CS651 Bitcoin Raw Data Analysis

Chengzhang Yang
20893095
Waterloo, ON
c367yang@uwaterloo.ca

Sijie Xu
20966037
Waterloo, ON
s362xu@uwaterloo.ca

Yulin Xue
20949719
Waterloo, ON
y79xue@uwaterloo.ca

Abstract

Bitcoin has received increasing attention from the media and investors in recent years. We add to the literature on Bitcoin by studying the price movement of this digital currency through understanding the primary raw block data via cluster computing. Through a battery of robust tests, evidence reveals a limited correlation between price of bitcoin and information shown in raw block data.

1 Introduction

Bitcoin is a digital currency, which can be sent from user to user on the peer-to-peer bitcoin network. [1] Bitcoin and their peers (Ethereum, Litecoin, and Daos) marked the beginning of crypto-currency and decentralized finance, which is an entirely new financial system but without the presence of government, centralized banks, and other institutions that we are familiar with in the real world. Everything in the crypto world is created and maintained solely and automatically by machines. Therefore, the system seems "predictable" because it is operated via pre-defined rules/protocols with only limited human interference.

One of our curiosity is to predict the bitcoin's price using information gathered from its raw blocks. Since bitcoins are transferred solely on the bitcoin network, we believe the raw bitcoin blocks should contain some insight about how bitcoin's price move over time. The bitcoin network has been running since 2009, which records every transaction in and out of the crypto-wallet. A new block is added to its blockchain every 10 minutes. Each block is about 1 MB, and the blockchain has more than 700,000 blocks. Currently, the size of the entire Bitcoin blockchain is more than 370 GB, and it is growing very fast. Hence, analyzing the raw bitcoin data has become a big data problem which continues to get more and more difficult. Big data workflow with Hadoop and Apache Spark are well suited for solving problems with a large amount of data.

It has been common to apply traditional methods such as relative strength index (RSI) to quantify market conditions and identify entrance and exit points for investors. However, for an asset as volatile as Bitcoin, traditional strategy usually performs poorly when back-testing. Additionally, As shown in the figure 1 below [1], the price of Bitcoin does not have a clear seasonality pattern like other assets. Hence instead of modeling using statistic approach such as time series, we apply classification and regression machine learning models trying to capture the future movement of the price of Bitcoin (BTC-USD).

2 Problem Formulation

In order to tackle the problem, we split it into two sections: data processing and model training.

In the data processing phase, we are given a sequence of raw data $\mathbf{X} = x_1, x_2...x_n$ in blk format. Here we seek a function $\psi(x_i)$ to parse x_i into several transactions, each with k features $f_i =$

[1]Data acquired from yahoo finance https://finance.yahoo.com/quote/BTC-USD/

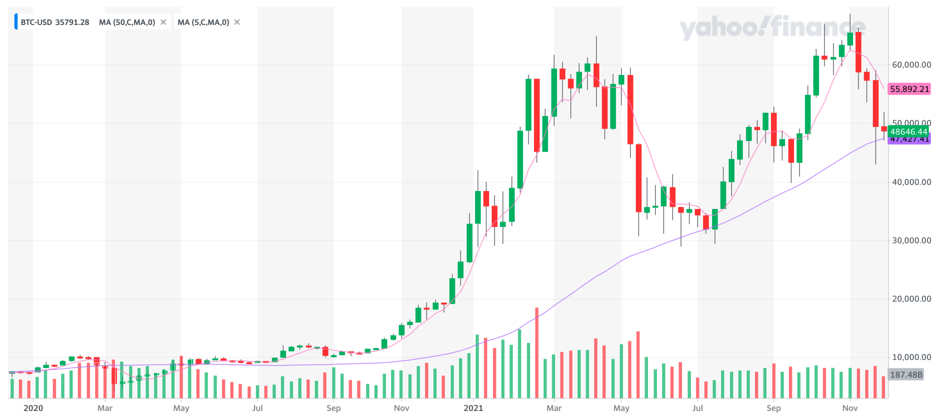


Figure 1: BTC-USD price chart

$[f_{i1}, f_{i2}....f_{1k}]^T$. Additionally, an aggregation function is applied:

$$\mathbf{\Sigma}(\mathbf{X}|\omega) = \sum_{i=1}^{n} \psi(x_i) \cdot \mathbf{I}(\mathbf{t}(x_i) \in \omega) \tag{1}$$

where $\omega \in (\omega_1 ... \omega_m)$ denotes a particular window in time (e.g. hour / day / week), where $\omega_i \cap \omega_k = \varnothing \; \forall i \neq k$. Function $\mathbf{t}(x_i)$ return the block generated time t. Hence the indicator function $\mathbf{I}(\mathbf{t}(x_i) \in \omega)$ means the raw data x_i in generated within the time period ω. Here we denote the result as $\mathbf{\Sigma}(\mathbf{X}|\omega) = y_\omega$, and $\mathbf{\Sigma}(\mathbf{X}) = \mathbf{Y}$.

