ethereum_brainrex

Python notebook using data from multiple data sources · 643 views · 5mo ago · data visualization, tutorial

8 Edit My Copy 13

Versic- 10

10 cc

Bienvenido to Blockmad Labs Meetup 4 de mayo

En este taller analizaremos la blockchain de Ethereum a traves de visualizaciones. Para eso usaremos el historial de datos de la blockchain de Ethereum, datos de mercados y metadatos sobre redes sociales proporcionados por la API de Brainrex.

Creditos : https://www.kaggle.com/yazanator/analyzing-ethereum-classic-via-google-bigquery (https://www.kaggle.com/yazanator/analyzing-ethereum-classic-via-google-bigquery) (We forked kernel and adapted to Ethereum)

Agenda.

1. Visualizacion de Ethereum Blockchain with BigQuery Public Datasets
2. Adding Ethereum crypto exchange (Coinbase USD/BTC) usando la API de Brainrex.
3. Integrating Social Media Stats from Brainrex API
4. Conclusiones

Que datos tenemos? El dataset de Ethereum cuenta con las siguientes tablas.

Datos que usaremos. Proveidos por Google Cloud y Brainrex Store.

Esta base de datos está construida por Google y está siendo actualizada en tiempo real. Puedes aprender más de cómo se construyo en este Articulo. https://cloud.google.com/blog/products/data-analytics/ethereum-bigquery-public-dataset-smart-contract-analytics (https://cloud.google.com/blog/products/data-analytics/ethereum-bigquery-public-dataset-smart-contract-analytics)

Datos disponibles en BigQuery Ethereum BLockchain .

Columnas	Description
blocks	This table contains a set of all blocks in the blockchain and their attributes.
contracts	This table contains a subset of Ethereum addresses that contain contract byte-code, as well as some basic analysis of that byte-code.
logs	This table is generally useful for reporting on any logged event type on the Ethereum blockchain.
token_transfers	This table contains the subset of those transactions and has further processed and denormalized the data to make it easier to consume for analysis of token transfer events.
tokens	Token data.
traces	Traces exported using Parity trace module.
transactions	This table contains a set of all transactions from all blocks, and contains a block identifier to get associated block-specific information associated with each transaction.

Datos disponibles en Brainrex Data Integrations .

Columns	Description
exchanges	This table contains a set of all blocks in the blockchain and their attributes.
currency_pairs	This table contains a subset of Ethereum addresses that contain contract byte-code, as well as some basic analysis of that byte-code.
generate_candles	This table is generally useful for reporting on any logged event type on the Ethereum blockchain.
twitter_stats	This table contains the subset of those transactions and has further processed and denormalized the data to make it easier to consume for analysis of token transfer events.

In [1]:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-python
# Here we import the packages we need for the analysis, they are installed in Kaggle's backend

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv
# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list the files in the
 input directory

import os
print(os.listdir("../input"))
# Any results you write to the current directory are saved as output.
!pip install plotly
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
init_notebook_mode(connected=True)
```

```
['ethereum-blockchain']
Requirement already satisfied: plotly in /opt/conda/lib/python3.6/site-packages (3.8.1)
Requirement already satisfied: decorator>=4.0.6 in /opt/conda/lib/python3.6/site-packages (from
plotly) (4.3.0)
Requirement already satisfied: six in /opt/conda/lib/python3.6/site-packages (from plotly) (1.1
2.0)
Requirement already satisfied: nbformat>=4.2 in /opt/conda/lib/python3.6/site-packages (from pl
otly) (4.4.0)
Requirement already satisfied: retrying>=1.3.3 in /opt/conda/lib/python3.6/site-packages (from
plotly) (1.3.3)
Requirement already satisfied: pytz in /opt/conda/lib/python3.6/site-packages (from plotly) (20
18.4)
Requirement already satisfied: requests in /opt/conda/lib/python3.6/site-packages (from plotly)
(2.21.0)
Requirement already satisfied: ipython_genutils in /opt/conda/lib/python3.6/site-packages (from
nbformat>=4.2->plotly) (0.2.0)
Requirement already satisfied: traitlets>=4.1 in /opt/conda/lib/python3.6/site-packages (from n
bformat>=4.2->plotly) (4.3.2)
Requirement already satisfied: jsonschema!=2.5.0,>=2.4 in /opt/conda/lib/python3.6/site-package
```

```
Requirement already satisfied: jsonschema!=2.5.0,>=2.4 in /opt/conda/lib/python3.6/site-package
s (from nbformat>=4.2->plotly) (2.6.0)
Requirement already satisfied: jupyter_core in /opt/conda/lib/python3.6/site-packages (from nbf
ormat>=4.2->plotly) (4.4.0)
Requirement already satisfied: idna<2.9,>=2.5 in /opt/conda/lib/python3.6/site-packages (from r
equests->plotly) (2.6)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /opt/conda/lib/python3.6/site-packages
(from requests->plotly) (3.0.4)
Requirement already satisfied: urllib3<1.25,>=1.21.1 in /opt/conda/lib/python3.6/site-packages
(from requests->plotly) (1.22)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/python3.6/site-packages (fr
om requests->plotly) (2019.3.9)
```

In [2]:

```
# Connects to BigQuery API through Kaggle, no authentification required from Kaggle.
from google.cloud import bigquery
import pandas as pd

client = bigquery.Client()
```

```
Using Kaggle's public dataset BigQuery integration.
```

Coste medio de transacciones por dia (Average Gas Price)

En esta consulta (query) calcularemos el coste medio de gas asociado con las transacciones

In [3]:

```
# Query by Allen Day, GooglCloud Developer Advocate (https://medium.com/@allenday)
query = """
SELECT
  SUM(value/POWER(10,18)) AS sum_tx_ether,
  AVG(gas_price*(receipt_gas_used/POWER(10,18))) AS avg_tx_gas_cost,
  DATE(timestamp) AS tx_date
FROM
  `bigquery-public-data.crypto_ethereum.transactions` AS transactions,
  `bigquery-public-data.crypto_ethereum.blocks` AS blocks
WHERE TRUE
  AND transactions.block_number = blocks.number
  AND receipt_status = 1
  AND value > 0
GROUP BY tx_date
HAVING tx_date >= '2017-01-01' AND tx_date <= '2019-05-04'
ORDER BY tx_date
"""
```

In [4]:

```
query_job = client.query(query)

iterator = query_job.result(timeout=30)
rows = list(iterator)

# Transform the rows into a nice pandas dataframe
df = pd.DataFrame(data=[list(x.values()) for x in rows], columns=list(rows[0].keys()))

