# Bitcoin’s Price History

By Group 5
Mike, William, Ryan

# Our Motivation

Our project focuses on a macro view of Bitcoin’s price history to uncover trends / patterns, and to better understand past & present valuation.

The questions we asked of the data were greatly influenced by Glassnode.com - They provide blockchain data, on-chain metrics, and a vast array of tools for any curious data explorers that may find conventional valuation metrics to be insufficient for analysing crypto markets.

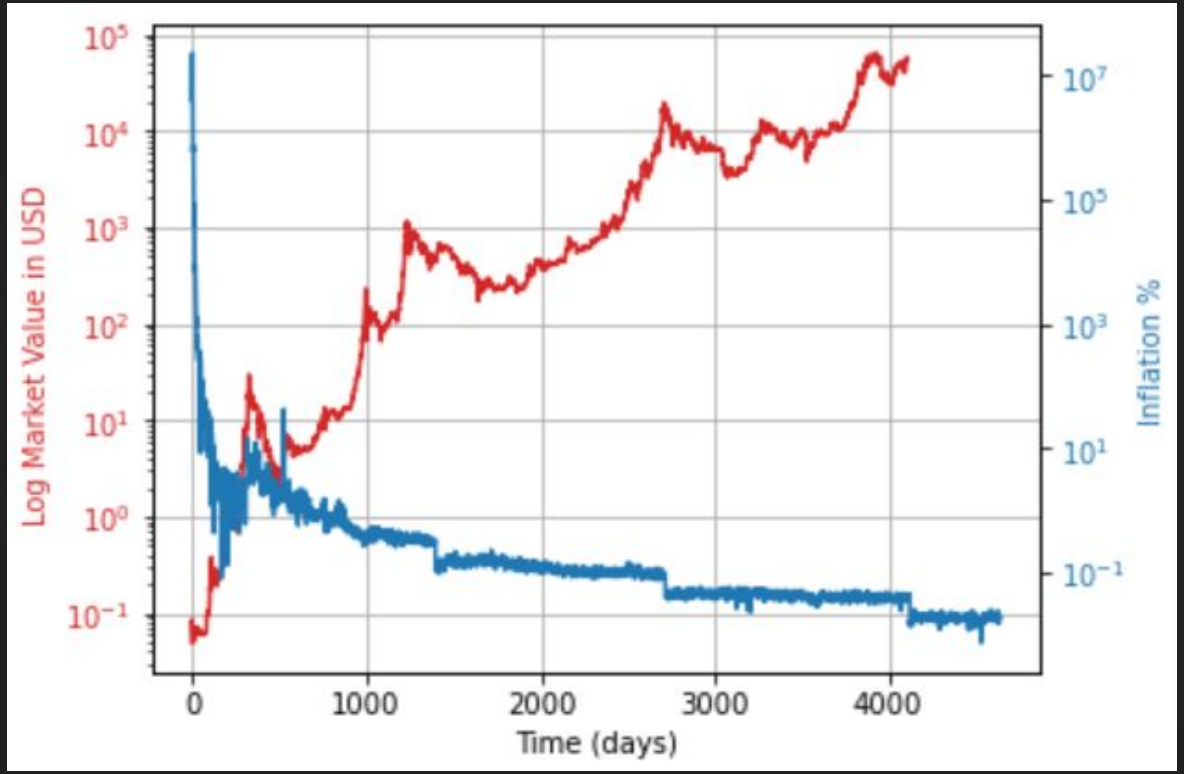
**How does Bitcoins inflation rate affect value?**
**What happens when inflation rate reaches 0?**

# Bitcoin Inflation Rate

Bitcoin has a fixed supply of 21 million coins that will ever be in existence. However only ~18.9 million have been mined so far. New bitcoins are created in every new block. Blocks are created every 10 minutes (on average), when a miner finds the hash that satisfies the PoW required for a valid block. The first transaction in each block, called the coinbase (not the exchange), contains the block reward for the miner that found the block. The block reward consists of the fees that people pay for transactions in that block and the newly created coins (called subsidy). The subsidy started at 50 bitcoins, and is halved every 210,000 blocks (about 4 years). That's why 'halvings' are very important for bitcoins money supply and stock-to-flow. Halvings also cause the supply growth rate (in bitcoin context usually called 'monetary inflation') to be stepped and not smooth.

## Inflation Rate = % of New Coins Issued / Current Supply

# Bitcoin Inflation Rate



# Bitcoin Inflation Rate Jupyter Notebook 1

```
#import dependences
import json
import requests
import pandas as pd
import hvplot.pandas
import matplotlib.pyplot as plt
import numpy as np
# insert your API key here
GLASSNODE_API_KEY = os.getenv('GLASSNODE_API_KEY')
# make API request
res = requests.get('https://api.glassnode.com/v1/metrics/supply/inflation_rate',
    params={'a': 'BTC','api_key': GLASSNODE_API_KEY})
res1 = requests.get('https://api.glassnode.com/v1/metrics/market/price_usd_close',
    params={'a': 'BTC','api_key': GLASSNODE_API_KEY})
#reading the data
price = pd.read_json(res1.text, convert_dates=['t'])
price.rename({'t' : 'Date', 'v': 'Price'}, axis=1, inplace=True)
price.set_index('Date', inplace=True)
inflation = pd.read_json(res.text, convert_dates=['t'])
inflation.rename({'t' : 'Date', 'v': 'Inflation Rate'}, axis=1, inplace=True)
inflation.set_index('Date', inplace=True)
#cleaning the data to concatenate data frames
price.reset_index(drop=True, inplace=True)
inflation.reset_index(drop=True, inplace=True)
btc = [price, inflation]
btc_df = pd.concat(btc, axis=1)
btc_df['Date'] = pd.date_range(start='2/1/2009', periods=len(btc_df), freq='D')
btc_df.set_index('Date', inplace=True)
btc_df
```

# Bitcoin Inflation Rate Jupyter Notebook 2

```
#plot the data
#a = btc_df.hvplot(logy=True, y=['Inflation Rate'], yaxis='left', ylabel='Inflation Rate %', xlabel='Year', title='Bitcoin Inflation Rate')
#b = btc_df.hvplot(logy=True, y=['Price'], yaxis='right', ylabel='Price', xlabel='Year', title='Bitcoin Inflation Rate')
#a * b
fig, ax1 = plt.subplots()
color = 'tab:red'
ax1.set_xlabel('Time (days)')
ax1.set_ylabel('Log Market Value in USD ', color=color)
ax1.set_yscale('log')
ax1.plot(price, color=color)
ax1.grid()
ax1.tick_params(axis='y', labelcolor=color)

ax2 = ax1.twinx()  # instantiate a second axes that shares the same x-axis

color = 'tab:blue'
ax2.set_ylabel('Inflation %', color=color)  # we already handled the x-label with ax1
ax2.set_yscale('log')
ax2.plot(inflation, color=color)
ax2.tick_params(axis='y', labelcolor=color)

fig.tight_layout()  # otherwise the right y-label is slightly clipped
plt.show()
```

