stats_ch16_permutation

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

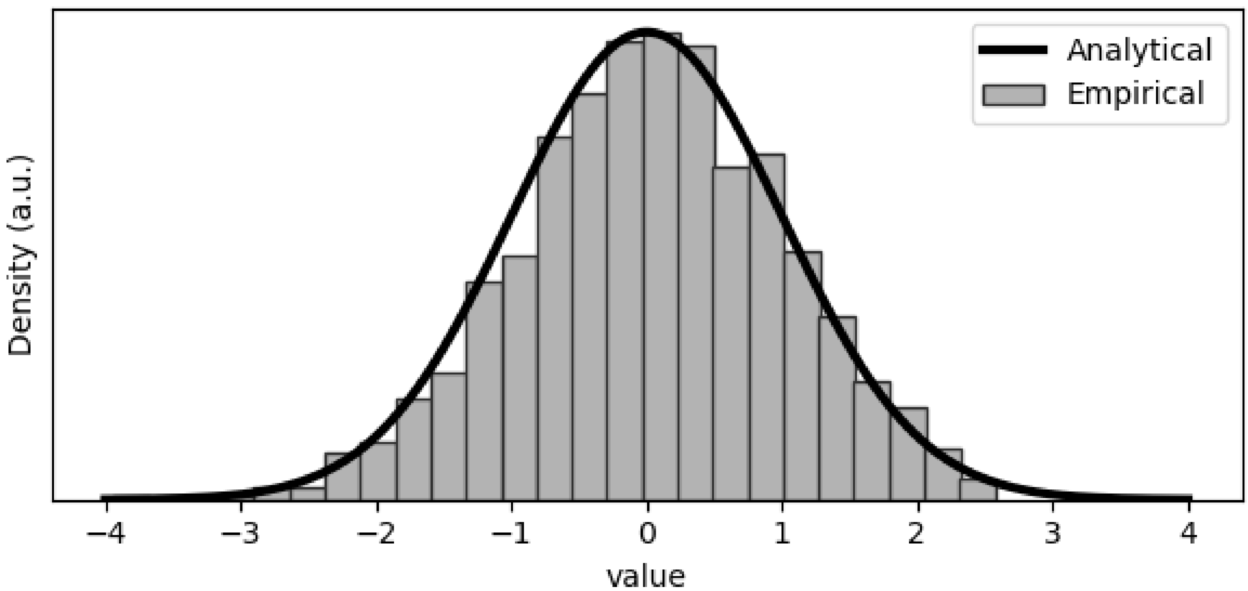
Code for chapter 16

[1]:
```
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt

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

2 Figure 16.1: Analytic vs empirical H0 distribution

[2]:
```
# x-axis grid
x = np.linspace(-4,4,1001)

# compute and normalize the analytic pdf
analytical = stats.norm.pdf(x)
analytical /= np.max(analytical)

# same for empirical
empirical = np.random.normal(loc=0,scale=1,size=len(x))
yy,xx = np.histogram(empirical,bins='fd')
yy = yy/np.max(yy)
xx = (xx[1:]+xx[:-1])/2

## draw the figure
plt.figure(figsize=(6,3))
plt.bar(xx,yy,width=.27,color=(.7,.7,.7),edgecolor=(.2,.2,.2),label='Empirical')
plt.plot(x,analytical,'k',linewidth=3,label='Analytical')

plt.xlabel('value')
plt.yticks([])
plt.ylabel('Density (a.u.)')
plt.legend()
```

```
plt.tight_layout()
#plt.savefig('permute_empVanalyH0.png')
plt.show()
```

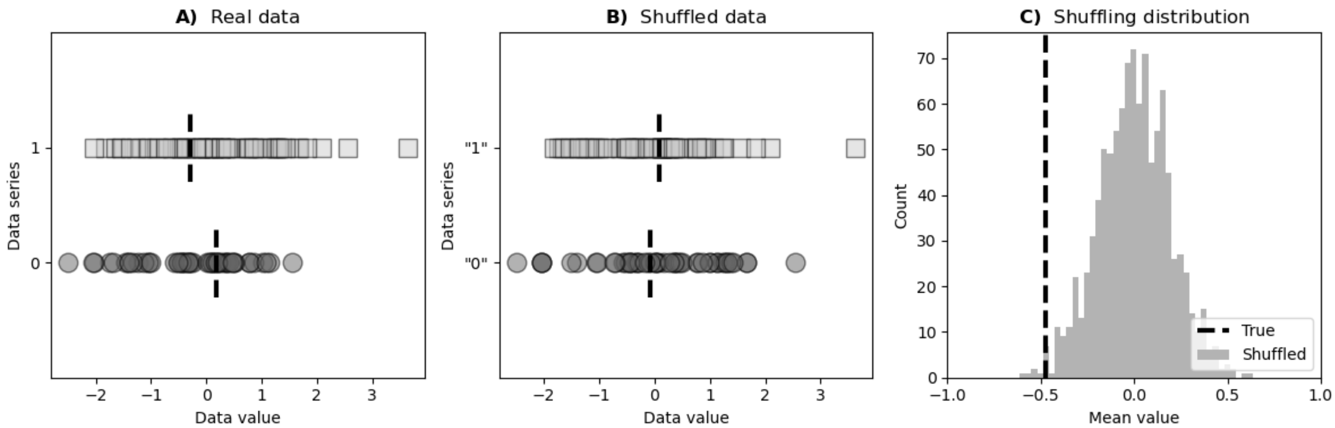


3 Figure 16.3/4: Example in comparing two sample means

[3]:
```
# number of 'trials' in each condition
n1 = 50
n2 = 70  # note the trial inbalance!

# create data
data1 = np.random.randn(n1,1)
data2 = np.random.randn(n2,1) + .3  # note the mean offset! This is set to .1␣
 ↪for Figure 16.4

# pool the data into one variable (convenient for shuffling)
alldata = np.concatenate((data1, data2))

# corresponding labels
truelabels = np.concatenate((np.ones(n1), 2*np.ones(n2)))

# compute the observed condition difference
true_conddif = np.mean(alldata[truelabels==1]) - np.mean(alldata[truelabels==2])

### creating a null-hypothesis (H0) distribution
# number of iterations for permutation testing
```

```
nIterations = 1000

# initialize output variable
permvals = np.zeros(nIterations)

for permi in range(nIterations):
  # random permutation to swap the labels
  shuflabels = np.random.permutation(truelabels)

  # mean differences in the shuffled data
  permvals[permi] = np.mean(alldata[shuflabels==1]) - np.
 ↪mean(alldata[shuflabels==2])

### visualizations
_,axs = plt.subplots(1,3,figsize=(12,4))

# show the real data and means
axs[0].plot(data1,np.zeros(n1),'ko',markersize=12,markerfacecolor=(.4,.4,.
 ↪4),alpha=.5)
axs[0].plot(data2,np.ones(n2),'ks',markersize=12,markerfacecolor=(.8,.8,.
 ↪8),alpha=.5)
axs[0].plot([np.mean(data1),np.mean(data1)],[.7,1.3],'k--',linewidth=3)
axs[0].plot([np.mean(data2),np.mean(data2)],[-.3,.3],'k--',linewidth=3)
axs[0].set(ylim=[-1,2],ylabel='Data series',yticks=[0,1],xlabel='Data value')
axs[0].set_title(r'$\bf{A})$  Real data')

# show one example shuffled data
axs[1].plot(alldata[shuflabels==1],np.
 ↪zeros(n1),'ko',markersize=12,markerfacecolor=(.4,.4,.4),alpha=.5)
axs[1].plot(alldata[shuflabels==2],np.
 ↪ones(n2),'ks',markersize=12,markerfacecolor=(.8,.8,.8),alpha=.5)
axs[1].plot([np.mean(alldata[shuflabels==1]),np.mean(alldata[shuflabels==1])],[.
 ↪7,1.3],'k--',linewidth=3)
axs[1].plot([np.mean(alldata[shuflabels==2]),np.mean(alldata[shuflabels==2])],[-.
 ↪3,.3],'k--',linewidth=3)
axs[1].set(ylim=[-1,2],ylabel='Data␣
 ↪series',yticks=[0,1],yticklabels=['"0"','"1"'],xlabel='Data value')
axs[1].set_title(r'$\bf{B})$  Shuffled data')

# distribution of shuffled means
axs[2].hist(permvals,bins=40,color=[.7,.7,.7])
axs[2].axvline(x=true_conddif,color='k',linestyle='--',linewidth=3)
axs[2].legend(['True', 'Shuffled'],loc='lower right')
axs[2].set(xlim=[-1,1],xlabel='Mean value',ylabel='Count')
axs[2].set_title(r'$\bf{C})$  Shuffling distribution')
```

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



4 Convert to p-value

[4]:
```
# based on normalized distance

zVal = (true_conddif-np.mean(permvals)) / np.std(permvals,ddof=1)
p_z  = (1-stats.norm.cdf(np.abs(zVal)))*2 # two-tailed!

print(f'Z = {zVal:.3f}, p = {p_z:.3f}')
```

```
Z = -2.492, p = 0.013
```

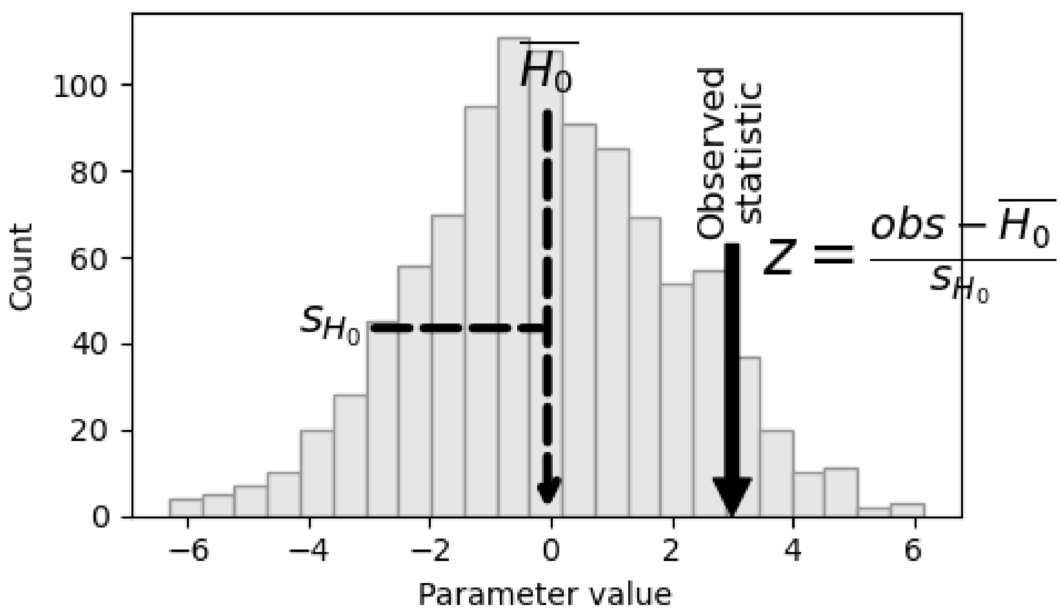
[5]:
```
# based on counts
p_c = np.sum(np.abs(permvals)>np.abs(true_conddif)) / nIterations

print(f'p_c = {p_c:.3f}')
```

```
p_c = 0.016
```

5 Figure 16.5/6: Margin figures about p-values

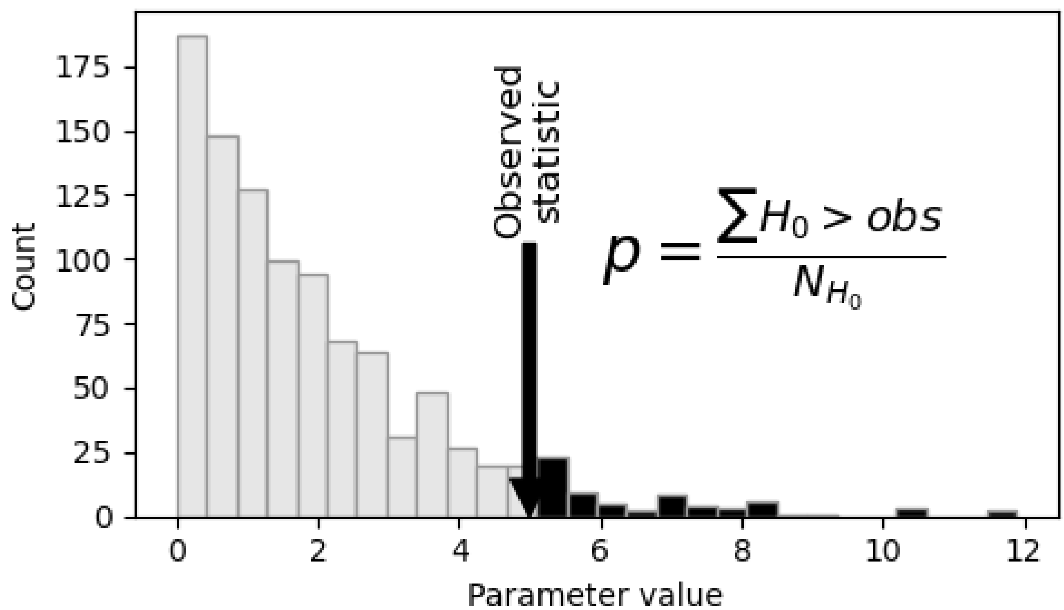
[8]:
```
# p_z is appropriate for (roughly) Gaussian H0 distributions

H0 = np.random.normal(0,2,size=1000)

plt.figure(figsize=(5,3))
h = plt.hist(H0,bins='fd',color=(.9,.9,.9),edgecolor=(.6,.6,.6))
plt.annotate('Observed\nstatistic',xytext=[3,.95*np.
 ↪max(h[0])],va='top',ha='center',rotation=90,size=12,
             xy=[3,0],arrowprops={'color':'k'})
```

```
plt.annotate(r'$\overline{H_0}$',xytext=[np.mean(H0),np.
↪max(h[0])],va='top',ha='center',rotation=0,size=15,weight='bold',
             xy=[np.mean(H0),0],arrowprops={'color':'k','arrowstyle':
↪'->','linestyle':'--','linewidth':3})
plt.annotate(r'$s_{H_0}$',xytext=[-np.std(H0,ddof=1)*2,np.
↪mean(h[0])],va='center',rotation=0,size=15,weight='bold',
             xy=[np.mean(H0),np.mean(h[0])],arrowprops={'color':'k','arrowstyle':
↪'-','linestyle':'--','linewidth':3})

plt.text(3.5,np.max(h[0])/2,r'$z = \frac{obs-\overline{H_0}}{s_{H_0}}$',size=20)
plt.xlabel('Parameter value')
plt.ylabel('Count')
plt.tight_layout()
plt.savefig('permute_pz.png')
plt.show()
```



[9]:
```
# p_c is appropriate for any shape H0 distributions

H0 = np.random.exponential(2,size=1000)

plt.figure(figsize=(5,3))
h = plt.hist(H0,bins='fd',color=(.9,.9,.9),edgecolor=(.6,.6,.6))
plt.annotate('Observed\nstatistic',xytext=[5,.95*np.
↪max(h[0])],va='top',ha='center',rotation=90,size=12,
             xy=[5,0],arrowprops={'color':'k'})

plt.text(6,np.max(h[0])/2,r'$p = \frac{\sum H_0>obs}{N_{H_0}}$',size=20)
```

```python
# paint bars black if greater than statistic
for p in h[2]:
  if p.get_x()>5: p.set_facecolor('k')

plt.xlabel('Parameter value')
plt.ylabel('Count')
plt.tight_layout()
#plt.savefig('permute_pc.png')
plt.show()
```

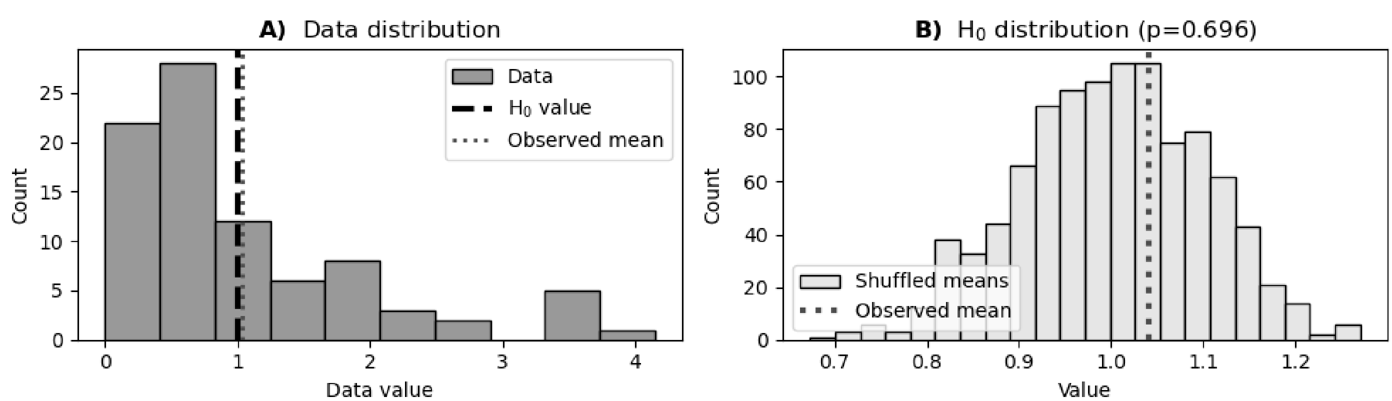


6 Figure 16.7: Permutation testing for the mean of one sample

[10]:
```python
# create non-normal data
N = 87
data = stats.gamma.rvs(1.2,size=N)
h0val = 1
sampleMean = np.mean(data)

# Note: creating the data in a separate cell lets you
# re-run the permutation testing multiple times on the same data.
```

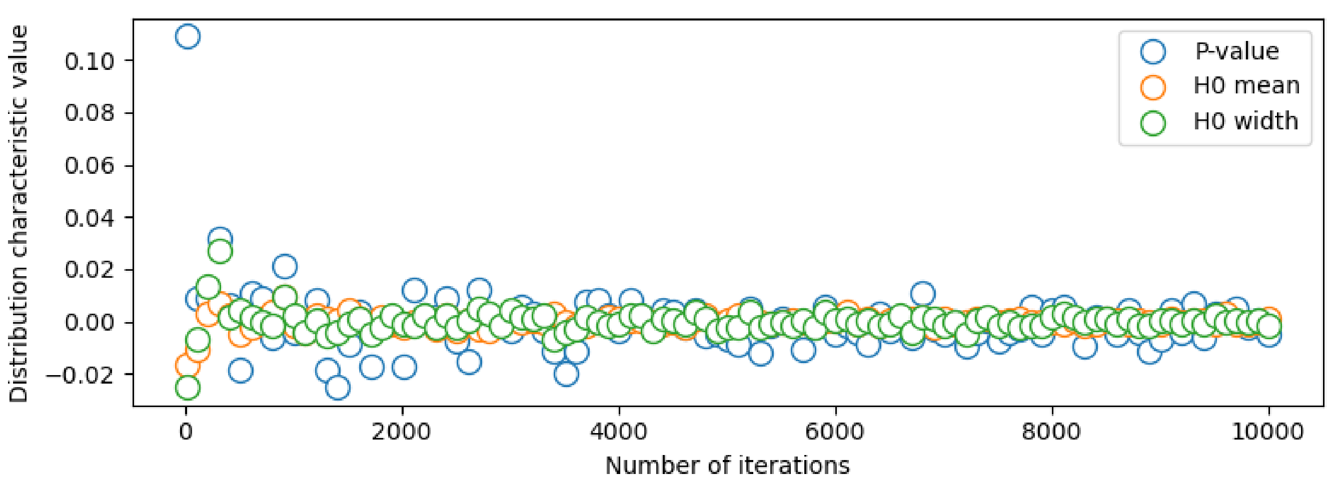
[11]:
```python
# permutation testing

data4perm = data - h0val # shift the problem such that H0=0
obsMean = np.mean(data4perm)
```

```
nPerms = 1000
permMeans = np.zeros(nPerms)

for permi in range(nPerms):

  # create a vector of +/- 1's
  randSigns = np.sign(np.random.randn(N))

  # mean of shuffled data
  permMeans[permi] = np.mean( randSigns*np.abs(data4perm) )

# compute p-value based on extreme count
pval = np.sum(np.abs(permMeans) > np.abs(obsMean)) / nPerms

# show distributions
_,axs = plt.subplots(1,2,figsize=(10,3))

axs[0].hist(data,bins='fd',color=(.6,.6,.6),edgecolor='k',label='Data')
axs[0].axvline(x=h0val,color='k',linestyle='--',linewidth=3,label=r'H$_0$ value')
axs[0].axvline(x=sampleMean,color=(.3,.3,.3),linestyle=':
↪',linewidth=2,label='Observed mean')
axs[0].legend()
axs[0].set_title(r'$\bf{A})$  Data distribution')
axs[0].set(xlabel='Data value',ylabel='Count')

# histogram of permutations (adding back h0 value for visualization)
axs[1].hist(permMeans+h0val,bins='fd',color=(.9,.9,.
↪9),edgecolor='k',label='Shuffled means')
axs[1].axvline(x=sampleMean,color=(.3,.3,.3),linestyle=':
↪',linewidth=3,label='Observed mean')
axs[1].set_title(rf'$\bf{{B}})$  H$_0$ distribution (p={pval})')
axs[1].set(xlabel='Value',ylabel='Count')
axs[1].legend(loc='lower left')

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



7 Figure 16.9: Number of iterations

[12]:
```python
numberOfIterations = np.arange(10,10011,100)

H0chars = np.zeros((len(numberOfIterations),3))

for ni,nPerms in enumerate(numberOfIterations):
  # permutation testing
  permMeans = np.zeros(nPerms)
  for permi in range(nPerms):
    permMeans[permi] = np.mean( np.sign(np.random.randn(N))*data4perm )

  # H0 distribution characteristics
  H0chars[ni,0] = np.mean(np.abs(permMeans) > np.abs(obsMean)) # p-value
  H0chars[ni,1] = np.mean(permMeans)                           # H0␣
↪distribution mean
  H0chars[ni,2] = stats.iqr(permMeans)                         # distribution␣
↪width (IQR)

# plotting
plt.figure(figsize=(8,3))
plt.plot(numberOfIterations,H0chars-np.mean(H0chars,axis=0),
         'o',markersize=10,markerfacecolor='w',linewidth=2)
plt.legend(['P-value','H0 mean','H0 width'])
plt.xlabel('Number of iterations')
plt.ylabel('Distribution characteristic value')

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



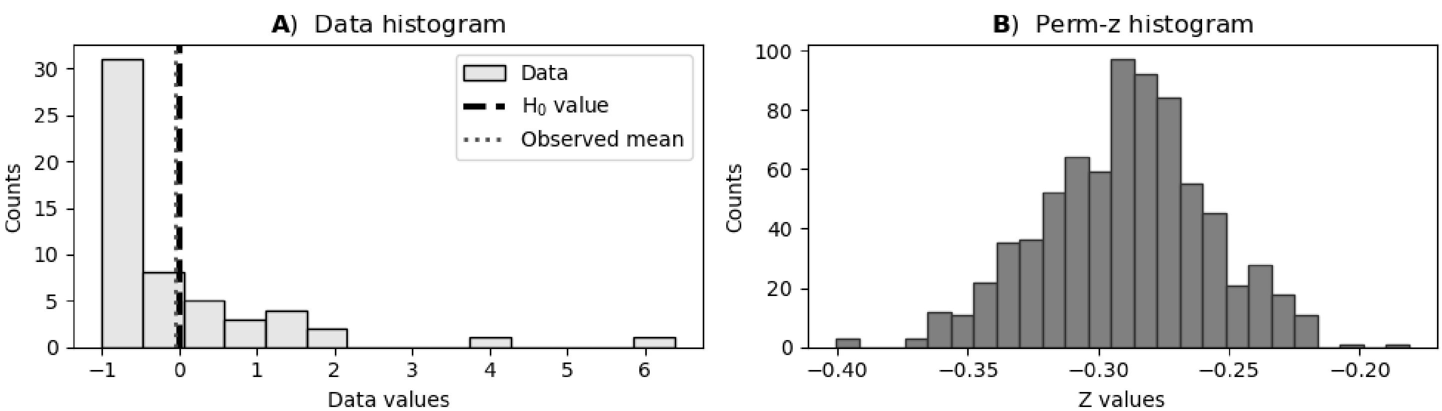
8 Exercise 1

[13]:
```
# simulation parameters
N = 55
nIterations = 1000

# the data and its mean
theData = np.random.normal(loc=0,scale=1,size=N)**2 - 1
theMean = np.mean(theData)

# one permutation test
permMeans = np.zeros(nIterations)
for permi in range(nIterations):
  # the data with random sign flips
  signFlippedData = np.sign(np.random.randn(N))*theData
  # and its mean
  permMeans[permi] = np.mean( signFlippedData )

# zscore relative to H0 distribution
zVal = (theMean-np.mean(permMeans)) / np.std(permMeans,ddof=1)
pVal = (1-stats.norm.cdf(np.abs(zVal)))*2

# print the z/p values
print(f'z = {zVal:.2f}, p = {pVal:.3f}')
```

```
z = -0.41, p = 0.681
```

[14]:
```
# simulation parameters (repeated from previous cell; feel free to modify!)
N = 55
nPermTests = 750
nIterations = 1000
theData = np.random.normal(loc=0,scale=1,size=N)**2 - 1
theMean = np.mean(theData)

# initialize output vector
zVals = np.zeros(nPermTests)

# loop over all the permutation tests
for ni in range(nPermTests):
  # permutation testing (same as above but with fewer lines of code)
  permMeans = np.zeros(nIterations)
  for permi in range(nIterations):
    permMeans[permi] = np.mean( np.sign(np.random.randn(N))*theData )
  # zscore relative to H0 distribution
  zVals[ni] = (theMean-np.mean(permMeans)) / np.std(permMeans,ddof=1)
```

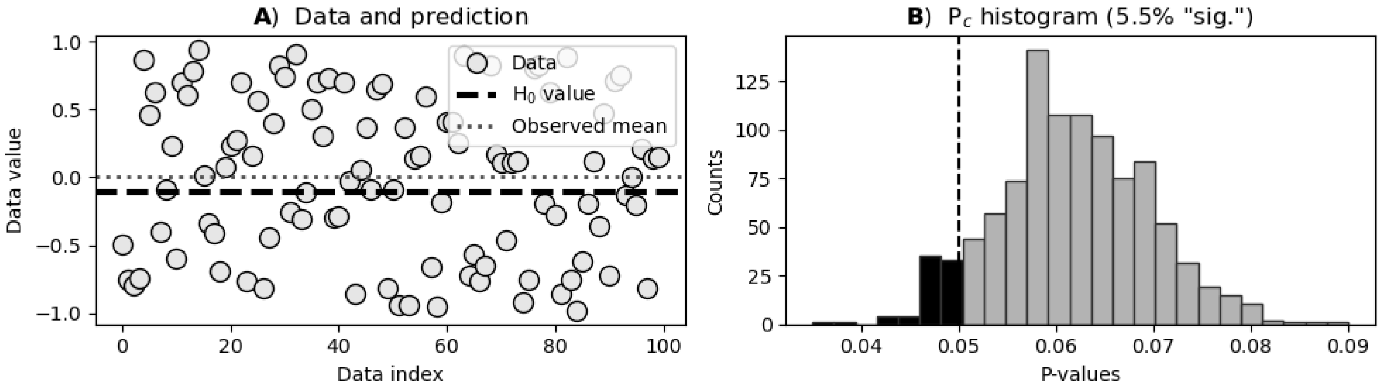
```
## plotting
_,axs = plt.subplots(1,2,figsize=(10,3))
axs[0].hist(theData,bins='fd',color=(.9,.9,.9),edgecolor='k',label='Data')
axs[0].axvline(x=0,color='k',linestyle='--',linewidth=3,label=r'H$_0$ value')
axs[0].axvline(x=theMean,color=(.3,.3,.3),linestyle=':
 ↪',linewidth=2,label='Observed mean')
axs[0].set(xlabel='Data values',ylabel='Counts')
axs[0].legend()
axs[0].set_title(r'$\bf{A}$)  Data histogram')

axs[1].hist(zVals,bins='fd',color=(.5,.5,.5),edgecolor=(.2,.2,.2))
axs[1].set(xlabel='Z values',ylabel='Counts')
axs[1].set_title(r'$\bf{B}$)  Perm-z histogram')

plt.tight_layout()
#plt.savefig('permute_ex1.png')
plt.show()
```



9 Exercise 2

[15]:
```
# create non-normal data
N = 100
data = np.random.uniform(low=-1,high=1,size=N)
data -= np.mean(data)
h0val = -.11

# other simulation parameters
nPerms = 1000
numPermTests = 1000

# permutation testing
data4perm = data - h0val
obsMean = np.mean(data4perm)
```

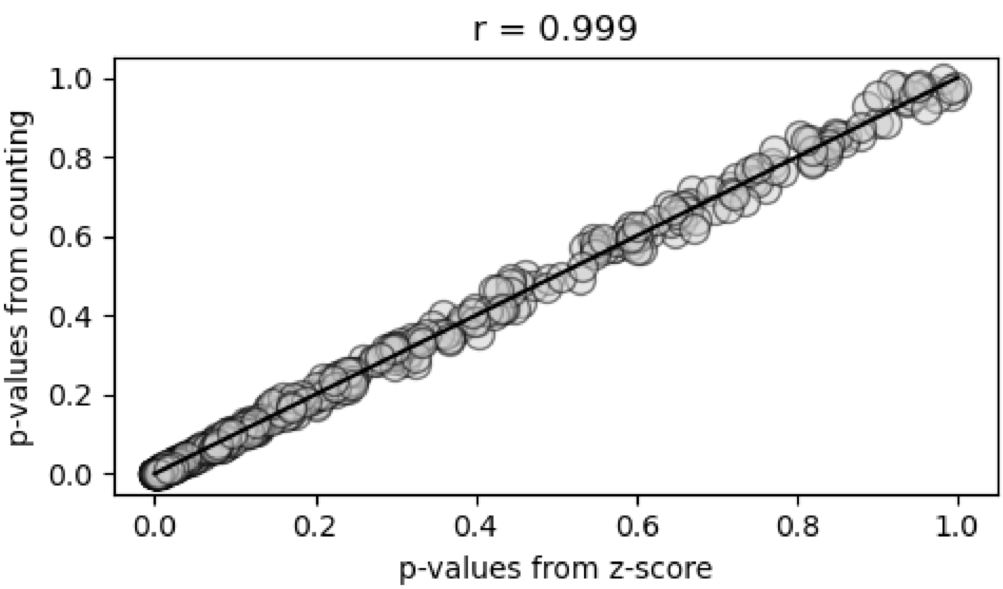
```
# initialize output variables
permMeans = np.zeros(nPerms)
pvals = np.zeros(numPermTests)

# the 'outer loop' over many permutation tests
for permRepeati in range(numPermTests):
  # permutation test (copied from previous code in this notebook)
  for permi in range(nPerms):
    randSigns = np.sign(np.random.randn(N))
    permMeans[permi] = np.mean( randSigns*data4perm )

  # compute and store the p-value
  pvals[permRepeati] = np.mean( np.abs(permMeans) > np.abs(obsMean) )

## plotting
_,axs = plt.subplots(1,2,figsize=(10,3))
axs[0].plot(data,'ko',markerfacecolor=(.9,.9,.9),markersize=10,label='Data')
axs[0].axhline(y=h0val,color='k',linestyle='--',linewidth=3,label=r'H$_0$ value')
axs[0].axhline(y=0,color=(.3,.3,.3),linestyle=':',linewidth=2,label='Observed␣
↪mean')
axs[0].set(xlabel='Data index',ylabel='Data value')
axs[0].legend(loc='upper right')
axs[0].set_title(r'$\bf{A}$)  Data and prediction')

h = axs[1].hist(pvals,bins='fd',color=(.7,.7,.7),edgecolor=(.2,.2,.2))
axs[1].set(xlabel='P-values',ylabel='Counts')
axs[1].set_title(r'$\bf{B}$)  P$_c$ histogram ' + f'({np.mean(pvals<.05)*100:.
↪1f}% "sig.")')
axs[1].axvline(x=.05,color='k',linestyle='--')

# paint bars black if "significant"
for p in h[2]:
  if p.get_x()<.05: p.set_facecolor('k')

plt.tight_layout()
#plt.savefig('permute_ex2.png')
plt.show()
```



10 Exercise 3

[16]:
```
# data parameters
n1,n2 = 50,70

# experiment parameters
nIterations = 1000 # in each permutation test
numRepeats = 541 # number of times to generate new data
# initializations
permvals = np.zeros(nIterations)
truelabels = np.concatenate((np.ones(n1), 2*np.ones(n2)))
pvals = np.zeros((numRepeats,2))

# run the experiment!
for expi in range(numRepeats):
  # create new data
  data1 = np.random.randn(n1,1)
  data2 = np.random.randn(n2,1) + .3  # note the mean offset!

  # pool the data into one variable (convenient for shuffling)
  alldata = np.concatenate((data1, data2))
  true_conddif = np.mean(alldata[truelabels==1]) - np.
↪mean(alldata[truelabels==2])
  ### creating a null-hypothesis (H0) distribution
  for permi in range(nIterations):
    shuflabels = np.random.permutation(truelabels)
    permvals[permi] = np.mean(alldata[shuflabels==1]) - np.
↪mean(alldata[shuflabels==2])

  # p_z
  zVal = (true_conddif-np.mean(permvals)) / np.std(permvals,ddof=1)
  pvals[expi,0] = (1-stats.norm.cdf(np.abs(zVal)))*2 # two-tailed!
  # p_c
  pvals[expi,1] = np.sum(np.abs(permvals)>np.abs(true_conddif)) / nIterations
```

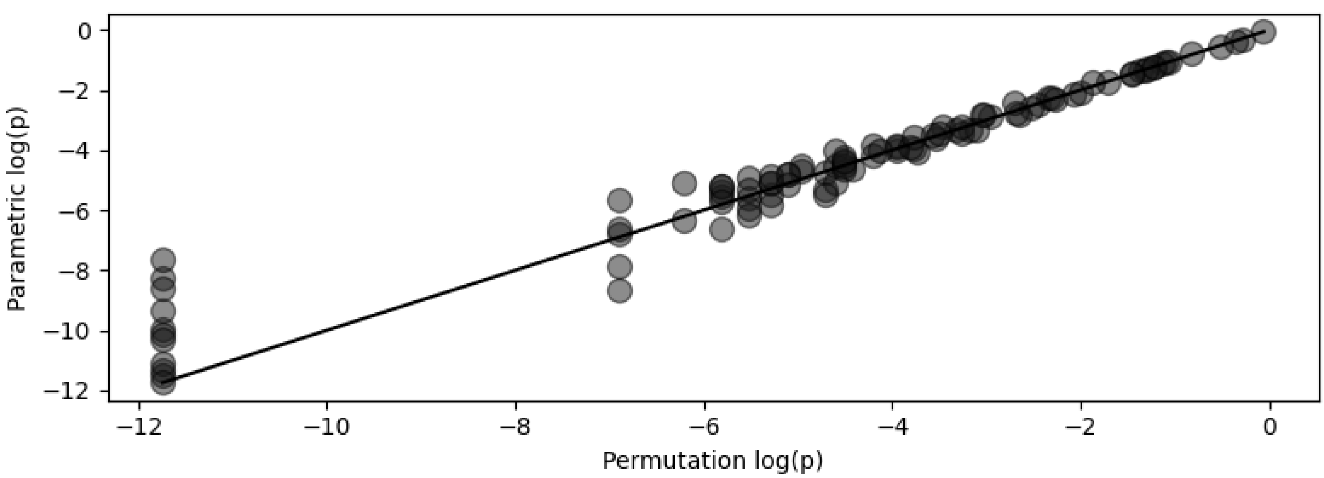
```
## and the visualization
plt.figure(figsize=(5,3))
plt.plot(pvals[:,0],pvals[:,1],'ko',markersize=10,markerfacecolor=(.8,.8,.
↪8),alpha=.5)
plt.plot([0,1],[0,1],'k')
plt.xlabel('p-values from z-score')
plt.ylabel('p-values from counting')
plt.title(f'r = {np.corrcoef(pvals.T)[0,1]:.3f}',loc='center')

plt.tight_layout()
#plt.savefig('permute_ex3.png')
plt.show()
```



11 Exercise 4

[17]:
```
nPerms = 1000
permMeans = np.zeros(nPerms)

N = 30
h0val = .5
pvals = np.zeros((100,2))
```

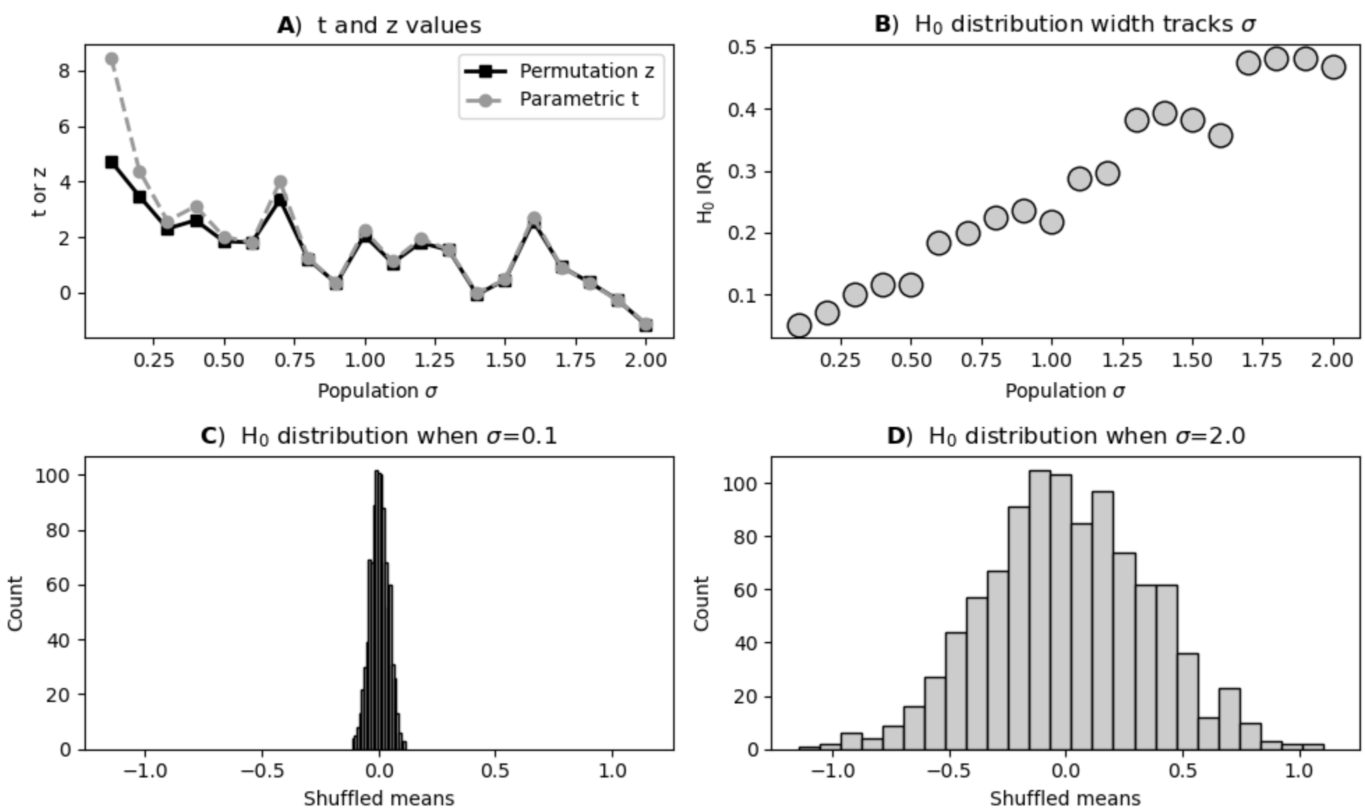
```python
# loop over the experiment iterations
for iter in range(100):
  # create the data (shifted by h0 such that H0=0)
  X = np.random.randn(N)**1 - h0val

  # permutation testing
  permMeans = np.zeros(nPerms)
  for permi in range(nPerms):
    permMeans[permi] = np.mean( np.random.choice((-1,1),N)*X )

  # p-value from permutation testing
  pvals[iter,0] = np.mean( np.abs(permMeans)>np.abs(np.mean(X)) )

  # p-value from 1-sample ttest
  pvals[iter,1] = stats.ttest_1samp(X,0).pvalue

# replace p=0 with p=min
pvals[pvals==0] = np.min(pvals[pvals>0])
pvals = np.log(pvals)

## visualization
plt.figure(figsize=(8,3))
plt.plot(pvals[:,0],pvals[:,1],'ko',
         markersize=10,markerfacecolor=(.1,.1,.1),alpha=.5)
prange = [ np.min(pvals),np.max(pvals) ] # min/max p-values for the unity line
plt.plot([prange[0],prange[1]],[prange[0],prange[1]],'k')
plt.xlabel('Permutation log(p)')
plt.ylabel('Parametric log(p)')

plt.tight_layout()
#plt.savefig('permute_ex4.png')
plt.show()
```



12 Exercise 5

[18]:
```
# simulation parameters
N = 30
nPerms = 1000
stdevs = np.linspace(.1,2,20)

# initializations
permMeans = np.zeros(nPerms)
results = np.zeros((len(stdevs),3))
H0dists = [0]*2

## run the experiment
for si,s in enumerate(stdevs):
  # create the data
  data = np.random.normal(.2,s,size=N)

  # permutation testing
  for permi in range(nPerms):
    permMeans[permi] = np.mean( np.random.choice((-1,1),N)*data )

  # store the t/z values
  results[si,0] = (np.mean(data)-np.mean(permMeans)) / np.std(permMeans,ddof=1)
  results[si,1] = stats.iqr(permMeans)
  results[si,2] = stats.ttest_1samp(data,0).statistic

  # store the extremiest H0 distributions
  if si==0:
    H0dists[0] = permMeans+0 # +0 makes a copy
  elif si==(len(stdevs)-1):
    H0dists[1] = permMeans

## plotting!
_,axs = plt.subplots(2,2,figsize=(10,6))

axs[0,0].plot(stdevs,results[:,0],'ks-',linewidth=2,label='Permutation z')
axs[0,0].plot(stdevs,results[:,2],'--o',color=(.6,.6,.
↪6),linewidth=2,label='Parametric t')
axs[0,0].legend()
axs[0,0].set(xlabel=r'Population $\sigma$',ylabel='t or z')
axs[0,0].set_title(r'$\bf{A}$)  t and z values')

axs[0,1].plot(stdevs,results[:,1],'ko',markerfacecolor=(.8,.8,.8),markersize=12)
axs[0,1].set(xlabel=r'Population $\sigma$',ylabel=r'H$_0$ IQR')
axs[0,1].set_title(r'$\bf{B}$)  H$_0$ distribution width tracks $\sigma$')
```

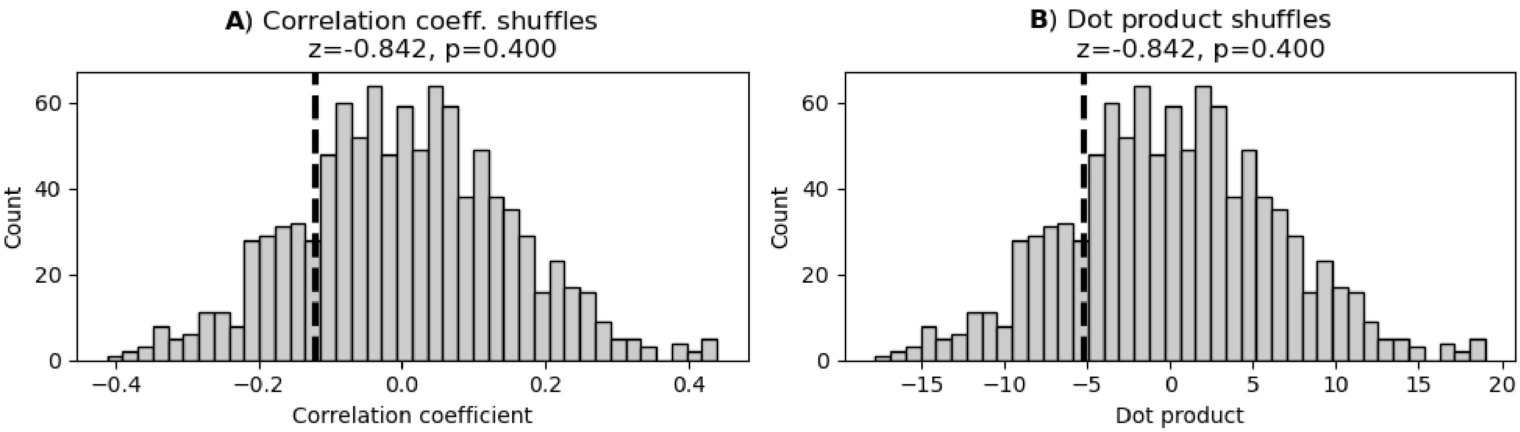
```
# histograms of the H0 distributions
axs[1,0].hist(H0dists[0],bins='fd',color=(.8,.8,.8),edgecolor='k')
axs[1,0].set_title(r'$\bf{C}$)  H$_0$ distribution when $\sigma$=' +␣
 ↪str(stdevs[0]))
axs[1,1].hist(H0dists[1],bins='fd',color=(.8,.8,.8),edgecolor='k')
axs[1,1].set_title(r'$\bf{D}$)  H$_0$ distribution when $\sigma$=' +␣
 ↪str(stdevs[-1]))

for a in axs[1,:]:
  a.set(xlabel='Shuffled means',ylabel='Count',xlim=np.array([-1.1,1.1])*np.
 ↪max(np.abs(H0dists[1])))

plt.tight_layout()
#plt.savefig('permute_ex5.png')
plt.show()
```



13 Exercise 6

[20]:
```
# simulation parameters
n = 50
r = .2
x = np.random.randn(n)
y = np.random.randn(n)
y = x*r + y*np.sqrt(1-r**2)
```

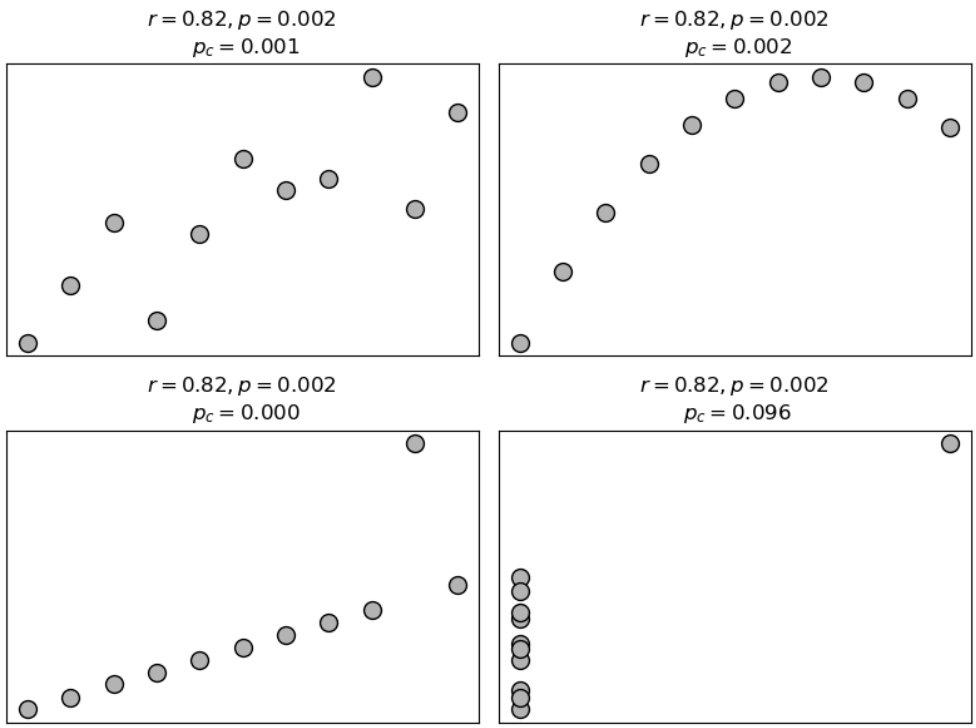
```python
# mean-center
x -= np.mean(x)
y -= np.mean(y)

# observed correlation and dot product
r,p = stats.pearsonr(x,y)
dp = np.dot(x,y)

# initialize output matrix
permRes = np.zeros((1000,2))

# permutation testing
for permi in range(len(permRes)):
  # shuffle y
  yshuf = y[np.random.permutation(n)]

  # pearson correlation
  permRes[permi,0] = stats.pearsonr(x,yshuf)[0]

  # mean-centered dot product
  permRes[permi,1] = np.dot(x,yshuf)

# z and p values
z_r = ( r-np.mean(permRes[:,0])) / np.std(permRes[:,0],ddof=1)
z_d = (dp-np.mean(permRes[:,1])) / np.std(permRes[:,1],ddof=1)

p_r = (1-stats.norm.cdf(np.abs(z_r)))*2
p_d = (1-stats.norm.cdf(np.abs(z_d)))*2

_,axs = plt.subplots(1,2,figsize=(10,3))
axs[0].hist(permRes[:,0],bins=40,color=(.8,.8,.8),edgecolor='k')
axs[0].axvline(x=r,linestyle='--',color='k',linewidth=3)
axs[0].set(xlabel='Correlation coefficient',ylabel='Count')
axs[0].set_title(r'$\bf{A}$)' + ' Correlation coeff. shuffles' + '\n' + f'    ␣
↪z={z_r:.3f}, p={p_r:.3f}')

axs[1].hist(permRes[:,1],bins=40,color=(.8,.8,.8),edgecolor='k')
axs[1].axvline(x=dp,linestyle='--',color='k',linewidth=3)
axs[1].set(xlabel='Dot product',ylabel='Count')
axs[1].set_title(r'$\bf{B}$)' + ' Dot product shuffles' + '\n' + f'     z={z_d:.
↪3f}, p={p_d:.3f}')

plt.tight_layout()
#plt.savefig('permute_ex6.png')
plt.show()
```



14 Exercise 7

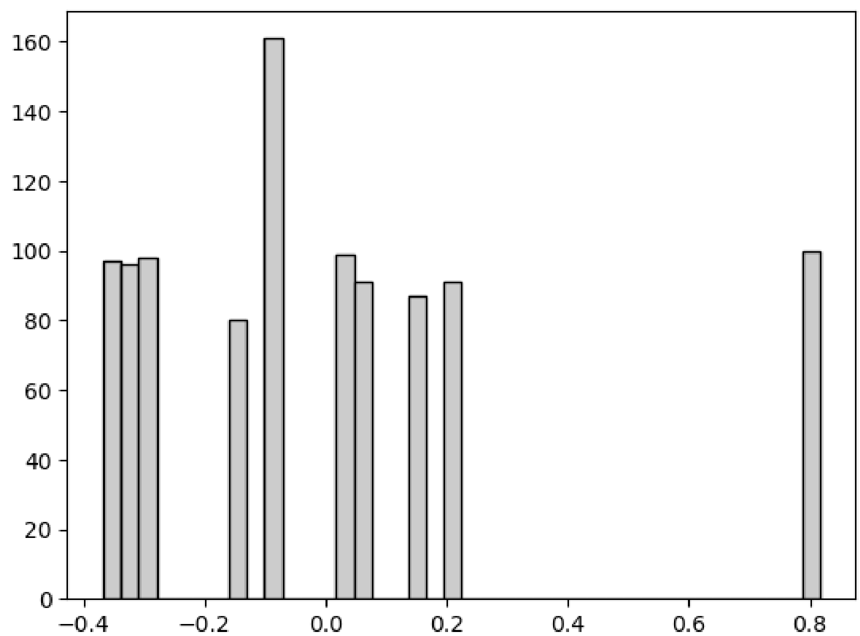
[21]:
```
anscombe = np.array([
     # series 1     series 2      series 3       series 4
    [10,  8.04,    10,  9.14,    10,  7.46,      8,  6.58, ],
    [ 8,  6.95,     8,  8.14,     8,  6.77,      8,  5.76, ],
    [13,  7.58,    13,  8.76,    13, 12.74,      8,  7.71, ],
    [ 9,  8.81,     9,  8.77,     9,  7.11,      8,  8.84, ],
    [11,  8.33,    11,  9.26,    11,  7.81,      8,  8.47, ],
    [14,  9.96,    14,  8.10,    14,  8.84,      8,  7.04, ],
    [ 6,  7.24,     6,  6.13,     6,  6.08,      8,  5.25, ],
    [ 4,  4.26,     4,  3.10,     4,  5.39,      8,  5.56, ],
    [12, 10.84,    12,  9.13,    12,  8.15,      8,  7.91, ],
    [ 7,  4.82,     7,  7.26,     7,  6.42,      8,  6.89, ],
    [ 5,  5.68,     5,  4.74,     5,  5.73,     19, 12.50, ]
    ])

nSamples = anscombe.shape[0]
permRs = np.zeros(1000)

# plot data and correlations
fig,ax = plt.subplots(2,2,figsize=(8,6))
ax = ax.ravel()

for i in range(4):
  # convenience
  x = anscombe[:,i*2]
  y = anscombe[:,i*2+1]

  # plot the points
  ax[i].plot(x,y,'ko',markersize=10,markerfacecolor=(.7,.7,.7))

  # compute the correlation and parametric p-value
  r,pp = stats.pearsonr(x,y)
```

```
  # permutation testing
  for permi in range(len(permRs)):
    permRs[permi] = stats.pearsonr(x,y[np.random.permutation(nSamples)])[0]
  pc = np.mean(np.abs(permRs)>=np.abs(r))

  # update the axis
  ax[i].set(xticks=[],yticks=[])
  ax[i].set_title(f'$r={r:.2f}, p={pp:.3f}$ \n $p_c={pc:.3f}$',loc='center')

plt.tight_layout()
#plt.savefig('permute_ex7.png')
plt.show()
```



[22]:
```
# btw, interesting to see that the possible permuted correlation values is␣
↪limited
# due to the small sample size and limited data values... not an ideal situation␣
↪for
# parametric or non-parametric analyses.
plt.hist(permRs,bins=40,color=(.8,.8,.8),edgecolor='k');

print(f'{len(permRs)} random permutations and only {len(np.unique(permRs))}␣
↪unique values!')
```

```
1000 random permutations and only 44 unique values!
```


160
140
120
100
80
60
40
20
0
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8