Scott Berry - Assignment 1

Part 1

1. Create a vector, scalar, matrix and tensor with values of your choosing using tf.constant().

In [1]:
```
import tensorflow as tf

vector = tf.constant([10, 10])
print("Vector:", vector)
scalar = tf.constant(7)
print("Scalar:", scalar)
matrix = tf.constant([[10, 7],
                      [7, 10]])
print("Matrix:", matrix)
tensor = tf.constant([[[1, 2, 3],
                       [4, 5, 6]],
                      [[7, 8, 9],
                       [10, 11, 12]],
                      [[13, 14, 15],
                       [16, 17, 18]]])
print("Tensor:", tensor)
```

```
Vector: tf.Tensor([10 10], shape=(2,), dtype=int32)
Scalar: tf.Tensor(7, shape=(), dtype=int32)
Matrix: tf.Tensor(
[[10  7]
 [ 7 10]], shape=(2, 2), dtype=int32)
Tensor: tf.Tensor(
[[[ 1  2  3]
  [ 4  5  6]]

 [[ 7  8  9]
  [10 11 12]]

 [[13 14 15]
  [16 17 18]]], shape=(3, 2, 3), dtype=int32)
```

1. Find the shape, rank and size of the tensors you created in 1.

In [2]:
```
vector_shape = vector.shape
print("Vector Shape:", vector_shape)
vector_rank = vector.ndim
print("Vector Rank:", vector_rank)
vector_size = tf.size(vector).numpy()
print("Vector Size:", vector_size)
scalar_shape = scalar.shape
print("Scalar Shape:", scalar_shape)
scalar_rank = scalar.ndim
print("Scalar Rank:", scalar_rank)
scalar_size = tf.size(scalar).numpy()
print("Scalar Size:", scalar_size)
matrix_shape = matrix.shape
print("Matrix Shape:", matrix_shape)
matrix_rank = matrix.ndim
print("Matrix Rank:", matrix_rank)
matrix_size = tf.size(matrix).numpy()
print("Matrix Size:", matrix_size)
```

```
tensor_shape = tensor.shape
print("Tensor Shape:", tensor_shape)
tensor_rank = tensor.ndim
print("Tensor Rank:", tensor_rank)
tensor_size = tf.size(tensor).numpy()
print("Tensor Size:", tensor_size)
```

```
Vector Shape: (2,)
Vector Rank: 1
Vector Size: 2
Scalar Shape: ()
Scalar Rank: 0
Scalar Size: 1
Matrix Shape: (2, 2)
Matrix Rank: 2
Matrix Size: 4
Tensor Shape: (3, 2, 3)
Tensor Rank: 3
Tensor Size: 18
```

1. Create two tensors containing random values between 0 and 1 with shape [6, 550].

In [3]:

```
import numpy as np

random_1 = tf.constant(np.random.rand(6, 550))
print("Random 1:", random_1)
random_2 = tf.constant(np.random.rand(6, 550))
print("Random 2:", random_2)
```

```
Random 1: tf.Tensor(
[[0.84081028 0.70837928 0.56244395 ... 0.04219212 0.50682778 0.73584442]
 [0.19882345 0.8500101  0.82918619 ... 0.74391826 0.33125914 0.35750661]
 [0.58420838 0.22981269 0.23256854 ... 0.05622672 0.31165212 0.49940957]
 [0.51813076 0.69389192 0.16953714 ... 0.4268685  0.19755616 0.25584982]
 [0.48211798 0.23143009 0.37700045 ... 0.58296046 0.13526705 0.20784057]
 [0.55047644 0.87509343 0.26051069 ... 0.84717237 0.85850185 0.09817579]], shape
=(6, 550), dtype=float64)
Random 2: tf.Tensor(
[[0.84536115 0.43079276 0.20174531 ... 0.41383705 0.94867111 0.8697899 ]
 [0.23716362 0.06923936 0.20588626 ... 0.61197001 0.51207317 0.06200452]
 [0.23066747 0.56514739 0.29157477 ... 0.54276849 0.44032987 0.10654555]
 [0.02956753 0.34200287 0.49852937 ... 0.44715254 0.48421384 0.16891558]
 [0.03351492 0.22460427 0.38503757 ... 0.59286707 0.76868504 0.56425776]
 [0.08522765 0.34133595 0.77864589 ... 0.54362684 0.03557339 0.27957459]], shape
=(6, 550), dtype=float64)
```

