Week 6 Attempt Summary

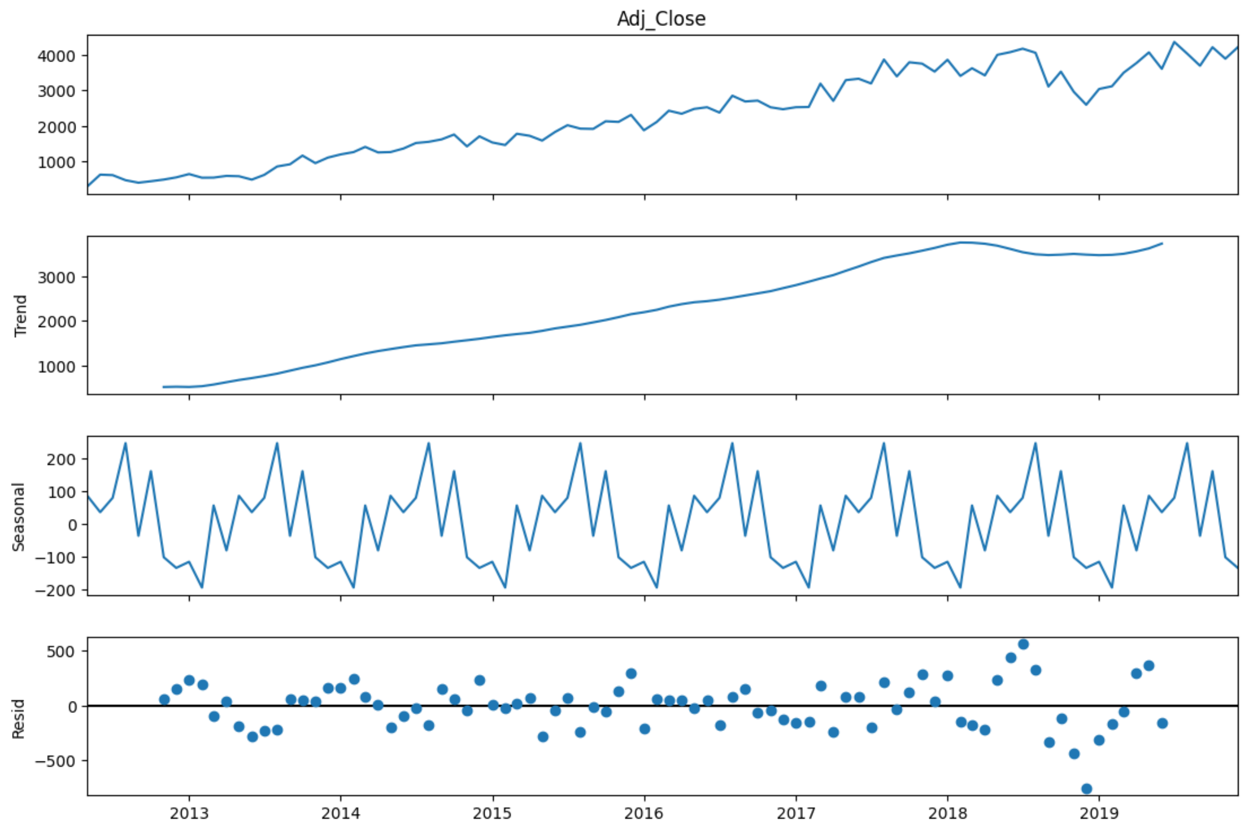
In this week we will be developing a ensemble model for our existing DLM (Deep Learning Model) but first we need to get a good idea on how to build and work with an ensemble model. So this we I created two Jupiter notebooks to show how came to this approach. The Jupiter notebooks are available in my GitHub.

