stats_ch06_transformations

August 2, 2024

1 Modern statistics: Intuition, Math, Python, R

1.1 Mike X Cohen (sincxpress.com)

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

Code for chapter 6 (transformations)

---

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

# define global figure properties used for publication
```

3 Z-score

[2]:
```
X = [ 1,4,-5,2,1 ]

mu = np.mean(X)
sigma = np.std(X,ddof=1)

X_z = (X-mu) / sigma
print(X_z)
```

```
[ 0.11899282  1.011439   -1.66589953  0.41647488  0.11899282]
```

[3]:
```
print(f'       Original | z-transformed')
print(f'Mean:      {np.mean(X):.2f} | {np.mean(X_z):.2f}')
print(f'stdev:     {np.std(X,ddof=1):.2f} | {np.std(X_z,ddof=1):.2f}')
```

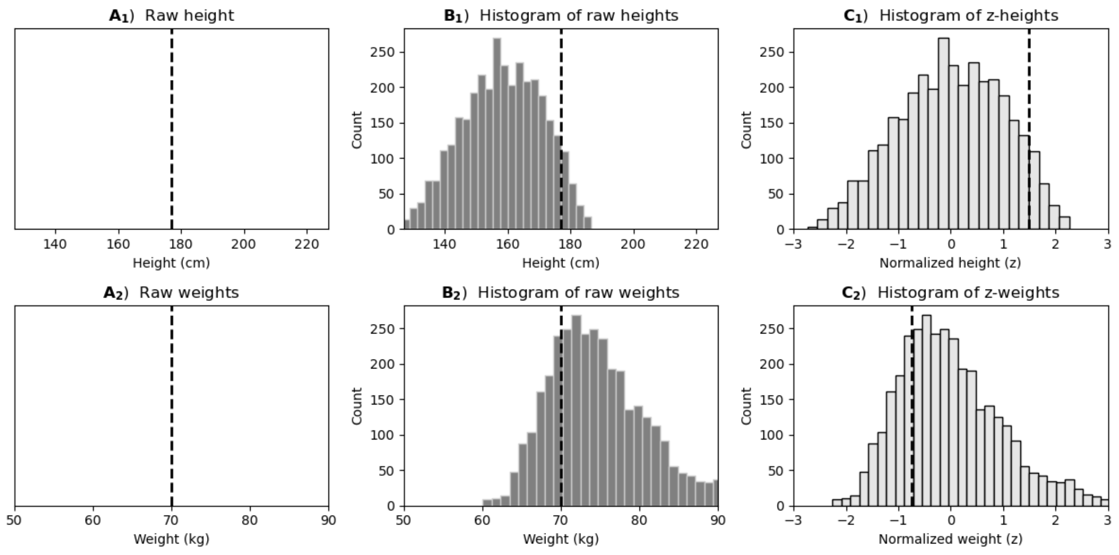
```
       Original | z-transformed
Mean:      0.60 | -0.00
stdev:     3.36 | 1.00
```

4 Figure 6.1: Example distributions of height and weight

[ ]:
```
### Note about this code: This code actually provides my solution to Exercise 6.
# So if you want the full experience of the exercise, please don't inspect this␣
↪code too carefully!
```

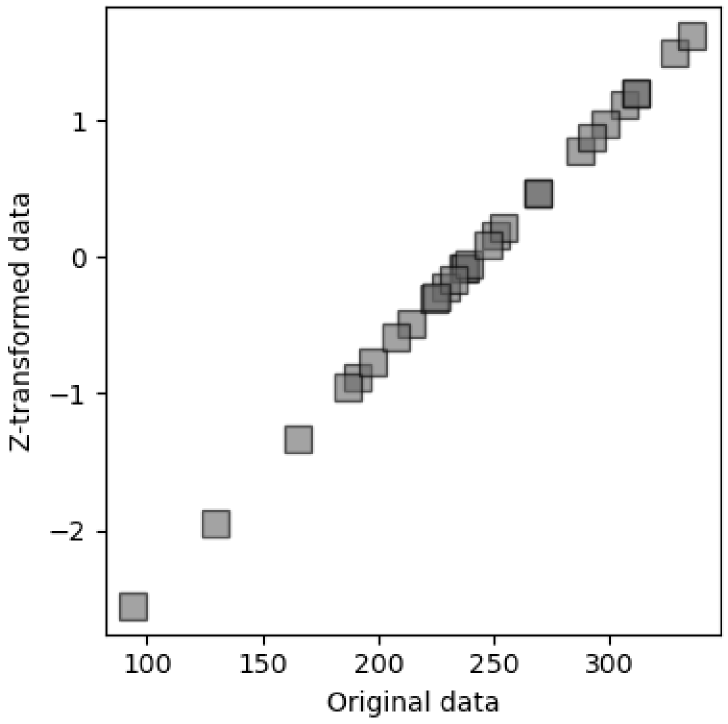
[4]:
```
# fake heights and weights, in units of cm and kg
N = 3425
height = np.arctanh(np.random.uniform(-.9,.8,size=N)) * 20 + 160 + np.random.
↪randn(N)*3
weight = np.arctanh(np.random.uniform(-.3,.99,size=N)) * 10 +  70 + np.random.
↪randn(N)*3

# our imaginary individual
ind_height = 177
ind_weight = 70

# z-score the distributions
height_z = (height-np.mean(height)) / np.std(height,ddof=1)
weight_z = (weight-np.mean(weight)) / np.std(weight,ddof=1)

# z-score the individual
ind_height_z = (ind_height-np.mean(height)) / np.std(height,ddof=1)
ind_weight_z = (ind_weight-np.mean(weight)) / np.std(weight,ddof=1)

# figure layout
_,axs = plt.subplots(2,3,figsize=(12,6))

# plot the values
axs[0,0].axvline(ind_height,color='k',linestyle='--',linewidth=2)
axs[0,0].set(xlabel='Height (cm)',yticks=[],xlim=[ind_height-50,ind_height+50])
axs[0,0].set_title(r'$\bf{A}_1$)  Raw height')

axs[1,0].axvline(ind_weight,color='k',linestyle='--',linewidth=2)
axs[1,0].set(xlabel='Weight (kg)',yticks=[],xlim=[ind_weight-20,ind_weight+20])
axs[1,0].set_title(r'$\bf{A}_2$)  Raw weights')
```

```
# plot the raw distributions with the individual
axs[0,1].hist(height,bins='fd',color=(.5,.5,.5),edgecolor=(.8,.8,.8))
axs[0,1].axvline(ind_height,color='k',linestyle='--',linewidth=2)
axs[0,1].set(xlabel='Height␣
 ↪(cm)',ylabel='Count',xlim=[ind_height-50,ind_height+50])
axs[0,1].set_title(r'$\bf{B}_1$)  Histogram of raw heights')

axs[1,1].hist(weight,bins='fd',color=(.5,.5,.5),edgecolor=(.8,.8,.8))
axs[1,1].axvline(ind_weight,color='k',linestyle='--',linewidth=2)
axs[1,1].set(xlabel='Weight␣
 ↪(kg)',ylabel='Count',xlim=[ind_weight-20,ind_weight+20])
axs[1,1].set_title(r'$\bf{B}_2$)  Histogram of raw weights')

# plot the z-normalized distributions
axs[0,2].hist(height_z,bins='fd',color=(.9,.9,.9),edgecolor='k')
axs[0,2].axvline(ind_height_z,color='k',linestyle='--',linewidth=2)
axs[0,2].set(xlabel='Normalized height (z)',ylabel='Count',xlim=[-3,3])
axs[0,2].set_title(r'$\bf{C}_1$)  Histogram of z-heights')

axs[1,2].hist(weight_z,bins='fd',color=(.9,.9,.9),edgecolor='k')
axs[1,2].axvline(ind_weight_z,color='k',linestyle='--',linewidth=2)
axs[1,2].set(xlabel='Normalized weight (z)',ylabel='Count',xlim=[-3,3])
axs[1,2].set_title(r'$\bf{C}_2$)  Histogram of z-weights')

# export
plt.tight_layout()
#plt.savefig('trans_zscore_example.png')
plt.show()
```



5 Figure 6.2: Z-scoring preserves relative values

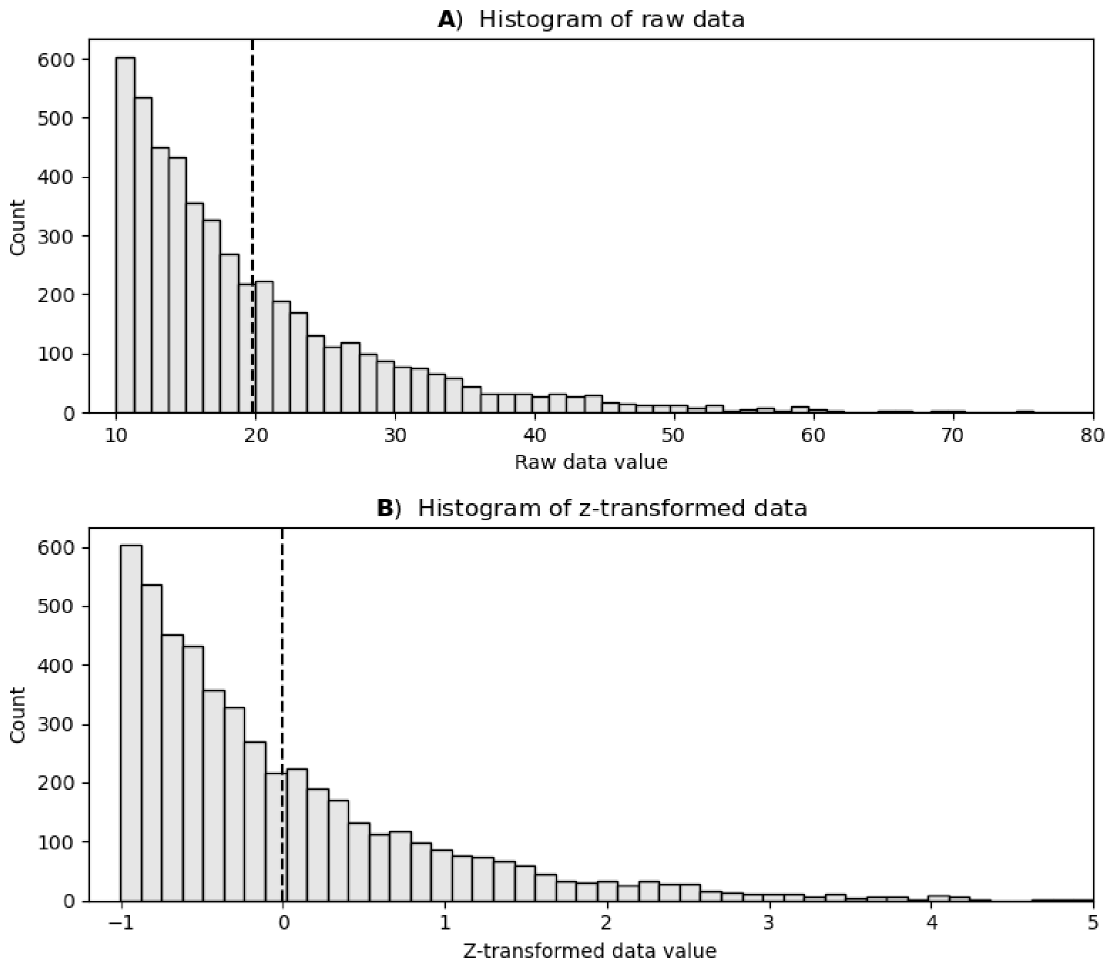
[5]:
```
x = np.random.normal(250,58,size=28)
zx = (x-np.mean(x)) / np.std(x,ddof=1)

plt.figure(figsize=(4,4))
plt.plot(x,zx,'ks',markersize=10,markerfacecolor=(.4,.4,.4),alpha=.6)
plt.xlabel('Original data')
plt.ylabel('Z-transformed data')

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



6 Figure 6.3: Z-scoring a non-normal distribution

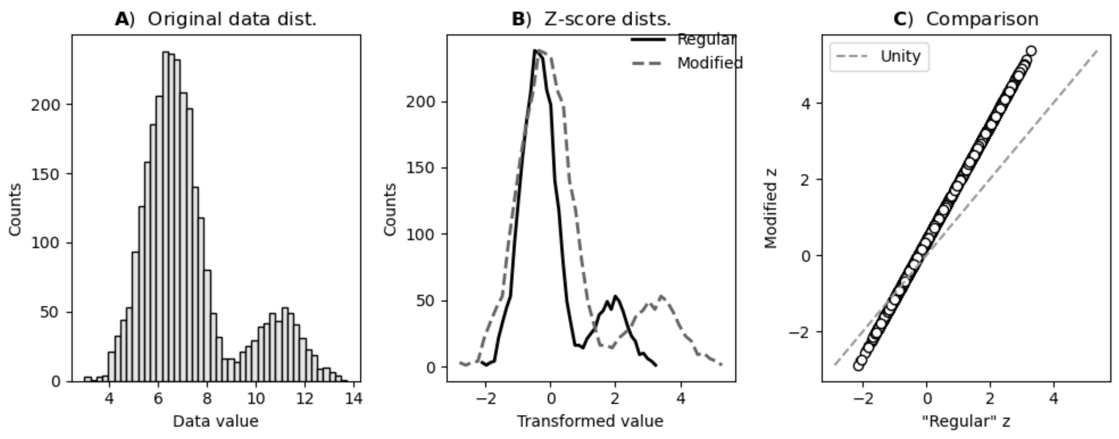
[6]:
```
N = 5000
X = (1+np.random.exponential(size=N))*10
Xz = (X-np.mean(X)) / np.std(X,ddof=1)

_,axs = plt.subplots(2,1,figsize=(8,7))
axs[0].hist(X,bins='fd',color=(.9,.9,.9),edgecolor='k')
```

```
axs[0].axvline(np.mean(X),color='k',linestyle='--')
axs[0].set(xlabel='Raw data value',ylabel='Count',xlim=[8,80])
axs[0].set_title(r'$\bf{A}$)  Histogram of raw data')

axs[1].hist(Xz,bins='fd',color=(.9,.9,.9),edgecolor='k')
axs[1].axvline(np.mean(Xz),color='k',linestyle='--')
axs[1].set(xlabel='Z-transformed data value',ylabel='Count',xlim=[-1.2,5])
axs[1].set_title(r'$\bf{B}$)  Histogram of z-transformed data')

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



7 Figure 6.4: Modified-z

[7]:
```
# Some non-normal data
x1 = np.random.randn(500) + 2.5
x2 = np.random.randn(2500) - 2
y  = np.concatenate((x1,x2))
y  = y - np.min(y)+3

# regular z-score
y_z = (y-np.mean(y)) / np.std(y,ddof=1)

# modified z
MAD = np.median( np.abs(y - np.median(y)) )
y_zm = stats.norm.ppf(3/4) * (y - np.median(y)) / MAD

# their histograms
yy_z,xx_z = np.histogram(y_z,bins='fd')
yy_zm,xx_zm = np.histogram(y_zm,bins='fd')

_,axs = plt.subplots(1,3,figsize=(10,4))

axs[0].hist(y,bins='fd',color=(.9,.9,.9),edgecolor='k')
axs[0].set(xlabel='Data value',ylabel='Counts',title=r'$\bf{A}$)  Original data␣
 ↪dist.')

axs[1].plot((xx_z[1:]+xx_z[:-1])/2,yy_z,linewidth=2,color='k',label='Regular')
axs[1].plot((xx_zm[1:]+xx_zm[:-1])/2,yy_zm,'--',linewidth=2,color=(.4,.4,.
 ↪4),label='Modified')
axs[1].legend(loc='upper right',frameon=False,bbox_to_anchor=[1.08,1.05])
axs[1].set(xlabel='Transformed value',ylabel='Counts',title=r'$\bf{B}$)  Z-score␣
 ↪dists.')

axs[2].plot(y_z,y_zm,'ko',markerfacecolor='w')
xval = np.min( [np.min(y_z),np.min(y_zm)] )
yval = np.max( [np.max(y_z),np.max(y_zm)] )
axs[2].plot([xval,yval],[xval,yval],'--',color=(.6,.6,.6),label='Unity')
axs[2].set(xlabel='"Regular" z',ylabel='Modified z',title=r'$\bf{C}$) ␣
 ↪Comparison')
axs[2].legend()

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

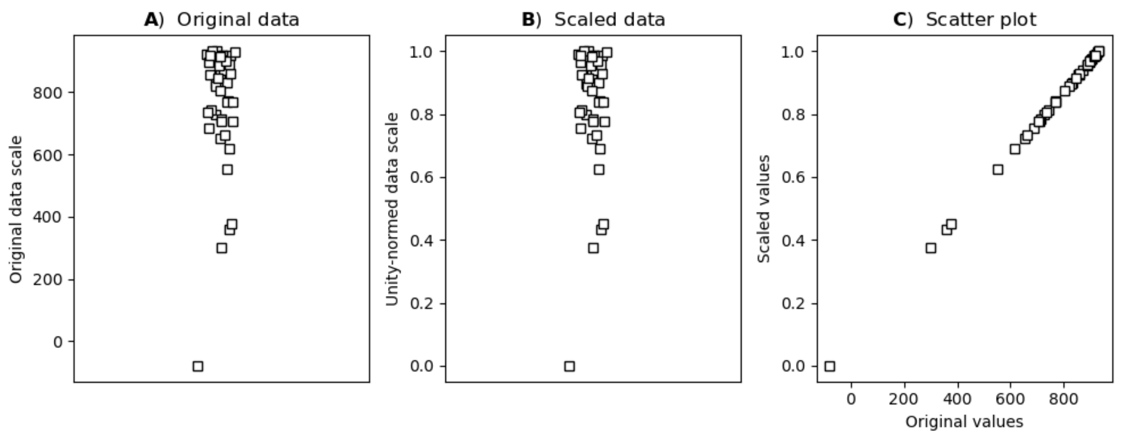


8 Figure 6.5: Min-max scaling

```
[3]: def minmaxScaling(x):

       # compute min/max values
       minx = np.min(x)
       maxx = np.max(x)

       # transformation
       xScaled = (x-minx) / (maxx-minx)

       # output
       return xScaled

     # Note: I wrote the function over multiple lines for clarity; you could reduce␣
      ↪it to one line!
```

```
[4]: ## create some data
     N = 42
     data = np.log(np.random.rand(N))*234 + 934

     # now min-max scale
     dataS = minmaxScaling(data)

     # now plot
     fig,ax = plt.subplots(1,3,figsize=(10,4))
     randomXoffsets = 1+np.random.randn(N)/20
     ax[0].plot(randomXoffsets,data,'ks',markerfacecolor='w')
     ax[0].set(xlim=[0,2],xticks=[],ylabel='Original data scale')
     ax[0].set_title(r'$\bf{A}$)  Original data')
```

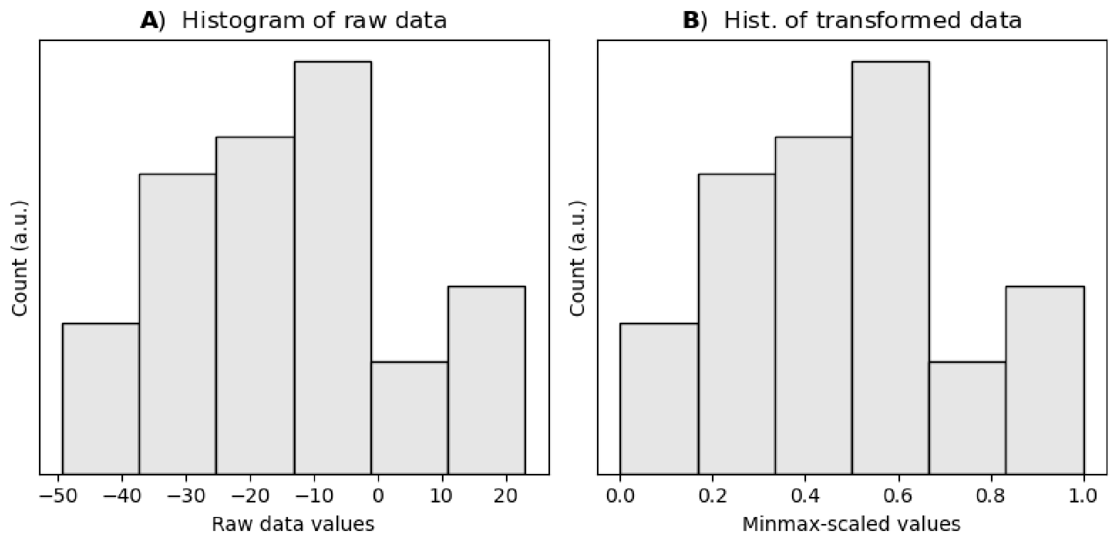
```
ax[1].plot(randomXoffsets,dataS,'ks',markerfacecolor='w')
ax[1].set(xlim=[0,2],xticks=[],ylabel='Unity-normed data scale')
ax[1].set_title(r'$\bf{B}$)  Scaled data')

ax[2].plot(data,dataS,'ks',markerfacecolor='w')
ax[2].set(xlabel='Original values',ylabel='Scaled values')
ax[2].set_title(r'$\bf{C}$)  Scatter plot')

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



9 Figure 6.6: Another example of min-max scaling

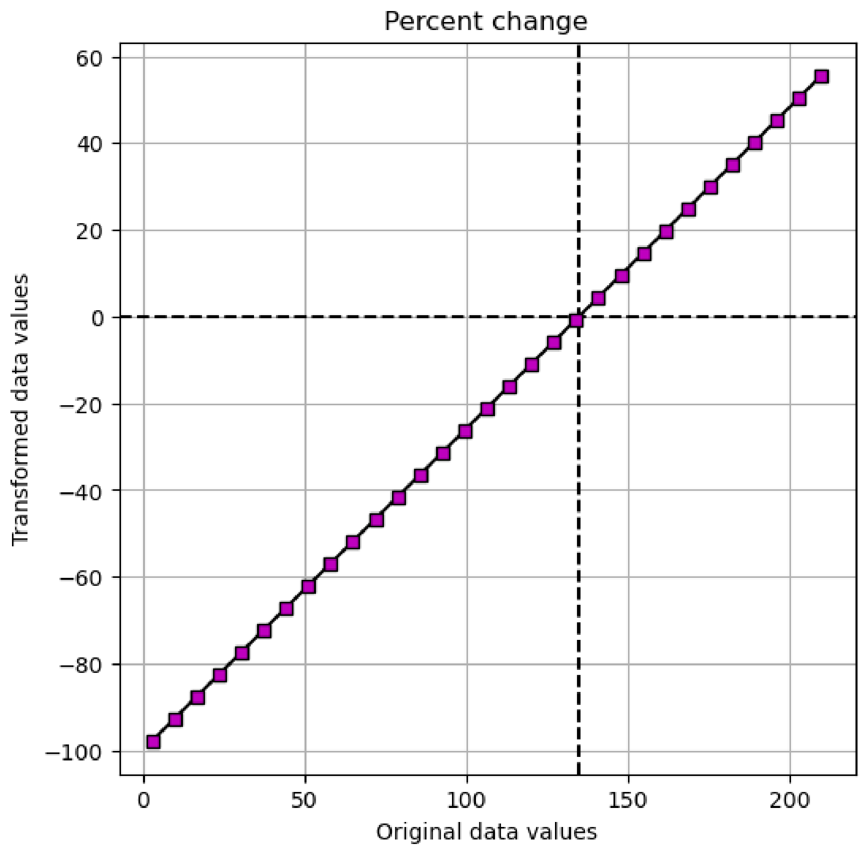
[5]:
```
# generate a Laplace distribution
x1 = np.exp(-np.abs(3*np.random.randn(40)))
x2 = np.exp(-np.abs(3*np.random.randn(40)))
x = (x1-x2)*42 - 13

# minmax scale
xm = minmaxScaling(x)

_,axs = plt.subplots(1,2,figsize=(8,4))

axs[0].hist(x,bins='fd',color=(.9,.9,.9),edgecolor='k')
axs[0].set(yticks=[],ylabel='Count (a.u.)',xlabel='Raw data values')
axs[0].set_title(r'$\bf{A}$)  Histogram of raw data')

axs[1].hist(xm,bins='fd',color=(.9,.9,.9),edgecolor='k')
axs[1].set(yticks=[],ylabel='Count (a.u.)',xlabel='Minmax-scaled values')
axs[1].set_title(r'$\bf{B}$)  Hist. of transformed data')
```

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



10 Percent change

[11]:
```
# Note: This code does not correspond to any figure in the book;
#  I include it here to illustrate the code.
#  Notice in the graph how the 'ref' data value turns into 0.

# a range of values for "new"
new = np.linspace(3,210,31)

# reference value
ref = 135

# compute percent change
pctchg = 100*(new-ref) / ref

# visualize the transformed data
plt.figure(figsize=(6,6))
plt.plot(new,pctchg,'ks-',markerfacecolor='m',zorder=30)
plt.axhline(0,color='k',linestyle='--')
plt.axvline(ref,color='k',linestyle='--')
plt.xlabel('Original data values')
plt.ylabel('Transformed data values')
plt.title('Percent change',loc='center')
plt.grid()
plt.show()
```



11 Rank transform

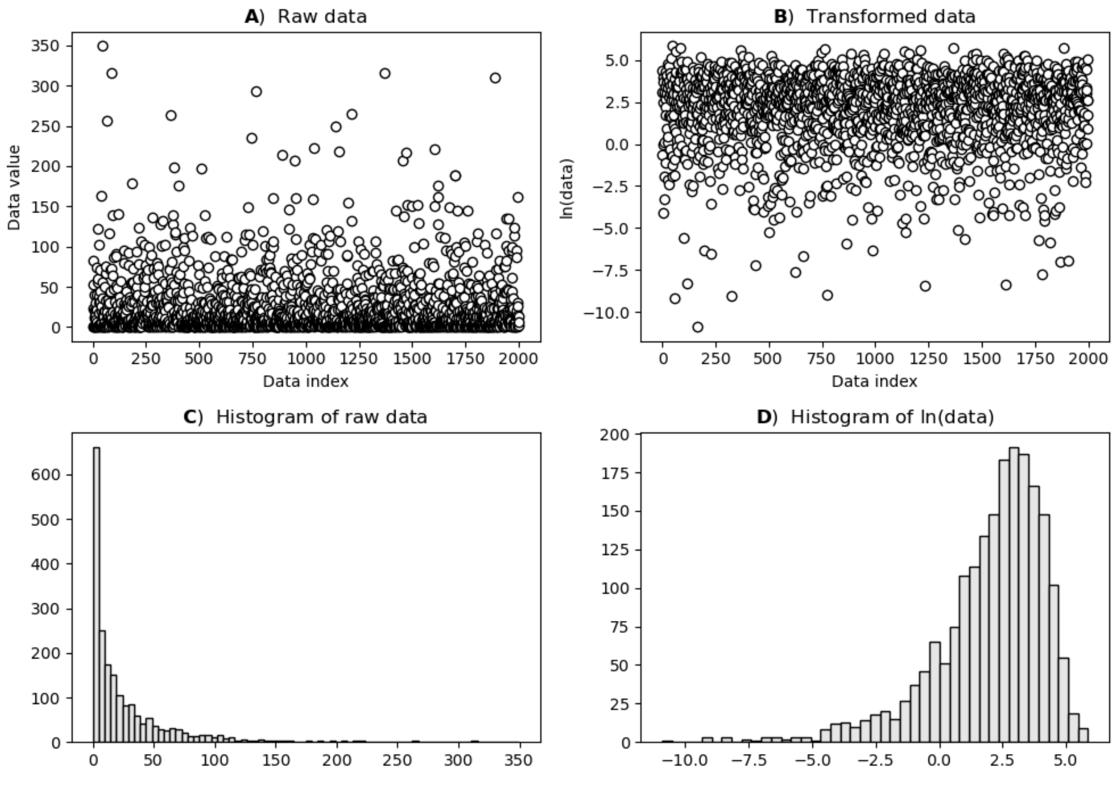
[12]:
```
x = [ 4,1,2,3,3 ]

stats.rankdata(x)
```

[12]:
```
array([5. , 1. , 2. , 3.5, 3.5])
```

12 Figure 6.7: Log transform

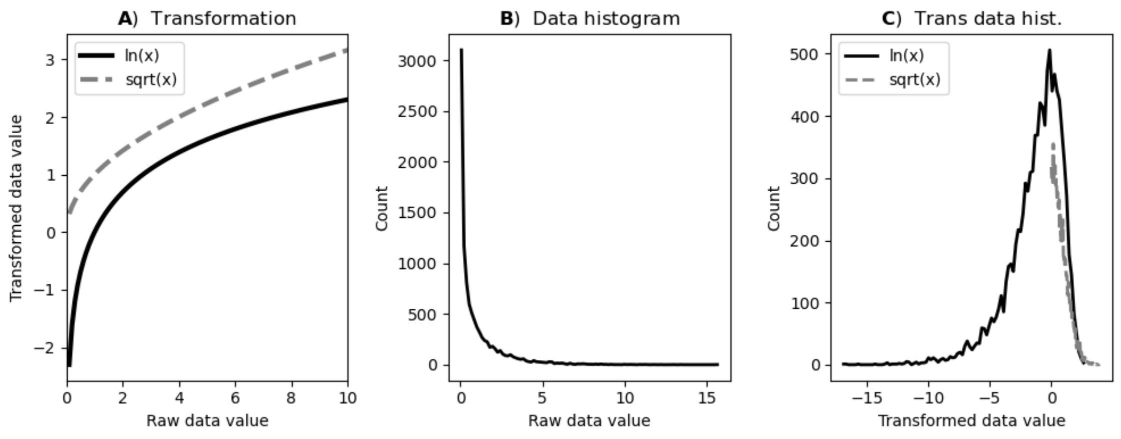
[13]:
```
# generate data from power-law distribution
X = np.random.normal(0,5,size=2000)**2
Xlog = np.log(X)

_,axs = plt.subplots(2,2,figsize=(10,7))
```

```
# scatter plots
axs[0,0].plot(X,'ko',markerfacecolor='w')
axs[0,0].set(xlabel='Data index',ylabel='Data value')
axs[0,0].set_title(r'$\bf{A}$)  Raw data')

axs[0,1].plot(Xlog,'ko',markerfacecolor='w')
axs[0,1].set(xlabel='Data index',ylabel='ln(data)')
axs[0,1].set_title(r'$\bf{B}$)  Transformed data')

# histograms
axs[1,0].hist(X,bins='fd',color=(.9,.9,.9),edgecolor='k')
axs[1,0].set_title(r'$\bf{C}$)  Histogram of raw data')

axs[1,1].hist(Xlog,bins='fd',color=(.9,.9,.9),edgecolor='k')
axs[1,1].set_title(r'$\bf{D}$)  Histogram of ln(data)')

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



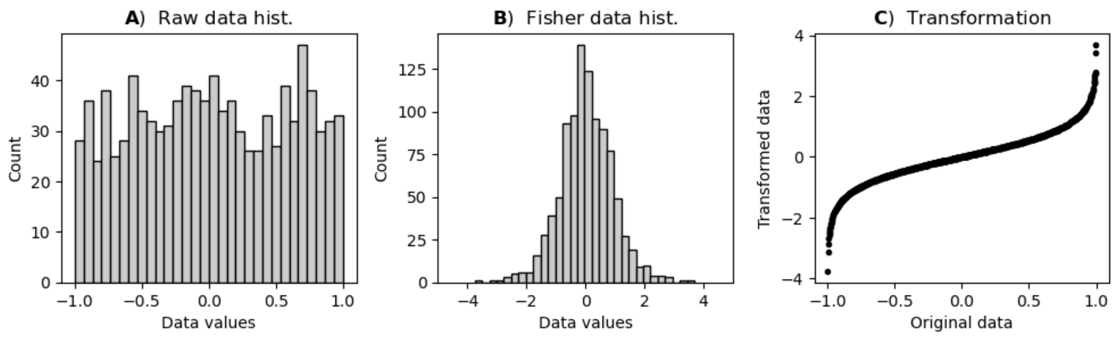
13 Figure 6.8: Logarithm vs square root

[14]:
```
# some right-tailed non-normally distributed data
X = np.random.randn(10000)**2

y_r,x_r = np.histogram(X,bins=100) # r = raw
y_l,x_l = np.histogram(np.log(X),bins=100) # l = log
y_s,x_s = np.histogram(np.sqrt(X),bins=100) # s = sqrt

_,axs = plt.subplots(1,3,figsize=(10,4))

# theory
q = np.linspace(.1,10,100)
axs[0].plot(q,np.log(q),'k',linewidth=3,label='ln(x)')
axs[0].plot(q,np.sqrt(q),'--',color='gray',linewidth=3,label='sqrt(x)')
axs[0].set(xlim=[0,10],xlabel='Raw data value',ylabel='Transformed data value')
axs[0].set_title(r'$\bf{A}$)  Transformation')
axs[0].legend()

# untransformed data
axs[1].plot((x_r[1:]+x_r[:-1])/2,y_r,'k',linewidth=2)
axs[1].set(xlabel='Raw data value',ylabel='Count')
axs[1].set_title(r'$\bf{B}$)  Data histogram')

axs[2].plot((x_l[1:]+x_l[:-1])/2,y_l,'k',linewidth=2,label='ln(x)')
axs[2].plot((x_s[1:]+x_s[:-1])/
 ↪2,y_s,'--',color='gray',linewidth=2,label='sqrt(x)')
axs[2].set(xlabel='Transformed data value',ylabel='Count')
axs[2].set_title(r'$\bf{C}$)  Trans data hist.')
axs[2].legend()

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



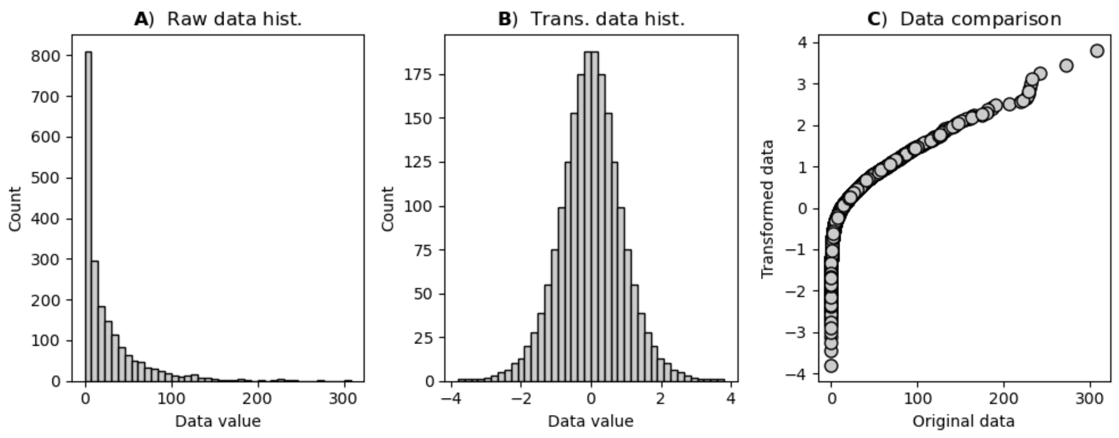
14 Figure 6.9: Fisher-z transform

[15]:
```
N = 1000

# uniform data in range [-1,1]
Y = np.random.uniform(-1,1,size=N)
# Note: Fisher-z is undefined for Y==|1|, but the probability of that happening␣
↪in
#       random uniform data is so vanishingly small that coding an exception is␣
↪unnecessary.

# transform
fY = np.arctanh(Y)

_,axs = plt.subplots(1,3,figsize=(10,np.pi))

axs[0].hist(Y,30,color=(.8,.8,.8),edgecolor='k')
axs[0].set(xlabel='Data values', ylabel='Count')
axs[0].set_title(r'$\bf{A}$)  Raw data hist.')

axs[1].hist(fY,30,color=(.8,.8,.8),edgecolor='k')
axs[1].set(xlim=[-5,5],xlabel='Data values', ylabel='Count')
axs[1].set_title(r'$\bf{B}$)  Fisher data hist.')

axs[2].plot(Y,fY,'k.')
axs[2].set_title(r'$\bf{C}$)  Transformation')
axs[2].set(xlabel='Original data',ylabel='Transformed data')

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



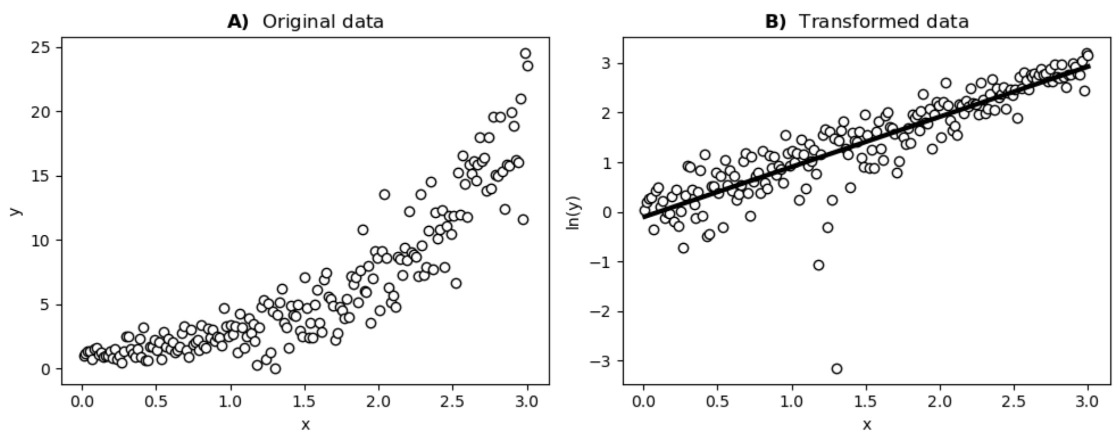
15 Figure 6.10: Transform any distribution to Gaussian

[16]:
```
X = np.random.normal(0,5,size=2000)**2

X_r = stats.rankdata( X )
a,b = -.999,.999
X_r2 = (X_r-np.min(X_r)) / (np.max(X_r)-np.min(X_r)) * (b-a) + a
X_t = np.arctanh( X_r2 )

_,axs = plt.subplots(1,3,figsize=(10,4))
axs[0].hist(X,40,color=(.8,.8,.8),edgecolor='k')
axs[0].set(xlabel='Data value',ylabel='Count')
axs[0].set_title(r'$\bf{A}$)  Raw data hist.')

axs[1].hist(X_t,40,color=(.8,.8,.8),edgecolor='k')
axs[1].set(xlabel='Data value',ylabel='Count')
axs[1].set_title(r'$\bf{B}$)  Trans. data hist.')

axs[2].plot(X,X_t,'ko',markersize=8,markerfacecolor=(.8,.8,.8))
axs[2].set(xlabel='Original data',ylabel='Transformed data')
axs[2].set_title(r'$\bf{C}$)  Data comparison')

plt.tight_layout()
#plt.savefig('trans_any2gauss.png')
plt.show()
```



16 Figure 6.11: Linearization of nonlinear relationship

[17]:
```python
# sample size
N = 200

# the data (note how the nonlinearity is implemented)
x = np.linspace(.01,3,N)
y = np.exp(x) + np.random.randn(N)*np.linspace(.1,1,N)*3
y = np.abs(y)

_,axs = plt.subplots(1,2,figsize=(10,4))

axs[0].plot(x,y,'ko',markerfacecolor='w')
axs[0].set(xlabel='x',ylabel='y')
axs[0].set_title(r'$\bf{A})$  Original data')

axs[1].plot(x,np.log(y),'ko',markerfacecolor='w')
axs[1].set(xlabel='x',ylabel='ln(y)')
axs[1].set_title(r'$\bf{B})$  Transformed data')

a,b = np.polyfit(x,np.log(y),1)
axs[1].plot(x,a*x+b,'k',linewidth=3)

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



17 Exercise 1

[18]:
```
# a function to scale to any arbitrary bounds
def my_minmaxScalar(x,newmin,newmax):

  # get min and max
  minx = np.min(x)
  maxx = np.max(x)

  # (intermediate) min-max scale to [0,1]
  xS = (x-minx) / (maxx-minx)

  # scale to [a,b]
  xSS = xS*(newmax-newmin) + newmin

  return xSS

  # this solution harder to read, but is what your equation should look like on␣
 ↪paper:
  # return (x-np.min(x)) / (np.max(x)-np.min(x)) * (newmax-newmin) + newmin
```

[19]:
```
# test it on some data
data = np.random.randn(10)
a,b = 14.3,34

# apply the transform
y = my_minmaxScalar(data,a,b)

print(f'Min value: {np.min(y)}')
print(f'Max value: {np.max(y)}')
```

```
Min value: 14.3
Max value: 34.0
```

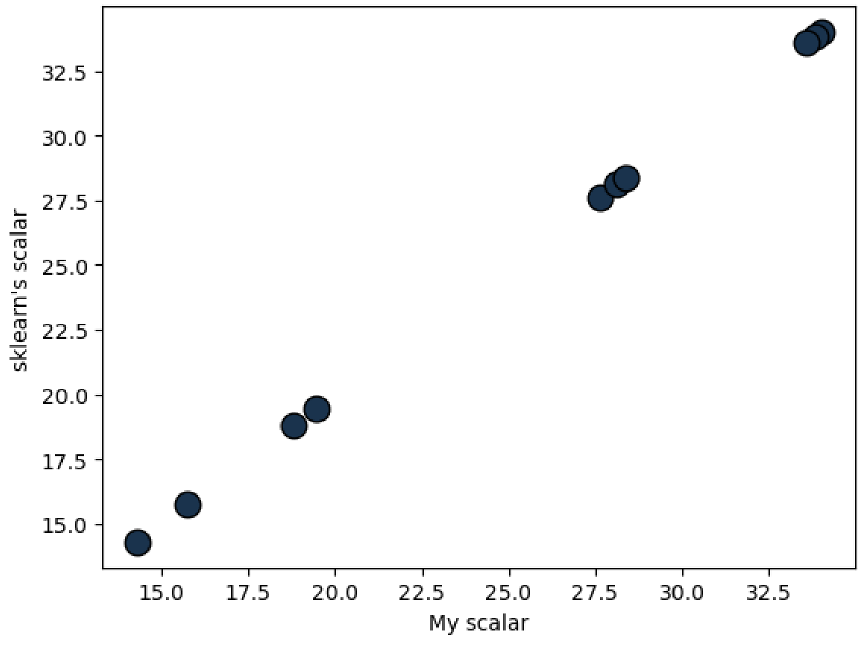
[20]:
```
# import scalar class from sklearn
from sklearn.preprocessing import MinMaxScaler

# define an instance of this class, with boundary parameters
scaler = MinMaxScaler(feature_range=(a,b))

# fit to the data (but the data need to be a column vector)
scaler.fit(data.reshape(-1,1))

# finally, call the transformer object with the data (again as column vector)
y2 = scaler.transform(data.reshape(-1,1))

# print the min and max values
print(f'Min value: {np.min(y2)}')
print(f'Max value: {np.max(y2)}')
```

```
Min value: 14.3
Max value: 34.0
```

[21]:
```
# print out all individual numbers to confirm they're the same
np.hstack((y.reshape(-1,1),y2.reshape(-1,1)))
```

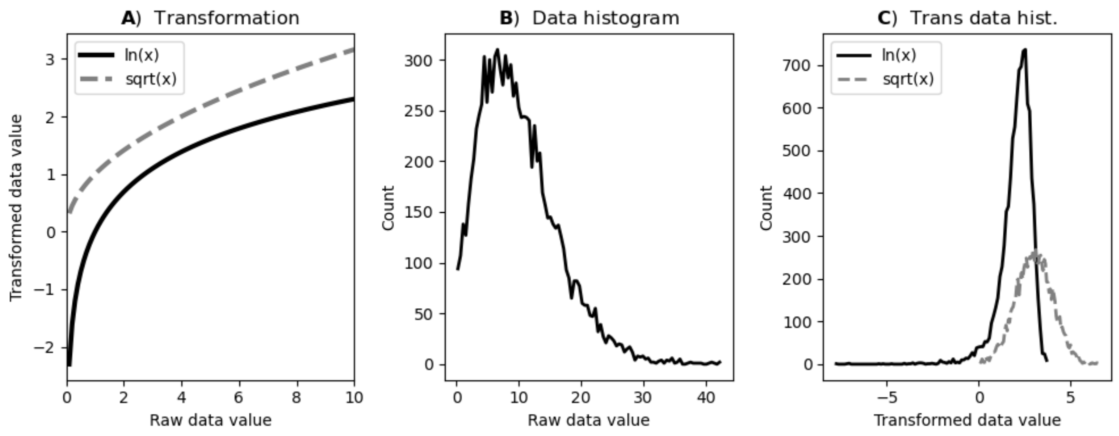
[21]:
```
array([[34.        , 34.        ],
       [33.82024842, 33.82024842],
       [33.58159811, 33.58159811],
       [18.79802667, 18.79802667],
       [27.62550985, 27.62550985],
       [28.11938553, 28.11938553],
       [28.36384991, 28.36384991],
       [19.44601043, 19.44601043],
       [15.74049553, 15.74049553],
       [14.3       , 14.3       ]])
```

[22]:
```
# scatter plot
plt.plot(y,y2,'ko',markerfacecolor=(.1,.2,.3),markersize=12)
plt.xlabel('My scalar')
plt.ylabel("sklearn's scalar")
plt.show()
```

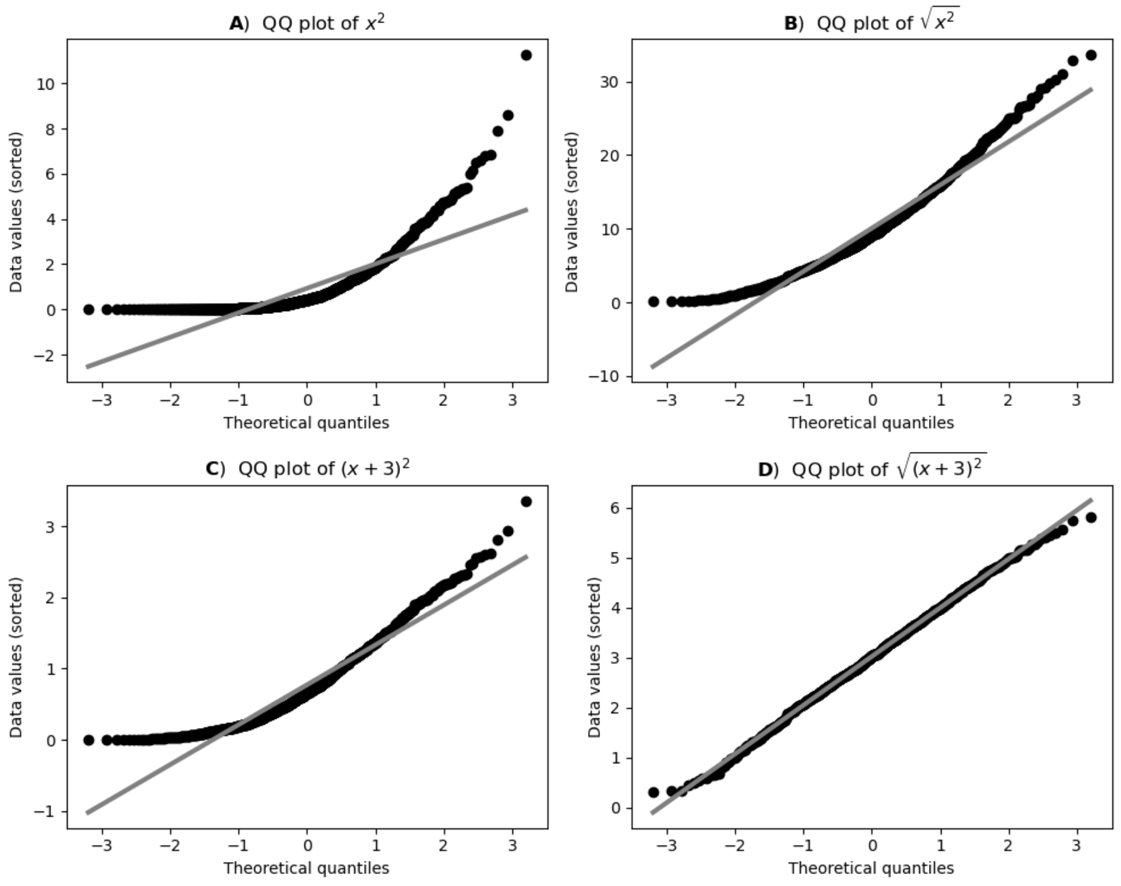


18 Exercise 2

[23]:
```
# this is the only line that's modified from the code earlier in the chapter
X = (np.random.randn(10000)+3)**2

y_r,x_r = np.histogram(X,bins=100) # r = raw
y_l,x_l = np.histogram(np.log(X),bins=100) # l = log
y_s,x_s = np.histogram(np.sqrt(X),bins=100) # s = sqrt

_,axs = plt.subplots(1,3,figsize=(10,4))

# theory
q = np.linspace(.1,10,100)
axs[0].plot(q,np.log(q),'k',linewidth=3,label='ln(x)')
axs[0].plot(q,np.sqrt(q),'--',color='gray',linewidth=3,label='sqrt(x)')
axs[0].set(xlim=[0,10],xlabel='Raw data value',ylabel='Transformed data value')
axs[0].set_title(r'$\bf{A}$)  Transformation')
axs[0].legend()

# untransformed data
axs[1].plot((x_r[1:]+x_r[:-1])/2,y_r,'k',linewidth=2)
axs[1].set(xlabel='Raw data value',ylabel='Count')
axs[1].set_title(r'$\bf{B}$)  Data histogram')

axs[2].plot((x_l[1:]+x_l[:-1])/2,y_l,'k',linewidth=2,label='ln(x)')
axs[2].plot((x_s[1:]+x_s[:-1])/
 ↪2,y_s,'--',color='gray',linewidth=2,label='sqrt(x)')
axs[2].set(xlabel='Transformed data value',ylabel='Count')
axs[2].set_title(r'$\bf{C}$)  Trans data hist.')
axs[2].legend()

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

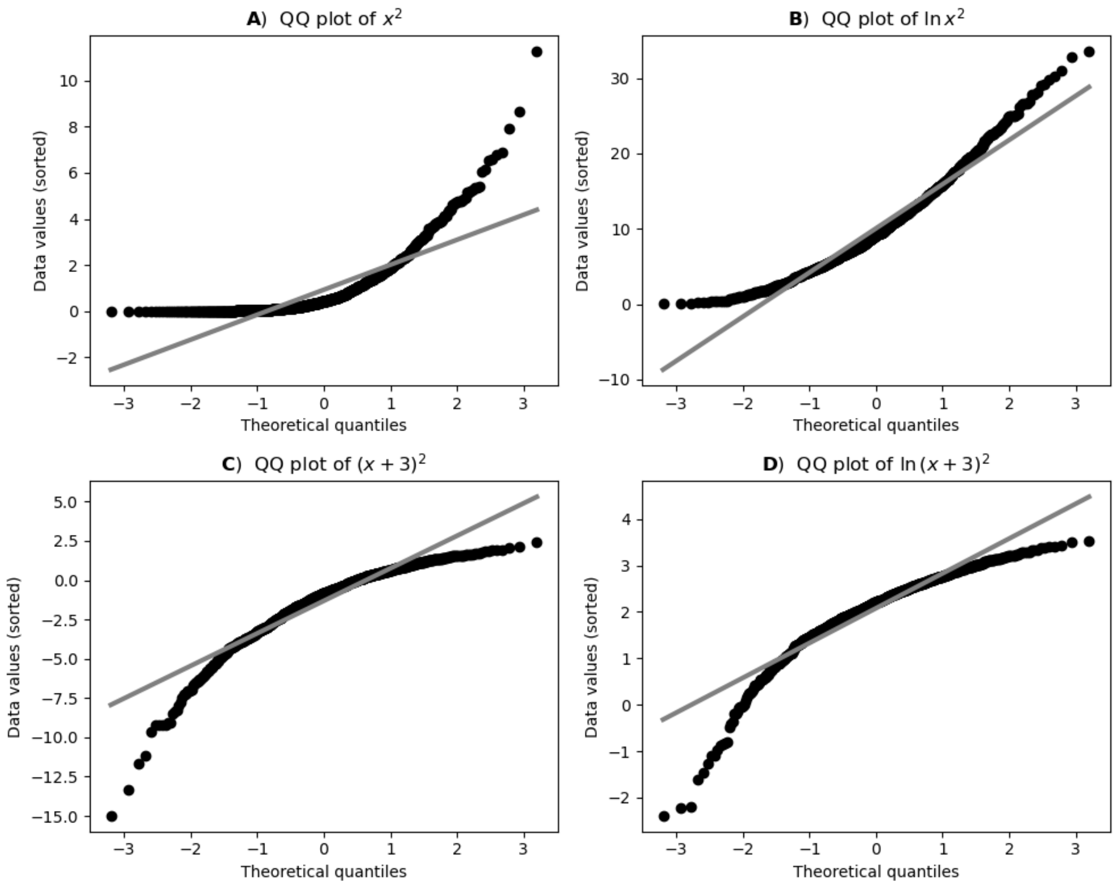


[24]:
```
# data
X1 = (np.random.randn(1000)+0)**2
X2 = (np.random.randn(1000)+3)**2

# sqrt transform
X1s = np.sqrt(X1)
X2s = np.sqrt(X2)

_,axs = plt.subplots(2,2,figsize=(10,8))

# QQ plots
stats.probplot(X1,plot=axs[0,0],fit=True)
stats.probplot(X2,plot=axs[0,1],fit=True)

stats.probplot(X1s,plot=axs[1,0],fit=True)
stats.probplot(X2s,plot=axs[1,1],fit=True)

for a in axs.flatten():
  a.get_lines()[0].set_markerfacecolor('k')
  a.get_lines()[0].set_markeredgecolor('k')
  a.get_lines()[1].set_color('gray')
  a.get_lines()[1].set_linewidth(3)
  a.set_title(' ')
  a.set_ylabel('Data values (sorted)')

axs[0,0].set_title(r'$\bf{A}$)  QQ plot of $x^2$')
axs[0,1].set_title(r'$\bf{B}$)  QQ plot of $\sqrt{x^2}$')
axs[1,0].set_title(r'$\bf{C}$)  QQ plot of $(x+3)^2$')
axs[1,1].set_title(r'$\bf{D}$)  QQ plot of $\sqrt{(x+3)^2}$')

plt.tight_layout()
#plt.savefig('trans_ex2.png')
plt.show()
```

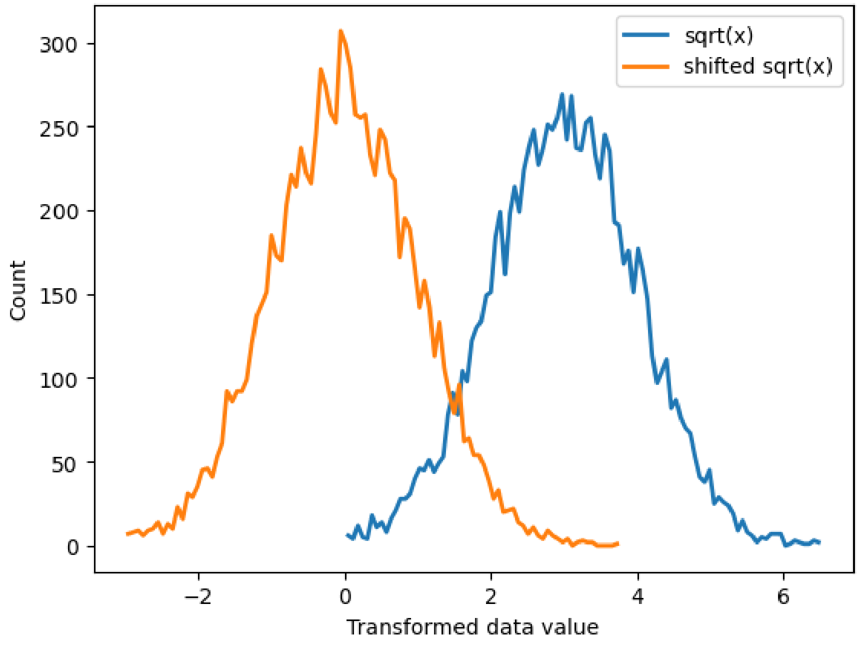


```
[25]: # repeat for log transform
      X1s = np.log(X1)
      X2s = np.log(X2)

      _,axs = plt.subplots(2,2,figsize=(10,8))

      # QQ plots
      stats.probplot(X1,plot=axs[0,0],fit=True)
      stats.probplot(X2,plot=axs[0,1],fit=True)

      stats.probplot(X1s,plot=axs[1,0],fit=True)
      stats.probplot(X2s,plot=axs[1,1],fit=True)

      for a in axs.flatten():
        a.get_lines()[0].set_markerfacecolor('k')
        a.get_lines()[0].set_markeredgecolor('k')
        a.get_lines()[1].set_color('gray')
        a.get_lines()[1].set_linewidth(3)
        a.set_title(' ')
```

```
  a.set_ylabel('Data values (sorted)')

axs[0,0].set_title(r'$\bf{A}$)  QQ plot of $x^2$')
axs[0,1].set_title(r'$\bf{B}$)  QQ plot of $\ln{x^2}$')
axs[1,0].set_title(r'$\bf{C}$)  QQ plot of $(x+3)^2$')
axs[1,1].set_title(r'$\bf{D}$)  QQ plot of $\ln{(x+3)^2}$')

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



[26]:
```
# Shift the data to be mean-centered
X = (np.random.randn(10000)+3)**2
Y = np.sqrt(X)
Y -= np.mean(Y)
y_s2,x_s2 = np.histogram(Y,bins=100)

plt.plot((x_s[1:]+x_s[:-1])/2,y_s,linewidth=2,label='sqrt(x)')
plt.plot((x_s2[1:]+x_s2[:-1])/2,y_s2,linewidth=2,label='shifted sqrt(x)')
plt.xlabel('Transformed data value')
plt.ylabel('Count')
plt.legend()
plt.show()
```

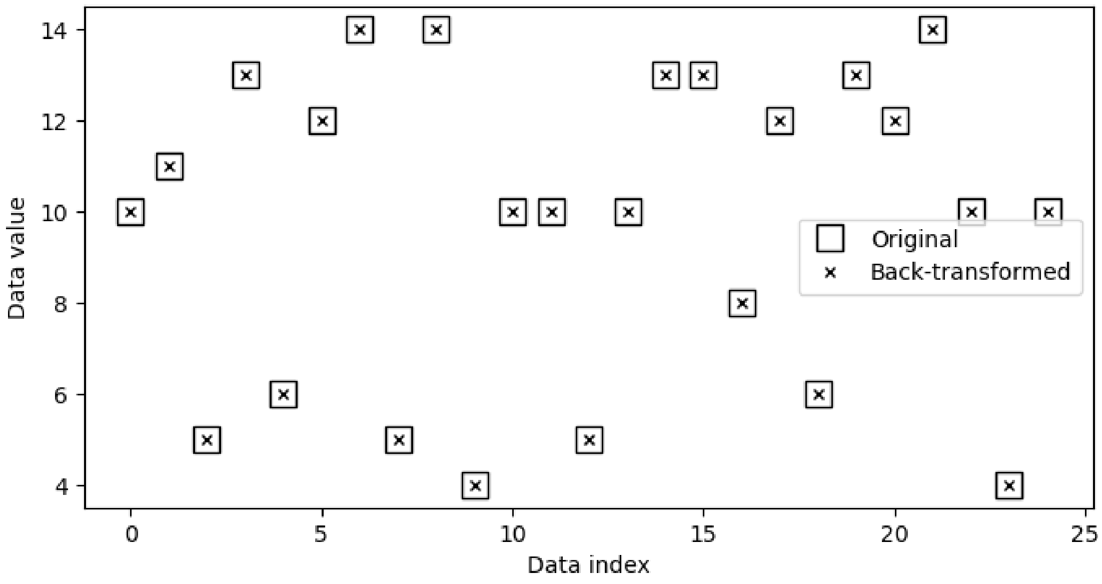


19 Exercise 3

[ ]:
```
# After a bit of algebra, you should arive at the equation:
#  zs+m = x
# where: z is the z-transformed data
#        s is the standard deviation of the original data
#        m is the mean (mu) of the original data
#        x is the reconstructed data
#
# The implication is Yes, the z-score transformation is invertible,
#    but only if you store the mean and std of the pre-transformed data.
```

[27]:
```
### Now for the empirical illustration:

# generate data
data = np.random.randint(4,15,25)

# pre-transformed descriptive statistics
orig_m = np.mean(data)
orig_s = np.std(data,ddof=1)
```

```
# transform to z-score
dataz = (data-orig_m) / orig_s

# back-transform
data_inv = dataz*orig_s + orig_m

# There are several ways to confirm the equality of data and data_inv.
#    - You can print both vectors and inspect them.
#    - You can subtract them to see that their difference is zero.
#    - You can plot them on top of each other and see that they overlap (what I␣
↪do below).

plt.figure(figsize=(8,4))
plt.plot(data,'ks',markersize=12,markerfacecolor='w',label='Original')
plt.plot(data,'kx',markersize=5,markerfacecolor='w',label='Back-transformed')

plt.legend()
plt.xlabel('Data index')
plt.ylabel('Data value')
plt.show()
```



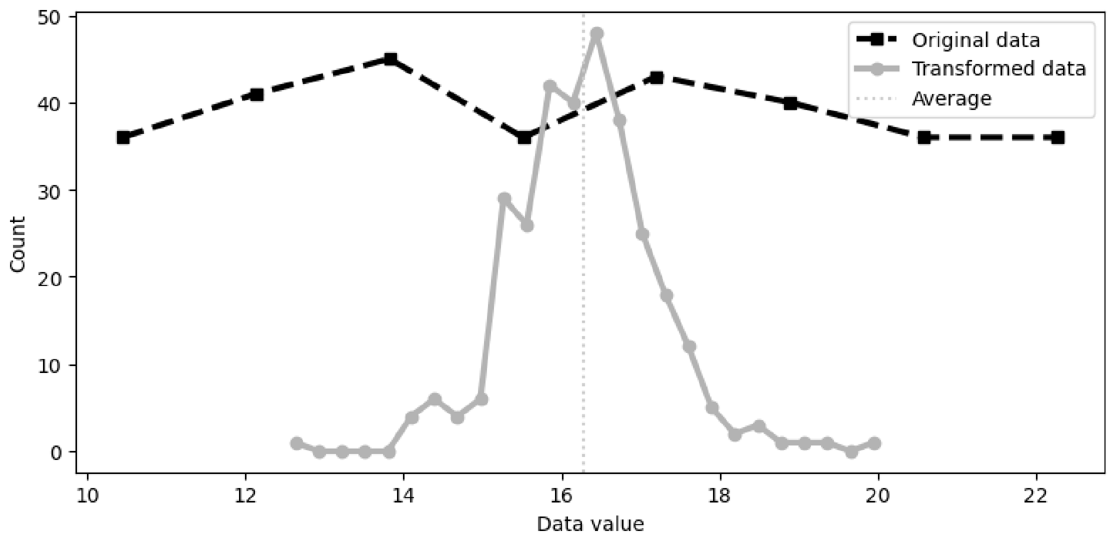
20 Exercise 4

[28]:
```
# parameters
N = 313
lo_bnd = 3*np.pi
hi_bnd = np.exp(np.pi)

# create data
data = np.random.uniform(lo_bnd,hi_bnd,size=N)

### my algorithm
# step 1: scale to [-1,1] (but not exactly -1 or +1 b/c of domain restrictions)
dataScale = my_minmaxScalar(data,-.999,.999) # using function from ex1

# step 2: Fisher-z to transform to normal
dataFish = np.arctanh(dataScale)

# step 3: shift mean
# (min/max scaling doesn't guarantee a particular mean,
#  so I mean-centered the data before shifting to the original mean)
dataFinal = dataFish - np.mean(dataFish) + (lo_bnd+hi_bnd)/2

# report the expected and empirical means
print(f'Expected mean : {(lo_bnd+hi_bnd)/2:.3f}')
print(f'Empirical mean: {np.mean(dataFinal):.3f}')
```

```
Expected mean : 16.283
Empirical mean: 16.283
```

[29]:
```
# create histogram and plot the distributions

# distribution values
y0,x0 = np.histogram(data,bins='fd')
yF,xF = np.histogram(dataFinal,bins='fd')

# bin centers
x0 = (x0[1:]+x0[:-1])/2
xF = (xF[1:]+xF[:-1])/2

# plot the histograms
plt.figure(figsize=(8,4))
plt.plot(x0,y0,'ks--',linewidth=3,label='Original data')
plt.plot(xF,yF,'o-',color=(.7,.7,.7),linewidth=3,label='Transformed data')

# overlay the means
plt.axvline(x=np.mean(data),color=(.8,.8,.8), linestyle=':',label='Average')

# flotsam and jetsam
```

```
plt.xlabel('Data value')
plt.ylabel('Count')
plt.legend()

plt.tight_layout()
#plt.savefig('trans_ex4.png')
plt.show()
```



[ ]:
```
# Observation: It's interesting that the Fisher-z transform often stretches out
# the data compared to its original uniform distribution, but here the␣
 ↪transformed
# distribution is narrower. That happens because the data are first compressed␣
 ↪to a
# range of [-1,1], and most of the data values lie towards zero.
#
# Oftentimes in statistics, the shape of the distribution is more important than
# the numerical range of the distribution.
```

21 Exercise 5

[30]:
```
# z-score a simple array
data = np.arange(3,10)

# compute z-score manually and via scipy.stats
data_z_me = (data-np.mean(data)) / np.std(data,ddof=1)
data_z_sp = stats.zscore(data,ddof=1) # note the ddof parameter!

# print the results to confirm they match
```

```
print(data_z_me)
print(data_z_sp)
```

```
[-1.38873015 -0.9258201  -0.46291005  0.          0.46291005  0.9258201
  1.38873015]
[-1.38873015 -0.9258201  -0.46291005  0.          0.46291005  0.9258201
  1.38873015]
```

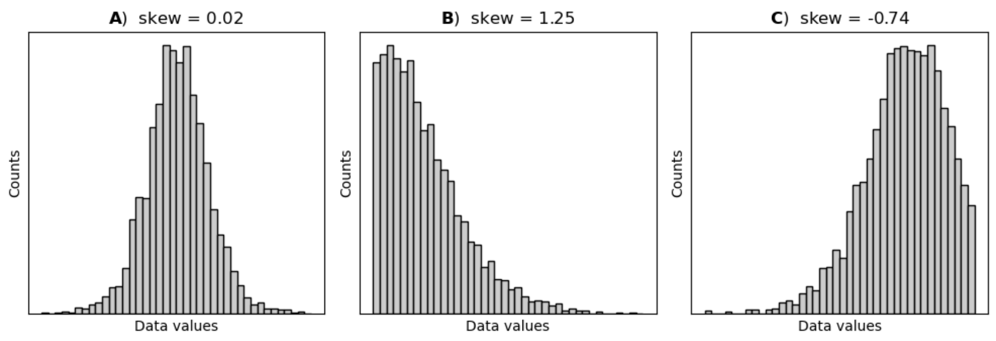
[31]:
```
# z-score columns vs whole matrix.
data = np.arange(0,12).reshape(4,3)**2

data_z_col = stats.zscore(data,ddof=1,axis=0) # axis=0 is the default; I'm␣
 ↪writing here so it's clear to you
data_z_mat = stats.zscore(data,ddof=1,axis=None) # gets the mean/std from the␣
 ↪whole matrix

print('Original data matrix:')
print(data)

print(' ')
print('Column-wise z-scoring:')
print(data_z_col)

print(' ')
print('Matrix-wise z-scoring:')
print(data_z_mat)
```

```
Original data matrix:
[[  0   1   4]
 [  9  16  25]
 [ 36  49  64]
 [ 81 100 121]]

Column-wise z-scoring:
[[-0.8660254  -0.92356057 -0.96284553]
 [-0.61858957 -0.5815011  -0.55436561]
 [ 0.12371791  0.17102973  0.20423996]
 [ 1.36089706  1.33403193  1.31297117]]

Matrix-wise z-scoring:
[[-1.02440065 -1.00010656 -0.9272243 ]
 [-0.80575387 -0.63569526 -0.41704848]
 [-0.14981353  0.16600959  0.53042089]
 [ 0.94342036  1.405008    1.91518381]]
```

22 Exercise 6

[ ]:
```
# The insight here is that the Fisher-z transform is nonlinear,
# and has a bigger impact on data that are closer to |1|.
# This means you can create skew from a uniform distribution
# with asymmetric boundaries.
```

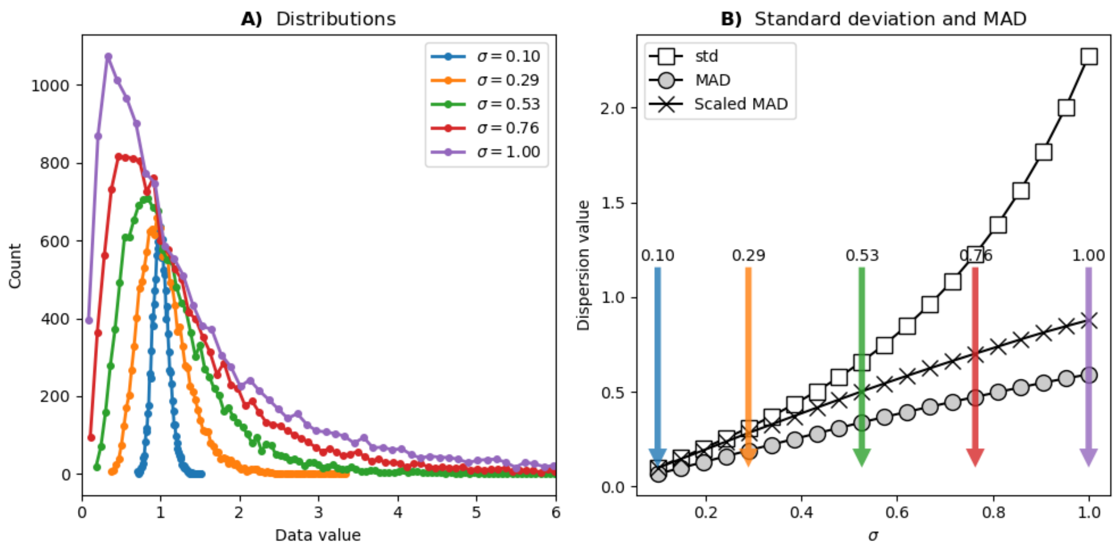
[32]:
```
# no skew
X1 = np.arctanh(np.random.uniform(-.999,.999,size=5000))

# strong positive skew
X2 = np.arctanh(np.random.uniform(-.2,.999,size=5000))
# slight negative skew
X3 = np.arctanh(np.random.uniform(-.999,.8,size=5000))
# compute the empirical skews
skews = [0]*3
skews[0] = stats.skew(X1)
skews[1] = stats.skew(X2)
skews[2] = stats.skew(X3)

_,axs = plt.subplots(1,3,figsize=(10,3.5))

axs[0].hist(X1,40,color=(.8,.8,.8),edgecolor='k')
axs[0].set_title(f'$\\bf{{A}}$)  skew = {skews[0]:.2f}')

axs[1].hist(X2,40,color=(.8,.8,.8),edgecolor='k')
axs[1].set_title(f'$\\bf{{B}}$)  skew = {skews[1]:.2f}')

axs[2].hist(X3,40,color=(.8,.8,.8),edgecolor='k')
axs[2].set_title(f'$\\bf{{C}}$)  skew = {skews[2]:.2f}')

for a in axs:
  a.set(yticks=[],ylabel='Counts',xticks=[],xlabel='Data values')

plt.tight_layout()
#plt.savefig('trans_ex6.png')
plt.show()
```



23 Exercise 7

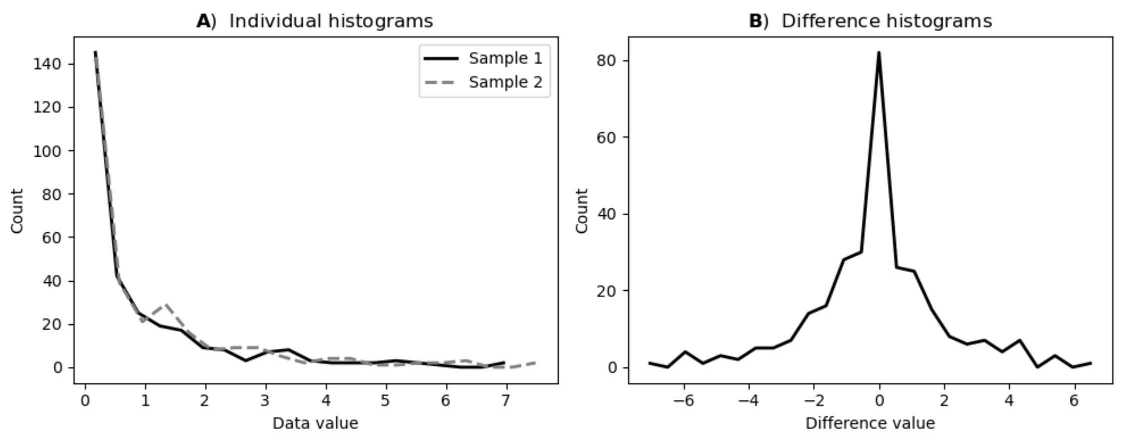
[33]:
```
# generate the data
sigmas = np.linspace(.1,1,20)
X = np.random.randn(13524)

# initialize the data results matrices
M = np.zeros((len(sigmas),2))
data = [0]*len(sigmas)

# compute and store all moments in a matrix
for i,s in enumerate(sigmas):

  # create the data
  data[i] = np.exp(X*s)

  # compute its stadard deviation
  M[i,0] = np.std( data[i] ,ddof=1)

  # compute its MAD
  M[i,1] = np.median( np.abs(data[i] - np.median(data[i])) )
```

[34]:
```
_,axs = plt.subplots(1,2,figsize=(10,5))

# plot the moments
m = 'osp*' # markers
l = ['-','--',':','-.'] # line type
c = [0,.2,.4,.6] # grayscale

# plot the std and MAD (plotted first for the vertical line heights)
axs[1].plot(sigmas,M[:,0],'ks-',markersize=10,markerfacecolor='w',label='std')
axs[1].plot(sigmas,M[:,1],'ko-',markersize=10,markerfacecolor=(.8,.8,.
 ↪8),label='MAD')
axs[1].plot(sigmas,M[:,1]/stats.norm.ppf(3/
 ↪4),'kx-',markersize=10,markerfacecolor=(.8,.8,.8),label='Scaled MAD')
axs[1].legend()
axs[1].set(xlabel=r'$\sigma$',ylabel='Dispersion value')
axs[1].set_title(r'$\bf{B)}$  Standard deviation and MAD')

# now plot selected distributions
for i in np.linspace(0,len(sigmas)-1,5).astype(int):

  # get histogram values
  y,x = np.histogram(data[i],bins='fd')
  # plot as line
  h = axs[0].plot((x[:-1]+x[1:])/2,y,'.-',linewidth=2,markersize=8,
              label=f'$\sigma={sigmas[i]:.2f}$')

  # add the vertical lines in the moments plot
  axs[1].annotate(f'{sigmas[i]:.2f}',xy=(sigmas[i],.
↪1),horizontalalignment='center',fontsize=10,
                  xytext=(sigmas[i],axs[1].get_ylim()[1]/
↪2),arrowprops=dict(width=4,linewidth=0,color=h[0].get_color(),alpha=.8))

# some niceties
axs[0].legend()
axs[0].set(xlim=[0,6],xlabel='Data value',ylabel='Count')
axs[0].set_title(r'$\bf{A)}$  Distributions')

plt.tight_layout()
#plt.savefig('trans_ex7.png')
plt.show()
```



24 Exercise 8

[35]:
```python
# create two non-normal distributions
N = 300
sample1 = np.random.randn(N)**2
sample2 = np.random.randn(N)**2

# and their difference
difference = sample1 - sample2

# compute their histograms
y1,x1 = np.histogram(sample1,bins='fd')
y2,x2 = np.histogram(sample2,bins='fd')
yd,xd = np.histogram(difference,bins='fd')

# and plot
_,axs = plt.subplots(1,2,figsize=(10,4))

axs[0].plot((x1[:-1]+x1[1:])/2,y1,'k',linewidth=2,label='Sample 1')
axs[0].plot((x2[:-1]+x2[1:])/2,y2,'--',color='gray',linewidth=2,label='Sample 2')
axs[0].set(xlabel='Data value',ylabel='Count')
axs[0].set_title(r'$\bf{A}$)  Individual histograms')
axs[0].legend()

axs[1].plot((xd[:-1]+xd[1:])/2,yd,'k',linewidth=2)
axs[1].set(xlabel='Difference value',ylabel='Count')
axs[1].set_title(r'$\bf{B}$)  Difference histograms')

plt.tight_layout()
#plt.savefig('trans_ex8.png')
plt.show()
```


A) Individual histograms
Sample 1
Sample 2
Count
Data value
B) Difference histograms
Count
Difference value