Develop Cleaning Algorithms from "Quality Money Management" by Andrew Kumiega and Benjamin Van Vliet

CHAPTER 15: Develop Cleaning Algorithms

Good inputs are the key to success in financial modeling, and forecasting, as well as risk management, requires good, clean data for successful testing and simulation. Over the course of backtesting, the use of a large amount of in-sample data will produce a more stable model and reduce the danger of curve-fitting, thereby increasing the probability of out-of-sample success. Virtually no data, however, is perfect and financial engineers spend large amounts of time cleaning errors and resolving issues in data sets, sometimes called preprocessing the data. It is very easy and very common to underestimate the amount of time preprocessing will take. Most financial engineers can recall wasting countless hours spent backtesting only to have drawn bad conclusions because of bad data. Easily half the time required for high quality backtesting can be spent cleaning data.

The problem is that nobody wants to clean. Everyone is too busy to clean, but failing to adequately consider the impact of bad data can lead to bad models or worse: systems that pass the backtest stage but lose millions in actual trading. The quality of data purchased from vendors can range from very clean to very dirty; usually there is a positive correlation between the price and...
quality of data. Using high quality, more expensive data almost always pays off in the long run, though even high quality data will have problems. Whatever the case, time spent finding good data and giving it a good once over is worth the time and effort. However, this is rarely done in the industry.

All data should be cleaned before use. In K/V, the product team preprocesses data up front, so as to have good clean data for backtesting in this stage as well as for risk pro cesses in Stage 4. Serious data cleaning involves more than just visually scanning and bagging data and updating bad records with better data. Rather, it requires that the product team decomposes and reassembles the data. And this takes time.

In this step, the product team investigates alternative data cleaning methods to arrive at benchmark processes, the ones that generate the best performance. Benchmarking data cleaning processes focuses the product team on improving the performance of the trading/investment system. A best practice for one system, though, may not be a best practice for every trading/investment system, because each will have its own unique input data. In the end, cleaning data will remove variation from the process, so the team can get a good clean look at the process, the common variation and the potential of the system.

As with any process, it pays to plan ahead before cleaning data. First, identify and categorize all the types of problems you expect to encounter in your data; then survey the available techniques to address those different types of errors; and finally develop methods to identify and resolve the problems. Data problems fall into one of four categories:

1. Bad, or incorrect, data.
2. Formatting problems.
3. Outliers, which skew results.

It may seem like an oxymoron, but not all data is the same, even if it is the same data. If you run a backtest with historical data from one vendor and then run the same backtest with data from another, you will likely end up with different results and different conclusions. This is because while some databases offer snapshots of data known at a certain time, others restate historical data to enable comparison over time. You see, the issues are complex.

Now, data cleaning (if it is done at all) is usually assigned to a low-level person with, more often than not, few, if any, tools to do the job right. High-level people often believe cleaning data is beneath them, so they assign the task to interns. In manufacturing, however, senior-level engineers are assigned to teams that perform full-scale experiment design in the production environment. These engineers ensure that the inputs meet the quality specifications and that both inputs and outputs of the experiments are properly captured. This task in manufacturing would never be assigned to junior-level engineers, since an error at this stage would be too costly. Trading firms need to have senior-level people design Data Transformation Management Systems, the tools and processes to clean data, just like in manufacturing. These tools should be automated, to the extent possible to allow junior-level financial engineers to investigate errors and outliers graphically, using scatterplots, SPC charts, and histograms, just like in manufacturing where quality personnel use similar tools. This is consistent with quality engineering where most tools are graphical, so management can train nontechnical people quickly to identify problems.

15.1. STEP 2, LOOP 1: Identify Required Cleaning Activities and Algorithms

In the first loop in this step, the product team should identify what data problems may exist and alternative methods to handle them. As with quantitative methods and trading algorithms, data cleaning algorithms should be benchmarked and documented using the research and documentation methods described in K/V 1.2.

All data, both real-time and historical, contains errors and issues, and the nature of the trading/investment system dictates what types of problems the team will likely encounter. For example, data issues for high frequency systems will center more on cleaning data, whereas those for systems with longer-term holding periods will focus more on, say, dividends and releases of and revisions to financial statements.

15.1.1. Bad Data and Reformatting

In all cases cleaning of bad data is a process that consists of first of detection, then classification of the root cause of the error, and then correction of the error—bad quotes, missing data, bad dates, column shifted data, file corruption, differing data formats. If we assume for a minute that there are 100,000 stocks and options that trade in the United States, on any given day there will be somewhere in the neighborhood of 250 bad end-of-day prints alone, based on our experience. That is a lot of bad data. Historical data with these errors, however, may have already been cleaned (at least according to the vendor). This may or may not be a good thing depending on the timeliness and repeatability of the process. Here are some common types of bad data.

TABLE 15-1
Cleaning of real-time data means including quality control measures. For example, bad quotes and network failures can lead to bad trades. Systems developed with quality in mind elegantly handle problems, such as bad data, exchange shutdowns, and incorrect third-party calculations (think incorrect index prices). When benchmarking data cleaning algorithms, the product team should be sure to address error handling and system shut-off/down procedures in the event of externally generated exceptions.

Whatever the methods to clean bad data or handle external exceptions are, data cleaning algorithms must be shown to operate on both live-time and historical data. Data cleaning algorithms can add latency to real-time systems. Algorithms that cannot be performed in real time prior to trade selection should not be used on historical data, or else the cleaned, historical data will skew backtesting results.

Cleaning of historical data corrects errors and updates the dirty data source with clean data, or more preferably creates a new data source to hold the corrections set. Maintaining the original, dirty data source in its original form allows the team to go back if a mistake was made in the cleaning algorithms that consequently further corrupted the data. This is true also when reformatting data.

Depending on the time interval desired, the format of the data may not match up with those increments, or bars. (Bars being fixed units of time with a date/time, open, a high, a low, and a close and maybe even a volume and/or open interest.) Given tick data, for example, and a trading strategy using bars, the team may want to analyze bars of different durations—a minute in length, five minutes, a day, a week, or a month. In order to convert the data, reformatting may have to take place, which can, if not controlled properly, introduce new problems.

Since many forecasting models, like GARCH, are extremely sensitive to even a few bad data points, we recommend the team look carefully at means, medians, standard deviations, histograms, and minimum and maximum values of time series data. A good way to do this is to sort or graph the data to highlight values outside an expected range, which may be good (but outlying) or bad data.

For other types of bad data, we recommend running scans to detect suspicious, missing, extraneous, or illogical data points. Here are a few methods used to scan data.

**TABLE 15-2**

<table>
<thead>
<tr>
<th>Scanning for bad data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intraperiod high tick less than closing price</td>
</tr>
<tr>
<td>Intraperiod low tick greater than opening price</td>
</tr>
<tr>
<td>Volume less than zero</td>
</tr>
<tr>
<td>Bars with wide high/low ranges relative to some previous time period</td>
</tr>
<tr>
<td>Closing deviance. Divide the absolute value of the difference between each closing price and the previous closing price by the average of the preceding 20 absolute values</td>
</tr>
<tr>
<td>Data falling on weekends or holidays</td>
</tr>
<tr>
<td>Data with out-of-order dates or duplicate bars</td>
</tr>
<tr>
<td>Price or volume greater than four standard deviations from rolling mean</td>
</tr>
</tbody>
</table>

**15.1.2. Winsorizing Outliers**

Outliers are extreme values, that is, data points far out on the tails of the distribution, that will disproportionately affect statistical analysis. Outliers (that are not errors) contain important information, but their presence should not obscure, or even obliterate, all other data and information. To reduce the distortion, data cleaning can either delete outliers from the sample, or, more likely, winsorize them using a compressing algorithm. Winsorizing pulls outliers in toward the mean by replacing them with a value at a specified limit, say three standard deviations. For example, for 90% winsorization, the lowest and highest 5% of observations are set equal to the value corresponding to the 5th and 95th percentile. A winsorized mean will be a more robust estimator because it is less sensitive to outliers. A problem with winsorizing all the data is that volatility may shift over the time series, so we recommend winsorizing on a rolling basis.

**15.1.3. The Point-in-Time Data Problem**

Dirty data is of course problematic, but cleaned data also has problems. Consider the following scenario: stock price data for a day.
is cleaned after the close of business and an adjustment file is sent out the next day, reflecting the corrected price data. This occurs regularly due to block trades, paper trades being entered late, and crossing trades, all of which become part of the end-of-day pricing set.

With the new price data, many calculations for derivatives, which were based on the incorrect data, are now wrong. Implied volatility calculations will be wrong. All OTC derivative contracts that are based on the stock’s implied volatility will be wrong.

End-of-day rebalancing algorithms could also be wrong due to dirty data.

The cleaned data arrives several hours, even up to a day, later. The cleaned data may or may not be the actual value that would have been seen in the market in real time. The cleaned data, now in a historical database, could be used to make a trade during backtesting. But, again, this data was not available when the trade would have taken place.

A cleaning algorithm for end-of-day prices is very important, but not as easy as it sounds. At the close, block orders affect prices, traders sometimes push bids and asks around, and small orders can move the market in the absence of late liquidity. All these things blur the definition of closing price. (Closing price data for futures is generally clean since contracts trade usually only on one exchange, but for stocks and bonds, the calculation is more difficult. With commodity futures, however, closing price data can pose a problem on days where prices are lock-limit down or up. One may not be able to ascertain which price is the closing price. Even where one can make such a determination, OTC contracts which continued to trade may differ. Likewise, the market for a stock can be closed for news or other anomalies. In such cases, the exchange may or may not set a closing price.)

1. **End-of-day stock prices.** To calculate a clean closing price, we recommend getting a nonofficial closing price five minutes (the closer to close you get the data, the higher the probability you will have an error) prior to close across selected exchanges. Throw out all locked or crossed bids and asks and then determine the midprice. Finally, weight those prices by order book quantity. We recommend you ask your data vendor how they calculate end-of-day NBBO values. How do they calculate VWAP for the day? For the whole market or just for certain exchanges? And can I select exchanges? Do they do average weighted bid based upon real size 100 lot or 1000 lot.

2. **End-of-day option prices.** We recommend first getting the option contract bid and ask prices five minutes before the close. Second, get the borrow and stock loan rebate rate for the day as well as the risk-free rate. Third, choose the appropriate option pricing model, preferably one that includes jumps for key events, and obtain (for stock options) the next earnings date and projected dividend dates and amounts. Fourth, calculate the implied volatility on a strike-by-strike basis and on a surface basis, based on clean inputs. Fifth, calculate a clean delta, and delta weight the implied volatility by strike for calls and puts. Sixth, deal with zero bid or ask prices, that is, calculate the wings. Seventh and last, calculate a clean option price, which should be between the market bid and ask to confirm accuracy.

We recommend (unlike stocks where we recommend blending multiple prices) using the exchange with the highest volume for that contract as the official market for that contract, and ignoring volume on other exchanges. If there are two that are close in volume for a particular contract, calculate an implied volatility for the bid and ask in each market and then blend the implied volatilities together.

In summary, if you are unable to take market bid and ask data, run it through a cleaning/smoothing algorithm, get an implied volatility, and recalculate a fair price that is not between the market bid and ask, then you probably do not have a good closing price cleaning algorithm.

The point-in-time issue also applies to fundamental data, which also may be cleaned by the vendor or revised according to accounting rules. In a backtest, the team may select fundamental data, for example, quarterly free cash flow data, that was revised sometime after the quarterly release date. This revised data taints the backtest; it is different data than was available on the release date. Based upon the new data, a stock that was originally bought may have been immediately sold since the original calculation was now in retrospect incorrect. The data adjustment may affect the entire sector as well, since the adjusted numbers may alter the sector mean and standard deviation, resulting in a complete reranking of the outputs of the trading/investment algorithm. To solve this point-in-time problem, many firms require a one to two month lag of data for backtesting. A lag is an artificial time interval introduced into the data to account for this point-in-time problem.

### 15.1.4. Demeaning and Standardization

Factor demeaning, where the average value is subtracted from the observed value, removes bias from the factor. For example, to demean book-to-price by industry, you subtract the average book-to-price for the industry from each company’s book-to-price figure. This reduces the industry bias, and makes companies from different industries or sectors more comparable in analysis. This is quite an important step in model construction, since book-to-price for a high tech firm will differ significantly from that of an electric utility, for example.

When combining factors into a model, it is useful to measure the factors in the same terms, or on the same scale. Standardization, or detrending, accomplishes this by rescaling the data distribution so that it has a specific mean and standard deviation (usually 0 and 1, respectively). Once a sample has been standardized, it is easy to determine a number’s relative position in that sample. To standardize a factor, the mean of the sample is subtracted from an observation, and the resulting difference is divided by the standard deviation.
15.1.5. Scaling and Ranking

The strongest and most direct way that scaling influences most nonlinear models is through the implied relative importance of the variables. When more than one variable is supplied, most nonlinear models implicitly or explicitly assume that variables having large variation are more important than variables having small variation. This occurs for both input and output. Most training algorithms minimize an error criterion involving the mean or sum of squared errors across all outputs. Thoughtless use of such criterion will cause the training algorithm to devote inordinate effort to minimizing the prediction error of the $100, while ignoring the $1, stock. The fact that 100 times as many shares of the $1 stock may be purchased is not taken into account. The scaling of each variable must be consistent with its relative importance.

We also recommend ranking fundamental data. For example, earnings should be reflected in percentile by sector, as should implied volatility. A biotech company will always have a higher implied volatility than a consumer products company. Therefore, call away returns for a biotech would always be higher since implied volatility is higher. We also recommend ranking the call away return to ensure against selling covered calls on all biotechs.

15.1.6. Synchronizing Data

Databases of the different types of data have different updating periods. Take splits, for example. Price data vendors update daily. Balance sheet vendors update weekly. As a result, a given ratio, such as sales-to-price, may contain an unsplit sales figure and split price. Fixing this problem is called synchronizing the data, accomplished by either buying synchronized data or performing the task in-house. (Because divide-by-zero errors can cause problems, we recommend adding a filters flag to very small divisors. Any calculation that has division as part of the calculation needs to have clean data to avoid problems.)

The real key to synchronizing data, or blending data, is a Rosetta Stone. A Rosetta Stone is the set of unique identifiers used to link data and instruments across vendors. A proper Rosetta Stone is highly valuable since it will allow the trading/investment system to trade many instruments—stock, options, bonds, CDs, and OTC products—on a single underlying. Furthermore, unique identifiers across underlyings and across vendors enable the blending of proprietary data with purchased data. We believe the ability to trade multiple instruments of a company using both vendor-supplied and proprietary data is a key to building a system that will beat its index or peer group benchmark.

15.2. STEP 2, LOOP 2: Clean and Adjust for Known Issues

The purpose of this step is to take the manually built cleaning algorithms (probably done in Excel) and convert them into tools that can be used by junior people, with well-defined GUIs and outputs along with algorithms that can be manually run against the entire database. The cleaning algorithms at this point should be viewed as prototypes. Also, the tools built for this step should be placed in a library for future use for all other projects that use the data set.

15.3. STEP 2, LOOP 3: Document Cleaning Algorithms

Over the last loop, the product team should produce and document the algorithms that will be run everyday to clean data. We recommend the team document the algorithms as use-cases or sample code from Loop 2. The use-cases should illustrate what the inputs are; what the outputs are; plus, a written description. The team needs detailed descriptions of the algorithms with sample code and test cases, so that in Stage 3 they can implement the algorithms as part of the software development process.

The team must also produce a schedule of cleaning activities and a time line, that is, what happens when and how long it will take. For example, historical price data for the day may come in at 3 p.m., and fundamental data at 8 p.m. The team must schedule jobs accordingly. The document should also outline what manual GUI tools need to be built, for example, charts, with user manuals.

If the pricing data is missing, or is late, everything else must stop until it shows up. The interactive tools let someone overwrite the data with clean data, for that we again suggest separate tables. (If you clean the data, why tell your vendor?) Overwrite the vendor’s data. There is no need to tell the vendor about their dirty data. All the corrections should be in a separate table of changes. This is the loop where the database programmers get involved in how much time, how much money, and how complex it will be to build the data cleaning algorithms and produce a plan on the time, cost, and structure to accomplish it.

15.3.1. Trade Cost Analysis

Backtesting results depend on execution assumptions. For a working system, in Stage 4, poor execution may cause nonconformance with performance metrics experienced during the backtest. In this step, we also recommend that the product
team document best execution policies, trade cost analysis, benchmarks, and algorithms. In a working system, we further recommend that the product team automate posttrade reporting and analysis. Execution performance should be monitored to ensure that it is delivering competitive advantage and reviewed on a periodic basis.

Developing and documenting a formal policy will make communication with top management and investors a straightforward exercise. Investors now demand that money managers both achieve and prove best execution, where best execution is generally benchmarked against implementation shortfall, arrival price, or volume-weighted average price.

15.4. Summary

For many systems, the size of the databases used to store data is in the half-terabyte plus range. So, you should expect with this amount of data to have errors, omissions, and issues. In a world where the difference between the 25th percentile and 75th percentile of returns is measured in basis points, not understanding and cleaning your own data relegates you to average performance at best. Therefore, we suggest that your most senior financial engineer along with your most senior programmer commit a large amount of time to cleaning data so that you have a competitive advantage over those who do not clean data.

15.4.1. Best Practices

- Design data cleaning algorithms to operate on live-time as well as historical data.
- Initially analyze distributions graphically with scatterplots and histograms. Build tools to allow junior-level people to quickly determine the quality of data.
- Winsorize, scale, rank, demean, and standardize data and define methods for dealing with point-in-time data problems. Also, create a Rosetta Stone to link data.
- Benchmark and document all cleaning algorithms.
- Standardize methods for calculating national best bid and offer closing prices.

CHAPTER ◆ 16

Perform In-Sample/Out-of-Sample Tests

A backtest is a simulation of a trading/investment strategy’s response to historical data. Essentially it is an elaborate quality assurance test to check the model parameters and assumptions, and verify the system’s ability to meet required performance specifications in a prototype production environment. Performing proper in-sample and out-of-sample tests is perhaps the most critical step in the trading/investment system development process.

![Diagram of backtesting process](image)

In backtesting, financial engineers are keenly aware of the extent to which in-sample results may differ from out-of-sample results and trading algorithms must be examined against both before progressing to the implementation stage. A well-developed system will perform similarly out of sample as it does in sample, so it is important to save some of the historical data for out-of-sample testing. A backtest will result in one of three outcomes for a trading/investment system:

1. Profitable both in sample and out of sample.
2. Profitable in sample, but not out of sample.

3. Unprofitable both in sample and out of sample.

If the system is profitable both in sample and out of sample, it will very likely receive capital to begin implementation and trading as soon as possible.

You can buy "Quality Money Management" at a 10% discount in the Automated Trader Bookshop