stats_ch04_descriptives

August 1, 2024

1 Modern statistics: Intuition, Math, Python, R

1.1 Mike X Cohen (sincxpress.com)

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

Code for chapter 4 (descriptives)

---

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.

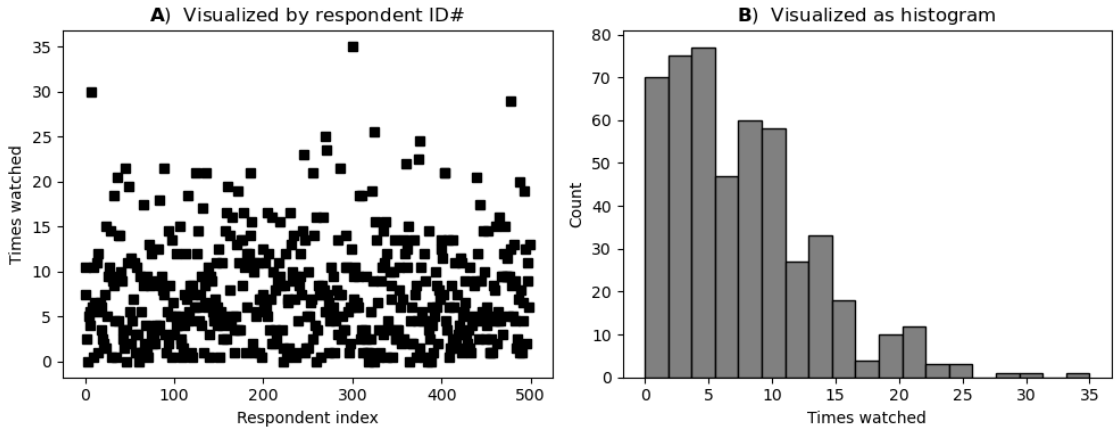
2.0.2 Solutions to all exercises are at the bottom of the notebook.

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 pandas as pd
import seaborn as sns
import scipy.stats as stats

import matplotlib.pyplot as plt

# define global figure properties used for publication
import matplotlib_inline.backend_inline
#matplotlib_inline.backend_inline.set_matplotlib_formats('svg') # display␣
 ↪figures in vector format
#plt.rcParams.update({'font.size':14,             # font size
#                     'savefig.dpi':300,          # output resolution
#                     'axes.titlelocation':'left',# title location
#                     'axes.spines.right':False,  # remove axis bounding box
#                     'axes.spines.top':False,    # remove axis bounding box
#                     })
```

3 Figure 4.2: Gangnam style video watching data

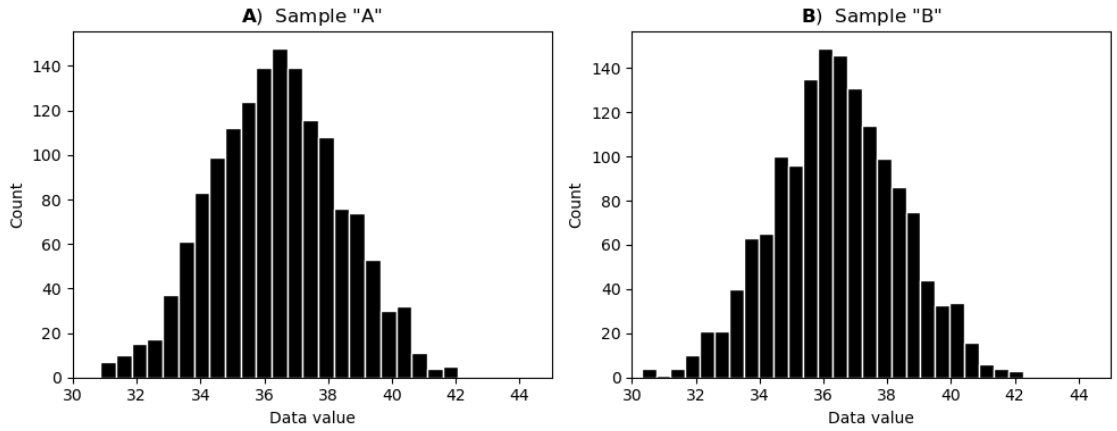
[2]:
```
# generate some data
timesWatched = np.round( np.abs(np.random.randn(500)*20) )/2
# force an outliner
timesWatched[300] = 35

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

axs[0].plot(timesWatched,'ks')
axs[0].set_xlabel('Respondent index')
axs[0].set_ylabel('Times watched')
axs[0].set_title(r'$\bf{A}$)  Visualized by respondent ID#')

axs[1].hist(timesWatched,bins='fd',color='gray',edgecolor='k')
axs[1].set_xlabel('Times watched')
axs[1].set_ylabel('Count')
axs[1].set_title(r'$\bf{B}$)  Visualized as histogram')

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



4 Figure 4.3: Two samples from the same population have similar distributions

[3]:
```
sampleA = np.random.randn(1500)*2 + np.pi**np.pi
sampleB = np.random.randn(1500)*2 + np.pi**np.pi

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

```
axs[0].hist(sampleA,bins='fd',color='k',edgecolor='w')
axs[0].set(xlabel='Data value',ylabel='Count',xlim=[30,45])
axs[0].set_title(r'$\bf{A}$)  Sample "A"')

axs[1].hist(sampleB,bins='fd',color='k',edgecolor='w')
axs[1].set(xlabel='Data value',ylabel='Count',xlim=[30,45])
axs[1].set_title(r'$\bf{B}$)  Sample "B"')

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



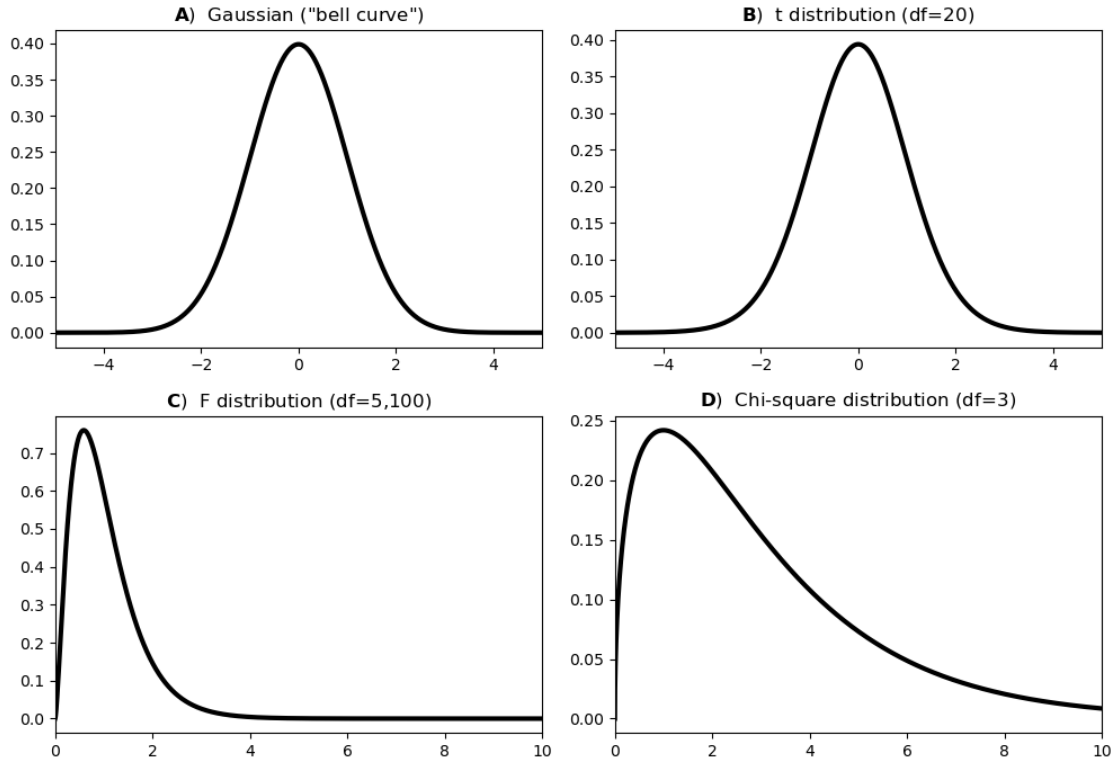
5 Figure 4.4: Analytical Gaussian

[ ]:
```
# Code isn't shown here, because it's part of Exercise 1 :P
```

6 Figure 4.5: Examples of distributions

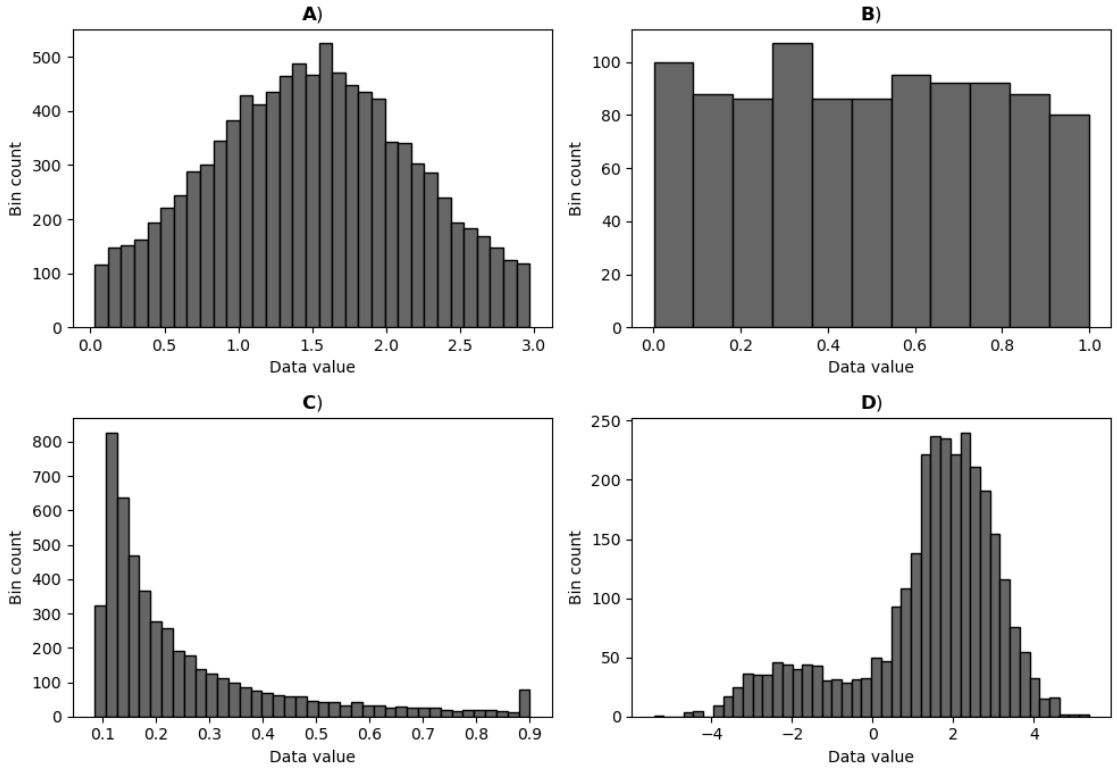
[4]:
```
x = np.linspace(-5,5,10001)

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

# Gaussian
y = stats.norm.pdf(x) # stats.norm.pdf(x*1.5-2.7) + stats.norm.pdf(x*1.5+2.7)
axs[0,0].plot(x,y,'k',linewidth=3)
axs[0,0].set_title(r'$\bf{A}$)  Gaussian ("bell curve")')
axs[0,0].set_xlim(x[[0,-1]])

# T
axs[0,1].plot(x,stats.t.pdf(x,20),'k',linewidth=3)
```

```
axs[0,1].set_title(r'$\bf{B}$)  t distribution (df=20)')
axs[0,1].set_xlim(x[[0,-1]])

# F
x = np.linspace(0,10,10001)
axs[1,0].plot(x,stats.f.pdf(x,5,100),'k',linewidth=3)
axs[1,0].set_title(r'$\bf{C}$)  F distribution (df=5,100)')
axs[1,0].set_xlim(x[[0,-1]])

# Chi
axs[1,1].plot(x,stats.chi2.pdf(x,3),'k',linewidth=3)
axs[1,1].set_title(r'$\bf{D}$)  Chi-square distribution (df=3)')
axs[1,1].set_xlim(x[[0,-1]])

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



7 Figure 4.6: Examples of empirical distributions

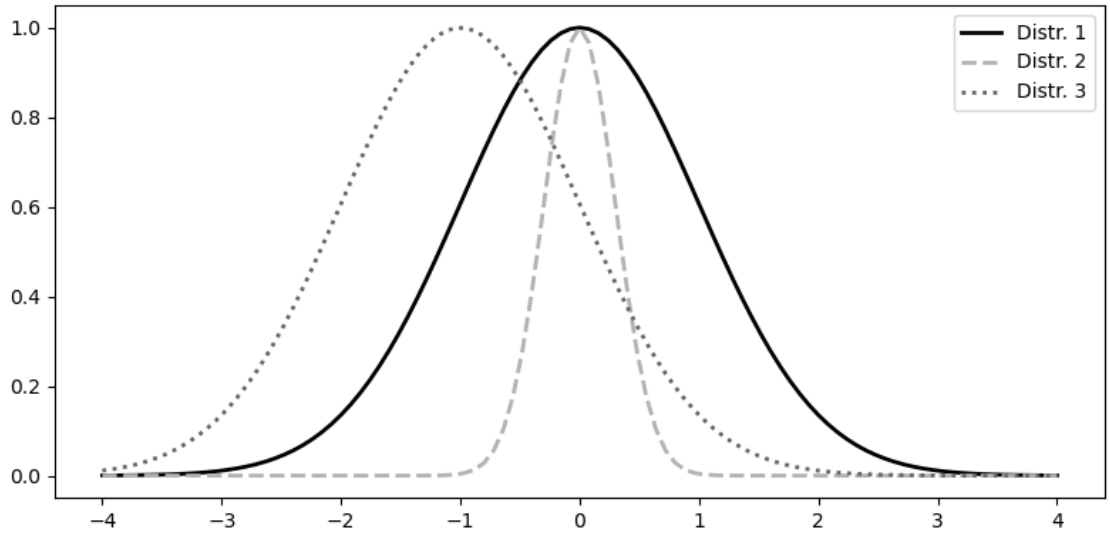
[5]:
```
# bimodal
_,axs = plt.subplots(2,2,figsize=(10,7))


