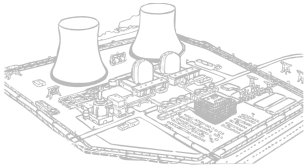
RISK ENGINEERING



Statistics and risk modelling using Python

Eric Marsden

<eric.marsden@risk-engineering.org>

Statistics is the science of learning from experience, particularly experience that arrives a little bit at a time.
— B. Efron, Stanford

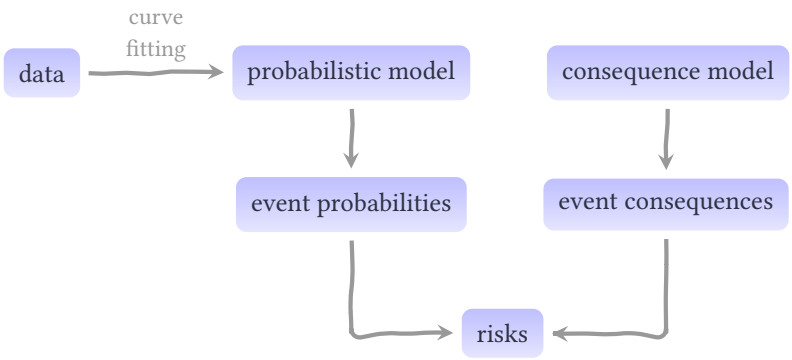
Learning objectives

Using Python/SciPy tools:

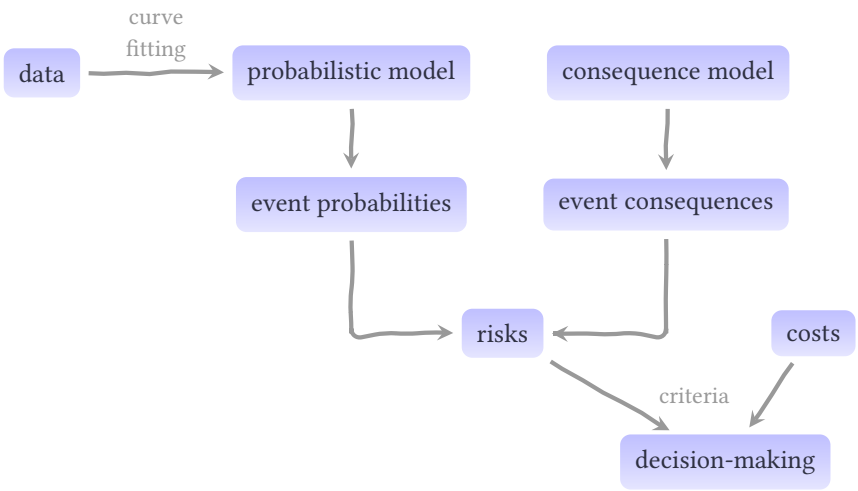
1. Analyze data using descriptive statistics and graphical tools
2. Fit a probability distribution to data (estimate distribution parameters)
3. Express various risk measures as statistical tests
4. Determine quantile measures of various risk metrics
5. Build flexible models to allow estimation of quantities of interest and associated uncertainty measures
6. Select appropriate distributions of random variables/vectors for stochastic phenomena

RISK ENGINEERING

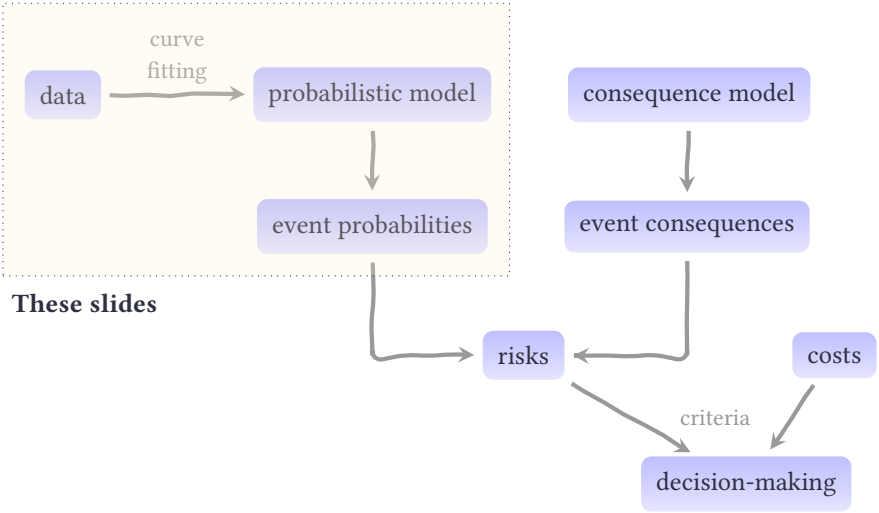
Where does this fit into risk engineering?



Where does this fit into risk engineering?



Where does this fit into risk engineering?



Angle of attack: computational approach to statistics

- Emphasize practical results rather than formulæ and proofs
- Include new statistical tools which have become practical thanks to power of modern computers
  - "resampling" methods, "Monte Carlo" methods
- Our target: "Analyze risk-related data using computers"
- If talking to a recruiter, use the term **data science**
  - very sought-after skill in 2019!

RISK ENGINEERING

ARTWORK: TAHAR COHEN, ANDREW J BUBOLTZ, 2011, SILK SCR
ON A PAGE FROM A HIGH SCHOOL YEARBOOK, 8.5" X 12"

DATA

Data Scientist: The Sexiest Job of the 21st Century

by Thomas H. Davenport and D.J. Patil

FROM THE OCTOBER 2012 ISSUE

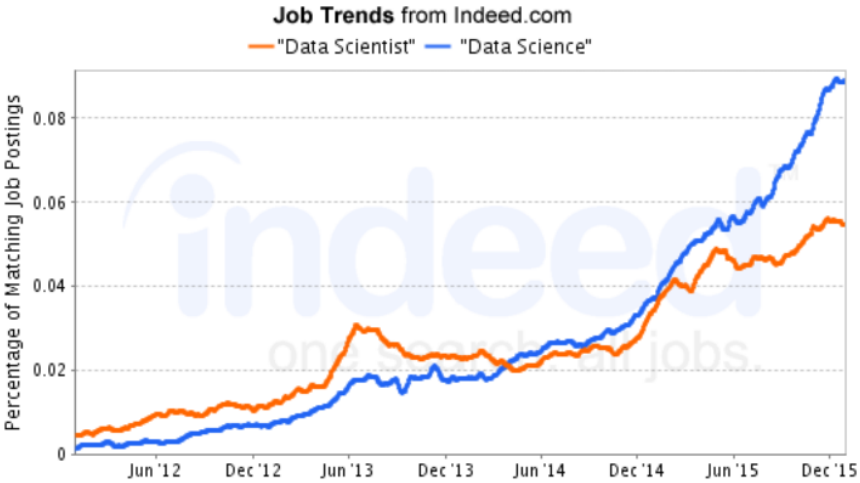
WHAT TO READ NEXT

Big Data: The Management Revolution

5 Essential Principles for Understanding Analytics

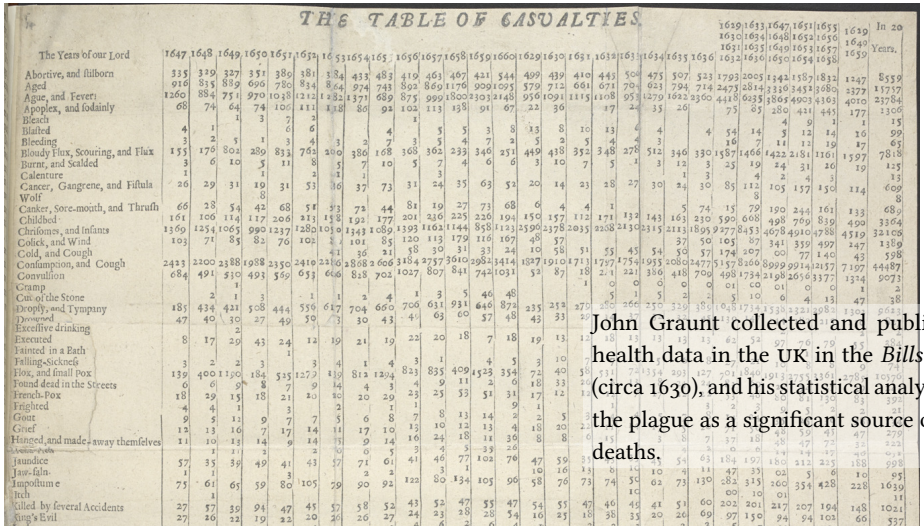
Data Scientists Don't Scale

A sought-after skill



Source: indeed.com/jobtrends

A long history



THE TABLE OF CASUALTIES

The Years of our Lord	1647	1648	1649	1650	1651	1652	1653	1654	1655	1656	1657	1658	1659	1660	1629	1630	1631	1632	1633	1634	1635	1636	1629 1630 1631 1632	1633 1634 1635 1636	1647 1648 1649 1650	1651 1652 1653 1654	1655 1656 1657 1658	1629 1630 1631	In 20 Years.
Abortive, and Stillborn	335	329	327	351	389	381	384	433	483	419	463	467	421	544	499	439	410	443	500	475	507	523	1793	2005	1442	1587	1832	1247	8559
Aged	916	835	889	696	780	834	864	974	743	892	869	1176	909	1095	579	712	661	671	704	623	794	714	2473	2814	3339	3452	3680	3377	13757
Ague, and Fever	1260	884	751	970	1038	1212	1282	1371	689	875	990	1800	2303	2148	956	1091	1115	1108	953	1279	1622	2300	4418	6235	3865	4903	4303	4010	23784
Apoplex, and sodainly	68	74	64	74	106	111	118	86	92	102	113	138	91	67	22	36		27	22	33	26		75	63	280	421	445	177	1306
Bleach			1	3	7	2				1																4	9	1	1	15
Blasted	4	1			6	6			4		5	5	3	8	13	8	10	13	6	4		4	54	14	5	12	14	16	99
Bleeding	3	2	5	1	3	4	3	2	7	3	5	4	7	2	5	2	5	4	2	3			7	6	11	12	19	17	65
Bloody Flux, Scouring, and Flux	155	179	802	289	833	782	300	386	168	368	362	233	346	251	449	438	352	348	278	512	346	330	1587	1466	1422	2181	1161	1597	7818
Burnt, and Scalded	3	6	10	5	11	8	5	7	10	5	7	4	6	6	3	10	7	5	4	3	12	3	25	19	24	31	20	19	125
Calenture				1		2	1				3										1	3	4	2	4	3			13
Cancer, Gangrene, and Fistula	26	29	31	19	31	53	16	37	73	31	24	35	63	52	20	14	23	28	27	30	24	30	85	112	105	157	150	114	609
Wolf				8																				8					8
Canker, Sore-mouth, and Thrush	66	28	54	42	68	51	53	72	44	81	19	27	73	68	6	4	4	1		5	74	15	79	190	244	161		133	689
Childbed	161	106	114	117	206	215	158	192	177	201	236	225	226	194	150	157	112	171	132	143	163	230	598	498	769	839	490	490	3264
Chrisoms, and Infants	1369	1254	1065	990	1237	1280	1030	1343	1089	1393	1162	1144	858	1123	2590	2378	2035	2208	2130	2115	2113	1895	2777	8453	4678	4910	4788	4519	32108
Colick, and Wind	103	71	85	82	76	102	2	101	85	120	113	179	116	167	48	57					37	50	105	87	341	359	497	247	1389
Cold, and Cough						41		36	21	58	30	31	33	24	10	58	51	55	45	54	50	57	174	207	60	77	140	43	598
Consumption, and Cough	2421	2200	2388	1988	2350	2410	2216	2808	2606	3154	2757	3960	2982	3414	1927	1910	1711	1797	1754	1955	2080	2477	5159	8260	8999	9914	8157	7197	44487
Convulsion	684	491	530	493	569	653	666	828	702	1027	807	841	742	1031	52	87	18	2.1	221	386	418	706	498	1734	2198	2616	3377	1314	9073
Cramp			1														1	0	0	0	0	0	0	1	0	0	0	1	2
Cut of the Stone		2	1	3		1	1	2	4	1	3	5	46	48				5	1	5	2	2	5	10	6	4	13	47	38
Dropsy, and Tympany	185	434	421	508	444	556	617	704	660	706	631	931	646	872	235	253	270	262	266	255	329	385	1045	1714	1538	2321	2982	1350	5623
Drowned		47	40	30	27	49	50	3	30	43	4	63	60	57	48	43	33	29											304
Excessive drinking			2																										
Executed	8	17	29	43	24	12	19	21	19	22	20	18	7	18	19	13	12	18	13	13	13	13	62	52	97	76	79	45	294
Fainted in a Bath						1																							
Falling-Sickness	3	2	2	3			3	4	1	4	3	1		4	5	3	10	7											
Flux, and small Pox	139	400	1150	184	535	1270	415	812	1204	823	835	409	1523	354	72	40	38	331	721	114	201	127	491	1846	1013	1	8	9	74
Found dead in the Streets	0	6	5	8	7	9	14	4	3	4	9	11	2	6	18	33	26						1	3	2	1	1	1	7
French-Pox	18	29	15	18	21	20	20	20	29	23	25	52	51	31	17	12	11	1	1	1	1	1	1	1	1	1	1	1	5
Frighted	4	4	1		3		2		1	1				9	1								3	2		3	3	2	21
Gout	9	5	12	9	7	7	5	6	8	7	8	13	14	2	2	5	8	8	3	8	8	8	15	3	8	7	37	18	48
Grief	12	13	16	7	17	14	11	17	10	13	10	12	13	4	18	20	22	19	20	20	20	20	37	18	48	59	43	47	279
Hanged, and made away themselves	11	10	13	14	9	14	13	9	14	16	24	18	11	36	8	8	6	13	3	8	7	3	37	18	48	37	72	20	223
Ague, Feb		1	1	3		2	0	6	5	3	4	2	3	26				0	0	0	0	0	4	14	14	17	17	0	2
Jaundice	57	35	39	49	41	43	57	71	61	41	46	77	102	76	47	59	35	50	54	63	184	192	180	212	225		188	998
Jaw-falm	1	1			3			2			1	2			10	16	13	8	12	10	9	11	47	31	48	57	77	20	223
Impoltume	75	61	65	59	80	105	79	90	92	122	80	134	105	96	58	76	73	74	50	62	73	130	382	353	202	5	6	10	95
Itch		1																	10				00	10	01				11
Killed by several Accidents	27	57	35	94	47	45	57	58	52	43	52	47	55	47	54	55	47	46	45	41	51	60	202	201	217	207	164	148	1021
King's Evil	27	26	22	19	22	20	26	26	27	24	23	28	28	54	16	25	18	38	35	20	26	69	97	150	94	94	102	60	537

John Graunt collected and published public health data in the UK in the *Bills of Mortality* (circa 1630), and his statistical analysis identified the plague as a significant source of premature deaths.

Image source: British Library, public domain

Python and SciPy

Environment used in this coursework:

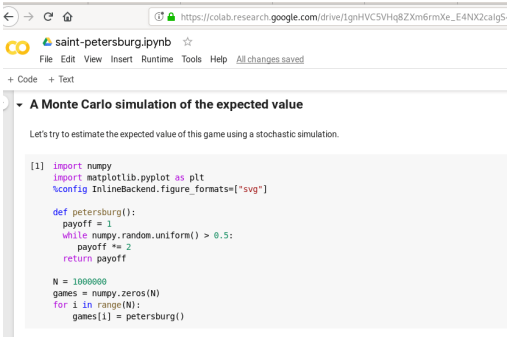
- Python programming language + SciPy + NumPy + matplotlib libraries
- Alternative to Matlab, Scilab, Octave, R
- Free software
- A real programming language with simple syntax
  - much, much more powerful than a spreadsheet!
- Rich scientific computing libraries
  - statistical measures
  - visual presentation of data
  - optimization, interpolation and curve fitting
  - stochastic simulation
  - machine learning, image processing...

How do I run it?

- **Cloud** without local installation
  - Google Colaboratory, at `colab.research.google.com`
  - CoCalc, at `cocalc.com`
- **Microsoft Windows**: install one of
  - Anaconda from `anaconda.com/download/`
  - pythonxy from `python-xy.github.io`
- **MacOS**: install one of
  - Anaconda from `anaconda.com/download/`
  - Pyzo, from `pyzo.org`
- **Linux**: install packages `python`, `numpy`, `matplotlib`, `scipy`
  - your distribution's packages are probably fine

RISK ENGINEERING

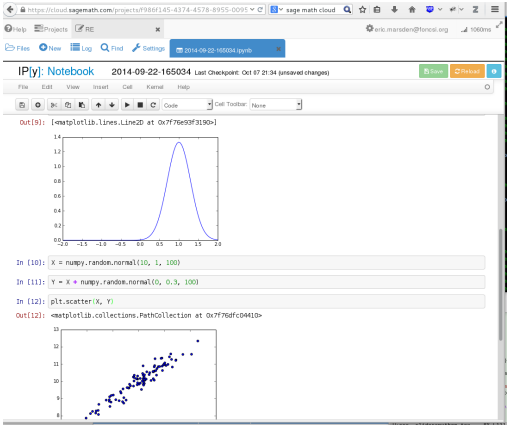
Google Colaboratory



→ `colab.research.google.com`

- ▷ Runs in the cloud, access via web browser
- ▷ No local installation needed
- ▷ Can save to your Google Drive
- ▷ Notebooks are live computational documents, great for "experimenting"

CoCalc



→ `cocalc.com`

- Runs in the cloud, access via web browser
- No local installation needed
- Access to Python in a Jupyter notebook, Sage, R
- Create an account for free
- Similar tools:
  - Microsoft Azure Notebooks
  - JupyterHub, at `jupyter.org/try`

Python as a statistical calculator

```
In [1]: import numpy

In [2]: 2 + 2
Out[2]: 4

In [3]: numpy.sqrt(2 + 2)
Out[3]: 2.0

In [4]: numpy.pi
Out[4]: 3.141592653589793

In [5]: numpy.sin(numpy.pi)
Out[5]: 1.2246467991473532e-16

In [6]: numpy.random.uniform(20, 30)
Out[6]: 28.890905809912784

In [7]: numpy.random.uniform(20, 30)
Out[7]: 20.58728078429875
```

Download this content as a Python notebook at `risk-engineering.org`

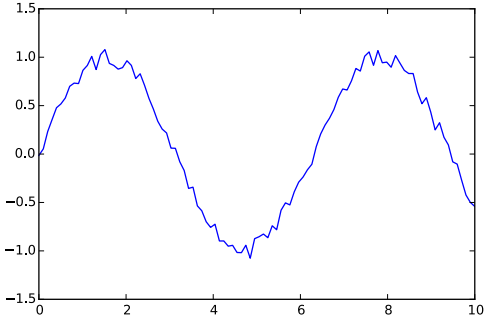
RISK ENGINEERING

Python as a statistical calculator

```
In [3]: obs = numpy.random.uniform(20, 30, 10)

In [4]: obs
Out[4]:
array([ 25.64917726,  21.35270677,  21.71122725,  27.94435625,
        25.43993038,  22.72479854,  22.35164765,  20.23228629,
        26.05497056,  22.01504739])

In [5]: len(obs)
Out[5]: 10

In [6]: obs + obs
Out[6]:
array([ 51.29835453,  42.70541355,  43.42245451,  55.8887125 ,
        50.87986076,  45.44959708,  44.7032953 ,  40.46457257,
        52.10994112,  44.03009478])

In [7]: obs - 25
Out[7]:
array([ 0.64917726, -3.64729323, -3.28877275,  2.94435625,
0.43993038,
       -2.27520146, -2.64835235, -4.76771371,  1.05497056,
-2.98495261])

In [8]: obs.mean()
Out[8]: 23.547614834213316

In [9]: obs.sum()
Out[9]: 235.47614834213317

In [10]: obs.min()
Out[10]: 20.232286285845483
```

Python as a statistical calculator: plotting

```
In [2]: import numpy, matplotlib.pyplot as plt

In [3]: x = numpy.linspace(0, 10, 100)

In [4]: obs = numpy.sin(x) + numpy.random.uniform(-0.1, 0.1, 100)

In [5]: plt.plot(x, obs)
Out[5]: [<matplotlib.lines.Line2D at 0x7f47ecc96da0>]

In [7]: plt.plot(x, obs)
Out[7]: [<matplotlib.lines.Line2D at 0x7f47ed42f0f0>]
```



Some basic notions in probability and statistics

Discrete vs continuous variables

Discrete

A discrete variable takes separate, **countable** values

Examples:

- outcomes of a coin toss: {head, tail}
- number of students in the class
- questionnaire responses {very unsatisfied, unsatisfied, satisfied, very satisfied}

Continuous

A continuous variable is the result of a **measurement** (a floating point number)

Examples:

- height of a person
- flow rate in a pipeline
- volume of oil in a drum
- time taken to cycle from home to university

Random variables

A **random variable** is a set of possible values from a stochastic experiment

Examples:

- sum of the values on two dice throws (a discrete random variable)
- height of the water in a river at time t (a continuous random variable)
- time until the failure of an electronic component
- number of cars on a bridge at time t
- number of new influenza cases at a hospital in a given month
- number of defective items in a batch produced by a factory

RISK ENGINEERING

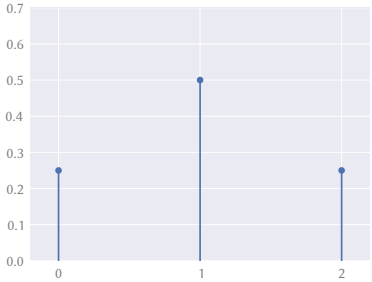
Probability Mass Functions

- For all values x that a discrete random variable X may take, we define the function

$$p_X(x) \stackrel{\text{def}}{=} \Pr(X \text{ takes the value } x)$$

- This is called the **probability mass function** (PMF) of X

- Example: $X =$ *"number of heads when tossing a coin twice"*
  - $p_X(0) \stackrel{\text{def}}{=} \Pr(X = 0) = 1/4$
  - $p_X(1) \stackrel{\text{def}}{=} \Pr(X = 1) = 2/4$
  - $p_X(2) \stackrel{\text{def}}{=} \Pr(X = 2) = 1/4$



Probability Mass Functions: two coins

- **Task**: simulate *"expected number of heads when tossing a coin twice"*

- Let's simulate a coin toss by random choice between 0 and 1

```
> numpy.random.randint(0, 2)
1
```

inclusive lower bound

exclusive upper bound

Download this content as a Python notebook at `risk-engineering.org`

Probability Mass Functions: two coins

- **Task**: simulate *"expected number of heads when tossing a coin twice"*

- Let's simulate a coin toss by random choice between 0 and 1

```
> numpy.random.randint(0, 2)
1
```

inclusive lower bound

exclusive upper bound

- Toss a coin twice:

```
> numpy.random.randint(0, 2, 2)
array([0, 1])
```

count

Download this content as a Python notebook at `risk-engineering.org`

Probability Mass Functions: two coins

- **Task**: simulate *"expected number of heads when tossing a coin twice"*

- Let's simulate a coin toss by random choice between 0 and 1

```
> numpy.random.randint(0, 2)
1
```

inclusive lower bound

exclusive upper bound

- Toss a coin twice:

```
> numpy.random.randint(0, 2, 2)
array([0, 1])
```

count

- Number of heads when tossing a coin twice:

```
> numpy.random.randint(0, 2, 2).sum()
1
```

Download this content as a Python notebook at `risk-engineering.org`

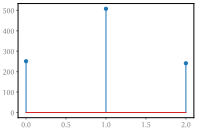
RISK ENGINEERING

Probability Mass Functions: two coins

- **Task**: simulate *"expected number of heads when tossing a coin twice"*

- Do this 1000 times and plot the resulting PMF:

```
import numpy
import matplotlib.pyplot as plt

N = 1000
heads = numpy.zeros(N, dtype=int)
for i in range(N):
    # second argument to randint is exclusive upper bound
    heads[i] = numpy.random.randint(0, 2, 2).sum()
plt.stem(numpy.bincount(heads))
```



`heads[i]`: *element number i of the array* `heads`

More information on Python programming

For more information on Python syntax, check out the book *Think Python*

Purchase, or read online for free at `greenteapress.com/wp/think-python-2e/`

Probability Mass Functions: properties

- A PMF is always non-negative

$$p_X(x) \geq 0$$

- Sum over the support equals 1

$$\sum_x p_X(x) = 1$$

- $$\Pr(a \leq X \leq b) = \sum_{x \in [a,b]} p_X(x)$$

Probability Density Functions

- For continuous random variables, the **probability density function** $f_X(x)$ is defined by

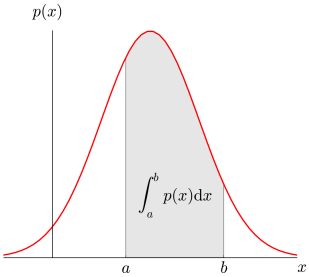
$$\Pr(a \leq X \leq b) = \int_a^b f_X(x)dx$$

- It is non-negative

$$f_X(x) \geq 0$$

- The area under the curve (integral over $\mathbb{R}$) is 1

$$\int_{-\infty}^{\infty} f_X(x)dx = 1$$

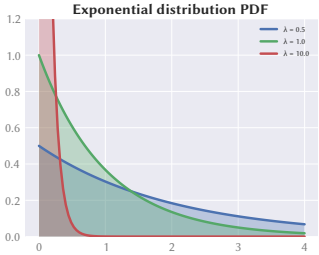


Probability Density Function

In reliability engineering, you are often concerned about the random variable T representing the **time at which a component fails**.

The PDF $f(t)$ is the "failure density function". It tells you the probability of failure around age t.

$$\lim_{\Delta t \to 0} \frac{\Pr(t < T < t + \Delta t)}{\Delta t} = \lim_{\Delta t \to 0} \frac{\int_t^{t+\Delta t} f(t)dt}{\Delta t}$$



Expectation of a random variable

- The **expectation** (or mean) is defined as a weighted average of all possible realizations of a random variable

- Discrete random variable:

$$\mathbb{E}[X] = \mu_X \stackrel{\text{def}}{=} \sum_{i=1}^{N} x_i \times \Pr(X = x_i)$$

- Continuous random variable:

$$\mathbb{E}[X] = \mu_X \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} u \times f(u) du$$

- Interpretation:
  - the center of gravity of the PMF or PDF
  - the average in a large number of independent realizations of your experiment

RISK ENGINEERING

Important concept: independence

- Definition of statistical independence of two events: the outcome of one has no effect on the outcome of the other
- Typical example: successive coin tosses
- The probability of two independent events both happening is the **product** of their individual probabilities
- Often important in safety analysis:
  - if you are responsible for a car accident, your car insurance will become more expensive, because the event indicates you are more likely to have future car accidents (they are not independent events)
  - if an accident of a specific type occurs on an industrial site, it is less likely it will occur again the following year because people will make extra efforts to prevent that type of accident

Illustration: expected value with coins

- X = *"number of heads when tossing a coin twice"*
- **PMF:**
  - $p_X(0) \stackrel{\text{def}}{=} \Pr(X = 0) = 1/4$
  - $p_X(1) \stackrel{\text{def}}{=} \Pr(X = 1) = 2/4$
  - $p_X(2) \stackrel{\text{def}}{=} \Pr(X = 2) = 1/4$
- $\mathbb{E}[X] \stackrel{\text{def}}{=} \sum_k k \times p_X(k) = 0 \times \frac{1}{4} + 1 \times \frac{2}{4} + 2 \times \frac{1}{4} = 1$

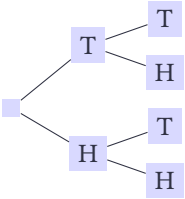


Illustration: expected value of a dice roll

- Expected value of a dice roll is $\sum_{i=1}^{6} i \times \frac{1}{6} = 3.5$
- If we toss a dice a large number of times, the mean value should converge to 3.5
- Let's check that in Python

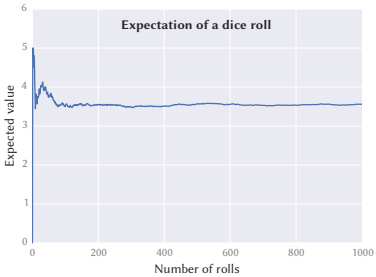
```
> numpy.random.randint(1, 7, 100).mean()
4.2
> numpy.random.randint(1, 7, 1000).mean()
3.478
```

(These numbers will be different for different executions. The greater the number of random "dice throws" we simulate, the greater the probability that the mean will be close to 3.5.)

Illustration: expected value of a dice roll

We can plot the speed of convergence of the mean towards the expected value as follows.

```
N = 1000
roll = numpy.zeros(N)
expectation = numpy.zeros(N)
for i in range(N):
    roll[i] = numpy.random.randint(1, 7)
for i in range(1, N):
    expectation[i] = numpy.mean(roll[0:i])
plt.plot(expectation)
```



Mathematical properties of expectation

If c is a constant and X and Y are random variables, then

- $\mathbb{E}[c] = c$
- $\mathbb{E}[cX] = c\mathbb{E}[X]$
- $\mathbb{E}[c + X] = c + \mathbb{E}[X]$
- $\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]$

Note: in general $\mathbb{E}[g(X)] \neq g(\mathbb{E}[X])$

Aside: existence of expectation

- Not all random variables have an expectation
- Consider a random variable X defined on some (infinite) sample space Ω so that for all positive integers i, X takes the value
  - 2^i with probability 2^{-i-1}
  - -2^i with probability 2^{-i-1}
- Both the positive part and the negative part of X have infinite expectation in this case, so $\mathbb{E}[X]$ would have to be $\infty - \infty$ (meaningless)

Variance of a random variable

- The **variance** provides a measure of the dispersion around the mean
  - intuition: how "spread out" the data is

- For a discrete random variable:

$$\mathrm{Var}(X) = \sigma_X^2 \stackrel{\text{def}}{=} \sum_{i=1}^{N} (X_i - \mu_X)^2 \Pr(X = x_i)$$

- For a continuous random variable:

$$\mathrm{Var}(X) = \sigma_X^2 \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} (x - \mu_x)^2 f(u) du$$

- In Python:
  - `obs.var()` if `obs` is a NumPy vector
  - `numpy.var(obs)` for any Python sequence (vector or list)

In Excel, function VAR

Variance with coins

- X = *"number of heads when tossing a coin twice"*
- PMF:
  - $p_X(0) \stackrel{\text{def}}{=} \Pr(X = 0) = {}^1/_4$
  - $p_X(1) \stackrel{\text{def}}{=} \Pr(X = 1) = {}^2/_4$
  - $p_X(2) \stackrel{\text{def}}{=} \Pr(X = 2) = {}^1/_4$
-

$$\begin{aligned}\mathrm{Var}(X) &\stackrel{\text{def}}{=} \sum_{i=1}^{N} (x_i - \mu_X)^2 \Pr(X = x_i) \\ &= \frac{1}{4} \times (0-1)^2 + \frac{2}{4} \times (1-1)^2 + \frac{1}{4} \times (2-1)^2 = \frac{1}{2}\end{aligned}$$

Variance of a dice roll

- Analytic calculation of the variance of a dice roll:

$$
\begin{aligned}
Var(X) &\stackrel{\text{def}}{=} \sum_{i=1}^{N} (X_i - \mu_X)^2 \Pr(X = x_i) \\
&= \frac{1}{6} \times \left((1-3.5)^2 + (2-3.5)^2 + (3-3.5)^2 + (4-4.5)^2 + (5-3.5)^2 + (6-3.5)^2 \right) \\
&= 2.916
\end{aligned}
$$

- Let's reproduce that in Python

count

```
> rolls = numpy.random.randint(1, 7, 1000)
> len(rolls)
1000
> rolls.var()
2.9463190000000004
```

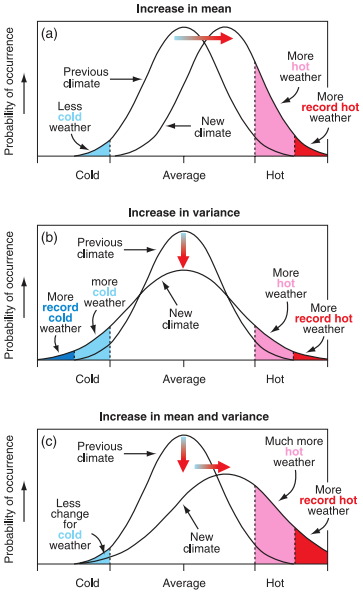


Figure 2.32: Schematic showing the effect on extreme temperatures when (a) the mean temperature increases, (b) the variance increases, and (c) when both the mean and variance increase for a normal distribution of temperature.

Source: IPPC report *Climate Change 2001: The Scientific Basis*, page 155. This is a famous figure that attempted to explain the effets of global warming. It also helps to understand the difference between mean (or mathematical expectation) and variance.

Properties of variance as a mathematical operator

If c is a constant and X and Y are random variables, then

- $\mathrm{Var}(X) \geq 0$ (variance is always non-negative)
- $\mathrm{Var}(c) = 0$
- $\mathrm{Var}(c + X) = \mathrm{Var}(X)$
- $\mathrm{Var}(cX) = c^2\ \mathrm{Var}(X)$
- $\mathrm{Var}(X + Y) = \mathrm{Var}(X) + \mathrm{Var}(Y)$, **if X and Y are independent**
- $\mathrm{Var}(X + Y) = \mathrm{Var}(X) + \mathrm{Var}(Y) + 2\mathrm{Cov}(X, Y)$ if X and Y are dependent

Beware:

- $\mathbb{E}[X^2] \neq (\mathbb{E}[X])^2$
- $\mathbb{E}[\sqrt{X}] \neq \sqrt{\mathbb{E}[X]}$

Properties of variance as a mathematical operator

If c is a constant and X and Y are random variables, then

- $\mathrm{Var}(X) \geq 0$ (variance is always non-negative)
- $\mathrm{Var}(c) = 0$
- $\mathrm{Var}(c + X) = \mathrm{Var}(X)$
- $\mathrm{Var}(cX) = c^2\ \mathrm{Var}(X)$
- $\mathrm{Var}(X + Y) = \mathrm{Var}(X) + \mathrm{Var}(Y)$, **if X and Y are independent**
- $\mathrm{Var}(X + Y) = \mathrm{Var}(X) + \mathrm{Var}(Y) + 2\mathrm{Cov}(X, Y)$ if X and Y are dependent

Beware:

- $\mathbb{E}[X^2] \neq (\mathbb{E}[X])^2$
- $\mathbb{E}[\sqrt{X}] \neq \sqrt{\mathbb{E}[X]}$

Note:

- $\mathrm{Cov}(X, Y) \stackrel{\text{def}}{=} \mathbb{E}\left[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])\right]$
- $\mathrm{Cov}(X, X) = \mathrm{Var}(X)$

RISK ENGINEERING

Standard deviation

- Formula for variance: $\mathrm{Var}(X) \stackrel{\text{def}}{=} \sum_{i=1}^{N} (X_i - \mu_X)^2 \Pr(X = x_i)$
- If random variable X is expressed in metres, $\mathrm{Var}(X)$ is in m^2
- To obtain a measure of dispersion of a random variable around its expected value which has the same units as the random variable itself, take the square root
- Standard deviation $\sigma \stackrel{\text{def}}{=} \sqrt{\mathrm{Var}(X)}$
- In Python:
  - `obs.std()` if `obs` is a NumPy vector
  - `numpy.std(obs)` for any Python sequence (vector or list)

In Excel, function STDEV

RISK ENGINEERING

Properties of standard deviation

- Suppose $Y = aX + b$, where
  - a and b are scalar
  - X and Y are two random variables
- Then $\sigma(Y) = |a|\,\sigma(X)$
- Let's check that with NumPy:

```
> X = numpy.random.uniform(100, 200, 1000)
> a = -32
> b = 16
> Y = a * X + b
> Y.std()
914.94058476118835
> abs(a) * X.std()
914.94058476118835
```

Properties of expectation & variance: empirical testing with numpy

- $\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]$

```
> X = numpy.random.randint(1, 101, 1000)
> Y = numpy.random.randint(-300, -199, 1000)
> X.mean()
50.575000000000003
> Y.mean()
-251.613
> (X + Y).mean()
-201.038
```

Properties of expectation & variance: empirical testing with numpy

- $\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]$

```
> X = numpy.random.randint(1, 101, 1000)
> Y = numpy.random.randint(-300, -199, 1000)
> X.mean()
50.575000000000003
> Y.mean()
-251.613
> (X + Y).mean()
-201.038
```

- $\mathrm{Var}(cX) = c^2\ \mathrm{Var}(X)$

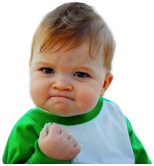
```
> numpy.random.randint(1, 101, 1000).var()
836.616716
> numpy.random.randint(5, 501, 1000).var()
20514.814318999997
> 5 * 5 * 836
20900
```

RISK ENGINEERING

Properties of expectation & variance: empirical testing with numpy

- $\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]$

```
> X = numpy.random.randint(1, 101, 1000)
> Y = numpy.random.randint(-300, -199, 1000)
> X.mean()
50.575000000000003
> Y.mean()
-251.613
> (X + Y).mean()
-201.038
```



- $\mathrm{Var}(cX) = c^2\ \mathrm{Var}(X)$

```
> numpy.random.randint(1, 101, 1000).var()
836.616716
> numpy.random.randint(5, 501, 1000).var()
20514.814318999997
> 5 * 5 * 836
20900
```

Cumulative Distribution Function

- The cumulative distribution function (CDF) of random variable X, denoted by $F_X(x)$, indicates the probability that X assumes a value $\leq x$, where x is any real number

$$F_X(x) = \Pr(X \leq x) \qquad -\infty \leq x \leq \infty$$

- Properties of a CDF:
  - $F_X(x)$ is a non-decreasing function of x
  - $0 \leq F_X(x) \leq 1$
  - $\lim_{x\to\infty} F_X(x) = 1$
  - $\lim_{x\to\infty} F_X(x) = 0$
  - if $x \leq y$ then $F_X(x) \leq F_X(y)$
  - $\Pr(a < X \leq b) = F_X(b) - F_X(a) \quad \forall b > a$

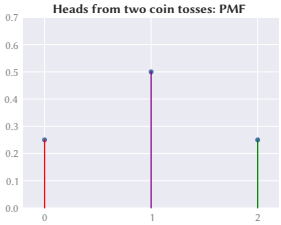
CDF of a discrete distribution

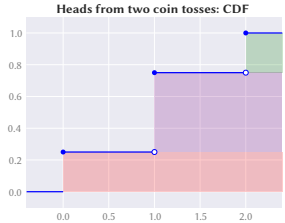
$$F_X(x) = \Pr(X \leq x) \qquad -\infty \leq x \leq \infty$$
$$= \sum_{x_i \leq x} \Pr(X = x_i)$$

The CDF is built by accumulating probability as x increases.

Consider the random variable $X =$ *"number of heads when tossing a coin twice"*.

x	0	1	2
PMF, $p_X(x) = \Pr(X = x)$	$1/4$	$2/4$	$1/4$
CDF, $F_X(x) = \Pr(X \leq x)$	$1/4$	$3/4$	1

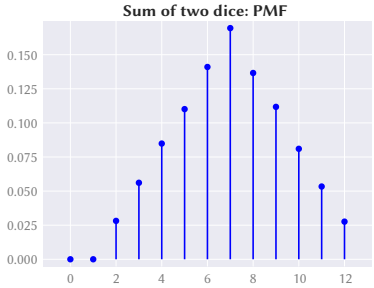


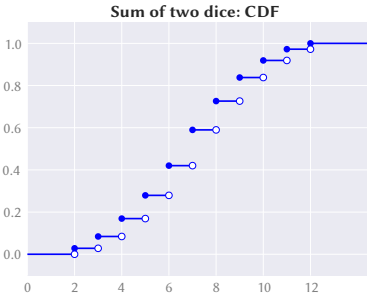


CDF of a discrete distribution

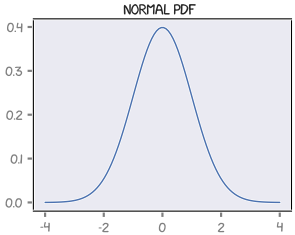
$$
\begin{aligned}
F_X(x) &= \Pr(X \leq x) \qquad -\infty \leq x \leq \infty \\
&= \sum_{x_i \leq x} \Pr(X = x_i)
\end{aligned}
$$

Example: sum of two dice



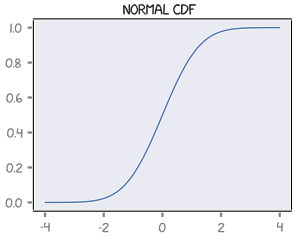


CDF of a continuous distribution



Python: `scipy.stats.norm(loc=mu, scale=sigma).pdf(x)`

$$F_X(x) = \Pr(X \leq x)$$
$$= \int_{-\infty}^{x} f(u)du$$



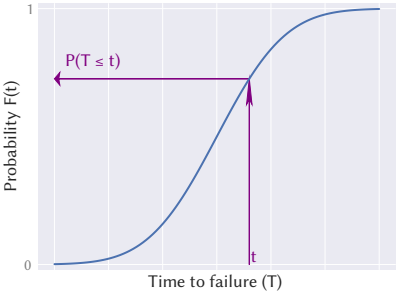
Python: `scipy.stats.norm(loc=mu, scale=sigma).cdf(x)`

Interpreting the CDF

In reliability engineering, we are often interested in the random variable T representing **time to failure** of a component.

The cumulative distribution function tells you the probability that lifetime is $\leq t$.

$$F(t) = \Pr(T \leq t)$$



Exercise

Problem

Field data tells us that the time to failure of a pump, X, is normally distributed. The mean and standard deviation of the time to failure are estimated from historical data as $\mu = 3200$ hours and $\sigma = 600$ hours.

What is the probability that a pump will fail before 2000 hours of operation?

Exercise

Problem

Field data tells us that the time to failure of a pump, X, is normally distributed. The mean and standard deviation of the time to failure are estimated from historical data as $\mu = 3200$ hours and $\sigma = 600$ hours.

What is the probability that a pump will fail before 2000 hours of operation?

Solution

We are interested in calculating $\Pr(X \leq 2000)$ and we know that X follows a `norm(3200, 600)` distribution. We can use the CDF to calculate $\Pr(X \leq 2000)$.

We want `norm(3200, 600).cdf(2000)`, which is 0.022750 (or 2.28%).

Exercise

Problem

Field data tells us that the time to failure of a pump, X, is normally distributed. The mean and standard deviation of the time to failure are estimated from historical data as $\mu = 3200$ hours and $\sigma = 600$ hours.

What is the probability that a pump will fail after it has worked for *at least* 2000 hours?

Exercise

Problem

Field data tells us that the time to failure of a pump, X, is normally distributed. The mean and standard deviation of the time to failure are estimated from historical data as $\mu = 3200$ hours and $\sigma = 600$ hours.

What is the probability that a pump will fail after it has worked for *at least* 2000 hours?

Solution

We are interested in calculating $\Pr(X > 2000)$ and we know that X follows a `norm(3200, 600)` distribution. We know that $\Pr(X > 2000) = 1 - \Pr(X \leq 2000)$.

We want `1 - norm(3200, 600).cdf(2000)`, which is 0.977 (or 97.7%).

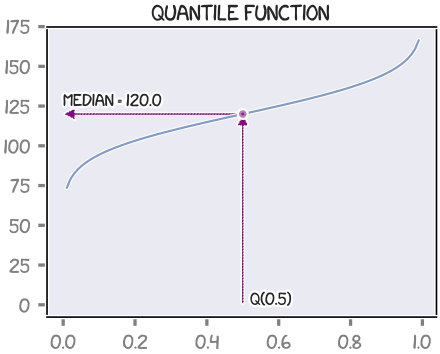
The quantile function

The quantile function is the inverse of the CDF.

The **median** is the point where half the population is below and half is above (it's the 0.5 quantile, and the 50th percentile).

Consider a normal distribution centered in 120 with a standard deviation of 20.

```
> import scipy.stats
> distrib = scipy.stats.norm(120, 20)
> distrib.ppf(0.5)
120.0
> distrib.cdf(120.0)
0.5
```

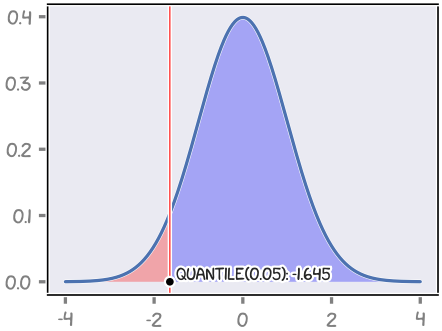


Quantile measures

A quantile measure is a **cutpoint in the probability distribution** indicating the value below which a given percentage of the sample falls.

The 0.05 quantile is the x value which has 5% of the sample smaller than x.

It's also called the 5th **percentile**.



The 0.05 quantile of the standard normal distribution (centered in 0, standard deviation of 1)

```
import scipy.stats
scipy.stats.norm(0, 1).ppf(0.05)
-1.6448536269514729
```

Quantile measures

Quantile measures are often used in health.

To the right, illustration of the range of baby heights and weights as a function of their age.

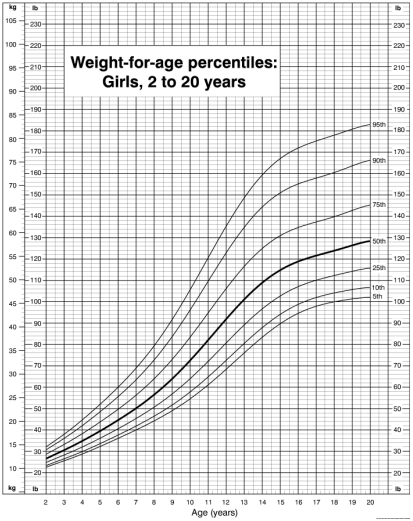


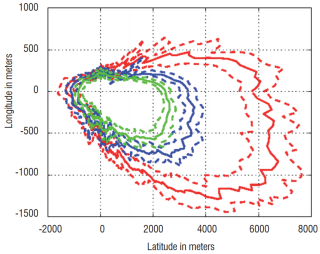
Image source: US CDC

Quantile measures in risk analysis

Risk analysis and reliability engineering: analysts are interested in the **probability of extreme events**

- ▷ what is the probability of a flood higher than my dike?
- ▷ how high do I need to build a dike to protect against hundred-year floods?
- ▷ what is the probability of a leak given the corrosion measurements I have made?

Problem: these are **rare events** so it's difficult to obtain confidence that a model representing the underlying mechanism works well for extremes



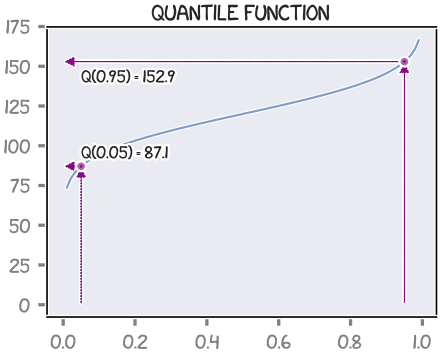
Three percentile measures (95% = green, 99% = blue, 99.99% = red) of the spatial risk of fallback from a rocket launcher. Dotted lines indicate uncertainty range.

Image source: `aerospacelab-journal.org/sites/www.aerospacelab-journal.org/files/AL04-13_0.pdf`

Quantiles and confidence intervals

A **90% confidence interval** is the set of points between the 0.05 quantile (5% of my observations are smaller than this value) and the 0.95 quantile (5% of my observations are larger than this value).

In the example to the right, the 90% confidence interval is [87.1, 152.9].



Scipy.stats package

- The `scipy.stats` module implements many continuous and discrete random variables and their associated distributions
  - binomial, poisson, exponential, normal, uniform, weibull...
- Usage: instantiate a distribution then call a method
  - `rvs`: random variates
  - `pdf`: Probability Density Function
  - `cdf`: Cumulative Distribution Function
  - `sf`: Survival Function (1-CDF)
  - `ppf`: Percent Point Function (inverse of CDF)
  - `isf`: Inverse Survival Function (inverse of SF)

RISK ENGINEERING

Simulating dice throws

- Maximum of 1000 throws

```
> dice = scipy.stats.randint(1, 7)
> dice.rvs(1000).max()
6
```

exclusive upper bound

- What is the probability of a die rolling 4?

```
> dice.pmf(4)
0.16666666666666666
```

- What is the probability of rolling 4 or below?

```
> dice.cdf(4)
0.66666666666666663
```

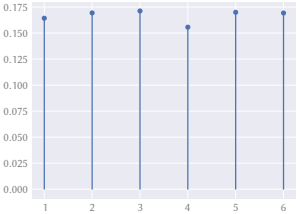
- What is the probability of rolling between 2 and 4 (inclusive)?

```
> dice.cdf(4) - dice.cdf(1)
0.5
```

RISK ENGINEERING

Simulating dice

```
> import numpy
> import matplotlib.pyplot as plt

> toss = numpy.random.choice(range(1, 7))
> toss
2
> N = 10000
> tosses = numpy.random.choice(range(1, 7), N)
> tosses
array([6, 6, 4, ..., 2, 4, 5])
> tosses.mean()
3.5088
> numpy.median(tosses)
4.0
> len(numpy.where(tosses > 3)[0]) / float(N)
0.5041
> x, y = numpy.unique(tosses, return_counts=True)
> plt.stem(x, y/float(N))
```



Scipy.stats examples

- Generate 5 random variates from a continuous uniform distribution between 90 and 100
- Check that the expected value of the distribution is around 95
- Check that around 20% of variates are less than 92

```
> import scipy.stats

> u = scipy.stats.uniform(90, 10)
> u.rvs(5)
array([ 94.0970853 ,  92.41951494,
  90.25127254, 91.69097729,
  96.1811148 ])
> u.rvs(1000).mean()
94.892801456986376
> (u.rvs(1000) < 92).sum() / 1000.0
0.193
```

Some important probability distributions

Some important probability distributions

Coin tossing with uneven coin	Bernoulli	`scipy.stats.bernoulli`
Rolling a dice	uniform	`scipy.stats.randint`
Counting errors/successes	Binomial	`scipy.stats.binom`
Trying until success	geometric	`scipy.stats.geom`
Countable, rare events whose occurrence is independent	Poisson	`scipy.stats.poisson`
Random “noise”, sums of many variables	normal	`scipy.stats.norm`

RISK ENGINEERING

Bernoulli trials

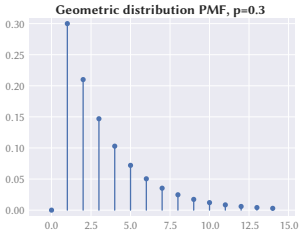
- A *trial* is an experiment which can be repeated many times with the same probabilities, each realization being independent of the others

- Bernoulli trial: an experiment in which N trials are made of an event, with probability p of "success" in any given trial and probability $1 - p$ of "failure"
  - "success" and "failure" are mutually exclusive
  - example: sequence of coin tosses
  - example: arrival of requests in a web server per time slot

Jakob Bernoulli (1654–1705)

The geometric distribution (trying until success)

- We conduct a sequence of Bernoulli trials, each with success probability p
- What's the probability that it takes k trials to get a success?
  - Before we can succeed at trial k, we must have had $k - 1$ failures
  - Each failure occurred with probability $1 - p$, so total probability $(1 - p)^{k-1}$
  - Then a single success with probability p
- $\Pr(X = k) = (1 - p)^{k-1} p$

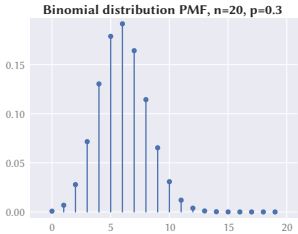


The geometric distribution: intuition

- Suppose I am at a party and I start asking girls to dance. Let X be the number of girls that I need to ask in order to find a partner.
  - If the first girl accepts, then $X = 1$
  - If the first girl declines but the next girl accepts, then $X = 2$
- $X = k$ means that I failed on the first $k - 1$ tries and succeeded on the kth try
  - My probability of failing on the first try is $(1 - p)$
  - My probability of failing on the first two tries is $(1 - p)(1 - p)$
  - My probability of failing on the first $n - 1$ tries is $(1 - p)^{k-1}$
  - Then, my probability of succeeding on the nth try is p
- Properties:
  - $\mathbb{E}[X] = \frac{1}{p}$
  - $\text{Var}(X) = \frac{1-p}{p^2}$

The binomial distribution (counting successes)

- Also arises when observing multiple Bernoulli trials
  - exactly two mutually exclusive outcomes, “success” and “failure”
- $Binomial(p, k, n)$: probability of observing k successes in n trials, where probability of success on a single trial is p
  - example: toss a coin n times ($p = 0.5$) and see k heads
- We have k successes, which happens with a probability of p^k
- We have $n - k$ failures, which happens with probability $(1 - p)^{n-k}$
- We can generate these k successes in many different ways from n trials, $\binom{n}{k}$ ways
- $\Pr(X = k) = \binom{n}{k}(1 - p)^{n-k}p^k$



Reminder: binomial coefficient $\binom{n}{k}$ is $\frac{n!}{k!(n-k)!}$

Binomial distribution: application

- Consider a medical test with an error rate of 0.1 applied to 100 patients
- What is the probability that we see at most 1 test error?
- What is the probability that we see at most 10 errors?
- If the random variable X represents the number of test errors, what is the smallest k such that $P(X \leq k)$ is at least 0.05?

```
> import scipy.stats

> test = scipy.stats.binom(n=100, p=0.1)
> test.cdf(1)
0.00032168805319411544
> test.cdf(10)
0.58315551226649232
> test.ppf(0.05)
5.0
```

When reporting results, make sure you pay attention to the number of significant figures in the input data (2 in this case).

RISK ENGINEERING

Binomial distribution: application

Q: A company drills 9 oil exploration wells, each with a 10% chance of success. Eight of the nine wells fail. What is the probability of that happening?

Binomial distribution: application

Q: A company drills 9 oil exploration wells, each with a 10% chance of success. Eight of the nine wells fail. What is the probability of that happening?

Analytic solution

Each well is a binomial trial with $p = 0.1$. We want the probability of exactly one success.

```
> import scipy.stats
> wells = scipy.stats.binom(n=9, p=0.1)
> wells.pmf(1)
0.38742048899999959
```

Answer by simulation

Run 20 000 trials of the model and count the number that generate 1 positive result.

```
> import scipy.stats
> N = 20_000
> wells = scipy.stats.binom(n=9, p=0.1)
> trials = wells.rvs(N)
> (trials == 1).sum() / float(N)
0.38679999999999998
```

RISK ENGINEERING

Binomial distribution: application

Q: A company drills 9 oil exploration wells, each with a 10% chance of success. Eight of the nine wells fail. What is the probability of that happening?

Analytic solution

Each well is a binomial trial with $p = 0.1$. We want the probability of exactly one success.

```
> import scipy.stats
> wells = scipy.stats.binom(n=9, p=0.1)
> wells.pmf(1)
0.38742048899999959
```

Answer by simulation

Run 20 000 trials of the model and count the number that generate 1 positive result.

```
> import scipy.stats
> N = 20_000
> wells = scipy.stats.binom(n=9, p=0.1)
> trials = wells.rvs(N)
> (trials == 1).sum() / float(N)
0.38679999999999998
```

The probability of all 9 wells failing is $0.9^9 = 0.3874$ (and also `wells.pmf(0)`).

The probability of *at least* 8 wells failing is `wells.cdf(1)`. It's also `wells.pmf(0) + wells.pmf(1)` (it's 0.7748).

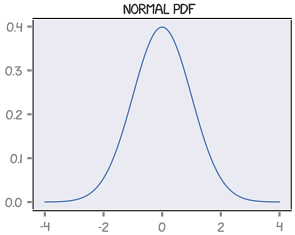
Binomial distribution: properties

Exercise: check empirically (with SciPy) the following properties of the binomial distribution:

- the mean of the distribution (μ_x) is equal to $n \times p$
- the variance (σ_X^2) is $n \times p \times (1 - p)$
- the standard deviation (σ_x) is $\sqrt{n \times p \times (1 - p)}$

Gaussian (normal) distribution

- The famous "bell shaped" curve, fully described by its mean and standard deviation
- Good representation of distribution of measurement errors and many population characteristics
  - size, mechanical strength, duration, speed
- Symmetric around the mean
- Mean = median = mode
- Python: `scipy.stats.norm(μ, σ)`
- Excel: `NORMINV(RAND(), μ, σ)`



Scipy.stats examples

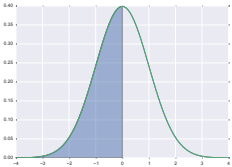
- Consider a Gaussian distribution centered in 5, standard deviation of 1
- Check that half the distribution is located to the left of 5
- Find the first percentile (value of x which has 1% of realizations to the left)
- Check that it is equal to the 99% survival quantile

```
> dist = scipy.stats.norm(5, 1)
> dist.cdf(5)
0.5
> dist.ppf(0.5)
5.0
> dist.ppf(0.01)
2.6736521259591592
> dist.isf(0.99)
2.6736521259591592
> dist.cdf(2.67)
0.0099030755591642452
```

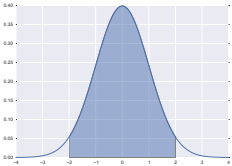
Area under the normal distribution

```
In [1]: import numpy

In [2]: from scipy.stats import norm

In [3]: norm.ppf(0.5)
Out[3]: 0.0

In [4]: norm.cdf(0)
Out[4]: 0.5

In [5]: norm.cdf(2) - norm.cdf(-2)
Out[5]: 0.95449973610364158

In [6]: norm.cdf(3) - norm.cdf(-3)
Out[6]: 0.99730020393673979
```

quantile function



there is a 95% chance that the number drawn falls within 2 standard deviations of the mean



In prehistoric times, statistics textbooks contained large tables of quantile values for the normal distribution. With cheap computing power, no longer necessary!

The “68–95–99.7 rule”

- The 68–95–99.7 rule (aka the *three-sigma rule*) states that if x is an observation from a normally distributed random variable with mean μ and standard deviation σ, then
  - $\Pr(\mu - \sigma \leq x \leq \mu + \sigma) \approx 0.6827$
  - $\Pr(\mu - 2\sigma \leq x \leq \mu + 2\sigma) \approx 0.9545$
  - $\Pr(\mu - 3\sigma \leq x \leq \mu + 3\sigma) \approx 0.9973$

- The 6σ quality management method pioneered by Motorola aims for 99.99966% of production to meet quality standards
  - 3.4 defective parts per million opportunities (DPMO)
  - actually, that’s only 4.5 sigma!
  - `(scipy.stats.norm.cdf(6) - scipy.stats.norm.cdf(-6)) * 100` ⟶ 99.999999802682453

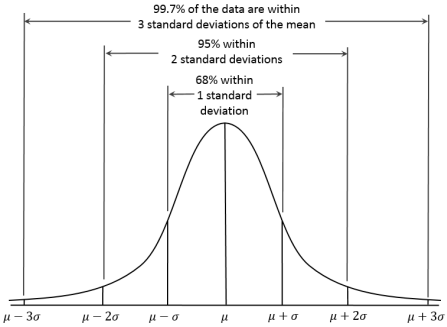


Image source: Wikipedia on 68–95–99.7 rule

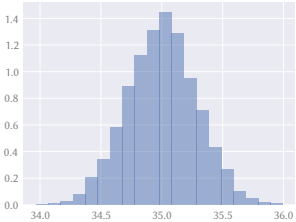
Central limit theorem

- Theorem states that the mean (also true of the sum) of a set of random measurements will tend to a normal distribution, no matter the shape of the original measurement distribution

- Part of the reason for the ubiquity of the normal distribution in science

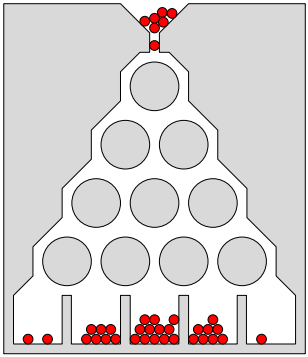
- Python simulation:

```
N = 10_000
sim = numpy.zeros(N)
for i in range(N):
    sim[i] = numpy.random.uniform(30, 40, 100).mean()
plt.hist(sim, bins=20, alpha=0.5, density=True)
```



- **Exercise**: try this with other probability distributions and check that the simulations tend towards a normal distribution

Galton board



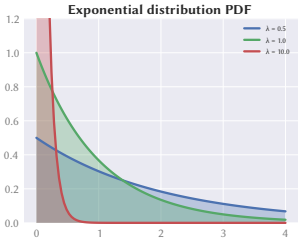
- The Galton board (or "bean machine") has two parts:
  - top: evenly-spaced pegs in staggered rows
  - bottom: evenly-spaced rectangular slots
- Balls introduced at the top bounce of the pegs, equal probability of going right or left at each successive row
  - each vertical step is a Bernoulli trial
- Balls collect in the slots at the bottom with heights following a binomial distribution
  - and for large number of balls, a normal distribution
- Interactive applet emulating a Galton board:

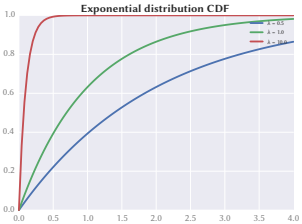
  → `randomservices.org/random/apps/GaltonBoardGame.html`

Named after English psychometrician Sir Francis Galton (1822–1911)

Exponential distribution

- PDF: $f_Z(z) = \lambda e^{-\lambda z}, z \geq 0$
- CDF: $\Pr(Z \leq z) = F_Z(z) = 1 - e^{-\lambda z}$
- The hazard function, or *failure rate*, is constant, equal to λ
  - $1/\lambda$ is the "mean time between failures", or MTBF
  - λ can be calculated by $\frac{\text{total number of failures}}{\text{total operating time}}$
- Often used in reliability engineering to represent failure of electronic equipment (no wear)
- Property: expected value of exponential random variable is $\frac{1}{\lambda}$





Exponential distribution

Let's check that the expected value of an exponential random variable is $\frac{1}{\lambda}$

```
> import scipy.stats
> lda = 25
> dist = scipy.stats.expon(scale=1/float(lda))
> obs = dist.rvs(size=1000)
> obs.mean()
0.041137615318791773
> obs.std()
0.03915081431615041
> 1/float(lda)
0.04
```

Exponential distribution: memoryless property

- An exponentially distributed random variable T obeys

$$\Pr(T > s + t \,|\, T > s) = \Pr(T > t), \qquad \forall s, t \geq 0$$

  where the vertical bar | indicates a conditional probability.

- Interpretation: if T represents time of failure
  - The distribution of the remaining lifetime does not depend on how long the component has been operating (item is "as good as new")
  - Distribution of remaining lifetime is the same as the original lifetime
  - An observed failure is the result of some suddenly appearing failure, not due to gradual deterioration

Failure of power transistors (1/2)

- Suppose we are studying the reliability of a power system, which fails if any of 3 power transistors fails
- Let X, Y, Z be random variables modelling failure time of each transistor (in hours)
  - transistors have no physical wear, so model by exponential random variables
  - failures are assumed to be independent
- $X \sim Exp(1/5000)$ (mean failure time of 5000 hours)
- $Y \sim Exp(1/8000)$ (mean failure time of 8000 hours)
- $Z \sim Exp(1/4000)$ (mean failure time of 4000 hours)

Failure of power transistors (2/2)

- System fails if any transistor fails, so time to failure T is $\min(X, Y, Z)$

$$
\begin{aligned}
\Pr(T \leq t) &= 1 - \Pr(T > t) \\
&= 1 - \Pr(\min(X, Y, Z) > t) \\
&= 1 - \Pr(X > t, Y > t, Z > t) \\
&= 1 - \Pr(X > t) \times \Pr(Y > t) \times \Pr(Z > t) \qquad \text{(independence)} \\
&= 1 - (1 - \Pr(X \leq t))\,(1 - \Pr(Y \leq t))\,(1 - \Pr(Z \leq t)) \\
&= 1 - \left(1 - (1 - e^{-t/5000})\right)\left(1 - (1 - e^{-t/8000})\right)\left(1 - (1 - e^{-t/4000})\right) \qquad \text{(exponential CDF)} \\
&= 1 - e^{-t/5000} e^{-t/8000} e^{-t/4000} \\
&= 1 - e^{-t\left(1/5000 + 1/8000 + 1/4000\right)} \\
&= 1 - e^{-0.000575t}
\end{aligned}
$$

- System failure time is also exponentially distributed, with parameter 0.000575
- Expected time to system failure is $1/0.000575 = 1739$ hours

Poisson process: exponential arrival times

- A Poisson process is any process where independent events occur at a constant average rate
  - time between each pair of consecutive events follows an exponential distribution with parameter λ (the *arrival rate*)
  - each of these inter-arrival times is assumed to be independent of other inter-arrival times
  - example: babies are born at a hospital at a rate of five per hour
- The process is *memoryless*: number of arrivals in any bounded interval of time after time t is independent of the number of arrivals before t
- Good model for many types of phenomena:
  - number of road crashes in a zone
  - number of faulty items in a production batch
  - arrival of customers in a queue
  - occurrence of earthquakes

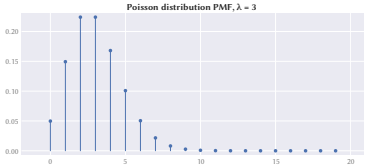
RISK ENGINEERING

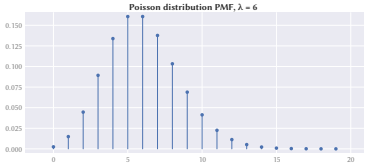
The Poisson distribution

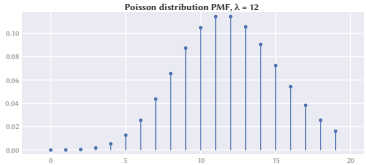
- The probability distribution of the *counting process* associated with a Poisson process
  - the number of events of the Poisson process over a time interval
- Probability mass function:

$$\Pr(Z = k) = \frac{\lambda^k e^{-\lambda}}{k!}, k = 0, 1, 2\ldots$$

- The parameter λ is called the *intensity* of the Poisson distribution
  - increasing λ adds more probability to larger values
- Python: `scipy.stats.poisson(λ)`



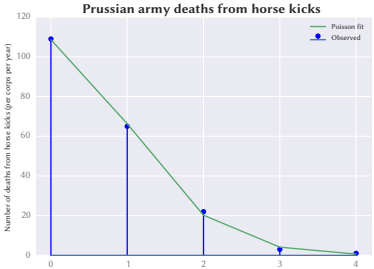




Poisson distribution and Prussian horses

- Number of fatalities for the Prussian cavalry resulting from being kicked by a horse was recorded over a period of 20 years
  - for 10 army corps, so total number of observations is 200

Deaths	Occurrences
0	109
1	65
2	22
3	3
4	1
> 4	0



- It follows a Poisson distribution

- **Exercise**: reproduce the plot on the right which shows a fit between a Poisson distribution and the historical data

The Poisson distribution: properties

- Expected value of the Poisson distribution is equal to its parameter μ
- Variance of the Poisson distribution is equal to its parameter μ
- The sum of independent Poisson random variables is also Poisson
- Specifically, if Y_1 and Y_2 are independent with $Y_i \sim Poisson(\mu_i)$ for $i = 1, 2$ then $Y_1 + Y_2 \sim Poisson(\mu_1 + \mu_2)$
- **Exercise**: test these properties empirically with Python

Notation: ~ means "follows in distribution"

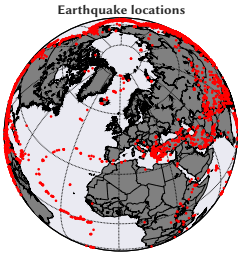
Simulating earthquake occurrences (1/2)

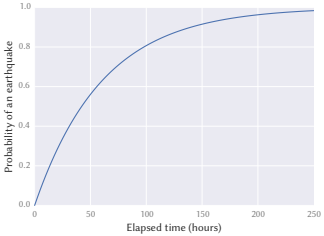
- Suppose we live in an area where there are typically 0.03 earthquakes of intensity 5 or more per year
- Assume earthquake arrival is a Poisson process
  - interval between earthquakes follows an exponential distribution
  - events are independent
- Simulate the random intervals between the next earthquakes of intensity 5 or greater
- What is the 25th percentile of the interval between 5+ earthquakes?

```
> from scipy.stats import expon

> expon(scale=1/0.03).rvs(size=15)
array([23.23763551,  28.73209684,  29.7729332,
       46.66320369, 4.03328973,  84.03262547,
       42.22440297,  14.14994806, 29.90516283,
       87.07194806,  11.25694683,  15.08286603,
       35.72159516,  44.70480237,  44.67294338])

> expon(scale=1/0.03).ppf(0.25)
9.5894024150593644
# answer is “around 10 years”
```

Simulating earthquake occurrences (2/2)

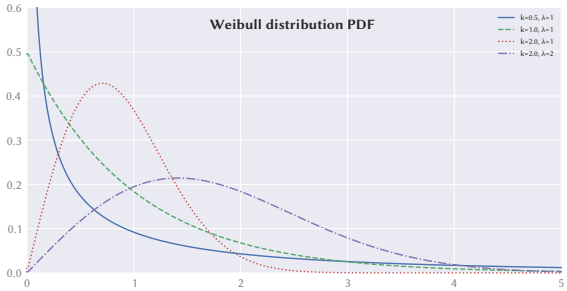
- Worldwide: 144 earthquakes of magnitude 6 or greater in 2013 (one every 60.8 hours on average)
- Rate: $\lambda = \frac{1}{60.8}$ per hour
- What's the probability that an earthquake of magnitude 6 or greater will occur (worldwide) in the next day?
  - right: plot of the CDF of the corresponding exponential distribution
  - `scipy.stats.expon(scale=60.8).cdf(24) = 0.326`





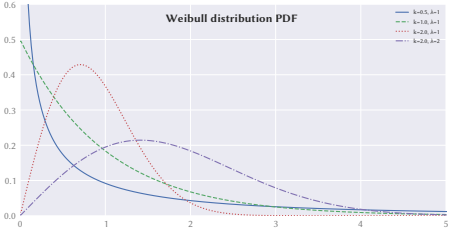
Data source: `earthquake.usgs.gov/earthquakes/search/`

Weibull distribution



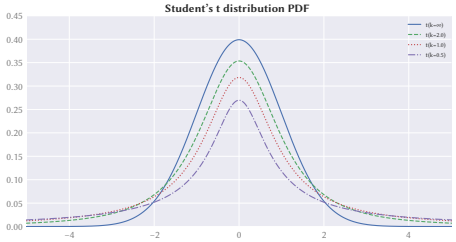
- Very flexible distribution, can model left-skewed, right-skewed, and symmetric data
- Widely used for modeling **reliability data**
- Python: `scipy.stats.dweibull(k, μ, λ)`

Weibull distribution



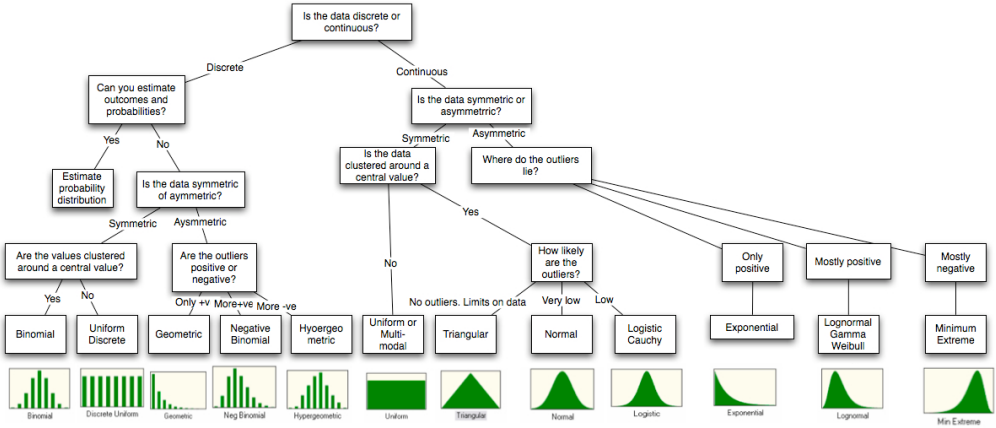
- $k < 1$: the failure rate **decreases over time**
  - significant "infant mortality"
  - defective items failing early and being weeded out of the population
- $k = 1$: the failure rate is **constant over time** (equivalent to an exponential distribution)
  - suggests random external events are causing failure
- $k > 1$: the failure rate **increases with time**
  - "aging" process causes parts to be more likely to fail as time goes on

Student's t distribution



- Symmetric and bell-shaped like the normal distribution, but with heavier tails
- As the number of degrees of freedom *df* grows, the t-distribution approaches the normal distribution with mean 0 and variance 1
- Python: `scipy.stats.t(df)`
- First used by W. S. Gosset (aka Mr Student, 1876-1937), for quality control at Guinness breweries

Distribution choice flowchart



Source of this useful flowchart is uncertain, possibly it is due to Aswath Damodaran, NYU

Image credits

- Dice on slide 39, `flic.kr/p/9SJ5g`, CC BY-NC-ND licence
- Galton board on slide 57: Wikimedia Commons, CC BY-SA licence
- Transistor on slide 61: `flic.kr/p/4d4XSj`, CC BY licence)
- Photo of Mr Gosset (aka Mr Student) on slide 72 from Wikimedia Commons, public domain
- Microscope on slide 73 adapted from `flic.kr/p/aeh1J5`, CC BY licence

For more free content on risk engineering, visit `risk-engineering.org`

RISK ENGINEERING

Further reading

- SciPy lecture notes: `scipy-lectures.org`
- Book *Statistics done wrong*, available online at `statisticsdonewrong.com`
- A gallery of interesting Python notebooks: `github.com/jupyter/jupyter/wiki/A-gallery-of-interesting-Jupyter-Notebooks`

For more free content on risk engineering, visit `risk-engineering.org`

Feedback welcome!

@LearnRiskEng

fb.me/RiskEngineering

Was some of the content unclear? Which parts were most useful to you? Your comments to `feedback@risk-engineering.org` (email) or `@LearnRiskEng` (Twitter) will help us to improve these materials. Thanks!

For more free content on risk engineering, visit `risk-engineering.org`