HODL-AI: Live Coding RNN on the Blockchain

Machine Learning Mega Bash

2018/04/23 - Blackbox, Belfast

This presentation and associated files are available at present.bolster.online

Who are ya?

- Andrew Bolster (@bolster)
- MEng Elec. & SW Eng. QUB
- PhD Autonomous Systems UoL
- Data Scientist at Alert Logic
- Director at Farset Labs

Whats the craic?

- Garth got me in a moment of weakness and I spouted a load of BS clickbait buzzwords at him. Seemed like a good idea at the time.
- AI / ML is a load of FUD, but sometimes it's worth going through with a bad idea to get an understanding of why the decisions we make in ML pipelines massively change the outcomes.

Caveats

- Bitcoin Valuations are BS and are the definition of an irrational market; the idea that a simple one notebook ML model would in any way accurately predict future variations is similarly BS.
- The purpose of this talk is to explore timeseries analysis using a Python/ScikitLearn/Keras stack, not to make you (or me) rich.
- I guarantee I will mess up at some point(s).
- **Spoilers**: This method does not work because of the simplicity of the networks used, instead I want to show the 'beginning' of a model search, not the answers.

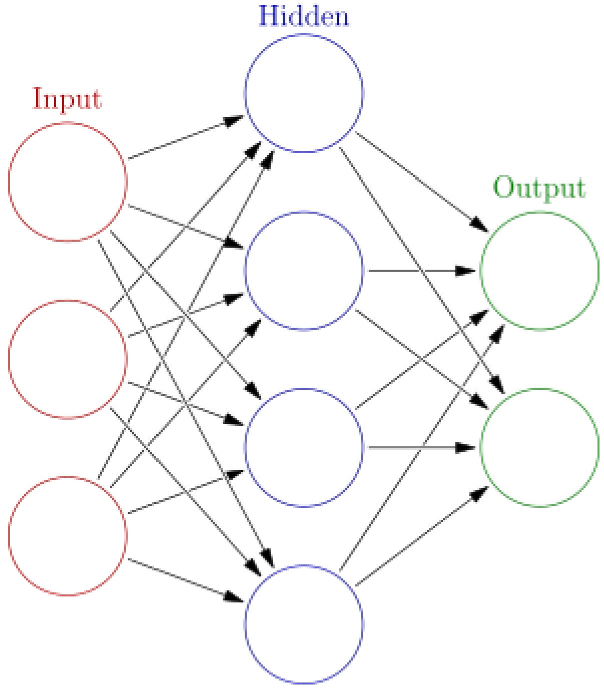
Background Info

What is Blockchain?

- A consensus based distributed ledger with (mostly) guaranteed proof of work.
- A fantastic solution in search of a problem.

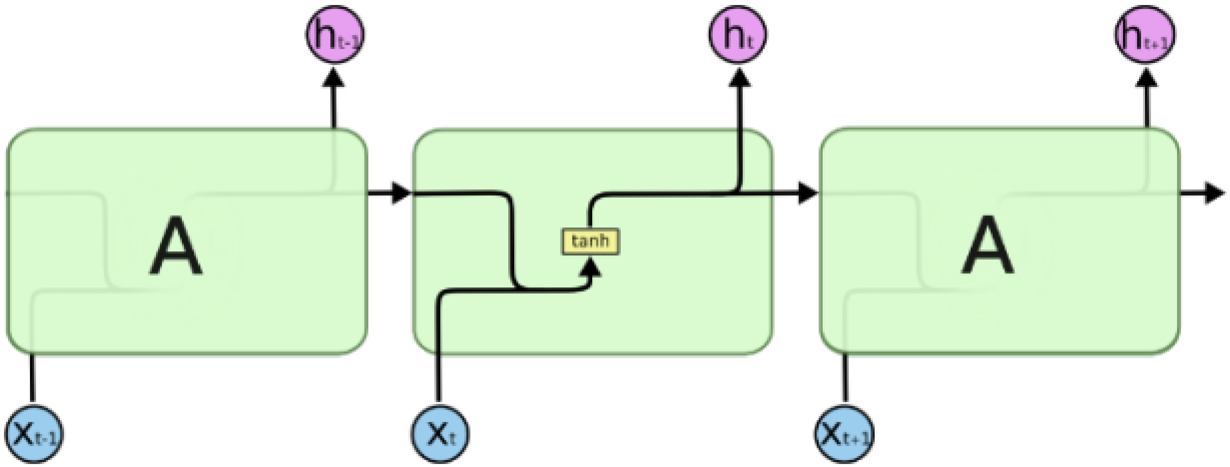
What are Neural Networks

- Simple cells connected together in particular ways to enable learning of abstract input/output mappings



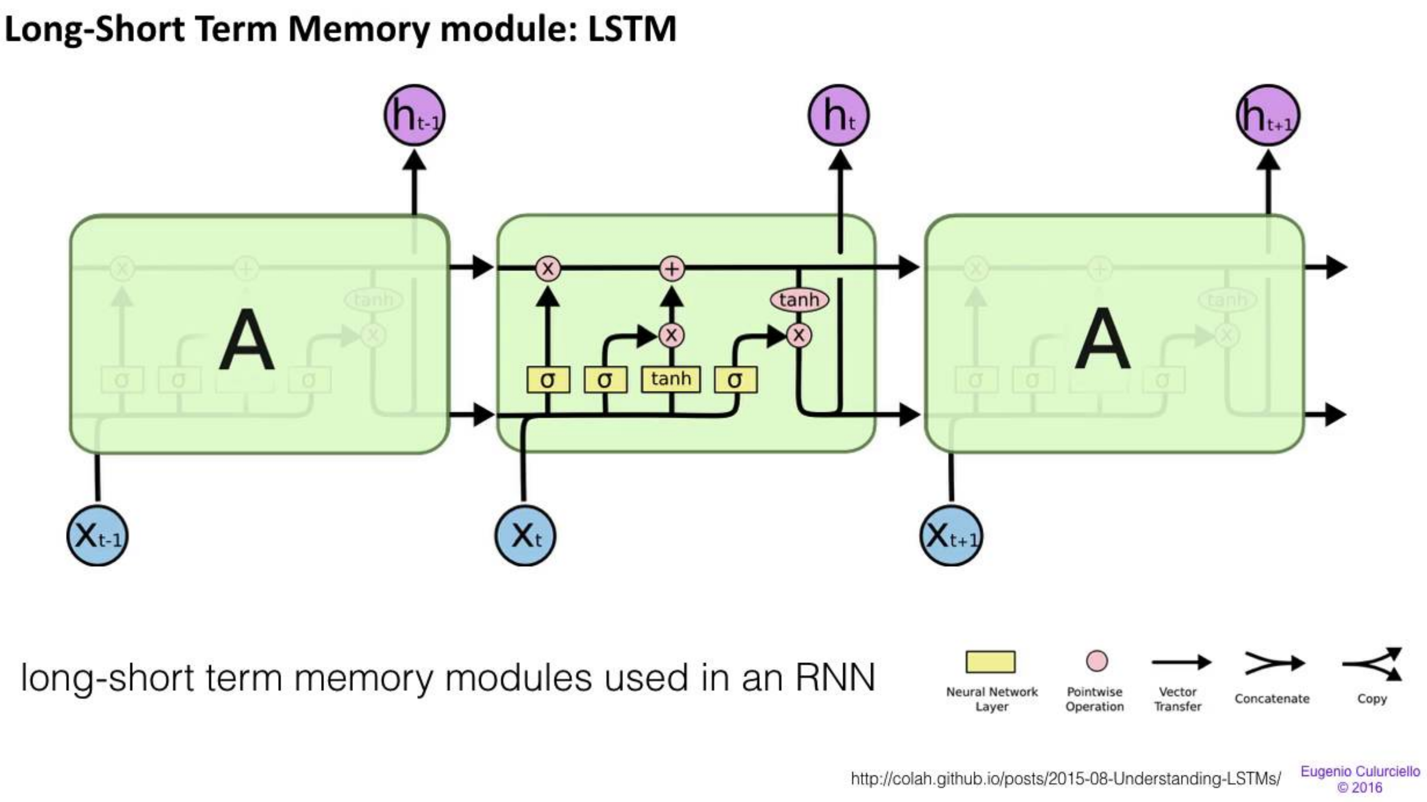
What are Recurrent Neural Networks

- Separated dimensionality of input/output (usually time)
- Changes process from 'state analysis' to 'sequence analysis'
- Can be 1-1/1-/-1/-



What are Long Short-Term Memory cells?

- They so fancy!
- Input-Output-Forget
- Corrects for vanishing gradient problems



What is Keras?

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.

Requirements

In [126]:

```python
requirements="autopep8 gdax pandas numpy cufflinks sklearn keras keras-tqdm"
# autopep8 is to make the jupyter notebook pretty
# gdax does bitcoin
# pandas numpy sklearn keras for data and machine learning
# See https://andrewbolster.info/2017/10/my-basic-python-data-science-setup
import pip
if 0== pip.main(f'install -q {requirements}'.split()):
    print("Requirements satisfied")
```

```
Requirements satisfied
```

Zee Plan

Part The First - It's The Data, Stupid

- Data Extraction (Collection/Acquisition/Ingestion)
- Data Transformation (Cleaning/Sanitising)

Part The Second - Prepare to Fail, Fail to Prepare

- Problem Transformation to a Supervised Learning Problem
- Scaling and Activation
- Basic Single Layer Univariate LSTM
- Performance Review

It’s the data, stupid

Where does it come from? What format is it in?

