School of Management, Economics, Law, Social Sciences and International Affairs

The effect of Twitter activity on Bitcoin price

Documentation

Software Engineering for Economists
(7,610,1.00)

Alen Stepic - 11-475-258
Dimitrios Koumnakes - 10-613-370
Severin Kranz - 13-606-355
Joël Sonderegger - 11-495-488
Chi Xu - 16-300-915

Fall Term 2017

Supervisor:
Prof. Dr. Philipp Zahn
FGN HSG
Varnbüelstrasse 19
9000 St. Gallen
Department of Economics

January 5, 2018

Contents

List of Figures

List of Abbreviations

API Application Programming Interface
ATOKEN Access Token
ASECRET Access Token Secret
BPI Bitcoin Price Index
BTC Bitcoin
CKEY Consumer Key
CSECRET Consumer Secret
JSON Java Script Object Notation
RPI Raspberry Pi
SSH Secure Shell
USD US Dollar

1 Introduction

In the academic environment both accountability and reproducibility are important to ensure a high quality of research. However, the publishing process of scientific papers and journals seems to be outdated, as reproducibility is not evaluated by leading journals. (McCullough & Vinod, 2003, p. 888) Driven by digitalization, new ways of data collection and processing arise, by using high computational capacity and advanced econometric methods. On the one hand, the usage of scripts, automation or machine learning can increase effectiveness and efficiency. Hence, much larger data sets can be proceeded. On the other hand, this creates also new challenges to trace and reproduce data sets, procedures or analyses. There are often cited academic papers, where the initial computation is neither clearly documented, nor reproducible (McCullough & Vinod, 2003, pp. 874–887). Replicating data or existing results does not provide any new knowledge at all. Nevertheless, the ability to understand what researchers have done is key. Non-reproducible work cannot be characterised as science or used as a basis for policy-making (McCullough & Vinod, 2003, p. 888). Therefore, clear documentation of processes, used scripts and tools should be the basis of every scientific publication.

1.1 Goal

The objective of this documentation is to provide a detailed description of the research process, used hardware, software, procedures and tools of the accompanying paper. This description enables the reader to reproduce the results discussed in our work. Thus, it contains an explanation of how input data have been gathered, stored, aggregated and analyzed.

1.2 Methodology

After pointing out the relevance of this documentation, the second chapter is an explanation of the required set-up in order to reproduce our results by running our Python scripts available from Github. The programming language Python has been chosen, as it comes with a high variety of modules and is especially suitable for data science. The third chapter discusses the input data of our project. This includes the process of gathering and storing tweets from Twitter, as well as the collecting Bitcoin price data for the same time frame. Both Twitter and Bitcoin data have been gathered by using an Application Programming Interface (API), which allows to access real-life data by a set of explicitly defined and documented methods. Further on, the fourth chapter explains, how the data has been aggregated and wrangled to enable the subsequent analysis. Finally, the statistical analysis has been conducted with STATA and is discussed in the last chapter of this documentation.

1.3 Scope

The scope of the documentation is to provide an overview of the different steps that have been conducted to obtain the results of the paper. Therefore, this documentation includes all information necessary to understand and reproduce our findings from the accompanying paper. However, it does not contain any further information about the content and results of our research and is only meant as supportive material. Furthermore, it does not provide a detailed description of the code itself, in order to avoid redundancy of information with the existing code documentation within the scripts.

2 Set-up

This section goes through the set-up that is necessary in order to execute the code discussed in this documentation.

2.1 Github repository

All code is available on the following Github repository:
`https://github.com/joelsonderegger/twitterbitcoin/`

2.2 Requirements to run Python scripts

Before you go any further, make sure that at least Python version 3.0.0 is available from your command line. You can check this by running in your command line:

```
python --version
```

You should get some output like Python 3.7.13. If you do not have at least Python version 3.0.0, please install the latest 3.x version from `python.org`.

On top, you need various Python packages. We created a requirements file `requirements.txt`, which contains all required packages to run the Python scripts of this project. You can find the `requirements.txt` in the root directory of our Github repository. The list of packages can be installed using pip install like so:

```
pip install -r requirements.txt
```

Now you should be ready to run the Python scripts.

2.3 STATA

For the statistical analysis of our data, we used the statistic software STATA (version 14.2). When seeking for statistical analysis, regression analysis or data management, STATA is a capable tool. To reproduce our results, you need to have the STATA software installed on your local computer. After starting the program, you can follow the guidelines in chapter 5 (Data Analysis).

3 Data Collection

The following section contains a detailed description of how the data for the sequential analysis is gathered and stored. Two separate data sets will be collected: (1) Tweets data and (2) Bitcoin price index data.

3.1 Tweets Data

With the Python script `collectTwitterData.py` real-time Twitter data are streamed and stored using the Python library Tweepy. A Raspberry Pi can be used to gather Twitter data over a longer period.

3.1.1 Python Script

Twitter offers different APIs for the collection of Twitter data. However, the time frame to gather data from the past with full access is limited to seven days (Twitter Inc., 2017b). To gather a data set over a longer period, without using payed services Twitter, data have to be streamed and saved in real-time. Python offers different Twitter libraries such as the package Tweepy.

Tweepy
Tweepy is a open-source libary. It provides a simple way to gather real time tweets, by managing authentication, connection and tweet information of Twitters streaming API. The Tweepy instance tweepy.streaming is used for the streaming session and forwards the data to the StreamListener instance. The `class listener(StreamListener)` calls the method `def on_data(self, data)`. With this method all data of filtered tweets are gathered in real-time. (Roesslein, 2009) The main function `def main()` manages the required authentication as follows:

```
auth = OAuthHandler (ckey, csecret)
auth.set_access_token(atoken, asecret)
```

Those authentications are demanded by Twitter and explained in more details in the next paragraph. Furthermore, the main function starts to run the stream, while filtering for the buzzword ”bitcoin” by processing `twitterStream.filter(track=["bitcoin"]`. Tweepy has been applied for this project because it simplifies the script and enables its users to gather a huge number of tweets without any costs.

Installing Tweepy
Tweepy is installed easily by running following commands in the command prompt. However,

this step should not be necessary if you have previously completed the above described setup using the requirements file `requirements.txt`.

```
pip install tweepy
```

If the previously downloaded Python installation package does not contain the tweepy library, the tweepy package has to be downloaded manually. The package can be downloaded for free from the following link:

```
https://pypi.python.org/pypi/tweepy
```

Twitter Authentication

To access the Twitter data, Twitter requests an identification of the user. The identification is assured by different keys and access tokens. Those are the (1) consumer key (CKEY), (2) consumer secret (CSECRET), (3) access token (ATOKEN) and (4) access token secret (ASECRET). (Twitter Inc., 2017c)

To obtain the mentioned keys and tokens a Twitter account is needed. Once, a Twitter account exists, an application has to be created by logging in the developer area with the credentials under the following link:

```
https://apps.twitter.com/
```

After the creation of an application, the keys and tokens can be extracted. Figure 1 illustrates how to retrieve the keys and tokens.

