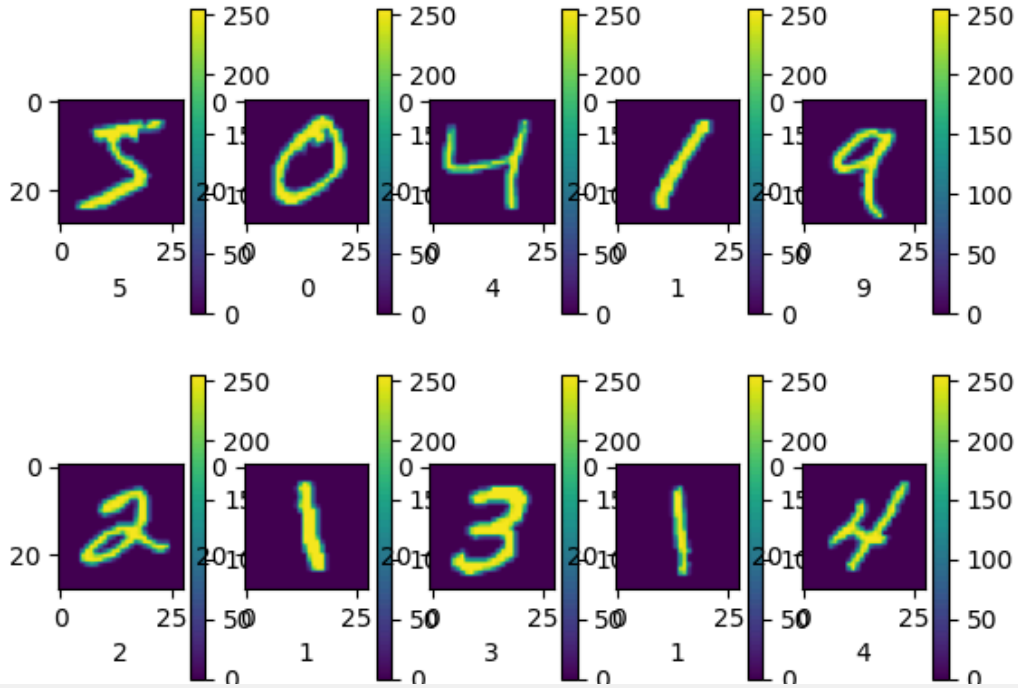
```
from tensorflow.keras import datasets
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.models import Sequential
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import tensorflow as tf

# Download the MNIST dataset
load_data = datasets.mnist.load_data()
(train_images, train_labels), (test_images, test_labels) = load_data
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
for i in range(10):
  plt.subplot(2, 5, i+1)
  plt.imshow(train_images[i])
  plt.xlabel(train_labels[i])
  plt.colorbar()
plt.show()
```




```
min(train_labels),max(train_labels)
```

```
(0, 9)
```

```
class_names= ["one","two","three","four","five","six","seven","eight","nine"]
```

```
model = Sequential([
  Flatten(input_shape=(28, 28)),
  Dense(128, activation='relu'),
  Dense(64,activation="relu"),
  Dense(32,activation="relu"),
  Dense(16,activation="relu"),
  Dense(8,activation="relu"),
  Dense(10, activation='softmax')
])
model.summary()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_dim` argume
  super().__init__(**kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 128)	100,480
dense_1 (Dense)	(None, 64)	8,256
dense_2 (Dense)	(None, 32)	2,080
dense_3 (Dense)	(None, 16)	528
dense_4 (Dense)	(None, 8)	136
dense_5 (Dense)	(None, 10)	90

Total params: 111,570 (435.82 KB)
Trainable params: 111,570 (435.82 KB)

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history=model.fit(train_images,
          train_labels,
          epochs=10,
          validation_data=(test_images,test_labels),
          batch_size=64,
          verbose=True)
```

```
Epoch 1/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.5474 - loss: 2.2453 - val_accuracy: 0.8424 - val_loss: 0.5664
Epoch 2/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.8935 - loss: 0.4491 - val_accuracy: 0.9299 - val_loss: 0.2963
Epoch 3/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - accuracy: 0.9414 - loss: 0.2503 - val_accuracy: 0.9495 - val_loss: 0.2234
Epoch 4/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.9561 - loss: 0.1828 - val_accuracy: 0.9554 - val_loss: 0.1966
Epoch 5/10
```

```
938/938 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.9625 - loss: 0.1473 - val_accuracy: 0.9581 - val_loss: 0.1749
Epoch 6/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.9684 - loss: 0.1201 - val_accuracy: 0.9634 - val_loss: 0.1387
Epoch 7/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.9726 - loss: 0.1017 - val_accuracy: 0.9657 - val_loss: 0.1360
Epoch 8/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9767 - loss: 0.0880 - val_accuracy: 0.9635 - val_loss: 0.1494
Epoch 9/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.9796 - loss: 0.0747 - val_accuracy: 0.9649 - val_loss: 0.1412
Epoch 10/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9811 - loss: 0.0689 - val_accuracy: 0.9662 - val_loss: 0.1371
```

