Week 1

Lab Logbook Requirement:

Insert a cell below (B)

1) Create a vector using np.arange.
Determine the number of the vector elements using the following method: Take the last two digits from your SID. It should be from 00 to 99. If this number is 10 or more, it becomes the required number of the vector elements. If it is less than 10, add 100 to your number.

For example, if your SID is 2287467, and the last two digits are 67, which is greater than 10. The required number is 67. If your SID is 2287407, and the last two digits are 07, which is less than 10. The required number is 107.

Then,

2. *Change matrix a to 2-d array with 1 row. Print the array. You should have the two sets of brackets for a 2-d array with one row.*
3. *Save it in another array. Print the array.*
4. *Check the shape attribute value.*
5. *Add the code and result to your Lab Logbook*

NOTE: DON'T FORGET TO SAVE AND BACK UP YOUR COMPLETED JUPYTER NOTEBOOK AND LAB LOGBOOK ON GITHUB OR ONEDRIVE.

[2]:
```
import numpy as np
```

[3]:
```
a = np.arange(43)  #sid = 2334343 as the last two digits of the sid is greater than 10
a1 = a.reshape(1, -1) #changing matrix a to 2-d array with 1 row
print(a1)
print('\n')
print(a1.shape)
```

```
[[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
  24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42]]


(1, 43)
```

Week 2

Lab Logbook Requirement: ¶

1. *Determine a number (n) equal to the last digit of your SID.*
2. *Group by "relationship" and "hours-per-week".*
3. *Reduce all "hours-per-week" column values in the original DataFrame by the value 'n'.*
4. *Group by "relationship" and reduced "hours-per-week".*
5. *Add the code and result to your Lab Logbook.*

NOTE: DON'T FORGET TO SAVE AND BACK UP YOUR COMPLETED JUPYTER NOTEBOOK AND LAB LOGBOOK ON GITHUB OR ONEDRIVE.

[20]:
```
#Group by before reducing hours
Group_by_relationship = data.groupby(["relationship", "hours-per-week"])
Group_by_relationship.size()
```

[20]:
```
relationship   hours-per-week
Husband        13.0             1
               40.0             2
               45.0             1
               80.0             1
Not-in-family  16.0             1
               40.0             2
               50.0             2
Own-child      30.0             1
Wife           40.0             2
dtype: int64
```

[21]:
```
n =3 #sid = 2334343

def func(x):
    return x - n

data['hours-per-week'] = data['hours-per-week'].apply(func)
```

[22]:
```
#Group by after reducing hours
Group_by_relationship = data.groupby(["relationship", "hours-per-week"])
Group_by_relationship.size()
```

[22]:
```
relationship   hours-per-week
Husband        10.0             1
               37.0             2
               42.0             1
               77.0             1
Not-in-family  13.0             1
               37.0             2
               47.0             2
Own-child      27.0             1
Wife           37.0             2
dtype: int64
```

Week 3

Lab Logbook Requirement:

1) Draw a bicolour features interaction diagram between the columns with the numbers of the last and second to last digits of your SID, where:

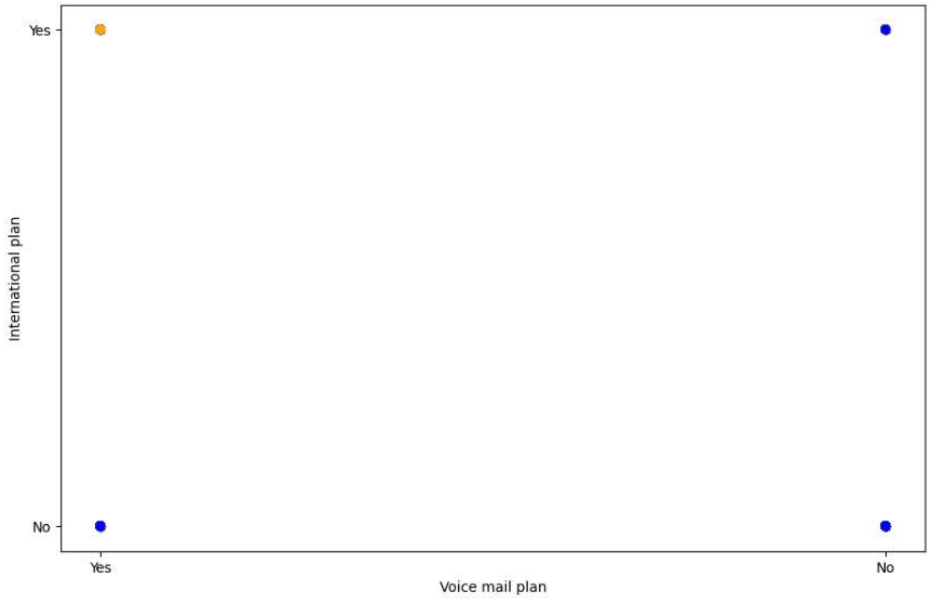
#	Column
1	Account length
2	Area code
3	International plan
4	Voice mail plan
5	Number vmail messages
6	Total day minutes
7	Total day calls
8	Total day charge
9	Total eve minutes
0	Total eve calls

In case these numbers are the same, then take the next number in order as another column number. For example, if your SID is 2287477, then you plot the bicolour diagram of the 7th and 8th columns. If your SID is 2287499, then the 9th and 0.

2. Add the code and result to your Lab Logbook.

NOTE: DON'T FORGET TO SAVE AND BACK UP YOUR COMPLETED JUPYTER NOTEBOOK AND LAB LOGBOOK ON GITHUB OR ONEDRIVE.

[131]:
```
#sid = 2334343 so plotted the bicolour diagram of the 6th and 7th columns.

fig = plt.figure(figsize=(11,7))
plt.scatter(data['Voice mail plan'], data['International plan'], color = Clr);
plt.xlabel('Voice mail plan');
plt.ylabel('International plan');
```



[ ]:

Week 4

Lab Logbook Requirement:

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.*
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 of your MLP architecture using model.summary() and the resulting MAE.*

NOTE: DON'T FORGET TO SAVE AND BACK UP YOUR COMPLETED JUPYTER NOTEBOOK AND LAB LOGBOOK ON GITHUB OR ONEDRIVE.

