

# Chapter 8. A Data Science Perspective<sup>\*</sup>

Jiangang Hao and Robert J. Mislevy

Educational Testing Service

## Abstract

Digitally based learning and assessment systems generate large volume of complex process data and the next generation psychometricians need to acquire new data science skills to meet the data challenge. In this chapter, we summarize data science skills and identify the subset that psychometricians need to prioritize. We introduce an evidence identification centered data design (EICDD) process for implementing the evidence centered design (ECD) for digitally based assessment, and describe techniques to parse and process complex process data with example codes in Python programming language. We also outline the general methodological strategies when dealing with process data from digitally based assessments.

**Keywords:** Data Science, Data Model, Process Data, Digitally Based Learning and Assessment

## 8.1 Introduction

Digitally Based Assessments (DBAs) enable the capture of test takers' response process information at finer time granularity than traditional forms of assessment, and these rich process data can provide new opportunities for validating assessments, improving measurement precision, revealing response patterns/styles, uncovering group difference (fairness), detecting test security breaches, identifying new constructs, and feedbacking to

---

<sup>\*</sup> The R or Python codes can be found at the GitHub repository of this book:  
[https://github.com/jgbrainstorm/computational\\_psychometrics](https://github.com/jgbrainstorm/computational_psychometrics)

learners and other stakeholders (Ercikan & Pellegrino, 2017; Mislevy et al., 2014). But these potential benefits do not come for free. The significantly increased volume, velocity, and variety of data pose new challenges to psychometricians for handling, analyzing, and interpreting the data to materialize their value. Data science is *an interdisciplinary field that uses scientific methods, processes, algorithms, and systems to extract knowledge and insights from data in various forms, both structured and unstructured* (Wikipedia contributors, 2018). Some techniques and practices used in data science could and should be adopted into the toolkit of next generation psychometrics (e.g., computational psychometrics, as suggested in this book) to help address the data challenges accompanying DBAs.

Broadly speaking, challenges concerning data can be put into two categories. The first is more on the engineering side, concerning capturing, hosting, transferring and manipulating data with proper software tools and techniques. The second is more substantive, concerning methods for making sense of data to realize their potential evidentiary value for intended interpretations and uses. The engineering side of handling data is generally less domain-dependent, while making sense of data usually depends heavily on the nature and intended use of the data in a specific discipline. Though the capabilities needed to address these two types of challenges may appear different, in practice they often overlap substantially. In recent years, some new professions have emerged to provide personnel and technological solutions to meet the needs of big data. For example, professionals with job titles of data architect and data engineer are usually expected to design and build enterprise data architectures and software infrastructures to host, transfer, and manage large-scale data. Those with a job title of data scientist are often expected to

make sense of the data and materialize their value to support business<sup>1</sup>. The current chapter is not intended to provide a text on big-data engineering or architecting (interested readers may consult relevant volumes such as Kleppmann, 2017). In the current chapter, we introduce some data science techniques and modeling strategies in the context of DBAs to help psychometricians become better prepared to work with the increasingly big and complex data from next-generation assessments<sup>2</sup>.

The chapter is organized as follows. In Section 2, we propose that many data challenges can be mitigated through careful design of data early in design. To this end, we introduce an Evidence Identification-Centered Data Design (EICDD) process for use with the Evidence Centered Design (ECD; Almond, Steinberg, & Mislevy, 2002; Mislevy, Steinberg, & Almond, 2003) framework for DBAs. In Section 3, we introduce some data techniques and examples for manipulating structured and unstructured process data from DBAs. In Section 4, we introduce some methodological strategies towards modeling and interpreting complex response process data from DBAs. Finally, in Section 5, we summarize the chapter.

## **8.2 Evidence Identification-Centered Data Design**

In many fields, a data scientist often starts with data that have arisen coincidentally for purposes other than the ones the scientist has in mind. Neither their structure nor their contents may be optimal for the scientist's purpose. In an assessment, however, as it is often the case that how to use the data later (e.g., scoring and analytics) needs to be foreseen to some degree, analysts and

---

<sup>1</sup> These job classifications are not very strict delineated in practice, given the overlapping skills needed to solve data-related problems.

<sup>2</sup> Some performance assessments use video, audio or other motion tracking mechanism (e.g., eye tracking) to capture test takers' performances. Specialized techniques are needed to process the video, audio or other motion tracking data, and there are established disciplines, such as computer vision and computational linguistics, for those topics. A discussion of these techniques is beyond the scope of this chapter. We focus primarily on the telemetry data that records test takers' interactions with the assessment.

psychometricians actually can and often should influence the data design early in the design process. Before delving into specific techniques to deal with messy data, it will be worth well to see whether some of the data challenges can be avoided if some careful considerations about data have been incorporated into the design process.

Evidence centered design (ECD) provides a general structure for designing complex assessments and spells out a methodological framework to relate the evidence and constructs of interest. One of the core processes in implementing ECD with DBAs is to identify the prospective evidence contained in the telemetry data that record the interactions between the test taker and the assessment task. This includes in particular specifying the more generally defined ECD objects called Work Product Specifications, as well as the input and output data structures to be used by the Evidence Identification and Presentation processes. In practice, identifying the evidence is not always a technically trivial task, especially when data are poorly structured, have incorrect entries due to software bugs, or even fail to include the evidence that should be captured. One of the leading causes of these unfavorable situations in practice is that the people who are in charge of the data generation and capture process (usually the software programmers) are usually not those who will analyze them later (usually the data analysts and psychometricians). They have little incentive to design the data to be analysis friendly.

One possible way to avoid these unfavorable situations is to introduce an evidence identification-centered data design (EICDD) process for carrying out ECD when designing DBAs. There are at least two main components in an EICDD: (1) a data model that specifies the data structure and data type of each field, and (2) a set of standard operational procedures to guide what information should be stored into which field of the data model.

### 8.2.1 Data Model

A data model is an abstract model that organizes elements of data and standardizes how they relate to one another and to the properties of real-world entities (Wikipedia contributors, 2019, October 29). There are three main types of data models, namely, the relational, document, and graph-like models (Kleppmann, 2017). The relational data model, such as a table in a SQL database, is better suited for the cases where the relationship among the attributes of data records is of the main concern in the application. The document data model, such as the XML or JSON string in a NoSQL database, is better geared for cases where the main interest is within each data record, and there is not much interest in the cross-record relationship. The graph-like model, such as those implemented in a graph database, is better suited for data where the cross-record relationship is too complicated to describe with the relational model.

In principle, one can store the same data by following different data models, and it is not always easy to tell which one is significantly better than the others. In most learning and assessment applications, the natural unit of data is usually a learning/assessment session by a learner/test-taker, which can be naturally specified through a document data model. However, many psychometric analyses, such as equating and item calibration, depend on the relationship among many sessions, making the relational data model a good fit. So, in practice, the raw telemetry data dumped into a data lake usually follow the document model, and the relational model generally specifies the processed data stored in a data warehouse. In particular, most relational databases today accept XML/JSON string as a record, making the integration almost seamless.

There are almost infinitely many ways to define specific data models. Some example data models for learning and assessment systems include the one specified under the Experience API

(also known as xAPI or Tin Can API; Kevan, & Ryan, 2016), the one specified under the Caliper Analytics (IMS Global Learning Consortium, 2015), and the document model specified for virtual performance assessments (VPAs) by Educational Testing Service (Hao, Smith, Mislevy, & von Davier, 2014; 2016). Figure 1 shows the structure of the ETS data model for VPAs.

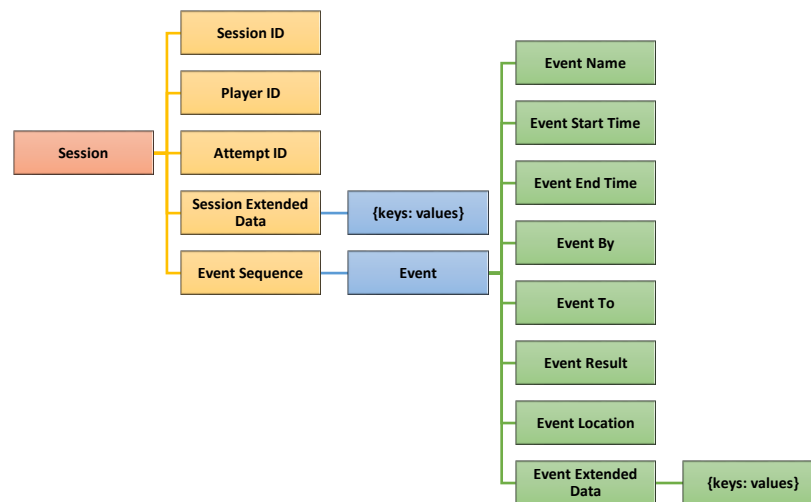


Figure 1. *ETS data model for virtual performance assessment tasks*

### 8.2.2 Standard Operational Procedure

A data model specifies the structure and data type but does not tell what specific information from a particular task should be recorded into which field of the data model. In fact, given the almost infinite possibilities of assessment or learning tasks, this type of information is less likely to be specified through a set of fixed rules. As such, perhaps the best way to put some regularity into such a process is to establish some general guidelines and a set of “standard” operation procedures. In the context of DBAs, though they vary considerably, we can still offer some general guidelines about what events should be captured as follows.

Events in DBAs can be classified as test taker-triggered and system-triggered events. The test taker-triggered events can be further divided into two types. The first type involves the direct

operations on elements designed for assessment, such as clicks on buttons, moves of game elements, typed responses, checked options, and verification of the DBA system status—quite specifically, instances of meaningful actions test takers take in situations with particular features. The entry for any such activity will include timing information and the identification of the elements of the DBA that are involved and, as needed, values of variables for relevant aspects of the state of the environment before and after the activity.

A challenge regarding the last of these is determining what constitutes a sufficient description of DBA states to associate with a given action. The same test taker’s action can take different evidentiary meanings depending on prior actions or the state of the simulation. Note that *identifying what additional information is necessary to interpret the meaning of actions is a substantive, or scientific, question, not simply a data-capture question*. Thus, an appropriate set of attributes and parameters for each state needs to be established for each activity in collaboration between the members of the interdisciplinary design team for the DBAs. In early stages of development, finer grained and lower level events may need to be captured in the data structure so that patterns among them can be further explored to determine precisely which actions and system state variables are needed to produce the required evidence (e.g., DiCerbo et al., 2015; Halverson & Owen, 2014; Roberts, Chung, & Parks, 2016). The subsequent operational version of the data structure may then need to contain only higher-level events computed on the fly if the required computation is straightforward to calculate.

The second type of test taker-triggered event involves activities that do not directly act on the DBA elements designed for assessment, such as clicks on the screen and typing on the keyboard. These may be of use in developing analytics around test takers’ engagement, motivation, and “gaming the system” behaviors of (Baker, Corbett, Koedinger, & Wagner, 2004).


In certain situations, this type of events may provide direct evidence of new constructs related to the process of the creation of the outcomes. For example, the keystroke events from the process of writing essays may be used to measure constructs related to writing fluency (Deane, 2014).

In addition to the test taker-triggered events, some system-triggered events should also be defined in a data specification. Some examples include events generated by the DBA itself, as in a change in a patient's condition occurring over time in a medical simulation independent of the test taker's actions; prespecified reports of the simulation system status, such as a “heartbeat” recording at regular intervals the values of a set of system-status variables that will be needed in subsequent processing to interpret test-taker actions (a set could include, for example, variables for simulation conditions such that certain configurations identify that “a particular dog is not barking;” Mislevy, 2018); and the “result” of a test-taker action can be a pointer to an accompanying more complex object that is itself a work product, such as an audio file or a graphics file containing a figure the test taker drew on a pressure-sensitive tablet.

Implementing the above guidelines in practice usually requires a synergy of different expertise, such as learning science, data science, psychometrics, and software development. When multiple actors are involved, it is always necessary to establish a set of operational procedures to specify the workflow and needed expertise in each step. Table 1 shows an example operational procedure developed at ETS to coordinate the process. Readers are encouraged to develop their own procedures in commensurate with their needs and available resources.



Table 1. *Example of operational procedures that specify the steps and expertise in the instantiation of a data model in practice*

X: participant      XX: coordinator		Roles				
		Learning Scientist/ Assessment Developer	Psychometrician	Programmer	Data Scientist	Data analyst
 Timeline	Step 1: All-party meeting to brief the procedures	X	X	X	XX	X
	Step 2: Specification of "Q matrix" for constructs and evidences	X				
	Step 3: Specification of additional evidence for psychometric modeling need		X			
	Step 4: Translate the evidences into operable task codes			X	X	XX
	Step 5: Design log file structure and schema				X	
	Step 6: Implement the evidence logging into the log file and tryout			X	X	XX
	Step 7: Log file parsing and evidence extraction based on the tryout data				X	
	Step 8: QA the log files to check whether the recovered evidence is sufficient. If not, restart from Step 6. If everything is good, proceed to next step	X	X		X	XX
	Step 9: Finalize the data reduction pipeline and QA procedure				XX	X
	Step 10: Implement the data reduction				X	XX

A careful design of data should be an integrated part of the solutions to the challenges that arise from the complex data of DBAs, but it often does not receive due attention in assessment design. Guiding the data design with evidence identification in mind, e.g., through an EICDD process as described above or similar ones, could help avoid many complications in later stages of evidence identification. Hao and Mislevy (2018) introduced the term *Evidence Trace File* (ETF) to denote a data file that went through an EICDD (or similar) process, as distinguished from the common term *log file* that is widely used in VPAs for files that capture process events but have not gone through a purposive design procedure particular to the evidentiary needs and intended purposes of assessment. In a broader sense, we believe that a clear specification of what data are collected and how they are used can be critical for establishing the credibility and legitimacy of the assessment and mitigating test takers' concerns regarding fairness and privacy.

### 8.3. Data Techniques

Roughly speaking, there are two types of data processing, batch and stream processing. Batch processing is dealing with data that have already been collected for some time while stream processing is analyzing a continuous data stream generated or loaded in real time. In practice, the choice of batch or stream processing is dependent on the intended use and data size. For example, if the intended use depends on the cumulative information that are not generated at the same time, batch processing is generally considered. If the intended use is for providing immediate feedback based on the data up to the current time point, stream processing may be better suited. If the data size is very big and cannot be comfortably loaded into a modern computer's memory, one can turn the data into a data stream (e.g., many small chunks) and then apply stream processing techniques. Data from assessments, especially those from high-stake assessments, are usually processed in batch as many psychometric procedures, such as equating and item response theory (IRT) modeling, are based on data that are not available simultaneously. But for data from adaptive learning or assessment systems that require real-time feedback or adaption, stream processing is obviously needed.

Different data infrastructure and ecosystems have been developed for different types of data processing. For example, in the big data paradigm, the ecosystems such as Apache Hadoop (<https://hadoop.apache.org/>) are primarily intended for batch processing, while ecosystems such as Apache Storm (<http://storm.apache.org/>) and Apache Kafka (<https://kafka.apache.org/>) are intended for stream processing. Some other ecosystems such as Apache Spark (<https://spark.apache.org/>) and Apache Flink (<https://flink.apache.org/>) are capable of handling both batch and stream processing. The details of these infrastructures are beyond the scope of this section and we refer the interested readers to the relevant tutorials on the above websites.

In the following subsections, we will introduce some basic data techniques for handling complex process data from some example DBAs. We will primarily focus on the batch processing, but also provide a lightweight introduction to stream processing using Python generator functions.

### *8.3.1 Python Ecosystem for Data Science*

Traditional psychometric analysis usually starts with data in the form of a response/score table, with a convention of rows representing test takers and columns representing items. Each cell of the table is the response/score of a test taker with respect to an item as specified by the corresponding row and column. However, the process data from DBAs are generally not a regular response/score table. Instead, they often appear as a sequence of time-stamped events with different attributes and are either saved into standalone files or as entries in a database. These data can be stored either as an unstructured stack of strings (often due to some poor data engineering designs) or in a structured format such as XML or JSON. To identify evidence from these types of data, one needs to be able to manipulate the data at any granularity, which usually cannot be accomplished by GUI-based software (e.g., Excel). Programming using certain programming/scripting languages is generally necessary.

Python programming language is becoming popular among data scientists and software developers (Robinson, 2017). Differing from the R programming language primarily used for statistical analysis, Python supports a much wider range of applications, including website development, desktop software development, scientific computation, and even game development. Python comes with some basic functionality but relies heavily on external packages to perform almost all numerical computations. After a natural selection process over the last decade, a set of packages that provide fundamental computing capabilities has been

widely accepted in the Python community. In Table 2, we list some of these core packages that are maintained by the Python community.

In addition to these backbone packages, the front end of the Python interpreter is also evolving. Probably the most user-friendly and interactive interface for Python at the moment is the Jupyter Notebook (Kluyver et al., 2016), or its new variant Jupyter Lab, which provides a browser-based interface for interactive data analysis. The easiest way to get Python, the core packages, and Jupyter Notebook installed on computers is through the Anaconda suite (<https://www.anaconda.com/download>). Anaconda can be viewed as a management tool for Python packages and virtual environments, through which one can easily install/update Python packages from a well-maintained repository and create as well as manage virtual environments to run different versions of Python and packages. We encourage readers to explore the corresponding website to get familiarized with installing the Python systems through anaconda and will assume in our subsequent discussion they have been installed.

Table 2. *Some Core Packages for Numerical Computation with Python*

<b><i>Package</i></b>	<b><i>Descriptions</i></b>	<b><i>Website</i></b>
Numpy	Package for enabling numerical computation.	<a href="https://www.numpy.org">https://www.numpy.org</a>
Scipy	Package for scientific computing based on numpy.	<a href="https://www.scipy.org">https://www.scipy.org</a>
Pandas	Package for the data frame and tabular data manipulation	<a href="https://pandas.pydata.org">https://pandas.pydata.org</a>
Scikit-learn	Package for machine learning.	<a href="https://scikit-learn.org">https://scikit-learn.org</a>
NLTK	Package for Natural Language Processing.	<a href="https://www.nltk.org">https://www.nltk.org</a>
SpaCy	Package for Natural Language Processing.	<a href="https://spacy.io">https://spacy.io</a>
Matplotlib	Package for plotting and visualization.	<a href="https://matplotlib.org">https://matplotlib.org</a>
Seaborn	Package for plotting and visualization based on matplotlib.	<a href="https://seaborn.pydata.org">https://seaborn.pydata.org</a>
Plotly	Package for interactive visualization.	<a href="https://plotly.com">https://plotly.com</a>

### 8.3.2 *Tabular and Sequential Data*

To work with data, we need to have a certain data structure (e.g., an array) to hold the data and then apply different operations on it. Tabular data is probably the most familiar data structure to psychometricians. As we discussed in section 8.2, data table is the typical

instantiation of a relational data model that is well-gearred for representing the inter-relationship among the data records. In Python, the Pandas package furnishes a data frame for holding tabular data. Pandas also comes with many convenient functions to facilitate the computation and visualization based on the data frame (more details can be found from the Pandas documentation website at <https://pandas.pydata.org/pandas-docs/stable/>). On the other hand, the response process data from a DBA session are usually a sequence of time-stamped events with different attributes, which are usually stored as XML or JSON strings under a document data model.

In practical analyses, researchers often want to extract different features from the sequential data of each session and then analyze them across the sessions. As such, it is sometimes more convenient and efficient to restructure the sequential data from a session into a tabular data frame. In particular, indexing the rows of the data frame as the sequence number of events and the columns as different attributes of the events leads to a natural mapping from sequential data to tabular data. One can then stack the converted tabular data from different sessions together to create a tabular representation of many sessions. It is worth noting that when doing such a mapping, there is a slightly increased redundancy, but this overhead is usually outweighed by increased convenience. Table 3 shows a schematic of a representation of sequential data in a tabular format.

Table 3. *A schematic of the data frame converted from sequential data.*

<i>Sequence Number</i>	<i>Event Time</i>	<i>Test Taker ID</i>	<i>Session ID</i>	<i>Event Name</i>	<i>Attribute 1</i>	<i>Attribute 2</i>	<i>...</i>
1	Timestamp 1	A	1	...	...	...	...
2	Timestamp 2	A	1	...	...	...	...
3	...	A	1	...	...	...	...
4	...	A	1	...	...	...	...
5	...	B	2	...	...	...	...
6	...	B	2	...	...	...	...
7	...	B	2	...	...	...	...

Psychometricians have a long history of dealing with tabular data and have developed a wide range of techniques and capabilities. The tabular data psychometricians usually handle are organized in wide format (e.g., each row is an observation/person, and each column is a feature/score). The tabular data converted from sequential data as described above are organized as a combination of wide format and long format (e.g., each row is an observation belonging to a specific category) due to the sequential nature of the data. Converting sequential data into appropriate tabular data frames is a crucial step in recasting the new data challenges into representations for which psychometricians are familiar. In the subsequent subsections, we will show specific examples of how to convert the sequential data, either structured or unstructured, into a tabular data frame for further analysis.

### 8.3.3 Parsing Unstructured Process Data

Some real-world sequential data are presented in unstructured format, though most of which are usually the results of inappropriate data engineering. Figure 2 shows a snippet of an unstructured log file from a simulation-based task. The data are stored in a plain text file named *unstructured\_example\_log.txt*. Each line of the file has a timestamp and some description about what happened at that time point.

```
[5/15/2013 2:17:26 PM] Session Start
[5/15/2013 2:17:26 PM] Leaving sequence: loadXML, moving forward.
[5/15/2013 2:17:30 PM] Player submitted name: Carl
[5/15/2013 2:17:30 PM] Leaving sequence: InputNameScreen, moving forward.
[5/15/2013 2:17:31 PM] Player submitted name: Carl
[5/15/2013 2:17:31 PM] Leaving sequence: startScreen, moving forward.
[5/15/2013 2:17:50 PM] Player submitted name: Carl
[5/15/2013 2:17:55 PM] Leaving sequence: slide2, moving forward.
[5/15/2013 2:17:55 PM] Player submitted name: Carl
[5/15/2013 2:17:55 PM] Leaving sequence: slide2b, moving forward.
[5/15/2013 2:18:34 PM] Player submitted name: Carl
[5/15/2013 2:18:34 PM] Leaving sequence: slide2c, moving forward.
[5/15/2013 2:20:09 PM] Player submitted name: Carl
[5/15/2013 2:20:09 PM] Leaving sequence: slide3, moving forward.
[5/15/2013 2:20:13 PM] Player submitted name: Carl
[5/15/2013 2:20:13 PM] Leaving sequence: slide4, moving forward.
```

Figure 2. A snippet of an unstructured log from a simulation-based assessment task.

Python provides a set of functionalities to read in and manipulate the fields for such a plain text file. We will show how to parse and rearrange the fields of the sequential data into a tabular data frame. Figure 3 shows how to load the needed packages and read in the text file as a list of strings. The input cell In[1] shows how to load the needed packages. The input cell In[2] shows how to open the text file and read the lines in the file as a list of strings in Python and store them into a <sup>3</sup> *txt*. The output cell Out[3] prints out the list *txt* and the output cell Out[4] shows how to reference the first character of the first element in the list variable *txt*. In this way, one can reference or index any character in the text file and can, therefore, manipulate the data at any desired granularity.

---

<sup>3</sup>Python list is specific variable type in the form of [a, b, c, ....].

### Step 1. Loading the needed packages

```
In [1]: 1 import pandas as pd
        2 import numpy as np
        3 from datetime import datetime
```

### Step 2. Open the unstructured data file and read it into a python list

```
In [2]: 1 with open('unstructured_example_log.txt') as f:
        2     txt = f.readlines()
```

```
In [3]: 1 txt
```

```
Out[3]: ['[5/15/2013 2:17:26 PM] Session Start\n',
         '[5/15/2013 2:17:26 PM] Leaving sequence: loadXML, moving forward.\n',
         '[5/15/2013 2:17:30 PM] Player submitted name: Carl\n',
         '[5/15/2013 2:17:30 PM] Leaving sequence: InputNameScreen, moving forward.\n',
         '[5/15/2013 2:17:31 PM] Player submitted name: Carl\n',
         '[5/15/2013 2:17:31 PM] Leaving sequence: startScreen, moving forward.\n',
         '[5/15/2013 2:17:50 PM] Player submitted name: Carl\n',
         '[5/15/2013 2:17:50 PM] Leaving sequence: slide2, moving forward.\n',
         '[5/15/2013 2:17:55 PM] Player submitted name: Carl\n',
         '[5/15/2013 2:17:55 PM] Leaving sequence: slide2b, moving forward.\n',
         '[5/15/2013 2:18:34 PM] Player submitted name: Carl\n',
         '[5/15/2013 2:18:34 PM] Leaving sequence: slide2c, moving forward.\n',
         '[5/15/2013 2:20:09 PM] Player submitted name: Carl\n',
         '[5/15/2013 2:20:09 PM] Leaving sequence: slide3, moving forward.\n',
         '[5/15/2013 2:20:13 PM] Player submitted name: Carl\n',
         '[5/15/2013 2:20:13 PM] Leaving sequence: slide4, moving forward.\n']
```

```
In [4]: 1 txt[0][0]
```

```
Out[4]: '['
```

Figure 3. *Loading the needed package and read in a text file as a list of strings.*

Python provides powerful functionalities for processing strings. In the output cell Out[3] of Figure 3, each element of the list ends with a line break character `\n`, which is usually



irrelevant to further analysis. Figure 4 shows how to remove them and get a clean list of strings.

**Step 3. Clean each line to strip off the \n**

```
In [5]: 1 txt = [t.strip() for t in txt]

In [6]: 1 txt

Out[6]: ['[5/15/2013 2:17:26 PM] Session Start',
'[5/15/2013 2:17:26 PM] Leaving sequence: loadXML, moving forward.',
'[5/15/2013 2:17:30 PM] Player submitted name: Carl',
'[5/15/2013 2:17:30 PM] Leaving sequence: InputNameScreen, moving forward.',
'[5/15/2013 2:17:31 PM] Player submitted name: Carl',
'[5/15/2013 2:17:31 PM] Leaving sequence: startScreen, moving forward.',
'[5/15/2013 2:17:50 PM] Player submitted name: Carl',
'[5/15/2013 2:17:50 PM] Leaving sequence: slide2, moving forward.',
'[5/15/2013 2:17:55 PM] Player submitted name: Carl',
'[5/15/2013 2:17:55 PM] Leaving sequence: slide2b, moving forward.',
'[5/15/2013 2:18:34 PM] Player submitted name: Carl',
'[5/15/2013 2:18:34 PM] Leaving sequence: slide2c, moving forward.',
'[5/15/2013 2:20:09 PM] Player submitted name: Carl',
'[5/15/2013 2:20:09 PM] Leaving sequence: slide3, moving forward.',
'[5/15/2013 2:20:13 PM] Player submitted name: Carl',
'[5/15/2013 2:20:13 PM] Leaving sequence: slide4, moving forward.']
```

Figure 4. *Python code example of cleaning strings in Python.*

Readers may have noticed that the time in the data file is in a string format that does not allow numerical computation directly. Python has a DateTime object to store date and time information and allows numerical operations. Figure 5 shows the code snippet for extracting, cleaning, and converting the date and time strings to Python DateTime objects, and then calculates the time difference between two entries in the unit of second.

#### Step 4. Sepate the time stamp and convert it to standard Python datetime object

```
In [7]: 1 s0 = txt[0].split(' ')[0].strip('[')
        2 s10 = txt[10].split(' ')[0].strip('[')

In [8]: 1 s0
Out[8]: '5/15/2013 2:17:26 PM'

In [9]: 1 s10
Out[9]: '5/15/2013 2:18:34 PM'

In [10]: 1 dtfmt = '%m/%d/%Y %I:%M:%S %p'  # %H -> 24 hours, %I-> 12 hours
        2 t0 = datetime.strptime(s0, dtfmt)
        3 t10 = datetime.strptime(s10, dtfmt)

In [11]: 1 t0
Out[11]: datetime.datetime(2013, 5, 15, 14, 17, 26)

In [12]: 1 t10
Out[12]: datetime.datetime(2013, 5, 15, 14, 18, 34)

In [13]: 1 (t10-t0).seconds
Out[13]: 68
```

Figure 5. Code snippet for converting date time string to Python DateTime object.

Next, in Figure 6, we show the codes that can restructure the data into a Pandas data frame. In the input cell In[14], we define a function where each line of the text file is divided into three chunks and stored into three lists, session\_time, event\_name, and event\_attribute. The lines 18 and 19 convert the lists into a Pandas data frame and the corresponding output cell Out[15] shows the resulting data frame. At this point, one can apply additional operations on the in-memory data frame to extract the desired evidence. We should note that the codes snippets here are motivated by logical clarity rather than execution efficiency. They are not necessarily the most efficient way to accomplish the goals. We encourage readers to explore different ways to complete the above exercise.

### Step 5. Restructure the information into a data frame

```
In [14]: 1 # define a function to combine all the above steps and turn the results into a pandas Data Frame
2
3 def generate_data_frame(txt):
4     session_time = []
5     event_name = []
6     event_attribute = []
7     dtfmt = '%m/%d/%Y %I:%M:%S %p'
8     for line in txt:
9         sl=line.split(' ')[0].strip(' ')
10        dt = datetime.strptime(sl, dtfmt)
11        session_time.append(dt)
12        s= line.split(' ')[1].strip().split(':')
13        event_name.append(s[0])
14        if len(s) == 2:
15            event_attribute.append(s[1].lstrip())
16        else:
17            event_attribute.append(np.nan)
18    df = pd.DataFrame([session_time,event_name,event_attribute]).T
19    df.columns=['session_time','event_name', 'event_attribute']
20    return df
```

```
In [15]: 1 generate_data_frame(txt)
```

```
Out[15]:
```

	session_time	event_name	event_attribute
0	2013-05-15 14:17:26	Session Start	NaN
1	2013-05-15 14:17:26	Leaving sequence	loadXML, moving forward.
2	2013-05-15 14:17:30	Player submitted name	Carl
3	2013-05-15 14:17:30	Leaving sequence	InputNameScreen, moving forward.
4	2013-05-15 14:17:31	Player submitted name	Carl
5	2013-05-15 14:17:31	Leaving sequence	startScreen, moving forward.
6	2013-05-15 14:17:50	Player submitted name	Carl
7	2013-05-15 14:17:50	Leaving sequence	slide2, moving forward.
8	2013-05-15 14:17:55	Player submitted name	Carl
9	2013-05-15 14:17:55	Leaving sequence	slide2b, moving forward.
10	2013-05-15 14:18:34	Player submitted name	Carl
11	2013-05-15 14:18:34	Leaving sequence	slide2c, moving forward.
12	2013-05-15 14:20:09	Player submitted name	Carl
13	2013-05-15 14:20:09	Leaving sequence	slide3, moving forward.
14	2013-05-15 14:20:13	Player submitted name	Carl
15	2013-05-15 14:20:13	Leaving sequence	slide4, moving forward.

Figure 6. Codes snippet for converting unstructured data into a Pandas data frame.

### 8.3.4 Parsing Structured Process Data

The data from most well-designed DBAs are structured, though not necessarily well-structured. JSON and XML are two most popular formats used to store such data, and the specifications are instantiated as JSON or XML schema. The schema can be used for compliance check against the incoming data, providing basic data quality assurance to ensure data integrity. There are several Python packages to assist the parsing of structured data in JSON and XML format. In this subsection, we use an example to show how to parse and restructure a structured

data file in JSON format into a Pandas data frame. The same example data used in the previous subsection have been cast into a structured JSON file named *structured\_example\_log.json*, as shown in Figure 8.

```
{
  "data": [
    {
      "session_time": "2013-05-15 14:17:26",
      "event_name": "Session Start",
      "event_attribute": "NaN"
    },
    {
      "session_time": "2013-05-15 14:17:26",
      "event_name": "Leaving sequence",
      "event_attribute": "loadXML, moving forward."
    },
    {
      "session_time": "2013-05-15 14:17:30",
      "event_name": "Player submitted name",
      "event_attribute": "Carl"
    },
    {
      "session_time": "2013-05-15 14:17:30",
      "event_name": "Leaving sequence",
      "event_attribute": "InputNameScreen, moving forward."
    },
    {
      "session_time": "2013-05-15 14:17:31",
      "event_name": "Player submitted name",
      "event_attribute": "Carl"
    },
    {
      "session_time": "2013-05-15 14:17:31",
      "event_name": "Leaving sequence",
      "event_attribute": "startScreen, moving forward."
    },
    {
      "session_time": "2013-05-15 14:17:50",
      "event_name": "Player submitted name",
      "event_attribute": "Carl"
    },
    {
      "session_time": "2013-05-15 14:17:50",
      "event_name": "Leaving sequence",
      "event_attribute": "slide2, moving forward."
    },
    {
      "session_time": "2013-05-15 14:17:55",
      "event_name": "Player submitted name",
      "event_attribute": "Carl"
    },
    {
      "session_time": "2013-05-15 14:17:55",
      "event_name": "Leaving sequence",
      "event_attribute": "slide2b, moving forward."
    },
    {
      "session_time": "2013-05-15 14:18:34",
      "event_name": "Player submitted name",
      "event_attribute": "Carl"
    },
    {
      "session_time": "2013-05-15 14:18:34",
      "event_name": "Leaving sequence",
      "event_attribute": "slide2c, moving forward."
    },
    {
      "session_time": "2013-05-15 14:20:09",
      "event_name": "Player submitted name",
      "event_attribute": "Carl"
    },
    {
      "session_time": "2013-05-15 14:20:09",
      "event_name": "Leaving sequence",
      "event_attribute": "slide3, moving forward."
    },
    {
      "session_time": "2013-05-15 14:20:13",
      "event_name": "Player submitted name",
      "event_attribute": "Carl"
    },
    {
      "session_time": "2013-05-15 14:20:13",
      "event_name": "Leaving sequence",
      "event_attribute": "slide4, moving forward."
    }
  ]
}
```

Figure 8. A snippet of a structured log in JSON format from a simulation-based assessment task.

As with parsing unstructured data, we must first load some needed packages. In Python, there is a package called “json” that can read in a JSON file as a Python dictionary<sup>4</sup>. The corresponding code snippet is shown in Figure 8. The input cell In[1] loads the needed packages. The input cell In[2] opens the JSON file and read it in as a python dictionary named *txt*. The output cell Out[3] prints out the *txt* dictionary. Pandas provides a convenient way for converting a list of dictionaries into a data frame and the codes are shown in Figure 9. These are examples based on the file in JSON format but similar methods exist for processing XML files too. We include the corresponding notebook in the appendix.

<sup>4</sup> Python dictionary is a specific variable type in the form of key-value pairs as {“key1”: “value1”, “key2”: “value2”, ...}.

## Step 1. Loading the needed packages

```
In [1]: 1 import pandas as pd
        2 import json
        3 from datetime import datetime
```

## Step 2. Open the unstructured data file and read it into a Python dictionary

```
In [2]: 1 with open('structured_example_log.json') as f:
        2     txt = json.load(f)
```

```
In [3]: 1 txt
```

```
Out[3]: {'data': [{'session_time': '2013-05-15 14:17:26',
                    'event_name': 'Session Start',
                    'event_attribute': 'NaN'},
                 {'session_time': '2013-05-15 14:17:26',
                    'event_name': 'Leaving sequence',
                    'event_attribute': 'loadXML, moving forward.'},
                 {'session_time': '2013-05-15 14:17:30',
                    'event_name': 'Player submitted name',
                    'event_attribute': 'Carl'},
                 {'session_time': '2013-05-15 14:17:30',
                    'event_name': 'Leaving sequence',
                    'event_attribute': 'InputNameScreen, moving forward.'},
                 {'session_time': '2013-05-15 14:17:31',
                    'event_name': 'Player submitted name',
                    'event_attribute': 'Carl'},
                 {'session_time': '2013-05-15 14:17:31',
                    'event_name': 'Leaving sequence',
                    'event_attribute': 'startScreen, moving forward.'},
                 {'session_time': '2013-05-15 14:17:50',
                    'event_name': 'Player submitted name',
                    'event_attribute': 'Carl'},
                 {'session_time': '2013-05-15 14:17:50',
                    'event_name': 'Leaving sequence',
                    'event_attribute': 'slide2, moving forward.'},
                 {'session_time': '2013-05-15 14:17:55',
                    'event_name': 'Player submitted name',
                    'event_attribute': 'Carl'},
                 {'session_time': '2013-05-15 14:17:55',
                    'event_name': 'Leaving sequence',
                    'event_attribute': 'slide2b, moving forward.'},
                 {'session_time': '2013-05-15 14:18:34',
                    'event_name': 'Player submitted name',
                    'event_attribute': 'Carl'},
                 {'session_time': '2013-05-15 14:18:34',
                    'event_name': 'Leaving sequence',
                    'event_attribute': 'slide2c, moving forward.'},
                 {'session_time': '2013-05-15 14:20:09',
                    'event_name': 'Player submitted name',
                    'event_attribute': 'Carl'},
                 {'session_time': '2013-05-15 14:20:09',
                    'event_name': 'Leaving sequence',
                    'event_attribute': 'slide3, moving forward.'},
                 {'session_time': '2013-05-15 14:20:13',
                    'event_name': 'Player submitted name',
                    'event_attribute': 'Carl'},
                 {'session_time': '2013-05-15 14:20:13',
                    'event_name': 'Leaving sequence',
                    'event_attribute': 'slide4, moving forward.'}]}
```

Figure 8. Code snippet for loading the JSON package and read in the JSON file as a Python

dictionary.

### Step 3. Convert a list of dictionaries to data frame

```
In [4]: 1 pd.DataFrame(txt.get('data'))
```

```
Out[4]:
```

	event_attribute	event_name	session_time
0	NaN	Session Start	2013-05-15 14:17:26
1	loadXML, moving forward.	Leaving sequence	2013-05-15 14:17:26
2	Carl	Player submitted name	2013-05-15 14:17:30
3	InputNameScreen, moving forward.	Leaving sequence	2013-05-15 14:17:30
4	Carl	Player submitted name	2013-05-15 14:17:31
5	startScreen, moving forward.	Leaving sequence	2013-05-15 14:17:31
6	Carl	Player submitted name	2013-05-15 14:17:50
7	slide2, moving forward.	Leaving sequence	2013-05-15 14:17:50
8	Carl	Player submitted name	2013-05-15 14:17:55
9	slide2b, moving forward.	Leaving sequence	2013-05-15 14:17:55
10	Carl	Player submitted name	2013-05-15 14:18:34
11	slide2c, moving forward.	Leaving sequence	2013-05-15 14:18:34
12	Carl	Player submitted name	2013-05-15 14:20:09
13	slide3, moving forward.	Leaving sequence	2013-05-15 14:20:09
14	Carl	Player submitted name	2013-05-15 14:20:13
15	slide4, moving forward.	Leaving sequence	2013-05-15 14:20:13

Figure 9. *Python code to convert a list of dictionaries to a data frame.*

#### 8.3.5 Microbatch and Stream Processing via Generator Function

So far, we have introduced some basic batch processing techniques for restructuring raw sequential data into a manageable tabular data frame for further analysis. The methods generally work well when the data size is not very large so that they can be read into the memory of a computer and all the data are available before performing the processing. However, if the data size is very large, more than the memory of a typical modern computer, or if new data keep added continuously, we need to consider different strategies. In this subsection, we introduce

how to handle very large log files and growing log files using the generator function in Python through microbatch processing and stream processing respectively.

To be specific, let's consider a hypothetical log file similar to the unstructured log file in section 8.3.3, but has one trillion lines instead of sixteen. If one runs the scripts as shown in Figure 2, it is likely to get into issues after executing the In[2] cell as the function

`f.readlines()` attempts to read all lines in this one trillion-line file into the computer memory.

A more favorable strategy may be processing the data in chunks of smaller size when the processing function is called and then release the corresponding computer memory after processing each chunk. Such a strategy is referred to as lazy evaluation in programming language theory (Reynolds, 2009). In Python, generator functions provide an elegant way to accomplish lazy evaluation. There are many nice tutorials about the generator function in Python (e.g., Beazley, 2018) and we refer readers to them for further details. In Figure 10, we show some code examples of using generator functions to parse and restructure the unstructured log used in section 8.3.3 to give readers a sense of a microbatch processing pipeline for very large data files.

## Step 1. Loading the needed packages

```
In [1]: 1 import pandas as pd
2 import numpy as np
3 from datetime import datetime
4 from itertools import islice
5 import sys
```

## Step 2. Define functions

```
In [2]: 1 # define generator function to read in files in chunks
2 def read_chunks(file_obj, chunk_size):
3     while True:
4         lines = list(islice(file_obj, chunk_size))
5         if lines:
6             yield lines
7         else:
8             print('end of file')
9             break
```

## Step 3. Reload the previously defined function

```
In [3]: 1 # this function is the same as the previous one but we reload it here
2 def generate_data_frame(txt):
3     session_time = []
4     event_name = []
5     event_attribute = []
6     dtfmt = '%m/%d/%Y %I:%M:%S %p'
7     for line in txt:
8         s1=line.split(' ')[0].strip('[')
9         dt = datetime.strptime(s1, dtfmt)
10        session_time.append(dt)
11        s= line.split(' ')[1].strip().split(':')
12        event_name.append(s[0])
13        if len(s) == 2:
14            event_attribute.append(s[1].lstrip())
15        else:
16            event_attribute.append(np.nan)
17    df = pd.DataFrame([session_time,event_name,event_attribute]).T
18    df.columns=['session_time','event_name', 'event_attribute']
19    return df
```



#### Step 4. Open files and read data into a generator object

```
In [4]: 1 # open the file
        2 f = open('unstructured_example_log.txt','r')

In [5]: 1 # read file into chunks of 4 lines and create a generator object
        2 chunk_generator = read_chunks(f,4)

In [6]: 1 chunk_generator
Out[6]: <generator object read_chunks at 0x120687ad0>

In [7]: 1 # check the memory usage of the generator object in bytes
        2 sys.getsizeof(chunk_generator)
Out[7]: 128

In [8]: 1 # get the first chunk using next()
        2 next(chunk_generator)
Out[8]: ['[5/15/2013 2:17:26 PM] Session Start\n',
        '[5/15/2013 2:17:26 PM] Leaving sequence: loadXML, moving forward.\n',
        '[5/15/2013 2:17:30 PM] Player submitted name: Carl\n',
        '[5/15/2013 2:17:30 PM] Leaving sequence: InputNameScreen, moving forward.\n']

In [9]: 1 # create a new generator function called df
        2 df = (generate_data_frame(txt) for txt in chunk_generator)

In [10]: 1 df
Out[10]: <generator object <genexpr> at 0x120687850>

In [11]: 1 # check the memory usage of the generator object in bytes
        2 sys.getsizeof(df)
Out[11]: 128

In [12]: 1 # get the first object in dfnext(df)
        2 next(df)
Out[12]:
```

	session_time	event_name	event_attribute
0	2013-05-15 14:17:31	Player submitted name	Carl
1	2013-05-15 14:17:31	Leaving sequence	startScreen, moving forward.
2	2013-05-15 14:17:50	Player submitted name	Carl
3	2013-05-15 14:17:50	Leaving sequence	slide2, moving forward.

```
In [13]: 1 f.close()
```

Figure 10. Code snippet of the stream processing pipeline using generator functions for the unstructured data file.

The input cell In[9] formally defines a generator function `df` that is essentially a set of procedures to read in a chunk of the unstructured data file and convert it to a data frame. The input cell In[12] uses “`next`” function to trigger the `df` to take action. One can repeatedly apply the `next` function or use a loop to exhaust all the lines in the log file without leading to a memory overflow error, which is desirable for designing a scalable processing pipeline. It is worth noting that the generator-based stream processing is slightly slower compared to a batch processing for smaller data file where the computer memory is not a concern. In practice, readers need to make their own decisions for choosing which processing pipeline, based on the data size, intended use, and project priorities.

The above method for continuously processing a very large file by dividing it into smaller batches is known as microbatch processing. It is something in between of batch processing and stream processing. A typical stream processing, however, usually deals with event data that keep coming, leading to files with potentially “infinite” size. For example, in an online shopping website, such as Amazon.com, customers from all over the world make various requests (e.g., put things into their shopping carts, make payments, and so on) from time to time without a clear ending point. Each request requires a near real time processing that leads to actions to address the customer’s needs. In practice, software infrastructure for stream data is implemented and an application programming interface (API) is often provided to allow users to interact with the data stream. An example of such implementations is the twitter data stream API and readers can get more details from the corresponding website (<https://help.twitter.com/en/rules-and-policies/twitter-api>). In the following, we show how to handle a growing log file through a simplified example. First, we create a growing log file by adding a new line to a log file

(stream\_data.txt) using a python code (stream\_data\_generation.py) every two seconds. The python codes and part of the log file are shown in Figure 11.

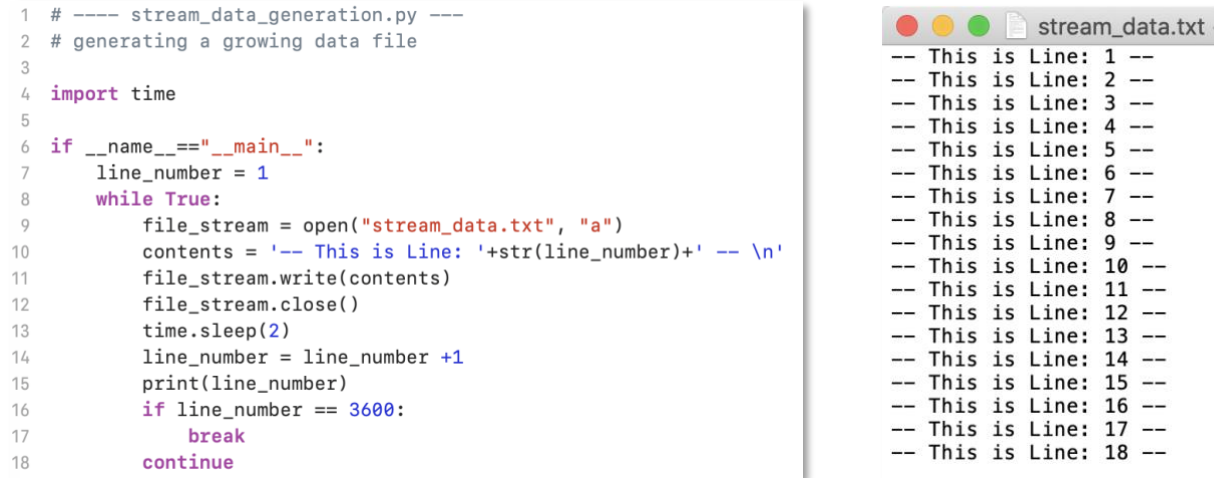
The figure consists of two side-by-side screenshots. The left screenshot shows a Python script named 'stream\_data\_generation.py' with line numbers 1 through 18. The script uses a while loop to write lines to a file named 'stream\_data.txt' every two seconds, up to line 3600. The right screenshot shows a window titled 'stream\_data.txt' displaying the first 18 lines of the log file, each starting with '-- This is Line: ' followed by the line number and ending with '-- \n'.

Figure 11. *Python codes (left) that add a new line of text to a log file every two seconds and the resulting log file (right).*

Next, we create a generator function to read the growing log file, either from the beginning or from the current ending point, and then iterate through the (potentially infinite) lines one by one. One can perform any filtering or processing on each line while iterating through the lines. The codes are shown in Figure 12.

### Step 1. Loading the needed packages

```
In [1]: 1 import time
```

### Step 2. Define a generator function to get the latest log entries

```
In [2]: 1 # Example function modified from David Beazley (https://www.dabeaz.com/generators/Generators.pdf)
2
3 def follow(thefile,start='head'):
4     if start == 'head':
5         thefile.seek(0,0) # start from the beginning of the file
6     if start == 'tail':
7         thefile.seek(0,2) # start from the current ending point of the file
8     while True:
9         line = thefile.readline()
10        if not line:
11            time.sleep(0.1) # Sleep briefly
12            continue
13        yield line
```

### Step 3. Working on the lines of the growing file

```
In [9]: 1 # open the log file
2 logfile = open('../data/stream_data.txt')
```

```
In [10]: 1 # follow all lines in the growing file
2 line_all = follow(logfile,start='head')
```

```
In [5]: 1 # get out the lines sequentially
2 next(line_all)
```

```
Out[5]: '-- This is Line: 1 -- \n'
```

```
In [6]: 1 # follow the new lines starting from the execution of the function
2 line_latest = follow(logfile,start='tail')
```

```
In [7]: 1 next(line_latest)
```

```
Out[7]: '-- This is Line: 8 -- \n'
```

```
In [8]: 1 # close the log file
2 logfile.close()
```

Figure 12. A Jupyter Notebook showing codes to handle a growing log file.

The above example is highly simplified, but it captures the essential feature of stream processing via generator functions in Python. In real-world applications, different software infrastructures are used for different type of data and processing. Users can directly use the corresponding APIs instead of programming from scratch using generators. For example, Apache Spark (<https://spark.apache.org/>) is designed for dealing with microbatch processing while Apache Kafka (<https://kafka.apache.org/>) and Flink (<https://flink.apache.org/>) are designed

for handling stream processing. They all provide high-level APIs to massage the data stream and we recommend readers to consult the corresponding documentations for more details.

## **8.4 Methodological Strategies for Making Sense of Process Data**

Most familiar psychometric methodologies, such as IRT and classical test theory (CTT), have been developed for simpler and regular assessment data that can be represented as a response/score table (Chapter 6). As argued in Chapter 3, however, often these methods cannot be applied, in their basic forms, to the more complex process data arising from DBAs. As such, new methodological strategies are needed to guide the sensible use of process data to support assessments. We envision the uses of process data from DBAs can be put into three broad categories and discuss the corresponding methodological strategies for each of them in the follows.

### *8.4.1 Process Data as Augmenting Standard Assessments*

DBAs can be in varying levels of complexity. The simplest kind of DBA is an assessment that is very similar to a familiar paper-based assessment but delivered in a digital environment. Usually, the intended use and method of scoring are the same as they would be in a nondigital form, based on designated features of final products such as correctness of multiple-choice responses or holistic or analytic scores of constructed verbal responses. The digital environment, however, makes it possible to capture more detailed aspects of the performances, such as timestamped individual keystrokes, mouse clicks to maneuver among parts of the stimulus materials, changed answer choices, moves from task to task, and deletions, additions, and movements of text elements in verbal responses. This subsection addresses the use of process data to improve or modestly extend traditional forms of testing and construct definition—that is, the enacted construct is operationally defined in terms of traditional aspects of performance, such

as correct selections or ratings of final written responses. Even when only final products and scoring methods are required for operational use; however, the augmented data hold much information for other purposes, notably examining data quality, improving tasks, and investigating validity (Ercikan & Pellegrino, 2017; Zumbo & Hubley, 2017). Seeing exactly how test takers have interacted with the interface adds insight into thinking processes for all students, information that was formerly available on a smaller scale through think-aloud studies, human observers in real-time, or time-consuming analyses of video captures.

The most straightforward strategy in such situations is to turn the complex process data into a score table-like representation with rows representing students and columns representing feature variables characterizing aspects of the process (e.g., patterns), which we referred to as the feature table hereafter. The general approaches to identify patterns to form feature representations is often known as analytics, which is defined as *the discovery, interpretation, and communication of meaningful patterns in data; and the process of applying those patterns towards effective decision making* (Wikipedia contributors, 2019). The feature table resembles the score table that is familiar to most psychometricians except that its elements cannot be simply interpreted as scores in psychometric sense<sup>5</sup>. The score table generally captures the information regarding the outcomes of the person-item interaction, while the feature table summarizes additional information about the process of the person-item interaction (e.g., response time). Generally speaking, the techniques for dealing with the score tables used in traditional psychometrics can also be applied to the feature table, which could lead to new item parameters in addition to the well-known ones, such as item difficulty and differentiability, based

---

<sup>5</sup> In some cases, the features can be considered as scores as well.

on the score table (e.g., regression parameters in an IRT model in which response time provides auxiliary information about test-takers' propensity to respond correctly; see Roskam, 1997).

This is not a new strategy, and psychometricians have been analyzing constructed response items using a similar strategy for a long time, though we may not think of it from this perspective. The text responses to a constructed response item can be viewed as a response process, e.g., a sequence of timestamped words, even though the timestamps are usually not explicitly present in paper-pencil tests. In this case, human raters with a scoring rubric will map the response sequence into scores. The response process data from DBAs are very similar to the word sequences from constructed responses except for the explicit timestamps. Computers and algorithms will replace human raters and scoring rubrics to map the response sequences to feature representations. In this regard, the research focus will be on the identification of appropriate algorithms or stochastic models that can map a sequence to a feature or map several subsequences to multiple features.

Identifying appropriate algorithms to turn response process data into meaningful features is more an art than science; some are discussed in other chapters of this volume. Suppose we have a DBA with 20 selected response items, and we capture the test takers' response process information in addition to the final responses. Based on the response process data, what features can we develop to characterize the process? There could be many from different perspectives. For example, the response time to each item, the number of visits to each item, the order of visiting items, and the transition from item to item, could all be process-based features, which may contain information relevant to the test taker's proficiency or the item properties. This additional information can help in detecting differences across demographic groups, improving estimates of test takers' proficiency, or item calibration by identifying aberrant responses or

vectors to remove from calibration. Many more complex algorithms and stochastic models have been explored in the literature to accomplish this goal, such as the social network analysis described in Chapter 13 and the edit distance approach (Hao, Shu, & von Davier, 2015). This methodological strategy is most suitable for DBAs consisting of relatively short response process data, though the analytic methods can also be applied to interactive simulation-based tasks as auxiliary data, even when further uses of process data as evidence for inferences about test takers' capabilities, as discussed in the following sections.

#### *8.4.2 Process Data as Evidence in Psychometric Models*

A second methodological strategy is directly relating the evidence from process data to constructs operationalized as latent variables in a psychometric measurement model, such as the cases discussed in Chapters 2 and 4. The model itself, at its highest layer, could be a familiar one such as IRT, cognitive diagnosis, or latent class analysis if the data suit their assumptions; it could be an extension of such models, such as Bayes nets (static or dynamic; see Chapter 3), hidden Markov processes, or learning models as used in simulation-based assessments or intelligent tutoring systems (Koedinger, Corbett, & Perfetti, 2012; VanLehn, 2008), or multivariate models used to synthesize evidence in multimodal assessments (Kahn, 2017). As discussed in Chapter 2, the process of moving from low-level process data to evidence entered into a latent-variable psychometric model may be carried out in multiple stages (also see Kerr, Andrews, & Mislevy, 2016; Khan, 2017; Mislevy, 2018).

The distinguishing feature of this strategy, however, is that the process data as used as evidence for the latent variables, and as such directly affects the *meaning* of those variables. That is, their operationalized meanings can extend beyond the kind of evidence in traditional forms of data, and directly express aspects of performance not directly captured in traditional



data, such as strategy choices, efficiency, and responses to changing situations. Process data can in these ways enrich, or in some cases even make possible, the assessment of constructs that lie beyond the scope of traditional assessment methods.

This strategy is aligned with ECD and is better suited for VPAs such as simulation-based and game-based assessments. Examples of its application with particular computational-psychometric methods appear in Chapters 2 and 4. As mentioned, this modeling strategy can be developed in conjunction with the more exploratory and auxiliary computational-psychometric analyses discussed in the previous section (DiCerbo et al., 2015).

#### *8.4.3 Process Data as Evidence in Cognitive Models*

The third methodological strategy is directly mapping the response process or selected features of it to an underlying cognitive model. Methods under this strategy directly map the events or event sequences in the response process to parameters of a cognitive processing model rather than to either intermediate feature representations or latent variables representing higher-level constructs. One example of this strategy is fitting production-rule models from information-processing cognitive psychology to a test taker's sequence of actions. Singley (1995) illustrates the idea with a model-tracing algorithm to infer students' goals and strategies from their step-by-step solutions of open-ended algebra problems. Another example is fitting a Markov decision process model to a test taker's sequence of actions, as functions of the test taker's goals, knowledge, and efficiency of actions. LaMar (2018) illustrates this model in the context of a game-based strategy assessment. Chapter 11 offers further discussion of this approach.

This strategy too can be combined with methods from the previous two subsections. The exploratory analyses discussed in Section 8.4.1 are useful whenever complex data are encountered, to better understand the nature of performance in the complex environment (and,

often, to cycle back among design and theorizing). When models such as the ones mentioned in this section are used to model performance *within tasks*, a psychometric model can be then built with higher-level constructs presumed to influence performance *across tasks*. Hierarchically, the within-task latent variables are modeled as probabilistically dependent on the across-task latent variables. This is a style of implementing the strategy of Section 8.4.2.

## 8.5 Summary

Advances in psychology have provided us with a better understanding of the nature of human capabilities, and how we acquire them and develop them in a complex adaptive sociocognitive system. Advances in technology enable us to design and implement digital environments in which students can interact with realistic environments and other students, in which we can elicit rich performances that should be able to provide evidence about those capabilities. What stands between our targeted inferences and students' rich performances? Design, data, and analyses. This chapter brings *data* to the foreground: how, in concert with psychology, technology, design, and computational-psychometric modeling, we need to think about data, if we are to realize the potential for a new generation of assessment awaits us.

Though we summarized three broad strategies towards modeling the process data from DBAs, readers should be aware that the process data from a real-world DBA often have some unique properties that may add additional complications to applying these strategies in practice. Four typical complications concerned with modeling real-world DBA data are as follows. First, response processes are usually subject to the specific task design, which can limit the generalizability of the algorithms/models across tasks. Second, an extended performance can be the result of multiple strategies in different parts of the performance, requiring different regimes that correspond to different stationary conditions for temporal modeling. Third, the mathematical

space spanned by the response sequence is usually very big and sparse—even sometimes the for the augmented files for multiple-choice tests, when timing, ordering, and revisions are captured—which makes it challenging even to identify a suitable statistical distribution to model randomness in the sequences. Fourth, typical response sequences to DBAs that are similar to traditional task forms are often short, which makes it challenging to estimate the parameters of stochastic models. We suggest that readers who work with real data have a thorough understanding of the data at hand before delving into applying different modeling strategies.

## References

- Beazley, D. (2018). Generator tricks for systems programmers, Retrieved September 2nd, 2019, from <https://www.dabeaz.com/generators/Generators.pdf>
- Deane, P. (2014). Using writing process and product features to assess writing quality and explore how those features relate to other literacy tasks. *ETS Research Report Series*, 2014(1), 1-23.
- DiCerbo, K., Bertling, M., Stephenson, S., Jia, Y., Mislevy, R.J., Bauer, M., & Jackson, T. (2015). The role of exploratory data analysis in the development of game-based assessments. In C.S. Loh, Y. Sheng, & D. Ifenthaler (Eds.), *Serious games analytics: Methodologies for performance measurement, assessment, and improvement* (pp. 319-342). New York: Springer.
- Ercikan, K., & Pellegrino, J. W. (2017). *Validation of score meaning for the next generation of assessments: The use of response processes*. New York: Taylor & Francis.

- Hao, J., & Mislevy, R. J. (2018). The evidence trace file: A data structure for virtual performance assessments informed by data analytics and evidence-centered design. *ETS Research Report Series*, 2018(1), 1-16.
- Hao, J., Smith III, L.L., Mislevy, R.J., & von Davier, A.A. (2014). Systems and methods for designing, parsing and mining of game log files. U.S. patent application # 14/527,591
- Hao, J., Smith, L., Mislevy, R., von Davier, A., & Bauer, M. (2016). Taming log files from game/simulation-based assessments: Data models and data analysis tools. *ETS Research Report Series*, 2016(1), 1–18.
- IMS Global Learning Consortium. (2015). Caliper analytics. Retrieved from the website of IMS Global Learning Consortium, <http://www.imsglobal.org/activity/caliper>.
- LaMar, M. M. (2018). Markov decision process measurement model. *Psychometrika*, 83(1), 67-88.
- Kerr, D., Andrews, J.J., & Mislevy, R.J. (2016). The In-task Assessment Framework for behavioral data. In A. Rupp & J. Leighton (Eds.), *Handbook of Cognition and Assessment* (pp. 472-507). Hoboken, NJ: Wiley-Blackwell.
- Kevan, J. M., & Ryan, P. R. (2016). Experience API: Flexible, decentralized and activity-centric data collection. *Technology, knowledge and learning*, 21(1), 143-149.
- Khan, S. M. (2017). Multimodal behavioral analytics in intelligent learning and assessment systems. In A.A. von Davier, M. Zhu, & P.C. Kyllonen (eds.), *Innovative assessment of collaboration* (pp. 173-184). Cham, Switzerland: Springer International.
- Kleppmann, M. (2017). *Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems*. Boston, MA: O'Reilly Media, Inc.

- Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B. E., Bussonnier, M., Frederic, J., ... & Ivanov, P. (2016). *Jupyter Notebooks-a publishing format for reproducible computational workflows*. In F. Loizides and B. Schmidt (Eds.), *Positioning and power in academic publishing: Players, agents and agendas* (pp. 87-90). Amsterdam: IOS Press.
- Koedinger, K. R., Corbett, A. T., & Perfetti, C. (2012). The Knowledge-Learning-Instruction framework: Bridging the science-practice chasm to enhance robust student learning. *Cognitive Science*, 36(5), 757-798.
- Mislevy, R., Oranje, A., Bauer, M. I., von Davier, A. A., Hao, J., Corrigan, S.,..., John, M. (2014). *Psychometric considerations in game-based assessments*. : Scotts Valley, Ca: CreateSpace.
- Press, G., (2016, March, 23), Cleaning big data: Most time-consuming, least enjoyable data science task, survey say [Blog post]. Retrieved from:  
<https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/#4364d2dd6f63>
- Reynolds, J. C. (2009). *Theories of programming languages*. Cambridge University Press.
- Robinson, D., (2017, Sept. 6), The incredible growth of Python. Retrieved from  
<https://stackoverflow.blog/2017/09/06/incredible-growth-python/>
- Roskam, E. E. (1997). Models for speed and time-limit tests. In W. J. van der Linden & R. K. Hambleton (Eds.), *Handbook of modern item response theory* (pp. 187-208). New York: Springer.
- Singley, M. K. (1995). Promoting transfer through model tracing. In A. McKeough, J.L. Lupart, & A. Marini (eds.), *Teaching for transfer: Fostering generalization in learning* (pp. 69-92). Mahwah, NJ: Erlbaum.

- VanLehn, K. (2008). Intelligent tutoring systems for continuous, embedded assessment. In C. A. Dwyer (Ed.), *The future of assessment: Shaping teaching and learning* (pp. 113-138). Mahwah, NJ: Erlbaum.
- Wikipedia contributors. (2018, December 29). Data science. In *Wikipedia, The Free Encyclopedia*. Retrieved 18:00, January 6, 2019, Retrieved from [https://en.wikipedia.org/w/index.php?title=Data\\_science&oldid=875831036](https://en.wikipedia.org/w/index.php?title=Data_science&oldid=875831036)
- Wikipedia contributors. (2019, March 18). Analytics. In *Wikipedia, The Free Encyclopedia*. Retrieved 14:22, April 11, 2019, from <https://en.wikipedia.org/w/index.php?title=Analytics&oldid=888300383>
- Wikipedia contributors. (2019, October 29). Data model. In *Wikipedia, The Free Encyclopedia*. Retrieved 20:33, November 7, 2019, from [https://en.wikipedia.org/w/index.php?title=Data\\_model&oldid=923547472](https://en.wikipedia.org/w/index.php?title=Data_model&oldid=923547472)
- Zumbo, B., & Hubley, A. (eds) (2017). *Understanding and investigating response processes in validation research*. Cham, Switzerland: Springer.