Model Evaluation

- Accuracy
- precission
- Recall
- F1 score

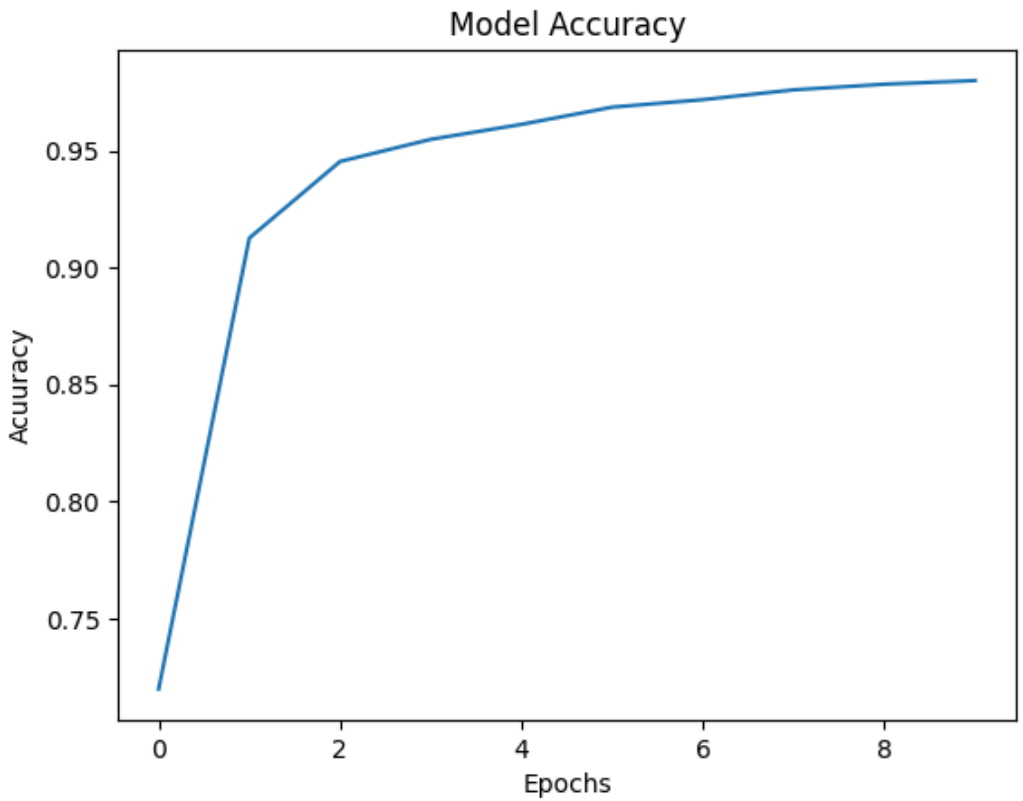
```
history.history
```

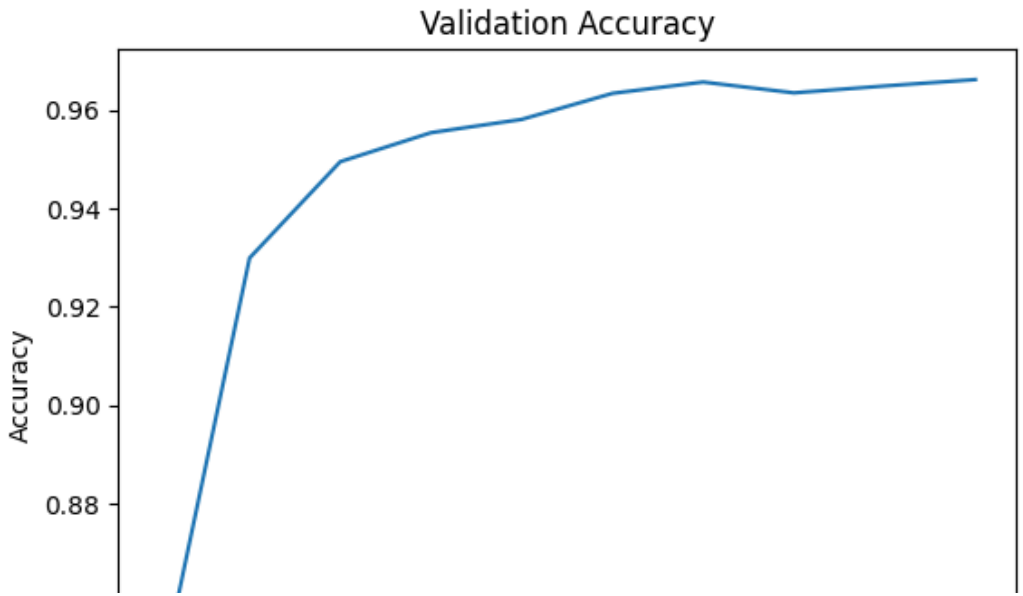
```
{'accuracy': [0.7198166847229004,
  0.9126166701316833,
  0.9453333616256714,
  0.954800009727478,
  0.9611999988555908,
  0.9685666561126709,
  0.9717666506767273,
  0.9759666919708252,
  0.9783666729927063,
  0.9799166917800903],
 'loss': [1.1107251644134521,
  0.3827242851257324,
  0.23274429142475128,
  0.18198643624782562,
  0.1486896127462387,
  0.11872318387031555,
  0.1058875322341919,
  0.0886390283703804,
  0.079163558781147,
```