[ ]:

[54]:
```
model = keras.Sequential([

    keras.layers.Dense(343, input_dim = 500, activation = tf.nn.relu, kernel_initializer = "normal"),   #sid = 2244067

    keras.layers.Dense(172, activation = 'relu', kernel_initializer = "normal"),

    keras.layers.Dense(1)

])

print(model.summary())
```

```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
dense_5 (Dense)	(None, 343)	171,843
dense_6 (Dense)	(None, 172)	59,168
dense_7 (Dense)	(None, 1)	173

```
Total params: 231,184 (903.06 KB)
Trainable params: 231,184 (903.06 KB)
Non-trainable params: 0 (0.00 B)
None
```

[55]:
```
model.compile(optimizer = "adam", loss = "mse", metrics = ["mae"])
```

[56]:
```
history = model.fit(X_train, y_train, batch_size =10, epochs = 10, validation_split = 0.2, verbose = 1)
```

```
Epoch 1/10
2640/2640 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step - loss: 0.0104 - mae: 0.0324 - val_loss: 0.0138 - val_mae: 0.1123
Epoch 2/10
2640/2640 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step - loss: 1.9849e-04 - mae: 0.0108 - val_loss: 0.0010 - val_mae: 0.0286
Epoch 3/10
2640/2640 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step - loss: 1.4838e-04 - mae: 0.0095 - val_loss: 0.0012 - val_mae: 0.0310
Epoch 4/10
2640/2640 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step - loss: 1.0791e-04 - mae: 0.0079 - val_loss: 2.8772e-04 - val_mae: 0.0140
Epoch 5/10
2640/2640 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step - loss: 7.6418e-05 - mae: 0.0067 - val_loss: 4.8730e-04 - val_mae: 0.0181
Epoch 6/10
2640/2640 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step - loss: 6.2446e-05 - mae: 0.0061 - val_loss: 4.7023e-04 - val_mae: 0.0176
Epoch 7/10
2640/2640 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step - loss: 6.2205e-05 - mae: 0.0059 - val_loss: 4.5517e-04 - val_mae: 0.0173
Epoch 8/10
2640/2640 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step - loss: 5.6142e-05 - mae: 0.0057 - val_loss: 3.1351e-04 - val_mae: 0.0147
Epoch 9/10
2640/2640 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step - loss: 5.2162e-05 - mae: 0.0055 - val_loss: 3.4261e-04 - val_mae: 0.0155
Epoch 10/10
2640/2640 ━━━━━━━━━━━━━━━━━━━━ 5s 1ms/step - loss: 5.0011e-05 - mae: 0.0053 - val_loss: 3.8917e-04 - val_mae: 0.0160
```

[57]:
```
print("Mean absolute error: %.5f" % mae)
```

```
Mean absolute error: 0.02263
```

Week 5

Lab Logbook Requirement:

1. *Modify the practical session CNN model by reducing the convolutional core size to 5.*
2. *Change the batch_size to 50.*
3. *Also, change the size of the number of epochs, which is calculated by the formula:*

   Z + Y, if Z = 0

   10 + Y, if Z = 0 and Y is not 0

   10, if Z = Y = 0

   , where your SID is: XXXXXZY
4. *Leave other parameters the same as in the practical session.*
5. *Compile the model.*
6. *Train your CNN with the same datasets and demonstrate the received test MAE. Compare your MAE with the MAE of the CNN in the practical session.*
7. *Please only add a print-screen of your CNN architecture using model.summary() and the resulting MAE to your Lab Logbook.*

¶

NOTE: DON'T FORGET TO SAVE AND BACK UP YOUR COMPLETED JUPYTER NOTEBOOK AND LAB LOGBOOK ON GITHUB OR ONEDRIVE.

[ ]:

[49]:
```
model = keras.Sequential([
    keras.layers.Conv1D(50,5, padding = 'same', input_shape= (50,5), activation=tf.nn.relu, kernel_initializer='normal'),
    keras.layers.MaxPooling1D(7),
    keras.layers.Conv1D(100,5,padding = 'same', activation = tf.nn.relu, kernel_initializer = "normal"),
    keras.layers.GlobalMaxPooling1D(),
    keras.layers.Dense(25, activation =tf.nn.relu, kernel_initializer = "normal"),
    keras.layers.Dense(2)
])
print(model.summary())
```

```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
conv1d_2 (Conv1D)	(None, 50, 50)	1,300
max_pooling1d_1 (MaxPooling1D)	(None, 7, 50)	0
conv1d_3 (Conv1D)	(None, 7, 100)	25,100
global_max_pooling1d_1 (GlobalMaxPooling1D)	(None, 100)	0
dense_2 (Dense)	(None, 25)	2,525
dense_3 (Dense)	(None, 2)	52

Total params: 28,977 (113.19 KB)

Trainable params: 28,977 (113.19 KB)

Non-trainable params: 0 (0.00 B)

None

[50]:
```
model.compile(optimizer = "adam", loss = "mse", metrics = ["mae"])
```

[51]:
```
history = model.fit(X_train, y_train, batch_size =50, epochs=13, validation_split=0.2, verbose=1)

#sid = 2244067 where Z = 6 and Y = 7
```

```
Epoch 1/13
3520/3520 ━━━━━━━━━━━━━━━━━━━━ 9s 2ms/step - loss: 0.0096 - mae: 0.0480 - val_loss: 0.0010 - val_mae: 0.0225
Epoch 2/13
3520/3520 ━━━━━━━━━━━━━━━━━━━━ 8s 2ms/step - loss: 7.7049e-04 - mae: 0.0189 - val_loss: 8.6747e-04 - val_mae: 0.0191
Epoch 3/13
3520/3520 ━━━━━━━━━━━━━━━━━━━━ 9s 3ms/step - loss: 7.2412e-04 - mae: 0.0182 - val_loss: 0.0011 - val_mae: 0.0235
Epoch 4/13
3520/3520 ━━━━━━━━━━━━━━━━━━━━ 9s 2ms/step - loss: 7.3431e-04 - mae: 0.0182 - val_loss: 8.3857e-04 - val_mae: 0.0188
Epoch 5/13
3520/3520 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - loss: 6.9110e-04 - mae: 0.0177 - val_loss: 8.4827e-04 - val_mae: 0.0190
Epoch 6/13
3520/3520 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - loss: 6.9858e-04 - mae: 0.0178 - val_loss: 8.4128e-04 - val_mae: 0.0188
Epoch 7/13
3520/3520 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - loss: 6.8478e-04 - mae: 0.0176 - val_loss: 8.3836e-04 - val_mae: 0.0187
Epoch 8/13
3520/3520 ━━━━━━━━━━━━━━━━━━━━ 11s 3ms/step - loss: 6.8631e-04 - mae: 0.0175 - val_loss: 8.5708e-04 - val_mae: 0.0192
Epoch 9/13
3520/3520 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - loss: 6.6906e-04 - mae: 0.0174 - val_loss: 8.3086e-04 - val_mae: 0.0186
Epoch 10/13
3520/3520 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - loss: 6.9581e-04 - mae: 0.0175 - val_loss: 8.3485e-04 - val_mae: 0.0186
Epoch 11/13
3520/3520 ━━━━━━━━━━━━━━━━━━━━ 9s 3ms/step - loss: 6.7223e-04 - mae: 0.0174 - val_loss: 8.5745e-04 - val_mae: 0.0191
Epoch 12/13
3520/3520 ━━━━━━━━━━━━━━━━━━━━ 9s 3ms/step - loss: 6.8770e-04 - mae: 0.0175 - val_loss: 8.1809e-04 - val_mae: 0.0183
Epoch 13/13
3520/3520 ━━━━━━━━━━━━━━━━━━━━ 9s 3ms/step - loss: 6.8676e-04 - mae: 0.0175 - val_loss: 8.2589e-04 - val_mae: 0.0186
```

