Week 1

Code

```
import numpy as np
a = np.arange(68)
print("vector a:", a)
a_2d = a.reshape(1, -1)
print("\n2D Array with 1 row:\n", a_2d)
b = a_2d.copy()
print("\ncopied array:\n", b)
print("\nShape of array b:", b.shape)
```

Output

```
[ ]: import numpy as np

[ ]: a = np.arange(68)
     print("vector a:", a)

vector a: [ 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]

[ ]: a_2d = a.reshape(1, -1)
     print("\n2D Array with 1 row:\n", a_2d)

2D Array with 1 row:
 [[ 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]]

[ ]: b = a_2d.copy()
     print("\ncopied array:\n", b)

copied 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]]

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

Shape of array b: (1, 68)
```

Week 2

Code

```
import pandas as pd

data = pd.DataFrame({
    "relationship": ["Husband", "Wife", "Not-in-family", "Own-child", "Husband", "Wife"],
    "hours-per-week": [40, 50, 60, 20, 55, 45]
})
```

```
n = 8

def reduce_hours(x):
    return x - n

data["reduced_hours"] = data["hours-per-week"].apply(reduce_hours)

grouped = data.groupby(["relationship", "reduced_hours"]).size().reset_index(name="count")

print("Original DataFrame:")
print(data, "\n")

print("Grouped DataFrame:")
print(grouped)
```

Output

```
Original DataFrame:
    relationship  hours-per-week  reduced_hours
0        Husband              40             32
1           Wife              50             42
2  Not-in-family              60             52
3      Own-child              20             12
4        Husband              55             47
5           Wife              45             37

Grouped DataFrame:
    relationship  reduced_hours  count
0        Husband             32      1
1        Husband             47      1
2  Not-in-family             52      1
3      Own-child             12      1
4           Wife             37      1
5           Wife             42      1
```

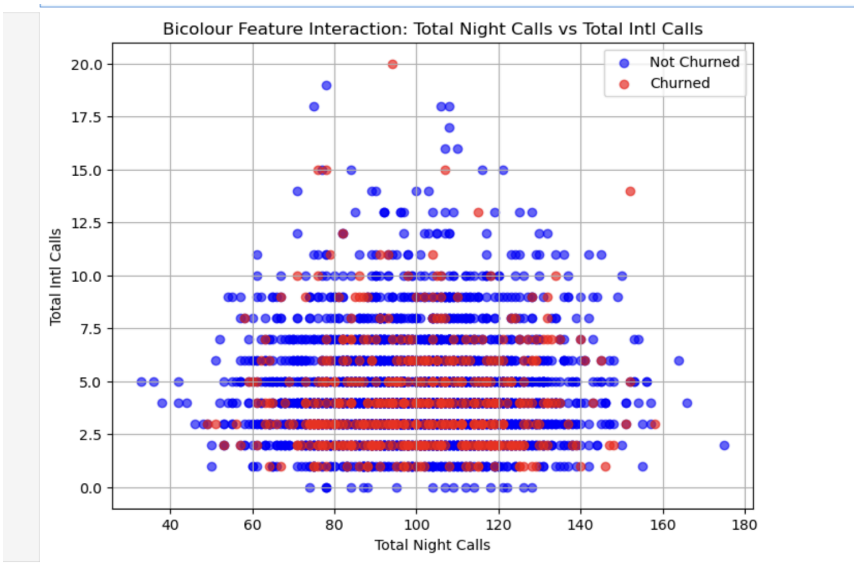
Week 3

Code

```
import os
print(os.getcwd())

import pandas as pd
```

```
file_path = "/Users/srinivas/Downloads/telecom_churn.csv"
data = pd.read_csv(file_path)

print("CSV loaded successfully!")
print(data.head())

import pandas as pd
import matplotlib.pyplot as plt

# Load dataset
df = pd.read_csv("telecom_churn.csv")

# Correct column names (case-sensitive)
x = df['Total night calls']
y = df['Total intl calls']
churn_col = 'Churn'

# Create bicolour scatter plot
plt.figure(figsize=(8,6))
plt.scatter(x[df[churn_col] == False], y[df[churn_col] == False],
            color='blue', label='Not Churned', alpha=0.6)
plt.scatter(x[df[churn_col] == True], y[df[churn_col] == True],
            color='red', label='Churned', alpha=0.6)

plt.title('Bicolour Feature Interaction: Total Night Calls vs Total Intl Calls')
plt.xlabel('Total Night Calls')
plt.ylabel('Total Intl Calls')
plt.legend()
plt.grid(True)
plt.show()
```

Output

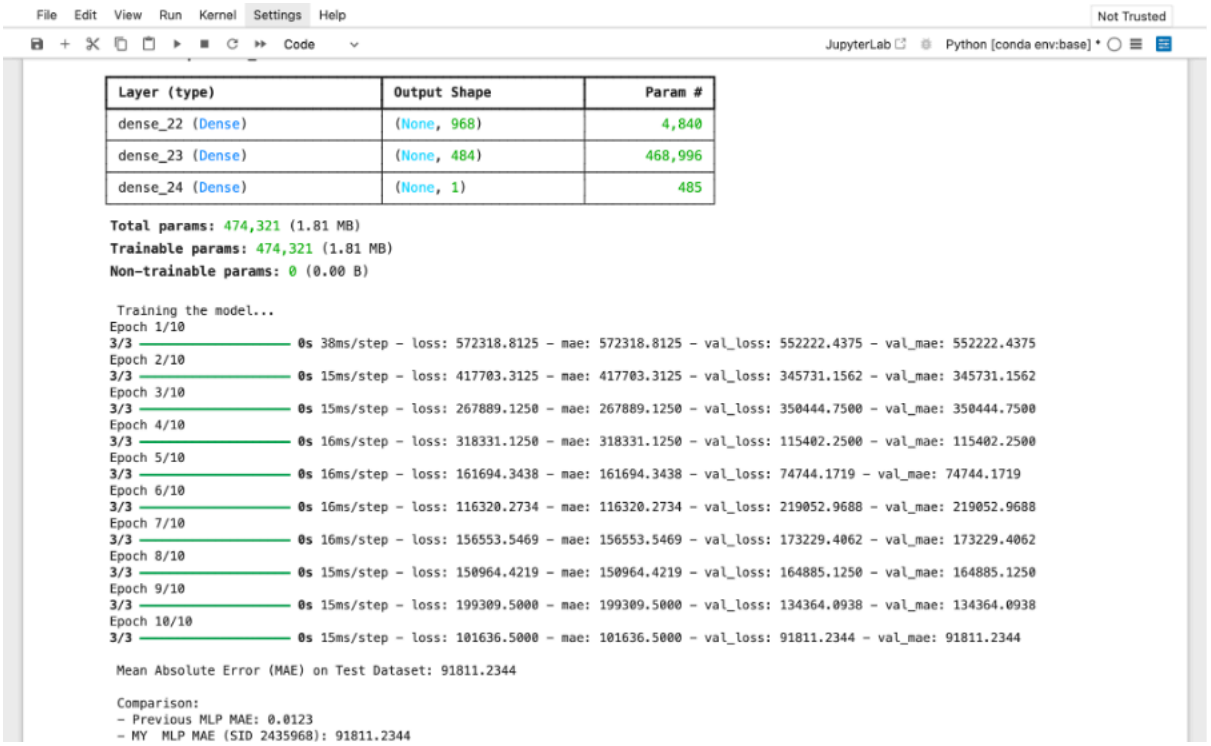


WEEK 4

CODE

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam

model_custom = Sequential([
    Dense(968, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(484, activation='relu'),
    Dense(1, activation='linear')
])

model_custom.compile(
    optimizer=Adam(learning_rate=0.001),  # same as practical session
    loss='mae',
    metrics=['mae']
)

print("\n Model Architecture Summary:")
model_custom.summary()

print("\n Training the model...")
history_custom = model_custom.fit(
    X_train, y_train,
    validation_data=(X_test, y_test),
    epochs=10,
    batch_size=32,
    verbose=1
)

test_loss, test_mae = model_custom.evaluate(X_test, y_test, verbose=0)
print("\n Mean Absolute Error (MAE) on Test Dataset:", round(test_mae, 4))

previous_mae = 0.0123

print("\n Comparison:")
print(f" - Previous MLP MAE: {previous_mae}")
print(f" - My MLP (SID 2435968): {round(test_mae, 4)}")
```

OUTPUT



WEEK 5

```
import tensorflow as tf
from tensorflow import keras
import numpy as np

model = keras.Sequential([
    keras.Input(shape=(50, 5)),
    keras.layers.Conv1D(50, 5, padding='same', activation='relu', kernel_initializer="normal"),
    keras.layers.MaxPooling1D(7),
    keras.layers.Conv1D(100, 7, padding='same', activation='relu',
kernel_initializer="normal"),
    keras.layers.GlobalMaxPooling1D(),
    keras.layers.Dense(25, activation='relu', kernel_initializer="normal"),
    keras.layers.Dense(2)
])

model.compile(optimizer='adam', loss='mse', metrics=['mae'])
model.summary()

X_train, y_train, X_test, y_test = map(np.array, [X_train, y_train, X_test, y_test])

min_len_train = min(len(X_train), len(y_train))
min_len_test = min(len(X_test), len(y_test))

X_train, y_train = X_train[:min_len_train], y_train[:min_len_train]
```

```
X_test, y_test = X_test[:min_len_test], y_test[:min_len_test]

X_train = X_train.reshape(-1, 50, 5)
X_test = X_test.reshape(-1, 50, 5)
y_train = y_train.reshape(-1, 2)
y_test = y_test.reshape(-1, 2)

history = model.fit(
    X_train, y_train,
    epochs=14,
    batch_size=50,
    validation_data=(X_test, y_test),
    verbose=1
)
```

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)	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: 38,977 (152.25 KB)
Trainable params: 38,977 (152.25 KB)
Non-trainable params: 0 (0.00 B)

