Data is Messy... Start Wrangling

Trifacta enables data scientists to explore, clean, transform and enrich data of all varieties and volumes for agile and exploratory analytics.

@start-wrangling
@trifacta
Data Wrangling
Techniques and Concepts for Agile Analytics

Tye Rattenbury, Joseph M. Hellerstein, Jeffrey Heer, and Sean Kandel
This Preview Edition of *Data Wrangling: Techniques and Concepts for Agile Analytics, Chapters 1–4*, is a work in progress. The final book is currently scheduled for release in March 2016 and will be available at oreilly.com and other retailers once it is published.
# Table of Contents

**Foreword.** ..........................................................  v

1. **Understanding the Practice of Data Wrangling.** .........................  1
   - Data Project Contexts 1
     - Exploration 1
     - Curation 2
     - Production 2
   - Roles and Common Projects 3
   - Data Wrangling Process 4

2. **Transformation + Profiling: the Inner Loop of Wrangling.** ............  7
   - Transformation 7
     - Structuring 8
     - Enriching 9
     - Cleaning 9
   - Fundamentals of Data Transformation: Processing Complexity 9
   - Profiling 11
     - Type-Based 11
     - Distributional 11
   - Transformation + Profiling Dynamics 12

3. **Overview of Data Wrangling Tools.** ..................................... 15
   - Programming Languages 16
   - Schema Mapping Tools 17
   - Spreadsheet Applications 17
   - Visual Data Preparation 18
     - Trifacta 20
   - Adjacent Tooling 22
Through the last decades of the 20th Century and into the 21st, data was largely a medium for bottom-line accounting: making sure that the books were balanced, the rules were followed, and the right numbers could be rolled up for executive decision-making. It was an era focused on a select group of IT staff engineering the “golden master” of organizational data; an era where mantras like “garbage in, garbage out” captured the attitude that only carefully-engineered data was useful.

Attitudes toward data have changed radically in the past decade, as new people, processes and technologies have come forward to define the hallmarks of a “data-driven organization”. In this context, data is a medium for top-line value generation: providing evidence and content for the design of new products, new processes and ever-more efficient operation. Today’s data-driven organizations have analysts working broadly across departments to find methods to use data creatively. It is an era where mantras like “extracting signal from the noise” capture an attitude of agile experimentation and exploitation of large, diverse sources of data.

Of course accounting still needs to get done in the 21st Century, and the need remains to curate select datasets. But the data sources and processes for accountancy are relatively small and slow-changing. The data that drives creative and exploratory analyses represents an (exponentially!) growing fraction of the data in most organizations, driving widespread rethinking of processes for data and computing—including the way that IT organizations approach their traditional tasks.

The phrase “data wrangling”, born in the modern context of agile analytics, is meant to describe the lion’s share of the time people spend working with data. There is a common misperception that data analysis is mostly a process of running statistical algorithms on high-performance data engines. In practice, this is just the final step of a longer and more complex process; 50-80% of an analyst’s time is spent wrangling data to get it to the point where this kind of analysis is possible. Not only does data wrangling consume most of analyst’s workday, it also represents much of the analysts’ professional process: it captures activities like understanding what data is available;
choosing what data to use and at what level of detail; understanding how to meaningfully combine multiple sources of data; deciding how to distill the results to a size and shape that can drive downstream analysis. These activities represent the hard work that goes into both traditional data “curation”, and modern data analysis. And in the context of agile analytics, these activities also capture the creative and scientific intuition of the analyst, which may dictate different decisions for each use case and data source.

We have been working on these issues with data-centric folks of various stripes — from the IT professionals who fuel data infrastructure in large organizations, to professional data analysts, to data-savvy “enthusiasts” in roles from marketing to journalism to science and social causes. Much is changing across the board here. This book is our effort to wrangle the lessons we have learned in this context into a coherent overview, with a specific focus on the more recent and quickly-growing agile analytic processes in data-driven organizations. Hopefully some of these lessons help clarify the importance — and yes, the satisfaction — of data wrangling done well.
This chapter summarizes the process of data wrangling and the contexts within which people wrangle data. We’ll move from a macro perspective to a micro, as we zoom in from contexts to roles, projects and processes.

Data Project Contexts

As described in the foreword, we generically use “data wrangling” to refer to any data transformations required to prepare a dataset for downstream analysis, visualization, or operational consumption. It is the downstream utilization of the dataset, or datasets, that sets the project context.

In broad strokes, there are 3 main kinds of data project contexts:

- exploration
- curation
- production

We’ll cover each in turn.

