PRINCIPLES AND METHODS OF DATA ANALYSIS IN THE ENERGY INDUSTRY

Lecture 3 — Lecture Notes (Conspectus)

Probability Mass, Cumulative, and Density Functions

Programme: Information Technologies for Sustainable Energy Engineering (Master's level)

Sources: A. B. Downey, *Think Stats: Exploratory Data Analysis in Python*, 2nd ed., Chapters 3–6 (greenteapress.com/thinkstats2) · A. B. Downey, *Elements of Data Science*, Chapter 8 (allendowney.github.io/ElementsOfDataScience). Both texts are distributed under CC BY-NC-SA.

Learning objectives

By the end of this lecture, students will be able to:

- Normalize a histogram into a probability mass function (PMF), and explain why PMFs — not raw histograms — are the fair way to compare groups of different sizes.
- Explain the class-size paradox / inspection paradox, and recognize the same bias in an energy-data sampling process.
- Compute and read a cumulative distribution function (CDF): percentiles, the median, the interquartile range, and quantiles.
- Explain the difference between an empirical and an analytic (theoretical) distribution, and identify which of four common analytic distributions (exponential, normal, lognormal, Pareto) best matches a given phenomenon and why.
- Explain what a probability density function (PDF) represents, how PMF/CDF/PDF relate to one another, and use kernel density estimation (KDE) to visualize a smooth distribution from a sample.

Picking up from Lecture 2: histograms could not fairly compare groups of different sample sizes. This lecture resolves that problem with PMFs, then builds CDFs, analytic models, and PDFs on top of that foundation.

Part A — Probability Mass Functions

Based on: A. Downey, *Think Stats 2e*, Chapter 3, “Probability mass functions.”

A.1 From counts to probabilities: normalization

A **probability mass function (PMF)** maps each value in a distribution to its probability, rather than to 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 exactly like the `Hist` class from Lecture 2, except it stores floating-point probabilities that sum to 1 instead of integer counts.

```
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)      # or pmf[2]
0.4
```

A Pmf can be modified with `Incr` (add to a probability) or `Mult` (scale a probability); after modification the total may no longer be 1, so `Normalize()` rescales it back to a proper distribution.

A.2 Plotting PMFs to compare groups fairly

Lecture 2 ended on an unresolved problem: histograms of pregnancy length for "first babies" versus "others" could not be compared bar-for-bar, because the two groups have different sample sizes. Plotting the PMFs instead removes that distortion, since both curves sit on the same 0-to-1 probability scale regardless of group size.

```
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])
```

Once plotted this way, first babies appear less likely to arrive exactly on time (week 39) and somewhat more likely to arrive a few days late (weeks 41-42) — a real, if small, pattern that the raw histograms in Lecture 2 could not reveal cleanly.

In this course's practical toolkit, `empiricaldist.Pmf.from_seq(data)` is the equivalent tool: `normalize=True` (the default) produces probabilities; `.bar()` plots them as a bar chart.

> **Energy application**
>
> Comparing fault-code frequency across a 50-turbine and a 200-turbine fleet: PMFs make the two fleets directly comparable, where raw counts would simply reflect fleet size.

A.3 The class-size paradox

A college's student-to-faculty ratio might be 10:1, yet students consistently report an average class size well above 10. The explanation has two parts: students take several classes per semester while professors teach few, and — more subtly — a student who happens to be in a large class is one of many students reporting that size, while a student in a small class is one of few. The distribution students actually experience is biased by class size itself.

```
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
```

In the textbook's example, the Dean's (true) mean class size is 23.7; the mean class size as experienced and reported by students is 29.1 — almost 25% higher. The same operation can be inverted (dividing by x instead of multiplying) to recover an unbiased distribution from a biased sample, which is useful when only the biased view is directly observable.

Energy application — the inspection paradox

If outage duration is sampled by picking a random moment in time and asking "how long is the outage happening right now?", long outages are systematically overrepresented — exactly the class-size bias. Sampling by event (each outage counted once, regardless of duration), not by moment, avoids the trap.

Part B — Cumulative Distribution Functions

Based on: A. Downey, *Think Stats 2e*, Chapter 4, "Cumulative distribution functions," and A. Downey, *Elements of Data Science*, Chapter 8.

B.1 Percentiles and percentile rank

PMFs work well for a small number of distinct values, but become 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 × 4/5 = 80. **Percentile** is the inverse: given a percentile rank, it returns the corresponding value (the 50th percentile of that set is 77).

B.2 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 the 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 or vertical sections mark common values (the mode appears as a jump); flat sections mean few or no values fall 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 from the sample is ≤ q.

```
from empiricaldist import Cdf
cdf = Cdf.from_seq(live.prglngth)
>>> cdf(39)
0.53    # ~53% of pregnancies are 39 weeks or shorter
cdf.plot(); plt.xlabel('weeks'); plt.ylabel('CDF')
```

B.3 Percentile-based statistics: median, IQR, quantiles

`cdf.inverse(p)` (called `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 based on rank rather than magnitude. The **interquartile range (IQR)** — `Percentile(75) -`

`Percentile(25)` — measures spread and is more robust to outliers than variance, because it ignores the extreme tails entirely.

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

Splitting a CDF into 4 equally spaced points gives quartiles; into 5, quintiles; into 10, deciles. Statistics based on equally spaced percentile ranks, in general, are called quantiles — commonly used to summarize skewed distributions like income.

B.4 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, since 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 — for example, first babies are slightly lighter throughout the birth-weight distribution, with the largest gap appearing above the mean.

> **General rule of thumb**
>
> Use CDFs for exploration — they give the least distorted view of a distribution's shape. Switch to a PMF or an ordinary bar chart only when presenting to an audience unfamiliar with CDFs and the variable has few unique values.

B.5 A bonus property: generating random numbers

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

Part C — Modeling Distributions

Based on: A. Downey, *Think Stats 2e*, Chapter 5, “Modeling distributions.”

C.1 Empirical vs. analytic distributions

An **empirical distribution** is based on observed data — necessarily a finite sample, carrying all of that sample's 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 earns its keep when a small set of parameters (e.g., just μ and σ for a normal distribution) summarizes a large dataset well enough for the purpose at hand.

C.2 The exponential distribution: time between events

CDF(x) = 1 − e^(−λx). Exponential distributions arise from **interarrival times** — the gaps between events that are equally likely to occur at any moment. **Visual test:** plot the complementary CDF, 1 − CDF(x), on a log-y scale; for exponential data, the result is a straight line with slope −λ. Here λ is a rate (events per unit time), and the mean interarrival time is 1/λ.

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

Textbook example: 44 births in 24 hours gives λ ≈ 0.0306/min, or a mean of 32.7 minutes between births — the real data is close to, but not perfectly, exponential (the underlying assumption that births are equally likely at any time of day is only approximately true). *Energy application:* time between grid faults, forced outages, or lightning strikes on a line is classically modeled as exponential.

C.3 The normal distribution

The **normal (Gaussian) distribution** is characterized by two parameters, mean μ and standard deviation σ; μ=0, σ=1 is the **standard normal distribution**. No closed-form solution exists for its CDF, but `scipy.stats.norm` evaluates it numerically and efficiently.

```
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)
```

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, σ=1.24 fits well overall, but below the 10th percentile there are more light babies than the model predicts — a reminder that a good overall fit can still miss the part of the distribution that matters most for a specific question (here, preterm births).

C.4 Testing normality: the normal probability plot

There is no simple straight-line transform for the normal CDF (unlike the exponential's log trick), so Think Stats uses a different visual test: (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 μ and slope σ.

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 the sample to full-term births removes some of that lower-tail discrepancy, since it excludes some of the lightest (preterm) weights.

C.5 The lognormal distribution

If log(x) is normally distributed, x is **lognormal**: CDF_lognormal(x) = CDF_normal(log x). Its parameters μ, σ are **not** the mean and standard deviation of x itself. A lognormal sample plotted on a log-x axis shows the familiar sigmoid CDF shape of a normal distribution; a normal probability plot of log(x) tests the fit directly.

Textbook example: adult body weight (BRFSS survey, roughly 400,000 respondents) is a poor fit to a normal model but a good fit to a lognormal one. As a rule of thumb, quantities arising from many small multiplicative effects (rather than additive ones) tend toward lognormal — a useful first guess when choosing a model.

> **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 a symmetric, normal shape.

C.6 The Pareto distribution: heavy tails

CDF(x) = 1 − (x / x_m)^(−α), where x_m is the minimum possible value. Named for Vilfredo Pareto, who used it to describe wealth distribution; since then it has described city sizes, earthquake magnitudes, and forest-fire sizes — phenomena with a small number of extreme values that dominate the total. **Visual test:** on a log-log scale, the complementary CDF (CCDF) of Pareto-distributed data looks like a straight line with slope −α.

A Pareto model often applies only to the extreme tail (e.g., the largest 1% of cities), while a lognormal model may fit the rest of the distribution better — which model is "right" depends on which part of the distribution matters for the question being asked.

> **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.

C.7 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 also a form of data compression: a handful of parameters can stand in for an entire large dataset. Sometimes a model's fit even hints at the generative process behind the data — 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 about how the data were generated. Whether a model is "good enough" always depends on what it will be used for.

Part D — Probability Density Functions

Based on: A. Downey, *Think Stats 2e*, Chapter 6, "Probability density functions," and A. Downey, *Elements of Data Science*, Chapter 8.

D.1 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, the density must be integrated over a range of x, just as physical density must be integrated over volume to get a mass.

```
class NormalPdf(Pdf):
    def Density(self, xs):
        return scipy.stats.norm.pdf(xs, self.mu, self.sigma)

>>> pdf = thinkstats2.NormalPdf(mean, std)
>>> pdf.Density(mean + std)
0.0333   # a density, not a probability
```

D.2 The distribution framework: how PMF, CDF, and PDF relate

Operation	What it does
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.

These are four representations of the same underlying distribution. Choose whichever is most useful for the task at hand: a PMF for a small number of discrete counts, a CDF for careful comparison, and a PDF or KDE for a smooth visual model.

D.3 Kernel density estimation (KDE)

Kernel density estimation (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 does not work: in a continuous sample every value tends to be unique, so the empirical 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.

```
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')
```

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, so a simulation explores more than just the exact observed values).

D.4 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 — it purely describes shape. **Sample skewness** (a standardized third moment) is sensitive to outliers; **Pearson's median skewness**, gp = 3(mean − median) / std, is more robust because it does not depend on a cubed term.

For NSFG birth weight, the mean (7.27 lb) is slightly below the median (7.38 lb) — consistent with the left skew already visible in the histogram from Lecture 2, and confirmed by both skewness statistics being negative.

Summary and preparation for the next session

Key terms introduced in this lecture

probability mass function (PMF) · normalization · class-size paradox · inspection paradox · percentile · percentile rank · cumulative distribution function (CDF) · median · interquartile range (IQR) · quantile · robust statistic · empirical distribution · analytic distribution · model · exponential distribution · interarrival time · complementary CDF (CCDF) · normal (Gaussian) distribution · standard normal distribution · normal probability plot · lognormal distribution · Pareto distribution · heavy tail · probability density function (PDF) · probability density · kernel density estimation (KDE) · raw moment · central moment · skewness · Pearson's median skewness

What you should be able to do after this lecture

- Normalize a histogram into a PMF, and explain why that fixes the group-comparison problem from Lecture 2.
- Recognize the class-size / inspection paradox in a new context, including energy sampling scenarios.

- Compute and read percentiles, the median, and the IQR from a CDF, and explain when to prefer a CDF over a PMF.
- Choose a plausible analytic model (exponential, normal, lognormal, or Pareto) for a described phenomenon, and state the visual test that would check the fit.
- Explain the difference between a PDF and a PMF, and use KDE to visualize a smooth distribution from sample data.

Preparation for 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 — Lab 3 builds directly on this, using an energy dataset in place of the NSFG/BRFSS examples.

Preview of Lecture 4

Relationships — scatter plots, the correlation coefficient, and simple regression (Think Stats 2e, later chapters; Elements of Data Science, Ch. 9–10).

References

- Downey, A. B. *Think Stats: Exploratory Data Analysis in Python*, 2nd ed. Green Tea Press. https://greenteapress.com/thinkstats2/html/index.html
- Downey, A. B. *Elements of Data Science*. https://allendowney.github.io/ElementsOfDataScience/