

Notes Data Wrangling

Has notes from Udacity's Data wrangling module

Chapter 1: Introduction to Data Wrangling

Assess

Low Quality Data is commonly referred as dirty data. which has issues with it's content.

Common quality issues are

missing data

invalid data

inaccurate data

inconsistent data

Data quality is a perception or an assessment of data's fitness to serve its purpose in a given context.

Tidiness

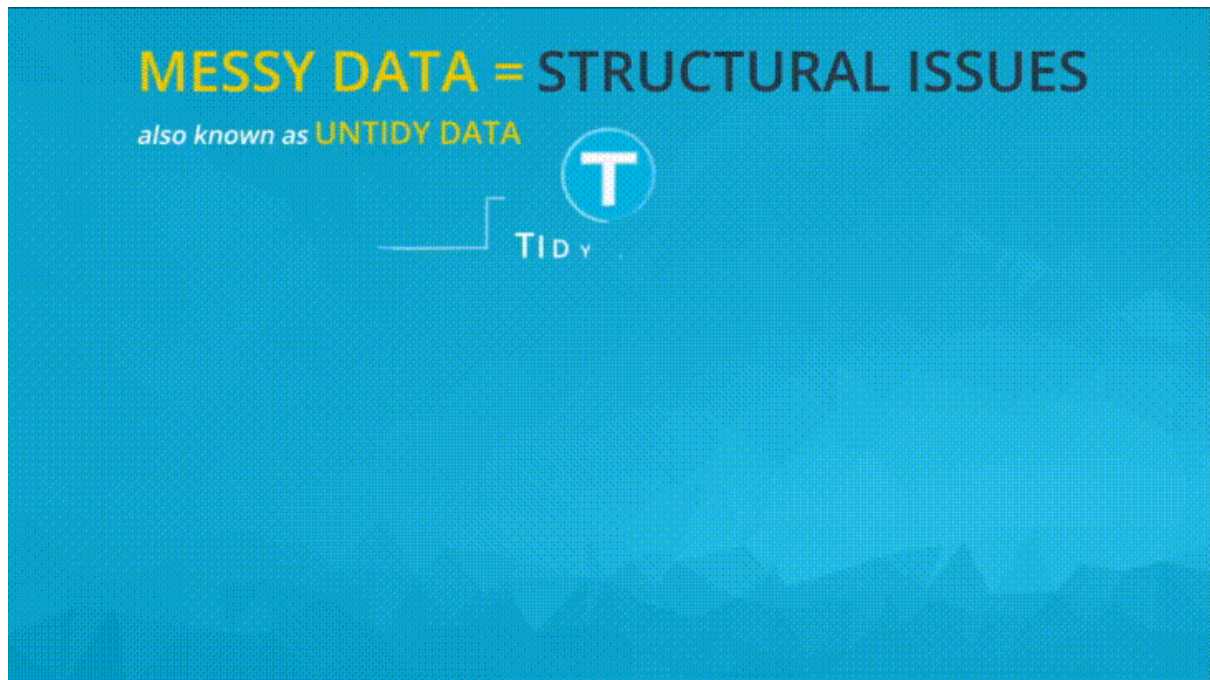
Untidy data is commonly referred to as "messy" data, data which has issues with it's structure

A dataset is messy or tidy depending on how rows, columns, and tables are matched up with observations, variables, and types. In tidy data:

Each variable forms a column.

Each observation forms a row.

Each type of observational unit forms a table.



this is gif explaining the concepts

Visual Assessment

when data is scanned by human eyes to look for quality and tidiness.

Programmatic Assessment

when we use computer program to check for quality and tidiness in the data
this is done for large datasets generally.

Make sure you make notes of what problems you find in dataset regarding
quality and tidiness.

Cleaning

Cleaning means acting on the assessments we made to improve the quality
and tidiness of the data.

```
animals.tail()
```

	Animal	Body Weight (kg)	Brain Weight (g)
24	Chimpanzee	52.160	440.0
25	Mouse	230.000	0.4
26	Apple	0.100	NaN
27	Brachiosaurus	87000.000	NaN
28	Mole	0.122	3.0

Examples of improving quality include:

- Correcting when inaccurate, like correcting the mouse's body weight to 0.023 kg instead of 230 kg
- Removing when irrelevant, like removing the row with "Apple" since an apple is a fruit and not an animal
- Replacing when missing, like filling in the missing value for brain weight for Brachiosaurus
- Combining, like concatenating the missing rows in the *more_animals* DataFrame displayed below

Improving Tidiness

Improving tidiness means transforming the dataset so that each variable is a column, each observation is a row, and each type of observational unit is a table.

Programmatic data cleaning process:

Define → Code → Test.

Defining means defining a data cleaning plan in writing, where we turn our assessments into defined cleaning tasks. This plan will also serve as an instruction list so others (or us in the future) can look at our work and reproduce it.

Coding means translating these definitions to code and executing that code.

Testing means testing our dataset, often using code, to make sure our cleaning operations worked.

Define

Select one of the assesment, and define how to fix this issue. somewhat like shown below.

Clean

Issue 1

Define

Code

In []:

Test

In []:

Issue 2

Define

Code

In []:

Test

In []:

After the cleaning of the data we reassess then iterate over every operation if necessary and may end the process if we are satisfied with the tidiness of the data.

Points:-

EDA: an analysis approach that focuses on identifying general patterns in the data, and identifying outliers and features of the data that might not have been anticipated.

ETL

You also may have heard of the extract-transform-load process also known as **ETL**. ETL differs from data wrangling in three main ways:

1. The users are different
2. The data is different
3. The use cases are different

Chapter 2 : Gathering Data

First Step into the data wrangling process.

Rotten Tomatoes Top 10 movies of all time that the topic we'll be working on.

Flat file structure:

Flat files contain tabular data in plain text format with one data record per line and each record or line having one or more fields. These fields are separated by delimiters, like commas, tabs, or colons.

Advantages of flat files include:

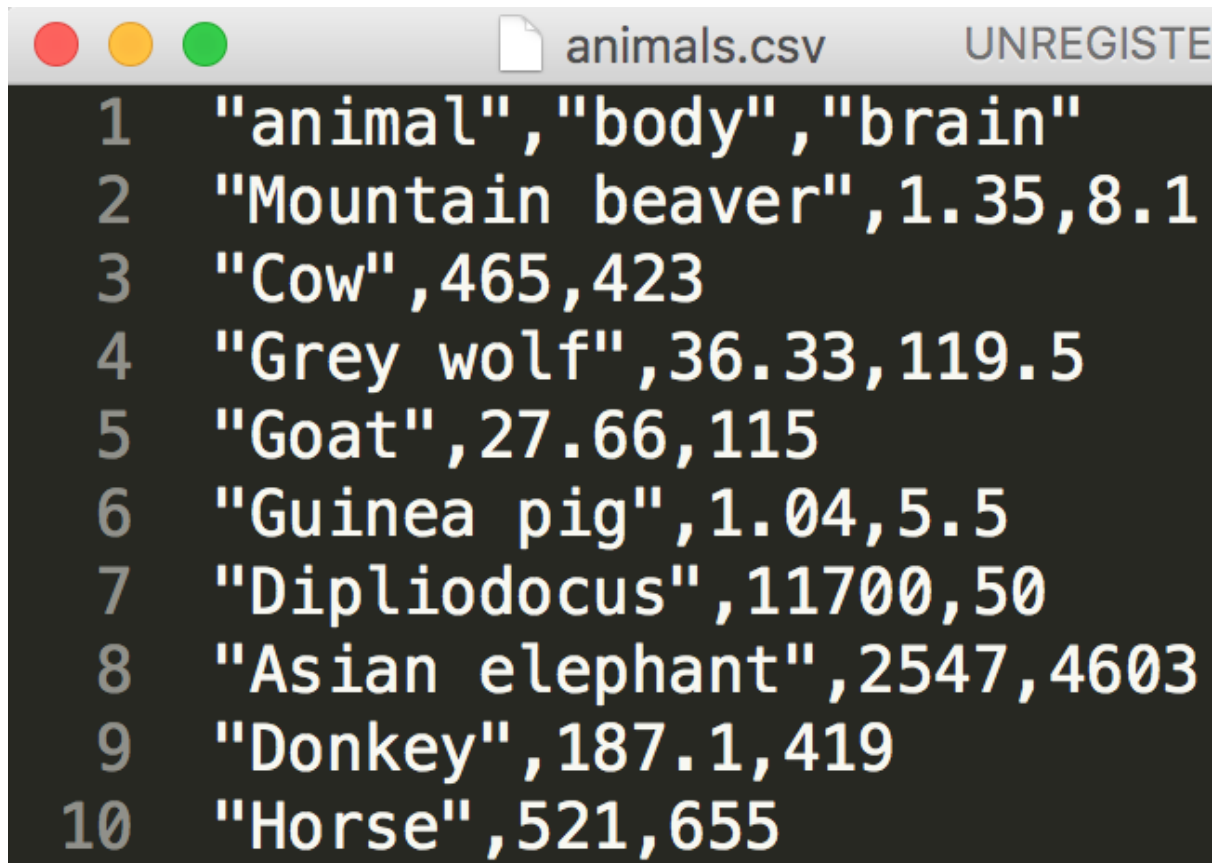
- They're text files and therefore human readable.
- Lightweight.
- Simple to understand.
- Software that can read/write text files is ubiquitous, like text editors.
- Great for small datasets.

Disadvantages of flat files, in comparison to relational databases, for example, include:

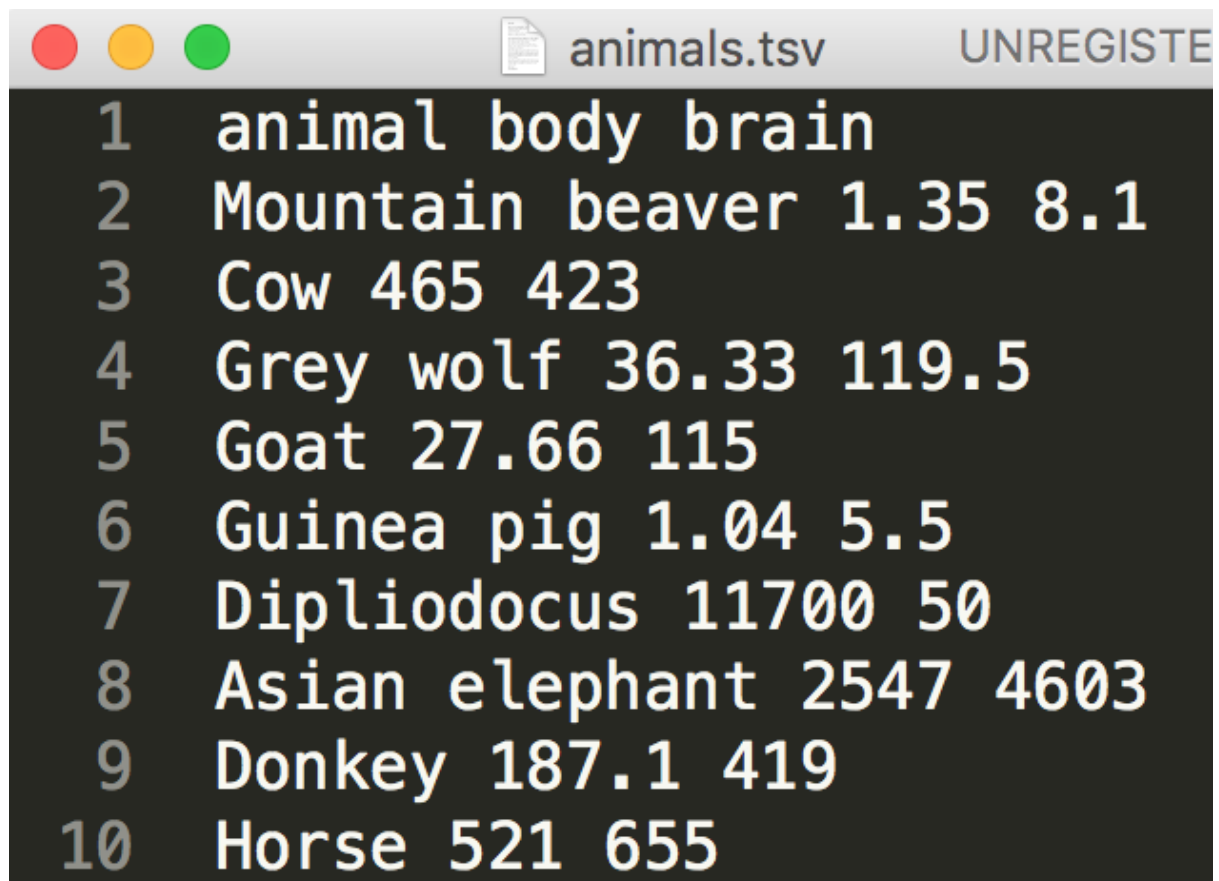
- Lack of standards.
- Data redundancy.

- Sharing data can be cumbersome.
- Not great for large datasets (see *"When does small become large?"* in the Cornell link in *More Information*).

Examples of flat files

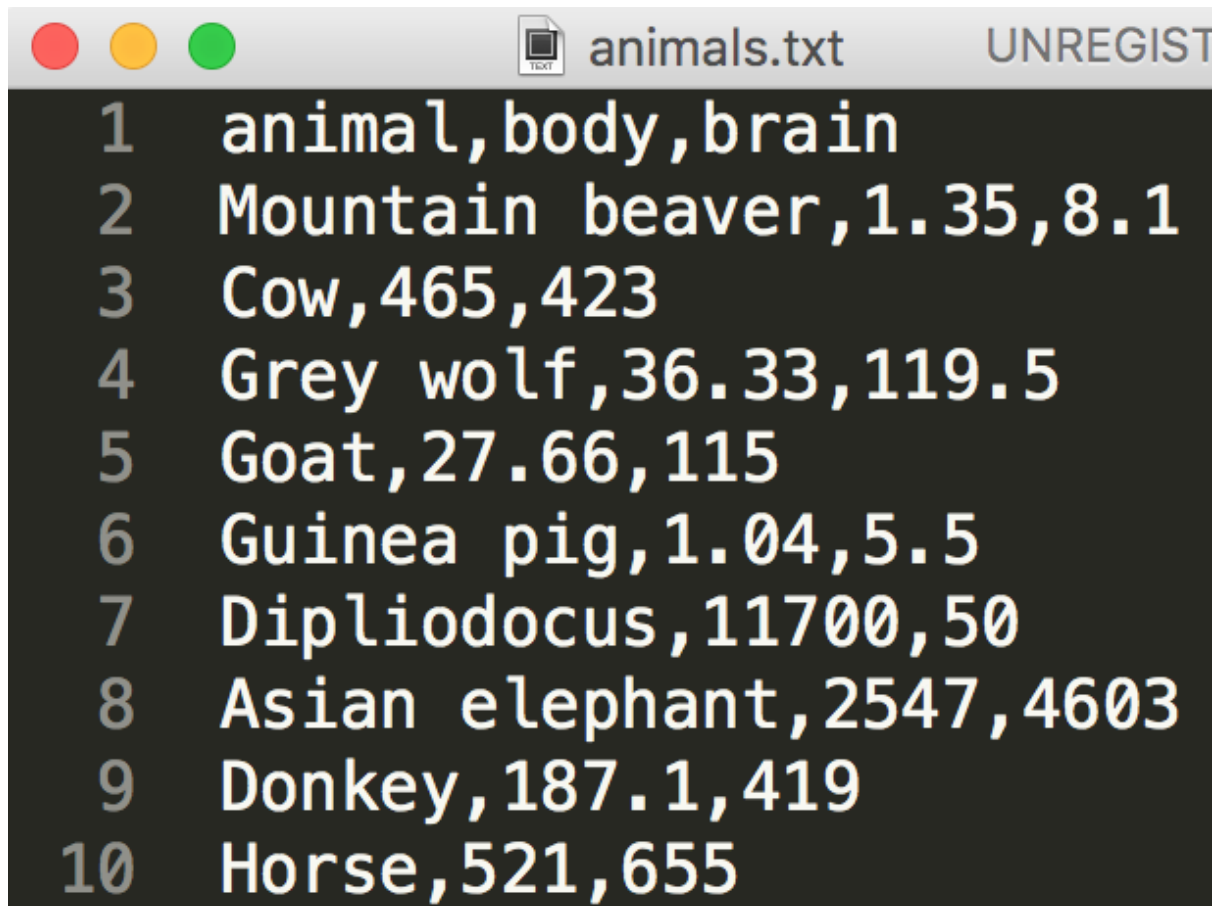


```
animals.csv  UNREGISTE
1  "animal","body","brain"
2  "Mountain beaver",1.35,8.1
3  "Cow",465,423
4  "Grey wolf",36.33,119.5
5  "Goat",27.66,115
6  "Guinea pig",1.04,5.5
7  "Dipliodocus",11700,50
8  "Asian elephant",2547,4603
9  "Donkey",187.1,419
10 "Horse",521,655
```



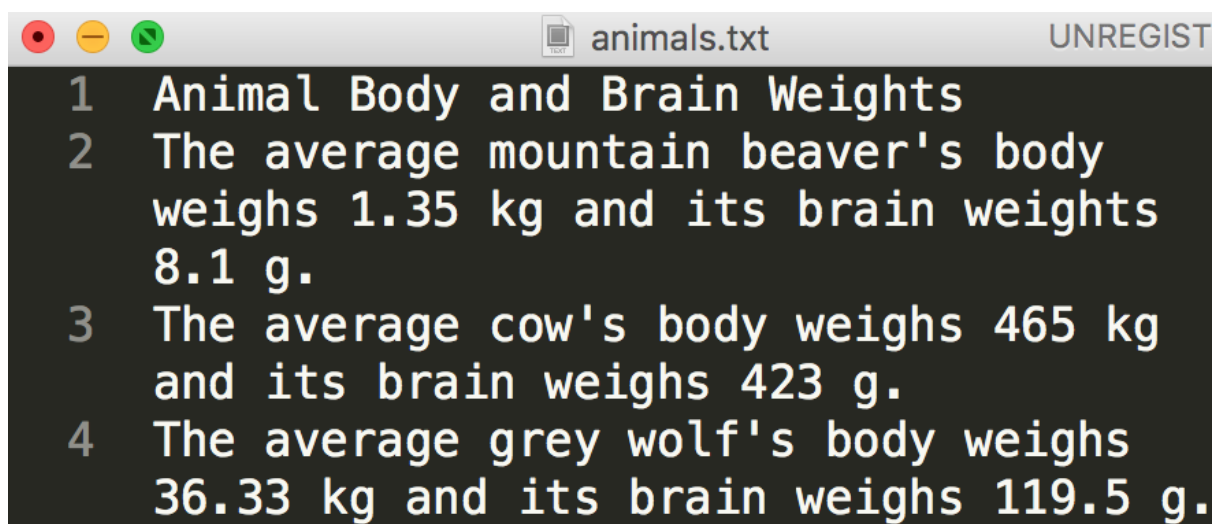
The image shows a screenshot of a text editor window. The title bar at the top has three colored window control buttons (red, yellow, green) on the left, a document icon and the filename 'animals.tsv' in the center, and the text 'UNREGISTE' on the right. The main area of the window has a dark background and displays 10 lines of text in a monospaced font. Each line is numbered from 1 to 10 on the left. The text represents a table with 4 columns: animal name, body weight, brain weight, and an unlabeled numerical value.

	animal	body	brain	
1	Mountain beaver	1.35	8.1	
2	Cow	465	423	
3	Grey wolf	36.33	119.5	
4	Goat	27.66	115	
5	Guinea pig	1.04	5.5	
6	Dipliodocus	11700	50	
7	Asian elephant	2547	4603	
8	Donkey	187.1	419	
9	Horse	521	655	
10				



```
1 animal,body,brain
2 Mountain beaver,1.35,8.1
3 Cow,465,423
4 Grey wolf,36.33,119.5
5 Goat,27.66,115
6 Guinea pig,1.04,5.5
7 Dipliodocus,11700,50
8 Asian elephant,2547,4603
9 Donkey,187.1,419
10 Horse,521,655
```

Example of not flat file



```
1 Animal Body and Brain Weights
2 The average mountain beaver's body
  weighs 1.35 kg and its brain weights
  8.1 g.
3 The average cow's body weighs 465 kg
  and its brain weighs 423 g.
4 The average grey wolf's body weighs
  36.33 kg and its brain weighs 119.5 g.
```

Flat Files in python

in python we use pandas to handle file and datasets , pandas has a generalized approach towards flat files , one can simply use **read_csv()**

function to read all kinds of flat files just by setting the **sep** argument = the separator used.

Web Scrapping

It's the process of getting of extracting data from HTML pages using parsers and scripts

The two main ways to work with HTML files are:

- Saving the HTML file to your computer (using the **Requests** library for example) library and reading that file into a **BeautifulSoup** constructor
- Reading the HTML response content directly into a **BeautifulSoup** constructor (again using the Requests library for example)

Text Files in Python

glob : The glob module finds all the pathnames matching a specified pattern according to the rules used by the Unix shell, although results are returned in arbitrary order. No tilde expansion is done, but *, ?, and character ranges expressed with [] will be correctly matched.

for creating a pandas dataframe the best method is to create a list and then append each feature using dictionary like `df_list.append({'title':title, 'var_name':var_value ...})`

then this list can be converted to dataframe

So scrapping is fun and labourous. but the point here is if the data is not available or the some features are missing then scrapping should only used when necessary , otherwise the use of APIs should be preferred. because they are easier and time saving.

APIs - Application programming interface:

here they are used to get data from different sites and organisations for educational purposes.

we are fetching data from wikipedia using wptools.

JSON DATA TYPE	PYTHON DATA STRUCTURE
JSON array	Python list
JSON object	Python dictionary

APIs that return JSON response in python the entire response can be treated like a dictionary in python and the items inside in the dictionary can be treated as lists.

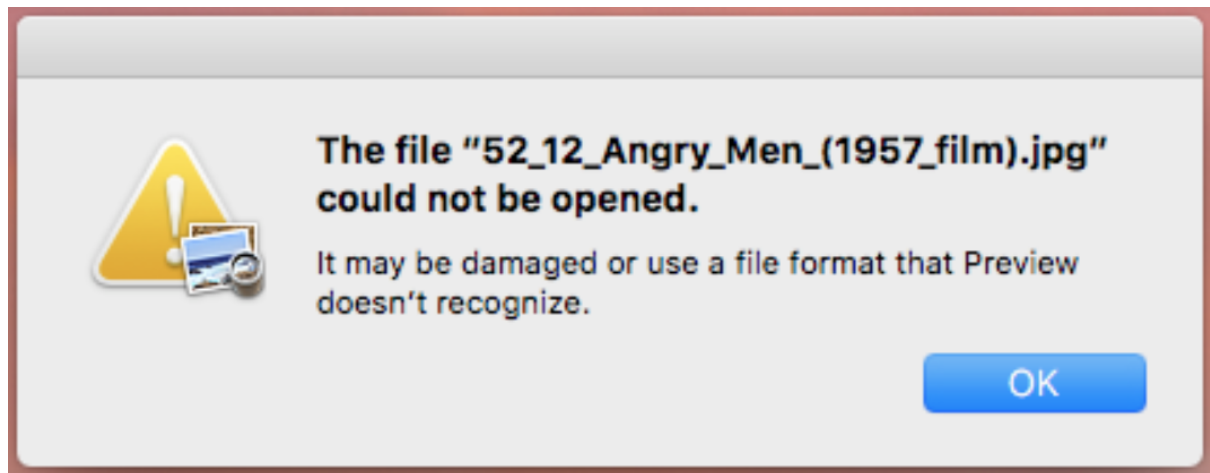
pandas also has JSON functions (the `read_json` function and the `to_json` DataFrame method), but the hierarchical advantage of JSON is wasted in pandas' tabular DataFrame so the uses are limited.

Downloading Image Files

Downloading images may seem tricky from a reading and writing perspective, in comparison to text files which you can read line by line, for example. But in reality, image files aren't special—they're just binary files. To interact with them, you don't need special software (like Photoshop or something) that "understands" images. You can use regular file opening, reading, and writing techniques, like this:

```
import requests
r = requests.get(url)
with open(folder_name + '/' + filename, 'wb') as f:
    f.write(r.content)
```

But this technique can be error-prone. It will work most of the time, but sometimes the file you write to will be damaged. This happened to me when preparing this lesson:



This type of error is why the *requests* library maintainers recommend using the *PIL* library (short for Pillow) and `BytesIO` from the *io* library for non-text requests, like images. They recommend that you access the response body as bytes, for non-text requests. For example, to create an image from binary data returned by a request:

```
import requests
from PIL import Image
from io import BytesIO
r = requests.get(url)
i = Image.open(BytesIO(r.content))
```

Though you may still encounter a similar file error, this code above will at least warn us with an error message, at which point we can manually download the problematic images.

Storing Data

Storing is usually done after cleaning, but it's not always done, which excludes it from being a core part of the data wrangling process. Sometimes you just analyze and visualize and leave it at that, without saving your new data.

sometimes the data is stored into file or sometimes it is stored into databases which are then used for fetching data.

SQL - Structured Query Language, advantages :- easy to replicate, and can work on large data.

It can help in answering more deeper and complex questions.

Why databases ?

- ▼ provides storage.
- ▼ provides concurrency
- ▼ data integrity
- ▼ much faster than flat files

How relational Databases store data ?

Tables - rows and columns

Database tables are organized by columns and each column has a unique name. and the datatype for every value in that column is same.

Types of SQL statements

Create - creates a table in database.

Drop - delete a table or column.

Select - read data and display it.

Relational Databases in Python

Data Wrangling and Relational Databases

In the context of data wrangling, we recommend that databases and SQL only come into play for gathering data or storing data. That is:

- **Connecting to a database and importing data** into a pandas DataFrame (or the analogous data structure in your preferred programming language), then assessing and cleaning that data, or
- **Connecting to a database and storing data** you just gathered (which could potentially be from a database), assessed, and cleaned

These tasks are especially necessary when you have large amounts of data, which is where SQL and other databases excel over flat files.

The two scenarios above can be further broken down into three main tasks:

- Connecting to a database in Python
- Storing data **from** a pandas DataFrame **in** a database to which you're connected, and
- Importing data **from** a database to which you're connected **to** a pandas DataFrame

Other File Formats

The types of files you mastered in this lesson are the ones you'll interact with for the vast majority of your wrangling projects in the future. Again, these were:

- Flat files (e.g. CSV and TSV)
- HTML files
- JSON files
- TXT files
- Relational database files

Additional, less common file formats include:

- Excel files
- Pickle files
- HDF5 files
- SAS files
- STATA files

pandas has functions to read (and write, to most of them) these files. Also, you now have the foundational understanding of **gathering** and file formats in general, so learning these additional formats won't be too hard if you need them.

Chapter 3 : Assessing Data

we'll assess for data quality and tidiness for the data we have gathered. This can be done visually or programmatically, but the important point is to take notes on what problems you have assessed, for any given dataset.

Assessing is the precursor to cleaning. You can't clean something that you don't know exists! In this lesson, you'll learn to identify and categorize common data quality and tidiness issues.

Dirty Data vs. Messy Data

Dirty data, also known as low quality data. Low quality data has content issues.

Messy data, also known as untidy data. Untidy data has structural issues.

Both assessment and cleaning can be done together for example, you spot an issue and clean it. but it is generally preferred that you make all the assessments first so that you know what issues the dataset had then clean the dataset.

There's a slight difference between assessing and exploring the data, exploring is a part of EDA (Exploratory Data Analysis) and assessing is a part of data wrangling.

Assessment you make won't make your model better it will just make your data clean and will help in addressing tidiness or dirty data.

Exploring helps in finding outliers, trends etc in the data that can help to make our model, graphs etc to be better.

Data Quality Dimensions

Data quality dimensions help guide your thought process while assessing and also cleaning. The four main data quality dimensions are:

- **Completeness:** do we have all of the records that we should? Do we have missing records or not? Are there specific rows, columns, or cells missing?
- **Validity:** we have the records, but they're not valid, i.e., they don't conform to a defined schema. A schema is a defined set of rules for data. These rules can be real-world constraints

(e.g. negative height is impossible) and table-specific constraints (e.g. unique key constraints in tables).

- **Accuracy:** inaccurate data is wrong data that is valid. It adheres to the defined schema, but it is still incorrect. Example: a patient's weight that is 5 lbs too heavy because the scale was faulty.
- **Consistency:** inconsistent data is both valid and accurate, but there are multiple *correct* ways of referring to the same thing. Consistency, i.e., a standard format, in columns that represent the same data across tables and/or within tables is desired.

the additional dimensions are super specific cases of these four dimensions listed above. Example: *currency*, defined as follows: *the degree to which data is current with the world that it models. Currency can measure how up-to-date data is.* Currency is a specific case of accuracy data in the sense that out-of-date data is (usually) valid but wrong. In other words, our definition of accuracy can include currency

Submit to check your answer choices!

DATA QUALITY ISSUE	DATA QUALITY DIMENSION
'Dsvld' given name typo in the <i>patients</i> table	Accuracy
'u' next to start dose and end dose in the <i>treatments</i> table	Validity
Lowercase given names and surnames in the <i>treatments</i> and <i>adverse_reactions</i> table	Consistency
280 records in the <i>treatments</i> table instead of 350	Completeness

Sources of Dirty Data

Dirty data = low quality data = content issues

There are lots of sources of dirty data. Basically, anytime humans are involved, there's going to be dirty data. There are lots of ways in which we touch data we work with.

- We're going to have user entry errors.
- In some situations, we won't have any data coding standards, or where we do have standards they'll be poorly applied, causing problems in the resulting data
- We might have to integrate data where different schemas have been used for the same type of item.
- We'll have legacy data systems, where data wasn't coded when disc and memory constraints were much more restrictive than they are now. Over time systems evolve. Needs change, and data changes.
- Some of our data won't have the unique identifiers it should.
- Other data will be lost in transformation from one format to another.
- And then, of course, there's always programmer error.
- And finally, data might have been corrupted in transmission or storage by cosmic rays or other physical phenomenon. So hey, one that's not our fault.

Sources of Messy Data

Messy data = untidy data = structural issues

Messy data is usually the result of poor data planning. Or a lack of awareness of the benefits of tidy data. Fortunately, messy data is usually much more easily addressable than most of the sources of dirty data mentioned above.

Chapter 4: Cleaning Data

Steps involved :

Define : Define issue in words.

Code : Code for solving that issue.

Test : Test that your code worked.

Manual vs Programmatic Cleaning

only one off occurrences should be cleaned manually.

One should focus on automating this task and save as much time as possible.

The Process

The very first thing to do before any cleaning occurs is to make a copy of each piece of data. All of the cleaning operations will be conducted on this copy so you can still view the original dirty and/or messy dataset later. Copying DataFrames in pandas is done using the `copy_` method. If the original DataFrame was called `df`, the soon-to-be clean copy of the dataset could be named `df_clean`.

```
df_clean = df.copy()
```

Note that simply assigning a DataFrame to a new variable name leaves the original DataFrame vulnerable to modifications

Clean

```
In [4]: df_clean = df.copy()
```

Define

- Remove 'bb' before every animal name using string slicing
- Replace ! with . in body weight and brain weight columns

Code

```
In [5]: # Remove 'bb' before every animal name using string slicing
df_clean['Animal'] = df_clean['Animal'].str[2:]
```

```
In [6]: # Replace ! with . in body weight and brain weight columns
df_clean['Body weight (kg)'] = df_clean['Body weight (kg)'].str.replace('!', '.')
df_clean['Brain weight (g)'] = df_clean['Brain weight (g)'].str.replace('!', '.')
```

Test

```
In [7]: df_clean.head()
```

```
Out[7]:
```

	Animal	Body weight (kg)	Brain weight (g)
0	Mountain beaver	1.35	8.1
1	Cow	465	423
2	Grey wolf	36.33	119.5
3	Goat	27.66	115
4	Guinea pig	1.04	5.5

either this way or method given below

Clean

bb before every animal name

Define

Remove 'bb' before every animal name using string slicing.

Code

```
In [ ]: df_clean['Animal'] = df_clean['Animal'].str[2:]
```

Test

```
In [ ]: df_clean.Animal.head()
```

! instead of . for decimal in body weight and brain weight

Define

Replace ! with . in body weight and brain weight columns

Code

```
In [ ]: df_clean['Body weight (kg)'] = df_clean['Body weight (kg)'].str.replace('!', '.')
df_clean['Brain weight (g)'] = df_clean['Brain weight (g)'].str.replace('!', '.')
```

Test

```
In [ ]: df_clean.head()
```

Addressing Missing Data : Completeness issues.

Cleaning for Tidiness , tidy dataset are easy to work with , first the tidiness should be fixed then clean the quality issues.

The programmatic data cleaning process:

1. Define: convert our assessments into defined cleaning tasks. These definitions also serve as an instruction list so others (or yourself in the future) can look at your work and reproduce it.
2. Code: convert those definitions to code and run that code.
3. Test: test your dataset, visually or with code, to make sure your cleaning operations worked.

Always make copies of the original pieces of data before cleaning!