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
Where does this fit into risk engineering?

- Data
- Probabilistic model
- Event probabilities
- Risks
- Consequence model
- Event consequences
- Decision-making criteria
Where does this fit into risk engineering?

- Data
- Probabilistic model
- Event probabilities
- Risks
- Decision-making
- Consequence model
- Event consequences
- Costs
- Criteria

curve fitting
Where does this fit into risk engineering?

These slides
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 2018!
Data Scientist: The Sexiest Job of the 21st Century

by Thomas H. Davenport and D.J. Patil

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

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...
Installation options

- **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

- **Tablet/cloud** without local installation: Sage Math Cloud
  - Python and Sage running in the cloud
Sage Math Cloud

→ cocalc.com

▷ Runs in cloud, access via browser
▷ No local installation needed
▷ Access to Python in a Jupyter notebook, Sage, R
▷ Create an account for free
▷ Similar tools:
  • Google Colaboratory
  • Microsoft Azure Notebooks
  • pythonanywhere.com
Spyder environment for Python

GUI interface to local Python installation

spyder-ide.org
Jupyter: interact with Python in a web browser

- Jupyter python notebook: interact via your web browser
- Type expressions into a cell then press Ctrl-Enter
- Results appear in the web page
- Jupyter notebooks are live computational documents
  - allow the embedding of rich media (anything that a web browser can display)
  - try online at jupyter.org/try, or install on your computer for free

More on Jupyter at jupyter.org
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
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, 0.07479854, -1.64835235, -4.76771371,  
       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>]

![Graph showing the plot of a sine wave with random noise added.]
A random variable is a set of possible values from a stochastic experiment.

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

\[
p_X(x) \overset{\text{def}}{=} \Pr(X \text{ takes the value } x)
\]

This is called the probability mass function (PMF) of \( X \).

Example: \( X = \) “number of heads when tossing a coin twice”

- \( p_X(0) \overset{\text{def}}{=} \Pr(X = 0) = \frac{1}{4} \)
- \( p_X(1) \overset{\text{def}}{=} \Pr(X = 1) = \frac{2}{4} \)
- \( p_X(2) \overset{\text{def}}{=} \Pr(X = 2) = \frac{1}{4} \)
Task: simulate “expected number of heads when tossing a coin twice”

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

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

inclusive lower bound exclusive upper bound
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
  ```python
  >>> numpy.random.randint(0, 2)
  1
  ```

- Toss a coin twice:
  ```python
  >>> numpy.random.randint(0, 2, 2)
  array([[0, 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
  
  ```python
  > numpy.random.randint(0, 2)
  1
  
  inclusive lower bound exclusive upper bound
  ```

- Toss a coin twice:
  
  ```python
  > numpy.random.randint(0, 2, 2)
  array([0, 1])
  
  count
  ```

- Number of heads when tossing a coin twice:
  
  ```python
  > numpy.random.randint(0, 2, 2).sum()
  1
  ```

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”

- Do this 1000 times and plot the resulting PMF:

```python
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), marker="o")
plt.margins(0.1)
```
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

\[ \sum_x p_X(x) = 1 \]

▷

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

  $$\int_{-\infty}^{\infty} f_X(x) \, dx = 1$$
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:
    \[
    \mathbb{E}[X] = \mu_X \overset{\text{def}}{=} \sum_{i=1}^{N} x_i \times \Pr(X = x_i)
    \]

  - Continuous random variable:
    \[
    \mathbb{E}[X] = \mu_X \overset{\text{def}}{=} \int_{-\infty}^{\infty} u \times f(u)du
    \]

- Interpretation:
  - the center of gravity of the PMF or PDF
  - the average in a large number of independent realizations of your experiment
Illustration: expected value with coins

\[ \begin{align*} \nabla \ X &= \text{“number of heads when tossing a coin twice”} \\
\nabla \ \text{PMF:} & \quad \begin{aligned} p_X(0) & \overset{\text{def}}{=} \Pr(X = 0) = \frac{1}{4} \\
p_X(1) & \overset{\text{def}}{=} \Pr(X = 1) = \frac{2}{4} \\
p_X(2) & \overset{\text{def}}{=} \Pr(X = 2) = \frac{1}{4} \end{aligned} \\
\n\nabla \ \mathbb{E}[X] \overset{\text{def}}{=} \sum_k k \times p_X(k) &= 0 \times \frac{1}{4} + 1 \times \frac{2}{4} + 2 \times \frac{1}{4} = 1 \end{align*} \]
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

```python
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)
```

![Graph showing the expected value of a dice roll converging to 3.5 over a large number of rolls.](Image)
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 $\Omega$ 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:

\[
\text{Var}(X) = \sigma^2_X \overset{\text{def}}{=} \sum_{i=1}^{N} (X_i - \mu_X)^2 \Pr(X = x_i)
\]

▶ For a continuous random variable:

\[
\text{Var}(X) = \sigma^2_X \overset{\text{def}}{=} \int_{-\infty}^{\infty} (x - \mu_x)^2 f(u) du
\]

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

▷ $X = \text{"number of heads when tossing a coin twice"}$

▷ PMF:
- $p_X(0) \overset{\text{def}}{=} \Pr(X = 0) = \frac{1}{4}$
- $p_X(1) \overset{\text{def}}{=} \Pr(X = 1) = \frac{2}{4}$
- $p_X(2) \overset{\text{def}}{=} \Pr(X = 2) = \frac{1}{4}$

▷

\[
\text{Var}(X) \overset{\text{def}}{=} \sum_{i=1}^{N} (x_i - \mu_X)^2 \Pr(X = x_i)
= \frac{1}{4} \times (0 - 1)^2 + \frac{2}{4} \times (1 - 1)^2 + \frac{1}{4} \times (2 - 1)^2 = \frac{1}{2}
\]
Variance of a dice roll

▷ Analytic calculation of the variance of a dice roll:

\[
Var(X) = \sum_{i=1}^{N} (X_i - \mu_X)^2 \Pr(X = x_i)
\]

\[
= \frac{1}{6} \times \left( (1 - 3.5)^2 + (2 - 3.5)^2 + (3 - 3.5)^2 + (4 - 4.5)^2 + (5 - 3.5)^2 + (6 - 3.5)^2 \right)
\]

\[
= 2.916
\]

▷ Let’s reproduce that in Python

```python
> rolls = numpy.random.randint(1, 6, 1000)
> len(rolls)
1000
> rolls.var()
2.9463190000000004
```
Properties of variance as a mathematical operator

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

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

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

Note:
- \( \text{Cov}(X, Y) \overset{\text{def}}{=} \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])] \)
- \( \text{Cov}(X, X) = \text{Var}(X) \)

Beware:
- \( \mathbb{E}[X^2] \neq (\mathbb{E}[X])^2 \)
- \( \mathbb{E}[\sqrt{X}] \neq \sqrt{\mathbb{E}[X]} \)
Standard deviation

- Formula for variance: $\text{Var}(X) \equiv \sum_{i=1}^{N} (X_i - \mu_X)^2 \Pr(X = x_i)$

- If random variable $X$ is expressed in metres, $\text{Var}(X)$ 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 $\sigma \equiv \sqrt{\text{Var}(X)}$

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

```python
> 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

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

```python
>>> 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

▷ \( E[X + Y] = E[X] + E[Y] \)

```python
> 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
```

▷ \( \text{Var}(cX) = c^2 \text{Var}(X) \)

```python
> numpy.random.randint(1, 101, 1000).var()
836.616716
> numpy.random.randint(5, 501, 1000).var()
20514.814318999997
> 5 * 5 * 836
20900
```
Properties of expectation & variance: empirical testing with numpy

▷ \( \mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y] \)

```python
> 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
```

▷ \( \text{Var}(cX) = c^2 \text{Var}(X) \)

```python
> numpy.random.randint(1, 101, 1000).var()
836.616716
> numpy.random.randint(5, 501, 1000).var()
20514.814318999997
> 5 * 5 * 836
20900
```
A percentile measure is a **cutpoint in the probability distribution** indicating the value below which a given percentage of the sample falls.

The 5\(^{th}\) percentile is the \(x\) value which has 5% of the sample smaller than \(x\).

It’s also called the 0.05 quantile.

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

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

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

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

\[ F_X(x) = \Pr(X \leq x) \quad -\infty \leq x \leq \infty \]
\[ = \sum_{x_i \leq x} \Pr(X = x_i) \]

The CDF is built by accumulating probability as \( x \) increases.

Consider the random variable \( X = \text{“number of heads when tossing a coin twice”} \).

<table>
<thead>
<tr>
<th>( x )</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMF, ( p_X(x) = \Pr(X = x) )</td>
<td>( \frac{1}{4} )</td>
<td>( \frac{2}{4} )</td>
<td>( \frac{1}{4} )</td>
</tr>
<tr>
<td>CDF, ( F_X(x) = \Pr(X \leq x) )</td>
<td>( \frac{1}{4} )</td>
<td>( \frac{3}{4} )</td>
<td>1</td>
</tr>
</tbody>
</table>
CDF of a discrete distribution

\[ F_X(x) = \Pr(X \leq x) \quad -\infty \leq x \leq \infty \]
\[ = \sum_{x_i \leq x} \Pr(X = x_i) \]

Example: sum of two dice

<table>
<thead>
<tr>
<th>Sum of two dice: PMF</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 2 4 6 8 10 12</td>
</tr>
<tr>
<td>0.000</td>
</tr>
<tr>
<td>0.025</td>
</tr>
<tr>
<td>0.050</td>
</tr>
<tr>
<td>0.075</td>
</tr>
<tr>
<td>0.100</td>
</tr>
<tr>
<td>0.125</td>
</tr>
<tr>
<td>0.150</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sum of two dice: CDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 2 4 6 8 10 12</td>
</tr>
<tr>
<td>0.0</td>
</tr>
<tr>
<td>0.2</td>
</tr>
<tr>
<td>0.4</td>
</tr>
<tr>
<td>0.6</td>
</tr>
<tr>
<td>0.8</td>
</tr>
<tr>
<td>1.0</td>
</tr>
</tbody>
</table>

\[ \frac{40}{81} \]
CDF of a continuous distribution

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

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

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 \textbf{time to failure} of a component.

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

\[
F(t) = \Pr(T \leq t)
\]
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?
**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 $\text{norm}(3200, 600)$ distribution.

We want $1 - \text{norm}(3200, 600).\text{cdf}(2000)$, 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 50\(^{th}\) percentile).

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

```python
import scipy.stats

distrib = scipy.stats.norm(120, 20)

distrib.ppf(0.5)
120.0

distrib.cdf(120.0)
0.5
```
The quantile function

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)
Simulating dice throws

- Maximum of 1000 throws
  - `dice = scipy.stats.randint(1, 7)`
  - `dice.rvs(1000).max()`
    - 6

- 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
Simulating dice

```python
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

```python
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

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Distribution</th>
<th>Library Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coin tossing with uneven coin</td>
<td>Bernoulli</td>
<td><code>scipy.stats.bernoulli</code></td>
</tr>
<tr>
<td>Rolling a dice</td>
<td>uniform</td>
<td><code>scipy.stats.randint</code></td>
</tr>
<tr>
<td>Counting errors/successes</td>
<td>Binomial</td>
<td><code>scipy.stats.binom</code></td>
</tr>
<tr>
<td>Trying until success</td>
<td>geometric</td>
<td><code>scipy.stats.geom</code></td>
</tr>
<tr>
<td>Countable, rare events whose occurrence is independent</td>
<td>Poisson</td>
<td><code>scipy.stats.poisson</code></td>
</tr>
<tr>
<td>Random “noise”, sums of many variables</td>
<td>normal</td>
<td><code>scipy.stats.norm</code></td>
</tr>
</tbody>
</table>
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
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 $k^{th}$ 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 $n^{th}$ 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”

- $\text{Binomial}(p, k, n)$: 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 $\frac{n!}{k!(n-k)!}$
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?

What is the smallest $k$ such that $P(X \leq k)$ is at least 0.05?

```python
import scipy.stats

# 100 patients, error rate of 0.1
test = scipy.stats.binom(n=100, p=0.1)

test.cdf(1)  # 0.00032168805319411544

test.cdf(10) # 0.5831551226649232

test.ppf(0.05) # 5.0
```
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?
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.

```python
>>> 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.

```python
>>> import scipy.stats
>>> N = 20000
>>> 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: 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.

\[
\text{import scipy.stats}
\]
\[
wells = \text{scipy.stats.binom(n=9, p=0.1)}
\]
\[
wells.pmf(1)
\]
0.38742048899999959

### Answer by simulation

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

\[
\text{import scipy.stats}
\]
\[
N = 20000
\]
\[
wells = \text{scipy.stats.binom(n=9, p=0.1)}
\]
\[
trials = wells.rvs(N)
\]
\[
(trials == 1).sum() / \text{float}(N)
\]
0.38679999999999998

The probability of all 9 wells failing is \( 0.9^9 = 0.3874 \) (and also \( \text{wells.pmf}(0) \)).

The probability of at least 8 wells failing is \( \text{wells.cdf}(1) \). It’s also \( \text{wells.pmf}(0) + \text{wells.pmf}(1) \) (it’s 0.7748).
Binomial distribution: properties

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

- the mean of the distribution ($\mu_x$) is equal to $n \times p$
- the variance ($\sigma^2_x$) is $n \times p \times (1 - p)$
- the standard deviation ($\sigma_x$) 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

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

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 different quantile values of 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 \( \mu \) and standard deviation \( \sigma \), 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 6\( \sigma \) 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!
  - \((\text{scipy.stats.norm.cdf}(6) - \text{scipy.stats.norm.cdf}(-6)) \times 100 \rightarrow 99.99999802682453\)

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:

```python
N = 10000
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 \( \lambda \)
  - \( 1/\lambda \) is the “mean time between failures”, or MTBF
  - \( \lambda \) 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 $\frac{1}{\lambda}$.

```python
import scipy.stats
lda = 25
dist = scipy.stats.expon(scale=1/float(lda))
obs = dist.rvs(size=1000)
obsm = obs.mean()
obsm
0.041137615318791773
obs.std()
obstd
0.03915081431615041
1/float(lda)
0.04
```
An exponentially distributed random variable $T$ obeys

$$\Pr(T > s + t \mid T > s) = \Pr(T > t), \quad \forall s, t \geq 0$$

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.
- $X \sim \text{Exp}(1/5000)$ (mean failure time of 5000 hours)
- $Y \sim \text{Exp}(1/8000)$ (mean failure time of 8000 hours)
- $Z \sim \text{Exp}(1/4000)$ (mean failure time of 4000 hours)
Failure of power transistors (2/2)

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

\[
\Pr(T \leq t) = 1 - \Pr(T > t)
\]
\[
= 1 - \Pr(\min(X, Y, Z) > t)
\]
\[
= 1 - \Pr(X > t, Y > t, Z > t)
\]
\[
= 1 - \Pr(X > t) \times \Pr(Y > t) \times \Pr(Z > t) \quad \text{(independence)}
\]
\[
= 1 - (1 - \Pr(X \leq t)) (1 - \Pr(Y \leq t)) (1 - \Pr(Z \leq t))
\]
\[
= 1 - (1 - (1 - e^{-t/5000})) (1 - (1 - e^{-t/8000})) (1 - (1 - e^{-t/4000})) \quad \text{(exponential CDF)}
\]
\[
= 1 - e^{-t/5000} e^{-t/8000} e^{-t/4000}
\]
\[
= 1 - e^{-t/(\frac{1}{5000} + \frac{1}{8000} + \frac{1}{4000})}
\]
\[
= 1 - e^{-0.000575t}
\]

▷ System failure time is also exponentially distributed, with parameter 0.000575

▷ Expected time to system failure is $1/0.000575 = 1739$ hours
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 $\lambda$ (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
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:

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

- The parameter \( \lambda \) is called the *intensity* of the Poisson distribution
  - increasing \( \lambda \) adds more probability to larger values

- Python: `scipy.stats.poisson(\lambda)`
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

<table>
<thead>
<tr>
<th>Deaths</th>
<th>Occurrences</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>109</td>
</tr>
<tr>
<td>1</td>
<td>65</td>
</tr>
<tr>
<td>2</td>
<td>22</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>&gt;4</td>
<td>0</td>
</tr>
</tbody>
</table>

- 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 $\mu$

▷ Variance of the Poisson distribution is equal to its parameter $\mu$

▷ 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: $\sim$ 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 25\textsuperscript{th} percentile of the interval between 5+ earthquakes?

```python
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
```
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
  - \texttt{scipy.stats.expon(scale=60.8).cdf(24)} = 0.326

Data source: earthquake.usgs.gov/earthquakes/search/
Weibull distribution

- Very flexible distribution (able to model left-skewed, right-skewed, symmetric data)
- Widely used for modeling reliability data
- Python: `scipy.stats.dweibull(k, mu, lambda_)`
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**
  - 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
Image credits

- Dice on slide 39, flíc.kr/p/9SJ5g, CC BY-NC-ND licence
- Galton board on slide 57: Wikimedia Commons, CC BY-SA licence
- Transistor on slide 61: flíc.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 flíc.kr/p/aeH1J5, CC BY licence

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

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

For more free content on risk engineering, visit risk-engineering.org
Feedback welcome!

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