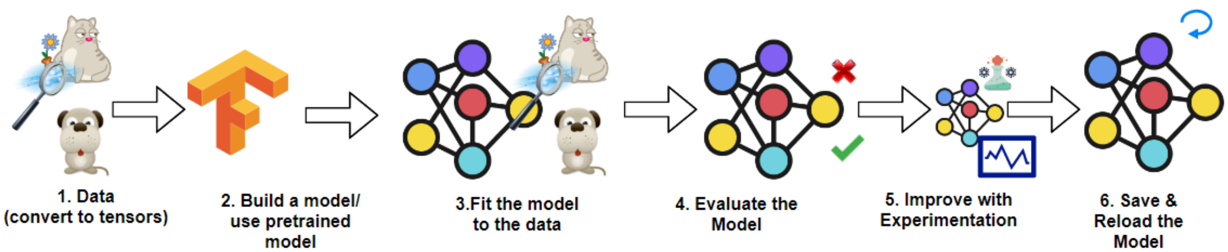
Steps to Create a TensorFlow Model

There are 3 fundamental steps to create a model

- **Create a Model ->** Connect the layers of NN yourself by using Sequential or Functional API or import a previously built model(Transfer Learning)
- **Compile a Model ->** Define how a model's performance should be measured(metrics) and how to improve it by using optimizer(Adam, SGD, etc.)
- **Fit a Model ->** Model try to find pattern in the data



Sequential and Functional API

Sequential Model : A Sequential model is appropriate for a plain stack of layers where each layer has exactly one input tensor and one output tensor. A Sequential model is not appropriate when:

- Your model has multiple inputs or multiple outputs
- Any of your layers has multiple inputs or multiple outputs
- You need to do layer sharing
- You want non-linear topology (e.g. a residual connection, a multi-branch model)

Functional API : The Keras functional API is a way to create models that are more flexible than the tf.keras.Sequential API. The functional API can handle models with non-linear topology, shared layers, and even multiple inputs or outputs.

The main idea is that a deep learning model is usually a directed acyclic graph (DAG) of layers. So the functional API is a way to build graphs of layers.

Regression with TensorFlow

In a [regression (https://medium.com/@priya1803/linear-regression-simplest-algorithm-3f91940d1403)](https://medium.com/@priya1803/linear-regression-simplest-algorithm-3f91940d1403) problem, the aim is to predict the output of a continuous value, like a price or a probability.

