LAB Logbook

Lab 1

Lab Logbook Requirement:

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*

[ ]:

[3]:
```
sid=23368529%100
a = np.arange(sid)  #sid = 2368529 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]]


(1, 29)
```

Lab 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.*

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

```
relationship   hours-per-week
Husband        13                1
               40                4
               45                1
               80                1
Not-in-family  16                1
               40                2
               50                2
Own-child      30                1
Wife           40                2
dtype: int64
```

```
sid = 2368529%100

def func(x):
    return x - sid

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

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

```
relationship   hours-per-week
Husband        -16               1
                11               4
                16               1
                51               1
Not-in-family  -13               1
                11               2
                21               2
Own-child       1                1
Wife            11               2
dtype: int64
```

Lab 3

Lab Logbook Requirement:

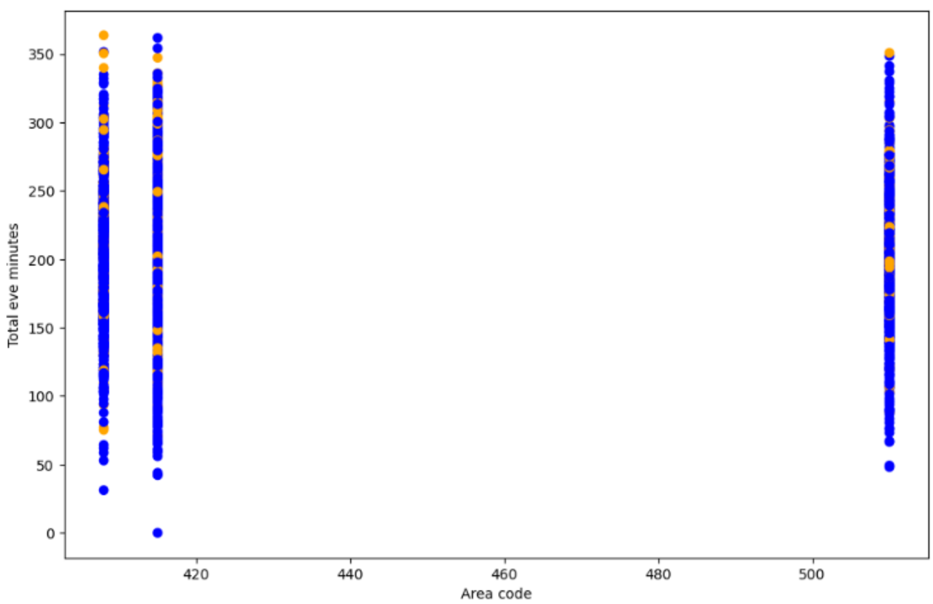
1) Draw a bicolour features interaction diagram between the columns with the numbers of the last and second to last digits of your SID, where:

```
 #   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.

[50]:
```python
#sid = 2368529 so plotted the bicolour diagram of the 2nd and 9th columns.

fig = plt.figure(figsize=(11,7))
plt.scatter(data['Area code'], data['Total eve minutes'], color = Clr);
plt.xlabel('Area code');
plt.ylabel('Total eve minutes');
```



Lab 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.*

¶

[47]:
```
model = keras.Sequential([

    keras.layers.Dense(529, input_dim = 500, activation = tf.nn.relu, kernel_initializer = "normal"),    #sid = 2368529

    keras.layers.Dense(265, activation = 'relu', kernel_initializer = "normal"),

    keras.layers.Dense(1)

])

print(model.summary())
```

Model: "sequential_4"

Layer (type)	Output Shape	Param #
dense_16 (Dense)	(None, 529)	265,029
dense_17 (Dense)	(None, 265)	140,450
dense_18 (Dense)	(None, 1)	266

```
Total params: 405,745 (1.55 MB)
Trainable params: 405,745 (1.55 MB)
Non-trainable params: 0 (0.00 B)
None
```

