GlobalLogic®

Artificial Intelligence in FinTech: Part 1

Authored by
Dmytro Vlasenko

Contents

Introduction

In India alone, about 100 million digital transactions happen every month, generating terabytes of data each day. Given this volume, identifying fraudulent transactions manually is no longer feasible and a well-trained AI model, with its continuous learning capabilities, can easily outperform any human analyst.

When searching for prospective clients for financing products, AI models produce accurate results and filter out probable defaulters. For clients looking to invest surplus income for maximum benefit, AI model's portfolio management features deliver.

In this white paper, we explore the applications of AI in all these avenues.

Cryptocurrency Portfolio Management

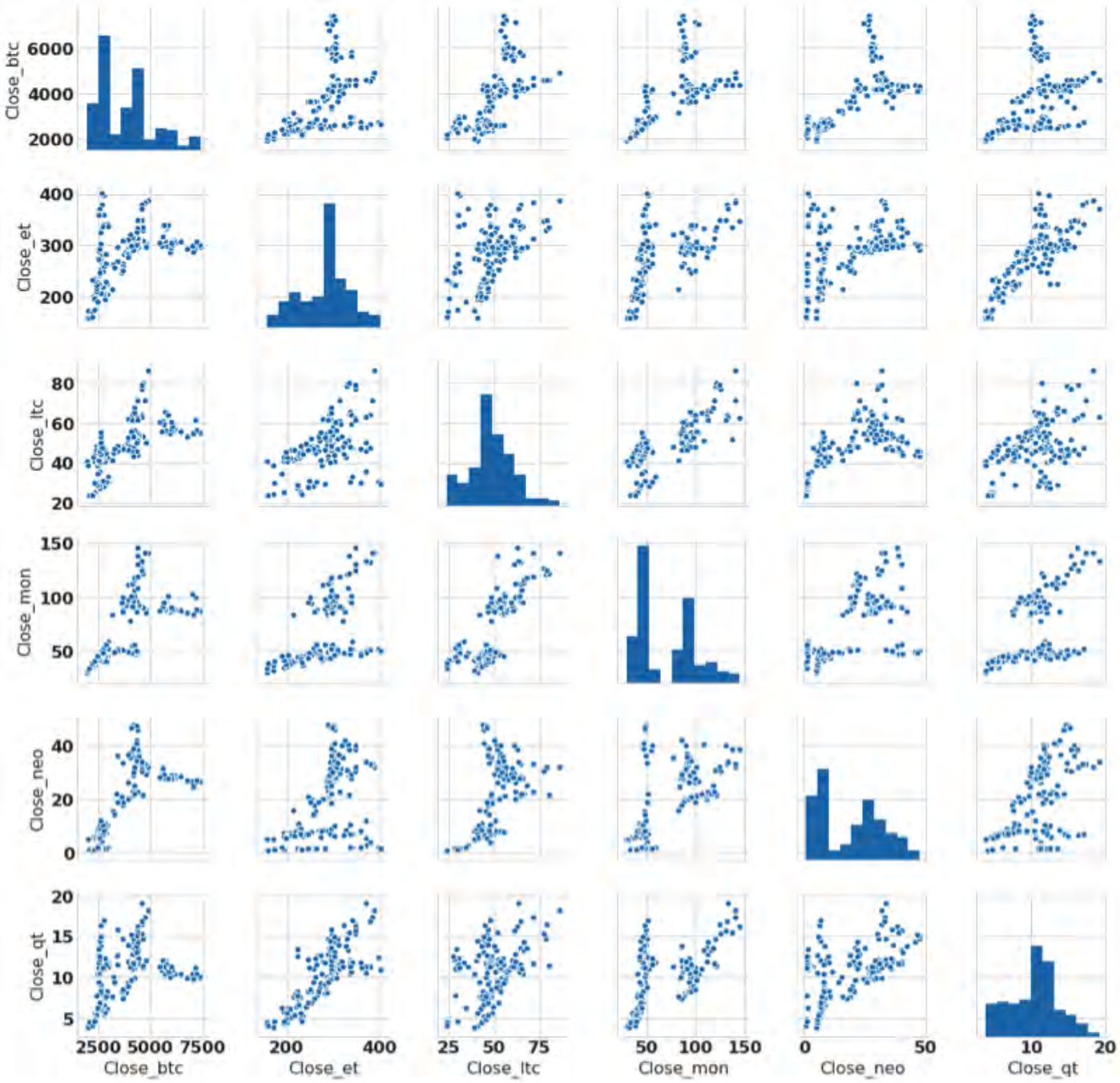
Any investment, especially a high risk one, needs a sound strategy for success. Investments as volatile as cryptocurrency are very difficult to track manually, but a well-trained machine learning model can rejig an investment portfolio 24/7 with continuous learning prowess. We will look at some methods to build successful cryptocurrency portfolios through the most popular cryptocurrency by market cap, Bitcoin, compared to other cryptocurrencies.