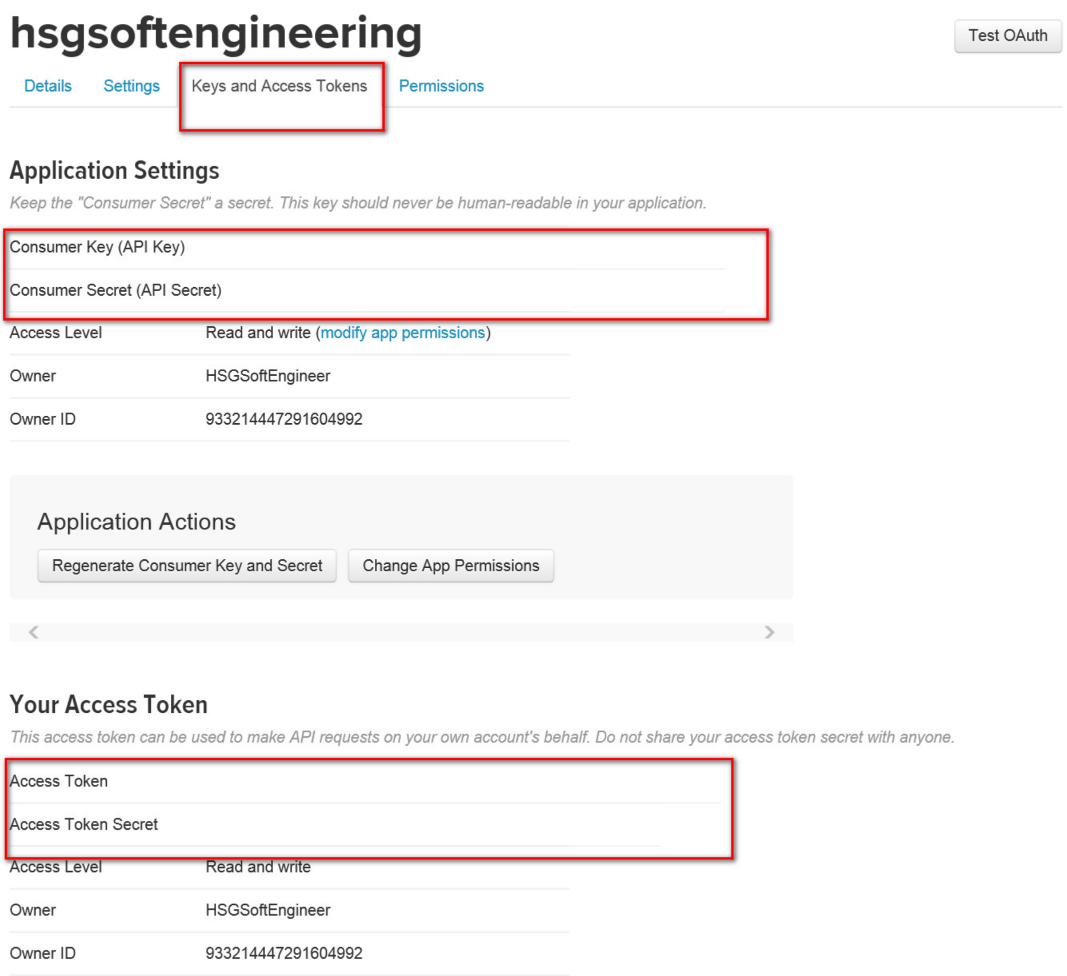

Figure 1: Twitter Keys and Access Tokens, based on Twitter Inc. (2017a).

Running the Python Script

By running the Python script `collectTwitterData.py` real-time Twitter data is gathered on an ongoing basis until you stop the script forcefully.

```
$ python collectTwitterData.py
```

The Twitter API provides the tweets data in a Java Script Object Notation (JSON) format. Tweets are only received in case a tweet contains the predefined key word `bitcoin`. While a single tweet consists of many attributes we are only interested in two attributes:

- created_at: Timestamp of the created tweet (in UTC time)
- text: Text of the tweet

An example for a response from the Tweepy API looks like this (of course this shows only the two attributes we are interested in):

```
{
```

```
    ...
    "created_at": Tue Dec 19 20:40:49 +0000 2017,
    "text": "Singapore issues bitcoin warning after price rise - #BTC #Bitcoin
       #Crypto"
    ...
}
```

The `text` attribute needs to be encoded from the raw JSON format into the UTF-8 format for further analysis. After successfully encoding the `text` attribute a tweet is appended to the `twitterData.csv` file as a new row and saved in the /data folder.

To ensure a high quality data set, without missing any data points for the observed period, we have built in an exception handler into the `collectTwitterData.py` script. If an error occurs an email will be sent with the error message. Additionally, any exception that occurs during the execution of the script will be added to the log file `/data/tweet_collection_error_log.csv`.

3.1.2 Hardware set-up

As explained in the previous section the Twitter API was used to gather tweets in real-time based on the buzzword "bitcoin" by using Tweepy. The Python script was running for seven days without interruption on a Raspberry Pi 2 (RPI) which was accessed with a Secure Shell (SSH) connection. This approach allowed us to ensure a continuous data stream while being able to access the program output regardless of time and location. Furthermore, due to a low energy consumption of the RPI, the method is also the most cost efficient option compared to a personal computer or NAS Server. The following section describes the different steps that have been taken in order to set up the RPI 2, SSH, Git and Python3.