In the modeling phase, we want to look for find a sequence of function $\hat{\mathbf{Y}} = \mathbf{\Phi}(\mathbf{X})$ which take the windows based features matrix $\mathbf{X}$ and generate a prediction $\hat{y}_i \in \hat{\mathbf{Y}}$ of the future next windows time ω_{i+1}, hence predicting the future price movement of BTCUSD.

3 Data Processing

3.1 Parsing Bitcoin Raw Data

As mentioned, Bitcoin data is stored in a distributed blockchain system, which structures data into a massive amount of blocks. These blocks contain information about the decentralized transactions of the Bitcoin market. In this section, we propose a workflow to download and process the raw block data into the human contents, and extract features from them.

3.1.1 Fetching Raw Data

Wget is a useful tool for downloading a single block of raw data. For example, after knowing the hash value of block 706515, we can use command[2] to download raw binary data of this specific block. First, to download multiple raw data, we construct a list of block numbers in a specific period. Then we request their corresponding hash value and form a hash value list. Using a multi-threading script and fetching a raw block each time can we get sufficient data to build the predicting model.

3.1.2 Parsing Header

Each block contains a header recording the version, timestamp, and several transactions. Please see table 1 for the detail.

Here, 1+ means there is at least one transaction in each block. The var_int is a special data type in the bitcoin protocol(See table 7) for further compaction. It can possess a different length of bytes from 1 to 9. When the value is smaller than OxFD, it only takes one byte. Since the majority of

[2]wget -O block706515.blk "https://blockchain.info/rawblock/00000000000000000000098ff7f0a7841b836064a4e46617e24f51164e626e043?format=hex"

[3]adapted from https://en.bitcoin.it/wiki/Protocol_documentation

Byte Size	Description	Data Type	Comments
4	version	int32_t	Block version information
32	prev_block	char[32]	The hash value of the previous block this particular block references
32	markle_block	char[32]	The reference to a Merkle tree collection which is a hash of all transactions related to this block
4	timestamp	uint32_t	A UNIX timestamp recording when this block was created
4	bits	uint32_t	The calculated difficulty target being used for this block
4	nonce	uint32_t	The nonce used to generate this block, allowing different versions of header and different hashes
1+	txn_count	var_int	Number of transaction entries

Table 1: Block Header[3]

values are in this circumstance, var_int can efficiently compact raw data and reduce communication costs. There is a fringe value element larger than OxFFFF FFFF, which takes a storage length of 9.

3.1.3 Parsing Transaction

We know that each block contains at least one transaction from the block header. The structure of transaction lists as Table 2:

Byte Size	Description	Data Type	Comments
4	version	uint32_t	Transaction data format version
0 or 2	flag	(optional) uint8_t[2]	If present, always 0001, and indicates the presence of witness data
1+	tx_in count	var_int	Number of Transaction inputs (never zero)
41+	tx_in	tx_in[]	A list of 1 or more transaction inputs or sources for coins
1+	tx_out count	var_int	Number of Transaction outputs
9+	tx_out	(optional) tx_out[]	A list of 1 or more transaction outputs or destinations for coins
0+	tx_witnesses	tx_witness[]	A list of witnesses, one for each input; omitted if flag is omitted above
4	lock_time	uint32_t	The block number or timestamp at which this transaction is unlocked.

Table 2: Block Transaction

The flag coexists with tx_witness when the following 4 bytes after version is 0001. It is guarantees that tx_in_count is larger than zero, and thus we don't need to worry about misinterpreting flag when parsing the tx_in_count and tx_in. The tx_in (See Table 8)and tx_out(See Table 9) are two special structures containing different information about the seller, buyer and the number of bitcoins in the transaction. Note that lock_time contains three cases. If this transaction does not lock, the value is zero. If the value is larger than 0 and less than 500000000, the value represents the block number at which the transaction was unlocked. If the number is larger than or equal to 500000000, it refers to the UNIX timestamp when the transaction was unlocked.