Correlation Paradigm – Bivariate Analysis

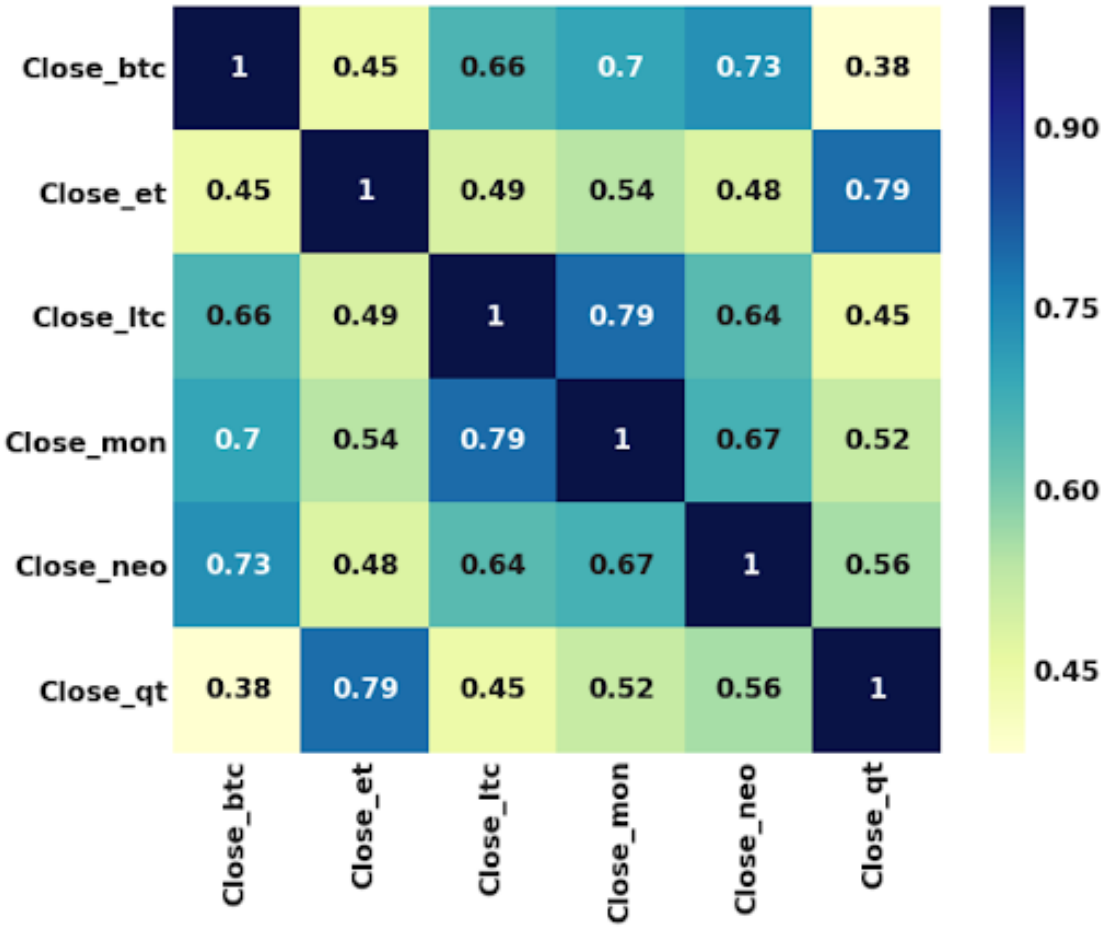
One of the most important aspects of any investment portfolio is its diversification. When it comes to money, putting all the eggs in one basket is never a good idea. This is where the correlation between different cryptocurrencies helps us select a few disjointed pairs. Such choices help to hedge against a loss in one cryptocurrency with a gain in the other. We used a dataset of previous prices of Bitcoin, Litecoin, Ethereum, Monero, Neo, Quantum, and Ripple to analyze the **Pearson Correlation** between them. The correlation coefficient is a number between -1 and 1, in which -1 signifies the cryptocurrencies are inversely correlated (i.e., an increase in one would trigger a decrease in the other), and +1 means that they have a positive correlation (one would increase or decrease as the other does). The merged dataset pruned to closing prices of cryptos looks something like this:

Close_btc	Close_et	Close_ltc	Close_mon	Close_neo	Close_qt
7144.38	294.66	61.30	99.76	26.23	11.21
7022.76	298.89	55.17	102.92	26.32	10.44
7407.41	296.26	54.75	86.35	26.38	10.13
7379.95	300.47	55.04	87.30	26.49	10.05
7207.76	305.71	56.18	87.99	26.82	10.38

When plotting the pair plots between these cryptocurrencies, a graph like the following is obtained:



From the above plots, it appears Bitcoin and Neo are highly correlated. Monero and Quantum also seem to have a high correlation. A heatmap does a good job quantifying the correlations between numerical variables by signifying the correlation with the use of colors. It also puts the actual correlation coefficient values on the correlation matrix.



The dark blue boxes show the most correlated currencies. Specifically, **Ethereum-Quantum (0.79)** and **Monero-Litecoin (0.79)** are the most correlated pairs. Also, **Neo-Bitcoin (0.73)** are highly correlated.

Thus, from a risk minimization point of view, investment should not be made in these pairs of cryptocurrencies since a decrease in one will likely occur simultaneously in the other. Instead, investment can be made in pairs that provide hedging capabilities where the correlation between the cryptocurrencies is negative.

The full notebook can be found here.

Predicting the Future Price Using Time Series Forecasting

Once the investment-worthy cryptocurrencies are chosen, the next step is to determine the right time and price to enter the market. This can only be done if we know the likely future price of the crypto. Here we will be taking the example of **Bitcoin** and trying to predict its future price using the following techniques of time series forecasting: **naive method, average method, moving average** and **ARIMA**.

Dataset

The dataset I have used for this analysis can be downloaded here. We put it into a Pandas DataFrame using the following piece of code:

btc = pd.read_csv("bitstampUSD_1-min_data_2012-01-01_to_2020-12-31.csv")

Once the dataset is loaded into a Pandas DataFrame, it looks something like this:

Timestamp	Open	High	Low	Close	Volume_(BTC)	Volume_(Currency)	Weighted_Price
1417411980	300.0	300.0	300.0	300.0	0.01	3.0	300.0
1417412040	300.0	300.0	300.0	300.0	0.01	3.0	300.0
1417412100	300.0	300.0	300.0	300.0	0.01	3.0	300.0
1417412160	300.0	300.0	300.0	300.0	0.01	3.0	300.0
1417412220	300.0	300.0	300.0	300.0	0.01	3.0	300.0

The dataset contains the Bitcoin price information for each second from January 1, 2012, to December 31, 2020.

The first step is to convert the Unix timestamp into a Python datetime object. This is achieved via the following line of code:

btc['Date'] = pd.to_datetime(btc['Timestamp'], unit='s')

Split Between Training and Test Dataset Resampling

The next step is to split the dataset between training and test datasets so we can measure the accuracy later on. In our particular case, we take the first 70% of observations as the training dataset and the last 30% of the observations as the test dataset.

Aggregation

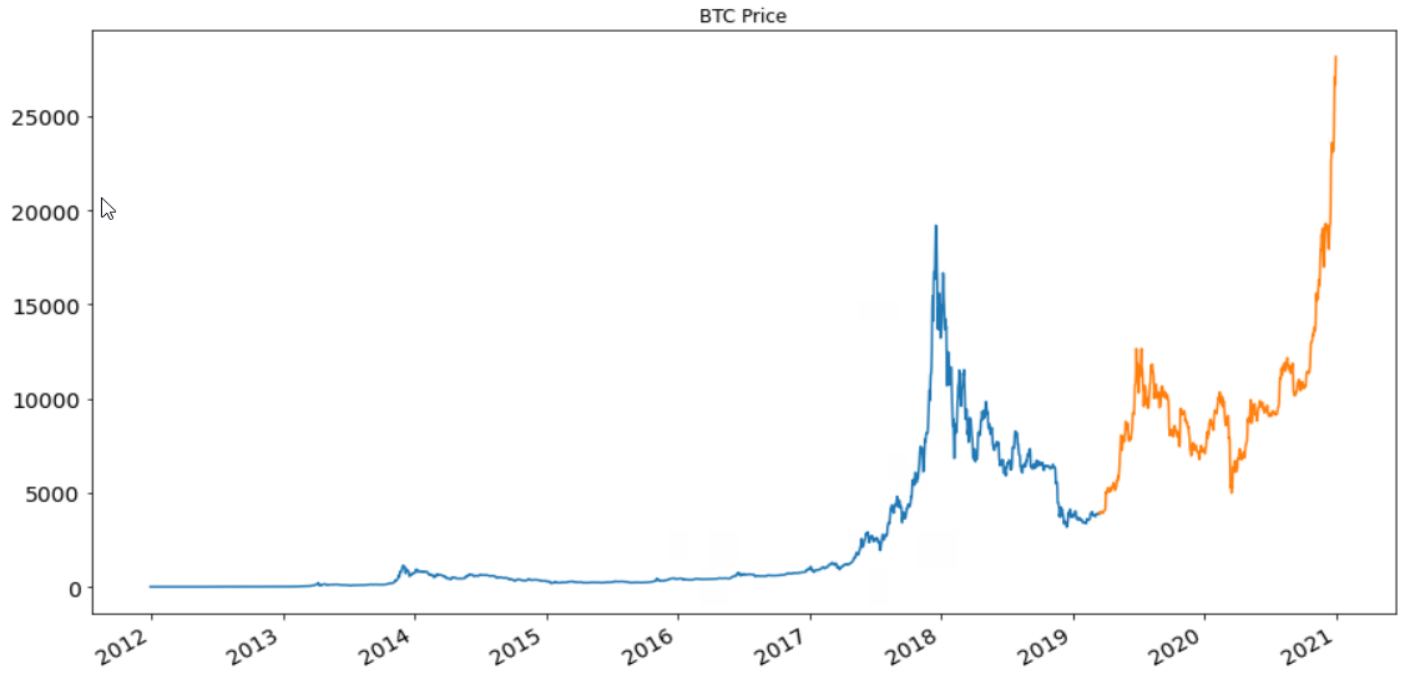
Our next step is to aggregate the data. Since we have the observations for every second, we can aggregate them for one day, which allows us to train our model faster. Additionally, the aggregated data is a good representation of the original data without any added bias. In order to aggregate the data, we first change the index of our dataframe to the date column. We then resample the date in our main dataset and our test and training data frame for one day.

btc.index = btc.Date

btc = btc.resample('D').mean()

Plotting the Training and Test Dataset

After that, we can plot the training and test dataset to visualize the Bitcoin price variation over time:



Naive Approach

The next step is to split the dataset between training and test datasets so we can measure the accuracy variation between individual time periods and remain mostly constant over time. Such a trend can be represented by the following equation:

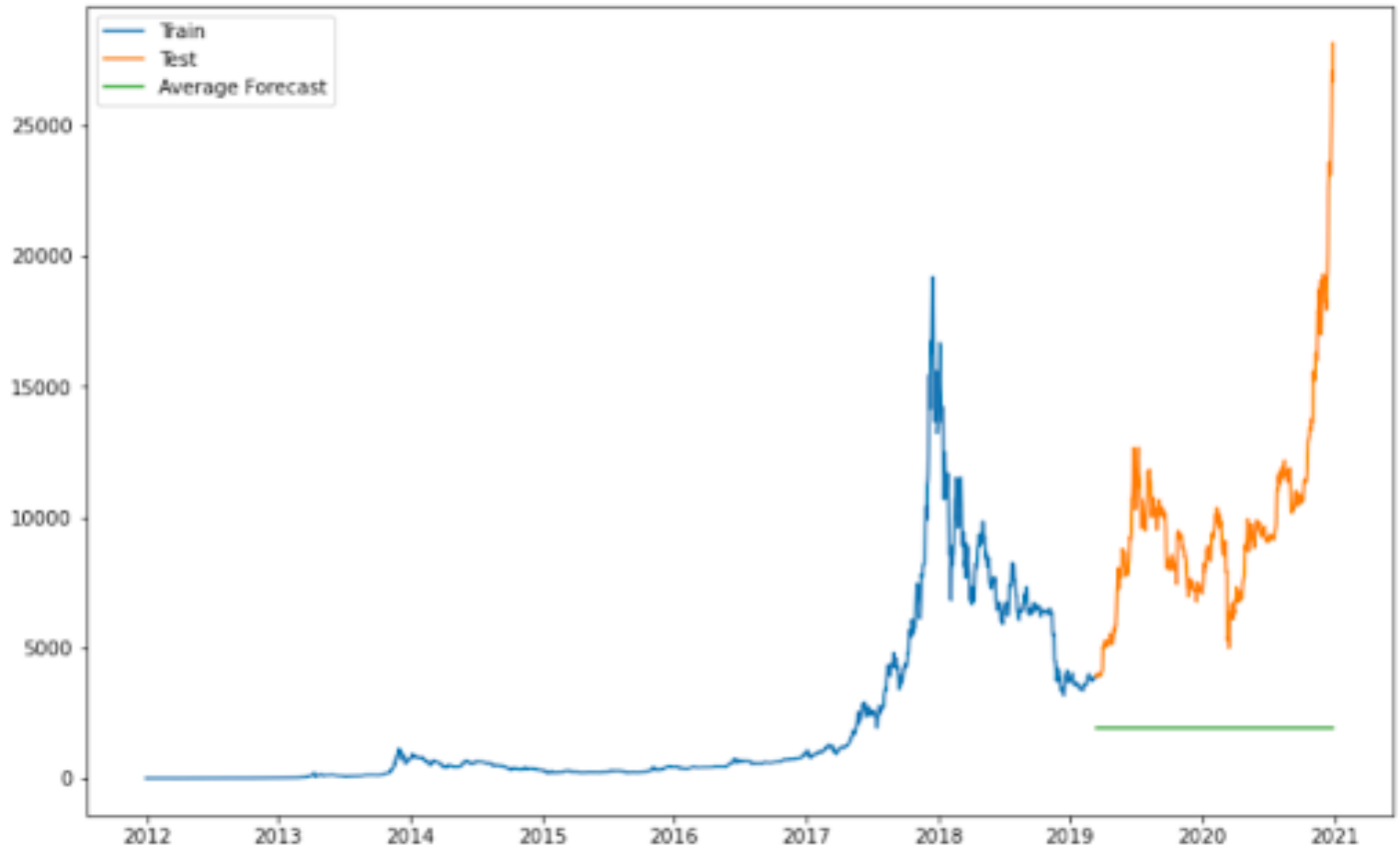
y(t) = ~y(t-1)

The next value can be forecasted by taking the last value. Taking the naive approach, we get a plot such as this:

As expected, the line of forecast is significantly different from the actual distribution. As a result, the root mean square error (RMSE) is 6981.48, which is very high. Therefore, we keep looking for newer methods.

Average Method

If the distribution varies mainly around the average of the dataset, then the average or mean can be a very good approximation for the forecast of values. If we use the average method forecast with our dataset, we see the following plot:



This method is clearly not very suitable for our distribution; this is confirmed by the high value of RMSE at 8697.548.

ARIMA Methodology

ARIMA methodology (Auto-Regression Integrated Moving Average) is a technique that makes use of both autoregression and a moving average to fit a distribution.

Autoregression

The autoregression model states that the value of distribution yt at time t depends only on its past values over a given time lag p or p past values. The equation can be represented as:

$$Y_t = b_0 + b_1 y_{t-1} + b_2 y_{t-2} + \ldots\ldots + b_p y_{t-p} + e_t$$

Here et is the error term associated with the regression. et represents the stochastic part of this regression, meaning it follows a purely white noise distribution where it represents any random residual that cannot be explained by the regression terms.

Moving Average

When Yt depends only on the random error terms of the last q hierarchical autoregression models, and those error terms follow a white noise process, the distribution is said to be described by a moving average (MA).

Representing the fact that for a **moving average**, Yt is a function of the error terms, we get:

$$Y_t = f(e_t, e_{t-1}, e_{t-2}, \ldots\ldots)$$

A common representation of a MA model that depends on only q of its previous error terms is as follows:

$$Y_t = b_0 + e_t + a_1 e_{t-1} + a_2 e_{t-2} + \ldots\ldots + a_q e_{t-q}$$

Here the error terms are assumed to have a white noise process, i.e., their mean is zero, and variance is constant across the distribution.

Stationarity of Time Series and Integration in ARIMA

A series is said to be strictly stationary if the mean, variance, and covariance of the time series Yt are irrelevant or constant. This means that a marginal distribution of Y at time t [p(Yt)] is the same at any other point in time. The condition to apply **ARIMA** is that the time series must be stationary, which is achieved via differentiation. If the nth order derivative is required to make a series stationary, then it is referred to as the integral of nth order. This is the word integral in the ARIMA model. This nth order derivative is referred to as d in the ARIMA model.

The reason why stationarity is important for time series analysis is that it prevents spurious regression and mistakes in calculation of autocorrelation.

ARIMA Explained

In a nutshell, the timeseries model depends on the p of its past values and q past values of white noise disturbances (i.e., both the **AR** and **MA** components) and is fitted on a stationary time series that is differentiated to the order d to make it stationary. Mathematically, this model can be represented as follows:

$$Y_t = b_0 + b_1 y_{t-1} + b_2 y_{t-2} + \ldots\ldots + b_p y_{t-p} + e_t + a_1 e_{t-1} + a_2 e_{t-2} + \ldots\ldots + a_q e_{t-q}$$

The parameters of the ARIMA model are defined as follows:

p: The number of lag observations included in the model, also called the lag order.

q: The size of the moving average window, also called the order of moving average.

d: The number of times that the raw observations are differentiated, also called the degree of differencing.

Side Note On Differencing

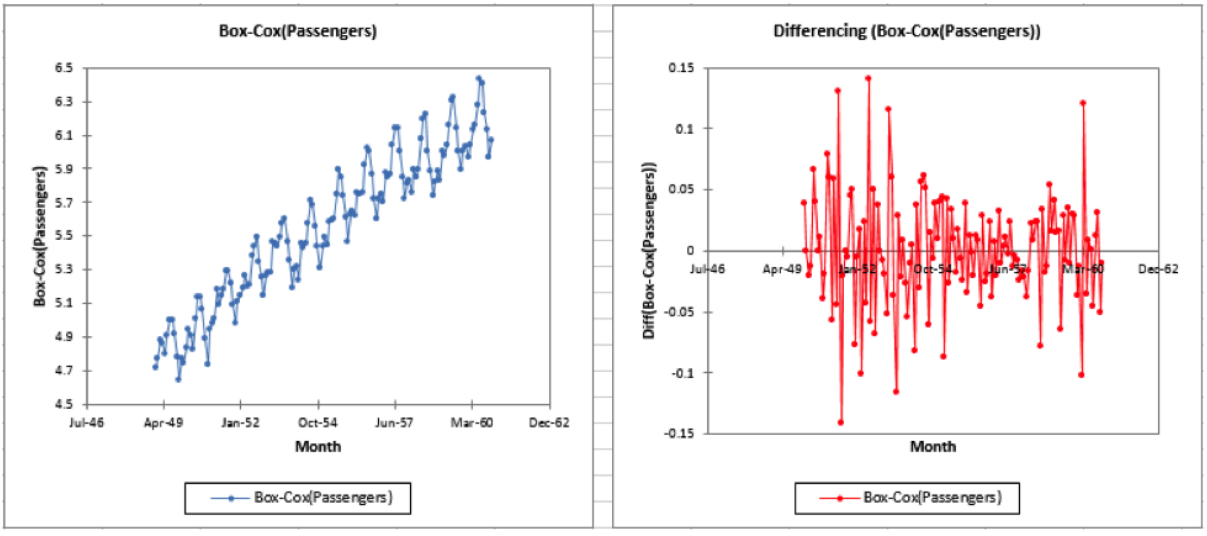
In order to make your series stationary, we take a difference between the data points. For example, if the original time series was:

X1, X2, X3,……..Xn

Your series with difference of degree 1 becomes:

(X2 – X1, X3 – X2, X4 – X3,…….Xn – X(n-1)

As a result of this, the difference between different time instants in the series becomes less prominent. If we see the difference on the curve reducing, we can continue to do 2nd and 3rd order differencing in order to make the series stationary. This can be further illustrated with the following diagram:



Fitting ARIMA To Bitcoin Data

We use the **statsmodels** library of Python to fit the ARIMA model on our univariate values of Bitcoin's closing price. The code snippet used to achieve this is as follows:

```
history = [x for x in btc_train['Close']]

predictions = list()
for t in range(len(btc_test)):

model = ARIMA(history, order=(5,1,0))

model_fit = model.fit(disp=0)

output = model_fit.forecast()

yhat = output[0]

predictions.append(yhat)

obs = btc_test.iloc[t,4]

history.append(obs)
```

Here, btc_train and btc_test are the training and test datasets created by dividing the original dataframe into a 80:20 ratio, where the first 80% of the observations in our dataset serve as the training data and the remaining 20% serve as the test data. We accomplish this as follows:

```
idx = math.floor(0.80*btc.count()[0])

btc_train = btc[0:idx]

btc_test = btc[(idx+1):(btc.count()[0]-1)]
```

On plotting the fitted model against our test dataset, we get the following graph:

We can see that our ARIMA model fits the dataset quite well, and we get an RMSE value of 306.025, which is significantly lower than that of the other time series method. Thus, we can predict the future BTC values by using the forecast method of the fitted model.

For the complete notebook, please visit here.

GlobalLogic®

GlobalLogic is a leader in digital product engineering. We help our clients design and build innovative products, platforms, and digital experiences for the modern world. By integrating strategic design, complex engineering, and vertical industry expertise, we help our clients imagine what's possible and accelerate their transition into tomorrow's digital businesses. Headquartered in Silicon Valley, GlobalLogic operates design studios and engineering centers around the world, extending our deep expertise to customers in the communications, automotive, healthcare, technology, media and entertainment, manufacturing, and semiconductor industries.



www.globallogic.com