[48]:
```
model.compile(optimizer = "adam", loss = "mse", metrics = ["mae"])
```

[49]:
```
history = model.fit(X_train, y_train, batch_size =10, epochs = 10, validation_split = 0.2, verbose = 1)
```

```
Epoch 1/10
2640/2640 ━━━━━━━━━━━━━━━━━━━━ 20s 7ms/step - loss: 0.0356 - mae: 0.0442 - val_loss: 0.0107 - val_mae: 0.0936
Epoch 2/10
2640/2640 ━━━━━━━━━━━━━━━━━━━━ 18s 7ms/step - loss: 2.1776e-04 - mae: 0.0115 - val_loss: 0.0076 - val_mae: 0.0794
Epoch 3/10
2640/2640 ━━━━━━━━━━━━━━━━━━━━ 18s 7ms/step - loss: 1.9438e-04 - mae: 0.0108 - val_loss: 0.0048 - val_mae: 0.0614
Epoch 4/10
2640/2640 ━━━━━━━━━━━━━━━━━━━━ 18s 7ms/step - loss: 1.1215e-04 - mae: 0.0081 - val_loss: 0.0013 - val_mae: 0.0300
Epoch 5/10
2640/2640 ━━━━━━━━━━━━━━━━━━━━ 18s 7ms/step - loss: 9.0387e-05 - mae: 0.0071 - val_loss: 6.4911e-04 - val_mae: 0.0200
Epoch 6/10
2640/2640 ━━━━━━━━━━━━━━━━━━━━ 22s 7ms/step - loss: 7.6753e-05 - mae: 0.0066 - val_loss: 0.0013 - val_mae: 0.0323
Epoch 7/10
2640/2640 ━━━━━━━━━━━━━━━━━━━━ 18s 7ms/step - loss: 6.4826e-05 - mae: 0.0061 - val_loss: 0.0012 - val_mae: 0.0310
Epoch 8/10
2640/2640 ━━━━━━━━━━━━━━━━━━━━ 18s 7ms/step - loss: 5.9723e-05 - mae: 0.0058 - val_loss: 3.6299e-04 - val_mae: 0.0152
Epoch 9/10
2640/2640 ━━━━━━━━━━━━━━━━━━━━ 18s 7ms/step - loss: 5.8251e-05 - mae: 0.0058 - val_loss: 4.3948e-04 - val_mae: 0.0172
Epoch 10/10
2640/2640 ━━━━━━━━━━━━━━━━━━━━ 21s 7ms/step - loss: 5.0773e-05 - mae: 0.0055 - val_loss: 5.6606e-04 - val_mae: 0.0202
```

[50]:
```
print("Mean absolute error: %.5f" % mae)
```

```
Mean absolute error: 0.01238
```

Lab 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.*

¶

[39]:
```
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_2"

Layer (type)	Output Shape	Param #
conv1d_4 (Conv1D)	(None, 50, 50)	1,300
max_pooling1d_2 (MaxPooling1D)	(None, 7, 50)	0
conv1d_5 (Conv1D)	(None, 7, 100)	25,100
global_max_pooling1d_2 (GlobalMaxPooling1D)	(None, 100)	0
dense_4 (Dense)	(None, 25)	2,525
dense_5 (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
```

[40]:
```
model.compile(optimizer = "adam", loss = "mse", metrics = ["mae"])
```

[41]:
```
history = model.fit(X_train, y_train, batch_size =50, epochs=11, validation_split=0.2, verbose=1)

#sid = 2368529 where Z = 2 and Y = 9
```

```
Epoch 1/11
3520/3520 ━━━━━━━━━━━━━━━━━━━━ 24s 6ms/step - loss: 0.0084 - mae: 0.0454 - val_loss: 9.3309e-04 - val_mae: 0.0202
Epoch 2/11
3520/3520 ━━━━━━━━━━━━━━━━━━━━ 20s 6ms/step - loss: 7.5019e-04 - mae: 0.0186 - val_loss: 8.8981e-04 - val_mae: 0.0195
Epoch 3/11
3520/3520 ━━━━━━━━━━━━━━━━━━━━ 20s 6ms/step - loss: 7.1502e-04 - mae: 0.0181 - val_loss: 8.3847e-04 - val_mae: 0.0186
Epoch 4/11
3520/3520 ━━━━━━━━━━━━━━━━━━━━ 21s 6ms/step - loss: 6.9510e-04 - mae: 0.0177 - val_loss: 8.6578e-04 - val_mae: 0.0193
Epoch 5/11
3520/3520 ━━━━━━━━━━━━━━━━━━━━ 21s 6ms/step - loss: 7.0214e-04 - mae: 0.0177 - val_loss: 8.2659e-04 - val_mae: 0.0186
Epoch 6/11
3520/3520 ━━━━━━━━━━━━━━━━━━━━ 21s 6ms/step - loss: 6.7842e-04 - mae: 0.0175 - val_loss: 8.4197e-04 - val_mae: 0.0189
Epoch 7/11
3520/3520 ━━━━━━━━━━━━━━━━━━━━ 21s 6ms/step - loss: 6.9582e-04 - mae: 0.0176 - val_loss: 8.3462e-04 - val_mae: 0.0187
Epoch 8/11
3520/3520 ━━━━━━━━━━━━━━━━━━━━ 21s 6ms/step - loss: 7.0237e-04 - mae: 0.0176 - val_loss: 8.2371e-04 - val_mae: 0.0185
Epoch 9/11
3520/3520 ━━━━━━━━━━━━━━━━━━━━ 43s 6ms/step - loss: 6.7808e-04 - mae: 0.0175 - val_loss: 8.7874e-04 - val_mae: 0.0196
Epoch 10/11
3520/3520 ━━━━━━━━━━━━━━━━━━━━ 39s 6ms/step - loss: 6.9168e-04 - mae: 0.0175 - val_loss: 8.3649e-04 - val_mae: 0.0189
Epoch 11/11
3520/3520 ━━━━━━━━━━━━━━━━━━━━ 21s 6ms/step - loss: 6.7548e-04 - mae: 0.0174 - val_loss: 8.6348e-04 - val_mae: 0.0194
```