In [1]:
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
# Create features using tensors
X = tf.constant(np.random.randint(low =0, high=100, size=50))
X
```

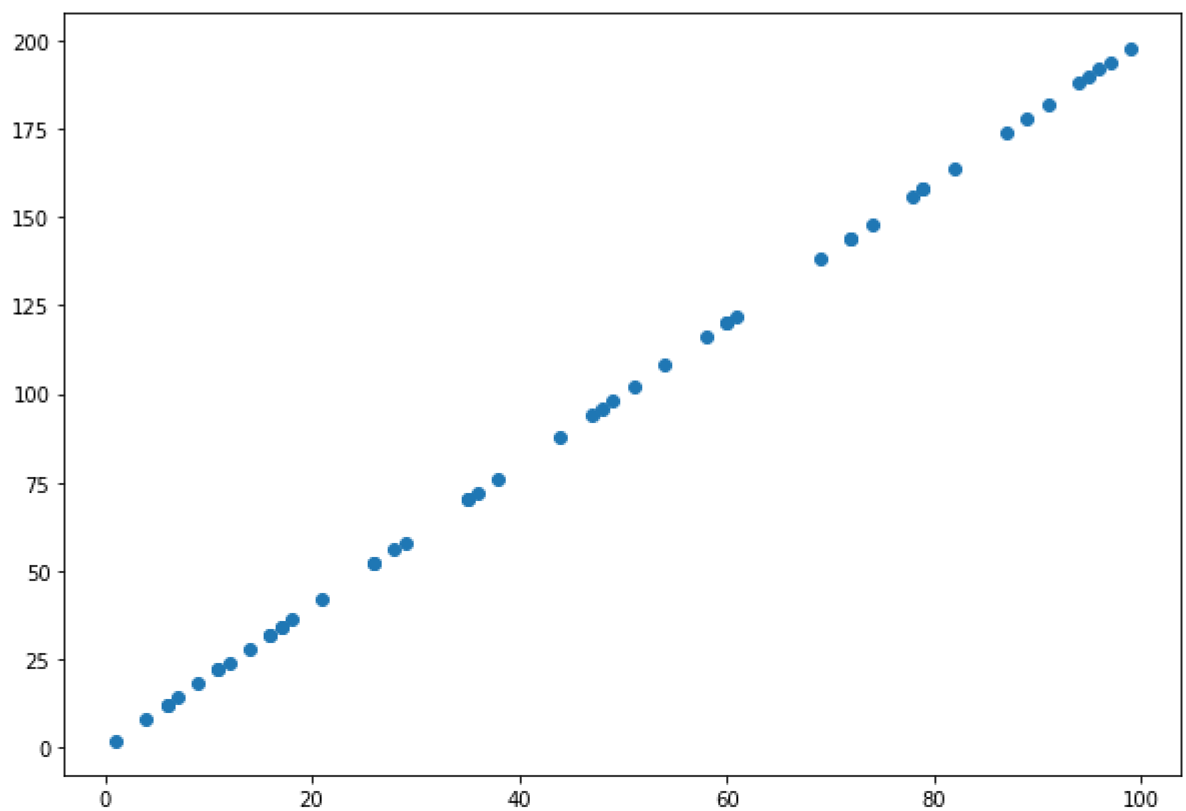
Out[2]:
```
<tf.Tensor: shape=(50,), dtype=int64, numpy=
array([18, 47, 12, 14,  6,  6, 26, 11, 74, 54, 11, 16, 79, 91, 35, 79, 28,
       97, 16, 38, 87, 78,  4, 60, 44, 51, 21, 48, 69, 17, 35, 49, 29, 96,
        9,  1, 72, 61, 95, 94,  7, 89, 72, 60, 26, 17, 58, 36, 82, 99])>
```

In [3]:
```
# Create labels using tensors
y = X*2
y
```

Out[3]:
```
<tf.Tensor: shape=(50,), dtype=int64, numpy=
array([ 36,  94,  24,  28,  12,  12,  52,  22, 148, 108,  22,  32, 158,
       182,  70, 158,  56, 194,  32,  76, 174, 156,   8, 120,  88, 102,
        42,  96, 138,  34,  70,  98,  58, 192,  18,   2, 144, 122, 190,
       188,  14, 178, 144, 120,  52,  34, 116,  72, 164, 198])>
```

In [4]:
```
# Visualize it
plt.figure(figsize=(10,7))
plt.scatter(X,y)
```

Out[4]:
```
<matplotlib.collections.PathCollection at 0x7f24f75ac850>
```



In [5]:
```
X[0].shape, y[0].shape
```

Out[5]:
```
