CISB60 - TensorFlow Lab Assignment

Part 1

In this lab we will perform various operations using TensorFlow in Python. We will be creating tensors, performing mathematical operations on tensors, and using some statistical functions.

steps:

- Import TensorFlow and check the version
- Tensor Operations
  - multiplication
- More Tensor Operations
  - create a tensor of ones with shape (3,3)
  - create another tensor with a random normal distribution with shape (4,4), mean 4, and standard deviation 2.
- Import libraries
  - NumPy, Matplotlib, Seaborn, and Warnings
- Create a Random Normal Distribution with shape (1000,) and visualize it with Seaborn
- Create a Uniform Distribution with shape (6,6) and visualize it with Seaborn
- TensorFlow Variable and NumPy Conversion
  - create a TensorFlow variable with a constant tensor
  - generate and print a NumPy array and convert it TensorFlow variable
- TensorFlow Range and Conditional Update
  - using TensorFlow's tf.range, create a range of values
- Tensor Operations and Statistical Functions
  - apply statistical functions to a tensor: sum, mean, max, min.
  - calculate the mean across all dimensions and along axis 0.

Import the tensorflow library under the alias tf

In [3]:
```python
import tensorflow as tf
```

Print version of TensorFlow

In [5]:
```python
print("TensorFlow version:", tf.__version__)
```

```
TensorFlow version: 2.17.0
```

Initialize two variables that are actually constants. Pass an array of four numbers to the constant() function.

for example: x1 = tf.constant([1,2,3,4])

In [8]:
```
x1 = tf.constant([1, 2, 3, 4])
x2 = tf.constant([5, 6, 7, 8])

print("x1:", x1)
print("x2:", x2)
```

```
x1: tf.Tensor([1 2 3 4], shape=(4,), dtype=int32)
x2: tf.Tensor([5 6 7 8], shape=(4,), dtype=int32)
```

use multiply() to multiply your two variables. Store the result in a variable called "result"

In [12]:
```
result = tf.multiply(x1, x2)
```

Lastly, print out the result with the help of the print() function to show the shape and dtype

In [15]:
```
print("Result of multiplication:", result)
print("Shape of result:", result.shape)
print("Data type of result:", result.dtype)
```

```
Result of multiplication: tf.Tensor([ 5 12 21 32], shape=(4,), dtype=int32)
Shape of result: (4,)
Data type of result: <dtype: 'int32'>
```

create a tensor of "ones" with a shape 3 x 3

In [18]:
```
ones_tensor = tf.ones((3, 3))

print("Tensor of Ones:\n", ones_tensor)
```

```
Tensor of Ones:
 tf.Tensor(
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]], shape=(3, 3), dtype=float32)
```

Using the random.normal() function, create a tf. Tensor with values sampled from a normal distribution with the following parameters:

- shape 4 x 4
- mean 4
- stddev 2

In [20]:
```
# random: gaussian
random_tensor = tf.random.normal(shape=(4, 4), mean=4, stddev=2)

print("Random Normal Tensor:\n", random_tensor)
```

```
Random Normal Tensor:
 tf.Tensor(
[[4.92795   1.3835137 4.7767177 2.870193 ]
 [3.6500955 3.7730808 2.3217902 3.931652 ]
 [4.8997374 0.399112  1.5203507 4.2825584]
 [4.83505   7.2337894 2.769227  6.142108 ]], shape=(4, 4), dtype=float32)
```

import numpy, matplotlib, and seaborn

In [26]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Create a TF Tensor with the values sampled from a normal distribution with shape = (1000,0) and name it x

In [28]:
```
import tensorflow as tf

x = tf.random.normal(shape=(1000,), mean=0, stddev=1)

print("Tensor x with normal distribution:\n", x)
```

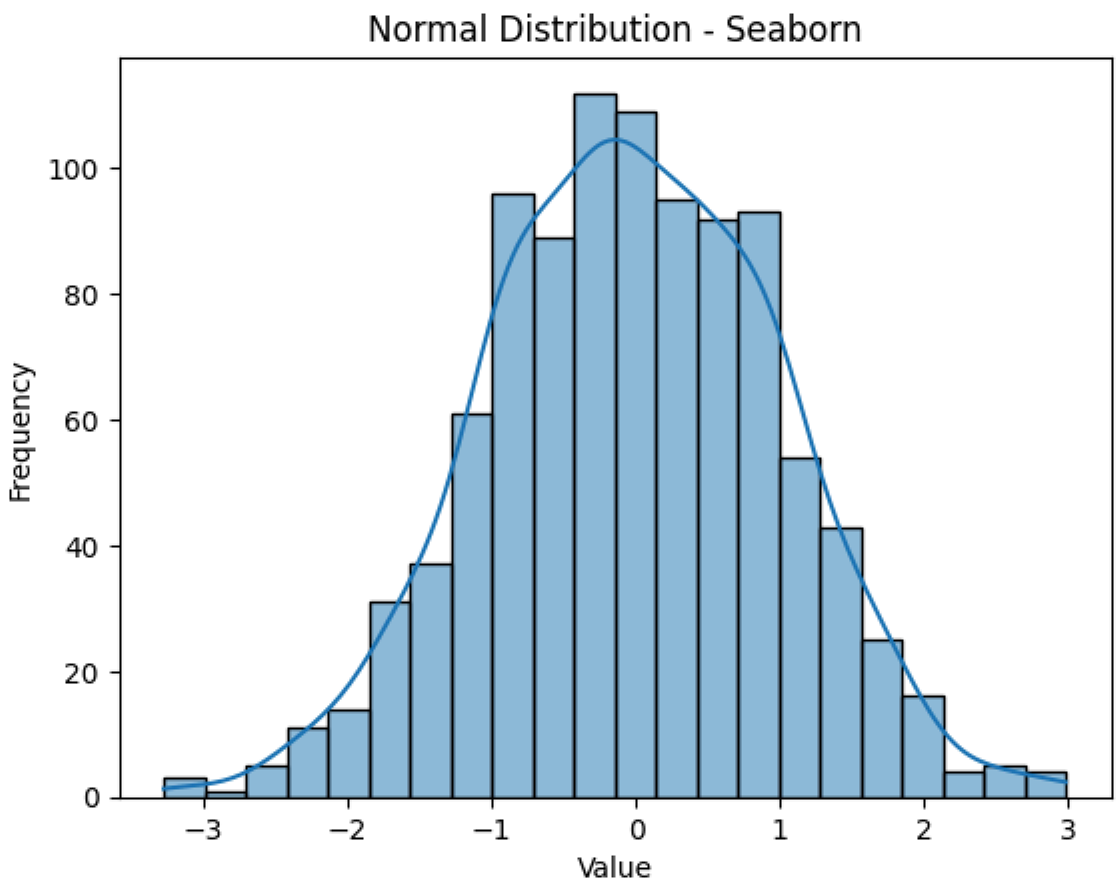
```
Tensor x with normal distribution:
 tf.Tensor(
[ 1.18578863e+00  3.72267866e+00 -1.56521797e+00  8.68656814e-01
 -5.73055863e-01 -1.09837949e+00  8.11640024e-01 -8.55692104e-02
  5.32908253e-02 -1.58845103e+00 -4.45458852e-02 -2.78721869e-01
 -3.92463580e-02  2.90007114e-01  3.13029140e-01 -2.29615545e+00
 -9.60833013e-01  1.95192134e+00  1.39154387e+00 -2.74391413e+00
  1.17492878e+00 -1.63094068e+00  8.88339520e-01 -6.80499196e-01
  4.19440836e-01  3.41154188e-01  6.18917763e-01 -8.32506537e-01
  1.78002107e+00 -1.29469946e-01 -2.63018787e-01  1.59049082e+00
 -2.25513220e+00 -2.30119869e-01 -1.11671376e+00  9.66658592e-01
  1.29689586e+00  1.25408399e+00 -1.32060134e+00  8.81832540e-01
 -6.08679175e-01 -2.16447830e+00 -1.47885442e+00  1.63991645e-01
  8.16944018e-02  2.24937677e-01 -1.63709319e+00  3.73152584e-01
  2.93470889e-01  2.10996604e+00  1.19010317e+00 -1.93489850e+00
 -6.73307553e-02  1.29171193e+00 -1.65402305e+00  6.31150842e-01
  1.45499206e+00  1.25369036e+00 -8.04224610e-01  6.26279593e-01
 -1.26731741e+00  1.46979237e+00  7.16250718e-01 -1.14773786e+00
 -5.37222326e-01 -1.61345080e-01 -2.80110687e-01 -7.48072088e-01
  8.93695712e-01  1.16366041e+00  6.26669705e-01  3.02442938e-01
 -1.66835463e+00 -2.04586178e-01 -2.02443862e+00 -1.17112911e+00
 -4.75236699e-02  8.15438986e-01 -3.84387851e-01 -6.06556416e-01
 -2.23792052e+00  2.86981344e-01  8.98756906e-02 -8.21209848e-01
 -7.19809681e-02 -1.67279279e+00 -3.46708924e-01 -2.46369541e-01
  1.67227793e+00  1.18795145e+00  1.64811254e-01 -9.35285032e-01
 -4.79571283e-01 -1.16501339e-01  1.39482450e+00 -1.81126869e+00
  2.33025655e-01 -1.00362033e-01  1.21883404e+00 -4.65724319e-01
 -9.65651795e-02 -1.30045843e+00  3.96555334e-01 -1.85564613e+00
 -7.32791722e-01 -8.60786200e-01 -1.11686981e+00  2.16917187e-01
  1.33617008e+00 -2.97078937e-02  1.59537649e+00  1.65257418e+00
 -4.02120084e-01  1.20646405e+00  4.96909231e-01  1.60654092e+00
  4.35058713e-01 -6.16126060e-01 -8.67983699e-02 -1.79918814e+00
 -4.28310275e-01  9.11129117e-01 -4.62929755e-02 -4.56054330e-01
 -2.34593779e-01 -1.18702948e+00 -6.38813525e-02  1.29923061e-01
  1.05018854e+00  1.04238617e+00 -1.90384109e-02 -2.32976545e-02
  2.21728611e+00  3.48958112e-02 -1.06930625e+00  1.85720325e-01
 -1.00902379e+00 -1.12869747e-01  1.95425391e-01  6.58256352e-01
 -9.34089661e-01  8.23701322e-01  9.65381682e-01 -2.16285557e-01
 -1.54472113e+00 -6.51981533e-01 -1.80575180e+00 -1.30892086e+00
 -1.29867804e+00  7.08454251e-01 -2.90908903e-01 -7.67282248e-01
 -3.46111655e-01  6.46970093e-01 -1.87902415e+00  8.82713571e-02
 -3.70557547e-01 -3.03445190e-01  6.34617984e-01  1.32660270e+00
  3.74580443e-01  1.77908272e-01  5.76188385e-01 -9.32048678e-01
 -3.87184769e-01  4.04034793e-01  2.52673388e-01  9.01714981e-01
  1.27943337e-01  1.31214762e+00  2.75065303e-01 -4.33529615e-01
  9.06077743e-01  2.12733340e+00 -7.31605351e-01  3.14684004e-01
 -3.04939717e-01 -1.58157933e+00  5.47934353e-01  2.38396859e+00
  1.53243315e+00  1.03005183e+00 -2.12538570e-01 -4.91898119e-01
 -3.58693451e-01 -9.11439717e-01 -5.82794845e-01 -9.25811604e-02
 -1.48489976e+00 -1.32843709e+00 -8.43822777e-01  6.86035275e-01
  1.02555418e+00 -9.30462778e-01  5.60021758e-01  1.08022368e+00
  1.23364866e-01  1.16110539e+00  2.75729090e-01  1.17819615e-01
  1.84680909e-01  6.10015154e-01 -1.68072641e-01 -9.05070841e-01
  6.53071702e-01  9.55092669e-01 -1.82161629e+00 -2.64804512e-01
 -4.94568259e-01  1.88533509e+00 -8.60172883e-02 -3.83839130e-01
  2.07478538e-01  1.38842329e-01  9.65097010e-01  2.24577832e+00
 -2.65008569e-01  1.72351933e+00  8.57271552e-02  7.75663316e-01
 -1.63026941e+00 -9.62931871e-01  3.31142098e-01  1.31538689e+00
  8.21454287e-01  7.47189939e-01 -2.53962278e+00 -2.12387480e-02
 -8.60472143e-01 -5.10164440e-01  4.65203077e-01  5.27584791e-01
  2.95763165e-01  8.00515562e-02 -1.68526876e+00  2.48845553e+00
 -9.81825411e-01 -2.44317278e-01 -3.50024492e-01  4.63972567e-03
 -1.45132005e+00  2.43357927e-01  5.83570451e-02  2.78305173e+00
  2.91733098e+00 -2.12496591e+00  2.07799882e-01  9.57011878e-01
 -1.49208857e-02  2.15542585e-01  7.08850145e-01  7.01040685e-01
 -8.33207816e-02  9.69646633e-01 -7.96869278e-01  3.80962402e-01
 -8.18421066e-01  8.98358285e-01 -1.88311666e-01 -5.71644127e-01
  1.18441093e+00  3.12394917e-01 -3.99957001e-01  1.85414279e+00
  1.14067286e-01 -1.10503510e-01 -1.15516937e+00  3.49619240e-01
  9.92366612e-01  1.12173092e+00  1.49378216e+00 -8.97918403e-01
 -5.13924956e-01 -6.92788064e-01  4.75247592e-01  2.73013562e-01
  1.72564638e+00  6.20501712e-02 -1.56225646e+00  9.61444736e-01
  2.30405641e+00  1.08544998e-01  1.00499618e+00  7.18468875e-02
 -8.36585045e-01  5.02715349e-01 -8.88256311e-01 -1.80410480e+00
 -1.42331815e+00 -1.26264572e+00 -3.07330787e-01  1.38781083e+00
 -1.08842218e+00  7.43698835e-01  1.70090854e+00  1.07346153e+00
  2.49530411e+00  6.10071540e-01  1.16660333e+00 -1.22157538e+00
  7.48330727e-02 -2.25522161e-01  1.17712736e+00  1.99818254e+00
  1.04439361e-02 -1.25568283e+00 -1.41097379e+00  4.88396734e-01
  1.04631388e+00 -1.02175033e+00 -3.03958267e-01  1.27446783e+00
  1.47859001e+00  1.29504645e+00 -8.22497427e-01 -1.36277592e+00
 -4.02091622e-01  6.85635030e-01  5.87338567e-01 -1.20903575e+00
 -3.91106993e-01 -5.05042672e-01 -2.03482318e+00 -4.05462801e-01
  2.31680885e-01 -9.65139419e-02 -2.70045847e-01  9.14934456e-01
  1.16444640e-01  5.97415939e-02  1.92307699e+00  8.28341961e-01
 -9.11821008e-01  3.96567822e-01 -7.04918921e-01  9.13238749e-02
 -6.03912473e-01  1.16896820e+00 -1.65494740e+00 -9.28980172e-01
  1.25786573e-01  8.66004452e-02 -4.17969465e-01 -6.26187980e-01
 -1.18143767e-01  9.24955964e-01 -5.35144687e-01  1.46933544e+00
 -1.78242862e-01 -1.12372085e-01  6.26438931e-02  1.09293211e+00
 -4.68932599e-01 -3.57379556e-01 -2.41990611e-01 -2.74918032e+00
 -2.40270376e+00 -8.54074478e-01 -1.27107716e+00  4.60877776e-01
 -3.69139552e-01  1.37877703e+00 -3.86337221e-01 -7.21480548e-01
  4.63214636e-01 -1.15071833e+00 -8.65034938e-01  1.52985203e+00
  1.94198859e+00 -1.28611386e+00  4.96164173e-01 -4.78992201e-02
 -5.10535613e-02 -9.43624079e-01  5.42336106e-01  1.03680420e+00
 -2.05126691e+00  5.51494360e-01 -1.59454572e+00 -5.48017263e-01
 -1.30743313e+00  6.27625108e-01  1.61771846e+00 -2.35086411e-01
 -1.67281479e-01 -2.97607422e-01  1.66205204e+00 -8.82497549e-01
 -1.42957759e+00  6.04724467e-01 -2.69940996e+00  1.07787013e+00
 -2.68306211e-02  2.56282282e+00  6.65077388e-01 -1.18440008e+00
  1.06367040e+00 -1.56037688e+00  5.36276400e-02  1.33565784e+00
  1.66155064e+00  1.50091672e+00  8.06642830e-01  1.97871673e+00
  3.84136975e-01  3.09676081e-01  1.60190725e+00 -5.24422765e-01
  9.90601122e-01  2.33236507e-01 -2.81114817e-01 -3.05631042e-01
  3.71080309e-01 -2.49138519e-01  9.56856132e-01  5.37780896e-02
 -1.45038080e+00  4.42295969e-01  2.05881655e-01  7.19876707e-01
  2.08166742e+00  1.84147012e+00  6.65461302e-01 -2.09969580e-01
  5.40308952e-01 -7.60009825e-01 -8.10544789e-01 -8.68284628e-02
  2.49437571e+00 -7.80179143e-01 -7.97421813e-01 -1.32816926e-01
  9.62312400e-01  4.57938105e-01  7.80296922e-01  5.77697493e-02
 -3.04546505e-01  1.77659050e-01  1.36771345e+00  1.95734516e-01
 -1.03721693e-01  1.21257317e+00  6.24195933e-01  1.02342606e+00
 -7.23202452e-02  3.57889831e-02  4.42556709e-01 -3.68068248e-01
 -1.04733169e+00 -1.39827633e+00 -8.67964253e-02  8.98158431e-01
  3.59083951e-01  2.56910443e-01  3.75266224e-01  3.53646219e-01
  3.66618186e-01 -1.16076517e+00  1.27143991e+00  8.31925273e-01
  1.78299531e-01  3.09864759e-01 -9.57010388e-02  4.85381871e-01
  1.62634552e+00 -7.54909694e-01 -2.05393240e-01 -2.75385231e-01
  9.01520252e-01  5.55379093e-02  5.01172960e-01 -1.04015493e+00
  7.52902254e-02  8.86422172e-02 -2.58214760e+00  7.87023664e-01
 -6.26838923e-01  8.57490957e-01 -7.94493079e-01  1.78836972e-01
  1.18054879e+00 -1.69517064e+00 -1.21288359e+00 -1.20042729e+00
 -1.61236487e-02 -2.97200680e-01  1.62102604e+00  1.76025212e+00
 -1.81875646e+00 -1.40194550e-01  7.81539440e-01 -1.20085990e+00
 -4.49058227e-03  1.36808562e+00 -1.05571306e+00 -1.15783250e+00
 -9.98385906e-01  1.58116543e+00 -3.55448127e-01 -1.60769865e-01
  8.94927308e-02 -5.56299463e-02 -1.41700613e+00  2.35371336e-01
  2.69524312e+00 -1.72509328e-01 -2.50368685e-01 -1.57536709e+00
  5.30594826e-01 -1.39894605e-01 -1.18132114e-01  7.55523741e-02
  1.22613084e+00  1.25007784e+00  1.40204161e-01  4.47994441e-01
 -5.87202251e-01  5.78511655e-01  1.17280655e-01  5.21844804e-01
 -9.33748335e-02  7.66856819e-02  8.65343809e-01  6.99893907e-02
 -2.19375896e+00  4.60745037e-01 -1.03834414e+00  8.40740979e-01
  7.17817992e-02  1.43580592e+00 -1.29371059e+00  4.41180974e-01
  5.07701710e-02 -9.46941003e-02 -7.05525577e-01 -1.47978449e+00
 -9.38834667e-01 -8.00096691e-01  5.24788141e-01 -8.63434911e-01
  1.40836394e+00 -9.04950559e-01  3.13096911e-01 -1.86882627e+00
 -5.96143425e-01 -9.18754280e-01 -1.17343855e+00 -2.53095031e-01
  5.36304653e-01 -9.22594905e-01 -1.80674052e+00 -8.05286229e-01
 -1.02809596e+00  1.11453414e+00 -1.52846307e-01  4.39555615e-01
 -7.51157880e-01 -3.24752003e-01  8.56805861e-01  1.15723503e+00
  1.35848725e+00 -2.55388498e-01 -3.84458899e-02 -3.61050338e-01
 -1.06919599e+00  5.15415549e-01  4.34053600e-01  3.59995365e-01
  8.09516370e-01 -1.09215707e-01 -7.98636794e-01 -1.54605716e-01
 -4.06829864e-01  3.21335942e-01 -8.75662044e-02  8.56710076e-01
  2.36596441e+00 -3.40825133e-02  2.49028295e-01  6.98369503e-01
  1.34357202e+00 -1.50243962e+00  1.13358927e+00 -2.10411295e-01
  2.32099697e-01 -1.59488261e+00  6.43980503e-01  1.31376064e+00
 -9.10311997e-01 -1.05583847e+00 -9.19876456e-01 -4.00567919e-01
  1.30073524e+00 -1.49008358e+00 -6.63631380e-01  9.02255952e-01
 -2.13364676e-01 -1.09926440e-01 -8.38579953e-01  5.47843039e-01
  1.31856859e+00  5.23015201e-01 -2.08413291e+00  1.34315968e+00
 -3.91488642e-01  2.62663221e+00 -1.20974325e-01  6.46316230e-01
  1.01762259e+00  1.66900635e-01 -9.87403691e-01  2.29211345e-01
 -3.73278469e-01  5.10265008e-02  9.25839961e-01 -9.84462678e-01
  8.83080363e-02 -3.80326509e-01 -1.99229407e+00  6.73711419e-01
  1.54751527e+00  3.99592131e-01  4.72367294e-02  3.07443768e-01
 -4.14292932e-01 -8.17186832e-01  1.56834155e-01 -9.40814257e-01
  1.14200795e+00 -2.55913250e-02 -2.06749153e+00 -1.53809279e-01
  1.49739575e+00  4.23451781e-01 -4.62215543e-01  1.57173133e+00
 -1.59983897e+00  1.28142285e+00 -5.79416215e-01 -1.74371290e+00
 -9.35631394e-01  4.36492443e-01 -7.18186677e-01 -8.06355178e-01
  5.90688169e-01 -7.47750461e-01  3.35031718e-01 -7.09158003e-01
  9.42774355e-01  3.58439058e-01  1.03168875e-01  9.81983423e-01
 -2.35092670e-01 -3.45733762e-01 -1.34458566e+00  2.47356948e-03
 -1.16438663e+00  2.21012965e-01  5.83003908e-02 -1.23448217e+00
 -1.98748142e-01  9.69588995e-01  6.79809928e-01  1.72278631e+00
```

```
  1.03902757e+00  6.72178626e-01 -4.53478485e-01 -2.80519307e-01
  8.49068820e-01  8.33008349e-01  2.56103694e-01  1.65657356e-01
 -9.17127058e-02  1.08471465e+00  1.00933087e+00 -9.62246239e-01
  1.29741788e+00  1.79988444e+00 -2.95133770e-01  3.71063560e-01
  1.10318637e+00  4.90352124e-01 -1.43745673e+00  8.82222354e-02
  9.14059997e-01  6.44439042e-01 -8.85722578e-01  1.05377865e+00
 -1.55081868e-01  4.65666115e-01 -1.27428889e+00  5.25220811e-01
  7.71696150e-01  1.85563242e+00  7.10725337e-02 -3.45014811e-01
 -1.00420702e+00  7.89937377e-01 -2.84262560e-03 -7.01118112e-01
  7.11212754e-02 -1.14316237e+00  9.99931872e-01 -8.45611870e-01
 -1.81435704e-01  9.41431344e-01  4.54553574e-01 -3.50366771e-01
 -1.09667599e-01  7.79058754e-01  3.29567313e-01 -6.94090009e-01
 -2.45656595e-01 -1.88469365e-01  2.18174410e+00  4.90905702e-01
 -3.45268101e-01  4.64978907e-03 -1.77393660e-01 -2.99714833e-01
  8.28246713e-01 -1.69046044e+00  1.34833187e-01 -4.17860448e-01
  1.30471826e+00 -6.48373663e-01  3.74794863e-02 -5.46568692e-01
 -1.05029964e+00 -3.60069275e-01  1.81015477e-01 -8.28321397e-01
  1.36661232e+00 -1.03406167e+00 -5.78294933e-01  1.34184882e-01
  2.30018854e-01  2.22658187e-01  4.65899110e-01  1.64993167e+00
  4.47759777e-01 -1.11285493e-01 -1.17583001e+00  1.80091095e+00
 -5.98747671e-01  5.98955929e-01  4.84606564e-01 -1.07799137e+00
  2.00483818e-02 -8.31659883e-02 -3.80807519e-01 -9.92318511e-01
  4.57762554e-02  1.07598913e+00 -6.08315170e-01  6.53789997e-01
 -7.41061196e-02  2.33299121e-01  4.72386390e-01  6.49226308e-01
 -1.21940768e+00 -4.71177965e-01 -6.17528319e-01  6.32061601e-01
 -1.27936161e+00 -3.52970570e-01 -2.20987145e-02 -7.45447159e-01
  1.03242326e+00 -1.77942246e-01  3.00206691e-01  1.83347255e-01
  5.84334694e-02 -2.99653560e-02 -9.12860811e-01 -1.49950123e+00
  7.21684039e-01  1.03314042e+00 -6.19755745e-01  2.29912713e-01
  1.69026591e-02  4.63764161e-01 -4.95323092e-01 -1.18850458e+00
  5.76034606e-01  5.14268935e-01 -1.08382154e+00  1.11175168e+00
 -1.33817255e+00  1.47685492e+00  4.48717833e-01  2.07448140e-01
 -2.21178010e-01  9.53557909e-01 -9.03755009e-01  3.36526394e-01
 -2.52940178e-01  1.50306642e+00 -2.13100463e-01 -1.27305138e+00
 -1.61344957e+00  3.93339247e-01  1.95928466e+00  1.35533661e-01
  1.04008503e-01  1.78355455e+00 -1.37265790e-02  2.21621558e-01
  9.76902962e-01 -1.16874754e+00  9.40307453e-02 -2.84058247e-02
  2.77459472e-01 -1.56592160e-01  5.58626115e-01  4.51481879e-01
  2.93265283e-01  2.16320470e-01  5.91637604e-02  7.81663001e-01
  8.40634704e-01 -4.11433935e-01 -4.98080403e-01 -1.34866011e+00
  2.54743248e-01 -3.15125704e-01 -1.38371515e+00 -1.29628778e+00
 -8.89316261e-01 -1.36078335e-02  2.00506759e+00 -5.85907817e-01
 -2.95195460e-01 -2.54368693e-01  4.99376655e-01 -4.16349411e-01
  1.19181538e+00 -1.59526432e+00 -1.04736142e-01 -5.41280150e-01
  3.31110477e-01  6.48152113e-01 -7.67483771e-01  2.44284227e-01
  1.64173633e-01  1.08771038e+00  1.06236136e+00 -4.03935522e-01
 -3.75180066e-01  9.73935843e-01  1.03057313e+00  1.41475201e-01
  2.79210472e+00 -2.04739046e+00  1.62255180e+00 -1.51790833e+00
  1.88343912e-01  3.80027682e-01  5.65427661e-01 -8.55795860e-01
 -1.24943519e+00  3.78414035e-01 -1.51475474e-01 -1.34469330e-01
 -6.04847968e-01 -8.79671395e-01  1.12669539e+00 -8.91741931e-01
  1.81570959e+00 -1.63525188e+00  8.34951639e-01  6.30612746e-02
  8.31995964e-01 -9.35142994e-01  1.30960596e+00 -1.68599281e-02
  3.11810166e-01 -2.61986345e-01  4.78357196e-01 -2.40787935e+00
 -2.35710517e-01  1.42955244e+00  1.65788150e+00 -6.21663570e-01
 -1.01496851e+00 -1.45336366e+00  8.52359653e-01  4.45208967e-01
```

```
  1.61813295e+00  5.08834422e-01 -8.85311723e-01  2.10841918e+00
  3.99100959e-01  2.60587901e-01 -3.13190722e+00 -2.09997082e+00
 -1.34703732e+00  1.18994877e-01 -3.11548948e-01 -4.16385323e-01
 -8.06280553e-01 -3.64086807e-01  4.32822436e-01  9.00708020e-01
  2.15367705e-01 -4.97623444e-01 -1.51349783e+00  9.42952454e-01
 -4.91914988e-01  4.26336288e-01 -5.15278697e-01 -7.39468038e-02
 -4.23374660e-02  1.07825947e+00  7.00849712e-01  3.80528182e-01
  7.62940049e-01 -6.07119918e-01 -9.09379050e-02  4.00282629e-02
 -7.83335268e-01 -8.64780962e-01  3.45896512e-01  9.38893199e-01
 -2.29603887e-01  3.90435457e-01 -1.05430312e-01 -1.70474029e+00
  1.18579316e+00  1.11773491e+00  2.03641832e-01 -1.61984575e+00
  1.81115493e-01 -7.11131930e-01 -1.62925255e+00 -9.93111208e-02
 -4.76916403e-01 -5.18865824e-01 -2.17544246e+00  6.60027385e-01
 -6.48131549e-01  3.55808586e-01 -1.47463620e-01 -1.28337324e+00
 -3.04945797e-01 -7.13108480e-01  9.60637152e-01  1.02595043e+00
 -1.00710189e+00 -1.17968440e+00  3.43009770e-01  8.27341557e-01
  1.03840101e+00 -2.25874138e+00  1.90780491e-01  3.36598307e-01
  2.98521399e-01 -1.23326075e+00 -1.06540835e+00 -5.97224355e-01
 -1.13251293e+00 -7.21805632e-01 -1.46611357e+00 -9.46125209e-01
  9.45555031e-01 -5.02633870e-01  9.03976619e-01  9.83590424e-01
  9.44352627e-01 -8.37844610e-01 -6.89557195e-01 -5.92166483e-02
  1.03155768e+00  5.51636398e-01  4.16954905e-02  1.52416289e+00
 -1.56295210e-01 -2.30887875e-01  1.71049225e+00  1.15012813e+00
 -1.34578431e+00 -1.16530871e+00  6.51491463e-01  1.32586825e+00
  3.36383522e-01  2.29262424e+00  7.05810666e-01 -1.19151580e+00
  1.67016160e+00  4.70115244e-01  3.53613682e-02  9.94555652e-01
  8.15048456e-01  2.85822582e+00 -1.65271246e+00 -1.31215617e-01
 -2.24312210e+00 -2.24884152e+00 -5.87991774e-01 -1.65532434e+00], shape=(1000,), dt
ype=float32)
```

Using seaborn, visualize the distribution

In [33]:
```python
import tensorflow as tf
import seaborn as sns
import matplotlib.pyplot as plt

x = tf.random.normal(shape=(1000,), mean=0, stddev=1)

x_numpy = x.numpy()

sns.histplot(x_numpy, kde=True)
plt.title("Normal Distribution - Seaborn")
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.show()
```



Using the random.uniform() function, create a tf. Tensor with values sampled from a uniform distribution with the following parameters:

- shape 6 * 6
- minval -10
- maxval 20
- dtype int32

In [36]:
```
uniform_tensor = tf.random.uniform(shape=(6, 6), minval=-10, maxval=20, dtype=tf.in

print("Tensor with Uniform Distribution:\n", uniform_tensor)
```

```
Tensor with Uniform Distribution:
 tf.Tensor(
[[  2  19  12  16   9   6]
 [ 12  -3   7  -4   9 -10]
 [ 16  -5   9   3  12  16]
 [ -2   1  18   0  12   2]
 [ -5  17  -9  -6   0   1]
 [  6   5  -3  -9  16  17]], shape=(6, 6), dtype=int32)
```

create a TF uniform distribution and call it x2
use a shape=(1000,)

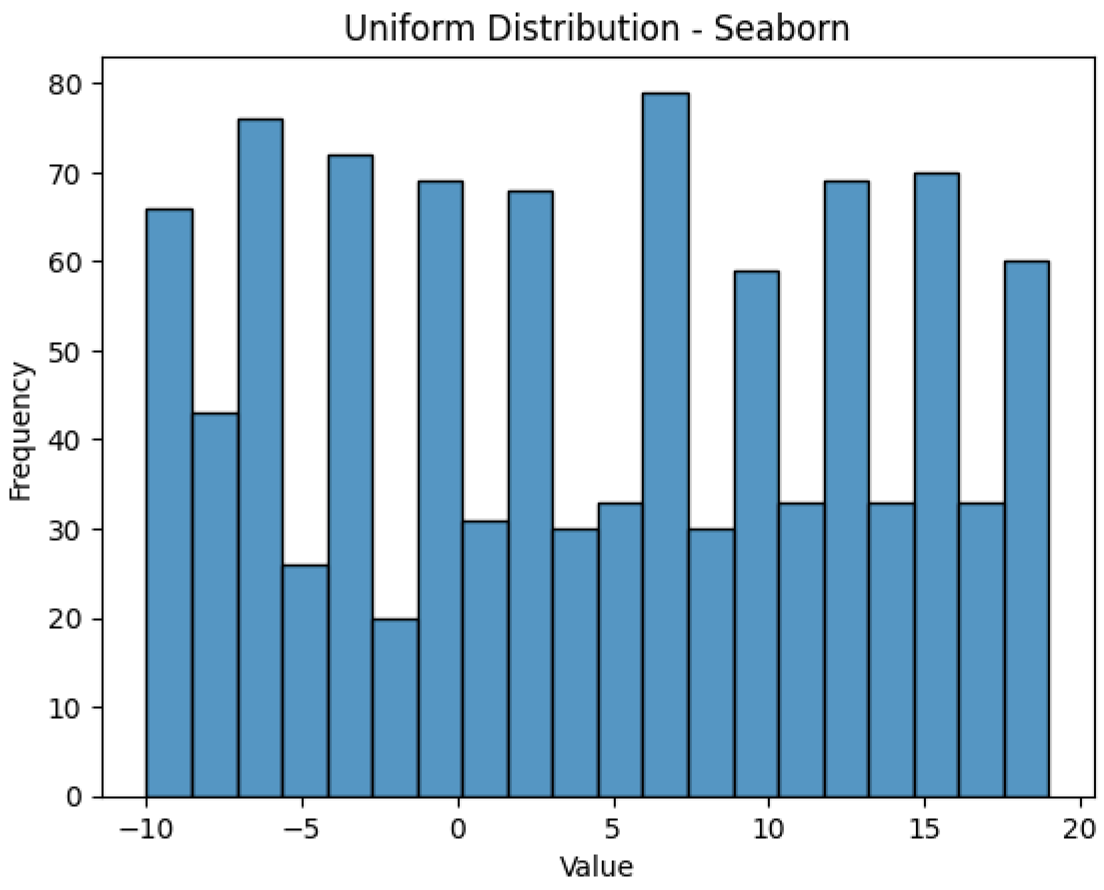
In [38]:
```
x2 = tf.random.uniform(shape=(1000,), minval=-10, maxval=20, dtype=tf.int32)
```

```
print("Tensor x2 with Uniform Distribution:\n", x2)
```

```
Tensor x2 with Uniform Distribution:
 tf.Tensor(
[ 11  17  -1  17  -6  12  -2   6   5  -3  12  18   5   1   0   3 -10   9
  19  13  17  11  12  -1  -7  -5 -10  -7   2  10  -5  -2  -1   8  -8   5
  -2  12  11   8  17   2  14  14   2   1   7  -4  -5  12  -7  13   2  10
   1  10   7   8  14   2  19   6  -9  13   0  -5   9  10  -1  16  -5  -7
  13   2   8  15  19   1  17  10  -2   3   3   0  10  16   3  -2  10  11
  -9   5  -2   5  12  18   6  -7  -2  -4   8   9  -9  -8   1  16   2   0
   2  -7  17   6   6 -10  -7   5  17  -3  -5  17   2  -3   0   1  13   5
  16  14   4  -5  16  -3   6  17  19  14   3  12   1  -4   9  -5  16   6
   4   8  13  -7   7  -4  -4   4 -10   0  12 -10  -5  14  -9  16   5  -1
  -4   4   2   9  16   4  18  14 -10 -10  10   3   6  13  13   9   1  16
   8   5  -2  16  -9  13  -4   4   9  -6  -8  16  16   0  -7  14   4  10
   0  10  -1  -7   7  19 -10  -5  -2  15  12  19   6   8  -1   0   9  -9
  -4   3  15  17  10  -8  -5  15   8  12   2  13  -3  -7  -3  10   2   0
  -5   3  -7  -2  15   0   4   9  14  17   8  12  -7   3  16   6  -3   9
  17  10 -10  -3  -6  -2   0  -7  -2   8  12  14  -2   2  -7  13   3  -2
   9   6   1  -2  13   9 -10   3   5   3  19 -10  -7  18   4   2  -3  10
   1  15  19  17   2   5   0   3  19   7  15 -10  -4  17   8  10   7  -9
   2  10   8  15  10  -5  -3  11   9  -2  13  -1   0   6  -6  17   2   5
  -3   0  18   3  12   9  10  16  15  17  -8  13  19  13  12   9  -9   7
  15  -1  -1  -9  19  -6   1   1  15   3 -10   7  -6   3  14   1   7   1
   6  16   1  17   5  13   4   5   0  -2  -5  18  16 -10  -7   0  -4   1
   5   3  -9  16   1   8   3  17  18   1  -7  15  -5  14   1  -1  16  18
  13   9   8  16   1  -3  -5  -4  -2  19   9   5  17   5  -4   8  11  11
  -4  18  -1  10  10 -10   8   3  -5  -2   3  -1   7  -5 -10  15  15   6
 -10   3  -2  14   1  -3  15  13  -4   2   7 -10   6  -3  14   5  -6 -10
   7  -5  18  -5   2  -9  -5   8  14   7   2  17 -10   2  10   9  18   9
   3  12  11  17  -6  -2   6  14  11   5   6 -10   8  10  -7   9  13  -1
   6  15   1  11   6  19  -9  11   9  -5   4 -10  12   7  -9   3  -1  15
  -5  11   4   0  17  -5  17  -8  11  -6  -1   1  19   4  16  -6  -3  13
  -4   8   9  10   8  19  14   7  18   3  -8  16   5  -4   5  -6  -1  10
   8  17  17  19   3   3  19  13   5  -1  19  -4   7   5   3  -5   3  11
  -6  18  15  18  12 -10  -6  -1   8  14   3  18  13   7  -2  -9  14   9
  -1   7  12  -3  -8  12   8   6   7  -1 -10   0   0  12   3   1  -9  -7
  -9  14   5  15   3  -3   0  11  10  -3   9  11  10  -3  -3   7   2  -4
   8   9   1   1  11  -2   7  -7  -1   9  18  18   4  -5  -7   7  11  14
  -2   2  -6  19  -7  -6  -6  17  14  -6   0   8   9  11  -8   7   3   4
 -10   0 -10  13  17   7  10  -6  17  -6   0  17   0  17   2  16  -4  15
  14   6  16   8  16  13  15   8  -9   7  -7  -1   7  -3   8   5   6  -8
   2  18   3  17   5  15   0  17 -10  19   4   6  -1  -5  14  -7  18 -10
  19   3  -8   7  -5  12   7  -9  -4   7  14   3  11  19  14  17   6  -4
  18  -9   5  -3   7  -7   5  -6  18  17  -1  -6  -4   6   3   2  -2  -8
  16  -1  -7  13  -7   0  -9  18  19   1  -2   6  14   5  -9  10   8  19
  16  -4   2  -9   5  -6 -10 -10   3  18   1  -9  16  15   8   0  10   4
  -3  16   8  11  -1   4  18   1  -1  12   3   7   0  10   6  -9  15  -2
  12  -7  16   1   3   2  -9  17   9  -5   7   1   8  17  18   9   7  12
  14  13   7  -4  -3  -8   5  10  -7  -9  -4   6   8  -4   7  -6  13  -7
   6  -6   1  -3  -3   7   7   9  -4  -1  -4  -2  12  -3  -9   1  -1  -9
  -9  15  -3 -10  19  -6  -1   5  19  17  13   6  -2   1   8   8  -4  15
  -5   6   5  -6  18  10  17   1  15  11  -9  17  -2  -4  17  17  13  -2
  -1  10   7  18  10  -8   4  -9  -5   6   1   4   9  -5  13  15  -5  -2
  -8  -7  -9  -5  -8   4  11  18   7  15   8  -1 -10  -5  -3   7 -10  18
  -8  -2  18   7  -1  -4   1  -7   0  -5   2  -7   1  -4   8   5  -1  -2
   5 -10  -7   5 -10  18   9   3  11   5  -4  -5  19  -3  12   9  10   4
  16  10 -10  -8   8  19  -2  -5   2   6  11  -9   8   3  -5  11  14   9
  15  -8  16   2  17 -10  14  -1 -10  -8  15   9   4 -10  16  11  -9  16
   2   9   9   5   4   6   3   8  19  -2], shape=(1000,), dtype=int32)
```

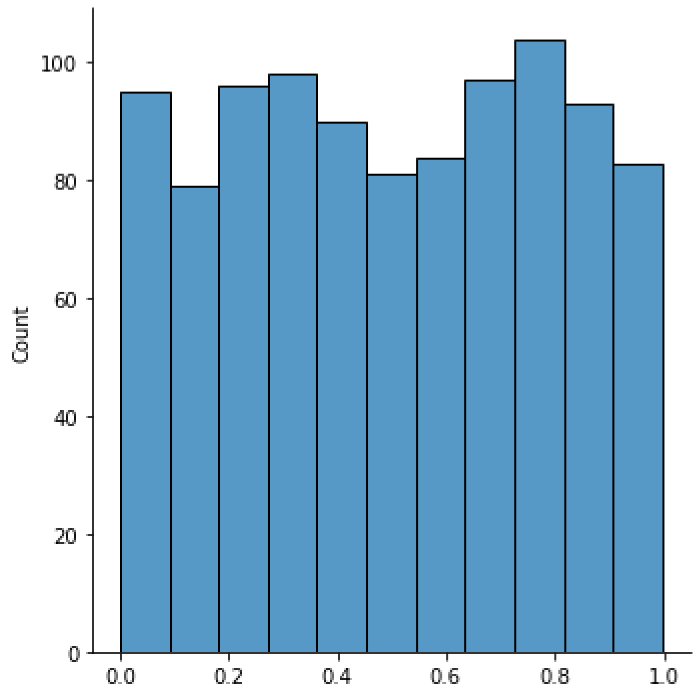
Using seaborn, visualize the uniform distribution (x2)

In [40]:
```
x2 = tf.random.uniform(shape=(1000,), minval=-10, maxval=20, dtype=tf.int32)

x2_numpy = x2.numpy()


sns.histplot(x2_numpy, kde=False, bins=20)
plt.title("Uniform Distribution - Seaborn")
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.show()
```



In [40]:
```
# sample
```

Out[40]:
```
<seaborn.axisgrid.FacetGrid at 0x269a1588a30>
```



Create a tensor with four digits based on a python list usign tf.Variable

In [43]:
```
python_list = [1, 2, 3, 4]


tensor_variable = tf.Variable(python_list)

print("Tensor Variable:\n", tensor_variable)
```

```
Tensor Variable:
 <tf.Variable 'Variable:0' shape=(4,) dtype=int32, numpy=array([1, 2, 3, 4])>
```

create a numpy ndarray with 4 matrices, 5 rows and 3 columns and call it np_arr

In [46]:
```
np_arr = np.random.randn(4, 5, 3)

print("NumPy ndarray:\n", np_arr)
```

```
NumPy ndarray:
 [[[ 1.16637053  1.88972155  0.38768833]
  [ 0.56797143 -1.33899746 -0.79650015]
  [ 0.81763405  0.43595466 -0.0598241 ]
  [-0.62377997 -0.72567091  0.54317106]
  [-0.0409049   1.18518042  0.27433473]]

 [[ 0.02501756 -0.43527078 -1.4947295 ]
  [-0.51377207 -0.68839786  0.50215331]
  [-0.83459184  0.25800725 -1.17162278]
  [-0.43350181  0.25519571  0.05549665]
  [-1.4455831   0.8036611   0.04935702]]

 [[ 0.031375    0.91795198 -0.59843246]
  [-0.69465554  0.93054987 -0.15856378]
  [-0.26814334 -0.06958831  1.46901608]
  [ 0.13716447 -0.91387489  0.18016075]
  [ 1.02170992  0.74184497 -0.5481294 ]]

 [[ 1.78017513 -0.62919283 -0.21033033]
  [ 0.4176535  -0.42508607  0.56896721]
  [ 1.75189333 -1.04537321 -0.1164876 ]
  [-0.6042983  -0.91042394  1.76663789]
  [ 1.04772412 -2.05160325 -1.47544677]]]
```

create a tensor using the ndarray from the step above and call it: tf_tens

In [48]:
```
np_arr = np.random.randn(4, 5, 3)


tf_tens = tf.convert_to_tensor(np_arr)

print("TensorFlow Tensor:\n", tf_tens)
```

```
TensorFlow Tensor:
 tf.Tensor(
[[[-0.01568786  0.88470831 -0.58947643]
  [ 0.53652153  1.20740925  0.64695274]
  [ 0.37447112  0.09720591 -0.57799115]
  [ 0.90687358  0.15537997  1.3793538 ]
  [-0.32806043 -2.18257449 -0.1715082 ]]

 [[ 0.12469373  0.34474936 -1.11825915]
  [-0.08913177  0.4073517  -1.59117261]
  [ 0.74689881  1.42851867 -0.27648229]
  [-1.08202353 -1.91649837 -0.12249094]
  [-0.3060889   2.38714406  0.67625511]]

 [[-1.07508604  1.02355133 -1.49400113]
  [ 0.14389499  0.21766064 -0.27696355]
  [-0.96145555 -0.02985926  0.72744302]
  [ 0.66906518  0.71953549  0.2947914 ]
  [-1.63218882  1.03190166 -0.45338768]]

 [[ 0.34118281  0.71856273  0.51229218]
  [ 1.78580545 -0.04125997  0.80347195]
  [-0.04254271  1.57874896 -1.20404833]
  [ 0.17506544 -0.07576294 -1.15541097]
  [-0.60221055  0.97041657  0.80873114]]], shape=(4, 5, 3), dtype=float64)
```

Basic Operations

Print the tensor tf_tens

In [51]:
```
print("Tensor tf_tens:\n", tf_tens)
```

```
Tensor tf_tens:
 tf.Tensor(
[[[-0.01568786  0.88470831 -0.58947643]
  [ 0.53652153  1.20740925  0.64695274]
  [ 0.37447112  0.09720591 -0.57799115]
  [ 0.90687358  0.15537997  1.3793538 ]
  [-0.32806043 -2.18257449 -0.1715082 ]]

 [[ 0.12469373  0.34474936 -1.11825915]
  [-0.08913177  0.4073517  -1.59117261]
  [ 0.74689881  1.42851867 -0.27648229]
  [-1.08202353 -1.91649837 -0.12249094]
  [-0.3060889   2.38714406  0.67625511]]

 [[-1.07508604  1.02355133 -1.49400113]
  [ 0.14389499  0.21766064 -0.27696355]
  [-0.96145555 -0.02985926  0.72744302]
  [ 0.66906518  0.71953549  0.2947914 ]
  [-1.63218882  1.03190166 -0.45338768]]

 [[ 0.34118281  0.71856273  0.51229218]
  [ 1.78580545 -0.04125997  0.80347195]
  [-0.04254271  1.57874896 -1.20404833]
  [ 0.17506544 -0.07576294 -1.15541097]
  [-0.60221055  0.97041657  0.80873114]]], shape=(4, 5, 3), dtype=float64)
```

Convert the tensor tf_tens to a numpy array and name it converted_np_arr

In [53]:
```
converted_np_arr = tf_tens.numpy()

print("Converted NumPy Array:\n", converted_np_arr)
```

```
Converted NumPy Array:
 [[[-0.01568786  0.88470831 -0.58947643]
  [ 0.53652153  1.20740925  0.64695274]
  [ 0.37447112  0.09720591 -0.57799115]
  [ 0.90687358  0.15537997  1.3793538 ]
  [-0.32806043 -2.18257449 -0.1715082 ]]

 [[ 0.12469373  0.34474936 -1.11825915]
  [-0.08913177  0.4073517  -1.59117261]
  [ 0.74689881  1.42851867 -0.27648229]
  [-1.08202353 -1.91649837 -0.12249094]
  [-0.3060889   2.38714406  0.67625511]]

 [[-1.07508604  1.02355133 -1.49400113]
  [ 0.14389499  0.21766064 -0.27696355]
  [-0.96145555 -0.02985926  0.72744302]
  [ 0.66906518  0.71953549  0.2947914 ]
  [-1.63218882  1.03190166 -0.45338768]]

 [[ 0.34118281  0.71856273  0.51229218]
  [ 1.78580545 -0.04125997  0.80347195]
  [-0.04254271  1.57874896 -1.20404833]
  [ 0.17506544 -0.07576294 -1.15541097]
  [-0.60221055  0.97041657  0.80873114]]]
```

tf.range

Create three variables: start, limit, and delta, and assign positive numbers to them. Make sure the limit is greater than the start.

```
In [58]: start = 5
         limit = 15
         delta = 2

         print("Start:", start)
         print("Limit:", limit)
         print("Delta:", delta)
```

```
Start: 5
Limit: 15
Delta: 2
```

Using tf.range, create a sequence using the three variables from the step above and save them in a variable called r

```
In [66]: r = tf.range(start, limit, delta)

         print("Tensor r:\n", r)
```

```
Tensor r:
 tf.Tensor([ 5  7  9 11 13], shape=(5,), dtype=int32)
```

Find any of the values from the previous range using tf.where and replace it with a value of 500, and save it in the same variable r

```
In [75]: r = tf.where(r == 7, 500, r)

         print("Updated Tensor r:\n", r)
```

```
Updated Tensor r:
 tf.Tensor([  5 500   9  11  13], shape=(5,), dtype=int32)
```

Create a multidimensional tensor of shape (5,7,3) with random values from 1 to 20 and name it "t"

hint: you can use random.uniform()

```
In [79]: t = tf.random.uniform(shape=(5, 7, 3), minval=1, maxval=21, dtype=tf.int32)
```

print the tensor t

```
In [82]: print("Tensor t:\n", t)
```

```
Tensor t:
 tf.Tensor(
[[[ 6 17 17]
  [14 16  2]
  [10  6  6]
  [11  5  4]
  [12 14 16]
  [ 2 17 13]
  [ 2  7 20]]

 [[13 11 19]
  [ 5 17 13]
  [13  9 15]
  [ 9 19  2]
  [ 6  1 14]
  [ 6 18 13]
  [ 6 11  3]]

 [[ 6 17  3]
  [ 7 19 18]
  [ 5 17 17]
  [13  9 19]
  [16  1  4]
  [ 6 11  7]
  [10  8 10]]

 [[13  3  2]
  [13 20  1]
  [18 10  8]
  [ 4  1  3]
  [12 20 19]
  [ 7  5 10]
  [ 2  7 20]]

 [[ 9 15  6]
  [20  9  6]
  [13 11  5]
  [12  4 16]
  [17 17  2]
  [14 15 17]
  [10  6 12]]], shape=(5, 7, 3), dtype=int32)
```

Compute the sum of elements across dimensions of tensor "t"

In [99]:
```python
total_sum = tf.reduce_sum(t)
print("Sum of all elements in tensor t:", total_sum)

sum_axis_0 = tf.reduce_sum(t, axis=0)
print("Sum along axis 0:\n", sum_axis_0)

sum_axis_1 = tf.reduce_sum(t, axis=1)
print("Sum along axis 1:\n", sum_axis_1)

sum_axis_2 = tf.reduce_sum(t, axis=2)
print("Sum along axis 2:\n", sum_axis_2)
```

```
Sum of all elements in tensor t: tf.Tensor(1097, shape=(), dtype=int32)
Sum along axis 0:
 tf.Tensor(
[[47 63 47]
 [59 81 40]
 [59 53 51]
 [49 38 44]
 [63 53 55]
 [35 66 60]
 [30 39 65]], shape=(7, 3), dtype=int32)
Sum along axis 1:
 tf.Tensor(
[[57 82 78]
 [58 86 79]
 [63 82 78]
 [69 66 63]
 [95 77 64]], shape=(5, 3), dtype=int32)
Sum along axis 2:
 tf.Tensor(
[[40 32 22 20 42 32 29]
 [43 35 37 30 21 37 20]
 [26 44 39 41 21 24 28]
 [18 34 36  8 51 22 29]
 [30 35 29 32 36 46 28]], shape=(5, 7), dtype=int32)
```

Get the mean, max, and min values of tensor "t"

In [92]:
```python
mean_value = tf.reduce_mean(t)
print("Mean value of tensor t:", mean_value)

max_value = tf.reduce_max(t)
print("Max value of tensor t:", max_value)

min_value = tf.reduce_min(t)
print("Min value of tensor t:", min_value)
```

```
Mean value of tensor t: tf.Tensor(10, shape=(), dtype=int32)
Max value of tensor t: tf.Tensor(20, shape=(), dtype=int32)
Min value of tensor t: tf.Tensor(1, shape=(), dtype=int32)
```

Were you expecting any or all of these values and why?

I expected the values to be in the range of 1 to 20, with the mean around 10.5, and the maximum and minimum values should lie at the boundaries of this range.

Compute the mean of elements across dimensions of tensor t

In [97]:
```python
mean_all = tf.reduce_mean(t)
print("Mean of all elements in tensor t:", mean_all)

mean_axis_0 = tf.reduce_mean(t, axis=0)
print("Mean along axis 0:\n", mean_axis_0)

mean_axis_1 = tf.reduce_mean(t, axis=1)
```

```
print("Mean along axis 1:\n", mean_axis_1)

mean_axis_2 = tf.reduce_mean(t, axis=2)
print("Mean along axis 2:\n", mean_axis_2)
```

```
Mean of all elements in tensor t: tf.Tensor(10, shape=(), dtype=int32)
Mean along axis 0:
 tf.Tensor(
[[ 9 12  9]
 [11 16  8]
 [11 10 10]
 [ 9  7  8]
 [12 10 11]
 [ 7 13 12]
 [ 6  7 13]], shape=(7, 3), dtype=int32)
Mean along axis 1:
 tf.Tensor(
[[ 8 11 11]
 [ 8 12 11]
 [ 9 11 11]
 [ 9  9  9]
 [13 11  9]], shape=(5, 3), dtype=int32)
Mean along axis 2:
 tf.Tensor(
[[13 10  7  6 14 10  9]
 [14 11 12 10  7 12  6]
 [ 8 14 13 13  7  8  9]
 [ 6 11 12  2 17  7  9]
 [10 11  9 10 12 15  9]], shape=(5, 7), dtype=int32)
```

Part 2

In this part, we will use TensorFlow and TensorBoard to train a neural network on the Fashion MNIST dataset.

steps:

- Load TensorBoard Extension and Import Libraries
- Load and Preprocess Fashion MNIST Dataset
- Define a Neural Network Model
- Create and Compile the Model
- Generate Logs for TensorBoard
    - Create a directory for TensorBoard logs with a timestamp, and define a TensorBoard callback to log information during training, including histograms of layer activations.
- Fit the Model

Load the TensorBoard notebook extension

In [102…

```
%load_ext tensorboard
```

Import datetime

In [104…
```
import datetime
```

load the fashion_mnist dataset and split it into train and test

In [106…
```
import tensorflow as tf

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.fashion_mnist.load_data()


print("Training data shape:", x_train.shape)
print("Testing data shape:", x_test.shape)
```

```
Training data shape: (60000, 28, 28)
Testing data shape: (10000, 28, 28)
```

Normalize the data for x

In [111…
```
x_train, x_test = x_train / 255.0, x_test / 255.0


print("Min value in x_train:", x_train.min())
print("Max value in x_train:", x_train.max())
print("Min value in x_test:", x_test.min())
print("Max value in x_test:", x_test.max())
```

```
Min value in x_train: 0.0
Max value in x_train: 0.00392156862745098
Min value in x_test: 0.0
Max value in x_test: 0.00392156862745098
```

Write a function that creates a simple Keras model for classifying the images into 10 classes.
def create_model():

- Use the sequential model
- flatten the data
- add a hidden layer with 64 neurons with activation function relu
- add the dropout technique to reduce model overfitting
- tf.keras.layers.Dropout(0.2),
- add an output layer with 10 neurons and activation function SoftMax

In [114…
```
def create_model():
    return tf.keras.models.Sequential([

  ])
```

When training with Keras's Model.fit(), adding the tf.keras.callbacks.TensorBoard callback ensures that logs are created and stored. Additionally, enable histogram computation every epoch with histogram_freq=1 (this is off by default)

Place the logs in a timestamped subdirectory to allow easy selection of different training runs.

Open a second notebook and execute the following code in that notebook.

%load_ext tensorboard

%tensorboard --logdir logs/fit

Create a model and compile using the optimizer adam, the loss "sparse_categorical_crossentropy", and metrics = accuracy.

In [120…

```
# Enter your code here:
import tensorflow as tf
import datetime


def create_model():
    return tf.keras.models.Sequential([

        tf.keras.layers.Flatten(input_shape=(28, 28)),


        tf.keras.layers.Dense(64, activation='relu'),


        tf.keras.layers.Dropout(0.2),


        tf.keras.layers.Dense(10, activation='softmax')
    ])


model = create_model()


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


# Generate the Logs for Tensorboard
log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_fr

# fit the model
history = model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test), c
```

```
Epoch 1/5
C:\Users\William\anaconda3\Lib\site-packages\keras\src\layers\reshaping\flatten.py:3
7: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When u
sing Sequential models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(**kwargs)
```

```
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - accuracy: 0.4624 - loss: 1.6899 - val_a
ccuracy: 0.7195 - val_loss: 0.8482
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2s 967us/step - accuracy: 0.7099 - loss: 0.8303 - val
_accuracy: 0.7550 - val_loss: 0.6847
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2s 979us/step - accuracy: 0.7443 - loss: 0.6993 - val
_accuracy: 0.7691 - val_loss: 0.6290
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2s 909us/step - accuracy: 0.7627 - loss: 0.6467 - val
_accuracy: 0.7826 - val_loss: 0.5922
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2s 928us/step - accuracy: 0.7770 - loss: 0.6088 - val
_accuracy: 0.7917 - val_loss: 0.5636
```

Interview Questions

How does Deep Learning differ from Machine Learning?

Deep learning and machine learning are both key areas within artificial intelligence (AI), but they differ in how they handle data and solve problems. Machine learning (ML) is a broader concept that allows systems to learn from data without being explicitly programmed. ML works well with structured data (like spreadsheets or tabular data) and uses algorithms such as linear regression or decision trees to make predictions. These models are typically simpler and don't require a lot of layers or complexity. Deep learning (DL), on the other hand, is a specialized subset of ML that uses neural networks with multiple layers. These deep networks are particularly effective at working with unstructured data, like images, text, or audio, and are capable of learning complex patterns directly from the raw data. In short, deep learning excels at tasks where traditional machine learning methods might struggle, especially when handling large and unstructured datasets.

What is a Computational Graph?

A computational graph is a framework used to represent mathematical operations in deep learning models. Think of it as a directed graph where each node represents an operation (such as addition or multiplication), and the edges represent how data flows between operations. Deep learning libraries like TensorFlow or PyTorch use these graphs to structure the computations for model training and inference. By using computational graphs, these frameworks can efficiently compute gradients during backpropagation and optimize the performance of the model. This helps in speeding up the training process and ensuring that the model learns effectively.

Why do we use the activation function?

Activation functions are crucial for making neural networks more powerful and flexible. Without them, neural networks would just be a series of linear operations, which limits their ability to solve complex tasks. These functions add non-linearity to the model, allowing it to learn and represent more complex relationships in the data. Common activation functions like ReLU, sigmoid, and softmax enable the network to perform well on tasks like image classification or language processing. For example, ReLU helps avoid issues like vanishing gradients and makes the network train faster, while softmax is used to convert the output into probabilities in multi-class classification problems. In essence, activation functions make neural networks capable of handling the complexity and variety of real-world data.

In [ ]: