LiamLarsen

Analyzing the Ethereum Blockchain

last run 2 months ago · IPython Notebook HTML · 6,693 views
using data from Ethereum Historical Data · Public

13
voters

Notebook

Analyzing the Ethereum Network

In [1]:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import datetime as dt # converting timestamp to date
import seaborn as sns # Visualizer for data
import matplotlib.pyplot as plt # matplot
import matplotlib.dates as mdates # for plotting dates
%matplotlib inline
from subprocess import check_output
# What are we working with?
print('File: \n', check_output(["ls", "../input"]).decode("utf8"))
```

```
File:
 EtherMarketCapChart.csv
EtherPriceHistory(USD).csv
EtherSupplyGrowthChart.csv
EthereumBlockDifficultyGrowth.csv
EthereumBlockRewardsChart.csv
EthereumBlockSizeHistory.csv
EthereumChainDataSizeGrowth(FASTSync).csv
EthereumChainDataSizeGrowth(FULLSync).csv
EthereumDailyGasUsedHistory.csv
EthereumGasLimitHistory.csv
EthereumGasPriceHistory.csv
EthereumNetworkHashRateGrowthRate.csv
EthereumTransactionFee.csv
```

```
EthereumTransactionHistory.csv
EthereumUnclesCount.csv
EthereumUniqueAddressGrowthRate.csv
```

We are going to take the CSVs one at a time and create a DataFrame for each of them

In [2]:

```
# Input files i'm using
address_gr = pd.read_csv('../input/EthereumUniqueAddressGrowthRate.csv')
blocksize_hist = pd.read_csv('../input/EthereumBlockSizeHistory.csv')
etherprice_usd = pd.read_csv('../input/EtherPriceHistory(USD).csv')
hashrate_gr = pd.read_csv('../input/EthereumNetworkHashRateGrowthRate.csv')
marketcap = pd.read_csv('../input/EtherMarketCapChart.csv')
tx_hist = pd.read_csv('../input/EthereumTransactionHistory.csv')
# Going to iterate and plot everything, except those with abnormalities
things_to_plot = [(blocksize_hist,"Blocksize History"),
                  (etherprice_usd, "Etherprice - USD"),
                  (hashrate_gr,"Hashrate Growth Rate"),
                # (address_gr, "Address Growth Rate"),
                # (marketcap, "Market Capital"),
                  (tx_hist, "Transaction History")]
```

This function was to convert the timestamps, but they no longer use them.

In [3]:

```
# the timestamp in the method is a dataframe column
# it returns a list of the format which can then be ploted if needed
def timeConvert(timestamps):
    timeValue = list(range(len(timestamps)))
    for i in range(len(timestamps)):
        timeValue[i] = (dt.datetime.fromtimestamp(timestamps[i]).strftime('%Y-%m-%d'))
    return timeValue;
```

Price plot

Note: they are USD

In [4]:

```
# Lets see:
print(marketcap.columns)
```

```
Index(['Date(UTC)', 'UnixTimeStamp', 'Supply', 'MarketCap', 'Price'], dtype='object')
```

1. # Function to plot data #

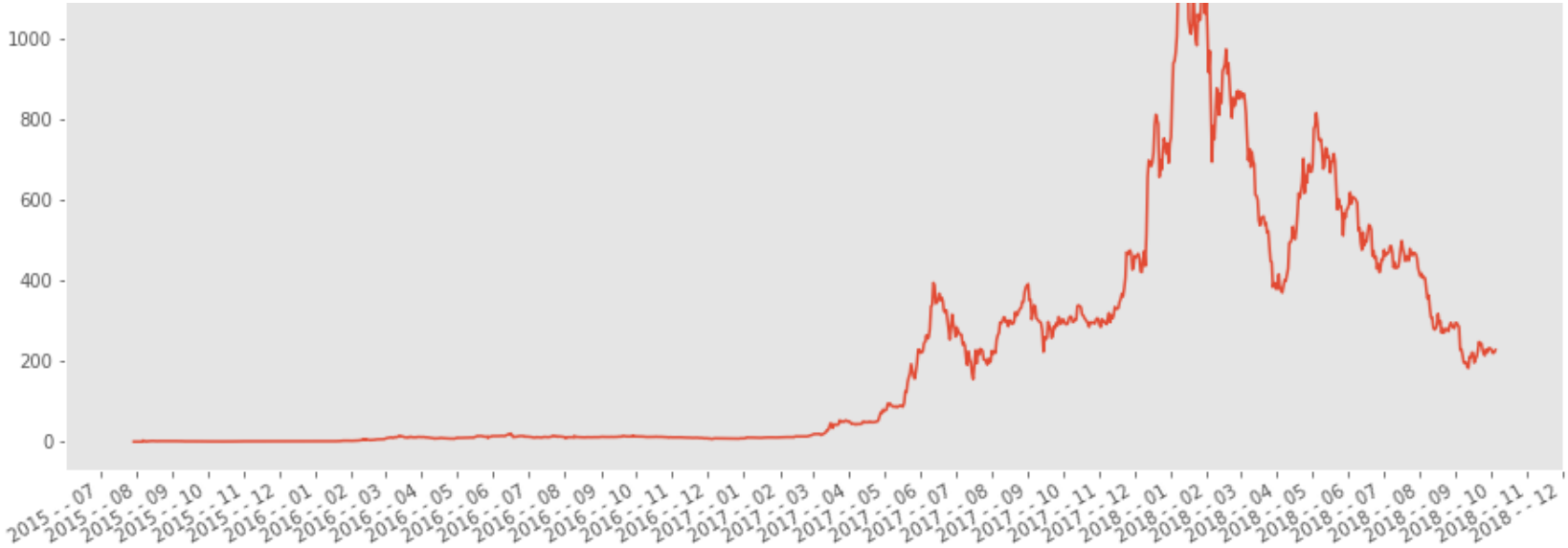
In [5]:

```
def plotit(data, title):
    # makes numpy array
    r = data.values#.view(np.recarray)
    #grab dates - convert to format
    date_df = r[:,0]
    date_df = pd.to_datetime(date_df)
```

```
    #grab values
    value_df = r[:,2]
    # make new plots
    plt.style.use('ggplot')
    fig, ax = plt.subplots(figsize=(15, 7))
    ax.set_title(title)
    ax.plot(date_df, value_df)
    ax.grid(False)
    # matplotlib date format object
    hfmt = mdates.DateFormatter('%Y - - %m')
    # format the ticks
    ax.xaxis.set_major_locator(mdates.MonthLocator())
    ax.xaxis.set_major_formatter(hfmt)
    # format the coords message box
    def yvals(x):
        return '$%1.2f' % x
    ax.format_xdata = hfmt
    ax.format_ydata = yvals
    # rotates and right aligns the x labels, and moves the bottom of the
    # axes up to make room for them
    fig.autofmt_xdate()
    plt.show()
```

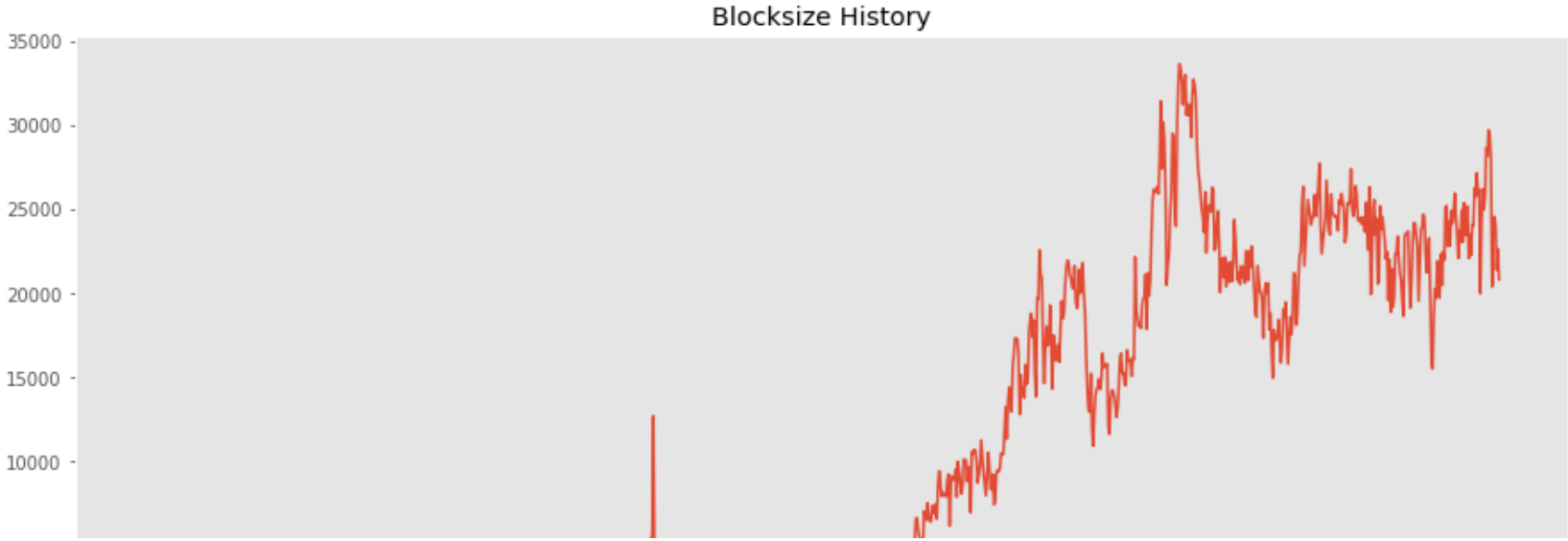
In [6]:

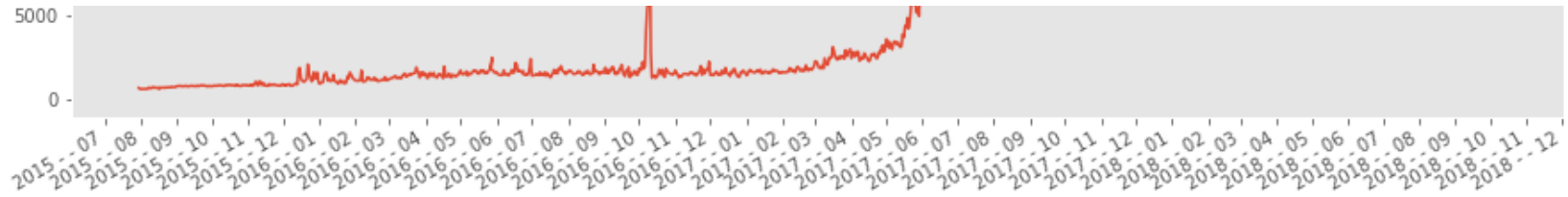
```
plotit(etherprice_usd, "Etherprice - USD")
```

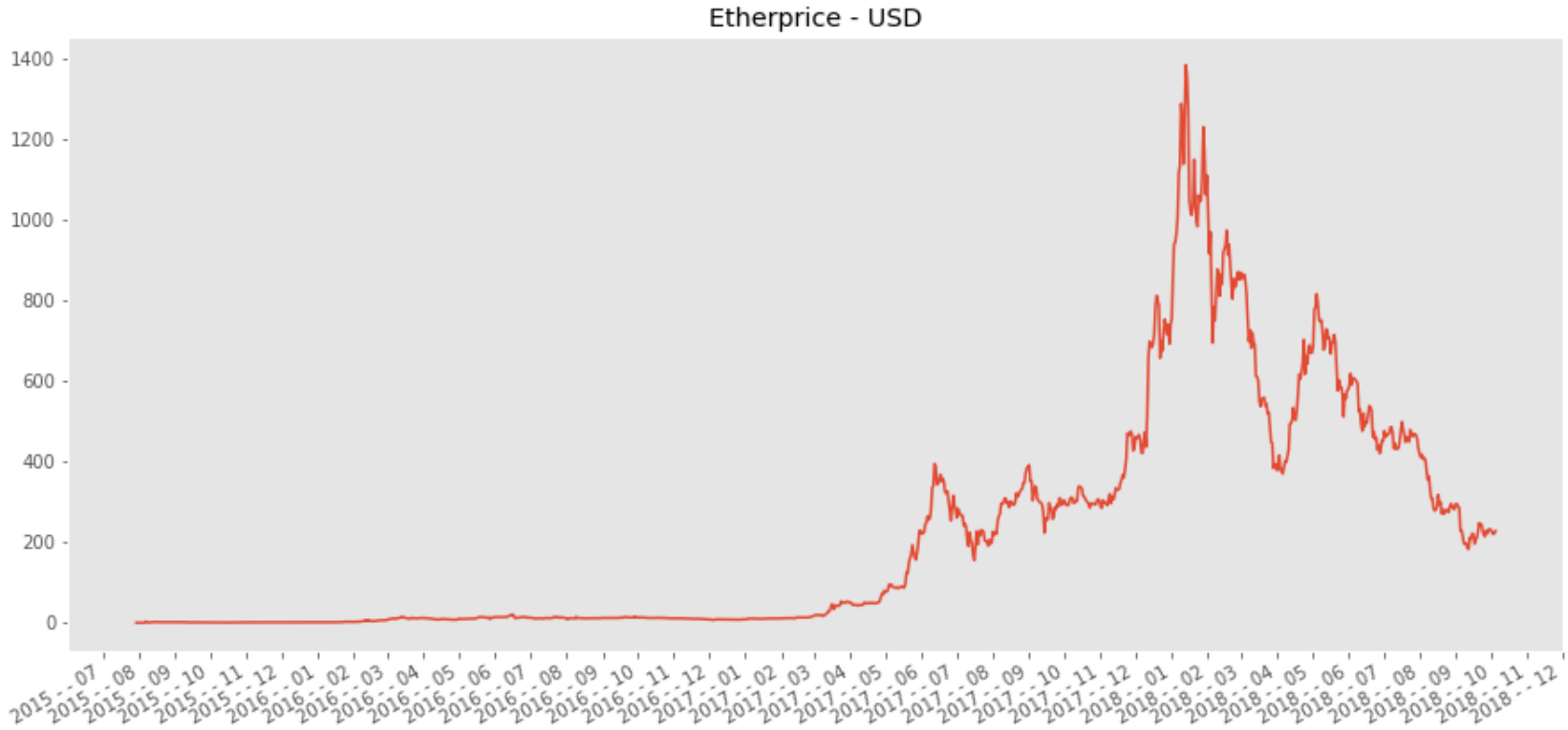


In [7]:

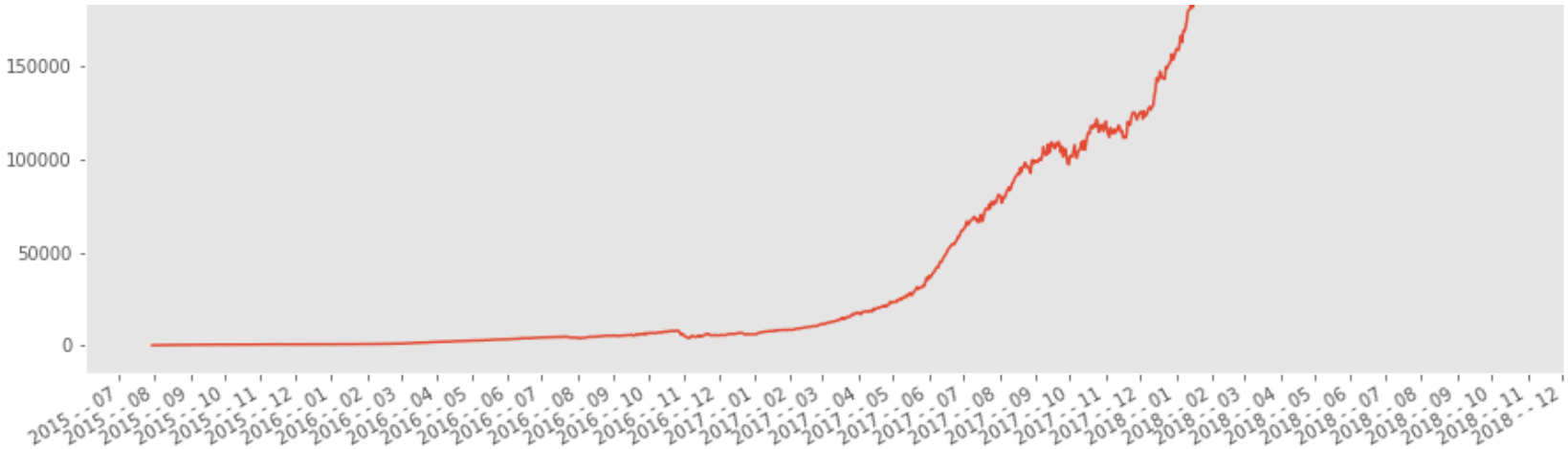
```
for plot,title in things_to_plot:
    plotit(plot, title)
```

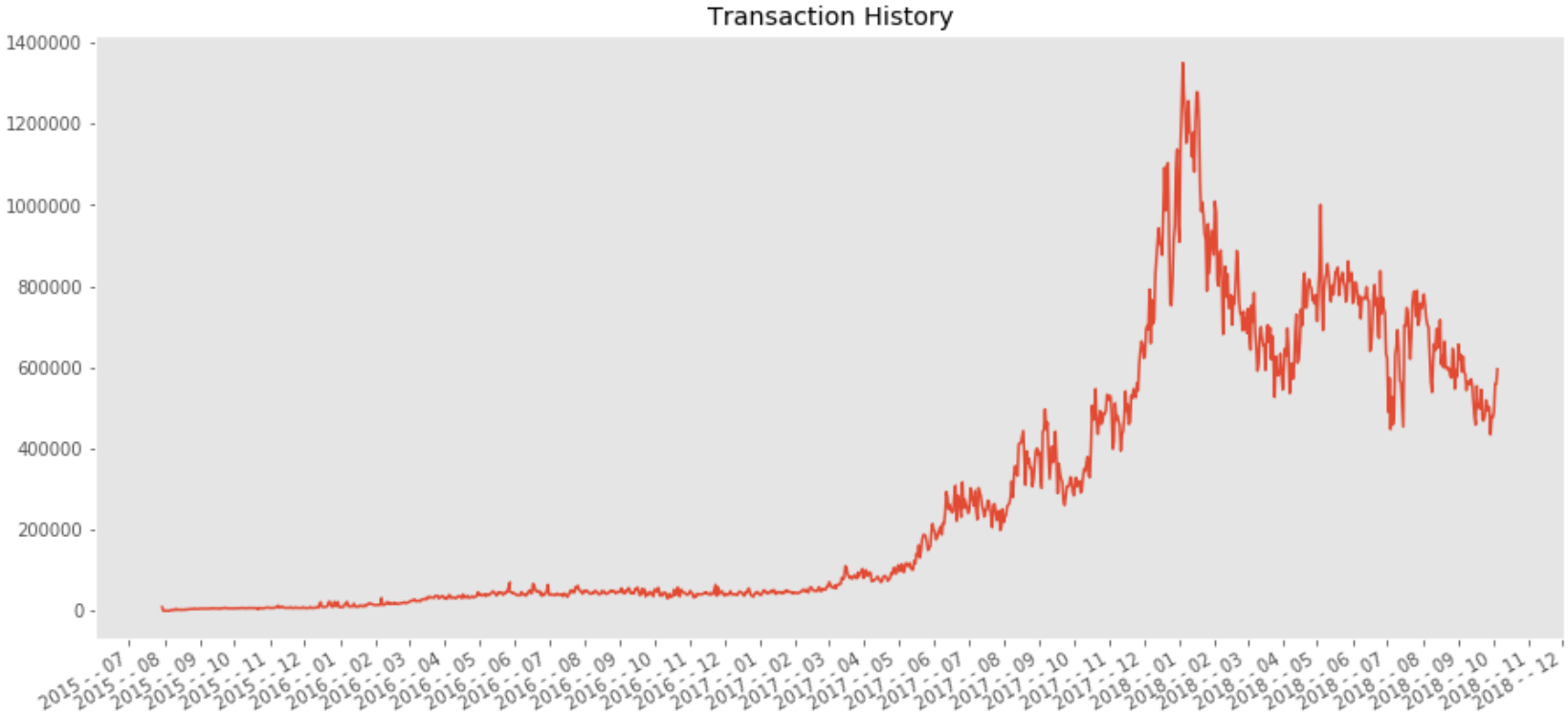



5000
0
2015-07 2015-08 2015-09 2015-10 2015-11 2015-12 2016-01 2016-02 2016-03 2016-04 2016-05 2016-06 2016-07 2016-08 2016-09 2016-10 2016-11 2016-12 2017-01 2017-02 2017-03 2017-04 2017-05 2017-06 2017-07 2017-08 2017-09 2017-10 2017-11 2017-12 2018-01 2018-02 2018-03 2018-04 2018-05 2018-06 2018-07 2018-08 2018-09 2018-10 2018-11 2018-12


Etherprice - USD
1400
1200
1000
800
600
400
200
0
