0   0   0   0   0   0 198 254  56   0   0   0   0   0   0]
 [  0   0   0 120 254 163   0   0   0   0   0   0   0   0   0   0   0   0  23 231 254  29   0   0   0   0   0   0]
 [  0   0   0 159 254 120   0   0   0   0   0   0   0   0   0   0   0   0 163 254 216  16   0   0   0   0   0   0]
 [  0   0   0 159 254  67   0   0   0   0   0   0   0   0   0  14  86 178 248 254  91   0   0   0   0   0   0   0]
 [  0   0   0 159 254  85   0   0   0  47  49 116 144 150 241 243 234 179 241 252  40   0   0   0   0   0   0   0]
 [  0   0   0 150 253 237 207 207 207 253 254 250 240 198 143  91  28   5 233 250   0   0   0   0   0   0   0   0]
 [  0   0   0   0 119 177 177 177 177 177  98  56   0   0   0   0   0 102 254 220   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 169 254 137   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 169 254  57   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 169 254  57   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 169 255  94   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 169 254  96   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 169 254 153   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 169 255 153   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  96 254 153   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   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   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   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]]
```

In [9]:
```
training_labels[2]
```

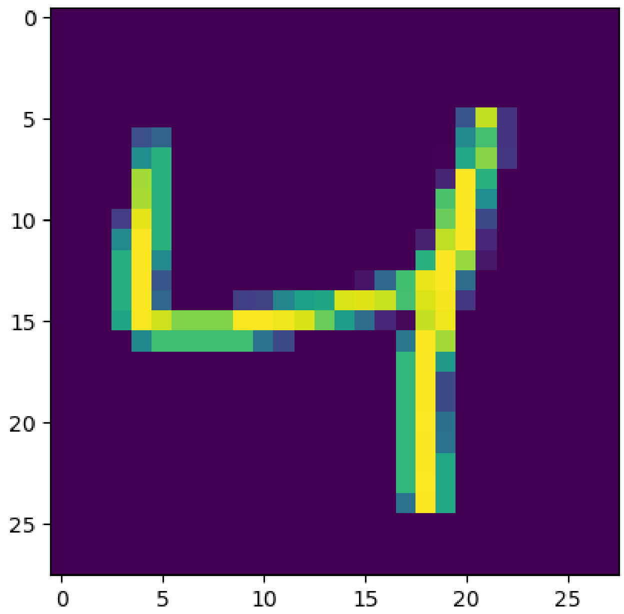
Out[9]: 4

NOTEBOOK EXAMPLE

❖ Notebook Example: MNIST Classification

In [10]:
```
img = 2
print("     Label of image {} is: {}".format(img, training_labels[img]))
plt.imshow(training_images[img]);
```

```
Label of image 2 is: 4
```



In [11]:
```
training_images.max()
```

Out[11]: 255

In [12]:
```
training_images.min()
```

Out[12]: 0

NOTEBOOK EXAMPLE

❖ Notebook Example: MNIST Classification

Preprocessing Data

Normalizing Data: We notice that all of the values in the number are between 0 and 255. If we are training a neural network, for various reasons it's easier if we treat all values as between 0 and 1, a process called 'normalizing'.

In [13]:
```
training_images  = training_images / 255.0
val_images = val_images / 255.0
```

In [14]:
```
print(training_images[2])
```

