stats_ch13_confidenceIntervals

August 6, 2024

1 Modern statistics: Intuition, Math, Python, R

1.1 Mike X Cohen (sincxpress.com)

1.1.1 https://www.amazon.com/dp/B0CQRGWGLY

Code for chapter 14 (confidence intervals)

---

2 About this code file:

2.0.1 This notebook will reproduce most of the figures in this chapter (some figures were made in Inkscape), and illustrate the statistical concepts explained in the text. The point of providing the code is not just for you to recreate the figures, but for you to modify, adapt, explore, and experiment with the code.

2.0.2 Solutions to all exercises are at the bottom of the notebook.

This code was written in google-colab. The notebook may require some modifications if you use a different IDE.

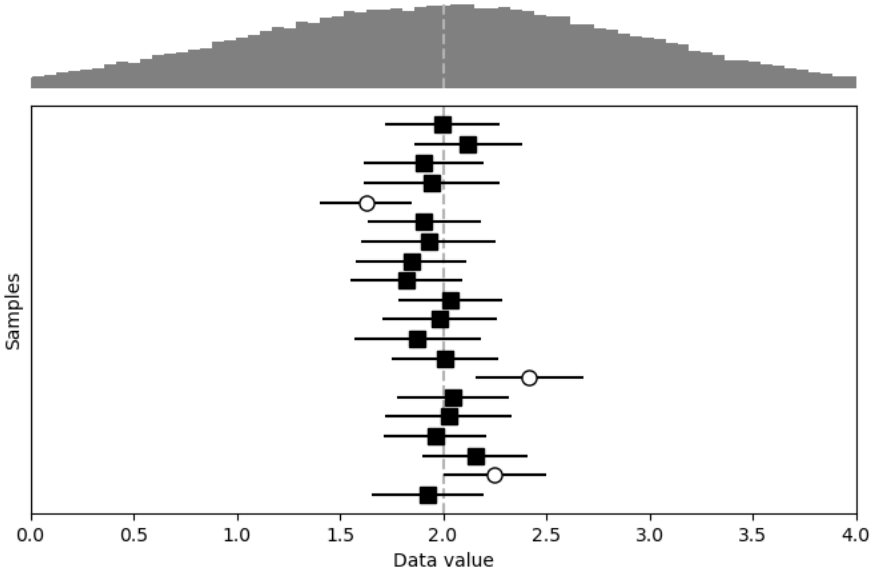
[1]:
```
# import libraries and define global settings
import numpy as np
import scipy.stats as stats
import pandas as pd

import matplotlib.pyplot as plt

# define global figure properties used for publication
import matplotlib_inline.backend_inline
```

3 Figure 13.1: Visualization of confidence intervals

[3]:
```
### Note about the code in this cell: This is to illustrate the concept of␣
 ↪confidence
#   intervals. You will learn all of the code here later in this chapter, so␣
 ↪don't worry
#   if it doesn't make sense now. You can come back after reading the chapter␣
 ↪and you'll
```

```
#   understand all of it!

# a population and its mean
popdata = np.random.randn(100000) + 2
popmean = np.mean(popdata)

# a bunch of samples and their confidence intervals
nSamples   = 20
sampleSize = 50

# setup the figure
fig = plt.figure(figsize=(8,5))
gs = plt.GridSpec(5,1)
ax1 = fig.add_subplot(gs[0])
ax2 = fig.add_subplot(gs[1:])

# draw the population distribution and its mean
ax1.hist(popdata,bins='fd',color='gray')
ax1.axvline(x=popmean,linestyle='--',color=(.7,.7,.7))
ax2.axvline(x=popmean,linestyle='--',color=(.7,.7,.7))

# run the experiment
for i in range(nSamples):
  # draw a sample
  sample = np.random.choice(popdata,sampleSize,replace=False)

  # compute its mean and stdev
  mean = np.mean(sample)
  sem  = np.std(sample,ddof=1) / np.sqrt(sampleSize)

  # confidence interval from scipy
  CI = stats.t.interval(.95,sampleSize-1,loc=mean,scale=sem)

  # plot it
  if popmean>CI[0] and popmean<CI[1]:
    c,s = 'k','s'
  else:
    c,s = 'w','o'
  ax2.
 ↪errorbar(mean,i,xerr=mean-CI[0],color='k',marker=s,markerfacecolor=c,markersize=8)

ax1.set_xlim([0,4])
ax1.axis('off')

ax2.set_ylabel('Samples')
ax2.set_yticks([])
ax2.set_xlim([0,4])
```

```
ax2.set_xlabel('Data value')

plt.figure(figsize=(9,3))
plt.tight_layout()
#plt.savefig('confint_whatIsCI.png')
plt.show()
```

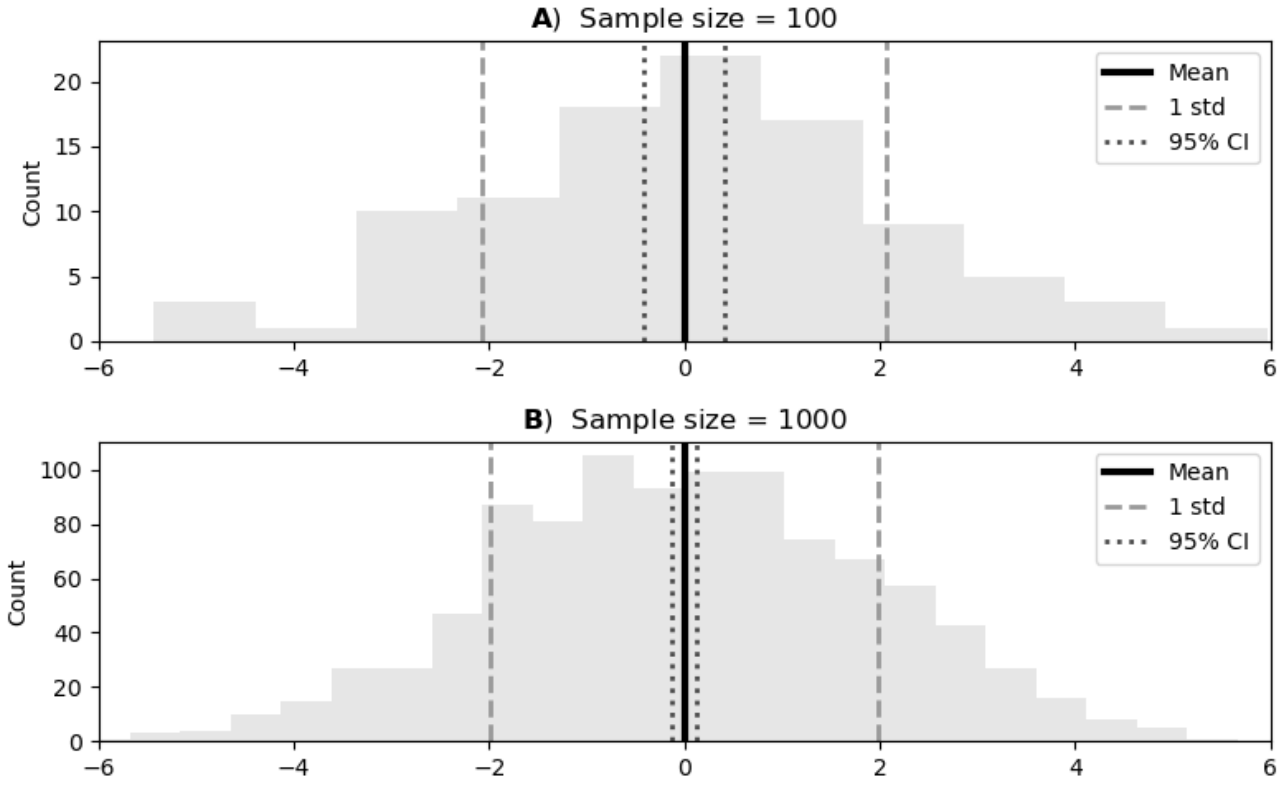



```
<Figure size 900x300 with 0 Axes>
```

