Task 6 - Machine Learning 3

Overview

Having completed Task 5 and advanced the project codebase to version v0.5, I was assigned the task of developing an ensemble approach for combining multiple machine learning and statistical analysis methods for improving the prediction quality of the existing version of the project, which is the implementation of a single model of Long Short-Term Memory (LSTM).

As introduced in the task brief, the basic idea supporting ensemble methods is to combine multiple learning algorithms to obtain better predictive performance that could be obtained from any of the single learning algorithms alone. In this task, I would have to develop an ensemble modelling approach consisting of at least two models, autoregressive integrated moving average (ARIMA) or seasonal ARIMA (SARIMA), and the existing DL model using LSTM, as well as experiment with different ensemble models (e.g., ARIMA/SARIMA/Random Forest/LSTM/RNN/GRU) and with different hyperparameters configurations.

SARIMA

SARIMA is a forecasting method for univariate time series data that works well with data having a seasonal component, supporting both autoregressive and moving average elements, while the integrated element of differencing allowing the method to support non-stationary time series data.

The seasonality of a time series refers to recurring and regular patterns at specific regular intervals less than a year (i.e., weekly, monthly, or quarterly). It may be caused by out-of-scope factors belonging to the natural and societal world (e.g., weather, vacation, holidays). As stock data is recorded on a daily basis (excepting non-working days of Saturdays and Sundays), and is strongly influenced by the events in the economics and politics world, it has a great likelihood to express seasonality.

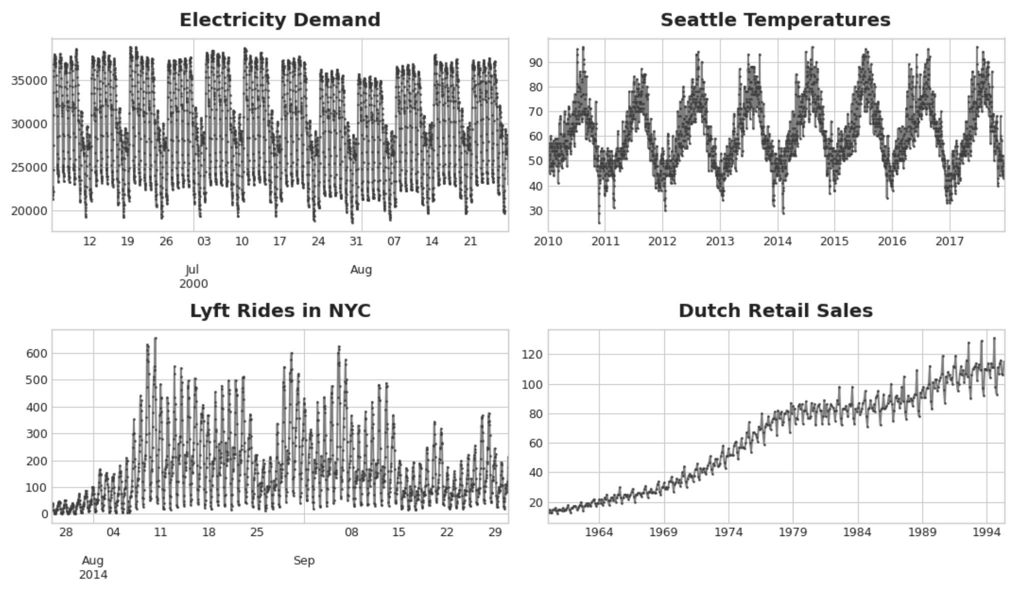
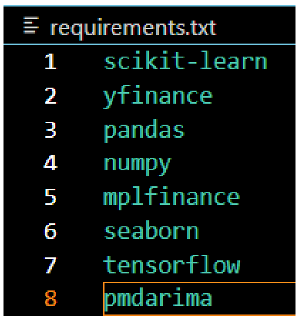


Figure 1: Example of seasonal time series data.

