DESIGN OF MACHINE ELEMENTS PROJECT

Group members:

Siddhartha Prasad -210030033

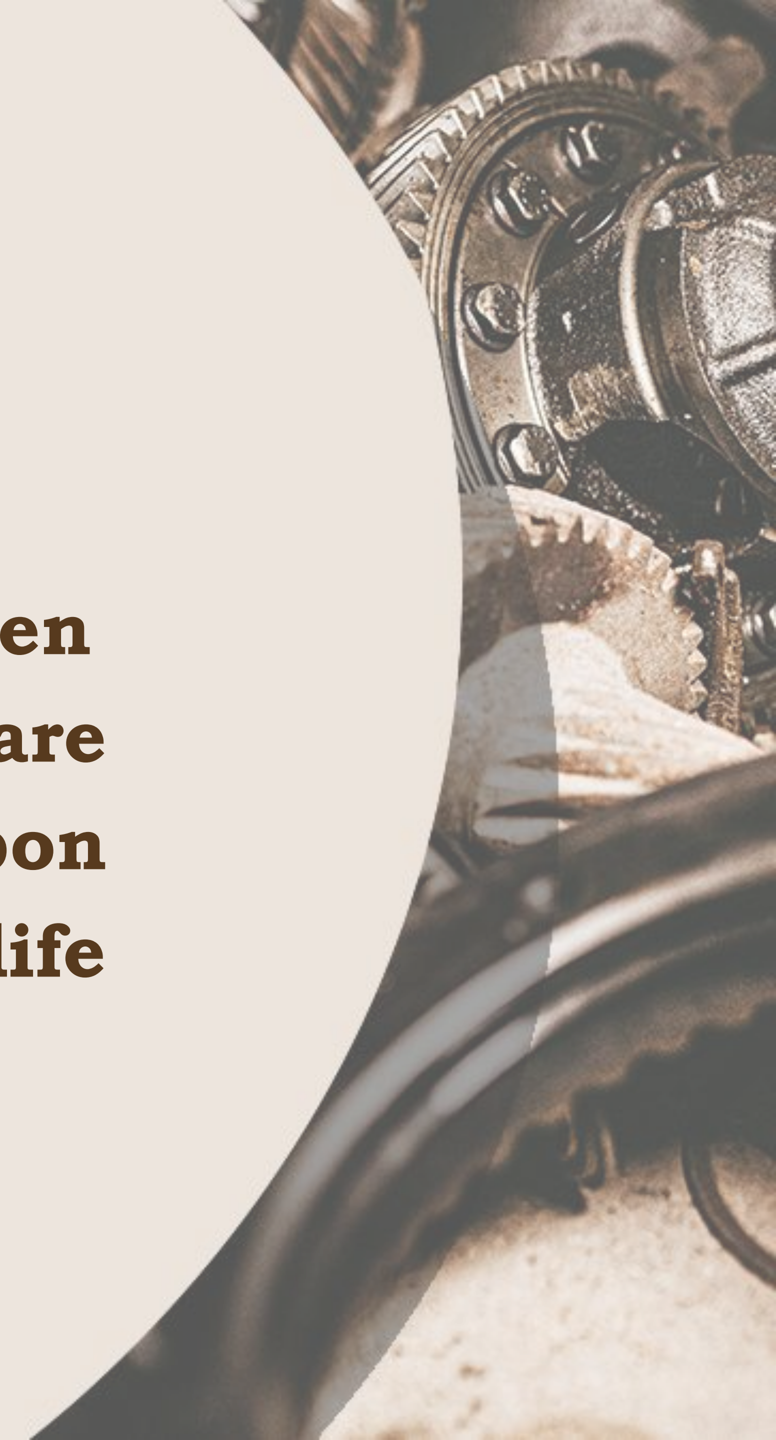
Swapnesh Sinha - 210030036

Thota Maniram - 210030041

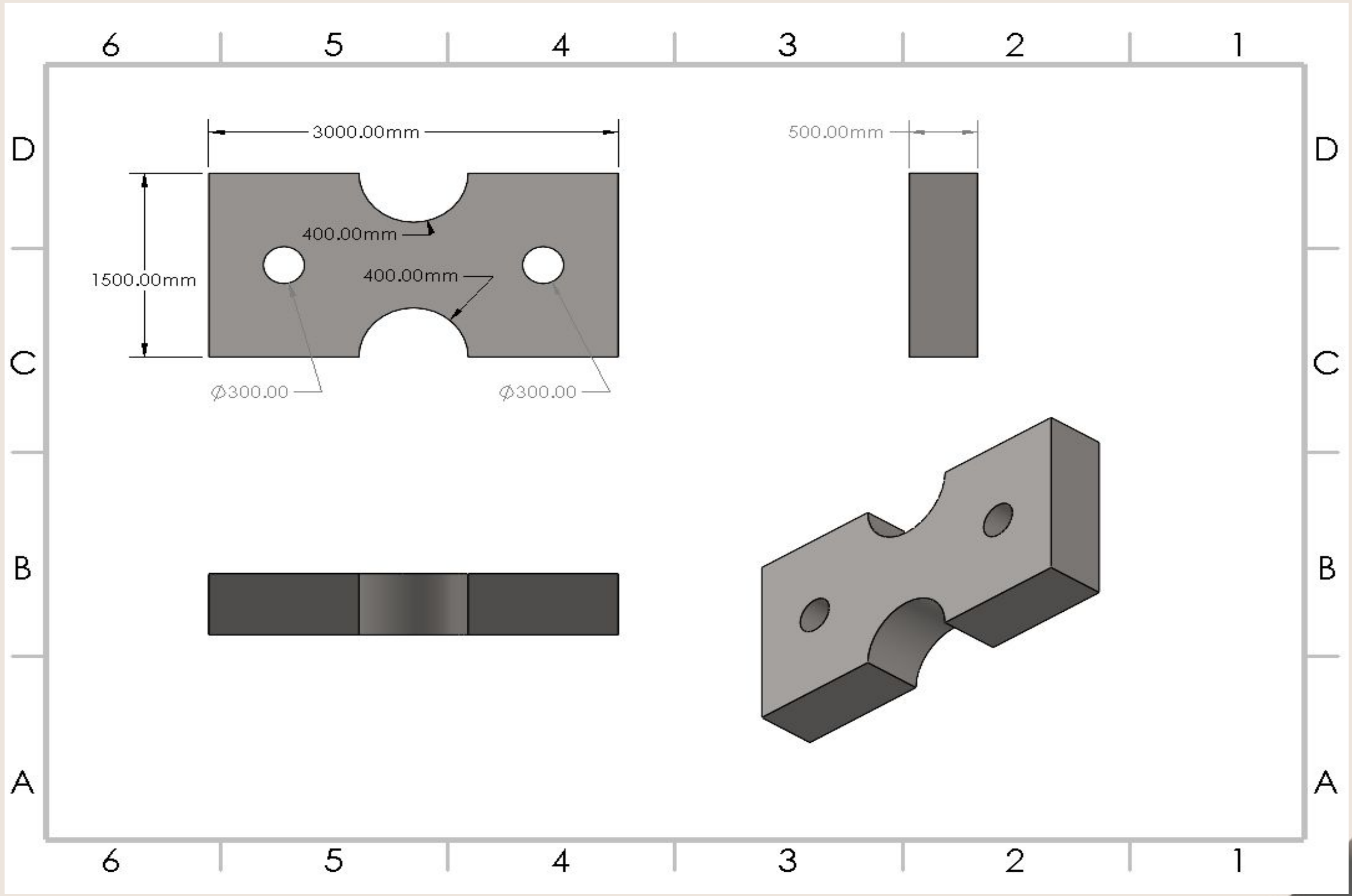
Pranam Chandra Shibaroor - 200030043

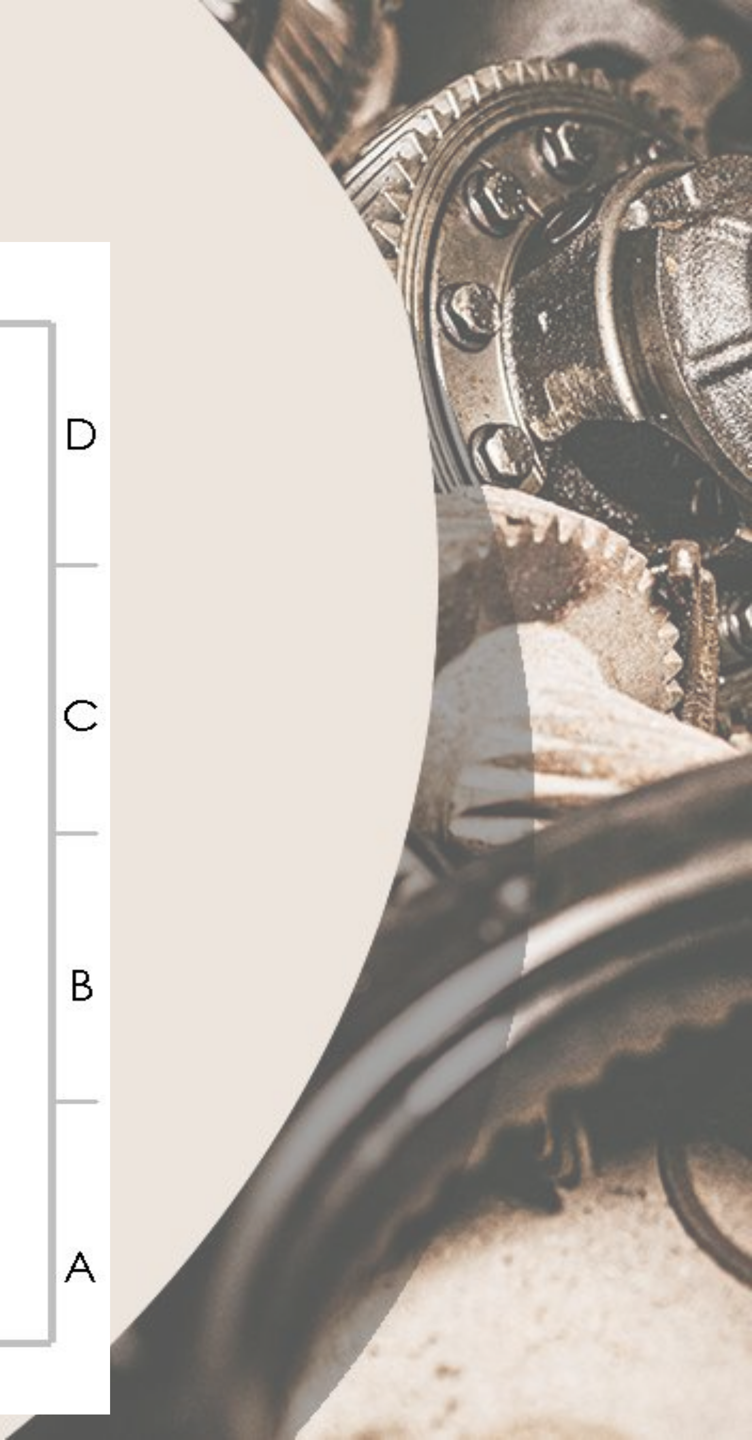
AIM OF THE PROJECT

- With the structure of our choice, been applied with a range of loads, we are Analyzing factor of safety and damage upon its life as well as predicting the fatigue life with the help of deep learning.

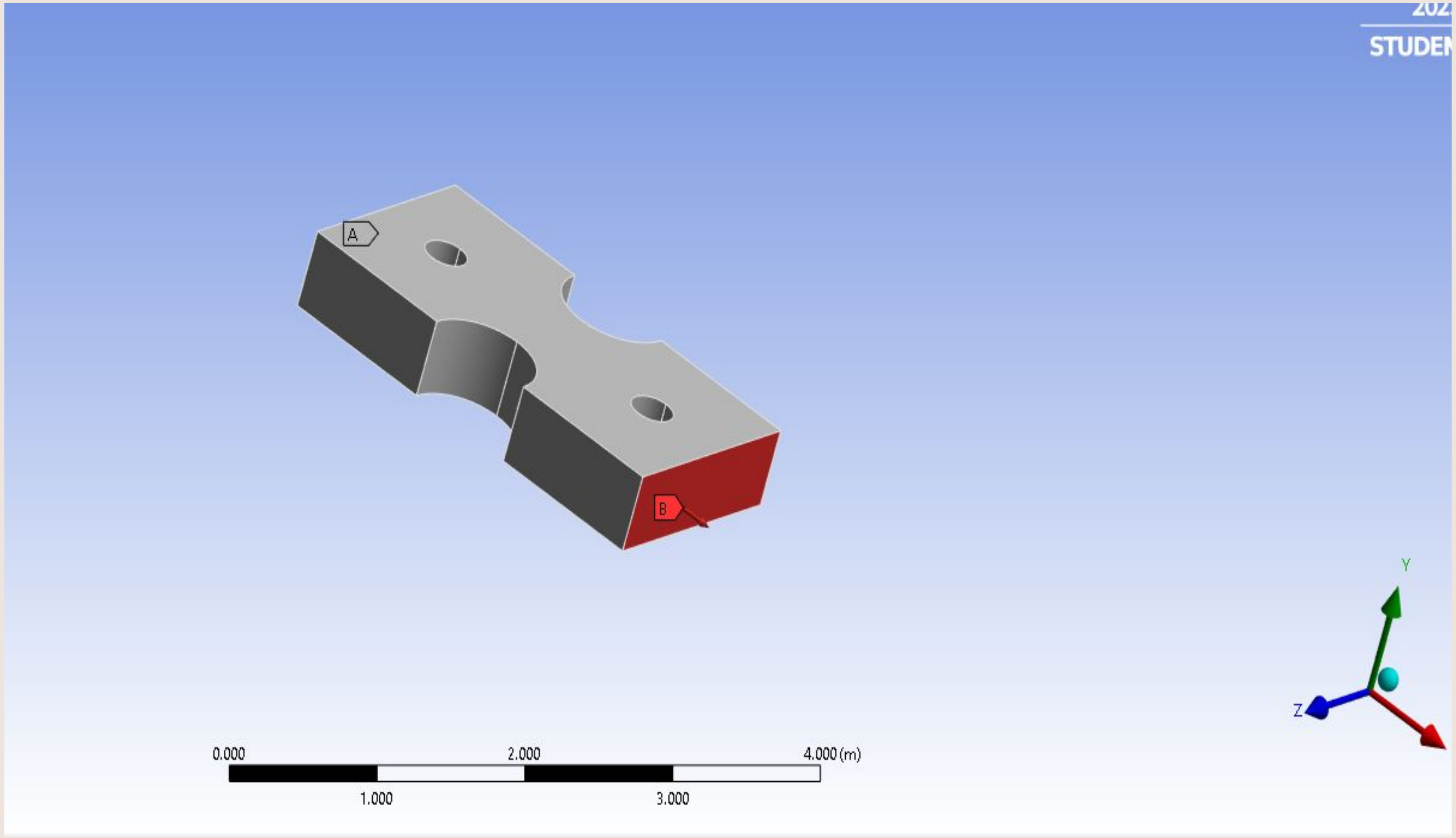


WHAT IS THE STRUCTURE?



