Introduction to Regression with Nueral Network in Tensorflow

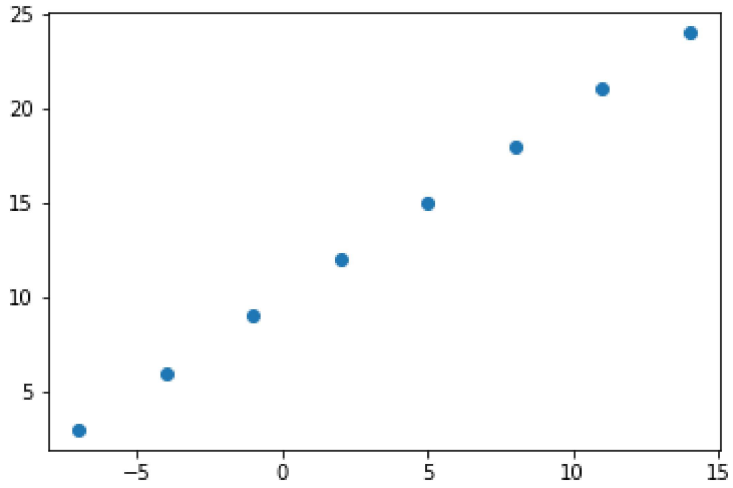
We are going to Predicting a number

In [1]:
```
# Import tensorflow
import tensorflow as tf
import matplotlib.pyplot as plt
print(tf.__version__)
```

```
2.9.1
```

Creating a data to view and fit

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt

# create feature
x = np.array([-7.0, -4., -1., 2., 5., 8., 11., 14])

# Create labels
y = np.array([3.,6.,9.,12.,15.,18.,21.,24.])

# Visualize it
plt.scatter(x,y);
```



In [3]:
```
y == x+10
```

Out[3]:
```
array([ True,  True,  True,  True,  True,  True,  True,  True])
```

Input and output shape

In [4]:
```
# # create a demo tensor for our housing price prediction problem
house_info = tf.constant(['bedroom','bathroom','garage'])
house_price = tf.constant([939700])
house_info, house_price
```

Out[4]:
```
(<tf.Tensor: shape=(3,), dtype=string, numpy=array([b'bedroom', b'bathroom',
b'garage'], dtype=object)>,
 <tf.Tensor: shape=(1,), dtype=int32, numpy=array([939700])>)
```

In [5]:
```
# turn our Numpy array into tensors
x = tf.constant(x)
y = tf.constant(y)
x, y
```

Out[5]:
```
(<tf.Tensor: shape=(8,), dtype=float64, numpy=array([-7., -4., -1.,  2.,  5.,
8., 11., 14.])>,
 <tf.Tensor: shape=(8,), dtype=float64, numpy=array([ 3.,  6.,  9., 12., 15., 1
8., 21., 24.])>)
```

Steps in modelling with Tensorflow

1. Creating a model - define the input and output layers as well as the hidden layers of deep learning model .
2. Compiling a model - define the loss function (in other words, the function which tells our model how wrong is it) and the optimizer (tells our model how to improve the patterns its learning) and evaluation metrics (what we can use to interpret the performance of our model) .
3. Fitting a model - letting the model try to find patterns between x & y (feature and labels)

In [6]:
```
# Set the random seed
tf.random.set_seed(42)

# 1. Create a model using Sequential API
model = tf.keras.Sequential([
    tf.keras.layers.Dense(1)
])
  # or we can do this way by making layers

# model = tf.keras.Sequential()
# model.add(tf.keras.layers.Dense(1)) by using .add we can make layers both work

# 2. Compile the model
model.compile(loss=tf.keras.losses.mae, #mae is short for mean absolute error
              optimizer=tf.keras.optimizers.SGD(), # sgd is short for stochasitc g
              # its the optimizer tell the nueral network how it should improved.
              metrics=["mae"])

# 3. Fit the model
model.fit(tf.expand_dims(x,axis=-1),y, epochs=5) # epochs means how many time you
```

```
Epoch 1/5
1/1 [==============================] - 1s 1s/step - loss: 11.5048 - mae: 11.504
8
Epoch 2/5
1/1 [==============================] - 0s 13ms/step - loss: 11.3723 - mae: 11.3
723
Epoch 3/5
1/1 [==============================] - 0s 3ms/step - loss: 11.2398 - mae: 11.23
98
Epoch 4/5
1/1 [==============================] - 0s 2ms/step - loss: 11.1073 - mae: 11.10
73
Epoch 5/5
1/1 [==============================] - 0s 3ms/step - loss: 10.9748 - mae: 10.97
48
```

Out[6]:
```
<keras.callbacks.History at 0x1c1405bedc0>
```

In [7]:
```
model.predict([5.])
```

```
1/1 [==============================] - 0s 58ms/step
```

Out[7]:
```
array([[3.7753]], dtype=float32)
```

In [8]:
```
# Lets recreate the model

# 1. Create the model
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(1))

# 2. Compile the model
model.compile(loss=tf.keras.losses.mae,
              optimizer=tf.keras.optimizers.SGD(),
              metrics=["mae"])

# 3. Fit the model
model.fit(tf.expand_dims(x,axis=-1),y,epochs=100)
```

```
Epoch 1/100
1/1 [==============================] - 0s 161ms/step - loss: 11.2219 - mae:
11.2219
Epoch 2/100
1/1 [==============================] - 0s 12ms/step - loss: 11.0894 - mae: 1
1.0894
Epoch 3/100
1/1 [==============================] - 0s 2ms/step - loss: 10.9569 - mae: 1
0.9569
Epoch 4/100
1/1 [==============================] - 0s 2ms/step - loss: 10.8244 - mae: 1
0.8244
Epoch 5/100
1/1 [==============================] - 0s 3ms/step - loss: 10.6919 - mae: 1
0.6919
Epoch 6/100
1/1 [==============================] - 0s 3ms/step - loss: 10.5594 - mae: 1
0.5594
Epoch 7/100
```

In [9]:
```
model.predict([17.])
```

```
1/1 [==============================] - 0s 31ms/step
```

Out[9]:
```
array([[29.739855]], dtype=float32)
```

In [10]:
```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(100,activation="relu"),
#     tf.keras.layers.Dense(100,activation="relu"),
#     tf.keras.layers.Dense(100,activation="relu"),

    tf.keras.layers.Dense(1)
])

model.compile(loss=tf.keras.losses.mae,
              optimizer=tf.keras.optimizers.SGD(),
              metrics=["mae"])

model.fit(tf.expand_dims(x,axis=-1),y,epochs=100)
```

```
Epoch 1/100
1/1 [==============================] - 0s 229ms/step - loss: 12.9513 - mae:
12.9513
Epoch 2/100
1/1 [==============================] - 0s 2ms/step - loss: 12.3835 - mae: 1
2.3835
Epoch 3/100
1/1 [==============================] - 0s 3ms/step - loss: 11.8019 - mae: 1
1.8019
Epoch 4/100
1/1 [==============================] - 0s 4ms/step - loss: 11.1297 - mae: 1
1.1297
Epoch 5/100
1/1 [==============================] - 0s 3ms/step - loss: 10.3797 - mae: 1
0.3797
Epoch 6/100
1/1 [==============================] - 0s 3ms/step - loss: 9.4756 - mae: 9.4
756
Epoch 7/100
```

In [11]:
```
model.predict([1.])
```

```
1/1 [==============================] - 0s 31ms/step
```

Out[11]:
```
array([[4.4722357]], dtype=float32)
```

In [12]:
```
model=tf.keras.Sequential([
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(100, activation="relu")
])

model.compile(loss=["mae"],
              optimizer=tf.keras.optimizers.Adam(lr=0.1),
              metrics=["mae"])

model.fit(tf.expand_dims(x,axis=-1),y,epochs=100)
```

```
Epoch 1/100

C:\Users\Amit\anaconda3\lib\site-packages\keras\optimizers\optimizer_v2\adam.py:110: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super(Adam, self).__init__(name, **kwargs)
```

In [13]:
```
a=model.predict([17.])
tf.math.reduce_mean(a)
```

