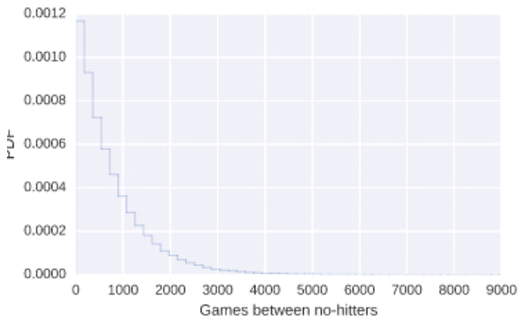
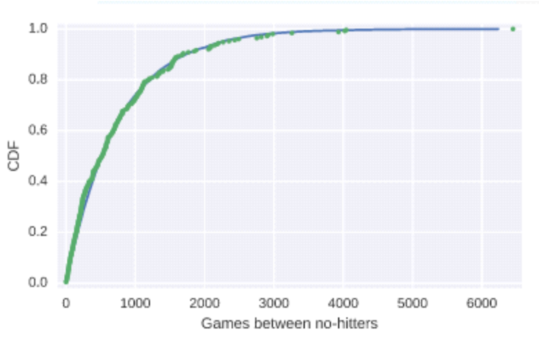
Statistical Thinking in Python (Part 2)

November-13-17 8:25 AM

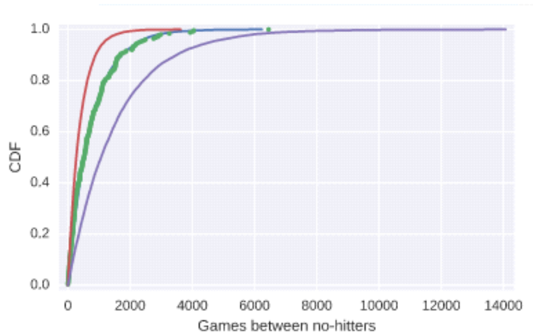
```
#Ch 1 Parameter estimation by optimization
```



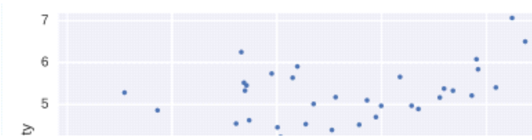

```
#How often do we get no-hitters?
# Seed random number generator
np.random.seed (42)

# 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()
```




```
#Do the data follow our story?
# 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()
```

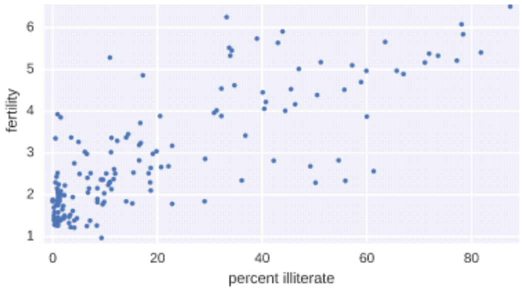



```
#How is this parameter optimal?
# Plot the theoretical CDFs
plt.plot(x_theor, y_theor)
plt.plot(x, y, marker='.', linestyle='none')
plt.margins(0.02)
plt.xlabel('Games between no-hitters')
plt.ylabel('CDF')

# 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()
```

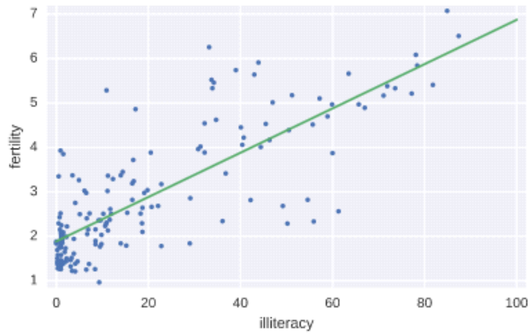



```
#EDA of literacy/fertility data
# Plot the illiteracy rate versus fertility
_ = plt.plot(illiteracy, fertility, marker='.', linestyle='none')
```

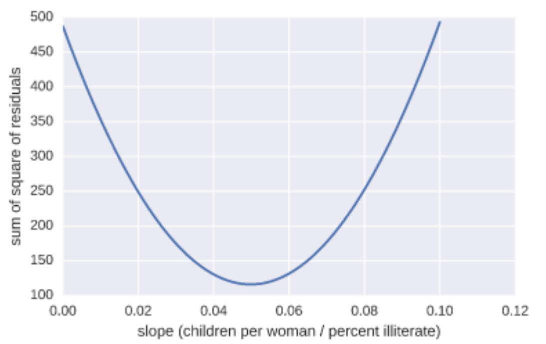
```
#EDA of literacy/fertility data
# Plot the illiteracy rate versus fertility
_ = plt.plot(illiteracy, fertility, marker='.', linestyle='none')

# 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(illiteracy, fertility))
```



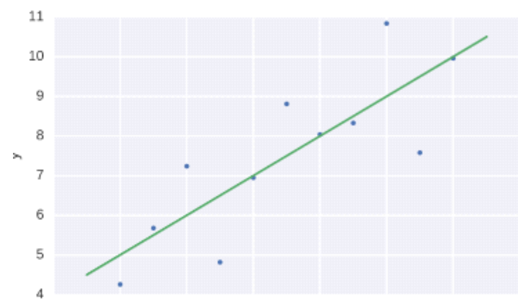

```
#Linear regression
# Plot the illiteracy rate versus fertility
_ = plt.plot(illiteracy, fertility, marker='.', linestyle='none')
plt.margins(0.02)
_ = plt.xlabel('percent illiterate')
_ = plt.ylabel('fertility')

# 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)
_=plt.xlabel ('illiteracy')
_ = plt.ylabel ('fertility')

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



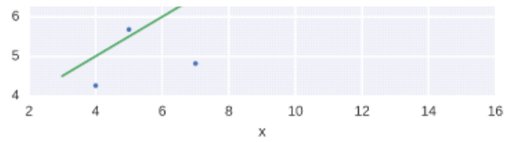

```
#How is it optimal?
# Specify slopes to consider: a_vals
a_vals = np.linspace (0, 0.1, 200)

# 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()
```




```
#Linear regression on appropriate Anscombe data
# 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
```



```
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, '-')

# Label the axes
plt.xlabel('x')
plt.ylabel('y')

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




```
#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)
```