4 Figure 13.2: CI vs std

[4]:
```
sampleSizes = [100,1000]
confLevel = .95

_,axs = plt.subplots(2,1,figsize=(8,5))

for ax,N,t in zip(axs,sampleSizes,['A','B']):

  # Generate a random sample of size N
  data = np.random.randn(N)*2
  # force the mean to be zero
  data -= np.mean(data)

  # mean and standard deviation
  mean = np.mean(data)
  stdev = np.std(data,ddof=1)

  # Calculate 95% confidence interval
  stderr = stdev / np.sqrt(len(data))
```

```
  conf_interval = stats.t.interval(confLevel, N-1, loc=mean, scale=stderr)

  # Plot the histogram
  ax.hist(data,bins='fd',color=(.9,.9,.9))

  # Plot the mean
  ax.axvline(mean,color='k',linewidth=3,label='Mean')

  # one standard deviation of the mean
  ax.axvline(mean-stdev,color=(.6,.6,.6),linewidth=2,linestyle='--',label='1␣
↪std')
  ax.axvline(mean+stdev,color=(.6,.6,.6),linewidth=2,linestyle='--')

  # Plot the confidence interval
  ax.axvline(conf_interval[0],color=(.3,.3,.3),linewidth=2,linestyle=':',␣
↪label='95% CI')
  ax.axvline(conf_interval[1],color=(.3,.3,.3),linewidth=2,linestyle=':')

  ax.set(xlim=[-6,6],ylabel='Count')
  ax.set_title(rf'$\bf{{{t}}}$)  Sample size = {N}')
  ax.legend()

plt.tight_layout()
#plt.savefig('confint_stdVsCI.png')
plt.show()
```



5 Analytic confidence interval

[5]:
```
conflevel = .95
n = 20
tStar = stats.t.isf((1-conflevel)/2,n-1)
print(tStar)
```

```
2.093024054408263
```

[6]:
```
# simulation parameters
mean = 2.3
stdev = 3.2
N = 48
conflevel = .95

# confidence interval from formula
tStar = stats.t.isf((1-conflevel)/2,N-1)
conf_int_me = [ mean - tStar*(stdev/np.sqrt(N)), \
                mean + tStar*(stdev/np.sqrt(N)) ]
# confidence interval from scipy
conf_int_sp = stats.t.interval(confLevel,N-1,
                               loc=mean,scale=stdev/np.sqrt(N))
print(conf_int_me)
print(conf_int_sp)
```

```
[1.3708168597897181, 3.2291831402102815]
(1.3708168597897181, 3.2291831402102815)
```

6 Bootstrapping

[7]:
```
S = [1,2,3,4]
S = [2,2,3,3]

print('    Sample    |  Mean')
print('----------------------')
print(f'{S}  |  {np.mean(S):.2f}')

for i in range(5):
  # bootstrap a random sample
  b = np.random.choice(S,len(S),replace=True)
  # note: replace=True is the default setting; I set it here to emphasize its␣
 ↪importance.

  # and print it and its mean
  print(f'{list(np.sort(b))}  |  {np.mean(b):.2f}')
```

```
    Sample    |  Mean
----------------------
[2, 2, 3, 3]  |  2.50
[2, 2, 3, 3]  |  2.50
[2, 3, 3, 3]  |  2.75
[2, 2, 3, 3]  |  2.50
[2, 2, 3, 3]  |  2.50
[2, 2, 3, 3]  |  2.50
```

[8]:
```
# FYI, function to compute nChooseK
from scipy.special import comb
comb(3,3,repetition=True) # annoying, the "replace" parameter is called␣
 ↪"repetition" and defaults to False
```

[8]:
```
10.0
```

7 [Don't peak!] How many boots?

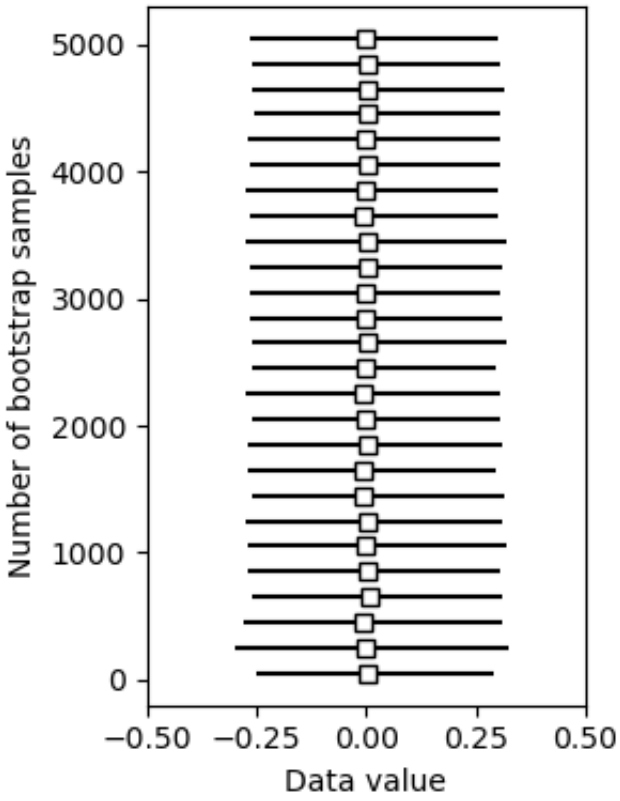
7.0.1 Note about the code below: This code includes the solution to Exercise 6 (empirical confidence intervals),

7.0.2 which produces Figure 13.3.

7.0.3 If you want to challenge yourself on Exercise 6, don't look at the code here :P

[10]:
```
# parameters
samplesize = 50

# draw a random sample from the population
dataSample = np.random.randn(samplesize)**2
dataSample -= np.mean(dataSample)

numboots = np.arange(50,5051,step=200)
CIs = np.zeros((len(numboots),2))
bmm = np.zeros(len(numboots))

for i,nb in enumerate(numboots):
  bootmeans = [np.mean(np.random.choice(dataSample,samplesize)) for booti in␣
 ↪range(nb)]
  CIs[i,:] = np.percentile(bootmeans,[2.5,97.5])
  bmm[i] = np.mean(bootmeans)

# and plot
plt.figure(figsize=(3,4))
plt.errorbar(bmm, numboots, xerr=[bmm-CIs[:,0],CIs[:,1]-bmm],
             marker='s', color='k', markerfacecolor='w', linestyle='None')
```

```
plt.xlim([-.5,.5])

plt.ylabel('Number of bootstrap samples')
plt.xlabel('Data value')

plt.tight_layout()
#plt.savefig('confint_nBoots.png')
plt.show()
```



[11]:
```
np.percentile(bootmeans,[2.5,97.5])
```

[11]:
```
array([-0.26657014,  0.30258864])
```

8 CI for hypothesis testing

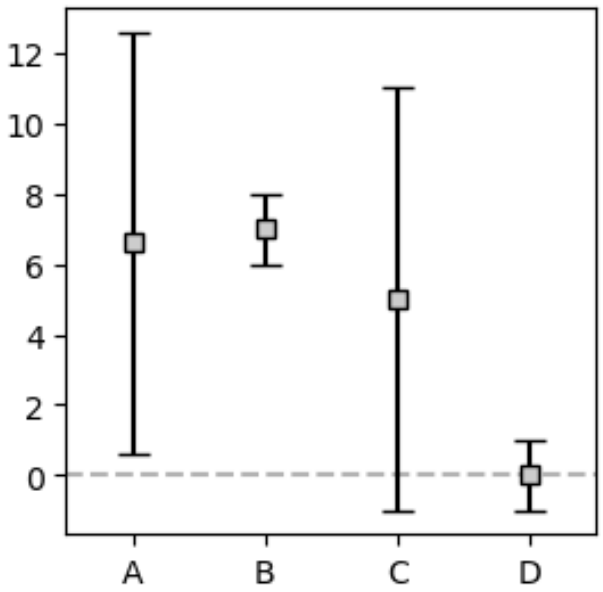
[12]:
```
# simulation parameters
mean = 1.3
stdev = 5.2
N = 48
conflevel = .95
```

```
# confidence interval from scipy
confint = stats.t.interval(confLevel,N-1,loc=mean,scale=stdev/np.sqrt(N))

print(confint)
```

```
(-0.20992260284170783, 2.809922602841708)
```