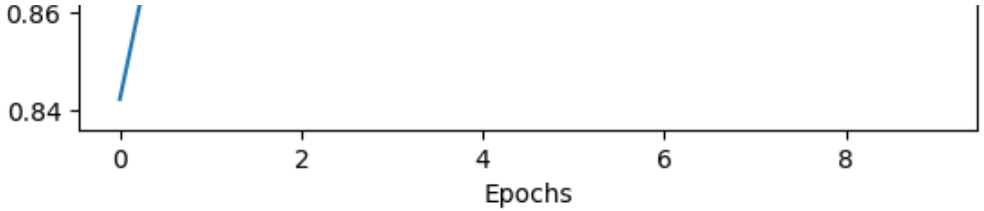
```
          0.07147206366062164],
         'val_accuracy': [0.8424000144004822,
          0.9298999905586243,
          0.9495000243186951,
          0.9553999900817871,
          0.9581000208854675,
          0.9634000062942505,
          0.9656999707221985,
          0.9635000228881836,
          0.964900016784668,
          0.9661999940872192],
         'val_loss': [0.5664058327674866,
          0.2962717115879059,
          0.22335302829742432,
          0.19664059579372406,
          0.17486678063869476,
          0.13872383534908295,
          0.13603471219539642,
          0.14944811165332794,
          0.14118197560310364,
          0.13705363869667053]}
```

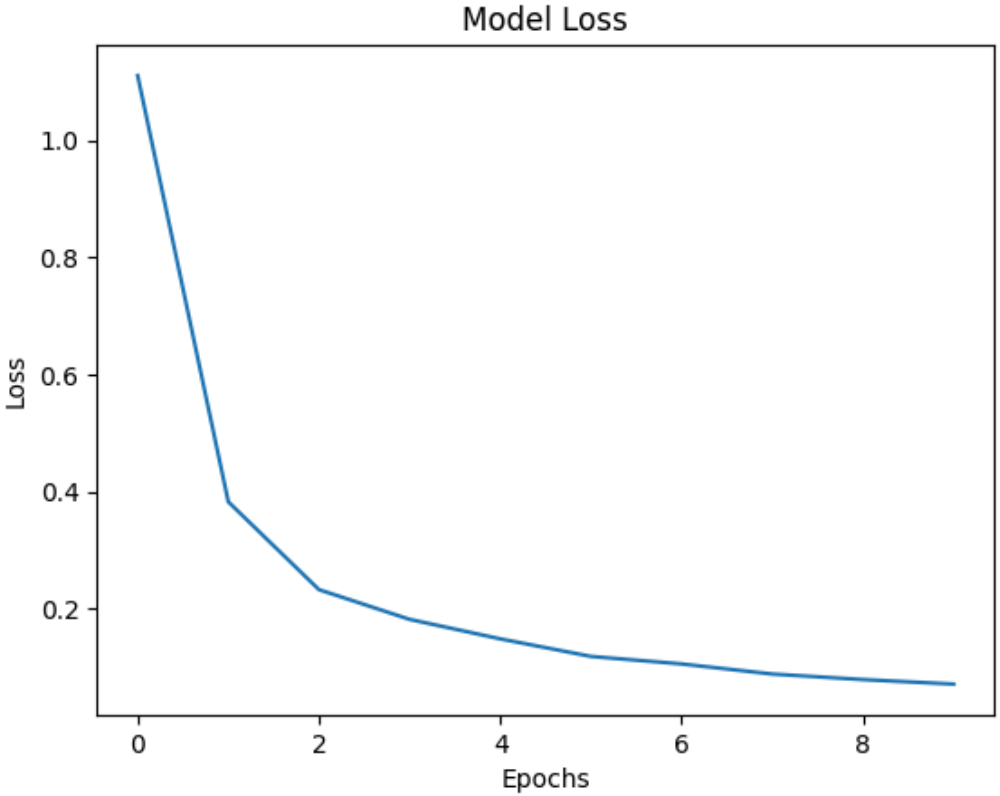
```
plt.plot(history.history["accuracy"])
plt.title("Model Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Acuuracy")
plt.show()

plt.plot(history.history["val_accuracy"])
plt.title("Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.show()

plt.plot(history.history["loss"])
plt.title("Model Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.show()

plt.plot(history.history["val_loss"])
plt.title("Validation Loss")
```