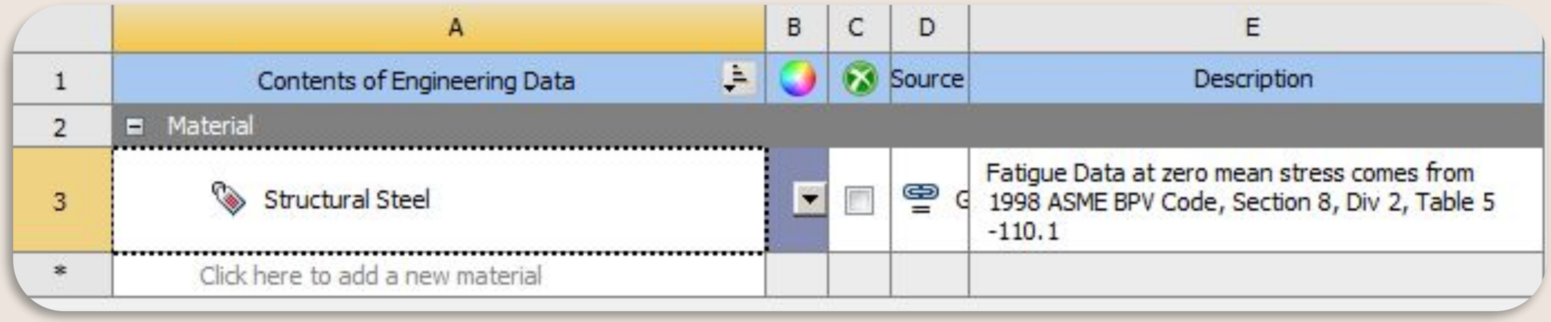


WHAT IS THE STRUCTURE?



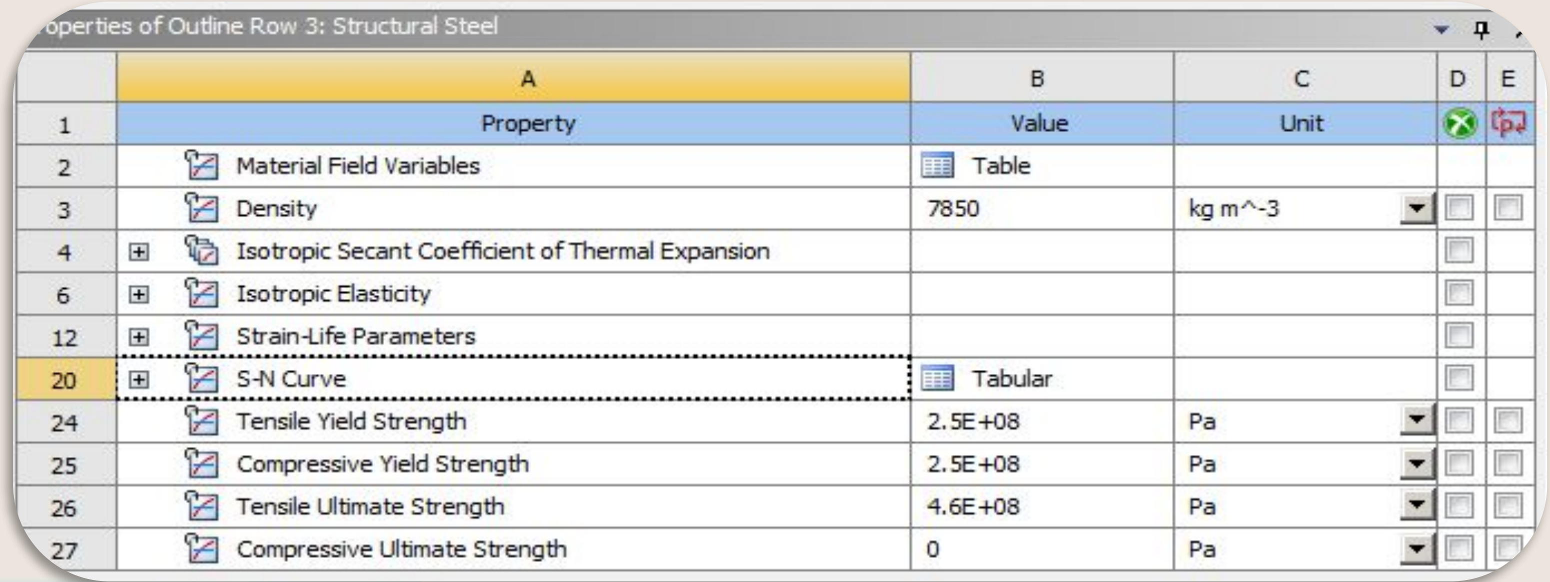
WHAT IS THE STRUCTURE?

- Material – MILD STEEL



	A	B	C	D	E
1	Contents of Engineering Data			Source	Description
2	⊟ Material				
3	Structural Steel		☐	G	Fatigue Data at zero mean stress comes from 1998 ASME BPV Code, Section 8, Div 2, Table 5 -110.1
*	Click here to add a new material				

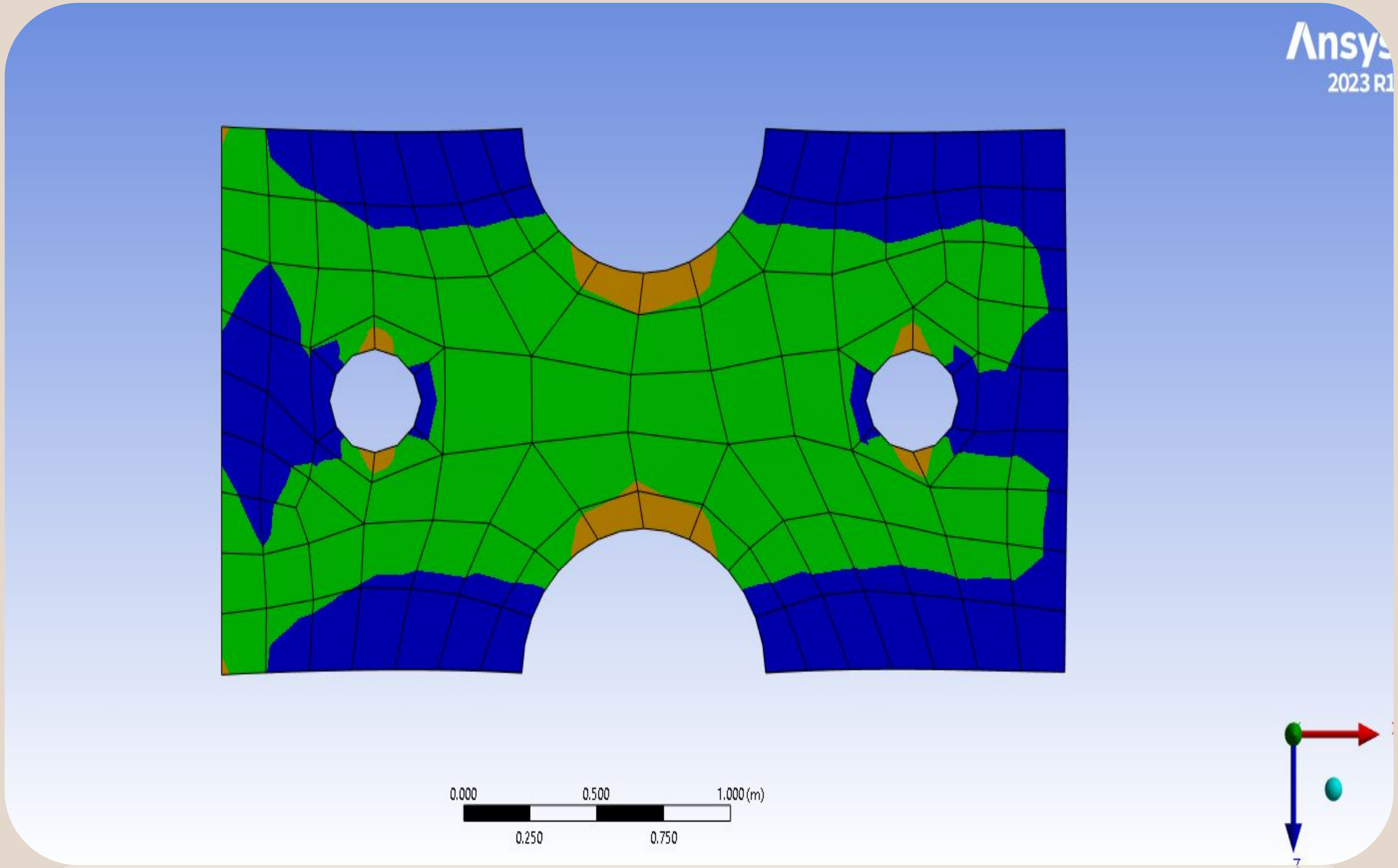
- Material PROPERTIES:

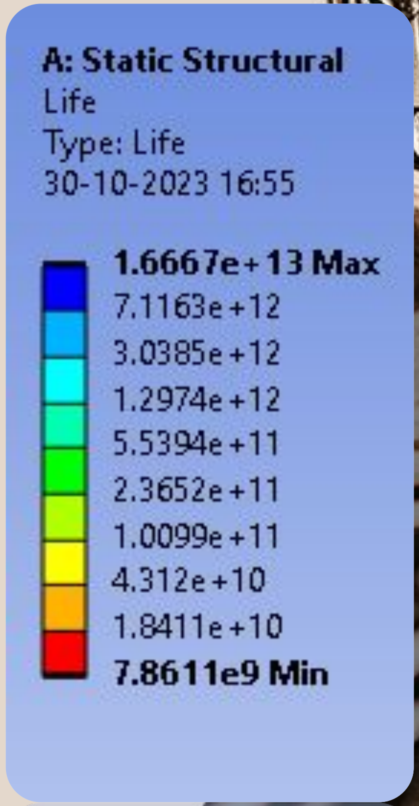


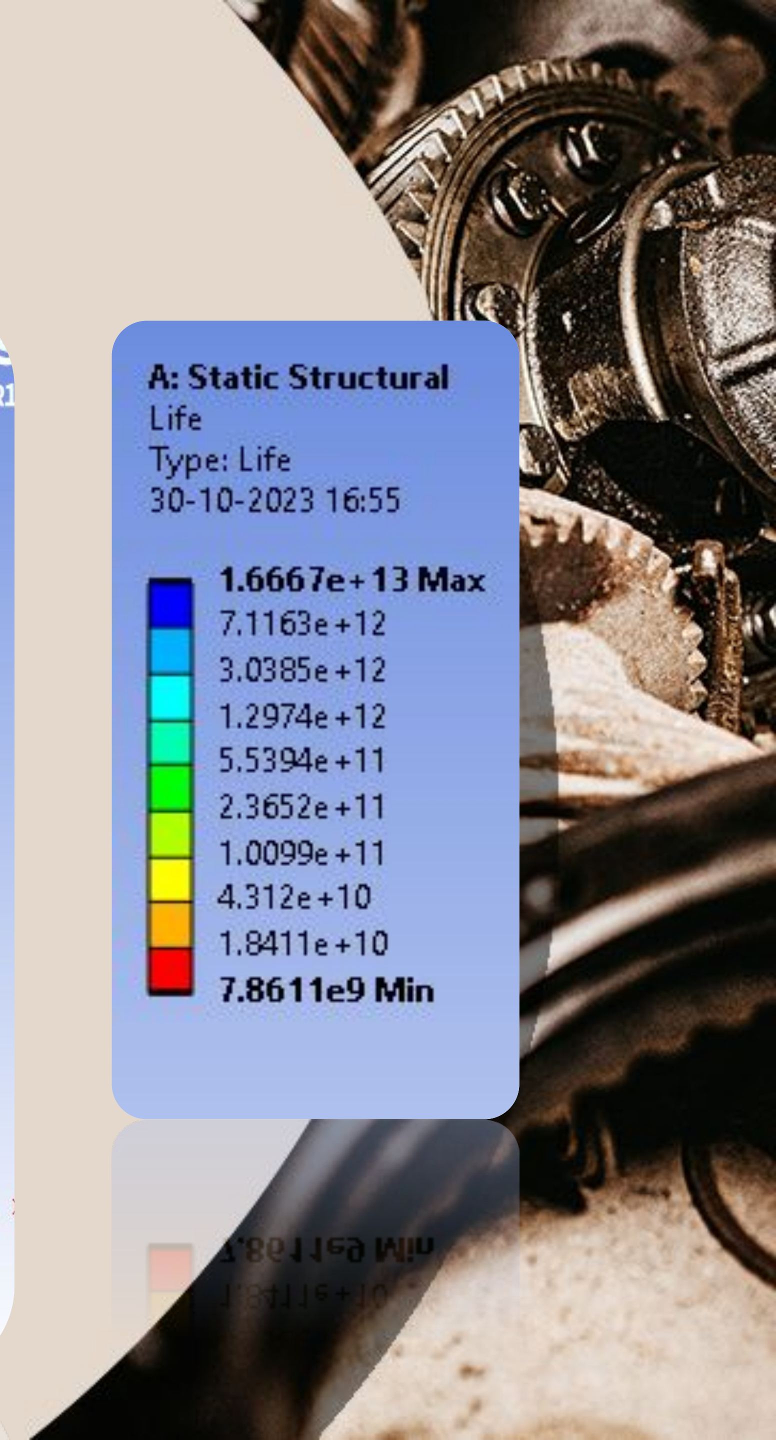
operties of Outline Row 3: Structural Steel

