Lecture 3

- Bayes' Theorem and Applications
- Statistical Independence

These are some of the theorems and corollaries that we have learned so far:

- $\forall E\in\mathcal{F}, 0 \leq P(E)\leq 1$
- $P(\Omega)=1$ and $P(\emptyset) = 0 $
- $P(A^c) = P(\overline{A}) = 1 - P(A)$
- If $A\subset B$, then $P(A)\leq P(B)$
- **DeMorgan's Law 1**: $\overline{A\cap B} = \overline{A}\cup\overline{B}$
- **DeMorgan's Law 2**: $\overline{A\cup B} = \overline{A}\cap\overline{B}$
- $P(A\cap B) = P(A) + P(B) - P(A\cup B)$
- If A and B are M.E. then $A\cap B=\emptyset \Rightarrow P(A\cap B) = 0$
- **Conditional Probability**: $P(A|B) = \frac{P(A\cap B)}{P(B)}$, for $P(B)>0$
- **Chain Rules**: $P(A\cap B) = P(A|B)P(B)$ and $P(A\cap B) = P(B|A)P(A)$
- **Multiplication Rule**: $P(\bigcap_{i=1}^n A_i) = P(A_1)P(A_2|A_1)P(A_3|A_1\cap A_2)\dots P\left(A_n|A_1\cap\dots \cap A_{n-1}\right)$
- **Total Probability**: if a set of events $\{C_i\}_{i=1}^n$ are partitions of the sample space Ω, then $P(A) = \sum_{i=1}^n P(A|C_i)P(C_i)$

In [1]:

```
import random
import numpy as np
import numpy.random as npr
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('bmh')
```

In [2]:

```
# Get familiar with npr.choice function
# random.choice
# random.choices
? npr.choice
```

```
Docstring:
choice(a, size=None, replace=True, p=None)

Generates a random sample from a given 1-D array

.. versionadded:: 1.7.0

.. note::
    New code should use the ``choice`` method of a ``default_rng()``
    instance instead; please see the :ref:`random-quick-start`.

Parameters
-----------
a : 1-D array-like or int
    If an ndarray, a random sample is generated from its elements.
    If an int, the random sample is generated as if it were ``np.arange(a)``
size : int or tuple of ints, optional
    Output shape.  If the given shape is, e.g., ``(m, n, k)``, then
    ``m * n * k`` samples are drawn.  Default is None, in which case a
    single value is returned.
replace : boolean, optional
    Whether the sample is with or without replacement. Default is True,
    meaning that a value of ``a`` can be selected multiple times.
p : 1-D array-like, optional
    The probabilities associated with each entry in a.
    If not given, the sample assumes a uniform distribution over all
    entries in ``a``.

Returns
--------
samples : single item or ndarray
    The generated random samples

Raises
-------
ValueError
    If a is an int and less than zero, if a or p are not 1-dimensional,
    if a is an array-like of size 0, if p is not a vector of
    probabilities, if a and p have different lengths, or if
    replace=False and the sample size is greater than the population
    size

See Also
--------
randint, shuffle, permutation
Generator.choice: which should be used in new code

Notes
-----
Setting user-specified probabilities through ``p`` uses a more general but less
efficient sampler than the default. The general sampler produces a different sample
than the optimized sampler even if each element of ``p`` is 1 / len(a).

Sampling random rows from a 2-D array is not possible with this function,
but is possible with `Generator.choice` through its ``axis`` keyword.

Examples
--------
Generate a uniform random sample from np.arange(5) of size 3:

>>> np.random.choice(5, 3)
array([0, 3, 4]) # random
```

```
>>> #This is equivalent to np.random.randint(0,5,3)

Generate a non-uniform random sample from np.arange(5) of size 3:

>>> np.random.choice(5, 3, p=[0.1, 0, 0.3, 0.6, 0])
array([3, 3, 0]) # random

Generate a uniform random sample from np.arange(5) of size 3 without
replacement:

>>> np.random.choice(5, 3, replace=False)
array([3,1,0]) # random
>>> #This is equivalent to np.random.permutation(np.arange(5))[:3]

Generate a non-uniform random sample from np.arange(5) of size
3 without replacement:

>>> np.random.choice(5, 3, replace=False, p=[0.1, 0, 0.3, 0.6, 0])
array([2, 3, 0]) # random

Any of the above can be repeated with an arbitrary array-like
instead of just integers. For instance:

>>> aa_milne_arr = ['pooh', 'rabbit', 'piglet', 'Christopher']
>>> np.random.choice(aa_milne_arr, 5, p=[0.5, 0.1, 0.1, 0.3])
array(['pooh', 'pooh', 'pooh', 'Christopher', 'piglet'], # random
      dtype='<U11')
Type:      builtin_function_or_method
```

In [3]:
```python
# use npr.choice to sample K outcomes with a given distribution, with replacement
Alist = [0,1] # 0 for heads 1 for tails
prob = [0.1,0.9]
npr.choice(Alist, size= 10, p = prob)
```

Out[3]:
```
array([1, 1, 1, 0, 1, 1, 1, 1, 1, 1])
```

In [4]:
```python
# use npr.choice to sample K outcomes without replacement
npr.choice(Alist, size= 2, replace = False, p = prob)
```

Out[4]:
```
array([1, 0])
```

Example 1: Consider the experiment where we select between a fair 6-sided die and a fair 12-sided die at random and flip it once. What is the probability that the die selected was the 12-sided die if face on top was 5?

Let's first solve this problem analytically using Bayes Thm

Let's validate our answers using Python random experiments and simulation

Bayes

In [5]:
```python
num_sims=100_000
dice = ['6-sided','12-sided']
face5 = 0
die12 = 0
for sim in range(num_sims):
    coin = random.choice(dice)
    if coin == '6-sided':
        S = list(range(1,7))
    else:
        S = list(range(1,13))
```

$$P(12\text{side} \mid \text{Face5}) = \frac{P(\text{Face5} \mid 12S)\, P(12S)}{P(\text{Face5})}$$

$$P(\text{Face5}) = P(F5 \mid 12\text{Side})\, P(12\text{Side}) + P(F5 \mid 6\text{Side})\, P(6\text{Side})$$

```python
    roll = random.choice(S)
    if roll == 5:
        face5 += 1
        if coin == '12-sided':
            die12 += 1

print('relative frequency that die 12 given face 5', die12/face5)
```

in relative frequency. we compute

$$P(12side / face 5) = \frac{P(12side \cap face5)}{P(face 5)} \cong \frac{\#N(12side \& 5f)}{\#(face 5)}$$

```
relative frequency that die 12 given face 5 0.3391921309360639
```

Example 2: A magician has two coins coins, one fair and one 2-headed coin. Consider the experiment where she picks one coin at random and flips it i times. Let H_i denote the event that the outcome of flip i is heads.

- Given the first two flips are heads, what is the probability that the third roll is also heads?

Let's first solve this analytically.

Implement a simulation to solve $P(H_3|H_1\cap H_2)$

In [6]:
```python
# complete in class
heads12 = 0
heads123= 0
num_sims = 10_000
for sim in range(num_sims):
    coin = random.choice(['F','2H'])
    if coin == 'F':
        S = [1,0] # 1 for heads, 0 for tails.
    else:
        S = [1,1] # two headed
    values = random.choices(S, k=3)
    #[1,0,1] for fair coin. [1,1,1]
    if values[0] ==1 and values[1] ==1:
        heads12 += 1

    if values[0] ==1 and values[1] ==1 and values[2]==1:
        heads123 +=1
print('rel. freq. of seeing H3 given H1 H2', heads123/heads12)
```

```
rel. freq. of seeing H3 given H1 H2 0.8965682362330407
```

In [ ]:

previous lecture. $P(F \mid H_1 \cap H_2) = \frac{1}{5}$. $P(2H \mid H_1 \cap H_2) = \frac{4}{5}$

$$P(H_3 \mid H_1 \cap H_2) = P(H_3 \mid F)\, P(F \mid H_1 \cap H_2) + P(H_3 \mid 2H)\, P(2H \mid H_1 \cap H_2)$$

$$= \frac{1}{2} \times \frac{1}{5} + 1 \times \frac{4}{5} = \frac{9}{10}.$$

in lecture we also computed the posterior

$$P(F \mid H_1 \cap H_2 \cap H_3) = \frac{P(H_3 \cap F \cap H_1 \cap H_2)}{P(H_1 \cap H_2 \cap H_3)} \quad *$$

① $P(H_3 \cap F \cap H_2 \cap H_1) = P(H_3 \mid F \cap H_1 \cap H_2)\, P(F \cap H_1 \cap H_2)$

$$= P(H_3 \mid F \cap H_1 \cap H_2) \cdot P(F \mid H_1 \cap H_2)\, P(H_1 \cap H_2)$$

② $P(H_3 \cap H_1 \cap H_2) = P(H_3 \mid H_1 \cap H_2)\, P(H_1 \cap H_2)$

$$(*) = \frac{①}{②} = \frac{\overset{= P(H_3 \mid F)}{P(H_3 \mid F \cap H_1 \cap H_2)}\, \overset{\frac{1}{5}}{P(F \mid H_1 \cap H_2)}\, \cancel{P(H_1 \cap H_2)}}{P(H_3 \mid H_1 \cap H_2)\, \cancel{P(H_1 \cap H_2)}}$$

$$= P(H_3 \mid F)\, P(F \mid H_1 \cap H_2) + P(H_3 \mid 2H)\, P(2H \mid H_1 \cap H_2)$$

$$= \frac{\frac{1}{2} \times \frac{1}{5}}{\frac{1}{2} \times \frac{1}{5} + 1 \times \frac{4}{5}} = \frac{1}{9}.$$