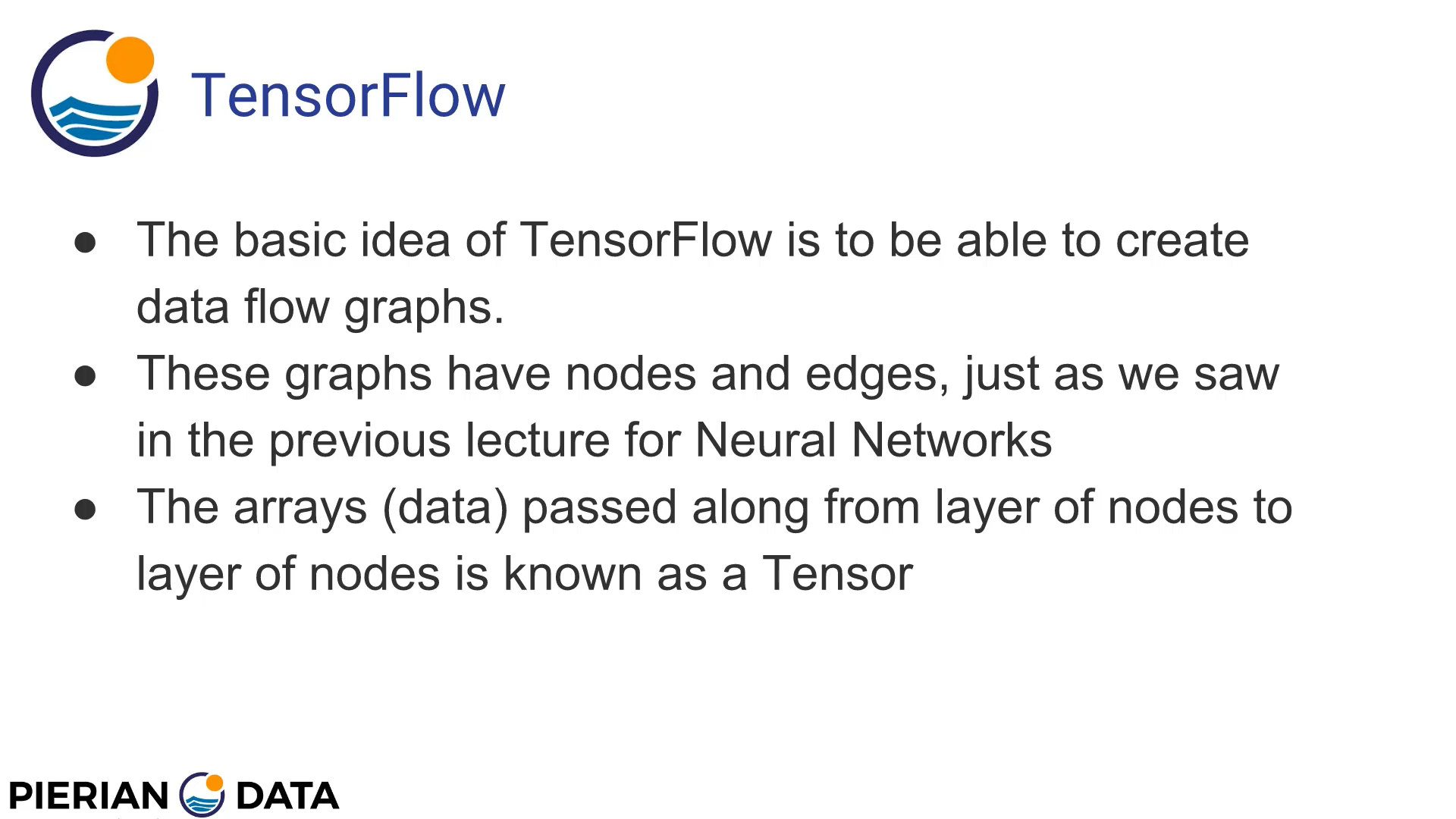
