Foundations of Data Science

Cambridge Computer Science Tripos Part IB, Paper 6

Ashwin Ahuja

2019

Table of Contents

[Probabilistic Modelling 2](#_Toc533822943)

[Likelihood 2](#_Toc533822944)

[Random Variables 2](#_Toc533822945)

[Independence and joint distributions 3](#_Toc533822946)

[Rules for Conditional Probability 3](#_Toc533822947)

[Rules for Joint Distributions of Random Variables 3](#_Toc533822948)

[Fitting Distributions 4](#_Toc533822949)

[Custom Distributions 4](#_Toc533822950)

[Inversion Method 4](#_Toc533822951)

[Fitting a model 5](#_Toc533822952)

[Random Variables 6](#_Toc533822953)

[Mean and Variance 6](#_Toc533822954)

[Confidence Intervals 6](#_Toc533822955)

[Convergence Theorems 7](#_Toc533822956)

[Monte Carlo integration 8](#_Toc533822957)

[Empirical Distribution 9](#_Toc533822958)

[Standard Random Variables 9](#_Toc533822959)

[Geometric 9](#_Toc533822960)

[Exponential 9](#_Toc533822961)

[Binomial 10](#_Toc533822962)

[Multinomial 10](#_Toc533822963)

[Poisson 10](#_Toc533822964)

[Normal / Gaussian 10](#_Toc533822965)

[Pareto and Lognormal 11](#_Toc533822966)

[Zipf 11](#_Toc533822967)

[Beta 11](#_Toc533822968)

[Dirichlet 11](#_Toc533822969)

[Gamma 11](#_Toc533822970)

[Inference 12](#_Toc533822971)

[Bayesianism 12](#_Toc533822972)

[Nuisance Parameters 14](#_Toc533822973)

[Frequentism 14](#_Toc533822974)

[Resampling 15](#_Toc533822975)

[Model Selection 15](#_Toc533822976)

[Hypothesis Testing and P-Values 15](#_Toc533822977)

[Alternative Hypothesis 16](#_Toc533822978)

[Creating a Model 16](#_Toc533822979)

[Logistic Regression 16](#_Toc533822980)

[Feature Spaces 16](#_Toc533822981)

[Fitting a linear model 16](#_Toc533822982)

[Features 17](#_Toc533822983)

[Linear Mathematics 18](#_Toc533822984)

[Orthogonal Projection 18](#_Toc533822985)

[Linear Regression and Least Squares 19](#_Toc533822986)

[Random Processes 19](#_Toc533822987)

[Markov Chains 19](#_Toc533822988)

[Limit theorems and equilibrium 20](#_Toc533822989)

[Detailed Balance 20](#_Toc533822990)

[Limiting Behaviour and Aperiodicity 21](#_Toc533822991)

# Probabilistic Modelling

## Likelihood

**Likelihood** is the probability of the observed outcome, viewed as a function of the unknown parameter. Important to note that likelihood is not a probability density – does not integrate to 1. **The likelihood function** measures how much evidence there is for a particular parameter value. **The maximum likelihood estimator** is the parameter value with the maximum likelihood.

**Example:** **take a biased coin**. Likelihood function is how much evidence there is for a particular value of p. The higher lik(p) is, the more likely that value of p is.

In order to get the value which maximise the likelihood, we solve:

Often, easier to maximise log(lik(.)) rather than lik(.) and it necessarily has the same solution:

## Random Variables

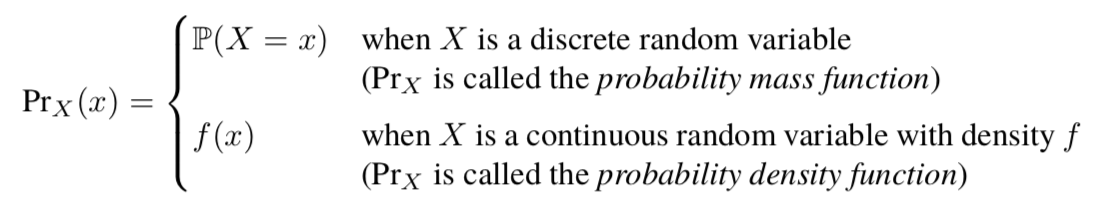
Random Variable is a function that can give different answers. It takes values in Ω means that the return value of the function is an element of the set Ω. Every random variable X has a probability distribution:

which specifies the probability that the return value lies in a subset A ⊆Ω – if Ω is countable, the random variable is said to be discrete and then:

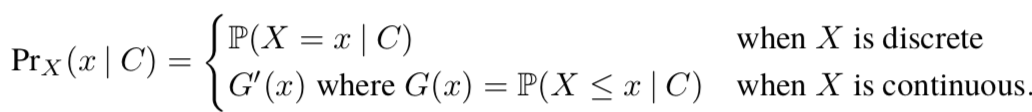
For many applications, work with real-valued random variables, and define the cumulative distribution function, to be:

If the function is differentiable, then X is said to be a continuous random variable with density function *f(x) = F’(x)* and:

For a continuous random variable, ℙ(X = x) = 0 for every x, and so:



A conditional random variable (X | C) is just a random variable whose distribution is conditional: if Y = (X | C) then ℙ(Y ∈ A) = ℙ(X ∈ A | C)



## Independence and joint distributions

Any pair of random variables (X, Y) has a joint distribution which specifies the probability of any joint event C. For a pair of continuous random variables, the joint distribution can be specified by a joint probability density PrX, Y (x, y)

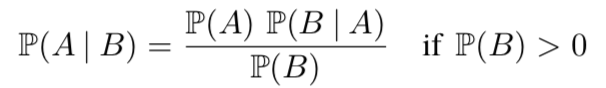
The marginal density of X is:

When random variables are discrete, you can just replace integrals by sums.

**Independence:** Random variables are independent iff:

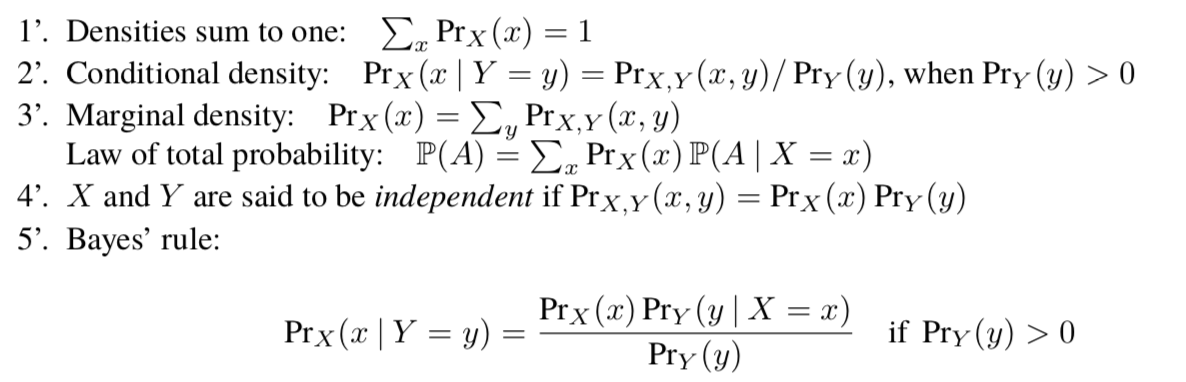
### Rules for Conditional Probability

A and B are events:

1. P(Ω) = 1 where Ω is the entire sample space
2. Conditional probability: ℙ(A | B) = ℙ(A ⋂ B) / ℙ(B), when ℙ(B) > 0
3. Sum Rule: If {B1, B2, …} partition Ω then ℙ(A) = ∑i ℙ(A ⋂ Bi)
   1. Law of total probability: ℙ(A) = ∑iℙ(Bi)ℙ(A | Bi)
4. A and B are independent iff
5. **Bayes Rule**
   1. 

