Week 1

Github - https://github.com/Abkhadar/m-l-in-finance/tree/main

Insert a cell below (B)

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*

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

Github - https://github.com/Abkhadar/m-l-in-finance/tree/main

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

Github - https://github.com/Abkhadar/m-l-in-finance/tree/main

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:

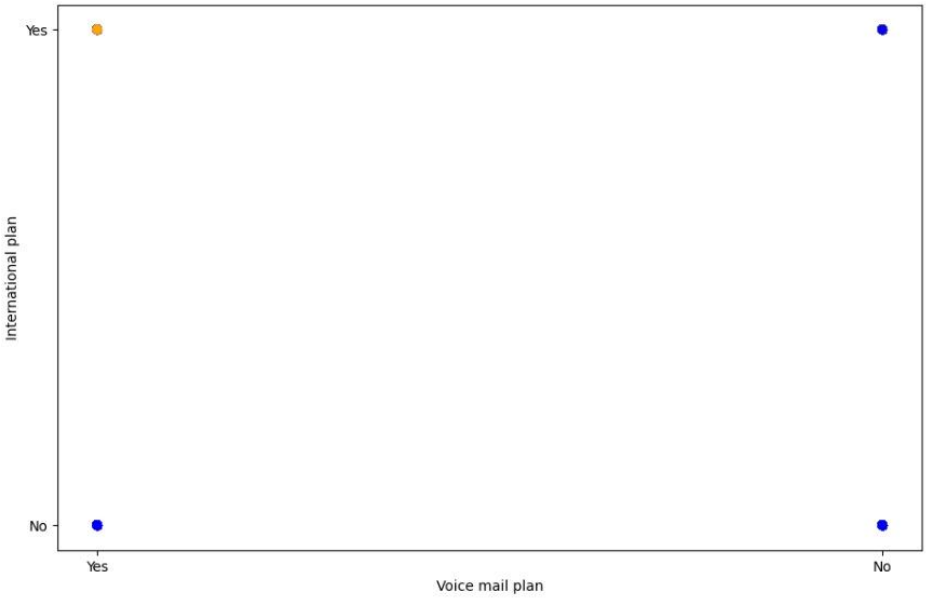
```
 #   Column
---  ------

 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]:
```python
#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

Github - https://github.com/Abkhadar/m-l-in-finance/tree/main

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

Github - https://github.com/Abkhadar/m-l-in-finance/tree/main

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

Github - https://github.com/Abkhadar/m-l-in-finance/tree/main

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.*

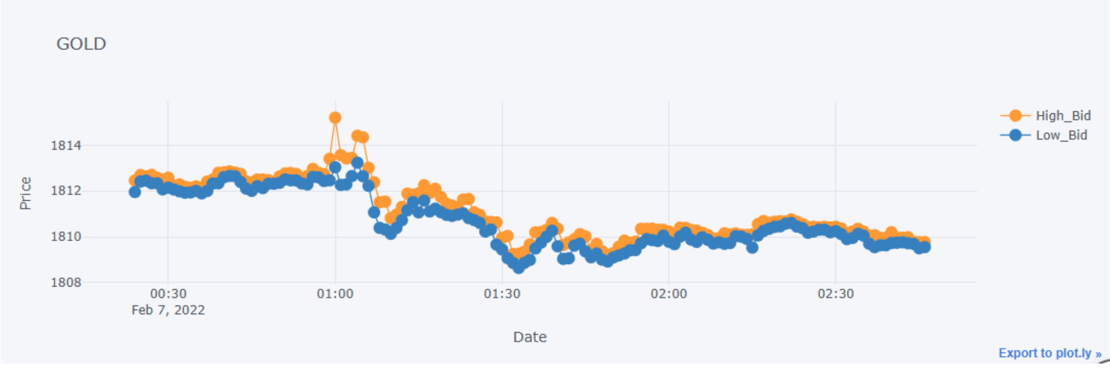
NOTE: DON'T FORGET TO SAVE AND BACK UP YOUR COMPLETED JUPYTER NOTEBOOK AND LAB LOGBOOK ON GITHUB OR ONEDRIVE.

[ ]:

[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

Github - https://github.com/Abkhadar/m-l-in-finance/tree/main

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]))
```

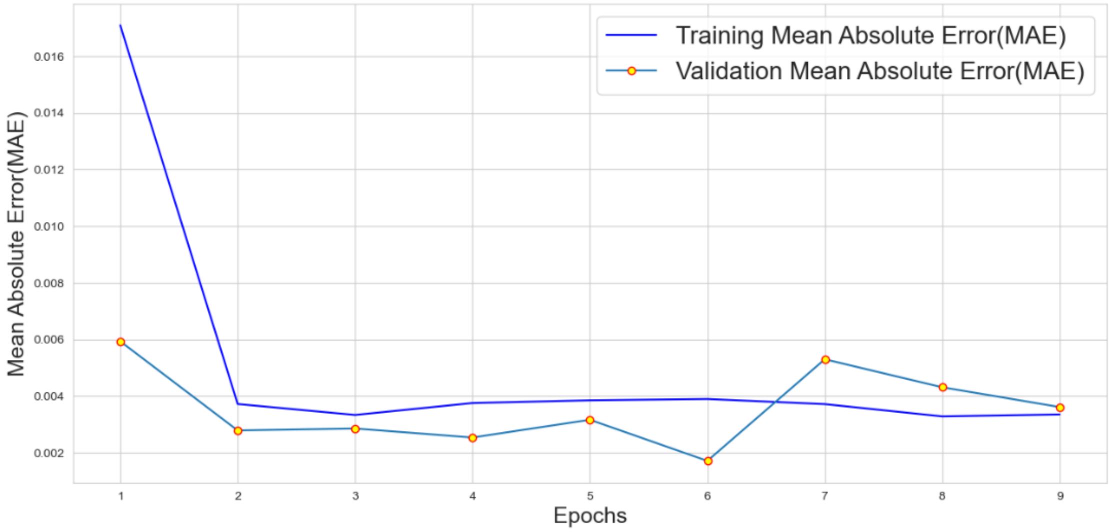
```
Mean squared error (mse): 0.000006355
```

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

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

```
Mean absolute error (mae): [illegible]
```

[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 – 8

Github - https://github.com/Abkhadar/m-l-in-finance/tree/main

WEEK-8 Mock Test

[136]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
%matplotlib inline

from matplotlib import *
import seaborn as sns

import random

import tensorflow as tf
from tensorflow import keras
```

[137]:
```
print(tf.__version__)
```

```
2.18.0
```

[138]:
```
np.random.seed(42)
```

1. Download Ask & Bid datasets ¶

[139]:
```
df_Ask = pd.read_csv("XAGUSD_5 Mins_Ask_2023.01.01_2023.06.30.csv")
df_Bid = pd.read_csv("XAGUSD_5 Mins_Bid_2023.01.01_2023.06.30.csv")
```

[140]:
```
print(df_Ask.head(3))
print(df_Ask.tail(3))
```

```
            Time (UTC)    Open    High     Low   Close  Volume
0  2023.01.02 23:00:00  24.102  24.125  24.083  24.125   0.351
1  2023.01.02 23:05:00  24.094  24.188  24.094  24.141   1.155
2  2023.01.02 23:10:00  24.143  24.148  24.022  24.027   0.882
                Time (UTC)    Open    High     Low   Close  Volume
35217  2023.06.30 20:45:00  22.781  22.781  22.776  22.776  0.4212
35218  2023.06.30 20:50:00  22.776  22.797  22.774  22.786  0.6836
35219  2023.06.30 20:55:00  22.786  22.811  22.786  22.811  0.3712
```

[141]:
```
print(df_Bid.head(3))
print(df_Bid.tail(3))
```

```
            Time (UTC)    Open    High     Low   Close  Volume
0  2023.01.02 23:00:00  24.036  24.059  24.017  24.059  0.3480
1  2023.01.02 23:05:00  24.064  24.130  24.064  24.092  1.8458
2  2023.01.02 23:10:00  24.094  24.098  23.972  23.977  0.9030
                Time (UTC)    Open    High     Low   Close  Volume
35217  2023.06.30 20:45:00  22.751  22.751  22.746  22.746   0.069
35218  2023.06.30 20:50:00  22.746  22.761  22.736  22.756   0.216
35219  2023.06.30 20:55:00  22.756  22.766  22.701  22.745   0.327
```

[142]:
```
df_Ask.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35220 entries, 0 to 35219
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Time (UTC)  35220 non-null  object 
 1   Open        35220 non-null  float64
 2   High        35220 non-null  float64
 3   Low         35220 non-null  float64
 4   Close       35220 non-null  float64
 5   Volume      35220 non-null  float64
dtypes: float64(5), object(1)
```

[143]:
```
df_Bid.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35220 entries, 0 to 35219
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Time (UTC)  35220 non-null  object 
 1   Open        35220 non-null  float64
 2   High        35220 non-null  float64
 3   Low         35220 non-null  float64
 4   Close       35220 non-null  float64
 5   Volume      35220 non-null  float64
dtypes: float64(5), object(1)
memory usage: 1.6+ MB
```

2. Merge Ask & Bid datasets

[144]:
```
df_2023 = df_Bid.merge(df_Ask, left_on = 'Time (UTC)', right_on = 'Time (UTC)', how = 'outer')
df_2023
```

[144]:

