Deeper into Data Science

- I taught the last course. Here's what we covered.
- In this course, we'll be covering:
-- Taking a data science solution from notebook to production. Included is a brief example.
-- Machine learning overview. What it is? What it isn't? Why is it the new sexy only now, when these models have been around for 75 years? What are the principal classes of machine learning models? What are some canonical use cases for each? We're going to build a model to predict which comments on a social network are childish insults, and which contain eloquent literary criticism (or something like that).
- Statistics overview. What is the goal of statistics? What are probability distributions? How do we perform basic statistical inference?
-- In this section, we'll walk through a toy example of Bayesian inference.
- Tools we need as Data Scientists. The answer to this question is typically also the answer to: "Why aren't PhD graduates immediately employable as Data Scientists? What skills are they lacking?"
- How to build a Data Science team.
- By the end of this course, you'll have a strong idea of:
-- how to take a Data Science solution from inception to production, and the tools we use therein.
-- how to use machine learning and statistics to make your data work for

Productionizing Your Things

We spoke in the last course about the triangle: this is the third side.
Productionizing your things means taking a solution, building an engineering system out of it, forgetting about it, and moving on to the next problem. This system might:
- On a nightly basis, fit a new model to predict merchant churn, make your churn predictions, then write the results to a field in Salesforce for our Customer Care team to use.
- On a weekly basis, compute your Discover Weekly playlist on Spotify, and serve those predictions to your device.
- On a monthly basis, compute recommended products on Amazon, compose an email containing these products, and send these emails to users.

Databases

SQL vs. NoSQL

Data needs to live somewhere. This is a database.
- Structured vs. unstructured.
- SQL example: logging airplane takeoffs. Every takeoff has: an airport code, a time, an airline, a weather condition, a terminal, etc.
- NoSQL database types include: document databses, graph databases, and key-value stores.
- NoSQL example: a post on Facebook. it might include a check-in, likes, and comments. the commenters might be friends of yours, or perhaps friends of your friends. the post might contain a photo. the comment might contain a link. every post is different, and trying to store this data in a standardized way is a poor solution. for this reason, we might use a NoSQL store.

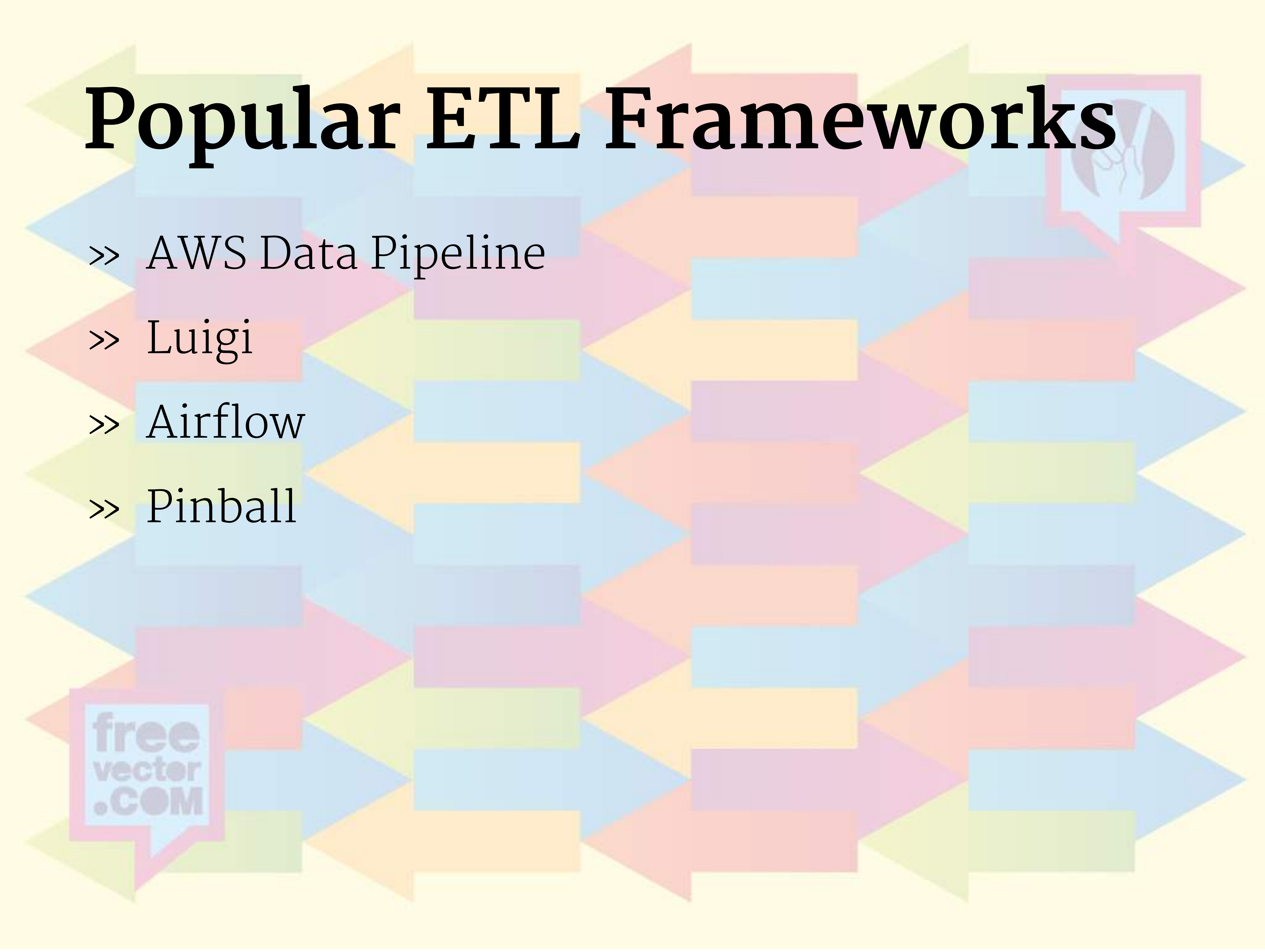
Popular Data Stores

- SQL: SQLite, MySQL, PostgreSQL, Redshift
- NoSQL: MongoDB, Redis, Giraph, Neo4j

Choosing one. Key considerations are:
- What types of queries will you be performing? Asking your database for large, flat chunks of data? Doing aggregations (SUM, MIN, MAX, etc.) at query time? Writing more than reading?
- In choosing a database: consult blogs, documentation, and others in your company and industry. It's quite case-specific. However, for many machine learning tasks, at least for writing the predictions to a data store, we're most likely working with structured data, and therefore a SQL store.

ETL

- Data comes in in raw formats. For example:
-- The votes that took place at a given voting booth in New York City
-- Mobile app usage data from AWS Mobile Analytics
-- Daily air quality statistics in Mexico City
- ETL is the process of taking that raw data, cleaning/re-arranging it, and storing it how you wish.
- For example, assume the the City of New York gives us, with each record, the name of the voter's mother, father, and pet turtle. Basically, we don't care. So, during ETL, we simply take the pieces we do care about, and write them to our database.
- Another example: text cleaning. Assume we're ingesting tweets, and we want to normalize these tweets before writing them to our database. So, during ETL, we lowercase the text, remove diacritics, remove numbers, and remove punctuation, and then store the cleaned tweet.
- How much do we do on the job?
-- Most companies will probably have 10 - 1000 pipelines. As Data Scientists, this is our raw data, and given the reality of teams in 2016, we end up writing a lot of these.
-- Don't worry, these are good: they engender strong backend engineering fundamentals: interacting with/writing API's, unit tests, databases, etc.



Popular ETL Frameworks

- AWS Data Pipeline
- Luigi
- Airflow
- Pinball

Luigi, by Spotify; Airflow, by AirBnB; Pinball, by Pinterest
- What goes into choosing a framework?
-- Services you need to interact with (S3, Redshift, etc.)
-- How often are you writing these things? How much configuration goes into launching one?
-- What languages do they support?
-- What's the community like behind each one?

Dashboards

Self-explanatory. Here's a few good reasons to use dashboards:
^- Internal business intelligence.
^-- Give your colleagues a way to ask and answer their own questions about company data. This promotes a "data-literate" culture, or at least one in which everyone believes that data is a thing we care about.
^- Monitoring. Have one place to look when monitoring server uptime, Customer Care availability, etc.
^- Reducing cognitive load. Pictures are pretty and make things easy. Simply eliminating the need to write a SQL query yourself is a small win.
^- Making the company look cool. Hang them around the office.
^-- The less external load on your Data Science team, the better. "Write me a SQL query" is bad. Certain dashboard technologies can alleviate this need.

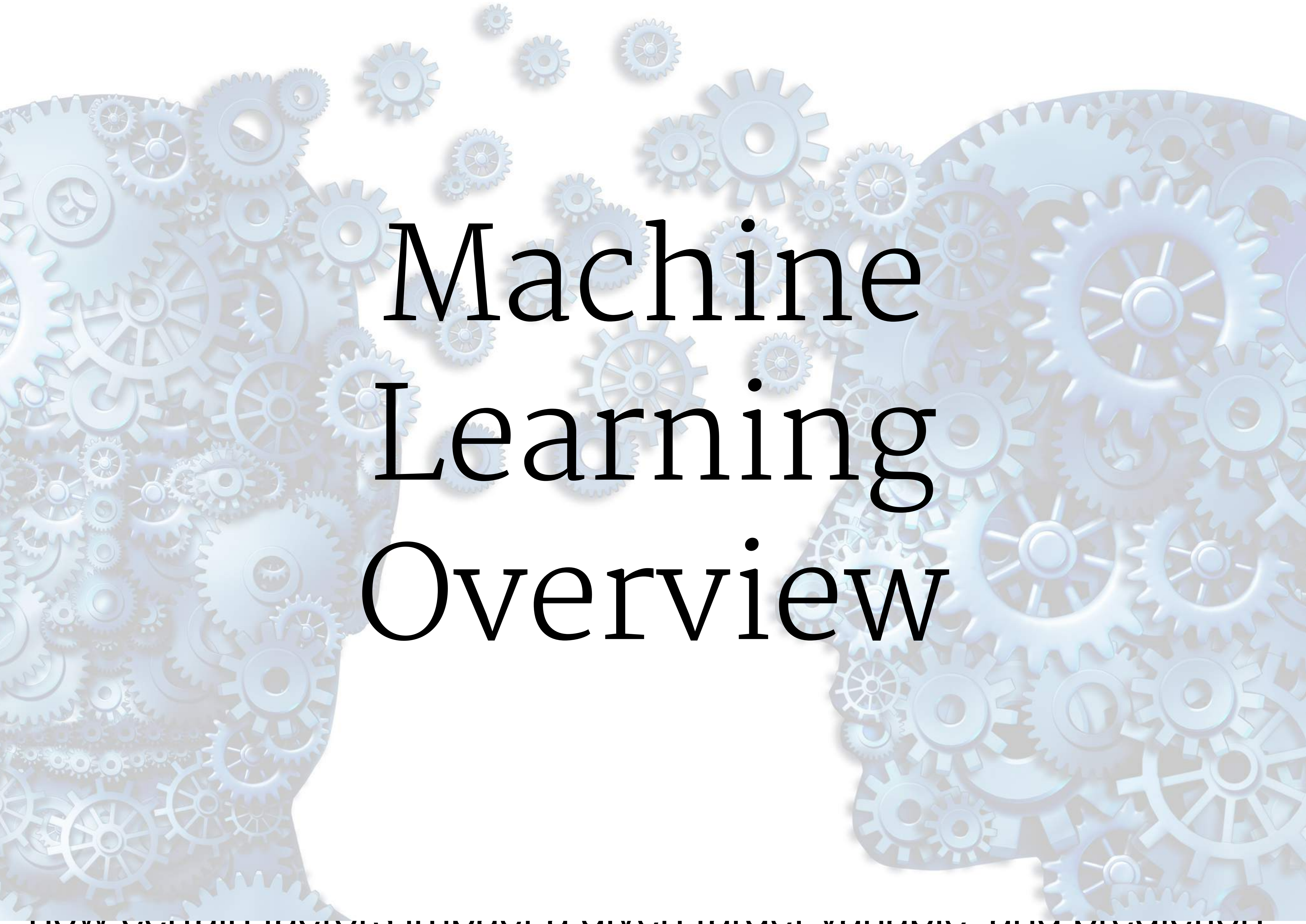
Business Pulse
Refresh All
LookML
Date
× Last 90 Days
State / Region
Total Orders Last 90 Days
35,557
Average Order Profit Last 90 Days
$52.05
First Purchasers Last 90 Days
13,203
Top Zip Codes Last 90 Days
Sales by Date and Category

