Week 5: Simulation Modeling

Session 9: Probabilistic Sampling

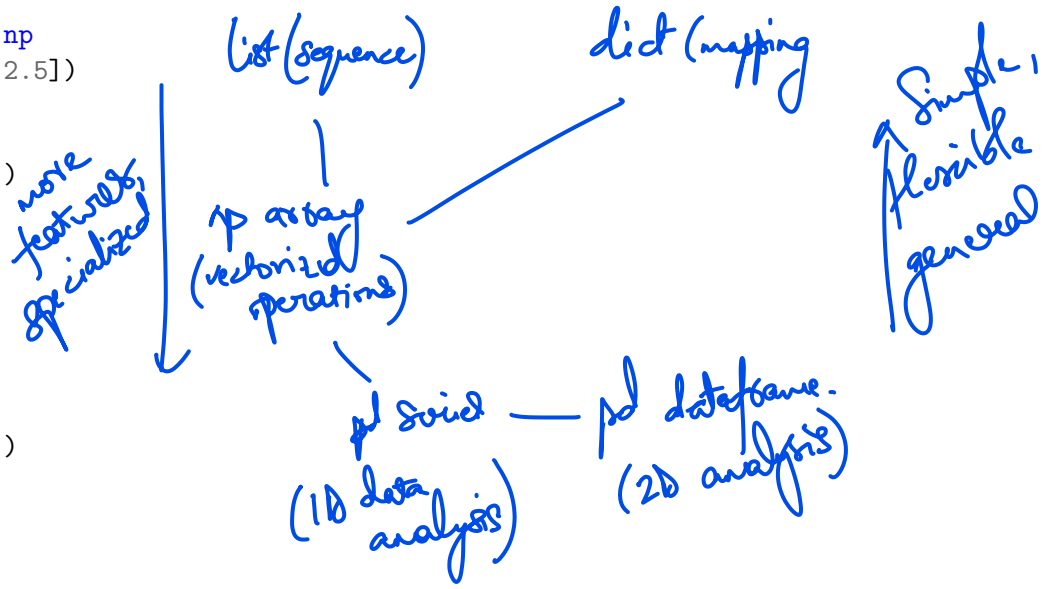
Simulation is a technique for generating realistic data when real data is difficult (or impossible) to obtain. It is based on combining algorithmic thinking with probabilistic sampling. The latter is the focus of this session.

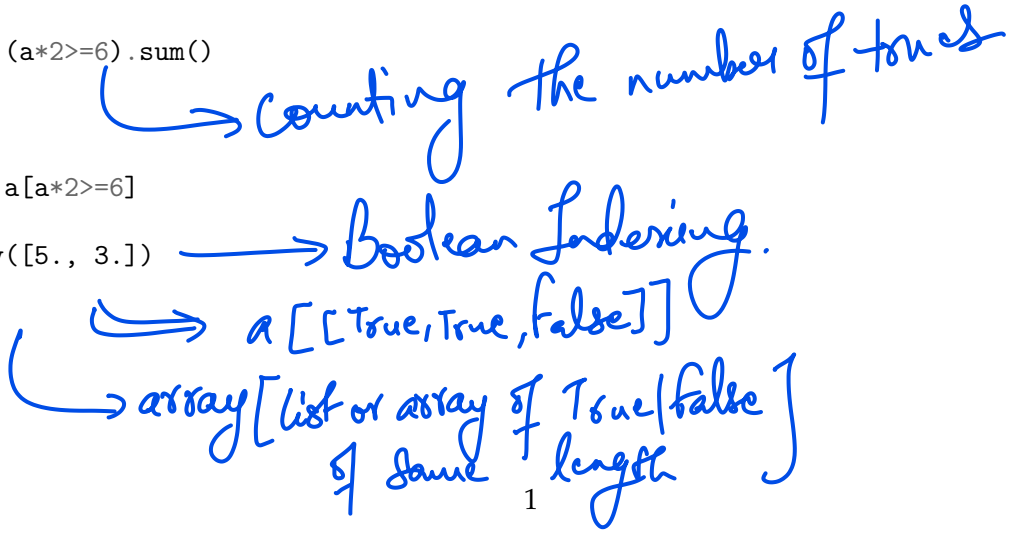
In the context of prescriptive analytics, simulation helps decision makers to rigorously quantify the trade-offs between possible decisions in the presence of uncertainties in the environment. It is especially important if large scale experimentation is prohibitively costly.

