Predicting Stock Price using Machine Learning Algorithms

Abstract: Linear Regression and AR, MA, ARIMA models are used in study to forecast stock prices. This study uses daily closing prices for crypto currencies recorded from start date 2013/04/28 and end date 2018/05/05. Prices and dates are used as parameters to Linear Regression and ARIMA model. Each model attempts to predict whether a stock for next ten days are in line with closing prices used as parameters. The models used below shows results on 'Bitcoin' but as well could be any other trade currency.

1) Introduction:

Crypto currencies prediction has been a topic of interest for analysts and researchers for various reasons. This paper aims for predicting closing price for next ten days of the time series sequence.

2) Data source: https://coinmarketcap.com/currencies/bitcoin/historical-data/

3) Algorithms:

3.1) Linear Regression:

Dependencies: Libraries: 1) pandas, 2) numpy, 3) sklearn.linear_model

3.2) Process flow:

- Data scrapping: Extraction of data from the website and formation of data frame.

```
url = ("https://coinmarketcap.com/currencies/"+input+"/historical-data/?start=20130428&end=20180501")
uClient = uReq(url)
page_html = uClient.read()
page_soup = soup(page_html,"html.parser")
page_soup.find_all('tr',limit = 100)[0].find_all('th')
page_soup.find_all('th',{'class':"text-left"})[0].text
column_headers = [th.getText() for th in
            page_soup.findAll('tr', limit=100)[0].findAll('th')]
data_rows = page_soup.find_all("tr")[1:]
historic_data = [[td.getText() for td in data_rows[i].findAll('td')]
        for i in range(len(data_rows))]
historic_data_02 = []
for i in range(len(data_rows)):
    historic_row = []
for td in data_rows[i].findAll('td'):
    historic_row.append(td.getText())
historic_data_02.append(historic_row)
historic_data == historic_data_02
df = pd.DataFrame(historic_data,columns=column_headers)
df['Date'] = pd.to_datetime(df.Date).dt.strftime('%Y-%m-%d')
df.set_index('Date')
df['Close'] = df.Close.astype(float)
df['Close'] = pd.to_numeric(df['Close'])
df = df[['Close']]
```

Data frame generated with columns:

Date	Unnamed: 0	Open	High	Low	Close	Volume
2018-04-27	0	9290.63	9375.47	8987.05	8987.05	7,566,290,000
2018-04-26	1	8867.32	9281.51	8727.09	9281.51	8,970,560,000
2018-04-25	2	9701.03	9745.32	8799.84	8845.74	11,083,100,000
2018-04-24	3	8934.34	9732.61	8927.83	9697.50	10,678,800,000
2018-04-23	4	8794.39	8958.55	8788.81	8930.88	6,925,190,000

The columns used for Model building are ‘Date’ and ‘Close’.

How Linear Regression works:

Regression works on the information (input)

The variables used for predictions are called ‘predictors’ and output variables are called ‘response’. For the given study ‘Predictor’ = ‘Date and ‘response’ = Close (Closing price).

Theoretical equation for model becomes:

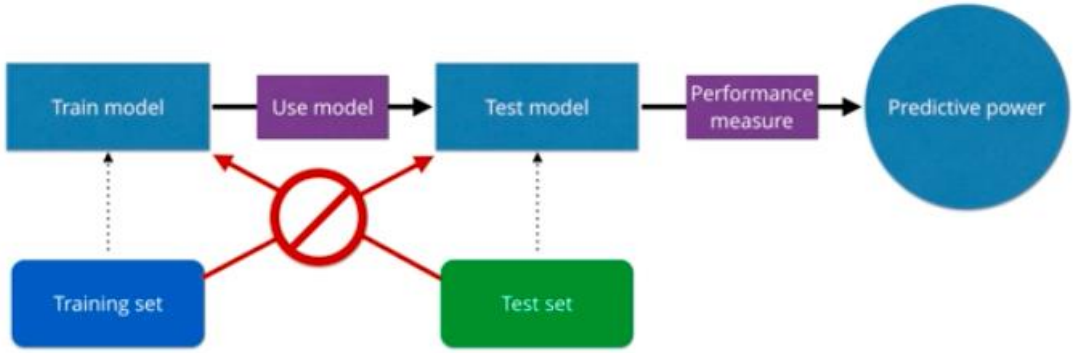
Close = Beta1 + Beta2 * Date

Where Beta1 and Beta2 are coefficients.

Beta1 and Beta2 are estimates from previous input and output.

3.3) Predictions and forecasting: The predictive model uses supervised learning.

- **How Predictive model works:**
- Division of dataset in training and test set
- Training set : Set used for model building
- Test set : Set used to evaluate performance of model.
- Sets are disjoint. **No overlap.**



Code for the dataset partition and fitting of model.

```
forecast_out = int(10)

df['prediction'] = df[['Close']].shift(-forecast_out)

print(df.head())


# defining Features and Label

x = np.array(df.drop(['prediction'],1))
x = preprocessing.scale(x)


x_forecast = x[-forecast_out:]
x = x[:-forecast_out]


y = np.array(df['prediction'])
y = y[:-forecast_out]

# Linear Regression

x_train,x_test,y_train,y_test = cross_validation.train_test_split(x,y,test_size = 0.2)


# Training

clf = LinearRegression()
clf.fit(x_train,y_train)

# Testing

confidence = clf.score(x_test,y_test)
print("confidence:",confidence)

forecast_prediction = clf.predict(x_forecast)
print(forecast_prediction)
```

The above code forecasts prediction for next 10 days as

forecast out = int(30)

3.4) Linear Regression: The data is split in train and test with (80:20) ratio. The training set contains known outputs, or prices, that model learns and test dataset is to test our model's predictions based on what it learned from the training set.

x_train,x_test,y_train,y_test = cross_validation.train_test_split(x,y,test_size = 0.2)

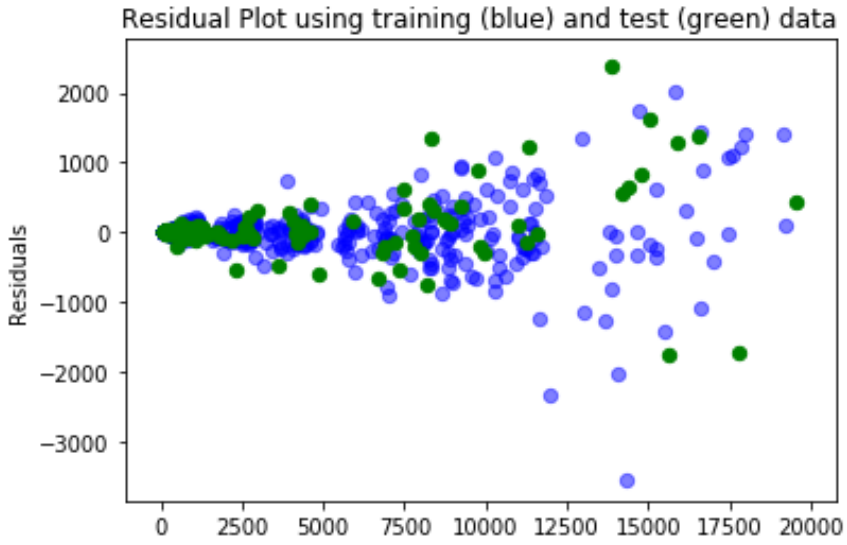
3.5) Accuracy of model: **r^2 (coefficient of determination**) is the measure used to check accuracy of model. The study case shows 97 % confidence in model.

```
In [5]: runfile('/Users/GD/spyder-py3/practice run.py', wdir='/Users/GD/spyder-
py3')
confidence: 0.972702138432
[ 8952.14467891  9310.62203938  9380.64528496  9203.57375475  9083.02665464]

In [6]:
```

Higher is the R^2 close to zero more is the accuracy of model.

3.6) Residual plots:



Residual plots are a good way to visualize the errors in data. If model is fitted good data should be randomly scattered around line zero. If the pattern is observed in data, it indicates that model is not capturing parameters with maximum efficiency.

3.7) Assumptions made while using Linear regression model:

- The relationship between the two variables is linear.
- The value of the independent variable is a set at various values, while the dependent variable is a random variable.
- The conditional distributions of the dependent variable have equal variances.
- If any interval estimation or hypothesis testing is done, additional required assumptions are:

➢ Successive observations of the dependent variable are uncorrelated.

➢ The conditional distributions of the dependent variable are normal distributions.

3.8) Shortcoming of linear regression:

- For the study mentioned in this thesis, the linear model works fine with accuracy of 97 % but linear regression is not the preferred method for

time series analysis**.** The scope of model is limited to two variables only with certain assumptions.

- Linear regression is sensitive to outliers, which affects prediction.

4) Machine learning algorithm II: Stationary models

AR, MA and ARIMA models are best approach to predict for time series analysis.

The step by step analysis of these models are explained below:

4.1) Test Stationary:

Definition: The stationarity is an essential property to define a time series process: A time series process is said to be covariance-stationary, or weakly stationary, if its first and second moments are time invariant.

The code shows plot and results of ad fuller test to check stationary:

```
def test_stationarity(timeseries):

    #Determing rolling statistics
    rolmean = pd.rolling_mean(timeseries, window=20)
    rolstd = pd.rolling_std(timeseries, window=20)

    #Plot rolling statistics:
    orig = plt.plot(timeseries, color='blue',label='Original')
    mean = plt.plot(rolmean, color='red', label='Rolling Mean')
    std = plt.plot(rolstd, color='black', label = 'Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.show(block=False)

### the Dickey-Fuller test is used to determine whether a unit root, a feature that can cause issues
### in statistical inference, is present in an autoregressive model.
### The formula is appropriate for trending time series like stock prices

    #Perform Dickey-Fuller test:
    print ('Results of Dickey-Fuller Test:')
    dftest = adfuller(timeseries, autolag='AIC')
    dfoutput = pd.Series(dftest[0:4], index=['Test static','P-Value','Lags', 'Number of observations'])
    for key, value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print (dfoutput)

test_stationarity(ts)
```

The results of Dickey fuller test produced are explained below:

4.2) Dicky Fuller test:

Dickey–Fuller test tests the null hypotheses that a unit root is present in an autoregressive model. Time series are stationary if it does not have trend or seasonal effects. Summary statistics calculated on the time series are consistent over time, like the mean or the variance of the observations.

When a time series is stationary, it can be easier to model. Statistical modeling methods assume or require the time series to be stationary to be effective. The purpose behind dickey fuller test is to determine how strongly the time series is defined by the trend.

The null hypothesis of the test is that a unit root can represent the time series, that it is not stationary (has some time-dependent structure). The alternate hypothesis (rejecting the null hypothesis) is that the time series is stationary.

- **Null Hypothesis (H0)**: If accepted, it suggests the time series has a unit root, meaning it is non-stationary. It has some time dependent structure.

- **Alternate Hypothesis (H1)**: The null hypothesis is rejected; it suggests the time series does not have a unit root, meaning it is stationary. It does not have time-dependent structure.

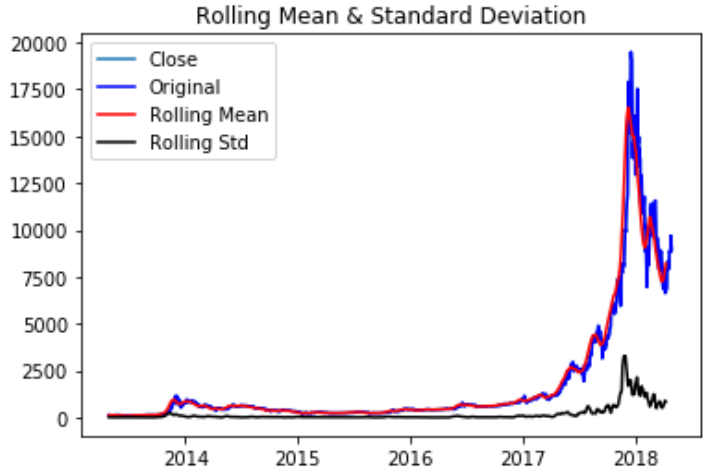
  We interpret this result using the p-value from the test. A p-value below a threshold (such as 5% or 1%) suggests we reject the null hypothesis (stationary), otherwise a p-value above the threshold suggests we accept the null hypothesis (non-stationary).

- **p-value > 0.05**: Accept the null hypothesis (H0), the data has a unit root and is non-stationary.

- **p-value <= 0.05**: Reject the null hypothesis (H0), the data does not have a unit root and is stationary.

  Model for the given study shows **p value = 0.1803** hence we accept the null hypothesis, data has unit root

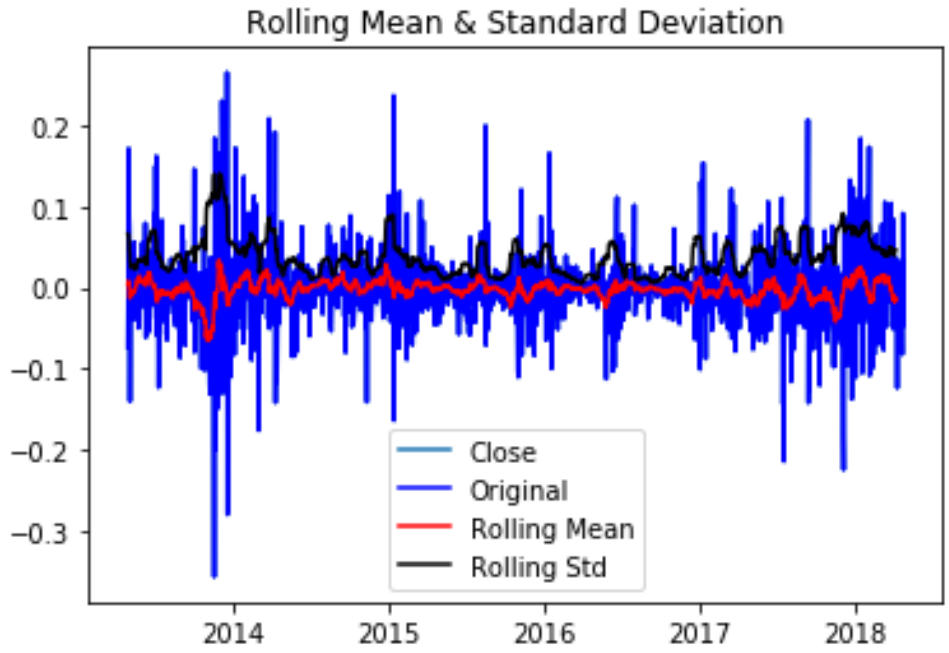
```
In [45]: runfile('/Users/GD/spyder-py3/Algo1.py', wdir='/Users/GD/spyder-py3')
```




```
Results of Dickey-Fuller Test:
Test static                  -2.274342
P-Value                       0.180377
Lags                         23.000000
Number of observations     1802.000000
Critical Value (1%)          -3.433984
Critical Value (5%)          -2.863145
Critical Value (10%)         -2.567625
dtype: float64
```

4.3) Exponential smoothing of time series:

Exponential smoothing refers to the use of an exponentially weighted moving average (EWMA) to “smooth” a time series. Looking at graph below from study model shows that smoothing significantly changes the original weight. The smaller the weight, the less influence each point has on smoothed time series**.**



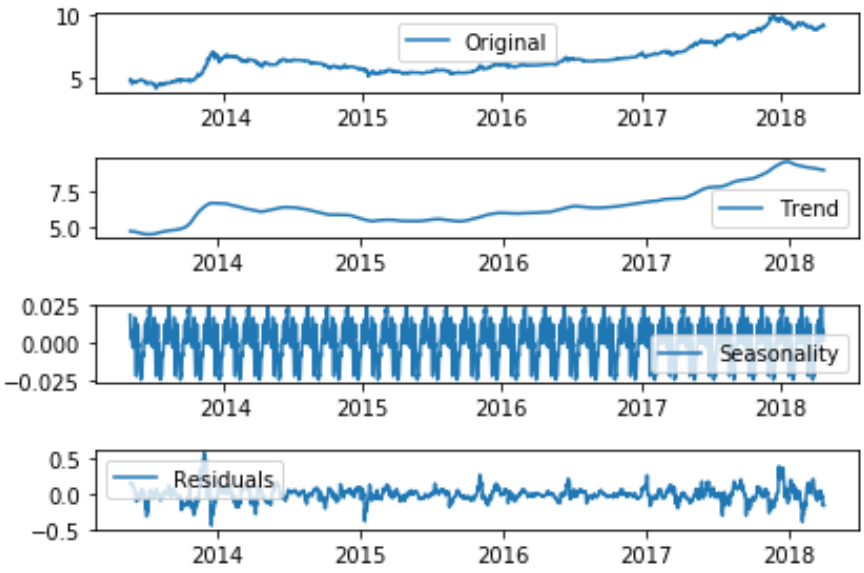

```
Results of Dickey-Fuller Test:
Test static                   -7.717691e+00
P-Value                        1.215846e-11
Lags                           2.200000e+01
Number of observations         1.802000e+03
Critical Value (1%)           -3.433984e+00
Critical Value (5%)           -2.863145e+00
Critical Value (10%)          -2.567625e+00
dtype: float64
```

4.4) Stationarity, Trend and seasonality:

There are many ways to characterize a time series, but we'll focus on three simple ones that are closely related: stationarity, trend, and seasonality. Stationarity refers to how stable the values of a time series are. For simplicity, let's just say that we consider a time series to be stationary if it has a constant mean. A stationary time series will not have any kind of increasing or decreasing pattern, and its points will generally hover around the same value, the mean. It's because of this characteristic that a simple EWMA, which estimates the mean, is so helpful for forecasts.

Trend refers to a long-term movement of a time series in a particular direction. With linear trend, time series points will approximately follow a line. It's also possible to have higher order trends, such as quadratic trend where points follow a parabola.

Graphs for trend in Study model for Bitcoin:



4.5) ACF and PACF model:

- **Significance of ACF and PACF graphs for ARIMA models:**

ARIMA stands for **Auto-Regressive Integrated Moving Averages**. The ARIMA forecasting for a stationary time series is nothing but a linear (like a linear regression) equation. The predictors depend on the parameters of the ARIMA model:

- **Number of AR (Auto-Regressive) terms (p):** AR terms are just lags of dependent variable. For instance if p is 5, the predictors for x (t) will be x(t-1)….x(t-5).
- **Number of MA (Moving Average) terms (q):** MA terms are lagged forecast errors in prediction equation. For instance if q is 5, the predictors for x(t) will be e(t-1)….e(t-5) where e(i) is the difference between the moving average at i^{th} instant and actual value.
- **Number of Differences (d):** These are the number of nonseasonal differences, i.e. in this case we took the first order difference. So either we can pass that variable and put d=0 or pass the original variable and put d=1. Both will generate same results.

An importance concern here is how to determine the value of 'p' and 'q'. We use two plots to determine these numbers.

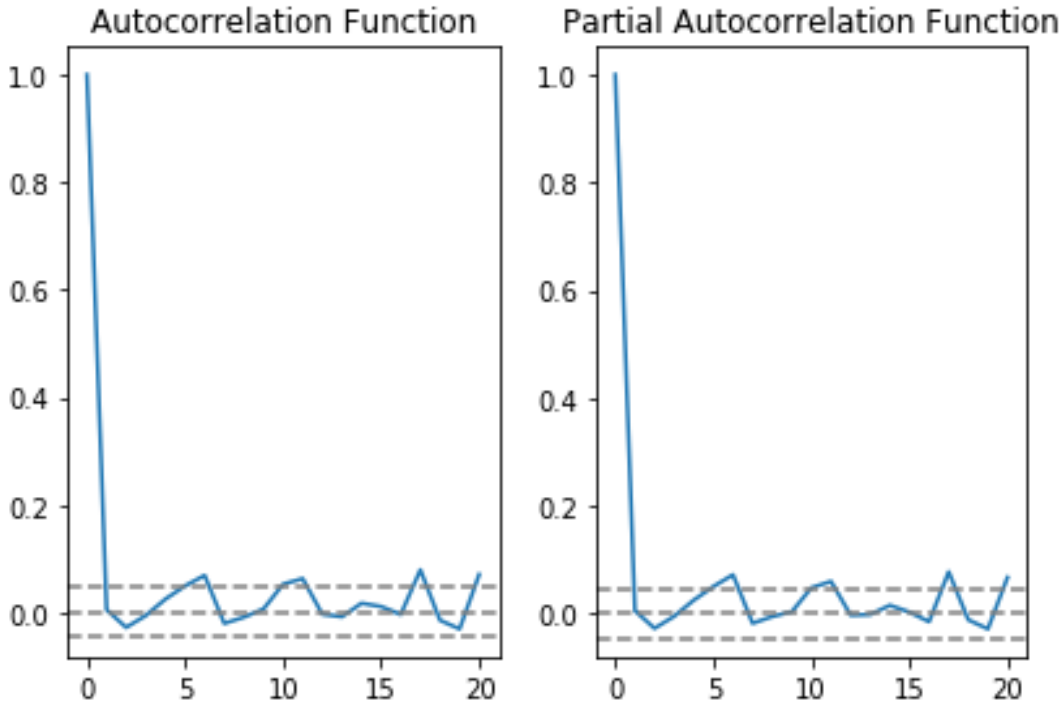
- **Autocorrelation Function (ACF):** It is a measure of the correlation between the time series with a lagged version of itself. For instance at lag 5, ACF would compare series at time instant 't1'…'t2' with series at instant 't1-5'…'t2-5' (t1-5 and t2 being end points).

- **Partial Autocorrelation Function (PACF):** This measures the correlation between the time series with a lagged version of itself but after eliminating the variations already explained by the intervening comparisons. for Ex:at lag 5, it will check the correlation but remove the effects already explained by lags 1 to 4.

Code :

```
#Plot ACF:
plt.subplot(121)
plt.plot(lag_acf)
plt.axhline(y=0,linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(ts_log_diff)),linestyle='--',color='gray')
plt.axhline(y=1.96/np.sqrt(len(ts_log_diff)),linestyle='--',color='gray')
plt.title('Autocorrelation Function')

#Plot PACF:
plt.subplot(122)
plt.plot(lag_pacf)
plt.axhline(y=0,linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(ts_log_diff)),linestyle='--',color='gray')
plt.axhline(y=1.96/np.sqrt(len(ts_log_diff)),linestyle='--',color='gray')
plt.title('Partial Autocorrelation Function')
plt.tight_layout()
```

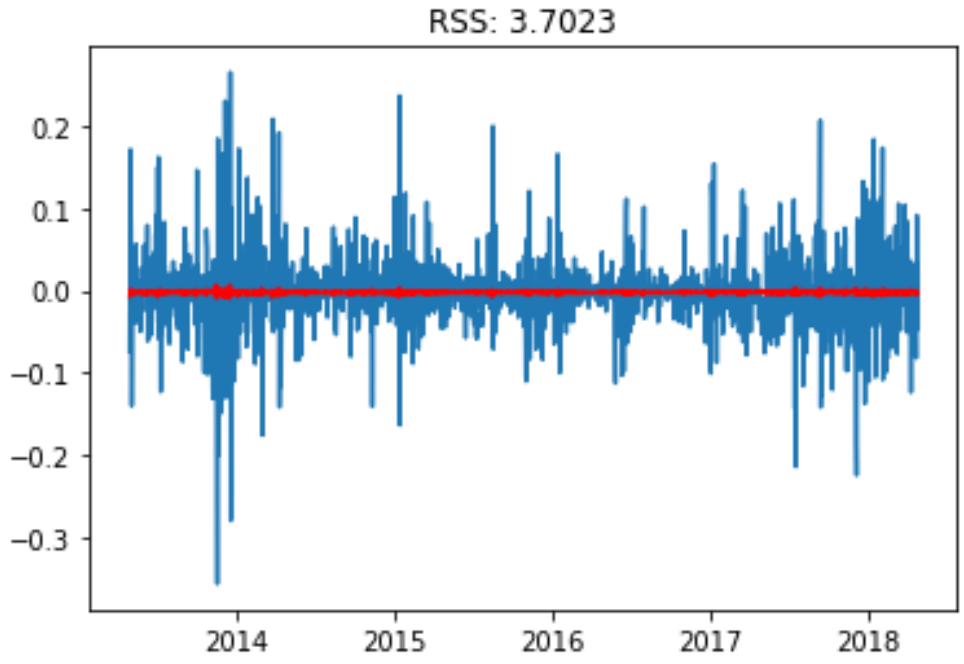


The p, d,q values can be specified using the order argument of ARIMA which take a tuple (p,d,q). Models below show 3 cases:

4.6) AR model:

```
#AR model:
model = ARIMA(ts_log, order=(2, 1, 0))
results_AR = model.fit(disp=-1)
plt.plot(ts_log_diff)
plt.plot(results_AR.fittedvalues, color='red')
plt.title('RSS: %.4f'% sum((results_AR.fittedvalues-ts_log_diff)**2))
```

```
...:
Out[47]: Text(0.5,1,'RSS: 3.7023')
```

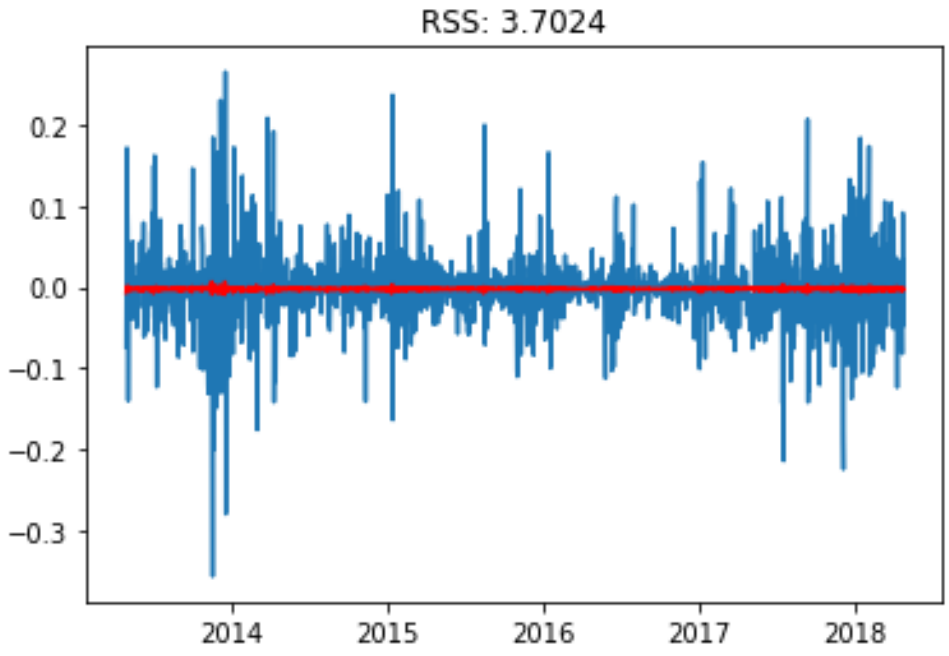


Note : RSS = 3.7023

4.7) MA model:

```
##MA model
model1 = ARIMA(ts_log, order=(0, 1, 2))
results_MA = model1.fit(disp=-1)
plt.plot(ts_log_diff)
plt.plot(results_MA.fittedvalues, color='red')
plt.title('RSS: %.4f'% sum((results_MA.fittedvalues-ts_log_diff)**2))
```

```
Out[48]: Text(0.5,1,'RSS: 3.7024')
```

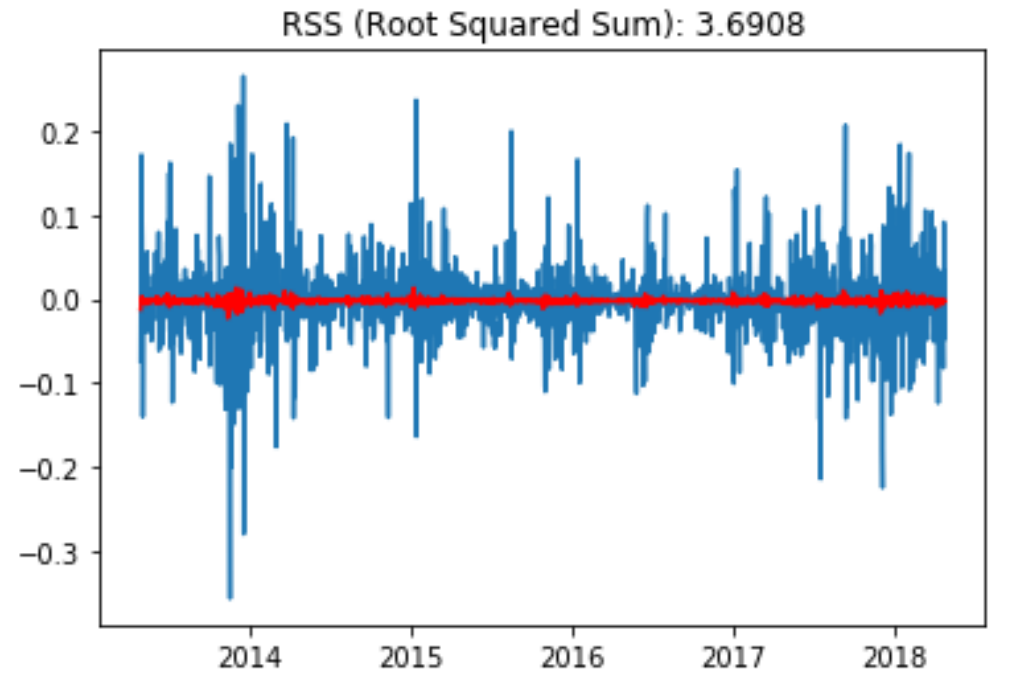


Note: RSS = 3.7024

4.8) ARIMA model (Combination model)

```
model2 = ARIMA(ts_log, order=(5, 1, 0))
results_ARIMA = model2.fit(disp=-1)
plt.plot(ts_log_diff)
plt.prredict(start='2013-04-28', end='2018-04-27')
plt.plot(results_ARIMA.fittedvalues, color='red')
plt.title('RSS (Root Squared Sum): %.4f'% sum((results_ARIMA.fittedvalues-ts_log_diff)**2))
```

```
...:
Out[50]: Text(0.5,1,'RSS (Root Squared Sum): 3.6908')
```



Note: RSS = 3.6908

Observation: Comparing RSS values for all three models, the combination model gives significantly better RSS result.

4.9) Final predictions:

Scaling the combination model back to original scale the prediction graph appeared as followed.

```
dtype: float64
Out[65]: [<matplotlib.lines.Line2D at 0x1c13f7ac50>]
```

The graph shows steep increase for prediction, so it is safe to predict that Bitcoin prices will shoot up for next quarter as well.(Prediction values are shown in DATA602-Assignment3.py)

5.0) Summary:

ARIMA model (in brief)

- Generalized random walk models fine-tuned to eliminate all residual autocorrelation
- Generalized exponential smoothing models that can incorporate long-term trends and seasonality

- Stationarized regression models that use lags of the dependent variables and/or lags of the forecast errors as regressors
- The most general class of forecasting models for time series that can be stationarized by transformations such as differencing, logging, and or deflating.

Source references:

- https://www.vividcortex.com/blog/exponential-smoothing-for-time-series-forecasting
- http://cs229.stanford.edu/proj2013/DaiZhang-MachineLearningInStockPriceTrendForecasting.pdf
- http://cs229.stanford.edu/proj2012/ShenJiangZhang-StockMarketForecastingusingMachineLearningAlgorithms.pdf
- http://cs229.stanford.edu/proj2015/009_report.pdf