I'll Speak about Looker

Model data, add business logic
^Whatever you do, probably don't build your own

Twitter Example -> Production

Machine Learning Overview



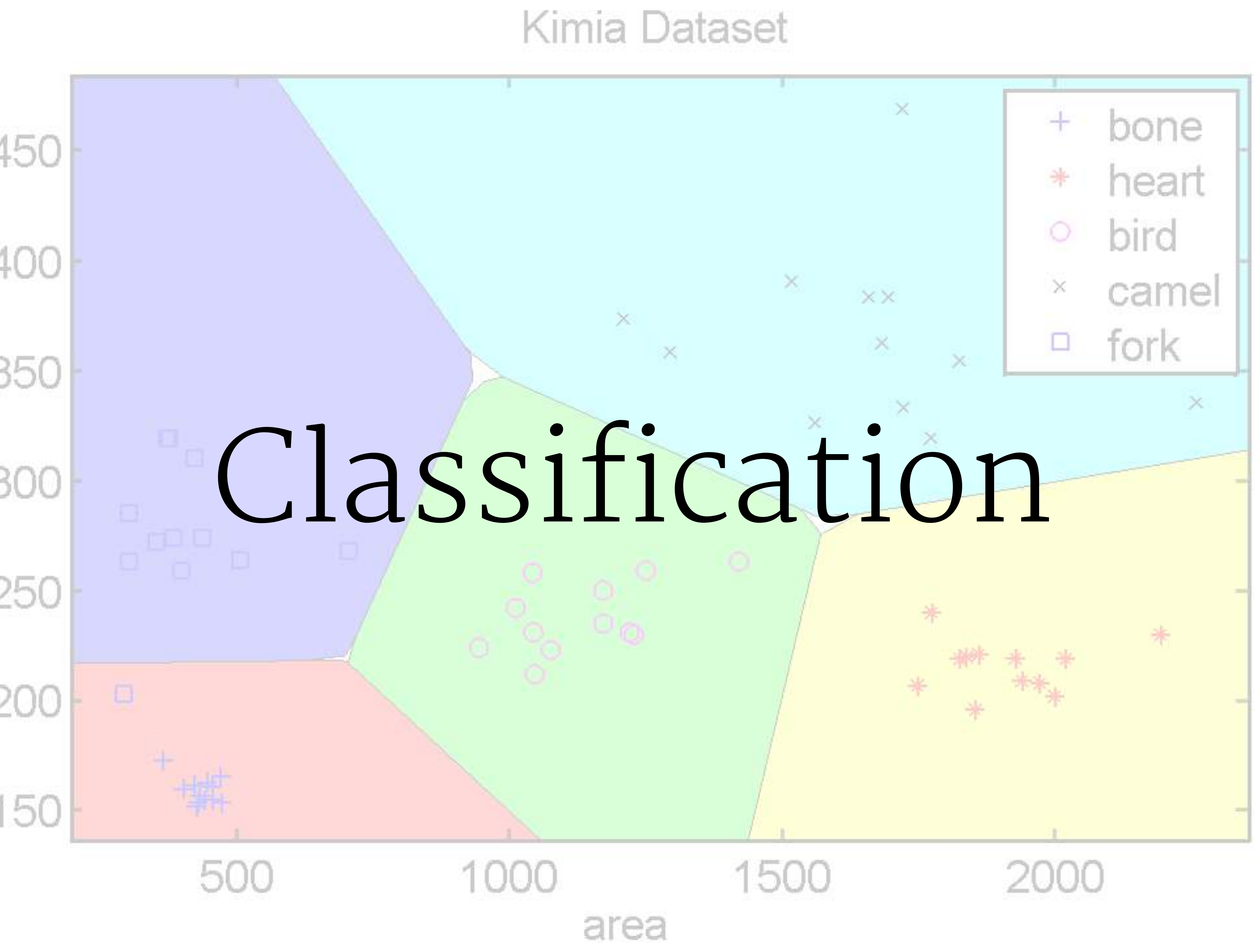
how certain factors impact a given target variable, and prediction, i.e. given some data, predict a label or a number
^How do ML models differ from traditional statistical models?
^- Reference Breiman "Two Cultures" paper.
^- ML models purportedly learn structure from our data; we do not presuppose that our data fit some statistical trend
^Why is machine learning a thing now, when these algorithms are not really new?
^On building machine learning models:
^- It's not a panacea: there's a fair bit of human guidance in the process
^- The better we understand our problem, the better the model
^- The better we understand the model, the better the engineering system we build to send it to production
^- ML is moving to a place of commoditization: we will soon use ML algorithms like we use sorting algorithms. To make our lives easier, yet without understanding what's going on under the hood.
^-- E.g. Hey Photos, give me all the pictures of my tent. We won't [illegible]

"Training" Machine Learning Models

Supervised vs. Unsupervised

Regression

Models that predict a continuous value.
^- The number of minutes the Transmilenio will be delayed at a given stop on a given day
^- The number of voters that a Minneapolis voting booth will need to accommodate
^- The number of inches of rainfall the Atacama Desert will receive next year
^Supervised learning algorithm.



Classification

Models that predict a label.
^- Who will win: Trump or Hillary?
^- Which friend's face appears in the photo you just uploaded to Facebook?
^- Does a video posted to Twitter contain inappropriate content (yes/no)?
^Supervised learning algorithm.

Clustering

A set of algorithms that group similar data points together. Clustering is often used as an exploratory step in data analysis.
^- Restaurant reviews on Yelp: how many different classes of reviews might we have? Some about food, others about ambiance, others about hygiene?
^- Clustering Spotify songs encoded as audio frequency data: how many different types of songs do we roughly have? Upbeat song? Slow songs? Songs with bass? Etc.
^For clustering algorithms, we make a distinction between true clustering algorithms, which don't necessarily put every point into a given cluster, and partitioning algorithms, that do.
^Clustering algorithms can offer a valid avenue for prediction, although these predictions typically aren't very good. This is for two reasons: we never had any hard labels, and cluster groups themselves can evolve.
^Unsupervised learning algorithm.

Dimensionality Reduction

Reduce a set of vectors of some high dimensionality to a lower dimensionality, typically to two ends:
^- Visualization: visualizing a dataset of, say, 1,000,000 dimensions in 2 dimensions (in a way that representatively captures how the data is spread in the original space). We do this because humans can't visualize anything in 1,000,000 dimensions.
^- Lower computational cost: in a high-dimensional space, there is often a lot of "redundancy" in our data. For example, if we are encoding web pages into a bag-of-words model using a web-wide vocabulary, there will probably be high correlation between values in the columns for "santa" and "claus." Dimensionality reduction algorithms can effectively remove this redundancy; in this way, they are in fact similar to data compression algorithms.
^-- If we're training a model on these documents, training in the reduced space, say, 1,000 dimensions, is certainly an improvement over the original space, which could be, say, in 300,000,000 dimensions.
^Unsupervised learning algorithm.

Canonical Models

As ML practitioners, we become familiar with what types of models work best with what types of problems. Work best means: computational cost, the types of relationships they derive, and the accuracy of prediction. This takes practice. Predictive modeling competitions, like those on Kaggle, are likely the very best way to obtain this practice.

Linear Regression

Linear regression models should be used in cases where we believe there exists a linear relationship between each of the input variables and our target.
^One of the simplest models. Calling it a "machine learning" model is more a point of fashion than anything else. I don't believe it had this name when it was invented.
^Example: modeling the relationship between centimeters of precipitation, hours of sunlight, and daily bike ridership.
^In situations where we try out a lot of models, this should likely be used as a baseline. It's not that powerful. Better models should perform better. If not, something else is probably wrong.

Logistic Regression

Part of the family of "generalized linear models." However, instead of our data varying linearly with a continuous value, it varies linearly with the log-odds of the probability of a binary event.
^Statistics aside, this model is the canonical model for binary classification. It is quick to fit even with lots of parameters, and offers a very intuitive interpretation of its predictions in the form of probabilities.
^E.g. Say we are building a service to connect developers with organizations building tools for refugees. To recommend organizations, we might ingest data about each developer, and create a binary prediction that they would be a good fit for each organization. Then, we'd rank these predictions in descending order, creating an ordered list.

Support Vector Machines

An algorithm with a long history, also used for binary classification.
^Finds the hyperplane between a set of points of two distinct classes which maximally separates them.
^If we visualize our data in 2 or 3 dimensions and observe a linear separation, then this is the algorithm for us.
^Can also be used for non-linear classification via the kernel trick.

Random Forest

Our first real non-linear algorithm in this list.
^Intrinsically uncovers interaction effects, i.e. "how does the *interaction* of two variables impact our target," by virtue of being a decision tree.
^Fits a bunch of bad decision trees, and averages the vote of each.
^Much slower to fit than the above linear models. This said, they are easily parallelize-able
^This is Breiman's algorithm. This is what he had in mind (I think!) in the "Two Cultures" paper.

Neural Networks

The hotness, of course.
^"Deep Learning" is simply deep neural networks, i.e. networks with lots of hidden layers
^Used to model extremely complex non-linear relationships between input and output. What this means is: the input is something like a video clip, encoded into pixel values and temporal elements, and the output is something like a simple label. The relationship between the two is probably crazy. Neural nets do a good job at learning these relationships, or, in other words, functions.
^When our input is: video, audio or photo, we are well-served by neural networks.
^Rather difficult to train. Lots of input parameters.
^When we're training on tons of data, the engineering challenge is extremely non-trivial as well.

Turning Things into Vectors

[1] Towards Anything2Vec

- I once read in a blog post1 that a lot of machine learning is just "turning things into vectors," i.e. a list of numbers. For example, an ML model can't "understand" a photo; instead, we turn this photo into a list of pixel values and pass it into our algorithm. This can loosely be called encoding the thing into numbers, or turning into a vector.
^- Each value in this vector is a "feature." A "feature" is simply a number that describes one aspect of each example.
^- When we train a model, we have a training set. The rows represent the patients, and the columns represent their features. Each row is this vector we've been discussing. A dataset sized 100 x 3 includes 100 patients, each with 3 features. Aside, we have a vector of labels, effectively sized 100 x 1: 1 label for each of 100 patients.
^- The 100x3 matrix is our X. The 100x1 vector is our y. A learning algorithm needs an X and a y.
^- Sometimes, our features are straightforward. Other times, it's a bit more artful.
^-- Give bag-of-words example.

Everything is a Hyper-Parameter

When building machine learning models, "everything is a hyper-parameter." In other words, everything is a knob to turn. For example: the features you create, the models you choose, the hyper-parameters with which you parameterize those models and the way in which you ensemble these models are all a thing you can change.
^In this way, fitting models is a matter of practice, experience, and efficient experimentation.
^This likely breaks down when you have Google-scale data. Then, pragmatism in terms of both time and money are your biggest hurdles, and you likely need true academics guiding the process.

Error metrics

- Things that quantify how bad it is to be wrong.
- Most models come with classic/typical/ advised error metrics in tow.
- You can implement your own error metric if you like.
- Example: we're building an algorithm to predict flight delay to inform us as to when to leave the house.

Classification

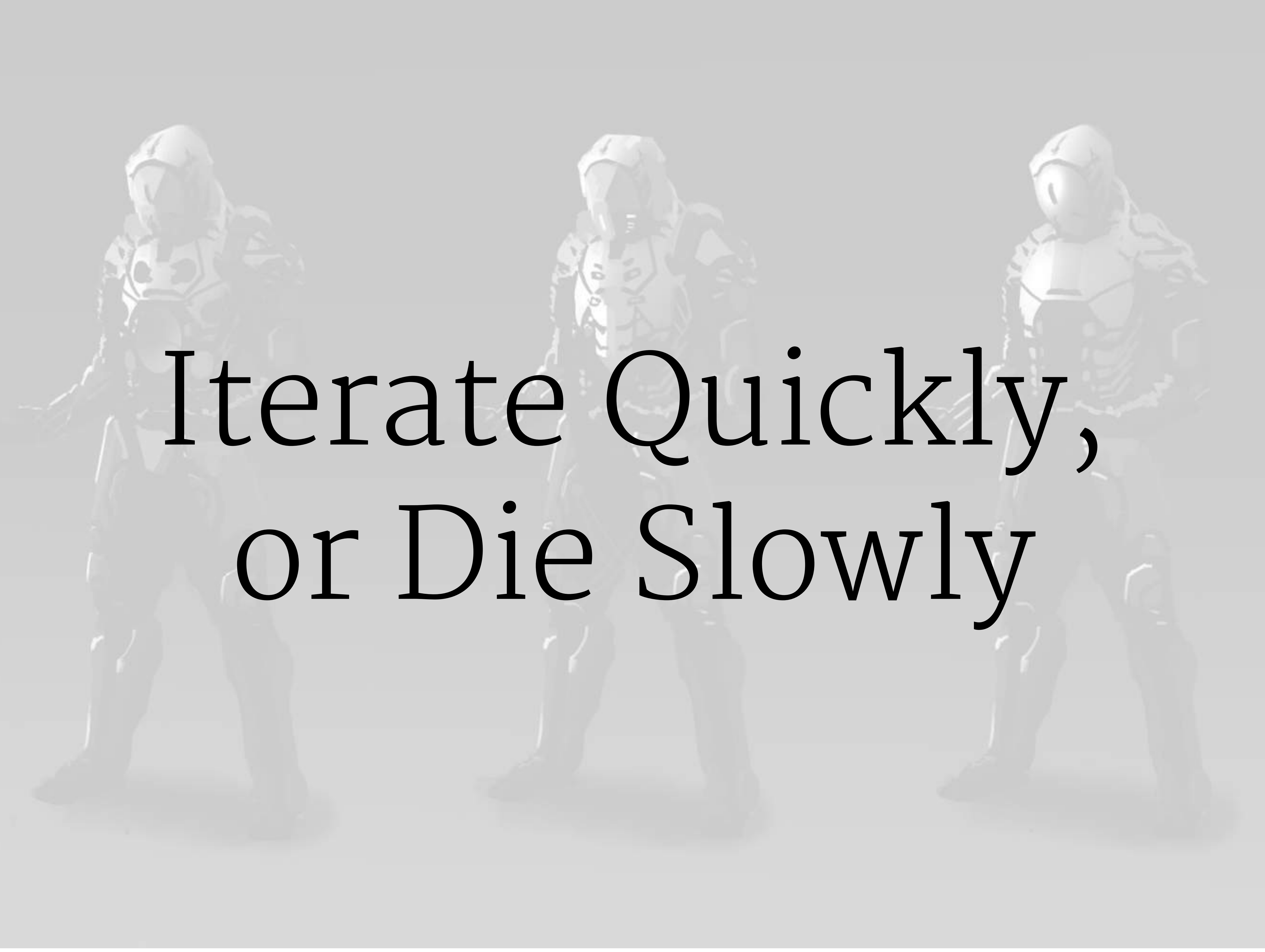
F1-Score, Logistic Loss, Accuracy, AUC ROC

Regression

Mean Squared Error, Mean Average Error, Mean Absolute Error

Cross Validation

Cancer example. We can't deploy that model without validating it.
^The goal: get a realistic view of how our model will perform on unseen data, without actually releasing it into the wild.
^Repeat the above a few times. The CV process needs to mimic the prediction scenario.
^Simplest case: split your data into 2 parts.
^K-folds cross-validation.
^This is an extremely important part of machine learning. Predicting on real data is extremely costly in many senses of the word.



Iterate Quickly, or Die Slowly

It is crucial to set up your workflow to allow for quick iteration. Having to refactor code for 20 minutes every time you want to try a new model is the antithesis of this statement. Spend time upfront making your workflow smooth.

Code, though?

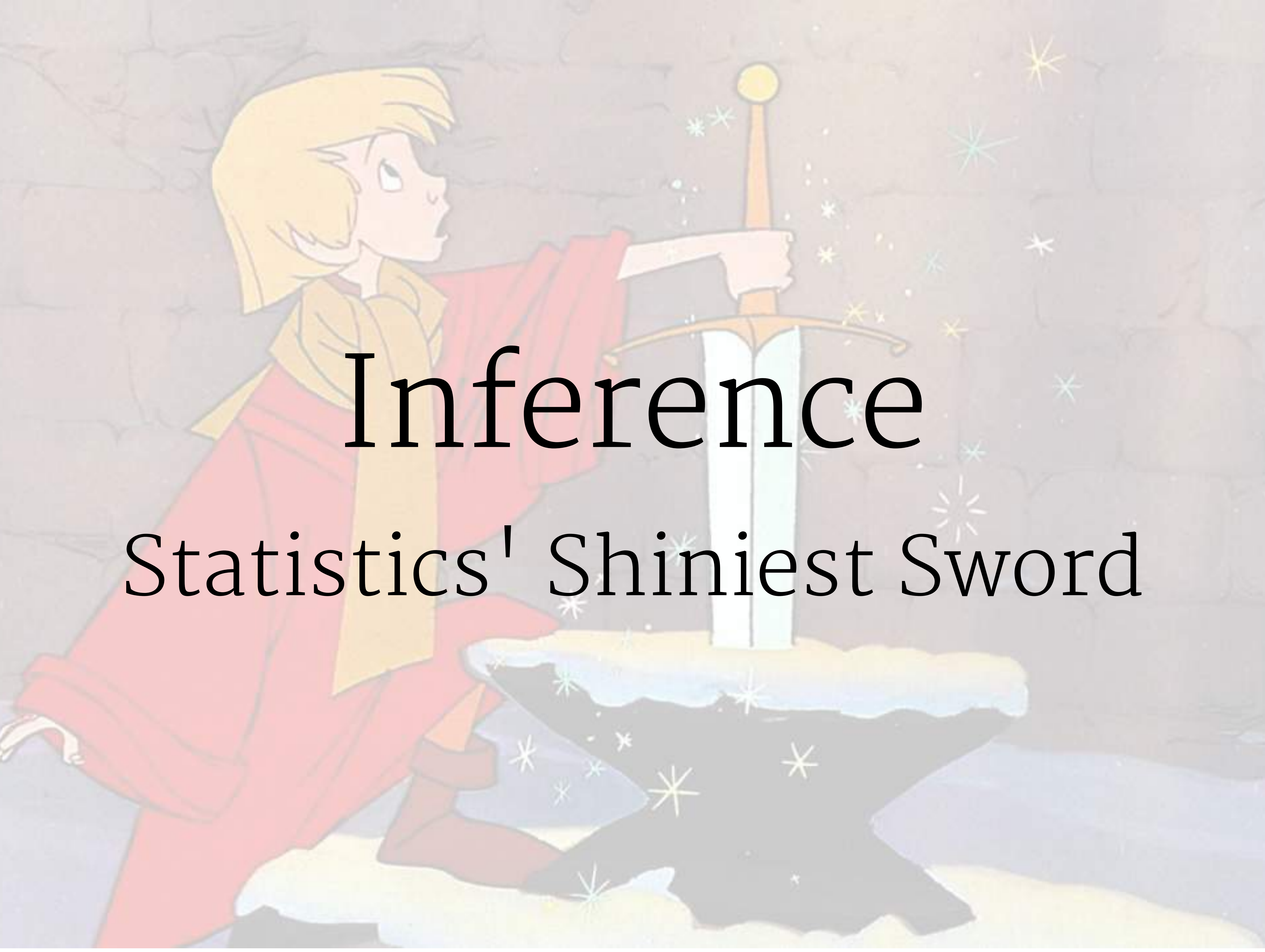
Right. We need to code. Scikit-learn. R. Spark. Probably don't build things yourself unless you have to. ^That said, building algorithms by hand is an amazing way to learn.

Insults Example

What a stupid thing to say. I feel much dumber after reading your comment.
What a brilliant idea. Can you explain further?
Miserable. What in the world were you thinking?
Perhaps we could try this a different way? I don't quite agree with your methodology.

Statistics Overview

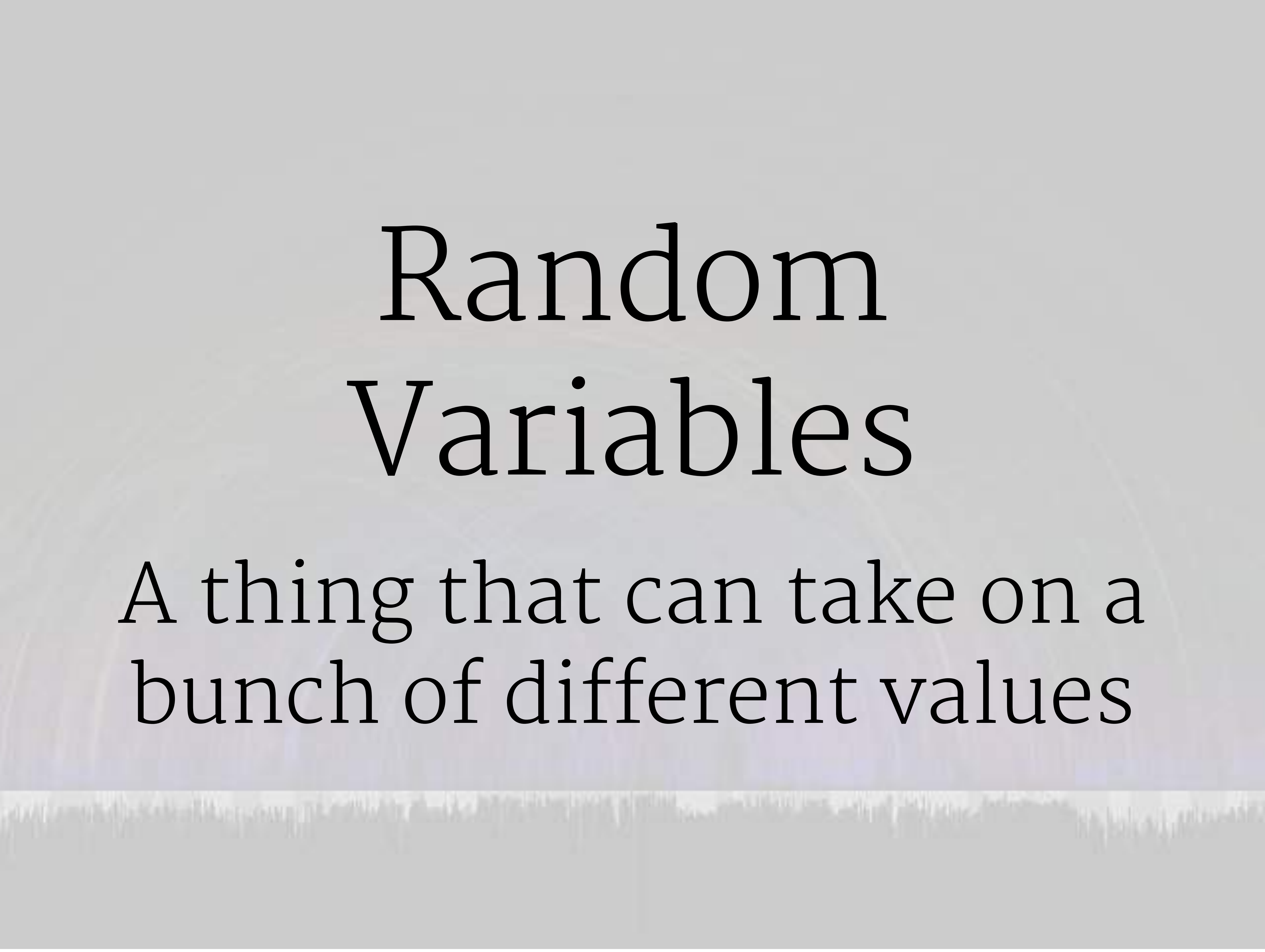
understand the world.
- Statistics allows us to effectively quantify and communicate uncertainty.
- A better understanding of statistics leads to more sober and self-aware decisions for our organization. Decisions based on a what the data say, and not a "feeling" we had on our commute home from work.
- On the job, I use statistics to:
-- For customers that were and were not assigned a person to their account, infer the true difference in 90-day retention probability
-- Assess whether there existed a difference in conversion for inbound chat vs. outbound chat
-- Determine the sample size needed for an experiment in order to obtain a desired level of confidence in its results
- In this section, we're going to cover what random variables are, what probability distributions are, and how



Inference

Statistics' Shiniest Sword

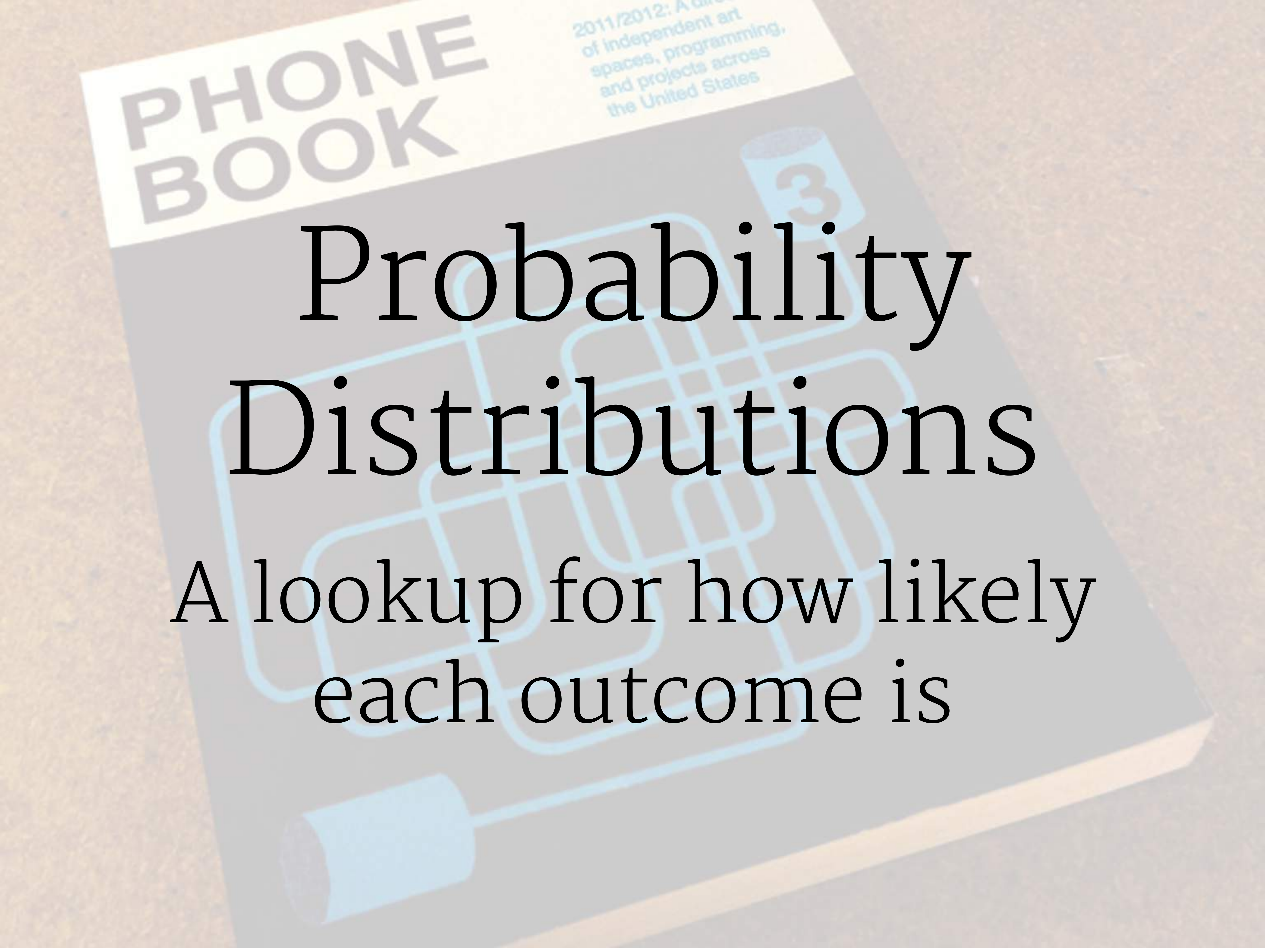
Suppose we have a die. We roll the die to generate values. Because we understand the die, we can ask all the questions and get all the answers.
^In the real world, we start with the data - the values that the die generates, and without knowing the "die" that is generating these values in the first place. Statistics helps us to uncovers what this "die" really is.



Random Variables

A thing that can take on a bunch of different values

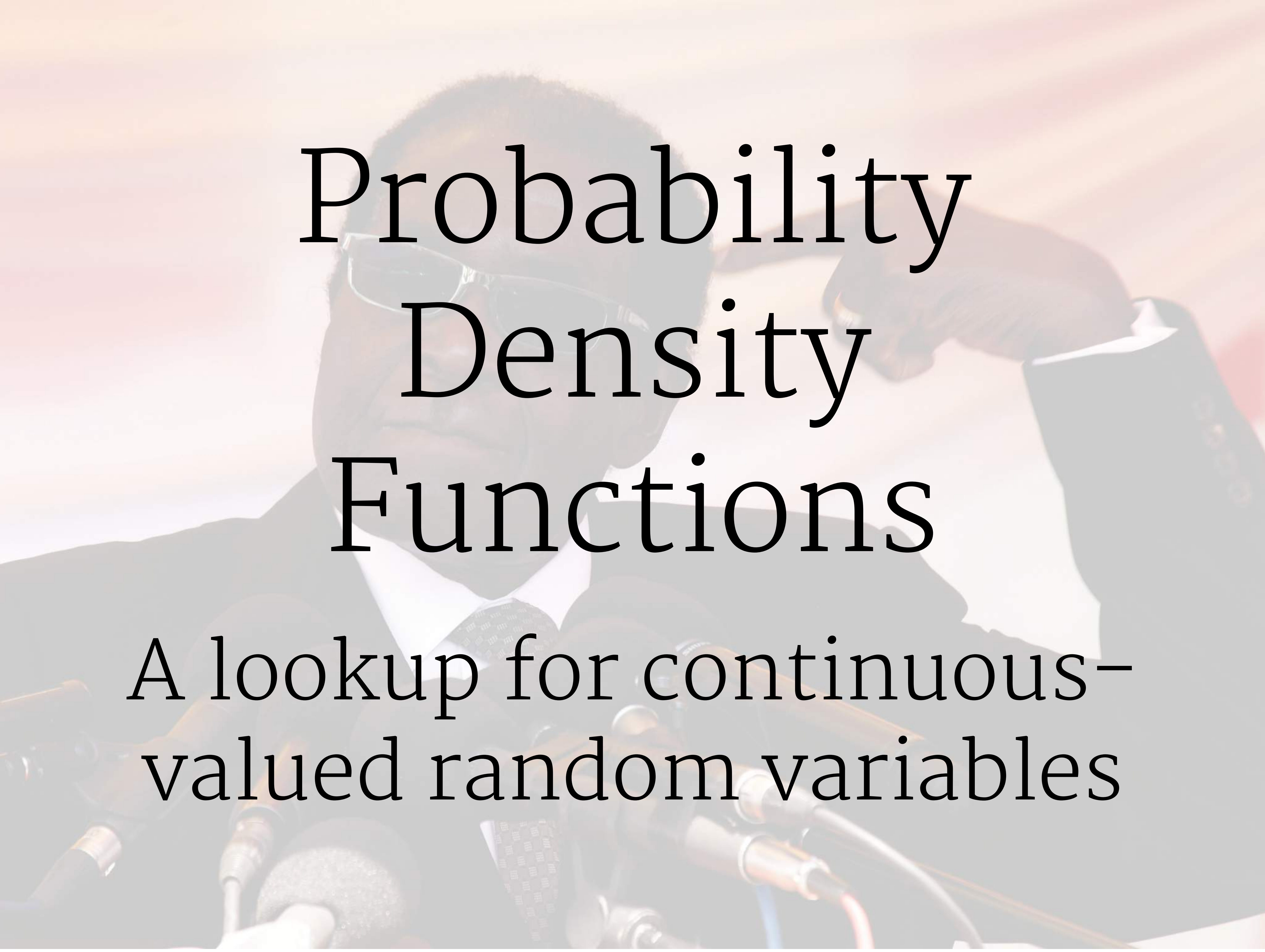
The color of my shirt that I wear to the office on Mondays
^The height of a person from Mr. Smith's History class
^The political tenure of despotic presidents in Central Africa


PHONE BOOK
2011/2012: A directory of independent art spaces, programming, and projects across the United States
3

Probability Distributions

A lookup for how likely each outcome is

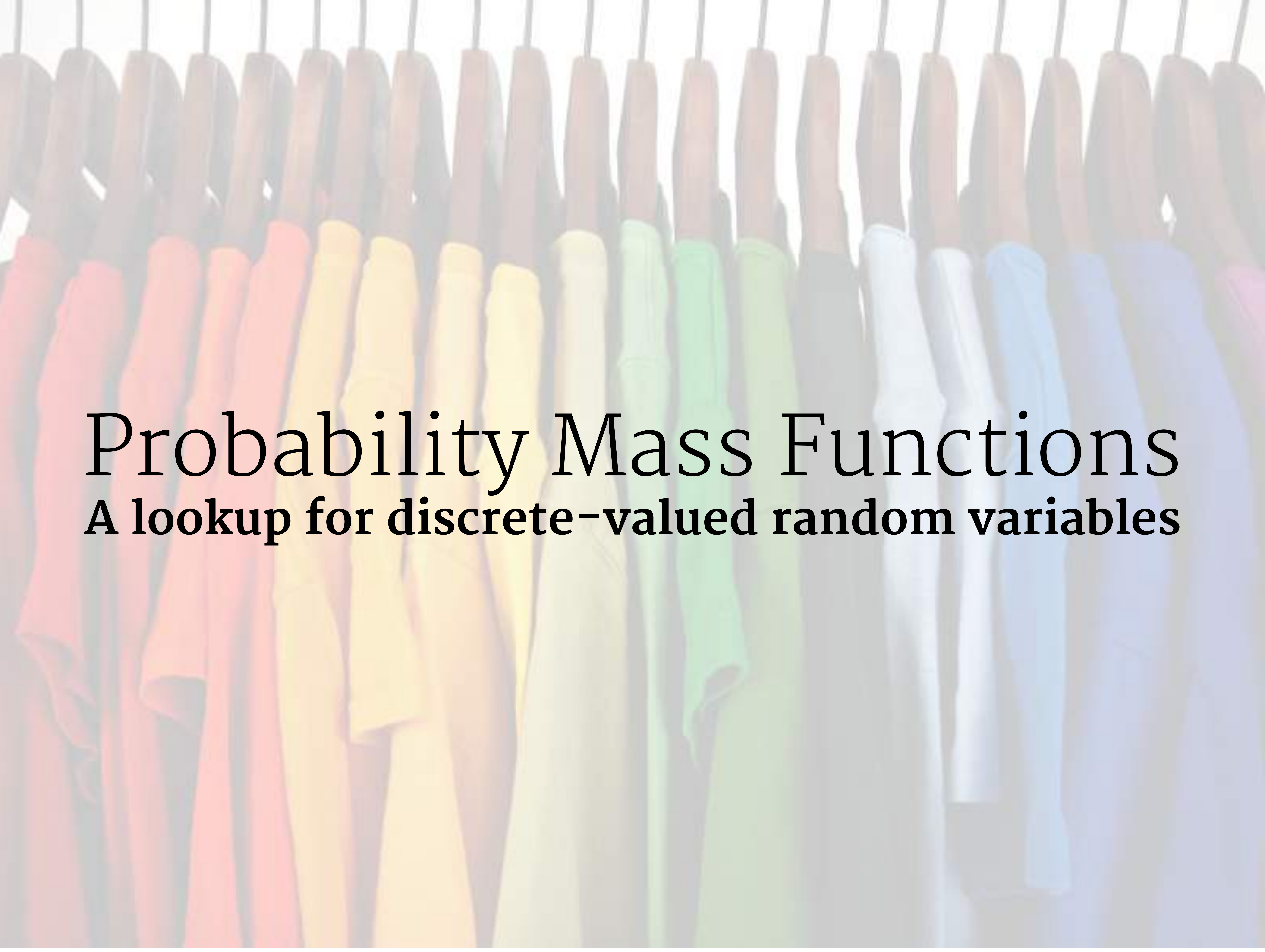
- Effectively a list of tuples: [('grey', .8), ('blue', .1), ('bright green', 1)]
- We can represent this visually, in blocks



Probability Density Functions

A lookup for continuous-valued random variables

The tenure of despotic rulers
^The probability at a single point is always 0
^Probabilities are measured over intervals, not single points
^The area under the curve between 2 distinct points defines the probability for that interval
^The height of a PDF can in fact be > 1
^The integral must equal 1! (just like in a probability mass function)



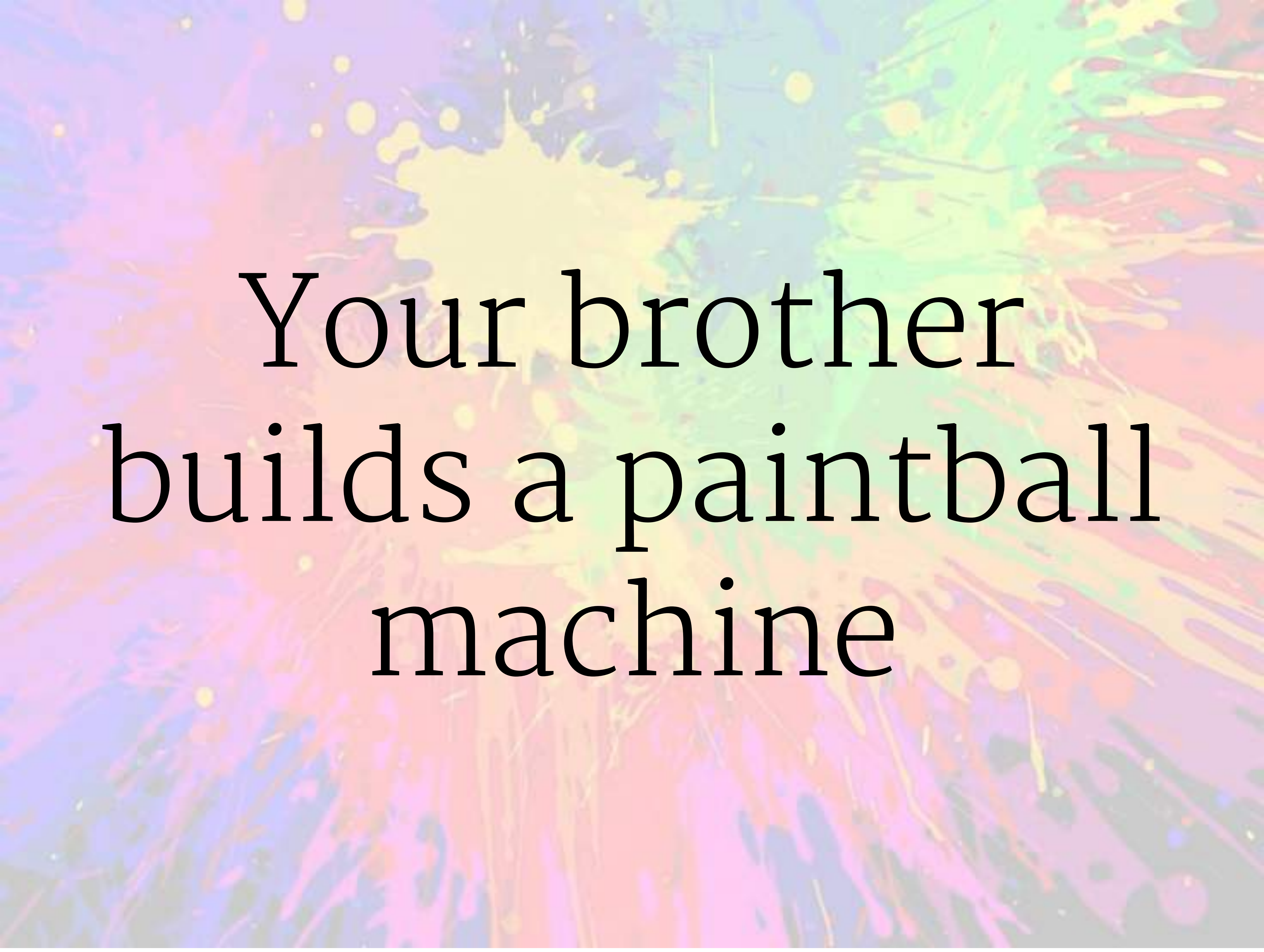
Probability Mass Functions

A lookup for discrete-valued random variables

Into the Sunset with a PDF at My Side

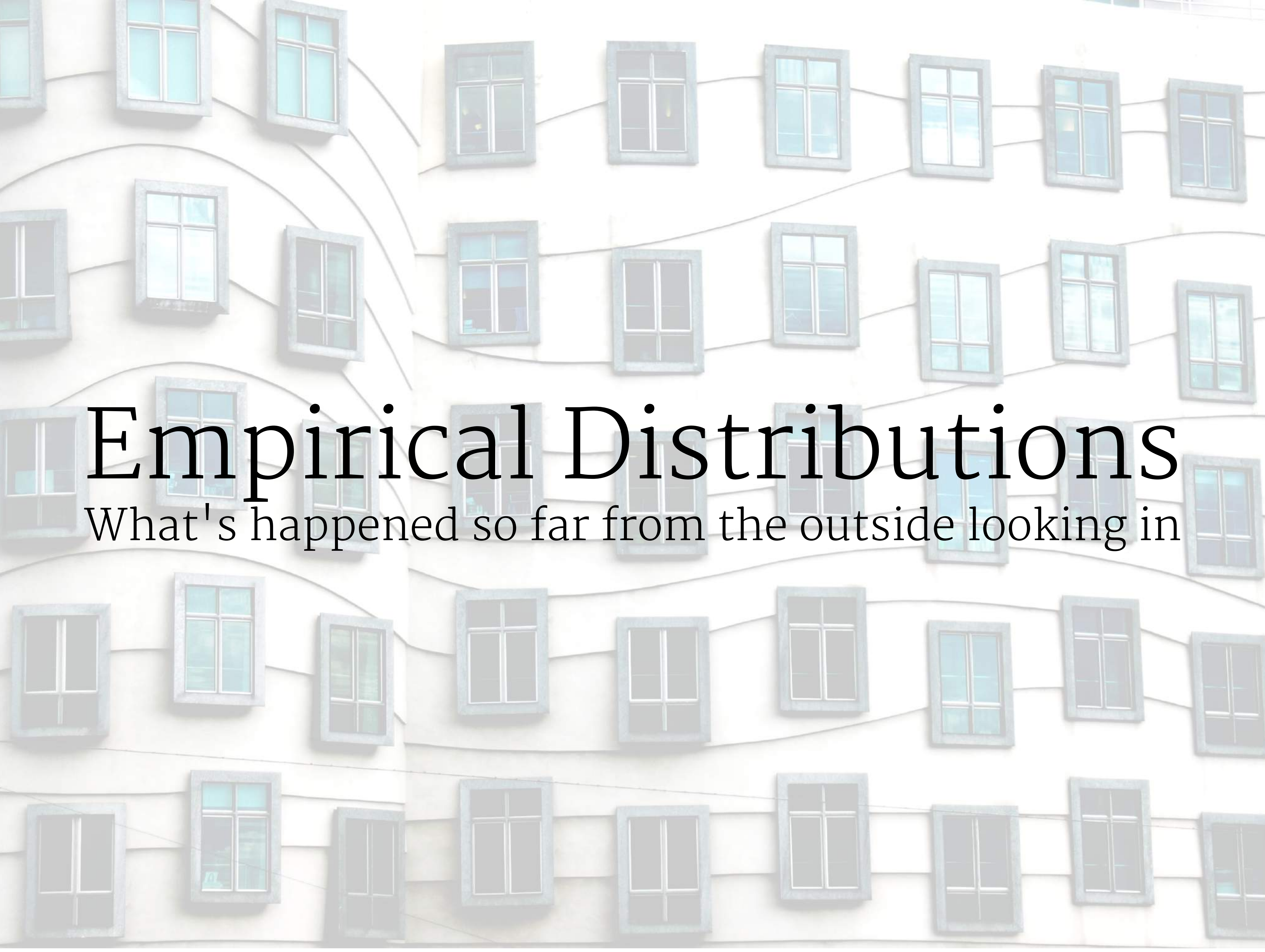
Of a PDF, we ask questions. All the questions. And get all the answers.

Of a PDF (probability density function), we can ask: "What is the probability that the ruler lasts for more than 8 years? What is the probability that the ruler lasts for between 11 and 14 years?"
^Conversely, we cannot ask: "What is the probability the ruler lasts for 9.764135 years?," because the probability of any given continuous value is ~0. Does it make sense why?
^Of a PMF (probability mass function), we can ask: "What is the probability that, on a given roll of a die, our die will show 6?" Each discrete outcome has a defined probability. We cannot ask: "What is the probability that our die will show a number between 3 and 6?" OK, I guess we can. But that's only because our random variable (the die) has an implicit ordinality (6 > 5 > 4 > etc.).
^If our random variable - and remember, a random variable is just a thing that can take on a bunch of different values - were a "the style of shirt that I will wear on a given day", then we could not ask of our PMF "what's the probability that the shirt is between green and polka dot?" Clearly, this does not make sense.



Your brother builds a paintball machine

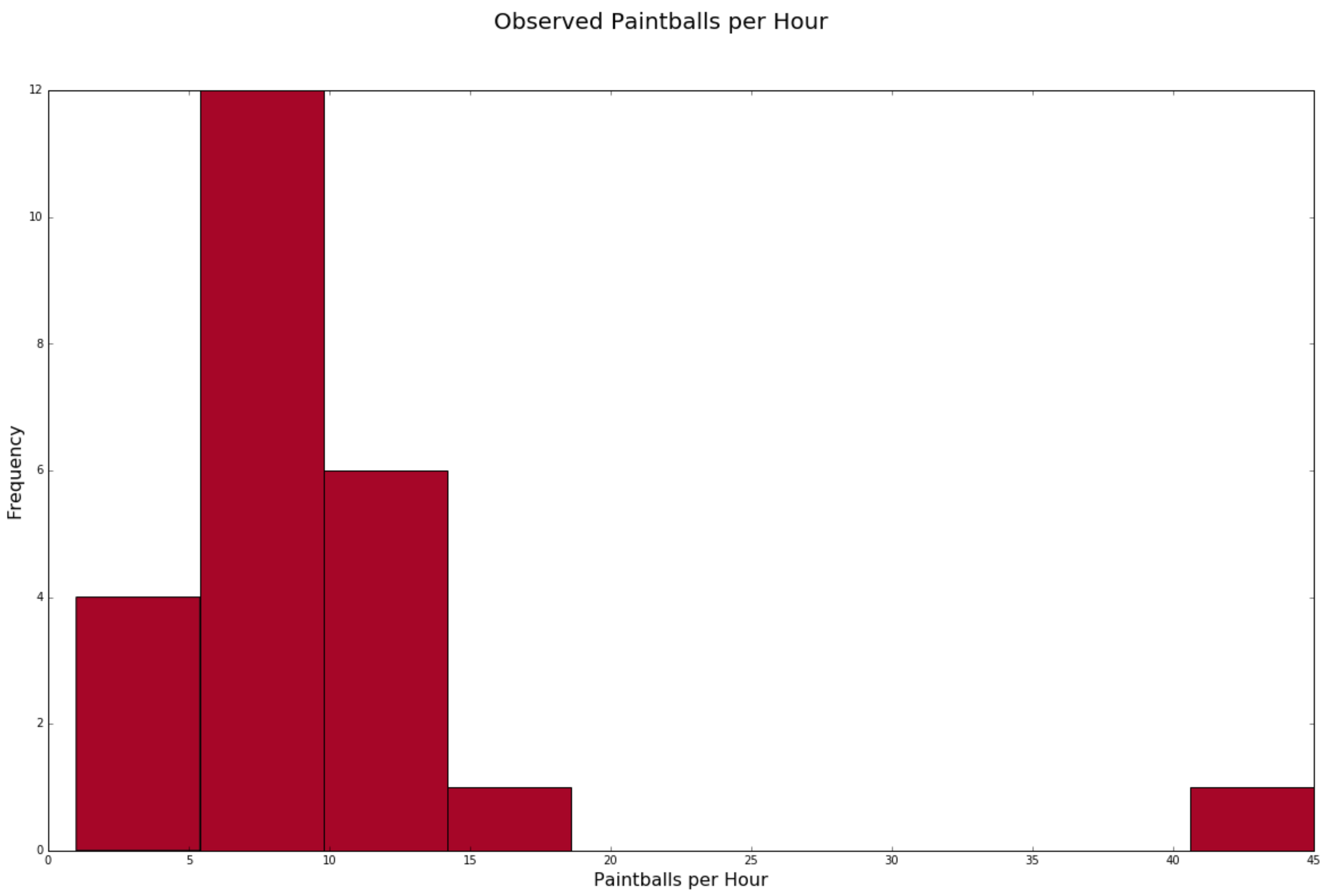
Your brother mounts a paintball gun outside of his window, and points it at you as you play outside in the yard. The gun is linked to a controller, which dictates the frequency with which the paintballs are fired. He sets the controller, and heads downstairs to make a turkey sandwich. You're in the yard and you want to keep playing. But you're under fire.

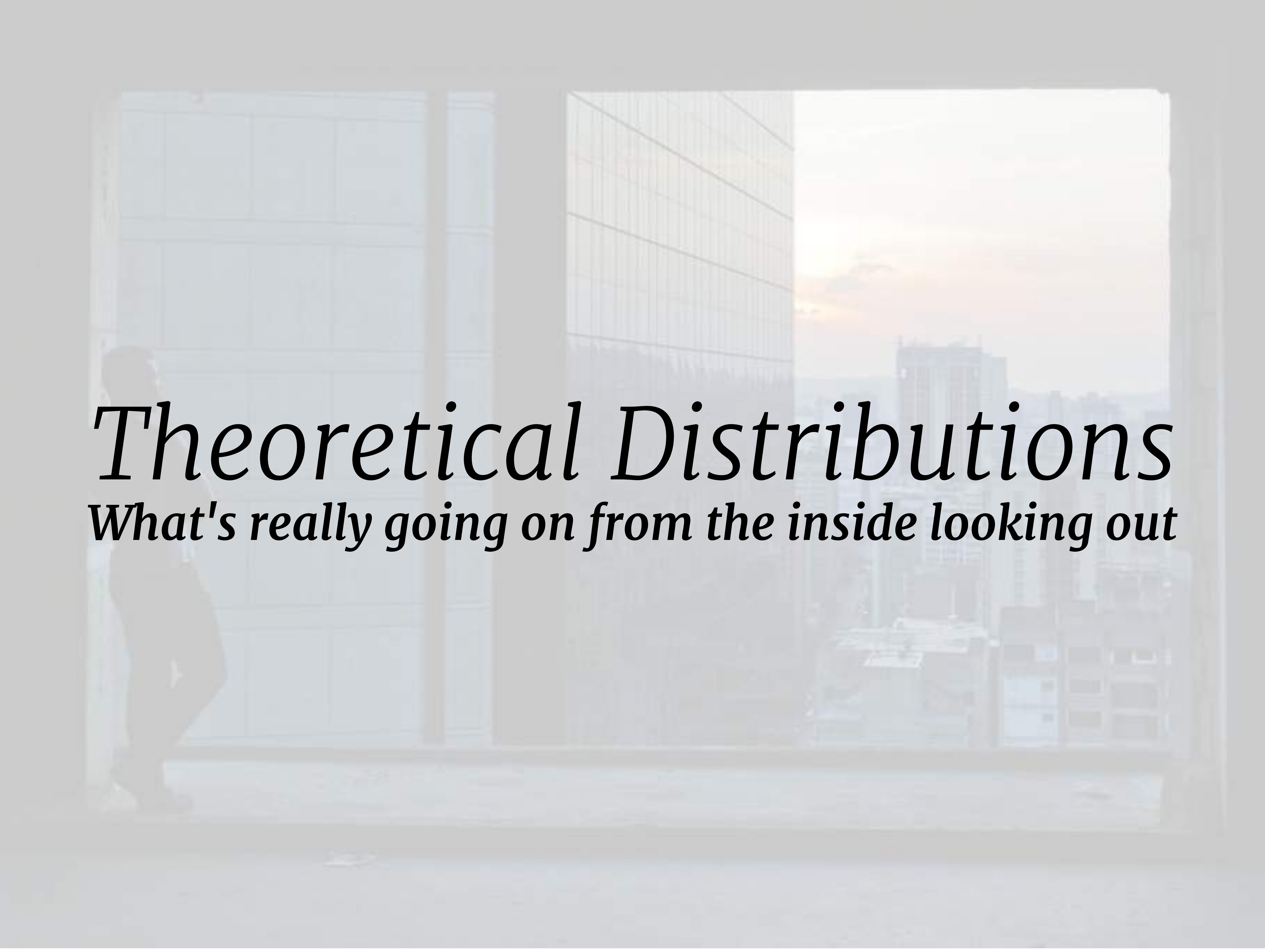


Empirical Distributions

What's happened so far from the outside looking in

So once more - we're out in the lawn. Summer is ablaze, and our yard is green. We need to figure out just what is going on with this paintball machine, so we can make the most of this beautiful backyard.
^In solution: we go into the tree-house and keep watch. Each hour, we record how many paintballs make landfall.
^Far more boring than they sound, it's just a picture of what we've seen so far.

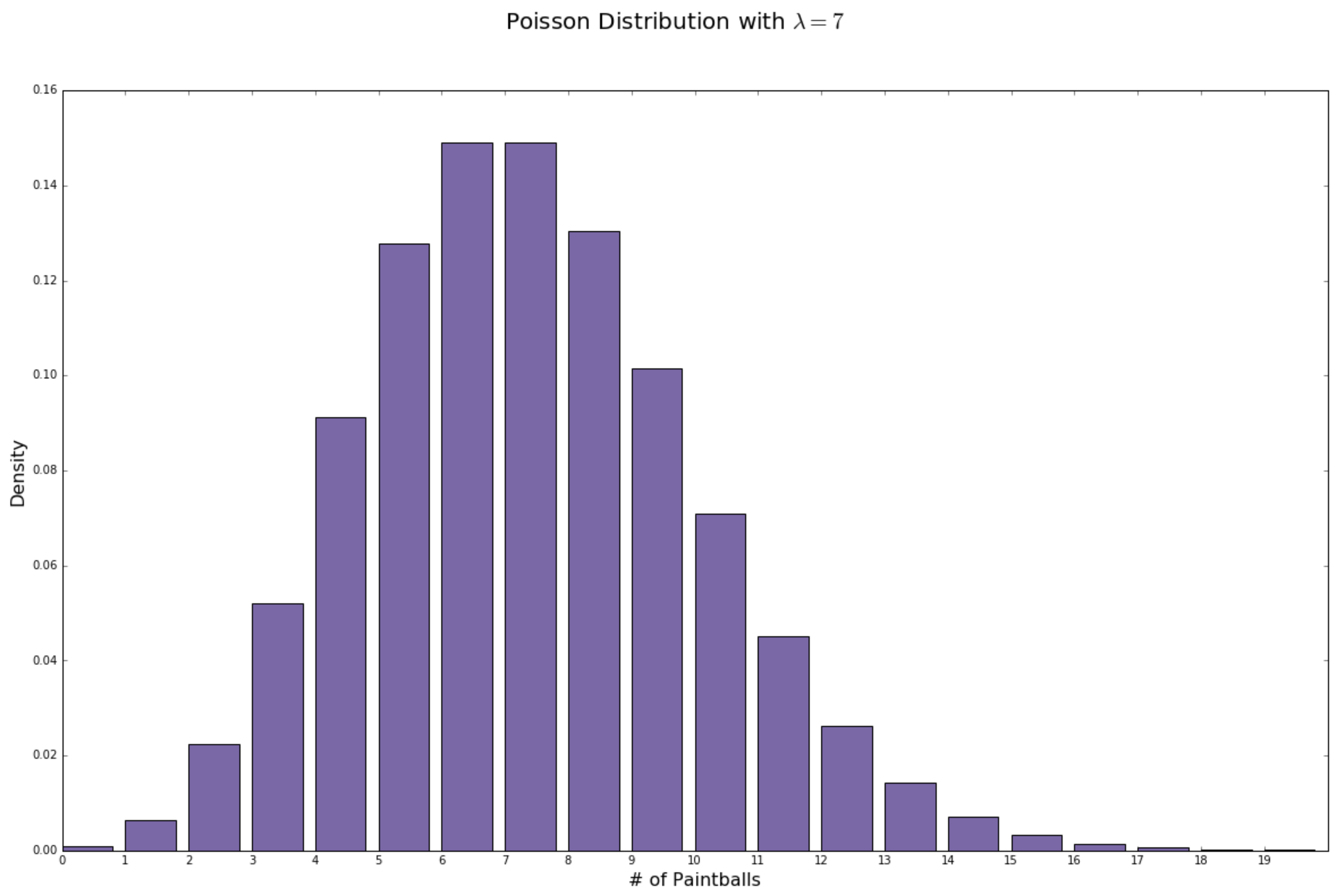

Observed Paintballs per Hour
Frequency
Paintballs per Hour
0
2
4
6
8
10
12
0
5
10
15
20
25
30
35
40
45



Theoretical Distributions

What's really going on from the inside looking out

In the theoretical case, we think of the problem in reverse. Smart math people have given us many canonical probability distributions that, in theory, describe the behavior of a given type of event. Exponential distributions describe survival events. Beta distributions describe coinflips. Poisson processes describe things that take on integer values. Dirichlet, Gamma, Binomial, Wishart, etc. distributions describe other things.
^Behind the window, we surmise that there is a Poisson process governing the number of paintballs fired per hour. And what does a Poisson process need? It needs a $\lambda$ and nothing else.



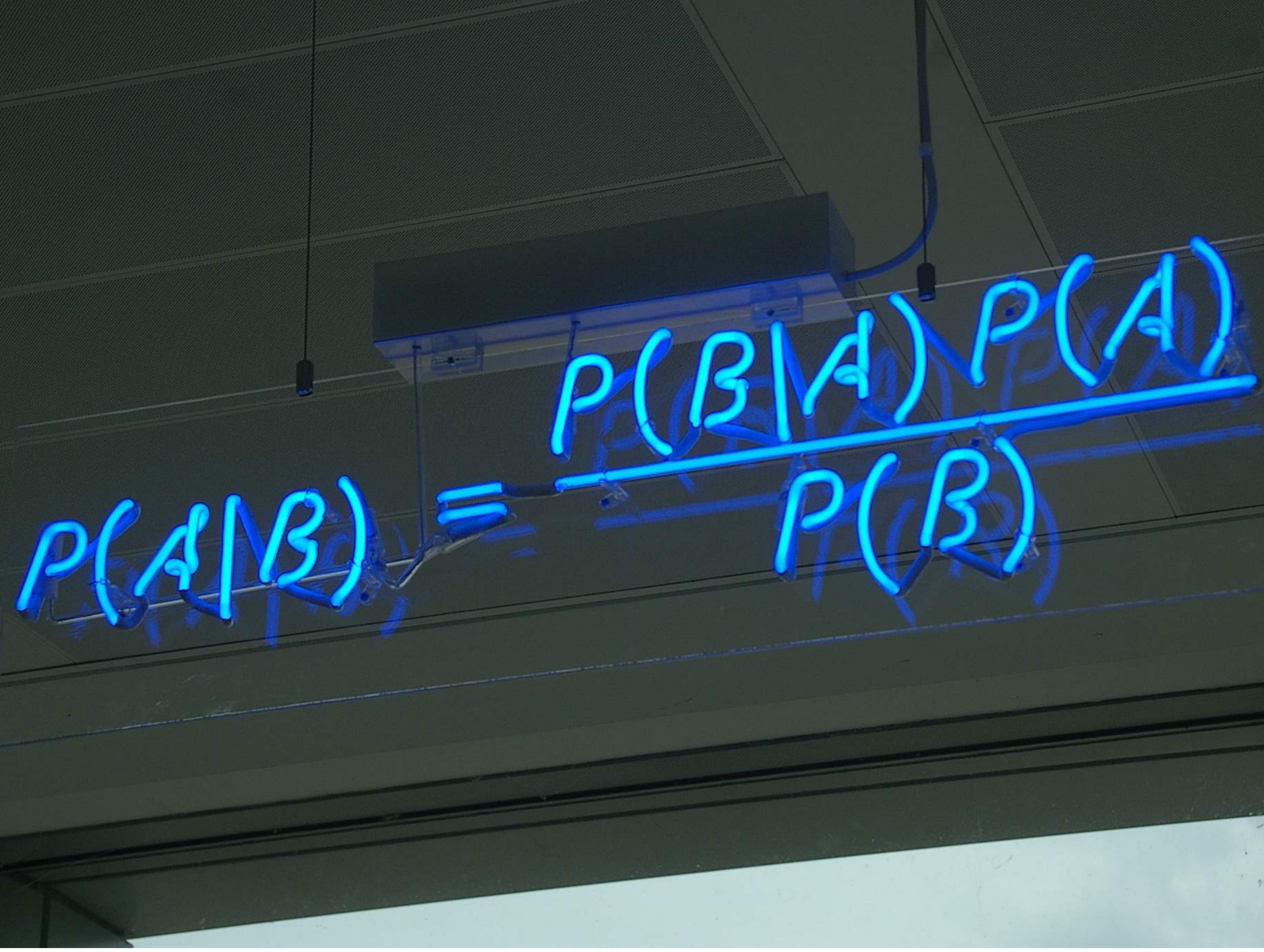
This, like any chart, is merely a function. Into the function we give a value. Out of the function we get a value. In other words, this chart is just a picture of a bunch of tuples.
^Assigns probability to every non-negative integer.

$$p(k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

We can use Bayesian statistics to infer probable values of the true λ using an appropriate prior and our observed data.
^For what it's worth: the expected value - a.k.a. the "long run average" - of a Poisson process is equal to λ.

Smashing the Wall

- Bayesian inference is concerned with beliefs about what our unknown parameter(s) might be. Rather than trying to guess our parameter(s) exactly, we assert what our beliefs in these parameters are, and what uncertainty we have in these beliefs.
- In frequentist statistics, this is known as maximum likelihood estimation. In Bayesian statistics, this is known as Bayesian inference.



Simply updating your beliefs by considering new evidence.
^Example: You frequently eat lunch at Vien. And for good reason too: Vien is pretty great. What do you believe the probability of your curry tofu being under-cooked is on any given day? Probably close to 0.
^One day, your tofu is clearly uncooked. Now, what do you believe this probability to be? Your belief probably increases - perhaps from 1% to 3% - but it's still pretty low. Once more, we have sufficient faith in Vien as a food-serving establishment.
^A month later, and 4/5 of our weekly Vien meals come with undercooked tofu. Now, we're starting to change our mind. Perhaps it wasn't a fluke. Perhaps the probability of getting undercooked tofu on a given day is something like 10%.
^You're thinking Bayesian: updating your beliefs based on evidence. It's pretty simple. You probably already think like this yourself.

Tools we all need

At the risk of sounding haughty, I like sharing the example of the PhD bootcamp.

Command Line

Likely the very best tool for inspecting and manipulating (in simple ways) large sets of data
^Accessing servers
^Basically, it's an indispensable tool of every software engineer. And because we're software engineers, we need the command line.

Git

project.
^Suppose you want to create a new feature. What you don't do is:
^- Try it on your main project.
^What if it fails? How do we just.. pick up where we left off?
^How do we keep track of changes when working with a team? We do not:
^- Email static code. If your organization is doing this, this is likely your single most pressing issue. Drop everything and solve it.
^Git is a programming language. It will take you 3 hours to learn its core pieces, and use it reasonably well. It will take you 2 months to develop a nice Git workflow.
^Data Scientists should learn this at the beginning

Notebooks

Our scrap paper.
^Notebooks allow us to:
^- Run code out of order
^- Create visualizations inline
^- Create a document detailing our workflow, that we can share with others
^- I like Jupyter. In all honesty, I haven't tried many others.

AWS

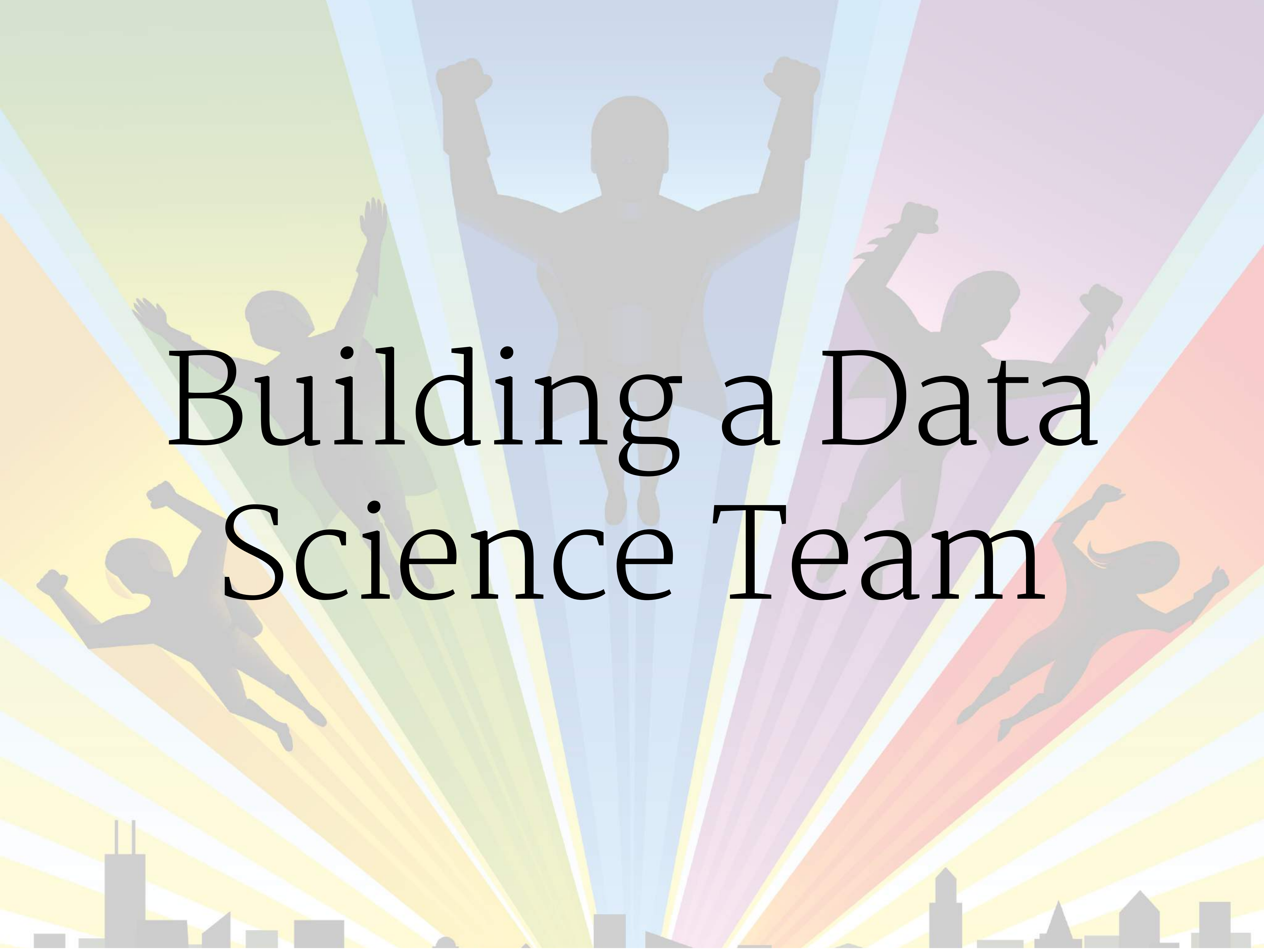
We need computers to run our jobs for us. We need databases. We need storage.
^Remember: the cloud doesn't exist. It's just a computer on a farm somewhere.
^EC2
^S3
^Redshift

Distributed Computing

Andreas' out-of-core machine learning anecdote: 'just get a bigger instance.'

Sometimes, we really need it.

One computer can only do so much. With a lot of data, we need more computers.
^Give "counting students" example. The one with the students in the hallway.
^Basically, parallelizing a task is not as simple as saying "hit it with more computers." It becomes an algorithmic and engineering challenge in its own rite.
^As the years move forward and data grows larger, distributed computing will likely move from a thing Google does to an occupational necessity.
^This said, we will likely have really good abstractions (such that we don't have to think about this stuff at all) just before we arrive at this point.



Building a Data Science Team

Still a work in progress. Your average team is probably 5-10 people. There are very, very few organizations with Data Science teams over 50.
^15 years ago, we hired 1 person to "do our computers" and wondered why we'd ever need to hire more. Now, companies can't hire enough engineers, nor pay them enough.
^In 10 years, Data Science will be the same. We're very much in that "in-between" phase right now.

Data Engineers

In a perfect world, you'd hire data engineers first. These would be the people that build ETL pipelines, making sure clean, fresh data is available every day.
^From there, a data scientist would play with this data and build models.
^I would submit that the number of Data Engineers needed is fairly straightforward. We can use the same heuristics that we do for sizing our Engineering team. There is not a ton of uncertainty around the complexity of the projects they will work on.

Product People

Data scientists are often asked, on top of the 3 sides of the unicorn, to come up with ways to make the company millions themselves. This should ultimately be a full-time job of a product person.
^In practice, Data Science teams often either: work adjacent to product teams, i.e. operating on their own, with periodic input from product, or work fully-integrated with product teams, i.e. there is a full-time product person on the Data Science team.
^Some Data Science teams are just doing internal business optimization, i.e. "Analytics." In this case, a product person probably isn't necessary at all.
^Other Data Science teams are building consumer-facing features. In this case, it's the product people that understand (or should understand) how the end-solution will be used by the end-user. They should be driving the project; they should be the gate between Data Scientist and thing-in-production. A full-time person is useful here.
^In any case, organizations should think meticulously about what their Data Science team will do, and the relationship this team should have with Product. This is a conversation that should be had earlier rather than later.

Data Scientists

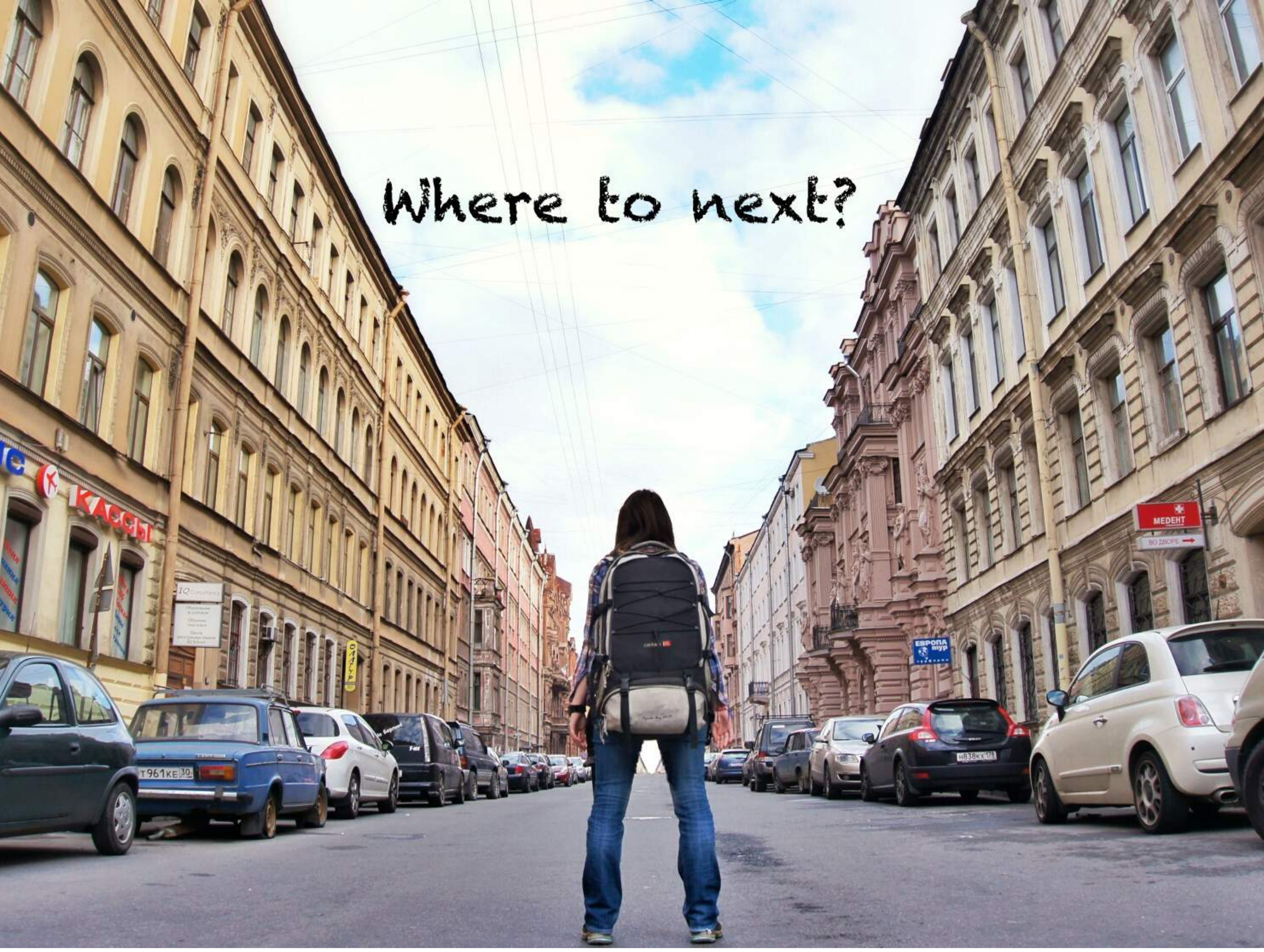
In a perfect world, they are building models and assisting with the implementation of their solutions.
^"Assisting with" is a term that needs definition. I've been a bit self-righteous about Data Scientists needing expertise in all camps.
^At a minimum, it is safe to say that in a perfect world, a Data Scientist should not be spending much more than 30% of their time building ETL systems. Where to go from there is still up in the air.
^How many? Good question. It really depends on the projects you're working on. Data Scientists have these powerful hammers, but can often lack nails. You don't want a bored Data Scientist. To keep them not bored, you need a steady flow of projects, which is almost analogous to saying that you have a management team that understands how Data Science can help the business.

Embedded vs. Siloed

One Data Science team? Data scientists embedded into other teams?
^Embedded allows us to do things in a data science-y way.

What's Worked, and What Hasn't

Hire data engineers first.
^Decide the relationship that data scientists and product managers should have, so as to best accomplish the projects at hand.
^Have data scientists touch other parts of the organization, whether through an embedded model, weekly presentations, etc.



Be the tornado. Consume, consume, consume.
^Kaggle. Meetups. Textbooks. Personal projects. The amount of information is far greater (and only growing quicker) than 1 human could consume in a lifetime. Promptly cancel your plans to play video games with your friends, and all other non-urgent social commitments, and get to work.
^I do promise: data science is really good fun.