```
Epoch 1/14
4400/4400 ━━━━━━━━━━━━━━━━━━━━ 10s 2ms/step - loss: 0.0084 - mae: 0.0446 - val_loss: 6.9367e-04 - val_mae: 0.0186
Epoch 2/14
4400/4400 ━━━━━━━━━━━━━━━━━━━━ 10s 2ms/step - loss: 7.9914e-04 - mae: 0.0191 - val_loss: 6.2007e-04 - val_mae: 0.0169
Epoch 3/14
4400/4400 ━━━━━━━━━━━━━━━━━━━━ 9s 2ms/step - loss: 7.4834e-04 - mae: 0.0184 - val_loss: 6.3785e-04 - val_mae: 0.0176
Epoch 4/14
4400/4400 ━━━━━━━━━━━━━━━━━━━━ 10s 2ms/step - loss: 7.3505e-04 - mae: 0.0181 - val_loss: 6.2307e-04 - val_mae: 0.0169
Epoch 5/14
4400/4400 ━━━━━━━━━━━━━━━━━━━━ 10s 2ms/step - loss: 7.3505e-04 - mae: 0.0182 - val_loss: 6.1726e-04 - val_mae: 0.0170
Epoch 6/14
4400/4400 ━━━━━━━━━━━━━━━━━━━━ 9s 2ms/step - loss: 7.4049e-04 - mae: 0.0180 - val_loss: 5.9040e-04 - val_mae: 0.0163
Epoch 7/14
4400/4400 ━━━━━━━━━━━━━━━━━━━━ 9s 2ms/step - loss: 7.2744e-04 - mae: 0.0179 - val_loss: 5.9399e-04 - val_mae: 0.0162
Epoch 8/14
4400/4400 ━━━━━━━━━━━━━━━━━━━━ 10s 2ms/step - loss: 7.1088e-04 - mae: 0.0178 - val_loss: 6.0743e-04 - val_mae: 0.0167
Epoch 9/14
4400/4400 ━━━━━━━━━━━━━━━━━━━━ 9s 2ms/step - loss: 7.1939e-04 - mae: 0.0178 - val_loss: 5.9310e-04 - val_mae: 0.0162
Epoch 10/14
4400/4400 ━━━━━━━━━━━━━━━━━━━━ 9s 2ms/step - loss: 7.2684e-04 - mae: 0.0178 - val_loss: 6.2682e-04 - val_mae: 0.0171
Epoch 11/14
4400/4400 ━━━━━━━━━━━━━━━━━━━━ 10s 2ms/step - loss: 7.0216e-04 - mae: 0.0176 - val_loss: 6.1509e-04 - val_mae: 0.0170
Epoch 12/14
4400/4400 ━━━━━━━━━━━━━━━━━━━━ 10s 2ms/step - loss: 7.1237e-04 - mae: 0.0178 - val_loss: 6.2858e-04 - val_mae: 0.0169
Epoch 13/14
4400/4400 ━━━━━━━━━━━━━━━━━━━━ 9s 2ms/step - loss: 7.1782e-04 - mae: 0.0177 - val_loss: 5.8923e-04 - val_mae: 0.0162
Epoch 14/14
4400/4400 ━━━━━━━━━━━━━━━━━━━━ 10s 2ms/step - loss: 7.0961e-04 - mae: 0.0177 - val_loss: 5.9114e-04 - val_mae: 0.0162
```

```
mse, mae = model.evaluate(X_test, y_test, verbose=1)
print("Test Mean Absolute Error (MAE): %.5f" % mae)
```

```
936/936 ━━━━━━━━━━━━━━━━━━━━ 1s 659us/step - loss: 5.3258e-04 - mae: 0.0154
Test Mean Absolute Error (MAE): 0.01621
```

Week 6

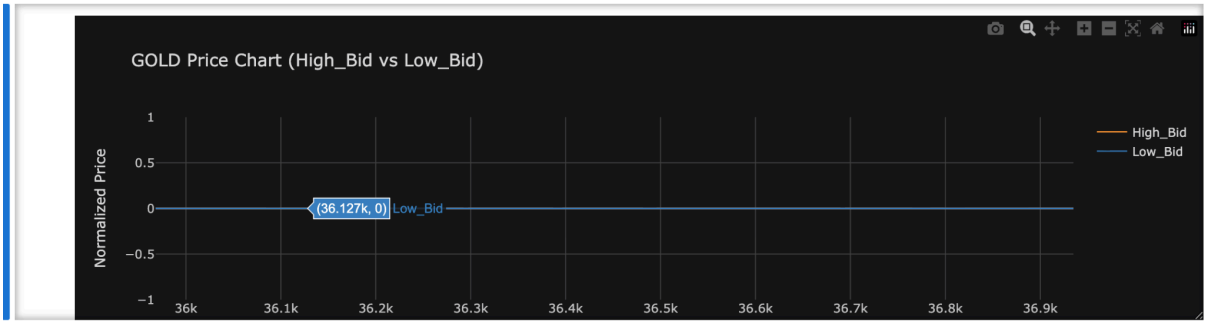
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 354694 entries, 0 to 354693
Data columns (total 17 columns):
 #   Column                     Non-Null Count   Dtype
---  ------                     --------------   -----
 0   Open_Bid                   354694 non-null  float32
 1   High_Bid                   354694 non-null  float32
 2   Low_Bid                    354694 non-null  float32
 3   Close_Bid                  354694 non-null  float32
 4   Volume_Bid                 354694 non-null  float32
 5   Volume_Ask                 354694 non-null  float32
 6   Volume_Delta               354693 non-null  float32
 7   Volume_Delta_abs           354694 non-null  float32
 8   Open_Delta                 354694 non-null  float32
 9   High_Delta                 354694 non-null  float32
 10  Low_Delta                  354694 non-null  float32
 11  Close_Delta                354694 non-null  float32
 12  Local_time_T_shift_1_Down  354693 non-null  datetime64[ns, UTC]
 13  New_day                    354694 non-null  int8
 14  New_week                   354694 non-null  int8
 15  Y _High_Bid                354694 non-null  float64
 16  Y_Low_Ask                  354694 non-null  float32
dtypes: datetime64[ns, UTC](1), float32(13), float64(1), int8(2)
memory usage: 23.7 MB
```

Code

```
import pandas as pd
import cufflinks as cf
cf.go_offline()

data2 = pd.read_csv('GOLD_2022_normalised.csv')

for col in data2.columns:
    try:
        data2[col] = pd.to_numeric(data2[col])
    except:
```

```
        pass

numeric_cols = data2.select_dtypes(include='number').columns

min_price = data2[numeric_cols].min().min()
max_price = data2[numeric_cols].max().max()

data2[numeric_cols] = (data2[numeric_cols] - min_price) / (max_price - min_price)

data_part = data2.iloc[35968 : 35968 + 968]

data_part.rename(columns={'High_x': 'High_Bid', 'Low_x': 'Low_Bid'}, inplace=True)

data_part[['High_Bid', 'Low_Bid']].iplot(
    title='GOLD Price Chart (High_Bid vs Low_Bid)',
    xTitle='Time (UTC)',
    yTitle='Normalized Price',
    theme='solar'
)
```

Output



Week 7

CODE

```
model = keras.Sequential([
    keras.layers.LSTM(78, activation='relu', input_shape=(50, 18)),
    keras.layers.Dense(2)
])

print(model.summary())
model.compile(optimizer="adam", loss="mse", metrics=["mae"])
# Early Stopping parameters

es = EarlyStopping(monitor='val_loss', mode='min', patience=3, verbose=1)
```

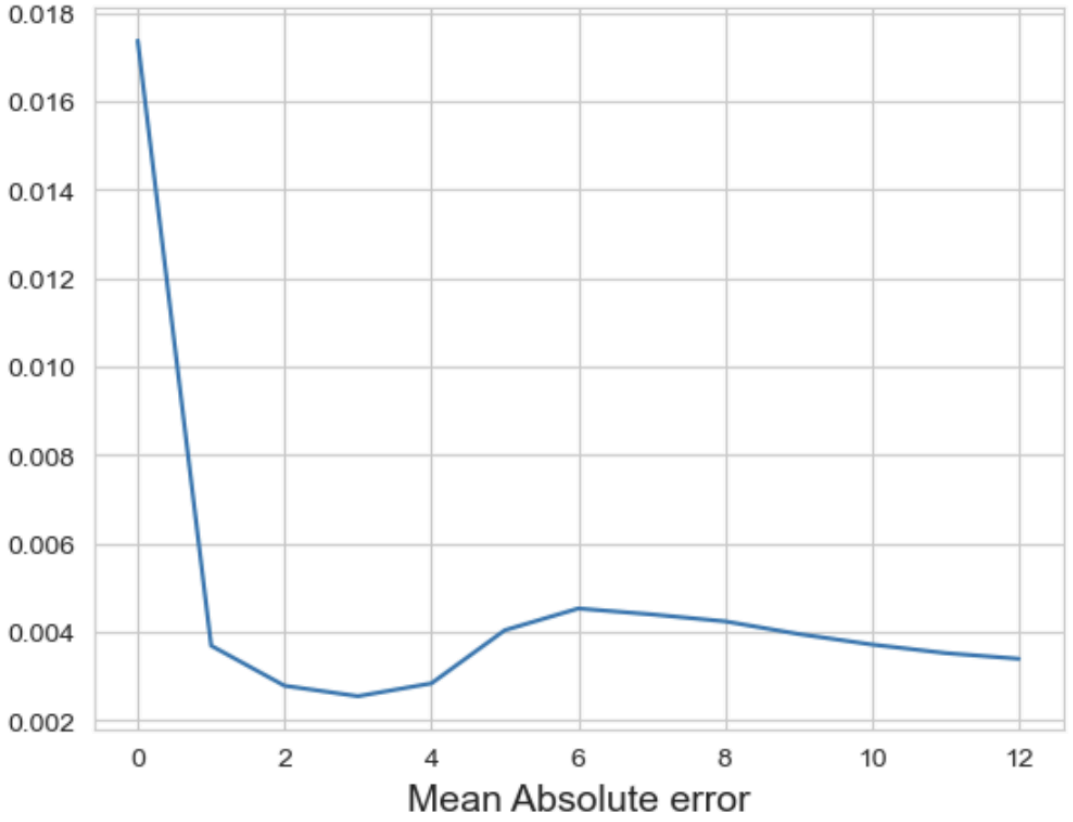
```
mc = ModelCheckpoint('best_model_LSTM_GOLD.keras', monitor='val_loss', mode='min',
verbose=1, save_best_only=True)
history = model.fit(
    X_train, y_train,
    batch_size=20,
    epochs=10,
    validation_split=0.1,
    shuffle=True,
    verbose=1,
    callbacks=[es, mc]
)
```

Model: "sequential_1"

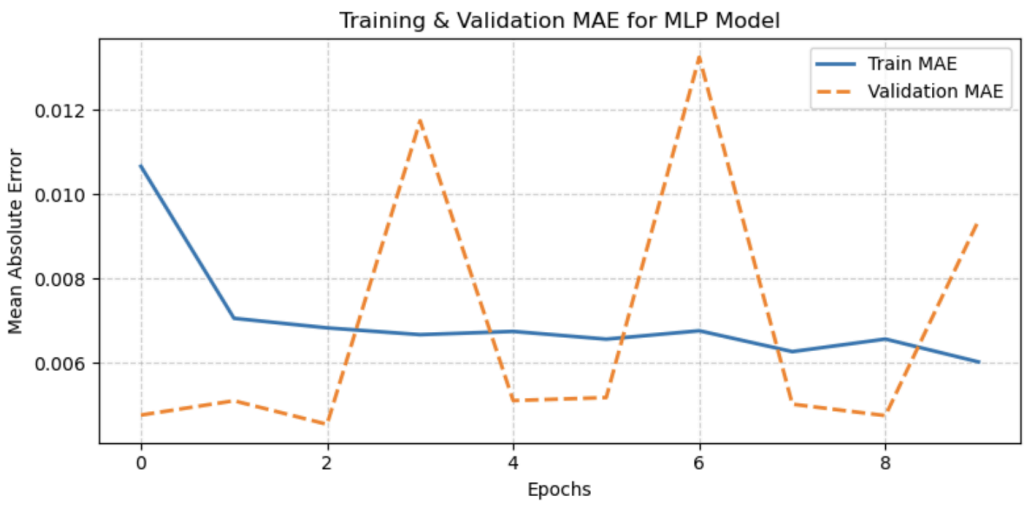
Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 78)	30,264
dense_1 (Dense)	(None, 2)	158

Total params: 30,422 (118.84 KB)
Trainable params: 30,422 (118.84 KB)
Non-trainable params: 0 (0.00 B)

```
None
Epoch 1/10
1208/1213 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0741 - mae: 0.0708
Epoch 1: val_loss improved from inf to 0.00006, saving model to best_model_LSTM_GOLD.keras
1213/1213 ━━━━━━━━━━━━━━━━━━━━ 11s 8ms/step - loss: 0.0737 - mae: 0.0705 - val_loss: 6.1189e-05 - val_mae: 0.0063
Epoch 2/10
1213/1213 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 2.6162e-05 - mae: 0.0038
Epoch 2: val_loss improved from 0.00006 to 0.00001, saving model to best_model_LSTM_GOLD.keras
1213/1213 ━━━━━━━━━━━━━━━━━━━━ 9s 8ms/step - loss: 2.6158e-05 - mae: 0.0038 - val_loss: 1.1086e-05 - val_mae: 0.0023
Epoch 3/10
1209/1213 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 1.2316e-05 - mae: 0.0026
Epoch 3: val_loss did not improve from 0.00001
1213/1213 ━━━━━━━━━━━━━━━━━━━━ 9s 7ms/step - loss: 1.2316e-05 - mae: 0.0026 - val_loss: 5.1869e-05 - val_mae: 0.0058
Epoch 4/10
1210/1213 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 1.4715e-05 - mae: 0.0029
Epoch 4: val_loss improved from 0.00001 to 0.00001, saving model to best_model_LSTM_GOLD.keras
1213/1213 ━━━━━━━━━━━━━━━━━━━━ 9s 7ms/step - loss: 1.4720e-05 - mae: 0.0029 - val_loss: 5.8150e-06 - val_mae: 0.0017
Epoch 5/10
1211/1213 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 2.4375e-05 - mae: 0.0037
Epoch 5: val_loss did not improve from 0.00001
1213/1213 ━━━━━━━━━━━━━━━━━━━━ 9s 7ms/step - loss: 2.4382e-05 - mae: 0.0037 - val_loss: 4.2092e-05 - val_mae: 0.0063
Epoch 6/10
1208/1213 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 3.4058e-05 - mae: 0.0046
Epoch 6: val_loss did not improve from 0.00001
1213/1213 ━━━━━━━━━━━━━━━━━━━━ 9s 7ms/step - loss: 3.4050e-05 - mae: 0.0046 - val_loss: 6.2009e-06 - val_mae: 0.0020
Epoch 7/10
1213/1213 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 3.3385e-05 - mae: 0.0045
Epoch 7: val_loss did not improve from 0.00001
1213/1213 ━━━━━━━━━━━━━━━━━━━━ 9s 7ms/step - loss: 3.3382e-05 - mae: 0.0045 - val_loss: 9.0448e-06 - val_mae: 0.0024
Epoch 7: early stopping
```


0.018
0.016
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0
2
4
6
8
10
12
Mean Absolute error

WEEK 8


Training & Validation MAE for MLP Model
Train MAE
Validation MAE
Mean Absolute Error
0.012
0.010
0.008
0.006
0
2
4
6
8
Epochs

Week 10

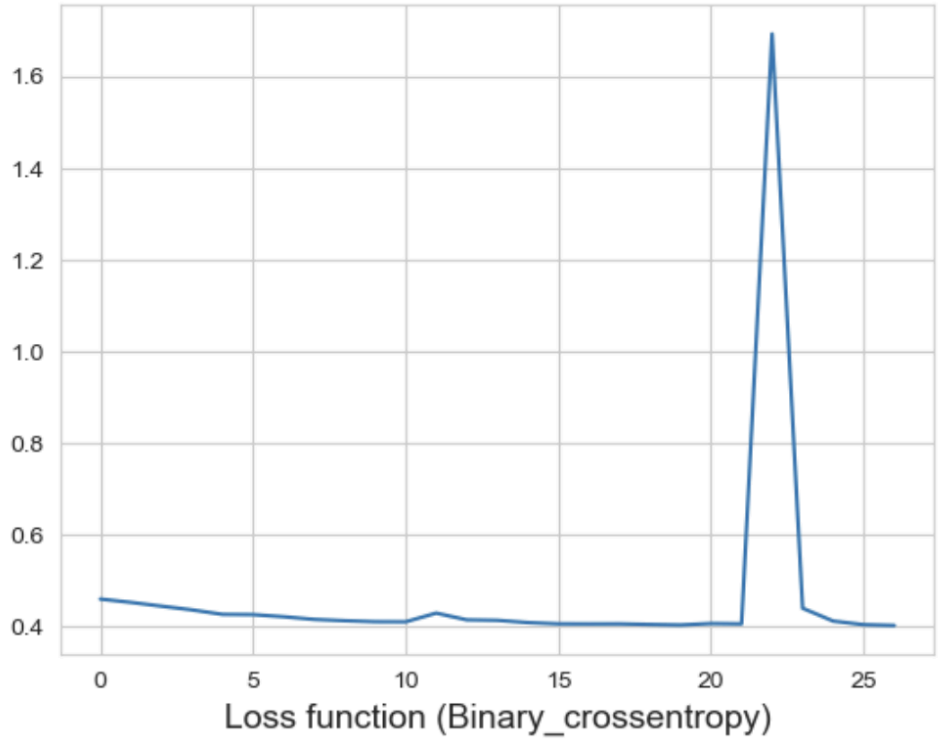
CODE

```
print(X_train.shape)
print(y_train.shape)
class_weight_dict
len(X_train)

history=model.fit(X_train, y_train, batch_size=18, epochs=38,
validation_split=0.2,shuffle=True,verbose=1,callbacks=[es, mc])
  # SID = 2435968 (Z=8)
```

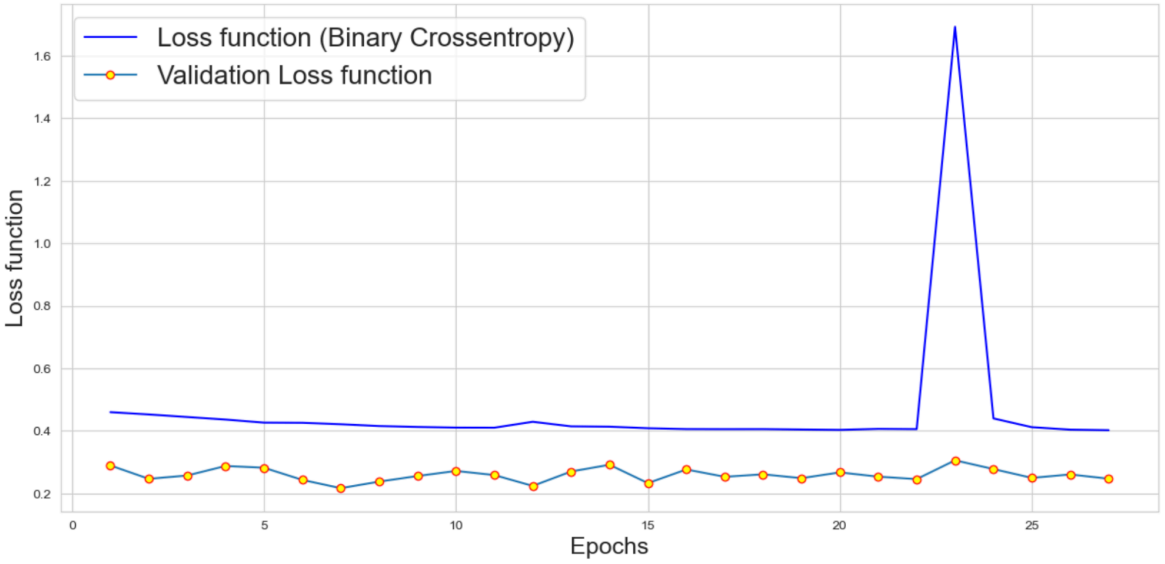
```
478/478 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.8351 - loss: 0.4060 - val_accuracy: 0.9135 - val_loss: 0.2612
Epoch 19/38
467/478 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8367 - loss: 0.4071
Epoch 19: val_loss did not improve from 0.21691
478/478 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.8366 - loss: 0.4070 - val_accuracy: 0.9233 - val_loss: 0.2489
Epoch 20/38
467/478 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8354 - loss: 0.4030
Epoch 20: val_loss did not improve from 0.21691
478/478 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.8354 - loss: 0.4030 - val_accuracy: 0.9252 - val_loss: 0.2673
Epoch 21/38
478/478 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8381 - loss: 0.4031
Epoch 21: val_loss did not improve from 0.21691
478/478 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - accuracy: 0.8380 - loss: 0.4031 - val_accuracy: 0.9317 - val_loss: 0.2537
Epoch 22/38
475/478 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8329 - loss: 0.4082
Epoch 22: val_loss did not improve from 0.21691
478/478 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.8329 - loss: 0.4081 - val_accuracy: 0.9256 - val_loss: 0.2459
Epoch 23/38
477/478 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8310 - loss: 1.6686
Epoch 23: val_loss did not improve from 0.21691
478/478 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.8310 - loss: 1.6687 - val_accuracy: 0.9326 - val_loss: 0.3060
Epoch 24/38
471/478 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8334 - loss: 0.4333
Epoch 24: val_loss did not improve from 0.21691
478/478 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.8333 - loss: 0.4334 - val_accuracy: 0.9326 - val_loss: 0.2780
Epoch 25/38
470/478 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8305 - loss: 0.4192
Epoch 25: val_loss did not improve from 0.21691
478/478 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.8305 - loss: 0.4191 - val_accuracy: 0.9326 - val_loss: 0.2498
Epoch 26/38
472/478 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8305 - loss: 0.4111
Epoch 26: val_loss did not improve from 0.21691
478/478 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8305 - loss: 0.4110 - val_accuracy: 0.9284 - val_loss: 0.2608
Epoch 27/38
476/478 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8284 - loss: 0.4074
Epoch 27: val_loss did not improve from 0.21691
478/478 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.8284 - loss: 0.4073 - val_accuracy: 0.9293 - val_loss: 0.2472
Epoch 27: early stopping
```

```
LSTM_saved_best_model =
keras.models.load_model('best_Trand-Flat_model_LSTM_GOLD_$10.keras')

scors=LSTM_saved_best_model.evaluate(X_test, y_test, verbose=1)

# Training Loss function

plt.plot(history.history['loss'])
plt.xlabel('Loss function (Binary_crossentropy)',size=14)
plt.show()
```

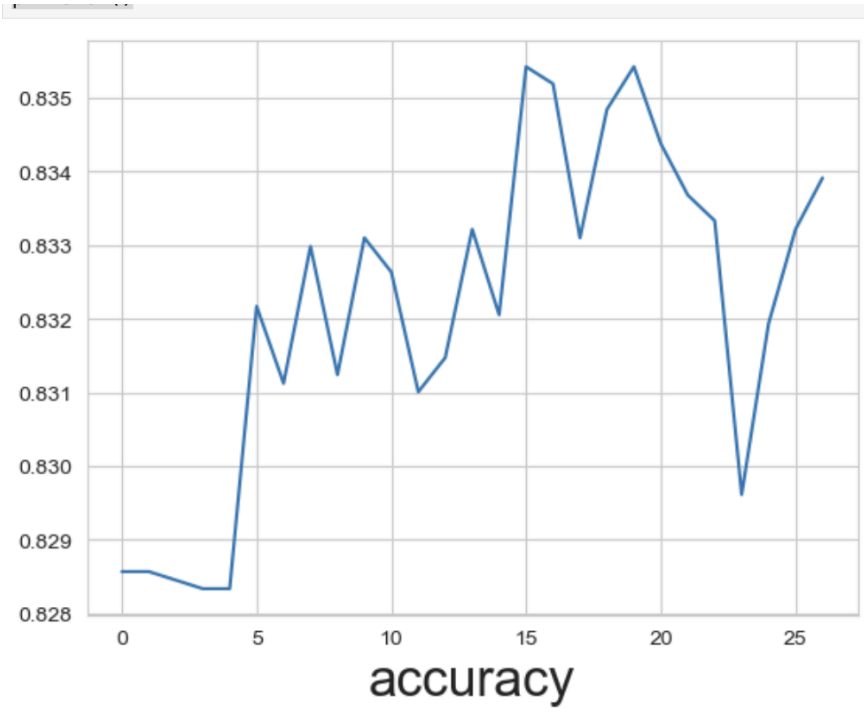
[408]:
```
# Training Loss function

plt.plot(history.history['loss'])
plt.xlabel('Loss function (Binary_crossentropy)',size=14)
plt.show()
```




```
# More detailed Loss function graph

history_dict = history.history

loss = history_dict['loss']
val_loss = history_dict['val_loss']

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

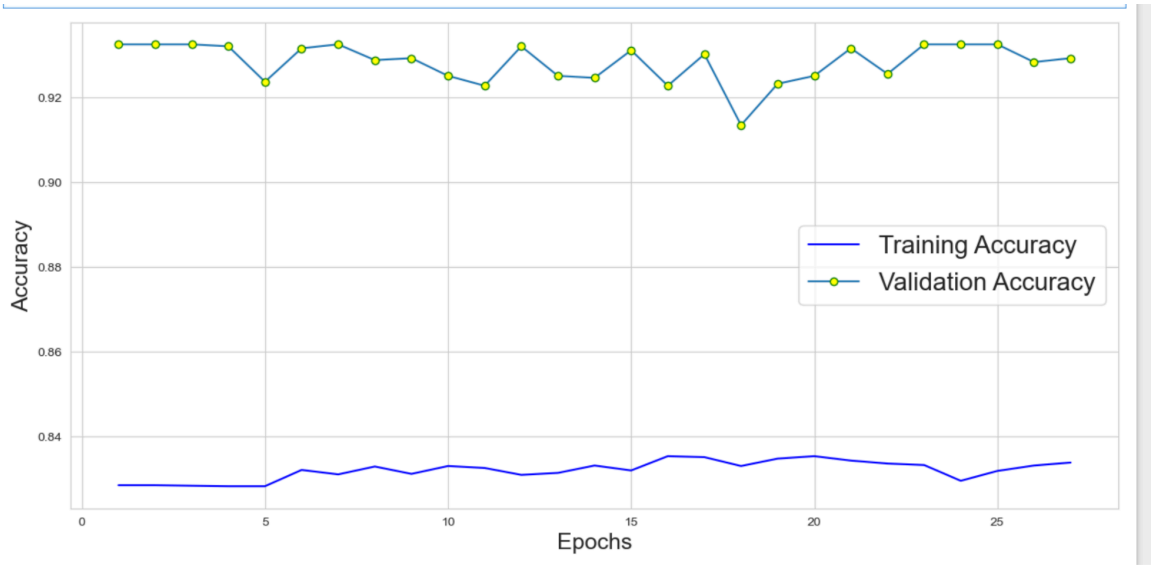


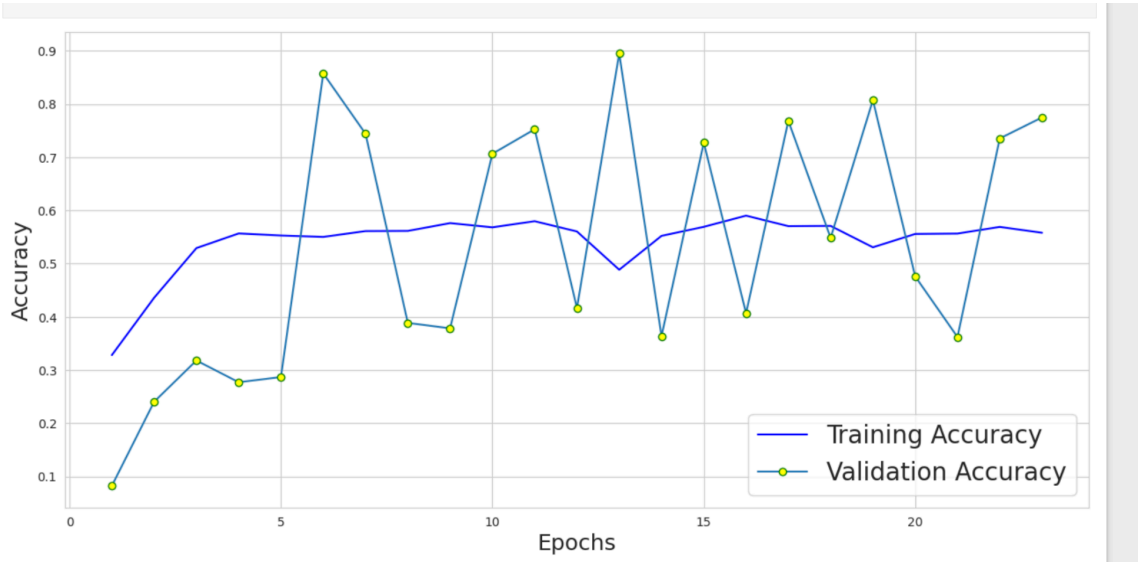

```
plt.plot(history.history['accuracy'])
plt.xlabel('accuracy', size=24)
plt.show()
```




```
y_pred = model.predict(X_test)
y_pred_labels = np.argmax(y_pred, axis=1)
y_true_labels = np.argmax(y_test, axis=1)


history_dict=history.history

Accuracy_values= history_dict['accuracy'][:]
val_accuracy_values=history_dict['val_accuracy'][:]

epochs=range(1, len(Accuracy_values)+1)
```

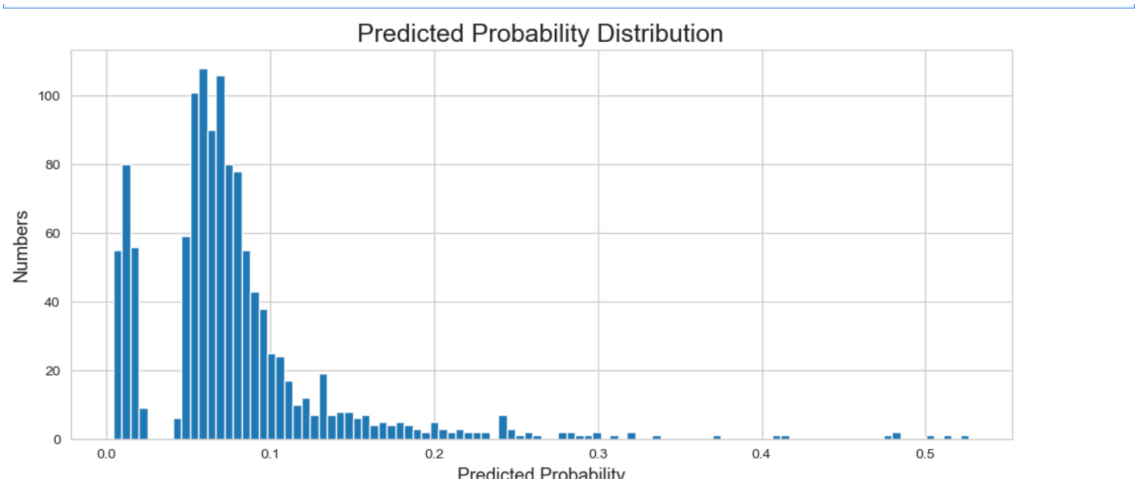
```
plt.figure(num=1, figsize=(15,7))
plt.plot(epochs, Accuracy_values, 'b', label='Training Accuracy')
plt.plot(epochs, val_accuracy_values, marker='o', markeredgecolor='green',
markerfacecolor='yellow', label='Validation Accuracy')
plt.xlabel('Epochs', size=18)
plt.ylabel('Accuracy', size=18)
plt.legend()
plt.show()
```






```
import random

pred = LSTM_saved_best_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])

plt.figure(num=1,figsize=(12,5))
plt.hist(pred,bins=100)
plt.title('Predicted Probability Distribution',size=18)
plt.ylabel('Numbers',size=13)
plt.xlabel('Predicted Probability',size=13)
plt.show();
```




```
from keras_tuner import RandomSearch
from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout
from keras.optimizers import AdamW
def build_model(hp):
    model = Sequential()
    model.add(LSTM(units=hp.Int('units', min_value=20, max_value=100, step=20),
            return_sequences=True, input_shape=(timesteps, features)))
    model.add(Dropout(hp.Float('dropout', min_value=0.1, max_value=0.5, step=0.1)))
    model.add(LSTM(units=hp.Int('units_2', min_value=10, max_value=50, step=10)))
    model.add(Dense(1, activation='sigmoid'))

    model.compile(optimizer=AdamW(learning_rate=hp.Choice('learning_rate', values=[1e-2,
1e-3, 1e-4])),
                loss='binary_crossentropy',
                metrics=['accuracy'])
    return model

tuner = RandomSearch(
    build_model,
    objective='val_accuracy',
    max_trials=5,          # How many different models to try
    #max_trials=20,          # Change for reseach
```

```
    executions_per_trial=3,  # How many times to train and test each model to average the
result
    directory='my_dir',
    project_name='lstm_tuning'
)
tuner.search(X_train, y_train,
             epochs=5,
             #epochs=25,   # Change for reseach
             validation_split=0.2, shuffle=True,
             verbose=1,
             #callbacks=[es, mc],
             class_weight=class_weight_dict)
```

Output

```
[473]: best_model = tuner.get_best_models(num_models=1)[0]
```

```
[474]: print(best_model.summary())
```

Model: "sequential"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 50, 40)	9,440
dropout (Dropout)	(None, 50, 40)	0
lstm_1 (LSTM)	(None, 30)	8,520
dense (Dense)	(None, 1)	31

Total params: 17,991 (70.28 KB)
Trainable params: 17,991 (70.28 KB)
Non-trainable params: 0 (0.00 B)
None