[52]:
```
mse,mae = model.evaluate(X_test, y_test, verbose =1)
print("Mean absolute error: %.5f" %mae)
```

```
936/936 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 0.0012 - mae: 0.0229
Mean absolute error: 0.02439
```

Week 6

Lab Logbook Requirement:

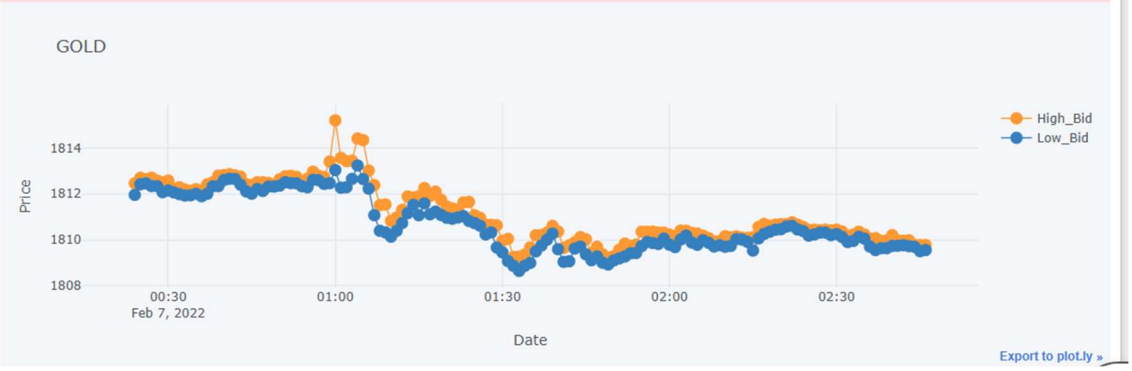
1. *Plot the price chart of the part of the whole dataset 'High_Bid' and 'Low_Bid' prices using iplot() library.*
2. *The start point should equal the 5 last digits of your SID Number.*
3. *The time period (in minutes) should equal the 3 last digits of your SID Number.*
4. *Please only add a print-screen of your code and final graph to your Lab Logbook.*

NOTE: DON'T FORGET TO SAVE AND BACK UP YOUR COMPLETED JUPYTER NOTEBOOK AND LAB LOGBOOK ON GITHUB OR ONEDRIVE.

[ ]:

[126]:
```
#sid = 2334343

#last 5 digits = 34343

#last 3 digits = 343 - added 143 minutes to the start point

data.iloc[34343:34486,:][['High_Bid', 'Low_Bid', 'Local_time_T']].iplot(
                                                x='Local_time_T', y=['High_Bid', 'Low_Bid'],
                                                mode='lines+markers',
                                                xTitle='Date', yTitle='Price',
                                                title='GOLD ')
```

```
C:\Users\abkha\anaconda3\Lib\site-packages\cufflinks\plotlytools.py:117: FutureWarning:

DatetimeIndex.format is deprecated and will be removed in a future version. Convert using index.astype(str) or index.map(formatter) instead.
```



Week 7

Lab Logbook Requirement:

1. *Modify the practical session LSTM model parameter from 100 to be calculated using the formula:*

   ZY + 10 , where your SID is: XXXXXZY

2. *Change the epochs to 10.*
3. *Change the patience to 3*
4. *Leave other parameters the same as in the practical session.*
5. *Compile the model.*
6. *Train your LSTM with the same datasets and demonstrate the received test MSE & MAE. Compare your test MSE & MAE with the MSE & MAE of the LSTM in the practical session.*
7. *Please only add to your Lab Logbook print-screens of:*

- *your LSTM architecture using model.summary(),*
- *the resulting test MSE & MAE and*
- *MAE detailed graph*

NOTE: DON'T FORGET TO SAVE AND BACK UP YOUR COMPLETED JUPYTER NOTEBOOK AND LAB LOGBOOK ON GITHUB OR ONEDRIVE.

[ ]:

[74]:
```
#sid =2334343 where Z=4 and Y=3
#ZY +10 = 43 + 10 = 53

model = keras.Sequential([

    keras.layers.LSTM(53, activation = 'relu', input_shape = (50, 18)),

    keras.layers.Dense(2)

])

print(model.summary())
```

```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 53)	15,264
dense_1 (Dense)	(None, 2)	108

```
Total params: 15,372 (60.05 KB)
Trainable params: 15,372 (60.05 KB)
Non-trainable params: 0 (0.00 B)
None
```

```
None
```

[75]:
```
model.compile(optimizer = "adam", loss = "mse", metrics =["mae"])
```

[76]:
```
es = EarlyStopping(monitor='val_loss', mode='min', patience=3, verbose=1)
mc = ModelCheckpoint('best_model_LSTM_GOLD.keras', monitor='val_loss', mode='min', verbose=1, save_best_only=True)
```

[77]:
```
history = model.fit(X_train, y_train, batch_size = 20, epochs = 10, validation_split = 0.1, shuffle = True, verbose =1, callbacks = [es,mc])
```

```
Epoch 1/10
1209/1213 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.0623 - mae: 0.0623
Epoch 1: val_loss improved from inf to 0.00006, saving model to best_model_LSTM_GOLD.keras
1213/1213 ━━━━━━━━━━━━━━━━━━━━ 10s 7ms/step - loss: 0.0621 - mae: 0.0621 - val_loss: 5.5415e-05 - val_mae: 0.0059
Epoch 2/10
1213/1213 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 3.2154e-05 - mae: 0.0044
Epoch 2: val_loss improved from 0.00006 to 0.00002, saving model to best_model_LSTM_GOLD.keras
1213/1213 ━━━━━━━━━━━━━━━━━━━━ 9s 7ms/step - loss: 3.2147e-05 - mae: 0.0044 - val_loss: 1.6773e-05 - val_mae: 0.0028
Epoch 3/10
1213/1213 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 1.7525e-05 - mae: 0.0033
Epoch 3: val_loss improved from 0.00002 to 0.00002, saving model to best_model_LSTM_GOLD.keras
1213/1213 ━━━━━━━━━━━━━━━━━━━━ 9s 7ms/step - loss: 1.7526e-05 - mae: 0.0033 - val_loss: 1.5374e-05 - val_mae: 0.0029
Epoch 4/10
1206/1213 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 2.1401e-05 - mae: 0.0037
Epoch 4: val_loss improved from 0.00002 to 0.00001, saving model to best_model_LSTM_GOLD.keras
1213/1213 ━━━━━━━━━━━━━━━━━━━━ 10s 8ms/step - loss: 2.1409e-05 - mae: 0.0037 - val_loss: 1.1462e-05 - val_mae: 0.0025
Epoch 5/10
1207/1213 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 2.1597e-05 - mae: 0.0036
Epoch 5: val_loss did not improve from 0.00001
1213/1213 ━━━━━━━━━━━━━━━━━━━━ 10s 8ms/step - loss: 2.1611e-05 - mae: 0.0036 - val_loss: 1.5321e-05 - val_mae: 0.0032
Epoch 6/10
1213/1213 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 2.3703e-05 - mae: 0.0038
Epoch 6: val_loss improved from 0.00001 to 0.00001, saving model to best_model_LSTM_GOLD.keras
1213/1213 ━━━━━━━━━━━━━━━━━━━━ 10s 8ms/step - loss: 2.3704e-05 - mae: 0.0038 - val_loss: 5.6829e-06 - val_mae: 0.0017
Epoch 7/10
1211/1213 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 1.9773e-05 - mae: 0.0035
Epoch 7: val_loss did not improve from 0.00001
1213/1213 ━━━━━━━━━━━━━━━━━━━━ 10s 8ms/step - loss: 1.9779e-05 - mae: 0.0035 - val_loss: 3.1170e-05 - val_mae: 0.0053
Epoch 8/10
1213/1213 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 1.8950e-05 - mae: 0.0034
Epoch 8: val_loss did not improve from 0.00001
1213/1213 ━━━━━━━━━━━━━━━━━━━━ 13s 10ms/step - loss: 1.8949e-05 - mae: 0.0034 - val_loss: 2.3394e-05 - val_mae: 0.0043
Epoch 9/10
1213/1213 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 1.9567e-05 - mae: 0.0035
Epoch 9: val_loss did not improve from 0.00001
1213/1213 ━━━━━━━━━━━━━━━━━━━━ 13s 10ms/step - loss: 1.9566e-05 - mae: 0.0035 - val_loss: 1.6013e-05 - val_mae: 0.0036
Epoch 9: early stopping
```

[78]:
```
scores = LSTM_saved_best_model.evaluate(X_test, y_test, verbose=1)
```