First an ARIMA is 3 consecutive parts,
AR – Auto Regressive, I, Integrated and MA – Moving Average

And SARIMA is basically and extension for the already existing ARIMA model, the extension being adding **seasonality or optimization for seasonal patterns**

we can check for seasonal patterns by using this code – this is implemented in “file – ARIMA model Development on GitHub”

```
decomposition = sm.tsa.seasonal_decompose(data_grouped_mouthly,
model="additive")
fig = decomposition.plot()
fig.set_size_inches(12, 8)
plt.show()
```

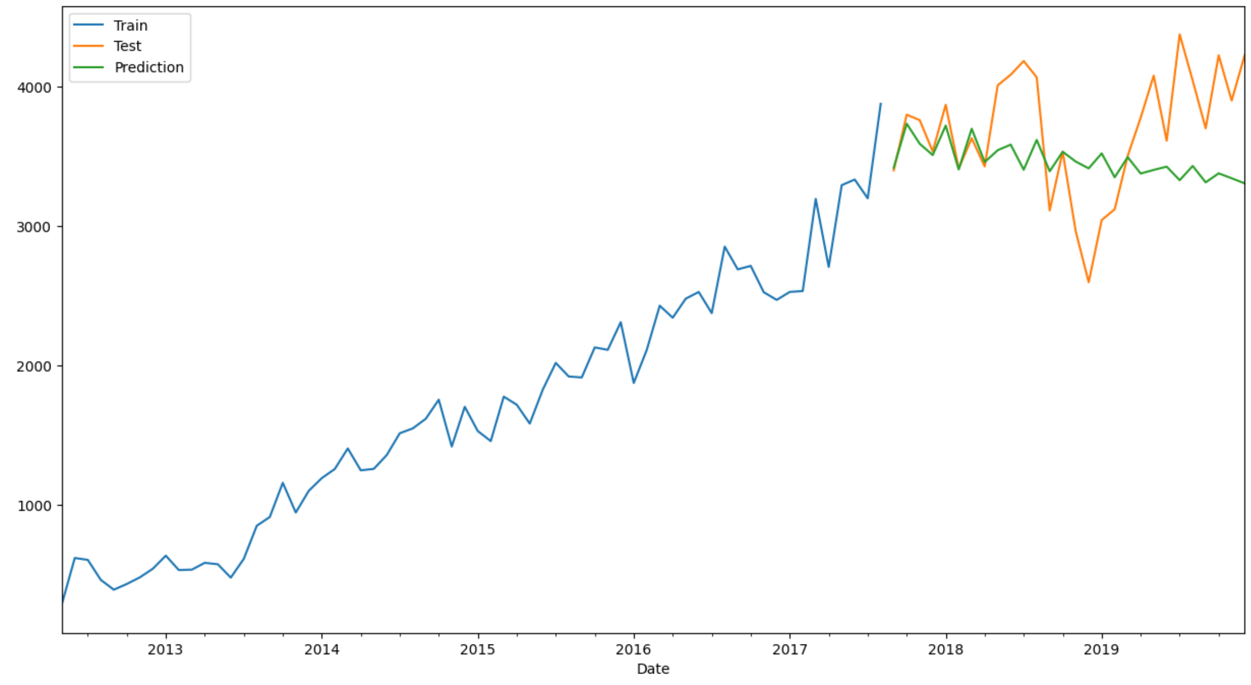


If your data frame is **non-seasonal** your seasonal component in the plot will be flatlined and won't display a trend in the dataset.

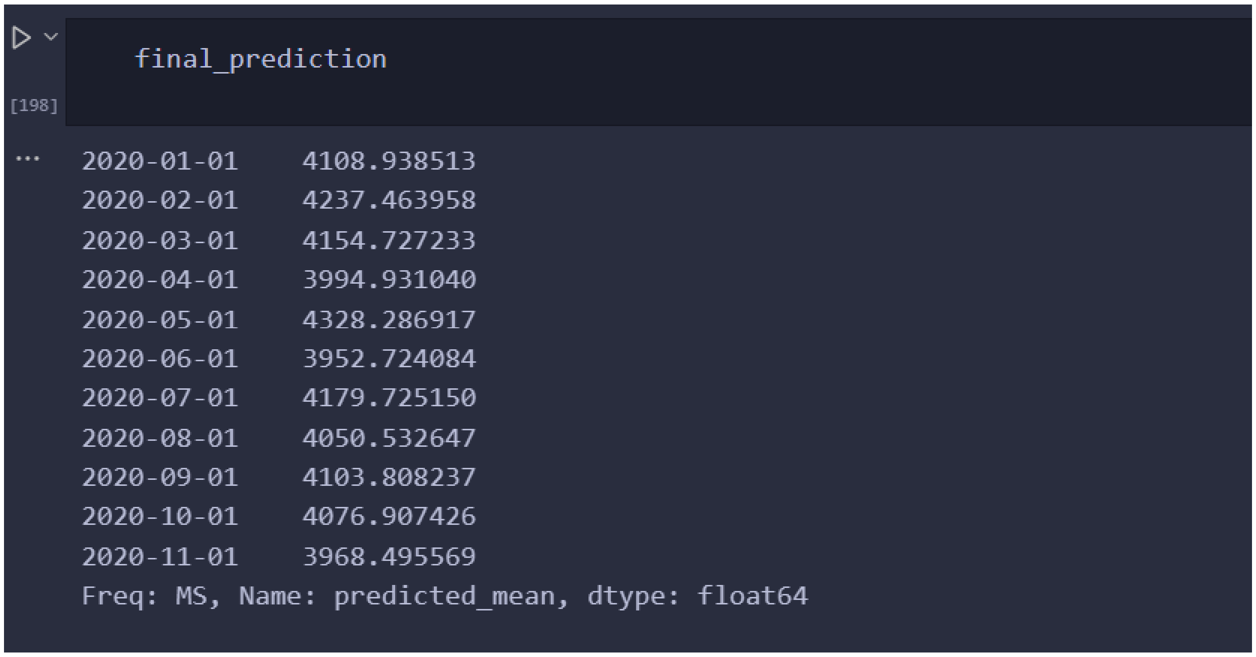
In my opinion, the linear regression model are better when dealing with continuous stock price data other than an ARIMA model. After the model was created, I fitted the data in to the model and checked the mean squared error to check the accuracy of the model, and it was not very appealing

```
 error = np.sqrt(mean_squared_error(test, prediction))
error
```

491.92365720281583 – mean squared error



But by fitting the entire data frame into the model and training the model on that gave a pretty good result when I predicted the future values.

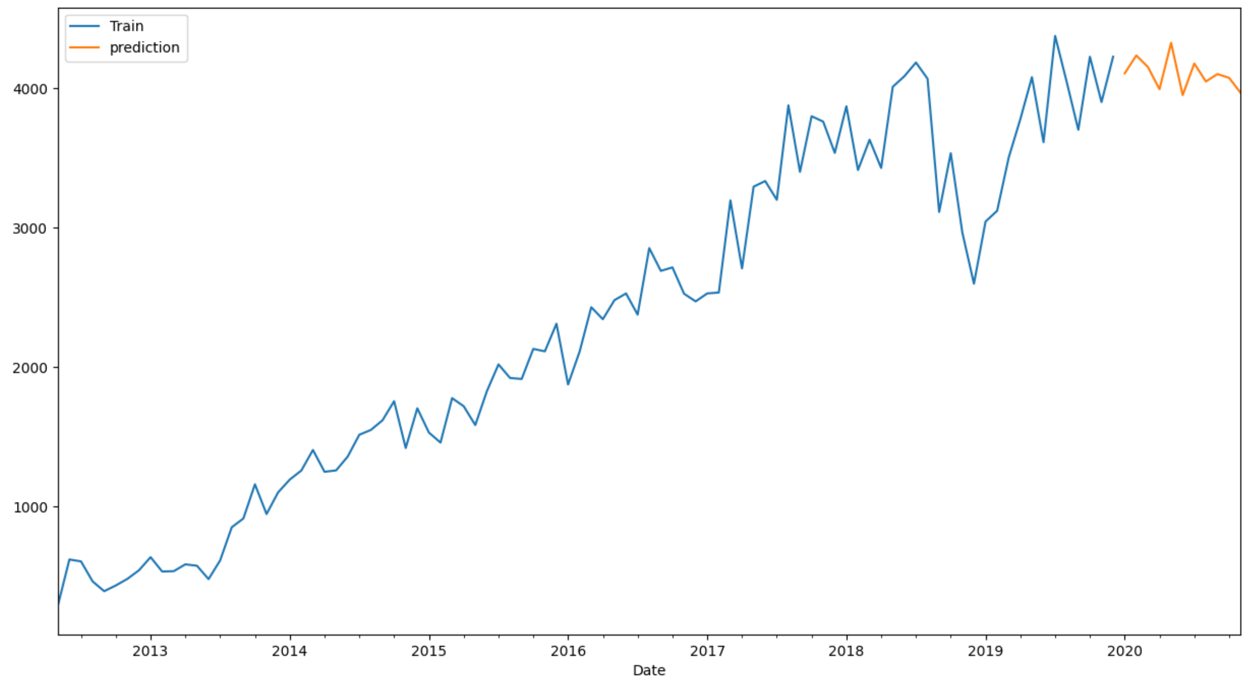


```
final_prediction
```

[198]

```
2020-01-01    4108.938513
2020-02-01    4237.463958
2020-03-01    4154.727233
2020-04-01    3994.931040
2020-05-01    4328.286917
2020-06-01    3952.724084
2020-07-01    4179.725150
2020-08-01    4050.532647
2020-09-01    4103.808237
2020-10-01    4076.907426
2020-11-01    3968.495569
Freq: MS, Name: predicted_mean, dtype: float64
```

I’m not implying that these values are accurate for real world purposes but I’m happy to get some results that does look out rages.



Finally with this collective wisdom I moved on to our current deep leaning model and integrated it with Arima model

Since our data frame is seasonal, I had to use SARIMA to this approach. I implemented a total of 4 functions.

```
def train_arima_model(train_data, order=(5,1,0),
seasonal_order=(0,0,0,0)):
    arima_model = sm.tsa.statespace.SARIMAX(train_data, order=order,
seasonal_order=seasonal_order, enforce_stationarity=False,
enforce_invertibility=False)
    arima_result = arima_model.fit(disp=False)
    return arima_result
```

The first function is just to train the SARIMA model set the seasonal order and creating the SARIMA model like the past few model we been working with

```
def arima_predict(arima_model, start, end):
    predictions = arima_model.predict(start=start, end=end,
dynamic=True)
    return predictions
```

This function is used to do the predictions on the testing set of the data frame and will return the predictions

```
def ensemble_predictions(lstm_preds, arima_preds, weight_lstm=0.5,
weight_arima=0.5):
    combined_preds = (weight_lstm * lstm_preds) + (weight_arima *
arima_preds)
    return combined_preds
```

This function is where we join our prediction to the LSTM model and with weights

```
def calculate_rmse(true_values, predicted_values):
    return math.sqrt(mean_squared_error(true_values,
predicted_values))
```

This function will return the mean squared error to be displayed in the terminal once we run the code.

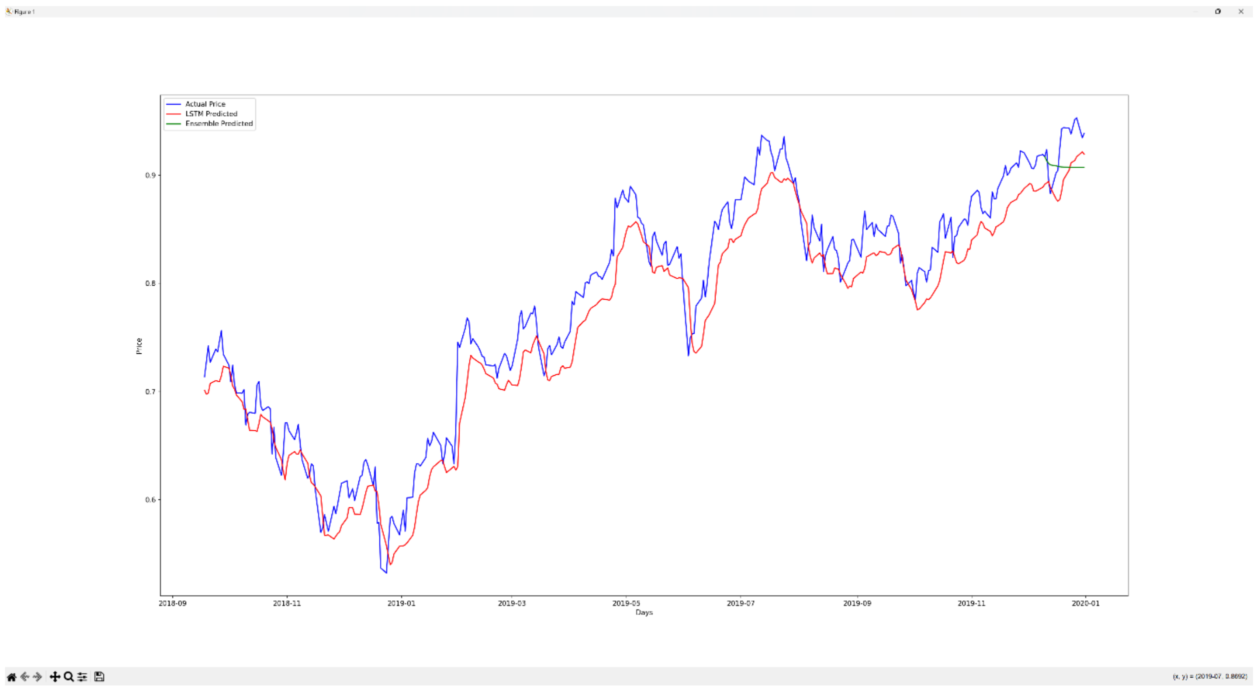
New submission additions

Usage of different hyperparameters is achieved by using matrix-based parameter grid

```
param_grid = {
    'n_estimators' : [100, 200, 300],
    'max_depth' : [10, 20, 30],
    'min_samples_split' : [2, 5, 10],
    'min_samples_leaf' : [1, 2, 4]
}
```

This code will go through different parameter combinations until it finds the best parameter combination that returns the best mean squared error

Combined Prediction Plot



Typically the root mean squared error of the model should return a smaller value, the smaller more better the and accurate the algorithm is

0.06873495343 ~ 0.07

The ensemble model is created to predict only the look_up steps which in this case is 15 days / steps.

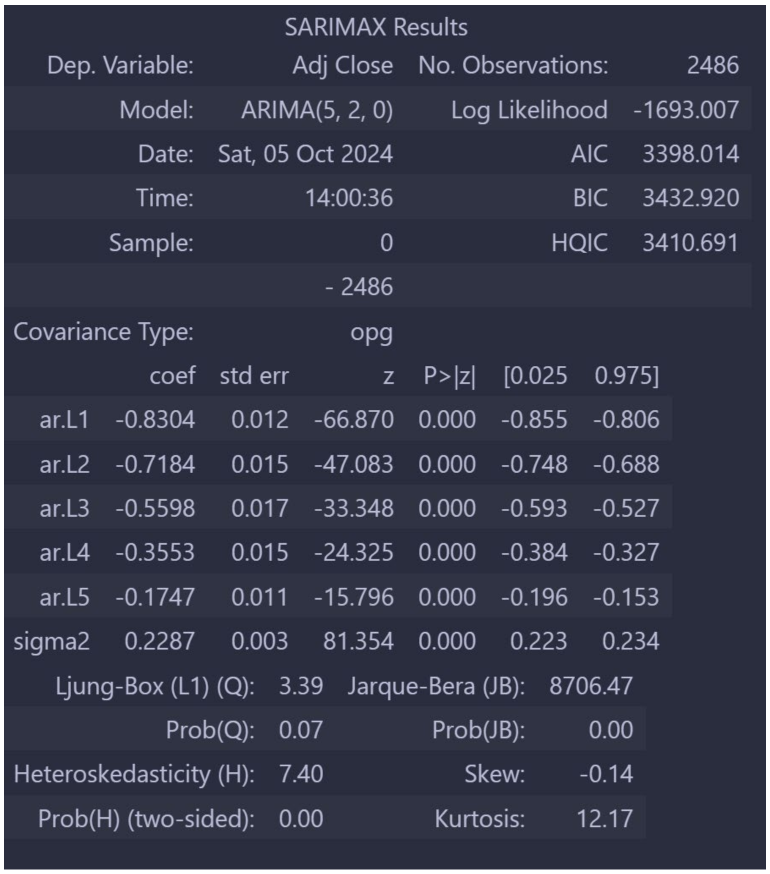
Model explanation

First off, this week we have a ARIMA train function, which takes in 3 arguments which is, training data, order, which was found during debugging stages of the code (using the summary function), and seasonal order because our data set has seasonality.

```
model = ARIMA(train['Adj Close'], order=(5,2,0))
model = model.fit()
model.summary()
```

```
def train_arima_model(train_data, order=(5,1,0),
seasonal_order=(0,0,0,0)):
    arima_model = sm.tsa.statespace.SARIMAX(train_data, order=order,
seasonal_order=seasonal_order, enforce_stationarity=False,
enforce_invertibility=False)
    arima_result = arima_model.fit(disp=False)
    return arima_result
```

then we train the model with the state space library and note that I have used SARIMAX instead of using ARIMAX, I then fit it with the training data then returns the trained model.



```
                         SARIMAX Results
   Dep. Variable:        Adj Close   No. Observations:      2486
           Model:      ARIMA(5, 2, 0)   Log Likelihood   -1693.007
            Date:  Sat, 05 Oct 2024              AIC    3398.014
            Time:          14:00:36              BIC    3432.920
          Sample:                 0             HQIC    3410.691
                             - 2486
Covariance Type:               opg
              coef   std err        z    P>|z|   [0.025   0.975]
   ar.L1   -0.8304    0.012   -66.870   0.000   -0.855   -0.806
   ar.L2   -0.7184    0.015   -47.083   0.000   -0.748   -0.688
   ar.L3   -0.5598    0.017   -33.348   0.000   -0.593   -0.527
   ar.L4   -0.3553    0.015   -24.325   0.000   -0.384   -0.327
   ar.L5   -0.1747    0.011   -15.796   0.000   -0.196   -0.153
  sigma2    0.2287    0.003    81.354   0.000    0.223    0.234
     Ljung-Box (L1) (Q):   3.39   Jarque-Bera (JB):   8706.47
                Prob(Q):   0.07           Prob(JB):      0.00
Heteroskedasticity (H):   7.40               Skew:     -0.14
   Prob(H) (two-sided):   0.00           Kurtosis:     12.17
```

“Arima predict” function which takes in 3 arguments, the created ARIMA model start date, end date and the scaler. Which then we go ahead and use the predict function to actually do the prediction. Then we store the predicted values in numpy array. Then we reshape the predicted values with the same scaler that we used to LSTM

```python
def arima_predict(arima_model, start, end, scaler):
    predictions = arima_model.predict(start=start, end=end,
dynamic=True)
    predictions_array = predictions.values
    predictions_scaled = scaler.transform(predictions_array.reshape(-
1, 1)).flatten()
    return predictions_scaled
```

Then we combine the ARIMA and LSTM predictions, using the ensemble prediction’s function, obviously the function takes in the 2 predictions from ARIMA and LSTM, but we have 2 more arguments, it’s the weight, by default the LSTM and ARIMA weights are set to 0.5 which means that this will contribute to 50% of the final result. The 2 weighted arrays are then added together elementwise to produce the final result.

```
def ensemble_predictions(lstm_preds, arima_preds, weight_lstm=0.5,
weight_arima=0.5):
    combined_preds = (weight_lstm * lstm_preds) + (weight_arima *
arima_preds)
    return combined_preds
```

then we have a calculate RMSE function that will calculate the root mean squared error

```
def calculate_rmse(true_values, predicted_values):
    return math.sqrt(mean_squared_error(true_values,
predicted_values))
```