### Rules for Joint Distributions of Random Variables

For continuous random variables, replace sums by integrals:



## Fitting Distributions

Random sample is a collection of random variables all drawn from the same distribution, and all independent of each other. If Y = (X1, …, Xn) is the random sample and y = (x1, …, xn) is a collection of values (dataset) then:

Where X is the common distribution. We also say X1, …, Xn are independent and identically distributed. If distribution of X depends on some parameter Θ which we’d like to estimate given a dataset. The likelihood given a single observation is:

And the likelihood given a dataset is:

Fitting the distribution means finding the maximum likelihood estimator for Θ, solving:

**Using Python:**

|  |
| --- |
| # Numerical solution (using a scaled loglik, for numerical stability)  def loglik(p, x):  return numpy.log(p) + (numpy.mean(x) − 1) \* numpy.log(1−p)  initial\_guess = numpy. array ([0.5])  mle = scipy.optimize.fmin(lambda p: −loglik(p,x), initial\_guess)  (p,) = mle #unpack mle, which is a list of length 1 |

**Multiple parameters:** Differentiate with respect to each in turn and then find where derivative is zero and then solve the simultaneous equations

## Custom Distributions

In order to design a distribution function is to plot the empirical cumulative distribution function:

From there, if the data moves too fast, take logs, then alter it – and split it into a thing where you can find straight lines from the distribution and then doing a MLE for all the parameters.

### Inversion Method

1. Generate a simple random variable U ∼ Uniform[0, 1]
2. Solve F(X) = U for X
3. Now X has cumulative distribution function F

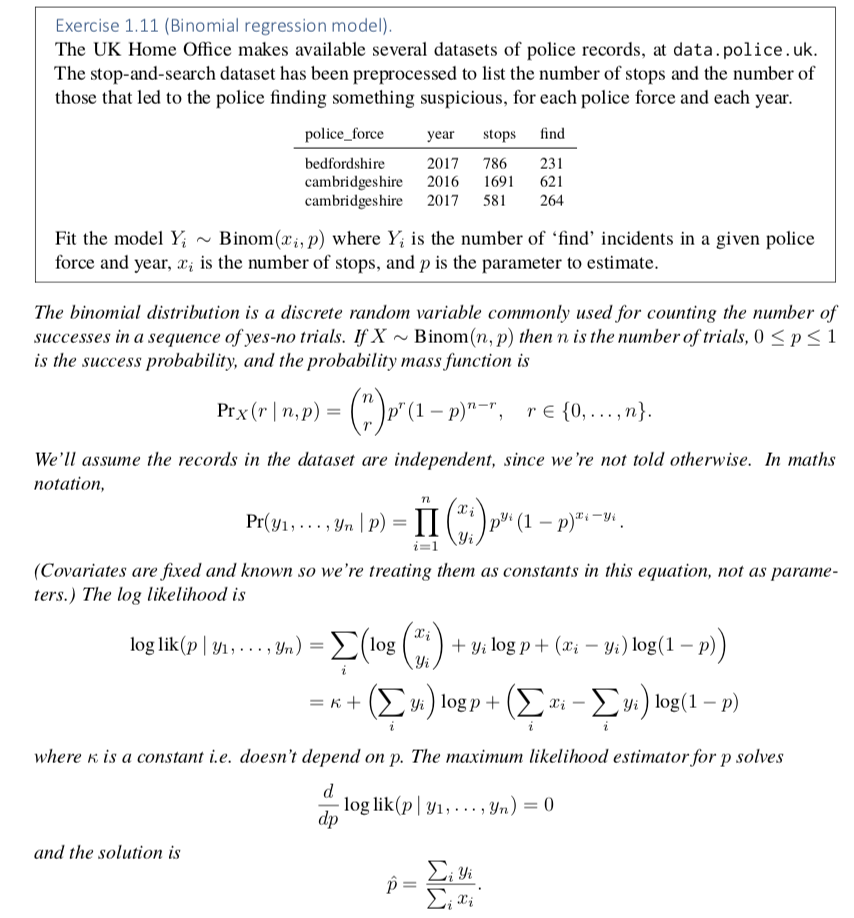
**Why it works:** ensures for ever x the event {X ≤ x} is precisely the event {U ≤ F(x)} which has the probability F(x). In regions where density of PrX is high then F will be steep, and so U is more likely to hit those regions.

This requires us to solve F(X) = U, which is easy to do algebraically for continuous functions like the straight lines fit. Method also correct for discrete random variables.

## Fitting a model

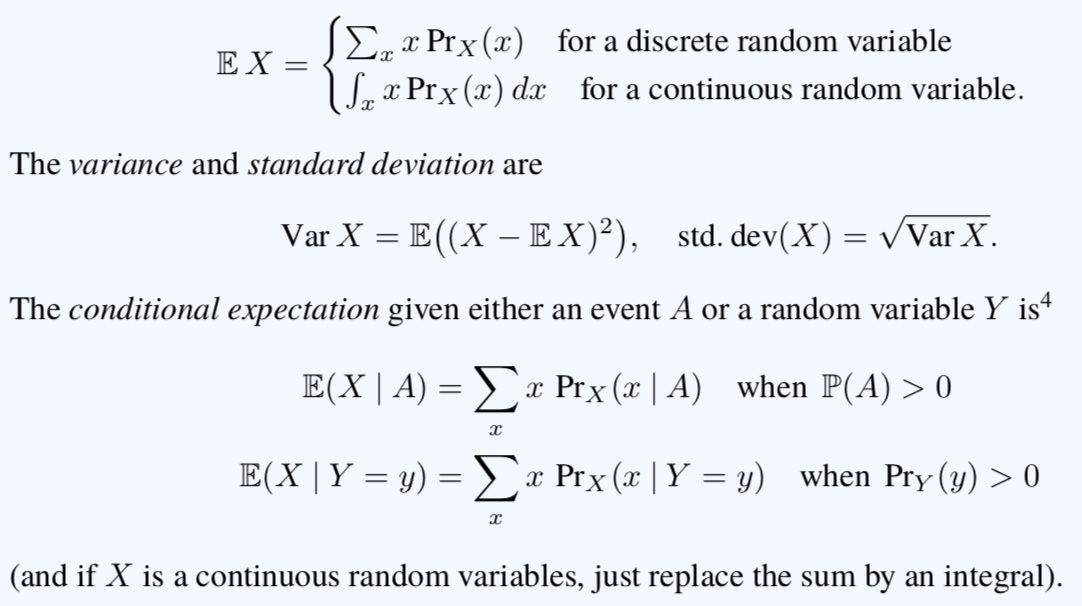
**Multivariate dataset:** contains records each consisting of a tuple of values. We want to understand how one item in the tuple depends on the others. The item we want to understand is called the **response variable** and the others are called **covariates or predictors**

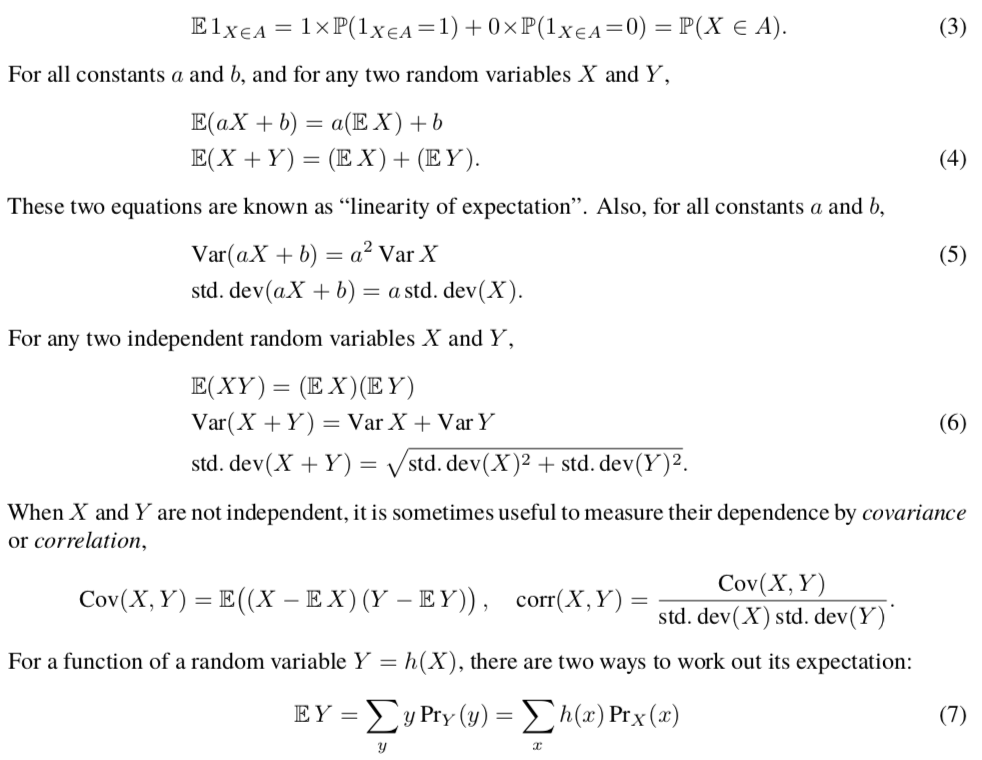
Invent a probabilistic model in which we treat the response as a random variable, whose distribution depends on both the covariates and on unknown parameters. Use the maximum likelihood estimation to estimate the unknown parameters – this is fitting the model.