1. Setup of the Raspberry Pi
   (a) Burn image 2017-11-29-raspbian-stretch.zip to micro SD card available from

   ```
   https://www.raspberrypi.org/downloads/raspbian/
   ```

   (b) Boot image from micro SD card
   (c) Open SSH session using PuTTY SSH client for Windows available from

   ```
   https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html
   ```

   (d) Login to server with credentials
2. Update system to the newest software
   (a) Update list of applications with the command

```
sudo apt-get update
```

(b) Update applications with the command

```
sudo apt-get upgrade
```

3. Setup Git

(a) Install Git using the command

```
 sudo apt-get install git
```

(b) Clone Git repository from

```
https://github.com/joelsonderegger/twitterbitcoin.git
```

4. Setup Python3

(a) Install Python3 using the command:

```
sudo apt-get install Python3
```

(b) Install the tweepy module for Python using the command

```
pip install tweepy
```

5. Collect Twitter Data

(a) Open a new screen using the command

```
screen -S 'name'
```

(b) Start Python program using the command

```
python3 collectTwitterData.py
```

(c) Detach the screen to keep the program running after secure shell connection has been terminated using the commands

```
Ctrl+A
Ctrl+D
```

(d) Re-attach the screen to check whether the script is still running using the command

```
screen -r 'name'
```

6. Download the collected data

(a) Install FileZilla on your local machine available from

```
https://filezilla-project.org/
```

(b) Connect to RPI and navigate to data location

(c) Download data manually

In summary, this approach worked well and ran smoothly. A clean set-up is recommended to ensure a continuous data stream over a longer period. The final data set used for our analysis is available through the link provided in the data folder on Github.

3.2 Bitcoin Price Data

We wrote a Python script which collects Bitcoin price data as there was no pre-existing data set that satisfied our needs. The Bitcoin price is best expressed by the Bitcoin Price Index (Kristoufek, 2015). The Bitcoin price index (BPI) is an index of the exchange rate between the Bitcoin (BTC) and the US dollar (USD). The objective of the script was to gather hourly Bitcoin Price Index data for at least the time period in which we gather the tweets data. We found an API by bitcoinaverage.com which was sufficed our needs. A detailed description of the API follows in section 3.2.3.

3.2.1 Execution

By executing the Python script `collectCryptocurrencyData.py` hourly data for the Bitcoin Price Index is retrieved.

```
$ python collectCryptocurrencyData.py
```

3.2.2 Output

After successfully running the Python script `CollectCryptocurrencyData.py` the file `bpi.csv` is generated in the folder `/data`. It is important to note that every execution of the script overwrites any existing `bpi.csv` file.

The file `bpi.csv` contains historical Bitcoin Price Index data for one month on an hourly basis. Each data point consists of the following parameters:

- time: Timestamp on an hourly basis in UTC time

- average: Average price (in USD)
- high: Highest price (in USD)
- low: Lowest price (in USD)
- open: Opening price (in USD)

3.2.3 API: bitcoinaverage.com

Bitcoinaverage.com offers a free API that provides real-time and historical price data for a range of crypto-currencies including Bitcoin. The following requests delivers data for an per hour monthly sliding window.

Request
The request to get the data for an per hour monthly sliding window looks as follows. This request requires an authentication that therefore the registration and generation of an API key. The registration and generation of an API key is freely available on bitcoinaverage.com. The `collectCryptocurrencyData.py` already contains the necessary keys. This means that you do not need to register or generate keys to execute the script `collectCryptocurrencyData.py`.

```
https://apiv2.bitcoinaverage.com/indices/global/history/BTCUSD?period=monthly&?format=json
```