```
94/94 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 7.0312e-06 - mae: 0.0021
```

[79]:
```
scores
```

[79]:
```
[6.354778633976821e-06, 0.0020055680070072412]
```

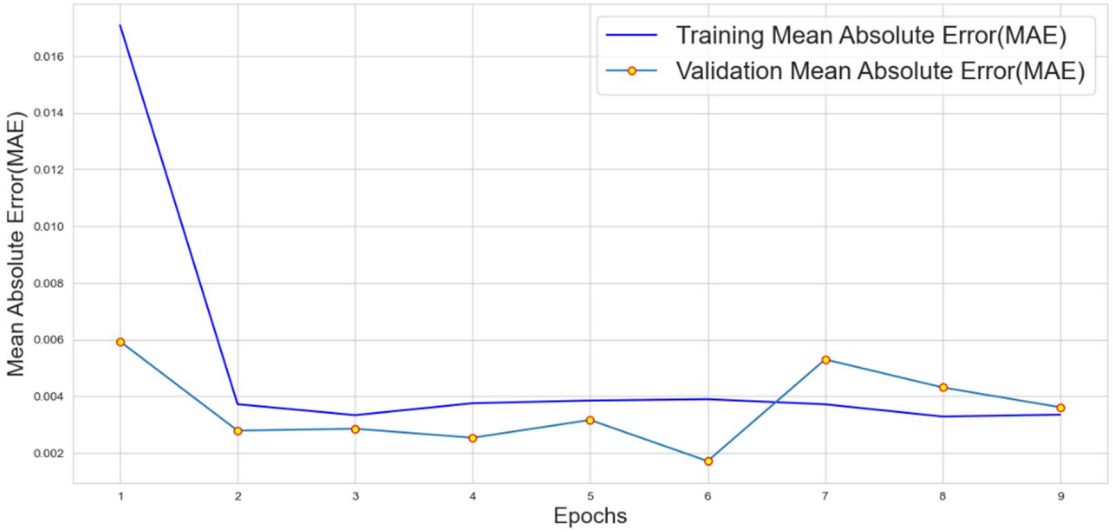
[80]:
```
print("Mean squared error (mse): %.9f " % (scores[0]))
```

```
Mean squared error (mse): 0.000006355
```

[81]:
```
print("Mean absolute error (mae): %.9f " % (scores[1]))
```

```
Mean absolute error (mae): 0.002005568
```

[82]:
```
history_dict = history.history

mae_values = history_dict['mae']
val_mae_values = history_dict['val_mae']

epochs = range(1, len(mae_values) + 1)
plt.figure(num=1, figsize=(15,7))
plt.plot(epochs, mae_values, 'b', label='Training Mean Absolute Error(MAE)')
plt.plot(epochs, val_mae_values, marker='o', markeredgecolor='red', markerfacecolor='yellow', label='Validation Mean Absolute Error(MAE)')
plt.xlabel('Epochs', size=18)
plt.ylabel('Mean Absolute Error(MAE)', size=18)
plt.legend()
plt.show()
```



[ ]:

Week 10

Lab Logbook Requirement:

- *Plot 4 graphs:*
  1. *Precision during training graph*
  2. *More detailed Precision graph*
  3. *Training accuracy graph*
  4. *More detailed Accuracy graph*

NOTE: DON'T FORGET TO SAVE AND BACK UP YOUR COMPLETED JUPYTER NOTEBOOK AND LAB LOGBOOK ON GITHUB OR ONEDRIVE.

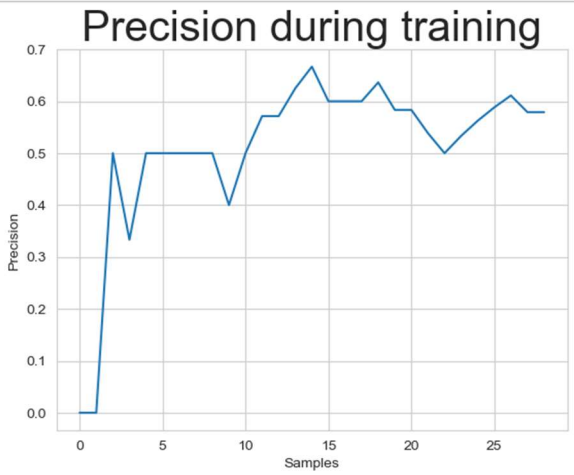
[ ]:

Precision during training

[152]:
```
import numpy as np
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score
import matplotlib.pyplot as plt


daily_data = pd.read_csv("XAUUSD_Daily_Ask_2024.01.01_2024.06.30.csv")
daily_data['Time (UTC)'] = pd.to_datetime(daily_data['Time (UTC)'])
daily_data.set_index('Time (UTC)', inplace=True)
daily_data = daily_data[['Close']]


daily_data['Target'] = (daily_data['Close'].shift(-1) > daily_data['Close']).astype(int)
daily_data.dropna(inplace=True)
daily_data['Close'] = (daily_data['Close'] - daily_data['Close'].mean()) / daily_data['Close'].std()


sequence_length = 10
X, y = [], []
for i in range(len(daily_data) - sequence_length):
    X.append(daily_data['Close'].iloc[i:i + sequence_length].values)
    y.append(daily_data['Target'].iloc[i + sequence_length])
X = np.array(X)
y = np.array(y)


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))


model = Sequential([
    LSTM(50, input_shape=(X_train.shape[1], X_train.shape[2]), activation='relu'),
    Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])


model.fit(X_train, y_train, epochs=20, batch_size=8, verbose=1, validation_split=0.2)
```

```
y_pred = (model.predict(X_test) > 0.5).astype(int)
precision_values = [precision_score(y_test[:i+1], y_pred[:i+1]) for i in range(len(y_test))]


plt.plot(precision_values)
plt.xlabel('Samples')
plt.ylabel('Precision')
plt.title('Precision during training')
plt.show()
```

```
Epoch 1/20
C:\Users\abkha\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer i
n the model instead.

