DATA 301 Introduction to Data Science Alexander Dekhtyar

Tabular Data

The most simple, the most common and the most popular way of representing data is tabular form.

The tabular representation of data is designed as follows:

- The tabular representation is used to represent a single dataset.
- Each data point in the dataset is represented in the same way.
- The dataset is represented in a from of a two-dimenstional table.
- Each *row* of the 2D table represents a *single data point*. Often, rows in the table are referred to as tuples or records.
- Each row consists of a number of columns.
- Each column is usually given a *name*. Often, columns in the table are also referred to as attributes or fields.
- Each column usually is assoiciated with a single atomic data type.
- Each column in the table is used to represent one single observation (*datum*) in a represented *data point*.
- Each data point (row) in a table is considered to be unique. A unique *rowId* is often associated with each data point in the table as the proof/verification of this uniqueness.

Often, *tabular data representation* is referred to as matrix data representation. We consider these two terms synonyms, but we recognize that the latter term has somewhat limited and specialize usage that often has to do with the specific origins of data.

Tabular data representation can be dense or sparse.

Dense tabular representation/dense matrices. All, or *almost all* columns in all rows possess at a value. Where a column has a *missing value* a typical assumption is that the value actually exists (and will eventually be restored), but is currently unknown.

Sparse tabular representation/sparse matrices. The majority of individual values in the table as missing. (A typical sparse matrix may have only 5-10% of non-missing values). In sparse data representation, the assumption is that missing values indicate not the *lack of knowledge* about an observation, but a *lack of observation itself*.

What can be stored in tabular data?

Use tabular representation to store the following data:

- **Collections of simple observations.** These datasets consist of data points where each observation yields values of *atomic types* (e.g., numbers, strings, etc.).
- **Data keyed to a single type of observations/objects.** A single 2D table is capable of representing fairly rich information about a collection of similar objects or observations. But storing information about different types of objects and observations (e.g., customers in a store, and store locations) in a signle table is difficult and inconvenient.

Examples

A few examples of tabular data a presented below.

Example 1. Baby names. The `kaggle.com` dataset of *Baby names* is a typical example of a simple tabular data. The dataset consists of two data tables, we discuss one of them, the National baby names dataset, in detail below.

- Each row in the data table shows a single observation: the number of times a specific baby name was used to name a baby of a specific gender in a given year.
- The following columns are found in the dataset:
  1. Id: the unique identifier of each observation (an *integer*)
  2. Name: the baby name (a *string*)
  3. Year: the year of the observation (a *integer* or a *date*)
  4. Gender: the gender of the baby who received the name (a *string* or a *character*)
  5. Count: the number of babies of the given gender who were given the name in the specified year (an *integer*).

The first few lines of the table look as follows (we place the headers for each column in the table for convenience):

Id	Name	Year	Gender	Count
1	Mary	1880	F	7065
2	Anna	1880	F	2604
3	Emma	1880	F	2003
4	Elizabeth	1880	F	1939

This is a dense table, representing a single observation (count) per data point, based on three *independent variables* (name, year and gender of the baby).

Example 2. Sales figures. A simple dataset for a company that produces a variety of (let's say) cell phone accessories and sells them on-line. The sales data consists of the following information:

- SKU: the bar code uniquely identifying each type of item the company makes and sells.
- Product: the name of the product.
- Price: nominal price of the product.
- Number of items sold by quarter: number of individual copies of the product sold in each quarter (four columns total).
- Revenue for each quarter: revenue from sale of the product for each quarter (four columns).
- Total revenue: total revenue for the year from sale of the product.
- Profit: total profit made off the sale of the product.
- Profit percent: profit made off the sale of the product as percent of revenue.
- Profit rank: the rank of the product by profit amount.

A portion of the dataset may look as follows:

SKU	Product	Price	Sales Numbers				Sales Revenue				Revenue	Profit	Profit%	Rank
			Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4				
10450432	iPhone 6 cover (red)	$11.99	32	53	42	65	383.68	635.47	503.58	779.35	2302.08	575.52	25%	7
10450433	iPhone 6 cover (purple)	$ 11.99	26	28	35	102	311.74	335.72	419.65	1222.98	2290.09	572.52	25%	8
10450501	iPad mini cover (green)	$18.99	183	22	19	21	3475.17	417.78	360.81	398.79	4652.55	2093.65	45%	3

This is a dense table that contains multiple observations in a single row/record.

Example 3. Graduate Course grades. Students in the MS in Computer Science program must take at least five graduate courses. A simple table can be used to track how they did specifically in the CSC 500-level coursework. The dataset can consist of the following columns:

1. Name: name of the graduate student.
2. Program: an indicator of whether the student is in the MS or BMS program.
3. Course grades: a course grade for each of the following courses: CSC 508, CSC 509, CSC 515, CSC 521, CSC 530, CSC 540, CSC 550, CSC 560, CSC 565, CSC 571, CSC 580 (11 columns).

Here is a fragment of a sample data table.

Name	Program	508	509	515	521	530	540	550	560	571	580
Jeffrey Han	MS			A		B			A		
Lisa Black	BMS	B		B	A	B				B	A
Clare Ott	BMS			A	C	A		B			
Max Logan	MS	A			A	B		A	A		
Dan Show	BMS						A	C	A	B	B

This is an example of a *sparse matrix* representation of the data. In this table a large number of cells have missing values because the students did not take the listed classes.

Formats for Tabular Data

Tabular data is often available in the following data formats:

- Comma-Separated Values file (CSV file)
- Tab-Separated Values file (TSV file)
- Spreadsheet file
- Database file(s)

CSV files. The most popular format for representing tabular data is CSV. Each line of the CSV file represents one data point, and the individual columns of the table are separated with commas. For example, the data from **Example 1** can be stored in a CSV file as follows (with the optional top line showing the column names):

```
Id,Name,Year,Gender,Count
1,Mary,1880,F,7065
2,Anna,1880,F,2604
3,Emma,1880,F,2003
4,Elizabeth,1880,F,1939
```

TSV files. These are similar to CSV files, except the separator between the column values is a `Tab` character. The effect is to show data separated by whitespace in a file. The data from **Example 1** would look as follows:

```
Id Name Year Gender Count
1 Mary 1880 F 7065
2 Anna 1880 F 2604
3 Emma 1880 F 2003
4 Elizabeth 1880 F 1939
```

Spreadsheet files. A lot of the tabular data is imported into popular spreadsheet packages (MS Excel, being one of them) and are stored in the proprietary data formats (e.g., as an `.xls` or `.xlsx` file) of those applications. Typcially, there are two ways to deal with these files:

- Conversion: most spreadsheet software allow for saving data in CSV and/or TSV formats.
- Direct access: some data formats may come with an API or a straightforward description of data format that allows direct access to the data without conversion programmatically.

Database files. The vast majority of tabular data comes from relational databases. In this case, the actual data is stored in proprietary `database table file formats`. Generally speaking, the data can be easily moved in these formats from one location to another (assuming both locations have access to the same DBMS).

More often though, access to the tabular data in the database is done through direct querying of the database, either via a collection of SQL scripts or via programmatic access to the database via database connectivity libraries that are available in most programming langauges. More about this in CSC 365.