# normal distribution with kurtosis
X = np.arctanh(np.random.rand(10000)*1.8-.9) + 1.5
axs[0,0].hist(X,bins='fd',color=(.4,.4,.4),edgecolor='k')
axs[0,0].set_xlabel('Data value')
axs[0,0].set_ylabel('Bin count')
axs[0,0].set_title(r'$\bf{A}$)')


# uniform distribution
X = np.random.rand(1000)
axs[0,1].hist(X,bins='fd',color=(.4,.4,.4),edgecolor='k')
axs[0,1].set_xlabel('Data value')
axs[0,1].set_ylabel('Bin count')
axs[0,1].set_title(r'$\bf{B}$)')



# power distribution
f = np.linspace(1,10,5001)
X = 1/f + np.random.randn(len(f))/200
X[X>.9] = .9 # some clipping
axs[1,0].hist(X,bins='fd',color=(.4,.4,.4),edgecolor='k')
axs[1,0].set_xlabel('Data value')
axs[1,0].set_ylabel('Bin count')
axs[1,0].set_title(r'$\bf{C}$)')


# bimodal distribution
x1 = np.random.randn(500) - 2
x2 = np.random.randn(2500) + 2
X = np.concatenate((x1,x2))
axs[1,1].hist(X,bins='fd',color=(.4,.4,.4),edgecolor='k')
axs[1,1].set_xlabel('Data value')
axs[1,1].set_ylabel('Bin count')
axs[1,1].set_title(r'$\bf{D}$)')



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



8 Figure 4.7: Characteristics of distributions

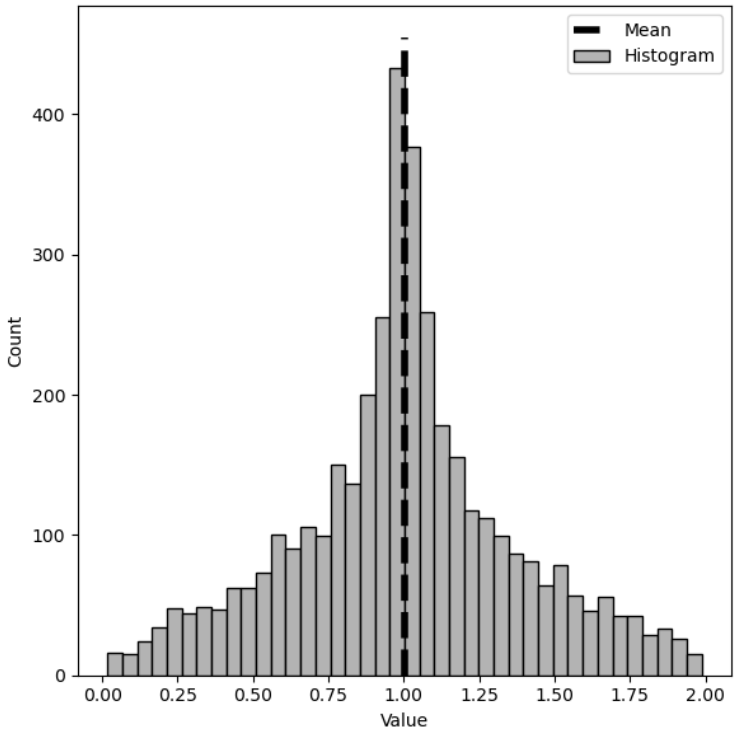
[6]:
```
# function variable
x = np.linspace(-4,4,101)

ss = [ 1,.3,1 ]
cs = [ 0,0,-1 ]

colors = [ 'k',(.7,.7,.7),(.4,.4,.4) ]
styles = [ '-','--',':' ]

plt.figure(figsize=(8,4))

for s,c,col,sty in zip(ss,cs,colors,styles):

  # create gaussian
  gaus = np.exp( -(x-c)**2 / (2*s**2) )
  plt.plot(x,gaus,color=col,linewidth=2,linestyle=sty)

plt.legend(['Distr. 1','Distr. 2','Distr. 3'])
plt.tight_layout()
```

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



9 Figure 4.9: Histogram showing mean

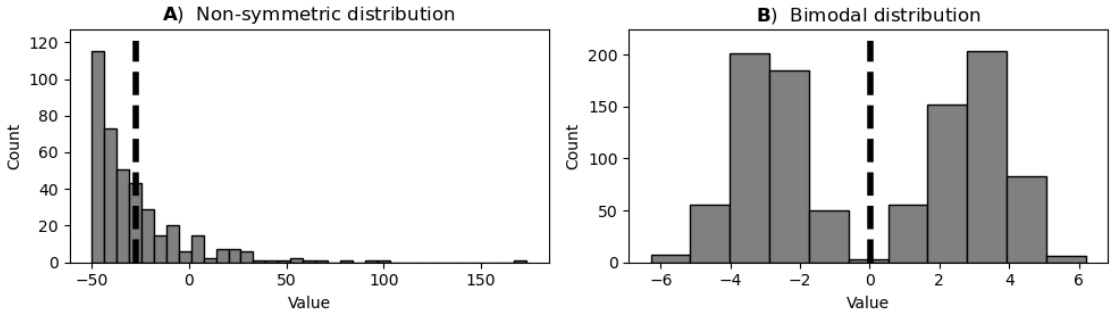
[7]:
```
# generate a Laplace distribution
x1 = np.exp(-np.abs(3*np.random.randn(4000)))
x2 = np.exp(-np.abs(3*np.random.randn(4000)))
x = x1-x2+1

# and compute its mean
xBar = np.mean(x)

# histogram
plt.figure(figsize=(6,6))
plt.hist(x,bins='fd',color=(.7,.7,.7),edgecolor='k')

# vertical line for mean
plt.plot([xBar,xBar],plt.gca().get_ylim(),'--',color='k',linewidth=4)

plt.legend(['Mean','Histogram'])
plt.xlabel('Value')
plt.ylabel('Count')

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



10 Figure 4.10: "Failure" scenarios for the mean

[8]:
```
_,axs = plt.subplots(1,2,figsize=(10,3))


## case 1: mean does not reflect the most common value
data = [0,0]
data[0] = (np.random.randn(400)+2.5)**3 - 50

## case 2: bimodal distribution
x1 = np.random.randn(500) - 3
x2 = np.random.randn(500) + 3
data[1] = np.concatenate((x1,x2))


# histograms and means
for i in range(2):

  # data average
  xBar = np.mean(data[i])

  # histogram with vertical line for mean
```

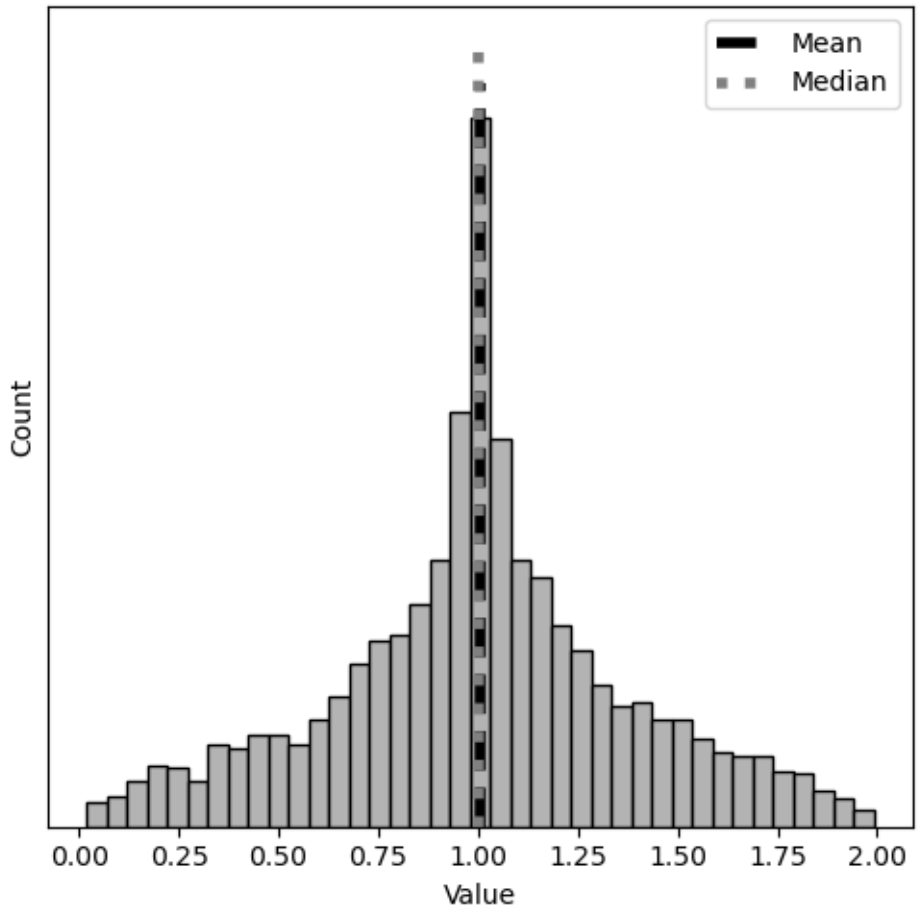
```
  axs[i].hist(data[i],bins='fd',color='gray',edgecolor='k')
  axs[i].plot([xBar,xBar],axs[i].get_ylim(),'--',color='k',linewidth=4)
  axs[i].set_xlabel('Value')
  axs[i].set_ylabel('Count')


axs[0].set_title(r'$\bf{A}$)  Non-symmetric distribution')
axs[1].set_title(r'$\bf{B}$)  Bimodal distribution')

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



11 Figure 4.11: Median and mean

[9]:
```
# generate a Laplace distribution
x1 = np.exp(-np.abs(3*np.random.randn(4000)))
x2 = np.exp(-np.abs(3*np.random.randn(4000)))
x = x1-x2+1

# and compute its mean
mean = np.mean(x)
median = np.median(x)

# histogram
fig = plt.figure(figsize=(5,5))
plt.hist(x,bins='fd',color=(.7,.7,.7),edgecolor='k')

# vertical lines for mean and median
plt.plot([mean,mean],plt.gca().get_ylim(),'--',
         color='k',linewidth=4,label='Mean')
plt.plot([median,median],plt.gca().get_ylim(),':',
         color='gray',linewidth=4,label='Median')
```

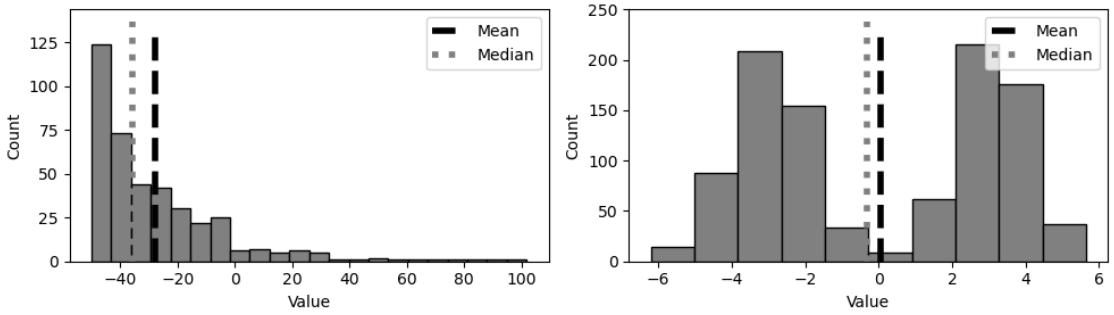
```
plt.legend()
plt.xlabel('Value')
plt.ylabel('Count')
plt.yticks([])

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



12 Figure 4.12: "Failures" of the median

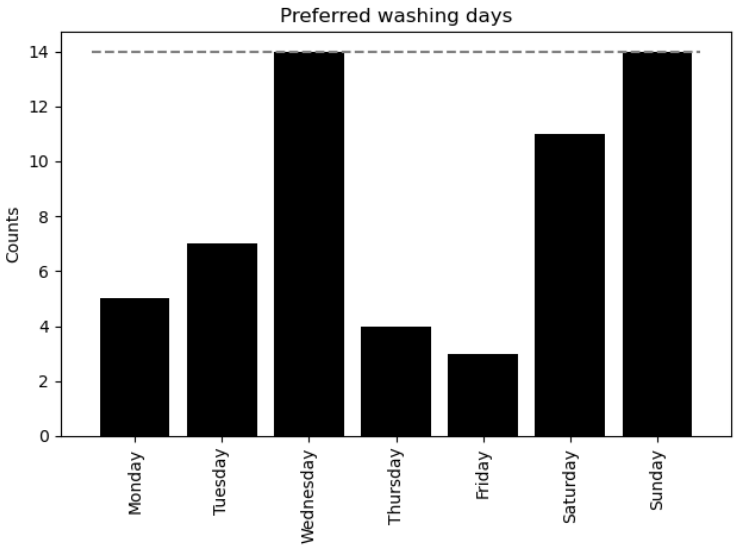
[10]:
```
_,axs = plt.subplots(1,2,figsize=(10,3))


## case 1: mean does not reflect
data = [0,0]
data[0] = (np.random.randn(400)+2.5)**3 - 50
```

```
## case 2: bimodal distribution
x1 = np.random.randn(500) - 3
x2 = np.random.randn(500) + 3
data[1] = np.concatenate((x1,x2))


# histograms and means
for i in range(2):

  # data average
  mean = np.mean(data[i])
  median = np.median(data[i])

  # histogram with vertical line for mean
  axs[i].hist(data[i],bins='fd',color='gray',edgecolor='k')
  axs[i].plot([mean,mean],axs[i].get_ylim(),'--',
          color='k',linewidth=4,label='Mean')
  axs[i].plot([median,median],axs[i].get_ylim(),':',
          color='gray',linewidth=4,label='Median')
  axs[i].set_xlabel('Value')
  axs[i].set_ylabel('Count')
  axs[i].legend()


