ISYE6644-Simulation and Modeling

Final Report

Topic 18: Random Variate Generation

Group 59 - Ting Yi Lee

Abstract

In this project, I made a library of various random variate generation routines (functions) in Python. The routines (functions) generate the desired number of random variates based on the parameter input values.

User's Guide

This library is not published yet, so your first step is to run the entire Python program to install the necessary packages and the random variate generation functions. You may have to download the package to your environment first. After that, find the function for the specific distribution. Next, understand what parameters are required for the function. Lastly, call the functions with your parameter inputs to get the random variates from the selected distribution.

Background

The starting place for random variate generation is usually the generation of random numbers, which are random variates that are uniformly distributed on the interval from 0 to 1 . So I am going to discuss the uniform random variate generation first before proceeding with the discrete random variate generation.

Methods / Findings

Continuous Distribution 1 # Uniform distribution

The statistical meaning is that within a predefined interval, all outcomes have the same probability of occurring. Real-life examples include the probability of each side of a six-sided dice; the probability of the head vs. tail of a coin; the probability of each shape of a deck of cards.

Parameter n is the number of outcomes that has the same probability of occurring. And parameter x is the number of samples, which is the number of outputs that you want to get.

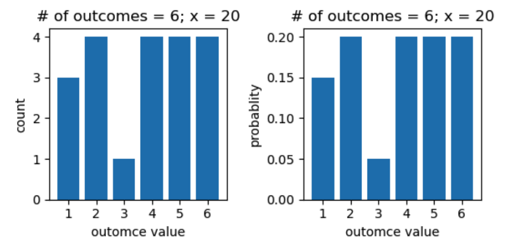
```
# generate random numbers that are uniformly distributed on the interval from 0 to 1
def generate_prn():
    return random.random()

# generate x uniform random variate for n equally likely outcomes
def generate_uniform_rv(n, x):
    outputs = []

    for i in range(x):
        intervals = np.array([j/n for j in range(1,n+1)])
        prn = generate_prn()

        first__guess = sum(prn >= intervals)
        if sum(prn == intervals) == 0: # prn is not on the boundary
            final_guess = first__guess + 1
        else: # prn is on the boundary
            final_guess = first__guess

        outputs.append(final_guess)

    return outputs
```

The example has six outcomes with the same probability of occurring. I presented two outputs by different sample sizes - the left one is a small size with 10 samples versus the right one is a large size with 1000 samples. The probability bar chart shows that each outcome has a similar probability as the sample size increases.

```
# draw_uniform_rv() runs the main generation function and creates chart(s) using the random variate
output, which provides supplementary information. More details in the complete code.
```

```
uniform_outcomes = 6
```

```
# small sample size
uniform_size = 20

uniform_rv_outputs = draw_uniform_rv(n=uniform_outcomes, x=uniform_size)
```

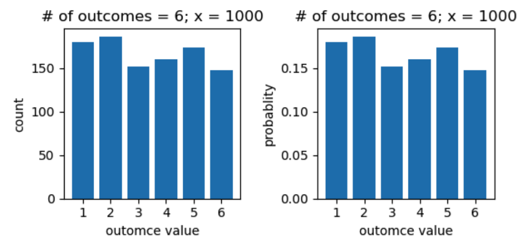


```
uniform_rv_outputs
```

```
[4, 5, 6, 2, 4, 1, 5, 2, 5, 4, 2, 4, 6, 1, 3, 6, 6, 1, 2, 5]
```

```
uniform_outcomes = 6
```

```
# big sample size
uniform_size = 1000

uniform_rv_outputs = draw_uniform_rv(n=uniform_outcomes, x=uniform_size)
```



```
print(uniform_rv_outputs)
```

```
[3, 4, 3, 5, 2, 2, 5, 5, 6, 5, 2, 4, 6, 2, 3, 4, 6, 6, 6, 6, 6, 5, 1, 4, 1, 6,
3, 2, 2, 2, 6, 1, 5, 1, 2, 5, 3, 1, 5, 3, 1, 1, 4, 5, 3, 6, 4, 1, 5, 6, 2, 4, 5
1, 2, 3, 2, 2, 1, 1, 1, 6, 3, 5, 5, 6, 2, 4, 5, 2, 1, 4, 3, 2, 5, 5, 4, 5, 5, 2
6, 6, 2, 2, 2, 2, 6, 2, 4, 5, 1, 3, 5, 6, 1, 6, 6, 6, 2, 4, 4, 2, 4, 1, 6, 4, 1
4, 3, 4, 2, 2, 1, 3, 4, 3, 2, 3, 6, 2, 6, 1, 3, 1, 4, 6, 1, 4, 5, 1, 5, 1, 2, 5
```

