

**PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
NÚCLEO DE EDUCAÇÃO A DISTÂNCIA**

Pós-graduação *Lato Sensu* em Ciência de Dados e Big Data

Ariel Bolzan Witczak

Ravencoin: Predição de preço e de data do primeiro *halving*

Belo Horizonte
2021

Ariel Bolzan Witczak

Aplicação de técnicas de regressão para previsão de preço e de momento de ocorrência do primeiro *halving* da criptomoeda Ravencoin

Trabalho de Conclusão de Curso apresentado ao Curso de Especialização em Ciência de Dados e Big Data como requisito parcial à obtenção do título de especialista.

Belo Horizonte

2021

SUMÁRIO

| | |
|---|----|
| 1. Introdução | 5 |
| 1.1. Contextualização | 6 |
| 1.1.1. Criptomoedas e o funcionamento conceitual de uma <i>blockchain</i> | 6 |
| 1.1.2. Bitcoin (BTC) | 6 |
| 1.1.3. Ravencoin (RVN) | 7 |
| 1.1.4. Comparativo Bitcoin x Ravencoin | 8 |
| 1.1.5. Modelo Stock to Flow (S2F) | 8 |
| 1.1.6. S2F aplicado ao Bitcoin..... | 9 |
| 1.1.7. Correlação de preços: BTC x <i>Altcoins</i> | 11 |
| 1.2. O problema proposto | 13 |
| 2. Coleta de Dados | 14 |
| 2.1. Preços das criptomoedas..... | 14 |
| 2.1 <i>Blockchain</i> RVN..... | 15 |
| 3. Processamento/Tratamento de Dados | 18 |
| 3.1. Padronização dos <i>dataframes</i> | 18 |
| 3.2. Objetivo Primário: <i>rvnS2F.csv</i> | 18 |
| 3.3. Objetivo Secundário: <i>rvnHalvingPrediction.csv</i> | 20 |
| 4. Análise e Exploração dos Dados | 21 |
| 4.1. Objetivo Primário: Previsão do preço da RVN..... | 21 |
| 4.2. Objetivo Secundário: Previsão de <i>halving</i> da RVN | 25 |
| 5. Criação de Modelos de Machine Learning | 29 |
| 5.1 Objetivo Primário: Previsão do preço da RVN..... | 29 |
| 5.2 Objetivo Secundário: Previsão de <i>halving</i> da RVN | 31 |
| 6. Apresentação dos Resultados | 32 |
| 6.1 Objetivo Primário: Previsão do preço da RVN..... | 32 |
| 6.1.1 Regressão múltipla linear..... | 32 |
| 6.1.2 Regressão polinomial | 35 |
| 6.1.3 Comparação de Resultados: | 37 |
| 6.1.4 Análise dos Erros de Previsão da Regressão Polinomial | 38 |
| 6.1.5 Conclusão sobre a aplicabilidade do modelo de Regressão Polinomial Múltipla para previsão do preço da RVN..... | 41 |

| | |
|---|----|
| 6.2 Objetivo Secundário: Previsão de halving da RVN | 42 |
| 6.2.1 Análise dos Erros de Previsão de Halving | 42 |
| 6.2.2 Conclusão sobre a aplicabilidade do modelo de Regressão Linear Simples para previsão de <i>halving</i> | 48 |
| 6.2.3 Previsão dos próximos <i>halvings</i> | 49 |
| 6.3 Conclusão Final..... | 52 |
| 7. Links | 52 |

1. Introdução

Não é objetivo deste trabalho detalhar o funcionamento de uma *blockchain*, tampouco esclarecer a complexa e intrincada dinâmica do mundo experimental das criptomoedas. Também **não serão objeto de estudo** as teorias econômicas ou mesmo a eficiência de modelos matemáticos que podem guardar relação com o tema.

O objetivo em essência é apresentar um projeto de Ciência de Dados e discutir as técnicas aplicadas. Nesses termos, a seção de contextualização foi construída com o objetivo de descrever os conhecimentos básicos necessários para entendimento do problema proposto.

O foco deste trabalho está na criptomoeda Ravencoin (RVN), protocolo baseado em um *fork* (uma bifurcação do código-fonte original para início de um desenvolvimento independente) do Bitcoin (BTC) que adiciona recursos especificamente voltados para tokenização (fragmentação de um ativo real em partes digitais sob a forma de *tokens*).

Iniciaremos abordando brevemente a temática das criptomoedas e o funcionamento conceitual de uma *blockchain*, as características do Bitcoin e da Ravencoin. Na sequência descreveremos brevemente um modelo matemático baseado em escassez que visa fazer uma predição de preço cuja aplicação tentaremos portar do Bitcoin para a Ravencoin.

1.1. Contextualização

1.1.1. Criptomoedas e o funcionamento conceitual de uma *blockchain*

O mundo das criptomoedas vem, de forma crescente, atraindo atenção mundial. Um possível motivo é que durante um *bull market* (mercado altista) vemos disparadas nos preços (*bull runs*) que são praticamente impossíveis de acontecer nos mercados tradicionais. Em abril de 2021, o Bitcoin atingiu uma máxima histórica. Ultrapassou os US\$ 63.000,00 por unidade e alcançou uma valorização de 900% acumulada em um anoⁱ.

Criptomoeda, ou *cryptocurrency*, designa uma moeda digital baseada em *blockchain* e criptografia para assegurar a validade das transações. Cada transação possui uma assinatura eletrônica que identifica origem, destinatário e quanto está sendo enviado.

Estas transações são registradas em uma blockchain, um livro de registro de contabilidade público compartilhado que garante imutabilidade, segurança e transparência.

Para que uma determinada transação seja completada, exige-se um processo de confirmação conhecido como “mineração”. A mineração recompensa os mineradores por meio de taxas e/ou pela criação de novos tokens. Pode empregar diferentes métodos tais como prova de trabalho (emprega força computacional para realizar cálculos matemáticos) ou prova de participação (um detentor de moedas valida blocos e recebe taxas, mas as perde caso os blocos sejam inválidos).ⁱⁱ

1.1.2. Bitcoin (BTC)

Em 2008, Satoshi Nakamoto, pseudônimo do criador do Bitcoin, publicou o white paper “*Bitcoin: A Peer-to-Peer Electronic Cash System*”ⁱⁱⁱ. Em 2 de janeiro de 2009 criou o primeiro bloco (bloco gênese) e disponibilizou o código fonte em 9 de janeiro de 2009^{iv}. Nakamoto, quando do lançamento do primeiro *release*, conceituou Bitcoin como um novo sistema de dinheiro eletrônico que usa uma rede ponto a ponto para evitar gasto duplos, totalmente descentralizado, sem servidor ou autoridade central^v.

O fornecimento total do Bitcoin é limitado no código fonte a 21 milhões de moedas. Novas moedas são criadas durante o processo de mineração, que ocorre a cada 10 minutos e emprega o método prova de trabalho (*Proof of work* – PoW).

Quando do lançamento do Bitcoin, a recompensa pela mineração era de 50 bitcoins por bloco: este número sofre um *halving* (diminuição pela metade) a cada 210.000 novos blocos minerados — o que acontece em média a cada quatro anos. Em 2020, a recompensa por bloco já passou por três *halvings*, sendo o valor atual da recompensa de 6,25 bitcoins.

1.1.3. Ravencoin (RVN)

Lançada em 3 de janeiro de 2018, no nono aniversário do lançamento do Bitcoin, Ravencoin é um *fork* do Bitcoin que adiciona recursos especificamente focados em permitir que novos tokens sejam emitidos na *blockchain* Ravencoin.

Estes tokens podem ser nomeados, limitados em quantidade e emitidos como títulos. Tais títulos podem representar bens virtuais (ingressos, licenças, acesso a serviço ou itens de jogos), colecionáveis (NFT), ativos custodiados no mundo real (como títulos ou ações) ou créditos (cartões de presente, milhas aéreas ou recompensas).^{vi}

Uma curiosidade interessante sobre o nome do projeto é que ele faz referência a ‘*Game of Thrones*’, série de televisão norte-americana. O *white paper* “Ravencoin: A Peer to Peer Electronic System for the Creation and Transfer of Assets” esclarece que: “No mundo fictício de Westeros, os corvos são usados como mensageiros que carregam declarações da verdade. Ravencoin é uma *blockchain* de caso de uso específico projetado para transportar declarações verdadeiras sobre quem possui quais ativos”^{vii}.

A mineração da Ravencoin, assim como o BTC, também emprega PoW e segue a mesma lógica de recompensas. Vale destacar que existe diferença entre os algoritmos de mineração: Ravencoin utiliza KAWPOW, desenvolvido com o objetivo de enfrentar o problema da centralização da mineração. Tal algoritmo funciona bem para mineração em placas de vídeo, mas é resistente a ASICs – *Application-Specific Integrated Circuit*, equipamentos usados em grandes fazendas de mineração.

1.1.4. Comparativo Bitcoin x Ravencoin

As principais diferenças entre os protocolos que estão listadas na tabela abaixo:

| | Bitcoin | Ravencoin |
|-------------------------------------|------------|------------|
| Fornecimento Total | 21 milhões | 21 bilhões |
| Tempo de Bloqueio | 10 minutos | 1 minuto |
| Blocos a cada <i>Halving</i> | 210.000 | 2.100.000 |
| Número de <i>Halvings</i> | 3 | Nenhum |
| Recompensa Inicial | 50 BTC | 5.000 RVN |
| Recompensa Corrente (2021) | 6,25 BTC | 5.000 RVN |

Tabela 1: Comparativo BTC x RVN

1.1.5. Modelo Stock to Flow (S2F)

É um modelo que prevê uma correlação entre preços e escassez. Existindo uma demanda crescente de um recurso finito, esse recurso se torna cada vez mais escasso. Quanto mais raro, mais caro.

O modelo S2F é geralmente aplicado a recursos naturais, como por exemplo o ouro. Embora as estimativas possam variar, o *World Gold Council* estima que cerca de 190.000 toneladas de ouro já foram extraídas. Essa é a oferta total, o que podemos chamar de estoque. Estima-se que cerca de 3.200 toneladas de ouro são extraídas a cada ano, esse valor refere-se ao fluxo. Ao dividir o suprimento total de 190.000 por 3.200, iremos obter a relação Estoque/Fluxo ($SF = 59$). Isso nos diz que, na taxa de produção atual, levaria cerca de 59 anos para extrair 190.000 toneladas de ouro^{viii}.

Quanto maior a relação Estoque / Fluxo, menos oferta nova entra no mercado em relação à oferta total. Como tal, um ativo com uma relação Estoque / Fluxo mais alta deve, em teoria, reter seu valor a longo prazo.

Vale destacar que a escassez por si só não significa necessariamente que um recurso deva ser valioso. Ouro, por exemplo, não é tão raro, existem 190.000 toneladas disponíveis. A relação Estoque / Fluxo sugere que é valioso porque a

produção anual em comparação com o estoque existente é relativamente pequena e constante.^{ix}

1.1.6. S2F aplicado ao Bitcoin

A aplicação do modelo *Stock to Flow* ao Bitcoin é frequentemente atribuída a PlanB e a seu artigo de 22 de março de 2019, “Modeling Bitcoin Value with Scarcity”^x. Este modelo compara o Bitcoin a *commodities* como ouro, prata ou platina. Conhecidas como *commodities* de ‘reserva de valor’ pois, dada a sua relativa escassez, retêm valor por longos períodos.

É difícil aumentar significativamente a oferta dessas *commodities*, o processo de extração é caro e demorado. O Bitcoin é semelhante porque também é escasso, é o primeiro objeto digital escasso a existir. Existe um fornecimento fixo de moedas (21 milhões) e será necessário muito esforço computacional para extrair os 3 milhões de moedas restantes, portanto, a taxa de fornecimento é consistentemente baixa^{xi}.

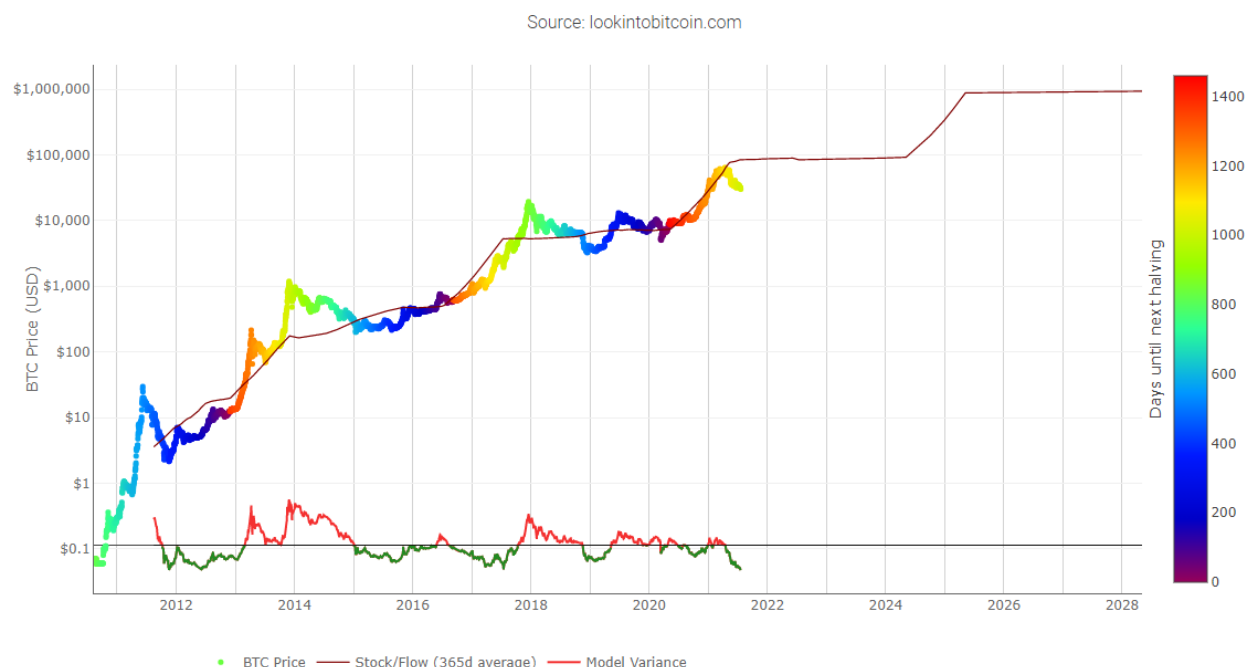


Figura 1: Modelo S2F do Bitcoin

A recompensa para a mineração do Bitcoin iniciou com 50 BTC e é reduzida pela metade a cada 210.000 blocos (cerca de 4 anos). Nesse cenário os *halvings* são importantes pois fazem com que a taxa de crescimento da oferta (no contexto da bitcoin geralmente é conhecida como ‘inflação monetária’) seja escalonada e não uniforme.

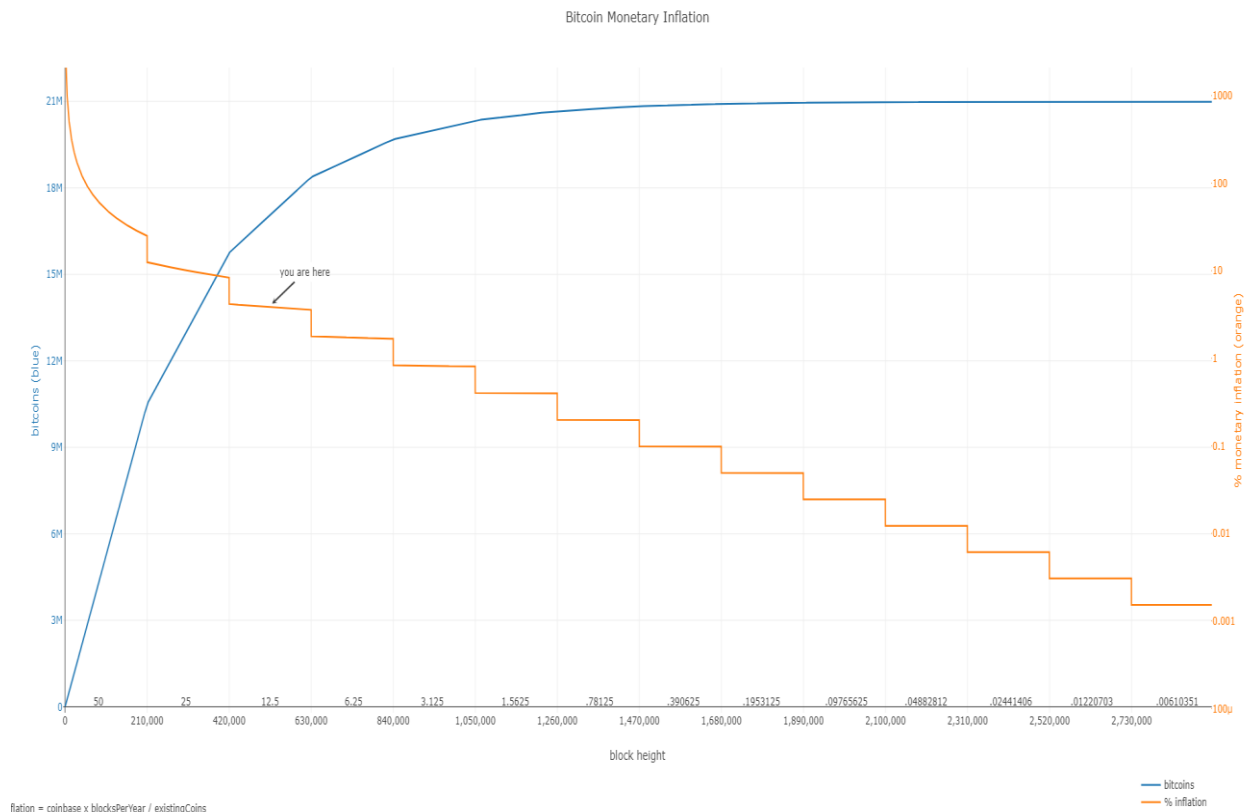


Figura 2: Inflação do Bitcoin^{xii}

Apesar do S2F ser um dos modelos de avaliação mais populares do Bitcoin, não é isento de críticas. Um relatório de autoria publicado em 2020 da equipe de pesquisa da ByteTree indica que esse modelo fornece uma previsão muito otimista para o Bitcoin. “Existem muitos motivos pelos quais o preço do Bitcoin pode subir ou cair, mas o S2F não é um deles”, afirma o autor do relatório.^{xiii}

Nesse ponto vale comentar que não é objetivo desse trabalho refutar ou mesmo confirmar a validade da teoria econômica que suporta o modelo S2F. Esse é um modelo bastante popular na comunidade cripto, fato que justifica o entendimento de sua aplicação e de seus pressupostos.

1.1.7. Correlação de preços: BTC x Altcoins

Uma questão que merece destaque é que existe alta correlação entre o preço do Bitcoin e das *altcoins* (criptomoedas alternativas ao Bitcoin). Notícia publicada em 2019, pelo site Livecoins, informa que a Binance (uma das maiores corretoras de criptomoedas do mundo) publicou um relatório sobre essas correlações. Segundo o site, a correlação média seria de 0,78. Indicando que o comportamento do preço do BTC impacta de maneira importante as principais *altcoins* em valor de mercado. Como a RVN é uma *altcoin*, é importante analisar essa potencial correlação no desenvolvimento do trabalho.^{xiv}

Gráfico 1 - Matriz de correlação de retorno diário de três meses do ativo large-cap (USD) 01 de dezembro de 2018 - 01 mar 2019

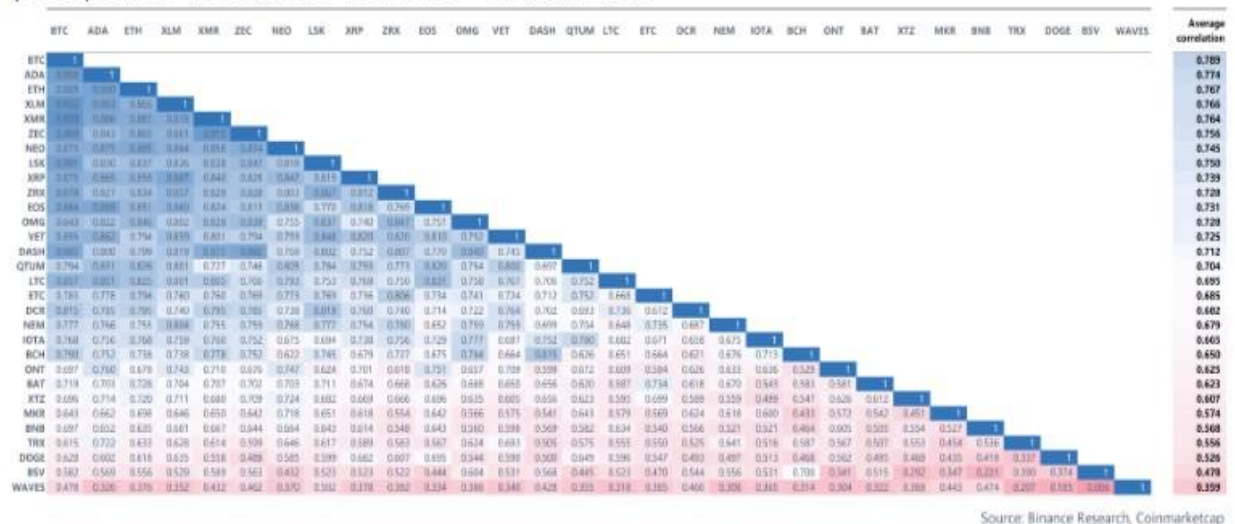


Figura 3: Correlações BTC x Altcoins

Artigo recente (05/2021), publicado em Analytics Vidhya, confirma a existência de correlação entre o preço das criptomoedas^{xv}.

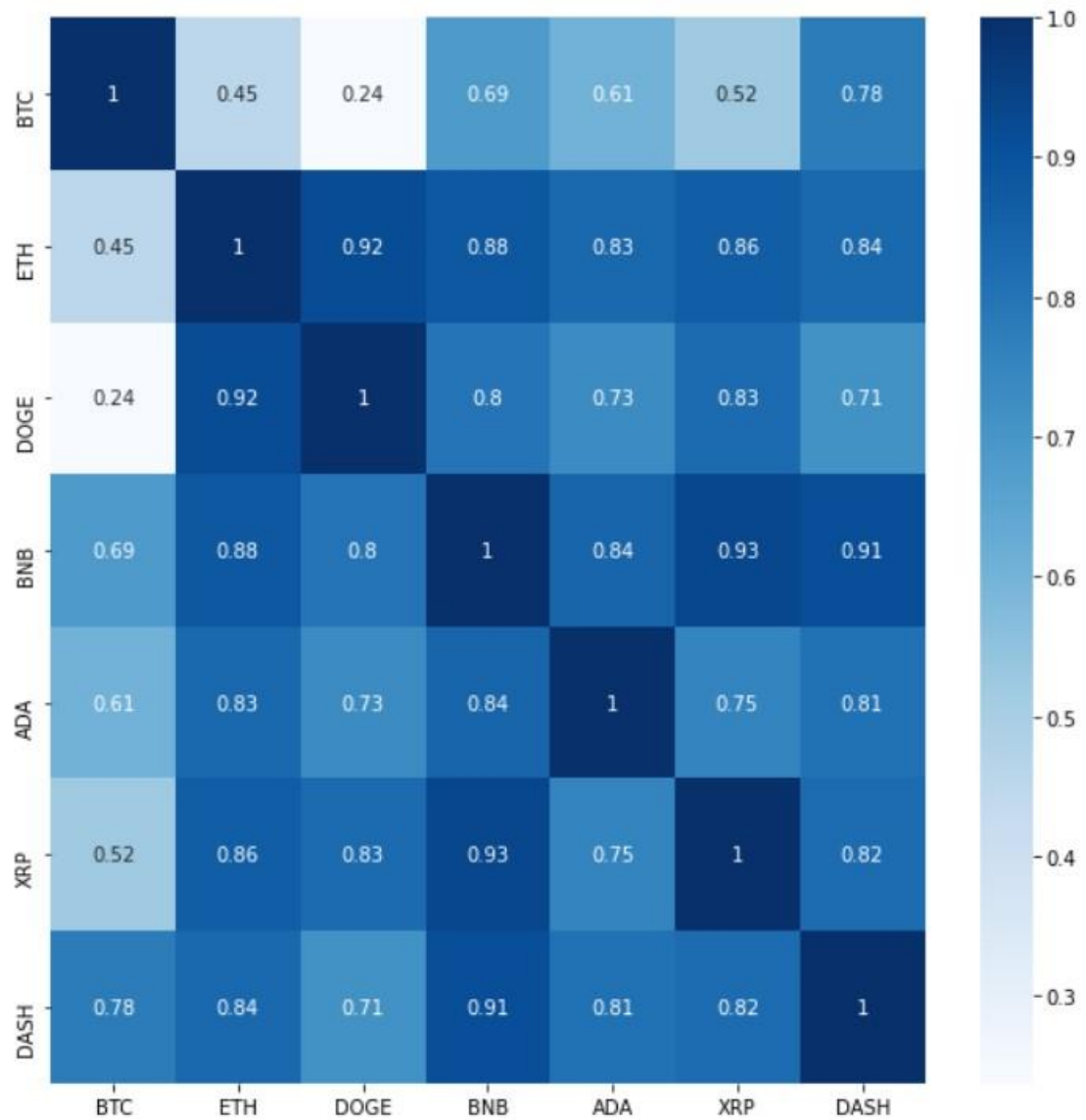


Figura 4 - Heatmap de Analytics Vidhya

1.2. Problema proposto

O **objetivo primário** deste trabalho será testar duas hipóteses com a finalidade de obter uma predição do preço da Ravencoin:

1. **HIPÓTESE 1:** A partir do modelo S2F adaptado para RVN e/ou
2. **HIPÓTESE 2:** A partir do preço e/ou volume (montante comprado e vendido em um determinado período) do BTC e/ou da RVN

Como **objetivo secundário**, utilizar os mesmos dados já coletados para prever a data do primeiro *halving* da RVN.

Serão utilizados apenas dados públicos:

- A. Dados da *blockchain* da RVN (<https://ravencoin.network/>) e
- B. Dados de preço e volume dos ativos RVN e BTC, obtidos no site *Yahoo Finance* (<https://finance.yahoo.com/>).

O período objeto de análise inicia com a data em que foi minerado o primeiro bloco da RVN, 03/01/2018, e vai até o dia 31/05/2021.

2. Coleta de Dados

2.1. Preços das criptomoedas

Os dados históricos de preços das criptomoedas foram obtidos no site Yahoo Finance (<https://finance.yahoo.com/>), em 03/07/2021, e estão estruturados conforme tabela abaixo:

| | | |
|------------------|--|--------------|
| <i>Date</i> | Data | <i>Date</i> |
| <i>Open</i> | Preço de abertura (US\$) | <i>Float</i> |
| <i>High</i> | Maior preço do dia (US\$) | <i>Float</i> |
| <i>Low</i> | Menor preço do dia (US\$) | <i>Float</i> |
| <i>Close</i> | Preço de fechamento (US\$) | <i>Float</i> |
| <i>Adj Close</i> | Preço de fechamento ajustado (US\$) No caso das criptomoedas, não há distribuição de dividendos. Logo o preço de fechamento (<i>Close</i>) é igual ao preço de fechamento ajustado (<i>Adj Close</i>) Fonte: https://help.yahoo.com/kb/SLN28256.html | <i>Float</i> |
| <i>Volume</i> | Montante comprado e vendido em um determinado período (US\$) | <i>Float</i> |

Link para download direto dos *datasets*:

- RVN:

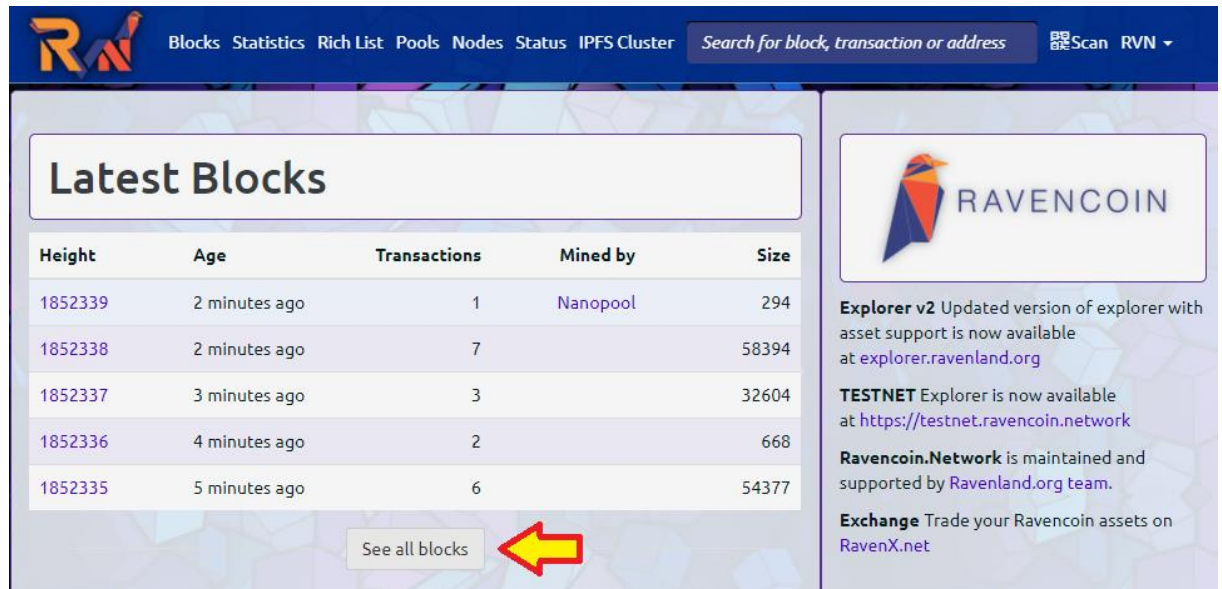
<https://au.finance.yahoo.com/quote/RVN-USD/history?period1=1520640000&period2=1622419200&interval=1d&filter=history&frequency=1d&includeAdjustedClose=true>

- BTC:

<https://au.finance.yahoo.com/quote/RVN-USD/history?period1=1520640000&period2=1622419200&interval=1d&filter=history&frequency=1d&includeAdjustedClose=true>

2.1 Blockchain RVN

Para obtenção dos dados da *blockchain* da RVN foi necessário construir um script de *web scraping* para o site <https://ravencoin.network/>.



The screenshot shows the RavenCoin Explorer homepage. The top navigation bar includes links for Blocks, Statistics, Rich List, Pools, Nodes, Status, and IPFS Cluster, along with a search bar and a 'Scan RVN' button. The main content area features a 'Latest Blocks' section with a table of recent blocks. A red arrow points to the 'See all blocks' button at the bottom of this section.

| Height | Age | Transactions | Mined by | Size |
|---------|---------------|--------------|----------|-------|
| 1852339 | 2 minutes ago | 1 | Nanopool | 294 |
| 1852338 | 2 minutes ago | 7 | | 58394 |
| 1852337 | 3 minutes ago | 3 | | 32604 |
| 1852336 | 4 minutes ago | 2 | | 668 |
| 1852335 | 5 minutes ago | 6 | | 54377 |

See all blocks

RAVENCOIN

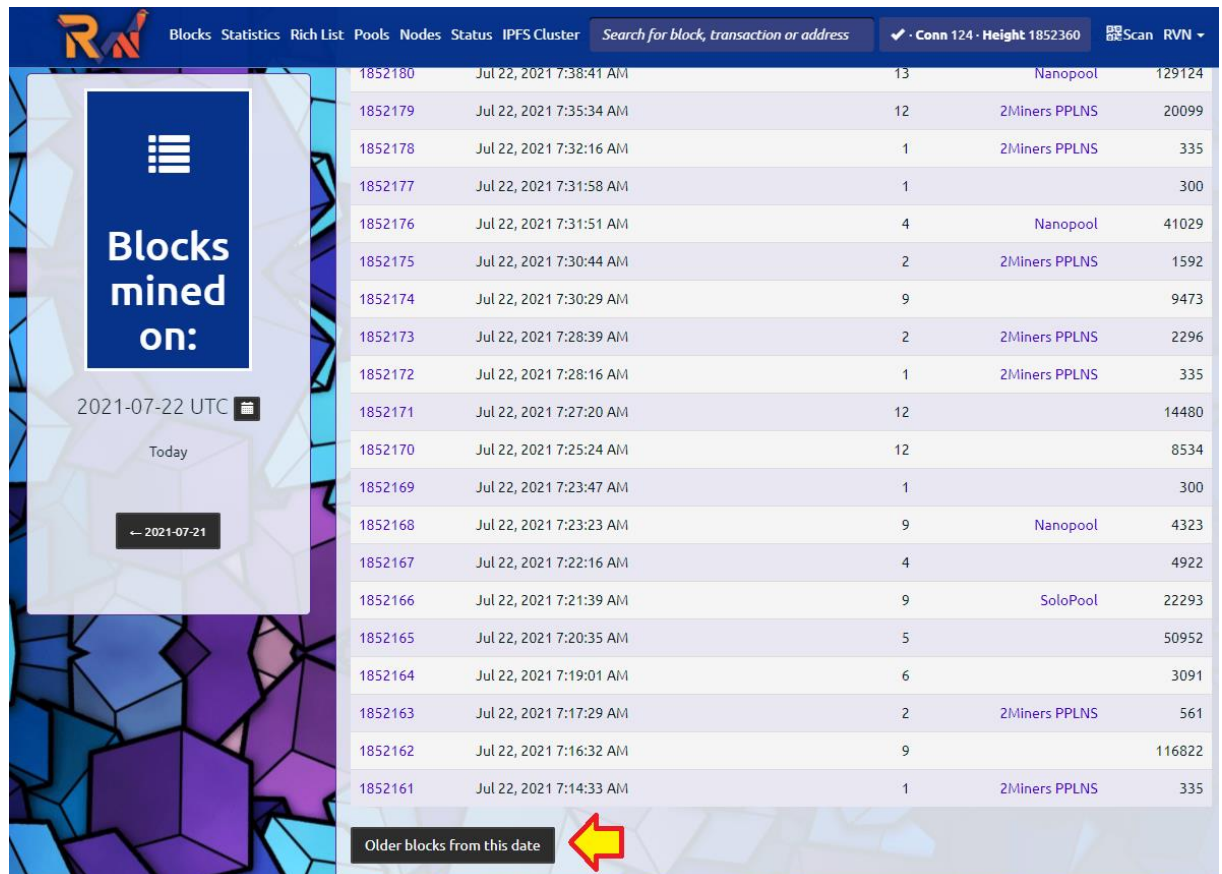
Explorer v2 Updated version of explorer with asset support is now available at explorer.ravenland.org

TESTNET Explorer is now available at <https://testnet.ravencoin.network>

Ravencoin.Network is maintained and supported by [Ravenland.org](https://ravenland.org) team.

Exchange Trade your Ravencoin assets on RavenX.net

Na página inicial, ao clicar sobre o botão “See all blocks”, o site redireciona para uma página que lista em tempo real todos os blocos minerados na *blockchain* da RVN.



The screenshot shows the 'Blocks mined on:' page on the RavenCoin Explorer. The left sidebar displays a calendar for July 22, 2021, with a date selector set to '2021-07-21'. The main content area is a table listing blocks mined on that date. A red arrow points to the 'Older blocks from this date' button at the bottom of the table.

| Height | Time | Transactions | Mined by | Size |
|---------|-------------------------|--------------|---------------|--------|
| 1852180 | Jul 22, 2021 7:38:41 AM | 13 | Nanopool | 129124 |
| 1852179 | Jul 22, 2021 7:35:34 AM | 12 | 2Miners PPLNS | 20099 |
| 1852178 | Jul 22, 2021 7:32:16 AM | 1 | 2Miners PPLNS | 335 |
| 1852177 | Jul 22, 2021 7:31:58 AM | 1 | | 300 |
| 1852176 | Jul 22, 2021 7:31:51 AM | 4 | Nanopool | 41029 |
| 1852175 | Jul 22, 2021 7:30:44 AM | 2 | 2Miners PPLNS | 1592 |
| 1852174 | Jul 22, 2021 7:30:29 AM | 9 | | 9473 |
| 1852173 | Jul 22, 2021 7:28:39 AM | 2 | 2Miners PPLNS | 2296 |
| 1852172 | Jul 22, 2021 7:28:16 AM | 1 | 2Miners PPLNS | 335 |
| 1852171 | Jul 22, 2021 7:27:20 AM | 12 | | 14480 |
| 1852170 | Jul 22, 2021 7:25:24 AM | 12 | | 8534 |
| 1852169 | Jul 22, 2021 7:23:47 AM | 1 | | 300 |
| 1852168 | Jul 22, 2021 7:23:23 AM | 9 | Nanopool | 4323 |
| 1852167 | Jul 22, 2021 7:22:16 AM | 4 | | 4922 |
| 1852166 | Jul 22, 2021 7:21:39 AM | 9 | SoloPool | 22293 |
| 1852165 | Jul 22, 2021 7:20:35 AM | 5 | | 50952 |
| 1852164 | Jul 22, 2021 7:19:01 AM | 6 | | 3091 |
| 1852163 | Jul 22, 2021 7:17:29 AM | 2 | 2Miners PPLNS | 561 |
| 1852162 | Jul 22, 2021 7:16:32 AM | 9 | | 116822 |
| 1852161 | Jul 22, 2021 7:14:33 AM | 1 | 2Miners PPLNS | 335 |

Older blocks from this date

Ao final dessa nova página encontramos o botão “Older blocks from this date”. Ao acioná-lo, com a guia Chrome DevTools ativa, é possível observar que existe uma API que fornece dados dos blocos a partir da informação da data e do respectivo *timestamp* (marca temporal formada por uma cadeia de caracteres que denota data e hora de determinado evento).

The screenshot shows the RavenCoin website interface. At the top, it says "Blocks mined on: 2021-07-22 UTC". Below this, there's a button labeled "Today" and another button labeled "← 2021-07-21". The main section is titled "Blocks by date. On 2021-07-22 before 7:17". It contains a table with the following data:

| Height | Timestamp | Transactions | Size |
|---------|-------------------------|--------------|--------|
| 1852162 | Jul 22, 2021 7:16:32 AM | 9 | 116822 |
| 1852161 | Jul 22, 2021 7:14:33 AM | 1 | 335 |
| 1852160 | Jul 22, 2021 7:14:30 AM | 4 | 3799 |
| 1852159 | Jul 22, 2021 7:13:32 AM | 1 | 300 |

On the right side, the Chrome DevTools Network tab is open, showing a list of requests. The request "blocks?blockDate=2021-07-22&startTimestamp=1626949049" is highlighted with a red box and a yellow arrow.

API: <https://ravencoin.network/api/blocks?blockDate=2021-07-22&startTimestamp=1626949049>

Ao explorar o link dessa API obteremos dados em um formato JSON (*JavaScript Object Notation*) segmentado em 3 estruturas, uma contendo dados dos blocos minerados ("*blocks*"), tamanho ("*length*") e paginação ("*pagination*").

```
{
  "blocks": [
    {
      "height": 1852162,
      "size": 116822,
      "hash": "00000000000083fb9a6dd9c1945b97c03daacb52a3396b2708fcbc6b70c6f53b",
      "time": 1626948992,
      "txlength": 9,
      "poolInfo": {},
      "isMainChain": true,
      "minedBy": "RTpp8G7Y5f9HZ1iGNz1gtbWazwnHvoHCxK"
    },
    {
      "height": 1852161,
      "size": 335,
      "hash": "000000000000548fe2933761f055da026513dd2aae897f4e26ff6caace7621e2",
      "time": 1626948873,
      "txlength": 1,
      "poolInfo": {
        "poolName": "2Miners PPLNS",
        "url": "https://rvn.2miners.com"
      },
      "isMainChain": true,
      "minedBy": "RHUC17zAVjNqXDTkqzLPRvQ2XgoRZsXeeG"
    }
  ],
  "length": 200,
  "pagination": {
    "next": "2021-07-22",
    "prev": "2021-07-21",
    "currentTs": 1626949048,
    "current": "2021-07-22",
    "isToday": true,
    "more": true,
    "moreTs": 1626937138
  }
}
```


Os dados dos blocos são capturados pelo script “2 - Data Collect - RVN Blocks (Extraction via API)”. O script inicia capturando dados a partir do *timestamp* 1622516262, que corresponde a 31/05/2021 23:57:42. Seguindo em *loop* até extrair o primeiro bloco criado na *blockchain* da RVN. Resulta no arquivo ‘rvnOriginal.csv’.

A tabela abaixo esclarece a estrutura de dados extraída:

| | | |
|--------------------|---|----------------|
| <i>height</i> | Número inteiro incremental que corresponde ao bloco minerado, o primeiro bloco começou em 1. | <i>Integer</i> |
| <i>size</i> | Tamanho do bloco dado em <i>bytes</i> | <i>Integer</i> |
| <i>hash</i> | <i>Hash</i> criptográfico que referencia um bloco na blockchain Também conhecido como <i>BlockHash</i> | <i>String</i> |
| <i>time</i> | <i>Timestamp</i> faz referência ao momento de criação do bloco | <i>Integer</i> |
| <i>txlength</i> | Número de transações contidas no bloco | <i>Integer</i> |
| <i>poolInfo</i> | Indica a <i>pool</i> de mineração*, caso tenha sido utilizada | <i>String</i> |
| <i>isMainChain</i> | Indica se o bloco pertence à corrente principal | <i>Boolean</i> |
| <i>minedBy</i> | Indica a carteira que recebeu a recompensa pelos blocos minerados | <i>String</i> |

* *Pools* de Mineração (ou piscinas de mineração) é uma forma cooperativa de mineração em que poder de mineração de cada nó é somado. A recompensa é distribuída para cada minerador com base no *quantum* de poder computacional com que cada um contribuiu. Ou seja, a recompensa é dividida proporcionalmente entre cada participante, tornando a mineração muito mais rápida. Como mais blocos são minerados, os pagamentos são mais frequentes.

Vale comentar que a maior parte dos dados coletados acabou não sendo necessária para atender os objetivos desse trabalho, mas ficam disponíveis para trabalhos futuros.

3. Processamento/Tratamento de Dados

Todos os *dataframes* descritos nessa seção são obtidos com a execução do script '3 - Data Processing - RVN Blocks'

3.1. Padronização dos *dataframes*

A estrutura dos *dataframes* passou por uma padronização em que todas as letras passam a ser minúsculas, o caractere correspondente ao espaço em branco é substituído por *underscore* e um sufixo é adicionado a cada nome de campo de forma a identificar a que criptomoeda corresponde uma determinada coluna.

```
# Normalization/standardization of column names
def renamingColumns(df, suffix):

    df.columns = [col.lower() for col in df.columns]
    df.columns = [col_name+'_'+suffix for col_name in df.columns]
    df.columns = [c.replace(' ', '_') for c in df.columns]
    df.rename(columns = {'date_'+suffix:'date'}, inplace = True)

    df.head

    return df
```

3.2. Objetivo Primário: rvnS2F.csv

Dataset construído a partir da junção dos dados da *blockchain* da RVN (rvnOriginal.csv) e dos preços históricos da RVN (RVN-USD.csv) do BTC (BTC-USD.csv).

Antes da junção foi eliminada a maioria das colunas da tabela que contém dados da *blockchain* da RVN. Foram mantidos apenas os campos *height* (que corresponde ao número do bloco) e *time* (*timestamp* que referencia o momento de criação do bloco). Demais campos não serão utilizados neste estudo.

O campo *time* é convertido para data e é adicionada a coluna *flow_RVN*, que corresponde à variável fluxo do modelo S2F. O valor do estoque é obtido multiplicando-se o número de blocos minerado ao dia por 5 mil, valor da recompensa por bloco.

```
# Convert timestamp to date format
# Useful for making the timestamp humanely friendly

def timestampToDate(timestamp):

    return datetime.fromtimestamp(timestamp).date()
```

```
# Creating the dataframe for the S2F model with the flow field , stores the amount of
#   blocks generated in one day * 5000 (amount of mined coins per block)
# Important: based on days

dfS2F = df.copy()

# Handling the dates
dfS2F['date'] = dfS2F['time']
dfS2F['date'] = dfS2F['date'].apply(timestampToDate)

dfS2F['flow'] = 0
dfS2F = dfS2F.groupby('date').count() * 5000

dfS2F = dfS2F.drop('time', 1)

# Renaming columns
renamingColumns(dfS2F, 'RVN')

dfS2F
```

Adiciona-se o campo *stock_RVN*, que corresponde a soma cumulativa da variável fluxo.

```
# Adding the stock variable

dfS2F['stock_RVN'] = dfS2F['flow_RVN'].cumsum()
dfS2F
```

Por fim é adicionada a coluna *SF_RVN*, que define a variável S2F (razão do estoque pelo fluxo), necessária para testar a validade da hipótese de que a escassez tem influência sobre o preço da RVN. Vide script '4 - Analysis, Exploration and Model - RVN Blocks'.

```
# The hypothesis in this study is that scarcity, as measured by SF, directly drives value.
df['SF_RVN'] = df['stock_RVN'] / df['flow_RVN']
df
```

Em relação ao *dataframe* de preços históricos da RVN (RVN-USD.csv), só foi possível encontrar dados de preço a partir de 09/03/2018. Então, após a junção, foi necessário excluir as linhas que não possuíam preços.

Ao final de todas as transformações obteve-se um dataset com 1.172 observações diárias, cada uma contendo número do bloco, data, preço e volume da RVN e da BTC.

| | date | adj_close_RVN | volume_RVN | adj_close_BTC | volume_BTC | SF_RVN |
|------------|------------|---------------|--------------|---------------|------------------|----------|
| height_RVN | | | | | | |
| 7145000 | 2018-03-10 | 0.02862 | 171820.00000 | 8866.00000 | 5386319872.00000 | 79.49615 |
| 7380000 | 2018-03-11 | 0.03188 | 279104.00000 | 9578.62988 | 6296370176.00000 | 77.96477 |
| 7345000 | 2018-03-12 | 0.03026 | 218114.00000 | 9205.12012 | 6457399808.00000 | 79.33628 |
| 7675000 | 2018-03-13 | 0.02790 | 167669.00000 | 9194.84961 | 5991139840.00000 | 76.92508 |
| 7455000 | 2018-03-14 | 0.02439 | 131838.00000 | 8269.80957 | 6438230016.00000 | 80.19517 |
| 7170000 | 2018-03-15 | 0.02447 | 114232.00000 | 8300.86035 | 6834429952.00000 | 84.38285 |
| 7595000 | 2018-03-16 | 0.02453 | 121884.00000 | 8338.34961 | 5289379840.00000 | 80.66096 |
| 7325000 | 2018-03-17 | 0.02277 | 108011.00000 | 7916.87988 | 4426149888.00000 | 84.63413 |
| 7260000 | 2018-03-18 | 0.02262 | 141594.00000 | 8223.67969 | 6639190016.00000 | 86.39187 |
| 7055000 | 2018-03-19 | 0.02158 | 91701.00000 | 8630.65039 | 6729110016.00000 | 89.90220 |

3.3. Objetivo Secundário: rvnHalvingPrediction.csv

Corresponde ao dataset rvnOriginal.csv, mantidos apenas os campos *height* e *time*. Obteve-se 1.774.760 observações, uma por bloco minerado desde o início da *blockchain* da RVN até 31/05/2021.

| | height | time |
|---------|---------|------------|
| 0 | 1778781 | 1622516234 |
| 1 | 1778780 | 1622516196 |
| 2 | 1778779 | 1622516185 |
| 3 | 1778778 | 1622516158 |
| 4 | 1778777 | 1622515956 |
| ... | ... | ... |
| 1774755 | 5 | 1515015840 |
| 1774756 | 4 | 1515015833 |
| 1774757 | 3 | 1515015816 |
| 1774758 | 2 | 1515015759 |
| 1774759 | 1 | 1515015723 |

1774760 rows × 2 columns

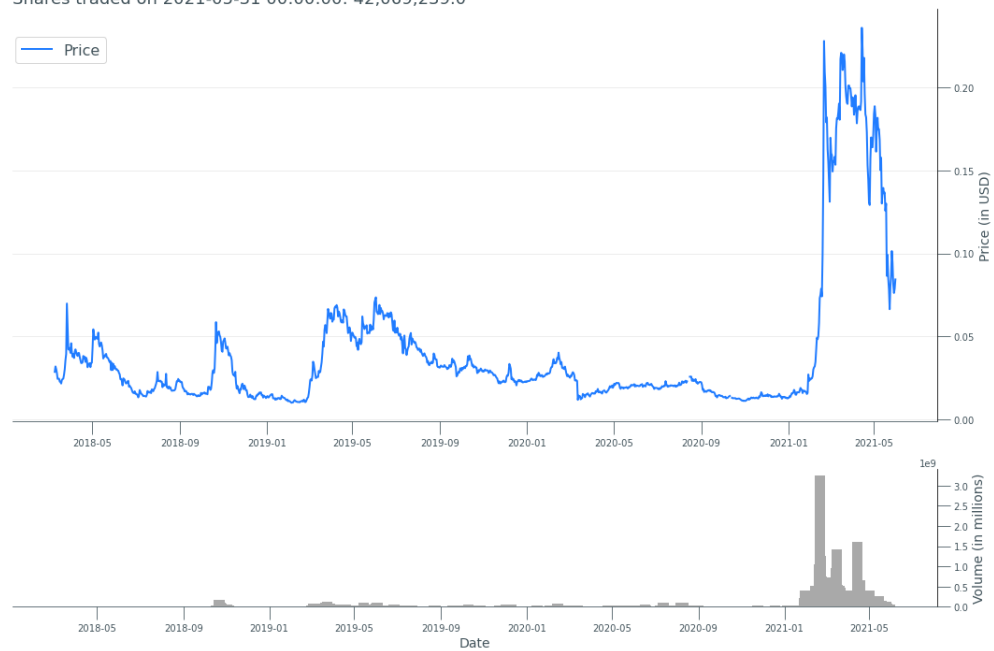
4. Análise e Exploração dos Dados

4.1. Objetivo Primário: Previsão do preço da RVN

Visualização de preço e volume de RVN e BTC, sem aplicar nenhum tipo de ajuste nos dados. Vide script '1 - Data Exploration - Price Volume History'.

RVN Price and Volume

Closing price on 2021-05-31 00:00:00: \$0.084311
Shares traded on 2021-05-31 00:00:00: 42,069,239.0

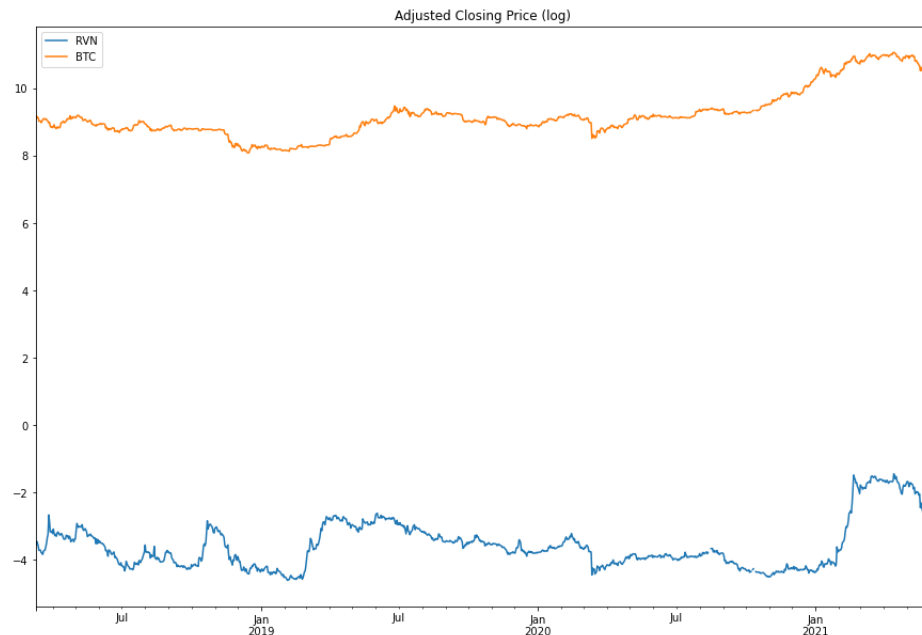


BTC Price and Volume

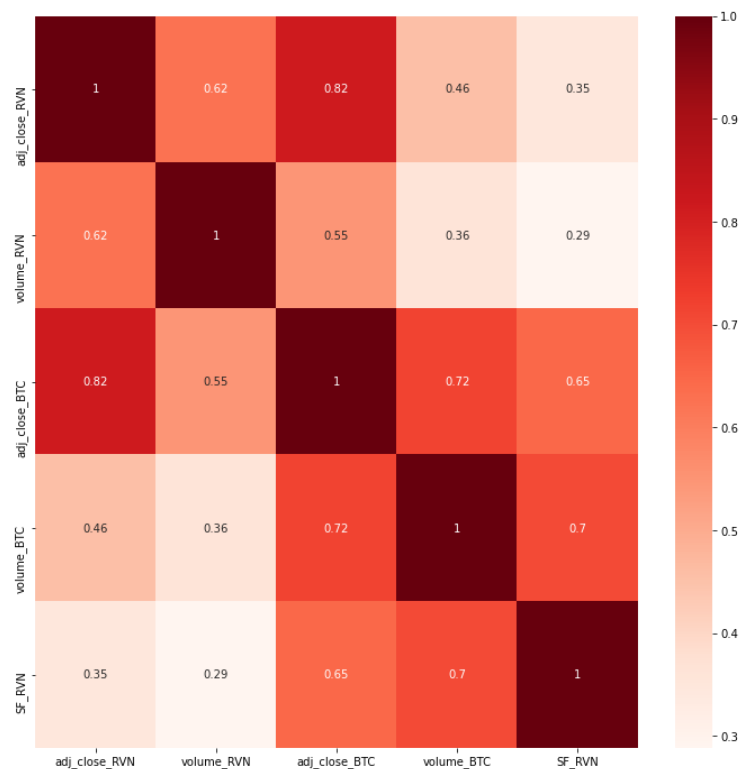
Closing price on 2021-05-31 00:00:00: \$37,332.855469
Shares traded on 2021-05-31 00:00:00: 39,009,847,639.0



No período em análise (03/10/2018 a 31/05/2021), a cotação do BTC era da ordem de milhares de dólares enquanto a do RVN era de centavos. Para que fosse possível apresentar esses dados no mesmo gráfico, foi necessário aplicar log aos preços.



A partir do gráfico já é possível inferir que pode haver alguma correlação entre o preço dos ativos, hipótese confirmada no *heatmap* abaixo. Vide script '4 - Analysis, Exploration and Model - RVN Blocks'.



Resultados da correlação do preço do Ravencoin:

- Tem uma forte correlação com o preço do Bitcoin.

HIPÓTESE 2 - VÁLIDA

- A segunda maior correlação identificada é com o volume de operações da RVN seguido pelo volume de BTC.
- Apresenta fraca correlação com a variável SF (estoque / fluxo), o que invalida a aplicação do modelo S2F ao Ravencoin.

HIPÓTESE 1 - INVÁLIDA

Análise dos resultados:

1. É relevante destacar que o período objeto deste estudo (03/10/2018 a 31/05/2021) antecede a ocorrência do primeiro *halving* da RVN, previsto para ocorrer em janeiro de 2022^{xvi}.

Note que o fenômeno “*halving*” não está presente nos dados, por isso é importante repetir esta análise após a sua ocorrência. Assim poderemos confirmar se realmente não possui relevância estatística.

2. Observamos que o preço do Bitcoin possui forte correlação com o preço do Ravencoin. Conforme artigo recente (05/2021) publicado no site *Analytics Vidhya*, esse fenômeno pode ser observado em várias *altcoins*^{xvii}.

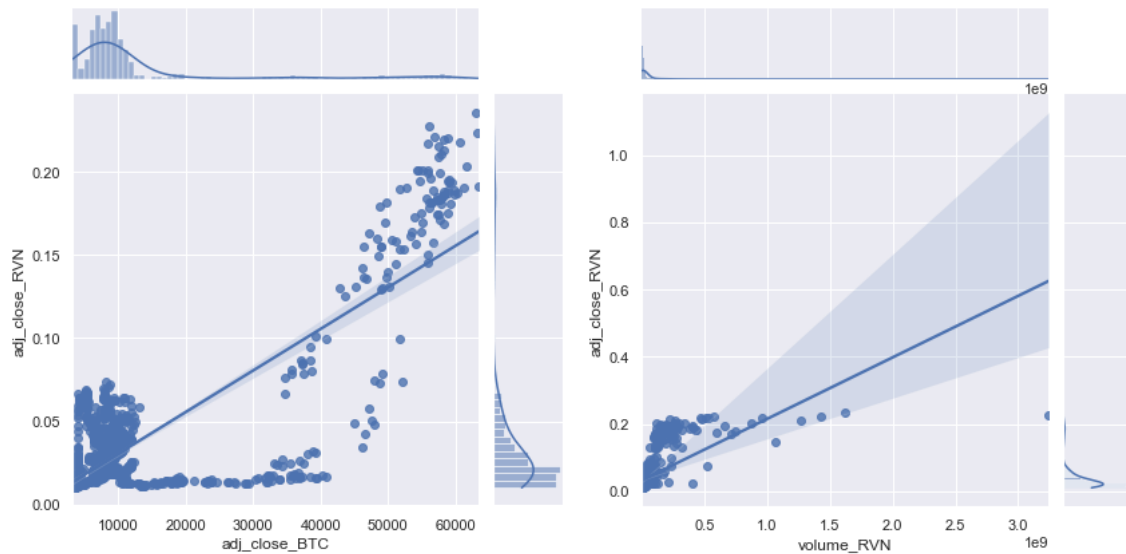
Vale observar que mantida essa alta correlação com o BTC, é possível que, mesmo após o primeiro *halving*, o modelo S2F não obtenha bons resultados em trabalhos futuros.

3. Uma correlação importante entre o preço do RVN e seu volume indica que o preço pode ser determinado pela oferta e demanda da própria RVN. Ou seja, o preço aumenta à medida que a demanda aumenta.

Vale lembrar que há um número limitado de Ravencoins em circulação e novos RVNs são criados a uma taxa previsível e decrescente. Isso denota uma tendência deflacionária de longo prazo. Porém, no curto prazo, observamos que o preço do BTC é mais relevante na determinação do preço RVN do que o próprio volume RVN.

4. O preço da RVN tem correlação mais forte com o volume de BTC do que com a variável S2F (estoque / fluxo), indicando que o modelo S2F não é eficiente para prever preços RVN. Isso confirma a segunda hipótese do objetivo primário.

O próximo passo é verificar a lineariedade dos dados de preço do BTC e de volume da RVN em relação ao preço da RVN.



Observa-se que não há um ajuste perfeito, logo uma podemos supor que uma técnica regressão polinomial poderia obter melhores do que uma regressão linear. Testaremos essa hipótese quando da construção dos modelos.

Por fim, antes de seguir para o modelo, é relevante verificar se existe multicolineariedade relevante entre os dados que servirão de insumo do modelo.

```
# The function used to check the multicollinearity hypothesis is an adaptation of the code created by A
# Source: https://www.analyticsvidhya.com/blog/2020/03/what-is-multicollinearity/

def calc_vif(X):
    # Calculating VIF
    vif = pd.DataFrame()
    vif["variables"] = X.columns
    vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

    return(vif)

myVif = df.iloc[:,1:]
calc_vif(myVif)
```

| | variables | VIF |
|---|---------------|---------|
| 0 | volume_RVN | 1.46578 |
| 1 | adj_close_BTC | 1.46578 |

Temos que VIF (*Variable Inflation Factors*) é igual a 1,46. Logo não há correlação relevante entre as variáveis independentes: preço do BTC e volume da RVN. Assim, não temos problemas relacionado à multicolineariedade.

4.2. Objetivo Secundário: Previsão de *halving* da RVN

Antes de entrar no tratamento e na análise dos dados, é importante esclarecer a estratégia que empregaremos para cumprir o desafio adicional de utilizar apenas os dados já coletados.

Nos dados disponíveis temos o número do bloco (*height*) e o momento em que o bloco foi criado (*timestamp*). A ocorrência do halving é determinada pelo número do bloco, ocorre a cada 2.100.000 blocos. Devemos observar também que apesar de *timestamp* referenciar um momento no tempo, trata-se de um número inteiro que pode ser obtido a partir de uma técnica de regressão. Desta forma tentaremos prever o momento em que determinado bloco é criado utilizando uma regressão linear simples.

Com esta estratégia em mente, iniciamos convertendo *timestamp* em um formato de data e hora. Vide script '5 - Extra - Halving Projection'.

```
# Convert timestamp to date format
# Useful for making the timestamp humanely friendly

def timestampToDate(timestamp):

    return timestampToDateTime(timestamp).date()
```

```
# Convert timestamp to date and time format
# Useful for making the timestamp humanely friendly

def timestampToDateTime(timestamp):

    return datetime.fromtimestamp(timestamp)
```

```
# Handling the dates
dfExploration['dateTime'] = dfExploration['time']
dfExploration['dateTime'] = dfExploration['dateTime'].apply(timestampToDateTime)
dfExploration['monthYear'] = dfExploration['dateTime'].dt.to_period('M')

dfExploration.head(10)
```

| | height | time | dateTime | monthYear |
|---|---------|------------|---------------------|-----------|
| 0 | 1778781 | 1622516234 | 2021-05-31 23:57:14 | 2021-05 |
| 1 | 1778780 | 1622516196 | 2021-05-31 23:56:36 | 2021-05 |
| 2 | 1778779 | 1622516185 | 2021-05-31 23:56:25 | 2021-05 |
| 3 | 1778778 | 1622516158 | 2021-05-31 23:55:58 | 2021-05 |
| 4 | 1778777 | 1622515956 | 2021-05-31 23:52:36 | 2021-05 |
| 5 | 1778776 | 1622515953 | 2021-05-31 23:52:33 | 2021-05 |
| 6 | 1778775 | 1622515941 | 2021-05-31 23:52:21 | 2021-05 |
| 7 | 1778774 | 1622515909 | 2021-05-31 23:51:49 | 2021-05 |
| 8 | 1778773 | 1622515865 | 2021-05-31 23:51:05 | 2021-05 |
| 9 | 1778772 | 1622515833 | 2021-05-31 23:50:33 | 2021-05 |

Agora devemos determinar o tempo real de bloqueio, intervalo de tempo entre a criação novos blocos, em uma escala de segundos. Essa variável (*secondsBetweenBlocks*) funciona como *proxy* para representar a força computacional aplicada na rede pelos mineradores e o grau de dificuldade de mineração.

```
# Time between blocks
```

```
dfExploration['secondsBetweenBlocks'] = dfExploration['dateTime'].diff().astype('timedelta64[s'])*-1
dfExploration['secondsBetweenBlocks'] = dfExploration['secondsBetweenBlocks'].fillna(0)
dfExploration.head(10)
```

| | height | time | dateTime | monthYear | secondsBetweenBlocks |
|---|---------|------------|---------------------|-----------|----------------------|
| 0 | 1778781 | 1622516234 | 2021-05-31 23:57:14 | 2021-05 | 0.00000 |
| 1 | 1778780 | 1622516196 | 2021-05-31 23:56:36 | 2021-05 | 38.00000 |
| 2 | 1778779 | 1622516185 | 2021-05-31 23:56:25 | 2021-05 | 11.00000 |
| 3 | 1778778 | 1622516158 | 2021-05-31 23:55:58 | 2021-05 | 27.00000 |
| 4 | 1778777 | 1622515956 | 2021-05-31 23:52:36 | 2021-05 | 202.00000 |
| 5 | 1778776 | 1622515953 | 2021-05-31 23:52:33 | 2021-05 | 3.00000 |
| 6 | 1778775 | 1622515941 | 2021-05-31 23:52:21 | 2021-05 | 12.00000 |
| 7 | 1778774 | 1622515909 | 2021-05-31 23:51:49 | 2021-05 | 32.00000 |
| 8 | 1778773 | 1622515865 | 2021-05-31 23:51:05 | 2021-05 | 44.00000 |
| 9 | 1778772 | 1622515833 | 2021-05-31 23:50:33 | 2021-05 | 32.00000 |

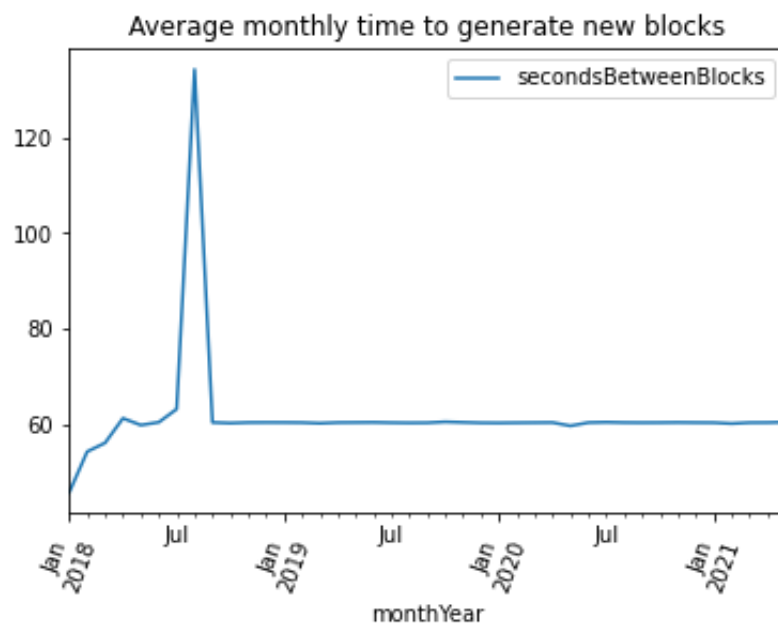
Podemos verificar que está correto o cálculo da diferença, dada em segundos, entre a criação de um bloco e outro. Agora vamos obter a média mensal.

```
# After verifying that the time between blocks is correct, time and dateTime can be deleted
dfExploration = dfExploration.drop('time', 1)
dfExploration = dfExploration.drop('dateTime', 1)
dfExploration = dfExploration.drop('height', 1)
dfExploration.head(10)
```

| | monthYear | secondsBetweenBlocks |
|---|-----------|----------------------|
| 0 | 2021-05 | 0.00000 |
| 1 | 2021-05 | 38.00000 |
| 2 | 2021-05 | 11.00000 |
| 3 | 2021-05 | 27.00000 |
| 4 | 2021-05 | 202.00000 |
| 5 | 2021-05 | 3.00000 |
| 6 | 2021-05 | 12.00000 |
| 7 | 2021-05 | 32.00000 |
| 8 | 2021-05 | 44.00000 |
| 9 | 2021-05 | 32.00000 |

```
# seconds between blocks
dfExploration = dfExploration.groupby('monthYear').mean('secondsBetweenBlocks')
dfExploration
```

| secondsBetweenBlocks | |
|----------------------|-----------|
| monthYear | |
| 2018-01 | 45.78702 |
| 2018-02 | 54.31378 |
| 2018-03 | 56.14346 |
| 2018-04 | 61.29363 |
| 2018-05 | 59.86616 |
| 2018-06 | 60.45927 |
| 2018-07 | 63.21432 |
| 2018-08 | 134.28796 |
| 2018-09 | 60.41391 |
| 2018-10 | 60.29142 |
| 2018-11 | 60.40072 |
| 2018-12 | 60.41646 |
| 2019-01 | 60.41121 |
| 2019-02 | 60.38584 |



Verifica-se um comportamento anormal nos primeiros 8 meses a partir da criação da Ravencoin. Para evitar que isso afete negativamente a previsão do *halving*, somente serão utilizados dados posteriores a 01/09/2018 00:00:00.

```

cutoff_date = datetime(2018, 9, 1)
cutoff_date

datetime.datetime(2018, 9, 1, 0, 0)

cutoff_datetime = datetime.combine(cutoff_date, datetime.min.time())
cutoff_datetime

datetime.datetime(2018, 9, 1, 0, 0)

cutoff_timestamp = time.mktime(cutoff_datetime.timetuple())
cutoff_timestamp = int(cutoff_timestamp)
cutoff_timestamp

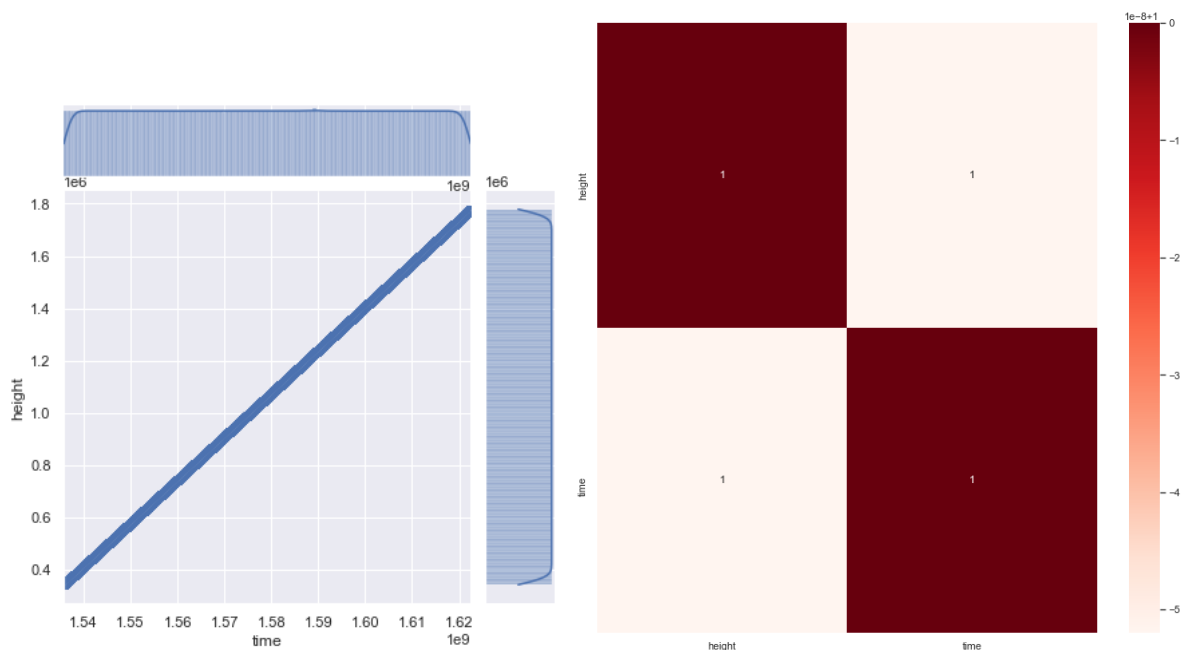
1535770800

df = df[df["time"] > cutoff_timestamp]
df

```

| | height | time |
|--------|--------|------------|
| 337717 | 341701 | 1535770808 |
| 337718 | 341702 | 1535770841 |
| 337719 | 341703 | 1535770887 |
| 337720 | 341704 | 1535770907 |
| 337721 | 341705 | 1535770972 |
| ... | ... | ... |

Nesse ponto devemos verificar a linearidade dos dados e a correlação.



Não identificamos nenhum problema de linearidade dos dados e verifica-se que existe uma correlação perfeita entre o número do bloco e o seu *timestamp*. Essa alta correlação já era esperada, uma vez que pudemos verificar que a média mensal do tempo entre blocos segue estável. Isso pode servir como indicador de que o *hashrate* (poder computacional) da rede de mineradores e que o grau dificuldade de mineração seguirem estáveis e equilibrados durante o período objeto de análise.

5. Criação de Modelos de Machine Learning

5.1 Objetivo Primário: Previsão do preço da RVN

Antes de trabalhar com os modelos, é necessário dividir a base de dados em treinamento e teste.

```
: # x = independent variable -> volume_RVN, adj close_BTC
# y = dependent variable -> adj close_RVN

# here we have 2 variables for multiple regression
X = df[['adj_close_BTC', 'volume_RVN']]
y = df['adj_close_RVN']

X_training, X_test, y_training, y_test = train_test_split( X, y,
                                                         test_size = 0.2, random_state = 20210713)

print(f"Training dataframe: {X_training.shape[0]} lines")
print(f"Test dataframe....: {X_test.shape[0]} lines")

Training dataframe: 937 lines
Test dataframe....: 235 lines

X_training2 = np.asarray(X_training)
X_test2 = np.asarray(X_test)
y_training2 = np.asarray(y_training)
y_test2 = np.asarray(y_test)
```

Na etapa de análise e exploração dos dados examinamos a linearidade dos dados de preço do BTC e volume da RVN em relação ao preço da RVN. Observamos que não há um ajuste perfeito e levantamos a hipótese de que a técnica de regressão polinomial poderia obter melhores do que uma regressão múltipla linear. Chegando o momento de verificar essa hipótese, começaremos pela linear e na sequência a polinomial.

Building the models:

1. Multiple Linear Regression With scikit-learn

Useful for getting the advanced statistical parameters of a model

```
X_training = sm.add_constant(X_training)
X_testOriginal = X_test
X_test = sm.add_constant(X_test)
```

```
# Create a model and fit it
model_SM = sm.OLS(y_training, X_training).fit()
type(model_SM)

# SM = statsmodels
```

statsmodels.regression.linear_model.RegressionResultsWrapper

2. Polynomial Regression With scikit-learn

```
# Data Preparation
# Include  $x^2$ 
poly = PolynomialFeatures(degree=2, include_bias=False)
X_training2_ = poly.fit_transform(X_training2)
X_test2_ = poly.fit_transform(X_test2)
```

```
# Define and train a model
model_POLY = LinearRegression().fit(X_training2_, y_training2)
type(model_POLY)
```

sklearn.linear_model._base.LinearRegression

5.2 Objetivo Secundário: Previsão de *halving* da RVN

Antes de partir para o modelo, é necessário segmentar a base de dados em treino e teste.

Defining training and testing dataframes

```
# x = independent variable -> height
# y = dependent variable -> time

# here we have 1 variable for linear regression
X = df['height']
y = df['time']

X_training, X_test, y_training, y_test = train_test_split( X, y,
                                                         test_size = 0.2, random_state = 20210713)

print(f"Training dataframe: {X_training.shape[0]} lines")
print(f"Test dataframe.....: {X_test.shape[0]} lines")

Training dataframe: 1149634 lines
Test dataframe.....: 287409 lines
```

Criação do modelo:

Linear Regression

```
X_training = sm.add_constant(X_training)
X_test = sm.add_constant(X_test)
```

```
# Create a model and fit it
model_SM = sm.OLS(y_training, X_training).fit()
type(model_SM)
```

```
# SM = statsmodels
```

```
statsmodels.regression.linear_model.RegressionResultsWrapper
```

6. Apresentação dos Resultados

6.1 Objetivo Primário: Previsão do preço da RVN

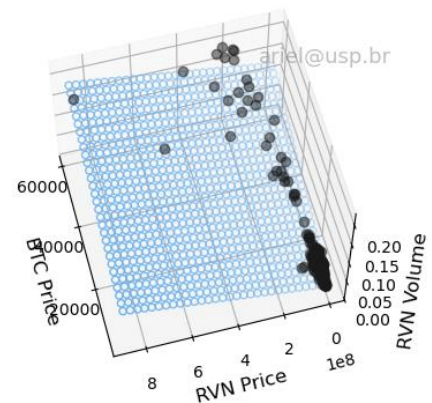
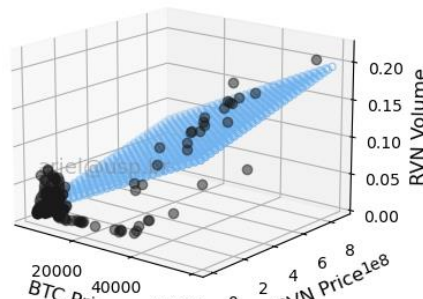
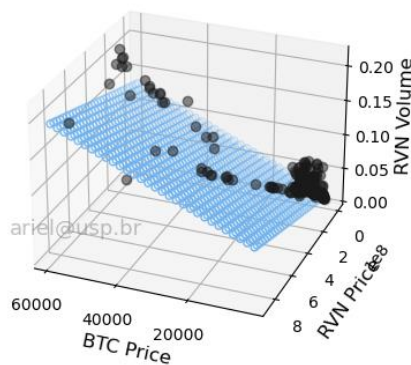
6.1.1 Regressão múltipla linear

```
model_SM.summary()
```

OLS Regression Results

| | | | | | | |
|-------------------|------------------|---------------------|-----------|-------|----------|----------|
| Dep. Variable: | adj_close_RVN | R-squared: | 0.710 | | | |
| Model: | OLS | Adj. R-squared: | 0.709 | | | |
| Method: | Least Squares | F-statistic: | 1143. | | | |
| Date: | Mon, 19 Jul 2021 | Prob (F-statistic): | 1.05e-251 | | | |
| Time: | 09:44:59 | Log-Likelihood: | 2184.2 | | | |
| No. Observations: | 937 | AIC: | -4362. | | | |
| Df Residuals: | 934 | BIC: | -4348. | | | |
| Df Model: | 2 | | | | | |
| Covariance Type: | nonrobust | | | | | |
| | coef | std err | t | P> t | [0.025 | 0.975] |
| const | 0.0084 | 0.001 | 7.701 | 0.000 | 0.006 | 0.011 |
| adj_close_BTC | 2.094e-06 | 6.5e-08 | 32.223 | 0.000 | 1.97e-06 | 2.22e-06 |
| volume_RVN | 7.044e-11 | 5.74e-12 | 12.268 | 0.000 | 5.92e-11 | 8.17e-11 |
| Omnibus: | 105.535 | Durbin-Watson: | 2.070 | | | |
| Prob(Omnibus): | 0.000 | Jarque-Bera (JB): | 268.541 | | | |
| Skew: | -0.609 | Prob(JB): | 4.87e-59 | | | |
| Kurtosis: | 5.323 | Cond. No. | 2.34e+08 | | | |

$$R^2 = 0.71$$



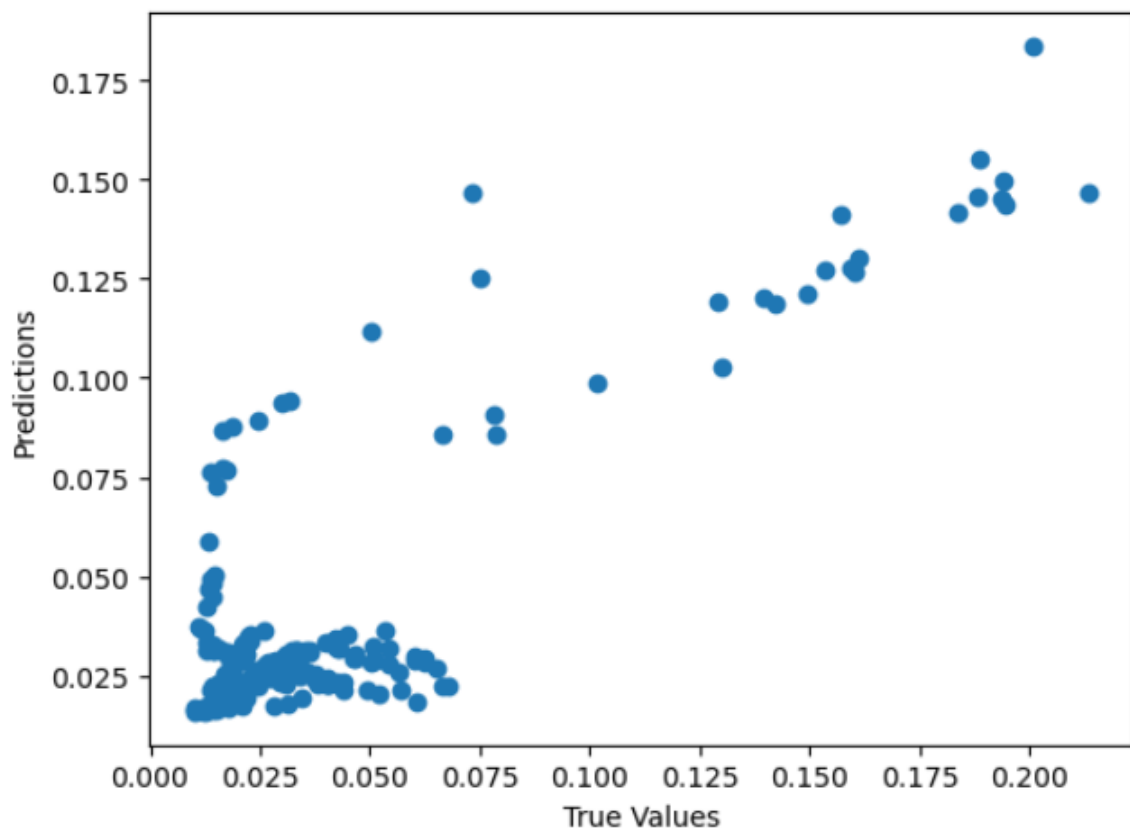
Evaluate

```
y_pred_SM = model_SM.predict(X_test)
print('predicted response:', y_pred_SM, sep='\n')
```

```
predicted response:
height_RVN
7070000  0.02768
7065000  0.08659
7180000  0.02139
7180000  0.02226
7165000  0.03163
...
7115000  0.03340
7185000  0.03317
7170000  0.03199
7060000  0.03075
7200000  0.02235
Length: 235, dtype: float64
```

```
## The Line / model
plt.scatter(y_test, y_pred_SM)
plt.xlabel("True Values")
plt.ylabel("Predictions")
```

```
Text(0, 0.5, 'Predictions')
```



Mean Square Error is an absolute measure of the goodness for the fit.

```
print('Mean Absolute Error(MAE): ', metrics.mean_absolute_error(y_test, y_pred_SM))
```

Mean Absolute Error(MAE): 0.01488225771514125

MSE gives an absolute number on how much your predicted results deviate from the actual number.

```
print('Mean Square Error(MSE): ', metrics.mean_squared_error(y_test, y_pred_SM))
```

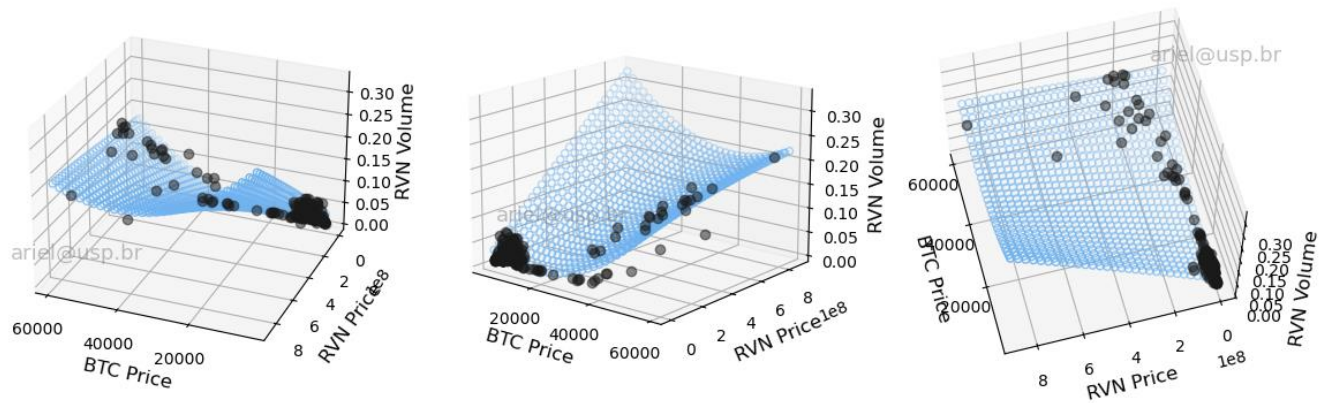
Mean Square Error(MSE): 0.0004892179886644842

```
print('Root Mean Square Error(RMSE): ', np.sqrt(metrics.mean_squared_error(y_test, y_pred_SM)))
```

Root Mean Square Error(RMSE): 0.022118272732392197

6.1.2 Regressão polinomial

$$R^2 = 0.85$$



Evaluate

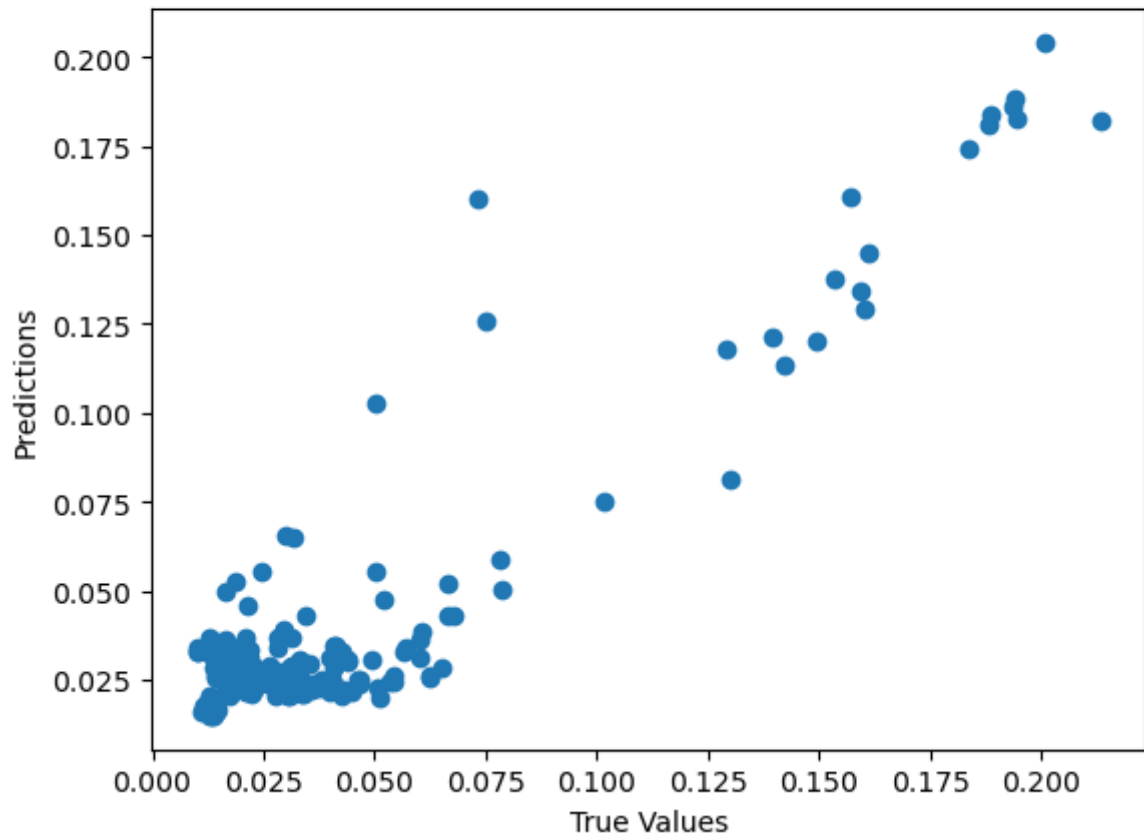
```
y_pred_POLY = model_POLY.predict(X_test2_)
print('predicted response:', y_pred_POLY, sep='\n')
```

predicted response:

```
[0.02593219 0.04989682 0.03072549 0.02566433 0.02061702 0.02859036
0.02820605 0.02569454 0.12606258 0.01596896 0.02468973 0.02385567
0.02381541 0.0247157 0.02577003 0.02519552 0.02470226 0.16052244
0.18228374 0.0266356 0.03297307 0.02583483 0.02242317 0.03441254
0.02612963 0.03156033 0.05261504 0.02395338 0.02613168 0.02229392
0.02609761 0.0279123 0.01837953 0.03607831 0.03224137 0.03365607
0.03240492 0.02103964 0.02675562 0.02644225 0.03402844 0.022944
0.02916831 0.03020356 0.04768952 0.02068638 0.02483165 0.02332719
0.05557363 0.02505205 0.02309836 0.0261766 0.02876328 0.03272287
0.02013516 0.02881199 0.07506079 0.02430705 0.03694287 0.01525676
0.13768454 0.03353884 0.04309475 0.02377019 0.121027 0.02652336
0.02412752 0.12007265 0.03047279 0.02005948 0.18361903 0.02697017
0.02139918 0.02214614 0.01503443 0.02060607 0.02160654 0.02278471
0.02847116 0.0277912 0.02220452 0.13403495 0.03303253 0.03377948
0.02283978 0.03607494 0.02544314 0.02361687 0.01953681 0.02596385
0.03709482 0.02457555 0.02979259 0.03412957 0.08146193 0.02517458
0.02657398 0.02568299 0.02067486 0.02534135 0.02444788 0.12912345
0.02443346 0.0256931 0.02324901 0.02551802 0.0278188 0.02511954
0.02552629 0.04605991 0.03112279 0.0250186 0.02341481 0.02751697
0.02289014 0.02477165 0.10280508 0.02564199 0.03486439 0.0326996
0.11350686 0.18613778 0.03007879 0.02644174 0.02280834 0.02533654
0.02543204 0.03461992 0.03316321 0.02554241 0.02067427 0.05553736
0.0306266 0.03363377 0.01960614 0.02491138 0.02471808 0.02838151
0.02174172 0.20411325 0.02554669 0.02456756 0.02617493 0.18111279
0.02707027 0.01668038 0.03244872 0.03134992 0.15994713 0.02666963
0.03346202 0.03003043 0.02877488 0.02868347 0.02382345 0.0328526
0.02210498 0.02185258 0.0203001 0.11813726 0.02598322 0.03366436
0.01493996 0.02479614 0.02629521 0.01616026 0.02589482 0.18839212
0.02634366 0.02552578 0.1826076 0.06571828 0.02409518 0.03376265
0.02817896 0.03417584 0.02522854 0.02666432 0.02223027 0.03691928
0.02238704 0.02536644 0.02448797 0.02368706 0.03378784 0.02532101
0.04277063 0.05233174 0.02488826 0.02519189 0.03335117 0.02349834
0.01505159 0.0253987 0.03586346 0.05885414 0.05044575 0.03376265
0.02476625 0.02040419 0.02207256 0.02084394 0.0385512 0.03307813
0.03342429 0.02428286 0.026111 0.03259739 0.17445745 0.03127511
0.04313103 0.03064181 0.03449638 0.0261361 0.01587153 0.02653831
0.03026745 0.0274224 0.03912005 0.02618402 0.020839 0.0295644
0.01856868 0.01751293 0.02740408 0.1447234 0.06492811 0.0366384
0.03401324 0.02817024 0.01875746 0.0217096 0.02473844 0.02180861
0.02544977]
```

```
## The Line / model
plt.scatter(y_test2, y_pred_POLY)
plt.xlabel("True Values")
plt.ylabel("Predictions")
```

```
Text(0, 0.5, 'Predictions')
```



Get results

```
print('Coefficient of Determination ( $R^2$ ):', model_POLY.score(X_training2_, y_training2))
```

```
Coefficient of Determination ( $R^2$ ): 0.8538252108360737
```

```
print('Mean Absolute Error(MAE): ', metrics.mean_absolute_error(y_test2, y_pred_POLY))
```

```
Mean Absolute Error(MAE): 0.01264981664892703
```

```
print('MSE: ', metrics.mean_squared_error(y_test2, y_pred_POLY))
```

```
MSE: 0.0002821002166365771
```

```
print('RMSE: ', np.sqrt(metrics.mean_squared_error(y_test2, y_pred_POLY)))
```

```
RMSE: 0.01679583926562103
```

6.1.3 Comparação de Resultados:

Comparação dos resultados obtidos ao aplicar diferentes modelos regressão sobre os mesmos dados:

| | Linear | Polinomial |
|---|-------------|-------------|
| MAE (Mean Absolute Error) | 0,014882258 | 0,012649817 |
| MSE (Mean Squared Error) | 0,000489218 | 0,0002821 |
| RMSE (Root Mean Squared Error) | 0,022118273 | 0,016795839 |
| R-squared (Coefficient of Determination) | 0,71 | 0,853825211 |

A tabela acima apresenta as métricas de avaliação tipicamente aplicadas a modelos de aprendizado de máquina baseado em regressão. Podemos observar que a regressão polinomial obteve resultados melhores em todos os itens.

Visualmente também é possível notar que há um melhor ajuste da reta de regressão polinomial aos dados.

$$R^2 = 0.71$$

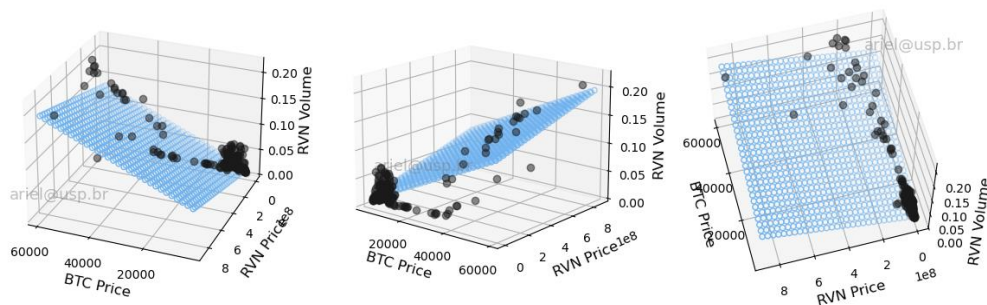


Figura 5 - Regressão Linear

$$R^2 = 0.85$$

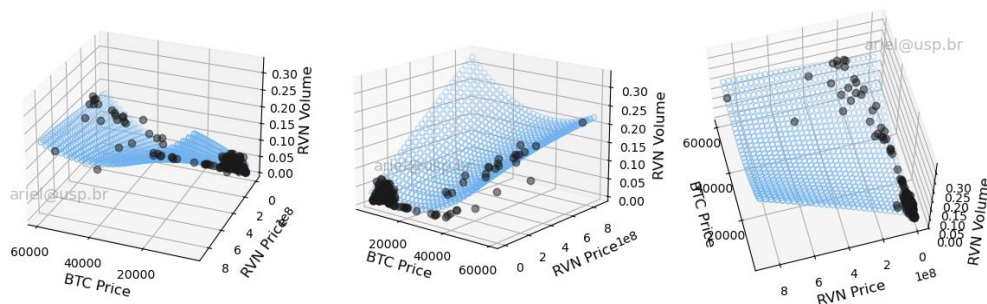


Figura 6 - Regressão Polinomial

Vale comentar que foram criados gráficos 3D animados para ambos os modelos (linear e polinomial), formato GIF (*Graphics Interchange Format*). Rotacionar a imagem 360° facilita a observação do ajuste da reta de regressão aos

pontos. Para visualizar, basta abrir o script do Jupyter Notebook '4 - Analysis, Exploration and Model - RVN Blocks'.

6.1.4 Análise dos Erros de Previsão da Regressão Polinomial

Antes de se posicionar quanto aos resultados obtidos é importante analisar o erro de previsão. É necessário entender como o erro se dá na prática para que seja possível decidir pela aplicação ou não de um determinado modelo. As métricas de acurácia indicam que temos um bom modelo, mas será que o erro é aceitável para o objetivo a que se destina? No mundo real, esse modelo atenderia aos requisitos/necessidades do negócio?

Faremos a análise dos erros apenas do modelo de regressão polinomial, uma vez que esta foi a técnica que obteve os melhores resultados nos indicadores de acurácia.

Iniciamos criando um *dataframe* que mantém o valor correto, o valor previsto e o erro. O valor correto origina-se dos dados destinados a teste (*dataset* original foi dividido em treino e teste), o valor previsto vem da aplicação do modelo previamente treinado aos dados de teste e o erro corresponde a mera subtração entre eles

About the results

```
dfResults = pd.DataFrame()
dfResults['adj_close_RVN'] = y_test
dfResults['prediction'] = y_pred_POLY
dfResults['error'] = dfResults['adj_close_RVN'] - dfResults['prediction']
dfResults.sort_index(inplace=True)
dfResults.reset_index(inplace=True)
dfResults
```

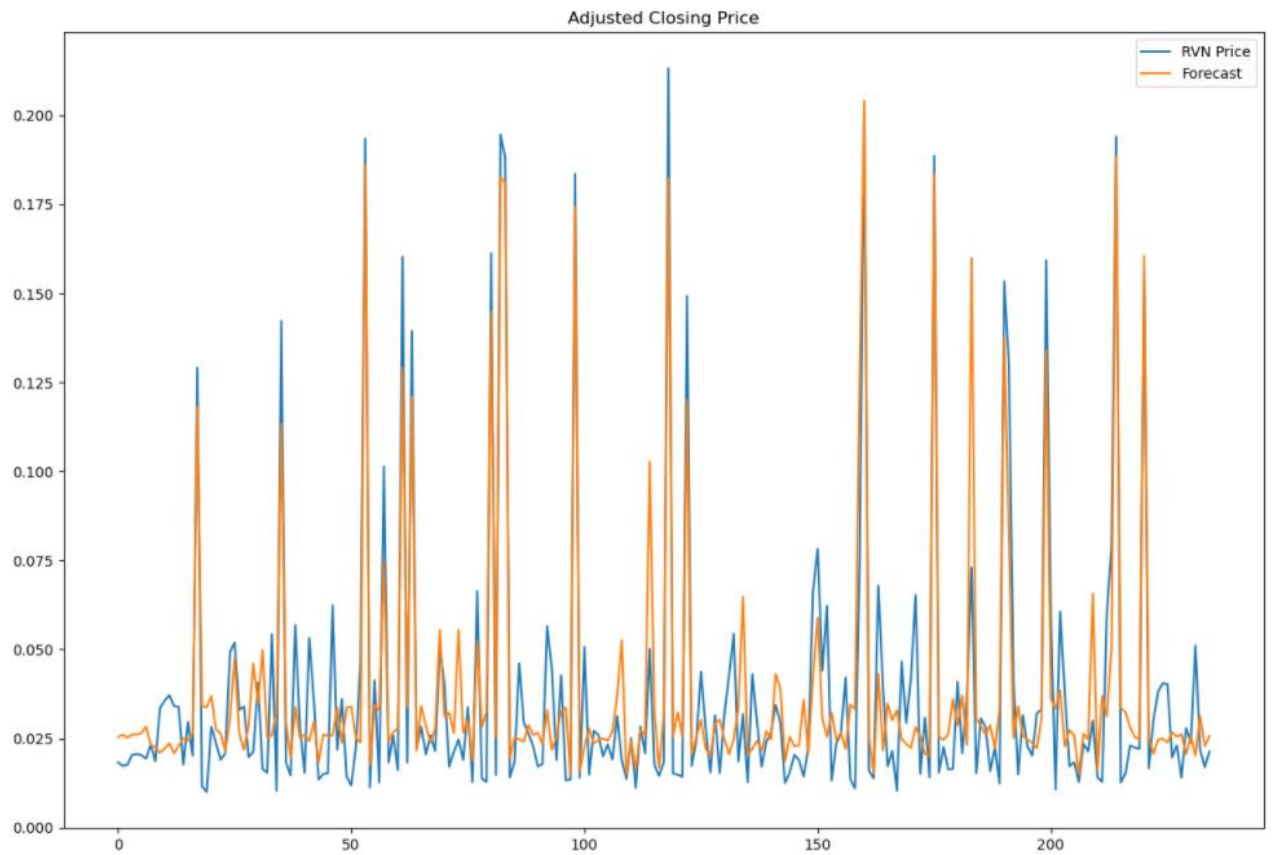
| | height_RVN | adj_close_RVN | prediction | error |
|-----|------------|---------------|------------|----------|
| 0 | 1125000 | 0.01832 | 0.02537 | -0.00705 |
| 1 | 1265000 | 0.01734 | 0.02610 | -0.00875 |
| 2 | 1330000 | 0.01761 | 0.02519 | -0.00759 |
| 3 | 1380000 | 0.02041 | 0.02614 | -0.00572 |
| 4 | 1395000 | 0.02066 | 0.02611 | -0.00545 |
| ... | ... | ... | ... | ... |
| 230 | 7715000 | 0.02376 | 0.02564 | -0.00188 |
| 231 | 8470000 | 0.05112 | 0.02014 | 0.03098 |
| 232 | 9500000 | 0.02191 | 0.03135 | -0.00944 |
| 233 | 10740000 | 0.01696 | 0.02289 | -0.00593 |
| 234 | 11265000 | 0.02133 | 0.02569 | -0.00436 |

235 rows x 4 columns

Abaixo temos um gráfico que apresenta de forma sobreposta o preço correto da RVN e o preço previsto.

```
dfResults['adj_close_RVN'].plot(label='RVN Price', figsize=(15,10), title='Adjusted Closing Price')
dfResults['prediction'].plot(label='Forecast')
plt.legend()
```

<matplotlib.legend.Legend at 0x1f80f994c70>



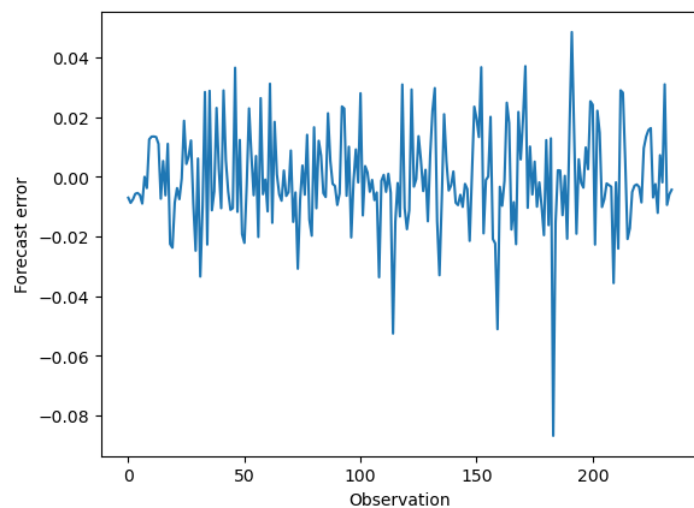
Abaixo temos a visualização do comportamento do erro no gráfico.

```
dfError = pd.DataFrame()
dfError['error'] = dfResults['error']
dfError.sort_index(inplace=True)
dfError
```

| | error |
|-----|----------|
| 0 | -0.00705 |
| 1 | -0.00875 |
| 2 | -0.00759 |
| 3 | -0.00572 |
| 4 | -0.00545 |
| ... | ... |
| 230 | -0.00188 |
| 231 | 0.03098 |
| 232 | -0.00944 |
| 233 | -0.00593 |
| 234 | -0.00436 |

235 rows x 1 columns

```
plt.plot(dfError)
plt.xlabel("Observation")
plt.ylabel("Forecast error")
plt.show()
```



Para ficar ainda mais claro, faremos a distribuição do erro em intervalo de classes.

```
dfErrorAnalysis = dfError['error'].value_counts(bins = 7, sort=False)

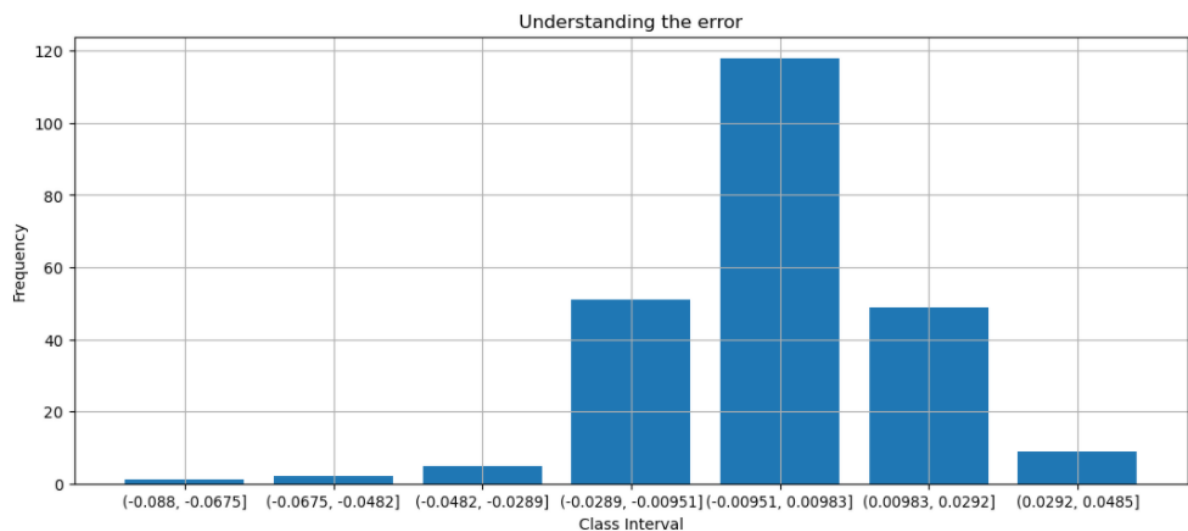
dfErrorAnalysis = dfErrorAnalysis.reset_index()
dfErrorAnalysis.rename(columns={'index':'Class_Interval'}, inplace=True)

dfErrorAnalysis.rename(columns={'error':'Frequency'}, inplace=True)

dfErrorAnalysis['Relative_Frequency'] = dfErrorAnalysis['Frequency']/dfErrorAnalysis['Frequency'].sum()
dfErrorAnalysis
```

| | Class_Interval | Frequency | Relative_Frequency |
|---|---------------------|-----------|--------------------|
| 0 | (-0.088, -0.0675] | 1 | 0.00426 |
| 1 | (-0.0675, -0.0482] | 2 | 0.00851 |
| 2 | (-0.0482, -0.0289] | 5 | 0.02128 |
| 3 | (-0.0289, -0.00951] | 51 | 0.21702 |
| 4 | (-0.00951, 0.00983] | 118 | 0.50213 |
| 5 | (0.00983, 0.0292] | 49 | 0.20851 |
| 6 | (0.0292, 0.0485] | 9 | 0.03830 |

```
fig = plt.figure(figsize=(10, 4))
ax = fig.add_axes([0,0,1,1])
ax.bar(dfErrorAnalysis['Class_Interval'].astype(str), dfErrorAnalysis['Frequency'])
plt.xticks(size = 9.5)
plt.grid(True)
plt.title('Understanding the error')
plt.xlabel('Class Interval')
plt.ylabel('Frequency')
plt.show()
```



Por fim vale verificar qual foi o valor de erro máximo, mínimo e médio. Também é importante verificar qual foi o último valor da cotação da Ravencoin disponível no dataset.

```
print('Highest error value: ', dfError['error'].max())
```

```
Highest error value: 0.048508072671860414
```

```
print('Lowest error value: ', dfError['error'].min())
```

```
Lowest error value: -0.08687013119396347
```

```
print('Average error: ', dfError['error_module'].mean())
```

```
Average error: 0.012649816648927033
```

```
print('RVNs latest closing quote price available in the dataset: ', df['adj_close_RVN'].iloc[-1])
```

```
RVNs latest closing quote price available in the dataset: 0.084311
```

6.1.5 Conclusão sobre a aplicabilidade do modelo de Regressão Polinomial Múltipla para previsão do preço da RVN

O uso da regressão polinomial para prever o preço do Ravencoin a partir do Preço do Bitcoin e do Volume do Ravencoin cumpre o objetivo didático de demonstrar a correta aplicação das técnicas aprendidas durante o curso. O modelo obteve bons resultados nas métricas de avaliação tipicamente aplicadas a modelos de aprendizado de máquina baseado em regressão.

No entanto, a depender do(s) requisito(s) de negócio pode não ser adequado utilizar um modelo com um erro médio de previsão da ordem de 1,2649 centavo de dólar. Devemos observar que a última cotação disponível para este ativo corresponde a 8,4311 centavos de dólar.

Recomenda-se para trabalhos futuros estudar a inclusão de novas variáveis a fim de aumentar a capacidade preditiva do modelo e/ou testar outros algoritmos preditivos.

6.2 Objetivo Secundário: Previsão de halving da RVN

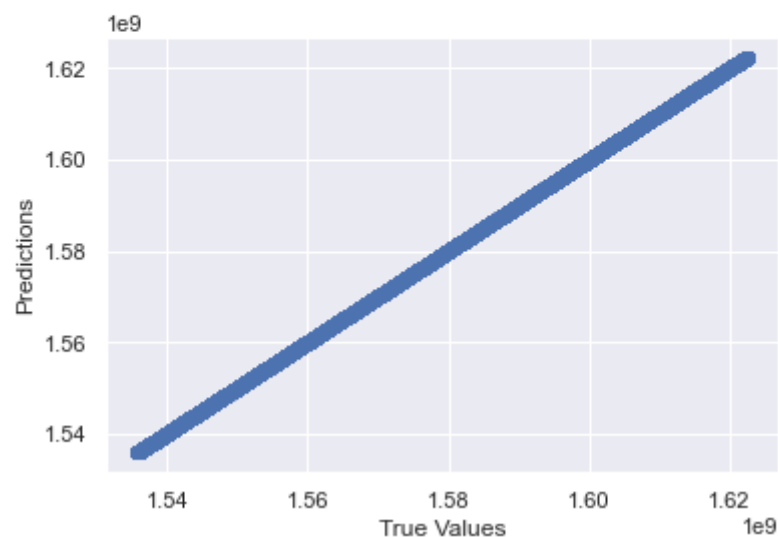
Evaluate

```
y_pred_SM = model_SM.predict(X_test)
print('predicted response:', y_pred_SM, sep='\n')
```

```
predicted response:
555866      1548945551.26831
821299      1564965866.96886
1572819     1610325386.93980
1427873     1601576454.34492
1133921     1583834272.32448
...
606496      1552001443.01035
1312800     1594630927.60509
1542586     1608500494.72317
623655      1553037073.56814
1676371     1616575383.25549
Length: 287409, dtype: float64
```

```
## The line / model
plt.scatter(y_test, y_pred_SM)
plt.xlabel("True Values")
plt.ylabel("Predictions")
```

```
Text(0, 0.5, 'Predictions')
```



```
model_SM.summary()
```

OLS Regression Results

| | | | |
|-------------------|------------------|---------------------|-------------|
| Dep. Variable: | time | R-squared: | 1.000 |
| Model: | OLS | Adj. R-squared: | 1.000 |
| Method: | Least Squares | F-statistic: | 1.105e+13 |
| Date: | Mon, 19 Jul 2021 | Prob (F-statistic): | 0.00 |
| Time: | 09:48:58 | Log-Likelihood: | -1.1974e+07 |
| No. Observations: | 1149634 | AIC: | 2.395e+07 |
| Df Residuals: | 1149632 | BIC: | 2.395e+07 |
| Df Model: | 1 | | |
| Covariance Type: | nonrobust | | |

| | coef | std err | t | P> t | [0.025 | 0.975] |
|--------|-----------|----------|----------|-------|----------|----------|
| const | 1.515e+09 | 20.672 | 7.33e+07 | 0.000 | 1.52e+09 | 1.52e+09 |
| height | 60.3549 | 1.82e-05 | 3.32e+06 | 0.000 | 60.355 | 60.355 |

| | | | |
|----------------|------------|-------------------|-----------|
| Omnibus: | 101837.268 | Durbin-Watson: | 2.000 |
| Prob(Omnibus): | 0.000 | Jarque-Bera (JB): | 66204.888 |
| Skew: | 0.469 | Prob(JB): | 0.00 |
| Kurtosis: | 2.290 | Cond. No. | 3.12e+06 |

Observamos que *R-squared* nos informa que 100% da variância da variável dependente (*timestamp*) é determinada pela variável independente (número do bloco). O gráfico de predições vs valores verdadeiros também nos indica um excelente resultado.

```
print('Mean Absolute Error(MAE): ', metrics.mean_absolute_error(y_test, y_pred_SM))
```

```
Mean Absolute Error(MAE): 6525.262966041408
```

```
print('Mean Square Error(MSE): ', metrics.mean_squared_error(y_test, y_pred_SM))
```

```
Mean Square Error(MSE): 65073053.70010137
```

```
print('Root Mean Square Error(RMSE): ', np.sqrt(metrics.mean_squared_error(y_test, y_pred_SM)))
```

```
Root Mean Square Error(RMSE): 8066.787074176519
```

Porém MAE, MSE e RMSE indicam que o modelo é péssimo, é necessário analisar o erro para entender o que está acontecendo.

6.2.1 Análise dos Erros de Previsão de Halving

```
dfResults = pd.DataFrame()
dfResults = pd.concat([y_test, y_pred_SM], axis=1)
dfResults.rename(columns={0:"prediction"}, inplace=True)
dfResults
```

| | time | prediction |
|---------|------------|------------------|
| 555866 | 1548943376 | 1548945551.26831 |
| 821299 | 1564970602 | 1564965866.96886 |
| 1572819 | 1610326237 | 1610325386.93980 |
| 1427873 | 1601571728 | 1601576454.34492 |
| 1133921 | 1583846688 | 1583834272.32448 |
| ... | ... | ... |
| 606496 | 1551997975 | 1552001443.01035 |
| 1312800 | 1594620203 | 1594630927.60509 |
| 1542586 | 1608500160 | 1608500494.72317 |
| 623655 | 1553035136 | 1553037073.56814 |
| 1676371 | 1616570133 | 1616575383.25549 |

287409 rows x 2 columns

Para que seja possível interpretar os dados precisamos converter para o formato data e hora.

```
# Handling the dates
dfResults.sort_index(inplace=True)

dfResults['dateTime'] = dfResults['time']
dfResults['dateTime'] = dfResults['dateTime'].apply(timestampToDateTime)

dfResults['dateTime_pred'] = dfResults['prediction']
dfResults['dateTime_pred'] = dfResults['dateTime_pred'].apply(timestampToDateTime)

dfResults
```

| | time | prediction | dateTime | dateTime_pred |
|---------|------------|------------------|---------------------|----------------------------|
| 337718 | 1535770841 | 1535778998.58797 | 2018-09-01 00:00:41 | 2018-09-01 02:16:38.587965 |
| 337720 | 1535770907 | 1535779119.29786 | 2018-09-01 00:01:47 | 2018-09-01 02:18:39.297862 |
| 337725 | 1535771192 | 1535779421.07260 | 2018-09-01 00:06:32 | 2018-09-01 02:23:41.072603 |
| 337726 | 1535771219 | 1535779481.42755 | 2018-09-01 00:06:59 | 2018-09-01 02:24:41.427551 |
| 337727 | 1535771479 | 1535779541.78250 | 2018-09-01 00:11:19 | 2018-09-01 02:25:41.782500 |
| ... | ... | ... | ... | ... |
| 1774728 | 1622514088 | 1622511956.32468 | 2021-05-31 23:21:28 | 2021-05-31 22:45:56.324683 |
| 1774730 | 1622514365 | 1622512077.03458 | 2021-05-31 23:26:05 | 2021-05-31 22:47:57.034580 |
| 1774732 | 1622514596 | 1622512197.74448 | 2021-05-31 23:29:56 | 2021-05-31 22:49:57.744476 |
| 1774736 | 1622514853 | 1622512439.16427 | 2021-05-31 23:34:13 | 2021-05-31 22:53:59.164269 |
| 1774738 | 1622515092 | 1622512559.87417 | 2021-05-31 23:38:12 | 2021-05-31 22:55:59.874166 |

287409 rows x 4 columns

Agora vamos determinar qual é o erro de previsão, em uma escala de minutos. Assim saberemos quão útil pode ser uma previsão obtida com este modelo.

```
dfResults['error'] = (dfResults['dateTime'] - dfResults['dateTime_pred']).astype('timedelta64[m]')
dfResults['error'] = dfResults['error'].fillna(0)
dfResults
```

| | time | prediction | dateTime | dateTime_pred | error |
|---------|------------|------------------|---------------------|----------------------------|------------|
| 337718 | 1535770841 | 1535778998.58797 | 2018-09-01 00:00:41 | 2018-09-01 02:16:38.587965 | -136.00000 |
| 337720 | 1535770907 | 1535779119.29786 | 2018-09-01 00:01:47 | 2018-09-01 02:18:39.297862 | -137.00000 |
| 337725 | 1535771192 | 1535779421.07260 | 2018-09-01 00:06:32 | 2018-09-01 02:23:41.072603 | -138.00000 |
| 337726 | 1535771219 | 1535779481.42755 | 2018-09-01 00:06:59 | 2018-09-01 02:24:41.427551 | -138.00000 |
| 337727 | 1535771479 | 1535779541.78250 | 2018-09-01 00:11:19 | 2018-09-01 02:25:41.782500 | -135.00000 |
| ... | ... | ... | ... | ... | ... |
| 1774728 | 1622514088 | 1622511956.32468 | 2021-05-31 23:21:28 | 2021-05-31 22:45:56.324683 | 35.00000 |
| 1774730 | 1622514365 | 1622512077.03458 | 2021-05-31 23:26:05 | 2021-05-31 22:47:57.034580 | 38.00000 |
| 1774732 | 1622514596 | 1622512197.74448 | 2021-05-31 23:29:56 | 2021-05-31 22:49:57.744476 | 39.00000 |
| 1774736 | 1622514853 | 1622512439.16427 | 2021-05-31 23:34:13 | 2021-05-31 22:53:59.164269 | 40.00000 |
| 1774738 | 1622515092 | 1622512559.87417 | 2021-05-31 23:38:12 | 2021-05-31 22:55:59.874166 | 42.00000 |

287409 rows x 5 columns

Vamos criar um gráfico que nos mostre o comportamento do erro.

```
dfError = pd.DataFrame()
dfError['error_min'] = dfResults['error'].astype(int)

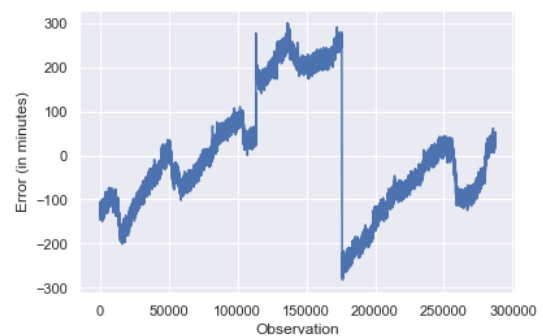
dfError.sort_index(inplace=True)

dfError.reset_index(inplace=True)
dfError = dfError.drop('index', 1)
dfError
```

| | error_min |
|--------|-----------|
| 0 | -136 |
| 1 | -137 |
| 2 | -138 |
| 3 | -138 |
| 4 | -135 |
| ... | ... |
| 287404 | 35 |
| 287405 | 38 |
| 287406 | 39 |
| 287407 | 40 |
| 287408 | 42 |

287409 rows x 1 columns

```
plt.plot(dfError)
plt.xlabel("Observation")
plt.ylabel("Error (in minutes)")
plt.show()
```



```
dfError['error_min_module'] = dfError['error_min'].abs()
dfError
```

| | error_min | error_min_module |
|--------|-----------|------------------|
| 0 | -136 | 136 |
| 1 | -137 | 137 |
| 2 | -138 | 138 |
| 3 | -138 | 138 |
| 4 | -135 | 135 |
| ... | ... | ... |
| 287404 | 35 | 35 |
| 287405 | 38 | 38 |
| 287406 | 39 | 39 |
| 287407 | 40 | 40 |
| 287408 | 42 | 42 |

287409 rows x 2 columns

```
def toHourMin(minutes):
    minutes = abs(minutes)

    h = 0
    m = 0

    if (minutes % 60 >= 0):
        h = int(minutes/60)
        m = int(minutes % 60)
    else:
        m = minutes

    return str(h)+'h '+str(m)+'m'
```

Converter minutos em horas para facilitar ainda mais o entedimento:

```
print('Highest error value: ', dfError['error_min'].max())
toHourMin(dfError['error_min'].max())
```

Highest error value: 301

'5h 1m'

```
print('Lowest error value: ', dfError['error_min'].min())
toHourMin(dfError['error_min'].min())
```

Lowest error value: -283

'4h 43m'

```
print('Average error: ', dfError['error_min_module'].mean())
toHourMin(dfError['error_min_module'].mean())
```

Average error: 108.8352626396529

'1h 48m'

Distribuição do erro em intervalo de classes:

```
dfErrorAnalysis = dfError['error_min'].value_counts(bins = 7, sort=False)

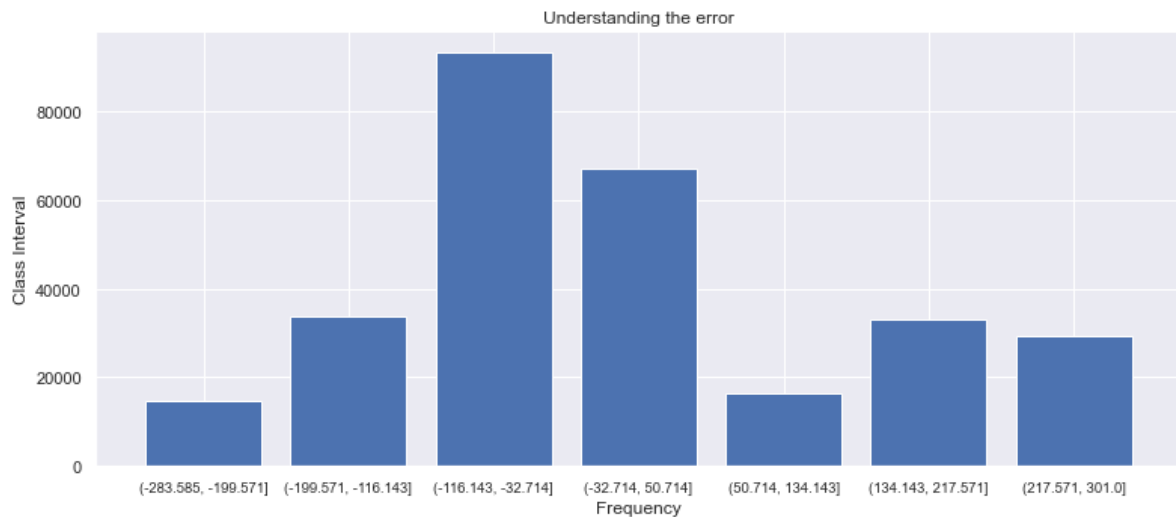
dfErrorAnalysis = dfErrorAnalysis.reset_index()
dfErrorAnalysis.rename(columns={'index': 'Class_Interval'}, inplace=True)

dfErrorAnalysis.rename(columns={'error_min': 'Frequency'}, inplace=True)

dfErrorAnalysis['Relative_Frequency'] = dfErrorAnalysis['Frequency']/dfErrorAnalysis['Frequency'].sum()
dfErrorAnalysis
```

| | Class_Interval | Frequency | Relative_Frequency |
|---|----------------------|-----------|--------------------|
| 0 | (-283.585, -199.571] | 14584 | 0.05074 |
| 1 | (-199.571, -116.143] | 33783 | 0.11754 |
| 2 | (-116.143, -32.714] | 93234 | 0.32439 |
| 3 | (-32.714, 50.714] | 66962 | 0.23299 |
| 4 | (50.714, 134.143] | 16423 | 0.05714 |
| 5 | (134.143, 217.571] | 33211 | 0.11555 |
| 6 | (217.571, 301.0] | 29212 | 0.10164 |

```
fig = plt.figure(figsize=(10, 4))
ax = fig.add_axes([0,0,1,1])
ax.bar(dfErrorAnalysis['Class_Interval'].astype(str), dfErrorAnalysis['Frequency'])
plt.xticks(size = 9.5)
plt.grid(True)
plt.title('Understanding the error')
plt.xlabel('Frequency')
plt.ylabel('Class Interval')
plt.show()
```



6.2.2 Conclusão sobre a aplicabilidade do modelo de Regressão Linear Simples para previsão de *halving*

Apesar dos indicadores MAE, MSE e RMSE indicarem que o modelo não é adequado, ao analisar o erro verificamos que:

- O erro é mensurável em uma escala de horas. De partida isso já parece ser algo bom.
- O erro máximo identificado é de cerca de 5 horas, algo que provavelmente já seria aceitável na maioria dos casos de uso da vida real.
- Ao analisar a distribuição de frequência com que o erro ocorre, verificamos que a moda estatística está presente em um intervalo de cerca de 85 minutos de defasagem em relação ao valor correto.

Intervalo: (-116,142, -32,714].

Assim, mesmo com a maioria dos indicadores apontando para a inadequação do modelo, a análise detalhada da manifestação do erro indica que o modelo pode sim ser útil para a proposta de identificar o mês de ocorrência do evento. É preciso mais cautela em relação a uma previsão do dia, mas esse tipo de previsão também pode ser feita.

Esse modelo foi criado como um desafio adicional que deveria empregar apenas os dados já disponíveis, apesar disso obtivemos bons resultados. Mas em trabalhos futuros recomenda-se incluir dados sobre o poder computacional aplicado pelos mineradores e o grau de dificuldade de mineração. Estes dados estiveram indiretamente representados pelo intervalo de tempo entre a criação de blocos, mas o ideal é trabalhar neles diretamente.

6.2.3 Previsão dos próximos *halvings*

Vamos aplicar o modelo treinado para prever os próximos 4 *halvings*, assim podemos criar um gráfico que facilita a compreensão dos resultados. Vide script '5 - Extra - Halving Projection'

Criando o dataset com o número dos blocos para posterior predição do *timestamp*.

```
blocks2HalvingRVN = 2100000
blocks2nextHalvingsRVN = [blocks2HalvingRVN, blocks2HalvingRVN*2, blocks2HalvingRVN*3, blocks2HalvingRVN*4]
new_x = np.array(blocks2nextHalvingsRVN).reshape((-1, 1))
new_x = sm.add_constant(new_x)
new_x

array([[1.0e+00, 2.1e+06],
       [1.0e+00, 4.2e+06],
       [1.0e+00, 6.3e+06],
       [1.0e+00, 8.4e+06]])
```

Realizando as previsões a partir do modelo previamente treinado

```
new_y = model_SM.predict(new_x)
print('predicted response:', new_y, sep='\n')

predicted response:
[1.64190098e+09 1.76864637e+09 1.89539177e+09 2.02213716e+09]
```

Convertendo em *dataframe* pandas

```
dfHalvingProjection = pd.DataFrame(new_y)
dfHalvingProjection.columns = ['predicted_timestamp']
dfHalvingProjection
```

| | predicted_timestamp |
|---|---------------------|
| 0 | 1641900983.46429 |
| 1 | 1768646374.88275 |
| 2 | 1895391766.30120 |
| 3 | 2022137157.71966 |

Transformando o *timestamp* em data

```
# Making timestamp humanly friendly
# For each timestamp a new block is generated, and each block corresponds to 5,000 new coins.

dfHalvingProjection['predicted_date'] = dfHalvingProjection['predicted_timestamp']
dfHalvingProjection['predicted_date'] = dfHalvingProjection['predicted_timestamp'].apply(timestampToDate)

dfHalvingProjection['predicted_datetime'] = dfHalvingProjection['predicted_timestamp']
dfHalvingProjection['predicted_datetime'] = dfHalvingProjection['predicted_timestamp'].apply(timestampToDateTime)
dfHalvingProjection
```

| | predicted_timestamp | predicted_date | predicted_datetime |
|---|---------------------|----------------|----------------------------|
| 0 | 1641900983.46429 | 2022-01-11 | 2022-01-11 08:36:23.464292 |
| 1 | 1768646374.88275 | 2026-01-17 | 2026-01-17 07:39:34.882746 |
| 2 | 1895391766.30120 | 2030-01-23 | 2030-01-23 06:42:46.301201 |
| 3 | 2022137157.71966 | 2034-01-29 | 2034-01-29 05:45:57.719656 |

Criando o *dataset* que será utilizado para gerar o gráfico de previsão dos próximos *halvings*.

```
dfPlot = pd.DataFrame()
dfPlot['Forecast'] = dfHalvingProjection['predicted_date']
dfPlot
```

| | Forecast |
|---|------------|
| 0 | 2022-01-11 |
| 1 | 2026-01-17 |
| 2 | 2030-01-23 |
| 3 | 2034-01-29 |

Adicionando a data em que o primeiro bloco de RVN foi minerado.

```
# Adding date of first mined block
dfPlot.loc[4] = [datetime.strptime('10/03/2018', '%d/%m/%Y').date()]
dfPlot.sort_values(by=['Forecast'], inplace=True)

dfPlot.reset_index(inplace=True)
dfPlot = dfPlot.drop('index', 1)

dfPlot.reset_index(inplace=True)
dfPlot.rename(columns={'index': 'Halving'}, inplace=True)
dfPlot
```

| | Halving | Forecast |
|---|---------|------------|
| 0 | 0 | 2018-03-10 |
| 1 | 1 | 2022-01-11 |
| 2 | 2 | 2026-01-17 |
| 3 | 3 | 2030-01-23 |
| 4 | 4 | 2034-01-29 |

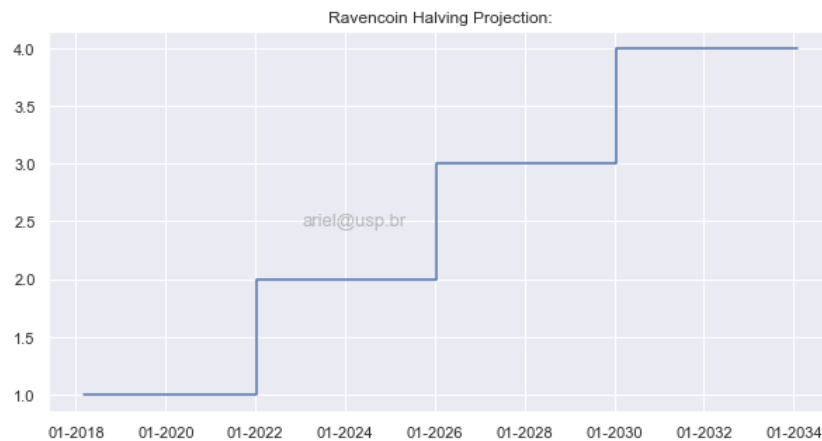
```
#The first halving only happen in 2022
dfPlot.at[0, 'Halving'] = 1
dfPlot
```

| | Halving | Forecast |
|---|---------|------------|
| 0 | 1 | 2018-03-10 |
| 1 | 1 | 2022-01-11 |
| 2 | 2 | 2026-01-17 |
| 3 | 3 | 2030-01-23 |
| 4 | 4 | 2034-01-29 |

```

from matplotlib.dates import DateFormatter
fig, ax = plt.subplots(figsize=(10, 5))
date_form = DateFormatter("%m-%Y")
ax.xaxis.set_major_formatter(date_form)
ax.text(0.39, 0.5, 'ariel@usp.br', fontsize=13, ha='center', va='center',
        transform=ax.transAxes, color='grey', alpha=0.5)
plt.title('Ravencoin Halving Projection:')
plt.step(dfPlot['Forecast'], dfPlot['Halving'])
plt.show()

```



A validade da projeção apresentada no gráfico depende da manutenção de um equilíbrio entre o poder de mineração alocado pelos mineradores e o ajuste de dificuldade. Mantidas as condições atuais, o primeiro *halving* deve ocorrer em 11 de janeiro de 2022.

Uma indicação de que a projeção provavelmente está correta é que o site oficial da Ravencoin informa que o primeiro *halving* está previsto para janeiro de 2022. Fonte: <https://ravencoin.org/halving/>

É muito cedo para fazer uma previsão assertiva dos *halvings* subsequentes, a tentativa de previsão foi feita apenas para gerar um gráfico com o objetivo de facilitar a compreensão dos resultados.

6.3 Conclusão Final

Os objetivos (primário e secundário) foram atingidos e foram cumpridas todas as etapas necessárias para o adequado desenvolvimento de um projeto de Ciência de Dados.

Foi desenvolvido um script para coleta de dados aplicando técnica de *web scraping*; utilizaram-se 3 fontes de dados distintas; foi feito tratamento e junção dos dados; foram criados modelos de *machine learning*; diversas análises foram realizadas, tanto na exploração dos dados quanto na análise dos erros de previsão; gráficos foram largamente utilizados e os resultados foram interpretados e comunicados adequadamente.

Como contribuição para a academia, apresentou-se a aplicação de uma técnica de regressão para previsão de momento de ocorrência de evento futuro.

7. Links

Link para o vídeo:

<https://youtu.be/zzupludmkhc>

Link para o repositório:

https://github.com/bwariel/TCC_PucMinasVirtual_DataScience

-
- ⁱ Por que cresce o interesse em bitcoin no Brasil e no mundo? G1, 16 de abril de 2021. Disponível em: <<https://g1.globo.com/economia/especial-publicitario/mercado-bitcoin/noticia/2021/04/16/por-que-cresce-o-interesse-em-bitcoin-no-brasil-e-no-mundo.ghml>>. Acesso em 19 de julho de 2021
- ⁱⁱ Nascimento, Daniela Pereira. Bê-a-bá Cripto: criptomoedas ou criptoativos? Money Times, 17 de outubro de 2020. Disponível em: <<https://www.moneytimes.com.br/be-a-ba-cripto-criptomoedas-ou-criptoativos/>>. Acesso em 20 de julho de 2021
- ⁱⁱⁱ Nakamoto, Satoshi. (2008) Bitcoin: A Peer-to-Peer Electronic Cash System. Disponível em: <<https://bitcoin.org/bitcoin.pdf/>>. Acesso em 20 de julho de 2021
- ^{iv} CoinMarketCap. Bitcoin. Disponível em: <<https://coinmarketcap.com/pt-br/currencies/bitcoin/>>. Acesso em 20 de julho de 2021
- ^v Nakamoto, Satoshi. (2009) Bitcoin v0.1 released. Disponível em: <<https://www.mail-archive.com/cryptography@metzdowd.com/msg10142.html/>>. Acesso em 20 de julho de 2021
- ^{vi} Ravencoin. Disponível em: <<https://ravencoin.org/>>. Acesso em 21 de julho de 2021
- ^{vii} Fenton, Bruce; Black, Tron. (2018) Ravencoin: A Peer to Peer Electronic System for the Creation and Transfer of Assets. Disponível em: <<https://ravencoin.org/assets/documents/Ravencoin.pdf>>. Acesso em 21 de julho de 2021
- ^{viii} Binance Academy. Bitcoin and the Stock to Flow Model. Disponível em: <<https://academy.binance.com/en/articles/bitcoin-and-the-stock-to-flow-model/>>. Acesso em 21 de julho de 2021
- ^{ix} Kapilkov, Michael. Pesquisador desmascara o modelo stock-to-flow, compara Bitcoin a um 'estoque de tecnologia'. Disponível em <<https://cointelegraph.com.br/news/a-researcher-debunks-stock-to-flow-model-likens-bitcoin-to-a-tech-stock/>>. Acesso em 21 de julho de 2021
- ^x PlanB. Modeling Bitcoin Value with Scarcity. Disponível em: <<https://medium.com/@100trillionUSD/modeling-bitcoins-value-with-scarcity-91fa0fc03e25>>. Acesso em 21 de julho de 2021
- ^{xi} Stock-to-Flow Model. Disponível em: <<https://www.lookintobitcoin.com/charts/stock-to-flow-model/>>. Acesso em 21 de julho de 2021
- ^{xii} Bitcoin Monetary Inflation. Disponível em <<https://chart-studio.plotly.com/~BashCo/5.embed/>>. Acesso em 21 de julho de 2021
- ^{xiv} Bertolucci, Gustavo. Estudos de correlação do Bitcoin podem ajudar traders. Disponível em: <<https://livecoins.com.br/estudos-de-correlacao-do-bitcoin-podem-ajudar-traders/>>. Acesso em 21 de julho de 2021
- ^{xv} Analytics Vidhya. Analysing the Cryptocurrency of May 2021 | Python for Finance basics. Disponível em: <<https://www.analyticsvidhya.com/blog/2021/05/analyzing-the-cryptocurrency-of-may-2021-python-for-finance-basics/>>. Acesso em 21 de julho de 2021

^{xvi} Ravencoin Halving. Disponível em: <<https://ravencoin.org/halving/>>. Acesso em 21 de julho de 2021