Notebook

February 17, 2019

Local date & time is : 02/17/2019 09:38:19 PST

```
In [105]: 2**(np.arange(4))

Out[105]: array([1, 2, 4, 8])

In [106]: def ev_W_run(p, n):
              return sum(p**(np.arange(n))) / p**(n)

In [107]: # should evaluate to 1 / p = 6
          ev_W_run(1/6, 1)

Out[107]: 6.0

In [108]: # should evaluate to (0.25 + 0.5 + 1) / 0.125 = 14
          ev_W_run(0.5, 3)

Out[108]: 14.0

In [109]: # Expected number of:

          # (1) fair coin tosses till 10 consecutive heads
          ans_1 = ev_W_run(0.5, 10)

          # (2) rolls of a die till 6 consecutive sixes
          ans_2 = ev_W_run(1/6, 6)

          # (3) runs of a random number generator till 000
          ans_3 = ev_W_run(1/10, 3)

          # (4) days till ZZZZZ by robot typist
          ans_4 = ev_W_run(1/26, 5) / (10*60*60*24)

          ans_1, ans_2, ans_3, ans_4

Out[109]: (2046.0, 55986.000000000015, 1109.9999999999998, 14.30165509259259)
```

1 newpage

In [111]:
```
stats.bernoulli.rvs(0.2, size=1)
```

Out[111]:
```
array([0])
```

In [112]:
```
def run(p, n):
    """Returns one simulated value of W_H,n
    in i.i.d. Bernoulli (p) trials"""

    tosses = 0                     # Number of tosses
    in_a_row = 0                   # Number of consecutive heads observed

    while in_a_row < n:            # While fewer than n consecutive heads
        tosses = tosses + 1               # update tosses
        if stats.bernoulli.rvs(p, size=1).item(0) == 1:
            in_a_row = in_a_row + 1                  # update in_a_row
        else:
            in_a_row = 0                  # reset in_a_row

    return tosses
```

In [113]:
```
# should return around 1 / p  = 1/ 0.9 ~ 1
run(0.9, 1)
```

Out[113]:
```
1
```

In [114]:
```
# should around return 1 / 0.1 = 10
run(0.1, 1)
```

Out[114]:
```
6
```

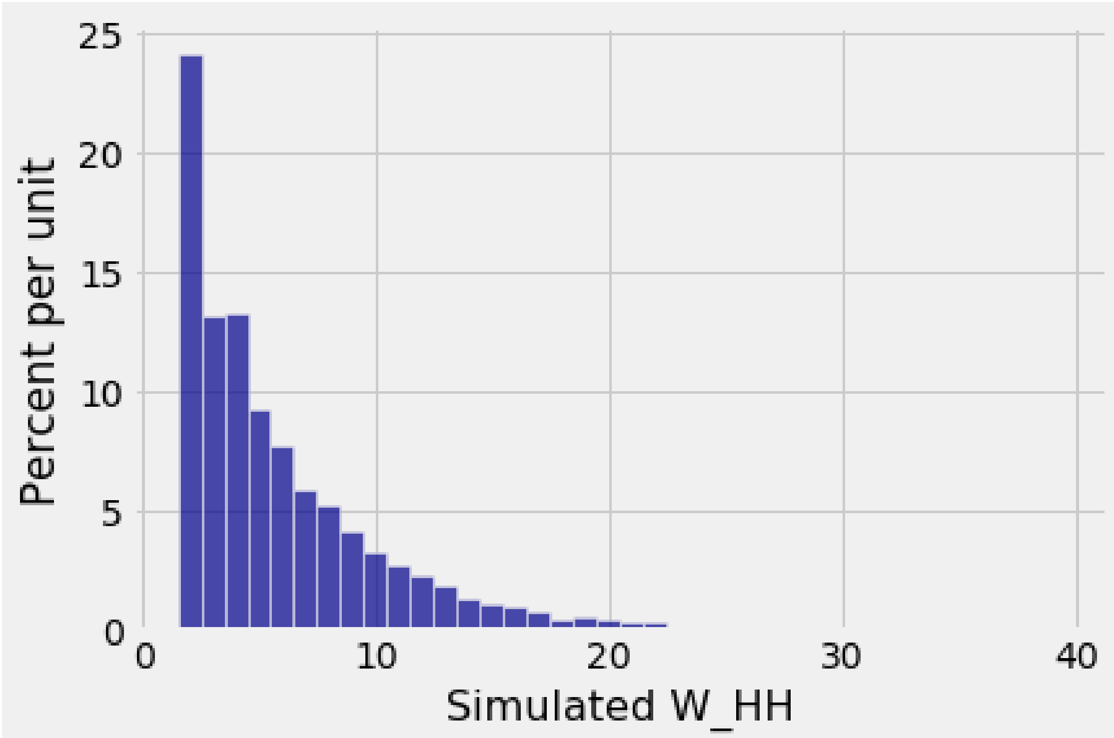
In [115]:
```
def simulate_run(p, n, repetitions):
    """Returns an array of length equal to repetitions,
    whose entries are independent simulated values of W_H,n
    in i.i.d. Bernoulli (p) trials"""
    results = make_array()
    for i in np.arange(repetitions):
        results = np.append(results, run(p, n))
    return results
```

In [117]:
```
# should be around 14
np.mean(simulate_run(0.5, 3, 10000))
```

Out[117]:
```
14.1493
```

In [118]:
```
sim_W_HH = simulate_run(0.5, 2, 10000)
```

In [119]:
```
Table().with_column('Simulated W_HH', sim_W_HH).hist(bins = np.arange(1.5, 40.5))
```




```
In [120]: ev_W_run(0.5, 2), np.mean(sim_W_HH)
```

```
Out[120]: (6.0, 5.9731)
```

(1) 1/2 * 1/2 = 0.25

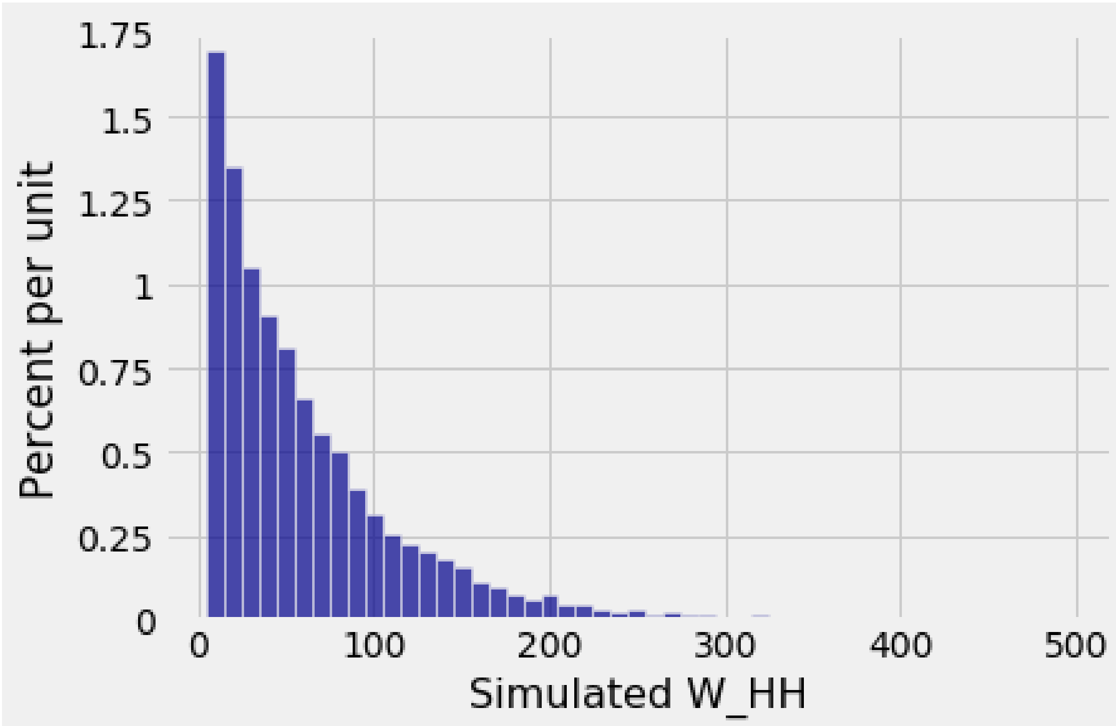
(2) want THH => 1/2 * 1/2 * 1/2 = 1/8 = 0.125

(3) want TTHH, HTHH => 2(1/2 1/2 * 1/2 * 1/2) = 0.125

Comparing to the value of the probability histrogram, the values match up.

```
In [121]: sim_W_H5 = simulate_run(0.5, 5, 10000)
```

```
In [122]: Table().with_column('Simulated W_HH', sim_W_H5).hist(bins = np.arange(5, 501, 10))
          plt.ylim(0, 0.0175); # ignore; forces the vertical scale to go up to 1.75 %/unit
```




```
In [123]: ev_W_run(0.5, 5), np.mean(sim_W_H5)
```

```
Out[123]: (62.0, 62.1123)
```

Your answer here.

(1) 62

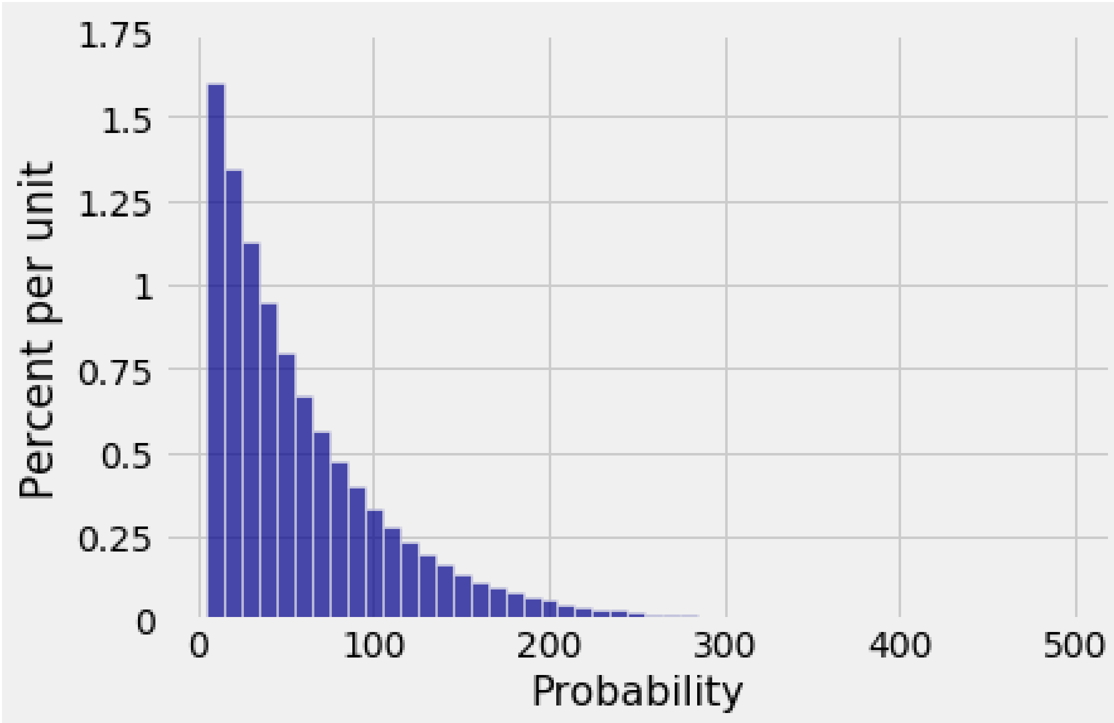
(2) 5 ~ 300

(3) Geometric

(4) The distribution of$W_{H,5}$is pretty close to the distribution of$X+4$where$X$has the*Geometric*distribution with parameter (or parameters)$1/58$.

```
In [128]: k = np.arange(1, 1001)
          approx_probs = stats.geom.pmf(k, 1/58)
          approx_dist = Table().values(k+4).probabilities(approx_probs)

          # Ignore; Forces hist to use the same scale as the empirical histogram
          approx_dist.hist(bin_column='Value', bins=np.arange(5, 501, 10))
          plt.ylim(0, 0.0175);
```




```
In [129]: (1 - 1/58)**100
```

```
Out[129]: 0.17566539272291076
```

```
In [130]: sum(sim_W_H5 > 100) / len(sim_W_H5)
```

```
Out[130]: 0.1885
```

2 newpage

Value of$W_{HT}$is 8. I waited till the first H and then I waited till the first$T$

$W_{HT} = W_H + V$where$V$is independent of$W_H$and has the*Geometric*distribution.

1/p + 1/q

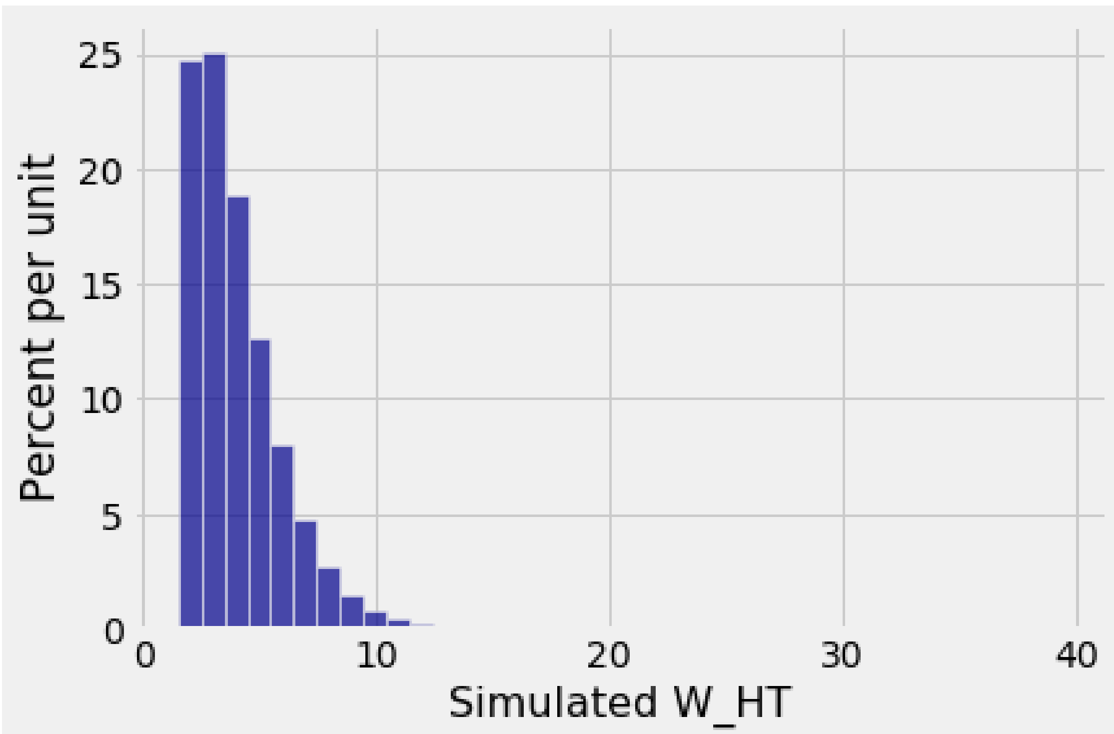
$E(W_{HH})$= (1+p) / p^2 = 6

$E(W_{HT})$= 1/p + 1/q = 4

In [137]:
```
# Fair coin: Empirical histogram of W_HT based on 10,000 simulated values

sim_W_HT = simulate_run(0.5, 1, 10000) + simulate_run(0.5, 1, 10000)

Table().with_column('Simulated W_HT', sim_W_HT).hist(bins = np.arange(1.5, 40.5))
```



In [139]:
```
np.mean(sim_W_HH), np.mean(sim_W_HT)
```

Out[139]: (5.9731, 3.9969)

To compareW_{HH}andW_{HT}in tosses of a fair coin, - in both cases it takes the same expected time to first get toW_H; - in both cases, the pattern appears on the next toss with the same chance 0.5; - in both cases, the pattern fails to appear on the next toss with the same chance 0.5, and then: - to get toW_{HH}you have to wait tillHHappears in the subsequent tosses, whereas - to get toW_{HT}you have to wait till$T$appears in the subsequent tosses.

3 newpage

In [1]:
```python
# A
ev_W_A = 26

# AZ
ev_W_AZ = 26**2

# AA
ev_W_AA = 26**2 + 26

# BOO
ev_W_BOO = 26**3

# BOB
ev_W_BOB = 26**3 + 26

# GAGA
ev_W_GAGA = 26**4 + 26*2

# RADIOGAGA
ev_W_RADIOGAGA = 26**9

# ABRACADABRA
ev_W_ABRACADABRA = 26**11 + 26 + 26**4

ev_W_A, ev_W_AZ, ev_W_AA, ev_W_BOO, ev_W_BOB, ev_W_GAGA, ev_W_RADIOGAGA, ev_W_ABRACADABRA
```

Out[1]:
```
(26, 676, 702, 17576, 17602, 457028, 5429503678976, 3670344487444778)
```