7 Machine Learning: The DL on ML

In order to create a more just technological world, we need more diverse voices at the table when we create technology. To do this, we need to employ conventional solutions like reducing barriers to entry and addressing the "leaky pipeline" issues that make mid-career professionals drop out or stall on their way to the top. I think we also need to add an unconventional solution: we need to add nuance to the way we talk about all things digital. This is easier said than done. One illustration of the difficulty of talking about computer science comes from an *xkcd* comic by Randall Munro. In it, a woman sits at a computer and a man stands behind her:

"When a user takes a photo, the app should check whether they're in a national park," says the man.

"Sure, easy GIS lookup," says the woman. "Gimme a few hours."

"And check whether the photo is a bird," says the man.

"I'll need a research team and five years," says the woman.

"In CS, it can be hard to explain the difference between the easy and the virtually impossible," reads the caption.[1]

Because it's complicated to explain why a computer might have trouble recognizing a bird in an image, or differentiating between a parrot and guacamole, we need more people (data journalists, perhaps?) explaining complex technical topics in plain language to demystify the more arcane corners of the AI world.

The difficulty of talking about computation has led to a lot of misunderstandings. One recurrent idea in this book is that computers are good at some things and very bad at others, and social problems arise from situations in which people misjudge how suitable a computer is for performing the task. The classic example of a thing that is very simple for people but very complex for computers is navigating a room with toys all over the

floor. The average toddler can navigate a room without stepping on toys (though of course she might not choose to do so). A robot can't. To get the robot to navigate the toy-strewn floor, we would have to program in all of the information about the toys and their exact dimensions and have the robot calculate a path around the toys. If the toys moved, the robot would need its schema updated. Self-driving cars, which we'll discuss in chapter 8, work like this hypothetical robot in the playroom: they constantly update their preprogrammed map of the world.

There are also predictable pitfalls to the robot method, as people who own both Roomba robotic vacuum cleaners and pets have discovered. When a pet leaves something disgusting on the floor, the Roomba will smear it all over the house. "Quite honestly, we see this a lot," a spokesman from iRobot, the company that makes the Roomba, said to the *Guardian* in August 2016. "We generally tell people to try not to schedule your vacuum if you know you have dogs that may create such a mess. With animals anything can happen."[2]

I can use a euphemism to talk about the disgusting things that pets do because everyday language allows us to refer to things without using precise words. If I say that my dog is adorable, but also gross, you will understand. You can hold the two competing ideas in your head at the same time, and you can guess what I mean by *gross*. There are no such euphemisms in mathematical language. In mathematical language, everything is highly precise. Part of the communication problem that exists in computational culture derives from the imprecision of everyday language and the precision of mathematical language. One example: in programming, there is a concept called a *variable*. You assign a value to a variable by writing something like "X = 2," and then you can use X in a routine. There are two kinds of variables: variables that change, which are called *variables*, and variables that don't change, which are called *constants*. This makes perfect sense to a programmer: a variable can be a constant. To a nonprogrammer, it likely doesn't make sense: constant is the opposite of varying, so a thing that varies is not a thing that is unvarying. It's confusing.

This naming problem is not new. Language has always evolved along with science. In biology, cells got their name because the man who discovered them in 1665, Robert Hooke, was reminded of the walls of monks' cells in monasteries. The naming problem is particularly acute right now, however, because of the rapid pace of technological change. We're adopting

new computational concepts and new hardware at a breathtaking rate, and people are inventing names for new things based on concepts or artifacts that already exist.

Although computer scientists and mathematicians tend to be talented at computer science and math, as a group they tend not to be sensitive to the nuances of language. If something needs a name, they don't obsess over picking the perfect name that has ideal connotations and Latin roots and what have you. They just pick a name, usually one that has to do with something they like. Python the programming language is named after Monty Python the comedy troupe (Monty Python is the ur-comedy text in computer science, like Star Wars is the ur-narrative text.) Django, a web framework, is named after Django Reinhardt the jazz guitarist, a favorite of the Django framework's inventor. Java the language is named after coffee. JavaScript, an unrelated language, was invented around the same time as Java and is also (unfortunately) named after coffee.

As the term *machine learning* has spread from computer science circles into the mainstream, a number of issues have arisen from linguistic confusion. Machine learning (ML) implies that the computer has agency and is somehow sentient because it "learns," and *learning* is a term usually applied to sentient beings like people (or partially sentient beings like animals). However, computer scientists know that machine "learning" is more akin to a metaphor in this case: it means that the machine can improve at its programmed, routine, automated tasks. It doesn't mean that the machine acquires knowledge or wisdom or agency, despite what the term *learning* might imply. This type of linguistic confusion is at the root of many misconceptions about computers.[3]

Imagination also complicates things. How you define AI depends on what you want to believe about the future. One of Marvin Minsky's students, Ray Kurzweil, is a proponent of the *singularity theory*, a hypothetical future merging of man and machine that he thinks will be achieved by 2045. (Kurzweil is famous for inventing a musical synthesizer that sounds like a grand piano.) Singularity is a major preoccupation of science fiction. I was once interviewed for a futurists' summit, and the interviewer asked me about the paperclip theory: What if you invented a machine that made paperclips, and then you taught the machine to want to make paperclips, and then you taught the machine to want to make other things, and then the machine made lots of other machines and all the machines took over?

"Is that the singularity?" the interviewer asked. "And aren't you worried about it?" That's fun to think about, but it's also not reasonable. You can unplug the paperclip machine. Problem solved. Also, this is a purely hypothetical situation. *It's not real*.

As psychologist Stephen Pinker told *IEEE Spectrum*, the magazine of the Institute of Electrical and Electronics Engineers (IEEE), in a special issue on the singularity: "There is not the slightest reason to believe in a coming singularity. The fact that you can visualize a future in your imagination is not evidence that it is likely or even possible. Look at domed cities, jet-pack commuting, underwater cities, mile-high buildings, and nuclear-powered automobiles—all staples of futuristic fantasies when I was a child that have never arrived. Sheer processing power is not a pixie dust that magically solves all your problems."[4]

Facebook's Yann LeCun is also a singularity skeptic. He told *IEEE Spectrum*: "There are people that you'd expect to hype the Singularity, like Ray Kurzweil. He's a futurist. He likes to have this positivist view of the future. He sells a lot of books this way. But he has not contributed anything to the science of AI, as far as I can tell. He's sold products based on technology, some of which were somewhat innovative, but nothing conceptually new. And certainly he has never written papers that taught the world anything on how to make progress in AI."[5] Reasonable, smart people disagree about what will happen in the future—in part because nobody can see the future.

I'm going to try to bring some clarity to the situation by defining machine learning and showing you an example of how someone might perform machine learning on a dataset. I'm going to explain machine learning a few different ways and also demonstrate some code. It's going to get technical. If the technical parts get confusing, don't worry; you can skim them first and return to them later.

AI enjoyed a popularity bump in 2017 in contrast to many years of what people call an *AI winter*. In the mainstream, people mostly ignored AI for the first decade of the 2000s. The Internet was the popular thing technologically, then mobile devices, and those were the focus of our collective imagination. In the middle of the 2010s, however, people started talking about machine learning. Suddenly, AI was on fire again. AI startups were founded and acquired. IBM's Watson beat a human player at Jeopardy!; an algorithm outfoxed a human player at playing Go. Even the words *machine learning* were cool. A machine could learn! The promise was delivered!

At first, I wanted to believe that some genius had figured out the truly hard problem of making a machine think—but when I looked closer, it turned out that the reality was far more nuanced. What had happened was that scientists had redefined the term *machine learning* so that it referred to their work. They used the term so much that its meaning changed.

This happens. Language is fluid. A good example is the word *literally*, which used to mean the opposite of *figuratively*. In the 1990s, if you said, "My mouth was literally on fire after eating that ghost pepper," it meant that there were actual flames in your mouth and you were talking from the other side of recovery from third-degree burns. However, in the 2000s, a critical mass of people started using *literally* as a synonym for figuratively and for emphasis. "I was ready to literally kill someone if I had to listen to that John Mayer song one more time" became understood as "I would really prefer not to listen to another John Mayer song," rather than a statement about murder or mayhem.

The term *machine learning* entered the lexicon in 1959, according to the *Oxford English Dictionary* (OED). The OED began including *machine learning* as a phrase in its third edition, published in 2000. The OED defines machine learning as follows:

machine learning n. Computing the capacity of a computer to learn from experience, i.e. to modify its processing on the basis of newly acquired information.

1959 IBM Jrnl. 3 211/1 We have at our command computers with adequate data-handling ability and with sufficient computational speed to make use of machine-learning techniques.

1990 New Scientist 8 Sept. 78/1 When Doug Lenat of Stanford developed Eurisko, a second generation machine learning system, he thought that he had created a real intellectual.[6]

This definition is true, but it doesn't quite capture the way that contemporary computer scientists use the term. A more comprehensive definition is found in Oxford's *A Dictionary of Computer Science*:

machine learning

A branch of artificial intelligence concerned with the construction of programs that learn from experience. Learning may take many forms, ranging from learning from examples and learning by analogy to autonomous learning of concepts and learning by discovery.

Incremental learning involves continuous improvement as new data arrives while *one-shot* or *batch learning* distinguishes a training phase from the application phase.

Supervised learning occurs when the training input has been explicitly labeled with the classes to be learned.

Most learning methods aim to demonstrate generalization whereby the system develops efficient and effective representations that encompass large chunks of closely related data.[7]

This is closer, but still not quite right. The documentation for scikit-learn, a popular software library for machine learning in Python, has a different definition: "Machine learning is about learning some properties of a data set and applying them to new data. This is why a common practice in machine learning to evaluate an algorithm is to split the data at hand into two sets, one that we call the training set on which we learn data properties and one that we call the testing set on which we test these properties."[8]

It's rare that a term has so much disagreement across different sources. The definition of a dog, for example, is pretty consistent across texts. However, machine learning is so new, and there is so little consensus, that it's not surprising that the linguistic definitions haven't caught up to reality.

Tom M. Mitchell, the E. Fredkin University Professor in the Machine Learning Department of Carnegie Mellon University's School of Computer Science, offers a good definition of machine learning in "The Discipline of Machine Learning." He writes: "We say that a machine learns with respect to a particular task T, performance metric P, and type of experience E, if the system reliably improves its performance P at task T, following experience E. Depending on how we specify T, P, and E, the learning task might also be called by names such as data mining, autonomous discovery, database updating, programming by example, etc."[9] I think this is a good definition because Mitchell uses very precise language to define learning. When a machine "learns," it doesn't mean that the machine has a brain made out of metal. It means that the machine has become more accurate at performing a single, specific task according to a specific metric that a person has defined.

This kind of learning does *not* imply intelligence. As programmer and consultant George V. Neville-Neil writes in the *Communications of the ACM*:

We have had nearly 50 years of human/computer competition in the game of chess, but does this mean that any of those computers are intelligent? No, it does not—for two reasons. The first is that chess is not a test of intelligence; it is the test of a particular skill—the skill of playing chess. If I could beat a Grandmaster at chess and yet not be able to hand you the salt at the table when asked, would I be intelligent? The second reason is that thinking chess was a test of intelligence was based on a false

cultural premise that brilliant chess players were brilliant minds, more gifted than those around them. Yes, many intelligent people excel at chess, but chess, or any other single skill, does not denote intelligence.[10]

There are three general types of machine learning: supervised learning, unsupervised learning, and reinforcement learning. Here are definitions of each from a widely used textbook called *Artificial Intelligence: A Modern Approach* by UC Berkeley professor Stuart Russell and Google's director of research, Peter Norvig:

> Supervised learning: The computer is presented with example inputs and their desired outputs, given by a "teacher," and the goal is to learn a general rule that maps inputs to outputs.
>
> Unsupervised learning: No labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal in itself (discovering hidden patterns in data) or a means toward an end (feature learning).
>
> Reinforcement learning: A computer program interacts with a dynamic environment in which it must perform a certain goal (such as driving a vehicle or playing a game against an opponent). The program is provided feedback in terms of rewards and punishments as it navigates its problem space.[11]

Supervised learning is the most straightforward. The machine is provided with the training data and labeled outputs. We essentially tell the machine what we want to find, then fine-tune the model until we get the machine to predict what we know to be true.

All three kinds of machine learning depend on *training data*, known datasets for practicing and tuning the machine-learning model. Let's say that my training data is a dataset of one hundred thousand credit card company customers. The dataset contains the data you would expect a credit card company to have for a person: name, age, address, credit score, interest rate, account balance, name(s) of any joint signers on the account, a list of charges, and a record of payment amounts and dates. Let's say that we want the ML model to predict who is likely to pay their bill late. We want to find these people because every time someone pays a bill late, the interest rate on the account increases, which means the credit card company makes more money on interest charges. The training data has a column that indicates who in this group of one hundred thousand *has* paid their bills late. We split the training data into two groups of fifty thousand names each: the training set and the test data. Then, we run a machine-learning algorithm against the training set to construct a model, a black box, that predicts what we already know. We can then apply the model to the test data and see the

model's prediction for which customers are likely to pay late. Finally, we compare the model's prediction to what we know is true—the customers in the test data who actually paid late. This gives us a score that measures the model's precision and recall. If we as model makers decide that the model's precision/recall score is high enough, we can deploy the model on real customers.

A handful of different machine-learning algorithms are available to apply to datasets. You may have come across some of the names, which include random forest, decision tree, nearest neighbor, naive Bayes, or hidden Markov. An *algorithm*, remember, is a series of steps or procedures that the computer is instructed to follow. In machine learning, the algorithm is coupled with variables to create a mathematical model. A wonderful explanation of models is found in Cathy O'Neil's *Weapons of Math Destruction*. O'Neil explains that we model things unconsciously all the time. When I decide what to make for dinner, I make a model: what food is in my refrigerator, what dishes I could possibly make with that food, who the people eating that night are (usually my husband and son and me), and what their food preferences are. I evaluate the various dishes and recall how each performed in the past—who took seconds of what, and what items are on the ever-changing list of shunned foods: cashews, frozen vegetables, coconut, organ meats. By deciding what to make based on what I have and what people like, I'm optimizing my meal choices for a set of features. Building a mathematical model means formalizing the features and the choices in mathematical terms.[12]

Let's say that I want to "do" machine learning. The first thing I do is grab a dataset. A variety of interesting datasets are available for machine-learning practice; they are collected in online repositories. There are datasets of facial expressions, of pets, or of YouTube videos. There are datasets of emails sent by people who worked at a failed company (Enron), datasets of newsgroup conversations in the 1990s (Usenet), datasets of friendship networks from failed social network companies (Friendster), datasets of movies that people watched on streaming services (Netflix), datasets of people saying common phrases in different accents, or datasets of people's messy handwriting. These datasets are collected from active corporations, from websites, from university researchers, from volunteers, and from defunct corporations. This small number of iconic datasets is posted online and the datasets form the backbone of all contemporary artificial intelligence. You might even find your

own data in them. A friend of mine once found a video of herself as a toddler in a behavioral science archive; her mother had participated in a parent-child behavioral study when my friend was little. Researchers still had the video and still used it for drawing conclusions about the world.

Now, let's go through a classic practice exercise: we'll use machine learning to predict who survived the *Titanic* crash. Think about what happened on the *Titanic* after it hit the iceberg. Did you picture Leonardo di Caprio and Kate Winslet sliding across the decks of the ship? That's not real—but it probably colors your recall of the event, if you've seen the movie as many times as I have. It's quite likely that you've seen the movie at least once. *Titanic* earned $659 million and $1.5 billion overseas, making it the biggest movie in the world in 1997 and the second-highest-grossing film ever worldwide. (*Titanic* director James Cameron also holds the number-one spot for his other blockbuster, *Avatar*.) The film stayed in theaters for almost a year, fueled in part by young people who went to the theater to watch it over and over again.[13] *Titanic* the movie has become a part of our collective memory, just like the actual *Titanic* maritime disaster. Our brains quite commonly confuse actual events with realistic fiction. It's unfortunate, but it's normal. This confusion complicates the way we perceive risk.

We draw conclusions about risk based on *heuristics*, or informal rules. These heuristics are affected by stories that are easy to recall and by emotionally resonant experiences. For example: When he was a little boy, *New York Times* columnist Charles Blow was attacked by a vicious dog. The dog almost tore his face off. As an adult, he writes in his memoir, he remains wary of strange dogs.[14] This makes perfect sense. Being a small child attacked by a large animal is traumatic, and of course it would be the first thing someone would think of when seeing a dog for the rest of his life. Reading the book, I empathized with the little boy and felt scared when he felt scared. The day after I read Blow's memoir, I saw a man walking a dog without a leash in a park near my house—and I immediately thought of Blow and how other people who are afraid of dogs would be made uncomfortable by the fact that this dog was not on a leash. I wondered if the dog would go berserk and, if so, what would happen. The story affected my perception of risk. This is the same thinking that leads people to carry pepper spray after watching a lot of episodes of *Law & Order: SVU* or to check the back seat of the car for nasty surprises after watching a horror movie. The technical

name is the *availability heuristic*.[15] The stories that spring to mind first are the ones we tend to think are the most important or occur most frequently.

Perhaps because it features so prominently in our collective imagination, the *Titanic* disaster is commonly used for teaching machine learning. Specifically, a list of the passengers on the *Titanic* is used to teach students how to generate predictions using data. It works well as a class exercise because almost all of the students have seen *Titanic* or know about the disaster. This is valuable for an instructor because you don't have to spend too much class time going over the historical context: you can get right to the fun part, which is the prediction.

I'm going to take you through the fun part using supervised learning. I think it is important to see exactly what happens when someone does machine learning. There are plenty of sites online that have ML tutorials if you're interested in going through the exercise yourself. I'm going to take you through a tutorial from a site called DataCamp, which was recommended as a first step for competing in data-science competitions by a different site, Kaggle.[16] Kaggle, which is owned by Google's parent company, Alphabet, is a site in which people compete to get the highest score for analyzing a dataset. Data scientists use it to compete in teams, sharpen their skills, or practice collaborating. It's also useful for teaching students about data science or for finding datasets.

We're going to do a DataCamp *Titanic* tutorial using Python and a few popular Python libraries: pandas, scikit-learn, and numpy. A *library* is a little bucket of functions sitting somewhere on the Internet. When we import a library, we make its functions available to the program we're writing. One way to think about it is to think about a physical library. I'm a member of the New York Public Library (NYPL). Whenever I go to stay somewhere for more than a week, for work or for vacation, I generally try to go to the local library and get a library card. Signing up for a local library card allows me to use all the books and resources available at that library. For the time that I'm a local library member, I can use all my core NYPL resources *plus* the unique resources of the local library. In a Python program, we start with a whole bunch of built-in functions: those are the NYPL. Importing a new library is like signing up for the local library card. Our program can use all the good stuff in the core Python library *plus* the nifty functions written by the researchers and open-source developers who made and published the scikit-learn library, for example.

Pandas, another library we'll use, has a container called a *DataFrame* that "holds" a set of data. This type of container is also called an *object*, as in *object-oriented programming*. *Object* is a generic term in programming, just as it is in the real world. In programming, an *object* is a conceptual wrapper for a little package of data, variables, and code. Having the label *object* gives us something to hold on to. We need to conceptualize our package of bits as something in order to think about it and talk about it.

The first thing we do is break our data into two sets: training data and test data. We're going to develop a model, train it on the training data, then run it on the test data. Remember how there is general AI and narrow AI? This is narrow. Let's start by typing the following:

```
import pandas as pd
import numpy as np
from sklearn import tree, preprocessing
```

We've just imported several libraries that we'll use for our analysis. We use an alias, *pd*, for pandas, and the alias *np* for numpy. We now have access to all of the functions in pandas and numpy. We can choose to import all of the functions or just a few. From scikit-learn, we'll import only two functions. One is named *tree* and the other is named *preprocessing*.

Next, let's import the data from a comma-separated values (CSV) file that is also sitting somewhere on the Internet. Specifically, this CSV file is sitting on a server owned by Amazon Web Services (AWS). We can tell because the base URL of the file (the first part after http://) is s3.amazonaws.com. A CSV file is a file of structured data in which each column is separated by a comma. We're going to import two different Titanic data files from AWS. One is a training data set, another is a test data set. Both data sets are in CSV format. Let's import the data:

```
train_url =
"http://s3.amazonaws.com/assets.datacamp.com/course/Kaggle/
train.csv"
train = pd.read_csv(train_url)

test_url = "http://s3.amazonaws.com/assets.datacamp.com/
course/Kaggle/test.csv"
test = pd.read_csv(test_url)
```

pd.read_csv() means "please invoke the read_csv() function, which lives in the pd (pandas) library." Technically, we created a DataFrame object and called one of its built-in methods. Regardless, the data is now imported

into two variables: *train* and *test*. We'll use the data in the *train* variable to create the model, and then we'll use the data in the *test* variable to test our model's accuracy.

Let's see what's in the *head*, or the first few lines, of the training data:

```
print(train.head())
```

	PassengerId	Survived	Pclass \
0	1	0	3
1	2	1	1
2	3	1	3
3	4	1	1
4	5	0	3

	Name	Sex	Age	SibSp \
0	Braund, Mr. Owen Harris	male	22.0	1
1	Cumings, Mrs. John Bradley (Florence Briggs Th…	female	38.0	1
2	Heikkinen, Miss. Laina	female	26.0	0
3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1
4	Allen, Mr. William Henry	male	35.0	0

	Parch	Ticket	Fare	Cabin	Embarked
0	0	A/5 21171	7.2500	NaN	S
1	0	PC 17599	71.2833	C85	C
2	0	STON/O2. 3101282	7.9250	NaN	S
3	0	113803	53.1000	C123	S
4	0	373450	8.0500	NaN	S

It looks like the data is twelve columns. The columns are labeled PassengerId, Survived, Pclass, Name, Sex, Age, SibSp, Parch, Ticket, Fare, Cabin, and Embarked. What do these column headings mean?

To answer this, we need a data dictionary, which is provided with most datasets. The data dictionary reveals the following:

```
Pclass = Passenger Class (1 = 1st; 2 = 2nd; 3 = 3rd)
Survived = Survival (0 = No; 1 = Yes)
Name = Name
Sex = Sex
```

```
Age = Age (in years; fractional if age less than one (1). If
the age is estimated, it is in the form xx.5)
Sibsp = Number of Siblings/Spouses Aboard
Parch = Number of Parents/Children Aboard
Ticket = Ticket Number
Fare = Passenger Fare (pre-1970 British pound)
Cabin = Cabin number
Embarked = Port of Embarkation (C = Cherbourg; Q = Queenstown;
S = Southampton)
```

For most of the columns, we have data. For some column values, we do not have data. For PassengerId 1, Mr. Owen Harris Braund, the value for Cabin is NaN. This means "not a number." NaN is different than zero; zero is a number. NaN means that there is no value for this variable. This distinction might seem unimportant for everyday life, but it's crucially important in computer science. Remember that mathematical language is precise. For example, NULL indicates an empty set, which is also different than NaN or zero.

Let's see what's in the first few lines of the test dataset:

```
print(test.head())
```

	PassengerId	Pclass	Name	Sex \
0	892	3	Kelly, Mr. James	male
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female
2	894	2	Myles, Mr. Thomas Francis	male
3	895	3	Wirz, Mr. Albert	male
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female

	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	34.5	0	0	330911	7.8292	NaN	Q
1	47.0	1	0	363272	7.0000	NaN	S
2	62.0	0	0	240276	9.6875	NaN	Q
3	27.0	0	0	315154	8.6625	NaN	S
4	22.0	1	1	3101298	12.2875	NaN	S

As you can see, *test* has the same type of data as *train*, minus the Survived column. Great! Our goal is to create a Survived column in the *test* data that contains a prediction for each passenger. (Of course, someone already

knows which passengers in the *test* data set survived—but it wouldn't be much of a tutorial if the data set already contained the answers.)

Next, we're going to run some basic summary statistics on the training dataset in order to get to know it a little better. When data journalists do this, we call it *interviewing the data*. We interview data just like we might interview a human source. A human has a name, an age, a background; a dataset has a size and a number of columns. Asking a column of data about its average value is a bit like asking someone to spell their last name.

We can get to know our data a bit by running a function called *describe* that assembles some basic summary statistics and puts them into a handy table, as follows:

```
train.describe()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

The training dataset has 891 records. Of these, only 714 records show the age of the passenger. For the data we have available, the average age of the passengers is 29.699118; normal people would say that the average age is thirty.

A few of these statistics require interpretation: Survived has a min of 0 and a max of 1. In other words, it is a Boolean value. Either someone survived (1), or they didn't (0). We can calculate an average, which turns out to be 0.38. Similarly, we can calculate an average for Pclass, or passenger class. Passengers' tickets were for first, second, or third class. The average doesn't literally mean that someone traveled 2.308 class.

Now that we've gotten to know our data a little bit, it's time to do some analysis. Let's first look at the number of passengers. We can use a function called *value_counts* to do this. Value_counts will show how many values there are for each distinct category in a column. In other words, how many passengers are traveling in each passenger class? Let's find out:

```
train["Pclass"].value_counts()
1    216
2    184
3    491
Name: Pclass, dtype: int64
```

The training data shows 491 passengers traveling third class, 184 passengers traveling second class, and 216 passengers traveling first class.

Let's look at the numbers for survival:

```
train["Survived"].value_counts()
0    549
1    342
Name: Survived, dtype: int64
```

The training data shows that 549 people perished and 342 survived.

Let's see those numbers normalized:

```
print(train["Survived"].value_counts(normalize = True))
0    0.616162
1    0.383838
Name: Survived, dtype: float64
```

Sixty-two percent of passengers perished, and 38 percent survived. In other words, most people died in the disaster. If we were to make a prediction about whether a random passenger survived, we'd likely predict that they did not survive.

We could stop here if we wanted. We just drew a conclusion that would allow us to make a reasonable prediction. We can do better, however, so let's keep going. Are there any factors that might help improve the prediction? In addition to survival, we have some other columns in the data: Pclass, Name, Sex, Age, SibSp, Parch, Ticket, Fare, Cabin, and Embarked.

Pclass is a proxy for the socioeconomic class of the passengers. That might be useful as a predictor. We could guess that first-class passengers got on the boats before third-class passengers. Sex is also a reasonable guess for a predictor. We know that "women and children first" was a principle used during maritime disasters. This principle dates to 1852, when the British HMS *Birkenhead*, a troop ship, ran aground off the coast of South Africa. It's not a uniformly applied principle, but it's recurrent enough to use for social analysis.

Now, let's do some comparisons to see if we can find variables that seem predictive:

```
# Passengers that survived vs passengers that passed away
print(train["Survived"].value_counts())
0    549
1    342
Name: Survived, dtype: int64

# As proportions
print(train["Survived"].value_counts(normalize = True))
0    0.616162
1    0.383838
Name: Survived, dtype: float64

# Males that survived vs males that passed away
print(train["Survived"][train["Sex"] == 'male'].value_counts())
0    468
1    109
Name: Survived, dtype: int64

# Females that survived vs females that passed away
print(train["Survived"][train["Sex"] == 'female'].value_counts())
1    233
0     81
Name: Survived, dtype: int64

# Normalized male survival
print(train["Survived"][train["Sex"] == 'male'].value_counts
(normalize=True))
0    0.811092
1    0.188908
Name: Survived, dtype: float64

# Normalized female survival
print(train["Survived"][train["Sex"] == 'female'].value_counts
(normalize=True))
1    0.742038
0    0.257962
Name: Survived, dtype: float64
```

We can see that 74 percent of females survived, and only 18 percent of males survived. Therefore, for a random person, we might adjust our guess to say that they survived if they were female, but not if they were male.

Remember that the goal at the beginning of this section was to create a Survived column in the test data that contains a prediction for each passenger. At this point, we could create a Survived column and fill in "1"

(meaning "yes, this passenger survived") for 74 percent of the females and "0" (meaning "no, this passenger did not survive") for the remaining females. We could fill in "1" for 18 percent of the male passengers and "0" for 81 percent of the remaining males.

But we won't, because that would mean assigning probable outcomes randomly based only on gender. We know there are other factors in the data that influence the outcome. (If you're truly curious to see the nitty-gritty of how this is determined, I encourage you to look at the DataCamp tutorial or something similar online.) What about women traveling third class? Women traveling first class? Women traveling with spouses? Women traveling with children? This quickly becomes tedious to calculate manually, so we're going to train a model to do the guessing for us based on the factors that we know.

To construct the model, we're going to use a *decision tree*, a type of algorithm. Remember, there are a handful of algorithms that are standard in machine learning. They have names like decision tree, or random forest, or artificial neural network, or naive Bayes, or k-nearest neighbor, or deep learning. Wikipedia's list of machine-learning algorithms is quite comprehensive.

These algorithms come packaged into software like pandas. Very few people write their own algorithms for machine learning; it's much easier to use one that already exists. Writing a new algorithm is like writing a new programming language. You really have to care *a lot* and you have to devote a lot of time to doing it. I'm going to wave my hands and say "math" to explain what happens inside the model. Sorry. If you really want to know, I encourage you to read more about it. It's very interesting, but it's beyond the scope of the current discussion.

Now, let's train the model on the training data. We know from our exploratory analysis that the features that matter are fare class and sex. We want to create a guess for survival. We already know whether the passengers in the training data survived or not. We're going to make the model guess, then compare the guesses to reality. Whatever the percentage is that we get right is our accuracy number.

Here's an open secret of the big data world: *all data is dirty*. All of it. Data is made by people going around and counting things or made by sensors that are made by people. In every seemingly orderly column of numbers, there is noise. There is mess. There is incompleteness. This is life. The

problem is, dirty data doesn't compute. Therefore, in machine learning, sometimes we have to make things up to make the functions run smoothly.

Are you horrified yet? I was, the first time I realized this. As a journalist, I don't get to make anything up. I need to fact-check each line and justify it for a fact-checker or an editor or my audience—but in machine learning, people often make stuff up when it's convenient.

Now, in physics you can do this. If you want to find the temperature at point A inside a closed container, you take the temperature at two other equidistant points (B and C) and assume that the temperature at point A is halfway between the B and C temperatures. In statistics … well, this is how it works, and the *missing-ness* contributes to the inherent uncertainty of the whole endeavor. We'll use a function called *fillna* to fill in all of the missing values:

```
train["Age"] = train["Age"].fillna(train["Age"].median())
```

The algorithm can't run with missing values. Thus, we need to make up the missing values. Here, DataCamp recommends using the median.

Let's take a look at the data to see what's in there:

```
# Print the train data to see the available features
print(train)
```

	PassengerId	Survived	Pclass \
0	1	0	3
1	2	1	1
2	3	1	3
3	4	1	1
4	5	0	3
5	6	0	3
6	7	0	1
7	8	0	3
8	9	1	3
9	10	1	2
10	11	1	3
11	12	1	1
12	13	0	3
13	14	0	3
14	15	0	3
15	16	1	2
16	17	0	3

	PassengerId	Survived	Pclass \
17	18	1	2
18	19	0	3
19	20	1	3
20	21	0	2
21	22	1	2
22	23	1	3
23	24	1	1
24	25	0	3
25	26	1	3
26	27	0	3
27	28	0	1
28	29	1	3
29	30	0	3
..	…	…	…
861	862	0	2
862	863	1	1
863	864	0	3
864	865	0	2
865	866	1	2
866	867	1	2
867	868	0	1
868	869	0	3
869	870	1	3
870	871	0	3
871	872	1	1
872	873	0	1
873	874	0	3
874	875	1	2
875	876	1	3
876	877	0	3
877	878	0	3
878	879	0	3
879	880	1	1
880	881	1	2
881	882	0	3
882	883	0	3
883	884	0	2
884	885	0	3
885	886	0	3
886	887	0	2
887	888	1	1

	PassengerId	Survived	Pclass \
888	889	0	3
889	890	1	1
890	891	0	3

	Name	Sex	Age	SibSp \
0	Braund, Mr. Owen Harris	male	22.0	1
1	Cumings, Mrs. John Bradley (Florence Briggs Th…	female	38.0	1
2	Heikkinen, Miss. Laina	female	26.0	0
3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1
4	Allen, Mr. William Henry	male	35.0	0
5	Moran, Mr. James	male	28.0	0
6	McCarthy, Mr. Timothy J	male	54.0	0
7	Palsson, Master. Gosta Leonard	male	2.0	3
8	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0
9	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1
10	Sandstrom, Miss. Marguerite Rut	female	4.0	1
11	Bonnell, Miss. Elizabeth	female	58.0	0
12	Saundercock, Mr. William Henry	male	20.0	0
13	Andersson, Mr. Anders Johan	male	39.0	1
14	Vestrom, Miss. Hulda Amanda Adolfina	female	14.0	0
15	Hewlett, Mrs. (Mary D Kingcome)	female	55.0	0
16	Rice, Master. Eugene	male	2.0	4
17	Williams, Mr. Charles Eugene	male	28.0	0
18	Vander Planke, Mrs. Julius (Emelia Maria Vande…	female	31.0	1
19	Masselmani, Mrs. Fatima	female	28.0	0
20	Fynney, Mr. Joseph J	male	35.0	0
21	Beesley, Mr. Lawrence	male	34.0	0
22	McGowan, Miss. Anna “Annie”	female	15.0	0
23	Sloper, Mr. William Thompson	male	28.0	0
24	Palsson, Miss. Torborg Danira	female	8.0	3
25	Asplund, Mrs. Carl Oscar (Selma Augusta Emilia…	female	38.0	1
26	Emir, Mr. Farred Chehab	male	28.0	0
27	Fortune, Mr. Charles Alexander	male	19.0	3
28	O’Dwyer, Miss. Ellen “Nellie”	female	28.0	0
29	Todoroff, Mr. Lalio	male	28.0	0
..	…	…	…	…

	Name	Sex	Age	SibSp \
861	Giles, Mr. Frederick Edward	male	21.0	1
862	Swift, Mrs. Frederick Joel (Margaret Welles Ba…	female	48.0	0
863	Sage, Miss. Dorothy Edith “Dolly”	female	28.0	8
864	Gill, Mr. John William	male	24.0	0
865	Bystrom, Mrs. (Karolina)	female	42.0	0
866	Duran y More, Miss. Asuncion	female	27.0	1
867	Roebling, Mr. Washington Augustus II	male	31.0	0
868	van Melkebeke, Mr. Philemon	male	28.0	0
869	Johnson, Master. Harold Theodor	male	4.0	1
870	Balkic, Mr. Cerin	male	26.0	0
871	Beckwith, Mrs. Richard Leonard (Sallie Monypeny)	female	47.0	1
872	Carlsson, Mr. Frans Olof	male	33.0	0
873	Vander Cruyssen, Mr. Victor	male	47.0	0
874	Abelson, Mrs. Samuel (Hannah Wizosky)	female	28.0	1
875	Najib, Miss. Adele Kiamie “Jane”	female	15.0	0
876	Gustafsson, Mr. Alfred Ossian	male	20.0	0
877	Petroff, Mr. Nedelio	male	19.0	0
878	Laleff, Mr. Kristo	male	28.0	0
879	Potter, Mrs. Thomas Jr (Lily Alexenia Wilson)	female	56.0	0
880	Shelley, Mrs. William (Imanita Parrish Hall)	female	25.0	0
881	Markun, Mr. Johann	male	33.0	0
882	Dahlberg, Miss. Gerda Ulrika	female	22.0	0
883	Banfield, Mr. Frederick James	male	28.0	0
884	Sutehall, Mr. Henry Jr	male	25.0	0
885	Rice, Mrs. William (Margaret Norton)	female	39.0	0
886	Montvila, Rev. Juozas	male	27.0	0
887	Graham, Miss. Margaret Edith	female	19.0	0
888	Johnston, Miss. Catherine Helen “Carrie”	female	28.0	1
889	Behr, Mr. Karl Howell	male	26.0	0
890	Dooley, Mr. Patrick	male	32.0	0

	Parch	Ticket	Fare	Cabin	Embarked
0	0	A/5 21171	7.2500	NaN	S
1	0	PC 17599	71.2833	C85	C
2	0	STON/O2. 3101282	7.9250	NaN	S
3	0	113803	53.1000	C123	S

	Parch	Ticket	Fare	Cabin	Embarked
4	0	373450	8.0500	NaN	S
5	0	330877	8.4583	NaN	Q
6	0	17463	51.8625	E46	S
7	1	349909	21.0750	NaN	S
8	2	347742	11.1333	NaN	S
9	0	237736	30.0708	NaN	C
10	1	PP 9549	16.7000	G6	S
11	0	113783	26.5500	C103	S
12	0	A/5. 2151	8.0500	NaN	S
13	5	347082	31.2750	NaN	S
14	0	350406	7.8542	NaN	S
15	0	248706	16.0000	NaN	S
16	1	382652	29.1250	NaN	Q
17	0	244373	13.0000	NaN	S
18	0	345763	18.0000	NaN	S
19	0	2649	7.2250	NaN	C
20	0	239865	26.0000	NaN	S
21	0	248698	13.0000	D56	S
22	0	330923	8.0292	NaN	Q
23	0	113788	35.5000	A6	S
24	1	349909	21.0750	NaN	S
25	5	347077	31.3875	NaN	S
26	0	2631	7.2250	NaN	C
27	2	19950	263.0000	C23 C25 C27	S
28	0	330959	7.8792	NaN	Q
29	0	349216	7.8958	NaN	S
..	…	…	…	…	…
861	0	28134	11.5000	NaN	S
862	0	17466	25.9292	D17	S
863	2	CA. 2343	69.5500	NaN	S
864	0	233866	13.0000	NaN	S
865	0	236852	13.0000	NaN	S
866	0	SC/PARIS 2149	13.8583	NaN	C
867	0	PC 17590	50.4958	A24	S
868	0	345777	9.5000	NaN	S
869	1	347742	11.1333	NaN	S
870	0	349248	7.8958	NaN	S
871	1	11751	52.5542	D35	S
872	0	695	5.0000	B51 B53 B55	S
873	0	345765	9.0000	NaN	S
874	0	P/PP 3381	24.0000	NaN	C

	Parch	Ticket	Fare	Cabin	Embarked
875	0	2667	7.2250	NaN	C
876	0	7534	9.8458	NaN	S
877	0	349212	7.8958	NaN	S
878	0	349217	7.8958	NaN	S
879	1	11767	83.1583	C50	C
880	1	230433	26.0000	NaN	S
881	0	349257	7.8958	NaN	S
882	0	7552	10.5167	NaN	S
883	0	C.A./SOTON 34068	10.5000	NaN	S
884	0	SOTON/OQ 392076	7.0500	NaN	S
885	5	382652	29.1250	NaN	Q
886	0	211536	13.0000	NaN	S
887	0	112053	30.0000	B42	S
888	2	W./C. 6607	23.4500	NaN	S
889	0	111369	30.0000	C148	C
890	0	370376	7.7500	NaN	Q

[891 rows x 12 columns]

If you read all of those hundreds of lines, bravo—but if you skipped ahead, I'm not surprised. I printed many rows of data here, instead of using a small subset, in order to illustrate what it feels like to be a data scientist. Working with columns of numbers feels value-neutral and occasionally tedious. There's a certain amount of dehumanization that occurs when you deal only with numbers. It's not easy to remember that each row in a dataset represents a real person with hopes, dreams, a family, and a history.

Now that we've looked at the raw data, it's time to start working with it. Let's turn it into *arrays*, which are structures that the computer can manipulate:

```
# Create the target and features numpy arrays: target,
features_one
target = train["Survived"].values

# Preprocess
encoded_sex = preprocessing.LabelEncoder()

# Convert into numbers
train.Sex = encoded_sex.fit_transform(train.Sex)
features_one = train[["Pclass," "Sex," "Age," "Fare"]].values
```

```
# Fit the first decision tree: my_tree_one
my_tree_one = tree.DecisionTreeClassifier()
my_tree_one = my_tree_one.fit(features_one, target)
```

What we're doing is running a function called *fit* on the decision tree classifier called *my_tree_one*. The features we want to consider are Pclass, Sex, Age, and Fare. We're instructing the algorithm to figure out what relationship among these four predicts the value in the target field, which is Survived:

```
# Look at the importance and score of the included features
print(my_tree_one.feature_importances_)
[ 0.12315342  0.31274009  0.22675108  0.3373554 ]
```

The feature_importances attribute shows the statistical significance of each predictor.

The largest number in this group of values is the considered the most important:

```
Pclass = 0.1269655
Sex = 0.31274009
Age = 0.23914906
Fare = 0.32114535
```

Fare is the largest number. We can conclude that passenger fare is the most important factor in determining whether a passenger survived the *Titanic* disaster.

At this point in our data analysis, we can run a function to show exactly how accurate our calculation is within the mathematical constraints of the universe represented by this data. Let's use the score function to find the mean accuracy:

```
print(my_tree_one.score(features_one, target))
0.977553310887
```

Wow, 97 percent! That feels great. If I got a 97 percent on an exam, I'd be perfectly content. We could call this model 97 percent accurate. The machine just "learned" in that it constructed a mathematical model. The model is stored in the object called *my_tree_one*.

Next, we'll take this model and apply it to the set of test data. Remember: the test data doesn't have a Survived column. Our job is to use the model to predict whether each passenger in the test data survived or perished. We know that fare is the most important predictor according to this model, but

age and sex and passenger class matter mathematically too. Let's apply the model to the test data and see what happens:

```
# Fill any missing fare values with the median fare
test["Fare"] = test["Fare"].fillna(test["Fare"].median())

# Fill any missing age values with the median age
test["Age"] = test["Age"].fillna(test["Age"].median())

# Preprocess
test_encoded_sex = preprocessing.LabelEncoder()
test.Sex = test_encoded_sex.fit_transform(test.Sex)

# Extract important features from the test set: Pclass, Sex,
Age, and Fare
test_features = test[["Pclass," "Sex," "Age," "Fare"]].values
print('These are the features:\n')
print(test_features)

# Make a prediction using the test set and print
my_prediction = my_tree_one.predict(test_features)
print('This is the prediction:\n')
print(my_prediction)

# Create a data frame with two columns: PassengerId & Survived
# Survived contains the model's prediction
PassengerId =np.array(test["PassengerId"]).astype(int)
my_solution = pd.DataFrame(my_prediction, PassengerId, columns
= ["Survived"])
print('This is the solution in toto:\n')
print(my_solution)

# Check that the data frame has 418 entries
print('This is the solution shape:\n')
print(my_solution.shape)

# Write the solution to a CSV file with the name my_solution.csv
my_solution.to_csv("my_solution_one.csv," index_label =
["PassengerId"])
```

Here's the output:

```
These are the features:
[[  3.       1.      34.5      7.8292]
 [  3.       0.      47.       7.    ]
 [  2.       1.      62.       9.6875] …,
 [  3.       1.      38.5      7.25  ]
 [  3.       1.      27.       8.05  ]
 [  3.       1.      27.      22.3583]]
```

```
This is the prediction:
[0 0 1 1 1 0 0 0 1 0 0 0 1 1 1 1 0 1 1 0 0 1 1 0 1 0 1 1 1 0 0 0 1 0 1 0 0
 0 0 1 0 1 0 1 1 0 0 0 1 1 1 0 1 1 1 0 0 0 1 1 0 0 0 1 0 0 1 0 0 1 1 0 0 0
 1 0 0 1 0 1 1 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 1 1 1 0 1 0 0 0 1 0 0 0 0 0 0
 0 1 1 1 0 1 1 0 1 1 0 1 0 0 1 0 1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 1 0
 1 0 1 0 0 1 0 0 1 1 0 1 1 1 1 1 0 1 1 0 0 0 0 1 0 1 0 1 1 0 1 1 0 0 1 0 1
 0 1 0 0 0 0 0 1 0 1 0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 1 1 0 1 0 0 1 0 1 0 1 0
 1 1 1 0 0 1 0 0 0 1 0 0 1 0 0 1 1 1 1 1 1 1 0 0 0 1 0 1 0 1 0 0 0 0 0 0 1
 0 0 0 1 1 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 1 1 0 1 0 0 0 1 0 1 0 1 0 0 0
 1 0 0 0 0 0 0 0 1 1 0 1 1 0 0 1 0 0 1 1 0 0 0 0 0 0 0 1 1 0 1 0 0 0 1 0 1
 1 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 1 1 0 0 0 1 0 1 0 0 1 0 1 1 1 1 0 0 0 1 0
 0 1 0 0 1 1 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 1 0 1 0 0 1 0 1 0 0 0 0
 0 1 1 1 1 0 0 1 0 0 0]
This is the solution in toto:
     Survived
892         0
893         0
894         1
895         1
896         1
897         0
898         0
899         0
900         1
901         0
902         0
903         0
904         1
905         1
906         1
907         1
908         0
909         1
910         1
911         0
912         0
913         1
914         1
915         0
916         1
917         0
918         1
919         1
```

```
920          1
921          0
…            …
1280         0
1281         0
1282         0
1283         1
1284         1
1285         0
1286         0
1287         1
1288         0
1289         1
1290         0
1291         0
1292         1
1293         0
1294         1
1295         0
1296         0
1297         0
1298         0
1299         0
1300         1
1301         1
1302         1
1303         1
1304         0
1305         0
1306         1
1307         0
1308         0
1309         0
[418 rows x 1 columns]
This is the solution shape:
(418, 1)
```

That new column, Survived, contains a prediction for each of the 418 passengers listed in the test data set. We can write the predictions to a CSV file called *my_solution_one.csv*, upload the file to DataCamp, and verify that our predictions were 97 percent accurate. Ta-da! We just did machine learning. It was entry level, but it was machine learning nonetheless. When someone says they have "used artificial intelligence to make a decision," usually they

mean "used machine learning," and usually they went through a process similar to the one we just worked through.

We created the Survived column and got a number that we can call 97 percent accurate. We learned that fare is the most influential factor in a mathematical analysis of *Titanic* survivor data. This was narrow artificial intelligence. It was not anything to be scared of, nor was it leading us toward a global takeover by superintelligent computers. "These are just statistical models, the same as those that Google uses to play board games or that your phone uses to make predictions about what word you're saying in order to transcribe your messages," Carnegie Mellon professor and machine learning researcher Zachary Lipton told the *Register* about AI. "They are no more sentient than a bowl of noodles."[17]

For a programmer, writing an algorithm is that easy. It gets made, it gets deployed, it seems to work. Nobody follows up. You maybe try turning the dials differently the next time to see if the accuracy seems to go up any. You try to get the highest number you can. Then, you move on to the next thing.

Meanwhile, out in the world, these numbers have consequences. It would be unwise to conclude from this data that people who pay more have a greater chance of surviving a maritime disaster. Nevertheless, a corporate executive could easily argue that it would be statistically legitimate to conclude this. If we were calculating insurance rates, we could say that people who pay higher ticket prices are less likely to die in iceberg accidents and thus represent a lower risk of early payout. People who pay more for tickets are wealthier than people who don't. This would allow us to charge rich people less for insurance. That's bad! The point of insurance is that risk is distributed evenly across a large pool of people. We've made more money for the insurance company, but we've not promoted the greatest good.

These types of computational techniques are used for *price optimization*, or grouping customers into very small segments to offer different prices to different groups. Price optimization is used in industries from insurance to travel—and it often results in price discrimination. A 2017 analysis by ProPublica and *Consumer Reports* found that in California, Illinois, Texas, and Missouri, some major insurers charged people who lived in minority neighborhoods as much as 30 percent more than people who lived in other areas with similar accident costs.[18] A 2014 analysis by the *Wall Street Journal* found that customers were being charged different prices for the same

ordinary stapler on Staples.com. The price was higher or lower based on the customer's estimated zip code.[19] Christo Wilson, David Lazer, and a team of other Northeastern University researchers found different prices were offered to customers on Homedepot.com and on travel sites depending on whether the users viewed the sites on mobile devices or desktops.[20] Amazon admitted to experimenting with differential pricing in 2000. CEO Jeff Bezos apologized, calling it "a mistake."[21]

In an unequal world, if we make pricing algorithms based on what the world looks like, women and poor and minority customers inevitably get charged more. Math people are often surprised by this; women and poor and minority people are not surprised by this. Race, gender, and class influence pricing in a variety of obvious and devious ways. Women are charged more than men for haircuts, dry cleaning, razors, and even deodorant. Asian-Americans are twice as likely to be charged more for SAT prep courses.[22] African American restaurant servers make less in tips than white colleagues.[23] Being poor often means paying more for necessities. Furniture on installment plans costs more than outright purchase. Payday loans have a far higher interest rate than bank loans. Housing is considered affordable if it takes 30 percent or less of a household's monthly income, but poor renters are often stuck paying more for housing because of a variety of factors related to economic instability. "In Milwaukee, the majority of poor renters devote at least half their income to rent, and a third pay at least 80 percent," sociologist Pat Sharkey writes in a review of two ethnographies, Matthew Desmond's *Evicted: Poverty and Profit in the American City* and Mitchell Dunier's *Ghetto: The Invention of a Place, the History of an Idea*.[24] Inequality is unfair, but it's not uncommon. If machine-learning models simply replicate the world as it is now, we won't move toward a more just society. "The allure of the technology is clear—the ancient aspiration to predict the future, tempered with a modern twist of statistical sobriety," law professor and AI ethics expert Frank Pasquale writes in *The Black Box Society.* "Yet in a climate of secrecy, bad information is as likely to endure as good, and to result in unfair and even disastrous predictions."[25]

Part of the reason we run into problems when making social decisions with machine learning is that the numbers camouflage important social context. In the *Titanic* example, we picked a classifier, survival. We used features to predict our classifier, but there are other possible factors. For example, our *Titanic* dataset includes only age, sex, and the other factors.

We built our predictor based on the information we had. However, because this was a human and not a mathematical event, there were other factors at work.

Let's look at the night of the *Titanic* disaster. The *Titanic* received multiple warnings of ice from nearby ships over the course of the day on April 14, 1912. At 11:40 p.m., the ship hit an iceberg. Just after midnight, the *Titanic*'s captain, Edward John Smith, mustered the passengers and began to evacuate the ship. Smith issued an order: "Put the women and children in and lower away." First Officer William Murdoch was in charge of the lifeboats on the starboard side. Second Officer Charles Lightoller was in charge of the boats on the port side. Each man interpreted the captain's command differently. Murdoch thought the captain meant women and children *first*. Lightoller thought the captain meant women and children *only*. Murdoch let men onto the boats if all the nearby women and children had been loaded. Lightoller loaded all the women and children nearby, then lowered the boat even if it had empty seats. Both men let the boats down into the water even if the full capacity of sixty-five people had not been reached. There were not enough lifeboats for the people on board: *Titanic* carried only twenty boats for a ship rated to carry 3,547 people. The best records show that the ship carried a light load of 892 crew members and 1,320 passengers.

There is a potentially interesting test to be done on lifeboat numbers. Murdoch's boats on the starboard side had odd numbers; Lightoller's boats had even numbers. Men probably survived at a different rate according to their lifeboat number, because Lightoller, who was in charge of even-numbered boats, didn't load men. However, the lifeboat number isn't in the data. This is a profound and insurmountable problem. Unless a factor is loaded into the model and represented in a manner a computer can calculate, it won't count. Not everything that counts is counted. The computer can't reach out and find out the extra information that might matter. A human can.

There's also the problem of false causality. If we did have the lifeboat numbers, from a computational perspective it might look like men in odd-numbered lifeboats had a better chance of surviving the *Titanic* disaster. If we made decisions based on data, we might decide that all lifeboats should be odd-numbered so that we could save more men in case of emergency. Of course, this is ridiculous; it was the officer, not the number of the boat, that made the difference.

Two young men also confound the pure mathematical explanation. Walter Lord's *A Night to Remember*, a bestselling nonfiction account of the *Titanic* disaster, is a moving account of the ship's last hours.[26] Lord tells the story of Jack Thayer, a seventeen-year-old who boarded the *Titanic* in Cherbourg, France, after a long European holiday with his parents. Thayer made a friend on the ship, Milton Long, another young man his age traveling in first class. As the crisis on the ship intensified, both young men helped to get other passengers to safety. By 2:00 a.m., almost all the lifeboats had launched, with Long and Thayer handing women and children into the lifeboats. By 2:15 a.m., the last lifeboats had washed away in the swells. The ship was listing to port. There was an explosion; a wave crashed over the boat deck. Chef John Collins was standing on the boat deck holding a baby, helping a steward and a woman from steerage who was traveling with two children. He and the others were swept out to sea. The baby was torn out of his arms by the force of the wave.

Thayer and Long saw the chaos on the decks. Suddenly, the lights winked out; the water had reached the fireboxes in boiler room two. The only light came from the moon and the stars and the lanterns on the lifeboats slowly rowing away from the sinking ship. The second funnel collapsed with a crash. Thayer and Long looked around: the lifeboats were gone and no rescue ship was in sight. They realized the moment had come to jump. They shook hands. They wished each other good luck. Lord writes:

> Long put his legs over the rail, while Thayer straddled it and began unbuttoning his overcoat. Long, hanging over the side and holding the rail with his hands, looked up at Thayer and asked, "You're coming, boy?"
>
> "Go ahead, I'll be right with you," Thayer reassured him.
>
> Long slid down, facing the ship. Ten seconds later Thayer swung his other leg over the rail and sat facing out. He was about ten feet above the water. Then with a push he jumped as far out as he could.
>
> Of these two techniques for abandoning ship, Thayer's was the only one that worked.

Thayer survived by swimming to a nearby overturned lifeboat and clinging to it with forty others. He watched as the Titanic cracked in half, the bow and stern slipping under the water amid a field of debris. Thayer heard people crying in the water. It sounded like locusts, he thought. Eventually, lifeboat twelve picked up Thayer and the others from the icy water. Help arrived hours later. Thayer shivered in the lifeboat until 8:30 the next morning, when the passengers were rescued by the *Carpathia*.

Thayer and Long were young men of the same age, same physical ability, same social status, and absolutely the same opportunity to survive the disaster. The difference came down to a jump. Thayer leaped out as far as he could away from the ship; Long dropped down next to the ship. Long was sucked into the abyss; Thayer wasn't. What I find unsettling is that whatever the computer predicts for Thayer or Long, it will be wrong. The prediction is based only on fare class, age, and sex—but what really happened was a difference of jumps. The computer just fundamentally misunderstands. Long's death, the randomness of it, is why our statistical prediction of who survived and who died on the Titanic will never be 100 percent accurate—no statistical prediction can or will ever be 100 percent accurate—because human beings are not and never will be statistics.

This speaks to a principle called the *unreasonable effectiveness of data*. Unless you are alert to the possibilities of discrimination and disarray, AI seems like it works beautifully. One of my favorite explanations of the search to explain the world through computer science comes from a paper by Google researchers Alon Halevy, Peter Norvig, and Fernando Pereira. They write:

> Eugene Wigner's article "The Unreasonable Effectiveness of Mathematics in the Natural Sciences" examines why so much of physics can be neatly explained with simple mathematical formulas such as f=ma or $e=mc^2$. Meanwhile, sciences that involve human beings rather than elementary particles have proven more resistant to elegant mathematics. Economists suffer from physics envy over their inability to neatly model human behavior. An informal, incomplete grammar of the English language runs over 1,700 pages. Perhaps when it comes to natural language processing and related fields, we're doomed to complex theories that will never have the elegance of physics equations. But if that's so, we should stop acting as if or goal is to author extremely elegant theories, and instead embrace complexity and make use of the best ally we have: the unreasonable effectiveness of data.[27]

Data is unreasonably effective—seductively so, even. This explains why we can build a classifier that seems to predict with 97 percent accuracy whether a passenger survives the *Titanic* disaster and why a computer can defeat a human Go champion. It also explains why, when we look closely at what happens during the machine-learning process, the machine doesn't take into account any of the flukes that humans know happen in real-life disaster situations. Data is very effective. However, the data-driven approach ignores a number of factors that humans think matter a great deal.

Law and society are set up to accommodate all of the things that humans think matter. Data-driven decisions rarely fit with these complex sets of rules. The same unreasonable effectiveness of data appears in translation, voice-controlled smart home gadgets, and handwriting recognition. Words and word combinations are not understood by machines the way that humans understand them. Instead, statistical methods for speech recognition and machine translation rely on vast databases full of short word sequences, or *n-grams*, and probabilities. Google has been working on these problems for decades and has the best scientific minds on these topics, and they have more data than anyone has ever before assembled. The Google Books corpus, the *New York Times* corpus, the corpus of everything everyone has ever searched for using Google: it turns out that when you load all of this in and assemble a massive database of how often words occur near each other, it's unreasonably effective. Let's take something simple. In n-grams, the word *boat* usually occurs near *water*, so the two are probably related. The probability is higher that *boat* is closer to *water* than to *electorate* or *stink bug*, so a search pulls up terms or documents related to boats and water rather than to boats and stink bugs. People generally talk about the same types of things and search for the same types of things, and common knowledge is really quite common. The machine is not really learning; the search process is just inspired by human learning. If you read the math, which is all posted online, it's very clear that these calculations are not magic and are just math. The computer will get enough things right enough of the time that we may be tempted to call it mostly correct—but it will get things right for exactly the wrong reasons.

Because social decisions are about more than just calculations, problems will always ensue if we use data alone to make decisions that involve social and value judgments. Traveling first class on the *Titanic* meant someone was more likely to survive—but it would be wrong to deploy a model that suggests first-class travelers *deserve* to survive disasters more than people who travel second or third class. Nor should we do other things that derive from a flawed model like the one we created. Our *Titanic* model could be used to justify charging first-class passengers less for travel insurance, but that's absurd: we shouldn't penalize people for not being rich enough to travel first class. Most of all, we should know by now that there are some things machines will never learn and that human judgment, reinforcement, and interpretation is always necessary.