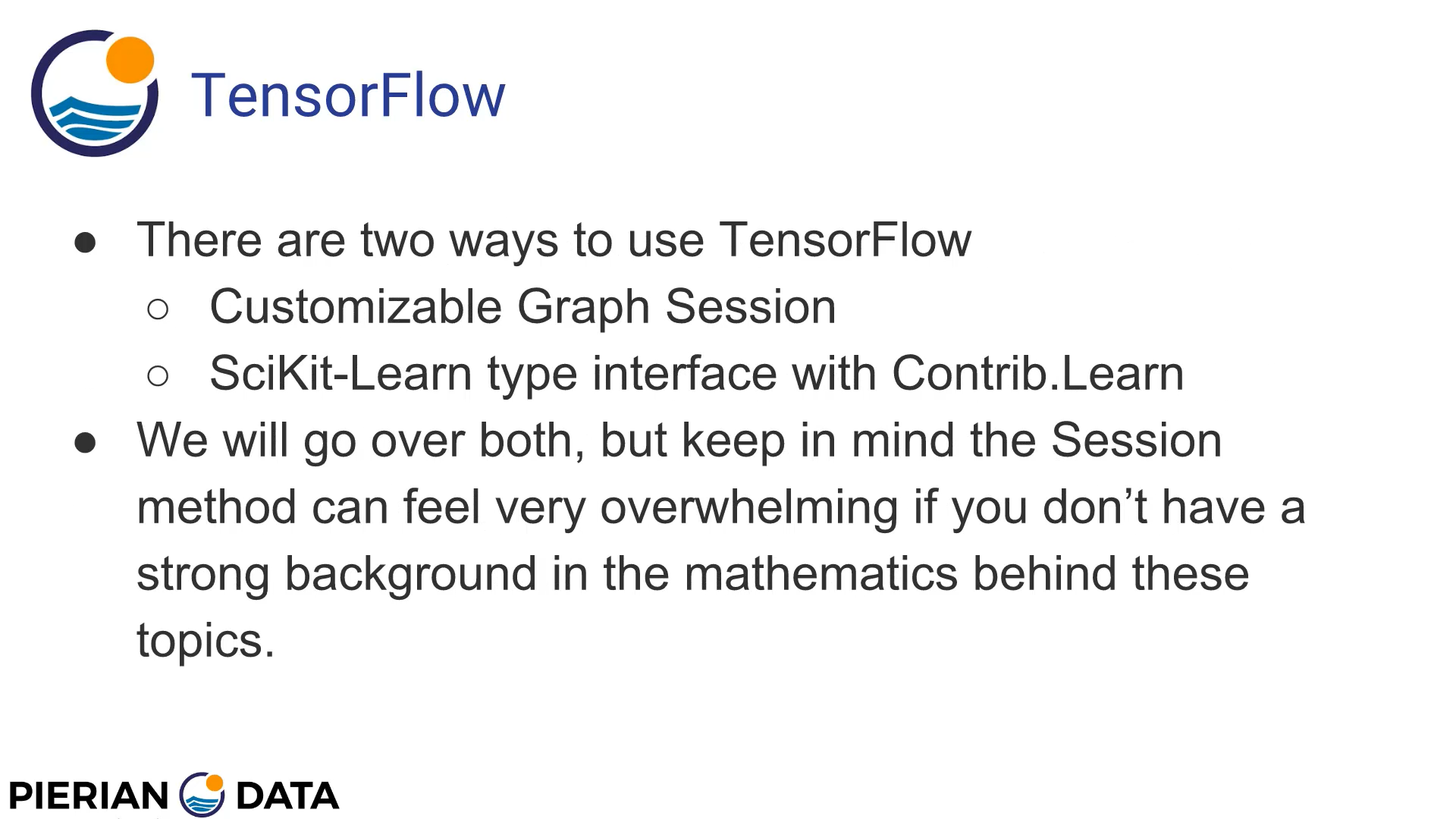
# Tensorflow Basics