# Look at the first 10
df.tail(10)
```

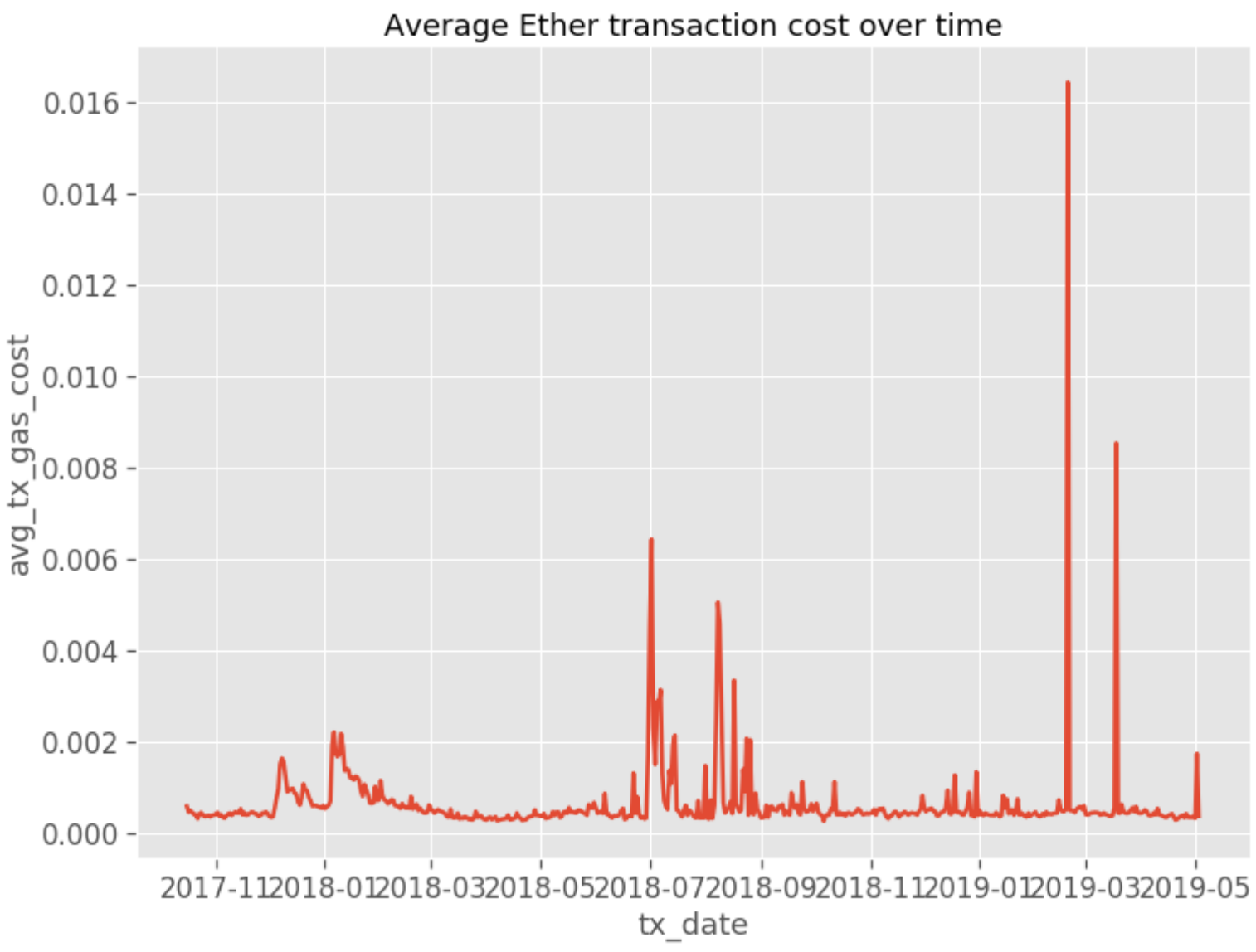
Out[4]:

	sum_tx_ether	avg_tx_gas_cost	tx_date
555	3.474018e+06	0.000390	2019-04-24
556	3.154835e+06	0.000337	2019-04-25
557	4.522744e+06	0.000426	2019-04-26
558	1.913235e+06	0.000350	2019-04-27
559	2.329010e+06	0.000344	2019-04-28
560	3.077766e+06	0.000348	2019-04-29
561	2.962549e+06	0.000370	2019-04-30
562	2.615887e+06	0.000326	2019-05-01
563	2.100681e+06	0.001738	2019-05-02
564	2.526933e+06	0.000371	2019-05-03

In [5]:

```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
plt.style.use('ggplot')
sns.set_context("notebook", font_scale=1.5, rc={"lines.linewidth": 2.5})

f, g = plt.subplots(figsize=(12, 9))
g = sns.lineplot(x="tx_date", y="avg_tx_gas_cost", data=df, palette="Blues_d")
plt.title("Average Ether transaction cost over time")
plt.show(g)
```



Ranking de Mineros por Recompensa en los ultimos 30 dias

Aquí intentamos averiguar quiénes son los principales mineros según la dirección del bloque explotado en los últimos 30 días. Ejecutaremos la siguiente consulta como una cadena en Python a través del cliente BigQuery.

```
> WITH mined_block AS (
  SELECT miner, DATE(timestamp)
  FROM `bigquery-public-data.ethereumblockchain.blocks`
  WHERE DATE(timestamp) > DATE_SUB(CURRENT_DATE(), INTERVAL 1 MONTH)
  ORDER BY miner ASC)
SELECT miner, COUNT(miner) AS total_block_reward
FROM mined_block
GROUP BY miner
ORDER BY total_block_reward ASC
```

Let's run it.

¡Ejecutémoslo!

In [6]:
```
client = bigquery.Client()
ethereum_classic_dataset_ref = client.dataset('crypto_ethereum', project='bigquery-public-data'
)
```

Using Kaggle's public dataset BigQuery integration.

In [7]:
```
query = """
WITH mined_block AS (
  SELECT miner, DATE(timestamp)
  FROM `bigquery-public-data.crypto_ethereum.blocks`
  WHERE DATE(timestamp) > DATE_SUB(CURRENT_DATE(), INTERVAL 1 MONTH)
  ORDER BY miner ASC)
SELECT miner, COUNT(miner) AS total_block_reward
FROM mined_block
GROUP BY miner
ORDER BY total_block_reward DESC
LIMIT 10
"""

