Statistical Thinking in Python Part 2

4). Parameter estimation by optimization

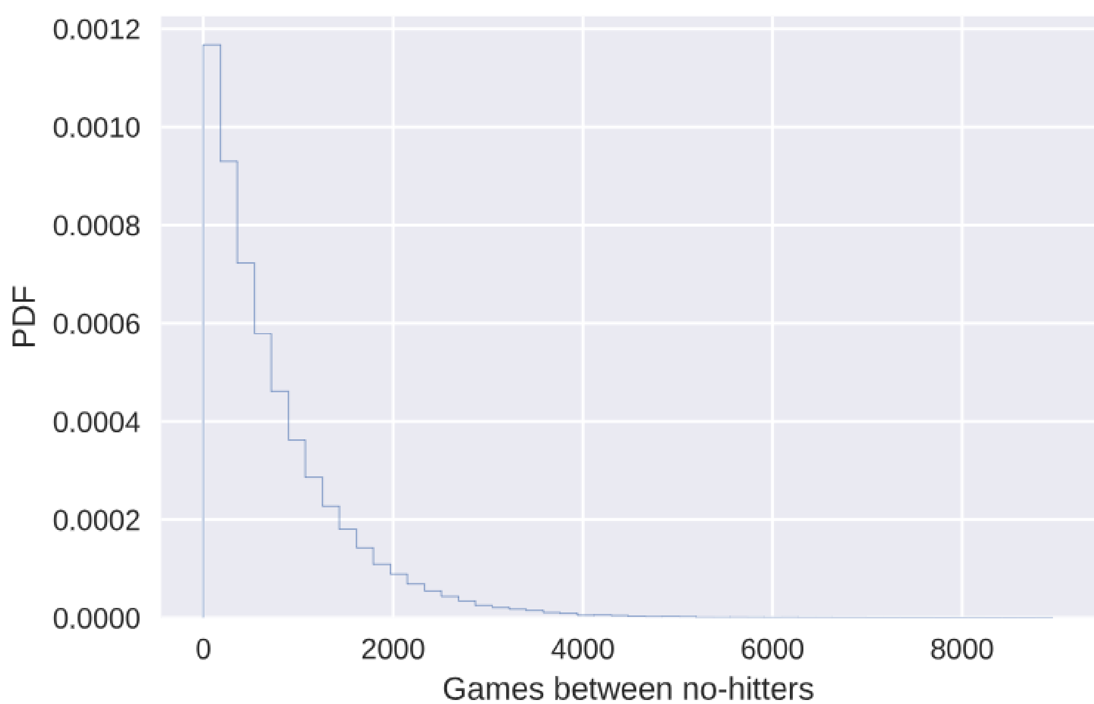
a). How often do we get no hitters?

```
import
numpy
as np
        import matplotlib.pyplot as plt


        nohitter_times = np.array([843, 1613, 1101,  215,  684,  814,  278,  324,  161,
        219,  545,
                715,  966,  624,   29,  450,  107,   20,   91, 1325,  124, 1468,
                104, 1309,  429,   62, 1878, 1104,  123,  251,   93,  188,  983,
                166,   96,  702,   23,  524,   26,  299,   59,   39,   12,    2,
                308, 1114,  813,  887,  645, 2088,   42, 2090,   11,  886, 1665,
               1084, 2900, 2432,  750, 4021, 1070, 1765, 1322,   26,  548, 1525,
                 77, 2181, 2752,  127, 2147,  211,   41, 1575,  151,  479,  697,
                557, 2267,  542,  392,   73,  603,  233,  255,  528,  397, 1529,
               1023, 1194,  462,  583,   37,  943,  996,  480, 1497,  717,  224,
                219, 1531,  498,   44,  288,  267,  600,   52,  269, 1086,  386,
                176, 2199,  216,   54,  675, 1243,  463,  650,  171,  327,  110,
                774,  509,    8,  197,  136,   12, 1124,   64,  380,  811,  232,
                192,  731,  715,  226,  605,  539, 1491,  323,  240,  179,  702,
                156,   82, 1397,  354,  778,  603, 1001,  385,  986,  203,  149,
                576,  445,  180, 1403,  252,  675, 1351, 2983, 1568,   45,  899,
               3260, 1025,   31,  100, 2055, 4043,   79,  238, 3931, 2351,  595,
                110,  215,    0,  563,  206,  660,  242,  577,  179,  157,  192,
                192, 1848,  792, 1693,   55,  388,  225, 1134, 1172, 1555,   31,
               1582, 1044,  378, 1687, 2915,  280,  765, 2819,  511, 1521,  745,
               2491,  580, 2072, 6450,  578,  745, 1075, 1103, 1549, 1520,  138,
               1202,  296,  277,  351,  391,  950,  459,   62, 1056, 1128,  139,
                420,   87,   71,  814,  603, 1349,  162, 1027,  783,  326,  101,
                876,  381,  905,  156,  419,  239,  119,  129,  467])
```

 # Seed random number generator

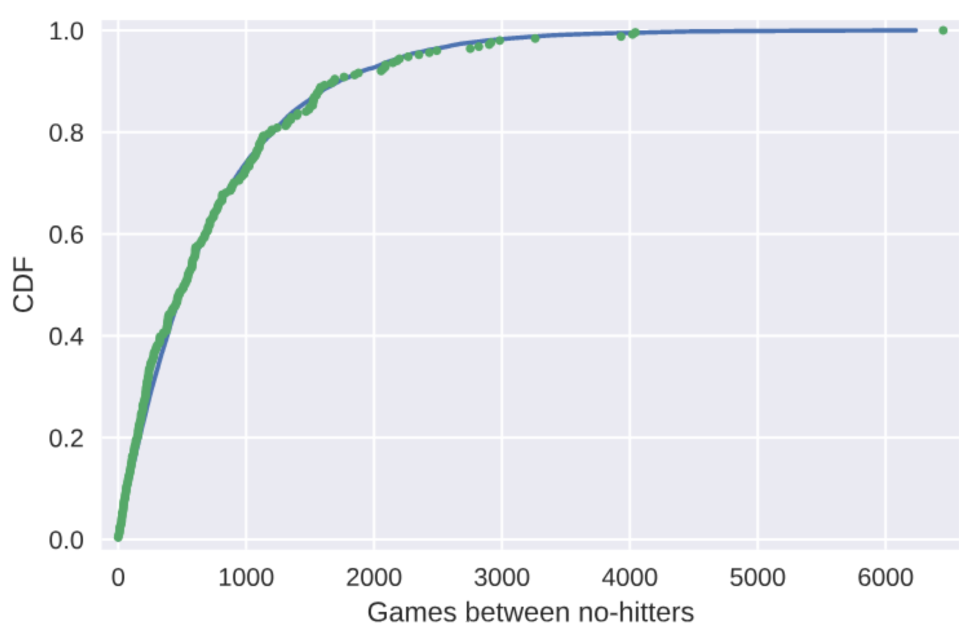
np.random.seed(42)

```
# Compute mean no-hitter time: tau
tau = np.mean(nohitter_times)

# Draw out of an exponential distribution with parameter tau: inter_nohitter_time
inter_nohitter_time = np.random.exponential(tau, 100000)

# Plot the PDF and label axes
_ = plt.hist(inter_nohitter_time,
         bins=50, normed=True, histtype='step')
_ = plt.xlabel('Games between no-hitters')
_ = plt.ylabel('PDF')

# Show the plot
plt.show()
```



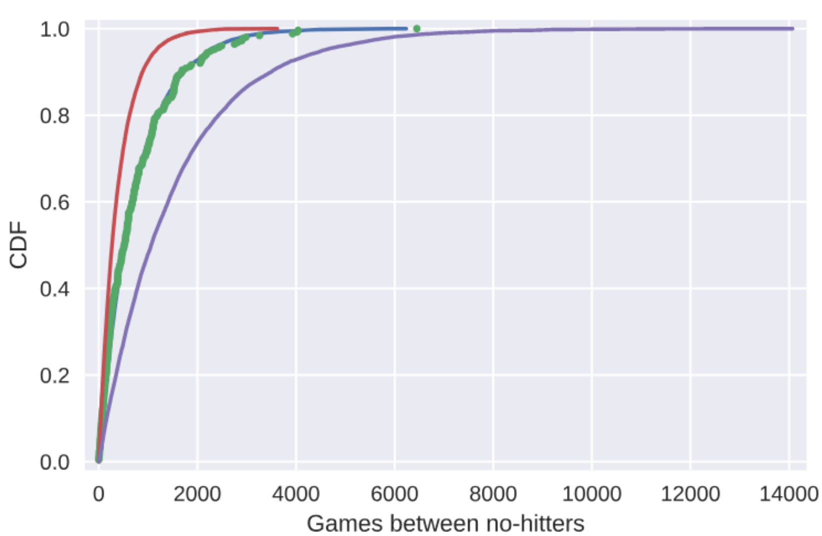
b). <u>Do the data follow our story?</u>

```
#Create an ECDF from real data: x, y
x, y = ecdf(nohitter_times)

# Create a CDF from theoretical samples: x_theor, y_theor
x_theor, y_theor = ecdf(inter_nohitter_time)

# Overlay the plots
plt.plot(x_theor, y_theor)
plt.plot(x, y, marker='.', linestyle='none')

# Margins and axis labels
plt.margins(0.02)
plt.xlabel('Games between no-hitters')
plt.ylabel('CDF')

# Show the plot
plt.show()
```



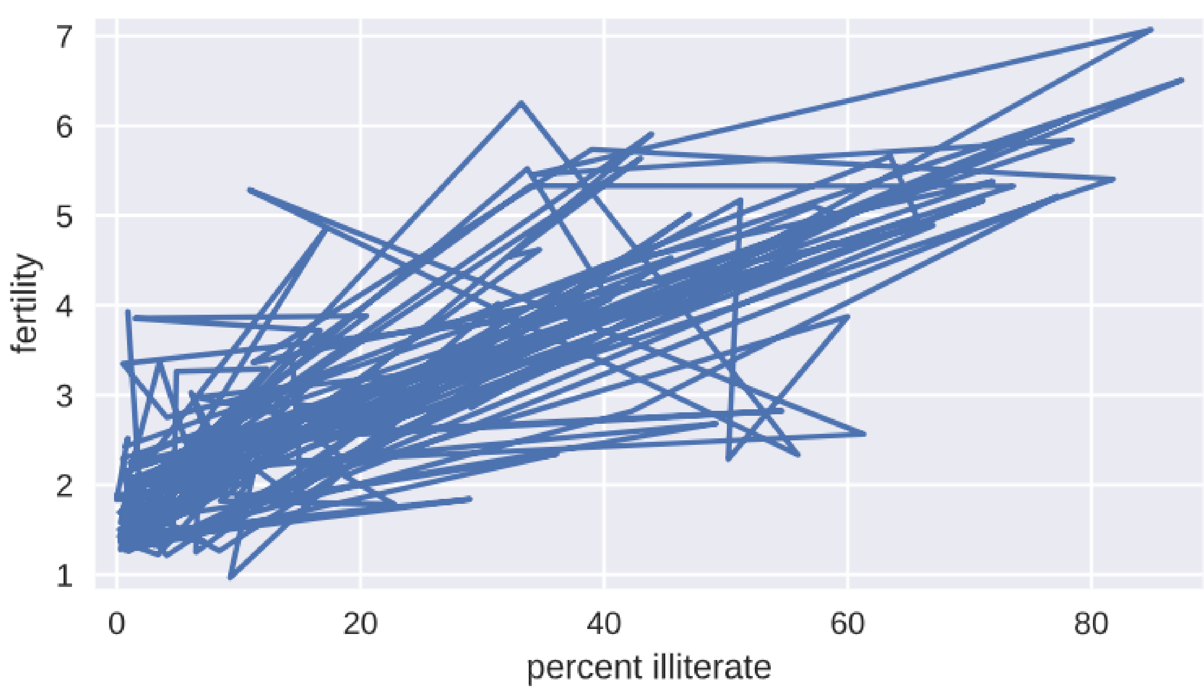
c). <u>How is this parameter optimal</u>

```
#Plot the theoretical CDFs
plt.plot(x_theor, y_theor)
plt.plot(x, y, marker='.', linestyle='none')
plt.margins(0.02)
plt.xlabel('Games between no-hitters')
plt.ylabel('CDF')
# Take samples with half tau: samples_half
samples_half = np.random.exponential(tau/2, 10000)
# Take samples with double tau: samples_double
samples_double = np.random.exponential(2*tau, 10000)
# Generate CDFs from these samples
x_half, y_half = ecdf(samples_half)
x_double, y_double = ecdf(samples_double)
# Plot these CDFs as lines
_ = plt.plot(x_half, y_half)
_ = plt.plot(x_double, y_double)
# Show the plot
plt.show()
```



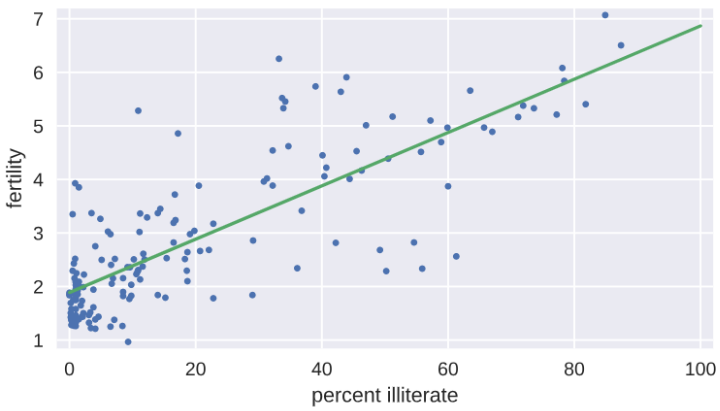
d). EDA of literacy/fertility data:

```
# Plot the illiteracy rate versus fertility
_ = plt.plot(illiteracy, fertility)

# Set the margins and label axes
plt.margins(0.02)
_ = plt.xlabel('percent illiterate')
_ = plt.ylabel('fertility')

# Show the plot
plt.show()

# Show the Pearson correlation coefficient
print(pearson_r(fertility, illiteracy))
```



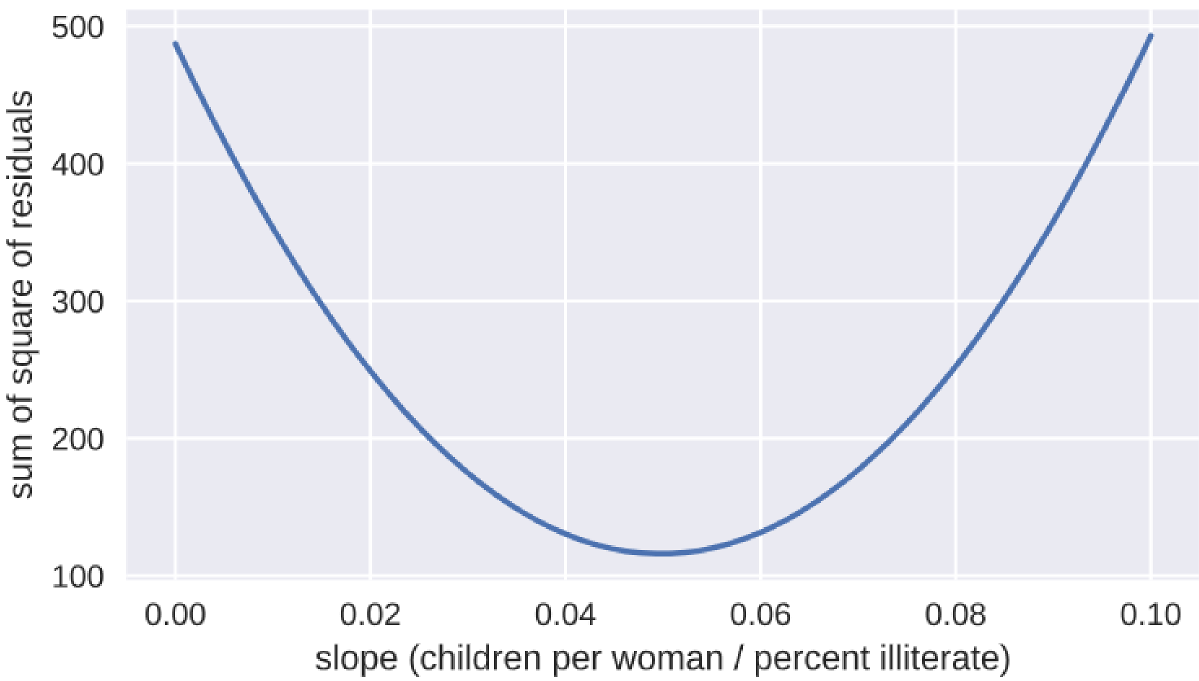
e). Linear Regression

```
# Plot the illiteracy rate versus fertility
_ = plt.plot(illiteracy, fertility, marker='.', linestyle='none')
plt.margins(0.02)
_ = plt.xlabel('percent illiterate')
_ = plt.ylabel('fertility')
# Perform a linear regression using np.polyfit(): a, b
a, b = np.polyfit(illiteracy, fertility, 1)

# Print the results to the screen
print('slope =', a, 'children per woman / percent illiterate')
print('intercept =', b, 'children per woman')

# Make theoretical line to plot
x = np.array([0,100])
y = a * x + b

# Add regression line to your plot
_ = plt.plot(x, y)
# Draw the plot
plt.show()
```



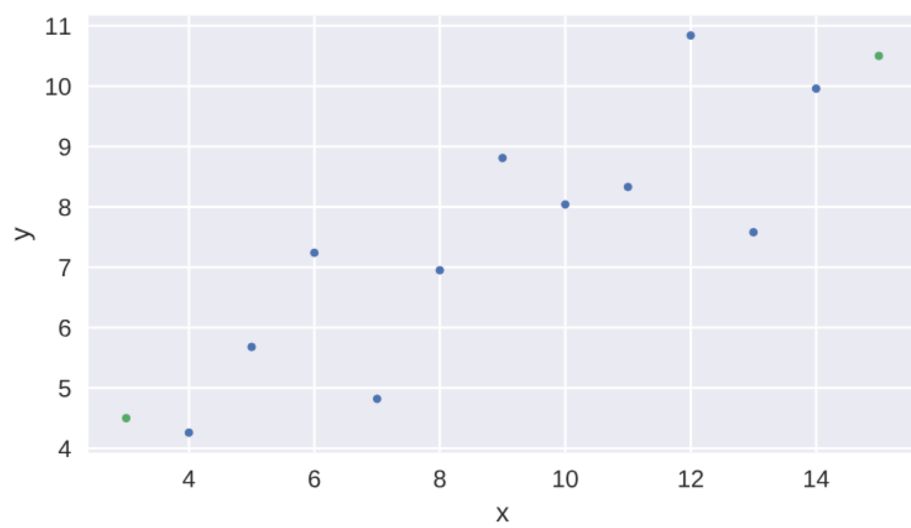
f). How is it optimal??

```
#Specify slopes to consider: a_vals
a_vals = np.linspace(0,0.1,200)

# Initialize sum of square of residuals: rss
rss = np.empty_like(a_vals)

# Compute sum of square of residuals for each value of a_vals
for i, a in enumerate(a_vals):
    rss[i] = np.sum((fertility - a* illiteracy - b)**2)

# Plot the RSS
plt.plot(a_vals, rss, '-')
plt.xlabel('slope (children per woman / percent illiterate)')
plt.ylabel('sum of square of residuals')
plt.show()
```



g). <u>Linear regression on appropriate Anscombe data</u>

```
# Perform linear regression: a, b
a, b = np.polyfit(x,y,1)

# Print the slope and intercept
print(a, b)

#Generate theoretical x and y data: x_theor, y_theor
x_theor = np.array([3, 15])
y_theor = a * x_theor + b

# Plot the Anscombe data and theoretical line
_ = plt.plot(x, y, marker='.', linestyle='none')
_ = plt.plot(x_theor ,y_theor, marker='.', linestyle='none')
# Label the axes
plt.xlabel('x')
plt.ylabel('y')
# Show the plot
plt.show()
```



h). Linear regression on all Anscombe data

```
# Iterate through x,y pairs
for x, y in zip(anscombe_x, anscombe_y):
    # Compute the slope and intercept: a, b
    a, b = np.polyfit(x,y,1)

    # Print the result
    print('slope:', a, 'intercept:', b)
```

<script.py> output:

```
    slope: 0.500090909091 intercept: 3.00009090909
    slope: 0.5 intercept: 3.00090909091
    slope: 0.499727272727 intercept: 3.00245454545
    slope: 0.499909090909 intercept: 3.00172727273
```