# Creating constants in tensorflow

Let's show how to create a simple constant with Tensorflow, which TF stores as a tensor object

import tensorflow as tf  
hello = tf.constant("Hello World")  
print(type(hello))  
print(hello)

output-

<class 'tensorflow.python.framework.ops.Tensor'>

Tensor("Const:0", shape=(), dtype=string)

# Tensorflow running sessions

Now you can create a TensorFlow Session, which is a class for running TensorFlow operations.

A Session object encapsulates the environment in which Operation objects are executed, and Tensor objects are evaluated. For example

import tensorflow as tf  
hello = tf.constant("Hello World")  
x = tf.constant(100)  
sess = tf.Session()  
print(sess.run(x))  
print(sess.run(hello))  
print(type(sess.run(hello)))

output-

WARNING: Logging before flag parsing goes to stderr.

W0824 10:16:15.094213 3076 deprecation\_wrapper.py:119] From E:/Projects/Python/NeuralPractice/FirstExercise.py:4: The name tf.Session is deprecated. Please use tf.compat.v1.Session instead.

2019-08-24 10:16:15.095141: I tensorflow/core/platform/cpu\_feature\_guard.cc:142] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2

100

b'Hello World'

<class 'bytes'>

# Tensorflow Operations

You can line up multiple Tensorflow operations in to be run during a session

import tensorflow as tf  
x = tf.constant(2)  
y = tf.constant(3)  
with tf.compat.v1.Session() as sess:  
 print('Operations with Constants')  
 print('Addition',sess.run(x+y))  
 print('Subtraction',sess.run(x-y))  
 print('Multiplication',sess.run(x\*y))  
 print('Division',sess.run(x/y))

output-

Operations with Constants

Addition 5

Subtraction -1

Multiplication 6

Division 0.6666666666666666

# Tensorflow Placeholder

You may not always have the constants right away, and you may be waiting for a constant to appear after a cycle of operations. **tf.placeholder** is a tool for this. It inserts a placeholder for a tensor that will be always fed.

**Important**: This tensor will produce an error if evaluated. Its value must be fed using the feed\_dict optional argument to Session.run(), Tensor.eval(), or Operation.run(). For example, for a placeholder of a matrix of floating point numbers:

x = tf.placeholder(tf.float32, shape=(1024, 1024))

Here is an example for integer placeholders

import tensorflow as tf  
x = tf.compat.v1.placeholder(tf.int32)  
y = tf.compat.v1.placeholder(tf.int32)  
print(type(x))  
print(x)  
# Defining operations  
add = tf.add(x,y)  
sub = tf.subtract(x,y)  
mul = tf.multiply(x,y)  
# Running operations with variable imput  
d = {x:20,y:30}  
with tf.compat.v1.Session() as sess:  
 print('Operations with Constants')  
 print('Addition',sess.run(add,feed\_dict=d))  
 print('Subtraction',sess.run(sub,feed\_dict=d))  
 print('Multiplication',sess.run(mul,feed\_dict=d))

output-

<class 'tensorflow.python.framework.ops.Tensor'>

Tensor("Placeholder:0", dtype=int32)

Operations with Constants

Addition 50

Subtraction -10

Multiplication 600

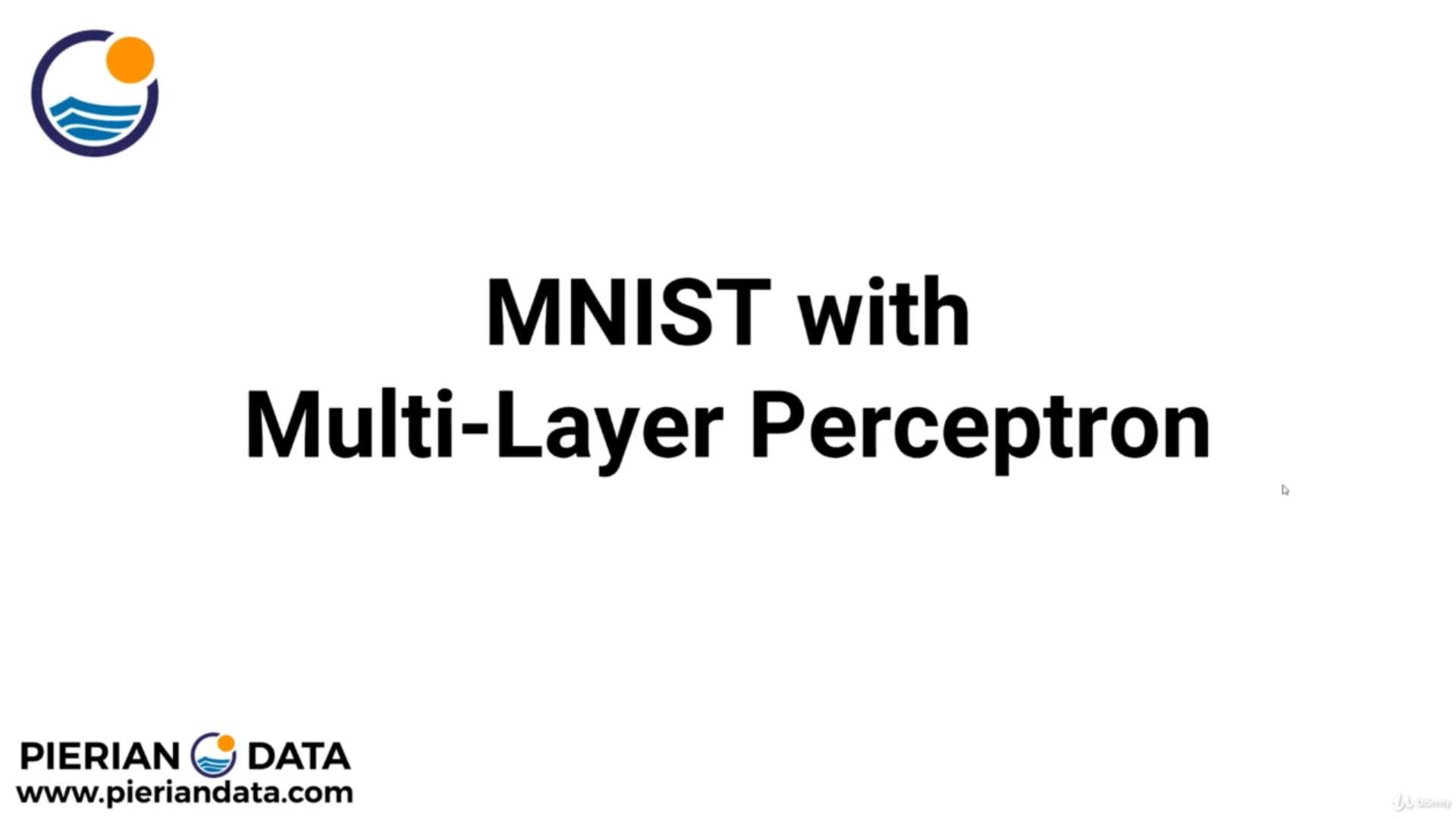
Now let's see an example of a more complex operation, using Matrix Multiplication.

