MASTER'S COURSE · LECTURE 3

Principles and Methods of Data Analysis in the Energy Industry

Lecture 3 — Probability Mass, Cumulative, and Density Functions

Information Technologies for Sustainable Energy Engineering

Sources: A. Downey, *Think Stats 2e*, Ch. 3–6 · A. Downey, *Elements of Data Science*, Ch. 8

ROADMAP

What we'll cover today

1

Part A

Probability mass functions (PMF)

Normalizing counts to probabilities, plotting PMFs to compare groups fairly, and the class-size paradox.

2

Part B

Cumulative distribution functions (CDF)

Percentiles, reading and comparing CDFs, percentile-based statistics, and generating random numbers.

3

Part C

Modeling distributions

Empirical vs. analytic distributions: exponential, normal, lognormal, and Pareto — and why (and when) to model at all.

4

Part D

Probability density functions (PDF)

Picking up from Lecture 2: histograms compared groups unfairly when sample sizes differ — PMFs are the fix, and everything else here builds on that idea.

PART A

Probability Mass Functions

Think Stats 2e, Chapter 3

From counts to probabilities: normalization

- A probability mass function (PMF) maps each value to its probability, rather than its raw count.
- A probability is a frequency expressed as a fraction of the sample size n. Dividing every frequency by n is called normalization.
- thinkstats2.Pmf works like Hist, but stores floating-point probabilities that sum to 1 instead of integer counts.
- Prob(x) (or the bracket operator) looks up a probability; Incr and Mult modify one; Normalize rescales so the total is 1 again after edits.

```
n = hist.Total()
d = {x: freq / n for x, freq in hist.Items()}

>>> pmf = thinkstats2.Pmf([1, 2, 2, 3, 5])
>>> pmf
Pmf({1: 0.2, 2: 0.4, 3: 0.2, 5: 0.2})
>>> pmf.Prob(2)
0.4
```

Plotting PMFs to compare groups fairly

- Lecture 2 ended on a problem: histograms of pregnancy length for 'first babies' vs. 'others' can't be compared bar-for-bar, because the groups have different sizes.
- Plotting the PMFs instead removes that distortion — both curves are on the same 0-to-1 probability scale regardless of group size.
- Result: first babies appear less likely to arrive exactly on time (week 39) and somewhat more likely to be a few days late (weeks 41-42).
- In this course's toolkit, empiricaldist.Pmf.from_seq(data) is the practical equivalent — normalize=True (default) gives probabilities; .bar() plots them.

```
first_pmf =
thinkstats2.Pmf(firsts.prglngth,
                                label='first')
other_pmf =
thinkstats2.Pmf(others.prglngth,
                                label='other')
thinkplot.PrePlot(2)
thinkplot.Pmfs([first_pmf, other_pmf])
```

Energy analogue: comparing fault-code PMFs for a 50-turbine and a 200-turbine fleet — directly comparable, unlike raw counts.

The class-size paradox

- A college's student-to-faculty ratio might be 10:1, yet students report an average class size well above 10. Why?
- The Dean's PMF weighs each class size equally; the students' experienced PMF is biased by class size itself — a student in a 40-person class is 40x more likely to report that size than a student in a 1-person tutorial.
- BiasPmf multiplies each probability by the value x itself, then renormalizes — turning the Dean's distribution into the one students actually observe.
- In this example, the true mean class size is 23.7; the student-observed mean is 29.1 — almost 25% higher.

```
def bias_pmf(pmf):
    new_pmf = pmf.copy()
    for x, p in pmf.items():
        new_pmf[x] *= x     # weight by x
    new_pmf.normalize()
    return new_pmf
```

Energy analogue: this is exactly the 'inspection paradox' behind biased load or downtime readings — see the callout below.

Energy application — the inspection paradox: if you sample outage duration by picking a random moment and asking “how long is the outage happening right now?”, long outages are overrepresented — exactly like the class-size bias. Sampling by event (not by moment) avoids the trap.

PART B

Cumulative Distribution Functions

Think Stats 2e, Chapter 4 · Elements of Data Science, Chapter 8

Percentiles and percentile rank

- PMFs work well for a small number of distinct values, but get noisy as the number of values grows — small random fluctuations start to dominate the picture.
- The percentile rank of a value is the fraction of the dataset less than or equal to it, expressed as a percentage. If you scored 88 out of {55, 66, 77, 88, 99}, your percentile rank is 100 x 4/5 = 80.
- Percentile (the inverse operation) takes a percentile rank and returns the corresponding value — e.g., the 50th percentile of that set is 77.
- This distinction — value to rank, or rank to value — is exactly the forward/inverse relationship the CDF formalizes next.

The CDF: mapping values to cumulative probability

- The cumulative distribution function (CDF) maps a value x to the fraction of the sample that is <= x — the same idea as percentile rank, expressed as a probability in [0, 1] rather than a percentage.
- For sample [1, 2, 2, 3, 5]: CDF(0)=0, CDF(1)=0.2, CDF(2)=0.6, CDF(3)=0.8, CDF(5)=1. The CDF of a sample is a step function.
- Reading a CDF: steep/vertical sections mark common values (the mode shows as a jump); flat sections mean few or no values in that range.
- empiricaldist.Cdf.from_seq(data) builds one directly; calling it like a function, cdf(q), returns the probability that a random value is <= q.

```
from empiricaldist import Cdf
cdf = Cdf.from_seq(live.prglngth)

>>> cdf(39)
0.53   # ~53% of pregnancies <= 39 wks

cdf.plot()
plt.xlabel('weeks'); plt.ylabel('CDF')
```

Percentile-based statistics: median, IQR, quantiles

- cdf.inverse(p) (or Percentile(p) in thinkstats2) goes from a probability back to a value — the inverse CDF.
- The median is Percentile(50) — a measure of central tendency, like the mean, but based on rank rather than magnitude.
- The interquartile range (IQR) — Percentile(75) minus Percentile(25) — measures spread, and is more robust to outliers than variance because it ignores the extreme tails.
- Splitting a CDF into 4 (quartiles), 5 (quintiles), or 10 (deciles) equally-spaced points are all quantiles — a general term for equally-spaced percentile ranks.

```
>>> cdf.inverse(0.5)      # median
39.0
>>> cdf.inverse(0.75) - cdf.inverse(0.25)
2.0   # IQR, in weeks
```

Comparing distributions: PMF vs. CDF

- Plotting two PMFs on the same axes (e.g., birth weight by group) is often too noisy to compare directly — small bin-to-bin fluctuations obscure real differences.
- The same comparison via CDFs is much smoother, because cumulative sums average out much of that noise.
- Reading a CDF comparison: if one CDF sits to the right of another, that group's values tend to be higher (e.g., first babies are slightly lighter throughout the birth-weight distribution, with the largest gap above the mean).
- General rule of thumb: use CDFs for exploration — they give the least distorted view. Switch to a PMF or bar chart only when presenting to an audience unfamiliar with CDFs and the variable has few unique values.

A bonus property: generating random numbers

- If you look up the percentile rank of a random sample drawn from a distribution, the resulting ranks are uniformly distributed between 0 and 100 — regardless of the original distribution's shape.
- This gives a simple, general algorithm for sampling from any CDF: (1) draw a probability p uniformly from [0, 1]; (2) return cdf.inverse(p).
- thinkstats2.Cdf.Random() implements exactly this; Cdf.Sample(n) returns n such values at once.
- This technique underlies Monte Carlo simulation of any empirical distribution — e.g., simulating a year of hourly demand draws from an observed load-duration curve.

PART C

Modeling Distributions

Think Stats 2e, Chapter 5

Empirical vs. analytic distributions

- An empirical distribution is based on observed data — necessarily a finite sample, with all its noise and idiosyncrasies.
- An analytic distribution is characterized by a CDF that is a mathematical function — it can serve as a model, a simplification that leaves out unneeded detail.
- A model is useful when a small set of parameters (e.g., just μ and sigma for a normal distribution) summarizes a large dataset well enough for the purpose at hand.
- This section surveys four widely used analytic distributions and the visual tests used to check whether one fits a real dataset.

The exponential distribution: time between events

- CDF(x) = 1 - e^(-lambda x). Exponential distributions arise from interarrival times — the gaps between events that are equally likely to happen at any moment.
- Visual test: plot the complementary CDF, 1 - CDF(x), on a log-y scale. For exponential data, log(CCDF) is a straight line with slope -lambda.
- lambda is a rate: events per unit time. The mean interarrival time is 1/lambda.
- Example from the text: 44 births in 24 hours gives lambda ~ 0.0306/min, so a mean of 32.7 minutes between births — and the real data is close to, but not perfectly, exponential.

```
cdf = Cdf.from_seq(interarrival_times)
cdf.plot(complement=True)
plt.yscale('log')
plt.xlabel('minutes'); plt.ylabel('CCDF')
```

Energy application: time between grid faults, forced outages, or lightning strikes on a line is classically modeled as exponential.

The normal distribution

- The normal (Gaussian) distribution is characterized by two parameters: mean μ and standard deviation sigma. μ=0, sigma=1 is the standard normal distribution.
- scipy.stats.norm provides a fast, accurate CDF (and PDF) — no closed-form solution exists, but scipy evaluates it numerically.
- To test the fit, plot the empirical CDF of real data against a normal CDF using the data's own mean and standard deviation as parameters.
- For NSFG birth weight, a normal model with μ=7.28, sigma=1.24 fits well overall, but under the 10th percentile there are more light babies than the model predicts — a reminder that a good overall fit can still miss what matters for a specific question (here, preterm births).

```
import scipy.stats
>>> scipy.stats.norm.cdf(0)
0.5

def eval_normal_cdf(x, mu=0, sigma=1):
    return scipy.stats.norm.cdf(
        x, loc=mu, scale=sigma)
```

Testing normality: the normal probability plot

- There's no simple straight-line transform for the normal CDF (unlike the exponential's log trick), so Think Stats uses a different visual test.
- Method: (1) sort the sample; (2) draw an equal-size random sample from the standard normal distribution and sort it; (3) plot the sorted sample against the sorted random values.
- If the sample is approximately normal, the result is a straight line with intercept mu and slope sigma.
- For NSFG birth weights, the plot is straight near the mean but deviates in the tails — the heaviest babies are heavier, and the lightest lighter, than a normal model expects. Restricting to full-term births removes some of that lower-tail discrepancy.

The lognormal distribution

- If log(x) is normally distributed, x is lognormal: CDF_lognormal(x) = CDF_normal(log x). Its parameters μ, sigma are not the mean and standard deviation of x itself.
- A lognormal sample plotted on a log-x axis has the classic sigmoid CDF shape of a normal distribution; a normal probability plot of log(x) tests the fit directly.
- Text example: adult body weight (BRFSS survey, ~400,000 respondents) is a poor fit to a normal model but a good fit to a lognormal one.
- Quantities that arise from many small multiplicative effects (rather than additive ones) tend toward lognormal — a useful rule of thumb when picking a first model to try.

Energy application: household electricity consumption and solar-project capacity are frequently closer to lognormal than normal — always check on a log-x axis before assuming symmetry.

The Pareto distribution: heavy tails

- CDF(x) = 1 - (x / x_m)^(-alpha), where x_m is the minimum possible value. Named for Vilfredo Pareto, who used it to describe wealth distribution.
- Visual test: on a log-log scale, the complementary CDF (CCDF) of Pareto-distributed data looks like a straight line with slope -alpha.
- Classic examples: city and town population sizes, earthquake magnitudes, forest-fire sizes — phenomena with a small number of extreme values that dominate the total.
- A Pareto model often applies only to the extreme tail (e.g., the largest 1% of cities); a lognormal model may fit the rest of the distribution better — which model is 'right' depends on which part of the distribution matters for your question.

Energy application: outage sizes (customers affected), wind-gust extremes, and wildfire-related de-energization events are classic heavy-tailed, Pareto-like phenomena in grid risk analysis.

Why model? — and why be careful

- Models smooth out measurement error and sample-specific idiosyncrasies that aren't relevant to the underlying phenomenon.
- A good-fitting model is a form of data compression: a handful of parameters (e.g., μ, sigma, or lambda) can stand in for an entire large dataset.
- Sometimes a model's fit hints at the generative process — Pareto distributions, for instance, often arise from positive-feedback ('preferential attachment') processes.
- But no real dataset fits an analytic distribution perfectly. Saying 'income is lognormal' is a useful simplification, not a literal claim — whether a model is 'good enough' always depends on what you plan to use it for.

PART D

Probability Density Functions

Think Stats 2e, Chapter 6 · Elements of Data Science, Chapter 8

The PDF: density, not probability

- The probability density function (PDF) is the derivative of a continuous CDF. Equivalently, the CDF is the integral of the PDF.
- Evaluating a PDF at one point does NOT give a probability — it gives a probability density, probability per unit of x. To get an actual probability, you must integrate over a range of x.
- Analogy: density in physics is mass per unit volume; you need to integrate (or multiply by volume) to get an actual mass. Probability density works the same way.
- thinkstats2.NormalPdf(mu, sigma).Density(x) evaluates the normal density function at x — useful for plotting the characteristic bell-curve shape, not for reading off probabilities directly.

The distribution framework: how PMF, CDF, and PDF relate

PMF → CDF	Cumulative sum of probabilities (“add up the masses”).
CDF → PMF	Differences between consecutive cumulative probabilities.
PDF → CDF	Integrate the density over x (continuous analogue of summing).
CDF → PDF	Differentiate the CDF (continuous analogue of differencing).
Discrete → continuous	Smoothing: assume an analytic distribution, or use KDE.
Continuous → discrete	Discretizing/quantizing: evaluate the PDF at discrete points.

Four representations of the same underlying distribution — choose whichever is most useful for the task: PMF for discrete counts, CDF for comparison, PDF/KDE for a smooth visual model.

Kernel density estimation (KDE)

- KDE is an algorithm that takes a sample and estimates a smooth PDF that fits it — a continuous counterpart to a histogram.
- Plotting a sample's raw PMF against a theoretical PDF fails: in a continuous sample, every value tends to be unique, so the PMF is a flat, uninformative line at probability 1/n.
- scipy.stats.gaussian_kde (wrapped by thinkstats2.EstimatedPdf) computes the estimate; Seaborn's sns.kdeplot() is the quickest way to plot one directly from a sample.
- KDE is useful for visualization (a smoother presentation than a raw histogram), interpolation (estimating density between observed points), and simulation (smoothing a small sample before resampling from it).

```
import seaborn as sns
sns.kdeplot(sample, label='estimated PDF')

xs = np.linspace(6, 14)
ys = norm(10, 1).pdf(xs)
plt.plot(xs, ys, color='gray',
         label='normal PDF')
```

Moments and skewness

- A raw moment of order k is the mean of x^k; the first raw moment (k=1) is just the ordinary mean. A central moment is based on deviations from the mean; the second central moment (k=2) is variance.
- Skewness describes asymmetry: negative skew means a longer left tail, positive skew a longer right tail — it says nothing about bias in how the sample was collected.
- Sample skewness (a standardized third moment) is sensitive to outliers. Pearson's median skewness, gp = 3(mean - median) / std, is more robust because it doesn't depend on a cubed term.
- NSFG birth weight: mean (7.27 lb) is slightly below the median (7.38 lb) — consistent with the left skew already visible in the histogram back in Lecture 2.

WRAP-UP

What you should walk away with

- PMFs fix the sample-size comparison problem left open in Lecture 2 — and the class-size paradox shows how a biased sampling process distorts an observed PMF.
- CDFs give the clearest exploratory view of a distribution: percentiles, IQR, group comparisons, and even a way to generate random numbers.
- Four analytic models — exponential, normal, lognormal, Pareto — cover a large share of real-world shapes, each with its own visual fit test.
- PDFs and KDE round out the toolkit: density (not probability) at a point, and a smooth, data-driven alternative to a histogram.

Coming up

Lab 3: build PMFs and CDFs for an energy dataset, and fit/test an analytic model against it

Lecture 4: Relationships — scatter plots, correlation, and simple regression

Required reading and sources for this lecture

- Allen B. Downey, Think Stats: Exploratory Data Analysis in Python, 2nd ed. — Chapters 3–6 (“Probability mass functions,” “Cumulative distribution functions,” “Modeling distributions,” “Probability density functions”). Free at greenteapress.com/thinkstats2
- Allen B. Downey, Elements of Data Science — Chapter 8 (“Distributions”), for the empiricaldist/Seaborn toolchain. Free at allendowney.github.io/ElementsOfDataScience
- Both texts are released under Creative Commons licenses (CC BY-NC-SA) — full attribution in the course syllabus.

Before Lab 3:

Install empiricaldist and seaborn (pip install empiricaldist seaborn). Skim Think Stats Chapters 3–6 and try computing a PMF and CDF for a dataset of your choice.