query_job = client.query(query)
iterator = query_job.result()
```

In [8]:
```
rows = list(iterator)
# Transform the rows into a nice pandas dataframe
top_miners = pd.DataFrame(data=[list(x.values()) for x in rows], columns=list(rows[0].keys()))
# Look at the first 10 headlines
top_miners.head(10)
```

Out[8]:

	miner	total_block_reward
0	0xea674fdde714fd979de3edf0f56aa9716b898ec8	48118
1	0x5a0b54d5dc17e0aadc383d2db43b0a0d3e029c4c	42664
2	0x829bd824b016326a401d083b33d092293333a830	23984
3	0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5	22356
4	0xb2930b35844a230f00e51431acae96fe543a0347	10322
5	0x2a5994b501e6a560e727b6c2de5d856396aadd38	3377
6	0x2a65aca4d5fc5b5c859090a6c34d164135398226	3164
7	0x1b5b5906306c96b842dc03105e3b38636a4eda0b	3112
8	0xd224ca0c819e8e97ba0136b3b95ceff503b79f53	2505
9	0x005e288d713a5fb3d7c9cf1b43810a98688c7223	2493

Plotly Library for Plotting / Librería de Visualizaciones Plotly

In this notebook, we will be using Plotly for plotting our charts. You can sign up for a free account to get your API key in order to generate the charts if you choose to run this notebook on your own.

En este Notebook, usaremos Plotly para dibujar nuestros gráficos. Puedes obtener una cuenta gratuita que te facilite una API Key para poder generar los gráficos, si decides ejecutar este Notebook en local.

Let's start by plotting the top miners by their block reward as a pie chart.

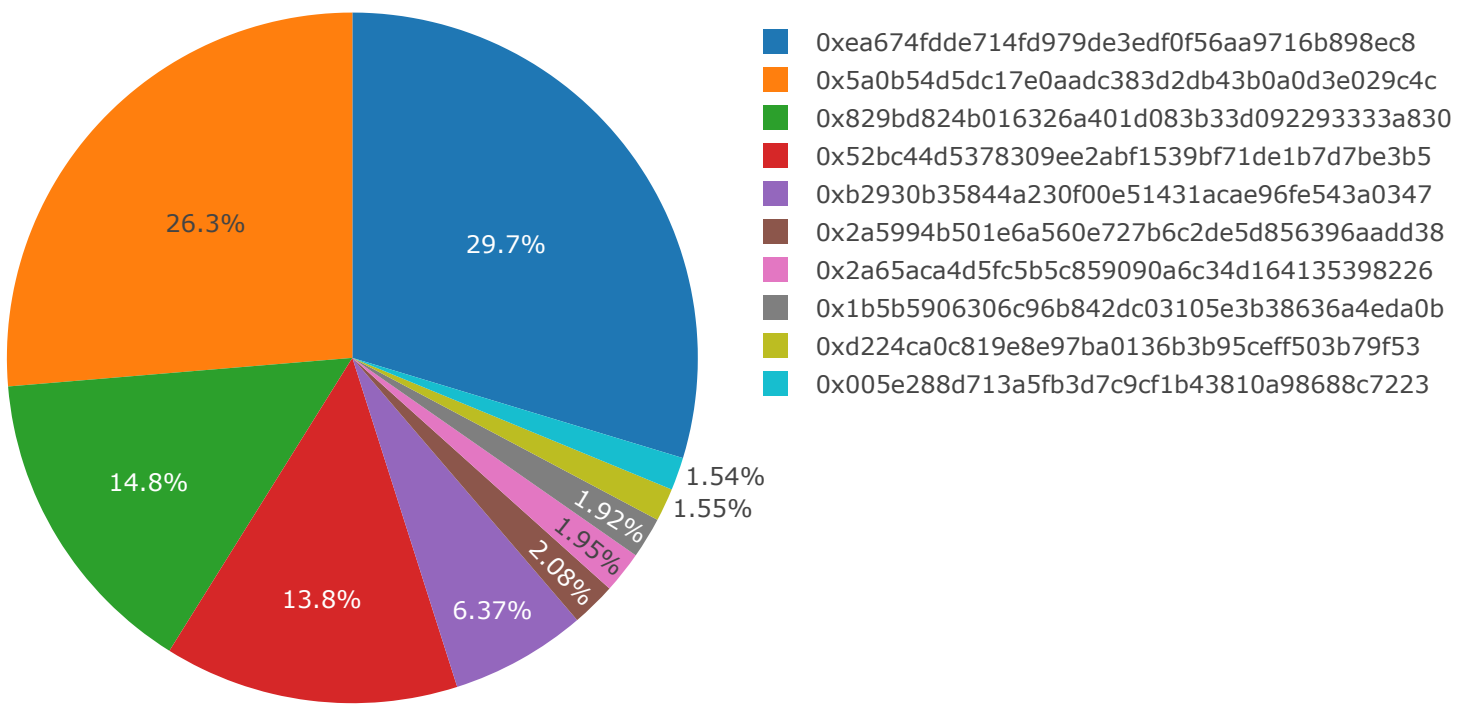
Empecemos dibujando el ranking de los principales mineros, según su recompensa por bloque, como un gráfico de tarta.

In [9]:
```
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
import plotly.graph_objs as go

labels = top_miners['miner']
values = top_miners['total_block_reward']

trace = go.Pie(labels=labels, values=values)

iplot([trace])
```



Top Mineros por recompensa acumulada desde el bloque genesis.

Ahora, hagamos que sea más interesante y planifiquemos (¿analicemos?) las recompensas totales de todos los que alguna vez hayan extraído Ethereum del bloque de génesis.

Lo limitaremos sólo a los mineros, cuyas recompensas diarias de bloques superan los 100. Esto nos permite ahorrar en la computación y trazar los rastros en este Notebook de Kaggle.

Podemos usar la siguiente consulta:

standardSQL

-- MIT License -- Copyright (c) 2019 Yaz Khoury, yaz.khoury@gmail.com

```
SELECT miner,
    DATE(timestamp) as date,
    COUNT(miner) as total_block_reward
FROM `bigquery-public-data.crypto_ethereum.blocks`
GROUP BY miner, date
HAVING COUNT(miner) > 100
ORDER BY date, COUNT(miner) ASC
```

In [10]:

```
query = """
#standardSQL
-- MIT License
-- Copyright (c) 2019 Yaz Khoury, yaz.khoury@gmail.com

SELECT miner,
    DATE(timestamp) as date,
    COUNT(miner) as total_block_reward
FROM `bigquery-public-data.crypto_ethereum.blocks`
GROUP BY miner, date
HAVING COUNT(miner) > f
ORDER BY date, COUNT(miner) ASC
"""
query_job = client.query(query)
iterator = query_job.result()
```

```
---------------------------------------------------------------------------
BadRequest                                Traceback (most recent call last)
<ipython-input-10-fb41c85c578c> in <module>()
     13 """
     14 query_job = client.query(query)
---> 15 iterator = query_job.result()

/opt/conda/lib/python3.6/site-packages/google/cloud/bigquery/job.py in result(self, timeout, re
try)
   2792             not complete in the given timeout.
   2793         """
-> 2794         super(QueryJob, self).result(timeout=timeout)
   2795         # Return an iterator instead of returning the job.
   2796         if not self._query_results:

/opt/conda/lib/python3.6/site-packages/google/cloud/bigquery/job.py in result(self, timeout, re
try)
    705             self._begin(retry=retry)
    706         # TODO: modify PollingFuture so it can pass a retry argument to done().
--> 707         return super(_AsyncJob, self).result(timeout=timeout)
    708
    709     def cancelled(self):

/opt/conda/lib/python3.6/site-packages/google/api_core/future/polling.py in result(self, timeou
t)
    125             # pylint: disable=raising-bad-type
    126             # Pylint doesn't recognize that this is valid in this case.
--> 127             raise self._exception
    128
    129         return self._result

BadRequest: 400 Unrecognized name: f at [11:23]
```

In [11]:

```
rows = list(iterator)
# Transform the rows into a nice pandas dataframe
```

```
top_miners_by_date = pd.DataFrame(data=[list(x.values()) for x in rows], columns=list(rows[0].k
eys()))
top_miners_by_date.head(10)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-11-dd8ccc0ecebf> in <module>()
----> 1 rows = list(iterator)
      2 # Transform the rows into a nice pandas dataframe
      3 top_miners_by_date = pd.DataFrame(data=[list(x.values()) for x in rows], columns=list(r
ows[0].keys()))
      4 top_miners_by_date.head(10)

/opt/conda/lib/python3.6/site-packages/google/api_core/page_iterator.py in __iter__(self)
    217         """
    218         if self._started:
--> 219             raise ValueError("Iterator has already started", self)
    220         self._started = True
    221         return self._items_iter()

ValueError: ('Iterator has already started', <google.cloud.bigquery.table.RowIterator object at
0x7f4712cbbfd0>)
```

In [12]:

```
date_series = top_miners_by_date['date'].unique()
date_series
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-12-ea458972a090> in <module>()
      1
----> 2 date_series = top_miners_by_date['date'].unique()
      3 date_series

NameError: name 'top_miners_by_date' is not defined
```

In [13]:

```
traces = []
miner_series = top_miners_by_date['miner'].unique()

for index, miner in enumerate(miner_series):
    miner_reward_by_date = top_miners_by_date.loc[top_miners_by_date['miner'] == miner]
    miner_reward = miner_reward_by_date['total_block_reward']
    miner_date = miner_reward_by_date['date']
    trace = dict(
        x=miner_date,
        y=miner_reward,
        mode='lines',
        stackgroup='one'
    )
    traces.append(trace)
fig = dict(data=traces)

iplot(fig)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-13-9c4e4a5186de> in <module>()
      1 traces = []
----> 2 miner_series = top_miners_by_date['miner'].unique()
      3
      4 for index, miner in enumerate(miner_series):
      5     miner_reward_by_date = top_miners_by_date.loc[top_miners_by_date['miner'] == miner]
```

```
NameError: name 'top_miners_by_date' is not defined
```

In [14]:
```
query = """
#standardSQL
-- MIT License
-- Copyright (c) 2019 Yaz Khoury, yaz.khoury@gmail.com

SELECT miner,
    DATE(timestamp) as date,
    COUNT(miner) as total_block_reward
FROM `bigquery-public-data.crypto_ethereum_classic.blocks`
GROUP BY miner, date
HAVING COUNT(miner) > f
ORDER BY date, COUNT(miner) ASC
"""
query_job = client.query(query)
iterator = query_job.result()
```

```
---------------------------------------------------------------------------
BadRequest                                Traceback (most recent call last)
<ipython-input-14-eb0907a07b0c> in <module>()
     13 """
     14 query_job = client.query(query)
---> 15 iterator = query_job.result()

/opt/conda/lib/python3.6/site-packages/google/cloud/bigquery/job.py in result(self, timeout, re
try)
   2792             not complete in the given timeout.
   2793         """
-> 2794         super(QueryJob, self).result(timeout=timeout)
   2795         # Return an iterator instead of returning the job.
   2796         if not self._query_results:

/opt/conda/lib/python3.6/site-packages/google/cloud/bigquery/job.py in result(self, timeout, re
try)
    705             self._begin(retry=retry)
    706         # TODO: modify PollingFuture so it can pass a retry argument to done().
--> 707         return super(_AsyncJob, self).result(timeout=timeout)
    708
    709     def cancelled(self):

/opt/conda/lib/python3.6/site-packages/google/api_core/future/polling.py in result(self, timeou
t)
    125             # pylint: disable=raising-bad-type
    126             # Pylint doesn't recognize that this is valid in this case.
--> 127             raise self._exception
    128
    129         return self._result

BadRequest: 400 Unrecognized name: f at [11:23]
```

Daily Gini Coefficient of Ethereum Mining Rewards By Miners

Ahora, vamos a calcular el coeficiente diario de Gini.

El coeficiente de Gini es una medida estadística de la distribución utilizada para medir la distribución del ingreso o la riqueza entre una población.

Cita de articulo de Investopedia:

> "Un país en el que cada residente tenga el mismo ingreso tendría un coeficiente de Gini de 0. Un país en el que un residente obtuviera todos los ingresos, mientras que todos los demás no ganarían nada, t

```
endría un coeficiente de Gini de 1."
```

Aquí estamos calculando la distribución diaria de recompensas de bloque entre las direcciones, según la dirección del minero que recibió un bloque diariamente.

La consulta utiliza lo que se construyó anteriormente y se toma de la implementación de la consulta Gini del Balance diario.

Además, solo por diversión, calcularemos el Promedio Móvil Simple del Gini, usando una ventana de 7 y 30 días. La media móvil simple de SMA toma un conjunto de valores y sus períodos de tiempo y promedia una suma de los valores divididos por la ventana de tiempo elegida.

El valor de consulta aquí será gini_sma_7 y gini_sma_30.

Tenga en cuenta que esto sólo calcula el gini de los mineros que obtuvieron más del 1% de las recompensas del bloque en un día, de lo contrario, siempre irá a 1 debido a que muchos minan sólo 1 bloque.

In [15]:
```
query = """
#standardSQL
-- MIT License
-- Copyright (c) 2019 Yaz Khoury, yaz.khoury@gmail.com