import tensorflow as tf  
import numpy as np  
# Make sure to use floats here, int64 will cause an error.  
a = np.array([[5.0,5.0]]) # 1\*2 matrix  
b = np.array([[2.0],[2.0]]) # 2\*1 matrix  
mat1 = tf.constant(a)  
mat2 = tf.constant(b)  
# The matrix multiplication operation:  
matrix\_multi = tf.matmul(mat1,mat2) # Also a tensor object  
with tf.compat.v1.Session() as sess:  
 result = sess.run(matrix\_multi)  
 print(result)

output-

[[20.]]

# Tensorflow using MNIST data with Multi-layer perceptron





## Importing data sets

import tensorflow as tf  
from tensorflow.examples.tutorials.mnist import input\_data  
mnist = input\_data.read\_data\_sets("E:/Py-DS-ML-Bootcamp-master/Refactored\_Py\_DS\_ML\_Bootcamp-master/22-Deep Learning/MNIST\_data/",one\_hot=True)  
print(type(mnist))  
print(mnist.train.images.shape)

out-

<class 'tensorflow.contrib.learn.python.learn.datasets.base.Datasets'>

(55000, 784)

It means training data is having 55000 images with 784 pixel values. We can also check number of examples in training set, test set and validation set as below-

mnist.train.num\_examples

mnist.test.num\_examples

mnist.validation.num\_examples

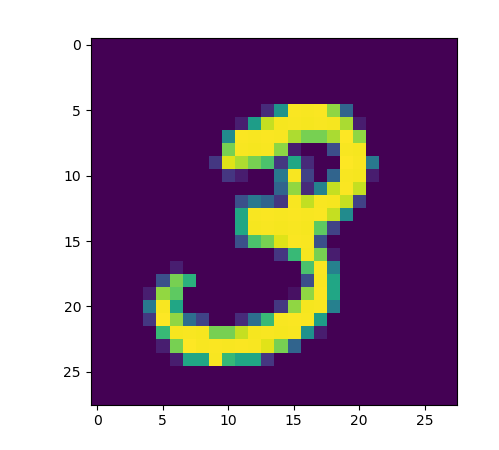
## Visualising data

If we take our first row from the data set, it will have 784 pixel values-

mnist.train.images[1].shape - (784,)

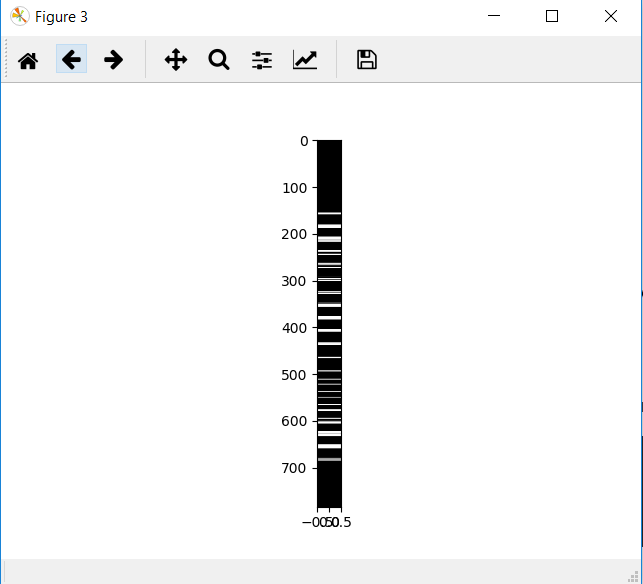
We need to reshape into (28,28) matrix which was the original matrix to visualise it.