	A	B	C	D	E
1	Property	Value	Unit		
2	Material Field Variables	Table			
3	Density	7850	kg m^-3	☐	☐
4	⊞ Isotropic Secant Coefficient of Thermal Expansion			☐	
6	⊞ Isotropic Elasticity			☐	
12	⊞ Strain-Life Parameters			☐	
20	⊞ S-N Curve	Tabular		☐	
24	Tensile Yield Strength	2.5E+08	Pa	☐	☐
25	Compressive Yield Strength	2.5E+08	Pa	☐	☐
26	Tensile Ultimate Strength	4.6E+08	Pa	☐	☐
27	Compressive Ultimate Strength	0	Pa	☐	☐

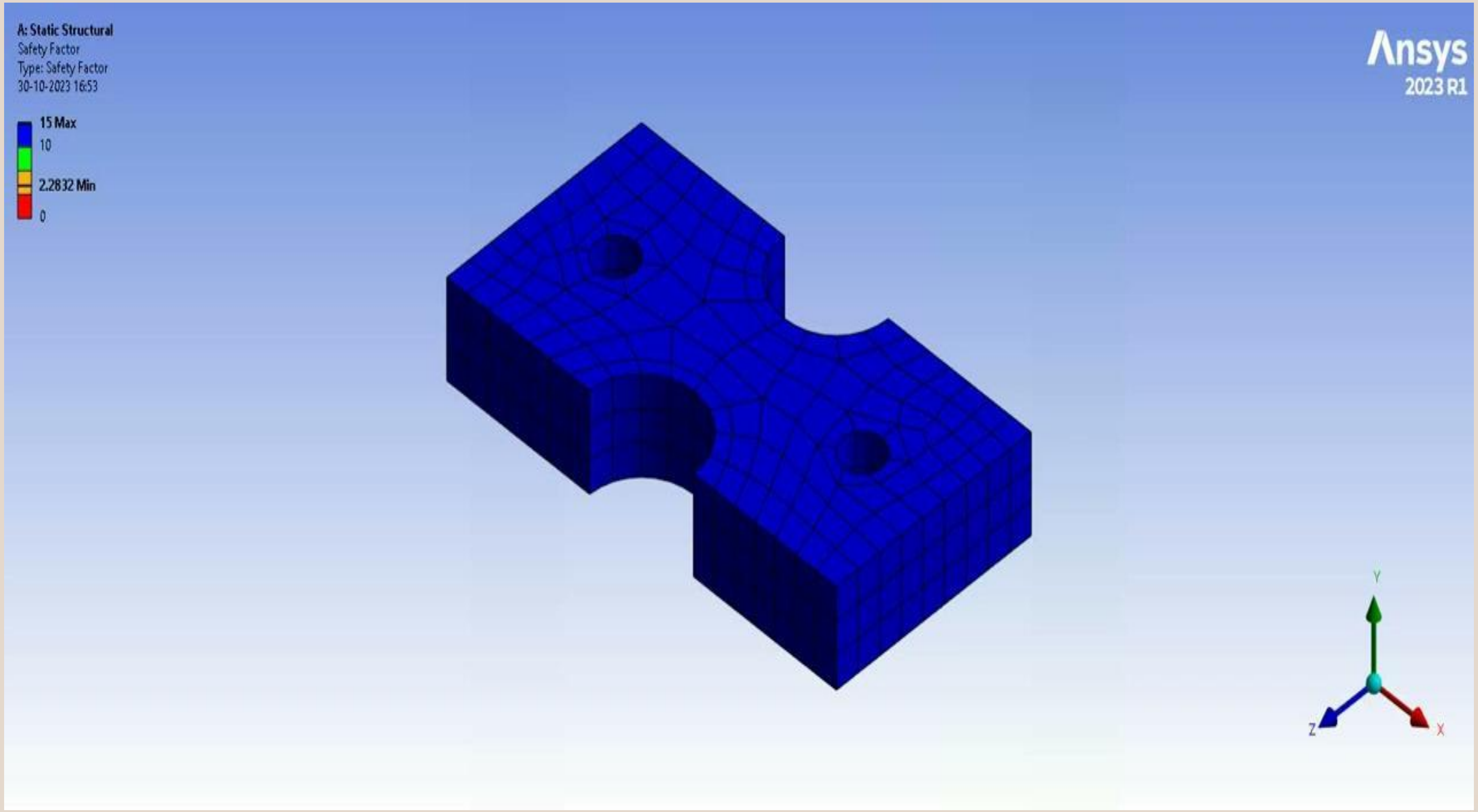
FACTOR OF SAFETY ANALYSIS

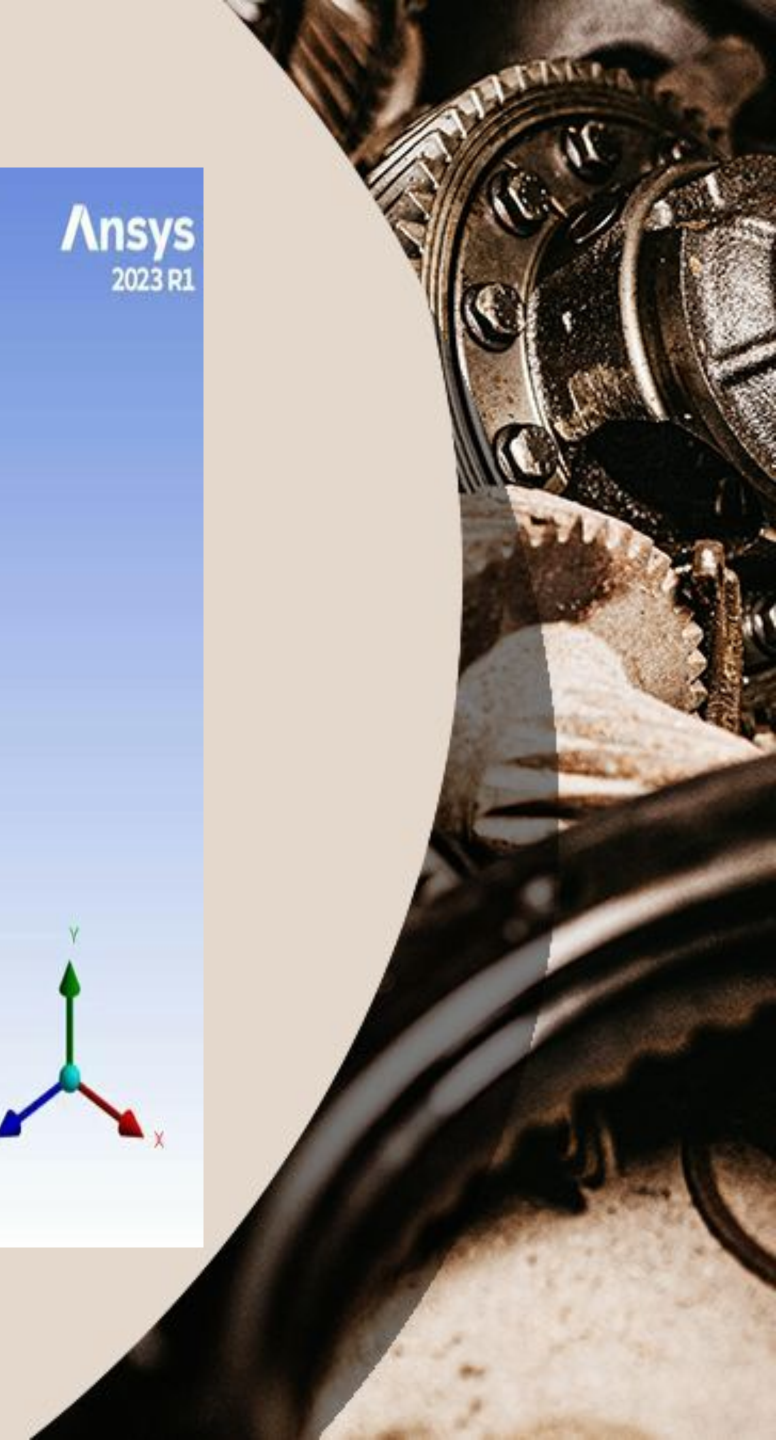




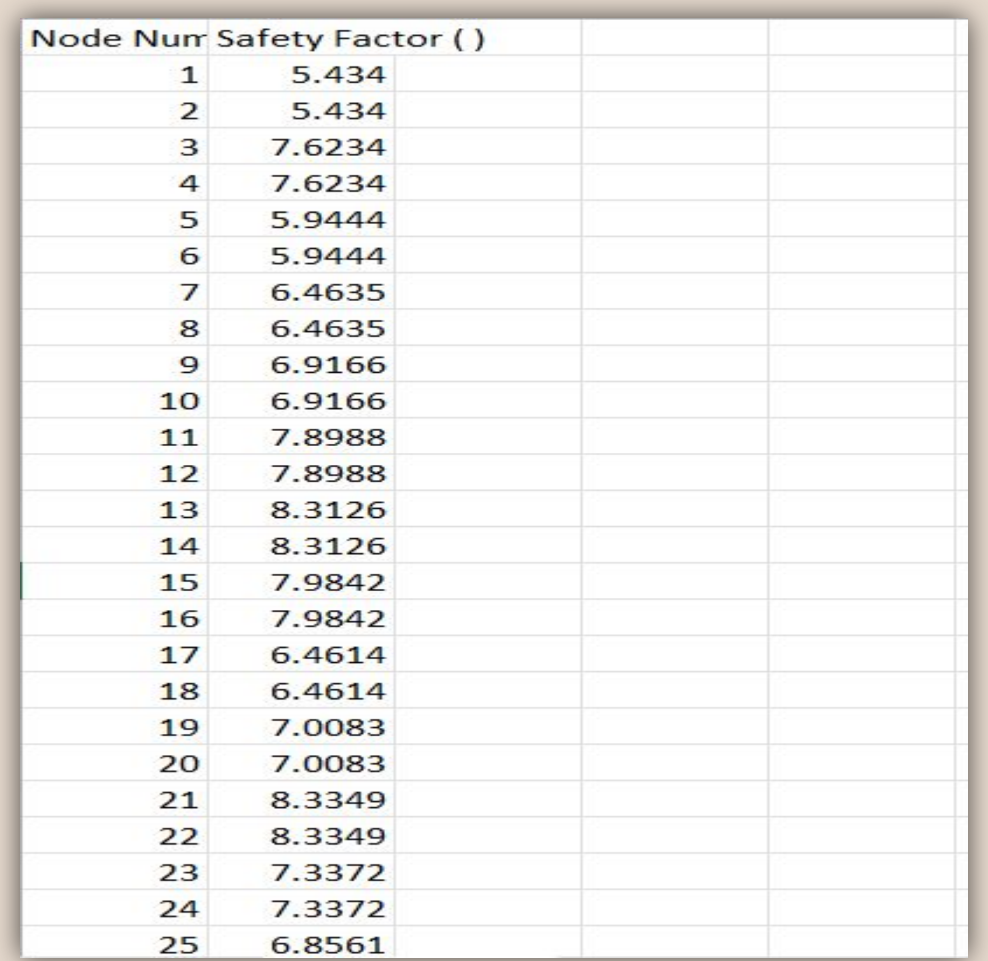


FACTOR OF SAFETY ANALYSIS



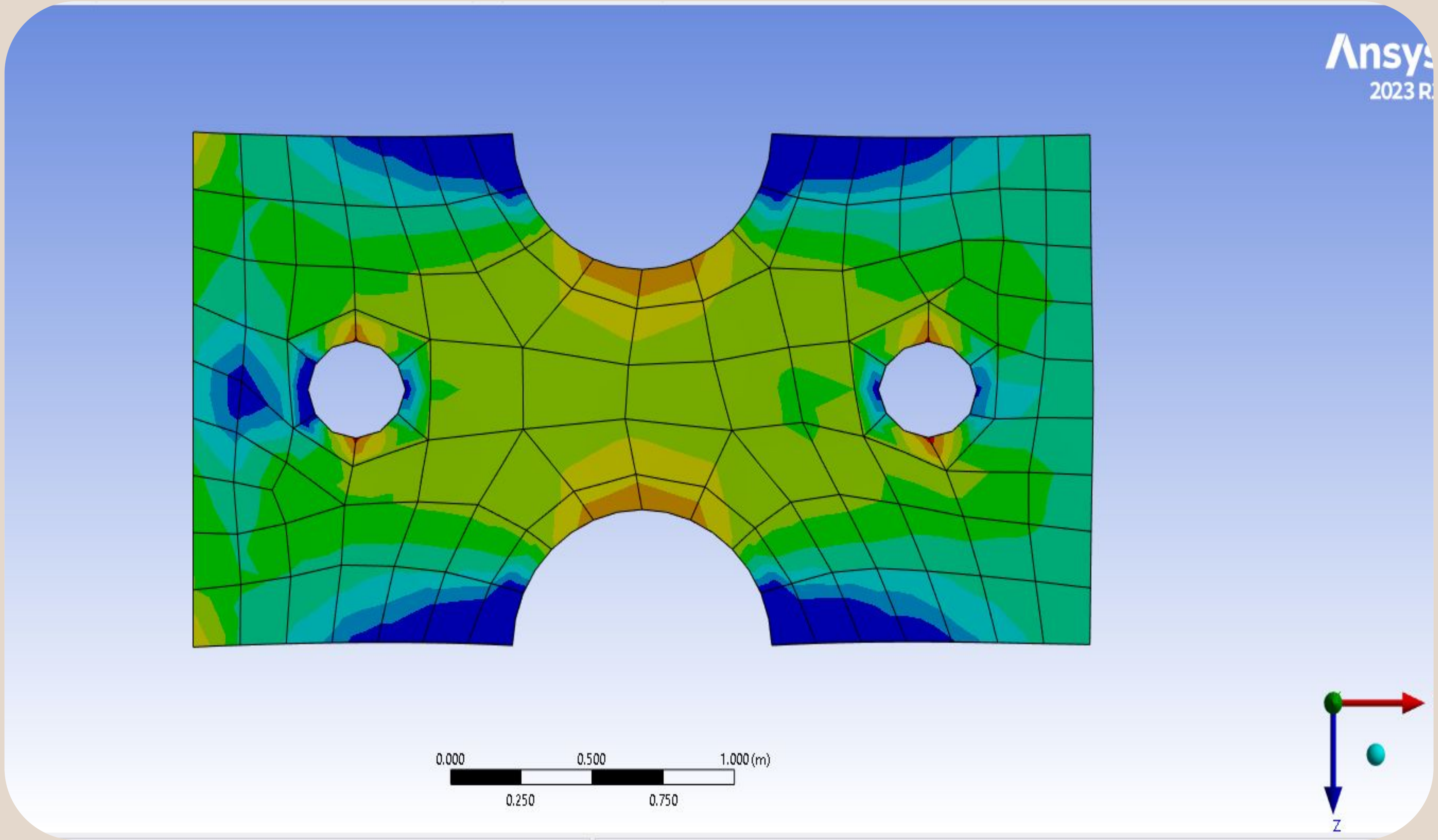


