# A Programmer's Guide to Data Mining

The Ancient Art of the Numerati

Ron Zacharski

**A Programmer's Guide to Data Mining: The Ancient Art of the Numerati**

www.guidetodatamining.com

by Ron Zacharski



Attribution information for all photographs is available on the website.

Thanks to ...

my wife Cheryl

my son Adam

Roper

Roz and Bodhi

also a huge thanks to all the photographers who put their work in the Creative Commons

# Preface

*If you continue this simple practice every day, you will obtain some wonderful power. Before you attain it, it is something wonderful, but after you attain it, it is nothing special.*

Shunryu Suzuki
Zen Mind, Beginner's Mind.

Before you work through this book you might think that systems like Pandora, Amazon's recommendations, and automatic data mining for terrorists, must be very complex and the math behind the algorithms must be extremely complex requiring a PhD to understand. You might think the people who work on developing these systems are like rocket scientists. One goal I have for this book is to pull back this curtain of complexity and show some of the rudimentary methods involved. Granted there are super-smart people at Google, the National Security Agency and elsewhere developing amazingly complex algorithms, but for the most part data mining relies on easy-to-understand principles. Before you start the book you might think data mining is pretty amazing stuff. By the end of the book, I hope you will be able to say nothing special.

The Japanese characters above, Shoshin, represent the concept of Beginner's Mind—the idea of having an open mind that is eager to explore possibilities. Most of us have heard some version of the following story (possibly from Bruce Lee's *Enter the Dragon*). A professor is seeking enlightenment and goes to a wise monk for spiritual direction. The professor dominates the discussion outlining everything he has learned in his life and summarizing papers he has written. The monk asks tea? and begins to pour tea into the professor's cup. And continues to pour, and continues to pour, until the tea over pours the teacup, the table, and spills onto the floor. *What are you doing?* the professor shouts. *Pouring tea* the monk says and continues: *Your mind is like this teacup. It is so filled with ideas that nothing else will go in. You must empty your mind before we can begin.*

To me, the best programmers are empty cups, who constantly explore new technology (noSQL, node-js, whatever) with open minds. Mediocre programmers have surrounded their minds with cities of delusion—C++ is good, Java is bad, PHP is the only way to do web programming, MySQL is the only database to consider. My hope is that you will find some of the ideas in this book valuable and I ask that you keep a beginner's mind when reading it. As Shunryu Suzuki says:

*In the beginner's mind there are many possibilities,*

*In the expert's mind there are few.*

Chapter 1 The Intro

# Intro to data mining & how to use this book

Imagine life in a small American town 150 years ago. Everyone knows one another. A crate of fabric arrives at the general store. The clerk notices that the pattern of a particular bolt would highly appeal to Mrs. Clancey because he knows that she likes bright floral patterns and makes a mental note to show it to her next time she comes to the store. Chow Winkler mentions to Mr. Wilson, the saloon keeper, that he is thinking of selling his spare Remington rifle. Mr. Wilson mentions that information to Bud Barclay, who he knows is looking for a quality rifle. Sheriff Valquez and his deputies know that Lee Pye is someone to keep an eye on as he likes to drink, has a short temper, and is strong. Life in a small town 100 years ago was all about connections.

People knew your likes and dislikes, your health, the state of your marriage. For better or worse, it was a personalized experience. And this highly personalized life in the community was true throughout most of the world.

Let's jump ahead one hundred years to the 1960s. Personalized interactions are less likely but they are still present. A regular coming into a local bookstore might be greeted with "The new James Michener is in"-- the clerk knowing that the regular loves James Michener books. Or the clerk might recommend to the regular The Conscience of a Conservative by Barry Goldwater, because the clerk knows the regular is a staunch conservative. A regular customer comes into a diner and the waitress says "The usual?"

Even today there are pockets of personalization.  I go to my local coffee shop in Mesilla and the barista says "A venti latte with an extra shot?" knowing that is what I get every morning. I take my standard poodle to the groomers and the groomer doesn't need to ask what style of clip I want. She knows I like the no frills sports clip with the German style ears.

But things have changed since the small towns of 100 years ago. Large grocery stores and big box stores replaced neighborhood grocers and other merchants  At the start of this change choices were limited. Henry Ford once said  "Any customer can have a car painted any color that he wants so long as it is black." The record store carried a limited number of records; the bookstore carried a limited number of books. Want ice cream? The choices were vanilla, chocolate, and maybe strawberry. Want a washing machine? In 1950 you had two choices at the local Sears: the standard model for $55 or the deluxe for $95.

# Welcome to the 21st century

In the 21st century those limited choices are a thing of the past. I want to buy some music? iTunes has some 11 million tracks to choose from. 11 million!  They have sold 16 billion tracks as of October 2011. I need more choices? I can go to Spotify which has over 15 million songs.

I want to buy a book? Amazon has over 2 million titles to chose from.

I want to watch a video? There are plenty of choices:


over 100,000 titles


nearly 50,000 titles


over 100,000 titles

I want to buy a laptop? When I type in *laptop* into the Amazon search box I get 3,811 results

I type in *rice cooker* and get over 1,000 possibilities.

In the near future there will be even more choice—billions of music tracks online—a wide variety of video—products that can be customized with 3D printing.

## Finding Relevant Stuff

The problem is finding relevant stuff. Amid all those 11 million tracks on iTunes, there are probably quite a number that I will absolutely love, but how do I find them. I want to watch a streaming movie from Netflix tonight, what should I watch. I want to download a movie using P2P, but which movie. And the problem is getting worse. Every minute terabytes of media are added to the net. Every minute 100 new files are available on usenet. Every minute 24 hours of video is uploaded to YouTube. Every hour 180 new books are published. Every day there are more and more options of stuff to buy in the real world. It gets more and more difficult to find the relevant stuff in this ocean of possibilities.

If you are a producer of media—say Zee Avi from Malaysia—the danger isn't someone downloading your music illegally—the danger is obscurity.


## But how to find stuff?

Years ago, in that small town, our **friends** helped us find stuff. That bolt of fabric that would be perfect for us; that new novel at the bookstore; that new 33 1/3 LP at the record store. Even today we rely on friends to help us find some relevant stuff.

We used **experts** to help us find stuff. Years ago Consumer Reports could evaluate all the washing machines sold—all 20 of them—or all the rice cookers sold-- all 10 of them and make recommendations. Today there are hundreds of different rice cookers available on Amazon and it is unlikely that a single expert source can rate all of them. Years ago, Roger Ebert would review virtually all the movies available. Today about 25,000 movies are made each year worldwide. Plus, we now have access to video from a variety of sources. Roger Ebert, or any single expert, cannot review all the movies that are available to us.

We also use **the thing itself** to help us find stuff. For example, I owned a Sears washing machine that lasted 30 years, I am going to buy another Sears washing machine. I liked one album by the Beatles—I will buy another thinking chances are good I will like that too.

These methods of finding relevant stuff—friends, experts, the thing itself—are still present today but we need some computational help to transform them into the 21st century where we have billions of choices.

These methods of finding relevant stuff—friends, experts, the thing itself—are still present today but we need some computational help to transform them into the 21st century where we have billions of choices. In this book we will explore methods of aggregating people's likes and dislikes, their purchasing history, and other data—exploiting the power of social net (friends)—to help us mine for relevant stuff. We will examine methods that use attributes of the thing itself. For example, I like the band Phoenix. The system might know attributes of Phoenix—that it uses electric rock instrumentation, has punk influences, has a subtle use of vocal harmony. It might recommend to me a similar band that has similar attributes, for example, The Strokes.

## It's just not stuff...

Data mining is just not about recommending stuff to us, or having merchants sell more stuff. Consider these examples.

The mayor of that small town of 100 years ago, knew everybody. When he ran for re-election he knew how to tailor what he said to each individual.

Martha, I know you are interested in schools and I will do everything in my power to bring another teacher to town.

John, how is your bakery doing? I promise to get more parking in your area of downtown.

My father belonged to the United Auto Workers' Union. Around election time I remember the union representative coming to our house to remind my father what candidates to vote for:

Frank Zeidler was the Socialist mayor of Milwaukee from 1948 to 1960.

*Hey Syl, how are the wife and kids? ... Now let me tell you why you should vote for Frank Zeidler, the Socialist candidate for mayor...*

This individualized political message changed to the homogenous ads during the rise of television. Everyone got the exact same message. A good example of this is the famous Daisy television ad in support of Lyndon Johnson ( a young girl pulling petals off a daisy while a nuclear bomb goes off in the background). Now, with elections determined by small margins and the growing use of data mining, individualization has returned. You are interested in a women's right to choose? You might get a robo-call directed at that very issue.

The sheriff of that small town knew who the trouble makers were. Now, threats seem to be hidden, terrorists can be anywhere. On October 11, 2001 the US government passed the USA Patriot Act (short for **U**niting and **S**trengthening **A**merica by **P**roviding **A**ppropriate **T**ools **R**equired to **I**ntercept and **O**bstruct **T**errorism). In part this bill enables investigators to obtain records for a variety of sources including libraries (what books we read), hotels (who stayed where and for how long), credit card companies, toll roads registering that we passed by. For the most part the government uses private companies to keep data on us. Companies like Seisint have data on almost all of us, photos of us, where we live, what we drive, our income, our buying behavior, our friends. Seisint owns supercomputers that use data mining techniques to make predictions about people. Their product by the way is called...

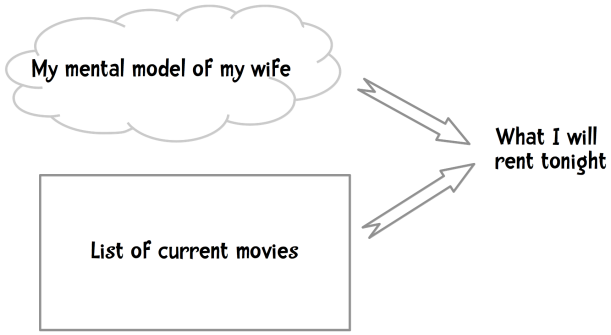
## The Matrix.

## Data Mining Extends what we already do!

Stephen Baker begins his book *The Numerati* this way:

> Imagine you are in a café, perhaps the noisy one I'm sitting in at this moment. A young women at a table to your right is typing on her laptop. You turn your head and look at her screen. She surfs the Internet. You watch.
>
> Hours pass. She reads an online paper. You notice that she reads three articles about China. She scouts movies for Friday night and watches the trailer for Kung Fu Panda. She clicks on an ad that promises to connect her to old high school classmates. You sit there taking notes. With each passing minute, you're learning more about her. Now imagine that you could watch 150 million people surfing at the same time.

Data mining is focused on finding patterns in data. At the small scale, we are expert at building mental models and finding patterns. I want to watch a movie tonight with my wife. I have a mental model of what she likes. I know she dislikes violent movies (she didn't like District 9 for that reason). She likes movies by Charlie Kaufman. I can use that mental model I have of her movie preferences to predict what movies she may or may not like.




A friend is visiting from Europe. I know she is a vegetarian and I can use that information to predict she would not like the local rib joint. People are good at making models and making predictions. Data mining expands this ability and enables us to handle large quantities of information—the 150 million people in the Baker quote above. It enables the Pandora Music Service to tailor a music station to your specific musical preferences. It enables Netflix to make specific personalized movie recommendations for you.

## Tera-mining is not something from Starcraft II

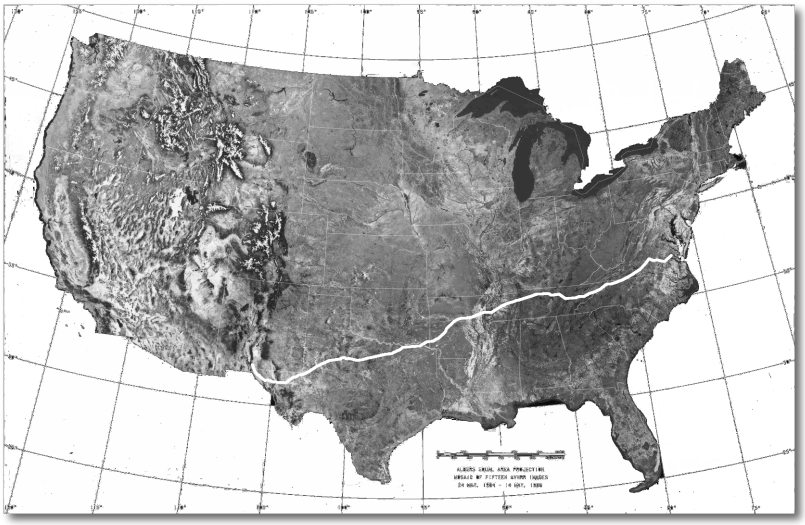
At the end of the 20th century a million word data set was considered large. When I was a graduate student in the 1990s (yes, I am that ancient) I worked as a programmer for a year on the Greek New Testament. It's only around 200,000 words but the analysis was too large to fit into the mainframe's memory necessitating spooling results off to magnetic tapes, which I had to request to be mounted.


The book resulting from this work is the Analytical Greek New Testament by Timothy and Barbara Friberg (available on Amazon). I was just one of three programmers on this project done at the University of Minnesota.

Today it is not unusual to be doing data mining on terabytes of information. Google has over 5 petabytes (that's 5,000 terabytes) of web data. In 2006 Google released a dataset to the research community based on one trillion words. The National Security Agency has call records for trillions of phone calls. Acxiom, a company that collects information (credit card purchases, telephone records, medical records, car registrations, etc) on 200 million adults in the US, has amassed over 1 petabyte of data.


a 1 petabyte server shipping container

Robert O'Harrow, Jr., author of No Place to Hide, in an effort to help us grasp how much information is 1 petabyte says it is the equivalent of 50,000 miles of stacked King James Bibles. I frequently drive 2,000 between New Mexico and Virginia. When I try to imagine bibles stacked along the entire way that seems like an unbelievable amount of data.



New Mexico to Virginia

The Library of Congress has around 20 terabytes of text. You could store the entire collection of the Library of Congress on a few thousand dollar's worth of hard drives! In contrast, Walmart has over 570 terabytes of data. All this data just doesn't sit there—it is constantly being mined, new associations made, patterns identified. Tera-mining.

Throughout this book we will be dealing with small datasets. It's good thing. We don't want our algorithm to run for a week only to discover we have some error in our logic. The biggest dataset we will use is under 100MB; the smallest just tens of lines of data.

# The format of the book.

This book follows a learn-by-doing approach. Instead of passively reading the book, I encourage you to work through the exercises and experiment with the Python code I provide. Experimenting around, code hacking, and trying out methods with different data sets is the key to really gaining an understanding for the techniques.

I try to strike a balance between hands-on, nuts-and-bolts discussion of Python data mining code that you can use and modify, and the theory behind the data mining techniques. To try to prevent the brain freeze associated with reading theory, math, and Python code, I tried to stimulate a different part of your brain by adding drawings and pictures.

Peter Norvig, Director of Research at Google, had this to say in his great Udacity course. *Design of a Computer Program*:

> "I'll show you and discuss my solution. It's important to note, there is more than one way to approach a problem. And I don't mean that my solution is the ONLY way or the BEST way. My solutions are there to help you learn a style and some techniques for programming. If you solve problems a different way, that's fine. Good for you.
>
> All the learning that goes on happens inside of your head. Not inside of my head. So what's important is that you understand the relation between your code and my code, that you get the right answer by writing out the solution yourself and then you can examine my code and maybe pick out some pointers and techniques that you can use later."

## I couldn't agree more!


This book is not a comprehensive textbook on data mining techniques. There are textbooks, like Introduction to Data Mining by Pang-Ning Tan, Michael Steinbach, and Vipin Kumar that provide significantly better coverage of data mining methods and provide more in-depth analysis of the mathematic underpinnings of these methods. This book—the one you are holding—is intended more as a quick, gritty, hands-on introduction designed to give you a basic foundation of data mining techniques. Later, you can pick up a more comprehensive book to fill in any gaps that you wish.

Part of the usefulness of this book is the accompanying Python code and the datasets. I think the inclusion of both these make it easier for the learner to understand key concepts, but at the same time, not shoe-horn the learner into a scripted exploration.

## What will you be able to do when you finish this book?

When you finish this book you will be able to design and implement recommendation systems for websites using Python or any language you know. For example, when you look at a product on Amazon, or a tune on Pandora, you are presented with a list of recommendations (You might also like ...). You will learn how to develop such systems. In addition, the book should provide you with the necessary vocabulary to enable you to work in development teams on data mining efforts.

As part of this goal, this book should help shed the mystery of recommendation systems, terrorist identification systems, and other data mining systems. You should at least have a rough idea of how they work.

## Why – why does this matter?

Why should you use your time reading (and working through) this book on data mining? At the beginning of this chapter I gave examples related to the importance of data mining. The summary of that section would go as follows. There's lots of stuff out there (movies, music, books, rice cookers). There's going to be a huge growth in the amount of stuff out there. The problem with having all this stuff available is finding the stuff that is relevant to us. Of all the movies out there, what movie should I watch. What's the next book I should read? This problem of identifying relevant stuff is what data mining is about. Most websites will have some component dealing with 'finding stuff'. In addition to the movies, music, books, and rice cookers mentioned above, you might want recommendations about what friends to follow. How about a personalized newspaper showing just the news you are most interested in? If you are a programmer, particularly a web developer, it would be useful to know data mining techniques.

Okay, so you can see the reason to devote some of your time to learning data mining, but why this book? There are books that give you a non-technical overview of data mining. They are a quick read, entertaining, inexpensive, and can be read late at night (no hairy technical bits). A great example of this is *The Numerati* by Stephen Baker. I recommend this book—I listened to the audio version of it while driving between Virginia and New Mexico. It was engrossing. On the other extreme are college textbooks on data mining. They are

comprehensive and provide an in-depth analysis of data mining theory and practice. Again, I recommend books in this category. I wrote this book to fill a gap. It's a book designed for people who love to program—hackers.


The book is intended to be read at a computer so the reader can participate and mess with code.

### Eeeks!

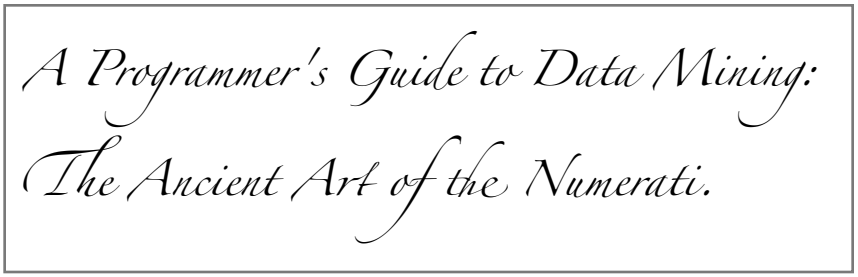
The book has math formulas but I try to explain them in a way that is intelligible to average programmers, who may have forgotten a hunk of the math they took in college.

$$s(i,j) = \frac{\sum_{u \in U}(R_{u,i} - \overline{R}_u)(R_{u,j} - \overline{R}_u)}{\sqrt{\sum_{u \in U}(R_{u,i} - \overline{R}_u)^2}\sqrt{\sum_{u \in U}(R_{u,j} - \overline{R}_u)^2}}$$

If that doesn't convince you, this book is also free (as in no cost) and free as in you can share it.

## What's with the 'Ancient Art of the Numerati' part of the title

In June of 2010 I was trying to come up with a title for this book. I like clever titles, but unfortunately, I have no talent in the area. I recently published a paper titled *Linguistic Dumpster Diving: Geographical Classification of Arabic Text* (yep, a data mining paper). I like the title and it is clever because it fits with the content of the paper, but I have to confess my wife came up with the title. I co-wrote a paper *Mood and Modality: Out of the theory and into the fray*. My co-author Marjorie McShane came up with the title. Anyway, back to June, 2010. All my clever title ideas were so vague that you wouldn't have a clue what the book was about. I finally settled on *A Programmer's Guide to Data Mining* as part of the title. I believe that bit is a concise description of the content of the book—I intend the book be a guide for the working programmer. You might wonder what is the meaning of the part after the colon:




The Numerati is a term coined by Stephen Baker. Each one of us generates an amazing amount of digital data everyday. credit card purchases, Twitter posts, Gowalla posts, Foursquare check-ins, cell phone calls, email messages, text messages, etc.

You get up. The Matrix knows you boarded the subway at the Foggy Bottom Station at 7:10 and departed the Westside Station at 7:32. The Matrix knows you got a venti latte and a blueberry scone at the Starbucks on 5th and Union at 7:45; you used Gowalla to check-in at work at 8:05; you made an Amazon purchase for the P90X Extreme Home Fitness Workout Program 13 DVD set and a chin-up bar at 9:35; you had lunch at the Golden Falafel.

Stephen Baker writes:

> The only folks who can make sense of the data we create are crack mathematicians, computer scientists, and engineers. What will these Numerati learn about us as they run us into dizzying combinations of numbers? First they need to find us. Say you're a potential SUV shopper in the northern suburbs of New York, or a churchgoing, antiabortion Democrat in Albuquerque. Maybe you're a Java programmer ready to relocate to Hyderabad, or a jazz-loving, Chianti-sipping Sagittarius looking for walks in the country and snuggles by the fireplace in Stockholm, or—heaven help us—maybe you're eager to strap bombs to your waist and climb onto a bus. Whatever you are—and each of us is a lot of things—companies and governments want to identify and locate you. Baker

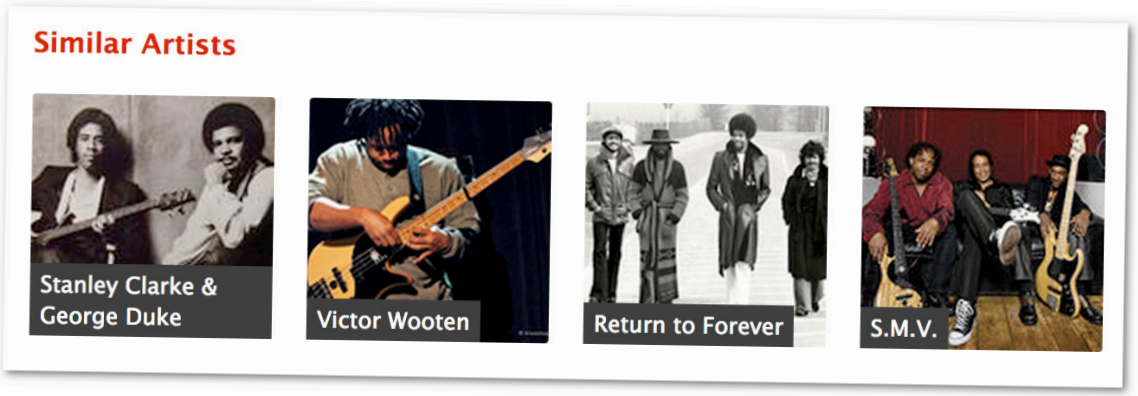
As you can probably guess, I like this term *Numerati* and Stephen Baker's description of it.

## Chapter 2: Collaborative Filtering

# I like what you like

We are going to start our exploration of data mining by looking at recommendation systems. Recommendation systems are everywhere—from Amazon:

to last.fm recommending music or concerts:



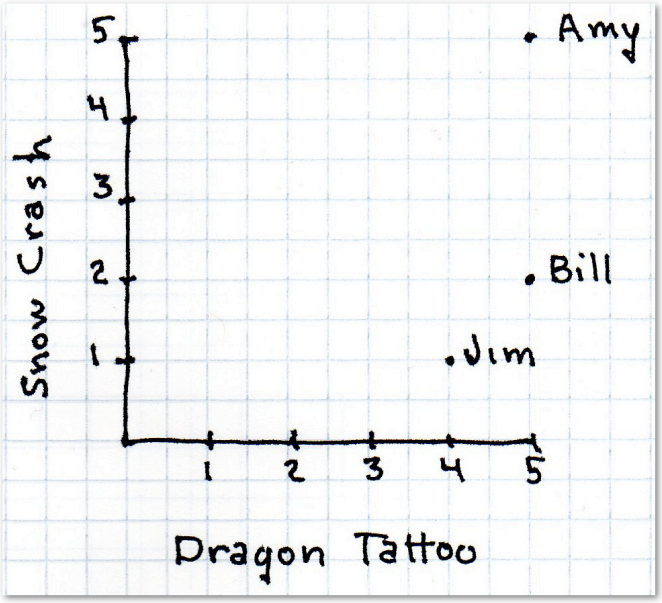


In the Amazon example, above, Amazon combines two bits of information to make a recommendation. The first is that I viewed *The Lotus Sutra* translated by Gene Reeves; the second, that customers who viewed that translation of the *Lotus Sutra* also viewed several other translations.

The recommendation method we are looking at in this chapter is called collaborative filtering. It's called collaborative because it makes recommendations based on other people—in effect, people collaborate to come up with recommendations. It works like this. Suppose the task is to recommend a book to you. I search among other users of the site to find one that is similar to you in the books she enjoys. Once I find that similar person I can see what she likes and recommend those books to you—perhaps Paolo Bacigalupi's *The Windup Girl*.

## How do I find someone who is similar?

So the first step is to find someone who is similar. Here's the simple 2D (dimensional) explanation. Suppose users rate books on a 5 star system—zero stars means the book is terrible, 5 stars means the book is great. Because I said we are looking at the simple 2D case, we restrict our ratings to two books: Neal Stephenson's *Snow Crash* and the Steig Larsson's *The Girl with the Dragon Tattoo*.




First, here's a table showing 3 users who rated these books

| | Snow Crash | Girl with the Dragon Tattoo |
|---|---|---|
| Amy | 5☆ | 5☆ |
| Bill | 2☆ | 5☆ |
| Jim | 1☆ | 4☆ |

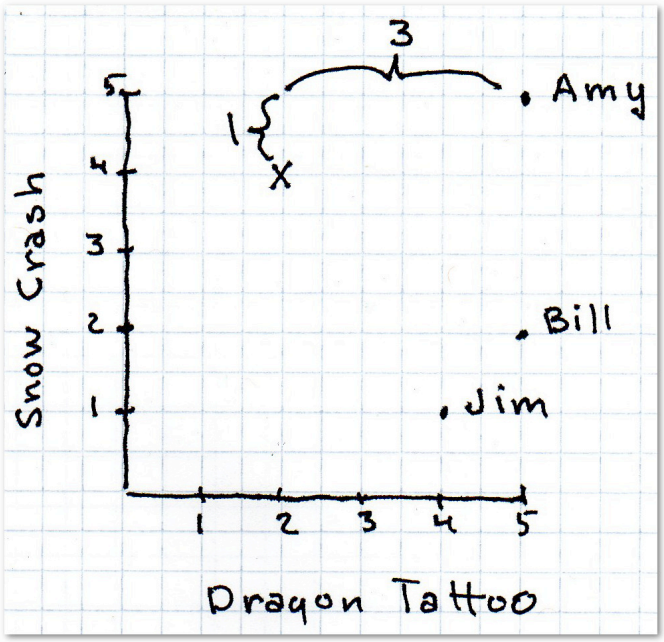
I would like to recommend a book to the mysterious Ms. X who rated *Snow Crash* 4 stars and *The Girl with the Dragon Tattoo* 2 stars. The first task is to find the person who is most similar, or closest, to Ms. X. I do this by computing distance.

# Manhattan Distance

The easiest distance measure to compute is what is called Manhattan Distance or cab driver distance. In the 2D case, each person is represented by an $(x, y)$ point. I will add a subscript to the $x$ and $y$ to refer to different people. So $(x_1, y_1)$ might be Amy and $(x_2, y_2)$ might be the elusive Ms. X. Manhattan Distance is then calculated by

$$| x_1 - x_2 | + | y_1 - y_2 |$$

(so the absolute value of the difference between the $x$ values plus the absolute value of the difference between the $y$ values). So the Manhattan Distance for Amy and Ms. X is 4:




Computing the distance between Ms. X and all three people gives us:

| | Distance from Ms. X |
|---|---|
| Amy | 4 |
| Bill | 5 |
| Jim | 5 |

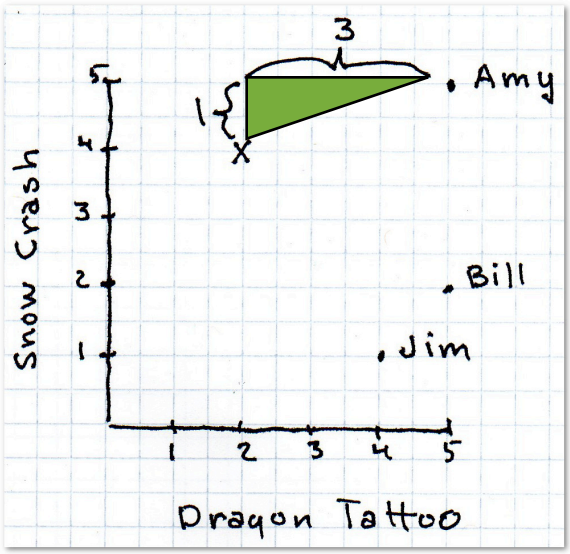
Amy is the closest match. We can look in her history and see, for example, that she gave five stars to Paolo Bacigalupi's *The Windup Girl* and we would recommend that book to Ms. X.

## Euclidean Distance

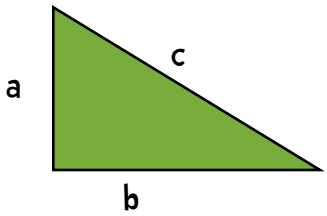
One benefit of Manhattan Distance is that it is fast to compute. If we are Facebook and are trying to find who among one million users is most similar to little Danny from Kalamazoo, fast is good.

### Pythagorean Theorem

You may recall the Pythagorean Theorem from your distant educational past. Here, instead of finding the Manhattan Distance between Amy and Ms. X (which was 4) we are going to figure out the straight line, as-the-crow-flies, distance



The Pythagorean Theorem tells us how to compute that distance.




$$c = \sqrt{a^2 + b^2}$$

This straight-line, as-the-crow-flies distance we are calling Euclidean Distance. The formula is

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Recall that $x_1$ is how well person 1 liked *Dragon Tattoo* and $x_2$ is how well person 2 liked it; $y_1$ is how well person 1 liked *Snow Crash* and $y_2$ is how well person 2 liked it.

Amy rated both *Snow Crash* and *Dragon Tattoo* a 5; The elusive Ms. X rated *Dragon Tattoo* a 2 and *Snow Crash* a 4. So the Euclidean distance between

$$\sqrt{(5-2)^2 + (5-4)^2} = \sqrt{3^2 + 1^2} = \sqrt{10} = 3.16$$

Computing the rest of the distances we get

| | Distance from Ms. X |
|---|---|
| Amy | 3.16 |
| Bill | 3.61 |
| Jim | 3.61 |

# N-dimensional thinking

Let's branch out slightly from just looking at rating two books (and hence 2D) to looking at something slightly more complex. Suppose we work for an online streaming music service and we want to make the experience more compelling by recommending bands. Let's say users can rate bands on a star system 1-5 stars and they can give half star ratings (for example, you can give a band 2.5 stars). The following chart shows 8 users and their ratings of eight bands.

| | Angelica | Bill | Chan | Dan | Hailey | Jordyn | Sam | Veronica |
|---|---|---|---|---|---|---|---|---|
| Blues Traveler | 3.5 | 2 | 5 | 3 | - | - | 5 | 3 |
| Broken Bells | 2 | 3.5 | 1 | 4 | 4 | 4.5 | 2 | - |
| Deadmau5 | - | 4 | 1 | 4.5 | 1 | 4 | - | - |
| Norah Jones | 4.5 | - | 3 | - | 4 | 5 | 3 | 5 |
| Phoenix | 5 | 2 | 5 | 3 | - | 5 | 5 | 4 |
| Slightly Stoopid | 1.5 | 3.5 | 1 | 4.5 | - | 4.5 | 4 | 2.5 |
| The Strokes | 2.5 | - | - | 4 | 4 | 4 | 5 | 3 |
| Vampire Weekend | 2 | 3 | - | 2 | 1 | 4 | - | - |

The hyphens in the table indicate that a user didn't rate that particular band. For now we are going to compute the distance based on the number of bands they both reviewed. So, for example, when computing the distance between Angelica and Bill, we will use the ratings for *Blues Traveler*, *Broken Bells*, *Phoenix*, *Slightly Stoopid*, and *Vampire Weekend*. So the Manhattan Distance would be:

| | Angelica | Bill | Difference |
|---|---|---|---|
| Blues Traveler | 3.5 | 2 | 1.5 |
| Broken Bells | 2 | 3.5 | 1.5 |
| Deadmau5 | - | 4 | |
| Norah Jones | 4.5 | - | |
| Phoenix | 5 | 2 | 3 |
| Slightly Stoopid | 1.5 | 3.5 | 2 |
| The Strokes | 2.5 | - | - |
| Vampire Weekend | 2 | 3 | 1 |
| Manhattan Distance: | | | 9 |

The Manhattan Distance row, the last row of the table, is simply the sum of the differences: (1.5 + 1.5 + 3 + 2 + 1).

Computing the Euclidean Distance is similar. We only use the bands they both reviewed:

| | Angelica | Bill | Difference | Difference$^2$ |
|---|---|---|---|---|
| Blues Traveler | 3.5 | 2 | 1.5 | 2.25 |
| Broken Bells | 2 | 3.5 | 1.5 | 2.25 |
| Deadmau5 | - | 4 | | |
| Norah Jones | 4.5 | - | | |
| Phoenix | 5 | 2 | 3 | 9 |
| Slightly Stoopid | 1.5 | 3.5 | 2 | 4 |
| The Strokes | 2.5 | - | - | |
| Vampire Weekend | 2 | 3 | 1 | 1 |
| Sum of squares | | | | 18.5 |
| Euclidean Distance | | | | 4.3 |

To parse that out a bit more:

$$Euclidean = \sqrt{(3.5-2)^2 + (2-3.5)^2 + (5-2)^2 + (1.5-3.5)^2 + (2-3)^2}$$

$$= \sqrt{1.5^2 + (-1.5)^2 + 3^2 + (-2)^2 + (-1)^2}$$

$$= \sqrt{2.25 + 2.25 + 9 + 4 + 1}$$

$$= \sqrt{18.5} = 4.3$$

## Got it?

Try an example on your own...

| | Angelica | Bill | Chan | Dan | Hailey | Jordyn | Sam | Veronica |
|---|---|---|---|---|---|---|---|---|
| Blues Traveler | 3.5 | 2 | 5 | 3 | - | - | 5 | 3 |
| Broken Bells | 2 | 3.5 | 1 | 4 | 4 | 4.5 | 2 | - |
| Deadmau5 | - | 4 | 1 | 4.5 | 1 | 4 | - | - |
| Norah Jones | 4.5 | - | 3 | - | 4 | 5 | 3 | 5 |
| Phoenix | 5 | 2 | 5 | 3 | - | 5 | 5 | 4 |
| Slightly Stoopid | 1.5 | 3.5 | 1 | 4.5 | - | 4.5 | 4 | 2.5 |
| The Strokes | 2.5 | - | - | 4 | 4 | 4 | 5 | 3 |
| Vampire Weekend | 2 | 3 | - | 2 | 1 | 4 | - | - |

## sharpen your pencil

Compute the Euclidean Distance between Hailey and Veronica.

Compute the Euclidean Distance between Hailey and Jordyn

## sharpen your pencil - solution

**Compute the Euclidean Distance between Hailey and Veronica.**

$$= \sqrt{(4-5)^2 + (4-3)^2} = \sqrt{1+1} = \sqrt{2} = 1.414$$

**Compute the Euclidean Distance between Hailey and Jordyn**

$$= \sqrt{(4-4.5)^2 + (1-4)^2 + (4-5)^2 + (4-4)^2 + (1-4)^2}$$

$$= \sqrt{(-0.5)^2 + (-3)^2 + (-1)^2 + (0)^2 + (-3)^2}$$

$$= \sqrt{.25 + 9 + 1 + 0 + 9} = \sqrt{19.25} = 4.387$$

## A Flaw

It looks like we discovered a flaw with using these distance measures. When we computed the distance between Hailey and Veronica, we noticed they only rated two bands in common (Norah Jones and The Strokes), whereas when we computed the distance between Hailey and Jordyn, we noticed they rated five bands in common. This seems to skew our distance measurement, since the Hailey-Veronica distance is in 2 dimensions while the Hailey-Jordyn

distance is in 5 dimensions. Manhattan Distance and Euclidean Distance work best when there are no missing values. Dealing with missing values is an active area of scholarly research. Later in the book we will talk about how to deal with this problem. For now just be aware of the flaw as we continue our first exploration into building a recommendation system.

## A generalization

We can generalize Manhattan Distance and Euclidean Distance to what is called the Minkowski Distance Metric:

$$d(x,y) = (\sum_{k=1}^{n} | x_k - y_k |^r)^{\frac{1}{r}}$$

When

- $r$ = 1: The formula is Manhattan Distance.
- $r$ = 2: The formula is Euclidean Distance
- $r$ = ∞: Supremum Distance

### Arghhhh Math!

When you see formulas like this in a book you have several options. One option is to see the formula--brain neurons fire that say *math formula*--and then you quickly skip over it to the next English bit. I have to admit that I was once a skipper. The other option is to see the formula, pause, and dissect it.

Many times you'll find the formula quite understandable. Let's dissect it now. When $r = 1$ the formula reduces to Manhattan Distance:

$$d(x,y) = \sum_{k=1}^{n} | x_k - y_k |$$

So for the music example we have been using throughout the chapter, x and y represent two people and $d(x, y)$ represents the distance between them. $n$ is the number of bands they both rated (both $x$ and $y$ rated that band). We've done that calculation a few pages back:

| | Angelica | Bill | Difference |
|---|---|---|---|
| Blues Traveler | 3.5 | 2 | 1.5 |
| Broken Bells | 2 | 3.5 | 1.5 |
| Deadmau5 | - | 4 | |
| Norah Jones | 4.5 | - | |
| Phoenix | 5 | 2 | 3 |
| Slightly Stoopid | 1.5 | 3.5 | 2 |
| The Strokes | 2.5 | - | - |
| Vampire Weekend | 2 | 3 | 1 |
| Manhattan Distance: | | | 9 |

That difference column represents the absolute value of the difference and we sum those up to get 9.

When r = 2, we get the Euclidean distance:

$$d(x,y) = \sqrt{\sum_{k=1}^{n} (x_k - y_k)^2}$$

Here's the scoop!

The greater the r, the more a large difference in one dimension will influence the total difference.

## Representing the data in Python (finally some coding)

There are several ways of representing the data in the table above using Python. I am going to use Python's dictionary (also called an associative array or hash table):

Remember,

All the code for the book is available at www.guidetodatamining.com.

```
users = {"Angelica": {"Blues Traveler": 3.5, "Broken Bells": 2.0,
                      "Norah Jones": 4.5, "Phoenix": 5.0,
                        "Slightly Stoopid": 1.5,
                        "The Strokes": 2.5, "Vampire Weekend": 2.0},

         "Bill":     {"Blues Traveler": 2.0, "Broken Bells": 3.5,
                      "Deadmau5": 4.0, "Phoenix": 2.0,
                      "Slightly Stoopid": 3.5, "Vampire Weekend": 3.0},

         "Chan":     {"Blues Traveler": 5.0, "Broken Bells": 1.0,
                      "Deadmau5": 1.0, "Norah Jones": 3.0,
                      "Phoenix": 5, "Slightly Stoopid": 1.0},

         "Dan":      {"Blues Traveler": 3.0, "Broken Bells": 4.0,
                      "Deadmau5": 4.5, "Phoenix": 3.0,
                      "Slightly Stoopid": 4.5, "The Strokes": 4.0,
                      "Vampire Weekend": 2.0},

         "Hailey":   {"Broken Bells": 4.0, "Deadmau5": 1.0,
                      "Norah Jones": 4.0, "The Strokes": 4.0,
                      "Vampire Weekend": 1.0},

         "Jordyn":   {"Broken Bells": 4.5, "Deadmau5": 4.0, "Norah Jones": 5.0,
                      "Phoenix": 5.0, "Slightly Stoopid": 4.5,
                      "The Strokes": 4.0, "Vampire Weekend": 4.0},

         "Sam":      {"Blues Traveler": 5.0, "Broken Bells": 2.0,
                      "Norah Jones": 3.0, "Phoenix": 5.0,
                      "Slightly Stoopid": 4.0,  "The Strokes": 5.0},

         "Veronica": {"Blues Traveler": 3.0, "Norah Jones": 5.0,
                      "Phoenix": 4.0,  "Slightly Stoopid": 2.5,
                      "The Strokes": 3.0}}
```

We can get the ratings of a particular user as follows:

```
>>> users["Veronica"]
{"Blues Traveler": 3.0, "Norah Jones": 5.0, "Phoenix": 4.0,
"Slightly Stoopid": 2.5, "The Strokes": 3.0}

>>>
```

## The code to compute Manhattan distance

I'd like to write a function that computes the Manhattan distance as follows:

```
def manhattan(rating1, rating2):
   """Computes the Manhattan distance. Both rating1 and rating2 are
   dictionaries of the form
   {'The Strokes': 3.0, 'Slightly Stoopid': 2.5 ..."""

   distance = 0
   for key in rating1:
      if key in rating2:
         distance += abs(rating1[key] - rating2[key])
   return distance
```

To test the function:

```
>>> manhattan(users['Hailey'], users['Veronica'])

2.0
>>> manhattan(users['Hailey'], users['Jordyn'])
7.5
>>>
```

Now a function to find the closest person (actually this returns a sorted list with the closest person first):

```
def computeNearestNeighbor(username, users):
    """creates a sorted list of users based on their distance to
    username"""
    distances = []
    for user in users:
        if user != username:
            distance = manhattan(users[user], users[username])
            distances.append((distance, user))
    # sort based on distance -- closest first
    distances.sort()
    return distances
```

And just a quick test of that function:

```
>>> computeNearestNeighbor("Hailey", users)
[(2.0, ''Veronica'), (4.0, 'Chan'),(4.0, 'Sam'), (4.5, 'Dan'), (5.0,
'Angelica'), (5.5, 'Bill'), (7.5, 'Jordyn')]
```

Finally, we are going to put this all together to make recommendations. Let's say I want to make recommendations for Hailey. I find her nearest neighbor—Veronica in this case. I will then find bands that Veronica has rated but Hailey has not. Also, I will assume that Hailey would have rated the bands the same as (or at least very similar to) Veronica. For example, Hailey has not rated the great band Phoenix. Veronica has rated Phoenix a '4' so we will assume Hailey is likely to enjoy the band as well. Here is my function to make recommendations.

```
def recommend(username, users):
    """Give list of recommendations"""
    # first find nearest neighbor
    nearest = computeNearestNeighbor(username, users)[0][1]
    recommendations = []
    # now find bands neighbor rated that user didn't
    neighborRatings = users[nearest]
    userRatings = users[username]
    for artist in neighborRatings:
        if not artist in userRatings:
            recommendations.append((artist, neighborRatings[artist]))
    # using the fn sorted for variety - sort is more efficient
    return sorted(recommendations,
                  key=lambda artistTuple: artistTuple[1],
                  reverse = True)
```

And now to make recommendations for Hailey:

```
>>> recommend('Hailey', users)
[('Phoenix', 4.0), ('Blues Traveler', 3.0), ('Slightly Stoopid', 2.5)]
```

That fits with our expectations. As we saw above, Hailey's nearest neighbor was Veronica and Veronica gave Phoenix a '4'. Let's try a few more:

```
>>> recommend('Chan', users)
```

```
[('The Strokes', 4.0), ('Vampire Weekend', 1.0)]

>>> recommend('Sam', users)
[('Deadmau5', 1.0)]
```

We think Chan will like The Strokes and also predict that Sam will not like Deadmau5.

```
>>> recommend('Angelica', users)
[]
```

Hmm. For Angelica we got back an empty set meaning we have no recommendations for her. Let us see what went wrong:

```
>>> computeNearestNeighbor('Angelica', users)
[(3.5, 'Veronica'), (4.5, 'Chan'), (5.0, 'Hailey'), (8.0, 'Sam'), (9.0,
'Bill'), (9.0, 'Dan'), (9.5, 'Jordyn')]
```

Angelica's nearest neighbor is Veronica. When we look at their ratings:

| | Angelica | Bill | Chan | Dan | Hailey | Jordyn | Sam | Veronica |
|---|---|---|---|---|---|---|---|---|
| Blues Traveler | 3.5 | 2 | 5 | 3 | - | - | 5 | 3 |
| Broken Bells | 2 | 3.5 | 1 | 4 | 4 | 4.5 | 2 | - |
| Deadmau5 | - | 4 | 1 | 4.5 | 1 | 4 | - | - |
| Norah Jones | 4.5 | - | 3 | - | 4 | 5 | 3 | 5 |
| Phoenix | 5 | 2 | 5 | 3 | - | 5 | 5 | 4 |
| Slightly Stoopid | 1.5 | 3.5 | 1 | 4.5 | - | 4.5 | 4 | 2.5 |
| The Strokes | 2.5 | - | - | 4 | 4 | 4 | 5 | 3 |
| Vampire Weekend | 2 | 3 | - | 2 | 1 | 4 | - | - |

We see that Angelica rated every band that Veronica did. We have no new ratings, so no recommendations.

Shortly, we will see how to improve the system to avoid these cases.

## exercise

1) Implement the Minkowski Distance function.

2) Alter the computeNearestNeighbor function to use Minkowski Distance.

## exercise - solution

### 1) Implement the Minkowski Distance function.

```
def minkowski(rating1, rating2, r):
    """Computes the Minkowski distance.
    Both rating1 and rating2 are dictionaries of the form
    {'The Strokes': 3.0, 'Slightly Stoopid': 2.5}"""
    distance = 0
    commonRatings = False
    for key in rating1:
        if key in rating2:
            distance +=
                 pow(abs(rating1[key] - rating2[key]), r)
            commonRatings = True
    if commonRatings:
        return pow(distance,  1/r)
    else:
        return 0 #Indicates no ratings in common
```

### 2) Alter the computeNearestNeighbor function to use Minkowski Distance.

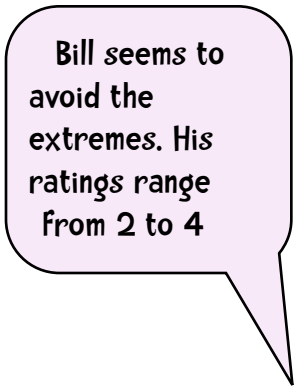
just need to alter the distance = line to

```
distance = minkowski(users[user], users[username], 2)
```

(the 2 as the r argument implements Euclidean)

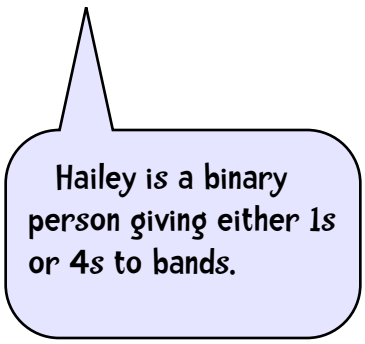
# Blame the users

Let's take a look at the user ratings in a bit more detail. We see that users have very different behaviors when it comes to rating bands





| | Angelica | Bill | Chan | Dan | Hailey | Jordyn | Sam | Veronica |
|---|---|---|---|---|---|---|---|---|
| Blues Traveler | 3.5 | 2 | 5 | 3 | - | - | 5 | 3 |
| Broken Bells | 2 | 3.5 | 1 | 4 | 4 | 4.5 | 2 | - |
| Deadmau5 | - | 4 | 1 | 4.5 | 1 | 4 | - | - |
| Norah Jones | 4.5 | - | 3 | - | 4 | 5 | 3 | 5 |
| Phoenix | 5 | 2 | 5 | 3 | - | 5 | 5 | 4 |
| Slightly Stoopid | 1.5 | 3.5 | 1 | 4.5 | - | 4.5 | 4 | 2.5 |
| The Strokes | 2.5 | - | - | 4 | 4 | 4 | 5 | 3 |
| Vampire Weekend | 2 | 3 | - | 2 | 1 | 4 | - | - |



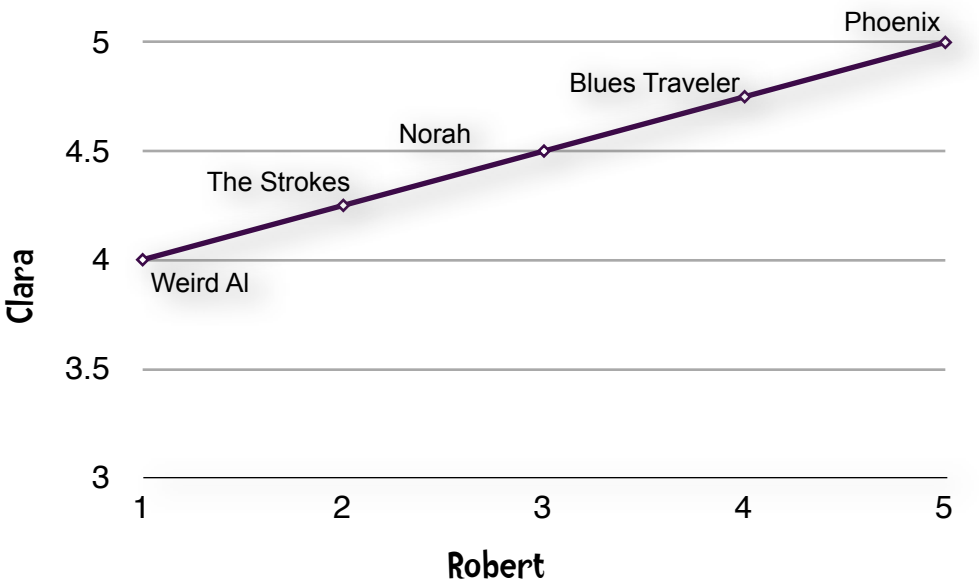
So how do we compare, for example, Hailey to Jordan? Does Hailey's '4' mean the same as Jordyn's '4' or Jordyn's '5'? I would guess it is more like Jordyn's '5'. This variability can create problems with a recommendation system.

# Pearson Correlation Coefficient

One way to fix this problem is to use the Pearson Correlation Coefficient. First, the general idea. Consider the following data (not from the data set above):

| | Blues Traveler | Norah Jones | Phoenix | The Strokes | Weird Al |
|---|---|---|---|---|---|
| Clara | 4.75 | 4.5 | 5 | 4.25 | 4 |
| Robert | 4 | 3 | 5 | 2 | 1 |

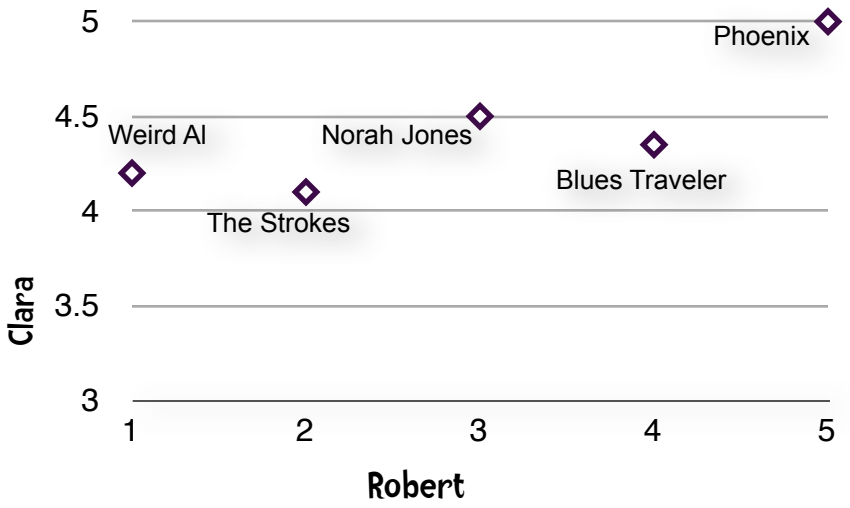
This is an example of what is called 'grade inflation' in the data mining community. Clara's lowest rating is 4—all her rating are between 4 and 5. If we are to graph this chart it would look like



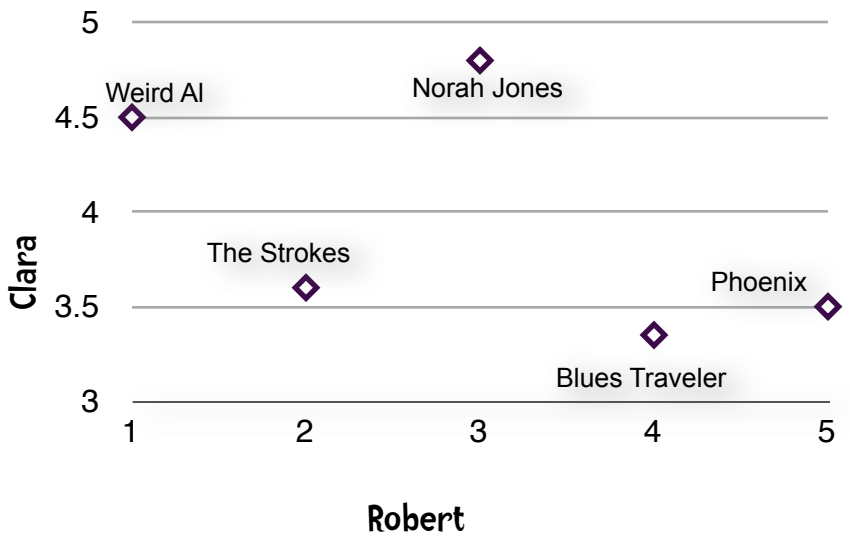

Straight line = Perfect Agreement!!!

The fact that this is a straight line indicates a perfect agreement between Clara and Robert. Both rated Phoenix as the best band, Blues Traveler next, Norah Jones after that, and so on. As Clara and Robert agree less, the less the data points reside on a straight line:

**Pretty Good Agreement:**




**Not So Good Agreement:**



So chart-wise, perfect agreement is indicated by a straight line. The Pearson Correlation Coefficient is a measure of correlation between two variables (in this specific case the correlation between Angelica and Bill). It ranges between -1 and 1 inclusive. 1 indicates perfect agreement. -1 indicates perfect disagreement. To give you a general feel for this, the chart above with the straight line has a Pearson of 1, the chart above that I labelled 'pretty good agreement' has a Pearson of 0.91, and the 'not so good agreement' chart has a Pearson of 0.81 So we can use this to find the individual who is most similar to the person we are interested in.

The formula for the Pearson Correlation Coefficient is

$$r = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \overline{y})^2}}$$

## Arghhhh Math Again!

Here's a personal confession. I have a Bachelor of Fine Arts degree in music. While I have taken courses in ballet, modern dance, and costume design, I did not have a single math course as an undergrad. Before that, I attended an all boys trade high school where I took courses in plumbing and automobile repair, but no courses in math other than the basics. Either due to this background or some innate wiring in my brain, when I read a book that has formulas like the one above, I tend to skip over the formulas and continue with the text below them. If you are like me I would urge you to fight

that urge and actually look at the formula. Many formulas that on a quick glimpse look complex are actually understandable by mere mortals.

Other than perhaps looking complex, the problem with the formula above is that the algorithm to implement it would require multiple passes through the data. Fortunately for us algorithmic people, there is an alternative formula, which is an approximation of Pearson:

$$r = \frac{\sum_{i=1}^{n} x_i y_i - \frac{\sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n}}{\sqrt{\sum_{i=1}^{n} x_i^2 - \frac{(\sum_{i=1}^{n} x_i)^2}{n}} \sqrt{\sum_{i=1}^{n} y_i^2 - \frac{(\sum_{i=1}^{n} y_i)^2}{n}}}$$

(Remember what I said two paragraphs above about not skipping over formulas) This formula, in addition to looking initially horribly complex is, more importantly, numerically unstable meaning that what might be a small error is amplified by this reformulation. The big plus is that we can implement it using a single-pass algorithm, which we will get to shortly. First, let's dissect this formula and work through the example we saw a few pages back:

| | Blues Traveler | Norah Jones | Phoenix | The Strokes | Weird Al |
|---|---|---|---|---|---|
| Clara | 4.75 | 4.5 | 5 | 4.25 | 4 |
| Robert | 4 | 3 | 5 | 2 | 1 |

To start with, let us compute

$$\sum_{i=1}^{n} x_i y_i$$

Which is in the first expression in the numerator. Here the x and y represent Clara and Robert.

| | Blues Traveler | Norah Jones | Phoenix | The Strokes | Weird Al |
|---|---|---|---|---|---|
| Clara | 4.75 | 4.5 | 5 | 4.25 | 4 |
| Robert | 4 | 3 | 5 | 2 | 1 |

For each band we are going to multiple Clara's and Robert's rating together and sum the results:

$$(4.75 \times 4) + (4.5 \times 3) + (5 \times 5) + (4.25 \times 2) + (4 \times 1)$$

$$= 19 + 13.5 + 25 + 8.5 + 4 = 70$$

Sweet! Now let's compute the rest of the numerator:

$$\frac{\sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n}$$

So the

$$\sum_{i=1}^{n} x_i$$

is the sum of Clara's ratings, which is 22.5. The sum of Robert's is 15 and they rated 5 bands:

$$\frac{22.5 \times 15}{5} = 67.5$$

So the numerator in the formula on page 26 is 70 - 67.5 = 2.5

Now let's dissect the denominator.

$$\sqrt{\sum_{i=1}^{n} x_i^2 - \frac{(\sum_{i=1}^{n} x_i)^2}{n}}$$

First,

| | Blues Traveler | Norah Jones | Phoenix | The Strokes | Weird Al |
|---|---|---|---|---|---|
| Clara | 4.75 | 4.5 | 5 | 4.25 | 4 |
| Robert | 4 | 3 | 5 | 2 | 1 |

$$\sum_{i=1}^{n} x_i^2 = (4.75)^2 + (4.5)^2 + (5)^2 + (4.25)^2 + (4)^2 = 101.875$$

We've already computed the sum of Clara's ratings, which is 22.5. Square that and we get 506.25. We divide that by the number of co-rated bands (5) and we get 101.25.

Putting that together:

$$\sqrt{101.875 - 101.25} = \sqrt{.625} = .79057$$

Next we do the same computation for Robert:

$$\sqrt{\sum_{i=1}^{n} y_i^2 - \frac{(\sum_{i=1}^{n} y_i)^2}{n}} = \sqrt{55 - 45} = 3.162277$$

Putting this altogether we get:

$$r = \frac{2.5}{.79057(3.162277)} = \frac{2.5}{2.5} = 1.00$$

So 1 means there was perfect agreement between Clara and Robert!

## Take a break before moving on!!

## exercise

Before going to the next page, implement the algorithm in Python. You should get the following results.

```
>>> pearson(users['Angelica'], users['Bill'])
-0.90405349906826993
>>> pearson(users['Angelica'], users['Hailey'])
0.42008402520840293
>>> pearson(users['Angelica'], users['Jordyn'])
0.76397486054754316
>>>
```

For this implementation you will need 2 Python functions sqrt (square root) and power operator ** which raises its left argument to the power of its right argument:

```
>>> from math import sqrt
>>> sqrt(9)
3.0
>>> 3**2
9
```

## exercise - solution

Here is my implementation of Pearson

```
def pearson(rating1, rating2):
    sum_xy = 0
    sum_x = 0
    sum_y = 0
    sum_x2 = 0
    sum_y2 = 0
    n = 0
    for key in rating1:
        if key in rating2:
            n += 1
            x = rating1[key]
            y = rating2[key]
            sum_xy += x * y
            sum_x += x
            sum_y += y
            sum_x2 += x**2
            sum_y2 += y**2
    # if no ratings in common return 0
    if n == 0:
        return 0
    # now compute denominator
    denominator = sqrt(sum_x2 - (sum_x**2) / n) *
                  sqrt(sum_y2 - (sum_y**2) / n)
    if denominator == 0:
        return 0
    else:
        return (sum_xy - (sum_x * sum_y) / n) / denominator
```