# Visualising data  
plt.figure(num=1)  
plt.imshow(mnist.train.images[1].reshape(28,28))  
plt.figure(num=2)  
plt.imshow(mnist.train.images[1].reshape(28,28),cmap='gist\_gray')



What we gonna actually pass to our neural network is which is intensity of pixels

plt.figure(num=3)  
plt.imshow(mnist.train.images[1].reshape(784,1),cmap='gist\_gray',aspect=0.02)



## Creating model

# Creating model  
# Specifying placeholder to process images, we will be sending images in batches  
x = tf.compat.v1.placeholder(dtype=tf.float64,shape=[None,784])  
W = tf.compat.v1.Variable(tf.zeros([784,10]))  
b = tf.compat.v1.Variable(tf.zeros([10]))  
  
y = tf.matmul(x,W) + b  
# Loss and Optimiser  
y\_true = tf.compat.v1.placeholder(tf.float64,[None,10])  
  
# Cross Entropy  
cross\_entropy = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits\_v2(labels=y\_true,logits=y))  
optimiser = tf.train.GradientDescentOptimizer(learning\_rate=0.3)  
train = optimiser.minimize(cross\_entropy)

## Creating Session

# Create Session  
init = tf.global\_variables\_initializer()  
  
with tf.compat.v1.Session as sess:  
 sess.run(init)  
  
 # Train the model for 1000 steps on the training set  
 # Using built in batch feeder from mnist for convenience  
 for step in range(1000):  
 batch\_x,batch\_y = mnist.train.next\_batch(100)  
 sess.run(train,feed\_dict={x:batch\_x,y\_true:batch\_y})  
  
# Test the trained model  
matches = tf.equal(tf.argmax(y,1),tf.argmax(y\_true,1))  
acc = tf.reduce\_mean(tf.cast(matches,tf.float64))  
  
print("Accuracy of model : ")  
print(sess.run(acc,feed\_dict={x:mnist.test.images,y\_true:mnist.test.labels}))

## Complete code

import matplotlib.pyplot as plt  
import tensorflow as tf  
from tensorflow.examples.tutorials.mnist import input\_data  
mnist = input\_data.read\_data\_sets("E:/Py-DS-ML-Bootcamp-master/Refactored\_Py\_DS\_ML\_Bootcamp-master/22-Deep Learning/MNIST\_data/",one\_hot=True)  
  
# Visualising data  
plt.figure(num=1)  
plt.imshow(mnist.train.images[1].reshape(28,28))  
plt.figure(num=2)  
plt.imshow(mnist.train.images[1].reshape(28,28),cmap='gist\_gray')  
plt.figure(num=3)  
plt.imshow(mnist.train.images[1].reshape(784,1),cmap='gist\_gray',aspect=0.02)  
  
# Creating model  
# Specifying placeholder to process images, we will be sending images in batches  
x = tf.compat.v1.placeholder(dtype=tf.float32,shape=[None,784])  
W = tf.compat.v1.Variable(tf.zeros([784,10]))  
b = tf.compat.v1.Variable(tf.zeros([10]))  
  
y = tf.matmul(x,W) + b  
# Loss and Optimiser  
y\_true = tf.compat.v1.placeholder(tf.float32,[None,10])  
  
# Cross Entropy  
cross\_entropy = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits\_v2(labels=y\_true,logits=y))  
optimiser = tf.train.GradientDescentOptimizer(learning\_rate=0.3)  
train = optimiser.minimize(cross\_entropy)  
  
# Create Session  
init = tf.global\_variables\_initializer()  
  
with tf.compat.v1.Session() as sess:  
 sess.run(init)  
   
 # Train the model for 1000 steps on the training set  
 # Using built in batch feeder from mnist for convenience  
 for step in range(1000):  
 print("Running step : " + str(step))  
 batch\_x,batch\_y = mnist.train.next\_batch(100)  
 sess.run(train,feed\_dict={x:batch\_x,y\_true:batch\_y})  
  
