Internet Measurement and Data Analysis (2)

Kenjiro Cho

2014-09-29

review of previous class

theme of the class

- looking at the Internet from different views
  - learn how to measure what is difficult to measure
  - learn how to extract useful information from huge data sets

Class 1 Introduction (9/22)

- Big Data and Collective Intelligence
- Internet measurement
- Large-scale data analysis
- exercise: introduction of Ruby scripting language

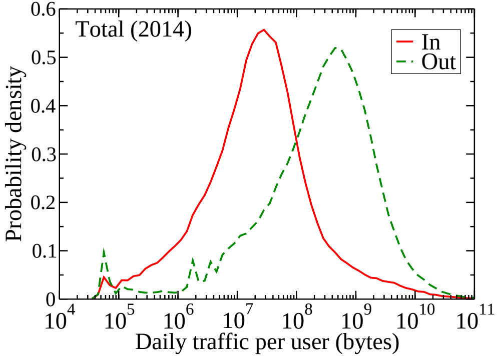
today's topics

Data and variability

- Summary statistics
- Sampling
- How to make good graphs
- exercise: computing summary statistics by Ruby
- exercise: graph plotting by Gnuplot

example: broadband traffic

- daily traffic usage per user (from IIJ, June 2014)
  - highly skewed usage (note that X-axis is log-scale)
  - IN (upload): mean 437MB, mode 28MB
  - OUT(download): mean 1287MB, mode 447MB
- mean is not a good metric in this case



data and variability

- variability of data
  - variability in measurements against the true value
    - the mean should be close to the true value
    - (but, to discuss the precision, we need to understand the variability)
  - variability in measured target itself
    - we need to understatnd the variability
- ways to understand the variability in data
  - summary statistics
  - visualization by graphs

summary statistics

numbers that summarize properties of data

- measure of location:
    - mean, median, mode
- measure of spread:
    - range, variance, standard deviation

measures of location

- mean: average, sensitive to outliers

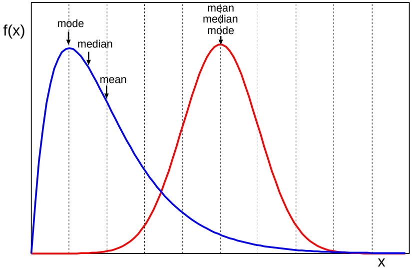
$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

- median: middle value (50th-percentile)

$$x_{median} = \begin{cases} x_{r+1} & \text{when } m \text{ is odd, } m = 2r+1 \\ (x_r + x_{r+1})/2 & \text{when } m \text{ is even, } m = 2r \end{cases}$$

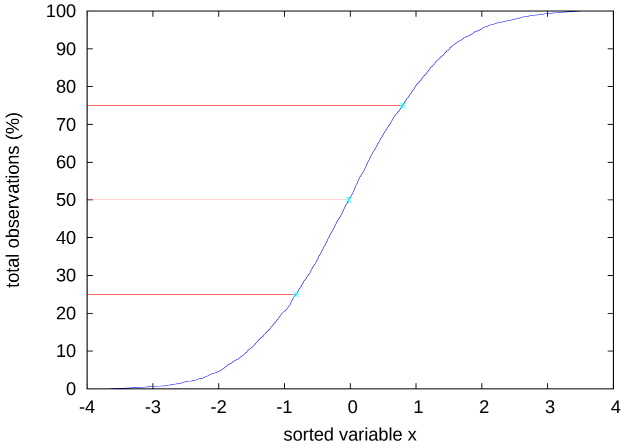
- mode: value with highest frequency

these are same if measurements have symmetric distribution



percentiles

- pth-percentile:
  - p% of the observed values are less than x_p in variable x_i
  - median $=$ 50th-percentile



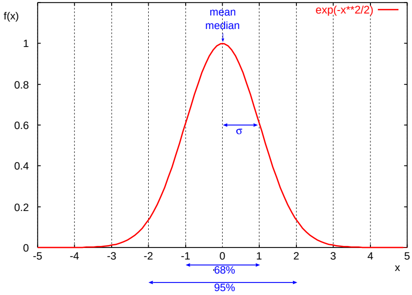
measures of spread

common measures of the spread of a data set

- range: difference between the max and min
- variance:

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

- standard deviation: σ
  - most common measure of statistical dispersion
  - can be directly compared with mean
- for a normal distribution, 68% fall into $(mean \pm stddev)$、95% fall into $(mean \pm 2stddev)$



computing variance

variance:

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

using the above formula, you need to compute the mean first, and then, compute the variance.

you can compute the variance in one-pass with the following formula.

$$\begin{aligned}
\sigma^2 &= \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2 \\
&= \frac{1}{n}\sum_{i=1}^{n}(x_i^2 - 2x_i\bar{x} + \bar{x}^2) \\
&= \frac{1}{n}(\sum_{i=1}^{n}x_i^2 - 2\bar{x}\sum_{i=1}^{n}x_i + n\bar{x}^2) \\
&= \frac{1}{n}\sum_{i=1}^{n}x_i^2 - 2\bar{x}^2 + \bar{x}^2 \\
&= \frac{1}{n}\sum_{i=1}^{n}x_i^2 - \bar{x}^2
\end{aligned}$$

sampling

- investigating the whole population (census): not realistic in most cases
- sampling is needed

sampling for the Internet

- observation points
- time, duration
- packet, flow, IP addresses, user IDs

example: packet sampling methods

- counter-based $1/N$ sampling (deterministic)
  - simple to implement, widely used
  - possible synchronization with targets of measurement
- probabilistic $1/N$ sampling
  - probabilistically select packets (or other objects)
- sampling by time
  - example: take the first minute every hour
- flow-based sampling
  - probabilistically sample new flows
  - observe all packets belonging to a selected flow
  - advantage: able to analyze flow behaviors
- many other sampling methods

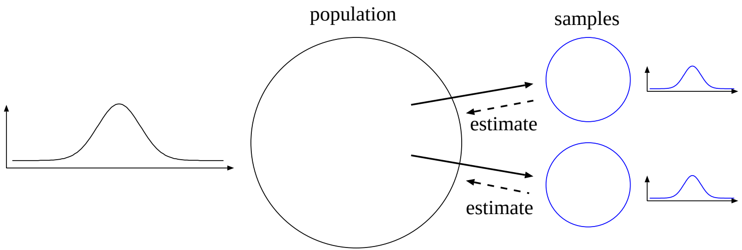
sampling: sample and population

summary statistics and statistical inference

- summary statistics: numbers that summarize properties of data (e.g., mean and standard deviation)
- statistical inference: makes inferences about the population based on samples using statistical methods

population: whole data (difficult or impossible to obtain for most cases)

- need to infer properties of the population from samples
- variables: properties of the population (fixed)
- statistics: inferred values based on samples (varying)



law of large numbers and central limit theorem

law of large numbers

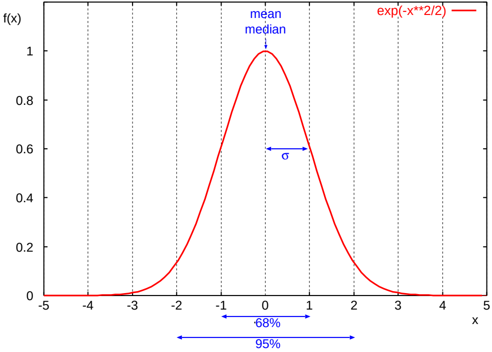
- as the number of samples increases, the sample mean converges to the population mean

central limit theorem

- the mean of a sufficiently large number of samples is approximately normally distributed, regardless of the original distribution. $N(\mu, \sigma/\sqrt{n})$
- when the population is normally distributed, it can be applied even when n is small

normal distribution

- also known as gaussian distribution
- $N(\mu, \sigma)$: defined by 2 parameters: μ:mean, σ:standard deviation
- sum of random variables follows normal distribution
- standard normal distribution: $\mu = 0, \sigma = 1$
- in normal distribution
  - 68% within $(mean - stddev, mean + stddev)$
  - 95% within $(mean - 2 * stddev, mean + 2 * stddev)$



sample mean

- sample mean: $\bar{x}$

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

- sample variance: s^2

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

- sample standard deviation: s
- note: divide sum of squares by $(n-1)$, not by n
  - degree of freedom: the number of independent variables in the sum of squares is $(n-1)$ because of $\bar{x}$

standard error

standard error: standard deviation of sample mean (SE)

$$SE = \sigma/\sqrt{n}$$

- you can improve the precision by increasing the number of samples n
  - standard error becomes smaller but with only $1/\sqrt{n}$
- the distribution of sample mean from a normal distribution $N(\mu, \sigma)$ will be a normal distribution with mean μ and standard deviation SE $= \sigma/\sqrt{n}$

more on sample variance

sample variance: s^2

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

the reason to divide sample variance by $(n-1)$

- sample mean $\bar{x}$ fluctuates around population mean μ
- if sample variance was computed by the same equation, its value S^2 would be smaller than population variance σ^2

if $\bar{x}$ is equal to μ, and its fluctuation follows $N(\mu, \sigma/\sqrt{n})$, the variance becomes $(n-1)/n$ of the population variance.

$$E(S^2) = \frac{n-1}{n}\sigma^2$$

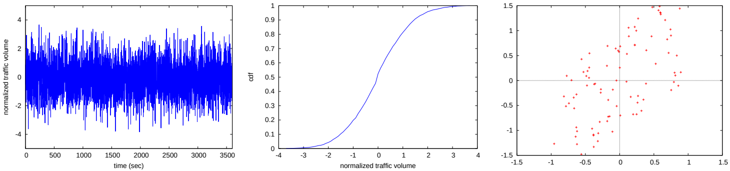
thus,

$$\sigma^2 = \frac{n}{n-1}S^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

graph plotting

it is not easy to understand variability in data only from summary statistics

try to plot several graphs to see characteristics of data



example: finish-time distribution of a city marathon

data:

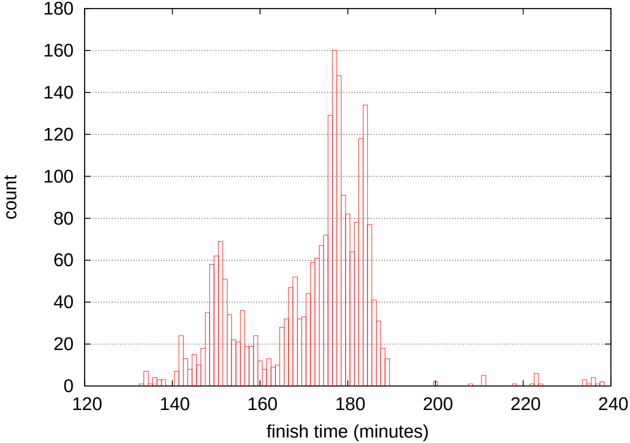
- sample data from a book: P. K. Janert "Gnuplot in Action"

```
# Minutes Count
133 1
134 7
135 1
136 4
137 3
138 3
141 7
142 24
...
```

number of finishers:2,355 mean:171.3(min) standard deviation:14.1 median:176(min)

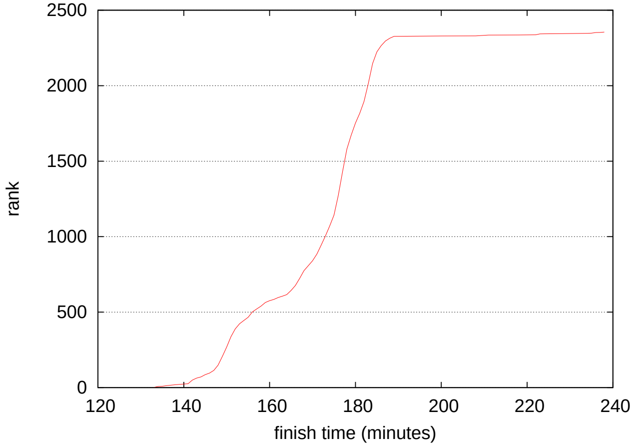
example: finish-time distribution of a city marathon (2)

histogram



example: finish-time distribution of a city marathon (3)

distribution of finish-time and their ranks



guidelines for plotting

require minimum effort from the reader

- label the axes clearly
- label the tics on the axes
- identify individual curves/bars
- select appropriate font size
- use commonly accepted practices
  - zero-origins, math symbols, acronyms
- show variation/distribution of variables
- select ranges properly
- do not present too many items in a single chart
- when comparing data sets, use appropriate normalization
- when comparing plots, use the same scale for the axes
- do not use pie-charts or 3D-effects for technical writing
- when using colors
  - make sure it is readable in black-and-white print
  - make sure readable on data projectors (e.g., do not use yellow)

plotting raw data

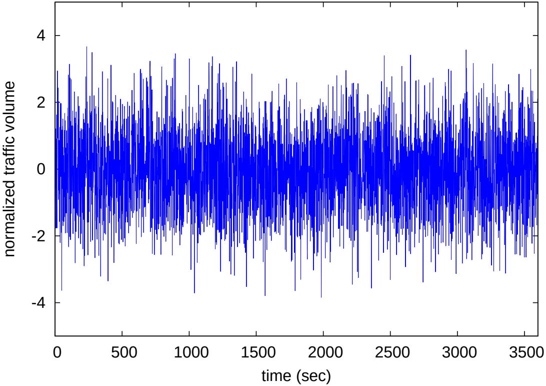
- time series plots
- histograms
- probability plots
- scatter plots

there are many other plotting techniques

time series plots

time-series plots (or other sequence plots) provides a feel for the data

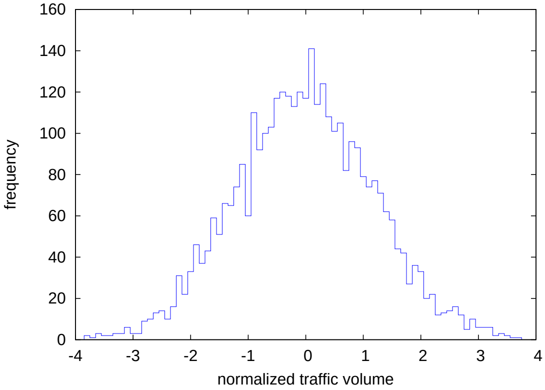
- you can identify
  - shifts in locations
  - shifts in variation
  - outliers



histograms (1/2)

to see distribution of the data set

- split the data into equal-sized bins by value
- count the frequency of each bin
- plot
  - X axis: variable
  - Y axis: frequency



histograms (2/2)

with histograms

- you can identify
  - center (i.e., the location) of the data
  - spread (i.e., the scale) of the data
  - skewness of the data
  - presence of outliers
  - presence of multiple modes in the data

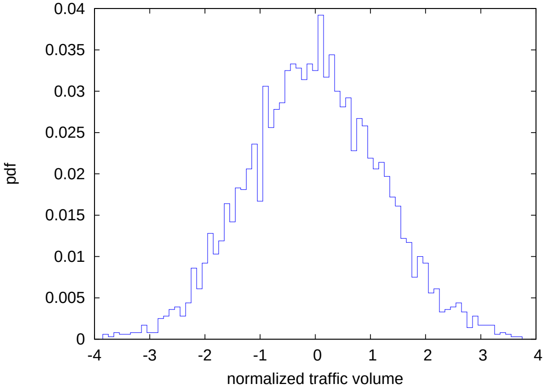
limitations of histograms

- needs appropriate bin size
  - too small: each bin doesn't have enough samples (e.g., empty bins)
  - too large: only few regions available
  - difficult for highly skewed distribution
- enough samples needed

probability density function (pdf)

- normalize the frequency (count)
  - sum of the area under the histogram to be 1
  - divide the count by the total number of observations times the bin width
- probability density function: probability of observing x

$$f(x) = P[X = x]$$



cumulative distribution function (cdf)

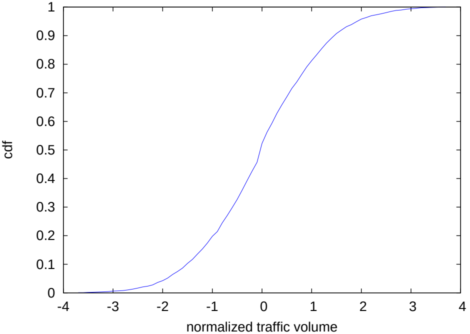
- density function: probability of observing x

$$f(x) = P[X = x]$$

- cumulative distribution function: probability of observing x or less

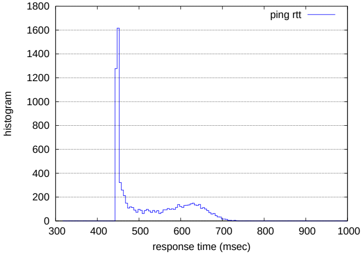
$$F(x) = P[X <= x]$$

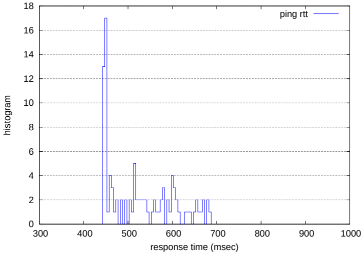
- better than histogram when distribution is highly skewed, sample count is not enough, or outliers are not negligible

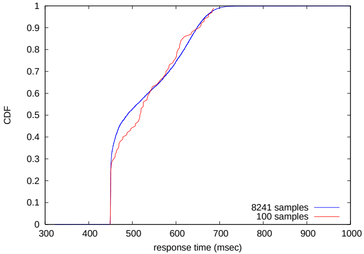


histogram vs cdf

- no need to worry about bin size or sample count for cdf



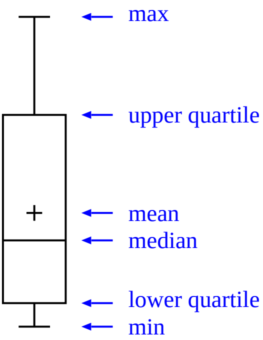




original data (left), 100 samples (right), cdfs (bottom)

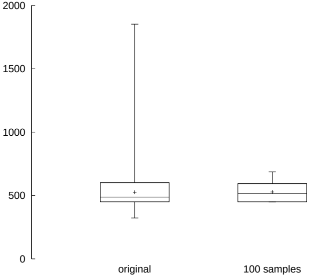
interquartile range

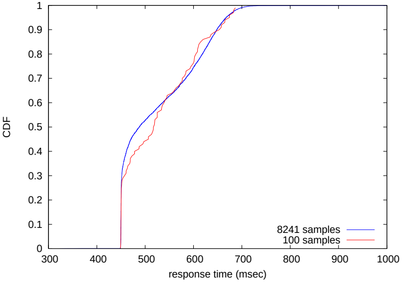
- interquartile range (IQR): range between 1st quartile and 3rd quartile (middle 50%)
- boxplot: one way to show the dispersion
  - box: 25/50/75-percentiles, whiskers: min/max
  - many variations
    - whisker to inner fance $(Q_1 - 1.5IQR, Q_3 + 1.5IQR)$, with outliers
    - use of mean and standard deviation rather than quartiles



boxplot example

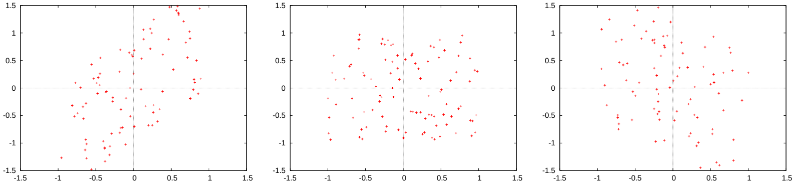
- applied to the previous data sets (original vs 100 samples)
- wiskers: min and max





scatter plots

- explores relationships between 2 variables
  - X-axis: variable X
  - Y-axis: corresponding value of variable Y
- you can identify
  - whether variables X and Y related
    - no relation, positive correlation, negative correlation
  - whether the variation in Y changes depending on X
  - outliers
- examples: positive correlation 0.7 (left), no correlation 0.0 (middle), negative correlation -0.5 (right)



examples: positive correlation 0.7 (left), no correlation 0.0 (middle), negative correlation -0.5 (right)

plotting tools

- gnuplot
  - command-line tool suitable for automated plotting
  - http://gnuplot.info/
- grace
  - comes with graphical user interface
  - powerful for fine-tuning the output
  - http://plasma-gate.weizmann.ac.il/Grace/

previous exercise: a program to count text lines

count the number of text lines in a file given by the argument

```
filename = ARGV[0]
count = 0
file = open(filename)
while text = file.gets
  count += 1
end
file.close
puts count
```

write to “count.rb” and then run it

```
$ ruby count.rb foo.txt
```

rewrite it in a more rubyish way

```
#!/usr/bin/env ruby
count = 0
ARGF.each_line do |line|
  count += 1
end
puts count
```

exercise: computing summary statistics

- mean
- standard deviation
- median

- finish-time data of a city marathon: from P. K. Janert "Gnuplot in Action"

http://web.sfc.keio.ac.jp/~kjc/classes/sfc2014f-measurement/marathon.txt

```
% head marathon.txt
# Minutes Count
133 1
134 7
135 1
136 4
137 3
138 3
141 7
142 24
143 13
```

exercise: computing mean

- read finish-time(in minutes) and the number of finishers from each line, sum up the product, and finally divide it by the total number of finishers

```
# regular expression to read minutes and count
re = /^(\d+)\s+(\d+)/

sum = 0 # sum of data
n = 0 # the number of data
ARGF.each_line do |line|
    if re.match(line)
      min = $1.to_i
      cnt = $2.to_i
      sum += min * cnt
      n += cnt
    end
end

mean = Float(sum) / n

printf "n:%d mean:%.1f\n", n, mean
```

```
% ruby mean.rb marathon.txt
n:2355 mean:171.3
```

exercise: computing standard deviation

- algorithm: $\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$

```
# regular expression to read minutes and count
re = /^(\d+)\s+(\d+)/

data = Array.new
sum = 0 # sum of data
n = 0 # the number of data
ARGF.each_line do |line|
    if re.match(line)
      min = $1.to_i
      cnt = $2.to_i
      sum += min * cnt
      n += cnt
      for i in 1 .. cnt
        data.push min
      end
    end
end
mean = Float(sum) / n
sqsum = 0.0
data.each do |i|
  sqsum += (i - mean)**2
end
var = sqsum / n
stddev = Math.sqrt(var)
printf "n:%d mean:%.1f variance:%.1f stddev:%.1f\n", n, mean, var, stddev
```

```
% ruby stddev.rb marathon.txt
n:2355 mean:171.3 variance:199.9 stddev:14.1
```

exercise: computing standard deviation in one-pass

- one-pass algorithm: $\sigma^2 = \frac{1}{n}\sum_{i=1}^{n} x_i^2 - \bar{x}^2$

```
# regular expression to read minutes and count
re = /^(\d+)\s+(\d+)/

sum = 0 # sum of data
n = 0 # the number of data
sqsum = 0 # su of squares
ARGF.each_line do |line|
    if re.match(line)
      min = $1.to_i
      cnt = $2.to_i
      sum += min * cnt
      n += cnt
      sqsum += min**2 * cnt
    end
end

mean = Float(sum) / n
var = Float(sqsum) / n - mean**2
stddev = Math.sqrt(var)

printf "n:%d mean:%.1f variance:%.1f stddev:%.1f\n", n, mean, var, stddev
```

```
% ruby stddev2.rb marathon.txt
n:2355 mean:171.3 variance:199.9 stddev:14.1
```

exercise: computing median

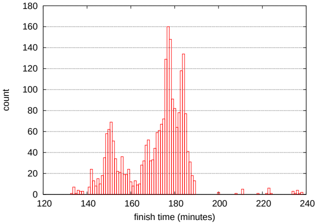
- create an array of each finish time, sort the array by value, and extract the central value

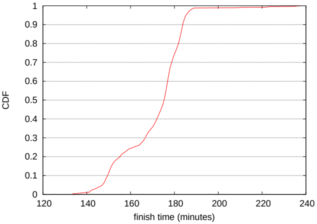
```
# regular expression to read minutes and count
re = /^(\d+)\s+(\d+)/

data = Array.new

ARGF.each_line do |line|
    if re.match(line)
      min = $1.to_i
      cnt = $2.to_i
      for i in 1 .. cnt
        data.push min
      end
    end
end

data.sort! # just in case data is not sorted

n = data.length # number of array elements
r = n / 2 # when n is odd, n/2 is rounded down
if n % 2 != 0
  median = data[r]
else
  median = (data[r - 1] + data[r])/2
end

printf "r:%d median:%d\n", r, median
```

```
% ruby median.rb marathon.txt
r:1177 median:176
```

exercise: gnuplot

- plotting simple graphs using gnuplot
  - to intuitively understand the data





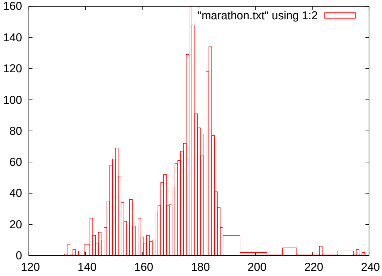
histogram

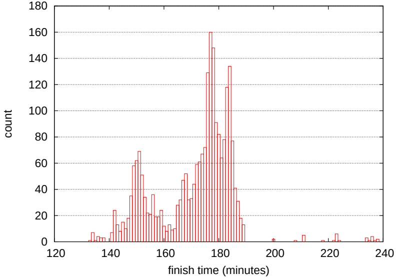
- distribution of finish time of a city marathon

```
plot "marathon.txt" using 1:2 with boxes
```

make the plot look better (right)

```
set boxwidth 1
set xlabel "finish time (minutes)"
set ylabel "count"
set yrange [0:180]
set grid y
plot "marathon.txt" using 1:2 with boxes notitle
```





exercise: plotting CDF of finish-time

original data:

```
# Minutes Count
133 1
134 7
135 1
136 4
137 3
138 3
141 7
142 24
...
```

add cumulative count:

```
# Minutes Count CumulativeCount
133 1 1
134 7 8
135 1 9
136 4 13
137 3 16
138 3 19
141 7 26
142 24 50
...
```

exercise: CDF (2)

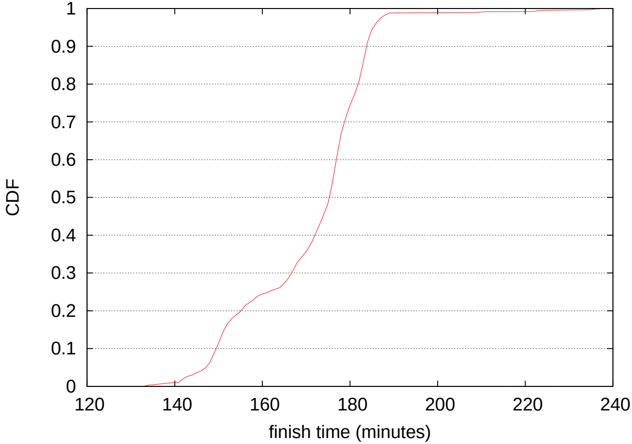
ruby code:

```
re = /^(\d+)\s+(\d+)/
cum = 0
ARGF.each_line do |line|
  begin
    if re.match(line)
      # matched
      time, cnt = $~.captures
      cum += cnt.to_i
      puts "#{time}\t#{cnt}\t#{cum}"
    end
  end
end
```

gnuplot command:

```
set xlabel "finish time (minutes)"
set ylabel "CDF"
set grid y
plot "marathon-cdf.txt" using 1:($3 / 2355) with lines notitle
```

CDF plot of finish-time of city marathon



exercise: saving a plot to an image file

to specify an image format and save to a file:

```
gnuplot> set terminal png
gnuplot> set output "plotfile.png"
gnuplot> replot
```

to run a script:

```
gnuplot> load "scriptfile"
```

to exit:

```
gnuplot> exit
```

summary

Data and variability

- Summary statistics
- Sampling
- How to make good graphs
- exercise: graph plotting by Gnuplot

next class

Class 3 Data recording and log analysis (10/6)

- Network management tools
- Data format
- Log analysis methods
- exercise: log data and regular expression