1. Multiply the two tensors you created in 3 using matrix multiplication.

In [4]:

```
random_product = tf.matmul(tf.transpose(random_1), random_2)
print("Random Product:", random_product)
```

```
Random Product: tf.Tensor(
[[0.97109371 1.1795306  1.25346812 ... 1.60349146 1.79777426 1.31935943]
 [0.9562933  1.08189092 1.50134787 ... 1.86127646 1.75349869 1.1857804 ]
 [0.76561771 0.66272437 0.78452376 ... 1.30736882 1.44173929 0.87959556]
 ...
 [0.32942943 0.66755691 1.27498436 ... 1.50027346 1.13067076 0.72670785]
 [0.66244595 0.80838717 1.08036111 ... 1.21685749 1.01784854 0.84428949]
 [0.84493729 0.79168417 0.65169334 ... 1.0853635  1.38819125 0.90334707]], shape
=(550, 550), dtype=float64)
```

1. Create a tensor with random values between 0 and 1 with shape [135, 135, 3].

In [5]:
```
random_3 = tf.constant(np.random.rand(135, 135, 3))
print("Random 3:", random_3)
```

```
Random 3: tf.Tensor(
[[[0.62950839 0.30741231 0.54384821]
  [0.23518832 0.50015858 0.3825448 ]
  [0.85519361 0.79433258 0.24745708]
  ...
  [0.63784458 0.7297547  0.77855286]
  [0.63730349 0.8807443  0.49172915]
  [0.27721736 0.16108746 0.99292545]]

 [[0.17819797 0.44643654 0.85856624]
  [0.62713376 0.90719009 0.24873355]
  [0.02225788 0.95350145 0.93248457]
  ...
  [0.18803622 0.91483133 0.34217824]
  [0.68822836 0.32003609 0.40238079]
  [0.88294595 0.43345437 0.49508597]]

 [[0.24785931 0.23674813 0.6374755 ]
  [0.3452088  0.10222166 0.55988664]
  [0.70606936 0.66354689 0.0103853 ]
  ...
  [0.97075929 0.58007231 0.021589  ]
  [0.45938877 0.55285507 0.6526225 ]
  [0.43548275 0.49697533 0.3559741 ]]

 ...

 [[0.61367635 0.05908614 0.10705448]
  [0.75064821 0.9780182  0.95366646]
  [0.27506839 0.87050606 0.09321931]
  ...
  [0.13828571 0.48080544 0.06637486]
  [0.95157573 0.39888733 0.02851612]
  [0.44889571 0.54396141 0.75105345]]

 [[0.54844306 0.38562161 0.80447216]
  [0.33063533 0.97480315 0.67241385]
  [0.76794444 0.30094521 0.1604224 ]
  ...
  [0.94457354 0.84747943 0.1277904 ]
  [0.9698413  0.60429866 0.18921402]
  [0.54774385 0.09492147 0.24006249]]

 [[0.10243276 0.95391576 0.19718402]
  [0.24227053 0.8952856  0.92615207]
  [0.13322524 0.15490596 0.69739127]
  ...
  [0.37884277 0.55799074 0.48957906]
  [0.5703387  0.7183951  0.09478065]
  [0.45085754 0.1507322  0.56206963]]], shape=(135, 135, 3), dtype=float64)
```

1. Find the min and max values of the tensor you created in 5.

In [6]:
```
random_min = tf.reduce_min(random_3)
print("Random Min:", random_min)
```

```
random_max = tf.reduce_max(random_3)
print("Random Max:", random_max)
```

```
Random Min: tf.Tensor(5.784291858934587e-06, shape=(), dtype=float64)
Random Max: tf.Tensor(0.9999946235657619, shape=(), dtype=float64)
```

