BTC Prediction Model

Overview

This project uses bitcoin (BTC) historical data from November of 2015 through November of 2021. The goal was to build a model that could create a roughly accurate short term price prediction for the cryptocurrency. Additionally, it was used as a personal research study to determine the current ability of semi-basic modelling to predict the price movements of BTC. The notebook contains sections pertaining to exploratory data analysis (EDA) and data cleaning, ARIMA modelling, and the Long Short-term Neural Network.

Business Problem

- My stakeholder is the everyday common investor looking to invest in cryptocurrencies but are discouraged due to the volatility.
- To try and alleviate some of this volatility, I aim to create a simple model that can roughly (by roughly, - I mean predict a positive or negative price movement) predict the price of bitcoin. Doing so would be able to help give traders insight as to where the price of BTC will be in the near future.
- The model does not aim to accurately predict the exact price movements for bitcoin and therefore should not be expected to be used as a way to 'beat' the market.

Data Understanding

- The data represents the last six years (November 2015 - November 2021 of BTC price and volume data and was sourced from yahoo Finance.
- The data includes the Volume, High and Low prices, and the Open, Close and Adjusted Close prices for each day.
- Another column was added to the data, that being the percent change for each day.
- The target variable for the ARIMA model will be the percent change data, as an adfuller test indicated that it was stationary.
- The target variable for the Neural Network will be the Adjusted Close prices.
- All of the data, besides the date, are floats.

```
#Importing all necesarry functions and libraries.
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import statsmodels.api as sm
from statsmodels.tsa.arima.model import ARIMA
import pmdarima as pmd
from pmdarima import model_selection
from pmdarima.arima.stationarity import ADFTest
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout

%matplotlib inline
```

EDA and Data Cleaning

```
#Reading the dataset and saving as 'all_btc' using pandas.

all_btc = pd.read_csv('/BTC-USD.csv')
```

```
#Changing the date to datetime format and setting the date as the index.

all_btc.Date = pd.to_datetime(all_btc.Date, format='%Y-%m-%d')
all_btc = all_btc.set_index('Date')
all_btc
```

	Open	High	Low	Close	Adj Close	Volume
Date						

```
#Creating a percent change column for the adjusted close data, as this will likely cre
#more stationary data.

all_btc['Percent change'] = all_btc['Adj Close'].pct_change()
```

```
#Exploring null values in the adjusted close column in the dataset.

all_btc['Adj Close'].isnull().sum()
```

0

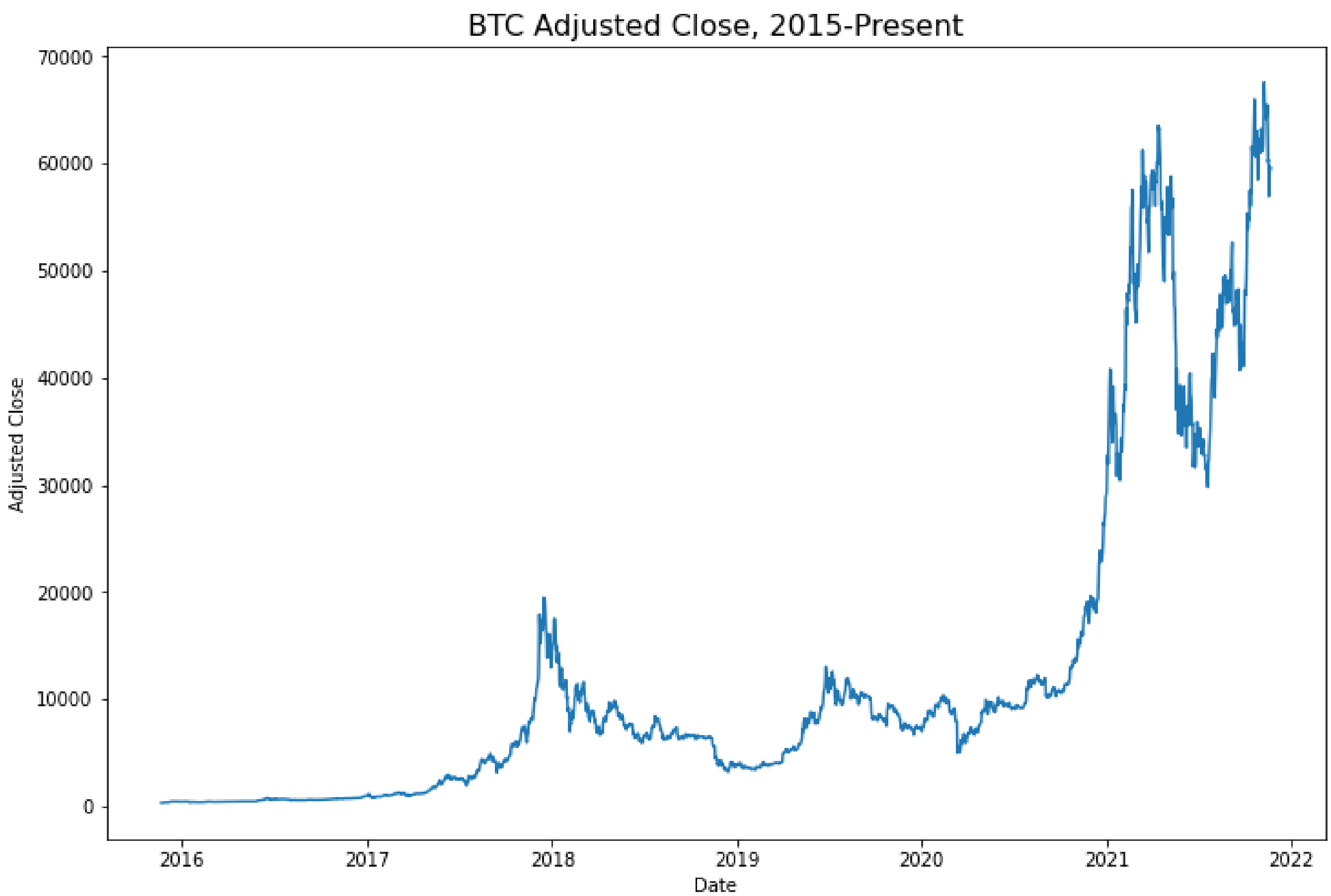
2015	320.045013	329.134003	316.769989	328.205994	328.205994	4.166690e+07

```
#Filling null values in the dataframe using their most previous values.

all_btc = all_btc.fillna(method='pad')
all_btc
```

	Open	High	Low	Close	Adj Close	Volume
Date						
2015-11-21	322.092010	328.158997	319.595001	326.927002	326.927002	2.820050e+07
2015-11-22	326.975006	327.010010	321.259003	324.536011	324.536011	2.343940e+07
2015-11-23	324.350006	325.118011	321.290009	323.045990	323.045990	2.747890e+07
2015-11-24	323.014008	323.058014	318.118011	320.045990	320.045990	2.936260e+07
2015-11-25	320.045013	329.134003	316.769989	328.205994	328.205994	4.166690e+07
...	...	...	...	...	...	...
2021-11-17	60139.621094	60823.609375	58515.410156	60368.011719	60368.011719	3.917839e+10
2021-11-18	60360.136719	60948.500000	56550.792969	56942.136719	56942.136719	4.138834e+10

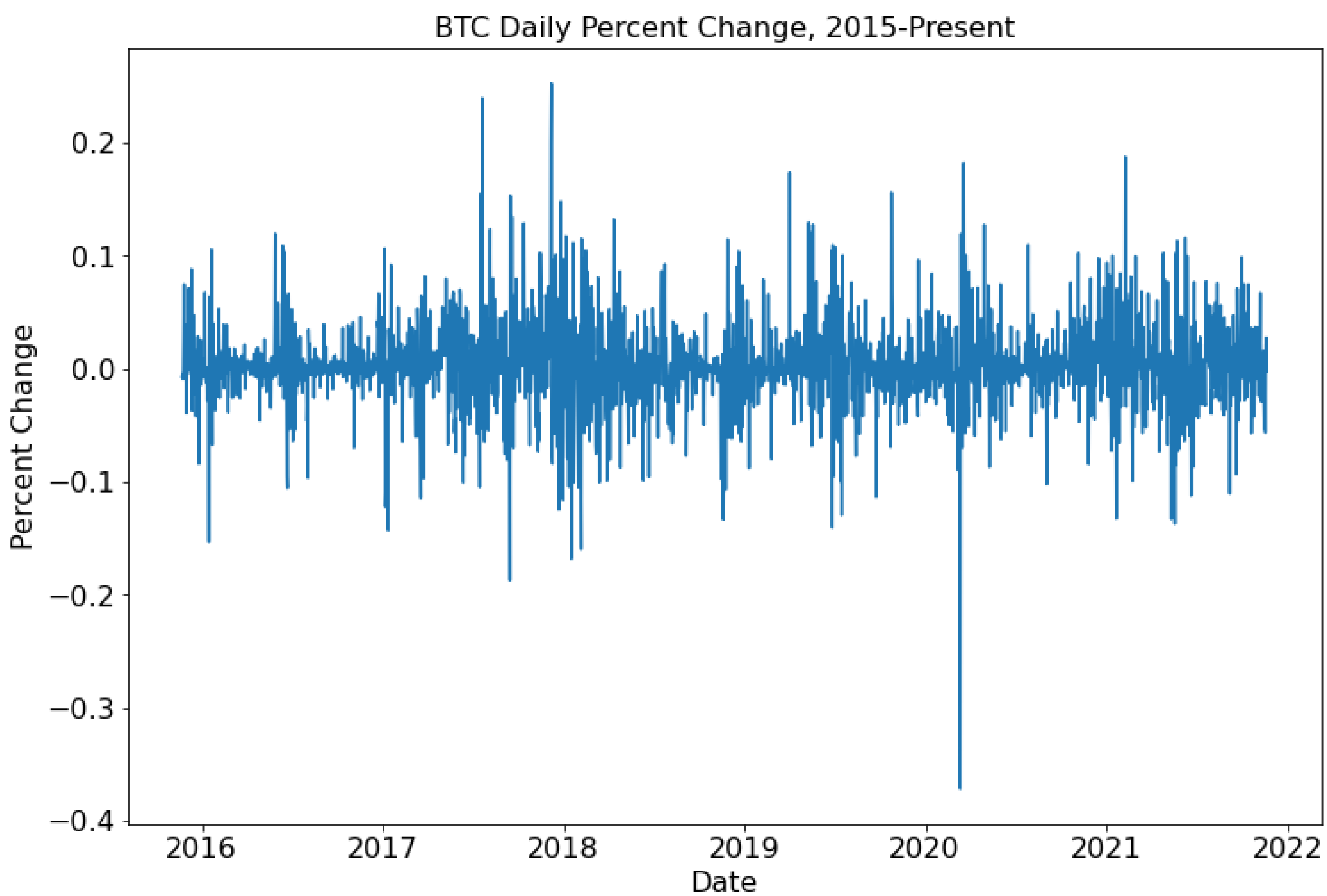
```
#Visualizing the adjusted close versus time for the dataset. Movements
#appear to be completely random.

font = {'family' : 'normal',
        'weight' : 'bold',
```

```
        'size'   : 22}

SMALL_SIZE = 16
MEDIUM_SIZE = 24
BIGGER_SIZE = 30

plt.figure(figsize=(12, 8))
plt.plot(all_btc['Adj Close'])
plt.rc('font', size=MEDIUM_SIZE)          # controls default text sizes
plt.rc('axes', titlesize=SMALL_SIZE)     # fontsize of the axes title
plt.rc('axes', labelsize=SMALL_SIZE)    # fontsize of the x and y labels
plt.rc('xtick', labelsize=SMALL_SIZE)    # fontsize of the tick labels
plt.rc('ytick', labelsize=SMALL_SIZE)    # fontsize of the tick labels
plt.rc('legend', fontsize=SMALL_SIZE)    # legend fontsize
plt.rc('figure', titlesize=BIGGER_SIZE)  # fontsize of the figure title
plt.xlabel('Date')
plt.ylabel('Adjusted Close', )
plt.title('BTC Adjusted Close, 2015-Present')
plt.show()
```




```
#Visualizing the 'percent change' data versus time. This graph  still
#looks random but it seems more manageable.

font = {'family' : 'normal',
```

```
        'weight' : 'bold',
        'size'   : 22}

SMALL_SIZE = 16
MEDIUM_SIZE = 24
BIGGER_SIZE = 30

plt.figure(figsize=(12, 8))
plt.plot(all_btc['Percent change'])
plt.rc('font', size=MEDIUM_SIZE)          # controls default text sizes
plt.rc('axes', titlesize=SMALL_SIZE)     # fontsize of the axes title
plt.rc('axes', labelsize=SMALL_SIZE)    # fontsize of the x and y labels
plt.rc('xtick', labelsize=SMALL_SIZE)    # fontsize of the tick labels
plt.rc('ytick', labelsize=SMALL_SIZE)    # fontsize of the tick labels
plt.rc('legend', fontsize=SMALL_SIZE)    # legend fontsize
plt.rc('figure', titlesize=BIGGER_SIZE)  # fontsize of the figure title
plt.xlabel('Date')
plt.ylabel('Percent Change')
plt.title('BTC Daily Percent Change, 2015-Present')
plt.show()
```




```
#importing and running adfuller test on the adjusted close and percent change columns.
```

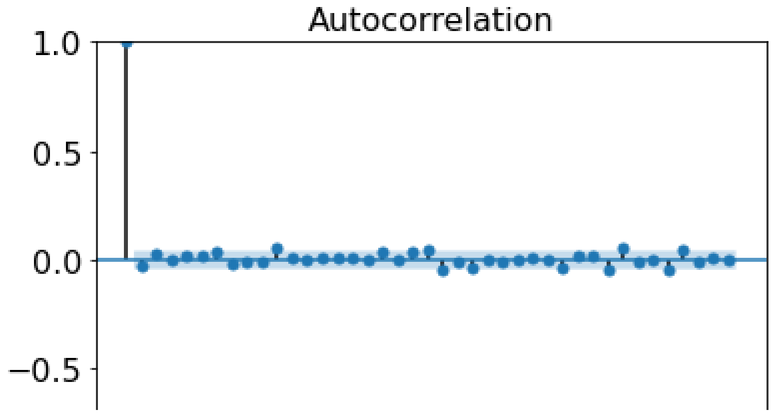
```
#The null hypothesis: The data is non stationary
#The alternative: The data is stationary

af_close = adfuller(all_btc['Adj Close'])
af_pct = adfuller(all_btc['Percent change'][1:2193])
af_close
```

```
(0.4648315536085863,
 0.9837640593857041,
 26,
 2166,
 {'1%': -3.433372653139527,
  '10%': -2.567480848042739,
  '5%': -2.8628753016111688},
 35082.23633473163)
```

```
#adfuller test for the percent change column shows that it is very
#stationary because the p value (0.0) is much smaller than 0.05.

af_pct
```

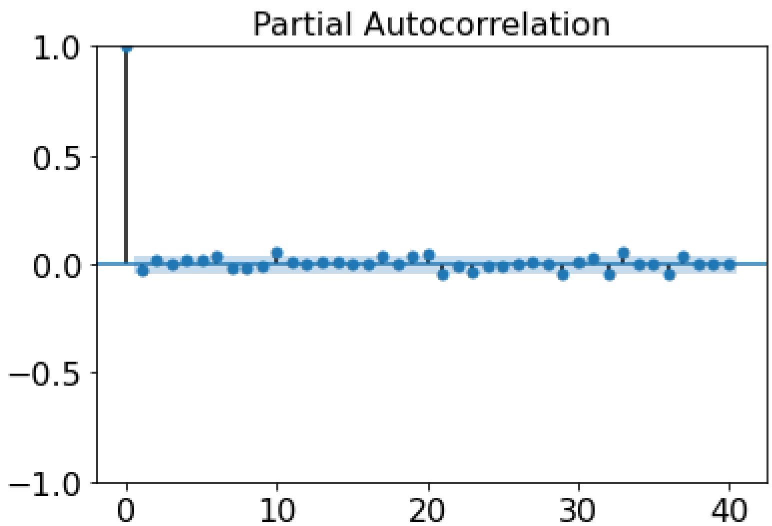
```
(-48.16909346786212,
 0.0,
 0,
 2191,
 {'1%': -3.433338123180619,
  '10%': -2.567472730288902,
  '5%': -2.862860055130884},
 -7826.636300292972)
```

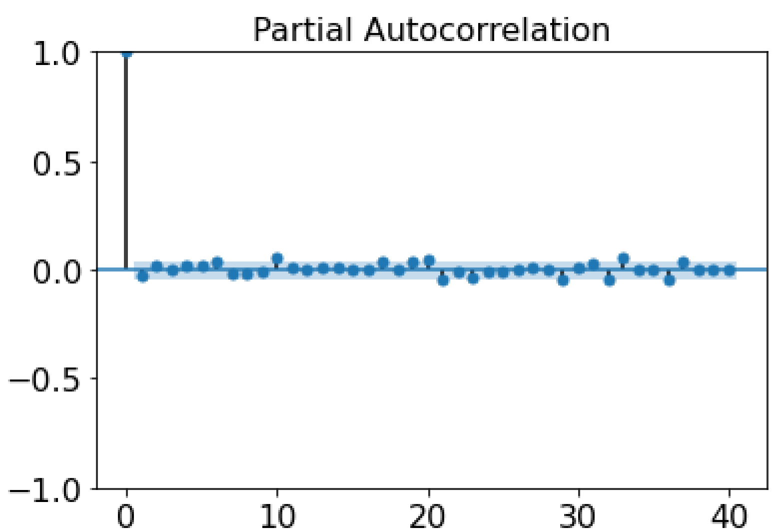
```
#Plotting the autocorrelation function for percent change.

plot_acf(all_btc['Percent change'].dropna(), lags=40)
```




```
#Plotting the partial autocorrelation function for percent change.

plot_pacf(all_btc['Percent change'].dropna(), lags=40)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/graphics/tsaplots.py:353: Futu
  FutureWarning,
```





ARIMA

```
#Fitting the ARIMA model and viewing the summary statistics for it.

first_model = ARIMA(all_btc['Percent change'], order=(1,1,1))
first_model_fit = first_model.fit()
first_model_fit.summary()
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:539: Va
  % freq, ValueWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:539: Va
  % freq, ValueWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:539: Va
  % freq, ValueWarning)
```

SARIMAX Results

Dep. Variable:	Percent change	**No. Observations:**	2193
Model:	ARIMA(1, 1, 1)	**Log Likelihood**	3954.513
Date:	Thu, 09 Dec 2021	**AIC**	-7903.026
Time:	06:08:36	**BIC**	-7885.948
Sample:	11-21-2015 - 11-21-2021	**HQIC**	-7896.784
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.0286	0.015	-1.958	0.050	-0.057	2.72e-05
ma.L1	-1.0000	0.044	-22.689	0.000	-1.086	-0.914
sigma2	0.0016	6.69e-05	23.411	0.000	0.001	0.002

Ljung-Box (L1) (Q):	0.00	**Jarque-Bera (JB):**	4924.52
Prob(Q):	0.99	**Prob(JB):**	0.00
Heteroskedasticity (H):	1.21	**Skew:**	-0.14
Prob(H) (two-sided):	0.01	**Kurtosis:**	10.34

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

```
#Viewing the predictions made by the first ARIMA model. As we can
#see, it did not work very well. Because of this, we will try auto
#ARIMA and then move on to a neural network.

predictions = first_model_fit.predict(start=2180, end=2200, dynamic=True)
predictions
```

```
2021-11-09    0.001419
2021-11-10    0.003294
2021-11-11    0.003240
2021-11-12    0.003242
2021-11-13    0.003242
2021-11-14    0.003242
2021-11-15    0.003242
2021-11-16    0.003242
```

```
2021-11-17    0.003242
2021-11-18    0.003242
2021-11-19    0.003242
2021-11-20    0.003242
2021-11-21    0.003242
2021-11-22    0.003242
2021-11-23    0.003242
2021-11-24    0.003242
2021-11-25    0.003242
2021-11-26    0.003242
2021-11-27    0.003242
2021-11-28    0.003242
2021-11-29    0.003242
Freq: D, Name: predicted_mean, dtype: float64
```

```
#Comparing the predictions to the actual values.

all_btc.tail(20)
```

	Open	High	Low	Close	Adj Close	Volume
Date						

```
#Performing an Augmented Dickey-Fuller test to determine whether the
#percent change data needed to be differenced prior to using auto
#ARIMA

adf_test = ADFTest(alpha=0.05)

p_val, should_diff = adf_test.should_diff(all_btc['Percent change'][1:2193].fillna(met

print('P-val: ', p_val)
print('Difference data? ', should_diff)
```

```
P-val:  0.01
Difference data?  False
```

2021-	61554.921875	63326.988281	61422.488281	63326.988281	63326.988281	2.473675e+10

```
#Creating the train, test, and validation sets of data to perform auto ARIMA on.

train, test, val = (all_btc['Percent change'][1:2100], all_btc['Percent change'][2100:
                    all_btc['Percent change'][2160:2193])
```

11-09	67549.734375	68530.335938	66382.062500	66971.828125	66971.828125	4.235799e+10

```
#Performing auto ARIMA on the training set.

arima = pmd.auto_arima(train, error_action='ignore', trace=True, supress_warnings=True
                     maxiter=100, seasonal=False)
```

```
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0]             : AIC=-7556.486, Time=0.42 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=-7560.393, Time=0.14 sec
 ARIMA(1,0,0)(0,0,0)[0]             : AIC=-7559.679, Time=0.30 sec
 ARIMA(0,0,1)(0,0,0)[0]             : AIC=-7559.587, Time=0.57 sec
 ARIMA(1,0,1)(0,0,0)[0]             : AIC=-7558.609, Time=0.29 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=-7571.843, Time=0.36 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=-7571.901, Time=0.40 sec
 ARIMA(2,0,0)(0,0,0)[0] intercept   : AIC=-7571.814, Time=0.34 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=-7570.914, Time=1.44 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=-7571.777, Time=0.28 sec
 ARIMA(2,0,1)(0,0,0)[0] intercept   : AIC=-7569.899, Time=0.41 sec

Best model:  ARIMA(1,0,0)(0,0,0)[0] intercept
Total fit time: 4.974 seconds
```

```
#Looking at the summary statistics for the auto ARIMA.
#Best model: ARIMA(1,0,0)(0,0,0)[0] intercept

arima.summary()
```

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	2099
Model:	SARIMAX(1, 0, 0)	**Log Likelihood**	3788.950
Date:	Thu, 09 Dec 2021	**AIC**	-7571.901
Time:	06:08:59	**BIC**	-7554.953
Sample:	0 - 2099	**HQIC**	-7565.693
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	0.0033	0.001	3.788	0.000	0.002	0.005
ar.L1	-0.0313	0.015	-2.113	0.035	-0.060	-0.002
sigma2	0.0016	2.26e-05	70.038	0.000	0.002	0.002

Ljung-Box (L1) (Q):	0.00	**Jarque-Bera (JB):**	4965.14
Prob(Q):	0.97	**Prob(JB):**	0.00
Heteroskedasticity (H):	1.26	**Skew:**	-0.11
Prob(H) (two-sided):	0.00	**Kurtosis:**	10.53

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

```
#Looking at the predictions made form using auto ARIMA. Once again,
#the predictions do not seem sufficient.

arima.predict(X=val)
```

```
array([0.00153377, 0.00324331, 0.00318977, 0.00319145, 0.00319139,
       0.0031914 , 0.0031914 , 0.0031914 , 0.0031914 , 0.0031914 ])
```

```
all_btc.tail(10)
```

Open High Low Close Adj Close Volume

Neural Network

2021-

```
#Creating a series type dataset for the adjusted close data and
#inspecting the first 15 rows.

adj_close_series = pd.Series(data=all_btc['Adj Close'], index=all_btc.index)
adj_close_series.head(15)
```

```
Date
2015-11-21    326.927002
2015-11-22    324.536011
2015-11-23    323.045990
2015-11-24    320.045990
2015-11-25    328.205994
2015-11-26    352.683990
2015-11-27    358.041992
2015-11-28    357.381012
2015-11-29    371.294006
2015-11-30    377.321014
2015-12-01    362.488007
2015-12-02    359.187012
2015-12-03    361.045990
2015-12-04    363.183014
2015-12-05    388.949005
Name: Adj Close, dtype: float64
```

```
#Creating a series type dataset for the percent change data. This
#will likely not be used as of yet but could be utilized in the future.

pct_change_series = pd.Series(data=all_btc['Percent change'], index=all_btc.index)
pct_change_series.head(15)
```

```
Date
2015-11-21         NaN
2015-11-22   -0.007314
2015-11-23   -0.004591
2015-11-24   -0.009287
2015-11-25    0.025496
2015-11-26    0.074581
2015-11-27    0.015192
2015-11-28   -0.001846
2015-11-29    0.038930
2015-11-30    0.016232
2015-12-01   -0.039311
2015-12-02   -0.009106
2015-12-03    0.005176
2015-12-04    0.005919
2015-12-05    0.070945
Name: Percent change, dtype: float64
```

```
#Defining a function to create a 'supervised' dataset.

def timeseries_to_supervised(data, lag=1):
    df = pd.DataFrame(data)
    columns = [df.shift(i) for i in range(1, lag+1)]
    columns.append(df)
    df = pd.concat(columns, axis=1)
    df.fillna(0, inplace=True)
    return df


#Using the timeseries_to_supervised function to create a supervised
#dataset for the adjusted close data.

adj_supervised = timeseries_to_supervised(adj_close_series)


#Defining a funtion that will difference a dataset for me.

def difference(dataset, interval=1):
    diff = list()
    for i in range(interval, len(dataset)):
        value = dataset[i] - dataset[i - interval]
        diff.append(value)
    return pd.Series(diff)


#Differencing the supervised adjusted close dataset.

adj_close_diff = difference(adj_close_series)


adj_close_supervised = timeseries_to_supervised(adj_close_diff)


#Performing a train, test, validation split of the adjusted close data.

train_adj, test_adj, val_adj = (adj_close_supervised[0:2100],
                                adj_close_supervised[2100:2160],
                                adj_close_supervised[2160:])


#Creating function that will scale data for me using the MinMax Scaler.

def scale(train, test):
    scaler=MinMaxScaler(feature_range=(-1, 1))
    #Training data
    scaler.fit(train)
    train_scaled = scaler.transform(train)
    #Testing data
    scaler.fit(test)
```

```
    test_scaled = scaler.transform(test)
    return scaler, train_scaled, test_scaled

#Scaling the test and train datasets using the MinMax scaler that is
#contained within the previously defined 'scale' function.

scaler, train_scaled_adj, test_scaled_adj = scale(train_adj, test_adj)


#Defining function to fit a Long Short-term Memory neural network to the data.

def fit_LSTM(train, batch_size, n_epoch, neurons):
    X, y = train[:, 0:-1], train[:, -1]
    X = X.reshape(X.shape[0], 1, X.shape[1])
    model = Sequential()
    model.add(LSTM(neurons, batch_input_shape=(batch_size, X.shape[1],
                                              X.shape[2])))
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer='adam', metrics=['acc'])
    for i in range (n_epoch):
        model.fit(X, y, epochs=1, batch_size=batch_size, verbose=0, shuffle=False)
        model.reset_states()
    return model


#Creating a function that will provide a forecast for a long short-term
#memory neural netowkr to be used in forming the predictions.

def forecast_lstm(model, batch_size, X):
    X = X.reshape(1, 1, len(X))
    yhat = model.predict(X, batch_size=batch_size)
    return yhat[0,0]


#Fitting the first long short-term memory neural network on the

lstm_model = fit_LSTM(train_scaled_adj, 50, 2000, 3)


#Reshaping the training data and saving it as a new dataset to be used
#in forming the predictions for our model.

train_reshaped_adj = train_scaled_adj[:, 0].reshape(len(train_scaled_adj), 1, 1)


#Creating a funtion to invert the changes made from scaling the data.
#This is done in order to get interpretable results from our predictions list.

def invert_scaler(scaler, X, value):
    new_row = [x for x in X] + [value]
    array = np.array(new_row)
    array = array.reshape(1, len(array))
```

```
    inverted = scaler.inverse_transform(array)
    return inverted[0, -1]
```

```
#Another function used to revert back to original values, effectively
#reversing changes made from differencing our data.

def inverse_difference(history, yhat, interval=1):
    return yhat + history[-interval]


#Creating a list of predictions from our first LSTM model. As we can
#see, it does not perform overly well

predictions = list()
for i in range(len(test_scaled_adj)):
    # make one-step forecast
    X, y = test_scaled_adj[i, 0:-1], test_scaled_adj[i, -1]
    yhat = forecast_lstm(lstm_model, 1, X)
    # invert scaling
    yhat = invert_scaler(scaler, X, yhat)
    # invert differencing
    yhat = inverse_difference(adj_close_series.values, yhat, len(test_scaled_adj)+1-i)
    # store forecast
    predictions.append(yhat)
    expected = adj_close_series.values[len(train_adj) + i + 1]
    print('Day=%d, Predicted=%f, Expected=%f' % (i+1, yhat, expected))
```

```
Day=1, Predicted=42926.120414, Expected=49321.652344
Day=2, Predicted=44256.588758, Expected=49546.148438
Day=3, Predicted=42198.425496, Expected=47706.117188
Day=4, Predicted=42129.786499, Expected=48960.789063
Day=5, Predicted=42580.895865, Expected=46942.218750
Day=6, Predicted=41672.349438, Expected=49058.667969
Day=7, Predicted=40413.275505, Expected=48902.402344
Day=8, Predicted=40918.083024, Expected=48829.832031
Day=9, Predicted=43145.564176, Expected=47054.984375
Day=10, Predicted=47522.723833, Expected=47166.687500
Day=11, Predicted=47068.560181, Expected=48847.027344
Day=12, Predicted=47576.052705, Expected=49327.722656
Day=13, Predicted=48475.343122, Expected=50025.375000
Day=14, Predicted=50880.365245, Expected=49944.625000
Day=15, Predicted=54716.021334, Expected=51753.410156
Day=16, Predicted=53182.974804, Expected=52633.535156
Day=17, Predicted=53335.850235, Expected=46811.128906
Day=18, Predicted=55549.005174, Expected=46091.390625
Day=19, Predicted=54123.672865, Expected=46391.421875
Day=20, Predicted=56844.567594, Expected=44883.910156
Day=21, Predicted=55422.366234, Expected=45201.457031
Day=22, Predicted=56761.133816, Expected=46063.269531
Day=23, Predicted=56689.288879, Expected=44963.074219
Day=24, Predicted=60953.631988, Expected=47092.492188
Day=25, Predicted=60270.974623, Expected=48176.347656
Day=26, Predicted=60924.109713, Expected=47783.359375
Day=27, Predicted=61377.872979, Expected=47267.519531
Day=28, Predicted=63613.404805, Expected=48278.363281
```

```
Day=29, Predicted=65362.468444, Expected=47260.218750
Day=30, Predicted=61567.359102, Expected=42843.800781
Day=31, Predicted=60830.416268, Expected=40693.675781
Day=32, Predicted=60850.552801, Expected=43574.507813
Day=33, Predicted=60312.082540, Expected=44895.097656
Day=34, Predicted=62412.844231, Expected=42839.750000
Day=35, Predicted=59805.825559, Expected=42716.593750
Day=36, Predicted=57836.469011, Expected=43208.539063
Day=37, Predicted=59984.742387, Expected=42235.730469
Day=38, Predicted=61583.996212, Expected=41034.542969
Day=39, Predicted=61252.361015, Expected=41564.363281
Day=40, Predicted=60682.115743, Expected=43790.894531
Day=41, Predicted=60383.642730, Expected=48116.941406
Day=42, Predicted=62609.008552, Expected=47711.488281
Day=43, Predicted=62321.780455, Expected=48199.953125
Day=44, Predicted=60814.785147, Expected=49112.902344
Day=45, Predicted=60494.098157, Expected=51514.812500
Day=46, Predicted=60907.407701, Expected=55361.449219
Day=47, Predicted=62709.379004, Expected=53805.984375
Day=48, Predicted=66951.861483, Expected=53967.847656
Day=49, Predicted=66329.604690, Expected=54968.222656
Day=50, Predicted=64364.736628, Expected=54771.578125
Day=51, Predicted=64303.264740, Expected=57484.789063
Day=52, Predicted=63536.807236, Expected=56041.058594
Day=53, Predicted=63846.299019, Expected=57401.097656
Day=54, Predicted=64840.241791, Expected=57321.523438
Day=55, Predicted=62912.457160, Expected=61593.949219
Day=56, Predicted=59543.833748, Expected=60892.179688
Day=57, Predicted=59719.968486, Expected=61553.617188
Day=58, Predicted=56307.182477, Expected=62026.078125
```

```
#Fitting the final LSTM.

final_lstm = fit_LSTM(train_scaled_adj, 10, 2000, 4)
```

```
#Looking at the predictions made using the final LSTM. Once again,
#the predictions are not great, however, at least we begin to see both
#positive and negative movement in our predictoins.

predictions = list()
for i in range(len(test_scaled_adj)):
    # make one-step forecast
    X, y = test_scaled_adj[i, 0:-1], test_scaled_adj[i, -1]
    yhat = forecast_lstm(final_lstm, 1, X)
    # invert scaling
    yhat = invert_scaler(scaler, X, yhat)
    # invert differencing
    Predicted = inverse_difference(adj_close_series.values, yhat, len(test_scaled_adj)
    # store forecast
    predictions.append(yhat)
    expected = adj_close_series.values[len(train_adj) + i + 1]
    print('Day=%d, Predicted=%f, Expected=%f' % (i+1, Predicted, expected))
```

```
/ARNING:tensorflow:Model was constructed with shape (10, 1, 1) for input KerasTens
)ay=1, Predicted=42944.335161, Expected=49321.652344
)ay=2, Predicted=44266.885412, Expected=49546.148438
)ay=3, Predicted=42211.111617, Expected=47706.117188
)ay=4, Predicted=42114.183111, Expected=48960.789063
)ay=5, Predicted=42581.505514, Expected=46942.218750
)ay=6, Predicted=41648.150257, Expected=49058.667969
)ay=7, Predicted=40407.924482, Expected=48902.402344
)ay=8, Predicted=40934.776243, Expected=48829.832031
)ay=9, Predicted=43161.517127, Expected=47054.984375
)ay=10, Predicted=47510.221911, Expected=47166.687500
)ay=11, Predicted=47082.576372, Expected=48847.027344
)ay=12, Predicted=47573.186845, Expected=49327.722656
)ay=13, Predicted=48484.820559, Expected=50025.375000
)ay=14, Predicted=50887.117791, Expected=49944.625000
)ay=15, Predicted=54732.051116, Expected=51753.410156
)ay=16, Predicted=53179.271911, Expected=52633.535156
)ay=17, Predicted=53340.418420, Expected=46811.128906
)ay=18, Predicted=55612.904008, Expected=46091.390625
)ay=19, Predicted=54141.314682, Expected=46391.421875
)ay=20, Predicted=56856.325126, Expected=44883.910156
)ay=21, Predicted=55421.626145, Expected=45201.457031
)ay=22, Predicted=56772.673018, Expected=46063.269531
)ay=23, Predicted=56694.069912, Expected=44963.074219
)ay=24, Predicted=60965.767021, Expected=47092.492188
)ay=25, Predicted=60265.564282, Expected=48176.347656
)ay=26, Predicted=60926.425667, Expected=47783.359375
)ay=27, Predicted=61395.972881, Expected=47267.519531
)ay=28, Predicted=63631.715135, Expected=48278.363281
)ay=29, Predicted=65365.565828, Expected=47260.218750
)ay=30, Predicted=61581.204127, Expected=42843.800781
)ay=31, Predicted=60885.642950, Expected=40693.675781
)ay=32, Predicted=60820.237433, Expected=43574.507813
)ay=33, Predicted=60304.324319, Expected=44895.097656
)ay=34, Predicted=62412.842861, Expected=42839.750000
)ay=35, Predicted=59779.881671, Expected=42716.593750
)ay=36, Predicted=57852.881909, Expected=43208.539063
)ay=37, Predicted=59994.076965, Expected=42235.730469
)ay=38, Predicted=61598.658298, Expected=41034.542969
)ay=39, Predicted=61261.976841, Expected=41564.363281
)ay=40, Predicted=60690.969874, Expected=43790.894531
)ay=41, Predicted=60377.812053, Expected=48116.941406
)ay=42, Predicted=62599.929627, Expected=47711.488281
)ay=43, Predicted=62339.920035, Expected=48199.953125
)ay=44, Predicted=60824.163920, Expected=49112.902344
)ay=45, Predicted=60498.288831, Expected=51514.812500
)ay=46, Predicted=60900.917339, Expected=55361.449219
)ay=47, Predicted=62700.510951, Expected=53805.984375
)ay=48, Predicted=66949.165183, Expected=53967.847656
)ay=49, Predicted=66343.039530, Expected=54968.222656
)ay=50, Predicted=64367.948506, Expected=54771.578125
)ay=51, Predicted=64320.275539, Expected=57484.789063
)ay=52, Predicted=63529.416054, Expected=56041.058594
)ay=53, Predicted=63848.031778, Expected=57401.097656
)ay=54, Predicted=64839.887587, Expected=57321.523438
)ay=55, Predicted=62928.475959, Expected=61593.949219
```

```
Day=56, Predicted=59534.773027, Expected=60892.179688
```

```
#Matching up the predictions with the correct day. The third prediction corresponds wi
#The date being 2021-08-24

all_btc.tail(90)
```

Date	Open	High	Low	Close	Adj Close	Volume
2021-08-24	49562.347656	49878.769531	47687.117188	47706.117188	47706.117188	3.536117e+10
2021-08-25	47727.257813	49202.878906	47163.613281	48960.789063	48960.789063	3.264635e+10
2021-08-26	49002.640625	49347.582031	46405.781250	46942.218750	46942.218750	3.266655e+10
2021-08-27	46894.554688	49112.785156	46394.281250	49058.667969	49058.667969	3.451108e+10
2021-08-28	49072.585938	49283.503906	48499.238281	48902.402344	48902.402344	2.856810e+10
...	...	...	...	...	...	...
2021-11-17	60139.621094	60823.609375	58515.410156	60368.011719	60368.011719	3.917839e+10
2021-11-18	60360.136719	60948.500000	56550.792969	56942.136719	56942.136719	4.138834e+10

Recommendations

Our recommendation is for everyday cryptocurrency investors to use our model as a baseline model and work from there. Whether that means actually coding to enhance the model or using the model in addition to other things to help make decisions when investing. The findings from this research also indicate that there will likely never be a truly accurate prediction model for cryptocurrencies or even stocks for that matter. This is because of something that we can learn from the model, that being that the price of bitcoin depends on far more than simply the recent prices and volumes.

1s completed at 2:32 PM