FACTOR OF SAFETY ANALYSIS



Node Num	Safety Factor ( )
1	5.434
2	5.434
3	7.6234
4	7.6234
5	5.9444
6	5.9444
7	6.4635
8	6.4635
9	6.9166
10	6.9166
11	7.8988
12	7.8988
13	8.3126
14	8.3126
15	7.9842
16	7.9842
17	6.4614
18	6.4614
19	7.0083
20	7.0083
21	8.3349
22	8.3349
23	7.3372
24	7.3372
25	6.8561

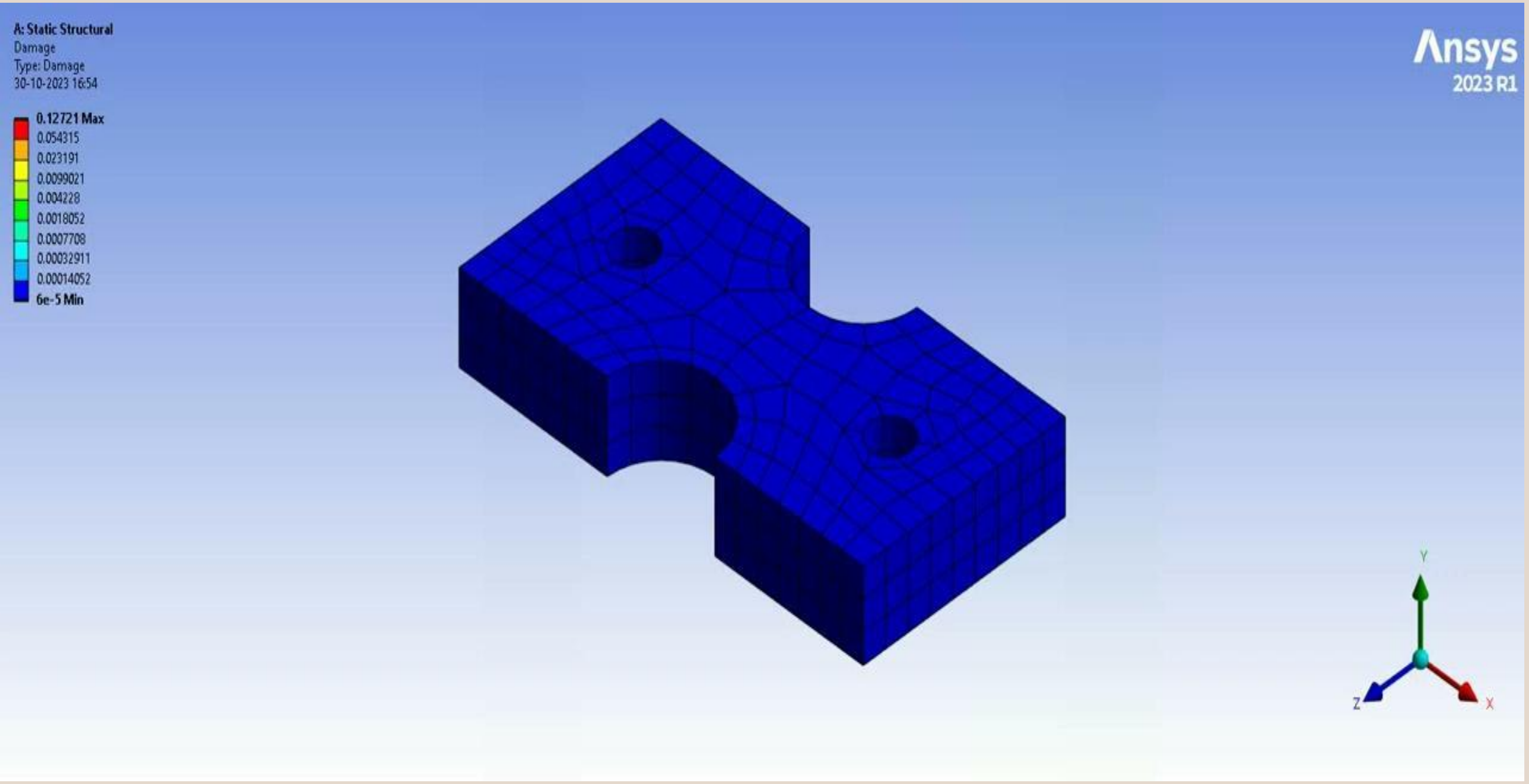
DAMAGE UPON THE STRUCTURE

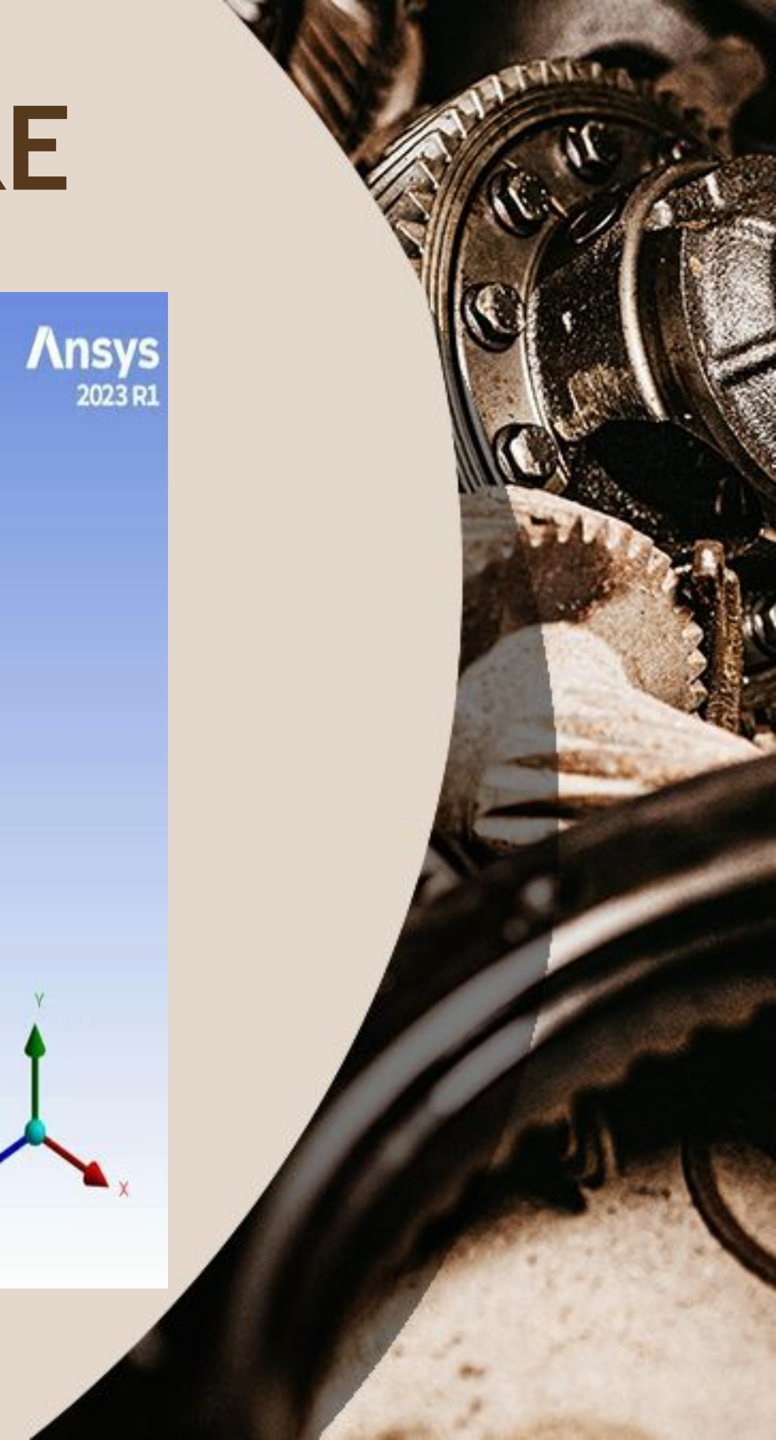


A: Static Structural
Damage
Type: Damage
30-10-2023 16:56

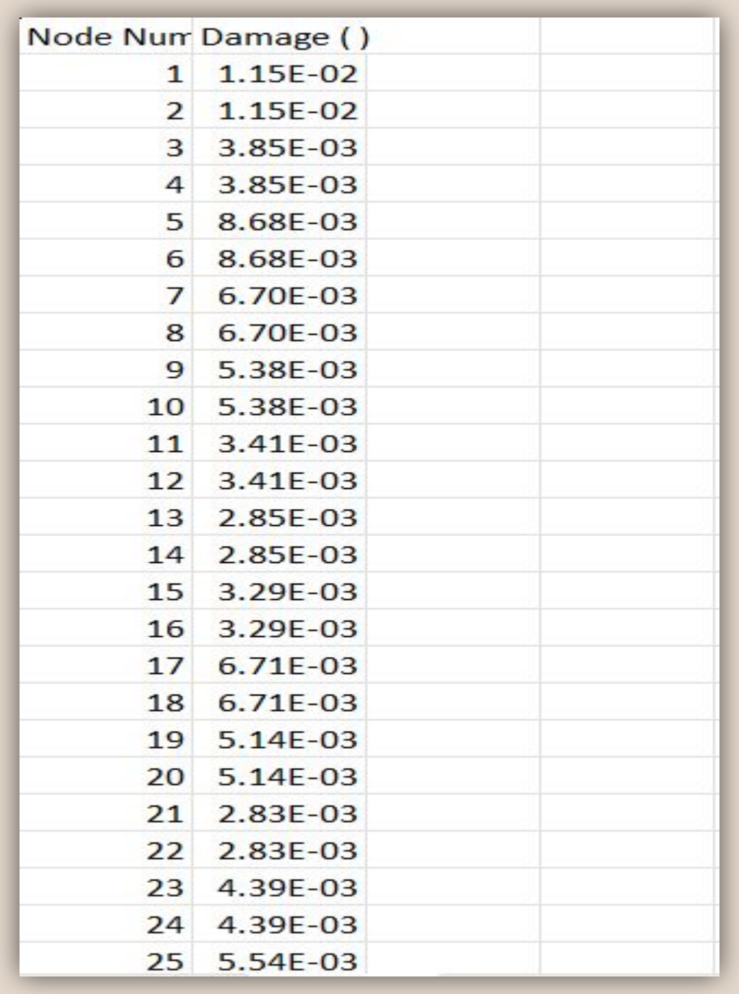
🟥	**0.12721 Max**
🟧	0.054315
🟨	0.023191
🟨	0.0099021
🟩	0.004228
🟩	0.0018052
🟦	0.0007708
🟦	0.00032911
🟦	0.00014052
🟦	**6e-5 Min**