```
[[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.         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.         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.         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.         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.         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.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.2627451  0.90980392 0.15294118 0.         0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.24313725 0.31764706 0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.47058824 0.70588235 0.15294118 0.         0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.49411765 0.63921569 0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
```

In [15]:
```
training_images.max()
```

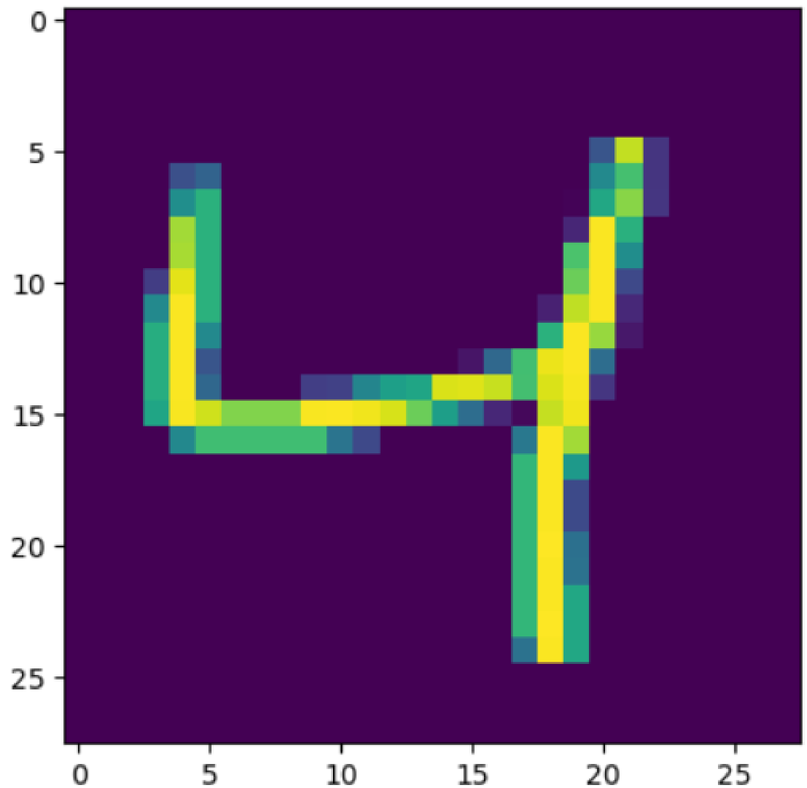
Out[15]: 1.0

NOTEBOOK EXAMPLE

❖ Notebook Example: MNIST Classification

In [12]:
```
img = 2
print("     Label of image {} is: {}".format(img, training_labels[img]))
plt.imshow(training_images[img]);
```

```
Label of image 2 is: 4
```



NOTEBOOK EXAMPLE

❖ Notebook Example: MNIST Classification

Define and Compile Model

In [7]:
```
model = tf.keras.models.Sequential([tf.keras.layers.Flatten(input_shape=(28,28)),
                                    tf.keras.layers.Dense(20, activation=tf.nn.relu),
                                    tf.keras.layers.Dense(10, activation=tf.nn.softmax)])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 20)                15700

 dense_1 (Dense)             (None, 10)                210

=================================================================
Total params: 15,910
Trainable params: 15,910
Non-trainable params: 0
_________________________________________________________________
```

In [8]:
```
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy', # Labels are not as an array
    metrics=['accuracy'] # Calculates how often predictions equal labels
    )
```

NOTEBOOK EXAMPLE

- Notebook Example: MNIST Classification

Train the model

In [9]:

```
%%time
history = model.fit(
    training_images,
    training_labels,
    epochs=20,
    )
```

NOTEBOOK EXAMPLE

- Notebook Example: MNIST Classification