Chasing the blockchain

- `gdax-python` (https://github.com/danpaquin/gdax-python) is awesome
- Great Hackernoon Writeup (https://hackernoon.com/beginners-guide-to-gdax-an-exchange-of-coinbase-to-trade-btc-eth-and-ltc-e418fd1acd1b)
- Allows programmatic trading.
- But we're not doing that today; KISS

```
In [129]: # Get products list from gdax public client
          import gdax
          public_client = gdax.PublicClient()
          public_client.get_products()
```


```
Out[129]: [{'id': 'BCH-BTC',
            'base_currency': 'BCH',
            'quote_currency': 'BTC',
            'base_min_size': '0.01',
            'base_max_size': '200',
            'quote_increment': '0.00001',
            'display_name': 'BCH/BTC',
            'status': 'online',
            'margin_enabled': False,
            'status_message': None,
            'min_market_funds': '0.001',
            'max_market_funds': '30',
            'post_only': False,
            'limit_only': False,
            'cancel_only': False},
           {'id': 'BCH-USD',
            'base_currency': 'BCH',
            'quote_currency': 'USD',
            'base_min_size': '0.01',
            'base_max_size': '350',
            'quote_increment': '0.01',
            'display_name': 'BCH/USD',
            'status': 'online',
            'margin_enabled': False,
            'status_message': None,
            'min_market_funds': '10',
            'max_market_funds': '1000000',
            'post_only': False,
            'limit_only': False,
            'cancel_only': False},
           {'id': 'BTC-EUR',
```

```
  'base_currency': 'BTC',
  'quote_currency': 'EUR',
  'base_min_size': '0.001',
  'base_max_size': '50',
  'quote_increment': '0.01',
  'display_name': 'BTC/EUR',
  'status': 'online',
  'margin_enabled': False,
  'status_message': None,
  'min_market_funds': '10',
  'max_market_funds': '600000',
  'post_only': False,
  'limit_only': False,
  'cancel_only': False},
 {'id': 'BTC-GBP',
  'base_currency': 'BTC',
  'quote_currency': 'GBP',
  'base_min_size': '0.001',
  'base_max_size': '20',
  'quote_increment': '0.01',
  'display_name': 'BTC/GBP',
  'status': 'online',
  'margin_enabled': False,
  'status_message': None,
  'min_market_funds': '10',
  'max_market_funds': '200000',
  'post_only': False,
  'limit_only': False,
  'cancel_only': False},
 {'id': 'BTC-USD',
  'base_currency': 'BTC',
  'quote_currency': 'USD',
  'base_min_size': '0.001',
  'base_max_size': '70',
  'quote_increment': '0.01',
  'display_name': 'BTC/USD',
  'status': 'online',
  'margin_enabled': False,
```

```
  'status_message': None,
  'min_market_funds': '10',
  'max_market_funds': '1000000',
  'post_only': False,
  'limit_only': False,
  'cancel_only': False},
 {'id': 'ETH-BTC',
  'base_currency': 'ETH',
  'quote_currency': 'BTC',
  'base_min_size': '0.01',
  'base_max_size': '600',
  'quote_increment': '0.00001',
  'display_name': 'ETH/BTC',
  'status': 'online',
  'margin_enabled': False,
  'status_message': None,
  'min_market_funds': '0.001',
  'max_market_funds': '50',
  'post_only': False,
  'limit_only': False,
  'cancel_only': False},
 {'id': 'ETH-EUR',
  'base_currency': 'ETH',
  'quote_currency': 'EUR',
  'base_min_size': '0.01',
  'base_max_size': '400',
  'quote_increment': '0.01',
  'display_name': 'ETH/EUR',
  'status': 'online',
  'margin_enabled': False,
  'status_message': None,
  'min_market_funds': '10',
  'max_market_funds': '400000',
  'post_only': False,
  'limit_only': False,
  'cancel_only': False},
 {'id': 'ETH-USD',
  'base_currency': 'ETH',
```

```
 'quote_currency': 'USD',
 'base_min_size': '0.01',
 'base_max_size': '700',
 'quote_increment': '0.01',
 'display_name': 'ETH/USD',
 'status': 'online',
 'margin_enabled': False,
 'status_message': None,
 'min_market_funds': '10',
 'max_market_funds': '1000000',
 'post_only': False,
 'limit_only': False,
 'cancel_only': False},
{'id': 'LTC-BTC',
 'base_currency': 'LTC',
 'quote_currency': 'BTC',
 'base_min_size': '0.1',
 'base_max_size': '2000',
 'quote_increment': '0.00001',
 'display_name': 'LTC/BTC',
 'status': 'online',
 'margin_enabled': False,
 'status_message': None,
 'min_market_funds': '0.001',
 'max_market_funds': '30',
 'post_only': False,
 'limit_only': False,
 'cancel_only': False},
{'id': 'LTC-EUR',
 'base_currency': 'LTC',
 'quote_currency': 'EUR',
 'base_min_size': '0.1',
 'base_max_size': '1000',
 'quote_increment': '0.01',
 'display_name': 'LTC/EUR',
 'status': 'online',
 'margin_enabled': False,
 'status_message': None,
```

```
  'min_market_funds': '10',
  'max_market_funds': '200000',
  'post_only': False,
  'limit_only': False,
  'cancel_only': False},
 {'id': 'LTC-USD',
  'base_currency': 'LTC',
  'quote_currency': 'USD',
  'base_min_size': '0.1',
  'base_max_size': '4000',
  'quote_increment': '0.01',
  'display_name': 'LTC/USD',
  'status': 'online',
  'margin_enabled': False,
  'status_message': None,
  'min_market_funds': '10',
  'max_market_funds': '1000000',
  'post_only': False,
  'limit_only': False,
  'cancel_only': False},
 {'id': 'BCH-EUR',
  'base_currency': 'BCH',
  'quote_currency': 'EUR',
  'base_min_size': '0.01',
  'base_max_size': '120',
  'quote_increment': '0.01',
  'display_name': 'BCH/EUR',
  'status': 'online',
  'margin_enabled': False,
  'status_message': None,
  'min_market_funds': '10',
  'max_market_funds': '200000',
  'post_only': False,
  'limit_only': False,
  'cancel_only': False}]
```

In [130]:
```
# use pandas to format then products list to be a bit prettier
import pandas as pd
pd.DataFrame(public_client.get_products())
```

Out[130]:

	base_currency	base_max_size	base_min_size	cancel_only	disp
0	BCH	200	0.01	False	BCH
1	BCH	350	0.01	False	BCH
2	BTC	50	0.001	False	BTC
3	BTC	20	0.001	False	BTC
4	BTC	70	0.001	False	BTC
5	ETH	600	0.01	False	ETH
6	ETH	400	0.01	False	ETH

	base_currency	base_max_size	base_min_size	cancel_only	disp
7	ETH	700	0.01	False	ETH
8	LTC	2000	0.1	False	LTC/
9	LTC	1000	0.1	False	LTC/
10	LTC	4000	0.1	False	LTC/
11	BCH	120	0.01	False	BCH

In [134]:
```
# get historic rates for 'BTC-USD', and give it to pandas
# (remember column order; TLHOCV)
pd.DataFrame(
    public_client.get_product_historic_rates('BTC-USD'),
    columns=['time','low','high','open','close','volume'])
```

Out[134]:

	time	low	high	open	close	volur
0	1524509700	8907.00	8907.00	8907.00	8907.00	0.28670
1	1524509640	8906.99	8907.00	8906.99	8907.00	0.50951
2	1524509580	8906.99	8907.00	8907.00	8907.00	0.19270
3	1524509520	8906.99	8907.00	8906.99	8906.99	1.94311
4	1524509460	8906.99	8907.00	8906.99	8907.00	3.65614
5	1524509400	8906.99	8907.00	8906.99	8907.00	2.25110
6	1524509340	8905.99	8907.00	8905.99	8906.99	6.48396
7	1524509280	8904.99	8905.00	8904.99	8905.00	6.43775
8	1524509220	8904.00	8905.00	8904.00	8904.99	12.7680
9	1524509160	8901.00	8904.00	8901.00	8904.00	4.72463
10	1524509100	8901.00	8901.01	8901.00	8901.01	4.01600
11	1524509040	8901.00	8901.01	8901.01	8901.01	1.75923

	time	low	high	open	close	volu
12	1524508980	8901.00	8901.01	8901.00	8901.00	1.37865
13	1524508920	8901.00	8901.01	8901.00	8901.00	1.02654
14	1524508860	8901.00	8901.01	8901.01	8901.00	5.31180
15	1524508800	8901.00	8901.01	8901.01	8901.00	1.55033
16	1524508740	8901.00	8901.01	8901.00	8901.00	2.35272
17	1524508680	8901.00	8909.00	8908.99	8901.00	77.2995
18	1524508620	8908.99	8909.00	8909.00	8908.99	4.70493
19	1524508560	8908.99	8909.00	8908.99	8908.99	3.01641
20	1524508500	8908.99	8909.00	8908.99	8909.00	0.40657
21	1524508440	8908.99	8909.00	8909.00	8908.99	3.70929
22	1524508380	8908.99	8909.00	8908.99	8908.99	2.49734
23	1524508320	8906.47	8909.00	8906.47	8908.99	3.57550
24	1524508260	8899.98	8906.45	8899.99	8906.45	15.6272
25	1524508200	8899.98	8899.99	8899.98	8899.99	14.5278
26	1524508140	8899.98	8899.99	8899.98	8899.99	7.25873
27	1524508080	8899.98	8899.99	8899.98	8899.98	2.92275
28	1524508020	8888.99	8899.99	8889.00	8899.98	35.9128

	time	low	high	open	close	volur
29	1524507960	8888.99	8889.00	8889.00	8888.99	2.41210
...	...	...	...	...	...	...
270	1524493500	8905.00	8905.01	8905.00	8905.01	4.56141
271	1524493440	8905.00	8908.15	8905.71	8905.00	5.25645
272	1524493380	8905.00	8905.66	8905.01	8905.66	1.81665
273	1524493320	8905.00	8911.69	8911.69	8905.00	11.7295
274	1524493260	8895.46	8911.70	8895.47	8911.69	18.9283
275	1524493200	8895.46	8895.47	8895.47	8895.46	2.19124
276	1524493140	8895.46	8895.47	8895.46	8895.46	1.84658
277	1524493080	8895.46	8895.47	8895.46	8895.47	15.1595
278	1524493020	8895.00	8903.00	8903.00	8895.34	19.1150
279	1524492960	8903.15	8915.00	8909.00	8903.15	4.35707
280	1524492900	8900.00	8908.00	8904.02	8908.00	29.6396
281	1524492840	8904.11	8921.00	8920.99	8904.11	42.5946
282	1524492780	8920.99	8921.00	8920.99	8920.99	1.67996
283	1524492720	8920.99	8921.00	8920.99	8920.99	3.59026
284	1524492660	8920.99	8925.99	8925.99	8921.00	11.0380

	time	low	high	open	close	volur
285	1524492600	8925.99	8930.01	8930.01	8926.00	3.65325
286	1524492540	8930.00	8930.09	8930.09	8930.01	1.59027
287	1524492480	8930.09	8930.10	8930.09	8930.09	1.76336
288	1524492420	8930.16	8931.40	8931.39	8930.16	4.53585
289	1524492360	8925.99	8931.40	8926.00	8931.40	8.16156
290	1524492300	8917.00	8926.00	8917.00	8925.99	2.84910
291	1524492240	8916.99	8917.00	8917.00	8917.00	5.36953
292	1524492180	8916.99	8917.00	8916.99	8917.00	10.4128
293	1524492120	8916.99	8917.00	8917.00	8917.00	7.62549
294	1524492060	8910.02	8917.01	8917.01	8917.00	7.58786
295	1524492000	8917.00	8920.11	8920.11	8917.00	15.1147
296	1524491940	8920.10	8920.11	8920.10	8920.10	1.58894
297	1524491880	8920.10	8936.20	8936.20	8920.11	6.45250
298	1524491820	8936.20	8936.21	8936.20	8936.21	4.50225
299	1524491760	8936.20	8936.21	8936.20	8936.21	4.39360

300 rows × 6 columns

In [137]:

```python
# convert 'time' to dt
df = pd.DataFrame(
    public_client.get_product_historic_rates('BTC-USD'),
    columns=['time','low','high','open','close','volume'])
df['time'] = pd.to_datetime(df['time'], unit='s')
df
```

