Kris Walsh

kmw2221

Columbia University Department of Economics

Master's Thesis First Draft

"Time Series Models in Foreign Exchange"

Advisor: Professor Steven Ho

November 13, 2020

Outline

I. Introduction

II. Literature Review

   A. Linear Models

   B. Neural Networks

III. Empirical Study

   A. Introduction

   B. Data

   C. ARIMA and random walk results

   D. Neural network results

IV. Conclusion

V. Appendices

   A. Appendix A: Figures and Results

   B. Appendix B: Bibliography

   C. Appendix C: Python Code

I. Introduction

Prediction of foreign exchange (FX) rates has a history that goes back centuries, though most of what we would consider modern currency valuation has been around for only 50 years. In 1971, U.S. President Nixon severed the dollar's direct convertibility into gold, beginning a new age of fiat money. This period would coincide with the development of econometric methods that allowed researchers to model FX markets deterministically. Rapid advancement in computing power quickly followed, opening the door for sophisticated machine learning techniques to explore seemingly limitless quantities of exchange rate time series data. Furthermore, the volume of exchange rate market turnover expanded exponentially, from barely anything in 1986 to more than $6 trillion in 2019 (Bank of International Settlements, 2019). (Figure 1).

Despite the rapid expansion of both the FX market and capabilities in analyzing it, a model that can successfully predict outcomes and create abnormal profit still eludes us. Debate about whether such prediction is theoretically possible remains vigorous in the literature. Central to this question is the efficient market hypothesis, which states that the market price reflects all current information (Fama, 1970). It follows that the history of the exchange rate cannot be used to increase investors' expected profits, and that the price of the currency is following a random walk. Even Fama has stated that the extreme version of this hypothesis is surely false (Fama, 1991), and that market randomness likely exists on a continuum. It therefore remains of academic interest to apply various time series models to FX markets, and to explore their power to detect patterns in the data, even if those patters are nonlinear or even unknown entirely.

The paper has two components. The first is a review of literature in the field of foreign exchange rate prediction, beginning with Meese and Rogoff's watershed 1983 article which sparked the debate on whether or not FX follows a random walk. This is followed by a review of various models that have been developed since, and a review of how they have performed. Of particular interest will be the application of machine learning, specifically neural networks which have shown promise in outperforming random walks and linear models in predicting FX.

The second component of this paper is an empirical study which compares the predictive performance of a small group of models in predicting six major currency pairs. The data set is

daily FX rates of GBP, EUR, JPY, CAD, MXN and ZAR from 1-Jan-2015 to 30-Jun-2020, for a total of 1434 observations. This study will restrict itself to the univariate case, and the performance measure will be MSE (mean squared error) relative to the actual path of the currency. The models selected are a baseline random walk, the autoregressive integrated moving average (ARIMA) model that best fits each currency pair, and a long short term memory (LSTM) neural network model that will allow for estimation without the assumption of linearity.

II. Literature Review

Linear Models

In their paper, Meese and Rogoff developed the puzzle which states that there is no better economic model for floating exchange rate forecasting than the simple random walk (Meese and Rogoff, 1983). This is a puzzle because to bring a wide range of predictive variables into a structural model, including macroeconomic fundamentals and market technicals, seemingly should provide some analytic advantage that leads to profitable investments. The authors considered a wide range of models: a univariate autoregression (ARIMA), a multivariate autoregression (VAR), the Frenkel-Bilson flexible-price and Dornbush-Frankel sticky-price monetary models, and the Hooper-Morton model which extends the D-F model with the inclusion of the current account balance as a variable. The Meese and Rogoff result that none of these models performed any better than a random walk established the null hypothesis that the FX market is itself a random walk.

Meese and Rogoff investigated only linear models, which raises the possibility that the foreign exchange market is indeed deterministic but has a nonlinear form (Alvarez-Diaz, 2008). ARIMA and VAR models are linear by their design, with the outcome variable determined by lags of itself and of the error terms. Sticky and flexible monetary price models were developed in the 1970s and apply ordinary least squares to output, price and money supply variables (Rosenberg, 1996). The authors considered these to be the most promising of linear models, compared to more simple valuations constructed by purchasing power parity (PPP) or interest rate differentials. Still all of them failed against the random walk.

Hope was not entirely lost for linear models. In the following decade John Williamson developed his fundamental equilibrium exchange rate (FEER) model which is designed to value

a currency based on deviations from optimal conditions, both internal as measured by output and unemployment, and external as measured by the balance of payments (Williamson, 1985). A major limitation of this model, even according to Williamson, is that it requires a large amount of subjective assumptions, from how to define external balance to estimating a country's noninflationary and full employment level of output. This means that there are a large number of FEER estimates that are arguably plausible, and to this day this measure is popular among investment professions as a baseline for a currency's fair value.

Several years later Hamid Faruqee introduced the behavioral equilibrium exchange rate (BEER) model which similar to the FEER model uses internal and external macroeconomic conditions to value the exchange rate (Faruqee, 1994). However, the BEER approach uses a stock equilibrium condition to measure the external balance directly, rather than estimating savings and investment as is done with the FEER approach. The BEER model has the advantage of breaking down the total exchange rate misalignment into the effect of transitory factors, random disturbances and macro imbalances (Macdonald and Clark, 1998). BEER models also remain popular, and a recent study showed that among a wide class of fundamental and theoretical linear models, a first difference specification BEER model was the only one considered which outperformed the random walk, and it did so by a wide margin, using a mean squared error and direction of change criteria (Cheung, 2018). Other models that did not perform well in this study included a Taylor rule model, valuations with real rates and yield curve sloped, a sticky price model with risk and liquidity factors, and a PPP model.

Neural Networks

The concept of building a mathematical model that resembles a biological neuron was developed by psychiatrists at the University of Illinois in 1943 (McCulloch and Pitts, 1943). Their basic neuron model takes in multiple inputs, performs an operation on them, and delivers an output (Figure 2). Structures can be formed with multiple layers of neurons that are either parallel or hierarchical, allowing output variables to be determined by a vast range of combinations of input variables. Most importantly, this structure allows for backpropagation, in which the model corrects itself based on its own performance. This mimics how biological

neurons become better and faster at performing tasks with repetition. All of these traits characterize a neural network.

The application of McCulloch and Pitts' neural model to financial markets began around 1990, coinciding with an expanding financial sector and rapidly growing computational power. Early studies noted that neural networks were useful in time series forecasting, since they do not require any assumption on functional form (Wong, 1990). Wong ran his neural network on a primitive Apple Macintosh II with a 20 megabyte drive, and found his model better predicted the stock market than a linear regression.

Of various mathematical tools available to address nonlinear forms, the neural network is particularly well suited to financial markets, including foreign exchange (Binner, 2005). The network has input nodes, hidden nodes and an output node (Figure 3). The hidden nodes provide the non-linear map (without them you are left with simply a linear model). An important trade off in designing a neural network is deciding how many layers of hidden nodes to include. If you have too many, the model will likely result in overfitting. With too few nodes, the network cannot learn. There are still more parameters to consider, including weights for each input function, the number of input variables, the combination of input variables, the types of activation functions for the hidden and outer layers, the value of the learning and momentum rates, and the amount of training. Similar to the FEER structural model, this wide array of choices for model selection leads to a high subjectivity factor in neural networks.

It has been shown that a three layer neural network with only one hidden layer can approximate any function to any degree of accuracy (Hornik, 1989). This result suggests that when it comes to financial market prediction, a simpler model is a better starting point than a more complicated one. If the model has too many hidden layers, it may overtrain itself when finding unique and highly profitable situations in the training set. The least complex networks turns out to be the most profitable because it will not overspecialize, and will seek a more general solution (Galeshchuk, 2016).

A particular form of neural network that has shown promise in a financial prediction context is the long short-term memory network (LSTM) (Fischer, 2018). A standard neural network may suffer from the vanishing gradient problem in a time series context (Yang, 2017).

The LSTM network has a memory unit that can store aged information as well as drop it at any time, a structure that allows for an unspecified gap length between important events and therefore provides a solution to the vanishing gradient. One recent study found that LSTM outperforms a standard neural network, a random forest algorithm, and a logistic regression in financial prediction (Fischer, 2018). The authors attribute the success of the LSTM to its ability to extract information from a time series that is particularly noisy, as is the case with FX.

An even more recent study compared four different types of neural networks (LSTM, Gated Recurrent Unit (GRU), Feedforward Neural Networks (FNN), and Simple Recurrent Neural Network (SRNN) (Dautel, 2020). The accuracy of all four models was better than that of a random walk, showing promise for the suitability of neural networks for FX prediction in general, but they also noted it is difficult to choose the right parameters for these models, and they did not get very different results among the four structures. They also echo earlier authors when noting that a simple network will likely perform better than a more complicated one, particularly with regard to trading profits.

Literature Review Summary

Neural networks have shown promise in exchange rate prediction compared to random walks. Both structural and time series linear models have shown lackluster performance, and neural networks offer the advantage of not insisting upon a linear structural form. Long short-term memory (LSTM) models are particularly well suited to chaotic financial time series. An LSTM with a minimalistic structure may be one of the most promising tools in discovering the deterministic nature of exchange rates. To measure the performance of an LSTM model, its mean squared error should be compared against that of a random walk. If it outperforms the random walk model in a meaningful way, the conclusion of Meese and Rogoff may not be correct, and it may indeed be possible to create abnormal profits in FX markets.

III. Empirical Study

Introduction

The empirical goal of this paper is to compare the performance of a group of univariate exchange rate predictive models: the random walk baseline, the linear autoregressive integrated moving average time series, and the LSTM neural network which will allow for unspecified

functional form. Only the univariate case is considered; the time series history of each currency will be the only input in determining the forecast.

Data

1434 observations daily close observations are collected on six currency pairs, all relative the U.S. dollar (EUR, GBP, JPY, CAD, MXN and ZAR). The inclusion of two emerging market currencies will allow us to test if higher transaction costs and a larger standard deviation in returns has any effect on the predictive power of a univariate model. Both the Mexican peso and South African rand are fully deliverable and can be easily traded, though will less ideal liquidity conditions compared to the other four currencies. The date range for the data set 1-Jan-2015 to 30-Jun-2020. The data is downloaded from the Yahoo Finance API via Python Script (Appendix C).

ARIMA model results

The linear ARIMA model regresses the exchange rate value on its own past values, as well as lags of the error terms in the regression. This is one of the models that did not outperform the random walk in Meese and Rogoff's study. Before an ARIMA model can be estimated, its parameters must be selected. The Akaike Information Criterion (AIC) is a measure that gives the optimal parameters, balancing both the maximum predictive power of the lags and the efficiency of the model.

In this study, a range ARIMA models are considered for each currency pairs, with any range of y(t-1) lags from 0 to 5, and any range of ϵ(t-1) from 0 to 5. In ARIMA models these parameters are identified as p and q, respectively. A grid search is performed over a reasonable range of permutations to find the ARIMA model that delivers the lowest AIC value. This model is considered the "best-fit" ARIMA model, and should be used when comparing the performance of the ARIMA model to any other model.

There is a wide precedent in the literature for using a grid search over a subset of candidate numbers of lags, and choosing the model with the lower AIC. Sk takes 1284 daily observations from 2010 to 2015 for six major currencies (USD, GBP, SGD, NZD, CHF and JPY, all relative to AUD), and grid searches across p and q values of 0, 1 and 2 to identify the best model. He assumes a d value of 1, and finds a different optimal combination of parameters for

each currency's ARIMA model (Sk, 2015). Bissoondeeal takes a much larger set of 5661 daily observations of AUD and GBP from 1986 to 2007, and scans across a wider range of p and q values from 0 to 5 in each case. He also assumes a d value of 1, and finds that ARIMA (5, 1, 2) best fits AUD, and (5, 1, 5) best fits GBP (Bissoondeeal, 2008). Both authors compared the performance of ARIMA models to neural networks, and got conflicting results. Sk found that ARIMA models outperform his neural network, while Bissoondeaal reached the opposite conclusion.

The other parameter to be set in an ARIMA model is the number of differences applied to the y variable before the regression, identified as d. In the context of time series analysis, the series must be stationary before statistical analysis can be done. The time series of a financial asset, including an exchange rate, is highly unlikely to be stationary. To be sure, we run an Augmented Dickey-Fuller (ADF) test on each of the six series to determine if it is stationary. With a p-value limit of 0.01, we reject the null hypothesis that the series is stationary in five out of the six cases. We cannot reject the null hypothesis in the case of the Canadian dollar. Therefore, the differencing parameter d in the ARIMA model must be at least 1, except in the case of the Canadian dollar where it may be 0. It is possible to difference more than once in an ARIMA model, so a grid search is done across potential d value from 1 to 3, and from 0 to 3 in the case fo the Canadian dollar.

It should be noted that an ARIMA model with the p, d, and q parameters of (0, 1, 0) is identical to a random walk. Zero values for p and q indicate the model has no autoregressive terms, neither of itself nor of the errors. The 1 in the d term implies that $y(t) = y(t-1)$, or that the best prediction of a currency's value at the end of any given day is its previous close. This "naïve" forecast is our random walk, and serves as the baseline against the performance of the other models considered in this study. The MSE for the random walk model is therefore reported as that of an ARIMA (0, 1, 0) model.

The best-fit ARIMA model as identified by the grid search varies for each currency pair. In the case of EUR, (1, 1, 1) is the optimal fit, the most parsimonious among the six. JPY, CAD, and ZAR are optimally fit by (2, 1, 2), meaning one differencing of the time series, and the inclusion of two lags of the dependent variable and two lags of the error term in the regression.

GBP and MXN are both optimally fit by a model that includes yet another autogressive term, y(t-3).

As the table of results shows, the best-fit univariate ARIMA model for each currency pair fails to outperform the random walk. This is despite the ARIMA parameters having been selected over a wide range of permutations of possible values. This implies that will the assumption of a linear form, a univariate autoregression does not have any predictive power for the path of a currency.

Neural Network Results

The LSTM model requires us to specify the number of previous observations to include in its estimation. With the ARIMA model, a grid search for ideal number of autoregressive terms ("p" values) was never any larger than 3, while the LSTM will be set to 10. Therefore the predictive power of an individual observation can be fed into the the model's prediction up to two weeks in advance (ten business days).

Another key parameter the LSTM asks for is the number of training epochs to be applied to the training data set (the first 80% of observations, as was the case with ARIMA and random walk). With all six currency pairs, the loss function approaches zero quickly, within 2-3 epochs, so the epoch number is set to 5 for all six currency pairs. Attempting larger epoch sizes increases computational time significantly and does not improve the results.

Adjusting the batch size does not seem to affect the results much either, so we leave the batch size set to 1. That is, the training set will be treated as a single entity with the model trained on the entire training set with each epoch.

Appendix A: Figures

Figure 1: Global Foreign Exchange Market Turnover, 1986-2019

Source: Bank of International Settlements, 2019

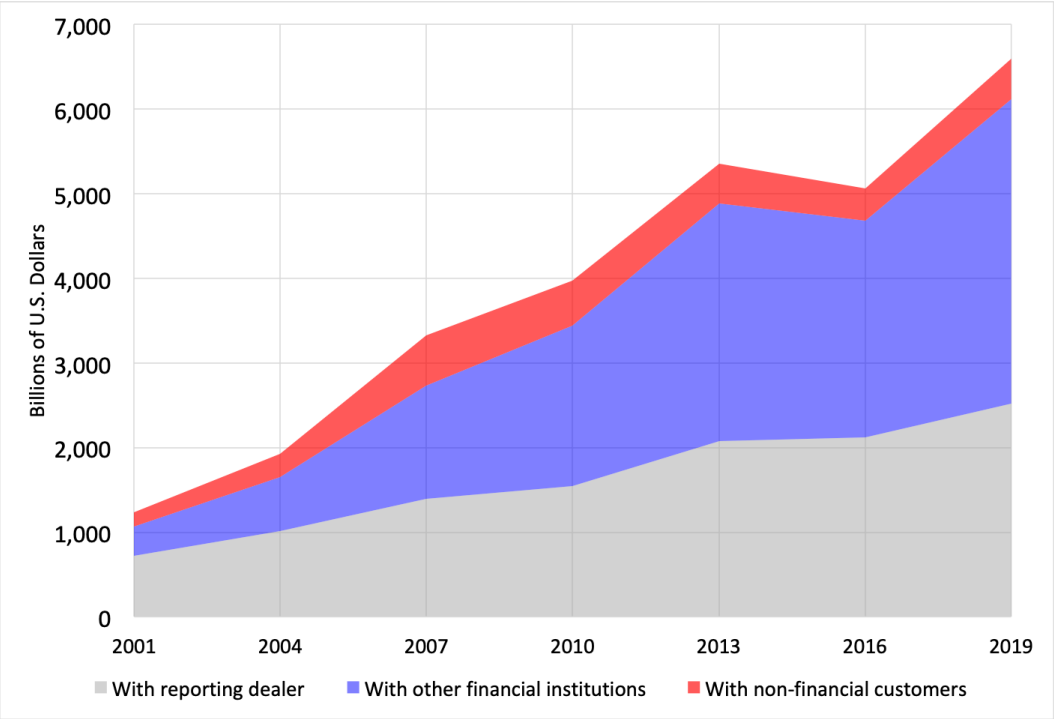


Figure 2: Data Set (6 Currency Pairs, Daily Observations)

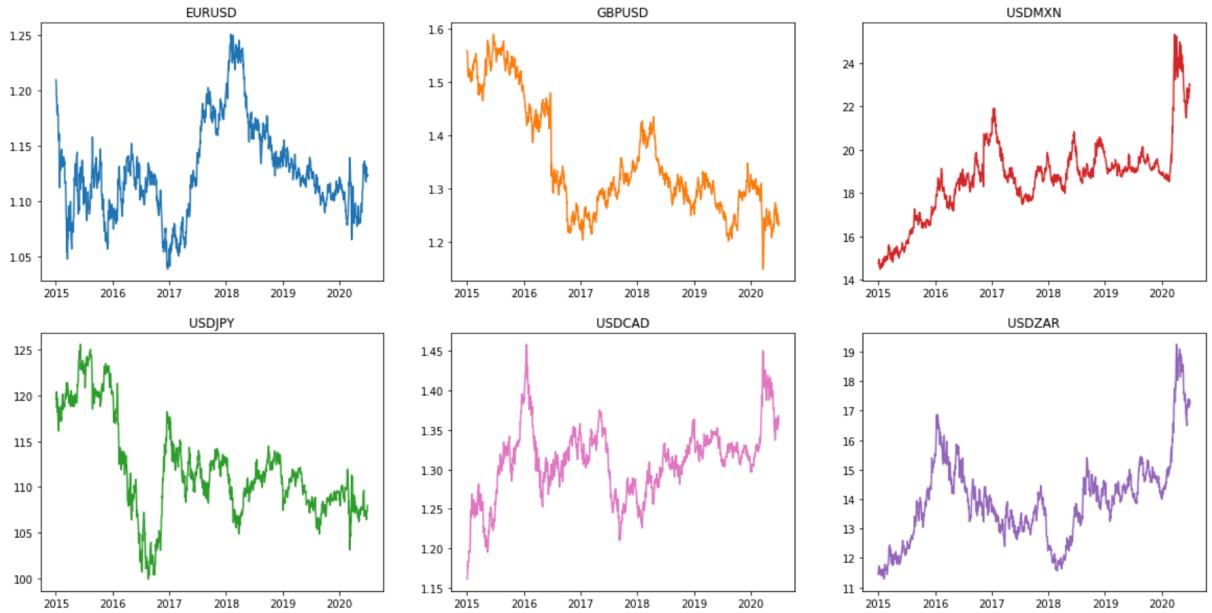


Figure 3: The McCulloch-Pitts Neuron (First Mathematical Model of a Biological Neuron)

Source: McCulloch and Pitts, 1943

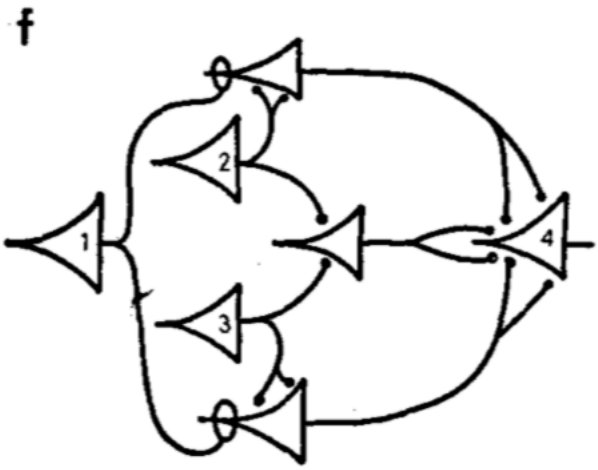


Figure 4: Structure of A Long Short-Term Memory Cell

Source: Fischer, 2018

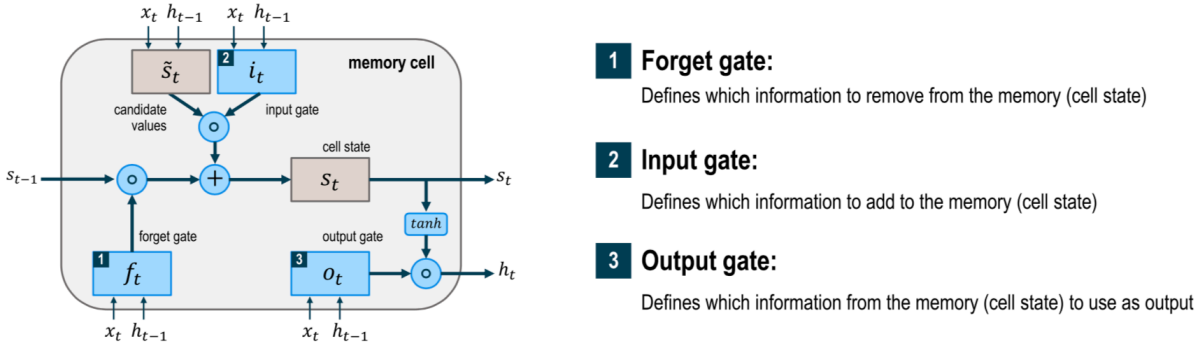


Figure 5: Results Table (Mean Squared Error for each model)

Results Table (MSE = Mean Squared Error)

	ADF Test	Best ARIMA	AIC	MSE ARIMA	MSE RW	MSE LSTM NN
Currency	(p-value)					
EUR	0.033	(0,1,1)	-8376	0.0002	0.0002	2*e^-5
GBP	0.389	(3,1,1)	-7679	0.0027	0.0027	0.0001
JPY	0.163	(2,1,2)	2131	2.32	2.32	0.311
CAD	0.005	(1,1,0)	-8275	0.0017	0.0017	7*e^-5
MXN	0.334	(2,1,2)	-1120	3.65	3.65	0.272
ZAR	0.478	(0,1,1)	-1162	2.24	2.25	0.094

Figure 6: Random Walk Results (GBP and MXN shown)

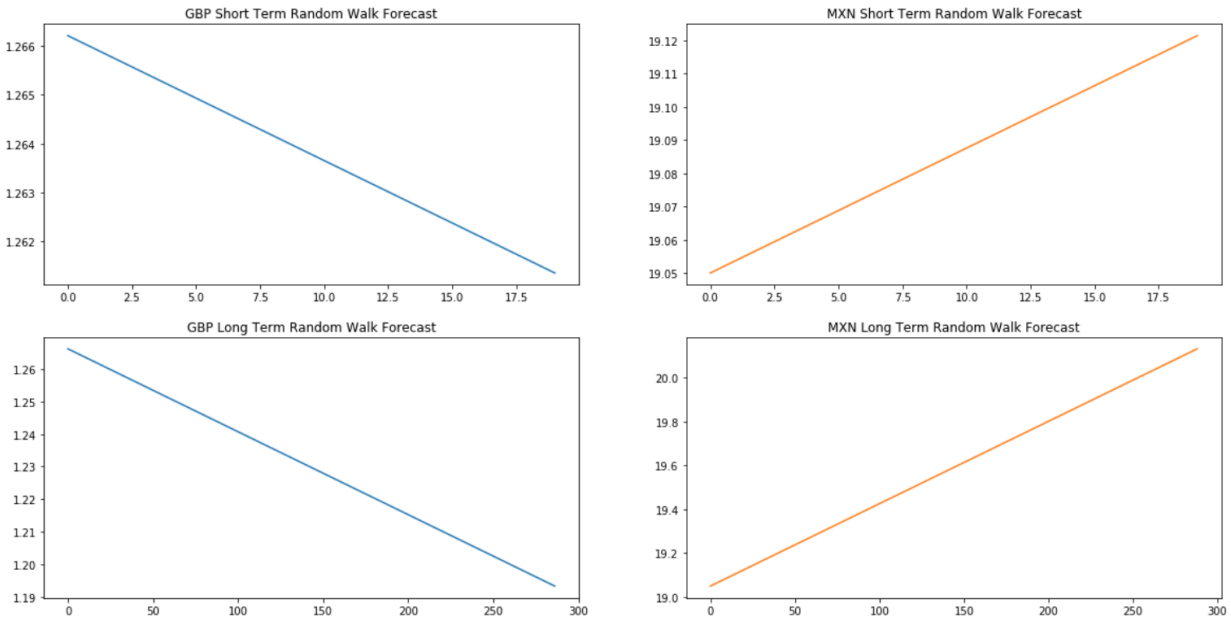


Figure 7: Best-Fit ARIMA Results (GBP and MXN shown)

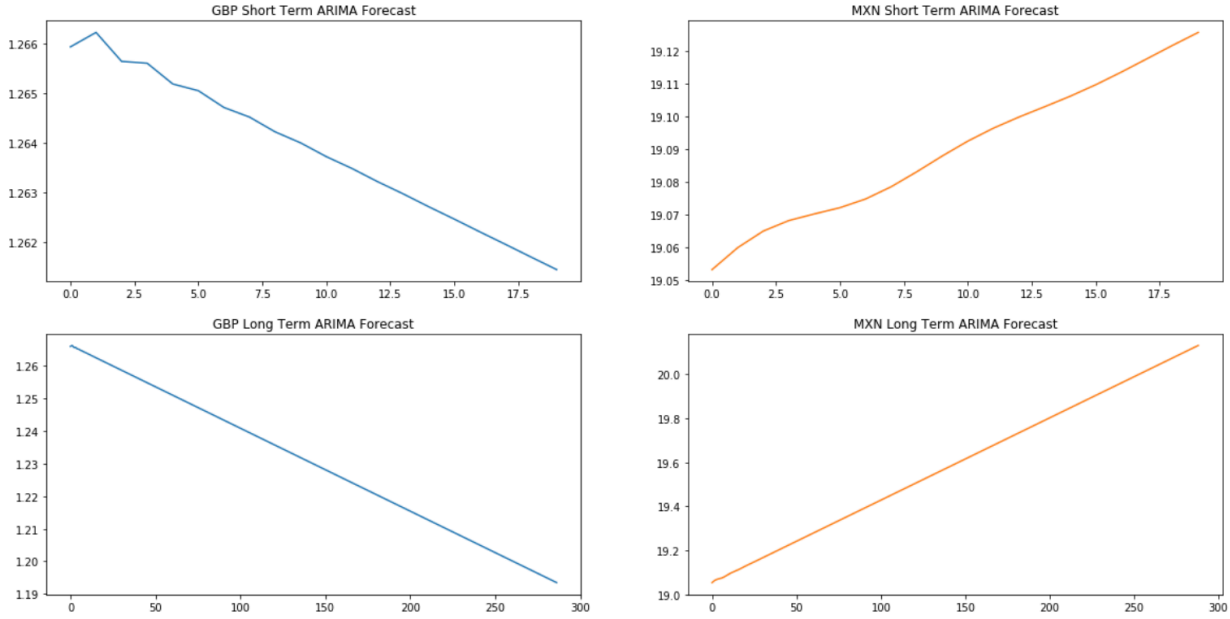


Figure 8: LSTM Neural Network Result (GBPUSD)

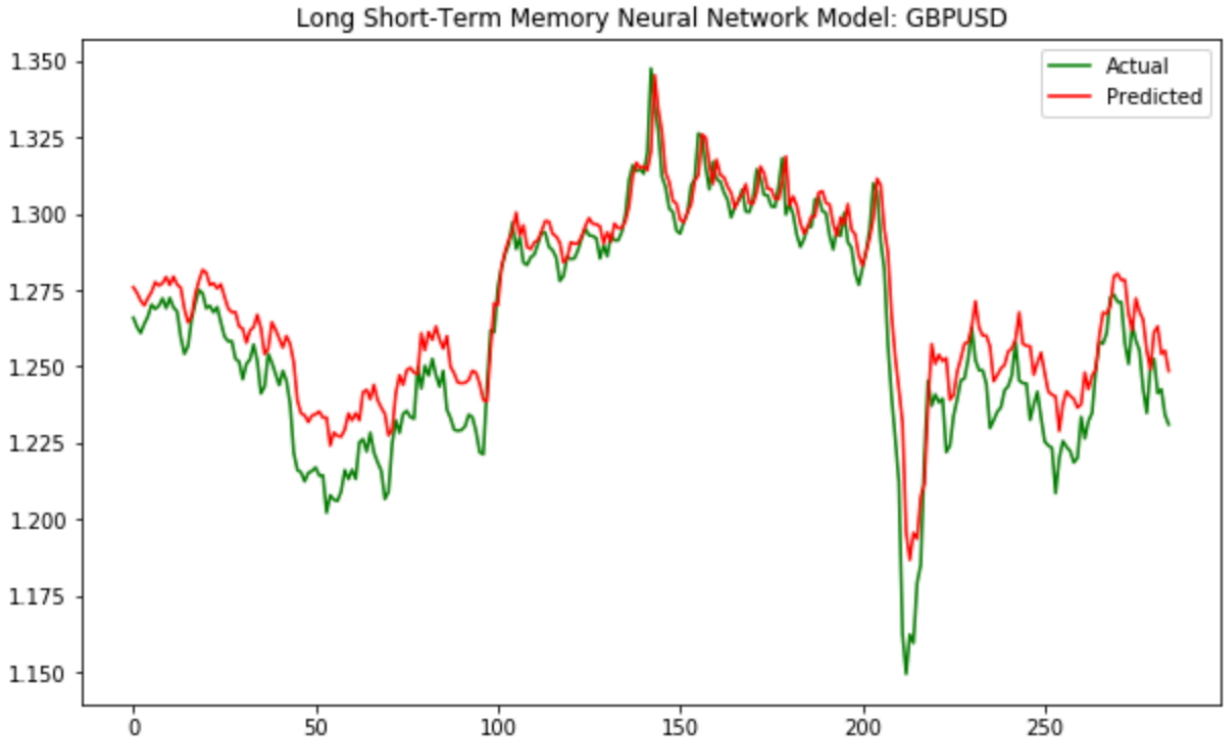


Figure 9: LSTM Neural Network Result (EURUSD)

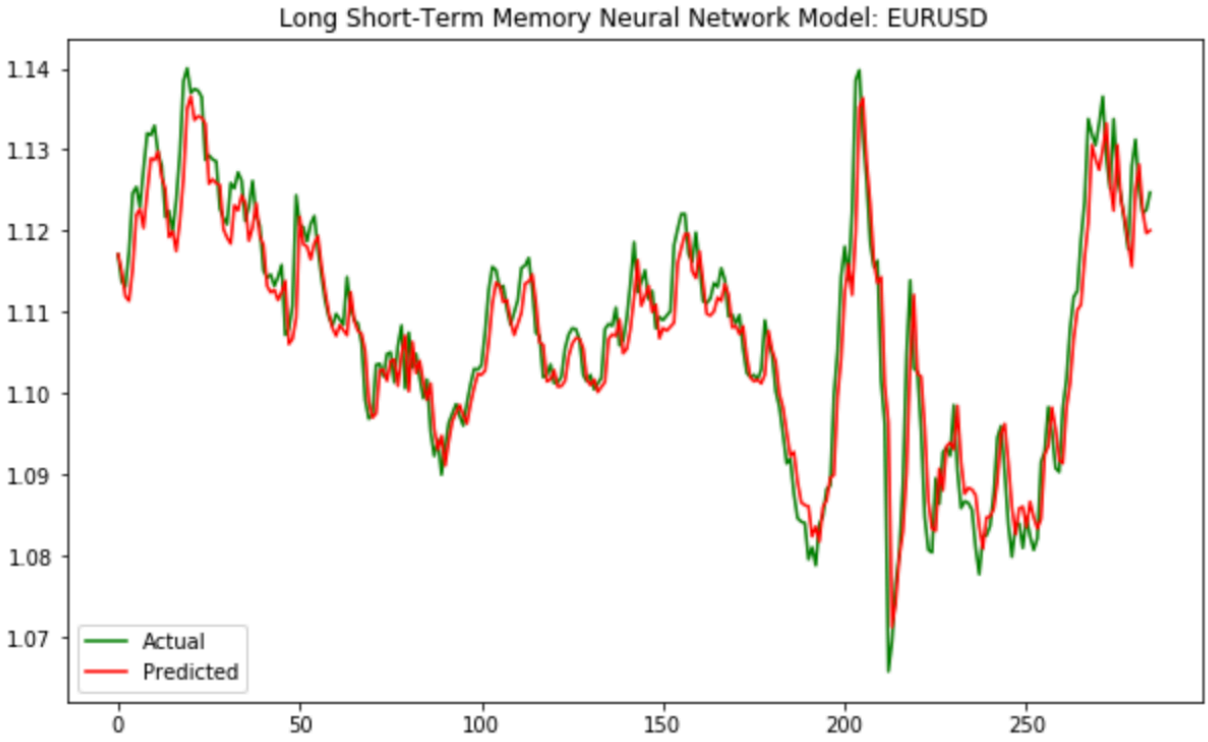


Figure 10: LSTM Neural Network Result (USDJPY)

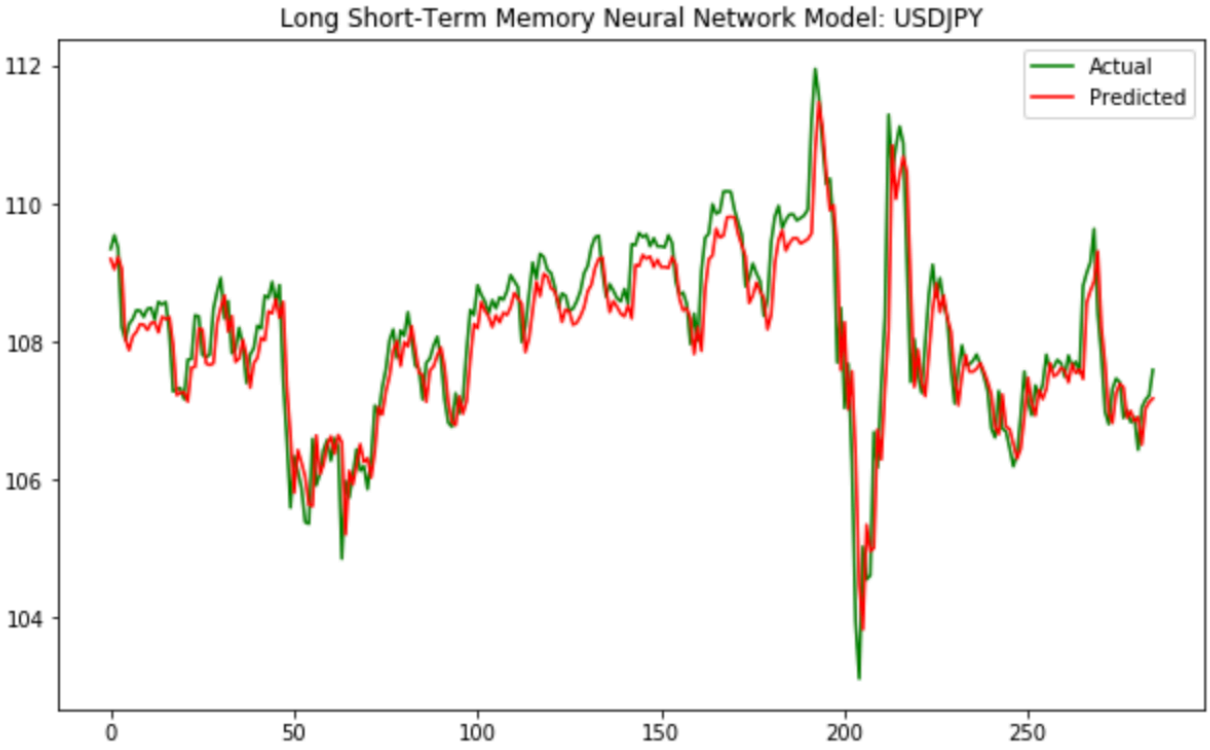


Figure 11: LSTM Neural Network Result (USDCAD)

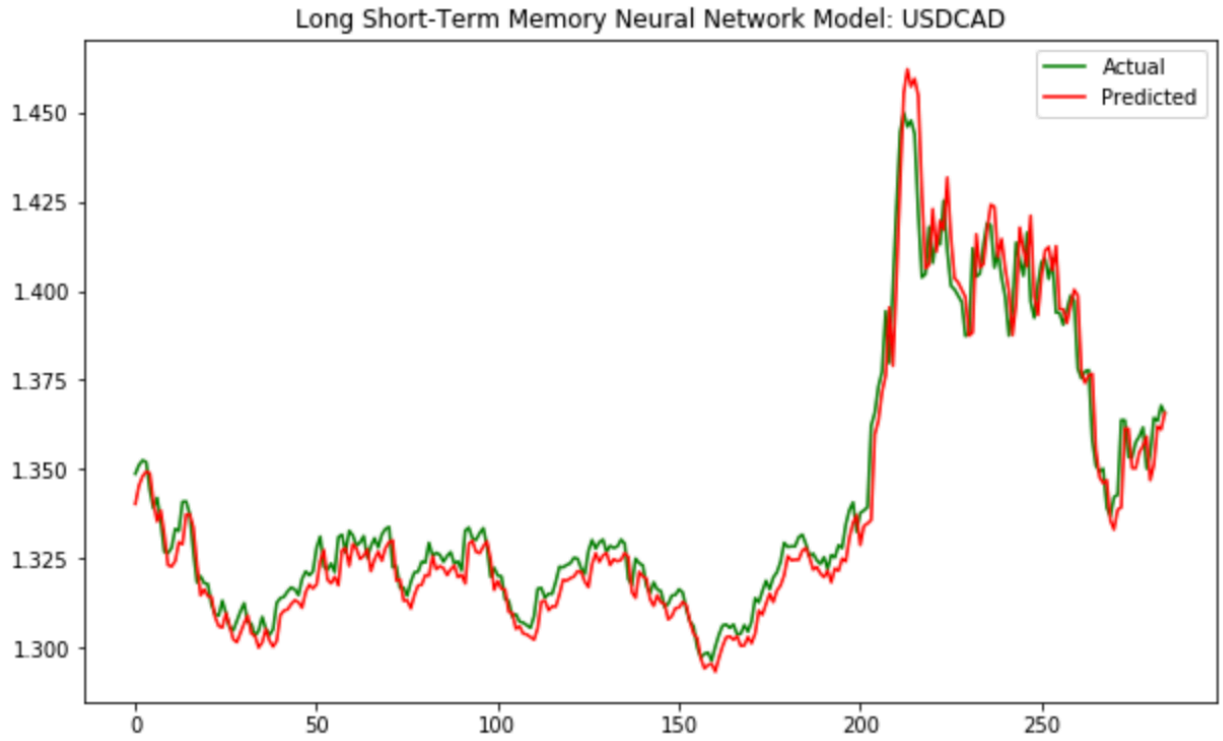


Figure 12: LSTM Neural Network Result (USDMXN)

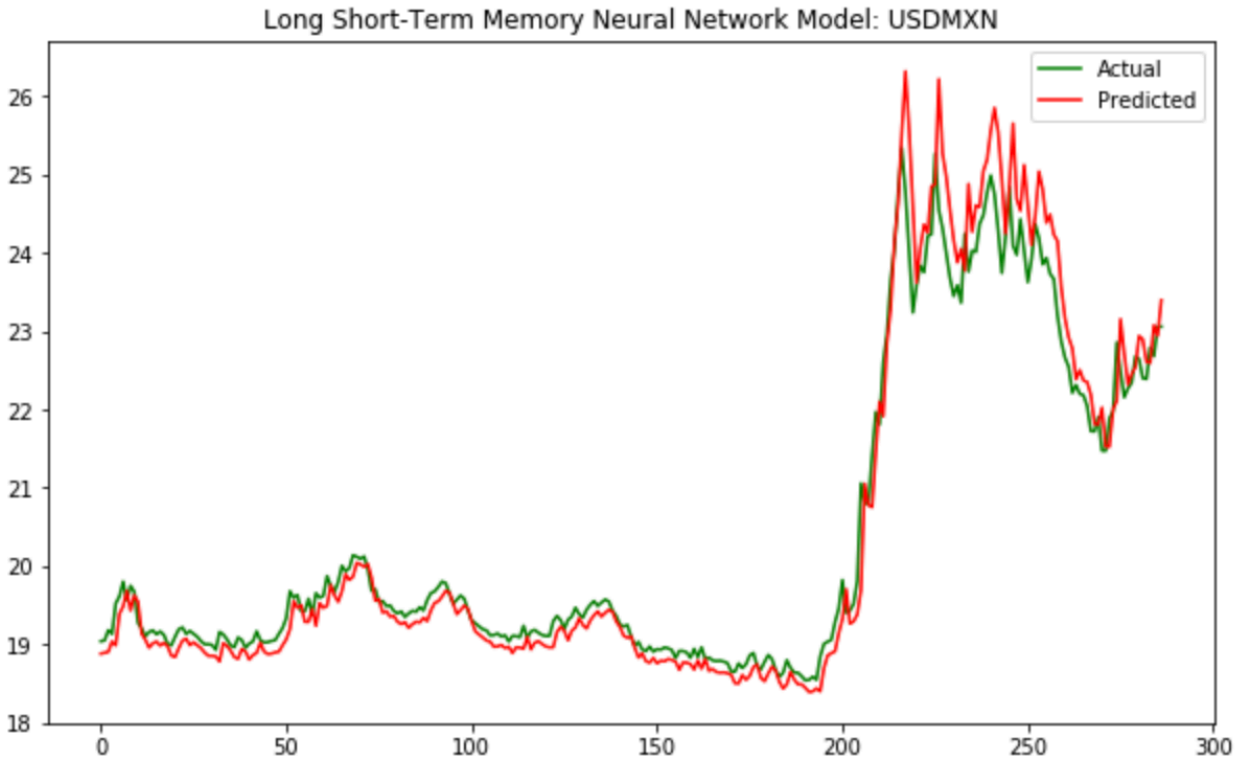
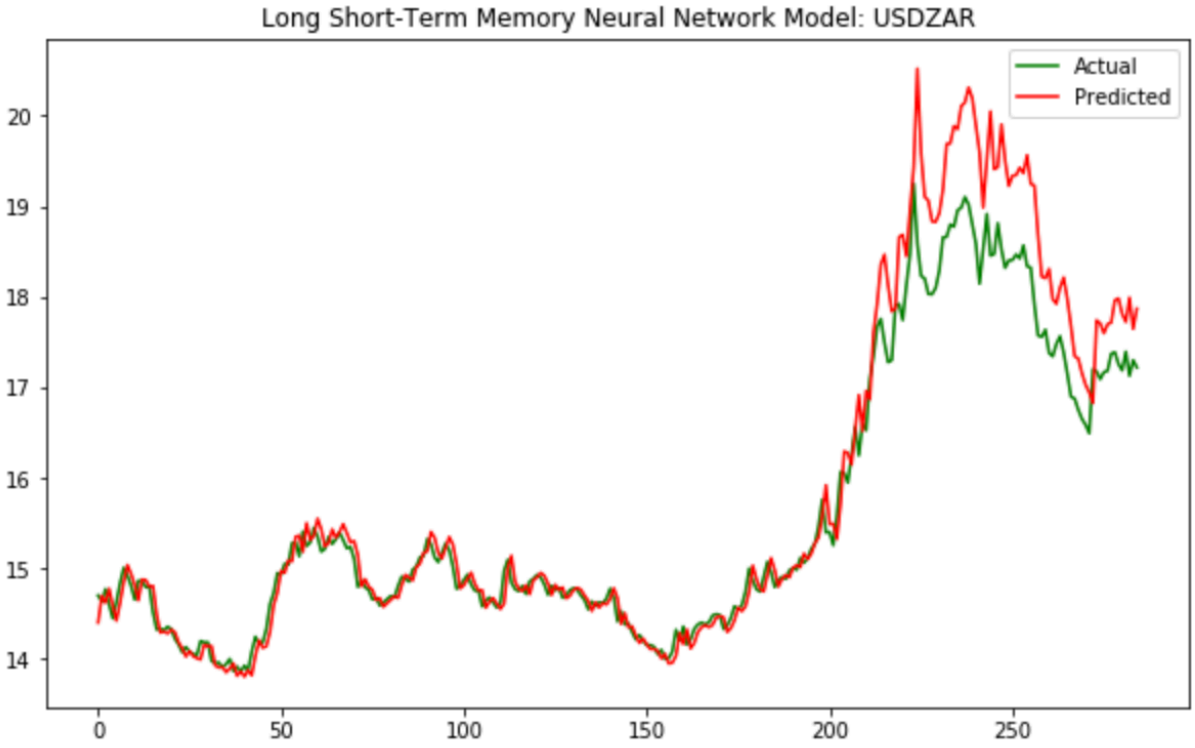


Figure 13: LSTM Neural Network Result (USDZAR)



Appendix B: Bibliography

Alvarez-Diaz, Marcos. "Exchange rates forecasting: local or global methods?" *Applied Economics* 40, no. 15 (2008). https://doi.org/10.1080/00036840600905308

Bank of International Settlements (BIS) Triennial Central Bank Survey of Foreign Exchange and Over-the-counter (OTC) Derivatives Markets in 2019. (2019). Retrieved from https://www.bis.org/statistics/rpfx19.htm

Binner, Jane, Rakesh Bissoondeeal, Thomas Elger, Alicia Gazely, and Andrew Mullineux. "A Comparison of Linear Forecasting Models and Neural Networks: an Application to Euro Inflation and Euro Divisia." *Applied Economics* 37, no. 6 (2005): 665–80. https://doi.org/10.1080/0003684052000343679

Bissoondeeal, Rakesh K., Jane M. Binner, Muddun Bhuruth, Alicia Gazely, and Veemadevi P. Mootanah. "Forecasting Exchange Rates with Linear and Nonlinear Models." *Global Business and Economics Review* 10, no. 4 (2008): 414. https://doi.org/10.1504/gber.2008.020593

Cheung, Yin-Wong, Menzie Chinn, Antonio Garcia Pascual, and Yi Zhang. "Exchange Rate Prediction Redux: New Models, New Data, New Currencies," *Journal of International Money and Finance (*2017). https://doi.org/10.3386/w23267

Colacito, R., Riddiough, S., & Sarno, L. "Business Cycles and Currency Returns." *National Bureau of Economic Research Working Paper* (2019). https://doi.org/10.3386/w26299

Dautel, A. J., Härdle, W. K., Lessmann, S., & Seow, H. "Forex exchange rate forecasting using deep recurrent neural networks." *Digital Finance, 2*(1-2), 69-96 (2020). https://doi.org/10.1007/s42521-020-00019-x

Fama, Eugene. F.. "Efficient Capital Markets: A Review of Theory and Empirical Work." *The Journal of Finance, 25*(2), 383 (1970). https://doi.org/10.2307/2325486

Fama, Eugene. F. "Efficient Capital Markets: II." *The Journal of Finance, 46*(5), 1575-1617 (1991). https://doi.org/10.1111/j.1540-6261.1991.tb04636.x

Faruqee, H. Long-Run Determinants of the Real Exchange Rate: A Stock-Flow Perspective. *IMF Working Papers, 94*(90), 1 (1994). doi:10.5089/9781451851359.001

Fischer, T., & Krauss, C. (2018). Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research, 270*(2), 654-669. https://doi.org/10.1016/j.ejor.2017.11.054

Galeshchuk, Svitlana. "Neural Networks Performance in Exchange Rate Prediction." *Neurocomputing* 172 (2016): 446–52. https://doi.org/10.1016/j.neucom.2015.03.100

Hopper, Gregory P. "What Determines the Exchange Rate: Economic Factors or Market Sentiment?" *Federal Reserve Bank of Philadelphia Business Review* (1997): 17-29

Hornik, K., Stinchcombe, M., & White, H. "Multilayer feedforward networks are universal approximators." *Neural Networks, 2*(5), 359-366 (1989). https://doi.org/10.1016/0893-6080(89)90020-8

Macdonald, R., & Clark, P. B. "Exchange Rates and Economic Fundamentals: A Methodological Comparison of Beers and Feers." *IMF Working Papers, 98*(67), 1 (1998). https://doi.org/10.5089/9781451961683.001

McCulloch, W. & Pitts, W. "A logical calculus of the ideas immanent in nervous activity" *Bulletin of Mathematical Biophysics* 5, 115-133 (1943). https://doi.org/10.1007/BF02478259

Meese, Richard A., and Kenneth Rogoff. "Empirical Exchange Rate Models of the Seventies." *Journal of International Economics* 14, no. 1-2: 3–24 (1983). https://doi.org/10.1016/0022-1996(83)90017-x

Rosenberg, Michael R. *Currency Forecasting: a Guide to Fundamental and Technical Models of Exchange Rate Determination*. Chicago: Irwin, 1996

Sk, Babu As Reddy. "Exchange Rate Forecasting Using ARIMA, Neural Network and Fuzzy Neuron." *Journal of Stock & Forex Trading* 04, no. 03 (2015). https://doi.org/10.4172/2168-9458.1000155.

Williamson, J. *The exchange rate system*. Washington, D.C.: Institute for International Economics, 1985

Wong, F. (1991). "Time series forecasting using backpropagation neural networks." *Neurocomputing, 2*(4), 147-159 (1991). https://doi.org/10.1016/0925-2312(91)90045-d

Yang, H., Pan, Z., & Tao, Q. Robust and Adaptive Online Time Series Prediction with Long Short-Term Memory. *Computational Intelligence and Neuroscience, 2017*, 1-9 (2017). https://doi.org/10.1155/2017/9478952

Zhang, Y., & Wan, X. "Statistical fuzzy interval neural networks for currency exchange rate time series prediction." *Applied Soft Computing, 7*(4), 1149-1156 (2007). https://doi.org/10.1016/j.asoc.2006.01.002

"Time Series Models in Foreign Exchange"

Appendix C: Python Code

Kris Walsh (kmw2221)

Columbia University Department of Economics

Master's Thesis First Draft

Advisor: Professor Steven Ho

November 13, 2020

Import Python Packages

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import datetime as dt
import yfinance as yf
import pandas_datareader as pdr
import itertools
import warnings
from pandas import DataFrame
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.tsa.stattools import adfuller
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler
import tensorflow as tf
import keras
from keras.models import Sequential
from keras.layers import Dense, LSTM
```

Download FX Time Series via Yahoo Finance API

In [2]:

```
fx_list = sorted(['eurusd=x', 'gbpusd=x', 'jpy=x', 'cad=x', 'mxn=x', 'zar=x'])
start = dt.datetime(2015,1,1)
end = dt.datetime(2020,6,30)
```

In [3]:

```
fx = pdr.DataReader(fx_list, 'yahoo', start, end)['Adj Close']
```

In [4]:

```
eur = fx['eurusd=x'].dropna()
gbp = fx['gbpusd=x'].dropna()
jpy = fx['jpy=x'].dropna()
cad = fx['cad=x'].dropna()
mxn = fx['mxn=x'].dropna()
zar = fx['zar=x'].dropna()
```

In [5]:

```
fx_basket = [eur, gbp, jpy, cad, mxn, zar]
```

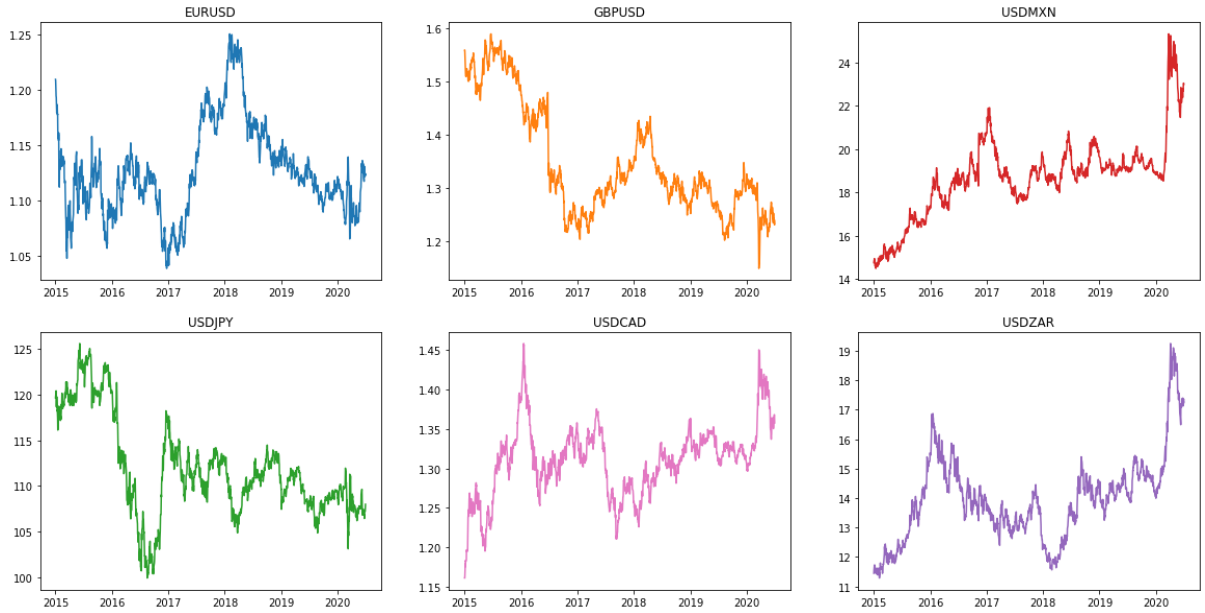
Plot of the 6 time series

In [6]:

```
fig, axs = plt.subplots(2, 3, figsize=(20,10))
axs[0, 0].plot(eur)
axs[0, 0].set_title('EURUSD')
axs[0, 1].plot(gbp, 'tab:orange')
axs[0, 1].set_title('GBPUSD')
axs[1, 0].plot(jpy, 'tab:green')
axs[1, 0].set_title('USDJPY')
axs[1, 1].plot(cad, 'tab:pink')
axs[1, 1].set_title('USDCAD')
axs[0, 2].plot(mxn, 'tab:red')
axs[0, 2].set_title('USDMXN')
axs[1, 2].plot(zar, 'tab:purple')
axs[1, 2].set_title('USDZAR')
```

Out[6]:

```
Text(0.5, 1.0, 'USDZAR')
```



ADF Test for Stationarity

In [7]:

```
def adf_test(currency):
    adf_test = adfuller(currency)
    print('stat=%.3f, p=%.3f' % adf_test[0:2])
    if adf_test[1] > 0.05:
         print('Probably not Stationary')
    else:
         print('Probably Stationary')
```

In [8]:

```
for i in fx_basket:
    adf_test(i)
```

```
stat=-3.023, p=0.033
Probably Stationary
stat=-1.783, p=0.389
Probably not Stationary
stat=-2.328, p=0.163
Probably not Stationary
stat=-3.622, p=0.005
Probably Stationary
stat=-1.896, p=0.334
Probably not Stationary
stat=-1.610, p=0.478
Probably not Stationary
```

Split into Training and Test for ARIMA and Random Walk

In [9]:

```
eur_train = eur[:1144]
eur_test = eur[1145:]
gbp_train = gbp[:1144]
gbp_test = gbp[1145:]
jpy_train = jpy[:1144]
jpy_test = jpy[1145:]
cad_train = cad[:1144]
cad_test = cad[1145:]
mxn_train = mxn[:1144]
mxn_test = mxn[1145:]
zar_train = zar[:1144]
zar_test = zar[1145:]
```

Grid Search for ARIMA Parameters

In [10]:

```
warnings.filterwarnings('ignore')
```

In [11]:

```
def grid_search(currency):
    p = range(0, 4)
    d = range(1, 3)
    q = range(0, 3)
    pdq = list(itertools.product(p, d, q))
    aics = []
    params = []
    for param in pdq:
        model = ARIMA(currency, order=param)
        model_fit = model.fit()
        aic = model_fit.aic
        aics.append(aic)
        params.append(param)
    combo = list(zip(aics, params))
    combo.sort()
    combo_array = np.array(combo)
    print(combo_array)
```

In [11]:

```
fx_train_basket = [eur_train, gbp_train, jpy_train, cad_train, mxn_train, zar_train
]
```

(Note: Grid Search may run for a few minutes)

In [13]:

```
for i in fx_train_basket:
    grid_search(i)
```

```
[[-8377.07926460606 (0, 1, 0)]
 [-8376.643272807625 (0, 1, 1)]
 [-8376.584601701035 (1, 1, 0)]
 [-8375.232795621481 (1, 1, 1)]
 [-8375.035461392245 (2, 1, 0)]
 [-8375.017673909657 (0, 1, 2)]
 [-8374.509415157665 (1, 1, 2)]
 [-8374.078956324534 (3, 1, 1)]
 [-8373.827880211886 (2, 1, 1)]
 [-8373.527394480545 (2, 1, 2)]
 [-8373.159511951706 (3, 1, 0)]
 [-8373.021895368869 (3, 1, 2)]
 [-8359.990406415993 (0, 2, 1)]
 [-8359.060025146677 (0, 2, 2)]
 [-8358.877446371665 (2, 2, 2)]
 [-8357.91392173284 (1, 2, 2)]
 [-8355.29734495656 (2, 2, 1)]
 [-8355.000927790197 (3, 2, 1)]
 [-8350.247861641381 (3, 2, 2)]
 [-8330.085198174134 (1, 2, 1)]
 [-8096.456140547243 (3, 2, 0)]
 [-8030.565245196905 (2, 2, 0)]
 [-7877.023336059554 (1, 2, 0)]
 [-7536.585038700723 (0, 2, 0)]]
[[-7678.279482555179 (3, 1, 1)]
 [-7677.820274978869 (1, 1, 2)]
 [-7677.603630289401 (2, 1, 2)]
 [-7677.314521416314 (2, 1, 1)]
 [-7677.128108194673 (3, 1, 0)]
 [-7676.901749403534 (0, 1, 0)]
 [-7676.210939763765 (3, 1, 2)]
 [-7675.466954546855 (1, 1, 0)]
 [-7675.447980059354 (0, 1, 1)]
 [-7673.849050199618 (0, 1, 2)]
 [-7673.750300338184 (2, 1, 0)]
 [-7673.508354229485 (1, 1, 1)]
 [-7660.399231183901 (0, 2, 1)]
 [-7659.752402985554 (3, 2, 2)]
 [-7658.9981579099585 (1, 2, 1)]
 [-7658.976950782977 (0, 2, 2)]
 [-7658.523502080507 (1, 2, 2)]
 [-7655.26314414807 (2, 2, 1)]
 [-7655.177851581371 (2, 2, 2)]
 [-7643.678483987898 (3, 2, 1)]
 [-7416.971673528125 (3, 2, 0)]
 [-7314.628909782701 (2, 2, 0)]
 [-7226.935138109966 (1, 2, 0)]
 [-6903.298563685793 (0, 2, 0)]]
[[2131.2095387124837 (2, 1, 2)]
 [2134.5898422175396 (1, 1, 1)]
 [2136.1087530079494 (1, 1, 2)]
 [2136.134894711842 (2, 1, 1)]
 [2137.5739697470194 (3, 1, 1)]
 [2137.676177014765 (3, 1, 2)]
 [2139.1258115848577 (0, 1, 0)]
 [2140.2639363031303 (1, 1, 0)]
 [2140.309459819902 (0, 1, 1)]
```

```
   [2141.364070237898 (2, 1, 0)]
   [2141.4615621401786 (0, 1, 2)]
   [2142.72233164838 (1, 2, 2)]
   [2142.873161767079 (3, 1, 0)]
   [2144.231097153117 (2, 2, 2)]
   [2145.7090519764943 (3, 2, 2)]
   [2147.213842070709 (0, 2, 1)]
   [2148.4001579302476 (1, 2, 1)]
   [2148.444456704413 (0, 2, 2)]
   [2149.450789456844 (2, 2, 1)]
   [2150.9968748538513 (3, 2, 1)]
   [2389.4525856937184 (3, 2, 0)]
   [2475.0256081014377 (2, 2, 0)]
   [2590.3096595335064 (1, 2, 0)]
   [2960.15474203291 (0, 2, 0)]]
  [[-8277.611633547587 (0, 1, 0)]
   [-8275.654036800324 (1, 1, 0)]
   [-8275.652609249972 (0, 1, 1)]
   [-8274.017752393835 (0, 1, 2)]
   [-8274.002087664818 (2, 1, 0)]
   [-8273.723309507408 (1, 1, 1)]
   [-8273.490768294232 (2, 1, 1)]
   [-8272.0178746326 (1, 1, 2)]
   [-8272.00556215283 (3, 1, 0)]
   [-8271.658502237226 (3, 1, 2)]
   [-8271.542414170035 (3, 1, 1)]
   [-8270.834595485863 (2, 1, 2)]
   [-8260.90668298264 (0, 2, 1)]
   [-8258.953980071385 (1, 2, 1)]
   [-8258.951817423263 (0, 2, 2)]
   [-8257.319994911993 (2, 2, 1)]
   [-8255.951764718062 (2, 2, 2)]
   [-8249.15459446466 (3, 2, 2)]
   [-8240.262403088163 (3, 2, 1)]
   [-8239.089155885284 (1, 2, 2)]
   [-8004.6050028368445 (3, 2, 0)]
   [-7944.5886144440265 (2, 2, 0)]
   [-7821.184077327072 (1, 2, 0)]
   [-7484.92375005609 (0, 2, 0)]]
  [[-1119.957824467529 (2, 1, 2)]
   [-1117.97599775974 (3, 1, 2)]
   [-1115.1726033993832 (0, 1, 0)]
   [-1113.6712455848751 (0, 1, 1)]
   [-1113.641228756887 (1, 1, 0)]
   [-1113.3651493268585 (2, 1, 1)]
   [-1113.0682488743091 (1, 1, 2)]
   [-1112.8890197910391 (0, 1, 2)]
   [-1112.7288970724571 (2, 1, 0)]
   [-1112.5690627866556 (3, 1, 1)]
   [-1112.2975274585147 (1, 1, 1)]
   [-1111.5043852757722 (3, 1, 0)]
   [-1105.0317385977482 (0, 2, 1)]
   [-1103.5383155250515 (0, 2, 2)]
   [-1103.5079199467705 (1, 2, 1)]
   [-1102.583483494404 (2, 2, 1)]
   [-1101.3521442626547 (3, 2, 1)]
   [-1101.12307263319 (1, 2, 2)]
```

```
 [-1100.102739979519 (3, 2, 2)]
 [-1099.2014239457353 (2, 2, 2)]
 [-800.6378451144383 (3, 2, 0)]
 [-759.3753152663107 (2, 2, 0)]
 [-633.8725145036392 (1, 2, 0)]
 [-345.2578603516397 (0, 2, 0)]]
[[-1161.9612498932242 (0, 1, 1)]
 [-1161.8768550305417 (2, 1, 1)]
 [-1161.800494194406 (1, 1, 2)]
 [-1161.7910364990034 (1, 1, 0)]
 [-1160.5229304255759 (2, 1, 0)]
 [-1160.380635748004 (0, 1, 2)]
 [-1160.1807483859225 (1, 1, 1)]
 [-1160.1060165158497 (0, 1, 0)]
 [-1159.8818992917736 (3, 1, 1)]
 [-1159.8817385094103 (2, 1, 2)]
 [-1159.2914508756526 (3, 1, 0)]
 [-1157.8892182881832 (3, 1, 2)]
 [-1151.1630466918255 (0, 2, 2)]
 [-1150.99468437334 (1, 2, 1)]
 [-1149.6829313385883 (2, 2, 1)]
 [-1149.2137691138237 (0, 2, 1)]
 [-1148.1840764306185 (3, 2, 1)]
 [-1147.3480408352457 (2, 2, 2)]
 [-1146.208790236417 (1, 2, 2)]
 [-1145.6712260357795 (3, 2, 2)]
 [-888.2846088970923 (3, 2, 0)]
 [-817.3508440356268 (2, 2, 0)]
 [-701.0075963441209 (1, 2, 0)]
 [-433.8462835401285 (0, 2, 0)]]
```

Run Best-Fit ARIMA Models

In [12]:

```
model_arima_eur = ARIMA(eur_train, (0,1,1))
model_arima_gbp = ARIMA(gbp_train, (3,1,1))
model_arima_jpy = ARIMA(jpy_train, (2,1,2))
model_arima_cad = ARIMA(cad_train, (1,1,0))
model_arima_mxn = ARIMA(mxn_train, (2,1,2))
model_arima_zar = ARIMA(zar_train, (0,1,1))
```

In [13]:

```
model_arima_fit_eur = model_arima_eur.fit()
model_arima_fit_gbp = model_arima_gbp.fit()
model_arima_fit_jpy = model_arima_jpy.fit()
model_arima_fit_cad = model_arima_cad.fit()
model_arima_fit_mxn = model_arima_mxn.fit()
model_arima_fit_zar = model_arima_zar.fit()
```

In [14]:

```
arima_pred_eur = model_arima_fit_eur.forecast(len(eur_test), alpha=0.05)[0]
arima_pred_gbp = model_arima_fit_gbp.forecast(len(gbp_test), alpha=0.05)[0]
arima_pred_jpy = model_arima_fit_jpy.forecast(len(jpy_test), alpha=0.05)[0]
arima_pred_cad = model_arima_fit_cad.forecast(len(cad_test), alpha=0.05)[0]
arima_pred_mxn = model_arima_fit_mxn.forecast(len(mxn_test), alpha=0.05)[0]
arima_pred_zar = model_arima_fit_zar.forecast(len(zar_test), alpha=0.05)[0]
```

Report MSE of ARIMA vs Actual Currency Path

In [15]:

```
print(mean_squared_error(eur_test,arima_pred_eur))
print(mean_squared_error(gbp_test,arima_pred_gbp))
print(mean_squared_error(jpy_test,arima_pred_jpy))
print(mean_squared_error(cad_test,arima_pred_cad))
print(mean_squared_error(mxn_test,arima_pred_mxn))
print(mean_squared_error(zar_test,arima_pred_zar))
```

```
0.0001941545578032443
0.0026633357811622106
2.321791999598577
0.0017357273828908875
3.6527211112489475
2.2369263000545745
```

Zoom in on ARIMA forecasts, short (20 days) and long (286 days)

In [16]:

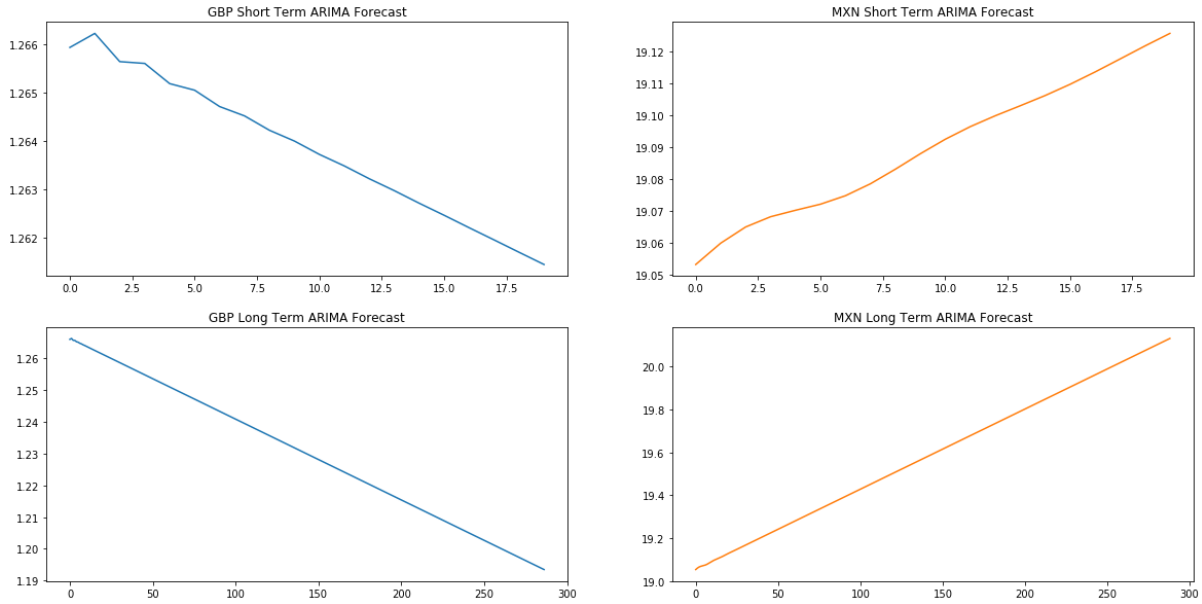
```
arima_pred_mxn_short = model_arima_fit_mxn.forecast(20, alpha=0.05)[0]
arima_pred_gbp_short = model_arima_fit_gbp.forecast(20, alpha=0.05)[0]
```

In [17]:

```
fig, axs = plt.subplots(2, 2, figsize=(20,10))
axs[0, 0].plot(arima_pred_gbp_short)
axs[0, 0].set_title('GBP Short Term ARIMA Forecast')
axs[1, 0].plot(arima_pred_gbp)
axs[1, 0].set_title('GBP Long Term ARIMA Forecast')
axs[0, 1].plot(arima_pred_mxn_short, 'tab:orange')
axs[0, 1].set_title('MXN Short Term ARIMA Forecast')
axs[1, 1].plot(arima_pred_mxn, 'tab:orange')
axs[1, 1].set_title('MXN Long Term ARIMA Forecast')
```

Out[17]:

```
Text(0.5, 1.0, 'MXN Long Term ARIMA Forecast')
```



Random Walk Results

In [18]:

```
model_arima_eur_rw = ARIMA(eur_train, (0,1,0))
model_arima_gbp_rw = ARIMA(gbp_train, (0,1,0))
model_arima_jpy_rw = ARIMA(jpy_train, (0,1,0))
model_arima_cad_rw = ARIMA(cad_train, (0,1,0))
model_arima_mxn_rw = ARIMA(mxn_train, (0,1,0))
model_arima_zar_rw = ARIMA(zar_train, (0,1,0))
```

In [19]:

```
model_arima_fit_eur_rw = model_arima_eur_rw.fit()
model_arima_fit_gbp_rw = model_arima_gbp_rw.fit()
model_arima_fit_jpy_rw = model_arima_jpy_rw.fit()
model_arima_fit_cad_rw = model_arima_cad_rw.fit()
model_arima_fit_mxn_rw = model_arima_mxn_rw.fit()
model_arima_fit_zar_rw = model_arima_zar_rw.fit()
```

In [20]:

```
arima_pred_eur_rw = model_arima_fit_eur_rw.forecast(len(eur_test), alpha=0.05)[0]
arima_pred_gbp_rw = model_arima_fit_gbp_rw.forecast(len(gbp_test), alpha=0.05)[0]
arima_pred_jpy_rw = model_arima_fit_jpy_rw.forecast(len(jpy_test), alpha=0.05)[0]
arima_pred_cad_rw = model_arima_fit_cad_rw.forecast(len(cad_test), alpha=0.05)[0]
arima_pred_mxn_rw = model_arima_fit_mxn_rw.forecast(len(mxn_test), alpha=0.05)[0]
arima_pred_zar_rw = model_arima_fit_zar_rw.forecast(len(zar_test), alpha=0.05)[0]
```

In [21]:

```
print(mean_squared_error(eur_test,arima_pred_eur_rw))
print(mean_squared_error(gbp_test,arima_pred_gbp_rw))
print(mean_squared_error(jpy_test,arima_pred_jpy_rw))
print(mean_squared_error(cad_test,arima_pred_cad_rw))
print(mean_squared_error(mxn_test,arima_pred_mxn_rw))
print(mean_squared_error(zar_test,arima_pred_zar_rw))
```

```
0.00019388120775006782
0.002674872365574409
2.3232498934950008
0.001734354839061851
3.6541375740629225
2.251010706606456
```

Zoom in on Random Walk forecasts, short (20 days) and long (286 days)

In [22]:

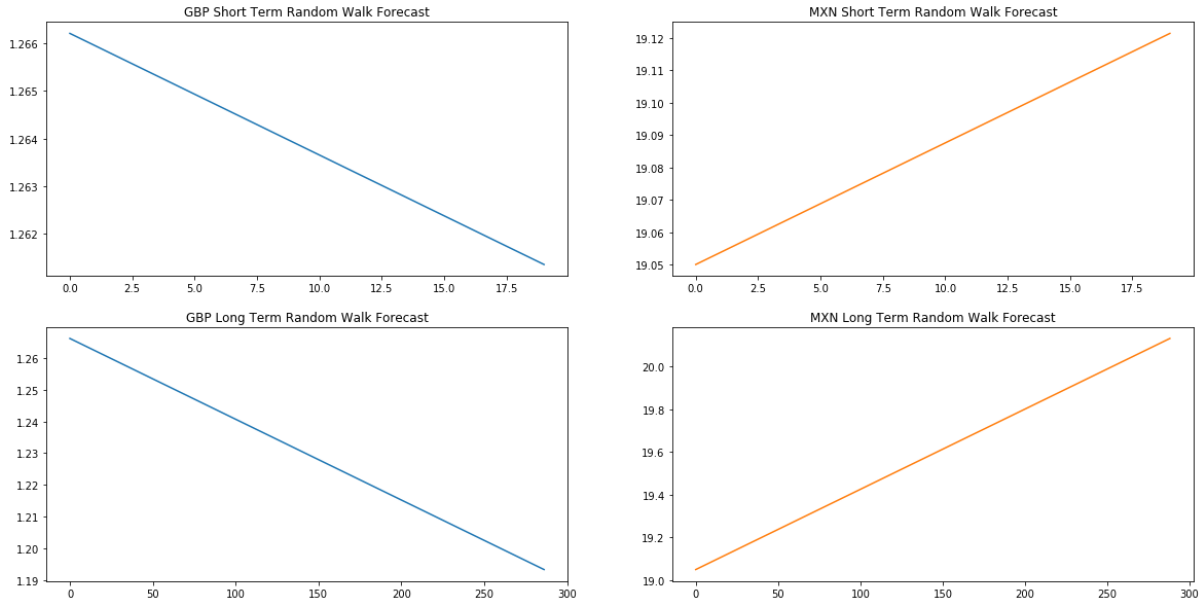
```
arima_pred_mxn_short_rw = model_arima_fit_mxn_rw.forecast(20, alpha=0.05)[0]
arima_pred_gbp_short_rw = model_arima_fit_gbp_rw.forecast(20, alpha=0.05)[0]
```

In [23]:

```
fig, axs = plt.subplots(2, 2, figsize=(20,10))
axs[0, 0].plot(arima_pred_gbp_short_rw)
axs[0, 0].set_title('GBP Short Term Random Walk Forecast')
axs[1, 0].plot(arima_pred_gbp_rw)
axs[1, 0].set_title('GBP Long Term Random Walk Forecast')
axs[0, 1].plot(arima_pred_mxn_short_rw, 'tab:orange')
axs[0, 1].set_title('MXN Short Term Random Walk Forecast')
axs[1, 1].plot(arima_pred_mxn_rw, 'tab:orange')
axs[1, 1].set_title('MXN Long Term Random Walk Forecast')
```

Out[23]:

```
Text(0.5, 1.0, 'MXN Long Term Random Walk Forecast')
```



Neural Network (Long Short-Term Memory)

In [126]:

```
df = gbp
```

In [127]:

```
scaler = MinMaxScaler()
```

In [128]:

```
df = np.array(df).reshape(-1,1)
```

In [129]:

```
df = scaler.fit_transform(df)
```

In [130]:

```
#Training and test sets
train = df[:1145]
test = df[1145:]
```

In [131]:

```
def get_data(data, look_back):
  data_x, data_y = [],[]
  for i in range(len(data)-look_back-1):
    data_x.append(data[i:(i+look_back),0])
    data_y.append(data[i+look_back,0])
  return np.array(data_x) , np.array(data_y)

look_back = 1

x_train , y_train = get_data(train, look_back)
```

In [132]:

```
x_test , y_test = get_data(test,look_back)
```

In [133]:

```
#Processing train and test sets for LSTM model
x_train = x_train.reshape(x_train.shape[0],x_train.shape[1], 1)
x_test = x_test.reshape(x_test.shape[0],x_test.shape[1], 1)
```

In [134]:

```
print(x_train.shape)
print(x_test.shape)
```

```
(1143, 1, 1)
(285, 1, 1)
```

In [135]:

```
#Defining the LSTM model
n_features=x_train.shape[1]
model=Sequential()
model.add(LSTM(10,activation='relu',input_shape=(1,1)))
model.add(Dense(n_features))
```

In [136]:

```
model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_6 (LSTM)                (None, 10)                480
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 11
=================================================================
Total params: 491
Trainable params: 491
Non-trainable params: 0
_________________________________________________________________
```

In [137]:

```
#Compiling
model.compile(optimizer='adam', loss = 'mse')
```

In [138]:

```
#Training
model.fit(x_train,y_train, epochs = 5, batch_size=1)
```

```
Epoch 1/5
1143/1143 [==============================] - 1s 845us/step - loss: 0.04
54
Epoch 2/5
1143/1143 [==============================] - 1s 796us/step - loss: 0.00
27
Epoch 3/5
1143/1143 [==============================] - 1s 825us/step - loss: 5.76
57e-04
Epoch 4/5
1143/1143 [==============================] - 1s 814us/step - loss: 5.75
64e-04
Epoch 5/5
1143/1143 [==============================] - 1s 936us/step - loss: 5.80
35e-04
```

Out[138]:

```
<tensorflow.python.keras.callbacks.History at 0x147522ad0>
```

In [139]:

```
#Prediction using the trained model
scaler.scale_
y_pred = model.predict(x_test)
y_pred = scaler.inverse_transform(y_pred)
```

In [140]:

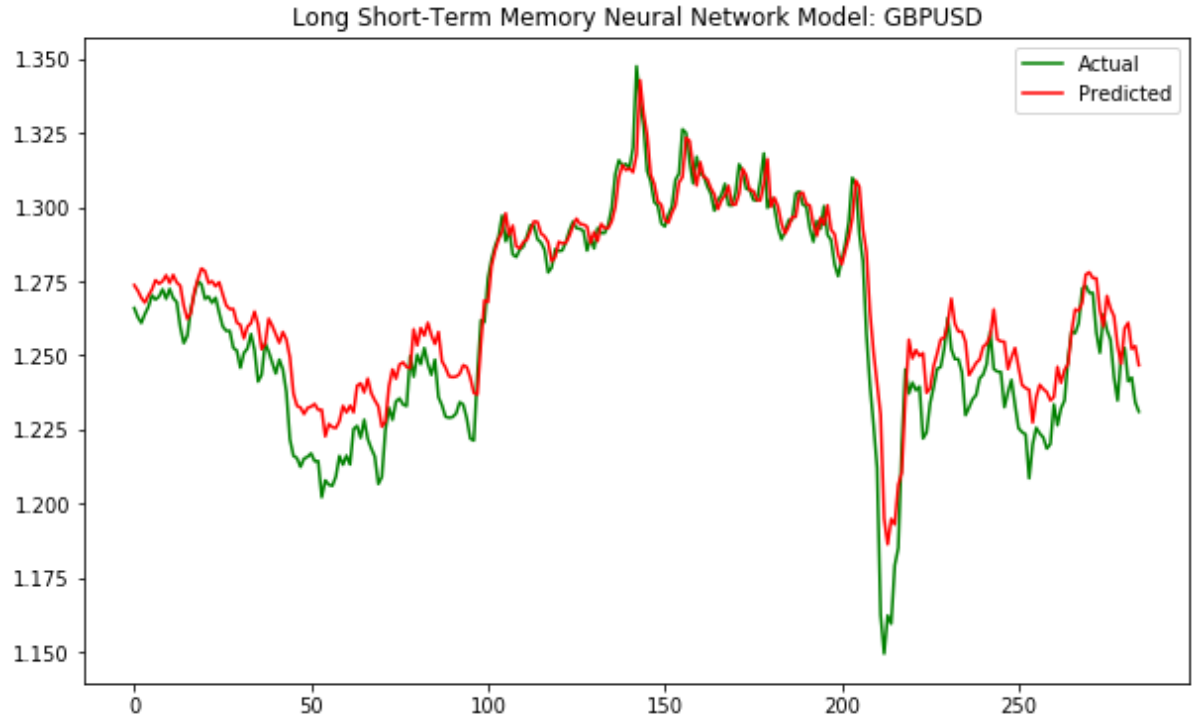
```
#Processing test shape
y_test = np.array(y_test).reshape(-1,1)
y_test = scaler.inverse_transform(y_test)
```

In [141]:

```
#Visualizing the results
plt.figure(figsize=(10,6))
plt.title('Long Short-Term Memory Neural Network Model: GBPUSD')
plt.plot(y_test , label = 'Actual', color = 'g')
plt.plot(y_pred , label = 'Predicted', color = 'r')
plt.legend()
```

Out[141]:

```
<matplotlib.legend.Legend at 0x14787dbd0>
```



In [43]:

```
mean_squared_error(y_test, y_pred)
```

Out[43]:

```
0.0001950241394013119
```