Response An excerpt of an example response looks like the following:


```
[
   ...
   {
      "high": 8271.04,
      "average": 8247.83,
      "open": 8242.39,
      "low": 8217.72,
      "time": "2017-11-22 15:00:00"
   },
   {
      "high": 8246.82,
      "average": 8203.19,
      "open": 8203.81,
      "low": 8157.25,
      "time": "2017-11-22 14:00:00"
   },
   {
      "high": 8267.27,
```

```
        "average": 8238.62,
        "open": 8248.77,
        "low": 8198.54,
        "time": "2017-11-22 13:00:00"
    }
    ...
]
```

4 Data Aggregation and Data Wrangling

After the tweets and BPI data set is generated it needs to be aggregated before the data can be analyzed. In addition, some data wrangling is necessary to bring the data in a format which than can be analysed. Some attibutes are not necessary for the subsequent analysis, thus, can be deleted in order to enhance performance and reduce data volume, while other attributes need to be generated. The script `aggregateTwitterBpi.py` does all of these steps.

4.1 Process

The process of the script `aggregateTwitterBpi.py` works as follows:

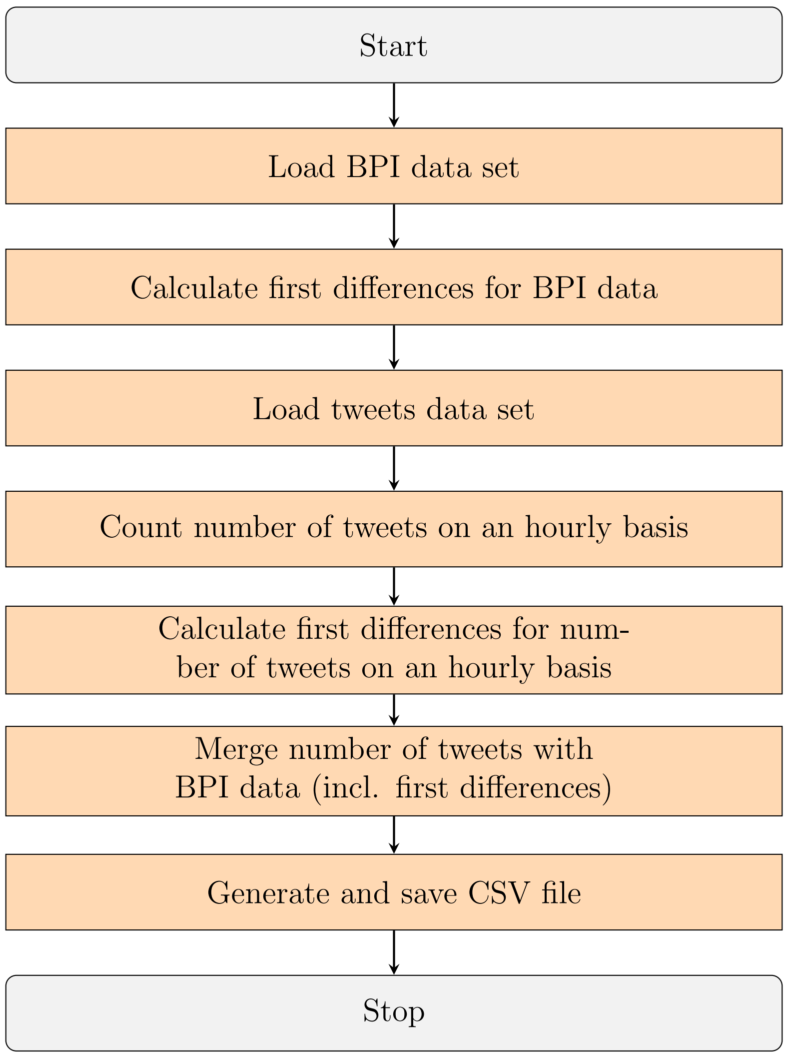

Figure 2: Data aggregation and data wrangling process

Here is a short description of what happens at the various steps:

Load BPI data set

The script takes the BPI data set that is located at `data/bpi.csv`. Then, BPI closing prices are added to every data point.

Calculate first differences for BPI data

The first differences of the BPI closing prices and the natural logarithm of the first differences of the BPI closing prices are calculated. The reason for calculating these values is provided in chapter 5 (Data Analysis).

