EPBI 414

Unit 2
Data Types & Structures

The Recap - Unit 1

- Data: organizing raw input through **epistemic models**
- Data types: how we represent information in the computer
- Very basic types: character and numeric
- Case report forms: frameworks for data

Unit 2 Overview

- More detail on general data types
    - Integer, decimal, and irrational numbers
    - Storing and representing text (length, encoding)
    - Dealing with open text (abstraction for analysis)
    - Storing and understanding categorical data, as well as Boolean categories
    - Representing dates to computers

Unit 2 Overview

- Introducing some conceptual data structures
  - Variables
  - Arrays and dimensions
  - Tables
- Translating raw input to data - case report forms
  - Missingness: representing data integrity problems

The basics: numbers

- Numbers seem pretty clear-cut at first
  - Actually, numbers are kind of crazy

  - Try PHIL 413, *Philosophy of Mathematics*, if you really want to **think** about numbers

- A computer is a computing machine

- No "common sense"
  - Strictly speaking, it doesn't think

Representing numbers

- To prevent "accidents", it often helps to be explicit with computers
- Some programming languages require more explicitness than others
- Those which require less often make assumptions under the hood
  - This will break your code once in a great while

Broad types of number

- **<u>integer</u>** - whole numbers, without fractional components (for instance, 3)

- **<u>decimal</u>** - numbers with a fractional component (for instance, 6.2)
    - Special case of decimal numbers: <u>irrational numbers</u>

    - These can cause trouble for computers - rely on built-in structures for handling these

Defining numbers

- Some languages (e.g. R) treat numbers as having only a few types (`numeric`, `integer`)
- Others (e.g. C++) have multiple types of numbers (`int`, `float`, `double`, etc)
- Databases usually have options to explicitly or implicitly define decimals

The height example

- Hypothetical study - gathering heights of students
- Your meterstick is marked in increments of 0.1m
- So your height data look like 2.2m, 1.9m, and so on

What type are the heights?

- Laziest: numeric
- More specific: decimals
- More specific: decimals with a single place to the right of the decimal point
- Most specific: decimals with two numerals, one of which is right of the decimal point

Increasing specificity

numeric	`3`	`2.83`	`110.7`
decimals	`3.0`	`2.83`	`110.7`
decimals to 1 place	`3.0`	`2.8`	`110.7`
decimals with 2 numerals to 1 place	`3.0`	`2.8`	`<ERR>`

Why be more explicit?

- The more specific your type, the less ambiguous your data
- Also can prevent data entry errors
- Removes "assumptions" from computations
- Makes explicit the accuracy / precision of your measurement tool

Range vs. type

- Type is fundamental to a piece of data
- Range checks are applied on top of type
- Type is "3 numerals, one decimal place" (the numbers 0.0 - 99.9)
- Range is "Values between 0.0 and 39.9"

Shorthand representations

- INT for integers
- DEC(3.2) for decimals with 3 numerals and then 2 numerals (i.e. 323.19)
- xxx.yy as a visual representation of decimals
- Use what is clear to you and what works
  - Might be language-dependent

Text data

- Text data is tricky for computers to handle

- Fundamental questions:
    - length?

    - content?

    - use?

Example "text" data

- What is your favorite color?
- What is your opinion of Abraham Lincoln as a president?
- Please list all medications you are taking currently.
- List five cities in the United States.

Properties of text data

- Commonly referred to as `string` or `character`
    - Sometimes a second type for longer values

- Often defined by length (30 characters)

- Beware so-called *invisible characters*
    - carriage returns, line feeds, tabs
    - Can wreak havoc on storing text data

Text encoding

- Text encoding changes letters to numbers
  - Computers really good at numbers
- Some common encodings:
  - UTF-8 (Unicode)
  - ISO 8859-1 (Latin-1)
  - ISO 646 (ASCII)
- Text encoding matters! (sometimes)

Why care about encoding?

- If you work across international boundaries
- If you want your text data to make sense in a different locale
- If you can't get the text data you just received to make sense now
- If you are asked what to save a file in

The problem of text

What makes sense on a data collection form is often not useful in a computer.

Opinions of Lincoln

What is your opinion of Abraham Lincoln as a president?

- "I think he was a good president."
- "He should have tried to end slavery without a war."
- "Overall, good, but his suspension of habeas corpus was dangerous."