12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.4099 - loss: 0.7031 - val_accuracy: 0.5417 - val_loss: 0.6900
Epoch 2/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5771 - loss: 0.6890 - val_accuracy: 0.5417 - val_loss: 0.6842
Epoch 3/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6789 - loss: 0.6755 - val_accuracy: 0.5417 - val_loss: 0.6801
Epoch 4/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5755 - loss: 0.6793 - val_accuracy: 0.5417 - val_loss: 0.6780
Epoch 5/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6051 - loss: 0.6829 - val_accuracy: 0.4583 - val_loss: 0.6773
Epoch 6/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6143 - loss: 0.6817 - val_accuracy: 0.4583 - val_loss: 0.6774
Epoch 7/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6461 - loss: 0.6746 - val_accuracy: 0.4583 - val_loss: 0.6786
Epoch 8/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6253 - loss: 0.6752 - val_accuracy: 0.4583 - val_loss: 0.6800
Epoch 9/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6284 - loss: 0.6832 - val_accuracy: 0.4583 - val_loss: 0.6817
Epoch 10/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5811 - loss: 0.6829 - val_accuracy: 0.4583 - val_loss: 0.6832
Epoch 11/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6133 - loss: 0.6695 - val_accuracy: 0.4583 - val_loss: 0.6851
Epoch 12/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7087 - loss: 0.6362 - val_accuracy: 0.4583 - val_loss: 0.6880
Epoch 13/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6355 - loss: 0.6680 - val_accuracy: 0.4583 - val_loss: 0.6893
Epoch 14/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6610 - loss: 0.6425 - val_accuracy: 0.4583 - val_loss: 0.6909
Epoch 15/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6431 - loss: 0.6640 - val_accuracy: 0.4583 - val_loss: 0.6932
Epoch 16/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6323 - loss: 0.6668 - val_accuracy: 0.4167 - val_loss: 0.6982
Epoch 17/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5878 - loss: 0.6895 - val_accuracy: 0.4583 - val_loss: 0.7018
Epoch 18/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5360 - loss: 0.6843 - val_accuracy: 0.5417 - val_loss: 0.7067
Epoch 19/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6422 - loss: 0.6601 - val_accuracy: 0.5000 - val_loss: 0.7147
Epoch 20/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6764 - loss: 0.6429 - val_accuracy: 0.5417 - val_loss: 0.7239
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 88ms/step
```



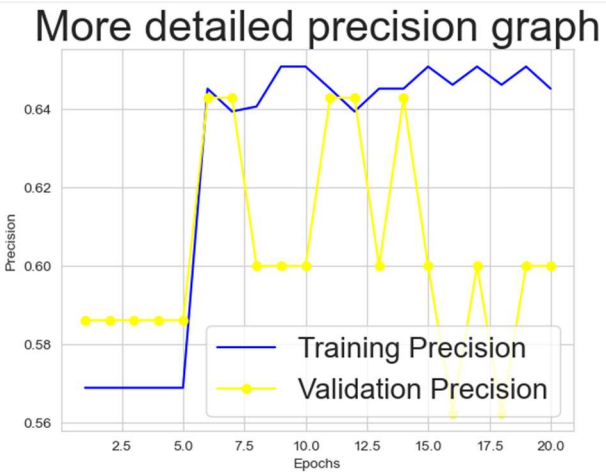
[153]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.callbacks import Callback
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score


daily_data = pd.read_csv("XAUUSD_Daily_Ask_2024.01.01_2024.06.30.csv")
daily_data['Time (UTC)'] = pd.to_datetime(daily_data['Time (UTC)'])
daily_data.set_index('Time (UTC)', inplace=True)
daily_data = daily_data[['Close']]


daily_data['Target'] = (daily_data['Close'].shift(-1) > daily_data['Close']).astype(int)
daily_data.dropna(inplace=True)
daily_data['Close'] = (daily_data['Close'] - daily_data['Close'].mean()) / daily_data['Close'].std()


sequence_length = 10
X, y = [], []
for i in range(len(daily_data) - sequence_length):
    X.append(daily_data['Close'].iloc[i:i + sequence_length].values)
    y.append(daily_data['Target'].iloc[i + sequence_length])
X = np.array(X)
y = np.array(y)


X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_val = X_val.reshape((X_val.shape[0], X_val.shape[1], 1))


class PrecisionCallback(Callback):
    def __init__(self):
        self.train_precision = []
        self.val_precision = []

    def on_epoch_end(self, epoch, logs=None):
```

```
    def on_epoch_end(self, epoch, logs=None):
        y_train_pred = (self.model.predict(X_train) > 0.5).astype(int)
        train_precision = precision_score(y_train, y_train_pred)
        self.train_precision.append(train_precision)

        y_val_pred = (self.model.predict(X_val) > 0.5).astype(int)
        val_precision = precision_score(y_val, y_val_pred)
        self.val_precision.append(val_precision)


model = Sequential([
    LSTM(50, input_shape=(X_train.shape[1], X_train.shape[2]), activation='relu'),
    Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])


precision_callback = PrecisionCallback()
model.fit(X_train, y_train, epochs=20, batch_size=8, verbose=1, validation_data=(X_val, y_val), callbacks=[precision_callback])


epochs = range(1, len(precision_callback.train_precision) + 1)
plt.plot(epochs, precision_callback.train_precision, label='Training Precision', color='blue')
plt.plot(epochs, precision_callback.val_precision, label='Validation Precision', color='yellow', marker='o')
plt.xlabel('Epochs')
plt.ylabel('Precision')
plt.title('More detailed precision graph')
plt.legend()
plt.show()
```

```
Epoch 1/20
C:\Users\abkha\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer i
n the model instead.

1/4 ━━━━━━━━━━━━━━━━━━━━ 0s 95ms/stepp - accuracy: 0.7500 - loss: 0.6656WARNING:tensorflow:5 out of the last 7 calls to <function TensorFlowTrainer.ma
ke_predict_function.<locals>.one_step_on_data_distributed at 0x0000022E2B17F380> triggered tf.function retracing. Tracing is expensive and the excessi
ve number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python ob
jects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=True option that ca
n avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.or
g/api_docs/python/tf/function for  more details.
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 2s 39ms/step - accuracy: 0.5725 - loss: 0.6861 - val_accuracy: 0.5862 - val_loss: 0.6869
Epoch 2/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/steptep - accuracy: 0.5000 - loss: 0.698
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.5955 - loss: 0.6839 - val_accuracy: 0.5862 - val_loss: 0.6861
Epoch 3/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step ep - accuracy: 0.6250 - loss: 0.679
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.5882 - loss: 0.6795 - val_accuracy: 0.5862 - val_loss: 0.6865
Epoch 4/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step ep - accuracy: 0.5000 - loss: 0.708
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.5230 - loss: 0.6923 - val_accuracy: 0.5862 - val_loss: 0.6880
Epoch 5/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step ep - accuracy: 0.5000 - loss: 0.665
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.5452 - loss: 0.6877 - val_accuracy: 0.5862 - val_loss: 0.6884
Epoch 6/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step ep - accuracy: 0.5000 - loss: 0.663
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.5566 - loss: 0.6715 - val_accuracy: 0.5517 - val_loss: 0.6911
Epoch 7/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step ep - accuracy: 0.5000 - loss: 0.776
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.5733 - loss: 0.7023 - val_accuracy: 0.5517 - val_loss: 0.6886
```