3.1.4 Extracting Features

After knowing how to parse the raw bitcoin data, we want to extract features from the raw data. In the block header part, we are interested in the bits, which evaluate the difficulty of building this block. The transaction part contains rich information about bitcoin trading. We decide to keep tx in(sum, max, min), tx_out(sum, max, min), value(sum, max, min) from the tx_out, and the lock_time. Since lock_ has a different meaning in different situations, we substitute lock_time with a timestamp in the block header when its value is less than 500000000. This is a good way to easily approximate transaction time when we could not get the exact transaction time from lock_ time.

3.2 Running on Clusters

Data processing workflow can be summarized in the following illustrator 2:

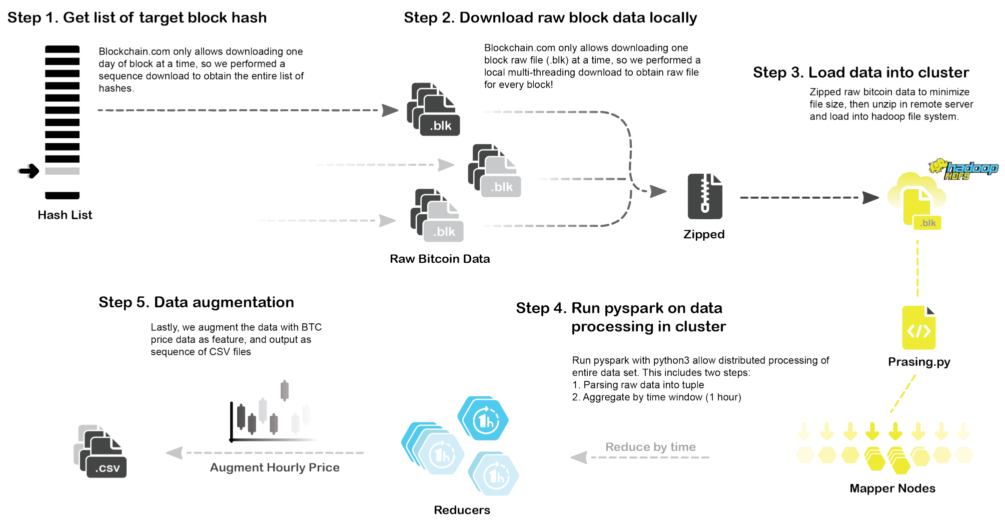

Step 1. Get list of target block hash
Blockchain.com only allows downloading one day of block at a time, so we performed a sequence download to obtain the entire list of hashes.
Hash List
Step 2. Download raw block data locally
Blockchain.com only allows downloading one block raw file (.blk) at a time, so we performed a local multi-threading download to obtain raw file for every block!
.blk
.blk
.blk
Raw Bitcoin Data
Zipped
Step 3. Load data into cluster
Zipped raw bitcoin data to minimize file size, then unzip in remote server and load into hadoop file system.
hadoop HDFS
.blk
Prasing.py
Step 4. Run pyspark on data processing in cluster
Run pyspark with python3 allow distributed processing of entire data set. This includes two steps:
1. Parsing raw data into tuple
2. Aggregate by time window (1 hour)
Mapper Nodes
Reduce by time
Reducers
Step 5. Data augmentation
Lastly, we augment the data with BTC price data as feature, and output as sequence of CSV files
Augment Hourly Price
.csv

Figure 2: Block download and processing

3.2.1 Parallel Processing in Spark

For processing data in reasonable space and time, we choose to use $|\omega|$ = 1 hour, and $\forall \omega_i \in (2017 - 07 - 021 - AM, 2020 - 11 - 2007 - AM)$.

To begin with, each raw data will be loaded into the spark workers. Then we use our parser to convert each raw data into several tokens. Each token includes [timestamp, bits, value, in_count, out_count, 1]. Then we exploit floor division to map keys in the same time window and create extra dimensions for reducing. Here, we get [timestamp // 3600, bits, value, in_count, out_count, 1, value, value, in_count, in_count, out_count, out_count]. Now, we use reducyByKey to aggregate data with the same key and generate [timestamp, sum bits, sum bits, sum value, sum in_count, sum out_count, count, min value, max value, min in_count, max in_count, min out_count, max out_count]. After aggregating, the number of data is dramatically smaller than the tokens generated from the raw data. Lastly, we recover the timestamp so that the time can match other data sources and store it into a CSV file.