```
Epoch 1/20
1875/1875 [==============================] - 3s 1ms/step - loss: 0.4069 - accuracy: 0.8868
Epoch 2/20
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2317 - accuracy: 0.9342
Epoch 3/20
1875/1875 [==============================] - 2s 1ms/step - loss: 0.1939 - accuracy: 0.9438
Epoch 4/20
1875/1875 [==============================] - 2s 1ms/step - loss: 0.1702 - accuracy: 0.9509
Epoch 5/20
1875/1875 [==============================] - 2s 1ms/step - loss: 0.1543 - accuracy: 0.9548
Epoch 6/20
1875/1875 [==============================] - 2s 1ms/step - loss: 0.1412 - accuracy: 0.9582
Epoch 7/20
1875/1875 [==============================] - 2s 1ms/step - loss: 0.1308 - accuracy: 0.9614
Epoch 8/20
1875/1875 [==============================] - 2s 1ms/step - loss: 0.1229 - accuracy: 0.9635
Epoch 9/20
1875/1875 [==============================] - 2s 1ms/step - loss: 0.1175 - accuracy: 0.9648
Epoch 10/20
1875/1875 [==============================] - 2s 1ms/step - loss: 0.1116 - accuracy: 0.9664
Epoch 11/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.1066 - accuracy: 0.9682
Epoch 12/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.1035 - accuracy: 0.9693
Epoch 13/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0997 - accuracy: 0.9702
Epoch 14/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0958 - accuracy: 0.9707
Epoch 15/20
1875/1875 [==============================] - 3s 1ms/step - loss: 0.0926 - accuracy: 0.9723
Epoch 16/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0912 - accuracy: 0.9720
Epoch 17/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0879 - accuracy: 0.9736
Epoch 18/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0865 - accuracy: 0.9737
Epoch 19/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0842 - accuracy: 0.9745
Epoch 20/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0823 - accuracy: 0.9756
CPU times: total: 4min 23s
Wall time: 55.7 s
```

NOTEBOOK EXAMPLE

❖ Notebook Example: MNIST Classification

Inspecting the model

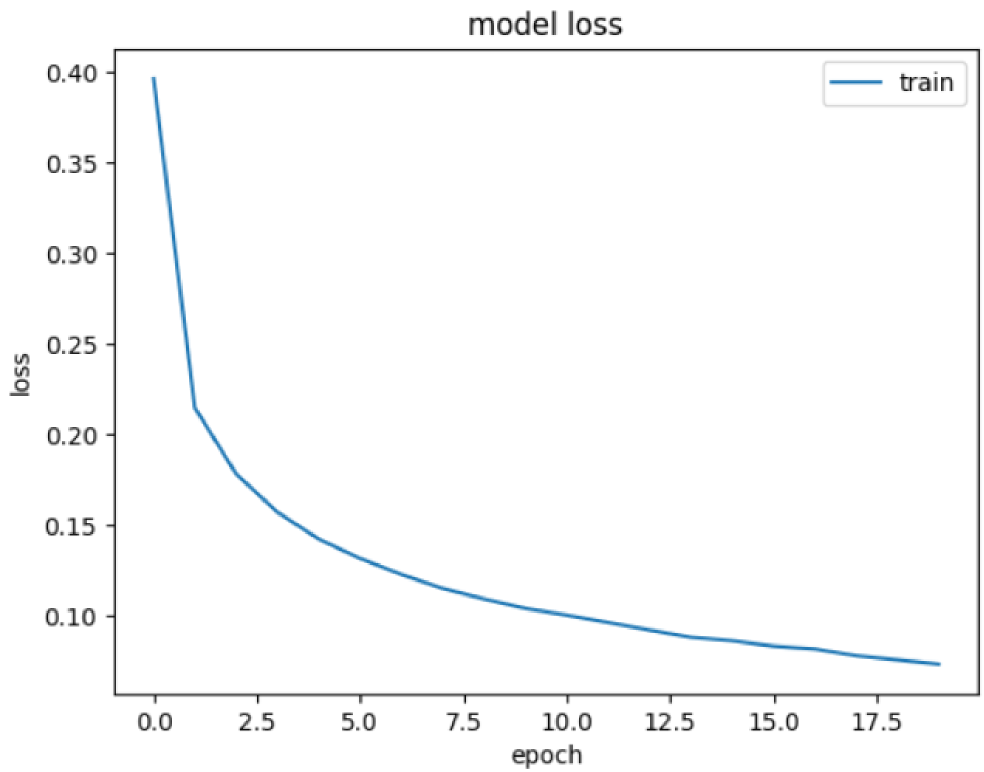
```
In [10]:  train_eval = model.evaluate(training_images, training_labels)
          print ("Training data Accuracy: {:.1f}%".format(train_eval[1]*100))

1875/1875 [==============================] - 2s 1ms/step - loss: 0.0732 - accuracy: 0.9783
Training data Accuracy: 97.8%
```

NOTEBOOK EXAMPLE

❖ Notebook Example: MNIST Classification

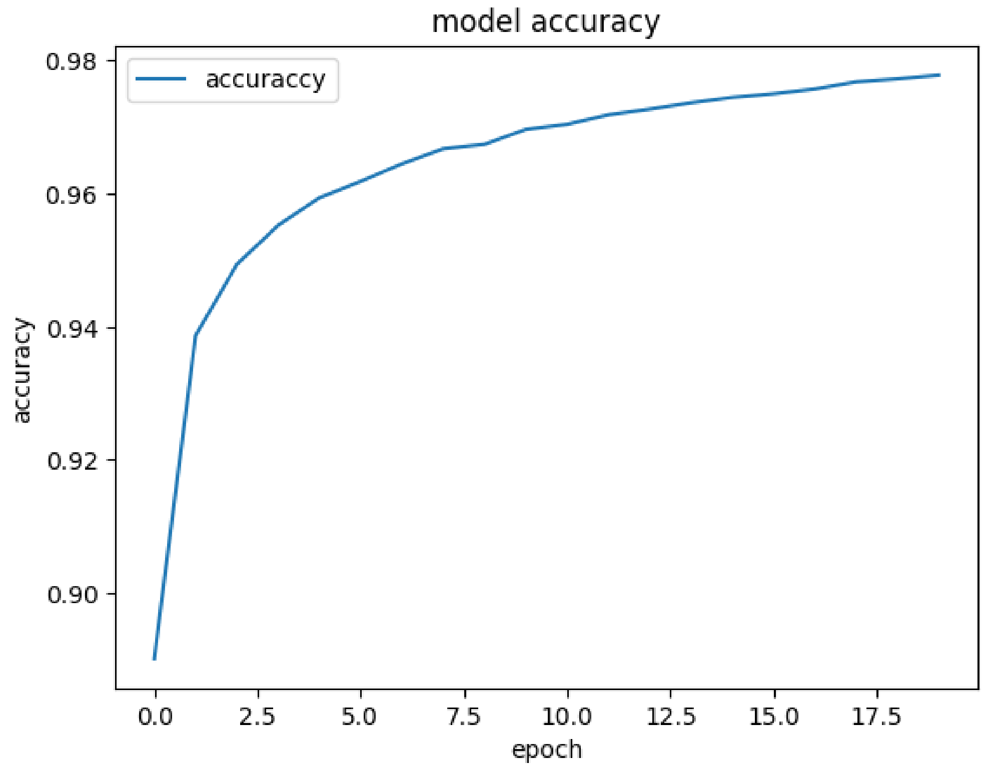
```
[22] plt.plot(history.history['loss'])
     plt.title('model loss')
     plt.ylabel('loss')
     plt.xlabel('epoch')
     plt.legend(['train'], loc='upper right')
     plt.show()
```



NOTEBOOK EXAMPLE

❖ Notebook Example: MNIST Classification