Solving the problem

- Decide before collection what is desired
    - Requires *ante hoc* abstraction from complex data
- Reduce existing data
    - Requires *post hoc* abstraction from complex data
- Quantitative vs. qualitative debate
    - Statistical programming class = quantitative basis
    - Focus will be on transforming text toward quantitative analysis

Fixing Lincoln: ante hoc

- Original question: "What is your opinion of Abraham Lincoln as a president?"
  - Way too broad
- Rating: "In your opinion, was Abraham Lincoln a good President, or a bad President?"
- Specific issue: "Do you believe Abraham Lincoln handled the Civil War correctly?"

Fixing Lincoln: post hoc

- Human element: read through and judge what each person said
  - Must be based on desired epistemic framework

- Look for certain words ("bad", "slavery", etc)

- Use new and fancy tools for textual analysis

Abstraction

- Both fixes for Lincoln involve abstraction
- Data is almost always an *abstract representation of reality*
- Reduces complexity by trying to generalize
- The level of abstraction depends on the situation and goal

The general rule of thumb

Greater levels of abstraction make it easier to analyze data, but make the results less comprehensive.

Categorical data

- Key tool for abstraction: create categories
- Often created during collection, but sometimes not
- Can be stored in computers in multiple ways
  - Sometimes, system dependent (e.g. R and SAS do it different)

Categorical Data: Race

- What is your race? Please select one item below:
  - Black or African-American
  - White or Caucasian
  - Asian
  - Native Hawaiian or Other Pacific Islander
  - American Indian or Alaska Native

Storing categories - 1

As Text

```
patient_id      race
----------      ----
101             Black
102             White
103             Asian
104             White
105             Pacific Islander
```

Storing categories - 2

As Numbers

```
patient_id    race
----------    ----
101           2
102           1
103           3
104           1
105           4
```

Why use numbers?

- Seems easier to use some text
- Much more human-readable
- However, invites two major problems
    - Allows inputs that you might not want (language-dependent)
    - Allows for misspellings, capitalizations, and other "false differences" to occur

A bit of history

In the dark days of computing, data storage was at a premium

Completely affordable[1]

Storage >>> Gold

- The number 1 takes up less memory than the word Caucasian
- So, store the number, and map it to a value later
- This has influenced languages and data storage for years
  - No longer strictly necessary

Checking for matches

- Trying to check if two values are the same can take longer if they are text
  - For the computer, of course
- Text has more elements - each must be checked
- A number can be checked in a single step
  - So, numeric values are often easier to search

Categories limit input

- Text can allow inputs you didn't intend
  - This can be language-dependent

- You could end up with "Multiracial", "Two Races", "More than One Race", "Hispanic", "Greek", "European", and so on
  - Sometimes your data capture tool can limit this

- Using numbers removes this possibility
  - A way of forcing an epistemic model

False differences

- Using text where people enter it is dangerous
- Computers don't "know" anything about correcting human reading
- Consider the following...

Some race data

```
patient_id     race
----------     ----
101            Black
102            White
103            Asian
104            white
105            Pacific Islander
106            WHITE
107            Whtie
108            Black
```

Human summary

```
patient_id        race
----------        ----
101               Black
102               White
103               Asian
104               white
105               Pacific Islander
106               WHITE
107               Whtie
108               Black
```

```
SUMMARY
race                        count
----                        -----
Asian                         1
Black                         2
Pacific Islander              1
White                         4
```

Computer summary

```
patient_id       race
----------       ----
101              Black
102              White
103              Asian
104              white
105              Pacific Islander
106              WHITE
107              Whtie
108              Black
```

```
SUMMARY
race                     count
----                     -----
Asian                      1
Black                      2
Pacific Islander           1
White                      1
white                      1
WHITE                      1
Whtie                      1
```

Explicit categories

- By using numeric categories, you avoid this problem
- Not *always* numeric - for instance, R uses "factors"
- Same core principle: creates structured categories rather than plain text

Multiple category data

- Sometimes, people can fall into more than one category on the same measure

- More categories = more challenge in representing data

- Can be done via relational structure
  - More coming on relational structures

Multiple categories - race

- A common way to ask about race: "What is your race? Please check all that apply."

- Makes your life frustrating as a data analyst
  - Almost always gets collapsed into a single category during analysis

  - Makes you spend time translating data for edge cases

Binary markers

- Questions that allow multiple categories to be selected often use multiple binary categories
- Binary categories have two states (commonly 0 and 1 in regression)
- Multiple categories = one binary for each possible category

Selecting multiple races

```
patient_id   r_white  r_black  r_asian  r_pacisland r_amerind
----------   -------  -------  -------  ----------- ---------
101             0        1        0          0          0
102             1        0        0          0          0
103             0        0        1          0          0
104             1        0        0          0          0
105             0        0        0          1          0
106             1        0        0          0          0
107             1        0        0          0          0
108             0        1        0          0          0
```

Collapsing binary data

- Requires a lot of programming logic
  - Touches on problems of combination
- For instance, race:
  - If one of the categories is 1, that is the race
  - If more than one is checked, the subject is multiracial
  - If none are checked, it's missing (should this be allowed?)

Boolean data

- The phrase "Boolean" comes from George Boole
  - The father of algebraic logic
- A Boolean variable represents truth and falsity in binary logic
  - There are other kinds of logic, but main focus is binary
- Could call it a special kind of binary category

Booleans in practice

- Commonly represented as TRUE and FALSE, or 0 (FALSE) and 1 (TRUE)

- May appear as character or as numeric, but often ***system reserved***
  - Sometimes, the words TRUE and FALSE are reserved by the language

- Booleans are essential to symbolic logic and control statements (upcoming units!)

Dates and times

- Dates and times are hard for computers
- Everyone remembers Y2K
    - even if nothing happened
- Why are dates and times so hard?
    - No intrinsic rules
    - Misunderstanding of formats
    - Heavy amount of human cultural baggage

An example

- One second is well defined by international standards[2]:

  "The second is the duration of 9,192,631,770 periods of the radiation corresponding to the transition between the two hyperfine levels of the ground state of the cesium 133 atom."

Why so bizarre?

- It used to be that a second was 1/86,400 of a solar day.
- Then we discovered that the solar day is variable
- Not a great way of measuring things objectively!

It gets worse

- 60 seconds in a minute
- 60 minutes in an hour
- 24 hours in a day (but not really)
- Between 28 and 31 days in a month
- 12 months a year, usually 365 days

And even worse

- You are working on a project in Cleveland
- Your collaborators are located in Tokyo
- You store dates in your database like this:

```
action   user    datetime
------   ----    --------
  1      101     2015-08-17 03:31:12
  2      101     2015-08-17 03:31:18
  1      102     2015-08-18 04:12:22
```

Zoning out

- Your colleagues ask what times a user performed action 2
- If you are exceedingly lucky, all the times are in the same time zone
- More likely, they are all in different time zones, and you can't answer

Another date error

- You obtain data from some colleagues in Europe. During data import, you keep having date errors. You visually inspect the data and see:

```
subject_id   dob
----------   ---
     1       8/7/12
     2       7/8/12
     3       12/7/8
     4       10/3/5
     5       9/8/2
```

Untangling date errors

- Most date and time issues are more tainted by us than by the computer

- Best handled through three key steps:
  - Try to conform to a date standard (i.e. ISO 8601)

  - During import, move dates into the language's date format

  - Use the language's preferred tools to manipulate dates

Found in Translation

- Dates come to us in many formats

- In general, we might call them "character" data to start

- Bringing them into a computing environment is telling the computer how to interpret them

A variety of the same date

Dates to us

- Thu, October 8, 2015
- 2015-10-08
- 10/8/2015
- 8/10/2015
- 2015/10-8

Dates to the computer

- "Thu, October 8, 2015"
- "2015-10-08"
- "10/8/2015"
- "8/10/2015"
- "2015/10-8"

How can we translate?

- Computers work best with numbers
- One way to represent a date: as a distance from some point in time (the "epoch")
- One very common epoch is Jan 1, 1970 - very common in Unix environments

From there to here

- Let the language do the work
  - It will work better that way
- Your job: tell the computer how to read it
- The computer's job: read it and do the math

Translated dates

Dates to us

- Thu, October 8, 2015
- 2015-10-08
- 10/8/2015
- 8/10/2015
- 2015/10-8

Dates to the computer

- 16716
- 16716
- 16716
- 16716
- 16716

Use the language's tools

- Common task: finding differences between dates
- The language will most likely have a tool for this: use it!
- Don't be tempted to write your own system

A homebrew attempt

Our awkward attempt

```
date 1: 2014/11/13
date 2: 2015/12/10

date     year     mon      day
  1      2014     11       13
  2      2015     12       10

if year.1 < year.2
 if mon.1 < mon.2
  if day.1 < day.2
   diff = (year.2-year.1)*365 + ...
   ...
   For quite awhile!
```

The much better way

```
date 1: 2014/11/13
date 2: 2015/12/10

diff = datediff("2015/12/10",
                "2014/11/13")
```

Why the built-in system?

- Someone smarter than us already built functions to handle these
- It is more consistent to use built-in tools
  - Handles weird things like leap years
- Better conformance with standards
- Use your time doing what you do best - analysis

The timezone issue

- Most likely to rear its head when you are working across borders
- Can be handled by ***doing everything*** in UTC
- Times can always be changed to display locally - so always store them in UTC

Back to Tokyo

```
action   user     datetime                tz
------   ----     --------                --
  1      101      2015-08-17 03:31:12     +9
  2      101      2015-08-17 03:31:18     +9
  1      102      2015-08-18 04:12:22     -4
  1      103      2015-08-18 12:11:18      0
  3      101      2015-08-19 15:12:22     +9
```

Add a timestamp

Like many things, Unix timestamps measure time since January 1, 1970.

Back to Tokyo

action	user	datetime	tz	timestamp
1	101	2015-08-17 03:31:12	+9	1439749872
2	101	2015-08-17 03:31:18	+9	1439749878
1	102	2015-08-18 04:12:22	-4	1439885542
1	103	2015-08-18 12:11:18	0	1439899878
3	101	2015-08-19 15:12:22	+9	1439964742

Epoch win!

- Dealing with dates means you need a plan
- Make date processing easier by doing three things:
  - Try to conform to a single date standard
  - Translate the incoming data to something the computer understands
  - Work with dates using the built-in tools, not your own experiments

Break Time

Data Structures

- Data structures are an important part of computer science
- Most people pursuing CS degrees start with a data structures class
- To be clear: this is not that "data structures" material

The basic data structures

- We are concerned about analytic applications

- The general data structures of interest to us:
    - **<u>The variable</u>** - a single piece of data

    - **<u>The array</u>** - a group of objects that all share the same type, connected in one or more dimensions

    - **<u>The table</u>** - specifically, data of multiple types, connected in exactly two dimensions

Language dependency

- It bears repeating that ***data structures are often language-dependent***
- For instance, R makes use of vectors and lists, while SAS is more focused on tables
- These are general, conceptual data structures

The variable

- The most basic unit to represent data is ***the variable***

- The stock variable is "x"

- Generally contains a single value (datum)

- May be typed in numerous ways (numeric, character, etc)

The array

- An array connects objects ***of the same type*** together in ***one or more dimensions***

- Sometimes, this is called a vector, especially variables in one dimension

- Example: a series of numeric variables is a numeric array
    - `[1,2,1,3,5,3,2]`

Dimensions and elements

- Generally, arrays may have numerous dimensions

- It gets hard for humans to conceptualize after the first few of them

- The elements of an array are at locations specified by the dimensions

One-dimensional array

1	2	1	3	5	9	2	1

Two-dimensional array

1	2	1	3	5	9	2	1
6	5	5	1	2	3	8	1
0	3	2	0	2	6	5	2

Three-dimensional array

1	2	1	3
2	5	9	1
8	2	4	3

4	2	9	6
7	9	3	5
5	2	9	6

3	6	1	0
2	6	1	6
2	3	2	2

Common array shorthand

- Common shorthand: `A[n]`
  - `A` is the name of the array
  - `n` the ***index***, or position in that dimension
- More than one dimension = more than one index
  - e.g. `A[n,o]`
- Holy war: do indices start at 0 or 1?
  - For us, usually with 1

Array shorthand, 2

- Common to use square brackets to refer to arrays
- `A[n]` is how you refer to *n*th element of `A`
- `[3,2,3,1,5]` is how you represent an entire array
  - To show name, use `A = [3,2,3,1,5]`

One-dimensional array

1	2	1	3	5	9	2	1

- `A[1]` = red, value 1, `A[6]` = blue, value 9
- n ranges from 1 to 8
- Full definition: `A = [1,2,1,3,5,9,2,1]`

Two-dimensional array

n

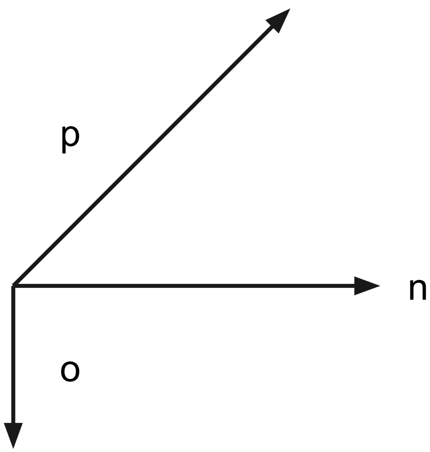
1	2	1	3	5	9	2	1
6	5	5	1	2	3	8	1
0	3	2	0	2	6	5	2

o

- Be sure you know which is which index
  - Language matters (as always)

- Here, `A[n,o]`
  - `A[3,2]` is red, value 5; `A[5,1]` is blue, value 5

Three-dimensional array



1	2	1	3
2	5	9	1
8	2	4	3

4	2	9	6
7	9	3	5
5	2	9	6

3	6	1	0
2	6	1	6
2	3	2	2

`A[n,o,p]`

`A[4,3,3]` = blue, 3

`A[1,1,1]` = green, 3

`A[2,2,2]` = red, 9

The value of arrays

- Arrays connect objects over dimensions
- This can be a useful ***epistemic model***
- Arrays are also useful for "convenience"
  - The power of iterative processing
  - Computers don't get bored repeating the same thing endlessly

Arrays of arrays

- Specifically stated: arrays connect objects ***of the same type*** together in ***one or more dimensions***

- Arrays can contain other arrays ("nested arrays")

- This allows us to create arrays that have multiple types

The table

- A table can be considered a specialized kind of one-dimensional array, where the elements are ***also*** one-dimensional arrays, each having the same length

- Restated: a table consists of columns of different type, each having the same length

Aren't tables just arrays?

- Yes - but not like you think
  - Arrays are always single type

  - You don't want this: `[1,2,"blue",4,-1]`

- Tables are useful because they let us combine data types

- To do this: make each column an array, then create an array of those columns

Nesting arrays

- Tables are arrays containing other arrays ("nested arrays")

- An array `B[n,o]` where n = {1,...,5} and o = {1,...,3} looks like this:
    - `<x,y>` gives location of each cell

<1,1>	<2,1>	<3,1>	<4,1>	<5,1>
<1,2>	<2,2>	<3,2>	<4,2>	<5,2>
<1,3>	<2,3>	<3,3>	<4,3>	<5,3>

Why nest?

- Array `B` only allows us to store one type of data
- If we want names in one dimension, and heights in the next, we will be in trouble
- We can get around this by making it an array of other arrays

Creating nested arrays

- Changing `B` into `C`, with nested arrays
- `C = [D1,D2,D3,D4,D5]`
  - `C[1] = D1`, where `D1 = [x,y,z]`
  - `C[2] = D2`, where `D2 = [a,b,c]`
  - And so on through `C[5]`

The result of nesting

- The result of nesting is C, as below:

`D1[x,y,z]`	`D2[a,b,c]`	`D3[d,e,f]`	`D4[i,j,k]`	`D5[q,r,s]`

`[D1,D2,D3,D4,D5] [[x,y,z],[a,b,c],[d,e,f],[i,j,k],[q,r,s]]`

- The location of `B[1,2]` is equivalent to the location `C[1][2]`
  - Remember, `C[1] = D1`
- But now, columns can have different types

From arrays to table, 1

A1 =	1	2	3	4	5	6	7	8
A2 =	James	Sasha	Roland	Carly	Philip	Kenji	Yvonne	Rob
A3 =	1	2	1	2	1	1	2	1
A4 =	1	3	1	1	2	3	2	4

From arrays to table, 2

A1	A2	A3	A4
1	James	1	1
2	Sasha	2	3
3	Roland	1	1
4	Carly	2	1
5	Philip	1	2
6	Kenji	1	3
7	Yvonne	2	2
8	Rob	1	4

From arrays to table, 3

A1	A2	A3	A4
1	James	1	1
2	Sasha	2	3
3	Roland	1	1
4	Carly	2	1
5	Philip	1	2
6	Kenji	1	3
7	Yvonne	2	2
8	Rob	1	4

```
T = [A1,A2,A3,A4]
```

Tables vs. Arrays

<u>Arrays</u>

- Group of objects, either variables or arrays
- Single type
- *n* dimensions

<u>Tables</u>

- Special type of array, comprised of other arrays
- *n* types
- Two dimensions

A simplified model

- Real languages each have their own restrictions and tools
  - In reality, you will work with built-in structures, rather than thinking "I'll make an array of an array!"
- This is a set of general rules to serve as a basic way of thinking about programming
- Once you start programming, you can make things crazy if you want

Data structure takeaways

- Variable: one piece of data, one type
- Array: multiple objects, one type
    - The object can be a variable, or another array
    - Array of arrays = nested arrays
- Table: multiple variables, multiple types
    - Conceptually, a one-dimensional array, where the elements are also one-dimensional arrays of the same length

The Case Report Form

- One of the most common ways that data is collected
- A place where you can solve all your problems before you have them!
- A concrete example of an epistemic model

Review your CRFs!

- Common request: "Please analyze this data!"

- Similarly, "You're going to be our data manager for this project."

- If you hear these, and you don't get to either review the CRFs, or have input on them...
    - ...consider carefully how you approach the project.

CRF Design

- If you will work with the resulting output, it helps to have input on the CRFs

- CRFs do not happen in a vacuum - usually, they are supported by many documents, such as:
  - Data management plan
  - Data validation plan
  - Data dictionary

Good Looking CRFs

- Obtain a real editing tool and use it
  - Word will not cut it in the professional world
- You need precision ability to lay out fields and items
  - Might start with Publisher
  - For $$$: Adobe InDesign
  - FOSS alternative: Scribus

Final Notes on CRF Design

- Laying out and designing useful paper forms is a real talent
- Consider working with a designer to make your forms usable
- Writing a CRF is not the same as designing a survey (though there is some overlap)

CRF Annotation

- One of the most useful tools for understanding data is ***annotated CRFs***

- Annotated CRFs overlay the data type and model information we need on the form everyone uses

- If you are at the start of a project: design the forms, finalize them, then ***annotate them***

Methods of Annotation

- Simplest method: pen & paper (scan in your notes)

- Can put annotations onto PDF files as well

- In a pinch, you can use an Excel file and describe everything
    - Can be a pain to line everything up

What needs annotated?

- The name of each variable (column)
- Any categories and what they mean
  - For example, 1 = FEMALE and 2 = MALE
- Anything needed to connect this to the data dictionary

The data dictionary

- A more comprehensive overview of the CRF
    - Can exist for all data sources
- Things you should include:
    - Length and formatting considerations
    - Response logic on the form
    - Material from annotations

CRF / Dictionary Example!

Missing data

- Missing data is the bane of most investigators
- When possible, build your form to account for it ahead of time
- Try to distinguish between "system missing" and "actually missing"

Missingness in data

- Most languages have a tool to represent missing or absent data
  - R: NA (reserved character)
  - SAS: . (for numerics)
  - SQL: NULL
- All it tells you is that something isn't there

The sources of absence

- Data can be missing for two key reasons
- We don't have it at all ("actually missing")
- We don't have it in the database ("system missing")
- Without proper design, you can't tell the difference

An example

- You ask a question about race for each subject
- One subject refuses to answer
- You didn't put that choice on your forms, so your data entry person leaves it blank

An example, continued

- You now need to report to your data monitoring committee
- They ask why patient X has no race...and you have no answer
  - Did your team forget to ASK the question?
  - Did your team forget to enter it?
  - Did the patient refuse to answer?

What you see...

- ...is what you ***know***, as data analysts
- If someone refused to answer, but the system just shows as missing...
- Then it's just missing

Options to avoid this

- Explicitly include a "missing" option for every question
  - For open text, include instructions for missing data (e.g. "Put -9 if the data is not available.")

- For items completed by subjects, include options indicating refusal
  - Helps to distinguish between a patient refusing, and data collection problems

- Build it into your EDCS

The good and the bad

The better way

patient_id	hgb.a1c
1	7.7
2	6.3
3	8.2
4	-9
5	4.9

The worse way

patient_id	hgb.a1c
1	7.7
2	6.3
3	8.2
4	NA
5	4.9

Attributions

1. Image obtained from http://www.belch.com/blog/wp-content/uploads/2013/12/20131214-182110.jpg; license is unclear.

2. Definition taken from the NIST page: http://physics.nist.gov/cuu/Units/second.html