Zürcher Hochschule für Angewandte Wissenschaften

School of Management and Law

Master of Science in Banking and Finance

Advanced Quantitative Methods – Spring Semester 2022

Can stable and robust regression models for crypto- and fiat currencies be identified, based on commodities and stock indices?

Submitted by Group I:

Aziz Yazen Ahmetkerib

Oeggerli Matthias

Schmid Rafael

Supervisor:
Dr. Tomasz Orpiszewski

Filed on:
Winterthur, May 30, 2022

I. Management Summary

Research has shown that the effects of digitalization are affecting the financial industry. Due to a constant and dynamic technological development, computers can capture and process data exceptionally well. This is essential nowadays and allows a deeper understanding of the financial sector. Because of the omnipresence, this paper aims to verify if there is a relationship between the chosen crypto- and fiat currencies with commodities and stock indices. To fulfill this goal, this paper investigates the research question:

> *"Can stable and robust regression models for crypto- and fiat currencies be identified, based on commodities and stock indices?"*

To answer the research question, 13 independent variables (commodities & stock indices) and ten dependent variables (crypto- and fiat currencies) were defined and imported to Python via the application programming interface (API) from Refinitiv Workspace. After cleansing and analyzing the data, the program SQLite was used to export and save the results. The relationship between the dependent variables (crypto- & fiat currencies) and the independent variables (commodities & stock indices) was analyzed by multiple ordinary least square (OLS) regressions. To assess the robustness and stability of the various regressions, statistical tests such as the Augmented Dickey-Fuller test, the Chow test, and the rolling window regression were used. The results have revealed substantial differences between cryptocurrencies and fiat currencies. Our models were able to predict the price changes of fiat currencies much more accurately than the price changes of cryptocurrencies. However, the Chow test showed a structural break for fiat currencies while the models for cryptocurrencies mostly were stable over time. On the other hand, the results of the rolling window regressions showed that the betas of fiat currencies are more stable over time than those of cryptocurrencies. In summary, the research question can partially be answered positively since it was possible to identify stable robust models over time. However, due to the significant differences in modeling the prices of crypto- and fiat currencies, it is recommended to differentiate the two groups in potential future analyses.

II. Table of Content

III. List of Figures

IV. List of Tables

1. Introduction

The MSc. Banking and Finance students of the Zurich University of Applied Sciences (ZHAW) have the opportunity to write a paper on a stochastic topic during the module *Advanced Quantitative Methods*. This paper aims to identify stable and robust regression models over time. First, the initial situation and the problem are explained. Afterwards, the objective, the research question, and the scope are pointed out. Finally, the structure of the paper will be explained.

1.1 Initial Situation and Problem definition

Digitalization is an omnipresent topic for enterprises in a wide range of sectors. A study by Deloitte Digital GmbH (2015) analyzed the time course (fuse) and the impact strength (bang) for various industries through a disruption map. The financial industry was classified as a "short fuse, big bang" which means that the effects of digitalization will be felt in a short-term. This conclusion is driven by the emergence of new digital technologies (Deloitte Digital GmbH, 2015, p. 1-5). Due to a constant and dynamic technological development, computers can capture and process data exceptionally well (World Economic Forum, 2016, p. 3-7). In comparison with other industries, applying statistical methods in banking is relatively new (Hand, 2011, p. 1). Despite this fact, statistics have a crucial role in finance. Statistical methods are used to valuate financial assets because markets are dynamic and complex. In addition, statistical methods can explain and show the interdependencies in the market. Therefore, their use is well established which leads to a growing demand for statistical know-how in all financial institutions (Gimeno & Mateos de Cabo, 2006, p. 1). The head of IT at Goldman Sachs, Joanne Hannaford, believes that the world markets have become more complex and that humans alone can no longer guarantee an overview. For this reason, it is necessary to have new technologies to evaluate data in a target-oriented manner (fintropolis, 2020).

1.2 Research objective and research question

This assignment aims to verify whether price changes of chosen crypto- and fiat currencies can be explained by analyzing the price changes of different commodities and stock indices. To reach this objective, the following research question was defined:

> "Can stable and robust regression models for crypto- and fiat currencies be identified, based on commodities and stock indices?"

1.3 Scope

Due to the complexity of the topic it is not possible to investigate all aspects. The following points outline the scope of this paper:

- The used know-how corresponds to the content of the modules quantitative methods & advanced quantitative methods from the Master in Banking and Finance at the University of applied science Zurich (ZHAW).
- The analysis is based exclusively on OLS regression.
- The data collection is done with the software "Refinitiv Workspace."
- As a database, the SQLite software is used.
- The development of the code is achieved with Python.
- Five cryptocurrencies and five fiat currencies were defined as response variables, which are analyzed by 13 explanatory variables (six commodities & seven stock indices.)
- For the assets (crypto- and fiat currencies), a stationarity- and Chow test is performed.
- To verify if the betas are stable over time, the analysis includes a rolling window regression.

1.4 Structure of the Assignment

This paper has six chapters. The next chapter presents the literature review, which builds the theoretical foundation of the further analysis. Chapter three is about the data set, focusing on the collection, pre-processing and transfer to a database in SQLite. The implementation of the theory is applied in chapter four. In the fifth chapter, the results of the analysis will be summarized. Finally, a conclusion, a critical self-reflection and an outlook on the research topic are shown in the last chapter.

2 Literature Review

2.1 Linear Regression Model

A linear regression line can be applied to forecast the value of the dependent variable or to analyze the interdependencies between the dependent (y) and independent (X) variables. The first mentioned component is the output, estimated by adding independent variables (Martin, 2021, p. 21-27). A regression model with one independent variable (X) is called a simple linear regression model. However, there are also regression models where the effects are attributed to multiple independent variables. This form is called a multiple linear regression. The deterministic part of the formula reflects the systematic impact of the X variable on y. If a coefficient (β) changes by one unit, the average effect on y is the corresponding marginal value of the linear coefficient (β). The random part of the formula reflects the remaining impact (Bachmann, 2021, p. 2-4). The illustration below shows the mathematical equation.

$$y = \underbrace{\beta_0 + \beta_1 x_1 + \cdots + \beta_K x_K}_{\text{deterministic part}} + \underbrace{\varepsilon}_{\text{random part}}$$

Figure 1:Mathematical equation for a linear regression model (Bachmann, 2021, p. 3)

Explanation:

- The dependent variable is y
- The independent variable is X
- The coefficient of the variable is β
- The error term is ε

2.1.1 Ordinary Least Square Regression (OLS)

To estimate the unknown parameters respectively, the coefficients, the method of OLS is applied (UZH, n. d., p. 1), where in this framework the sum of squared errors (SSE) is minimized (Bachmann, 2021, p. 19-20). The following illustrations show the determination of the beta coefficients for a simple as well as for a multiple linear regression model.

$$b_0 = \bar{y} - b_1\bar{x}$$

$$b_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} = \frac{\sum_{i=1}^{n} x_i y_i - n\bar{y}\bar{x}}{\sum_{i=1}^{n} x_i^2 - n\bar{x}^2} = \frac{S_{xy}}{S_x^2} = r_{xy}\frac{S_y}{S_x}$$

$$b_0 = \bar{y} - b_1\bar{x}_1 - b_2\bar{x}_2$$

$$b_1 = \frac{s_y(r_{x_1y} - r_{x_1x_2} \cdot r_{x_2y})}{s_{x_1}(1 - r_{x_1x_2}^2)}$$

$$b_2 = \frac{s_y(r_{x_2y} - r_{x_1x_2} \cdot r_{x_1y})}{s_{x_2}(1 - r_{x_1x_2}^2)}$$

Figure 2: Determination of the beta coefficients for a simple (left) & multiple (right) linear regression (Bachmann, 2021, p. 22-24)

2.1.2 Assumption for the ordinary least square (OLS) estimation

To ensure that the OLS regression is valid, the following assumptions must be checked (Bachmann, 2021, p. 31):

1. **Linearity in parameters:** $\boldsymbol{Y_t = \beta_0 + B_1 X_{1,t} + \cdots + B_k x_{k,t} + \varepsilon_t}$

There must be a linear relationship between the predictor and the outcome (Martin, 2021, p. 52-53).

2. **Mean independence:** $\boldsymbol{E\left(\varepsilon_t \middle| X_{1,t} \ldots, X_{K,t}\right) = 0}$

The errors must be independent of each other, otherwise the confidence intervals or the coefficients will lead to biased hypothesis tests (Martin, 2021, p. 52-53).

3. **No perfect multicollinearity:** $\boldsymbol{X_{1,t}, \ldots X_{K,t}}$ **are linear independent**

Suppose the correlation coefficient between the independent variables is ± 1, the model cannot be used. However, it is common to have high multicollinearity (high correlation) between the independent variables. This can be explained by the fact that the independent variables capture identical points (e.g., crime rate and poverty rate) or an expected time trend between them (e.g., annual GDP and annual real estate prices). A high degree of multicollinearity causes significant standard errors in the estimated coefficients, which leads to falsified hypothesis tests. Removing a highly correlated variable can be used to deal with this problem. However, this must be considered carefully because it may lead to a variable bias. Alternatively, the sample can be expanded. (Bachman, 2021, p. 10-15).

4. **Homoscedasticity:** $\boldsymbol{Var(\varepsilon_t) = \sigma^2}$

If the variance of the error terms are the same at every point on the regression line, there exists homoscedasticity. If there are different variances, there is heteroscedasticity. In the last-mentioned case, the hypothesis tests or the confidence intervals and coefficients are biased and

possibly lead false results (Martin, 2021, p 52.). Potential remedies are the use of white-corrected standard errors or a log model. The Breusch-Pagan test allows to check whether heteroscedasticity or homoscedasticity is present (Bachmann, 2021, p. 21-30). Figure 3 serves as a visual orientation (Carlson, Newbold & Thorne, 2013, p. 578).

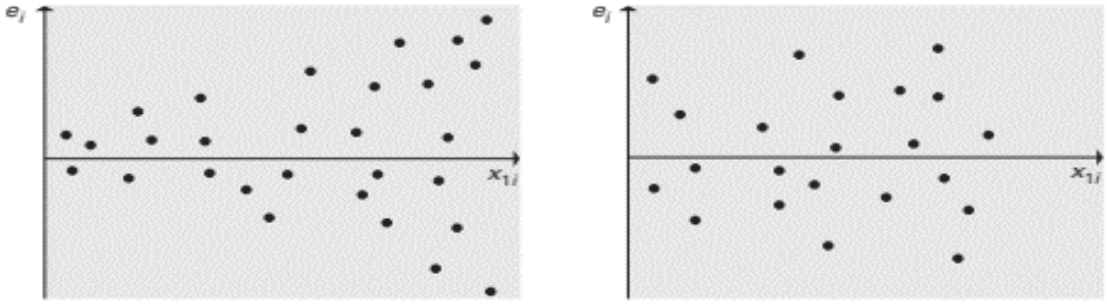


Figure 3: Heteroscedasticity (left) homoscedasticity (right) (Carlson, Newbold & Thorne, 2013, p. 588)

5. **No autocorrelation: $Cov\left(\varepsilon_K, \varepsilon_J\right) = Corr\left(\varepsilon_K, \varepsilon_J\right) = 0\,, K \neq J$**

A common problem during the analysis of time series regression models is the autocorrelation, which can be defined as follows: "*When the observations have a natural sequential order, and as a result, the correlation structure is related to that order, this correlation is called autocorrelation"* (Martin, 2021, p. 81). This issue causes the OLS standard errors to be biased, and therefore the confidence intervals and hypothesis tests are not valid. By using the Durbin-Watson test, it is possible to check if there is autocorrelation (Bachmann, 2021, p. 27-31)

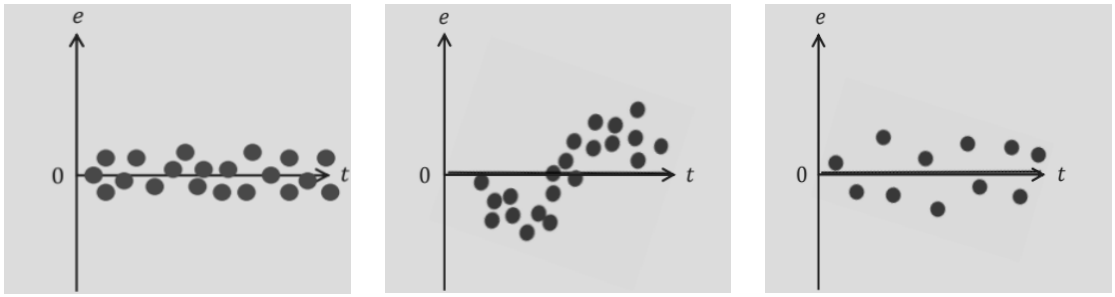


Figure 4: No autocorrelation, positive autocorrelation & negative autocorrelation (Bachmann, 2021, p. 28-29)

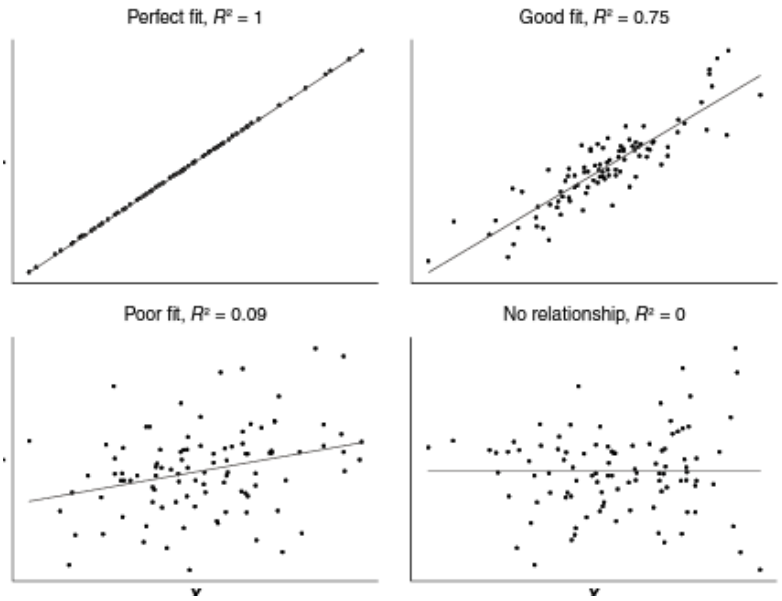
6. **Normally distributed error term: $\varepsilon \sim N\,(0, \sigma_\varepsilon^2)$**

The errors must have a normal distribution. If this is not the case, the corresponding confidence intervals and coefficients are biased, leading to false results of hypothesis tests (Martin, 2021, p. 52).