Discrete Distribution 1 # Bernoulli distribution

The statistical meaning is that a discrete probability distribution gives only two possible results in a trial (aka an experiment). Real-life examples include the probability of head versus tail of flipping a coin; the probability of donated versus ignored when a charity asks for donations.

Parameter p is the probability of one of the two possible outcomes. And parameter x is the number of outputs that you want to get.

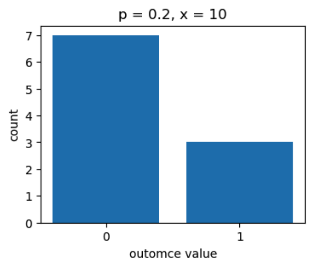
```
# generate x bernoulli random variates i.e. trials with probability p
def generate_bernoulli_rv(p, x):
    outputs = []
    prns = []

    for i in range(x):
        prn = generate_prn()
        prns.append(prn)

        if prn < 1-p:
            outputs.append(0)
        else:
            outputs.append(1)

    return outputs, prns
```

The example is in a trial of two possible results, one result has a probability of 0.2. I presented two outputs by different sample sizes - the left one is a small size with 10 samples versus the right one is a large size with 1000 samples (random variate output being truncated).

```
# draw_bernoulli_rv() runs the main generation function and creates chart(s) using the random
variate output, which provides supplementary information. More details in the complete code.
```

```
bernoulli_p = 0.2
```

```
# small sample size
bernoulli_size = 10

bernoulli_rv_outputs = draw_bernoulli_rv(p=bernoulli_p, x=bernoulli_size)
```

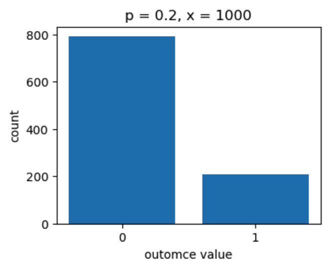


```
print(bernoulli_rv_outputs)
```

```
[0, 0, 0, 0, 0, 0, 0, 1, 1, 1]
```

```
bernoulli_p = 0.2
```

```
# big sample size
bernoulli_size = 1000

bernoulli_rv_outputs = draw_bernoulli_rv(p=bernoulli_p, x=bernoulli_size)
```



```
print(bernoulli_rv_outputs)
```

```
[0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0,
0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
```

Discrete Distribution 2 # Binomial distribution

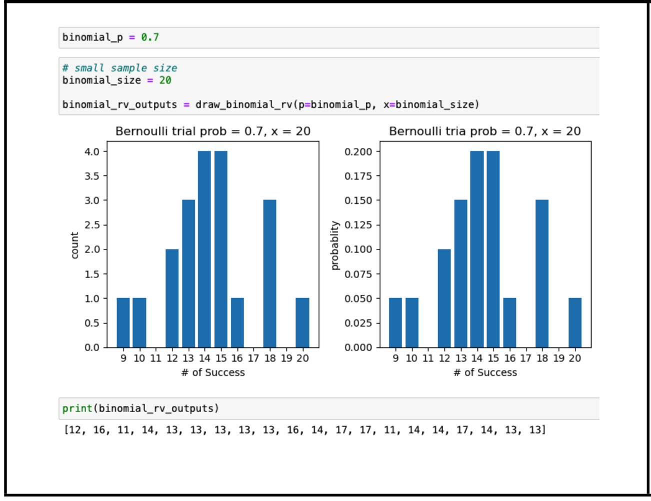
The statistical meaning is that for an experiment that gives only 'Success' and 'Failure' outcomes (aka Bernoulli trial), the number of ‘Success’ in a sequence of X experiments (aka Bernoulli trials). Real-life examples include the number of heads and tails in X times of flipping a coin (aka trial or experiments of flipping a coin); the number of donations in a month to the charity that asks for donations.