(TensorShape([]), TensorShape([]))
```

Split data into training/test set

One of the other most common and important steps in a machine learning project is creating a training and test set (and when required, a validation set).

Training set - the model learns from this data, which is typically 70-80% of the total data available (like the course materials you study during the semester). **Validation set** - the model gets tuned on this data, which is typically 10-15% of the total data available (like the practice exam you take before the final exam). **Test set** - the model gets evaluated on this data to test what it has learned, it's typically 10-15% of the total data available (like the final exam you take at the end of the semester). For now, we'll just use a training and test set We can create them by splitting our X and y arrays.

In [6]:
```
# No of samples
len(X)
```

Out[6]:
```
50
```

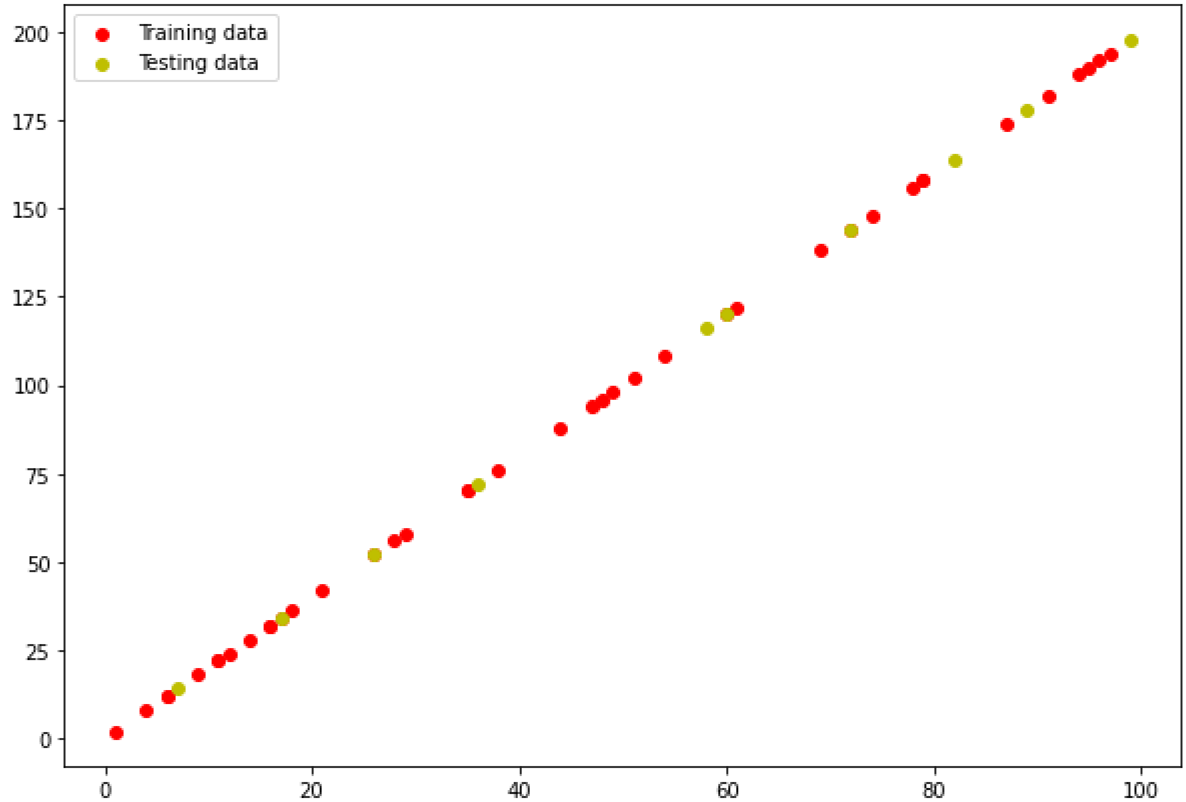
In [7]:
```
# Train Test Split(We can use train test split of scikit learn also)
X_train = X[:40] # first 40 examples (80% of data)
y_train = y[:40]

