# Parsiq Bounty

## Eth Top Accounts Dynamic Index

This project analyzes the money flow of the top 5.000 eth addreses.

The hypothesis is that there is a relationship between the price of the asset and the flow of money in this group. That should be of particular interest to develop trading strategies or insights about ETH price behaviour. The user can change parameters to aggregate different number of addreses and visually discover correlations between it and the ETH price during the same period.

**Only running the query during several weeks, and reviewing the data again, we should evaluate the usefulness of the idea for trading strategies.**
PARSIQ DATA: All the eth transactions has been recorded through Parsiq "Triggers and Transports". The data has been collected in two periods (with a gap of one day) due to the limitation of the parsiq trial account.

The analysis has been realized using Julia language and Pluto(Julia notebooks)

1.- A static view of the notebook is available in the TopAccountsDynamicIndex.pdf

2.- A working version is provided using ngrok link.

3.- All the data files are included in the repository.

The project is based on a "offline version" of Parsiq data. It is possible to get a online version trought a web hook and run the following code online. But in this notebook, I have copied the telegram events to a file in order to provide an analysis without waiting for new data.

## 1.- Load the top accounts by ETH balance.

```
Any[
    1:  "0xc02aaa39b223fe8d0a0e5c4f27ead9083c756cc2"
    2:  "0x00000000219ab540356cbb839cbe05303d7705fa"
    3:  "0xbe0eb53f46cd790cd13851d5eff43d12404d33e8"
    4:  "0x73bceb1cd57c711feac4224d062b0f6ff338501e"
    5:  "0x4ddc2d193948926d02f9b1fe9e1daa0718270ed5"
    6:  "0x9bf4001d307dfd62b26a2f1307ee0c0307632d59"
    7:  "0x53d284357ec70ce289d6d64134dfac8e511c8a3d"
    8:  "0xc61b9bb3a7a0767e3179713f3a5c7a9aedce193c"
    9:  "0x61edcdf5bb737adffe5043706e7c5bb1f1a56eea"
    10:  "0xc098b2a3aa256d2140208c3de6543aaef5cd3a94"
]
```

```julia
begin
    path = "/Volumes/ROCKET-XTRM/ParsiqBounty/"
    topAccounts = []
    open(path * "eth_accounts_rank.csv", "r") do io
        while !eof(io)
            rw = split(readline(io),"\t")
            if !(rw[2] in topAccounts)
                push!(topAccounts,rw[2])
            end
        end
    end
    topAccounts[1:10] # list first 10
end
```

## 2.- Addresses to be excluded.

We are going to track addresses of big holders, so some of them have to be excluded:

a) contract

b) Exchanges, etc...

0xc02aaa39b223fe8d0a0e5c4f27ead9083c756cc2,Wrapped Ether
0x00000000219ab540356cbb839cbe05303d7705fa,Eth2 Deposit Contract
0xbe0eb53f46cd790cd13851d5eff43d12404d33e8,Binance 7
0x4ddc2d193948926d02f9b1fe9e1daa0718270ed5,Compound: cETH Token
0x53d284357ec70ce289d6d64134dfac8e511c8a3d,Kraken 6
0xc61b9bb3a7a0767e3179713f3a5c7a9aedce193c,Bitfinex: MultiSig 3
0x61edcdf5bb737adffe5043706e7c5bb1f1a56eea,Gemini 3
0xc098b2a3aa256d2140208c3de6543aaef5cd3a94,FTX Exchange 2
0xf66852bc122fd40bfecc63cd48217e88bda12109,Huobi 37
0xdc24316b9ae028f1497c275eb9192a3ea0f67022,Lido: Curve Liquidity Farming Pool Contract
0xe853c56864a2ebe4576a807d26fdc4a0ada51919,Kraken 3
0xdf9eb223bafbe5c5271415c75aecd68c21fe3d7f,Liquity: Active Pool
0x8484ef722627bf18ca5ae6bcf031c23e6e922b30,Polygon (Matic): Ether Bridge
0xa929022c9107643515f5c777ce9a910f0d1e490c,HECO Chain: Bridge
0xde0b295669a9fd93d5f28d9ec85e40f4cb697bae,EthDev
0x011b6e24ffb0b5f5fcc564cf4183c5bbbc96d515,Arbitrum: Bridge
0x220866b1a2219f40e72f5c628b65d54268ca3a9d,Vb 3
0x3bfc20f0b9afcace800d73d2191166ff16540258,Polkadot: MultiSig
0x07ee55aa48bb72dcc6e9d78256648910de513eca,Gemini: Contract 1
0xa7efae728d2936e78bda97dc267687568dd593f3,OKEx 3
0x66f820a414680b5bcda5eeca5dea238543f42054,Bittrex 3
0xbddf00563c9abd25b576017f08c46982012f12be,CONTRACT

...

```
Any[
    1:  "0xc02aaa39b223fe8d0a0e5c4f27ead9083c756cc2"
    2:  "0x00000000219ab540356cbb839cbe05303d7705fa"
    3:  "0xbe0eb53f46cd790cd13851d5eff43d12404d33e8"
    4:  "0x4ddc2d193948926d02f9b1fe9e1daa0718270ed5"
    5:  "0x53d284357ec70ce289d6d64134dfac8e511c8a3d"
    6:  "0xc61b9bb3a7a0767e3179713f3a5c7a9aedce193c"
    7:  "0x61edcdf5bb737adffe5043706e7c5bb1f1a56eea"
    8:  "0xc098b2a3aa256d2140208c3de6543aaef5cd3a94"
    9:  "0xf66852bc122fd40bfecc63cd48217e88bda12109"
    10:  "0xdc24316b9ae028f1497c275eb9192a3ea0f67022"
]
```

```julia
begin
    exAccounts = []
    open(path * "eth_accounts_exreason.csv", "r") do io
        while !eof(io)
            rw = readline(io)
            rw = split(rw,",")
            if !(rw[1] in exAccounts)
                push!(exAccounts,rw[1])
            end
        end
    end
    exAccounts[1:10] # list first 10
end
```

## 3.- Accounts to be loaded -> Parsiq user data.

The list is sliced in 5 files to have the capability of loading manually.

```
begin
    accounts = []
    for ac in topAccounts
        if !(ac in exAccounts)
            push!(accounts, ac)
        end
    end
    """
    open("/Volumes/ROCKET-XTRM/ParsiqBounty/accounts1.csv", "a") do io
    for i=1:1000
        println(io,accounts[i])
    end
    end
    open("/Volumes/ROCKET-XTRM/ParsiqBounty/accounts2.csv", "a") do io
    for i=1001:2000
        println(io,accounts[i])
    end
    end
    open("/Volumes/ROCKET-XTRM/ParsiqBounty/accounts3.csv", "a") do io
    for i=2001:3000
        println(io,accounts[i])
    end
    end
    open("/Volumes/ROCKET-XTRM/ParsiqBounty/accounts4.csv", "a") do io
    for i=3001:4000
        println(io,accounts[i])
    end
    end
    open("/Volumes/ROCKET-XTRM/ParsiqBounty/accounts5.csv", "a") do io
    for i=4001:length(accounts)
        println(io,accounts[i])
    end
    end
    """
end;

```

```
    Any[
       1:  "0x73bceb1cd57c711feac4224d062b0f6ff338501e"
       2:  "0x9bf4001d307dfd62b26a2f1307ee0c0307632d59"
       3:  "0x1b3cb81e51011b549d78bf720b0d924ac763a7c2"
       4:  "0xe92d1a43df510f82c66382592a047d288f85226f"
       5:  "0xca8fa8f0b631ecdb18cda619c4fc9d197c8affca"
       6:  "0x8103683202aa8da10536036edef04cdd865c225e"
       7:  "0x0a4c79ce84202b03e95b7a692e5d728d83c44c76"
       8:  "0x2b6ed29a95753c3ad948348e3e7b1a251080ffb9"
       9:  "0x2fe0260b44eac48cfe4f3bb1ba380bbe2979b468"
       10:  "0x9845e1909dca337944a0272f1f9f7249833d2d19"
       11:  "0x189b9cbd4aff470af2c0102f365fc1823d857965"
       12:  "0x59448fe20378357f206880c58068f095ae63d5a5"
       13:  "0xdc1487e092caba080c6badafaa75a58ce7a2ec34"
       14:  "0x558553d54183a8542f7832742e7b4ba9c33aa1e6"
       15:  "0x98ec059dc3adfbdd63429454aeb0c990fba4a128"
       16:  "0x40f0d6fb7c9ddd9cbc1c02a208380c08cf77189b"
       17:  "0xd65bd7f995bcc3bdb2ea2f8ff7554a61d1bf6e53"
       18:  "0x7712bdab7c9559ec64a1f7097f36bc805f51ff1a"
       19:  "0x19184ab45c40c2920b0e0e31413b9434abd243ed"
       20:  "0x90a9e09501b70570f9b11df2a6d4f047f8630d6d"

          more

       4496:  "0x59cf247531145cf96d2381b853bdb190deed7cd5"
       4497:  "0x1b705bec5ca8406fd621ef09010a4c3a233e19eb"
       4498:  "0xb0d928f08ae6f9d30ee7a83fb53236833ff9abc4"
       4499:  "0x9d8df144cf3ca343c8747d5a911005edb8337a74"
       4500:  "0x446a6d1ae02e845f76db54a4b1df038e334e3ea9"
       4501:  "0x9b30a4c4e5006fc789ffa8ba8bacbeb198bd4281"
       4502:  "0x42d19684d4c941570ad8b347e2890d6db0da3778"
       4503:  "0xba46d1acfd75643925f80d6f78d86c4eee6b389d"
       4504:  "0xfab9b89d9a0317f0f2a0e6643a1c956c66550ed4"
       4505:  "0xce7d28b2a232004c0e102519fb2b50e6360d5840"
]
```

```julia
accounts
```

```julia
begin
    # Dictionary that store the pairs account => ranking
    accRank = Dict()
    ct =1
    for acc in accounts
        accRank[acc] = ct
        ct += 1
    end
end
```

```
1
```

```julia
accRank["0x73bceb1cd57c711feac4224d062b0f6ff338501e"]
```

```
4505
```

```julia
accRank["0xce7d28b2a232004c0e102519fb2b50e6360d5840"]
```

# 4.- Parsiq trigger.

Note: more fields than necessary in the trigger. I wanted to add more things to the project but the limitations of the number of events made that impossible.

(@from in BigAcc && !(@to in BigAcc)) || (!(@from in BigAcc) && @to in BigAcc)

Parsiq code:

stream AccMovements

from Transfers

where (@from in BigAcc && !(@to in BigAcc)) || (!(@from in BigAcc) && @to in BigAcc)

process

let txInfo = { txHash: @tx_hash }

let ts = { blockts: @block_timestamp }

let isCrystalProviderDataReady = false

let crystalProviderData = {percentRiskScore: "0", decimalRiskScore: {value: 0, decimals: 0}, targetName: "", targetType: "", signals: {atm: "", darkMarket: "", darkService: "", exchange: "", gambling: "", illegalService: "", marketplace: "", miner: "", mixer: "", payment: "", ransom: "", riskyExchange: "", scam: "", stolenCoins: "", trustedExchange: "", wallet: ""}}

let assetSymbol = "ETH"

let cryptorankProvider_fiatRateUSD = getRate(assetSymbol)

let cryptorankProvider*fiatRateUSD*transferAmount = mul({ value: @value, decimals: 18}, cryptorankProvider_fiatRateUSD)

emit { assetSymbol, @from, @to, @value, txInfo, ts, cryptorankProvider*fiatRateUSD, cryptorankProvider*fiatRateUSD*transferAmount, @sender*balance*before, @sender*balance*after, @receiver*balance*before, @receiver*balance_after }

end

## 5.- Delivery Channel. Telegram.

```
Transfer from {from} to {to} for {value|eth} ETH detected, {txInfo.txHash}, {ts.blockts},
{cryptorankProviderfiatRateUSDtransferAmount.value|cryptorankProviderfiatRateUSDtransferAmount.d
USD (1 {assetSymbol} =
{cryptorankProviderfiatRateUSD.value|cryptorankProviderfiatRateUSD.decimals} USD)
```

## 6.- Offline version.

It is possible to get a online version trought a web hook and run the following code online. But in this notebook I have copied the telegram events to a file for the following reasons:

a) Here we are limited by the number of events that the account can use.

b) Also the judges will have a better view of the bounty having the data prerecorded in a static view.

### Only running the query during several weeks, and reviewing the data again, we should evaluate the usefulness of the idea for trading strategies.

## 7.- Loading events from file.

```
 Any[
    1: "Transfer from 0x88d840caed39f1d7511ccd50d308e3f76940be67 to 0xefb2e870b14d7e555
    2: "Transfer from 0x6b8b778410311352868 62e878dcbfb8bd5ae61d2 to 0xefb2e870b14d7e555
    3: "Transfer from 0x03c8c937384f0c79eb09debe390652e0de98e2b0 to 0xefb2e870b14d7e555
]
```

```
begin
    events = []
    open(path * "parsiq_events.csv", "r") do io
        while !eof(io)
            rw = readline(io)
            push!(events,rw)
        end
    end
    events[1:3] # list first 3
end
```

```
916
```

```
# Number of file events
length(events)
```

# 8.- Money Flow.

For every event: each account will be update with the dollar amount. "+" if the account receive money and "-" if the account is the sender.

As we are interested in the aggregate behaviour, there will be two versions, "up" and "down".

Example:

account nº5 "up" will add cumulatively the flows of 1+2+3+4+5 "down" will add cumulatevely the flows of 5+6+...

That will help to manipulate the parameters of the charts below without recalculation.

```
begin
    prices = []
    cumUp = Dict()
    cumDown = Dict()
    ethPrice = []

    for i=1:length(accounts) # for plotting set all pairs to keys => []
        cumUp[i] = [0.0]
        cumDown[i] = [0.0]
    end

    for event in events
        ev = split(event," ")
        evFrom = ev[3]
        evTo = ev[5]
        evEthQ = parse(Float64, ev[7])
        if evEthQ > 0
            # transaccion > 0 eth. Another iteration will add acc with 0 to the
filter
            evTx = ev[10]
            flowSign = 0
            aRank = 0
            if evFrom in accounts
                flowSign = -1
                aRank = accRank[evFrom] #account ranking
            else evTo in accounts
                flowSign = 1
                aRank = accRank[evTo] #account ranking
            end
            evDollarAmount = parse(Float64, ev[12])
            evEthPrice = evDollarAmount/evEthQ
            push!(ethPrice, evEthPrice)

            for i=1:length(accounts)
                # Add dollar flow to the cumUp
                cv = cumUp[i]
                if i <= aRank
                    push!(cv, cv[end] + flowSign * evDollarAmount)
                else
                    push!(cv, cv[end] + 0)
                end
                cumUp[i] = cv
                # Add dollar flow to the cumDown
                cv = cumDown[i]
                if i >= aRank
                    push!(cv, cv[end] + flowSign * evDollarAmount)
                else
                    push!(cv, cv[end] + 0)
                end
                cumDown[i] = cv
            end
        end
    end
end
```

## 9.- Interactive Plots.

```
begin
    using Plots
    using Statistics
end
```

## a) Aggregation Down

Select the index of the aggregation.

1 = only the top account selected

...

n = 1+2+3...+n cumulative money flow

**Both time series are normalized to plot.**

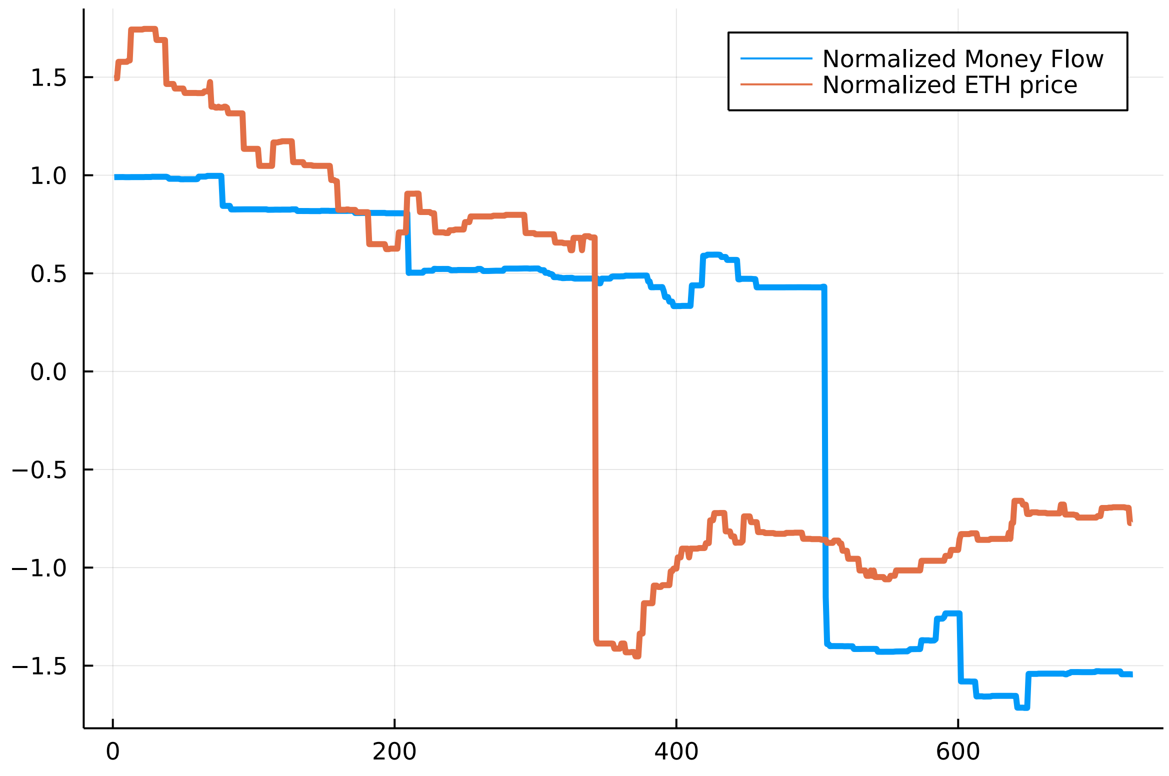
**---------------------**

**Setting the index "i" to 2500 shows that the top half of the 4505 are strongly correlated to ETH price**

```
2507
```

```
i
```

```
@bind i html"<input type=range min=1 max=4505>"
```



```
plot(
    [(cumDown[i] .- mean(cumDown[i])) ./ std(cumDown[i]), (ethPrice .-
mean(ethPrice)) ./ std(ethPrice)], label=["Normalized Money Flow" "Normalized ETH
price"], lw = 3)
```

## b) Aggregation Up

Select the index of the aggregation.

1 = 1 + 2 + ... + 4505

...

n = n + (n+1) + ... + 4505 cumulative money flow

**Both time series are normalized to plot.**

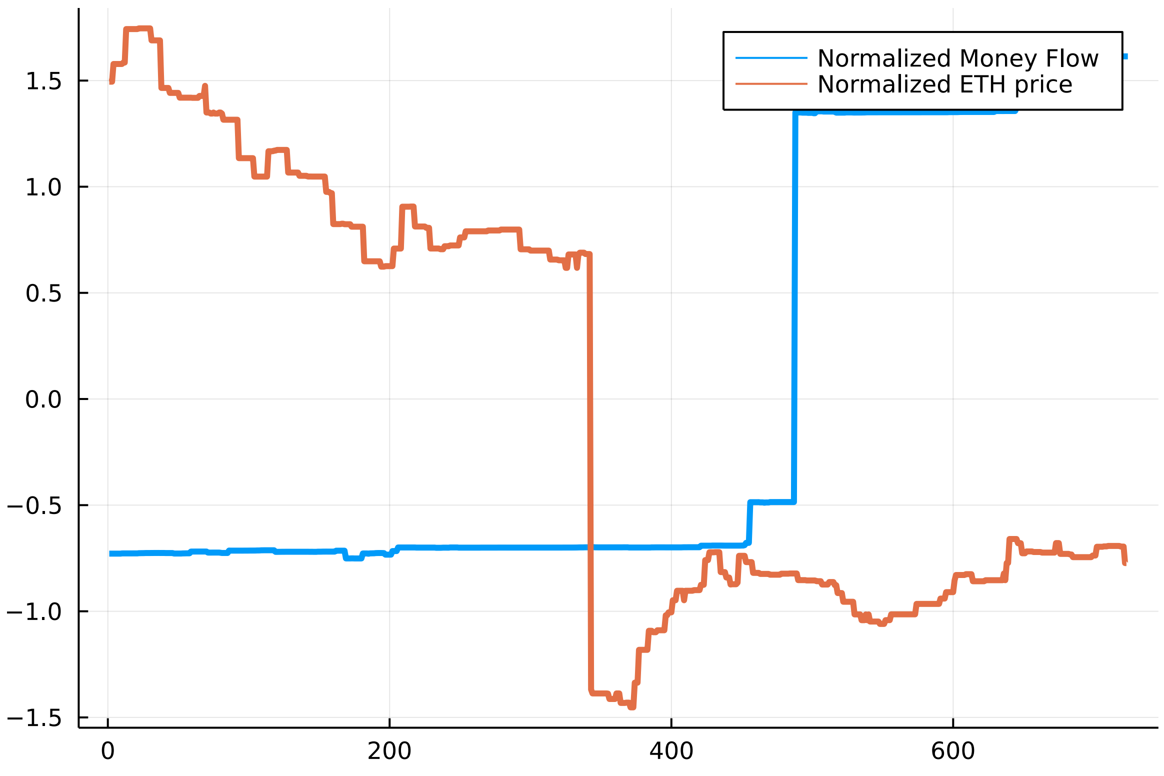
---------------------

**Setting the index "j" to 2547 shows that the bottom half of the 4505 are poorly correlated to ETH price**

2558

```
j
```

```
@bind j html"<input type=range min=1 max=4505>"
```



```
plot(
    [(cumUp[j] .- mean(cumUp[j])) ./ std(cumUp[j]), (ethPrice .- mean(ethPrice)) ./
std(ethPrice)], label=["Normalized Money Flow" "Normalized ETH price"], lw = 3)
```