3.2.2 Price Augmentation

Bitcoin is traded on multiple global cryptocurrency exchanges, such as Binance and Coinbase, which uses a bid and ask system like the one used in equity trading. Despite that different markets place may have a slightly different price at the same moment, it should be negligible for ω >10 minutes. This is because the differences across bitcoin exchanges are potentially exploitable [2] (When there is an arbitrage opportunity, you can execute ”complimentary” transactions on both exchanges). Because it is so profitable and easy to perform, the bitcoin price will tend to converge because of these extra liquidities provided.

The price information we obtain is from an exchange named Bittrex, which has publicized the hourly bitcoin pricing information since 2017. We process the data into the format described in table 10:

4 Price Prediction

The data is split into a training set and testing set with time 2020-08-01; data points prior to the time period are labeled training, and the others are testing. No data shuffling is involved in the preparation. The complete features are listed in the table 3:

Features	Description	Source
Time	UNIX Timestampe in epoch	Block Header 1
nBits	Aggregation of Bits	Block Header
Open	Price at the beginning of period	Price Augmentation 10
Close	Price at the end of period	Price Augmentation
Moving_Direction	The direction of price movement (Close - Open)	Price Augmentation
Volume_BTC	Total volume of BTC traded on exchange	Price Augmentation
Value	Total volume of BTC transferred in Satoshi	Block Transaction 2
Value_min/max	Min / Max value by transactions	Block Transaction
Count	Total number of transactions	Block Transaction
In_count	Aggregation of tx_in_count	Block Transaction
In_min/max	Min / Max tx_in_count by transactions	Block Transaction
Out_count	Aggregation of tx_out_count	Block Transaction
Out_min/max	Min / Max tx_out_count by transactions	Block Transaction

Table 3: List of features

In order to make proper predictions, we designed two different setups for predicting the future price movement of BTCUSD.

- Future Price Prediction
  Here we use the "Close" of current frame ω_t as target $\hat{y}_i$.
- Future Price Moving Direction Prediction
  Here we use the "Moving_Direction" of in the current time frame ω_t as target $\hat{y}_i$.

These features $\mathbf{x}$ = [Open, Time, Volume_BTC, nBits, Value, Value_Max, count, in_count, in_max, out_count, out_max] are selected to train models from both setups, in_min, out_min and value_min are omitted since they only carry limited information as most of their values are exactly zero. Since the main bitcoin network has been slow at processing transfers since 2017, the "Time" feature may not accurately match with the time period when transactions are first initiated. Hence We also aggregate and test the data in two different scopes, namely (1) $|\omega|$ = 1 hour, (2) $|\omega|$ = 1 day, in order to minimize the error mentioned above.

4.1 Models

4.1.1 Linear / Logistic Regression (LR)

We choose to use linear regression model from the python Sklearn package, aiming to minimize the least squared error $S = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$ where

$$F(\mathbf{x}) = \hat{\mathbf{y}} = \beta_0 + \sum_k \beta_k \mathbf{x}_k \tag{2}$$

Linear regression model is only used in predicting numeric target, namely the future price prediction. Additionally, we adapt logistic regression for classification in future price moving direction prediction. The logistic regression model assume a linear relationship between the $\mathbf{X}$ and log-odds of the event that $\mathbf{Y} = 1$.

4.1.2 Decision Tree (DT)

Secondly, we apply a non-parametric supervised learning method decision tree regressor model from the python Sklearn package described in [3]. A tree can be seen as a piecewise constant approximation. In general, decision trees are greedy algorithms that recursively partitions the feature space into small components, it asks several questions and makes prediction based on the answers. The goal of the algorithm is to find the tree node with the max IG(D,s) at each step, where IG(D,s) is the information gain. [4]

$$Impurity(D) = \frac{1}{N}\sum_{N}(y_i - \mu)^2 \tag{3}$$

$$IG(D,s) = Impurity(D) - \frac{N_{left}}{N}Impurity(D_{left}) - \frac{N_{right}}{N}Impurity(D_{right}) \tag{4}$$

4.1.3 Gradient Boosting Tree (GBT)

Eventually, we test the gradient-boosting Tree regressor from the python pyspark package, namely the GBTRegressor module. Gradient boosting tree gives a prediction model in the form of an ensemble of a weak prediction model. Here we use the decision trees. It is similar to the random forest model but yields much more significant results.[5] A gradient-boosted trees model is built in a stage-wise fashion as in other boosting methods. It starts by fitting an initial model to the data. Then a second model is built that focuses on accurately predicting the cases where the first model performs poorly.