X_test = X[40:] # last 10 examples (20% of data)
y_test = y[40:]

len(X_train), len(X_test)
```

Out[7]:
```
(40, 10)
```

In [8]:
```
plt.figure(figsize=(10, 7))
# Plot training data in blue
plt.scatter(X_train, y_train, c='r', label='Training data')
# Plot test data in green
plt.scatter(X_test, y_test, c='y', label='Testing data')
# Show the legend
plt.legend();
```



In [9]:
```
# Set random seed
tf.random.set_seed(13)

# 1. Create a model using Sequential API
model = tf.keras.Sequential([
    tf.keras.layers.Dense(1, input_shape=[1]) # define the input_shape to our
 model
])

# 2. Compile the model
model.compile(loss="mae", #Computes the mean absolute error between labels and
predictions
              optimizer="SGD",
              metrics=["mae"])
```

In [10]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 1)                 2

=================================================================
Total params: 2
Trainable params: 2
Non-trainable params: 0
_________________________________________________________________
```

Calling `summary()` on our model shows us the layers it contains, the output shape and the number of parameters.

- `Total params` - total number of parameters in the model.
- `Trainable parameters` - these are the parameters (patterns) the model can update as it trains.
- `Non-trainable parameters` - these parameters aren't updated during training (this is typical when you bring in the already learned patterns from other models during transfer learning).