	Time (UTC)	Open_x	High_x	Low_x	Close_x	Volume _x	Open_y	High_y	Low_y	Close_y	Volume _y
0	2023.01.02 23:00:00	24.036	24.059	24.017	24.059	0.3480	24.102	24.125	24.083	24.125	0.3510
1	2023.01.02 23:05:00	24.064	24.130	24.064	24.092	1.8458	24.094	24.188	24.094	24.141	1.1550
2	2023.01.02 23:10:00	24.094	24.098	23.972	23.977	0.9030	24.143	24.148	24.022	24.027	0.8820
3	2023.01.02 23:15:00	23.977	23.980	23.938	23.980	0.8940	24.026	24.028	23.986	24.028	0.8940
4	2023.01.02 23:20:00	23.978	24.024	23.976	24.023	0.8880	24.026	24.073	24.023	24.073	0.9044
...	...	...	...	...	...	...	...	...	...	...	...
35215	2023.06.30 20:35:00	22.752	22.752	22.741	22.746	0.2752	22.782	22.782	22.771	22.776	1.8002
35216	2023.06.30 20:40:00	22.736	22.751	22.736	22.746	0.1410	22.775	22.781	22.775	22.776	0.7568
35217	2023.06.30 20:45:00	22.751	22.751	22.746	22.746	0.0690	22.781	22.781	22.776	22.776	0.4212
35218	2023.06.30 20:50:00	22.746	22.761	22.736	22.756	0.2160	22.776	22.797	22.774	22.786	0.6836
35219	2023.06.30 20:55:00	22.756	22.766	22.701	22.745	0.3270	22.786	22.811	22.786	22.811	0.3712

35220 rows × 11 columns

[145]:
```
df_2023.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35220 entries, 0 to 35219
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Time (UTC)  35220 non-null  object 
 1   Open_x      35220 non-null  float64
 2   High_x      35220 non-null  float64
 3   Low_x       35220 non-null  float64
 4   Close_x     35220 non-null  float64
 5   Volume _x   35220 non-null  float64
 6   Open_y      35220 non-null  float64
 7   High_y      35220 non-null  float64
 8   Low_y       35220 non-null  float64
 9   Close_y     35220 non-null  float64
 10  Volume _y   35220 non-null  float64
dtypes: float64(10), object(1)
memory usage: 3.0+ MB
```

[146]:
```
# rename columns

df_2023.columns = ['Local time', 'Open_Bid', 'High_Bid', 'Low_Bid', 'Close_Bid', 'Volume_Bid',
                   'Open_Ask', 'High_Ask', 'Low_Ask', 'Close_Ask', 'Volume_Ask']
```

[147]:
```
df_2023.head(3)
```

[147]:

	Local time	Open_Bid	High_Bid	Low_Bid	Close_Bid	Volume_Bid	Open_Ask	High_Ask	Low_Ask	Close_Ask	Volume_Ask
0	2023.01.02 23:00:00	24.036	24.059	24.017	24.059	0.3480	24.102	24.125	24.083	24.125	0.351
1	2023.01.02 23:05:00	24.064	24.130	24.064	24.092	1.8458	24.094	24.188	24.094	24.141	1.155
2	2023.01.02 23:10:00	24.094	24.098	23.972	23.977	0.9030	24.143	24.148	24.022	24.027	0.882

[148]:
```
df_2023.tail(3)
```

[148]:

	Local time	Open_Bid	High_Bid	Low_Bid	Close_Bid	Volume_Bid	Open_Ask	High_Ask	Low_Ask	Close_Ask	Volume_Ask
35217	2023.06.30 20:45:00	22.751	22.751	22.746	22.746	0.069	22.781	22.781	22.776	22.776	0.4212
35218	2023.06.30 20:50:00	22.746	22.761	22.736	22.756	0.216	22.776	22.797	22.774	22.786	0.6836
35219	2023.06.30 20:55:00	22.756	22.766	22.701	22.745	0.327	22.786	22.811	22.786	22.811	0.3712

[149]:
```
file_obj2 = open('df_2023.csv', 'w')
df_2023.to_csv('df_2023.csv', encoding = 'utf-8', index = False)
file_obj2.close()
```

[150]:
```
df_2023 = []
df_2023
```

[150]: []

[151]:
```
df = pd.read_csv('df_2023.csv', low_memory=False, sep=',')
```

[152]:
```
df["Volume_Delta"] = df["Volume_Ask"] - df["Volume_Bid"]
df["Volume_Delta_abs"] = (df["Volume_Ask"] - df["Volume_Bid"]).abs()
```

[153]:
```
df["Open_Delta"] = df["Open_Ask"]  - df["Open_Bid"]
df["High_Delta"] = df["High_Ask"]  - df["High_Bid"]
df["Low_Delta"]  = df["Low_Ask"]   - df["Low_Bid"]
df["Close_Delta"]= df["Close_Ask"] - df["Close_Bid"]
```

[154]:
```
df.head(3)
```

[154]:

	Local time	Open_Bid	High_Bid	Low_Bid	Close_Bid	Volume_Bid	Open_Ask	High_Ask	Low_Ask	Close_Ask	Volume_Ask	Volume_Delta	Volume_Delta_abs	Open_
0	2023.01.02 23:00:00	24.036	24.059	24.017	24.059	0.3480	24.102	24.125	24.083	24.125	0.351	0.0030	0.0030	
1	2023.01.02 23:05:00	24.064	24.130	24.064	24.092	1.8458	24.094	24.188	24.094	24.141	1.155	-0.6908	0.6908	
2	2023.01.02 23:10:00	24.094	24.098	23.972	23.977	0.9030	24.143	24.148	24.022	24.027	0.882	-0.0210	0.0210	

[155]:
```
df.tail(3)
```

[155]:

	Local time	Open_Bid	High_Bid	Low_Bid	Close_Bid	Volume_Bid	Open_Ask	High_Ask	Low_Ask	Close_Ask	Volume_Ask	Volume_Delta	Volume_Delta_abs	(
35217	2023.06.30 20:45:00	22.751	22.751	22.746	22.746	0.069	22.781	22.781	22.776	22.776	0.4212	0.3522	0.3522	

35218	2023.06.30 20:50:00	22.746	22.761	22.736	22.756	0.216	22.776	22.797	22.774	22.786	0.6836	0.4676	0.4676
35219	2023.06.30 20:55:00	22.756	22.766	22.701	22.745	0.327	22.786	22.811	22.786	22.811	0.3712	0.0442	0.0442

[156]:
```
df.describe()
```

[156]:

	Open_Bid	High_Bid	Low_Bid	Close_Bid	Volume_Bid	Open_Ask	High_Ask	Low_Ask	Close_Ask	Volume_Ask	Volume_Delta
count	35220.000000	35220.000000	35220.000000	35220.000000	35220.000000	35220.000000	35220.000000	35220.000000	35220.000000	35220.000000	35220.000000
mean	23.357365	23.374611	23.338437	23.357367	8.283187	23.387577	23.404824	23.368703	23.387501	35.095338	26.812151
std	1.368781	1.368825	1.368926	1.368867	8.976407	1.368779	1.368797	1.368944	1.368861	39.024921	33.422609
min	19.893000	19.914000	19.888000	19.893000	0.000000	19.923000	19.944000	19.918000	19.923000	0.000000	-48.791200
25%	22.421750	22.440000	22.400000	22.421000	2.611050	22.452000	22.470000	22.430000	22.451000	9.467675	5.170550
50%	23.560000	23.576000	23.543000	23.560000	5.331100	23.590000	23.607000	23.573000	23.590000	22.237450	14.907800
75%	24.122000	24.144000	24.103000	24.122000	10.686275	24.153000	24.174000	24.133000	24.152000	46.639375	36.154925
max	26.118000	26.123000	26.098000	26.118000	120.651700	26.148000	26.153000	26.128000	26.148000	365.632200	317.585800

[157]:
```
data = df.drop(['Open_Ask', 'High_Ask', 'Low_Ask', 'Close_Ask'], axis = 1)
```

[158]:
```
data.shape
```

[158]:
```
(35220, 13)
```

[159]:
```
data.head(3)
```

[159]:

	Local time	Open_Bid	High_Bid	Low_Bid	Close_Bid	Volume_Bid	Volume_Ask	Volume_Delta	Volume_Delta_abs	Open_Delta	High_Delta	Low_Delta	Close_Delta
0	2023.01.02 23:00:00	24.036	24.059	24.017	24.059	0.3480	0.351	0.0030	0.0030	0.066	0.066	0.066	0.066
1	2023.01.02 23:05:00	24.064	24.130	24.064	24.092	1.8458	1.155	-0.6908	0.6908	0.030	0.058	0.030	0.049
2	2023.01.02 23:10:00	24.094	24.098	23.972	23.977	0.9030	0.882	-0.0210	0.0210	0.049	0.050	0.050	0.050

[160]:
```
data.tail(3)
```

[160]:

	Local time	Open_Bid	High_Bid	Low_Bid	Close_Bid	Volume_Bid	Volume_Ask	Volume_Delta	Volume_Delta_abs	Open_Delta	High_Delta	Low_Delta	Close_D
35217	2023.06.30 20:45:00	22.751	22.751	22.746	22.746	0.069	0.4212	0.3522	0.3522	0.03	0.030	0.030	0
35218	2023.06.30 20:50:00	22.746	22.761	22.736	22.756	0.216	0.6836	0.4676	0.4676	0.03	0.036	0.038	0
35219	2023.06.30 20:55:00	22.756	22.766	22.701	22.745	0.327	0.3712	0.0442	0.0442	0.03	0.045	0.085	0

[161]:
```
import datetime
```

[162]:
```
data['Local_time_T'] = pd.to_datetime(data['Local time'], utc = True)
```

[162]:
```
data['Local_time_T'] = pd.to_datetime(data['Local time'], utc = True)
```

