LOGBOOK

MACHINE LEARNING IN FINANCE

Subhro Mukherjee

SID - 2333889

Github link - https://github.com/subhromukherjee/ML-in-finance

Table of Contents

WEEK 1

Lab Logbook Requirement:

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

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

Then,

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

¶

In [80]:
```
# SID is 2333889, last two digits are 89
sid_last_two_digits = 89

# Checking if the number is less than 10, if so add 100
vector_length = sid_last_two_digits if sid_last_two_digits >= 10 else sid_last_two_digits + 100

# Creating a vector using np.arange with the determined number of elements
vector = np.arange(vector_length)

# Output the length of the vector and the vector itself
vector_length, vector
```

Out[80]:
```
(89,
 array([ 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, 43, 44, 45, 46, 47, 48, 49, 50,
        51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
        68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,
        85, 86, 87, 88]))
```

In [81]:
```
a = vector

# Reshaping matrix a to a 2D array with 1 row
a_reshaped = a.reshape(1, -1)

a_reshaped
```

Out[81]:
```
array([[ 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, 43, 44, 45, 46, 47,
        48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63,
        64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79,
        80, 81, 82, 83, 84, 85, 86, 87, 88]])
```

In [82]:
```
a_reshaped = a.reshape(1, -1)

# Saving the reshaped array into another array
b = a_reshaped

b
```

Out[82]:
```
array([[ 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, 43, 44, 45, 46, 47,
        48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63,
        64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79,
        80, 81, 82, 83, 84, 85, 86, 87, 88]])
```

In [83]:
```
print("\nShape of array b:", b.shape)
```

```
Shape of array b: (1, 89)
```

WEEK 2

Lab Logbook Requirement:

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

In [52]:
```
# SID = 2333889
n = 9

data = pd.read_csv('adult_data_mini.csv', header=0)
df = pd.DataFrame(data)

Group_by_relationship = data.groupby(['relationship', 'hours-per-week'])
print(type(Group_by_relationship))
Group_by_relationship.size()
```

```
<class 'pandas.core.groupby.generic.DataFrameGroupBy'>
```

Out[52]:
```
relationship    hours-per-week
 Husband        13              1
                40              4
                45              1
                80              1
 Not-in-family  16              1
                40              2
                50              2
 Own-child      30              1
 Wife           40              2
dtype: int64
```

In [55]:
```
# Reducing all "hours-per-week" values by n
df['reduced-hours-per-week'] = df['hours-per-week'] - n
# Printing the updated DataFrame
print("Updated DataFrame with reduced hours-per-week:")
print(df[['hours-per-week', 'reduced-hours-per-week']])
```

```
Updated DataFrame with reduced hours-per-week:
    hours-per-week  reduced-hours-per-week
0               40                      31
1               13                       4
2               40                      31
3               40                      31
4               40                      31
5               40                      31
6               16                       7
7               45                      36
8               50                      41
9               40                      31
10              80                      71
11              40                      31
12              30                      21
13              50                      41
14              40                      31
```

In [54]:
```
# Grouping by "relationship" and original "hours-per-week"
grouped_original = df.groupby(['relationship', 'hours-per-week']).size().reset_index(name='count')

# Grouping by "relationship" and reduced "hours-per-week"
grouped_reduced = df.groupby(['relationship', 'reduced-hours-per-week']).size().reset_index(name='count')

# Printing the results
print("Grouped by relationship and original hours-per-week:")
print(grouped_original)

print("\nGrouped by relationship and reduced hours-per-week:")
print(grouped_reduced)
```

```
Grouped by relationship and original hours-per-week:
     relationship  hours-per-week  count
0         Husband              13      1
1         Husband              40      4
2         Husband              45      1
3         Husband              80      1
4   Not-in-family              16      1
5   Not-in-family              40      2
6   Not-in-family              50      2
7       Own-child              30      1
8            Wife              40      2

Grouped by relationship and reduced hours-per-week:
     relationship  reduced-hours-per-week  count
0         Husband                       4      1
1         Husband                      31      4
2         Husband                      36      1
3         Husband                      71      1
4   Not-in-family                       7      1
5   Not-in-family                      31      2
6   Not-in-family                      41      2
7       Own-child                      21      1
8            Wife                      31      2
```

WEEK 3

Lab Logbook Requirement:

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

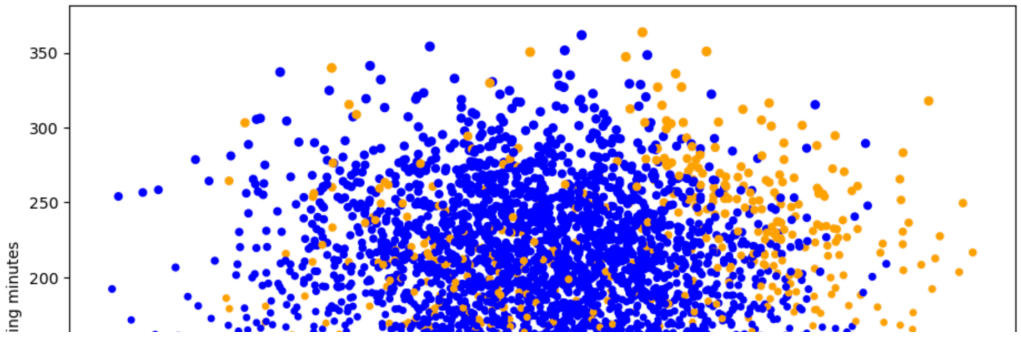
#	Column
1	Account length
2	Area code
3	International plan
4	Voice mail plan
5	Number vmail messages
6	Total day minutes
7	Total day calls
8	Total day charge
9	Total eve minutes
0	Total eve calls