Out[137]:

	time	low	high	open	close	volume
0	2018-04-23 18:56:00	8906.99	8907.00	8906.99	8907.00	0.637195
1	2018-04-23 18:55:00	8906.99	8907.00	8907.00	8906.99	1.507100
2	2018-04-23 18:54:00	8906.99	8907.00	8906.99	8907.00	0.509510
3	2018-04-23 18:53:00	8906.99	8907.00	8907.00	8907.00	0.192700

	time	low	high	open	close	volume
4	2018-04-23 18:52:00	8906.99	8907.00	8906.99	8906.99	1.943114
5	2018-04-23 18:51:00	8906.99	8907.00	8906.99	8907.00	3.656145
6	2018-04-23 18:50:00	8906.99	8907.00	8906.99	8907.00	2.251107
7	2018-04-23 18:49:00	8905.99	8907.00	8905.99	8906.99	6.483964
8	2018-04-23 18:48:00	8904.99	8905.00	8904.99	8905.00	6.437750
9	2018-04-23 18:47:00	8904.00	8905.00	8904.00	8904.99	12.768002

	time	low	high	open	close	volume
10	2018-04-23 18:46:00	8901.00	8904.00	8901.00	8904.00	4.724637
11	2018-04-23 18:45:00	8901.00	8901.01	8901.00	8901.01	4.016008
12	2018-04-23 18:44:00	8901.00	8901.01	8901.01	8901.01	1.759232
13	2018-04-23 18:43:00	8901.00	8901.01	8901.00	8901.00	1.378653
14	2018-04-23 18:42:00	8901.00	8901.01	8901.00	8901.00	1.026541
15	2018-04-23 18:41:00	8901.00	8901.01	8901.01	8901.00	5.311800

	time	low	high	open	close	volume
16	2018-04-23 18:40:00	8901.00	8901.01	8901.01	8901.00	1.550333
17	2018-04-23 18:39:00	8901.00	8901.01	8901.00	8901.00	2.352722
18	2018-04-23 18:38:00	8901.00	8909.00	8908.99	8901.00	77.299596
19	2018-04-23 18:37:00	8908.99	8909.00	8909.00	8908.99	76.701947
20	2018-04-23 18:36:00	8908.99	8909.00	8908.99	8908.99	3.016413
21	2018-04-23 18:35:00	8908.99	8909.00	8908.99	8908.99	0.496679

	time	low	high	open	close	volume
22	2018-04-23 18:34:00	8908.99	8909.00	8909.00	8908.99	3.709299
23	2018-04-23 18:33:00	8908.99	8909.00	8908.99	8908.99	2.497345
24	2018-04-23 18:32:00	8906.47	8909.00	8906.47	8908.99	4.148702
25	2018-04-23 18:31:00	8899.98	8906.48	8899.99	8906.48	16.156576
26	2018-04-23 18:30:00	8899.98	8899.99	8899.98	8899.99	14.527836
27	2018-04-23 18:29:00	8899.98	8899.99	8899.98	8899.99	7.258738

	time	low	high	open	close	volume
28	2018-04-23 18:28:00	8899.98	8899.99	8899.98	8899.98	2.922756
29	2018-04-23 18:27:00	8888.99	8899.99	8889.00	8899.99	37.816936
...	...	...	...	...	...	...
270	2018-04-23 14:26:00	8905.00	8905.01	8905.00	8905.00	8.249743
271	2018-04-23 14:25:00	8905.00	8905.01	8905.00	8905.01	4.561419
272	2018-04-23 14:24:00	8905.00	8908.15	8905.71	8905.00	5.256459
273	2018-04-23 14:23:00	8905.00	8905.66	8905.01	8905.66	1.816654

	time	low	high	open	close	volume
274	2018-04-23 14:22:00	8905.00	8911.69	8911.69	8905.00	11.729551
275	2018-04-23 14:21:00	8895.46	8911.70	8895.47	8911.69	18.928392
276	2018-04-23 14:20:00	8895.46	8895.47	8895.47	8895.47	2.908838
277	2018-04-23 14:19:00	8895.46	8895.47	8895.46	8895.46	1.846582
278	2018-04-23 14:18:00	8895.46	8895.47	8895.46	8895.47	15.159551
279	2018-04-23 14:17:00	8895.00	8903.00	8903.00	8895.34	19.115037

In [138]:

```
from time import sleep

def get_loads(symbol, start=None, end=None, granularity=86400):
    """ This was boring so I'm not live-coding this one"""
    if end is None:
        end = pd.to_datetime('now')
    if start is None:
        start = end-pd.Timedelta(seconds=granularity)

    while True:
        response = public_client.get_product_historic_rates(
                product_id=symbol,
                granularity=granularity,
                start=start.isoformat(),
                end=end.isoformat()
            )

        if not response:
            raise StopIteration()
        if not isinstance(response,list):
            raise ValueError(response)

        for r in response:
            yield r
        sleep(3)
        end = pd.to_datetime(r[0], unit='s')
        start = end-pd.Timedelta(seconds=granularity*len(response))
        print(f"{start}-{end}")
```

In [143]:
```
# use get_loads to build a dataframe, remember TLHOCV
df= pd.DataFrame(
    get_loads('BTC-USD'),
    columns=['time','low','high','open','close','volume'])
df['time'] = pd.to_datetime(df['time'], unit='s')
df.head()
```

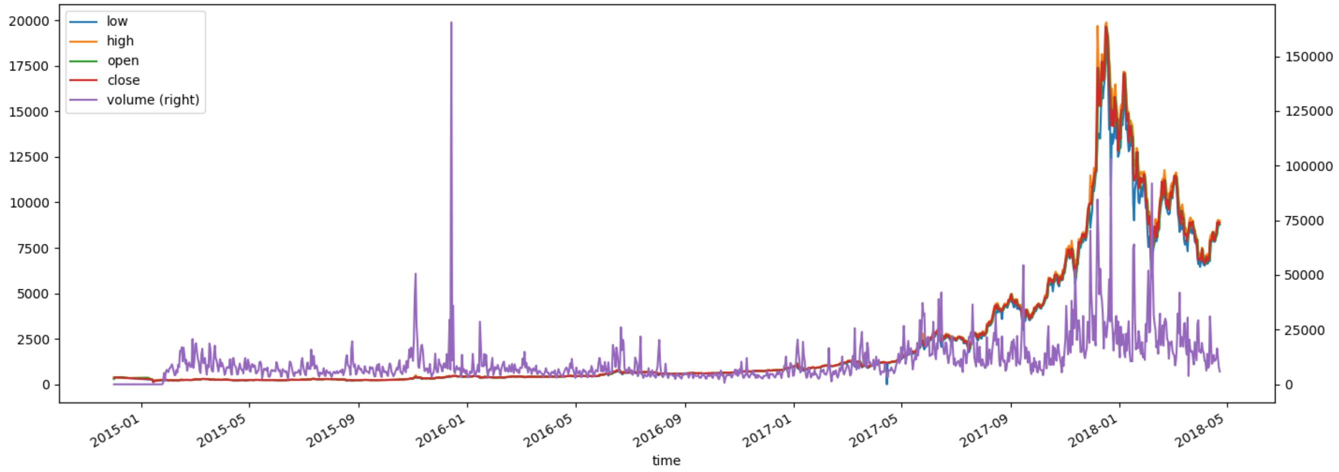
```
2016-09-01 00:00:00-2017-06-28 00:00:00
2015-11-06 00:00:00-2016-09-01 00:00:00
2015-01-10 00:00:00-2015-11-06 00:00:00
2014-03-23 00:00:00-2015-01-13 00:00:00
2014-11-21 00:00:00-2014-12-01 00:00:00
```

Out[143]:

	time	low	high	open	close	volume
0	2018-04-23	8775.10	8991.00	8795.00	8888.33	5842.771784
1	2018-04-22	8754.01	9015.00	8915.42	8795.01	7803.469852
2	2018-04-21	8610.70	9038.87	8866.27	8915.42	12270.503231
3	2018-04-20	8216.21	8932.57	8274.00	8866.27	16412.808992
4	2018-04-19	8101.47	8300.00	8152.05	8274.00	11932.907048

In [145]:

```
# Plot dataframe using matplotlib
df.set_index('time').plot(secondary_y='volume')
```



Out[145]: <matplotlib.axes._subplots.AxesSubplot at 0x1a2ee86dd8>

In [146]:

```
# This is just an alternative visualisation so I'm not rewriting this
import plotly.plotly as py
from plotly.offline import init_notebook_mode, plot, iplot
import plotly.graph_objs as go
from datetime import datetime
init_notebook_mode(connected=False)
import cufflinks

data = [go.Candlestick(
    x=df.time,
    open=df.open,
    close=df.close,
    high=df.high,
    low=df.low
)]
```

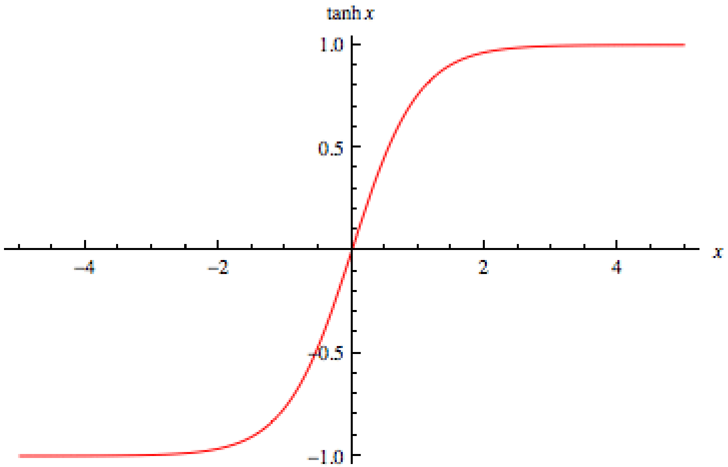
```
In [147]: py.iplot(data)
```

Out[147]:

Data Cleaning

LSTM operates best with values in the range -1,1 because of the `tanh` function in the middle

But lets pretend we don't know that and have a look at what other `sklearn` scales give



In [91]:

```
# use sklearn.preprocessing fuctions to try out a few scaling methods
# remember iplot and boxplot
```

In [95]:

```
# create a static mmscaler and show that you can fit/transform the close valuse
# *and* get the original back from the scaled using nverse_transform
```

In [96]:

```
from sklearn.preprocessing import MinMaxScaler
import numpy as np
scaler = MinMaxScaler(feature_range=(-1,1))
X = df.close.values
print(X.shape)
X = X.reshape(len(X),1)
print(X.shape)
scaler = scaler.fit(X)
scaled_X = scaler.transform(X)
inverted_X = scaler.inverse_transform(scaled_X)
np.allclose(X,inverted_X)
```

```
(1206,)
(1206, 1)
```

Out[96]:

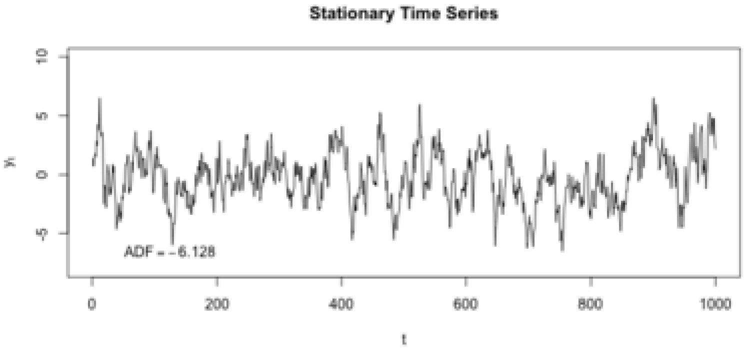
```
True
```

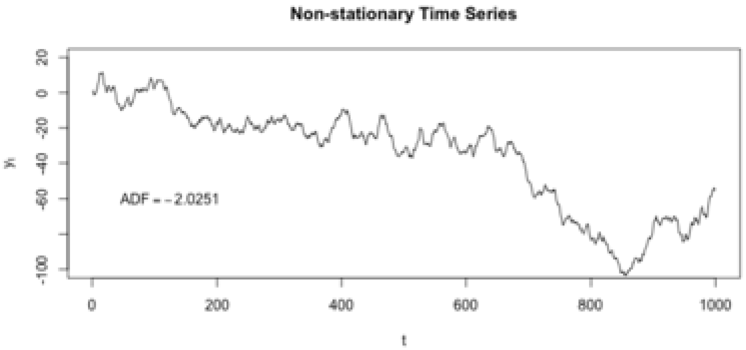
Hold your horses hotshot, it’s not that simple;

A few other things to bear in mind;

- Predictive modeling means it's 'Supervised' learning but we don't have any labels?
- We want to 'predict' the future value based on the past
- The 'past' of bitcoin as had significant shifts in scale (almost logish)

Sidebar: What is ’stationarity’?