```
1/1 [==============================] - 0s 36ms/step
```

Out[13]:
```
<tf.Tensor: shape=(), dtype=float32, numpy=25.022833>
```

Evaluate the model

In practise, a typical workflow you'll go through when building a nueral network is:

```
Build a model ->  fit it -> evaluate it-> tweak a model -> fit it -> evaluate it-> tweak a mode l-> fit it -> evaluate it....
```

When it comes to evalution... there are 3 words we should visualize

```
"Visualize","Visualize","Visualize"
```

It's a good idea to visualize:

- The data - what data are we working with? What does it look like?
- The model itself - how does a model perform while it learns?
- The training of a model - how does a model perform while it learns?
- The prediction of the model - how do the prediction of a model line up against the ground truth (the original label)

In [14]:
```
# Make a bigger dataset
x = tf.range(-100,100,4)
x
```

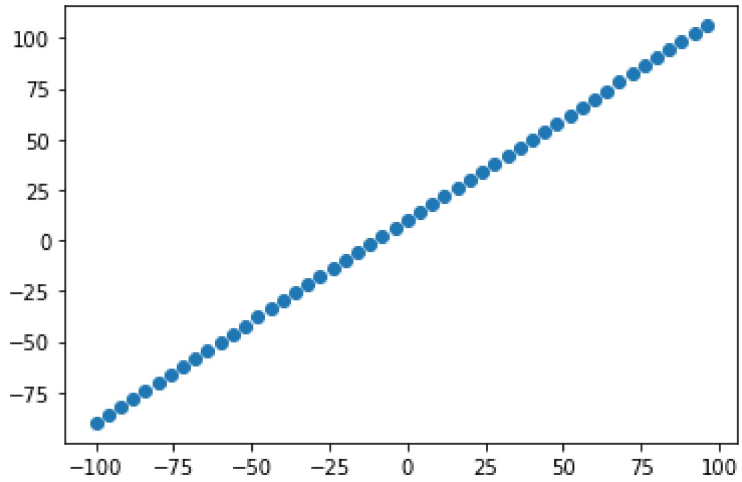
Out[14]:
```
<tf.Tensor: shape=(50,), dtype=int32, numpy=
array([-100,  -96,  -92,  -88,  -84,  -80,  -76,  -72,  -68,  -64,  -60,
        -56,  -52,  -48,  -44,  -40,  -36,  -32,  -28,  -24,  -20,  -16,
        -12,   -8,   -4,    0,    4,    8,   12,   16,   20,   24,   28,
         32,   36,   40,   44,   48,   52,   56,   60,   64,   68,   72,
         76,   80,   84,   88,   92,   96])>
```

In [15]:
```
# Make labels for the dataset
y=x+10
y
```

Out[15]:
```
<tf.Tensor: shape=(50,), dtype=int32, numpy=
array([-90, -86, -82, -78, -74, -70, -66, -62, -58, -54, -50, -46, -42,
       -38, -34, -30, -26, -22, -18, -14, -10,  -6,  -2,   2,   6,  10,
        14,  18,  22,  26,  30,  34,  38,  42,  46,  50,  54,  58,  62,
        66,  70,  74,  78,  82,  86,  90,  94,  98, 102, 106])>
```

In [16]:
```
# Lets visualize the data
import matplotlib.pyplot as plt
plt.scatter(x,y)
```

Out[16]:
```
<matplotlib.collections.PathCollection at 0x1c15c6519a0>
```



The 3 sets....

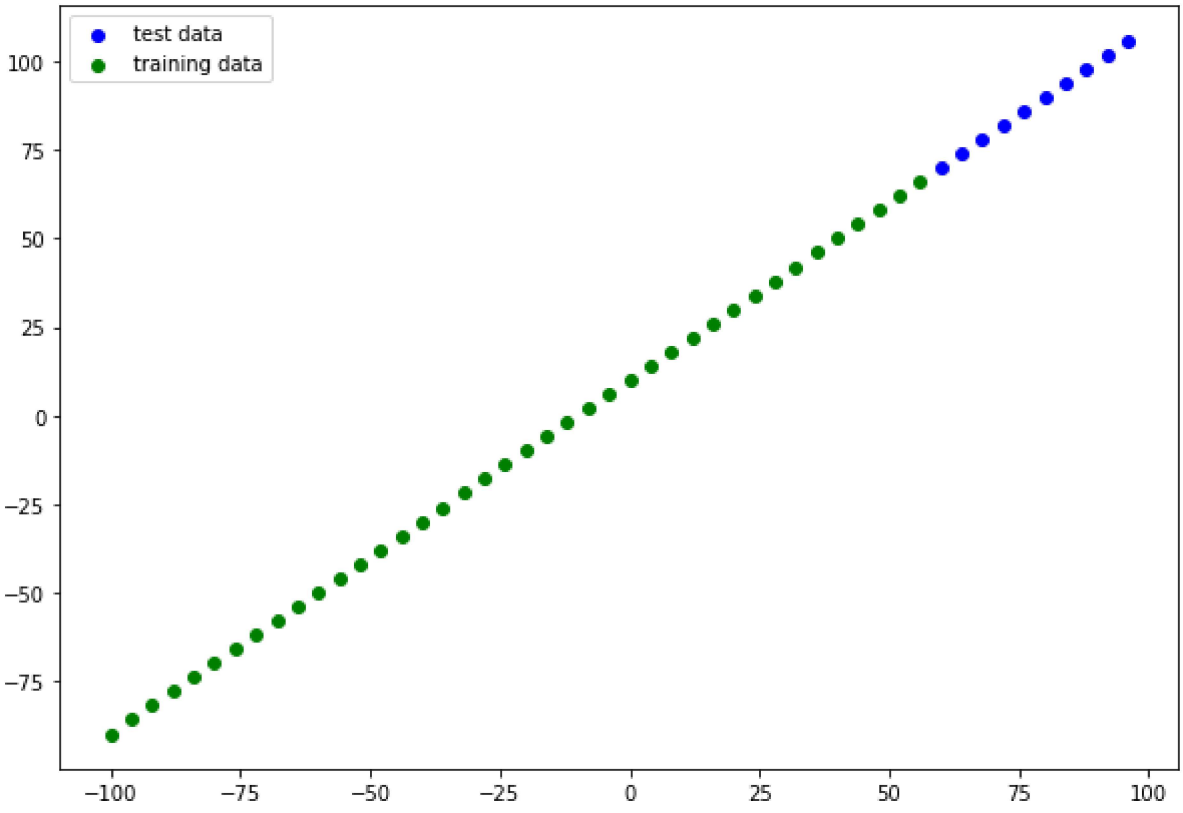
- **Training set** - the model learns from this data, which is typically 70-80% of data total you have available.
- **Validation set** - the model that is got tuned on this data, which is 10-15% of the data available.
- **Test set** - the model get evaluated on this data to test what it has learned , this set is typically 10-15% of the total data avaiable.

In [17]:
```
# split the data into train and test data
x_train = x[:40] # first 40 are training samples (80% of the data)
y_train = y[:40]

x_test = x[40:] # last 10 are testing samples (20% of the data)
y_test = y[40:]
len(x_train),len(x_test)
```

Out[17]: (40, 10)

Visualising the data

In [18]:
```
plt.figure(figsize=(10,7))
# Plot the test data as blue
plt.scatter(x_test,y_test,c="b",label="test data")
#Plot the train data as green
plt.scatter(x_train,y_train,c="g",label="training data")
plt.legend();
```



In [19]:
```
# Let's take a look how to make a nueral network for our data

# Create a model
model =tf.keras.Sequential([
    tf.keras.layers.Dense(1)
])

# Compiile the model
model.compile(loss=tf.keras.losses.mae,
              optimizer=tf.keras.optimizers.SGD(),
              metrics=['mae'])

