COS30018- Option B- Task 3: Machine Learning 3

Cody Cronin-Sporys

103610020

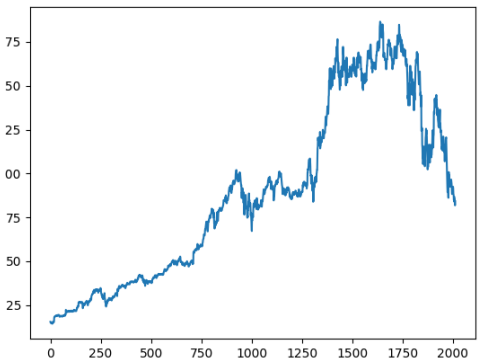
Creating the ensemble model:

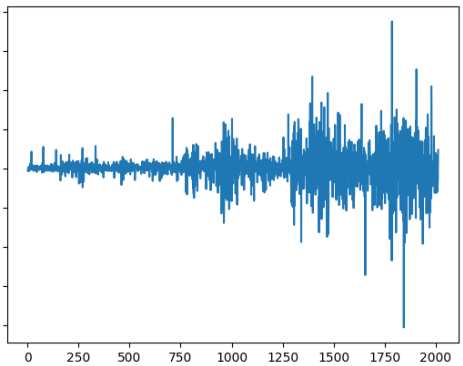
To create the ARIMA model I am using the guide linked to in the CANVAS task instructions and this guide here: https://blog.quantinsti.com/forecasting-stock-returns-using-arima-model/

The first step for creating the arima model is making sure the series is stationary, meaning that it has no trend and has a constant mean and variance over time. If it is not stationary, then we need to apply differencing to the data. To check for a stationary series we use the augmented dickey-fuller test (ADF), for this we'll need the statsmodel library (which is also added to the requirements file).

```
#returns true if the ADF test p value is less than 0.05 (meaning the series is stationary)
def check_stationary(series):
    result = adfuller(series)
    return result[1] <= 0.05

def create_arima_model(data):
    df = data['df']
    #apply differecing if needed
    if not check_stationary(df['adjclose']):
        df['close_diff'] = df['adjclose'].diff()
```

This code checks to see if the series is stationary then applies differencing through the panda diff() function if the series is not stationary. We can see the results by plotting series before and after differencing:

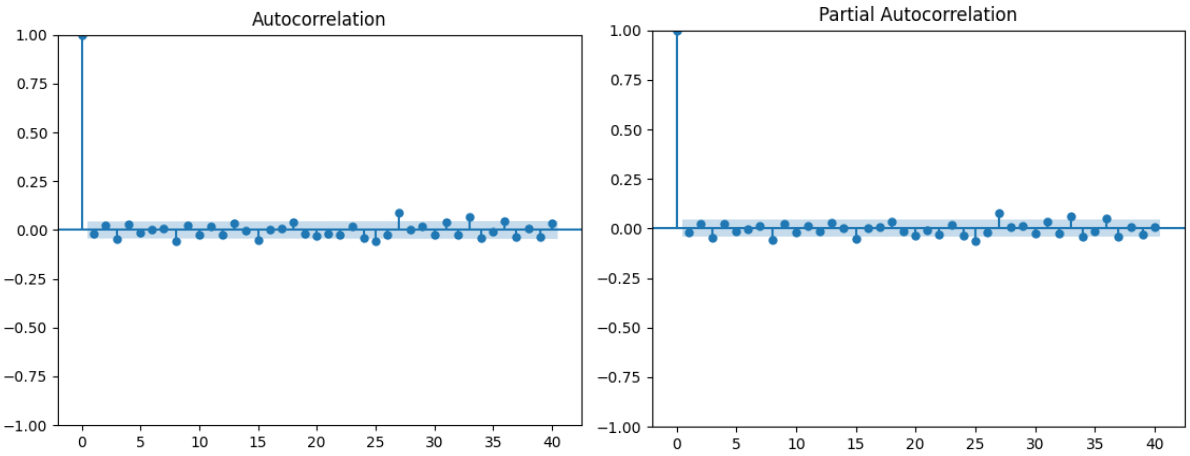




The next step of the model is to find the appropriate p and q values. These are the order for the Autoregressive (AR) and Moving average (MA) models. They can be found using an autocorrelation function (ACF) for the p value and a Partial Autocorrelation function (PACF) for the q value.

We first need to plot the acf and pacf models using the statsmodel library. After having some issues trying to get this working, I found the issues was NaN values in my dataframe caused by the differencing, so using the panda dropna() function solved the issue and produced these plots:

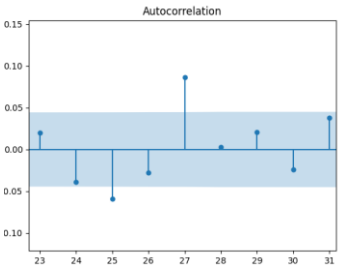
```
plot_acf(df['close_diff'].dropna(), lags=40)
plt.show()

plot_pacf(df['close_diff'].dropna(), lags=40)
plt.show()
```



To understand how to use these graphs to chose a p and q value, I used this guide here: https://www.baeldung.com/cs/acf-pacf-plots-arma-modeling

This explains that for the p value, I should pick any lag point that's outside of the blue shaded area on the PACF plot. Zooming into the graph I can see a point at 27, so I'll use the for my p value.

For the q value, we can do the same, but looking at the at the ACF plot. Again, I can see a strong correlation at 27 that falls outside the blue shaded area, so I'll use 27 for the q value.



I then adjusted the code slightly to allow for multiple differencing levels. The combination of all steps has given the d, p and q values needed to create the ARIMA model:

d = 1, p = 27, q = 27

```
p = 27
q = 27


model = ARIMA(df['adjclose'], order=(p, d, q))
return model
```

This caused an error due to an issue with the library refencing being outdated, so I update it and ran fine.

Now that we have the model, we can return to the stock_prediction file. Here we will perform fitting with the new arima model and make a forecast to prediction the price in the future.

```
model = create_model(PREDICTION_DAYS, len(FEATURE_COLUMNS), loss=LOSS, units=UNITS, cell=CELL, n_layers=N_LAYERS,
                     dropout=DROPOUT, optimizer=OPTIMIZER, bidirectional=BIDIRECTIONAL)
arima_model = create_arima_model(data)


# Now we are going to train this model with our training data
# (x_train, y_train)
model.fit(x_train, y_train, epochs=EPOCHS, batch_size=BATCH_SIZE)

print('model fitting')
arima_fit = arima_model.fit()
print('model fitted')

forecast = arima_fit.forecast(steps=LOOKUP_DAYS)
```

Once we have the forecast price, we can combine it with the predicted price from the original model (LSTM) and find an average predicted price. We can also add a weighting value to each model's prediction to form a weighted average based on which models perform better.

```
ARIMA_WEIGHT = 0.9
LSTM_WEIGHT = 1.1

ensemble_prediction = ((forecast * ARIMA_WEIGHT) + (predicted_price * LSTM_WEIGHT)) / 2
print(f"Future price after {LOOKUP_DAYS} days is {predicted_price:.2f}$")
print(f"Future ensemble price after {LOOKUP_DAYS} days is {ensemble_prediction:.2f}$")
```

This was producing errors as the forecast variable contains multiple values in an array, so I adjusted the forecasting line to only store the actual prediction value.

As an example for ARIMA and LSTM models with no weighting, we get a LSTM prediction value of $84.53 and an ARIMA prediction value of $84.16, giving an ensemble prediction value of $84.35