[42]:
```
mse,mae = model.evaluate(X_test, y_test, verbose =1)
print("Mean absolute error: %.5f" %mae)
```

```
936/936 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 0.0012 - mae: 0.0237
Mean absolute error: 0.02526
```

Lab 6

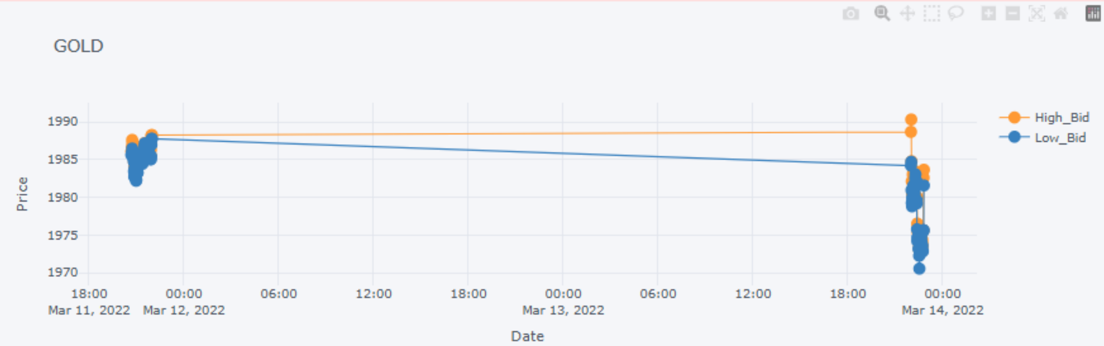
Lab Logbook Requirement:

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.*

[86]:

```
#sid = 2368529

#last 5 digits = 68529

#last 3 digits = 529 - added 129 minutes to the start point

data.iloc[68529:68658,:][['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 ')
```

```
D:\anaconda navigator\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.
```



Lab 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*

```
[50]: #sid =2368529 where Z=2 and Y=9
      #ZY +10 = 29 + 10 = 39

      model = keras.Sequential([

          keras.layers.LSTM(39, activation = 'relu', input_shape = (50, 18)),

          keras.layers.Dense(2)

      ])

      print(model.summary())
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
lstm_3 (LSTM)	(None, 39)	9,048
dense_3 (Dense)	(None, 2)	80

Total params: 9,128 (35.66 KB)
Trainable params: 9,128 (35.66 KB)
Non-trainable params: 0 (0.00 B)
None

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