2.1.3 Coefficient of Determination: R^2

The coefficient of determination (R^2) is a crucial indicator of regression analysis. Its value can range between 0 and 1. It provides information about the quality of the model and about how well the independent variables can predict the dependent variable. Expressed in other words,

this indicator shows the proportion of variance of the dependent variable (y) that can be attributed to the independent variables (X). The higher this value is, the better the prediction can be made. (Martin, 2021, p. 31-33). The figures below illustrate various examples and the formula.



$$R^2 = \frac{SSR}{SST} = 1 - \frac{SSE}{SST}$$

Figure 5: Examples of coefficient of determination (R^2) (Martin, 2021, p. 31-33)

Explanation:

- Sum of squared errors (SSE)
- Regression sum of squares (SSR)
- Total sum of squares (SST)

2.1.4 Adjusted R^2

The adjusted R^2 is used for the correction of the small decrease in the sum of squared errors due to irrelevant independent variables. According to this, the adjusted R^2 can be better than the conventional R^2 to fit multiple regression models with different numbers of X variables (Carlson, Newbold & Thorne, 2013, p . 492).

$$R^2_{adj.} = \frac{\frac{SSE}{n-K-1}}{\frac{SST}{(n-1)}}$$

Explanation:

- The number of observations in the data set is n
- The different number of x variables between the models is k

2.2 Stationarity and Non-stationary

Often time series data show trends and cycles. When this is the case, it is an non-stationary time series data, which is not suitable for a regression model because it shows incorrect variable relationships. Therefore, time series must be stationary to allow their use in a regression model. However, non-stationary time series can be transformed, which results in a constant mean, variance, and covariance. For instance, this can be done with the first Y-difference from t-1 to t. Figure 6 illustrates stationarity and nonstationary (Bachmann, 2021, p. 2-7). If the following formal conditions are satisfied, stationarity is present:

- No trends are observed respectively the mean value remains constant
- The variance is constant
- The covariance stays constant

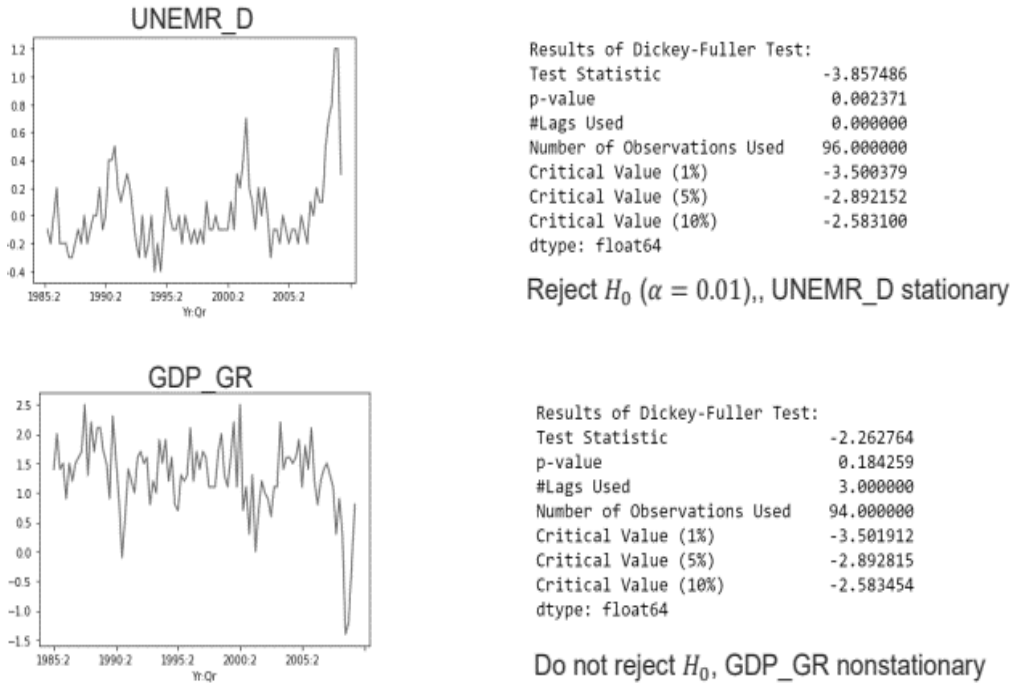


Figure 6: Stationary (above) & nonstationary (below) (Bachmann, 2021 p. 26)

2.2.1 Augmented Dickey-Fuller Test

By using the Augmented Dickey-Fuller test it can be checked whether stationarity is present (Bachmann, 2021, p. 22-23). It is performed by applying the formula $Y_t = const + \emptyset Y_{t-1} + \emptyset_1 \Delta Y_{t-1}$. If no significance (p-value < α or test statistic > critical value) is detected regarding the parameter, there is no stationarity respectively H_0 cannot be rejected (Orpiszewski, 2021, p. 21-26).

H_0: Time series is nonstationary

H_1: Time series is stationary

Explanation:

- The first difference of the time series is Y_t
- Const is the abbreviation for constant
- The parameter $\emptyset Y_{t-1}$ is multiplied by the previous observation

2.3 Rolling Window Regression (RWR)

"RWR analysis evaluates the degree of covariation between two records and how it changes through time" (Oehlert & Swart, 2019, p. 1). To run such an analysis, a constant subsample size must be defined at the beginning, corresponding to the windows width. For example, a data set has a period of ten years. In this regard, we specify that the constant subsample size is one year at a time. Next, an ordinary least squares regression is created for the first year's subsample. Now an iterative process takes place in which the subsample (12 months) is shifted by one month at a time. Thus, the timeline for the second oversample is months 2-13. This procedure is carried out until the monthly shift is no longer possible. The output of this iteration are multiple OLS regressions (+1 month each) for 12 months. Through this procedure, the beta coefficients of the separate OLS regressions can be observed. This provides insights into whether or not a β-coefficient has an ongoing relationship with the dependent variable (target variable) (Peng, 2010, p. 3806-3807).

2.4 Chow test

The Chow test checks for a divided data set (sub-dataset) if the regression coefficients are not identical respectively, it checks if there are break or structure points. If the coefficients are similar, then one regression line is sufficient. However, two regression lines must be used if they are not. One possible example that clarifies the issue is the stock market price before and after black Friday. The test consists of the following null and alternative hypotheses (Orpiszewski, 2022, p. 17-20):

H_0: The time series has no structural break

H_1: The time series has a structural break

To perform the hypothesis test, the error sum of squares of the data must be quantified for the whole data set and the half data sets. In a further step, the values are inserted into the following formula, which is based on the F-Statistic (Orpiszewski, 2022, p. 17-20):

$$\frac{(RSS_p - (RSS_1 + RSS_2))/k}{(RSS_1 + RSS_2)/(N_1 + N_2 - 2k)}$$

Explanation:

- RSS_p is the Error sum of squares data of the pooled regression line
- RSS_1 is the Error sum of squares data of the regression line before the break
- RSS_2 is the Error sum of squares data of the regression line after the break
- k are the degrees of freedom
- N_1 is the number of observations of the regression line before the break
- N_2 is the number of observations of the regression line after the break

Based on the obtained result (F-statistic), the P-value of the F-statistic can be compared with a significance level (Orpiszewski, 2022, p. 17-20).

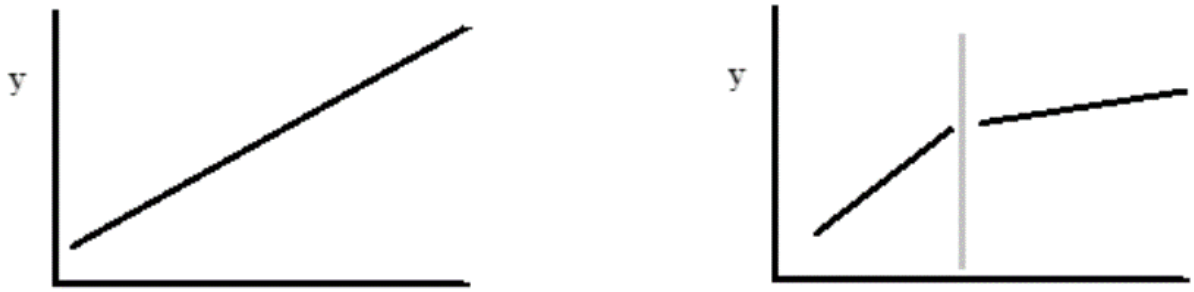


Figure 7: Dataset with a single regression line (left) and a set with a breakpoint and two regression lines (right)(statisticshowto, 2016)

.

3 Data

The handling of the data was divided into three tasks; collection of the raw data, data cleansing and analysis. As source of data, Refinitiv Workspace, an application provided by the media group Thomson Reuters which contains comprehensive financial and macroeconomic data, was used. The relational database SQLite was used to manage the data in an efficient manner. The data processing was done with the Python programming language. The appendix (chapter 8.4) provides the detailed process and the used Python code.

3.1 Raw Data

In a first step, all required variables for the building and evaluation of the robust regression models were chosen. Table 1 shows the categories, the Refinitiv instrument codes (RIC) and a brief description of the variables. The categories "Cryptocurrencies" and "Fiat Currencies" will be used as dependent variables and the "Stock indices" and "Commodities" as independent variables.

Table 1: Overview of used variables (own illustration)

Categories	RIC	Description
Cryptocurrencies	BTC=	Bitcoin
Cryptocurrencies	ETH=	Ethereum
Cryptocurrencies	XRP=	Ripple
Cryptocurrencies	LTC=	Litecoin
Cryptocurrencies	BCH=	Bitcoin Cash
Fiat currencies	EUR=	Euro
Fiat currencies	GBP=	Pound Sterling
Fiat currencies	JPY=	Japanese Yen
Fiat currencies	CHF=	Swiss Francs
Fiat currencies	CAD=	Canadian Dollar
Commodities	LCOc1	Crude Oil ICE
Commodities	WTC-	Crude Oil WTI
Commodities	XAU=	Gold
Commodities	XAG=	Silver
Commodities	.BCOM	Bloomberg Commodity Index
Commodities	.dMIWO0EN00PUS	MSCI World Energy Index USD USD

Stock indices	.NDX	Nasdaq 100 Index
Stock indices	.SPX	S&P 500 Index
Stock indices	.FTSE	FTSE 100 Index
Stock indices	.CSI300	China Securities Index 300
Stock indices	.dMIWO00000PUS	MSCI World Price Index USD
Stock indices	.dMIEF00000PUS	MSCI Emerging Markets Price Index USD
Stock indices	.dMIEU00000PUS	MSCI Europe Price Index USD

Initially, all relevant Python libraries were imported to the notebook and a connection to Refinitiv Workspace was established by using the individual app key. With that, time series of the different variables could be retrieved via API into Python. Since Refinitiv Workspace does have a limit of datapoints which can be retrieved in a single request, a loop was applied to download the data of every variable separately. Figure 8 shows an example of such a loop where the closing prices on a daily basis for the last ten years were downloaded. After downloading the time series of all four categories separately, the data was merged into one DataFrame. This ensures an optimal transfer of the raw data into the SQLite database.

```
#Loop to download time series for cryptos
crypto_histo = pd.DataFrame()

for i in range (0,len(rics_crypto)):
    data_input = ek.get_timeseries(rics_crypto[i],
                                   start_date=startdate,
                                   end_date=enddate,
                                   fields='CLOSE',
                                   interval='daily',
                                   corax = 'adjusted')
    data_input['RIC'] = rics_crypto[i]
    crypto_histo = crypto_histo.append(data_input)
```

Figure 8: Loop example (own illustration)

3.2 Data Cleansing

Once the data has been exported into the SQLite database, the eikon API connection is no longer needed since the data can be retrieved from the database. When pulling the data from the database to Python, there are a few things which must be modified (cf. figure 9) to facilitate the data cleansing.

```python
#pull the data from the SQLite database to python
conn = sqlite3.connect('AQM_Project_Aziz_Oeggerli_Schmid.db')
c = conn.cursor()
c.execute("SELECT * FROM raw_data")
new_data_raw = c.fetchall()

#convert the list to a dataframe using the before defined columns as header
new_data_raw = pd.DataFrame(new_data_raw,columns=['Date','CLOSE','RIC'])

#pivot the tables after RIC and set the date as index --> with the '[rics]', the order of the columns stays the same
new_data_raw = new_data_raw.pivot(index='Date',columns='RIC',values='CLOSE')[rics]

#change the names of the columns of the dataframe
new_data_raw.columns = columns

#switch format of the index to datetime
new_data_raw.index = pd.to_datetime(new_data_raw.index)
```

Figure 9: Retrieve of the Raw data from our SQL-database (own illustration)

After the retrieval, we do some pre-analysis in order to figure out the earliest date on which daily closing prices are available for all asset categories. The assumption, that cryptocurrencies have the shortest data availability, was verified. Ripple does have the least data availability of all 23 variables, starting from 04.03.2019, which is why the period of the analysis was changed to 04.03.2019-31.03.2022.

```
new_data_raw
```

Date	Bitcoin	Ethereum	Ripple	Litecoin	Bitcoin Cash	EUR	GBP	JPY	CHF	CAD	Crude Oil ICE	Crude Oil WTI	Gold	Silver	Bloomberg Commodity Index
2019-03-04	3702.48	125.31	0.30114	45.44	122.91	1.1337	1.3186	111.74	0.9988	1.3302	65.67	56.60	1286.4150	15.0742	80.8184
2019-03-05	3937.60	138.78	0.31261	53.16	130.20	1.1306	1.3175	111.89	1.0040	1.3345	65.86	56.55	1287.1801	15.1222	81.1937
2019-03-06	3945.10	136.35	0.31471	56.39	130.07	1.1305	1.3169	111.75	1.0048	1.3440	65.99	56.22	1286.3600	15.0711	80.7408

Figure 10:Overview DataFrame without NaN (own illustration)

In order to check for further missing data (represented as NaN in Python), the code *new_data_raw.isnull().sum()* can be applied. The output shows that cryptocurrencies do not have any missing values, but the other asset categories do. Since they contain a comparable amount of NaN's it can be concluded that the missing data results from public holidays and weekends, on which only cryptocurrencies have price changes but not the other asset categories. By plotting the data, we conduct an additional visual check (see figure 11). Since the plots do not show any anomalies, the final step of the data cleansing is to fill the remaining NaN with the values of the previous day. This approach can be used for data which follows a trend, but does not show seasonality and is known as linear interpolation in the forwarding direction.

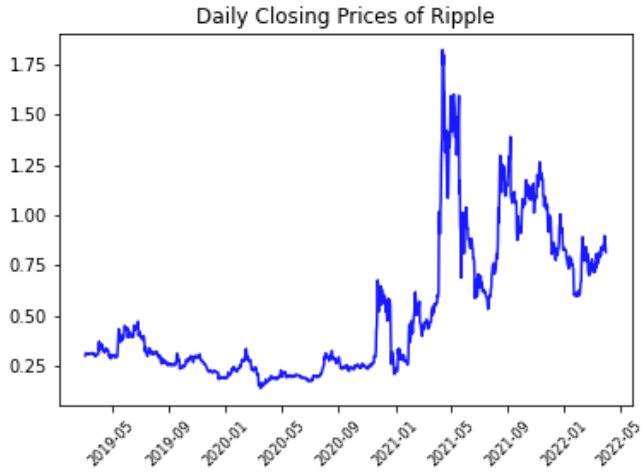


Figure 11: Plot - daily closing prices of Ripple (own illustration)

3.3 Data Analysis

The process of data analysis starts with the retrieval of the clean data from the SQLite database. To conduct the following analysis in an efficient manner, three different datasets shall be retrieved. One table includes all variables while the second and third only contain the dependent or the independent variables. In a next step, the data is split into four sub-periods and the descriptive statistics (mean, standard deviation, min, and max) for each of them are calculated. The results can be found in the appendix chapter 8.2. Afterwards, we ran the stationarity test and conducted several OLS regressions with different combinations of at least five independent variables in order to identify the most promising model for every dependent variable. The following chapters will provide a deeper inside.

4 Empirical Model

4.1 Identification of Non-Stationarity

In chapter 2 stationarity and non-stationarity were discussed and its consequences were illuminated. The quintessence is that the time series must be stationary to perform a correct regression analysis. To check this, we completed the Augmented-Dickey-Fuller test in Python. Here, the first thing to do is to import the relevant library "statsmodels" to be able to apply the "adfuller" function. The selected dependent variables were all non-stationary, i.e. a trend was present. With the help of lags (absolute Y-difference from t-1 to t), the respective movements could be eliminated. After applying two lags, the H_o hypothesis *(Time series is nonstationary)* could be rejected in the Dickey-Fuller test.

4.2 Implementation of Ordinary Least Square Regression (OLS)

Chapter 2.1.1 presented the methodology of OLS regression. Once the dataset was cleaned, the data was converted from absolute closing prices to daily returns, which is necessary to make the different variables comparable. The OLS regression is performed by importing the library "statsmodels" with the function "OLS". In our analysis, we did not conduct a single OLS regression but worked with a loop to test all different combinations of independent variables in order to identify the most promising model.

4.2.1 Variable Selection and Analysis of Robust Models

For the multiple OLS regressions, each crypto- or fiat currency was used once as the target variable with a constant and 13 independent variables (stock indices & commodities). In order to identify the most promising model, several OLS regressions with all different combinations of at least five independent variables were conducted. During that process, a total of 14'912 OLS regressions for every dependent variable was performed. The adjusted-R^2 has to be calculated for each regression, and the one with the highest value can be classified as the best model. Once the most promising model for each crypto- and fiat currency was identified, the Chow test had to be applied to identify any structural breaks and therefore, to determine whether the model is stable over time. In a further step, a rolling window regression was used to assess the stability of the regression coefficients over time. The corresponding plots of the rolling window regression are presented in the appendix, chapter 8.3.

