Random Numbers

Concept Module 5

Template by: Laurent Lessard

Why generate random numbers?

1. Use randomness in your code.
   Example: random centroid initialization in K-Means.
2. Generate random samples.
   Example: pick 10 points at random from a data set
3. Generate random data to test if code is working correctly.

Uniform random numbers

- The function `numpy.random.rand()` generates a random number uniformly in the interval $0.0 \leq x < 1.0$ (all numbers are equally likely)

```
# Import numpy only

import numpy as np

np.random.rand()
```

```
0.417022004702574
```

```
# Directly import function rand

from numpy.random import rand

rand()
```

```
0.7203244934421581
```

Uniform random numbers

- Specifying arguments to `np.random.rand()` creates a numpy array (`np.array`) of the specified size.

```
from numpy.random import rand

# make a 1D array of size 3
rand(3)
```

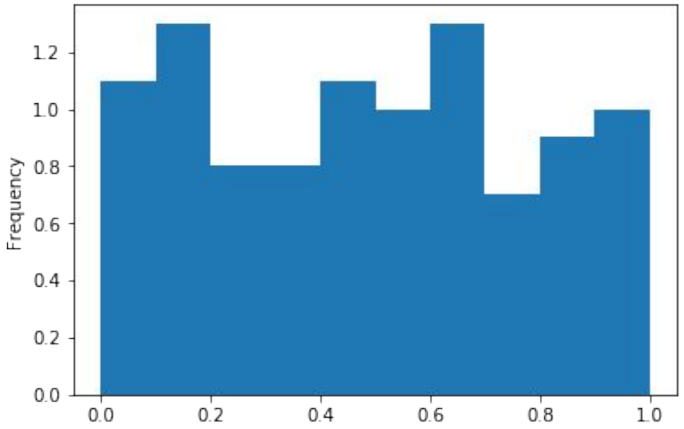
```
array([ 0.43538468,  0.60830368,  0.7106929 ])
```

```
# make a 2D array of size 2x3
rand(2,3)
```

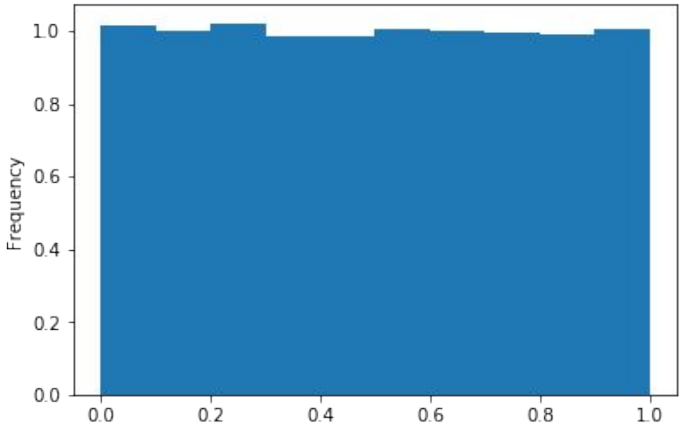
```
array([[ 0.90869999,  0.99930707,  0.41091365],
       [ 0.88678767,  0.74792913,  0.80540392]])
```

Check uniformity

```
# Make a histogram with 100 samples
import numpy as np
import pandas as pd

s = pd.Series( np.random.rand(100) )

s.plot.hist( range=(0,1), density=True )
```



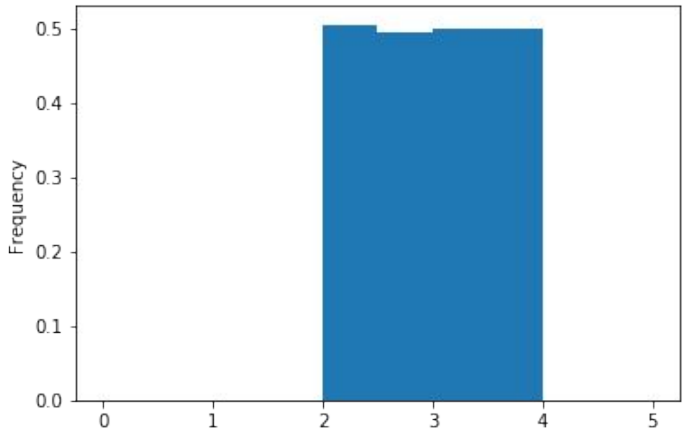

```
# Make a histogram with 100,000 samples
import numpy as np
import pandas as pd

s = pd.Series( np.random.rand(100000) )

s.plot.hist( range=(0,1), density=True )
```



Arbitrary uniform intervals

- if x is uniform in the interval (0,1), then `(b-a)*x+a` will be uniform in the interval (a,b).

```
# Uniform between 2 and 4
from numpy.random import rand
import pandas as pd

a,b = 2,4
s = pd.Series( (b-a)*rand(100000) + a )

s.plot.hist( range=(0,5), density=True )
```



Normal random numbers

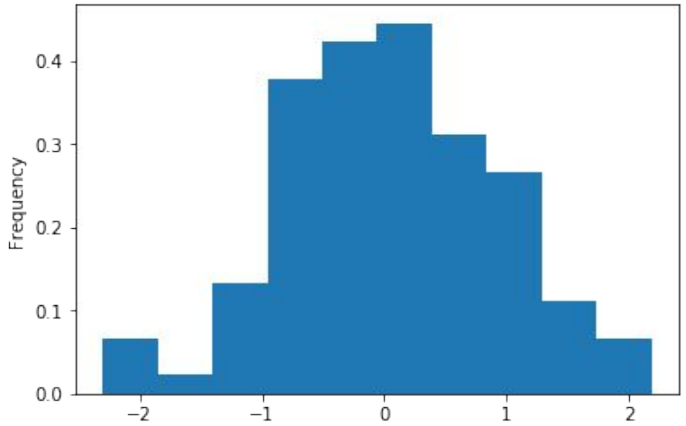
- `numpy.random.randn()` generates a random normally distributed number with mean 0 and standard deviation 1.
- Parameters are the same as with `numpy.random.rand()`

```
import numpy as np

np.random.randn(3,4)
```

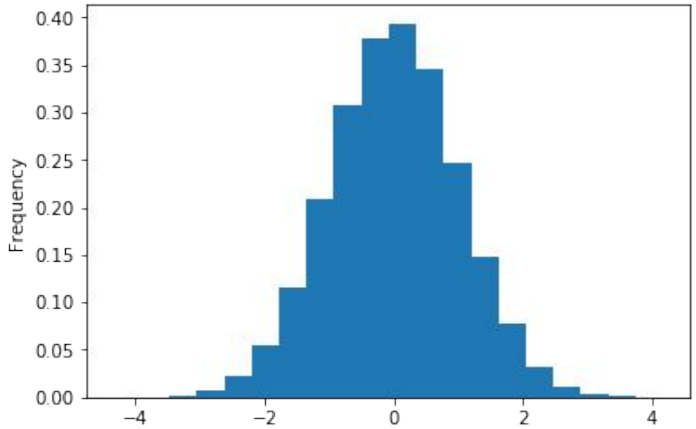
```
array([[ 1.62434536, -0.61175641, -0.52817175, -1.07296862],
       [ 0.86540763, -2.3015387 ,  1.74481176, -0.7612069 ],
       [ 0.3190391 , -0.24937038,  1.46210794, -2.06014071]])
```

Check normality

```
# Make a histogram with 100 samples
import numpy as np
import pandas as pd

s = pd.Series( np.random.randn(100) )

s.plot.hist( density=True )
```



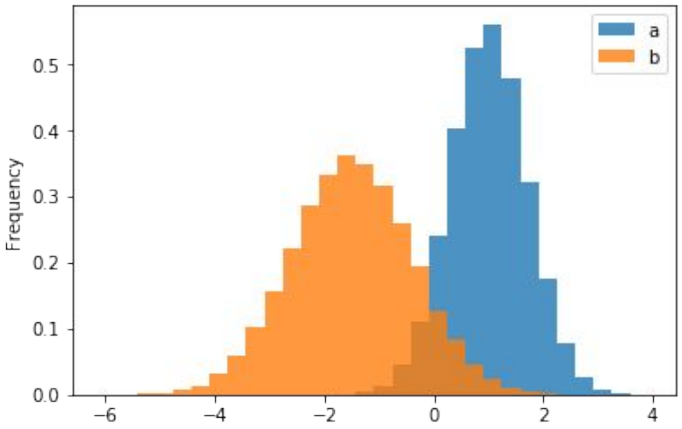

```
# Make a histogram with 100,000 samples
import numpy as np
import pandas as pd

s = pd.Series( np.random.rand(100000) )

s.plot.hist( bins=20, density=True )
```



Rescaling normal variables

- if x is normal with mean 0 and standard deviation 1, then s*x+m is normal with mean m and standard deviation s.

```
# Make two overlapping normals!
from numpy.random import randn
import pandas as pd

n = 100000
df = pd.DataFrame( {'a': 0.7*randn(n)+1.0,
                    'b': 1.1*randn(n)-1.5 } )

df.plot.hist(bins=30, density=True, alpha=0.8)
```



Random integers

- The function `numpy.random.randint(a,b)` generates a random integer a ≤ x < b. All integers are equally likely.
- Size can be included using the `size` named argument.

```
8 random integers from 1 to 5
import numpy as np

np.random.randint(1,6,size=8)
```

```
array([5, 1, 4, 4, 4, 2, 4, 3])
```

```
3x6 array of rand ints from -3 to 2
import numpy as np

np.random.randint(-3,3,size=(3,6))
```

```
array([[ 1,  2, -3,  0,  0,  0],
       [-2,  0,  2, -1,  1, -3],
       [-3,  1, -1, -2, -3, -2]])
```

Random integers as indices

- Numpy arrays can be indexed using arrays

```
numbers = np.random.rand(5)   # make a list of numbers
```

```
array([ 0.79172504,  0.52889492,  0.56804456,  0.92559664,  0.07103606])
```

```
indices = np.random.randint(5,size=4)   # make a list of indices
```

```
array([1, 1, 0, 2])
```

```
numbers[indices]   # index using indices!
```

```
array([ 0.52889492,  0.52889492,  0.79172504,  0.56804456])
```

Randomly sampling a DataFrame

- One method: pick random indices, then index!

```
# pick 5 rows at random (may have repetitions!)
ix = np.random.randint(0,10, size=5)

df.iloc[ix]
```

	age
David	26
David	26
Harry	18
Alice	23
Bob	29

	age
Alice	23
Bob	29
Carol	30
David	26
Emily	27
Fred	29
Gretchen	23
Harry	18
Isabelle	18
John	19

Randomly sampling a DataFrame

- Better method: use `pd.DataFrame.sample()`.

```
# pick 5 DISTINCT rows at random

df.sample(5)
```

	age
David	26
Carol	30
Fred	29
John	19
Harry	18

```
# pick 5 rows with replacement

df.sample(5, replace=True)
```

	age
Alice	23
Gretchen	23
John	19
John	19
Harry	18

	age
Alice	23
Bob	29
Carol	30
David	26
Emily	27
Fred	29
Gretchen	23
Harry	18
Isabelle	18
John	19

Shuffling a list

- The function `np.random.shuffle()` rearranges a list in random order (this actually modifies the list).

```
# make a list of names and shuffle it!

names = ['Alice','Bob','Carol','David','Emily','Fred','Gretchen','Harry']
np.random.shuffle(names)
print(names)
```

```
['Harry', 'Gretchen', 'Alice', 'David', 'Fred', 'Carol', 'Bob', 'Emily']
```

Example: Deck of cards

```
suits = ['♠', '♣', '♡', '♢']  # google "unicode characters"
values = [str(i) for i in range(2,11)] + ['J', 'Q', 'K', 'A']
deck = [v+s for v in values for s in suits]  # double list comprehension!
print(deck)
```

```
['2♠', '3♠', '4♠', '5♠', '6♠', '7♠', '8♠', '9♠', '10♠', 'J♠',
'Q♠', 'K♠', 'A♠', '2♣', '3♣', '4♣', '5♣', '6♣', '7♣', '8♣',
'9♣', '10♣', 'J♣', 'Q♣', 'K♣', 'A♣', '2♡', '3♡', '4♡', '5♡', '6♡',
'7♡', '8♡', '9♡', '10♡', 'J♡', 'Q♡', 'K♡', 'A♡', '2♢', '3♢', '4♢', '5♢',
'6♢', '7♢', '8♢', '9♢', '10♢', 'J♢', 'Q♢', 'K♢', 'A♢']
```

```
# shuffle the deck and deal a 7-card hand
np.random.shuffle(deck)
hand = deck[:7]
print(hand)
```

```
['6♠', 'A♣', '5♠', 'Q♡', '5♢', '3♣', 'Q♢']
```

Warning: Random isn't really random!

- There is no such thing as a truly random number. Python uses a pseudorandom number generator (PRNG).
- Given a number (called the seed), the sequence of random numbers subsequently generated is uniquely determined (not actually random!)
- Using the same seed produces the same random numbers.
- If you don't set the seed, Python will typically use the computer clock time as a seed.

Warning: Random isn't really random!

- Seed can be set using `np.random.seed()`

```
# set the seed to "1"
np.random.seed(1)
print( np.random.randn(5) )

# set the seed to "1" again
np.random.seed(1)
print( np.random.randn(3) )
```

```
[ 1.62434536 -0.61175641 -0.52817175 -1.07296862  0.86540763]
[ 1.62434536 -0.61175641 -0.52817175]
```

Best practices when using randomness

Anytime you write code that involves randomness (picking numbers, randomly shuffling, randomly sampling) you should **always** set the random seed at the beginning.

1. Other people will get the same results you did if they run your code.
2. YOU will get the same results you did if you run your code again.
3. Makes debugging (finding errors) easier because code is reproducible.

Summary

- Generating random numbers
  - Uniform random floats: `np.random.rand()`
  - Normal random floats: `np.random.randn()`
  - Random integers: `np.random.randint()`
- Sampling
  - numpy arrays can be indexed using arrays
  - Randomly sample a DataFrame: `pd.DataFrame.sample()`
  - Randomly shuffle a list in-place: `np.random.shuffle()`
- Random seed
  - Always set the seed before randomizing! `np.random.seed()`