To kick off implementations of SARIMA, I added the `pmdarima` package to the `requirements.txt` file of the project, and create a new virtual environment. Here is the point where I encountered an error of inability to create a new virtual environment, because of the incompatibility of the `pmdarima` package with Python 3.13. As a result, I downgrade Python by installing Python 3.12, and created a new virtual environment using the earlier version. The process turned out to be successful.



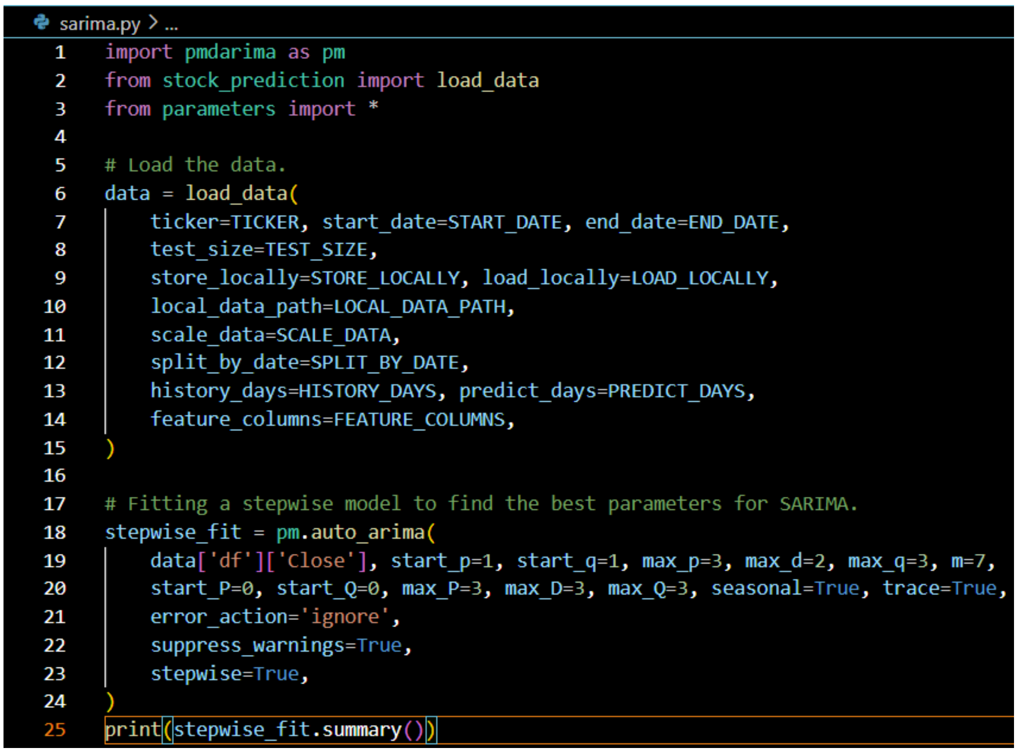
```
requirements.txt
scikit-learn
yfinance
pandas
numpy
mplfinance
seaborn
tensorflow
pmdarima
```

Figure 2: All required packages of the project, up until this point of implementing SARIMA.

As mentioned in the attached blog post, there are a total of 7 parameters requiring configuration for the SARIMA model:

- `p`: Trend autoregression order.
- `d`: Trend difference order.
- `q`: Trend moving average order.
- `P`: Seasonal autoregressive order.
- `D`: Seasonal difference order.
- `Q`: Seasonal moving average order.
- `m`: The number of time steps for a single seasonal period.

The optimal values of each parameter could be found by doing a grid search fitting a pre-specified range of parameter values. The set of parameter values with the lowest AIC (aka. Akaike information criterion), which is an estimator of prediction error, would be chosen. The following code snippet load the data needed and perform the aforementioned process, after having imported the necessary modules and functions, and followed the blog post's procedure.



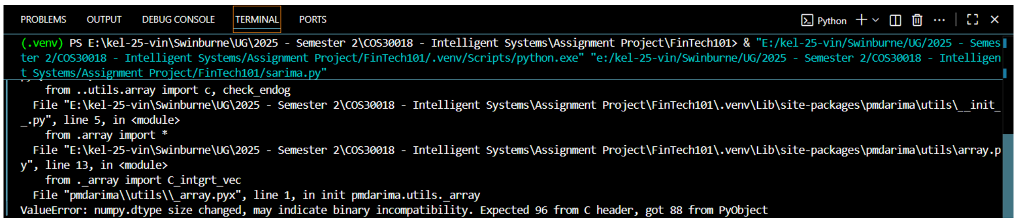
```python
sarima.py > ...
import pmdarima as pm
from stock_prediction import load_data
from parameters import *

# Load the data.
data = load_data(
    ticker=TICKER, start_date=START_DATE, end_date=END_DATE,
    test_size=TEST_SIZE,
    store_locally=STORE_LOCALLY, load_locally=LOAD_LOCALLY,
    local_data_path=LOCAL_DATA_PATH,
    scale_data=SCALE_DATA,
    split_by_date=SPLIT_BY_DATE,
    history_days=HISTORY_DAYS, predict_days=PREDICT_DAYS,
    feature_columns=FEATURE_COLUMNS,
)

# Fitting a stepwise model to find the best parameters for SARIMA.
stepwise_fit = pm.auto_arima(
    data['df']['Close'], start_p=1, start_q=1, max_p=3, max_d=2, max_q=3, m=7,
    start_P=0, start_Q=0, max_P=3, max_D=3, max_Q=3, seasonal=True, trace=True,
    error_action='ignore',
    suppress_warnings=True,
    stepwise=True,
)
print(stepwise_fit.summary())
```

Figure 3: The code to load the data, and perform grid search for the optimal set of parameters for the SARIMA model.

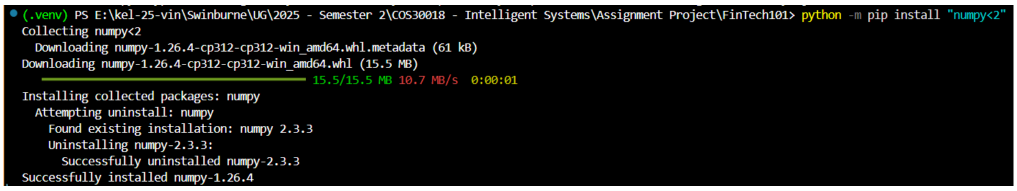
When trying to run this, I encountered a `ValueError` indicating a possible binary incompatibility of the `pmdarima` module towards the `numpy` one.



```
(.venv) PS E:\kel-25-vin\Swinburne\UG\2025 - Semester 2\COS30018 - Intelligent Systems\Assignment Project\FinTech101> & "E:/kel-25-vin/Swinburne/UG/2025 - Semes
ter 2/COS30018 - Intelligent Systems/Assignment Project/FinTech101/.venv/Scripts/python.exe" "e:/kel-25-vin/Swinburne/UG/2025 - Semester 2/COS30018 - Intelligen
t Systems/Assignment Project/FinTech101/sarima.py"
    from ..utils.array import c, check_endog
  File "E:\kel-25-vin\Swinburne\UG\2025 - Semester 2\COS30018 - Intelligent Systems\Assignment Project\FinTech101\.venv\Lib\site-packages\pmdarima\utils\__init_
_.py", line 5, in <module>
    from .array import *
  File "E:\kel-25-vin\Swinburne\UG\2025 - Semester 2\COS30018 - Intelligent Systems\Assignment Project\FinTech101\.venv\Lib\site-packages\pmdarima\utils\array.p
y", line 13, in <module>
    from ._array import C_intgrt_vec
  File "pmdarima\\utils\\_array.pyx", line 1, in init pmdarima.utils._array
ValueError: numpy.dtype size changed, may indicate binary incompatibility. Expected 96 from C header, got 88 from PyObject
```

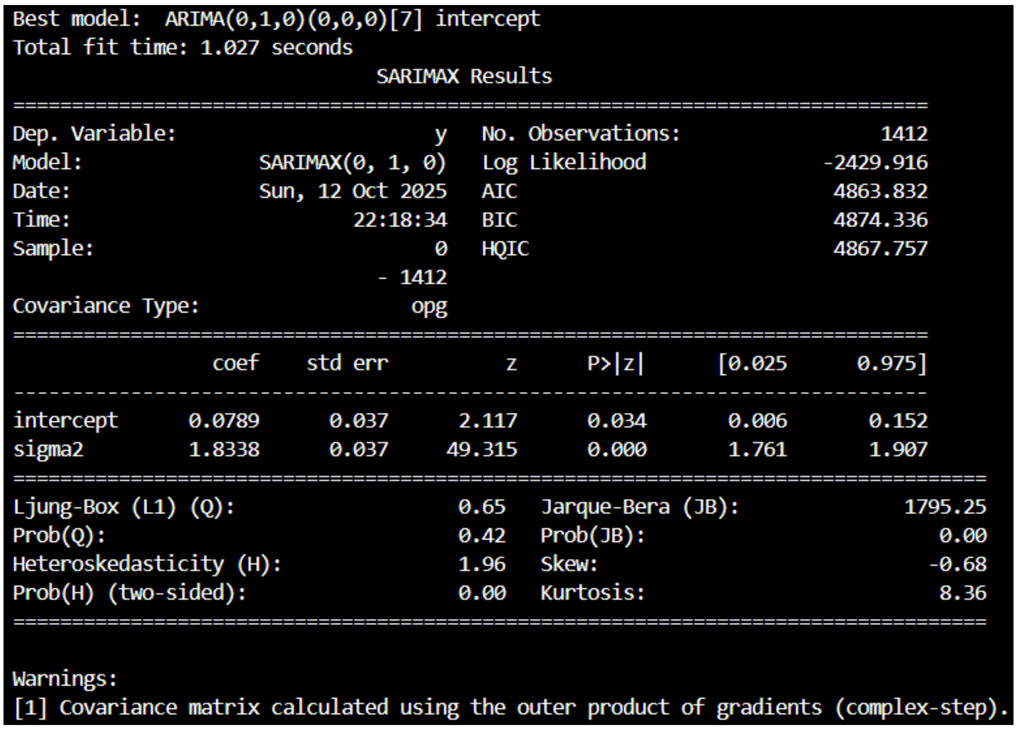
Figure 4: The `ValueError` suggesting incompatibility of `pmdarima` with `numpy`.

After doing some research on the Internet, I found out that the issue here is somewhat similar to the issue making the virtual environment unable to be created. It's the fact that `pmdarima` is only compatible with `numpy` version 1.x, because the model was compiled using that version, not the version I was using, `numpy` 2.x. Consequently, I had to downgrade `numpy` to 1.x by running a simple command, and the code worked perfectly.



```
(.venv) PS E:\kel-25-vin\Swinburne\UG\2025 - Semester 2\COS30018 - Intelligent Systems\Assignment Project\FinTech101> python -m pip install "numpy<2"
Collecting numpy<2
  Downloading numpy-1.26.4-cp312-cp312-win_amd64.whl.metadata (61 kB)
Downloading numpy-1.26.4-cp312-cp312-win_amd64.whl (15.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 15.5/15.5 MB 10.7 MB/s  0:00:01
Installing collected packages: numpy
  Attempting uninstall: numpy
    Found existing installation: numpy 2.3.3
    Uninstalling numpy-2.3.3:
      Successfully uninstalled numpy-2.3.3
Successfully installed numpy-1.26.4
```

Figure 5: The command used to downgrade `numpy`.

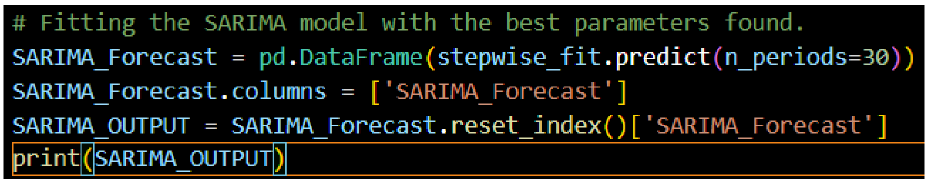


```
Best model:  ARIMA(0,1,0)(0,0,0)[7] intercept
Total fit time: 1.027 seconds
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                 1412
Model:               SARIMAX(0, 1, 0)   Log Likelihood               -2429.916
Date:                Sun, 12 Oct 2025   AIC                           4863.832
Time:                        22:18:34   BIC                           4874.336
Sample:                             0   HQIC                          4867.757
                                - 1412
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept      0.0789      0.037      2.117      0.034       0.006       0.152
sigma2         1.8338      0.037     49.315      0.000       1.761       1.907
===================================================================================
Ljung-Box (L1) (Q):                   0.65   Jarque-Bera (JB):              1795.25
Prob(Q):                              0.42   Prob(JB):                         0.00
Heteroskedasticity (H):               1.96   Skew:                            -0.68
Prob(H) (two-sided):                  0.00   Kurtosis:                         8.36
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

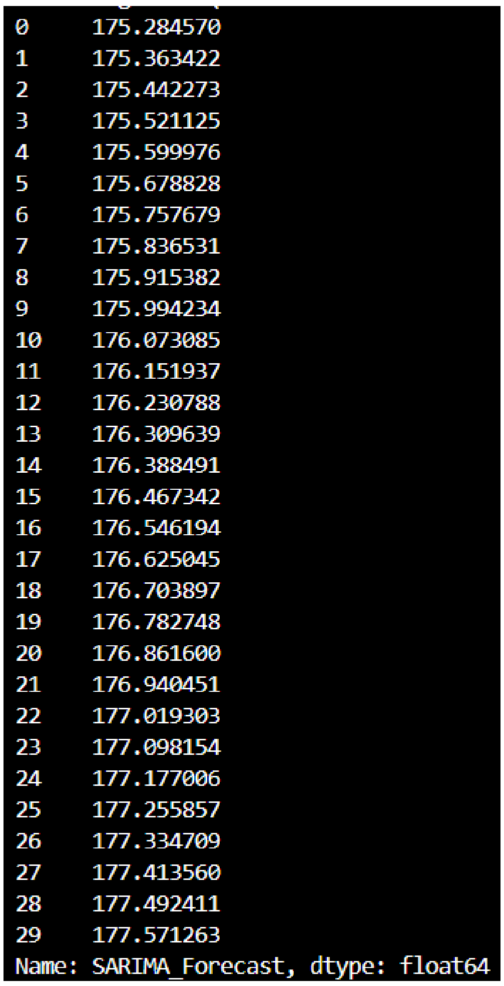
Figure 6: The result of the stepwise grid search process.

With that done, I proceeded to make predictions with the result model. The predictions are made for the following 30 days of the input data.



```python
# Fitting the SARIMA model with the best parameters found.
SARIMA_Forecast = pd.DataFrame(stepwise_fit.predict(n_periods=30))
SARIMA_Forecast.columns = ['SARIMA_Forecast']
SARIMA_OUTPUT = SARIMA_Forecast.reset_index()['SARIMA_Forecast']
print(SARIMA_OUTPUT)
```

Figure 7: The code to make prediction using the output SARIMAX model.

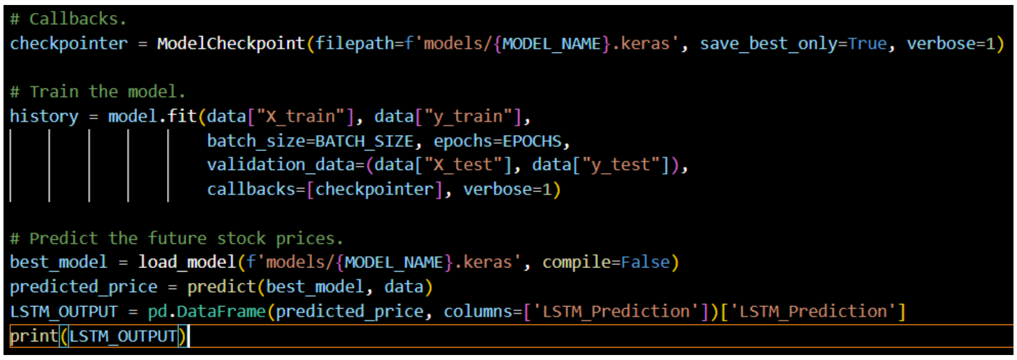


```
0     175.284570
1     175.363422
2     175.442273
3     175.521125
4     175.599976
5     175.678828
6     175.757679
7     175.836531
8     175.915382
9     175.994234
10    176.073085
11    176.151937
12    176.230788
13    176.309639
14    176.388491
15    176.467342
16    176.546194
17    176.625045
18    176.703897
19    176.782748
20    176.861600
21    176.940451
22    177.019303
23    177.098154
24    177.177006
25    177.255857
26    177.334709
27    177.413560
28    177.492411
29    177.571263
Name: SARIMA_Forecast, dtype: float64
```

Figure 8: The predictions made by the SARIMAX model.

Ensemble with LSTM

First, I made some modifications with the LSTM model’s training process, by adding a `ModelCheckpointer` callback which allow me to retrieve and save the best model over the whole period of training. After training, the system would load the best model saved, and make predictions on the testing data. I also made a change which would help display the final predictions in the format of a DataFrame.



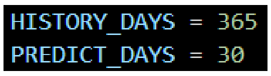
```python
# Callbacks.
checkpointer = ModelCheckpoint(filepath=f'models/{MODEL_NAME}.keras', save_best_only=True, verbose=1)

# Train the model.
history = model.fit(data["X_train"], data["y_train"],
                    batch_size=BATCH_SIZE, epochs=EPOCHS,
                    validation_data=(data["X_test"], data["y_test"]),
                    callbacks=[checkpointer], verbose=1)

# Predict the future stock prices.
best_model = load_model(f'models/{MODEL_NAME}.keras', compile=False)
predicted_price = predict(best_model, data)
LSTM_OUTPUT = pd.DataFrame(predicted_price, columns=['LSTM_Prediction'])['LSTM_Prediction']
print(LSTM_OUTPUT)
```

Figure 9: The modified version of the `train.py` module, which now includes a checkpoint callback for model persistence.

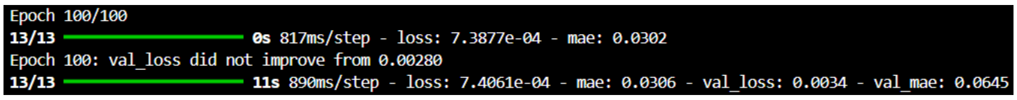
Then, I changed the parameter values of `HISTORY_DAYS` and `PREDICT_DAYS` to 365 and 30 respectively. This change indicates that the model now would use data from a year-long window to predict the price of roughly a month after.



```python
HISTORY_DAYS = 365
PREDICT_DAYS = 30
```

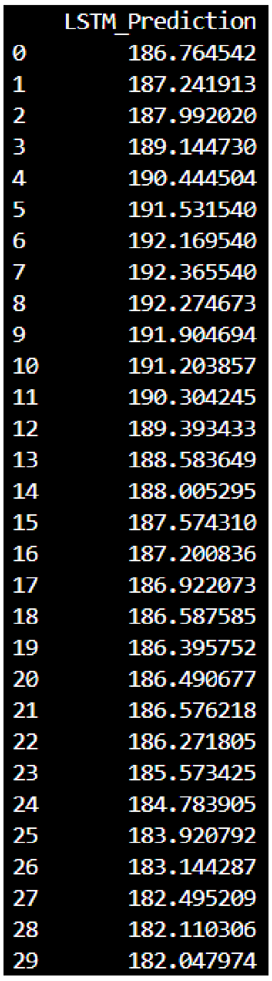
Figure 10: New parameter configurations allowing the model to use a year of price data to predict the price of approximately a month later.

With these done, I proceeded to train the LSTM model for 100 epochs, and make predictions with the best model.



```
Epoch 100/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 817ms/step - loss: 7.3877e-04 - mae: 0.0302
Epoch 100: val_loss did not improve from 0.00280
13/13 ━━━━━━━━━━━━━━━━━━━━ 11s 890ms/step - loss: 7.4061e-04 - mae: 0.0306 - val_loss: 0.0034 - val_mae: 0.0645
```

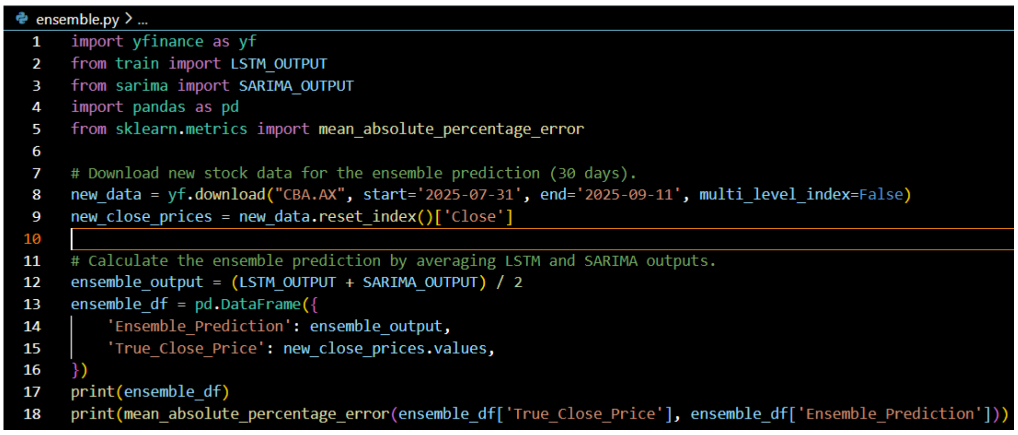
Figure 11: Training done for the LSTM model.



```
    LSTM_Prediction
0        186.764542
1        187.241913
2        187.992020
3        189.144730
4        190.444504
5        191.531540
6        192.169540
7        192.365540
8        192.274673
9        191.904694
10       191.203857
11       190.304245
12       189.393433
13       188.583649
14       188.005295
15       187.574310
16       187.200836
17       186.922073
18       186.587585
19       186.395752
20       186.490677
21       186.576218
22       186.271805
23       185.573425
24       184.783905
25       183.920792
26       183.144287
27       182.495209
28       182.110306
29       182.047974
```

Figure 12: Predictions made by the best saved model.

Having gathered the predictions of LSTM and SARIMA models, I proceeded to compute and evaluate the ensemble predictions. This code snippet below from the newly created `ensemble.py` module downloads the new data for 30 following days, calculate the ensemble prediction values, output the DataFrame constructed by the two Series of predicted and true prices respectively, and eventually evaluate the predictions using the mean absolute percentage error function.



ensemble.py > ...

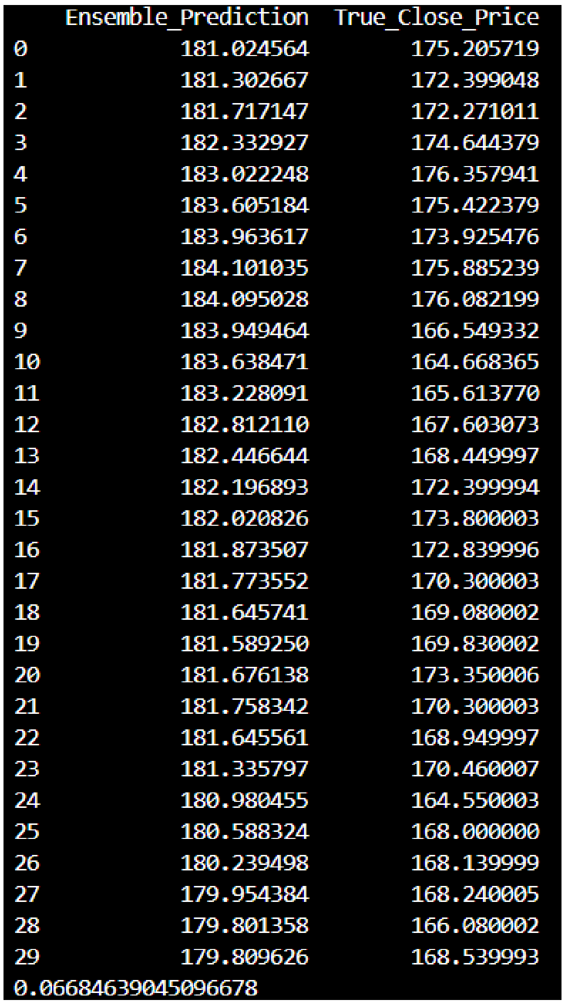
```python
import yfinance as yf
from train import LSTM_OUTPUT
from sarima import SARIMA_OUTPUT
import pandas as pd
from sklearn.metrics import mean_absolute_percentage_error

# Download new stock data for the ensemble prediction (30 days).
new_data = yf.download("CBA.AX", start='2025-07-31', end='2025-09-11', multi_level_index=False)
new_close_prices = new_data.reset_index()['Close']

# Calculate the ensemble prediction by averaging LSTM and SARIMA outputs.
ensemble_output = (LSTM_OUTPUT + SARIMA_OUTPUT) / 2
ensemble_df = pd.DataFrame({
    'Ensemble_Prediction': ensemble_output,
    'True_Close_Price': new_close_prices.values,
})
print(ensemble_df)
print(mean_absolute_percentage_error(ensemble_df['True_Close_Price'], ensemble_df['Ensemble_Prediction']))
```

Figure 13: Calculating and evaluating the ensemble predictions.

The recorded MAPE value was **6.68%**, showing better predictions than using only the existing LSTM model.



```
    Ensemble_Prediction  True_Close_Price
0            181.024564        175.205719
1            181.302667        172.399048
2            181.717147        172.271011
3            182.332927        174.644379
4            183.022248        176.357941
5            183.605184        175.422379
6            183.963617        173.925476
7            184.101035        175.885239
8            184.095028        176.082199
9            183.949464        166.549332
10           183.638471        164.668365
11           183.228091        165.613770
12           182.812110        167.603073
13           182.446644        168.449997
14           182.196893        172.399994
15           182.020826        173.800003
16           181.873507        172.839996
17           181.773552        170.300003
18           181.645741        169.080002
19           181.589250        169.830002
20           181.676138        173.350006
21           181.758342        170.300003
22           181.645561        168.949997
23           181.335797        170.460007
24           180.980455        164.550003
25           180.588324        168.000000
26           180.239498        168.139999
27           179.954384        168.240005
28           179.801358        166.080002
29           179.809626        168.539993
0.06684639045096678
```

Figure 14: Ensemble results showing performance improvements.

References

- Seasonality on Kaggle. https://www.kaggle.com/code/ryanholbrook/seasonality.
- The solution to the SARIMA's `ValueError` on Reddit. https://www.reddit.com/r/pythonhelp/comments/1dtofq7/import_pmdarima_as_pm_keeps_giving_valueerror/.
- The attached blog post. https://medium.com/analytics-vidhya/combining-time-series-analysis-with-artificial-intelligence-the-future-of-forecasting-5196f57db913.