5 Analysis of Results

In this section, the results of the best regression models are presented. The evaluation is based on the adjusted R^2, the P-Value of the Chow tests and the results of the rolling window regressions.

5.1 Summary of Empirical Results

Table 2 provides the results from the various regression models. The three highest adjusted R^2 values are highlighted in green. The adjusted R^2 range between a value of 0.058660 (Ripple) and a value of 0.454870 (CAD). All dependent variables, which show structural breaks according to the Chow test (H_o rejected), are red highlighted. An interesting finding was, that all fiat currencies show a structural break, whereas the data of the cryptocurrencies seem to be stable over time with the exception of Ripple. The graphical plots of the rolling window regression are available in the appendix (chapter 8.3). When comparing the results of the crypto- with the fiat currencies, it is visible that the betas change more by the first mentioned. For the cryptos, the independent variables CSI 300, MSCI Emerging Markets, Nasdaq, S&P 500, and Silver were used in five regression models. Therefore they are the most frequently used independent variables. Among fiat currencies, the independent variables Crude Oil WTI, FTSE 100, and MSCI World were the ones with the best predictive power. These results are available in the tables in the chapters 8.3.1 and 8.3.2.

Table 2: Highest empirical results (own illustration)

Dependent variables	Adjusted R^2	P-Value of Chow test / interpretation
Bitcoin	0.121620	0.15 (no structural break)
Ethereum	0.116140	0.23 (no structural break)
Ripple	0.058660	0.02 (structural break)
Litecoin	0.094113	0.25 (no structural break)
Bitcoin Cash	0.076428	0.56 (no structural break)
EUR	0.349762	0.0 (structural break)
GBP	0.450972	0.0 (structural break)
JPY	0.423140	0.0 (structural break)
CHF	0.336599	0.0 (structural break)
CAD	0.454870	0.0 (structural break)

6 Discussion

6.1 Conclusion

As seen in the results (c.f. table 2 in chapter 5.1), price changes of crypto- and fiat currencies can be modelled by the chosen stock indices and commodities up to a certain point. However, the results have also revealed substantial differences between crypto- and fiat currencies. Despite the 14'912 OLS regressions per dependent variable, fiat currencies achieved substantially higher adjusted R^2. This indicates, that our models were more accurate when predicting price changes of fiat currencies. However, all fiat currencies showed structural breaks in the analyzed dataset and therefore, the modelling with multiple regressions would result in a better outcome. On the other hand, the datasets of the cryptocurrencies, with the exception of Ripple, were shown to be stable over time. Furthermore, the analysis of the rolling window regression revealed that the betas of fiat currencies are more stable over time that the betas of the cryptocurrencies. It can be summarized that we were able identify stable robust models. However, the models for the fiat currencies have substantially better predictive power than the ones for the cryptocurrencies. Nevertheless, the revealed structural breaks need to be taken into account when applying the models.

6.2 Limitation of this Study

This paper focused on building stable and robust regression models for crypto- and fiat currencies. Relatively viewed, cryptocurrencies are a new asset, and as a consequence, there are less complete data available than for conventional currencies. Adding economic parameters could have yielded better results, especially for the countries where cryptocurrencies are mined. However, these economic parameters are only available monthly and would therefore not be compatible with our model.

6.3 Recommendations for Further Research

For future research, we recommend that macroeconomic parameters should be included. National statistical organizations would be the right contact to ensure that the data quality is proper. To minimize the data problem, this research should be carried out with older cryptocurrencies so that the amount of data is sufficient. Furthermore, we recommend to differentiate the two groups, crypto- and fiat currencies in potential future analyses due to the stated explanations.

7 List of References

Bachmann, O. (2021). Econometrics 1 – OLS Estimation of the Linear Regression Model. Fall semester 2021. Zurich: University of applied sciences Zurich, School of Management and Law.

Bachmann, O. (2021). Econometrics 2 – Statistical Inference. Fall semester 2021. Zurich: University of applied sciences Zurich, School of Management and Law.

Bachmann, O. (2021). Econometrics 4 – Model Diagnostics. Fall semester 2021. Zurich: University of applied sciences Zurich, School of Management and Law.

Bachmann, O. (2021). Econometrics 5 – Static and Dynamic Time Series Regression Models. Fall semester 2021. Zurich: University of applied sciences Zurich, School of Management and Law.

Carlson, W. L., Newbold, P., & Thorne B. M. (2013). Statistics for Business and Economics. Global Edition. Essex: Pearson Education Limited.

Deloitte Digital GmbH (2015). «Überlebensstrategie Digital Leadership». Retrieved from https://www2.deloitte.com/content/dam/Deloitte/at/Documents/strategy/ueberlebensstrategie-digital-leadership_final.pdf.

Fintropolis (2020). Investmentbanking: Maschinen statt Rockstars. Warum künstliche Intelligenz für Investmentbanker Fluch und Segen zugleich ist. Retrieved from https://www.fintropolis.de/article/investmentbanking.

Gimeno, R. & Mateos de Cabo, R. (2006). Statistics and Finance: Living on the "Hedge". Working Paper Nr. ICOTS-7. Salvador: International Association for Statistical Education.

Hand, D. (2011). Statistics in Banking. Retrieved from https://doi.org/10.1002/0471667196.ess6000.pub3.

Martin, P. (2021). Linear Regression: an Introduction to statistical models. SAGE Publications.

Oehlert, A. M., Swart, P. K. (2019). Rolling window regression of δ13C and δ18O values in carbonate sediments: Implications for source and diagenesis. Retrieved from https://doi.org/10.1002/dep2.88.

Orpiszewski, T. (2022). Advanced Quantitative Methods Block 5. Spring semester 2022. Zurich: University of applied sciences Zurich, School of Management and Law.

Peng, G. (2010). Zipf's law for Chinese cities: Rolling sample regressions. Physica A: Statistical Mechanics and its Applications, 2010 (Vol. 389) Issue 18.

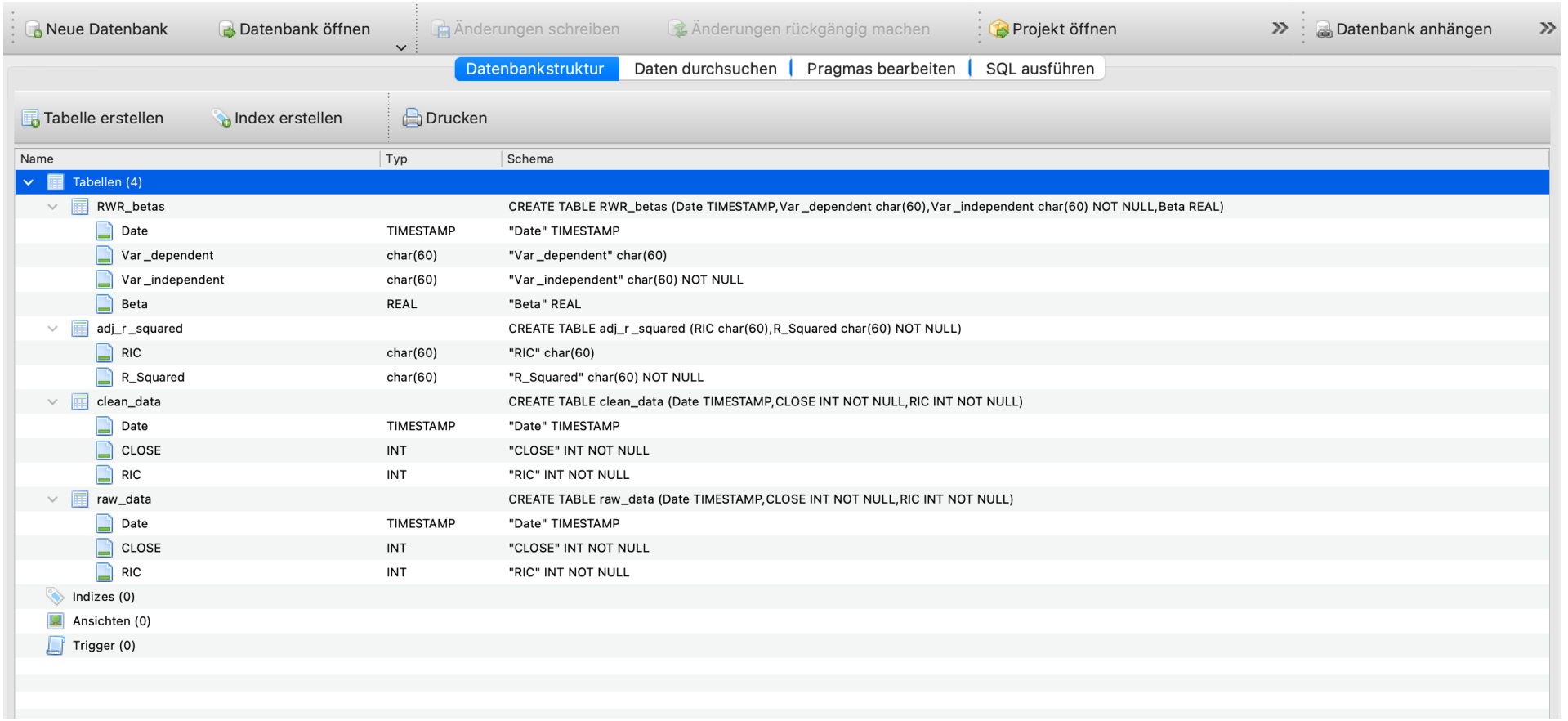
Statistics How to (2016). Chow Test: Definition & Examples. Retrieved from https://www.statisticshowto.com/chow-test/.

Universität Zürich. (o. J.). Kapitel 4. Die Methode der kleinsten Quadrate. Retrieved from https://www.physik.uzh.ch/dam/jcr:8848a522-9271-468d-a889-5307f0dd8bd3/Kap4.pdf.

World Economic Forum (2016). Digital Transformation of Industries: Digital Enterprises. Retrieved from http://reports.weforum.org/digital-transformation/wp-content/blogs.dir/94/mp/files/pages/files/digital-enterprise-narrative-final-january-2016.pdf.

8 Appendix

8.1 SQLite



8.2 Descriptive statistics of the sub-periods

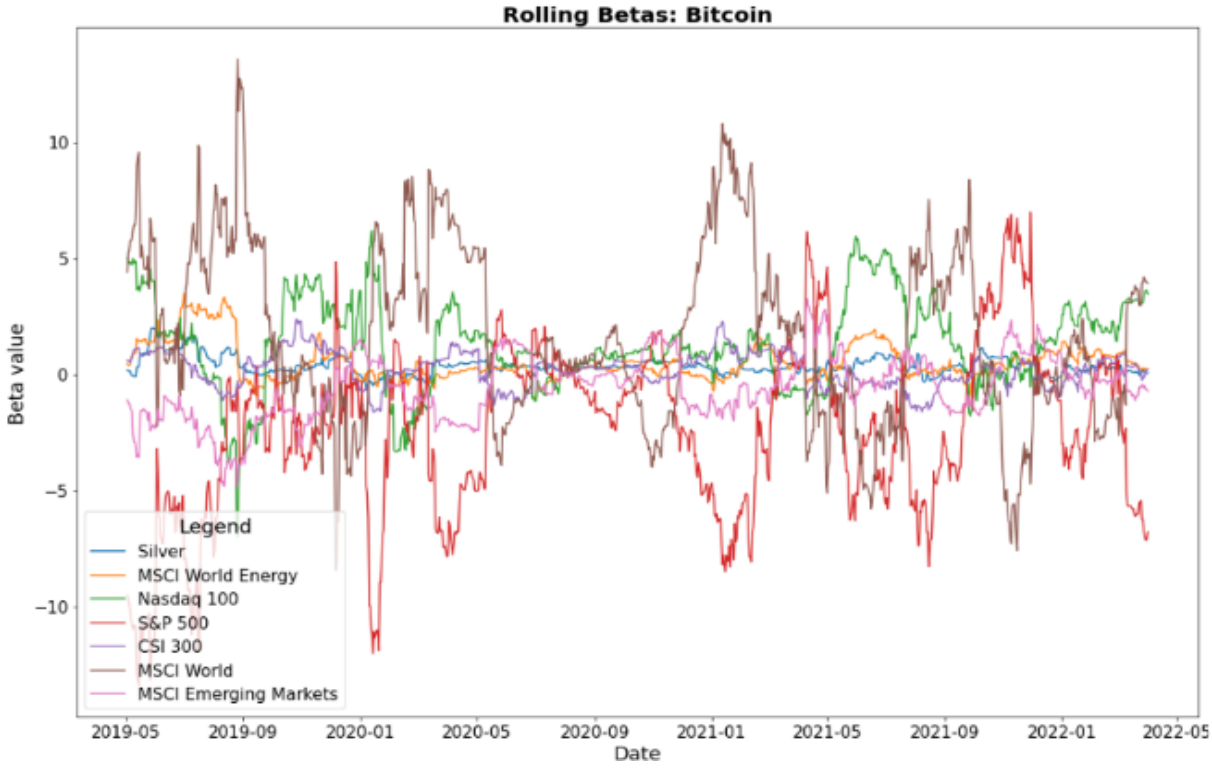
		Statistic	2019-04-01 to 2019-12-31	2020-01-01 to 2020-09-30	2020-10-01 to 2021-06-30	2021-07-01 to 2022-03-31
Mean	**0**	Mean Bitcoin	8533.58	9187.88	36028.85	46380.05
	1	Mean Ethereum	197.20	243.85	1535.01	3292.35
	2	Mean Ripple	0.30	0.22	0.63	0.90
	3	Mean Litecoin	77.44	50.39	155.80	155.46
	4	Mean Bitcoin Cash	299.21	268.28	520.29	485.84
	5	Mean EUR	1.11	1.12	1.20	1.15
	6	Mean GBP	1.27	1.27	1.37	1.36
	7	Mean JPY	108.64	107.54	106.62	113.29
	8	Mean CHF	0.99	0.95	0.91	0.92
	9	Mean CAD	1.33	1.35	1.27	1.26
	10	Mean Crude Oil ICE	64.34	42.66	58.37	83.37
	11	Mean Crude Oil WTI	57.70	38.36	55.40	80.82
	12	Mean Gold	1422.51	1735.05	1828.74	1820.43
	13	Mean Silver	16.40	19.22	25.76	23.89
	14	Mean Bloomberg Commodity Index	79.04	68.64	82.49	103.32
	15	Mean MSCI World Energy	194.15	133.88	142.85	180.72
	16	Mean Nasdaq 100	7853.31	9665.84	12982.05	15188.59
	17	Mean S&P 500	2977.23	3104.48	3864.87	4497.56
	18	Mean FTSE 100	7348.76	6294.71	6618.14	7250.49
	19	Mean CSI 300	3849.61	4193.18	5132.56	4804.36
	20	Mean MSCI World	2189.59	2210.83	2745.57	3092.53
	21	Mean MSCI Emerging Markets	1036.69	1014.71	1299.81	1245.44
	22	Mean MSCI Europe	1664.57	1561.31	1867.83	2025.00
Std	**0**	Std. Dev. Bitcoin	1890.91	1559.87	16197.58	8628.55
	1	Std. Dev. Ethereum	45.95	86.95	937.57	723.24
	2	Std. Dev. Ripple	0.07	0.04	0.41	0.20
	3	Std. Dev. Litecoin	25.84	10.53	76.73	36.94
	4	Std. Dev. Bitcoin Cash	72.41	61.94	265.81	129.20
	5	Std. Dev. EUR	0.01	0.04	0.02	0.03

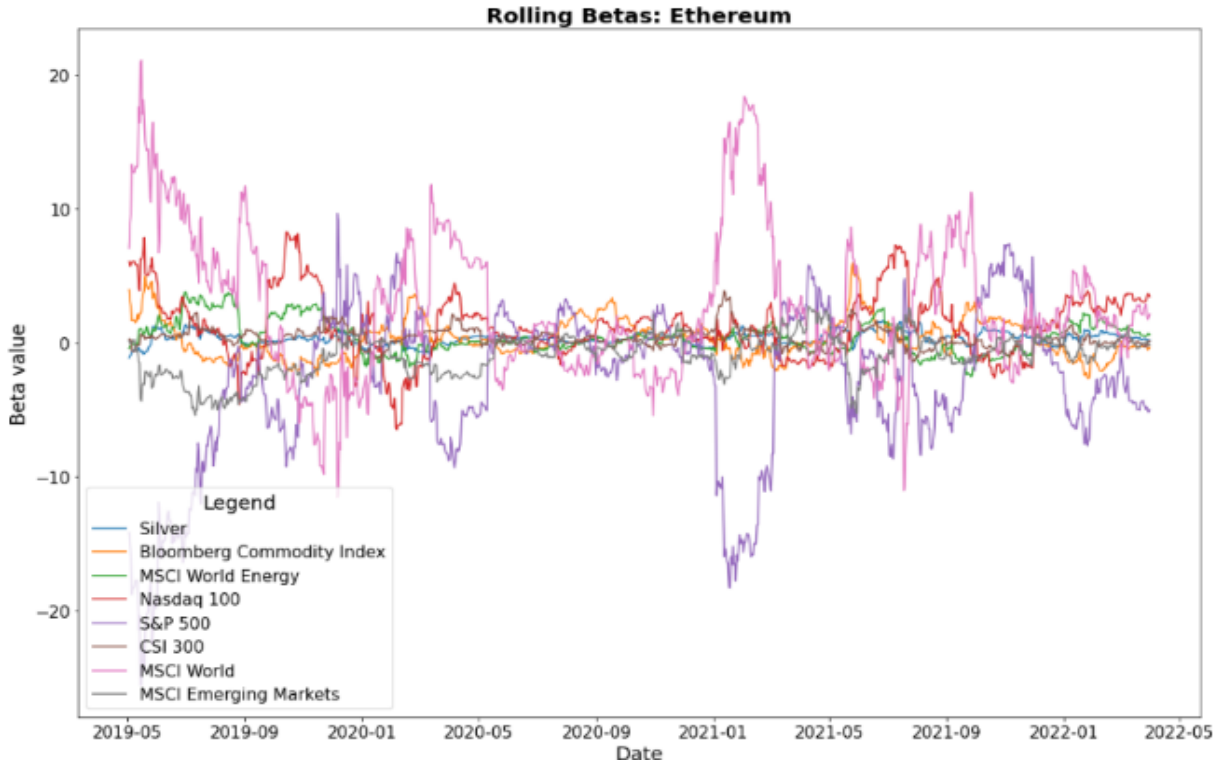
	6	Std. Dev. GBP	0.03	0.04	0.04	0.02
	7	Std. Dev. JPY	1.57	1.65	2.54	2.95
	8	Std. Dev. CHF	0.01	0.03	0.02	0.01
	9	Std. Dev. CAD	0.01	0.04	0.04	0.01
	10	Std. Dev. Crude Oil ICE	4.30	11.55	10.88	13.05
	11	Std. Dev. Crude Oil WTI	3.45	12.65	10.75	12.76
	12	Std. Dev. Gold	88.14	146.06	64.40	58.87
	13	Std. Dev. Silver	1.28	4.33	1.35	1.19
	14	Std. Dev. Bloomberg Commodity Index	1.53	5.95	7.32	9.57
	15	Std. Dev. MSCI World Energy	9.01	29.41	22.31	21.98
	16	Std. Dev. Nasdaq 100	362.64	1224.28	825.55	770.05
	17	Std. Dev. S&P 500	108.09	278.01	273.52	140.53
	18	Std. Dev. FTSE 100	136.17	674.40	391.02	188.68
	19	Std. Dev. CSI 300	128.57	370.82	274.55	233.27
	20	Std. Dev. MSCI World	71.18	202.30	192.81	90.97
	21	Std. Dev. MSCI Emerging Markets	38.62	97.82	91.04	59.79
	22	Std. Dev. MSCI Europe	50.14	160.76	144.09	77.27
Min	**0**	Minimum Bitcoin	4127.67	4926.30	10544.55	29856.77
	1	Minimum Ethereum	122.95	109.31	337.39	1794.37
	2	Minimum Ripple	0.18	0.14	0.21	0.53
	3	Minimum Litecoin	36.87	31.95	44.94	98.04
	4	Minimum Bitcoin Cash	166.49	166.35	216.99	273.32
	5	Minimum EUR	1.09	1.07	1.16	1.09
	6	Minimum GBP	1.20	1.15	1.29	1.30
	7	Minimum JPY	105.29	102.34	102.72	109.04
	8	Minimum CHF	0.97	0.90	0.88	0.90
	9	Minimum CAD	1.30	1.30	1.20	1.23
	10	Minimum Crude Oil ICE	56.23	19.33	37.46	65.18
	11	Minimum Crude Oil WTI	51.13	-36.98	35.64	62.25
	12	Minimum Gold	1270.29	1469.80	1681.24	1726.11
	13	Minimum Silver	14.35	11.98	22.60	21.51
	14	Minimum Bloomberg Commodity Index	75.98	59.48	69.80	91.16

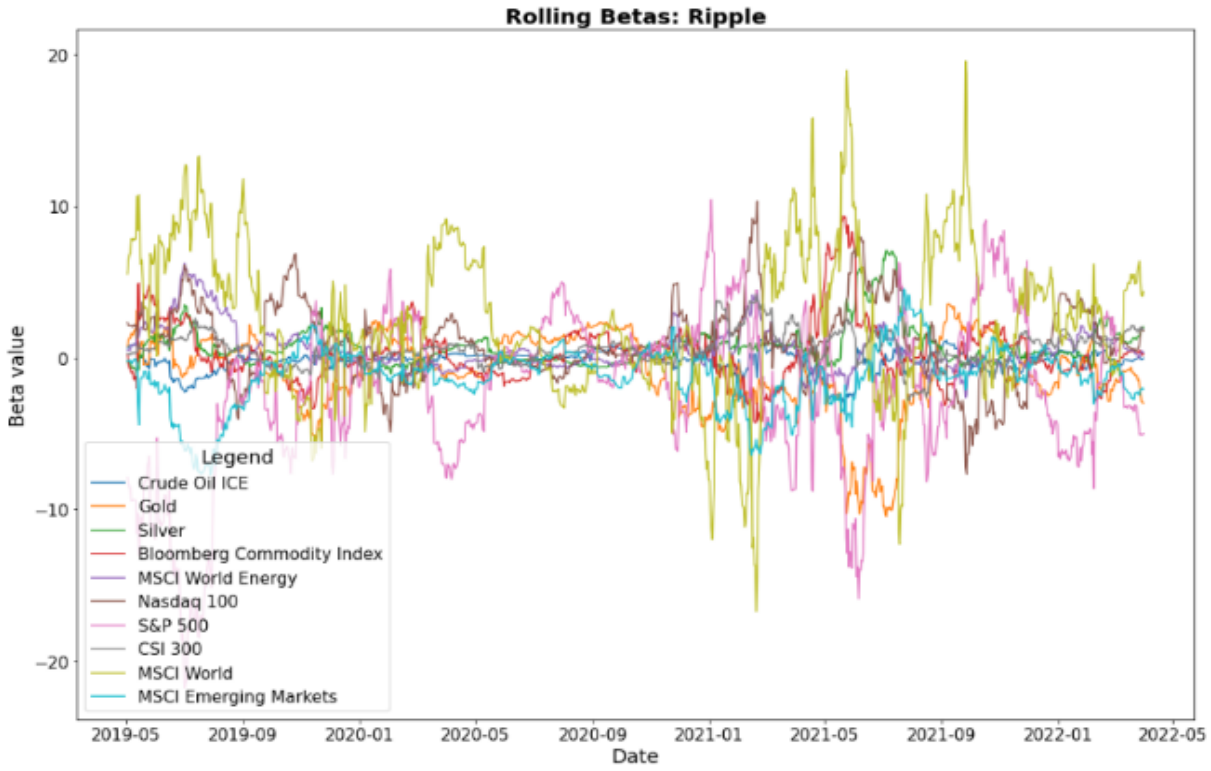
	15	Minimum MSCI World Energy	176.66	80.50	95.15	146.88
	16	Minimum Nasdaq 100	6978.02	6994.29	11052.95	13046.64
	17	Minimum S&P 500	2744.45	2237.40	3269.96	4170.70
	18	Minimum FTSE 100	7067.01	4993.89	5577.27	6844.39
	19	Minimum CSI 300	3564.68	3530.31	4587.40	3983.81
	20	Minimum MSCI World	2045.06	1602.11	2292.93	2797.70
	21	Minimum MSCI Emerging Markets	960.81	758.20	1081.71	1026.77
	22	Minimum MSCI Europe	1555.44	1152.70	1503.61	1720.88
Max	**0**	Maximum Bitcoin	12706.00	12321.37	63398.46	67707.33
	1	Maximum Ethereum	337.44	481.90	4143.45	4800.10
	2	Maximum Ripple	0.47	0.34	1.83	1.39
	3	Maximum Litecoin	139.28	82.68	373.69	270.50
	4	Maximum Bitcoin Cash	475.63	492.32	1466.78	787.10
	5	Maximum EUR	1.14	1.19	1.23	1.19
	6	Maximum GBP	1.33	1.34	1.42	1.40
	7	Maximum JPY	112.17	112.11	111.10	123.91
	8	Maximum CHF	1.02	0.99	0.94	0.94
	9	Maximum CAD	1.35	1.45	1.33	1.29
	10	Maximum Crude Oil ICE	74.57	68.91	76.18	127.98
	11	Maximum Crude Oil WTI	66.24	63.27	74.21	123.64
	12	Maximum Gold	1552.35	2063.19	1951.51	2052.41
	13	Maximum Silver	19.57	29.15	28.97	26.46
	14	Maximum Bloomberg Commodity Index	83.06	81.64	95.03	132.63
	15	Maximum MSCI World Energy	215.59	200.58	175.56	231.43
	16	Maximum Nasdaq 100	8778.31	12420.54	14572.75	16573.34
	17	Maximum S&P 500	3240.02	3580.84	4297.50	4796.56
	18	Maximum FTSE 100	7686.61	7674.56	7184.95	7672.40
	19	Maximum CSI 300	4120.61	4852.96	5807.72	5229.66
	20	Maximum MSCI World	2364.90	2494.10	3025.21	3248.12
	21	Maximum MSCI Emerging Markets	1118.61	1146.83	1444.93	1368.22
	22	Maximum MSCI Europe	1791.26	1801.31	2103.22	2117.18

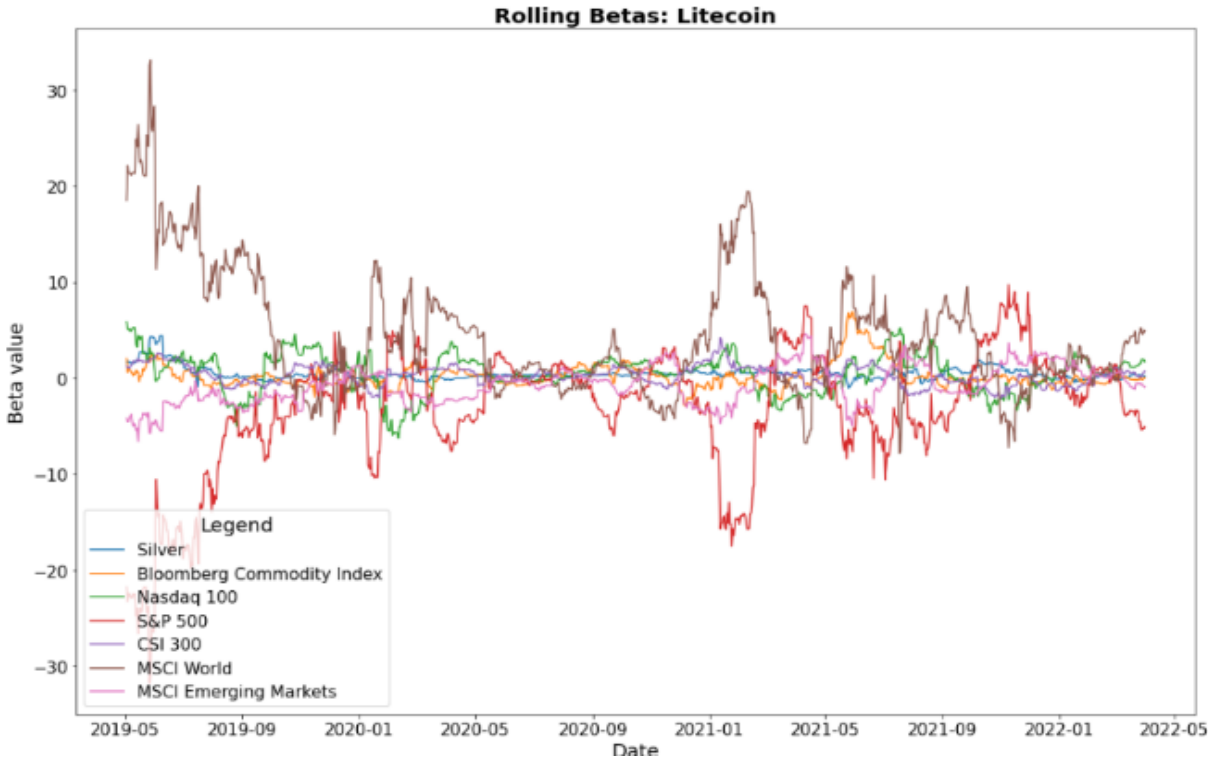
8.3 Results of Rolling Window Regression