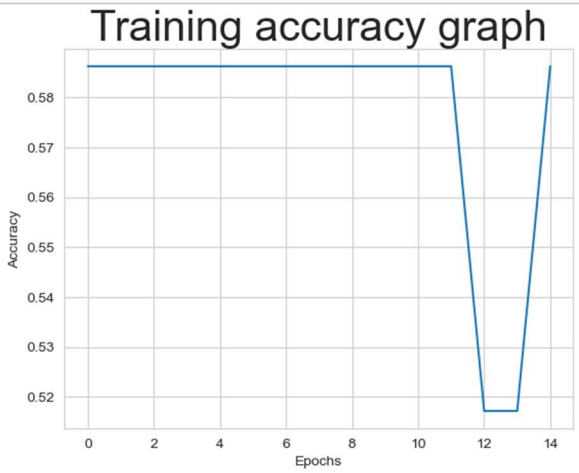
```
Epoch 8/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step ep - accuracy: 0.7500 - loss: 0.605
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.6319 - loss: 0.6518 - val_accuracy: 0.5172 - val_loss: 0.6893
Epoch 9/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step ep - accuracy: 0.7500 - loss: 0.684
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.5990 - loss: 0.6867 - val_accuracy: 0.5172 - val_loss: 0.6869
Epoch 10/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step ep - accuracy: 0.5000 - loss: 0.701
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.5759 - loss: 0.6939 - val_accuracy: 0.5172 - val_loss: 0.6870
Epoch 11/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step ep - accuracy: 0.3750 - loss: 0.813
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.5150 - loss: 0.7071 - val_accuracy: 0.5517 - val_loss: 0.6867
Epoch 12/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step ep - accuracy: 0.5000 - loss: 0.721
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.5974 - loss: 0.6793 - val_accuracy: 0.5517 - val_loss: 0.6897
Epoch 13/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step ep - accuracy: 0.5000 - loss: 0.649
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.5814 - loss: 0.6775 - val_accuracy: 0.5172 - val_loss: 0.6871
Epoch 14/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step ep - accuracy: 0.7500 - loss: 0.683
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6398 - loss: 0.6717 - val_accuracy: 0.5517 - val_loss: 0.6879
Epoch 15/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step ep - accuracy: 0.8750 - loss: 0.621
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.5984 - loss: 0.6819 - val_accuracy: 0.5172 - val_loss: 0.6848
Epoch 16/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step ep - accuracy: 0.3750 - loss: 0.664
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.5484 - loss: 0.6646 - val_accuracy: 0.4828 - val_loss: 0.6847
Epoch 17/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step ep - accuracy: 0.7500 - loss: 0.582
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.6272 - loss: 0.6528 - val_accuracy: 0.5172 - val_loss: 0.6841
Epoch 18/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step ep - accuracy: 0.3750 - loss: 0.667
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.5370 - loss: 0.6850 - val_accuracy: 0.4828 - val_loss: 0.6831
Epoch 19/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step ep - accuracy: 0.3750 - loss: 0.685
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5870 - loss: 0.6647 - val_accuracy: 0.5172 - val_loss: 0.6837
Epoch 20/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step ep - accuracy: 0.3750 - loss: 0.703
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.5464 - loss: 0.6658 - val_accuracy: 0.5172 - val_loss: 0.6842
```



[154]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from sklearn.model_selection import train_test_split
from tensorflow.keras.callbacks import Callback


daily_data = pd.read_csv("XAUUSD_Daily_Ask_2024.01.01_2024.06.30.csv")
daily_data['Time (UTC)'] = pd.to_datetime(daily_data['Time (UTC)'])
daily_data.set_index('Time (UTC)', inplace=True)
daily_data = daily_data[['Close']]


daily_data['Target'] = (daily_data['Close'].shift(-1) > daily_data['Close']).astype(int)
daily_data.dropna(inplace=True)
daily_data['Close'] = (daily_data['Close'] - daily_data['Close'].mean()) / daily_data['Close'].std()


sequence_length = 10
X, y = [], []
for i in range(len(daily_data) - sequence_length):
    X.append(daily_data['Close'].iloc[i:i + sequence_length].values)
    y.append(daily_data['Target'].iloc[i + sequence_length])
X = np.array(X)
y = np.array(y)


X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_val = X_val.reshape((X_val.shape[0], X_val.shape[1], 1))


class AccuracyCallback(Callback):
    def __init__(self):
        self.accuracy = []

    def on_epoch_end(self, epoch, logs=None):
        self.accuracy.append(logs['val_accuracy'])
```

```
model = Sequential([
    LSTM(50, input_shape=(X_train.shape[1], X_train.shape[2]), activation='relu'),
    Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])


accuracy_callback = AccuracyCallback()
model.fit(X_train, y_train, epochs=15, batch_size=8, verbose=1, validation_data=(X_val, y_val), callbacks=[accuracy_callback])


plt.plot(accuracy_callback.accuracy)
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training accuracy graph')
plt.show()
```

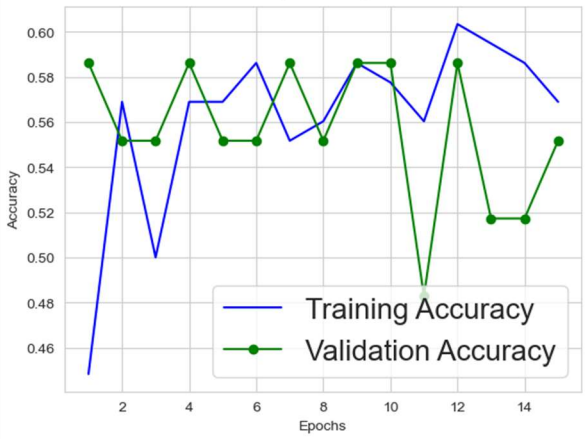
```
Epoch 1/15
C:\Users\abkha\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer i
n the model instead.

