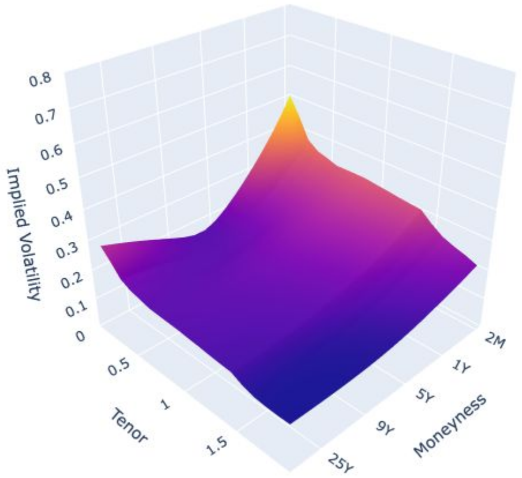
Wells Fargo Quantitative AI Hackathon

Presented By:
Team CSS
Chetan Reddy N (SHA2200554)
Sriram S M (SHA2200940)
K J Sashank (SHA2201660)

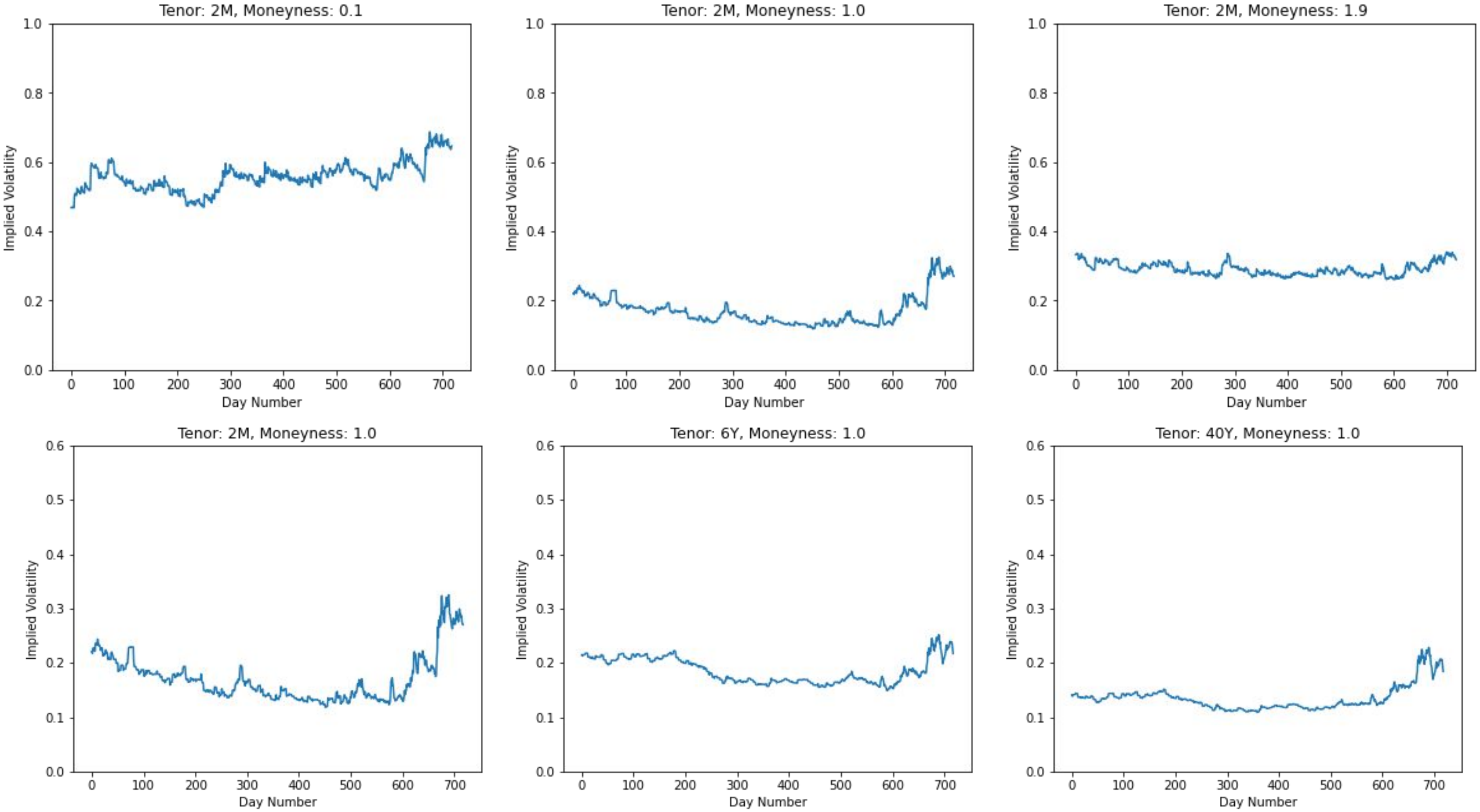
Some Insights from Exploratory Data Analysis

1. The implied volatility is lower when Strike price and Stock Price are nearer (moneyness is close to 1) and becomes higher when there is a higher difference between the strike price and stock price. This essentially explains the popular smile profile (Implied Volatility vs Moneyness).

2. There isn't any evident pattern between Implied Volatility and Tenor for a given time step and moneyness.

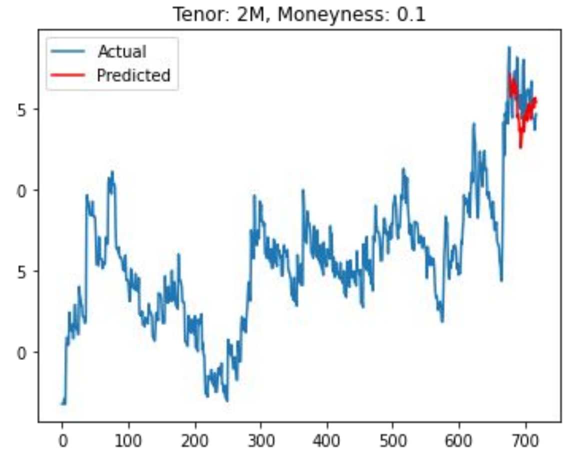


Independent Time Series Plots



Forecasting Volatility - ARIMA

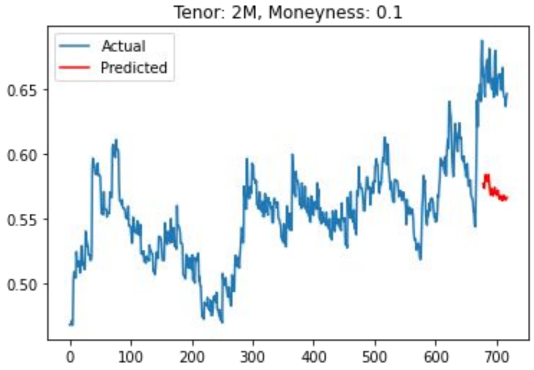
- In this approach, we used the popular ARIMA algorithm which stands for **AutoRegressive Integrated Moving Average.**
- In order to implement it, the volatility value for a given tenor and moneyness is considered to be independent from other tenor and moneyness values.
- Consequently, 1D time series for a given tenor and moneyness values was visualised and forecasted



Forecasting Volatility - ARIMA

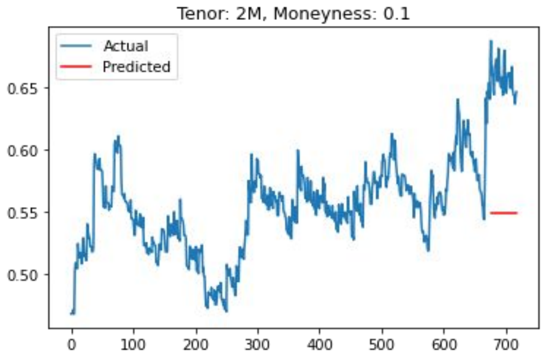
- AR : **AutoRegressive Model - parameter (p) [linear regression model]**
  - We use partial autocorrelation function and significance limit to find p
  - Y_{t-r} is the r^{th} lag or the volatility of the r^{th} day before the predicted date
  - We got the value of p to be around 30

$$Y_t = \alpha + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + .. + \beta_p Y_{t-p} + \epsilon_1$$



- MA : **Moving Average Model - parameter (q)**
  - Size of moving average window denoted by q
  - Et is the error generated from the MA wrt the original time series
  - We got the value of q to be 60

$$Y_t = \alpha + \epsilon_t + \phi_1 \epsilon_{t-1} + \phi_2 \epsilon_{t-2} + .. + \phi_q \epsilon_{t-q}$$

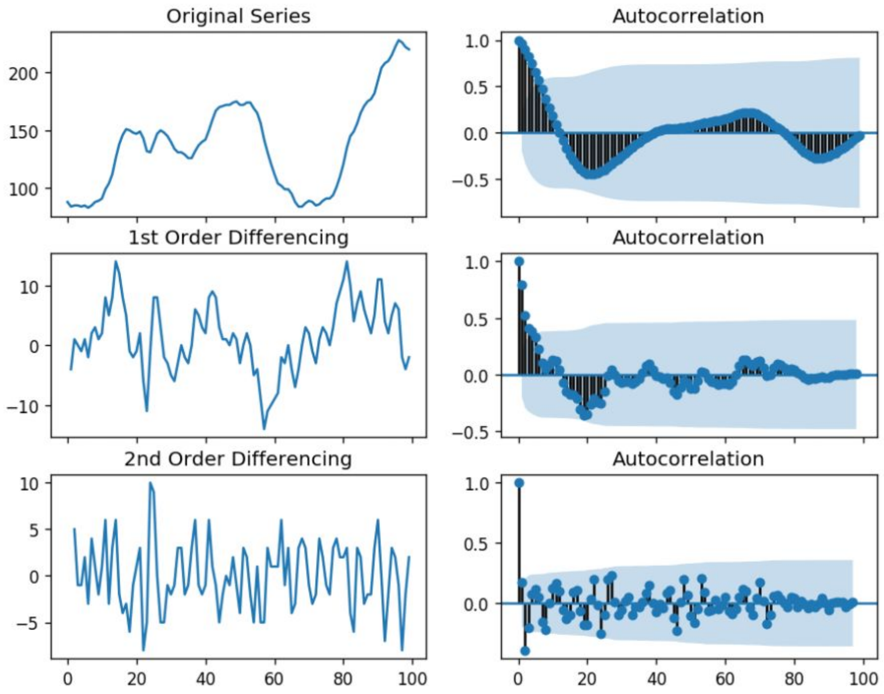


Forecasting Volatility - ARIMA

- **ARIMA (Integrated) Model - parameter(d)**
  - AR model works for a time series with constant trend (nor increasing nor decreasing) i.e the average should be a constant value
  - This is not possible in practical systems, as shown by our given problem statement
  - So we difference the time series shifted by one to find a constant series to apply ARMA on, we keep differencing the new series until we get a constant series with noise

$$Y_t = A_t - A_{t-1}$$

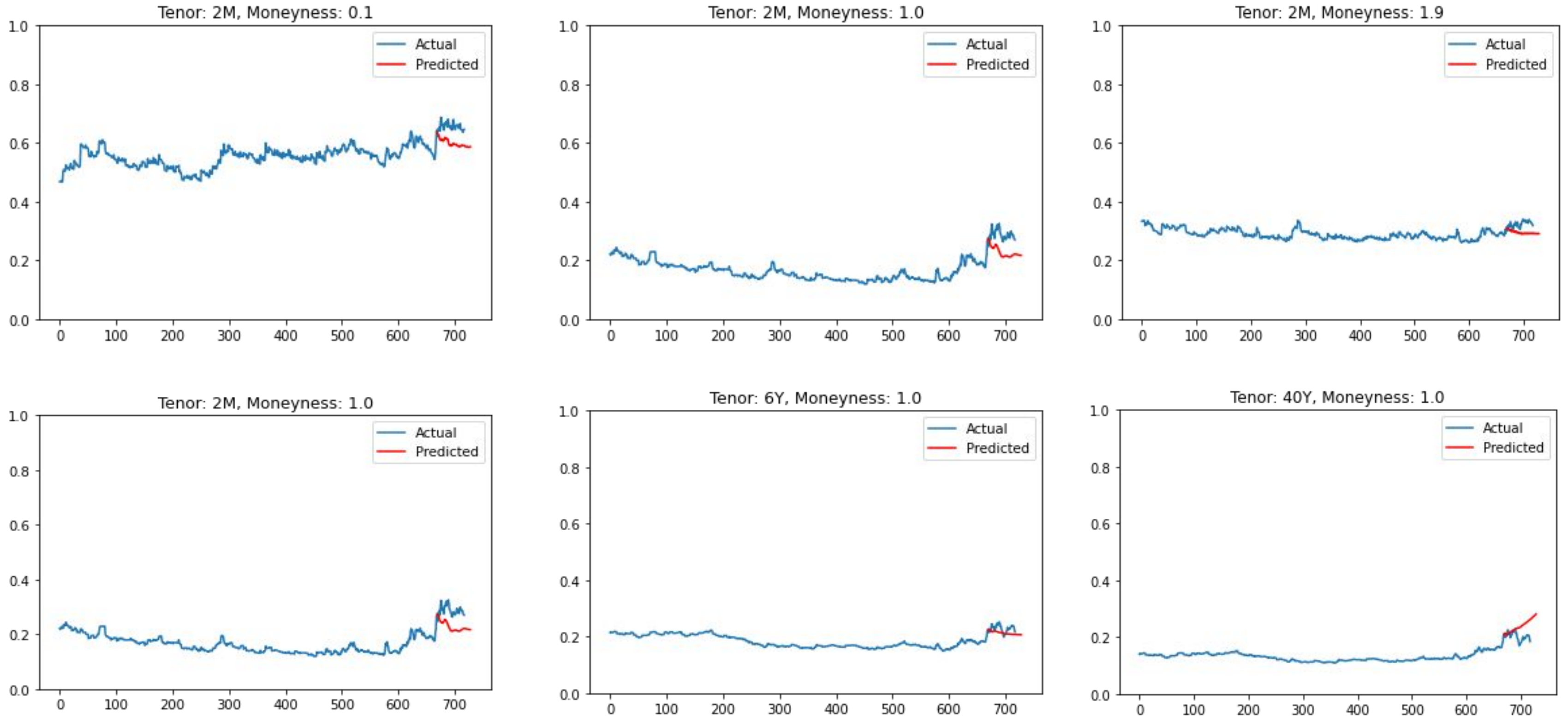
$$Y_t = \alpha + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + .. + \beta_p Y_{t-p} \epsilon_t + \phi_1 \epsilon_{t-1} + \phi_2 \epsilon_{t-2} + .. + \phi_q \epsilon_{t-q}$$



Order of Differencing

❖ With this model we got the RMSE of 0.033

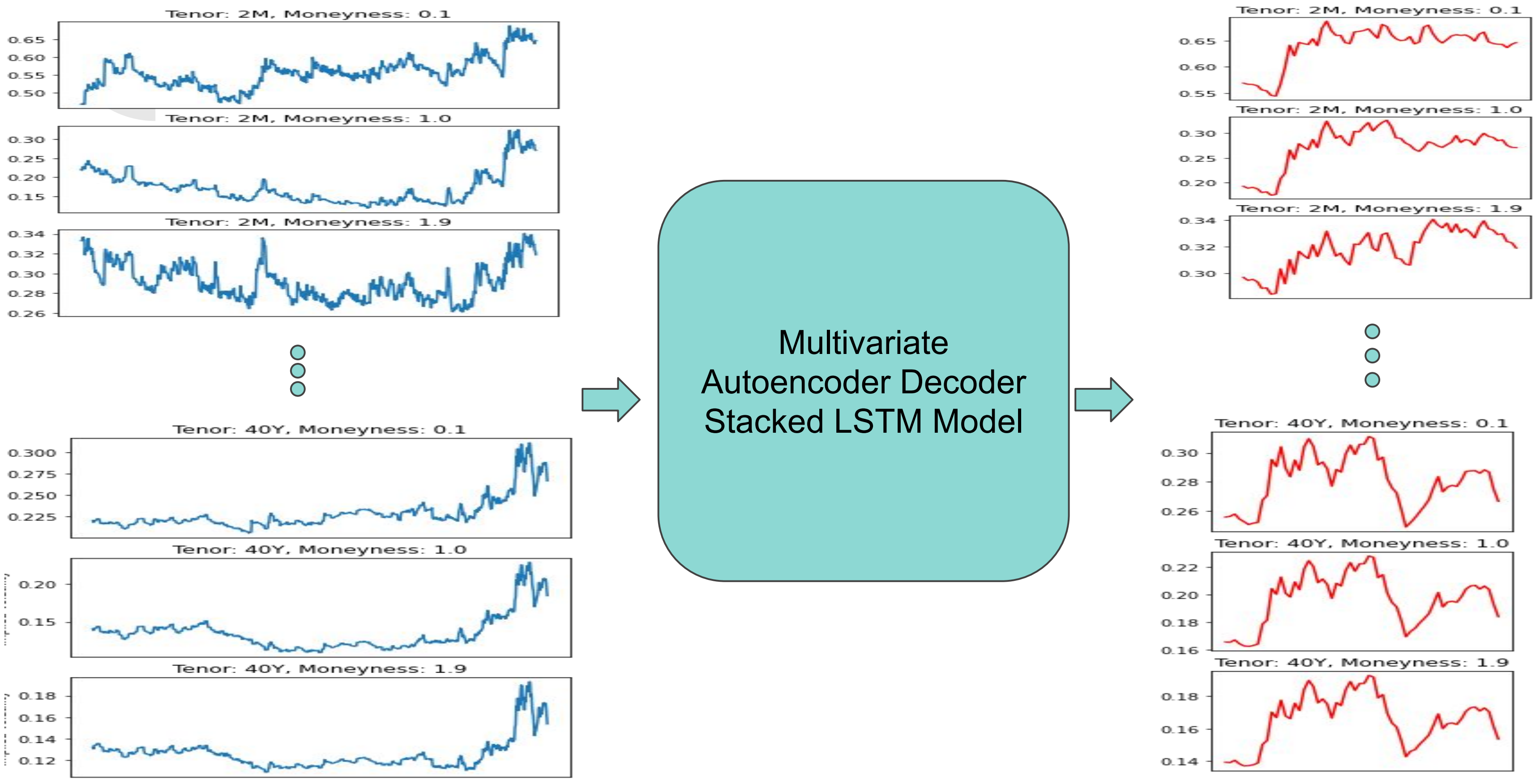
Predictions



Forecasting Volatility - Stacked LSTM

- In the previous approach, the individual points on the volatility surface was assumed to independent of the values of the surrounding points. However, this assumption did not have a strong justification. Therefore, we adopted a multivariate approach where the volatility series for each of the grid points (19x19=361) were assumed to be dependent and a multivariate time series forecasting was done.
- The algorithm used is a stacked LSTM with a autoencoder-decoder architecture.
- **An rmse value of 0.028 was obtained**
- The model can be improved with better hyperparameter tuning

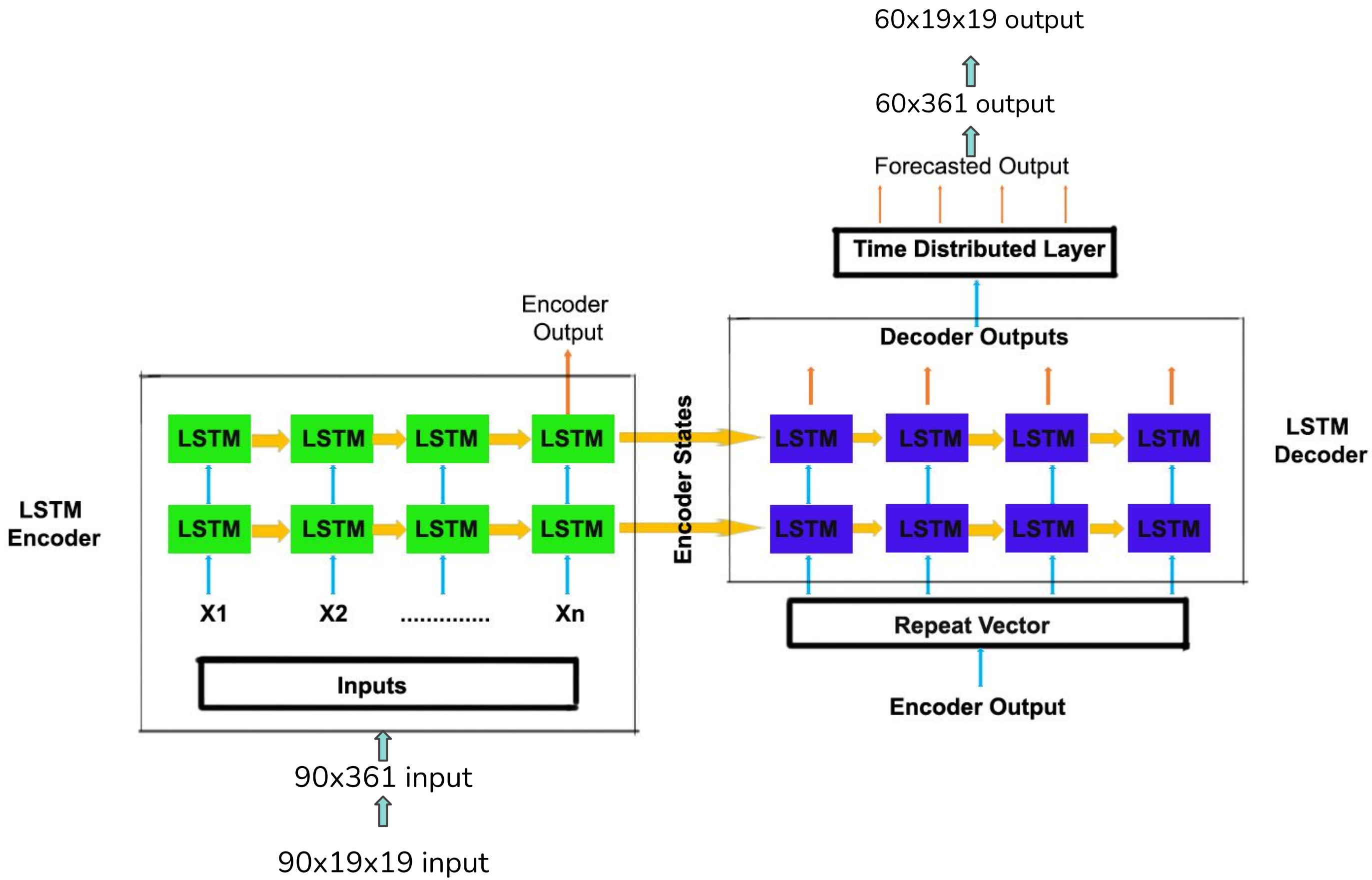
Forecasting Volatility - Stacked LSTM



381 time series plots input over 90 days (90x381)

381 Time Series Predictions for 60 days (60x381)

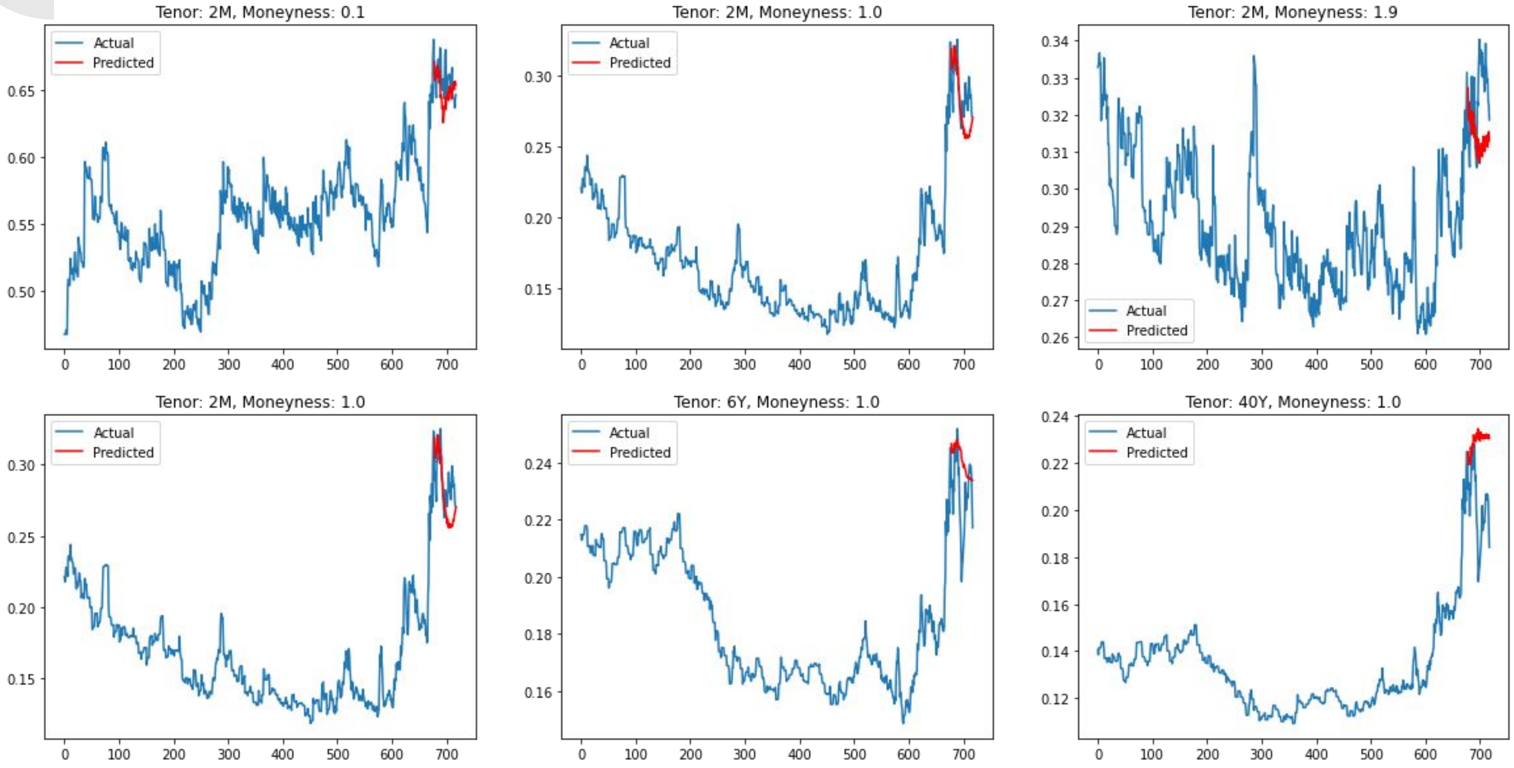
Multivariate Autoencoder Decoder Stacked LSTM Model



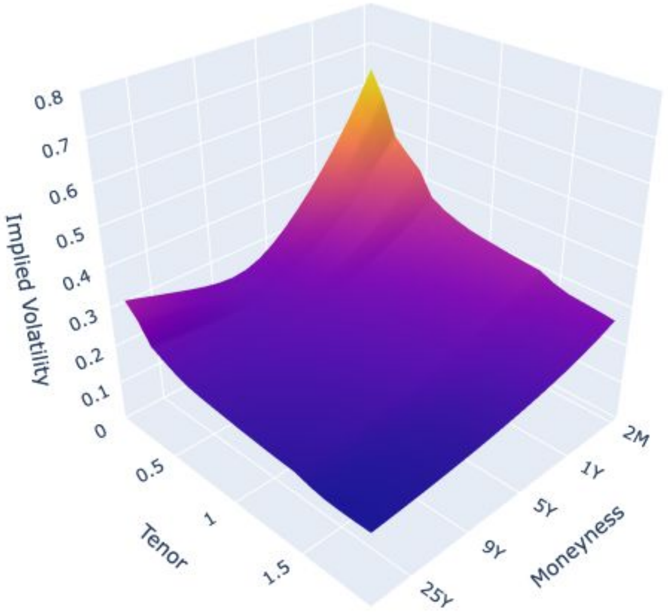
Model Architecture Code

```
lstm_output_dim = 100
#Encoder
encoder_inputs = tf.keras.layers.Input(shape=(n_past, n_features))
encoder_l1 = tf.keras.layers.LSTM(lstm_output_dim,return_sequences = True, return_state=True)
encoder_outputs1 = encoder_l1(encoder_inputs)
encoder_states1 = encoder_outputs1[1:]
encoder_l2 = tf.keras.layers.LSTM(lstm_output_dim, return_state=True)
encoder_outputs2 = encoder_l2(encoder_outputs1[0])
encoder_states2 = encoder_outputs2[1:]

#Decoder
decoder_inputs = tf.keras.layers.RepeatVector(n_future)(encoder_outputs2[0])

decoder_l1 = tf.keras.layers.LSTM(lstm_output_dim, return_sequences=True)(decoder_inputs,initial_state =
encoder_states1)
decoder_l2 = tf.keras.layers.LSTM(lstm_output_dim, return_sequences=True)(decoder_l1,initial_state = encoder_states2)
decoder_outputs2 = tf.keras.layers.TimeDistributed(tf.keras.layers.Dense(n_features))(decoder_l2)

model_e2d2 = tf.keras.models.Model(encoder_inputs,decoder_outputs2)

model_e2d2.summary()
```

LSTM Predictions



LSTM Predictions



Prediction on 10/12/2019

Thank You!