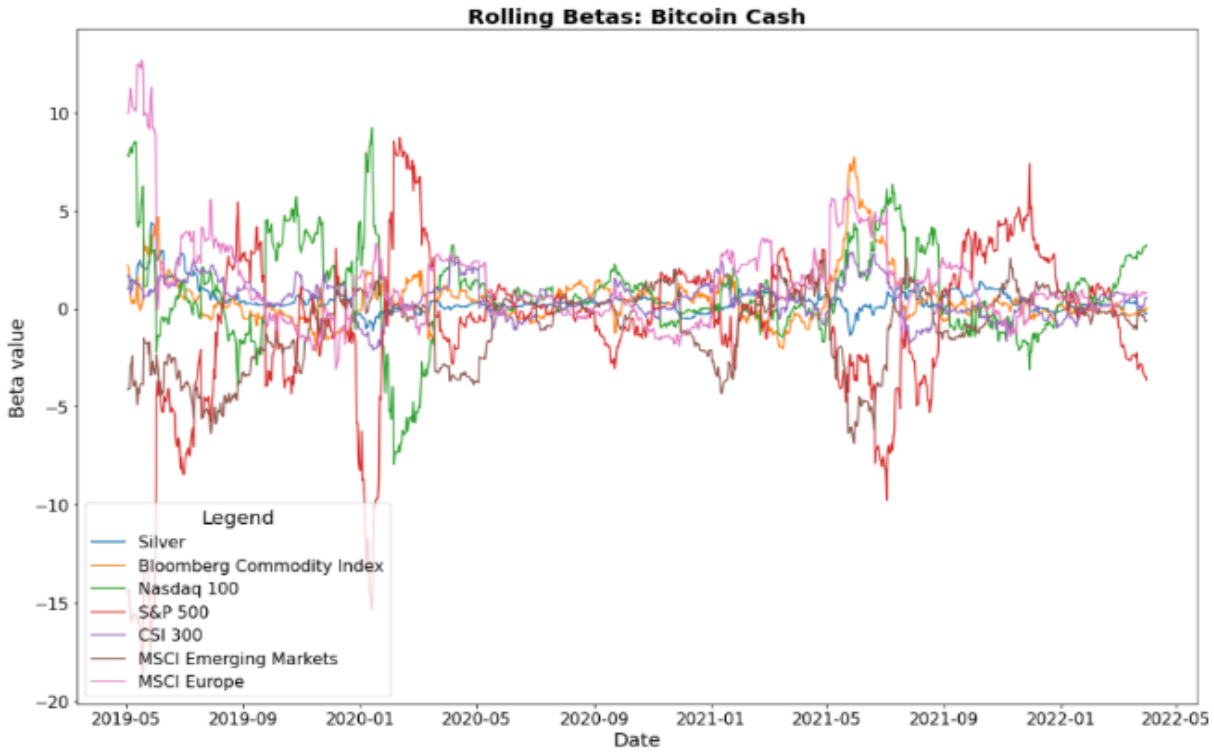
8.3.1 Cryptocurrencies





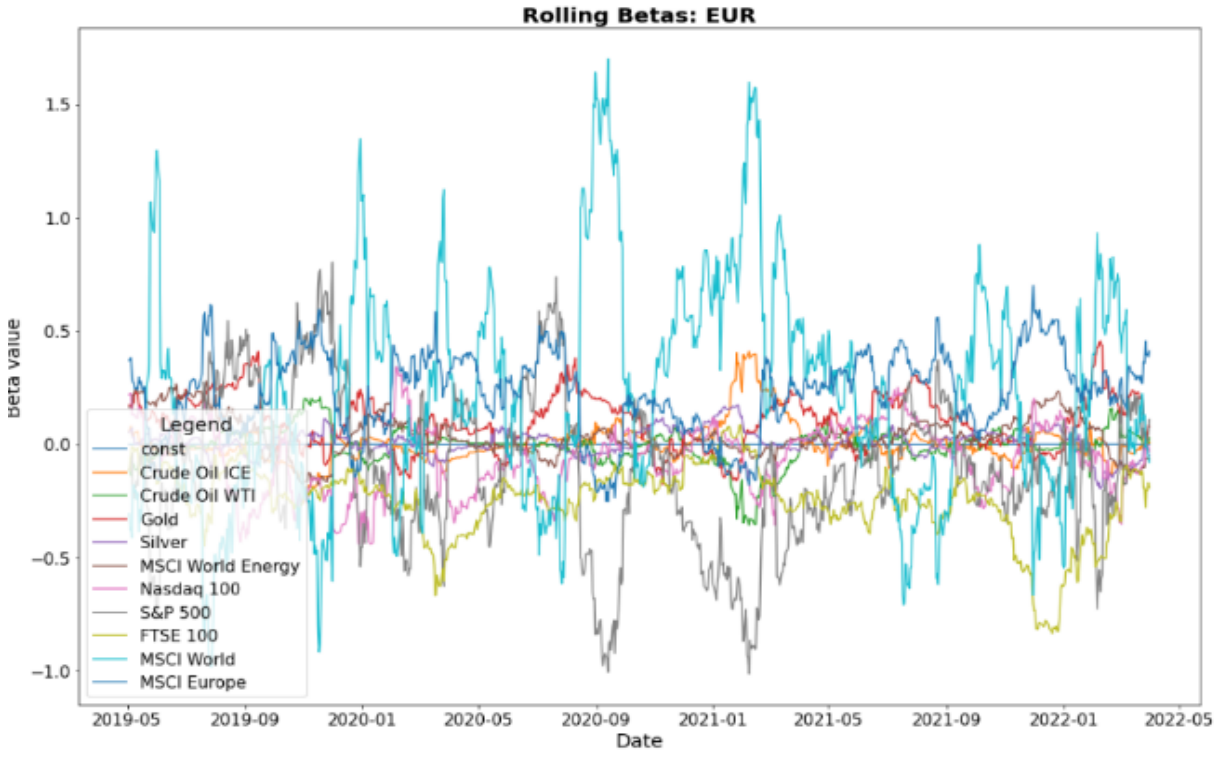

Rolling Betas: Ripple
Beta value
Date
Legend
Crude Oil ICE
Gold
Silver
Bloomberg Commodity Index
MSCI World Energy
Nasdaq 100
S&P 500
CSI 300
MSCI World
MSCI Emerging Markets

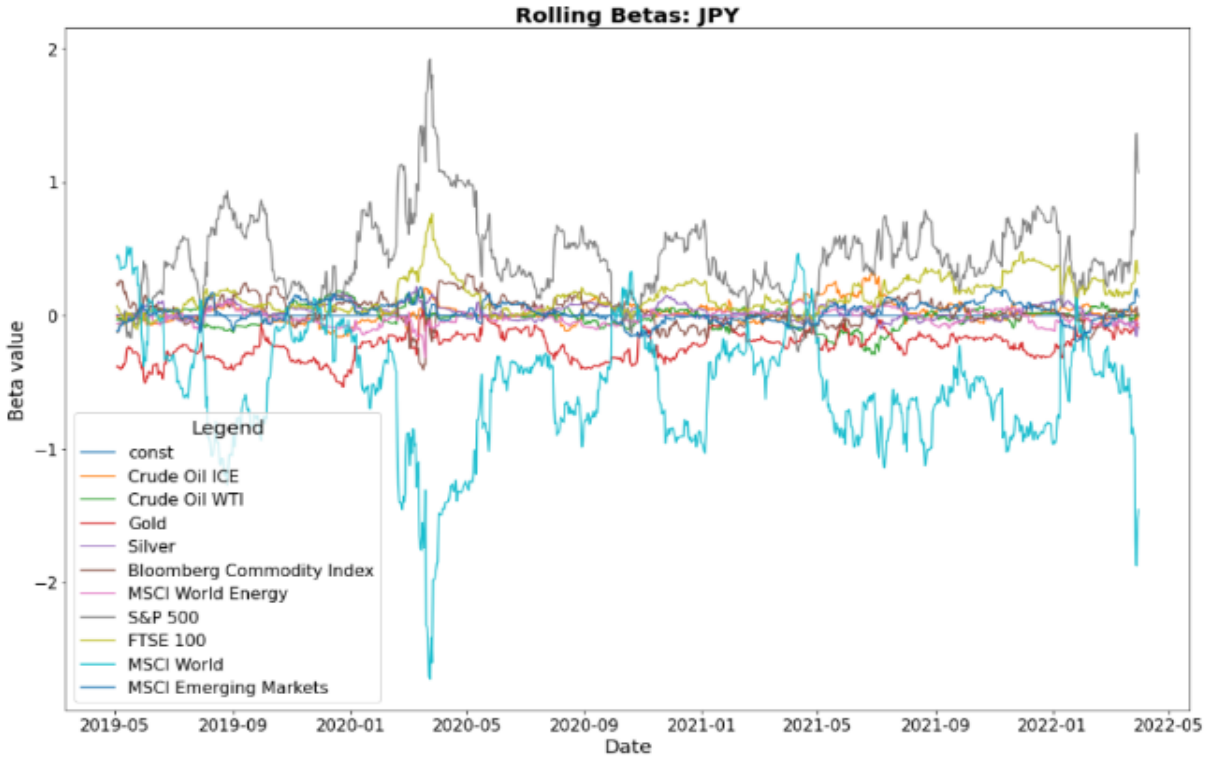

Rolling Betas: Litecoin
Beta value
Date
Legend
Silver
Bloomberg Commodity Index
Nasdaq 100
S&P 500
CSI 300
MSCI World
MSCI Emerging Markets

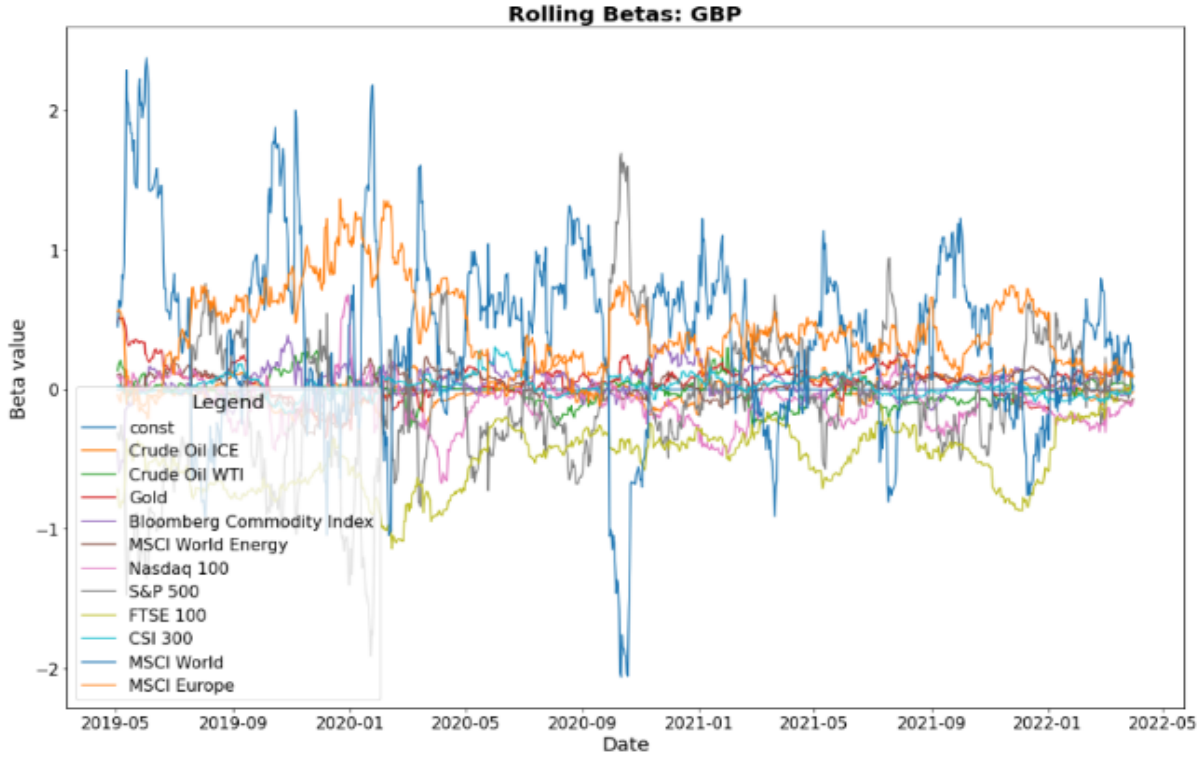


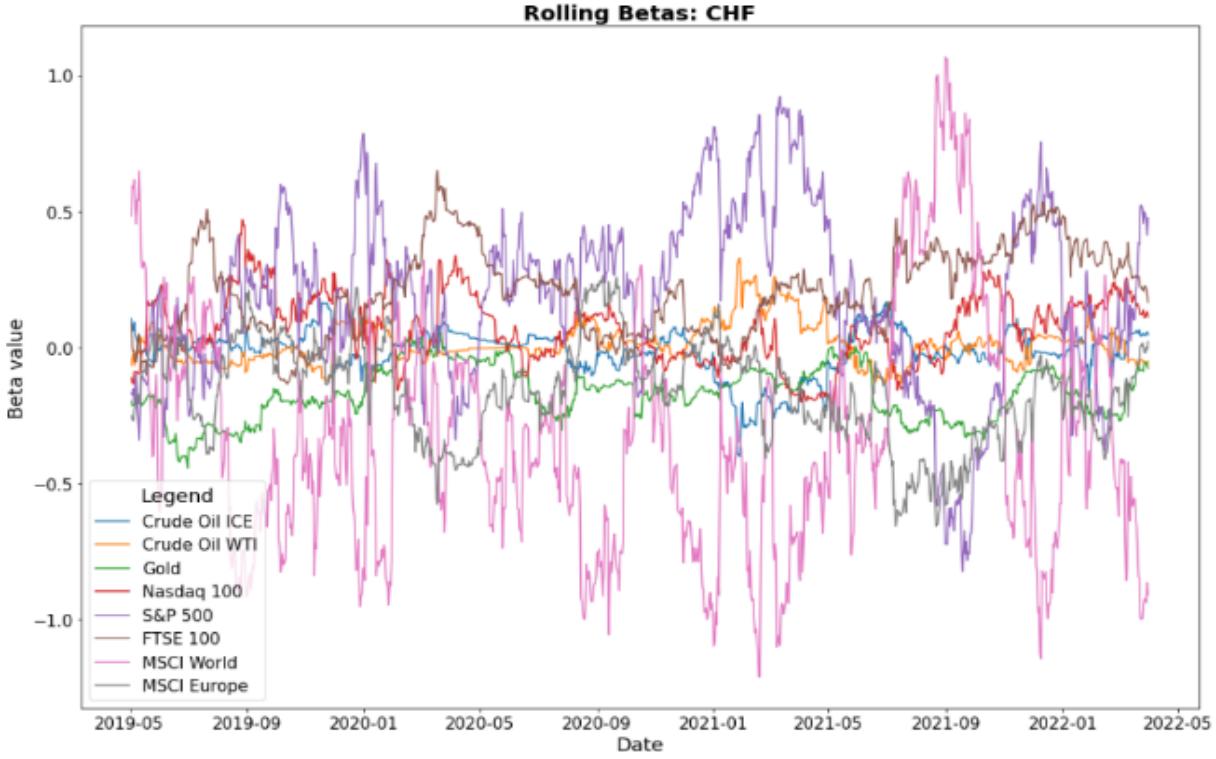
Crypto	
Independent Var.	**Frequency**
CSI 300	5
MSCI Emerging Markets	5
Nasdaq 100	5
S&P 500	5
Silver	5
Bloomberg Commodity Index	4
MSCI World	4
MSCI World Energy	3
Crude Oil ICE	1
Gold	1
MSCI Europe	1