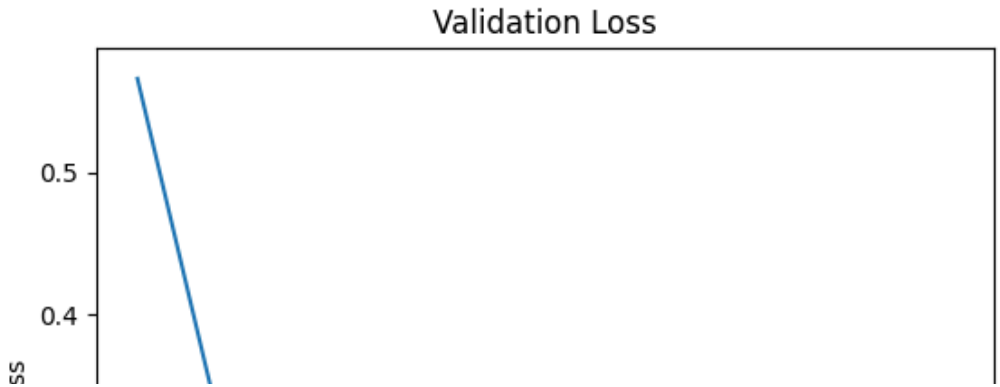
```
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.show()
```

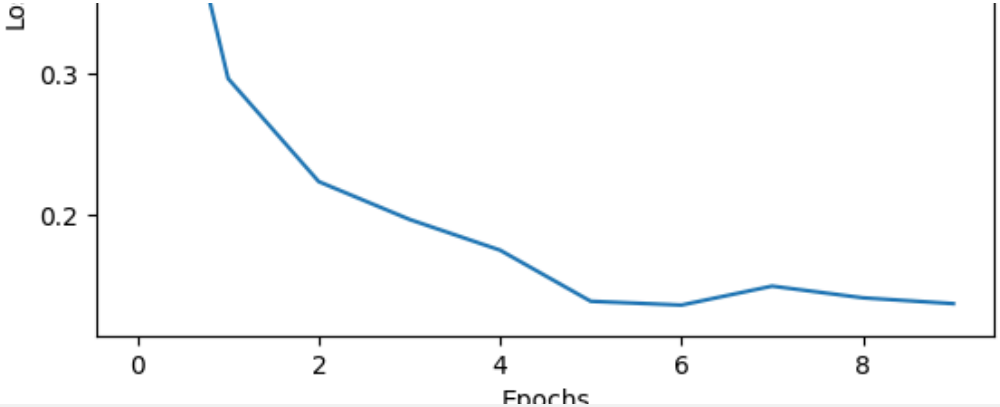

Model Accuracy
Acuuracy
0.95
0.90
0.85
0.80
0.75
0
2
4
6
8
Epochs


Validation Accuracy
Accuracy
0.96
0.94
0.92
0.90
0.88


0.86
0.84
0
2
4
6
8
Epochs


Model Loss
1.0
0.8
0.6
0.4
0.2
Loss
0
2
4
6
8
Epochs


Validation Loss
0.5
0.4


0.3
0.2
0
2
4
6
8

```
test_loss,test_accuracy=model.evaluate(test_images,test_labels)
print("Test Accuracy:",test_accuracy)
print("Test Loss:",test_loss)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9624 - loss: 0.1539
Test Accuracy: 0.9661999940872192
Test Loss: 0.1370537281036377
```

```
image_predict= model.predict(test_images)
image_predict
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
array([[3.4812631e-15, 4.4337839e-10, 4.7836220e-05, ..., 9.9991673e-01,
        7.3521193e-14, 1.0101631e-05],
       [4.2549104e-07, 7.1857584e-04, 9.9656129e-01, ..., 4.2690412e-04,
        2.9041953e-04, 7.2650175e-07],
       [6.2745284e-19, 9.9876690e-01, 1.0123138e-04, ..., 8.5591331e-05,
        1.1160931e-05, 8.3217036e-04],
       ...,
       [8.0223160e-08, 1.3318642e-06, 8.7898127e-07, ..., 2.6745378e-04,
        4.2092409e-11, 2.6647868e-03],
```