```
[52]: 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)
```

```
[53]: 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
1212/1213 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.1920 - mae: 0.1231
Epoch 1: val_loss improved from inf to 0.00005, saving model to best_model_LSTM_GOLD.keras
1213/1213 ━━━━━━━━━━━━━━━━━━━━ 25s 18ms/step - loss: 0.1918 - mae: 0.1230 - val_loss: 4.9154e-05 - val_mae: 0.0049
Epoch 2/10
1212/1213 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 4.8671e-05 - mae: 0.0052
Epoch 2: val_loss improved from 0.00005 to 0.00003, saving model to best_model_LSTM_GOLD.keras
1213/1213 ━━━━━━━━━━━━━━━━━━━━ 21s 17ms/step - loss: 4.8660e-05 - mae: 0.0052 - val_loss: 2.9932e-05 - val_mae: 0.0039
Epoch 3/10
1211/1213 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 3.3265e-05 - mae: 0.0045
Epoch 3: val_loss did not improve from 0.00003
1213/1213 ━━━━━━━━━━━━━━━━━━━━ 21s 17ms/step - loss: 3.3264e-05 - mae: 0.0045 - val_loss: 6.5819e-05 - val_mae: 0.0068
Epoch 4/10
1212/1213 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 3.3284e-05 - mae: 0.0045
Epoch 4: val_loss did not improve from 0.00003
1213/1213 ━━━━━━━━━━━━━━━━━━━━ 44s 19ms/step - loss: 3.3282e-05 - mae: 0.0045 - val_loss: 6.8866e-05 - val_mae: 0.0069
Epoch 5/10
1213/1213 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 3.2081e-05 - mae: 0.0045
Epoch 5: val_loss improved from 0.00003 to 0.00002, saving model to best_model_LSTM_GOLD.keras
1213/1213 ━━━━━━━━━━━━━━━━━━━━ 22s 18ms/step - loss: 3.2080e-05 - mae: 0.0045 - val_loss: 2.0336e-05 - val_mae: 0.0033
Epoch 6/10
1212/1213 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 3.7082e-05 - mae: 0.0048
Epoch 6: val_loss did not improve from 0.00002
1213/1213 ━━━━━━━━━━━━━━━━━━━━ 21s 17ms/step - loss: 3.7076e-05 - mae: 0.0048 - val_loss: 3.4457e-05 - val_mae: 0.0047
Epoch 7/10
1213/1213 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 3.2006e-05 - mae: 0.0045
Epoch 7: val_loss did not improve from 0.00002
1213/1213 ━━━━━━━━━━━━━━━━━━━━ 19s 15ms/step - loss: 3.2006e-05 - mae: 0.0045 - val_loss: 2.4473e-05 - val_mae: 0.0044
Epoch 8/10
1211/1213 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 2.4711e-05 - mae: 0.0039
Epoch 8: val_loss improved from 0.00002 to 0.00001, saving model to best_model_LSTM_GOLD.keras
1213/1213 ━━━━━━━━━━━━━━━━━━━━ 21s 17ms/step - loss: 2.4712e-05 - mae: 0.0039 - val_loss: 1.3461e-05 - val_mae: 0.0030
Epoch 9/10
1211/1213 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 2.2413e-05 - mae: 0.0037
Epoch 9: val_loss did not improve from 0.00001
1213/1213 ━━━━━━━━━━━━━━━━━━━━ 45s 21ms/step - loss: 2.2414e-05 - mae: 0.0037 - val_loss: 4.0357e-05 - val_mae: 0.0050
Epoch 10/10
1213/1213 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 2.0940e-05 - mae: 0.0036
Epoch 10: val_loss improved from 0.00001 to 0.00001, saving model to best_model_LSTM_GOLD.keras
1213/1213 ━━━━━━━━━━━━━━━━━━━━ 24s 20ms/step - loss: 2.0939e-05 - mae: 0.0036 - val_loss: 9.3247e-06 - val_mae: 0.0026
```

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

```
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 9.4234e-06 - mae: 0.0026
```

```
[68]: scores
```

```
[68]: [8.694913049112074e-06, 0.0024753068573772907]
```

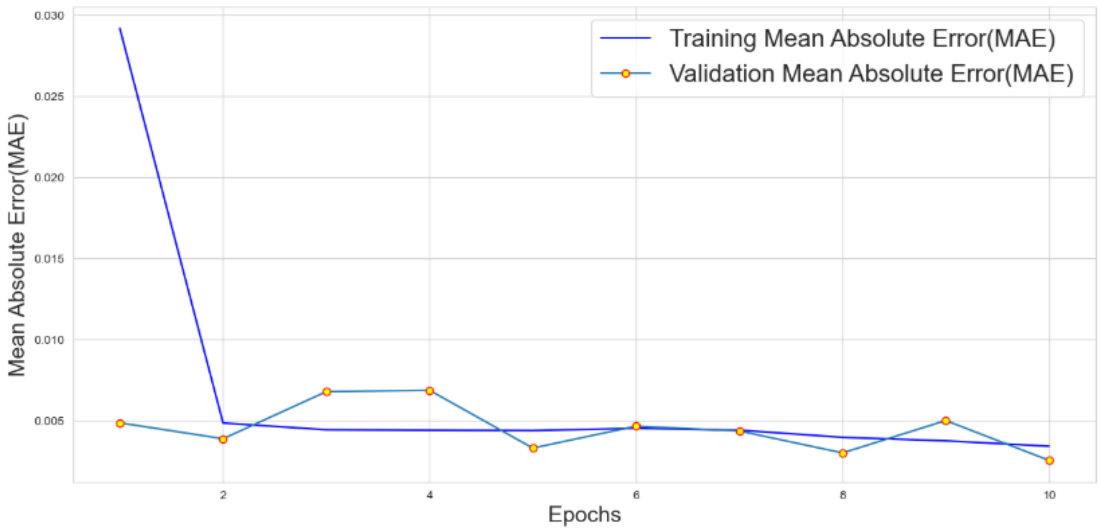
```
[69]: print("Mean squared error (mse): %.9f " % (scores[0]))
```

