Cryptocurrency Analysis

*By **Van Chien Dinh**, **Nguyen Dang Khanh** and **Marco Infante Serrano**.*

***NOTE**: The **Form of the project** is in a **Jupyter notebook** can be found in the GitHub repository and it is called "**Cryptocurrency Analysis.ipynb** (https://github.com/dinv00/-ECON2206---Cryptocurrency-Analysis/blob/main/Cryptocurrency%20Analysis.ipynb)"*

EXPLAINING THE PROYECT

Motivation

Our goal for the project was to analyze cryptocurrency prices in order to get a better understanding of different currencies and possibly gain some insight into their behavioral patterns. Another point to our motivation was to contribute knowledge about cryptocurrency trading and provide basis of information to investors and other stakeholders.

Question

The questions that we, ultimately, decided to focus on are as follow:

***QUESTION 1*: Which currencies are the most volatile? *QUESTION 2*: On which months are the currencies more volatile?**

Data sources

The dataset we used for this project is called "**Cryptocurrency prices for the year**" by the Kaggle user *ARTEM BURENOK*, it was uploaded the 6th of March of 2022.

This dataset contains multiple csv files relating to different cryptocurrencies. However, we will just use the following: Bitcoin *BTC*, Ethereum *ETH*, Bitcoin Cash *BCH*, Dai *DAI*, Doge Coin *Doge* and Ethereum Classic *ETC*.

The csv has the following columns: |Name |Meaning | |---------|----------| |Date |The corresponding day of the year. | |Open |The value in USD of the currency at the beginning of the day| |High |The highest value reached in a given day. | |Low |The lowest value reached in a given day. | |Close |The value in USD of the currency at the end of the day | |Adj. Close|The value in USD of the currency at the beginning of the day|

Some more columns will be added, however they will be detailed further in the document.

The dataset can be found by clicking here (https://www.kaggle.com/datasets/artemburenok/cryprocurrency-prices-for-the-year?select=BTC-USD.csv).

EXPLAINING THE CODE AND RESULTS

1 Data Retrieval

1.1 Scraping Methodology

We used the API of the Kaggle website, which allowed us to access the files (https://www.kaggle.com/datasets/artemburenok/cryprocurrency-prices-for-the-year?select=BTC-USD.csv).

```
import kaggle

# set the API credentials
kaggle.api.authenticate()
```

We downloaded the files using the authentication token.

```
# download the dataset
kaggle.api.dataset_download_files('artemburenok/cryprocurrency-prices-for-the-year', path='./data', unzip=True)
```

We designated a path to save the downloaded files and we specified which coins to download.

```
# Create a new directory to store the selected files
os.makedirs('./data_data', exist_ok=True)

# Define the list of files to select
selected_files = ['BTC-USD.csv', 'BCH-USD.csv', 'ETH-USD.csv', 'ETC-USD.csv', 'DOGE-USD.csv', 'SHIB-USD.csv', 'DAI-USD.csv', 'LTC-USI

# Copy the selected files to the new directory
for file_name in selected_files:
    src_path = f'./data/{file_name}'
    dst_path = f'./data_data/{file_name}'
    shutil.copy(src_path, dst_path)
```

1.2 Data Cleaning

1.2.1 Merging

In **1.1 Scraping Methodology** we explained that we downloaded 6 different csv files, each of these correspond to a different cryptocurrency (Bitcoin *BTC*, Ethereum *ETH*, Bitcoin Cash *BCH*, Dai *DAI*, Doge Coin *Doge* and Ethereum Classic *ETC*).

```
import pandas as pd
df1 = pd.read_csv('./data/BTC-USD.csv')
df2= pd.read_csv('./data/ETH-USD.csv')
df3= pd.read_csv('./data/BCH-USD.csv')
df4= pd.read_csv('./data/DAI-USD.csv')
df5= pd.read_csv('./data/DOGE-USD.csv')
df6= pd.read_csv('./data/ETC-USD.csv')

datasets = [df1,df2,df3,df4,df5,df6]
```

But each of these files is independent from one another, so as the first step of data cleaning we must merge them.

- 1: We defined a function called "**addName**" which receives the data frame called "*data*" and a string called "*name*" this function adds a new column to *data* in which it puts in the *name*...

  ```
  def addName(data,name):
      # add a new column named 'Currency'
      for index, row in df1.iterrows():
          data.at[index, 'Currency'] = name
  ```

  we did it for each *currency*.

  ```
  addName(df1,'BTC')
  addName(df2,'ETH')
  addName(df3,'BCH')
  addName(df4,'DAI')
  addName(df5,'DOGE')
  addName(df6,'ETC')
  ```

- 2: Once we have added the name of each currency we need to combine together all of these dataframes into one.

  ```
  merge = pd.concat(datasets)
  ```

1.2.2 Data adding

1.2.2.1 Difference

In order to be able to graph it is important to know the changes that each coin suffered each day, so we need to know the "**Difference**" between the value when it opened ("**Open**") from when it closed ("**Close**").

```
# Add new column that subtracts "Close" from "Open"
df.insert(7,column = "Difference", value=df["Open"]-df["Close"])
```