WITH total_reward_book AS (
  SELECT miner,
    DATE(timestamp) as date,
    COUNT(miner) as total_block_reward
  FROM `bigquery-public-data.crypto_ethereum.blocks`
  GROUP BY miner, date
  HAVING COUNT(miner) > 100
),
total_reward_book_by_date AS (
 SELECT date,
        miner AS address,
        SUM(total_block_reward / POWER(10,0)) AS value
  FROM total_reward_book
  GROUP BY miner, date
),
daily_rewards_with_gaps AS (
  SELECT
    address,
    date,
    SUM(value) OVER (PARTITION BY ADDRESS ORDER BY date) AS block_rewards,
    LEAD(date, 1, CURRENT_DATE()) OVER (PARTITION BY ADDRESS ORDER BY date) AS next_date
  FROM total_reward_book_by_date
),
calendar AS (
  SELECT date
  FROM UNNEST(GENERATE_DATE_ARRAY('2015-07-30', CURRENT_DATE())) AS date
),
daily_rewards AS (
  SELECT address,
    calendar.date,
    block_rewards
  FROM daily_rewards_with_gaps
  JOIN calendar ON daily_rewards_with_gaps.date <= calendar.date
  AND calendar.date < daily_rewards_with_gaps.next_date
),
supply AS (
  SELECT date,
    SUM(block_rewards) AS total_rewards
  FROM daily_rewards
  GROUP BY date
),
ranked_daily_rewards AS (
  SELECT daily_rewards.date AS date,
    block_rewards,
```

```
    ROW_NUMBER() OVER (PARTITION BY daily_rewards.date ORDER BY block_rewards DESC) AS rank
  FROM daily_rewards
  JOIN supply ON daily_rewards.date = supply.date
  WHERE SAFE_DIVIDE(block_rewards, total_rewards) >= 0.01
  ORDER BY block_rewards DESC
),
daily_gini AS (
  SELECT date,
    -- (1 - 2B) https://en.wikipedia.org/wiki/Gini_coefficient
    1 - 2 * SUM((block_rewards * (rank - 1) + block_rewards / 2)) / COUNT(*) / SUM(block_reward
s) AS gini
  FROM ranked_daily_rewards
  GROUP BY DATE
)
SELECT date,
  gini,
  AVG(gini) OVER (ORDER BY date ASC ROWS 7 PRECEDING) AS gini_sma_7,
  AVG(gini) OVER (ORDER BY date ASC ROWS 30 PRECEDING) AS gini_sma_30
FROM daily_gini
ORDER BY date ASC
"""

query_job = client.query(query)
iterator = query_job.result()
```

In [16]:

```
rows = list(iterator)
# Transform the rows into a nice pandas dataframe
mining_reward_gini_by_date = pd.DataFrame(data=[list(x.values()) for x in rows], columns=list(r
ows[0].keys()))
mining_reward_gini_by_date.head(10)
```

Out[16]:

	date	gini	gini_sma_7	gini_sma_30
0	2015-07-30	0.465213	0.465213	0.465213
1	2015-07-31	0.524871	0.495042	0.495042
2	2015-08-01	0.586563	0.525549	0.525549
3	2015-08-02	0.576192	0.538210	0.538210
4	2015-08-03	0.514817	0.533531	0.533531
5	2015-08-04	0.514918	0.530429	0.530429
6	2015-08-05	0.497703	0.525754	0.525754
7	2015-08-06	0.520803	0.525135	0.525135
8	2015-08-07	0.511116	0.530873	0.523577
9	2015-08-08	0.502773	0.528111	0.521497

In [17]:

```
traces = []
x = mining_reward_gini_by_date['date']
gini_list = ['gini', 'gini_sma_7', 'gini_sma_30']
for gini in gini_list:
    y = mining_reward_gini_by_date[gini]
    trace = dict(
        x=x,
        y=y,

        mode='lines'
    )
    traces.append(trace)
fig = dict(data=traces)
```

```
iplot(fig, validate=False)
```

Top 20 rich list

In [18]:

```
query = """
with double_entry_book as (
    -- debits
    select to_address as address, value as value
    from `bigquery-public-data.crypto_ethereum.traces`
    where to_address is not null
    and status = 1
    and (call_type not in ('delegatecall', 'callcode', 'staticcall') or call_type is null)
    union all
    -- credits
    select from_address as address, -value as value
    from `bigquery-public-data.crypto_ethereum.traces`
    where from_address is not null
    and status = 1
    and (call_type not in ('delegatecall', 'callcode', 'staticcall') or call_type is null)
    union all
    -- transaction fees debits
    select miner as address, sum(cast(receipt_gas_used as numeric) * cast(gas_price as numeri
c)) as value
    from `bigquery-public-data.crypto_ethereum.transactions` as transactions
    join `bigquery-public-data.crypto_ethereum.blocks` as blocks on blocks.number = transaction
s.block_number
    group by blocks.miner
    union all
    -- transaction fees credits
    select from_address as address, -(cast(receipt_gas_used as numeric) * cast(gas_price as num
eric)) as value
    from `bigquery-public-data.crypto_ethereum.transactions`
)
select address,
sum(value) / 1000000000 as balance
from double_entry_book
group by address
order by balance desc
limit 20
```

```
limit 20
"""

query_job = client.query(query)
iterator = query_job.result()
```

In [19]:

```
rows = list(iterator)
# Transform the rows into a nice pandas dataframe
top_address_rich_list = pd.DataFrame(data=[list(x.values()) for x in rows], columns=list(rows[0
].keys()))
top_address_rich_list.head(10)
```

Out[19]:

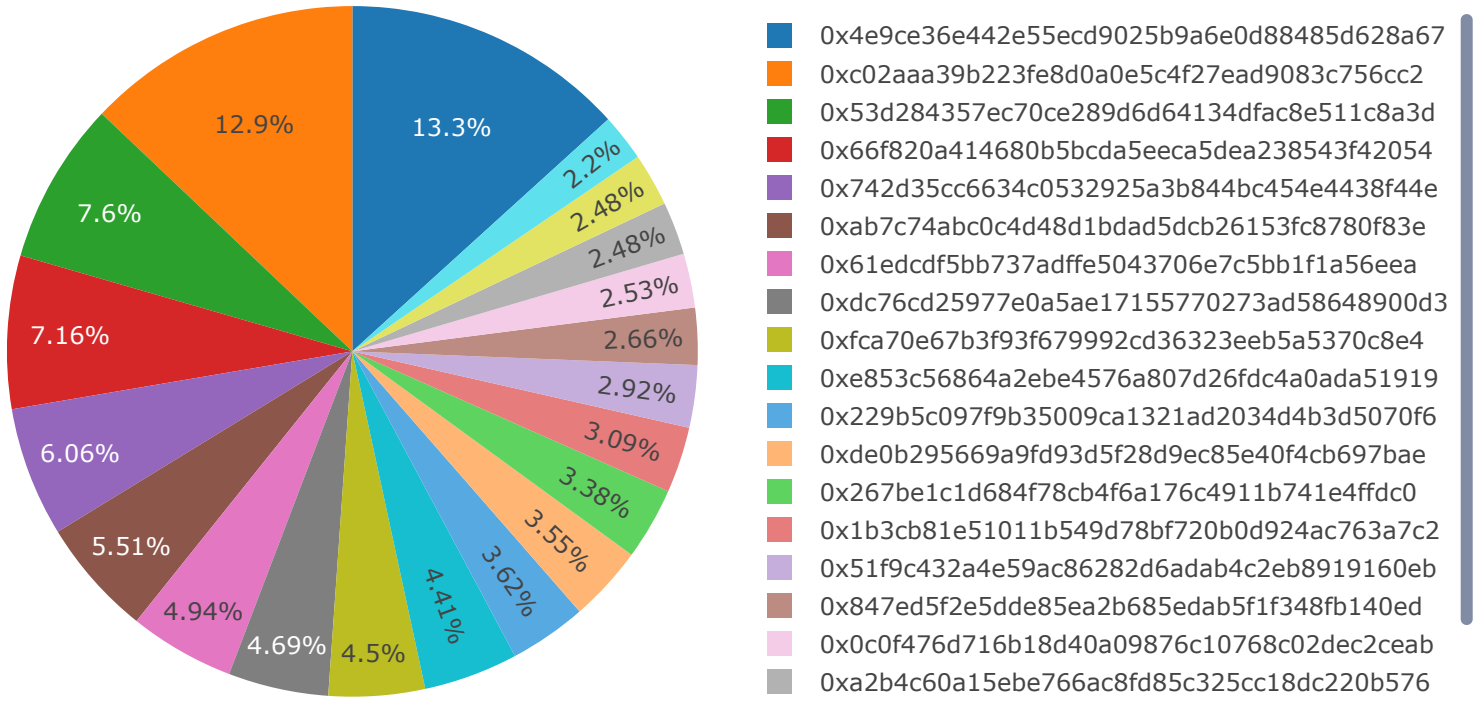
	address	balance
0	0x4e9ce36e442e55ecd9025b9a6e0d88485d628a67	2408262784824309.564346354
1	0xc02aaa39b223fe8d0a0e5c4f27ead9083c756cc2	2344759046598473.828182341
2	0x53d284357ec70ce289d6d64134dfac8e511c8a3d	1378754143068188.911481302
3	0x66f820a414680b5bcda5eeca5dea238543f42054	1300001608942567.945276777
4	0x742d35cc6634c0532925a3b844bc454e4438f44e	1100506156233832.5
5	0xab7c74abc0c4d48d1bdad5dcb26153fc8780f83e	999999012463125.37777777
6	0x61edcdf5bb737adffe5043706e7c5bb1f1a56eea	895999000010000
7	0xdc76cd25977e0a5ae17155770273ad58648900d3	850860641968887.01041
8	0xfca70e67b3f93f679992cd36323eeb5a5370c8e4	816999895751008.6
9	0xe853c56864a2ebe4576a807d26fdc4a0ada51919	801052799959724.594886

In [20]:

```
labels = top_address_rich_list['address']
values = top_address_rich_list['balance']

trace = go.Pie(labels=labels, values=values)

iplot([trace])
```



Daily Top Balance Gini Coefficient

Daily Top Balance Gini Coefficient

Ranking Diario del Balance de Coeficiente de Gini

Now, we will try getting daily top rich list from the genesis until now and then calculate the gini coefficient of the rich list.

Ahora, intentaremos conseguir la lista del ranking de los más ricos, desde el bloque génesis hasta ahora y después calcularemos el coeficiente Gini de dicha lista.

In this context, the gini coefficient will be a measure of income inequality among wallet addresses based on how much ether balance is in each wallet. Of course, this assumes 1 person = 1 wallet, but a person can have multiple wallets. It will also query for top 10k addresses to be used in gini analysis. That will include exchange account balances, which we don't take into account eliminating from the dataset.

En este contexto, el coeficiente Gini será una medida de la desigualdad de ingresos entre las direcciones de carteras virtuales, basándose en cuanto balance de Ethers hay en cada cartera. Por supuesto, se asume que cada cartera corresponde a una única persona, pero una persona puede disponer de múltiples carteras. También se harán consultas sobre las 10.000 direcciones más altas, que serán usadas en los análisis Gini. Esto incluye los balances de las cuentas de los Exchanges, que no tendremos en cuenta, eliminándolos del dataset.

The query was written by from Evegeny Medvedev and Allen Day for this Google Blog Post. I added some further analysis towards the end to measure the Simple Moving Average of the Gini for the past 7 and 30 days.

In [21]:

```
query = """
with
double_entry_book as (
    -- debits
    select to_address as address, value as value, block_timestamp
    from `bigquery-public-data.crypto_ethereum.traces`
    where to_address is not null
    and status = 1
    and (call_type not in ('delegatecall', 'callcode', 'staticcall') or call_type is null)
    union all
    -- credits
    select from_address as address, -value as value, block_timestamp
    from `bigquery-public-data.crypto_ethereum.traces`
    where from_address is not null
    and status = 1
    and (call_type not in ('delegatecall', 'callcode', 'staticcall') or call_type is null)
    union all
    -- transaction fees debits
    select miner as address, sum(cast(receipt_gas_used as numeric) * cast(gas_price as numeri
c)) as value, block_timestamp
    from `bigquery-public-data.crypto_ethereum.transactions` as transactions
    join `bigquery-public-data.crypto_ethereum.blocks` as blocks on blocks.number = transaction
s.block_number
    group by blocks.miner, block_timestamp
    union all
    -- transaction fees credits
    select from_address as address, -(cast(receipt_gas_used as numeric) * cast(gas_price as num
eric)) as value, block_timestamp
    from `bigquery-public-data.crypto_ethereum.transactions`
),
double_entry_book_by_date as (
    select
        date(block_timestamp) as date,
        address,
        sum(value / POWER(10,0)) as value
    from double_entry_book
    group by address, date
),
daily_balances_with_gaps as (
    select
        address,
        date,
        sum(value) over (partition by address order by date) as balance,
```

```
        lead(date, 1, current_date()) over (partition by address order by date) as next_date
        from double_entry_book_by_date
),
calendar as (
    select date from unnest(generate_date_array('2015-07-30', current_date())) as date
),
daily_balances as (
    select address, calendar.date, balance
    from daily_balances_with_gaps
    join calendar on daily_balances_with_gaps.date <= calendar.date and calendar.date < daily_b
alances_with_gaps.next_date
),
 supply as (
    select
        date,
        sum(balance) as daily_supply
    from daily_balances
    group by date
),
ranked_daily_balances as (
    select
        daily_balances.date,
        balance,
        row_number() over (partition by daily_balances.date order by balance desc) as rank
    from daily_balances
    join supply on daily_balances.date = supply.date
    where safe_divide(balance, daily_supply) >= 0.0001
    ORDER BY safe_divide(balance, daily_supply) DESC
),
gini_daily as (
   select
    date,
    -- (1 - 2B) https://en.wikipedia.org/wiki/Gini_coefficient
    1 - 2 * sum((balance * (rank - 1) + balance / 2)) / count(*) / sum(balance) as gini
  from ranked_daily_balances
  group by date
)
select date,
    gini,
    avg(gini) over (order by date asc rows 7 preceding) as gini_sma7,
    avg(gini) over (order by date asc rows 30 preceding) as gini_sma30
from gini_daily
order by date asc
"""

query_job = client.query(query)
iterator = query_job.result()
```

In [22]:

```
rows = list(iterator)
# Transform the rows into a nice pandas dataframe
daily_balance_gini = pd.DataFrame(data=[list(x.values()) for x in rows], columns=list(rows[0].k
eys()))
daily_balance_gini.head(10)
```

Out[22]:

	date	gini	gini_sma7	gini_sma30
0	2015-07-30	0.689518	0.689518	0.689518
1	2015-07-31	0.689473	0.689495	0.689495
2	2015-08-01	0.689358	0.689449	0.689449
3	2015-08-02	0.689275	0.689406	0.689406
4	2015-08-03	0.689175	0.689360	0.689360
5	2015-08-04	0.689103	0.689317	0.689317

6	2015-08-05	0.689097	0.689285	0.689285
7	2015-08-06	0.689067	0.689258	0.689258
8	2015-08-07	0.697556	0.690263	0.690180
9	2015-08-08	0.702019	0.691831	0.691364

In [23]:
```
traces = []
x = daily_balance_gini['date']
gini_list = ['gini', 'gini_sma7', 'gini_sma30']
for gini in gini_list:
    y = daily_balance_gini[gini]
    trace = dict(
        x=x,
        y=y,
        mode='lines'
    )
    traces.append(trace)
fig = dict(data=traces)

iplot(fig, validate=False)
```

Daily Hashrate

Hashrate is a measure of difficulty over block time. We can measure this by getting the delta time of each block timestamp from the previous block timestamp.

We can average it out by day. That is, the query can average out all difficulty and delta times per day and divide them by one another. We can further divide by 1 billion to get the GH/s.

We will use the following query I wrote for the Daily Hashrate.

```
#standardSQL
-- MIT License
-- Copyright (c) 2019 Yaz Khoury, yaz.khoury@gmail.com

WITH block_rows AS (
  SELECT *, ROW_NUMBER() OVER (ORDER BY timestamp) AS rn
```

```
  SELECT *, ROW_NUMBER() OVER (ORDER BY timestamp) AS rn
  FROM `bigquery-public-data.crypto_ethereum.blocks`
),
delta_time AS (
  SELECT
  mp.timestamp AS block_time,
  mp.difficulty AS difficulty,
  TIMESTAMP_DIFF(mp.timestamp, mc.timestamp, SECOND) AS delta_block_time
  FROM block_rows mc
  JOIN block_rows mp
  ON mc.rn = mp.rn - 1
),
hashrate_book AS (
  SELECT TIMESTAMP_TRUNC(block_time, DAY) AS block_day,
  AVG(delta_block_time) as daily_avg_block_time,
  AVG(difficulty) as daily_avg_difficulty
  FROM delta_time
  GROUP BY TIMESTAMP_TRUNC(block_time, DAY)
)
SELECT block_day,
(daily_avg_difficulty/daily_avg_block_time)/1000000000 as hashrate
FROM hashrate_book
ORDER BY block_day ASC
```

In [24]:

```
query = """
#standardSQL
-- MIT License
-- Copyright (c) 2019 Yaz Khoury, yaz.khoury@gmail.com

WITH block_rows AS (
  SELECT *, ROW_NUMBER() OVER (ORDER BY timestamp) AS rn
  FROM `bigquery-public-data.crypto_ethereum.blocks`
),
delta_time AS (
  SELECT
  mp.timestamp AS block_time,
  mp.difficulty AS difficulty,
  TIMESTAMP_DIFF(mp.timestamp, mc.timestamp, SECOND) AS delta_block_time
  FROM block_rows mc
  JOIN block_rows mp
  ON mc.rn = mp.rn - 1
),
hashrate_book AS (
  SELECT TIMESTAMP_TRUNC(block_time, DAY) AS block_day,
  AVG(delta_block_time) as daily_avg_block_time,
  AVG(difficulty) as daily_avg_difficulty
  FROM delta_time
  GROUP BY TIMESTAMP_TRUNC(block_time, DAY)
)
SELECT block_day,
(daily_avg_difficulty/daily_avg_block_time)/1000000000 as hashrate
FROM hashrate_book
ORDER BY block_day ASC
"""

query_job = client.query(query)
iterator = query_job.result()
```

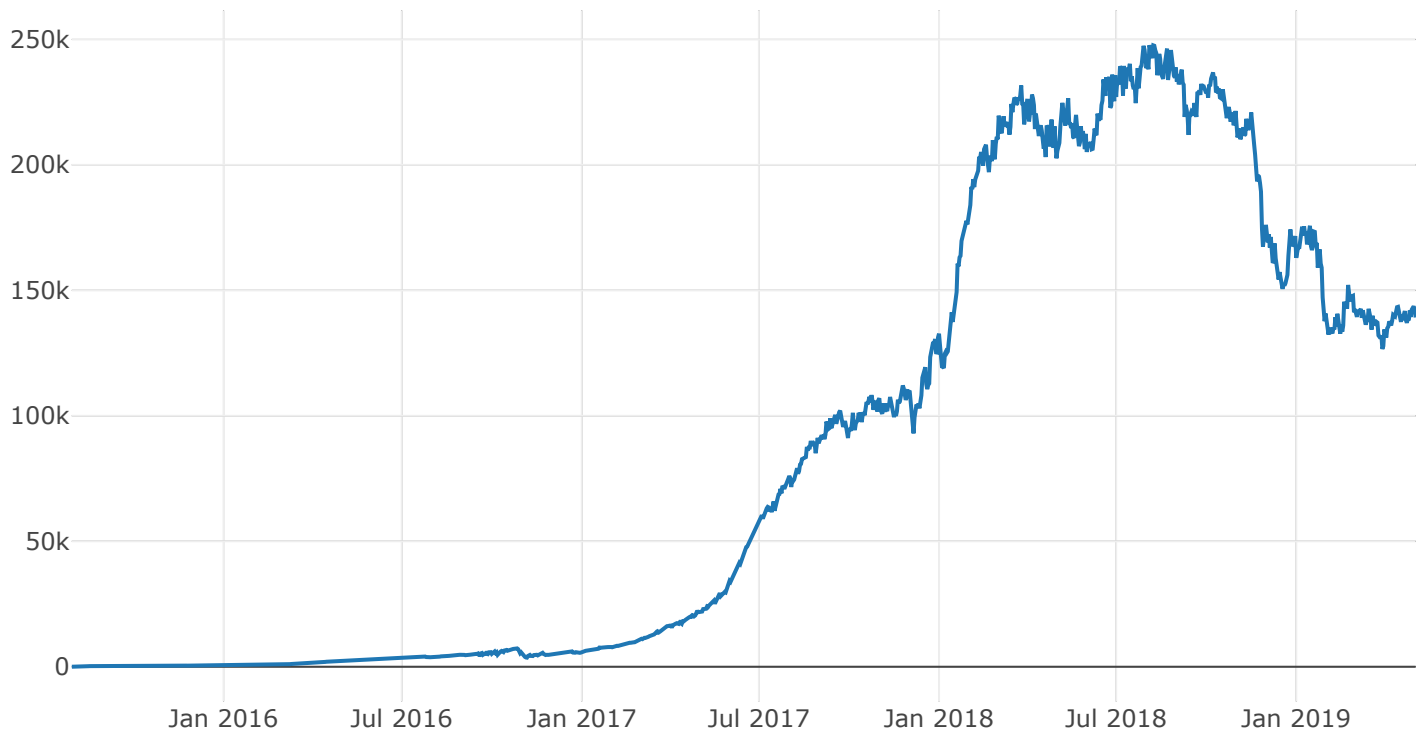
In [25]:

```
rows = list(iterator)
# Transform the rows into a nice pandas dataframe
daily_hashrate = pd.DataFrame(data=[list(x.values()) for x in rows], columns=list(rows[0].keys
()))
daily_hashrate.head(10)
```

Out[25]:

	block_day	hashrate
0	2015-07-30 00:00:00+00:00	0.000583
1	2015-07-31 00:00:00+00:00	47.939455
2	2015-08-01 00:00:00+00:00	54.303050
3	2015-08-02 00:00:00+00:00	63.291144
4	2015-08-03 00:00:00+00:00	68.788433
5	2015-08-04 00:00:00+00:00	75.580872
6	2015-08-05 00:00:00+00:00	80.448478
7	2015-08-06 00:00:00+00:00	81.771576
8	2015-08-07 00:00:00+00:00	88.339527
9	2015-08-08 00:00:00+00:00	96.474665

In [26]:
```
trace = go.Scatter(
    x=daily_hashrate['block_day'],
    y=daily_hashrate['hashrate'],
    mode='lines'
)
data = [trace]
iplot(data)
```



Segunda Parte. Add Market Data from Coinbase and Twitter stats from the Brainrex API

Para esta parte necesitas sacar una API key en https://console.brainrex.com/register (https://console.brainrex.com/register)

In [27]:
```
train = pd.read_csv('..input/etherclose/eth-24.csv')
```

```
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-27-8df84df731e2> in <module>()
----> 1 train = pd.read_csv('..input/etherclose/eth-24.csv')

/opt/conda/lib/python3.6/site-packages/pandas/io/parsers.py in parser_f(filepath_or_buffer, se
p, delimiter, header, names, index_col, usecols, squeeze, prefix, mangle_dupe_cols, dtype, engi
ne, converters, true_values, false_values, skipinitialspace, skiprows, nrows, na_values, keep_d
efault_na, na_filter, verbose, skip_blank_lines, parse_dates, infer_datetime_format, keep_date_
col, date_parser, dayfirst, iterator, chunksize, compression, thousands, decimal, lineterminato
r, quotechar, quoting, escapechar, comment, encoding, dialect, tupleize_cols, error_bad_lines,
 warn_bad_lines, skipfooter, doublequote, delim_whitespace, low_memory, memory_map, float_preci
sion)
    676                     skip_blank_lines=skip_blank_lines)
    677
--> 678         return _read(filepath_or_buffer, kwds)
    679
    680     parser_f.__name__ = name

/opt/conda/lib/python3.6/site-packages/pandas/io/parsers.py in _read(filepath_or_buffer, kwds)
    438
    439     # Create the parser.
--> 440     parser = TextFileReader(filepath_or_buffer, **kwds)
    441
    442     if chunksize or iterator:

/opt/conda/lib/python3.6/site-packages/pandas/io/parsers.py in __init__(self, f, engine, **kwd
s)
    785             self.options['has_index_names'] = kwds['has_index_names']
    786
--> 787         self._make_engine(self.engine)
    788
    789     def close(self):

/opt/conda/lib/python3.6/site-packages/pandas/io/parsers.py in _make_engine(self, engine)
```


Data

Data Sources

- [Private Dataset]
- Ethereum Blockchain
  - blocks — 18 columns
  - contracts — 8 columns
  - logs — 9 columns
  - token_transfers — 9 columns
  - tokens — 5 columns
  - traces — 19 columns
  - transactions — 17 columns
- Ethereum Classic Blockchain
  - blocks — 18 columns
  - contracts — 8 columns

[Private Dataset]

Access to this Dataset is restricted.
You are seeing this placeholder because you have access to the Kernel.

logs	9 columns
token_transfers	9 columns
tokens	5 columns
traces	19 columns
transactions	17 columns

Comments (0)

Click here to comment...

Similar Kernels