# Intelligent System

# Assignment 1 - Option B

Week 5 Report

Xuan Tuan Minh Nguyen - 103819212

# Environment Setup

## 1. Create a new environment using Conda CLI

There are several different procedures to create an environment, but the given procedure below will **encapsulate** all of the **required steps** to create **a safe and clean environment** using Conda.

- Navigate to the Github repository that contains the source code for v0.4.
- Once navigated, download the source code by clicking on **Code → Download ZIP** or use the following command in the **CLI (Terminal):**

  **git clone https://github.com/cobeo2004/cos30018.git**

- Once the source code is successfully cloned (downloaded), navigate to the Week 5/v0.4 folder and execute the file conda-config.sh using the following command:

  **bash conda-config.sh**

- The given file config.sh will execute the following procedure:
  - Generate an environment with a pre-defined name (you can change the name if you want to) in Python 3.10.9 by using the command: **conda create -n cos30018_env_w4_v0.4 python=3.10.9**
  - Activate the created environment using: **conda activate cos30018_env_w4_v0.4.**
  - Check and validate if the conda environment is successfully initialized by running **conda info –envs** for listing conda environments and see which environment that we are in and current Python version using **python –version.**

## 2. Installing required dependencies

Once the **environment** is **successfully initialized**, we can start **installing** the **dependencies (libraries)** that are **required by the program**. There are multiple pathways to install dependencies in Python, but the **most popular steps** are:

- Scan through the code to find out the required dependencies; for example, consider the file stock_prediction.py. We could see that there are quite a few required dependencies, such as: numpy matplotlib pandas tensorflow scikit-learn pandas-datareader yfinance TA-lib. However, there will be a new library called mplfinance that helps us to efficiently create a beautiful and easy to analyze **candlestick chart** without having to manually set up.

```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import yfinance as yf
import talib as ta
import mplfinance as mpf
import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn import metrics

import tensorflow as tf
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Dense, LSTM, Dropout, InputLayer, Input, Activation, Bidirectional, GRU, SimpleRNN
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.utils import plot_model

import pickle
import os
from typing_extensions import Annotated, Doc, TypeVar, Literal, Tuple, Literal, Required, TypedDict, Optional, List, NotRequired
```

***Figure 1: Required Dependencies in stock_prediction.ipynb***

- Once dependencies are scanned, use the following command to install the dependencies: pip install numpy matplotlib pandas tensorflow scikit-learn pandas-datareader yfinance TA-lib mplfinance.
- Another step is to list all required libraries into a requirements.txt file, and using the following command to install the required dependencies: pip install -U -r requirements.txt.

```
numpy
matplotlib
pandas
tensorflow
scikit-learn
pandas-datareader
yfinance
TA-lib
mplfinance
```

***Figure 2: Example of requirements.txt***

# Understanding the Machine Learning 1

## *1. Function to prepare dataset for predictions*

### *Codebase*

```python
# Function to prepare dataset for prediction
def make_predict_dataset(start_date: str, end_date: str, tick: str, k: int):
  # Load raw data
  print("Started loading data")
  data = load_data(start_date, end_date, tick)
  print(f"Raw shape: {data.shape}")
  print("Raw head:", data.head())

  # Process and validate the data
  df = validate_data(start_date, end_date, tick)
  print(f"Processed shape: {df.shape}")
  print("Processed head:", df.head())

  # Define feature columns and target column
  feature_cols = ['Open', 'High', 'Low', 'Close_RSI', 'Close_EMA20', 'Close_EMA100', 'Close_EMA200']
  target_cols = 'TargetNextClose'

  # Scale the feature data
  scaled_data, scaled_data_train_scaler = scaling_data(df[feature_cols])
  print("Scaled data: ", scaled_data.shape)

  # Scale the target data
  scaled_target_next_close_train, scaled_target_next_close_scaler = scaling_data(df[target_cols].values.reshape(-1, 1))
  print("Scaled target next close: ", scaled_target_next_close_train.shape)

  # Create sequences for x_test and y_test
  x_test, y_test = [], []
  for i in range(num_look_back_days, len(scaled_data)):
    x_test.append(scaled_data[i - num_look_back_days:i])
    y_test.append(scaled_target_next_close_train[i])

  # Convert lists to numpy arrays
  x_test, y_test = np.array(x_test), np.array(y_test)
  print("Converted x_test: ", x_test.shape)
  print("Converted y_test: ", y_test.shape)

  # Ensure the index is in datetime format for both data and df
  if not isinstance(data.index, pd.DatetimeIndex):
    if "Date" in data.columns:
      data["Date"] = pd.to_datetime(data["Date"])
      data.set_index("Date", inplace=True)

  if not isinstance(df.index, pd.DatetimeIndex):
    if "Date" in df.columns:
      df["Date"] = pd.to_datetime(df["Date"])
      df.set_index("Date", inplace=True)

  # Return processed dataframe, scaled data, scaler for target, and test sequences
  return df, scaled_data, scaled_target_next_close_scaler, x_test, y_test
```

***Figure 3: Codebase for function to prepare dataset for predictions***

## *Parameters*

- start_date: str: The start date for the data to be loaded in the format of YYYY-MM-DD.
- end_date: str: The end date for the data to be loaded in the format of YYYY-MM-DD.
- tick: str: The symbol of the stock for which the data is to be loaded.
- k: int: An integer parameter that can be used for additional processing or resampling within the function.

## *Functionalities*

- The make_predict_dataset() function is designed to create a dataset for stock price prediction. It performs several steps to load, process, scale the data and return the processed data and sequences required for model prediction. Below is a step-by-step description of what the function does:

+ **Load and Validate Data**: Firstly, the function will load the stock data based on the specified date range with the ticker symbol using the created load_data() function. Once the data is loaded, the function then processes the data using the created validate_data()function.
+ **Define Feature and Target Columns**: Once the data is loaded and validated, the function will use the defined feature columns (feature_cols) and the target column (target_cols) for scaling and creating sequences.
+ **Scale Feature and Target data**: The feature and target data will then be scaled using the defined scaling_data() function, which normalizes the data to a specified range (typically 0 to 1).
+ **Create testing sequences**: The function will then create sequences for x_test and y_test based on the defined number of look back days (num_look_back_days). It iterates over the scaled data to create sequences of the specified length, then adding them to the two variables x_test and y_test. The function finally transforms the lists to numpy arrays.
+ **Ensure index type and return the value**: If the index is not in datetime format, it converts the "Date" column to datetime and sets it as the index to ensure that both raw data (data) and converted data (df) has the index type of Datetime. Once the indexes are ensured it will return the processed data (df), scaler for the target data (scaled_target_next_close_scaler) and the test sequences (x_test, y_test) for further usages.

## *2. Function to multivariate prediction for single day price*

### *Codebase*

```python
# Function to predict stock price for a single day using multivariate data
def multivariate_predict_for_single_day(model: any, tick: str, date: str):
  # Convert input date to datetime object
  pred_date = datetime.strptime(date, "%Y-%m-%d")
  # Set start date 3 years before prediction date
  start_date = pred_date - timedelta(365 * 3)
  # Print date range for prediction
  print(f"Date for multivariate predict: {start_date.strftime('%Y-%m-%d')} - {pred_date.strftime('%Y-%m-%d')}")

  # Prepare dataset for prediction
  df, scaled_date, scaler, x_test, y_test = make_predict_dataset(start_date.strftime('%Y-%m-%d'), pred_date.strftime('%Y-%m-%d'), tick)
  # Check if there's data available for prediction
  if len(x_test) == 0 or x_test[-1].shape[0] == 0:
    raise IndexError("No data is found for prediction")

  # Make prediction
  print("Predict")
  pred = model.predict(x_test[-1].reshape(1, -1, x_test.shape[-1]))
  print("Predicted")

  # Inverse transform the prediction to get actual price
  pred_price = scaler.inverse_transform(pred)

  # Print and return the predicted price
  print(f"Predicted Price for: {pred_date.strftime('%Y-%m-%d')} is: {pred_price[0][0]}")
  return pred_price
```

***Figure 4: Codebase for multivariate prediction function for single day price***

## *Parameters*

- model: Any: The created machine learning model using create_dynamic_model() that will be used to predict the stock price.
- tick: str: The symbol of the stock for which the data is to be loaded
- date: str: The target date for the prediction, it should be in the format of YYYY-MM-DD.

## *Functionalities*

- The purpose of this function is to predict the stock price at the specified day using a multivariate approach. The function performs the following procedures:
  + **Transform and set start date**: The function starts by enforcing the input date (a string) into a datetime object using datetime.strptime() function from the datetime library with a format of YYYY-MM-DD. After that, using the timedelta() function, it calculates the start date, which is shifted back to 3 years before the prediction date.
  + **Process data and check data availability**: Once the date has been processed, the function will call the defined make_predict_dataset() function with the start date, prediction date, ticker symbol as the parameters. This function prepares the dataset required for making the prediction, including scaling the data and creating sequences for the model input and returning the results as stated above.
  + **Predict and transform the prediction result**: The function then reshapes the last element of x_test and passes it to the model's predict method. Once the element is passed, it uses the model to

predict the stock price for the target date and transformed using inverse_transform() function from scikit-learn

+ **Return the predicted price**: Finally, the function will return the predicted price.

## *3. Function to create multivariate multistep prediction*

### *Codebase*

```python
def multivariate_multistep_prediction(model: any, tick: str, start_date: str, end_date: str, k:int | None = 10):
  # Print the start and end dates for the prediction
  print(f"Start multivariate multistep prediction from: {start_date} to {end_date}")

  # Prepare the dataset for prediction
  df, scaled_data, scaler, x_test, y_test = make_predict_dataset(start_date, end_date, tick, k)

  # Extract the actual closing prices
  current_price = df['Close'].values

  # Make predictions for the past data
  past_predict_res = model.predict(x_test)
  # Inverse transform the predictions to get actual prices
  past_predict_res = scaler.inverse_transform(past_predict_res)
  past_predict_res = np.array(past_predict_res)

  # Initialize lists for future predictions
  future_predict_res = []
  future_input_data = x_test[-1]
  future_predict_dates = []

  # Convert the end date to datetime object
  last_predict_date = datetime.strptime(end_date, "%Y-%m-%d")

  # Generate future dates for prediction
  for i in range(1, k + 1):
    future_predict_dates.append(last_predict_date + timedelta(i))

  # Make predictions for future dates
  for i in range(k):
    # Predict the next price
    predicted_res = model.predict(future_input_data.reshape(1, -1, x_test.shape[-1]))
    # Inverse transform the prediction to get actual price
    predicted_price = scaler.inverse_transform(predicted_res)

    # Add the predicted price to the results
    future_predict_res.append(predicted_price)

    # Get the current prediction date
    curr_date = future_predict_dates[i]
    # Print the predicted price for the current date
    print(f"Predicted price for {curr_date.strftime('%Y-%m-%d')} is: {predicted_price}")

    # Update the input data for the next prediction
    future_input_data = np.roll(future_input_data, -1, axis=0)
    future_input_data[-1] = predicted_res

  # Convert future predictions to numpy array
  future_predict_res = np.array(future_predict_res)

  # Return the original dataframe, current prices, past predictions, and future predictions
  return df, current_price, past_predict_res, future_predict_res
```

***Figure 5: Codebase for multivariate multistep prediction function***

### *Parameters*

- model: Any: The created machine learning model using `create_dynamic_model()` that will be used to predict the stock price.
- tick: str: The symbol of the stock for which the data is to be loaded
- start_date: str: The start date for the prediction; it should be in the format of YYYY-MM-DD.
- end_date: str: The end date for the prediction; it should be in the format of YYYY-MM-DD.

- k: int | None = 10: The number of future days for predictions to be made. Default is 10.

### *Functionalities*

- **Prepare Dataset for Prediction**: Initially, this function will call the make_predict_dataset() function to prepare the dataset for prediction. Once the data set is prepared, this function will receive the processed dataframe containing historical stock data (df), the scaled feature data (scaled_data), the scaler function used for transforming the predictions (scaler), and the input and target values for testing (x_test, y_test).
- **Take the actual closing prices and make predictions on the past data**: It then extracts and stores the actual closing prices from the dataframe df to the current_price array. Once the closing prices are stored, the function will use the model to predict the stock prices for the past data (x_test), transforming the predictions to get the actual prices and storing them in past_predict_res array.
- **Initialise future prediction results and generate future dates for prediction**: It then initialises two empty lists called future_predict_res and future_predict_dates to store the future prediction results and future prediction dates. After that, it will generate and add the future predict date by taking the latest predict date and adding i days from 1 to k + 1.
- **Predicting future dates**: Once the future days are added, the function will loop through k times to make predictions for future dates by using the inputted model to predict the next price based on input data (future_input_data). Once the predicting process is finished, it will transform the prediction to get the actual price and add it to the result array. Then it will update the input data (future_input_data) by rolling and setting the last element to the predicted result.
- **Converting data and return data**: Finally, the function will convert the predict result array (future_predict_res) to a numpy array and return the original dataframe that contains historical stock data (df), the real price (current_price), the past result array (past_predict_res) and the future result array (future_predict_res) for further usages.

# Deploying and Testing the Codebase

## *1. Result*

- Testing result for the multivariate_predict_for_single_day() function

```python
predict_date = '2023-09-22'
predicted_price = multivariate_predict_for_single_day(model, tick=ticker, date=predict_date)
```

*Figure 6: Codebase for testing the function*

```
Date for multivariate predict: 2020-09-22 - 2023-09-22
Started loading data
[*********************100%%**********************]  1 of 1 completed
Raw shape: (755, 6)
Raw head:                        Open        High         Low       Close   Adj Close  \
Date
2020-09-22  143.199997  145.919998  139.199997  141.410004  141.410004
2020-09-23  135.053329  137.383331  125.293335  126.786667  126.786667
2020-09-24  121.266670  133.166672  117.099998  129.263336  129.263336
2020-09-25  131.156662  136.243332  130.433334  135.779999  135.779999
2020-09-28  141.539993  142.693329  138.516663  140.399994  140.399994

               Volume
Date
2020-09-22  238742400
2020-09-23  285222600
2020-09-24  289683300
2020-09-25  201625500
2020-09-28  149158800
Processing Raw Data ...
Type of index after converted:  <class 'pandas.core.indexes.datetimes.DatetimeIndex'>
Processed shape: (555, 13)
Processed head:                        Open        High         Low       Close   Adj Close  \
Date
2021-07-08  209.456665  218.143326  206.820007  217.603333  217.603333
2021-07-09  217.726669  219.636673  214.896667  218.983337  218.983337
2021-07-12  220.733337  229.080002  220.720001  228.566666  228.566666
2021-07-13  228.773331  231.093338  222.100006  222.846664  222.846664
...
Predict
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
Predicted
Predicted Price for: 2023-09-22 is: 269.75164794921875
```

*Figure 7: Result for the test*

- Testing result for the multivariate_multistep_prediction() function

```python
predict_date = '2023-09-22'
predicted_price = multivariate_predict_for_single_day(model, tick=ticker, date=predict_date)
```

*Figure 8: Codebase for testing the function*

```
Start multivariate multistep prediction from: 2021-09-24 to 2024-09-23
Started loading data
Raw shape: (752, 7)
Raw head:          Date        Open        High         Low       Close   Adj Close  \
0  2021-09-24  248.630005  258.266663  248.186661  258.130005  258.130005
1  2021-09-27  257.706665  266.333344  256.436676  263.786682  263.786682
2  2021-09-28  262.399994  265.213318  255.393326  259.186676  259.186676
3  2021-09-29  259.933319  264.500000  256.893341  260.436676  260.436676
4  2021-09-30  260.333344  263.043335  258.333344  258.493347  258.493347

     Volume
0  64119000
1  84212100
2  76144200
3  62828700
4  53868000
Loading Prepared Data ...
Processed shape: (552, 13)
Processed head:          Open        High         Low       Close   Adj Close    Volume  \
0  236.846664  239.773331  228.369995  233.070007  233.070007  87930900
1  225.500000  242.059998  225.033340  237.039993  237.039993  97954500
2  234.896667  238.653336  229.333328  238.313339  238.313339  78557400
3  240.000000  243.623337  236.889999  240.066666  240.066666  69683100
4  244.936661  250.516663  239.603333  240.546661  240.546661  82537500
...
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step
Predicted price for 2024-10-02 is: [[227.3487]]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step
Predicted price for 2024-10-03 is: [[227.6761]]
```

***Figure 9: Result for the test***