Question:
Can we predict future price movements in financial markets by analyzing prior price and volume data via data science techniques?

More specifically, we will be looking at intraday data for NSE Nifty index futures, and trying to predict returns (in percentage terms, but also as binary / categorical variables) on several time frames. We have price and volume data, and we will construct features from these.

Raw Data:
The dataset is 1-minute open-high-low-close-volume data downloaded from Bloomberg. The data run from August 3, 2015 to February 1, 2016, inclusive.

The NSE Nifty futures trade from 11:45 Singapore time to 18:00 Singapore time. Each row of data represents 1 minute of trading, and there is typically a row of data reported after 18:00, which represents the daily settlement price.

The original fields downloaded from Bloomberg are Date, OPEN, HIGH, LOW, LAST_PRICE, NUMBER_TICKS, and VOLUME.
OPEN: the first traded price in the one-minute period
HIGH: the highest traded price in the one-minute period
LOW: the lowest traded price in the one-minute period
LAST_PRICE: the last traded price in the one-minute period
It is not clear what NUMBER_TICKS represents.
VOLUME: the number of contracts traded in the one-minute period. Do note that the exchange considers the minimum trade to be 75 lots, but Bloomberg reports this as 1. So, if the VOLUME field in our data shows 100, that means that in that minute, we would have seen trades totaling 7500 lots.

As this is a futures contract, there is a monthly roll. i.e., trading is typically most active in the nearest to expiry contract, and then when that expires, trading activity moves to the next contract. As an example, on August 3, 2015, trading was concentrated in the August 2015 contract “NZQ5” (“Q” is the code for August). Then on August 28, 2015, trading moved to the September 2015 contract “NZU5” (“U” is the code for September), because the last trading day of the August 2015 contract was August 27, 2015. Bloomberg takes care of adjusting the reported prices to make a continuous contract, by adjusting the earlier prices. So, the data would be consistent if we wanted to form longer term indicators (e.g., constructing moving averages over more than 1 futures contract cycle such as a 60-day moving average). But, we will focus on short-term (less than 1-day) predictions, and will using indicators that do not look further back than the current day, so the futures contract roll will not have an impact on our analysis even if it was incorrectly done.

Figure 1 - OHLC chart

In our dataset, the Nifty price has ranged between approximately 7000 – 9000, as shown in Figure 1. Note that the tick size of this contract is very small, at 0.05. Thus, the traded price typically fluctuates through many ticks, and is only very rarely unchanged within a one-minute bar.

	OPEN	HIGH	LOW	LAST_PRICE	NUMBER_TICKS	VOLUME
Count	45,609	45,609	45,609	45,609	45,609	45,609
Mean	7939.33	7941.39	7937.24	7939.31	43.12	904.53
Std	303.85	303.72	303.99	303.84	10.96	2008.37
Min	7237.50	7241.30	7234.55	7238.20	1	-
0.25	7773.75	7775.45	7771.65	7773.70	36	239
0.5	7888.60	7890.95	7886.00	7888.45	45	501
0.75	8136.05	8138.00	8134.40	8136.00	52	1,049
Max	8646.00	8647.60	8643.90	8647.00	64	319,857

Table 1 - Summary of Bloomberg data

Table 1 suggests that there may be outliers in the VOLUME data; it does suggest all other fields seem sensible.

quantile	VOLUME
0.75	1049
0.9	2023
0.95	2971.6
0.975	4132
0.99	6154.96
0.999	13213.26
0.9999	32182.85

Table 2 - VOLUME percentiles

Date	
8/25/2015 14:34	319857
8/25/2015 15:07	60292
9/29/2015 13:30	49179
8/24/2015 11:45	47967
8/21/2015 11:45	36381
8/17/2015 11:52	28895
9/9/2015 11:45	27515
8/4/2015 15:12	24862
8/28/2015 11:45	24682
8/4/2015 16:17	23441

Table 3 - Top 10 VOLUME values

Table 2 and Table 3 together suggest that there is an outlier in VOLUME, we'll drop the highest value of 319,857 as implausible and more likely to be a data error.

Nonetheless, VOLUME takes very high and very low values, such that it only appears normally distributed (with extra weight at volume 0) on a log-log plot:

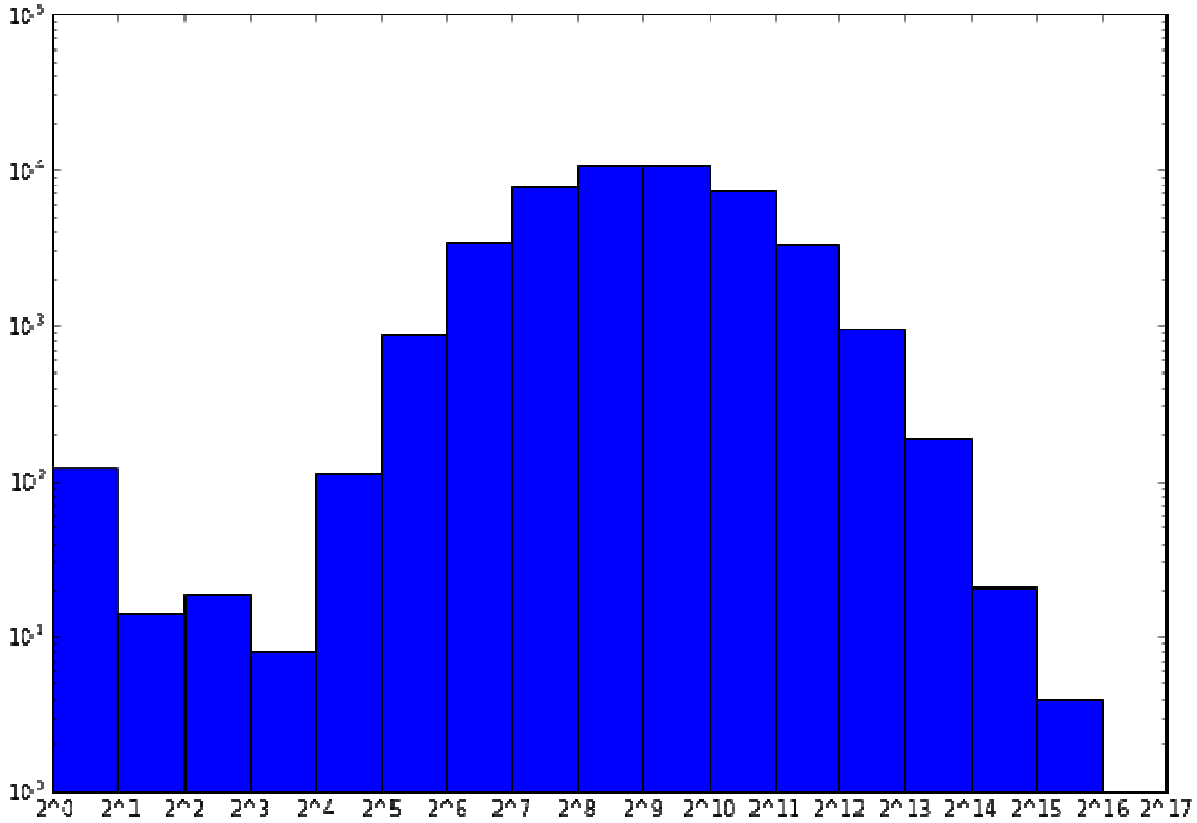

Figure 2- log-log histogram of VOLUME

We may want to consider scaling volume in some cases.

Processing the data

First, we remove the single row where volume was > 300k, as that is likely an outlier. We'll also remove all rows after 1800 (exclusive) as we do not want settlement data – these are not tradable times and these prices are not relevant. We do want the close row (at exactly 1800), because that closing price is important for computing returns on holding periods extending beyond the end of the trading day.

We'll add:
DayOfWeek: 0 for Monday, …, 4 for Friday
Hour: the hour (Singapore time), which ranges from 11 to 18 (but there are few rows at 11 as the market opens at 1145.
We set a time-series index to our dataframe, which will allow us to use 'resample' to create some features on a minute-by-minute basis.

Response variables

We'd like to explore predictability on multiple intraday time frames. We'll standardize this list to [1, 5, 10, 20, 40, 80]. i.e., as responses, we create the 1-minute ahead percentage return, but also the 5-minute, …, and 80-minute ahead percentage returns. This will be, for any given row, the (LAST_PRICE x-minutes ahead) / (current LAST_PRICE) – 1. In cases where the x-minute-ahead period extends beyond the end of the trading session, we'll use the LAST_PRICE at 1800 to compute the return. (This is accomplished by 'ffill' in the resample method.)

Table 4 and Figure 3 show the x-minute ahead returns, these appear sensible.

	1_min_return	5_min_return	10_min_return	20_min_return	40_min_return	80_min_return
count	45341	45338	45333	45324	45305	45265
mean	-0.0002%	-0.0011%	-0.0022%	-0.0042%	-0.0068%	-0.0153%
std	0.0412%	0.0898%	0.1241%	0.1730%	0.2438%	0.3411%
min	-0.5672%	-0.9208%	-1.1510%	-1.4103%	-1.7253%	-2.1843%
25%	-0.0208%	-0.0464%	-0.0633%	-0.0860%	-0.1179%	-0.1681%
50%	0.0000%	-0.0006%	-0.0006%	-0.0013%	-0.0020%	-0.0094%
75%	0.0208%	0.0453%	0.0611%	0.0815%	0.1092%	0.1459%
max	1.3354%	1.4400%	1.5734%	1.6781%	1.3665%	1.8271%

Table 4 - x-min return summary

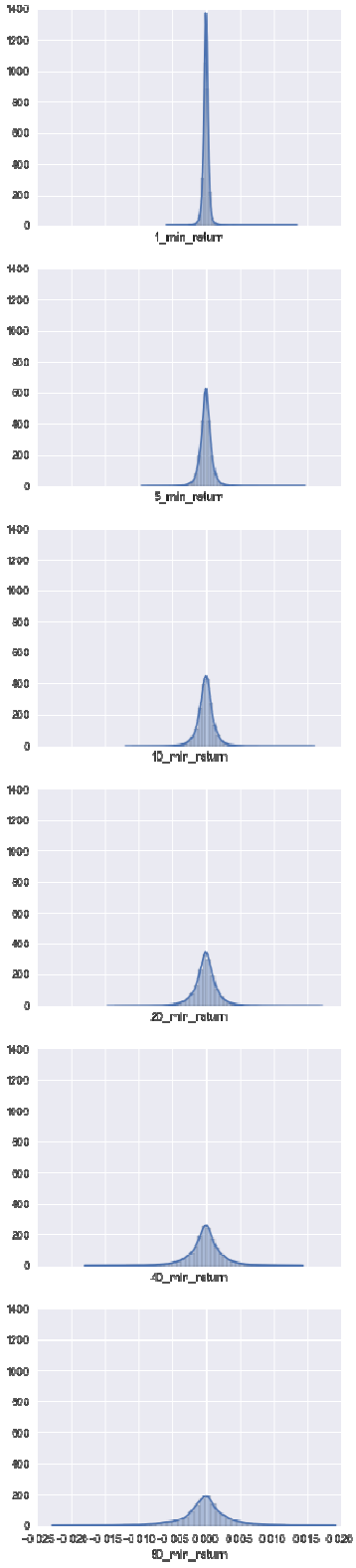

Figure 3 - x-min ahead returns

We look at skewness, kurtosis, and normality of the returns, with the following results:

```
We are using Fisher kurtosis, so a sample from a  normal distribution should
report a 0 kurtosis on this measure
For 1_min_return, skewness is 0.548331; kurtosis is 30.846228
For 5_min_return, skewness is 0.000061; kurtosis is 10.517373
For 10_min_return, skewness is -0.138187; kurtosis is 7.741777
For 20_min_return, skewness is -0.158083; kurtosis is 5.673287
For 40_min_return, skewness is -0.205899; kurtosis is 4.020208
For 80_min_return, skewness is -0.257549; kurtosis is 3.373104

For 1_min_return, p-value of rejecting hypothesis that sample is from a norma
l distribution is 0.000000.
For 5_min_return, p-value of rejecting hypothesis that sample is from a norma
l distribution is 0.000000.
For 10_min_return, p-value of rejecting hypothesis that sample is from a norm
al distribution is 0.000000.
For 20_min_return, p-value of rejecting hypothesis that sample is from a norm
al distribution is 0.000000.
For 40_min_return, p-value of rejecting hypothesis that sample is from a norm
al distribution is 0.000000.
For 80_min_return, p-value of rejecting hypothesis that sample is from a norm
al distribution is 0.000000.
```

Figure 4 - distributional tests on x-minute returns

i.e., the x-minute return data are heavy tailed and thus not normal.

We'll also create 'x_min_updown' columns which are coded +1 if the x-min return was positive, -1 if it was negative, and is blank if it was exactly 0. This ignores very few returns:

```
1_min_updown has 1124 (2.48%) missing values
5_min_updown has 538 (1.19%) missing values
10_min_updown has 453 (1.00%) missing values
20_min_updown has 379 (0.84%) missing values
40_min_updown has 356 (0.79%) missing values
80_min_updown has 351 (0.78%) missing values
```

Figure 5 - missing values in x-minute binary variables

	1_min_updown	5_min_updown	10_min_updown	20_min_updown	40_min_updown	80_min_updown
count	44344	44945	45031	45104	45131	45289
mean	0.500113	0.494872	0.495858	0.493814	0.492455	0.482523
std	0.500006	0.499979	0.499988	0.499967	0.499949	0.4997
min	0	0	0	0	0	0
25%	0	0	0	0	0	0
50%	1	0	0	0	0	0
75%	1	1	1	1	1	1
max	1	1	1	1	1	1

Table 5 - x-minute binary variables summary

Table 5 shows no obvious issues with the binary variables.

To get a sense of the baseline, we look at the percent of returns that are up:

```
For 1 min returns: +ve: 22176; -ve: 22167; i.e., 50.01% up
For 5 min returns: +ve: 22235; -ve: 22694; i.e., 49.49% up
For 10 min returns: +ve: 22319; -ve: 22695; i.e., 49.58% up
For 20 min returns: +ve: 22265; -ve: 22823; i.e., 49.38% up
For 40 min returns: +ve: 22220; -ve: 22891; i.e., 49.26% up
For 80 min returns: +ve: 21765; -ve: 23351; i.e., 48.24% up
```

Figure 6 - percent positive binary returns

So, the returns are almost 50% up

We also make categorical variables for the return, where we code the data as '2' if it's a significantly positive return, '1' if it's a small return, and '0' if a significantly negative return. The cut-offs for these classifications are based on percentiles of absolute values, such that approximately 1/3 of the data are '2', 1/3 are '1', and 1/3 are '0'. Figure 7 shows the distribution of the categorical variables, there do not seem to be any obvious issues.

```
For 1 min returns: +1: 15169; 0: 15114; -1: 15057
For 5 min returns: +1: 14994; 0: 15113; -1: 15230
For 10 min returns: +1: 14943; 0: 15111; -1: 15278
For 20 min returns: +1: 14876; 0: 15108; -1: 15339
For 40 min returns: +1: 14794; 0: 15102; -1: 15408
For 80 min returns: +1: 14302; 0: 15088; -1: 15874
```

Figure 7 - count of categorical return variables

Finally, we make 'high return' categorical variables for the returns. In this case we code the data as '2' if it's a very high positive return, '1' if it's a moderate positive or negative return, and '0' if it's a very high negative return. Again these are based on percentiles of absolute values – but here 5% of the data are '2', 5% are '0', and 90% are 1. This represents 'high opportunity trades'. Figure 8 shows no obvious issues with the categorization.

```
For 1 min returns: +1: 2209; 0: 40806; -1: 2325
For 5 min returns: +1: 2198; 0: 40804; -1: 2335
For 10 min returns: +1: 2137; 0: 40799; -1: 2396
For 20 min returns: +1: 2082; 0: 40791; -1: 2450
For 40 min returns: +1: 2093; 0: 40774; -1: 2437
For 80 min returns: +1: 1974; 0: 40738; -1: 2552
```

Figure 8 - count of high categorical return variables

Feature variables

Our first features are the x-minute prior returns, and their binary versions. i.e., for a given minute, we compare the current LAST_PRICE to the LAST_PRICE x minutes ago, and compute the percent change. That is x_min_prior. The binary version is x_min_prior_updown, and simply is 1 if x_min_prior is positive, 0 if it was negative, and missing otherwise.

	1_min_prior	5_min_prior	10_min_prior	20_min_prior	40_min_prior	80_min_prior

count	45466	45462	45457	45447	45427	45387
mean	-0.000002	-0.000009	-0.000018	-0.000036	-0.000059	-0.000083
std	0.000411	0.000904	0.001257	0.001757	0.00247	0.00341
min	-0.005705	-0.009293	-0.011644	-0.014304	-0.017556	-0.022331
0.25	-0.000207	-0.000466	-0.000638	-0.000873	-0.001207	-0.001685
0.5	0	0	0	-0.000006	-0.000007	-0.000076
0.75	0.000206	0.000459	0.000625	0.000848	0.001168	0.00161
max	0.013178	0.014195	0.01549	0.016504	0.013481	0.017943

Table 6 - x-minute prior return summary

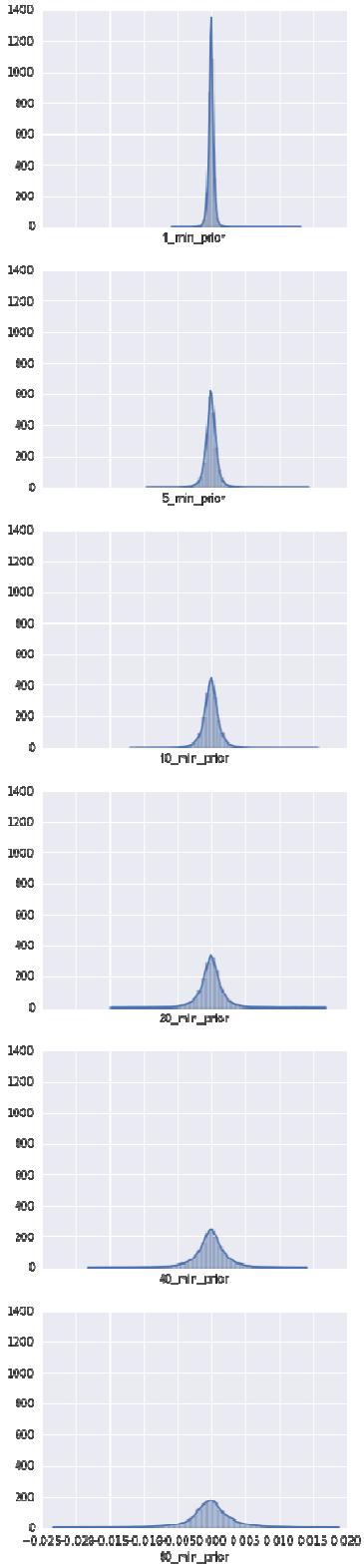

Figure 9 - x-minute prior distribution

Table 6 and Figure 9 show the prior returns, there are no red flags here.

We next make x-minute moving average columns. So, this is the average LAST_PRICE over the last x minutes, including the current minute (because, we assume you would put on a trade at the LAST_PRICE). When we are at times near the start of the day, and there are < x minutes available, we just average over all minutes since the open. So, if it is 1148, the 5-minute moving average would only average over the LAST_PRICE for 1145, 1146, 1147, and 1148. Table 7 summarizes these columns – there seems to be no cause for concern. Figure 10 shows a sample of these series.

