POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

# Real-world model for Bitcoin price prediction

Author: **Francesco Re**

Student ID: 232687
Person code: 10664111
Academic Year: 2022-23

# Contents

# 1 | Introduction

## 1.1. Cryptocurrencies

Cryptocurrencies, have gained significant attention and popularity in recent years. The motivations behind this popularity can be attributed to several factors, one of which is that it is not reliant on any central authority, such as a government or bank, to uphold or maintain it, which offers increased security and privacy compared to traditional financial systems. Moreover, the potential for high profits, coming from the volatility of the price, has attracted many individual investors, further increasing the demand. However, this high volatility and dynamism also poses significant challenges when it comes to predicting future price movements using machine learning models.

## 1.2. Bitcoin vs. Stocks

There are a few reasons behind the fact that predicting stocks is easier than cryptocurrencies. For example, when trading stocks, there are many variables available to be used as features, like *P/E, ROE, ROCE, EBITDA* etc. Instead, when dealing with cryptos, only price, market cap and volume can be used for prediction. Furthermore, Bitcoin doesn't precisely adhere to past data, and patterns change rapidly, which reduces the accuracy of predictions.

## 1.3. Aim of this study

With that being said, there is an urgent need to produce a new **machine learning model** that can outperform previous models without requiring special tuning. The model must be capable of handling complex trends and patterns that change dynamically within a short period of time. It is also crucial for the algorithm to be resilient to **seasonality**, **missing values** and the presence of **outliers** in data. Additionally, it is important that the model does not require much prior knowledge or experience in forecasting time series data. The reference paper for this study is [1].

# 2 | Current models

There are many models and algorithms in the literature that are able to predict Bitcoin price. In order to create a new state-of-the-art model that could potentially achieve better accuracy, we analyzed two of them exploring their strengths and weaknesses.

## 2.1. ARIMA

The name stands for **A**uto**R**egressive **I**ntegrated **M**oving **A**verage, it is a very common model used in time series analysis. This model is really powerful since it is able to fully describe a stochastic process, provided that the process is stationary (see Appendix A).

**STRENGTHS** They are very robust and good at forecasting time series, exhibiting, in some cases, even better performance compared to deep learning models like LSTM [2].

**WEAKNESSES** They may not perform well in the presence of non-stationarity with sudden changes in trends or with seasonal data. Also, it is really difficult to train a model that can make long term predictions.

## 2.2. LSTM

**L**ong **S**hort-**T**erm **M**emory is a deep learning model, in particular, it's an evolution of the Recurrent Neural Network which, as the name suggests, has feedback connections that enable the network to process variable length sequences of data. LSTM can also be used to predict Bitcoin price in a different manner, in which a network is trained to forecast the sentiment of a set of tweets regarding Bitcoin, and based on that a prediction of the price is made.[3]

**STRENGTHS** They can obtain very high performance with a correct hyperparameters tuning.

**WEAKNESSES** They are challenging to understand and tune, and acquiring insight into their performance is demanding.

# 3 Dataset

The dataset used in this work can be found at `https://www.kaggle.com/team-ai/bitcoin-price-prediction/version/1`. It consists of a collection of 1556 records from 28th April 2013 to 31st July 2017 of the Bitcoin price.

## 3.1. Data format

Data is provided as a **csv** file with the structure described in Table 3.1.

| Column | Type | Example |
|---|---|---|
| Date | string | "Jul 31, 2017" |
| Open | float | 2763.24 |
| High | float | 2889.62 |
| Low | float | 2720.61 |
| Close | float | 2875.34 |
| Volume | string | "860,575,000" |
| Market Cap | string | "45,535,800,000" |

Table 3.1: Dataset format

## 3.2. Data used in this study

In this study, the columns **Open, High** and **Low** are discarded since our aim is to make a model that is able to predict the Bitcoin value on a weekly basis, as opposed to the so called **day trading** in which a trader buys and sells a financial instrument within the same trading day.[4]
Despite **Volume** and **Market Cap** could potentially carry precious information about the asset and therefore be used as features, in this case they are discarded because we want our model to be solely based on Bitcoin price.

