stats_ch11_ttest

August 5, 2024

1 Modern statistics: Intuition, Math, Python, R

1.1 Mike X Cohen (sincxpress.com)

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

Code for chapter 11 (t-tests)

---

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.

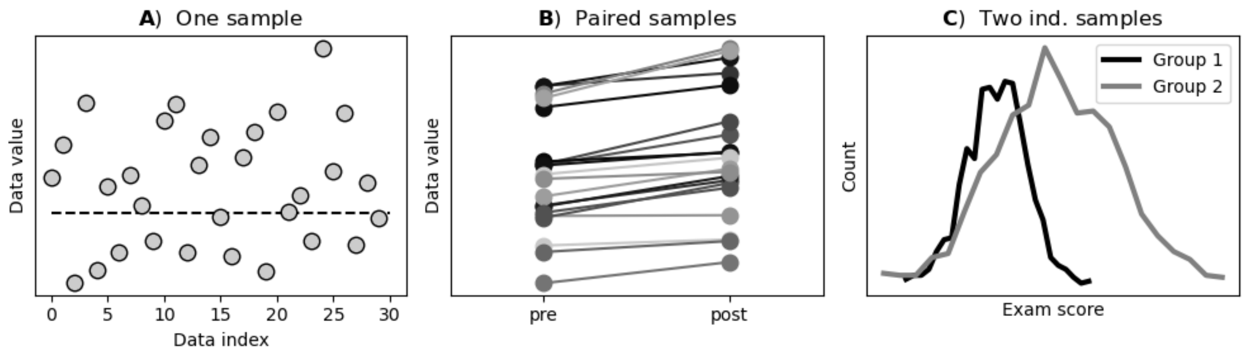
This code was written in google-colab. The notebook may require some modifications if you use a different IDE.

[1]:
```python
# import libraries and define global settings
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt

# pandas/seaborn for ex12+
import pandas as pd
import seaborn as sns

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

3 Figure 11.1: Goals of the t-test

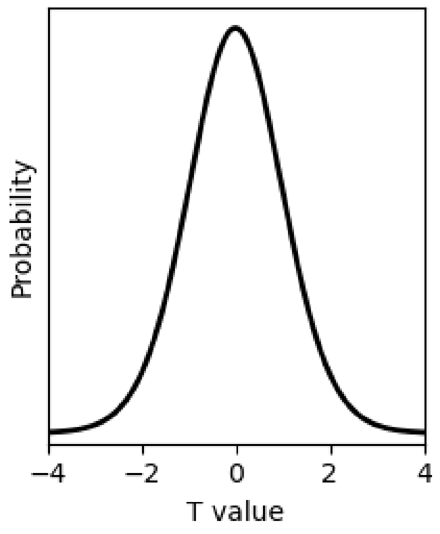
[2]:
```python
_,axs = plt.subplots(1,3,figsize=(10,3))

## panel A: one-sample t-test
```

```
data = np.random.normal(loc=.5,size=30)
axs[0].plot(data,'ko',markersize=9,markerfacecolor=(.8,.8,.8))
axs[0].plot([0,len(data)],[0,0],'k--',zorder=-10)
axs[0].set(xlabel='Data index',ylabel='Data value',yticks=[],
           title=r'$\bf{A}$)  One sample')

## panel B: paired-samples t-test
N = 20
data1 = np.random.normal(size=N)
data2 = data1 + .5 + np.random.randn(N)*.4
for x,y in zip(data1,data2):
  # pick a random color
  c = np.random.uniform(low=0,high=.8)

  # plot it
  axs[1].
 ↪plot([0,1],[x,y],'o-',markersize=9,markerfacecolor=(c,c,c),color=(c,c,c))

# plot adjustments
axs[1].set(xlim=[-.5,1.5],xticks=[0,1],xticklabels=['pre','post'],
           yticks=[],ylabel='Data value',title=r'$\bf{B}$)  Paired samples')

## panel C: two-samples t-test
for i in range(2):
  # create the data
  data = np.random.normal(loc=i,scale=(i+1)/2,size=1000)

  # take their histogram
  yy,xx = np.histogram(data,bins='fd')
  xx = (xx[1:]+xx[:-1])/2

  # plot
  c = i/2
  axs[2].plot(xx,yy,linewidth=3,color=(c,c,c))

# plot adjustments
axs[2].set(xlabel='Exam␣
 ↪score',ylabel='Count',xticks=[],yticks=[],title=r'$\bf{C}$)  Two ind. samples')
axs[2].legend(['Group 1','Group 2'],loc='upper right',fontsize=10)

# final figure adjustments
plt.tight_layout()
#plt.savefig('ttest_ttestGoals.png')
plt.show()
```



4 Figure 11.2: A t-pdf

[5]:
```python
t = np.linspace(-4,4,573)

# a pdf with df=20
tpdf = stats.t.pdf(t,20)

plt.figure(figsize=(2.5,3))
plt.plot(t,tpdf,'k',linewidth=2)
plt.xlabel('T value')
plt.ylabel('Probability')
plt.yticks([])
plt.ylim([-.01,np.max(tpdf)*1.05])
plt.xlim(t[[0,-1]])

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



5 Computing p-values for one-tailed and two-tailed tests

[6]:
```
tval = 2.1
df = 13

pvalL = stats.t.cdf(-tval,df)
pvalR = stats.t.sf(tval,df)#1-stats.t.cdf(tval,df)
pval2 = pvalR+pvalL

print(f'One-tailed p-value on the left:  {pvalL}')
print(f'One-tailed p-value on the right: {pvalR}')
print(' ')
print(f'Two-tailed p-value as the sum:   {pvalL+pvalR}')
print(f'Two-tailed p-value by doubling:  {2*pvalL}')
```

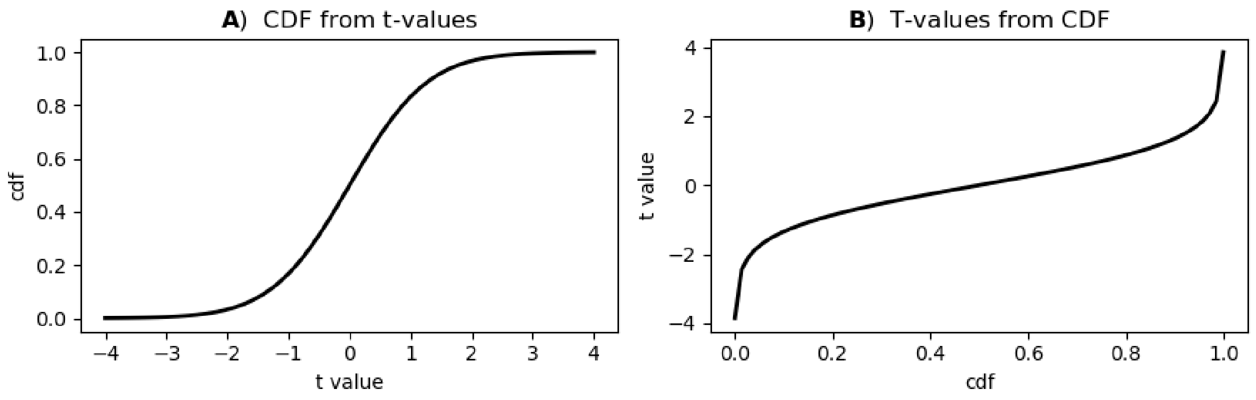
```
One-tailed p-value on the left:  0.027906302135628887
One-tailed p-value on the right: 0.027906302135628887

Two-tailed p-value as the sum:   0.055812604271257775
Two-tailed p-value by doubling:  0.055812604271257775
```

[7]:
```
# 1-cdf vs survival function:
pvalC = 1-stats.t.cdf(tval,df)
pvalS = stats.t.sf(tval,df) # sf = survival function

print(f'P-value from 1-cdf: {pvalC}')
print(f'P-value from s.f.:  {pvalS}')
print(f'Difference:         {pvalC-pvalS}')

# Conclusion: The difference for this particular t-value is at machine precision.
#  Still, there's no harm in being slightly more accurate, so you can use sf␣
 ↪instead of 1-cdf.
```

```
P-value from 1-cdf: 0.027906302135628946
P-value from s.f.:  0.027906302135628887
Difference:         5.898059818321144e-17
```

6 Figure 11.3: T-values from p-values

[9]:
```
# parameters
t = np.linspace(-4,4,75)
df = 13

# cdf based on t-values
cdf = stats.t.cdf(t,df)

# t-values based on cdf
pvals = np.linspace(.001,.999,73)
tVals = -stats.t.isf(pvals,df) # sf is 1-cdf, and isf is the inverse of the sf.␣
 ↪so 1-isf is inv(cdf)

# note: the above line is the same as below, but isf has slightly higher accuracy
#tVals = stats.t.ppf(pvals,df) # ppf = Percent point function, this is the␣
 ↪inverse of the cdf

# visualize
_,axs = plt.subplots(1,2,figsize=(9,3))
axs[0].plot(t,cdf,'k',linewidth=2)
axs[0].set(xlabel='t value',ylabel='cdf',title=r'$\bf{A}$)  CDF from t-values')

axs[1].plot(pvals,tVals,'k',linewidth=2)
axs[1].set(ylabel='t value',xlabel='cdf',title=r'$\bf{B}$)  T-values from CDF')

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



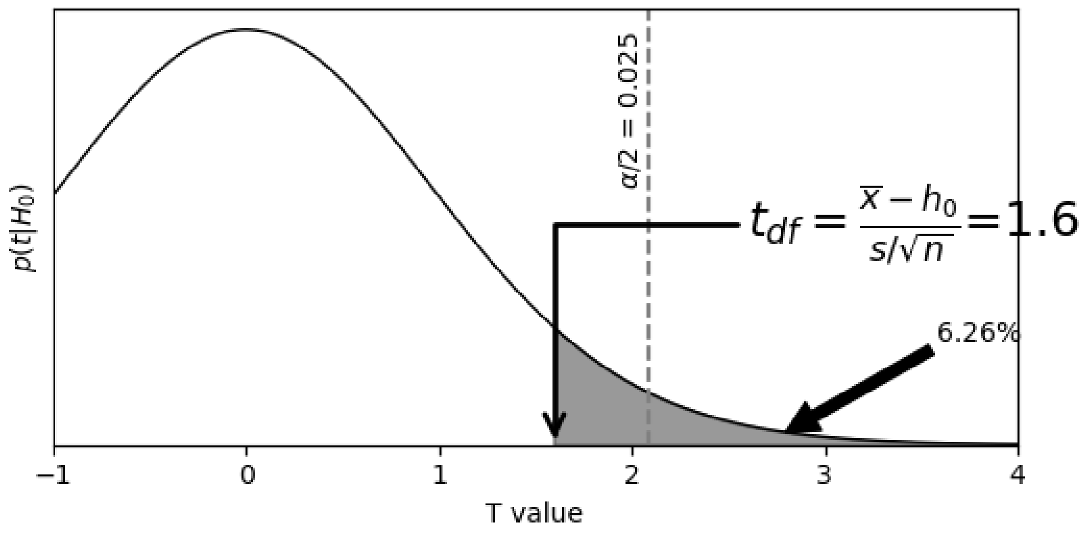
```
[10]: # example usage to get the t-value associated with p=.05 and df=13
pval = .05
tFromP_L  = -stats.t.isf(  pval,df) # negative of the opposite-cdf for the left␣
 ↪tail
tFromP_R1 = -stats.t.isf(1-pval,df) # negative of the opposite-cdf for the right␣
 ↪tail
tFromP_R2 =  stats.t.isf(  pval,df) # opposite-cdf for the left tail

print(f'Variable tFromP_L:  {tFromP_L:.3f}')
print(f'Variable tFromP_R1: {tFromP_R1:.3f}')
print(f'Variable tFromP_R2: {tFromP_R2:.3f}')
```

```
Variable tFromP_L:  -1.771
Variable tFromP_R1: 1.771
Variable tFromP_R2: 1.771
```

7 Figure 11.4: Example t-value

```
[12]: # empirical t-value and df
tval = 1.6
df   = 20
alpha = .05

# redefine the t-values and corresponding pdf
t = np.linspace(-4,4,573)
tpdf = stats.t.pdf(t,20)

# its associated p-value (but this is one-tailed for visualization; see text and␣
 ↪next cell!)
pval = 1-stats.t.cdf(tval,df)

# critical t-value for alpha
tCrit = stats.t.isf(alpha/2,df) # /2 for two-tailed!
pHalf = np.max(tpdf)/2 # 1/2 max. (vertical) p(t), used for plotting

plt.figure(figsize=(6,3))

# plot the t distribution
plt.plot(t,tpdf,'k',linewidth=1,label=r'$t_{20}$-pdf under H$_0$')

# plot the dashed line for the critical t-value
plt.axvline(tCrit,linestyle='--',color='gray')
plt.text(tCrit-.02,pHalf*2,r'$\alpha/2$ = %g'%(alpha/
 ↪2),rotation=90,va='top',ha='right')

# arrow and formula for the empirical t-value
```

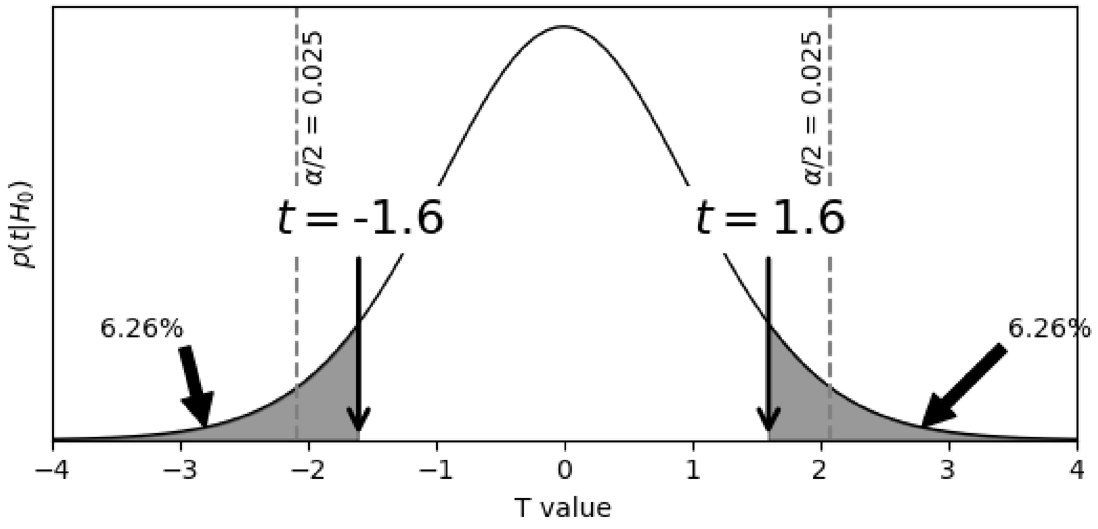
```
plt.gca().annotate(r'$t_{df}=\frac{\overline{x}-h_0}{s/
 ↪\sqrt{n}}$=%g'%tval,xytext=(tval+1,pHalf),
                xy=(tval,0), xycoords='data',size=18,
                arrowprops=dict(arrowstyle='->', color='k',linewidth=2,
                                                     ␣
 ↪connectionstyle='angle,angleA=0,angleB=-90,rad=0'))

# shaded area to the right of the empirical t-value
tidx = np.argmin(np.abs(t-tval))
plt.gca().fill_between(t[tidx:],tpdf[tidx:],color='k',alpha=.4)

# and its annotation
tidx = np.argmin(np.abs(t-(tval+t[-1])/2))
plt.gca().annotate(f'{100*pval:.2f}%',xy=(t[tidx],tpdf[tidx]),
            xytext=(t[tidx]+1,pHalf/2),ha='center',arrowprops={'color':'k'})

# some final adjustments
plt.xlabel('T value')
plt.ylabel(r'$p(t|H_0)$')
plt.yticks([])
plt.ylim([-.01,np.max(tpdf)*1.05])
plt.xlim([-1,t[-1]])
plt.ylim([0,pHalf*2.1])

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



8 Figure 11.5: Completion of the previous figure to show both tails

[14]:
```python
plt.figure(figsize=(6,3))

# plot the t distribution
plt.plot(t,tpdf,'k',linewidth=1,label=r'$t_{20}$-pdf under H$_0$')

# plot the dashed line for the critical t-value on the right side
plt.axvline(tCrit,linestyle='--',color='gray')
plt.text(tCrit-.02,pHalf*2,r'$\alpha/2$ = %g'%(alpha/
 ↪2),rotation=90,va='top',ha='right')

# and again for the left side
plt.axvline(-tCrit,linestyle='--',color='gray')
plt.text(-tCrit+.028,pHalf*2,r'$\alpha/2$ = %g'%(alpha/
 ↪2),rotation=90,va='top',ha='left')

# arrow and formula for the empirical t-value
plt.gca().annotate(r'$t=$%g'%tval,xytext=(tval,pHalf),
                xy=(tval,0),␣
 ↪xycoords='data',size=18,ha='center',bbox=dict(fc='w',edgecolor='none'),
                arrowprops=dict(arrowstyle='->', color='k',linewidth=2))

# repeat on the left
plt.gca().annotate(r'$t=$-%g'%tval,xytext=(-tval,pHalf),
                xy=(-tval,0),␣
 ↪xycoords='data',size=18,ha='center',bbox=dict(fc='w',edgecolor='none'),
                arrowprops=dict(arrowstyle='->', color='k',linewidth=2))

# shaded area to the right of the empirical t-value
tidx = np.argmin(np.abs(t-tval))
plt.gca().fill_between(t[tidx:],tpdf[tidx:],color='k',alpha=.4)
tidx = np.argmin(np.abs(t--tval))
plt.gca().fill_between(t[:tidx],tpdf[:tidx],color='k',alpha=.4)

# and its annotation for the right side
tidx = np.argmin(np.abs(t-(tval+t[-1])/2))
plt.gca().annotate(f'{100*pval:.
 ↪2f}%',xy=(t[tidx],tpdf[tidx]),xytext=(t[tidx]+1,pHalf/
 ↪2),ha='center',arrowprops={'color':'k'})

# now for the left side
tidx = np.argmin(np.abs(t-(-tval+t[0])/2))
plt.gca().annotate(f'{100*pval:.2f}%',xy=(t[tidx],tpdf[tidx]),xytext=(t[tidx]-.
 ↪5,pHalf/2),ha='center',arrowprops={'color':'k'})

# some final adjustments
```

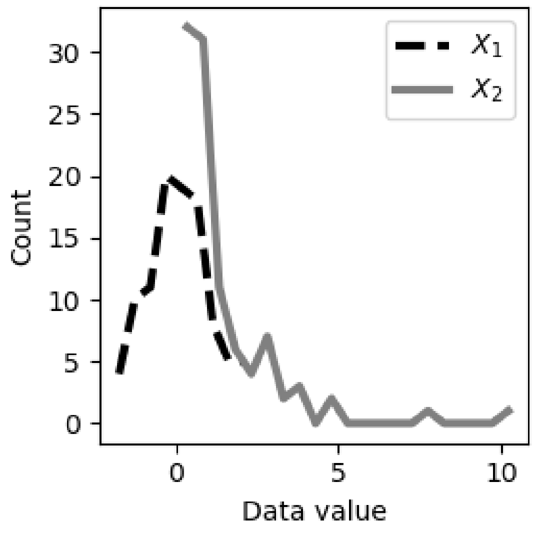
```
plt.xlabel('T value')
plt.ylabel(r'$p(t|H_0)$')
plt.yticks([])
plt.ylim([-.01,np.max(tpdf)*1.05])
plt.xlim(t[[0,-1]])
plt.ylim([0,pHalf*2.1])

plt.tight_layout()
#plt.savefig('ttest_tEmpWEq2.png')
plt.show()
```



9 Figure 11.6: Testing for normality

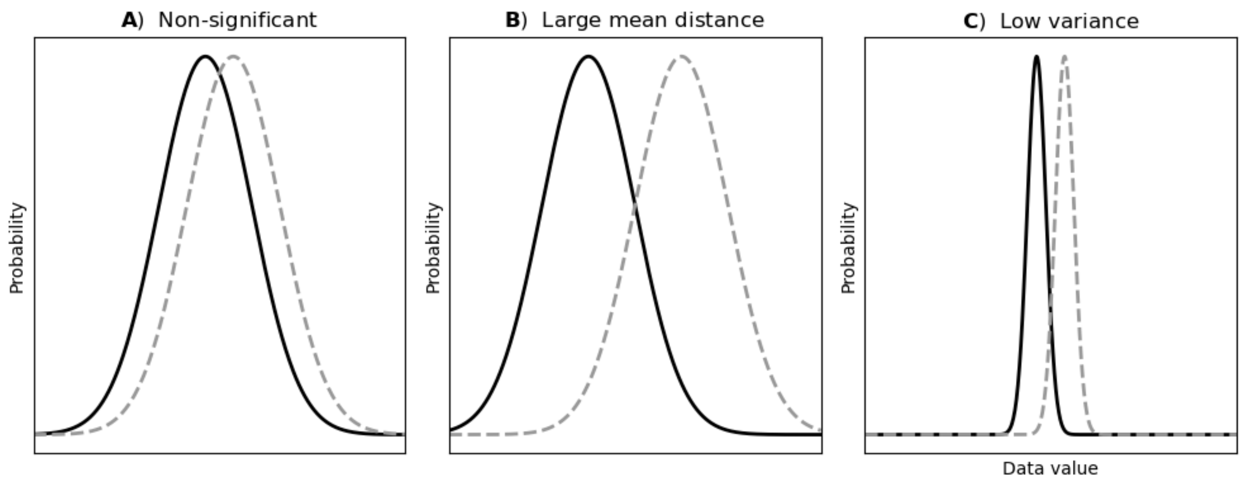
[17]:
```
# the data
data1 = np.random.randn(100)
data2 = np.exp( np.random.randn(100) )

# omnibus test
Otest1 = stats.normaltest(data1)
Otest2 = stats.normaltest(data2)

# Shapiro's test
Stest1 = stats.shapiro(data1)
Stest2 = stats.shapiro(data2)

# report the results
print(f'Omnibus test in X1 (H0=normal): p={Otest1.pvalue:.3f}')
print(f'Omnibus test in X2 (H0=normal): p={Otest2.pvalue:.3f}')
print('')
print(f'Shapiro test in X1 (H0=normal): p={Stest1.pvalue:.3f}')
print(f'Shapiro test in X2 (H0=normal): p={Stest2.pvalue:.3f}')

# show the histograms
yy1,xx1 = np.histogram(data1,bins='fd')
xx1 = (xx1[1:]+xx1[:-1])/2
yy2,xx2 = np.histogram(data2,bins='fd')
xx2 = (xx2[1:]+xx2[:-1])/2

# plotting
plt.figure(figsize=(3,3))
plt.plot(xx1,yy1,'k--',linewidth=3,label=r'$X_1$')
plt.plot(xx2,yy2,linewidth=3,color=(.5,.5,.5),label=r'$X_2$')
plt.gca().set(xlabel='Data value',ylabel='Count')
plt.legend()

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

```
Omnibus test in X1 (H0=normal): p=0.539
Omnibus test in X2 (H0=normal): p=0.000

Shapiro test in X1 (H0=normal): p=0.499
Shapiro test in X2 (H0=normal): p=0.000
```



10 Figure 11.7: Increasing the t-value

[18]:
```python
x = np.linspace(-4,4,501)

_,axs = plt.subplots(1,3,figsize=(10,4))
## panel A: probably not significant
g1 = stats.norm.pdf(x,loc=-.3,scale=1)
g2 = stats.norm.pdf(x,loc= .3,scale=1)

axs[0].plot(x,g1,color='k',linewidth=2)
axs[0].plot(x,g2,'--',color=(.6,.6,.6),linewidth=2)
axs[0].set(xticks=[],xlim=x[[0,-1]],yticks=[],ylabel='Probability',
           title=r'$\bf{A}$)  Non-significant')
## panel B: significant by larger mean difference
g1 = stats.norm.pdf(x,loc=-1,scale=1)
g2 = stats.norm.pdf(x,loc= 1,scale=1)

axs[1].plot(x,g1,color='k',linewidth=2)
axs[1].plot(x,g2,'--',color=(.6,.6,.6),linewidth=2)
axs[1].set(xticks=[],xlim=x[[0,-1]],yticks=[],ylabel='Probability',
           title=r'$\bf{B}$)  Large mean distance')
## panel C: significant by reduced variance
g1 = stats.norm.pdf(x,loc=-.3,scale=.2)
g2 = stats.norm.pdf(x,loc= .3,scale=.2)

axs[2].plot(x,g1,color='k',linewidth=2)
axs[2].plot(x,g2,'--',color=(.6,.6,.6),linewidth=2)
axs[2].set(xticks=[],xlim=x[[0,-1]],yticks=[],xlabel='Data␣
 ↪value',ylabel='Probability',
           title=r'$\bf{C}$)  Low variance')
plt.tight_layout()
#plt.savefig('ttest_sigMecs.png')
plt.show()
```



11 One-sample t-test

```
[19]: # given data
X = np.array([80, 85, 90, 70, 75, 72, 88, 77, 82, 65, 79, 81, 74, 86, 68])
h0 = 75

# descriptives
meanX = np.mean(X)
stdX  = np.std(X,ddof=1)
ssize = len(X)

# t-value
tval = (meanX-h0) / (stdX/np.sqrt(ssize))

# p-value
pval = 1-stats.t.cdf(tval,ssize-1)
pval *= 2 # two-tailed!

# print everything out!
print(f'Sample mean: {meanX:.2f}')
print(f'Sample std:  {stdX:.2f}')
print(f'Sample size: {ssize}')
print('')
print(f'T-value: {tval:.3f}')
print(f'p-value: {pval:.3f}')
```

```
Sample mean: 78.13
Sample std:  7.47
Sample size: 15

T-value: 1.624
p-value: 0.127
```

```
[20]: # Repeat using the stats libary:
ttest = stats.ttest_1samp(X,h0)

# the output variable is its own type
print( type(ttest) )

# which contains three elements:
print('')
print(ttest)

# let's print the results
print('')
print('Results from stats.ttest_1samp:')
print(f't({ttest.df})={ttest.statistic:.3f}, p<{ttest.pvalue:.3f}')
```

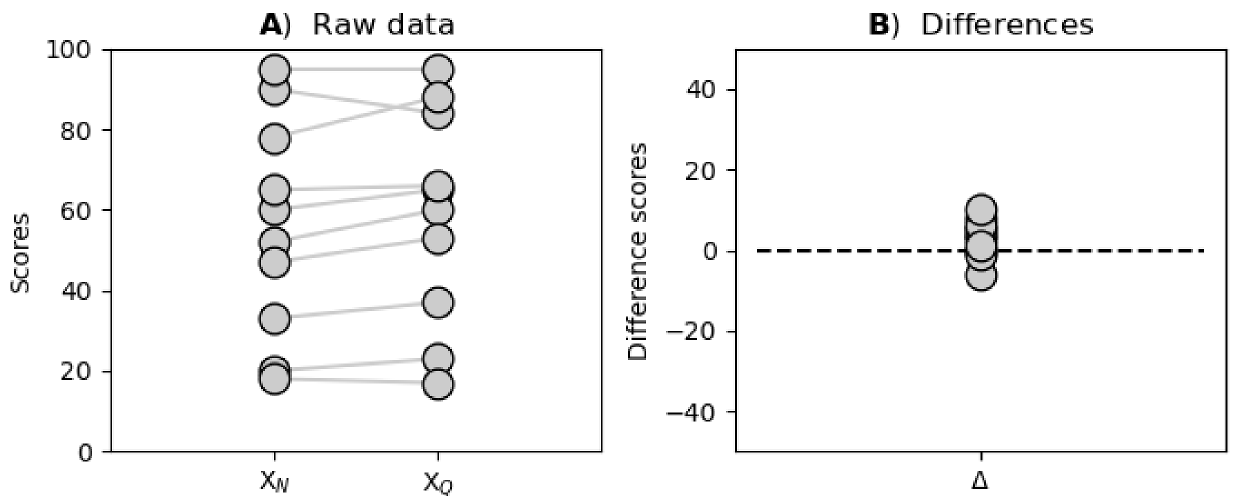
```
<class 'scipy.stats._stats_py.TtestResult'>

TtestResult(statistic=1.624003387693718, pvalue=0.12666866536383706, df=14)

Results from stats.ttest_1samp:
t(14)=1.624, p<0.127
```

[21]:
```
# btw, data are consistent with a normal distribution
print(f'Shapiro p-value = {stats.shapiro(X).pvalue:.2f}')
```

```
Shapiro p-value = 0.96
```

12 Figure 11.8: Paired-samples t-test

[22]:
```
# the data
Xn = np.array([ 60, 52, 90, 20, 33, 95, 18, 47, 78, 65 ])
Xq = np.array([ 65, 60, 84, 23, 37, 95, 17, 53, 88, 66 ])
sampsize = len(Xn)

# their difference
Delta = Xq-Xn

# visualize
_,axs = plt.subplots(1,2,figsize=(7,3))

## draw the individual lines
for i,j in zip(Xn,Xq):
  axs[0].plot([0,1],[i,j],'o-',color=(.8,.8,.8),
              markersize=12,markerfacecolor=(.8,.8,.8),markeredgecolor='k')

axs[0].
↪set(xlim=[-1,2],xticks=[0,1],ylabel='Scores',xticklabels=[r'X$_N$',r'X$_Q$'],
           ylim=[0,100],title=r'$\bf{A}$)  Raw data')

## draw the difference scores
axs[1].plot(np.zeros(sampsize),Delta,'ko',markersize=12,markerfacecolor=(.8,.8,.
↪8))
axs[1].plot([-.1,.1],[0,0],'k--',zorder=-1)
axs[1].set(xticks=[0],ylabel='Difference scores',xticklabels=[r'$\Delta$'],
           ylim=[-50,50],title=r'$\bf{B}$)  Differences')
# export
plt.tight_layout()
#plt.savefig('ttest_pairedTtest.png')
plt.show()
```



[23]:
```
# test it!
ttest = stats.ttest_1samp(Delta,0)

# print the results
print(f't({ttest.df})={ttest.statistic:.3f}, p<{ttest.pvalue:.3f}')
```

```
t(9)=2.023, p<0.074
```

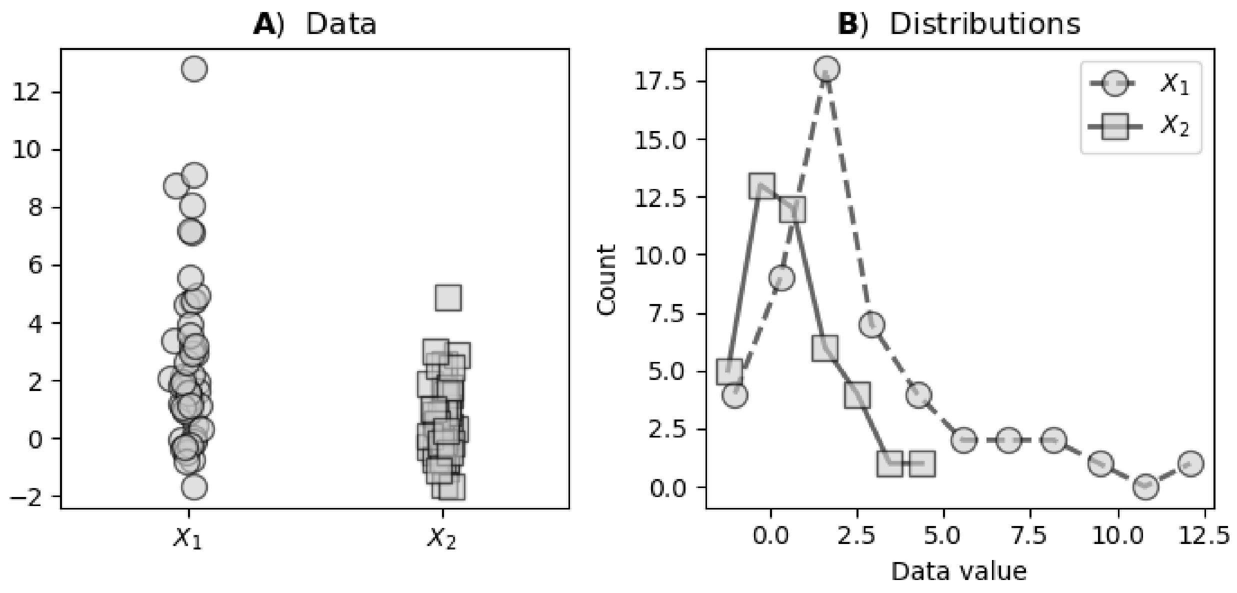
[24]:
```
# btw, data are consistent with a normal distribution
print(f'Xn Shapiro p-value = {stats.shapiro(Xn).pvalue:.2f}')
print(f'Xq Shapiro p-value = {stats.shapiro(Xq).pvalue:.2f}')
print(f'Xy Shapiro p-value = {stats.shapiro(Delta).pvalue:.2f}')
```

```
Xn Shapiro p-value = 0.68
Xq Shapiro p-value = 0.63
Xy Shapiro p-value = 0.98
```

13 Figure 11.9: Example of 2-sample ttest

[25]:
```
# generate data
data1 = stats.exponnorm.rvs(3,size=50)
data2 = stats.gumbel_r.rvs(size=42)

# compute their histograms
yy1,xx1 = np.histogram(data1,bins='fd')
xx1 = (xx1[1:]+xx1[:-1])/2
yy2,xx2 = np.histogram(data2,bins='fd')
xx2 = (xx2[1:]+xx2[:-1])/2

# show the data!
_,axs = plt.subplots(1,2,figsize=(7,3.5))
```

```
## raw data
axs[0].plot(np.random.randn(len(data1))/40,data1,
            'ko',markersize=10,markerfacecolor=(.8,.8,.8),alpha=.6)
axs[0].plot(np.random.randn(len(data2))/40+1,data2,
            'ks',markersize=10,markerfacecolor=(.8,.8,.8),alpha=.6)
axs[0].set(xlim=[-.5,1.5],xticks=[0,1],xticklabels=[r'$X_1$',r'$X_2$'],
           title=r'$\bf{A}$)  Data')

## histograms
axs[1].plot(xx1,yy1,'ko--',markersize=10,markerfacecolor=(.8,.8,.8),alpha=.
 ↪6,linewidth=2,label=r'$X_1$')
axs[1].plot(xx2,yy2,'ks-',markersize=10,markerfacecolor=(.8,.8,.8),alpha=.
 ↪6,linewidth=2,label=r'$X_2$')
axs[1].set(xlabel='Data value',ylabel='Count',title=r'$\bf{B}$)  Distributions')
axs[1].legend()

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



[26]:
```
# doubling rubric
s1 = np.std(data1,ddof=1)
s2 = np.std(data2,ddof=1)

# report
print(f'Standard deviations are {s1:.2f} and {s2:.2f}')
print(f'Ratio of max:min stdevs is {np.max([s1,s2])/np.min([s1,s2]):.2f}')
```

```python
# Levene's test
lres = stats.levene(data1,data2)
print('')
print(f"Levene's test for homogeneity of variance: W={lres.statistic:.2f},␣
 ↪p={lres.pvalue:.3f}")
```

```
Standard deviations are 2.92 and 1.36
Ratio of max:min stdevs is 2.15

Levene's test for homogeneity of variance: W=7.93, p=0.006
```

[27]:
```python
## tests for normal distribution

# omnibus test
Otest1 = stats.normaltest(data1)
Otest2 = stats.normaltest(data2)

print(f'Omnibus test in X1 (H0=normal): p={Otest1.pvalue:.3f}')
print(f'Omnibus test in X2 (H0=normal): p={Otest2.pvalue:.3f}')
print('')

# Shapiro's test
Stest1 = stats.shapiro(data1)
Stest2 = stats.shapiro(data2)

print(f'Shapiro test in X1 (H0=normal): p={Stest1.pvalue:.3f}')
print(f'Shapiro test in X2 (H0=normal): p={Stest2.pvalue:.3f}')
```

```
Omnibus test in X1 (H0=normal): p=0.000
Omnibus test in X2 (H0=normal): p=0.044

Shapiro test in X1 (H0=normal): p=0.000
Shapiro test in X2 (H0=normal): p=0.146
```

[28]:
```python
# now for the t-test
tres = stats.ttest_ind(data1,data2,equal_var=False)
print(f't={tres.statistic:.2f}, p={tres.pvalue:.3f}')
```

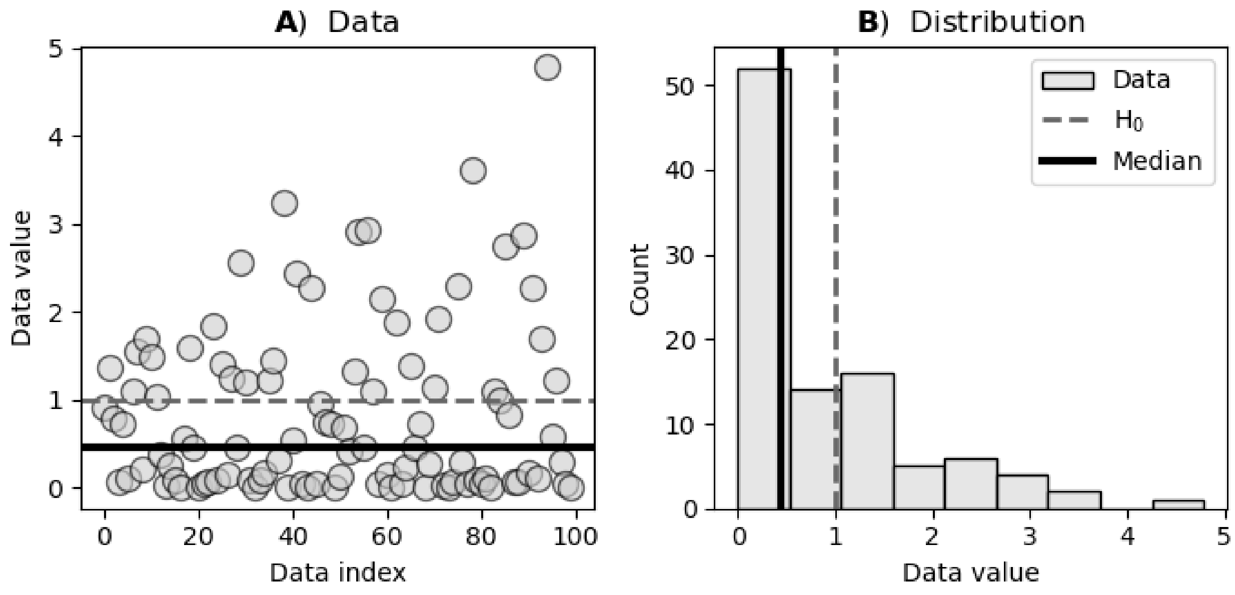
```
t=4.15, p=0.000
```

[29]:
```python
# FYI, here's the result assuming equal variance (see also Exercise 9)
tres = stats.ttest_ind(data1,data2,equal_var=True)
print(f't={tres.statistic:.2f}, p={tres.pvalue:.3f}')
```

```
t=3.92, p=0.000
```

14 Figure 11:10: Wilcoxon signed-rank

[30]:
```
# the data
data = np.random.randn(100)**2
h0 = 1

# show the data!
_,axs = plt.subplots(1,2,figsize=(7,3.5))

## raw data
axs[0].plot(data,'ko',markersize=10,markerfacecolor=(.8,.8,.8),alpha=.6)
axs[0].axhline(h0,linestyle='--',color=(.4,.4,.4),linewidth=2)
axs[0].axhline(np.median(data),color='k',linewidth=3)
axs[0].set(xlabel='Data index',ylabel='Data value',title=r'$\bf{A}$)  Data')

## histogram
axs[1].hist(data,bins='fd',facecolor=(.9,.9,.9),edgecolor='k',label='Data')
axs[1].axvline(h0,linestyle='--',color=(.4,.4,.4),linewidth=2,label=r'H$_0$')
axs[1].axvline(np.median(data),color='k',linewidth=3,label='Median')
axs[1].set(xlabel='Data value',ylabel='Count',title=r'$\bf{B}$)  Distribution')
axs[1].legend()

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

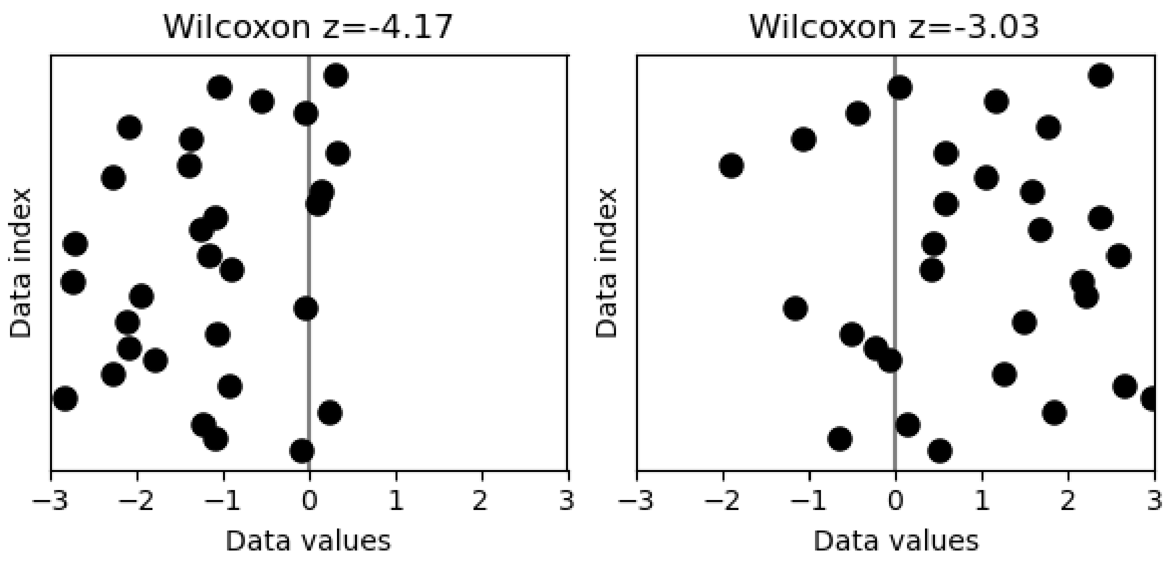


```
[31]: # the test!
wtest = stats.wilcoxon(data-h0,method='approx')

# and print the results
print(f'Wilcoxon test: z={wtest.zstatistic:.2f}, p={wtest.pvalue:.3f}')
```

```
Wilcoxon test: z=-2.68, p=0.007
```

15 Figure 11.11: Margin figure about the sign of z

```
[33]: # create the data, shifted by H0=1
_,axs = plt.subplots(1,2,figsize=(6,3))

for i in range(2):

  # create and shift data
  d = np.random.randn(30)
  d += i*2-1

  # Wilcoxon z-score
  z = stats.wilcoxon(d,method='approx',alternative='two-sided').zstatistic

  # draw the figure
  axs[i].plot(d,range(len(d)),'ko',markersize=8)
  axs[i].axvline(0,zorder=-1,color='gray')
  axs[i].set(xlabel='Data values',ylabel='Data indercise␣
↪3',yticks=[],xlim=[-3,3])
  axs[i].set_title(f'Wilcoxon z={z:.2f}',loc='center')

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



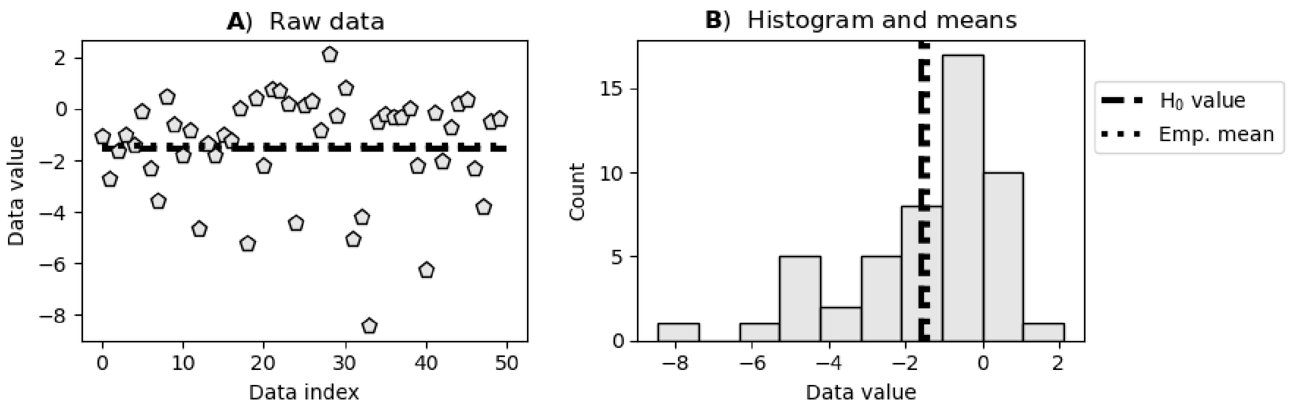
16 Mann-Whitney U test

[34]:
```python
# same data as we used for the independent-samples t-test
data1 = stats.exponnorm.rvs(3,size=50)
data2 = stats.gumbel_r.rvs(size=42)

# MW-U test
mwu = stats.mannwhitneyu(data1,data2)
print(f'U = {mwu.statistic:.0f}, p = {mwu.pvalue:.3f}')

# parametric t-test (gives the same statistical conclusion as the MWU)
tres = stats.ttest_ind(data1,data2,equal_var=False)
print(f't = {tres.statistic:.2f}, p = {tres.pvalue:.3f}')
```

```
U = 1587, p = 0.000
t = 4.39, p = 0.000
```

17 71

[35]:
```python
# parameters
N  = 50
h0 = -np.pi/2

# create the dataset
X = stats.laplace_asymmetric.rvs(2,size=N)
dataMean = np.mean(X)

# visualize the data
_,axs = plt.subplots(1,2,figsize=(9,3))

axs[0].plot(X,'kp',markersize=8,markerfacecolor=(.9,.9,.9),label='Data')
axs[0].plot([0,N],[h0,h0],'k--',zorder=-10,linewidth=3,label=r'H$_0$ value')
axs[0].plot([0,N],[dataMean,dataMean],'k:',linewidth=3,label='Emp. mean')
axs[0].set(xlabel='Data index',ylabel='Data value')
axs[0].set_title(r'$\bf{A}$)  Raw data')

axs[1].hist(X,bins='fd',color=(.9,.9,.9),edgecolor='k')
axs[1].axvline(h0,linestyle='--',color='k',linewidth=3,label=r'H$_0$ value')
axs[1].axvline(dataMean,linestyle=':',color='k',linewidth=3,label=r'Emp. mean')
axs[1].set(xlabel='Data value',ylabel='Count')
axs[1].set_title(r'$\bf{B}$)  Histogram and means')
axs[1].legend(bbox_to_anchor=[1,.9])

plt.tight_layout()
#plt.savefig('ttest_ex1.png')
plt.show()
```



[36]:
```
# now for the t-tests

## manual calculation
t_num = dataMean - h0
t_den = np.std(X,ddof=1) / np.sqrt(N)

tval  = t_num / t_den
pval  = 1-stats.t.cdf( np.abs(tval) ,N-1)
pval *= 2 # double it for 2-tailed test

## using scipy.stats
r   = stats.ttest_1samp(X,h0)
t   = r.statistic
df  = r.df
p   = r.pvalue

# print both results
print(f'Manual ttest: t({N-1})={tval:.3f}, p={pval:.3f}')
print(f'Scipy  ttest: t({df})={t:.3f}, p={p:.3f}')
```

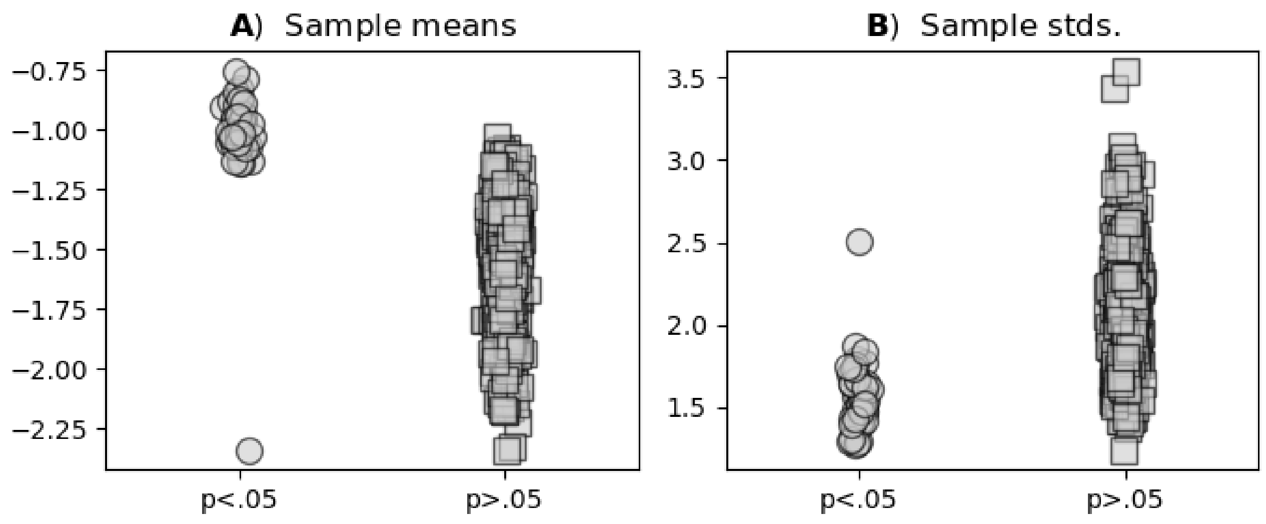
```
Manual ttest: t(49)=0.443, p=0.660
Scipy  ttest: t(49)=0.443, p=0.660
```

18 112

[37]:
```
# how often do we get subthreshold results?

nExps = 500
issig = np.zeros(nExps,dtype=bool) # variable type 'bool' for convenience in␣
 ↪plotting
means = np.zeros(nExps)
stds  = np.zeros(nExps)
```

```
# run the experiment
#   (Note: For a small extra challenge, you could re-implement this without
#          a for-loop using matrix input, after completing the next exercise.)
for i in range(nExps):

  # generate data and store the mean/std
  X = stats.laplace_asymmetric.rvs(2,size=N)
  means[i] = np.mean(X)
  stds[i]  = np.std(X,ddof=1)

  # run the ttest and store if "significant"
  r = stats.ttest_1samp(X,h0)
  issig[i] = r.pvalue<.05

# print the results
print(f'p<.05 in {np.sum(issig)}/{nExps} times.')

# show the results
_,axs = plt.subplots(1,2,figsize=(7,3))

# means
axs[0].plot(np.random.randn(sum(issig))/40,means[issig],
            'ko',markersize=10,markerfacecolor=(.8,.8,.8),alpha=.6)
axs[0].plot(np.random.randn(sum(~issig))/40+1,means[~issig],
            'ks',markersize=10,markerfacecolor=(.8,.8,.8),alpha=.6)
axs[0].set(xlim=[-.5,1.5],xticks=[0,1],xticklabels=['p<.05','p>.05'],
           title=r'$\bf{A}$)  Sample means')

# stds
axs[1].plot(np.random.randn(sum(issig))/40,stds[issig],
            'ko',markersize=10,markerfacecolor=(.8,.8,.8),alpha=.6)
axs[1].plot(np.random.randn(sum(~issig))/40+1,stds[~issig],
            'ks',markersize=10,markerfacecolor=(.8,.8,.8),alpha=.6)
axs[1].set(xlim=[-.5,1.5],xticks=[0,1],xticklabels=['p<.05','p>.05'],
           title=r'$\bf{B}$)  Sample stds.')

plt.tight_layout()
#plt.savefig('ttest_ex2.png')
plt.show()
```

```
p<.05 in 42/500 times.
```



19 Exercise 3

[38]:
```
NperSample = 40
MDatasets = 25

# data
X = np.random.normal(loc=1,scale=1,size=(NperSample,MDatasets))

# confirm data size
print('Data size should be sample-size X datasets:')
print(X.shape)
```

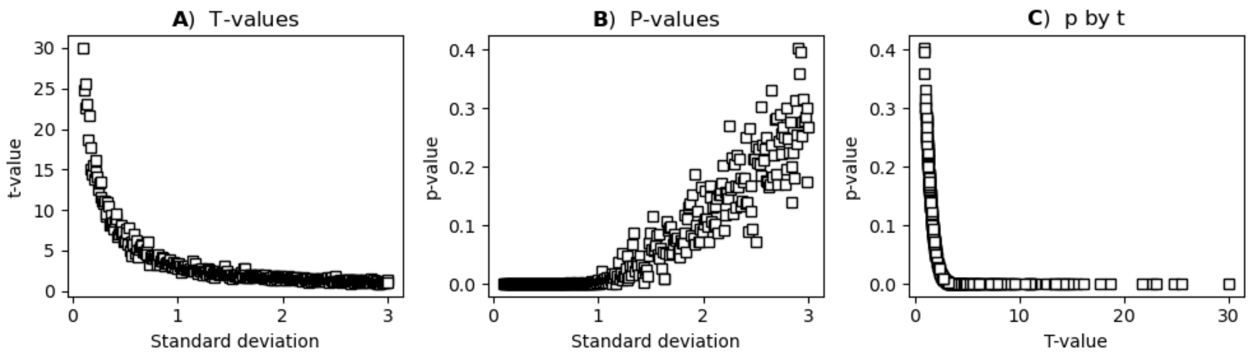
```
Data size should be sample-size X datasets:
(40, 25)
```

[39]:
```
# ttest with matrix input
ttest_matrix = stats.ttest_1samp(X,0)

# ttest in for-loop over each column (each dataset)
ttest_4loop = np.zeros(MDatasets)
for i in range(MDatasets):
  ttest_4loop[i] = stats.ttest_1samp(X[:,i],0).statistic

# print the results
print('Matrix  |  Vector')
print('--------|--------')
for i in range(MDatasets):
  print(f'{ttest_matrix.statistic[i]:.4f}  |  {ttest_4loop[i]:.4f}')
```

```
Matrix   |  Vector
---------|--------
5.4770   |  5.4770
6.4995   |  6.4995
8.6500   |  8.6500
8.4548   |  8.4548
5.6364   |  5.6364
5.9598   |  5.9598
5.2110   |  5.2110
6.2969   |  6.2969
6.5679   |  6.5679
5.4774   |  5.4774
5.5437   |  5.5437
6.5186   |  6.5186
6.0120   |  6.0120
5.9611   |  5.9611
5.8085   |  5.8085
6.8033   |  6.8033
6.0656   |  6.0656
5.4532   |  5.4532
6.6632   |  6.6632
7.8656   |  7.8656
6.9182   |  6.9182
5.9154   |  5.9154
5.2762   |  5.2762
5.9897   |  5.9897
6.2182   |  6.2182
```

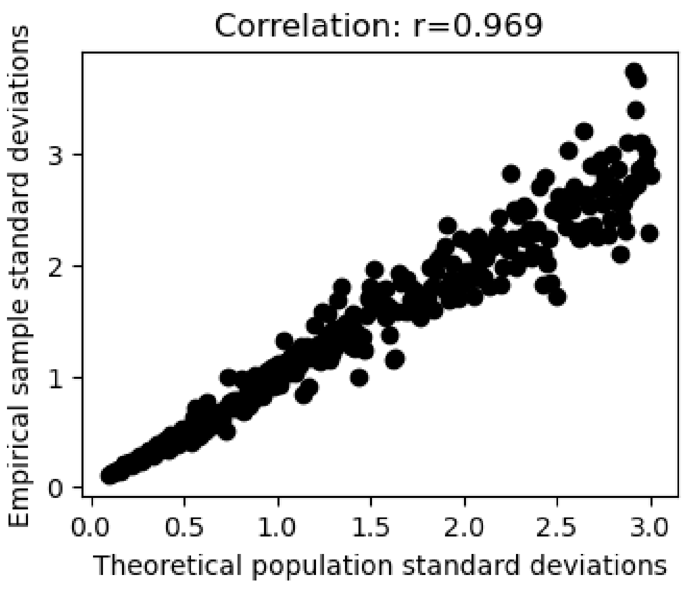
20 Exercise 4

[40]:
```
# data parameters
N = 40
k = 300

# list of standard deviations
stds = np.linspace(.1,3,k)

# initialize the t/p vectors
t = np.zeros(k)
p = np.zeros(k)
s = np.zeros(k) # this line is for exercise 5

for i in range(len(stds)):
  X = np.random.normal(0,stds[i],size=N)
  X = X-np.mean(X) + .5 # force mean=.5
  ttest = stats.ttest_1samp(X,0)
  t[i]  = ttest.statistic
  p[i]  = ttest.pvalue

  # get the sample std (used in exercise 5)
  s[i] = np.std(X,ddof=1)

# and now the plotting
_,axs = plt.subplots(1,3,figsize=(10,3))

# t's
axs[0].plot(stds,t,'ks',markerfacecolor='w')
axs[0].set(xlabel='Standard deviation',ylabel='t-value',title=r'$\bf{A}$) ␣
↪T-values')

# p's
axs[1].plot(stds,p,'ks',markerfacecolor='w')
axs[1].set(xlabel='Standard deviation',ylabel='p-value',title=r'$\bf{B}$) ␣
↪P-values')

# t and p
axs[2].plot(t,p,'ks',markerfacecolor='w')
axs[2].set(xlabel='T-value',ylabel='p-value',title=r'$\bf{C}$)  p by t')

plt.tight_layout()
#plt.savefig('ttest_ex4.png')
plt.show()
```



21 Exercise 5

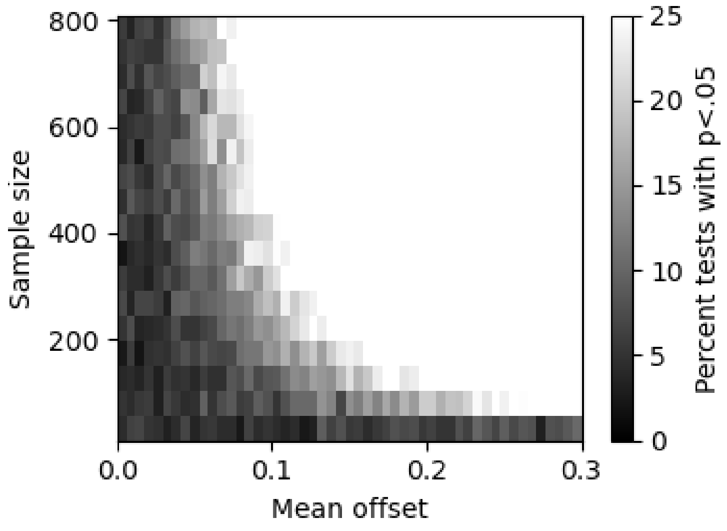
[47]:
```
# No, it doesn't really matter, because even with N=40, the sample standard␣
↪deviation
# is a fairly accurate estimate of the population standard deviation, certainly␣
↪for
# this range of standard deviation values.
# You can recreate the figure by replacing variable 'stds' with 's' in the code␣
 ↪above,
# and by demonstrating the strong correlation between sample and theoretical␣
 ↪standard deviation.

# correlation coefficient (values close to 1 indicate a very strong relationship)
r = np.corrcoef(stds,s)

# plot
plt.figure(figsize=(4,3))
plt.plot(stds,s,'ko')
plt.xlabel('Theoretical population standard deviations')
plt.ylabel('Empirical sample standard deviations')
plt.title(f'Correlation: r={r[0,1]:.3f}',loc='center')
plt.show()
```



22 Exercise 6

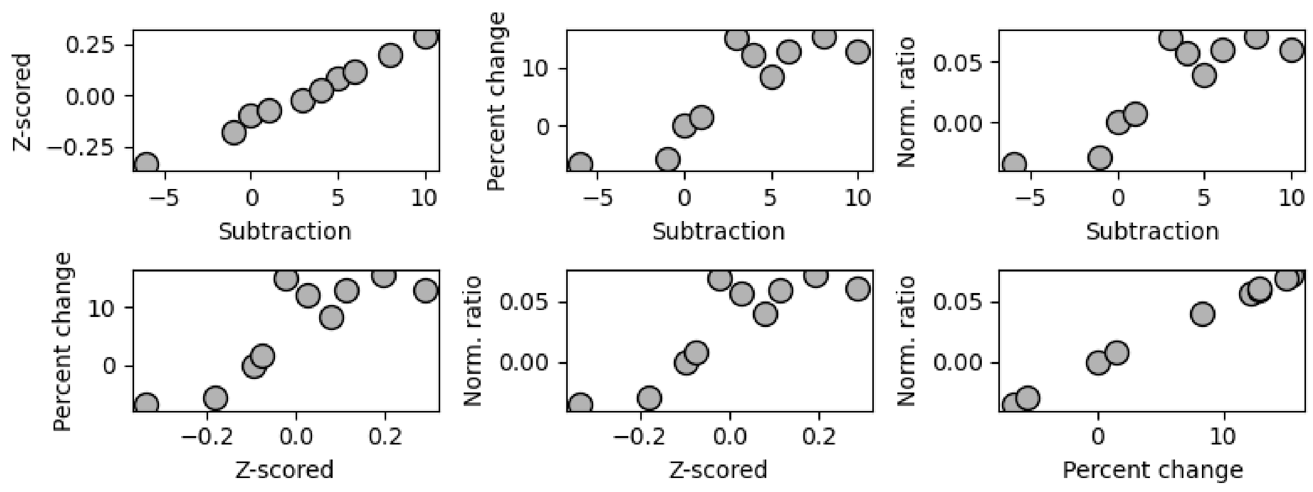
[49]:
```
nExperiments = 250
meanoffsets = np.linspace(0,.3,51)
samplesizes = np.arange(10,811,step=50)

# initialize
propSig = np.zeros((len(samplesizes),len(meanoffsets)))

# loop over sample sizes
```

```
for sidx,ssize in enumerate(samplesizes):

  # loop over effect sizes
  for eidx,effect in enumerate(meanoffsets):

    # generate the data
    X = np.random.normal(loc=effect,scale=1.5,size=(ssize,nExperiments))

    # run the t-test and store the results
    T = stats.ttest_1samp(X,0)
    propSig[sidx,eidx] = 100*np.mean( T.pvalue<.05 )

# visualize in a matrix
plt.figure(figsize=(4,3))
plt.
 ↪imshow(propSig,extent=[meanoffsets[0],meanoffsets[-1],samplesizes[0],samplesizes[-1]],
           vmin=0,vmax=25,origin='lower',aspect='auto',cmap='gray')
plt.xlabel('Mean offset')
plt.ylabel('Sample size')
cbar = plt.colorbar()
cbar.set_label('Percent tests with p<.05')

plt.tight_layout()
#plt.savefig('ttest_ex6.png')
plt.show()
```



23 Exercise 7

[50]:
```python
Xn = np.array([ 60, 52, 90, 20, 33, 95, 18, 47, 78, 65 ])
Xq = np.array([ 65, 60, 84, 23, 37, 95, 17, 53, 88, 66 ])
sampsize = len(Xn)

# simple subtraction (Y1 in the text)
Ysub = Xq-Xn
# zscore subtraction (Y2 in the text)
Ysbz = stats.zscore(Xq) - stats.zscore(Xn)
# percent change (Y3 in the text)
Ypct = 100*(Xq-Xn) / Xn
# normalized ratio (Y4 in the text)
Ynrt = (Xq-Xn) / (Xq+Xn)

# plot
_,axs = plt.subplots(2,3,figsize=(8,3))
axs[0,0].plot(Ysub,Ysbz,'ko',markersize=10,markerfacecolor=(.7,.7,.7))
axs[0,0].set(xlabel='Subtraction',ylabel='Z-scored')

axs[0,1].plot(Ysub,Ypct,'ko',markersize=10,markerfacecolor=(.7,.7,.7))
axs[0,1].set(xlabel='Subtraction',ylabel='Percent change')

axs[0,2].plot(Ysub,Ynrt,'ko',markersize=10,markerfacecolor=(.7,.7,.7))
axs[0,2].set(xlabel='Subtraction',ylabel='Norm. ratio')

axs[1,0].plot(Ysbz,Ypct,'ko',markersize=10,markerfacecolor=(.7,.7,.7))
axs[1,0].set(xlabel='Z-scored',ylabel='Percent change')

axs[1,1].plot(Ysbz,Ynrt,'ko',markersize=10,markerfacecolor=(.7,.7,.7))
axs[1,1].set(xlabel='Z-scored',ylabel='Norm. ratio')

axs[1,2].plot(Ypct,Ynrt,'ko',markersize=10,markerfacecolor=(.7,.7,.7))
axs[1,2].set(xlabel='Percent change',ylabel='Norm. ratio')

plt.tight_layout()
plt.show()
```



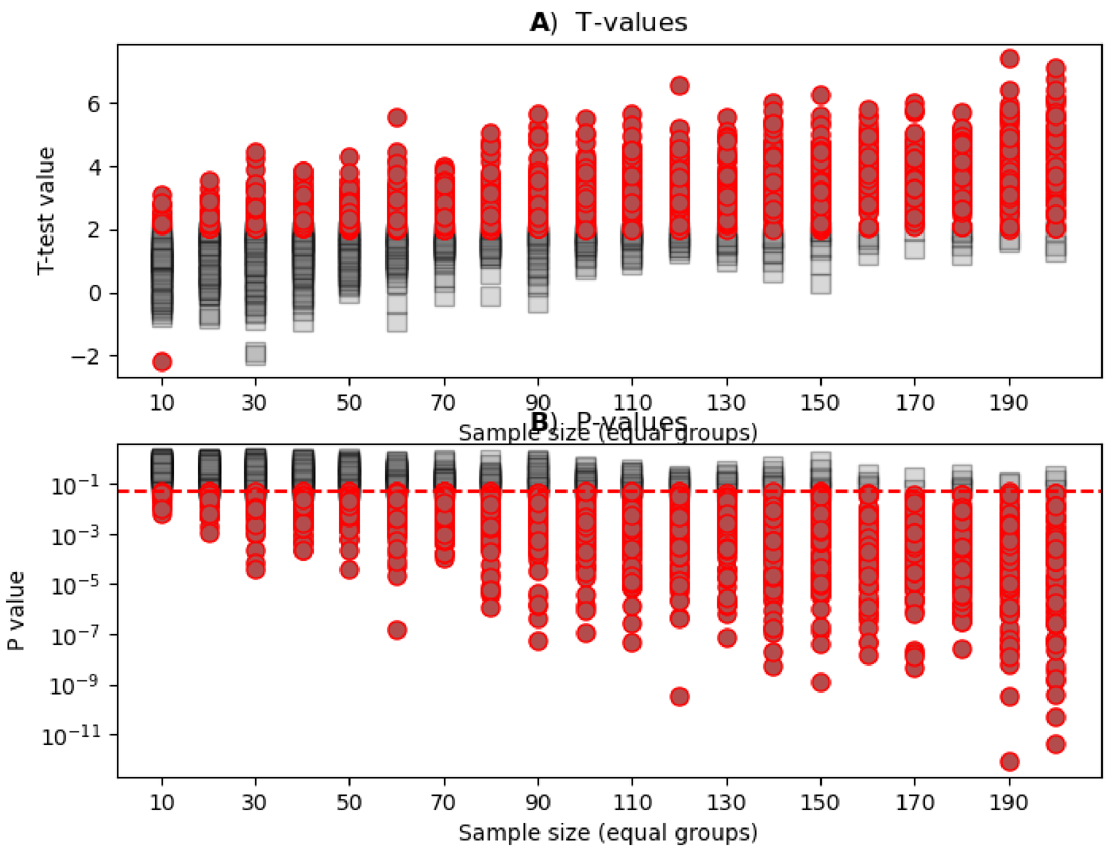
```
[51]: # t-tests
tSub = stats.ttest_1samp(Ysub,0)
tPct = stats.ttest_1samp(Ypct,0)
tsbz = stats.ttest_1samp(Ysbz,0)
tnrt = stats.ttest_1samp(Ynrt,0)

# print the results
print(f'Subtraction (Y1): t({tSub.df})={tSub.statistic:.3f}, p<{tSub.pvalue:.
 ↪3f}')
print(f'Percent chg (Y2): t({tPct.df})={tPct.statistic:.3f}, p<{tPct.pvalue:.
 ↪3f}')
print(f'Z subtract  (Y3): t({tsbz.df})={tsbz.statistic:.3f}, p<{tsbz.pvalue:.
 ↪3f}')
print(f'Norm. ratio (Y4): t({tnrt.df})={tnrt.statistic:.3f}, p<{tnrt.pvalue:.
 ↪3f}')
```

```
Subtraction (Y1): t(9)=2.023, p<0.074
Percent chg (Y2): t(9)=2.445, p<0.037
Z subtract  (Y3): t(9)=0.000, p<1.000
Norm. ratio (Y4): t(9)=2.353, p<0.043
```

24 Exercise 8

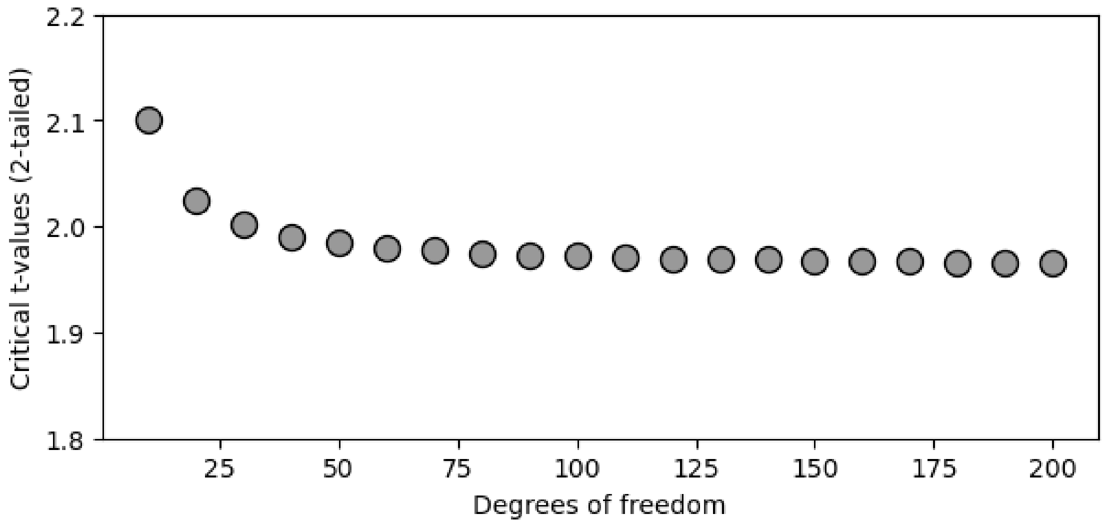
```
[53]: # parameters
mu1 = 1.2 # population mean in dataset 1
mu2 = 1   # population mean in dataset 2

# sample sizes
ns = np.arange(10,201,step=10)

# setup the figure
_,axs = plt.subplots(2,1,figsize=(8,6))
```

```
# start the experiment!
for ni,N in enumerate(ns):
  # generate the data (100 datasets at a time)
  data1 = np.random.normal(loc=mu1,scale=.5,size=(N,100))
  data2 = np.random.normal(loc=mu2,scale=.5,size=(N,100))

  # run the ttest
  ttest = stats.ttest_ind(data1,data2)
  t = ttest.statistic;
  p = ttest.pvalue;

  # plot the t-value, colored by significance
  axs[0].plot(np.full(np.sum(p>.05),N),t[p>.
↪05],'ks',markersize=8,markerfacecolor=(.5,.5,.5),alpha=.3)
  axs[0].plot(np.full(np.sum(p<.05),N),t[p<.
↪05],'ro',markersize=8,markerfacecolor=(.7,.3,.3))

  # plot the p-values
  axs[1].plot(np.full(np.sum(p>.05),N),p[p>.
↪05],'ks',markersize=8,markerfacecolor=(.5,.5,.5),alpha=.3)
  axs[1].plot(np.full(np.sum(p<.05),N),p[p<.
↪05],'ro',markersize=8,markerfacecolor=(.7,.3,.3))

## rest of the visualization
axs[0].set(xlabel='Sample size (equal groups)',xticks=ns[::2],ylabel='T-test␣
↪value')
axs[0].set_title(r'$\bf{A}$)  T-values')

# adjust the p-values panel
axs[1].set(xlabel='Sample size (equal groups)',xticks=ns[::2],ylabel='P value')
axs[1].set_yscale('log')
axs[1].axhline(.05,linestyle='--',color='r')
axs[1].set_title(r'$\bf{B}$)  P-values')

plt.figure(figsize=(8,3))
plt.tight_layout()
#plt.savefig('ttest_ex8.png')
plt.show()
```



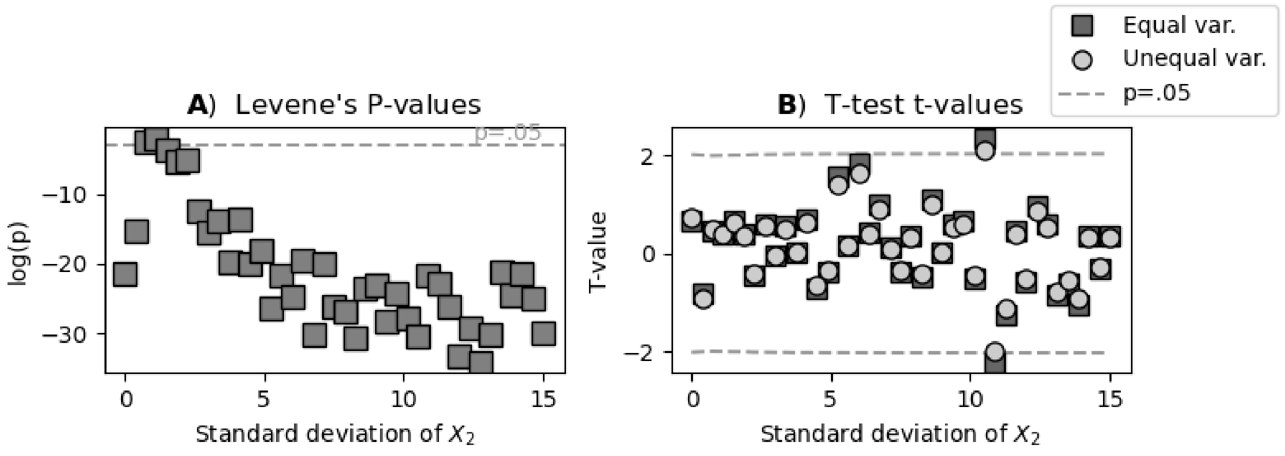
```
<Figure size 800x300 with 0 Axes>
```

[54]:
```
# compute critical t-values for the degrees of freedom
tCrit = stats.t.isf(.05/2,2*ns-2)

# and visualize
plt.figure(figsize=(7,3))
plt.plot(ns,tCrit,'ko',markersize=10,markerfacecolor=(.6,.6,.6))
plt.ylim([1.8,2.2])
plt.xlabel('Degrees of freedom')
plt.ylabel('Critical t-values (2-tailed)')
plt.show()
```

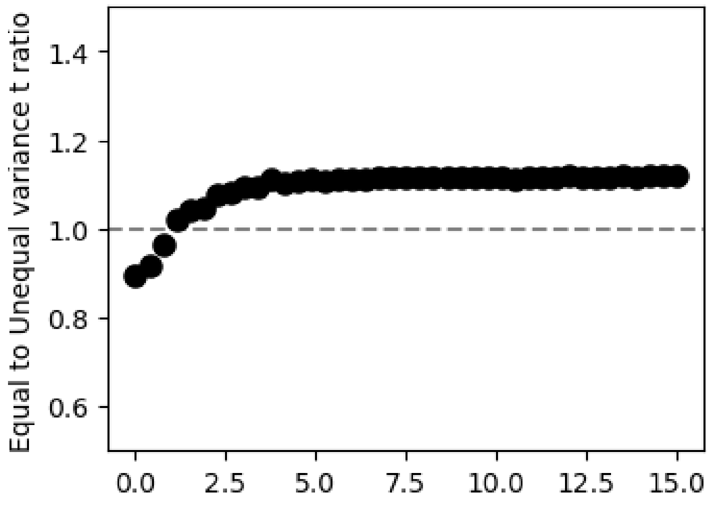


25 Exercise 9

[55]:
```python
# range of standard deviations
stdevs = np.linspace(.01,15,41)

# initialize results matrix
results = np.zeros((3,len(stdevs)))
tCrit = np.zeros(len(stdevs))

# start the experiment!
for si,std in enumerate(stdevs):
  # create two groups of data
  X1 = np.random.normal(loc=1,scale=1,size=50)
  X2 = np.random.normal(loc=1.1,scale=std,size=40)

  # levene's test
  results[0,si] = np.log( stats.levene(X1,X2).pvalue )

  # difference of t-values
  same_var = stats.ttest_ind(X1,X2,equal_var=True)  # equal variance
  diff_var = stats.ttest_ind(X1,X2,equal_var=False) # unequal variance
  results[1,si] = same_var.statistic # equal variance
  results[2,si] = diff_var.statistic # unequal variance

  # compute df for tCrit
  s1,s2 = np.var(X1,ddof=1),np.var(X2,ddof=1)
  n1,n2 = len(X1),len(X2)
  df_num = (s1/n1 + s2/n2)**2
  df_den = s1**2/(n1**2*(n1-1)) + s2**2/(n2**2*(n2-1))

  tCrit[si] = stats.t.isf(.05/2,df_num/df_den)

# plot
_,axs = plt.subplots(1,2,figsize=(8,3))

# levene's test results
axs[0].plot(stdevs,results[0,:],'ks',markersize=10,markerfacecolor='gray')
axs[0].axhline(np.log(.05),color=(.6,.6,.6),linestyle='--',zorder=-1)
axs[0].text(np.max(stdevs),np.log(.1),'p=.05',ha='right',color=(.6,.6,.6))
axs[0].set(xlabel=r'Standard deviation of␣
 ↪$X_2$',ylabel='log(p)',title=r"$\bf{A}$)  Levene's P-values")

# t-tests
axs[1].plot(stdevs,results[1,:],'ks',markersize=8,markerfacecolor=(.4,.4,.
 ↪4),label='Equal var.')
axs[1].plot(stdevs,results[2,:],'ko',markersize=8,markerfacecolor=(.8,.8,.
 ↪8),label='Unequal var.')
axs[1].plot(stdevs,tCrit,'--',color=(.6,.6,.6),zorder=-1,label='p=.05')
axs[1].plot(stdevs,-tCrit,'--',color=(.6,.6,.6),zorder=-1)
axs[1].set(xlabel=r'Standard deviation of␣
 ↪$X_2$',ylabel='T-value',title=r"$\bf{B}$)  T-test t-values")
axs[1].legend(fontsize=10,bbox_to_anchor=[.8,1])

plt.tight_layout()
#plt.savefig('ttest_ex9.png')
plt.show()
```



[57]:
```
# Not in the instructions, but I think it's also interesting to
#  plot the ratio of t-values as a function of standard deviation
# values >1 indicate a larger t-value for equal compared to unequal variance␣
 ↪formula
plt.figure(figsize=(4,3))
plt.plot(stdevs,results[1,:]/results[2,:],'ko',markersize=8)
plt.axhline(1,linestyle='--',color='gray',zorder=-1)
plt.ylim([.5,1.5])
plt.ylabel('Equal to Unequal variance t ratio')
plt.show()
```



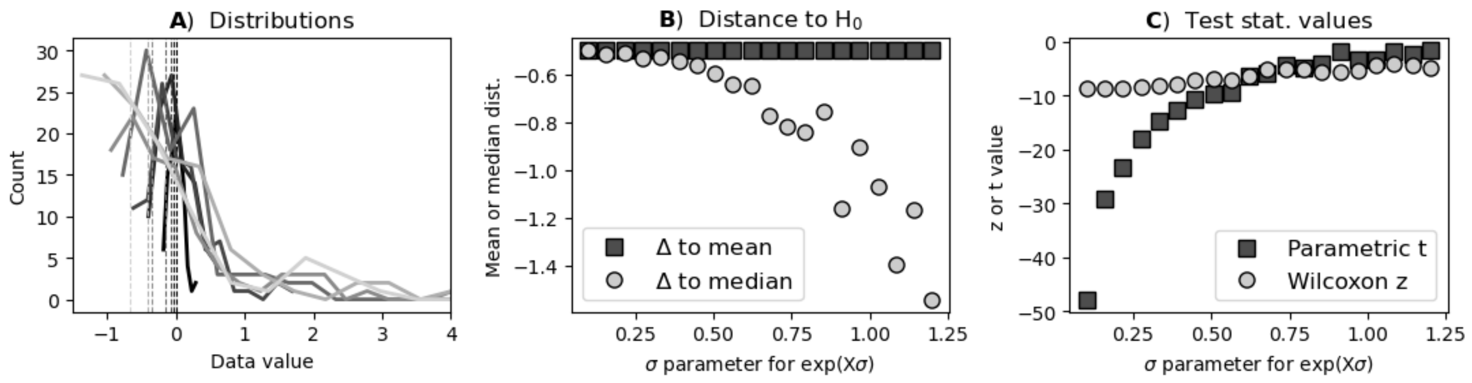
[58]:
```
# If you're bored on a Saturday night, you can also use this code
# to explore the impact of sample sizes and mean offsets.
```

26 Exercise 10

[59]:
```
# generate the data
sigmas = np.linspace(.1,1.2,20)

# null hypothesis value
h0 = .5

# initialize the results matrices
tvals = np.zeros((2,len(sigmas)))
cents = np.zeros((2,len(sigmas)))

_,axs = plt.subplots(1,3,figsize=(11,3))

# compute and store all moments in a matrix
```

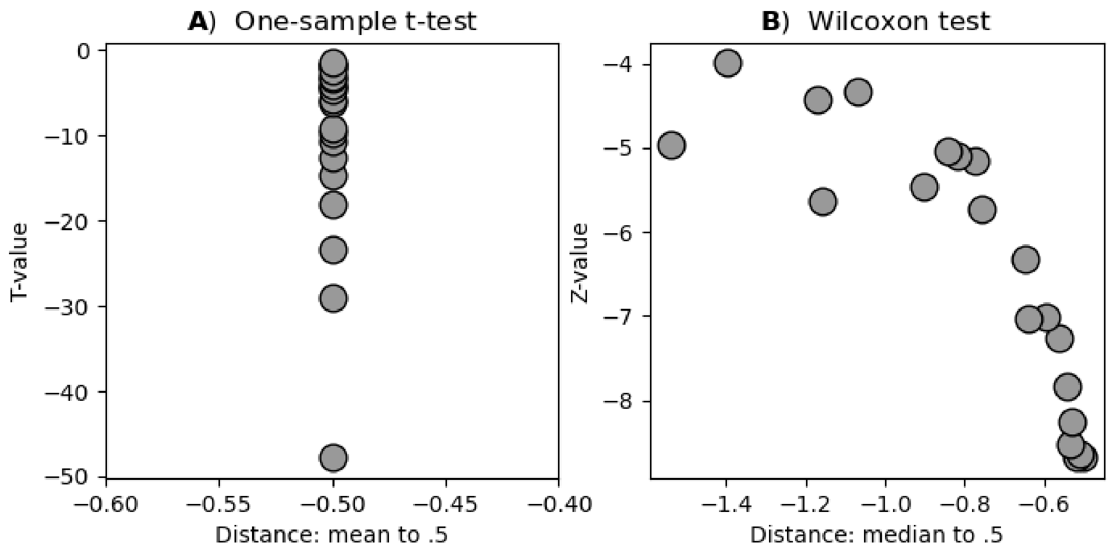
```
for i,s in enumerate(sigmas):
  # generate mean-centered data
  X = np.exp(np.random.randn(100)*s)
  X -= np.mean(X)

  # compute and store the descriptives
  cents[0,i] = np.mean(X) - h0
  cents[1,i] = np.median(X) - h0

  # draw the histogram
  if i%3==0:
    mc = len(sigmas)+2
    y,x = np.histogram(X,bins='fd')
    axs[0].plot((x[:-1]+x[1:])/2,y,color=(i/mc,i/mc,i/mc),linewidth=2)
    axs[0].axvline(np.median(X),color=(i/mc,i/mc,i/mc),linestyle='--',linewidth=.
↪8)

  # parametric t-test
  tvals[0,i] = stats.ttest_1samp(X,h0).statistic

  # Wilcoxon test
  tvals[1,i] = stats.wilcoxon(X-h0,method='approx').zstatistic

## plot
axs[0].set(xlim=[-1.5,4],xlabel='Data value',ylabel='Count')
axs[0].set_title(r'$\bf{A}$)  Distributions')

axs[1].plot(sigmas,cents[0,:],'ks',markersize=8,markerfacecolor=(.3,.3,.
↪3),label=r'$\Delta$ to mean')
axs[1].plot(sigmas,cents[1,:],'ko',markersize=8,markerfacecolor=(.8,.8,.
↪8),label=r'$\Delta$ to median')
axs[1].legend(fontsize=12)
axs[1].set(xlabel=r'$\sigma$ parameter for exp(X$\sigma$)',ylabel='Mean or␣
↪median dist.')
axs[1].set_title(r'$\bf{B}$)  Distance to H$_0$')

axs[2].plot(sigmas,tvals[0,:],'ks',markersize=8,markerfacecolor=(.3,.3,.
↪3),label='Parametric t')
axs[2].plot(sigmas,tvals[1,:],'ko',markersize=8,markerfacecolor=(.8,.8,.
↪8),label='Wilcoxon z')
axs[2].legend(fontsize=12)
axs[2].set(xlabel=r'$\sigma$ parameter for exp(X$\sigma$)',ylabel='z or t value')
axs[2].set_title(r'$\bf{C}$)  Test stat. values')

plt.tight_layout()
#plt.savefig('ttest_ex10a.png')
```

```
plt.show()
```



[61]:
```
# plot showing relationship between central tendency distances and test␣
↪statistic values
_,axs = plt.subplots(1,2,figsize=(8,3.5))

axs[0].plot(cents[0,:],tvals[0,:],'ko',markersize=12,markerfacecolor=(.6,.6,.6))
axs[0].set(xlabel='Distance: mean to .5',ylabel='T-value',xlim=[-.6,-.4],
           title=r'$\bf{A}$)  One-sample t-test')

axs[1].plot(cents[1,:],tvals[1,:],'ko',markersize=12,markerfacecolor=(.6,.6,.6))
axs[1].set(xlabel='Distance: median to .5',ylabel='Z-value',
           title=r'$\bf{B}$)  Wilcoxon test')
plt.figure(figsize=(8,3))
plt.tight_layout()
#plt.savefig('ttest_ex10b.png')
plt.show()
```

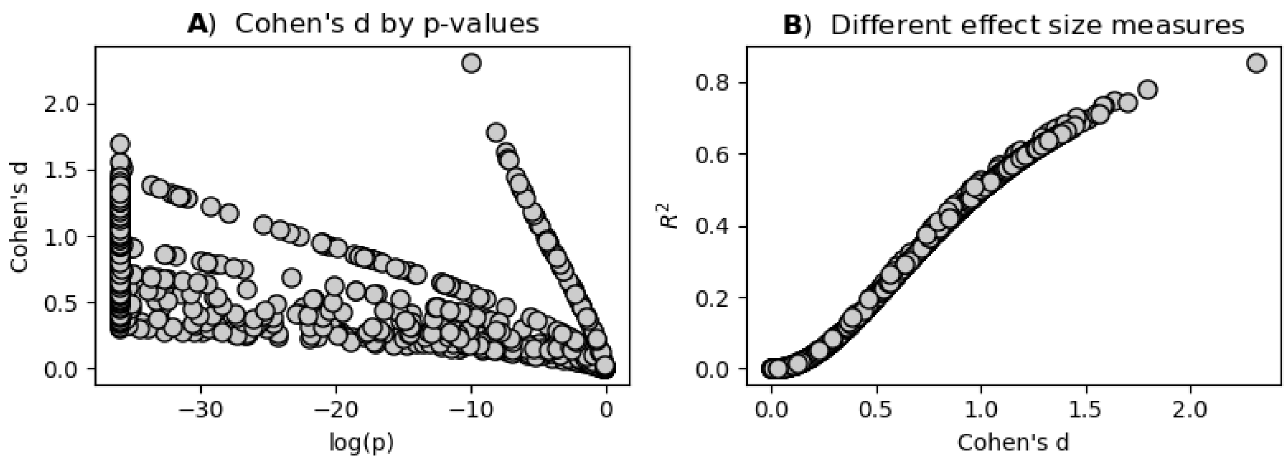
```
<Figure size 800x300 with 0 Axes>
```



27 Exercise 11

[62]:
```python
# params
meanoffsets = np.linspace(0,2,71)
samplesizes = np.arange(10,811,step=50)

# initialize
pvals = np.zeros((len(samplesizes),len(meanoffsets)))
cohend = np.zeros((len(samplesizes),len(meanoffsets)))
r2 = np.zeros((len(samplesizes),len(meanoffsets)))

# loop over sample sizes
for sidx,ssize in enumerate(samplesizes):
  # loop over effect sizes
  for eidx,effect in enumerate(meanoffsets):

    # generate the data
    X = np.random.normal(loc=effect,scale=1.5,size=(ssize))

    # run the t-test and store the results
    T = stats.ttest_1samp(X,0)
    pvals[sidx,eidx] = T.pvalue

    # Cohen's d
    cohend[sidx,eidx] = np.abs(  np.mean(X)/np.std(X,ddof=1)  )

    # R^2
    r2[sidx,eidx] = T.statistic**2 / (T.statistic**2 + T.df)
```

[63]:
```python
_,axs = plt.subplots(1,2,figsize=(8,3))

axs[0].plot(np.log(pvals+np.finfo(float).
 ↪eps),cohend,'ko',markersize=8,markerfacecolor=(.8,.8,.8))
axs[0].set(xlabel='log(p)',ylabel="Cohen's d")
axs[0].set_title(r"$\bf{A}$)  Cohen's d by p-values")

axs[1].plot(cohend,r2,'ko',markersize=8,markerfacecolor=(.8,.8,.8))
axs[1].set(xlabel="Cohen's d",ylabel=r'$R^2$')
axs[1].set_title(r'$\bf{B}$)  Different effect size measures')

plt.tight_layout()
#plt.savefig('ttest_ex11.png')
plt.show()
```



28 Exercise 12

[64]:
```
# data reference

# P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis.
# Modeling wine preferences by data mining from physicochemical properties. In␣
↪Decision Support Systems, Elsevier, 47(4):547-553, 2009.

# https://archive.ics.uci.edu/ml/datasets/Wine+Quality
```

[65]:
```
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/
↪winequality-red.csv"

data = pd.read_csv(url,sep=';')
data
```

[65]:
```
      fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0               7.4             0.700         0.00             1.9      0.076
1               7.8             0.880         0.00             2.6      0.098
2               7.8             0.760         0.04             2.3      0.092
3              11.2             0.280         0.56             1.9      0.075
4               7.4             0.700         0.00             1.9      0.076
...             ...               ...          ...             ...        ...
1594            6.2             0.600         0.08             2.0      0.090
1595            5.9             0.550         0.10             2.2      0.062
1596            6.3             0.510         0.13             2.3      0.076
1597            5.9             0.645         0.12             2.0      0.075
1598            6.0             0.310         0.47             3.6      0.067
```

```
      free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                    11.0                  34.0  0.99780  3.51       0.56   
1                    25.0                  67.0  0.99680  3.20       0.68   
2                    15.0                  54.0  0.99700  3.26       0.65   
3                    17.0                  60.0  0.99800  3.16       0.58   
4                    11.0                  34.0  0.99780  3.51       0.56   
...                   ...                   ...      ...   ...        ...   
1594                 32.0                  44.0  0.99490  3.45       0.58   
1595                 39.0                  51.0  0.99512  3.52       0.76   
1596                 29.0                  40.0  0.99574  3.42       0.75   
1597                 32.0                  44.0  0.99547  3.57       0.71   
1598                 18.0                  42.0  0.99549  3.39       0.66   

      alcohol  quality  
0         9.4        5  
1         9.8        5  
2         9.8        5  
3         9.8        6  
4         9.4        5  
...       ...      ...  
1594     10.5        5  
1595     11.2        6  
1596     11.0        6  
1597     10.2        5  
1598     11.0        6  

[1599 rows x 12 columns]
```

[66]:
```
# describe the data
data.describe()
```

[66]:
```
       fixed acidity  volatile acidity  citric acid  residual sugar  \
count    1599.000000       1599.000000  1599.000000     1599.000000   
mean        8.319637          0.527821     0.270976        2.538806   
std         1.741096          0.179060     0.194801        1.409928   
min         4.600000          0.120000     0.000000        0.900000   
25%         7.100000          0.390000     0.090000        1.900000   
50%         7.900000          0.520000     0.260000        2.200000   
75%         9.200000          0.640000     0.420000        2.600000   
max        15.900000          1.580000     1.000000       15.500000   
```

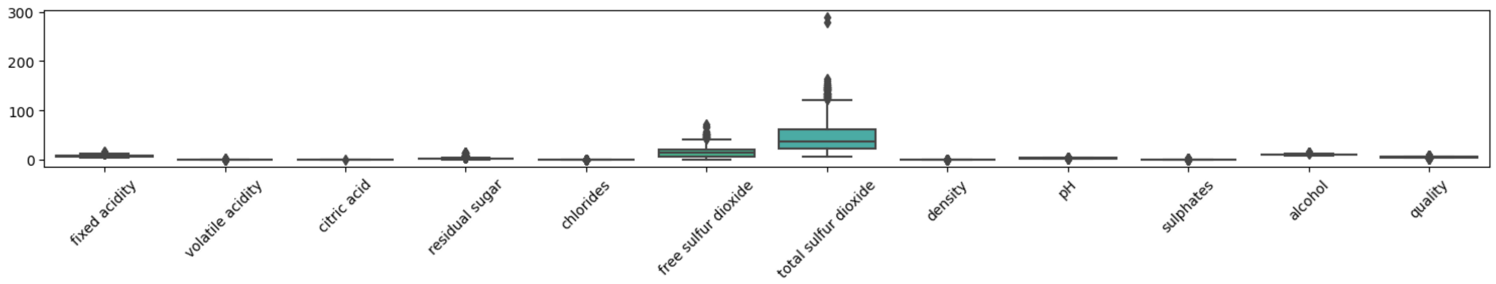
```
         chlorides  free sulfur dioxide  total sulfur dioxide      density  \
count  1599.000000          1599.000000           1599.000000  1599.000000
mean      0.087467            15.874922             46.467792     0.996747
std       0.047065            10.460157             32.895324     0.001887
min       0.012000             1.000000              6.000000     0.990070
25%       0.070000             7.000000             22.000000     0.995600
50%       0.079000            14.000000             38.000000     0.996750
75%       0.090000            21.000000             62.000000     0.997835
max       0.611000            72.000000            289.000000     1.003690

                pH    sulphates      alcohol      quality
count  1599.000000  1599.000000  1599.000000  1599.000000
mean      3.311113     0.658149    10.422983     5.636023
std       0.154386     0.169507     1.065668     0.807569
min       2.740000     0.330000     8.400000     3.000000
25%       3.210000     0.550000     9.500000     5.000000
50%       3.310000     0.620000    10.200000     6.000000
75%       3.400000     0.730000    11.100000     6.000000
max       4.010000     2.000000    14.900000     8.000000
```

[67]:
```python
# list number of unique values per column
for i in data.keys():
  print(f'{i:>20} has {len(np.unique(data[i])):>3} unique values')
```

```
       fixed acidity has  96 unique values
    volatile acidity has 143 unique values
         citric acid has  80 unique values
      residual sugar has  91 unique values
           chlorides has 153 unique values
 free sulfur dioxide has  60 unique values
total sulfur dioxide has 144 unique values
             density has 436 unique values
                  pH has  89 unique values
           sulphates has  96 unique values
             alcohol has  65 unique values
             quality has   6 unique values
```

[68]:
```python
# plot some data
fig,ax = plt.subplots(1,figsize=(15,3))
ax = sns.boxplot(data=data)
ax.set_xticklabels(ax.get_xticklabels(),rotation=45)
```

```
plt.tight_layout()
plt.show()

# optionally remove outliers based on visual inspection (not used for the␣
 ↪exercise)
# data = data[data['total sulfur dioxide']<200]
```

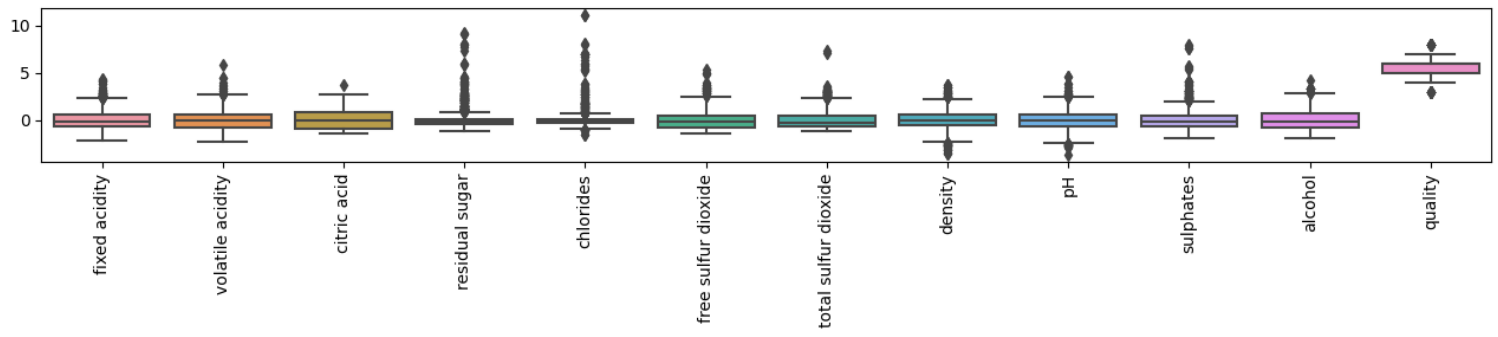


[69]:
```
### z-score all variables except for quality

# find the columns we want to normalize (all except quality)
cols2zscore = data.keys()
cols2zscore = cols2zscore.drop('quality')

# make a copy of the dataframe to change
dataz = data.copy()

# z-score (written out for clarity)
for col in cols2zscore:
  meanval   = np.mean(data[col])
  stdev     = np.std(data[col],ddof=1)
  dataz[col] = (data[col]-meanval) / stdev

# can also do more compactly
#data[cols2zscore] = data[cols2zscore].apply(stats.zscore)

dataz.describe()
```

[69]:
```
       fixed acidity  volatile acidity   citric acid  residual sugar  \
count   1.599000e+03      1.599000e+03  1.599000e+03    1.599000e+03
mean    4.088176e-16      1.599721e-16 -8.887339e-17   -1.155354e-16
std     1.000000e+00      1.000000e+00  1.000000e+00    1.000000e+00
min    -2.136377e+00     -2.277567e+00 -1.391037e+00   -1.162333e+00
25%    -7.004996e-01     -7.696903e-01 -9.290275e-01   -4.530767e-01
50%    -2.410190e-01     -4.367545e-02 -5.634264e-02   -2.402999e-01
75%     5.056370e-01      6.264921e-01  7.650078e-01    4.340257e-02
max     4.353787e+00      5.876138e+00  3.742403e+00    9.192806e+00
```

```
          chlorides  free sulfur dioxide  total sulfur dioxide       density  \
count  1.599000e+03         1.599000e+03          1.599000e+03  1.599000e+03
mean   3.554936e-16        -4.443669e-17          3.554936e-17 -3.466062e-14
std    1.000000e+00         1.000000e+00          1.000000e+00  1.000000e+00
min   -1.603443e+00        -1.422055e+00         -1.230199e+00 -3.537625e+00
25%   -3.711129e-01        -8.484502e-01         -7.438076e-01 -6.075656e-01
50%   -1.798892e-01        -1.792441e-01         -2.574163e-01  1.759533e-03
75%    5.382858e-02         4.899619e-01          4.721707e-01  5.766445e-01
max    1.112355e+01         5.365606e+00          7.372847e+00  3.678904e+00

                 pH     sulphates       alcohol     quality
count  1.599000e+03  1.599000e+03  1.599000e+03  1599.000000
mean   2.879498e-15  6.754377e-16  8.887339e-17     5.636023
std    1.000000e+00  1.000000e+00  1.000000e+00     0.807569
min   -3.699244e+00 -1.935902e+00 -1.898325e+00     3.000000
25%   -6.549356e-01 -6.380200e-01 -8.661079e-01     5.000000
50%   -7.210449e-03 -2.250577e-01 -2.092427e-01     6.000000
75%    5.757422e-01  4.238832e-01  6.352984e-01     6.000000
max    4.526866e+00  7.916200e+00  4.201138e+00     8.000000
```

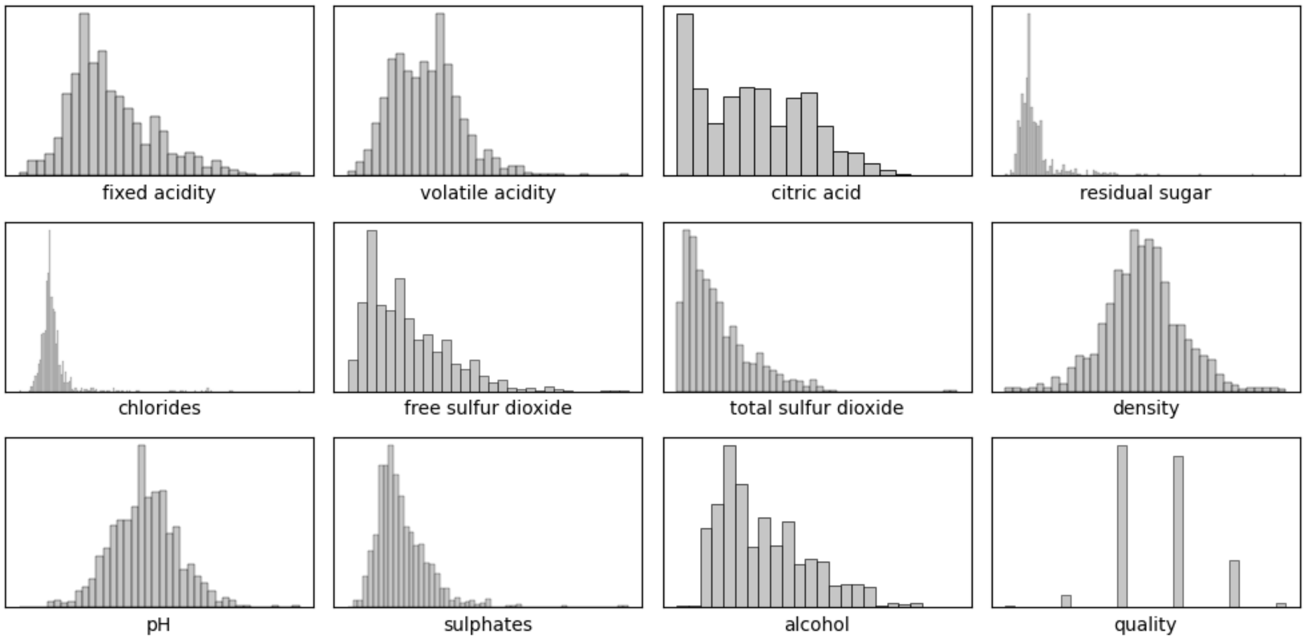
[70]:
```python
# check the plot again
fig,ax = plt.subplots(1,figsize=(13,3))
ax = sns.boxplot(data=dataz)
ax.set_xticklabels(ax.get_xticklabels(),rotation=90)

plt.tight_layout()
plt.show()
```



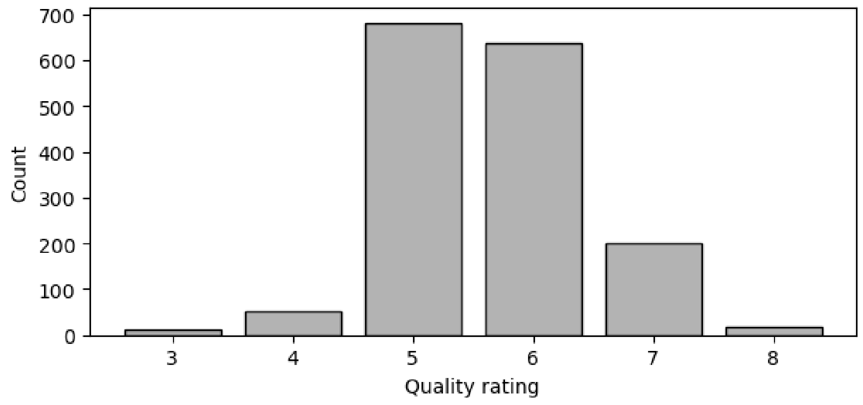
[71]:
```python
# test each variable for normality

# loop through all variables
for col in cols2zscore:

  # compute and print the test
  Stest = stats.shapiro(dataz[col])
  print(f'{col:>25}: p<{Stest.pvalue:.4f}')
```

```
          fixed acidity: p<0.0000
       volatile acidity: p<0.0000
            citric acid: p<0.0000
         residual sugar: p<0.0000
              chlorides: p<0.0000
    free sulfur dioxide: p<0.0000
   total sulfur dioxide: p<0.0000
                density: p<0.0000
                     pH: p<0.0000
              sulphates: p<0.0000
                alcohol: p<0.0000
```

[73]:
```
# visualize the histograms
_,axs = plt.subplots(3,4,figsize=(10,5))

# loop through columns and create histograms
for (idx,col),a in zip(enumerate(dataz.columns),axs.flatten()):
  sns.histplot(data=dataz,x=col,ax=a,color=(.7,.7,.7))
  a.set(xticks=[],yticks=[],ylabel='')

plt.tight_layout()
#plt.savefig('ttest_ex12.png')
plt.show()
```



[74]:
```
# binarize quality ratings into a new variable
fig = plt.figure(figsize=(7,3))

counts = dataz['quality'].value_counts()
plt.bar(list(counts.keys()),counts,color=(.7,.7,.7),edgecolor='k')
plt.xlabel('Quality rating')
plt.ylabel('Count')
plt.show()

# create a new column for binarized (boolean) quality
dataz['boolQuality'] = False
dataz['boolQuality'][dataz['quality']>5] = True
dataz[['quality','boolQuality']]
```



```
C:\Users\user\AppData\Local\Temp\ipykernel_11404\3323436096.py:12:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  dataz['boolQuality'][dataz['quality']>5] = True
```

[74]:
```
      quality  boolQuality
0           5        False
1           5        False
2           5        False
3           6         True
4           5        False
...       ...          ...
1594        5        False
1595        6         True
1596        6         True
1597        5        False
1598        6         True

[1599 rows x 2 columns]
```

29 Exercise 13

[75]:
```
# run all t-tests and store the results

# Note about this and the next code cell: You need to compute all p-values in␣
↪order
# to conduct the FDR correction. That's why I run the t-tests here and then␣
↪report
# the results in the following cell.

# initialize results matrix as a dictionary
results = {}

# loop over column
for col in cols2zscore:
  # for convenience, extract the numerical variables
  Xh = dataz[col][dataz['boolQuality']==True].values  # high rating
  Xl = dataz[col][dataz['boolQuality']==False].values # low rating

  # compute df
  s1,s2 = np.var(Xh,ddof=1),np.var(Xl,ddof=1)
  n1,n2 = len(Xh),len(Xl)
  df_num = (s1/n1 + s2/n2)**2
  df_den = s1**2/(n1**2*(n1-1)) + s2**2/(n2**2*(n2-1))

  # run the t-test and store the results in a dictionary
  tres = stats.ttest_ind(Xh,Xl,equal_var=False)
  #tres = stats.mannwhitneyu(Xh,Xl) # uncomment for Mann-Whitney U test

  results[col] = { 't' : tres.statistic,
                   'p' : tres.pvalue,
                   'df': df_num/df_den }
```

[76]:
```
# need FDR correction function
from statsmodels.stats.multitest import fdrcorrection

# bonferroni threshold
bonP = .05/len(cols2zscore)

# FDR correction (don't need p-values, only keep outcome)
fdrH = fdrcorrection([results[col]['p'] for col in cols2zscore],.05)[0]

# loop through columns and report the results!
for i,col in enumerate(cols2zscore):
  # extract values
  t  = results[col]['t']
  p  = results[col]['p']
  df = results[col]['df']

  # determine if significant
  issigB = [' ','*'][int(p<bonP)] # convert from bool to int to index
  issigF = [' ','+'][int(fdrH[i])]

  print(f'{col:>20}: t({df:.0f})={t:6.2f}, p={p:.4f}, {issigB}{issigF}')
```

```
       fixed acidity: t(1596)=  3.86, p=0.0001, *+
    volatile acidity: t(1515)=-13.48, p=0.0000, *+
         citric acid: t(1593)=  6.48, p=0.0000, *+
      residual sugar: t(1575)= -0.09, p=0.9311,
           chlorides: t(1266)= -4.29, p=0.0000, *+
 free sulfur dioxide: t(1523)= -2.46, p=0.0141,  +
total sulfur dioxide: t(1355)= -9.34, p=0.0000, *+
             density: t(1576)= -6.55, p=0.0000, *+
                  pH: t(1567)= -0.13, p=0.8962,
           sulphates: t(1495)=  8.85, p=0.0000, *+
             alcohol: t(1517)= 19.78, p=0.0000, *+
```