COS30018 - Option B - Task 6: Machine Learning 3

Student Name: Nguyen Duc Le Nguyen

Id: 104224493

Summary of Effort to Implement the Ensemble Models

Introduction

In this report, I outline my efforts to implement ensemble models combining ARIMA/SARIMA and deep learning models such as LSTM and GRU. The goal was to create a robust predictive model for stock prices, utilizing both traditional statistical methods and advanced machine learning techniques.

Implementing the Ensemble Models

1. ARIMA and LSTM Model

a. Data Preparation

For this task, I used stock price data for Apple Inc. (AAPL) from Yahoo Finance, covering the period from January 1, 2015, to January 1, 2024. I focused on the features: Open, High, Low, Close, Adj Close, and Volume. I split the data into training and test sets, with 90% of the data used for training and the remaining 10% for testing.

```
df = yf.download("AAPL", start="2015-01-01", end="2024-01-01")
company = 'Apple'
stock_data = df[['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume']]

# Create a new dataframe with relevant features
data = stock_data
data.index = pd.to_datetime(data.index)

train_data, test_data = data[0:int(len(data)*0.9)], data[int(len(data)*0.9):]
```

Figure 1 - Prepare Data

c. ARIMA model

I implemented the ARIMA model using the statsmodels library. I trained the model on the 'Open' prices from the training set and performed rolling forecasts to predict prices for the test set.

Here's a quick overview:

1. Seasonal Decomposition: I decomposed the series to understand its seasonal patterns.
2. Model Training: I trained the ARIMA model on historical data and made forecasts.

```
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error, mean_absolute_error
import math

train_arima = train_data['Open']
test_arima = test_data['Open']

history = [x for x in train_arima]
predictions = list()

# Make the first prediction
model = ARIMA(history, order=(5,1,0))
model_fit = model.fit()
yhat = model_fit.forecast(steps=1)[0]
predictions.append(yhat)
history.append(test_arima.iloc[0])  # Append the first observation to history

# Rolling forecasts
for i in range(1, len(test_arima)):
    # Fit model on the updated history
    model = ARIMA(history, order=(1,1,0))
    model_fit = model.fit()
    yhat = model_fit.forecast(steps=1)[0]
    predictions.append(yhat)
    # Append the actual observation to history
    history.append(test_arima.iloc[i])

# Convert predictions to a Pandas Series for alignment
predictions_series = pd.Series(predictions, index=test_arima.index[:len(predictions)])
```

Figure 2 - ARIMA model

I then calculated performance metrics:

- MSE (Mean Squared Error)
- MAE (Mean Absolute Error)
- RMSE (Root Mean Squared Error)

```
# Report performance
mse = mean_squared_error(test_arima, predictions_series)
print('MSE: ' + str(mse))
mae = mean_absolute_error(test_arima, predictions_series)
print('MAE: ' + str(mae))
rmse = math.sqrt(mse)
print('RMSE: ' + str(rmse))
```

Figure 3 - Perfomance Metrics

b. LSTM Model

Next, I used an LSTM model using the keras library. I scaled the 'Open' prices, created sequences for training, and then defined and trained the LSTM network.

Here's a summary:

1. Data Scaling: I used MinMaxScaler to normalize the data.
2. Model Architecture: I defined an LSTM model with dropout layers to prevent overfitting.
3. Model Training: I trained the model on the prepared sequences.

```
model = Sequential()

# Adding the first LSTM layer
# Here return_sequences=True means whether to return the last output in the output sequence, or the full sequence.
# it basically tells us that there is another(or more) LSTM layer ahead in the network.
model.add(LSTM(units = 50, return_sequences = True, input_shape = (X_train.shape[1], 1)))
# Dropout regularisation for tackling overfitting
model.add(Dropout(0.20))

model.add(LSTM(units = 50, return_sequences = True))
model.add(Dropout(0.25))

model.add(LSTM(units = 50, return_sequences = True))
model.add(Dropout(0.2))

model.add(LSTM(units = 50))
model.add(Dropout(0.25))

# Adding the output layer
model.add(Dense(units = 1))

# Compiling the RNN
# RMSprop is a recommended optimizer as per keras documentation
# check out https://keras.io/optimizers/ for more details
model.compile(optimizer = 'adam', loss = 'mean_squared_error')
```

Figure 4 - LSTM model

c. Ensemble Prediction

To combine the predictions from ARIMA and LSTM, I averaged their results. This ensemble approach helps to balance the strengths of both models.

```
# Ensure predictions are aligned with test_data index
arima_predictions = predictions_series
lstm_predictions = pd.Series(predicted_stock_price.flatten(), index=test_data.index)

# Compute ensemble predictions (simple average)
ensemble_predictions = (arima_predictions + lstm_predictions) / 2

# Create DataFrame
results_df = pd.DataFrame({
    'Date': test_data.index,
    'Real Stock Price': real_stock_price.flatten(),
    'ARIMA Prediction': arima_predictions,
    'LSTM Prediction': lstm_predictions,
    'Ensemble Prediction': ensemble_predictions
})
```

Figure 5 - Ensemble Model ARIMA + LSTM

2. Random Forest and GRU Model

a. Data Preparation

In this section, I prepared the data for the Random Forest and GRU models. The aim was to ensure that both models had appropriately processed inputs for training and testing.

```
# Download data
df = yf.download("AAPL", start="2015-01-01", end="2024-01-01")
company = 'Apple'
stock_data = df[['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume']]
# Create a new dataframe with relevant features
data = stock_data
data.index = pd.to_datetime(data.index)
train_data, test_data = data[0:int(len(data)*0.9)], data[int(len(data)*0.9):]
# Prepare the data
feature_columns = ['Open', 'High', 'Low', 'Volume']  # Example features
target_column = 'Close'
# Create feature and target arrays
features = data[feature_columns]
target = data[target_column]
# Split data into train and test sets
train_size = int(len(data) * 0.9)
train_features = features.iloc[:train_size]
test_features = features.iloc[train_size:]
train_target = target.iloc[:train_size]
test_target = target.iloc[train_size:]
```

Figure 6 - Data Preparation

b. Random Forest Model

I used the Random Forest Regressor to model the stock prices based on selected features. Grid Search was used to find the optimal hyperparameters for the Random Forest model.

```
# Define parameter grid for Grid Search
param_grid = {
    'n_estimators': [50, 100],
    'max_depth': [None, 10],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}
# Initialize RandomForestRegressor
rf_model = RandomForestRegressor(random_state=0)
# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid, cv=5, n_jobs=-1, verbose=2)
# Fit GridSearchCV
grid_search.fit(train_features, train_target)
# Get the best model
best_rf_model = grid_search.best_estimator_
# Make predictions
rf_predictions = best_rf_model.predict(test_features)
# Evaluate the model
mse = mean_squared_error(test_target, rf_predictions)
mae = mean_absolute_error(test_target, rf_predictions)
rmse = math.sqrt(mse)
print('Best parameters found: ', grid_search.best_params_)
print('MSE: ' + str(mse))
print('MAE: ' + str(mae))
print('RMSE: ' + str(rmse))
```

Figure 7- RF Model

c. GRU Model

```
model = Sequential()
model.add(GRU(units = 50, return_sequences = True, input_shape = (X_train.shape[1], 1)))
model.add(Dropout(0.20))
model.add(GRU(units = 50, return_sequences = True))
model.add(Dropout(0.25))
model.add(GRU(units = 50, return_sequences = True))
model.add(Dropout(0.2))
model.add(GRU(units = 50))
model.add(Dropout(0.25))
model.add(Dense(units = 1))
model.compile(optimizer = 'adam', loss = 'mean_squared_error')
model.fit(X_train, y_train, epochs = 10, batch_size = 32)
real_stock_price = test_data.iloc[:,0:1].values
combine = pd.concat((train_data['Close'], test_data['Close']), axis = 0)
test_inputs = combine[len(combine) - len(test_data) - timesteps:].values
test_inputs = test_inputs.reshape(-1,1)
test_inputs = scaler.transform(test_inputs)
```

Figure 8 - GRU Model

```
X_test = []
for i in range(timesteps, test_data.shape[0]+timesteps):
    X_test.append(test_inputs[i-timesteps:i, 0])
X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
predicted_stock_price = model.predict(X_test)
predicted_stock_price = scaler.inverse_transform(predicted_stock_price)
```

Figure 9 - Predict GRU Model

d. Ensemble Prediction

To combine the predictions from both Random Forest and GRU models, I computed the average of their predictions. This ensemble approach aims to leverage the strengths of both models.

```
# Ensure predictions are aligned with test_data index
ran_pre = rf_predictions
gru_predictions = pd.Series(predicted_stock_price.flatten(), index=test_data.index)

# Compute ensemble predictions (simple average)
ensemble_predictions = (ran_pre + gru_predictions) / 2

# Create DataFrame
results_df = pd.DataFrame({
    'Date': test_data.index,
    'Real Stock Price': real_stock_price.flatten(),
    'Random Forest': ran_pre,
    'GRU Prediction': gru_predictions,
    'Ensemble Prediction': ensemble_predictions
})
```

Figure 10 - Ensemble GRU+RF

Experiments with Different Configurations

Experiment 1: ARIMA + LSTM

Objective:

The objective of this experiment was to combine ARIMA and LSTM models to improve stock price prediction accuracy for Apple Inc. The aim was to leverage the strengths of both models and observe how combining them affects prediction performance.

Details:

 ARIMA **Model**:

- **Configuration**: ARIMA with order (5,1,0) was used based on prior experimentation.
- **Evaluation Metrics**:

```
MSE: 4.502555273693753
MAE: 1.6306260079012214
RMSE: 2.1219225418694605
```

Figure 11 - ARIMA Perfomance Metrics

 LSTM **Model**:

- **Configuration**: A sequential LSTM model with 4 GRU layers, dropout for regularization, and an output layer to predict the stock price.
- **Evaluation Metrics**:

```
MSE: 111.76694547405081
MAE: 9.33861323285208
RMSE: 10.571988718971035
```

Figure 12 - LSTM Perfomance Metrics

Results:

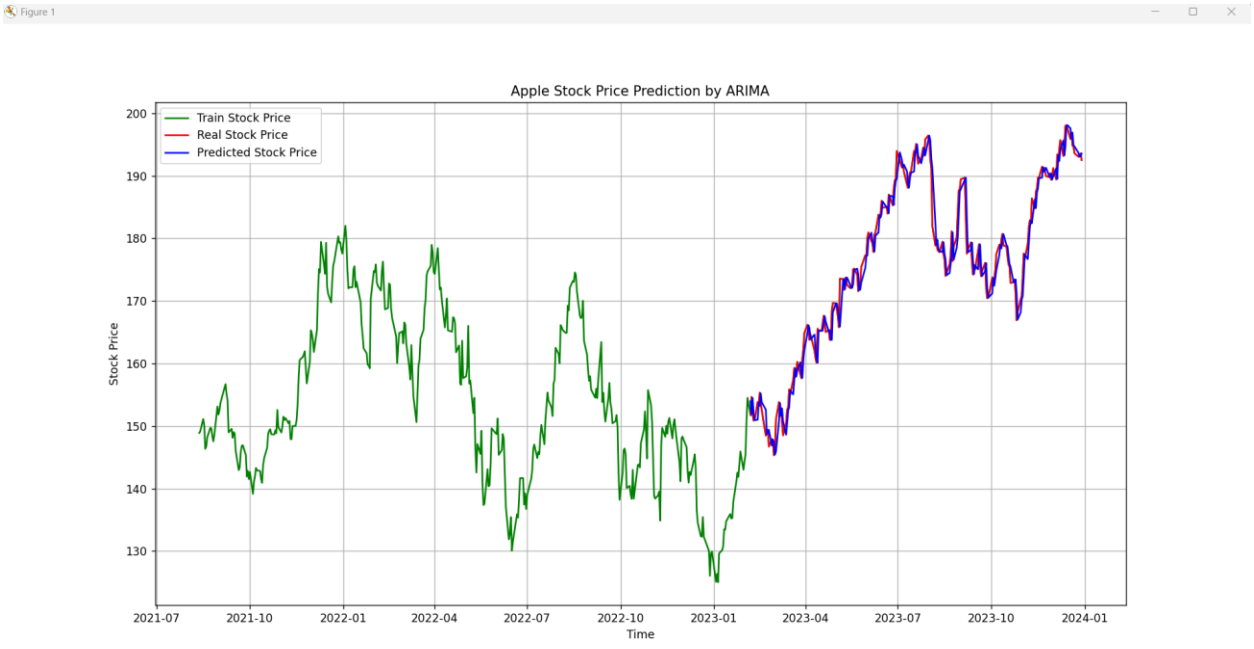

Figure 13 - ARIMA prediction

Figure 14 - LSTM prediction

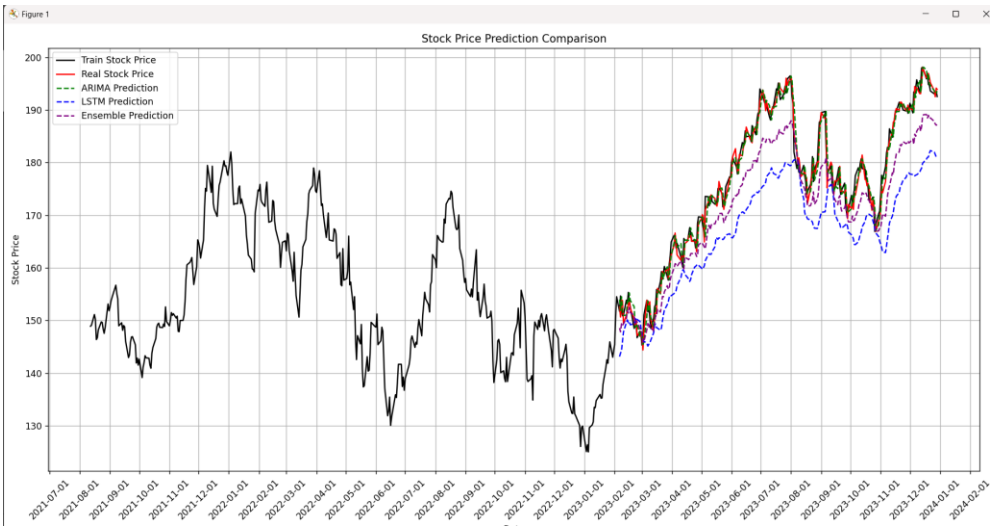

Figure 15 - Ensemble Prediction

	Date	Real Stock Price	ARIMA Prediction	LSTM Prediction	Ensemble Prediction
Date					
2023-02-06	2023-02-06	152.570007	153.951345	147.597275	150.774310
2023-02-07	2023-02-07	150.639999	151.875013	148.284790	150.079901
2023-02-08	2023-02-08	153.880005	154.493687	148.762619	151.628153
2023-02-09	2023-02-09	153.779999	152.069273	150.546692	151.307982
2023-02-10	2023-02-10	149.460007	150.926884	152.408264	151.667574
...	...	...	...	...	...
2023-12-22	2023-12-22	195.179993	194.685456	186.543869	190.614663
2023-12-26	2023-12-26	193.610001	193.639316	186.283615	189.961465
2023-12-27	2023-12-27	192.490005	193.069976	185.857620	189.463798
2023-12-28	2023-12-28	194.139999	193.146362	185.393250	189.269806
2023-12-29	2023-12-29	193.899994	193.564387	185.151749	189.358068

Figure 16 - Result DataFrame

Performance:

The ARIMA model demonstrated better performance on all evaluation metrics compared to the LSTM model. The ensemble approach, combining ARIMA and LSTM, aimed to leverage the strengths of both models but needs further tuning and experimentation to achieve better performance.

Experiment 2: RF + GRU

Objective:

The goal of this experiment was to evaluate the effectiveness of combining Random Forest (RF) and Gated Recurrent Units (GRU) models for stock price prediction. By integrating these two models, we aimed to enhance prediction accuracy and compare the results to the individual performances of RF and GRU.

Details

1. **GRU Model**:
    - **Configuration**: A GRU model with 4 layers, dropout for regularization, and an output layer to predict stock prices.
    - **Evaluation Metrics**:

```
MSE: 19.92572191504183
MAE: 3.7820202663606484
RMSE: 4.4638236877190645
```

Figure 17 - GRU perfomance metrics

2. **Random Forest (RF) Model**:
    - **Configuration**: Random Forest Regressor tuned with grid search for optimal hyperparameters.
    - **Evaluation Metrics**:

```
MSE: 51.74976780919269
MAE: 4.573334096496852
RMSE: 7.193731146574265
```

Figure 18 - RF perfomance metrics

Results:

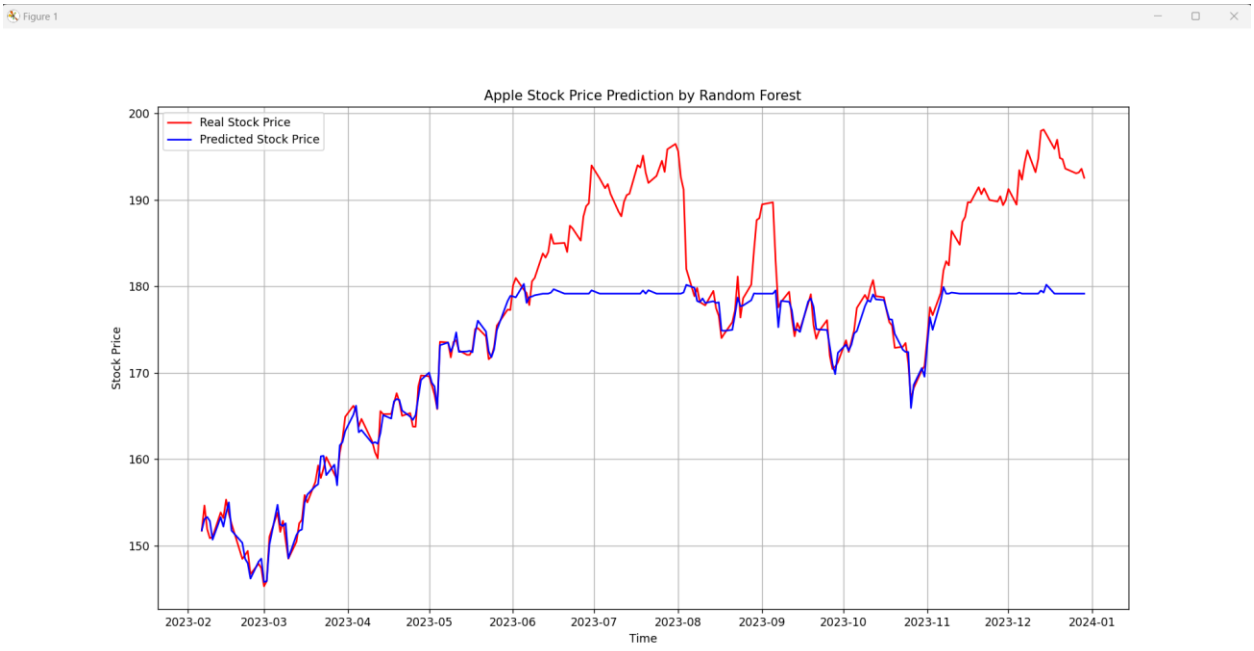

Figure 19 - RF prediction

Figure 20 - GRU prediction

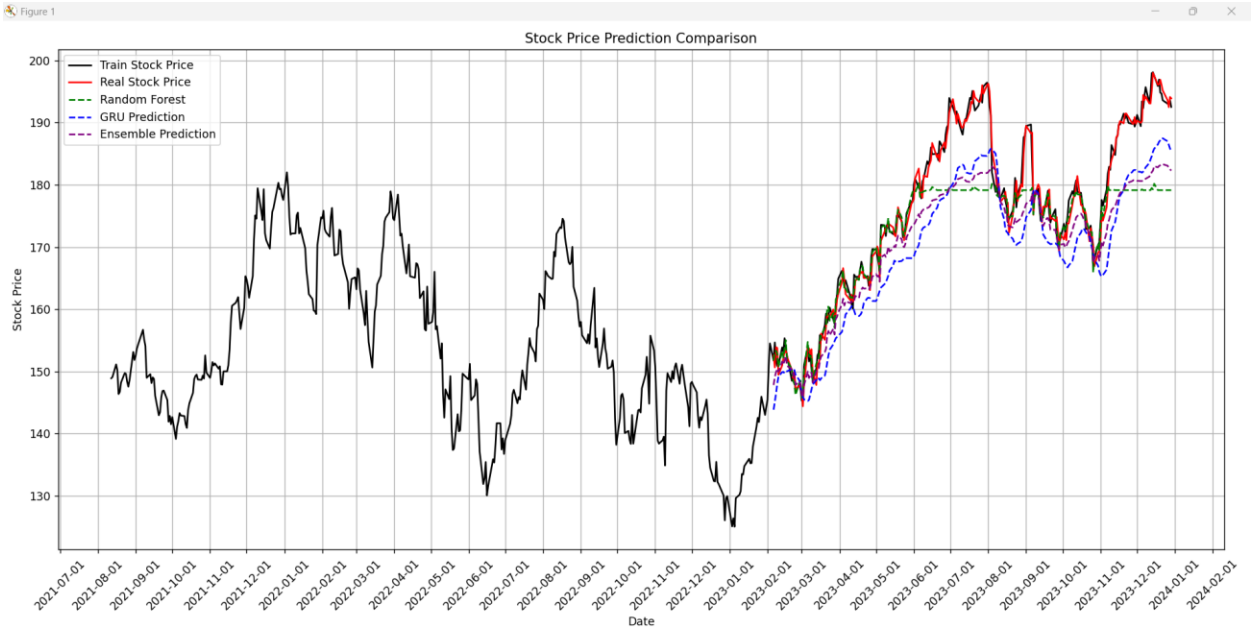

Figure 21 - Ensemble prediction

Date	Real Stock Price	Random Forest	GRU Prediction	Ensemble Prediction
Date				
2023-02-06 2023-02-06	152.570007	151.767003	146.500015	149.133509
2023-02-07 2023-02-07	150.639999	153.071198	147.849915	150.460556
2023-02-08 2023-02-08	153.880005	153.324199	149.253220	151.288709
2023-02-09 2023-02-09	153.779999	152.854399	150.914841	151.884620
2023-02-10 2023-02-10	149.460007	150.703800	152.113647	151.408724
... ...	...	...	...	...
2023-12-22 2023-12-22	195.179993	179.142597	192.407455	185.775026
2023-12-26 2023-12-26	193.610001	179.142597	191.895203	185.518900
2023-12-27 2023-12-27	192.490005	179.142597	191.274490	185.208544
2023-12-28 2023-12-28	194.139999	179.142597	190.668396	184.905497
2023-12-29 2023-12-29	193.899994	179.142597	190.245041	184.693819

Figure 22 - Result Dataframe (GRU + RF)

Performance:

 GRU **Model** demonstrated superior performance with lower MSE, MAE, and RMSE compared to the Random Forest model. This indicates that the GRU model was more effective in capturing the temporal dependencies in the stock price data.

 Random **Forest Model** had higher MSE, MAE, and RMSE, suggesting that while it captured non-linear patterns effectively, it did not perform as well as the GRU in this context.

Conclusion

In our experiments evaluating ensemble modeling approaches for stock price prediction, we compared ARIMA combined with LSTM and Random Forest (RF) combined with GRU. The ARIMA + LSTM combination demonstrated that ARIMA was more effective individually, providing lower error metrics compared to LSTM. This suggests that while LSTM has potential, it may need further tuning to match ARIMA's performance in this context. On the other hand, the RF + GRU combination highlighted the GRU model's superior capability in capturing temporal patterns and sequential dependencies, outperforming the Random Forest model. The GRU model's lower error metrics indicate its strength in time-series forecasting, whereas Random Forest, though useful in capturing non-linear relationships, showed less accuracy here. These results emphasize the importance of model selection and configuration, suggesting that while ensembles have potential, each model's individual strengths should be carefully considered to enhance prediction accuracy. Further refinement and experimentation with these ensembles could yield even more precise stock price forecasts.

References

Baraban, B. (2023). *ARIMA + LSTM: Combining Time Series Models for Forecasting*. Kaggle. Retrieved from https://www.kaggle.com/code/bogdanbaraban/ar-arima-lstm#Plotting-autocorrelation

Razamh. (2023). *CAC 40 Stock Price Forecast with ARIMA & LSTM*. Kaggle. Retrieved from https://www.kaggle.com/code/razamh/1cac-40-stock-price-forecast-with-arima-lstm#Part-3.-CAC-40-Stock-Price-Forecast-with-LSTM-[optional]