[163]:
```
data = data.drop(['Local time'], axis = 1)
```

[164]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35220 entries, 0 to 35219
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Open_Bid          35220 non-null  float64
 1   High_Bid          35220 non-null  float64
 2   Low_Bid           35220 non-null  float64
 3   Close_Bid         35220 non-null  float64
 4   Volume_Bid        35220 non-null  float64
 5   Volume_Ask        35220 non-null  float64
 6   Volume_Delta      35220 non-null  float64
 7   Volume_Delta_abs  35220 non-null  float64
 8   Open_Delta        35220 non-null  float64
 9   High_Delta        35220 non-null  float64
 10  Low_Delta         35220 non-null  float64
 11  Close_Delta       35220 non-null  float64
 12  Local_time_T      35220 non-null  datetime64[ns, UTC]
dtypes: datetime64[ns, UTC](1), float64(12)
memory usage: 3.5 MB
```

[165]:
```
data.dtypes
```

[165]:
```
Open_Bid                        float64
High_Bid                        float64
Low_Bid                         float64
Close_Bid                       float64
Volume_Bid                      float64
Volume_Ask                      float64
Volume_Delta                    float64
Volume_Delta_abs                float64
Open_Delta                      float64
High_Delta                      float64
Low_Delta                       float64
Close_Delta                     float64
Local_time_T        datetime64[ns, UTC]
dtype: object
```

[166]:
```
data.head(3)
```

[166]:

	Open_Bid	High_Bid	Low_Bid	Close_Bid	Volume_Bid	Volume_Ask	Volume_Delta	Volume_Delta_abs	Open_Delta	High_Delta	Low_Delta	Close_Delta	Local_tim
0	24.036	24.059	24.017	24.059	0.3480	0.351	0.0030	0.0030	0.066	0.066	0.066	0.066	2023-01 23:00:00+0C
1	24.064	24.130	24.064	24.092	1.8458	1.155	-0.6908	0.6908	0.030	0.058	0.030	0.049	2023-01 23:05:00+0C
2	24.094	24.098	23.972	23.977	0.9030	0.882	-0.0210	0.0210	0.049	0.050	0.050	0.050	2023-01 23:10:00+0C

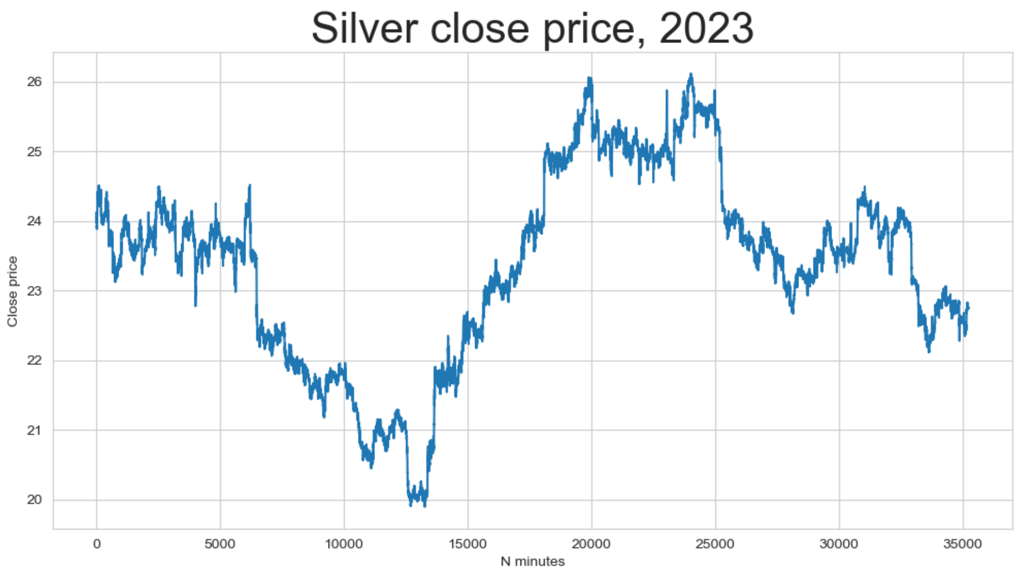
[167]:
```
data.tail(3)
```

[167]:

	Open_Bid	High_Bid	Low_Bid	Close_Bid	Volume_Bid	Volume_Ask	Volume_Delta	Volume_Delta_abs	Open_Delta	High_Delta	Low_Delta	Close_Delta	Loca
35217	22.751	22.751	22.746	22.746	0.069	0.4212	0.3522	0.3522	0.03	0.030	0.030	0.030	202 20:45:C
35218	22.746	22.761	22.736	22.756	0.216	0.6836	0.4676	0.4676	0.03	0.036	0.038	0.030	202 20:50:C
35219	22.756	22.766	22.701	22.745	0.327	0.3712	0.0442	0.0442	0.03	0.045	0.085	0.066	202 20:55:C

3. Plot Price & Volume charts

[168]:
```
plt.figure(figsize=(12,6))
plt.plot(data['Close_Bid'])
plt.title('Silver close price, 2023')
plt.xlabel('N minutes')
plt.ylabel('Close price')
plt.show()
```

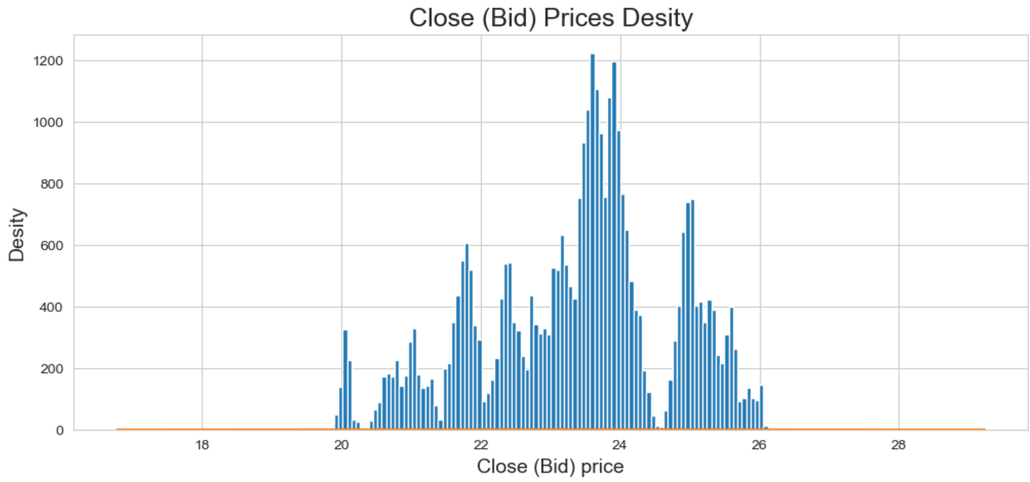


[169]:
```
plt.figure(num=1,figsize=(12,5))
plt.hist(data['Close_Bid'],bins=100)
plt.title('Close (Bid) price distribution',size=18)
plt.ylabel('Numbers',size=14)
plt.xlabel('Close (Bid) price',size=14);
```

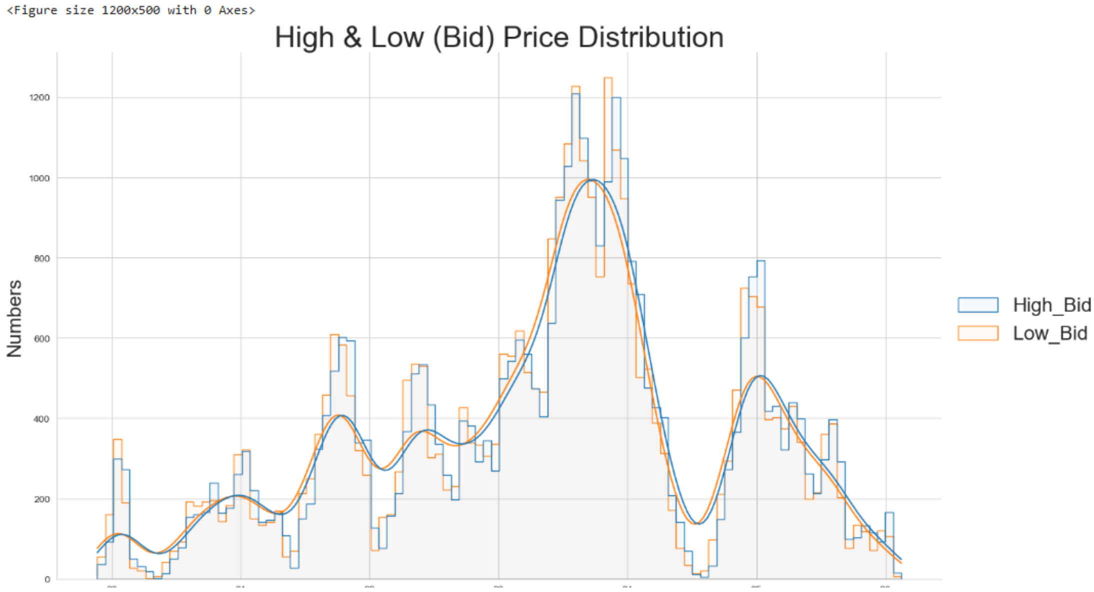
[170]:
```
plt.figure(num=1,figsize=(10,5))
data['Close_Bid'].plot.kde()
plt.title('Close (Bid) Prices Desity',size=18)
plt.ylabel('Desity',size=14)
plt.xlabel('Close (Bid) price',size=14);
```

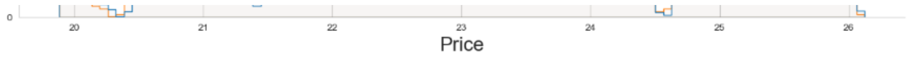
[171]:
```
fig = plt.figure(figsize=(12, 5))
plt.rc('axes', titlesize= 30 )
sns.set_style('whitegrid')
sns.set_context(rc={'legend.fontsize': 20.0})

sns.displot(
            data[['High_Bid', 'Low_Bid']],
            height=8,
            aspect=1.7,
            #hue="species",
            kde=True,
            element="step",
            bins=100,
```

```
            alpha=0.03,

)

plt.title('High & Low (Bid) Price Distribution')
plt.xlabel('Price', size= 20)
#plt.ylabel('count (%)')
plt.ylabel('Numbers', size= 20)
plt.show();
```