Exploration

*Exploration* projects are, as you might expect, focused on learning just what is “in” the data – from the perspective of the meta-data as well as the particular values contained within the dataset(s). This context has grown quickly in recent years, in part because data is an increasingly inexpensive commodity to capture. Computing and storage costs have gone way down in recent years. At the same time, new software systems have made it cheap and easy to store data without requiring a schema in
advance: a “schema-on-use” approach, rather than “schema-on-write”\(^1\). And with the decreasing cost of data has come more widespread and aggressive usage of data. More and more industries, academic fields, and general human endeavors seek to leverage data to choose smarter, more effective actions; as a result we see the breadth and depth of exploratory analytics increasing. Not only has the volume of data been growing within organizations, but also the variety of that data. The core of these exploration use cases is the incorporation of more and different datasets. Quite simply, the sheer growth in the variety of data available makes exploration the common—and likely the majority—data project context.

**Curation**

The second most common context, *curation*, involves deliberate efforts to build, structure, and store datasets in the most useful way for downstream utilization. In the recent past, curation was the purview of data architects and database administrators; it typically involved a heavyweight process of designing schemas and views to support the widest distribution of downstream analysis while also enabling an ease of maintainability as the datasets grew or required editorial changes.

In a modern analytics-oriented context, curation is still important – it is the logical next context after exploration. Once we know what data we have, we can effectively structure and store it for reuse. The effectiveness of the structure and storage of a dataset is, of course, relative to the variety of downstream use cases that can leverage the data without doing redundant processing – e.g., if most downstream analyses start by joining the same curated datasets via the same join logic, it likely makes sense for the curated data to be stored post-join.

**Production**

The third major data project context is *production*. Also referred to as operationalization, projects in this context arise when the value of a data source or feed is established (e.g., its predictive capabilities or, when properly visualized, its marketing capabilities), and the organization seeks to harvest that value in an ongoing and automated way (e.g., by delivering regular recommendations to customers, like which shows to watch or which products to buy). The systematic exploitation of a recurring dataset engenders a new set of concerns – namely the robustness and maintainability of the data pipelines that prepare and then extract the value from that source of data.

---

\(^1\) In the traditional, accounting driven data use cases data was only written, or stored, when it conformed to a pre-specified schema. The adherence to a specific schema enabled well-defined computational operations like summing revenues over product lines or measuring the difference between last year’s revenue and this year’s. Alternatively, schema-on-use refers to the practice of storing any and all data. And, when analysts seek to leverage that data, requiring them to transform the data into a use-case specific schema.
Roles and Common Projects

Zooming in from the macro contexts we can look at data projects at a more granular level. It’s useful to consider the people involved in these projects, and the analytic goals of the projects themselves.

We begin with a list of typical job roles, and example projects that are associated with those roles. Although this list is by no means exhaustive—in either the range of roles or projects—it hopefully provides a sense of the variety of people and objectives involved in analyzing data.

- Data Journalist – craft compelling stories that surface in, or are supported by, data.
- Data Analyst – close the feedback loop in modern companies by managing core reporting pipelines and fielding ad-hoc follow-up queries.
- Data Scientist – assess the potential value of dataset(s) for analytical or data-driven product innovations.
- Data Architect / DB Manager – design the most effective structure and storage of a dataset (or datasets) to support downstream use cases.
- Privacy / Security Analyst – assess information leaks in published or publicly leveraged datasets.
- Financial Quants – identify predictive value in datasets to help drive investment decisions.

Next we discuss some conceptual categories of projects. Similarly not exhaustive, this list should provide some basic awareness of the common kinds of data projects that involve a heavy amount of wrangling:

- Basic Reporting – listed above as the purview of Data Analysts, basic reporting is the primary commercial analytics use case. And it applies to every kind of business: from small to giant, from retail to manufacturing to service. At its core, data-driven reporting is about providing feedback on the efficacy of actions taken. Want to know if your sales activity is up or down across different product lines? The answer should be in a report.
- Causal Analysis – in many cases knowing what happened simply leads to questions of ‘why’ and ‘what can we change to affect the outcome’. From health outcomes to brand awareness, causal analyses appear in a variety of forms and generally involve careful, often complex, statistical analyses to ensure that variations in the outcome data can be explained by variations in the input data.
- 360° View – a precursor to most causal analyses, these kinds of projects are primarily concerned with the integration of a variety of data sources to generate a more complete picture of, for example, a customer or a business operation or a natural phenomena in the world. Building the 360° View can also drive reporting, where patterns across datasets can now be surfaced.
• Predictive Modeling – a cousin to causal analysis, the goal here is to find aspects of historical data that predict future events. In the service industries, a common flavor of this kind of project is the “customer churn analysis”, which seeks to predict which customers are likely to stop being customers in the near future. More recently, a number of new services have emerged that leverage historical behavior data to predict what kinds of content and services a customer is most likely to buy or want – enabling more efficient product delivery and higher customer satisfaction. In the financial world, predictive modeling is often used to predict market movements or shifts in risk profiles.