In [11]:
```
# Fit the model to the training data
model.fit(X_train, y_train, epochs=50) # verbose controls how much gets output
# if we add “verbose = 0” we will not see the training output
```

```
Epoch 1/50
2/2 [==============================] - 3s 10ms/step - loss: 68.7835 - mae: 6
8.7835
Epoch 2/50
2/2 [==============================] - 0s 8ms/step - loss: 25.6554 - mae: 25.
6554
Epoch 3/50
2/2 [==============================] - 0s 8ms/step - loss: 7.3881 - mae: 7.38
81
Epoch 4/50
2/2 [==============================] - 0s 7ms/step - loss: 8.4377 - mae: 8.43
77
Epoch 5/50
2/2 [==============================] - 0s 8ms/step - loss: 7.3554 - mae: 7.35
54
Epoch 6/50
2/2 [==============================] - 0s 8ms/step - loss: 5.6257 - mae: 5.62
57
Epoch 7/50
2/2 [==============================] - 0s 12ms/step - loss: 9.6886 - mae: 9.6
886
Epoch 8/50
2/2 [==============================] - 0s 12ms/step - loss: 13.7157 - mae: 1
3.7157
Epoch 9/50
2/2 [==============================] - 0s 11ms/step - loss: 9.3677 - mae: 9.3
677
Epoch 10/50
2/2 [==============================] - 0s 5ms/step - loss: 14.3006 - mae: 14.
3006
Epoch 11/50
2/2 [==============================] - 0s 6ms/step - loss: 9.5754 - mae: 9.57
54
Epoch 12/50
2/2 [==============================] - 0s 7ms/step - loss: 6.8066 - mae: 6.80
66
Epoch 13/50
2/2 [==============================] - 0s 6ms/step - loss: 6.4226 - mae: 6.42
26
Epoch 14/50
2/2 [==============================] - 0s 6ms/step - loss: 11.0607 - mae: 11.
0607
Epoch 15/50
2/2 [==============================] - 0s 7ms/step - loss: 9.5298 - mae: 9.52
98
Epoch 16/50
2/2 [==============================] - 0s 5ms/step - loss: 6.0347 - mae: 6.03
47
Epoch 17/50
2/2 [==============================] - 0s 6ms/step - loss: 8.2790 - mae: 8.27
90
Epoch 18/50
2/2 [==============================] - 0s 7ms/step - loss: 9.5435 - mae: 9.54
35
Epoch 19/50
2/2 [==============================] - 0s 4ms/step - loss: 14.2450 - mae: 14.
2450
```

```
Epoch 20/50
2/2 [==============================] - 0s 6ms/step - loss: 14.9592 - mae: 14.
9592
Epoch 21/50
2/2 [==============================] - 0s 8ms/step - loss: 20.6498 - mae: 20.
6498
Epoch 22/50
2/2 [==============================] - 0s 4ms/step - loss: 11.1542 - mae: 11.
1542
Epoch 23/50
2/2 [==============================] - 0s 6ms/step - loss: 8.5319 - mae: 8.53
19
Epoch 24/50
2/2 [==============================] - 0s 5ms/step - loss: 10.0237 - mae: 10.
0237
Epoch 25/50
2/2 [==============================] - 0s 4ms/step - loss: 8.6615 - mae: 8.66
15
Epoch 26/50
2/2 [==============================] - 0s 6ms/step - loss: 5.0340 - mae: 5.03
40
Epoch 27/50
2/2 [==============================] - 0s 7ms/step - loss: 7.4462 - mae: 7.44
62
Epoch 28/50
2/2 [==============================] - 0s 11ms/step - loss: 5.8410 - mae: 5.8
410
Epoch 29/50
2/2 [==============================] - 0s 6ms/step - loss: 6.1126 - mae: 6.11
26
Epoch 30/50
2/2 [==============================] - 0s 6ms/step - loss: 5.1069 - mae: 5.10
69
Epoch 31/50
2/2 [==============================] - 0s 5ms/step - loss: 6.1732 - mae: 6.17
32
Epoch 32/50
2/2 [==============================] - 0s 8ms/step - loss: 10.0334 - mae: 10.
0334
Epoch 33/50
2/2 [==============================] - 0s 4ms/step - loss: 8.1525 - mae: 8.15
25
Epoch 34/50
2/2 [==============================] - 0s 18ms/step - loss: 16.9635 - mae: 1
6.9635
Epoch 35/50
2/2 [==============================] - 0s 18ms/step - loss: 18.8534 - mae: 1
8.8534
Epoch 36/50
2/2 [==============================] - 0s 5ms/step - loss: 13.9301 - mae: 13.
9301
Epoch 37/50
2/2 [==============================] - 0s 14ms/step - loss: 17.4251 - mae: 1
7.4251
Epoch 38/50
2/2 [==============================] - 0s 9ms/step - loss: 11.8462 - mae: 11.
8462
```

```
Epoch 39/50
2/2 [==============================] - 0s 6ms/step - loss: 4.3827 - mae: 4.38
27
Epoch 40/50
2/2 [==============================] - 0s 8ms/step - loss: 5.3077 - mae: 5.30
77
Epoch 41/50
2/2 [==============================] - 0s 6ms/step - loss: 7.1247 - mae: 7.12
47
Epoch 42/50
2/2 [==============================] - 0s 4ms/step - loss: 7.2455 - mae: 7.24
55
Epoch 43/50
2/2 [==============================] - 0s 5ms/step - loss: 5.2537 - mae: 5.25
37
Epoch 44/50
2/2 [==============================] - 0s 5ms/step - loss: 6.7664 - mae: 6.76
64
Epoch 45/50
2/2 [==============================] - 0s 4ms/step - loss: 8.0975 - mae: 8.09
75
Epoch 46/50
2/2 [==============================] - 0s 11ms/step - loss: 5.0891 - mae: 5.0
891
Epoch 47/50
2/2 [==============================] - 0s 8ms/step - loss: 5.9796 - mae: 5.97
96
Epoch 48/50
2/2 [==============================] - 0s 18ms/step - loss: 5.8086 - mae: 5.8
086
Epoch 49/50
2/2 [==============================] - 0s 4ms/step - loss: 6.0153 - mae: 6.01
53
Epoch 50/50
2/2 [==============================] - 0s 5ms/step - loss: 2.1267 - mae: 2.12
67
```

Out[11]:

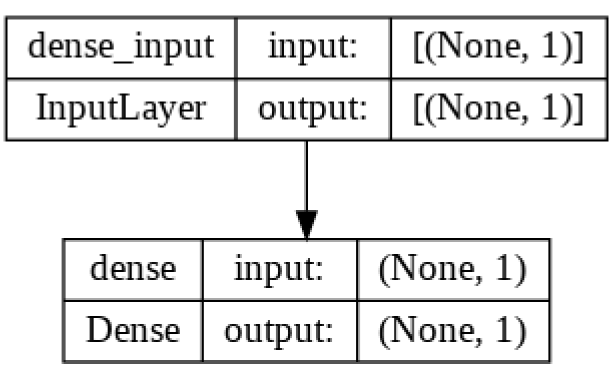
```
<keras.callbacks.History at 0x7f247c7c4e10>
```