```
<Figure size 1200x500 with 0 Axes>
```





[172]:
```
plt.figure(num=1,figsize=(12,5))
plt.hist(data['Volume_Bid'],bins=100)
plt.title('Volume (Bid) distribution',size=18)
plt.ylabel('Numbers',size=14)
plt.xlabel('Volume (Bid)',size=14);
```

[173]:
```
plt.figure(num=1,figsize=(12,5))
plt.hist(data['Volume_Ask'],bins=100)
plt.title('Volume (Ask) distribution',size=18)
plt.ylabel('Numbers',size=14)
plt.xlabel('Volume (Ask)',size=14);
```

4. Normalisation

[174]:
```
data.shape
```

[174]:
```
(35220, 13)
```

[175]:
```
data2 = data.drop(['Local_time_T'],axis =1)
```

[176]:
```
data2.head(3)
```

[176]:

	Open_Bid	High_Bid	Low_Bid	Close_Bid	Volume_Bid	Volume_Ask	Volume_Delta	Volume_Delta_abs	Open_Delta	High_Delta	Low_Delta	Close_Delta
0	24.036	24.059	24.017	24.059	0.3480	0.351	0.0030	0.0030	0.066	0.066	0.066	0.066
1	24.064	24.130	24.064	24.092	1.8458	1.155	-0.6908	0.6908	0.030	0.058	0.030	0.049
2	24.094	24.098	23.972	23.977	0.9030	0.882	-0.0210	0.0210	0.049	0.050	0.050	0.050

[177]:
```
data2.tail(3)
```

[177]:

	Open_Bid	High_Bid	Low_Bid	Close_Bid	Volume_Bid	Volume_Ask	Volume_Delta	Volume_Delta_abs	Open_Delta	High_Delta	Low_Delta	Close_Delta
35217	22.751	22.751	22.746	22.746	0.069	0.4212	0.3522	0.3522	0.03	0.030	0.030	0.030
35218	22.746	22.761	22.736	22.756	0.216	0.6836	0.4676	0.4676	0.03	0.036	0.038	0.030
35219	22.756	22.766	22.701	22.745	0.327	0.3712	0.0442	0.0442	0.03	0.045	0.085	0.066

[178]:
```
data2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35220 entries, 0 to 35219
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Open_Bid          35220 non-null  float64
 1   High_Bid          35220 non-null  float64
 2   Low_Bid           35220 non-null  float64
 3   Close_Bid         35220 non-null  float64
 4   Volume_Bid        35220 non-null  float64
 5   Volume_Ask        35220 non-null  float64
 6   Volume_Delta      35220 non-null  float64
 7   Volume_Delta_abs  35220 non-null  float64
 8   Open_Delta        35220 non-null  float64
 9   High_Delta        35220 non-null  float64
 10  Low_Delta         35220 non-null  float64
 11  Close_Delta       35220 non-null  float64
dtypes: float64(12)
memory usage: 3.2 MB
```

[179]:
```
data2['Y_High_Bid'] = data2['High_Bid']
data2['Y_Low_Ask'] = data2['Low_Bid'] + data2['Low_Delta']
```

[180]:
```
data2.tail()
```

[180]:

	Open_Bid	High_Bid	Low_Bid	Close_Bid	Volume_Bid	Volume_Ask	Volume_Delta	Volume_Delta_abs	Open_Delta	High_Delta	Low_Delta	Close_Delta	Y_High
35215	22.752	22.752	22.741	22.746	0.2752	1.8002	1.5250	1.5250	0.030	0.030	0.030	0.030	2.
35216	22.736	22.751	22.736	22.746	0.1410	0.7568	0.6158	0.6158	0.039	0.030	0.039	0.030	2.
35217	22.751	22.751	22.746	22.746	0.0690	0.4212	0.3522	0.3522	0.030	0.030	0.030	0.030	2.
35218	22.746	22.761	22.736	22.756	0.2160	0.6836	0.4676	0.4676	0.030	0.036	0.038	0.030	2.
35219	22.756	22.766	22.701	22.745	0.3270	0.3712	0.0442	0.0442	0.030	0.045	0.085	0.066	2.

[181]:
```
data_length = len(data)
data_length
```

[181]: 35220

[182]:
```
train_size = int(round(data_length*0.8, -3))
train_size
```

[182]: 28000

[183]:
```
train =data2.iloc[:train_size]
train.shape
```

[183]: (28000, 14)

[184]:
```
train.tail(2)
```

[184]:

	Open_Bid	High_Bid	Low_Bid	Close_Bid	Volume_Bid	Volume_Ask	Volume_Delta	Volume_Delta_abs	Open_Delta	High_Delta	Low_Delta	Close_Delta	Y_High
27998	23.049	23.091	23.049	23.067	4.7839	34.8313	30.0474	30.0474	0.03	0.03	0.03	0.03	2
27999	23.074	23.080	23.029	23.039	5.0757	27.9602	22.8845	22.8845	0.03	0.03	0.03	0.03	2

[185]:
```
max_price = (train['High_Bid'] + train['High_Delta']).max()
max_price
```

[185]: 26.153

[186]:
```
min_price = train['Low_Bid'].min()
min_price
```

[186]: 19.888

[187]:
```
max_volume = max(max(train['Volume_Bid']), max(train['Volume_Ask']))
max_volume
```

[187]: 365.6322

[188]:
```
max_Delta = max(max(train['Open_Delta']), max(train['High_Delta']), max(train['Low_Delta']), max(train['Close_Delta']))
max_Delta
```

[188]: 0.30900000000000105

[189]:
```
max_Delta = round(max_Delta,3)
max_Delta
```

[189]: 0.309

[190]:
```
min_Delta = min(min(train['Open_Delta']), min(train['High_Delta']), min(train['Low_Delta']), min(train['Close_Delta']))
min_Delta
```

[190]: 0.0009999999999976694

[191]:
```
min_Delta = round(min_Delta,3)
min_Delta
```

[191]: 0.001

[192]:
```
max_volume_Delta = train['Volume_Delta'].max()
max_volume_Delta
```

[192]: 317.5858

[193]:
```
min_volume_Delta = train['Volume_Delta'].min()
min_volume_Delta
```

[193]: -48.7912

[194]:
```
max_volume_Delta_abs = train['Volume_Delta_abs'].max()
max_volume_Delta_abs
```

[194]: 317.5858

[195]:
```
min_volume_Delta_abs = train['Volume_Delta_abs'].min()
min_volume_Delta_abs
```

[195]: 0.0

[196]:
```
data2['Open_Bid'] = ( data2['Open_Bid'] - min_price ) / (max_price-min_price)
data2['High_Bid']  = ( data2['High_Bid']  - min_price ) / (max_price-min_price)
data2['Low_Bid']   = ( data2['Low_Bid']   - min_price ) / (max_price-min_price)
data2['Close_Bid'] = ( data2['Close_Bid'] - min_price ) / (max_price-min_price)
data2['Y_High_Bid'] = ( data2['Y_High_Bid'] - min_price ) / (max_price-min_price)
data2['Y_Low_Ask']  = ( data2['Y_Low_Ask']  - min_price ) / (max_price-min_price)
```

[197]:
```
data2['Volume_Ask']  = data2['Volume_Ask'] / max_volume
data2['Volume_Bid']  = data2['Volume_Bid'] / max_volume
```

[198]:
```
data2['Volume_Delta'] = ( data2['Volume_Delta'] - min_volume_Delta ) / (max_volume_Delta-min_volume_Delta)
```

[199]:
```
data2['Volume_Delta_abs']  = data2['Volume_Delta_abs'] / max_volume_Delta_abs
```

[200]:
```
data2['Open_Delta']  = ( max_Delta - data2['Open_Delta'] )  / (max_Delta-min_Delta)
data2['High_Delta']  = ( max_Delta - data2['High_Delta'] )  / (max_Delta-min_Delta)
data2['Low_Delta']   = ( max_Delta - data2['Low_Delta'] )   / (max_Delta-min_Delta)
data2['Close_Delta'] = ( max_Delta - data2['Close_Delta'] ) / (max_Delta-min_Delta)
```

[201]:
```
data2.head()
```

[201]:

	Open_Bid	High_Bid	Low_Bid	Close_Bid	Volume_Bid	Volume_Ask	Volume_Delta	Volume_Delta_abs	Open_Delta	High_Delta	Low_Delta	Close_Delta	Y_High_Bid
0	0.662091	0.665762	0.659058	0.665762	0.000952	0.000960	0.133180	0.000009	0.788961	0.788961	0.788961	0.788961	0.665762
1	0.666560	0.677095	0.666560	0.671030	0.005048	0.003159	0.131287	0.002175	0.905844	0.814935	0.905844	0.844156	0.677095
2	0.671349	0.671987	0.651875	0.652674	0.002470	0.002412	0.133115	0.000066	0.844156	0.840909	0.840909	0.840909	0.671987
3	0.652674	0.653152	0.646449	0.653152	0.002445	0.002445	0.133172	0.000000	0.844156	0.847403	0.847403	0.847403	0.653152
4	0.652833	0.660176	0.652514	0.660016	0.002429	0.002474	0.133217	0.000052	0.847403	0.844156	0.850649	0.840909	0.660176

[202]:
```
data2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35220 entries, 0 to 35219
Data columns (total 14 columns):
```