# Does scarcity drive value?

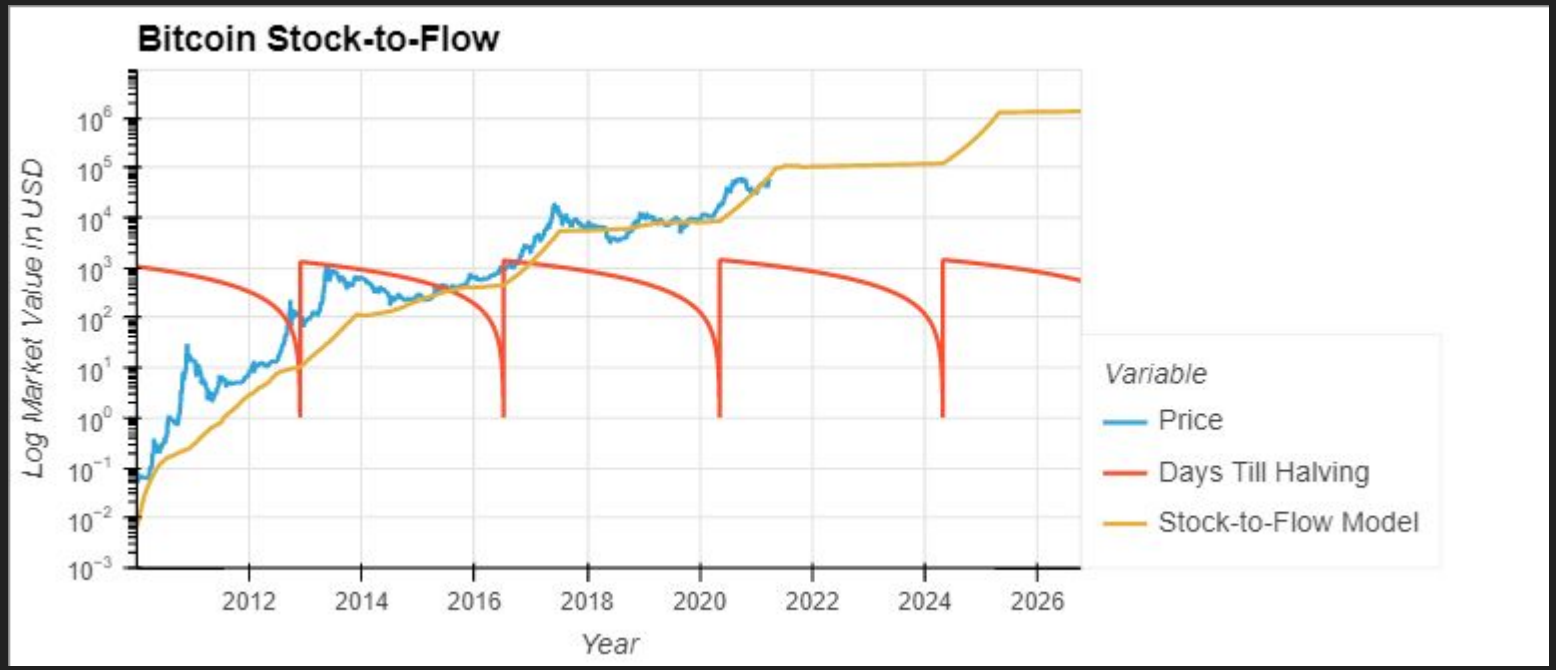
# Bitcoin Stock-to-Flow

The Stock to Flow (S/F) Ratio is a popular model that assumes that scarcity drives value. Stock to Flow is defined as the ratio of the current stock of a commodity (i.e. circulating Bitcoin supply) and the flow of new production (i.e. newly mined bitcoins). Bitcoin's price has historically followed the S/F Ratio and therefore it is a model that can be used to predict future Bitcoin valuations.

$$SF = Stock / Flow$$

$$\text{Stock-to-Flow Model: BTC Price} = \exp(-1.84) \cdot SF \text{ ^ } 3.36$$

# Bitcoin Stock-to-Flow



# Bitcoin Stock-to-Flow Jupyter Notebook 1

```python
#import dependences
import json
import requests
import pandas as pd
import hvplot.pandas
# insert your API key here
GLASSNODE_API_KEY = os.getenv('GLASSNODE_API_KEY')
# make API request
res = requests.get('https://api.glassnode.com/v1/metrics/indicators/stock_to_flow_ratio',
    params={'a': 'BTC','api_key': GLASSNODE_API_KEY})
res1 = requests.get('https://api.glassnode.com/v1/metrics/market/price_usd_close',
    params={'a': 'BTC','api_key': GLASSNODE_API_KEY})
# convert to pandas dataframe
price = pd.read_json(res1.text, convert_dates=['t'])
price.rename({'t' : 'Date', 'v': 'Price'}, axis=1, inplace=True)
price.set_index('Date', inplace=True)
#read and loop json btc data to build separte lists
df = pd.read_json(res.text, convert_dates=['t'])
s2f = df['o'].to_list()

lst = []
days_til_halving = []
ratio = []

for i in range(len(s2f)):
    for k in s2f[i]:
        lst.append(s2f[i][k])

lst2 = iter(lst)
for x,y in zip(lst2,lst2):
    days_til_halving.append(x)
    ratio.append(y)
```

```
[26]: s2f

[26]: [{'daysTillHalving': 1061, 'ratio': 0.006670564545956},
       {'daysTillHalving': 1060, 'ratio': 0.006871940699592001},
       {'daysTillHalving': 1059, 'ratio': 0.007069040973185001},
       {'daysTillHalving': 1058, 'ratio': 0.007286633171392},
       {'daysTillHalving': 1057, 'ratio': 0.0074672716921 7},
       {'daysTillHalving': 1056, 'ratio': 0.007639155422806},
       {'daysTillHalving': 1055, 'ratio': 0.007811067170201 0005},
       {'daysTillHalving': 1054, 'ratio': 0.007992897580761},
       {'daysTillHalving': 1053, 'ratio': 0.008259015060790001},
```

```
]: lst

]: [1061,
    0.006670564545956,
    1060,
    0.006871940699592001,
    1059,
    0.007069040973185001,
    1058,
    0.007286633171392,
    1057,
    0.00746727169217,
    1056,
    0.007639155422806,
```

