Data Science on Blockchain with R. Part II: Tracking the NFTs

By Thomas de Marchin and Milana Filatenkova

OCTOBER 2021

Thomas is Senior Data Scientist at Pharmalex. He is passionate about the incredible possibility that blockchain technology offers to make the world a better place. You can contact him on Linkedin (https://www.linkedin.com/in/tdemarchin/) or Twitter (https://twitter.com/tdemarchin).

Milana is Data Scientist at Pharmalex. She is passionate about the power of analytical tools to discover the truth about the world around us and guide decision making. You can contact her on Linkedin (https://www.linkedin.com/in/mfilatenkova/).

Examples of Weird Whale NFTs. These NFTs (token ids 525, 564, 618, 645, 816, 1109, 1523 and 2968) belong to the creator of the collection Benyamin Ahmed (Benoni) who gave us the permission to show them in this article.

1 Introduction

What is the Blockchain: A blockchain is a growing list of records, called blocks, that are linked together using cryptography. It is used for recording transactions, tracking assets, and building trust between participating parties. Primarily known for Bitcoin and cryptocurrencies application, Blockchain is now used in almost all domains, including supply chain, healthcare, logistic, identity management… Some blockchains are public and can be accessed from everyone while some are private. Hundreds of blockchains exist with their own specifications and applications: Bitcoin, Ethereum, Tezos…

What are NFTs: Non-Fungible Tokens are used to represent ownership of unique items. They let us tokenize things like art, collectibles, patents, even real estate. They can only have one official owner at a time and they're secured by the Blockchain, no one can modify the record of ownership or copy/paste a new NFT into existence.

What is R: R language is widely used among statisticians and data miners for developing data analysis software.

What is a smart contract: Smart contracts are simply programs stored on a blockchain that run when predetermined conditions are met. They typically are used to automate the execution of an agreement so that all participants can be immediately certain of the outcome, without any intermediary's involvement or time loss. They can also automate a workflow, triggering the next action when conditions are met. An example of a smart contract use case is the lottery: people buy tickets and at a predefined time, a winner is automatically selected and money is transferred to his account, all this without involvement of a third party.

This is the second article on a series on Blockchain. Part I focused on how to read the blockchain and we introduced a few concepts. If you haven't read it, I strongly encourage you to do so to get familiar with the tools and terminology we use in this second article: Part I (https://towardsdatascience.com/data-science-on-blockchain-with-r-afaf09f7578c).

are popular with
It is not uncommon to hear that cryptocurrencies ~~is heavily used by~~ Mafia as it is anonymous and confidential. This is only partially true. While
everywhere
we don't know exactly who is behind an address, the transactions made by this address are visible from ~~everyone~~. ~~And~~ unless you are very
is
careful, it is practically possible to determine who behind the address by crossing databases. There are now companies specialized in doing only
that a breakthrough path towards a and fair World linked to lack of traceability and
~~this~~. This is ~~an incredible way~~ to ~~make the world~~ more transparent. Blockchain has the potential to resolve major problems ~~the world currently~~ accountability
has
~~face~~. Take for example the cacao culture in Ivory Coast. The Country lost more than 60% of its "protected" forests during the last 25 years
ing

official inefficiency
despite the engagement of the agribusiness to fight against deforestation. The main reason for the inefficacy of current strategies to protect wild
down the origin of
forests is that it is extremely difficult to trace where the cocoa beans have been grown, the existing traceability solutions being easily
certain of drug production industry
manipulated. Blockchain could help to solve this issue. In the pharmaceutical world, blockchain has also the potential to improve many aspects such as tracking the supply chain and the manufacturing or securing the clinical trial data management, for example.
For example, this technology would enhance the transparancy into manufacturing process and supply chain, as well as management of the clinical data.
topic of the to do this
The question we will cover in this article is how to extract and visualize the blockchain transactions. We will explore some tools available in R. I
to avoid messing up with the mafia :D
more benign but currently very popular application of blockchain technology we will look at
don't want to have any problem with the mafia so let's focus on a hot topic at the moment: the art NFTs. We will look here more specifically at the
The approach described here, is of course extendible to
Weird Whales NFTs but we could look at any NFTs. This can of course be used to investigate anything stored on the blockchain.

The *Weird Whales* project is a collection of 3350 whales which have been programmatically generated from an ocean of combinations, each
their
with unique characteristics and different traits: https://weirdwhalesnft.com/ (https://weirdwhalesnft.com/). This project was created by a 12-year-old programmer named Benyamin Ahmed who puts on sale on the famous NFT marketplace *OpenSea*. The 3,350 computer-generated *Weird Whales* almost instantly sold out based on the heartwarming story and Benyamin made more than 400,000$ in two months. Whales were initially sold at approximately 60$ but since then, there price has been multiplied by 100… Read this (https://www.cnbc.com/2021/08/25/12-year-old-coder-made-6-figures-selling-weird-whales-nfts.html) to learn more about this incredible story.

2 Data

dedicated to consisting in This is a very interesting topic
This section is about downloading the sales data (which tokens were transferred to which address) as well as the sale prices. **It is very**
in the data analysis
interesting but also a bit technical so if you are only interested by the analysis, you can skip this section and go directly to section 3.

2.1 Transfers

The *Weird Whales* are managed by a specific smart contract. This contract is stored on a specific address and you can read its code on EtherScan: https://etherscan.io/address/0x96ed81c7f4406eff359e27bff6325dc3c9e042bd#code (https://etherscan.io/address/0x96ed81c7f4406eff359e27bff6325dc3c9e042bd#code) To make it easier to extract information from the blockchain, which can be fairly complicated due to how the information is stored on the ledger, we can read the events. In Solidity, events are dispatched signals the smart contracts can fire. Any app connected to Ethereum network can listen to these events and act accordingly. Here is a list of recent *Weird Whales* events: https://etherscan.io/address/0x96ed81c7f4406eff359e27bff6325dc3c9e042bd#events (https://etherscan.io/address/0x96ed81c7f4406eff359e27bff6325dc3c9e042bd#events)

mostly gets written
We are particularly interested by the a specific type of event: the transfer. Every time a transfer of a token takes place, an event is stored on the
the following
blockchain with that structure: Transfer (index_topic_1 *address from*, index_topic_2 *address to*, index_topic_3 uint256 *tokenId*). As indicated by its names, this event records the address from which the token is transferred, the address to which it is transferred and the token ID, which goes from 1 to 3350 (as there were 3350 *Weird Whales* generated).

We will therefore extract all transfer events related to *Weird Whales*. For this, we can filter on the hash signature of this event (also called Topic 0). By doing a bit of reverse engineering on EtherScan (https://etherscan.io/tx/0xa677cfc3b4084f7a5f2e5db5344720bb2ca2c0fe8f29c26b2324ad8c8d6c2ba3#eventlog (https://etherscan.io/tx/0xa677cfc3b4084f7a5f2e5db5344720bb2ca2c0fe8f29c26b2324ad8c8d6c2ba3#eventlog)), we see that topic 0 for this event is "0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef".

Below, we outline a process to create a databases containing trade data of the *Weird Whales*. EtherScan limits the number of result per call to 1000. That's not enough to analyze the Weird Whale transactions as only the minting (process of creation of the token on the blockchain) generates 3350 transaction (1 transaction per NFT minted). And that's without all the subsequent transfers! That's why we have to use a dirty while loop. Note that if you are ready to pay a bit, there are other blockchain database available without restriction. For example, the Ethereum database is available on Google BigQuery.

```
# First, let's load a few useful packages
library(knitr)
library(tidyverse)
library(httr)
library(jsonlite)
library(plotly)
library(patchwork)
library(cowplot)
library(network)
library(ggraph)
library(networkDynamic)
library(ndtv)
library(tsna)
```

```
# EtherScan requires a token, have a look at their website. This is my token but please use your own!
EtherScanAPIToken <- "UJP16VCE9D29XFAA86RWADATJ5K4PBSYD9"

dataEventTransferList <- list()
continue <- 1
i <- 0

while(continue == 1){ # We will run trough the earliest blocks mentioning Weird whales to the most recent.
  i <- i + 1
  print(i)
  if(i == 1){fromBlock = 12856383} #first block mentioning Weird Whale contract
    load
  # Get the transfer events from the Weird Whale contract
  resEventTransfer <- GET("https://api.etherscan.io/api",
                          query = list(module = "logs",
                                       action = "getLogs",
                                       fromBlock = fromBlock,
                                       toBlock = "latest",
                                       address = "0x96ed81c7f4406eff359e27bff6325dc3c9e042bd", # address of the Weird Whale contract
                                       topic0 = "0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef", # hash of the transfer event
                                       apikey = EtherScanAPIToken))

  dataEventTransferList[[i]] <- fromJSON(rawToChar(resEventTransfer$content), flatten = T)$result %>%
    select(-gasPrice, -gasUsed, -logIndex) # reformat the data in a dataframe

  if(i > 1){
    if(all_equal(dataEventTransferList[[i]], dataEventTransferList[[i-1]]) == T){continue <- 0}
  } #at some point, we reached the latest transactions and always the same data so we can stop

  fromBlock <- max(as.numeric(dataEventTransferList[[i]]$blockNumber)) # increase the block to start looking at for the next iteration
}

dataEventTransfer <- bind_rows(dataEventTransferList) %>% # coerce the list to dataframe
  distinct() # eliminate potential duplicated rows

# data management
dataEventTransfer <- dataEventTransfer %>%
  rename(contractAddress = address) %>%
  mutate(dateTime = as.POSIXct(as.numeric(timeStamp), origin = "1970-01-01")) %>%
  mutate(topics = purrr::map(topics, setNames, c("eventHash","fromAddress","toAddress","tokenId"))) %>% # it is important to set the names otherwise unnest_wider will print many warning messages.
  unnest_wider(topics) %>% # reshape the topic column (list) to get a column for each topic.
  mutate(tokenId = as.character(as.numeric(tokenId)), # convert Hexadecimal to numeric
         blockNumber = as.numeric(blockNumber),
         fromAddress = paste0("0x", str_sub(fromAddress,-40,-1)), # reshape the address format
         toAddress = paste0("0x", str_sub(toAddress,-40,-1))) %>%
  mutate(tokenId = factor(tokenId, levels = as.character(sort(unique(as.numeric(tokenId)))))) %>%
  select(-data, -timeStamp, -transactionIndex)

saveRDS(dataEventTransfer, "data/dataEventTransfer.rds")
```

This is how the Transfer dataset looks like:

```
dataEventTransfer <- readRDS("data/dataEventTransfer.rds")
glimpse(dataEventTransfer)
```

```
## Rows: 11,338
## Columns: 8
## $ contractAddress <chr> "0x96ed81c7f4406eff359e27bff6325dc3c9e042bd", "0x96ed8~
## $ eventHash       <chr> "0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f~
## $ fromAddress     <chr> "0x0000000000000000000000000000000000000000", "0x00000~
## $ toAddress       <chr> "0x8a502e0e3eda70eae505a6fa0fa49eb29b85fe5b", "0x8a502~
## $ tokenId         <fct> 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 3, 10, 11, 12, 13, 14, 1~
## $ blockNumber     <dbl> 12856383, 12856383, 12856383, 12856383, 12856383, 1285~
## $ transactionHash <chr> "0x17654fa3a9b49fc1688df27d195ffb59a17e2b7b03d69aefc39~
## $ dateTime        <dttm> 2021-07-19 12:06:00, 2021-07-19 12:06:00, 2021-07-19 ~
```

2.2 Sales price

While both the transfer and the sales can be managed by the same contract, it is done differently ~~on~~ in *OpenSea*. The sale is managed by the *OpenSea* contract and if it is approved (asked price reached), the main *OpenSea* contract calls the *Weird whales* contract which then triggers the transfer. If we want to know the price at which the NFTs were sold (in addition to the transfer discussed above), we need to extract data from this second contract. The sales are recorded by an event ~~named~~ called *OrderMatch*.

Note that this loop can take a while to run as we download all the sales prices for all the NFT sales on *OpenSea*, not only the *Weird Whales*. ~~Knowing~~ Given that at the time of writing, there were on average 30 transactions per minute on *OpenSea*, it can take a while to download… This code block is very similar to the one above so if you are unsure about what a line does exactly, read the code comments above. it

```
# Get the OrderMatch events from the Weird Whale contract
dataEventOrderMatchList <- list()
continue <- 1
i <- 0

while(continue == 1){
  i <- i + 1
  print(i)
  if(i == 1){fromBlock = 12856383}

  resEventOrderMatch <- GET("https://api.etherscan.io/api",
                            query = list(module = "logs",
                                         action = "getLogs",
                                         fromBlock = fromBlock,
                                         toBlock = "latest",
                                         address = "0x7be8076f4ea4a4ad08075c2508e481d6c946d12b", # address of the Open Sea
contract
                                         topic0 = "0xc4109843e0b7d514e4c093114b863f8e7d8d9a458c372cd51bfe526b588006c9", #
 hash of the OrderMatch event
                                         apikey = EtherScanAPIToken))

  dataEventOrderMatchList[[i]] <- fromJSON(rawToChar(resEventOrderMatch$content), flatten = T)$result %>%
    select(-gasPrice, -gasUsed, -logIndex)

  if(i > 1){
    if(all_equal(dataEventOrderMatchList[[i]], dataEventOrderMatchList[[i-1]]) == T){continue <- 0}
  }

  fromBlock <- max(as.numeric(dataEventOrderMatchList[[i]]$blockNumber))
}

dataEventOrderMatch <- bind_rows(dataEventOrderMatchList) %>%
  distinct()

dataEventOrderMatch <- dataEventOrderMatch %>%
  mutate(topics = purrr::map(topics, setNames, c("eventHash","fromAddress","toAddress","metadata"))) %>%
  unnest_wider(topics)
```

If we look at the orderMatch event structure, we see that the price is encoded in uint256 type in the data field. It is preceded by two others fields, buyHash and sellHash, both in bytes32 types. The uint256 and bytes32 types are both 32 bytes long, which makes 64 Hexadecimal characters. We are not interested by the buyHash and sellHash data but only by the price sale. We thus have to retrieve the last 64 characters and convert them ~~to~~ into decimal to get the sale price.

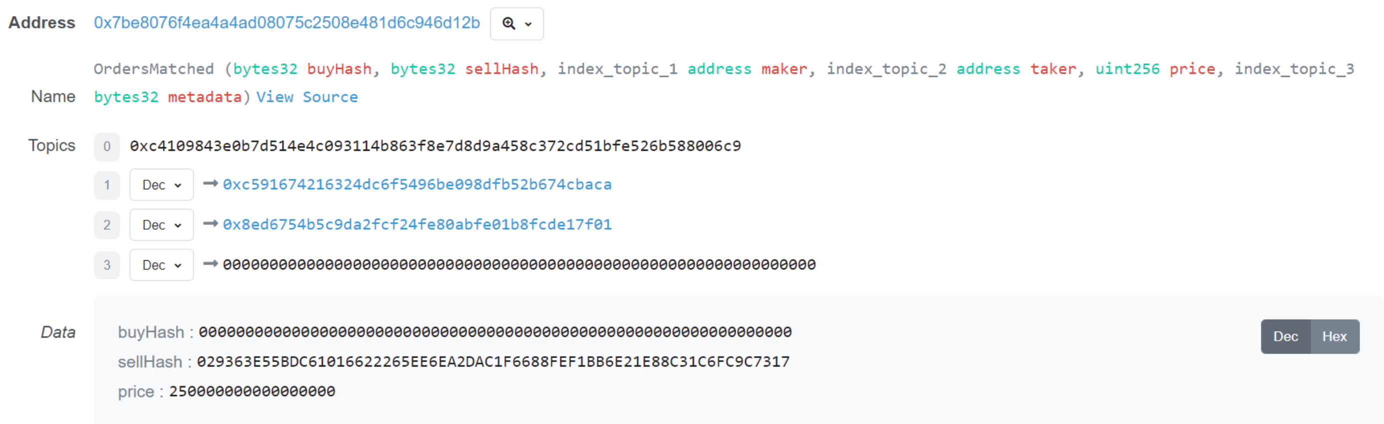


Figure 1: Structure of an orderMatch event

```
dataEventOrderMatch <- dataEventOrderMatch %>%
  mutate(priceETH = str_sub(data, start = -64),
         priceETH = as.numeric(paste0("0x", priceETH)),
         priceETH = priceETH / 10^18)  %>% # this is expressed in Wei, the smallest denomination of ether. 1 ether = 1,00
0,000,000,000,000,000 Wei (10\^18).
  select(priceETH, transactionHash)
```

2.3 Combine the two events

Let's now merge the two dataset by the transactionHash of the *Weird Whales* transfers.

```
# Merge the transfer and orderMatch events
dataWeirdWhales <- left_join(dataEventTransfer, dataEventOrderMatch, by = "transactionHash")

# Minting does not involve any real "sale" but still cost money. As there is no orderMatch event for minting, the priceETH
is NA and we will manually update it to the minting cost. Weird Whales can also be transfered for free (or the cryptocurre
ncy transaction is made outside openSea) and in this case, there is no sale price either. Similarly, we will manually set
 the price of these transfers to 0 ETH.
dataWeirdWhales <- dataWeirdWhales %>%
  mutate(priceETH = case_when(
    fromAddress == "0x0000000000000000000000000000000000000000" ~ 0.025,
    is.na(priceETH) ~ 0,
    TRUE ~ priceETH
  )
)

saveRDS(dataWeirdWhales, "data/dataWeirdWhales.rds")
```

2.4 Convert the ETH price in USD

As we are working on the Ethereum blockchain, the transaction price are given in ETH. Ethereum / USD rate is highly volatile so if we want to convert ETH to USD, we cannot just apply a multiplicative factor. We thus have to download the historical ETH USD price. This time, we won't download the data from EtherScan (you need a pro account for that) but from the Poloniex exchange, which provide free access to this functionality.

We will use a spline function approximation to smooth and interpolate the conversion rate. The reason is that the resolution of the timestamp of the transaction event is the second while the resolution of the historical price dataset is much lower. We thus have to interpolate.

```
dataWeirdWhales <- readRDS("data/dataWeirdWhales.rds")

# download historical price, see https://docs.poloniex.com/#returnchartdata for more information
resHistoricalPrice <- GET("https://poloniex.com/public",
                          query = list(command = "returnChartData",
                                       currencyPair = "USDT_ETH",
                                       start = as.numeric(min(dataWeirdWhales$dateTime)),
                                       end = as.numeric(max(dataWeirdWhales$dateTime)),
                                       period = 1800)) # resolution of the dataset. 1800 corresponds to one row for every
 30 minutes.

dataHistoricalPrice <- fromJSON(rawToChar(resHistoricalPrice$content), flatten = T)

dataHistoricalPrice <- dataHistoricalPrice %>%
  select(date, weightedAverage) %>% # we need only the price per date
  mutate(date = as.POSIXct(as.numeric(date), origin = "1970-01-01")) %>%
  rename(ETHtoUSDRate = weightedAverage)

# try the interpolation spline on the historical conversion rates
historicalInterpolationETHUSD <- approx(x=dataHistoricalPrice$date,
                 y=dataHistoricalPrice$ETHtoUSDRate,
                 xout=seq(min(dataHistoricalPrice$date),
                          max(dataHistoricalPrice$date),
                          length.out=1000)) %>%
  bind_rows()

# plot the historical conversion rates together with the spline: it works quite well!
pETHUSDConversionRate <- ggplot(aes(x = date, y = ETHtoUSDRate), data = dataHistoricalPrice) +
  geom_point() +
  scale_x_datetime(date_breaks = "2 week") +
  geom_line(aes(x = x, y = y), col = "red", data = historicalInterpolationETHUSD)
ggplotly(pETHUSDConversionRate)
```

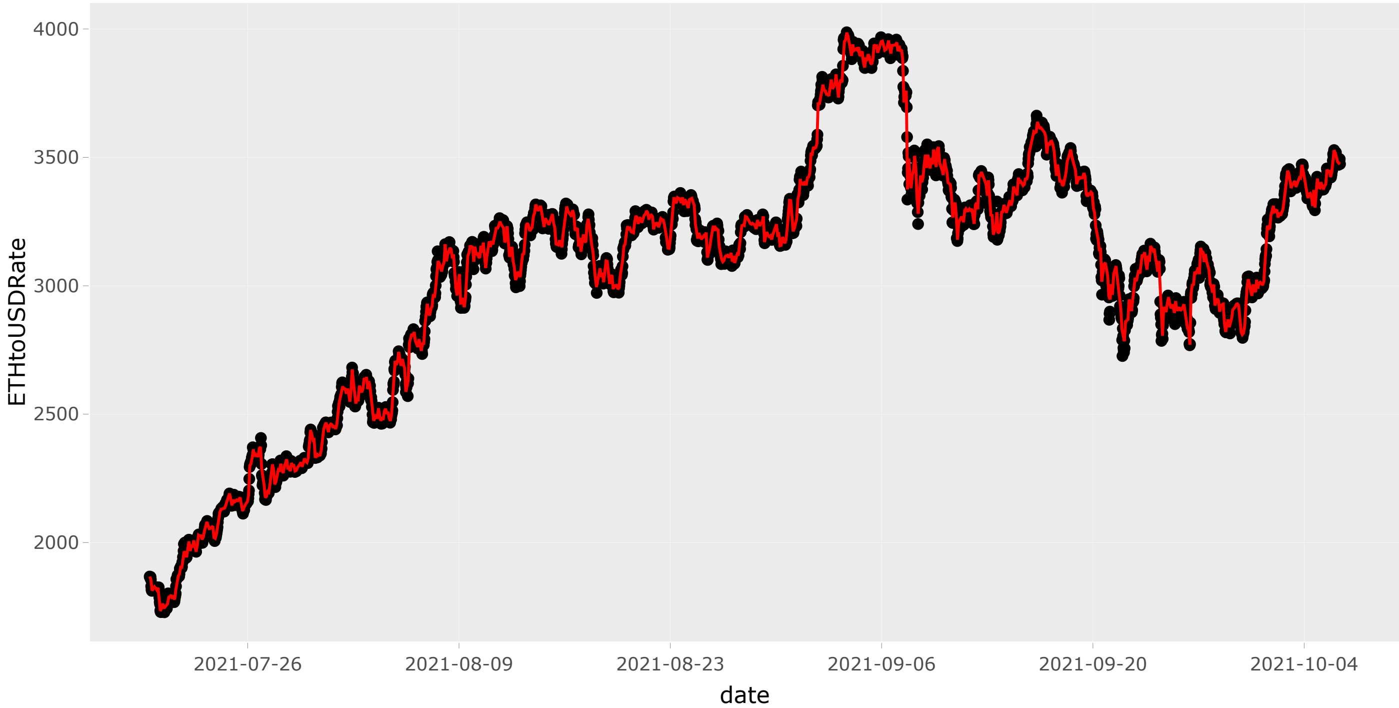


Figure 2: Historical ETH to USD rate. Red line: spline.

Let's now use our spline to convert the ETH in USD for our *WeirdWhales* transactions.

```
# let's use it to convert ETH to USD
dataWeirdWhales <- dataWeirdWhales %>%
  mutate(ETHtoUSDRate = bind_rows(approx(x = dataHistoricalPrice$date,
               y = dataHistoricalPrice$ETHtoUSDRate,
               xout = dateTime))$y,
         priceUSD = round(priceETH * ETHtoUSDRate,3)
  )

saveRDS(dataWeirdWhales, "data/dataWeirdWhalesFinal.rds")
```

2.5 Final dataset

Note that if you didn't manage to download all the data from EtherScan, you can just load the dataset available on the github. This is how the final dataset looks like:

```
dataWeirdWhales <- readRDS("data/dataWeirdWhalesFinal.rds")
glimpse(dataWeirdWhales)
```

```
## Rows: 11,338
## Columns: 11
## $ contractAddress <chr> "0x96ed81c7f4406eff359e27bff6325dc3c9e042bd", "0x96ed8~
## $ eventHash       <chr> "0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f~
## $ fromAddress     <chr> "0x0000000000000000000000000000000000000000", "0x00000~
## $ toAddress       <chr> "0x8a502e0e3eda70eae505a6fa0fa49eb29b85fe5b", "0x8a502~
## $ tokenId         <fct> 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 3, 10, 11, 12, 13, 14, 1~
## $ blockNumber     <dbl> 12856383, 12856383, 12856383, 12856383, 12856383, 1285~
## $ transactionHash <chr> "0x17654fa3a9b49fc1688df27d195ffb59a17e2b7b03d69aefc39~
## $ dateTime        <dttm> 2021-07-19 12:06:00, 2021-07-19 12:06:00, 2021-07-19 ~
## $ priceETH        <dbl> 0.025, 0.025, 0.025, 0.025, 0.025, 0.025, 0.025, 0.025~
## $ ETHtoUSDRate    <dbl> 1867.824, 1867.824, 1867.824, 1867.824, 1867.824, 1867~
## $ priceUSD        <dbl> 46.696, 46.696, 46.696, 46.696, 46.696, 46.696, 46.696~
```

3 Analysis

3.1 Descriptive statistics

Here are a few summary descriptive statistics on the content of the dataset:

```
# summary statistics
dataWeirdWhales %>%
  summarise(`Number of transactions` = n(),
            `Unique tokens` = length(unique(tokenId)),
            `Unique senders` = length(unique(fromAddress)),
            `Unique receivers` = length(unique(toAddress)),
            `Date range` = paste(min(dateTime), max(dateTime), sep = " - "),
            `Duration` = max(dateTime) - min(dateTime),
            `Total sales volume (USD)` = sum(priceUSD, na.rm = TRUE),
            `Average sale price (USD)` = mean(priceUSD, na.rm = TRUE)
            )  %>%
  t() %>%
  kable(caption = "Table 1: Summary statistics on the content of the dataset.")
```

Table 1: Summary statistics on the content of the dataset.

Number of transactions	11338
Unique tokens	3350
Unique senders	2401
Unique receivers	3634
Date range	2021-07-19 12:06:00 - 2021-10-06 09:04:49
Duration	78.87418 days
Total sales volume (USD)	4666739
Average sale price (USD)	411.6379

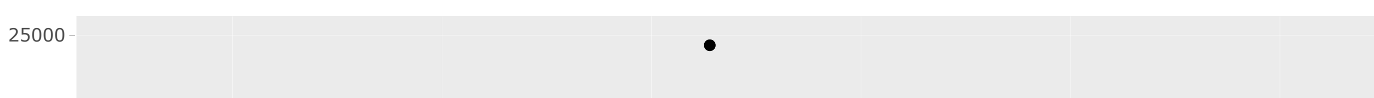
We can determine the number of transactions per address. The first address (0x00..) is not really an address, it refers to the minting of the NFTs. If we omit that one, we see that some addresses have been involved in more than 200 transactions!

```
# summary statistics by address
tibble(address = c(dataWeirdWhales$fromAddress, dataWeirdWhales$toAddress)) %>%
  group_by(address) %>%
  summarise(`Number of transactions` = n()) %>%
  rename(`Address` = address) %>%
  arrange(desc(`Number of transactions`))
```

```
## # A tibble: 3,635 x 2
##    Address                                    `Number of transactions`
##    <chr>                                                         <int>
##  1 0x0000000000000000000000000000000000000000                     3350
##  2 0x8b7c94bc9ec137d67fbddb203b2814f0f1f9b377                      219
##  3 0x6761bcaf2b2156c058634d9772f07374d6edef1d                      218
##  4 0xbff79922fcbf93f9c30abb22322b271460c6bebb                      200
##  5 0x9859a67e2191e1198d9260c2ff5ea858d09116e0                      182
##  6 0x7768a6fa6f9812f3aa91a9f827e5948f7e0b5486                      176
##  7 0x74be0af0bf7254328ddffc09425ff71d64a1a836                      157
##  8 0x7bad2138b818079db0bc7df5185b6bae9589370d                      156
##  9 0x0d08ad2ab7893c04ecb460cbb6822b11c9e8904a                      132
## 10 0x0717ddfe299daa44282b26d8703cede2163bed48                      124
## # ... with 3,625 more rows
```

Let's now visualize the price transactions as a function of the time, irrespective of the token ID. We see a high variability in the first few days. This is followed by a quieter period and we can observe the beginning of an upward period in the last days of August. This price increase match to an intensive activity on the social media as the creator story become viral. We even see one transaction close to 25000 USD, this corresponds to an increase of about 55555% compared to the initial price of 45 USD!

```
pSalesPrice <- ggplot(aes(x = dateTime, y = priceUSD), data = dataWeirdWhales) +
  geom_point() +
  scale_x_datetime(date_breaks = "2 week") +
  labs(y = "Price (USD)", x = "Date")

ggplotly(pSalesPrice)
```



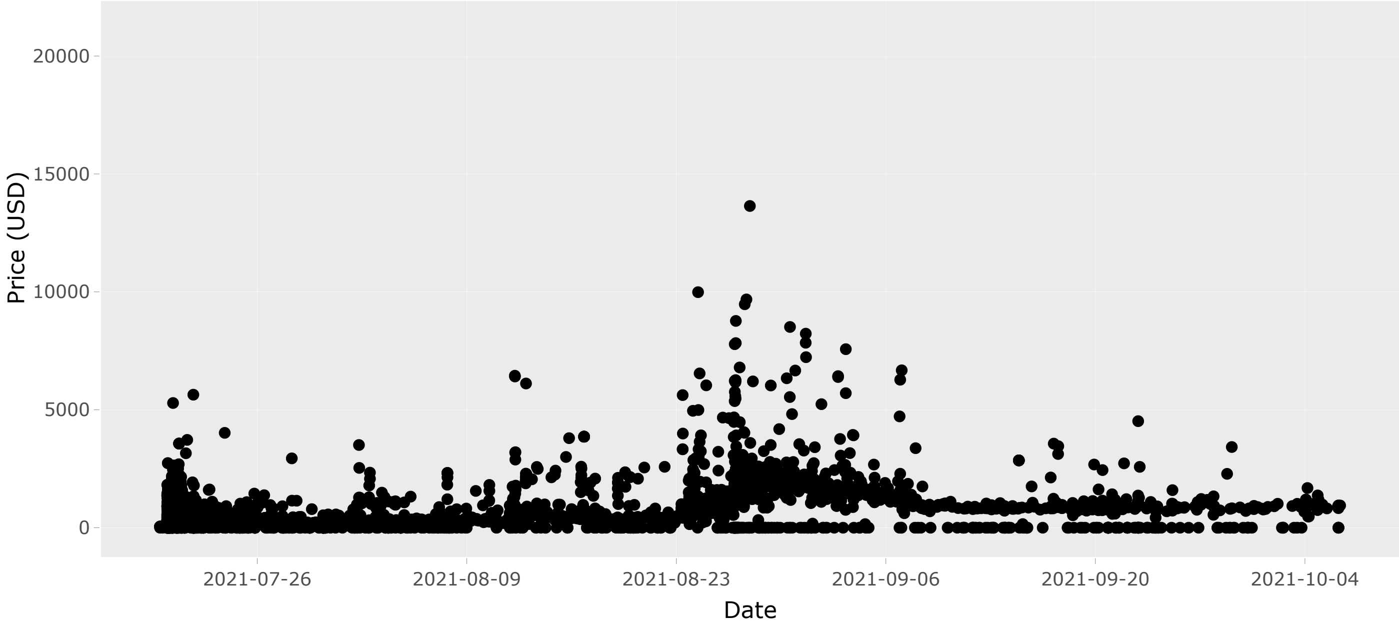


Figure 3: Sales price (USD) evolution for the *Weird Whales* NFTs transactions.

3.2 Visualizing the network

Up to now, we looked at summary statistics on the transactions price. But how to visualize the transactions exactly, knowing that each NFT is unique and needs to be differentiated from the others? The dataset we assembled is perfectly adapted to be plotted as a network. Networks are described by vertex and edges. A vertex (or node) of a graph is one of the objects that are connected together. This will be here the wallet addresses involved in the transactions. The connections between the vertices are called edges (or links). This will be here the transactions.

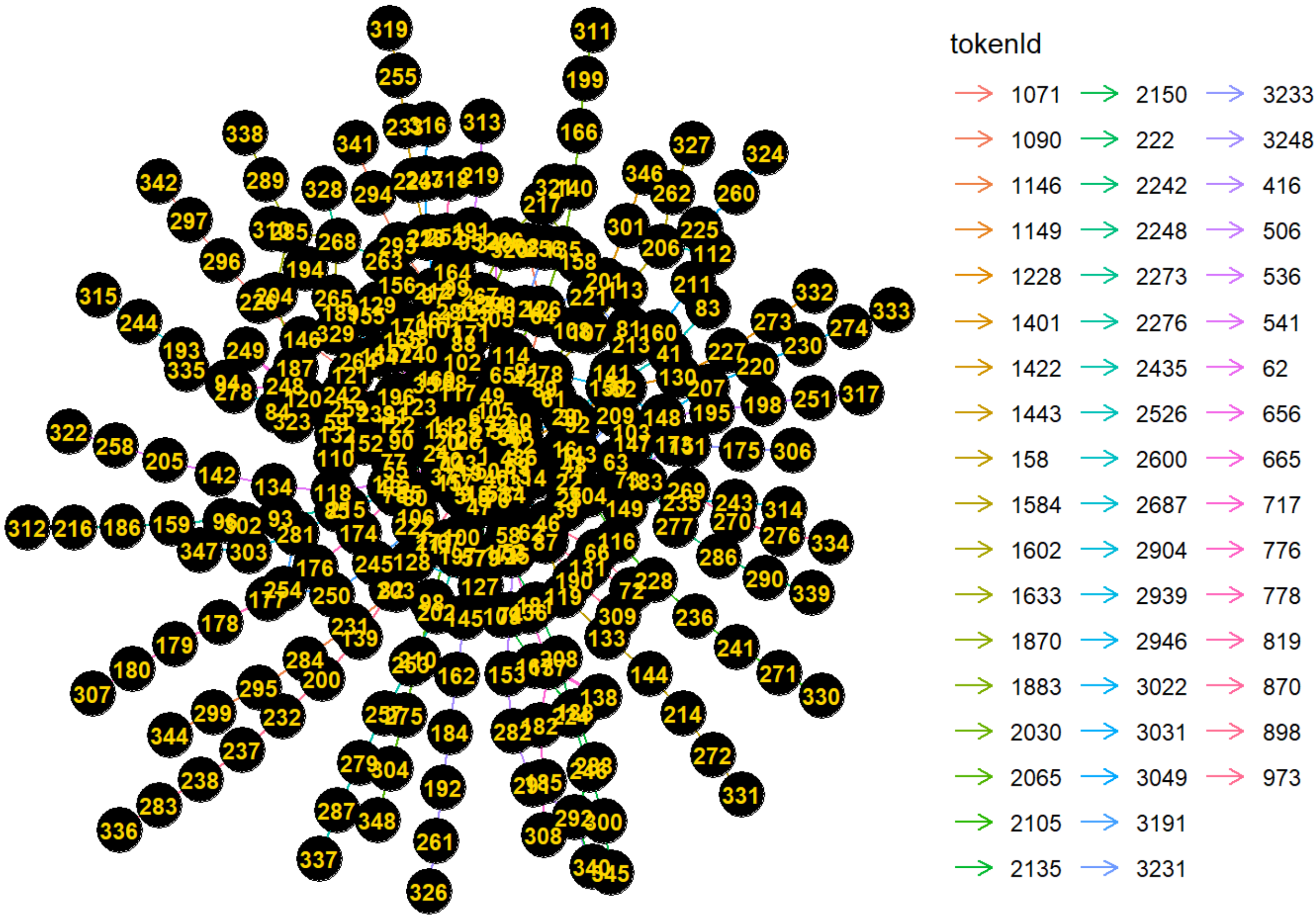
There are several class and corresponding packages available on R to plot networks, the most famous being *network* and *igraph*. Have a look at the References section for a few amazing tutorials on this topic. I have a preferences for the *network* package as it gives the possibility to create interactive plots via the *networkDynamic* and *ndtv* packages. On top of that, other packages have been developed to facilitate manipulation and plotting of such objects, such as the *ggraph* package which brings the *ggplot2* framework to the network class.

Let’s give a try! We will first create a simple static (i.e. without the temporal dimension) network and plot it using the *ggraph* package. There are too many data to display in one plot so we will subset our data by plotting only the NFTs involved in more than 7 transactions.

```
# subset the dataset
tokenIdFilter <- dataWeirdWhales %>%
  group_by(tokenId) %>%
  summarise(n = n()) %>%
  filter(n>7)

dataWeirdWhalesFiltered <- dataWeirdWhales %>%
  filter(tokenId %in% tokenIdFilter$tokenId) %>%
  droplevels()

# restructure the timing to create a temporak network (below)
dataWeirdWhalesFiltered <- dataWeirdWhalesFiltered %>%
  mutate(dateHour = round.POSIXt(dateTime, "hour")) %>% # The time resolution is seconds. That's nice but it leads to a lo
t of computation (frames) for our network. Let's round to hours.
  mutate(dateHourNumeric = as.numeric(dateHour)/3600) %>%
  mutate(dateHourNumeric = dateHourNumeric-min(dateHourNumeric))

# vertices is a listing of all the addresses involved in the transactions
vertices <- tibble(label = unique(c(dataWeirdWhalesFiltered$fromAddress,
                                    dataWeirdWhalesFiltered$toAddress))) %>%
  rowid_to_column("id") %>% # instead of using the addresses to visually identify the vertices, we will use shorter ID num
bers
  mutate(onset = 0,
         terminus = max(dataWeirdWhalesFiltered$dateHourNumeric))

# edges is a listing of the transactions
edges <- dataWeirdWhalesFiltered %>%
  left_join(vertices, by = c("fromAddress" = "label")) %>%
  rename(from = id) %>%
  left_join(vertices, by = c("toAddress" = "label")) %>%
  rename(to = id)

# This will be useful to create a temporal dynamic network (below). Edges will appear at `onset` and disappear at `terminu
s.`
edges <- edges %>%
  rename(onset = dateHourNumeric) %>%
  mutate(terminus = max(onset), #
         tokenId = as.character(tokenId)) %>%
  select(from, to, onset, terminus, tokenId, priceUSD)

# create the network using network
network <- network(edges,
                  vertex.attr = vertices,
                   matrix.type = "edgelist",
                loops = T,
                multiple = T,
                ignore.eval = F)
```

We can now plot our network. We see that all transactions originate from the minting address (1). Some addresses are involved in multiple transactions and that's why we see several (curved) edges for these ones. We also see that some token were transferred to one address and then sent back to the sender.

```
pNetwork <- ggraph(network) +
  geom_edge_fan(aes(color = tokenId), arrow = arrow(length = unit(6, "pt"), type = "open")) +
  geom_node_point(color = "black", size = 8) +
  theme_void() +
  geom_node_text(aes(label = id), color = "gold", size = 3, fontface = "bold")
pNetwork
```



Let's now use the timestamp of the transaction to add a temporal dimension to our network. For this, we will use the amazing *networkDynamic* package.

```
# create a dynamic temporal network
dNetwork <- networkDynamic(edge.spells = as.data.frame(edges[,c("onset", "terminus", "from", "to", "tokenId")]),
                           vertex.spells = as.data.frame(vertices[,c("onset", "terminus", "id", "label")]),
                           create.TEAs = T)
```

```
## Initializing base.net of size 348 imputed from maximum vertex id in edge records
## Activated TEA vertex attributes:  labelActivated TEA edge attributes:  tokenIdCreated net.obs.period to describe networ
k
##  Network observation period info:
##   Number of observation spells: 1
##   Maximal time range observed: 0 until 1852
##   Temporal mode: continuous
##   Time unit: unknown
##   Suggested time increment: NA
```

We can create a timeline plot showing the **frequence** of the activity. We see that 2/3 of the transactions happened very shortly after the NFT's creation. This is followed by a relatively calm period and then a very active period near the end.

```
plot(tEdgeFormation(dNetwork, time.interval = 5), ylab = "Frequence")
```

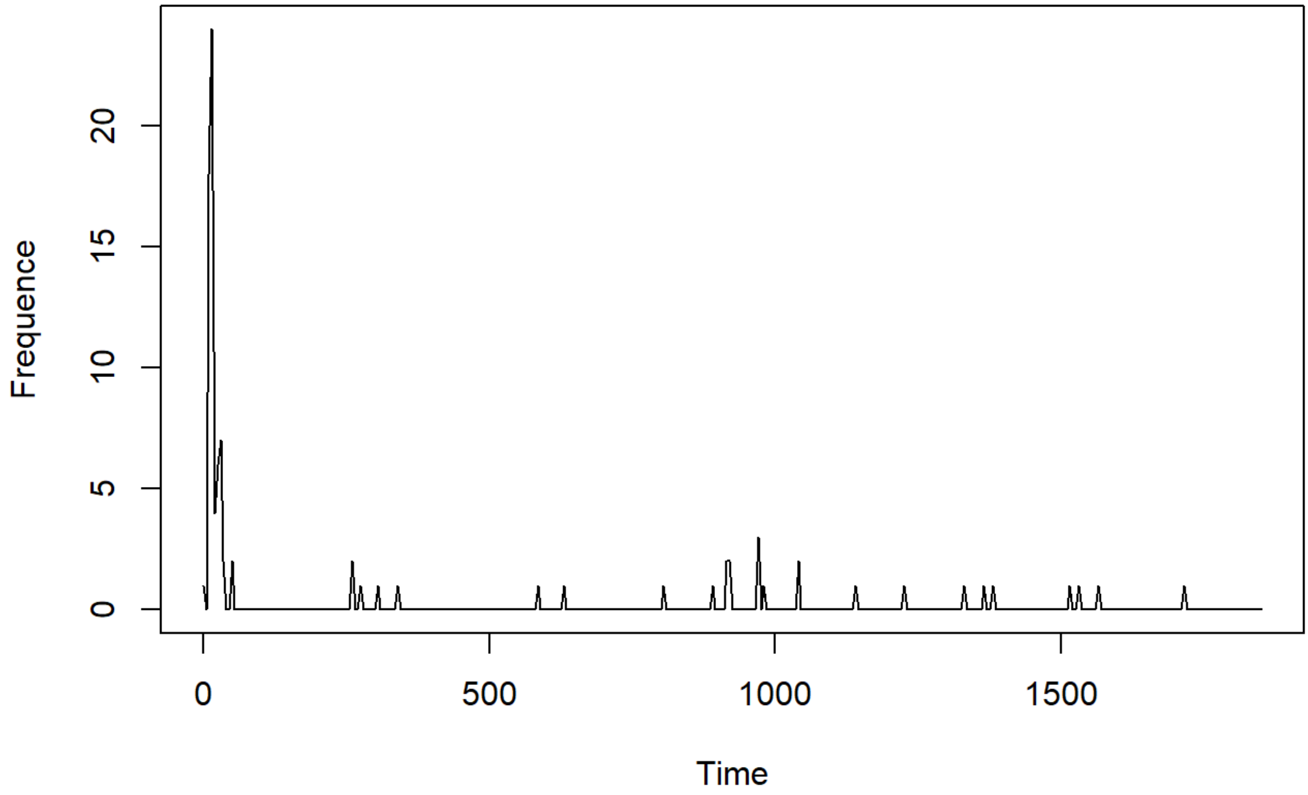


Figure 5: Timeline plot showing the frequence of the transations.

Below, we create an amazing animation which can be rendered directly in the browser. Edges and vertices can be clicked to show more information about the address and the tokens involved in the transactions.

```
# compute a sequence of layout for the rendering (this can take some time)
compute.animation(dNetwork,
                  animation.mode = 'MDSJ',
                  slice.par = list(interval = 50,
                                   start = 1,
                                   end = max(edges$terminus),
                                   aggregate.dur = 50,
                                   rule = 'any'),
                  verbose = F)

render.d3movie(dNetwork,
               output.mode = 'htmlWidget',
               vertex.tooltip = paste("<b>Address:</b>", (network %v% 'label')),
               edge.tooltip = paste("<b>TokenId:</b>", (network %e% 'tokenId')),
               launchBrowser = T,
               main = 'Transactions of Weird Whales NFTs',
               edge.col = function(slice){slice%e%'tokenId'},
               usearrows = F,
               verbose = F)
```

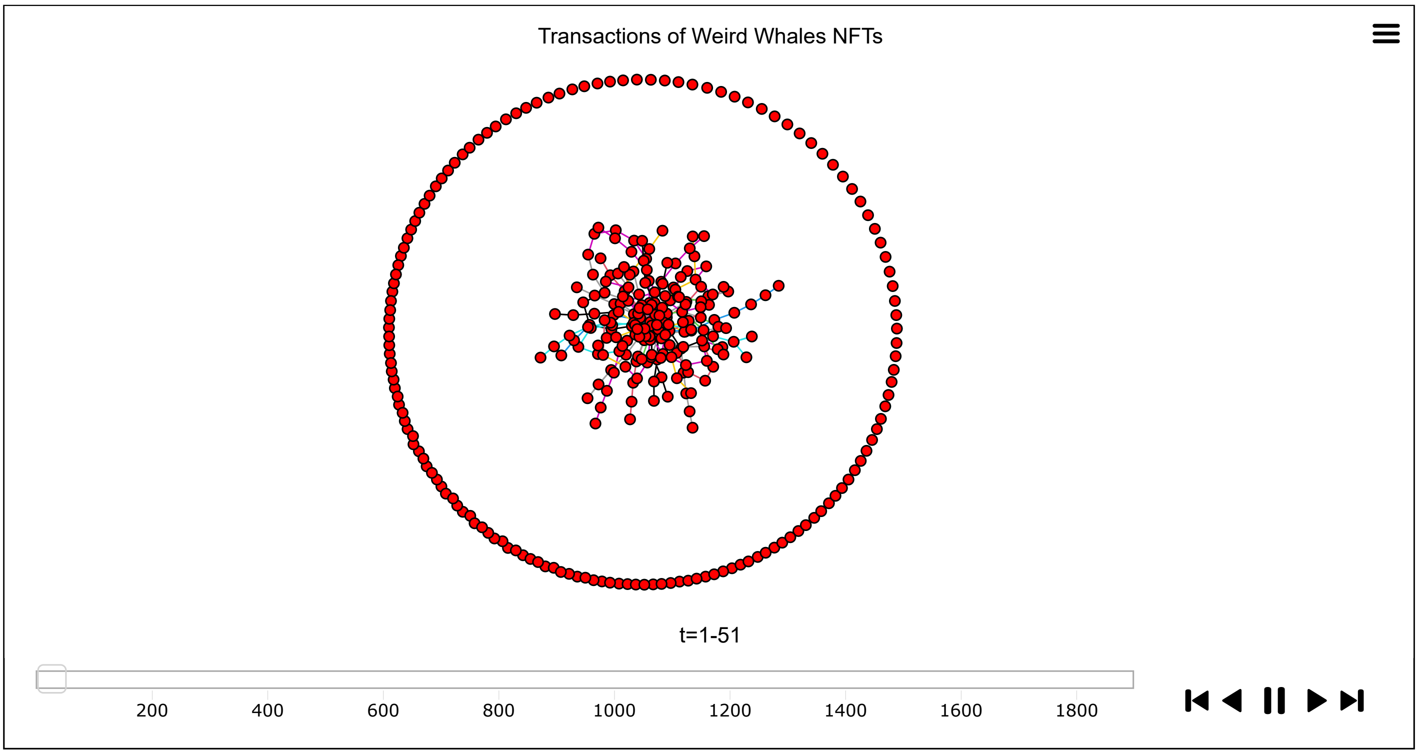


Figure 6: Animated network of the WheirdWhales NFTs transactions. Each address is represented by a node (circle) and the transations are represented by the edges (lines). The edge color refers to the token ID. Edges and vertices can be clicked to show more information about the address and the tokens involved in the transactions.

4 Conclusion

This article is an introduction into how to visualize the blockchain transactions. Here, we have shown an example of how to download and plot a network of the transactions associated to the *WeirdWhales* NFTs. We have a few ideas for the next article (Part III). We can explore the Tezos blockchain, the new place to be for NFTs, or we could also investigate the Helium blockchain, a physical decentralized wireless blockchain-powered network for Internet of Things (IoT) devices. Please contact us if you have any question or request! **Que dire d'autre**?

Note that the code used to generate this article is available on my Github: https://github.com/tdemarchin/DataScienceOnBlockchainWithR-PartII (https://github.com/tdemarchin/DataScienceOnBlockchainWithR-PartII)

If you want to help us continue working on blockchain, don't hesitate to donate to our Ethereum address 0xf5fC137E7428519969a52c710d64406038319169 or Tezos address tz1ffZLHbu9adcobxmd411ufBDcVgrW14mBd

5 References

Powered by Etherscan.io and Poloniex APIs:

https://etherscan.io/ (https://etherscan.io/)

https://docs.poloniex.com/ (https://docs.poloniex.com/)

General:

https://ethereum.org/en (https://ethereum.org/en)

https://www.r-bloggers.com/ (https://www.r-bloggers.com/)

Network:

https://kateto.net/network-visualization (https://kateto.net/network-visualization)

https://www.jessesadler.com/post/network-analysis-with-r/ (https://www.jessesadler.com/post/network-analysis-with-r/)

https://programminghistorian.org/en/lessons/temporal-network-analysis-with-r (https://programminghistorian.org/en/lessons/temporal-network-analysis-with-r)

https://ggraph.data-imaginist.com/index.html (https://ggraph.data-imaginist.com/index.html)