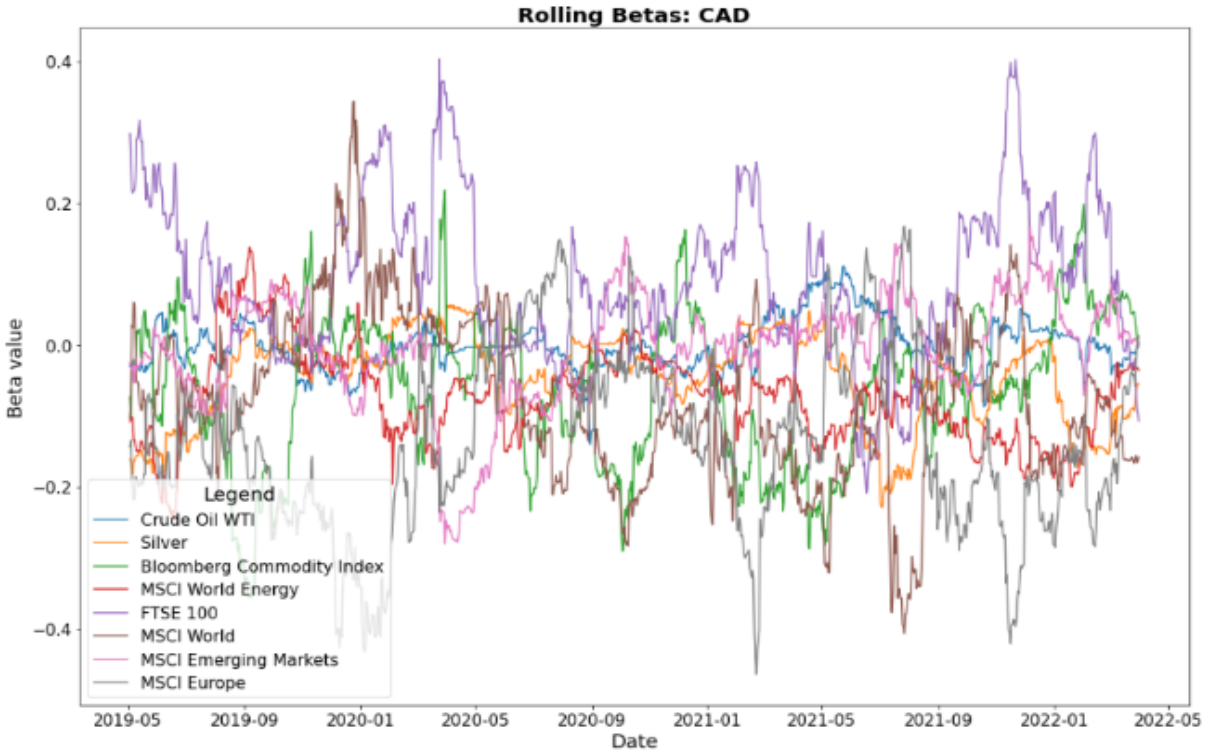
8.3.2 Fiat currencies





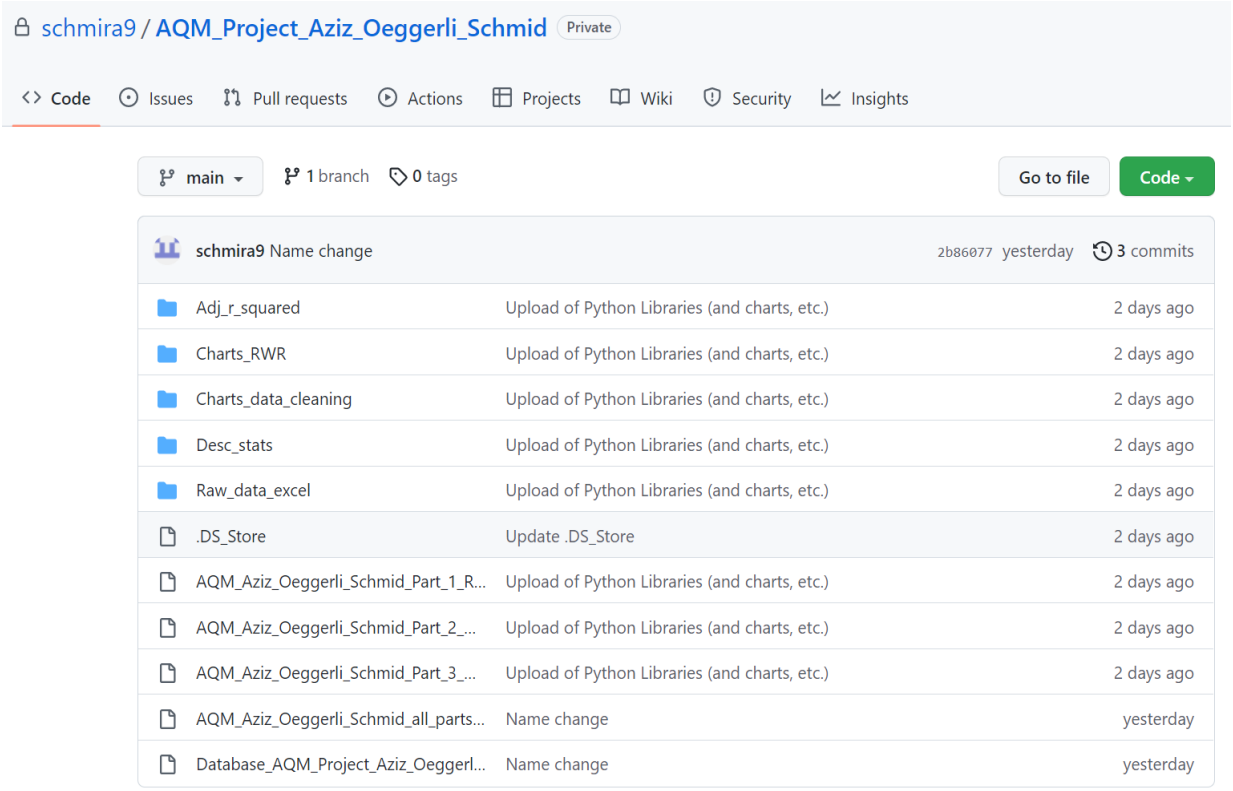

Rolling Betas: GBP
Beta value
Date
Legend
const
Crude Oil ICE
Crude Oil WTI
Gold
Bloomberg Commodity Index
MSCI World Energy
Nasdaq 100
S&P 500
FTSE 100
CSI 300
MSCI World
MSCI Europe
2
1
0
−1
−2
2019-05
2019-09
2020-01
2020-05
2020-09
2021-01
2021-05
2021-09
2022-01
2022-05


Rolling Betas: CHF
Beta value
Date
Legend
Crude Oil ICE
Crude Oil WTI
Gold
Nasdaq 100
S&P 500
FTSE 100
MSCI World
MSCI Europe
1.0
0.5
0.0
−0.5
−1.0
2019-05
2019-09
2020-01
2020-05
2020-09
2021-01
2021-05
2021-09
2022-01
2022-05



Fiat	
Independent Var.	**Frequency**
Crude Oil WTI	5
FTSE 100	5
MSCI World	5
Crude Oil ICE	4
Gold	4
MSCI Europe	4
MSCI World Energy	4
S&P 500	4
Bloomberg Commodity Index	3
Nasdaq 100	3
Silver	3
MSCI Emerging Markets	2
CSI 300	1

8.4 Python Code (Github)



https://github.com/schmira9/AQM_Project_Aziz_Oeggerli_Schmid.git

8.4.1 Part 1 – Raw Data

```
# Part 1 - Raw Data

1. Preparation
2. Download the necessary data from refinitiv/eikon
3. Create SQLite database and insert the raw data into it

## 1. Preparation

#import of the relevant libraries and connect to the eikon database

import eikon as ek
ek.set_app_key("31414bf6047543af95f34c11c29c6cdb1872fe35") #Key Rafael
#ek.set_app_key("5977dcba0e8742aaa39ed8524b65c51ba9cf90e9") #Key Matthias
#ek.set_app_key("9995d23e109d4cb9bc4cd3b5436667901bb26b35") #Key Ahmed
import pandas as pd
pd.set_option("display.max_rows", None, "display.max_columns", None)
import sqlite3
from sqlite3 import Error
import numpy as np
import scipy
from statsmodels.tsa.stattools import adfuller
from statsmodels.regression.rolling import RollingOLS
import statsmodels.api as sm
import matplotlib.pyplot as plt
from matplotlib.dates import DateFormatter
import matplotlib.dates as mdates
import itertools as it

#### 1.1 - Generate multiple lists for the data under investigation
```

```
30.
31. #generate a list of the cryptocurrencies
32. rics_crypto = ['BTC=', #Bitcoin
33.                'ETH=', #Ethereum
34.                'XRP=', #Ripple
35.                'LTC=', #Litecoion
36.                'BCH='] #Bitcoin Cash
37.
38. #generate a list of the fiat currencies
39. rics_currency = ['EUR=', #Euro
40.                  'GBP=', #Pound Sterling
41.                  'JPY=', #Japanese Yen
42.                  'CHF=', #Swiss Francs
43.                  'CAD='] #Canadian Dollar
44.
45. #generate a list of the commodities
46. rics_commodities = ['LCOc1', #Crude Oil (ICE Europe Brent Crude Electronic Energy Future)
47.                     'WTC-', #Crude Oil (WTI Cushing US FOB)
48.                     'XAU=', #Gold
49.                     'XAG=', #Silver
50.                     '.BCOM', #Bloomberg Commodity Index
51.                     '.dMIWO0EN00PUS'] #MSCI World Energy Index USD (End of Day)
52.
53. #generate a list of the stock indices
54. rics_stockindex = ['.NDX', #Nasdaq 100 Index
55.                    '.SPX', #S&P 500 Index
56.                    '.FTSE', #FTSE 100 Index
57.                    '.CSI300', #China Securities Index 300
58.                    '.dMIWO00000PUS', #MSCI World Price Index USD (End of Day)
59.                    '.dMIEF00000PUS', #MSCI Emerging Markets Price Index USD (End of Day)
60.                    '.dMIEU00000PUS'] #MSCI Europe Price Index USD (End of Day)
61.
62. #### 1.2 - Generate some further lists which will be helpful later on
63.
64. #generate a list of all rics
65. rics = rics_crypto + rics_currency + rics_commodities + rics_stockindex
66.
67. #generate a list of the dependent variables (y)
68. rics_dependent_variables = rics_crypto + rics_currency
69.
70. #generate a list of the independent variables (x)
71. rics_independent_variables = rics_commodities + rics_stockindex
72.
73. #generate a list of the desired names of the columns
74. columns =['Bitcoin',
75.           'Ethereum',
76.           'Ripple',
77.           'Litecoin',
78.           'Bitcoin Cash',
79.           'EUR',
80.           'GBP',
81.           'JPY',
82.           'CHF',
83.           'CAD',
84.           'Crude Oil ICE',
85.           'Crude Oil WTI',
86.           'Gold',
87.           'Silver',
88.           'Bloomberg Commodity Index',
89.           'MSCI World Energy',
90.           'Nasdaq 100',
91.           'S&P 500',
92.           'FTSE 100',
93.           'CSI 300',
94.           'MSCI World',
95.           'MSCI Emerging Markets',
96.           'MSCI Europe']
97.
98. #generate a list of the desired names of the columns of the dependent variables (crypto
    and other currencies)
```

```
columns_dependent_variables = ['Bitcoin',
                               'Ethereum',
                               'Ripple',
                               'Litecoin',
                               'Bitcoin Cash',
                               'EUR',
                               'GBP',
                               'JPY',
                               'CHF',
                               'CAD']

#generate a list of the desired names of the columns of the independent variables (commodities & indices)
columns_independent_variables = ['Crude Oil ICE',
                               'Crude Oil WTI',
                               'Gold',
                               'Silver',
                               'Bloomberg Commodity Index',
                               'MSCI World Energy',
                               'Nasdaq 100',
                               'S&P 500',
                               'FTSE 100',
                               'CSI 300',
                               'MSCI World',
                               'MSCI Emerging Markets',
                               'MSCI Europe']

#generate a list of the desired start and end date of the analysis
startdate = '2012-01-04'
enddate = '2022-03-31'

## 2. Download the necessary data from refinitiv/eikon

#### 2.1 - We create loops to download all data for the last 10 years and merge all four dataframes into one

#Loop to download time series for cryptos (the loop helps to deal with the datapoint limit per request)
crypto_histo = pd.DataFrame()

for i in range (0,len(rics_crypto)):
    data_input = ek.get_timeseries(rics_crypto[i],
                                   start_date=startdate,
                                   end_date=enddate,
                                   fields='CLOSE',
                                   interval='daily',
                                   corax = 'adjusted')
    data_input['RIC'] = rics_crypto[i]
    crypto_histo = crypto_histo.append(data_input)


#Loop to download time series for fiat currencies
currency_histo = pd.DataFrame()

for i in range (0,len(rics_currency)):
    data_input = ek.get_timeseries(rics_currency[i],
                                   start_date=startdate,
                                   end_date=enddate,
                                   fields='CLOSE',
                                   interval='daily',
                                   corax = 'adjusted')
    data_input['RIC'] = rics_currency[i]
    currency_histo = currency_histo.append(data_input)


#Loop to download time series for commodities
commodities_histo = pd.DataFrame()

for i in range (0,len(rics_commodities)):
    data_input = ek.get_timeseries(rics_commodities[i],
```

```
                                    start_date=startdate,
                                    end_date=enddate,
                                    fields='CLOSE',
                                    interval='daily',
                                    corax = 'adjusted')
        data_input['RIC'] = rics_commodities[i]
        commodities_histo = commodities_histo.append(data_input)


    #Loop to download time series for stockindices
    stockindex_histo = pd.DataFrame()

    for i in range (0,len(rics_stockindex)):
        data_input = ek.get_timeseries(rics_stockindex[i],
                                    start_date=startdate,
                                    end_date=enddate,
                                    fields='CLOSE',
                                    interval='daily',
                                    corax = 'adjusted')
        data_input['RIC'] = rics_stockindex[i]
        stockindex_histo = stockindex_histo.append(data_input)


    #merge all four time series into one dataframe
    list_data_parameters = [crypto_histo, currency_histo, commodities_histo, stockin-
dex_histo]
    all_data_merged = pd.concat(list_data_parameters)
    print(all_data_merged.shape)
    all_data_merged.head()

    #### 2.2 - Verify the correctness of the data from the API by comparing it with the
data which we retrieved via Excel

    #convert the dataframe "all_data_merged* into wide format to make it comparable
    raw_data_API = all_data_merged.copy()
    raw_data_API_index = raw_data_API.index.strftime('%Y-%m-%d')
    raw_data_API['Date'] = raw_data_API_index
    raw_data_API = raw_data_API.pivot(index='Date',columns='RIC',values='CLOSE')[rics]
    raw_data_API.columns = columns
    raw_data_API.index = pd.to_datetime(raw_data_API.index)

    #load the excel file (with the data from Refinitiv/Eikon) to python and prepare it for
the comparison
    raw_data_excel = pd.read_excel("Raw_data_excel/raw_data_excel.xlsx")
    raw_data_excel=raw_data_excel.iloc[:,1:]
    raw_data_excel.dropna(axis=0, how="any", inplace=True)
    raw_data_excel.columns = ['RIC','Date','CLOSE']
    print(raw_data_excel.isnull().sum())
    raw_data_excel = raw_data_excel.pivot_table(index='Date',columns='RIC',values='CLOSE',
aggfunc="sum")[rics]
    raw_data_excel.columns = columns
    raw_data_excel.index = pd.to_datetime(raw_data_excel.index)

    #compare the tail of the two dataframes to verify the correctness of the dataset
    #NaN's/<NA> indicate that both datasets(API and excel) do not show a value (this will
be cleaned in a next step)
    raw_data_API.tail()==raw_data_excel.tail()

    ## 3. Create SQLite database and insert the raw data into it

    #create a SQLite database
    conn = sqlite3.connect('AQM_Project_Aziz_Oeggerli_Schmid.db')
    c = conn.cursor()
    c.execute('''CREATE TABLE raw_data
             (Date TIMESTAMP,
             CLOSE           INT    NOT NULL,
             RIC             INT    NOT NULL)''')

    print("Table created successfully")

```

```
231.
232.  #create a new column with the date
233.  #this will later be used to re-genereate the dattime index of the dataframe after pull-
    ing the data from our SQLite database
234.  index = all_data_merged.index.strftime('%Y-%m-%d')
235.  all_data_merged['Date1'] = index
236.
237.
238.  #transform the dataframe into a list
239.  #this needs to be done so that we can insert the data into the before created SQLite
    database
240.  all_data_raw_to_insert = all_data_merged.values.tolist()
241.
242.
243.  #upload the data to our SQLite database
244.  c = conn.cursor()
245.  c.executemany("INSERT INTO raw_data(CLOSE, RIC, Date) VALUES (?,?,?)",
    all_data_raw_to_insert)
246.  conn.commit()
247.
248.  print("Upload to SQL-database successful")
```