```
Mean squared error (mse): 0.000008695
```

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

```
Mean absolute error (mae): 0.002475307
```

```
[71]: 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()
```



<u>Lab 10</u>

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*

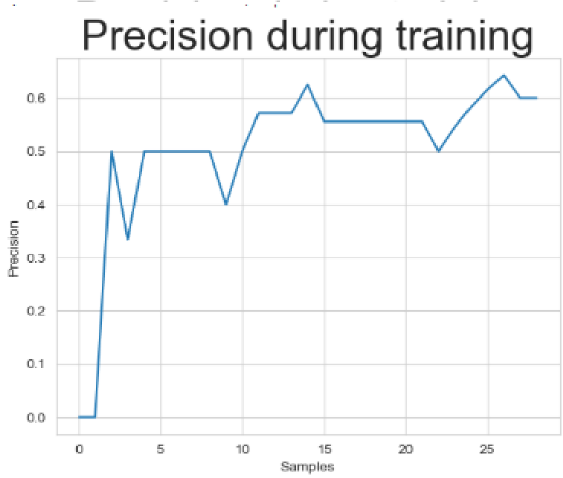
Precision during training

[280]:

```
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)
```

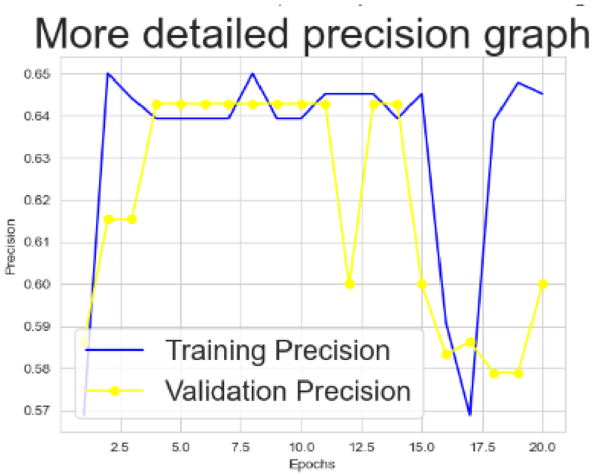
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))


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
12/12 ━━━━━━━━━━━━━━━━━━━━ 8s 100ms/step - accuracy: 0.4964 - loss: 0.6892 - val_accuracy: 0.5417 - val_loss: 0.6971
Epoch 2/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.5676 - loss: 0.6906 - val_accuracy: 0.5417 - val_loss: 0.6903
Epoch 3/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.6327 - loss: 0.6751 - val_accuracy: 0.5417 - val_loss: 0.6885
Epoch 4/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.6105 - loss: 0.6727 - val_accuracy: 0.5000 - val_loss: 0.6823
Epoch 5/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6093 - loss: 0.6765 - val_accuracy: 0.5000 - val_loss: 0.6825
Epoch 6/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5913 - loss: 0.6666 - val_accuracy: 0.4583 - val_loss: 0.6805
Epoch 7/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.6574 - loss: 0.6594 - val_accuracy: 0.4583 - val_loss: 0.6818
Epoch 8/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6020 - loss: 0.6804 - val_accuracy: 0.4583 - val_loss: 0.6837
Epoch 9/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5893 - loss: 0.6843 - val_accuracy: 0.4583 - val_loss: 0.6855
Epoch 10/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.6747 - loss: 0.6499 - val_accuracy: 0.4583 - val_loss: 0.6873
Epoch 11/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.5657 - loss: 0.6932 - val_accuracy: 0.4583 - val_loss: 0.6892
Epoch 12/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.6586 - loss: 0.6717 - val_accuracy: 0.4583 - val_loss: 0.6929
Epoch 13/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6440 - loss: 0.6404 - val_accuracy: 0.4583 - val_loss: 0.6938
Epoch 14/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6297 - loss: 0.6202 - val_accuracy: 0.4583 - val_loss: 0.6952
Epoch 15/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5458 - loss: 0.6875 - val_accuracy: 0.4583 - val_loss: 0.6987
Epoch 16/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6407 - loss: 0.6522 - val_accuracy: 0.4583 - val_loss: 0.7001
Epoch 17/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5890 - loss: 0.6839 - val_accuracy: 0.5000 - val_loss: 0.7023
Epoch 18/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6303 - loss: 0.6463 - val_accuracy: 0.5417 - val_loss: 0.7073
Epoch 19/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6384 - loss: 0.6702 - val_accuracy: 0.4583 - val_loss: 0.7147
Epoch 20/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6079 - loss: 0.6543 - val_accuracy: 0.4583 - val_loss: 0.7219
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 591ms/step
```