# 4 | Implementation

The development of our model, based on **Prophet** library, goes through a series of steps reported below.

## 4.1. Data import

As seen in chapter 3 data is provided in a `csv` file, so we want to import the file, remove the unwanted columns and rename some of them with a better name using `pandas` library.

```
df = pd.read_csv("dataset/bitcoin_price_train.csv", index_col="Date",
↪  parse_dates=True)

df = df[["Close"]].rename(columns={"Close": "price"})
```

As soon as our data is uploaded in a dataframe, we want to take a look at the data by plotting it with the support function `plot_series(...)`.

```
plot_series(df['price'], title='Bitcoin price', xlabel='Date', ylabel='Price')
```

The result is shown in Figure 4.1

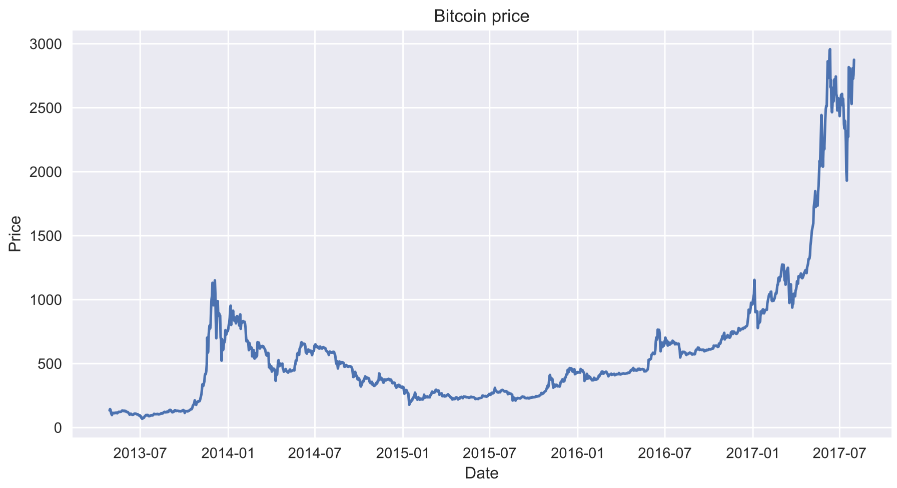

Figure 4.1: Bitcoin price

## 4.2. Exploratory analysis

Before starting building our model, we did some rough analysis by plotting different views of the dataset. This is useful to obtain an initial understanding of the data's trends and patterns.