# fit the model
model.fit(tf.expand_dims(x_train,axis=1),y_train,epochs=100)
```

```
Epoch 1/100
2/2 [==============================] - 0s 9ms/step - loss: 92.7088 - mae: 9
2.7088
Epoch 2/100
2/2 [==============================] - 0s 2ms/step - loss: 57.2533 - mae: 5
7.2533
Epoch 3/100
2/2 [==============================] - 0s 2ms/step - loss: 20.5631 - mae: 2
0.5631
Epoch 4/100
2/2 [==============================] - 0s 3ms/step - loss: 9.4142 - mae: 9.4
142
Epoch 5/100
2/2 [==============================] - 0s 2ms/step - loss: 10.2951 - mae: 1
0.2951
Epoch 6/100
2/2 [==============================] - 0s 3ms/step - loss: 9.5465 - mae: 9.5
465
Epoch 7/100
```

Visualize the model

In [20]:
```
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_9 (Dense)             (None, 1)                 2

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

In [21]:
```
x[0],y[0]
```

Out[21]:
```
(<tf.Tensor: shape=(), dtype=int32, numpy=-100>,
 <tf.Tensor: shape=(), dtype=int32, numpy=-90>)
```

In [22]:
```
# Let's create a model which build automaically by defining the input_shape argum
tf.random.set_seed(42)

# Create a model (Same as above)
model = tf.keras.Sequential([
    tf.keras.layers.Dense(10,input_shape=[1]), # see above section where we defir
    tf.keras.layers.Dense(1, name="output_layer")
],name="model007")

# Compile the model
model.compile(loss=tf.keras.losses.mae,
              optimizer=tf.keras.optimizers.SGD(),
              metrics=['mae'])
```

In [23]:
```
model.summary()
```

```
Model: "model007"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_10 (Dense)            (None, 10)                20

 output_layer (Dense)        (None, 1)                 11

=================================================================
Total params: 31
Trainable params: 31
Non-trainable params: 0
_________________________________________________________________
```

- **Total params**-total number of parameter in the model.
- **Trainanle parameters**-these are the parameters(patterns) the model can update as it trains.
- **Non-trainable params**- these parameter aren't updated during training (this is typical when you build parameters from other model during **transfer learning.**

For more info - check out the MIT's introduction to deep learning video

Try playing around with the number of hidden units in the dense layer, see how it effect the number of parameter (total and trainable) by calling `model.summary()`

In [24]:
```
# fit the model
model.fit(tf.expand_dims(x_train,axis=1),y_train,epochs=100,verbose=0)
```

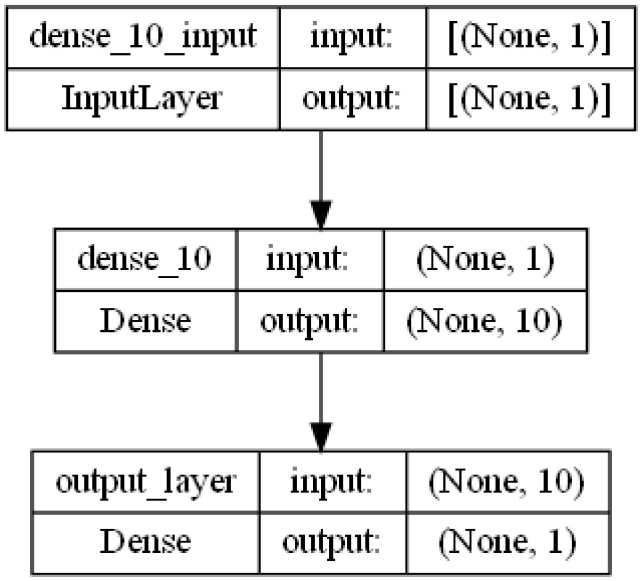
Out[24]:
```
<keras.callbacks.History at 0x1c22c52e460>
```

In [25]:
```
# Get the summary of our model
model.summary()
```

```
Model: "model007"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_10 (Dense)            (None, 10)                20

 output_layer (Dense)        (None, 1)                 11

=================================================================
Total params: 31
Trainable params: 31
Non-trainable params: 0
_________________________________________________________________
```

In [26]:
```
from tensorflow.keras.utils import plot_model
plot_model(model=model, show_shapes=True)
```

Out[26]:



Practise

In [27]:
```
tf.random.set_seed(23)
```

In [28]:
```
model=tf.keras.Sequential([

    tf.keras.layers.Dense(1,activation="relu")
#     tf.keras.layers.Dense(10,input_shape=[10]),
#     tf.keras.layers.Dense(1,input_shape=[10]),
#     tf.keras.layers.Dense(5,input_shape=[2]),
#     tf.keras.layers.Dense(11,input_shape=[2]),
#     tf.keras.layers.Dense(1),
#     tf.keras.layers.Dense(11,input_shape=[2]),

],name="practise1")

model.compile(loss=tf.keras.losses.mae,
              optimizer=tf.keras.optimizers.Adam(lr=.1),
              metrics=["mae"])
model.fit(tf.expand_dims(x_train,axis=-1),y_train,epochs=100,verbose=0)
```

Out[28]:
```
<keras.callbacks.History at 0x1c23525fd30>
```

In [29]:
```
model.summary()
```

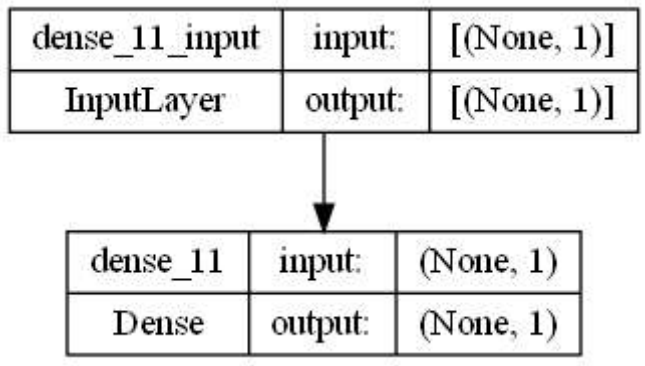
```
Model: "practise1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_11 (Dense)            (None, 1)                 2

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

In [30]:
```
model.layers
```

Out[30]:
```
[<keras.layers.core.dense.Dense at 0x1c23503e0d0>]
```

In [31]:
```
plot_model(model=model,show_shapes=True)
```

Out[31]:



Visualizing our model's prediction

often we see in terms of `y_test` or `y_true` or y_preds

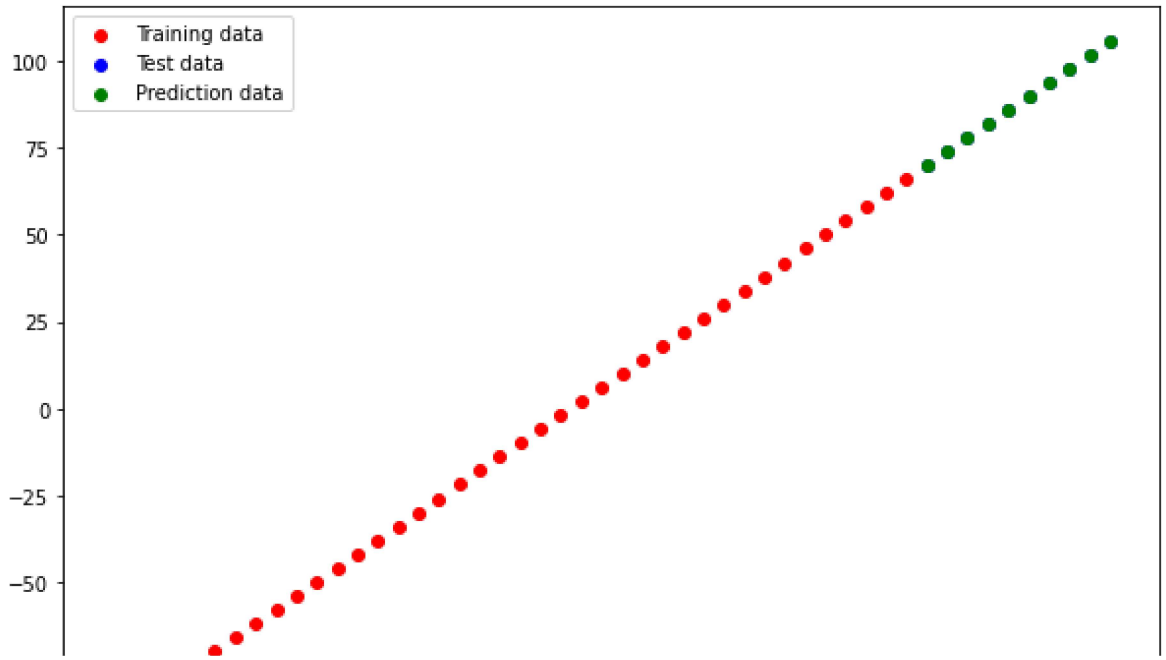
In [32]:
```
y_preds = model.predict(x_test)
y_preds
```

