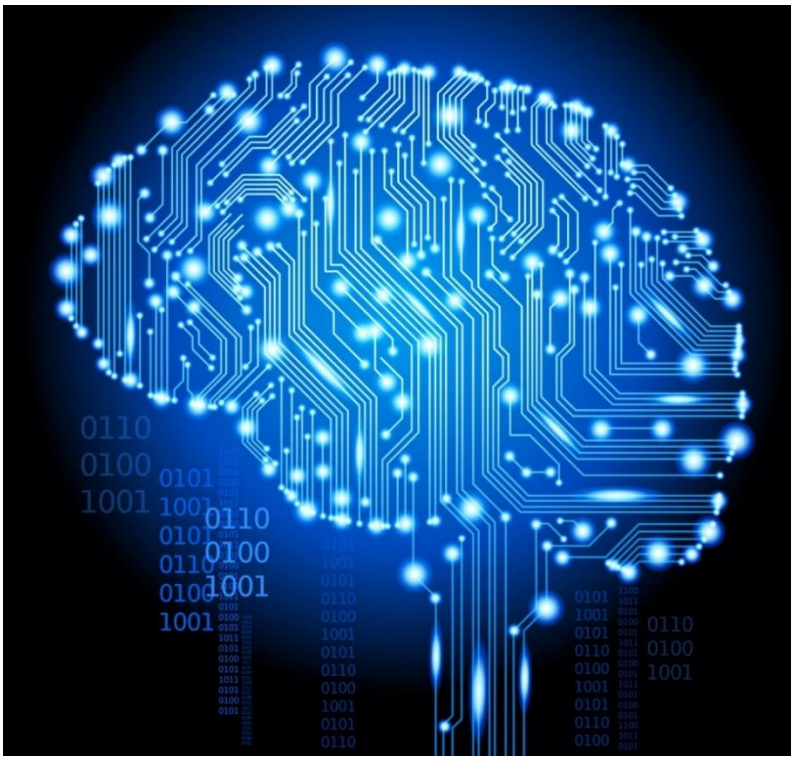
COS30018 – Intelligent Systems

REPORT

PROJECT ASSIGNMENT – OPTION B



Trung Kien Nguyen 104053642

TABLE OF CONTENTS

Table of contents……2

I. Introduction……..3

II. Overall system architecture……..3

III. Implemented data processing techniques……4

IV. Experimented machine learning……6

V. Extension……..8

VI. Demonstration……8

VII. Summary……10

INTRODUCTION

My project is Stock Price Prediction using Python. The main idea is based on the use models of LSTM/RNN/GRU, ensembling with ARIMA and Random Forest technique, providing a solution to predict future stock price, including the multivariate and multistep prediction. This project is only developed for learning and research purposes, it does not meet the standards and reliability for practical purposes.

OVERALL SYSTEM ARCHITECTURE

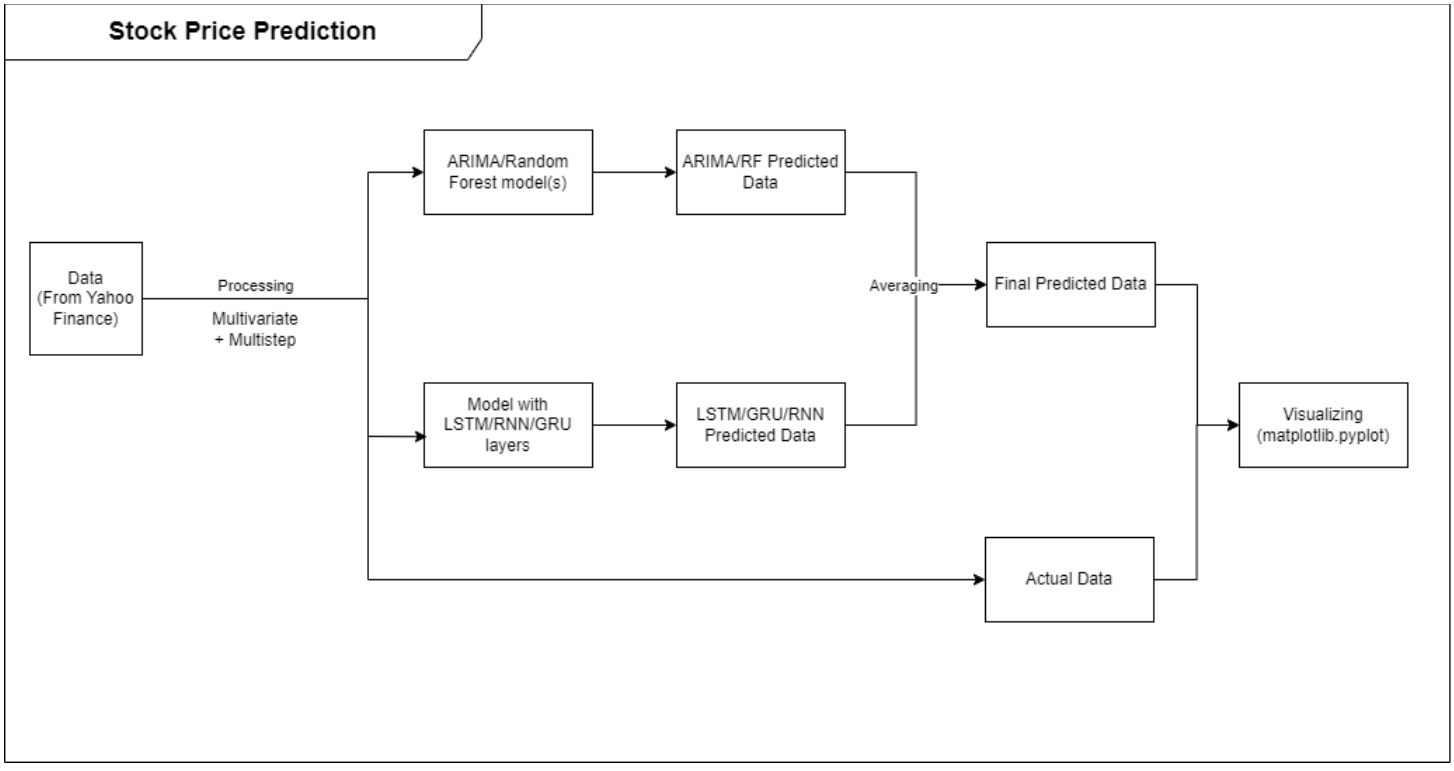


Fig 1. My Stock price prediction project architecture design

The system architecture for my project is provided in the above illustration. The instruction to run my project will be present in the “DEMONSTRATION” part.

IMPLEMENTED DATA PROCESSING TECHNIQUES

In my project, I created a file named “dataProcessing.py”, consisting of “processData()” method, which is responsible for downloading the data from Yahoo Finance (if the data has not been saved, if yes, load it), scaling and splitting the data, and returning the results.

Initially, the dictionary of “processedData” is created, containing all the outputs of the data processing stage, which would finally be the output for the method

The data is retrieved from Yahoo Finance, and in a Pandas Dataframe format

```
# Fetch data from Yahoo Finance
data = yf.download(company, start=startDate, end=endDate, progress=False)
```

Or loaded if it has been dowloaded before:

```
data = pd.read_csv(dataFilePath, index_col=0, parse_dates=True)
```

Next, the data will be scaled using MinMaxScaler:

```
for col in featureColumns:
    scalerFilePath = "stockdatas/{}_scaler_{}.pkl".format(col, dataFileName)

    # Scale the data with the new scaler
    if (not os.path.exists(scalerFilePath)):
        # Scale data
        scaler = MinMaxScaler(feature_range=featureRange)

        data[col] = scaler.fit_transform(np.expand_dims(data[col], axis=1))
```

After that, sequence and target data (“x” and “y”) are created, before being splitted to xTrain, yTrain, xTest and yTest:

```
if (randomSplit):
    (xTrain, xTest, yTrain, yTest) = train_test_split(x, y, test_size=(1 - trainRatio), shuffle=False)
else:
    trainSamples = int(trainRatio * len(x))

    (xTrain, xTest, yTrain, yTest) = (x[:trainSamples], x[trainSamples:], y[:trainSamples], y[trainSamples:])
```

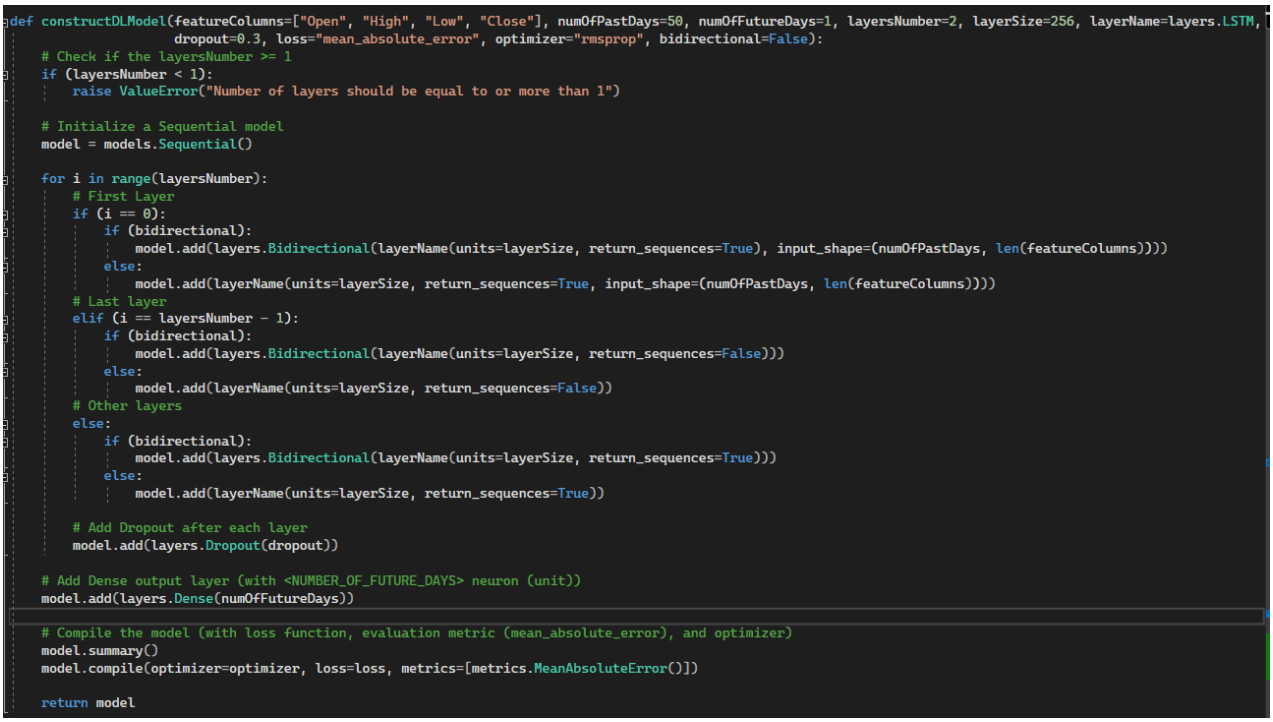
The “processedData” dictionary stored:

- The initial pandas dataframe: `processedData["Data"] = data`
- The scalers used for scaling step: `processedData["ColumnScalers"] = featureColumnScalers`
- The splitted train and test data:

```
# Get the xTest dates
dates = processedData["XTest"][:, -1, -1]
# Retrieve test features dates from the original data
processedData["TestData"] = processedData["Data"].loc[dates]
# Remove dupliacate dates
processedData["TestData"] = processedData["TestData"][~processedData["TestData"].index.duplicated(keep="first")]
# Remove dates from xTrain and xTest and convert to float32
processedData["XTrain"] = processedData["XTrain"][:, :, :len(featureColumns)].astype(np.float32)
processedData["XTest"] = processedData["XTest"][:, :, :len(featureColumns)].astype(np.float32)
```

EXPERIMENTED MACHINE LEARNING

In task 4, I created a method of "constructDLModel()" to create a Deep Learning model using LSTM/RNN/GRU layers:



```
def constructDLModel(featureColumns=["Open", "High", "Low", "Close"], numOfPastDays=50, numOfFutureDays=1, layersNumber=2, layerSize=256, layerName=layers.LSTM,
                     dropout=0.3, loss="mean_absolute_error", optimizer="rmsprop", bidirectional=False):
    # Check if the layersNumber >= 1
    if (layersNumber < 1):
        raise ValueError("Number of layers should be equal to or more than 1")

    # Initialize a Sequential model
    model = models.Sequential()

    for i in range(layersNumber):
        # First Layer
        if (i == 0):
            if (bidirectional):
                model.add(layers.Bidirectional(layerName(units=layerSize, return_sequences=True), input_shape=(numOfPastDays, len(featureColumns))))
            else:
                model.add(layerName(units=layerSize, return_sequences=True, input_shape=(numOfPastDays, len(featureColumns))))
        # Last layer
        elif (i == layersNumber - 1):
            if (bidirectional):
                model.add(layers.Bidirectional(layerName(units=layerSize, return_sequences=False)))
            else:
                model.add(layerName(units=layerSize, return_sequences=False))
        # Other layers
        else:
            if (bidirectional):
                model.add(layers.Bidirectional(layerName(units=layerSize, return_sequences=True)))
            else:
                model.add(layerName(units=layerSize, return_sequences=True))

        # Add Dropout after each layer
        model.add(layers.Dropout(dropout))

    # Add Dense output layer (with <NUMBER_OF_FUTURE_DAYS> neuron (unit))
    model.add(layers.Dense(numOfFutureDays))

    # Compile the model (with loss function, evaluation metric (mean_absolute_error), and optimizer)
    model.summary()
    model.compile(optimizer=optimizer, loss=loss, metrics=[metrics.MeanAbsoluteError()])

    return model
```

https://github.com/trungkiennguyen22082004/009_Python_Stock_Price_Prediction/wiki/Task-B4-%E2%80%90-Machine-Learning-1

The multivariate and multistep problems were quite easy to solve.

- For multivariate, in the processing data, I simply splitted the "x" with all used feature columns, then when training and testing the model, use that splitted train and test data.
- For multistep, in the processing data, I reshape the targets ("y") into 2D array, instead of a 1D list of a feature column from the data (each element of the reshaped array is a 1D array of k values, where k = number of want-to-predict days), then fit the model normally.

https://github.com/trungkiennguyen22082004/009_Python_Stock_Price_Prediction/wiki/Task-B5-%E2%80%90-Machine-Learning-2

For the task B6, I chose 2 types of models to ensamble with the existing LSTM/RNN/GRU models. After create, fit and do prediction with the new models, I simply get the average values of the predicted data as the final output

- ARIMA: For this model, I used a lot of different ARIMA models, each corresponds to a future day in the prediction period.

```
# Create and fit ARIMA models
for i in range(0, len(singleFeatureStationaryData)):
    modelFilePath = "models/{}-type_{}-company_{}-feature_{}-traingDataShape_{}-index.pkl".format("ARIMA",

    if (not os.path.exists(modelFilePath)):
        # Find the best parameters for the model for each sequence
        model = auto_arima(singleFeatureStationaryData[i], start_p=0, start_q=0, test="adf", max_p=7, max_
                           m=1, d=None, seasonal=False, start_P=0, D=0, trace=True,
                           error_action="ignore", suppress_warnings=True, stepwise=True)

        bestOrder = model.order
        bestSeasonalOrder = model.seasonal_order

        # Create model for each sequence
        model = ARIMA(singleFeatureStationaryData[i], order=bestOrder, seasonal_order=bestSeasonalOrder)
        # Fit the model
        fittedModel = model.fit()
```

- RandomForest: I used the RandomForestRegressor model from “sklearn.ensemble” module. The parameters for this model were decided using hyperparameters technique

```
# Number of trees in Random Forest
numOfEstimators = [500, 1000, 2000]
# Maximum depth of levels
maxDepth = [10, 20, 50]
# Minimum number of samples required for node splitting
minSamplesSplitting = [50, 100, 200]
# Minimum number of samples required at each node (leaf)
minSamplesEachNode = [2, 4, 8]

testAccurateData = pd.DataFrame(columns=["NumOfEstimators", "MaxDepth", "MinSamplesSplitting", "MinSamplesEachNode", "AccuracyTrain", "AccuracyTest"])

for element in list(itertools.product(numOfEstimators, maxDepth, minSamplesSplitting, minSamplesEachNode)):
    # Create the model
    model = RandomForestRegressor(n_estimators=element[0], max_depth=element[1], min_samples_split=element[2], min_samples_leaf=element[3])
    # Fit the model
    model.fit(xTrain, yTrain)
```

```
    # Train the model
    #   Predict with RF prediction method
    predictedTrainData = model.predict(xTrain)
    #   Compute the absolute errors
    errorsTrain = abs(predictedTrainData - yTrain)
    #   Compute the mean absolute error (in precentage)
    mapeTrain = 100 * (errorsTrain / yTrain)
    #   Compute the accuracy
    accuracyTrain = 100 - np.mean(mapeTrain)

    # Test data
    #   Predict with RF prediction method
    predictedTestData = model.predict(xTest)
    #   Compute the absolute errors
    errorsTest = abs(predictedTestData - yTest)
    #   Compute the mean absolute error (in precentage)
    mapeTest = 100 * (errorsTest / yTest)
    #   Compute the accuracy
    accuracyTest = 100 - np.mean(mapeTest)
```

```
    testAccurateDataElement = pd.DataFrame(index = range(1), columns = ["NumOfEstimators", "MaxDepth", "MinSamplesSplitting", "MinSamplesEachNode", "Acc

    testAccurateDataElement.loc[:, "NumOfEstimators"] = element[0]
    testAccurateDataElement.loc[:, "MaxDepth"] = element[1]
    testAccurateDataElement.loc[:, "MinSamplesSplitting"] = element[2]
    testAccurateDataElement.loc[:, "MinSamplesEachNode"] = element[3]
    testAccurateDataElement.loc[:, "AccuracyTrain"] = accuracyTrain
    testAccurateDataElement.loc[:, "AccuracyTest"] = accuracyTest

    testAccurateData = pd.concat([testAccurateData, testAccurateDataElement], ignore_index=True)

bestParameters = testAccurateData.loc[testAccurateData["AccuracyTest"] == max(testAccurateData["AccuracyTest"])]
```

https://github.com/trungkiennguyen22082004/009_Python_Stock_Price_Prediction/wiki/Task-B6-%E2%80%90-Machine-Learning-3

EXTENSION

DEMONSTATION

My project use the following libraries/packages:

- Numpy (1.24.3)
- Pandas (2.1.0)
- Yfinance (0.2.28)
- Scikit-learn (1.3.0)
- Matplotlib (3.7.2)
- Mplfinance (0.12.10b0)
- Keras (2.13.1)
- Pmdarima (2.0.3)
- Statsmodels (0.14.0)
- Tensorflow (2.13.0)

Instruction to run my project:

- Create a virtual environment:
    - python -m venv myenv

```
C:\Users\ADMIN\OneDrive\Desktop>python -m venv myenv

C:\Users\ADMIN\OneDrive\Desktop>
```

- Activate the virtual environment: change directory to myenv\Scripts and run the “activate.bat”
  - cd …\myenv\Scripts
  - activate.bat

```
C:\Users\ADMIN>cd OneDrive\Desktop\myenv\Scripts

C:\Users\ADMIN\OneDrive\Desktop\myenv\Scripts>activate.bat
```

- Install the necessary libraries/packages:
  - pip install numpy==1.24.3 pandas==2.1.0 yfinance==0.2.28 tensorflow==2.13.0 scikit-learn==1.3.0 keras==2.13.1 matplotlib==3.7.2 mplfinance==0.12.10b0 pmdarima==2.0.3 statsmodels==0.14.0

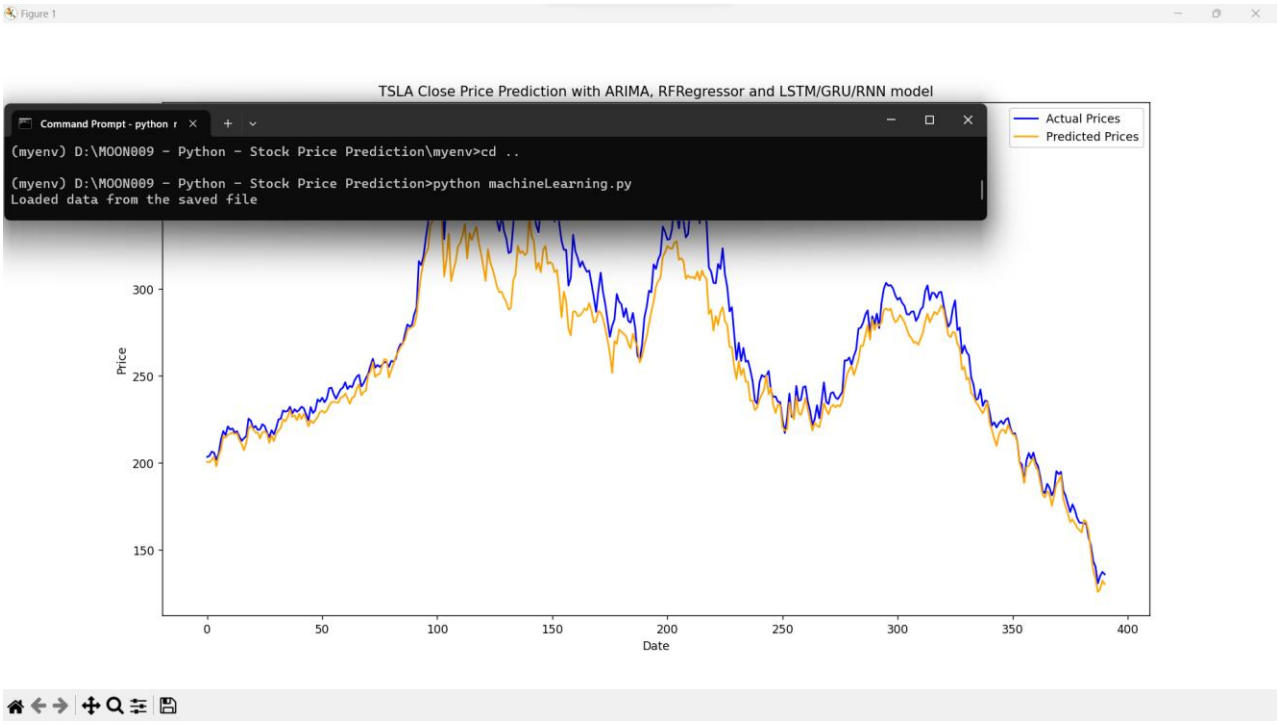
```
(myenv) D:\MOON009 - Python - Stock Price Prediction\myenv\Scripts>pip install numpy==1.24.3 pandas==2.1.0 yfinance==0.2.28 tensorflow==2.13.0 scikit-learn=
=1.3.0 keras==2.13.1 matplotlib==3.7.2 mplfinance==0.12.10b0 pmdarima==2.0.3 statsmodels==0.14.0
```

- Return to the project root directory:

```
(myenv) D:\MOON009 - Python - Stock Price Prediction\myenv\Scripts>cd ..

(myenv) D:\MOON009 - Python - Stock Price Prediction\myenv>cd ..
```

- Run the “machineLearning.py” file:



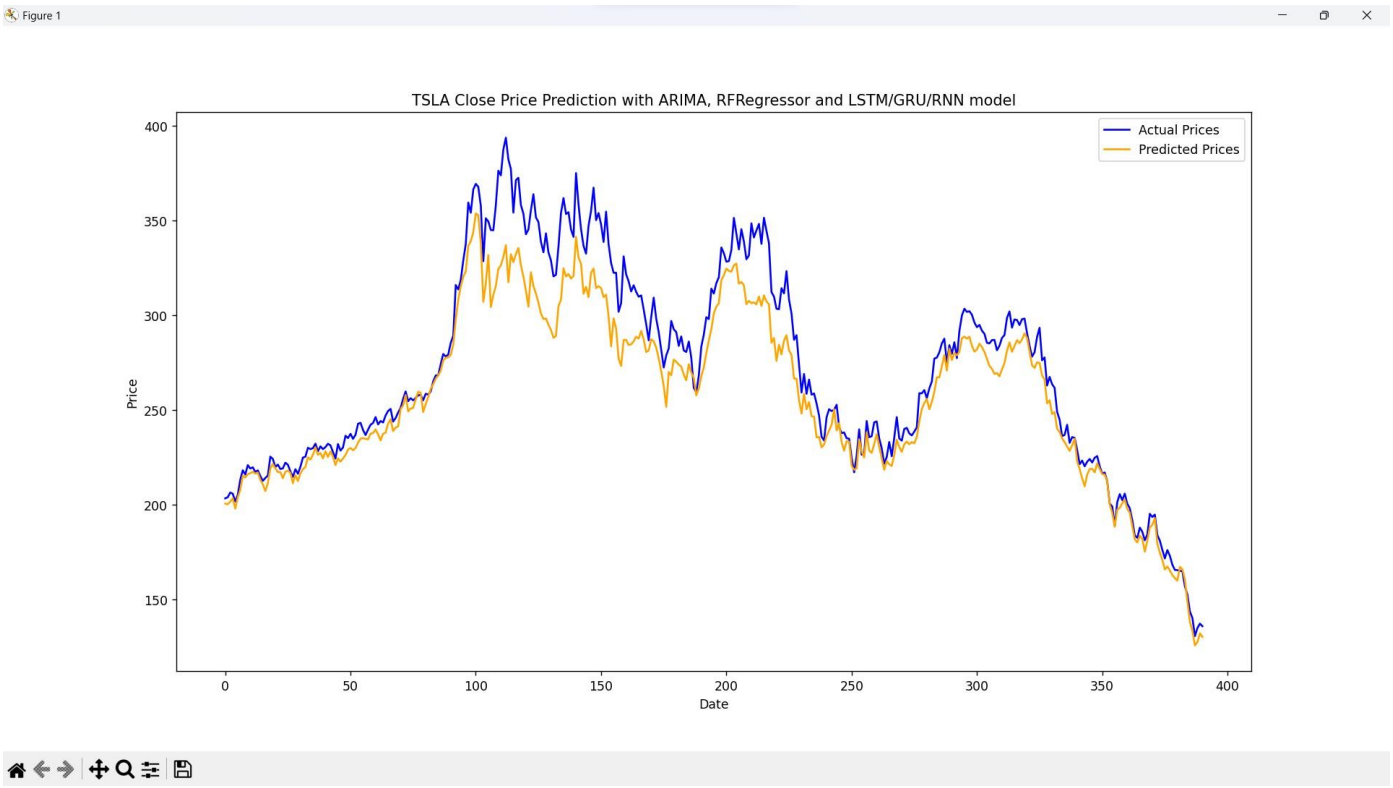


Fig 2. Final output

SUMMARY

Throughout this semester, I believe that my work on the project B aligns well with the specified ULOs:

- My project explores and implements a range of techniques used in intelligent systems, including the use of LSTM/RNN/GRU models, ARIMA, and Random Forest ensembling models for stock price

prediction, along with the detailed explanations in the weekly reports. I believed this demonstrates my practical understanding of “the AI/IA techniques”.

- While working with the project in the teaching period, I have applied some algorithms and techniques learned in the weekly lectures and tutorials, as well as gained by further researches. This is why, I am confident now in the ability to “use different AI/IA algorithms to address real-world challenges”