1. Create your own Multi-layer Perceptron (MLP) with two hidden layers, where the first hidden layer cells' number equals the last three digits of your SID. The number of cells in the next hidden layer is approximately two times smaller.

For example, if your SID is 2287167, the number of cells on the first hidden layer is 167, and on the second - 84. Take epochs=10. Leave other parameters the same as in the practical session.

2. Compile the model.

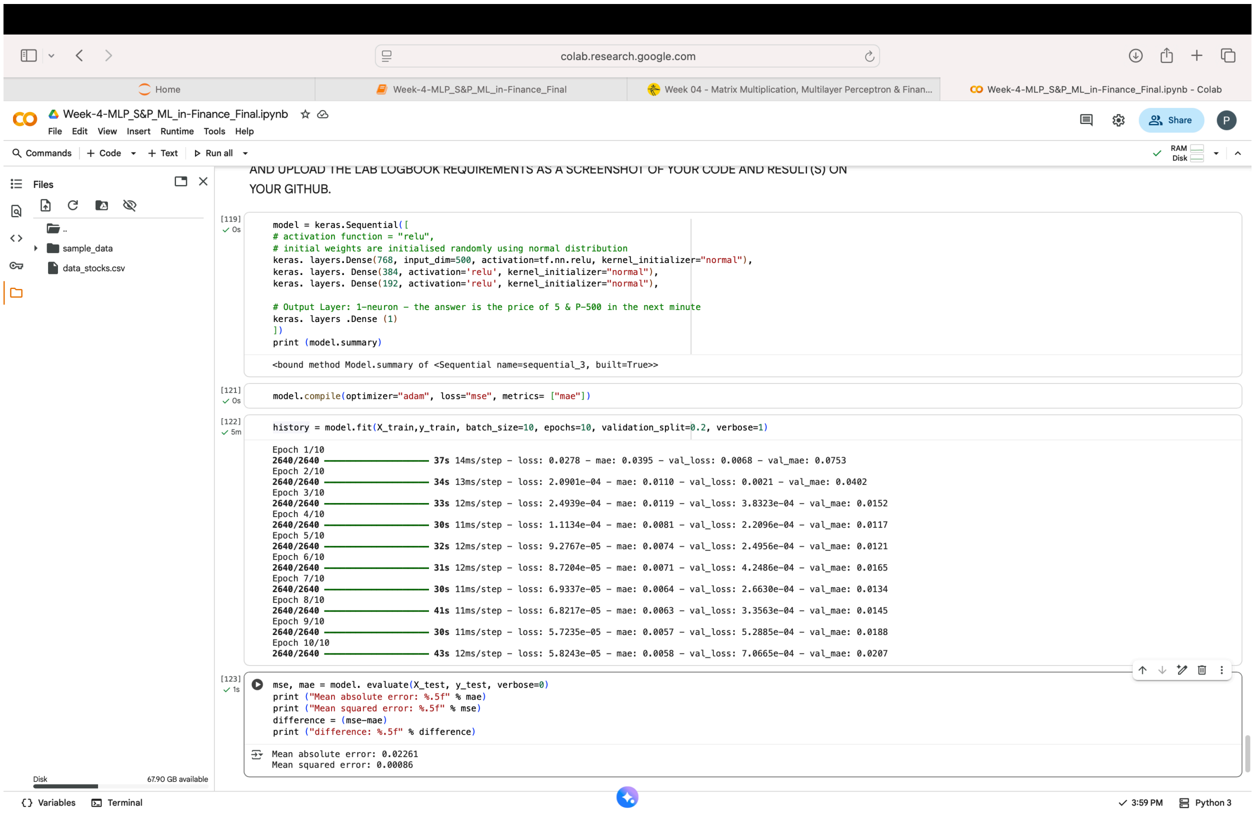
3. Train your MLP with the same datasets and demonstrate the received MAE on the test dataset.

4. Compare your MAE with the MAE of the MLP in the practical session.

5. Please only add to your Lab Logbook a print-screen(s) of your MLP architecture using model.summary(), MLP training code and process, and the resulting MAE on the test dataset.

```
model = keras.Sequential([
# activation function = "relu",
# initial weights are initialised randomly using normal distribution
keras. layers.Dense(768, input_dim=500, activation=tf.nn.relu,
kernel_initializer="normal"),
keras. layers. Dense(384, activation='relu', kernel_initializer="normal"),
keras. layers. Dense(192, activation='relu', kernel_initializer="normal"),

# Output Layer: 1-neuron - the answer is the price of 5 & P-500 in the
next minute
keras. layers .Dense (1)
])
print (model.summary)
model.compile(optimizer="adam", loss="mse", metrics= ["mae"])
```


colab.research.google.com
Home
Week-4-MLP_S&P_ML_in-Finance_Final
Week 04 - Matrix Multiplication, Multilayer Perceptron & Finan...
Week-4-MLP_S&P_ML_in-Finance_Final.ipynb - Colab
Week-4-MLP_S&P_ML_in-Finance_Final.ipynb
File Edit View Insert Runtime Tools Help
Share
P
Commands
+ Code
+ Text
Run all
RAM
Disk
Files
..
sample_data
data_stocks.csv
Disk
67.90 GB available
Variables
Terminal
3:59 PM
Python 3

AND UPLOAD THE LAB LOGBOOK REQUIREMENTS AS A SCREENSHOT OF YOUR CODE AND RESULT(S) ON YOUR GITHUB.

[119] ✓ 0s

```
model = keras.Sequential([
# activation function = "relu",
# initial weights are initialised randomly using normal distribution
keras. layers.Dense(768, input_dim=500, activation=tf.nn.relu, kernel_initializer="normal"),
keras. layers. Dense(384, activation='relu', kernel_initializer="normal"),
keras. layers. Dense(192, activation='relu', kernel_initializer="normal"),

# Output Layer: 1-neuron - the answer is the price of 5 & P-500 in the next minute
keras. layers .Dense (1)
])
print (model.summary)
```

```
<bound method Model.summary of <Sequential name=sequential_3, built=True>>
```

[121] ✓ 0s

```
model.compile(optimizer="adam", loss="mse", metrics= ["mae"])
```

[122] ✓ 5m

```
history = model.fit(X_train,y_train, batch_size=10, epochs=10, validation_split=0.2, verbose=1)
```

```
Epoch 1/10
2640/2640 ━━━━━━━━━━━━━━━━━━━━ 37s 14ms/step - loss: 0.0278 - mae: 0.0395 - val_loss: 0.0068 - val_mae: 0.0753
Epoch 2/10
2640/2640 ━━━━━━━━━━━━━━━━━━━━ 34s 13ms/step - loss: 2.0901e-04 - mae: 0.0110 - val_loss: 0.0021 - val_mae: 0.0402
Epoch 3/10
2640/2640 ━━━━━━━━━━━━━━━━━━━━ 33s 12ms/step - loss: 2.4939e-04 - mae: 0.0119 - val_loss: 3.8323e-04 - val_mae: 0.0152
Epoch 4/10
2640/2640 ━━━━━━━━━━━━━━━━━━━━ 30s 11ms/step - loss: 1.1134e-04 - mae: 0.0081 - val_loss: 2.2096e-04 - val_mae: 0.0117
Epoch 5/10
2640/2640 ━━━━━━━━━━━━━━━━━━━━ 32s 12ms/step - loss: 9.2767e-05 - mae: 0.0074 - val_loss: 2.4956e-04 - val_mae: 0.0121
Epoch 6/10
2640/2640 ━━━━━━━━━━━━━━━━━━━━ 31s 12ms/step - loss: 8.7204e-05 - mae: 0.0071 - val_loss: 4.2486e-04 - val_mae: 0.0165
Epoch 7/10
2640/2640 ━━━━━━━━━━━━━━━━━━━━ 30s 11ms/step - loss: 6.9337e-05 - mae: 0.0064 - val_loss: 2.6630e-04 - val_mae: 0.0134
Epoch 8/10
2640/2640 ━━━━━━━━━━━━━━━━━━━━ 41s 11ms/step - loss: 6.8217e-05 - mae: 0.0063 - val_loss: 3.3563e-04 - val_mae: 0.0145
Epoch 9/10
2640/2640 ━━━━━━━━━━━━━━━━━━━━ 30s 11ms/step - loss: 5.7235e-05 - mae: 0.0057 - val_loss: 5.2885e-04 - val_mae: 0.0188
Epoch 10/10
2640/2640 ━━━━━━━━━━━━━━━━━━━━ 43s 12ms/step - loss: 5.8243e-05 - mae: 0.0058 - val_loss: 7.0665e-04 - val_mae: 0.0207
```

[123] ✓ 1s

```
mse, mae = model. evaluate(X_test, y_test, verbose=0)
print ("Mean absolute error: %.5f" % mae)
print ("Mean squared error: %.5f" % mse)
difference = (mse-mae)
print ("difference: %.5f" % difference)
```

```
Mean absolute error: 0.02261
Mean squared error: 0.00086
```

```
mse, mae = model. evaluate(X_test, y_test, verbose=0)
print ("Mean absolute error: %.5f" % mae)
print ("Mean squared error: %.5f" % mse)
difference = (mse-mae)
print ("difference: %.5f" % difference)
```