1. Created a tensor with random values of shape [1, 135, 135, 28] then squeeze it to change the shape to [135, 135, 28].

In [7]:

```
random_4 = tf.random.Generator.from_seed(42)
random_4 = random_4.normal(shape=(1, 135, 135, 28))
print("Random 4:", random_4)
random_4_squeezed = tf.squeeze(random_4)
print("Random 4 Squeezed:", random_4_squeezed)
```

```
Random 4: tf.Tensor(
[[[[-7.56580293e-01 -6.85470179e-02  7.59502575e-02 ... -5.24474740e-01
    -1.03453290e+00  1.30669010e+00]
   [-1.51845729e+00 -4.58521098e-01  5.71466327e-01 ...  2.67910528e+00
     1.09148061e+00  3.31496149e-01]
   [-6.79589152e-01  4.47236776e-01 -1.78115845e-01 ...  5.08094728e-02
     1.72120607e+00 -6.57512486e-01]
   ...
   [ 5.89750670e-02  7.70883441e-01 -1.90485513e+00 ... -1.07608676e-01
    -3.42550814e-01  9.89552081e-01]
   [-1.80061555e+00  3.57340217e-01  1.93566871e+00 ...  1.13655913e+00
    -4.56918061e-01  1.02984536e+00]
   [ 4.87509787e-01 -1.97188854e-01 -1.52990854e+00 ...  7.91458249e-01
    -4.80736762e-01 -9.96623933e-01]]

  [[-6.87561035e-02 -1.54220068e+00 -4.98402983e-01 ... -1.26698911e+00
     2.26338580e-03 -1.17119980e+00]
   [-3.25447656e-02 -8.13786328e-01 -2.02301621e-01 ... -1.11319587e-01
    -9.89492834e-01 -1.23322263e-01]
   [-1.94080842e+00  5.85251629e-01  2.45584026e-01 ...  9.68298018e-02
     1.59199917e+00  1.07180655e+00]
   ...
   [-1.12680888e+00 -3.30979854e-01 -1.09712279e+00 ...  7.70933405e-02
    -4.06645000e-01 -1.31154370e+00]
   [-9.44205344e-01 -1.19631313e-01 -5.86558461e-01 ...  2.94479847e-01
    -9.58905637e-01  1.08998966e+00]
   [ 1.49113095e+00  8.63313556e-01  1.54518270e+00 ...  8.68564546e-01
    -1.18864441e+00 -2.01495075e+00]]

  [[-9.15264487e-01  1.08560789e+00  2.13917911e-01 ... -1.12349105e+00
    -6.86748326e-01  1.08165407e+00]
   [ 1.36272788e-01  1.61550236e+00 -5.73008545e-02 ... -1.67737687e+00
     2.23104656e-01  8.18086624e-01]
   [ 7.24310994e-01  8.16616774e-01  3.75427127e-01 ...  1.00838280e+00
    -9.02757421e-02  8.99282396e-01]
   ...
   [ 2.02776217e+00 -2.59815872e-01  6.08524263e-01 ...  1.27692983e-01
     4.91895616e-01  4.98485953e-01]
   [ 1.01725948e+00 -6.09773636e-01  5.08597255e-01 ... -1.01565695e+00
    -6.83121085e-01  6.94719255e-01]
   [ 3.95272225e-01  1.76689947e+00 -3.66062284e-01 ...  4.34721351e-01
     7.60397494e-01  2.47445941e+00]]

  ...

  [[-8.76862407e-01  1.61419821e+00 -3.51916581e-01 ...  3.40533108e-01
     1.51632547e+00  8.12785387e-01]
```

```
   [ 2.70662069e-01 -4.49911743e-01 -1.44915417e-01 ...  7.26420105e-01
     1.42138481e+00 -3.79321992e-01]
   [-1.17407329e-02 -1.62321895e-01 -7.51385272e-01 ...  9.52308238e-01
    -1.93774566e-01  6.71820045e-01]
   ...
   [-1.33273959e+00 -9.26882505e-01 -1.14526391e+00 ... -7.44448543e-01
     6.03747606e-01  5.98903954e-01]
   [-6.81876481e-01  1.04795432e+00  2.01240063e+00 ...  6.90456688e-01
     8.51347111e-03 -3.49511027e-01]
   [-1.12040555e+00 -9.71779287e-01  1.45581079e+00 ...  5.67285717e-01
