# Exchange-Traded Funds (ETFs) Prediction with LSTM-ARIMA hybrid model

Group 18

Hsu, Ya Cheng (3035550250)
Abheeshek Gupta (3035601621)

# Project Settings and Objectives

Exchange-Traded Funds or more commonly known as ETFs are investment funds that are usually traded on stock exchanges. They are very similar to stocks. They can hold assets such as commodities, stocks, or bonds. Some famous examples of ETFs are SPDR S&P 500 ETF. (SPY), iShares MSCI Emerging Markets ETF (EEM) and Vanguard Total Stock Market ETF(VTI).

Now, many would-be wondering what makes ETFs so special, why could we not design a simple model that predicted the price of some S&P500 companies like Apple, Alphabet Facebook. Below, we have done a comparison between ETFs and Stocks, and why ETF has become a preferred investment option.

Advantages of ETFs over Stocks:

1. **Lower Costs**: As ETFs trade like stocks, you can buy a diversified portfolio with a low commission. ETFs also have low expense ratios when compared to other forms of investments.
2. **Tax Efficiency**: As ETFs are usually not actively managed, but are usually trained to follow a specific index, thus they might not have high income and capital gains that are needed to be passed on to owners each year. This means investors have more control over when they incur taxes.
3. **Available in Alternative Investments:** ETFs allow investors to take positions for the alternative investments. New products become are available regularly and these can include ETFs in hedges, commodities and leveraged long and short positions in indices and sectors.

Due to these advantages of ETFs when compared to stocks and mutual funds, have been gaining popularity. ETFs have become a favorite choice for investors all across the globe. Hence the need for prediction of the ETF trend in the near future, to help investors make good decisions becomes important.

Most existing models that are present usually are used for predicting stocks of top companies. We could not find an ideal model for predicting ETF prices. This was also due to the factor that ETF trading is very dynamic and changes rapidly, and most models that are present at the moment give unsatisfactory performance.

For this project, we propose to create a model that predicts the short term performance of 4 popular Exchange-Traded Funds. By short term performance, we mean the result of the 100th

day after the training is completed. To improve the efficiency and performance of our predicted result we intend to use a Hybrid ARIMA-LSTM model.

The reason we chose a hybrid model was that we spotted a better performance using the Autoregressive Integrated Moving Average (ARIMA) model than the Long Short-Term Memory (LSTM) model. However, the question for us now is - can we do better? Thus, we want to use a hybrid ARIMA-LSTM model to see if we can obtain an even better prediction power. To predict the ETF market better, we can't ignore the trend as well as its seasonal fluctuation. According to the "Time series forecasting using a hybrid ARIMA and LSTM model" (Fathi, 2019), we can use the ARIMA model to isolate the periodic pattern and to obtain the model parameter that will be extremely helpful for the next LSTM model. Hence, we have utilized the known strength of both models, namely ARIMA in picking up the periodic component and LSTM in picking up the trend.

The dataset we are planning to use for our model is the "Huge Stock Market dataset" (Boris Marjanovic, 2017), that is available on Kaggle and is provided by Boris Marjanovic. The dataset contains Date, Open, High, Low, Close, Volume, OpenInt for all the stocks and ETFs in the CSV format. The Licence of the dataset is CC0: Public Domain and the Usability of the dataset is 7.5. This dataset contains historical daily prices of all US ETFs and Stocks till 10th November 2017, thus the dataset helps in providing us a general idea of how the ETFs and Stock are performing.

The ETFs that we are using in this project are all based in the United States of America and can be traded on the NASDAQ and NYSE.

The ETFs are as follows:

- SPDR S&P 500 ETF (SPY)
- iShares MSCI All Country Asia ex Japan ETF (AAXJ)
- Direxion Daily Russia Bull 2X Shares (RUSL)
- iSHARES INC/EDGE MSCI MIN VOLAT (ACWV)

The reasons we decided to use specifically these 4 ETF Dataset as they are all different types of ETFs, they also are of varying size, even the variance of our all the 4 ETFs is also different. Inorder to prove our model is robust and can predict in all circumstances we have chosen these 4 ETFs.

The machine learning framework we are using for this project is Tensorflow2.0, as it is widely used in models that require stock prediction. We ran our model on Google Colab because it provides us shared editing and free GPU usage that makes it better than jupyter notebook.

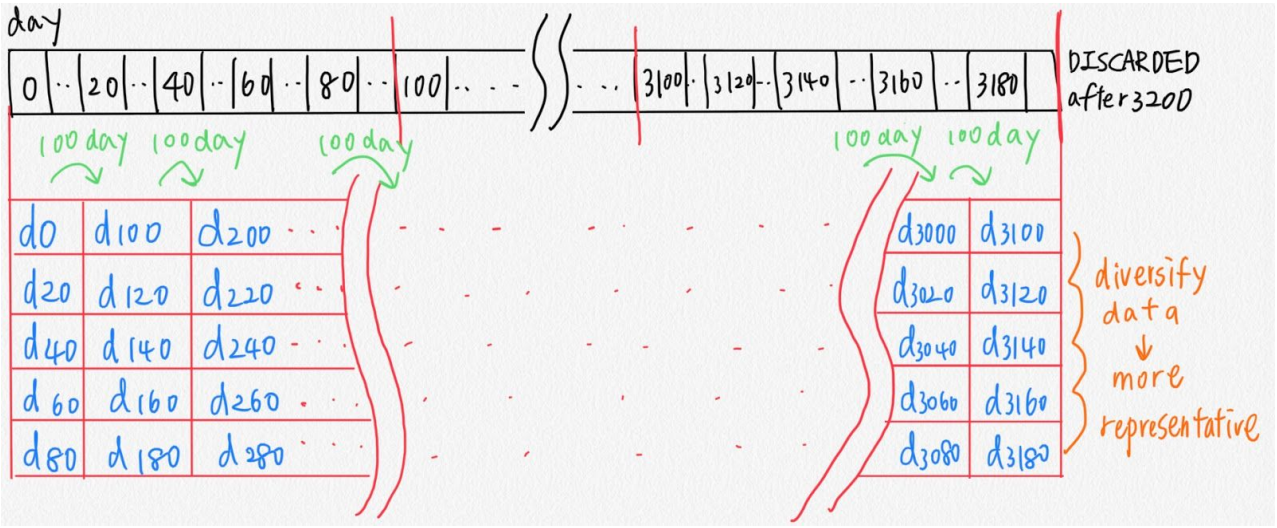
## Data Pre-processing

As explained in the project objectives, our datasets are gathered through "Huge Stock Market Dataset" from Kaggle, containing price data for almost a thousand American stocks as well as exchange-traded funds (ETFs). We chose four datasets whose symbols are SPY, AAXJ, RUSL, and ACWV respectively. Through Google Colab `files.upload()` method, we read the original csv file given by the Kaggle dataset into DataFrame format with columns including "Date", "Open" (open price of the day), "High" (maximum price of the day), "Low", "Close", "Volume" (number of shares traded that day), "OpenInt" (open interest on that day). Since we're most interested in the price, we drop the columns of "Volume" and "OpenInt" with `drop()` method since they won't directly contribute to the time-series trend. Following the drop of irrelevant columns, we further cast the "Date" column from type 'string' to type 'datetime' using the `to_datetime()` method from pandas, then replace the index as "Date" using `set_index()` method for easier manipulation on time series data. By now, the dataset should contain four columns named "Open", "Close", "High", "low" with index representing the date the price was present, where the number of indices may be different across SPY, AAXJ, RUSL, and ACWV. We show the number of days recorded in each exchange-traded fund (ETF) in the following table.

| **ETF Symbol** | SPY | AAXJ | RUSL | ACWV |
|---|---|---|---|---|
| **# of days recorded** | 3201 | 2325 | 1627 | 1491 |

[number of days recorded for each ETF]

Despite the difference in days recorded as well as the large volume of our datasets, we reduced the volatility of each dataset by enabling a 100-stock-day (since ETF trading doesn't operate on weekends and holidays) stride among each data point in "Open", "Close", "High", and "Low". To make each data point more representative, we enrich the dataset by taking different 'start' days. Take "Open" price for example, the first column generated by open price has day 0 as the first entry, the second column has day 20 as the first entry, the third column has day 40 as the first entry, the fourth column has day 60 as the first entry, the fifth column has day 80 as the first entry. We repeated the same for "Close", "High", and "Low", all implementing the 100-stock-day stride to keep our dataset less volatile, but retain the representativity through different start days. The following graph can illustrate the aforementioned idea clearer.




[hand-drawn representation of the organization of dataset]

We achieve the aforementioned volatility reduction and diversification code by slicing the original data with stride and by taking different values (day 0, day 20, day 40, day 60, day 80) as the start of slicing. Then, we also allocate the training, developing, and testing dataset in this step, each different by a lag. The process as illustrated in the following pseudo complete data pre-processing on our financial time series data.

```
for i in range (4) :  // open, close, high, low
−for j in range (0, 100, 20) :  // different 'start' day
−− slice the i column with 100 day stride til the end  --(1)
−− append the data in (1) to a  LIST
concatenate df in the LIST along columns  // align day 0, 20 along y axis
```

[pseudo code for volatility reduction and diversification]

# ARIMA

Autoregressive Integrated Moving Average (ARIMA) model is a linear regression model specifically designed to deal with stationary time series data. By 'stationary', ARIMA model can achieve stellar performance while catching linear trends in time series data. The hyperparameters of ARIMA include p (parameters for AR model), d (order of differentiation), and q (parameters for MA model). We'll explain how these parameters are determining factors of a model later.

P is the auto-regressive term, which means the number of previous observations necessary to predict the price of an Exchange Traded Fund at the present time. In other words, the present time value $X_t$ can be modeled as the weighted sum of the values of its previous p time stages

$X_{t-1}$, $X_{t-2}$, …, $X_{t-j}$. We can determine the value of p by inspecting the autocorrelation plot using `plot_pacf` from `statsmodels.graphics.tsaplot`. If the plot appears to cut off (directly near 0) after p lags, the p-value is then determined. Empirically speaking, there might be spikes after the graph cuts off, which may be the seasonal effect of your interested time-series data. However, if the phenomenon isn't consistent, it may be due to emergency events occurring within our chosen time frame.

$$X_t = \sum_{j=1}^{p} \varphi_1 X_{t-j}$$

[p: auto-regressive term]

D is the order of differentiation, which will be considered when the time series isn't stationary. To transform the time series into a stationary series $X_t^*$, we differencing the series by its previous entries $X_{t-d}$ until it reaches a stationary process. Empirically speaking in business time series data forecasting, taking first-order differentiation should be more than enough (Adhikari & Agrawal, 2013).

$$X_t^* = X_t - X_{t-1} - \ldots - X_{t-d}$$

[d: order of differentiation]

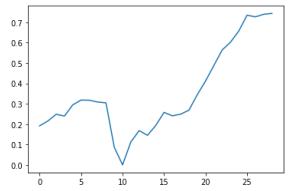
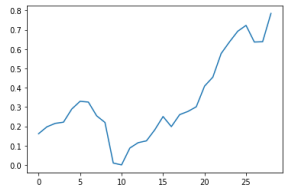
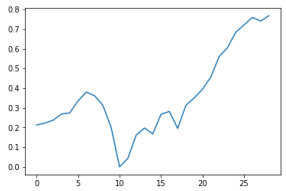
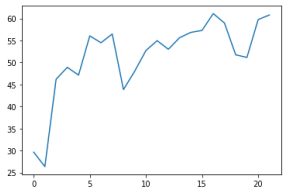
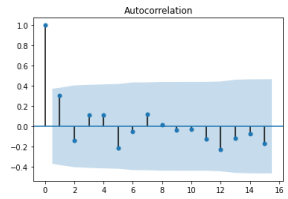
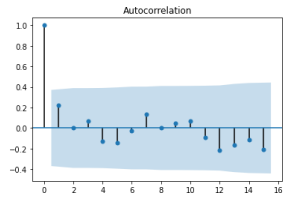
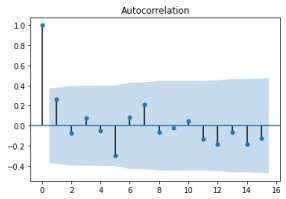
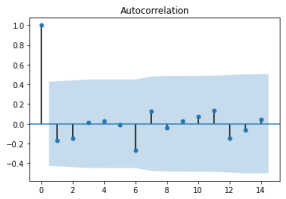
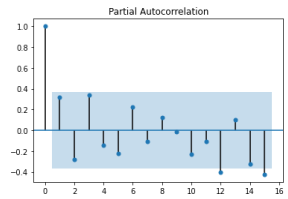
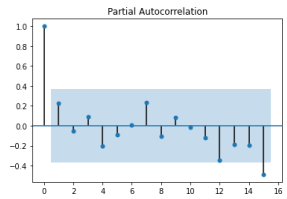
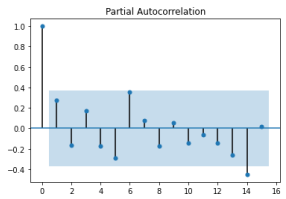
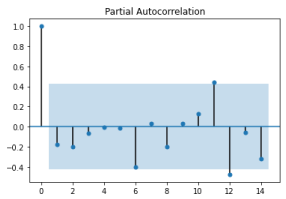
Q is the moving average term used to control the noise, which achieves error correction by the number of moving average jumps deployed to predict the current value. In other words, $X_t$ can be modeled as the weighted sum of its innovation processes $\varepsilon_{t-j}$ (i.e. the difference between the actual value and the forecast based on previous j time stages). We can determine the value of q by inspecting the `plot_acf` from `statsmodels.graphics.tsaplot`. If the plot appears to cut off (suddenly near 0) after q lags, the q value is determined. Similarly to the autoregressive term, there might be spikes after the graph cuts off, which may be the seasonal effect of your interested time-series data. However, if the phenomenon isn't consistent, it may be due to emergency events occurring within our chosen time frame.

$$X_t = \sum_{j=1}^{q} \theta_j \varepsilon_{t-j}$$

[q: moving average term]

We looped over twenty columns (5 for Open, Close, High, Low respectively), plotting out its exchange-traded fund (ETF) price time series, as well as the crucial `plot_pacf` and `plot_acf` that are essential to determine the p and q value in ARIMA(p,d,q). Here we take the order of differentiation to be 1 since the time series will already be stationary after one

differentiation. The following showcase some of the results we acquired and its respective parameters p, d, and q.

| ARIMA(0,1,2) | ARIMA(1,1,1) | ARIMA(0,1,1) | ARIMA(2,1,2) |
|---|---|---|---|
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

To avoid a manual error when determining the value of p, d, and q through human eyes, we first check out possible values of p and q through the plots, where the results give that p and q is both within [0,3] (2). To check whether a certain model is the best-fit, we compute the error metric Akaike Information Criterion (AIC) over all possible sets of (p,d,q) the model can take in (2) - the model with the least value shall be the best fit for our data. We did the same not only on our training dataset but also on our as developing dataset as well as the testing dataset. This looping process as described by below's pseudo code ensures we fit datasets on its best-suited ARIMA model.

```
for column in columns :  // five each for open/close/high/low
−for d in [train, dev, test ] :
−− best index = 0 ; AIC min  = 1000  // initialization
−− for j in all possible parameter orders :  // p and q is in  [0,3] while d=1
−−− fit column of d with ARIMA(j)
−−− if the new AIC < AIC min: // update best-fit model while looping
−−−− best index and AIC min updated with the new model
```

```
-- final best index of the j for loop is best fit model, fit the dataset again
```

[pseudo code for fitting model]

Next step, we can make in-sample prediction using the model fitted onto our dataset earlier, and quite reasonably, we can observe some differences between the ground-truth time series and the prediction time series. Thus, we calculate the residual by subtracting prediction with the ground-truth across time. Since the ARIMA model is expected to filter out the linear trend, the residual is likely to be non-linear trends that the LSTM model is good at capturing or pure noises/outliers that result from unexpected events occurring in our time frame. The residual itself is a time series, so we intended to feed those into the LSTM model to do further prediction on the time series data's non-linear trends.

# LSTM

LSTM or Long-Short Term Memory is an artificial recurrent neural network,which is commonly used for deep learning. LSTM works on feedback connections. It is one of the few Neutral Networks that can process single data points as well as entire sequences of data. LSTM can perform various tasks such as  processing, making predictions and classifying based on the time-series data.

Usually Neural Networks have a good performance ratio on non linear tasks, this is due to the reason it has a large dimension of parameters, it also has a nonlinear activation function for each layer. This all enables the model to adapt to non-linear trends of the data.

To get a better understanding of LSTM, it becomes important to understand how Recurrent Neural Networks work.

RNN takes the sequence of the vectors of the time series data as input

$$X = [x_1, x_2, x_3, \cdots, x_t]$$

It outputs a vector value, which is commuted through a neural network in the model

To get a clearer understanding of the RNN, let us suppose there is a ' Vector X' which is supposed to be time series data that spans time t periods. Then let the values that are held by 'Vector X' be passed through 'Cell A'  in sequential order. During each step the 'Cell A' outputs a value that is then concatenated or merged with the time step data available next, and the 'Cell state C' .

The output value and of C and the output value are used as input for the next time step The process is continuous and repeats itself until the last time step data.

LSTMS cells usually consist of multiple neural networks and each of them represents a forget gate, input gate,input candidate gate along with the output gate.

The forget gates purpose is to serve as a forgetter which is then multiplied to the cell state $C_{t-1}$ from the former time step to drop values that are not needed and keep those that are necessary for the prediction. It's done to drop the non needed values and retrain the necessary ones. The output by the forget gate is in the form of a vector whose element values are between 0 and1.

$$f_t = \sigma(W_f * [h_{t-1}, x_t] + b_f)$$

The input gate and the input candidate gate will operate together to form a new cell $C_t$.$C_t$ is passed to the next time step and becomes a renewed cell state. Input gate uses the Sigmoid function as the main activation function and the input candidate gate is utilized as a hyperbolic tangent. Both these gates output $i_t$ and $C`_t$. The input gate and the input candidate gate operate together to render the new cell state $C_t$, Then $i_t$ is used to select the feature of C` that should be reflected in $C_t$It can be better understood by the equations below.

$$i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i)$$
$$\tilde{C_t} = tanh(W_C * [h_{t-1}, x_t] + b_C)$$

The output gate is used to decide which of the values are to be selected, combining $o_t$ with the tanh-applied state of $C_t$ as output $h_t$ . The new cell state that is formed by the combination of the forget gate applied to the former cell state $C_{t-1}$ and the new tanh-applied state $C_t$, The equation will give more clarity.

$$o_t = \sigma(W_o * [h_{t-1}, x_t] + b_o)$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C_t}$$
$$h_t = o_t * tanh(C_t)$$

The output $h_t$ and cell state $C_t$ will be passed to the next time step and will continue to go through the same process. Depending on the task, further activation functions such as the Softmax or Hyperbolic tangent can be applied to $h_t$ 's.

For this project we will use residual data that we obtained from the ARIMA Model of the ETF. The residual will act as the input for the LSTM model. These datasets will contain Train X and Train Y, Dev X and Dev Y as well as Test X and Y.

In the model we are going to use 33 LSTM units. The total number of units is considered a parameter referring to the dimensionality of the output state and dimensionality of the hidden state . The architecture of the model for our task is an RNN neural network that employs 33 LSTM units .

The final outputs of 33 LSTM units are concatenated into a single value with a fully connected layer. The value obtained is then passed through a doubled-hyperbolic tangent activation function. In our code it is defined by double_tanh(x). This function outputs a final single prediction.

The dropout method used in the code is used to prevent overfitting. It prevents the neurons to develop an interdependency that results in overfitting. This is executed by just turning off the neurons in the network during the training phase with a probability p. During the testing phase, dropout is disabled and each of the weight values are multiplied by the probability p, this is done to scale down the output value into a desired boundary.

To prevent overfitting regularization is also used. The two types of regularization used are ridge regularization and lasso regularization. These regularizers help in keeping the weight of the values of each network in LSTM from becoming too large. For our model, through trial and error we found that if we don't apply any regularization , the results yielded are better.
Another important aspect is that we are using the ADAM optimizer as it helps in giving good results.

A walk-forward approach is used to fit our model. Our model is required to be fitted in rolling time intervals. For each time interval, the newly trained model is tested on the next time step. Our model is an optimal model with the Mean Squared Error (MSE) metric. For further evaluation, the Mean Absolute Error (MAE) is also investigated. This helps in the robustness of our model.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y_i})^2$$

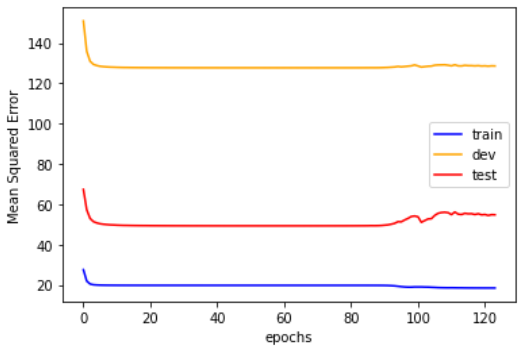
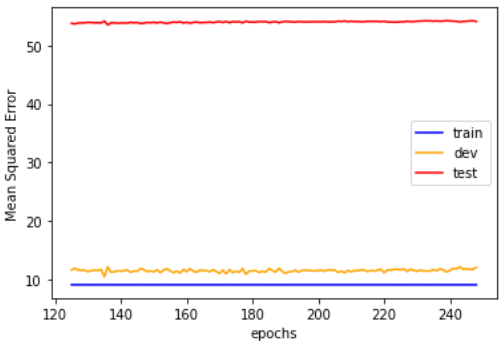
$$MAE = \frac{1}{n}\sum_{i=1}^{n}\left|y_i - \tilde{y_i}\right|$$

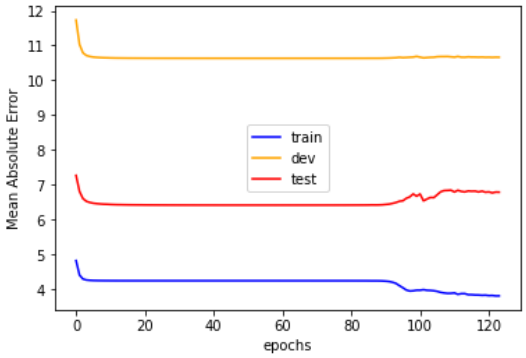
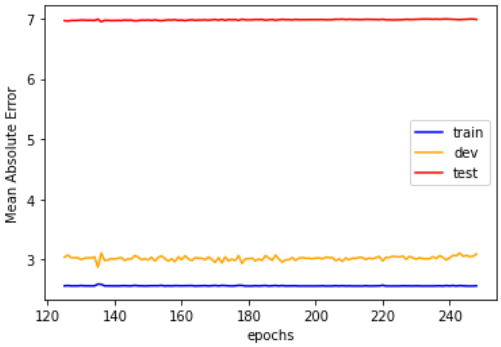
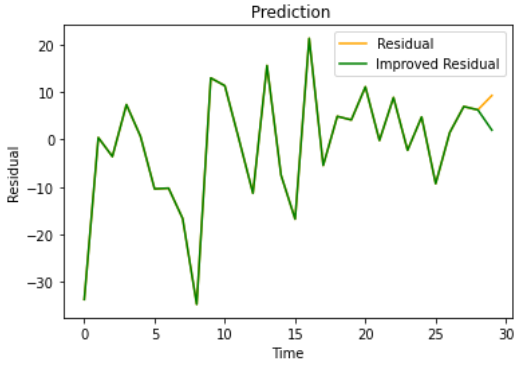
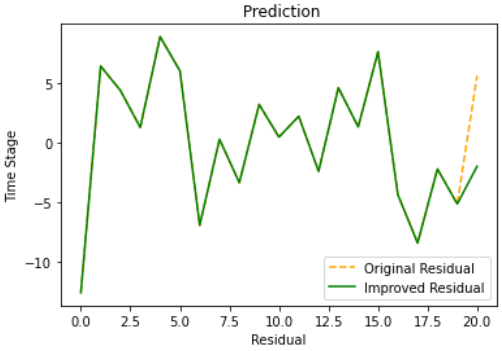
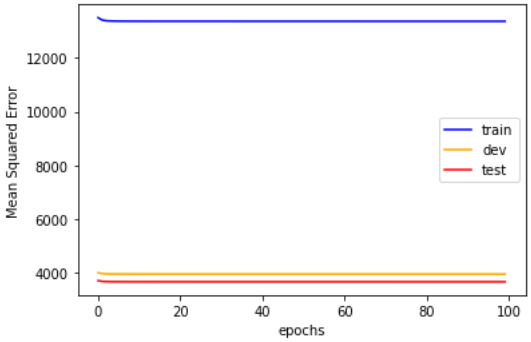
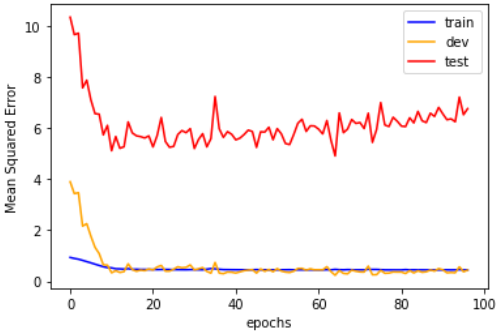
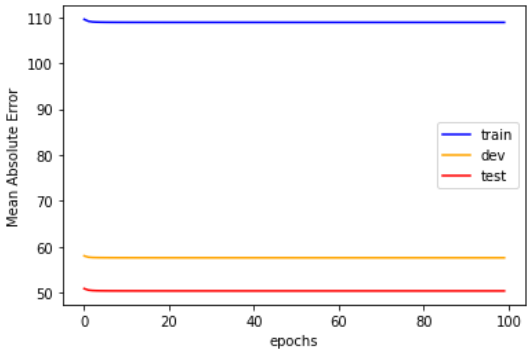
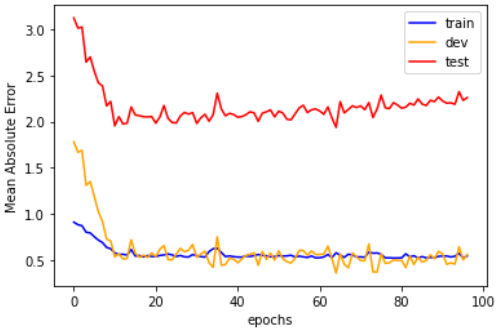
For better understanding and clarity on how our code works, we are providing the algorithm for the LSTM model.

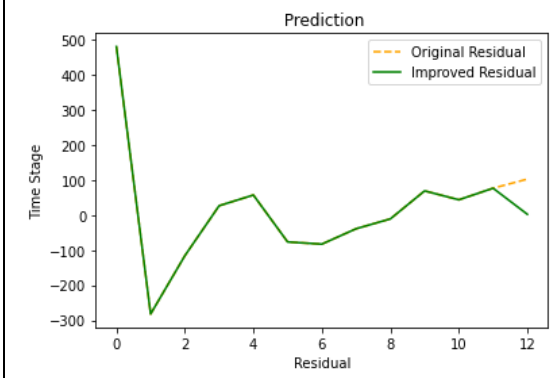
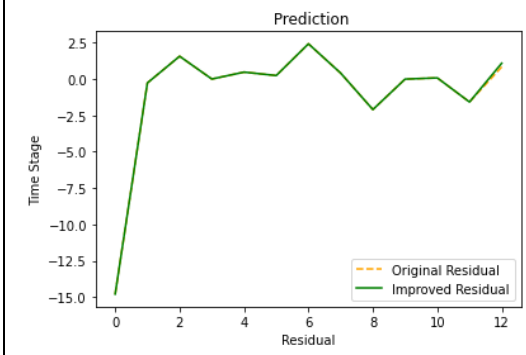
*Define model*
*− add LSTM (units=33)*
*− add Dense(shape=(12,1), activation='double-tanh'*
*Repeat*
*− Forward_Propogate model using train_X*
*− Backward_Propogate model using train Y*
*− UPDATE model parameters*
*− train_MSE, train_MAE=model (train_X,train_Y)*
*End Repeat*
*dev_MSE, dev_MAE=model (dev_X,dev_Y)*
*test_MSE, test_MAE=model (test_X,test_Y)*

[pseudo code for LSTM model]

# Conclusion

| ETF | SPY | AAXJ |
|---|---|---|
| **MSE** |  |  |

| | | |
|---|---|---|
| **MAE** |  |  |
| **Residual** |  |  |
| **ETF** | **RUSL** | **ACWV** |
| **MSE** |  |  |
| **MAE** |  |  |

| | | |
|---|---|---|
| **Residual** |  |  |

Analysis

The Time Train  MAE and MSE are smaller (3 out of 4 cases). For RUSL we can observe that the train error is very high. We believe this is due to the fact that the unprecedented steep decrease in the value of the ETF dataset. If you look in the notebook, it can be seen that the dataset achieved its high in the beginning and has just gone downhill with huge fluctuations. This resulted in RUSL having a high train error.

Looking at the MAE and MSE graphs above it can be concluded that the SPY and ACWV are overfitted because they have a less avoidable bias and the Degree of overfitting to Dev Set which is measured by the difference between Test Error and Dev Error, the Test Error is greater than the Dev Error. But on the other hand for  RUSL and AAXJ Test Error is smaller than the Dev Error and has a high avoidable bias, thereby we can conclude, they are underfitted.

Residual

Since we feed the residual of the ARIMA model as the input for our LSTM model, we planned to tune the residual into a smaller one through training using the LSTM model. To get a brief look into how the residuals have been minimized, we plot out the residuals of all time stages for the first column, one using the predicted residuals based on the model, the other using its ground-truth residuals. The following table gives the comparison of the residual of the last time step in the four datasets chosen, by prediction and ground truth.

| | SPY | AAXJ | RUSL | ACWV |
|---|---|---|---|---|
| Ground-truth last timestep residual | 9.3204303 | 5.5930640 | 102.170776 | 0.8382262 |

| | | | | |
|---|---|---|---|---|
| Predicted last timestep residual | 1.9972272 | -1.991371 | 1.9997895 | 1.0894804 |

[the residual of prediction v.s. Ground-truth]

The residual of SPY, AAXJ, RUSL has significantly declined at least by 60%, while ACWV doesn't show significant results if solely judged on the residual of the first column. From the residual graph of ACWV, we can see that the residuals started to hover around 0 consistently from second time stage until the last one, which suggests the ARIMA model has successfully captured the linear trend of the time-series, while it doesn't have much non-linear trend for LSTM model capture, thus leading to an unsatisfactory performance of the LSTM model on ACWV residuals.

Limitations

Even though our model is robust, we have noticed some limitations that our model model may contain. Firstly, even though our model is very good in predicting data for the next time step. Our model can not predict data for more than one time step. This is also due to the volatility of the times series data., because the price of ETFs highly depends on the everchanging condition of the Markets, and sometimes markets can perform badly due to uncontrollable factors for instance the Covid 19 Pandemic or Political Instability and Uncertainty. Secondly, even though our model can be used to predict different types of ETF datasets, we believe that for better, efficient and accurate predictions, you should customize the hyperparameters according to the given dataset. This will enable our model to have a better performance ratio.

## Link to walkthrough video

https://youtu.be/ZJhx2Mq4wxU

## References

Stock Market Prediction by Recurrent Neural Network on LSTM Model

https://blog.usejournal.com/stock-market-prediction-by-recurrent-neural-network-on-lstm-model-56de700bff68

Stock Price Prediction using RNN

https://github.com/TannerGilbert/Tutorials/blob/master/Keras-Tutorials/5.%20Stock%20Price%20Prediction%20using%20a%20Recurrent%20Neural%20Network/Stock%20Price%20Prediction.ipynb

ARIMA Model – Complete Guide to Time Series Forecasting in Python

https://reurl.cc/b5VVW6

An Introductory Study on Time Series Modeling and Forecasting

https://arxiv.org/abs/1302.6613

What is an ETF? Advantages & Disadvantages

https://www.arborinvestmentplanner.com/what-is-an-etf-advantages-disadvantages-newsletter/

Stock Price Correlation Coefficient Prediction with ARIMA-LSTMHybrid Model
https://arxiv.org/pdf/1808.01560.pdf?fbclid=IwAR36Xn6RyV_nFRyAL794NDY8nHB31Yqdp9tR0TlsLCwo4anQrlr6iERFGkU