 # Test the trained model  
 matches = tf.equal(tf.argmax(y,1),tf.argmax(y\_true,1))  
 acc = tf.reduce\_mean(tf.cast(matches,tf.float32))  
  
 print("Accuracy of model : ")  
 print(sess.run(acc,feed\_dict={x:mnist.test.images,y\_true:mnist.test.labels}))  
  
plt.show()

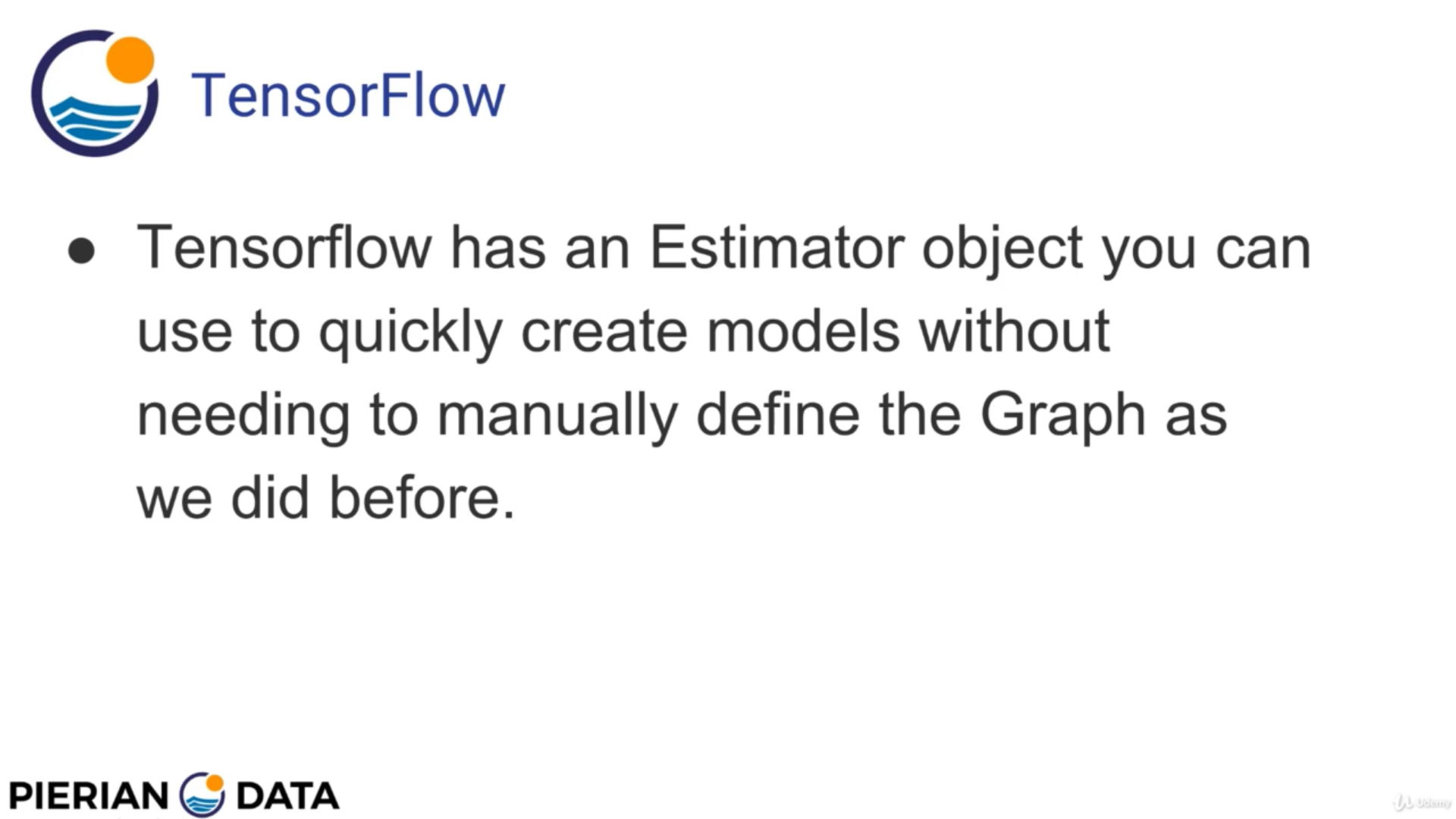
Accuracy comes out to be 0.9168 which may seem pretty good but we can actually do much better, the best models can get above 99% accuracy.

