LAB Logbook

**Lab 1:**

import numpy as np

# SID

sid = "2420077"

# Extract last two digits

last\_two\_digits = int(sid[-2:]) # 77

# Determine number of elements

num\_elements = last\_two\_digits if last\_two\_digits >= 10 else last\_two\_digits + 100

# Create vector using np.arange

a = np.arange(num\_elements)

# Change to 2D array with 1 row

a\_2d = a.reshape(1, -1)

# Print the 2D array

print("2D array a\_2d:")

print(a\_2d)

# Save it in another array

b = a\_2d.copy()

# Print the new array

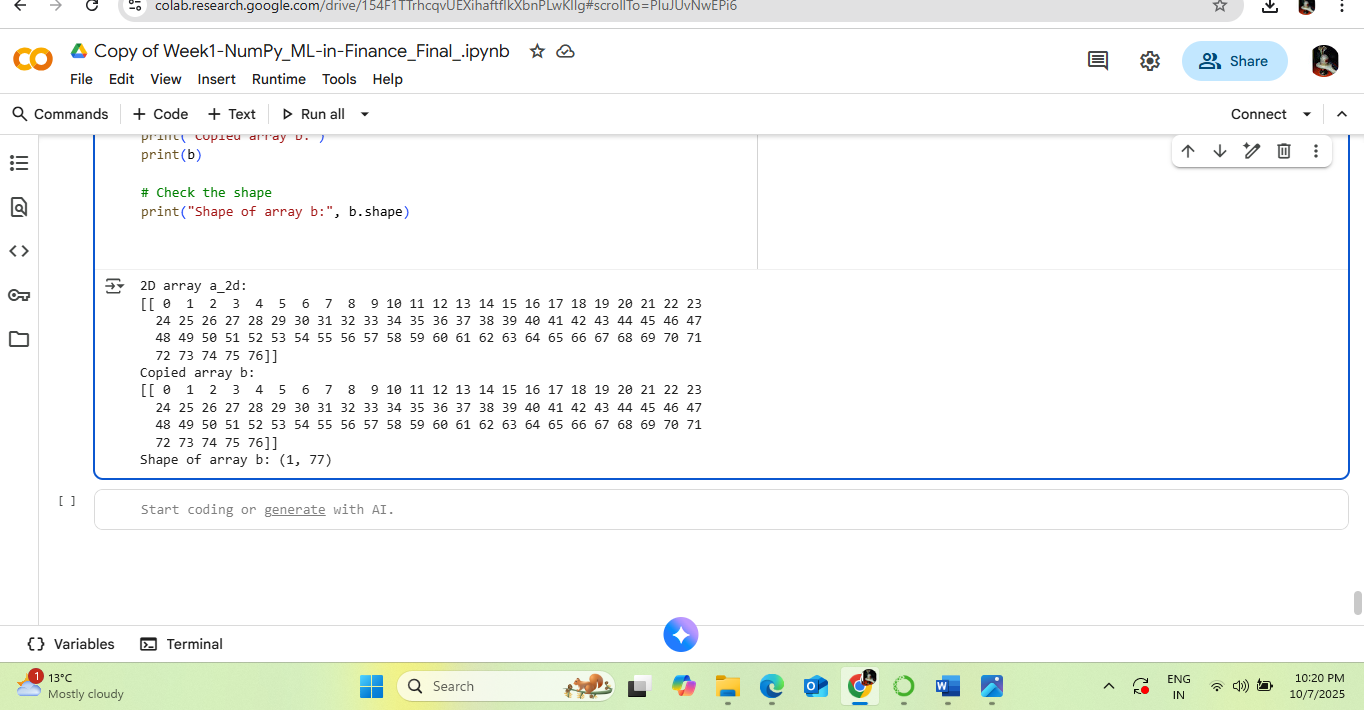
print("Copied array b:")

print(b)

# Check the shape

print("Shape of array b:", b.shape)

**Output:**



Lab 2:

import pandas as pd

# SID

sid = "2420077"

# Determine n from last digit

n = int(sid[-1])

n = 10 if n == 0 else n

# Load dataset (assuming data is already loaded from 'adult\_data\_mini.csv')

# Function to reduce hours-per-week

def reduce\_hours(x):

return x - n

# Apply reduction

data['reduced\_hours'] = data['hours-per-week'].apply(reduce\_hours)

# First grouping by 'relationship' and 'hours-per-week'

group1 = data.groupby(['relationship', 'hours-per-week']).size()

# Second grouping by 'relationship' and reduced 'hours-per-week'

group2 = data.groupby(['relationship', 'reduced\_hours']).size()

# Print results

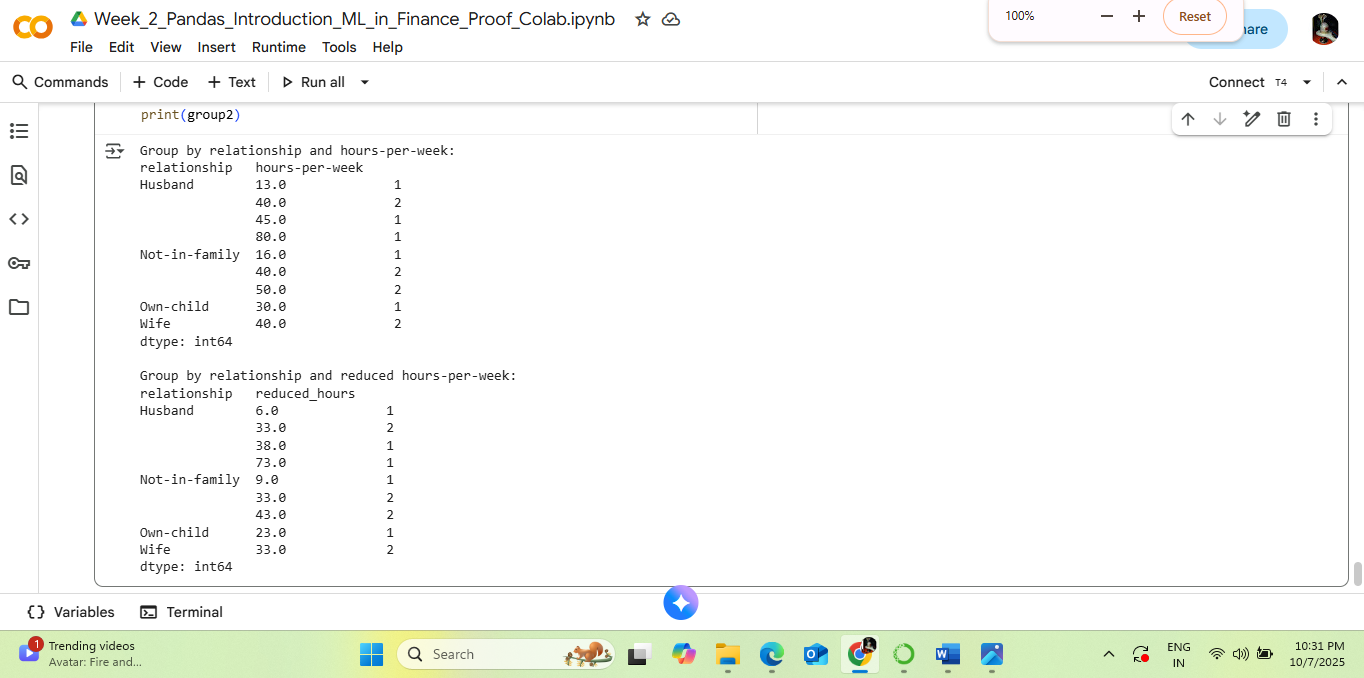
print("Group by relationship and hours-per-week:")

print(group1)

print("\nGroup by relationship and reduced hours-per-week:")

print(group2)

**Output:**

****

Lab 3:

import seaborn as sns

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

# Load dataset

iris = load\_iris()

df = pd.DataFrame(iris.data, columns=iris.feature\_names)

# Select columns 7 and 8 → index 6 and 7 (zero-based indexing)

# Since Iris has only 4 columns, we'll simulate extra columns for this example

# Extending DataFrame with dummy features for demonstration (if needed)

for i in range(4, 10):

df[f'feature\_{i+1}'] = df[iris.feature\_names[i % 4]] \* (i + 1)

# Define the two columns based on SID logic

col1 = df.columns[6] # 7th column

col2 = df.columns[7] # 8th column

# Bicolour interaction plot using seaborn

plt.figure(figsize=(8, 6))

sns.scatterplot(data=df, x=col1, y=col2, hue=iris.target, palette='coolwarm')

plt.title(f'Bicolour Feature Interaction: {col1} vs {col2}')

plt.xlabel(col1)

plt.ylabel(col2)

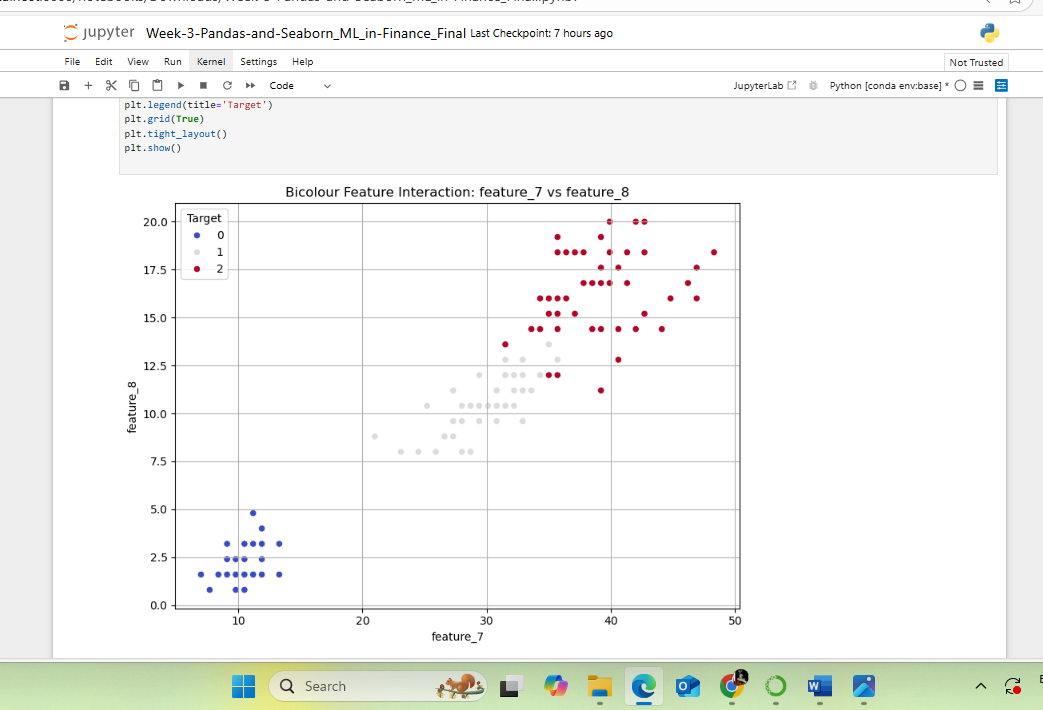
plt.legend(title='Target')

plt.grid(True)

plt.tight\_layout()

plt.show()

**Output:**

****

Lab 4:

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

# Make sure your datasets are already loaded and preprocessed:

# x\_train, y\_train, x\_test, y\_test

# Define the MLP model with two hidden layers

model = Sequential([

Dense(77, input\_dim=x\_train.shape[1], activation='relu', kernel\_initializer='normal'),

Dense(38, activation='relu', kernel\_initializer='normal'),

Dense(1)

])

# Compile the model (same parameters as practical session)

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

# Print the architecture

print("MLP Architecture:")

model.summary()

# Train the model

history = model.fit(

x\_train, y\_train,

batch\_size=10,

epochs=10,

validation\_split=0.2,

verbose=1

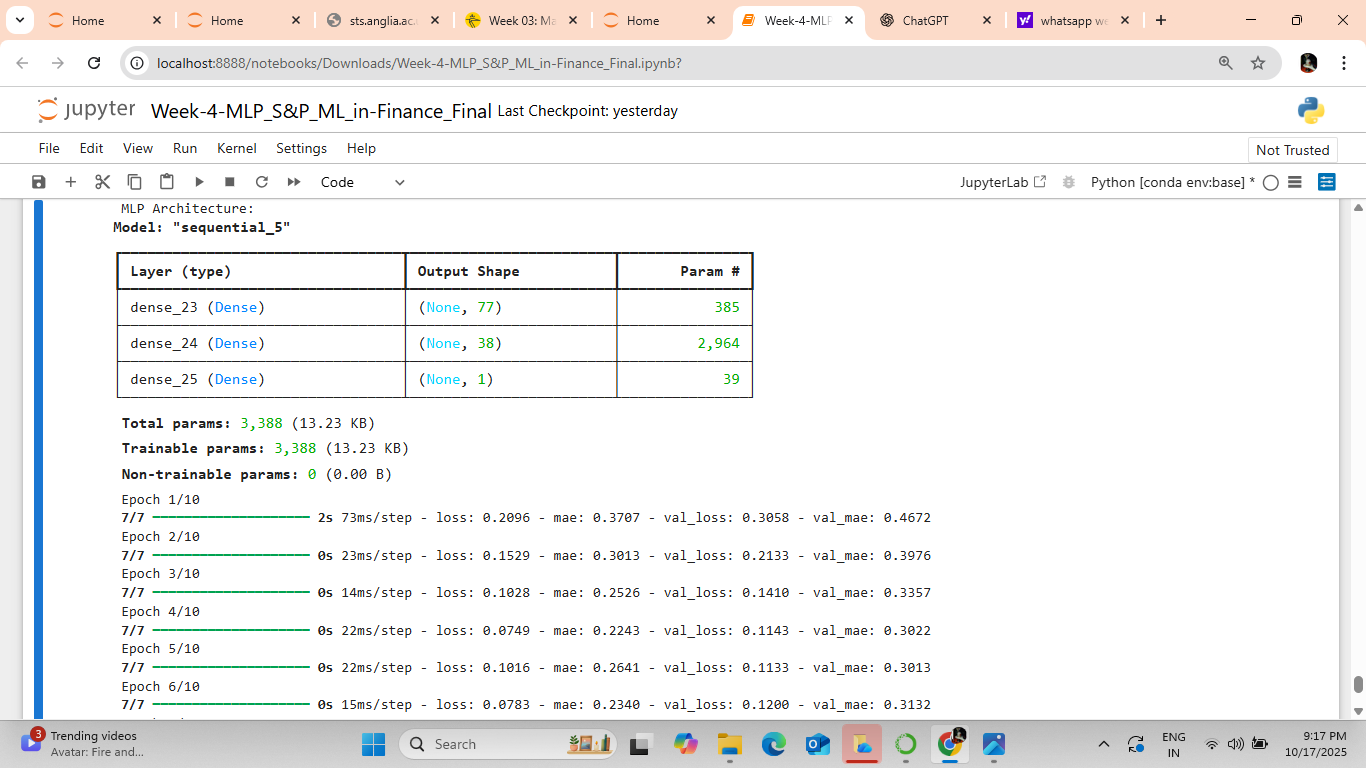
)

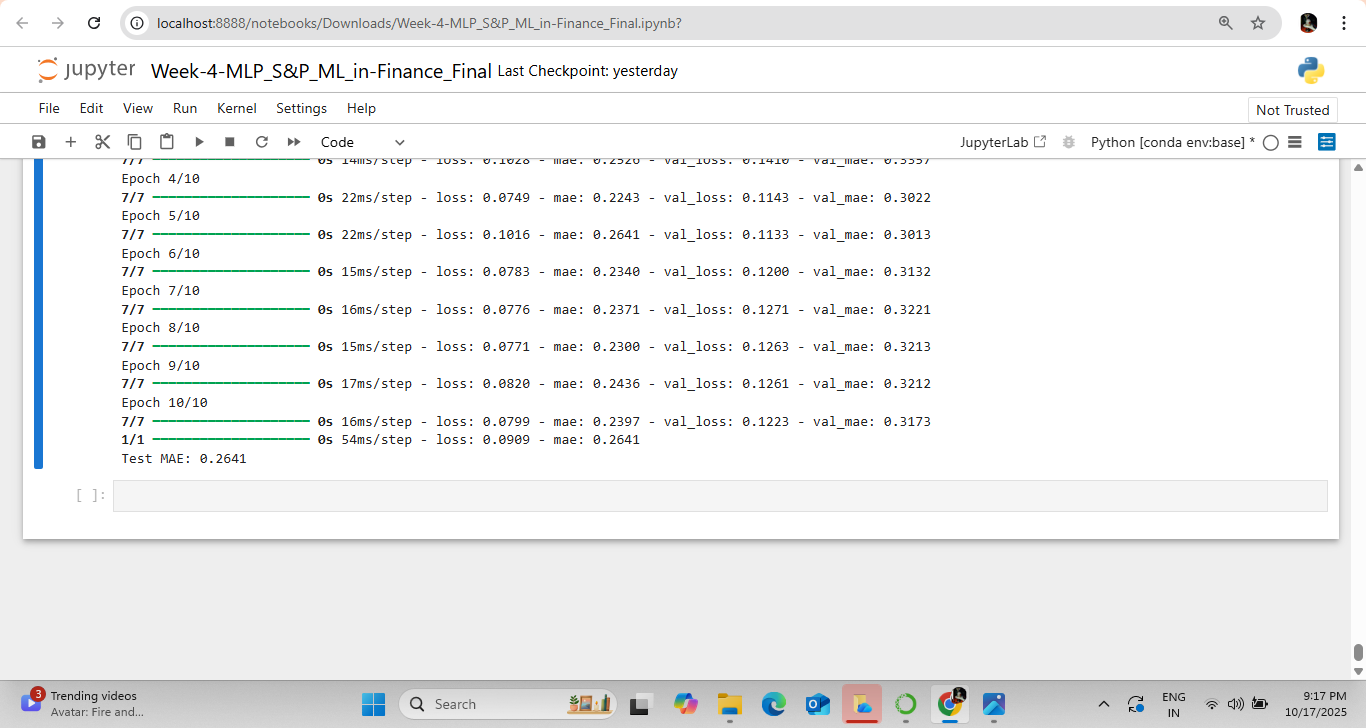
# Evaluate on test set

test\_loss, test\_mae = model.evaluate(x\_test, y\_test)

print(f"Test MAE: {test\_mae:.4f}")

**Output:**





Lab 5:

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

import matplotlib.pyplot as plt

# Define the CNN model with modified parameters

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) # Output: Bid and Ask

])

# Compile the model

model.compile(optimizer='adam',loss='mean\_absolute\_error', metrics=['mae'])

# Display model summary

print(model.summary())

# Train the model

history = model.fit(X\_train, y\_train,

batch\_size=50, # Changed

epochs=14, # Based on your SID 2420077

validation\_split=0.2,

verbose=1)

# Evaluate on test data

test\_loss, test\_mae = model.evaluate(X\_test, y\_test, verbose=1)

print(f"\nTest MAE: {test\_mae:.6f}")

# Predict using trained model

pred = model.predict(X\_test)

# Plot the real vs predicted values

plt.ion()

fig = plt.figure(figsize=(23, 8))

ax1 = fig.add\_subplot(111)

line1, = ax1.plot(y\_test[:400, 0], label='Bid')

line2, = ax1.plot(pred[:400, 0], label='Predicted Bid')

line3, = ax1.plot(y\_test[:400, 1], label='Ask')

line4, = ax1.plot(pred[:400, 1], label='Predicted Ask')

plt.title('Real & Predicted Normalised Prices', size=18)

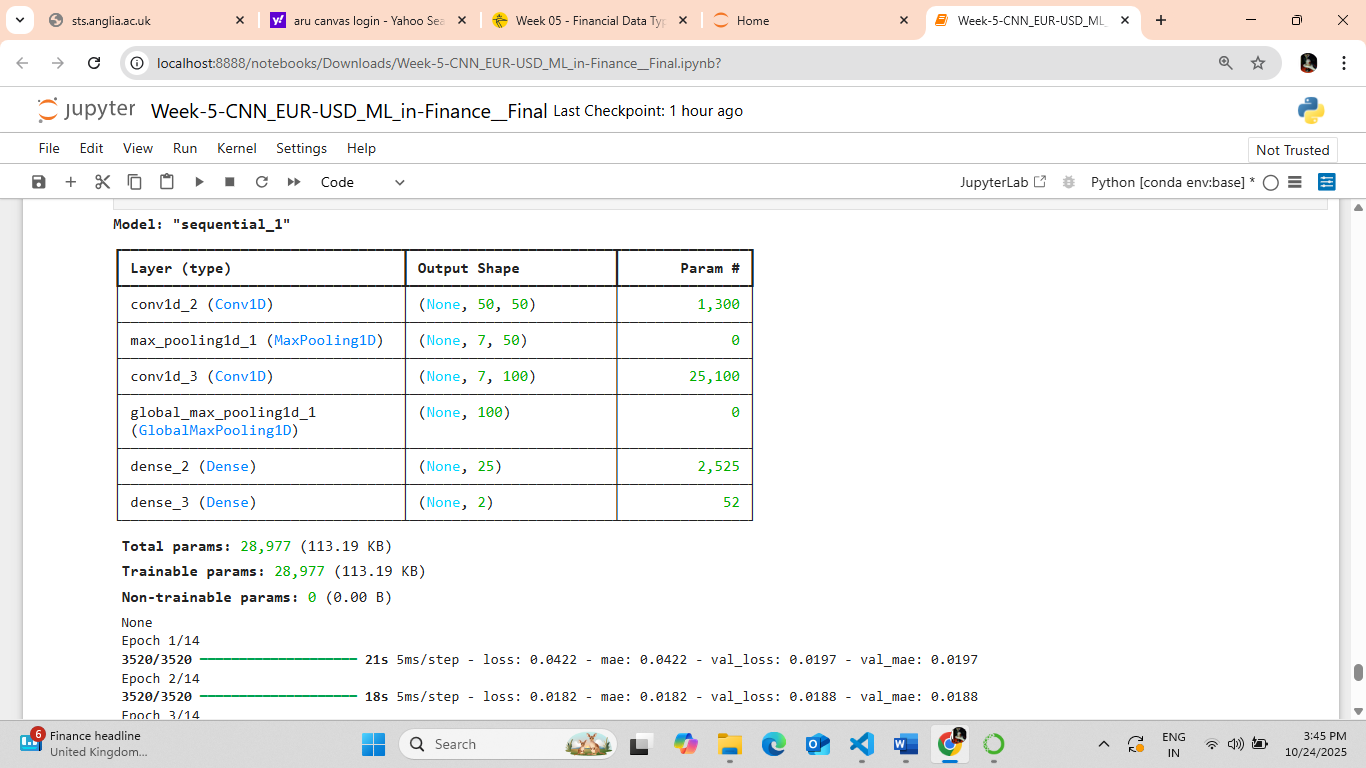
plt.ylabel('Price', size=14)

plt.xlabel('Ticks', size=14)

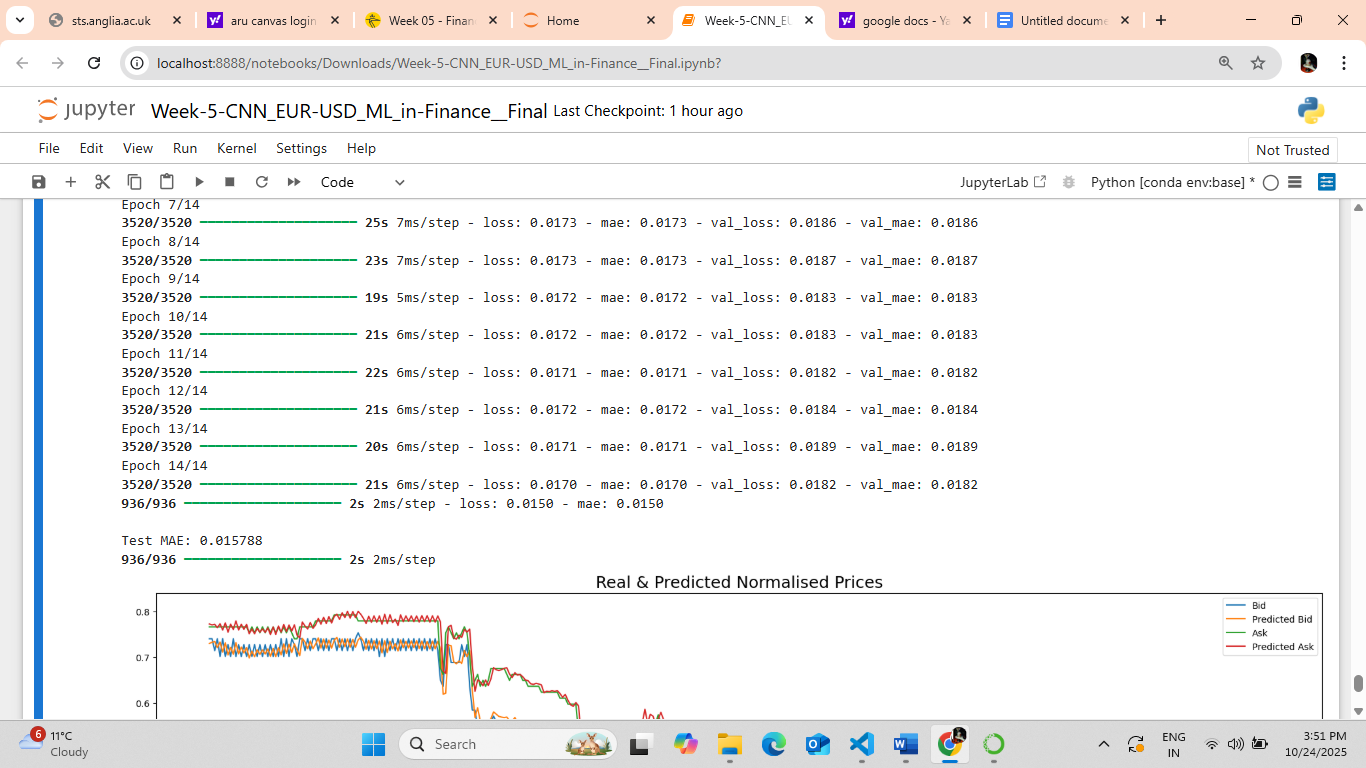
plt.legend(loc='upper right')

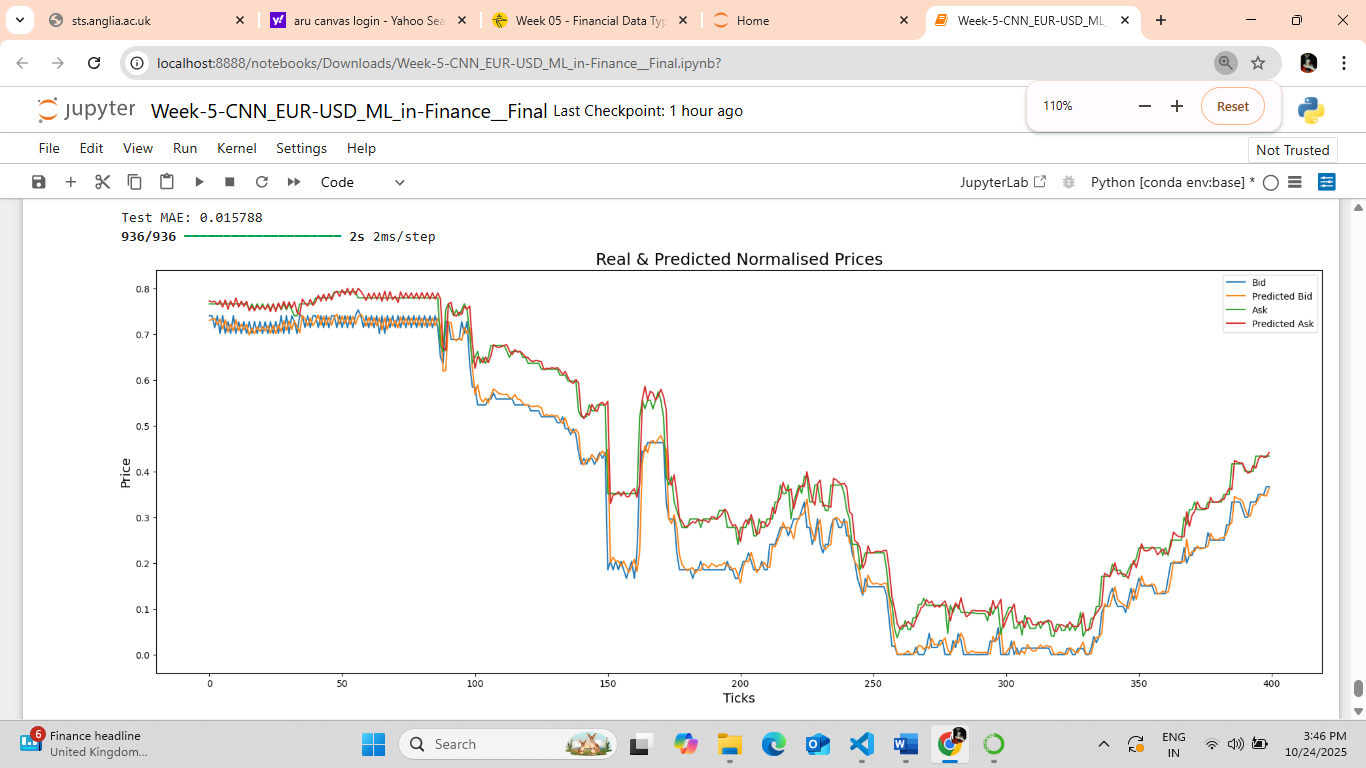
plt.show()

**Output:**

****

****

****

****

Lab 6:

# Student ID: 2420077

# Start index = 20077 (last 5 digits of SID)

# Time period = 77 minutes (last 3 digits of SID)

import pandas as pd

import cufflinks as cf

cf.go\_offline()

# Load the dataset

data2 = pd.read\_csv('GOLD\_2022\_normalised.csv')

# Convert all numeric columns properly

for col in data2.columns:

try:

data2[col] = pd.to\_numeric(data2[col])

except:

pass

# Select only numeric columns

numeric\_cols = data2.select\_dtypes(include='number').columns

# Normalise numeric 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)

# Select dataset part based on Student ID

start\_index = 20077

period = 77

data\_part = data2.iloc[start\_index : start\_index + period]

# Rename columns for clarity (if necessary)

data\_part.rename(columns={'High\_x': 'High\_Bid', 'Low\_x': 'Low\_Bid'}, inplace=True)

# Plot the chart

data\_part[['High\_Bid', 'Low\_Bid']].iplot(

title='GOLD Price Chart (High\_Bid vs Low\_Bid) - Student ID: 2420077',

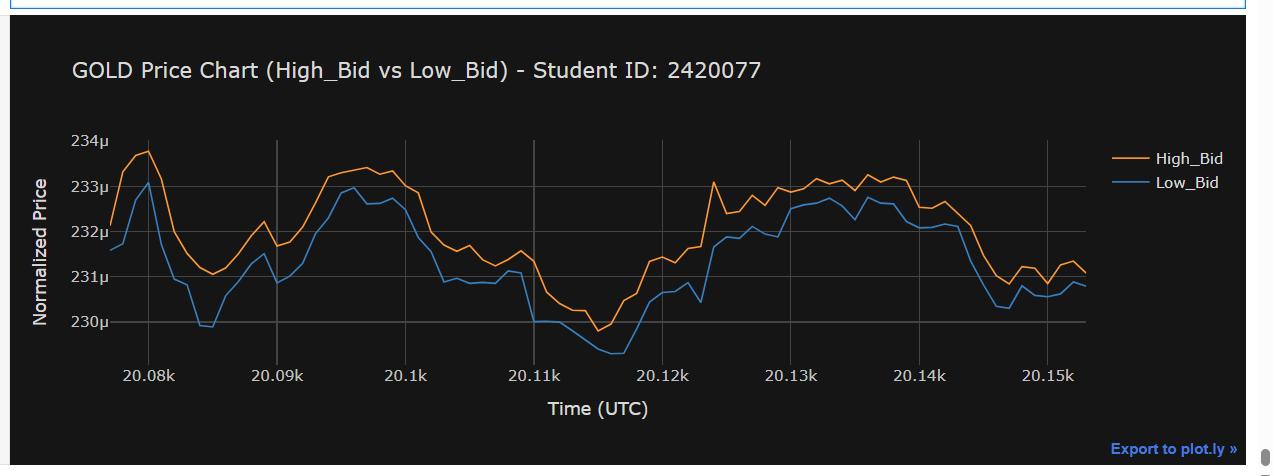
xTitle='Time (UTC)',

yTitle='Normalized Price',

theme='solar'

)

**Output:**

****

Lab 7:

import numpy as np

import pandas as pd

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense

from tensorflow.keras.callbacks import EarlyStopping

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error

import matplotlib.pyplot as plt

# ------------------------------

# Load and preprocess your dataset

# ------------------------------

data = pd.read\_csv('GOLD\_2022\_normalised\_NEW-1000.csv')

# Select the target column (change to 'Close\_Ask' if needed)

values = data['Close\_Bid'].values.reshape(-1, 1)

scaler = MinMaxScaler()

scaled\_values = scaler.fit\_transform(values)

# Create your train-test split

train\_size = int(len(scaled\_values) \* 0.8)

train, test = scaled\_values[:train\_size], scaled\_values[train\_size:]

# Function to create sequences

def create\_dataset(dataset, time\_step=1):

X, Y = [], []

for i in range(len(dataset) - time\_step - 1):

X.append(dataset[i:(i + time\_step), 0])

Y.append(dataset[i + time\_step, 0])

return np.array(X), np.array(Y)

time\_step = 10

X\_train, y\_train = create\_dataset(train, time\_step)

X\_test, y\_test = create\_dataset(test, time\_step)

# Reshape input to [samples, time steps, features]

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)

# ------------------------------

# Build the LSTM model

# ------------------------------

model = Sequential()

model.add(LSTM(87, input\_shape=(time\_step, 1))) # SID 2420077 → 77+10=87

model.add(Dense(1))

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

# Show model summary for screenshot

model.summary()

# ------------------------------

# Train the model

# ------------------------------

early\_stop = EarlyStopping(monitor='val\_loss', patience=3, restore\_best\_weights=True)

history = model.fit(

X\_train, y\_train,

validation\_data=(X\_test, y\_test),

epochs=10, # Changed to 10

batch\_size=32,

verbose=1,

callbacks=[early\_stop]

)

# ------------------------------

# Evaluate the model

# ------------------------------

y\_pred = model.predict(X\_test)

y\_pred\_rescaled = scaler.inverse\_transform(y\_pred)

y\_test\_rescaled = scaler.inverse\_transform(y\_test.reshape(-1, 1))

test\_mse = mean\_squared\_error(y\_test\_rescaled, y\_pred\_rescaled)

test\_mae = mean\_absolute\_error(y\_test\_rescaled, y\_pred\_rescaled)

print(f"Test MSE: {test\_mse:.4f}")

print(f"Test MAE: {test\_mae:.4f}")

# ------------------------------

# Plot MAE graph

# ------------------------------

plt.figure(figsize=(8,5))

plt.plot(history.history['mae'], label='Train MAE')

plt.plot(history.history['val\_mae'], label='Validation MAE')

plt.title('LSTM Model MAE over Epochs')

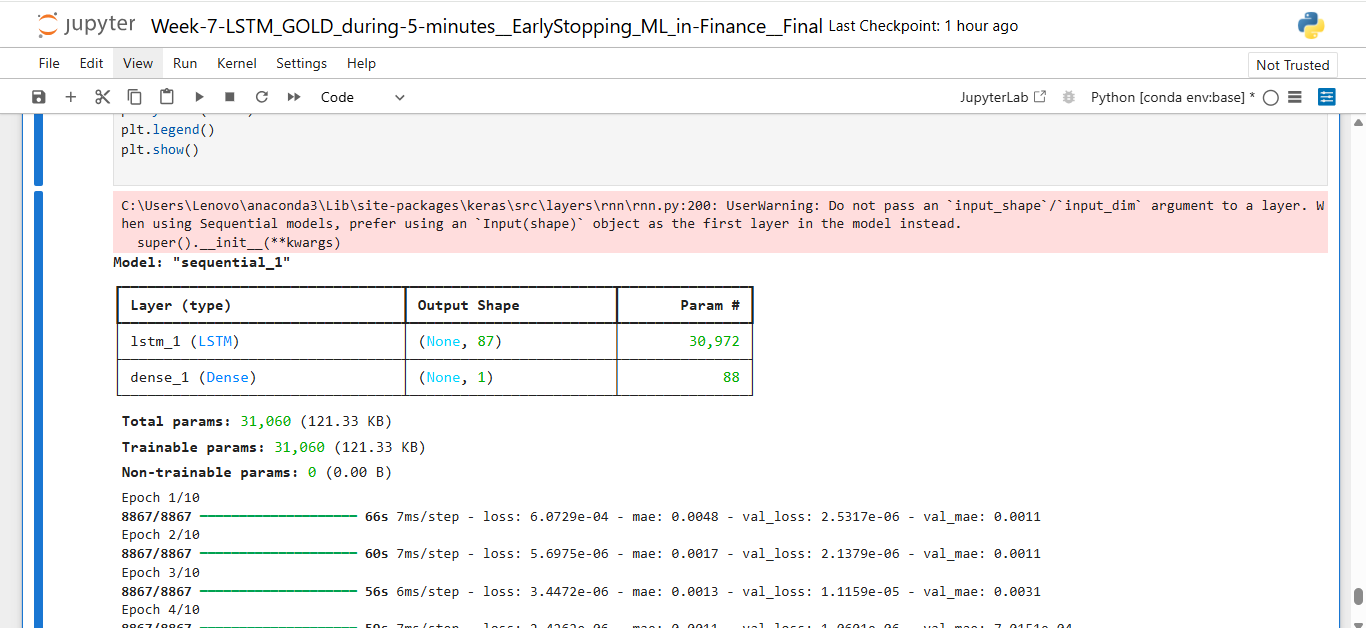
plt.xlabel('Epochs')

plt.ylabel('MAE')

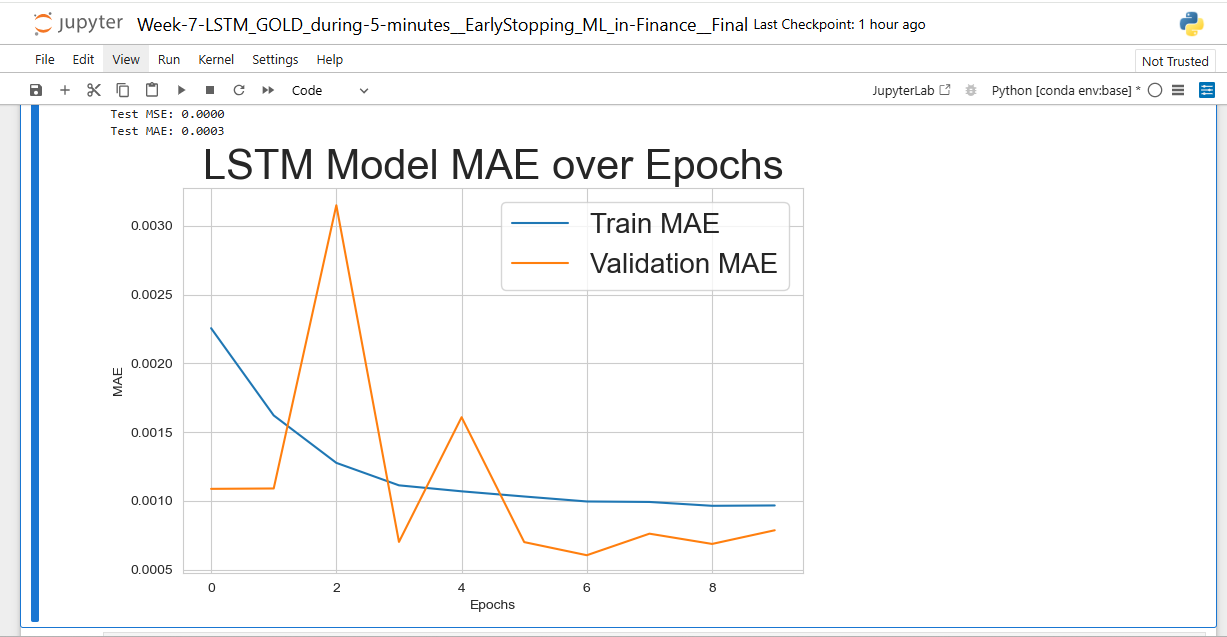
plt.legend()

plt.show()

**Output:**

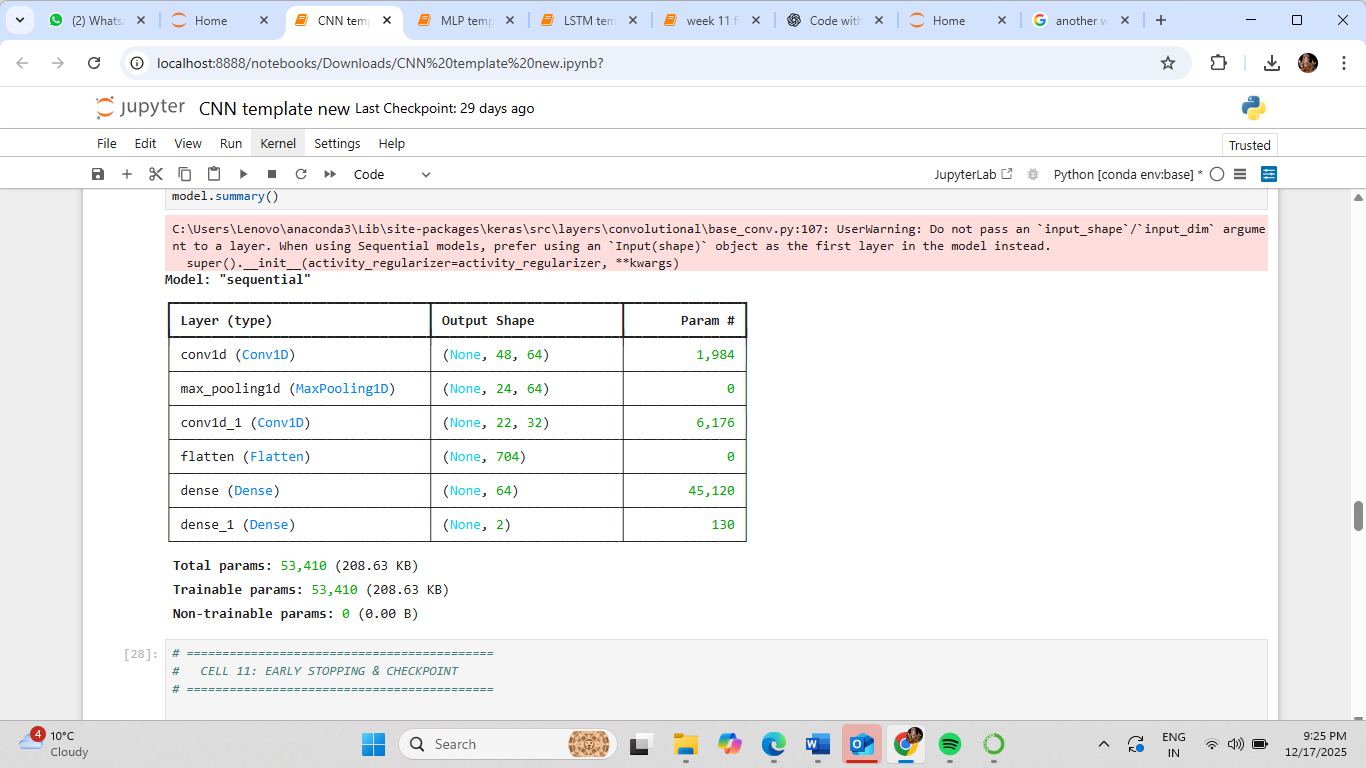
****

****

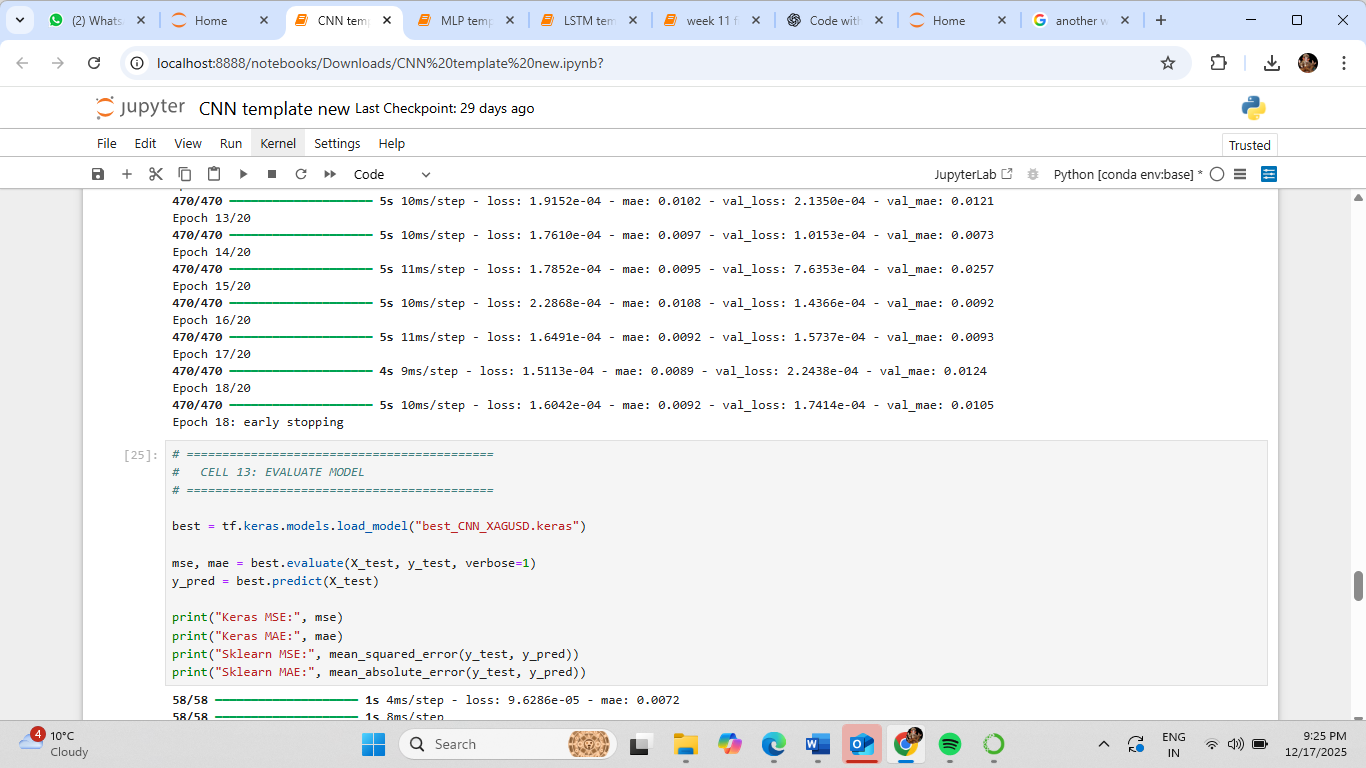
****

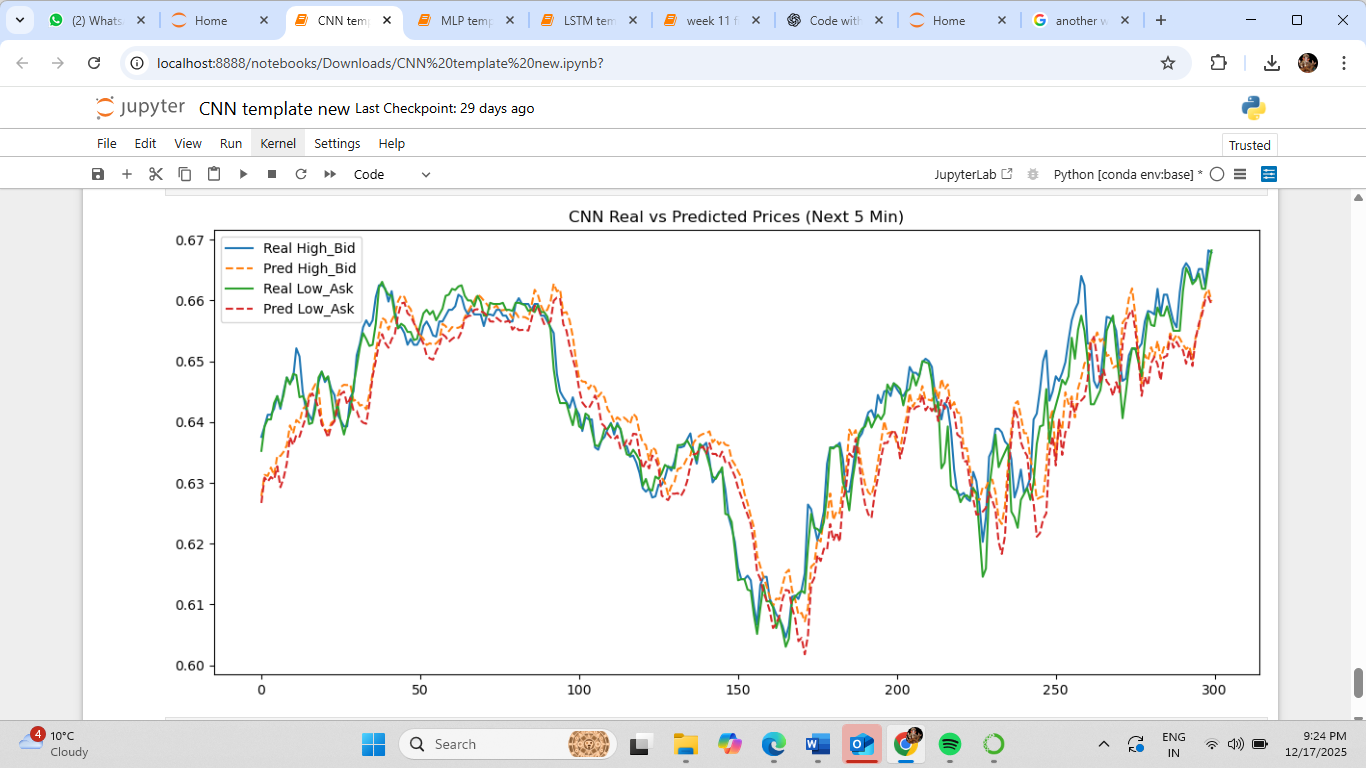
Lab 8:

CNN:

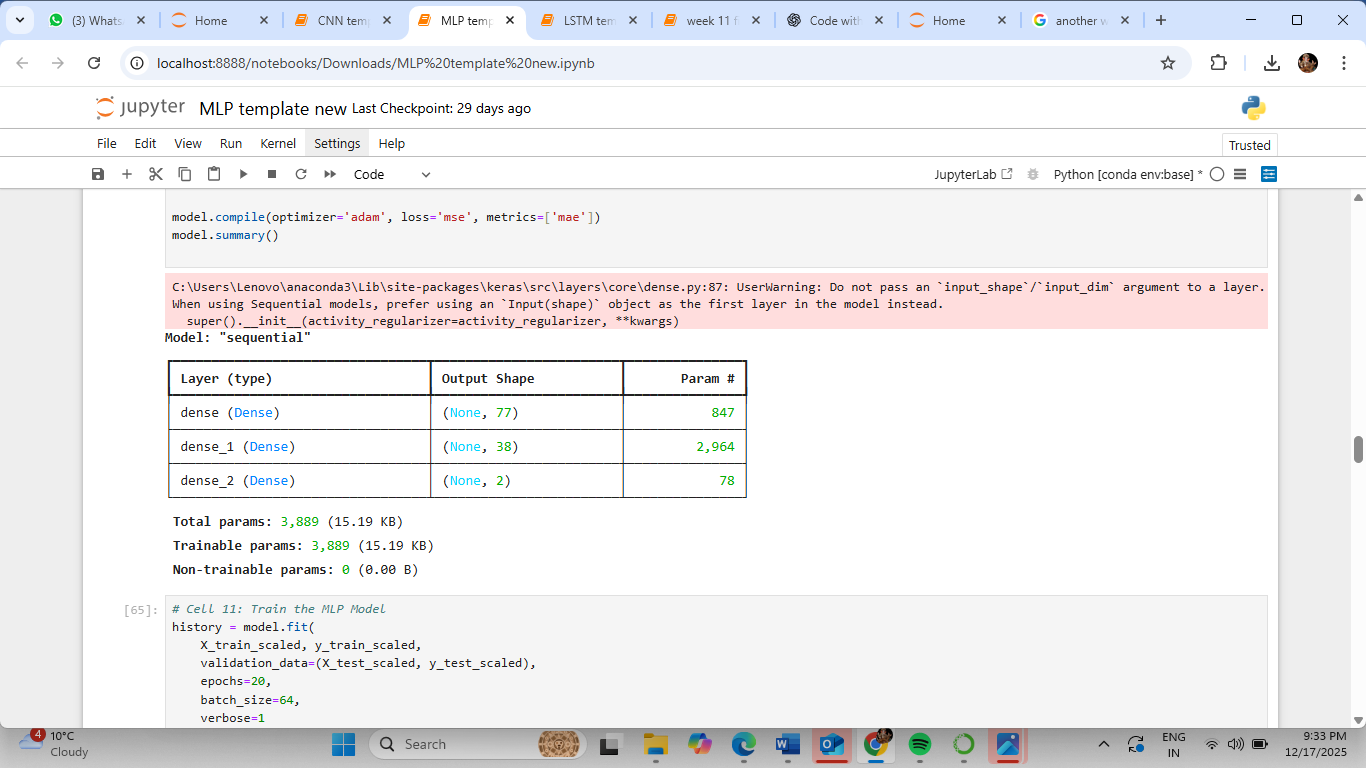


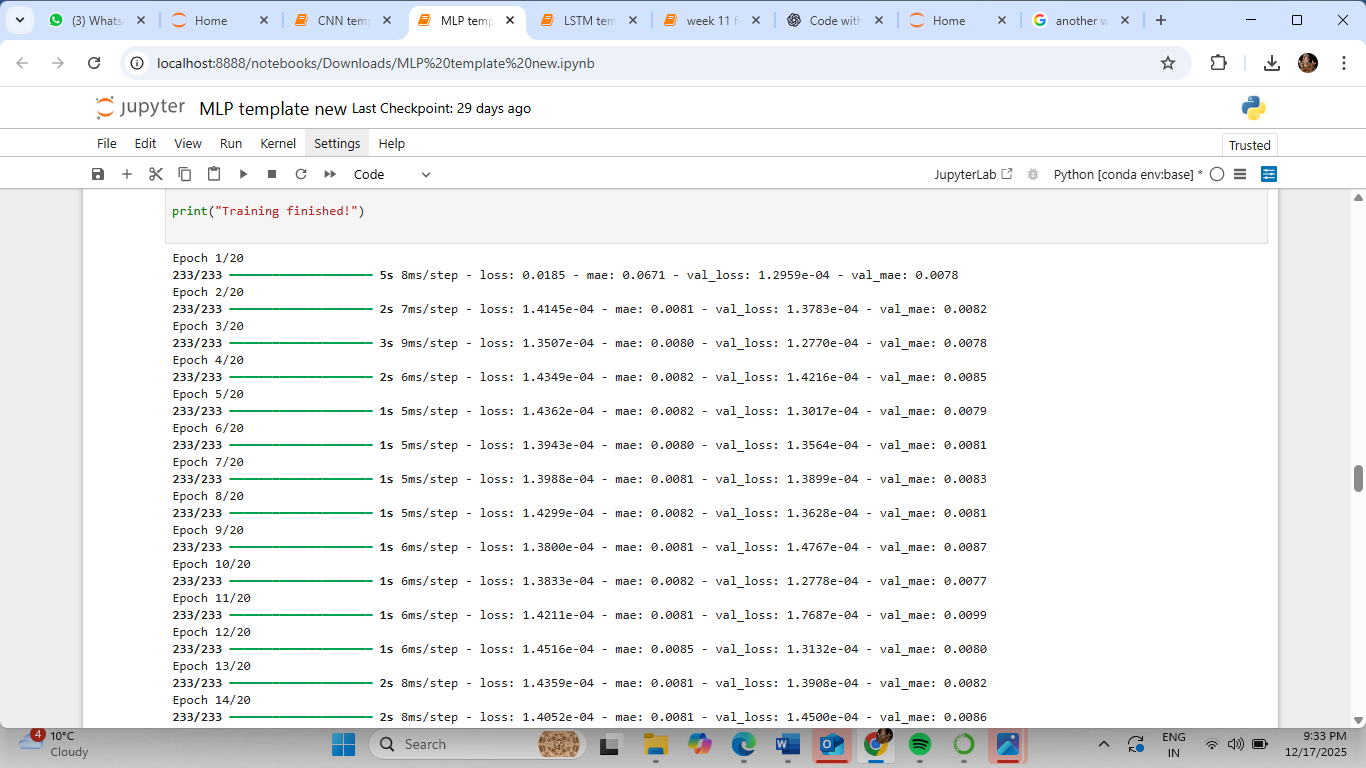


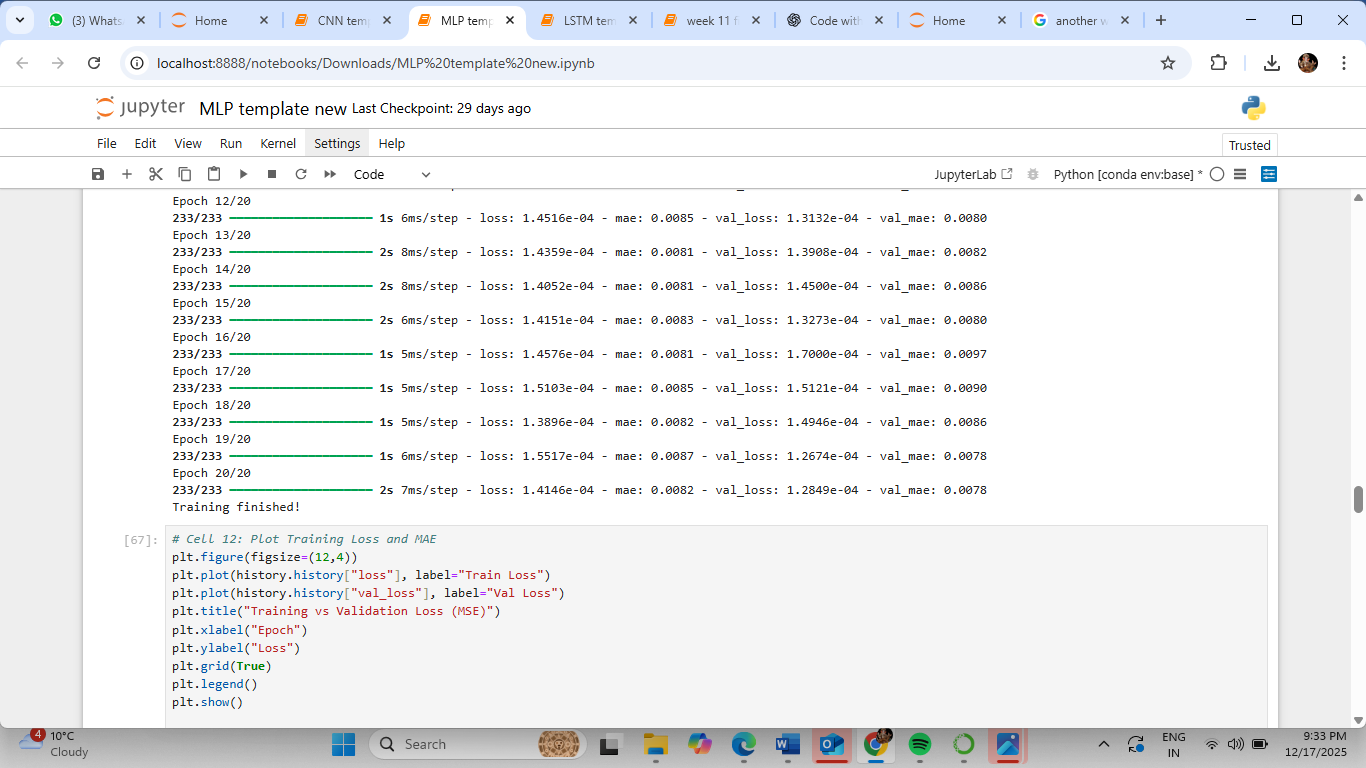


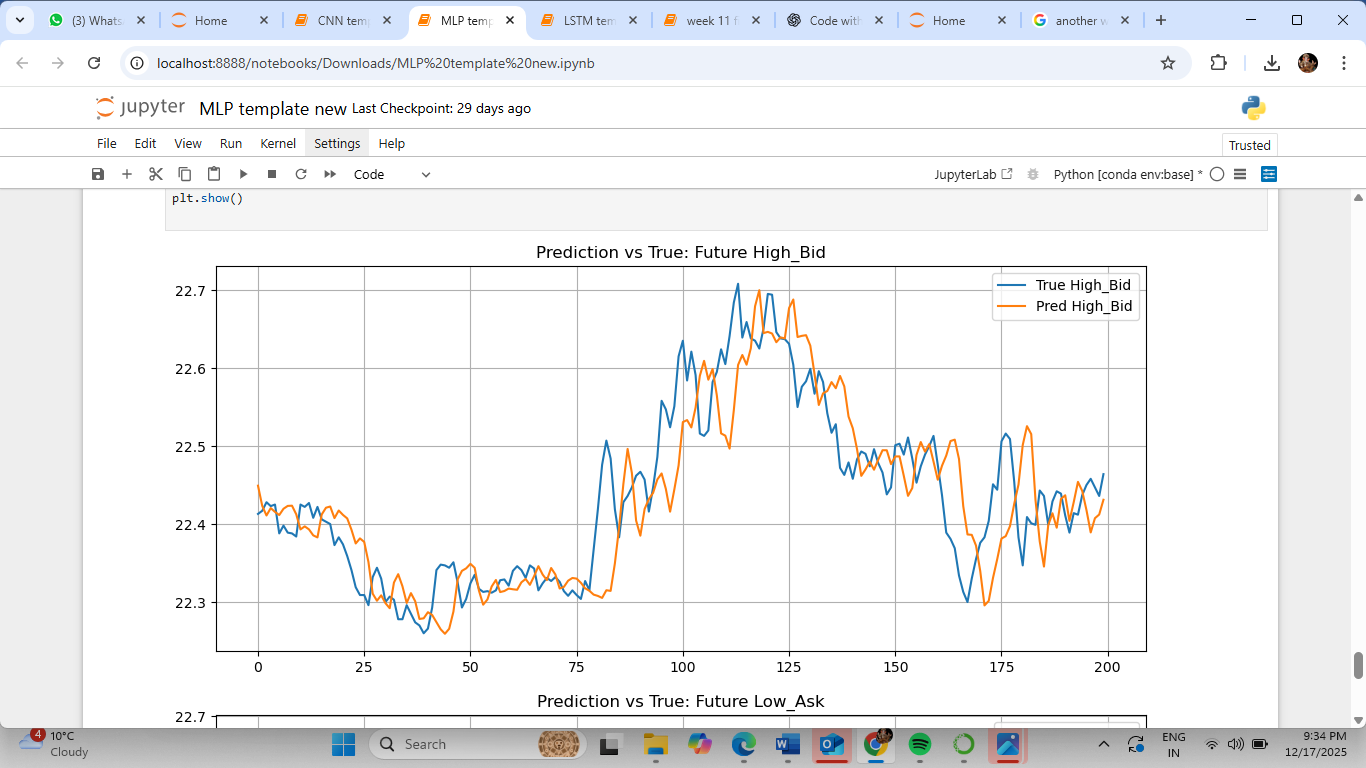


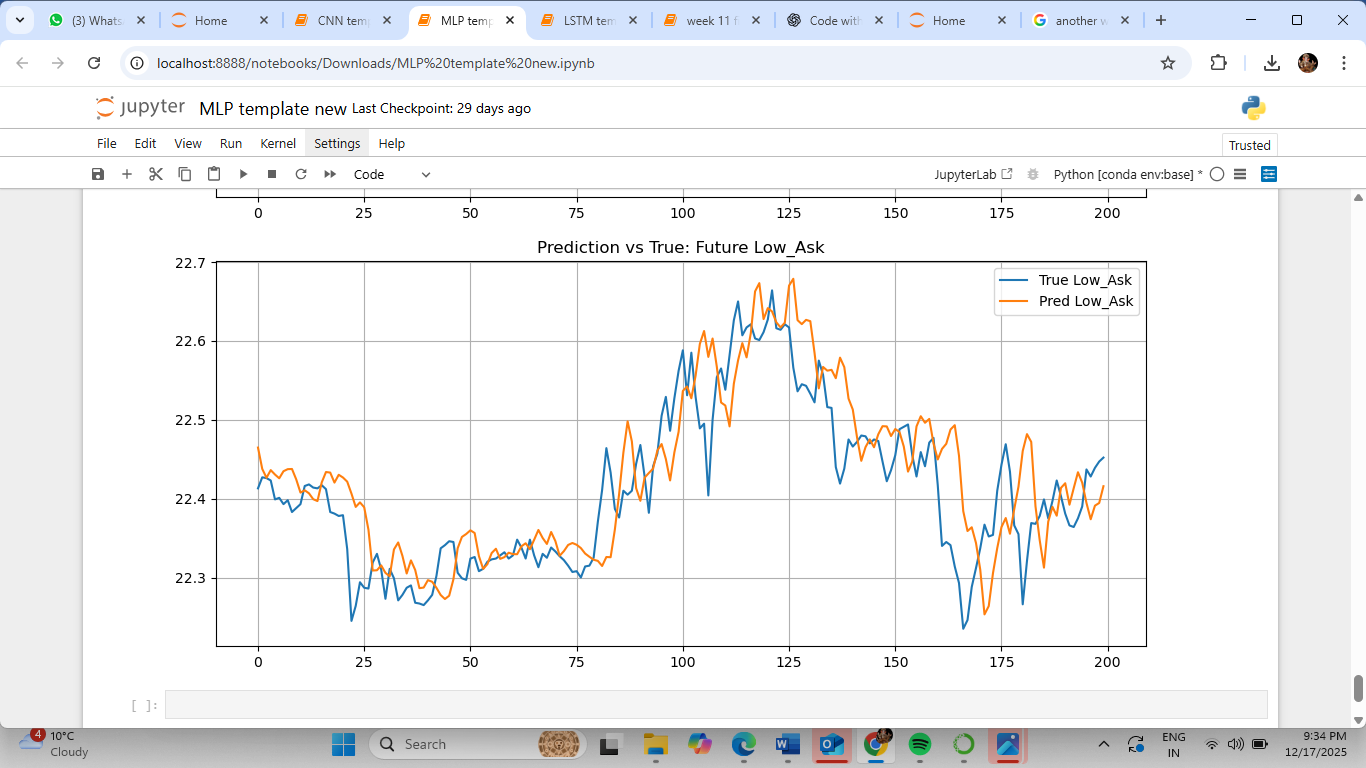
MLP:



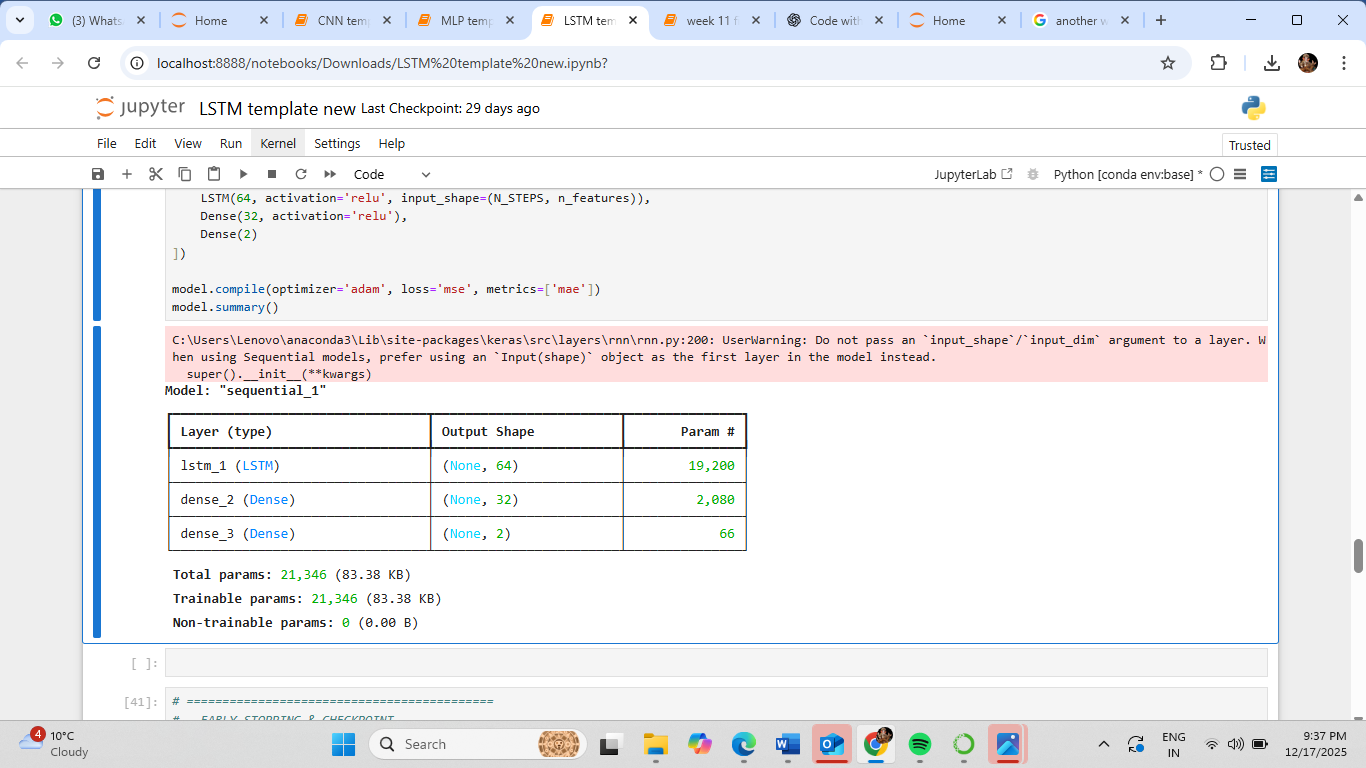


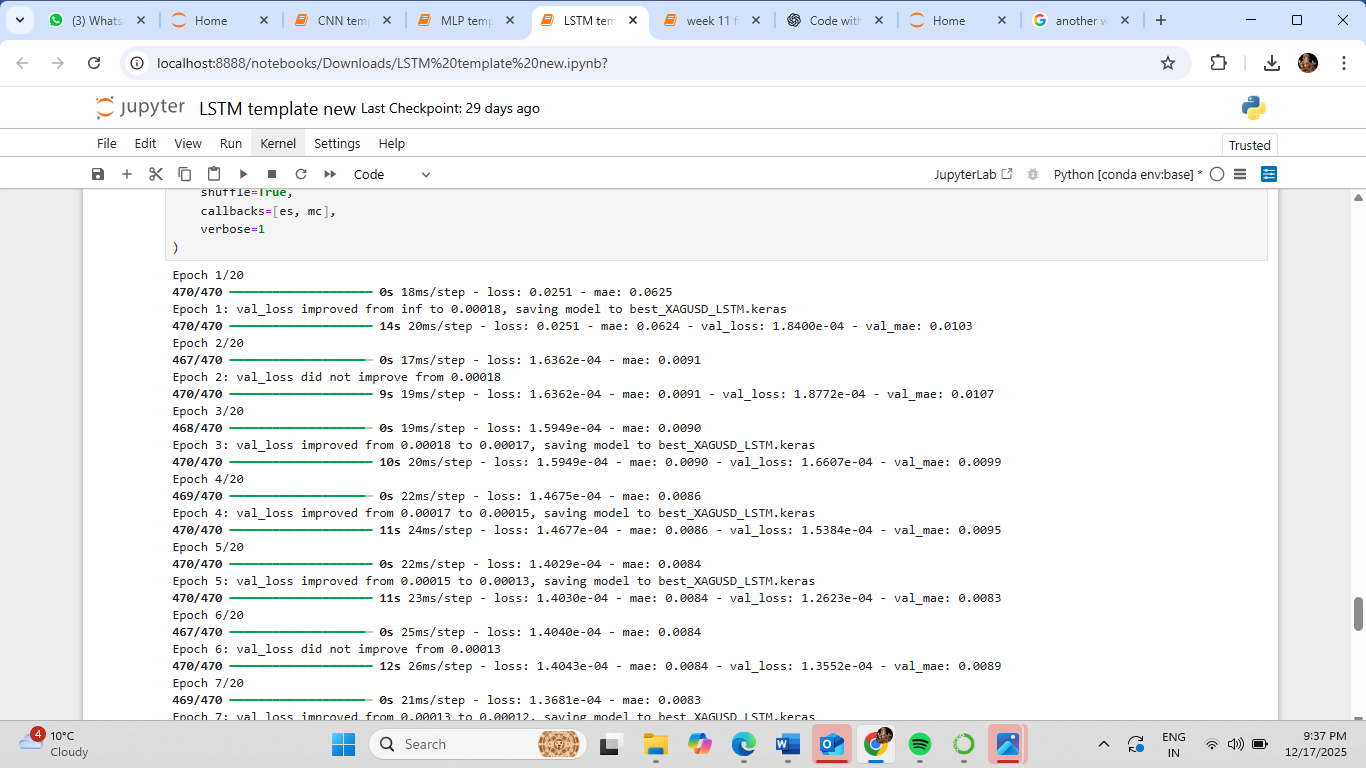


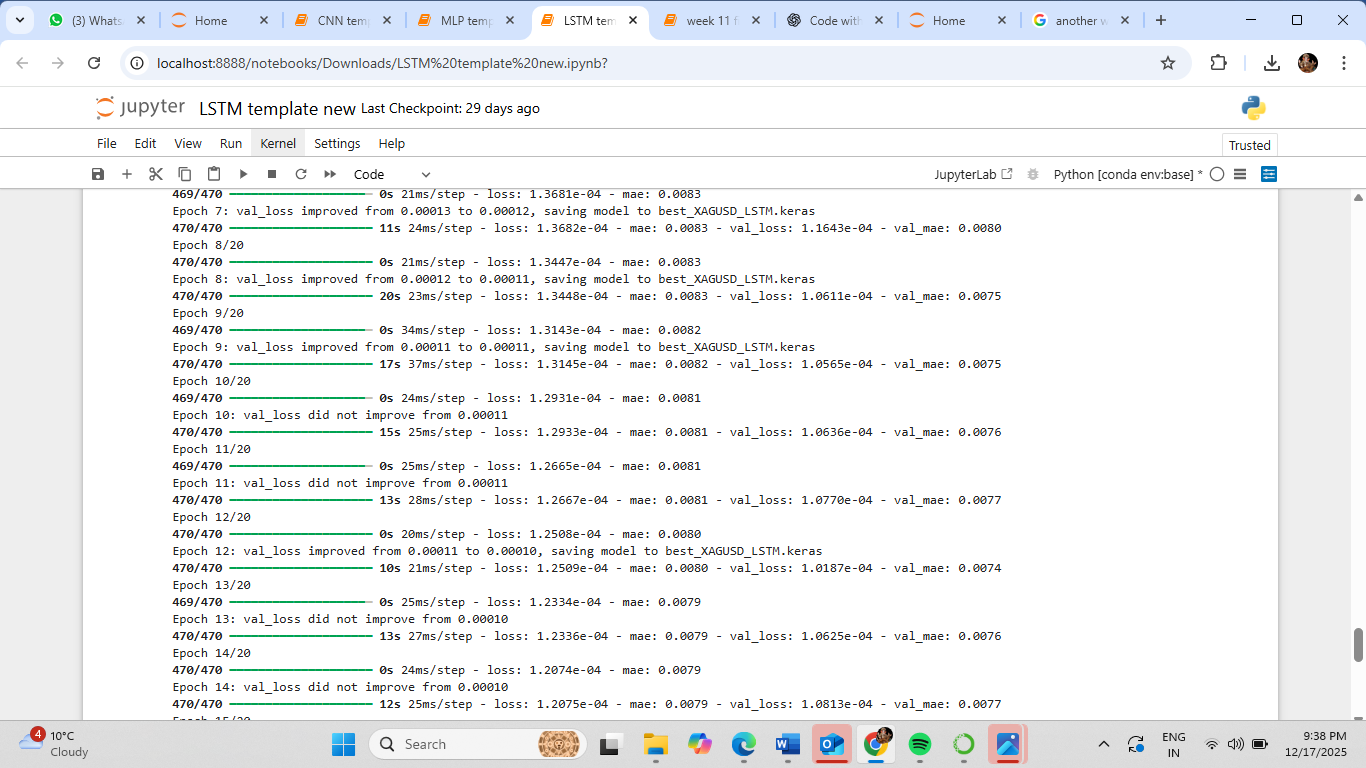


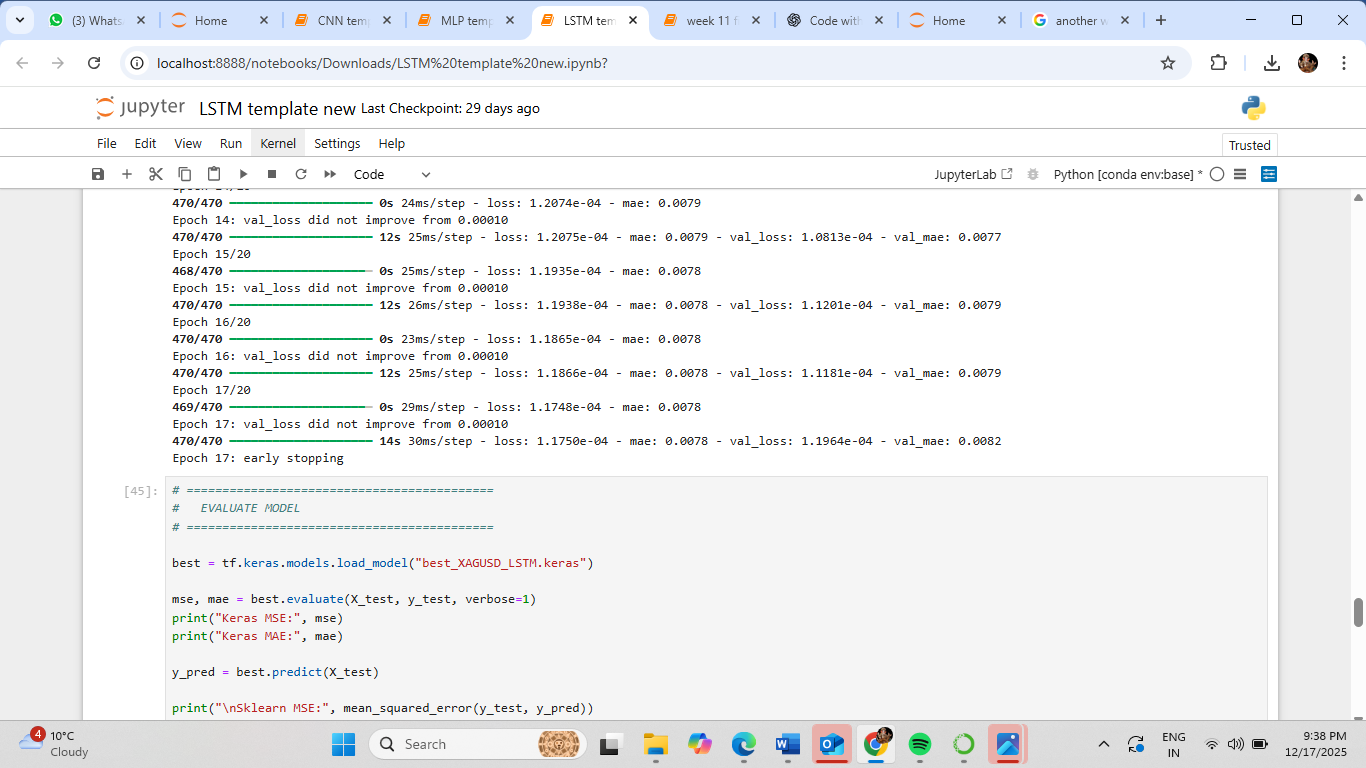


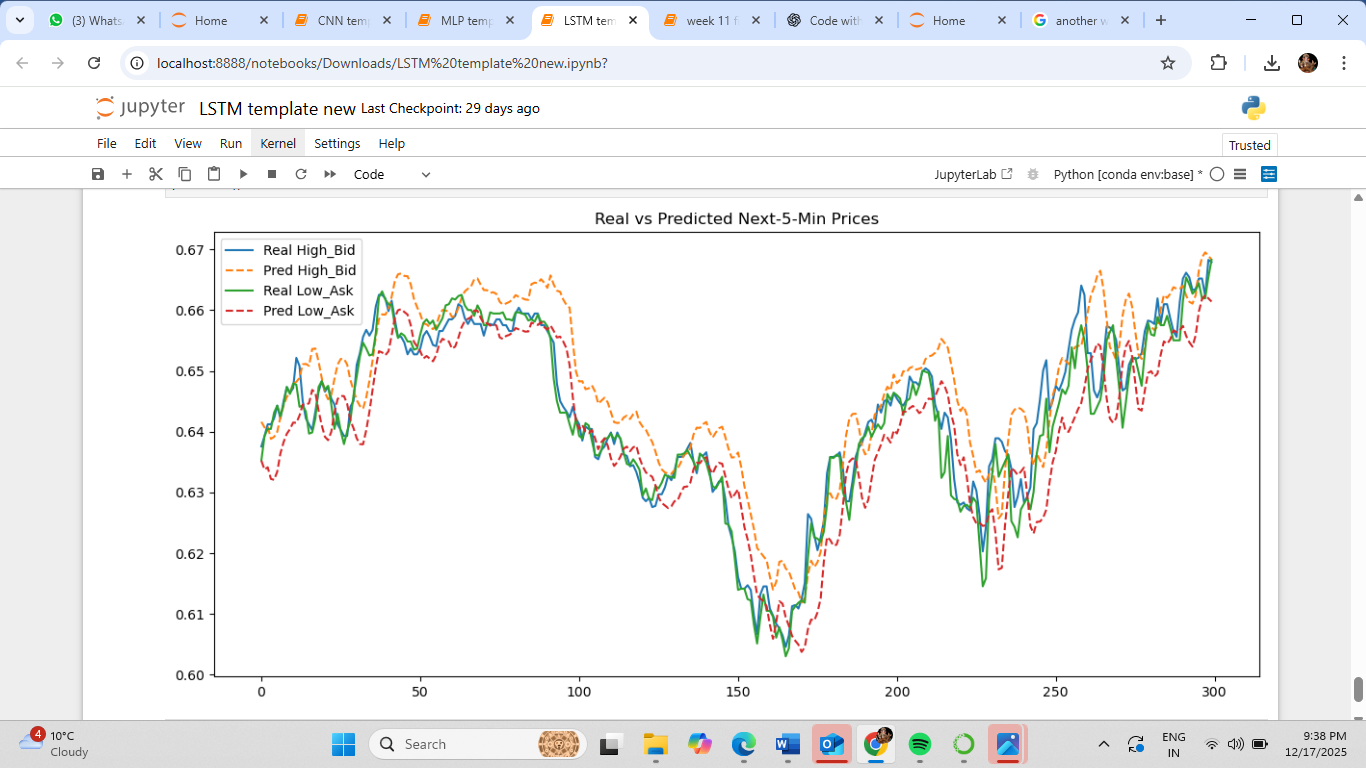
LSTM:











Lab 9 :

In class test

Lab 10:

# SID: 2420077

# batch\_size = 17

# epochs = 37

from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

es = EarlyStopping(monitor='val\_loss', mode='min', verbose=1, patience=10)

mc = ModelCheckpoint(

'best\_Trand-Flat\_model\_LSTM\_GOLD\_2420077.keras',

monitor='val\_loss',

mode='min',

verbose=1,

save\_best\_only=True

)

history = model.fit(

X\_train,

y\_train,

batch\_size=17,

epochs=37,

validation\_split=0.2,

shuffle=True,

verbose=1,

callbacks=[es, mc],

class\_weight=class\_weight\_dict

)

pred = LSTM\_saved\_best\_model.predict(X\_test)

pred\_binary = (pred >= 0.6).astype(int)

from sklearn.metrics import classification\_report

error\_metrics\_report = classification\_report(y\_test, pred\_binary)

print("Test Accuracy and Error Metrics:")

print(error\_metrics\_report)

import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))

plt.plot(history.history['accuracy'], label='Training Accuracy', marker='o', linewidth=2)

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy', marker='s', linewidth=2)

plt.title(

'More Detailed Accuracy - Training vs Validation (Modified Parameters)',

fontsize=14,

fontweight='bold'

)

plt.xlabel('Epoch', fontsize=12)

plt.ylabel('Accuracy', fontsize=12)

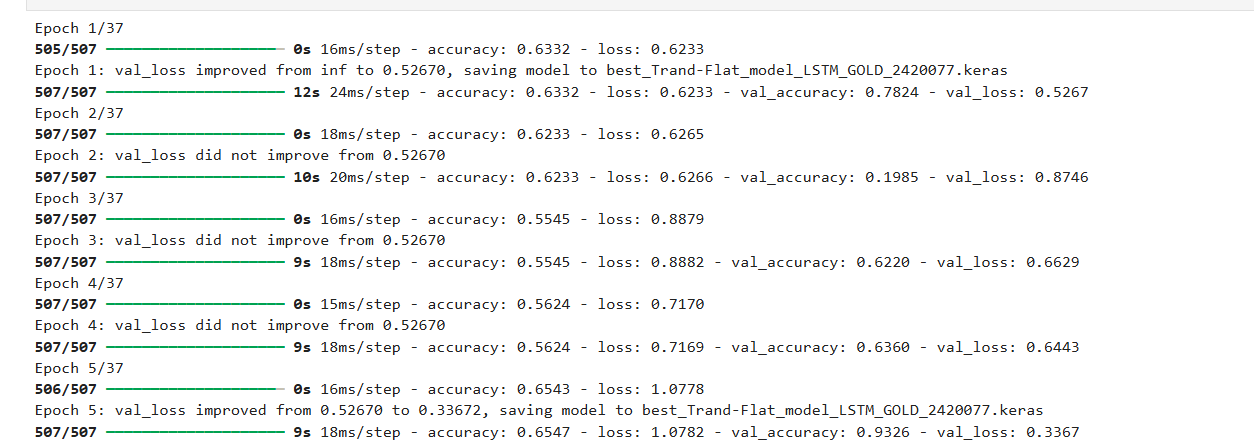
plt.legend(loc='best', fontsize=11)

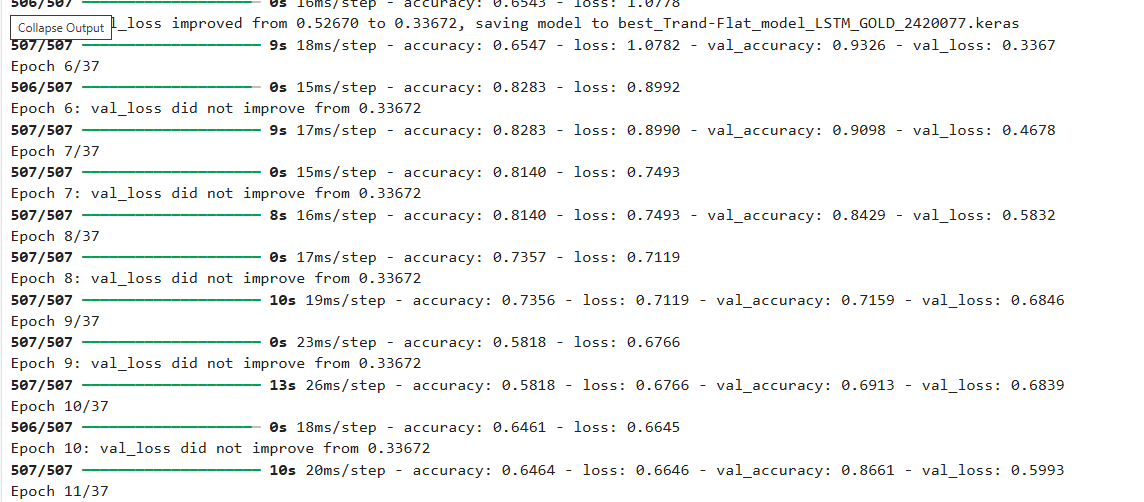
plt.grid(True, alpha=0.3)

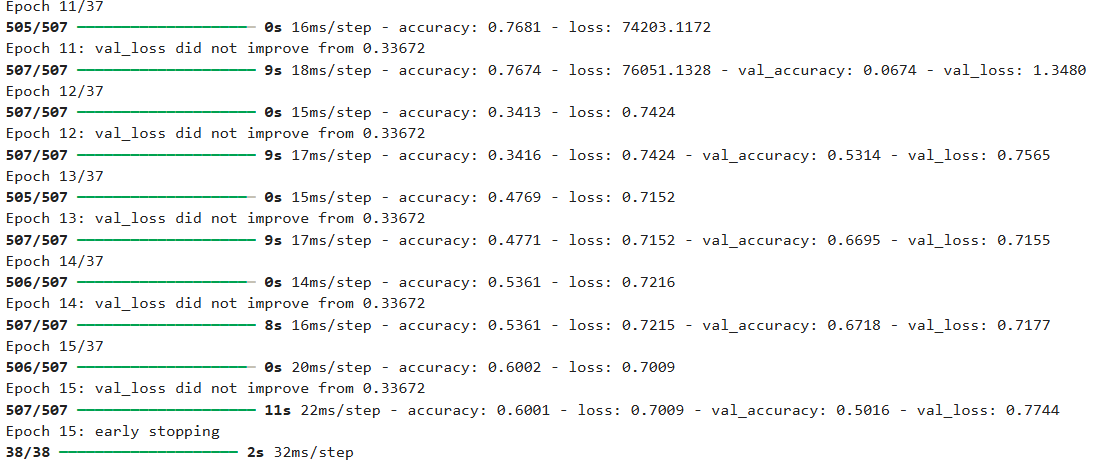
plt.tight\_layout()

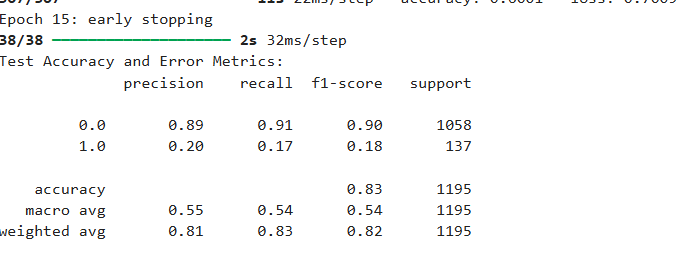
plt.show()

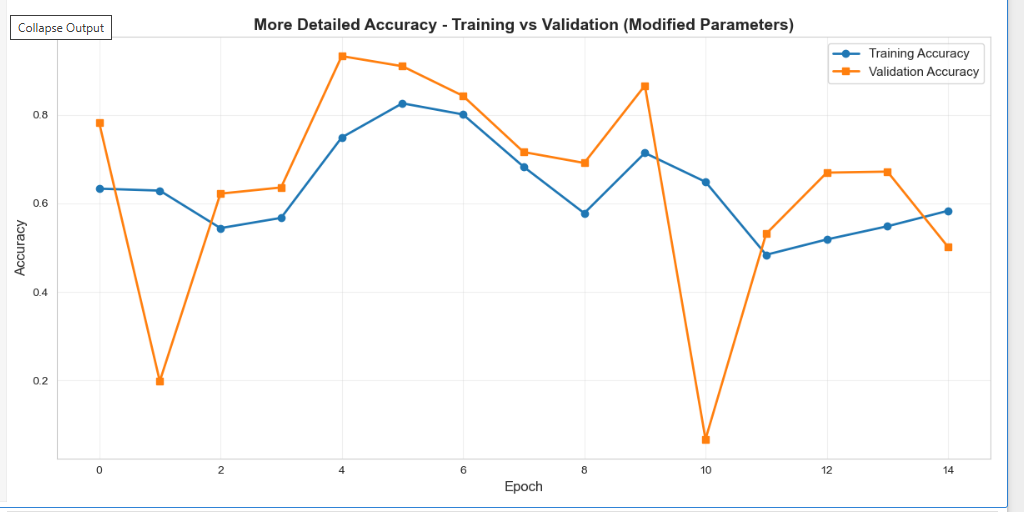
**output:**











Lab 11:

# ============================================

# Cell 7

# Create and train YOUR OWN LSTM model

# SID: 2420077 → Z = 7

# epochs = 37

# batch\_size = 17

# ============================================

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense, Dropout, BatchNormalization

from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

EPOCHS = 37

BATCH\_SIZE = 17

lstm\_model = Sequential([

LSTM(128, input\_shape=(1, n\_features), return\_sequences=True),

BatchNormalization(),

Dropout(0.25),

LSTM(64, return\_sequences=False),

BatchNormalization(),

Dropout(0.25),

Dense(32, activation='relu'),

Dropout(0.15),

Dense(1, activation='sigmoid')

])

lstm\_model.compile(

optimizer=tf.keras.optimizers.Adam(learning\_rate=0.001),

loss='binary\_crossentropy',

metrics=['accuracy']

)

es = EarlyStopping(

monitor='val\_loss',

mode='min',

patience=10,

restore\_best\_weights=True,

verbose=1

)

mc = ModelCheckpoint(

'best\_LSTM\_2420077.keras',

monitor='val\_loss',

mode='min',

save\_best\_only=True,

verbose=1

)

history = lstm\_model.fit(

X\_train\_lstm,

y\_train,

epochs=EPOCHS,

batch\_size=BATCH\_SIZE,

validation\_split=0.2,

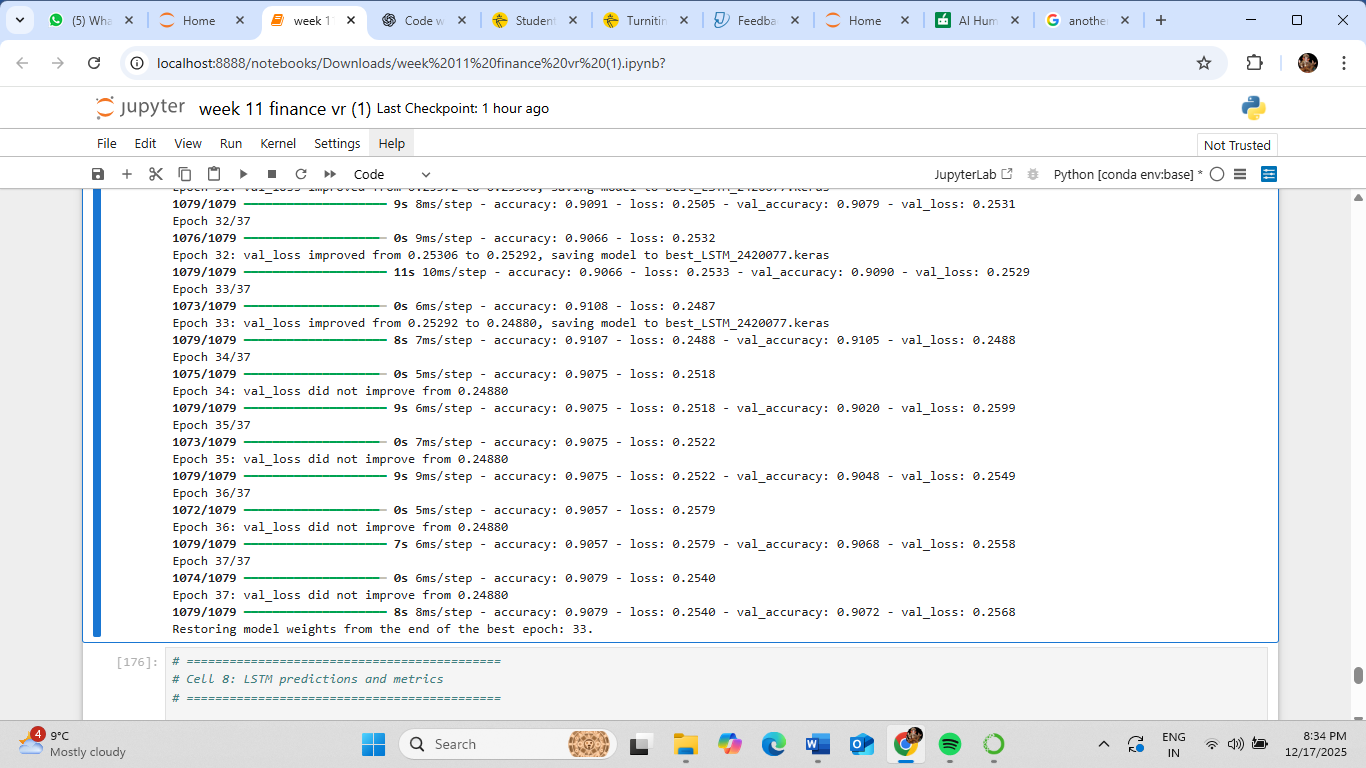
shuffle=True,

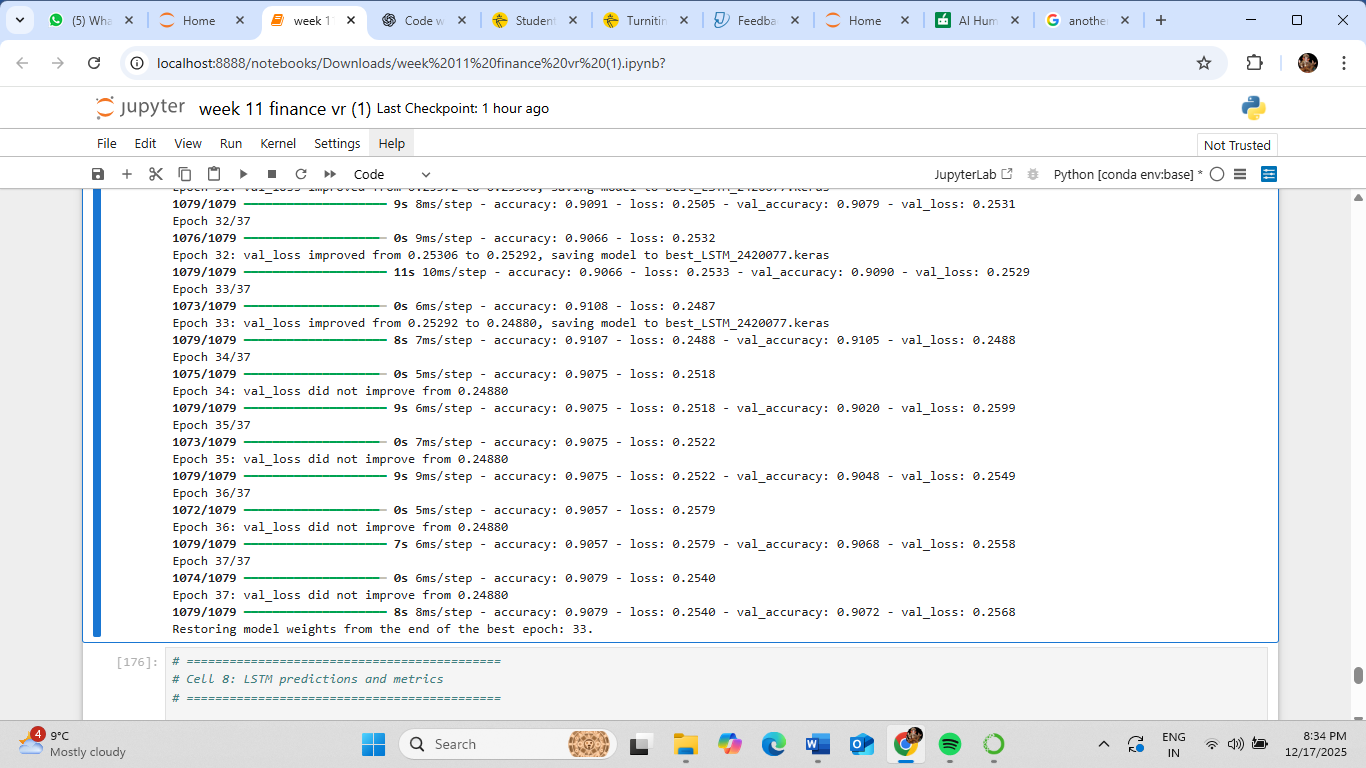
callbacks=[es, mc],

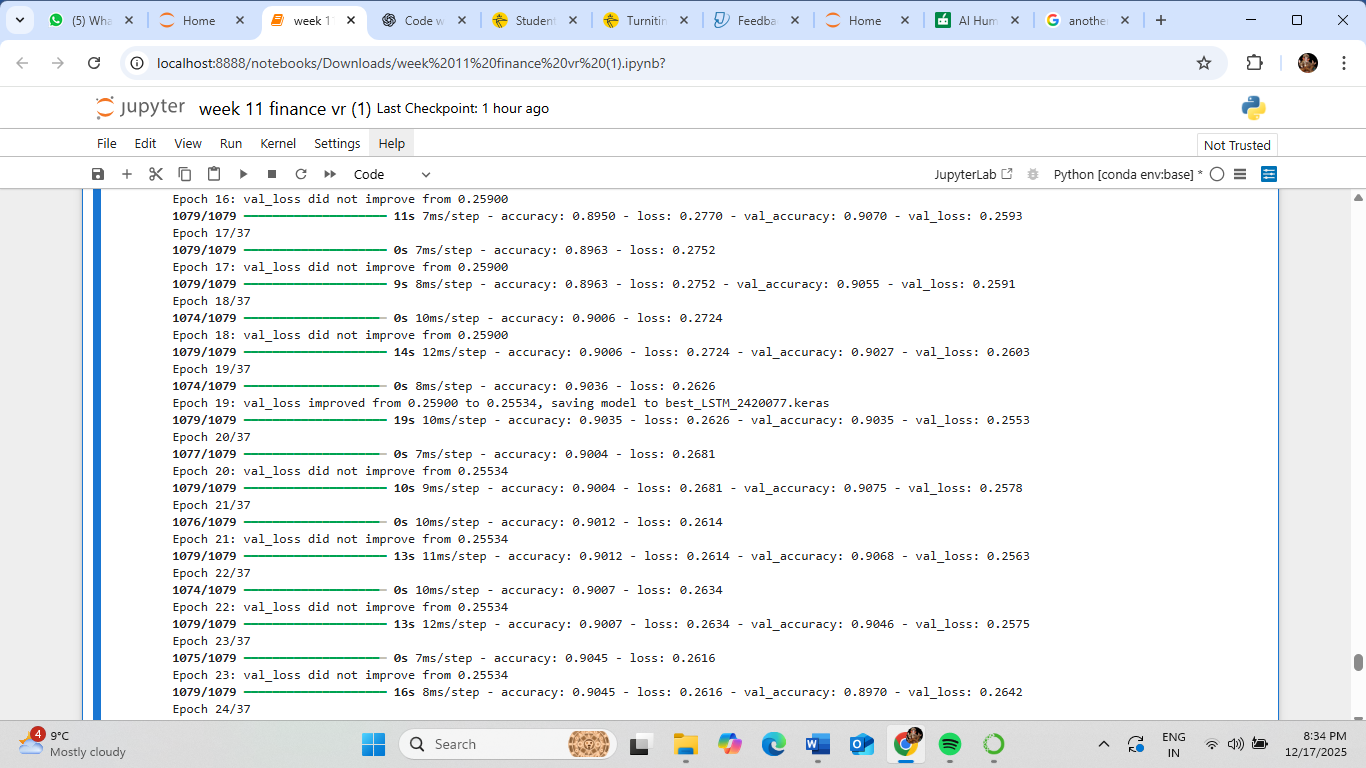
verbose=1

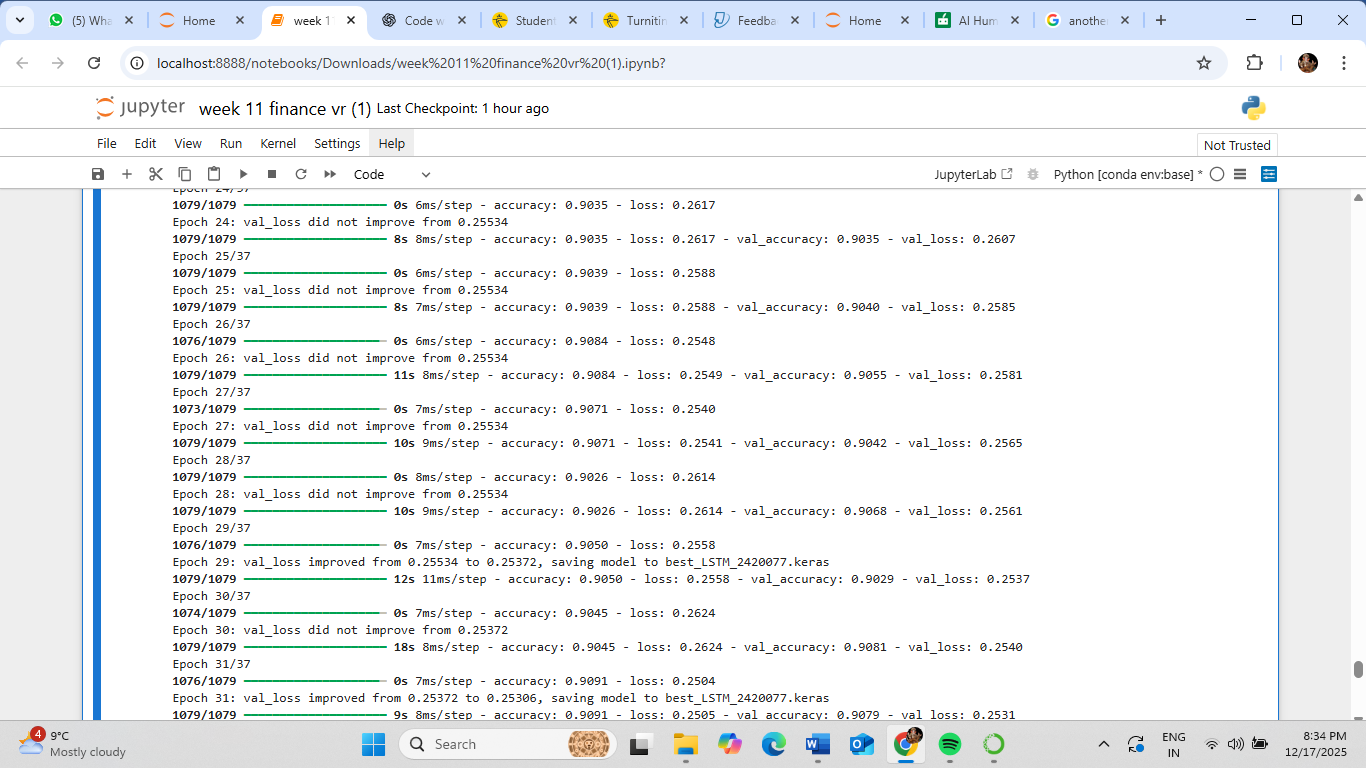
)

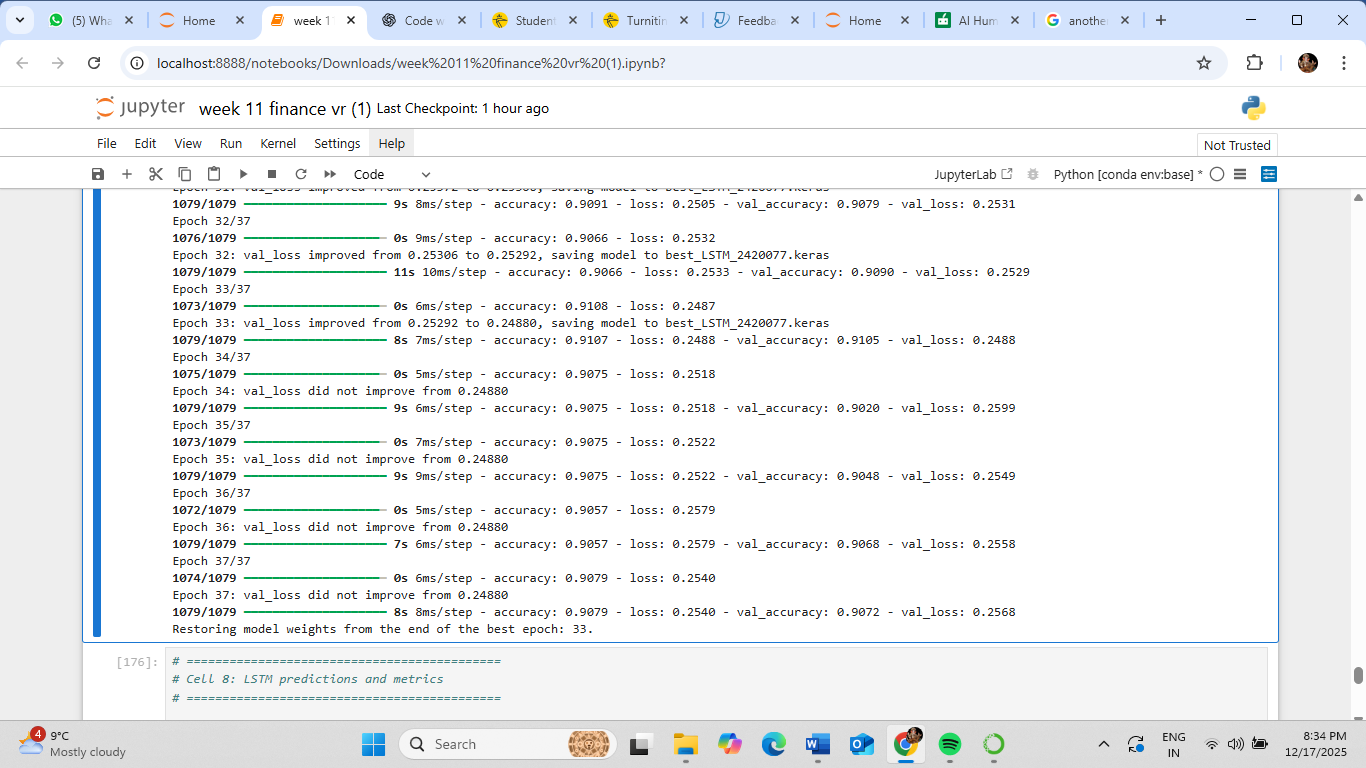
**Output:**



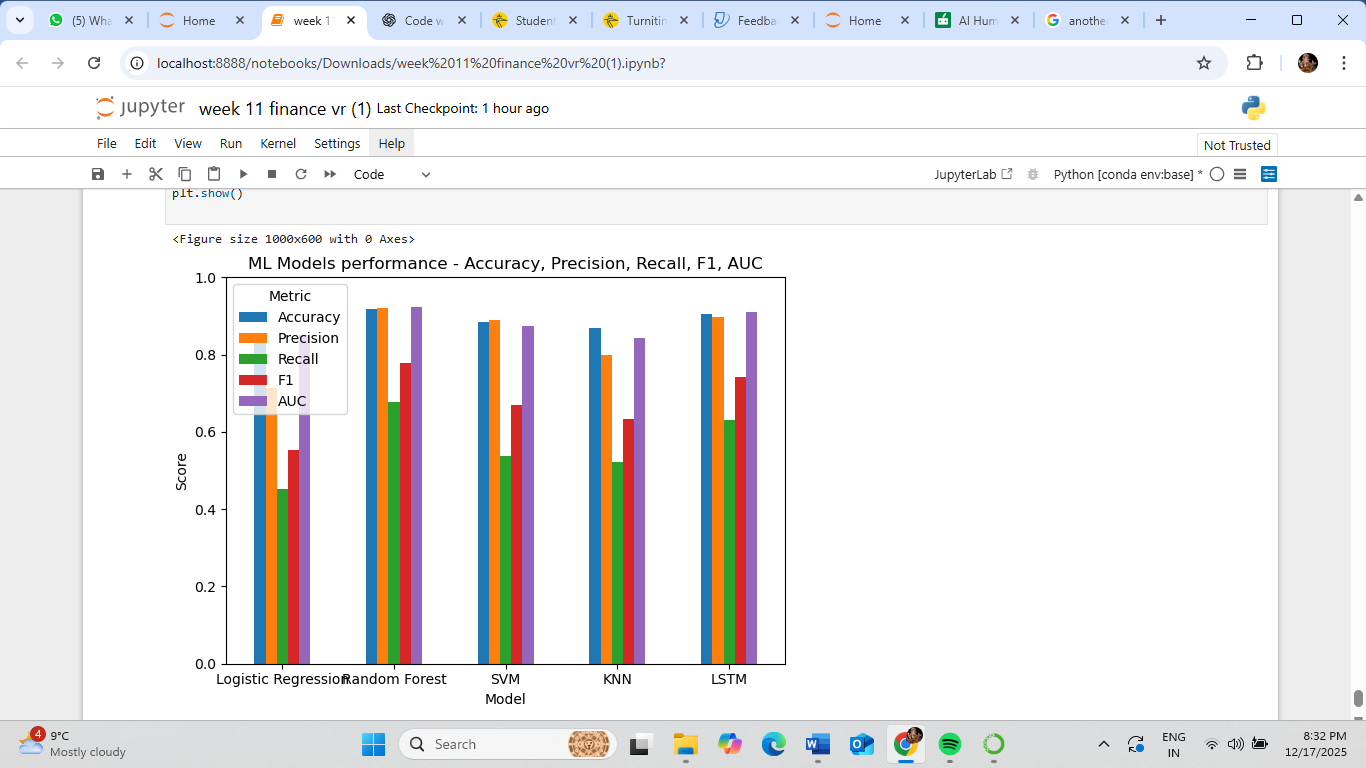












Lab 12:

Final code with report