Crypto Trading Strategies - Implementation, Backtesting & Transfer Learning

Final project presentation - ML/DL financial application

Arian NAJAFY ABRANDABADY - Lucas RODRIGUEZ - Bastien TRIDON

May 16th, 2023

Master in Quantitative Finance (M2QF)

⇝ Overview of academic deliverables

As of today, 6 documents :

1. Technical **report**
2. **Slides**
3. **GitHub repository** [1] [2]
4. **Online project homepage** [3]
5. **Gantt charts**
6. **Scoping notes** (summary & matrix)

1. https://github.com/lcsrodriguez/cryptotrading
2. https://github.dev/lcsrodriguez/cryptotrading
3. https://lcsrodriguez.github.io/qf/ml/

Quick outline

Introduction

Problem definition

Context

- Crypto. financial market $\mathcal{M}$ on $(\Omega, \mathcal{F}, \mathbb{F} := (\mathcal{F}_t)_{t \in \mathbb{R}^+}, \mathbb{P})$
- OHLCV of trades data for several assets

Objectives

- State-of-the-art of cryptocurrencies trading strategies & Algo. trading techniques
- Implementation of a crypto trading strategy for **BTC** using ML/DL techniques
- Results analysis & Backtesting on existing datasets
- Use of transfer learning on **ETH** for instance
- Final analysis & Conclusions

Constraints

- Python & Jupyter Notebook
- Clean pre-processing, ML usage for **only relevant cases** [4]
- Adoption of a highly-professional framework

4. Otherwise, use of non-ML techniques as ARIMA, ...

Financial framework

Hypothesis of research project

1. Short-selling authorized
2. Underlying perfectly divisible
3. Friction-less market
4. High-frequency data without **market microstructure noise**[5]
5. Close price $(C_t)_{t \geq 0}$ considered as the **transaction price**

5. Due to market participants behaviors footprints, trade operations, LOB movements, ...

Technical framework

Development environment

General informations

- Dataset from Kaggle competition
- Development : **Python 3.10+**
- Environment : **Jupyter Notebook** *(local or Kaggle)*
- Dependencies tracking : **pip**[6] & **Dependabot**
- Version control : **Git/GitHub**[7]
- LaTeX report writing
- Data handling & Numerical analysis : **NumPy**, **Pandas** & **PyArrow**
- Plotting : **Matplotlib**, **Pyplot** & **Seaborn**
- UML & Class diagram **Pyreverse**
- CI/CD workflow : GitHub Actions *UML, Dependabot, release*

$\Longrightarrow$ Professional development framework for **best implementation quality**

6. See complete list of dependencies on GitHub
7. GitHub project repo : https://github.com/lcsrodriguez/cryptotrading

Pre-processing & Data preparation

Dataset description & Data Processing

Original dataset

- 2.5+ GB CSV files of trades data
- Date time range : **2018-01-01 00 :01 :00** to **2022-01-24 00 :00 :00**
- Frequency : **1-min** sampling period
- Columns : OHLCV data, VWAP, Trading activity (*Count*)

Pre-processing

1. Removing previous split : **Train**, **Train2** & **Test**
2. Re-constructing clean data files combining **Train** & **Train2** *Concatenation*
3. Removing NaN and $\pm\infty$ values
4. Type-casting **Count** ($\in \mathbb{N}$) & **Datetime**[8]
5. Removing useless precision $\rightsquigarrow$ Compression
6. On-disk saving as text and binary files
   - 13 files for each cryptocurrency + 1 file for all
   - Formats : **CSV** + **Apache Parquet**[9]

---

8. Timestamp conversion with minute precision as Pandas object
9. Interesting file compression ratio (binary files) : $\tau \sim 3$

Pre-processing & Data preparation

Data Processing

State-of-the-art

- Use of 10-min data (additional resampling) [10]
- Adding financial indicators from **technical analysis**
  - RSI
  - Moving Averages
  - Bollinger bands
- $\varnothing$ data aggregation from external sources to avoid new NaN values

Observation Tradeoff between **Computational speed** & **Classification accuracy**

Solution (Data)

~~Tick data~~ $\longrightarrow$ 1-min data $\longrightarrow$ 10-min data $\longrightarrow$ 20-min data $\longrightarrow$ $\boxed{\textbf{30-min data}}$

$\rightsquigarrow$ *Potential loss of information* ***but*** *related to some custom strategy limitations* [11]