     4.66269672e-01  9.83668506e-01]]

  [[-1.90112874e-01 -1.54444054e-01  7.44098067e-01 ...  2.03086829e+00
     8.23700368e-01 -4.42231894e-01]
   [ 6.62773430e-01 -2.49471590e-01  1.30280674e+00 ... -8.49573493e-01
    -8.05570781e-01 -1.15533948e+00]
   [ 6.81911707e-01  5.72562218e-01 -1.22234869e+00 ...  7.07355917e-01
     9.64198768e-01  1.73979416e-01]
   ...
   [-1.30547595e+00  6.75993800e-01 -3.77649635e-01 ...  2.21032572e+00
     1.13975704e+00  2.02571702e+00]
   [-6.37577653e-01 -5.54566383e-01  1.21582174e+00 ...  1.58388495e+00
    -1.15655589e+00 -4.20873731e-01]
   [-8.46455395e-01 -7.27019738e-03 -1.21285784e+00 ...  7.80251503e-01
    -1.29149354e+00  1.43585324e+00]]

  [[ 9.14727390e-01 -9.19673860e-01 -7.24058986e-01 ...  6.52686596e-01
     9.00972914e-03 -7.72573411e-01]
   [-7.79203475e-01  7.02897012e-01  4.92625296e-01 ... -8.78040045e-02
    -5.21770835e-01 -1.15092111e+00]
   [-9.77871835e-01 -7.75122464e-01 -1.19471931e+00 ... -8.77241969e-01
     1.19297671e+00  2.74731547e-01]
   ...
   [ 3.20593894e-01 -3.03335547e-01 -1.41493201e+00 ...  8.79611492e-01
    -2.90839601e+00  1.09960198e+00]
   [-1.81233689e-01  1.49994695e+00  1.99870300e+00 ... -1.37261912e-01
     5.21750867e-01 -8.03798437e-01]
   [ 6.31250083e-01 -1.58112694e-03 -1.19338119e+00 ... -3.88699584e-02
    -1.25972462e+00 -4.78388458e-01]]]], shape=(1, 135, 135, 28), dtype=float32)
Random 4 Squeezed: tf.Tensor(
[[[-7.56580293e-01 -6.85470179e-02  7.59502575e-02 ... -5.24474740e-01
   -1.03453290e+00  1.30669010e+00]
  [-1.51845729e+00 -4.58521098e-01  5.71466327e-01 ...  2.67910528e+00
    1.09148061e+00  3.31496149e-01]
  [-6.79589152e-01  4.47236776e-01 -1.78115845e-01 ...  5.08094728e-02
    1.72120607e+00 -6.57512486e-01]
  ...
  [ 5.89750670e-02  7.70883441e-01 -1.90485513e+00 ... -1.07608676e-01
   -3.42550814e-01  9.89552081e-01]
  [-1.80061555e+00  3.57340217e-01  1.93566871e+00 ...  1.13655913e+00
   -4.56918061e-01  1.02984536e+00]
  [ 4.87509787e-01 -1.97188854e-01 -1.52990854e+00 ...  7.91458249e-01
   -4.80736762e-01 -9.96623933e-01]]

 [[-6.87561035e-02 -1.54220068e+00 -4.98402983e-01 ... -1.26698911e+00
    2.26338580e-03 -1.17119980e+00]
  [-3.25447656e-02 -8.13786328e-01 -2.02301621e-01 ... -1.11319587e-01
   -9.89492834e-01 -1.23322263e-01]
  [-1.94080842e+00  5.85251629e-01  2.45584026e-01 ...  9.68298018e-02
    1.59199917e+00  1.07180655e+00]
  ...
  [-1.12680888e+00 -3.30979854e-01 -1.09712279e+00 ...  7.70933405e-02
   -4.06645000e-01 -1.31154370e+00]
  [-9.44205344e-01 -1.19631313e-01 -5.86558461e-01 ...  2.94479847e-01
   -9.58905637e-01  1.08998966e+00]
```

```
 [ 1.49113095e+00  8.63313556e-01  1.54518270e+00 ...  8.68564546e-01
  -1.18864441e+00 -2.01495075e+00]]

[[-9.15264487e-01  1.08560789e+00  2.13917911e-01 ... -1.12349105e+00
  -6.86748326e-01  1.08165407e+00]
 [ 1.36272788e-01  1.61550236e+00 -5.73008545e-02 ... -1.67737687e+00
   2.23104656e-01  8.18086624e-01]
 [ 7.24310994e-01  8.16616774e-01  3.75427127e-01 ...  1.00838280e+00
  -9.02757421e-02  8.99282396e-01]
 ...
 [ 2.02776217e+00 -2.59815872e-01  6.08524263e-01 ...  1.27692983e-01
   4.91895616e-01  4.98485953e-01]
 [ 1.01725948e+00 -6.09773636e-01  5.08597255e-01 ... -1.01565695e+00
  -6.83121085e-01  6.94719255e-01]
 [ 3.95272225e-01  1.76689947e+00 -3.66062284e-01 ...  4.34721351e-01
   7.60397494e-01  2.47445941e+00]]

...

[[-8.76862407e-01  1.61419821e+00 -3.51916581e-01 ...  3.40533108e-01
   1.51632547e+00  8.12785387e-01]
 [ 2.70662069e-01 -4.49911743e-01 -1.44915417e-01 ...  7.26420105e-01
   1.42138481e+00 -3.79321992e-01]
 [-1.17407329e-02 -1.62321895e-01 -7.51385272e-01 ...  9.52308238e-01
  -1.93774566e-01  6.71820045e-01]
 ...
 [-1.33273959e+00 -9.26882505e-01 -1.14526391e+00 ... -7.44448543e-01
   6.03747606e-01  5.98903954e-01]
 [-6.81876481e-01  1.04795432e+00  2.01240063e+00 ...  6.90456688e-01
   8.51347111e-03 -3.49511027e-01]
 [-1.12040555e+00 -9.71779287e-01  1.45581079e+00 ...  5.67285717e-01
   4.66269672e-01  9.83668506e-01]]

[[-1.90112874e-01 -1.54444054e-01  7.44098067e-01 ...  2.03086829e+00
   8.23700368e-01 -4.42231894e-01]
 [ 6.62773430e-01 -2.49471590e-01  1.30280674e+00 ... -8.49573493e-01
  -8.05570781e-01 -1.15533948e+00]
 [ 6.81911707e-01  5.72562218e-01 -1.22234869e+00 ...  7.07355917e-01
   9.64198768e-01  1.73979416e-01]
 ...
 [-1.30547595e+00  6.75993800e-01 -3.77649635e-01 ...  2.21032572e+00
   1.13975704e+00  2.02571702e+00]
 [-6.37577653e-01 -5.54566383e-01  1.21582174e+00 ...  1.58388495e+00
  -1.15655589e+00 -4.20873731e-01]
 [-8.46455395e-01 -7.27019738e-03 -1.21285784e+00 ...  7.80251503e-01
  -1.29149354e+00  1.43585324e+00]]

[[ 9.14727390e-01 -9.19673860e-01 -7.24058986e-01 ...  6.52686596e-01
   9.00972914e-03 -7.72573411e-01]
 [-7.79203475e-01  7.02897012e-01  4.92625296e-01 ... -8.78040045e-02
  -5.21770835e-01 -1.15092111e+00]
 [-9.77871835e-01 -7.75122464e-01 -1.19471931e+00 ... -8.77241969e-01
   1.19297671e+00  2.74731547e-01]
 ...
 [ 3.20593894e-01 -3.03335547e-01 -1.41493201e+00 ...  8.79611492e-01
  -2.90839601e+00  1.09960198e+00]
 [-1.81233689e-01  1.49994695e+00  1.99870300e+00 ... -1.37261912e-01
   5.21750867e-01 -8.03798437e-01]
 [ 6.31250083e-01 -1.58112694e-03 -1.19338119e+00 ... -3.88699584e-02
  -1.25972462e+00 -4.78388458e-01]]]], shape=(135, 135, 28), dtype=float32)
```

Part 2

1. Then for what closed-form solutions of w_0 and w_1, will the above equation be minimum? If,

$$h_w(x^i) = w_0 + w_1 x^i$$

and

$$C = \sum_{i=1}^{n} \frac{1}{2n}(h_w(x^i) - y^i)^2$$

Minimum exists when C = 0 which occurs when:

$$h_w(x^i) = y^i$$

and therefore

$$y^i = w_0 + w_1 x^i$$

1. What are the problems you might encounter if you wanted to compute the second derivatives of the loss function with respect to weights? How would you fix them?

The first derivative could be a constant, which would result in the second derivative being equal to 0 and thus the weights being rendered useless. This issue could be resolved by instead taking partial derivatives.