15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.5938 - loss: 0.6855 - val_accuracy: 0.5862 - val_loss: 0.6855
Epoch 2/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5126 - loss: 0.6965 - val_accuracy: 0.5862 - val_loss: 0.6850
Epoch 3/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5701 - loss: 0.6830 - val_accuracy: 0.5862 - val_loss: 0.6857
Epoch 4/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5936 - loss: 0.6724 - val_accuracy: 0.5862 - val_loss: 0.6882
Epoch 5/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5909 - loss: 0.6760 - val_accuracy: 0.5862 - val_loss: 0.6901
Epoch 6/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5815 - loss: 0.6658 - val_accuracy: 0.5862 - val_loss: 0.6879
Epoch 7/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.4945 - loss: 0.7194 - val_accuracy: 0.5862 - val_loss: 0.6856
Epoch 8/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5954 - loss: 0.6712 - val_accuracy: 0.5862 - val_loss: 0.6863
Epoch 9/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5683 - loss: 0.6685 - val_accuracy: 0.5862 - val_loss: 0.6869
Epoch 10/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5763 - loss: 0.6741 - val_accuracy: 0.5862 - val_loss: 0.6880
Epoch 11/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6052 - loss: 0.6766 - val_accuracy: 0.5862 - val_loss: 0.6864
Epoch 12/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6343 - loss: 0.6574 - val_accuracy: 0.5862 - val_loss: 0.6868
Epoch 13/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5599 - loss: 0.6934 - val_accuracy: 0.5172 - val_loss: 0.6848
Epoch 14/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6385 - loss: 0.6571 - val_accuracy: 0.5172 - val_loss: 0.6863
Epoch 15/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6067 - loss: 0.6665 - val_accuracy: 0.5862 - val_loss: 0.6855
```



[155]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from sklearn.model_selection import train_test_split
from tensorflow.keras.callbacks import Callback


daily_data = pd.read_csv("XAUUSD_Daily_Ask_2024.01.01_2024.06.30.csv")
daily_data['Time (UTC)'] = pd.to_datetime(daily_data['Time (UTC)'])
daily_data.set_index('Time (UTC)', inplace=True)
daily_data = daily_data[['Close']]


daily_data['Target'] = (daily_data['Close'].shift(-1) > daily_data['Close']).astype(int)
daily_data.dropna(inplace=True)
daily_data['Close'] = (daily_data['Close'] - daily_data['Close'].mean()) / daily_data['Close'].std()


sequence_length = 10
X, y = [], []
for i in range(len(daily_data) - sequence_length):
    X.append(daily_data['Close'].iloc[i:i + sequence_length].values)
    y.append(daily_data['Target'].iloc[i + sequence_length])
X = np.array(X)
y = np.array(y)


X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_val = X_val.reshape((X_val.shape[0], X_val.shape[1], 1))


class AccuracyCallback(Callback):
    def __init__(self):
        self.train_accuracy = []
        self.val_accuracy = []

    def on_epoch_end(self, epoch, logs=None):
        self.train_accuracy.append(logs['accuracy'])
```

```
model = Sequential([
    LSTM(50, input_shape=(X_train.shape[1], X_train.shape[2]), activation='relu'),
    Dense(1, activation='sigmoid')  # Binary classification
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])


accuracy_callback = AccuracyCallback()
model.fit(X_train, y_train, epochs=15, batch_size=8, verbose=1, validation_data=(X_val, y_val), callbacks=[accuracy_callback])


epochs = range(1, len(accuracy_callback.train_accuracy) + 1)
plt.plot(epochs, accuracy_callback.train_accuracy, label='Training Accuracy', color='blue')
plt.plot(epochs, accuracy_callback.val_accuracy, label='Validation Accuracy', color='green', marker='o')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

```
Epoch 1/15
C:\Users\abkha\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer i
n the model instead.

15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.4457 - loss: 0.6957 - val_accuracy: 0.5862 - val_loss: 0.6903
Epoch 2/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5582 - loss: 0.6954 - val_accuracy: 0.5517 - val_loss: 0.6912
Epoch 3/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5197 - loss: 0.6895 - val_accuracy: 0.5517 - val_loss: 0.6897
Epoch 4/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5922 - loss: 0.6832 - val_accuracy: 0.5862 - val_loss: 0.6871
Epoch 5/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5641 - loss: 0.6796 - val_accuracy: 0.5517 - val_loss: 0.6866
Epoch 6/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6246 - loss: 0.6684 - val_accuracy: 0.5517 - val_loss: 0.6871
Epoch 7/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5276 - loss: 0.6891 - val_accuracy: 0.5862 - val_loss: 0.6854
Epoch 8/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5929 - loss: 0.6647 - val_accuracy: 0.5517 - val_loss: 0.6865
Epoch 9/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5970 - loss: 0.6714 - val_accuracy: 0.5862 - val_loss: 0.6871
Epoch 10/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6268 - loss: 0.6684 - val_accuracy: 0.5862 - val_loss: 0.6863
Epoch 11/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6406 - loss: 0.6488 - val_accuracy: 0.4828 - val_loss: 0.6897
Epoch 12/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6226 - loss: 0.6653 - val_accuracy: 0.5862 - val_loss: 0.6864
Epoch 13/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6236 - loss: 0.6629 - val_accuracy: 0.5172 - val_loss: 0.6869
Epoch 14/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6287 - loss: 0.6685 - val_accuracy: 0.5172 - val_loss: 0.6880
Epoch 15/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5468 - loss: 0.6777 - val_accuracy: 0.5517 - val_loss: 0.6856
```



[156]:
```python
# Calculate the prediction vector

# Verify and reshape X_test
print("Original X_test shape:", X_test.shape)
if len(X_test.shape) == 2:
    X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))
print("Reshaped X_test shape:", X_test.shape)

# Check model input shape
print("Expected input shape:", model.input_shape)

# Make predictions
pred = model.predict(X_test)
print("Predictions:", pred)
```

```
Original X_test shape: (29, 10, 1)
Reshaped X_test shape: (29, 10, 1)
Expected input shape: (None, 10, 1)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 105ms/step
Predictions: [[0.58433414]
 [0.49977234]
 [0.6387988 ]
 [0.6441331 ]
 [0.64323467]
 [0.4861143 ]
 [0.4964926 ]
 [0.49578205]
 [0.49096143]
 [0.64601076]
 [0.6439542 ]
 [0.5828674 ]
 [0.49726105]
 [0.50423884]
 [0.55051243]
 [0.6547956 ]
 [0.48645017]
 [0.50186586]
 [0.5050316 ]
 [0.5065441 ]
 [0.4976433 ]
 [0.5035812 ]
```

```
 [0.48645017]
 [0.50186586]
 [0.5050316 ]
 [0.5065441 ]
 [0.4976433 ]
 [0.5035812 ]
 [0.64251864]
 [0.5837255 ]
 [0.64924896]
 [0.6402886 ]
 [0.60773844]
 [0.5164255 ]
 [0.4884752 ]]
```

[157]:
```
print(pred)
```