# Bitcoin Stock-to-Flow Jupyter Notebook 2

```
    # convert stock to flow Data into a Dataframe
stock_to_flow = pd.read_json(res.text, convert_dates=['t'])
stock_to_flow.rename({'t': 'Date'}, axis=1, inplace=True)
stock_to_flow = stock_to_flow.set_index('Date', drop=True)
#stock_to_flow = pd.DateFrame(days_til_halving, columns=['days_til_halving'])


# Ratio Dataframe
#stock_to_flow['Days Till Halving']=days_til_halving
stock_to_flow['Stock-to-Flow Model']=ratio
stock_to_flow.drop(columns=['o'], inplace=True)

#halving Dataframe
halving = pd.read_json(res.text, convert_dates=['t'])
halving.rename({'t': 'Date'}, axis=1, inplace=True)
halving = halving.set_index('Date', drop=True)
halving['Days Till Halving']=days_til_halving
halving.drop(columns=['o'], inplace=True)

#reset index's to prepare data to concat
price.reset_index(drop=True, inplace=True)
halving.reset_index(drop=True, inplace=True)
stock_to_flow.reset_index(drop=True, inplace=True)

#concatenate dataframes
btc = [stock_to_flow, halving, price]
btc_df = pd.concat(btc, axis=1)
btc_df['Date'] = pd.date_range(start='1/2/2010', periods=len(df), freq='D')
btc_df.set_index('Date', inplace=True)
btc_df

#plot data
btc_df.hvplot(logy=True, y=['Price', 'Days Till Halving', 'Stock-to-Flow Model'], ylabel='Log Market Value in USD', xlabel='Year', title='Bitcoin Stock-to-Flow', shared_axes=False, grid=True)
```

# When is the best time to enter & exit the market?

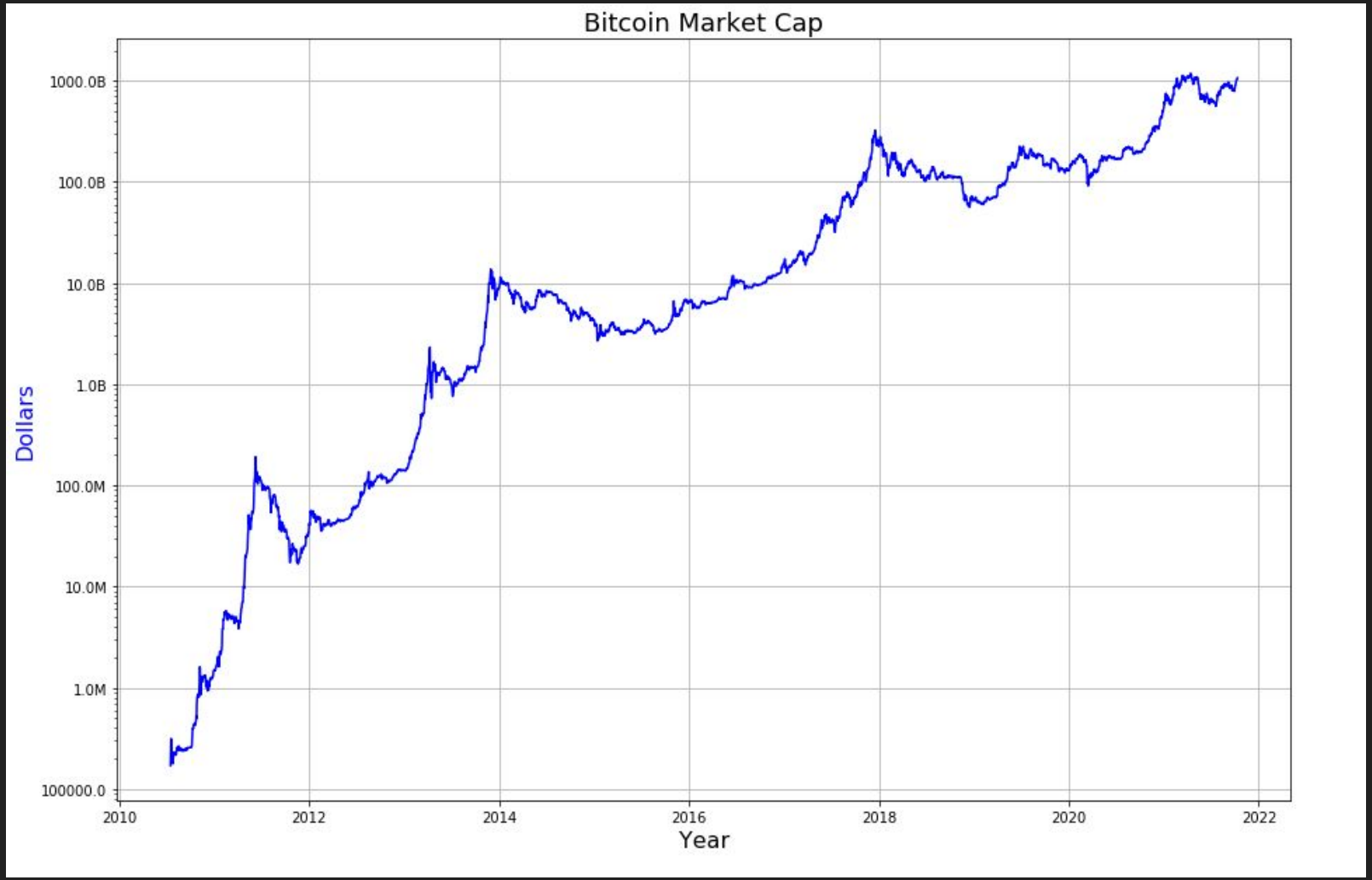
# MVRV

- Market-Value-to-Realized-Value = Market Cap / Realized Cap
- By comparing these two metrics, MVRV can be used to get a sense of when price is above or below "fair value"
- But first let’s define Market Cap & Realized Cap!