```
WARNING:tensorflow:5 out of the last 5 calls to <function Model.make_predict_fu
nction.<locals>.predict_function at 0x000001C230F97AF0> triggered tf.function r
etracing. Tracing is expensive and the excessive number of tracings could be du
e to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with d
ifferent shapes, (3) passing Python objects instead of tensors. For (1), please
define your @tf.function outside of the loop. For (2), @tf.function has reduce_
retracing=True option that can avoid unnecessary retracing. For (3), please ref
er to https://www.tensorflow.org/guide/function#controlling_retracing (https://
www.tensorflow.org/guide/function#controlling_retracing) and https://www.tensor
flow.org/api_docs/python/tf/function (https://www.tensorflow.org/api_docs/pytho
n/tf/function) for  more details.
1/1 [==============================] - 0s 24ms/step
```

Out[32]:
```
array([[ 69.895424],
       [ 73.89169 ],
       [ 77.887955],
       [ 81.88422 ],
       [ 85.88048 ],
       [ 89.87675 ],
       [ 93.87301 ],
       [ 97.86927 ],
       [101.86553 ],
       [105.8618  ]], dtype=float32)
```

In [33]:
```
# Lets create a plotting function
def plot_prediction(train_data=x_train,
                    train_label=y_train,
                    test_data=x_test,
                    test_label=y_test,
                    prediction=y_preds):
    plt.figure(figsize=(10,7))
    plt.scatter(train_data,train_label,c="r",label="Training data"),
    plt.scatter(test_data,test_label,c="b",label="Test data"),
    plt.scatter(test_data,prediction,c="g",label="Prediction data")
    plt.legend()
```

In [34]:
```
plot_prediction()
```



Evaluating our model's prediction with regression evaluation metrics

Depend on the problem you're working on , there will be different evaluation metrics to evaluate your's model performance.

Since we're working on a regression, two of the main metrics are:

- MAE - mean absolute error, "on average,how wrong is each my model's prediction"
- MSE - mean square error, "square the average errors"

In [35]:
```
#  Evaluate the model
model.evaluate(x_test,y_test)
```

```
1/1 [==============================] - 0s 76ms/step - loss: 0.1214 - mae: 0.121
4
```

Out[35]:
```
[0.12138748168945312, 0.12138748168945312]
```

In [36]:
```
# Calcuate the MAE
mae = tf.metrics.mean_absolute_error(y_true=y_test,
                                     y_pred=y_preds.squeeze()) # use squeeze() to
mae
```

Out[36]:
```
<tf.Tensor: shape=(), dtype=float32, numpy=0.12138748>
```

In [37]:
```
y_test,tf.constant(y_preds)
```

Out[37]:
```
(<tf.Tensor: shape=(10,), dtype=int32, numpy=array([ 70,  74,  78,  82,  86,  9
0,  94,  98, 102, 106])>,
 <tf.Tensor: shape=(10, 1), dtype=float32, numpy=
 array([[ 69.895424],
        [ 73.89169 ],
        [ 77.887955],
        [ 81.88422 ],
        [ 85.88048 ],
        [ 89.87675 ],
        [ 93.87301 ],
        [ 97.86927 ],
        [101.86553 ],
        [105.8618  ]], dtype=float32)>)
```

Extra but important

we want to compare two tensors so they need to be in same shape ,as y_test and y_pred have different shape so lets make it equal with help of `tf.squeeze()` it will get rid of 1 dimension

In [38]:
```
tf.constant(y_preds)
y_preds.squeeze()
y_preds
```

Out[38]:
```
array([[ 69.895424],
       [ 73.89169 ],
       [ 77.887955],
       [ 81.88422 ],
       [ 85.88048 ],
       [ 89.87675 ],
       [ 93.87301 ],
       [ 97.86927 ],
       [101.86553 ],
       [105.8618  ]], dtype=float32)
```

In [39]:
```
# Calculate the mean absolute error
mae = tf.metrics.mean_absolute_error(y_true=y_test,
                                     y_pred=y_preds)
mae
```

Out[39]:
```
<tf.Tensor: shape=(10,), dtype=float32, numpy=
array([18.104576, 14.886645, 12.467227, 10.846313, 10.023905, 10.      ,
       10.774602, 12.347708, 14.719319, 17.88944 ], dtype=float32)>
```

In [40]:
```
# Calculate the MSE
mse = tf.metrics.mean_squared_error(y_true=y_test,
                                    y_pred=y_preds.squeeze())
mse
```

Out[40]:
```
<tf.Tensor: shape=(), dtype=float32, numpy=0.014850098>
```

In [41]:
```
def mae(y_test, y_preds):
  """
  Calcuates mean absolute error between y_test and y_preds.
  """
  return tf.metrics.mean_absolute_error(y_test,
                                        tf.squeeze(y_preds))

def mse(y_test, y_preds):
  """
  Calculates mean squared error between y_test and y_preds.
  """
  return tf.metrics.mean_squared_error(y_test,
                                       tf.squeeze(y_preds))
```

In [42]:
```
mae(y_test,y_preds), mse(y_test,y_preds)
```

Out[42]:
```
(<tf.Tensor: shape=(), dtype=float32, numpy=0.12138748>,
 <tf.Tensor: shape=(), dtype=float32, numpy=0.014850098>)
```

Running experiments to improve our model

- **Get more data** - get more examples for your model to train on (more opportunities to learn patterns or relationship between features and labels).
- **Make your model larger(using a more complex model)** - this might come in the form of more layers or more hidden units in each layer.
- **Train for longer** - give your model more of a chance to find psttern in the data.

Let's do 3 modelling experiments:

1. `model_1` = same as the original model , 1 layer, trained for 100 epochs.
2. `model_2` = 2 layers, trained for 100 epochs
3. `model_3` = 2 layers, trained for 500 epochs

Build `model_1`

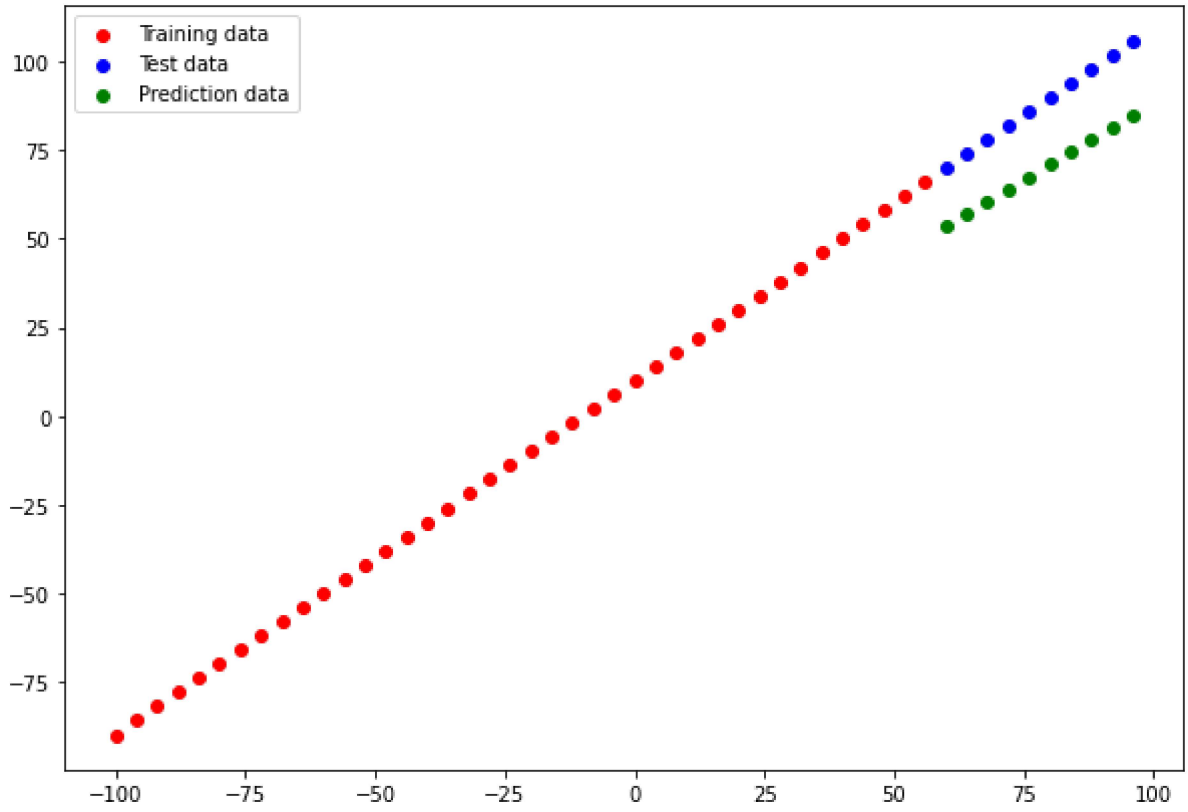
In [43]:

```
# Set random seed
tf.random.set_seed(42)

# Replicate original model
model_1 = tf.keras.Sequential([
  tf.keras.layers.Dense(1)
])

# Compile the model
model_1.compile(loss=tf.keras.losses.mae,
                optimizer=tf.keras.optimizers.SGD(),
                metrics=['mae'])

# Fit the model
model_1.fit(tf.expand_dims(x_train, axis=-1), y_train, epochs=100)
```

```
Epoch 1/100
2/2 [==============================] - 0s 16ms/step - loss: 15.9024 - mae: 1
5.9024
Epoch 2/100
2/2 [==============================] - 0s 2ms/step - loss: 11.2837 - mae: 1
1.2837
Epoch 3/100
2/2 [==============================] - 0s 3ms/step - loss: 11.1074 - mae: 1
1.1074
Epoch 4/100
2/2 [==============================] - 0s 3ms/step - loss: 9.2990 - mae: 9.2
990
Epoch 5/100
2/2 [==============================] - 0s 2ms/step - loss: 10.1677 - mae: 1
0.1677
Epoch 6/100
2/2 [==============================] - 0s 6ms/step - loss: 9.4303 - mae: 9.4
303
Epoch 7/100
```

In [44]:
```
# Make and plot predictions for model_1
y_preds_1 = model_1.predict(x_test)
plot_prediction(prediction=y_preds_1)
```

```
WARNING:tensorflow:6 out of the last 6 calls to <function Model.make_predict_fu
nction.<locals>.predict_function at 0x000001C240CA45E0> triggered tf.function r
etracing. Tracing is expensive and the excessive number of tracings could be du
e to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with d
ifferent shapes, (3) passing Python objects instead of tensors. For (1), please
define your @tf.function outside of the loop. For (2), @tf.function has reduce_
retracing=True option that can avoid unnecessary retracing. For (3), please ref
er to https://www.tensorflow.org/guide/function#controlling_retracing (https://
www.tensorflow.org/guide/function#controlling_retracing) and https://www.tensor
flow.org/api_docs/python/tf/function (https://www.tensorflow.org/api_docs/pytho
n/tf/function) for  more details.
1/1 [==============================] - 0s 24ms/step
```



In [45]:
```
# Calculate model_1 metrics
mae_1 = mae(y_test, y_preds_1.squeeze())
mse_1 = mse(y_test, y_preds_1.squeeze())
mae_1, mse_1
```

Out[45]:
```
(<tf.Tensor: shape=(), dtype=float32, numpy=18.745327>,
 <tf.Tensor: shape=(), dtype=float32, numpy=353.5734>)
```

Build `model_2`

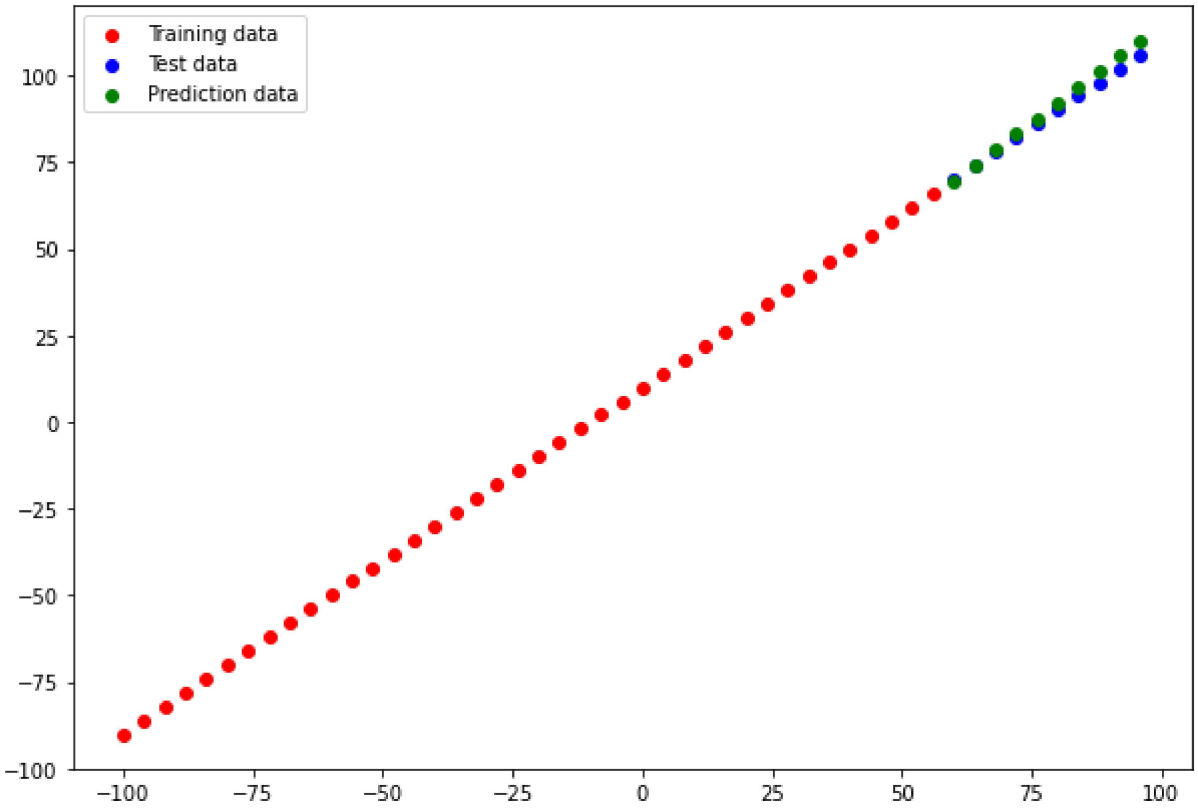
- 2 dense layers,trained for 100 epochs

In [46]:
```
# Set random seed
tf.random.set_seed(42)

# Replicate model_1 and add an extra layer
model_2 = tf.keras.Sequential([
  tf.keras.layers.Dense(1),
  tf.keras.layers.Dense(1) # add a second layer
])

# Compile the model
model_2.compile(loss=tf.keras.losses.mae,
                optimizer=tf.keras.optimizers.SGD(),
                metrics=['mae'])

# Fit the model
model_2.fit(tf.expand_dims(x_train, axis=-1), y_train, epochs=100, verbose=0) # s
```

Out[46]:
```
<keras.callbacks.History at 0x1c242edffd0>
```

In [47]:
```
# Make and plot predictions for model_2
y_preds_2 = model_2.predict(x_test)
plot_prediction(prediction=y_preds_2)
```

```
1/1 [==============================] - 0s 20ms/step
```



In [48]:
```
tf.size(x_test),tf.size(y_preds)
```

Out[48]:
```
(<tf.Tensor: shape=(), dtype=int32, numpy=10>,
 <tf.Tensor: shape=(), dtype=int32, numpy=10>)
```

In [49]:
```
# calculate model_2 evaluation metrics
mae_2=mae(y_test,y_preds) # itss not necessary to squeeze here as we already did
mse_2=mse(y_test,y_preds)
mae_2.numpy(),mse_2.numpy()
```

Out[49]:
```
(0.12138748, 0.014850098)
```

Build `model_3`

- 2 layers, trained for 500 epochs

In [50]:
```
tf.random.set_seed(42)

model_3=tf.keras.Sequential([
    tf.keras.layers.Dense(10),
    tf.keras.layers.Dense(1)
])
model_3.compile(loss=tf.keras.losses.mae,
                optimizer=tf.keras.optimizers.SGD(),
                metrics=["mae"])

model_3.fit(tf.expand_dims(x_train,axis=-1),y_train,epochs=500,verbose=0)
```

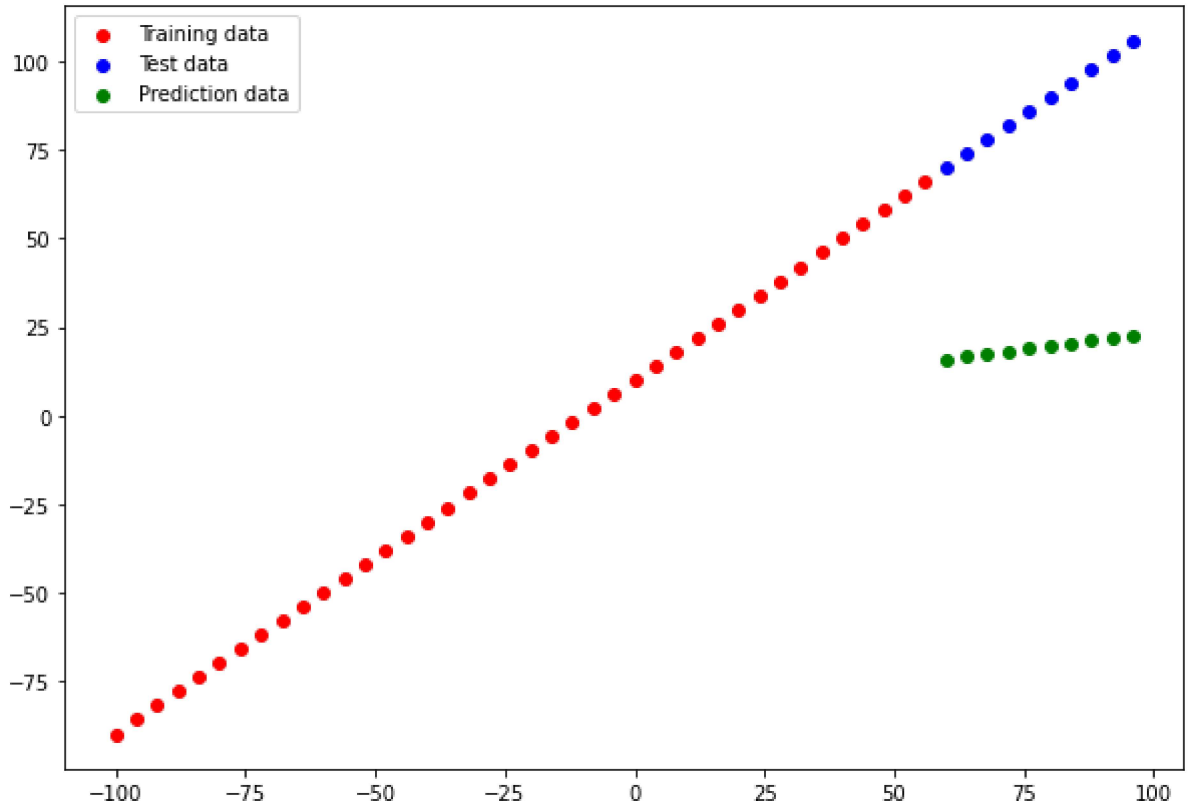
Out[50]:
```
<keras.callbacks.History at 0x1c2461a2d60>
```

In [51]:
```
# calculate model_3 evaluation metrics
mae_3=mae(y_test,y_preds)
mse_3=mse(y_test,y_preds)
mae_3.numpy(),mse_3.numpy()
```

Out[51]:
```
(0.12138748, 0.014850098)
```

In [52]:
```
# Make and plot predictions for model_3
y_preds_3 = model_3.predict(x_test)
plot_prediction(prediction=y_preds_3)
```

```
1/1 [==============================] - 0s 16ms/step
```



Comparing the result of our experiment

In [53]:
```
# Let's compare our model's result using pandas DataFrame
import pandas as pd

model_results = [["model_1",mae_1,mse_1],
                 ["model_2",mae_2,mse_2],
                 ["model_3",mae_3,mse_3]]

all_results = pd.DataFrame(model_results, columns=["model","mae","mse"])
all_results
```

Out[53]:

	model	mae	mse
0	model_1	tf.Tensor(18.745327, shape=(), dtype=float32)	tf.Tensor(353.5734, shape=(), dtype=float32)
1	model_2	tf.Tensor(0.12138748, shape=(), dtype=float32)	tf.Tensor(0.014850098, shape=(), dtype=float32)
2	model_3	tf.Tensor(0.12138748, shape=(), dtype=float32)	tf.Tensor(0.014850098, shape=(), dtype=float32)

it's hard to read and understand so get the numpy value of it

In [95]:
```
# Let's compare our model's result using pandas DataFrame
import pandas as pd

model_results = [["model_1",mae_1.numpy(),mse_1.numpy()],
                 ["model_2",mae_2.numpy(),mse_2.numpy()],
                 ["model_3",mae_3.numpy(),mse_3.numpy()]]

all_results = pd.DataFrame(model_results, columns=["model","mae","mse"])
all_results
```

Out[95]:

	model	mae	mse
0	model_1	18.745327	353.573395
1	model_2	0.121387	0.014850
2	model_3	0.121387	0.014850

Tracking our work

one really good habit in machine learning is to track the result of our experience so for this we can use these two tools.

- **`TensorBoard`** - a component of tensorflow library to help track modelling experiment
- **`Weight & Biases`** -

Saving our models

We can save it by two ways -

- The SavedModel format
- The HDF5 format

In [55]:
```
# Save the model using savedmodel format
model_2.save("save_model_1_savedmodel_format")
```

```
INFO:tensorflow:Assets written to: save_model_1_savedmodel_format\assets
```

In [56]:
```
# Save model using HDF5 format
model_2.save("save_model_1_HDF5_format")
```

```
INFO:tensorflow:Assets written to: save_model_1_HDF5_format\assets
```

Loading the saved model

In [57]:
```
load_model_2=tf.keras.models.load_model("save_model_1_savedmodel_format")
load_model_2.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_13 (Dense)            (None, 1)                 2

 dense_14 (Dense)            (None, 1)                 2

=================================================================
Total params: 4
Trainable params: 4
Non-trainable params: 0
_________________________________________________________________
```

In [58]:
```
model_2.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_13 (Dense)            (None, 1)                 2

 dense_14 (Dense)            (None, 1)                 2

=================================================================
Total params: 4
Trainable params: 4
Non-trainable params: 0
_________________________________________________________________
```

In [59]:
```
# compare the loaded saved model with actual model
model_2_preds = model_2.predict(x_test)
load_model_2_preds = load_model_2.predict(x_test)
model_2_preds == load_model_2_preds
```

```
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 20ms/step
```

Out[59]:
```
array([[ True],
       [ True],
       [ True],
       [ True],
       [ True],
       [ True],
       [ True],
       [ True],
       [ True],
       [ True]])
```

In [60]:
```
load_model_2.predict(x_test)
```

```
1/1 [==============================] - 0s 12ms/step
```

Out[60]:
```
array([[ 69.46714 ],
       [ 73.98562 ],
       [ 78.5041  ],
       [ 83.022575],
       [ 87.54106 ],
       [ 92.05955 ],
       [ 96.578026],
       [101.096504],
       [105.61498 ],
       [110.13346 ]], dtype=float32)
```

A larger example

In [61]:
```
# Import required libraries
import tensorflow as tf
import pandas as pd
import matplotlib.pyplot as plt
```

In [62]:
```
# Read in the insurance dataset
insurance = pd.read_csv("insurance.csv")
insurance
```

Out[62]:

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520
...	...	...	...	...	...	...	...
1333	50	male	30.970	3	no	northwest	10600.54830
1334	18	female	31.920	0	no	northeast	2205.98080
1335	18	female	36.850	0	no	southeast	1629.83350
1336	21	female	25.800	0	no	southwest	2007.94500
1337	61	female	29.070	0	yes	northwest	29141.36030

1338 rows × 7 columns

one hot encode our dataframe so it's all number

In [63]:
```
insurance_one_hot=pd.get_dummies(insurance)
insurance_one_hot
```

Out[63]:

	age	bmi	children	charges	sex_female	sex_male	smoker_no	smoker_yes	region_r
0	19	27.900	0	16884.92400	1	0	0	1	
1	18	33.770	1	1725.55230	0	1	1	0	
2	28	33.000	3	4449.46200	0	1	1	0	
3	33	22.705	0	21984.47061	0	1	1	0	
4	32	28.880	0	3866.85520	0	1	1	0	
...	...	...	...	...	...	...	...	...	
1333	50	30.970	3	10600.54830	0	1	1	0	
1334	18	31.920	0	2205.98080	1	0	1	0	
1335	18	36.850	0	1629.83350	1	0	1	0	
1336	21	25.800	0	2007.94500	1	0	1	0	
1337	61	29.070	0	29141.36030	1	0	0	1	

1338 rows × 12 columns

create x and y values

In [64]:
```
x=insurance_one_hot.drop("charges",axis=1)
x
```

Out[64]:

	age	bmi	children	sex_female	sex_male	smoker_no	smoker_yes	region_northeast	reg
0	19	27.900	0	1	0	0	1	0	
1	18	33.770	1	0	1	1	0	0	
2	28	33.000	3	0	1	1	0	0	
3	33	22.705	0	0	1	1	0	0	
4	32	28.880	0	0	1	1	0	0	
...	...	...	...	...	...	...	...	...	
1333	50	30.970	3	0	1	1	0	0	
1334	18	31.920	0	1	0	1	0	1	
1335	18	36.850	0	1	0	1	0	0	
1336	21	25.800	0	1	0	1	0	0	
1337	61	29.070	0	1	0	0	1	0	

1338 rows × 11 columns

In [65]:
```
y=insurance_one_hot["charges"]
y
```

Out[65]:
```
0       16884.92400
1        1725.55230
2        4449.46200
3       21984.47061
4        3866.85520
           ...
1333    10600.54830
1334     2205.98080
1335     1629.83350
1336     2007.94500
1337    29141.36030
Name: charges, Length: 1338, dtype: float64
```

Creating training and test sets

In [66]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
```

x_train_tf=tf.constant(x_train) y_train_tf=tf.constant(y_train) x_test_tf=tf.constant(x_test) y_test_tf=tf.constant(y_test)

you dont need to do this as it will automatically convert it into tensor as pandas build upon numpy so whole csv is big numpy and we know tensorflow know how to deal with numpy so it directly convert it into tensors

Creating model

In [67]:
```
# Set random seed
tf.random.set_seed(42)

# Create a new model (same as model_2)
insurance_model = tf.keras.Sequential([
  tf.keras.layers.Dense(10),
  tf.keras.layers.Dense(1)
])

# Compile the model
insurance_model.compile(loss=tf.keras.losses.mae,
                        optimizer=tf.keras.optimizers.SGD(),
                        metrics=['mae'])

# Fit the model
insurance_model.fit(x_train, y_train, epochs=100)
```

```
Epoch 24/100
34/34 [==============================] - 0s 2ms/step - loss: 7539.2666 - ma
e: 7539.2666
Epoch 25/100
34/34 [==============================] - 0s 1ms/step - loss: 7619.9668 - ma
e: 7619.9668
Epoch 26/100
34/34 [==============================] - 0s 1ms/step - loss: 7644.1719 - ma
e: 7644.1719
Epoch 27/100
34/34 [==============================] - 0s 1ms/step - loss: 7709.0376 - ma
e: 7709.0376
Epoch 28/100
34/34 [==============================] - 0s 1ms/step - loss: 7366.8662 - ma
e: 7366.8662
Epoch 29/100
34/34 [==============================] - 0s 1ms/step - loss: 7444.3154 - ma
e: 7444.3154
Epoch 30/100
34/34 [==============================] - 0s 728us/step - loss: 7616.4092 - m
```

In [68]:
```
# Check the results of the insurance model
insurance_model.evaluate(x_test, y_test)
```

```
9/9 [==============================] - 0s 2ms/step - loss: 7023.3296 - mae: 702
3.3296
```

Out[68]:
```
[7023.32958984375, 7023.32958984375]
```

In [69]:
```
y_preds=insurance_model.predict(x_test)
```

```
9/9 [==============================] - 0s 2ms/step
```

In [70]:
```
# # Lets create a plotting function
# def plot_prediction(train_data=tf.squeeze(x_train),
#                     train_label=y_train,
#                     test_data=tf.squeeze(x_test),
#                     test_label=y_test,
#                     prediction=tf.squeeze(tf.squeeze(y_preds))):
#     plt.figure(figsize=(10,7))
#     plt.scatter(train_data,train_label,c="r",label="Training data"),
#     plt.scatter(test_data,test_label,c="b",label="Test data"),
#     plt.scatter(test_data,prediction,c="g",label="Prediction data")
#     plt.legend()
```

In [71]:
```
# plot_prediction()
```

In [72]:
```
len(x_train),len(y_train),len(x_test),len(y_preds)
```

Out[72]:
```
(1070, 1070, 268, 268)
```

In [73]:
```
# plt.scatter(tf.squeeze(x_train),y_train,c="r",label="Training data")
```

Let's imporve the model by

- Add an extra layer with more hidden units
- Train for longer
- (insert your own experiment`)

In [74]:
```
# set random seed
tf.random.set_seed(43)

# 1. Create the model
insurance_model_2=tf.keras.Sequential([
    tf.keras.layers.Dense(100),
    tf.keras.layers.Dense(10),
    tf.keras.layers.Dense(1)
])

insurance_model_2.compile(loss=tf.keras.losses.mae,
                          optimizer=tf.keras.optimizers.Adam(lr=0.01),
                          metrics=["mae"])

insurance_model_2.fit(x_train,y_train,epochs=100,verbose=1)
```

```
Epoch 14/100
34/34 [==============================] - 0s 1ms/step - loss: 4083.5107 - ma
e: 4083.5107
Epoch 15/100
34/34 [==============================] - 0s 2ms/step - loss: 3871.4404 - ma
e: 3871.4404
Epoch 16/100
34/34 [==============================] - 0s 1ms/step - loss: 3867.1135 - ma
e: 3867.1135
Epoch 17/100
34/34 [==============================] - 0s 1ms/step - loss: 3842.1672 - ma
e: 3842.1672
Epoch 18/100
34/34 [==============================] - 0s 1ms/step - loss: 3776.9885 - ma
e: 3776.9885
Epoch 19/100
34/34 [==============================] - 0s 1ms/step - loss: 3840.0327 - ma
e: 3840.0327
Epoch 20/100
34/34 [==============================] - 0s 1ms/step - loss: 3759.6519 - ma
```

In [75]:
```
insurance_model_2.evaluate(x_test,y_test)
```

```
9/9 [==============================] - 0s 2ms/step - loss: 3288.8774 - mae: 328
8.8774
```