Parameter p is the probability of one of the two possible outcomes. Parameter x is the number of experiments (trials) to run, it is also the number of outputs that you would like to get.

```
# generate x binomial random variates of x bernoulli trials with probability p
def generate_binomial_rv(p, x):
    outputs = []

    for i in range(x):
        # the number of successes and failures out of x Bernoulli trials
        bernoulli_output = generate_bernoulli_rv(p, x)[0]
        # the number of successes (=1)
        outputs.append(bernoulli_output.count(1))

    return outputs
```

The example has a Bernoulli trial of probability of 0.7 by sample sizes of 20 vs 1000. The output of 1000 samples is very close to 700, that is the product of 1000 and 0.7.

```
# draw_binomial_rv() runs the generation function and creates chart(s). Details in the code.
```

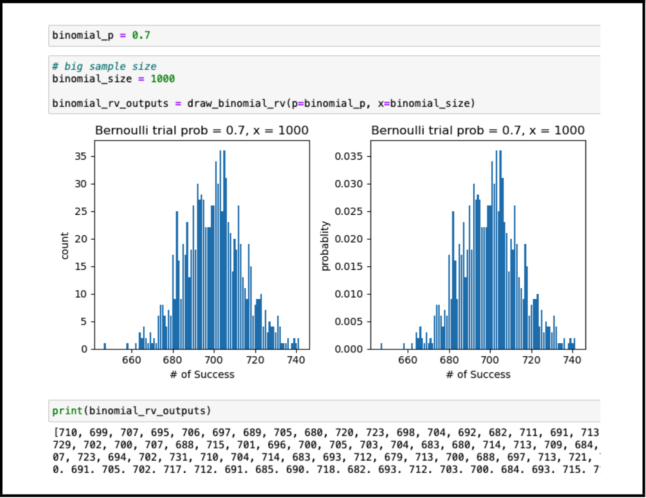


```
binomial_p = 0.7

# small sample size
binomial_size = 20

binomial_rv_outputs = draw_binomial_rv(p=binomial_p, x=binomial_size)
```

Bernoulli trial prob = 0.7, x = 20
count
# of Success
Bernoulli tria prob = 0.7, x = 20
probability
# of Success

```
print(binomial_rv_outputs)
```

```
[12, 16, 11, 14, 13, 13, 13, 13, 13, 16, 14, 17, 17, 11, 14, 14, 17, 14, 13, 13]
```



```
binomial_p = 0.7

# big sample size
binomial_size = 1000

binomial_rv_outputs = draw_binomial_rv(p=binomial_p, x=binomial_size)
```

Bernoulli trial prob = 0.7, x = 1000
count
# of Success
Bernoulli trial prob = 0.7, x = 1000
probability
# of Success

```
print(binomial_rv_outputs)
```

```
[710, 699, 707, 695, 706, 697, 689, 705, 680, 720, 723, 698, 704, 692, 682, 711, 691, 713
729, 702, 700, 707, 688, 715, 701, 696, 700, 705, 703, 704, 683, 680, 714, 713, 709, 684,
07, 723, 694, 702, 731, 710, 704, 714, 683, 693, 712, 679, 713, 700, 688, 697, 713, 721,
0. 691. 705. 702. 717. 712. 691. 685. 690. 718. 682. 693. 712. 703. 700. 684. 693. 715.
```

Discrete Distribution 3 # Geometric distribution

The statistical meaning is that the probability of having X Bernoulli trials with probability p until a Success occurs, i.e. the first success. Real-life examples include job search, such as how many applications until receiving an offer; defect products, such as how many items produced until the first defective unit.

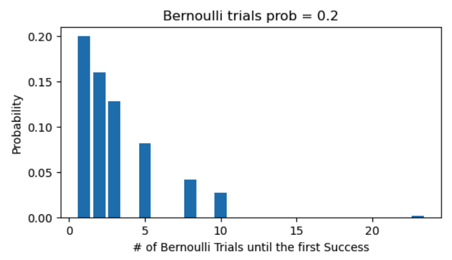
Parameter p is the probability of one of the two possible outcomes. Parameter x is the number of samples, that is the number of outputs that you want to get. For each output, it will keep generating Bernoulli trials until a Success (= 1) occurs.

```
# generate x geometric random variate of bernoulli trials with probability p
def generate_geometric_rv(p, x):
    outputs = []
    trial_counts = []

    for i in range(x):
        bernoulli_outcome = 0
        trial_count = 0

        while bernoulli_outcome != 1: # keep going until a Success occurs
            bernoulli_outcome = generate_bernoulli_rv(p, 1)[0][0]
            trial_count += 1
        outputs.append((1-p)**(trial_count-1)*p)
        trial_counts.append(trial_count)

    return outputs, trial_counts
```

The example has a Bernoulli trial of probability of 0.2. The left one has a small sample size of 10, and the right one has a large sample size of 1000 that shows a smooth curve.

```
# draw_geometric_rv() runs the generation function and creates chart(s). Details in the code.
```

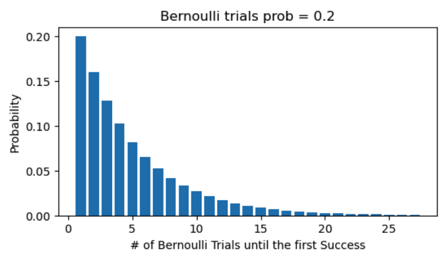
```
geometric_p = 0.2

# small sample size
geometric_size = 10

geometric_rv_outputs = draw_geometric_rv(p=geometric_p, x=geometric_size)
```



```
print(geometric_rv_outputs)
```

```
[10, 10, 5, 23, 1, 3, 2, 3, 2, 8]
```

```
geometric_p = 0.2

# big sample size
geometric_size = 1000

geometric_rv_outputs = draw_geometric_rv(p=geometric_p, x=geometric_size)
```



```
print(geometric_rv_outputs)
```

```
[3, 20, 1, 2, 1, 12, 3, 2, 15, 7, 3, 2, 5, 1, 1, 1, 4, 12, 14, 6, 4, 2, 1,
4, 3, 1, 3, 2, 4, 5, 11, 2, 5, 2, 1, 3, 1, 4, 3, 1, 5, 1, 4, 2, 2, 6, 7, 2
9, 17, 1, 2, 2, 4, 3, 4, 5, 5, 2, 5, 10, 11, 3, 1, 6, 7, 6, 1, 13, 2, 11,
11, 10, 5, 12, 9, 3, 1, 2, 2, 13, 1, 3, 5, 11, 2, 3, 6, 8, 3, 2, 1, 2, 3,
```

Discrete Distribution 4 # Poisson distribution

The statistical meaning is that the probability of an event happening X times within a time period. Real-life examples include in the call center, the probability that the call center receives more than 5 phone calls during the noon, given the average 3 calls per hour during that time period; given the average 2 "uh" per broadcast, the probability that the news reporter says "uh" more than three times during a broadcast.

Parameter `l (lambda)` is the number of events occurring within a time period. And parameter x is the number of outputs that you want to get.

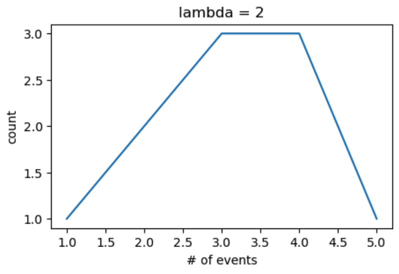
```
# generate x poisson random variate within a time period
def generate_poisson_rv(l, x):
    outputs = []

    for i in range(x):
        prn = generate_prn()
        time_count = 0
        cmf = 0
        while prn > cmf:
            pmf = math.e**-l*l**time_count/math.factorial(time_count)
            cmf = cmf + pmf
            time_count += 1
        outputs.append(time_count)

    return outputs
```

The example has a λ of 2. I presented two outputs by different sample sizes - the left one is a small size with 10 samples versus the right one is a large size with 1000 samples.

```
# draw_poisson_rv() runs the generation function and creates chart(s). Details in the code.
```

```
poisson_lambda = 2
```

```
# small sample size
lambda_size = 10

poisson_rv_outputs = draw_poisson_rv(l=poisson_lambda, x=lambda_size)
```

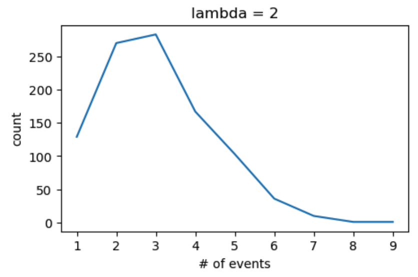


```
print(poisson_rv_outputs)
```

```
[1, 2, 3, 4, 3, 4, 4, 2, 5, 3]
```

```
poisson_lambda = 2
```

```
# big sample size
poisson_size = 1000

poisson_rv_outputs = draw_poisson_rv(l=poisson_lambda, x=poisson_size)
```



```
print(poisson_rv_outputs)
```

```
[3, 3, 1, 4, 1, 4, 2, 3, 2, 2, 3, 3, 4, 4, 1, 6, 2, 1, 5, 2, 4, 3, 3, 3
2, 2, 2, 1, 5, 6, 4, 7, 4, 4, 1, 4, 5, 1, 2, 1, 3, 4, 3, 4, 1, 4, 4, 5,
3, 1, 2, 2, 1, 1, 3, 3, 4, 3, 2, 4, 2, 6, 5, 2, 1, 1, 2, 2, 3, 2, 2, 1,
3, 3, 2, 1, 3, 4, 1, 7, 3, 2, 1, 5, 4, 2, 6, 2, 4, 2, 5, 5, 2, 3, 2, 1,
3, 7, 6, 2, 6, 6, 4, 2, 5, 5, 2, 3, 3, 2, 2, 4, 3, 2, 3, 4, 2, 1, 4, 3
```

<u>Continuous Distribution 2 # Exponential distribution</u>

The statistical meaning is that the probability of X time units elapsed between two consecutive events, given the event has a Poisson rate of λ, meaning on average events occur λ times within a time period. Real-life examples include the amount of time from now until an earthquake occurs.

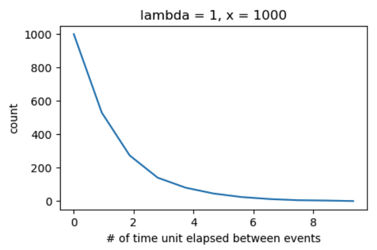
Parameter `l` (λ) is the number of events occurring within a time period. And parameter x is the number of outputs that you want to get.

```
# generate x exponential random variate between two consecutive events
def generate_exponential_rv(l, x):
    outputs = []

    for i in range(x):
        prn = generate_prn()

        rv = -1/l*math.log(prn, 2)

        outputs.append(rv)

    return outputs
```

The example has different λ of 1 and 5 with the same sample sizes of 1000. I also prepared a chart showing four exponential distributions with varying λ values.

```
# More details in the complete code.
```

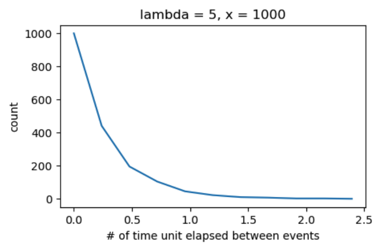
```
exp_lambda = 1

exp_size = 1000

exponential_rv_outputs = draw_exponential_rv(l=exp_lambda,
                                             x=exp_size)
```



```
for i in range(10):
    print(exponential_rv_outputs[i])
```

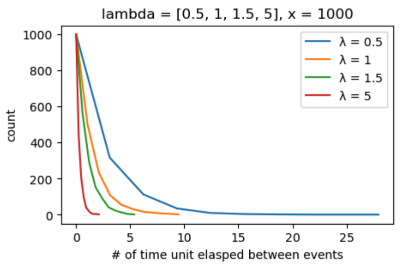
```
7.2781754898814271
1.498453034654843
1.1646536078310834
2.1837587413028277
0.7558722081718126
3.126039069741296
0.10867343813924862
```

```
exp_lambda = 5

exp_size = 1000

exponential_rv_outputs = draw_exponential_rv(l=exp_lambda,
                                             x=exp_size)
```



```
for i in range(10):
    print(exponential_rv_outputs[i])
```

```
0.1024848626878246
0.3709762101637842
0.19113244947357502
0.7967800243037764
0.00991517221901405
0.8912033889384111
0.016077531535608116
```



Continuous Distribution 3 # Weibull distribution

The statistical meaning is a generalization of Exponential Distribution, which is the probability of X time unit elapsed between events, given the event has a Poisson rate of λ. Note that k in the Weibull distribution is important: < 1 meaning failure decreases with time; > 1 meaning failure increases with time; = 1 meaning failure constant with time, that is Exponential distribution. Real-life examples include for defect products, the amount of times until the first defective unit or the amount of times between failure events; for the web use, the amount of time a user spends on a web page (a.k.a between websites).

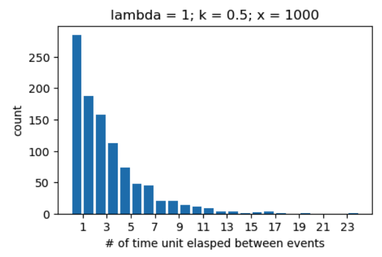
Parameter `l` (λ) is the number of events occurring within a time period. Parameter k is the shape parameter. Parameter x is the number of outputs that you want to get.

```
# generate x weibull random variate between two consecutive events
def generate_weibull_rv(l, x, k):
    outputs = []

    for i in range(x):
        prn = generate_prn()

        logged_num = (-math.log(prn, 2))**1/k

        X = 1/l * logged_num

        outputs.append(X)

    return outputs
```

The example has a λ of 1 and a sample size of 1000 with different k. As k becomes bigger, the amount of time units elapsed until the first event increases from 280, 500, to 700. Moreover, when k is larger than 1, it is less likely to have as many time units elapsed until the first event as when k is less than 1.

```
# More details in the complete code.
```

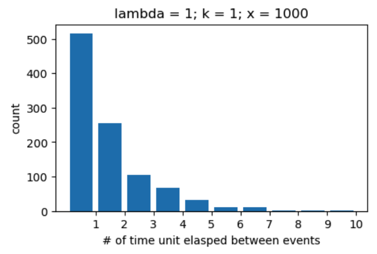
```
weibull_lambda = 1
weibull_k = 0.5
weibull_size = 1000

weibull_rv_outputs = draw_weibull_rv(l=weibull_lambda,
                                     x=weibull_size,
                                     k=weibull_k)
```



```
for i in range(15):
    print(weibull_rv_outputs[i])
```

```
0.2242879085786229
0.22198259664814857
0.9176185497861726
1.9200017666054354
4.371354837210938
3.0875978792101257
1.8084406324859885
```

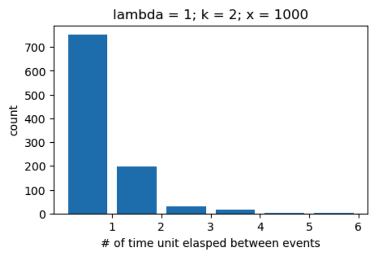
```
weibull_lambda = 1
weibull_k = 1
weibull_size = 1000

weibull_rv_outputs = draw_weibull_rv(l=weibull_lambda,
                                     x=weibull_size,
                                     k=weibull_k)
```



```
for i in range(15):
    print(weibull_rv_outputs[i])
```

```
0.7164679041775696
0.13111501326273567
3.3225412132232375
1.1139164957680139
6.629577738558458
0.005792216381656251
0.2559085400045857
```

```
weibull_lambda = 1
weibull_k = 2
weibull_size = 1000

weibull_rv_outputs = draw_weibull_rv(l=weibull_lambda,
                                     x=weibull_size,
                                     k=weibull_k)
```



```
for i in range(15):
    print(weibull_rv_outputs[i])
```

```
1.2037464450739281
0.41249978083977895
0.6983936478735885
1.0013621676897013
0.14866773518727014
0.06287253634407856
0.7991609975069589
```

Continuous Distribution 4 # Normal distribution

The statistical meaning is a continuous probability distribution with mean and variance, and the data is symmetrically distributed with no skew. Real-life examples include heights, SAT scores, and restaurant reviews.

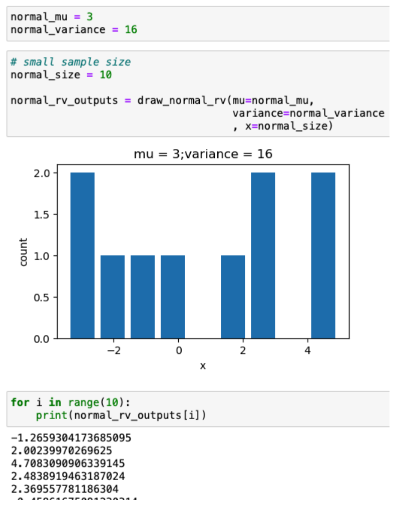
Parameter `mu` is the mean or expectation of the distribution. Parameter `variance` is the variance of the distribution. And parameter x is the number of outputs that you want to get.

```
# generate x normal random variate with mean and variance
```

```
def generate_normal_nv(mu, variance, x):
    outputs = []

    for i in range(x):
        U = generate_prn()
        Z = (U**0.135-(1-U)**0.135)/0.1975
        X = mu + math.sqrt(variance)*Z
        outputs.append(X)

    return outputs
```

The example has a mean of 3 and a variance of 16. I presented three outputs by different sample sizes - the left one is a small size of 10 samples, the middle one is a bigger size of 1000 samples, and the right one is a large size with 6000 samples. The bar chart shows more random variates centered around the mean value while the sample size becomes larger.

```
# More details in the complete code.
```

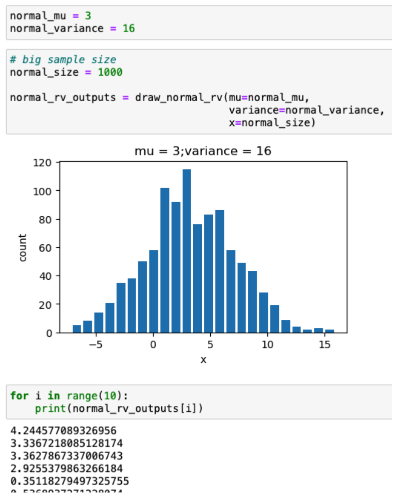


```
normal_mu = 3
normal_variance = 16
```

```
# small sample size
normal_size = 10

normal_rv_outputs = draw_normal_rv(mu=normal_mu,
                                   variance=normal_variance
                                   , x=normal_size)
```

```
for i in range(10):
    print(normal_rv_outputs[i])
```

```
-1.2659304173685095
2.00239970269625
4.7083090906339145
2.4838919463187024
2.369557781186304
```

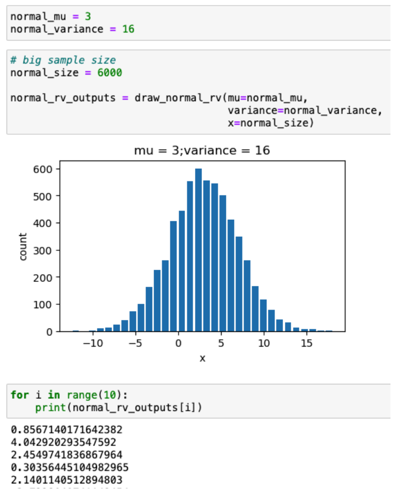


```
normal_mu = 3
normal_variance = 16
```

```
# big sample size
normal_size = 1000

normal_rv_outputs = draw_normal_rv(mu=normal_mu,
                                   variance=normal_variance,
                                   x=normal_size)
```

```
for i in range(10):
    print(normal_rv_outputs[i])
```

```
4.244577089326956
3.3367218085128174
3.362786733700674З
2.9255379863266184
0.3511827949732575５
```



```
normal_mu = 3
normal_variance = 16
```

```
# big sample size
normal_size = 6000

normal_rv_outputs = draw_normal_rv(mu=normal_mu,
                                   variance=normal_variance,
                                   x=normal_size)
```

```
for i in range(10):
    print(normal_rv_outputs[i])
```

```
0.8567140171642382
4.042920293547592
2.4549741836867964
0.30356445104982965
2.140114051289480З
```

Conclusions

In this project, I have found that these distributions require different criteria and logics to generate random variates. They focus on and aim at solving their own problems.

The generated random variates have to go through examination and validation to prove its usefulness and correctness. One of the examinations I used was making graphs out of the random variates. For a few distributions, I chose line charts; but for most distributions, I went with histogram or bar charts. It is interesting to see that when there are enough observations, such as 1000 outputs, the histogram or bar charts start to converge to the true distribution.

To expand this project, possible future work includes conducting statistical tests such as goodness-of-fit test and independence test, and publishing the entire Python program as a shareable library.