# Market Cap

- Market Capitalization is defined as the total network value of an asset
- Traditionally can be calculated as:

  Market Cap = Share Price * Shares Outstanding
- Calculated by multiplying the last traded price of the BTC/USD pair by the number of Bitcoin mined so far ( ~18,840,240 as of October 2021)


Bitcoin Market Cap
1000.0B
100.0B
10.0B
1.0B
100.0M
10.0M
1.0M
100000.0
Dollars
2010
2012
2014
2016
2018
2020
2022
Year

# Market Cap Notebook

[1]:
```
import json
import requests
import pandas as pd
import os
from dotenv import load_dotenv
import matplotlib.pyplot as plt
import datetime
import numpy as np
import matplotlib.ticker as tick

# insert your API key here
GLASSNODE_API_KEY = os.getenv('GLASSNODE_API_KEY')

# make API request
MVRV_data = requests.get('https://api.glassnode.com/v1/metrics/market/mvrv',
    params={'a': 'BTC', 'api_key': GLASSNODE_API_KEY})
Price_data = requests.get('https://api.glassnode.com/v1/metrics/market/price_usd_close',
    params={'a': 'BTC', 'api_key': GLASSNODE_API_KEY})
Realized_Cap_data = requests.get('https://api.glassnode.com/v1/metrics/market/marketcap_realized_usd',
    params={'a': 'BTC', 'api_key': GLASSNODE_API_KEY})
Market_Cap_data = requests.get('https://api.glassnode.com/v1/metrics/market/marketcap_usd',
    params={'a': 'BTC', 'api_key': GLASSNODE_API_KEY})
MVRV_Z_data = requests.get('https://api.glassnode.com/v1/metrics/market/mvrv_z_score',
    params={'a': 'BTC', 'api_key': GLASSNODE_API_KEY})


# convert to pandas dataframe
MVRV_df = pd.read_json(MVRV_data.text, convert_dates=['t'])
Price_df = pd.read_json(Price_data.text, convert_dates=['t'])
Realized_Cap_df = pd.read_json(Realized_Cap_data.text, convert_dates=['t'])
Market_Cap_df = pd.read_json(Market_Cap_data.text, convert_dates=['t'])
MVRV_Z_df = pd.read_json(MVRV_Z_data.text, convert_dates=['t'])
```

[2]:
```
Market_Cap_df.rename(columns={'t':'Date', 'v':'Market_Cap'}, inplace=True)
```

[3]:
```
Market_Cap_df = Market_Cap_df.set_index('Date')
```

[4]:
```
Market_Cap_df
```

[4]:

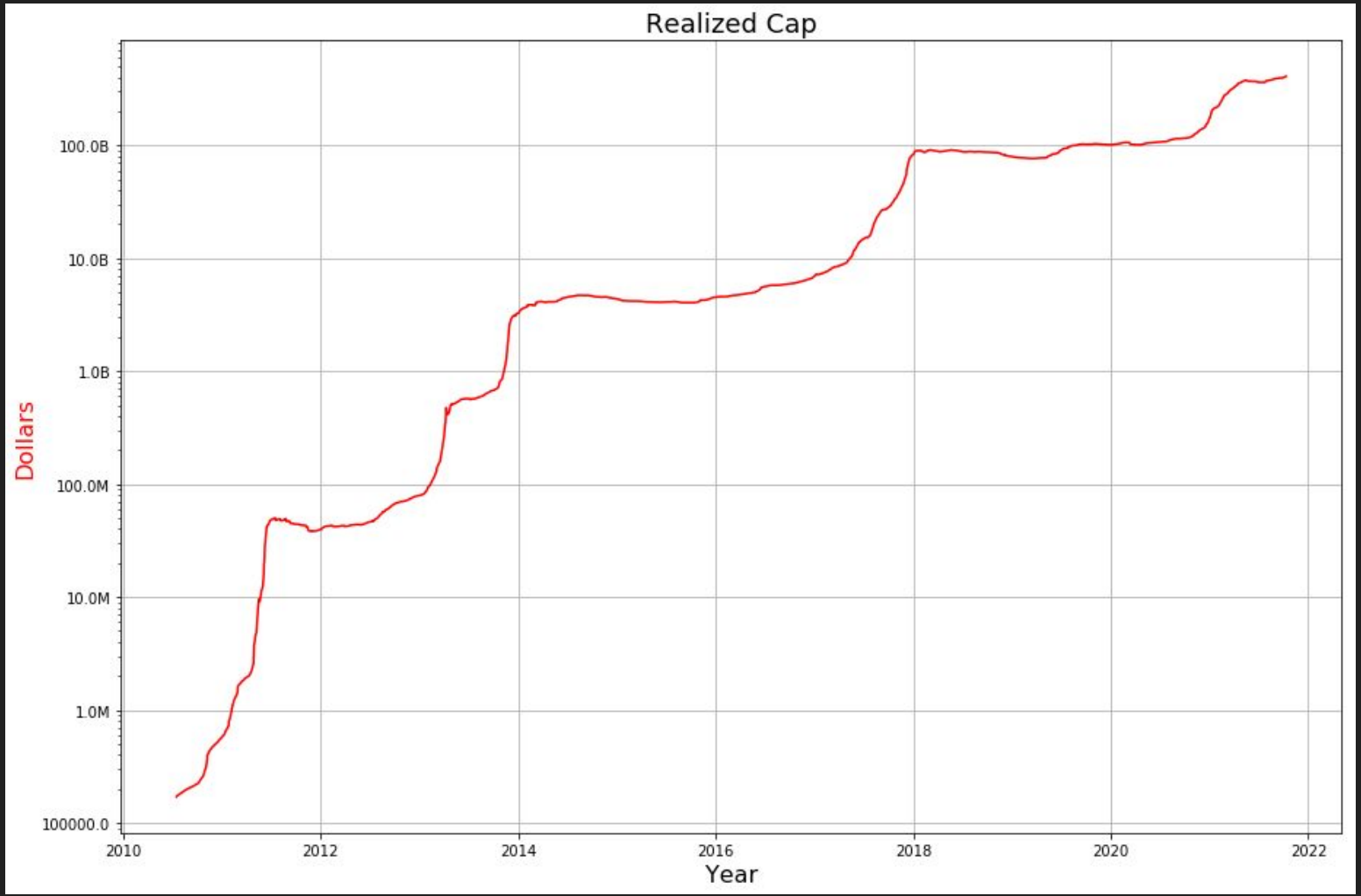
| Date | Market_Cap |
|---|---|
| 2010-07-17 | 1.702649e+05 |
| 2010-07-18 | 2.225672e+05 |
| 2010-07-19 | 3.138020e+05 |
| 2010-07-20 | 2.757412e+05 |
| 2010-07-21 | 2.444626e+05 |
| ... | ... |
| 2021-10-07 | 1.024017e+12 |
| 2021-10-08 | 1.025949e+12 |
| 2021-10-09 | 1.031192e+12 |
| 2021-10-10 | 1.039171e+12 |
| 2021-10-11 | 1.068554e+12 |