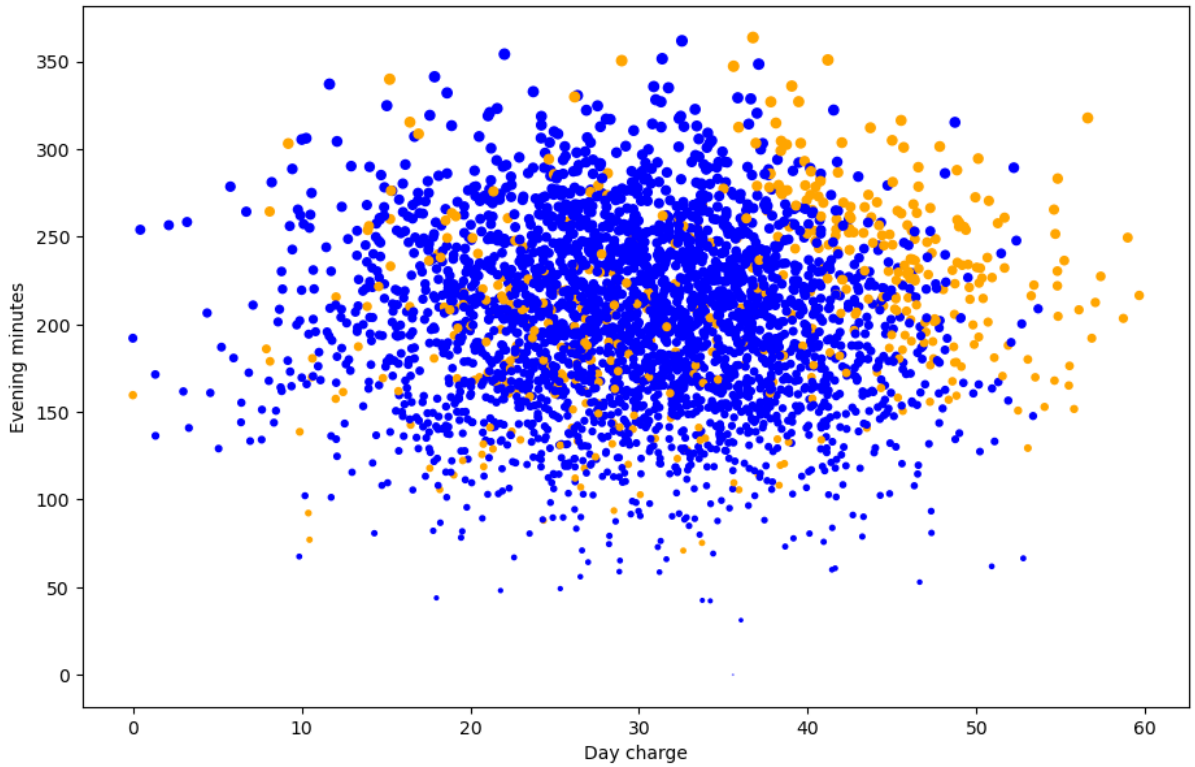
In case these numbers are the same, then take the next number in order as another column number. For example, if your SID is 2287477, then you plot the bicolour diagram of the 7th and 8th columns. If your SID is 2287499, then the 9th and 0.

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

In [66]:
```
# SID = 2333889
# Plotting between col 8 and 9
```

In [64]:
```
fig = plt.figure(figsize=(11,7))
plt.scatter(data['Total day charge'], data['Total eve minutes'], color = clr, s=(data['Total eve charge']+0.05));
plt.xlabel('Day charge');
plt.ylabel('Evening minutes');
```




Evening minutes
Day charge
0
50
100
150
200
250
300
350
0
10
20
30
40
50
60

WEEK 4

Lab Logbook Requirement:

1. ***Create your own Multi-layer Perceptron (MLP) with two hidden layers, where the first hidden layer cells' number equals the last three digits of your SID. The number of cells in the next hidden layer is approximately two times smaller. For example, if your SID is 2287167, the number of cells on the first hidden layer is 167, and on the second - 84. Take epochs=10. Leave other parameters the same as in the practical session.***
2. ***Compile the model.***
3. ***Train your MLP with the same datasets and demonstrate the received MAE.***
4. ***Compare your MAE with the MAE of the MLP in the practical session.***
5. ***Please only add to your Lab Logbook a print-screen of your MLP architecture using model.summary() and the resulting MAE.***

¶

[132]:
```
# SID 2333889
# Define the MLP model
model = keras.Sequential([
    keras.layers.Dense(889, input_dim=500, activation=tf.nn.relu, kernel_initializer="normal"),  # First hidden layer (889 neurons)
    keras.layers.Dense(444, activation='relu', kernel_initializer="normal"),  # Second hidden layer (444 neurons)
    keras.layers.Dense(1)  # Output layer for regression (1 neuron)
])
print(model.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_5 (Dense)             (None, 889)               445389

 dense_6 (Dense)             (None, 444)               395160

 dense_7 (Dense)             (None, 1)                 445

=================================================================
Total params: 840,994
Trainable params: 840,994
Non-trainable params: 0
_________________________________________________________________
None
```