DAMAGE UPON THE STRUCTURE



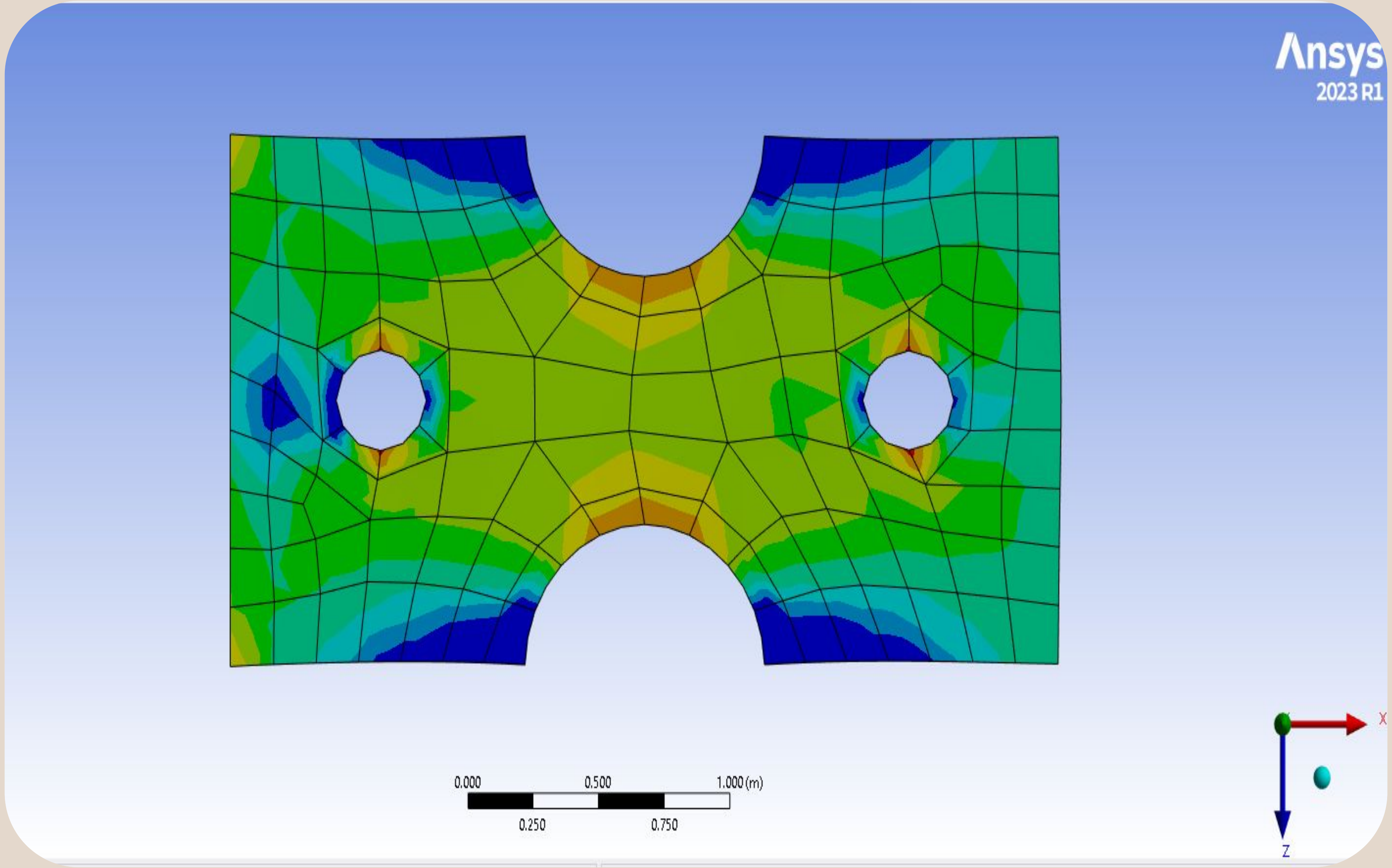


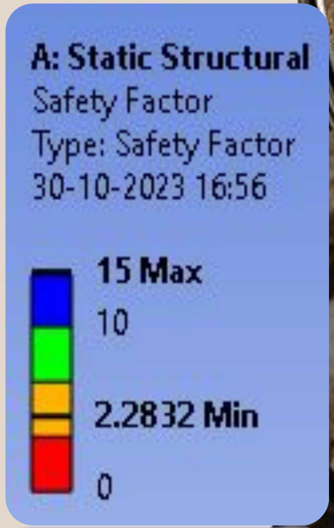
DAMAGE UPON THE STRUCTURE



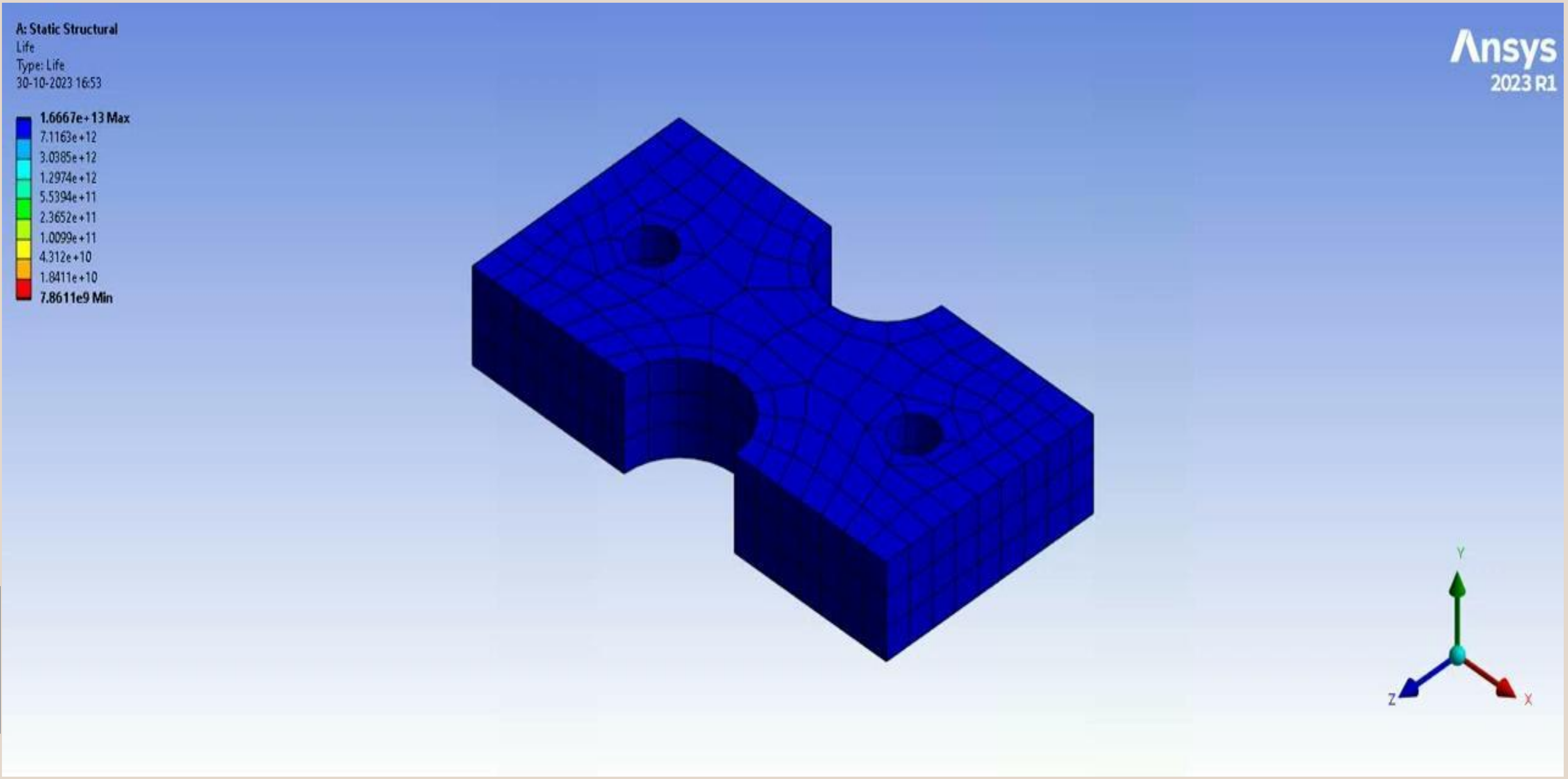
Node Num	Damage ( )		
1	1.15E-02		
2	1.15E-02		
3	3.85E-03		
4	3.85E-03		
5	8.68E-03		
6	8.68E-03		
7	6.70E-03		
8	6.70E-03		
9	5.38E-03		
10	5.38E-03		
11	3.41E-03		
12	3.41E-03		
13	2.85E-03		
14	2.85E-03		
15	3.29E-03		
16	3.29E-03		
17	6.71E-03		
18	6.71E-03		
19	5.14E-03		
20	5.14E-03		
21	2.83E-03		
22	2.83E-03		
23	4.39E-03		
24	4.39E-03		
25	5.54E-03		

FATIGUE LIFE

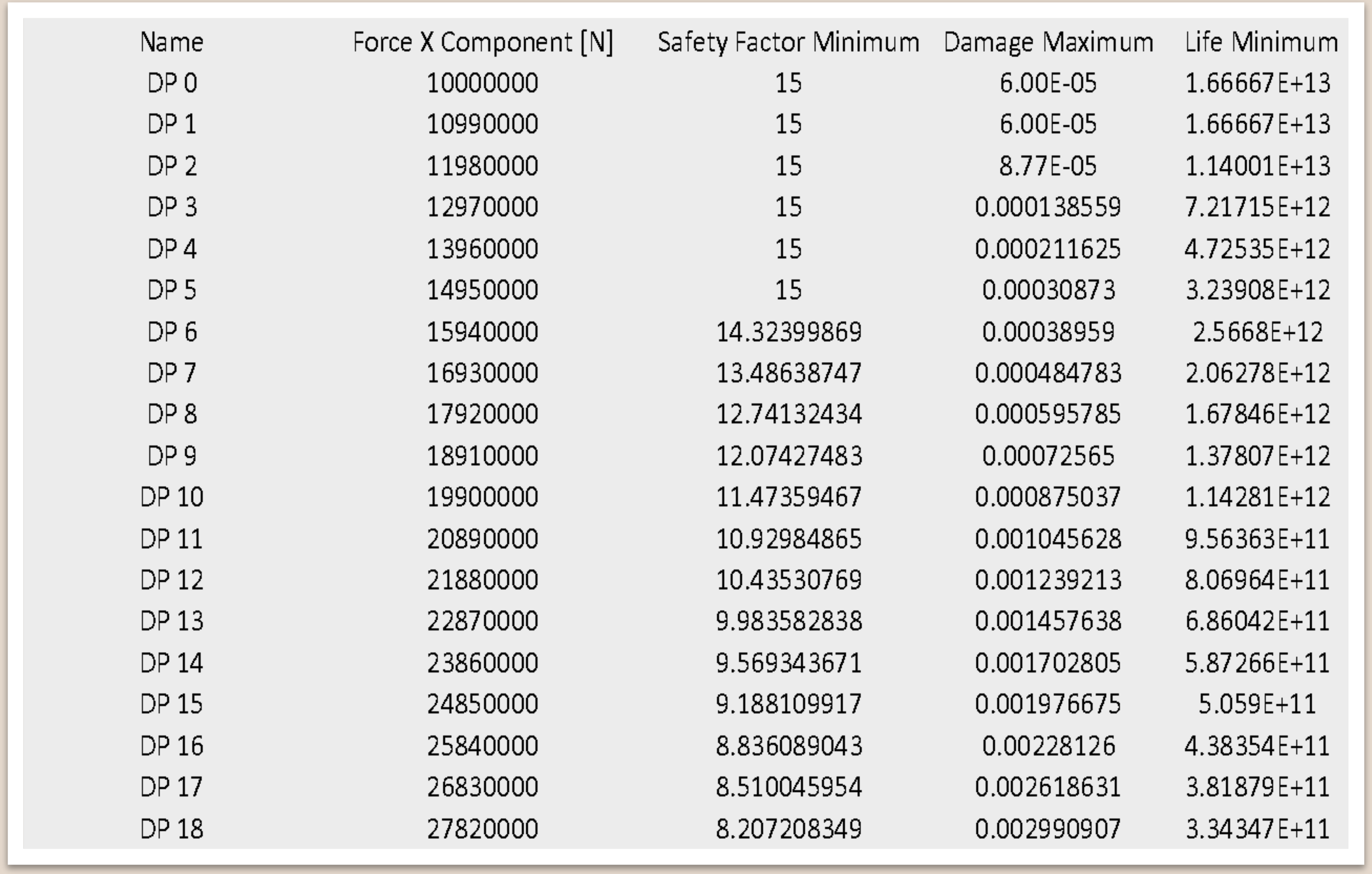




FATIGUE LIFE



FATIGUE LIFE

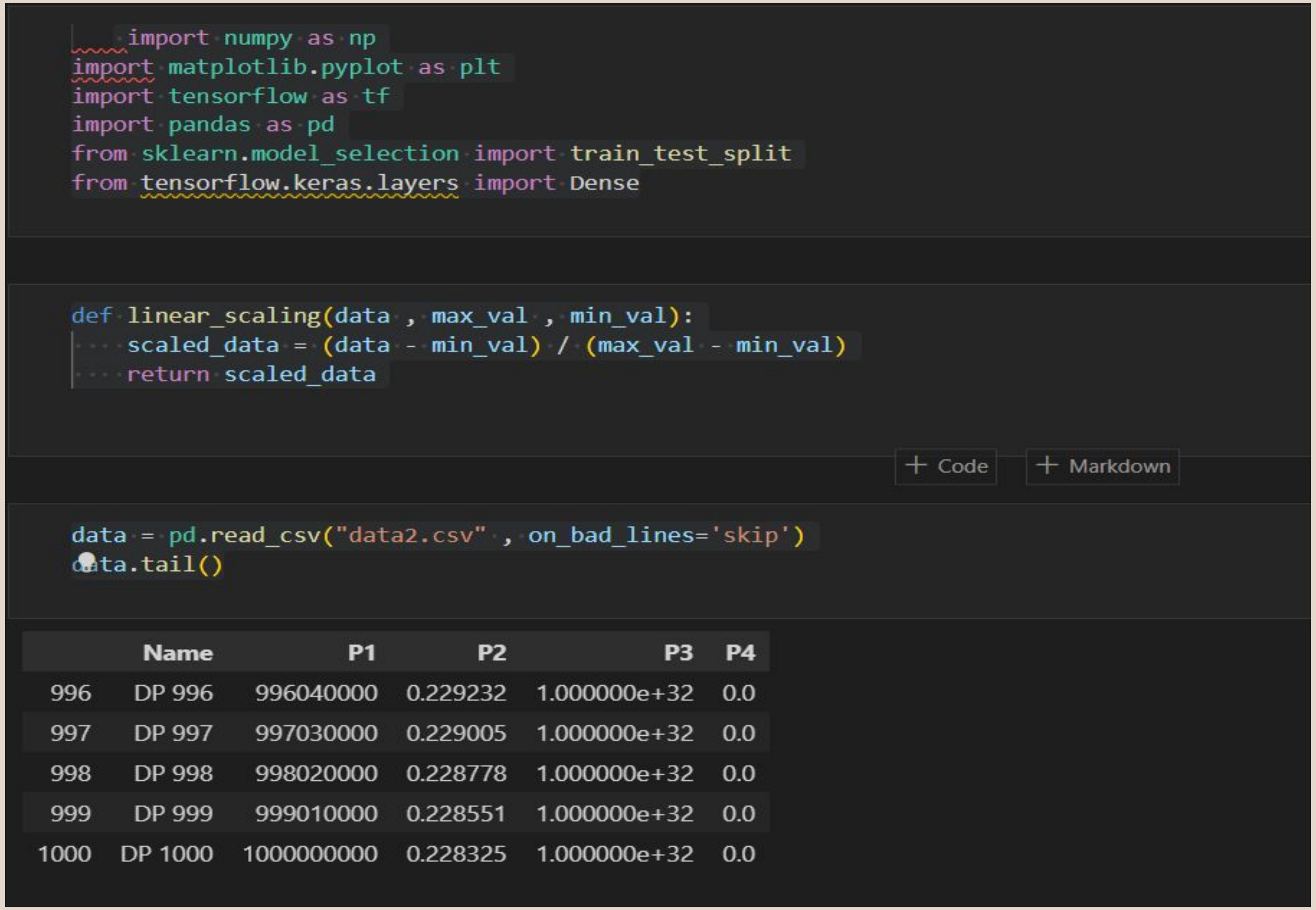


Name	Force X Component [N]	Safety Factor Minimum	Damage Maximum	Life Minimum
DP 0	10000000	15	6.00E-05	1.66667E+13
DP 1	10990000	15	6.00E-05	1.66667E+13
DP 2	11980000	15	8.77E-05	1.14001E+13
DP 3	12970000	15	0.000138559	7.21715E+12
DP 4	13960000	15	0.000211625	4.72535E+12
DP 5	14950000	15	0.00030873	3.23908E+12
DP 6	15940000	14.32399869	0.00038959	2.5668E+12
DP 7	16930000	13.48638747	0.000484783	2.06278E+12
DP 8	17920000	12.74132434	0.000595785	1.67846E+12
DP 9	18910000	12.07427483	0.00072565	1.37807E+12
DP 10	19900000	11.47359467	0.000875037	1.14281E+12
DP 11	20890000	10.92984865	0.001045628	9.56363E+11
DP 12	21880000	10.43530769	0.001239213	8.06964E+11
DP 13	22870000	9.983582838	0.001457638	6.86042E+11
DP 14	23860000	9.569343671	0.001702805	5.87266E+11
DP 15	24850000	9.188109917	0.001976675	5.059E+11
DP 16	25840000	8.836089043	0.00228126	4.38354E+11
DP 17	26830000	8.510045954	0.002618631	3.81879E+11
DP 18	27820000	8.207208349	0.002990907	3.34347E+11

FATIGUE LIFE PREDICTION USING DEEP LEARNING

Training code:
https://colab.research.google.com/drive/1UA4OvC56gpr016dVZQaJPqDG8BFQfKnl?usp=sharing

Deep Learning code

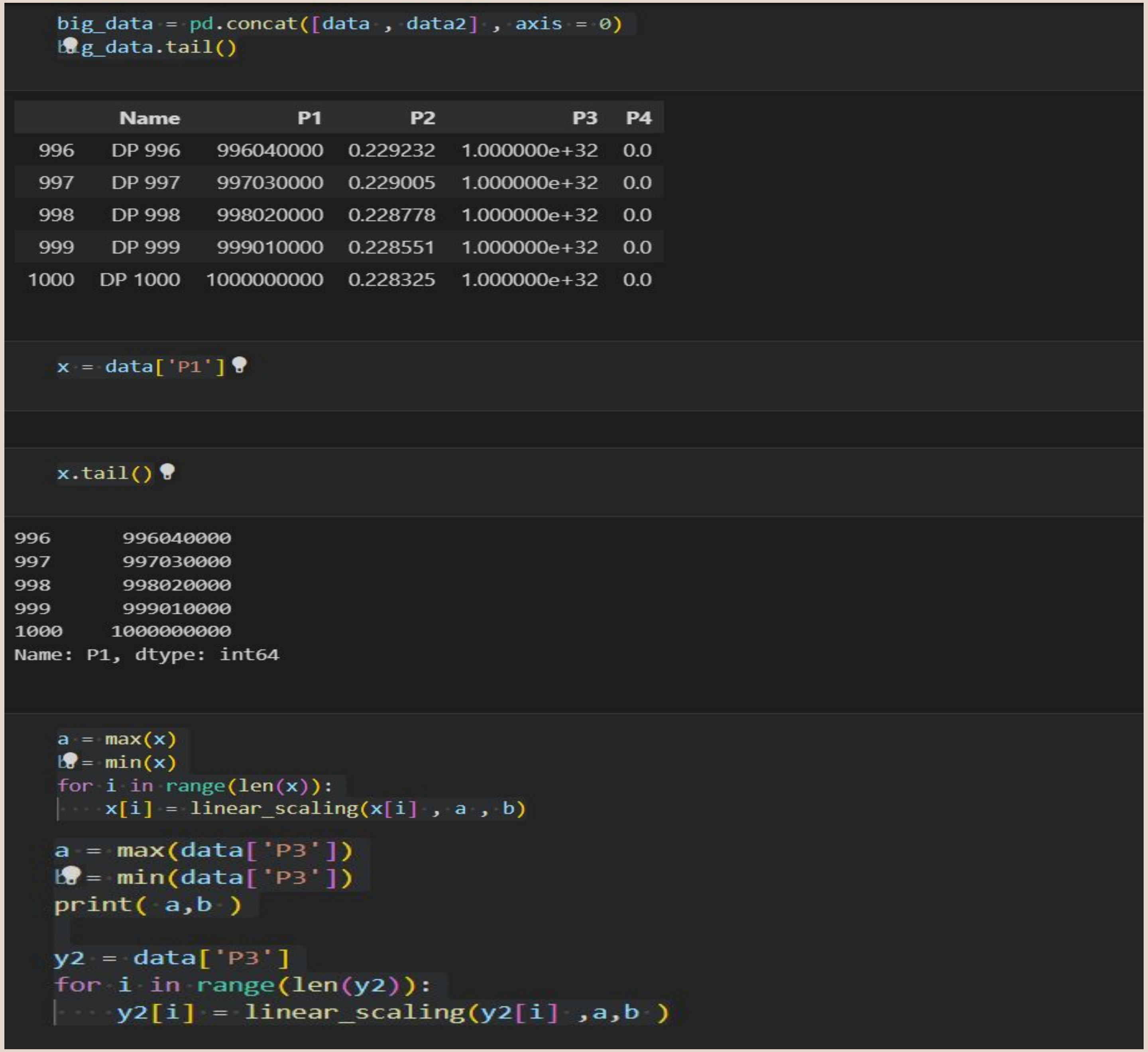


```
    import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import pandas as pd
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import Dense
```

```
def linear_scaling(data , max_val , min_val):
    scaled_data = (data - min_val) / (max_val - min_val)
    return scaled_data
```

```
data = pd.read_csv("data2.csv" , on_bad_lines='skip')
data.tail()
```

	Name	P1	P2	P3	P4
996	DP 996	996040000	0.229232	1.000000e+32	0.0
997	DP 997	997030000	0.229005	1.000000e+32	0.0
998	DP 998	998020000	0.228778	1.000000e+32	0.0
999	DP 999	999010000	0.228551	1.000000e+32	0.0
1000	DP 1000	1000000000	0.228325	1.000000e+32	0.0



```python
big_data = pd.concat([data , data2] , axis = 0)
big_data.tail()
```

	Name	P1	P2	P3	P4
996	DP 996	996040000	0.229232	1.000000e+32	0.0
997	DP 997	997030000	0.229005	1.000000e+32	0.0
998	DP 998	998020000	0.228778	1.000000e+32	0.0
999	DP 999	999010000	0.228551	1.000000e+32	0.0
1000	DP 1000	1000000000	0.228325	1.000000e+32	0.0

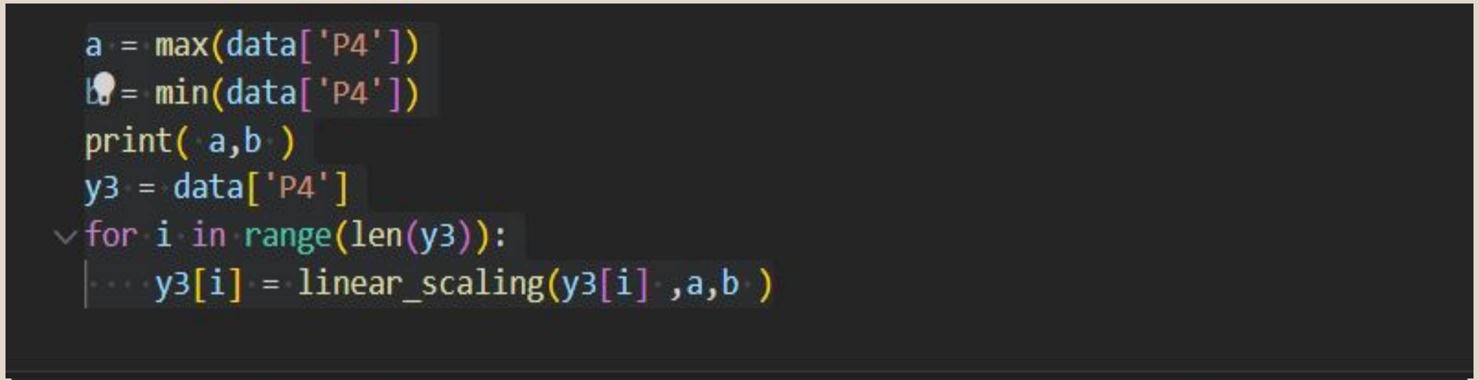
```python
x = data['P1']
```

```python
x.tail()
```