```
[[0.58433414]
 [0.49977234]
 [0.6387988 ]
 [0.6441331 ]
 [0.64323467]
 [0.4861143 ]
 [0.4964926 ]
 [0.49578205]
 [0.49096143]
 [0.64601076]
 [0.6439542 ]
 [0.5828674 ]
 [0.49726105]
 [0.50423884]
 [0.55051243]
 [0.6547956 ]
 [0.48645017]
 [0.50186586]
 [0.5050316 ]
 [0.5065441 ]
 [0.4976433 ]
 [0.5035812 ]
 [0.64251864]
 [0.5837255 ]
 [0.64924896]
 [0.6402886 ]
 [0.60773844]
 [0.5164255 ]
 [0.4884752 ]]
```

[158]:
```
len(pred)
```

[158]:
```
29
```

[ ]:
```

```

[159]:
```
import random

pred = model.predict(X_test)

# Check: we take a random element random.randint() and look: what is the difference between test and predict

n_rec = random.randint(0, X_test.shape[0])
print(n_rec)


print("Predicted probability:", pred[n_rec], ", right answer:", y_test[n_rec])
```

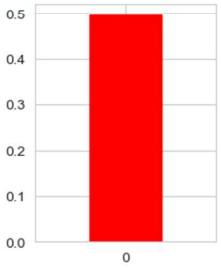
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
20
Predicted probability: [0.4976433] , right answer: 0
```

[160]:
```
classes=['0 is Flat', '1 is Trend']

index = random.randint(0, y_test.shape[0])
```

```
print('Right answer: ', y_test[index])

x = X_test[index]
x = np.expand_dims(x, axis=0)

prediction = model.predict(x)
sample = x

ans = round(float(prediction))

fig = plt.figure(figsize=(5,3))

ax = fig.add_subplot(1, 2, 2)
bar_list = ax.bar(np.arange(1), prediction[0], align='center')
if ans == y_test[index]:
    bar_list[0].set_color('g')
else:
    bar_list[0].set_color('r')

ax.set_xticks(np.arange(1))
ax.set_xlim([-1, 1])
ax.grid('on')

plt.show()
print('Predicted answer: {}'.format(classes[ans]), "\n ")
print('Right answer: {}'.format(classes[y_test[index].astype(int)]) )
print(classes)
```

```
Right answer:  1
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 154ms/step
```

```
C:\Users\abkha\AppData\Local\Temp\ipykernel_22804\2073967640.py:12: DeprecationWarning:

Conversion of an array with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you extract a single element from your array before p
erforming this operation. (Deprecated NumPy 1.25.)
```




```
Predicted answer: 0 is Flat

Right answer: 1 is Trend
['0 is Flat', '1 is Trend']
```

Week 11

Lab Logbook Requirement:

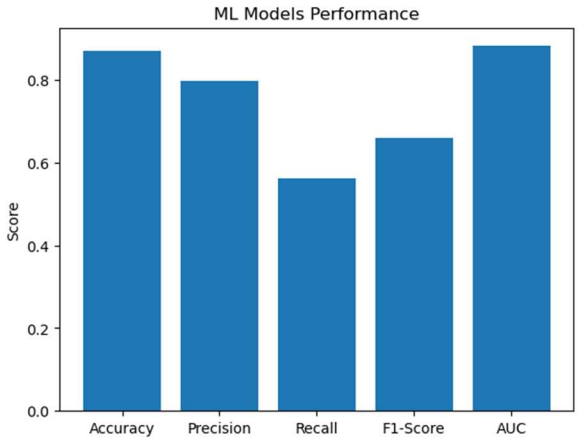
1. *Create and train your own LSTM model*
2. *Add all the LSTM's Error metrics: Accuracy, Precision, Recall, F1-Score and AUC to the final histogram "ML Models performance...".*

NOTE: DON'T FORGET TO SAVE AND BACK UP YOUR COMPLETED JUPYTER NOTEBOOK AND LAB LOGBOOK ON GITHUB OR ONEDRIVE.

[74]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
import matplotlib.pyplot as plt


credit_data = pd.read_csv("credit_risk_dataset.csv")


credit_data['person_home_ownership'] = LabelEncoder().fit_transform(credit_data['person_home_ownership'])
credit_data['loan_intent'] = LabelEncoder().fit_transform(credit_data['loan_intent'])
credit_data['cb_person_default_on_file'] = LabelEncoder().fit_transform(credit_data['cb_person_default_on_file'])


credit_data.fillna(credit_data.median(), inplace=True)


X = credit_data.drop('loan_status', axis=1).values
y = credit_data['loan_status'].values


scaler = StandardScaler()
X = scaler.fit_transform(X)


X = X.reshape((X.shape[0], 1, X.shape[1]))


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


model = Sequential([
    LSTM(50, input_shape=(X_train.shape[1], X_train.shape[2]), activation='relu'),
    Dense(1, activation='sigmoid')
])


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


history = model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=1, validation_data=(X_test, y_test))


y_pred = (model.predict(X_test) > 0.5).astype(int)
```

```python
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
auc = roc_auc_score(y_test, model.predict(X_test))


metrics = {
    'Accuracy': accuracy,
    'Precision': precision,
    'Recall': recall,
    'F1-Score': f1,
    'AUC': auc
}


plt.bar(metrics.keys(), metrics.values())
plt.title("ML Models Performance")
plt.ylabel("Score")
plt.show()
```

```
C:\Users\abkha\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer.
When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
Epoch 1/10
815/815 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.8072 - loss: 0.4953 - val_accuracy: 0.8377 - val_loss: 0.3735
Epoch 2/10
815/815 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8500 - loss: 0.3587 - val_accuracy: 0.8501 - val_loss: 0.3554
Epoch 3/10
815/815 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8633 - loss: 0.3393 - val_accuracy: 0.8584 - val_loss: 0.3469
Epoch 4/10
815/815 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8643 - loss: 0.3320 - val_accuracy: 0.8608 - val_loss: 0.3412
Epoch 5/10
815/815 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8663 - loss: 0.3305 - val_accuracy: 0.8653 - val_loss: 0.3365
Epoch 6/10
815/815 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8703 - loss: 0.3248 - val_accuracy: 0.8613 - val_loss: 0.3371
Epoch 7/10
815/815 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8681 - loss: 0.3296 - val_accuracy: 0.8670 - val_loss: 0.3310
Epoch 8/10
815/815 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8700 - loss: 0.3258 - val_accuracy: 0.8685 - val_loss: 0.3250
Epoch 9/10
815/815 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8745 - loss: 0.3150 - val_accuracy: 0.8733 - val_loss: 0.3219
Epoch 10/10
815/815 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8750 - loss: 0.3130 - val_accuracy: 0.8710 - val_loss: 0.3209
204/204 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
204/204 ━━━━━━━━━━━━━━━━━━━━ 0s 984us/step
```



[ ]: