stats_ch12_correlation

August 5, 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 12 (correlation)

---

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.

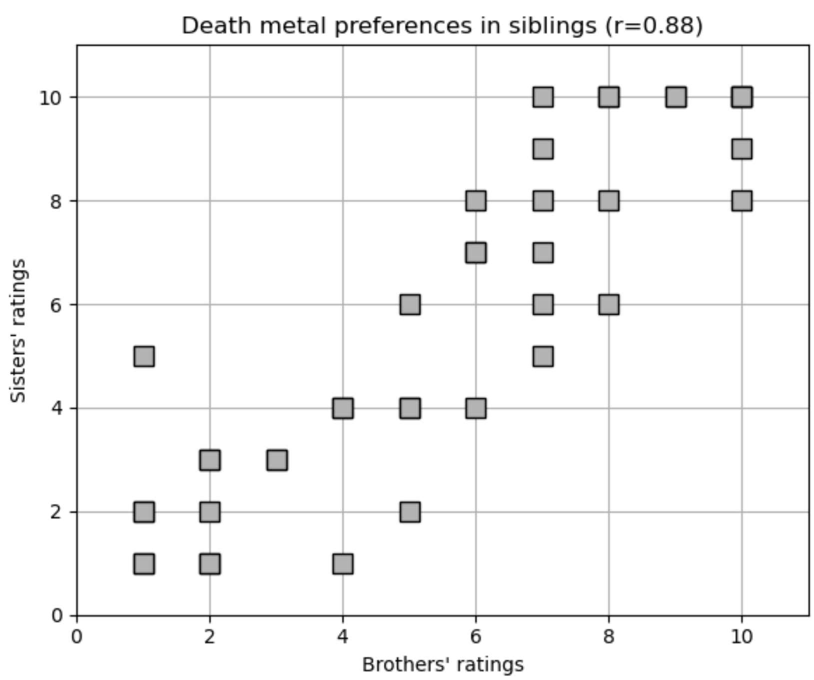
2.0.2 Solutions to all exercises are at the bottom of the notebook.

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 matplotlib.pyplot as plt

# import module from scipy (for cosine similarity)
from scipy import spatial

# pandas and seaborn for the exercises
import pandas as pd
import seaborn as sns

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

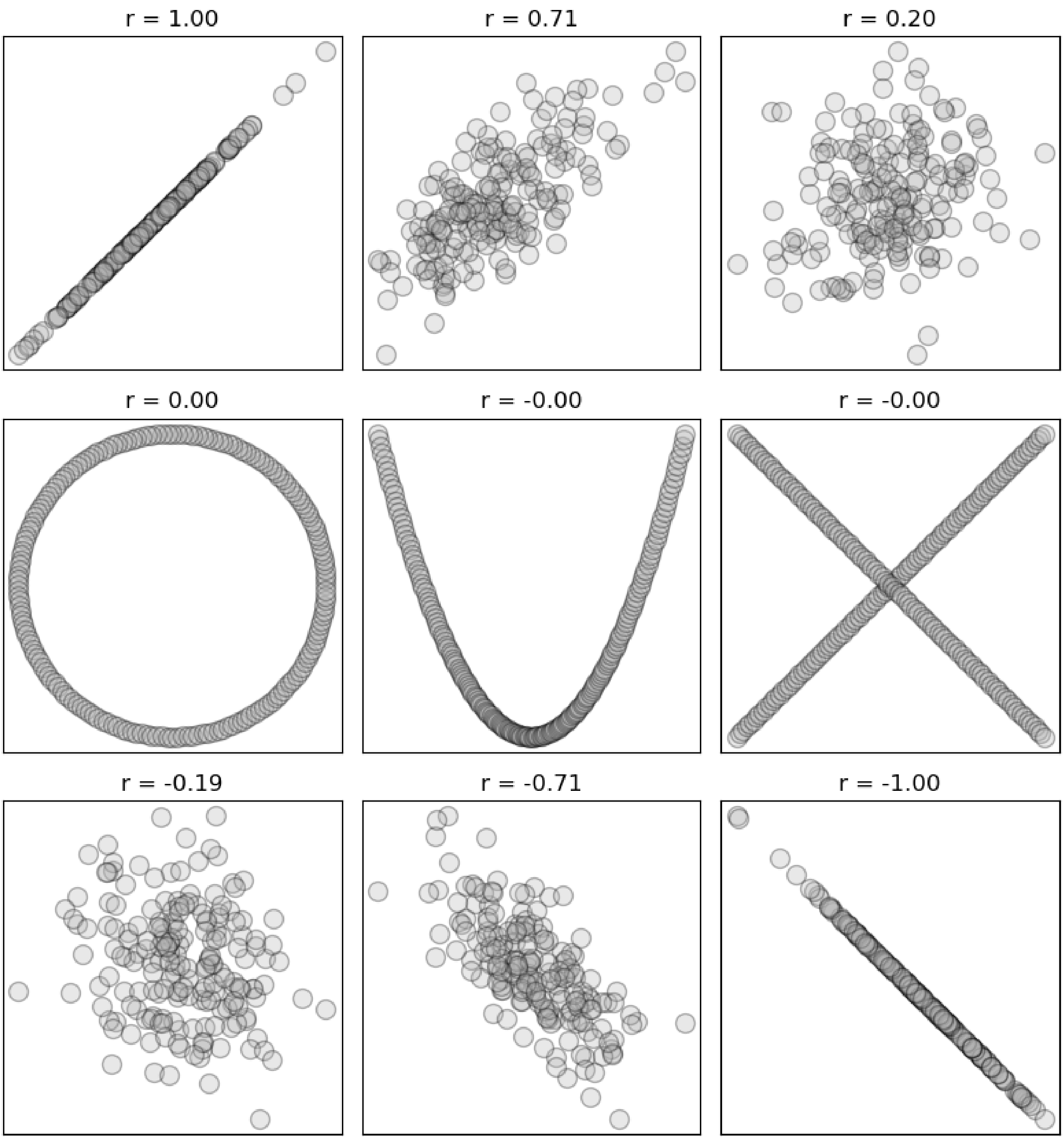
3 Figure 12.1: Example of scatter plot showing correlated data

[2]:
```
# Number of sibling pairs
num_pairs = 40

# Simulate ratings for brothers
brothers_ratings = np.random.randint(1,11,num_pairs)
# Simulate correlated ratings for sisters based on brothers' ratings
noise = np.random.normal(0,2,num_pairs)  # some random noise
sisters_ratings = brothers_ratings + noise  # sister's ratings are brother's␣
↪ratings plus some noise
# Ensure ratings are within bounds 1 and 10
sisters_ratings = np.clip(np.round(sisters_ratings),1,10)
# correlation
r = stats.pearsonr(brothers_ratings,sisters_ratings)[0]
# Create a scatter plot of the data
plt.figure(figsize=(6,5))
plt.plot(brothers_ratings, sisters_ratings,'ks',markersize=10,markerfacecolor=(.
↪7,.7,.7))
plt.title(f'Death metal preferences in siblings (r={r:.2f})',loc='center')
plt.xlabel("Brothers' ratings")
plt.ylabel("Sisters' ratings")
plt.xlim(0,11)
plt.ylim(0,11)
plt.grid(True)
plt.tight_layout()
#plt.savefig('cor_death.png')
plt.show()
```



4 Figure 12.2: Different correlation coefficients

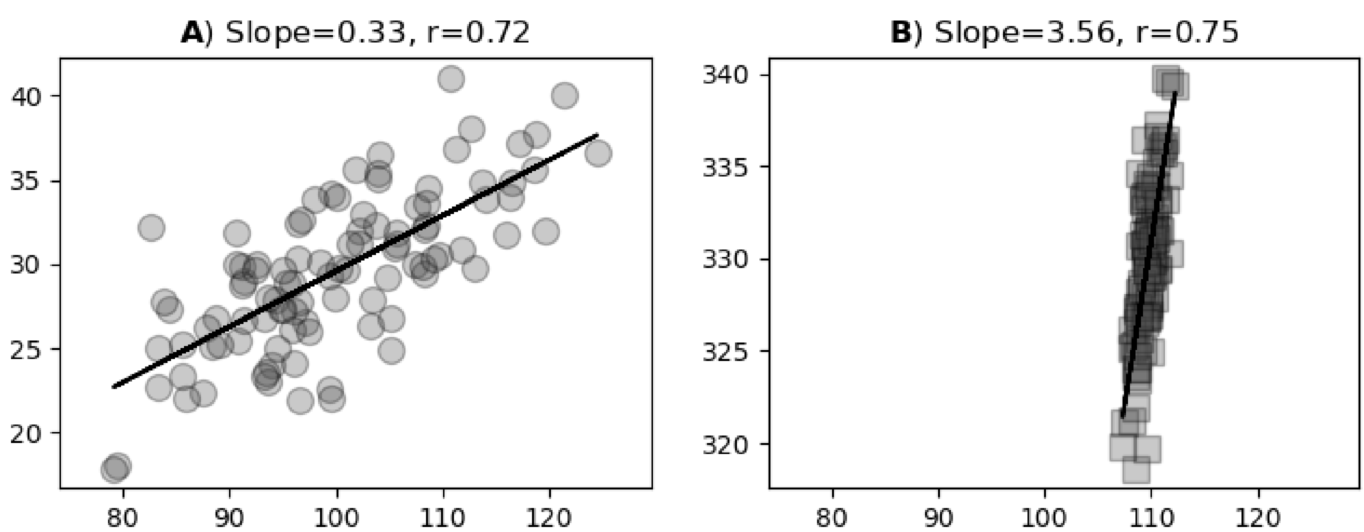
[3]:
```python
# correlation values
rs = [ 1,.7,.2,0,0,0,-.2,-.7,-1 ]

# sample size
N = 188

# start the plotting!
_,axs = plt.subplots(3,3,figsize=(8,8.5))

for r,ax,i in zip(rs,axs.flatten(),range(9)):
  # generate data
  x = np.random.randn(N)
  y = x*r + np.random.randn(N)*np.sqrt(1-r**2)

  # exceptions for r=0
  if i==3:
    x = np.cos(np.linspace(0,2*np.pi-2*np.pi/N,N))
    y = np.sin(np.linspace(0,2*np.pi-2*np.pi/N,N))
  elif i==4:
    x = np.linspace(-2,2,N)
    y = x**2
  elif i==5:
    x = np.linspace(-2,2,N//2)
    y = np.concatenate((x,-x),0)
    x = np.concatenate((x,x),0)

  # empirical correlation
  rho = np.corrcoef(x,y)[0,1]

  # plot
  ax.plot(x,y,'ko',markersize=10,markerfacecolor=(.7,.7,.7),alpha=.3)
  ax.set(xticks=[],yticks=[])
  ax.set_title(f'r = {rho:.2f}',loc='center')

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



5 Figure 12.3: Same correlation, different slopes

[10]:
```
N = 100

# Dataset 1
x1 = np.random.normal(100,10,N)
y1 = .3*x1 + np.random.randn(N)*3
slope1,intercept1,r1,_,_ = stats.linregress(x1,y1)
```

```
# Dataset 2
x2 = np.random.normal(10,1,N) + np.mean(x1)
y2 = 3*x2 + np.random.randn(N)*3
slope2,intercept2,r2,_,_ = stats.linregress(x2,y2)

# x-axis limits
xmin,xmax = np.min(x1)-5,np.max(x1)+5

# Plot datasets and their regression lines
_,axs = plt.subplots(1,2,figsize=(9,3))

axs[0].plot(x1,y1,'ko',markersize=10,markerfacecolor=(.3,.3,.3),alpha=.3)
axs[0].plot(x1,intercept1 + slope1*x1,'k')
axs[0].set_title(fr'$\bf{{A}}$) Slope={slope1:.2f}, r={r1:.2f}')
axs[0].set(xlim=[xmin,xmax])

axs[1].plot(x2,y2,'ks',markersize=10,markerfacecolor=(.3,.3,.3),alpha=.3)
axs[1].plot(x2, intercept2 + slope2*x2,'k')
axs[1].set_title(fr'$\bf{{B}}$) Slope={slope2:.2f}, r={r2:.2f}')
axs[1].set(xlim=[xmin,xmax])

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




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

6 numpy vs scipy

[6]:
```
# vector input
x = np.random.randn(50)
y = x + np.random.randn(len(x))/2

usingScipy = stats.pearsonr(x,y)
usingNumpy = np.corrcoef(x,y)

print('scipy.stats.pearsonr:')
print(usingScipy)

print(' ')
print('numpy.corrcoef:')
print(usingNumpy)
```

```
scipy.stats.pearsonr:
PearsonRResult(statistic=0.8767490760934157, pvalue=7.150206750458781e-17)

numpy.corrcoef:
[[1.         0.87674908]
 [0.87674908 1.        ]]
```

[7]:
```
# matrix input (features by observations)
X = np.vstack((x[None,:],y[None,:]))

# usingScipy = stats.pearsonr(X) ## gives an error!
usingNumpy = np.corrcoef(X)

print('numpy.corrcoef:')
print(usingNumpy)
```

```
numpy.corrcoef:
[[1.         0.87674908]
 [0.87674908 1.        ]]
```

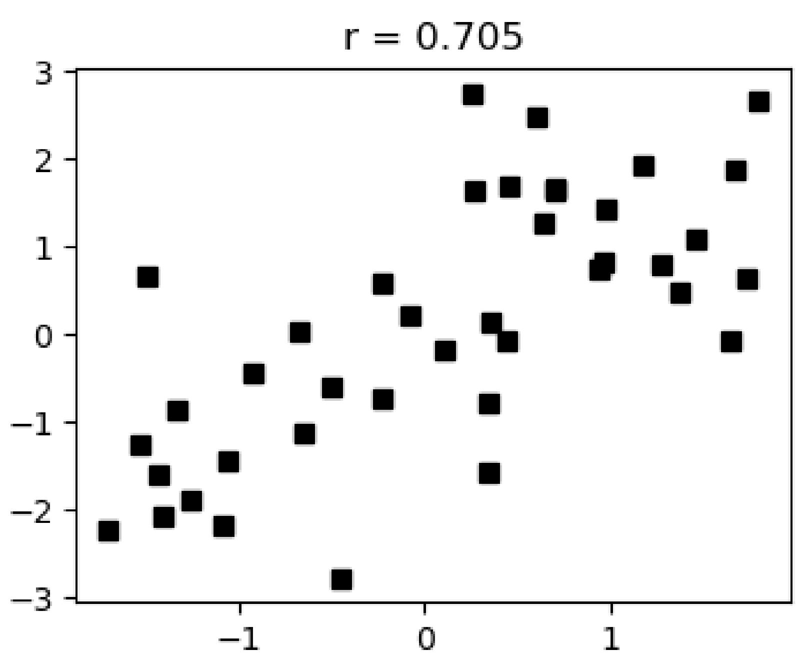
[8]:
```
# to get a matrix of R and p values:

R = np.zeros((2,2))
P = np.zeros((2,2))

# Calculate Pearson correlation for each pair of variables
for i in range(2):
  for j in range(2):
    R[i,j], P[i,j] = stats.pearsonr(X[i,:],X[j,:])

print('Correlation matrix:')
print(R)
```

```
print('')
print('Associated p-values:')
print(P)

# Note about this code cell: you don't actually need to compute all matrix␣
↪elements;
# you can compute only the upper-triangle and then copy the results to the lower␣
↪triangle.
```

```
Correlation matrix:
[[1.         0.87674908]
 [0.87674908 1.        ]]

Associated p-values:
[[0.00000000e+00 7.15020675e-17]
 [7.15020675e-17 0.00000000e+00]]
```

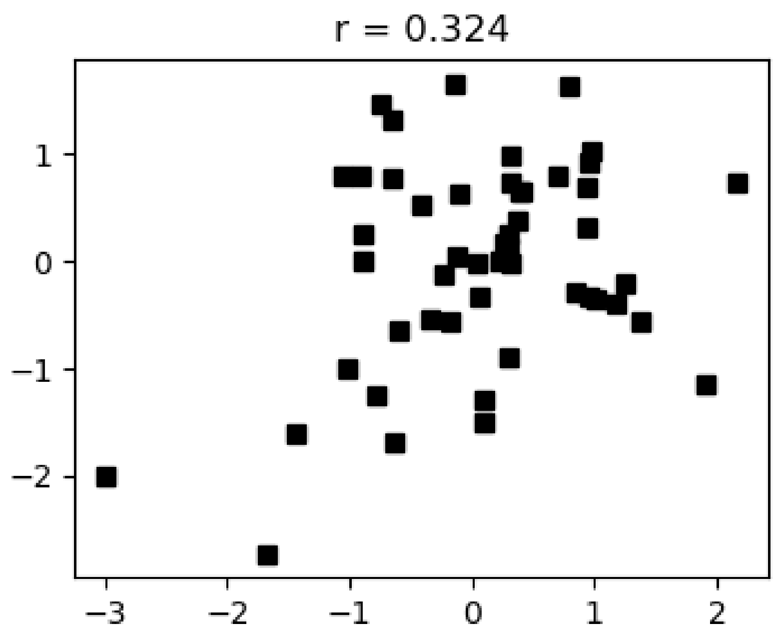
7 Creating correlated data

[11]:
```
# Method 1 (quick&dirty but effective)
x = np.random.randn(40)
y = x + np.random.randn(len(x))
r = stats.pearsonr(x,y).statistic

plt.figure(figsize=(4,3))
plt.plot(x,y,'ks')
plt.title(f'r = {r:.3f}',loc='center')
plt.show()
```



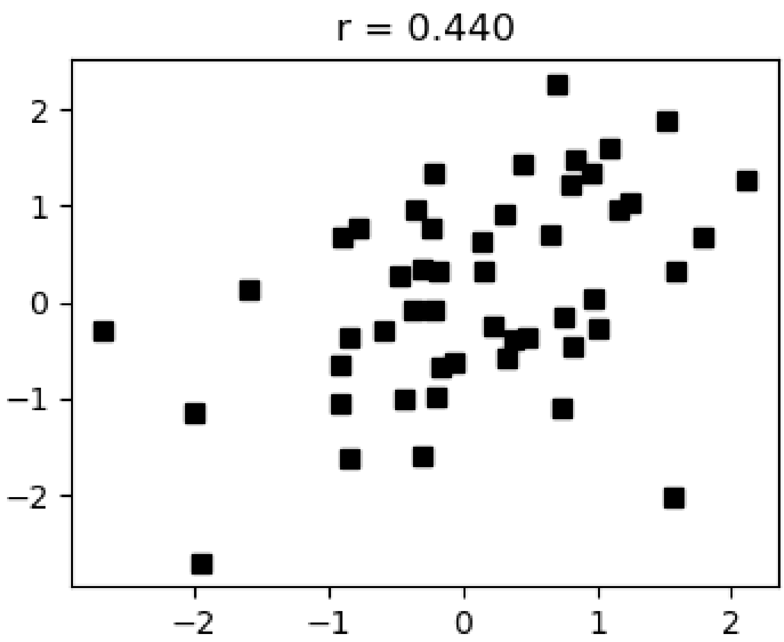
[13]:

```
# method 2 (more control over the correlation)
r = .4
x = np.random.randn(50)
y = np.random.randn(len(x))
y = x*r + y*np.sqrt(1-r**2)

rr = stats.pearsonr(x,y).statistic

plt.figure(figsize=(4,3))
plt.plot(x,y,'ks')
plt.title(f'r = {rr:.3f}',loc='center')
plt.show()
```



[14]:

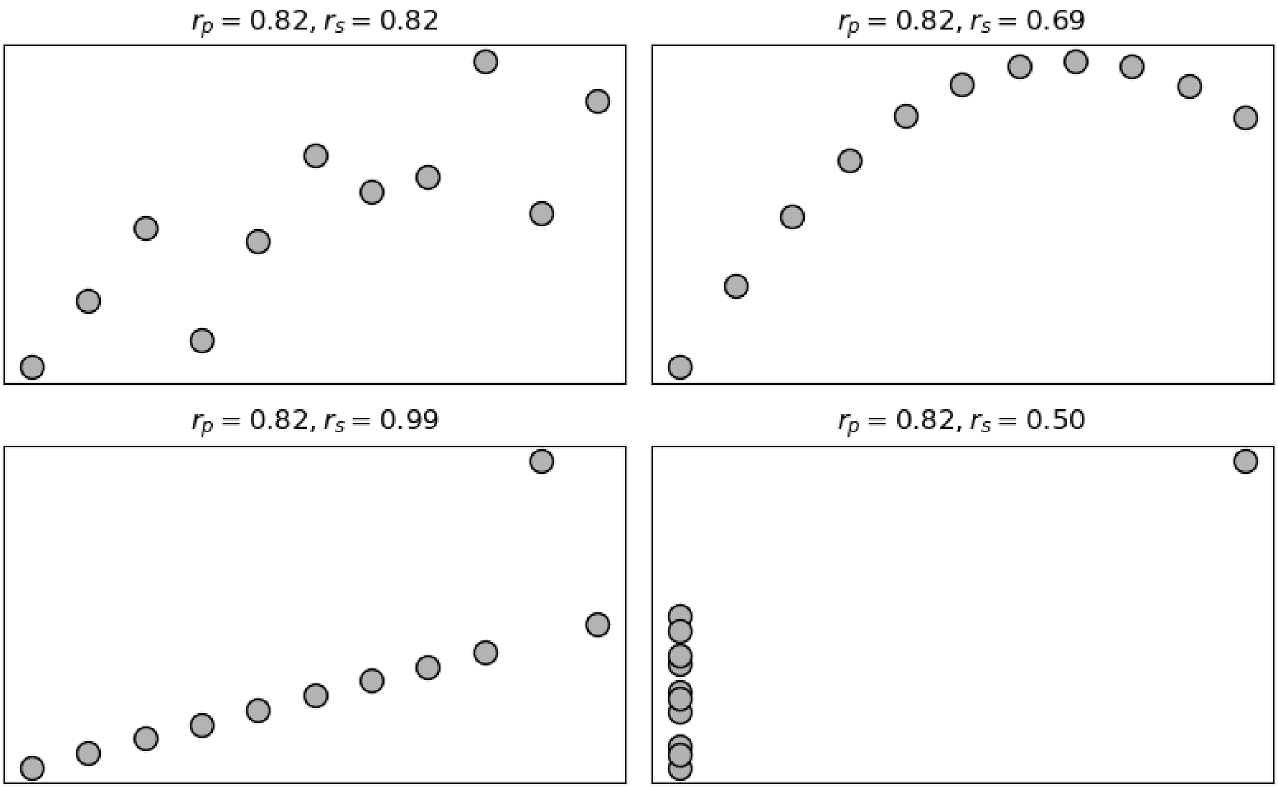
```
# method 3: multivariate
C = np.array([ [1,.4],[.4,1] ])
means = np.zeros(2)

X = np.random.multivariate_normal(means,C,50)
r = np.corrcoef(X.T)

plt.figure(figsize=(4,3))
plt.plot(X[:,0],X[:,1],'ks')
plt.title(f'r = {r[0,1]:.3f}',loc='center')
plt.show()
```



8 Figure 12.6: Anscobe's quartet

[15]:
```
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, ]
    ])

# plot data and correlations
fig,ax = plt.subplots(2,2,figsize=(8,5))
ax = ax.ravel()

for i in range(4):
  # plot the points
  ax[i].plot(anscombe[:,i*2],anscombe[:
↪,i*2+1],'ko',markersize=10,markerfacecolor=(.7,.7,.7))

  # compute the corrs
```

```
    corr_p = stats.pearsonr(anscombe[:,i*2],anscombe[:,i*2+1])[0]
    corr_s = stats.spearmanr(anscombe[:,i*2],anscombe[:,i*2+1])[0]

    # update the axis
    ax[i].set(xticks=[],yticks=[])
    ax[i].set_title(f'$r_p = {corr_p:.2f},  r_s = {corr_s:.2f}$',loc='center')

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



9 Toy covariance example

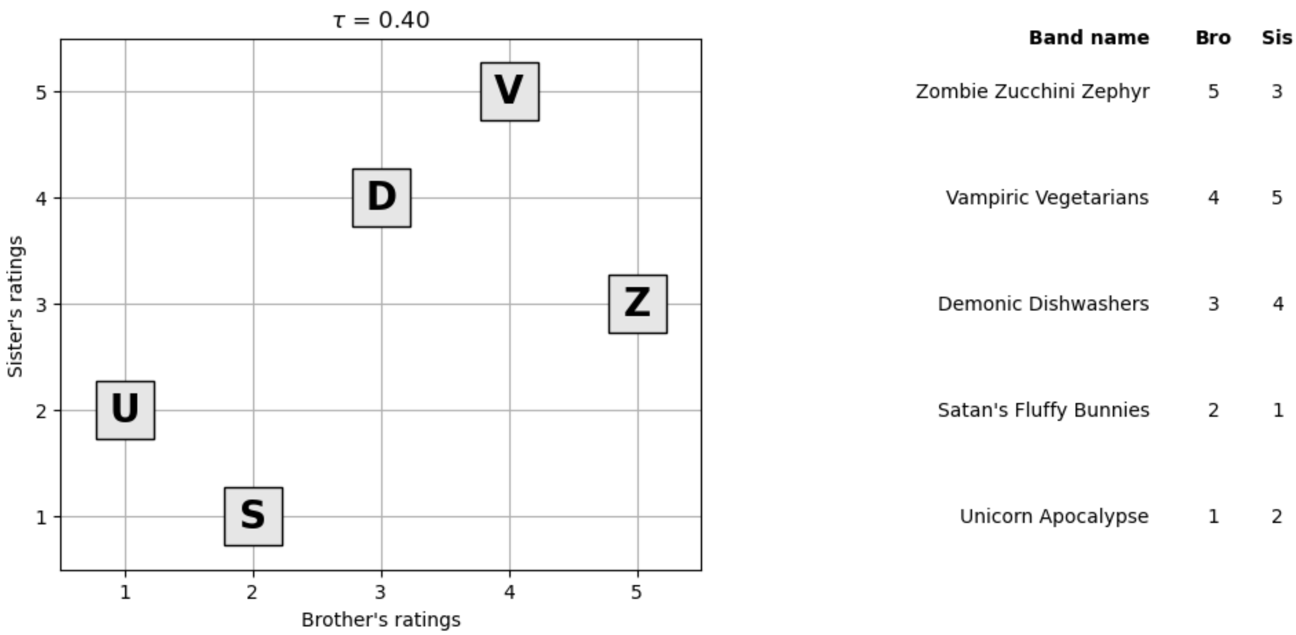
[16]:
```
# raw scores
h = np.array([74,63,58,70])
s = np.array([ 4, 7, 2, 9])
N = len(h)

# demeaned
hd = h-np.mean(h)
sd = s-np.mean(s)

cov = np.sum(hd*sd) / (N-1)
cov, hd*sd
```

[16]:
```
(8.5, array([-11.625,  -4.875,  28.875,  13.125]))
```

10 Figure 12.7: Kandall tau

[20]:
```
# The data
bro = np.array([ 1,2,3,4,5 ])
sis = np.array([ 2,1,4,5,3 ])

# the correlation
k = stats.kendalltau(bro,sis)

# band names (lol)
bands = [ 'Unicorn Apocalypse',
          "Satan's Fluffy Bunnies",
          'Demonic Dishwashers',
          'Vampiric Vegetarians',
          'Zombie Zucchini Zephyr' ]

# the plot
_,ax = plt.subplots(1,figsize=(6,5))
ax.plot(bro,sis,'ks',markersize=30,markerfacecolor=(.9,.9,.9))
ax.set(xlabel="Brother's ratings",ylabel="Sister's ratings",
       yticks=range(1,6),xticks=range(1,6),
       xlim=[.5,5.5],ylim=[.5,5.5])
ax.grid()
ax.set_title(fr'$\tau$ = {k[0]:.2f}',loc='center')

# band names
for i in range(len(bro)):

  # band names in plot markers
  ax.text(bro[i],sis[i],bands[i][0],size=20,
```

```
           weight='bold',va='center',ha='center')
  # raw data
  ax.text(9,i+1,bands[i],ha='right',va='center')
  ax.text(9.5,i+1,bro[i],ha='center',va='center')
  ax.text(10,i+1,sis[i],ha='center',va='center')

# column labels
ax.text(9,5.5,'Band name',ha='right',va='center',weight='bold')
ax.text(9.5,5.5,'Bro',ha='center',va='center',weight='bold')
ax.text(10,5.5,'Sis',ha='center',va='center',weight='bold')

plt.figure(figsize=(10,3))
#plt.tight_layout()
#plt.savefig('cor_kendall.png')
plt.show()
```

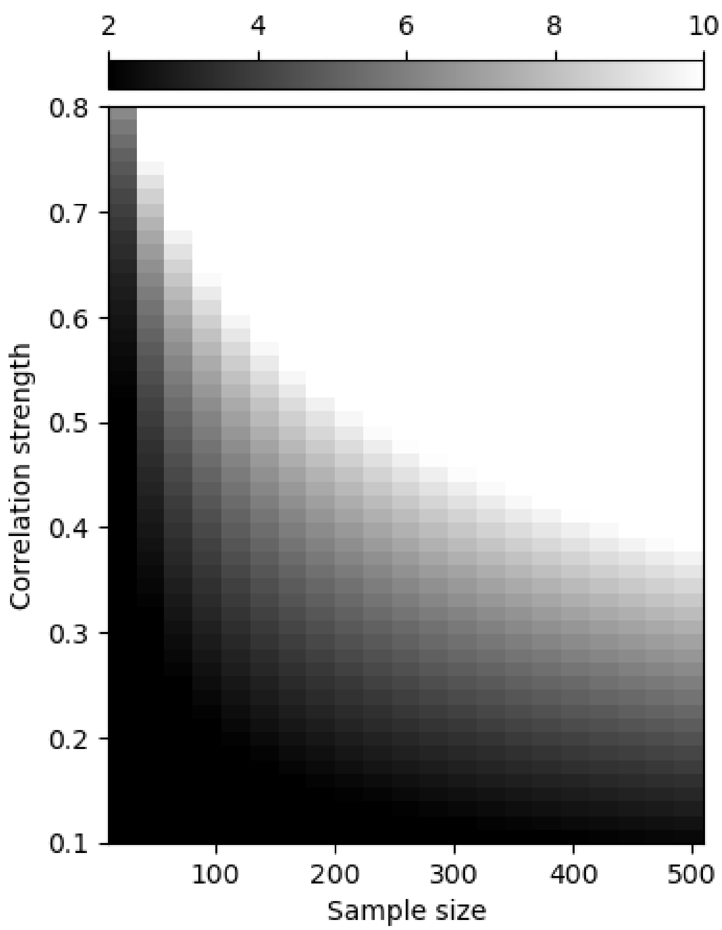



```
<Figure size 1000x300 with 0 Axes>
```

11 Figure 12.8: Statistical significance of r based on n

[23]:
```
# simulation ranges
rs = np.linspace(.1,.8,53) # r values
ns = np.arange(10,511,step=25) # sample sizes

# compute the matrix of t-values
num = rs[:,None]*np.sqrt(ns-2)
den = 1-rs[:,None]**2
tmat = num/den
```

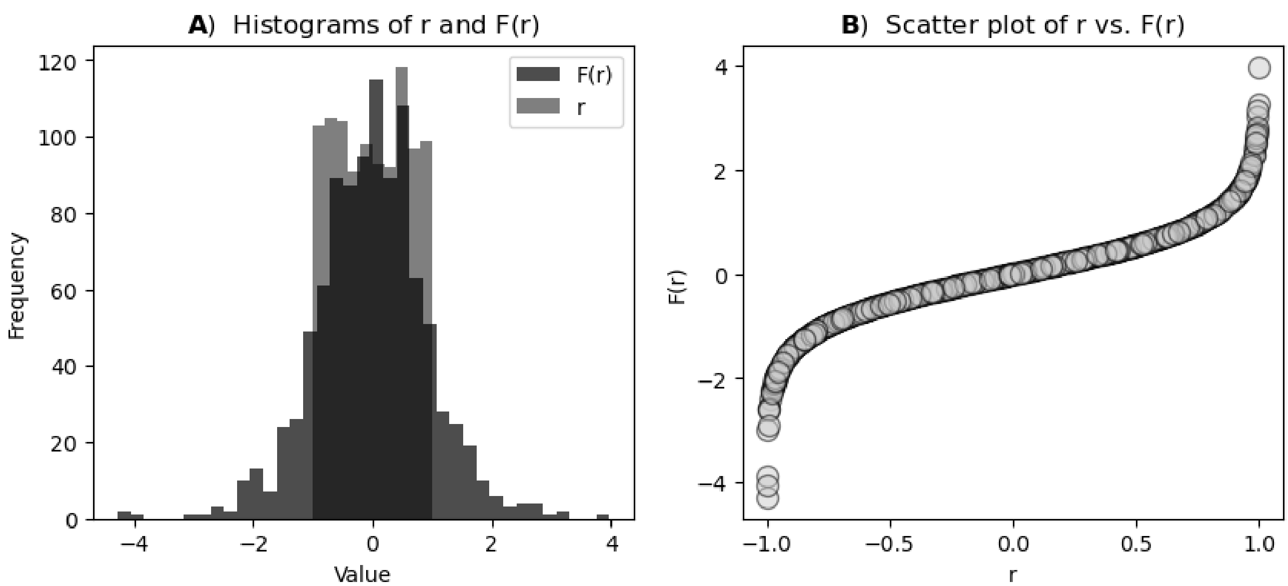
```python
## show the matrix!
fig,ax = plt.subplots(1,figsize=(4,6))

cax = ax.imshow(tmat,vmin=2,vmax=10,aspect='auto',cmap='gray',
           extent=[ns[0],ns[-1],rs[0],rs[-1]],origin='lower')
ax.set(xlabel='Sample size',ylabel='Correlation strength')

# and make it look a bit nicer
fig.colorbar(cax,orientation='horizontal',pad=.02,ax=ax,location='top')
ax.spines[['right','top']].set_visible(True)

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




```
<Figure size 300x300 with 0 Axes>
```

12 Figure 12.9: Fisher-z transform on uniform data

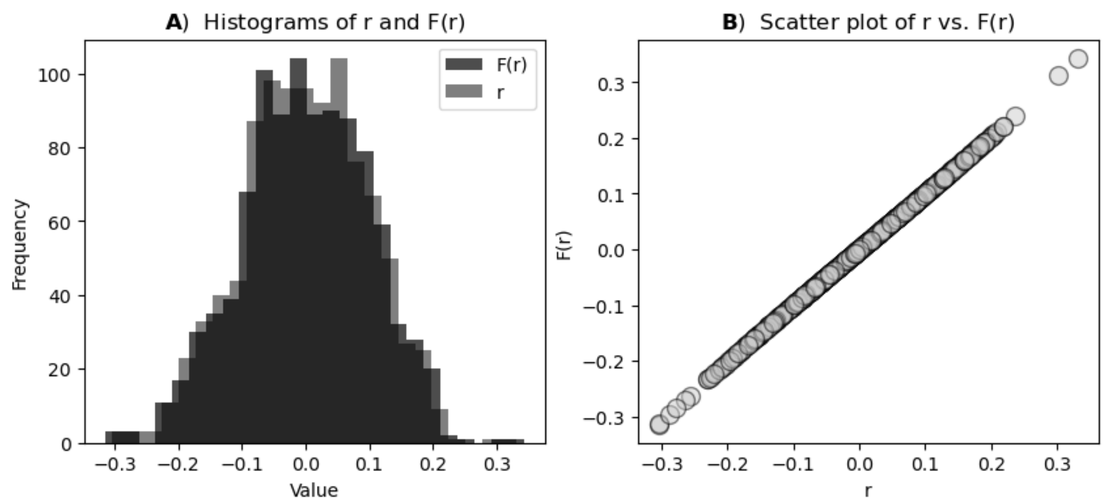
[25]:
```
# "correlation" data and its Fisher transform
r = np.random.uniform(-1,1,size=1000)
fish_r = np.arctanh(r)

# now for plotting
fig,axs = plt.subplots(1,2,figsize=(10,4))

# histograms
axs[0].hist(fish_r, bins='fd', color=(.3,.3,.3),alpha=1, label='F(r)')
axs[0].hist(r,bins='fd', color='k', alpha=.5, label='r')
axs[0].legend(loc='upper right')
axs[0].set_xlabel('Value')
axs[0].set_ylabel('Frequency')
axs[0].set_title(r'$\bf{A}$)  Histograms of r and F(r)')

# scatter plot to visualize the transform
axs[1].plot(r,fish_r,'ko',markersize=10,alpha=.5,markerfacecolor=(.8,.8,.8))
axs[1].set_xlabel('r')
axs[1].set_ylabel('F(r)')
axs[1].set_title(r'$\bf{B}$)  Scatter plot of r vs. F(r)')

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




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

13 Figure 12.10: Fisher-z transform on numbers close to zero

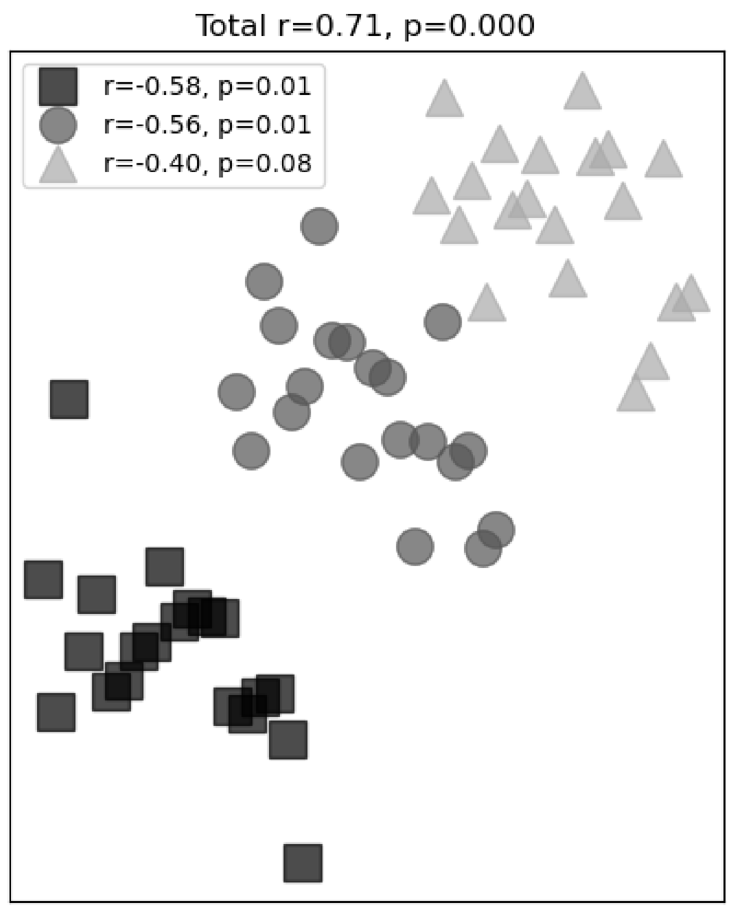
[27]:
```
# Same as above but with simulated H0 coefficients

# random data
X = np.random.randn(100,45)
# compute the correlation matrix and extract the unique r values
R = np.corrcoef(X.T)
utri = np.triu(R,k=1)
r = utri[utri!=0]
# fisher transform
fish_r = np.arctanh(r)
# now for plotting
fig,axs = plt.subplots(1,2,figsize=(10,4))
# histograms
axs[0].hist(fish_r, bins='fd', color=(.3,.3,.3),alpha=1, label='F(r)')
axs[0].hist(r,bins='fd', color='k', alpha=.5, label='r')
axs[0].legend(loc='upper right')
axs[0].set_xlabel('Value')
axs[0].set_ylabel('Frequency')
axs[0].set_title(r'$\bf{A}$)  Histograms of r and F(r)')
# scatter plot to visualize the transform
axs[1].plot(r,fish_r,'ko',markersize=10,alpha=.5,markerfacecolor=(.8,.8,.8))
axs[1].set_xlabel('r')
axs[1].set_ylabel('F(r)')
axs[1].set_title(r'$\bf{B}$)  Scatter plot of r vs. F(r)')
plt.figure(figsize=(9,3))
plt.tight_layout()
#plt.savefig('cor_fisherReal.png')
plt.show()
```




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

14 Figure 12.11: Subgroups paradox

[29]:
```
# initializations
n = 20 # sample points per group
offsets = [2, 3.5, 5] # mean offsets

allx = np.array([])
ally = np.array([])

s = 'so^' # marker shapes

# generate and plot data
plt.figure(figsize=(5,6))
for datai in range(3):
  # generate data
  x = np.linspace(offsets[datai]-1,offsets[datai]+1,n)
  y = -x/3 + np.mean(x) + np.random.randn(n)/3

  # subgroup correlation
  r,p = stats.pearsonr(x,y)

  # plot
  plt.plot(x,y,s[datai],color=(datai/3,datai/3,datai/3),
           markersize=14,alpha=.7,label=f'r={r:.2f}, p={p:.2f}')
  # gather the data into one array
  allx = np.append(allx,x)
  ally = np.append(ally,y)

# % now correlate the groups
[r,p] = stats.pearsonr(allx,ally)
plt.title(f'Total r={r:.2f}, p={p:.3f}',loc='center')

plt.xticks([])
plt.yticks([])
plt.legend()

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




```
<Figure size 300x300 with 0 Axes>
```

15 Cosine similarity

[30]:

```
# variables
x = np.array([ 1,2,3,4 ])
y = np.array([ 1,2,3,4 ])
z = np.array([ 101,102,103,104])

# correlations
r_xy = np.corrcoef(x,y)[0,1]
r_xz = np.corrcoef(x,z)[0,1]

# cosine similarities
c_xy = 1-spatial.distance.cosine(x,y)
c_xz = 1-spatial.distance.cosine(x,z)
```

```
# print out the results
print(f'r_xy: {r_xy:.3f}')
print(f'c_xy: {c_xy:.3f}')
print('')
print(f'r_xz: {r_xz:.3f}')
print(f'c_xz: {c_xz:.3f}')
```

```
r_xy: 1.000
c_xy: 1.000

r_xz: 1.000
c_xz: 0.917
```

16 Exercise 1

[31]:
```
# two random correlated variables
v = np.random.randn(10)
w = v + np.random.randn(len(v))

### correlation using mean-centered dot products
# mean-center
vm = v-np.mean(v)
wm = w-np.mean(w)

# dot products
r_me = np.dot(vm,wm) / np.sqrt(np.dot(vm,vm)*np.dot(wm,wm))

### correlation using numpy
r_np = np.corrcoef(v,w)[0,1]

# print results
print(f'r from np.corr: {r_np:.3f}')
print(f'r from np.dot : {r_me:.3f}')
```

```
r from np.corr: 0.710
r from np.dot : 0.710
```

17 Exercise 2

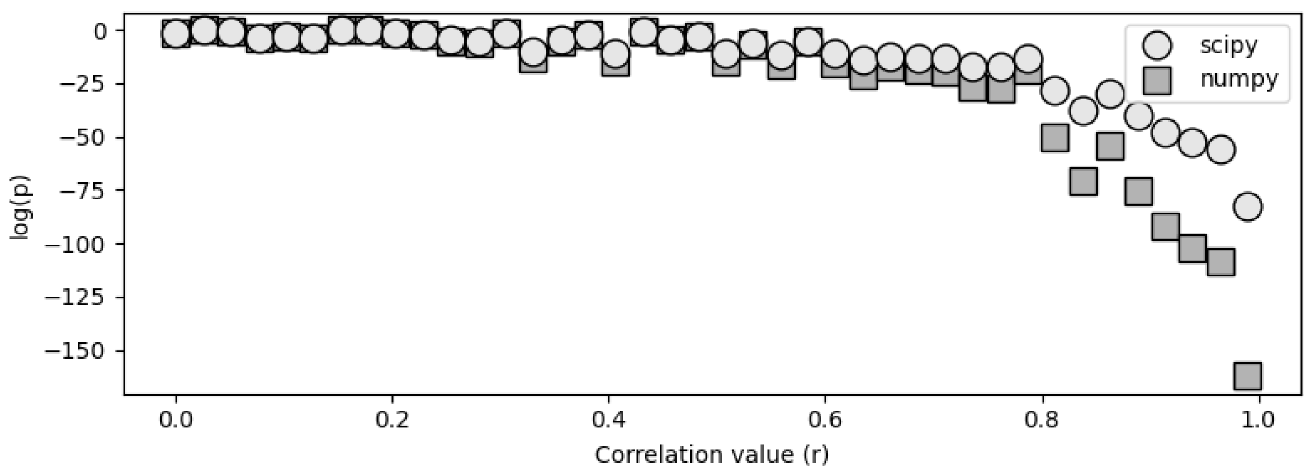
[32]:
```
# create random data
N = 43
r = .4
x = np.random.randn(N)
y = np.random.randn(N)
y = x*r + y*np.sqrt(1-r**2)

# r,p from numpy
```

```
r_np = np.corrcoef(x,y)[0,1]
t = r_np*np.sqrt(N-2) / (1-r_np**2)
p_np = stats.t.sf(np.abs(t),N-2) * 2  # times 2 for a two-sided test

# r,p from scipy
r_sp,p_sp = stats.pearsonr(x,y)

# print correlation values
print(f'r (p) from numpy: {r_np:.4f} ({p_np:.4f})')
print(f'r (p) from scipy: {r_sp:.4f} ({p_sp:.4f})')
```

```
r (p) from numpy: 0.3994 (0.0041)
r (p) from scipy: 0.3994 (0.0080)
```

[33]:
```
# now in a loop

# I wrote a function to output the two p-values,
# which makes the experiment code below easier to read.
def getpvals(x,y):

  # r,p from numpy
  r_np = np.corrcoef(x,y)[0,1]
  t = r_np*np.sqrt(len(x)-2) / (1-r_np**2)
  p_np = stats.t.sf(np.abs(t),len(x)-2) * 2  # times 2 for a two-sided test

  # r,p from scipy
  r_sp,p_sp = stats.pearsonr(x,y)

  return p_np,p_sp

# range of correlation values
rvals = np.linspace(0,.99,40)

# results matrix
pvalues = np.zeros((len(rvals),2))

## run the experiment
for ri in range(len(rvals)):

  # create the data
  x = np.random.randn(44)
  y = np.random.randn(44)
  y = x*rvals[ri] + y*np.sqrt(1-rvals[ri]**2)

  # get the two p-values
  pvalues[ri,:] = getpvals(x,y)
```

```
# plot
plt.figure(figsize=(8,3))
plt.plot(rvals,np.log(pvalues[:,1]),'ko',markersize=12,markerfacecolor=(.9,.9,.
↪9),label='scipy')
plt.plot(rvals,np.log(pvalues[:,0]),'ks',markersize=12,markerfacecolor=(.7,.7,.
↪7),label='numpy',zorder=-1)
plt.legend()
plt.xlabel('Correlation value (r)')
plt.ylabel('log(p)')

plt.tight_layout()
#plt.savefig('cor_ex2.png')
plt.show()
```

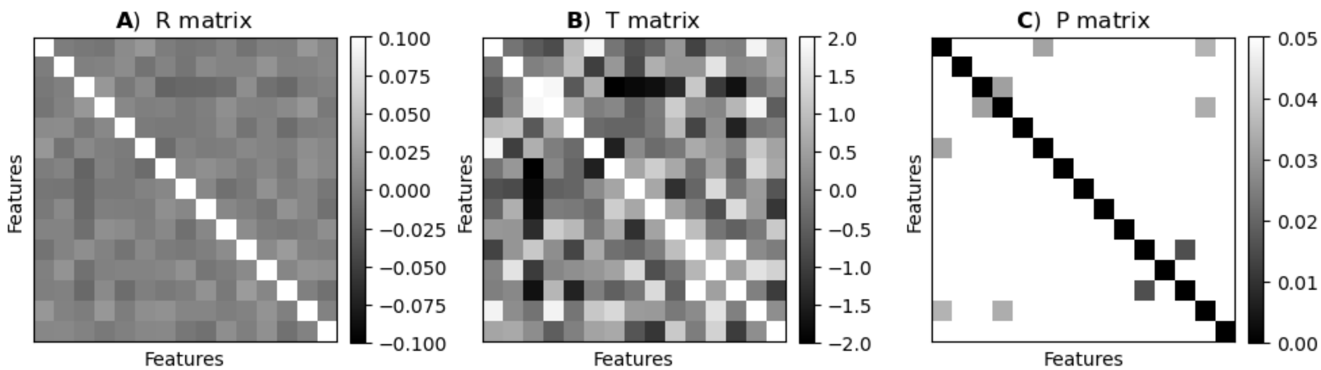


18 Exercise 3

[34]:
```
# matrix of p-values

N = 10000 # observations
M = 15 # features

# data matrix
X = np.random.randn(N,M)

# correlation matrix
R = np.corrcoef(X.T)

# confirm that it's the right shape
print(f'Correlation matrix shape: {R.shape}')
```

```
Correlation matrix shape: (15, 15)
```

[35]:
```
# compute the t-values
Tnum = R*np.sqrt(N-2)
Tden = 1-R**2 + np.finfo(float).eps # adding a tiny number to avoid n/0

T = Tnum / Tden

# compute the p-values
P = stats.t.sf(T,N-2)

# visualize all matrices
fig,axs = plt.subplots(1,3,figsize=(10,5))

cax = axs[0].imshow(R,vmin=-.1,vmax=.1,cmap='gray')
axs[0].set_title(r'$\bf{A}$)  R matrix')
c = fig.colorbar(cax,fraction=.046,pad=.04); c.ax.tick_params(labelsize=10)

cax = axs[1].imshow(T,vmin=-2,vmax=2,cmap='gray')
axs[1].set_title(r'$\bf{B}$)  T matrix')
c = fig.colorbar(cax,fraction=.046,pad=.04); c.ax.tick_params(labelsize=10)

cax = axs[2].imshow(P,vmin=0,vmax=.05,cmap='gray')
axs[2].set_title(r'$\bf{C}$)  P matrix')
c = fig.colorbar(cax,fraction=.046,pad=.04); c.ax.tick_params(labelsize=10)

# properties common to all axes
for a in axs:
  a.set(xticks=[],yticks=[],xlabel='Features',ylabel='Features')
  a.spines[['right','top']].set_visible(True)

plt.tight_layout()
#plt.savefig('cor_ex3.png')
plt.show()
```



19 Exercise 4

[36]:
```
# create the sigma matrix
Sigma = np.std(X,ddof=1,axis=0)
Sigma = np.diag(Sigma)

# compute C from R
C_me = Sigma@R@Sigma # from formula
C_np = np.cov(X.T,ddof=1) # from numpy

# check for equality (these should all be True)
(C_np-C_me)
```

[36]:
```
array([[-4.21884749e-15,  5.85469173e-18,  2.25514052e-17,
         8.67361738e-18, -1.56125113e-17, -3.12250226e-17,
         6.93889390e-18,  1.47451495e-17,  1.38777878e-17,
        -9.54097912e-18,  2.42861287e-17, -2.16840434e-19,
         2.16840434e-19, -1.00613962e-16, -1.73472348e-18],
       [ 5.63785130e-18, -2.33146835e-15, -1.89735380e-19,
        -6.50521303e-19, -1.04083409e-17,  6.93889390e-18,
        -1.17093835e-17,  1.04083409e-17, -1.90819582e-17,
        -6.07153217e-18, -5.63785130e-18,  1.04083409e-17,
        -3.03576608e-18, -9.54097912e-18,  3.46944695e-18],
       [ 2.16840434e-17, -1.76182853e-19, -3.55271368e-15,
        -2.08166817e-17,  9.54097912e-18, -8.67361738e-18,
         7.97972799e-17,  3.12250226e-17,  5.55111512e-17,
         2.94902991e-17, -2.77555756e-17,  1.73472348e-18,
         4.51028104e-17,  1.04083409e-17,  0.00000000e+00],
       [ 8.67361738e-18, -6.50521303e-19, -1.73472348e-17,
         1.77635684e-15,  6.93889390e-18, -3.79470760e-19,
        -1.30104261e-18, -5.20417043e-18,  3.25260652e-19,
         5.20417043e-18,  8.67361738e-19,  2.81892565e-18,
         0.00000000e+00, -4.85722573e-17, -1.38777878e-17],
         ...........
         ...........

       [-1.04083409e-16, -9.54097912e-18,  1.08420217e-17,
        -4.85722573e-17,  5.63785130e-18, -3.29597460e-17,
        -7.45931095e-17, -1.56125113e-17, -1.99493200e-17,
         3.79470760e-18, -7.37257477e-18, -2.77555756e-17,
        -5.01443505e-19, -6.99440506e-15, -4.33680869e-18],
       [-2.60208521e-18,  4.33680869e-18,  0.00000000e+00,
        -1.38777878e-17,  1.35525272e-19,  1.38777878e-17,
        -1.73472348e-18, -1.21430643e-17, -5.20417043e-18,
         1.56125113e-17, -4.87890978e-19,  4.51028104e-17,
        -1.04083409e-17, -4.33680869e-18,  3.55271368e-15]])
```

[37]:
```python
# Now compute R from C
invSigma = 1/np.std(X,ddof=1,axis=0)
invSigma = np.diag(Sigma)

R_me = invSigma@C_np@invSigma

# check for equality (these should all be True)
(R-R_me) < 1e-16
```

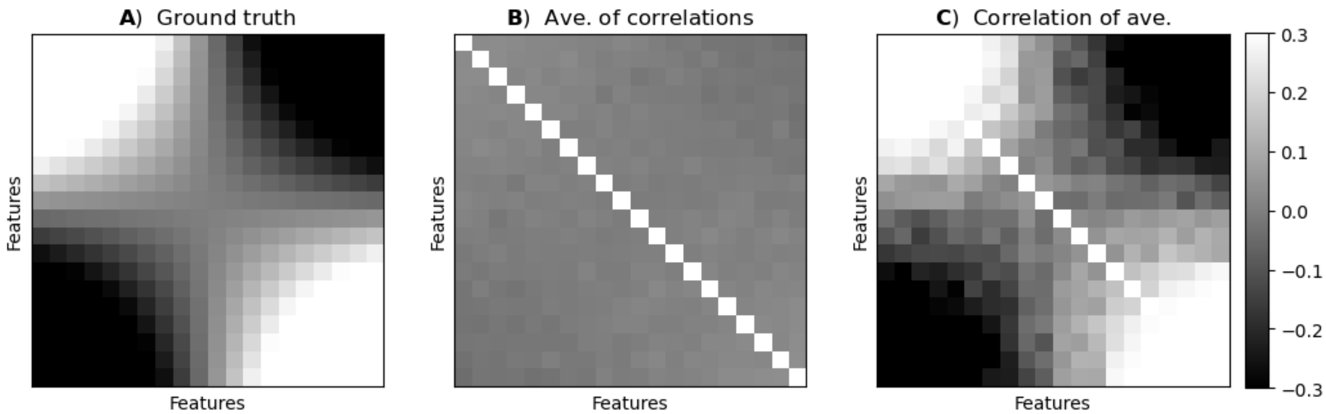
[37]:
```
array([[ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True]])
```

20 Exercise 5

[38]:
```python
# simulation parameters

N = 1000  # observations
M =   20  # features
```

```python
# number of repetitions
numReps = 30

# initialize data lists
alldata = np.zeros((M,N))
corrmats = np.zeros((M,M,numReps+1))

# "pure" data
covars = np.linspace(-1,1,M)[:,None]
dataOG = np.random.randn(N) * covars


# random noise in each repetition
for idx in range(numReps):
  # this run's data
  thisdata = dataOG + np.random.randn(M,N)*5

  # its correlation matrix
  corrmats[:,:,idx] = np.corrcoef(thisdata)

  # sum the data
  alldata += thisdata

# correlation of data average
corrmats[:,:,-1] = np.corrcoef(alldata)

### plotting
fig,axs = plt.subplots(1,3,figsize=(12,5))

axs[0].imshow(covars@covars.T,vmin=-.3,vmax=.3,cmap='gray')
axs[0].set_title(r'$\bf{A}$)  Ground truth')

axs[1].imshow(np.mean(corrmats[:,:,:-1],axis=2),vmin=-.3,vmax=.3,cmap='gray')
axs[1].set_title(r'$\bf{B}$)  Ave. of correlations')

cax = axs[2].imshow(corrmats[:,:,-1],vmin=-.3,vmax=.3,cmap='gray')
axs[2].set_title(r'$\bf{C}$)  Correlation of ave.')

cbar_ax = fig.add_axes([.91,.22,.015,.55])
cbar = plt.colorbar(cax,cax=cbar_ax)

# properties common to all axes
for a in axs:
  a.set(xticks=[],yticks=[],xlabel='Features',ylabel='Features')
  a.spines[['right','top']].set_visible(True)
```

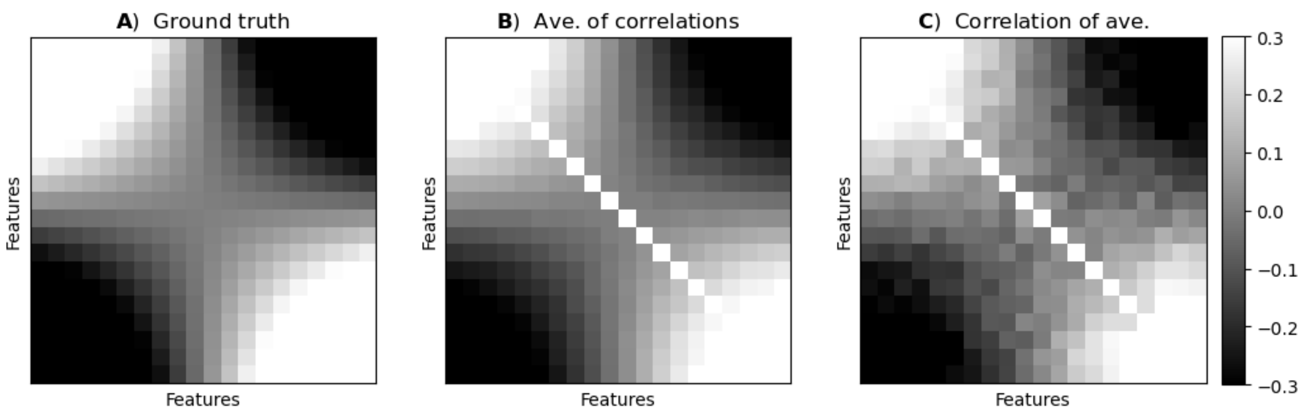
```
# plt.tight_layout()
#plt.savefig('cor_ex5.png')
plt.show()
```



21 Exercise 6

[39]:
```
# re-initialize data lists
alldata = np.zeros((M,N))

### run the experiment
for idx in range(numReps):
  # this run's data (only 'covars' is constant across repetitions)
  thisdata = np.random.randn(N)*covars + np.random.randn(M,N)

  # its correlation matrix
  corrmats[:,:,idx] = np.corrcoef(thisdata)

  # store the data
  alldata += thisdata

# correlation of data average
corrmats[:,:,-1] = np.corrcoef(alldata)

### plotting
fig,axs = plt.subplots(1,3,figsize=(12,5))

axs[0].imshow(covars@covars.T,vmin=-.3,vmax=.3,cmap='gray')
axs[0].set_title(r'$\bf{A}$)  Ground truth')

axs[1].imshow(np.mean(corrmats[:,:,:-1],axis=2),vmin=-.3,vmax=.3,cmap='gray')
axs[1].set_title(r'$\bf{B}$)  Ave. of correlations')

cax = axs[2].imshow(corrmats[:,:,-1],vmin=-.3,vmax=.3,cmap='gray')
```

```
axs[2].set_title(r'$\bf{C}$)  Correlation of ave.')

cbar_ax = fig.add_axes([.91,.22,.015,.55])
cbar = plt.colorbar(cax,cax=cbar_ax)

# properties common to all axes
for a in axs:
  a.set(xticks=[],yticks=[],xlabel='Features',ylabel='Features')
  a.spines[['right','top']].set_visible(True)

# plt.tight_layout()
plt.savefig('cor_ex6.png')
plt.show()
```



22 Exercise 7

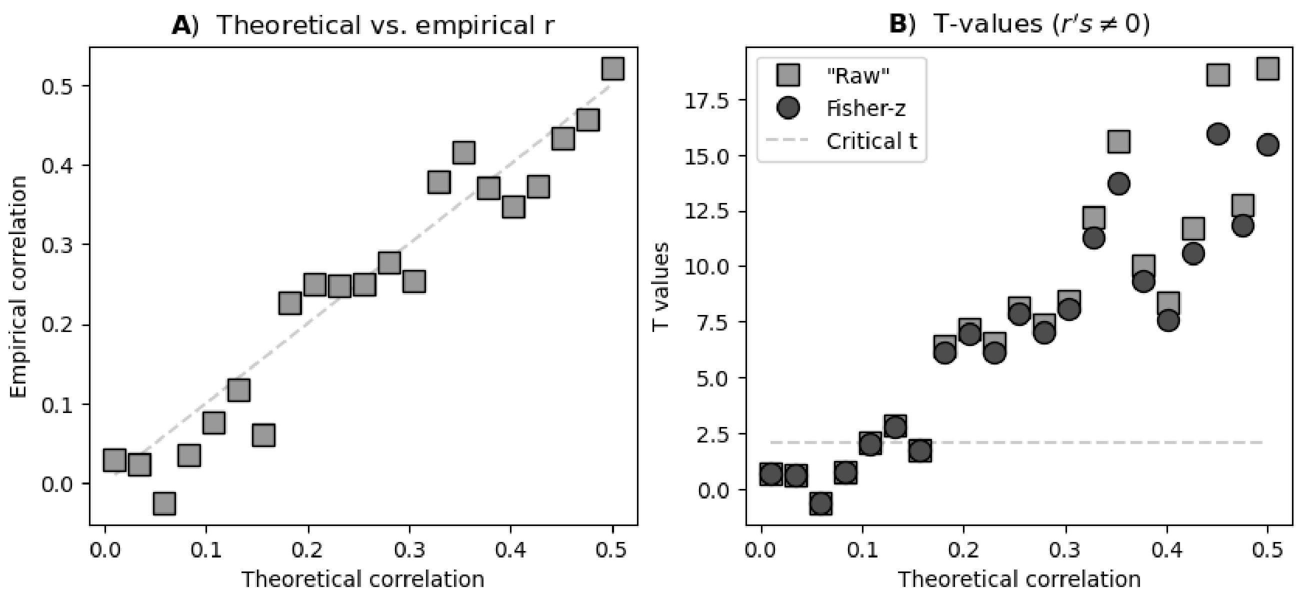
[40]:
```
samplesize = 30
nSamples = 23

corrs = np.zeros(nSamples)
tres  = np.zeros(2)

# loop over experiments
for ni in range(nSamples):
  # create the data
  x = np.random.randn(samplesize)
  y = np.random.randn(samplesize)
  y = x*.1 + y*np.sqrt(1-.1**2)

  # correlation
  corrs[ni] = np.corrcoef(x,y)[0,1]

# now for a t-test on r values
```

```
tres[0] = stats.ttest_1samp(corrs,0).statistic
tres[1] = stats.ttest_1samp(np.arctanh(corrs),0).statistic

# critical t-value
tCrit = stats.t.isf(.05/2,samplesize-2)

print(f't-value from "raw" coefficients   : {tres[0]:.4f}')
print(f't-value from Fisher-z coefficients: {tres[1]:.4f}')
print(f'Critical t-value for p<.05        : {tCrit:.4f}')
```

```
t-value from "raw" coefficients   : 2.8156
t-value from Fisher-z coefficients: 2.8027
Critical t-value for p<.05        : 2.0484
```

[42]:
```
# now for a wider range of r values
rs = np.linspace(.01,.5,21)
samplesize = 30
nSamples = 23

corrs = np.zeros((len(rs),nSamples))
tres  = np.zeros((len(rs),2))

# critical t (doesn't depend on the population r or sample size)
tCrit = stats.t.isf(.05/2,samplesize-2)

# run the experiment!
for ri,r in enumerate(rs):
  # loop over experiments
  for ni in range(nSamples):

    # create the data
    x = np.random.randn(samplesize)
    y = np.random.randn(samplesize)
    y = x*r + y*np.sqrt(1-r**2)

    # correlation
    corrs[ri,ni] = np.corrcoef(x,y)[0,1]

  # now for a t-test on r values
  tres[ri,0] = stats.ttest_1samp(corrs[ri,:],0).statistic
  tres[ri,1] = stats.ttest_1samp(np.arctanh(corrs[ri,:]),0).statistic

## plot
_,axs = plt.subplots(1,2,figsize=(10,4))
axs[0].plot(rs,np.mean(corrs,axis=1),'ks',markersize=10,markerfacecolor=(.6,.6,.
 ↪6))
axs[0].plot([rs[0],rs[-1]],[rs[0],rs[-1]],'--',color=(.8,.8,.8),zorder=-3)
```

```
axs[0].set(xlabel='Theoretical correlation',ylabel='Empirical correlation')
axs[0].set_title(r'$\bf{A}$)  Theoretical vs. empirical r')

axs[1].plot(rs,tres[:,0],'ks',markersize=10,markerfacecolor=(.6,.6,.
 ↪6),label='"Raw"')
axs[1].plot(rs,tres[:,1],'ko',markersize=10,markerfacecolor=(.3,.3,.
 ↪3),label='Fisher-z')
axs[1].plot(rs,np.full(len(rs),tCrit),'--',color=(.8,.8,.
 ↪8),zorder=-3,label='Critical t')
axs[1].legend()
axs[1].set(xlabel='Theoretical correlation',ylabel='T values')
axs[1].set_title(r"$\bf{B}$)  T-values ($r's\neq 0$)")

plt.figure(figsize=(7,3))
plt.tight_layout()
#plt.savefig('cor_ex7.png')
plt.show()
```




```
<Figure size 700x300 with 0 Axes>
```

23 Exercise 8

[43]:
```
# generate some correlated random data
x = np.random.randn(40)
y = x + np.random.randn(len(x))

# manual cosine similarity
cs_num = sum(x*y)
cs_den = np.sqrt(sum(x*x)) * np.sqrt(sum(y*y))
```

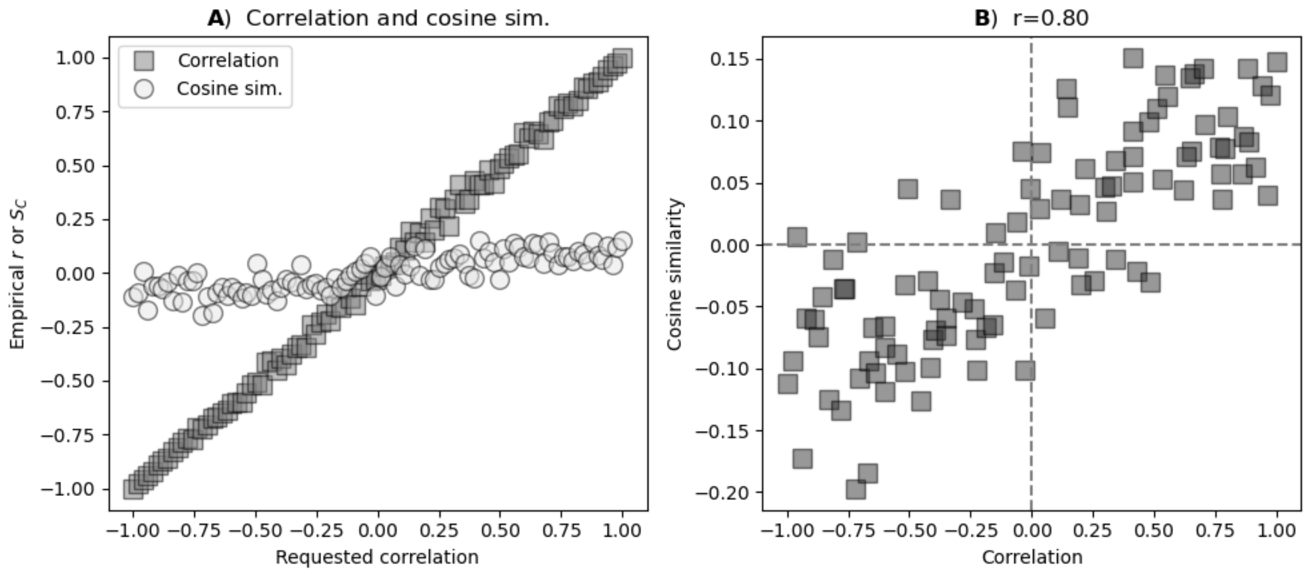
```
cs_me = cs_num / cs_den

# using the  distance function in the scipy.spatial library
# Note: using this function is confusing, because it computes *distance*␣
↪although we want *similarity*.
# Fortunately, the two are simple inverses, so one is 1- the other.
cs_sp = 1-spatial.distance.cosine(x,y)

print(f'Manual result: {cs_me:.3f}')
print(f'Scipy.spatial: {cs_sp:.3f}')
```

```
Manual result: 0.700
Scipy.spatial: 0.700
```

24 Exercise 9

[44]:
```
#  range of requested correlation coefficients
rs = np.linspace(-1,1,100)

# sample size
N = 500

# initialize output matrix
corrs = np.zeros((len(rs),2))

# loop over a range of r values
for ri in range(len(rs)):
  # generate data
  x = np.random.randn(N)
  y = x*rs[ri] + np.random.randn(N)*np.sqrt(1-rs[ri]**2)

  # mean de-centering
  x = x-10

  # compute correlation
  corrs[ri,0] = np.corrcoef(x,y)[0,1]

  # compute cosine similarity
  corrs[ri,1] = 1-spatial.distance.cosine(x,y)

## visualize the results
_,axs = plt.subplots(1,2,figsize=(10,4.5))

axs[0].plot(rs,corrs[:,0],'ks',markersize=10,markerfacecolor=(.5,.5,.5),alpha=.
↪5,label='Correlation')
axs[0].plot(rs,corrs[:,1],'ko',markersize=10,markerfacecolor=(.9,.9,.9),alpha=.
↪5,label='Cosine sim.')
```

```
axs[0].legend()
axs[0].set(xlabel='Requested correlation',ylabel=r'Empirical $r$ or $S_C$')
axs[0].set_title(r'$\bf{A}$)  Correlation and cosine sim.')

axs[1].plot(corrs[:,0],corrs[:,1],'ks',markersize=10,markerfacecolor=(.2,.2,.
 ↪2),alpha=.5)
axs[1].axhline(y=0,color='gray',linestyle='--')
axs[1].axvline(x=0,color='gray',linestyle='--')
axs[1].set(xlabel='Correlation',ylabel='Cosine similarity')
axs[1].set_title(rf'$\bf{{B}}$)  r={np.corrcoef(corrs.T)[1,0]:.2f}')

plt.tight_layout()
#plt.savefig('cor_ex9.png')
plt.show()
```



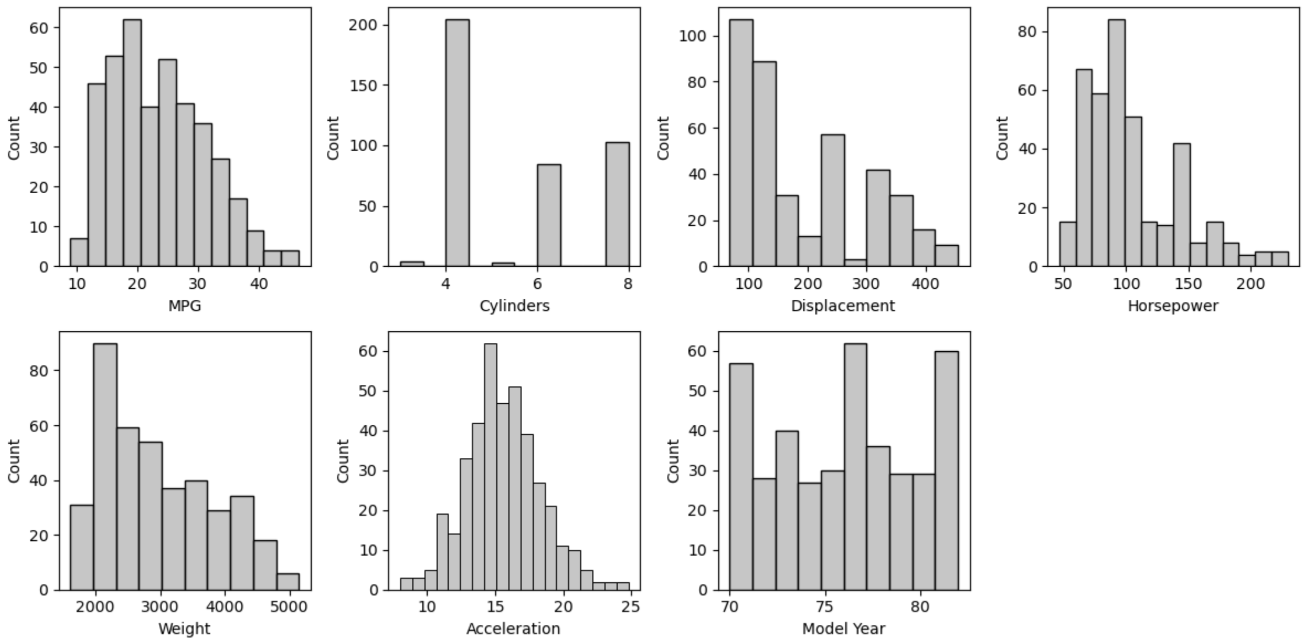
25 Exercise 10

[45]:
```
# import the data
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/
 ↪auto-mpg.data"
column_names =␣
 ↪['MPG','Cylinders','Displacement','Horsepower','Weight','Acceleration','Model␣
 ↪Year','Origin','Car Name']

data = pd.read_csv(url,delim_whitespace=True,names=column_names, na_values="?")
data
```

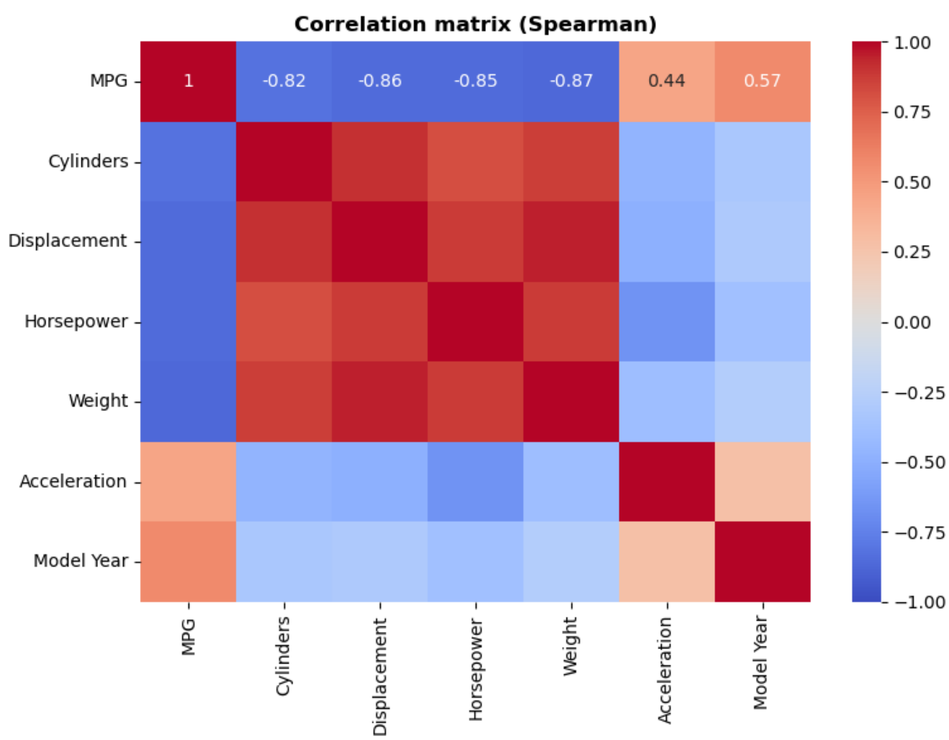
[45]:

```
     MPG  Cylinders  Displacement  Horsepower  Weight  Acceleration  \
0    18.0          8         307.0       130.0  3504.0          12.0
1    15.0          8         350.0       165.0  3693.0          11.5
2    18.0          8         318.0       150.0  3436.0          11.0
3    16.0          8         304.0       150.0  3433.0          12.0
4    17.0          8         302.0       140.0  3449.0          10.5
..    ...        ...           ...         ...     ...           ...
393  27.0          4         140.0        86.0  2790.0          15.6
394  44.0          4          97.0        52.0  2130.0          24.6
395  32.0          4         135.0        84.0  2295.0          11.6
396  28.0          4         120.0        79.0  2625.0          18.6
397  31.0          4         119.0        82.0  2720.0          19.4

     Model Year  Origin                   Car Name
0            70       1  chevrolet chevelle malibu
1            70       1          buick skylark 320
2            70       1         plymouth satellite
3            70       1              amc rebel sst
4            70       1                ford torino
..          ...     ...                        ...
393          82       1            ford mustang gl
394          82       2                  vw pickup
395          82       1              dodge rampage
396          82       1                ford ranger
397          82       1                 chevy s-10

[398 rows x 9 columns]
```

[46]:
```
# examine distributions

# include only numerical variables
data_numerical = data.drop(columns=['Car Name','Origin'])

# draw histograms with seaborn
fig,axs = plt.subplots(2,4,figsize=(12,6))
for a,column in zip(axs.flatten(),data_numerical.columns):
  sns.histplot(data=data_numerical, x=column, ax=a, color=(.7,.7,.7))

# only 7 columns, so switch off the empty 8th :P
axs[-1,-1].axis('off')

plt.tight_layout()
#plt.savefig('cor_ex10b.png')
plt.show()
```

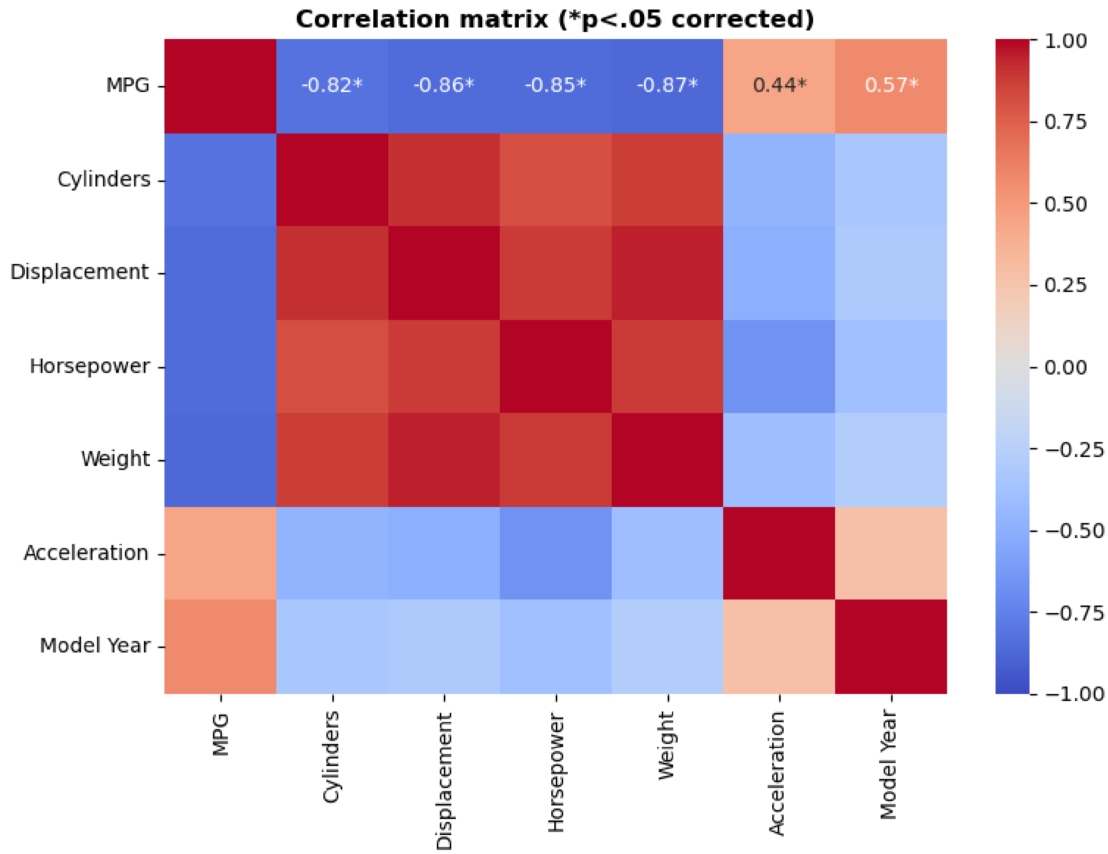


[47]:
```
# Create a correlation matrix
R = data_numerical.corr(method='spearman')
plt.figure(figsize=(8,6))
sns.heatmap(R, annot=True, cmap='coolwarm',vmin=-1,
            xticklabels=R.columns,yticklabels=R.columns)
plt.title('Correlation matrix (Spearman)',loc='center',weight='bold')
plt.tight_layout()
#plt.savefig('cor_ex10c.png')
plt.show()
```



26 Exercise 11

[48]:
```
# Calculate R and P matrices from Spearman correlation
R,P = stats.spearmanr(data_numerical,nan_policy='omit')

# store as dataframes for seaborn plotting (note: "df" here means "dataframe")
R_df = pd.DataFrame(R, columns=data_numerical.columns, index=data_numerical.
 ↪columns)
P_df = pd.DataFrame(P, columns=data_numerical.columns, index=data_numerical.
 ↪columns)

# Bonferroni correction [ formula is (M*(M-1))/2 ]
num_comparisons = (data_numerical.shape[1]*(data_numerical.shape[1]-1)) / 2
bonferroni_thresh = .05 / num_comparisons
significant = P_df < bonferroni_thresh

# Create a matrix of annotations
annot_array = R_df.astype(str).values

# loop through all elements of the matrix and create a string to display
for i in range(R_df.shape[0]):
  for j in range(R_df.shape[1]):
    # the string depends on the significance
    if not significant.iloc[i,j]:
      # if non-significant, just the correlation coefficient
      annot_array[i,j] = f'{R_df.iloc[i, j]:.2f}'
    else:
      # if significant, add an asterisk to the coefficient
      annot_array[i,j] = f'{R_df.iloc[i, j]:.2f}*'

    # don't need to report the diagonals (trivially=1)
    if i==j:
      annot_array[i,j] = ''

## now show the image
plt.figure(figsize=(8,6))
sns.heatmap(R_df,annot=annot_array,fmt='s',cmap='coolwarm',vmin=-1,
            xticklabels=R_df.columns,yticklabels=R_df.columns)
plt.title('Correlation matrix (*p<.05 corrected)',loc='center',weight='bold')
plt.tight_layout()
#plt.savefig('cor_ex11.png')
plt.show()
```


Correlation matrix (*p<.05 corrected)
MPG
Cylinders
Displacement
Horsepower
Weight
Acceleration
Model Year
-0.82*
-0.86*
-0.85*
-0.87*
0.44*
0.57*
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00