Other than basic reporting, which is pretty firmly a production style data project, the above data projects could be involved in exploration, curation, or production contexts. Explored initially, as these projects show consistent value the incorporated datasets might spur parallel curation efforts and eventually become full production pipelines.

**Data Wrangling Process**

Switching focus from contexts, roles and common data projects, we’ll spend the rest of this chapter introducing the primary elements in the data wrangling process.

In broad strokes, working with data involves the following activities:

- access
- transformation
- profiling
- output

In classic enterprise data settings, **access** is the Extract portion of the “ETL” pipeline. In the world of “big data”, access often involves pulling data either by scraping various internet endpoints or by linking to existing data stores (both internal and external).

**Transformation** is the functional aspect of wrangling—when the data changes shape and the contents are validated, and possibly altered, to meet the needs of downstream activities.

**Profiling** motivates and validates transformation. Transformation does not happen in a vacuum; it requires a user to understand what is in the data, both before and after the various steps of transformation. Profiling provides descriptive statistics that enable users to decide which transformations to apply, and whether the applied transformations achieved the intended effect. Especially in the realm of “big data”—which we can conservatively characterize as data at scales beyond the ability for humans to manually audit and manipulate—profiling is the only mechanism for assessing a dataset.
Together, transformation and profiling form the core of data wrangling. In subsequent chapters, we will delve into the kinds of actions that are taken to transform and profile data. Of note here is the iterative cycle between transformation and profiling – action, feedback, action, feedback, etc. One way modern data wrangling tools make this process more efficient is by providing targeted suggestions to arrive at the most effective transformation sooner, and by providing ongoing feedback to confirm that the transformation is performing as intended.

Finally, output corresponds to the completion of the wrangling process. There are two main material outputs: the wrangled dataset and the script of transformation logic (often containing profiling checks that confirm the output dataset has the intended structure and content). For production projects, in addition to outputting the script of transformation and profiling logic, there is often a need to document the lineage/provenance of the data (where it was sourced from, which systems applied the transformations, and so on) as well as the reasoning behind the transformations (akin to comments in software, to make the script easier to maintain and update over time).
Transformation and profiling are the core steps of data wrangling. In this chapter, we’ll discuss the common actions involved in both, and the dynamics that link them in actual workflows.

**Transformation**

Although not always explicitly represented, a common defining characteristic of a dataset is the repetition of structure, corresponding to a collection of similar “records” or “objects”. When it is explicit, this repeated structure is often referred to as the *schema* of the dataset; the individual data records instantiate the schema’s template with specific measurements or values.

Data transformation is “complete”, at least to a degree, when a dataset has a well-understood schema/structure and the data values populating the dataset conform to some defined measures of quality and consistency. Sometimes the transformation starts with data that either has no schema (i.e., is unstructured or semi-structured), or is heterogeneously structured and requires manipulation to bring the various structures into alignment. Other times, the transformation work starts with well-structured data and mostly involves mutating that structure and the data contents to serve new analytical needs.

A second defining quality of well-structured datasets is the ability for each individual record to be self-contained: interpretable on its own, with all the values of its schema filled in. This is not always the case when data arrives! It is common, for example in spreadsheet data or semistructured formats like XML, for a value in a record to be empty, and implicitly defined by the value of a neighboring value (often “above” it in
the spreadsheet or file). From an operational perspective, getting data to be self-contained enables certain “record-at-a-time” operations to work correctly and independently, without reference to other records. Self-contained records are also useful for analyses that span records like joins, summaries and correlations—even in these cross-record cases, the algorithms that perform these tasks typically benefit from having self-contained records.

To illustrate this point, consider again the customer transaction example. To minimize the original storage and transfer costs of this data, only the top of transaction logs contain store information, including a store ID and address. Individual transaction records in the remainder of the log do not store the store ID or address; they implicitly link back to this header information. To render this dataset more useful in a variety of analytical packages, the store ID and store address must be “filled down” through the log—appended to each transaction record below. This enables, for example, an analysis of how many different stores a particular customer shops in.

A more complicated scenario exists when the data within the dataset requires augmentation with metadata existing outside of the dataset. For example, consider a dataset stored as a file. It is common practice to give the file a name that describes the data contained within, include perhaps an indication of the time the file was generated and the source of the file. The filename itself, then, contains valuable metadata about the origins of the data. It is often desirable to include the filename, or derivations thereof, within the records of the dataset itself.

In the parlance of this book, we will refer to actions that pull metadata down into the dataset as **preprocessing**. We separate out preprocessing actions from the remainder of the transformation actions that we will discuss for two reasons. First, these steps often occur before any other transformation can take place. Second, they often do not require significant iteration to get right.

Once all of the desired data is contained within the dataset itself (possibly still requiring filling to generate complete records), we move into the more iterative transformation actions: **structuring**, **enriching**, and **cleaning**.

**Structuring**

**Structuring** refers to actions that manipulate the schema of the dataset using just the data from within the dataset. Most commonly this involves modifying the schema by splitting a column/field into multiple, collapsing multiple columns/fields into one, or removing columns/fields entirely. However this can often get messier. Modern datasets increasingly come in log or protocol formats that involve hierarchical groups of fields, e.g., “map” (a.k.a “hash” or “dictionary”) structures with multiple key-value pairs stored in a single field of a single record. Downstream tools often require simple tabular formats, requiring these hierarchical fields to be unnested into single-value-per-field formats.
While we will dive into specific transformation actions that execute structuring changes to a dataset in subsequent chapters, of note here are structuring changes that also correspond to changes in the granularity of a dataset. For example, suppose we have a dataset of customer transactions. We might decide a better dataset for our analysis purposes has the coarser granularity of customers. To transform the original dataset towards the latter, we might group together and aggregate (summarize) transactions for each customer, replacing the original transaction amount column/field with new a column/field corresponding to, say, the total amount transacted over the lifetime of the customer.

**Enriching**

*Enriching* refers to the addition of columns/fields that add new information to the dataset. This can happen simply via computation on the data already in the dataset. As a simple example, we might compute an arithmetic expression over our data—say converting counts into percentages. As a more sophisticated example, suppose we had a dataset of customer support interactions. We could process the text of those interactions and derive an overall sentiment score to assess whether the customer was satisfied.

In practice, enrichment by computation is the exception; more commonly, data gets enriched by *joining* additional data. For example, to the customer level data generated by aggregating customer transactions, we might want to join in additional customer profile information stored elsewhere, such as name, email address, gender and age.

**Cleaning**

So far we’ve covered transformation actions that are characterized primarily by their impact to the dataset schema. Structuring alters this schema deductively using just the data already present in the dataset. Enrichment augments a dataset and its schema with new data, either by joins or non-deductive derivations (involving, for example, inferences and predictions). *Cleaning*, by contrast, is focused exclusively on the values within the dataset. Given an intended set of semantics behind the columns/fields (typically expressed as *constraints* on the data), cleaning seeks to ensure that the records provide “valid” values for every column/field in the schema. Profiling insights, which we’ll cover in the next section of this chapter, help analysts remove or remediate data quality issues using a variety of cleaning transformation actions.

**Fundamentals of Data Transformation: Processing Complexity**

There are certain fundamentals to data transformation that every analyst should know, independent of the transformation actions described above. In the rest of this
section, we'll cover one: processing complexity. There are essentially three main classes of transformation processing: (1) scalar (“map”) actions that can be executed on each record completely independently of every other record, (2) stream-group actions that require bringing together more than one record but can do so in a streaming and randomly ordered manner, and (3) holistic-group actions that require bringing multiple records together and processing them in a particular order and/or all at once.

Scalar actions are the cheapest, because the records can be handled in any order—even via parallel “workers” (machines, processes, threads, etc.) A good example of this kind of action is something like a currency conversion on transaction records—it can be done independently for each record. If you have 100 workers and 100 records, each worker could independently handle 1 record without coordinating with its peers. Thus, the total time to finish the action involves three parts: (a) the time to divvy up the records, (b) the slowest worker’s time to handle one record, and (c) the time to collect all the results back together.

The stream-group actions are more expensive than scalar actions. They correspond to tasks like aggregations—e.g., summing the total transaction amounts per customer. Whereas in the former type the divvying up of records could happen randomly (it doesn’t matter which worker gets which record), divvying up records in this case requires a bit more organization. Specifically, all the records that belong in a group (for example, all the transaction records for a single customer) should end up at the same worker\(^1\). Consider again the case of 100 workers and 100 records. Let’s assume 70 of these records came from one customer and the remaining 30 came from a second customer. In this case, a natural solution would use only 2 workers to get the job done—one per customer—and the time to complete the action would be based largely on the time it takes 1 worker to handle 70 records (assuming the processing time of each record is about the same).

The holistic-group actions are the most expensive class of the three. A canonical example is to compute the median in each group—in our example, we might want to know the median price of an item purchased per customer. Recall that the median item per customer is the one that is higher-priced than half the items the customer bought, and lower-priced than the other half. While the divvying up of records to workers follows the same logic as in the second type, a new wrinkle is added here by the constraint that workers must process the records in a particular order, and/or keep all the records available together during the computation. In our example, com-

\(^1\) In fact, the actual constraint is that one worker must produce the final result of the computation. Depending on the kind of computation, more sophisticated processing flows—called partial pre-aggregation—can sometimes be used. In these flows, multiple workers handle subsets of the records and produce partially-aggregated results, which are combined together by a final output worker.
puting the median exactly requires sorting each customer’s items by price. When records need to be processed in a particular order, additional computational time is required to sort the records. When records need to be processed all at once (because they need to be cross-referenced, for example), the workers processing the records must have enough space to store the records. For large amounts of records, workers may not be able to fit them in memory, and will require mechanisms to leverage backup storage during processing. The additional overhead of moving records in and out of this backup storage can add significant processing time.

In subsequent chapters of this book we link these three types of processing complexity to specific transformation actions to help readers assess the overall costs of the transformation steps they are applying to their datasets.

**Profiling**

As we touched on above in our brief description of cleaning transformation actions, profiling is the core mechanism that provides feedback to an analyst on the quality of the data values in a dataset. In this book, we’ll focus on two main kinds of profiling: type based and distributional/cross-distributional.

**Type-Based**

Columns/fields in a dataset are often annotated with an intended type, like `integer` or `email_address` or `URL`. Types are essentially a named collection of constraints, which can involve both semantic and syntactic requirements. For example, a column/field might be typed as `bank_account_balance`, which would likely entail floating point (i.e., decimal) numbers with 2 digits of precision. And, depending on the type of account, might have the additional semantic constraint of being non-negative.

In its most basic form, type-based profiling involves calculating the percent of values in a column/field that satisfy the constraints of the type. More sophisticated type-based profiling might provide additional categorization of invalid values—grouping them based on the way(s) they violate the type constraints.

**Distributional**

*Distributional profiling* enables the detection of deviation-based anomalies—values that are in some sense “too far” from what’s expected of the data. For example if a column/field consists of numbers, the distributional profile would clearly highlight anomalies like extreme values on the low or high end of the value range, or significant overall trends like multiple modes (“humps in the curve”). For columns/fields typed categorically (e.g., categories like names, email addresses, URLs), the distributional profiling should provide feedback on the number of unique values in the column/field (its cardinality) as well as on the frequency of occurrence of each unique value.
value—of particular interest are the values that appear unusually often. An extension to this profiling would involve mapping the categorical values into a smaller number of classes and profiling the classes. One way to think about this is as a standardization process—e.g., converting the various ways to notate US states (Florida, FL, Flor., etc.) into one canonical representation—followed by categorical profiling of the canonicalized result.

Cross-distributional profiling enables the detection of quality issues that only surface when patterns across multiple columns/fields are considered. For example, consider the pair of columns “transaction amount” and “customer ID”. Transaction amounts might range from $0.99 to $1500.00, and the distribution of those amounts might appear to match expectations. However, highlighting a customer ID that appears significantly more often than others might highlight a pattern of transaction amounts (say all at the 99¢ level) that could indicate fraudulent, or at least invalid tracking of that customer’s behavior.

Extending beyond the information presented in profiling, we should also consider the usability of the profile information. Three primary concerns are notable. First is the format that these profiles are presented in. In many cases, the ideal format will be visual—giving analysts the ability to quickly glance at a chart or graph to understand the quality of the data contained in that column/field. To support cross-dimensional profiling, it helps to make these charts and graphs interactive and linked—so that interactions in the profiling visualizations of one column/field trigger changes in the other column/field visualizations as well. Finally, and most importantly, the profiling information will inherently involve summaries over the entire dataset (e.g., the percent of values in a column/field that are valid relative to the intended type) so it is imperative that enough granularity is preserved to interpret the profiling information and link it back to the actual data. Providing some specific column/field values as part of the profiling information is often sufficient to give users useful context for the summarized values.

Transformation + Profiling Dynamics

In the remainder of this chapter, we discuss a conceptual framework for understanding how typical real-world data workflows iterate between transformation actions and profiling. Thinking along a temporal dimension, we’ll consider three scales of transformation–profiling iteration: (1) within-transformation action, (2) across-transformation logic in a script of data transformations, and (3) profiling in a larger data pipeline that involves multiple transformation scripts.

Within a transformation action—or, more accurately, during the steps taken to define the transformation you wish to apply to your data—it is often useful to get feedback on how small changes in the transformation specification affect the impact to the overall dataset. Ideally one gets to visually observe the impact to all of the involved
columns/fields at once. Recall from our discussion above that we may need to understand the impact not just on a handful of records, but over the entire dataset—this requires the profiling software to generate new profiles over the entire set of records to drive this visualization. Ideally this profiling changes in real time along with changes to the transformation you are building.

Across blocks of transformation logic within a script, it is necessary to ensure that entire blocks have the intended impact on the dataset. While checking each individual step within a block offers some confirmation of this, additional checks at the block boundaries are generally also required. For example, checking for and flagging anomalies at these boundaries helps enforce the data quality assumptions that each block is making.

Zooming out yet again, a very similar set of boundary profiling needs to happen between transformation scripts that are linked together in a more extensive production pipeline. Generally at this level the profiling takes on the additional capability of alerting you when intended data quality checks are not met – triggering triage and remediation activities.

For the remainder of this book, we’ll focus primarily on the first two time-scales of wrangling: specifying a single transformation and making sure blocks of transformations are functioning as intended. Before diving deeper into the specifics, we provide a brief overview of existing tools and products for data wrangling.
Tools for wrangling data span a number of dimensions: from general purpose programming languages to commodity spreadsheet applications such as excel to more specialized, job management tools like schema mappers to, more recently, visual transformation and profiling products. We’ll provide a survey of these tools along with some commentary on key features, strengths, and weaknesses. First, some key features:

- **Scale**: Does the tool scale to meet the volume, latency, and throughput requirements of your use case? Can the tool generate code deployable in a distributed environment?

- **Expressive Power**: Does the tool express the variety of transformation and profiling queries necessary to complete your tasks?

- **Assisted Transform Specification**: How much assistance does the tool provide while authoring transformations? Do you need to fully specify transformations and their parameters or can the tool help you do that?

- **Integrated Profiling**: Does the tool offer background anomaly detection or summary visualizations that assist you in detecting data quality issues such as missing values, outliers, and inconsistent values? Are these features integrated in the transformation process or offered as a separate, post-hoc feature?

- **User Persona**: Do you prefer hand-coding transformations or working in more visual applications?

- **Use Case**: Do you have highly iterative, exploratory use cases or are you developing and operationalizing a known, recurring workflow?
**Programming Languages**

Expert environments for data wrangling include command line tools, general purpose programming languages, code libraries for charting and statistics, and declarative data transformation languages.

The Unix shell remains a popular command line environment for wrangling data. Unix provides a set of powerful tools including those typical in data cleaning tasks. For example, `sed`, `grep`, and `sort` are well known commands for transforming, filtering and sorting. Analysts compose workflows by piping these commands together in shell scripts.

General purpose programming languages such as Python and Java and statistical packages such as R and SAS are powerful tools for data manipulation. Most of these languages now have libraries specifically designed to improve authoring data transformations, e.g., `dplyr` in R and `Pandas` in Python. Such libraries provide high-level representations for datasets as well as commands for manipulating data. Analysts can leverage the full power of the host language while taking advantage of the utilities of the library.

Special-purpose languages for data manipulation, notably SQL, are typically thought of as languages for performing analysis. However, these languages are also widely used for data transformation workflows, particularly because some of the parallel database engines that run SQL can provide significant scalability beyond what's possible with languages like Python and R. Many SQL implementations now support nested data types natively, making them suitable for wrangling semi-structured data as well. Big Data languages in the Hadoop environment have become relatively widely-used alternatives to standard SQL. For example Hive and Pig have gained some popularity for wrangling data in Hadoop.

Visualization programming languages have also begun including common utilities for wrangling data. Such utilities are necessary to shape source data into a form digestible for visualization. The popular visualization libraries D3 and Vega provide various utilities for reshaping and aggregating data to power both static and interactive visualizations.

Many of these expert environments are expressive enough for a wide range of use cases, though there are limitations. For example, some Big Data languages and even some SQL implementations do not offer advanced functionality such as windowing. By contrast, some SQL implementations and the Big Data languages can scale up and run natively on clusters of multiple machines. By contrast, it is very difficult to work on more than one machines’ worth of data with any of the other solutions, including programming languages like Python and Java, statistical packages like R and SAS, and visualization libraries like D3 and Vega.
Even under the best of circumstances, these programming environments place the burden of writing transformations entirely on the user. Hand-coding transformations in these languages can be tedious and time-consuming even for those with programming expertise. Additionally, users must author and integrate any profiling code necessary to verify their workflow. For these reasons, these tools are typically most appropriate for data engineers or analysts with advanced engineering skills that require highly custom code to operationalize known workflows. Generally, these tools are not a good choice for highly exploratory wrangling tasks, since they interrupt the flow of exploring data with the distracting and complex burden of programming and debugging transformations.

**Schema Mapping Tools**

Traditional ETL tools, such as Alteryx, Informatica, Pentaho and Talend, provide interfaces for creating mappings between well-structured input data schemas and target outputs. These mapping tools typically abstract code-authoring as a task of assembling a graphical “boxes-and-arrows” workflows of linked icons, each icon representing a schema or a built-in data transformation. Users build a graph of transformations by linking together these icons with dataflow arrows to form a directed graph on a canvas. Such tools typically provide facilities for scheduling mappings and data quality tools to profile data in the workflow. They offer visual affordances for simple transformations such as mapping columns across schemas.

Though mapping tools provide graphical icons for individual transformations, end-users must explicitly specify parameters of such transformations. Users primarily interact with and visualize the boxes-and-arrows representation of the mapping during construction. The visual abstraction is at the level of schemas and operators; the actual content of the data is not typically shown on screen. As a result the visualization is mostly helpful in authoring and reviewing the composition of the transformations, not the specifics of the individual steps. The lack of information about the data values means that these tools are not well suited for exploratory use cases, where example data and profiles help users understand what is in their data and what they need to change. Rather, these tools are best suited for data architects and engineers operationalizing and monitoring fairly regular production workflows.

**Spreadsheet Applications**

Spreadsheet applications, like Excel, remain one of the most popular tools for data wrangling (and analysis). Analysts often organize spreadsheet data in non-relational formats that require reshaping to load into other applications. Common examples of spreadsheet reshaping include unpivoting and transposing data, extracting sub-regions of data, and removing annotations and formatting intended for human view-
ers. Users can visualize, and—for a limited set of data transformations—directly manipulate their underlying data in spreadsheets.

Though popular, spreadsheets do not scale to large data volumes. Moreover, the visual interfaces in spreadsheets provide limited wrangling features; to get beyond those basics, analysts must explicitly author transformations and their parameters as formulas or as programs in embedded macro languages like Visual Basic.

Figure 3-1. Spreadsheets allow users to visualize and directly interact with raw data. However, users must explicitly generate most transformations by editing formulas. Here a user must construct a regular expression by hand.

**Visual Data Preparation**

Visual applications such as Datameer, OpenRefine, Paxata, Rev, Trifacta and Wrangler provide interactive interfaces and additional tooling to help users construct data transformations and assess their output, making them accessible to non-technical users.
Figure 3-2. Stanford Data Wrangler: Wrangler generates a set of suggested transformations based on the data and a user’s interactions. Visual previews help users assess the transformations.

Specifying complex transformations is difficult in programming languages, spreadsheet and schema mapping tools. The research community and industry have offered tools using *programming-by-demonstration* and *programming-by-example* techniques that help end users construct transformation without having to explicitly write or parameterize commands. For example, Wrangler suggests transformations based on features of the dataset and interactions with the data. The application can generate regular expressions for parsing transformations based on a user’s text selections or suggest reshaping transformations such as unpivoting data.

Assessing the output of transformations, especially over large datasets, can also pose challenges for users. As a result, many visual applications couple statistical analysis with scalable visual summaries to aid anomaly detection and assessment. Such background profiling is most effective when embedded throughout the transformation process, allowing analysts to iteratively assess transformations during construction.

Additionally, many of these tools package data and APIs to aid common tasks such as reconciliation and augmentation. Many applications support cleaning common data types such as geographic entities or business names. OpenRefine provides APIs to FreeBase, an extensive knowledge graph, for cleaning data across a large collection of curated data types.
By aiding transform specification and embedding profiling into the transformation process, visual applications lower the expertise required to author transformations and reduce the time needed to author transformations for more advanced users. Many visual data preparation products support wrangling at scale by leveraging distributed engines such as Spark. These products are typically well suited for exploratory use cases.

**Trifacta**

As an example of a modern system for visual data preparation we offer more detail on Trifacta, a system we know well. We highlight three unique features of Trifacta here: *predictive transformation, visual profiling, and scalable execution.*

Trifacta is derived from academic research at Berkeley and Stanford, embodied in the Stanford Data Wrangler and Profiler open source prototypes. The focus of the initial research began with the user experience of exploratory data wrangling: understanding what users found difficult, and providing interfaces that would let them easily experiment with transformations and assess the effectiveness of the result.
Traditional visual interfaces like menus or box-and-arrow canvasses do not provide significant assistance to users who are authoring transformations. They place roughly the same burden on users as programming-based interfaces: users have to choose the next command (whether it be by typing, by choosing from a menu, or by picking an icon from a palette) and then choose the arguments to the command (typically in a dialog box in the visual interfaces.)

Trifacta's *predictive transformation* works differently than prior visual interfaces. It starts by letting users see and manipulate input visualizations (tables, charts). Rather than requiring users to visually specify exact commands, Trifacta lets users simply highlight features of interest: whether that be a selection of text in a table cell, a column of a table, or a bar in a bar chart. Based on the user's interaction, as well as the contents of the data and past history, Trifacta predicts one or more candidates for the next operation that the user will want. In essence, Predictive Interaction writes code on behalf of the user; Trifacta's visualizations of the output of the command allow the user to decide which candidate achieves the desired result. Unsophisticated users need not even be aware of the program they've generated—they simply point at problems in their data, and see visualizations of those problems areas after being changed.

A critical companion to predictive interaction is Trifacta's *visual profiling* functionality, which ensures that users have continuous visual cues about the content and quality of their data. Trifacta provides type-aware visual distributional profiles—bar charts, histograms, maps, etc—for every column of data that users are transforming. Moreover, every visualization is interactive, enabling intuitive cross-distributional exploration by “brushing” one visualization, and seeing the “linking” proportions highlighted across other visualizations. For example, by clicking on a picture of the state of California in a visualization of states, you can see the proportion of sales that originated in California in a corresponding visualization of revenue. As users apply transformations in Trifacta and change the shape and content of the dataset, visual profiles continually update to reflect the state of the transformed data. This allows users to explore transformations quite freely, knowing they can visually assess each step of transformation, compare the output of a given step to its input, and fall back to an earlier state as needed.

A third unique feature of Trifacta is its ability to generate transforms at *arbitrary scale*. It achieves this by using language and compiler technology. As users interact with Trifacta, a record of their transformations is captured in a domain-specific language for transformation called Wrangle. Wrangle is a declarative language for manipulating data—something like SQL or Pig—but tailored in two ways. First, it focuses on the domain of data transformation, to include tasks like pivoting and unnesting that cannot be expressed in traditional data languages like SQL that are based on first-order logic. Second, Wrangle was designed to make sense visually: for every operation in Wrangle there is a natural visual metaphor to preview and assess its effect on a dataset.
Wrangle is a carefully-designed, domain-specific language for data transformation, which is cross-compiled to other scalable data-centric languages. Trifacta's version of Wrangle compiles to multiple targets today, including Hadoop MapReduce, Spark, and a lightweight Trifacta runtime as well. A key advantage of this approach is the ability to harness various technologies for scaling—today that's MapReduce for truly huge jobs, Spark of medium-size jobs, and the Trifacta runtime for smaller jobs.

![Figure 3-4. Stanford Profiler. Histograms and visualizations aid in assessment of data quality.](image)

**Adjacent Tooling**

Data wrangling is one part of a larger analysis pipeline that includes data movement, reporting and advanced analytics. Data movement tasks include moving data from applications into data stores or from one datastore to another. Ingestion tools like Sqoop and Gobblin offer adapters and other functionality to support moving from a variety of data sources such as SQL systems to sinks like Hadoop. Flume and Kafka provide flexible frameworks to aid data movement for use cases such as log aggregation and event processing. Though some data wrangling tools offer support for limited forms of ingestion, it is typical in most use cases to move data to a desired source before wrangling.

The most common downstream applications for wrangled data are Business Intelligence and Reporting software, such as such as Tableau, QlikView and MicroStrategy. These applications allow analysts to rapidly construct custom interactive dashboards and reports. Another important downstream application are advanced analytic use cases, such as building models to predict customer churn, offer product recommendations, or detect machine failures. These models are typically performed in statisti-
cal packages like R or SAS (and are often executed in scalable data processing engines such as Spark). These downstream applications often set the context for the wrangling activities – whether it is exploration, curation, or production oriented. Also, in terms of workflow, these downstream applications often reveal other potential data quality issues, leading to an interactive cycle of data wrangling and visualization.

Using our survey of data wrangling contexts, processes, activities and common tools as an orienting map, we will use the remainder of the book to dive deeper into each of these topics from the perspective of a practicing data wrangler.