Out[75]:
```
[3288.87744140625, 3288.87744140625]
```

In [76]:
```
insurance_model.evaluate(x_test,y_test)
```

```
9/9 [==============================] - 0s 3ms/step - loss: 7023.3296 - mae: 702
3.3296
```

Out[76]:
```
[7023.32958984375, 7023.32958984375]
```

Let's try to improve more

In [77]:
```
# set the random seed
tf.random.set_seed(44)

# create the model
insurance_model_3=tf.keras.Sequential([
    tf.keras.layers.Dense(100),
    tf.keras.layers.Dense(10),
    tf.keras.layers.Dense(1)
])
# compile the model
insurance_model_3.compile(loss=tf.keras.losses.mae,
                          optimizer=tf.keras.optimizers.Adam(lr=0.1),
                           metrics=["mae"])

history=insurance_model_3.fit(x_train,y_train,epochs=200,verbose=0)
```

```
C:\Users\Amit\anaconda3\lib\site-packages\keras\optimizers\optimizer_v2\adam.p
y:110: UserWarning: The `lr` argument is deprecated, use `learning_rate` instea
d.
  super(Adam, self).__init__(name, **kwargs)
```

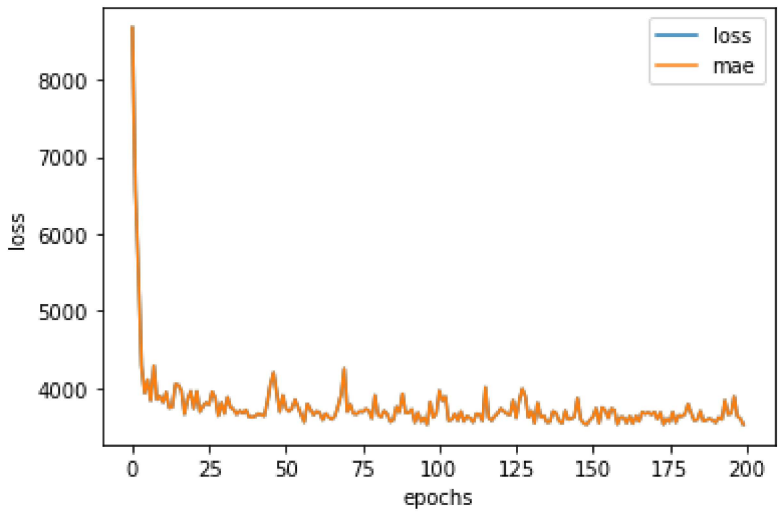
In [78]:
```
insurance_model_3.evaluate(x_test,y_test)
```

```
9/9 [==============================] - 0s 3ms/step - loss: 3223.8066 - mae: 322
3.8066
```

Out[78]:
```
[3223.806640625, 3223.806640625]
```

In [79]:
```
# PLot the insurance_model_3 (history), also known as a loss cure or a training c
pd.DataFrame(history.history).plot()
plt.ylabel("loss"),
plt.xlabel("epochs")
```

Out[79]:
```
Text(0.5, 0, 'epochs')
```



In [80]:
```
# trying another method
tf.random.set_seed(45)

insurance_model_4= tf.keras.Sequential([
    tf.keras.layers.Dense(100,activation="sigmoid"),
    tf.keras.layers.Dense(10),
    tf.keras.layers.Dense(1)
])

insurance_model_4.compile(loss=tf.keras.losses.mae,
                          optimizer=tf.keras.optimizers.Adam(lr=0.1),
                          metrics=["mae"])

insurance_model_4.fit(x_train,y_train,epochs=200,verbose=0)
```

```
C:\Users\Amit\anaconda3\lib\site-packages\keras\optimizers\optimizer_v2\adam.p
y:110: UserWarning: The `lr` argument is deprecated, use `learning_rate` instea
d.
  super(Adam, self).__init__(name, **kwargs)
```

Out[80]:
```
<keras.callbacks.History at 0x1c273eab790>
```

In [81]:
```
insurance_model_4.evaluate(x_test,y_test)
```

```
9/9 [==============================] - 0s 2ms/step - loss: 8685.0117 - mae: 868
5.0117
```

Out[81]:
```
[8685.01171875, 8685.01171875]
```

Preprocessing Data (normalization and standardization)

In [82]:
```
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf

# Read in the insurance dataframe
insurance = pd.read_csv("insurance.csv")
```

In [83]:
```
insurance
```

Out[83]:

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520
...	...	...	...	...	...	...	...
1333	50	male	30.970	3	no	northwest	10600.54830
1334	18	female	31.920	0	no	northeast	2205.98080
1335	18	female	36.850	0	no	southeast	1629.83350
1336	21	female	25.800	0	no	southwest	2007.94500
1337	61	female	29.070	0	yes	northwest	29141.36030

1338 rows × 7 columns

In [84]:
```
# To preapare our data

from sklearn.compose import make_column_transformer
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder
from sklearn.model_selection import train_test_split

# Create a column transformer
ct = make_column_transformer(
     (MinMaxScaler(),["age","bmi","children"]), # turn all values in these columr
     (OneHotEncoder(handle_unknown="ignore"),["sex","smoker","region"]) # handle_
    )
# Create x and y
x = insurance.drop("charges",axis=1)
y = insurance["charges"]

# build our train and test set
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_s

# Fit the column transformer to our training data
ct.fit(x_train)

# Transform training and test data with normalization (MinMaxScaler) and OneHotEr
x_train_normal = ct.transform(x_train)
x_test_normal = ct.transform(x_test)
```

In [85]:
```
# What does our data look like
x_train.loc[0]
```

Out[85]:
```
age               19
sex           female
bmi             27.9
children           0
smoker           yes
region     southwest
Name: 0, dtype: object
```

In [86]:
```
x_train_normal[0] # see it cinverts all things in numerical format
```

Out[86]:
```
array([0.60869565, 0.10734463, 0.4       , 1.        , 0.        ,
       1.        , 0.        , 0.        , 1.        , 0.        ,
       0.        ])
```

In [87]:
```
x_train_normal.shape,x_train.shape
```

Out[87]:
```
((1070, 11), (1070, 6))
```

Beautiful our data is normalise and one hot encode

Let's make our nueral network

In [90]:
```
tf.random.set_seed(42)
insurance_model=tf.keras.Sequential([
    tf.keras.layers.Dense(100),
    tf.keras.layers.Dense(10),
    tf.keras.layers.Dense(1)
])

insurance_model.compile(loss=tf.keras.losses.mae,
                        optimizer=tf.keras.optimizers.Adam(lr=0.1),
                        metrics=["mae"])

insurance_model.fit(x_train_normal,y_train,epochs=100)
```

```
Epoch 1/100
34/34 [==============================] - 0s 3ms/step - loss: 9233.7920 - ma
e: 9233.7920
Epoch 2/100
34/34 [==============================] - 0s 3ms/step - loss: 4625.0542 - ma
e: 4625.0542
Epoch 3/100
34/34 [==============================] - 0s 2ms/step - loss: 3805.1753 - ma
e: 3805.1753
Epoch 4/100
34/34 [==============================] - 0s 1ms/step - loss: 3661.0764 - ma
e: 3661.0764
Epoch 5/100
34/34 [==============================] - 0s 1ms/step - loss: 3663.1360 - ma
e: 3663.1360
Epoch 6/100
34/34 [==============================] - 0s 2ms/step - loss: 3695.3201 - ma
e: 3695.3201
Epoch 7/100
```

In [92]:
```
insurance_model.summary()
```

```
Model: "sequential_14"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_34 (Dense)            (None, 100)               1200

 dense_35 (Dense)            (None, 10)                1010

 dense_36 (Dense)            (None, 1)                 11

=================================================================
Total params: 2,221
Trainable params: 2,221
Non-trainable params: 0
_________________________________________________________________
```

In [93]:
```
# Evaluate our insurance model trained on normalized data
insurance_model.evaluate(x_test_normal,y_test)
```

```
9/9 [==============================] - 0s 4ms/step - loss: 3261.8979 - mae: 326
1.8979
```

Out[93]:
```
[3261.89794921875, 3261.89794921875]
```

In [ ]:
```

```