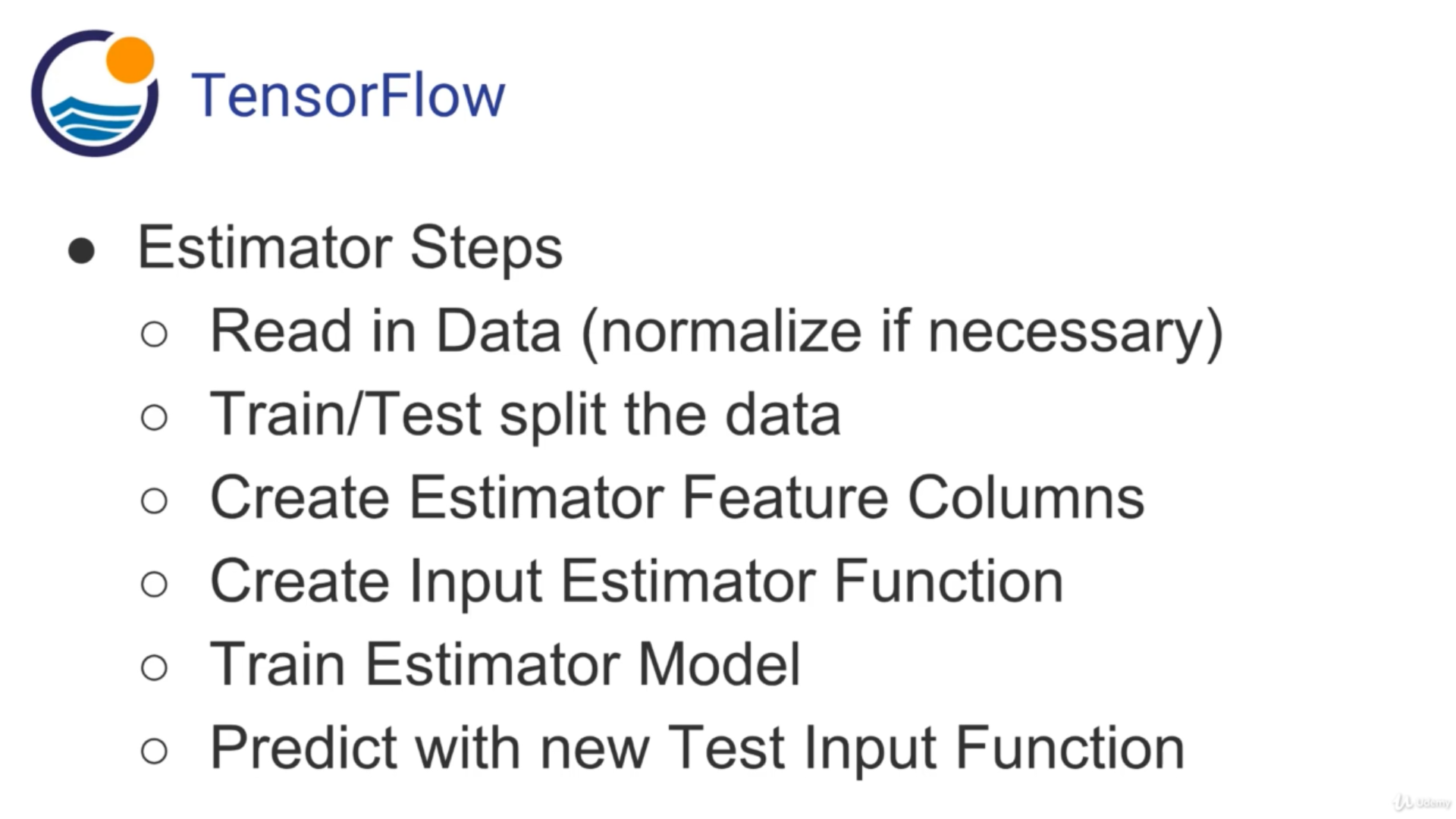
We can increase accuracy by increasing number of steps.

# Tensorflow Estimators

As we saw previously how to build a full Multi-Layer Perceptron model with full Sessions in Tensorflow. Unfortunately this was an extremely involved process. However developers have created Estimators that have an easier to use flow!

It is much easier to use, but you sacrifice some level of customization of your model. Let's go ahead and explore it!





## Get the data

import pandas as pd  
  
df = pd.read\_csv("E:/Py-DS-ML-Bootcamp-master/Refactored\_Py\_DS\_ML\_Bootcamp-master/22-Deep Learning/iris.csv")  
print(df.head())  
  
# We need to remove spaces and special characters from the feature name columns  
# and also, our target class must be int, so doing that  
df.columns = ['sepal\_length','sepal\_width','petal\_length','petal\_width','target']  
X = df.drop('target',axis=1)  
y = df['target'].apply(int)

## Train test Split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33)

## Estimator

Let's show you how to use the simpler Estimator interface!