9 Figure 13.4: Qualitative interpretation of confidence intervals

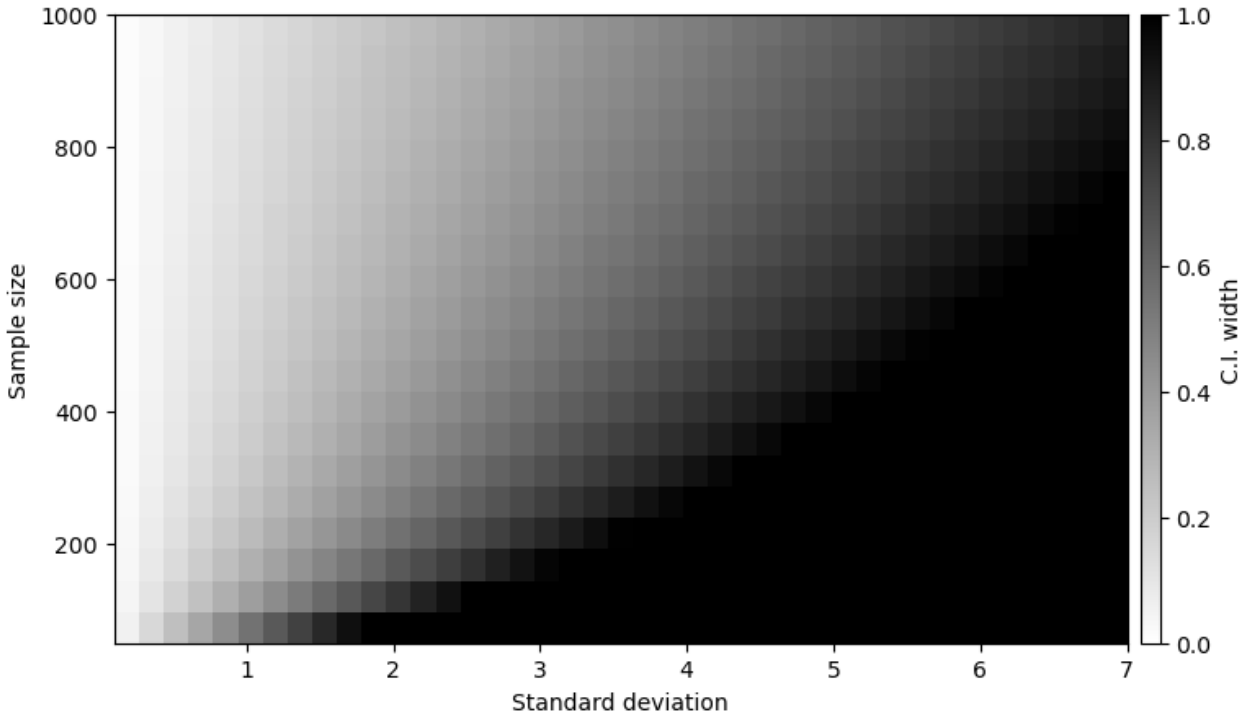
[13]:
```
eSizes = [6,1,6,1]
means = [6.6,7,5,0]

plt.figure(figsize=(3,3))
plt.errorbar(range(4),means,eSizes,marker='s',color='k', markerfacecolor=(.8,.8,.
 ↪8),capsize=5,linestyle='None')
plt.axhline(y=0,color=(.7,.7,.7),linestyle='--',zorder=-1)
plt.xticks(range(4),labels=['A','B','C','D'])
plt.xlim([-.5,3.5])

#plt.savefig('confint_qualitative.png')
plt.show()
```



10 Exercise 1

[15]:
```
# parameters
samplesizes = np.arange(50,1001,step=50)
stdevs = np.linspace(.1,7,41)
```

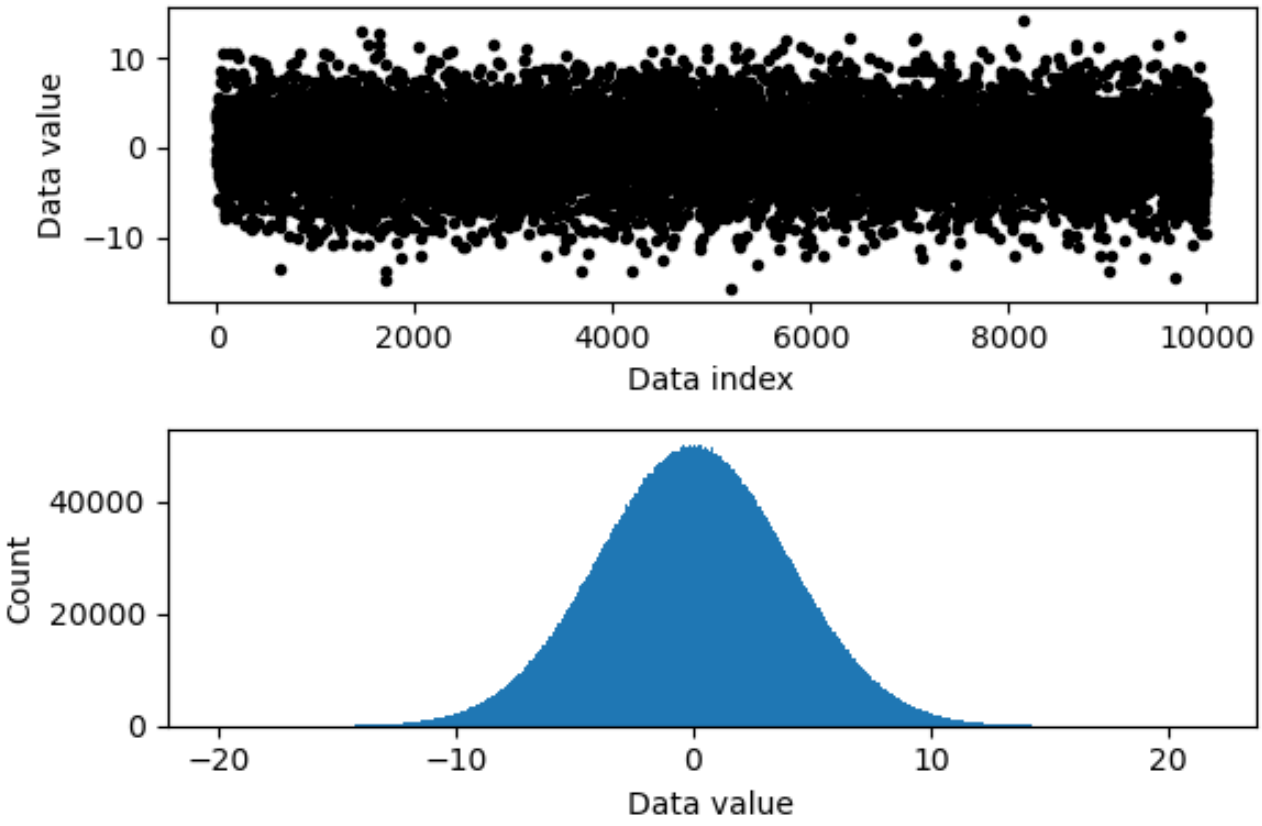
```
# initialization
CIs = np.zeros((len(samplesizes),len(stdevs)))

# experiment
for ni,N in enumerate(samplesizes):
  for si,s in enumerate(stdevs):
    # SEM
    sem = s/np.sqrt(N)
    # When mean=0, the CI is symmetric, so its width is double the positive side
    CIs[ni,si] = 2*stats.t.interval(.95,N-1,loc=0,scale=sem)[1]

# draw the figure
fig,ax = plt.subplots(1,figsize=(8,5))
cax = ax.imshow(CIs,origin='lower',aspect='auto',vmin=0,vmax=1,cmap='gray_r',
          extent=[stdevs[0],stdevs[-1],samplesizes[0],samplesizes[-1]])
ax.set(xlabel='Standard deviation',ylabel='Sample size')

# colorbar
cbar_ax = fig.add_axes([.91,.11,.015,.77])
cbar = plt.colorbar(cax,cax=cbar_ax,label='C.I. width')

plt.figure(figsize=(6,3))
#plt.savefig('confint_ex1.png')
plt.show()
```



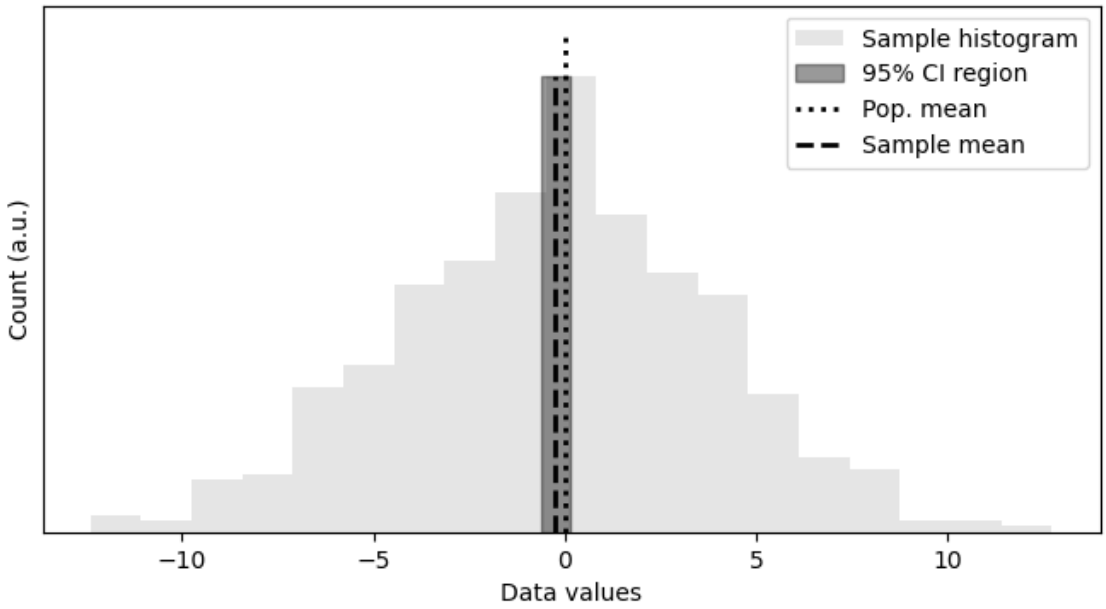

```
<Figure size 600x300 with 0 Axes>
```

11 Exercise 2

[16]:
```
## simulate data
popN = int(1e7)  # lots and LOTS of data!!

# the data
population = (4*np.random.randn(popN))**1 # the "1" here is for exercise 2-4;␣
↪change to "2" for exercise 5

# we can calculate the exact population mean
popMean = np.mean(population)

# let's see it
fig,ax = plt.subplots(2,1,figsize=(6,4))

# TIP: plot only every 1000th sample
ax[0].plot(population[::1000],'k.')
ax[0].set_xlabel('Data index')
ax[0].set_ylabel('Data value')

ax[1].hist(population,bins='fd')
ax[1].set_ylabel('Count')
ax[1].set_xlabel('Data value')
plt.tight_layout()
plt.show()
```



[18]:
```
## draw a random sample

# parameters
samplesize = 500
confidence =  95 # in percent

# compute sample mean
dataSample = np.random.choice(population,samplesize)
samplemean = np.mean(dataSample)
samplestd  = np.std(dataSample,ddof=1)

# compute confidence intervals
confint = stats.t.interval(confidence/100,samplesize-1,
                           loc=samplemean,scale=samplestd/np.sqrt(samplesize))
# graph everything
fig,ax = plt.subplots(1,figsize=(8,4))

# the histogram
h = ax.hist(dataSample,bins='fd',color='k',alpha=.1,label='Sample histogram')
ytop = np.max(h[0]) # convenient variable for histogram peak value

# confidence interval area
ax.fill_between([confint[0],confint[1]],[0,0],[ytop,ytop],color='k',alpha=.
 ↪4,label=f'{confidence}% CI region')

# now add the linesindicating population means
ax.plot([popMean,popMean],[0,ytop*1.1],'k:',linewidth=2,label='Pop. mean')
ax.plot([samplemean,samplemean],[0,ytop],'k--',linewidth=2,label='Sample mean')

# uncomment to zoom in
# ax.set_xlim([confint[0]-np.diff(confint),confint[1]+np.diff(confint)])

# some more adjustments
ax.legend()
ax.set(yticks=[],xlabel='Data values',ylabel='Count (a.u.)')

plt.figure(figsize=(6,3))
plt.tight_layout()
#plt.savefig('confint_ex2.png')
plt.show()
```




```
<Figure size 600x300 with 0 Axes>
```

12 Exercise 3

[19]:
```
## Test whether more samples have the population mean inside their CI

# parameters
numExperiments = 5000
samplesize = 500

# initialize the matrix of whether the population mean is inside the CI
withinCI = np.zeros(numExperiments)

# run the experiment
for expi in range(numExperiments):
  # compute sample mean and CI
  dataSample = np.random.choice(population,samplesize)
  samplemean = np.mean(dataSample)
  samplestd  = np.std(dataSample,ddof=1)
  confint    = stats.t.interval(confidence/100,samplesize-1,
                            loc=samplemean,scale=samplestd/np.sqrt(samplesize))
  # determine whether the true mean is inside this CI
  if popMean>confint[0] and popMean<confint[1]:
    withinCI[expi] = 1
```

```
print('%g%% of sample C.I.''s contained the true population mean.'%(100*np.
 ↪mean(withinCI)))
```

```
95.58% of sample C.I.s contained the true population mean.
```

13 Exercise 4

[20]:
```
# print out some sample confint's for different sample sizes:
confint
```

[20]:
```
(-0.43992821284096884, 0.25131402332192854)
```

14 Exercise 5

[ ]:
```
# The discrepency here is due to the assumptions of the analytic formula for␣
 ↪computing confidence intervals.
#
# In Exercises 2-4, the normality assumption was met. Even when the sample size␣
 ↪was tiny, the purely random
# sampling in combination with the purely Gaussian distribution meant that the␣
 ↪assumptions underlying the
# confidence intervals were still met (although with a small sample size, the␣
 ↪confidence intervals were so huge
# as to be completely useless from a practical perspective, but still valid␣
 ↪mathematically).
#
# In contrast, Exerise 5 violated the normality assumption. Now, with large␣
 ↪samples, the CLT kicked in and still
# gave us a good result. But the small sample sizes stretched the CLT to its␣
 ↪limits, meaning it was no longer applicable.
# And that in turn meant that the confidence intervals were junk and not␣
 ↪reliable.
#
```

15 Exercise 6

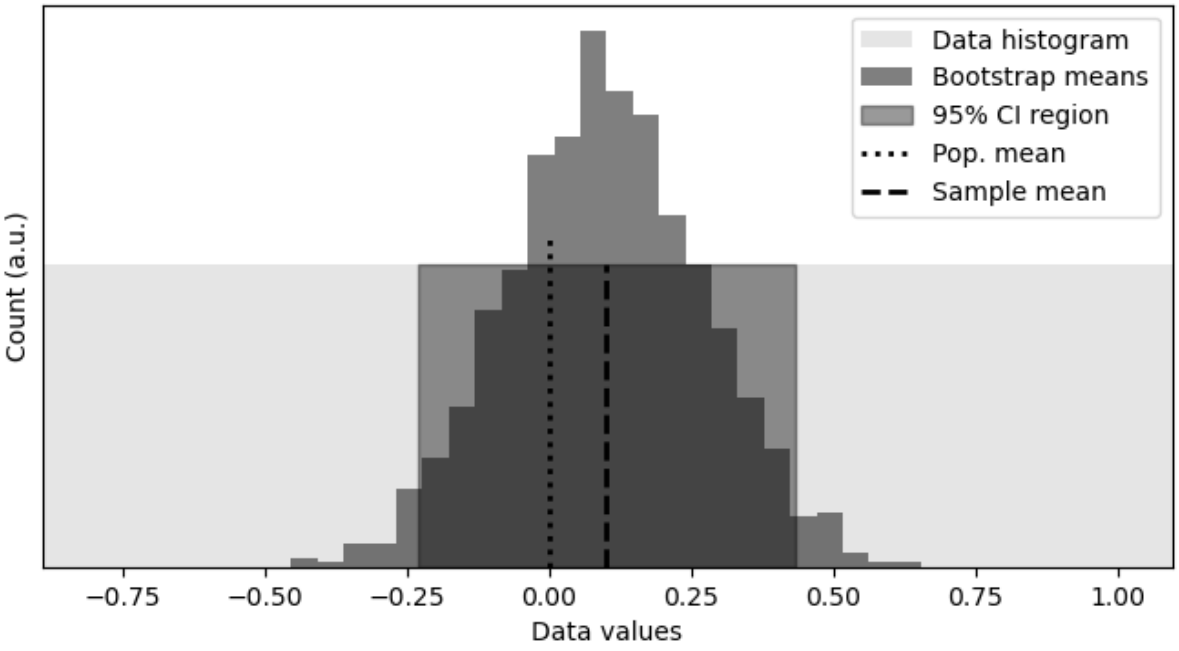
[21]:
```
# parameters
samplesize = 500
numBoots   = 1000

# draw a random sample from the population
dataSample = np.random.choice(population,samplesize)

# we'll need these statistics later
samplemean = np.mean(dataSample)
```

```
samplestd  = np.std(dataSample,ddof=1)

# initialize a vector to store the bootstrapped means
bootmeans = np.zeros(numBoots)

## now for bootstrapping
for booti in range(numBoots):
  # create a bootstrap sample
  bootsample = np.random.choice(dataSample,samplesize)

  # and compute its mean
  bootmeans[booti] = np.mean(bootsample)

# Coding note: I used a multi-line for-loop above for procedural clarity. A␣
 ↪list-comprehension is more compact:
#bootmeans = [np.mean(np.random.choice(dataSample,samplesize)) for booti in␣
 ↪range(numBoots)]

# find confidence intervals (hard-coded to 95%!)
confintB = np.percentile(bootmeans,[2.5,97.5]) # B for bootstrap
confintB
```

[21]:
```
array([-0.22909179,  0.43329552])
```

[23]:
```
# graph everything
fig,ax = plt.subplots(1,figsize=(8,4))

# the histogram
h = ax.hist(dataSample,bins='fd',color='k',alpha=.1,label='Data histogram')
ax.hist(bootmeans,bins='fd',color='k',alpha=.5,label='Bootstrap means')
ytop = np.max(h[0]) # convenient variable for histogram peak value

# confidence interval area
ax.fill_between([confintB[0],confintB[1]],[0,0],[ytop,ytop],color='k',alpha=.
 ↪4,label=f'{confidence}% CI region')

# lines indicating population means
ax.plot([popMean,popMean],[0,ytop*1.1],'k:',linewidth=2,label='Pop. mean')
ax.plot([samplemean,samplemean],[0,ytop],'k--',linewidth=2,label='Sample mean')

# uncomment to zoom in
ax.set_xlim([confintB[0]-np.diff(confintB),confintB[1]+np.diff(confintB)])

# some more adjustments
ax.legend()
ax.set(yticks=[],xlabel='Data values',ylabel='Count (a.u.)')
```

```
plt.figure(figsize=(6,3))
plt.tight_layout()
#plt.savefig('confint_ex6.png')
plt.show()
```




```
<Figure size 600x300 with 0 Axes>
```

[24]:
```
## compare against the analytic confidence interval

# compute confidence intervals (again, hard-coding to 95%)
confintA = stats.t.interval(.975,samplesize-1,
                            loc=samplemean,scale=samplestdev/np.sqrt(samplesize))
print(f'Empirical CI(95%) = ({confintB[0]:.3f},{confintB[1]:.3f})')
print(f'Analytic  CI(95%) = ({confintA[0]:.3f},{confintA[1]:.3f})')
```

```
Empirical CI(95%) = (-0.229,0.433)
Analytic  CI(95%) = (-0.286,0.485)
```

16 Exercise 7

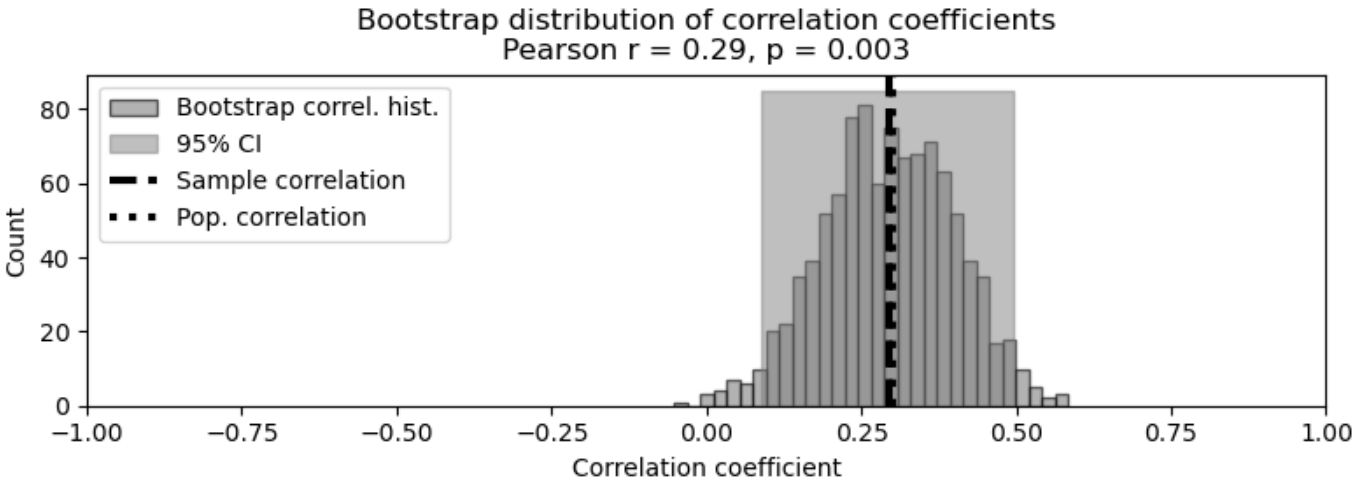
[25]:
```
# simulation params
samplesize = 100
trueR = .3 # true population correlation

# generate the data
X = np.random.randn(samplesize,2)
X[:,1] = X[:,0]*trueR + X[:,1]*np.sqrt(1-trueR**2)
```

```
# confirmation
np.corrcoef(X.T)
```

[25]:
```
array([[1.        , 0.29452738],
       [0.29452738, 1.        ]])
```

[26]:
```
### the python function (note: hard-coded to 95% confidence level)
def corr_CI(X,nBoots=1000):

  # initialize bootstrap sample-mean differences
  bootstrap_r = np.zeros(nBoots)

  # empirical sample size
  samplesize = X.shape[0]

  # generate bootstrap samples and correlate
  for i in range(nBoots):
    boot_idx = np.random.choice(range(samplesize), size=samplesize)
    bootstrap_r[i] = np.corrcoef(X[boot_idx,:].T)[0,1]
    # I prefer numpy's matrix input here for dealing with sampled data

  # Compute the percentiles for the bootstrapped coefficients distribution
  return np.percentile(bootstrap_r,[2.5,97.5]),bootstrap_r
### end function definition

# observed correlation coefficient (here using scipy to get a p-value)
obs_r,obs_p = stats.pearsonr(X[:,0],X[:,1])

# get empirical confidence intervals (using default of 1000 bootstraps)
CI,bootstrap_r = corr_CI(X)

# choose a color for the CI area based on significance
areacolor = 'gray' if np.sign(CI[0])==np.sign(CI[1]) else 'red'

### plotting
plt.figure(figsize=(8,3))

# histogram of the bootstrapped coefficients
plt.hist(bootstrap_r, bins=30, edgecolor='k', color='gray', alpha=.
 ↪6,label='Bootstrap correl. hist.')

# area for confidence interval (using fill_betweenx here for some variety :P )
plt.fill_betweenx([0, plt.gca().get_ylim()[1]], CI[0], CI[1], color=areacolor,␣
 ↪alpha=.5, label='95% CI')

# lines indicating coefficients
```

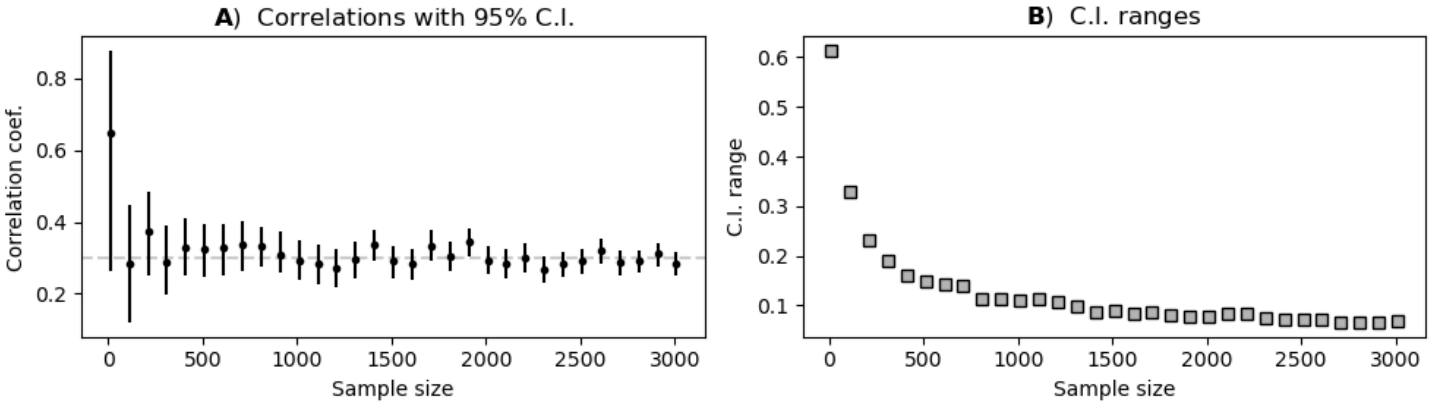
```
plt.axvline(obs_r,color='k',linestyle='--',linewidth=3, label='Sample␣
↪correlation')
plt.axvline(trueR,color='k',linestyle=':',linewidth=3, label='Pop. correlation')

plt.legend()
plt.xlim([-1,1])
plt.xlabel('Correlation coefficient')
plt.ylabel('Count')
plt.title(f'Bootstrap distribution of correlation coefficients\nPearson r =␣
↪{obs_r:.2f}, p = {obs_p:.3f}',loc='center')

plt.tight_layout()
#plt.savefig('confint_ex7.png')
plt.show()
```



17 Exercise 8

[27]:
```
# simulation params
samplesizes = np.arange(10,3011,step=100)

# matrix to store the sample sizes
bootCI = np.zeros((len(samplesizes),2))
obs_r = np.zeros(len(samplesizes))
# reduce the number of bootstraps
nBoots = 500 # number of samples

# run the experiment!
for idx,N in enumerate(samplesizes):
  # generate the data
  X = np.random.randn(N,2)
  X[:,1] = X[:,0]*trueR + X[:,1]*np.sqrt(1-trueR**2)
```

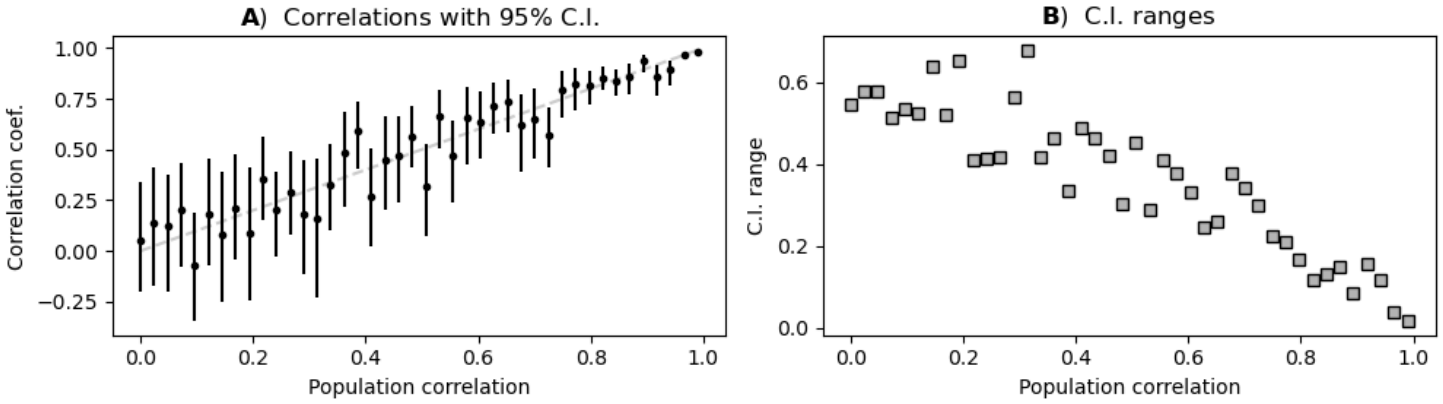
```
  # observed correlation coefficient
  obs_r[idx] = np.corrcoef(X.T)[0,1]

  # get confidence intervals
  bootCI[idx,:] = corr_CI(X,nBoots)[0]

## the plot
_,axs = plt.subplots(1,2,figsize=(10,3))
axs[0].errorbar(samplesizes, obs_r, yerr=[obs_r-bootCI[:,0],bootCI[:,1]-obs_r],
                marker='.',color='k',linestyle='None')
axs[0].axhline(y=trueR,color=(.8,.8,.8),linestyle='--',zorder=-10)
axs[0].set(xlabel='Sample size',ylabel='Correlation coef.')
axs[0].set_title(r'$\bf{A}$)  Correlations with 95% C.I.')

axs[1].plot(samplesizes,np.diff(bootCI,axis=1),'ks',markerfacecolor=(.7,.7,.7))
axs[1].set(xlabel='Sample size',ylabel='C.I. range')
axs[1].set_title(r'$\bf{B}$)  C.I. ranges')

plt.tight_layout()
#plt.savefig('confint_ex8.png')
plt.show()
```



18 Exercise 9

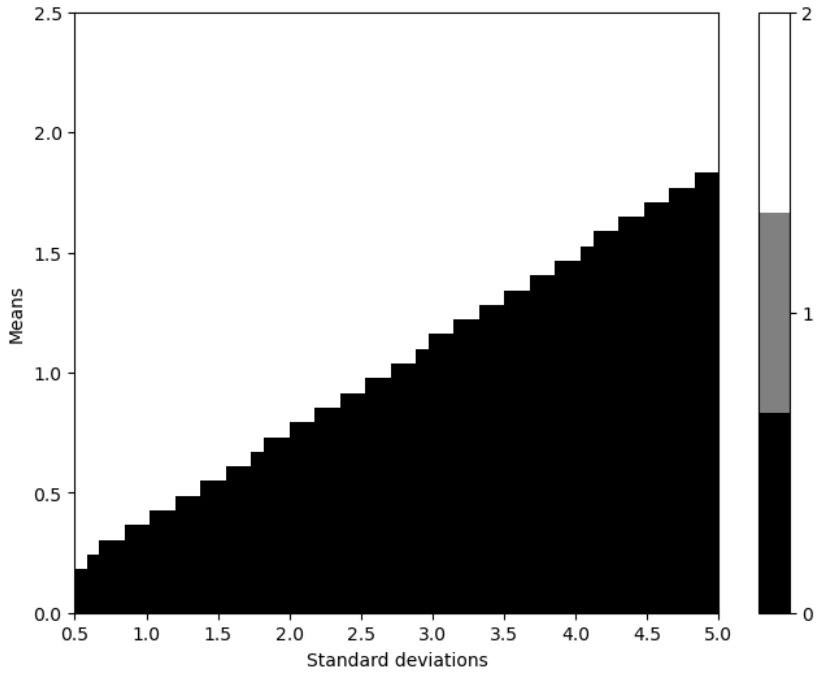
[28]:
```
# simulation params
samplesize = 50

# range of correlations
coefs = np.linspace(0,.99,42)

# matrix to store the sample sizes
bootCI = np.zeros((len(coefs),2))
obs_r = np.zeros(len(coefs))
```

```
# run the experiment!
for idx,r in enumerate(coefs):
  # generate the data
  X = np.random.randn(samplesize,2)
  X[:,1] = X[:,0]*r + X[:,1]*np.sqrt(1-r**2)

  # observed correlation coefficient
  obs_r[idx] = np.corrcoef(X.T)[0,1]

  # confidence intervals
  bootCI[idx,:] = corr_CI(X,nBoots)[0]

## the plot
_,axs = plt.subplots(1,2,figsize=(10,3))
axs[0].errorbar(coefs, obs_r, yerr=[obs_r-bootCI[:,0],bootCI[:,1]-obs_r],
                marker='.',color='k',linestyle='None')
axs[0].plot(coefs,coefs,color=(.8,.8,.8),linestyle='--',zorder=-10)
axs[0].set(xlabel='Population correlation',ylabel='Correlation coef.')
axs[0].set_title(r'$\bf{A}$)  Correlations with 95% C.I.')

axs[1].plot(coefs,np.diff(bootCI,axis=1),'ks',markerfacecolor=(.7,.7,.7))
axs[1].set(xlabel='Population correlation',ylabel='C.I. range')
axs[1].set_title(r'$\bf{B}$)  C.I. ranges')

plt.tight_layout()
#plt.savefig('confint_ex9.png')
plt.show()
```



19 Exercise 10

[31]:
```
# simulation parameters
means = np.linspace(0,2.5,41)
stds = np.linspace(.5,5,51)
sampsize = 30

# initialize output matrix
statsmatrix = np.zeros((len(means),len(stds)))

# run the experiment!
for mi in range(len(means)):
  for si in range(len(stds)):
    # SEM
    sem = stds[si]/np.sqrt(sampsize)

    # confidence interval
    CI = stats.t.interval(.95,sampsize-1,loc=means[mi],scale=sem)

    # t/p values
    tval = means[mi] / sem
    pval = 2*stats.t.sf(tval,sampsize-1)

    # build up the stats matrix according to significances
    statsmatrix[mi,si] += int(CI[0]>0) # only positive means, so we need only␣
 ↪test whether the lower bound is negative
    statsmatrix[mi,si] += int(pval<.05)

# customized colormap
import matplotlib.colors as colors
cmap = plt.get_cmap('gray',3)

plt.figure(figsize=(8,6))
plt.imshow(statsmatrix,vmin=0,vmax=2,cmap=cmap,origin='lower',aspect='auto',
           extent=[stds[0],stds[-1],means[0],means[-1]])
plt.xlabel('Standard deviations')
plt.ylabel('Means')
plt.gca().spines[['right','top']].set_visible(True)
plt.colorbar(ticks=[0,1,2])

plt.figure(figsize=(4,3))
plt.tight_layout()
#plt.savefig('confint_ex10.png')
plt.show()
```



<Figure size 400x300 with 0 Axes>

[32]:
```
# Confirm that no pixels are 1
np.unique(statsmatrix)
```

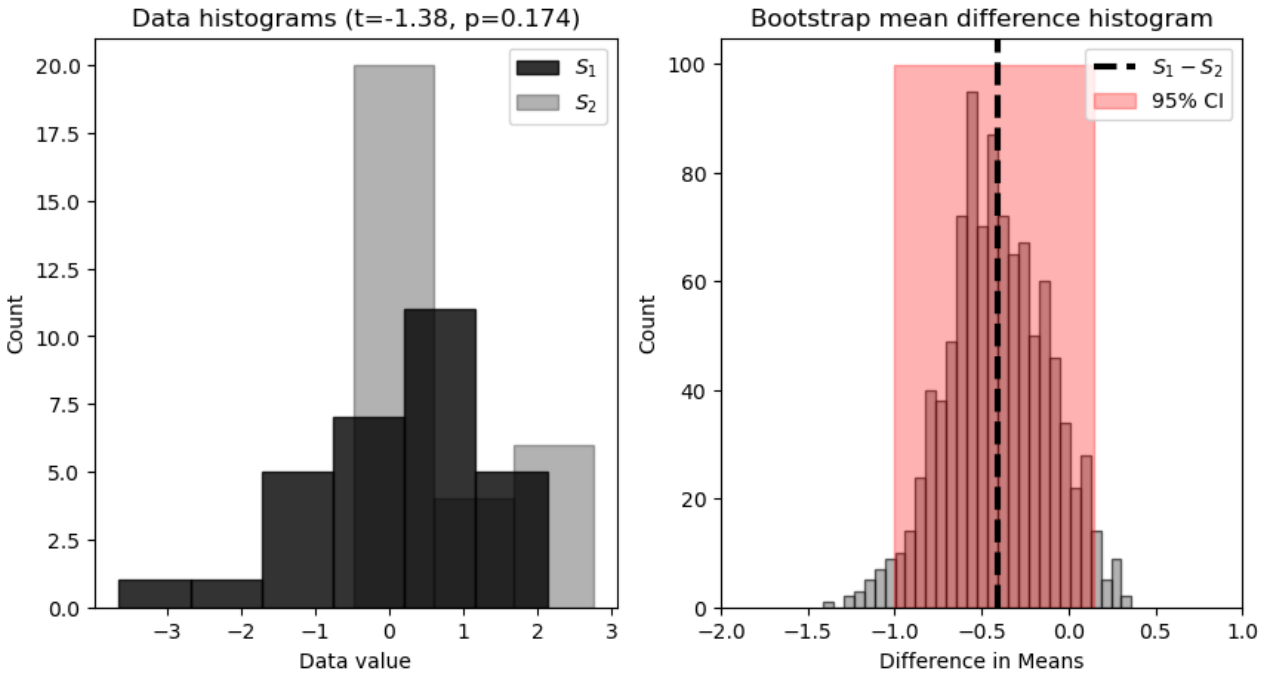
[32]: array([0., 2.])

20 Exercise 11

[33]:
```
# generate the data
sampsize = 30

sample1 = np.random.randn(sampsize)
sample2 = np.random.randn(sampsize)**2
sample2 = sample2-np.mean(sample2) + .5 # mean-center then mean-shift

# compute the ttest
tres = stats.ttest_ind(sample1,sample2)
```

[35]:
```
# CI parameters
nBoots = 1000 # number of samples

# observed difference in means
obs_diff = np.mean(sample1) - np.mean(sample2)

# initialize bootstrap sample-mean differences
bootstrap_diffs = np.zeros(nBoots)
```

```
# Generate bootstrap samples.
# Note that each sample is independently resampled, and then the difference of␣
 ↪means is calculated.
for i in range(nBoots):
  boot_sample1 = np.random.choice(sample1, size=sampsize)
  boot_sample2 = np.random.choice(sample2, size=sampsize)
  bootstrap_diffs[i] = np.mean(boot_sample1) - np.mean(boot_sample2)

# empirical confidence intervals (hard-coded to 95%)
CI_B = np.percentile(bootstrap_diffs,[2.5,97.5])

# choose a color for the CI area based on significance
areacolor = 'gray' if np.sign(CI_B[0])==np.sign(CI_B[1]) else 'red'

### plotting
_,axs = plt.subplots(1,2,figsize=(10,5))

# data distributions
axs[0].hist(sample1, bins='fd', color='k', edgecolor='k', alpha=.8,␣
 ↪label=r'$S_1$')
axs[0].hist(sample2, bins='fd', color='k', edgecolor='k', alpha=.3,␣
 ↪label=r'$S_2$')
axs[0].set(xlabel='Data value',ylabel='Count')
axs[0].legend()
axs[0].set_title(f'Data histograms (t={tres.statistic:.02f}, p={tres.pvalue:.
 ↪03f})')

# bootstrapping distribution
axs[1].hist(bootstrap_diffs, bins=30, edgecolor='k',color='gray', alpha=.6)
axs[1].axvline(obs_diff,color='k',linestyle='--',linewidth=3, label=r'$S_1-S_2$')
axs[1].fill_betweenx([0, plt.gca().get_ylim()[1]], CI_B[0], CI_B[1],␣
 ↪color=areacolor, alpha=.3, label='95% CI')
axs[1].legend()
axs[1].set(xlim=[-2,1],xlabel='Difference in Means',ylabel='Count')
axs[1].set_title(f'Bootstrap mean difference histogram',loc='center')

plt.figure(figsize=(8,3))
plt.tight_layout()
#plt.savefig('confint_ex11.png')
plt.show()
```

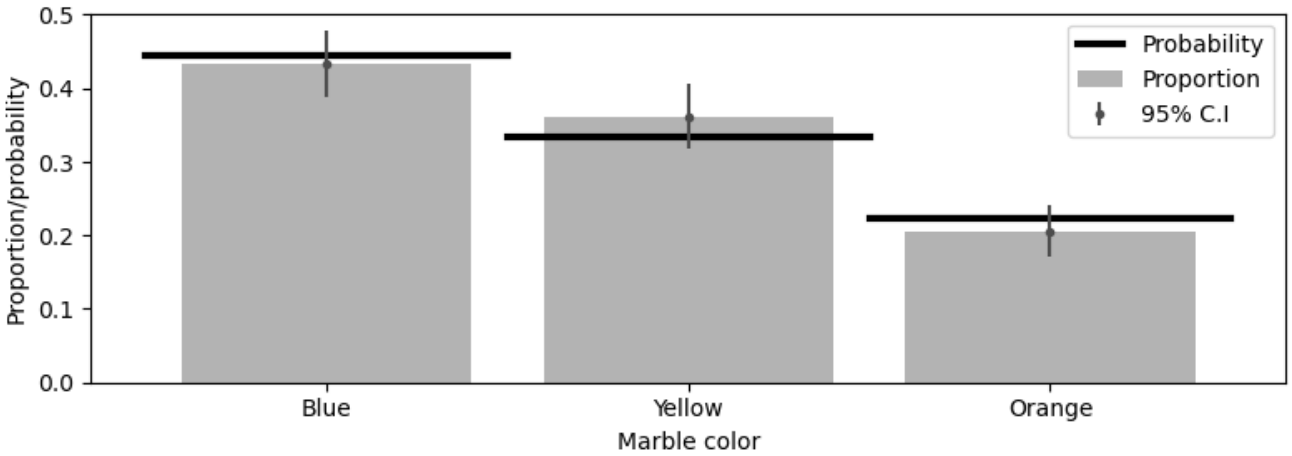


<Figure size 800x300 with 0 Axes>

21 Exercise 12

[36]:
```
# colored marble counts
blue   = 40
yellow = 30
orange = 20
totalMarbs = blue + yellow + orange

# put them all in a jar
jar = np.hstack((1*np.ones(blue),2*np.ones(yellow),3*np.ones(orange)))

# now we draw a sample of 500 marbles (with replacement)
numDraws = 500
marbSample = np.random.choice(jar,size=numDraws)

# bootstrapping for empirical confidence intervals
nBoots = 1000
bootProps = np.zeros((3,nBoots))
for i in range(nBoots):
  # bootstrap sample
  bootmarbs = np.random.choice(marbSample,size=numDraws)

  # empirical proportions of this sample
  for j in range(1,4):
    bootProps[j-1,i] = sum(bootmarbs==j)/numDraws
```

```
# confidence intervals
CI = np.array([ np.percentile(bootProps[0,:],[2.5,97.5]),
                np.percentile(bootProps[1,:],[2.5,97.5]),
                np.percentile(bootProps[2,:],[2.5,97.5]) ])

# empirical proportions of colors drawn
props = np.array([ sum(marbSample==1) / numDraws,
                   sum(marbSample==2) / numDraws,
                   sum(marbSample==3) / numDraws ] )

# plot those against the theoretical probability
plt.figure(figsize=(8,3))
plt.bar([1,2,3],props,label='Proportion',color=(.7,.7,.7))
plt.plot([0.5, 1.5],[blue/totalMarbs, blue/
 ↪totalMarbs],'k',linewidth=3,label='Probability')
plt.plot([1.5, 2.5],[yellow/totalMarbs,yellow/totalMarbs],'k',linewidth=3)
plt.plot([2.5, 3.5],[orange/totalMarbs,orange/totalMarbs],'k',linewidth=3)

plt.errorbar([1,2,3],props,yerr=[props-CI[:,0],CI[:,1]-props],
                marker='.',color=(.3,.3,.3),linestyle='None',label='95% C.I')

plt.xticks([1,2,3],labels=('Blue','Yellow','Orange'))
plt.xlabel('Marble color')
plt.ylabel('Proportion/probability')
plt.legend()

plt.tight_layout()
#plt.savefig('confint_ex12.png')
plt.show()
```



22 Exercise 13

[37]:
```
# import data
df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/
 ↪arrhythmia/arrhythmia.data',
                 usecols = np.arange(9),
                 names   =␣
 ↪['age','sex','height','weight','qrs','p-r','q-t','t','p'])

# make a copy of the original data matrix
df_z = df.copy()
cols2zscore = []
for col in df_z.columns:
  if not (col=='sex'):
    df_z[col] = (df[col] - df[col].mean()) / df[col].std(ddof=1)

zThresh = 3.29 # p<.001
df_clean = df.copy()
df_clean[np.abs(df_z)>zThresh] = np.nan  # both tails
```

[38]:
```
# Note that the methods mean(), std(), and count() in Pandas exclude NaN's.

for col in df.columns:
  # original data
  mean = df[col].mean()
  std  = df[col].std(ddof=1)
  n    = df[col].count()
  ci   = stats.t.ppf(.975,n-1) * std/np.sqrt(n)
  print(f'{col:>6} initial: {mean:6.2f} +/- {ci:.2f}')

  # repeat for cleaned
  mean = df_clean[col].mean()
  std  = df_clean[col].std(ddof=1)
  n    = df_clean[col].count()
  ci   = stats.t.ppf(.975,n-1) * std/np.sqrt(n)
  print(f'{col:>6} cleaned: {mean:6.2f} +/- {ci:.2f}\n')
```

```
   age initial:  46.47 +/- 1.52
   age cleaned:  46.47 +/- 1.52

   sex initial:   0.55 +/- 0.05
   sex cleaned:   0.55 +/- 0.05

height initial: 166.19 +/- 3.44
height cleaned: 163.84 +/- 0.96

weight initial:  68.17 +/- 1.53
weight cleaned:  68.33 +/- 1.36

   qrs initial:  88.92 +/- 1.42
   qrs cleaned:  87.48 +/- 1.07

   p-r initial: 155.15 +/- 4.15
   p-r cleaned: 160.38 +/- 2.49

   q-t initial: 367.21 +/- 3.09
   q-t cleaned: 368.05 +/- 2.84

     t initial: 169.95 +/- 3.29
     t cleaned: 168.28 +/- 2.97

     p initial:  90.00 +/- 2.39
     p cleaned:  91.14 +/- 1.73
```