	1_min_ma	5_min_ma	10_min_ma	20_min_ma	40_min_ma	80_min_ma
count	45467	45467	45467	45467	45467	45467
mean	7939.34	7939.38	7939.42	7939.52	7939.67	7939.86
std	303.86	303.85	303.85	303.84	303.83	303.85
min	7238.20	7241.47	7242.00	7247.19	7252.31	7264.39
25%	7773.78	7773.52	7773.52	7773.31	7773.77	7773.96
50%	7888.60	7888.72	7888.74	7888.64	7889.38	7890.20
75%	8136.00	8136.17	8135.83	8135.95	8136.53	8139.01
max	8647.00	8644.35	8643.02	8642.07	8641.23	8639.23

Table 7 - x-minute moving average summary

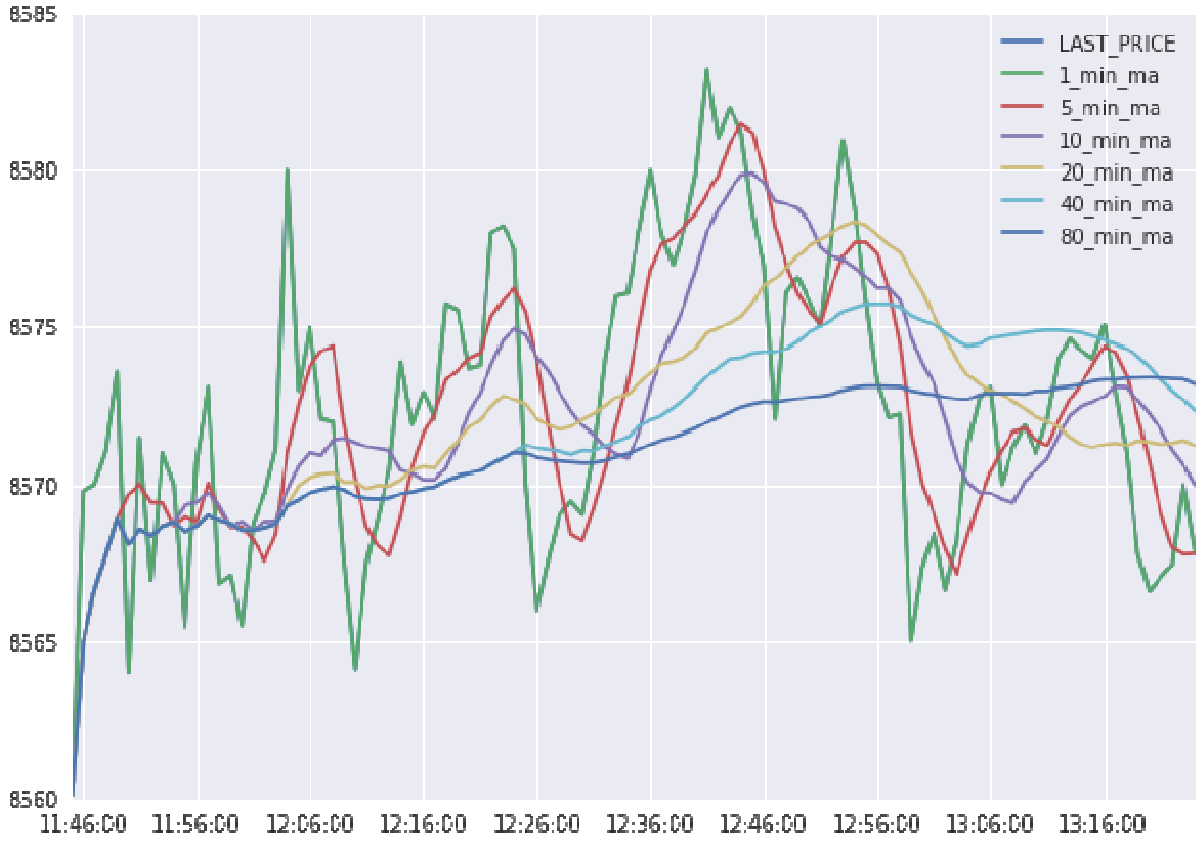

Figure 10 - example of moving averages

We also create x-minute difference from moving average columns, and binary ("updown") versions of these.

	1_last_vs_ma	5_last_vs_ma	10_last_vs_ma	20_last_vs_ma	40_last_vs_ma	80_last_vs_ma
count	45,467	45,467	45,467	45,467	45,467	45,467
mean	0.00	-0.03	-0.08	-0.17	-0.33	-0.51
std	0.00	3.51	5.30	7.63	10.87	15.18
min	0.00	-51.77	-59.05	-74.44	-92.14	-122.13
25%	0.00	-1.75	-2.66	-3.74	-5.18	-7.11
50%	0.00	0.00	-0.03	-0.04	-0.05	-0.25
75%	0.00	1.73	2.61	3.59	4.89	6.64
max	0.00	82.23	94.20	104.04	112.01	109.29

Table 8 - price vs moving average summary

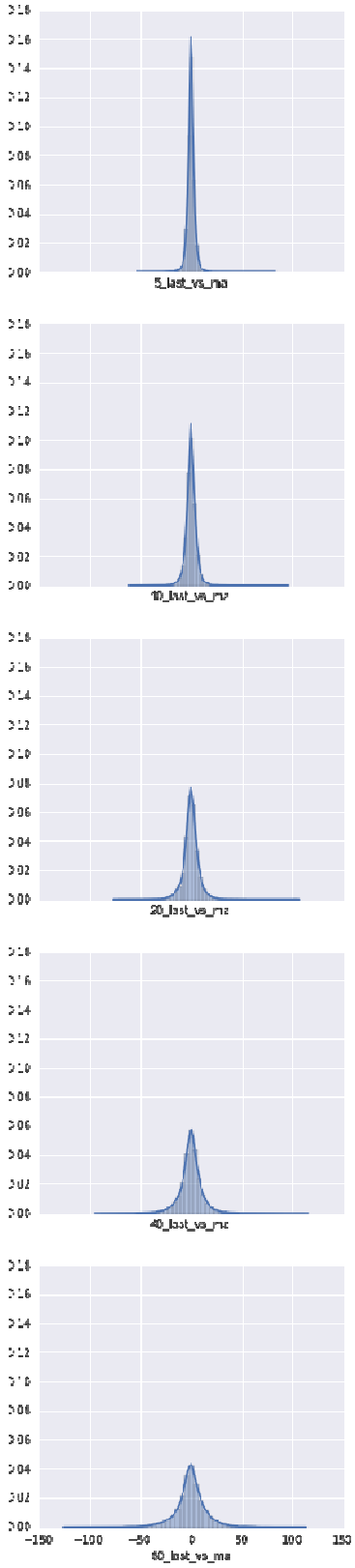

Figure 11 - last price vs moving average distributions

Table 8 summarizes the difference from moving average columns, Figure 11 plots their distributions. This looks fine; we shouldn't use the 1-minute columns (this is just the price minus itself).

We add some features for volume data. These are the rolling 1, 5, …, 80 minute average volume columns. We'll also add some columns where we average the log of (volume + 1) over these periods. We need to add 1 to make sure we get defined numbers in cases where volume = 0. We do these averages by taking rolling sum over rolling count. So, at the beginning of the day, when less than x minutes have elapsed, the x minute average will just be the average over the number of minutes elapsed since the start of the day.

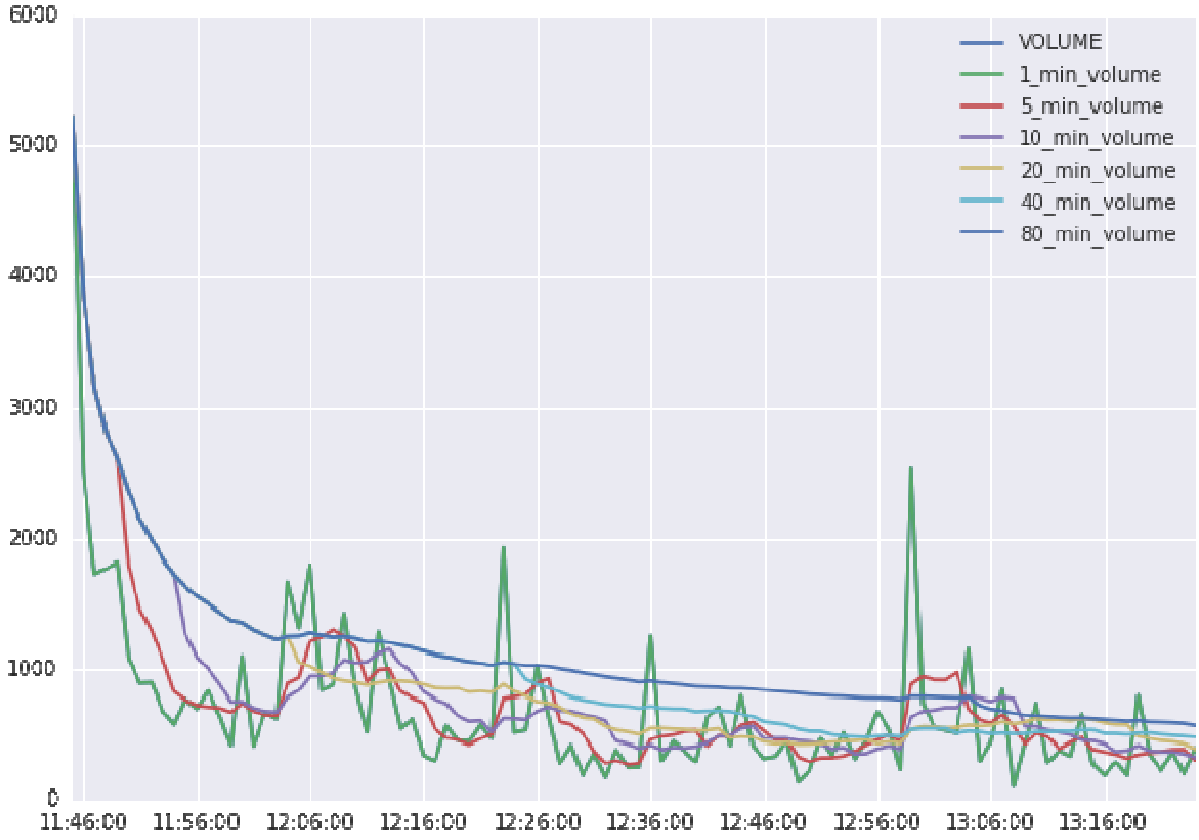

Figure 12 – volume

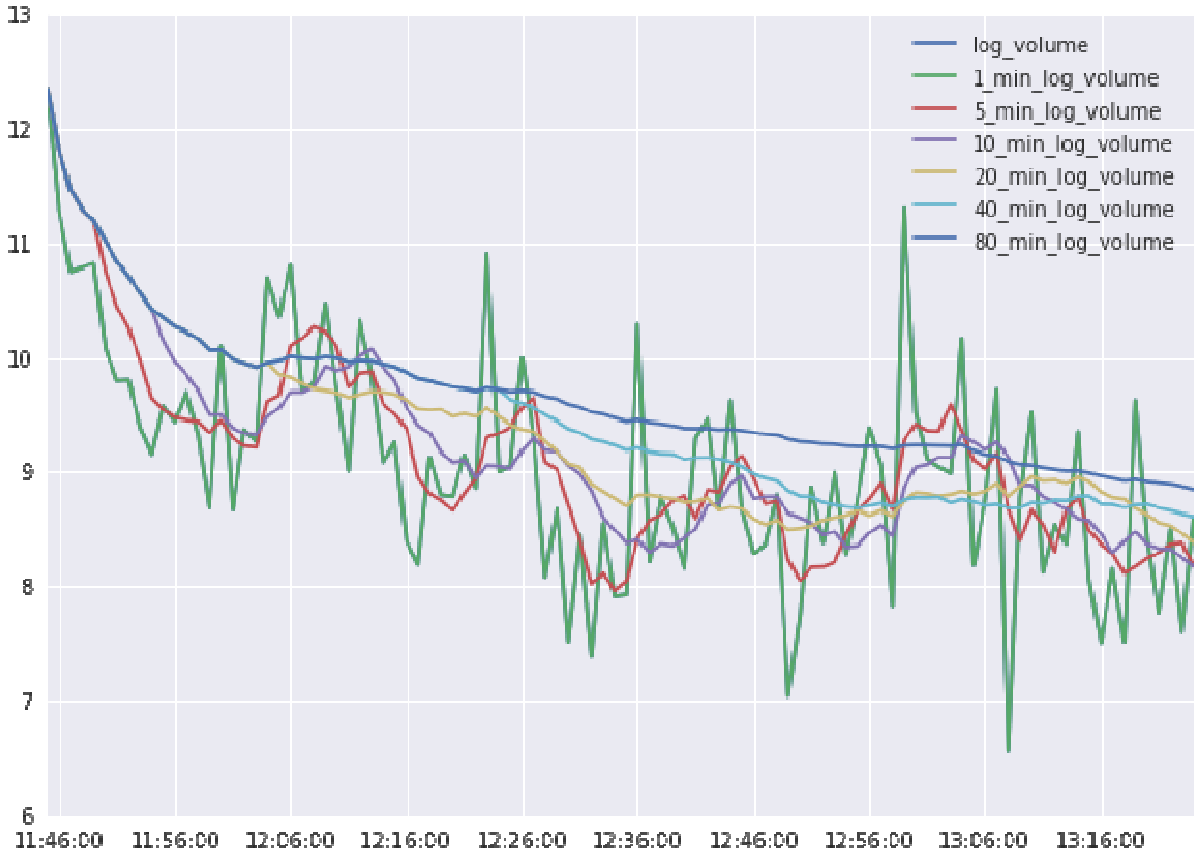

Figure 13 - log volume

Figure 12 and Figure 13 show examples of the volume and log volume columns. As volume varies quite wildly over each minute, log volume could be preferred as a feature as it has less extreme variations which may cause some analyses to be led astray.

We'll look at volume versus day of week and time of day. Figure 14 demonstrates that mean volume over x-minute periods does not fluctuate much by day of week. But, volume does vary considerably by hour, as seen in Figure 15. We can see in Figure 16 that even the log-volume has different distributions by hour. We'd probably want to produce a variable telling us how the current x-minute volume compares to some prior distribution of x-minute volumes, given the current hour of day. This is so that we can express in a meaningful day that 'x-minute volume was high, considering the current hour of day.' If we did not condition on hour of day, our indicator of 'high volume' would be true a lot near the beginning and end of day, and mostly false in the middle of the day. It might be useful to be able to indicate 'unusually high volume' as this, combined with some other indicator, could be predictive of the next move. E.g., if there is a big prior return on high volume, perhaps this move is likely to continue. But, if there is a big prior return on low volume, maybe this move is more likely to reverse. Figure 17 shows this 'time-adjusted' log-volume by hour of day. Clearly, the distribution is much more constant across hour of day; a high value on this variable does mean unusually high volume *given the current hour of day*. Note that we've not created this variable for the rows with time >= 1800, as this is not really relevant for our predictions, being volume at the closing minute / settlement minute.

Given what we observe about volume by time of day, we'll create indicator variables for 'near the open' and 'near the close'. The former will be 1 before (and including) 1230, 0 otherwise. The latter will be 1 from (and including) 1630, 0 otherwise. Looking ahead, Figure 24 will corroborate that there is some difference in market characteristics at these times – in particular that the distribution of returns is more variable than at other times,

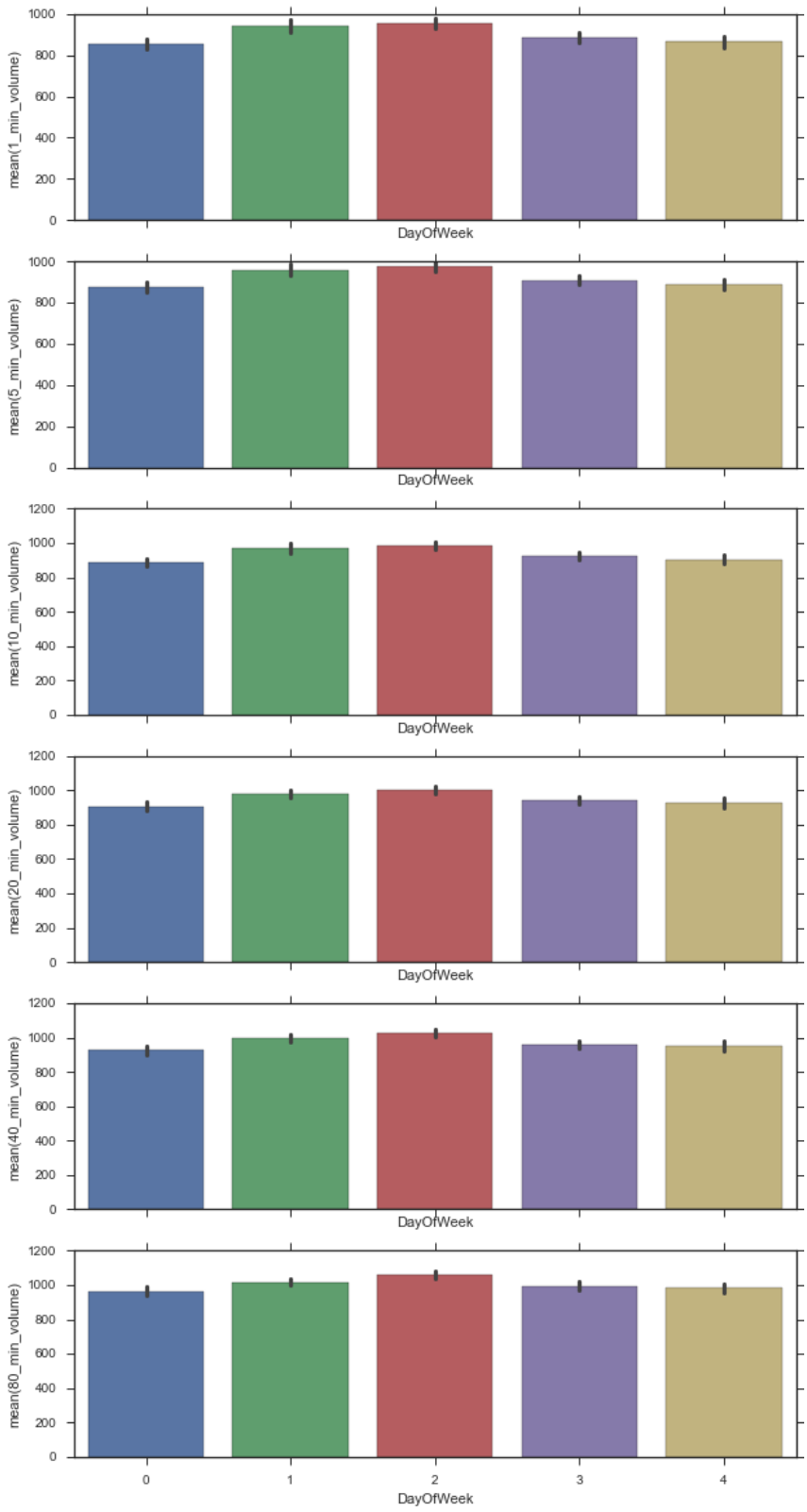

Figure 14 - Volume by day of week

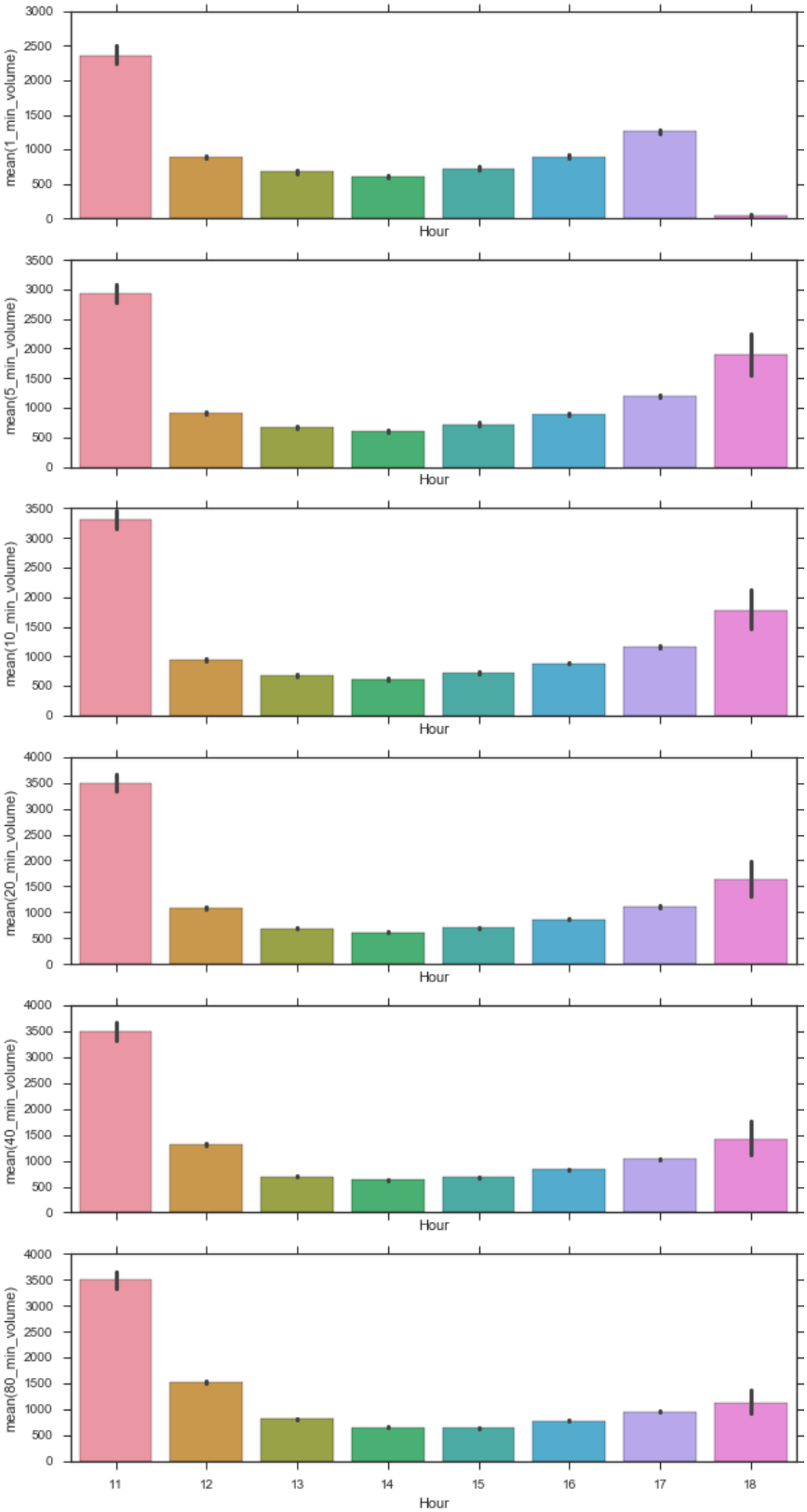

Figure 15 - Volume by hour

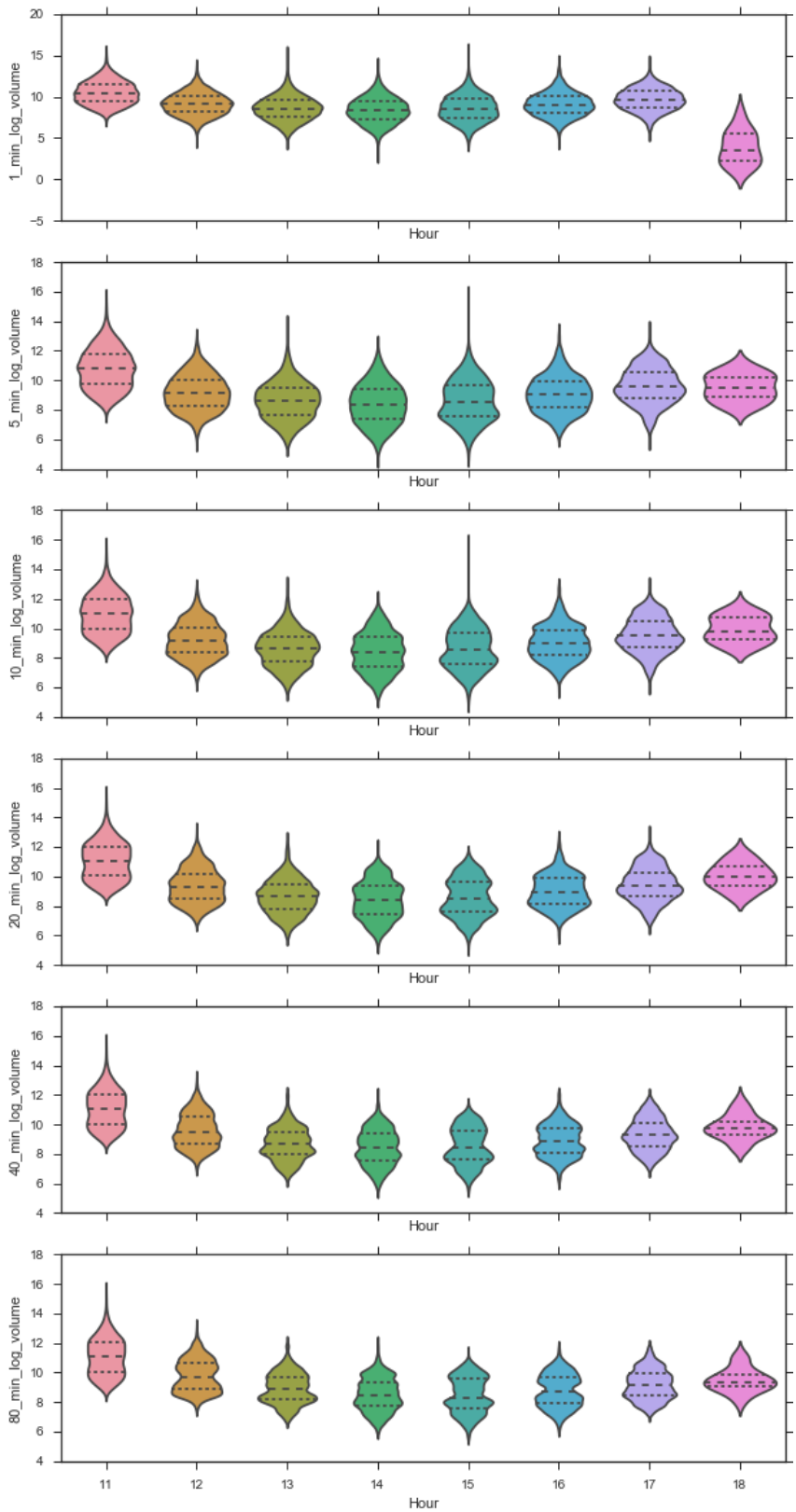

Figure 16 - log-volume distributions by hour

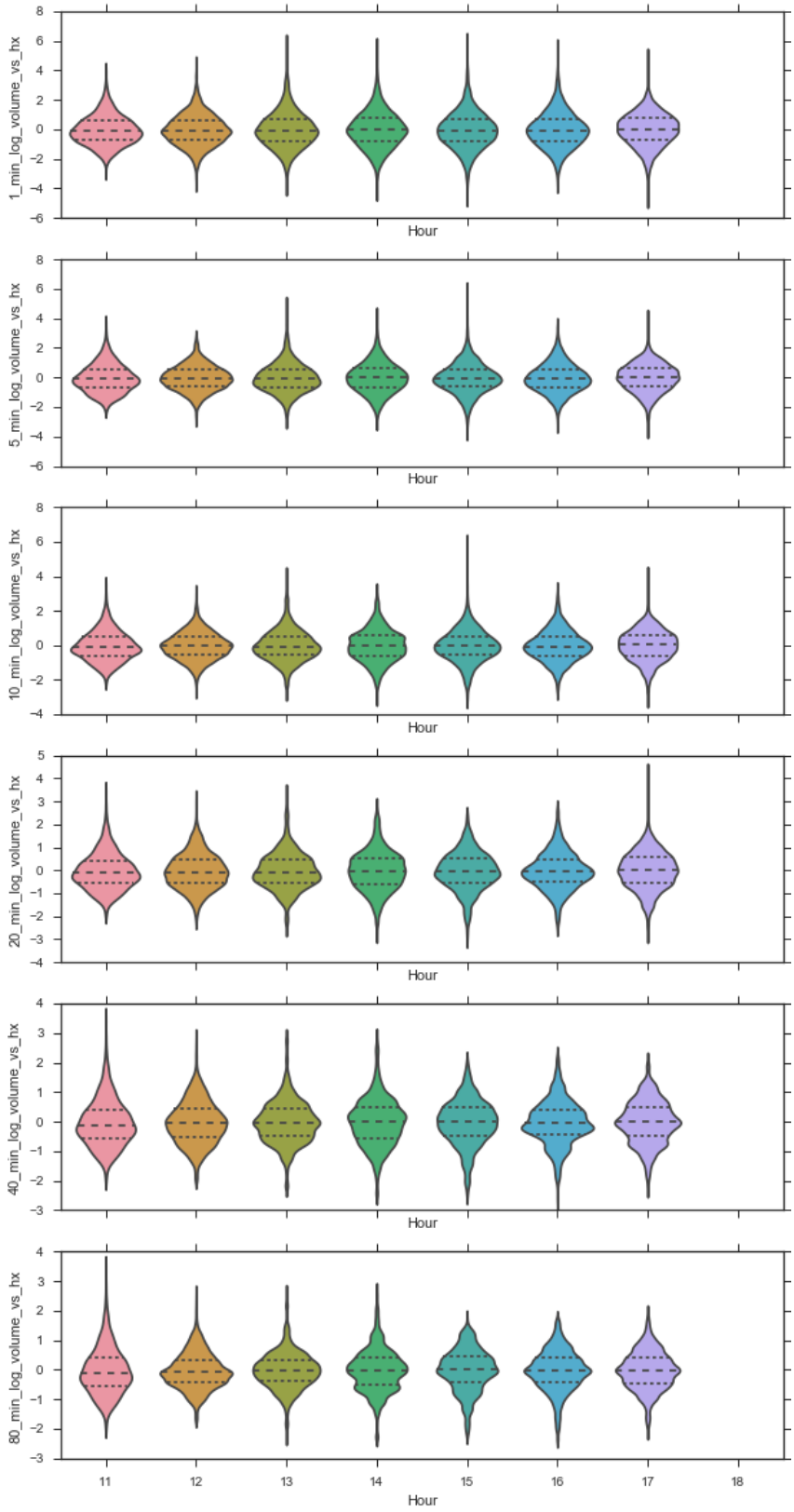

Figure 17 - time-adjusted log-volume by hour distributions

The next set of columns represent buying up / selling down behavior. This is, loosely, a measure of how much volume went behind a particular directional move. It's possible that upmoves on high volume are more predictive of subsequent upmoves than upmoves on low volume. So, we'll mark a minute as 'up' if its LAST_PRICE is higher than the LAST_PRICE in the previous minute. For the opening minute of the day, we'll compare LAST_PRICE to OPEN. Then, we sign the volume in that minute as the raw 'busd' value. E.g., if LAST_PRICE at 1148 is 8889 and LAST_PRICE at 1149 is 8888, and VOLUME in the 1149 minute is 1000, then 'busd' is -1000. i.e., we simply return the volume in an up minute, and -1 times the volume in a down minute. Then we average over x minutes. We'll do the same for log volume too. Again, log values may be better for some analyses, as seen in Figure 18 and Figure 19. Finally, we'll look at busd_time, which is our time-adjusted log-volume measure over 1-minute, multiplied by the sign of the 1-minute prior return, then averaged over 1-minute, 5-minutes, etc. An example for this is shown in Figure 20.

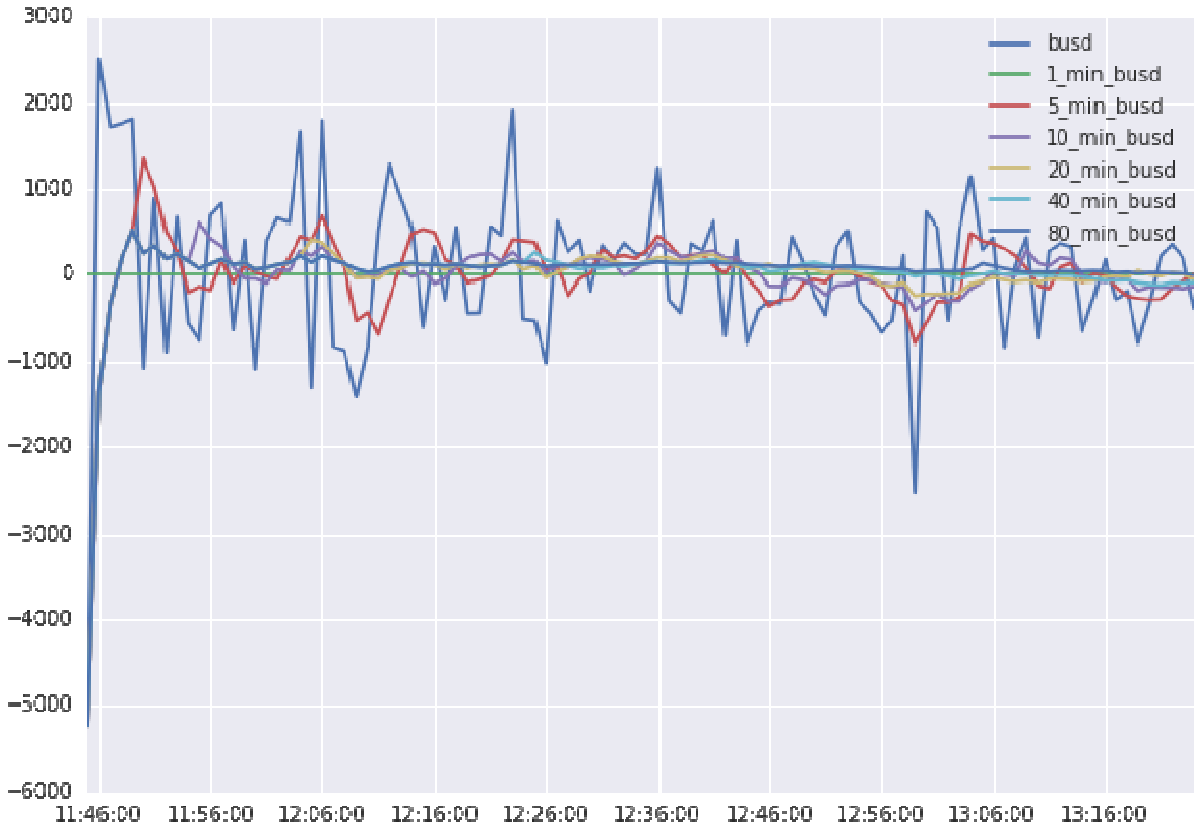

Figure 18 – buying up / selling down example

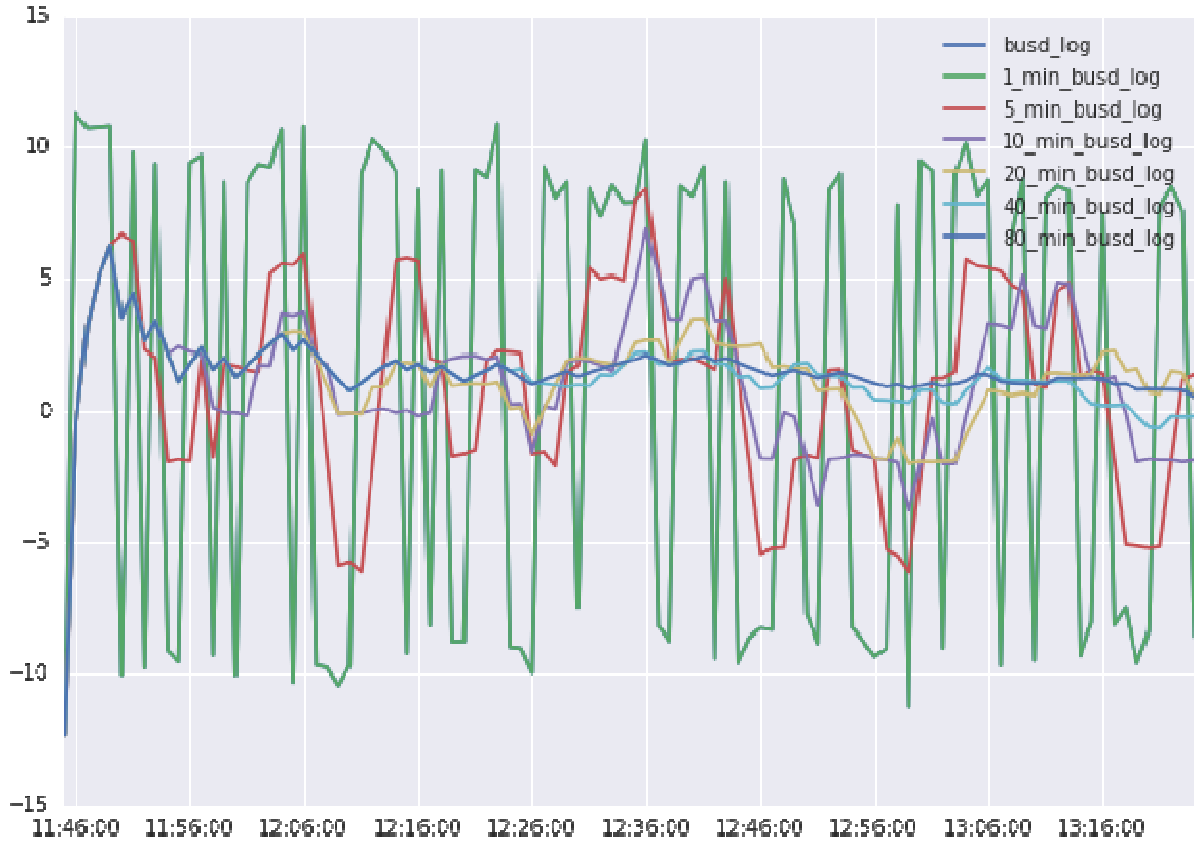

Figure 19 - log buying up / selling down example

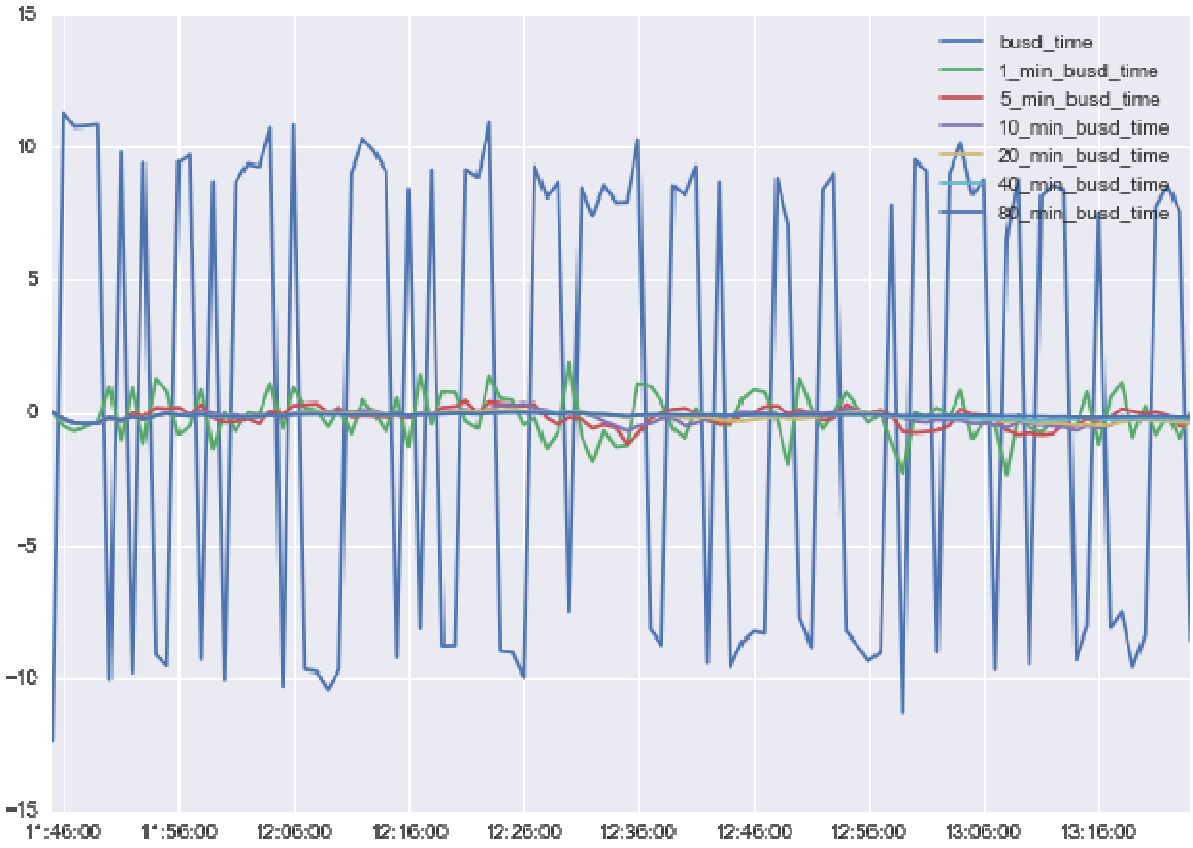

Figure 20 - busd time example plot

Next we look at high-low ranges over x-minute prior periods. To construct this we take the difference between the maximum of the x-minute HIGH less the minimum of the x-minute LOW. This may be useful as it proxies the recent level of price volatility. The distributions of these features are shown in Figure 21, they're heavy right tailed.

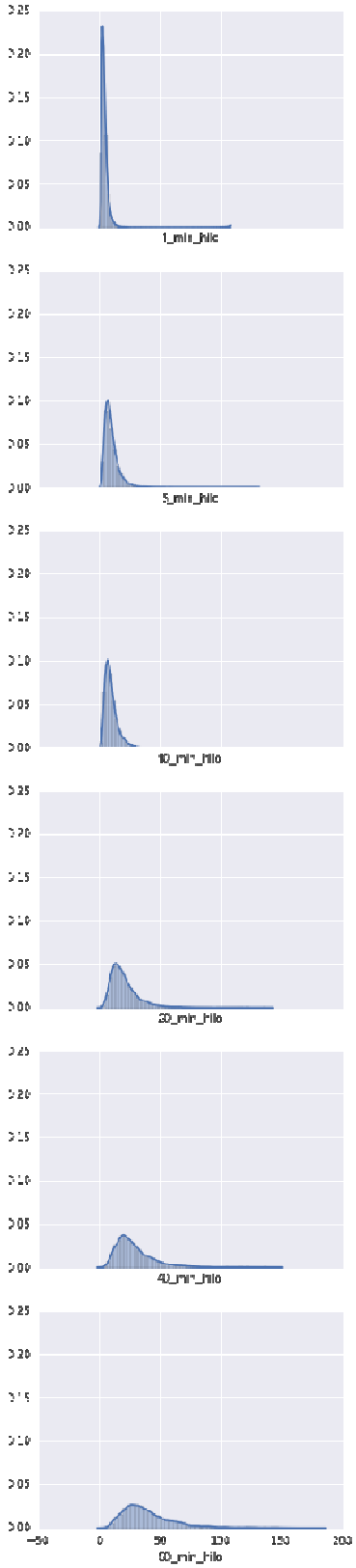

Figure 21 - high-low range distributions

The final set of indicators is the current LAST_PRICE relative to the 1-minute-ago x-minute high-low range. E.g., if at 1241 the LAST_PRICE is 8801, and between 1200 and 1240 the highest price was 8800 and the lowest price was 8700, then we return 1.01. We scale this such that if the current price matches the high, it will be 1; if it matches the low, it will be -1; if it's at the mean of the high and low, it will be 0; and so on. This can be a good proxy for breakouts of a range, which may suggest continuation of previous movements. Figure 22 provides an example at the 10-minute timeframe.

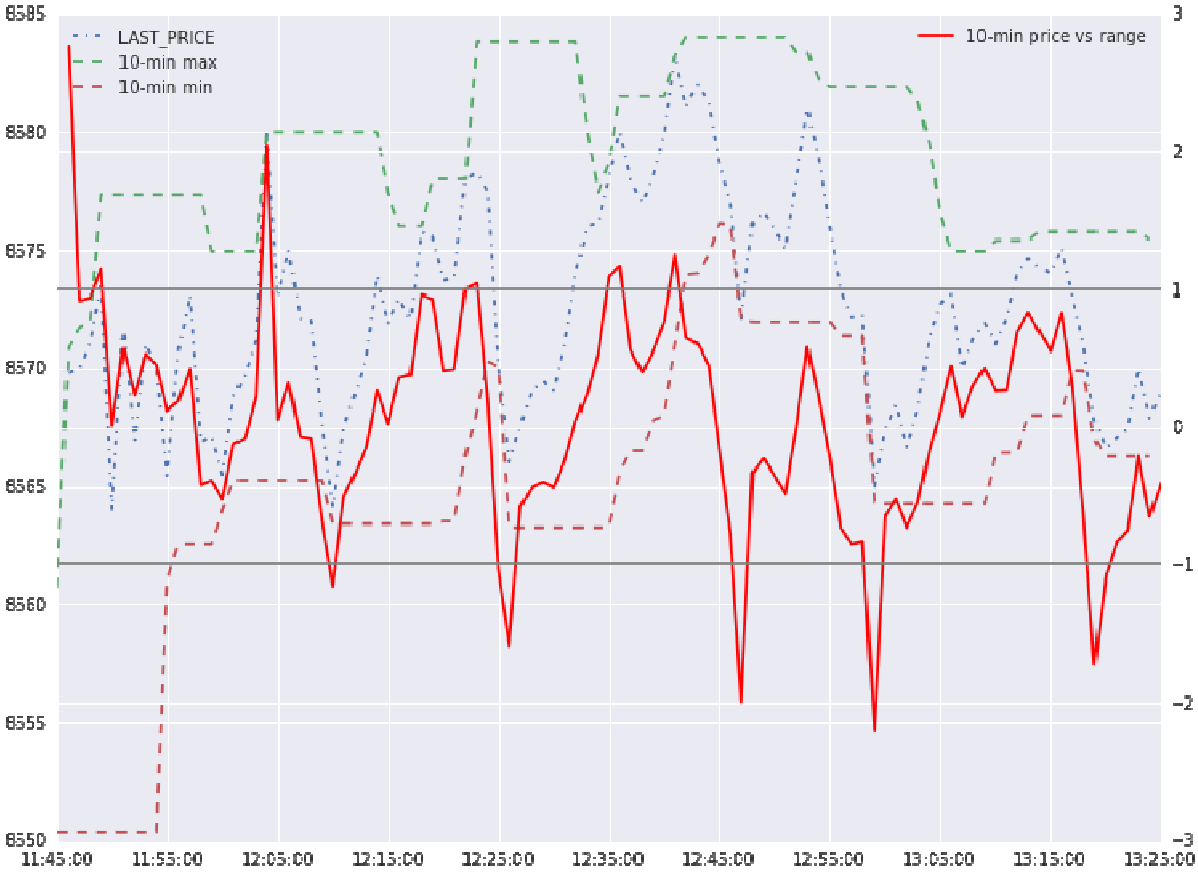

Figure 22 - Price vs range example

Exploratory Analysis

Figure 23 shows violinplots of the x-minute returns versus day of week. The width of the violin at a give return level indicates the density of the distribution i.e., returns near 0 are more common than extreme returns. There isn't any obvious pattern by day of week. There are some large returns i.e., the distribution is heavy tailed, but the magnitude of returns does not suggest outliers that should be removed. (e.g., 1.5% return within 1 minute is plausible, this is an extreme event but not unreasonable.)

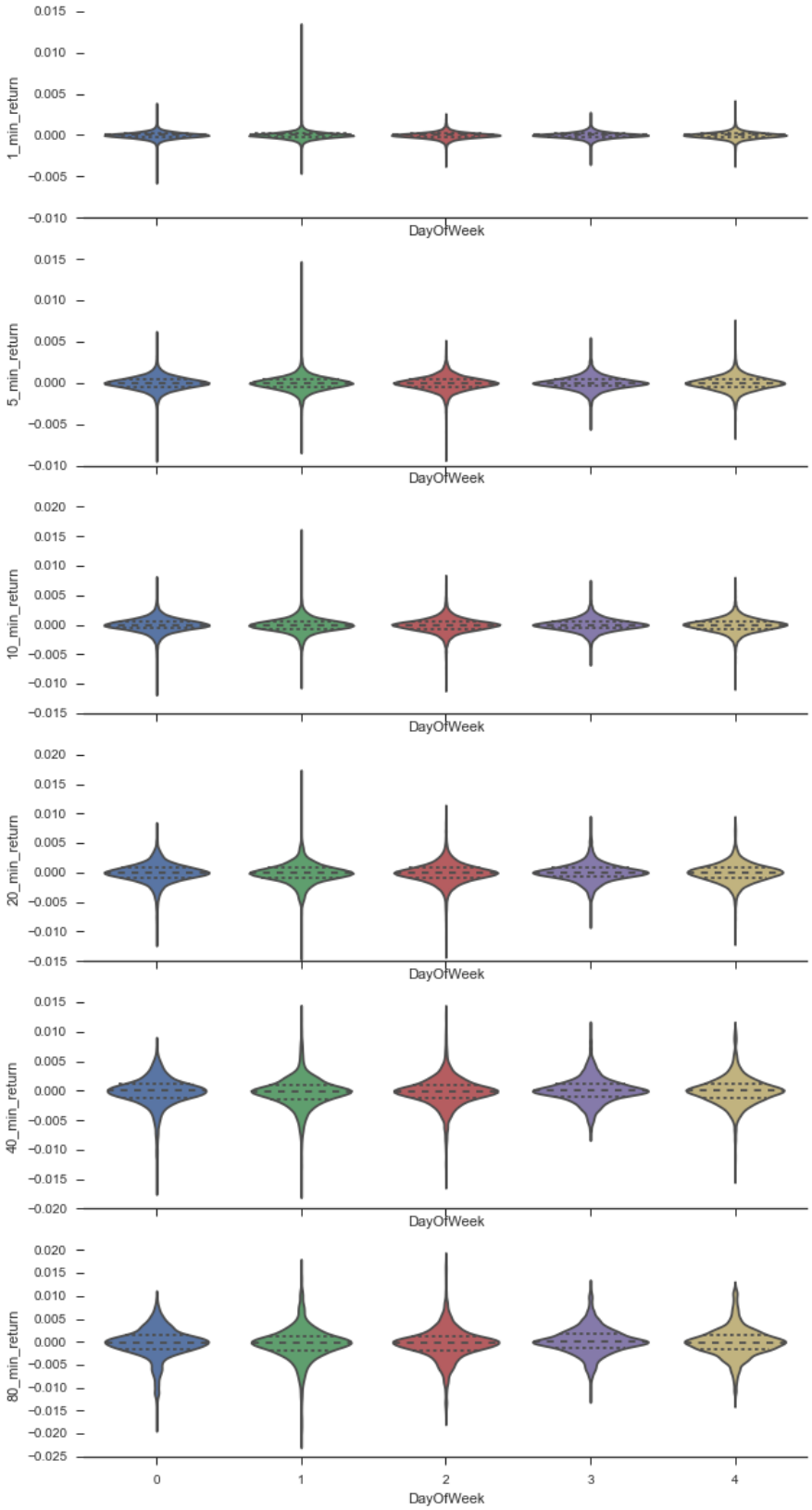

Figure 23 - return vs Day of Week

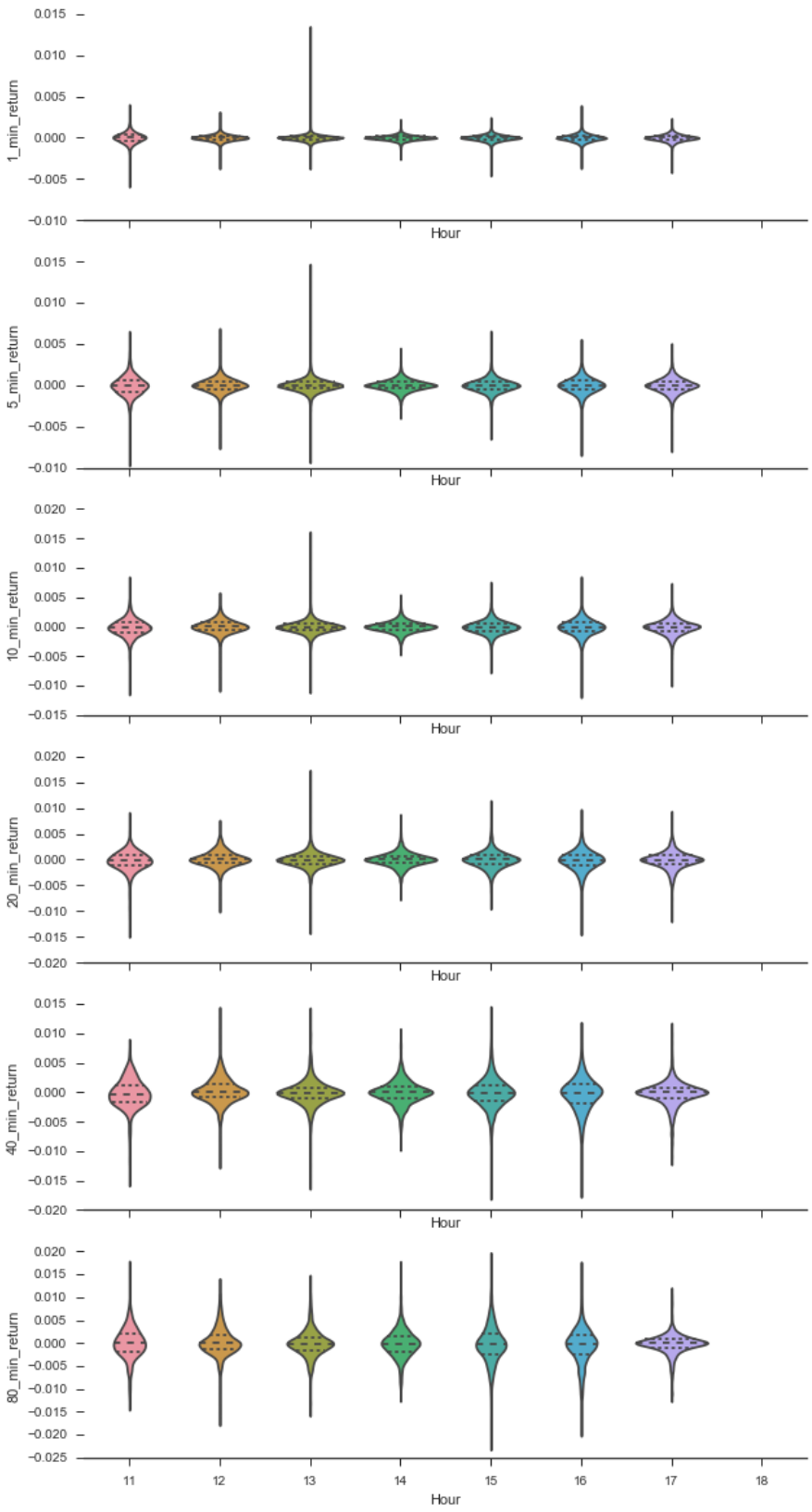

Figure 24 - x-minute return by hour

Figure 24 shows the distribution of x-minute returns by hour. There's no trend of higher or lower returns by hour, but there may be some systematic tendency to have lower dispersion of returns nearer the middle of the day (around hour 14). i.e., returns are more variable near the open (hour 11) and close (hour 17). Figure 25 explores this more closely – the heatmap shows the standard deviation of x-minute returns by hour of day. It's reasonably clear that standard deviation of returns are lower in the middle of the day than at nearer the open or close.

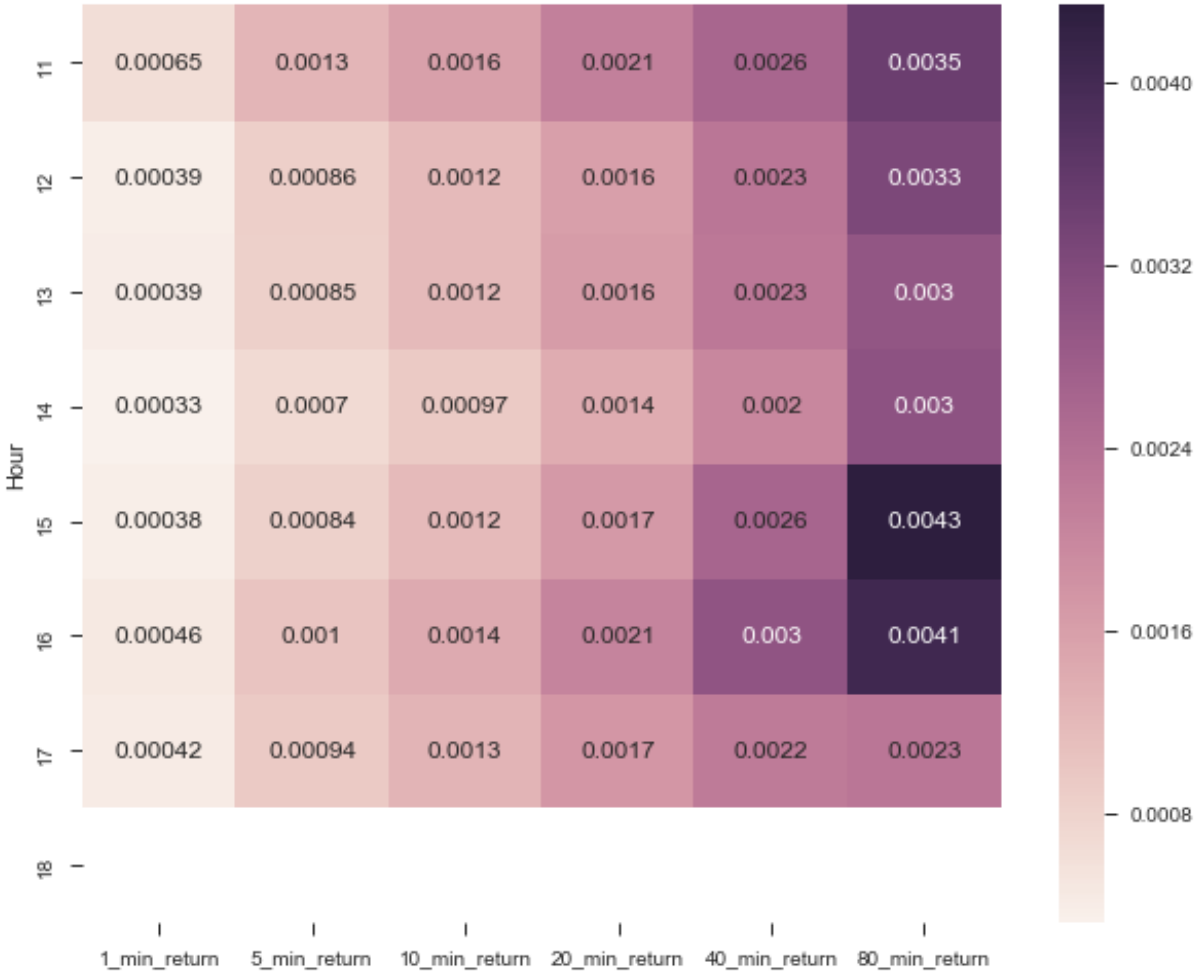

Figure 25 - standard deviation of x-minute returns by hour

Now we look at the relationship between x-minute prior return and x-minute (ahead) returns.

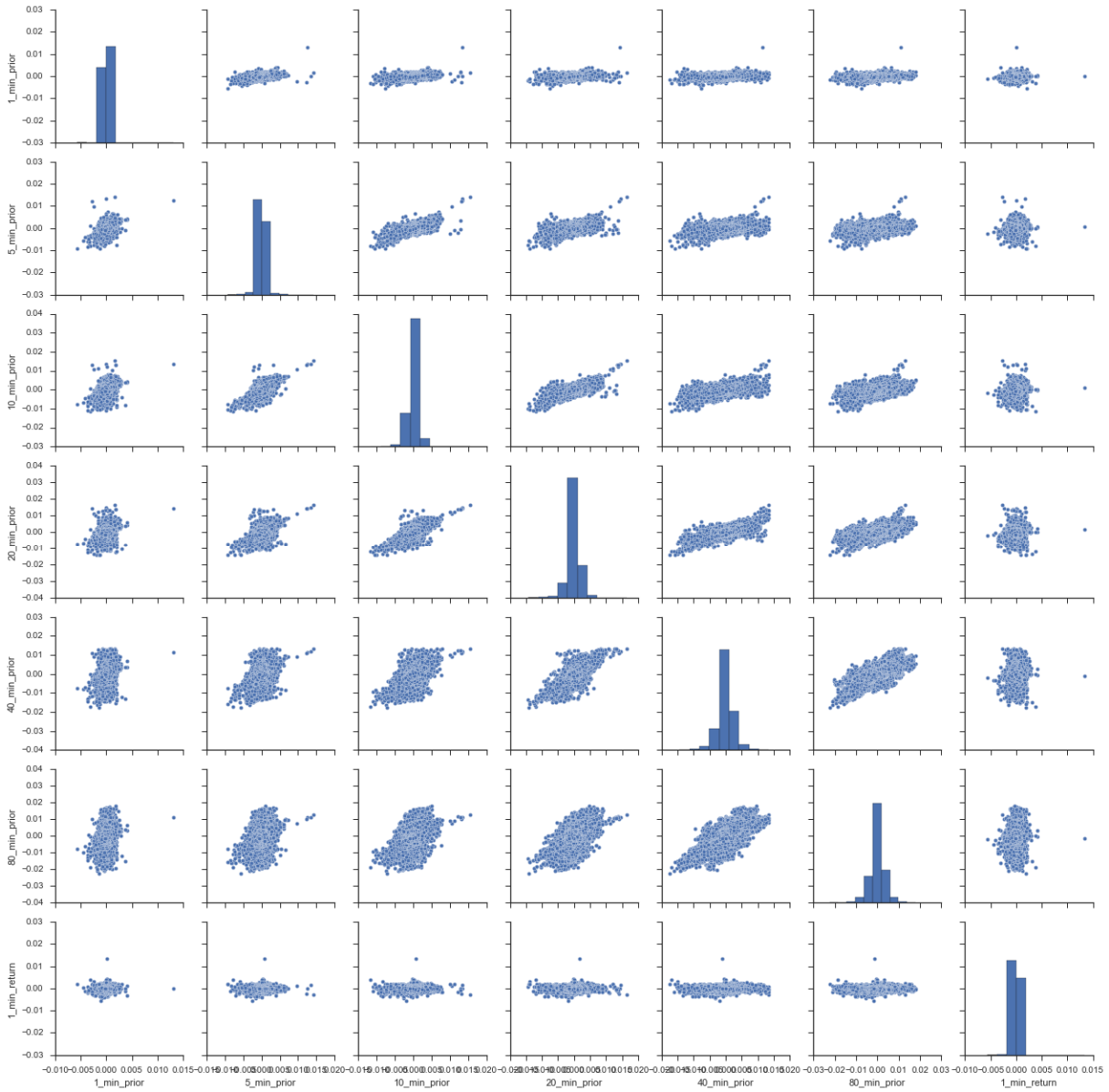

Figure 26 - 1-minute return vs x-minute prior return scatterplot matrix

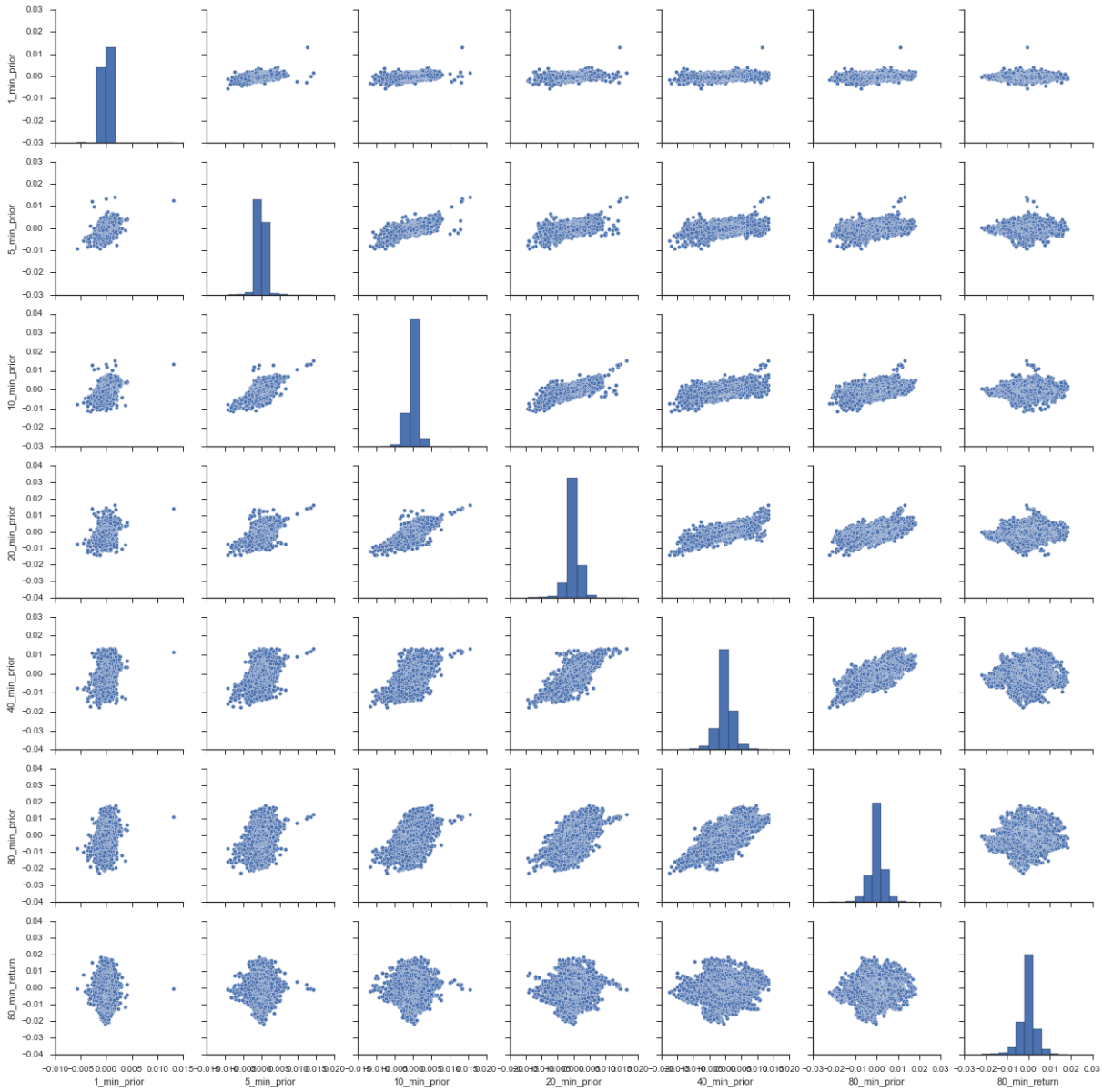

Figure 27 - 80-minute return vs x-minute prior scatterplot matrix

Figure 26 shows the 1-minute (ahead) return and all the x-minute prior returns in a scatterplot matrix. The x-minute prior returns are positively correlated with each other, which makes sense – if the 80-minute prior return was positive, it's much more likely that the 40-minute prior return was also positive, and so on. But, none of these x-minute prior returns seem to exhibit any visible pattern with the 1-minute return. Figure 27 shows the same scatterplot, but for 80-minute (ahead) returns instead. Here the scatterplots of x-minute prior returns versus 80-minute ahead returns are potentially more interesting, but the relationship is not clear. Figure 28 explores these relationships a little more closely – there is a small positive correlation between prior returns and 80-minute ahead returns; the longer the prior return the higher the correlation with the 80-minute ahead returns. This is consistent with a small momentum effect.

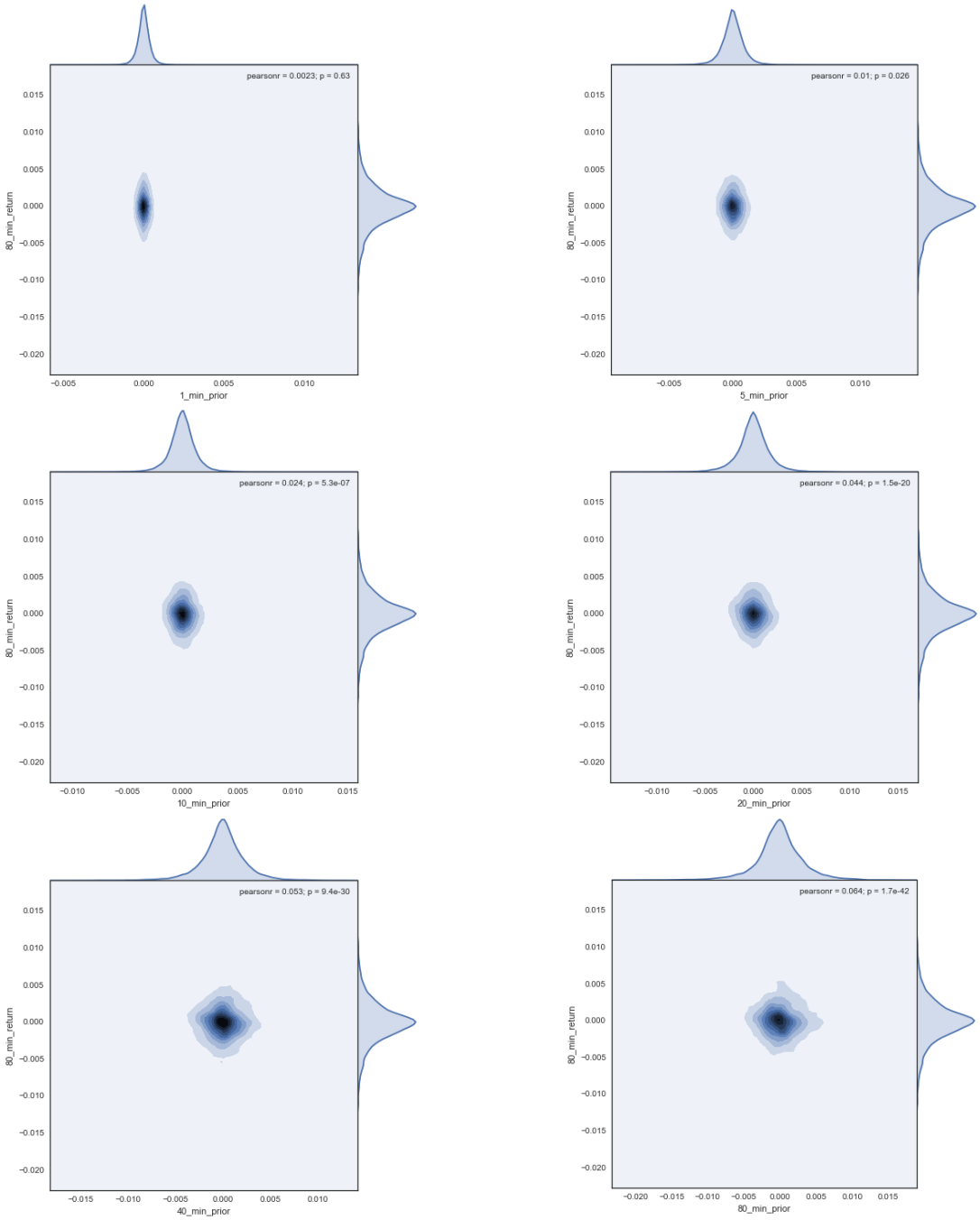

Figure 28 - kdeplots of x-minute prior vs 80-minute returns

Let's explore this more generally, for all timescales. Figure 29 shows that there is a small negative correlation between short term prior returns and short term ahead returns, and a small positive relation between longer-term prior returns and longer-term ahead returns. i.e., in the short-term there is mean-reversion, and in the longer term there is momentum. These effects are quite small. But, the mean-reversion of short-term prior returns and momentum of longer-term returns is statistically significant, as seen in Figure 30.

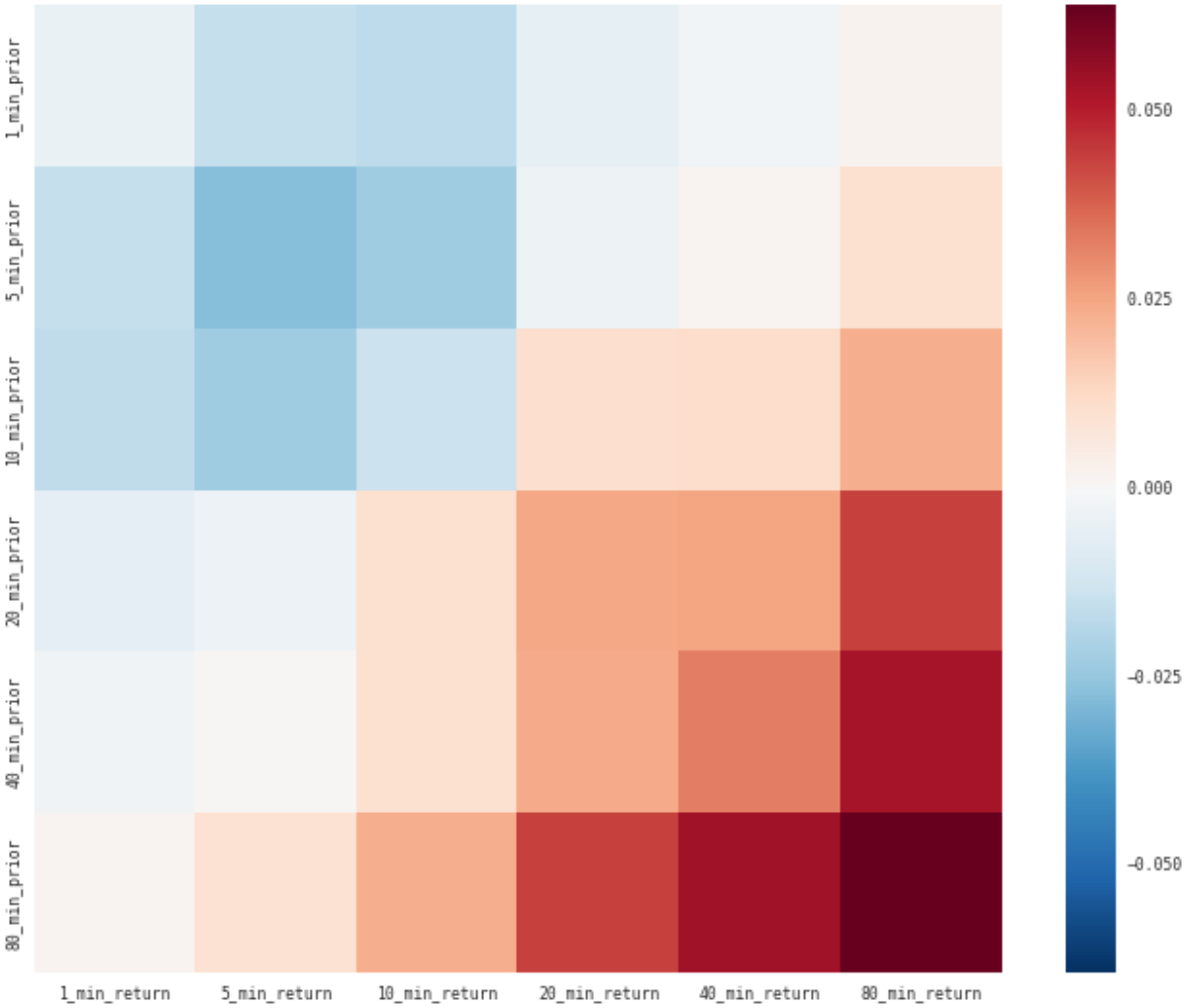

Figure 29 - heatmap of correlations between prior and ahead returns

Figure 30 - p-value of correlations between x-minute prior and x-minute returns

Next we look at volume. Figure 31 shows a heatmap of the correlations between our various volume measures. All of these are positively correlated, which makes sense – they are all measuring whether volume is high or low. Within each measure, volumes at timescales that are closer together are more highly correlated (e.g., 40-minute and 80-minute volumes are more correlated than 1-minure and 80-minute volumes). The time-adjusted log-volume measures are least correlated to the raw volume measures.

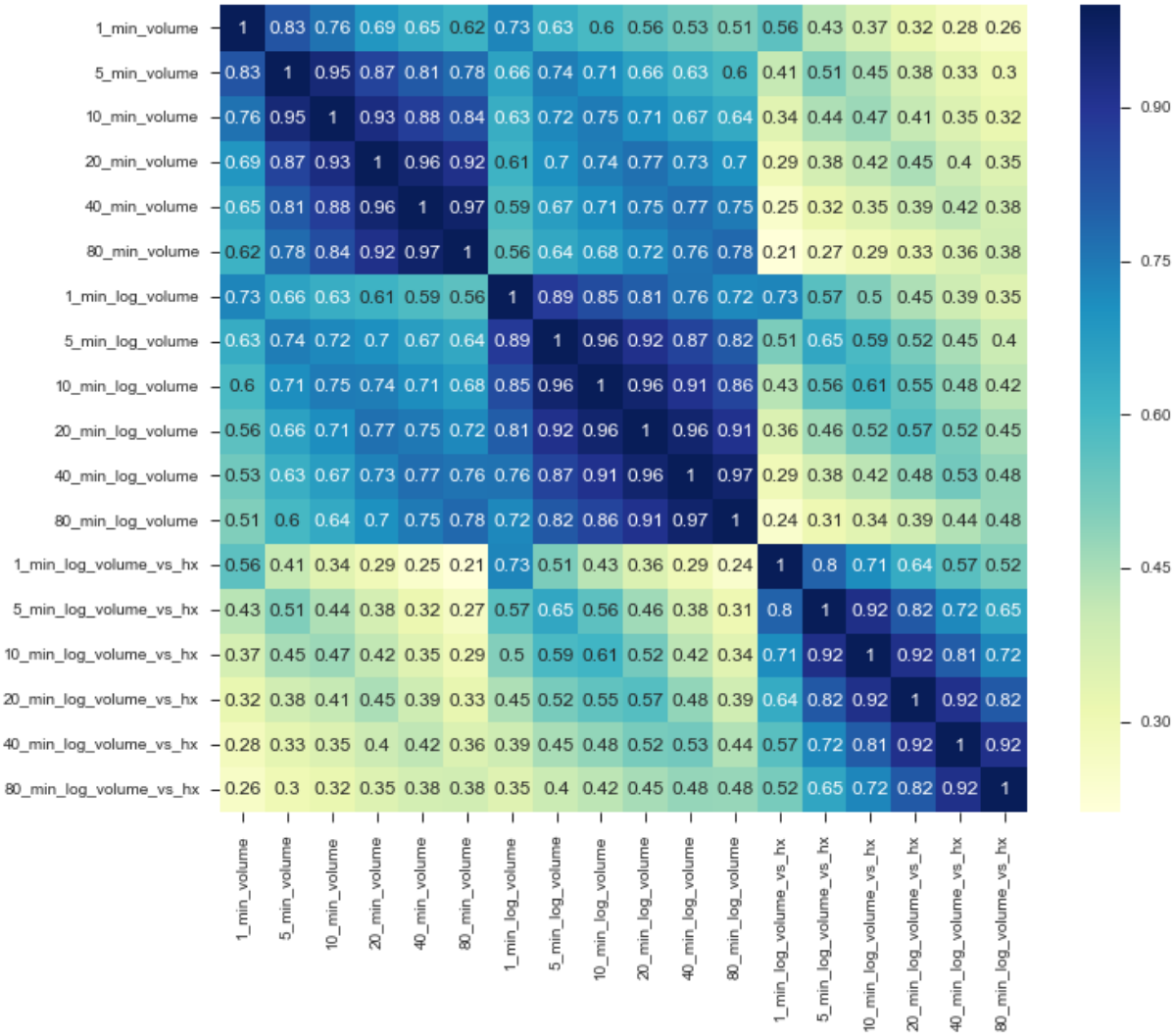

Figure 31 - Correlation heatmap for various volume measures

There isn't any reason to expect that volume should directly correlate with returns. Figure 32 demonstrates this quite clearly.

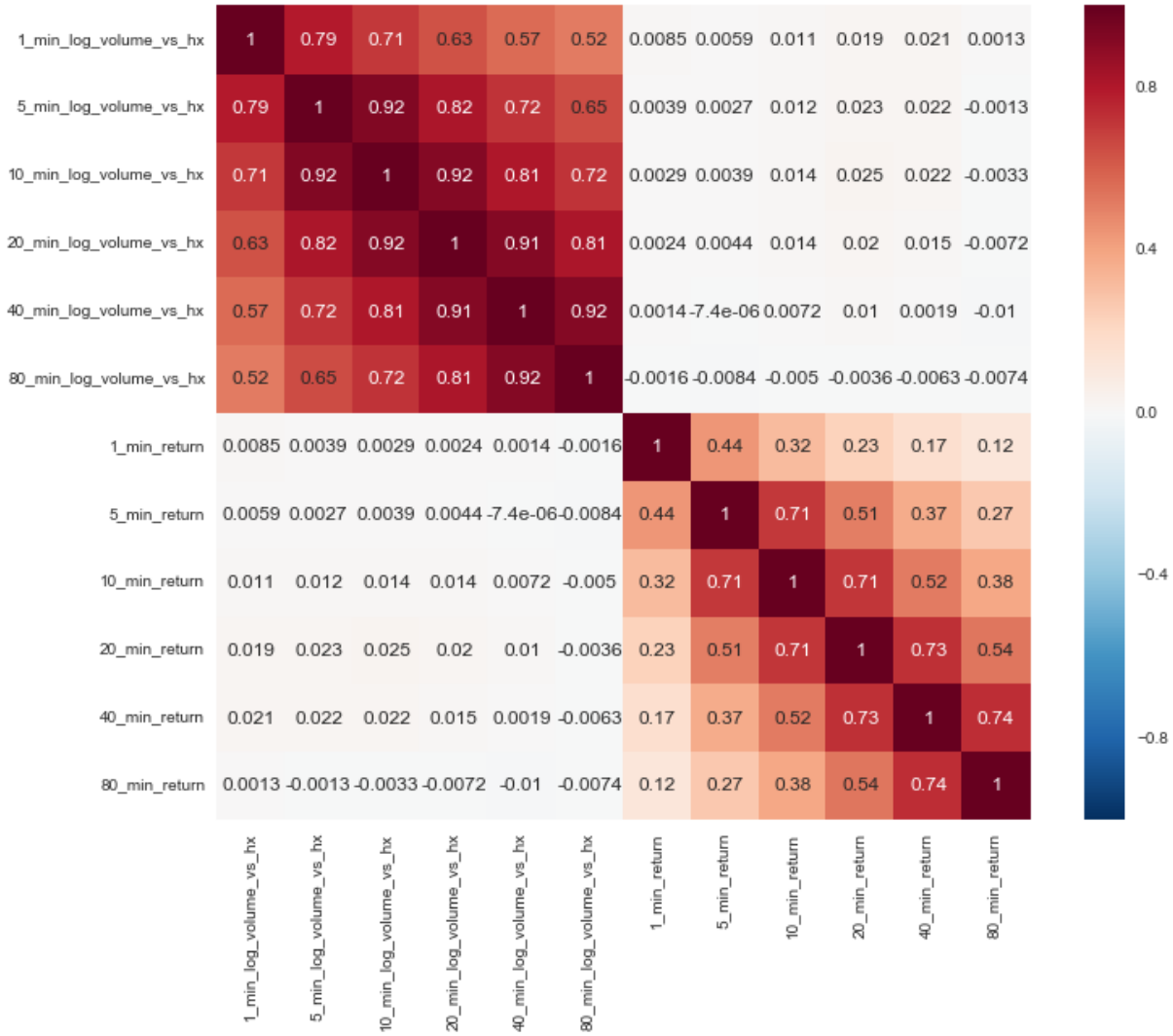

Figure 32 - time-adjusted log-volumes vs x-minute returns

We'll look at the interaction between volume, prior moves, and returns. Figure 33 suggests high time-adjusted 1-minute volume and negative prior 1-minute leads to positive 1-minute returns; Figure 34 suggests high time-adjusted 80-minute volume and positive prior 80-minute leads to positive 80-minute returns. Interestingly, Figure 34 may also be suggesting that a big 80-minute prior move on low time-adjusted 80-minute log-volume may lead to a reversal.

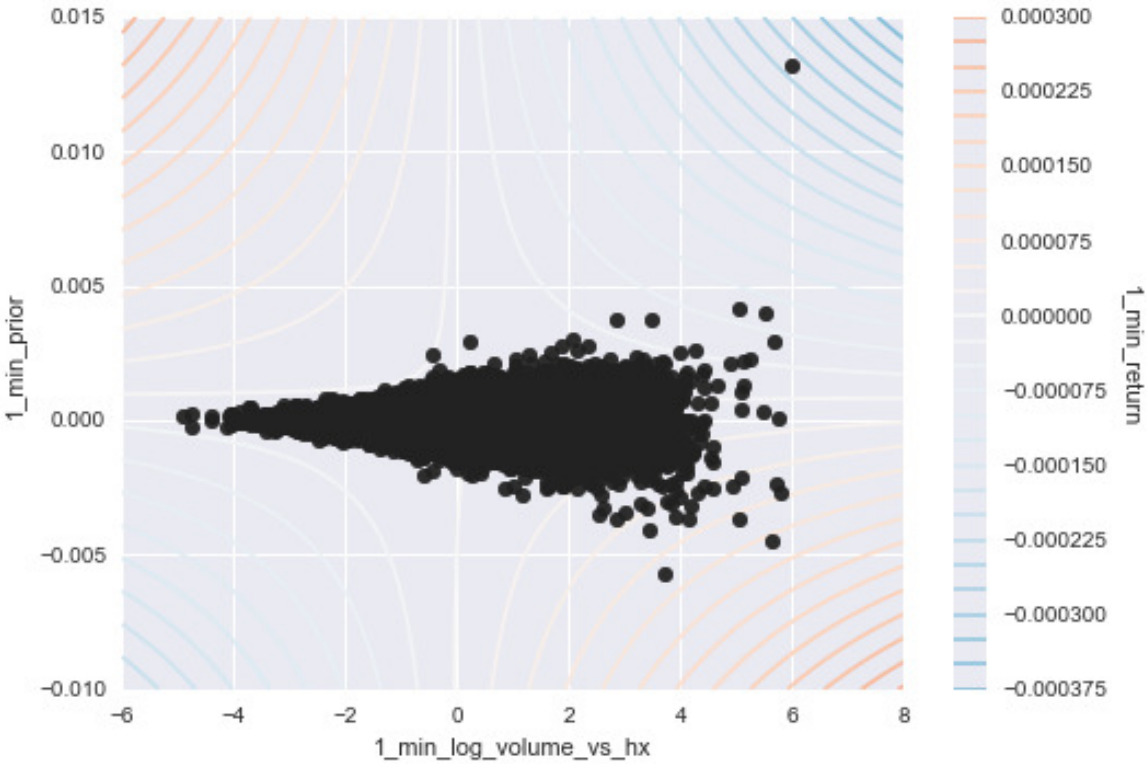

Figure 33 - volume and prior interaction on 1 minute returns

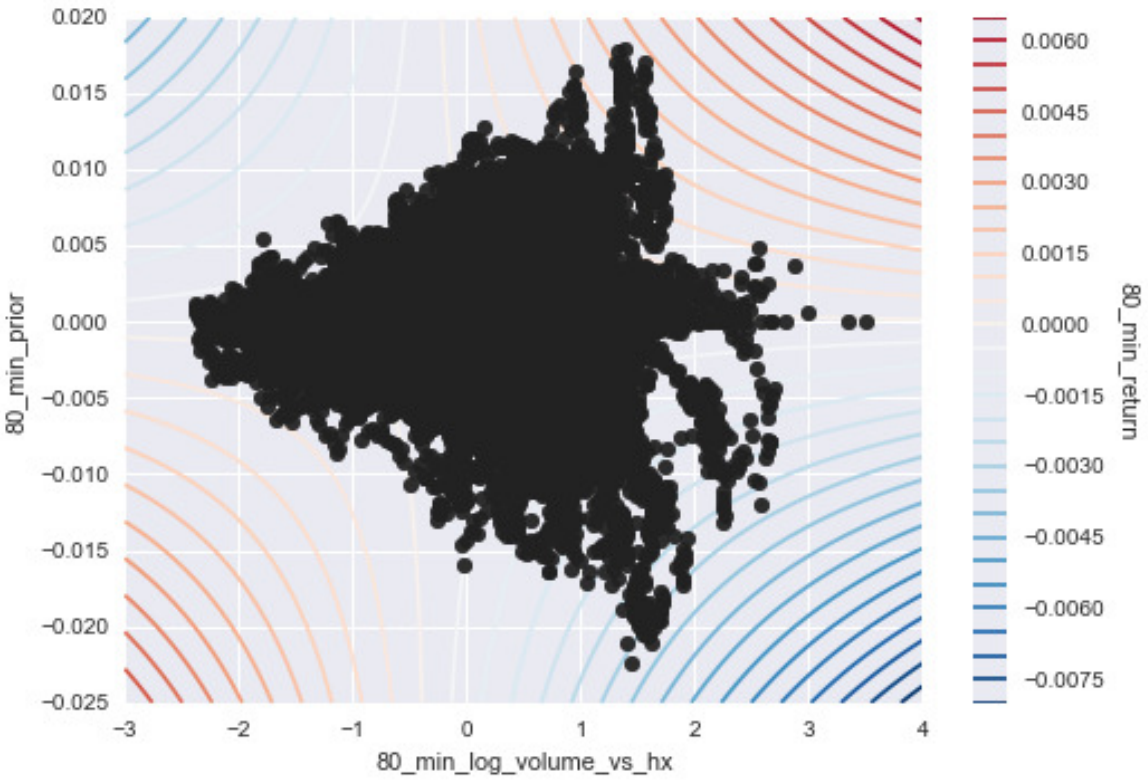

Figure 34 - volume and prior interaction on 80 minute returns

Now we look at the buying up / selling down indicators. Figure 35 shows that these are all positively correlated with each other, which is reassuring as they are all measuring a similar idea. Longer term indicators are more correlated with each other than shorter term indicators. The various styles of indicators can be relatively different from each other (though they are positively correlated).

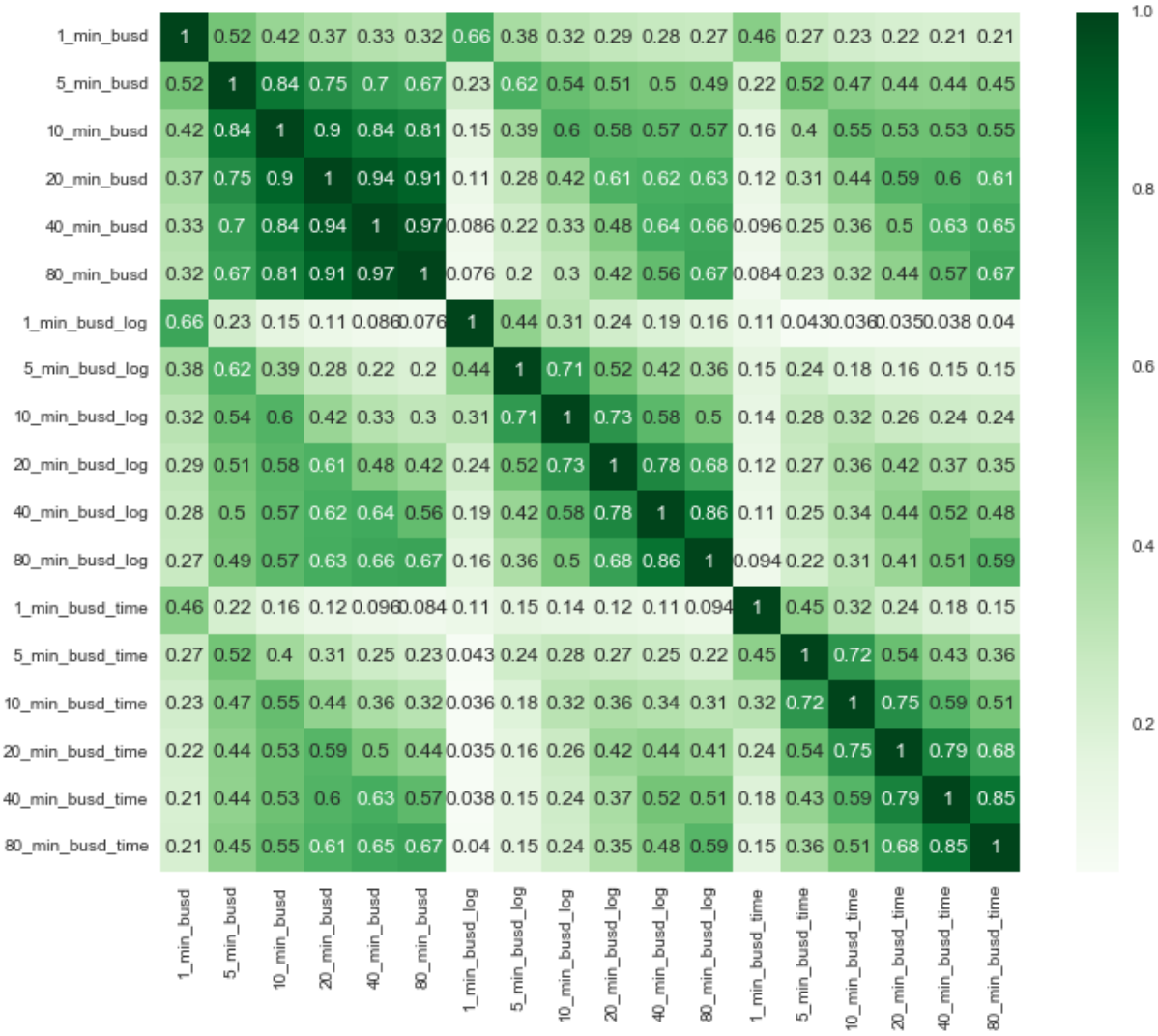

Figure 35 - heatmap of various busd features

Figure 36 shows the heatmaps of correlations between the buying up / selling down features and the x-minute return responses. There is perhaps some relationship at the 80-minute busd variables and the 80-minute returns; these are positively correlated. There may be other relationships too, but this is not entirely clear.

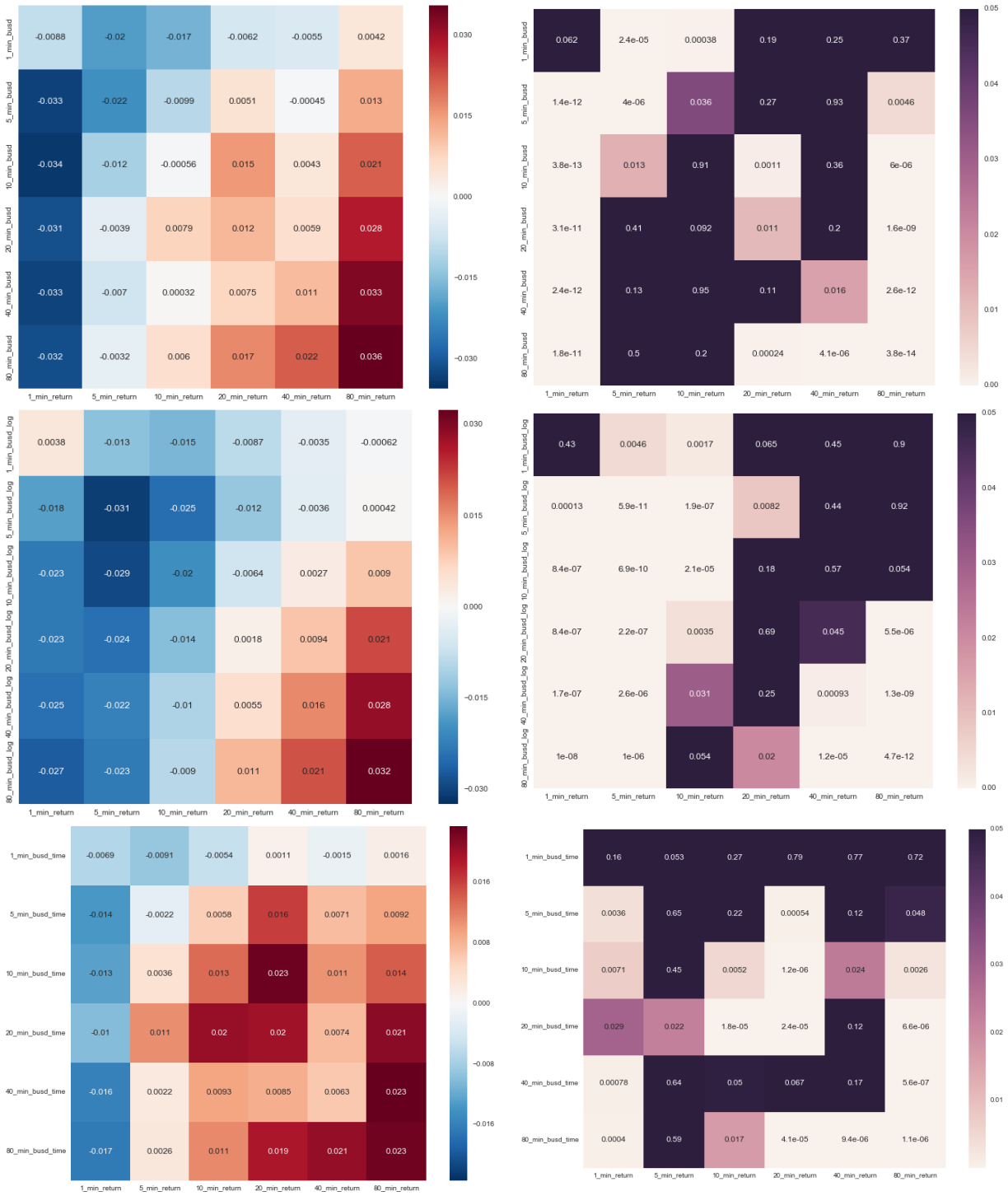

Figure 36 - busd features vs x-minute return responses

Finally, we look at the price vs. high-low range, vs returns. Figure 37 and Figure 38 show there is a positive correlation between longer term price vs range values and longer term returns. Figure 39 and Figure 40 try to visualize this relationship, but the magnitude of the association is so small that it is not

obvious. (Figure 40 shows the distribution of returns when price is above the range, in the upper half of the range, in the lower half of the range, and below the range of the last 80-min high-low respectively.)

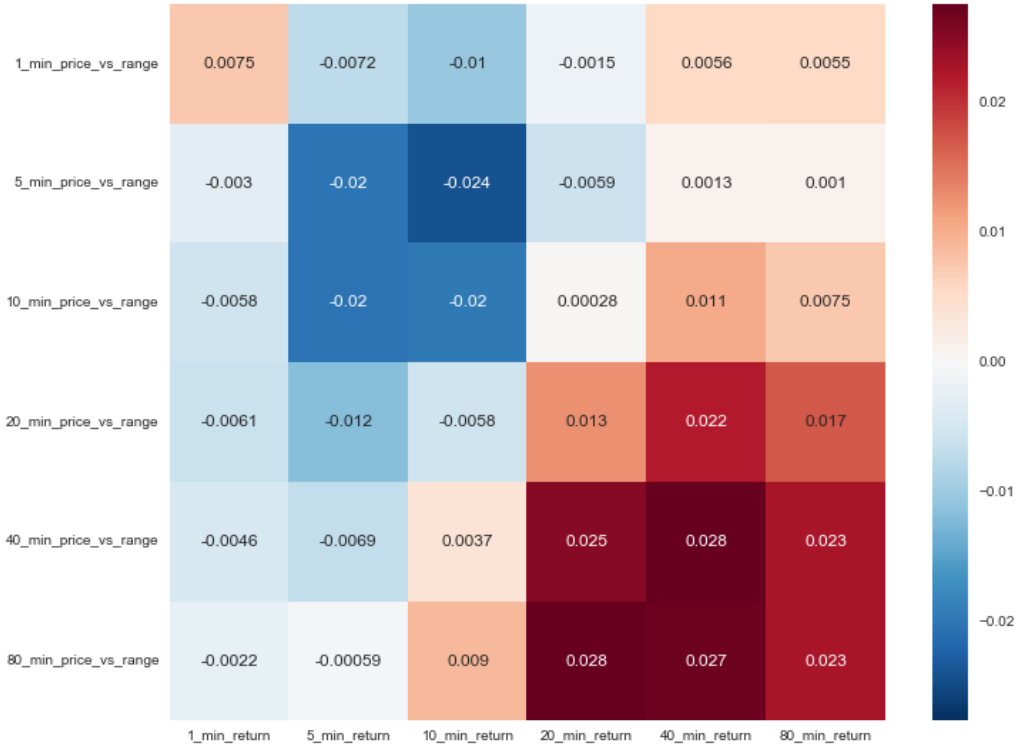

Figure 37 - price vs range vs x-minute return correlation heatmap

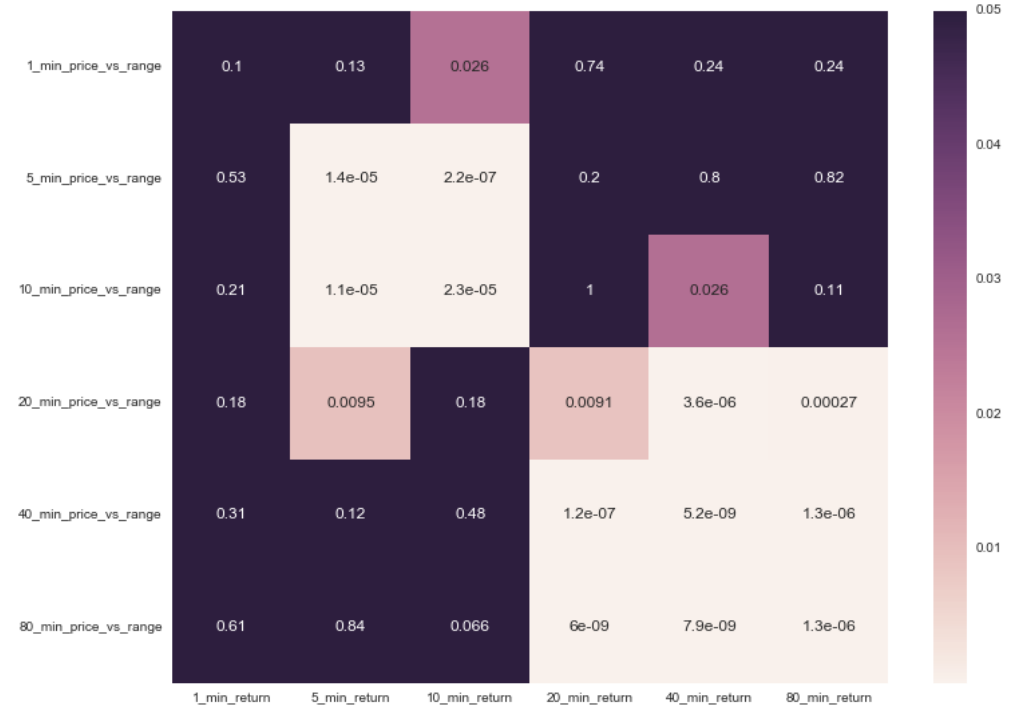

Figure 38 - price vs range vs x-minute return correlation p-value heatmap

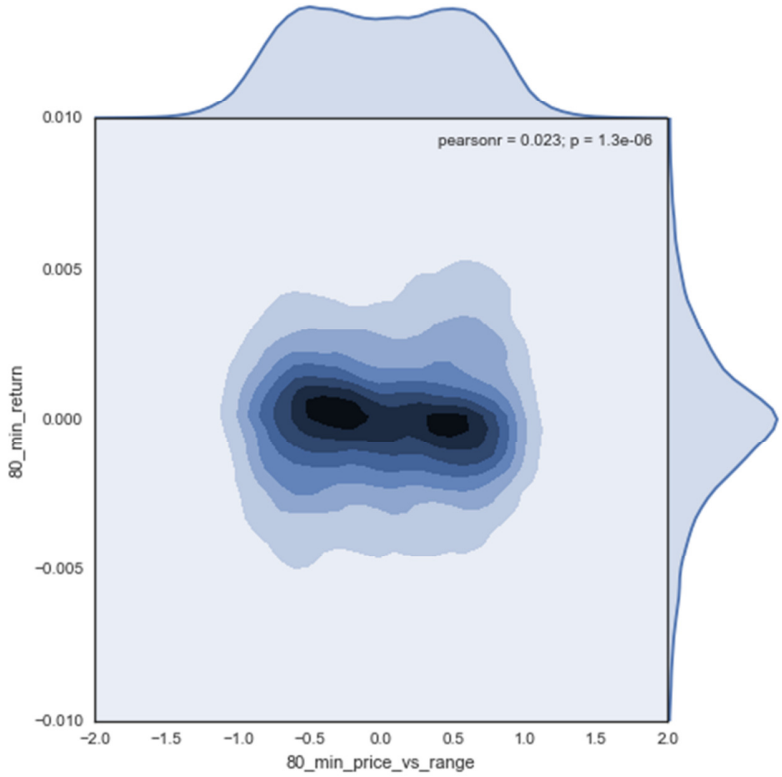

Figure 39 - 80-minute price vs high-low range vs 80-minute return kde plot

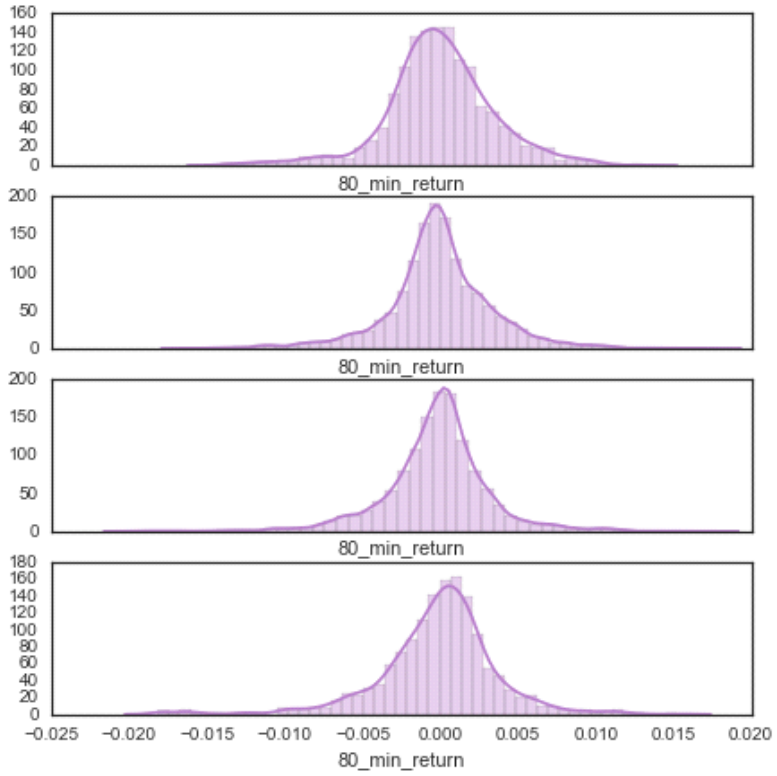

Figure 40 - 80-minute return distribution striated by price vs 80-minute high-low range values

Cross validation strategy

As we are working with time series data, there is a good case to be made that the ordering in the data is important. In particular, we want to avoid 'look ahead' bias. This is because when we do use the model for real world trading, it can only possibly know about things that have happened in the past, and will be called upon to make predictions about the future. Thus, it may not make sense to have the model train on data that is chronologically later, and make predictions about chronologically earlier data.

Thus, our n-fold cross validation strategy is to divide the data into n+1 chronologically contiguous folds. In the first case we have the earliest fold be the training data, and the next fold the test data. In the second case we have the earliest 2 folds be training data, and the third fold the test data. In the last case we have the earliest n folds be the training data, and the last fold the test data. Thus, at each test we do not look ahead, each test fold is the same size, and we eventually utilize all of the data. This is illustrated in Figure 41.

Train	Test	Not used	Not used	Not used	Not used
Train	Train	Test	Not used	Not used	Not used
		…			
Train	Train	Train	Train	Train	Test

Figure 41 - Cross-validation strategy

Dimensionality reduction

We use PCA to attempt a dimensionality reduction on the data. Starting with 43 feature columns (near open, near close, Prior return (x6), last vs MA (x5), log volume vs hx (x6), busd log (x6), busd time (x6), high-low (x6), and price vs range (x6)), we try PCA on 3 folds of the data just to see how stable the results are. This is not a proper cross-validation, in that we are not looking to see how the PCA 'performs' on a test set; rather we just want to check that the PCA results are reasonably consistent on each fold.

Figure 42 shows the elbow plots of explained variance by number of principal components. These look similar across all three folds. Figure 43 attempts to visualize the first 10 eigenvectors for each fold (the first few eigenvectors do indeed seem similar across folds, though the definition of positive and negative values is sometimes different across folds). Overall, it's not entirely clear where a 'sharp' cutoff encapsulating most of the variance would be. Looking at the total variance explained – we could pick the first 8 eigenvectors and thus retain 80% of the variance, or the first 14 eigenvectors for 90% of the variance. Both of these represent a substantial reduction in the dimensionality of the problem

Figure 44 shows the elbow plot for the PCA on the entire dataset. Figure 45 shows the eigenvectors for the PCA on the entire dataset. These are both consistent with the individual folds we tested earlier. We save the PCA transformed variables as pca_v0, pca_v1, … , pca_v42.

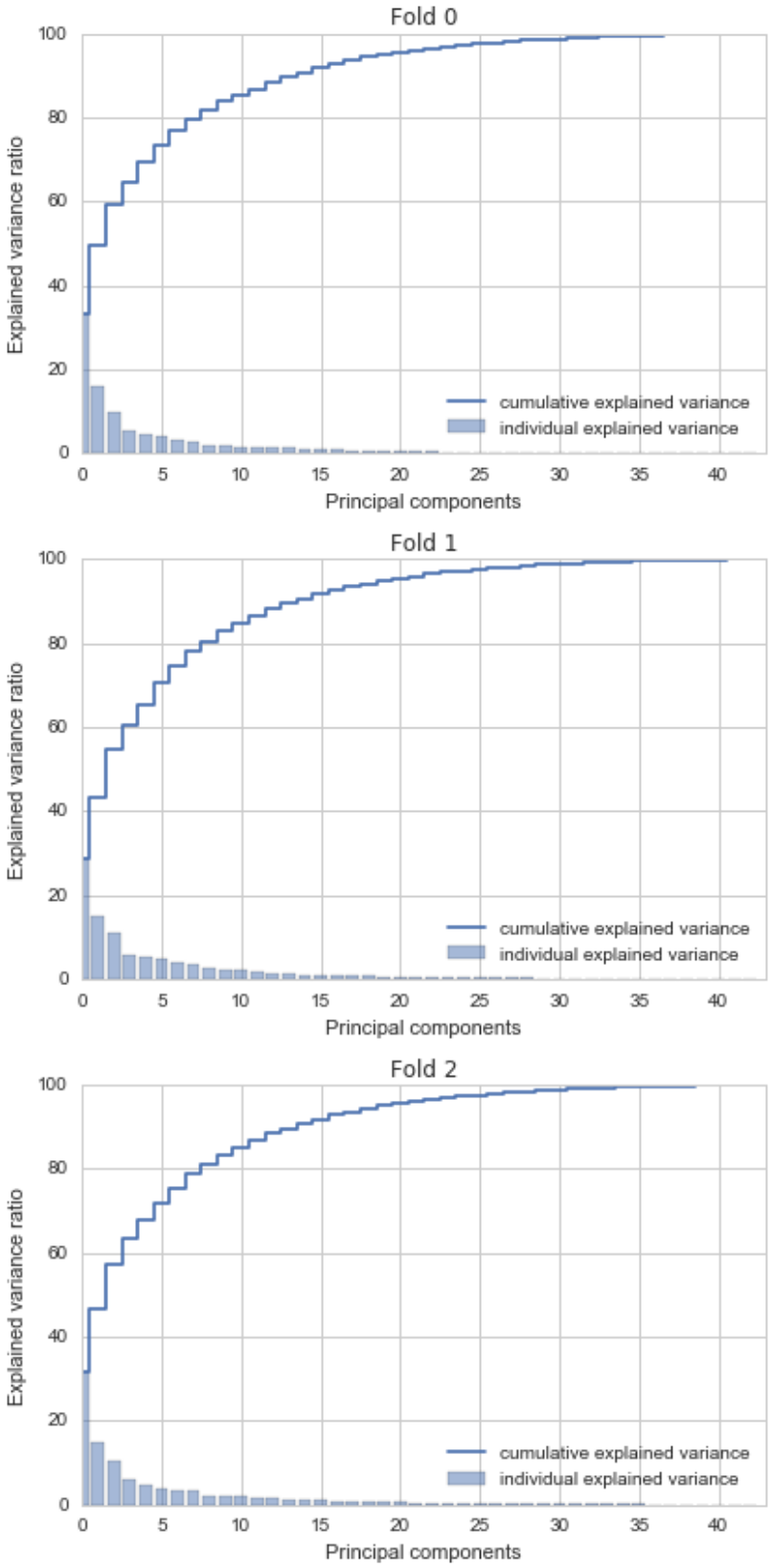

Figure 42 - PCA elbow plots

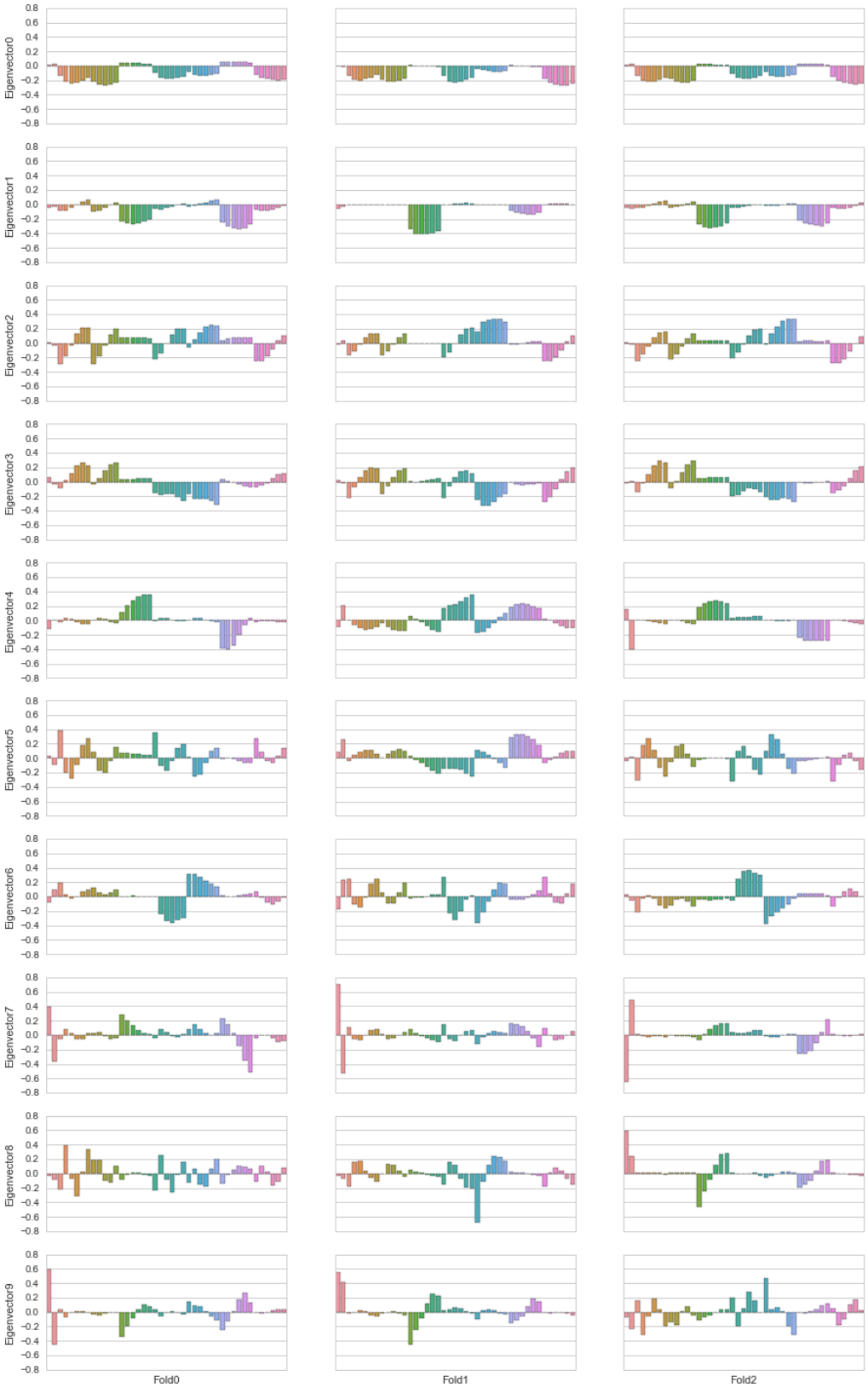

Figure 43 - PCA eigenvectors by fold

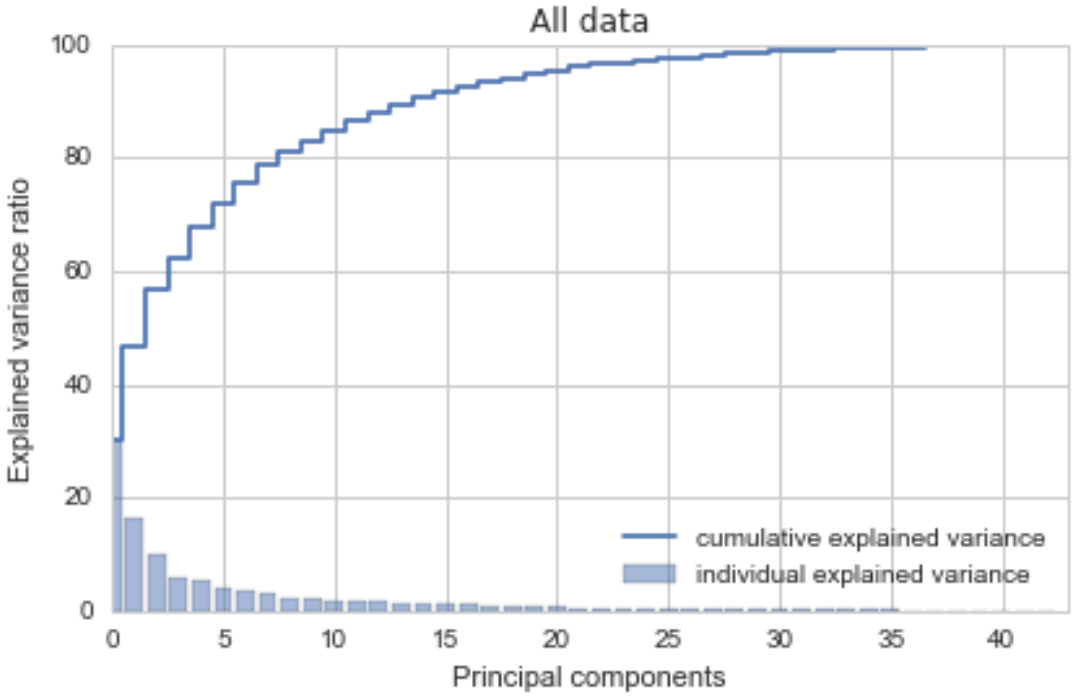

Figure 44 - Elbow plot for PCA on entire dataset

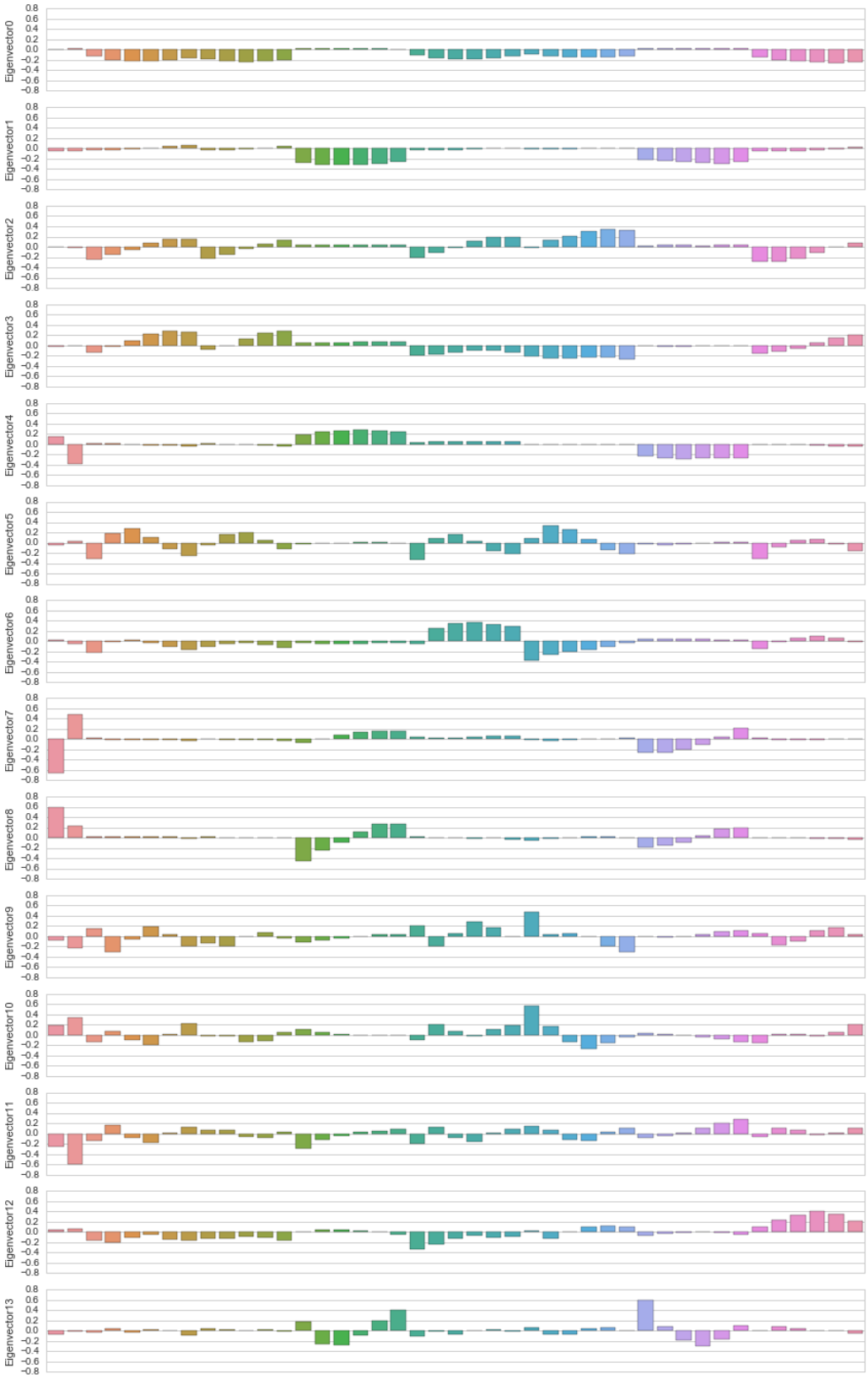

Figure 45 - Eigenvectors for PCA on entire dataset

Overview of general fitting approach

We apply regression methods to fit the continuous x-minute return responses; we use classification methods to fit the binary and categorical x-minute up/down, 'cat', and 'high cat' responses. For each method, we use cross validation as described in 'Cross-Validation Strategy'. For all models except the support vector models, we use 6 folds. For support vector models we use 4 folds (this is purely to reduce the time taken for fitting). Given the dimensionality reduction results, for each method we try a fit with the first 4, 8, or 15 components. For several methods we also explore using all 43 components – this is in cases where we expect the models to inherently be able to use such a large number of features. Further, we perform a grid search to tune the hyperparameters of the models (from the sklearn.grid_search. GridSearchCV package). The range of models and hyperparameters tuned is summarized in Table 9 and Table 10. Also, in each case the model performance is assessed once by the default settings (typically 'accuracy' for classification and mean squared error for regression) and once by our own custom trade-z-score method.

The trade-z-score method converts predicted response values in trades. So, in a regression, positive predictions are buys and negative predictions are sells; in binary classification a 1 is a buy and a 0 a sell; in the three-category classification, 0 is a sell, 1 is 'do nothing', and 2 is a buy. The trade-z-score is then the ratio of the mean return to the standard deviation of the return for these trades. For example, in the x-minute up/down case, if our prediction is a 1, and the x-minute return is -0.05, then the return in this case is -0.05. But if our prediction is a -1 and the x-minute return is -0.03, then the return is +0.03. We look at all these returns (when non-zero), and compute the mean over the standard deviation as the trade-z-score. This is similar to a Sharpe ratio, and indicates the stability of the predictions as a trading system; higher scores are better.

Finally, we try to pick the best fit from each model for each response period, based on the 2 scoring methodologies (default and trade-z).

Model	Sklearn package	Tuned hyperparameters
Ordinary Least Squares Regression	sklearn.linear_model. LinearRegression	'normalize': [True, False]
Ridge Regression	sklearn.linear_model. Ridge	'normalize': [True, False] 'alpha': [0.1,1.0,10.0]
K Nearest Neighbors Regression	sklearn.neighbors. KNeighborsRegressor	'weights': ['uniform', 'distance'], 'n_neighbors': [5, 25, 100, 300] 'p': [1,2]
Epsilon-Support Vector Regression	sklearn.svm. SVR	'C': [1e-4,1e-2,1] 'epsilon': [0.25,1.0]*mean(abs(y))
Decision Tree Regression	sklearn.tree. DecisionTreeRegressor	'max_depth': [1, 2, 3, 4, 5, 7, 10, 14, 20, 28, 38, 52, 70, 98]
Random Forest Regression	sklearn.ensemble. RandomForestRegressor	'n_estimators': [10,30,100] 'max_depth': [1, 2, 4, 8, 16, 32, 64]
Extra-trees Regression	sklearn.ensemble. ExtraTreesRegressor	'n_estimators': [10,30,100],'max_depth': [1, 2, 4, 8, 16, 32, 64]
Bagging Regression (base estimator: Decision Tree	sklearn.ensemble. BaggingRegressor	'n_estimators': [50,150,300]

Regression)		
AdaBoost Regression (base estimator: Decision Tree Regression)	sklearn.ensemble. AdaBoostRegressor	'n_estimators': [50,100,200,400] Decision Tree Regression 'max_depth': [1,2,4]
AdaBoost Regression (base estimator: Ordinary Least Squares Regression)	sklearn.ensemble. AdaBoostRegressor	'n_estimators': [50,100,200,400] Ordinary Least Squares Regression 'normalize': [True, False]

Table 9 - Summary of regression models

Model	Sklearn package	Tuned hyperparameters
Logistic Regression Classification	sklearn.linear_model. LogisticRegression	'C': [1e-4, 1e-3, 1e-2, 1e-1, 1, 1e1, 1e2, 1e3, 1e4]
Gaussian Naïve Bayes Classifier	sklearn.naive_bayes. GaussianNB	Nil
K Nearest Neighbors Classification	sklearn.neighbors. KNeighborsClassifier	'weights': ['uniform', 'distance'] 'n_neighbors': [5, 25, 100, 300] 'p': [1,2]
C-Support Vector Classification	sklearn.svm. SVC	Binary responses 'C': [1e-4,1e-2,1] Categorical responses 'C': [1e-4,1e-2,1] 'class_weight': [None, 'balanced', {0: 100, 2: 100}]
Decision Tree Classification	sklearn.tree. DecisionTreeClassifier	'criterion': ['gini','entropy'] 'max_depth': [1, 2, 3, 4, 5, 7, 10, 14, 20, 28, 38, 52, 70, 98]
Random Forest Classification	sklearn.ensemble. RandomForestClassifier	'n_estimators':[10,30,100] 'max_depth': [1, 2, 4, 8, 16, 32, 64]
Extra-trees Classification	sklearn.ensemble. ExtraTreesClassifier	'n_estimators':[10,30,100] 'max_depth': [1, 2, 4, 8, 16, 32, 64]
Bagging Classification (base estimator: Decision Tree Classification)	sklearn.ensemble. BaggingClassifier	'n_estimators': [50,150,300]
AdaBoost Classification (base estimator: Decision Tree Classification)	sklearn.ensemble. AdaBoostClassifier	'n_estimators': [50,100,200,400] Decision Tree Classification 'max_depth': [1,2,4]
Voting Classification (LogisticRegression, GaussianNB, KNeighborsClassifier, SVC, RandomForestClassifier)	sklearn.ensemble. VotingClassifier	'voting': ['hard,'soft']

Table 10 - Summary of classification models

Regression models

Ordinary Least Squares Regression

Besides using different numbers of principal components, the only other parameter searched over for the Ordinary Least Squares Regression is whether or not to normalize the regressors Z before regression.

Ridge Regression

Ridge regression helps to get around multicollinearity issues that might impact Ordinary Least Squares regression. (In a sense we are unlikely to have these issues as our data are PCA reduced and therefore orthogonal, however it's interesting to see if there is any difference in performance of the models fitted via this methodology.) We also try up to 43 features for this estimator, to see if perhaps the different regularization methodology allows it to more successfully use more features. (Again, this would be more interesting on the raw pre-PCA data; nonetheless let's see if it makes a difference.)

Besides the number of features, we also vary the alpha parameter – this controls the amount of shrinkage; so a higher alpha value makes the fitted coefficients more robust to collinearity. Looking at the results, does not appear that ridge regression produces anything vastly different from Ordinary Least Squares in terms of the scoring metrics we looked at. There are some marginally higher trade Z scores in the longer time periods, but these differences might not be significant.

K Nearest Neighbors Regression

We try out regression using k-nearest neighbors. In this case, in addition to varying the number of features, we vary whether the weights are uniform or vary by distance, the number of nearest neighbors used, and the power parameter of the Minkowski metric – i.e., since we used p=1 or 2, whether distance is calculated based on manhattan distances or Euclidean distances. The trade Z scores at the longer intervals seem slightly better than those for the ridge regression, but again these differences may not be significant.

Epsilon-Support Vector Regression

Here we vary the C penalty parameter of the error term and the epsilon-tube width within which no penalty is assigned for prediction deviations from actual values. In order to get this to work, we had to scale the epsilon values as a percentage of the absolute magnitude of our response variables. (The default is 0.1 but our returns are much smaller than that, being percentage returns over very small time frames.)

Decision Tree Regression

We vary the max depth, and let the remaining parameters be at the default settings. On the max depth, we experiment with small values first but step up quickly to high settings. After the fact it would appear that some of the values used permitted too much overfitting, and that since the cross-validation performance is fairly random, this perhaps failed to weed out overfit models.

Random Forest Regression

This fits many decision trees on bootstrap samples of the data and averages the result. We varied the number of trees and the maximum depth of each.

Extra-trees Regression

This fits some randomized decision trees on various sub-samples of the data and averages the result. Again we varied the number of trees and the maximum depth of each.

Bagging Regression

This is another way to fit many base estimators and average them. We used the default base estimator of decision trees and just tuned the number of estimators.

AdaBoost Regression

This attempts to weight more heavily observations on which the prediction performs poorly and refits. We run it with a Decision Tree base estimator, in which we vary the max depth. We also run it with an Ordinary Least Squares base estimator, and vary whether to normalize the data or not. In both cases we tune the number of estimators.

Classification Models

Logistic Regression Classification

We perform a logistic regression, varying the number of features as well as the C parameter. This C parameter controls the regularization strength; the smaller the C value the stronger the regularization. i.e., smaller C values constrain the model more (smaller C – less likely to tweak coefficients to adjust for small perturbations in the data).

Gaussian Naïve Bayes Classification

In this case there were no hyperparameters to tune.

K Nearest Neighbors Classification

Classifying by looking at nearest neighbors. We tune what kind of distance measure to use: manhattan or Euclidean. We tune the weighting of points – uniform or by distance. We also tune the number of neighbors.

C-Support Vector Classification

We tune the C penalty parameter. In the case of categorical responses, we also try class weights, so that the penalty may be applied differently for different classes. E.g., we tried one setting where penalty for mis-classifying a ‘0’ or ‘2’ was 100x the penalty for mis-classifying a ‘1’.

Decision Tree Classification

Here we tune the criterion (gini or entropy) and the max depth

Random Forest Classification, Extra-trees Classification, Bagging classification, AdaBoost Classification

As in the regression case, these are ensemble methods averaging over multiple trees. (In the case of AdaBoost we chose to use the default decision trees).

Voting Classification

This is an interesting ensemble method. Here we combined Logistic Regression, Gaussian Naïve Bayes, K Neighbors Classification, C-Support Vector Classification, and Random Forest Classification. We try both hard voting (just take the modal prediction) and soft voting (look at the average of the predicted probability of class membership over all the methods.

Results

Table 11 shows the 'best' estimator for each scoring method and horizon. Table 12 shows the cross validation scores. These are not stellar. Decision tree and ensemble methods based on decision trees are popular, as are support vector machine methods. It seems like decision tree methods are more popular on trade z criteria than on the default scoring criteria.

Trade Z	**1 min**	**5 min**	**10 min**	**20 min**	**40 min**	**80 min**
Regression	Decision Tree	Decision Tree	Decision Tree	AdaBoost (Decision Tree)	Random Forest	Random Forest
Up/Down	Random Forest	Random Forest	SVC	SVC	Naïve Bayes	Decision Tree
Categorical	SVC	Decision Tree	SVC	SVC	SVC	Decision Tree
High Categorical	Random Forest	Random Forest	AdaBoost (Decision Tree)	Decision Tree	Random Forest	Random Forest
Default scoring	**1 min**	**5 min**	**10 min**	**20 min**	**40 min**	**80 min**
Regression	Decision Tree	Extra-trees	Random Forest	Ridge	Ridge	Extra-trees
Up/Down	Voting (soft)	Random Forest	Logistic	SVC	SVC	Decision Tree
Categorical	Logistic	Logistic	Logistic	Logistic	Logistic	Voting (hard)
High Categorical	SVC	SVC	SVC	SVC	SVC	SVC

Table 11 - Best estimator by score and horizon

Trade Z	**1 min**	**5 min**	**10 min**	**20 min**	**40 min**	**80 min**
Regression	0.163	0.040	0.101	0.037	0.071	0.121
Up/Down	0.021	0.023	0.025	0.021	0.040	0.042
Categorical	0.029	0.027	0.034	0.033	0.039	0.075
High Categorical	0.178	0.193	0.176	0.291	0.338	0.717
Default scoring	**1 min**	**5 min**	**10 min**	**20 min**	**40 min**	**80 min**
Regression	9.08×10^{-4}	-1.13×10^{-4}	-8.16×10^{-4}	-2.52×10^{-3}	-4.88×10^{-3}	-5.73×10^{-3}
Up/Down	52.34%	53.08%	52.71%	52.19%	52.23%	52.06%
Categorical	40.60%	40.30%	39.87%	38.85%	38.56%	38.97%
High Categorical	90.84%	90.96%	90.73%	90.97%	91.04%	90.94%

Table 12 - Best estimator cross-validation performance

Figure 46 shows the equity curve / 'net asset value (NAV)' as a way of assessing performance of the predictions. We start at 1, then add the performance of each trade. i.e., if the model predicts an

upmove, we add the return; if it predicts a downmove, we add the negative of the return. For regression models we consider a buy trade if the prediction is positive and a sell trade if it's negative. For categorical variables a '0' is a sell and a '2' is a buy. The results shown are obtained based on fitting the best cross-validated predictor on the entire dataset. Note that some trade z scores are low despite the equity curves looking quite straight, because the scale of the graph masks the standard deviation. But indeed the z scores for the trade_z_updown and trade_z_high_cat are quite high. Exact numbers are in the appendix.

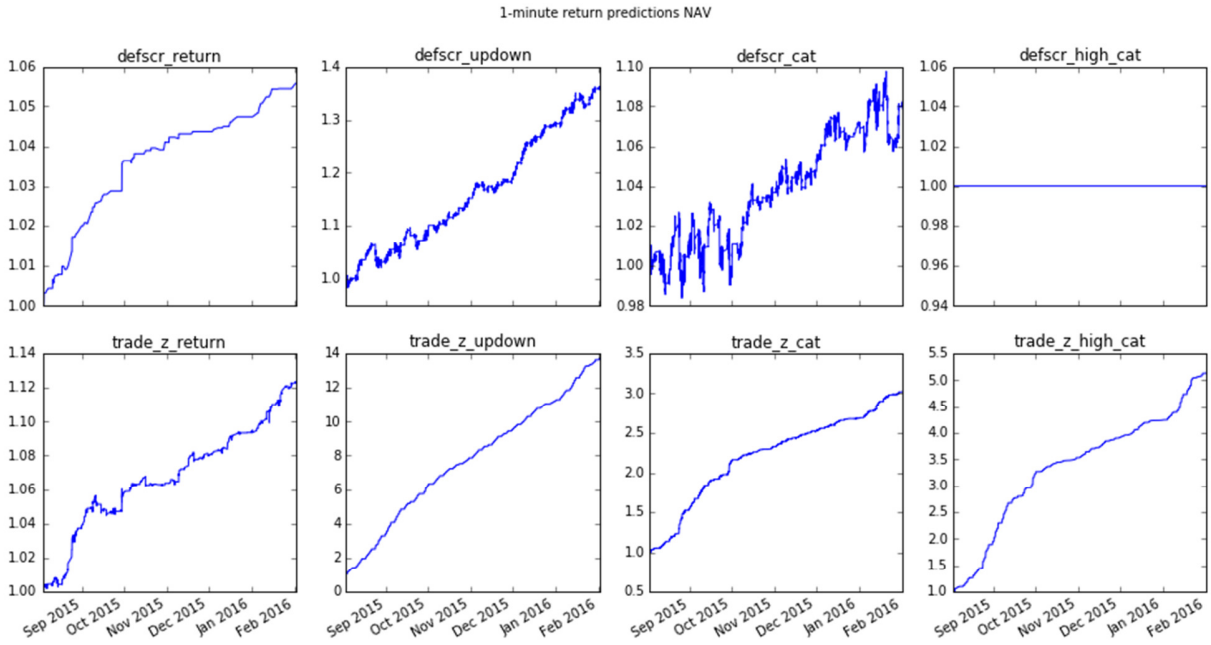

Figure 46 - NAV for 1-minute return predictions

To get a better sense of how the models are actually performing, we use truly out-of-sample data, from Feb 2 to March 29. The models have not seen any of this data. Figure 47 shows the additional data obtained. Note that the Indian budget was released on Feb 29, and the volatility of that trading day is visible on the chart. Figure 48 shows performance of the models on the true-out-of-sample, which is notably worse. Table 13 gives details of the in-sample vs out-of-sample performance. There is a degradation between in-sample and out-of-sample; but, the performance in the out-of-sample is very much in line with the scores obtained in the cross-validation. Additional return horizons exhibit the same phenomenon, these are shown in the appendix.

Figure 47 - true out-of-sample data

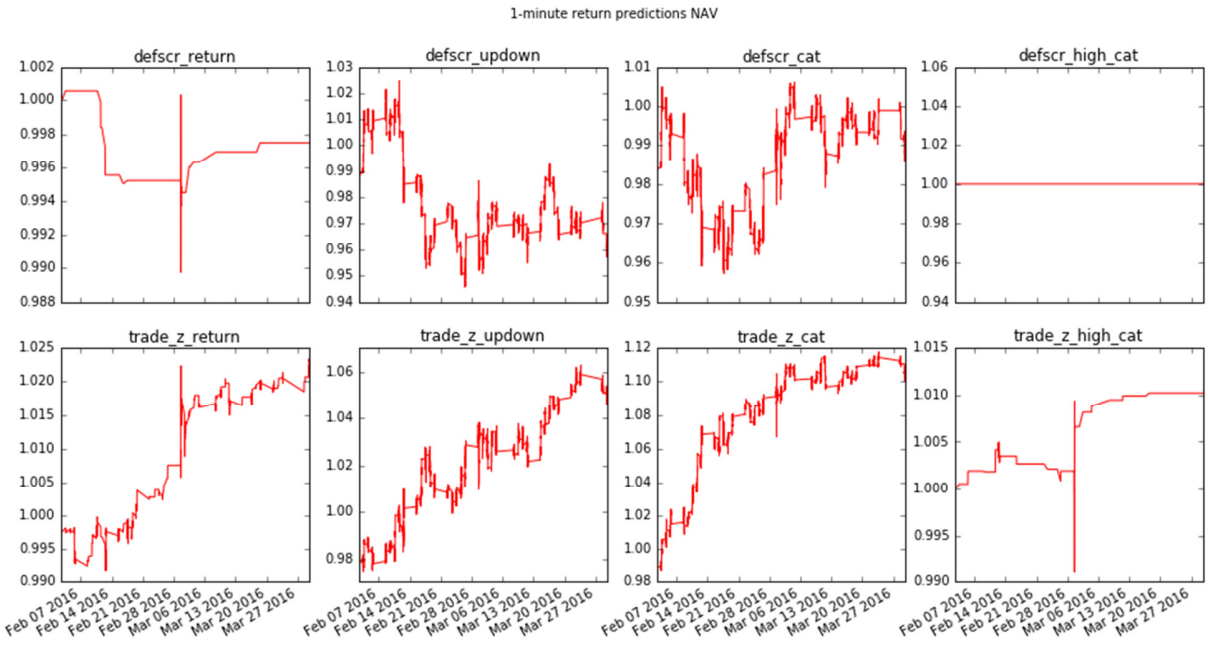

Figure 48 - out-of-sample performance of 1-minute return predictions

buys: 57 | sells: 4 %up: 85.25% mean: 0.000913 std: 0.000978 trade_z: 0.934	buys: 20387 | sells: 22732 %up: 53.46% mean: 8.37e-06 std: 0.000417 trade_z: 0.0201	buys: 12084 | sells: 11670 %up: 52.85% mean: 3.49e-06 std: 0.000503 trade_z: 0.00694	buys: 0 | sells: 0 %up: 0.00% mean: nan std: nan trade_z: nan raw_accuracy: 89.67%

	raw_accuracy: 53.46%	raw_accuracy: 40.99%	
buys: 47 | sells: 1042 %up: 60.61% mean: 0.000113 std: 0.000671 trade_z: 0.169	buys: 21568 | sells: 21551 %up: 100.00% mean: 0.000294 std: 0.000296 trade_z: 0.994 raw_accuracy: 100.00%	buys: 11507 | sells: 12170 %up: 59.89% mean: 8.56e-05 std: 0.000499 trade_z: 0.172 raw_accuracy: 46.25%	buys: 2146 | sells: 2263 %up: 100.00% mean: 0.000939 std: 0.000413 trade_z: 2.27 raw_accuracy: 99.90%
buys: 22 | sells: 1 %up: 39.13% mean: -0.000110 std: 0.00259 trade_z: -0.0423	buys: 7101 | sells: 6446 %up: 51.69% mean: -2.87e-06 std: 0.000520 trade_z: -0.00552 raw_accuracy: 51.69%	buys: 4490 | sells: 4421 %up: 52.31% mean: -8.39e-07 std: 0.000594 trade_z: -0.00141 raw_accuracy: 39.37%	buys: 0 | sells: 0 %up: 0.00% mean: nan std: nan trade_z: nan raw_accuracy: 89.71%
buys: 30 | sells: 444 %up: 55.91% mean: 4.89e-05 std: 0.00104 trade_z: 0.0469	buys: 6962 | sells: 6585 %up: 51.69% mean: 3.47e-06 std: 0.000520 trade_z: 0.006678 raw_accuracy: 51.69%	buys: 5076 | sells: 5191 %up: 52.49% mean: 9.89e-06 std: 0.000567 trade_z: 0.0174 raw_accuracy: 38.26%	buys: 26 | sells: 53 %up: 53.16% mean: 0.0001293 std: 0.00201 trade_z: 0.0645 raw_accuracy: 89.67%

Table 13 - performance statistics

Conclusion:

Predicting intraday returns is not simple; the methods here perform at best randomly in the true out-of-sample test. It may be that smarter features could be constructed from the data to help this process. As a side note, because the models are performing so randomly, the hyperparameter tuning in cross-validation may not be weeding out the over-fitting models. If we did not look at true out-of-sample data, we might not have realized this, and would have concluded that the in-sample fit would deliver good results.