Brief Overview of the `numpy` and `pandas` Packages

`numpy` stands for numerical Python, and it was created in 2005 to provide Python with the capability to do fast **vectorized operations**, which means computations on entire vectors or matrices of numbers. The key data structure is the Numpy Array, which is similar to a `list` but has additional capabilities.

```
[1]: import numpy as np
     a=np.array([5,3,2.5])
     a

array([5. , 3. , 2.5])

[2]: a[0]

5.0

[3]: a*2

array([10.,  6.,  5.])

[4]: a.mean()

3.5

[5]: a*2>=6

array([ True,  True, False])

[6]: (a*2>=6) & (a<5)

array([False,  True, False])

[7]: (a*2>=6).sum()

2

[8]: a[a*2>=6]

array([5., 3.])
```





pandas is a newer package based on numpy in the back-end, and it provides additional capabilities for data analysis. The key data structures are the **Pandas Series** and the **Pandas DataFrame**. Series is for one-dimensional data, whereas DataFrame is for two-dimensional data (i.e. tables).

Beyond replicating what numpy arrays can do, Pandas data structures provide the additional ability to handle datas involving labels, as well as to filter, aggregate, rearrange, and plot data easily.

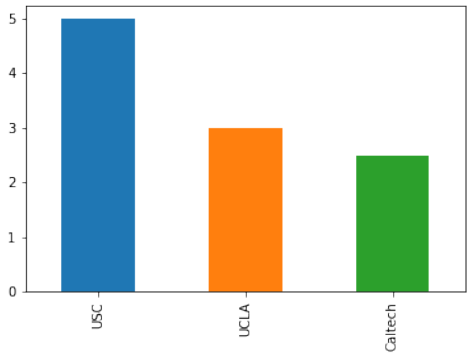
```
[9]: import pandas as pd
     s=pd.Series(a)
     s
```

```
0    5.0
1    3.0
2    2.5
dtype: float64
```

```
[10]: import pandas as pd
      s=pd.Series(a,index=['USC','UCLA','Caltech'])
      s
```

```
USC        5.0
UCLA       3.0
Caltech    2.5
dtype: float64
```

```
[35]: s.plot(kind='bar')
      import matplotlib.pyplot as plt
      plt.show()
```




```
[12]: # .loc for indexing by label
      s.loc['USC']
```

```
5.0
```

```
[13]: # .iloc for indexing by position, as in a Python list.
      s.iloc[1]
```

```
3.0
```

In-class exercise

Using vectorized operations, write one line of code to count the number of numbers at are between 5 and 10 (including the end points) in the following Series.

Then modify the line to count the proportion of such numbers.

```
[14]: import pandas as pd
      nums=pd.Series([3,5,30,20,10.5,9.5,10,4,6,11])
```

Sampling from a Simple Distribution

```
[17]: from numpy.random import default_rng
      rng=default_rng(seed=0)
      rng.normal(20,10)
```

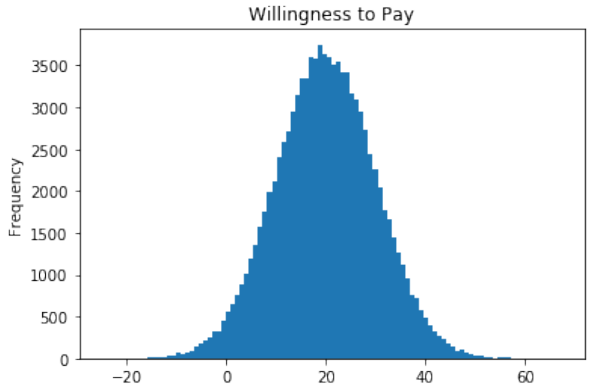
creates random number generator.

```
21.257302210933933
```

```
[18]: import pandas as pd
      valueList=pd.Series(rng.normal(20,10,size=100000))
      valueList.head()
```

```
0    18.678951
1    26.404227
2    21.049001
3    14.643306
4    23.615951
dtype: float64
```

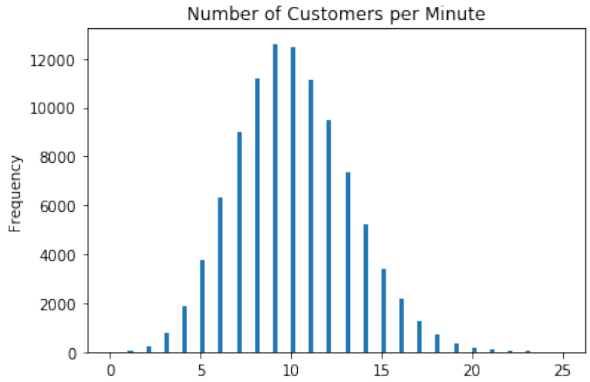
```
[19]: valueList.plot(kind='hist',bins=100,title='Willingness to Pay')
      import matplotlib.pyplot as plt
      plt.show()
```




```
[20]: demand=pd.Series(rng.poisson(10,size=100000))
      demand.head()
```

```
0    10
1     6
2    11
3     6
4    10
dtype: int64
```

[21]:
```
demand.plot(kind='hist',bins=100,title='Number of Customers per Minute')
plt.show()
```



[22]:
```
coin=pd.Series(rng.binomial(1,0.5,size=10))
coin
```

```
0    0
1    1
2    0
3    0
4    1
5    1
6    1
7    0
8    0
9    0
dtype: int64
```

[23]:
```
order=pd.Series(rng.choice(['Noodles','Rice','Sandwitch'],p=[0.4,0.3,0.3],size=100))
order.head()
```

```
0         Rice
1         Rice
2      Noodles
3    Sandwitch
4         Rice
dtype: object
```

[24]:
```
order.value_counts().plot(kind='bar',title='Customer Orders')
plt.show()
```

rng.distribution(parameters, size)

Common Probability Distributions

rng.something
e.g. rng.normal

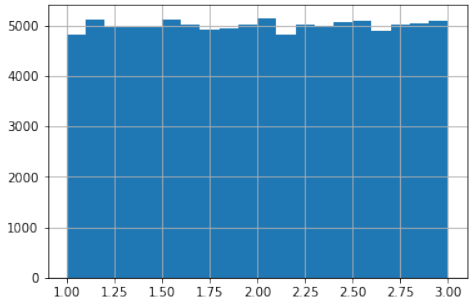
rng function	Parameters	Description
`normal`	`loc`(mean) , `scale` (std)	Bell curve with given mean and standard deviation.
`binomial`	`n`, `p`	# of heads among n independent coin tosses. (Probability of heads is p.)
`uniform`	`low`, `high`	Equally likely to be any decimal number in this range.
`poisson`	`lam` (mean)	# of independent events within a period; each event occurs at a random time.
`geometric`	`p`	# of coin tosses before landing a heads. (Probability of heads is p.)

In addition, `rng.choice` can be used to sample from any given list at the given probabilities.

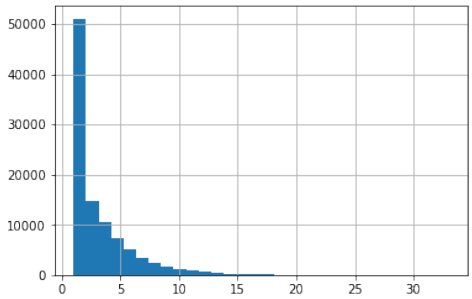
Exercise 5.1 Sampling from Simple Distributions using Numpy

Download the Jupyter notebook attached to the Blackboard link for this exercise and submit it there after completing it. The notebook contains the following questions.

a) Generate 100,000 samples from a uniform distribution between 1 and 3, and plot the histogram with 20 bins, as below.



b) Generate 100,000 samples from a geometric probability with $p = 0.3$, and plot the histogram with 30 bins, as below.



c) Suppose that the number of customers arriving to a grocery store every 10 minutes is Poisson distributed with mean 8.5. Estimate the chance that the number of arrivals in a given 10 minute interval is 15 or higher.

Hint: Generate 1,000,000 samples from the Poisson distribution and apply the techniques from the earlier in-class exercise to count the proportion of samples that are at least 15.

```
0.027507
```

d) Suppose there are two products, and a customer's willingness to pay for each of the two products are drawn independently from the following distributions:

- Product 1: Normal(25,10)
- Product 2: Normal(20,15)

Estimate the proportion of customers whose willingness to pay is higher for the second product than the first, AND whose willingness to pay for the second product is at least zero.

Hint: Draw 1,000,000 samples from each distribution and compare the two Series using vectorized operations as in the earlier in-class exercise.

```
0.391546
```

Sampling from Complex Distributions using Algorithmic Thinking

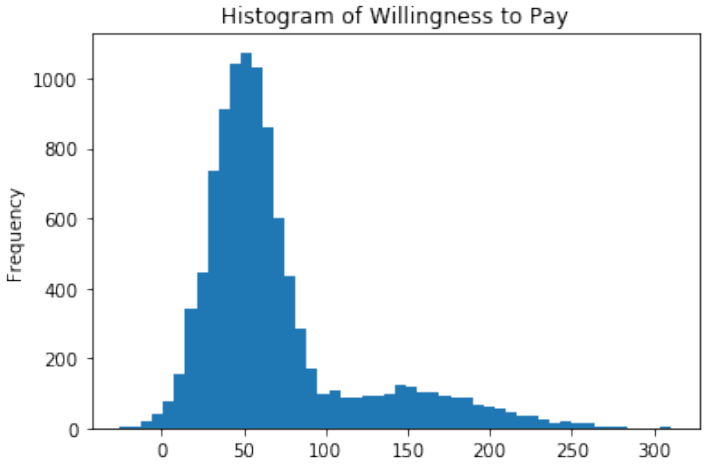
Example 1: Mixture of Distributions

Suppose that there are two market segments, "A" and "B", and the willingness to pay for a product depends on the segment:

- Segment A: Normal(150,50)
- Segment B: Normal(50,20)

Moreover, each customer has a 0.2 chance of being in segment A and 0.8 chance of being in segment B. The following code generates a simulated dataset of the willingness to pay of 10,000 customers. Similar code logic can be used to sample from any mixture of simple distributions.

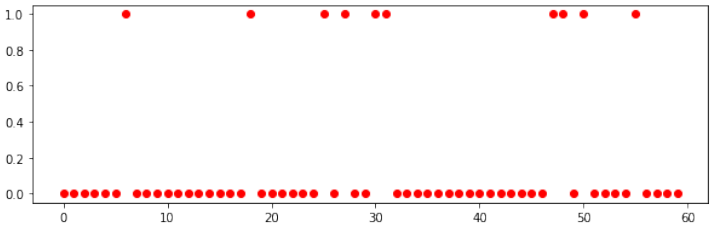
[29]:
```
import pandas as pd
import matplotlib.pyplot as plt
from numpy.random import default_rng
rng=default_rng()
data=[]
for i in range(10000):
    segment=rng.choice(['A','B'],p=[0.2,0.8])
    if segment=='A':
        data.append(rng.normal(150,50))
    else:
        data.append(rng.normal(50,20))
valuations=pd.Series(data)
valuations.plot(kind='hist',bins=50,title='Histogram of Willingness to Pay')
plt.show()
```



Example 2: Serial Correlation

Suppose we want to simulate whether it will rain for 60 days. Knowing that the overall proportion of rainy days is 0.2, the first attempt might be as follows.

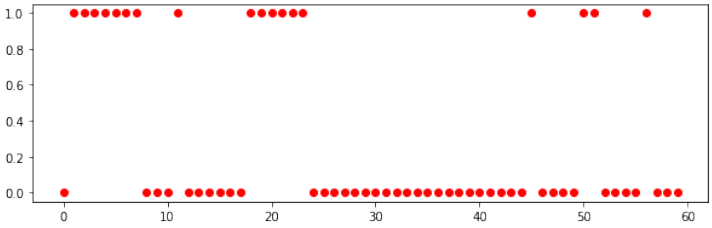
```
[30]: rain=pd.Series(rng.binomial(1,0.2,size=60))
      rain.plot(style='ro',figsize=(10,3))
      plt.show()
```



However, it might be more realistic to take into account the correlation in whether between two consecutive days. Suppose that the probability whether it will rain today is:

- 0.6 if it rained yesterday.
- 0.1 if it did not rain yesterday.

```
[31]: data=[rng.binomial(1,0.2)]
      for i in range(1,60):
          if data[-1]==1:
              data.append(rng.binomial(1,0.6))
          else:
              data.append(rng.binomial(1,0.1))
      rain=pd.Series(data)
      rain.plot(style='ro',figsize=(10,3))
      plt.show()
```



Exercise 5.2 Accounting for Uncertain Product Quality when Forecasting Demand

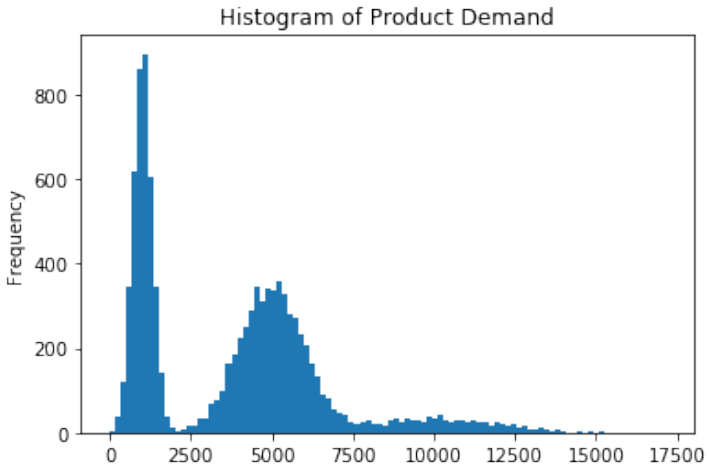
Download the Jupyter notebook attached to the Blackboard link for this exercise and submit it there after completing it. The notebook contains the following question.

You are tasked with forecasting demand for a new product. Based on past data and your knowledge of the product, you estimate that the product quality will be Amazing with probability 0.1, Mediocre with probability 0.5, and Terrible with probability 0.4. You model the demand as normally distributed, with mean and standard deviation dependent on the product quality as follows.

Prod. Quality:	Amazing	Mediocre	Terrible
mean	10000	5000	1000
standard deviation	2000	1000	300

Create a `Series` called "forecast" with 10,000 samples of the demand forecast, and compute the mean and standard deviation of the samples, as well as the probability that demand is more than 6000. Finally, plot a histogram of the samples with 100 bins. (For this question, you can ignore the constraint that demand should be integer and non-negative.) Due to randomness, your estimates and plot may be slightly different, but they should be roughly the same as the sample output below.

```
Sample mean: 3844.004687406584
Sample standard deviation: 2883.2215386137937
Probability demand more than 6000: 0.1679
```



Exercise 5.3 Simulating Serially Correlated Demand

Download the Jupyter notebook attached to the Blackboard link for this exercise and submit it there after completing it. The notebook contains the following question.

A logistic company would like to simulate demand for a given product. Assume that there are Good or Bad weeks. On a Good week, the demand is Normally distributed with mean 200 and standard deviation 50. On a Bad week, demand is Normally distributed with mean 100 and standard deviation 30. Moreover, you should round the demand to the nearest integer and set it to zero if it is ever negative.

Whether a week is Good or Bad is not independent across time. Conditional on a given week being Good, the next week remains Good with probability 0.9. Similarly, conditional on a given week being Bad, the next week remains Bad with probability 0.9.

Write Python code to generate a Series called "demand" for 100 weeks, assuming that the first week is Good. You should plot the demand over time as well. Due to randomness, you are NOT expected to reproduce the exact same plot.