Load tweets data set

The script takes the tweets data set that is located at `data/twitterData.csv`.

Count number of tweets on an hourly basis

First, groups the tweets by hour. Second, counts the number of tweets for every hour.

Calculate first differences for number of tweets on an hourly basis

The first differences of the number of tweets and the natural logarithm of the first differences of the number of tweets are calculated. Again, the reason for calculating these values is given in chapter 5 (Data Analysis).

Merge number of tweets with BPI data (incl. first differences)

Up to this point, there are the two separate data frames df_tweets_per_hour and df_bpi. Here is an excerpt of an the two data frames look at this point:

df_tweets_per_hour

Y	M	D	H	nr_tweets	df_nr_of_tweets	log_df_nr_of_tweets
2017	12	17	20	5851	NaN	NaN
			21	18097	12246	1.129133592
			22	2163	-15934	-2.124250032
			23	5411	3248	0.916937772

df_bpi

Y	M	D	H	time	close	df_bpi_close	log_df_bpi_close
2017	12	17	20	2017-12-19 20:00:00	17653.05	NaN	NaN
			21	2017-12-19 21:00:00	17041.93	-611.12	-0.035231795
			22	2017-12-19 22:00:00	17287.08	245.15	0.014282624
			23	2017-12-19 23:00:00	17614.62	327.54	0.018769837

By using the `pandas.DataFrame.join` function the columns of the `df_tweets_per_hour` are joined with `df_bpi` on the `MultiIndex(Y, M, D, H)`. The code snipped for the join

```
...
df_merged = df_tweets_per_hour.join(df_bpi, how='left')
...
```

An example of the joined data frame looks the following:

df_merged

Y	M	D	H	time	nr_of_tweets	df_nr_of_tweets	log_df_nr_of_tweets	bpi_closing_price	df_bpi_closing_price	log_df_bpi_closing_price
2017	12	17	20	2017-12-19 20:00:00	5851	NaN	NaN	17653.05	NaN	NaN
			21	2017-12-19 21:00:00	18097	12246	1.129133592	17041.93	-611.12	-0.035231795
			22	2017-12-19 22:00:00	2163	-15934	-2.124250032	17287.08	245.15	0.014282624
			23	2017-12-19 23:00:00	5411	3248	0.916937772	17614.62	327.54	0.018769837

4.2 Execution

By executing the Python script `aggregateTwitterBpi.py` an the aggregated data set that contains number of tweets and Bitcoin Price Index, both on an hourly basis, gets generated.

```
$ python aggregateTwitterBpi.py
```

4.3 Output

After successfully running the Python script `aggregateTwitterBpi.py`, the file `nr_of_tweets_bpi_closing_price.csv` is generated in the folder `/data`. It is important to note that every execution of the script overwrites any existing `nr_of_tweets_bpi_closing_price.csv` file.

The file `aggregateTwitterBpi.csv` contains historical tweets and Bitcoin Price Index data. Each data point consists of the following parameters:

- time: Timestamp on an hourly basis in UTC time
- nr_of_tweets: Sum of tweets
- df_nr_of_tweets: First difference of the sum of tweets
- log_df_nr_of_tweets: Natural logarithm of the first difference of the sum of tweets
- bpi_closing_price: BPI closing price
- df_bpi_closing_price: First difference of the BPI closing price
- log_df_bpi_closing_price: Natural logarithm of the first difference of the BPI closing price

5 Data Analysis

The data analysis was performed with the statistical software STATA (version 14.2). The corresponding .dta file and .do file are stored in the folder `/documentation/stata_analysis/` in the Github repository. Following, we describe all the commands used to produce our econometric model.

Step 1:
After starting STATA the first step was to import the data file (.csv file). We did so by following command (be careful, your location of the .csv file will probably differ):

```
import delimited C:\Users\Dimi\Desktop\nr_of_tweets_bpi_closing_price.csv
```

Step 2:
STATA is not recognizing our variable ”time” as a date variable. Therefore, we had to create a new variable ”time2” and give it the same values as ”time” in the same format as ”time”. At the end, we define ”time2” to be a time series with an hourly progress.

```
gen double time2 = clock(time, "YMD hms")\\
format time2 %tcNN-DD-CCYY_HH:MM:SS\\
order time2, after(time)\\
tsset time2, format(%tcNN-DD-CCYY_HH:MM:SS) delta(1 hours)
```

Step 3:
Next we plotted all variables by the following commands and exported the produced graphs (and saved in local repository):

```
twoway (tsline nr_of_tweets)
twoway (tsline df_nr_of_tweets)
twoway (tsline log_df_nr_of_tweets)
twoway (tsline bpi_closing_price)
twoway (tsline df_bpi_closing_price)
twoway (tsline log_df_bpi_closing_price)
```

Step 4:
To check for stationarity, we run the Augmented Dickey-Fuller (ADF) test for all variables with a time delay of 0 periods (lag 0).

```
dfuller nr_of_tweets, lags(0)
dfuller df_nr_of_tweets, lags(0)
```

```
dfuller log_df_nr_of_tweets, lags(0)
dfuller bpi_closing_price, lags(0)
dfuller df_bpi_closing_price, lags(0)
dfuller log_df_bpi_closing_price, lags(0)
```

Step 5:
Calling the lag-order selection criteria statistics for our VAR model. The maximum amount of lags is 7 and the variables that we chose for the later VAR model are df_nr_of_tweets and df_bpi_closing_price.

```
varsoc df_nr_of_tweets df_bpi_closing_price, maxlag(7)
```

Step 6:
Running the VAR regression for our two variables. With recommended lag of 1 (from Step 5).

```
var df_nr_of_tweets df_bpi_closing_price, lags(1/1)
```

Step 7:
Calling the granger causality test with lag length 1.

```
Vargranger
```

Step 8:
The VAR regression results were exported to a latex format by installing and using the outreg2 package. The generated piece of latex code was then inserted into in the latex file of the paper. All other results were exported with an image software and used as .png files in latex

```
ssc install outreg2
outreg2 using var_gegression_results.tex, replace
```

6 References

Kristoufek, L. (2015). What are the main drivers of the bitcoin price? evidence from wavelet coherence analysis. *PloS one*, *10*(4), e0123923.

McCullough, B. D. & Vinod, H. D. (2003). Verifying the solution from a nonlinear solver a case study. *The American Economic Review*, *93*(3), 873–892.

Roesslein, J. (2009). Streaming with tweepy. Retrieved from http://tweepy.readthedocs.io/en/v3.5.0/streaming_how_to.html

Twitter Inc. (2017a). Application management. Retrieved from https://apps.twitter.com/

Twitter Inc. (2017b). Search tweets. Retrieved from https://developer.twitter.com/en/docs/tweets/search/overview

Twitter Inc. (2017c). Twitter terms of service. Retrieved from https://twitter.com/en/tos

7 Declaration of Authorship

We hereby declare,

- that we have written this thesis without any help from others and without the use of documents and aids other than those stated above;
- that we have mentioned all the sources used and that we have cited them correctly according to established academic citation rules;
- that we have acquired any immaterial rights to materials we may have used such as images or graphs, or that we have produced such materials ourself;
- that the topic or parts of it are not already the object of any work or examination of another course unless this has been explicitly agreed on with the faculty member in advance and is referred to in the thesis;
- that we are aware that our work can be electronically checked for plagiarism and that we hereby grant the University of St.Gallen copyright in accordance with the Examination Regulations in so far as this is required for administrative action;
- that we are aware that the University will prosecute any infringement of this declaration of authorship and, in particular, the employment of a ghostwriter, and that any such infringement may result in disciplinary and criminal consequences which may result in our expulsion from the University or us being stripped of our degree.

Alen Stepic - 11-475-258

Dimitrios Koumnakes - 10-613-370

Severin Kranz - 13-606-355

Joël Sonderegger - 11-495-488

Chi Xu - 16-300-915

By submitting this academic term paper, we confirm through my conclusive action that we are submitting the Declaration of Authorship, that we have read and understood it, and that it is true.