[135]:
```
# Evaluate the model on the test data
mse, mae = model.evaluate(X_test, y_test, verbose=0)
print("Mean absolute error: %.5f" % mae)
print("Mean squared error: %.5f" % mse)
```

```
Mean absolute error: 0.03973
Mean squared error: 0.00231
```

Lab mae

[103]:
```
mse, mae = model.evaluate(X_test, y_test, verbose=0)
print("Mean absolute error: %.5f" % mae)
print("Mean squared error: %.5f" % mse)
```

```
Mean absolute error: 0.02403
Mean squared error: 0.00087
```

WEEK 5

Lab Logbook requirements:

Please make sure to document the following in your Lablogbook:

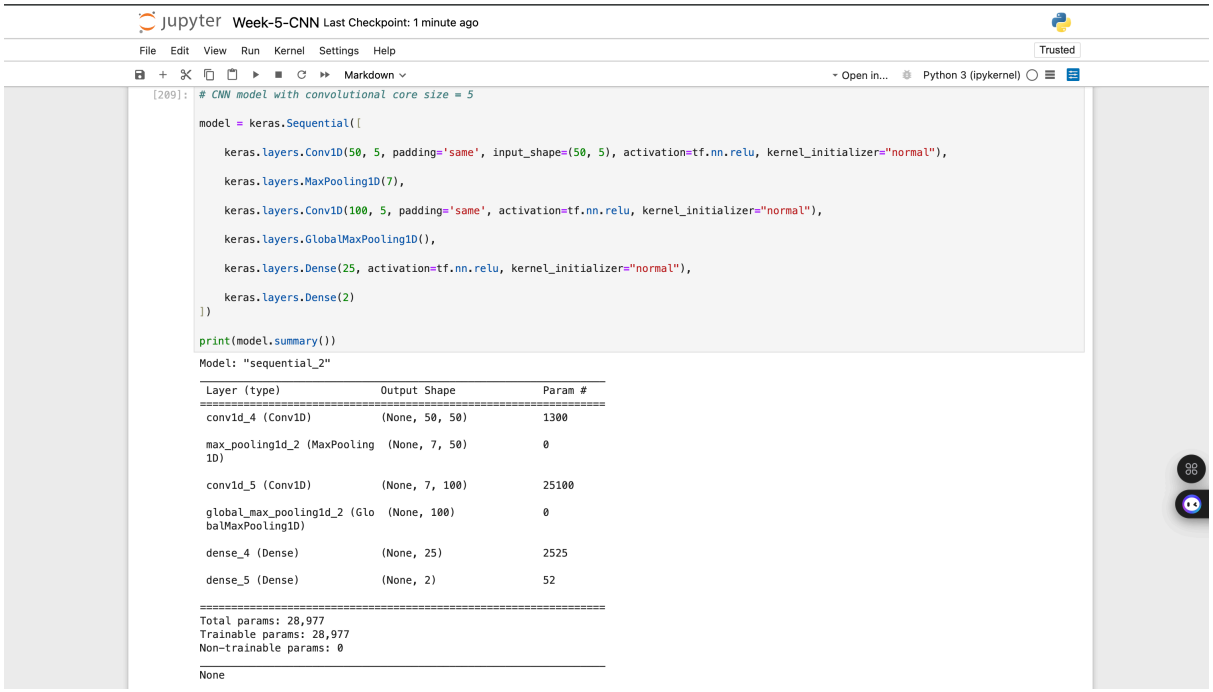
1. Modify the practical session CNN model by reducing the convolutional core size to 5.
2. Change the batch_size to 50.
3. Also, change the size of the number of epochs, which is calculated by the formula:

   Z + Y, if Z and Y are not 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.




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

936/936 [==============================] - 1s 788us/step - loss: 0.3163 - mae: 0.4722
Mean absolute error: 0.47221
```

Lab mae

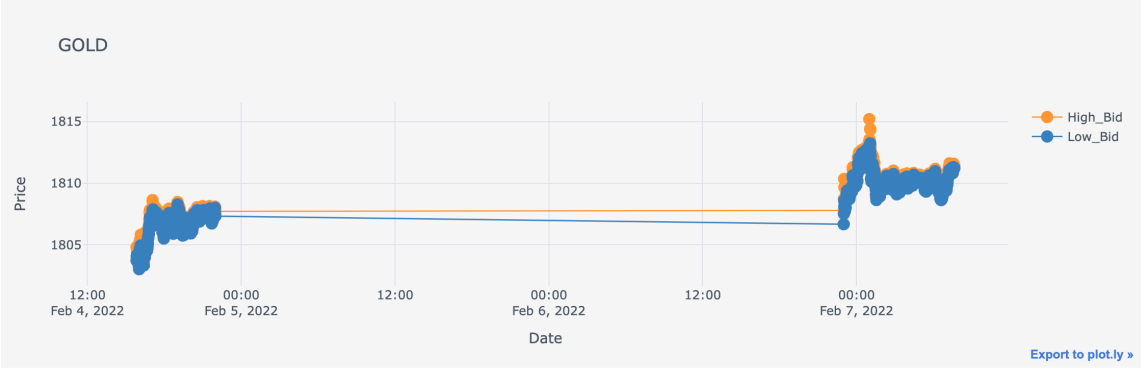
```
[94]: mse, mae = model.evaluate(X_test, y_test, verbose=1)
      print("Mean absolute error: %.5f" % mae)