```
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Open_Bid          35220 non-null  float64
 1   High_Bid          35220 non-null  float64
 2   Low_Bid           35220 non-null  float64
 3   Close_Bid         35220 non-null  float64
 4   Volume_Bid        35220 non-null  float64
 5   Volume_Ask        35220 non-null  float64
 6   Volume_Delta      35220 non-null  float64
 7   Volume_Delta_abs  35220 non-null  float64
 8   Open_Delta        35220 non-null  float64
 9   High_Delta        35220 non-null  float64
 10  Low_Delta         35220 non-null  float64
 11  Close_Delta       35220 non-null  float64
 12  Y_High_Bid        35220 non-null  float64
 13  Y_Low_Ask         35220 non-null  float64
dtypes: float64(14)
memory usage: 3.8 MB
```

[203]:
```
data2.describe()
```

[203]:

	Open_Bid	High_Bid	Low_Bid	Close_Bid	Volume_Bid	Volume_Ask	Volume_Delta	Volume_Delta_abs	Open_Delta	High_Delta	Low_Del
count	35220.000000	35220.000000	35220.000000	35220.000000	35220.000000	35220.000000	35220.000000	35220.000000	35220.000000	35220.000000	35220.0000
mean	0.553769	0.556522	0.550748	0.553770	0.022654	0.095985	0.206354	0.084729	0.905157	0.905155	0.9049
std	0.218481	0.218488	0.218504	0.218494	0.024550	0.106733	0.091225	0.104995	0.010534	0.010974	0.0162
min	0.000798	0.004150	0.000000	0.000798	0.000000	0.000000	0.000000	0.000000	0.402597	0.282468	-0.3246
25%	0.404429	0.407342	0.400958	0.404310	0.007141	0.025894	0.147285	0.016490	0.905844	0.905844	0.9058
50%	0.586113	0.588667	0.583400	0.586113	0.014580	0.060819	0.173862	0.047034	0.905844	0.905844	0.9058
75%	0.675818	0.679330	0.672785	0.675818	0.029227	0.127558	0.231854	0.113856	0.905844	0.905844	0.9058
max	0.994413	0.995211	0.991221	0.994413	0.329981	1.000000	1.000000	1.000000	0.967532	0.944805	1.0000

[204]:
```
columns_float =['Open_Bid', 'High_Bid', 'Low_Bid', 'Close_Bid',
                'Volume_Bid', 'Volume_Ask', 'Volume_Delta', 'Volume_Delta_abs',
                'Open_Delta', 'High_Delta', 'Low_Delta', 'Close_Delta',
                'Y_High_Bid', 'Y_Low_Ask']
```

[205]:
```
for column in columns_float:
    data2[column] = pd.to_numeric(data2[column], downcast = 'float')

data2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35220 entries, 0 to 35219
Data columns (total 14 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Open_Bid          35220 non-null  float32
 1   High_Bid          35220 non-null  float32
 2   Low_Bid           35220 non-null  float32
 3   Close_Bid         35220 non-null  float32
 4   Volume_Bid        35220 non-null  float32
 5   Volume_Ask        35220 non-null  float32
 6   Volume_Delta      35220 non-null  float32
 7   Volume_Delta_abs  35220 non-null  float32
 8   Open_Delta        35220 non-null  float32
 9   High_Delta        35220 non-null  float32
 10  Low_Delta         35220 non-null  float32
 11  Close_Delta       35220 non-null  float32
 12  Y_High_Bid        35220 non-null  float32
 13  Y_Low_Ask         35220 non-null  float32
```

```
dtypes: float32(14)
memory usage: 1.9 MB
```

[206]:
```
# Writing a normalised dataset to disk in file Silver_2023_normilised.csv

file_obj1 = open('Silver_2023_normalised.csv', 'w')
data2.to_csv('Silver_2023_normalised.csv', encoding='utf-8', index=False)
file_obj1.close()
```

[207]:
```
data2 = pd.read_csv('Silver_2023_normalised.csv', low_memory = False, sep = ',')
data2.head()
```

[207]:

	Open_Bid	High_Bid	Low_Bid	Close_Bid	Volume_Bid	Volume_Ask	Volume_Delta	Volume_Delta_abs	Open_Delta	High_Delta	Low_Delta	Close_Delta	Y_High_Bid
0	0.662091	0.665762	0.659058	0.665762	0.000952	0.000960	0.133180	0.000009	0.788961	0.788961	0.788961	0.788961	0.665762
1	0.666560	0.677095	0.666560	0.671029	0.005048	0.003159	0.131287	0.002175	0.905844	0.814935	0.905844	0.844156	0.677095
2	0.671349	0.671987	0.651876	0.652674	0.002470	0.002412	0.133115	0.000066	0.844156	0.840909	0.840909	0.840909	0.671987
3	0.652674	0.653152	0.646449	0.653152	0.002445	0.002445	0.133172	0.000000	0.844156	0.847403	0.847403	0.847403	0.653152
4	0.652833	0.660176	0.652514	0.660016	0.002429	0.002474	0.133217	0.000052	0.847403	0.844156	0.850649	0.840909	0.660176

[208]:
```
data2.tail()
```

[208]:

	Open_Bid	High_Bid	Low_Bid	Close_Bid	Volume_Bid	Volume_Ask	Volume_Delta	Volume_Delta_abs	Open_Delta	High_Delta	Low_Delta	Close_Delta	Y_Higl
35215	0.457143	0.457143	0.455387	0.456185	0.000753	0.004924	0.137334	0.004802	0.905844	0.905844	0.905844	0.905844	0.45
35216	0.454589	0.456983	0.454589	0.456185	0.000386	0.002070	0.134853	0.001939	0.876623	0.905844	0.876623	0.905844	0.45
35217	0.456983	0.456983	0.456185	0.456185	0.000189	0.001152	0.134133	0.001109	0.905844	0.905844	0.905844	0.905844	0.45
35218	0.456185	0.458579	0.454589	0.457781	0.000591	0.001870	0.134448	0.001472	0.905844	0.886364	0.879870	0.905844	0.45
35219	0.457781	0.459377	0.449002	0.456026	0.000894	0.001015	0.133293	0.000139	0.905844	0.857143	0.727273	0.788961	0.45

[209]:
```
data2.info()
```

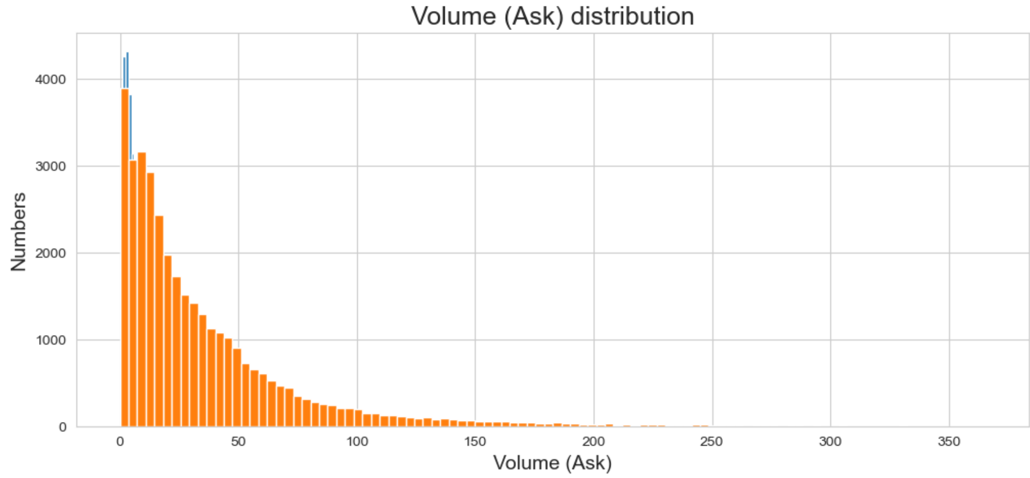
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35220 entries, 0 to 35219
Data columns (total 14 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Open_Bid          35220 non-null  float64
 1   High_Bid          35220 non-null  float64
 2   Low_Bid           35220 non-null  float64
 3   Close_Bid         35220 non-null  float64
 4   Volume_Bid        35220 non-null  float64
 5   Volume_Ask        35220 non-null  float64
 6   Volume_Delta      35220 non-null  float64
 7   Volume_Delta_abs  35220 non-null  float64
 8   Open_Delta        35220 non-null  float64
 9   High_Delta        35220 non-null  float64
 10  Low_Delta         35220 non-null  float64
 11  Close_Delta       35220 non-null  float64
 12  Y_High_Bid        35220 non-null  float64
 13  Y_Low_Ask         35220 non-null  float64
dtypes: float64(14)
memory usage: 3.8 MB
```

[210]:
```
data2.shape
```

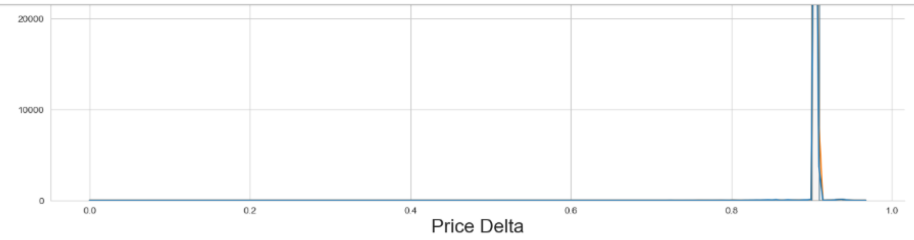
[210]:
```
(35220, 14)
```

[211]:
```
fig = plt.figure(figsize=(12, 5))
plt.rc('axes', titlesize= 30 )
```

```
plt.rc('axes', titlesize= 30 )
sns.set_style('whitegrid')
sns.set_context(rc={'legend.fontsize': 20.0})

sns.displot(
            data2[['Open_Delta', 'Close_Delta']],
            height=8,
            aspect=1.7,
            kde=True,
            element="step",
            bins=100,
            alpha=0.03,

)

plt.title('Open_Delta & Close_Delta Price Distribution')
plt.xlabel('Price Delta', size= 20)
plt.ylabel('Numbers', size= 20)
plt.show();
```




```
<Figure size 1200x500 with 0 Axes>
```



5. Shift the Min & Max prices

[212]:
```
n = data2.shape[0]
n
```

[212]:
```
35220
```

[213]:
```
p = data2.shape[1]
p
```

[213]:
```
14
```

[214]:
```
data_ax = data2.drop(['Y_High_Bid', 'Y_Low_Ask'], axis=1)
```

[215]:
```
data_ax.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35220 entries, 0 to 35219
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Open_Bid          35220 non-null  float64
 1   High_Bid          35220 non-null  float64
 2   Low_Bid           35220 non-null  float64
 3   Close_Bid         35220 non-null  float64
 4   Volume_Bid        35220 non-null  float64
 5   Volume_Ask        35220 non-null  float64
 6   Volume_Delta      35220 non-null  float64
 7   Volume_Delta_abs  35220 non-null  float64
 8   Open_Delta        35220 non-null  float64
 9   High_Delta        35220 non-null  float64
 10  Low_Delta         35220 non-null  float64
 11  Close_Delta       35220 non-null  float64
dtypes: float64(12)
memory usage: 3.2 MB
```

[216]:
```
data_ax.shape
```

[216]:
```
(35220, 12)
```

[217]:
```
data_ay = data2[['Y_High_Bid', 'Y_Low_Ask']]
```

[218]:
```
data_ay.shape
```

[218]:
```
(35220, 2)
```

[219]:
```
data_ay.head(6)
```

[219]:

	Y_High_Bid	Y_Low_Ask
0	0.665762	0.669593
1	0.677095	0.671349
2	0.671987	0.659856

5	0.666081	0.667997

[220]:
```
data_ay.tail(10)
```

[220]:

	Y_High_Bid	Y_Low_Ask
35210	0.458420	0.460335
35211	0.457302	0.459697
35212	0.457302	0.459697
35213	0.456345	0.458101
35214	0.457143	0.459856
35215	0.457143	0.460176
35216	0.456983	0.460814
35217	0.456983	0.460974
35218	0.458579	0.460654
35219	0.459377	0.462570

[221]:
```
data_ay = data_ay.shift(-1)
```

[222]:
```
data_ay.head(6)
```

[222]:

	Y_High_Bid	Y_Low_Ask
0	0.677095	0.671349
1	0.671987	0.659856
2	0.653152	0.654110
3	0.660176	0.660016
4	0.666081	0.667997
5	0.672466	0.670710

[223]:
```
data_ay.tail(10)
```

[223]:

	Y_High_Bid	Y_Low_Ask
35210	0.457302	0.459697
35211	0.457302	0.459697
35212	0.456345	0.458101
35213	0.457143	0.459856
35214	0.457143	0.460176
35215	0.456983	0.460814
35216	0.456983	0.460974
35217	0.458579	0.460654
35218	0.459377	0.462570
35219	NaN	NaN

[224]:
```
# create new columns: MIN_Lowest(Low_Ask) and MAX_Highest(High_Bid) prices of DURING NEXT 5 minutes

indexer = pd.api.indexers.FixedForwardWindowIndexer(window_size=5)

data_ay['Y_High_Bid_5'] = data_ay['Y_High_Bid'].rolling(window=indexer).max()
```

```
data_ay['Y_High_Bid_5'] = data_ay['Y_High_Bid'].rolling(window=indexer).max()
data_ay['Y_Low_Ask_5']  = data_ay['Y_Low_Ask'].rolling(window=indexer).min()
```

[225]:
```
data_ay.head(6)
```

[225]:

	Y_High_Bid	Y_Low_Ask	Y_High_Bid_5	Y_Low_Ask_5
0	0.677095	0.671349	0.677095	0.654110
1	0.671987	0.659856	0.672466	0.654110
2	0.653152	0.654110	0.672466	0.654110
3	0.660176	0.660016	0.672466	0.660016
4	0.666081	0.667997	0.672466	0.666241
5	0.672466	0.670710	0.672466	0.666241

[226]:
```
data_ay.tail(10)
```

[226]:

	Y_High_Bid	Y_Low_Ask	Y_High_Bid_5	Y_Low_Ask_5
35210	0.457302	0.459697	0.457302	0.458101
35211	0.457302	0.459697	0.457302	0.458101
35212	0.456345	0.458101	0.457143	0.458101
35213	0.457143	0.459856	0.458579	0.459856
35214	0.457143	0.460176	0.459377	0.460176
35215	0.456983	0.460814	NaN	NaN
35216	0.456983	0.460974	NaN	NaN
35217	0.458579	0.460654	NaN	NaN
35218	0.459377	0.462570	NaN	NaN
35219	NaN	NaN	NaN	NaN

[227]:
```
data_ay = data_ay.drop(['Y_High_Bid', 'Y_Low_Ask'], axis =1)
```

[228]:
```
data_ay.tail(6)
```

[228]:

	Y_High_Bid_5	Y_Low_Ask_5
35214	0.459377	0.460176
35215	NaN	NaN
35216	NaN	NaN
35217	NaN	NaN
35218	NaN	NaN
35219	NaN	NaN

[229]:
```
# delete the 5 last rows in data_ay (because we don't have answers for the last 5 minutes)

for i in range(5):
        data_ay.drop(data_ay.shape[0]-1, axis=0, inplace=True)
```

[230]:
```
# delete the last 5 rows in data_ax (because we don't have answers for the last 5 minutes)

for i in range(5):
    data_ax.drop(data_ax.shape[0]-1, axis=0, inplace=True)
```

Ba
Co

[230]:
```
# delete the last 5 rows in data_ax (because we don't have answers for the last 5 minutes)

for i in range(5):
    data_ax.drop(data_ax.shape[0]-1, axis=0, inplace=True)
```

[231]:
```
data_ay.shape
```

[231]:
```
(35215, 2)
```

[232]:
```
data_ax.shape
```

[232]:
```
(35215, 12)
```

6. Separate inputs & outputs matrices

[233]:
```
# Create inputs Numpy

data_a = np.array(data_ax)

with np.printoptions(precision=4):
    print("data_a:")
    print(data_a[:2,:])
    print('\n')
    print(data_a[-2:,:])

print("numpy size: = ", data_a.shape)
print("type: ", data_a.dtype)
```

```
data_a:
[[6.6209e-01 6.6576e-01 6.5906e-01 6.6576e-01 9.5178e-04 9.5998e-04
  1.3318e-01 9.4463e-06 7.8896e-01 7.8896e-01 7.8896e-01 7.8896e-01]
 [6.6656e-01 6.7710e-01 6.6656e-01 6.7103e-01 5.0482e-03 3.1589e-03
  1.3129e-01 2.1752e-03 9.0584e-01 8.1494e-01 9.0584e-01 8.4416e-01]]


[[0.4541 0.4563 0.4533 0.4555 0.0053 0.0095 0.1374 0.0049 0.9058 0.9058
  0.9058 0.9058]
 [0.4555 0.4571 0.4551 0.4571 0.001  0.004  0.1362 0.0034 0.9058 0.9058
  0.9058 0.9058]]
numpy size: =  (35215, 12)
type:  float64
```

[234]:
```
# Create outputs Numpy

data_y = np.array(data_ay)

print("data_a:")
print(data_y[:3,:])
print('\n')
print(data_y[-3:,:])

print("numpy size: = ", data_y.shape)
print("type: ", data_y.dtype)
```

```
data_a:
[[0.677095   0.65411013]
 [0.6724661  0.65411013]
 [0.6724661  0.65411013]]


[[0.45714286 0.45810056]
 [0.45857942 0.45985633]
 [0.4593775  0.46017557]]
numpy size: =  (35215, 2)
type:  float64
```

7. Create a 3D Tensor

[235]:
```
import tensorflow as tf
from tensorflow import keras

#from tensorflow.keras.callbacks import EarlyStopping
#from tensorflow.keras.callbacks import ModelCheckpoint
```

[236]:
```
print(tf.__version__)
```

```
2.18.0
```

[237]:
```
np.random.seed(42)
```

[238]:
```
from tqdm import tqdm
```

[239]:
```
n_small = 30000
N=50

L=n_small-N
t=data_a.shape[1]
print('Size of a three-dimensional inputs tensor: ', L,N,t)
```

```
Size of a three-dimensional inputs tensor:  29950 50 12
```

[240]:
```
# create inputs zeros 3D tensors with 'float32'

data_b = np.zeros( (L, N, t), dtype= 'float32' )
```

[241]:
```
# create a two-dimensional zeros vector of answers - normalised the High_Bid and Low_Ask prices DURING next 5 minutes

Y = np.zeros((L, 2), dtype= 'float32') # We will predict two prices - normalised High_Bid and normalised Low_Ask
```

[242]:
```
# fill the inputs 3D tensor (data_b)

print('L = n_small - N - 5 = ', n_small-N-5, L)
for k in tqdm(range(L)):
    data_b[k, :, :] = data_a[k:k+N, :]

    Y[k,0] = data_y[k+N-1,0]   # normalised Y_High_Bid - max 5 minutes future price
    Y[k,1] = data_y[k+N-1,1]   # normalised Y_Low_Ask -  min 5 minutes future price

print(k) # index value (for control)

print('data_b:', '\n', data_b)
print("Numpy size:        ", data_b.shape, '\n')

print( Y[:5,:])
print( Y[-10:,:])
print(      "Numpy size:         ", Y.shape)
```

```
L = n_small - N - 5 =  29945 29950
100%|████████████████████████████████████████████████████████████| 29950/29950 [00:00<00:00, 163022.89it/s]
29949
data_b:
 [[[0.66209096 0.6657622  0.6590583  ... 0.78896105 0.78896105 0.78896105]
  [0.66656023 0.677095   0.66656023 ... 0.8149351  0.90584415 0.84415585]
  [0.67134875 0.67198724 0.6518755  ... 0.84090906 0.84090906 0.84090906]
  ...
  [0.69114125 0.6964086  0.6901836  ... 0.90584415 0.90584415 0.90584415]
  [0.6936951  0.6970471  0.6914605  ... 0.91883117 0.90584415 0.90584415]
  [0.6952913  0.7122107  0.694174   ... 0.90584415 0.90584415 0.90584415]]

 [[0.66656023 0.677095   0.66656023 ... 0.8149351  0.90584415 0.84415585]
  [0.67134875 0.67198724 0.6518755  ... 0.84090906 0.84090906 0.84090906]
  [0.6526736  0.6531524  0.64644855 ... 0.8474026  0.8474026  0.8474026 ]
  ...
```

