Summary:

Table of Contents:

Section 1:

Goals:
- Predict/Future stock price movement
    - Ideally a month out

Steps
- Retrieve stock price data
- Calculate technical indicators which might aid in price forecasting
- Process the data for the models (Multivariate LSTM)
- Develop a SINGLE-STEP model
- Develop a handful of SINGLE SHOT multi-step models
- Evaluate the performance

Retrieving Stock Price Data:
- From fmpcloud.io API (Have a subscription. I recommend, it is great)
- LLY stock data from 04-15-2015 - Present (Eli Lilly & co.)

Calculate Technical Indicators
(Not going into the T.I. formulas/explanations behind them, that is not the purpose of this project)
- Used Technical Analysis library
    - Relative Strength Index (RSI)
        - Helps identify overbought/oversold market conditions
    - Awesome Index
        - Measures market momentum based on H/L & MAs
    - Kaufman's Adaptive Moving Average (Kama)
        - Moving Average that considers market noise/volatility
    - Rate of Change (ROC)
        - Pure momentum indicator
    - The Percentage Volume Oscillator (PVO)
        - Volume momentum indicator
    - Average Directional Movement Index (ADX)
        - 3 total indicators
        - Works together to identify directionality & strength of trends
    - Daily Log Return (DLR)
        - Displays "returns" in another format which is linked to compounding
    - Commodity Channel Index (CCI)
        - Measures difference between price change & average price change
        - Helps determine above/below average prices which indicate strength

- Average True Range (ATR)
    - Indicates the degree of price volatility
- Schaff Trend Cycle (STC)
    - Cyclical Oscillator which indicates market trends based on EMAs & price windows
- Detrended Price Oscillator (DPO)
    - Oscillator which removes pricing trends & identifies cycles

Preprocessing

- Merged data
- Dropped irrelevant columns
- Indexed the columns for the models
- Split the data Training/Validation/Testing
- Defined number of features for the model
- Normalized each data set based of the training mean/stdev (0-1)
- Imported the TensorFlow developed functions for windowing & modeling

TensorFlow Functions

- Referenced from https://www.tensorflow.org
- The functions referenced were developed by TensorFlow
- This is why they were sourced to another ipynb script
- I did not want to cause confusion or assume credit for functions developed by another person
- So, I treated them like any another library of functions that was imported
- The imported functions include
    - Windowing the provided data into datasets which are TF model compatible
    - Splitting each window
    - Plotting the outputs
    - Compile & fit function for each model (edited)
        - This source code was edited to test with different optimizers etc.

Defining Windows

- Define the window using the Window Generator Function
- The models will make a set of predictions based on a window of consecutive samples

Width = number of timesteps of the input & label windows
Offset = the timestep offset between them

Examples:
input_width = 30
label_width = 1
offset = 30

- Predicts a single days price 30 days in the future based on 30 days of previous data

input_width = 6

label_width = 1
shift = 1
 - Predicts a single days price 1 day in the future based on the previous 6 days of data

input_width = 20
label_width = 10
shift = 10
 - Predicts a 10 days of prices immediately following the learning period of 20 days

For our models, it makes the most sense to have a higher label_width since we are trying to predict price movement.
In other words, the overall movement is more important than short term fluctuations. (This is why we skim over the single step model & focus on the single shot multi step model)

Section 2:

LSTM Summary:
 - LSTMs are a special kind of RNN.
 - LSTMs keep track of both long-term & short-term inputs
 - LSTM[cells] are made up of the following:
    - Input Gates
    - Hidden State
    - Forget Gates
    - Output Gates

  - The main idea of LSTM models:
    - A “memory cell” can maintain its state/explicit memory overtime & through gating, regulates the information flow in/out of the memory of the cell
      - Gating allows us to only pass-through prevalent information from the previous state & also decide which information is valuable from the current state

Forget Gate: The first part of the LSTM cell
 - Acts as a filter for information passed from the previous cell
 -The previous output & current inputs (with bias) combined via element-wise concatenation.
 - Then passed to a "gate" which is just a sigmoid function followed by pointwise multiplication (recurrent activation parameter)
    - Pointwise multiplication: Given functions f1 and f2, the pointwise product is f = f1*f2 or f(x) = f1(x) * f2(x)

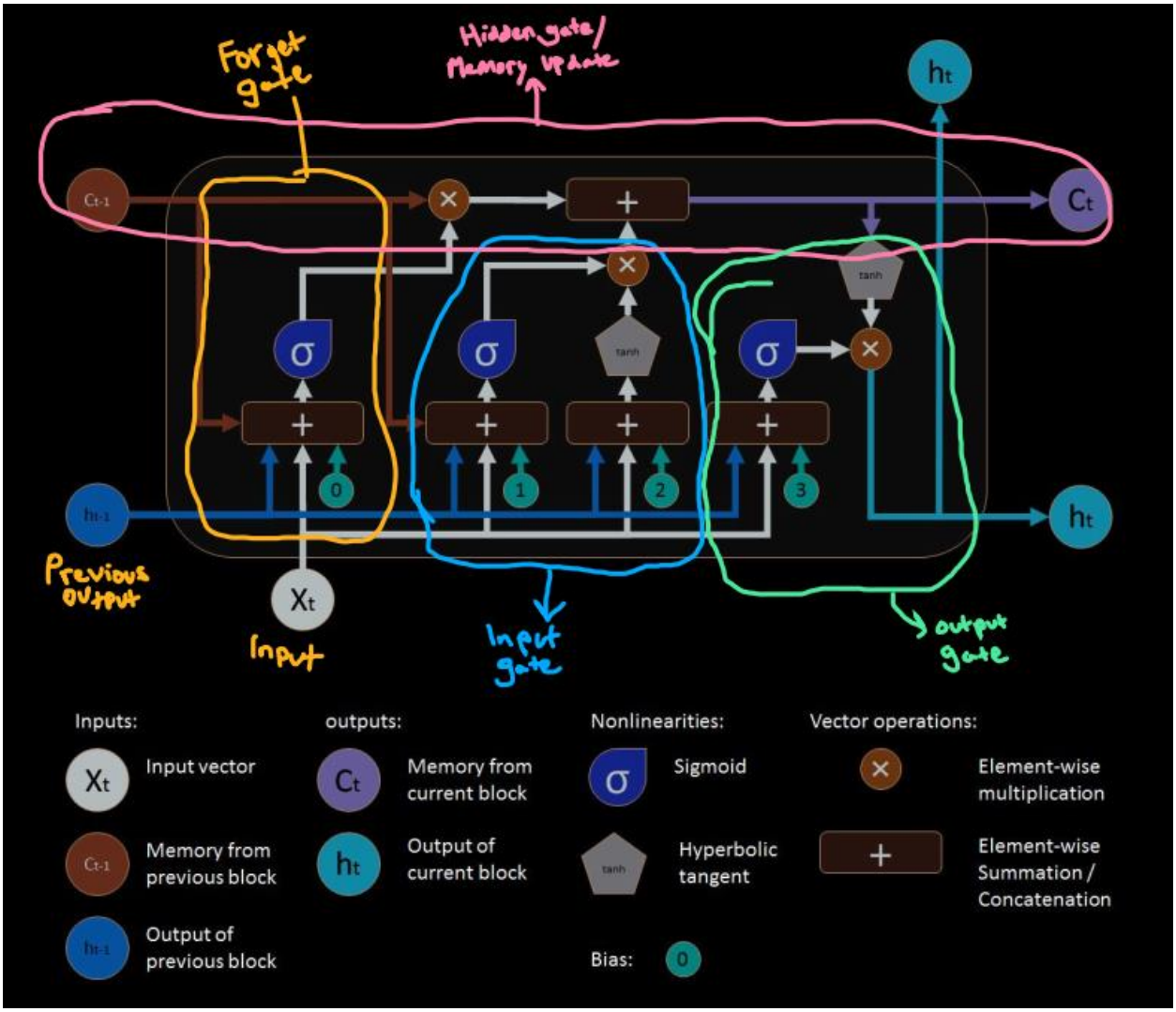
ft = σ(Wf*[ht-1,xt] + bf)
Where…
- ft = Gate output
- σ = a sigmoid function
- Wf*[ht-1,xt] = Concatenation of previous output & current input
- bf = bias added

Input Gate: The second part of the LSTM cell
- Decides which information is to be added in the cell state after the pair/element-wise summation
- We have 2 layers, a sigmoid & tanh
- The sigmoid layer determines which information will be updated
    - ft = σ(Wf*[ht-1,xt] + bf) same form as before
    - Except instead of determining which information will be passed, this determines what is updated
- The tanh layer creates new candidate values (updating weights) (activation parameter)
    - Ct = tanh(Wc*[ht-1,xt]+bc)
    - Essentially the same function as before, except this introduces a tanh activation function rather than sigmoid
- Tanh Activation:
    - Similar to sigmoid (except normalized outputs [-1,1])
    - Works in determining which values to pass

Hidden State/Memory Update: The third part of the LSTM cell
- Updates what new information to be added to the call state from the current cell
- The Cell State then aggregates the two components produced via the Forget Gate and the Input Gate
- Ct = ft*Ct-1 + it*Ct
    Where the first section is the Forget layers output combined with Ct-1, which is the updated memory output from the previous cell (Ct-1)
    And the second section is the output from the Input Gate (Both the sigmoid and tanh parts of the input gate)

Output Gate: The final part of the LSTM cell
- This is the section which determines the short-term memory output
- This concatenates the previous output, current input and bias through a sigmoid activation layer which decides which parts of the cell will be used as output
- The second part passes the hidden state/memory update output through a tanh activation layer
- Both parts are combined through element/pointwise multiplication
- The product is then used as the output of the current block

Please see the diagram below for an illustration of the gates discussed above.
Provided by:
*Long Short-Term Memory Networks.pdf (wisc.edu)
^^Fantastic Explanation
Also, another reference: RNN and LSTM—an overview. Kumar (2021)



Single Step Models:

- These models are not extremely useful for our purposes, but we will make one anyways
- They do not take the full window of input data into account
- Predicts one future day at a time, for a 30-day sequence based on the previous 30 days of prices (one at a time)

Multi-Step Models:

- Learns & predicts on a range/sequence of future values
- Unlike a single-step model, which only predicts one step at a time

For multi-step models there are 2 approaches

1. Single-Shot: The entire future sequence is predicted at once based on the previous input data
2. AutoRegressive: Predicts a future range/sequence, BUT the model produces single-step predictions & feeds them back into the model as inputs

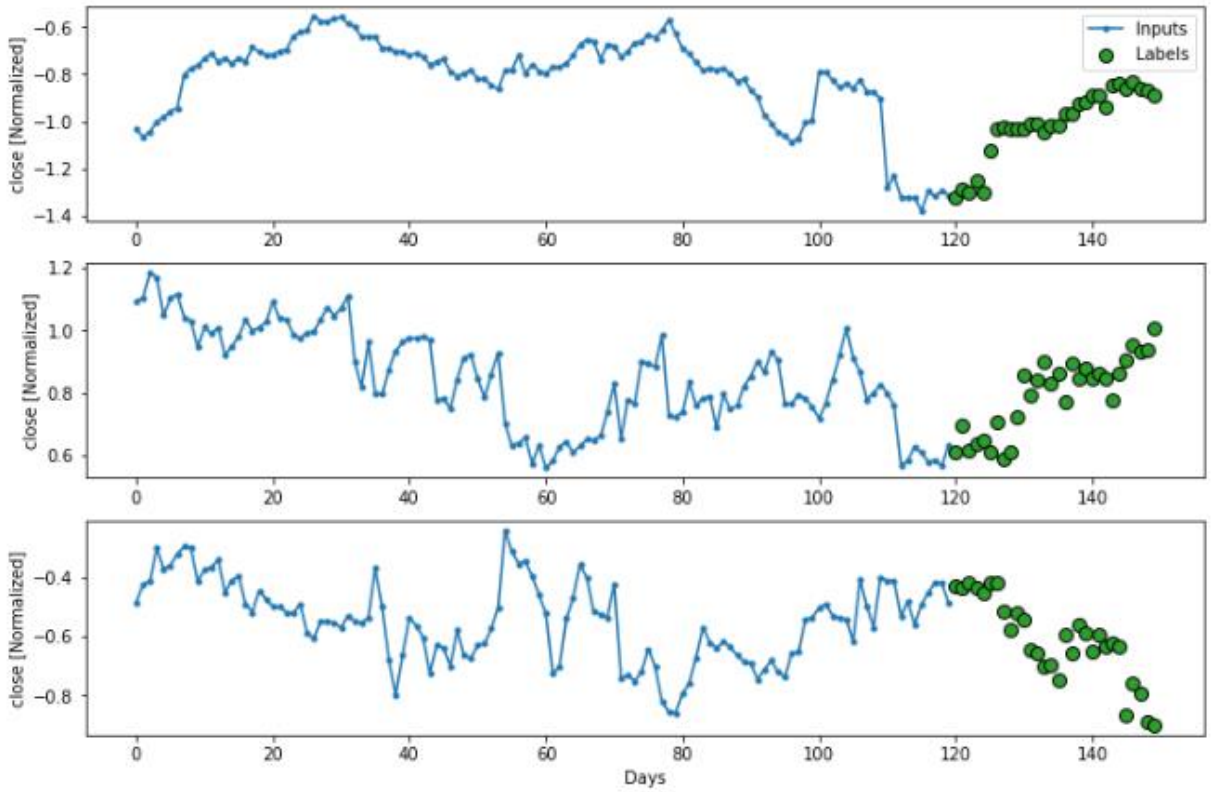
The models we developed are Multivariate Single-Shot Multi-Step meaning,

- The model has multiple input features (the technical indicators)
  - The history of multiple variables was collected as input for the LSTM analysis
  - The goal is to predict future price movement based on past price movement along with other indicators past values
- The model predicts an entire sequence at once based on the previous values

LSTM Multi-Step Model Summary:

input_width = 120
label_width = 30
shift = 30

This means the model learned on 120 days of price data & predicted the following 30 days. See below for the inputs/labels graphs.
I tried to get a good sequence of data which included price increases, decreases and stagnations.



- Seven models were developed with an additional “baseline” which continuously predicted the most recent value.
    - Each had parameters adjusted in an adhoc fashion according to the performance of the previous model
    - The adhoc “tuning” primarily consisted of experimenting with different dense/dropout/normalization layers after the initial LSTM layer
        - For the dense layers we could mainly tweak the kernel initializer which controlled the weight distributions

Overall Model:

- One LSTM layer followed by 1-2 Dense layers with a Dropout or Normalization layer sparsely included for experimentation for some models.
- Only one LSTM layer was utilized to reduce model complexity & avoid any overfitting that might be introduced

LSTM Layer Parameters:

- Units: 256
    - Analogous to the number of nodes in a standard neural network
    - Units is the size of the hidden gate (discussed above) for each LSTM state/cell

    - In other words, the length of the hidden gate (otherwise known as the memory state, internal vector state, etc – is it the top section of the LSTM cell)
        - Each unit can be seen as an LSTM unit
        - So if we have a unit size of 2, we would have 2 LSTM units inside a single LSTM cell – and we would have many LSTM cells inside one LSTM layer
    - Cannot remember the reference, but often – multiples of 32 are chosen – specifically 128. First tested with 128 & got better results with 256
- Activation: Tanh
    - This is the default activation function for the LSTM layer
    - It is discussed above in the cell section
- Recurrent Activation: Sigmoid
    - Discussed above in the cell section
- Kernel Initializers:
    - Determines the distribution for kernel weight initialization
        - Variance Scaling
            - Returns an initializer that generates tensors without scaling variance. (TF)
            - Keeps the scale of the input variance constant
                - This means the scale of the gradients/weights should be similar thorough out all layers
- Recurrent Initializer: Orthogonal/Truncated Normal
    - Determines the distribution for the recurrent weights (the recurrent activation parameter)
        - Truncated Normal
            - Allows for more generalization compared with the default
            - It is essentially a normal distribution & setting all values outside the truncated range to 0
            - Hence, more generalization
        - Orthogonal
            - RNNs & LSTMs contain repeated matrix multiplication
            - Repeatedly updating an internal state using a single weight matrix can cause stability issues
            - Orthogonal allows for repeated multiplication without the matrix exploding or vanishing
                - This allows for more effective back propagation during optimization
                - *Explaining and illustrating orthogonal initialization for recurrent neural networks (2016)*
- Return Sequences: False
    - The layer will only return the last hidden state output
        - Set to false since
            - The LSTM layer is followed by a dense layer
            - For a single-shot model, we can set to false since the model only needs to output at the final timestep.

Dense Layer Parameters:

- Activation: None (Linear)
- Kernel Initializer: Variance Scaling, Random Uniform
    - Variance Scaling:
        - See above – keeps the scale of input variance constant
        - Provides similar gradients/weights
    - Random Uniform
        - Randomly assigns weights with a uniform distribution
        - Also experimented with Constant
            - Generates constant values

Optimization: Adam
- Adam Optimizer (Adaptive Moment Estimation)
    - Optimization technique for gradient descent
    - "Combination" of Gradient Descent with Momentum & RMSProp
        - Momentum
            - Uses exponential weighted averages to accelerate the gradient descent
            - Calculate position change
                - update = $\alpha * m_t$
                - Where α = Step Size
                    - A controls movement AKA: Learning Rate
                - And m_t = aggregate of current gradients
            - Update the old positions using the update
                - $W_t + 1 = w_t - \alpha * m_t = (w_t - \text{update})$
            - Where $m_t = \beta m_{t-1} + (1-\beta)[\delta L / \delta w_t]$
                - Where w_t = weights
                - δL = derivative loss function
                - δw_t = derivative weights
                - β = moving average parameter
            - So, update = learning rate * aggregate of gradients
            - And the aggregate of gradients = Moving avg parameter*previous aggregate of weights + (1 – moving average parameter) * [derivative of loss / derivative of weights]
        - This is how the weights are updated using the momentum method

        - RMSProp
            - Attempts to improve on AdaGrad
            - Uses exponential moving average instead of cumulative sum of squared gradient (like AdaGrad)
            - The formula is $W_t + 1 = w_t - (\alpha_t / \text{sqrt}(v_t) + e) * (\delta L / \delta w_t)$
                - Very similar to the one above
                - Where,
                    - $v_t = \beta * v_t + (1-\beta) * (\delta L / \delta w_t)^2$
                - Where,
                    - α = Learning Rate (Step Size)
                    - δL = derivative loss function

        - δw_t = derivative weights
        - β = average parameter
        - e = constant
      - So we have,
      - Weight update + 1 = weight update – (tunable learning rate / sqrt(average parameter*v_t + (1-average parameter)*(derivative of loss / derivative of weights)^2) + some constant * (derivative of loss / derivative of weights)
      - All of this means that
        - The weight updates are bound by the learning rate
        - As the learning rate is decreasing, the weight updates will be smaller and smaller
  - So, now knowing what Adam consists of…
    - Adam works to implement bias corrected m_t and v_t from the equations above
      - Where the new m_t and v_t equal
        - $m_{tHat} = m_t /(1-\beta_1)$
        - $v_{tHat} = v_t /(1-\beta_2)$
      - And we substitute the new forms in the Adam optimizer general function to get:
        - $W_{t+1} = w_t - m_{tHat}(\alpha/(sqrt(v_t)+e))$
        - Or, $w_t = w(t-1) – \alpha*(m_{tHat}/(sqrt(v_t)+e))$
        - Which is how adam updates the weights & optimizes the model

References for the Optimizer discussions above:
optimization.cbe.cornell.edu
geeksforgeeks.org

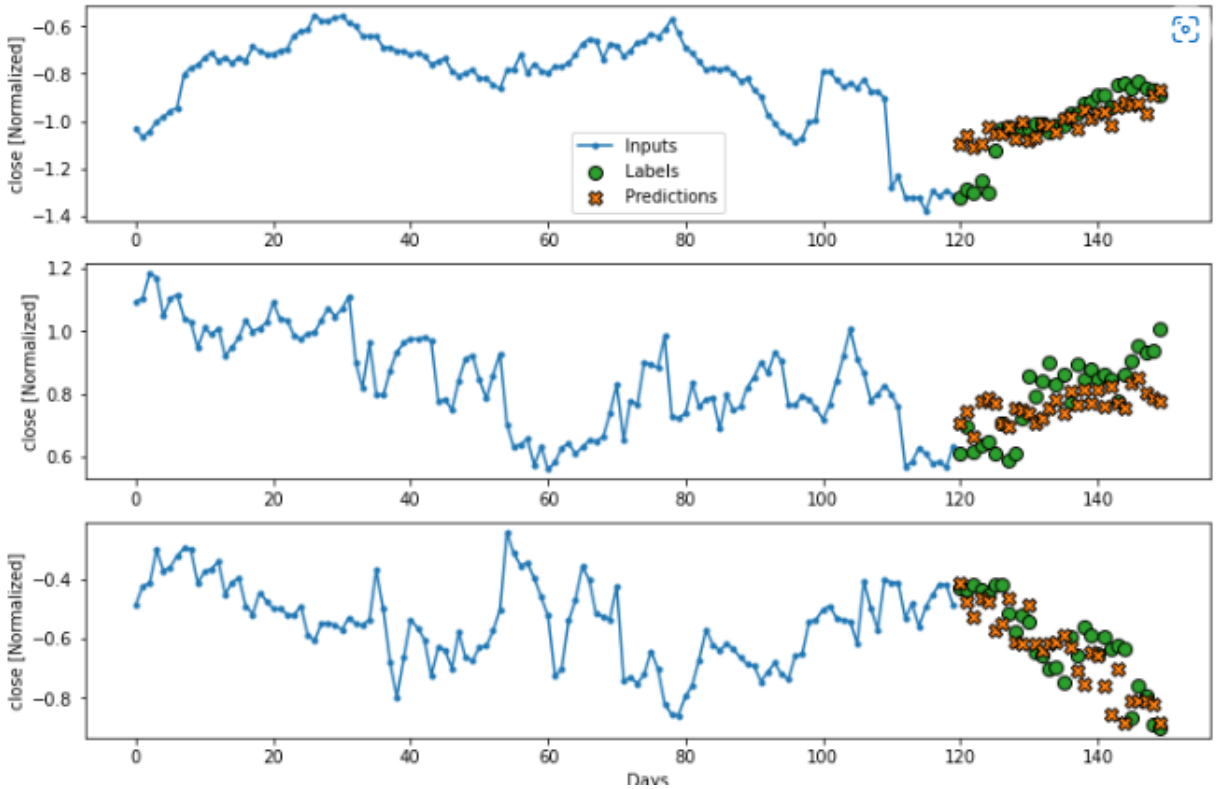
Results:
- The performance was evaluated on the validation MAE.
- The “baseline” model discussed in a previous section was used as a reference point
- All other models were compared with each other
- Since there were seven, each having different parameters, we are only going to cover the best
  - Given, the “best” may not have been the model will the lowest MAE.
  - The goal of the project was to predict price movement with broad strokes, rather than pinpointing peaks/troughs
- For example, a model may produce a smaller MAE, but it may not have generalized as well with respect to the overall monthly price movement (which was the intention)

None the less, here are the results & best models

Results:

	Val Loss	Val MAE
Baseline	20.338121	3.661733
LSTM_1	8.878493	2.650110
LSTM_2	12.956505	3.278734
LSTM_3	13.041225	3.293097
LSTM_4	14.067743	3.488424
LSTM_5	13.076362	3.289307
LSTM_6	13.592550	3.411404
LSTM_7	12.942861	3.321061

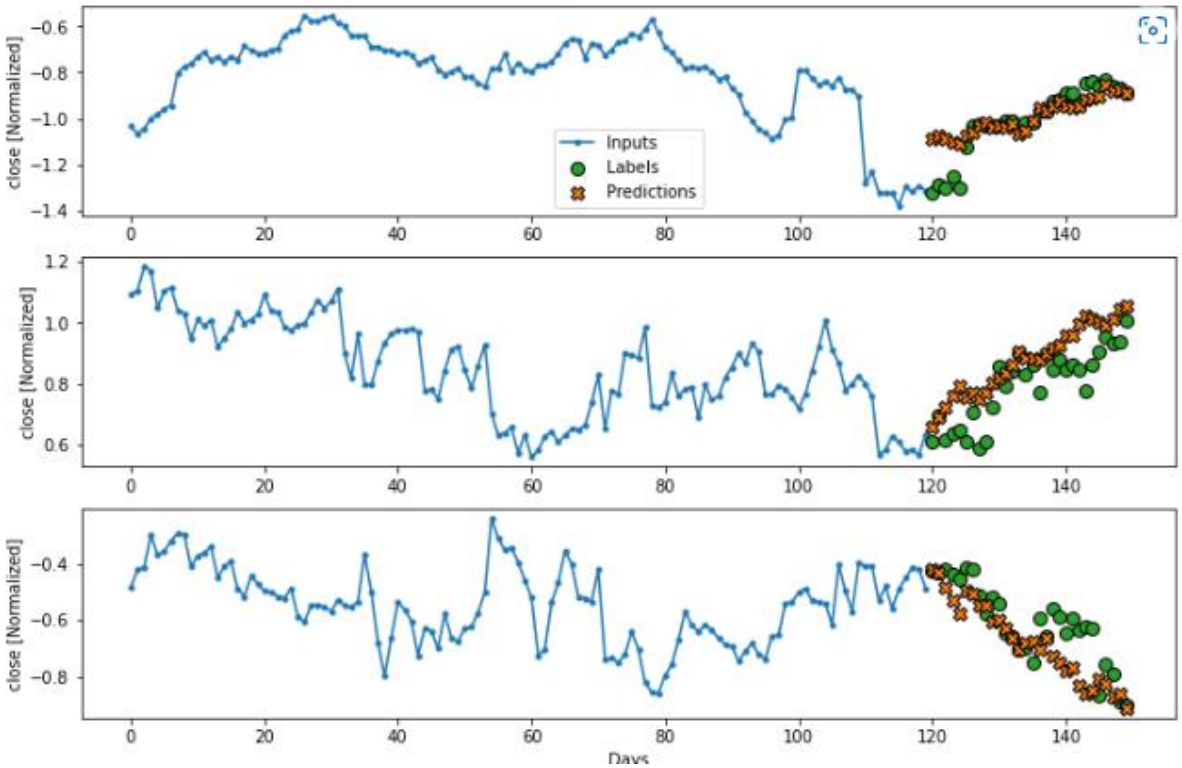
Model1 (LSTM_1): MAE = 2.65



Parameters: The initial/basic model

- Single LSTM layer (256 units) & Tanh Activation with glorot uniform weight distribution & an orthogonal recurrent initializer followed by one default Dense Layer (Linear activation, with glorot uniform weight distribution) & an Adam optimizer
- The parameter meanings are discussed in the ***LSTM Layer Parameters*** section.
- This model generalized moderately well, but was mainly a starting point for the adhoc model tweaking
- The main fallback of these predictions was that they often underestimated the “slope” of the labels

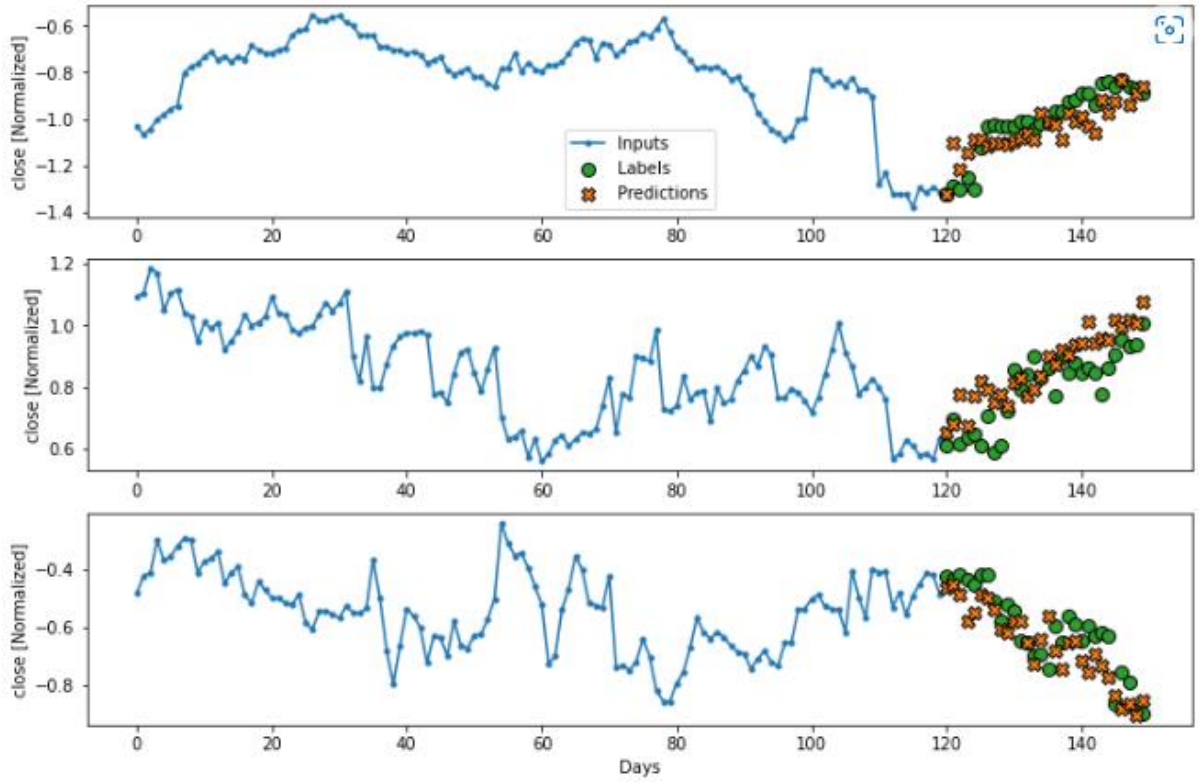
Model3 (LSTM_3): MAE = 3.29



Parameters:

- Single LSTM layer with 256 units, tanh activation, Variance Scaling weight distribution & an orthogonal recurrent initializer. Followed by a dense layer with a linear activation function & a Constant weight initializer. Followed by another dense layer with a linear activation function and variance scaling weight initializer. Adam optimizer.
- The parameter meanings are discussed in the ***LSTM Layer Parameters*** section.
- Including a dense layer with constant weight initializers seemed to add more linearity & decreased the variance of the predictions.
    - This tended to align with the purpose of this project.
        - (Broadly predicting price movement)
- Even though this had a lower MAE than Model 1, I would argue that this model has more practical use, due to the generalization of the predictions.

Model 7 (LSTM_7): MAE = 3.32



Parameters:

- Single LSTM layer with 256 units, tanh activation, Variance Scaling weight distribution & a truncated normal recurrent initializer. Followed by a normalization layer, one dense layer with a linear activation & Random Uniform weight initializer, another normalization layer, & finally, a second dense layer with linear activation & a variance scaling weight initializer.
- The parameter meanings are discussed in the ***LSTM Layer Parameters*** section.
- This model was somewhat of a medium between the previous two.
- We see that removing the “constant” initializer allowed for more variance in the predictions.
- Including normalization layers & an additional dense layer with a random initializer seemed to allow the predictions to change directionality more often.
    - This could be beneficial for longer term predictions. Since (usually), the price of a stock does not have any dramatic reversals in a 30 day window.
    - If the prediction window was around 6 months, which would include quarterly reports, a model that is more accepting to directional changes might be better suited.

Final Summary:

Overall, I would argue that for the purpose of 30-day pricing movement predictions, model 3 (second model reviewed) would be the best choice. This model accurately predicted price movement in broad strokes while retaining the ability to gauge the intensity of the movements. While the MAE was not necessarily the lowest, considering the goal of the project, model 3 was the highest achieving.

Other references:

https://keras.io
https://www.tensorflow.org
Long Short Term Memory Paper Hochriter/Schmidhuber (1997)
The Truncated Normal Distribution Burkardt (214)
https://medium.com
https://pages.cs.wisc.edu
https://colah.github.io
https://www.knime.com
StackExchange
GitHub
StackOverflow