936/936 [==============================] - 1s 806us/step - loss: 0.2658 - mae: 0.4221
Mean absolute error: 0.42214
```

WEEK 6

Lab Logbook Requirement:

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

[209]:
```
# plot the part of the whole dataset using iplot()
# The time period equals 889 minutes from 33889

sid = 2333889
start_point = int(str(sid)[-5:])  # Last 5 digits of SID for start point
time_period = int(str(sid)[-3:])  # Last 3 digits of SID for time period

# Plotting data from index 33889 to (33889 + 889) for 'High_Bid' and 'Low_Bid' columns
data.iloc[start_point:start_point + time_period, :][['High_Bid', 'Low_Bid', 'Local_time_T', 'Volume_Ask', 'Volume_Bid']].iplot(
    x='Local_time_T', y=['High_Bid', 'Low_Bid'],
    mode='lines+markers',
    xTitle='Date',
    yTitle='Price',
    title='GOLD'
)
```



WEEK 7

Lab Logbook Requirement:

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

   ZY + 10 , where your SID is: XXXXXZY

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

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

[136]:
```
# SID = 2333889
# Z=8,Y=9,
model2 = keras.Sequential([
    keras.layers.LSTM(82, activation='relu', input_shape=(50, 18)),
    keras.layers.Dense(2)

])
```

[138]:
```
print(model2.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_1 (LSTM)               (None, 82)                33128

 dense_1 (Dense)             (None, 2)                 166

=================================================================
Total params: 33,294
Trainable params: 33,294
Non-trainable params: 0
_________________________________________________________________
None
```

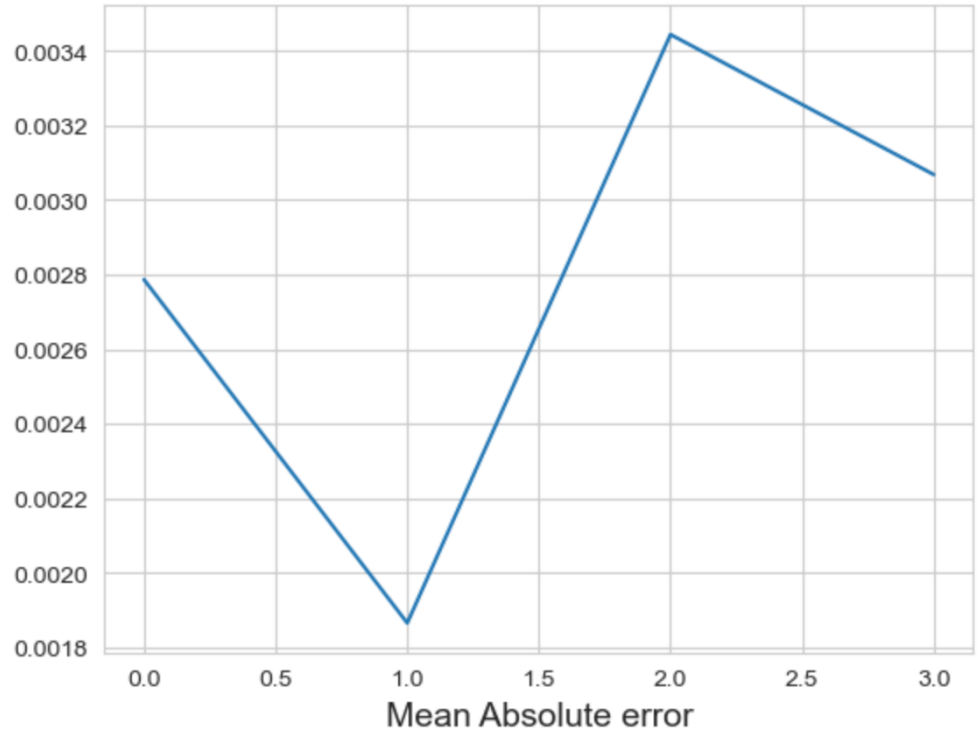
[160]:
```
print("Mean squared error (mse): %.9f " % (scores2[0]))
```

```
Mean squared error (mse): 0.000000941
```

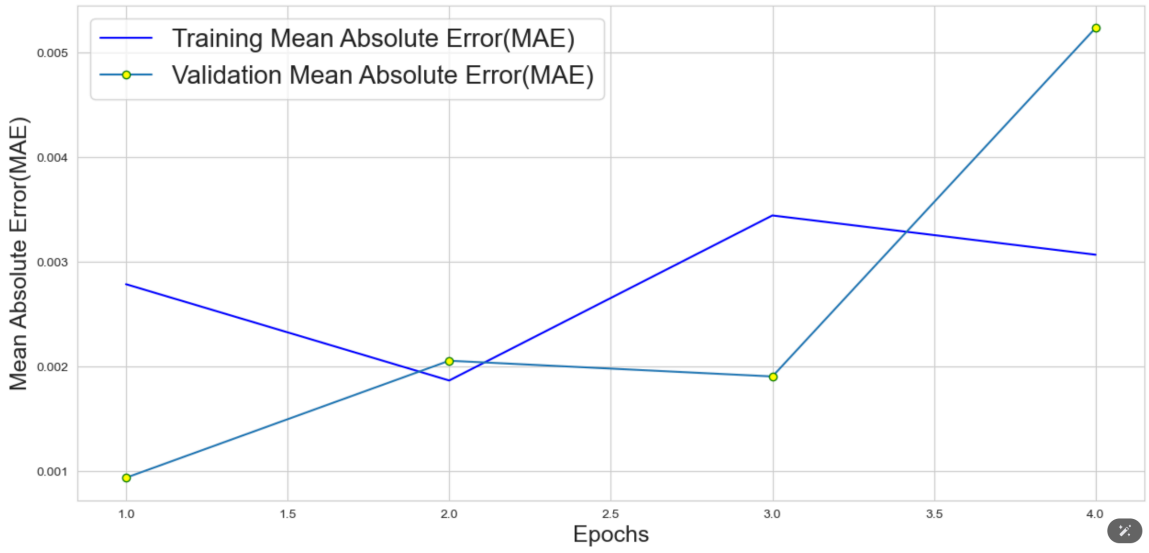
[162]:
```
print("Mean absolute error (mae): %.9f " % (scores2[1]))
```

```
Mean absolute error (mae): 0.000700275
```

[164]:
```
plt.plot(history.history['mae'])
plt.xlabel('Mean Absolute error', size=14)
plt.show()
```




```
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='green', 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 ()
```



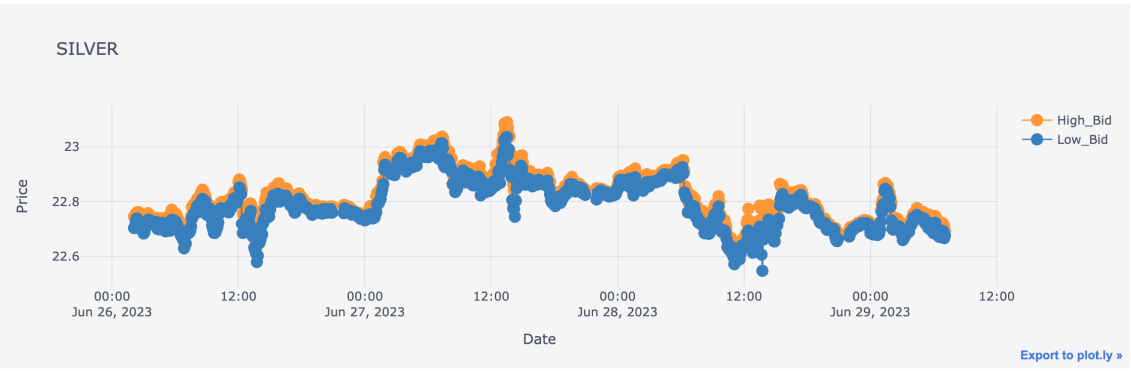
WEEK 8

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.

Plot the price chart of the part of the whole dataset 'High_Bid' and 'Low_Bid' prices using iplot() library.

[251]:
```
# plot the part of the whole dataset using iplot()
# The time period equals 889 minutes from 33889

sid = 2333889
start_point = int(str(sid)[-5:])  # Last 5 digits of SID for start point
time_period = int(str(sid)[-3:])  # Last 3 digits of SID for time period

# Plotting data from index 33889 to (33889 + 889) for 'High_Bid' and 'Low_Bid' columns
data.iloc[start_point:start_point + time_period, :][['High_Bid', 'Low_Bid', 'Local_time_T', 'Volume_Ask', 'Volume_Bid']].iplot(
    x='Local_time_T', y=['High_Bid', 'Low_Bid'],
    mode='lines+markers',
    xTitle='Date',
    yTitle='Price',
    title='SILVER'
)
```



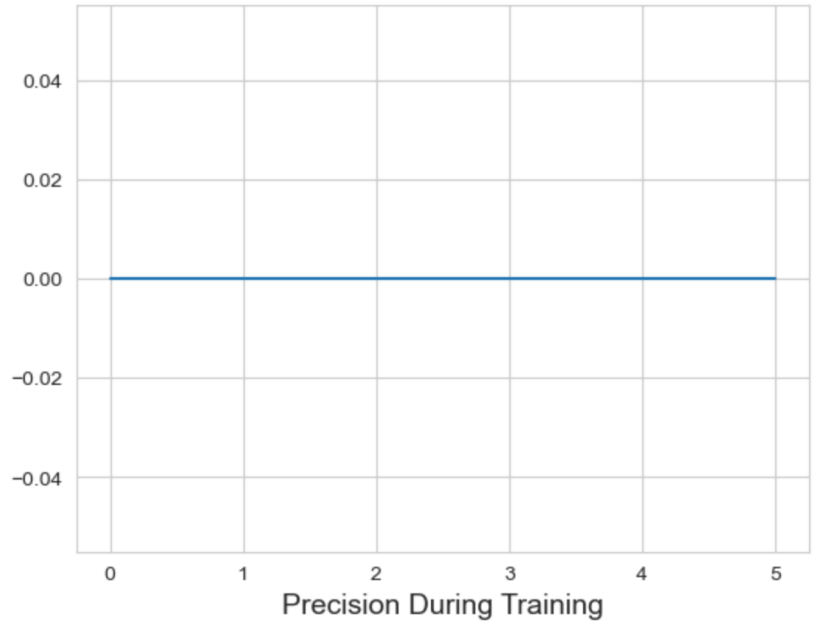
WEEK 10

Lab Logbook Requirement:

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

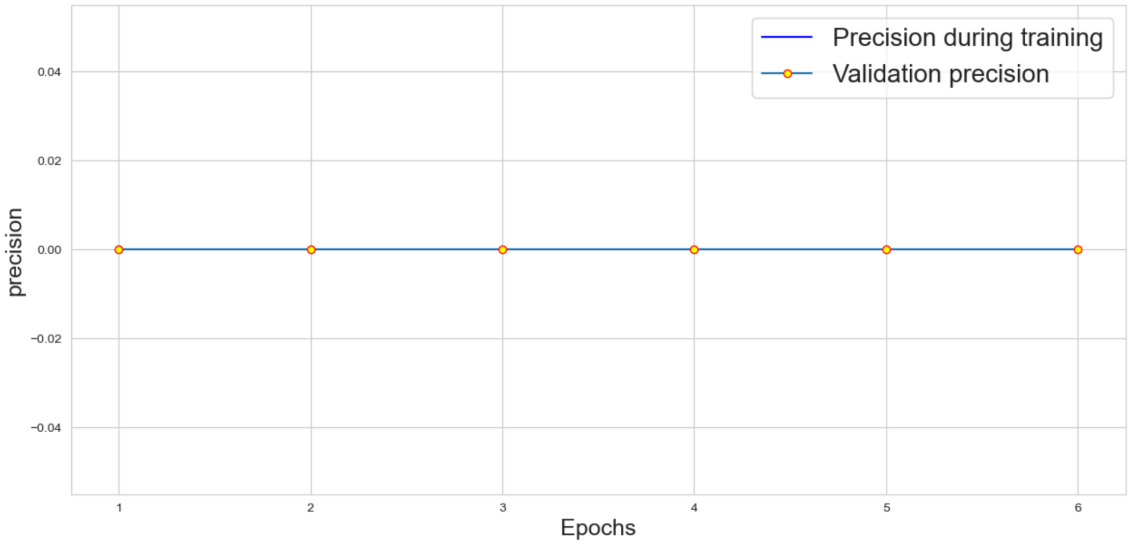
Precision during training

[253]:
```
plt.plot(history.history['precision'])
plt.xlabel('Precision During Training', size=14)
plt.show()
```



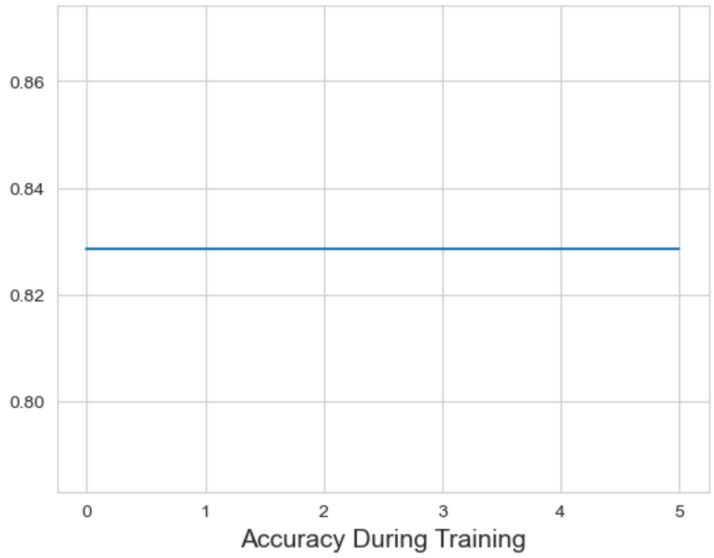
More detailed precision graph

[500]:
```
history_dict = history.history

precision = history_dict['precision']
val_precision = history_dict['val_precision']

epochs = range(1, len(precision) + 1)
plt.figure(num=1, figsize=(15,7))
plt.plot(epochs, precision, 'b', label='Precision during training')
plt.plot(epochs, val_precision, marker='o', markeredgecolor='red', markerfacecolor='yellow', label='Validation precision')
plt.xlabel('Epochs', size=18)
plt.ylabel('precision', size=18)
plt.legend()
plt.show()
```



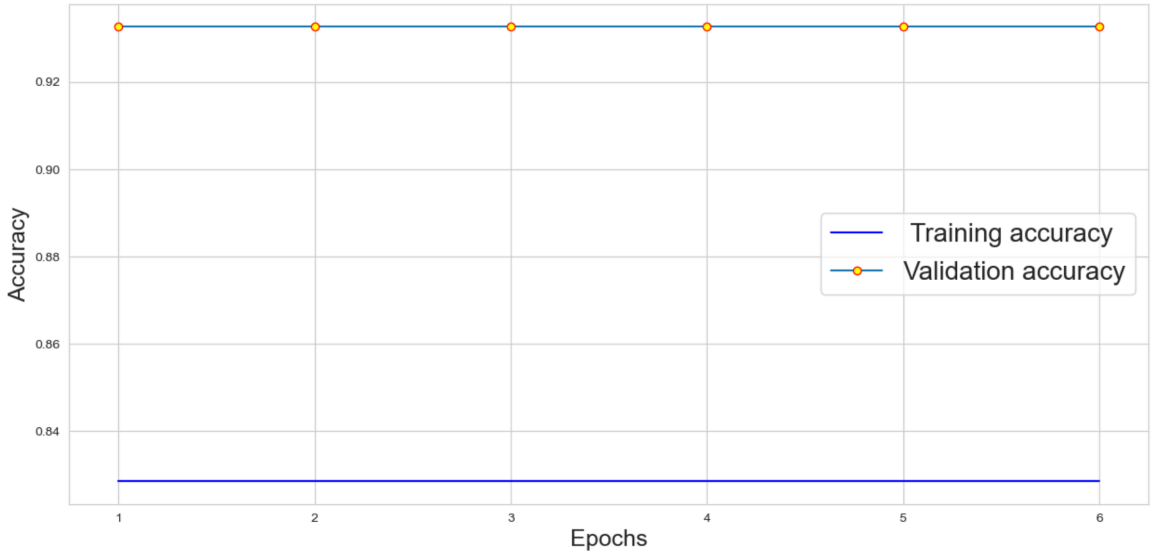
Training accuracy graph

[255]:
```
plt.plot(history.history['accuracy'])
plt.xlabel('Accuracy During Training', size=14)
plt.show()
```



More detailed accuracy graph

[502]:
```
history_dict = history.history

accuracy = history_dict['accuracy']
val_accuracy = history_dict['val_accuracy']

epochs = range(1, len(loss) + 1)
plt.figure(num=1, figsize=(15,7))
plt.plot(epochs, accuracy, 'b', label=' Training accuracy')
plt.plot(epochs, val_accuracy, marker='o', markeredgecolor='red', markerfacecolor='yellow', label='Validation accuracy')
plt.xlabel('Epochs', size=18)
plt.ylabel('Accuracy', size=18)
plt.legend()
plt.show()
```



WEEK 11

Lab Logbook Requirement:

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

[142]:
```
X_train = X_train.values if hasattr(X_train, 'values') else X_train
X_test = X_test.values if hasattr(X_test, 'values') else X_test

# Check the original shape
print("Original X_train shape:", X_train.shape)

# Reshape to add the time dimension
X_train = X_train.reshape(X_train.shape[0], 1, X_train.shape[1])
X_test = X_test.reshape(X_test.shape[0], 1, X_test.shape[1])

# Check the new shape
print("Reshaped X_train shape:", X_train.shape)


model = keras.Sequential([
    keras.layers.LSTM(100, activation='relu', input_shape=(X_train.shape[1], X_train.shape[2])),
    keras.layers.Dense(1, activation='sigmoid')

])
print(model.summary())
```

```
Original X_train shape: (25928, 10)
Reshaped X_train shape: (25928, 1, 10)
/Users/subhromukherjee/anaconda3/lib/python3.11/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)
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 100)	44,400
dense_4 (Dense)	(None, 1)	101

```
Total params: 44,501 (173.83 KB)
Trainable params: 44,501 (173.83 KB)
Non-trainable params: 0 (0.00 B)
None
```

[143]:
```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=["f1_score","recall","precision","accuracy","AUC"])
```

