Project B

Task 6 – Machine Learning 3

Tasks:

1. Develop an ensemble modelling approach consisting of at least two models ARIMA (or SARIMA) and our existing DL model. (Starting with the LSTM one).

In developing our ensemble model, we included an ARIMA model alongside our existing LSTM model. Here's how we integrated ARIMA into our code:

Preparing the dataset for ARIMA:

```
# Prepare data for ARIMA/SARIMA model
train_series = train_data['Adj Close']
test_series = test_data['Adj Close']

# If scaled, inverse transform for ARIMA/SARIMA
if SCALE:
    train_series_values = scalers["Adj
Close"].inverse_transform(train_series.values.reshape(-1, 1)).flatten()
    train_series = pd.Series(train_series_values, index=train_data.index)
    test_series_values = scalers["Adj
Close"].inverse_transform(test_series.values.reshape(-1, 1)).flatten()
    test_series = pd.Series(test_series_values, index=test_data.index)
```

Making the series stationary:

```
def make_stationary(series):
    result = adfuller(series)
    if result[1] > 0.05:
        print("Series is not stationary. Differencing...")
        series_diff = series.diff().dropna()
        return series_diff
    else:
        print("Series is stationary.")
        return series
```

Fitting the ARIMA model:

```
# Fit the SARIMA model
model_order = pm.auto_arima(train_series_stationary,
                            seasonal=False,
                            trace=True,
                            error_action='ignore',
                            suppress_warnings=True)
```

Forecasting with ARIMA:

```
# Forecast
n_periods = len(test_series_stationary)
arima_forecast_diff = model_order.predict(n_periods=n_periods)

# Convert to NumPy array
arima_forecast_diff = np.array(arima_forecast_diff)

# Invert differencing to restore original scale
last_train_value = train_series.iloc[-1]
arima_forecast = np.r_[last_train_value, arima_forecast_diff].cumsum()
```

Combining ARIMA predictions with LSTM:

```
# Generate final DataFrame with predictions
final_df = get_final_df(model, X_test, y_test, test_data, scalers, LOOKUP_STEP,
SCALE, arima_forecast=arima_forecast)
# Compute MAE for ARIMA
arima_mae = mean_absolute_error(final_df[f'true_adjclose_{LOOKUP_STEP}'],
final_df[f'arima_pred_{LOOKUP_STEP}'])
```

Despite incorporating an ARIMA model alongside my existing LSTM model to form an ensemble, I noticed that the ARIMA model performed very poorly. I believe the main reasons for this is that ARIMA models are fundamentally designed to handle linear and stationary time series data. Stock prices, however, are volatile and influenced by a multitude of external factors.

ARIMA models also rely heavily on past values and assume that future values are a linear function of past observations. On the other hand, LSTM models are specifically designed to capture long-term dependencies and can handle non-linear patterns more effectively. They are better suited to model the complex relationships in financial data.

2. Experiment with different ensemble models (e.g., ARIMA/SARIMA/Random Forest/LSTM/RNN/GRU, etc.) and with different hyperparameter configurations.

I chose to implement a Random Forest and RNN model along with the original LSTM to create an ensemble prediction.

Random Forest Model:

The Random Forest model is an ensemble learning method that builds multiple decision trees and averages their predictions for more accurate results. I incorporated the Random Forest regressor from scikit-learn to predict future stock prices.

Here's how I implemented it:

```
# Reshape X_train and X_test for Random Forest
X_train_rf = X_train.reshape((X_train.shape[0], X_train.shape[1]*X_train.shape[2]))
X_test_rf = X_test.reshape((X_test.shape[0], X_test.shape[1]*X_test.shape[2]))

# Initialize Random Forest Regressor
rf = RandomForestRegressor(n_estimators=100, random_state=42)

# Fit the model
rf.fit(X_train_rf, y_train)

# Make predictions
y_pred_rf = rf.predict(X_test_rf)

# Inverse transform predictions if scaled
if SCALE:
    y_pred_rf = scalers["Adj Close"].inverse_transform(y_pred_rf.reshape(-
1,1)).flatten()

# Add Random Forest predictions to final_df
final_df['rf_pred'] = y_pred_rf

# Compute MAE for Random Forest predictions
rf_mae = mean_absolute_error(final_df[f'true_adjclose_{LOOKUP_STEP}'],
final_df['rf_pred'])
print(f"Random Forest MAE: {rf_mae}")
```

Hyperparameters:

- N_estimators=100: Number of trees in the forest. A higher number can improve performance but increases computation time.
- Random_state=42: Allows for reproducibility of results.

After reshaping the input data to fit the Random Forest's expected 2D format, I trained the model using training set (X_train_rf, y_train). Predictions are then made on the test set (X_test_rf), and the Mean Absolute Error (MAE) was calculated to assess performance.

Recurrent Neural Network (RNN) Model:

RNNs are well-suited for sequential data like stock prices due to their ability to capture temporal dependencies.

Implementation:
I used this method to define the model architecture (the same method as was used for LSTM)

```
def create_model(sequence_length, n_features, units=256, cell=LSTM, n_layers=2,
dropout=0.3,
                 loss="mean_absolute_error", optimizer="rmsprop",
bidirectional=False):
```

```
    model = Sequential()

    # Add Input layer
    model.add(Input(shape=(sequence_length, n_features)))

    for i in range(n_layers):
        if i == 0:
            # first layer
            if bidirectional:
                model.add(Bidirectional(cell(units, return_sequences=True)))
            else:
                model.add(cell(units, return_sequences=True))
        elif i == n_layers - 1:
            # last layer
            if bidirectional:
                model.add(Bidirectional(cell(units, return_sequences=False)))
            else:
                model.add(cell(units, return_sequences=False))
        else:
            # hidden layers
            if bidirectional:
                model.add(Bidirectional(cell(units, return_sequences=True)))
            else:
                model.add(cell(units, return_sequences=True))

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

    # Output layer
    model.add(Dense(1, activation="linear"))

    # Compile the model
    model.compile(loss=loss, metrics=["mean_absolute_error"], optimizer=optimizer)

    return model
```

Here is where that method was used:

```
rnn_model = create_model(N_STEPS, len(FEATURE_COLUMN), loss=LOSS, units=UNITS,
cell=SimpleRNN, n_layers=N_LAYERS,
                    dropout=DROPOUT, optimizer=OPTIMIZER,
bidirectional=BIDIRECTIONAL)
```

And here is the where the RNN model is trained:

```
rnn_model.fit(X_train, y_train,
                    batch_size=BATCH_SIZE,
                    epochs=EPOCHS,
                    validation_data=(X_test, y_test),
```

```
                    callbacks=[checkpointer_rnn, tensorboard_rnn],
                    verbose=1)
```

This is all the same as previous weeks.

Ensemble Prediction:

After training I created a 3 way ensemble prediction by averaging the predictions from each LSTM, RNN, and Random Forest models.

```
# Create ensemble prediction
final_df['ensemble_pred'] = (final_df[f'lstm_adjclose_{LOOKUP_STEP}'] +
final_df[f'rnn_adjclose_{LOOKUP_STEP}'] + final_df['rf_pred']) / 3
```

Then the MAE was calculated for this ensemble:

```
# Compute MAE for ensemble predictions
ensemble_mae = mean_absolute_error(final_df[f'true_adjclose_{LOOKUP_STEP}'],
final_df['ensemble_pred'])
print(f"Ensemble MAE: {ensemble_mae}")
```

Results:
Below will be a series of trainings and evaluations with varying hyperparameters for LSTM, RNN, and Random Forest. All evaluation will have:

- N_STEPS = 14
- LOOKUP_STEP = 7
- TEST_SIZE = 0.20
- NAN_HANDLER = “drop”
- SCALE = TRUE
- LOSS = MAE
- OPTIMIZER = Adam
- As well as the same start and end date for the same company (Amazon).

Attempt 1:

LSTM and RNN:

- N_Layers = 2
- Units = 256
- Dropout = 0.4
- Bidirectional = False
- Batch Size = 64
- Epochs = 50

Random Forest:

- N_ESTIMATORS = 100
- RANDOM_STATE = 42

Stats:

LSTM MAE: 2.4491474274591307

RNN MAE: 3.0879108753441558

Random Forest MAE: 0.18953565743748343

Ensemble MAE: 1.7053124662390717

Accuracy Score: 0.8828451882845189

Total Buy Profit: 444.4021034138782

Total Sell Profit: 993.8826694664234

Total Profit: 1438.2847728803017

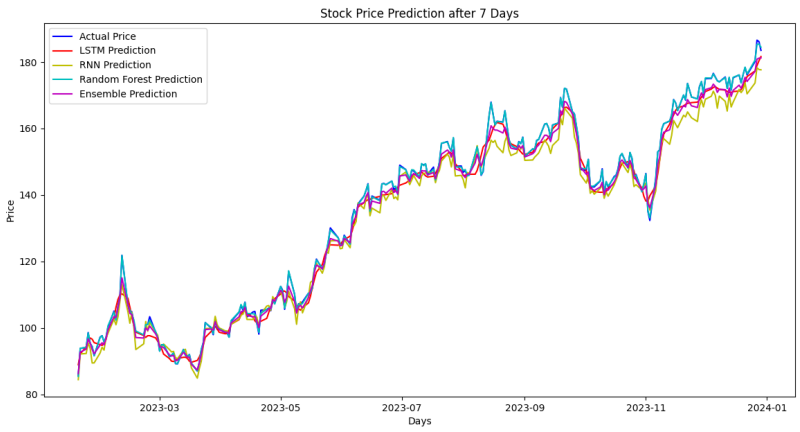
Profit Per Trade: 6.017927920001262

Predicted future price after 7 days using Ensemble: 181.25$

Predicted future price after 7 days using Random Forest: 184.46$

Predicted future price after 7 days using LSTM: 181.64$

Predicted future price after 7 days using RNN: 177.66$



Attempt 2:

LSTM and RNN:

- N_Layers = 4
- Units = 256
- Dropout = 0.4
- Bidirectional = True
- Batch Size = 64
- Epochs = 100

Random Forest:

- N_ESTIMATORS = 100
- RANDOM_STATE = 42

LSTM MAE: 2.853399380274252

RNN MAE: 4.48345768229524

Random Forest MAE: 0.18953565743748343

Ensemble MAE: 2.0319787532434654

Accuracy Score: 0.8619246861924686

Total Buy Profit: 431.7448909894931

Total Sell Profit: 981.2254570420382

Total Profit: 1412.9703480315313

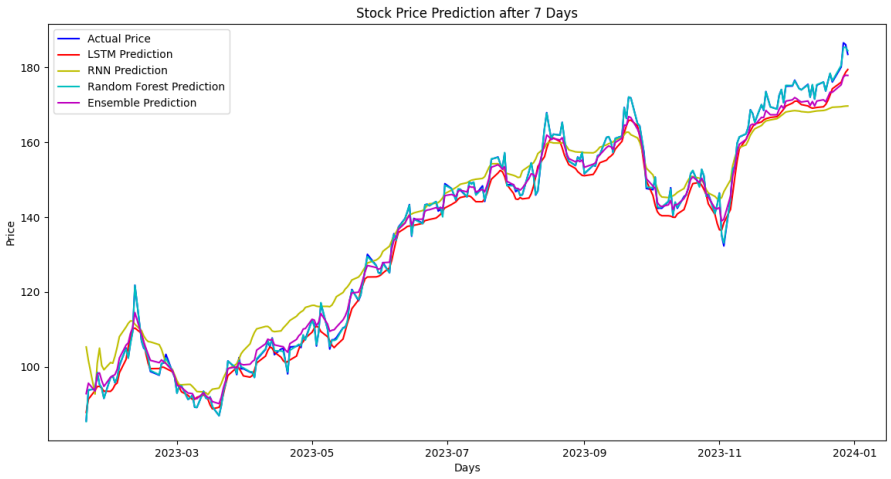
Profit Per Trade: 5.912009824399712

Predicted future price after 7 days using Ensemble: 177.87$

Predicted future price after 7 days using Random Forest: 184.46$

Predicted future price after 7 days using LSTM: 179.46$

Predicted future price after 7 days using RNN: 169.71$



Attempt 3:

LSTM and RNN:

- Units = 256
- Dropout = 0.4
- Batch Size = 64

LSTM:

- N_Layers = 4
- Bidirectional = True
- Epochs = 100

RNN:

- N_Layers = 2
- Bidirectional = False
- Epochs = 50

Random Forest:

- N_ESTIMATORS = 100
- RANDOM_STATE = 42

LSTM MAE: 2.0643937298267767

RNN MAE: 2.5496496676345792

Random Forest MAE: 0.18953565743748343

Ensemble MAE: 1.1798423626865018

Accuracy Score: 0.9037656903765691

Total Buy Profit: 455.7385202608748

Total Sell Profit: 1005.2190863134199

Total Profit: 1460.9576065742947

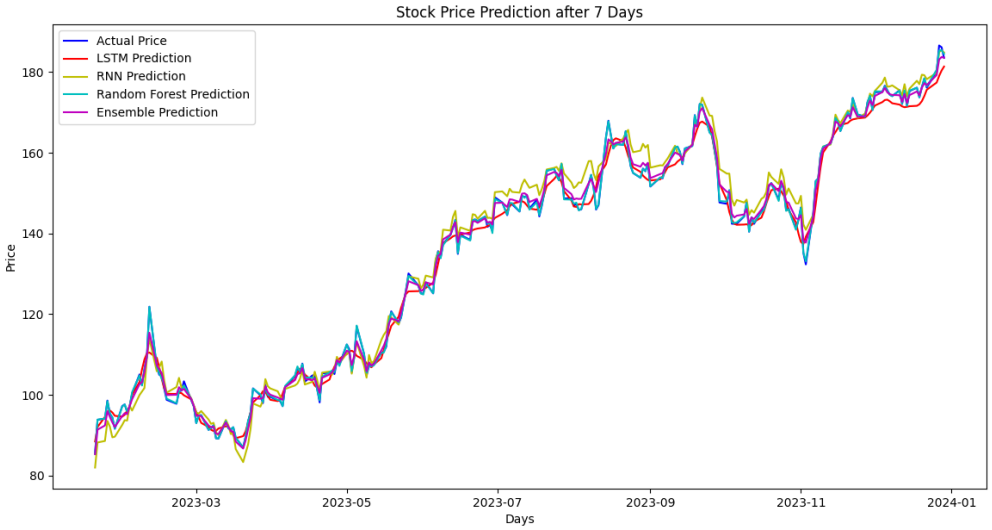
Profit Per Trade: 6.11279333294684

Predicted future price after 7 days using Ensemble: 183.54$

Predicted future price after 7 days using Random Forest: 184.46$

Predicted future price after 7 days using LSTM: 181.35$

Predicted future price after 7 days using RNN: 184.82$



Attempt 4:

LSTM and RNN:

- Units = 256
- Dropout = 0.4
- Batch Size = 12

LSTM:

- N_Layers = 6
- Bidirectional = True
- Epochs = 50

RNN:

- N_Layers = 2
- Bidirectional = False
- Epochs = 50

Random Forest:

- N_ESTIMATORS = 100
- RANDOM_STATE = 42

LSTM MAE: 2.1995593256227433

RNN MAE: 3.181067571119698

Random Forest MAE: 0.18953565743748343

Ensemble MAE: 1.4305076180008587

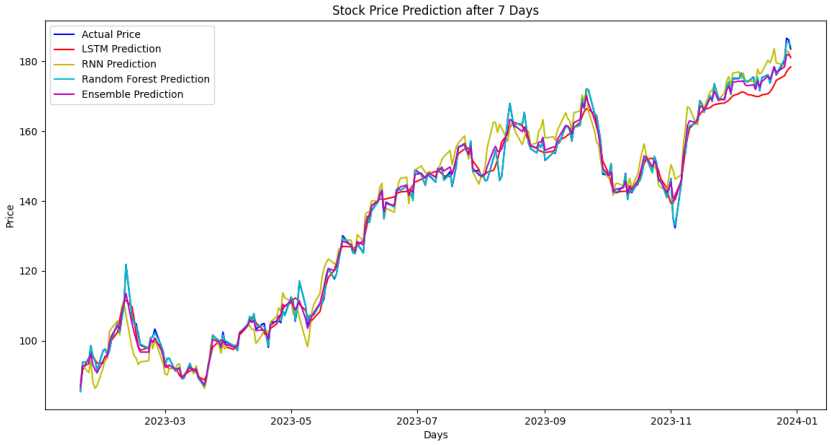
Accuracy Score: 0.9037656903765691

Total Buy Profit: 458.5687108132008

Total Sell Profit: 1008.049276865746

Total Profit: 1466.617987678947

Profit Per Trade: 6.136476935895176



Attempt 5:

LSTM and RNN:

- Units = 256
- Dropout = 0.4
- Batch Size = 12
- N_Layers = 6
- Bidirectional = True
- Epochs = 150

Random Forest:

- N_ESTIMATORS = 100
- RANDOM_STATE = 42