10. Most relevant tradeoff of **compression** & **data importance**
11. As limit number of trades per hour for ex.

Enhancement of trading strategy engine

Use of ML/DL prediction

$\longrightarrow$ Use of ML/DL to :

- Enhance our strategy
- Features importance analysis for better selection

$\rightsquigarrow$ 2 solutions :

1. Price forecasting (*regression*)
2. Price movement prediction (*binary or ternary classification*)

Studied solution : Price movement prediction $\longrightarrow$ **UP**/**DOWN**

- ML $\rightsquigarrow$ Logistic Regression [12] & Boosting (tree-based method) *sklearn*/*xgboost*
- DL $\rightsquigarrow$ LSTM [13] (Recurrent Neural Network) *keras*

12. As our baseline model
13. Long-Short-Term-Memory : RNN used in timeseries forecasts

Trading strategies on **BTC**

Protocol outline

1. Dataset pre-processing
2. Indicators selection & computations
3. Target construction & ML/DL usages
4. Strategy core implementation
5. Backtesting [14] & Results analysis
6. Comparison with other strategies **Buy & Hold** [15], **Dollar Cost Average** [16]

Target

$$\textbf{target}_t = \begin{cases} 1 & \text{if } C_{t+1} > C_t \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

14. Using the Python library **backtesting.py**
15. Passive trading strategy
16. Also a passive strategy *(not implemented)*

A First Trading Algorithm

Principle & Function arguments

Principle

- Predicted signals for buy/sell decisions
- Plotly library for visualization [17]

Main arguments

- `Y_pred` : Predicted signals array
- `X_test` : Market data DataFrame
- `transaction_cost` (optional) : Transaction cost percentage
- `candlestick_chart` (optional) : Plot candlestick chart
- `candlestick_chart_daily` (optional) : Plot daily candlestick chart

17. Use of **Plotly-Resampler** : https://github.com/predict-idlab/plotly-resampler

A First Trading Algorithm

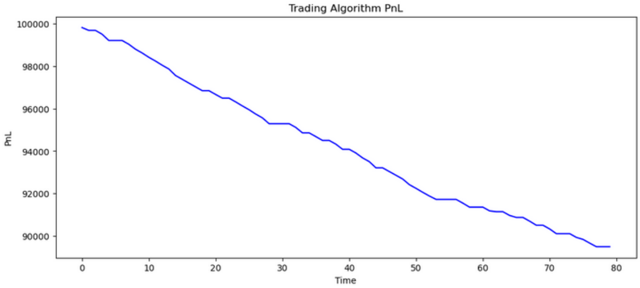
Algorithm actions

Initialization : Parameters initialized & Technical framework set up

Main loop [18]

- Buy one share if buy signal and no shares in the portfolio
- Sell one share if sell signal and shares in the portfolio
- Hold shares if buy signal and shares in the portfolio

18. Iterating over the tick-time clock t




```
Please note that plotly may experience performance issues or crashes when handling large input datasets. If you encounter any issue
s, you may need to reduce the size of your input data or use data downsampling techniques to improve performance.
```

A First Trading Algorithm

Results & Remarks

Logistic regression performance

- Accuracy score : $\approx$ 50%
- Not better than random guessing
- Trading based on predictions likely to result in losses

Algorithm limitations

- Buys/sells one share at a time
- Assumes fixed transaction cost

Improvement suggestions

- Consider more sophisticated models (e.g. neural networks)
- Incorporate additional financial and technical indicators
- Use of time-series nature to enhance our ML model

Refinement : Exploiting the timeseries for better predictions

Transaction price

The last (close) transaction price $\{C_t\}_{t\geq 1}$ is the simplest definition of the price of a financial asset.

Transformation Time-indexed dataset $\Longrightarrow$ Supervised-learning dataset

- Features to be lagged $\rightsquigarrow$ New features (columns)
- **Ex** : At t, explain y_t by $y_{t-1}, y_{t-2}, \ldots, y_{t-n}, n \geq 1$ [19]

Mathematical framework Time-series $\rightsquigarrow$ Embed the *growing* knowledge into the fitting stage as $t \uparrow +\infty$

For each datetime t, $\exists$ 2 solutions to *grow*/*evolve* the knowledge :

- Train set taking knowledge from 0 to $t-1$ (Anchored Walk-Forward) $\mathcal{F}_t$
- Train set taking knowledge from $t-H$ to $t-1$ $\sigma\left(\mathcal{F}_t \setminus \mathcal{F}_{t-H}\right)$
- In each case, test set $= K$ next observations $K = 10$ *here*