The Fixes

- Transform the dataset so we use the 'future' as a label for the 'past'
- Make the data timeseries stationary to correct for relative scaling
- Rescale again

In [149]:
```
# Need to order the data ascending for supervisfication
X=df.set_index('time').close.sort_index()
X
```

Out[149]:
```
time
2014-12-01    370.00
2014-12-02    378.00
2014-12-03    378.00
2014-12-04    377.10
2014-12-06    378.00
2014-12-08    375.00
2014-12-10    360.50
2014-12-12    350.00
2014-12-18    340.00
2015-01-08    288.99
2015-01-13    260.00
2015-01-14    120.00
2015-01-15    204.22
2015-01-16    199.46
2015-01-17    184.00
2015-01-19    225.51
2015-01-20    218.00
2015-01-21    225.51
2015-01-22    226.32
2015-01-23    235.00
2015-01-24    240.00
2015-01-25    254.53
2015-01-26    274.48
2015-01-27    263.65
2015-01-28    236.09
2015-01-29    235.03
2015-01-30    229.07
2015-01-31    218.45
2015-02-01    228.99
2015-02-02    237.83
                ...
```

```
2018-03-25    8453.00
2018-03-26    8145.00
2018-03-27    7793.61
2018-03-28    7942.72
2018-03-29    7079.99
2018-03-30    6848.01
2018-03-31    6928.50
2018-04-01    6816.01
2018-04-02    7045.01
2018-04-03    7424.90
2018-04-04    6791.68
2018-04-05    6785.85
2018-04-06    6619.01
2018-04-07    6894.01
2018-04-08    7020.01
2018-04-09    6771.13
2018-04-10    6824.99
2018-04-11    6942.99
2018-04-12    7916.00
2018-04-13    7893.19
2018-04-14    8003.11
2018-04-15    8355.25
2018-04-16    8048.93
2018-04-17    7892.10
2018-04-18    8152.05
2018-04-19    8274.00
2018-04-20    8866.27
2018-04-21    8915.42
2018-04-22    8795.01
2018-04-23    8888.33
Name: close, Length: 1206, dtype: float64
```

In [150]:

```python
# Folks, we all know this live coding is a faff, lets get a move on.
def ts_df_to_supervised(df, lag=1):
    columns = [df.shift(i) for i in range(1, lag+1)]
    columns.append(df)
    _df = pd.concat(columns, axis=1)
    _df.dropna(inplace=True)
    _df.columns = ['today','tomorrow']
    return _df

sup_df = ts_df_to_supervised(X)
sup_df.head()
```

Out[150]:

	today	tomorrow
time		
2014-12-02	370.0	378.0
2014-12-03	378.0	378.0
2014-12-04	378.0	377.1
2014-12-06	377.1	378.0
2014-12-08	378.0	375.0

In [151]:

```python
## now make the values stationary
def difference(X, lag=1):
    return X.diff(lag).dropna()
difference(sup_df).head()
```

Out[151]:

	today	tomorrow
time		
2014-12-03	8.0	0.0
2014-12-04	0.0	-0.9
2014-12-06	-0.9	0.9
2014-12-08	0.9	-3.0
2014-12-10	-3.0	-14.5

In [152]:
```python
def inverse_difference(history, yhat, interval=1):
    return yhat + history[-interval]

differenced = difference(X)
undifferenced = pd.Series(
    [inverse_difference(X, differenced[i], len(X)-i)
     for i in range(len(differenced))],
    index = X.index[1:]
)
(undifferenced-X).sum()
```

Out[152]:
```
0.0
```

In [102]:
```
from sklearn.preprocessing import MinMaxScaler
display(X.head())
_X = X.values.reshape(len(X), 1)

scaler = MinMaxScaler(feature_range=(-1, 1))
scaler = scaler.fit(_X)
scaled_X = scaler.transform(_X)
scaled_series = pd.Series(scaled_X[:, 0], index=X.index)
display(scaled_series.head())
# invert transform to validate
inverted_X = scaler.inverse_transform(scaled_X)
inverted_series = pd.Series(inverted_X[:, 0], index=X.index)
display(inverted_series.head())
```

```
time
2014-12-01    370.0
2014-12-02    378.0
2014-12-03    378.0
2014-12-04    377.1
2014-12-06    378.0
Name: close, dtype: float64

time
2014-12-01   -0.974398
2014-12-02   -0.973579
2014-12-03   -0.973579
2014-12-04   -0.973671
2014-12-06   -0.973579
dtype: float64

time
2014-12-01    370.0
2014-12-02    378.0
2014-12-03    378.0
2014-12-04    377.1
```

```
2014-12-06    378.0
dtype: float64
```

FINALLY: We can start doing some actual ML

We have:

- Methods to scale/invert data to -1,1
- Methods to transform / invert data to stationary
- Methods to transform data to supervised with configurable 'lag'

Keras LSTM Cavaets

- LSTM must be 'stateful' (i.e. doesn't forget between batches) and must be manually cleared
- Expected input to be sample/timestep/feature 3-matrix (we only have one feature atm so that's easy)

In [103]:
```python
def twodim_to_threedim(X):
    return X.values.reshape(X.shape[0],1,X.shape[1])
twodim_to_threedim(difference(ts_df_to_supervised(X)))
```

Out[103]:
```
array([[[   8.  ,    0.  ]],

       [[   0.  ,   -0.9 ]],

       [[  -0.9 ,    0.9 ]],

       ...,

       [[ 592.27,   49.15]],

       [[  49.15, -120.41]],

       [[-120.41,  120.97]]])
```

Splitting Training / Test Datasets

Important to remove bias / overfitting

In [153]:

```
raw_X = df.set_index('time')['close'].sort_index() # Sort Closes
dX = difference(raw_X)                                 # Stationarise
sup_dX = ts_df_to_supervised(dX)                   # Supervise
i_split = 2*sup_dX.shape[0]//3
sup_dX_train = sup_dX.iloc[:i_split]          # Training Set
sup_dX_test = sup_dX.iloc[i_split:]           # Test Set
batch_size=2                                        # Use online learning
n_epoch = 600
n_neurons = 10
```

In [154]:

```
def scale(train, test):
    # fit scaler to both sets
    scaler = MinMaxScaler(feature_range=(-1, 1))
    scaler = scaler.fit(train.values)
    # transform train
    train = train.values.reshape(train.shape[0], train.shape[1])
    train_scaled = scaler.transform(train)
    # transform test
    test = test.values.reshape(test.shape[0], test.shape[1])
    test_scaled = scaler.transform(test)
    return scaler, train_scaled, test_scaled

def invert_scale(scaler, X, value):
    new_row = [x for x in X] + [value]
    array = np.array(new_row)
    array = array.reshape(1, len(array))
    inverted = scaler.inverse_transform(array)
    return inverted[0, -1]
```

In [155]:

```
scaler, train_scaled, test_scaled = scale(sup_dX_train, sup_dX_test)
```

FIRST BIT OF ACTUAL ML

Yes Jase, Tensorflow is in the background ☹️

- And there's no way in fuck I'm doing this bit live
- Single layer 10 stage LSTM with 200 epochs on diffed scaled single-memory input

In [156]:
```
from keras.layers.core import Dense, Activation, Dropout, Flatten
from keras.layers.recurrent import LSTM
from keras.models import Sequential, load_model
import time
from keras_tqdm import TQDMNotebookCallback
import tqdm
```

In [157]:

```
def fit_lstm(train, batch_size, nb_epoch, neurons):
    X, y = train[:, 0:-1], train[:, -1]
    X = X.reshape(X.shape[0], 1, X.shape[1])
    model = Sequential()
    model.add(LSTM(neurons, activation='sigmoid', batch_input_shape=(batch_size, X
.shape[1], X.shape[2]), stateful=True))
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer='adam')

    model.fit(X, y, epochs=200, batch_size=batch_size,
              verbose=0, shuffle=False,
              callbacks=[TQDMNotebookCallback(leave_inner=False,leave_outer=False
)]
             )
    model.reset_states()
    return model

def forecast(model, batch_size, X):
    X = X.reshape( len(X),1, 1,)
    yhat = model.predict(X, batch_size=batch_size)
    return yhat[0,0]
```