# Random Variables

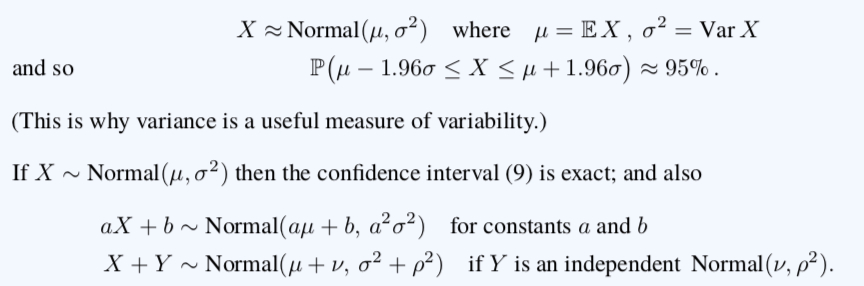
## Mean and Variance



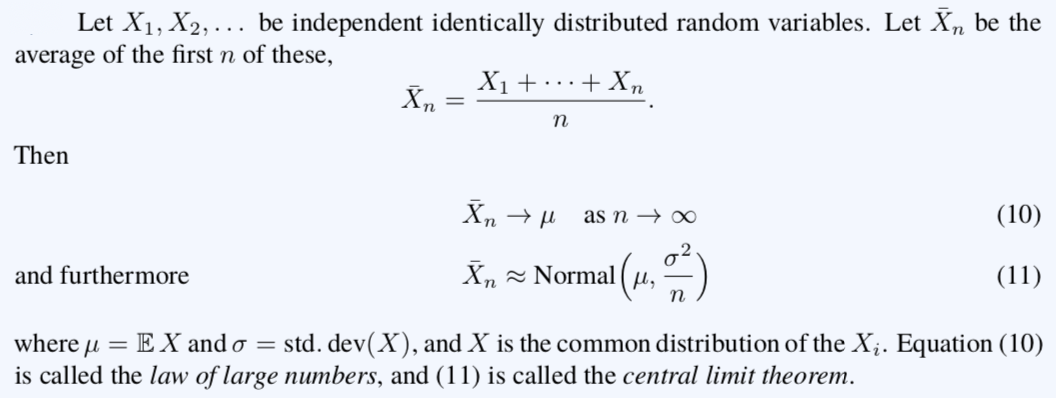


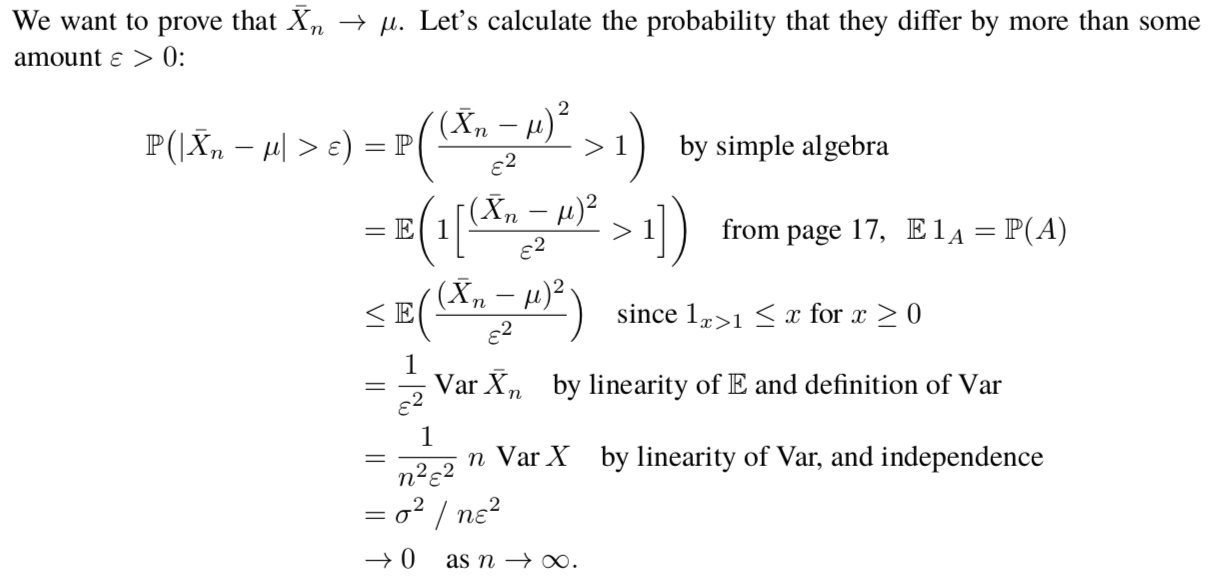
## Confidence Intervals

A random variable can be approximated by:



## Convergence Theorems



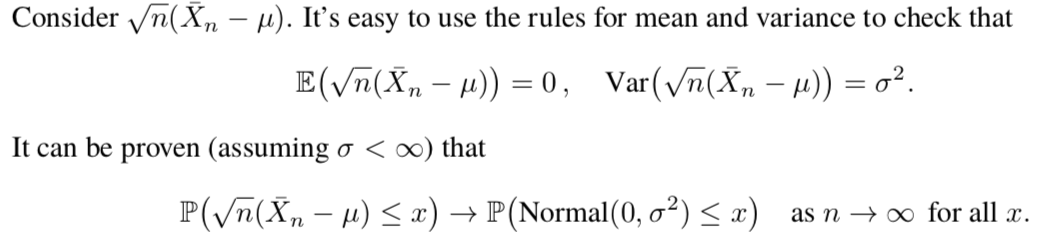
**Law of Large Numbers** ****

**Weak law of large numbers:** ℙ(| – μ| > ε) -> 0

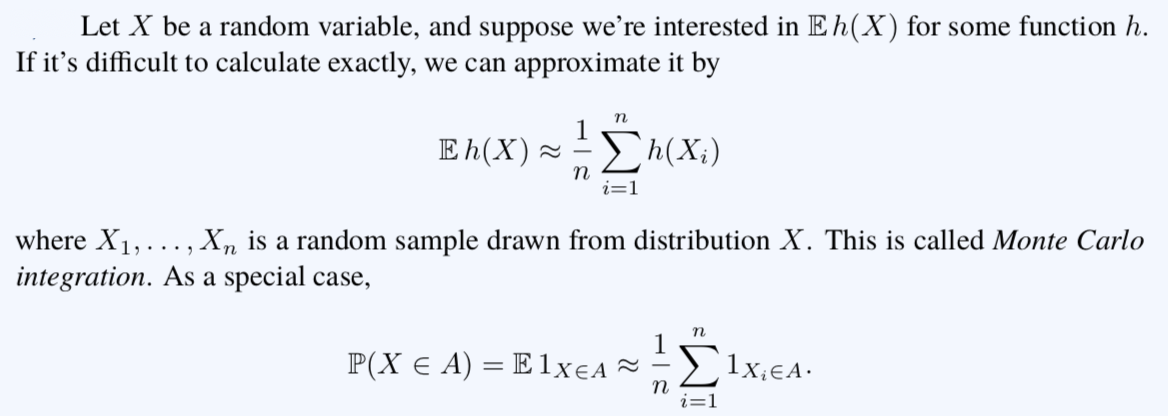
**Strong law of large numbers:** ℙ( -> μ) = 1 (this implies the weak law)

Therefore approaches μ as n -> ∞ and the smaller σ is, the smaller the error is likely to be. The central limit theorem additionally lets us find a confidence interval for the error.

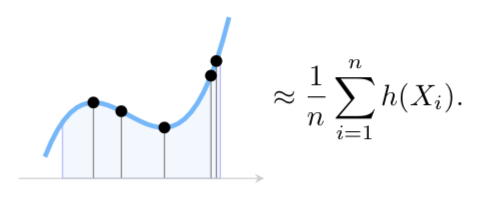
**Central Limit Theorem**

****

## Monte Carlo integration



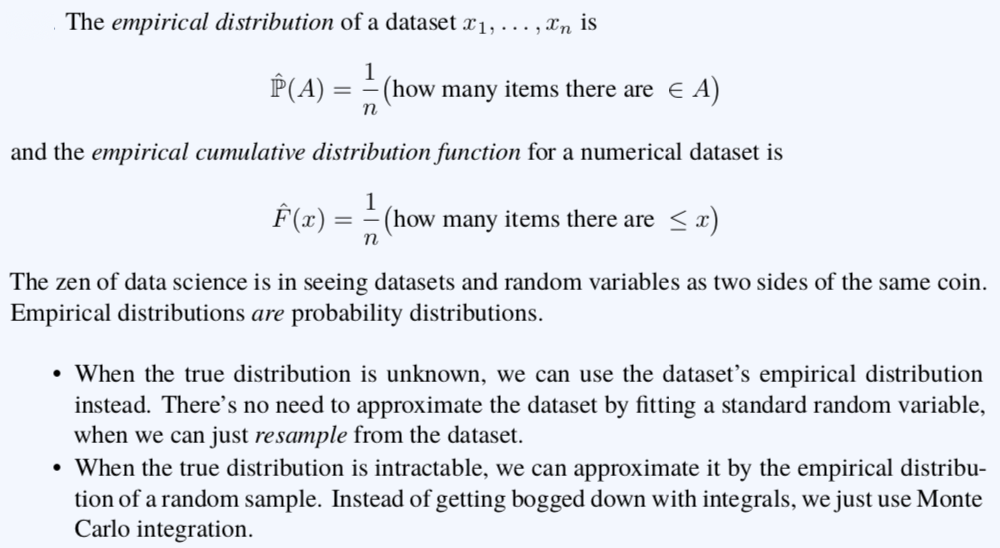
**Why it works:** where trapezium method, etc, split the integral into equally sized pieces, there is no real reason to do this, can just pick independent random variables and approximate:



**Error of the Monte Carlo estimator**: O(σ/) – however, need to be able to calculate σ, which is easier to estimate using the Monte Carlo method rather than calculating

can plug in estimate for μ using Monte Carlo method – can keep a running estimate of σ and can stop when the error gets small enough.

## Empirical Distribution



**Resampling:** Sample from the empirical distribution function using the inversion method – the same as picking a value at random from the dataset, with each item being equally likely. For a parametric distribution, we only need to store a handful of parameters, rather than the full dataset. Can also have **high-dimensional modelling (active research topic)** – modelling with more parameters than there are samples in the dataset.

## Standard Random Variables

### Geometric

X ∼ Geom(p)

**Takes values** in {1, 2, …, n}



**Mean:** 1/p

**Variance:** (1-p)/p2

**Python:** numpy.random.geometric(p)

### Exponential

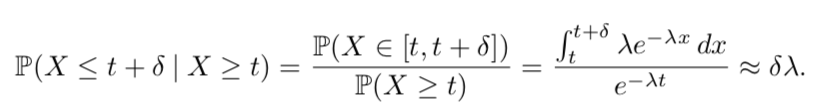
Used to model the time until an event – such as the time until a lump of radioactive matter emits its next particle.

X ∼ Exp(λ) – λ is called the rate

**Takes values** in [0, ∞]



**Chance of an event in a short interval of time [t, t + δ] is**:



**Mean:** 1/λ

**Variance:** 1/ λ2

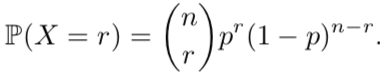
**Python:** numpy.random.exponential(scale = 1/λ)

### Binomial

Toss a biased coin n times, and each coin has chance p of heads, total number of heads. When n is 1, it’s called a Bernoulli random variable.

X ∼ Binom(n, p)

**Takes values in** {0, 1, …, n}



**Mean:** np

**Variance:** np(1-p)

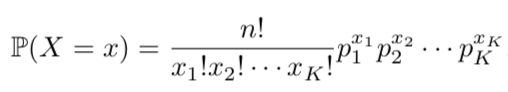
**Python:** numpy.random.binomial(n, p)

### Multinomial

N individuals each of whom falls into one of K categories and probability of falling into category k is pk

X ∼ Multinom(n, p)

**Takes values in** {0, 1, …, n}K



**Python:** numpy.random.multinomial(n, p)

### Poisson

If time between events is exp(λ) then total number of events in time t.

X ∼ Poisson(λt)

**Takes value in** {0, 1, …}

**Mean and Variance:** λ

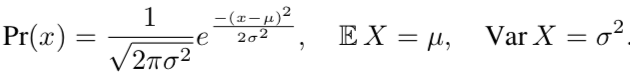
**Python:** numpy.random.poisson(lam = λ)

### Normal / Gaussian

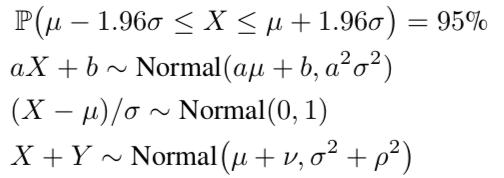
Good aggregate of lots of small pieces:

X ∼ Normal(μ, σ2)

**Takes value in** reals



If you have two independent distributions X ∼ Normal(μ, σ2) and Y ∼ Normal(ν, ⍴2) and a and b are real numbers



**Python:** numpy.random.normal(loc=μ, scale=σ)

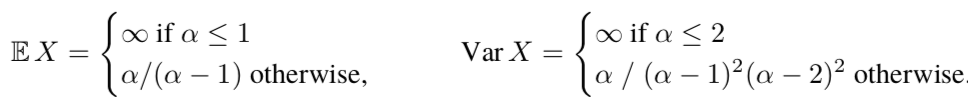
### Pareto and Lognormal

Avoid reliance on a single black swan event.

X ∼ Pareto(α)

**Takes values** in [0, ∞]

****

****

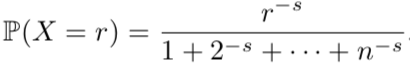
For α < 2, it tends to produce many small values and very occasional huge values

### Zipf

Describes frequencies of words in texts – comparing rank and frequency.

X ∼ Zipf(n, s)

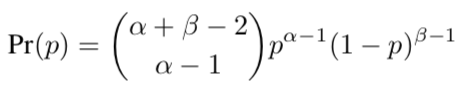
**Takes values in** {0, 1, …, n}



### Beta

If we toss a biased coin n times, and each coin has chance p of heads, then number of heads takes Bin(n, p). A common prior distribution for p is Beta(α, β).

**Takes values in** (0, 1) and has parameters α > 0 and β > 0



**Mean:** α/(α + β)

**Python:** numpy.random.beta(a=α, b=β)

### Dirichlet

Generalisation of Beta – has k categories and α is a vector in ℝK

**Takes value in**

****

Generates probability distributions over the k categories – used in Bayesian inference to describe belief about a multinomial distribution – interpretation is “seen αk items in category k”



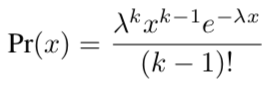
**Python:** numpy.random.dirichlet(alpha=α)

### Gamma

Sum of k independent exponential random variables – common choice of prior distribution for 1/σ2 in Bayesian calculations with Normal(μ, σ2)

X ∼ Γ(k, λ)

**Takes values** in [0, ∞]



(k-1)! Is replaced by gamma function Γ(k) for non-integer k

**Mean:** kλ

**Variance:** k/λ2

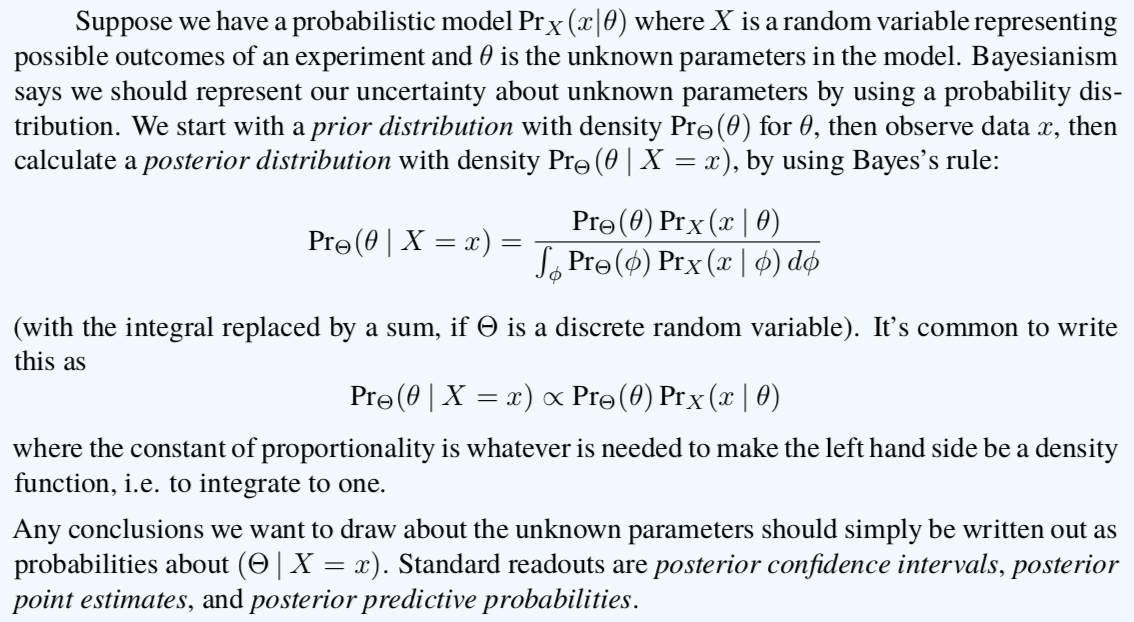
**Python:** numpy.random.gamma(shape=k, scale=1/λ)

# Inference

Reaching conclusions on the basis of data and reasoning.

## Bayesianism

Doctrine that one should represent uncertainty about unknown parameters by describing them as random variables. It insists that we set down a prior belief for all unknown parameters before we even think about incorporating data – doesn’t really matter what the prior is as long as the dataset is large enough

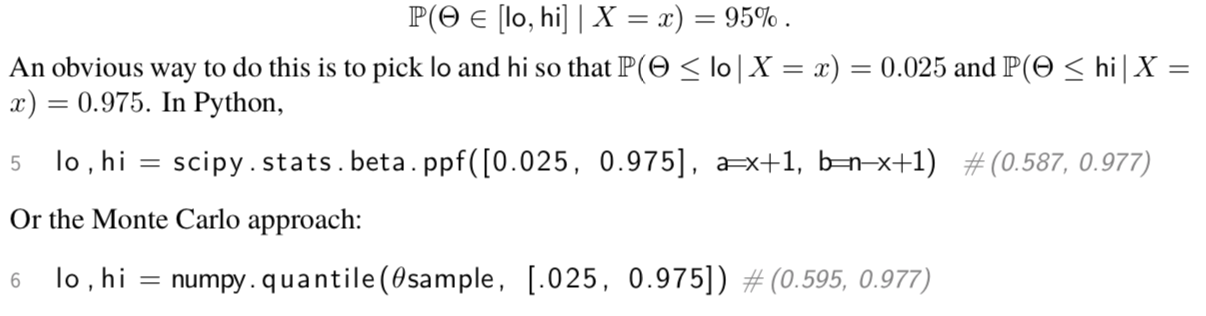


**Conjugate Priors:** When posterior distribution ends up belonging to the same family as the prior distribution, just with different parameters

**Sampling from posterior distribution:** Can use the Gibbs sampler (based on Markov chains) to simple from a posterior distribution

**Readouts from posterior distribution:** For example, to see if a coin is biased, see if ℙ(Θ > ½ | X = 9) – can get this through integrating the probability from ½ to 1, or by resampling using the posterior distribution then using Monte Carlo integration

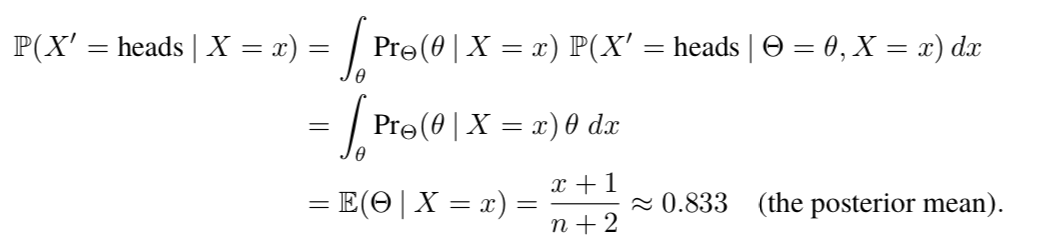
**Confidence Interval:** Can report confidence interval, therefore for Θ is a Uniform[0, 1]



**Posterior Point Estimates:** When asked given the observed data what is the value of the parameter:

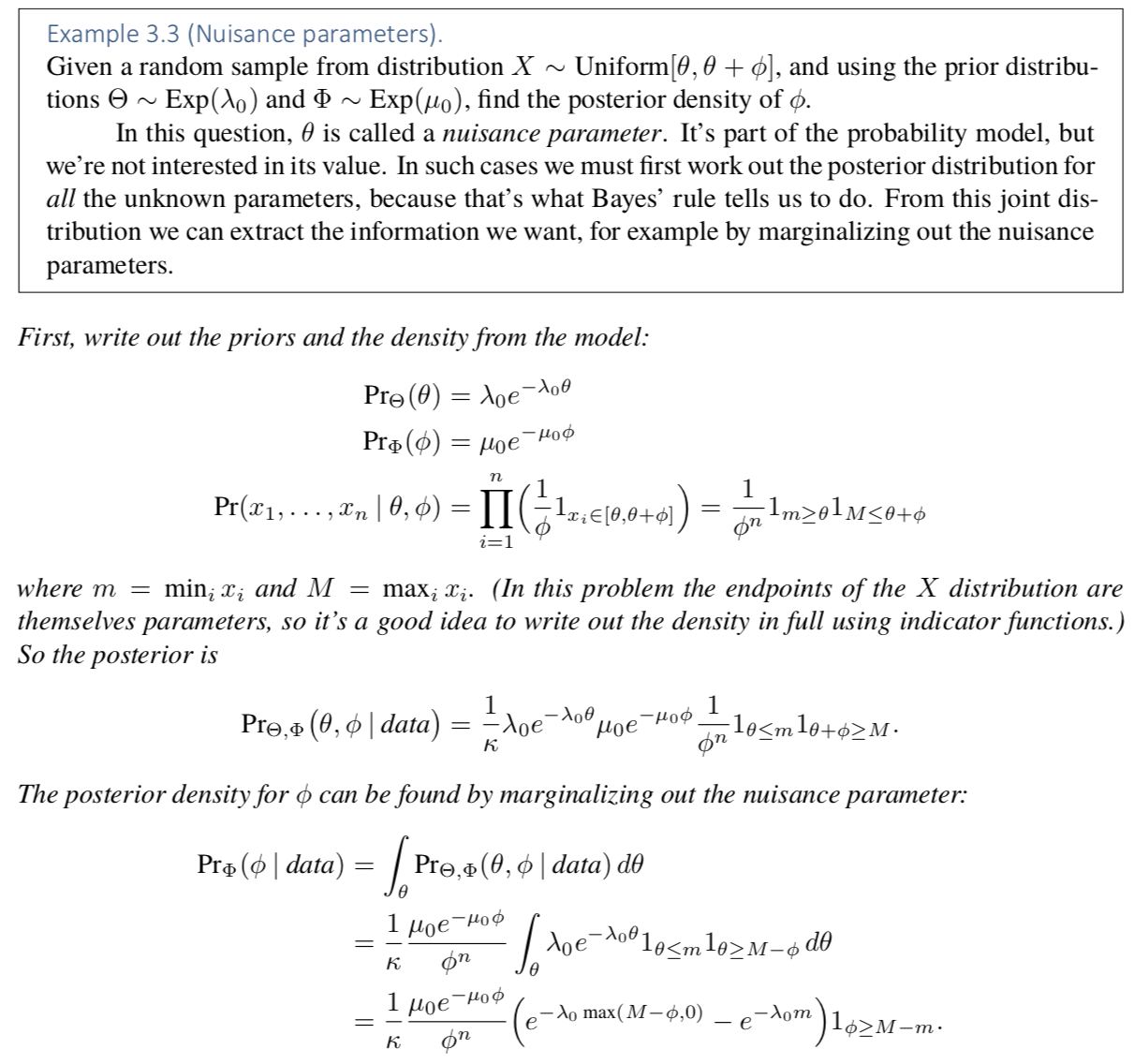
1. Report value of parameter that maximises the mode of the posterior distribution. **This is the maximum a posteriori (MAP) estimate**
2. Return mean or median of the posterior distribution
3. Report a loss function, which measures the price you pay if you report the estimate and the true value. Report Θ’ which minimises the expected posterior loss.

**Posterior Predictive Probability:** Probability that the next event will be something. Can use the law of total probability. For example, for the example of a coin flip – looking for probability next flip X’ will be a head

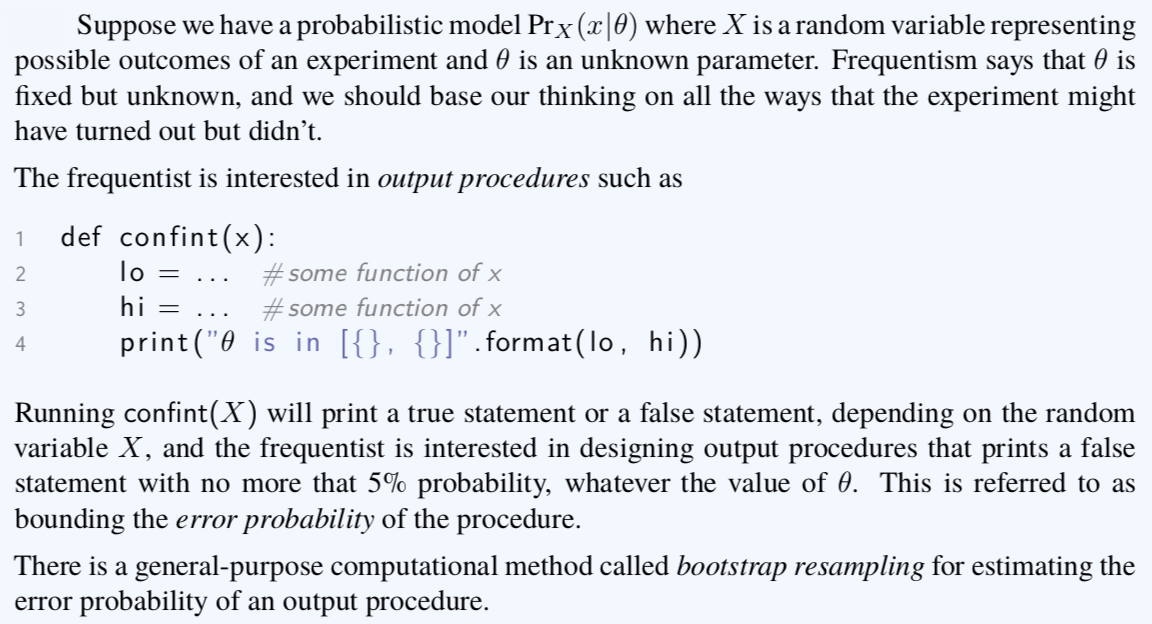


The Python for this is: numpy.mean(Θsample)

### Nuisance Parameters



## Frequentism



### Resampling

If the trial were run again, what is a good way to use the data at hand to synthesize a result I could plausibly see

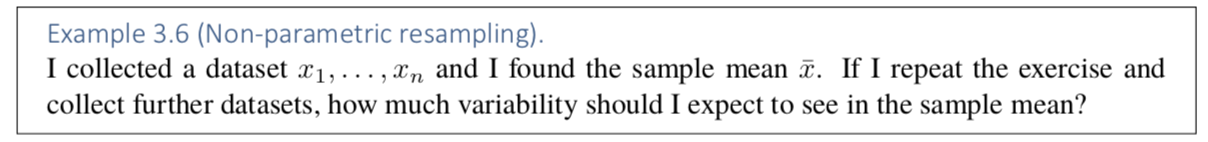
**Parametric:** (1) specify a probability model with unknown parameters, (2) fit the parameters using maximum likelihood (3) sample from the fitted probability

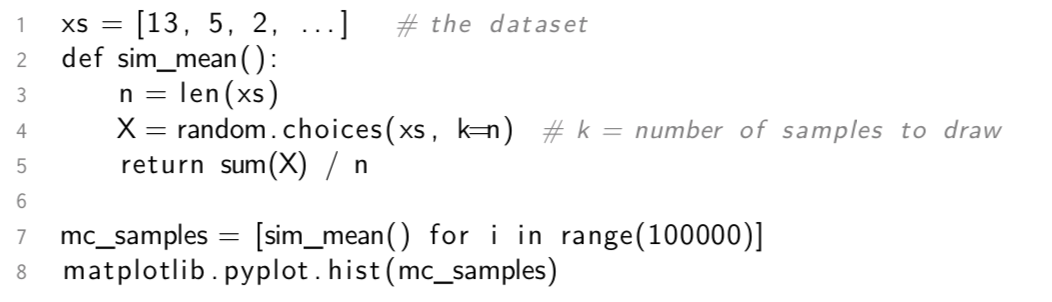
This relies on some sort of intuition – resample using intuition and parameters which have been found.

**The Bootstrap:** removes any need for maths or exhaustive optimization for bounding the error probability

1. Write probability you’re interested in
2. Replace any unknown parameters by the MLE given the data – replace any random variables by their resampled versions
3. Use the Monte Carlo method to estimate the probability of the expression in step 2

**Non-Parametric:** Sample from the empirical distribution, which is equivalent to picking a value at random from the observed dataset. The idea is that the best-fitting distribution is the dataset itself – can then do Monte Carlo integration or draw a histogram





## Model Selection

**Model:** explanation for the data – simpler models are likely to be closer to the truth and to generalise better to new scenarios. Can also be used to make predictions about new data.

### Hypothesis Testing and P-Values

Come up with a null hypothesis and see if you reject it or not.

1. Define a function test\_statistic(y) – can be any function of the data. We aim for a function that is likely to be small if the null hypothesis is true and large if it is false.
2. Assuming the null hypothesis is true, find the distribution of T = test\_statistic(Y) – through resampling
3. From the actual data – compute t = test\_statistic(y). Mark t on histogram and measure p = ℙ(T > t)
4. If p ≤ 5%, reject null hypothesis, otherwise don’t reject it

p is the p-value or significance level, and it measures the probability of seeing results as extreme as we can actually saw, assuming null hypothesis is true.

### Alternative Hypothesis

test\_statistic(Y) = lik(model = A | Y = y) / lik(model = B | Y = y) where the likelihood of a model is found by maximising over all parameters in the model.

This is the likelihood ratio test and allows us to decide between two different models – if model B is false, then the denominator will be small, so test statistic will be large.

# Creating a Model

Purposes of unknown parameters:

1. Make the model expressive
2. When we fit the model and inspect the estimated parameters, we learn which patterns are actually present in the dataset.

How to choose parameters:

1. Use parameters that correspond to the questions we want to ask and the quantities we want to measure – **identifiability**
2. Run maximum likelihood estimators, using a numerical optimisation library – **natural parameters**
   1. Should map the likelihood to between 0 and 1 through a/1+a – **logit and softwmax**

### Logistic Regression

Logistic – uses the logic transform for parameters

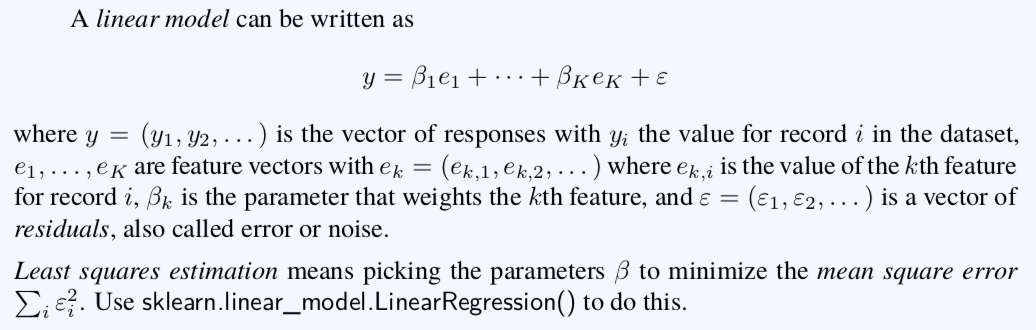
Regression – has a response variable predicted by covariates

# Feature Spaces

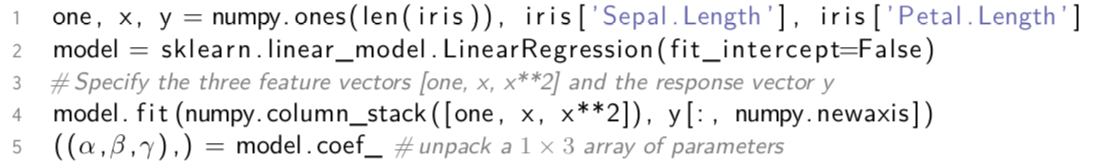
**Feature:** any measurable property of the objects being studied.

**Linear Model:** model with unknown parameters in which the parameters are weighted by features and combined linearly

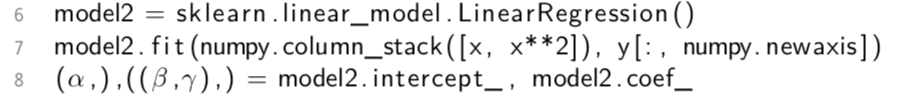
## Fitting a linear model



**Important Note:** Linear doesn’t mean straight line – simply means linear algebra (adding vectors), but can have **x2** as a vector

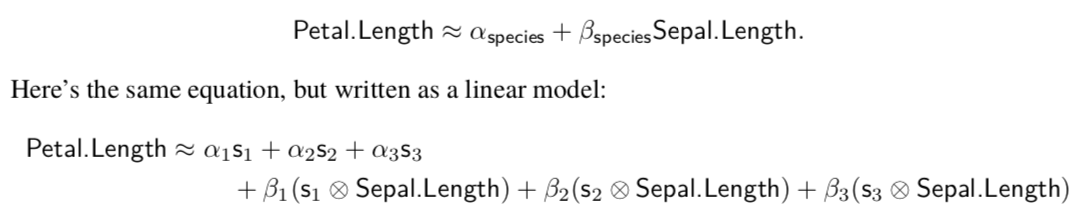


sklearn model fitting function includes a one vector unless we explicitly tell it otherwise with fit\_intercept=False. Can also write this as:



## Features

**One-Hot Encoding:** Turn an enum feature into a collection of binary features so it could be used in a linear model.



Each sk is a binary vector marking out which rows belong to the kth species – **one-hot coding** of the vector. ⊗ means elementwise multiplication

As you add more features, the better the fit and the smaller the residual we can achieve. But a model with too many features tends to be bad at generalizing to new data. Therefore, we want to be careful about this:

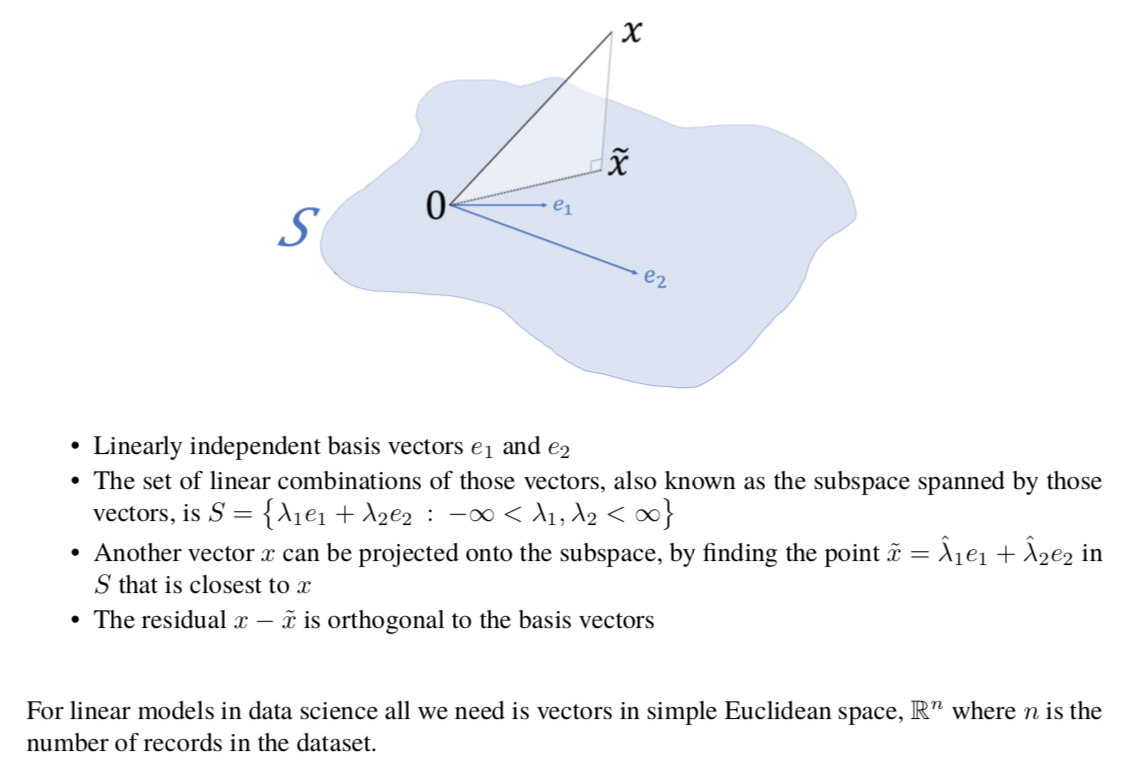
**Dimension Reduction:** General term for finding lower-dimensional representations

1. Start with long list of possible features {e1, e2, …, ek}
2. Construct two new features f1 and f2 each of which a linear combination of the raw features. The goal is to construct them to minimise the errors of:
   1. Y = a + B1f1 + B2f2
3. Procedure gives us two features f1 and f2 that capture as much information as they can about Y – **two-dimensional embedding of the dataset’s K dimensions**
4. Can show data on scatter plot of f1 vs f2 – likely to reveal useful clusters

**Feature Selection:**

1. Start with long list of possible features
2. Pick m, a number of features to use, and find the best fitting model subject to the constraint that it’s only allowed to use m of the possible features

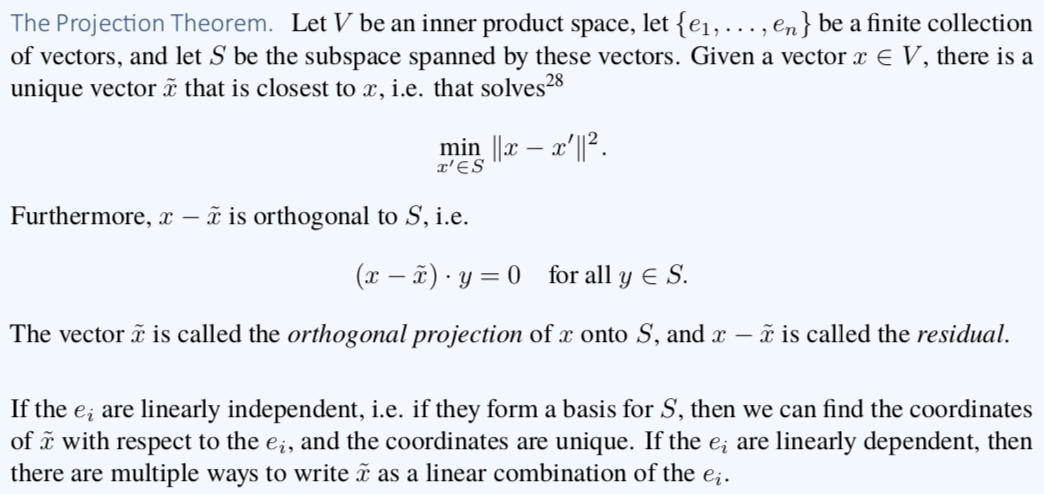
## Linear Mathematics



**Other Definitions:**

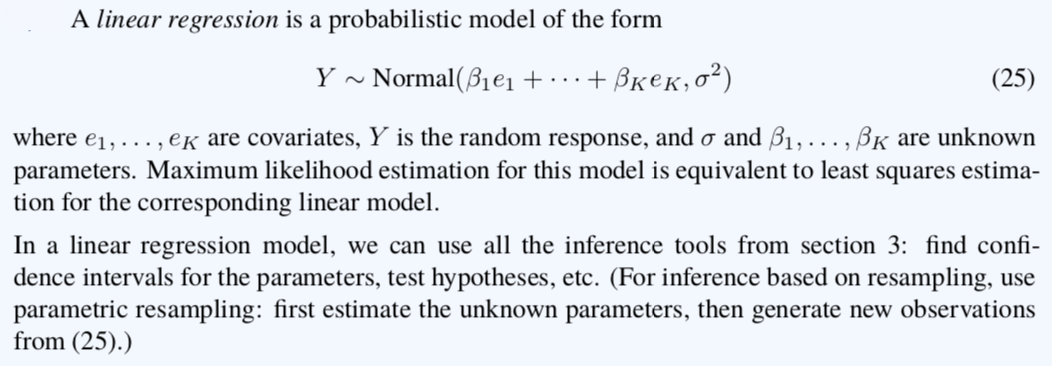
* V is the set whose elements are called vectors
* F is a field whose elements are called scalars – either reals or complex numbers
* Binary operation: V x V -> V is addition written: v + w
* Binary operation: F x V -> V is scalar multiplication, written: λv
* Binary operation: V x V -> F is inner product, written: v . w

### Orthogonal Projection



**Collinearity and matrix rank:** Stacked collection of vectors to form a matrix then the rank of the matrix is the dimension of the space spanned by those vectors. Use: numpy.linalg.matrix\_rank(numpy.column\_stack([e1, e2])))

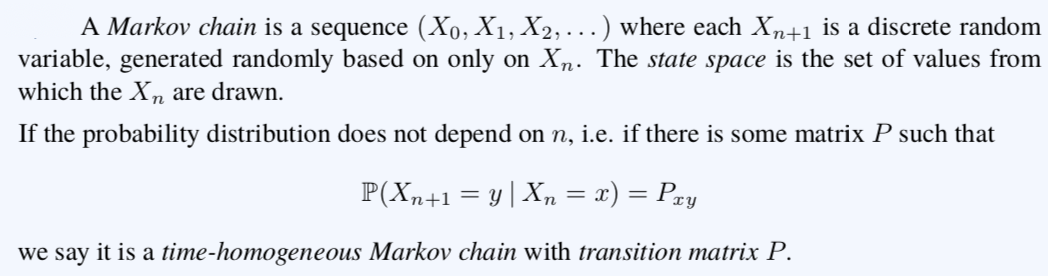
## Linear Regression and Least Squares



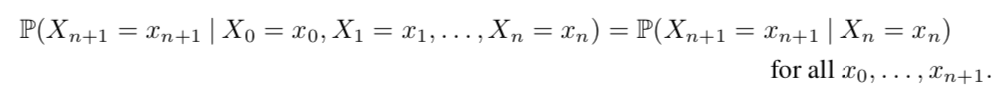
# Random Processes

**Random (Stochastic) Process:** Using probabilistic laws to describe how the system changes

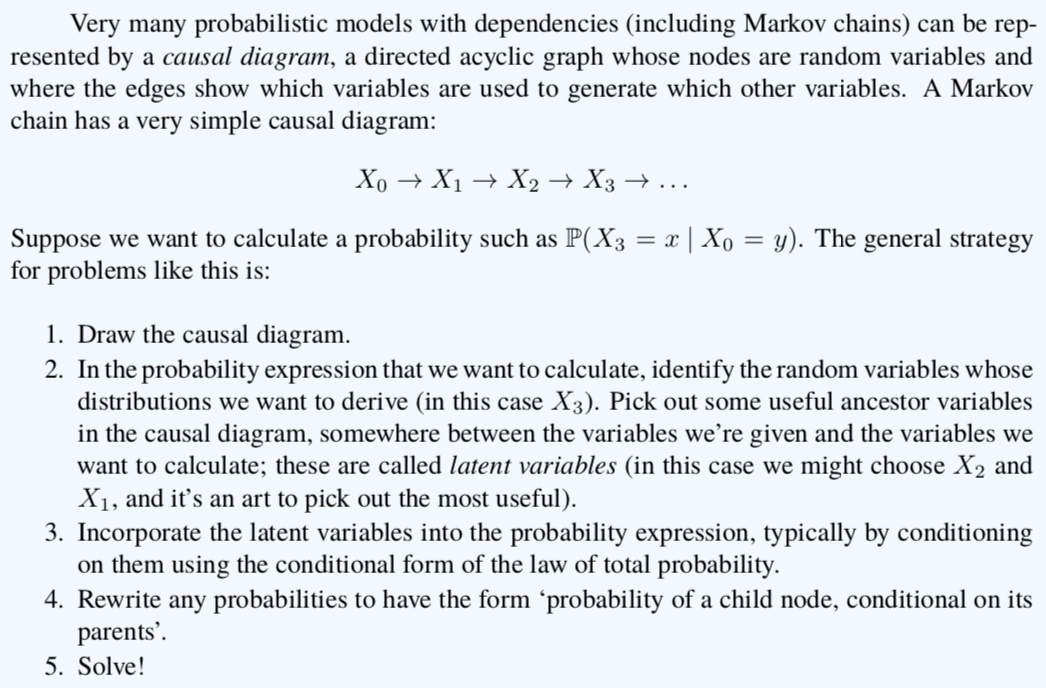
## Markov Chains



**Statement of Markov Assumption:**

****

This is a statement that whatever the history, all that matters for the purposes of deciding the next state is the current state – the is memorylessness



**Hitting Probability Theorem:** Let A be a subset of a Markov chain’s state space. The hitting probability from x is:

These hitting probabilities solve:

If the system of equations has multiple solution, the hitting probability from is the minimum of over all solutions

## Limit theorems and equilibrium

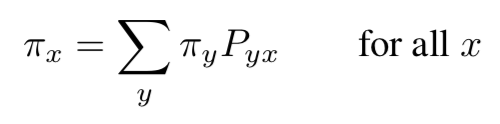
Aim: Look at the long-run average behaviour of a time-homogeneous Markov chain with a finite state space

**Stationary:** Markov chain is stationary if its distribution does not change over time, ie if there is a vector π such that ℙ(Xn = x) = πx for all n. Conversely, if π is a probability distribution st:



then π is called a stationary or equilibrium distribution. If π is a stationary distribution, and we pick the Markov chain’s initial state X0 randomly according to π, then X1 will have distribution π and so on. If we pick the initial state in some other way, this is not true

Stationary distribution π must satisfy:



where P is the transition matrix

**Theorem (Uniqueness of Stationary Distribution):** Consider a Markov chain with transition matrix P and a finite state space. The Markov chain is irreducible if it is possible to get from any state to any other – this implies that the there is a unique stationary distribution and it is the unique solution to:

### Detailed Balance

Can sometimes use a trick to help us find stationary distribution for a Markov chain with very little algebra – using Gibbs sampling. It also can be used to generate random variables from Bayesian posterior distribution.

**Theorem (detailed balance):** Let X be a Markov chain with transition matrix P. If there is a vector such that:

then solves .

**Theorem (ergodicity):** Let X be an irreducible Markov chain with stationary distribution . Then, the long-run average time spent in each state converges to . Therefore:

If Markov chain’s initial state X0 were chosen from , then Xn would have for every n, therefore , for all I, and so the equation would be true exactly.

### Limiting Behaviour and Aperiodicity

**Theorem:** Let X be a Markov chain – a state x is said to be aperiodic if there exists an n0 st for all . If the chain is irreducible and has an aperiodic state, then all its states are aperiodic and:

Can use this as a tool to generate a random variable from a distribution. Can create transition probabilities of a Markov chain to ensure it has stationary distribution as required. Then we can generate a random variable by starting the Markov chain in an arbitrary state and running it for a large number of steps and returning the state.