Remarks

- Time-series nature $\longrightarrow$ Existing time order $\rightsquigarrow \varnothing$ **Possible parallel exec.**
- K-Fold + GridSearch $\rightsquigarrow$ Extensive computational cost $\longrightarrow \exists$ **Possible parallelism**

19. Performed for RSI, Target & Close.

Refinement : New model, target & strategy

Model : Use of Gradient boosted decision trees [20]

Target : Same as before $\textbf{target}_t \in \{0, 1\}$

Notations : At time t, one denotes predictions for $t+1$ to $t+K = t+10 \rightsquigarrow 2$ targets

- Sum of **UP** predictions :

$$U(t, K) := \sum_{i=1}^{K} \textbf{target}_{t+i} \tag{2}$$

- Sum of **DOWN** predictions :

$$D(t, K) := K - \sum_{i=1}^{K} \textbf{target}_{t+i} \tag{3}$$

Strategy core [21] :

- $U(t, K) \geq 5 \implies$ **BUY** signal sent to exchange
- $D(t, K) \leq 7 \implies$ **SELL** signal sent to exchange

20. From the library `xgboost`
21. Calibrated empirically

Refinement : Backtesting results (1/5)

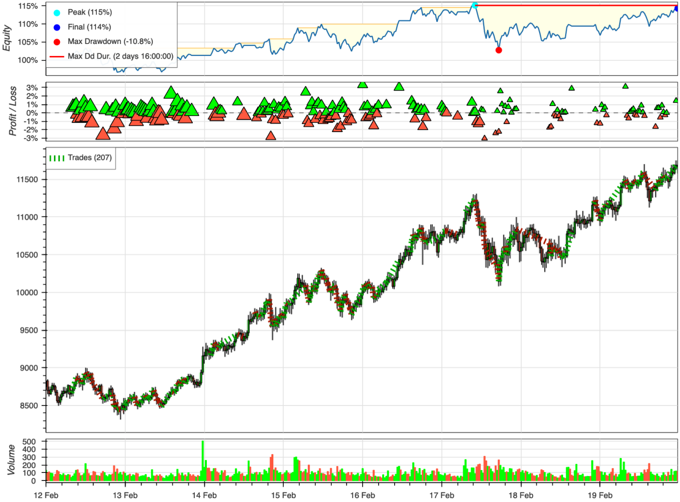


Figure – Backtesting results

Refinement : Backtesting results (2/5)

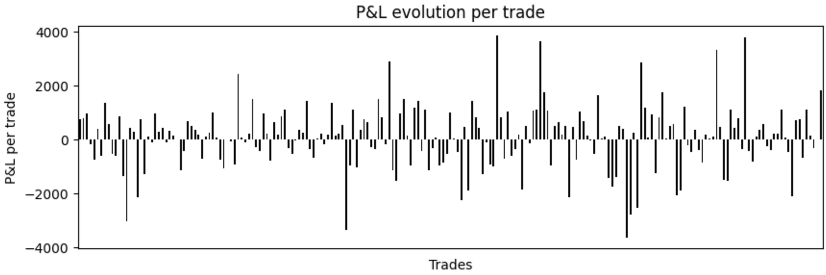


Figure – Evolution of the individual P&L for each trade

Refinement : Backtesting results (3/5)

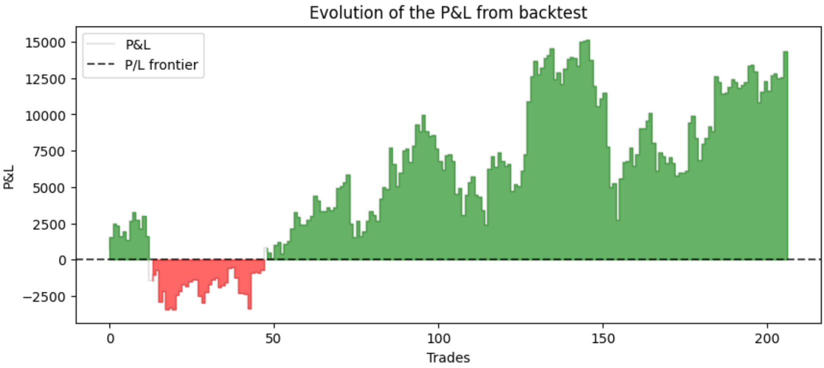


Figure – Evolution of the overall P&L

Refinement : Backtesting results (4/5)

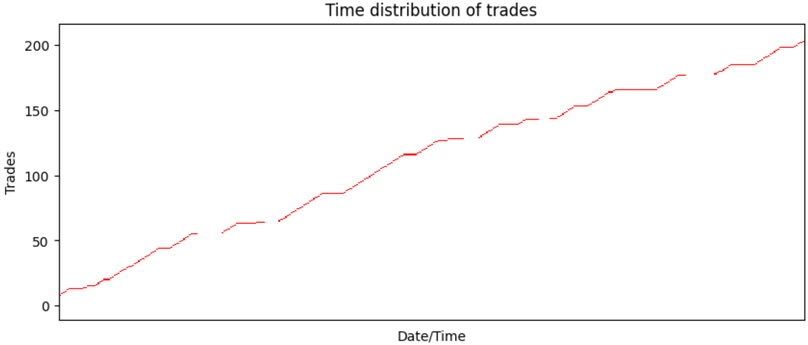


Figure – Time-schedule of the backtesting trades [22]

22. Data visualization as a Gantt chart

Refinement : Backtesting results (5/5)

Indicators	Value
Duration	8 days 07 :30 :00
Equity Final [$]	114325.535918
Equity Peak [$]	115151.981125
Return [%]	14.325536
Buy & Hold Return [%]	32.239443
Return (Ann.) [%]	7527.131325
Volatility (Ann.) [%]	9948.457655
Sharpe Ratio	0.756613
Sortino Ratio	122.700239
Calmar Ratio	697.296602
Max. Drawdown [%]	-10.794734
Avg. Drawdown [%]	-3.890339
# Trades	207
Win Rate [%]	58.937198
Best Trade [%]	3.31398
Worst Trade [%]	-3.038891

Table – Resulting indicators from the given simulation

Refinement : Use of LSTM instead of Boosting

Model : Use of a classic LSTM architecture

Performances : Same as logistic regression

$\rightsquigarrow$ Solution dropped and **Boosting** selected as ultimate predictive solution

Extensions

- Adding dropouts & other layers for regularization and better predictive power
- Skills development needed

Transfer learning : A *knowledge bridge* between **BTC** $\longrightarrow$ **ETH**

Idea : Use of ML fitting job on **BTC** to predict price movement over **ETH** dataset

Constraints : Same resampling frequency & Same date range (**start-end**)

Implementation : Use of **on-disk snapshots** of the model after each **ML fitting step** to fit on **ETH**

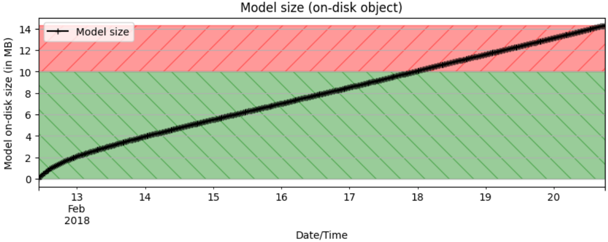


Figure – On-disk model size for each fitting iteration

$\rightsquigarrow$ At each time t, **model**$\left(\textbf{BTC}\right)_t$ applied to **ETH**

Transfer learning : Correlation between **BTC** & **ETH** (1/2)

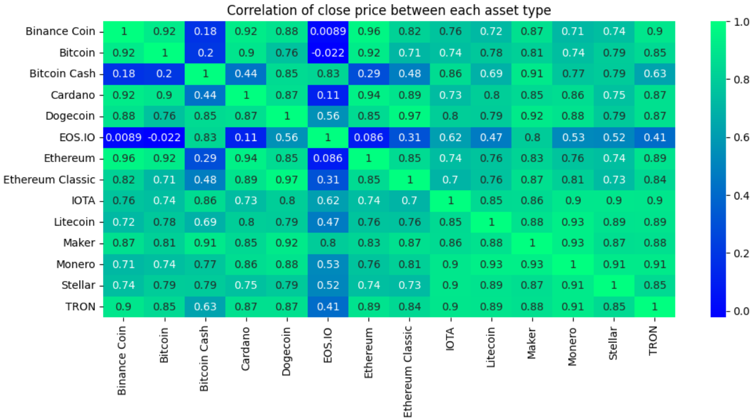


Figure – Close price $(C_t)_t$ correlation matrix

$\longrightarrow$ Correlation between **BTC** & **ETH** : $\boxed{0.92}$

Transfer learning : Correlation between BTC & ETH (2/2)

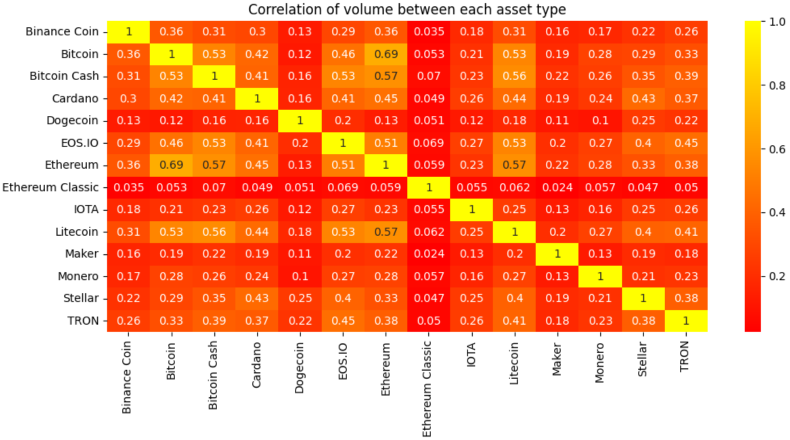


Figure – Volume $(V_t)_t$ correlation matrix

$\longrightarrow$ Correlation between BTC & ETH : $\boxed{0.69}$

Transfer learning : Classification results & TL impact

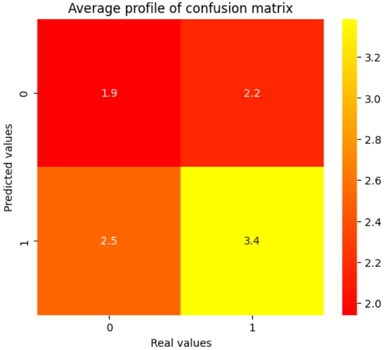


Figure – Confusion matrix - Original learning **ETH**

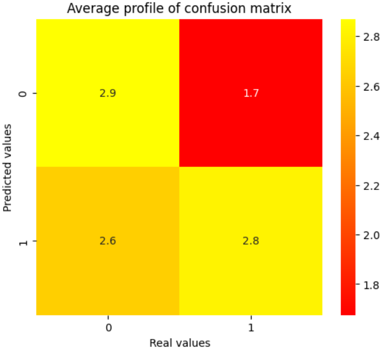


Figure – Confusion matrix - Transfer learning **ETH**

⟶ Relevant performances from *transfer learning*

Transfer learning : Backtesting results

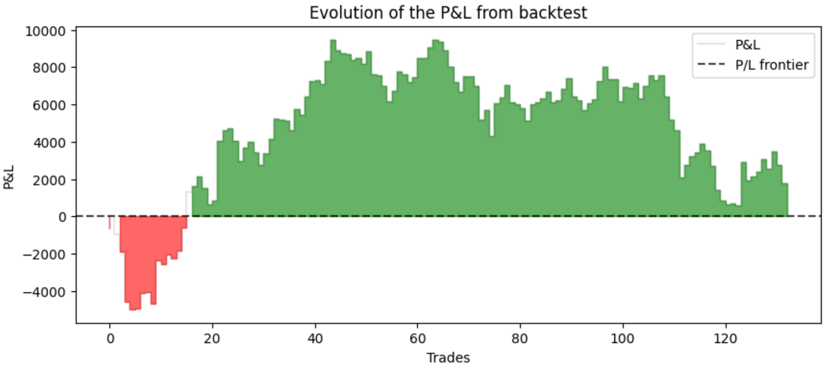


Figure – Evolution of the overall P&L

Conclusion & Perspectives

Criticism

- Skill development in Crypto & Algo-trading $\rightsquigarrow$ Difficult tasks
- Bar data $(\bullet : \mathcal{X}^n \longrightarrow \mathcal{X}'^m)$[23] reduces precision & embedded information *$m \ll n$*
- $\varnothing$ external data for multiple reasons *(price, data asymmetry)*

Synthesis

- Important pre-processing to better handle the OHLC dataset
- State-of-the-art of current **simple** trading strategies
- Implementation & Backtesting of current strategy on **BTC**
- Transfer Learning from a ML to **ETH** specific case

23. With $\bullet$ as aggregation function (avg, median, min, max, count, ...)

Conclusion & Perspectives

Extensions

- Use of L1/L2 data (trades & quotes) for greater granularity [24]
- Introduce parallelism/multithreading to speed up fitting jobs
- Use of cloud computing instances with larger CPU cores *AWS Lambda, SageMaker*
- *Final thoughts* : Extension to real-time world (Binance, FTX, ... RT WS API)
- Building a REST API [25] to automate in a *user-friendly* UI the strategy runs
- Building a CLI [26] for strategies running automation

- Enable auto. hyper-parameter tuning [27]
- Enable K-Fold for timeseries with **Incremental Learning** usage & **Time-series** nature
- Outliers detection
- PCA if huge lagged timeseries processing

24. Bar data (OHLC) always overestimates the profits generated by the strategy.
25. Flask, FastAPI
26. Click, argparse, ...
27. GridSearch, RandomSearch

Appendices

Appendix : Why XGBoost ?

Advantages

- No need for feature normalization/scaling
- Easy to measure the relative importance of each feature
- Can handle categorical and numerical features

Drawbacks

- Can take a long time to train with a large number of trees
- They're not easily interpretable
- Will not necessarily exhibit lower bias than individual decision trees

⟶ Previous benchmark against other classifiers[28] from SkLearn

28. GitHub repo : https://github.com/lcsrodriguez/ML-IceQuantity-Infrasound/

Appendix : Gantt charts

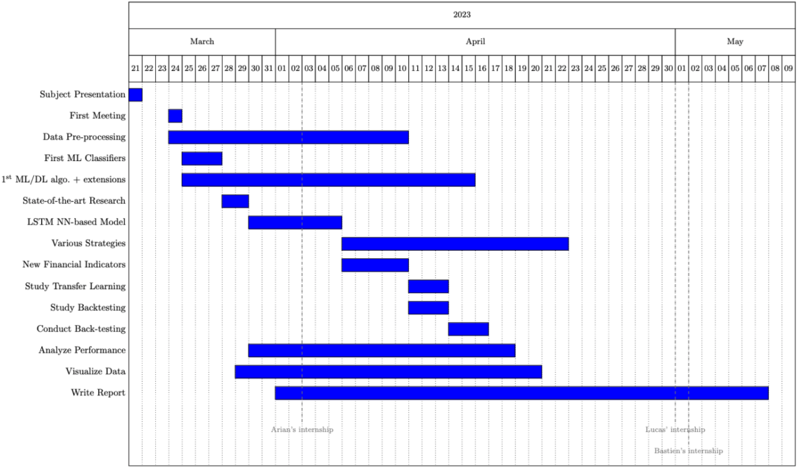


Figure – Project Gantt chart

Appendix : Bollinger bands

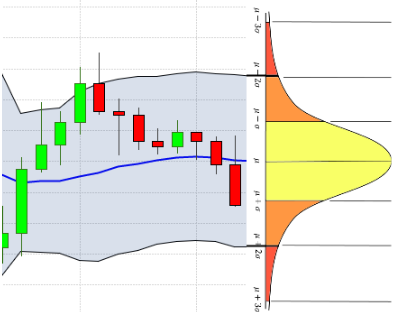


Figure – Bollinger bands schema

Appendix : BTC data visualization (1/2)

Figure – BTC Close price $(C_t)_t$ evolution

Appendix : **BTC** data visualization (2/2)

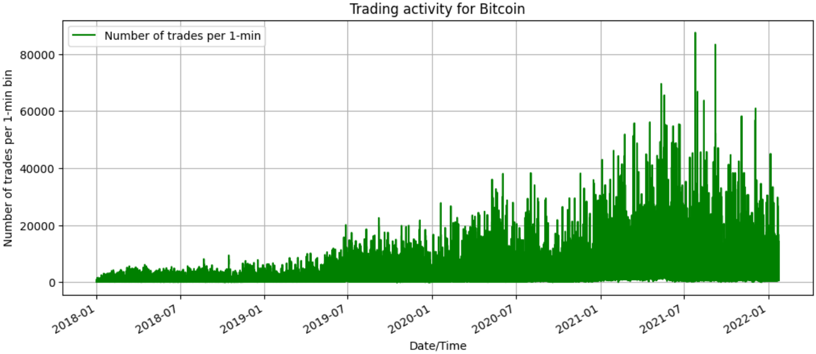


Figure – **BTC** Volume $(V_t)_t$ evolution