In [12]:

```python
from keras.utils.vis_utils import plot_model

plot_model(model, show_shapes=True)
```

Out[12]:



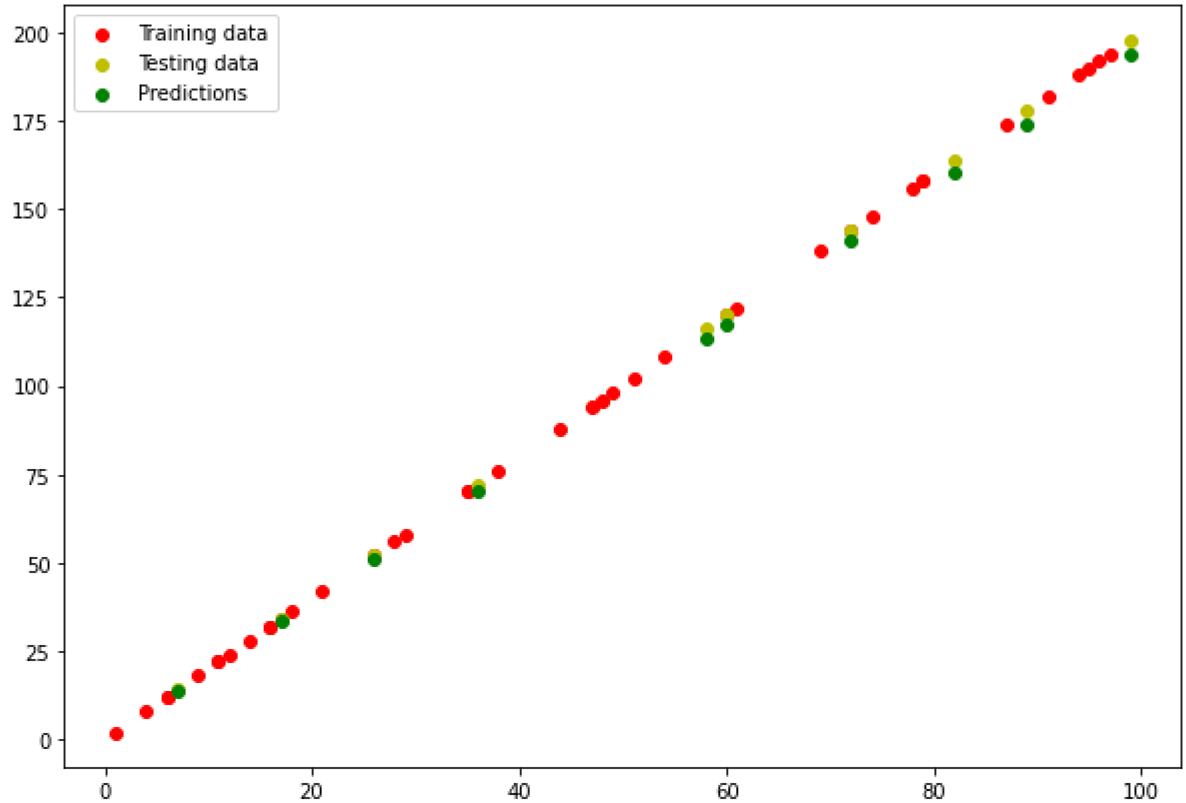
In [13]:
```
# Make predictions
y_preds = model.predict(X_test)
# View the predictions
y_preds
```

```
1/1 [==============================] - 0s 110ms/step
```

Out[13]:
```
array([[ 13.722492],
       [174.12025 ],
       [140.86707 ],
       [117.39422 ],
       [ 50.887825],
       [ 33.283195],
       [113.48208 ],
       [ 70.44853 ],
       [160.42776 ],
       [193.68095 ]], dtype=float32)
```