2015-07 2015-08 2015-09 2015-10 2015-11 2015-12 2016-01 2016-02 2016-03 2016-04 2016-05 2016-06 2016-07 2016-08 2016-09 2016-10 2016-11 2016-12 2017-01 2017-02 2017-03 2017-04 2017-05 2017-06 2017-07 2017-08 2017-09 2017-10 2017-11 2017-12 2018-01 2018-02 2018-03 2018-04 2018-05 2018-06 2018-07 2018-08 2018-09 2018-10 2018-11 2018-12


Hashrate Growth Rate
300000
250000
200000


150000
100000
50000
0


Transaction History
1400000
1200000
1000000
800000
600000
400000
200000
0

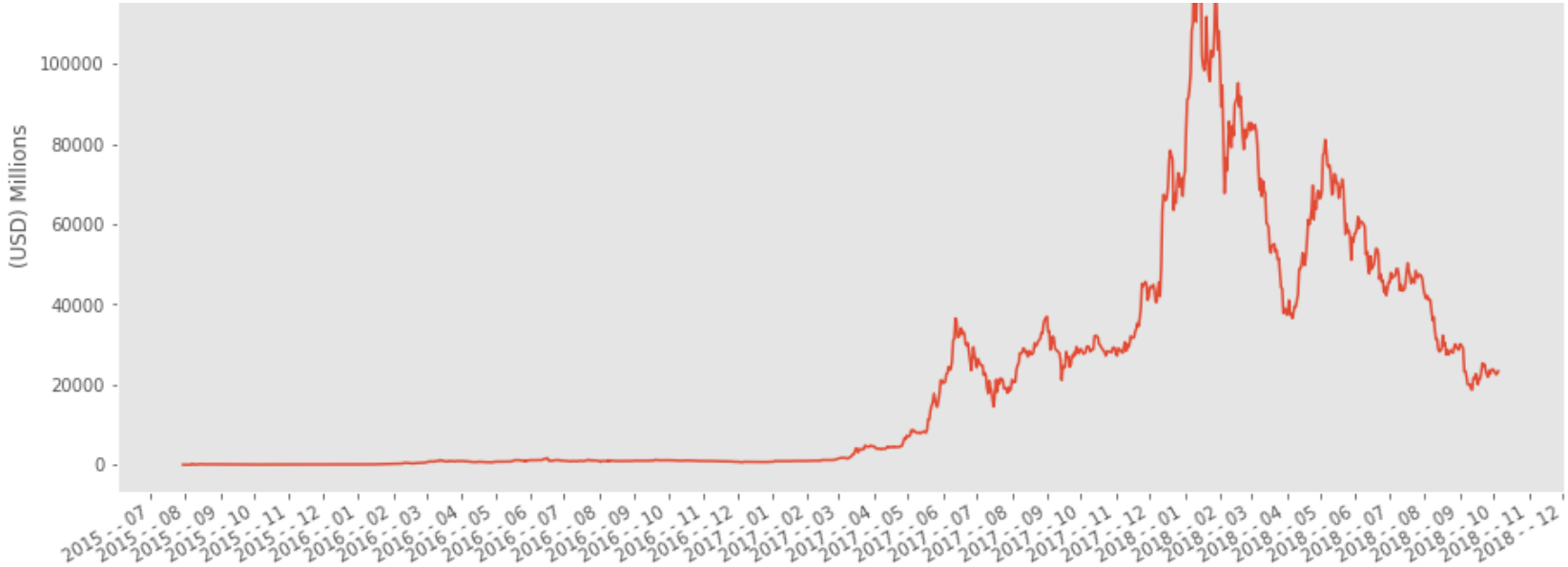
Plot the things we missed

In [8]:

```
mkp = marketcap.values#.view(np.recarray)

date_df = mkp[:,0]
date_df = pd.to_datetime(date_df)
value_df = mkp[:,3]
prices_df = mkp[:,4]

plt.style.use('ggplot')
fig, ax = plt.subplots(figsize=(15, 7))
ax.set_title("Market Capital")
ax.set_ylabel("(USD) Millions")
ax.plot(date_df, value_df)
ax.grid(False)
# Format dates
hfmt = mdates.DateFormatter('%Y - - %m')
ax.xaxis.set_major_locator(mdates.MonthLocator())
ax.xaxis.set_major_formatter(hfmt)
def yvals(x):
    return '$%1.2f' % x
ax.format_xdata = hfmt
ax.format_ydata = yvals
fig.autofmt_xdate()
plt.show()
```



In [9]:

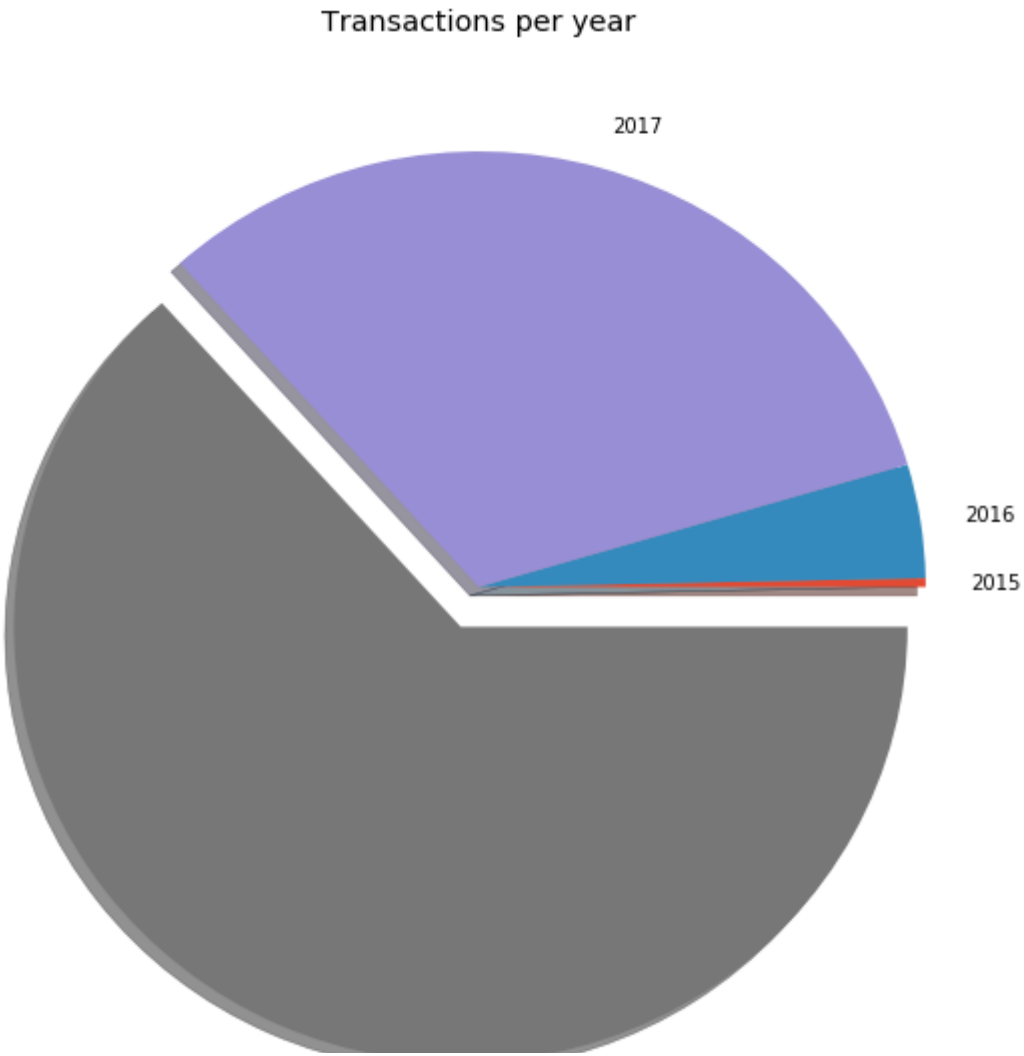
```
txs = tx_hist.copy()
txs['Date(UTC)'] = pd.to_datetime(txs['Date(UTC)']).dt.year
#txs['Date(UTC)'] = txs['Date(UTC)'].dt.year
txs = txs.groupby('Date(UTC)')['Value'].apply(lambda x: (x.unique().sum()))
txs
```

Out[9]:

```
Date(UTC)
2015      1048393
2016     13500530
2017    102941034
2018    202968209
Name: Value, dtype: int64
```

In [10]:

```
fig, ax = plt.subplots(figsize=(10, 10))
shap = txs
labels = '2015','2016','2017','2018'
explode = (0, 0, 0, 0.1)
ax.pie(shap, explode=explode, labels=labels, shadow=True)
plt.title('Transactions per year')
plt.show()
```



Good stuff, obviously there is so much more we could do with this

This is one of my first ML applications and it was years ago

Don't mind it, I was only 15

Machine learning (can't use kaggle):

```
from keras.layers.core import Dense, Activation, Dropout
from keras.layers.recurrent import LSTM
from keras.models import Sequential
import lstm, time #helper libraries
```

Using TensorFlow backend.

```
# Load data
X_train, y_train, X_test, y_test = lstm.load_data('../input/ether.csv', 30, True)


# Build model
model = Sequential()

model.add(LSTM(
    input_dim=1
```

```
    input_dim=1,
    output_dim=30,
    return_sequences=True,))
model.add(Dropout(0.14))

model.add(LSTM(
    100,
    return_sequences=False))
model.add(Dropout(0.14))
model.add(Dense(
    output_dim=1))
model.add(Activation('linear'))

start = time.time()
model.compile(loss='mse', optimizer='rmsprop')
print('Compilation time: ', time.time()-start)
```

Compilation time: 0.010754108428955078

```
# Train model
model.fit(
    X_train,
    y_train,
    batch_size=512,
    nb_epoch=10,
    validation_split=0.05)
```

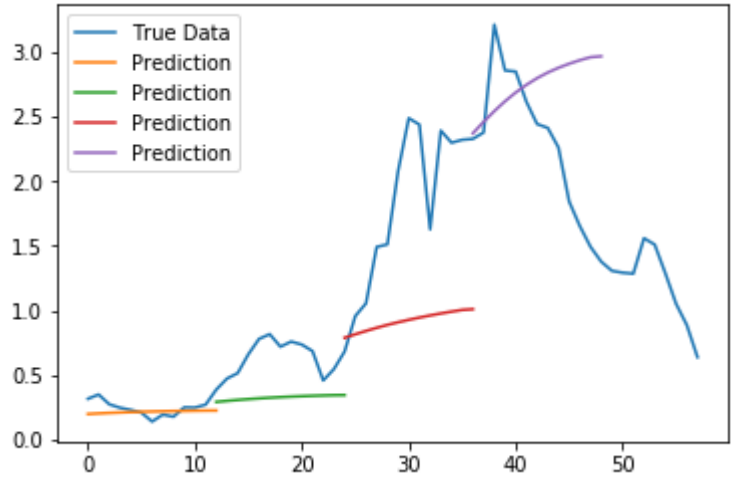
**Train on 498 samples, validate on 27 samples Epoch 1/10 498/498 [==============================] - 0s - loss: 0.6696 - val_loss: 1.0974 Epoch 2/10 498/498 [==============================] - 0s - loss: 0.4631 - val_loss: 0.8331 Epoch 3/10 498/498 [==============================] - 0s - loss: 0.3102 - val_loss: 0.5372 Epoch 4/10 498/498 [==============================] - 0s - loss: 0.1673 - val_loss: 0.3015 Epoch 5/10 498/498

[==========] - 0s - loss: 0.1224 - val_loss: 0.4458 Epoch 6/10 498/498
[==========] - 0s - loss: 0.1267 - val_loss: 0.2709 Epoch 7/10 498/498
[==========] - 0s - loss: 0.1037 - val_loss: 0.3876 Epoch 8/10 498/498
[==========] - 0s - loss: 0.1067 - val_loss: 0.2650 Epoch 9/10 498/498
[==========] - 0s - loss: 0.0929 - val_loss: 0.3726 Epoch 10/10 498/498
[==========] - 0s - loss: 0.0967 - val_loss: 0.2667

<keras.callbacks.History at 0x7fe404259e48>**

```
# Try to predict
predictions = lstm.predict_sequences_multiple(model, X_test, 11, 13)
lstm.plot_results_multiple(predictions, y_test, 12)
```

Amount preedicted: 58 with true data: 58 , and prediction length: 12



Obviously not the best...

Did you find this Kernel useful?
Show your appreciation with an upvote

13

Comments (11)

All Comments | Sort by Hotness

Please sign in to leave a comment.

Matthew Callens • Posted on Version 21 • a year ago • Options

1

I was just looking through the data set and noticed that for every entry/date, the values of the total growth and market cap are identical. Is this an error?

anokas • Posted on Version 21 • a year ago • Options

1

Really nice dataset and notebook :) I was looking for a dataset like this

LiamLarsen Kernel Author • Posted on Version 21 • a year ago • Options

0

Dude Anokas, awesome to see you on one of my datasets. Enjoy :)

Amar Patel • Posted on Version 21 • 2 years ago • Options

0

How are you pulling in Ethereum data? Can you pull in minute-level or 15-minute candle stick level data?

LiamLarsen Kernel Author • Posted on Version 21 • 2 years ago • Options 0

I theoretically have it by the hour, though posting it that often on kaggle would be tedious. I use etherscan API. :)

wgkkaggle3 • Posted on Version 21 • a year ago • Options 0

I'm very interested in what you've done here. can you post the code used to generate these csv files from etherscan API?

Josh Miller • Posted on Version 21 • a year ago • Options 0

Was interested in the same thing. After a bit of searching, I finally figured it out.
If you go here: https://etherscan.io/charts
and then click any of the charts, you can download a csv file with the data for each of the values. I'm assuming the 'all data' csv is a compilation of a few of those listed on etherscan.

TravisRivera • Posted on Version 21 • a year ago • Options 0

Thanks for the dataset!

Muhammad Salek ... • Posted on Version 21 • 10 months ago • Options

0

Great dataset! Thanks!

Ruben Alvarado • Posted on Version 21 • 9 months ago • Options

0

Thanks for the dataset !!

George Siaminos • Posted on Version 21 • 6 months ago • Options

0

Thank u very useful