In [158]:
```
lstm = fit_lstm(train_scaled, batch_size, n_epoch, n_neurons)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-158-7b660b798042> in <module>()
----> 1 lstm = fit_lstm(train_scaled, batch_size, n_epoch, n_neurons)

<ipython-input-157-c9012426def1> in fit_lstm(train, batch_size, nb_epoch, neur
ons)
      9     model.fit(X, y, epochs=200, batch_size=batch_size,
     10               verbose=0, shuffle=False,
---> 11               callbacks=[TQDMNotebookCallback(leave_inner=False,leave_
outer=False)]
     12              )
     13     model.reset_states()

~/anaconda3/envs/bash/lib/python3.6/site-packages/keras/models.py in fit(self,
 x, y, batch_size, epochs, verbose, callbacks, validation_split, validation_da
ta, shuffle, class_weight, sample_weight, initial_epoch, steps_per_epoch, vali
dation_steps, **kwargs)
    961                               initial_epoch=initial_epoch,
    962                               steps_per_epoch=steps_per_epoch,
--> 963                               validation_steps=validation_steps)
    964
    965     def evaluate(self, x=None, y=None,

~/anaconda3/envs/bash/lib/python3.6/site-packages/keras/engine/training.py in
fit(self, x, y, batch_size, epochs, verbose, callbacks, validation_split, vali
dation_data, shuffle, class_weight, sample_weight, initial_epoch, steps_per_ep
```

```
och, validation_steps, **kwargs)
   1703                                         initial_epoch=initial_epoch,
   1704                                         steps_per_epoch=steps_per_epoch,
-> 1705                                         validation_steps=validation_steps)
   1706 
   1707     def evaluate(self, x=None, y=None,

~/anaconda3/envs/bash/lib/python3.6/site-packages/keras/engine/training.py in
_fit_loop(self, f, ins, out_labels, batch_size, epochs, verbose, callbacks, va
l_f, val_ins, shuffle, callback_metrics, initial_epoch, steps_per_epoch, valid
ation_steps)
   1233                             ins_batch[i] = ins_batch[i].toarray()
   1234 
-> 1235                         outs = f(ins_batch)
   1236                         if not isinstance(outs, list):
   1237                             outs = [outs]

~/anaconda3/envs/bash/lib/python3.6/site-packages/keras/backend/tensorflow_bac
kend.py in __call__(self, inputs)
   2476         session = get_session()
   2477         updated = session.run(fetches=fetches, feed_dict=feed_dict,
-> 2478                               **self.session_kwargs)
   2479         return updated[:len(self.outputs)]
   2480 

~/anaconda3/envs/bash/lib/python3.6/site-packages/tensorflow/python/client/ses
sion.py in run(self, fetches, feed_dict, options, run_metadata)
    776     try:
    777       result = self._run(None, fetches, feed_dict, options_ptr,
--> 778                          run_metadata_ptr)
    779       if run_metadata:
    780         proto_data = tf_session.TF_GetBuffer(run_metadata_ptr)

~/anaconda3/envs/bash/lib/python3.6/site-packages/tensorflow/python/client/ses
sion.py in _run(self, handle, fetches, feed_dict, options, run_metadata)
    980     if final_fetches or final_targets:
    981       results = self._do_run(handle, final_targets, final_fetches,
--> 982                              feed_dict_string, options, run_metadata)
```

```
    983     else:
    984       results = []

~/anaconda3/envs/bash/lib/python3.6/site-packages/tensorflow/python/client/ses
sion.py in _do_run(self, handle, target_list, fetch_list, feed_dict, options,
 run_metadata)
   1030     if handle is None:
   1031       return self._do_call(_run_fn, self._session, feed_dict, fetch_li
st,
-> 1032                            target_list, options, run_metadata)
   1033     else:
   1034       return self._do_call(_prun_fn, self._session, handle, feed_dict,

~/anaconda3/envs/bash/lib/python3.6/site-packages/tensorflow/python/client/ses
sion.py in _do_call(self, fn, *args)
   1037   def _do_call(self, fn, *args):
   1038     try:
-> 1039       return fn(*args)
   1040     except errors.OpError as e:
   1041       message = compat.as_text(e.message)

~/anaconda3/envs/bash/lib/python3.6/site-packages/tensorflow/python/client/ses
sion.py in _run_fn(session, feed_dict, fetch_list, target_list, options, run_m
etadata)
   1019         return tf_session.TF_Run(session, options,
   1020                                  feed_dict, fetch_list, target_list,
-> 1021                                  status, run_metadata)
   1022
   1023     def _prun_fn(session, handle, feed_dict, fetch_list):

KeyboardInterrupt:
```

In [159]:

```
## THIS TAKES A LONG TIME SO HERES ONE I MADE (a few) EARLIER
#lstm.save(f'lstm_{batch_size}_{n_epoch}_{n_neurons}.h5')
#lstm = load_model('lstm_batchwise_2.h5')
#lstm.save('lstm_batchwise_2.h5')
lstm = load_model(f'lstm_1_3000_4.h5')
lstm.reset_states()
```

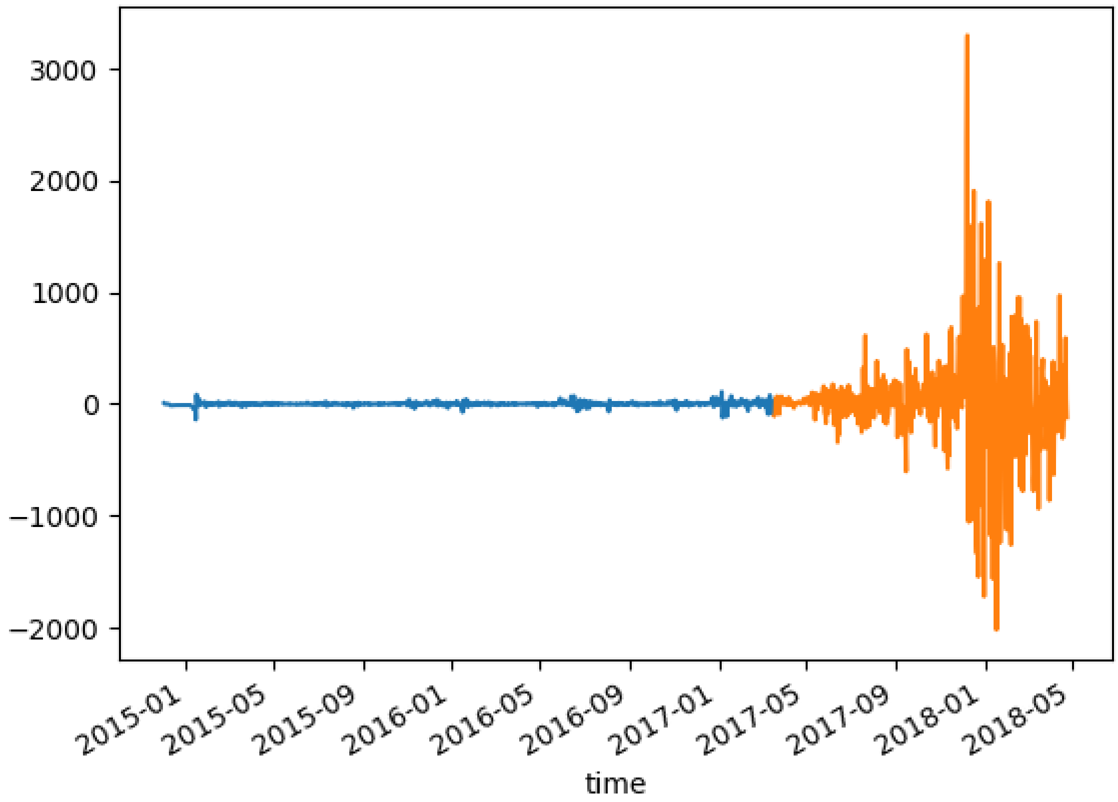
Model / Structure Validation

"Walk the line"

- Does the model map test data at all?
- Model has been trained on the first 2/3 of data
- Functionally; model is a 1-1 mapping of diff->next diff, not to value

But first, what's the data actually look like between training and test sets?

In [160]:
```
%matplotlib nbagg
sup_dX_train.iloc[0:,0].plot()
sup_dX_test.iloc[0:,0].plot()
```



Out[160]: <matplotlib.axes._subplots.AxesSubplot at 0x1a311e3cf8>

In [161]:

```
# walk-forward validation on the test data
## NOTE: THIS LOOKS AWESOME BUT IS JUST MAKING SURE
## IT WORKS; note the extra knowledge of `raw_x`
## in the diff inversion
predictions = []
yhats = []
train_reshaped = train_scaled[:, 0].reshape(len(train_scaled), 1, 1)
for i in range(len(test_scaled)):
    # make one-step forecast
    X, y = test_scaled[i, 0:-1], test_scaled[i, -1]
    yhat = forecast(lstm, 1, X)
    yhats.append(yhat)
    # invert scaling
    yhat = invert_scale(scaler, X, yhat)
    # invert differencing
    yhat = inverse_difference(raw_X, yhat, len(test_scaled)+1-i)
    # store forecast
    predictions.append(yhat)
    expected = raw_X[len(sup_dX_train) + i]
    print('Day=%d, Predicted=%f, Expected=%f' % (i+1, yhat, expected))
```

```
Day=1, Predicted=1145.414933, Expected=1175.110000
Day=2, Predicted=1072.023637, Expected=1069.570000
Day=3, Predicted=1031.286891, Expected=970.000000
Day=4, Predicted=1056.892778, Expected=1019.490000
Day=5, Predicted=1119.638865, Expected=1044.960000
Day=6, Predicted=989.077785, Expected=1114.420000
Day=7, Predicted=1049.893891, Expected=1034.570000
Day=8, Predicted=954.035492, Expected=1025.140000
Day=9, Predicted=980.269689, Expected=934.870000
Day=10, Predicted=991.228337, Expected=963.720000
Day=11, Predicted=1058.290810, Expected=973.080000
Day=12, Predicted=1063.636146, Expected=1042.080000
Day=13, Predicted=1060.670082, Expected=1045.400000
Day=14, Predicted=1059.919691, Expected=1043.270000
Day=15, Predicted=1105.561472, Expected=1042.340000
```

```
Day=16, Predicted=1109.670684, Expected=1088.990000
Day=17, Predicted=1130.716002, Expected=1092.000000
Day=18, Predicted=1169.323573, Expected=1113.990000
Day=19, Predicted=1161.514553, Expected=1152.600000
Day=20, Predicted=1150.233271, Expected=1143.990000
Day=21, Predicted=1208.372951, Expected=1132.990000
Day=22, Predicted=1211.410066, Expected=1192.300000
Day=23, Predicted=1201.614343, Expected=1194.000000
Day=24, Predicted=1227.407582, Expected=1184.500000
Day=25, Predicted=1227.067474, Expected=1210.970000
Day=26, Predicted=1240.433854, Expected=1210.000000
Day=27, Predicted=1231.244233, Expected=1223.990000
Day=28, Predicted=1211.653618, Expected=1214.170000
Day=29, Predicted=1181.072530, Expected=1177.050000
Day=30, Predicted=1188.588764, Expected=1173.740000
Day=31, Predicted=1193.364115, Expected=1178.850000
Day=32, Predicted=1204.840758, Expected=1177.990000
Day=33, Predicted=1216.988701, Expected=1189.910000
Day=34, Predicted=1229.255227, Expected=1201.940000
Day=35, Predicted=1250.922171, Expected=1214.210000
Day=36, Predicted=1264.899806, Expected=1236.150000
Day=37, Predicted=1262.427555, Expected=1249.990000
Day=38, Predicted=1267.351764, Expected=1247.000000
Day=39, Predicted=1272.781805, Expected=1251.980000
Day=40, Predicted=1296.194925, Expected=1257.290000
Day=41, Predicted=1313.539452, Expected=1281.160000
Day=42, Predicted=1364.002792, Expected=1298.440000
Day=43, Predicted=1368.816864, Expected=1349.260000
Day=44, Predicted=1380.650969, Expected=1353.340000
Day=45, Predicted=1399.609198, Expected=1365.430000
Day=46, Predicted=1451.254539, Expected=1384.550000
Day=47, Predicted=1486.845221, Expected=1436.500000
Day=48, Predicted=1547.583521, Expected=1471.990000
Day=49, Predicted=1578.149315, Expected=1533.000000
Day=50, Predicted=1567.035634, Expected=1563.390000
Day=51, Predicted=1600.290507, Expected=1551.300000
Day=52, Predicted=1624.518539, Expected=1585.390000
Day=53, Predicted=1718.747964, Expected=1609.570000
```

```
Day=54, Predicted=1737.670068, Expected=1713.000000
Day=55, Predicted=1810.014106, Expected=1720.430000
Day=56, Predicted=1853.992253, Expected=1794.990000
Day=57, Predicted=1595.546283, Expected=1837.930000
Day=58, Predicted=1786.918341, Expected=1695.610000
Day=59, Predicted=1813.310876, Expected=1792.730000
Day=60, Predicted=1755.946364, Expected=1799.990000
Day=61, Predicted=1794.039907, Expected=1747.810000
Day=62, Predicted=1831.839039, Expected=1777.480000
Day=63, Predicted=1915.500492, Expected=1813.230000
Day=64, Predicted=1993.919279, Expected=1899.160000
Day=65, Predicted=2075.788826, Expected=1976.230000
Day=66, Predicted=2076.341009, Expected=2058.910000
Day=67, Predicted=2139.726507, Expected=2057.000000
Day=68, Predicted=2272.928173, Expected=2123.290000
Day=69, Predicted=2425.039988, Expected=2272.750000
Day=70, Predicted=2365.160100, Expected=2432.970000
Day=71, Predicted=2203.118203, Expected=2355.000000
Day=72, Predicted=2019.899789, Expected=2272.700000
Day=73, Predicted=2263.448578, Expected=2099.990000
Day=74, Predicted=2293.129505, Expected=2232.780000
Day=75, Predicted=2120.128204, Expected=2279.480000
Day=76, Predicted=2367.707131, Expected=2191.580000
Day=77, Predicted=2431.786891, Expected=2303.290000
Day=78, Predicted=2498.155168, Expected=2419.990000
Day=79, Predicted=2563.868803, Expected=2478.990000
Day=80, Predicted=2540.501432, Expected=2548.050000
Day=81, Predicted=2695.833498, Expected=2521.360000
Day=82, Predicted=2856.136998, Expected=2698.000000
Day=83, Predicted=2590.985427, Expected=2871.290000
Day=84, Predicted=2859.705242, Expected=2685.640000
Day=85, Predicted=2823.186891, Expected=2799.730000
Day=86, Predicted=2939.324016, Expected=2811.390000
Day=87, Predicted=3016.082284, Expected=2931.150000
Day=88, Predicted=2559.177148, Expected=2998.980000
Day=89, Predicted=2763.267491, Expected=2655.710000
Day=90, Predicted=2354.911801, Expected=2709.010000
Day=91, Predicted=2455.291338, Expected=2432.210000
```

```
Day=92, Predicted=2492.226891, Expected=2409.980000
Day=93, Predicted=2632.023345, Expected=2480.430000
Day=94, Predicted=2460.842393, Expected=2634.940000
Day=95, Predicted=2637.056518, Expected=2515.250000
Day=96, Predicted=2736.876891, Expected=2596.980000
Day=97, Predicted=2570.951052, Expected=2725.080000
Day=98, Predicted=2726.566866, Expected=2643.350000
Day=99, Predicted=2702.556891, Expected=2679.990000
Day=100, Predicted=2478.573804, Expected=2690.760000
Day=101, Predicted=2528.775449, Expected=2574.840000
Day=102, Predicted=2404.966025, Expected=2505.610000
Day=103, Predicted=2578.015081, Expected=2407.910000
Day=104, Predicted=2569.365020, Expected=2575.750000
Day=105, Predicted=2551.928374, Expected=2553.120000
Day=106, Predicted=2479.809034, Expected=2530.000000
Day=107, Predicted=2457.090947, Expected=2455.190000
Day=108, Predicted=2528.932857, Expected=2423.630000
Day=109, Predicted=2563.040619, Expected=2516.660000
Day=110, Predicted=2617.827040, Expected=2542.410000
Day=111, Predicted=2636.030239, Expected=2602.000000
Day=112, Predicted=2622.916735, Expected=2616.960000
Day=113, Predicted=2453.608491, Expected=2604.840000
Day=114, Predicted=2585.952860, Expected=2501.150000
Day=115, Predicted=2543.592592, Expected=2561.110000
Day=116, Predicted=2261.269493, Expected=2508.990000
Day=117, Predicted=2357.241762, Expected=2331.050000
Day=118, Predicted=2395.216891, Expected=2310.010000
Day=119, Predicted=2341.436656, Expected=2383.420000
Day=120, Predicted=2176.684982, Expected=2340.000000
Day=121, Predicted=1888.404456, Expected=2217.240000
Day=122, Predicted=1939.006977, Expected=1964.310000
Day=123, Predicted=2256.571939, Expected=1911.780000
Day=124, Predicted=2325.476601, Expected=2235.190000
Day=125, Predicted=2278.478424, Expected=2308.150000
Day=126, Predicted=2829.428473, Expected=2258.990000
Day=127, Predicted=2610.881451, Expected=2873.480000
Day=128, Predicted=2826.786765, Expected=2657.450000
Day=129, Predicted=2765.798945, Expected=2825.270000
```

```
Day=130, Predicted=2774.056891, Expected=2754.280000
Day=131, Predicted=2505.784398, Expected=2762.260000
Day=132, Predicted=2566.596497, Expected=2564.000000
Day=133, Predicted=2676.786891, Expected=2525.990000
Day=134, Predicted=2790.835471, Expected=2664.990000
Day=135, Predicted=2693.486170, Expected=2786.070000
Day=136, Predicted=2727.705015, Expected=2700.210000
Day=137, Predicted=2845.938246, Expected=2723.990000
Day=138, Predicted=2635.269790, Expected=2856.880000
Day=139, Predicted=2726.499387, Expected=2732.590000
Day=140, Predicted=2798.816891, Expected=2699.900000
Day=141, Predicted=2863.936665, Expected=2787.020000
Day=142, Predicted=3195.168223, Expected=2857.340000
Day=143, Predicted=3236.741289, Expected=3243.490000
Day=144, Predicted=3398.839768, Expected=3222.220000
Day=145, Predicted=3442.449519, Expected=3398.230000
Day=146, Predicted=3332.806482, Expected=3422.430000
Day=147, Predicted=3435.991960, Expected=3342.800000
Day=148, Predicted=3633.446213, Expected=3444.980000
Day=149, Predicted=3840.699465, Expected=3656.150000
Day=150, Predicted=4033.970635, Expected=3874.000000
Day=151, Predicted=4280.833422, Expected=4060.470000
Day=152, Predicted=4086.554909, Expected=4320.600000
Day=153, Predicted=4349.237983, Expected=4159.930000
Day=154, Predicted=4286.676262, Expected=4370.010000
Day=155, Predicted=3999.198426, Expected=4280.010000
Day=156, Predicted=4222.632917, Expected=4101.720000
Day=157, Predicted=4056.586251, Expected=4157.410000
Day=158, Predicted=4039.043048, Expected=4050.990000
Day=159, Predicted=4103.796891, Expected=4002.000000
Day=160, Predicted=4160.625089, Expected=4092.000000
Day=161, Predicted=4311.340314, Expected=4143.490000
Day=162, Predicted=4376.026316, Expected=4312.030000
Day=163, Predicted=4362.329991, Expected=4360.000000
Day=164, Predicted=4358.288040, Expected=4344.320000
Day=165, Predicted=4401.992021, Expected=4340.110000
Day=166, Predicted=4584.039420, Expected=4384.990000
Day=167, Predicted=4605.092588, Expected=4599.000000
```

```
Day=168, Predicted=4750.599442, Expected=4581.980000
Day=169, Predicted=4927.125702, Expected=4743.940000
Day=170, Predicted=4558.742035, Expected=4947.990000
Day=171, Predicted=4645.380709, Expected=4649.990000
Day=172, Predicted=4412.941336, Expected=4626.050000
Day=173, Predicted=4426.486492, Expected=4498.250000
Day=174, Predicted=4638.479355, Expected=4432.510000
Day=175, Predicted=4639.609913, Expected=4616.180000
Day=176, Predicted=4265.616735, Expected=4624.180000
Day=177, Predicted=4364.642342, Expected=4350.000000
Day=178, Predicted=4275.973012, Expected=4334.360000
Day=179, Predicted=4246.466743, Expected=4251.360000
Day=180, Predicted=4197.167053, Expected=4210.720000
Day=181, Predicted=3789.932549, Expected=4164.520000
Day=182, Predicted=3203.840192, Expected=3855.320000
Day=183, Predicted=3745.369716, Expected=3250.400000
Day=184, Predicted=3749.248821, Expected=3740.020000
Day=185, Predicted=3736.599597, Expected=3726.510000
Day=186, Predicted=4049.593571, Expected=3719.970000
Day=187, Predicted=3835.692642, Expected=4100.000000
Day=188, Predicted=3888.713087, Expected=3910.110000
Day=189, Predicted=3520.276753, Expected=3872.060000
Day=190, Predicted=3673.776126, Expected=3617.470000
Day=191, Predicted=3799.126891, Expected=3619.010000
Day=192, Predicted=3571.983451, Expected=3787.330000
Day=193, Predicted=3997.124159, Expected=3669.010000
Day=194, Predicted=3896.886137, Expected=3919.780000
Day=195, Predicted=4188.312416, Expected=3885.090000
Day=196, Predicted=4208.062216, Expected=4200.000000
Day=197, Predicted=4186.598951, Expected=4189.420000
Day=198, Predicted=4350.796891, Expected=4156.990000
Day=199, Predicted=4408.014815, Expected=4339.000000
Day=200, Predicted=4409.935427, Expected=4394.810000
Day=201, Predicted=4292.506990, Expected=4392.710000
Day=202, Predicted=4252.870909, Expected=4307.990000
Day=203, Predicted=4331.836891, Expected=4214.840000
Day=204, Predicted=4382.584289, Expected=4320.040000
Day=205, Predicted=4440.915141, Expected=4362.990000
```

```
Day=206, Predicted=4595.883277, Expected=4425.000000
Day=207, Predicted=4757.122020, Expected=4603.490000
Day=208, Predicted=4771.613566, Expected=4769.550000
Day=209, Predicted=4830.354995, Expected=4750.000000
Day=210, Predicted=5388.987672, Expected=4814.990000
Day=211, Predicted=5591.395561, Expected=5440.000000
Day=212, Predicted=5799.660009, Expected=5624.800000
Day=213, Predicted=5648.409014, Expected=5819.130000
Day=214, Predicted=5763.280653, Expected=5693.700000
Day=215, Predicted=5550.222736, Expected=5754.900000
Day=216, Predicted=5614.143804, Expected=5594.000000
Day=217, Predicted=5715.806891, Expected=5574.440000
Day=218, Predicted=5961.804688, Expected=5704.010000
Day=219, Predicted=6038.622159, Expected=5989.100000
Day=220, Predicted=6022.755022, Expected=6024.860000
Day=221, Predicted=5857.893805, Expected=6005.050000
Day=222, Predicted=5456.416511, Expected=5905.990000
Day=223, Predicted=5759.565018, Expected=5525.430000
Day=224, Predicted=5899.309142, Expected=5739.970000
Day=225, Predicted=5689.118419, Expected=5891.610000
Day=226, Predicted=5778.064712, Expected=5780.000000
Day=227, Predicted=6172.850303, Expected=5752.010000
Day=228, Predicted=6128.297918, Expected=6140.010000
Day=229, Predicted=6438.638662, Expected=6124.160000
Day=230, Predicted=6742.044101, Expected=6445.010000
Day=231, Predicted=6995.437835, Expected=6783.690000
Day=232, Predicted=7157.806920, Expected=7039.980000
Day=233, Predicted=7385.529732, Expected=7170.010000
Day=234, Predicted=7416.171340, Expected=7412.550000
Day=235, Predicted=6886.349040, Expected=7392.000000
Day=236, Predicted=7166.637974, Expected=6969.760000
Day=237, Predicted=7488.406568, Expected=7126.630000
Day=238, Predicted=7063.754815, Expected=7467.960000
Day=239, Predicted=6503.642208, Expected=7156.000000
Day=240, Predicted=6299.191724, Expected=6577.620000
Day=241, Predicted=5868.508248, Expected=6346.700000
Day=242, Predicted=6525.301948, Expected=5886.350000
Day=243, Predicted=6612.486631, Expected=6535.870000
```

```
Day=244, Predicted=7244.304108, Expected=6605.000000
Day=245, Predicted=7788.321910, Expected=7294.000000
Day=246, Predicted=7714.027408, Expected=7838.530000
Day=247, Predicted=7780.953733, Expected=7714.710000
Day=248, Predicted=7993.395186, Expected=7777.010000
Day=249, Predicted=8210.236894, Expected=8031.820000
Day=250, Predicted=8056.334067, Expected=8256.010000
Day=251, Predicted=8242.313298, Expected=8109.000000
Day=252, Predicted=7934.900782, Expected=8250.000000
Day=253, Predicted=8290.886670, Expected=8031.160000
Day=254, Predicted=8822.010656, Expected=8215.010000
Day=255, Predicted=9372.592563, Expected=8795.500000
Day=256, Predicted=9724.990441, Expected=9401.110000
Day=257, Predicted=9923.373194, Expected=9768.710000
Day=258, Predicted=9957.448268, Expected=9949.000000
Day=259, Predicted=9933.106929, Expected=9935.980000
Day=260, Predicted=10818.847081, Expected=9903.000000
Day=261, Predicted=10927.262554, Expected=10869.840000
Day=262, Predicted=11246.051091, Expected=10930.240000
Day=263, Predicted=11598.085138, Expected=11290.000000
Day=264, Predicted=11719.223965, Expected=11643.980000
Day=265, Predicted=14038.987054, Expected=11718.350000
Day=266, Predicted=17338.997054, Expected=14090.000000
Day=267, Predicted=16314.188955, Expected=17390.010000
Day=268, Predicted=15255.082826, Expected=16367.030000
Day=269, Predicted=15312.118895, Expected=15309.980000
Day=270, Predicted=16891.994649, Expected=15290.010000
Day=271, Predicted=17691.892885, Expected=16885.760000
Day=272, Predicted=16636.431690, Expected=17730.120000
Day=273, Predicted=16770.524764, Expected=16689.610000
Day=274, Predicted=17707.775912, Expected=16749.780000
Day=275, Predicted=19602.157518, Expected=17738.670000
Day=276, Predicted=19325.434215, Expected=19650.010000
Day=277, Predicted=19046.705693, Expected=19378.990000
Day=278, Predicted=17856.221246, Expected=19039.010000
Day=279, Predicted=16513.346702, Expected=17838.730000
Day=280, Predicted=15764.101731, Expected=16496.890000
Day=281, Predicted=14232.556180, Expected=15758.800000
```

```
Day=282, Predicted=15039.851282, Expected=14210.570000
Day=283, Predicted=14165.077511, Expected=15075.890000
Day=284, Predicted=14183.234196, Expected=14221.940000
Day=285, Predicted=15800.962907, Expected=14171.980000
Day=286, Predicted=15299.562032, Expected=15790.880000
Day=287, Predicted=14398.042686, Expected=15367.080000
Day=288, Predicted=14583.640555, Expected=14450.010000
Day=289, Predicted=12795.872324, Expected=14565.050000
Day=290, Predicted=13827.091282, Expected=12839.990000
Day=291, Predicted=13418.188640, Expected=13863.130000
Day=292, Predicted=14745.471282, Expected=13480.010000
Day=293, Predicted=15054.549540, Expected=14781.510000
Day=294, Predicted=15153.311992, Expected=15098.140000
Day=295, Predicted=16909.052627, Expected=15144.990000
Day=296, Predicted=17076.809976, Expected=16960.010000
Day=297, Predicted=16092.833432, Expected=17098.990000
Day=298, Predicted=14921.175359, Expected=16174.220000
Day=299, Predicted=14408.547851, Expected=14993.740000
Day=300, Predicted=14875.873046, Expected=14480.990000
Day=301, Predicted=13241.258409, Expected=14875.180000
Day=302, Predicted=13783.961505, Expected=13308.060000
Day=303, Predicted=14142.411403, Expected=13820.000000
Day=304, Predicted=13578.834105, Expected=14187.950000
Day=305, Predicted=13605.902042, Expected=13656.230000
Day=306, Predicted=11494.665436, Expected=13590.000000
Day=307, Predicted=11120.305567, Expected=11570.010000
Day=308, Predicted=11348.016272, Expected=11200.010000
Day=309, Predicted=11510.786891, Expected=11305.530000
Day=310, Predicted=12759.496514, Expected=11498.990000
Day=311, Predicted=11467.059432, Expected=12762.800000
Day=312, Predicted=10705.295634, Expected=11518.170000
Day=313, Predicted=10851.979664, Expected=10766.700000
Day=314, Predicted=11355.018088, Expected=10824.940000
Day=315, Predicted=11067.072148, Expected=11356.790000
Day=316, Predicted=11106.705883, Expected=11118.000000
Day=317, Predicted=11341.956063, Expected=11086.890000
Day=318, Predicted=11532.006562, Expected=11319.000000
Day=319, Predicted=11031.981257, Expected=11536.000000
```

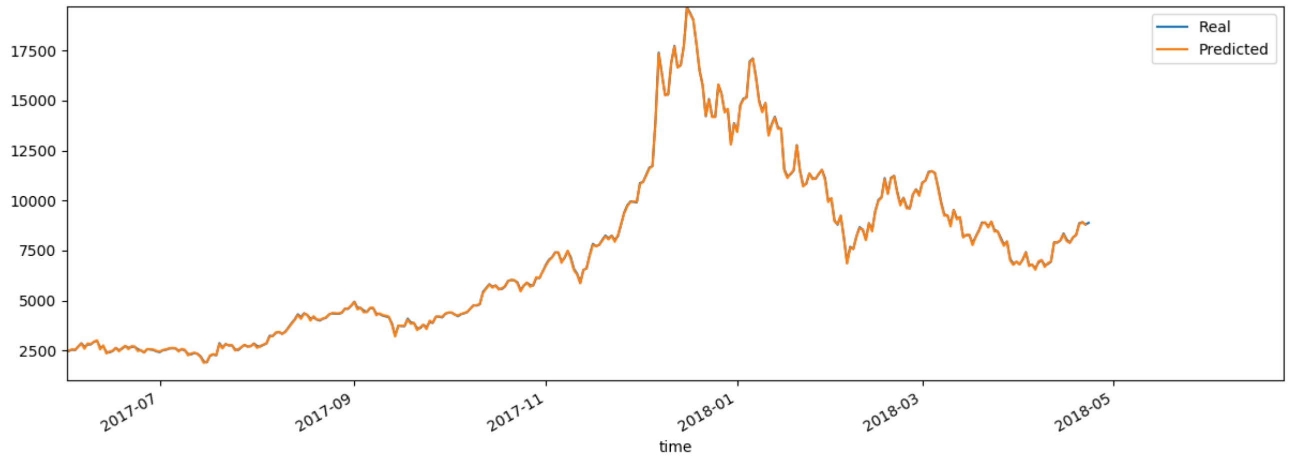
```
Day=320, Predicted=9920.584160, Expected=11123.010000
Day=321, Predicted=10122.109961, Expected=9995.000000
Day=322, Predicted=8958.746961, Expected=10099.990000
Day=323, Predicted=8827.811328, Expected=9014.230000
Day=324, Predicted=9243.252519, Expected=8787.520000
Day=325, Predicted=8113.153701, Expected=9240.000000
Day=326, Predicted=6854.156246, Expected=8167.910000
Day=327, Predicted=7652.421282, Expected=6905.190000
Day=328, Predicted=7571.887943, Expected=7688.460000
Day=329, Predicted=8180.645152, Expected=7575.750000
Day=330, Predicted=8624.127130, Expected=8218.100000
Day=331, Predicted=8548.441608, Expected=8671.010000
Day=332, Predicted=8016.978143, Expected=8547.490000
Day=333, Predicted=8858.130988, Expected=8072.990000
Day=334, Predicted=8455.376578, Expected=8872.280000
Day=335, Predicted=9437.930129, Expected=8520.010000
Day=336, Predicted=9982.644097, Expected=9472.980000
Day=337, Predicted=10146.591131, Expected=10031.230000
Day=338, Predicted=11070.532897, Expected=10167.490000
Day=339, Predicted=10321.810828, Expected=11121.500000
Day=340, Predicted=11103.961282, Expected=10380.040000
Day=341, Predicted=11228.784842, Expected=11140.000000
Day=342, Predicted=10369.544627, Expected=11235.570000
Day=343, Predicted=9754.547711, Expected=10454.270000
Day=344, Predicted=10127.351351, Expected=9830.000000
Day=345, Predicted=9601.311517, Expected=10144.990000
Day=346, Predicted=9617.787028, Expected=9688.620000
Day=347, Predicted=10311.780302, Expected=9597.990000
Day=348, Predicted=10543.506886, Expected=10300.000000
Day=349, Predicted=10232.733021, Expected=10566.570000
Day=350, Predicted=10871.681571, Expected=10307.270000
Day=351, Predicted=10999.922046, Expected=10895.920000
Day=352, Predicted=11382.009445, Expected=11000.000000
Day=353, Predicted=11472.734797, Expected=11432.500000
Day=354, Predicted=11383.543619, Expected=11469.900000
Day=355, Predicted=10622.088313, Expected=11377.540000
Day=356, Predicted=9849.706684, Expected=10700.000000
Day=357, Predicted=9237.079436, Expected=9920.000000
```

```
Day=358, Predicted=9270.235960, Expected=9304.900000
Day=359, Predicted=8703.579619, Expected=9255.000000
Day=360, Predicted=9524.324304, Expected=8795.440000
Day=361, Predicted=9053.399642, Expected=9533.880000
Day=362, Predicted=9178.474940, Expected=9120.000000
Day=363, Predicted=8149.625942, Expected=9145.410000
Day=364, Predicted=8288.812356, Expected=8207.030000
Day=365, Predicted=8289.461508, Expected=8259.990000
Day=366, Predicted=7770.995014, Expected=8275.000000
Day=367, Predicted=8174.956255, Expected=7865.020000
Day=368, Predicted=8461.118322, Expected=8192.000000
Day=369, Predicted=8852.178199, Expected=8497.800000
Day=370, Predicted=8909.157852, Expected=8900.000000
Day=371, Predicted=8653.555479, Expected=8891.810000
Day=372, Predicted=8954.141580, Expected=8715.090000
Day=373, Predicted=8446.590039, Expected=8927.100000
Day=374, Predicted=8474.241704, Expected=8531.340000
Day=375, Predicted=8047.228856, Expected=8453.000000
Day=376, Predicted=7733.378346, Expected=8145.000000
Day=377, Predicted=7961.712574, Expected=7793.610000
Day=378, Predicted=6995.940436, Expected=7942.720000
Day=379, Predicted=6781.322123, Expected=7079.990000
Day=380, Predicted=6957.171936, Expected=6848.010000
Day=381, Predicted=6804.514098, Expected=6928.500000
Day=382, Predicted=7060.510191, Expected=6816.010000
Day=383, Predicted=7405.089404, Expected=7045.010000
Day=384, Predicted=6718.649055, Expected=7424.900000
Day=385, Predicted=6814.231034, Expected=6791.680000
Day=386, Predicted=6536.876798, Expected=6785.850000
Day=387, Predicted=6961.234960, Expected=6619.010000
Day=388, Predicted=7031.806891, Expected=6894.010000
Day=389, Predicted=6680.952446, Expected=7020.010000
Day=390, Predicted=6877.338561, Expected=6771.130000
Day=391, Predicted=6954.786891, Expected=6824.990000
Day=392, Predicted=7899.233303, Expected=6942.990000
Day=393, Predicted=7917.595351, Expected=7916.000000
Day=394, Predicted=8015.971720, Expected=7893.190000
Day=395, Predicted=8305.786542, Expected=8003.110000
```

```
Day=396, Predicted=7971.297661, Expected=8355.250000
Day=397, Predicted=7890.112991, Expected=8048.930000
Day=398, Predicted=8163.991735, Expected=7892.100000
Day=399, Predicted=8279.162879, Expected=8152.050000
Day=400, Predicted=8817.661079, Expected=8274.000000
Day=401, Predicted=8916.104731, Expected=8866.270000
Day=402, Predicted=8793.735333, Expected=8915.420000
```

In [162]:

```
%matplotlib nbagg
import matplotlib.pyplot as plt
f,ax = plt.subplots()
pred = pd.Series(predictions, index=pd.date_range(start=sup_dX_train.index[-1], pe
riods=len(predictions)), name='Prediction')
X = df.set_index('time')['close'].sort_index()
X.plot(label='Real', ax=ax)
pred.plot(label='Predicted', ax=ax)
ax.legend()
```



Out[162]:

```
<matplotlib.legend.Legend at 0x1a2d497198>
```

In [163]:

```
# use mse to work out if you're totally fucked
from sklearn.metrics import mean_absolute_error
mean_absolute_error(X[pred.index].values, pred.values)
```

Out[163]:

```
32.28381575116524
```

In [164]:
```
from sklearn.metrics import mean_squared_error
validation = X[pred.index].values,pred.values
mean_squared_error(*validation)
```

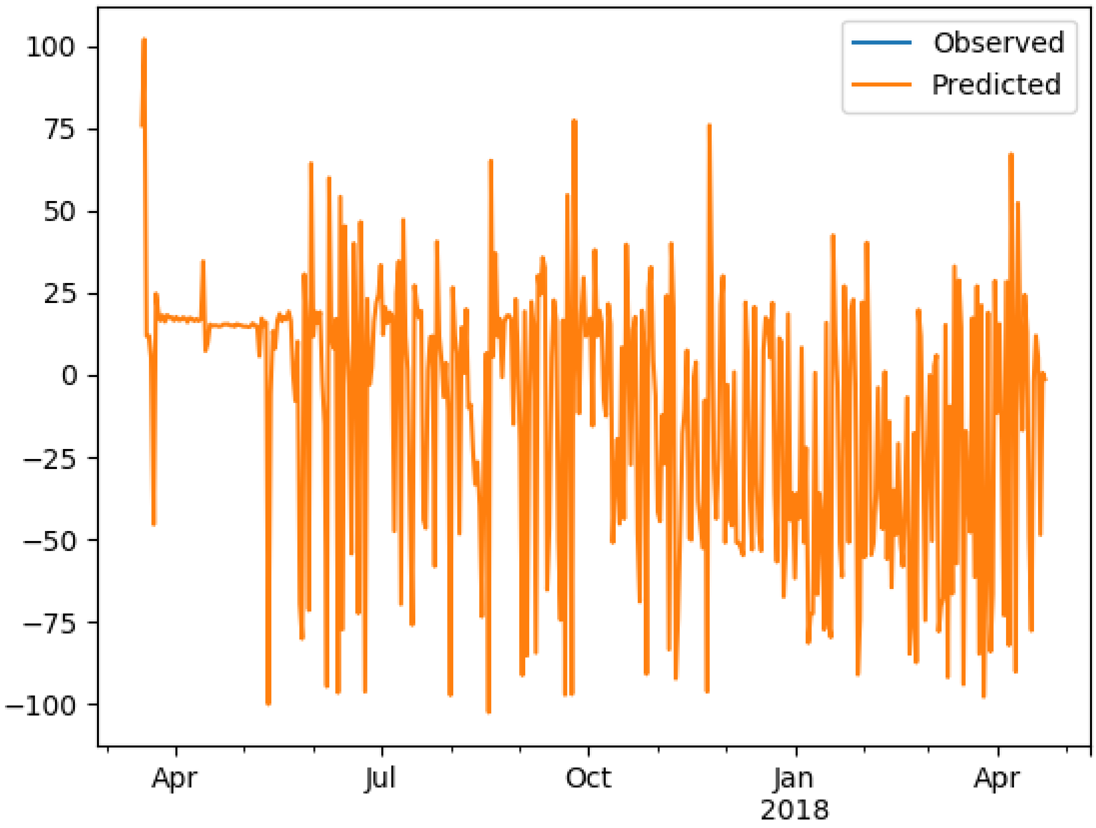
Out[164]: 1706.048181115777

In [115]:

```
# create a new dataframe with the validation values to see how wrong you are
```

In [165]:

```
%matplotlib nbagg
_validation = X[pred.index].copy().to_frame('Observed')
_validation['Predicted'] = pred
_validation.diff(axis=1).plot()
```



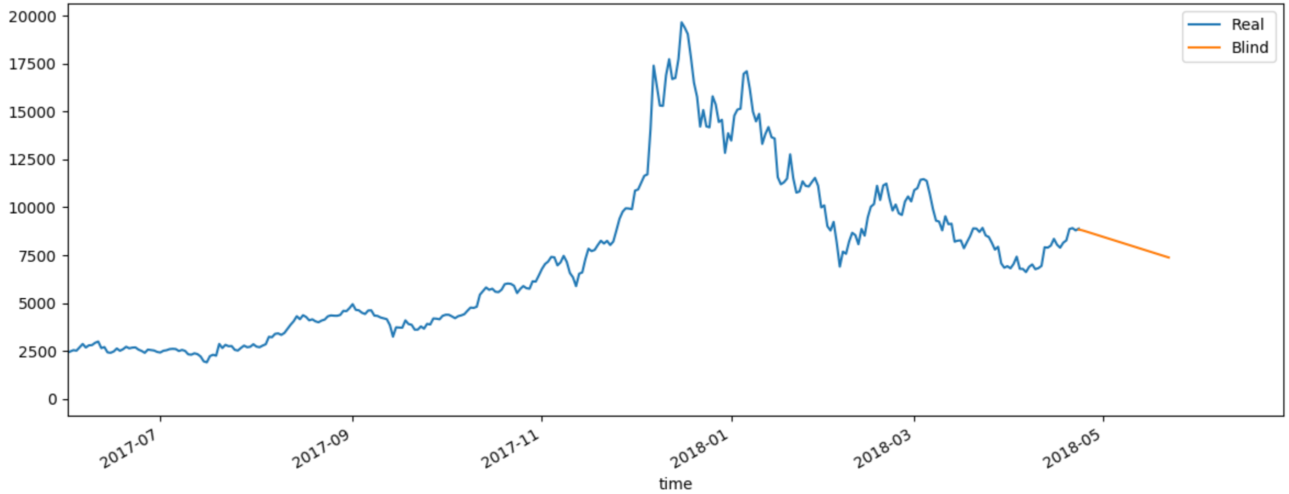
Out[165]: <matplotlib.axes._subplots.AxesSubplot at 0x1a30ffc860>

In [166]:

```
# Future predictions are fucked anyway so you're on your own.
X, y = test_scaled[-1, 0:-1], test_scaled[-1, -1]
yhats = []
new_history = raw_X.copy()
for day in tqdm.tqdm_notebook(range(30)):
    yhat = forecast(lstm, 1, X)
    # invert scaling
    yhat = invert_scale(scaler, X, yhat)
    # invert differencing
    yhat = new_history[-1]+yhat
    X=np.asarray([yhat])
    yhats.append(yhat)
    new_history = np.append(new_history,X)
```

In [167]:

```
%matplotlib nbagg
import matplotlib.pyplot as plt
f,ax = plt.subplots()
pred = pd.Series(yhats, index=pd.date_range(start=sup_dX.index[-1], periods=len(yh
ats)), name='Blind Prediction')
X = df.set_index('time')['close'].sort_index()
X.plot(label='Real', ax=ax)
pred.plot(label='Blind', ax=ax)
ax.legend()
ax.set_xlim([pd.to_datetime('2017-06-01'),pd.to_datetime('2018-06-28')])
```



Out[167]: (736481.0, 736873.0)

What I'd do differently / next?

- Multi-timestep pipeline and hyperparameter optimisation
- i.e. does stationarity correction make a difference?
- Re-training on dynamic timesteps (2/3/4-delay hops)
- Sliding window (needs dive in to TF underbelly)
- Treat multi-currency valuations as correlated features
- Try Attention Modelling (https://towardsdatascience.com/memory-attention-sequences-37456d271992) instead of LSTM

Wrap up, Conclusions, Questions?

- This is *crap*.
- Yes, it models the input data fairly well given a split but the predictive model doesn't include the randomness we'd see in the real market
- Could be worthwhile adding in other features (OHLC) and multiple timesteps, but this *explodes* training time.
- It *might* be right about bitcoin going down tomorrow, lemme know if anyone makes any money off it.
- LSTM is *awesome* but it's fiddly. Keras makes life a bit easier but it does constrain a bit.
- I am not an ML expert, I may have made fundamental mistakes
- These models ran a hell of a lot better on a cloud GPU, but there's only so much 'live' I'm willing to risk!

Addendum: Places I was shamelessly ’insipired by’