Additionally, we optimize the model with the Grid search technique and K-fold [4]. Grid search is a tuning technique that loops through predefined hyperparameters and finds the best-fitting model on the training set. K-fold cross-validator split the dataset into k consecutive folds without shuffling. Each fold is used once as validation while the k - 1 remaining folds form the training set.

5 Main Results

The regression models are evaluated using "RegressionEvaluator" from the python pyspark package, which calculates the root mean square deviation (RMSE), mean absolute error (MAE), and coefficient of determination R^2 of the model. And we find the importance of each features in each model using the "FeatureImportances" attribute in the generated model. The classification models are evaluated with the area under the receiver operating characteristic curve (AUC), which shows the trade-off between sensitivity (or TPR) and specificity (1 – FPR). The result of the models are presented in table 4 and table 5.

Model	Regression			Classification
Evaluation	RMSE	MAE	R^2	Area under ROC
Linear Regression	67.3	42.4	0.998	NA
Logistic Regression	NA	NA	NA	0.5112
Decision Tree	804.16	506.213	0.812	0.5106
Gradient Boosting Tree	480.30	294.978	0.933	0.50996

Table 4: Hourly aggregated data trained model

Model	Regression			Classification
Evaluation	RMSE	MAE	R^2	Area under ROC
Linear Regression	336.47	221.96	0.9711	NA
Logistic Regression	NA	NA	NA	0.4340
Decision Tree	929.233	562.95	0.780	0.509
Gradient Boosting Tree	973.84	647.816	0.758	0.488

Table 5: Daily aggregated data trained model

From the table above, we can see all classification models result in AUC $\approx$ 0.5, with the best performing linear classifiers resulting in AUC = 0.5112, which means the classification model performs merely better than a uniformly random classifier. Among the regression models, all models achieve $R^2 > 0.75$, with the linear regression model outperforming the rest by achieving $R^2 \approx 1.0$ in both

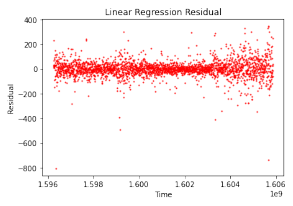
[4]We choose to use k = 3 due to the size of the samples

data set. We notice that all regression models perform better on the hourly aggregated set than the daily aggregated one. This contradicted our previous assumption that daily data should perform better [5]. Hence we further examine the dominant predictor and its importance score in all models. The result is listed in the table 6 below:

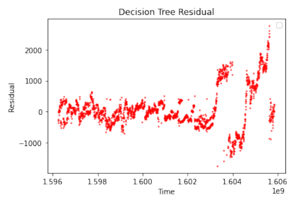
Method of aggregating	Hourly		Daily	
Evaluation	Dominant Predictor	Score	Dominant Predictor	Score
Linear Regression	Open	0.9997	Open	0.9803
Decision Tree	Open	0.996	Open	0.967
Gradient Boosting Tree	Open	0.985	Open	0.980

Table 6: Daily aggregated data trained model

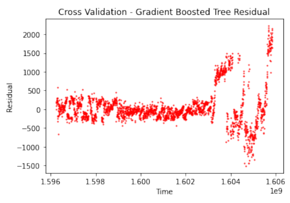
It is clear that the feature ”Open” is the dominant predictor in all models, with a score ≈ 1 in all cases. Thus, all models found that it is most suitable to predict close price with open prices, meaning that the raw bitcoin data converge limited information regarding the price movement. Additionally, this explains why our model performs worst when switching to daily data since the difference between the close and open price is much larger compared to hourly data. We further look into the residuals of our model, which are plotted in the figure 3 below.



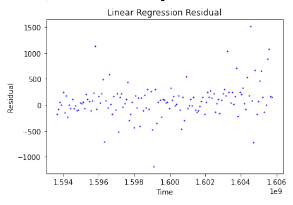
(a) LR hourly residual



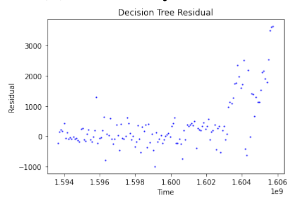
(b) DT hourly residual



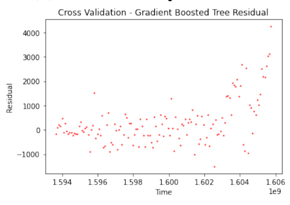
(c) GBT hourly residual



(d) LR daily residual



(e) DT daily residual



(f) GBT daily residual

Figure 3: Model residuals

The residuals from linear regression are approximately randomly distributed, while the residuals for both tree models show an increasing variance through time. Hence, the linear regression model is better at capturing the bitcoin’s price movement.

6 Conclusion

Throughout our analysis, we discover that the classification model performs poorly in predicting price movement direction. Among the regression model, we conclude that the linear model can capture most of the information considering that (1) the linear model has the best R^2 value among other models, (2) the residuals plots produced by the linear model is centered around zero with constant variance. However, all of our models failed to utilize information from raw bitcoin data for predicting price movement due to the fact that ”Open” price remains a dominant predictor in all of our models with importance $> 98\%$. Hence, we can conclude that our raw bitcoin data has limited power in predicting future bitcoin prices. We summarized some of the reason behind it below:

- Most bitcoin are traded in a centralized crypto wallet app such as ”wealth simple,” ”Paypal,” or ”coin base,” which does not require a trader to set up a digital wallet and

[5]Some bitcoin transactions can not be verified within the hour due to high volume on the bitcoin network, by switching to a lower precision, we may be able to minimize these cases.

physically interact with the bitcoin network. Even though the company behind these applications still need to back up their trades with real digital asset, the time gap between the price change in the bitcoin marketplace and actual transaction on the network may be significantly large.

- The recent favored ”layer 2” payment protocol such as ”lightning network” (LN) built on top of the bitcoin blockchain may also introduce more complexity on the raw bitcoin data. Since LN records transactions via the payment channel, it does not broadcast the entire transaction to the blockchain before closing the channel.
- Not all bitcoins are traded in USD, but the bitcoin network records all global transactions. There may be inconsistency in the different exchanges that causes price movement to be least correlated to raw transaction data.
- Not all bitcoin transactions are trades; it also records gifts to other, money laundering, black market trades, coin staking, exchanges on liquidity pool, or yield farmings. Hence, the correlation between the bitcoin price in USD and the raw transaction itself may not be significant.

References

[1] Financial Crimes Enforcement Network. “Statement of Jennifer Shasky Calvery, Director Financial Crimes Enforcement Network United States Department of the Treasury” (2013) (cit. on p. 1).

[2] Joshua Kolden. “Is there an efficient way to exploit arbitrages between the different exchanges”. *Stack Exchange* (2011) (cit. on p. 4).

[3] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. *Journal of Machine Learning Research*, vol. 12 (2011), pp. 2825–2830 (cit. on p. 5).

[4] Apache Spark. “Decision Trees - RDD-based API - Spark 3.2.0 Documentation” (2021) (cit. on p. 6).

[5] Glen S. “Decision Tree vs Random Forest vs Gradient Boosting Machines: Explained Simply”. *Data Science Central* (2019) (cit. on p. 6).

Appendices

Value	Byte Size	Format
$\leq$ OxFD	1	uint8_t
$\leq$ OxFFFF	3	OxFD followed by the length as uint16_t
$\leq$ OxFFFF FFFF	5	OxFE followed by the length as uint16_t
others	9	Oxff followed by the length as uint64_t

Table 7: Var_int

Byte Size	Description	Data type	Comments
36	previous_output	outpoint	The previous output transaction reference, as an OutPoint structure
1+	script length	var_int	The length of the signature script
?	signature script	uchar[]	Computational Script for confirming transaction authorization
4	sequence	uint32_t	Transaction version as defined by the sender. Intended for "replacement" of transactions when information is updated before inclusion into a block.

Table 8: Transaction In

Byte Size	Description	Data type	Comments
8	value	int64_t	Transaction Value
1+	pk_script length	var_int	Length of the pk_script
?	pk_script	uchar[]	Usually contains the public key as a Bitcoin script setting up conditions to claim this output

Table 9: Transaction Out

Column Name	Description	Data type	example
UNIX Timestampe	Time in epoch	Int	1605855600
Date	Human readable date and time	String	2020-11-20 06-AM
Open	Price at the beginning of period	Double	18168.59
High	Highest Price within the period	Double	18390
Low	Lowest Price within the period	Double	18100
Close	Price at the end of period	Double	18194.27
Volume BTC	Total Volumne of BTC traded	Double	38.38
Volume USD	Total Volumne of USD traded	Double	383800

Table 10: Bittrex Hourly BTCUSD