## Data Dictionaries

The purpose of a data dictionary is to provide a roadmap for anyone who wants to use a dataset. Suppose you're a government agency publishing data about a pandemic. You would have to provide anyone who wants to use this data with a general outline of what the fields mean because data only have meaning in a context. For example, look at this table of data:

| | |
|---|---|
| 2022-02-01 | 98.6 |
| 2022-02-02 | 99.1 |
| 2022-02-03 | 98.5 |
| 2022-02-04 | 98.2 |

What does it mean?

While we can be reasonably sure that the left column is a date, is it in the format YYYYMM-DD or YYYY-DD-YY (are these days in February or the second day in four different months)? The right column could be temperature by day, grade on a test on a given day, the radio station someone listened to on a given day, etc. Without labels, it's very hard to know or even guess. That's why any important data set will be accompanied with a "data dictionary."

Generally, data dictionaries are documented in a table. They aren't all consistently formatted the same way, but there are some general things anyone who wants to use a dataset will be happy to know. Thus, you'll need to find out if there is a style preferred by your employer / team. If they are new to data dictionaries, you'll have the privilege of setting the standard yourself and you can structure them however you prefer.

Here's an example of one way to format a data dictionary that would be helpful for anyone who wants to use your data.

**Person**

| Property | Column | Type | Length | Nullable? | Values | Example |
|---|---|---|---|---|---|---|
| uniqueID | 1 | integer | 10 | N | | 1 |
| firstName | 2 | string | 256 | Y | all | Beyonce |
| Sex | 3 | enum | 1 | Y | 0 – none<br>1 – female<br>2 – male | 1 |

Notice that the table has a title – "Person", it's common to capitalize the name of the data object / model / class / entity.

Then, for each property of the “person” object (example: firstName), we’ll have a few columns to give us information about that property:

Property – this is where you give a name to the property. I’ve used camel case, but it’s your data, so feel free to choose some other convention (e.g., snake_case). Just be consistent!

Column – if you’re explaining tabular data (e.g., data that would fit in an excel spreadsheet table) then you may or may not have a header row in the data files themselves. It’s good to tell which column the in either the first or second column of your data dictionary.

Type – this is where you explain the type of data this is. Common types would be numbers (floating point, decimals, integers), strings to store text, or you may have an “enumerated” type – “enum” – that could reference a constrained list of values. In the example above, “sex” is enumerated – encoded – so that 0 means the person preferred not to identify their sex, 1 means “female” and 2 means “male.” Very commonly you’ll have an enumerated field to store a US State – (AL, AK, AZ, AR, etc.) Another common type is “boolean” (yes/no 0/1).

Length – this helps to explain to the person using your data how long to expect the values to be. For example, a traditional text message would be type = string and length = 140.

Nullable – this indicates whether the specific field can be missing or not (yes or no). A nullable field is one that anyone using the dataset should be prepared to not have. For example, maybe the field didn’t always get collected from people whose records were created in a case management system. Or maybe there’s a good reason for the field’s not being there. Imagine a health informatics database for a veterinary records system. If it contains male and female house pets, a field for “Date Neutered” would only apply to male animals and “Date Spayed” would only apply to female animals. In the case of a male animal, “Date Spayed” would be “NULL” or simply missing / blank.

Values – this is the column where you could document the allowed values for an enumerated field. For the field “usState” you might include “2-letter Abbreviations” or even an explicit list: “AL, AK, AZ, AR, CA, CO, CT.”

Example – it’s often helpful to provide an example of what data fields look like. In the case of “firstName” above, we provide “Beyonce” as an example of a human first name.

## Sample Dictionary

Let's fill out our data dictionary for a common situation involving a person. Very often, systems require the management of data about people. As I'm sure you have noticed, people are complicated. Thus, this data object we'll design will be a little simplified but allow us to talk through some important aspects.

### Person

| Property | Column | Type | Length | Null OK? | Values | Example | Notes |
|---|---|---|---|---|---|---|---|
| uniqueID | 1 | integer | 10 | N | | 1 | |
| firstName | 3 | string | 256 | Y | | Beyoncé | utf-8 |
| middleInitial | 4 | char | 1 | Y | A-Z | G | |
| lastName | 5 | string | 256 | Y | | Knowles-Carter | |
| sex | 6 | enum | 1 | Y | 0 – none<br>1 – female<br>2 – male | 1 | |
| addressLine1 | 7 | string | 256 | Y | | 123 Smith Street | |
| addressLine2 | 8 | string | 256 | Y | | Suite 100 | |
| city | 9 | string | 256 | Y | | Starkville | |
| state | 10 | char | 3 | N | USPS State Abbreviations | MS | Ref |
| postalCode | 11 | string | 9 | N | Zip+4 | 39759-1232 | |
| email | 12 | string | 128 | Y | | beyonce@gmail.com | RFC 3696 |
| phone | 13 | string | 10 | Y | 0-9 | 6623250123 | |

## Questions to Answer

1. In what column would I find the user's city?
2. Is the email field long enough based on the referenced standard?
3. Could I represent an international phone number in a dataset that conformed to this data dictionary?
4. This question requires Google – How should I securely store a value in this database that could be matched to Beyonce's password if I were going use this data in a system with a login?