8.4.2 Part 2 – Data Cleansing

```
249.  # Part 2 - Data Cleansing
250.
251.  1. Retrieve the raw data from our SQL-database
252.  2. Pre-analysis of the data and plotting
253.  3. Create SQLite database and insert the clean data into it
254.
255.  ## 1. Retrieve the raw data from our SQL-database
256.
257.  #pull the raw data from the SQLite database to python
258.  conn = sqlite3.connect('AQM_Project_Aziz_Oeggerli_Schmid.db')
259.  c = conn.cursor()
260.  c.execute("SELECT * FROM raw_data")
261.  new_data_raw = c.fetchall()
262.
263.
264.  #convert the list to a dataframe using the before defined columns as header
265.  new_data_raw = pd.DataFrame(new_data_raw,columns=['Date','CLOSE','RIC'])
266.
267.
268.  #pivot the tables after RIC and set the date as index --> with the '[rics]' at the end,
    the order of the columns stays the same
269.  new_data_raw = new_data_raw.pivot(index='Date',columns='RIC',values='CLOSE')[rics]
270.
271.
272.  #change the names of the columns of the dataframe
273.  new_data_raw.columns = columns
274.
275.
276.  #switch format of the index to datetime
277.  new_data_raw.index = pd.to_datetime(new_data_raw.index)
278.
279.  print(new_data_raw.shape)
280.  new_data_raw.head()
281.
282.  ## 2. Pre-analysis of the data and plotting
283.
284.  #### 2.1 - We do some pre-analysis to figure out the earliest date on which we have
    data available for all assets
285.
286.  #in our case the assumption is, that cryptocurrencies have the shortest data availabil-
    ity
```

```
#with the following codes we check which cryptocurrency does have the least data availability

# new_data_raw[new_data_raw['Bitcoin'].isnull()].tail(5)
# new_data_raw[new_data_raw['Ethereum'].isnull()].tail(5)
# new_data_raw[new_data_raw['Ripple'].isnull()].tail(5)
# new_data_raw[new_data_raw['Litecoin'].isnull()].tail(5)
# new_data_raw[new_data_raw['Bitcoin Cash'].isnull()].tail(5)


#the result shows, that Ripple does have the least data availability (starting from 04.03.2019)
new_data_raw[new_data_raw['Ripple'].isnull()].tail(5)


#therefore, we can now drop all columns in which Ripple shows a NaN
new_data_raw.dropna(subset=['Ripple'],inplace=True)

#Verification:
new_data_raw.isnull().sum()

#results show, that cryptos do not have any NaN anymore and the rest +/- have a similar amount of NaN
#which indicates that these can mostly be explained by public holidays or weekends
#in a next step, this assumption will be verified by plotting the different columns to conduct

#### 2.2 - Plotting our data to conduct a visual check and identify potential anomalies in the dataset

#plot and save individual time series --> takes approx. 5-10 minutes
#as already assumed, the charts do not show any anomalies --> therefore, we can go on with the data cleansing

for col in new_data_raw.iteritems():
    print('plotting'+col[0])
    fig, ax = plt.subplots()
    indicator=new_data_raw[col[0]]
    ax.plot(indicator, alpha=0.9, color='blue')
    plt.xticks(fontsize=8,rotation=45)
    plt.title('Daily Closing Prices of '+col[0])
    plt.savefig('Charts_data_cleaning/Plot_'+col[0]+'.png')
    plt.close()


#the next and final step would be the fill the NaN due to WE/public holidays with the value of the previous day

#### 2.3 - The final step of the data cleansing is to fill the remaining NaN

#our previous analysis has shown that the remaining NaN can be lead back to weekends/public holidays on which fiat currencies, commidities and stock indices do not have a price
#since our data follows a trend, but no seasonality, we can use a linear interpolation in forwarding direction (filling the NaN with the data of the previous day)
all_data_clean = new_data_raw.fillna(axis=0, method='ffill')

#verify that no more NaN are available in the dataset
print(all_data_clean.isnull().sum().sum())
all_data_clean.head()

## 3. Create SQLite database and insert the clean data into it

#create a SQLite database
conn = sqlite3.connect('AQM_Project_Aziz_Oeggerli_Schmid.db')
c = conn.cursor()
c.execute('''CREATE TABLE clean_data
          (Date TIMESTAMP,
          CLOSE           INT    NOT NULL,
          RIC            INT     NOT NULL)''')
```

```
347.
348.  print("Table created successfully")
349.
350.
351.  #change the names of the columns back to the RIC's (so that we have consistent names in
    our different SQL databases)
352.  all_data_clean.columns = rics
353.
354.
355.  #switch format of the index to datetime
356.  all_data_clean.index = pd.to_datetime(all_data_clean.index)
357.
358.
359.  #create a new column with the date
360.  #this will later be used to re-genereate the dattime index of the dataframe after pull-
    ing them from our SQLite database
361.  index = all_data_clean.index.strftime('%Y-%m-%d')
362.  all_data_clean['Date1'] = index
363.
364.
365.  #transformation into long format and set new column names
366.  all_data_clean_to_insert = pd.melt(all_data_clean, id_vars = ['Date1'], value_vars =
    all_data_clean.columns.values.tolist()[:-1])
367.  all_data_clean_to_insert.columns = ['Date','RIC','CLOSE']
368.
369.
370.  #transform the dataframe into a list
371.  #this needs to be done so that we can insert the data into the before created SQLite
    database
372.  all_data_clean_to_insert_list = all_data_clean_to_insert.values.tolist()
373.
374.
375.  #upload the data to our SQLite database
376.  c = conn.cursor()
377.  c.executemany("INSERT INTO clean_data(Date, RIC, CLOSE) VALUES (?,?,?)",
    all_data_clean_to_insert_list)
378.  conn.commit()
379.
380.  print("Upload to SQL-database successful")
```

8.4.3 Part 3 – Data Analysis

```
381.  # Part 3 - Data Analysis
382.
383.  1. Retrieve the clean data from our SQL-database
384.  2. Split the data into 4 sub-periods and calculate the descriptive statistics for each
    of them
385.  3. For each asset, run the stationarity test and decide which variable is stationary
386.  4. Conduct several OLS regressions with different combinations of independent varia-
    bles, choose the most promising model and verify whether the model and the betas are sta-
    ble over time
387.
388.  ## 1. Retrieve the clean data from our SQL-database
389.
390.  #### 1.1 - Pull all data (dependent & independent variables) from the SQL-database
391.
392.  #pull the data from the SQLite database to python
393.  conn = sqlite3.connect('AQM_Project_Aziz_Oeggerli_Schmid.db')
394.  c = conn.cursor()
395.  c.execute("SELECT * FROM clean_data")
396.  new_data_clean = c.fetchall()
397.
398.
399.  #convert the list to a dataframe using the before defined columns as header
400.  new_data_clean = pd.DataFrame(new_data_clean,columns=['Date','CLOSE','RIC'])
```

```
401.
402.
403.  #pivot the tables after RIC and set the date as index --> with the '[rics]', the order
    of the columns stays the same
404.  new_data_clean = new_data_clean.pivot(index='Date',columns='RIC',values='CLOSE')[rics]
405.
406.
407.  #change the names of the columns of the dataframe
408.  new_data_clean.columns = columns
409.
410.
411.  #switch format of the index to datetime
412.  new_data_clean.index = pd.to_datetime(new_data_clean.index)
413.
414.  print(new_data_clean.shape)
415.  new_data_clean.head()
416.
417.  #### 1.2. Pull only the dependent variables from the SQL-database
418.
419.  #pull the data of the dependent variables from the SQLite database to python
420.  conn = sqlite3.connect('AQM_Project_Aziz_Oeggerli_Schmid.db')
421.  c = conn.cursor()
422.  c.execute("SELECT * FROM clean_data WHERE RIC in" + str(tuple(rics_dependent_varia-
    bles)))
423.  dependent_variables = c.fetchall()
424.
425.
426.  #convert the list to a dataframe using the before defined columns as header
427.  dependent_variables = pd.DataFrame(dependent_variables,columns=['Date','CLOSE','RIC'])
428.
429.
430.  #pivot the tables after RIC and set the date as index --> with the '[rics]', the order
    of the columns stays the same
431.  dependent_variables = dependent_variables.pivot(index='Date',columns='RIC',val-
    ues='CLOSE')[rics_dependent_variables]
432.
433.
434.  #change the names of the columns of the dataframe
435.  dependent_variables.columns = columns_dependent_variables
436.
437.
438.  #switch format of the index to datetime
439.  dependent_variables.index = pd.to_datetime(dependent_variables.index)
440.
441.  print(dependent_variables.shape)
442.  dependent_variables.head()
443.
444.  #### 1.3 - Pull the independent variables from the SQL-database
445.
446.  #pull the data of the independent variables from the SQLite database to python
447.  conn = sqlite3.connect('AQM_Project_Aziz_Oeggerli_Schmid.db')
448.  c = conn.cursor()
449.  c.execute("SELECT * FROM clean_data WHERE RIC in" + str(tuple(rics_independent_varia-
    bles)))
450.  independent_variables = c.fetchall()
451.
452.
453.  #convert the list to a dataframe using the before defined columns as header
454.  independent_variables = pd.DataFrame(independent_variables,col-
    umns=['Date','CLOSE','RIC'])
455.
456.
457.  #pivot the tables after RIC and set the date as index --> with the '[rics]', the order
    of the columns stays the same
458.  independent_variables = independent_variables.pivot(index='Date',columns='RIC',val-
    ues='CLOSE')[rics_independent_variables]
459.
460.
461.  #change the names of the columns of the dataframe
462.  independent_variables.columns = columns_independent_variables
```

```
463.
464.
465.  #switch format of the index to datetime
466.  independent_variables.index = pd.to_datetime(independent_variables.index)
467.
468.  print(independent_variables.shape)
469.  independent_variables.head()
470.
471.  ## 2. Split the data into 4 sub-periods and calculate the descriptive statistics for
    each of them
472.
473.  #### 2.1 - Create smaller datasets (4 sub-periods)
474.
475.  #create smaller datasets by selecting dates from the index
476.  sample_1 = new_data_clean['2019-04-01':'2019-12-31']
477.  sample_2 = new_data_clean['2020-01-01':'2020-09-30']
478.  sample_3 = new_data_clean['2020-10-01':'2021-06-30']
479.  sample_4 = new_data_clean['2021-07-01':'2022-03-31']
480.
481.  #### 2.2 - Calculate the descriptive statistices for each of the 4 sub-periods
482.
483.  #create a loop to calculate the most important descriptive statistics
    (mean,std,min,max) for the 4 subsamples
484.  #if required, this could be amended with additional descr. statistics
485.
486.
487.  #create a list which will then be used as columns
488.  subsample_list=['Statistic','2019-04-01 to 2019-12-31','2020-01-01 to 2020-09-
    30','2020-10-01 to 2021-06-30','2021-07-01 to 2022-03-31']
489.
490.
491.  #create an empty table and fill it with NaN
492.  def nans(shape, dtype=float):
493.      a = np.empty(shape, dtype)
494.      a.fill(np.nan)
495.      return a
496.  table_empty = nans([23,5])
497.
498.
499.  #create 4 dataframes, each is used for another descr. statistic (mean,std,min,max), us-
    ing the before defined list and table
500.  desc_stat_mean = pd.DataFrame(table_empty,columns=subsample_list)
501.  desc_stat_std = pd.DataFrame(table_empty,columns=subsample_list)
502.  desc_stat_min = pd.DataFrame(table_empty,columns=subsample_list)
503.  desc_stat_max = pd.DataFrame(table_empty,columns=subsample_list)
504.
505.  #fill the 4 dataframes (mean, std, min and max) with data of the 4 sub-periods
506.  samples = [sample_1,sample_2,sample_3,sample_4]
507.  a=1
508.  for b in samples:
509.      for i in range(b.shape[1]):
510.          if a == 1:
511.              desc_stat_mean.iloc[i, 0]= "Mean " + b.columns[i]                    #mean
512.              desc_stat_mean.iloc[i, a]= round (np.average(b[b.columns[i]]),3)     #mean
513.              desc_stat_std.iloc[i, 0]= "Standard Deviation " + b.columns[i]       #std
514.              desc_stat_std.iloc[i, a]= round (np.std(b[b.columns[i]]),3)          #std
515.              desc_stat_min.iloc[i, 0]= "Minimum " + b.columns[i]                  #min
516.              desc_stat_min.iloc[i, a]= round (np.min(b[b.columns[i]]),3)          #min
517.              desc_stat_max.iloc[i, 0]= "Maximum " + b.columns[i]                  #max
518.              desc_stat_max.iloc[i, a]= round (np.max(b[b.columns[i]]),3)          #max
519.          else:
520.              desc_stat_mean.iloc[i, a]= round (np.average(b[b.columns[i]]),3)     #mean
521.              desc_stat_std.iloc[i, a]= round (np.std(b[b.columns[i]]),3)          #std
522.              desc_stat_min.iloc[i, a]= round (np.min(b[b.columns[i]]),3)          #min
523.              desc_stat_max.iloc[i, a]= round (np.max(b[b.columns[i]]),3)          #max
524.      a=a+1
525.
526.  desc_stat_list = [desc_stat_mean, desc_stat_std, desc_stat_min, desc_stat_max]
527.  desc_stat_final = pd.concat(desc_stat_list, keys=["Mean", "Std", "Min", "Max"])
528.  desc_stat_final
```

```
529.
530.  #### 2.3 - Export and save the data
531.
532.  #export this table to excel and save it
533.  desc_stat_final.to_excel("Desc_stats/AQM_Project_Aziz_Oeggerli_Schmid_descriptive_sta-
    tistics.xls")
534.
535.  ## 3. For each asset, run the stationarity test and decide which variable is stationary
536.
537.  ?adfuller
538.
539.  #### 3.1 - We check whether we have stationarity in the dependent variables (cryptos &
    fiat currencies --> y-values)
540.
541.  #run the Dickey-Fuller-Test to test the dependent variables for stationarity (in a
    first step for the actual values)
542.
543.  for lags in range(1,3):
544.      print('Number of lags used:',lags)
545.      print('ADF P-Val for Prices (Level)')
546.      for i in columns_dependent_variables:
547.          adf_library = adfuller(dependent_variables[i], maxlag=lags, regression='nc',au-
    tolag=None)
548.          adf_library_d = adfuller(np.diff(dependent_variables[i]), maxlag=lags, regres-
    sion='nc',autolag=None)
549.
550.          print(i,':',"%.2f" %  adf_library[1],)
551.      print('_____','\n')
552.
553.  #the result shows, that the p-value lies above the critical p-value (on all confidence
    levels - 10%, 5% & 1%)
554.  #therefore, we take the absolute returns (1st Difference) and run a second Dickey-
    Fuller-Test
555.  #this time the results lies under the critical p-value (on all confidence levels - 10%,
    5% & 1%)
556.  #meaning that we can reject the null hypothesis (H0) which means that there is no unit
    root (=stationarity)
557.
558.  for lags in range(1,3):
559.      print('Number of lags used:',lags)
560.      print('ADF P-Val for Absolute Returns (1st Difference)')
561.      for i in columns_dependent_variables:
562.          adf_library = adfuller(dependent_variables[i], maxlag=lags, regression='nc',au-
    tolag=None)
563.          adf_library_d = adfuller(np.diff(dependent_variables[i]), maxlag=lags, regres-
    sion='nc',autolag=None)
564.
565.          print(i,':',"%.2f" %  adf_library_d[1])
566.      print('_____','\n')
567.
568.  #### 3.2 - We check whether we have stationarity in the independent variables (commodi-
    ties & stock indices --> x-values)
569.
570.  #run the Dickey-Fuller-Test to test the independent variables for stationarity (in a
    first step for the actual values)
571.
572.  for lags in range(1,3):
573.      print('Number of lags used:',lags)
574.      print('ADF P-Val for Prices (Level)')
575.      for i in columns_independent_variables:
576.          adf_library = adfuller(independent_variables[i], maxlag=lags, regres-
    sion='nc',autolag=None)
577.          adf_library_d = adfuller(np.diff(independent_variables[i]), maxlag=lags, re-
    gression='nc',autolag=None)
578.
579.          print(i,':',"%.2f" %  adf_library[1],)
580.      print('_____','\n')
581.
582.  #the result shows, that the p-value lies above the critical p-value (on all confidence
    levels - 10%, 5% & 1%)
```

```
583.  #therefore, we take the absolute returns (1st Difference) and run a second Dickey-
    Fuller-Test
584.  #this time the results lies under the critical p-value (on all confidence levels - 10%,
    5% & 1%)
585.  #meaning that we can reject the null hypothesis (H0) which means that there is no unit
    root (=stationarity)
586.
587.
588.  for lags in range(1,3):
589.      print('Number of lags used:',lags)
590.      print('ADF P-Val for Absolute Returns (1st Difference)')
591.      for i in columns_independent_variables:
592.          adf_library = adfuller(independent_variables[i], maxlag=lags, regres-
    sion='nc',autolag=None)
593.          adf_library_d = adfuller(np.diff(independent_variables[i]), maxlag=lags, re-
    gression='nc',autolag=None)
594.
595.          print(i,':',"%.2f" %  adf_library_d[1])
596.      print('_____','\n')
597.
598.  ## 4. Conduct several OLS regressions with different combinations of independent varia-
    bles, choose the most promising model and verify whether the model and the betas are sta-
    ble over time
599.
600.  #### 4.1 - We generate a new table with daily returns
601.
602.  #generate a new dataframe which shows the daily returns of the assets (which makes the
    different data comparable)
603.  daily_returns = new_data_clean.pct_change()
604.  daily_returns.dropna(inplace=True)
605.
606.  #test for stationarity of the new data (daily returns)
607.  #the results shows, that there is no unit root (=stationarity)
608.
609.  for lags in range(1,3):
610.      print('Number of lags used:',lags)
611.      print('ADF P-Val for Prices (Level)')
612.      for i in columns:
613.          adf_library = adfuller(daily_returns[i], maxlag=lags, regression='nc',au-
    tolag=None)
614.          adf_library_d = adfuller(np.diff(daily_returns[i]), maxlag=lags, regres-
    sion='nc',autolag=None)
615.
616.          print(i,':',"%.2f" %  adf_library[1],)
617.      print('_____','\n')
618.
619.  #### 4.2 - Conduct a random/single OLS analysis and compare the prediction with the
    acutal value (not used later on, only for illustration purposes)
620.
621.
622.  #conduct a random/single OLS analysis
623.  #(not used, because we will later on create a loop to run all different combinations of
    x and y variables)
624.
625.  Y = daily_returns['Bitcoin']
626.  X = daily_returns[['Silver','S&P 500','MSCI World','MSCI Europe']]
627.  X = sm.add_constant(X)
628.  model = sm.OLS(Y,X)
629.  results = model.fit()
630.  results.summary()
631.
632.  #compare the prediction with the actual value (in a dataframe)
633.  regression1 = pd.DataFrame(Y)
634.  prediction = results.predict()
635.  regression1['Prediction']=prediction
636.  print(regression1.head())
637.
638.  #and in a chart
639.  plt.plot(regression1)
640.  labels=['Observed','Prediction']
```

```
641.  plt.legend(labels)
642.  plt.title('Observed vs. Predicted Inflation')
643.  plt.show
644.
645.  #### 4.3 - Test all different combinations of independent variables (x) for every de-
    pendent variable (y) and choose the most promissing regression
646.
647.  4.3.1 - We define some functions, which will be used for our analysis
648.
649.  #function for the chow test
650.
651.  def chow_test(Y,X,p_val=0.05):
652.
653.      # Model for the first half
654.      Y_1 = Y[:round(Y.shape[0]/2)]
655.      X_1 = X[:round(Y.shape[0]/2)]
656.
657.      # Model for the second half
658.      Y_2 = Y[round(Y.shape[0]/2):]
659.      X_2 = X[round(Y.shape[0]/2):]
660.
661.
662.      J = X.shape[1] #Amount of explanatory variables
663.      k = X_1.shape[1] #Amount of explanatory Variables
664.      N1 = X_1.shape[0] #length first period
665.      N2 = X_2.shape[0] #length second period
666.
667.      #Fit OLS Regression
668.      model_full = sm.OLS(Y,X).fit() #fit regression on the full period
669.      RSS = model_full.ssr #RSS full period
670.
671.      model_1 = sm.OLS(Y_1,X_1).fit() #fit regression
672.      RSS1 = model_1.ssr #RSS second period
673.      model_2 = sm.OLS(Y_2,X_2).fit() #fit regression
674.      RSS2 = model_2.ssr #RSS first period
675.
676.      chow = ((RSS-(RSS1+RSS2))/J)/((RSS1+RSS2)/(N1+N2-2*k))
677.      f_cdf = scipy.stats.f.cdf(chow, J, N1+N2-2*k)
678.      if (1-f_cdf) < p_val:
679.          print("P-Value for "+Y.name+" is " + str(round(1-f_cdf,2))+": Null Hypothesis
    rejected.\nThere is a structural break and therefore a unexpected/unconventional change
    over time in the parameters of the OLS regression model")
680.      else:
681.          print("P-Value for "+Y.name+" is " + str(round(1-f_cdf,2))+": Null Hypothesis
    cannot be rejected.\nThere is no structural break and therefore no unexpected/unconven-
    tional change over time in the parameters of the OLS regression model")
682.      return (f_cdf)
683.
684.
685.  #function for the rolling regression
686.
687.  def rolling_regression(y, x, window,method = "package"):
688.      #Check if NaN are available
689.      if y.isnull().values.any() == True or x.isnull().values.any()==True:
690.          print("There are some NaN in the data.\nPlease clean up the data.")
691.          return None
692.      #Confirmation that all time series have the same length
693.      if x.index.size > y.index.size:
694.          x = x[y.index]
695.          print("The length of the explanatory variables is larger than the length of the
    target variable")
696.      elif x.index.size < y.index.size:
697.          y = y[x.index]
698.          print("The length of the target variable is larger than the length of the ex-
    planatory variables")
699.      else:
700.          print("The length of the explanatory variables corresponds to the length of the
    target variable")
701.      #Check window-size
702.      if y.index.size < window:
```

```
703.          print("Window-size is larger than the length of the data => impossible - try
    smaller window-size")
704.          return None
705.      #calculate rolling regression - own calculation
706.      if method == "own":
707.          estimated_betas = pd.DataFrame(index = x.index, columns = [i for i in x.col-
    umns]) #empty DataFrame
708.          for i in range(window, x.index.size+1):
709.              X_slice = X.values[i-window:i,:] #values (length = window) => rolling
710.              y_slice = y.values[i-window:i] #values  (length = window) => rolling
711.              coeff = np.dot(np.dot(np.linalg.inv(np.dot(X_slice.T, X_slice)),
    X_slice.T), y_slice) #(XT*X)^-1*XT*Y #estimated Betas
712.              estimated_betas.iloc[i-1,:] = coeff
713.      else:
714.          #Package RollingOLS from statsmodels
715.          rolling_ols = RollingOLS(endog =y , exog=x,window = window)
716.          rolling_ols_results = rolling_ols.fit(cov_type = "HC0")
717.          estimated_betas = rolling_ols_results.params
718.          # === Assemble ========================================================
719.          #estimate = pd.Series(data=estimate_data, index=x.index[window-1:])
720.      return estimated_betas
721.
722.  #function which allows us to test all different combinations of independent variables
    (x) for every dependent variable (y)
723.
724.  def OLS_regression_all_combinations(y,x,features_list):
725.      rsquared_adj_list = [] #create empty list
726.      for i in features_list:
727.          X_features = x[i] #different conbination of explanatory variables
728.          model = sm.OLS(y,X_features) #model
729.          results = model.fit(cov_type = "HC0") #model fit
730.          rsquared_adj_list.append(results.rsquared_adj) #append adjusted r squared to
    list
731.      return(rsquared_adj_list)
732.
733.  4.3.2 - Create all necessary datasets for the OLS
734.
735.  #generate a new dataframe which shows the daily returns of all assets
736.  daily_returns = new_data_clean.pct_change()
737.  daily_returns.dropna(inplace=True)
738.
739.  #generate a new dataframe which shows the daily returns of the dependent variables
    (cryptos & other currencies)
740.  y_input = dependent_variables.pct_change()
741.  y_input = y_input.dropna()
742.  display(y_input.head())
743.
744.  #generate a new dataframe which shows the daily returns of the independent variables
    (commodities & stock indices)
745.  x_input = independent_variables.pct_change()
746.  x_input = sm.add_constant(x_input)
747.  x_input = x_input.dropna()
748.  display(x_input.head())
749.
750.  4.3.3 - Create a list of all possible combinations of dependent and independent varia-
    bles (y and x) if we use at least 5 different independent variables
751.
752.  #define the number of all possible combinations of x for each y (if we use at least 5
    independent variables)
753.
754.  features = x_input.columns.values
755.  features_list = []
756.  for i in range(5,len(features)+1):
757.      combinations = list(it.combinations(features,i)) #create all possible combinations
    with the length i => list of tuples
758.      combinations = [list(elem) for elem in combinations] #transform into list of list
759.      features_list += combinations
760.  features_list
761.  print('Number of independent variables is: ' + str(len(features)))
```

```
print('Number of different combinations of independent variables for each dependent variable is: ' + str(len(features_list)))

4.3.4 - Run all different combinations of independent variables (at least 5) for every dependent variable (y) and save the results (the adjusted r-squared value) into a txt-file

#this will take about 5-10min

for i in y_input.columns:
    Y = y_input[i] #get target variable
    print(i+"_rsquared_adj_list.txt") #print name of the txt file as confirmation
    rsquared_adj_list = OLS_regression_all_combinations(Y,x_input,features_list) #call function
    np.savetxt('Adj_r_squared/'+i+"_rsquared_adj_list.txt",rsquared_adj_list) #save adjusted r squared list as txt file

4.3.5 - Load the adj. r-squared values back into python, create a list out of it and combine the different lists into one

Bitcoin_r_squared_adj_list = list(np.loadtxt('Adj_r_squared/Bitcoin_rsquared_adj_list.txt'))
Ethereum_r_squared_adj_list = list(np.loadtxt('Adj_r_squared/Ethereum_rsquared_adj_list.txt'))
Ripple_r_squared_adj_list = list(np.loadtxt('Adj_r_squared/Ripple_rsquared_adj_list.txt'))
Litecoin_r_squared_adj_list = list(np.loadtxt('Adj_r_squared/Litecoin_rsquared_adj_list.txt'))
Bitcoin_Cash_r_squared_adj_list = list(np.loadtxt('Adj_r_squared/Bitcoin Cash_rsquared_adj_list.txt'))
EUR_r_squared_adj_list = list(np.loadtxt('Adj_r_squared/EUR_rsquared_adj_list.txt'))
GBP_r_squared_adj_list = list(np.loadtxt('Adj_r_squared/GBP_rsquared_adj_list.txt'))
JPY_r_squared_adj_list = list(np.loadtxt('Adj_r_squared/JPY_rsquared_adj_list.txt'))
CHF_r_squared_adj_list = list(np.loadtxt('Adj_r_squared/CHF_rsquared_adj_list.txt'))
CAD_r_squared_adj_list = list(np.loadtxt('Adj_r_squared/CAD_rsquared_adj_list.txt'))

#combine all lists into one list
r_squared_adjusted_list = [Bitcoin_r_squared_adj_list,
                           Ethereum_r_squared_adj_list,
                           Ripple_r_squared_adj_list,
                           Litecoin_r_squared_adj_list,
                           Bitcoin_Cash_r_squared_adj_list,
                           EUR_r_squared_adj_list,
                           GBP_r_squared_adj_list,
                           JPY_r_squared_adj_list,
                           CHF_r_squared_adj_list,
                           CAD_r_squared_adj_list]

4.3.6 - Create a dataframe of the adj. r-squared values and show the highest adj. r-squared value (which represents the best model)

#create a dataframe with all adj. r-squared values and show the highest value (=the best regression model) for each dependent variable

all_r_squared_adjusted = pd.DataFrame(r_squared_adjusted_list) #convert list to df
all_r_squared_adjusted = all_r_squared_adjusted.T #switch columns & rows (=transpose)
all_r_squared_adjusted.columns = columns_dependent_variables #switch columns
display(all_r_squared_adjusted.head())
all_r_squared_adjusted.max()

#Upload the data into out SQL table

#create a SQLite table
conn = sqlite3.connect('AQM_Project_Aziz_Oeggerli_Schmid.db')
c = conn.cursor()
c.execute('''CREATE TABLE adj_r_squared
             (RIC char(60),
              R_Squared char(60) NOT NULL)''')
print("Table created successfully")
```

```
819.
820.  #change the names of the columns back to the RIC's (so that the names are the same all
    over the database)
821.  all_r_squared_adjusted.columns = rics_dependent_variables
822.
823.
824.  #transformation into long format and new column names
825.  all_r_squared_adjusted_to_insert = pd.melt(all_r_squared_adjusted, value_vars =
    all_r_squared_adjusted.columns.values.tolist()[:-1])
826.  all_r_squared_adjusted_to_insert.columns = ['RIC','R_Squared']
827.
828.
829.  #transform the all_data_merged dataframe to a list
830.  #this needs to be done so that we can insert the data into the before created SQLite
    database
831.  all_r_squared_adjusted_to_insert_list = all_r_squared_adjusted_to_insert.val-
    ues.tolist()
832.
833.
834.  #upload the data to our SQLite database
835.  c = conn.cursor()
836.  c.executemany("INSERT INTO adj_r_squared(RIC, R_Squared) VALUES (?,?)",
    all_r_squared_adjusted_to_insert_list)
837.  conn.commit()
838.
839.  print("Upload to SQL-database successful")
840.
841.  #### 4.4 - For each regression run the Chow test in the middle of the sample and roll-
    ing window regressions to see whether model and the Betas are stable over time
842.
843.  #Loop to take combination of explanatory variables with the highest adjusted r-squared
844.  #run the rolling window regression and the chow test
845.  #plot and print the results and save the charts
846.
847.  for i in range(len(r_squared_adjusted_list)):
848.      print(i)
849.      print(y_input.columns[i])
850.      #Set the data
851.      Y = y_input[y_input.columns[i]] #target value => index
852.      r_squared_list_index = r_squared_adjusted_list[i] #list of list of adjusted r
    squared (per (crypto-)currency)
853.      index_max_r_squared_adj = r_squared_list_index.index(max(r_squared_list_index))
    #index of maximal adjusted r squared
854.      print("Maximal R Squared: "+ str(max(r_squared_list_index)))
855.
856.      features_best = features_list[index_max_r_squared_adj] #explanatory variables ac-
    cording to the adjusted r squared
857.      X_best = x_input[features_best] #DataFrame with the explanatory variables according
    to the adjusted r squared
858.      rolling_ols = rolling_regression(Y,X_best,60,method = "package") #run function for
    rolling window regression
859.      chowtest = chow_test(Y,X_best)
860.      print("_____")
861.
862.      fig, ax = plt.subplots(figsize=(18, 12)) #Plot
863.      ax.plot(rolling_ols)
864.      plt.title("Rolling Betas: " + Y.name, fontweight = "bold", fontsize = 20)
865.      ax.set(xlabel = "date", ylabel = "Beta Value")
866.      ax.set_xlabel('Date',fontsize = 18)
867.      ax.set_ylabel('Beta value',fontsize = 18)
868.      ax.tick_params(axis = "both", labelsize = 15)
869.      plt.legend(rolling_ols.columns, title = "Legend", fontsize = 15, title_fontsize =
    18, loc = "lower left")
870.      plt.savefig("Charts_RWR/RWR_"+Y.name+"_.png")
871.
872.  #since pandas.index.strftime() results in errors when plotting, we run the loop for the
    chow test and rolling window regression another time
873.  #and save the results in a newly created SQLite database
874.
875.
```

```
#create a SQLite table
conn = sqlite3.connect('AQM_Project_Aziz_Oeggerli_Schmid.db')
c = conn.cursor()
c.execute('''CREATE TABLE RWR_betas
             (Date TIMESTAMP,
             Var_dependent char(60),
             Var_independent char(60) NOT NULL,
             Beta REAL)''')
print("Table created successfully")


for i in range(len(r_squared_adjusted_list)):
    print(i)
    print(y_input.columns[i])

    #Set the data
    Y = y_input[y_input.columns[i]] #target value => index
    r_squared_list_index = r_squared_adjusted_list[i] #list of list of adjusted r squared (per index)
    index_max_r_squared_adj = r_squared_list_index.index(max(r_squared_list_index)) #index of max adj. r squared
    features_best = features_list[index_max_r_squared_adj] #explanatory variables according to the adj. r squared
    X_best = x_input[features_best] #DataFrame with the explanatory variables according to the adj. r squared
    rolling_ols = rolling_regression(Y,X_best,60,method = "package") #run function for rolling window regression


    #Transform Rolling OLS DataFrame into wide format and upload it into the SQLite database
    index = rolling_ols.index.strftime('%Y-%m-%d')
    rolling_ols['Date1'] = index #create new index
    rolling_ols['Name'] = y_input.columns.values.tolist()[i]
    rolling_ols_to_insert = pd.melt(rolling_ols, id_vars = ['Date1','Name'], value_vars = rolling_ols.columns.values.tolist()[:-2])
    rolling_ols_to_insert_list = rolling_ols_to_insert.values.tolist()


    #Upload data to SQL-database
    c.executemany('INSERT INTO RWR_betas(Date,Var_dependent,Var_independent,Beta) VALUES(?,?,?,?)',rolling_ols_to_insert_list)
    conn.commit()
    print("The Betas from the rolling window regression of the " + y_input.columns[i]+" were uploaded sucessfully into the SQLite database")
    print("_____________________________________________________")
```