[144]:
```
# Early Stopping parameters

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

[145]:
```
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
1116/1167 ━━━━━━━━━━━━━━━━━━━━ 0s 768us/step - AUC: 0.5818 - accuracy: 0.7133 - f1_score: 0.3732 - loss: 63.3629 - precision: 0.3318 - recal
l: 0.3173
Epoch 1: val_loss improved from inf to 11.93167, saving model to best_model_LSTM.keras
1167/1167 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - AUC: 0.5831 - accuracy: 0.7141 - f1_score: 0.3748 - loss: 61.3967 - precision: 0.3338 - recall:
0.3189 - val_AUC: 0.5118 - val_accuracy: 0.7725 - val_f1_score: 0.4411 - val_loss: 11.9317 - val_precision: 1.0000 - val_recall: 0.0067
Epoch 2/10
1165/1167 ━━━━━━━━━━━━━━━━━━━━ 0s 956us/step - AUC: 0.6335 - accuracy: 0.7426 - f1_score: 0.4140 - loss: 7.1105 - precision: 0.4040 - recall:
0.3988
Epoch 2: val_loss improved from 11.93167 to 2.11123, saving model to best_model_LSTM.keras
1167/1167 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - AUC: 0.6335 - accuracy: 0.7425 - f1_score: 0.4140 - loss: 7.1104 - precision: 0.4040 - recall:
0.3988 - val_AUC: 0.7502 - val_accuracy: 0.8268 - val_f1_score: 0.4513 - val_loss: 2.1112 - val_precision: 0.6566 - val_recall: 0.5118
Epoch 3/10
1147/1167 ━━━━━━━━━━━━━━━━━━━━ 0s 792us/step - AUC: 0.6454 - accuracy: 0.7388 - f1_score: 0.4187 - loss: 5.3256 - precision: 0.4100 - recall:
0.4142
Epoch 3: val_loss improved from 2.11123 to 1.17082, saving model to best_model_LSTM.keras
1167/1167 ━━━━━━━━━━━━━━━━━━━━ 1s 859us/step - AUC: 0.6454 - accuracy: 0.7388 - f1_score: 0.4186 - loss: 5.3387 - precision: 0.4099 - recall:
0.4141 - val_AUC: 0.7868 - val_accuracy: 0.7397 - val_f1_score: 0.3777 - val_loss: 1.1708 - val_precision: 0.4565 - val_recall: 0.7155
Epoch 4/10
1108/1167 ━━━━━━━━━━━━━━━━━━━━ 0s 820us/step - AUC: 0.6379 - accuracy: 0.7297 - f1_score: 0.4136 - loss: 6.4366 - precision: 0.3937 - recall:
0.4049
Epoch 4: val_loss did not improve from 1.17082
1167/1167 ━━━━━━━━━━━━━━━━━━━━ 1s 876us/step - AUC: 0.6379 - accuracy: 0.7299 - f1_score: 0.4136 - loss: 6.4118 - precision: 0.3937 - recall:
0.4047 - val_AUC: 0.7994 - val_accuracy: 0.7964 - val_f1_score: 0.3918 - val_loss: 1.2173 - val_precision: 0.5481 - val_recall: 0.6330
Epoch 5/10
1153/1167 ━━━━━━━━━━━━━━━━━━━━ 0s 788us/step - AUC: 0.6506 - accuracy: 0.7417 - f1_score: 0.4188 - loss: 5.0866 - precision: 0.4143 - recall:
0.4245
Epoch 5: val_loss did not improve from 1.17082
1167/1167 ━━━━━━━━━━━━━━━━━━━━ 1s 845us/step - AUC: 0.6505 - accuracy: 0.7416 - f1_score: 0.4188 - loss: 5.0967 - precision: 0.4141 - recall:
0.4243 - val_AUC: 0.7356 - val_accuracy: 0.8261 - val_f1_score: 0.4775 - val_loss: 2.2865 - val_precision: 0.7314 - val_recall: 0.3805
Epoch 6/10
1156/1167 ━━━━━━━━━━━━━━━━━━━━ 0s 784us/step - AUC: 0.6476 - accuracy: 0.7415 - f1_score: 0.4242 - loss: 5.2207 - precision: 0.4085 - recall:
0.4152
Epoch 6: val_loss did not improve from 1.17082
```

[272]:
```
from sklearn.metrics import recall_score, f1_score, precision_score, accuracy_score, roc_auc_score

y_pred_LSTM = (model.predict(X_test))
y_pred_LSTM_labels = (y_pred_LSTM >= 0.5).astype(int)

imbalance_ai_models = ['Random Forest', 'Logistic Regression', 'MLP', 'LSTM']
prediction_imb = [y_pred_random, y_pred_log, y_pred_ANN, y_pred_LSTM]

print("Type of prediction_imb:", type(prediction_imb))
print("Type of each element in prediction_imb:", type(prediction_imb[0]))

threshold = 0.5
prediction_imb = [np.where(pred >= threshold, 1, 0) for pred in prediction_imb]

accuracy_imb = []
precision_imb = []
recall_imb =[]
f1_imb = []
auc_imb = []
for x in range(0, len(prediction_imb)):
    acc_score = np.round(accuracy_score(y_test, prediction_imb[x]) * 100, 2)
    accuracy_imb.append(acc_score)
    pre_score = np.round(precision_score(y_test, prediction_imb[x], average='weighted') * 100, 2)
    precision_imb.append(pre_score)
    rc_score = np.round(recall_score(y_test, prediction_imb[x], average='weighted') * 100, 2)
    recall_imb.append(rc_score)
    f_score = np.round(f1_score(y_test, prediction_imb[x], average='weighted') * 100, 2)
    f1_imb.append(f_score)
    auc_sc = np.round(roc_auc_score(y_test, prediction_imb[x]) * 100, 2)
    auc_imb.append(auc_sc)

model_perform_imb  = {'Imbalance AI Models':imbalance_ai_models,
              'Accuracy':accuracy_imb,
               'Precision':precision_imb,
               'Recall':recall_imb,
               'F1-Score':f1_imb,
               'AUC':auc_imb
              }
model_data_imb  = pd.DataFrame(model_perform_imb)
print('Result of testing with an Imbalanced Dataset')
print(model_data_imb)
```

```
model_data_imb.set_index('Imbalance AI Models').plot(
    kind='bar', figsize=(14, 6), width=0.6
)
plt.title('ML Models performance on the predicton of credit risk on imbalanaced dataset')
plt.ylabel('Performance')
plt.xlabel('ML Models')
plt.xticks(rotation=0)
plt.legend(loc='lower right')
plt.show()
```

```
203/203 ━━━━━━━━━━━━━━━━━━━━ 0s 375us/step
Type of prediction_imb: <class 'list'>
Type of each element in prediction_imb: <class 'numpy.ndarray'>
Result of testing with an Imbalanced Dataset
   Imbalance AI Models  Accuracy  Precision  Recall  F1-Score    AUC
0        Random Forest     91.38      91.58   91.38     90.75  81.60
1  Logistic Regression     83.93      82.69   83.93     82.32  69.00
2                  MLP     89.36      88.99   89.36     88.79  79.90
3                 LSTM     82.42      81.96   82.42     78.62  61.72
```