### Feature Columns

feat\_cols = []  
for col in X.columns:  
 feat\_cols.append(tf.feature\_column.numeric\_column(col))  
print("Specialised feature columns : "),print(feat\_cols),print()

### Input Function

# batch\_size: int, size of batches to return.  
# num\_epochs: int, number of epochs to iterate over data i.e how many times to iterate over the data, its  
# the limit  
# shuffle: bool, whether to read the records in random order.  
input\_func = tf.estimator.inputs.pandas\_input\_fn(x=X\_train,y=y\_train,batch\_size=10,num\_epochs=5,shuffle=True)  
  
# hidden units : takes list which will specify the hidden layers in the neural network  
# having that many number of neurons. In this case, there are three hidden layes having  
# 10,20 and 10 units resp  
# n\_classes : number of classes to expect  
classifier = tf.estimator.DNNClassifier(hidden\_units=[10,20],n\_classes=3,feature\_columns=feat\_cols)  
  
# Training our classifier by specifying input function and number of steps  
classifier.train(input\_fn=input\_func,steps=50)

## Model Evaluation

# Predicting values and evaluating model  
pred\_func = tf.estimator.inputs.pandas\_input\_fn(x=X\_test,batch\_size=len(X\_test),shuffle=False)  
predictions = list(classifier.predict(input\_fn=pred\_func))  
# these predictions is a list storing values in the form of dictionaries,  
# prediction is defined by class\_ids, there are logits and probabilities as well  
print(predictions[0]),print()  
  
# Extracting exact classed from predictions  
final\_preds = []  
for pred in predictions:  
 final\_preds.append(pred["class\_ids"][0])  
  
# Creating classification report and confusion matrix  
print("Classification report :")  
print(classification\_report(y\_test,final\_preds)),print()  
print("Confusion matrix :")  
print(confusion\_matrix(y\_test,final\_preds))

This gives around 96% accuracy.

## Complete code

import pandas as pd  
from sklearn.model\_selection import train\_test\_split  
import tensorflow as tf  
from sklearn.metrics import classification\_report,confusion\_matrix  
  
df = pd.read\_csv("E:/Py-DS-ML-Bootcamp-master/Refactored\_Py\_DS\_ML\_Bootcamp-master/22-Deep Learning/iris.csv")  
print(df.head()),print()  
  
# We need to remove spaces and special characters from the feature name columns  
# and also, our target class must be int, so doing that  
df.columns = ['sepal\_length','sepal\_width','petal\_length','petal\_width','target']  
X = df.drop('target',axis=1)  
y = df['target'].apply(int)  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33)  
  
feat\_cols = []  
for col in X.columns:  
 feat\_cols.append(tf.feature\_column.numeric\_column(col))  
print("Specialised feature columns : "),print(feat\_cols),print()  
  
# batch\_size: int, size of batches to return.  
# num\_epochs: int, number of epochs to iterate over data i.e how many times to iterate over the data, its  
# the limit  
# shuffle: bool, whether to read the records in random order.  
input\_func = tf.estimator.inputs.pandas\_input\_fn(x=X\_train,y=y\_train,batch\_size=10,num\_epochs=5,shuffle=True)  
  
# hidden units : takes list which will specify the hidden layers in the neural network  
# having that many number of neurons. In this case, there are three hidden layes having  
# 10,20 and 10 units resp  
# n\_classes : number of classes to expect  
classifier = tf.estimator.DNNClassifier(hidden\_units=[10,20],n\_classes=3,feature\_columns=feat\_cols)  
  
# Training our classifier by specifying input function and number of steps  
classifier.train(input\_fn=input\_func,steps=50)  
  
# Predicting values and evaluating model  
pred\_func = tf.estimator.inputs.pandas\_input\_fn(x=X\_test,batch\_size=len(X\_test),shuffle=False)  
predictions = list(classifier.predict(input\_fn=pred\_func))  
# these predictions is a list storing values in the form of dictionaries,  
# prediction is defined by class\_ids, there are logits and probabilities as well  
print(predictions[0]),print()  
  
# Extracting exact classed from predictions  
final\_preds = []  
for pred in predictions:  
 final\_preds.append(pred["class\_ids"][0])  
  
# Creating classification report and confusion matrix  
print("Classification report :")  
print(classification\_report(y\_test,final\_preds)),print()  
print("Confusion matrix :")  
print(confusion\_matrix(y\_test,final\_preds))