[243]:
```
# Control of the correctness of filling the array Y
# (should show MAX of the normalised prices High_Bid and Low_Ask - during 5 next minutes(step) ahead)


import random

pp = random.randint(50, L) # (any number before 29950) - just to look at the middle of data_b
print('random int = ', pp)

print('data_b:', '\t\t','Y:')
print('Y_High_Bid ','\t\t','Y_High_Bid')

for i in range(15):
    print(data_b[pp+i,N-1,1], '\t\t', Y[pp+i,0])
```

```
random int =  16760
data_b:                 Y:
Y_High_Bid              Y_High_Bid
0.55115724              0.5535515
0.55035913              0.5535515
0.55035913              0.5535515
0.5527534               0.5535515
0.5535515               0.5527534
0.5527534               0.5527534
0.5527534               0.5527534
0.5527534               0.5527534
0.5519553               0.5527534
0.5527534               0.5527534
0.5527534               0.55115724
0.55115724              0.55115724
0.55115724              0.55035913
0.55035913              0.549561
0.549561                0.548763
```

[244]:
```
data = []
data2 = []
data_ax = []
data_ay = []
data_a = []
data_y = []
```

8. Create and Train Neural Network(LSTM)

[245]:
```
from sklearn.model_selection import train_test_split
```

[246]:
```
X_train, X_test, y_train, y_test = train_test_split(data_b, Y, test_size = 0.1, shuffle = False, stratify = None, random_state =101)
```

[247]:
```
train_start = 0
train_end = int(np.floor(0.9*L))
print(L, train_end)
```

```
29950 26955
```

[248]:
```
train_end -= 955
print(train_end)
```

```
26000
```

[249]:
```
test_start = train_end + 1
test_end = L
print(test_start, test_end)
```

```
26001 29950
```

[250]:
```
print(test_end - train_end)
```

```
3950
```

[251]:
```
print(X_train.shape)
```

[249]:

```
test_start = train_end + 1
test_end = L
print(test_start, test_end)
```

```
26001 29950
```

[250]:

```
print(test_end - train_end)
```

```
3950
```

[251]:

```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(26955, 50, 12)
(26955, 2)
(2995, 50, 12)
(2995, 2)
```

[252]:

```
model = keras.Sequential([
    keras.layers.Conv1D(50,9, padding = 'same', input_shape= (50,12), activation=tf.nn.relu, kernel_initializer='normal'),
    keras.layers.MaxPooling1D(7),
    keras.layers.Conv1D(100,7,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())
```

```
C:\Users\abkha\anaconda3\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:107: 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__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv1d_2 (Conv1D)	(None, 50, 50)	5,450
max_pooling1d_1 (MaxPooling1D)	(None, 7, 50)	0
conv1d_3 (Conv1D)	(None, 7, 100)	35,100
global_max_pooling1d_1 (GlobalMaxPooling1D)	(None, 100)	0
dense_2 (Dense)	(None, 25)	2,525
dense_3 (Dense)	(None, 2)	52

Total params: 43,127 (168.46 KB)

Trainable params: 43,127 (168.46 KB)

Non-trainable params: 0 (0.00 B)

None

[253]:

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

[254]:

```
#es = EarlyStopping(monitor='val_loss', mode='min', patience=5, verbose=1)
#mc = ModelCheckpoint('best_model_LSTM_SILVER.keras', monitor='val_loss', mode='min', verbose=1, save_best_only=True)
```

[255]:

```
history = model.fit(X_train, y_train, batch_size =30, epochs=15, validation_split=0.2, verbose=1)
```

```
Epoch 1/15
719/719 ━━━━━━━━━━━━━━━━━━━━ 9s 8ms/step - loss: 0.0133 - mae: 0.0440 - val_loss: 1.9814e-04 - val_mae: 0.0091
Epoch 2/15
719/719 ━━━━━━━━━━━━━━━━━━━━ 7s 9ms/step - loss: 2.2739e-04 - mae: 0.0112 - val_loss: 1.5894e-04 - val_mae: 0.0085
Epoch 3/15
719/719 ━━━━━━━━━━━━━━━━━━━━ 7s 10ms/step - loss: 1.6707e-04 - mae: 0.0096 - val_loss: 1.1846e-04 - val_mae: 0.0072
```

9. Calculate MSE & MAE on Test dataset

[256]:
```
#LSTM_saved_best_model = keras.models.load_model('best_model_LSTM_SILVER.keras')
```

[257]:
```
#scores = LSTM_saved_best_model.evaluate(X_test, y_test, verbose=1)
```

[258]:
```
#scores
```

[259]:
```
#print("Mean squared error (mse): %.9f " % (scores[0]))
```

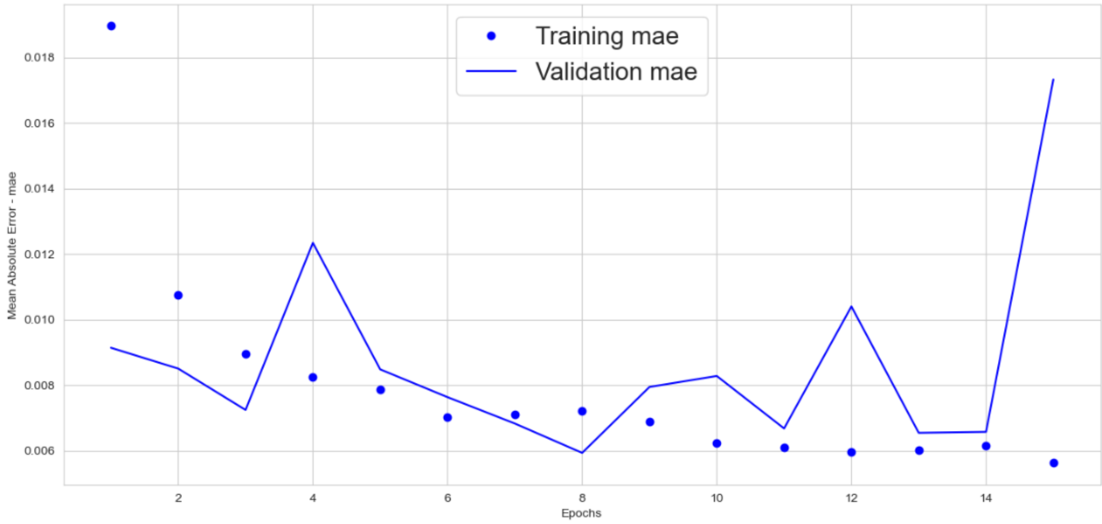
[260]:
```
#print("Mean absolute error (mae): %.9f " % (scores[1]))
```

[261]:
```
mse,mae = model.evaluate(X_test, y_test, verbose =1)
print("Mean absolute error: %.5f" %mae)
```

```
94/94 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 2.5491e-04 - mae: 0.0150
Mean absolute error: 0.01462
```

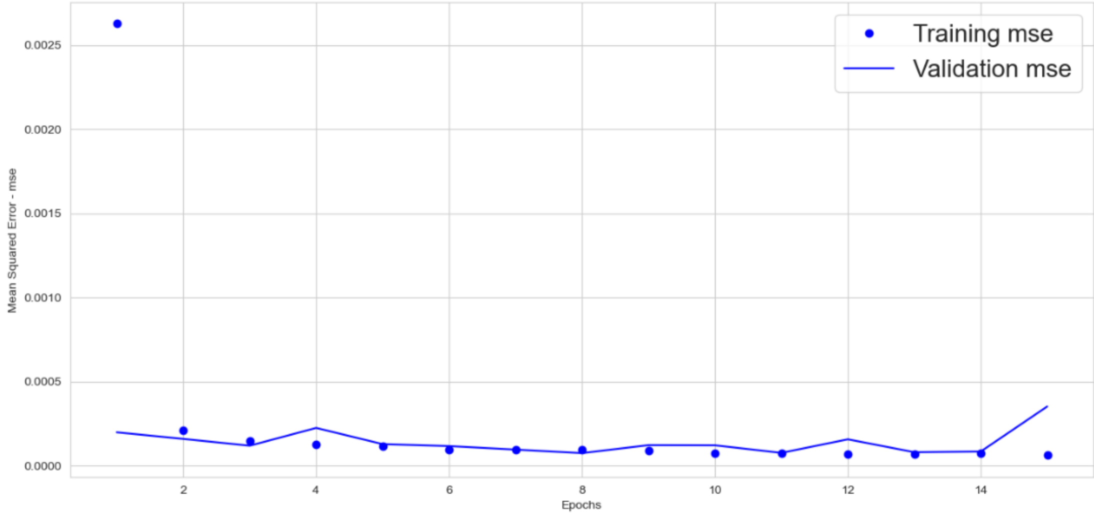
10. Plotting the result graphs

[262]:
```
history_dict = history.history

mean_absolute_error_values = history_dict['mae']
val_mean_absolute_error_values = history_dict['val_mae']
epochs = range(1, len(mean_absolute_error_values) + 1)

plt.figure(num=1, figsize=(15,7))
plt.plot(epochs, mean_absolute_error_values, 'bo', label='Training mae')
plt.plot(epochs, val_mean_absolute_error_values, 'b', label='Validation mae')
plt.xlabel('Epochs')
plt.ylabel('Mean Absolute Error - mae')
plt.legend()
plt.show()
```



[263]:

```
history_dict = history.history

mean_absolute_error_values = history_dict['loss']
val_mean_absolute_error_values = history_dict['val_loss']
epochs = range(1, len(mean_absolute_error_values) + 1)

plt.figure(num=1, figsize=(15,7))
plt.plot(epochs, mean_absolute_error_values, 'bo', label='Training mse')
plt.plot(epochs, val_mean_absolute_error_values, 'b', label='Validation mse')
plt.xlabel('Epochs')
plt.ylabel('Mean Squared Error - mse')
plt.legend()
plt.show()
```



[264]:

```
plt.plot(history.history['loss'])
plt.xlabel('Mean Squared Error', size =14)
```

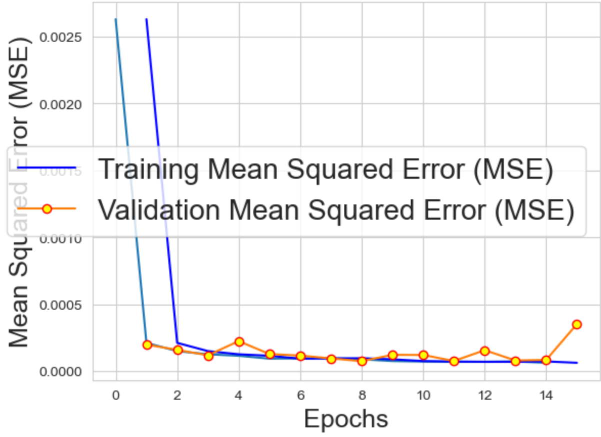
[264]: Text(0.5, 0, 'Mean Squared Error')

[265]:

```
# More detailed MSE graph

history_dict = history.history

mse_values = history_dict['loss']
val_mse_values = history_dict['val_loss']

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

[265]:

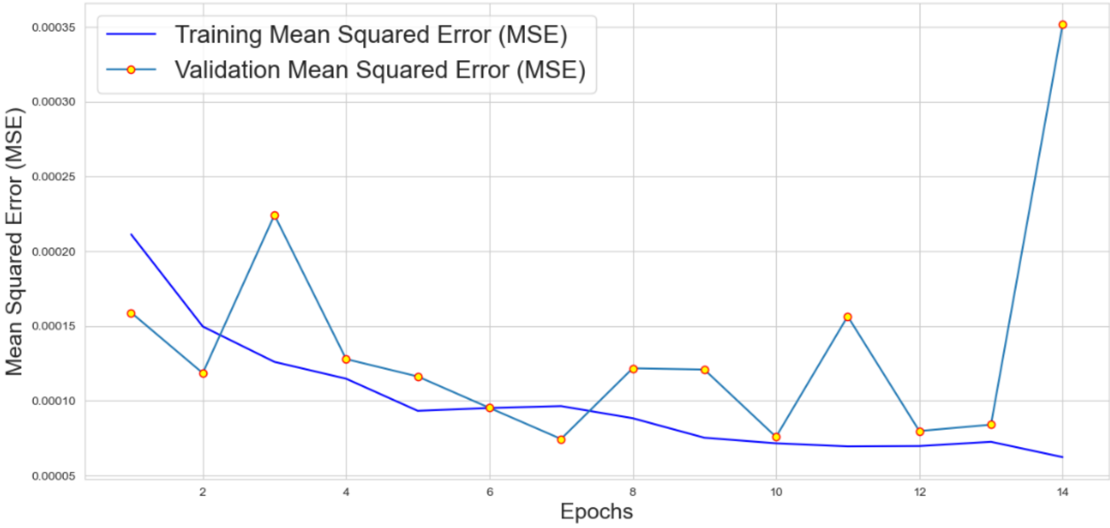
```python
# More detailed MSE graph

history_dict = history.history

mse_values = history_dict['loss']
val_mse_values = history_dict['val_loss']

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



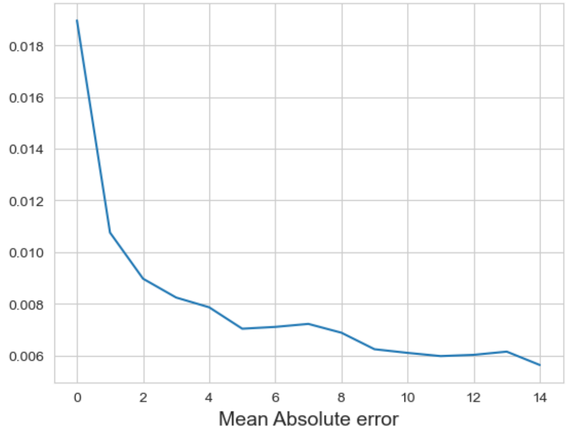
[266]:

```python
# deleted the 1st epoch
# to examine in detail the results of the remaining epochs

history_dict = history.history

mse_values = history_dict['loss'][1:]
val_mse_values = history_dict['val_loss'][1:]

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

[266]:
```python
# deleted the 1st epoch
# to examine in detail the results of the remaining epochs

history_dict = history.history

mse_values = history_dict['loss'][1:]
val_mse_values = history_dict['val_loss'][1:]

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



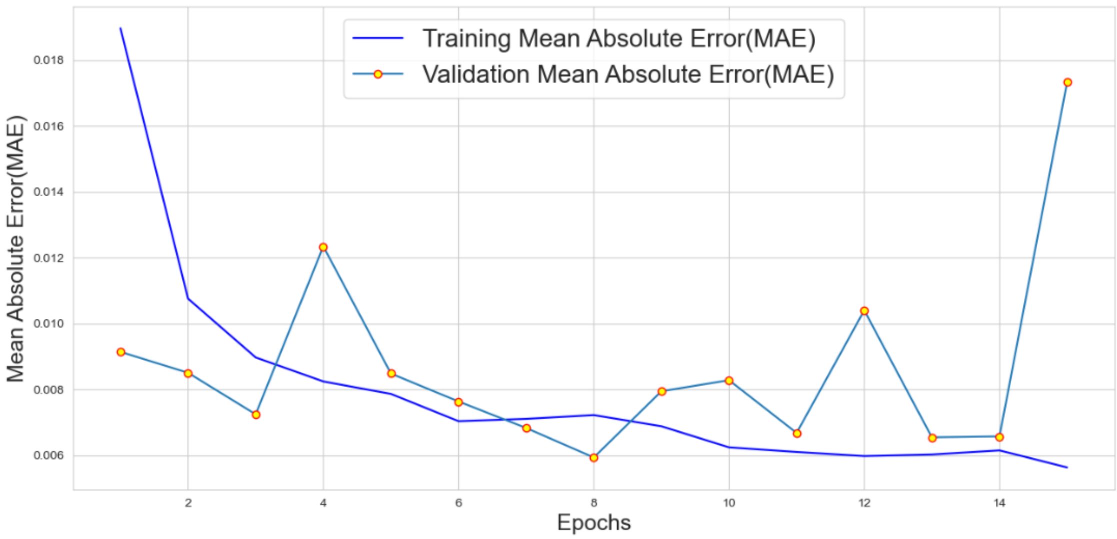
```
[267]: plt.plot(history.history['mae'])
plt.xlabel('Mean Absolute error', size=14)
plt.show()
```




```
[268]: 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()
```

Mean Absolute error


```
[268]: 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

Github - https://github.com/Abkhadar/m-l-in-finance/tree/main

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
```

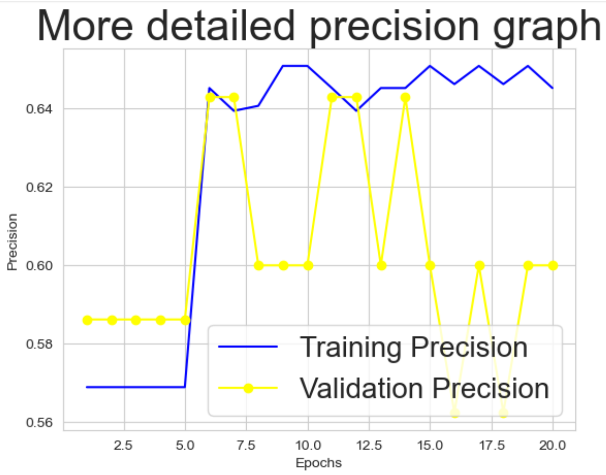
[153]:

```
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):
```

```python
    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]:
```
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'])
```

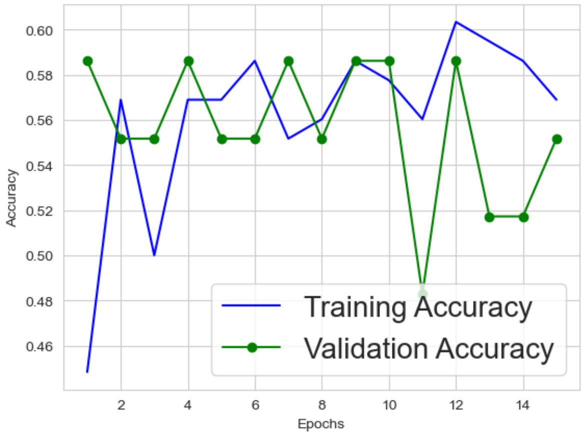
```python
model = Sequential([
    LSTM(50, input_shape=(X_train.shape[1], X_train.shape[2]), activation='relu'),
    Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])


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'])
```

```python
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]:

```
# 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
```

[ ]:
```

```

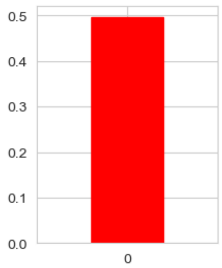
[159]:
```
import random

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

Github - https://github.com/Abkhadar/m-l-in-finance/tree/main

Lab Logbook Requirement:

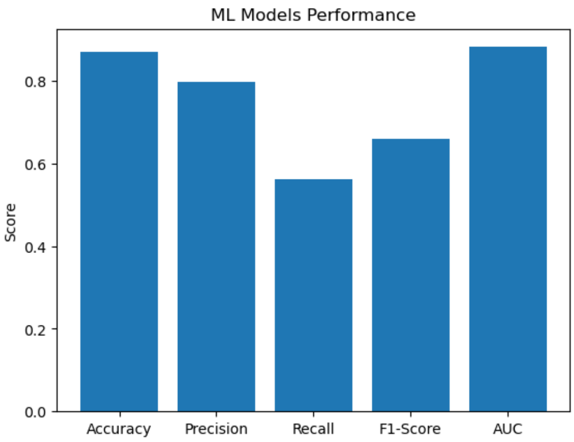
1. *Create and train your own LSTM model*
2. *Add all the LSTM's Error metrics: Accuracy, Precision, Recall, F1-Score and AUC to the final histogram "ML Models performance...".*

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)
```

```
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
```



[ ]: