The Data Analysis Process

Problems Solved by Data Analysts

Before we jump into the data analysis process, let's talk about some of the kinds of problems you might solve as a data analyst. When many people hear the term data analyst, they picture someone who works at a tech company and probably in Silicon Valley. Many data analysts do in fact work at tech companies.

- For example, Netflix uses data analysis to provide personalized movie recommendations for its users.
- Facebook uses it in its news feed ranking algorithm.
- And OkCupid uses it to predict which people are likely to be good romantic matches.

In addition to building these kinds of systems, many companies also use data analysis to publish papers or write blog posts about their findings.

For example, Facebook has studied the ideological diversity of people's political posts. They found that when posts appear on people's news feed written from a different perspective, users are less likely to click on those posts.

OkCupid wrote a blog post about what are the best questions to ask someone on a first date. Did you know that long term couples are very likely to agree about whether they like horror movies? Data analysts don't just work at tech companies though.

Wal-Mart looks through their purchase records and people's posts on social media in order to figure out what to stock in their stores.

Apparently, Strawberry Pop-Tarts are more likely to sell right before a hurricane.

Bill James is famous for applying data analysis to baseball. He used it to understand who the top performers are and how to best predict future performance.

And pharmaceutical companies have started to use machine learning to predict which chemical compounds are most likely to make effective drugs.

Using this, they can make better decisions about which compounds are likely to be a good use of the resources involved in running a clinical trial.

If you'd like to learn more about any of these applications, check out the links in the instructor notes.

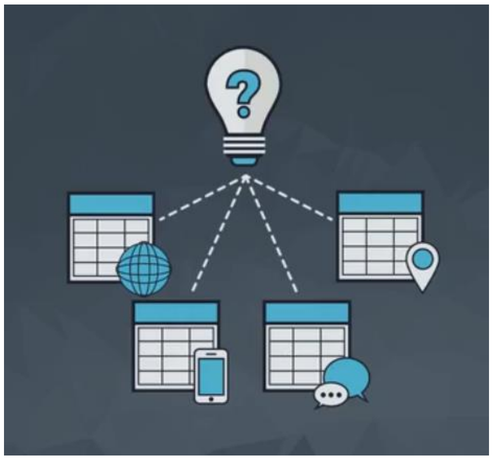
Data Analysis Process Overview

So what is this data analysis process? We broke it up into five steps: question, wrangle, explore, draw conclusions and communicate. This process will help you understand, explore and use your data intelligently so that you make the most of the information you're given. Whether you're building your reporting dashboard, analyzing A-B test results or doing more advanced analysis with machine learning and AI.

Let's see what each step is about.

Step 1: Ask questions

Either you're given data and ask questions based on it, or you ask questions first and gather data based on that later. In both cases, great questions help you focus on relevant parts of your data and direct your analysis towards meaningful insights.

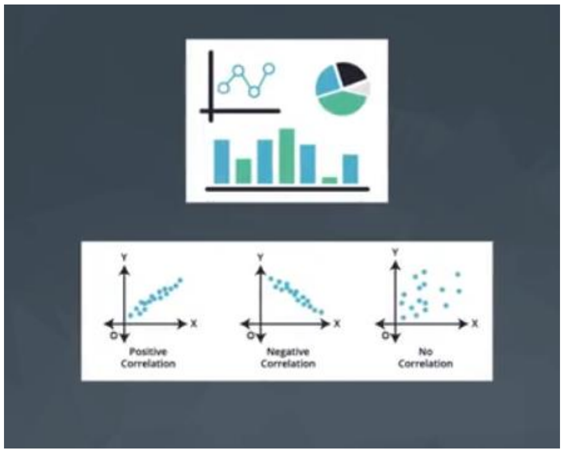


Step 2: Wrangle data

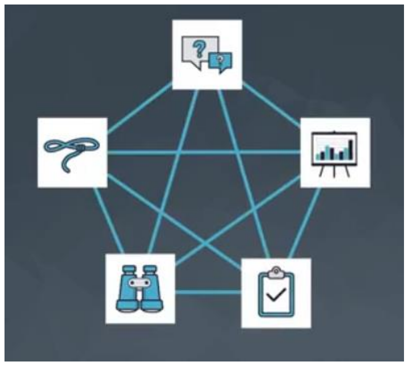
You get the data you need in a form you can work with in three steps: gather, assess, clean. You gather the data you need to answer your questions, assess your data to identify any problems in your data's quality or structure, and clean your data by modifying, replacing, or removing data to ensure that your dataset is of the highest quality and as well-structured as possible.

Step 3: Perform EDA (Exploratory Data Analysis)

You explore and then augment your data to maximize the potential of your analyses, visualizations, and models. Exploring involves finding patterns in your data, visualizing relationships in your data, and building intuition about what you're working with. After exploring, you can do things like remove outliers and create better features from your data, also known as feature engineering.



As you become more familiar with your data in this EDA step, you'll often revisit previous steps. For example, you might discover new problems in your data and go back to wrangle them. Or you might discover exciting, unexpected patterns and decide to refine your questions. The data analysis process isn't always linear.



This exploratory step in particular is very intertwined with the rest of the process. It's usually where you discover and learn the most about your data.

Step 4: Draw conclusions (or even make predictions)

This step is typically approached with machine learning or inferential statistics that are beyond the scope of this course, which will focus on drawing conclusions with descriptive statistics.

Step 5: Communicate your results

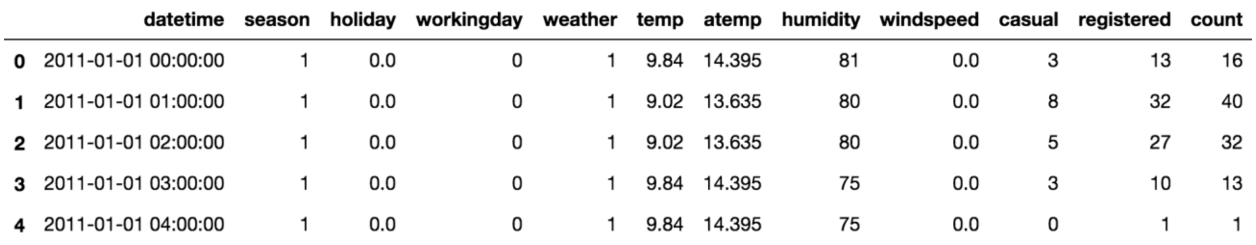
You often need to justify and convey meaning in the insights you've found. Or, if your end goal is to build a system, you usually need to share what you've built, explain how you reached design decisions, and report how well it performs. There are many ways to communicate your results: reports, slide decks, blog posts, emails, presentations, or even conversations. Data visualization will always be very valuable.

Before walking through each of these steps with real datasets using Python, let's build a bit of intuition for each step!

Data Analysis Process Quiz

Let's build some intuition for the steps in the data analysis process with some questions you might have at each step. We'll use data from **Kaggle's Bike Sharing Demand(opens in a new tab)** competition. In this dataset, you are given hourly rental data spanning two years from the Capital Bikeshare program in Washington, D.C. Below is a screenshot of the first 5 rows of this dataset. *Note:* This dataset has been slightly modified.

Note: Solutions to all quiz questions are provided in downloadable link at bottom of page.



	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 00:00:00	1	0.0	0	1	9.84	14.395	81	0.0	3	13	16
1	2011-01-01 01:00:00	1	0.0	0	1	9.02	13.635	80	0.0	8	32	40
2	2011-01-01 02:00:00	1	0.0	0	1	9.02	13.635	80	0.0	5	27	32
3	2011-01-01 03:00:00	1	0.0	0	1	9.84	14.395	75	0.0	3	10	13
4	2011-01-01 04:00:00	1	0.0	0	1	9.84	14.395	75	0.0	0	1	1

Click on images to make them larger!

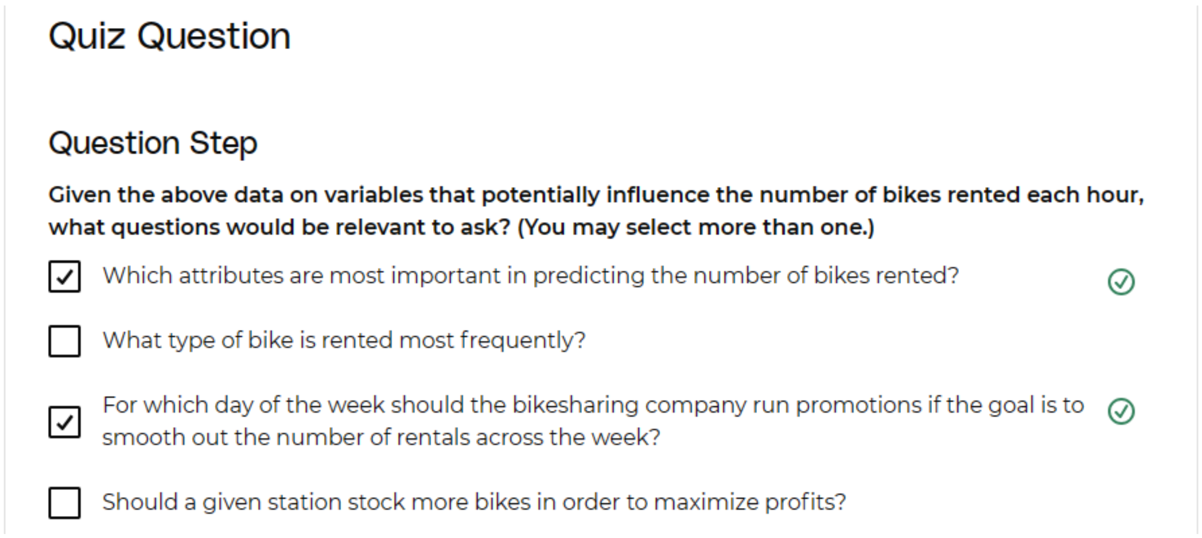
Use the table below to help you answer the first question. Each feature is a column in the dataset.

Feature	**Description**
datetime	hourly date + timestamp
season	1 = spring, 2 = summer, 3 = fall, 4 = winter
holiday	whether the day is considered a holiday
workingday	whether the day is neither a weekend nor holiday
weather *	1, 2, 3, 4 (see descriptions below)
temp	temperature in Celsius
atemp	"feels like" temperature in Celsius
humidity	relative humidity

Feature	Description
windspeed	wind speed
casual	number of non-registered user rentals initiated
registered	number of registered user rentals initiated
count	number of total rentals

*** Keys for Weather Feature**

1 = clear, few clouds, partly cloudy, partly cloudy
2 = mist + cloudy, mist + broken clouds, mist + few clouds, mist
3 = light snow, light rain + thunderstorm + scattered clouds, light rain + scattered clouds
4 = heavy rain + ice pallets + thunderstorm + mist, snow + fog

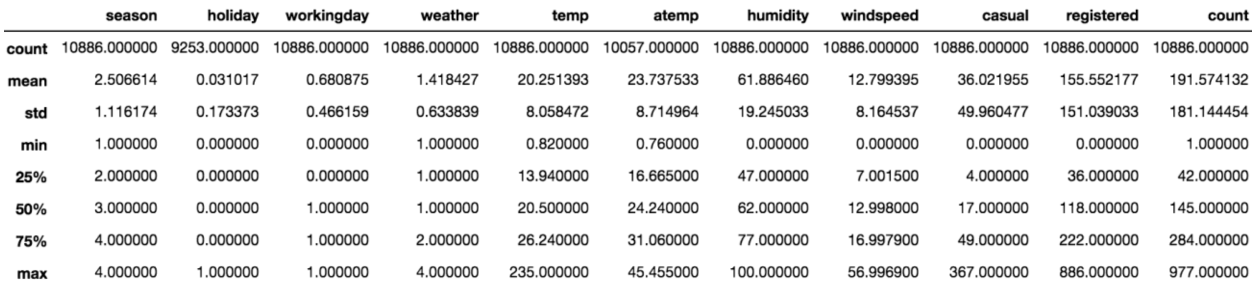


Quiz Question

Question Step

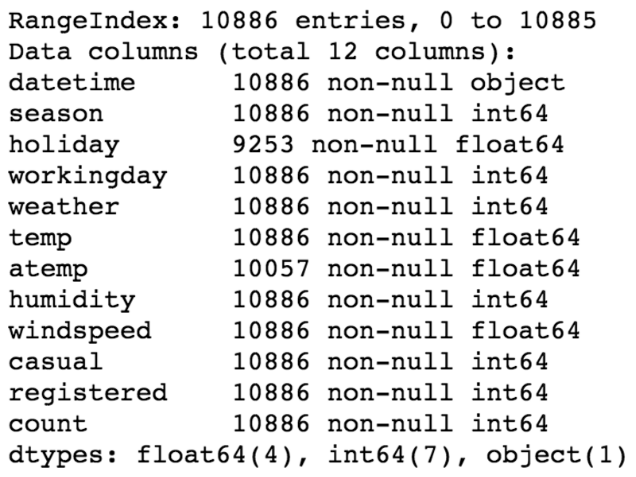
Given the above data on variables that potentially influence the number of bikes rented each hour, what questions would be relevant to ask? (You may select more than one.)

- [✓] Which attributes are most important in predicting the number of bikes rented? ✓
- [ ] What type of bike is rented most frequently?
- [✓] For which day of the week should the bikesharing company run promotions if the goal is to smooth out the number of rentals across the week? ✓
- [ ] Should a given station stock more bikes in order to maximize profits?



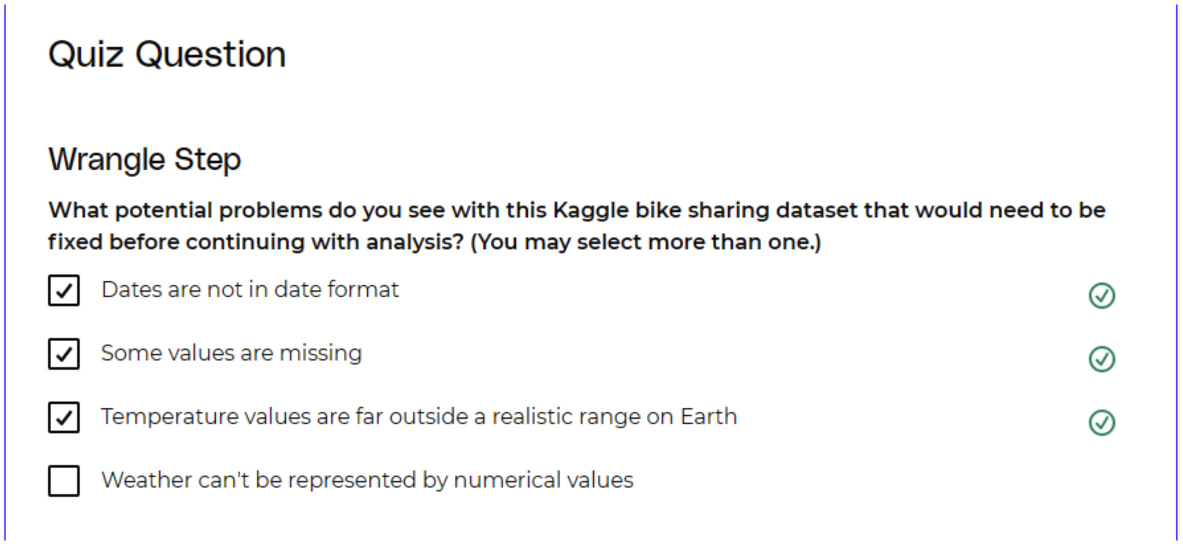
	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
count	10886.000000	9253.000000	10886.000000	10886.000000	10886.000000	10057.000000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000
mean	2.506614	0.031017	0.680875	1.418427	20.251393	23.737533	61.886460	12.799395	36.021955	155.552177	191.574132
std	1.116174	0.173373	0.466159	0.633839	8.058472	8.714964	19.245033	8.164537	49.960477	151.039033	181.144454
min	1.000000	0.000000	0.000000	1.000000	0.820000	0.760000	0.000000	0.000000	0.000000	0.000000	1.000000
25%	2.000000	0.000000	0.000000	1.000000	13.940000	16.665000	47.000000	7.001500	4.000000	36.000000	42.000000
50%	3.000000	0.000000	1.000000	1.000000	20.500000	24.240000	62.000000	12.998000	17.000000	118.000000	145.000000
75%	4.000000	0.000000	1.000000	2.000000	26.240000	31.060000	77.000000	16.997900	49.000000	222.000000	284.000000
max	4.000000	1.000000	1.000000	4.000000	235.000000	45.455000	100.000000	56.996900	367.000000	886.000000	977.000000

Helpful stats on each column for the next question



```
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 12 columns):
datetime      10886 non-null object
season        10886 non-null int64
holiday       9253 non-null float64
workingday    10886 non-null int64
weather       10886 non-null int64
temp          10886 non-null float64
atemp         10057 non-null float64
humidity      10886 non-null int64
windspeed     10886 non-null float64
casual        10886 non-null int64
registered    10886 non-null int64
count         10886 non-null int64
dtypes: float64(4), int64(7), object(1)
```

# Non-null values and datatype for each column

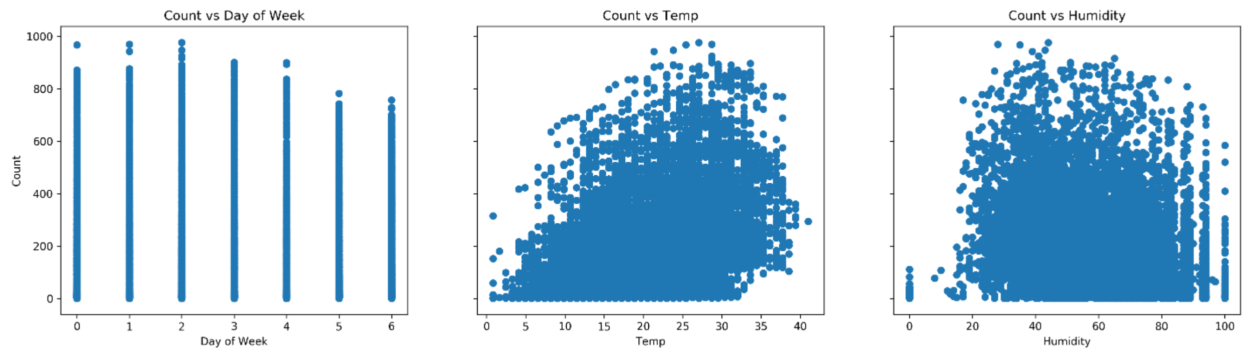


Quiz Question

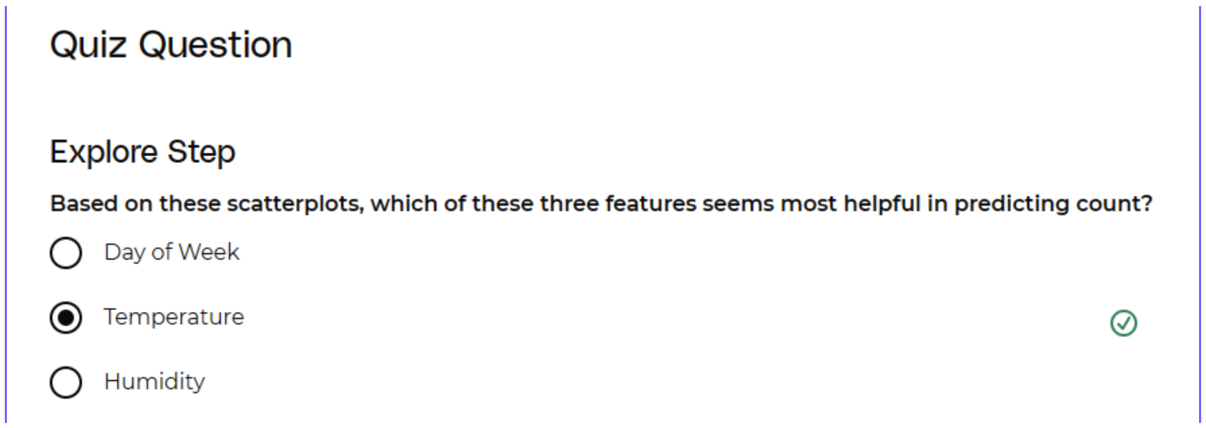
Wrangle Step

What potential problems do you see with this Kaggle bike sharing dataset that would need to be fixed before continuing with analysis? (You may select more than one.)

- [✓] Dates are not in date format ✓
- [✓] Some values are missing ✓
- [✓] Temperature values are far outside a realistic range on Earth ✓
- [ ] Weather can't be represented by numerical values



Scatterplots on the number of bikes rented vs. day of week, temperature, and humidity

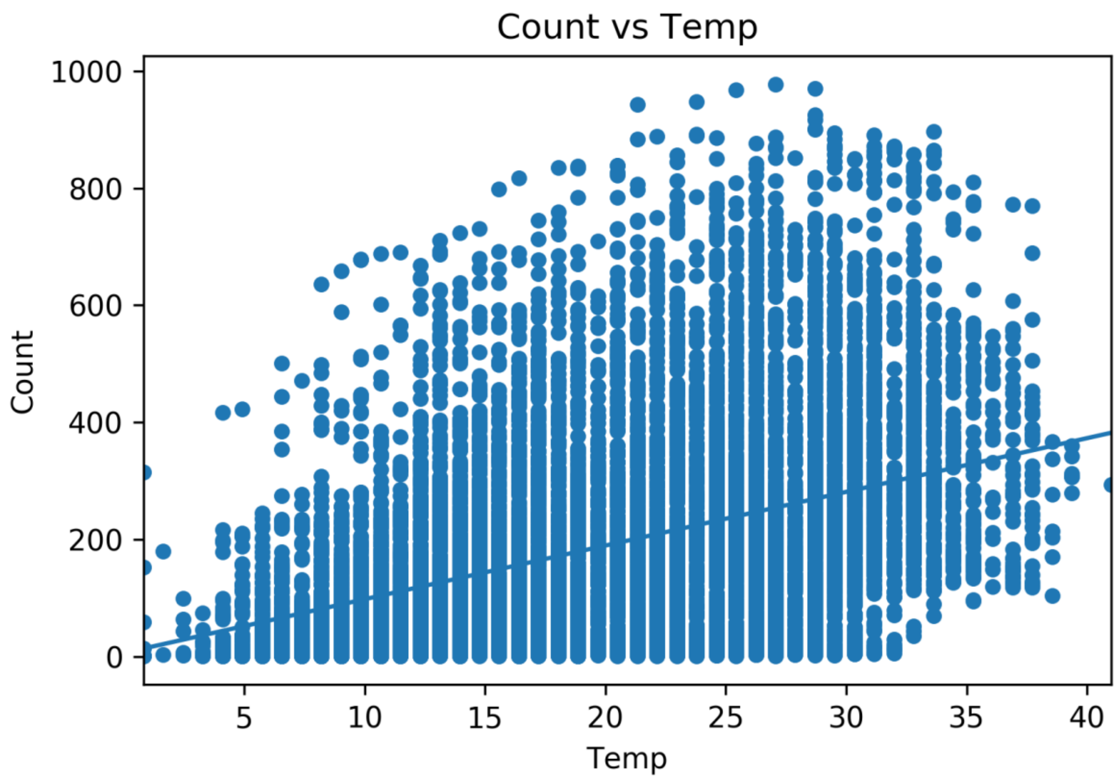


Quiz Question

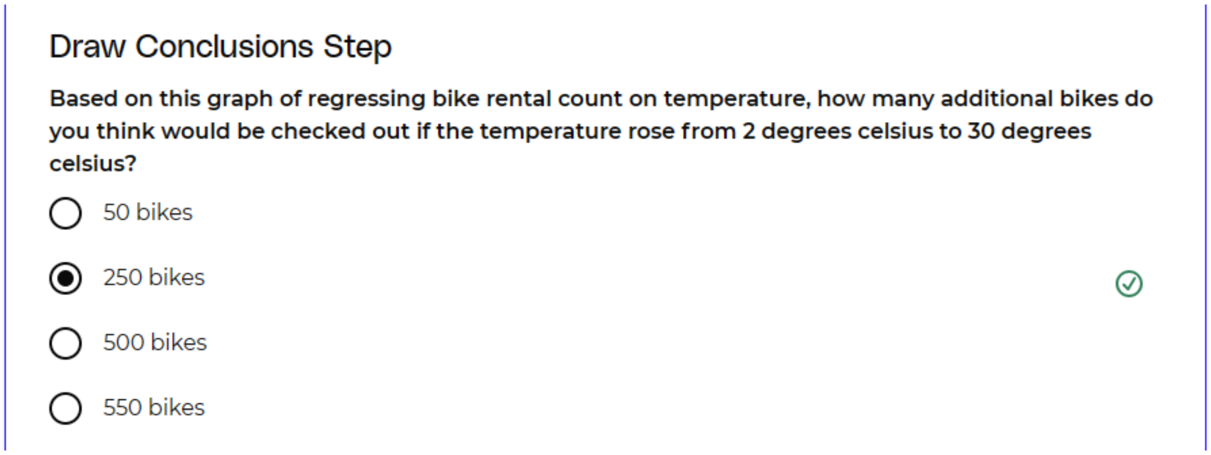
Explore Step

Based on these scatterplots, which of these three features seems most helpful in predicting count?

- Day of Week
- [✓ Temperature] ✓
- Humidity



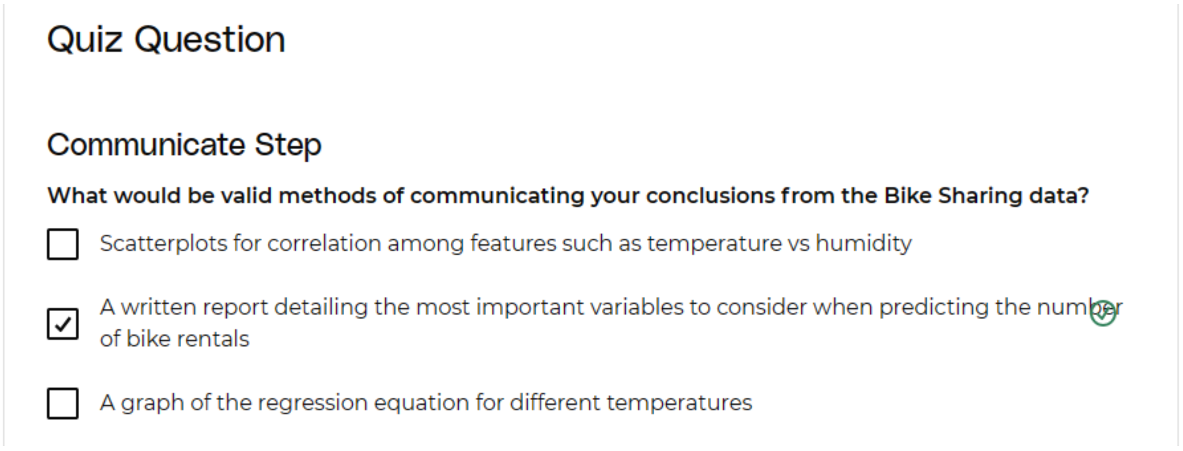
Scatter plot of count vs. temperature with a line of best fit for the next question



Draw Conclusions Step

Based on this graph of regressing bike rental count on temperature, how many additional bikes do you think would be checked out if the temperature rose from 2 degrees celsius to 30 degrees celsius?

- ○ 50 bikes
- [✓ 250 bikes] ✓
- ○ 500 bikes
- ○ 550 bikes



Quiz Question

Communicate Step

What would be valid methods of communicating your conclusions from the Bike Sharing data?

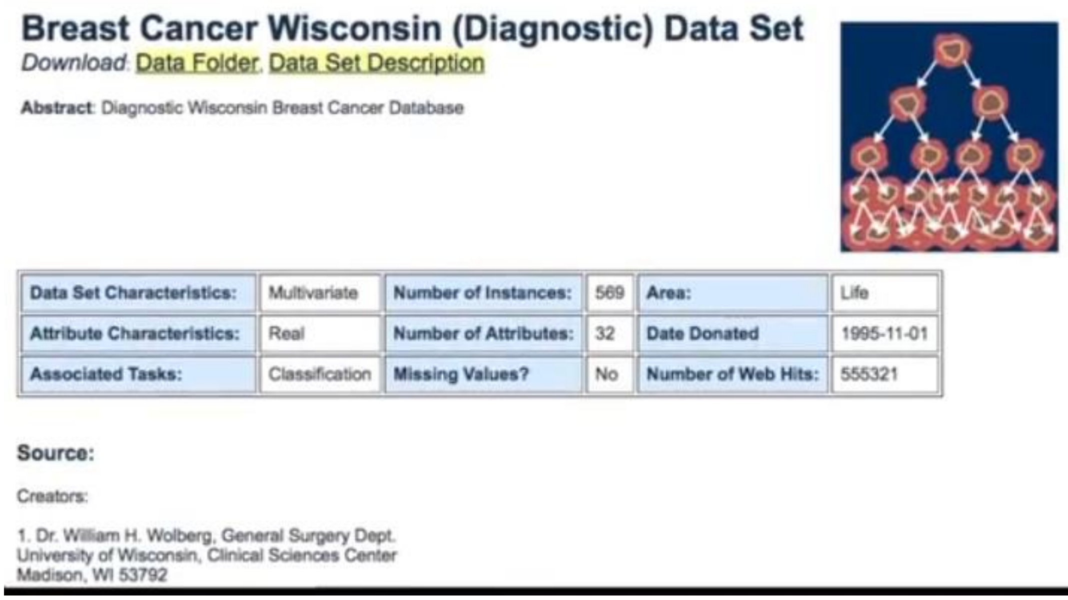
- [ ] Scatterplots for correlation among features such as temperature vs humidity
- [✓] A written report detailing the most important variables to consider when predicting the number of bike rentals
- [ ] A graph of the regression equation for different temperatures

Great job! Next, let's learn about Python packages, which were used to prepare this quiz!

Asking Questions

Let's get started with the first step of the data analysis process: asking questions. If you remember from the overview video, sometimes we ask our questions first and use them to direct what types of data we'll get later. Other times, we'll get our data first and ask questions based on it. Here, we're going to practice creating questions with the real data set.

This is data from the University of Wisconsin General Surgery Lab and the measurements of tumors extracted during biopsies along with the designation of whether or not the tumor was cancerous.



Breast Cancer Wisconsin (Diagnostic) Data Set

Download: Data Folder, Data Set Description

Abstract: Diagnostic Wisconsin Breast Cancer Database

Data Set Characteristics:	Multivariate	**Number of Instances:**	569	**Area:**	Life
Attribute Characteristics:	Real	**Number of Attributes:**	32	**Date Donated**	1995-11-01
Associated Tasks:	Classification	**Missing Values?**	No	**Number of Web Hits:**	555321

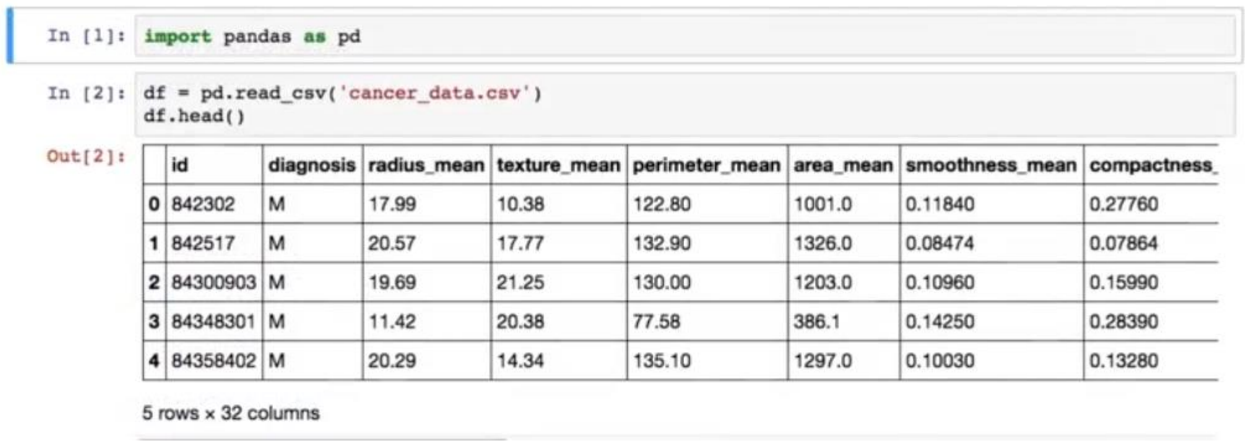
Source:

Creators:

1. Dr. William H. Wolberg, General Surgery Dept.
University of Wisconsin, Clinical Sciences Center
Madison, WI 53792

A natural question would be: how do we know if a tumor is cancerous? This is a pretty broad question. It will be helpful to formulate intermediate questions to guide our analysis.

Here is the Notebook I made, which you'll be working through later.



```
In [1]: import pandas as pd

In [2]: df = pd.read_csv('cancer_data.csv')
        df.head()
```

Out[2]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280

5 rows × 32 columns

Take a closer look at the features and think about questions that would help you learn more about these features and their relationships in the data.

One example would be, does the size of the tumor determine its malignancy? You'll find out more information about this data set in the instructor notes below.

What additional questions you have about the data? How do you go about analyzing the data?

Breast Cancer Wisconsin (Diagnostic) Dataset from UCI Machine Learning Lab

(The dataset is included in the workspace here for you as "cancer_data.csv." If you're interested, you can explore it further **here, on Kaggle(opens in a new tab)**.)

Attribute Information:

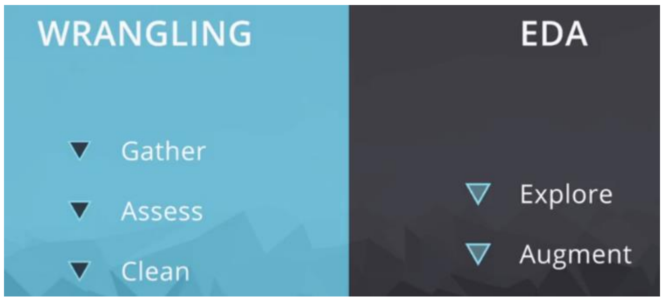
1. ID number
2. Diagnosis (M = malignant, B = benign)
3. 30 features

The following ten features are computed for each cell nucleus. For each of these ten features, a column is created for the mean, standard error, and max value.

Feature	**Description**
Radius	Mean of distances from center to points on the perimeter
Texture	Standard deviation of gray-scale values
Perimeter	
Area	
Smoothness	Local variation in radius lengths
Compactness	Perimeter^2 / Area - 1.0
Concavity	Severity of concave portions of the contour
Concave Points	Number of concave portions of the contour
Symmetry	
Fractal Dimension	"Coastline approximation" - 1

Data Wrangling and EDA

Wrangling and EDA are sometimes used anonymously because their purposes often overlap.



In this course, when I say wrangling it means you'll be gathering, assessing, and cleaning your data, but the modifications you make when cleaning the dataset won't make your analysis, visualizations, or models better, they just make them work. Alternatively, EDA means you'll be exploring and augmenting your data to maximize the potential of your analysis, visualizations, and models.

One example is engineering new features or moving outliers, they can make visualizations more impactful, for instance, removing outliers to zoom into the heart of a distribution or they can improve a model’s R-squared value. We'll dive into this in more detail later in the class, but we'll take a first pass now. And remember, don't worry if you don't get everything right away. We have entire courses that cover each of these steps.

Gathering Data

Let's start with gathering. Data acquisition can happen in a variety of ways.

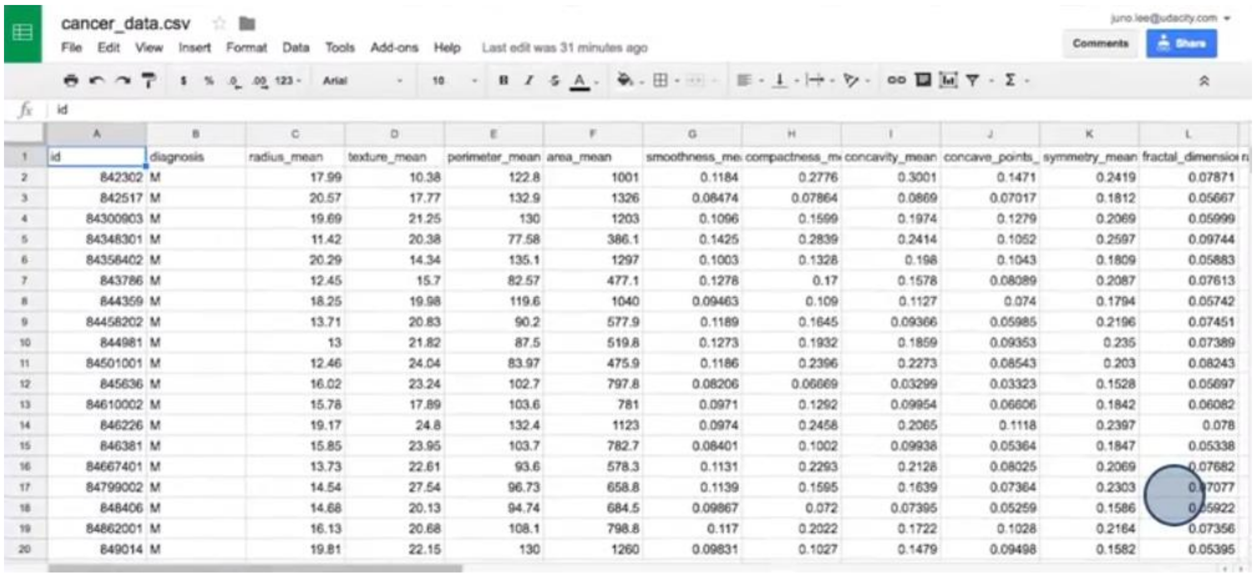
In this case, you already acquired the data by downloading some files that were available. In other cases, you might need to get the data from an API or scrape it from a web page. In many companies, they might have an existing database from which you can pull data and answer questions.

You'll also often need to combine data from multiple different formats. It's okay if some of these terms are unfamiliar to you, they'll be covered in more depth in our data wrangling course which is linked in the instructor notes.

The lesson on gathering data within that course is great if you'd like to learn more about how to get data from a variety of sources. Your files are in a format called CSV which stands for a comma separated values. A CSV file is a text file.

It's similar to a spreadsheet in terms of its tabular structure. But if you've ever seen IF statements or other functions in your spreadsheets, this isn't possible in CSV files. They hold only the raw data. The contents of a CSV file are also very easy to process manually using code. Unlike for example, an XLSX file, which is the format used by Microsoft Excel.

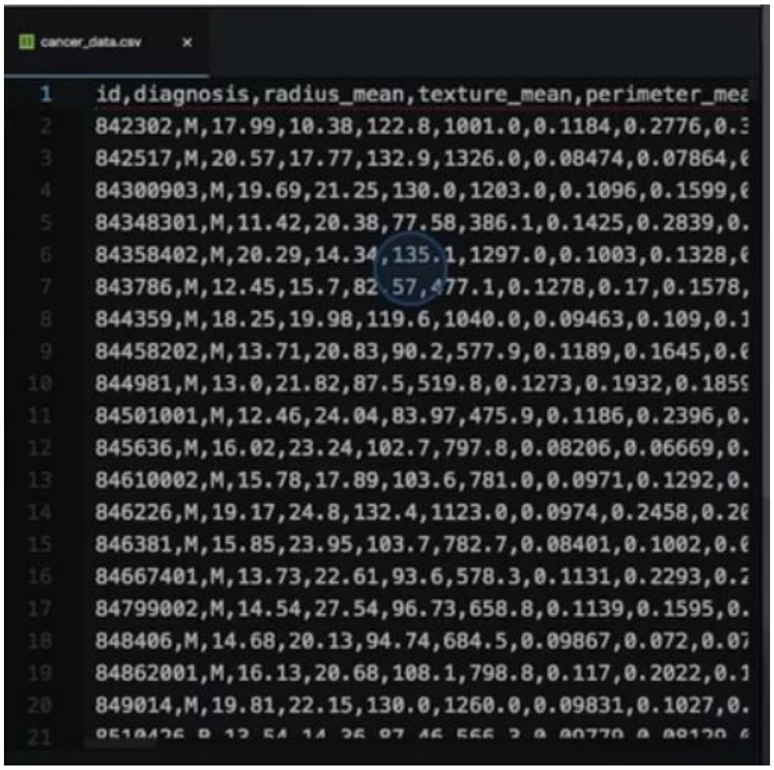
Let's take a look at one of your files which contains cancer data. Here's what the file looks like in Google Spreadsheets. There's a row for each time a tumor was extracted and columns for different pieces of information, such as the ID, diagnosis, radius, texture, area, and smoothness.


cancer_data.csv
File Edit View Insert Format Data Tools Add-ons Help
Last edit was 31 minutes ago
juno.lee@udacity.com
Comments
Share
Arial
10
fx id

	A	B	C	D	E	F	G	H	I	J	K	L
1	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_me	compactness_m	concavity_mean	concave_points_	symmetry_mean	fractal_dimensio
2	842302	M	17.99	10.38	122.8	1001	0.1184	0.2776	0.3001	0.1471	0.2419	0.07871
3	842517	M	20.57	17.77	132.9	1326	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667
4	84300903	M	19.69	21.25	130	1203	0.1096	0.1599	0.1974	0.1279	0.2069	0.05999
5	84348301	M	11.42	20.38	77.58	386.1	0.1425	0.2839	0.2414	0.1052	0.2597	0.09744
6	84358402	M	20.29	14.34	135.1	1297	0.1003	0.1328	0.198	0.1043	0.1809	0.05883
7	843786	M	12.45	15.7	82.57	477.1	0.1278	0.17	0.1578	0.08089	0.2087	0.07613
8	844359	M	18.25	19.98	119.6	1040	0.09463	0.109	0.1127	0.074	0.1794	0.05742
9	84458202	M	13.71	20.83	90.2	577.9	0.1189	0.1645	0.09366	0.05985	0.2196	0.07451
10	844981	M	13	21.82	87.5	519.8	0.1273	0.1932	0.1859	0.09353	0.235	0.07389
11	84501001	M	12.46	24.04	83.97	475.9	0.1186	0.2396	0.2273	0.08543	0.203	0.08243
12	845636	M	16.02	23.24	102.7	797.8	0.08206	0.06669	0.03299	0.03323	0.1528	0.05697
13	84610002	M	15.78	17.89	103.6	781	0.0971	0.1292	0.09954	0.06606	0.1842	0.06082
14	846226	M	19.17	24.8	132.4	1123	0.0974	0.2458	0.2065	0.1118	0.2397	0.078
15	846381	M	15.85	23.95	103.7	782.7	0.08401	0.1002	0.09938	0.05364	0.1847	0.05338
16	84667401	M	13.73	22.61	93.6	578.3	0.1131	0.2293	0.2128	0.08025	0.2069	0.07682
17	84799002	M	14.54	27.54	96.73	658.8	0.1139	0.1595	0.1639	0.07364	0.2303	0.07077
18	848406	M	14.68	20.13	94.74	684.5	0.09867	0.072	0.07395	0.05259	0.1586	0.05922
19	84862001	M	16.13	20.68	108.1	798.8	0.117	0.2022	0.1722	0.1028	0.2164	0.07356
20	849014	M	19.81	22.15	130	1260	0.09831	0.1027	0.1479	0.09498	0.1582	0.05395

Now, here's what the file looks like if you open it in a plain text editor.

A text editor is a program like Notepad, Sublime or Adam, that shows exactly what's present in the file. As you can see, the actual content of the file are very simple.



```
id,diagnosis,radius_mean,texture_mean,perimeter_mea
842302,M,17.99,10.38,122.8,1001.0,0.1184,0.2776,0.
842517,M,20.57,17.77,132.9,1326.0,0.08474,0.07864,
84300903,M,19.69,21.25,130.0,1203.0,0.1096,0.1599,
84348301,M,11.42,20.38,77.58,386.1,0.1425,0.2839,0.
84358402,M,20.29,14.34,135.1,1297.0,0.1003,0.1328,
843786,M,12.45,15.7,82.57,477.1,0.1278,0.17,0.1578,
844359,M,18.25,19.98,119.6,1040.0,0.09463,0.109,0.
84458202,M,13.71,20.83,90.2,577.9,0.1189,0.1645,0.
844981,M,13.0,21.82,87.5,519.8,0.1273,0.1932,0.185
84501001,M,12.46,24.04,83.97,475.9,0.1186,0.2396,0.
845636,M,16.02,23.24,102.7,797.8,0.08206,0.06669,0.
84610002,M,15.78,17.89,103.6,781.0,0.0971,0.1292,0.
846226,M,19.17,24.8,132.4,1123.0,0.0974,0.2458,0.2
846381,M,15.85,23.95,103.7,782.7,0.08401,0.1002,0.
84667401,M,13.73,22.61,93.6,578.3,0.1131,0.2293,0.
84799002,M,14.54,27.54,96.73,658.8,0.1139,0.1595,0.
848406,M,14.68,20.13,94.74,684.5,0.09867,0.072,0.0
84862001,M,16.13,20.68,108.1,798.8,0.117,0.2022,0.
849014,M,19.81,22.15,130.0,1260.0,0.09831,0.1027,0.
8510426,B,13.54,14.36,87.46,566.3,0.09779,0.08129,
```

The header row from the spreadsheet is present as the first line of the CSV file.

The second row of the spreadsheet is the second line of the CSV file and so on.

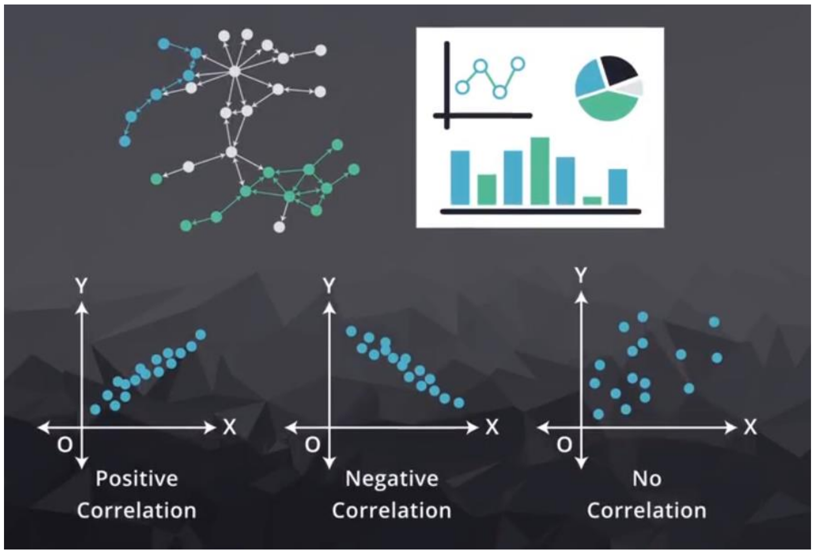
Within each row, you'll see the first cell followed by a comma, followed by the second cell, followed by a comma and so on. This makes CSVs very easy to process in programming languages like Python and packages such as Pandas have specific functions for loading data from CSVs.

Cleaning Data

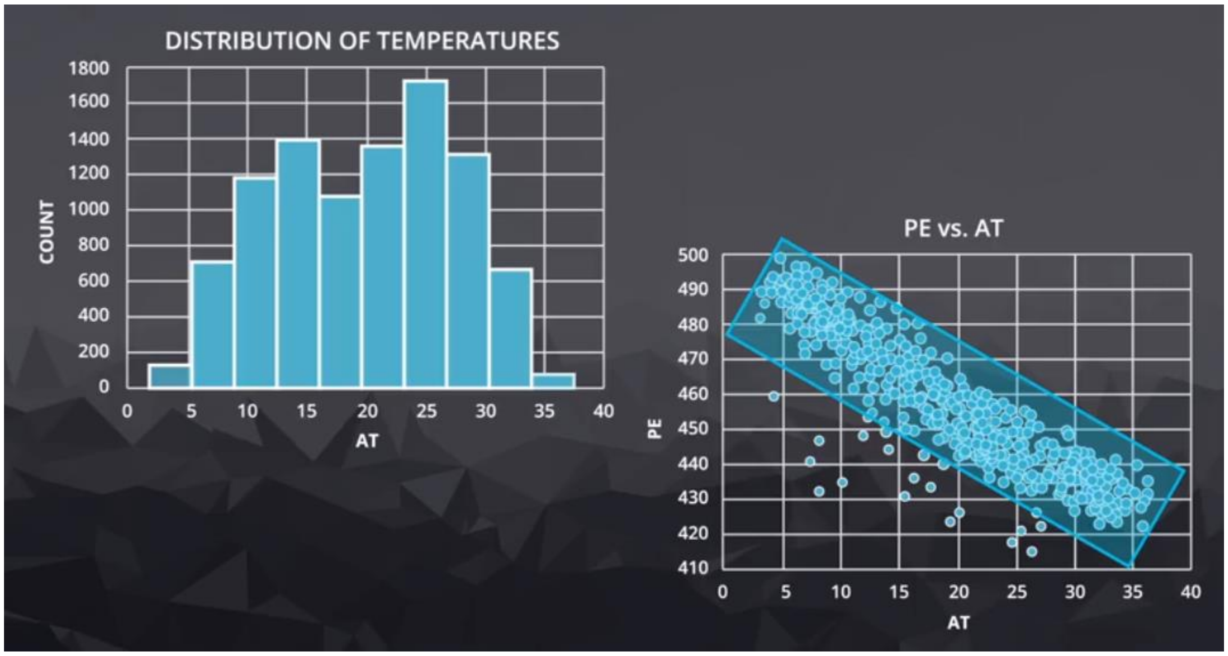
After assessing, you often need to fix problems in your data. Common problems include incorrect data types, missing data, duplicates, and structural problems, such as different column names in CSV files for the same data, or mismatched number of records. Fortunately, Pandas has useful functions that can help with these issues.

Exploring Data with Visuals

Now that you've cleaned your data, you can find patterns in your data by creating plots. Histograms and scatterplots can help you determine which variables will be used in analysis.



For example, you can plot a histogram to view the distribution of temperatures, or a scatterplot to view the relationship between temperature and energy output. This correlation looks strong, so we will probably be using this feature in our analysis.



Drawing Conclusions

Conclusions are typically drawn from machine learning algorithms or inferential statistics, both of which are offered in other Udacity courses.

Here, we will draw conclusions that can be made from descriptive statistics or visualizations.

Communicating Results

Visuals can be used to efficiently communicate conclusions drawn from your analysis.

Note the labeling, color, size, and data selection arguments for histograms, bar charts, and pie charts in the following examples.

You can use them to customize your visualizations.

Data Wrangling

Intro to Data Wrangling

In this course, we'll be learning all about data wrangling. Data wrangling is the process of gathering your data, assessing its quality and structure, and cleaning it before you do things like analysis, visualization, or building predictive models using machine learning.

Sometimes, it's as simple as downloading a file, spotting a few typos, and fixing those typos. But other times, your data really isn't clean. That is, you'll have missing records, duplicates, and inaccurate data.

Sometimes, the data itself is fine but structurally, it's difficult to work with. Taking care of all this is necessary or else, you risk making mistakes, missing insights and wasting time.

Wrangling is a core skill that you'll use to varying degrees on every data set that you work with because so much of the world's data isn't clean.

Fortunately, when done with code, wrangling is fast and almost magical. And by the end of this course, data wrangling will become instinctual. So, are you ready to wrangle? Let's get to it.

Course Outline

Data wrangling is a core skill that ***everyone*** who works with data should be familiar with since so much of the world's data isn't clean. Though this course is geared towards those who use Python to analyze data, the high-level concepts can be applied in all programming languages and software applications for data analysis.

Lesson 1: The Walkthrough

In the first lesson of this course, we'll walk through an example of data wrangling so you get a feel for the full **process**. We'll introduce **gathering** data, then download a file from the web and import it into a Jupyter Notebook. We'll then introduce **assessing** data and assess the dataset we just downloaded both visually and programmatically. We'll be looking for quality and structural issues. Finally, we'll introduce **cleaning** data and use code to clean a few of the issues we identified while assessing.

The goal of this walkthrough is awareness rather than mastery, so you'll be able to start wrangling your own data even after just this first lesson.

Lessons 2-4: Gathering, Assessing, and Cleaning Data (in Detail)

In the following lessons, you'll master **gathering**, **assessing**, and **cleaning** data. We'll cover the full data wrangling process with real datasets too, so think of this course as a series of wrangling journeys. You'll learn by doing and leave each lesson with tangible skills.

A Definition and An Analogy

A Definition

Wrangling is a weird word. Let’s check the definition. This is exactly what I did when I first heard the term and was perplexed just as you may be right now.

So wrangling means to *round up, herd, or take charge of livestock,* like horses or sheep. Let's focus in on the sheep example.

A shepherd's main goals are to get their sheep to their pastures to let them graze, guide them to market to shear them, and put them in the barn to sleep. Before any of that though, they must be rounded up in a nice and organized group. The consequences if they're not? These tasks take longer. If they're all scattered, some could also run off and get lost. A wolf could even sneak into the pack and feast on a few of them.

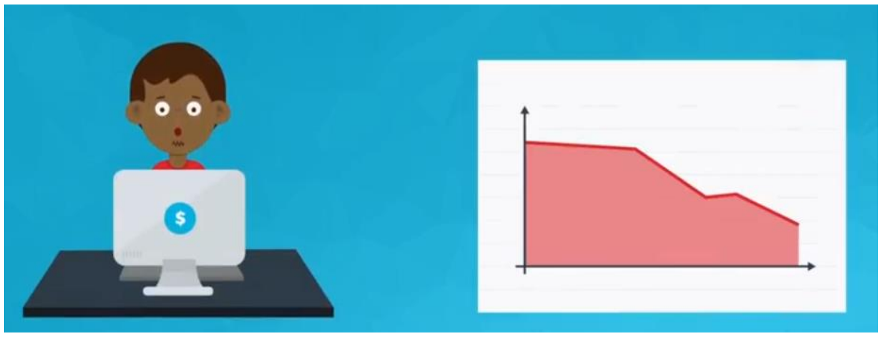
An Analogy

The same idea of organizing before acting is true for those who are shepherds of data. We need to wrangle our data for good outcomes, otherwise there could be consequences. If we analyze, visualize, or model our data before we wrangle it, our consequences could be making mistakes, missing out on cool insights, and wasting time. So best practices say wrangle. Always.

The development of Python and its libraries have made wrangling easier. In this course, you'll learn how to wrangle data like modern day data professionals.

Data Wrangling Example

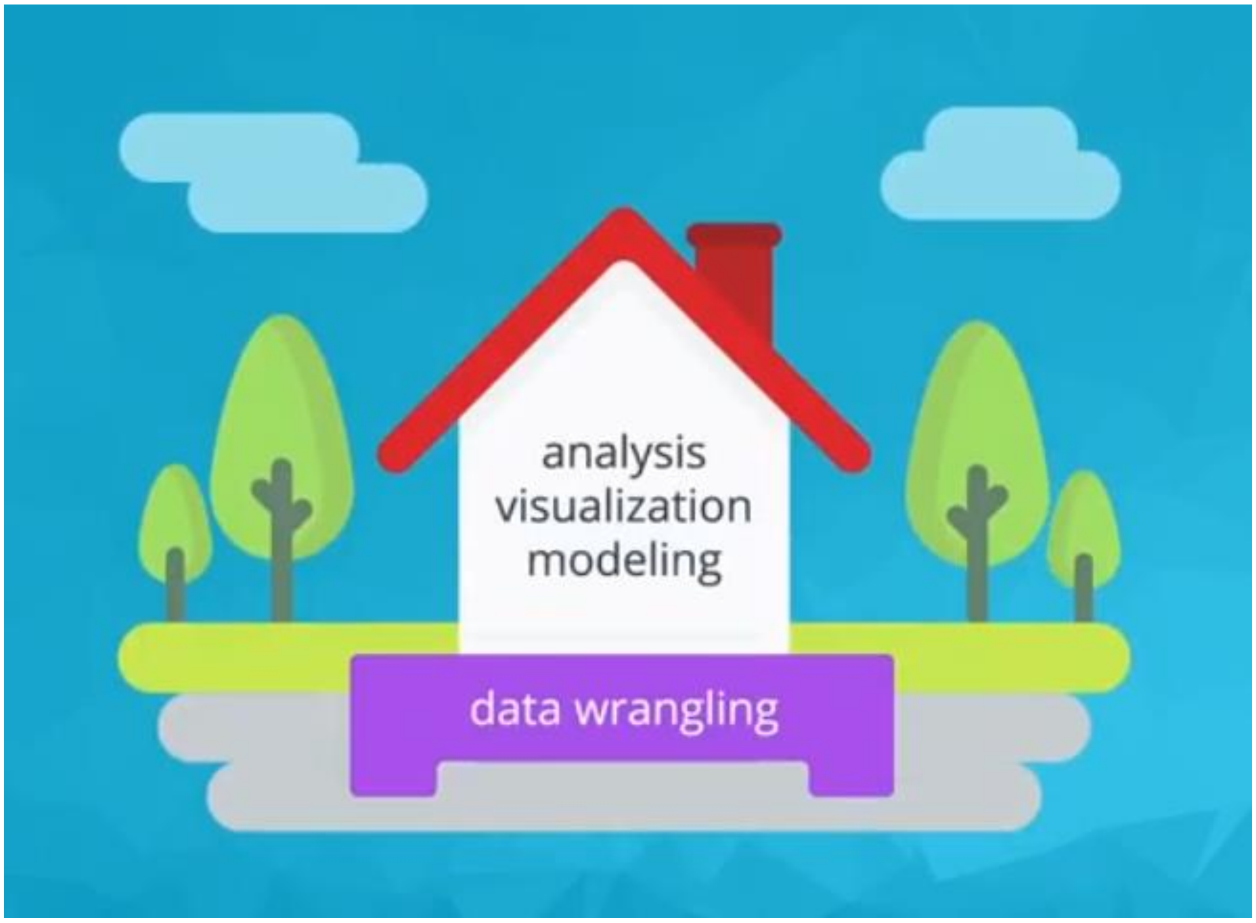
Data wrangling, or more appropriately, the consequences from a lack of data wrangling, can be serious business. For sure. Imagine you're a financial analyst creating models to make million-dollar trades for you. Your data better be clean or you could go broke.



Or imagine you're a scientist for a drug company that's about to start human trials for a life-saving new drug and you need to determine the right dosage for humans based on your lab and animal tests.

Your data needs to be clean or your drug might not work and you could seriously hurt people.

So while you're going to get some analysis and visualization in this course. It's really about getting your data ready so that anything you create is built on a solid foundation.



Gather

Gathering data is always the first step in data wrangling. The idea is before gathering, we have no data and after it, we do. This is sometimes called acquiring your data, or collecting it.

And depending on where you find your data and what format it's in, the steps of the gathering process can vary. If the data is in a file, gathering often means downloading the file and importing it into your programming environment like a Jupyter Notebook.

Other methods of gathering are things like collecting data from files and databases which is what you'll usually do in the workplace.

Or you can scrape data off a website or get it from an API, which stands for application programming interface. APIs let us programmatically access data from applications like Twitter and Facebook. There are even more on top of that. But let's stay focused and get to our walk through.

Gathering data is fun to me because it can vary so widely but we'll get to the fancier methods in Lesson 2 of the course, so stay tuned.

For now we're going to download a file from a website and import it into our programming environment.

Assess

Now we need to assess our data to determine what's clean and potentially what else to gather if we're missing some data. We're not exploring our dataset.

GATHER
ASSESS
CLEAN

We just want to make sure our data is in a form that makes our analysis easier later on. More on this in a bit. Okay. So what are we assessing?

What is dirty data? What is messy data? We're looking for two main things: Our data's quality and it's tidiness.

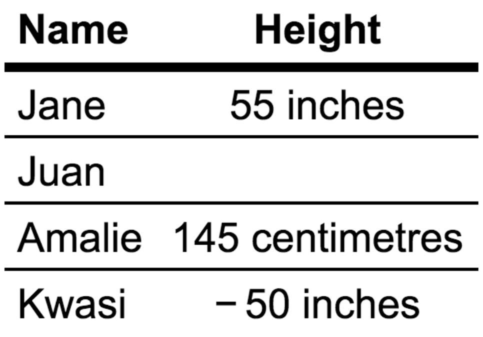
Low quality data is dirty. Untidy data is messy. We don't have to look for these in order, but you can. Let's formerly define quality and tidiness next.

Properties of Data

Quality

Low quality data is commonly referred to as *dirty data*. Dirty data has issues with its *content*.

Imagine you had a table with two columns: *Name* and *Height*, like below:



Name	Height
Jane	55 inches
Juan	
Amalie	145 centimetres
Kwasi	−50 inches

Common data quality issues include:

- missing data, like the missing height value for Juan.
- invalid data, like a cell having an impossible value, e.g., like negative height value for Kwasi. Having "inches" and "centimetres" in the height entries is technically invalid as well, since the datatype for height becomes a string when those are present. The datatype for height should be integer or float.
- inaccurate data, like Jane actually being 58 inches tall, not 55 inches tall.
- inconsistent data, like using different units for height (inches and centimetres).

We'll go over more tips and tricks to identify data quality issues and categorize them in the third lesson of the course.

Data quality is a perception or an assessment of data's fitness to serve its purpose in a given context. Unfortunately, that's a bit of an evasive definition but it gets to something important: there are no hard and fast rules for data quality. One dataset may be high enough quality for one application but not for another.

Tidiness

Untidy data is commonly referred to as *"messy" data*. Messy data has issues with its *structure*.

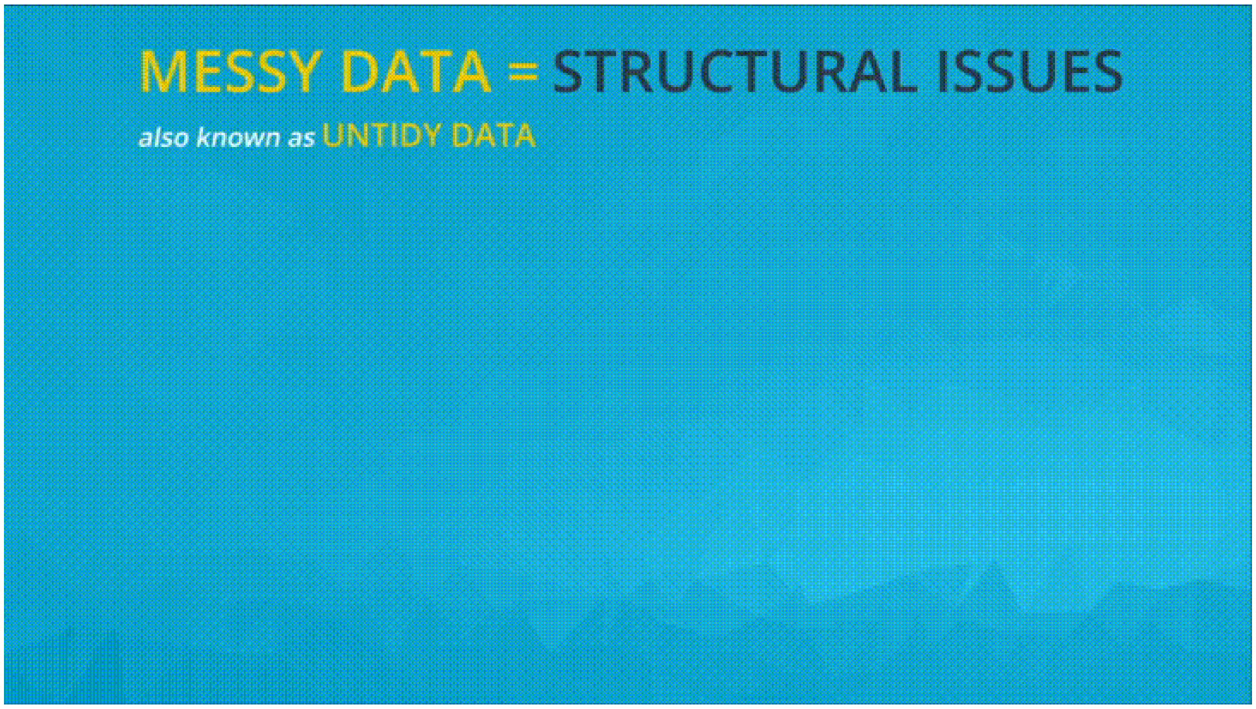
Tidy data is a relatively new concept coined by statistician, professor, and all-round data expert **Hadley Wickham(opens in a new tab)**. I'm going to take a quote from his excellent **paper(opens in a new tab)** on the subject:

It is often said that 80% of data analysis is spent on the cleaning and preparing data. And it's not just a first step, but it must be repeated many times over the course of analysis as new problems come to light or new data is collected. To get a handle on the problem, this paper focuses on a small, but important, aspect of data cleaning that I call data tidying: structuring datasets to facilitate analysis.

...

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.



Tidy data animation from Lesson 3: Assessing Data

Tidiness will be covered in detail in Lesson 3. If you'd like more information now, this **Tidy Data in Python(opens in a new tab)** article by Jean-Nicholas Hould is a good start.

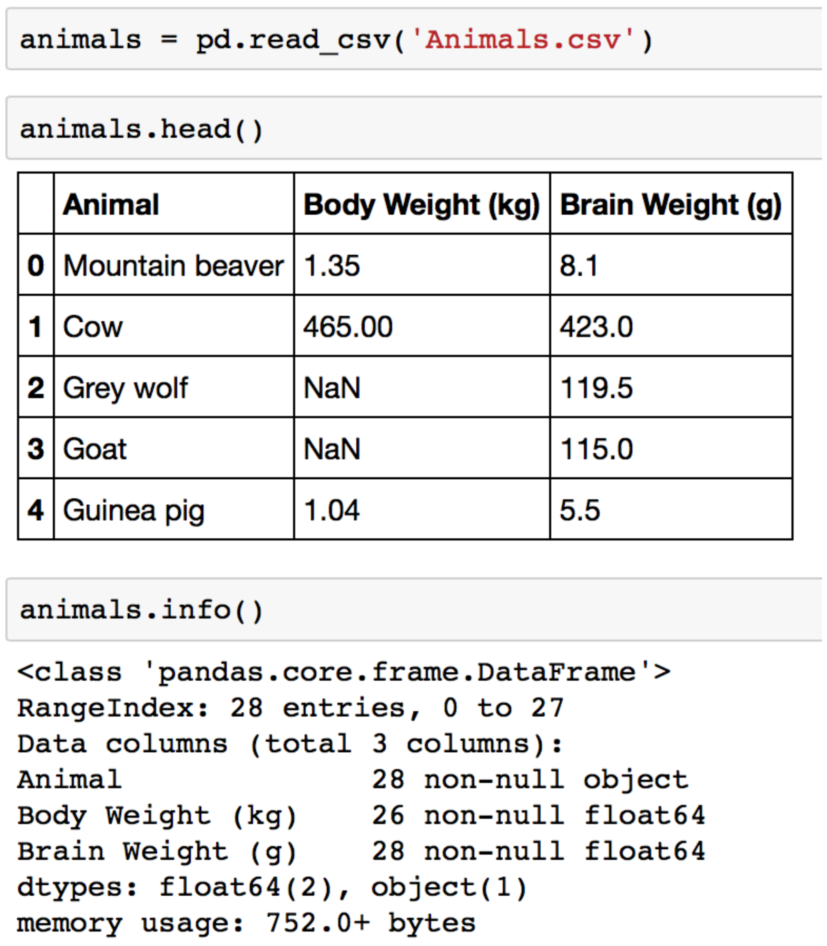
Types of Assessment

Visual Assessment

Visual assessment is simple. Open your data in your favorite software application (Google Sheets, Excel, a text editor, etc.) and scroll through it, looking for quality and tidiness issues.

Programmatic Assessment

Programmatic assessment tends to be more efficient than visual assessment. One simple example of a programmatic assessment is pandas' info method, which gives us the basic info of your DataFrame—like number of entries, number of columns, the types of each column, whether there are missing values, and more.



```
animals = pd.read_csv('Animals.csv')
```

```
animals.head()
```

	Animal	Body Weight (kg)	Brain Weight (g)
0	Mountain beaver	1.35	8.1
1	Cow	465.00	423.0
2	Grey wolf	NaN	119.5
3	Goat	NaN	115.0
4	Guinea pig	1.04	5.5

```
animals.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28 entries, 0 to 27
Data columns (total 3 columns):
Animal              28 non-null object
Body Weight (kg)    26 non-null float64
Brain Weight (g)    28 non-null float64
dtypes: float64(2), object(1)
memory usage: 752.0+ bytes
```

pandas' info method in action on a DataFrame called animals

Another example is using pandas' plotting capabilities through the plot method, though simple visualizations are more common in exploratory data analysis (we'll discuss this later in this lesson) rather than data wrangling.

These types of assessments are handy for gauging your data's structure and also for quickly spotting things that we'll need to clean.

Explore Common Programmatic Assessments

Now it's time to explore programmatic assessments for yourself! Again, this is where we use code to help detect problems in our data that aren't as easily detectable with the human eye.

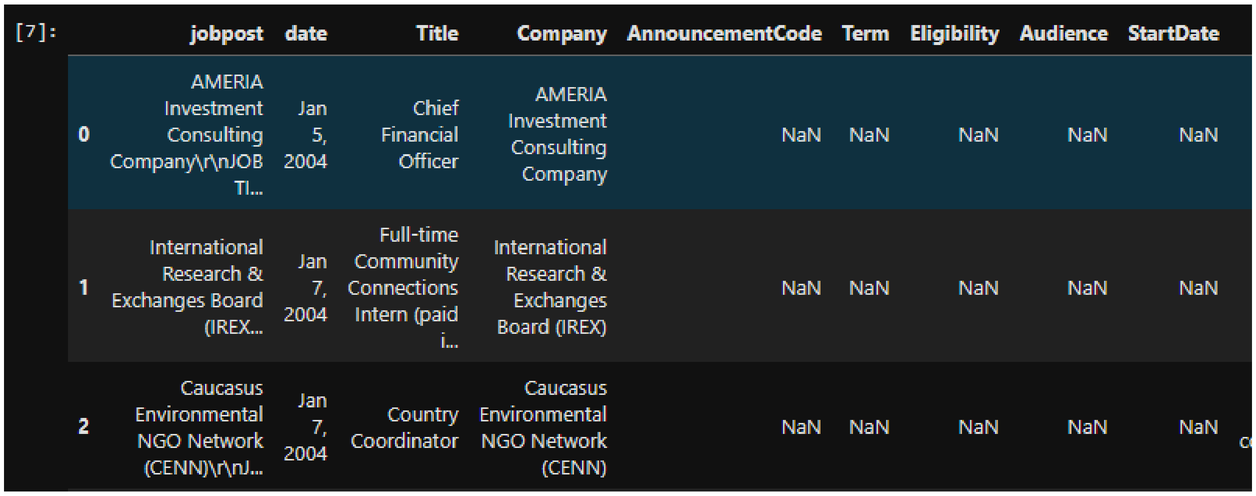
The data wrangling template is displayed in the Jupyter Notebook below with empty cells for four common programmatic assessment methods in pandas (documention pages linked below):

- **head(opens in a new tab)**
- **tail(opens in a new tab)**
- **info(opens in a new tab)**
- **value_counts(opens in a new tab)**

Execute these assessment as per the instructions in those cells. In the following quizzes, you'll be asked to replicate these statements.

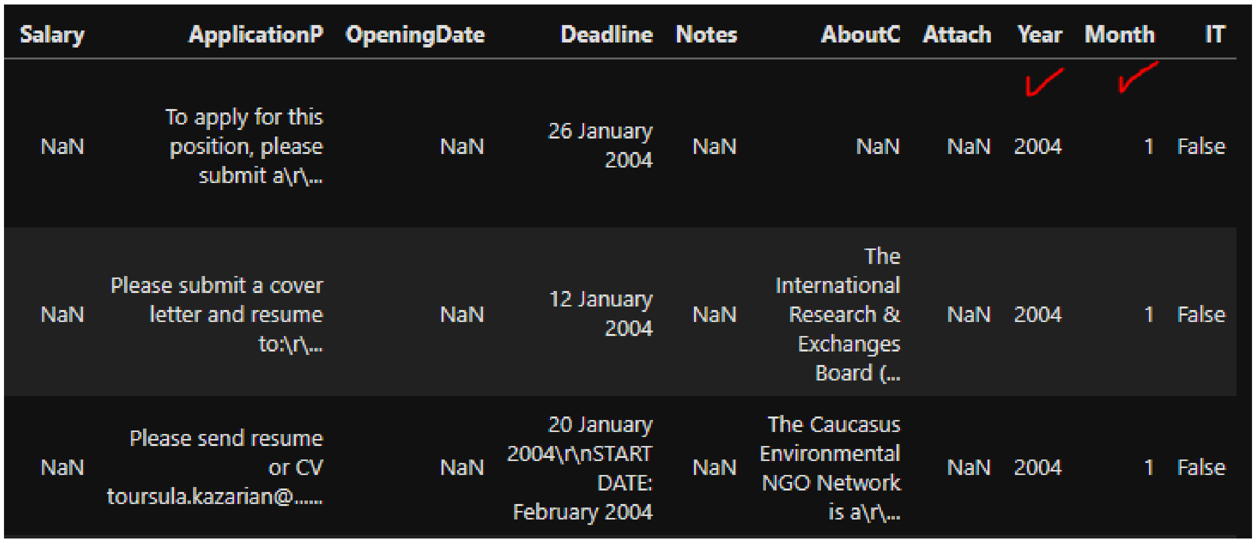
For these quizzes and all quizzes going forward, don't go into them just trying to get them right. Exploring is a key part of learning. Get the code right, then I encourage you to explore the documentation, try various parameters, try new things and see where things break. Error messages are your friend, because you can learn from them.

Is this data tidy?



[7]:

	jobpost	date	Title	Company	AnnouncementCode	Term	Eligibility	Audience	StartDate
0	AMERIA Investment Consulting Company\r\nJOB TI...	Jan 5, 2004	Chief Financial Officer	AMERIA Investment Consulting Company	NaN	NaN	NaN	NaN	NaN
1	International Research & Exchanges Board (IREX...	Jan 7, 2004	Full-time Community Connections Intern (paid i...	International Research & Exchanges Board (IREX)	NaN	NaN	NaN	NaN	NaN
2	Caucasus Environmental NGO Network (CENN)\r\nJ...	Jan 7, 2004	Country Coordinator	Caucasus Environmental NGO Network (CENN)	NaN	NaN	NaN	NaN	NaN



Salary	ApplicationP	OpeningDate	Deadline	Notes	AboutC	Attach	Year	Month	IT
							✓	✓	
NaN	To apply for this position, please submit a\r\...	NaN	26 January 2004	NaN	NaN	NaN	2004	1	False
NaN	Please submit a cover letter and resume to:\r\...	NaN	12 January 2004	NaN	The International Research & Exchanges Board (...	NaN	2004	1	False
NaN	Please send resume or CV toursula.kazarian@......	NaN	20 January 2004\r\nSTART DATE: February 2004	NaN	The Caucasus Environmental NGO Network is a\r\...	NaN	2004	1	False

So no. This data set isn't tidy.

For tidy data, we need each variable to be a column, each observation to be a row, and each type of observational unit to be a table.

So first, we have duplicate representation of data.

One in this column here, the **date** column, January 5th 2004 for example, month, day, year that also again at the end of the data set here with **Year** and **Month** having their own columns. Here is the duplicate representation of date data.

So that means we have two representations of year and month data. For tidy, we need one day, one month and one year. So if we have to make updates to the data we only have to do it in one place. I would also argue that there are two types of observational units in this dataset; job posting data and then company data.

To make this tidy, we would have to separate these into two tables: One, the job posting data having everything in this dataset except for this AboutC column.

And then a second, the Company table with only this company column and this AboutC column or About Company column.

For this walk through though we are going to skip documenting this lack of tidiness and not fix it in the clean step.

I want to keep this lesson short. For one it also because non-tidy data is okay sometimes.

Clean

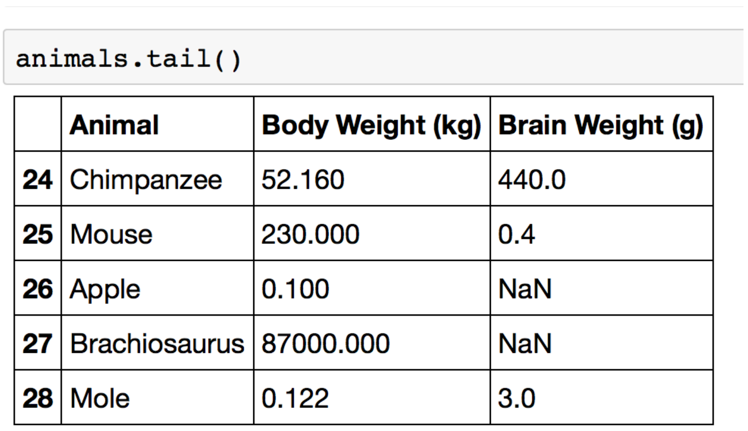
Improving Quality and Tidiness

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

Improving Quality

Improving quality doesn’t mean changing the data to make it say something different—that's data fraud.

Consider the *animals* DataFrame, which has headers for name, body weight (in kilograms), and brain weight (in grams). The last five rows of this DataFrame are displayed below:

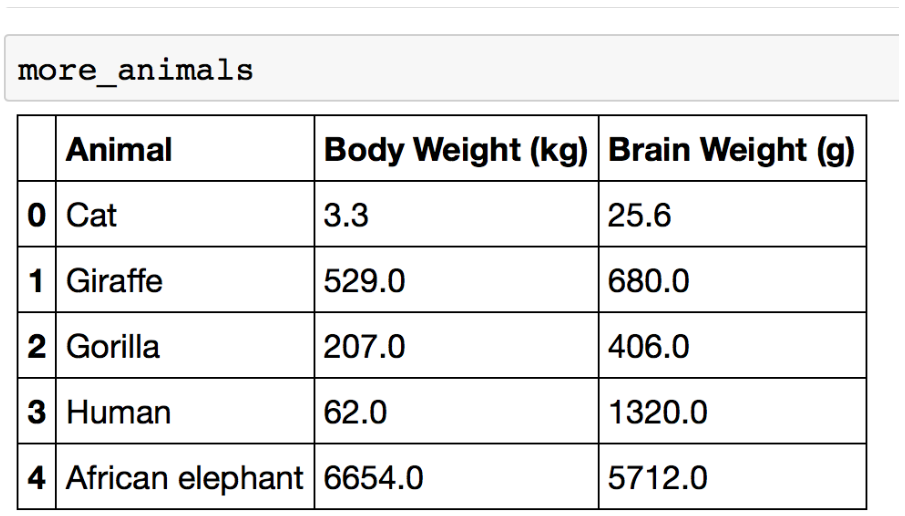


```
animals.tail()
```

	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



```
more_animals
```

	Animal	Body Weight (kg)	Brain Weight (g)
0	Cat	3.3	25.6
1	Giraffe	529.0	680.0
2	Gorilla	207.0	406.0
3	Human	62.0	1320.0
4	African elephant	6654.0	5712.0

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. There are special functions in pandas that help us do that. We'll dive deeper into those in Lesson four of this course.

The programmatic data cleaning process:

1. Define
2. Code
3. 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.

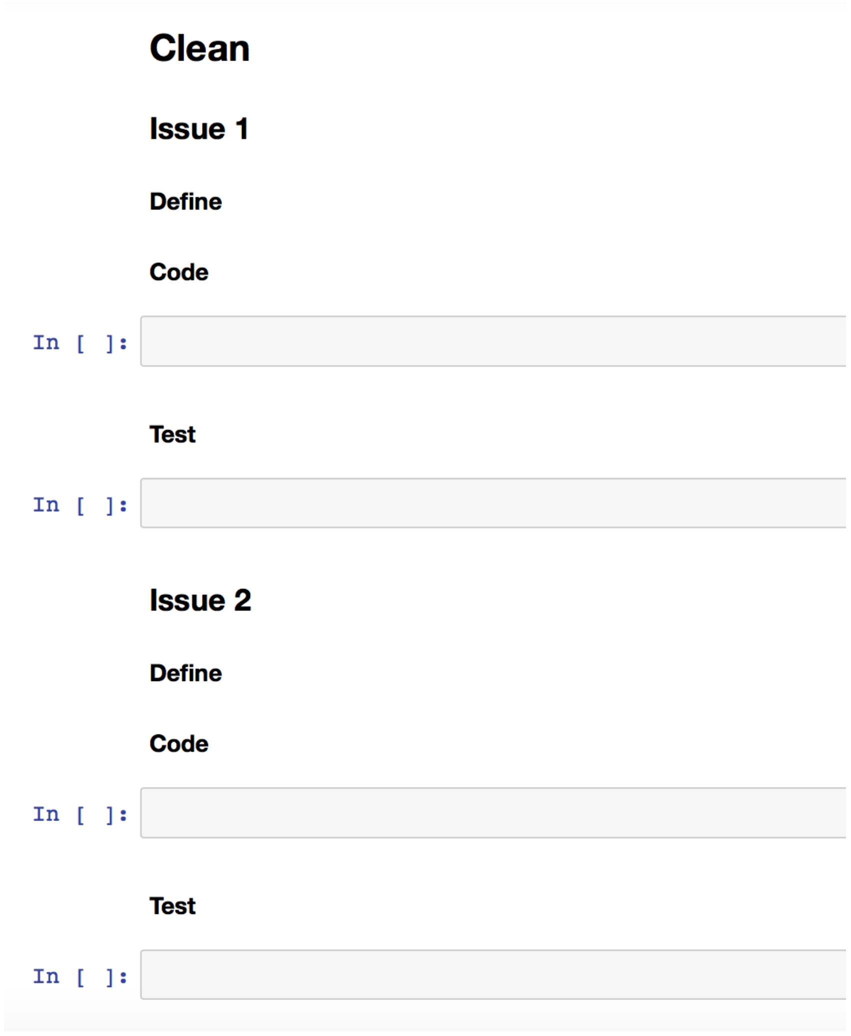
Coding means translating these definitions to code and executing that code.

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

Defining, then Coding and Testing Immediately

For pedagogical purposes in this lesson, we will be performing the define, code, and test steps of cleaning data programmatically in order. In other words, we write *all* of the definitions, then convert *all* of the definitions to code, then *test* all of the cleaning operations.

In reality, it is often more practical to define a cleaning operation, then immediately code and test it. The data wrangling template still applies here, except you'll have multiple **Define**, **Code**, and **Test** subheadings, with third level headers (###) denoting each issue, as displayed below.



Clean

Issue 1

Define

Code

```
In [ ]:
```

Test

```
In [ ]:
```

Issue 2

Define

Code

```
In [ ]:
```

Test

```
In [ ]:
```

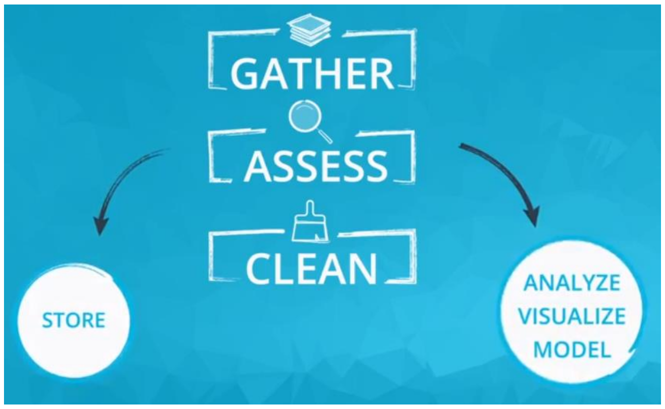
Reassess and Iterate

We've gathered, assessed and just cleaned our data. Are we done? No. After cleaning, we always reassess and then iterate on any of the steps if we need to. If we're happy with the quality and tidiness of our data, we can end our wrangling process and move on to storing our clean data, or analyzing, visualizing, or modeling it.

Sometimes we realize we need to gather more data. Sometimes we miss assessments.

It's hard to catch everything on the first go, and it's also very common to find new issues as you're fixing the ones you've already identified.

Sometimes our cleaning operations don't work as we intended. Once we go through each step once, we can revisit any step in the process any time. Even after we've finished wrangling and moved on to analysis, visualization, or modeling. So don't worry if wrangling seems daunting. Feel free to start small and cycle through the process a few times.



Storing Data (Optional)

After reassessing your data and revisiting any steps of the data wrangling process deemed necessary, storing your cleaned data can be the next logical step. Storing data is important if you need to use your cleaned data in the future.

Storing data isn't always necessary, though. Sometimes the Jupyter Notebook that you gathered, assessed, cleaned, analyzed, and visualized your data in, plus the original data files is good enough. Sometimes the analysis and visualization are the final products and you won't be using the cleaned data any further. If you ever want to reproduce the analysis, the Jupyter Notebook suffices.

Storing your data, in files and databases for example, will be covered in detail in a later lesson in the course.

Wrangling vs. EDA vs. ETL

Wrangling vs. EDA

At another point in the Data Analyst Nanodegree or if you're taking this course individually, another point in your data journey, you will encounter a topic called exploratory data analysis (EDA). If you've encountered EDA already you might be thinking: data wrangling seems very similar to exploratory data analysis. And that's because they *are* very similar and often get confused. Here is one definition of 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.*

So where does data wrangling end and EDA start?

Data wrangling is about gathering the right pieces of data, assessing your data's quality and structure, then modifying your data to make it clean. But the assessments you make and convert to cleaning operations won't make your analysis, viz, or model better, though. The goal is to just make them possible, i.e., functional.

EDA is about exploring your data to later augment it to maximize the potential of our analyses, visualizations, and models. When exploring, simple visualizations are often used to summarize your data's main characteristics. From there you can do things like remove outliers and create new and more descriptive features from existing data, also known as **feature engineering(opens in a new tab)**. Or detect and remove outliers so your model's fit is better. In practice, wrangling and EDA can and often do occur together, but we're going to separate them for teaching purposes.

ETL

You also may have heard of the **extract-transform-load process(opens in a new tab)** 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

This article (**Data Wrangling Versus ETL: What’s the Difference?(opens in a new tab)**) by Wei Zhang explains these three differences well.

Data Wrangling Summary

Gather

- Depending on the source of your data, and what format it's in, the steps in gathering data vary.
- High-level gathering process: obtaining data (downloading a file from the internet, scraping a web page, querying an API, etc.) and importing that data into your programming environment (e.g., Jupyter Notebook).

Assess

- Assess data for:
  - **Quality**: issues with content. Low quality data is also known as dirty data.
  - **Tidiness**: issues with structure that prevent easy analysis. Untidy data is also known as messy data. Tidy data requirements:
    i. Each variable forms a column.
    ii. Each observation forms a row.
    iii. Each type of observational unit forms a table.
- **Types of assessment:**
  - **Visual assessment:** scrolling through the data in your preferred software application (Google Sheets, Excel, a text editor, etc.).
  - **Programmatic assessment:** using code to view specific portions and summaries of the data (pandas' head, tail, and info methods, for example).

Clean

- **Types of cleaning:**
    - Manual (not recommended unless the issues are single occurrences)
    - Programmatic
- **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!**

Reassess and Iterate

- After cleaning, always reassess and iterate on any of the data wrangling steps if necessary.

Store (Optional)

- Store data, in a file or database for example, if you need to use it in the future.

Gathering Data

You've reached lesson two of our data wrangling course. In lesson one, we walk through the whole wrangling process from gathering, to assessing, to cleaning data, and iterating if necessary. In the next three lessons, you're going to dive deeper into each step of the process.

First, you're going to master gathering data. Depending on your project, gathering data varies a lot. Data you need may be spread across dozens of sources and in different file formats, which might be the most challenging part of working with data. Not analyzing, not visualizing, not even building fancy machine-learning models, simply gathering.

It's also glossed over in most data wrangling courses where complete datasets are just provided for you. Not here. In this lesson, you'll gather real data from several different sources and file formats while learning about each and how to handle them using Python.

You'll combine them all to create a master dataset which we'll then use to answer interesting questions and produce stunning visualizations.

You're going to leave this lesson with the gathering skills required for the vast majority of your wrangling projects in the future.

So let's do this. This is going to be fun,

challenging, and is going to require a bit of creativity.

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:

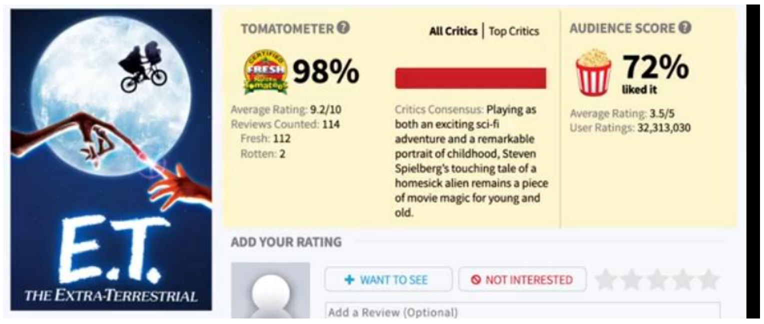
- They're text files and therefore human readable.
- Lightweight.
- Simple to understand.
- Software that can read/write text files is ubiquitous, like text editors.
- Great for small datasets.

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.

Web Scraping

All right. So we want to grab this audience score here to add to our dataset and number of audience reviews to match our number of critic reviews.



98%

Average Rating: 9.2/10
Reviews Counted: 114
Fresh: 112
Rotten: 2

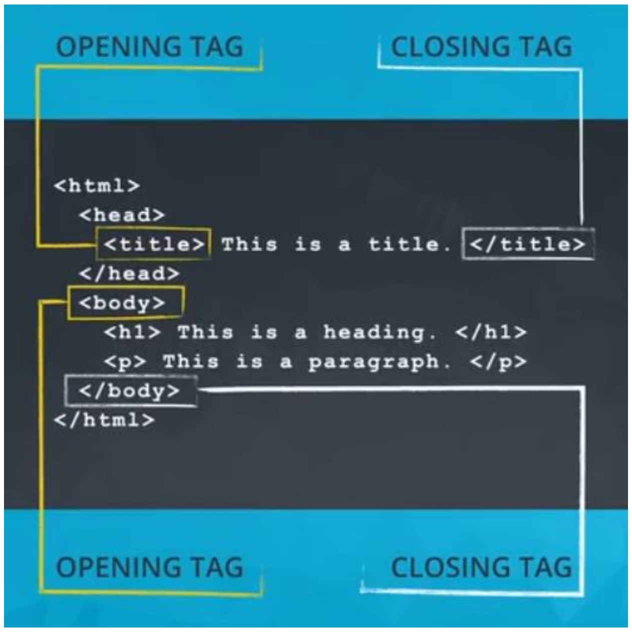
Critics Consensus: Playing as both an exciting sci-fi adventure and a remarkable portrait of childhood, Steven Spielberg's touching tale of a homesick alien remains a piece of movie magic for young and old.

72%
liked it

Average Rating: 3.5/5
User Ratings: 32,313,030

Unfortunately, Rotten Tomatoes hasn't made this easily accessible for us.

One way we can get these pieces of data though, web scraping. Web scraping is a fancy way of extracting data from web sites using code. It's one of the first magical things that drew me to programming. Behind the scenes though, web scraping is quite simple. The data that lives on web pages is called HTML, HyperText Markup Language. It's made up of these things called tags which give the web page structure.



```
<html>
  <head>
    <title> This is a title. </title>
  </head>
  <body>
    <h1> This is a heading. </h1>
    <p> This is a paragraph. </p>
  </body>
</html>
```

Because HTML code is just text, these tags and the content within them can be accessed using parsers, and in Python, there is an awesome one called Beautiful Soup.

We can download HTML and access it offline, or we can do it in real time over the Internet.

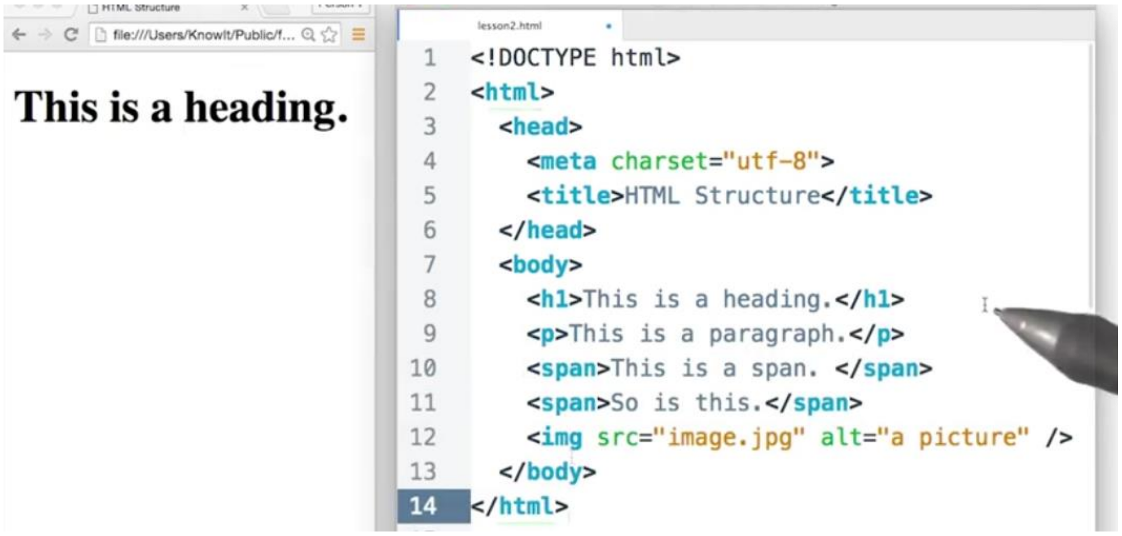
For this lesson, we're going to download the HTML files and parse them using Beautiful Soup. Before we do though, let's explore the structure of HTML files.

The two main ways to work with HTML files are:

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

HTML File Structure

You are looking at an HTML document, and this is what this document looks like inside the browser.



```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>HTML Structure</title>
  </head>
  <body>
    <h1>This is a heading.</h1>
    <p>This is a paragraph.</p>
    <span>This is a span. </span>
    <span>So is this.</span>
    <img src="image.jpg" alt="a picture" />
  </body>
</html>
```

The similarities should be obvious. You can see this is a heading here inside the HTML, and you can see the same, this is a heading here inside the browser.

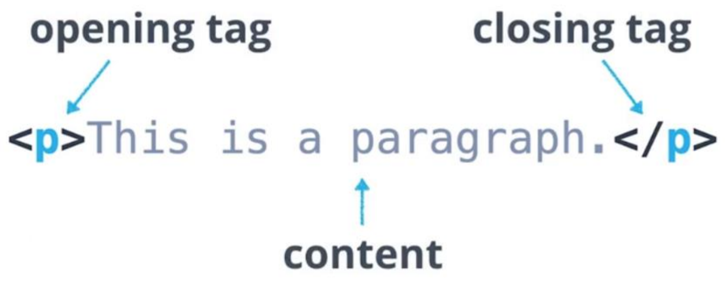
There's a lot here. But I don't want you to worry about everything. Right now, just pay attention to what you see right here, which is coming inside these two tags.

That's a body. This thing right here with, this is a heading, and is an HTML element.

And this is a special kind of element called h1 element. H stands for heading, and one means it's the biggest. That means that there are h2, h3, h4 tags and so on, and they're all smaller than each one.

Specifically, this part is called a tag, the part with an angle bracket h1, and then another angle bracket. There are many tags in HTML. Here's a p tag, here are some span tags, and this is an image tag. Each one of these represents a different kind of element.

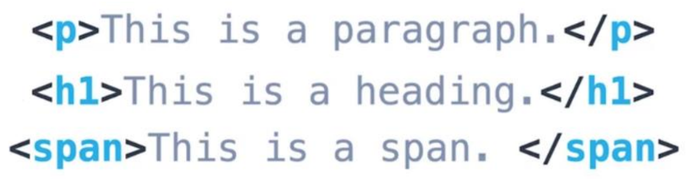
This p, inside this tag, stands for paragraph. This whole thing is called a paragraph element. Notice that this whole element has two tags, here and here, and these tags are slightly different.



```
<p>This is a paragraph.</p>
```

There's an opening tag, and there is a closing tag, and notice the closing tag has the slash in it. Between the opening and the closing tag comes the content. This content is generally what people will see inside the browser.

So here are a few different elements. Notice that the text in both tags is always the same.



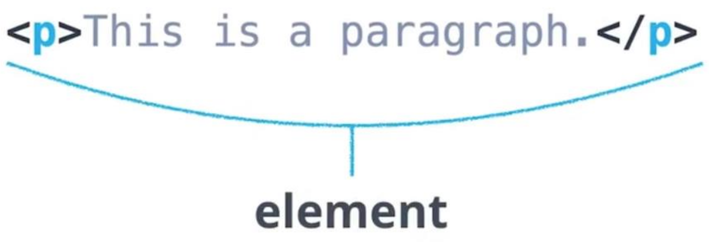
```
<p>This is a paragraph.</p>
<h1>This is a heading.</h1>
<span>This is a span. </span>
```

And notice that they all start with a less than sign, they all have some kind of text in here indicating what type of element this is, and then every tag has a greater than sign.

And just to point it out again, closing tags have a slash inside them, but otherwise they look the same.

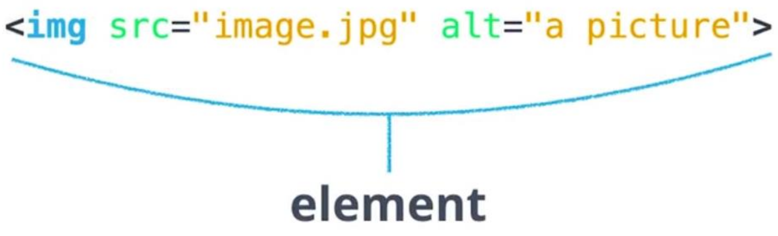
And the slash always comes right here after the less than sign.

And then together, everything here represents an element.



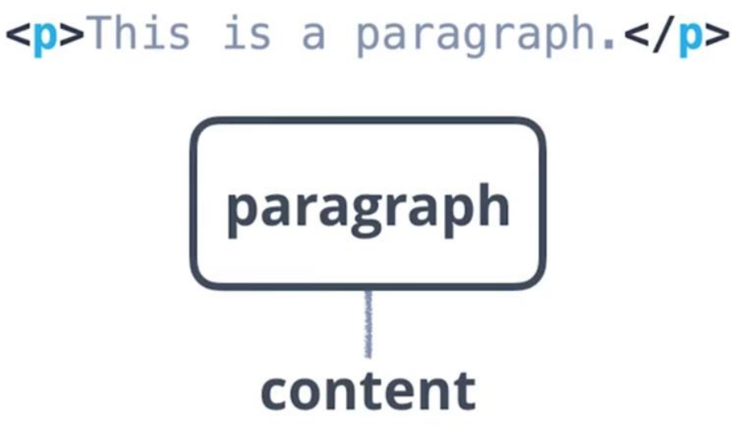
```html
<p>This is a paragraph.</p>
```

There are some elements, like images, that are slightly different. Notice that images do not have a closing tag. And that's okay. It's a special one that you'll learn more about later.



```html
<img src="image.jpg" alt="a picture">
```

This is a conceptual way of representing the same element. In this case, the content belongs to the paragraph. The content in a way is a child of the paragraph.

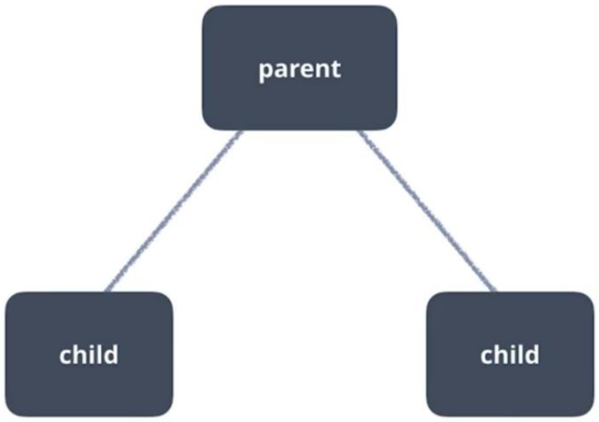


```html
<p>This is a paragraph.</p>
```

This is going to be useful later. Don't worry about it now, but I just wanted to show you because you'll be seeing more trees like this.

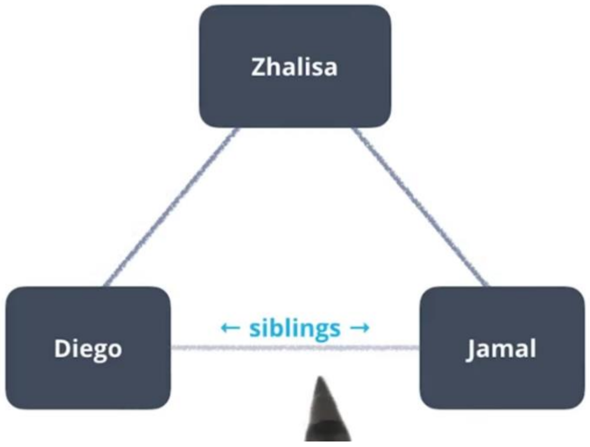
Trees

This is a family tree. You've probably seen something like this before. There's a parent. And two children.



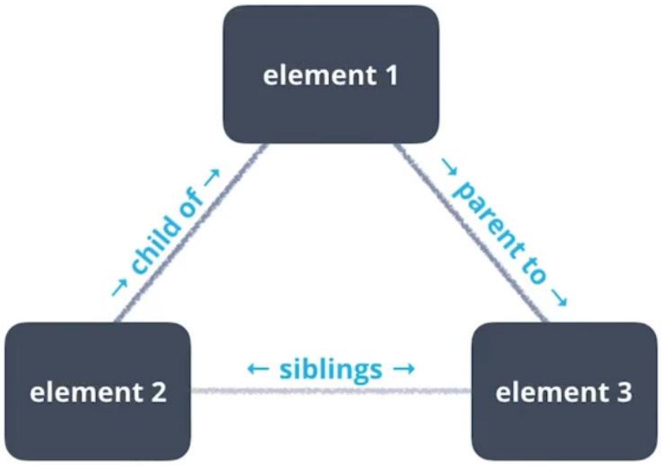
And yeah. I know it takes two parents to make a kid but, just ignore that for now.

Pretend it only takes one parent. For this family, you would say that Diego and Jamal are children of Zhalisa, and you would say that Zhalisa is the parent to Jamal and Diego.

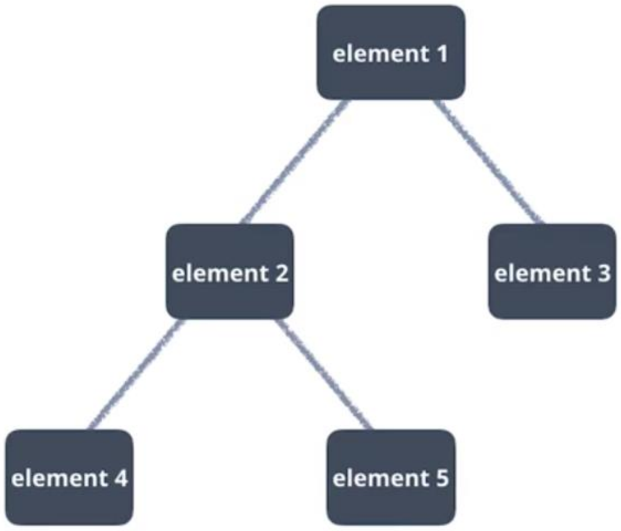


You could also say that Diego and Jamal are siblings. We use the same kind of family terms like child, siblings and parent to describe relationships like these inside a data tree.

Now a tree is just a type of data in computer science. They're used to represent hierarchical data.



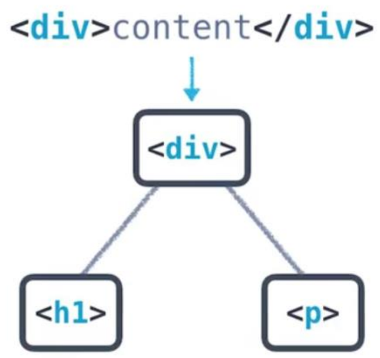
That is to say, bits of information that belong to each other. In this case, element 3 and element 2 both belong to element 1. And just like a family, the children can be parents too and that's it, that's a tree. I brought this tree up because your HTML turns into a tree a lot like this one. Elements will belong to each other or be descended from one another.



There are children and parents and children have children. Now I know that can sound a little confusing so here, let me show you what I mean.

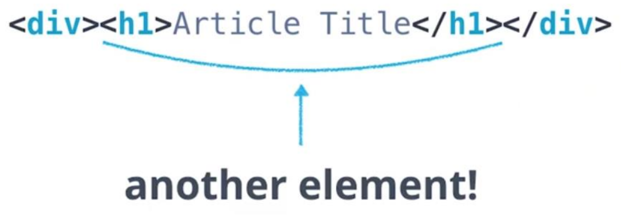
HTML and Trees

So, how do you get from here to here? The answer is content.



```
<div>content</div>
```

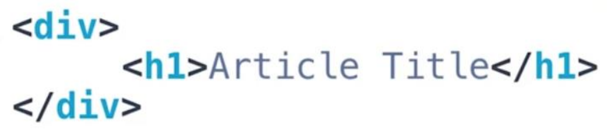
As it turns out, elements can go inside other elements. Content isn't just text, it is also other elements.



```
<div><h1>Article Title</h1></div>
```

So this is a new tag, it's a div which means division. A division element, or we just call them divs, are used to group chunks of content together and you'll be learning a lot more about divs later on. But either way, inside this div comes an h1 element.

This h1 is a child of the div. Here, let me reorganize to make it a little bit clearer.



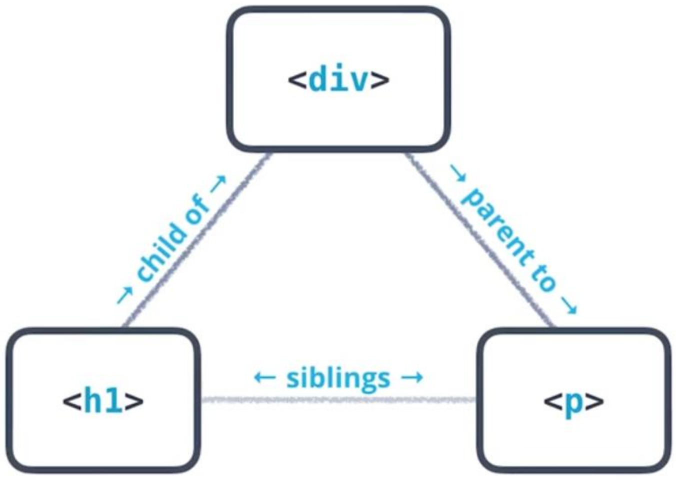
```
<div>
    <h1>Article Title</h1>
</div>
```

I've moved the h1 to its own row and I've tagged it in. This makes it really clear that the h1 belongs to its parent div. You can see that the h1 comes between the opening and the closing div tags. You'll need to use this same format when you're writing HTML. Let me go ahead and add another element to the mix.

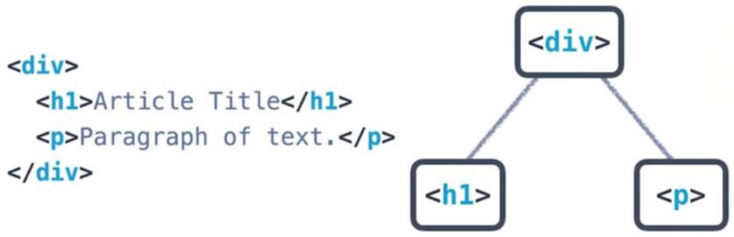
Okay, pause for one second.

```
<div>
     <h1>Article Title</h1>
     <p>Paragraph of text.</p>
</div>
```

What do you think the tree from this HTML looks like? What's the parent? What are the children? What are the siblings? Okay, I'm going to pause for a second and let you think. Here it is.



You can see that the div is the parent of the h1 and the p, and that h1 and p are both siblings. So just to be clear, this is the code that you write and this is a way of thinking about it.



```
<div>
  <h1>Article Title</h1>
  <p>Paragraph of text.</p>
</div>
```

As I keep saying, it's going to be really handy later on knowing that your code. Turns into this. Of course, as you're writing your code, the trees that you create with HTML can grow as wide or as deep as you need them to be. It doesn't matter.

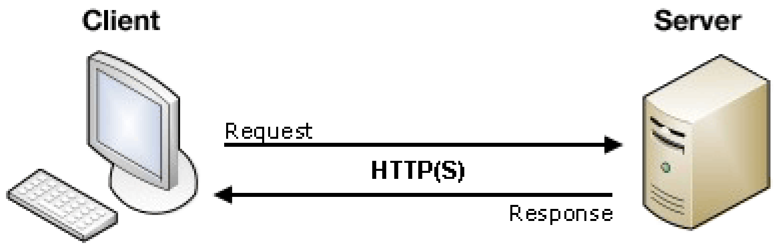
Downloading Files from the Internet

HTTP (Hypertext Transfer Protocol)

HTTP, the Hypertext Transfer Protocol, is the language that web browsers (like Chrome or Safari) and web servers (basically computers where the contents of a website are stored) speak to each other. Every time you open a web page, or download a file, or watch a video, it's HTTP that makes it possible.

HTTP is a request/response protocol:

- Your computer, a.k.a. the client, sends a request to a server for some file. For this lesson: *"Get me the file 1-the-wizard-of-oz-1939-film.txt*"*, for example. GET is the name of the HTTP request method (of which there are multiple) used for retrieving data.
- The web server sends back a response. If the request is valid: *"Here is the file you asked for:"*, then followed by the contents of the ***1-the-wizard-of-oz-1939-film.txt*** file itself.



Source: **MDN web docs(opens in a new tab)**

If you'd like to learn more, or are feeling like there are knowledge gaps you'd like to fill in, I encourage you to check out the following videos in our free **Web Development course(opens in a new tab)** in Lesson 1 ("How the Web Works").

Text File Structure

Encodings and Character Sets Articles

The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!)(opens in a new tab) by Joel Spolsky

An excerpt:

The Single Most Important Fact About Encodings

If you completely forget everything I just explained, please remember one extremely important fact. It does not make sense to have a string without knowing what encoding it uses. You can no longer stick your head in the sand and pretend that "plain" text is ASCII.

There Ain't No Such Thing As Plain Text

If you have a string, in memory, in a file, or in an email message, you have to know what encoding it is in or you cannot interpret it or display it to users correctly.

Almost every stupid "my website looks like gibberish" or "she can't read my emails when I use accents" problem comes down to one naive programmer who didn't understand the simple fact that if you don't tell me whether a particular string is encoded using UTF-8 or ASCII or ISO 8859-1 (Latin 1) or Windows 1252 (Western European), you simply cannot display it correctly or even figure out where it ends. There are over a hundred encodings and above code point 127, all bets are off."

What Every Programmer Absolutely, Positively Needs To Know About Encodings And Character Sets To Work With Text(opens in a new tab)

An article by Joel Spolsky entitled The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!) is a nice introduction to the topic and I greatly enjoy reading it every once in a while. I hesitate to refer people to it who have trouble understanding encoding problems though since, while entertaining, it is pretty light on actual technical details. I hope this article can shed some more light on what exactly an encoding is and just why all your text screws up when you least need it.

Any character can be encoded in many different bit sequences and any particular bit sequence can represent many different characters, depending on which encoding is used to read or write them. The reason is simply because different encodings use different numbers of bits per characters and different values to represent different characters.”

Unicode and Python

In Python 3, there is:

- one text type: **str**, which holds Unicode data and
- two byte types: **bytes** and **bytearray**

The Stack Overflow answers **here(opens in a new tab)** explain the different use cases well.

More Information

- If you’re still confused about the difference between character sets and encoding, check out these articles:
  - **The difference between UTF-8 and Unicode?(opens in a new tab)**
  - **More About Unicode in Python 2 and 3**

Storing Data

So, you've got all the pieces of data you need for our two visualizations. In its own pandas dataframe. What do you do with the data in that dataframe now that you're done? You want to save or store it somehow.

Two popular options. Saving to a database and saving to a file. For tabular data like we have here, both files, like CSV files, and databases are used a ton.

The right solution depends on your dataset and the infrastructure you use in your daily work.

Saving to a CSV file is easy and is probably the best solution for a simple dataset like this one. You'll learn how to do this next.

But you're also going to learn how to load the data into a database. They're fast, they scale well as your data grows, and you can ask them questions using languages like SQL. They're very powerful, and database skills are hugely in demand in today's workplace.

Relational Database Structure

A database is an organized collection of data that is structured to facilitate the storage, retrieval, modification, and deletion of data. There are two main types of databases: relational databases and non-relational databases, with relational being the most popular. SQL, or Structured Query Language, is the standard language for communicating with relational databases.

Why do Data Analysts love SQL?

A lot of the world's data lives in databases, and most of the world's databases are accessed using structured query language, which is typically abbreviated as SQL, or SQL. It's been around since the 1970s and is the most common method of accessing data and databases today. SQL has a variety of functions that allow its users to read, manipulate, and change data.

It's also popular for data analysis for a few reasons.

First, it's semantically easy to understand and learn. It can be used to access large amounts of data directly where it's stored. You don't have to copy data into other applications to view it, and you don't need to worry about your spreadsheet program crashing because of data overload.

It's easy to audit and replicate. In a spreadsheet tool like Excel, you have to click in to each cell to know how they're calculated. With SQL, you can just read a query from top to bottom, nothing's hidden.

Finally, SQL can run queries on multiple tables at once, across large datasets. It's great for performing the types of aggregations you might normally do in an Excel pivot table, sums, counts, minimums and maximums, etcetera, but Excel maxes out at just over 1 million rows. With SQL, you can query across billions of rows at a time.

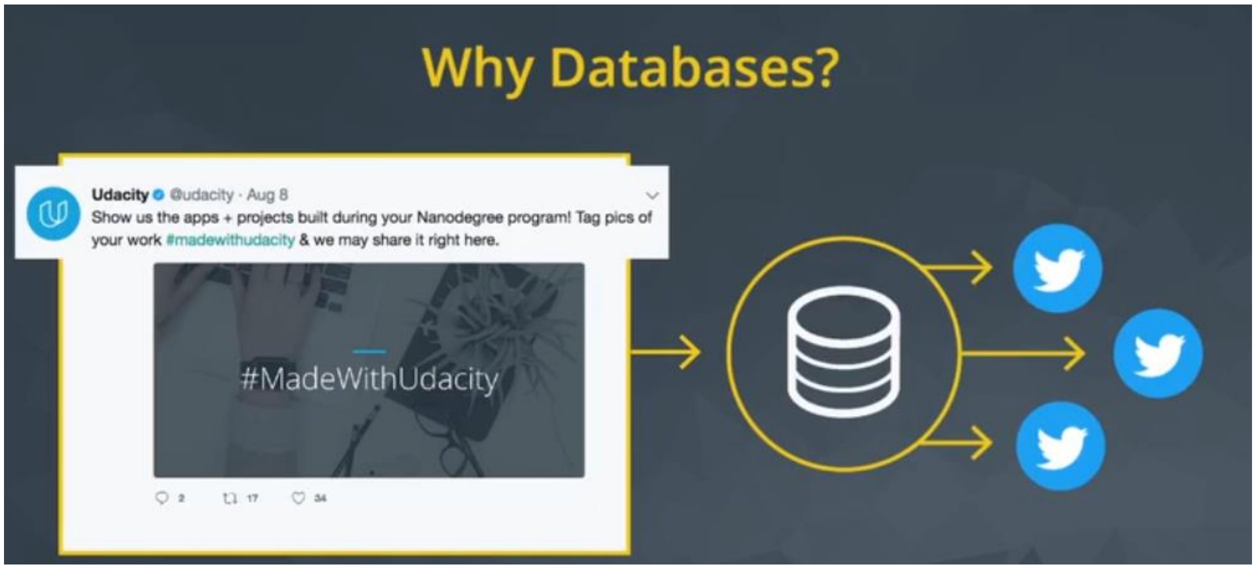
Maybe most importantly, SQL's incredibly flexible, especially when compared to out of the box dashboard products like Google Analytics.

In Google Analytics, you might be able to answer questions like, which pages on my site receive the most traffic? And where does it come from? But you can use SQL's filtering capabilities to answer much deeper, more complex questions like, how many viewers return to my site between one and three weeks after their first visit? And what typically brings them back? These features make SQL one of the primary data analysis tools for both beginners and experts.

Why Do Businesses Choose Relational Databases & SQL?

Nearly all applications have a need to store data so that it can be accessed later.

Let's use Twitter as an example. Every time I read a tweet, the text of my tweet needs to be stored somewhere so that all my followers can read it. Twitter stores a bunch of information about my tweet.


Why Databases?
Udacity @udacity · Aug 8
Show us the apps + projects built during your Nanodegree program! Tag pics of your work #madewithudacity & we may share it right here.
#MadeWithUdacity
2
17
34

Who the author is? Me. The time I wrote it, links to any other relevant tweets if I'm writing a reply or a re-tweet, etcetera. All this information is used to determine who sees my tweet and when they see it. This is the function databases serve.

They store information so that it can be accessed later, and SQL is the language that allows analysts and others to access that information.

Databases have a number of attributes that make them great for this. Most importantly, databases do a lot to check for **data integrity**.

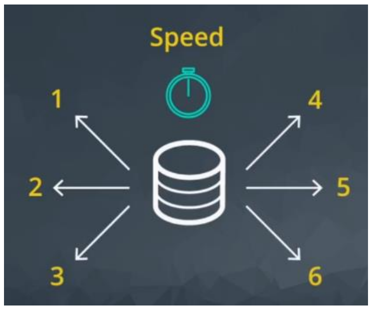
They can make sure that the data entered is consistent.

Data Integrity

Num	Text	Date
1	abc	m/y
2	abc	m/y
3	abc	m/y
4	abc	m/y
5	abc	m/y
6	abc	m/y

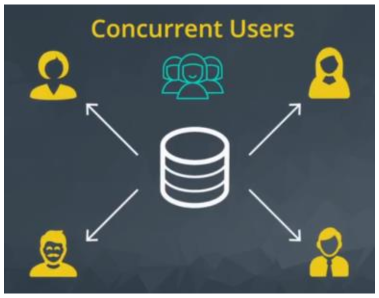
For example, if you're storing the number of children a given person has, the database can enforce that only whole numbers be used, since you can't have half a child.

Databases are also really **fast**, they can perform operations very quickly across very large data sets and can be optimized for extra speed.



Databases are **shared entities**. Many people can access a database concurrently.

More importantly, all of those people will be working with the same data. To facilitate this, databases have lots of administrative features like access controls.



In the Flat File Structure concept, we mentioned that:

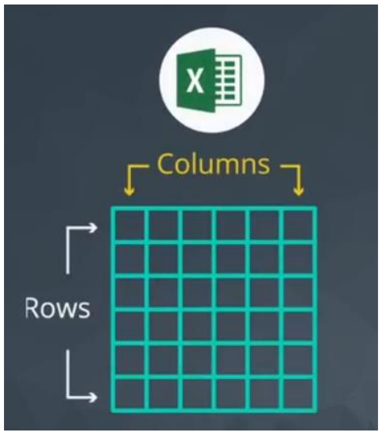
Disadvantages of flat files ... include:

- 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*)

As Derek mentioned, relational databases alleviate these issues. If these issues matter to you, relational databases are a better alternative than saving to a flat file like a CSV file.

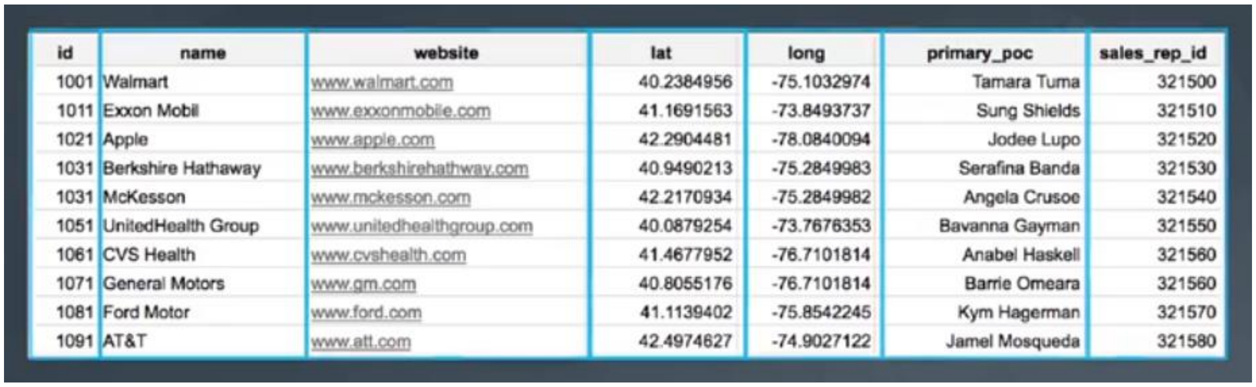
How Relational Databases Store Data

If you've used Excel, you should already be familiar with tables, they're similar to spreadsheets. Tables have rows and columns just like Excel, but have some more rigid rules.



Database tables for instance, are organized by column, and each column must have a unique name. You'll notice that some columns contain numbers, while others contain text.

In a spreadsheet, each cell can have its own data type. But in database tables, all the data in a column must be of the same type.



id	name	website	lat	long	primary_poc	sales_rep_id
1001	Walmart	www.walmart.com	40.2384956	-75.1032974	Tamara Tuma	321500
1011	Exxon Mobil	www.exxonmobile.com	41.1691563	-73.8493737	Sung Shields	321510
1021	Apple	www.apple.com	42.2904481	-78.0840094	Jodee Lupo	321520
1031	Berkshire Hathaway	www.berkshirehathway.com	40.9490213	-75.2849983	Serafina Banda	321530
1031	McKesson	www.mckesson.com	42.2170934	-75.2849982	Angela Crusoe	321540
1051	UnitedHealth Group	www.unitedhealthgroup.com	40.0879254	-73.7676353	Bavanna Gayman	321550
1061	CVS Health	www.cvshealth.com	41.4677952	-76.7101814	Anabel Haskell	321560
1071	General Motors	www.gm.com	40.8055176	-76.7101814	Barrie Omeara	321560
1081	Ford Motor	www.ford.com	41.1139402	-75.8542245	Kym Hagerman	321570
1091	AT&T	www.att.com	42.4974627	-74.9027122	Jamel Mosqueda	321580

This makes performing analysis on database tables simple. While the data type must be consistent, the database doesn't necessarily know that a number means latitude, or that the text is a name of a company. That's why descriptive column names are important.

Types of SQL Statements

The SQL Language has a few different elements, the most basic of which is a statement. Think of a statement as simply a piece of correctly written SQL code.

Statements tell the database what you'd like to do with the data.

The **CREATE** statement for example, is how you make a new table in the database.

The **DROP TABLE** statement is how you remove a table from the database.

In this lesson, we'll focus on the **SELECT** statement, which allows you to read data and display it.

SELECT statements are commonly referred to as **queries**. There are a few other types of statements that SQL understands. They all have different functions, most of which relate to the creation and removal of data.

Eventually, you'll learn how to do that, but most of the analytical work doesn't involve adding, removing, or modifying data within a database.

It requires reading data and manipulating it, but only occasionally updating the underlying source.

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

Data Wrangling in SQL?

Data wrangling can be performed in SQL. We believe that pandas is better equipped for gathering (pandas has a huge simplicity advantage in this area), assessing, and cleaning data, so we usually recommend that you use pandas if given the choice. If wrangling in a work setting, sometimes your tool of choice for data wrangling depends on your company infrastructure, though.

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

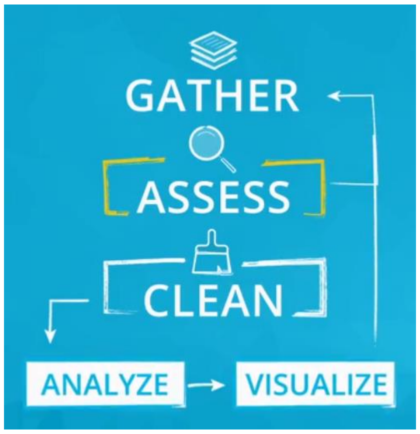
Additional, less common file formats include:

- **Excel files(opens in a new tab)**
- **Pickle files(opens in a new tab)**
- **HDF5 files(opens in a new tab)**
- **SAS files(opens in a new tab)**
- **STATA files(opens in a new tab)**

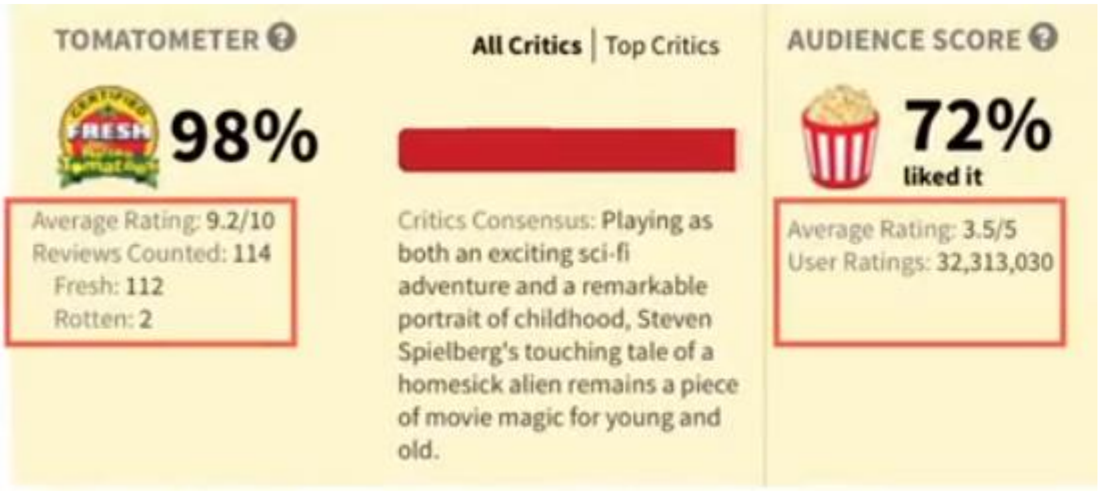
pandas has **functions(opens in a new tab)** to read (and write, to most of them) these files. Also, you now have the foundational understanding of **gathering** and file formats in general, so learning these additional formats won't be too hard if you need them.

You can Iterate.

Remember, data wrangling can be an iterative process. Whether you're in the gather, assess, or clean phase, you can iterate on them. So, you just gathered a bunch of data. For this lesson, I handled the assessing, cleaning, then analyzing, and visualizing. But at any point in that process, I could have stopped gathering more data.



Maybe we wanted different data like scraping the actual average ratings, instead of these critic and audience score metrics. These are the percentage of reviewers that gave the movie a 70% rating or higher.



Average Rating: 9.2/10
Reviews Counted: 114
Fresh: 112
Rotten: 2

Critics Consensus: Playing as both an exciting sci-fi adventure and a remarkable portrait of childhood, Steven Spielberg's touching tale of a homesick alien remains a piece of movie magic for young and old.

Average Rating: 3.5/5
User Ratings: 32,313,030

We get a different take on the same question. Starting small and iterating is very okay.

Gathering Summary

Gather: Summary

Gathering is the first step in the data wrangling process:

- **Gather**
- Assess
- Clean

Depending on the source of your data, and what format it's in, the steps in gathering data vary.

The high-level gathering process:

- obtaining data (downloading a file from the internet, scraping a web page, querying an API, etc.)
- importing that data into your programming environment (e.g. Jupyter Notebook)

Assessing Data

Introduction

Now that you know how to gather data, you're going to master assessing it.

Assessing your data is the second step in the data wrangling process. When assessing, you're like a detective at work. Inspecting your data set for two things: **Data quality issues and lack of tidiness**.

Data that has quality issues has issues with content like missing, duplicate or incorrect data. This is called dirty data.

An untidy or messy data has specific structural issues. These structural issues slow you down when cleaning and analyzing, visualizing or modeling your data later.

You can search for these issues in two ways: **Visually**, with a simple scroll and **programmatically** using code.

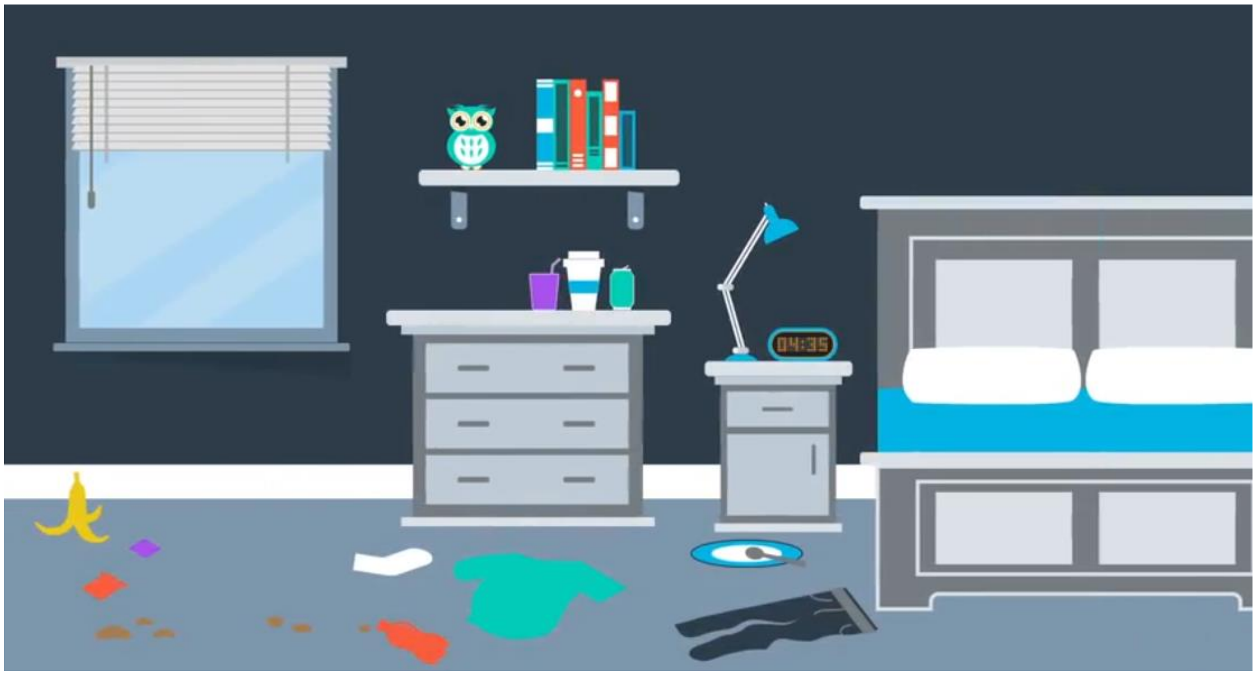
And when you detect an issue, you document it to make cleaning easier. In this lesson, you will assess a pre-gathered data set that's full of issues and you're going to find all of them. Some visually and others using the functions in Python's pandas library. At the end of the lesson, we'll flash forward to analysis and visualization where you'll see how your assessments were vital.

You're going to leave this lesson with an eagle eye for unclean data and there's a lot of it in the world.

Unclean Data: Dirty vs. Messy

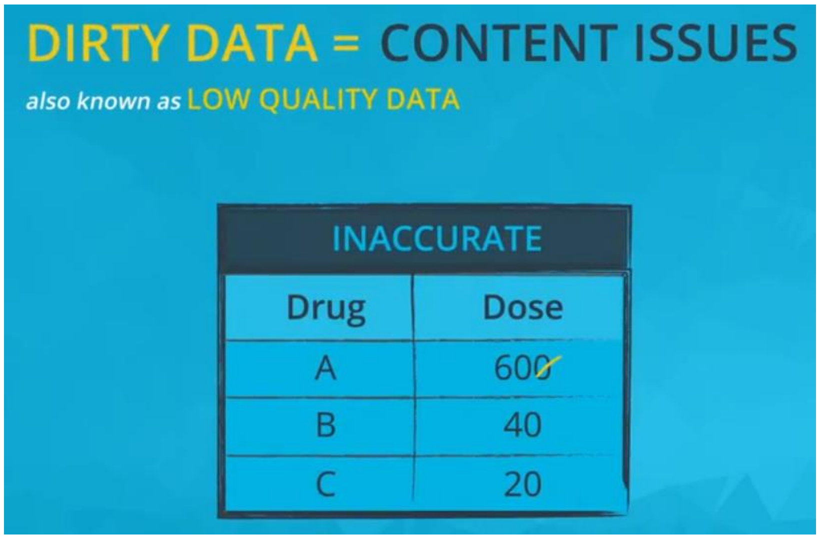
Let's explicitly define the terms dirty data and messy data. To help with this, think of your data like your bedroom. Everyone wants clean data like everyone, or at least your mother wants a clean bedroom, and this bedroom clearly isn't clean.

It's both dirty and messy. A dirty bedroom might have dirty plates and cutlery, garbage like candy wrappers or banana peels, maybe even physical dirt tracked in from outside. These don't belong in your bedroom.



The messy part of the bedroom are different. Messiness is structural or organizational issues. Like those clothes in the ground, clothes belong in your bedroom but not in the ground. Or unmade bed, your bed should probably be made, that's messy. And data can be thought of in the same way.

Dirty data which has issues with its content is often called low quality data and can include things like inaccurate data, corrupted data and duplicate data.



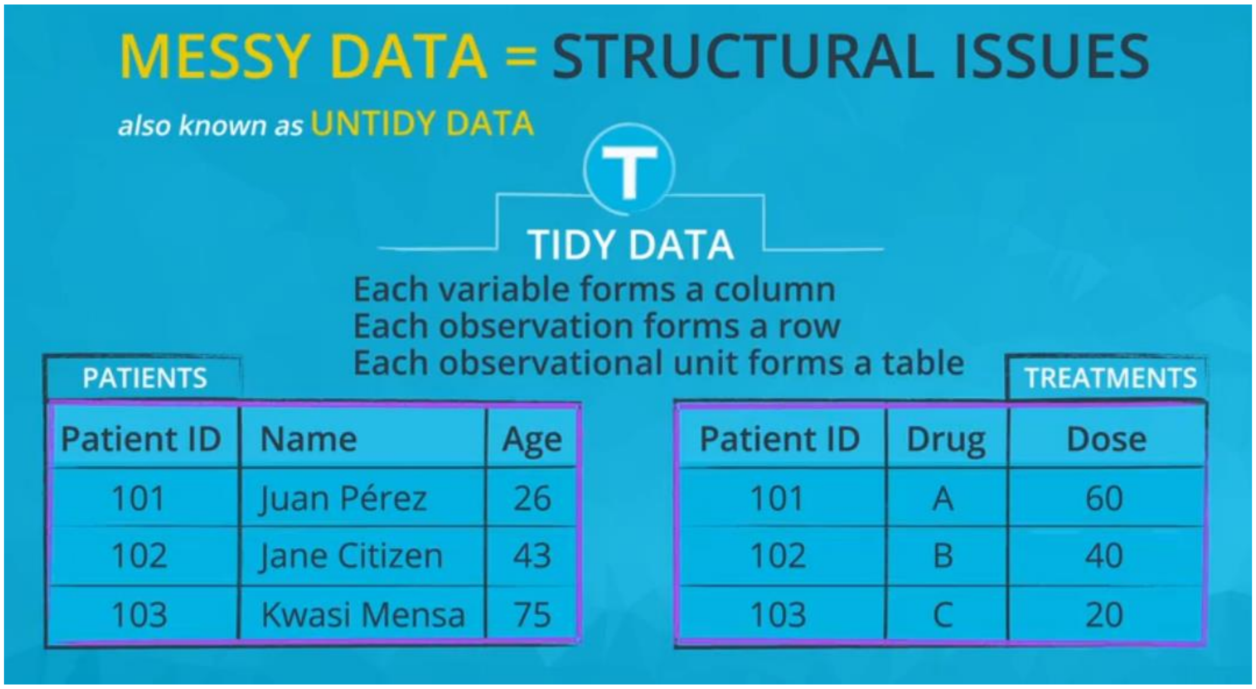
DIRTY DATA = CONTENT ISSUES

also known as LOW QUALITY DATA

INACCURATE

Drug	Dose
A	60~~0~~
B	40
C	20

Messy data on the other hand had issues with its structure. It is often referred to as untidy. Tidy data means each variable forms a column, each observation forms a row and each type of observational unit forms a table.



MESSY DATA = STRUCTURAL ISSUES

also known as UNTIDY DATA

TIDY DATA

Each variable forms a column
Each observation forms a row
Each observational unit forms a table

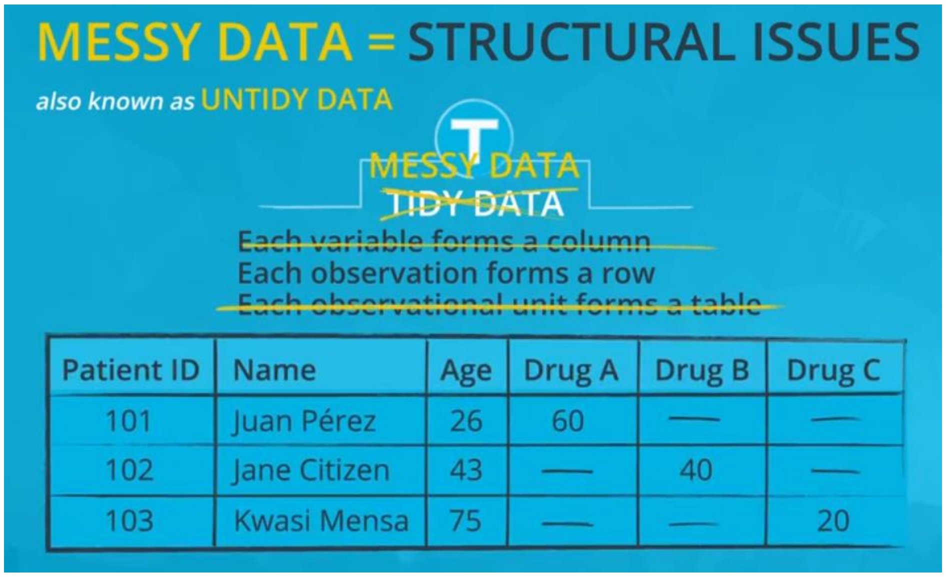
PATIENTS

Patient ID	Name	Age
101	Juan Pérez	26
102	Jane Citizen	43
103	Kwasi Mensa	75

TREATMENTS

Patient ID	Drug	Dose
101	A	60
102	B	40
103	C	20

Any other arrangement is messy which is the exact term that Hadley Wickham, the inventor of the tidy data format uses in his tidy data paper.



MESSY DATA = STRUCTURAL ISSUES

also known as UNTIDY DATA

MESSY DATA

~~TIDY DATA~~

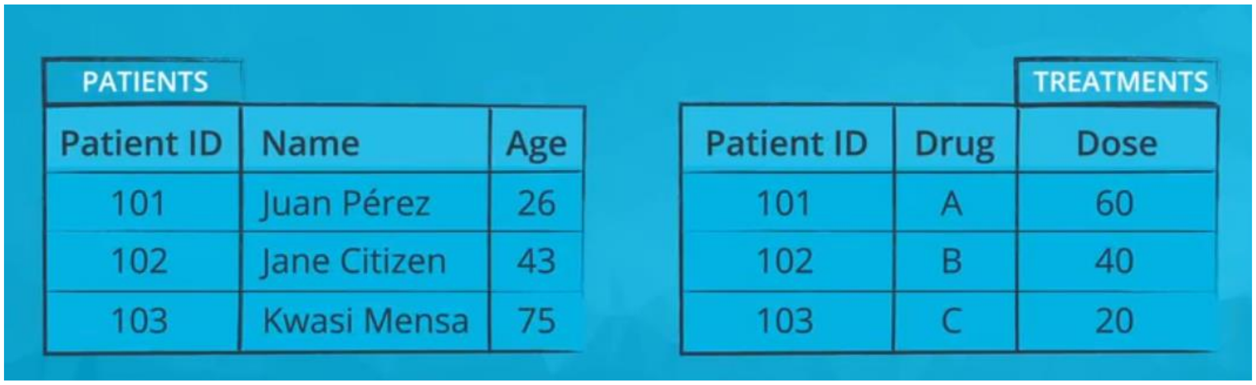
~~Each variable forms a column~~

Each observation forms a row

~~Each observational unit forms a table~~

Patient ID	Name	Age	Drug A	Drug B	Drug C
101	Juan Pérez	26	60	—	—
102	Jane Citizen	43	—	40	—
103	Kwasi Mensa	75	—	—	20

And again, here's what tidy looks like in comparison.



PATIENTS

Patient ID	Name	Age
101	Juan Pérez	26
102	Jane Citizen	43
103	Kwasi Mensa	75

TREATMENTS

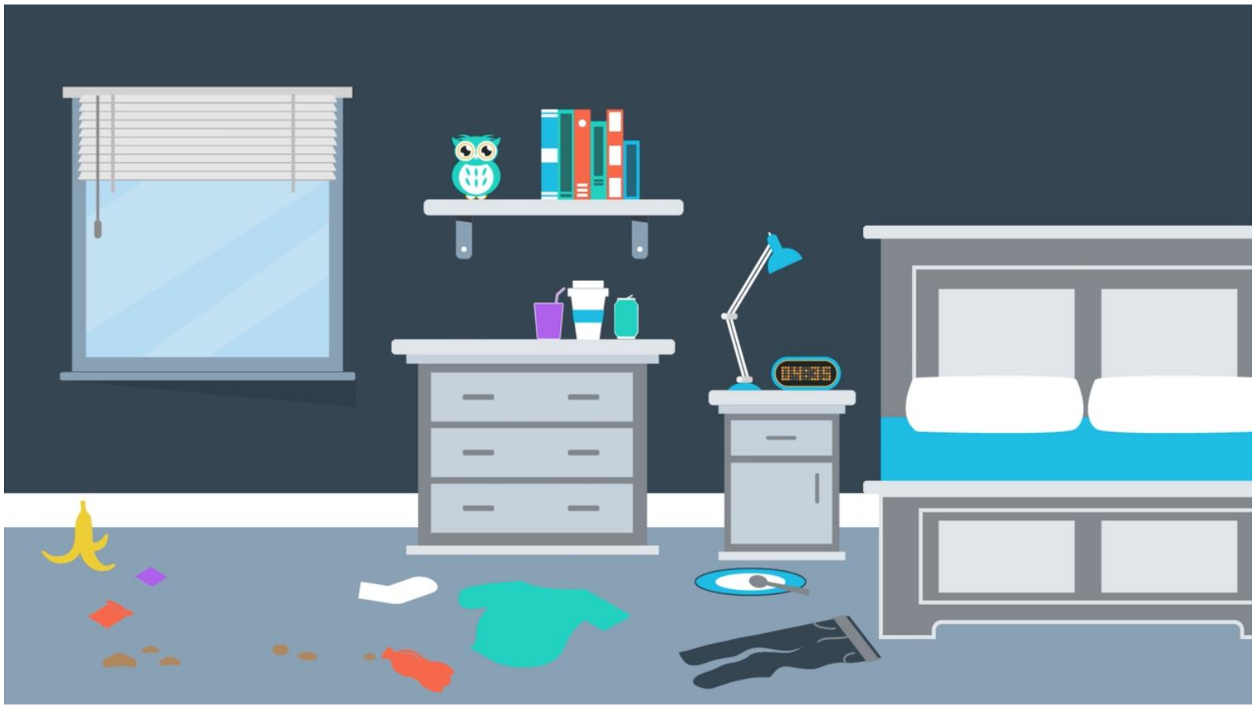
Patient ID	Drug	Dose
101	A	60
102	B	40
103	C	20

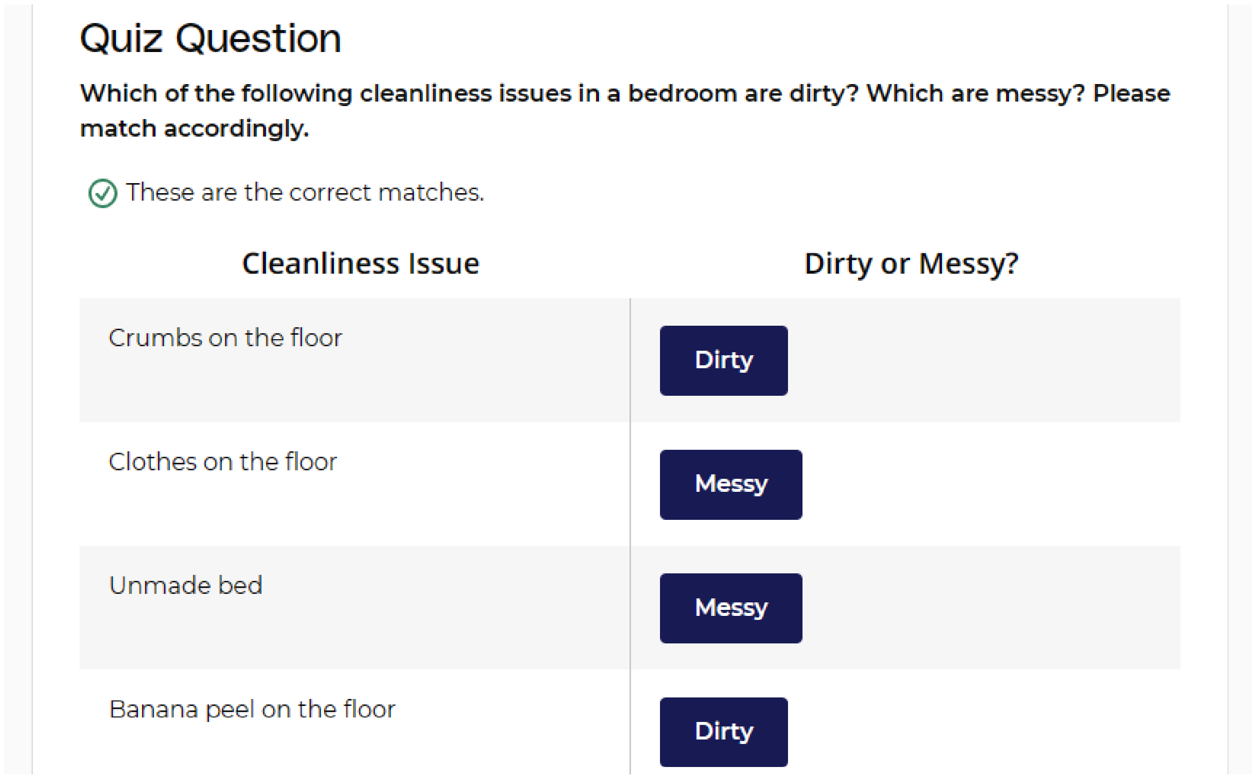
Just like you can clean your bedroom, you can clean your data and it takes time and skill. Fortunately, programming tools like Python and its libraries can help us, like a vacuum might help you vacuum up dirt, or a dresser might help you organize your clothes.

Immaculately clean data is the goal and it's a beautiful thing once you have it.

Quiz 1

Use the image below to answer the first quiz.





Quiz Question

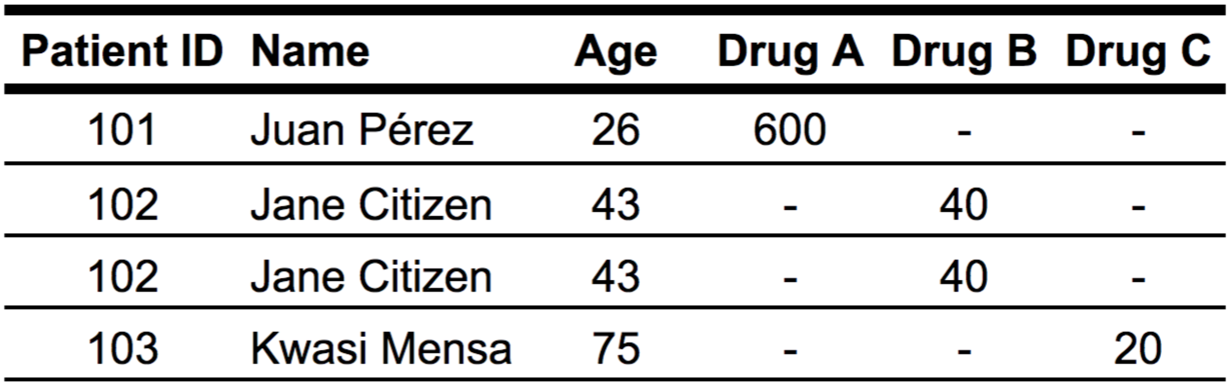
Which of the following cleanliness issues in a bedroom are dirty? Which are messy? Please match accordingly.

✓ These are the correct matches.

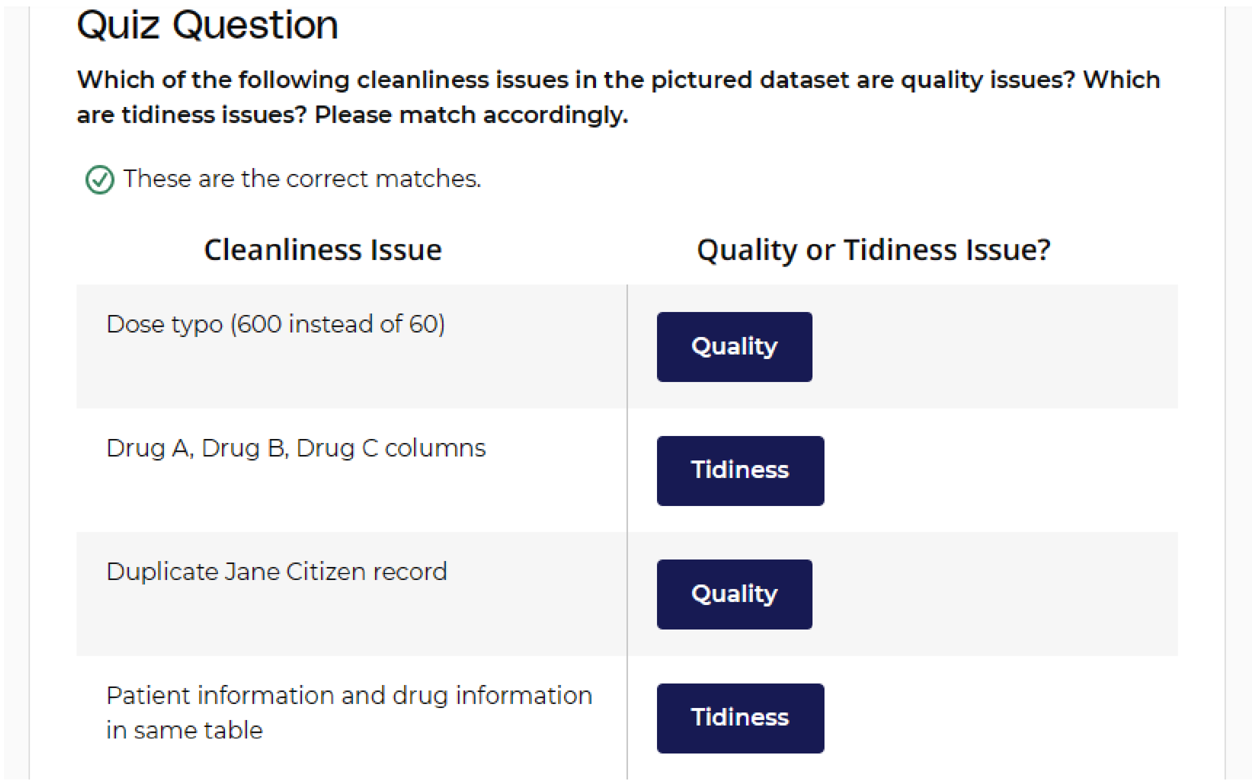
Cleanliness Issue	Dirty or Messy?
Crumbs on the floor	Dirty
Clothes on the floor	Messy
Unmade bed	Messy
Banana peel on the floor	Dirty

Quiz 2

Use the image below to answer the second quiz.



Patient ID	Name	Age	Drug A	Drug B	Drug C
101	Juan Pérez	26	600	-	-
102	Jane Citizen	43	-	40	-
102	Jane Citizen	43	-	40	-
103	Kwasi Mensa	75	-	-	20



Quiz Question

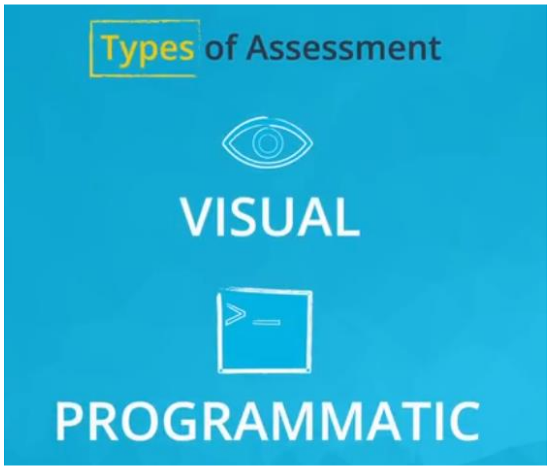
Which of the following cleanliness issues in the pictured dataset are quality issues? Which are tidiness issues? Please match accordingly.

✓ These are the correct matches.

Cleanliness Issue	Quality or Tidiness Issue?
Dose typo (600 instead of 60)	Quality
Drug A, Drug B, Drug C columns	Tidiness
Duplicate Jane Citizen record	Quality
Patient information and drug information in same table	Tidiness

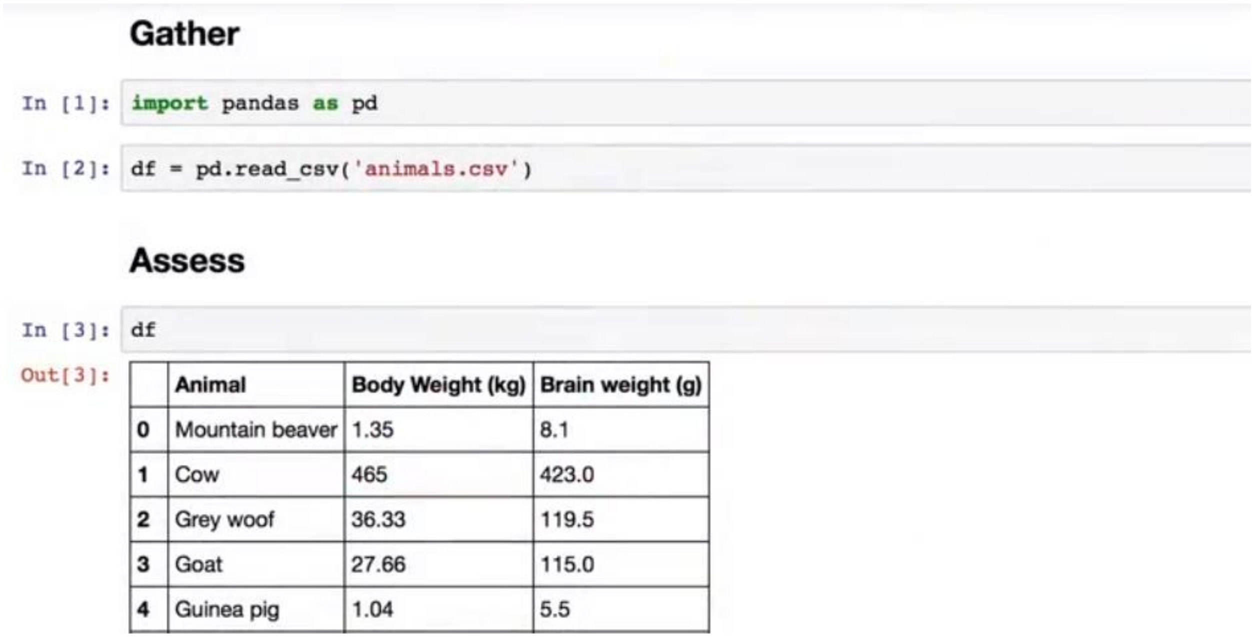
Assessment: Types vs. Steps

There are two types or styles of assessing your data. You can assess it visually and you can assess it programmatically.



Visual assessment is just opening it and looking through the data in its entirety, in pandas, a text editor or a spreadsheet application for example.

This is what it looks like in pandas for this animals.csv file, which has three columns; animal, body weight in kilograms then brain weight in grams.



Gather

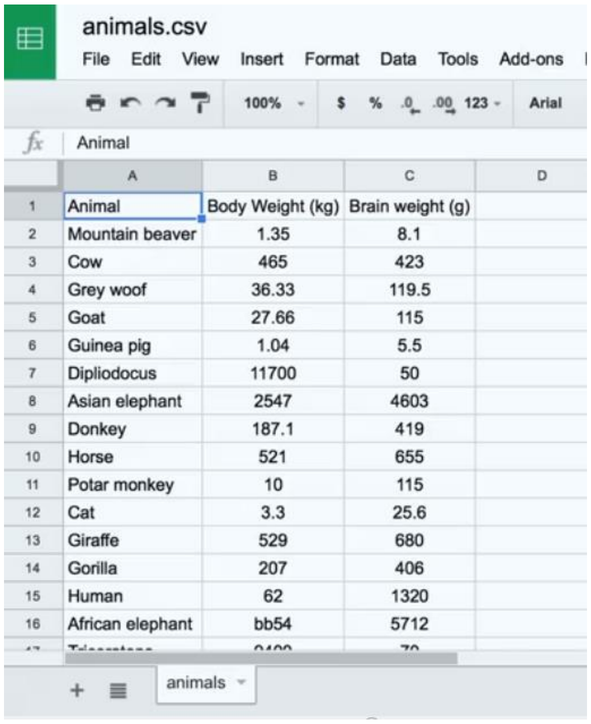
```
In [1]: import pandas as pd
In [2]: df = pd.read_csv('animals.csv')
```

Assess

```
In [3]: df
Out[3]:
```

	Animal	Body Weight (kg)	Brain weight (g)
0	Mountain beaver	1.35	8.1
1	Cow	465	423.0
2	Grey woof	36.33	119.5
3	Goat	27.66	115.0
4	Guinea pig	1.04	5.5

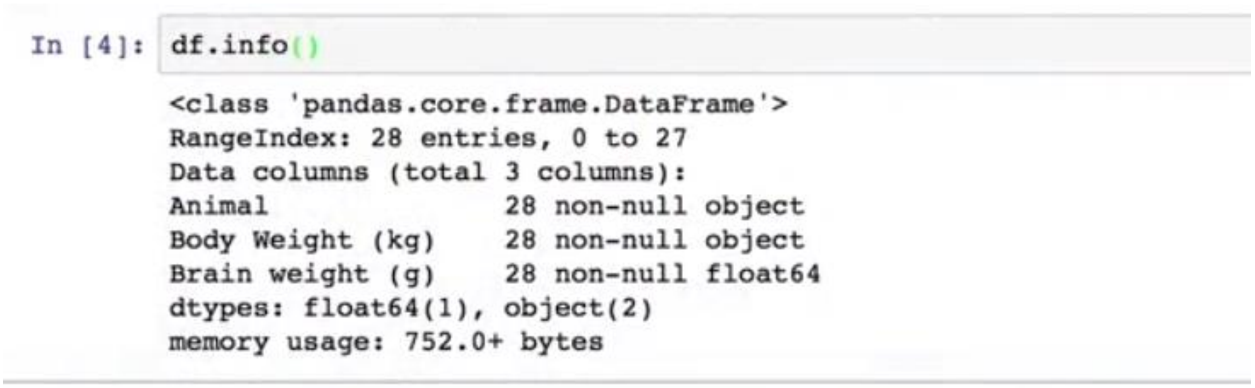
This data set's pretty small, but for larger data sets pandas will collapse rows and columns. So in those cases, visual assessment is best done in something like Google Sheets for example.


animals.csv
File Edit View Insert Format Data Tools Add-ons
100% $ % .0 .00 123 Arial
fx Animal
animals

	A	B	C	D
1	Animal	Body Weight (kg)	Brain weight (g)	
2	Mountain beaver	1.35	8.1	
3	Cow	465	423	
4	Grey woof	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	
11	Potar monkey	10	115	
12	Cat	3.3	25.6	
13	Giraffe	529	680	
14	Gorilla	207	406	
15	Human	62	1320	
16	African elephant	bb54	5712	
[illegible]	[illegible]	[illegible]	[illegible]	

Programmatic assessment like its name suggests, is anything that uses code to view specific parts of the data, like using functions or methods to summarize the data.

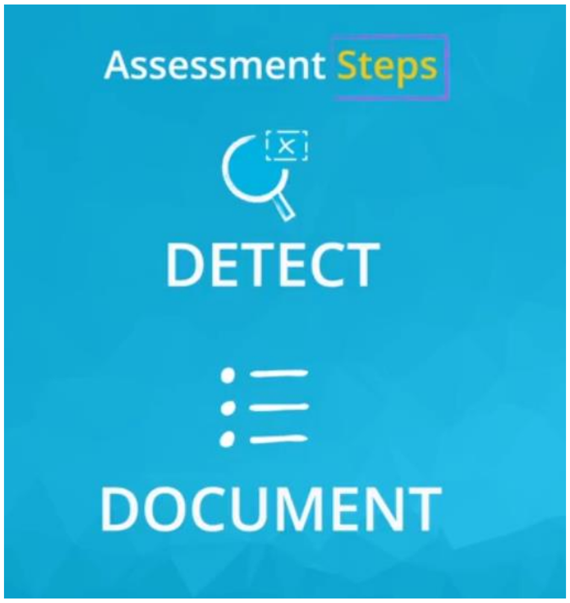
Like this ***info*** method for example.



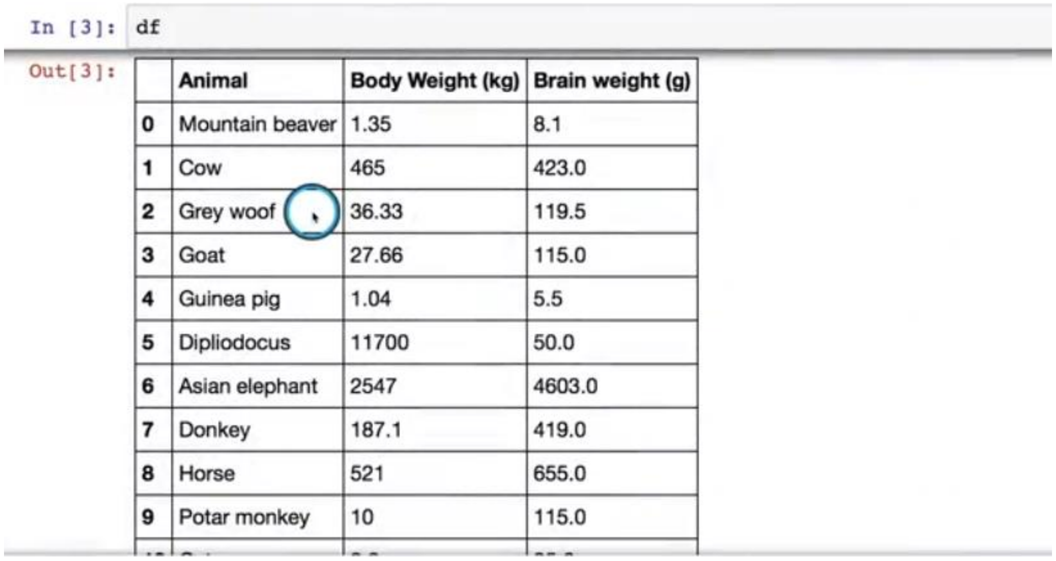
```
In [4]: df.info()

        <class 'pandas.core.frame.DataFrame'>
        RangeIndex: 28 entries, 0 to 27
        Data columns (total 3 columns):
        Animal              28 non-null object
        Body Weight (kg)    28 non-null object
        Brain weight (g)    28 non-null float64
        dtypes: float64(1), object(2)
        memory usage: 752.0+ bytes
```

You can even plot the data, but plotting isn't done very often when wrangling. This is more for exploring during exploratory data analysis, which we'll get into later.

Regardless of the type of assessment you're using, assessing data can be broken down into two steps; detecting an issue and documenting that issue.



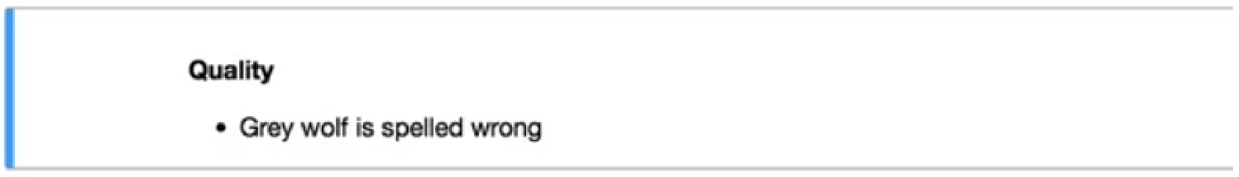
This process applies whether you're assessing your data visually or programmatically, which are the two types of assessment again. For example, we will visually assess the entirety of this data set in pandas and we detect that Gray wolf is spelt wrong, Gray woof.



```
In [3]: df
Out[3]:
```

	Animal	Body Weight (kg)	Brain weight (g)
0	Mountain beaver	1.35	8.1
1	Cow	465	423.0
2	Grey woof	36.33	119.5
3	Goat	27.66	115.0
4	Guinea pig	1.04	5.5
5	Dipliodocus	11700	50.0
6	Asian elephant	2547	4603.0
7	Donkey	187.1	419.0
8	Horse	521	655.0
9	Potar monkey	10	115.0

Then we document that issue with a bullet point.


Quality

- Grey wolf is spelled wrong

I'd like to separate the issues by quality and tidiness. So Gray wolf is spelled wrong, that's our documentation. This is simply what we see on observation.

You don't have to write how to fix it which is part of the cleaning step in this data wrangling framework.

So, an observation would be, Gray wolf is spelled wrong as opposed to something like change Gray woof to Gray wolf which would be more of an action item rather than an observation.

Why should we first document unclean issues we observe, rather than just write what we need to do to fix the issues? When your data's issues get complicated, writing how to fix each can get confusing, lengthy, and time-consuming. It can get overwhelming trying to think of how to clean something complicated immediately after documenting it.

If you are separating the *assessing* and *cleaning* steps of data wrangling, as we are in this lesson, writing only observations as a first step is good practice.

If you choose to assess an issue then immediately clean that issue (which is very much allowed), you can skip the observation and go straight to defining how to clean it (which is part of the *Define-Code-Test* cleaning framework you'll see in Lesson 4).

Note: Visualizing your data (i.e., creating plots) is part of Programmatic rather than Visual Assessment. Tricky! This is because plotting data requires coding, or programming.

So, assessment is often guided by what you need to analyze.

Assessing .vs Exploring

There is a subtle difference between assessing your data and exploring your data.

Assessing is part of the data wrangling process, while exploring is part of Exploratory Data Analysis or EDA. Data wrangling is about gathering the right pieces of data, assessing your data's quality and structure, then modifying your data to make it clean. But the assessments you make and convert to cleaning operations, won't make your analysis, VIS, or model better though. The goal is just to make them work.

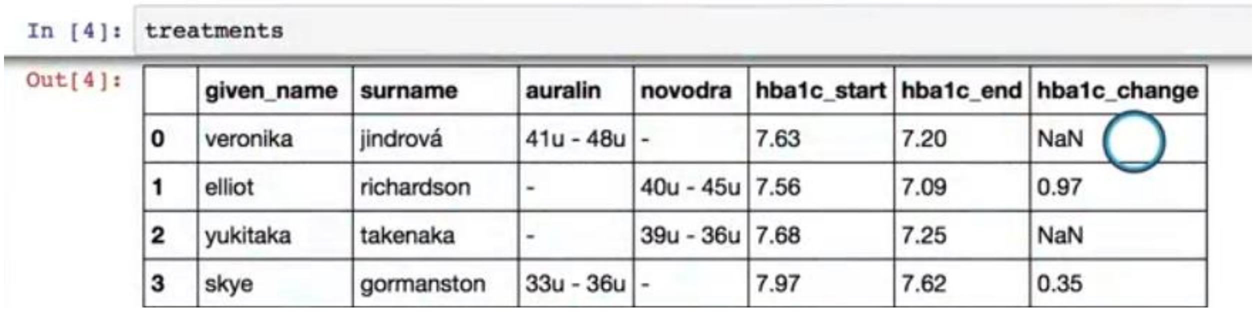
EDA is about exploring your data to later augment it, to maximize the potential of your analyses, visualizations and models. When exploring, simple visualizations are often used to summarize your data's main characteristics. From there, you can do things like remove outliers and create new and more descriptive features from existing data, also known as feature engineering.

In practice, wrangling and EDA can occur together, but we're going to separate them for teaching purposes.

Data Quality Dimensions

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

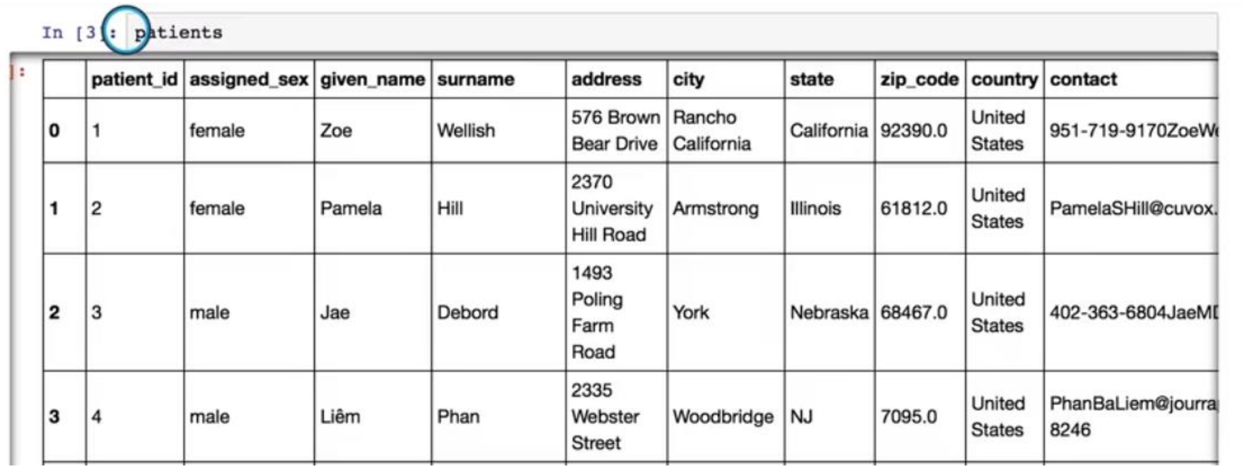
- **Completeness**: do we have all of the records that we should? Do we have missing records or not? Are there specific rows, columns, or cells missing?
    - An example of this in our dataset for this lesson is the missing hba1c change variable, represented by NaN's here



```
In [4]: treatments
Out[4]:
```

	given_name	surname	auralin	novodra	hba1c_start	hba1c_end	hba1c_change
0	veronika	jindrová	41u - 48u	-	7.63	7.20	NaN
1	elliot	richardson	-	40u - 45u	7.56	7.09	0.97
2	yukitaka	takenaka	-	39u - 36u	7.68	7.25	NaN
3	skye	gormanston	33u - 36u	-	7.97	7.62	0.35

- **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).
    - A good example of this is a primary key in a database, or unique key constraints in tables. In this dataset, that means there can't be duplicate patient IDs. If systems are good, they aren't always though, invalid record creation will be denied. For example, in this dataset, if you try to create another patient with the patient ID of one, you'd be denied, ideally at least.
    - Another example in this dataset of invalid data is our zip code data here. This is invalidated because zip codes must be five digits in an integer. Whereas this one is actually a float and sometimes four digits.

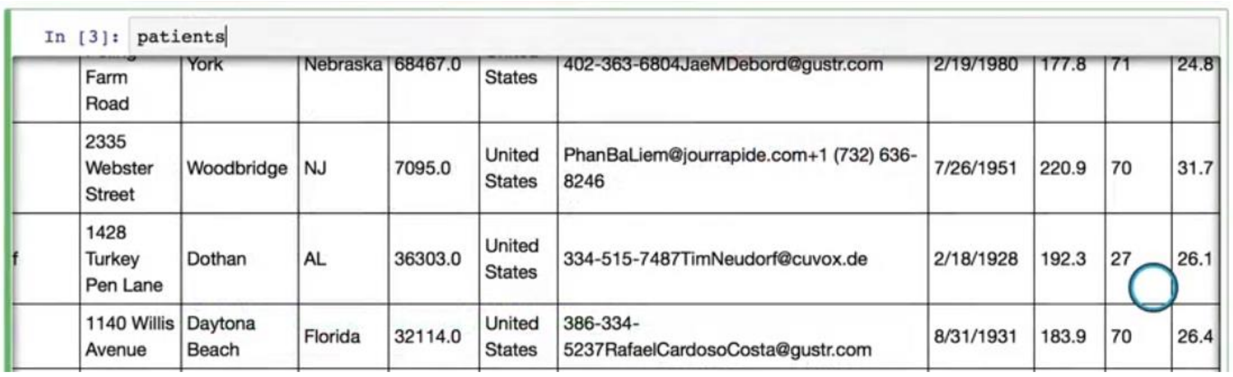


```
In [3]: patients
```

	patient_id	assigned_sex	given_name	surname	address	city	state	zip_code	country	contact
0	1	female	Zoe	Wellish	576 Brown Bear Drive	Rancho California	California	92390.0	United States	951-719-9170ZoeW
1	2	female	Pamela	Hill	2370 University Hill Road	Armstrong	Illinois	61812.0	United States	PamelaSHill@cuvox.
2	3	male	Jae	Debord	1493 Poling Farm Road	York	Nebraska	68467.0	United States	402-363-6804JaeMI
3	4	male	Liêm	Phan	2335 Webster Street	Woodbridge	NJ	7095.0	United States	PhanBaLiem@jourra 8246

- **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.
    - For example, looking at weight in this table. Imagine at the scale that each patient was measured on was slightly faulty and overestimated each patient's weight by five pounds. The weight's slightly off, but still valid.

- The inaccuracy issue that we already identified was the height entry for this patient being 27 inches. It's possible for an adult human to be 27 inches tall. The shortest adult ever measured 21 inches approximately. And since the inclusion criteria for this clinical trial required patients to be at least 18 years old, we can ignore the possibility of this being a child's height. So again, 27 inches is possible, but it's unlikely. And we already know that the correct height is actually 72 inches and that these digits were switched around. So, this is accurate data.

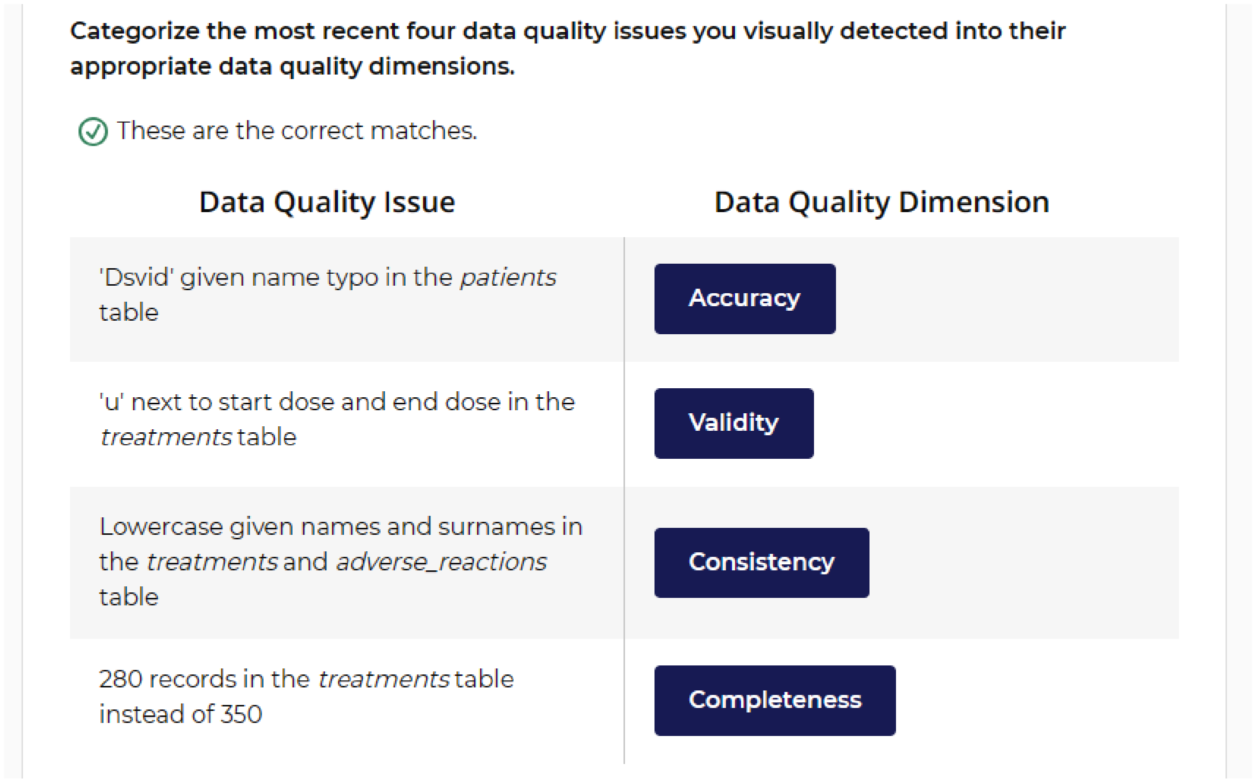


```
In [3]: patients
```

	Farm Road	York	Nebraska	68467.0	States	402-363-6804JaeMDebord@gustr.com	2/19/1980	177.8	71	24.8
	2335 Webster Street	Woodbridge	NJ	7095.0	United States	PhanBaLiem@jourrapide.com+1 (732) 636-8246	7/26/1951	220.9	70	31.7
f	1428 Turkey Pen Lane	Dothan	AL	36303.0	United States	334-515-7487TimNeudorf@cuvox.de	2/18/1928	192.3	27	26.1
	1140 Willis Avenue	Daytona Beach	Florida	32114.0	United States	386-334-5237RafaelCardosoCosta@gustr.com	8/31/1931	183.9	70	26.4

- **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.
    - In this dataset, we've identified inconsistent representations of state, full state name versus abbreviation. For example, California and CA, or New York and NY. That's inconsistent. This inconsistency means we can't analyze our dataset based on state. At least until we clean this issue.

Quiz



Categorize the most recent four data quality issues you visually detected into their appropriate data quality dimensions.

✓ These are the correct matches.

Data Quality Issue	Data Quality Dimension
'Dsvid' given name typo in the *patients* table	Accuracy
'u' next to start dose and end dose in the *treatments* table	Validity
Lowercase given names and surnames in the *treatments* and *adverse_reactions* table	Consistency
280 records in the *treatments* table instead of 350	Completeness

How Data Gets Dirty and Messy

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

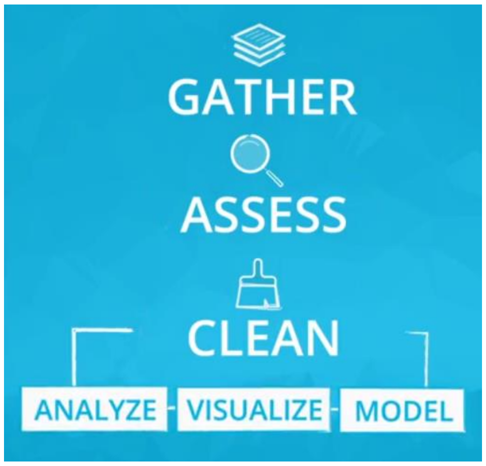
Messy data = untidy data = structural issues

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

You Can Iterate!

Reminder, data wrangling can be iterative. You just finished a round of assessments. You can move on to cleaning at this point, or revisit gathering if you determine that you didn't gather enough data.

You can even assess some more. Once you go through each step of the data wrangling process once, you can revisit them at any time, even after we've finished wrangling and have moved on to analysis, visualization or modeling.



The concept of iterating isn't that applicable for clinical trials given the rigor involved in their planning. But, theoretically, the following situations could arise that require iteration:

- Maybe you (as the data analyst or data scientist on the clinical trial research team) realized your statistical power calculations were wrong, and you needed to recruit more patients to make your study statistically significant. You'd also have to do revisit *gathering* in this scenario.
- Maybe you realized you were missing a key piece of patient information, like patient blood type (again, unlikely given the rigor of clinical trials, but mistakes happen) because you discovered new research that related insulin resistance to blood type. You'd also have to do revisit *gathering* in this scenario.
- Maybe you finished assessing, started cleaning, and spotted another data quality issue. Revisiting *assessing* to add these assessments to your notes is fine.

Data Cleaning

Now that you know how to gather and assess data you're going to master cleaning it. Cleaning your data is the third and final step in the data wrangling process.

This is where the quality and tidiness issues you identified in the Assess step are remedied. It can be done manually in spreadsheet programs or text editors, but data cleaning is often best done using code and in three steps.

First you define how you're going to clean issue in words, then you convert these words to code, and finally test your data to make sure if that code worked.

Programmatic Data Cleaning:
DEFINE
CODE
TEST

Watching your data transform to clean feels like magic sometimes. In this lesson you'll take your assessments from the last lesson and define code and test cleaning operations for each.

You also become intimately familiar with the cleaning and testing functions in Python in the pandas library.

At the end the lesson we'll flash forward to analysis and visualization and you'll see how cleaning was absolutely necessary.

You're going to leave with the tools required to clean pretty much everything you come across in the future.

And since it's the last lesson of the course, you're also going to leave an expert data wrangler.

Data Cleaning Process

Think of programmatic data cleaning as its own separate process within data wrangling. It also has three steps: **defining, coding, and testing**.

First, you'll define a data cleaning plan in writing. You convert your assessments into cleaning tasks by writing little how-to guides. Think of this like pseudo code.

In the future this plan also serves as documentation so others or yourself in the future can look at your work and reproduce it.

Second, you'll translate these words to code and actually run it.

And finally, you'll test your dataset often using code to make sure your cleaning code worked.

This is kind of like a revisit of the assess step. Similar functions are used too.

Imputing means filling in missing data values with other values, using some appropriate method.

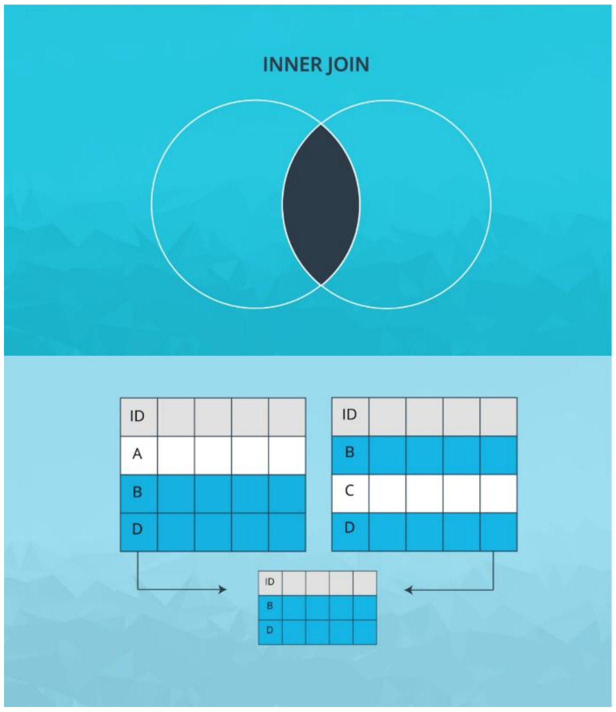
Types of Merges

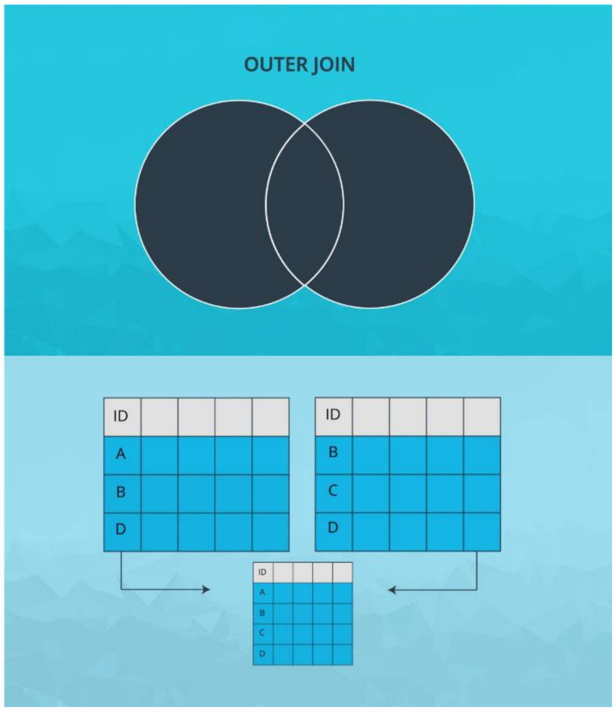
So far, we've learned about appending dataframes. Now we'll learn about **pandas Merges(opens in a new tab)**, a different way of combining dataframes. This is similar to the database-style "join." If you're familiar with SQL, this **comparison with SQL(opens in a new tab)** may help you connect these two.

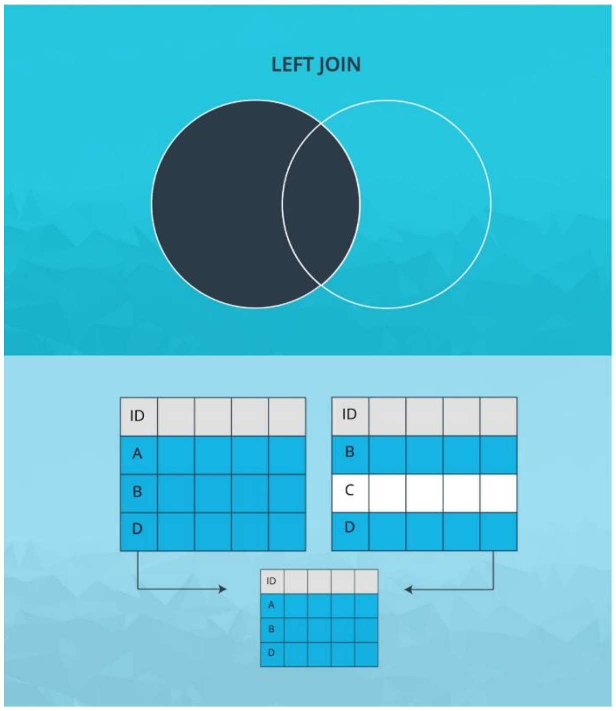
Here are the four types of merges in pandas. Below, "key" refers to common columns in both dataframes that we're joining on.

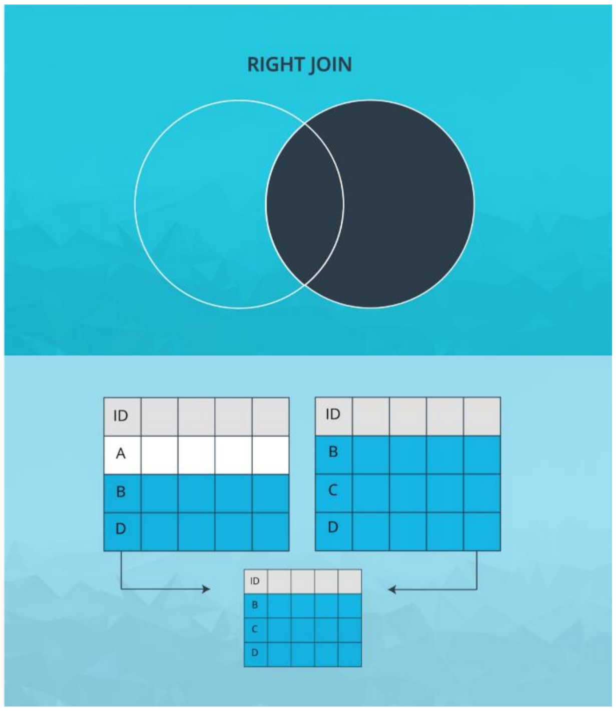
1. Inner Join - Use intersection of keys from both frames.
2. Outer Join - Use union of keys from both frames.
3. Left Join - Use keys from left frame only.
4. Right Join - Use keys from right frame only.

Below are diagrams to visualize each type.









Read the documentation for pandas Merges **here(opens in a new tab)**.