```
[23] plt.plot(history.history['accuracy'])
     plt.title('model accuracy')
     plt.ylabel('accuracy')
     plt.xlabel('epoch')
     plt.legend(['accuraccy'], loc='upper left')
     plt.show()
```



NOTEBOOK EXAMPLE

❖ Notebook Example: MNIST Classification

Testing the trained model

Using `model.evaluate`, you can get metrics for a test set. In this case we only have a training set and a validation set, so we can try it out with the validation set. The accuracy will be slightly lower, at maybe 96%. This is because the model hasn't previously seen this data and may not be fully generalized for all data. Still it's a pretty good score. You can also predict images, and compare against their actual label. The [0] image in the set is a number 7, and here you can see that neuron 7 has a 9.9e-1 (99%+) probability, so it got it right!

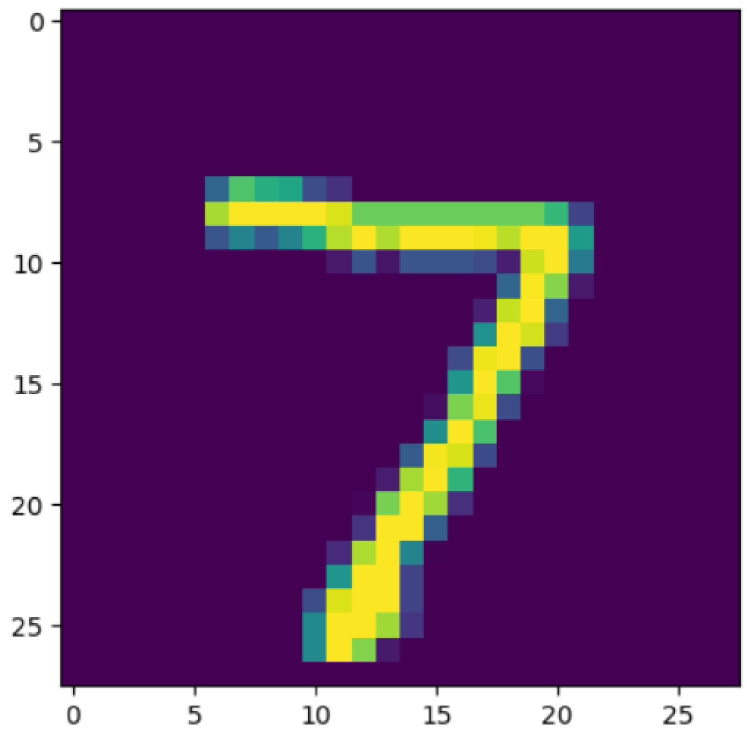
```
[24] test_eval = model.evaluate(val_images, val_labels)
     print ("Testing data Accuracy: {:.1f}%".format(test_eval[1]*100))
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.1450 - accuracy: 0.9625
Testing data Accuracy: 96.2%
```

NOTEBOOK EXAMPLE

- Notebook Example: MNIST Classification

```
[25] plt.imshow(val_images[0]);
```




```
[26] print(val_labels[0])
```

```
7
```

NOTEBOOK EXAMPLE

❖ Notebook Example: MNIST Classification

```
[27] classifications = model.predict(val_images)
     print(classifications[0])

313/313 [==============================] - 1s 2ms/step
[7.2781484e-05 3.8751616e-11 6.5620356e-05 1.0001573e-03 6.9909703e-12 1.5672607e-05 8.4830062e-15 9.9873394e-01 1.7328633e-05 9.4513402e-05]
```

```
[28] # Returns the indices of the maximum values along an axis.
     np.argmax(classifications[0])

7
```