[282]:
```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 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):
        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
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 198ms/step - accuracy: 0.6091 - loss: 0.6
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 9s 204ms/step - accuracy: 0.5720 - loss: 0.6900 - val_accuracy: 0.5862 - val_loss: 0.6891
Epoch 2/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step p - accuracy: 0.5556 - loss: 0.685
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 72ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 46ms/step - accuracy: 0.5554 - loss: 0.6857 - val_accuracy: 0.5172 - val_loss: 0.6894
Epoch 3/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step p - accuracy: 0.5978 - loss: 0.681
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 52ms/step - accuracy: 0.5953 - loss: 0.6819 - val_accuracy: 0.5172 - val_loss: 0.6895
Epoch 4/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step p - accuracy: 0.5181 - loss: 0.688
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 45ms/step - accuracy: 0.5219 - loss: 0.6883 - val_accuracy: 0.5517 - val_loss: 0.6879
Epoch 5/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step p - accuracy: 0.5645 - loss: 0.667
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 45ms/step - accuracy: 0.5658 - loss: 0.6685 - val_accuracy: 0.5517 - val_loss: 0.6885
Epoch 6/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step p - accuracy: 0.5862 - loss: 0.671
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 47ms/step - accuracy: 0.5856 - loss: 0.6718 - val_accuracy: 0.5517 - val_loss: 0.6886
Epoch 7/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step p - accuracy: 0.5729 - loss: 0.681
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.5732 - loss: 0.6814 - val_accuracy: 0.5517 - val_loss: 0.6884
Epoch 8/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step ep - accuracy: 0.5728 - loss: 0.677
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 49ms/step - accuracy: 0.5747 - loss: 0.6776 - val_accuracy: 0.5517 - val_loss: 0.6903
Epoch 9/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step p - accuracy: 0.5767 - loss: 0.656
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 45ms/step - accuracy: 0.5767 - loss: 0.6576 - val_accuracy: 0.5517 - val_loss: 0.6909
Epoch 10/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step p - accuracy: 0.5584 - loss: 0.664
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 46ms/step - accuracy: 0.5596 - loss: 0.6654 - val_accuracy: 0.5517 - val_loss: 0.6895
Epoch 11/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step p - accuracy: 0.5833 - loss: 0.665
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.5835 - loss: 0.6662 - val_accuracy: 0.5517 - val_loss: 0.6883
Epoch 12/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step p - accuracy: 0.5861 - loss: 0.6585
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.5785 - loss: 0.6666 - val_accuracy: 0.5172 - val_loss: 0.6876
Epoch 13/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step p - accuracy: 0.5218 - loss: 0.6591
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 46ms/step - accuracy: 0.5550 - loss: 0.6667 - val_accuracy: 0.5517 - val_loss: 0.6881
Epoch 14/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step p - accuracy: 0.5251 - loss: 0.6976
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 46ms/step - accuracy: 0.5530 - loss: 0.6876 - val_accuracy: 0.5517 - val_loss: 0.6882
Epoch 15/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step p - accuracy: 0.5137 - loss: 0.6850
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 0.5407 - loss: 0.6822 - val_accuracy: 0.5172 - val_loss: 0.6868
Epoch 16/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step p - accuracy: 0.5399 - loss: 0.6563
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.5542 - loss: 0.6635 - val_accuracy: 0.5517 - val_loss: 0.6876
Epoch 17/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step p - accuracy: 0.6219 - loss: 0.668
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 67ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.6164 - loss: 0.6690 - val_accuracy: 0.5862 - val_loss: 0.6838
Epoch 18/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step p - accuracy: 0.5672 - loss: 0.6456
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 0.5487 - loss: 0.6577 - val_accuracy: 0.5172 - val_loss: 0.6852
Epoch 19/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step ep - accuracy: 0.6452 - loss: 0.654
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 49ms/step - accuracy: 0.6284 - loss: 0.6578 - val_accuracy: 0.5172 - val_loss: 0.6860
Epoch 20/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step p - accuracy: 0.5823 - loss: 0.668
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.5758 - loss: 0.6715 - val_accuracy: 0.5172 - val_loss: 0.6851
```

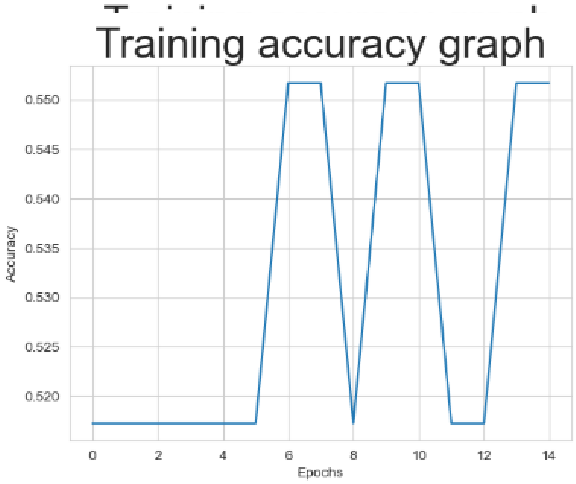


[286]:

```
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
15/15 ━━━━━━━━━━━━━━━━━━━━ 8s 86ms/step - accuracy: 0.6213 - loss: 0.6871 - val_accuracy: 0.5172 - val_loss: 0.6905
Epoch 2/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5919 - loss: 0.6855 - val_accuracy: 0.5172 - val_loss: 0.6887
Epoch 3/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5701 - loss: 0.6864 - val_accuracy: 0.5172 - val_loss: 0.6883
Epoch 4/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6151 - loss: 0.6628 - val_accuracy: 0.5172 - val_loss: 0.6899
Epoch 5/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5518 - loss: 0.6906 - val_accuracy: 0.5172 - val_loss: 0.6900
Epoch 6/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5695 - loss: 0.6979 - val_accuracy: 0.5172 - val_loss: 0.6900
Epoch 7/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.5799 - loss: 0.6797 - val_accuracy: 0.5517 - val_loss: 0.6918
Epoch 8/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.5804 - loss: 0.7042 - val_accuracy: 0.5517 - val_loss: 0.6901
Epoch 9/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.5763 - loss: 0.6813 - val_accuracy: 0.5172 - val_loss: 0.6892
Epoch 10/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.5349 - loss: 0.7019 - val_accuracy: 0.5517 - val_loss: 0.6892
Epoch 11/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5925 - loss: 0.6821 - val_accuracy: 0.5517 - val_loss: 0.6887
Epoch 12/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.5529 - loss: 0.6778 - val_accuracy: 0.5172 - val_loss: 0.6890
Epoch 13/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.5066 - loss: 0.6645 - val_accuracy: 0.5172 - val_loss: 0.6893
Epoch 14/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.6401 - loss: 0.6639 - val_accuracy: 0.5517 - val_loss: 0.6882
Epoch 15/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.5553 - loss: 0.6868 - val_accuracy: 0.5517 - val_loss: 0.6869
```

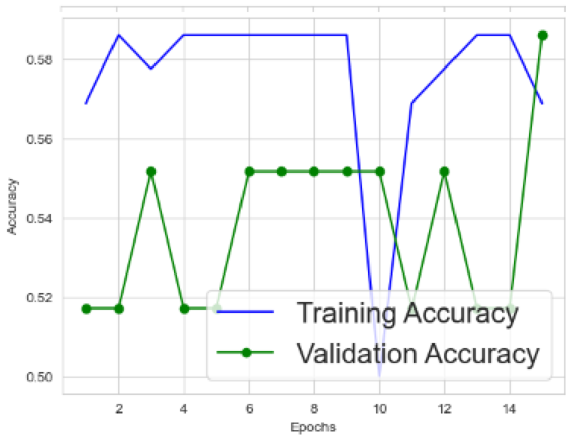


[288]:

```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 = []
```

```python
    def on_epoch_end(self, epoch, logs=None):
        self.train_accuracy.append(logs['accuracy'])
        self.val_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')  # 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
15/15 ━━━━━━━━━━━━━━━━━━━━ 9s 99ms/step - accuracy: 0.6340 - loss: 0.6865 - val_accuracy: 0.5172 - val_loss: 0.6900
Epoch 2/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5411 - loss: 0.6914 - val_accuracy: 0.5172 - val_loss: 0.6886
Epoch 3/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5361 - loss: 0.6769 - val_accuracy: 0.5517 - val_loss: 0.6877
Epoch 4/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5294 - loss: 0.6881 - val_accuracy: 0.5172 - val_loss: 0.6884
Epoch 5/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5850 - loss: 0.6820 - val_accuracy: 0.5172 - val_loss: 0.6890
Epoch 6/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.6284 - loss: 0.6660 - val_accuracy: 0.5517 - val_loss: 0.6928
Epoch 7/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.6018 - loss: 0.6597 - val_accuracy: 0.5517 - val_loss: 0.6906
Epoch 8/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.6017 - loss: 0.6935 - val_accuracy: 0.5517 - val_loss: 0.6886
Epoch 9/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.6411 - loss: 0.6592 - val_accuracy: 0.5517 - val_loss: 0.6882
Epoch 10/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5075 - loss: 0.6570 - val_accuracy: 0.5517 - val_loss: 0.6870
Epoch 11/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.5645 - loss: 0.6803 - val_accuracy: 0.5172 - val_loss: 0.6855
Epoch 12/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.5788 - loss: 0.6932 - val_accuracy: 0.5517 - val_loss: 0.6872
Epoch 13/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5461 - loss: 0.6774 - val_accuracy: 0.5172 - val_loss: 0.6867
Epoch 14/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5766 - loss: 0.6606 - val_accuracy: 0.5172 - val_loss: 0.6861
Epoch 15/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.5700 - loss: 0.6740 - val_accuracy: 0.5862 - val_loss: 0.6840
```