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



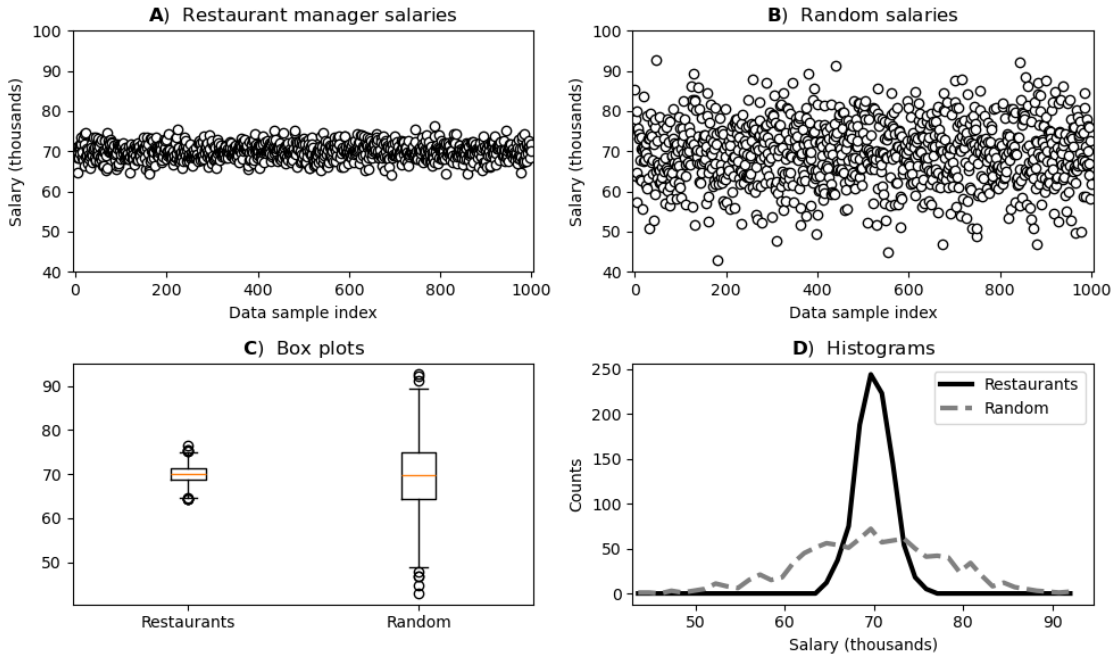
13 Figure 4.13: Mode

[11]:
```
# the data
data = {
    'Monday'    :  5,
    'Tuesday'   :  7,
    'Wednesday': 14,
    'Thursday'  :  4,
    'Friday'    :  3,
    'Saturday'  : 11,
    'Sunday'    : 14
}


# compute the mode
mode = stats.mode(list(data.values()),keepdims=True)
mode[0]


# plot the data and mode
plt.bar(range(len(data)),data.values(),color='k')
plt.plot([-.5,len(data)-.5],[mode[0],mode[0]],'--',color='gray')
plt.xticks(range(len(data)),labels=data.keys(),rotation=90)
plt.yticks(range(0,15,2))
plt.ylabel('Counts')
plt.title('Preferred washing days',loc='center')

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



14 Figure 4.14: Dispersion

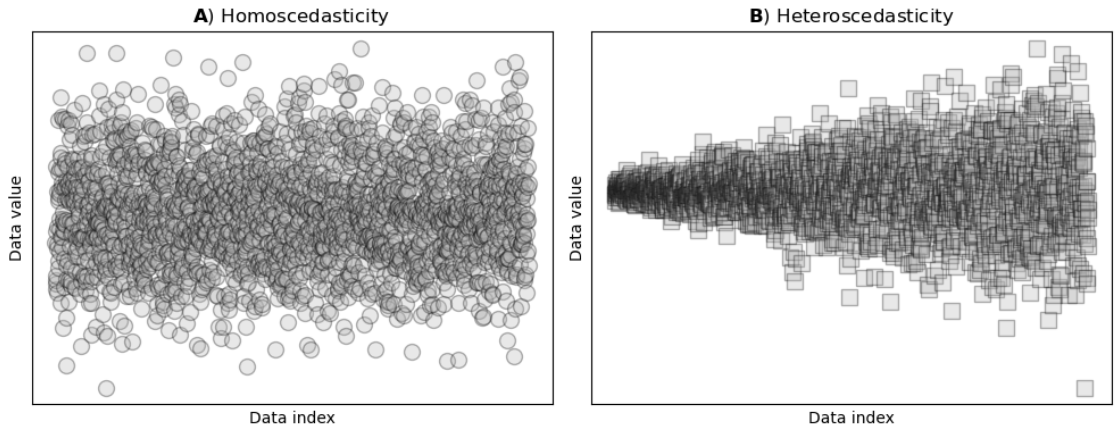
[12]:
```
# restaurant managers salaries and randomly selected salaries
X1 = np.random.randn(1000)*2 + 70
X2 = np.random.randn(1000)*8 + 70

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

axs[0,0].plot(X1,'ko',markerfacecolor='w')
axs[0,0].set_ylim([40,100])
axs[0,0].set_xlim([-5,len(X1)+5])
axs[0,0].set_title(r"$\bf{A}$)  Restaurant manager salaries")
axs[0,0].set_ylabel('Salary (thousands)')
axs[0,0].set_xlabel('Data sample index')


axs[0,1].plot(X2,'ko',markerfacecolor='w')
axs[0,1].set_ylim([40,100])
axs[0,1].set_xlim([-5,len(X2)+5])
axs[0,1].set_title(r"$\bf{B}$)  Random salaries")
axs[0,1].set_ylabel('Salary (thousands)')
axs[0,1].set_xlabel('Data sample index')



axs[1,0].boxplot([X1,X2])
axs[1,0].set_xticklabels(['Restaurants','Random'])
axs[1,0].set_title(r'$\bf{C}$)  Box plots')



y1,x = np.histogram(X1,bins=np.linspace(np.min(X2),np.max(X2),41))
y2,x = np.histogram(X2,bins=np.linspace(np.min(X2),np.max(X2),41))
x = (x[:-1]+x[1:])/2
axs[1,1].plot(x,y1,'k',linewidth=3,label='Restaurants')
axs[1,1].plot(x,y2,'--',color='gray',linewidth=3,label='Random')
axs[1,1].set_xlim([np.min(X2),np.max(X2)+2])
axs[1,1].legend()
axs[1,1].set_xlabel('Salary (thousands)')
axs[1,1].set_ylabel('Counts')
axs[1,1].set_title(r'$\bf{D}$)  Histograms')

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



15 Figure 4.16: Homo/heteroscedasticity

[13]:
```
# sample size and x-axis grid
N = 2345
x = np.linspace(1,10,N)

# generate some data
ho = np.random.randn(N)
he = np.random.randn(N) * x


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

axs[0].plot(x,ho,'ko',markersize=10,markerfacecolor=(.7,.7,.7),alpha=.3)
axs[0].set(xlabel='Data index',xticks=[],yticks=[],ylabel='Data value')
axs[0].set_title(r'$\bf{A}$) Homoscedasticity')

axs[1].plot(x,he,'ks',markersize=10,markerfacecolor=(.7,.7,.7),alpha=.3)
axs[1].set(xlabel='Data index',xticks=[],yticks=[],ylabel='Data value')
axs[1].set_title(r'$\bf{B}$) Heteroscedasticity')

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

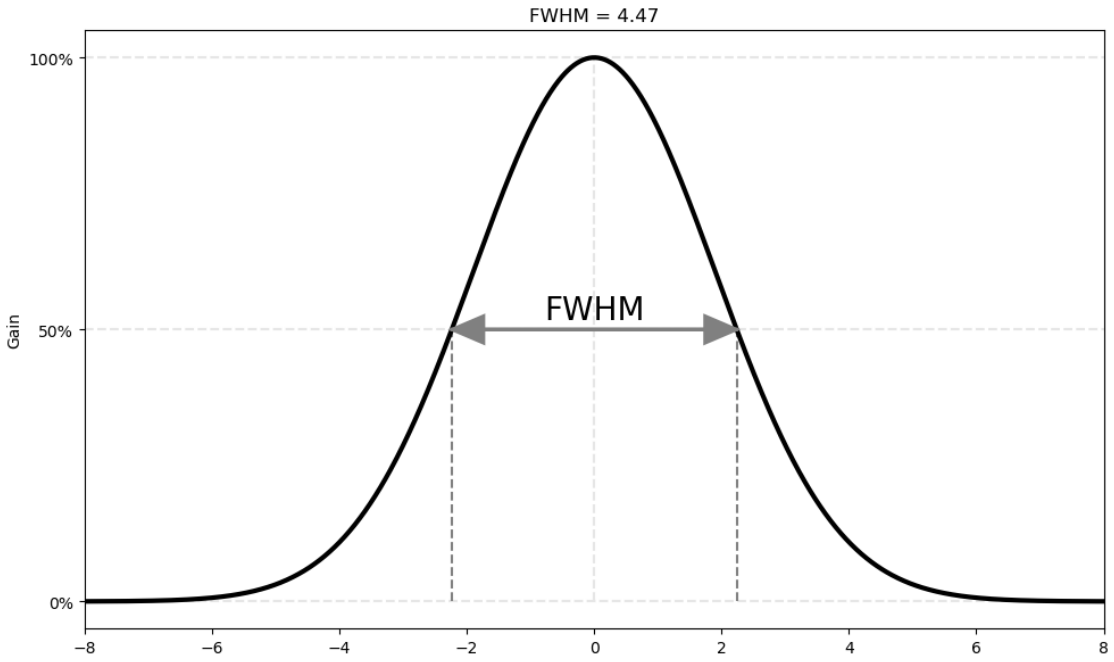
```
plt.show()
```



16 Figure 4.17: FWHM

[14]:
```
# try on pdf to comapre with analytic
x = np.linspace(-8,8,1001)
s = 1.9

# create the Gaussian and compute its analytic FWHM
pureGaus = np.exp( (-x**2)/(2*s**2) )
fwhm = 2*s*np.sqrt(2*np.log(2))

plt.figure(figsize=(10,6))

# plot guide lines
plt.plot(x[[0,-1]],[.5,.5],'--',color=(.9,.9,.9))
plt.plot(x[[0,-1]],[1,1],'--',color=(.9,.9,.9))
plt.plot(x[[0,-1]],[0,0],'--',color=(.9,.9,.9))
plt.plot([0,0],[0,1],'--',color=(.9,.9,.9))
plt.plot([-fwhm/2,-fwhm/2],[0,.5],'--',color=(.5,.5,.5))
plt.plot([fwhm/2,fwhm/2],[0,.5],'--',color=(.5,.5,.5))

# plot the gaussian
plt.plot(x,pureGaus,'k',linewidth=3)

# plot arrows
plt.arrow(-fwhm/2,.5,fwhm,0, color=(.5,.5,.5),linewidth=2,zorder=10,
          head_width=.05,head_length=.5,length_includes_head=True)
plt.arrow(fwhm/2,.5,-fwhm,0, color=(.5,.5,.5),linewidth=2,zorder=10,
          head_width=.05,head_length=.5,length_includes_head=True)
```

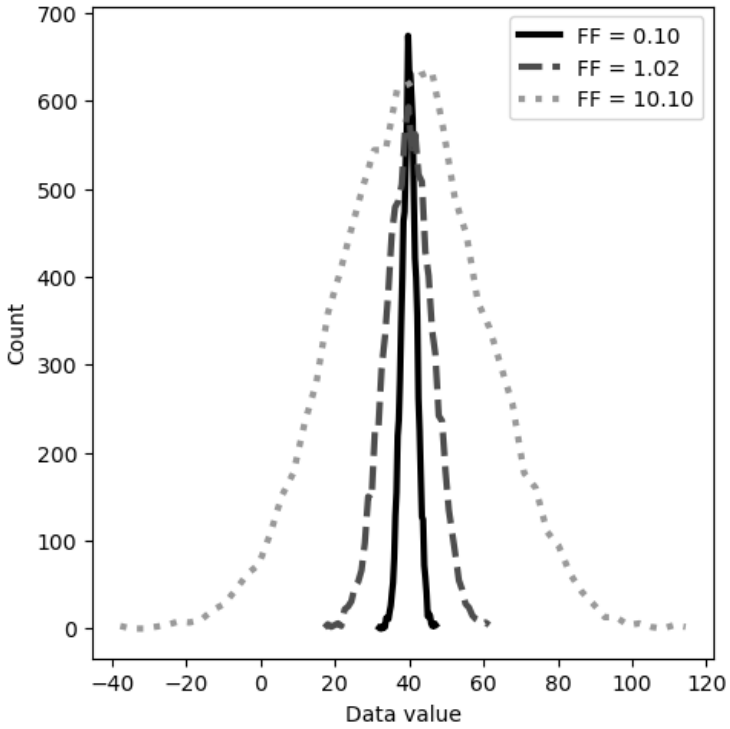
```
plt.text(0,.52,'FWHM',horizontalalignment='center',fontsize=20)
plt.xlim(x[[0,-1]])
plt.yticks([0,.5,1],labels=['0%','50%','100%'])
plt.ylabel('Gain')
plt.title(f'FWHM = {fwhm:.2f}',loc='center')

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



17 Figure 4.18: Fano factor

[15]:
```
# keep mean fixed while varying sigma and show histograms

fanos = [ .1,1,10 ]
mean = 40

# deriving the std from the fano factor:
# ff = s^2/m
#  s = sqrt(ff*m)


plt.figure(figsize=(5,5))
lines = [ '-','--',':' ]
```

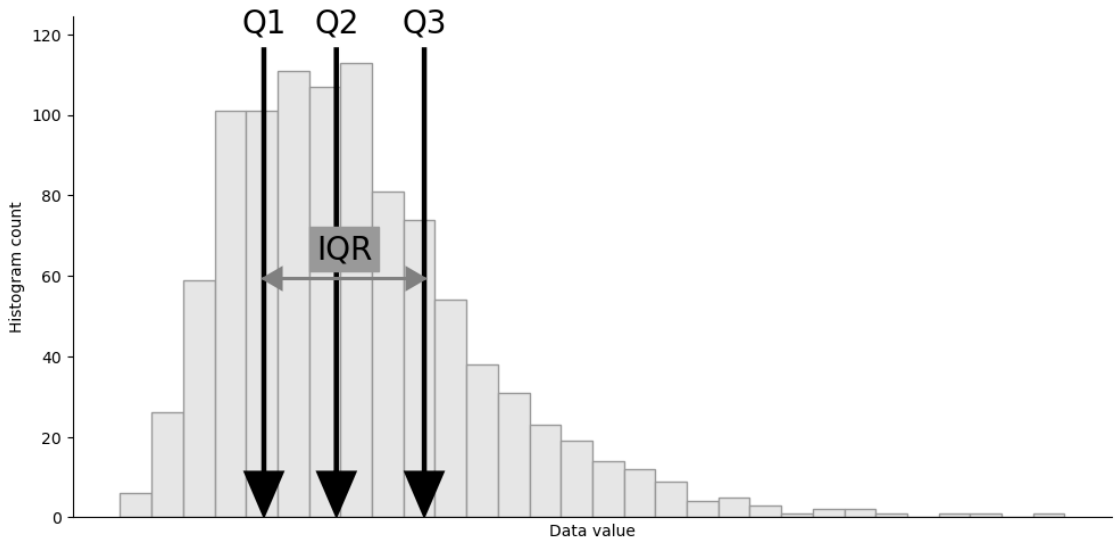
```python
for i in range(len(fanos)):

  # generate data
  sigma = np.sqrt(fanos[i]*mean)
  x = np.random.normal(mean,sigma,size=10000)

  # compute empirical fano factor
  ff = np.var(x,ddof=1) / np.mean(x)

  # get histogram
  yy,xx = np.histogram(x,bins=50)

  # plot the line
  c = i*.9/len(fanos) # grayscale defined by index
  plt.plot((xx[0:-1]+xx[1:])/2,yy,linestyle=lines[i],
           linewidth=3,color=(c,c,c),label=f'FF = {ff:.2f}')


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

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



18 Figure 4.20: IQR

[16]:
```
# Create a dataset to work with
X = np.exp( np.random.randn(1000)/3 )

# and find its quartiles
quartiles = np.percentile(X,[25,50,75])

# plot the histogram
plt.figure(figsize=(10,5))
plt.hist(X,bins='fd',color=(.9,.9,.9),edgecolor=(.6,.6,.6))

# and draw lines for the quartiles
ylim = plt.gca().get_ylim()
for q in quartiles:
  plt.arrow(q,ylim[1],0,-ylim[1]+1, color='k',linewidth=3,
          head_width=.1,head_length=10,length_includes_head=True)

# horizontal arrow
plt.arrow(quartiles[0],ylim[1]/2,quartiles[2]-quartiles[0],0, color=(.5,.5,.
 ↪5),linewidth=2,zorder=10,
          head_width=5,head_length=.05,length_includes_head=True)
plt.arrow(quartiles[2],ylim[1]/2,quartiles[0]-quartiles[2],0, color=(.5,.5,.
 ↪5),linewidth=2,zorder=10,
          head_width=5,head_length=.05,length_includes_head=True)

plt.text(np.mean(quartiles[[0,2]]),ylim[1]/
 ↪2+5,'IQR',horizontalalignment='center',fontsize=20,backgroundcolor=(.6,.6,.6))

for i in range(3):
  plt.
 ↪text(quartiles[i],ylim[1],'Q'+str(i+1),horizontalalignment='center',verticalalignment='bottom


# some additional plotting niceties...
plt.xlabel('Data value')
plt.xticks([])
plt.ylabel('Histogram count')
plt.gca().spines['right'].set_visible(False)
plt.gca().spines['top'].set_visible(False)


plt.tight_layout()
```

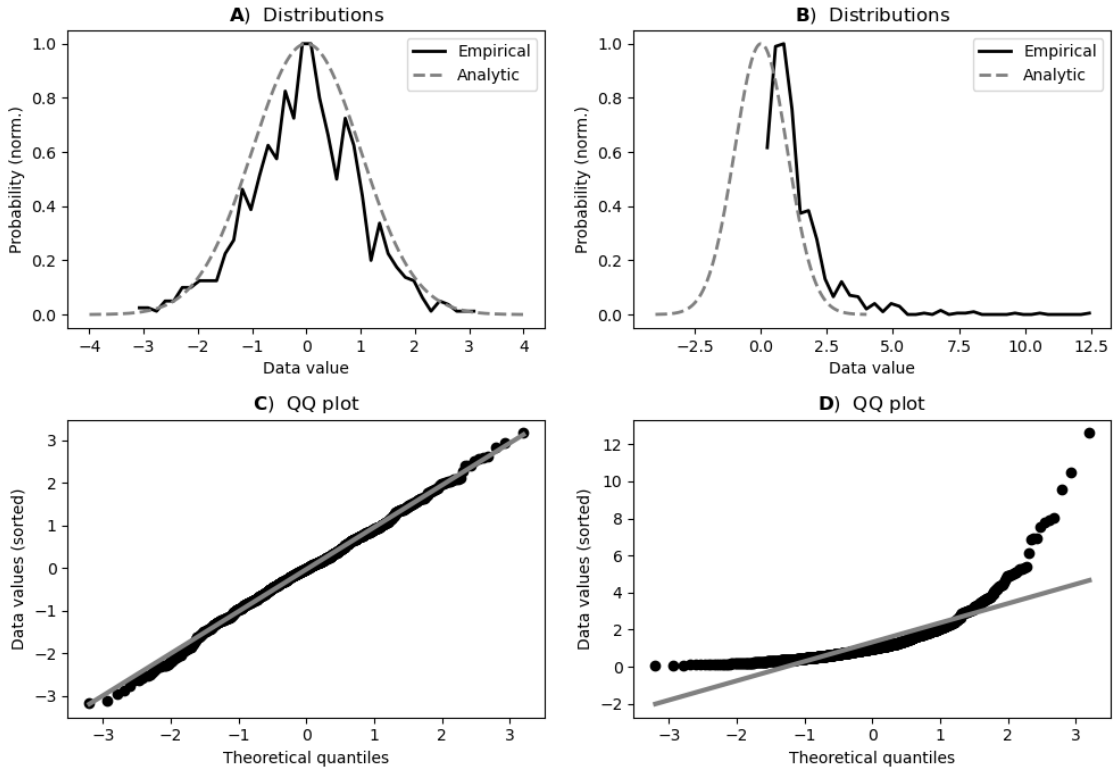
```
#plt.savefig('desc_IQR.png')
plt.show()
```



19 Figure 4.21: Create a QQ plot

[17]:
```
# generate two datasets
N = 1000
d1 = np.random.randn(N) # normal
d2 = np.exp( d1*.8 )    # non-normal


# data for histograms
y1,x1 = np.histogram(d1,bins=40)
y1 = y1/np.max(y1)
x1 = (x1[:-1]+x1[1:])/2

y2,x2 = np.histogram(d2,bins=40)
y2 = y2/np.max(y2)
x2 = (x2[:-1]+x2[1:])/2



# analytic normal distribution
x = np.linspace(-4,4,10001)
norm = stats.norm.pdf(x)
norm = norm/np.max(norm)
```

```
## now generate the plots
_,axs = plt.subplots(2,2,figsize=(10,7))
axs[0,0].plot(x1,y1,'k',linewidth=2,label='Empirical')
axs[0,0].plot(x,norm,'--',color='gray',linewidth=2,label='Analytic')
axs[0,0].legend()
axs[0,0].set_xlabel('Data value')
axs[0,0].set_ylabel('Probability (norm.)')
axs[0,0].set_title(r'$\bf{A}$)  Distributions')

axs[0,1].plot(x2,y2,'k',linewidth=2,label='Empirical')
axs[0,1].plot(x,norm,'--',color='gray',linewidth=2,label='Analytic')
axs[0,1].legend()
axs[0,1].set_xlabel('Data value')
axs[0,1].set_ylabel('Probability (norm.)')
axs[0,1].set_title(r'$\bf{B}$)  Distributions')




# QQ plots
stats.probplot(d1,plot=axs[1,0],fit=True)
stats.probplot(d2,plot=axs[1,1],fit=True)

for i in range(2):
  axs[1,i].get_lines()[0].set(markerfacecolor='k',markeredgecolor='k')
  axs[1,i].get_lines()[1].set(color='gray',linewidth=3)
  axs[1,i].set_title(' ')
  axs[1,i].set_ylabel('Data values (sorted)')

axs[1,0].set_title(r'$\bf{C}$)  QQ plot')
axs[1,1].set_title(r'$\bf{D}$)  QQ plot')

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



20 Figure 4.22: Table for moments

[18]:
```
# table data
colLabs = ['Moment number','Name','Description','Formula']

tableData = [ [ 'First'  ,  'Mean'     ,  'Average'      ,  r'$m_1 =␣
 ↪N^{-1}\sum_{i=1}^N x_i$' ],
              [ 'Second' ,  'Variance' ,  'Dispersion'   ,  r'$m_2 =␣
 ↪N^{-1}\sum_{i=1}^N (x_i-\bar{x})^2$' ],
              [ 'Third'  ,  'Skew'     ,  'Asymmetry'    ,  r'$m_3 =␣
 ↪(N\sigma^3)^{-1}\sum_{i=1}^N (x_i-\bar{x})^3$' ],
              [ 'Fourth' ,  'Kurtosis' ,  'Tail fatness' ,  r'$m_4 =␣
 ↪(N\sigma^4)^{-1}\sum_{i=1}^N (x_i-\bar{x})^4$' ]
]


# draw the table
fig, ax = plt.subplots(figsize=(10,4))
ax.set_axis_off()
ht = ax.table(
        cellText   = tableData,
```

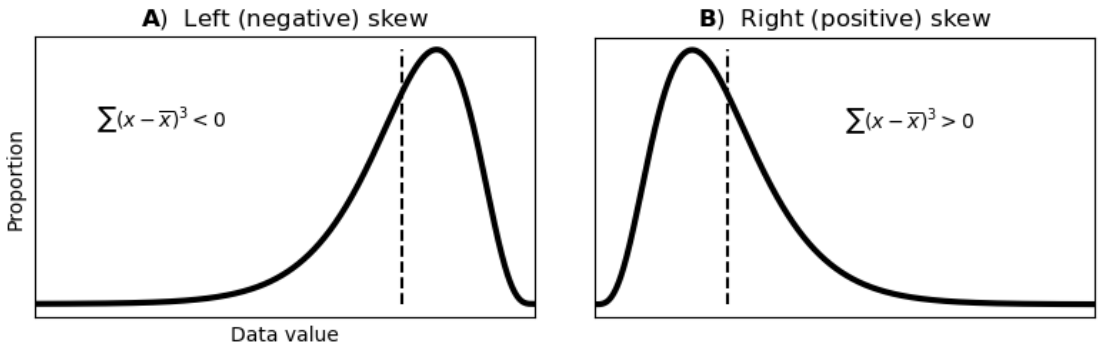
```
        colLabels  = colLabs,
        colColours = [(.8,.8,.8)] * len(colLabs),
        cellLoc    = 'center',
        loc        = 'upper left',
        )


# some adjustments to the fonts etc
ht.scale(1,3.8)
ht.auto_set_font_size(False)
ht.set_fontsize(14)

from matplotlib.font_manager import FontProperties
for (row, col), cell in ht.get_celld().items():
  cell.set_text_props(fontproperties=FontProperties(family='serif'))
  if row==0: cell.
 ↪set_text_props(fontproperties=FontProperties(weight='bold',size=16))
  if col<3 and row>0: cell.set_text_props(fontproperties=FontProperties(size=16))

# export
plt.tight_layout()
#plt.savefig('desc_table_moments1.png', bbox_inches='tight')
plt.show()
```

Moment number	Name	Description	Formula
First	Mean	Average	$m_1 = N^{-1}\sum_{i=1}^{N} x_i$
Second	Variance	Dispersion	$m_2 = N^{-1}\sum_{i=1}^{N} (x_i - \bar{x})^2$
Third	Skew	Asymmetry	$m_3 = (N\sigma^3)^{-1}\sum_{i=1}^{N} (x_i - \bar{x})^3$
Fourth	Kurtosis	Tail fatness	$m_4 = (N\sigma^4)^{-1}\sum_{i=1}^{N} (x_i - \bar{x})^4$

[ ]:
```
## Note about tables, in case you were wondering:
# I find latex-created tables to be ugly and really difficult to customize and␣
 ↪fit on the page.
# Making the tables as a matplotlib figure looks nicer.
```

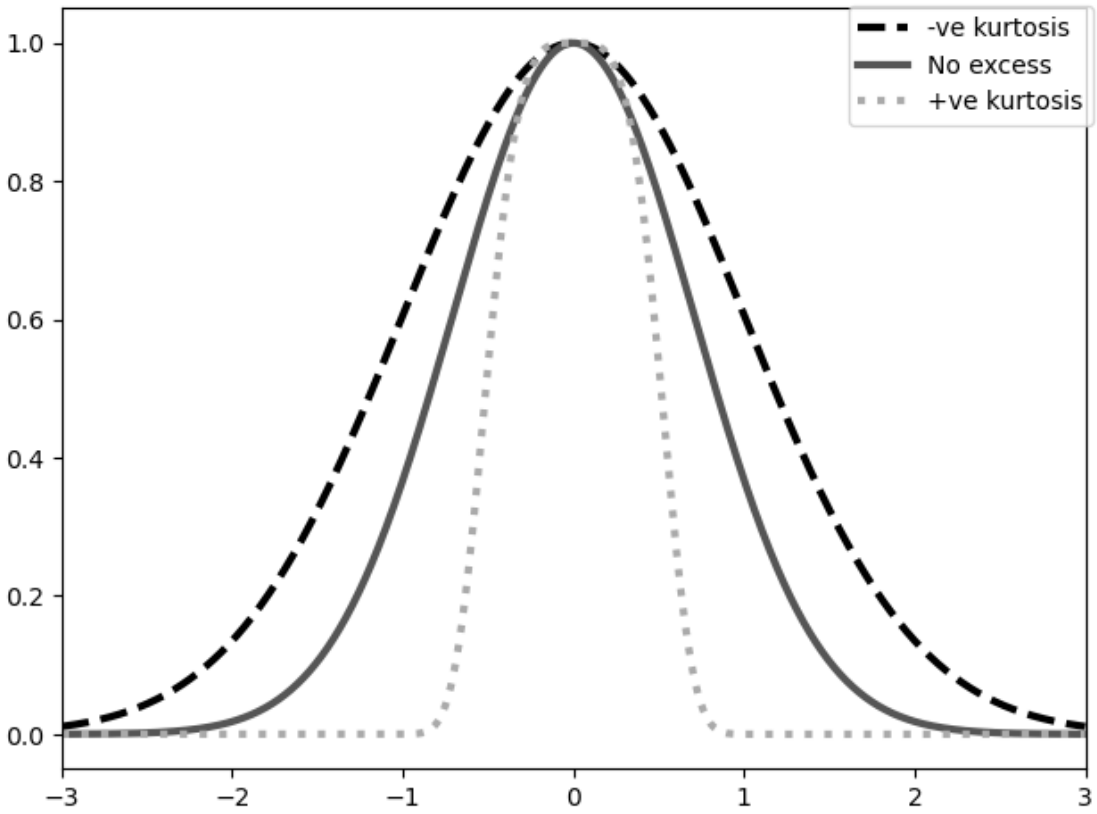
21 Figure 4.23: Illustration of skew

[19]:
```
# create data using F distribution
x = np.linspace(0,4,10001)
y = stats.f.pdf(x,10,100)
y = y/np.max(y)
yBar = np.sum(y)*np.mean(np.diff(x))

# plot the distributions
_,axs = plt.subplots(2,1,figsize=(4,5))
axs[0].plot(-x,y,'k',linewidth=3)
axs[0].plot([-yBar,-yBar],[0,1],'k--')
axs[0].set_title(r'$\bf{A}$)  Left (negative) skew')
axs[0].set(xlim=-x[[-1,0]])
axs[0].set_ylabel('Proportion')
axs[0].set_xlabel('Data value')
axs[0].text(-3.5,.7,r'$\sum (x-\overline{x})^3 < 0$')

axs[1].plot(x,y,'k',linewidth=3)
axs[1].plot([yBar,yBar],[0,1],'k--')
axs[1].set_title(r'$\bf{B}$)  Right (positive) skew')
axs[1].set(xlim=x[[0,-1]])
axs[1].text(2,.7,r'$\sum (x-\overline{x})^3 > 0$')


for a in axs: a.set(xticks=[],yticks=[])


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



22 Figure 4.24: Kurtosis

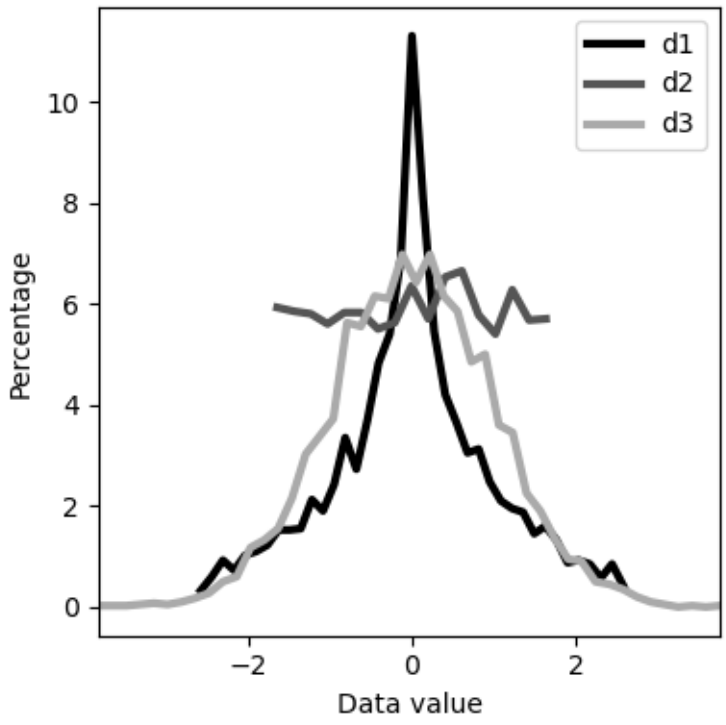
[20]:
```
# generate the distributions
x = np.linspace(-3,3,10001)
g = [None]*3
g[0] = np.exp( -.5*x**2 )
g[1] = np.exp(    -x**2 )
g[2] = np.exp( -10*x**4 )


# generate the plot
s = ['--','-',':']
n = ['-ve kurtosis','No excess','+ve kurtosis']
for i in range(3):
  plt.plot(x,g[i],color=(i/3,i/3,i/3),linewidth=3,linestyle=s[i],label=n[i])

plt.legend(loc='upper right',bbox_to_anchor=[1.02,1.02])
plt.xlim(x[[0,-1]])

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



23 Figure 4.25: Different kurtosis with the same variance

[21]:
```
# use the following 5 lines to manipulate kurtosis
x1 = np.exp(-np.abs(3*np.random.randn(4000)))
x2 = np.exp(-np.abs(3*np.random.randn(4000)))
d1 = x1-x2+1
d2 = np.random.rand(4000)
d3 = np.random.randn(4000)

# uncomment the following 3 lines to manipulate skew
# d1 = np.random.randn(4000)
# d2 = np.exp(np.random.randn(4000)/2)
# d3 = -np.exp(np.random.randn(4000)/2)

# gather into a list
data = [d1,d2,d3]

S = np.zeros((len(data),4))
i = 0
datalabel = []

plt.figure(figsize=(4,4))

for X in data:

  # optional normalization
  X = (X-np.mean(X)) / np.std(X)

  # histogram and plot
  y1,x1 = np.histogram(X,bins='fd')
  x1 = (x1[1:]+x1[:-1])/2
  y1 = 100*y1/np.sum(y1)
  plt.plot(x1,y1,linewidth=3,color=(i/3,i/3,i/3))
  datalabel.append('d'+str(i+1))

  # gather stats
  S[i,:] = np.mean(X),np.var(X,ddof=1),stats.skew(X),stats.kurtosis(X)
  i += 1


plt.legend(datalabel)
plt.xlim([np.min(x1),np.max(x1)])
plt.xlabel('Data value')
plt.ylabel('Percentage')
```

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


# now print out the descriptive stats
df = pd.DataFrame(S.
 ↪T,columns=datalabel,index=['Mean','Variance','Skew','Kurtosis'])
from IPython.display import display
with pd.option_context('display.float_format','{:5.3f}'.format):
  display(df)
```



	d1	d2	d3
Mean	0.000	0.000	-0.000
Variance	1.000	1.000	1.000
Skew	0.005	-0.026	-0.012
Kurtosis	0.160	-1.184	0.105

24 Figure 4.27: Histogram bins

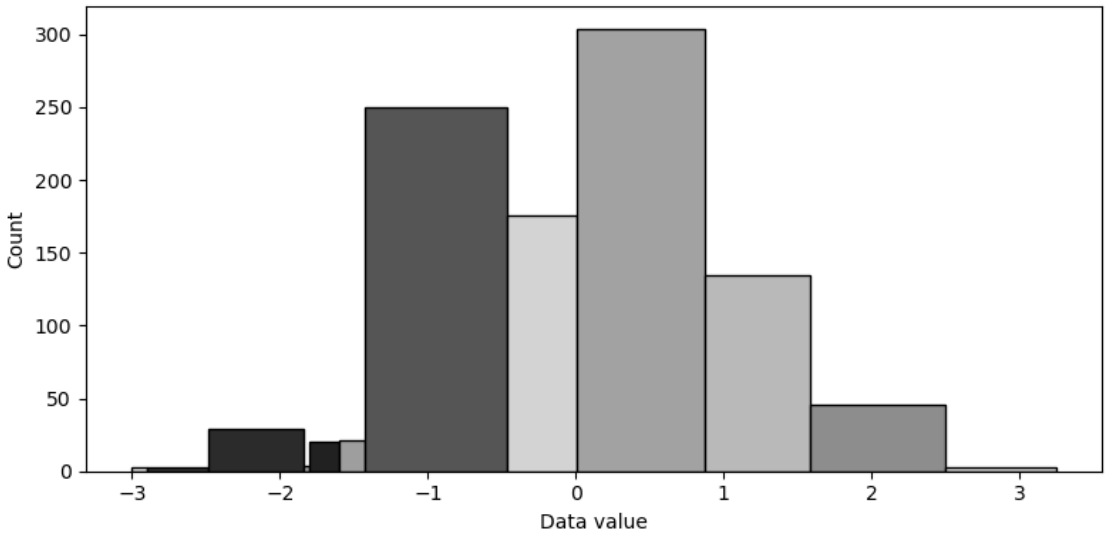
[22]:
```
# table data
colLabs = [ 'Method','Formula','Key advantage' ]
```

```
tableData = [ [ 'Arbitrary'          ,  r'$k=40$'                                  , ␣
 ↪'Simple' ],
              [ 'Sturges'            ,  r'$k=\lceil \log_2(N)\rceil+1$'            , ␣
 ↪'Depends on count' ],
              [ 'Friedman-Diaconis' ,  r'$w=2\frac{IQR}{\sqrt[3]{N}}$' , ␣
 ↪'Depends on count and spread' ],
]


# draw the table
fig, ax = plt.subplots(figsize=(10,2))
ax.set_axis_off()
ht = ax.table(
        cellText   = tableData,
        colLabels  = colLabs,
        colColours = [(.8,.8,.8)] * len(colLabs),
        cellLoc    = 'center',
        loc        = 'upper left',
        )


# some adjustments to the fonts etc
ht.scale(1,2.5)
ht.auto_set_font_size(False)
ht.set_fontsize(14)

from matplotlib.font_manager import FontProperties
for (row, col), cell in ht.get_celld().items():
  cell.set_text_props(fontproperties=FontProperties(family='serif'))
  if row==0: cell.
 ↪set_text_props(fontproperties=FontProperties(weight='bold',size=16))

# export
plt.tight_layout()
#plt.savefig('desc_table_moments2.png', bbox_inches='tight')
plt.show()
```

Method	Formula	Key advantage
Arbitrary	$k = 40$	Simple
Sturges	$k = \lceil \log_2(N) \rceil + 1$	Depends on count
Friedman-Diaconis	$w = 2\frac{IQR}{\sqrt[3]{N}}$	Depends on count and spread

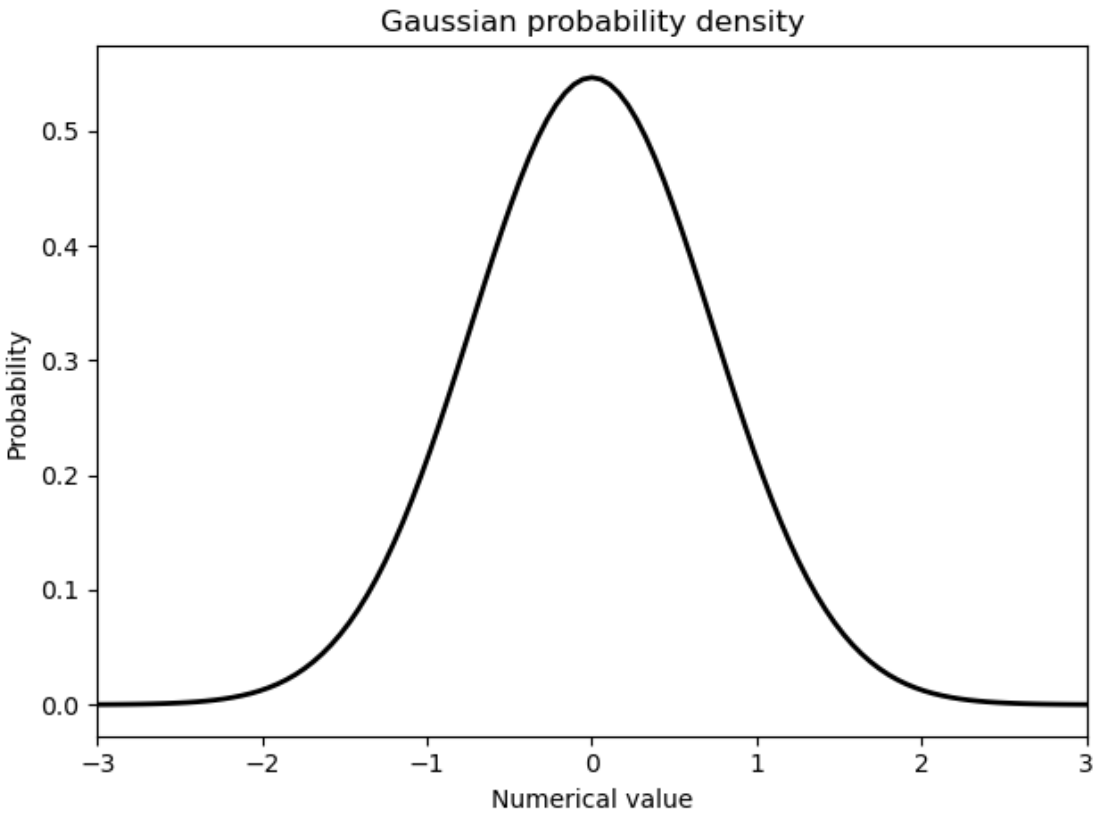
25 Figure 4.28: Cityscape of variable histogram bins

[23]:
```
# define random bin edges
bw = np.array([-3])
while bw[-1]<3:
  bw = np.append(bw,bw[-1]+np.random.rand(1))


# create the histogram
plt.figure(figsize=(8,4))
_,_,hs = plt.hist(np.random.randn(1000),bins=bw,edgecolor='k')

# assign random greyscale color to each bar
for h in hs:
  c = np.random.uniform(low=.1,high=.9,size=1)[0]
  h.set_facecolor((c,c,c))

plt.xlabel('Data value')
plt.ylabel('Count')
plt.tight_layout()
#plt.savefig('desc_variableBins.png')
plt.show()
```



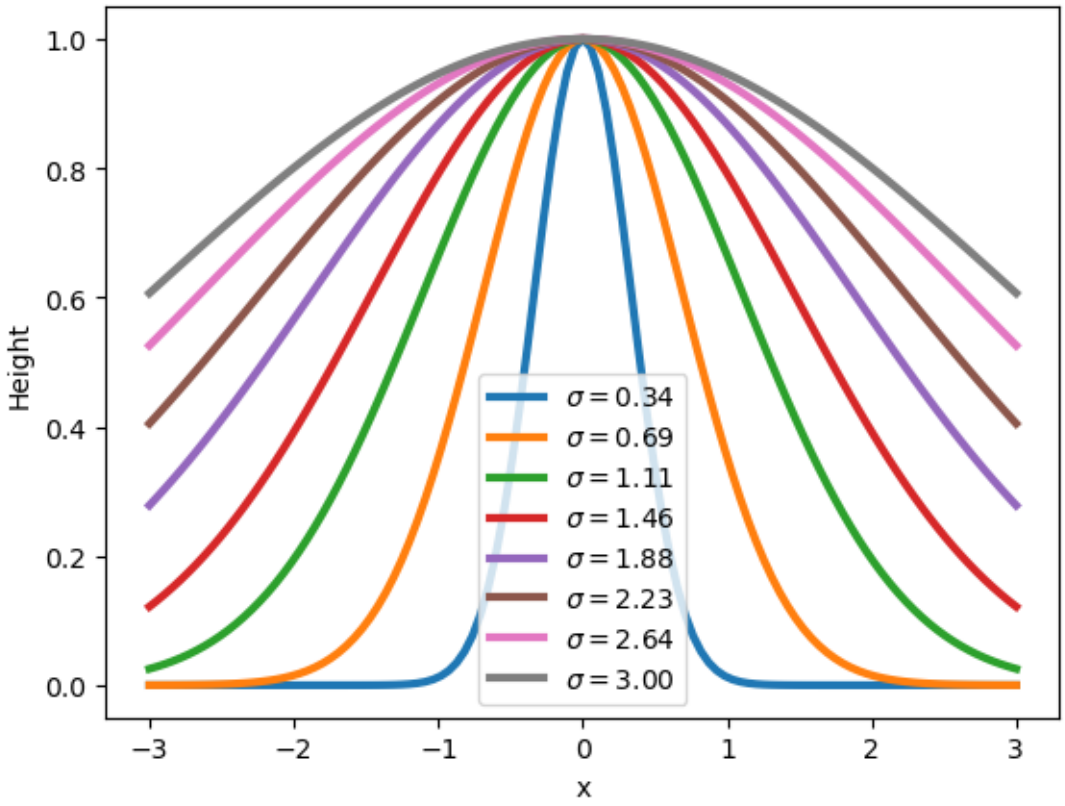
26 Exercise 1

[24]:
```
# function variable
x = np.linspace(-3,3,111)

sigma = .73

# one gaussian
a = 1 / (sigma*np.sqrt(2*np.pi))
eTerm = -x**2 / (2*sigma**2)
gaus = a * np.exp( eTerm )

plt.plot(x,gaus,'k',linewidth=2)
plt.xlim(x[[0,-1]])
plt.xlabel('Numerical value')
plt.ylabel('Probability')
plt.title('Gaussian probability density',loc='center')

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

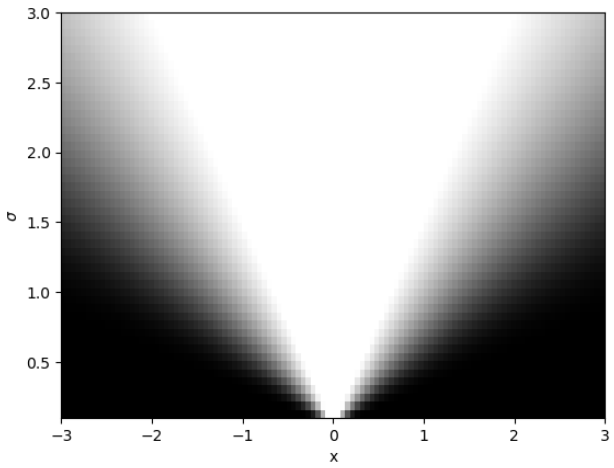


[25]:
```
# Create a family of Gaussians

N = 50
sigmas = np.linspace(.1,3,N)

G = np.zeros((N,len(x)))

for i in range(N):
  a = 1 / (sigmas[i]*np.sqrt(2*np.pi))
  eTerm = -x**2 / (2*sigmas[i]**2)
  G[i,:] = np.exp( eTerm )
```

[26]:
```
# visualize a few

whichGaussians = np.round(np.linspace(4,N-1,8)).astype(int)

plt.plot(x,G[whichGaussians,:].T,linewidth=3)
plt.xlabel('x')
plt.ylabel('Height')
plt.legend([f'$\sigma = {s:.2f}$' for s in sigmas[whichGaussians]])
plt.show()
```

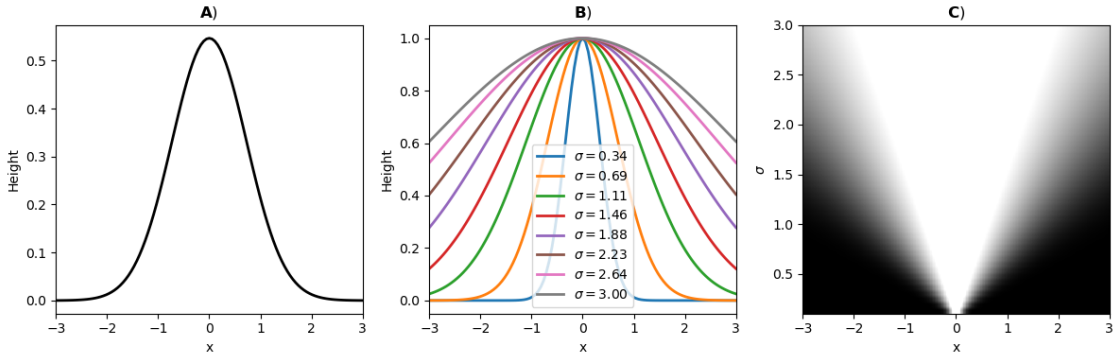


[27]:
```
# now show as a matrix
plt.imshow(G,extent=[x[0],x[-1],sigmas[0],sigmas[-1]],
           cmap='gray',aspect='auto',origin='lower',
           vmin=0,vmax=.8)
plt.xlabel('x')
plt.ylabel('$\sigma$')
plt.show()
```



[28]:
```
# this code is identical to that above, just in subplots for the book figure

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

# one gaussian
axs[0].plot(x,gaus,'k',linewidth=2)
axs[0].set_xlabel('x')
axs[0].set_xlim(x[[0,-1]])
axs[0].set_ylabel('Height')
axs[0].set_title(r'$\bf{A}$)')

# a few gaussians
axs[1].plot(x,G[whichGaussians,:].T,linewidth=2)
axs[1].set_xlabel('x')
axs[1].set_ylabel('Height')
axs[1].set_xlim(x[[0,-1]])
axs[1].legend([f'$\sigma = {s:.2f}$' for s in␣
 ↪sigmas[whichGaussians]],fontsize=10)
axs[1].set_title(r'$\bf{B}$)')

# the gaussian family portrait
axs[2].imshow(G,extent=[x[0],x[-1],sigmas[0],sigmas[-1]],
```

```
            cmap='gray',aspect='auto',origin='lower',
            vmin=0,vmax=.8)
axs[2].set_xlabel('x')
axs[2].set_ylabel('$\sigma$')
axs[2].set_title(r'$\bf{C}$)')

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



27 Exercise 2

[29]:
```
# here's the sum
print( np.sum(G,axis=1) )
```

```
[ 4.59548517  7.31526211 10.03503905 12.75481598 15.47459292 18.19436986
 20.91414679 23.63392364 26.35369728 29.07342827 31.79287171 34.51112086
 37.22587839 39.93274947 42.62492198 45.29339702 47.92767093 50.51663089
 53.04943113 55.51619775 57.90850142 60.2196032  62.44451434 64.57992248
 66.6240336  68.57637012 70.43755438 72.20909703 73.89320204 75.49259437
 77.01037227 78.44988397 79.81462686 81.10816684 82.33407493 83.49587855
 84.59702499 85.64085481 86.63058324 87.56928795 88.4599019  89.30520995
 90.10784861 90.87030788 91.59493482 92.28393824 92.93939423 93.56325217
 94.15734102 94.72337578]
```

[30]:
```
# we don't want the mean...
print( np.mean(G,axis=1) ) #* np.mean(np.diff(x))
```

```
[0.04140077 0.06590326 0.09040576 0.11490825 0.13941075 0.16391324
 0.18841574 0.21291823 0.2374207  0.26192278 0.28642227 0.310911
 0.33536827 0.3597545  0.38400831 0.40804862 0.43178082 0.45510478
 0.4779228  0.50014593 0.52169821 0.54251895 0.56256319 0.5818011
 0.60021652 0.61780514 0.63457256 0.65053241 0.66570452 0.68011346
 0.69378714 0.70675571 0.71905069 0.73070421 0.74174842 0.75221512
```

```
 0.76213536 0.77153923 0.7804557  0.7889125  0.79693605 0.80455144
 0.81178242 0.81865142 0.82517959 0.83138683 0.83729184 0.84291218
 0.84826433 0.85336375]
```

[31]:
```
# we want the discrete integral
print( np.sum(G,axis=1) * np.mean(np.diff(x)) )
```

```
[0.25066283 0.3990143  0.54736577 0.69571724 0.8440687  0.99242017
 1.14077164 1.28912311 1.4374744  1.58582336 1.73415664 1.88242477
 2.03050246 2.17814997 2.32499574 2.47054893 2.6142366  2.75545259
 2.89360533 3.02815624 3.15864553 3.28470563 3.40606442 3.52254123
 3.6340382  3.74052928 3.84204842 3.93867802 4.03053829 4.11777787
 4.20056576 4.27908458 4.3535251  4.42408183 4.49094954 4.55432065
 4.61438318 4.67131935 4.72530454 4.77650662 4.82508556 4.87119327
 4.91497356 4.95656225 4.99608735 5.03366936 5.0694215  5.10345012
 5.13585496 5.16672959]
```

28 Exercise 3

[32]:
```
### the function

def computeStats(data):

  # for convenience
  N = len(data)

  ## mean
  myMean = sum(data)/N

  ## median
  # first sort the data
  dataSort = sorted(data)

  # then compute the median based on whether it's odd or even N
  if N%2==1: # odd
    myMedian = dataSort[N//2] # N//2 == (N+1)/2
  else: # even
    myMedian = sum(dataSort[N//2-1:N//2+1])/2


  ## variance
  myVar = sum([(i-myMean)**2 for i in data]) / (N-1)

  return myMean,myMedian,myVar
```

[33]:
```
# the specified data
data = [ 1,7,2,7,3,7,4,7,5,7,6,7 ]
```

```
# uncomment the following line for random integers (note the exclusive upper␣
↪bound!)
data = np.random.randint(low=4,high=21,size=24)

# my results
myMean,myMedian,myVar = computeStats(data)

# ground-truth to compare your results:
npMean   = np.mean(data)
npMedian = np.median(data)
npVar    = np.var(data,ddof=1)
```

[34]:
```
# print the results

print('          |  Mine  |  numpy')
print('----------------------------')
print(f'Mean     | {myMean:5.2f}  | {npMean:5.2f}')
print(f'Median   | {myMedian:5.2f}  | {npMedian:5.2f}')
print(f'Variance | {myVar:5.2f}  | {npVar:5.2f}')
```

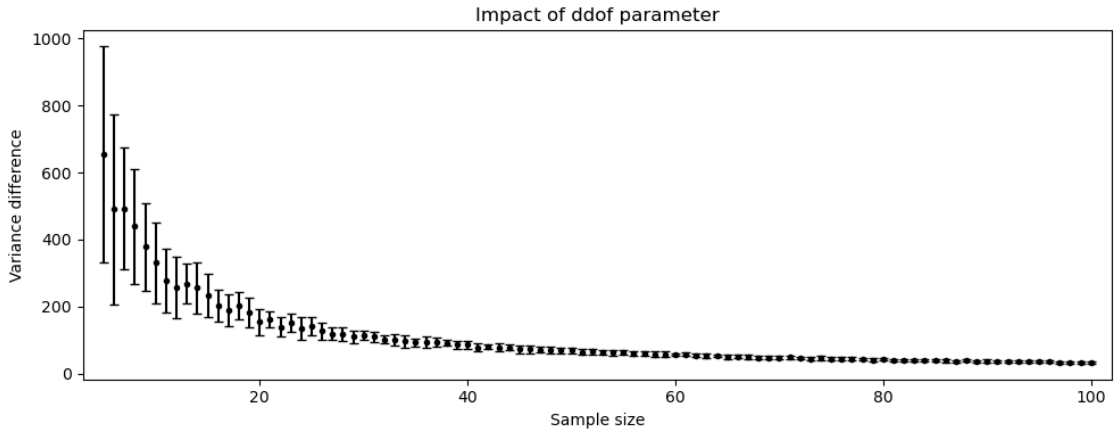
```
          |  Mine  |  numpy
----------------------------
Mean     | 12.75  | 12.75
Median   | 13.00  | 13.00
Variance | 25.15  | 25.15
```

29 Exercise 4

[35]:
```
# experiment parameters
sampleSizes = np.arange(5,101)
numExperiments = 25

# initialize results matrix
ddofImpact = np.zeros((len(sampleSizes),numExperiments))

# double for-loop over sample sizes and experiment repetitions
for ni in range(len(sampleSizes)):
  for expi in range(numExperiments):

    # generate random data
    data = np.random.randint(low=-100,high=101,size=sampleSizes[ni])

    # compute variance difference
    varDiff = np.var(data,ddof=1)-np.var(data,ddof=0)

    # uncomment the lines below for exercise 5
```

```
    d = np.var(data,ddof=1)-np.var(data,ddof=0)
    a = np.var(data,ddof=1)+np.var(data,ddof=0)
    #varDiff = d/a

    # store magnitude
    ddofImpact[ni,expi] = varDiff
```

[51]:
```
# compute average and std across experiment runs
meanDiffs = np.mean(ddofImpact,axis=1)
stdDiffs  = np.std(ddofImpact,axis=1,ddof=1)


# now for visualization
plt.figure(figsize=(10,4))
plt.errorbar(sampleSizes,meanDiffs,stdDiffs,
             color='k',marker='.',fmt=' ',capsize=3)

# make the plot look a bit nicer
plt.title('Impact of ddof parameter',loc='center')
plt.xlabel('Sample size')
plt.ylabel('Variance difference')
plt.xlim([sampleSizes[0]-2,sampleSizes[-1]+2])

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



30 Exercise 5

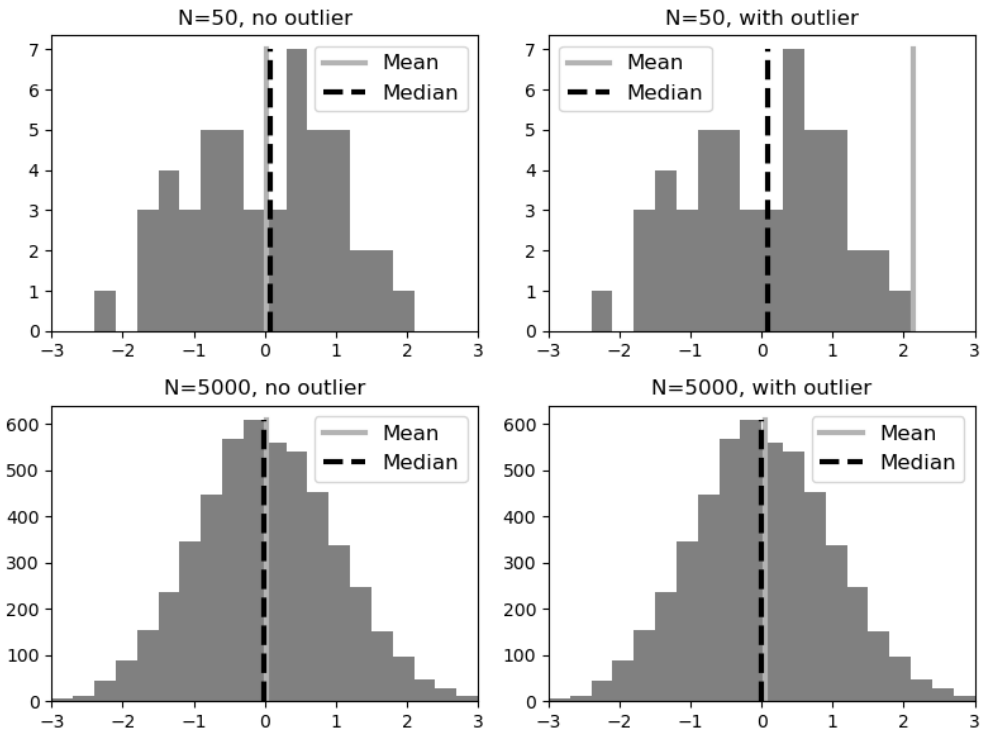
[ ]:
```
# The code solution to this exercise is in the previous exercise.
# Just uncomment the second calculation of variable varDiff.
```

[ ]:
```
# The reason why the normalized differences are simply 1/(2n-1) comes from
# writing out the difference using the formula for variance. You'll find
# that the summation terms cancel and only the 1/n or 1/(n-1) terms remain.
# Then you apply a bit of algebra to reduce to 1/(2n-1).
```

31 Exercise 6

[36]:
```
# Compare mean and median with and without outliers, for large and small N

Ns = [ 50,5000 ]

# initialize results matrices
means = np.zeros((2,2))
medians = np.zeros((2,2))


_,axs = plt.subplots(2,2,figsize=(8,6))

for ni in range(len(Ns)):

  # create the data as normal random numbers
  data = np.random.randn(Ns[ni])

  for outi in range(2):

    # I created an outlier by squaring the largest random sample
    maxval,maxidx = np.max(data),np.argmax(data)
    data[maxidx] = maxval**([1,4][outi])

    # store results in matrices
    means[ni,outi] = np.mean(data)
    medians[ni,outi] = np.median(data)

    # plot
    h = axs[ni,outi].hist(data,bins=np.linspace(-3,3,21),color='gray')
    maxY = np.max(h[0])
    axs[ni,outi].plot([np.mean(data),np.mean(data)],[0,maxY],color=[.7,.7,.
↪7],linewidth=3,label='Mean')
    axs[ni,outi].plot([np.median(data),np.
↪median(data)],[0,maxY],'k--',linewidth=3,label='Median')
    axs[ni,outi].legend(fontsize=12)
    axs[ni,outi].set_xlim([-3,3])
```

```
    axs[ni,outi].set_title(f'N={Ns[ni]}, {("no","with")[outi]}␣
↪outlier',loc='center')



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



[37]:
```
# print results
for ni in range(2):
  print(f'With N = {Ns[ni]}, the mean increased by {means[ni,1]-means[ni,0]:.
↪2f}')
  print(f'With N = {Ns[ni]}, the median increased by␣
↪{medians[ni,1]-medians[ni,0]:.2f}')
  print(' ')
```

```
With N = 50, the mean increased by 2.10
With N = 50, the median increased by 0.00

With N = 5000, the mean increased by 0.03
With N = 5000, the median increased by 0.00
```

32 Exercise 7

[38]:
```
mean, variance, skew, kurtosis = stats.norm.stats(loc=1,scale=2,moments='mvsk')
# for other distributions, replace 'norm' with uniform, lognorm, or expnorm

print(f' Average: {mean:.2f}')
print(f'Variance: {variance:.2f}')
print(f'Skewness: {skew:.2f}')
print(f'Kurtosis: {kurtosis:.2f}')
```

```
 Average: 1.00
Variance: 4.00
Skewness: 0.00
Kurtosis: 0.00
```

[ ]:
```
# url to site with list of scipy distribution options:
# https://docs.scipy.org/doc/scipy/reference/stats.html#continuous-distributions
```

33 Exercise 8

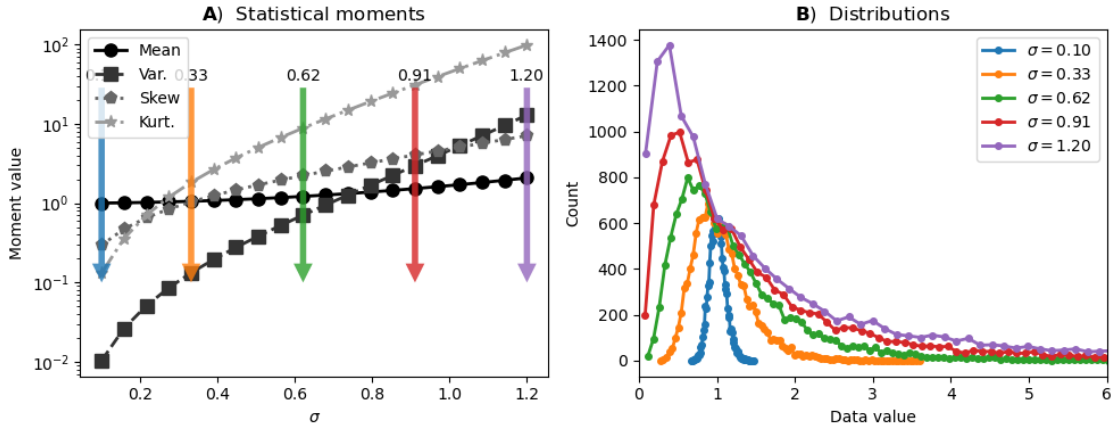
[39]:
```
# a function to compute and return moments
def moments(data):
  # 1st moment is the mean
  mean = np.mean(data)

  # 2nd moment is variance
  var = np.var(data,ddof=1)

  # 3rd moment is skew
  skew = stats.skew(data)

  # 4th moment is kurtosis
  kurt = stats.kurtosis(data)

  # put them all together
  return mean,var,skew,kurt
```

[40]:
```
# generate the data
sigmas = np.linspace(.1,1.2,20)
X = np.random.randn(13524)

# initialize the data results matrices
M = np.zeros((len(sigmas),4))
data = [0]*len(sigmas)

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

```
  data[i] = np.exp(X*s) # create the data
  M[i,:] = moments( data[i] ) # compute its moments
```

[41]:
```
_,axs = plt.subplots(1,2,figsize=(10,4))

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

# plot each line separately
for i in range(4):
  axs[0].plot(sigmas,M[:,i],m[i]+l[i],color=np.ones(3)*c[i],
              linewidth=2,markersize=8)

axs[0].set_xlabel('$\sigma$')
axs[0].set_ylabel('Moment value')
axs[0].legend(['Mean','Var.','Skew','Kurt.'])
axs[0].set_title(r'$\bf{A}$)  Statistical moments')

# now to 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[1].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[0].annotate(f'{sigmas[i]:.2f}',xy=(sigmas[i],.
 ↪1),horizontalalignment='center',fontsize=10,
                  xytext=(sigmas[i],axs[0].get_ylim()[1]/
 ↪3),arrowprops=dict(width=4,linewidth=0,color=h[0].get_color(),alpha=.8))

# some niceties
axs[1].legend()
axs[1].set_xlim([0,6])
axs[1].set_xlabel('Data value')
axs[1].set_ylabel('Count')
axs[1].set_title(r'$\bf{B}$)  Distributions')

# optional y-axis logarithmic scaling to see subtle changes in the other moments
axs[0].set_yscale('log')

plt.tight_layout()
```

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

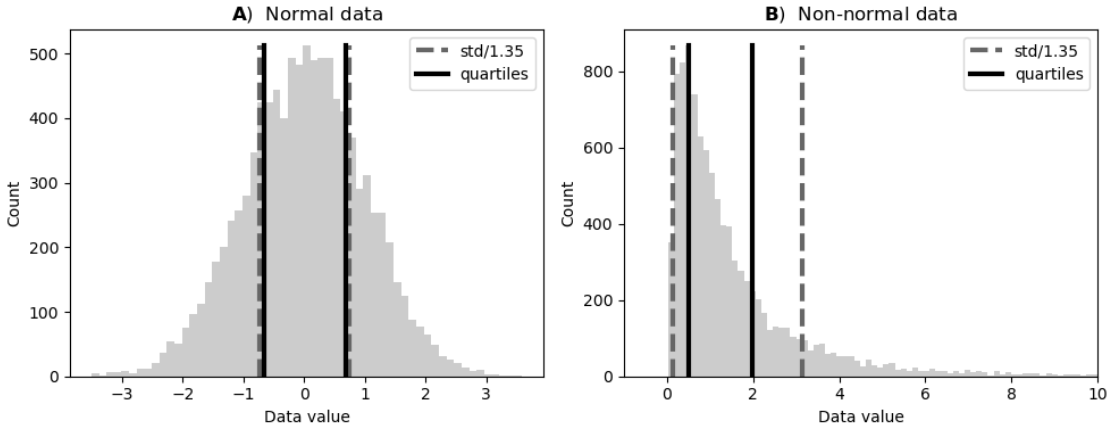


34 Exercise 9

[42]:
```
# generate random datasets
X = np.random.randn(10000)
eX = np.exp(X)

# iqr and std
iqr = stats.iqr(X)
std = np.std(X,ddof=1)
eiqr = stats.iqr(eX)
estd = np.std(eX,ddof=1)

# print their comparison
print('Normal distribution:')
print(f'    IQR = {iqr:.3f}')
print(f'1.35std = {1.35*std:.3f}')

print('\nLog-normal distribution:')
print(f'    IQR = {eiqr:.3f}')
print(f'1.35std = {1.35*estd:.3f}')
```

```
Normal distribution:
    IQR = 1.343
1.35std = 1.341

Log-normal distribution:
    IQR = 1.455
1.35std = 2.722
```

[43]:
```
_,axs = plt.subplots(1,2,figsize=(10,4))

for i in range(2):

  # select data
  if i==0:
    data = X
  else:
    data = eX

  # convenience variable
  mean = np.mean(data)
  std  = np.std(data,ddof=1)/1.35


  # histogram of the data and maximum height value
  h = axs[i].hist(data,bins='fd',color=[.8,.8,.8])
  maxY = np.max(h[0])

  # standard deviation lines
  axs[i].plot([mean+std,mean+std],[0,maxY],color=(.4,.4,.
↪4),linestyle='--',linewidth=3,label='std/1.35')
  axs[i].plot([mean-std,mean-std],[0,maxY],color=(.4,.4,.
↪4),linestyle='--',linewidth=3)

  # quartiles lines
  quartiles = np.percentile(data,[25,75])
  axs[i].
↪plot([quartiles[0],quartiles[0]],[0,maxY],'k',linewidth=3,label='quartiles')
  axs[i].plot([quartiles[1],quartiles[1]],[0,maxY],'k',linewidth=3)

  axs[i].set(xlabel='Data value',ylabel='Count')
  axs[i].legend()
  if i==1: axs[i].set(xlim=[-1,10]) # manually set for e^X


axs[0].set_title(r'$\bf{A}$)  Normal data')
axs[1].set_title(r'$\bf{B}$)  Non-normal data')

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



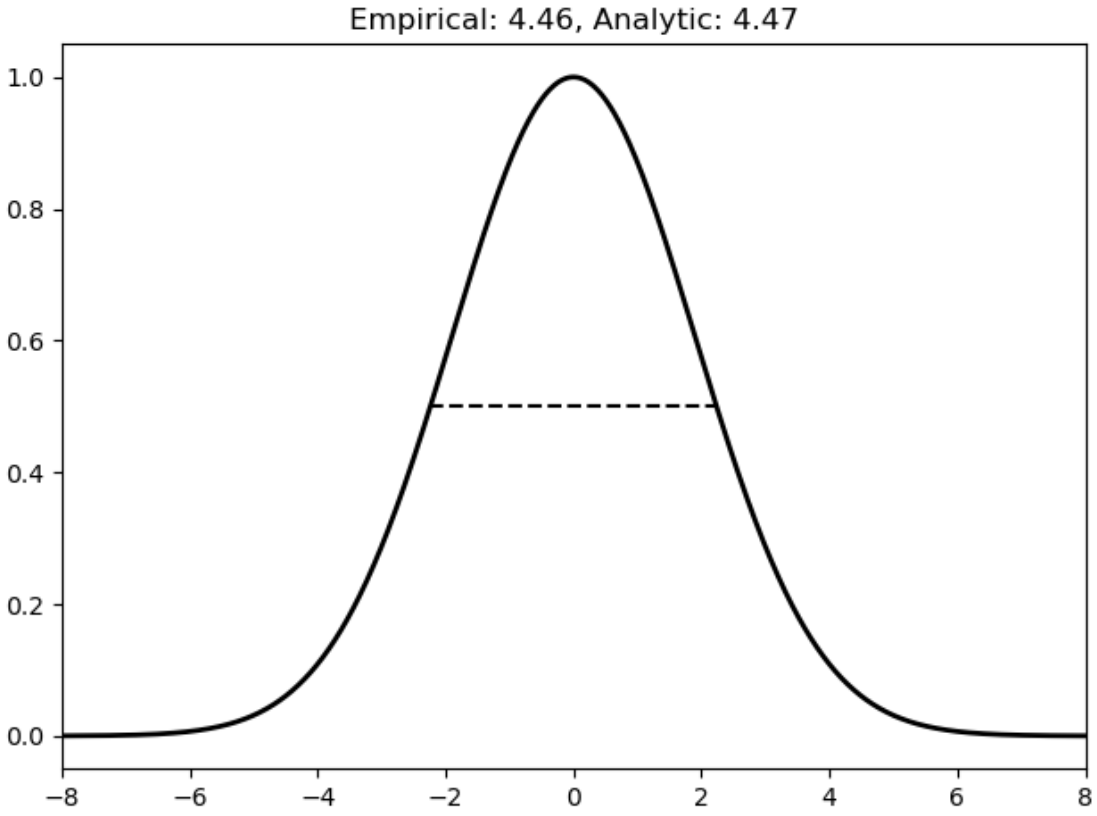
35 Exercise 10

[44]:
```
# FWHM function
def empFWHM(x,y):

  # normalize to [0,1]
  y = y-np.min(y)
  y = y/np.max(y)

  # find peak
  idx = np.argmax(y)

  # find value before peak
  prePeak = x[ np.argmin(np.abs(y[:idx]-.5)) ]

  # find value after peak
  pstPeak = x[ idx-1+np.argmin(np.abs(y[idx:]-.5)) ]

  # return fwhm as that distance
  return pstPeak-prePeak,(prePeak,pstPeak)
```

[45]:
```
# try on pdf to comapre with analytic
x = np.linspace(-8,8,1001)
s = 1.9

# create an analytic Gaussian
pureGaus = np.exp( (-x**2)/(2*s**2) )

# empirical and analytic FWHM
fwhm,halfpnts = empFWHM(x,pureGaus)
afwhm = 2*s*np.sqrt(2*np.log(2))
```

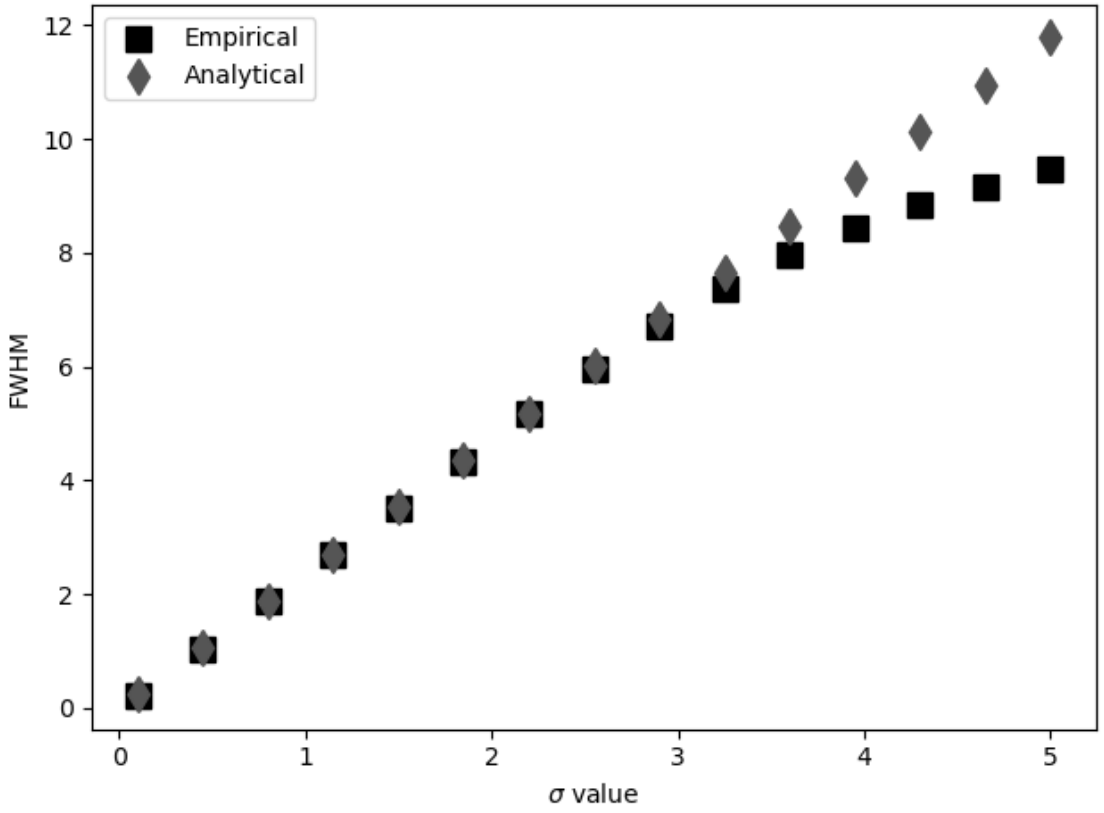
```
print(f'Empirical  FWHM = {fwhm:.2f}')
print(f'Analytical FWHM = {afwhm:.2f}')

# show the plot
plt.plot(x,pureGaus,'k',linewidth=2)
plt.plot([halfpnts[0],halfpnts[1]],[.5,.5],'k--')
plt.title(f'Empirical: {fwhm:.2f}, Analytic: {afwhm:.2f}',loc='center')
plt.xlim([x[0],x[-1]])

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

```
Empirical  FWHM = 4.46
Analytical FWHM = 4.47
```

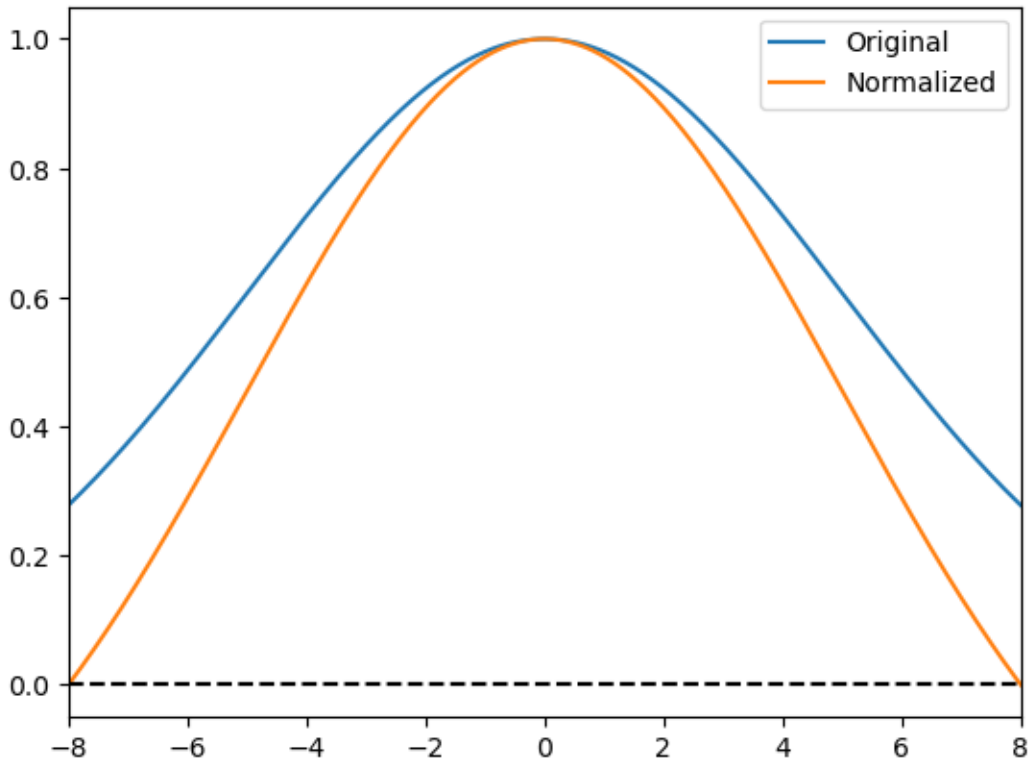


[47]:
```
# try on pdf to comapre with analytic
ss = np.linspace(.1,5,15)

fwhms = np.zeros((len(ss),2))
```

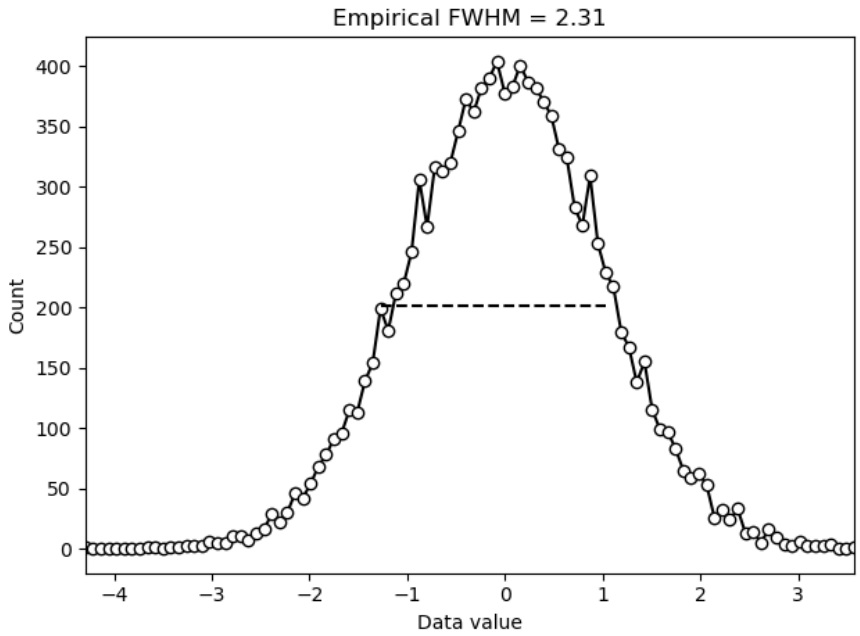
```
for i,s in enumerate(ss):

  # create the Gaussian (don't need the 'a' term b/c we're already normalizing)
  gx = np.exp( (-x**2)/(2*s**2) )

  # compute FWHM and other related quantities
  fwhms[i,0] = empFWHM(x,gx)[0]
  fwhms[i,1] = 2*s*np.sqrt(2*np.log(2))


m = ('s','d')
for i in range(2):
  plt.plot(ss,fwhms[:,i],m[i],color=(i/3,i/3,i/3),markerfacecolor=(i/3,i/3,i/
 ↪3),markersize=10)

plt.legend(['Empirical','Analytical'])
plt.xlabel(r'$\sigma$ value')
plt.ylabel('FWHM')


plt.tight_layout()
#plt.savefig('desc_ex_fwhm1.png')
plt.show()
```



[48]:
```
# The problem is that the later Gaussians don't have a wide enough x-axis range,
# so the normalization distorts the Gaussian. This is illustrated in the code
↪below.
# Increasing the range of the x-axis grid will fix the problem.

# normalized version
y = gx-np.min(gx)
y = y/np.max(y)

# plot original and normalized
plt.plot(x,gx,label='Original')
plt.plot(x,y,label='Normalized')

# some additional thingies
plt.plot(x[[0,-1]],[0,0],'k--')
plt.ylim([-.05,1.05])
plt.xlim(x[[0,-1]])
plt.legend()

plt.show()
```

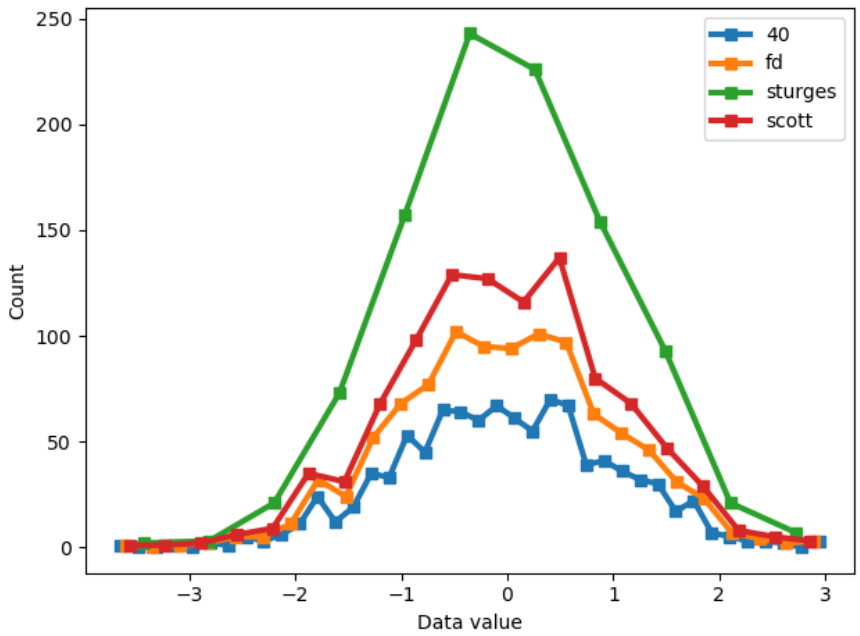


[49]:
```
# Now for an empirical histogram

# try on random data
data = np.random.randn(12345)

# histogram
y,x = np.histogram(data,bins=100)
x = (x[1:]+x[:-1])/2

# estimate the FWHM
h,peeks = empFWHM(x,y)
midheight = (np.max(y)-np.min(y))/2

# and plot
plt.plot(x,y,'ko-',markerfacecolor='w')
plt.plot([peeks[0],peeks[1]],[midheight,midheight],'k--')
plt.title(f'Empirical FWHM = {h:.2f}',loc='center')
plt.xlim([x[0],x[-1]])
plt.xlabel('Data value')
plt.ylabel('Count')

plt.tight_layout()
#plt.savefig('desc_ex_fwhm2.png')
plt.show()
```



36 Exercise 11

[50]:
```
# generate data
N = 1000
data = np.random.randn(N)
# data = np.random.rand(N) # uncomment for uniform noise

binOpt = [ 40,'fd','sturges','scott' ]

for bin in binOpt:

  # extract histogram
  y,x = np.histogram(data,bins=bin)
  x = (x[1:]+x[:-1])/2
  coeff= 1/np.max(y)

  # plot it
  plt.plot(x,y,'s-',linewidth=3,label=bin)


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

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



[ ]:
```
### Some observations about this exercise:
# - Some binnings give "taller" distributions, because they have fewer bins;
#   fewer bins means more data points per bin. Try normalizing the histograms
#   by plotting y/np.max(y)
#
# - When you use uniformly distributed data, it looks like some histograms␣
 ↪disappear,
#   but in fact the different rules give identical bin counts so the histograms␣
 ↪overlap.
#
```