In [14]:
```
def plot_predictions(train_data=X_train,
                     train_labels=y_train,
                     test_data=X_test,
                     test_labels=y_test,
                     predictions=y_preds):
  """
  Plot training data, test data and compares predictions.
  """
  plt.figure(figsize=(10, 7))
  # Plot training data in blue
  plt.scatter(train_data, train_labels, c="r", label="Training data")
  # Plot test data in green
  plt.scatter(test_data, test_labels, c="y", label="Testing data")
  # Plot the predictions in red (predictions were made on the test data)
  plt.scatter(test_data, predictions, c="g", label="Predictions")
  # Show the legend
  plt.legend();
```

In [15]:
```
plot_predictions(train_data=X_train,
                 train_labels=y_train,
                 test_data=X_test,
                 test_labels=y_test,
                 predictions=y_preds)
```



In [16]:
```
# Evaluate the model on the test set
model.evaluate(X_test, y_test)
```

```
1/1 [==============================] - 0s 107ms/step - loss: 2.3686 - mae: 2.
3686
```

Out[16]:
```
[2.3685619831085205, 2.3685619831085205]
```

In [17]:
```
# Calculate MAE with TensorFlow's inbuilt function
mae = tf.metrics.mean_absolute_error(y_true=y_test, y_pred=y_preds)
mae
```

Out[17]:
```
<tf.Tensor: shape=(10,), dtype=float32, numpy=
array([95.477516, 70.47215 , 55.773415, 51.6      , 69.06731 , 79.77344 ,
       52.103584, 61.020588, 62.971107, 85.344765], dtype=float32)>
```

We are getting 10 different values for MAE but ideally it should be 1 only. Let's figure out the issue

In [18]:
```
y_test.shape
```

Out[18]:
```
TensorShape([10])
```

In [19]:
```
y_preds.shape
```

Out[19]: (10, 1)

As we can see the shape of y_test and y_pred are different. Here comes in picture the use of squeeze method which will remove the dimensions of size 1 from the tensor `tf.squeeze()`

In [20]:
```
y_preds = tf.squeeze(y_preds)
y_preds.shape
```

Out[20]: TensorShape([10])

In [21]:
```
# Calulate MAE using TF
mae = tf.metrics.mean_absolute_error(y_true=y_test, y_pred=y_preds)
mae.numpy()
```

Out[21]: 2.368562

Saving a Model in TensorFlow

We can save TensorFlow/Keras model using `model.save()` . We can save model in two ways:

- SavedModel Format : Models saved in this format can be restored using `tf.keras.models.load_model` and are compatible with TensorFlow Serving. The `SavedModel` format is a directory containing a protobuf binary and a TensorFlow checkpoint.
- HDF5 Format

In [22]:
```
# Save a model using the SavedModel format
model.save('model_SavedModel_format')
```

In [23]:
```
# Check the  outputs a protobuf binary file (.pb) as well as other files
!ls model_SavedModel_format
```

```
assets  keras_metadata.pb  saved_model.pb  variables
```

In [24]:
```
# Save the model using HDF5 format
model.save('model_hdf5_format.h5')
```

Load the Model in TensorFlow

We can use `load_model()` for this

In [25]:
```
# SavedModel Format
load_saved_model = tf.keras.models.load_model('model_SavedModel_format')
load_saved_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense (Dense)               (None, 1)                 2         
                                                                 
=================================================================
Total params: 2
Trainable params: 2
Non-trainable params: 0
_________________________________________________________________
```

In [26]:
```
# HDF5 Model
load_hdf5_model = tf.keras.models.load_model('model_hdf5_format.h5')
load_hdf5_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense (Dense)               (None, 1)                 2         
                                                                 
=================================================================
Total params: 2
Trainable params: 2
Non-trainable params: 0
_________________________________________________________________
```

In [ ]: