Notebook

March 8, 2019

Local date & time is : 03/08/2019 16:32:04 PST

1 newpage

1.1 Part 1: Ranks

We will develop the method by revisiting Deflategate, a storm in the world of American football and familiar to us from Data 8.

Here are some extracts from the Data 8 textbook:

"On January 18, 2015, the Indianapolis Colts and the New England Patriots played the American Football Conference (AFC) championship game to determine which of those teams would play in the Super Bowl. After the game, there were allegations that the Patriots' footballs had not been inflated as much as the regulations required; they were softer. This could be an advantage, as softer balls might be easier to catch ...

At half-time, all the game balls were collected for inspection. Two officials, Clete Blakeman and Dyrol Prioleau, measured the pressure in each of the balls.

Here are the data. Each row corresponds to one football. Pressure is measured in psi [pounds per square inch]. The Patriots ball that had been intercepted by the Colts was not inspected at half-time. Nor were most of the Colts' balls – the officials simply ran out of time and had to relinquish the balls for the start of second half play."

Each team had 12 footballs. Eleven of the Patriots' footballs were measured, and four of the Colts'.

```
In [46]: start = np.append(np.ones(11)*12.5, np.ones(4)*13)

         blakeman_drops = start - football[1]
         prioleau_drops = start - football[2]

In [47]: drops = football.drop(1, 2).with_column(
             'Blakeman', blakeman_drops,
             'Prioleau', prioleau_drops
         )

         drops.show()
```

```
<IPython.core.display.HTML object>
```

```
In [50]: blakeman_ranks = stats.rankdata(blakeman_drops, method = 'ordinal')
         prioleau_ranks = stats.rankdata(prioleau_drops, method = 'ordinal')

In [52]: drops.show()
```

```
<IPython.core.display.HTML object>
```

It is easier to compare the ranks as it is only one digit to compare. The rank 13, 14, 11, 15, 12, are consistent with each other, while others are not. So the higher ranks are more consistent. Also all the higher ranks happen to be patriots.

2 newpage

2.1 Part 2: Wilcoxon's Rank Sum Statistic

it is 2 + 1 + 4 + 3 = 10

```
In [53]: total_samples = special.comb(15, 4)
         total_samples
```

```
Out[53]: 1365.0
```

the smallest possible sum is 10 as 1+2+3+4. There is 4! = 4 * 3 * 2 * 1 = 24 subsets of this sum.

Not exactly, the chance of having the rank sum is 24/1365, which is rather a low probabilty. Hence we can not conclude that colts' rank are like a random sample of four ranks.

```
In [55]: both_sums = prioleau.group(0, sum)
         prioleau_colts_sum = both_sums[1].item(0)

         both_sums
```

```
Out[55]: Team     | Prioleau Ranks sum
         Colts    | 18
         Patriots | 102
```

```
In [56]: # The total of the ranks is the sum of 1 to 15

         (1+15)*(15/2)
```

```
Out[56]: 120.0
```

```
In [58]: rank_sums = all_samples.apply(sum, 0)

         all_samples = all_samples.with_column('Rank Sum', rank_sums)

         all_samples
```

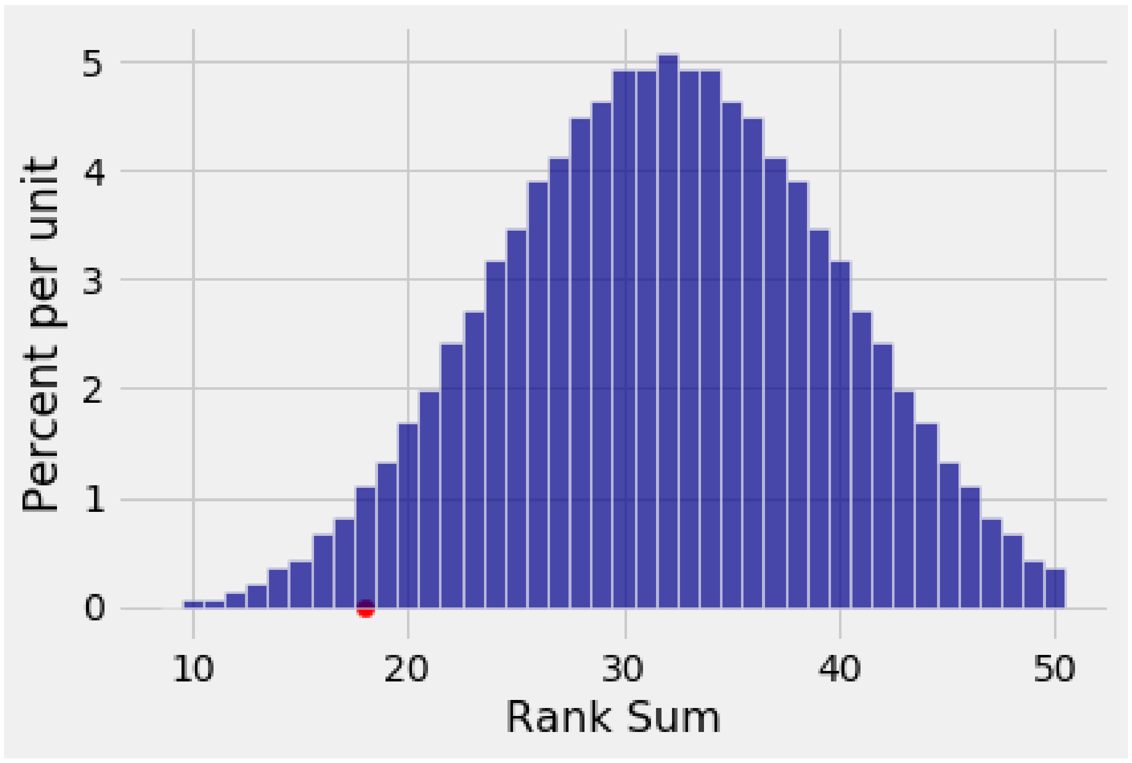
```
Out[58]: Ranks          | Rank Sum
         [1 2 3 4]      | 10
         [1 2 3 5]      | 11
         ... Omitting 5 lines ...
         [ 1  2  3 12] | 18
         [ 1  2  3 13] | 19
         ... (1355 rows omitted)
```

```
In [59]: smallest = 10
         largest = 12+13+14+15

         smallest, largest
```

```
Out[59]: (10, 54)
```

```
In [60]: all_samples.hist(1, bins = np.arange(9, 52)-0.5)
         plt.scatter(prioleau_colts_sum, 0, color='red', s=40);
```



```
In [61]: sum(all_samples[1] <= 18) / len(all_samples[1])
```

```
Out[61]: 0.03882783882783883
```

If we use the P-value test of 0.05, our observed p-value is less than 0.05. Hence we should reject the null hypothesis. The hypothesis that the balls were inflated intentionally is better supported.

3 newpage

3.1 Part 3: Normal Curves

The probability distribution of the rank sum statistic looks very much like the normal curve, but not exactly. For example, look at the peak of the curve. You will see two flat bits on either side.

Still, the distribution doesn't look too far from normal, so it is worth reminding ourselves about the normal curve. This part of the lab takes you quickly through some code that you can use to display normal curves and areas under them, and also to find numerical values of the areas.

```
In [67]: stats.norm.cdf(71, 68, 3 ) - stats.norm.cdf(65, 68, 3)
```

```
Out[67]: 0.6826894921370859
```

4 newpage

4.1 Part 4: Normal Approximation

To use a normal curve to approximate a distribution, you must first identify the two parameters of the curve. For reasons that are not surprising and will become precise later in the course, the right parameters are the expectation and SD of the distribution being approximated.

Let's find the normal curve that approximates the distribution of the rank sum statistic in Part 2. Why approximate a distribution we already know exactly? The answer is that we will need the method of approximation when the sample sizes are too large for us to be able to enumerate all possible samples. Finding the approximation in a case where we know the exact answer helps us see that the approximation is good.

```
In [68]: def ev_ranksum(n, N):
             return (1+N)/2 * n

         null_expectation = ev_ranksum(4, 15)
         null_expectation
```

```
Out[68]: 32.0
```

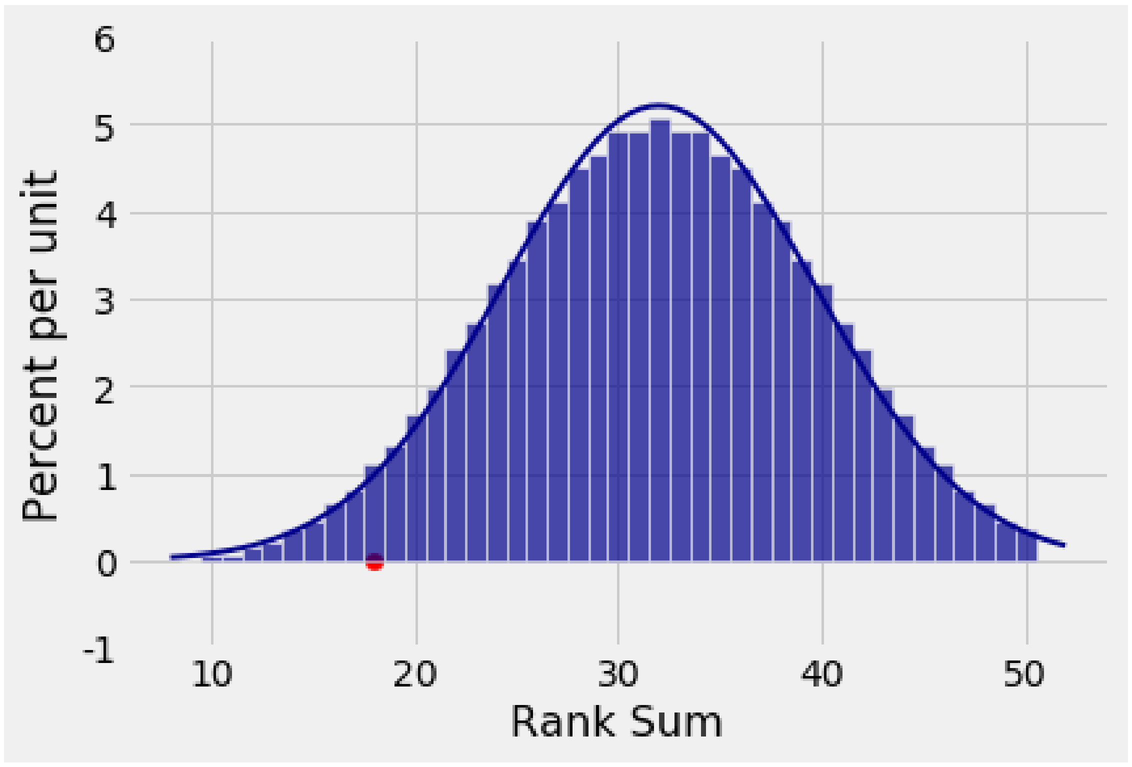
```
In [69]: def sd_ranksum(n, N):
             return (n*((N**2-1)/12)*(N-n)/(N-1))**(0.5)

         null_sd = sd_ranksum(4, 15)
         null_sd
```

```
Out[69]: 7.659416862050705
```

```
In [71]: all_samples.hist(1, bins = np.arange(9, 52)-0.5)
         plt.scatter(prioleau_colts_sum, 0, color='red', s=40)

         Plot_norm((8,52), null_expectation, null_sd)
         plt.ylim(-0.01, 0.06);
```

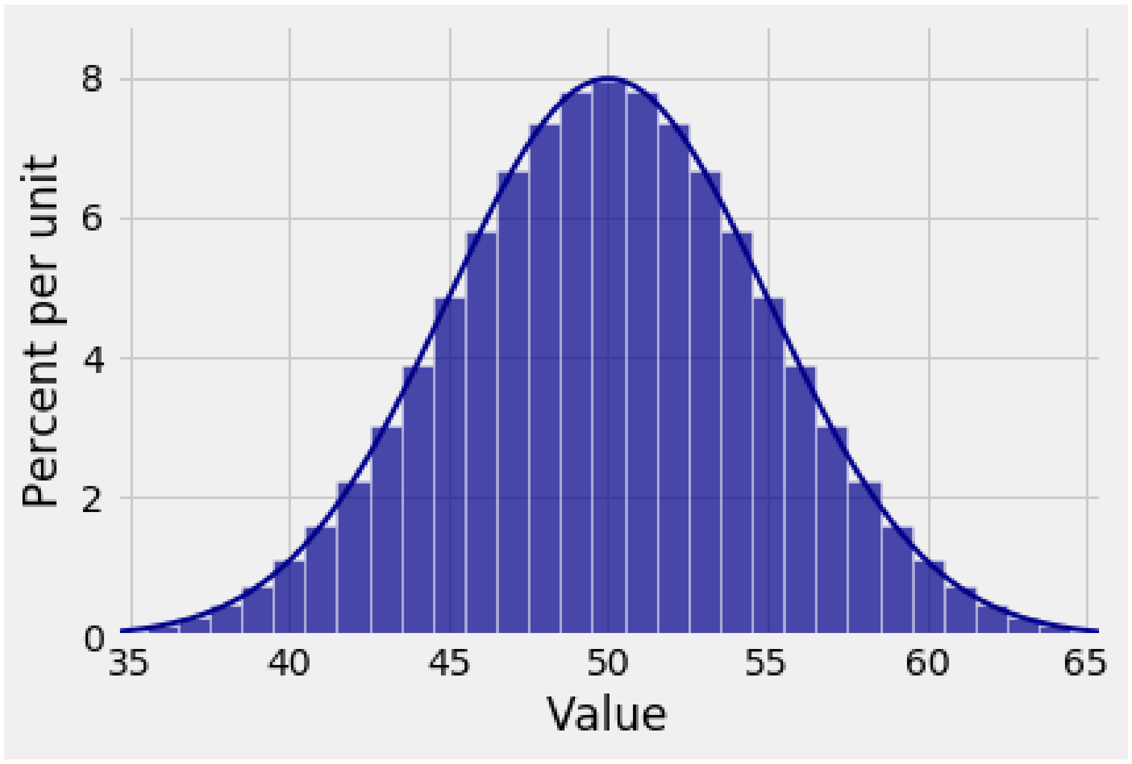

Percent per unit
Rank Sum
6
5
4
3
2
1
0
-1
10
20
30
40
50

5 newpage

5.1 Part 5: Continuity Correction

```
In [73]: b100 = stats.binom.pmf(np.arange(101), 100, 0.5)
         coins100 = Table().values(range(101)).probabilities(b100)

         Plot(coins100, edges=True)
         Plot_norm((34.5, 65.5), 50, 5)
         plt.xlim(34.5, 65.5);
```

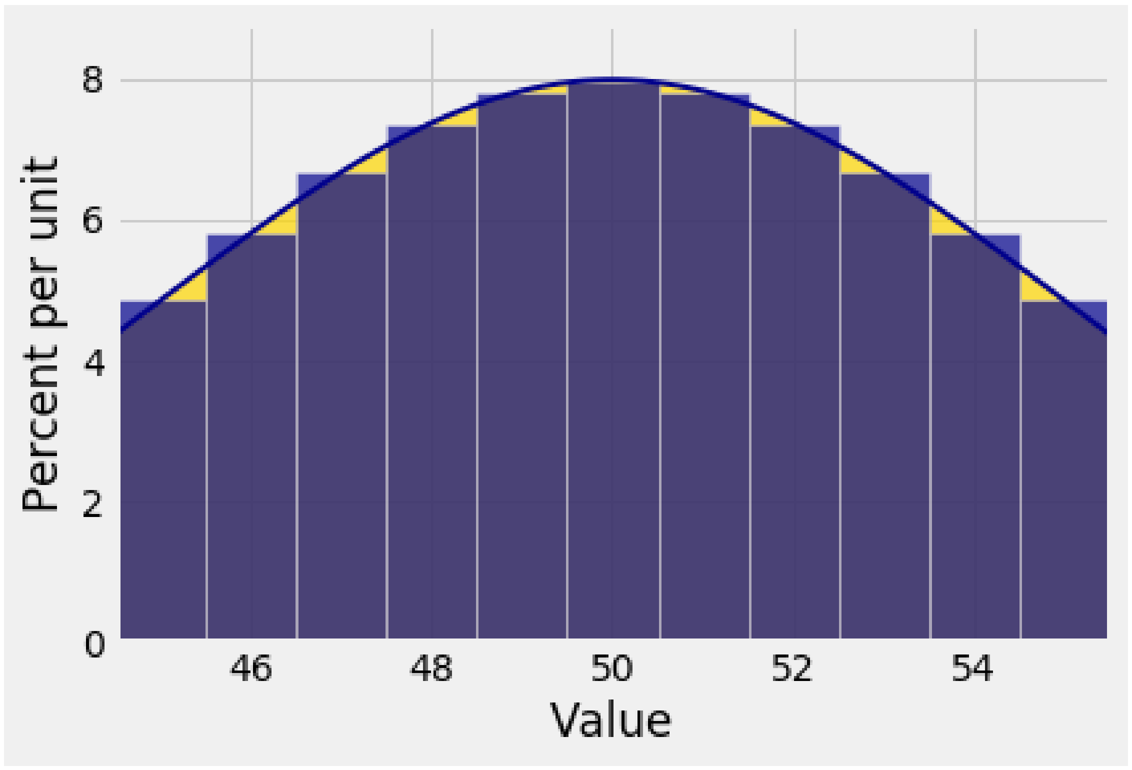


```
In [75]: sum(stats.binom.pmf(np.arange(45, 56), 100, 0.5))
```

```
Out[75]: 0.728746975926143
```

No, it leaves out the area the both end that is below the graph. The area before 45 and after 55 is left out.

```
In [77]: Plot(coins100, edges=True)
         Plot_norm((43.5, 56.5), 50, 5, left_end = 44.5, right_end = 55.5)
         plt.xlim(44.5, 55.5);
```



```
In [106]: stats.norm.cdf(55.5, 50, 5) - stats.norm.cdf(44.5, 50, 5)

Out[106]: 0.7286678781072347

In [84]: (stats.norm.cdf(prioleau_colts_sum+0.5,
                         null_expectation,
                         null_sd),
          sum(all_samples[1] <= 18) / len(all_samples[1]))

Out[84]: (0.0389893672505451, 0.03882783882783883)
```

6 newpage

6.1 Part 6: Large Sample Analysis

The beauty of the rank-based method is that it extends easily to large samples. Here is a sample of the ages of 231 men and women, part of a larger data set that included other attributes as well. The code for gender was 0 for female and 1 for male; no other categories were recorded. The ages include fractional parts of a year, which is why the decimals aren't pretty.

```
In [87]: ages.group(0)
```

```
Out[87]: Gender | count
         0      | 177
         1      | 54
```

```
In [97]: age_ranks = stats.rankdata(ages[1], method = 'ordinal')
         ranked = ages.with_column('Age Rank', age_ranks).select(0,2)
         ranked
```

```
Out[97]: Gender | Age Rank
         0      | 130
         0      | 49
         ... Omitting 5 lines ...
         0      | 158
         0      | 154
         ... (221 rows omitted)
```

```
In [102]: ranked.group(0, sum)
```

```
Out[102]: Gender | Age Rank sum
          0      | 20815
          1      | 5981
```

```
In [103]: exp_W = ev_ranksum(54, 231)
          sd_W = sd_ranksum(54, 231)

          exp_W, sd_W
```

```
Out[103]: (6264.0, 429.8697477143513)
```

Our observed sum of the male ranks is within one standard deviation from the mean, hence the probabilty of getting the observed sum is greater than 0.05 which is our typical p-value. So I would decide that the ranks of men look like a simple random sample of all 231 ranks.

```
In [108]: stats.norm.cdf(5981, exp_W, sd_W)
```

```
Out[108]: 0.2551601762650195
```