[336]:

```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 116ms/step
Predictions: [[0.5556013 ]
 [0.4996255 ]
 [0.6276852 ]
 [0.63105965]
 [0.6306144 ]
 [0.49330345]
 [0.5002903 ]
 [0.4950856 ]
 [0.49369597]
 [0.63282484]
 [0.63177073]
 [0.5741288 ]
 [0.5018753 ]
 [0.50046986]
 [0.5219916 ]
 [0.6392977 ]
 [0.49119493]
 [0.50059724]
 [0.50257075]
 [0.50231224]
 [0.500361  ]
 [0.50128865]
 [0.63047624]
 [0.5686562 ]
 [0.63633466]
 [0.6273813 ]
 [0.5925585 ]
 [0.5055052 ]
 [0.49101394]]
```

[296]:
```
print(pred)
```

```
[[0.5556013 ]
 [0.4996255 ]
 [0.6276852 ]
 [0.63105965]
 [0.6306144 ]
 [0.49330345]
 [0.5002903 ]
 [0.4950856 ]
 [0.49369597]
 [0.63282484]
 [0.63177073]
 [0.5741288 ]
 [0.5018753 ]
 [0.50046986]
 [0.5219916 ]
 [0.6392977 ]
 [0.49119493]
 [0.50059724]
 [0.50257075]
 [0.50231224]
 [0.500361  ]
 [0.50128865]
 [0.63047624]
 [0.5686562 ]
 [0.63633466]
 [0.6273813 ]
 [0.5925585 ]
 [0.5055052 ]
 [0.49101394]]
```

[298]:
```
len(pred)
```

[298]: 29

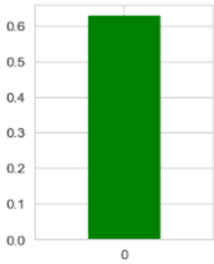
[ ]:

[308]:
```
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 83ms/step
5
Predicted probability: [0.49330345] , right answer: 1
```

```
[326]: 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 80ms/step
```




```
Predicted answer: 1 is Trend

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

Lab 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...".*

[152]:
```
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)


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()
```

```
Epoch 1/10
815/815 ━━━━━━━━━━━━━━━━━━━━ 18s 7ms/step - accuracy: 0.7968 - loss: 0.4926 - val_accuracy: 0.8373 - val_loss: 0.3728
Epoch 2/10
815/815 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.8531 - loss: 0.3549 - val_accuracy: 0.8558 - val_loss: 0.3509
Epoch 3/10
815/815 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.8613 - loss: 0.3392 - val_accuracy: 0.8599 - val_loss: 0.3430
Epoch 4/10
815/815 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.8677 - loss: 0.3317 - val_accuracy: 0.8601 - val_loss: 0.3385
Epoch 5/10
815/815 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.8646 - loss: 0.3358 - val_accuracy: 0.8665 - val_loss: 0.3320
Epoch 6/10
815/815 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.8702 - loss: 0.3224 - val_accuracy: 0.8690 - val_loss: 0.3287
Epoch 7/10
815/815 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - accuracy: 0.8682 - loss: 0.3284 - val_accuracy: 0.8668 - val_loss: 0.3277
Epoch 8/10
815/815 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.8719 - loss: 0.3188 - val_accuracy: 0.8700 - val_loss: 0.3238
Epoch 9/10
815/815 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - accuracy: 0.8742 - loss: 0.3157 - val_accuracy: 0.8682 - val_loss: 0.3220
Epoch 10/10
815/815 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.8783 - loss: 0.3084 - val_accuracy: 0.8752 - val_loss: 0.3207
204/204 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step
204/204 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
```