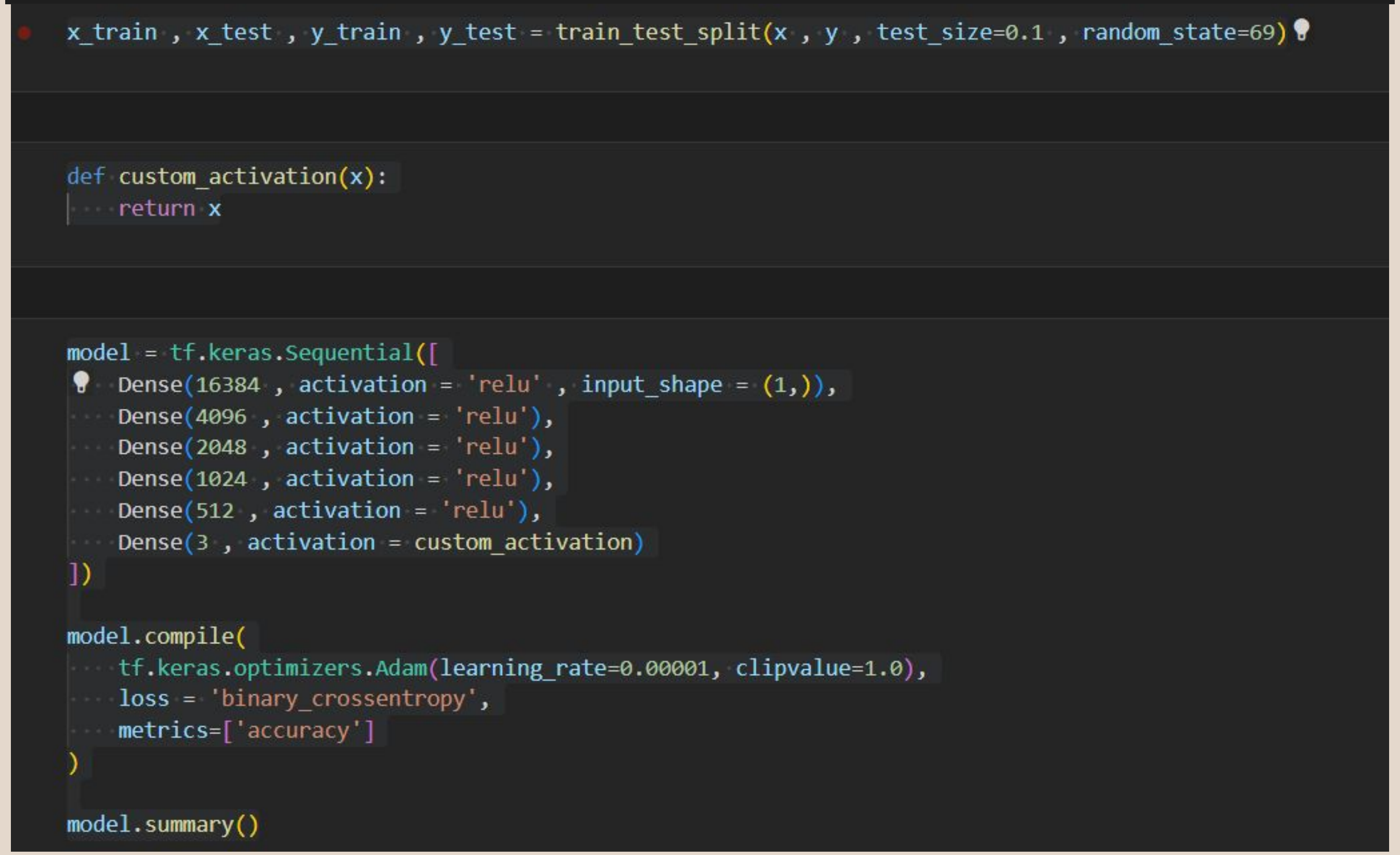
```
996      996040000
997      997030000
998      998020000
999      999010000
1000    1000000000
Name: P1, dtype: int64
```

```python
a = max(x)
b = min(x)
for i in range(len(x)):
    x[i] = linear_scaling(x[i] , a , b)
```

```python
a = max(data['P3'])
b = min(data['P3'])
print( a,b )

y2 = data['P3']
for i in range(len(y2)):
    y2[i] = linear_scaling(y2[i] ,a,b )
```



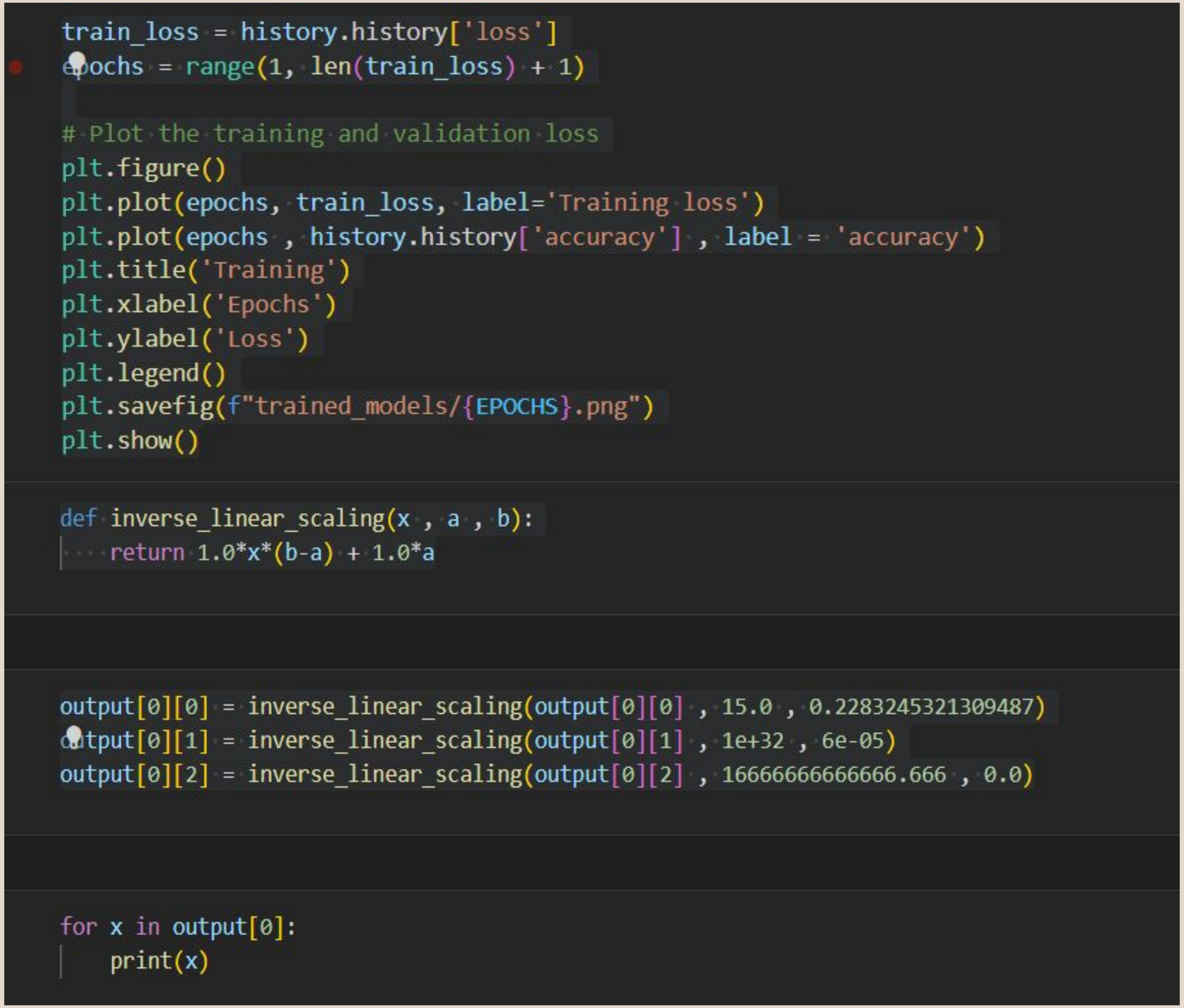
```python
a = max(data['P4'])
b = min(data['P4'])
print( a,b )
y3 = data['P4']
for i in range(len(y3)):
    y3[i] = linear_scaling(y3[i] ,a,b )
```



```python
x_train , x_test , y_train , y_test = train_test_split(x , y , test_size=0.1 , random_state=69)
```

```python
def custom_activation(x):
    return x
```

```python
model = tf.keras.Sequential([
    Dense(16384 , activation = 'relu' , input_shape = (1,)),
    Dense(4096 , activation = 'relu'),
    Dense(2048 , activation = 'relu'),
    Dense(1024 , activation = 'relu'),
    Dense(512 , activation = 'relu'),
    Dense(3 , activation = custom_activation)
])

model.compile(
    tf.keras.optimizers.Adam(learning_rate=0.00001, clipvalue=1.0),
    loss = 'binary_crossentropy',
    metrics=['accuracy']
)

model.summary()
```



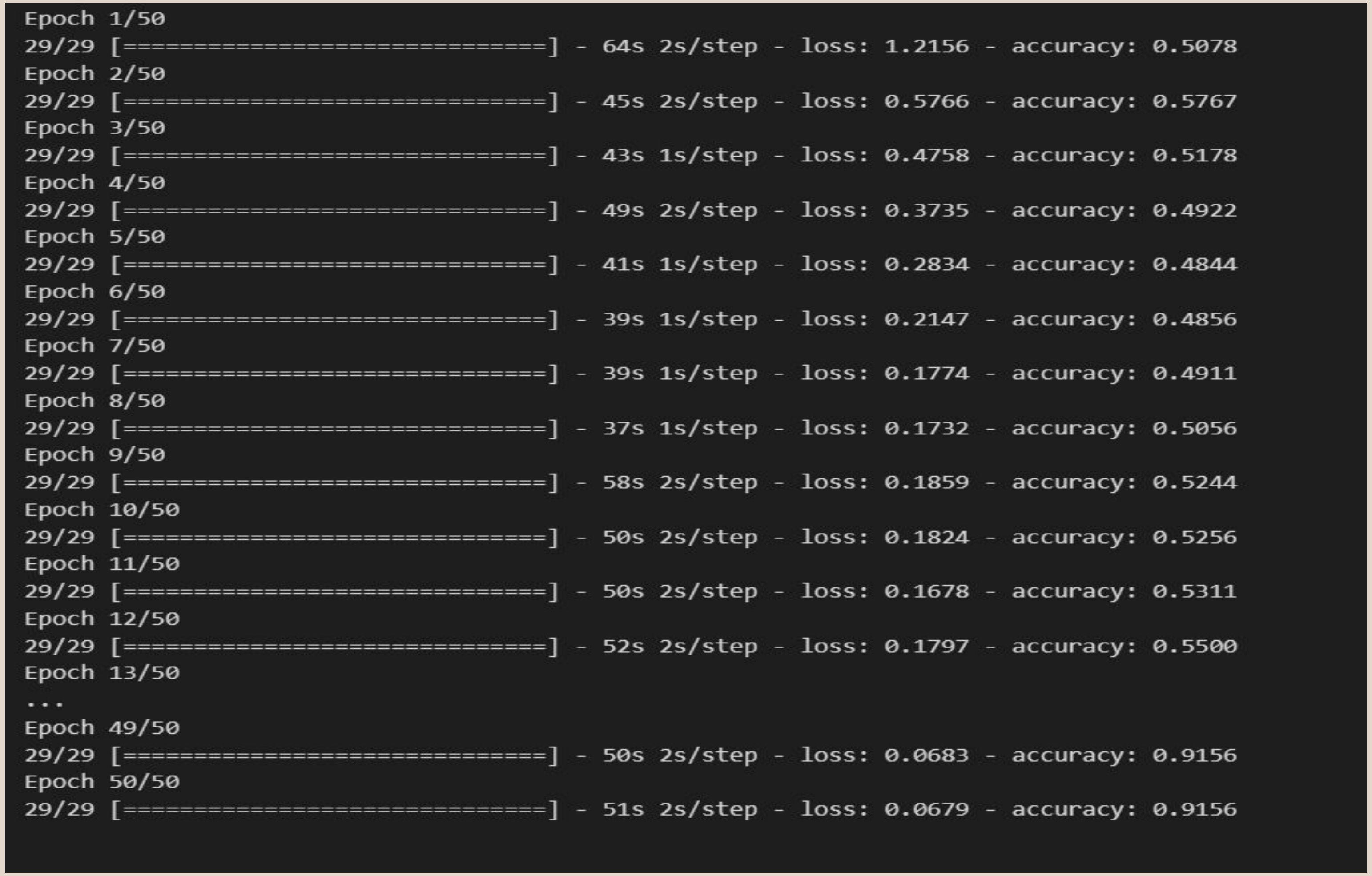
```python
train_loss = history.history['loss']
epochs = range(1, len(train_loss) + 1)

# Plot the training and validation loss
plt.figure()
plt.plot(epochs, train_loss, label='Training loss')
plt.plot(epochs , history.history['accuracy'] , label = 'accuracy')
plt.title('Training')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.savefig(f"trained_models/{EPOCHS}.png")
plt.show()
```

```python
def inverse_linear_scaling(x , a , b):
    return 1.0*x*(b-a) + 1.0*a
```

```python
output[0][0] = inverse_linear_scaling(output[0][0] , 15.0 , 0.2283245321309487)
output[0][1] = inverse_linear_scaling(output[0][1] , 1e+32 , 6e-05)
output[0][2] = inverse_linear_scaling(output[0][2] , 16666666666666.666 , 0.0)
```

```python
for x in output[0]:
    print(x)
```

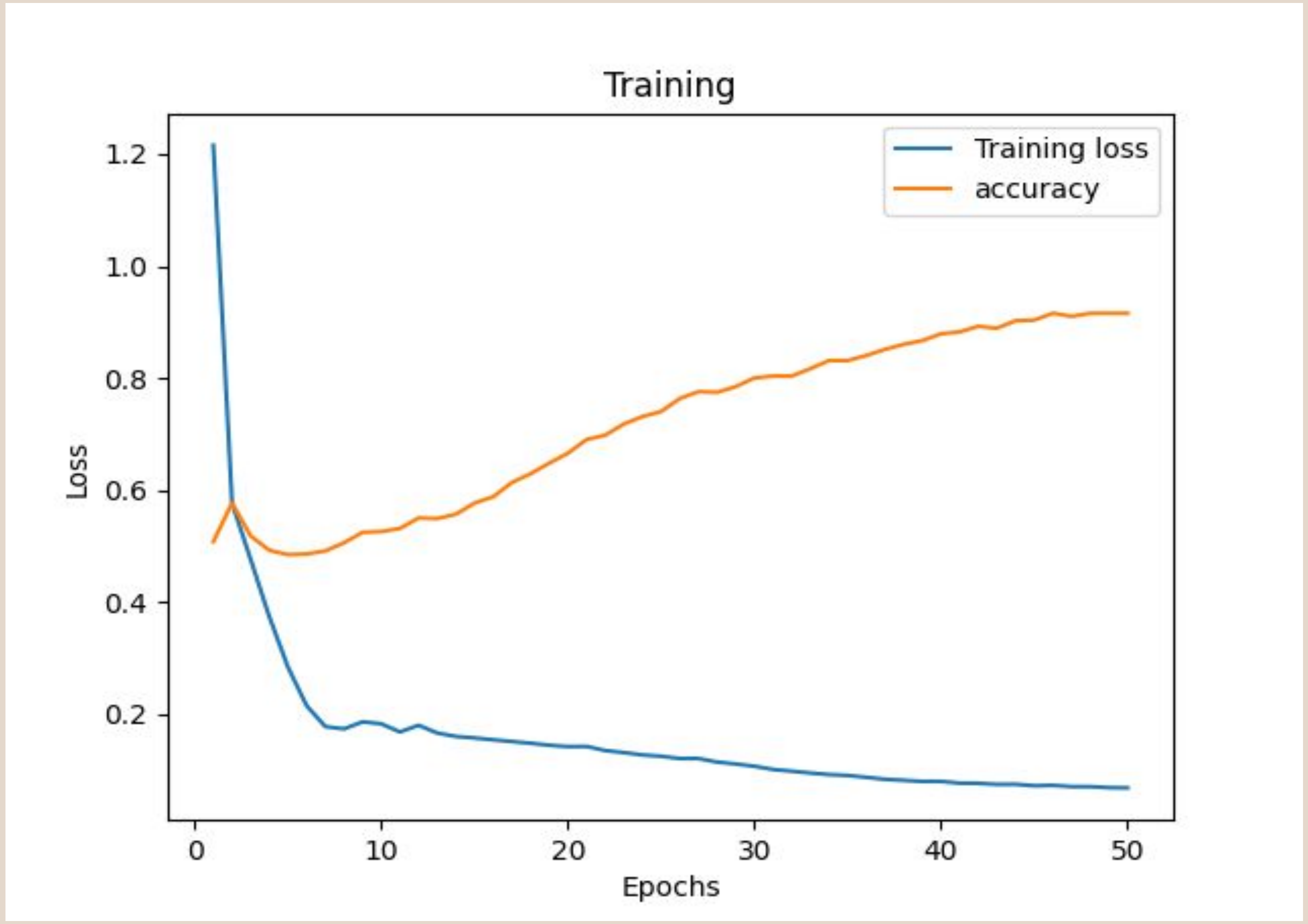
Code generated accuracy and loss



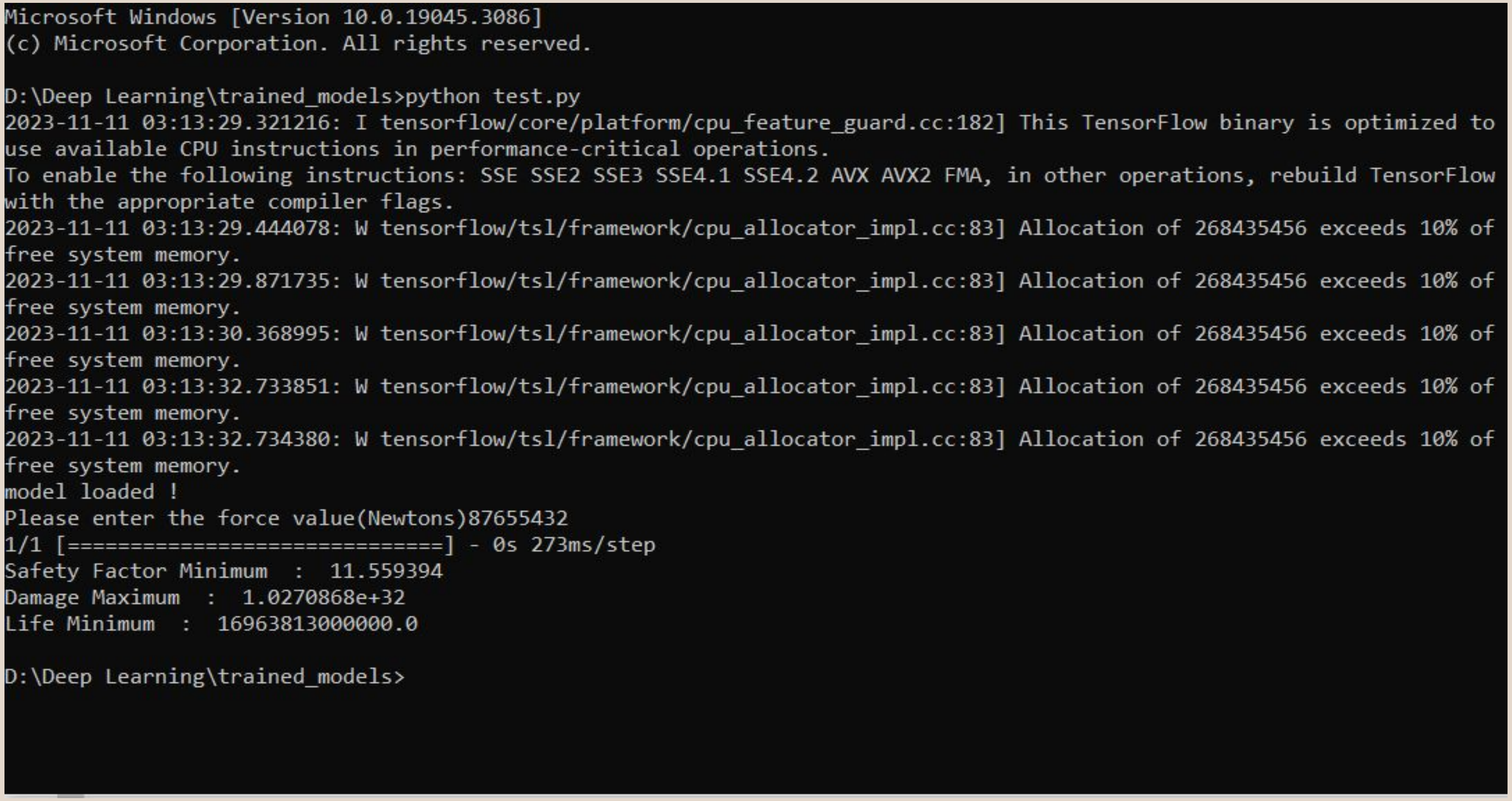
```
Epoch 1/50
29/29 [==============================] - 64s 2s/step - loss: 1.2156 - accuracy: 0.5078
Epoch 2/50
29/29 [==============================] - 45s 2s/step - loss: 0.5766 - accuracy: 0.5767
Epoch 3/50
29/29 [==============================] - 43s 1s/step - loss: 0.4758 - accuracy: 0.5178
Epoch 4/50
29/29 [==============================] - 49s 2s/step - loss: 0.3735 - accuracy: 0.4922
Epoch 5/50
29/29 [==============================] - 41s 1s/step - loss: 0.2834 - accuracy: 0.4844
Epoch 6/50
29/29 [==============================] - 39s 1s/step - loss: 0.2147 - accuracy: 0.4856
Epoch 7/50
29/29 [==============================] - 39s 1s/step - loss: 0.1774 - accuracy: 0.4911
Epoch 8/50
29/29 [==============================] - 37s 1s/step - loss: 0.1732 - accuracy: 0.5056
Epoch 9/50
29/29 [==============================] - 58s 2s/step - loss: 0.1859 - accuracy: 0.5244
Epoch 10/50
29/29 [==============================] - 50s 2s/step - loss: 0.1824 - accuracy: 0.5256
Epoch 11/50
29/29 [==============================] - 50s 2s/step - loss: 0.1678 - accuracy: 0.5311
Epoch 12/50
29/29 [==============================] - 52s 2s/step - loss: 0.1797 - accuracy: 0.5500
Epoch 13/50
...
Epoch 49/50
29/29 [==============================] - 50s 2s/step - loss: 0.0683 - accuracy: 0.9156
Epoch 50/50
29/29 [==============================] - 51s 2s/step - loss: 0.0679 - accuracy: 0.9156
```

An accuracy of **91.56%** is obtained by using the model

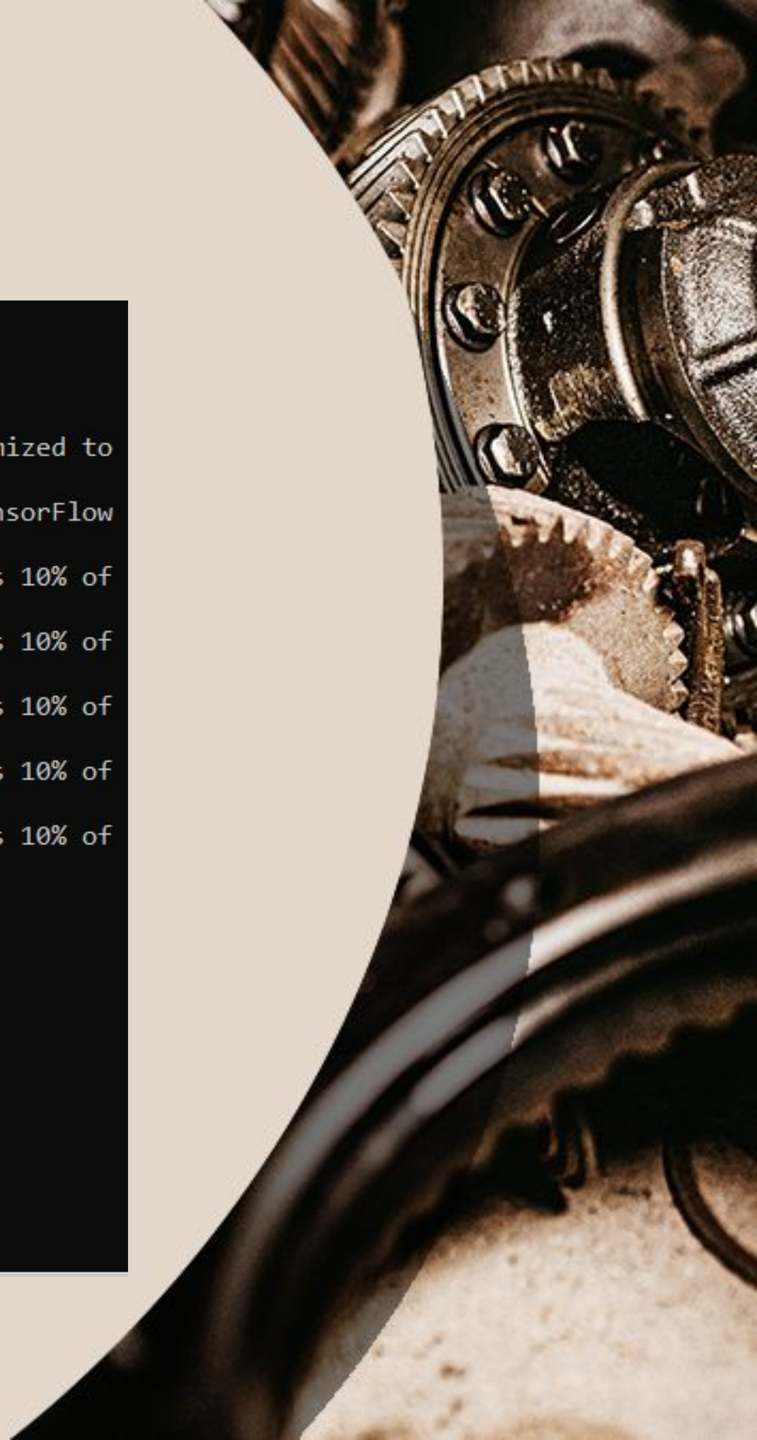
Loss and Accuracy plot



Sample Results



```
Microsoft Windows [Version 10.0.19045.3086]
(c) Microsoft Corporation. All rights reserved.

D:\Deep Learning\trained_models>python test.py
2023-11-11 03:13:29.321216: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to
use available CPU instructions in performance-critical operations.
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow
with the appropriate compiler flags.
2023-11-11 03:13:29.444078: W tensorflow/tsl/framework/cpu_allocator_impl.cc:83] Allocation of 268435456 exceeds 10% of
free system memory.
2023-11-11 03:13:29.871735: W tensorflow/tsl/framework/cpu_allocator_impl.cc:83] Allocation of 268435456 exceeds 10% of
free system memory.
2023-11-11 03:13:30.368995: W tensorflow/tsl/framework/cpu_allocator_impl.cc:83] Allocation of 268435456 exceeds 10% of
free system memory.
2023-11-11 03:13:32.733851: W tensorflow/tsl/framework/cpu_allocator_impl.cc:83] Allocation of 268435456 exceeds 10% of
free system memory.
2023-11-11 03:13:32.734380: W tensorflow/tsl/framework/cpu_allocator_impl.cc:83] Allocation of 268435456 exceeds 10% of
free system memory.
model loaded !
Please enter the force value(Newtons)87655432
1/1 [==============================] - 0s 273ms/step
Safety Factor Minimum  :  11.559394
Damage Maximum  :  1.0270868e+32
Life Minimum  :  16963813000000.0

D:\Deep Learning\trained_models>
```



THANK YOU!!