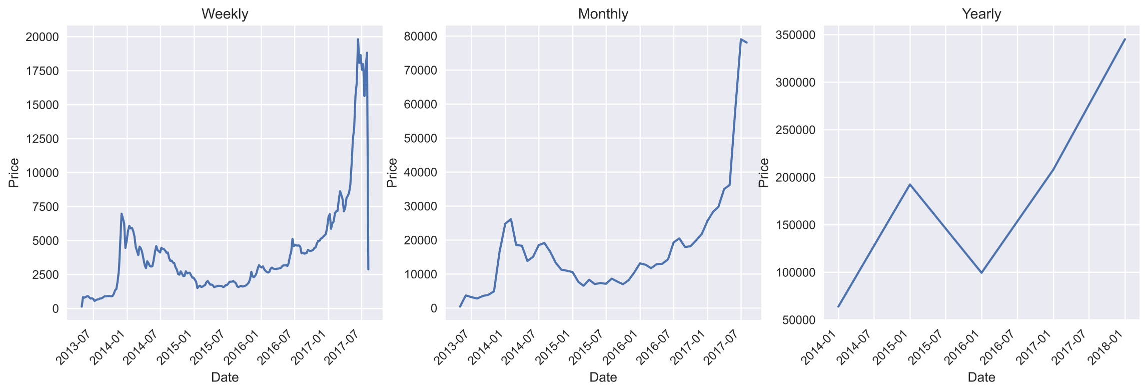

Figure 4.2: Weekly, monthly and yearly resamples

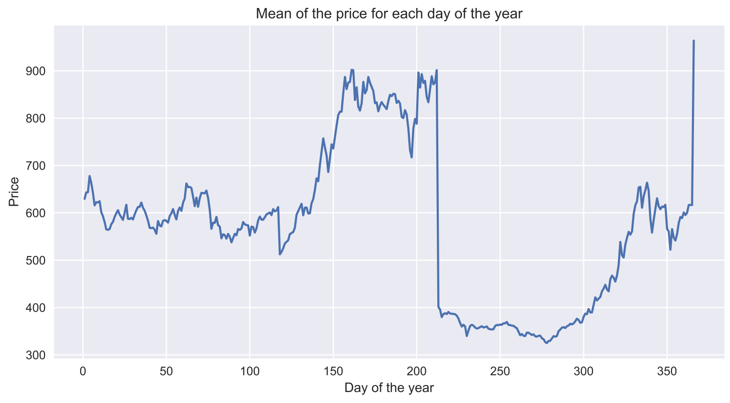

Figure 4.3: Mean of the price for each day of the year

## 4.3. Naive forecast

As a baseline for our evaluations we're building a naive model. Naive forecasting models **rely only on historical observations**, considering the events of the previous period to predict a repeat occurrence. In practice, there are two ways to accomplish this:

- $Y(t+1) = Y(t)$ considers just the previous value
- $Y(t+1) = Y(t) + (Y(t) - Y(t-1))$ considers also the trend of the previous values

```
# naive forecast 1
# Y(t+1) = Y(t)
df['naive_forecast_1'] = df['price'].shift(1)
fig, _ = plot_series(df[['price', 'naive_forecast_1']], title='Naive forecast vs
↪  Actual value', xlabel='Date', ylabel='Price', labels=['Actual value', 'Naive
↪  forecast'])
```

```python
save_fig('07_naive_forecast_1', fig)

# naive forecast 2
# y(t+1) = y(t) + (y(t) - y(t-1))
df['naive_forecast_2'] = df['price'] + (df['price'] - df['price'].shift(-1))
fig, _ = plot_series(df[['price', 'naive_forecast_2']], title='Naive forecast vs
↪  Actual value', xlabel='Date', ylabel='Price', labels=['Actual value', 'Naive
↪  forecast'])
save_fig('07_naive_forecast_2', fig)

# compute mean squared error of naive forecast 1
df.dropna(inplace=True)
MSE_1 = mean_squared_error(df['price'], df['naive_forecast_1'])
RMSE_1 = np.sqrt(mean_squared_error(df['price'], df['naive_forecast_1']))

# compute mean squared error of naive forecast 2
df.dropna(inplace=True)
MSE_2 = mean_squared_error(df['price'], df['naive_forecast_2'])
RMSE_2 = np.sqrt(mean_squared_error(df['price'], df['naive_forecast_2']))

print(f"{MSE_1=}")
print(f"{RMSE_1=}")

print(f"{MSE_2=}")
print(f"{RMSE_2=}")
```

The two approaches are very similar, also for what concerns the results. Indeed, we obtain $MSE = 1386.3$ and $RMSE = 37.2$ for the first approach and $MSE = 1378.3$ and $RMSE = 37.1$ for the second. Despite the marginal improvement of the latter, the overall outcome is not impressive, as expected from a baseline model. Additionally, it is important to note that this approach is limited to forecasting within a **one-day time horizon**.

## 4.4. Stationarity test

In order to understand if the Bitcoin price can be forecasted with a good accuracy we need to make sure it is stationary. We're applying a statistical tool known as **ADFuller test** (see section A.2). Firstly, we need to apply a log transformation to the price column of the dataframe and then compute the 7 days rolling mean of that value. The difference between those values is plotted in Figure 4.4.

Now, we apply the ADFuller test on the last value we computed.

```python
# apply adfuller test to check if the series is stationary
result = adfuller(df['log_diff'].dropna())
print(f"ADF Statistic: {result[0]}")
```

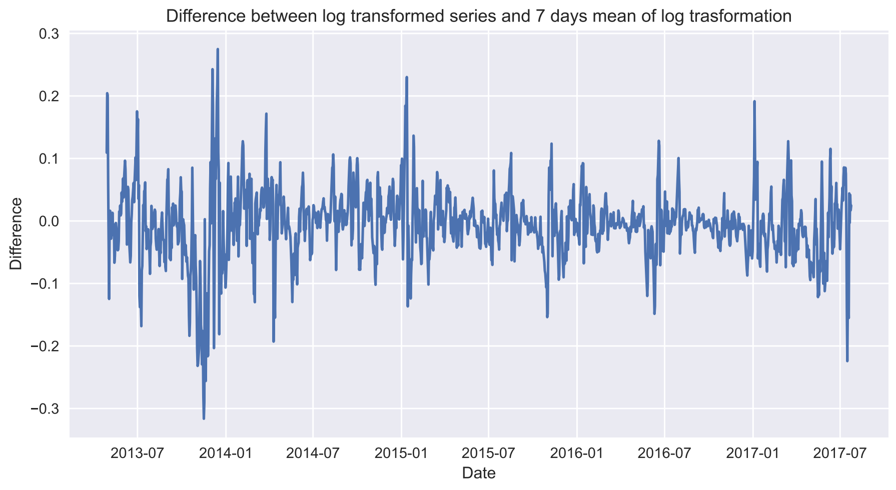

Figure 4.4: Log difference

```
print(f"p-value: {result[1]}")
print(f"Critical Values:")
for key, value in result[4].items():
    print(f"\t{key}, {value}")
```

The results of the ADFuller test are the following.

```
ADF Statistic: -7.188368504758797
p-value: 2.541567012646908e-10
Critical Values:
        1%, -3.4346424863111396
        5%, -2.8634358661653803
        10%, -2.5677793320839823
```

We are obtaining a **very small p-value**, this means that there is a strong evidence that the time series is **stationary**. Now we are ready to train the FbProphet model.

## 4.5. Train Prophet model

**Prophet**, an open-source toolkit developed by Facebook, is designed for univariate time series forecasting. It employs a **Bayesian-based curve fitting method** to predict time series data. What sets Prophet apart is its ability to detect seasonal trends in data without the need for prior knowledge or expertise in time series forecasting. It offers a set of easily understandable parameters and can effectively handle known holidays, missing data, and outliers. Prophet is resilient to missing data and trend shifts, and generally performs well in handling outliers.
The FbProphet library is pretty straightforward, requiring just a simple data preparation before training. The code is the following.

```python
# prepare data for prophet
df['ds'] = df.index
df['y'] = df['price']

#build model
model = Prophet()

# train
model.fit(df)

# predict
df_forecast = model.predict(model.make_future_dataframe(periods=500, freq='D'))

# plot forecast
model.plot(df_forecast)
model.plot_components(df_forecast)
```

The result of the prediction is plotted in Figure 4.5. It is clearly visible that the prediction confidence decreases as the forecast horizon increases.

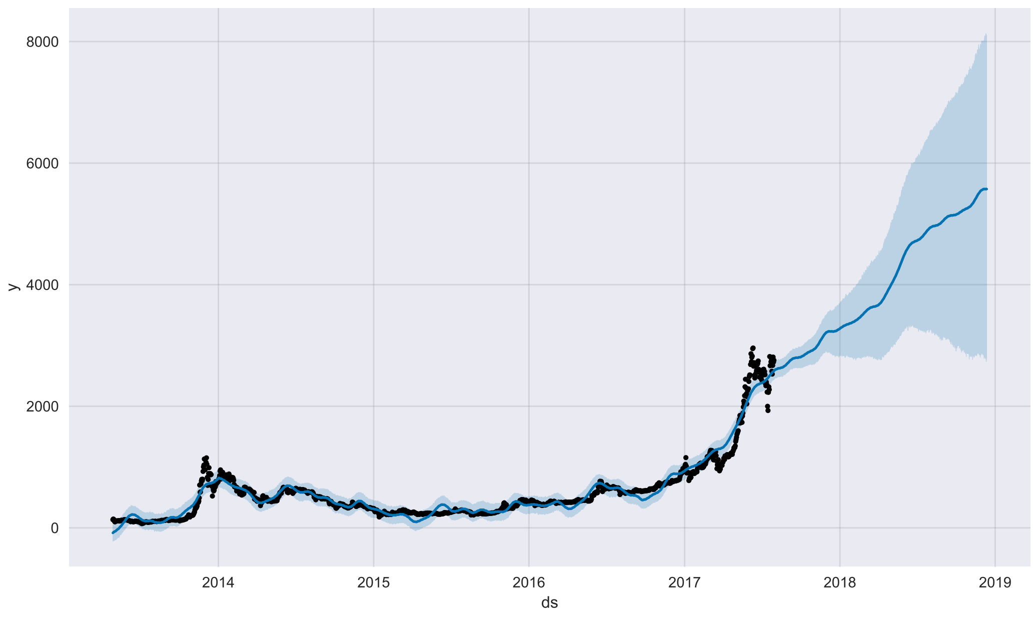

Figure 4.5: Prophet forecast

## 4.6. Evaluate final model

After training the model with FbProphet, the next step is to evaluate its performance, which can be done by utilizing the `cross_validation` function provided by the FbProphet library.

```python
# perform cross validation
df_cv = cross_validation(model, horizon='365 days')
plot_cross_validation_metric(df_cv, metric='rmse')
```

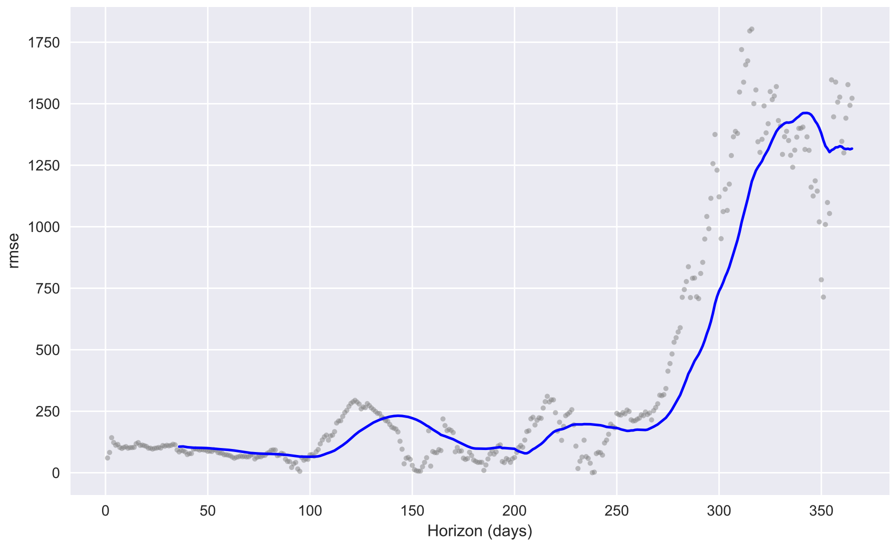

Figure 4.6: Cross validation evaluation of the model,

As we can see in Figure 4.6 the RMSE increases significantly when the prediction horizon is greater than 250. Hence, it becomes more and more unpredictable as soon as the forecast period increases, this is obviously due to the high volatily of the cryptocurrencies.

# 5 | Conclusions and future developments

The purpose of this study was to develop a machine learning model for predicting the price of Bitcoin. We explored different models, including ARIMA and LSTM, and identified their strengths and weaknesses. We then implemented a model using the FbProphet library, which is specifically designed for time series forecasting.

The evaluation of our model showed promising results, with accurate predictions for short-term forecasts. However, as the forecast horizon increased, the accuracy decreased, which is expected due to the high volatility of cryptocurrencies.

In conclusion, our model demonstrates the potential for predicting the price of Bitcoin. However, there is room for further improvement, especially in handling long-term forecasts and incorporating additional features such as volume and market capitalization. Future developments could also involve exploring other machine learning algorithms and incorporating sentiment analysis from social media data to enhance the predictive capabilities of the model. The code and the resources of this entire project can be found at this repository[1].

[1]https://github.com/francesco-re-1107/bitcoin-price-prediction

# Appendices

# A | Time Series concepts

A time series is often modeled with a discrete-time stochastic process, the reason is that there exists a lot of different statistical tools for stochastic processes that could help us understand better the time series.

## A.1. Stochastic process

A **stochastic process** (SP) is a mathematical object usually defined as a **sequence of random variables**, where the index of the sequence have the interpretation of time [5].
An example could be $Y(t)$ where for a given time instant $\bar{t}$ the following is a random variable $Y(\bar{t}) \sim \mathcal{N}(\mu, \sigma^2)$

### Mean of SP

The mean function of a stochastic process $Y(t)$ is defined as the expected value of the random variable at a given time $t$.

$$m(t) = E[Y(t)] \tag{A.1}$$

### Covariance of SP

The covariance function of a stochastic process $Y(t)$ is defined as

$$\gamma_Y(t_1, t_2) = E[(Y(t_1) - m(t_1))(Y(t_2) - m(t_2))] \tag{A.2}$$

If the process is stationary (see section A.2) then the covariance function can be defined as

$$\gamma_Y(\tau) = E[(Y(0) - m(0))(Y(\tau) - m(\tau))] \tag{A.3}$$

## A.2. Stationarity of SP

### Intuitive definition

Intuitively, stationarity means that the statistical properties of a process generating a time series do not change over time. It does not mean that the series does not change over time, just that the way it changes does not itself change over time. [6]

## Formal definition

A stochastic process $Y(t)$ is said to be **wide-sense stationary** if Equation A.4 and Equation A.5 are satisfied

$$E[Y(t)] = m, \forall t \text{ with m constant} \tag{A.4}$$

$$\gamma_Y(t_1, t_2) = \gamma_Y(t_3, t_4) \;\; \forall t_1, t_2, t_3, t_4 \text{ with } t_2 - t_1 = t_4 - t_3 \tag{A.5}$$

## ADFuller test

The **A**ugmented **D**ickey–**F**uller test (ADF) is the most common statistical tool used to test whether a given time series is stationary or not. It does so by checking the presence of a unit root, which indicates non-stationarity.

The null hypothesis of the test is that the time series has a unit root, and the alternative hypothesis is that it does not. The result of the test is the so called p-value. A **small p-value** (typically less than 0.05) indicates that there is strong evidence to reject the null hypothesis and conclude that the **time series is stationary**, while a **high p-value** suggests that the null hypothesis cannot be rejected, and the **time series is non-stationary**.

# Bibliography

[1] R. K. Rathore, D. Mishra, P. S. Mehra, O. Pal, A. S. HASHIM, A. Shapi'i, T. Ciano, and M. Shutaywi, "Real-world model for bitcoin price prediction," 2022.

[2] A. Azari, "Bitcoin price prediction: An arima approach," 2019.

[3] H. GB and S. N. B, "Cryptocurrency price prediction using twitter sentiment analysis," 2023.

[4] Wikipedia, "Day trading — Wikipedia, the free encyclopedia." `http://en.wikipedia.org/w/index.php?title=Day_trading`.

[5] Wikipedia, "Stochastic process — Wikipedia, the free encyclopedia." `http://en.wikipedia.org/w/index.php?title=Stochastic_process`.

[6] S. P. Affek, "Stationarity in time series analysis — Towards Data Science." `https://towardsdatascience.com/stationarity-in-time-series-analysis-90c94f27322`.