Then translate that as a percentage ("**PercentageChange**").

```
# Add new column that calculates percentage change of "Difference"
df.insert(7,column = "PercentageChange", value=(df['Difference'] / df['Close']) * 100)
```

1.2.2.2 Months

For some of the calculations and graphing it is essential to know the "**Month**", however this is not included in the original data, so we need to add a new column that contains the respective month. It is also important that we convert the "**Date**" column into *datetime* format.

```
# Convert "Date" column to datetime format
df['Date'] = pd.to_datetime(df['Date'])

# Extract month from "Date" column and add it as "Month" column
df['Month'] = df['Date'].dt.month
```

For some of the labels on the graphs it is clearer if we have the names of the months, so we also added a column that contains the months in string format ("**MonthStr**").

```
# Add a new column "MonthStr" with month names
df['MonthStr'] = df['Date'].dt.strftime('%B')
```

2 Data Visualization

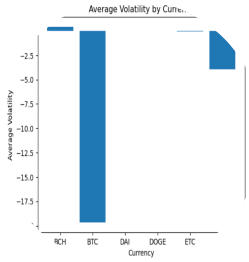
2.1 Bar Graphs

2.1.1 Per Currency

The purpose of this code is to visualize the "**Average volatility**" by "**Currency**" using a bar graph. The graph shows the average volatility on the y-axis and the months on the x-axis. This allows us to easily see the fluctuation in average volatility over the different months.

```
# Create the bar graph
plt.bar(avg_volatility.index, avg_volatility)
plt.title('Average Volatility by Currency')
plt.xlabel('Currency')
plt.ylabel('Average Volatility')

plt.show()
```



It is important to note that *Bitcoin* was highly volatile, so the other cryptocurrencies are not as visible in the previous graph, so we find the currency with the highest absolute value of average volatility. After that, we exclude that currency. Finally, we create a bar graph to visualize the average volatility by currency. The purpose of this code is to identify and visualize the average volatility for each currency, providing insights into the currency market's fluctuation and identifying any outliers in terms of volatility.

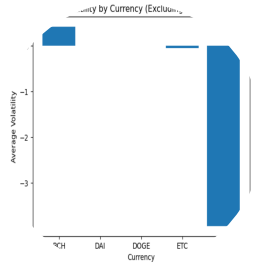
```
# Group the data by currency and calculate the average volatility
avg_volatility = df.groupby('Currency')['Volatility'].mean()

# Find the currency with the highest absolute value of average volatility
currency_max_volatility = avg_volatility.abs().idxmax()

# Exclude the currency with the highest absolute value of average volatility
filtered_avg_volatility = avg_volatility[avg_volatility.index != currency_max_volatility]

# Create the bar graph
plt.bar(filtered_avg_volatility.index, filtered_avg_volatility)
plt.title('Average Volatility by Currency (Excluding Highest Absolute)')
plt.xlabel('Currency')
plt.ylabel('Average Volatility')

plt.show()
```



2.1.2 Per Month

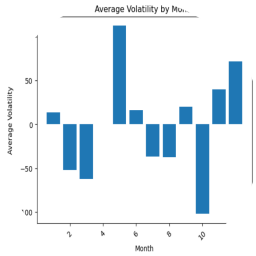
We group the data by "**Month**" and calculate the "**Average Volatility**" of each...

```
# Group the data by month and calculate the average volatility
monthly_avg_volatility = df.groupby('Month')['AverageVolatility'].mean()
```

To be able to create the graph.

```
# Create the bar graph
plt.bar(monthly_avg_volatility.index, monthly_avg_volatility)
plt.title('Average Volatility by Month')
plt.xlabel('Month')
plt.ylabel('Average Volatility')
plt.xticks(rotation=45)

plt.show()
```



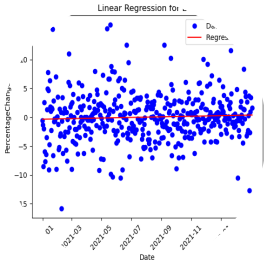
2.2 Line Graphs

2.2.1 Per Currency

This is done after "**3.1.2 Linear Regression**". By analyzing the regression lines for different currencies, we can understand how the date influences the percentage change for each currency.

```
# Plot the regression line and data points
plt.plot(pd.to_datetime(data['Date']), y, 'bo', label='Data Points')
plt.plot(pd.to_datetime(x_pred * 10**9), y_pred, 'r-', label='Regression Line')
plt.title(f'Linear Regression for {currency}')
plt.xlabel('Date')
plt.ylabel('PercentageChange')
plt.legend()
plt.xticks(rotation=45)

plt.show()
```



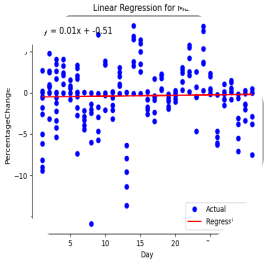
2.2.2 Per Month

This is done after "**3.2.2 Linear Regression**". By visualizing the trend and pattern in the data, it enables us to make predictions and draw conclusions based on the regression model. The equation of the regression line is displayed on the graph for each month, providing additional insights into the relationship between the variables.

```
# Plot the regression line
plt.plot(x_dates.dt.day, y, 'bo', label='Actual')
plt.plot(x_pred.day, y_pred, 'r-', label='Regression Line')
plt.title(f'Linear Regression for {month}')
plt.xlabel('Day')
plt.ylabel('PercentageChange')
plt.legend()

# Display the equation in the graph
plt.text(0.05, 0.95, equation, transform=plt.gca().transAxes, fontsize=12, verticalalignment='top')

plt.show()
```



3 Data Modeling

3.1 Per Month

3.1.1 Volatility

We created the column for average monthly **Volatility** for each **Currency**, by firstly calculating the **Daily volatility**, then we were able to group the average volatility by each month.

```
# Calculate the daily volatility
df['Volatility'] = df['Open'] - df['Close']

# Calculate the average volatility for each month
monthly_volatility = df.groupby('Month')['Volatility'].mean().reset_index()

# Rename the 'Volatility' column to 'AverageVolatility'
monthly_volatility.rename(columns={'Volatility': 'AverageVolatility'}, inplace=True)

# Merge the monthly volatility data with the original dataframe
merged_df = pd.merge(df, monthly_volatility, on='Month')
```

The results were as follows: |Month |Volatility | |-------- |----------| |January |13.586257 | |February |-51.706983 | |March |-62.307014 | |April |0.056703 | |May |112.268686 | |June |15.963668 | |July |-36.646614 | |August |-37.462263 | |September|19.932025 | |October |-102.045796| |November |39.619183 | |December |71.464429 |

With this information we are ready to tackle ***QUESTION 2*: On which months are the currencies more volatile?** as we can see *May* is the month with the highest absolute volatility as it had an average volatility of "112.6..." which was a positive change, on the other hand *October* was the month with the second highest absolute volatility however this value was negative, which means that in average on the days on *October* the currencies lost their value. Another notable month would be *April* on which we have the lowest absolute volatility.

3.1.2 Linear Regression

This part of the code performs *linear regression analysis* on a dataset, specifically focusing on the relationship between the "**Date**" and "**PercentageChange**" variables for each "**Month**". It groups the data by "**Month**"...

```
# Group the data by month
grouped = df.groupby('MonthStr')

#
regeq = {}
```

and calculates the regression line equation for each month. The code uses the regression model to predict values and plots the regression line alongside the actual data points.

```
# Perform linear regression for each month
for month, data in grouped:
    # Prepare the data
    x_dates = pd.to_datetime(data['Date'])
    y = data['PercentageChange']

    # Create a linear regression model
    model = LinearRegression()
    model.fit(x_dates.dt.day.values.reshape(-1, 1), y)

    # Predict the values based on the regression model
    x_pred = pd.date_range(start=x_dates.min(), end=x_dates.max(), freq='D')
    y_pred = model.predict(x_pred.day.values.reshape(-1, 1))

    # Get the regression line equation
    slope = model.coef_[0]
    intercept = model.intercept_
    equation = f'y = {slope:.2f}x + {intercept:.2f}'
    regeq[month]= equation
```

3.2 Per Currency

3.2.1 Volatility

We group the data by "**Currency**" and calculate the "**Average volatility**" for each "**Currency**".

```
# Group the data by currency and calculate the average volatility
avg_volatility = df.groupby('Currency')['Volatility'].mean()
```

The results are as follow: |Currency|Volatility| |--------|----------| |*BCH* |0.421422 | |*BTC* |-19.663061| |*DAI* |0.000010 | |*DOGE* |-0.000324 | |*ETC* |-0.040643 | |*ETH* |-3.940411 |

With this results we are able to answer ***QUESTION 1*: Which currencies are the most volatile?**.

Notable results are that:

- *BTC* or Bitcoin had a value of -19.663061 which means that in average in the timeframe studied this coin had a change of -19.663061 each day. Which makes it the most volatile of the currencies and a coin which overall has a tendency to lose value.
- *DOGE* or Doge coin had a value of 0.000010 which means that in average this coin almost had no changes in it's value.

3.2.2 Linear Regression

This code performs linear regression analysis for different currencies in a dataset. It groups the data by currency and applies linear regression to each group.

```
# Group the data by currency
grouped = df.groupby('Currency')
```

The purpose is to examine the relationship between the date and percentage change for each currency. The code prepares the data, creates a regression model, predicts values...

```
# Perform linear regression for each currency
for currency, data in grouped:
    # Prepare the data
    x_dates = pd.to_datetime(data['Date']).values.astype(float) / 10**9
    y = data['PercentageChange']

    # Create a linear regression model
    model = LinearRegression()
    model.fit(x_dates.reshape(-1, 1), y)

    # Predict the values based on the regression model
    x_pred = pd.date_range(start=pd.to_datetime(data['Date']).min(), end=pd.to_datetime(data['Date']).max(), freq='D').values.astype
    y_pred = model.predict(x_pred.reshape(-1, 1))
```

and plots the regression line along with the data points on a graph (as shown in "**2.2.2 Per Month**").