4105 rows × 1 columns

[15]:
```
# create figure and axis objects with subplots()
fig,ax = plt.subplots(figsize=(15,10))
# make a plot
ax.plot(Market_Cap_df.index, Market_Cap_df.Market_Cap, color="blue")
# set x-axis label
ax.set_xlabel("Year",fontsize=16)
# set y-axis label and scale
ax.set_ylabel("Dollars",fontsize=16, color='blue')
ax.set_yscale('log')
# title
ax.set_title('Bitcoin Market Cap', fontsize=18)
ax.grid(True)

# define function to make y-values more readable, turning large tick values (in the billions, millions) such as 450,000,000,000 into 450.0B
def reformat_large_tick_values(tick_val, pos):

    if tick_val >= 1000000000:
        val = round(tick_val/1000000000, 1)
        new_tick_format = '{:}B'.format(val)
    elif tick_val >= 1000000:
        val = round(tick_val/1000000, 1)
        new_tick_format = '{:}M'.format(val)
    else:
        new_tick_format = tick_val

    # make new_tick_format into a string value
    new_tick_format = str(new_tick_format)


    return new_tick_format

# format ax3 y-labels and positioning
ax.formatter = tick.FormatStrFormatter('$%1.2f')
ax.yaxis.set_major_formatter(tick.FuncFormatter(reformat_large_tick_values))


# plot
plt.show()
```

# Realized Cap

- Invented by Nic Carter and Antoine Le Calvez in 2018
- Realized Capitalization is a variation of Market Cap that values each UTXO (Unspent Transaction Output) at the price when it was last moved - NOT the current price of Bitcoin
- Reduces the impact of lost and long dormant coins
- Weighs coins according to their actual presence in the economy of Bitcoin


Realized Cap
Dollars
100.0B
10.0B
1.0B
100.0M
10.0M
1.0M
100000.0
2010
2012
2014
2016
2018
2020
2022
Year

# Realized Cap Notebook

[14]:
```
# create figure and axis objects with subplots()
fig,ax = plt.subplots(figsize=(15,10))
# make a plot
ax.plot(combined_df.index, combined_df.Realized_Cap, color="red")
# set x-axis label
ax.set_xlabel("Year",fontsize=16)
# set y-axis label and scale
ax.set_ylabel("Dollars",fontsize=16, color='red')
ax.set_yscale('log')
# title
ax.set_title('Realized Cap', fontsize=18)
ax.grid(True)

# define function to make y-values more readable
def reformat_large_tick_values(tick_val, pos):

    if tick_val >= 1000000000:
        val = round(tick_val/1000000000, 1)
        new_tick_format = '{:}B'.format(val)
    elif tick_val >= 1000000:
        val = round(tick_val/1000000, 1)
        new_tick_format = '{:}M'.format(val)
    else:
        new_tick_format = tick_val

    # make new_tick_format into a string value
    new_tick_format = str(new_tick_format)


    return new_tick_format

# format ax3 y-labels and positioning
ax.formatter = tick.FormatStrFormatter('$%1.2f')
ax.yaxis.set_major_formatter(tick.FuncFormatter(reformat_large_tick_values))


# plot
plt.show()
```

[9]:
```
Realized_Cap_df.rename(columns={'t':'Date', 'v':'Realized_Cap'}, inplace=True)
```

[10]:
```
Realized_Cap_df
```

[10]:

| | Date | Realized_Cap |
|---|---|---|
| 0 | 2009-01-03 | 2.475500e+00 |
| 1 | 2009-01-04 | NaN |
| 2 | 2009-01-05 | NaN |
| 3 | 2009-01-06 | NaN |
| 4 | 2009-01-07 | NaN |
| ... | ... | ... |
| 4660 | 2021-10-07 | 4.065050e+11 |
| 4661 | 2021-10-08 | 4.073032e+11 |
| 4662 | 2021-10-09 | 4.078484e+11 |
| 4663 | 2021-10-10 | 4.085543e+11 |
| 4664 | 2021-10-11 | 4.097016e+11 |

4665 rows × 2 columns

# Back to MVRV

Some things to keep in mind:

- Best when applied to multi-year analysis
- As market cap decreases in volatility, the upper threshold of 3.7 might not prove as reliable
- Realized cap may drop given a black-swan shock event
- It is of particular interest whenever market value goes below a 1:1 ratio to realized value - these periods account for both undervaluation and the capitulation stages of market psychology.


Market Value / Realized Value
Ratio
Year
7
6
5
4
3
2
1
2010
2012
2014
2016
2018
2020
2022

# MVRV Notebook

```
[5]: MVRV_df.rename(columns={'t':'Date', 'v':'MVRV'}, inplace=True)
```

```
[6]: MVRV_df
```

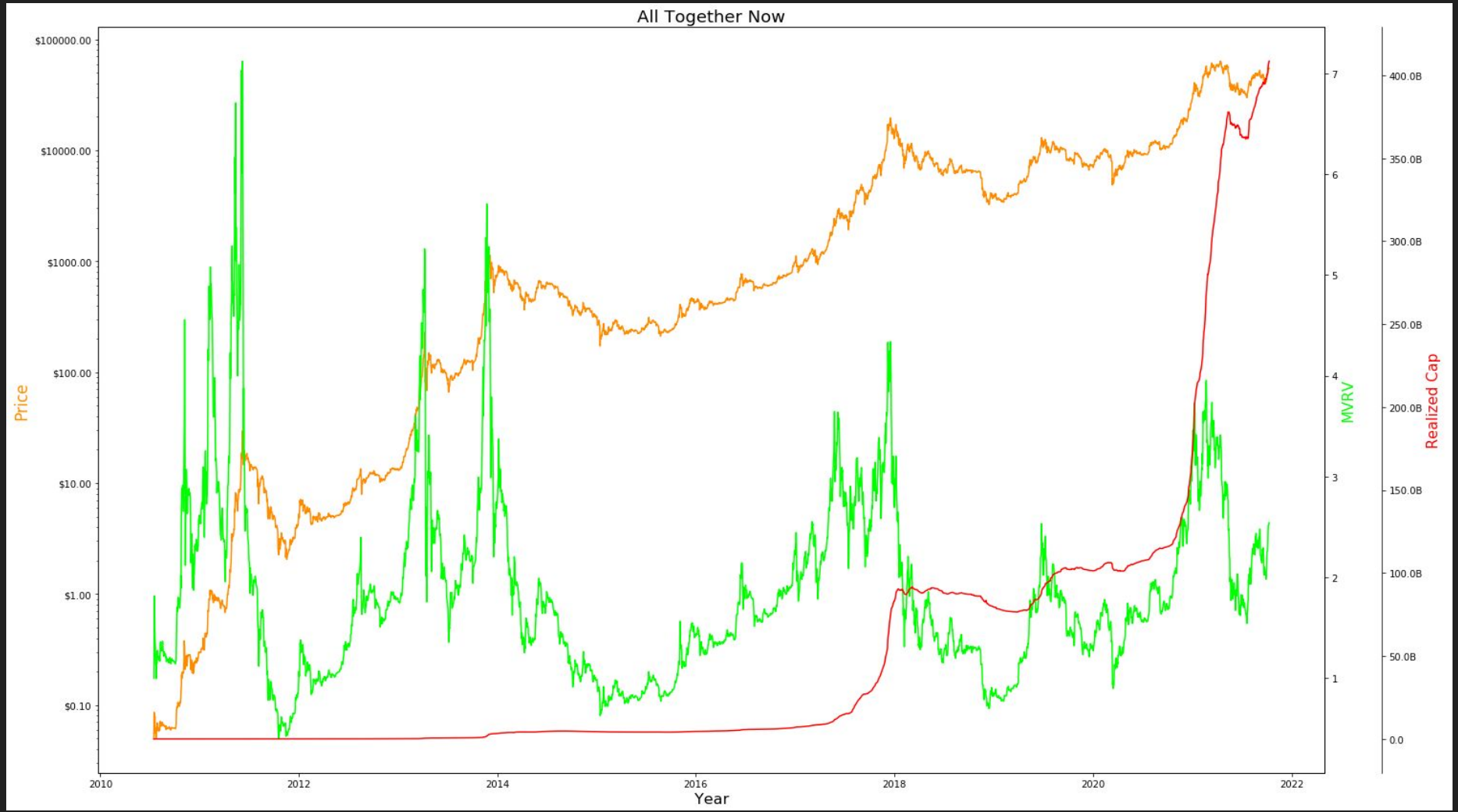
[6]:

| | Date | MVRV |
|---|---|---|
| 0 | 2010-07-17 | 1.000000 |
| 1 | 2010-07-18 | 1.300600 |
| 2 | 2010-07-19 | 1.814750 |
| 3 | 2010-07-20 | 1.586253 |
| 4 | 2010-07-21 | 1.399411 |
| ... | ... | ... |
| 4100 | 2021-10-07 | 2.519077 |
| 4101 | 2021-10-08 | 2.518883 |
| 4102 | 2021-10-09 | 2.528370 |
| 4103 | 2021-10-10 | 2.543532 |
| 4104 | 2021-10-11 | 2.608126 |

4105 rows × 2 columns

```
[16]: # create figure and axis objects with subplots()
      fig,ax = plt.subplots(figsize=(15,10))

      # make a plot
      ax.plot(combined_df.index, combined_df.MVRV, color="lime")

      # set x-axis label
      ax.set_xlabel("Year",fontsize=16)
      # set y-axis label and scale
      ax.set_ylabel("Ratio",fontsize=16, color='lime')


      ax.set_title('Market Value / Realized Value', fontsize=18)
      ax.grid(True)

      l1=ax.axhline(1,color='black',ls='--')
      l2=ax.axhline(3.7,color='black',ls='--')


      # plot
      plt.show()
```

# And all together now!


All Together Now
Price
$100000.00
$10000.00
$1000.00
$100.00
$10.00
$1.00
$0.10
MVRV
7
6
5
4
3
2
1
Realized Cap
400.0B
350.0B
300.0B
250.0B
200.0B
150.0B
100.0B
50.0B
0.0
2010
2012
2014
2016
2018
2020
2022
Year

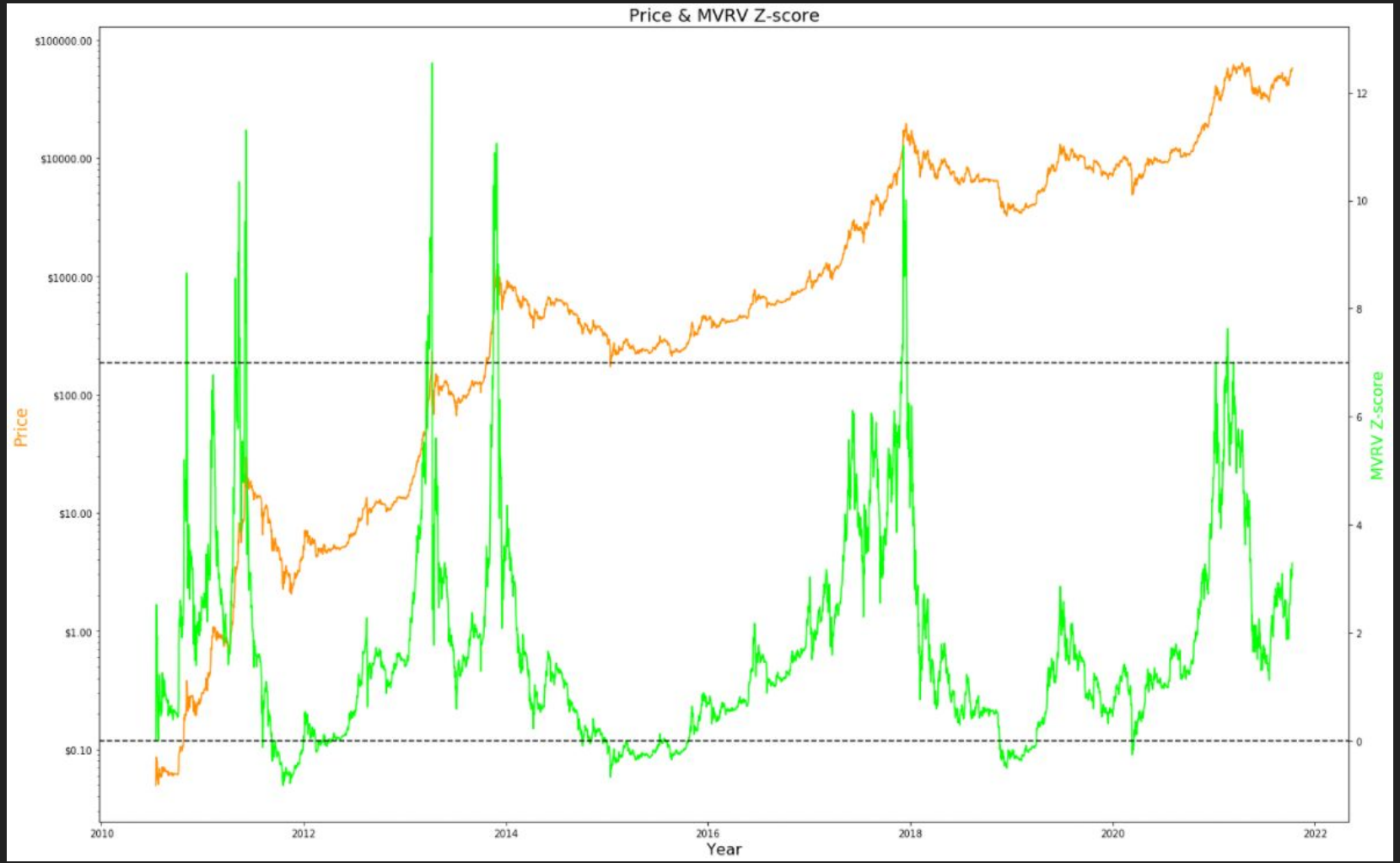
# All Together Notebook

[21]:
```
# create figure and axis objects with subplots()
fig,ax = plt.subplots(figsize=(23,15))
# make a plot
ax.plot(combined_df.index, combined_df.Price, color="darkorange")
# set x-axis label
ax.set_xlabel("Year",fontsize=16)
# set y-axis label and scale
ax.set_ylabel("Price",fontsize=17, color='darkorange')
ax.set_yscale('log')
# title
ax.set_title('All Together Now', fontsize=18)
# format price axis to dollars
formatter = tick.FormatStrFormatter('$%1.2f')
ax.yaxis.set_major_formatter(formatter)


# twin object for two different y-axis on the sample plot
ax2=ax.twinx()
# make a plot with different y-axis using second axis object
ax2.plot(combined_df.index, combined_df.MVRV, color="lime")
ax2.set_ylabel("MVRV",color="lime",fontsize=16)

# third axis
ax3=ax.twinx()
ax3.plot(combined_df.index, combined_df.Realized_Cap, color='red')
ax3.set_ylabel('Realized Cap', color='red', fontsize=16)

# define function to make ax3 y-values more readable
def reformat_large_tick_values(tick_val, pos):

    if tick_val >= 1000000000:
        val = round(tick_val/1000000000, 1)
        new_tick_format = '{:}B'.format(val)
    elif tick_val >= 1000000:
        val = round(tick_val/1000000, 1)
        new_tick_format = '{:}M'.format(val)
    else:
        new_tick_format = tick_val

    # make new_tick_format into a string value
    new_tick_format = str(new_tick_format)


    return new_tick_format

# format ax3 y-labels and positioning
ax3.formatter = tick.FormatStrFormatter('$%1.2f')
ax3.yaxis.set_major_formatter(tick.FuncFormatter(reformat_large_tick_values))
ax3.spines['right'].set_position(('outward', 60))


# plot
plt.show()
```

**Can we use a standard deviation metric to even better identify extreme periods of over / undervaluation?**

# MVRV Z-Score

- MVRV Z-Score =  (Market Cap - Realized Cap) / **STD Dev**(Market Cap)
- Used to identify periods when Bitcoin is extremely over or undervalued relative to its “fair value”
- Entering long positions when the ratio is near Zero or below, has historically produced outsized returns
- Particularly effective at identifying cycle tops - when market value is unusually high above realized value


Price & MVRV Z-score
$100000.00
$10000.00
$1000.00
$100.00
$10.00
$1.00
$0.10
Price
12
10
8
6
4
2
0
MVRV Z-score
2010
2012
2014
2016
2018
2020
2022
Year

# NUPL- Net Unrealized Profit/Loss

## Interpretation

NUPL was created in order to determine the total amount of profits or losses from the circulating supply of a coin.

NUPL is calculated as the difference between market cap and realized cap divided by the market cap. If market cap is greater than realized cap, then NUPL>0, which means Bitcoin on-chain expected value is less than what they actually have. This value indicates the increase of selling pressure.

- Net Unrealized Profit (NUP): Sum of UTXO being in profit with the price difference between created and destroyed.
- Net Unrealized Loss (NUL): Sum of UTXO being in loss with the price difference between created and destroyed.
- Net Unrealized Profit and Loss (NUPL): The difference between market value and realized value.

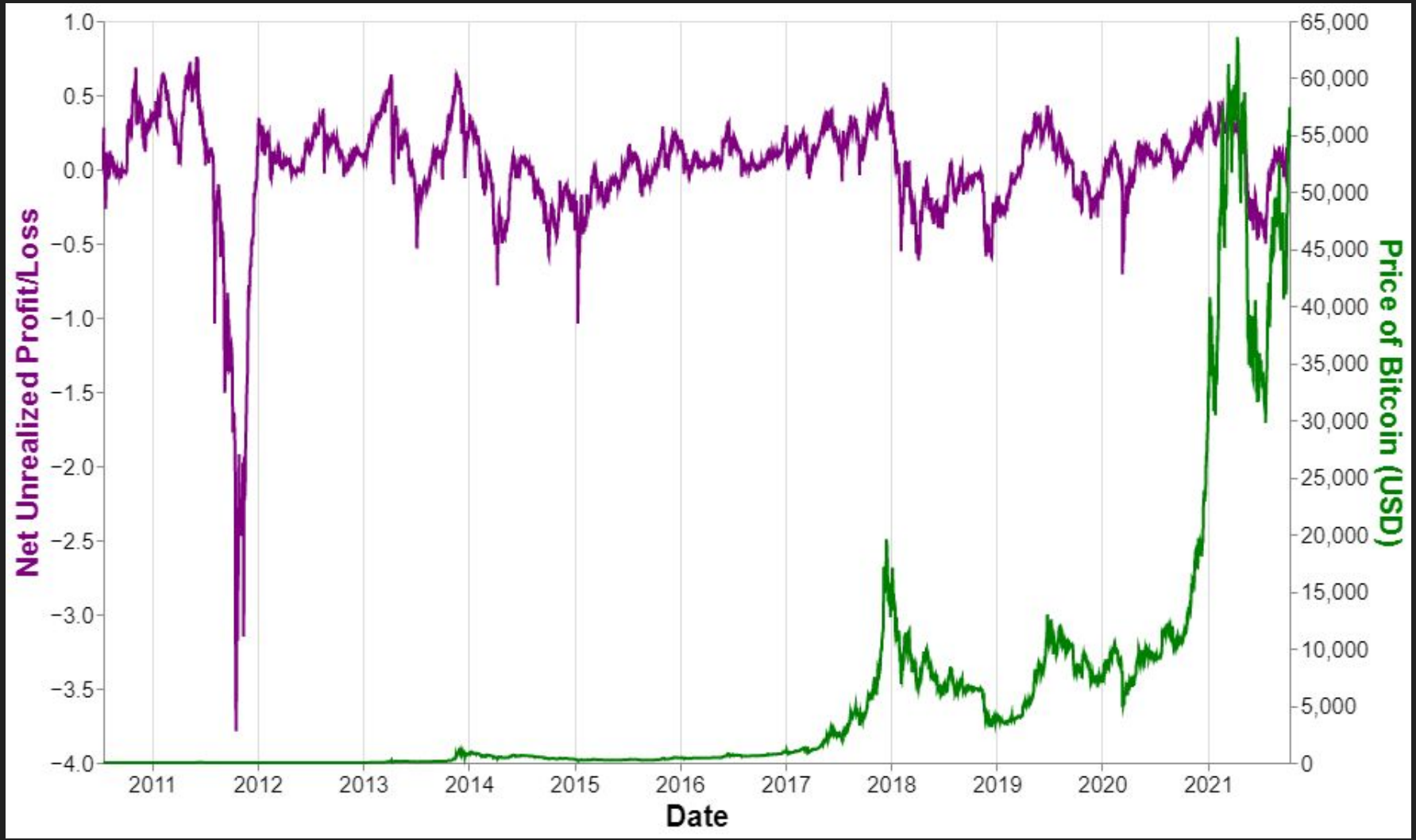
A UTXO is the amount of digital currency remaining after a cryptocurrency transaction is executed.

UTXO transactions sound complicated, but they really are fairly simple. UTXO or unspent transaction outputs are used in cryptocurrency transactions. These are the transactions that are left unspent after someone completes a transaction, similar to the change someone receives after conducting a cash transaction at the store.

# Why is NUPL a useful metric?

Whenever you look at this metric and you see it at a certain point you can determine whether it’s a good time to buy/sell/stay your position on your Bitcoin. This metric can be a very useful tool to better your investment strategy with Bitcoin to make you more profitable.

- Under 0% is the capitulation phase which is the best time to buy Bitcoin
- Between 0-25% is the hope/fear stage
- Between 25-50% is the optimism/anxiety stage
- Between 50-75% is the belief/denial stage
- Once above 75% is the euphoria/greed stage and the historical data shows that Bitcoin is at a market top. Best time to sell


Net Unrealized Profit/Loss
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
−3.0
−3.5
−4.0
Price of Bitcoin (USD)
65,000
60,000
55,000
50,000
45,000
40,000
35,000
30,000
25,000
20,000
15,000
10,000
5,000
0
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
Date

bitcoin.ipynb

Code

Python 3 (ipykernel)

[2]:
```
import json
import requests
import pandas as pd
import os
from dotenv import load_dotenv
import hvplot.pandas
import altair as alt

# insert your API key here
GLASSNODE_API_KEY = 'glassnode_api'

# make API request
res = requests.get('https://api.glassnode.com/v1/metrics/indicators/nupl_less_155',
    params={'a': 'BTC', 'api_key': GLASSNODE_API_KEY})
res1 = requests.get('https://api.glassnode.com/v1/metrics/market/price_usd_close',
    params={'a': 'BTC', 'api_key': GLASSNODE_API_KEY})

# convert to pandas dataframe
nupl = pd.read_json(res.text, convert_dates=['t'])
price = pd.read_json(res1.text, convert_dates=['t'])

# renaming my columns
nupl.rename({'t' : 'Date', 'v' : 'NUPL'}, axis=1, inplace=True)
price.rename({'t' : 'Date', 'v' : 'Price'}, axis=1, inplace=True)
price
```

[2]:

| | Date | Price |
|---|---|---|
| 0 | 2010-07-17 | 0.049510 |
| 1 | 2010-07-18 | 0.085840 |
| 2 | 2010-07-19 | 0.080800 |
| 3 | 2010-07-20 | 0.074733 |
| 4 | 2010-07-21 | 0.079210 |
| ... | ... | ... |
| 4101 | 2021-10-08 | 53884.290533 |
| 4102 | 2021-10-09 | 55068.643546 |
| 4103 | 2021-10-10 | 54675.093215 |
| 4104 | 2021-10-11 | 57434.240990 |
| 4105 | 2021-10-12 | 56745.906828 |

4106 rows × 2 columns

[3]:
```
# combine the data frames
combined_df = pd.concat([price.set_index('Date'), nupl.set_index('Date')], join='inner', axis='columns').reset_index()
```

[4]:
```
base = alt.Chart(combined_df).mark_line().transform_fold(
    ['NUPL', 'Price'],
    as =['Measure', 'Value']
```

bitcoin.ipynb

Code

Python 3 (ipykernel)

[4]:
```
base = alt.Chart(combined_df).mark_line().transform_fold(
    ['NUPL', 'Price'],
    as_=['Measure', 'Value']
).encode(
    alt.Color('Measure:N'),
    alt.X('Date:T')
).properties(width=800,height=500)

line_A = base.transform_filter(
    alt.datum.Measure == 'NUPL'
).encode(
    alt.Y('Value:Q', axis=alt.Axis(titleColor='purple', title='Net Unrealized Profit/Loss')),
     color=alt.value("purple")
)

line_B = base.transform_filter(
    alt.datum.Measure == 'Price'
).encode(
   alt.Y('Value:Q',axis=alt.Axis(titleColor='green', title='Price of Bitcoin (USD)')),
     color=alt.value("green")
)

alt.layer(line_A, line_B).resolve_scale(y='independent').configure_axis(
    labelFontSize=15,
    titleFontSize=20,

)
```

[4]: