stats_ch07_dataQC

August 2, 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 7

---

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.

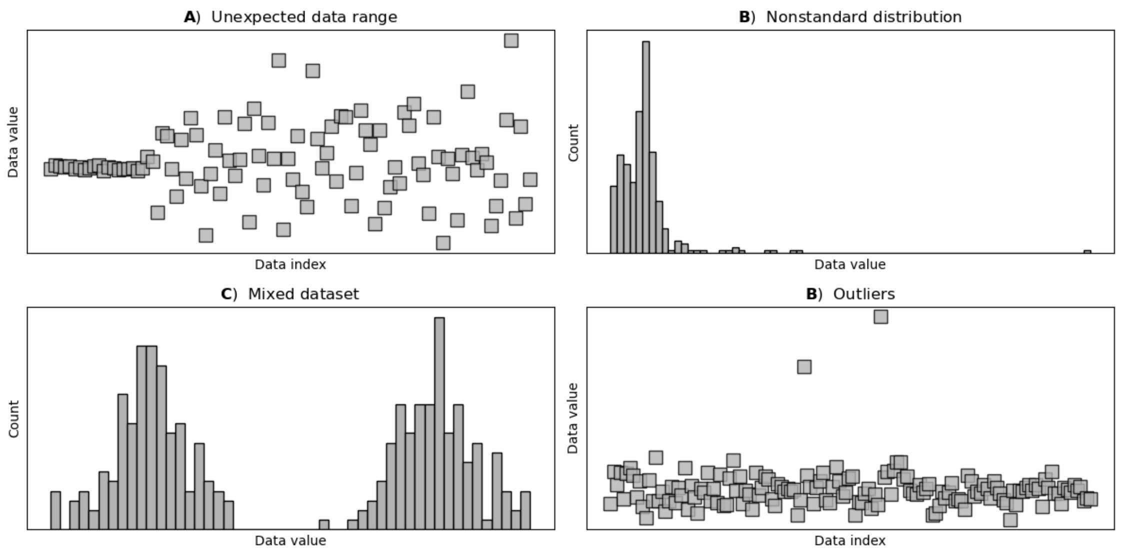
This code was written in google-colab. The notebook may require some modifications if you use a different IDE.

[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
```

3 Figure 7.2: What to look for in visual inspection of data

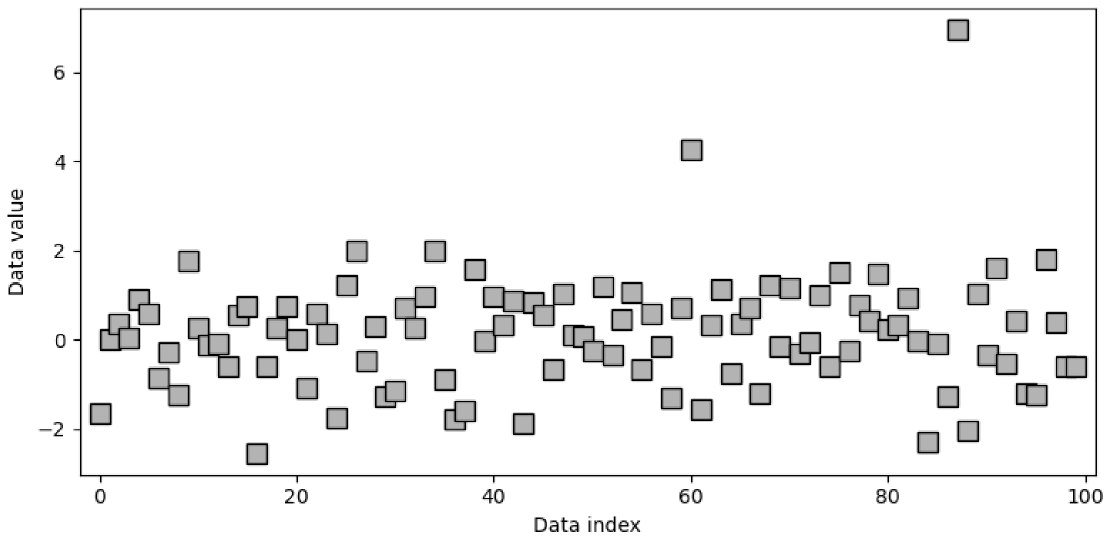
[2]:
```
_,axs = plt.subplots(2,2,figsize=(12,6))

# panel A: unexpected range
x = np.concatenate((np.random.randn(20),np.random.randn(80)*30),axis=0)
axs[0,0].plot(x,'ks',markersize=10,markerfacecolor=(.7,.7,.7),alpha=.8)
```

```
axs[0,0].set(xlabel='Data index',xticks=[],yticks=[],ylabel='Data value')
axs[0,0].set_title(r'$\bf{A}$)  Unexpected data range')

# panel B: distribution shape
x = np.concatenate((5+np.random.randn(150),np.exp(1+np.random.
 ↪randn(150))),axis=0)
axs[0,1].hist(x,bins='fd',edgecolor='k',facecolor=(.7,.7,.7))
axs[0,1].set(xlabel='Data value',xticks=[],yticks=[],ylabel='Count')
axs[0,1].set_title(r'$\bf{B}$)  Nonstandard distribution')

# panel C: mixed datasets
x = np.concatenate((4+np.random.randn(150),np.random.randn(150)-4),axis=0)
axs[1,0].hist(x,bins=50,edgecolor='k',facecolor=(.7,.7,.7))
axs[1,0].set(xlabel='Data value',xticks=[],yticks=[],ylabel='Count')
axs[1,0].set_title(r'$\bf{C}$)  Mixed dataset')

# panel D: outliers
x = np.random.randn(150)
x[60] = 10
x[84] = 14
axs[1,1].plot(x,'ks',markersize=10,markerfacecolor=(.7,.7,.7),alpha=.8)
axs[1,1].set(xlabel='Data index',xticks=[],yticks=[],ylabel='Data value')
axs[1,1].set_title(r'$\bf{B}$)  Outliers')

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



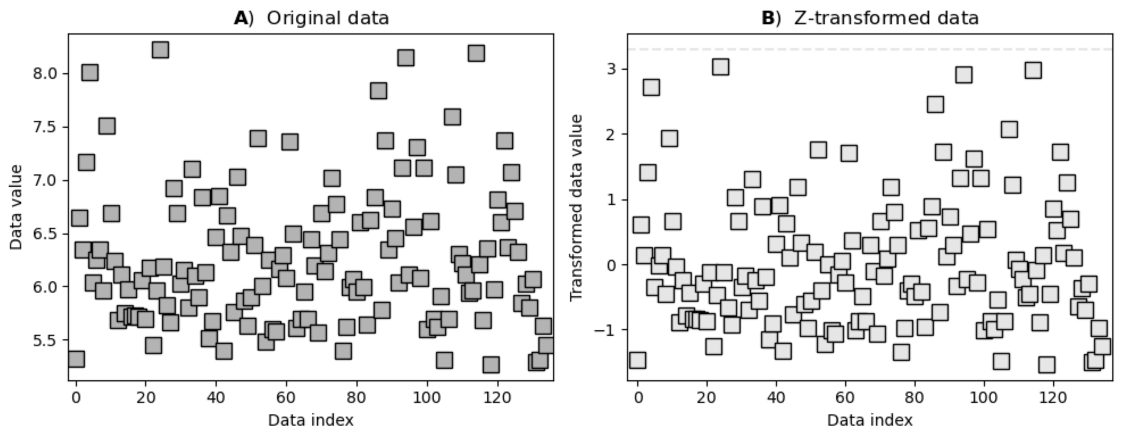
4 Figure 7.3: Example of dataset with outliers

[3]:
```
# Create normally distributed data
N = 100
data = np.random.randn(N)

# and add two random outliers in random positions
data[np.random.choice(np.arange(N),2)] = np.random.uniform(2,3,2)**2

# and plot
plt.figure(figsize=(8,4))
plt.plot(data,'ks',markersize=10,markerfacecolor=(.7,.7,.7))
plt.xlim([-2,N+1])
plt.xlabel('Data index')
plt.ylabel('Data value')

plt.tight_layout()
#plt.savefig('dataQC_example2outliers.png')
plt.show()
```



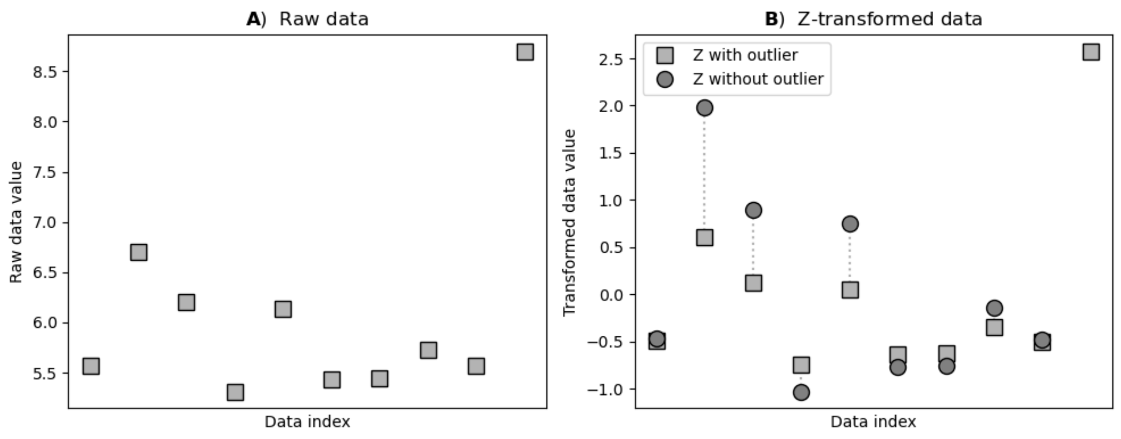
5 Figure 7.5: Z-score method for identifying outliers

[4]:
```
# outlier threshold
zThreshold = 3.29

# create some raw data
N = 135
data = np.exp(np.random.randn(N)/2) + 5

# zscore the data
dataZ = (data-np.mean(data)) / np.std(data,ddof=1)

# identify data indices containing outliers
outliers = np.where(np.abs(dataZ)>zThreshold)[0]

# and plot
_,axs = plt.subplots(1,2,figsize=(10,4))
axs[0].plot(data,'ks',markersize=10,markerfacecolor=(.7,.7,.7))
axs[0].set(xlim=[-2,N+1],xlabel='Data index',ylabel='Data value')
axs[0].set_title(r'$\bf{A}$)  Original data')


axs[1].plot(dataZ,'ks',markersize=10,markerfacecolor=(.9,.9,.9))
axs[1].axhline(zThreshold,linestyle='--',color=(.9,.9,.9))
axs[1].plot(outliers,dataZ[outliers],'kx',markersize=10,markeredgewidth=2)
axs[1].set(xlim=[-3,N+2],xlabel='Data index',ylabel='Transformed data value')
axs[1].set_title(r'$\bf{B}$)  Z-transformed data')

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



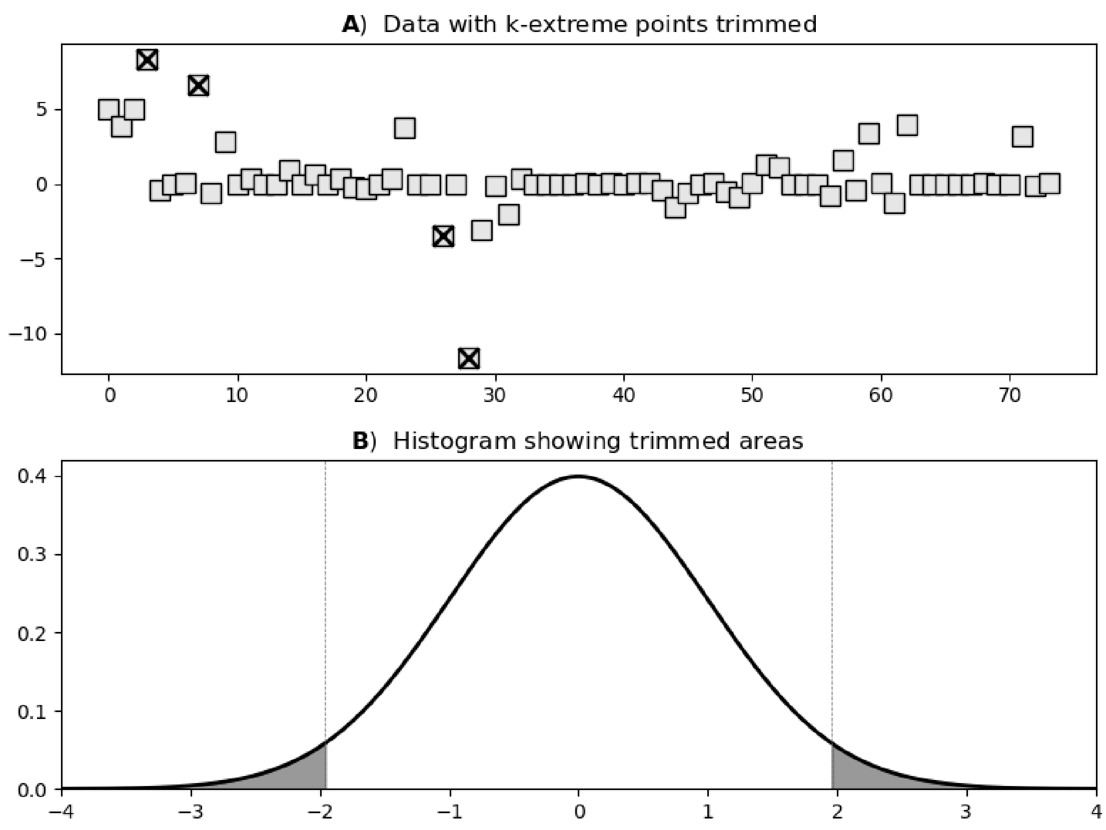
6 Figure 7.6: Impact of removing outliers on z-values

[5]:
```
# create some raw data
N = 10 # sample size
data = np.exp(np.random.randn(N)/2) + 5
data[-1] = np.max(data)+2 # impose an outlier (at the end for convenience)
xvals = np.arange(N)

dataZ1 = (data-np.mean(data)) / np.std(data,ddof=1)
dataZ2 = (data[:-1]-np.mean(data[:-1])) / np.std(data[:-1],ddof=1)

_,axs = plt.subplots(1,2,figsize=(10,4))
axs[0].plot(xvals,data,'ks',markersize=10,markerfacecolor=(.7,.7,.7))
axs[0].set(xticks=[],xlabel='Data index',ylabel='Raw data value')
axs[0].set_title(r'$\bf{A}$)  Raw data')

axs[1].plot(xvals,dataZ1,'ks',markersize=10,markerfacecolor=(.7,.7,.7),label='Z␣
 ↪with outlier')
axs[1].plot(xvals[:-1],dataZ2,'ko',markersize=10,markerfacecolor=(.5,.5,.
 ↪5),label='Z without outlier')
axs[1].set(xticks=[],xlabel='Data index',ylabel='Transformed data value')
axs[1].legend()
axs[1].set_title(r'$\bf{B}$)  Z-transformed data')

# draw lines connection pre/post-removal values
for d,z,x in zip(dataZ1[:-1],dataZ2,xvals[:-1]):
  axs[1].plot([x,x],[d,z],':',color=(.7,.7,.7),zorder=-10)

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



7 Figure 7.8: Data trimming

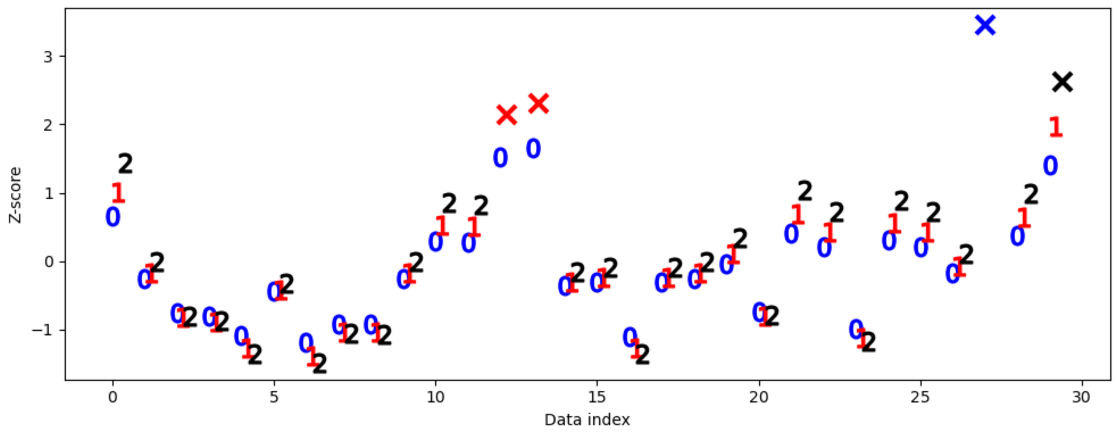
[6]:
```
N = 74
data = np.random.randn(N)**3

# find largest and smallest values
k = 2
sortidx = np.argsort(data)
minvals = sortidx[:k]
maxvals = sortidx[-k:]

_,axs = plt.subplots(2,1,figsize=(8,6))
axs[0].plot(data,'ks',markersize=10,markerfacecolor=(.9,.9,.9))
axs[0].plot(minvals,data[minvals],'kx',markersize=10,markeredgewidth=2)
axs[0].plot(maxvals,data[maxvals],'kx',markersize=10,markeredgewidth=2)
axs[0].set_title(r'$\bf{A}$)  Data with k-extreme points trimmed')

# create a Gaussian probability curve for the panel B
x = np.linspace(-4,4,401)
gpdf = stats.norm.pdf(x)

# the find the indices of the 2.5% and 97.5%
lbndi = np.argmin(np.abs(x-stats.norm.ppf(.025))) # lbndi = Lower BouND Index
ubndi = np.argmin(np.abs(x-stats.norm.ppf(1-.025)))

# plot the probability function and the vertical lines
axs[1].plot(x,gpdf,'k',linewidth=2)
axs[1].axvline(x[lbndi],color=(.5,.5,.5),linewidth=.5,linestyle='--')
axs[1].axvline(x[ubndi],color=(.5,.5,.5),linewidth=.5,linestyle='--')
axs[1].set(xlim=x[[0,-1]],ylim=[0,.42])
axs[1].set_title(r'$\bf{B}$)  Histogram showing trimmed areas')

# now create patches for the rejected area
axs[1].fill_between(x[:lbndi+1],gpdf[:lbndi+1],color='k',alpha=.4)
axs[1].fill_between(x[ubndi:],gpdf[ubndi:],color='k',alpha=.4)

# and save
plt.tight_layout()
#plt.savefig('dataQC_trimming.png')
plt.show()
```



8 Exercise 1

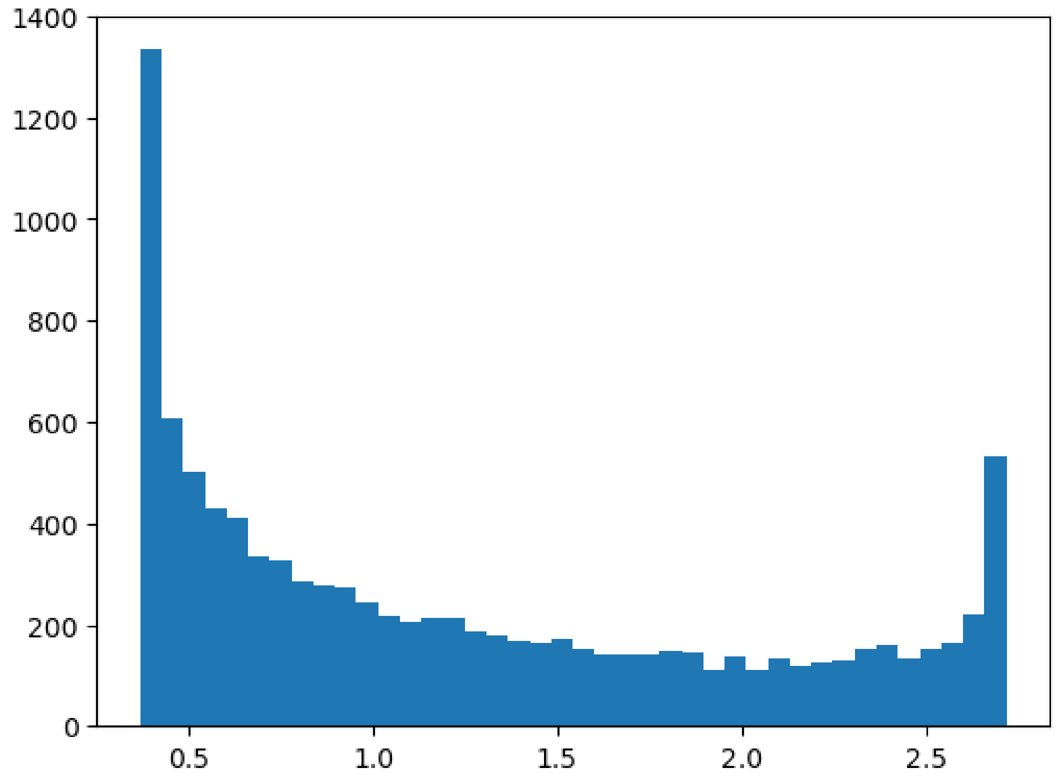
[7]:
```
## iterative method
# Note about this code: Because of random numbers, you are not guaranteed to get␣
↪a result
# that highlights the method. Try running the code several times.

N = 30
data = np.random.randn(N)
data[data<-1] = data[data<-1]+2
data[data>1.5] = data[data>1.5]**2; # try to force a few outliers

# pick a lenient threshold just for illustration
zscorethresh = 2
dataZ = (data-np.mean(data)) / np.std(data,ddof=1)

plt.figure(figsize=(10,4))

colorz = 'brkmc'
numiters = 0 # iteration counter
```

```
while True:
  # convert to z
  datamean = np.nanmean(dataZ)
  datastd  = np.nanstd(dataZ,ddof=1)
  dataZ = (dataZ-datamean) / datastd

  # find data values to remove
  toremove = dataZ>zscorethresh

  # break out of while loop if no points to remove
  if sum(toremove)==0:
    break
  else:
    # otherwise, mark the outliers in the plot
    plt.plot(np.where(toremove)[0]+numiters/
↪5,dataZ[toremove],'%sx'%colorz[numiters],
              markersize=12,markeredgewidth=3)
    dataZ[toremove] = np.nan

  # replot
  plt.plot(np.arange(N)+numiters/
↪5,dataZ,linestyle='None',marker=f'${numiters}$',markersize=12,
            color=colorz[numiters])

  # update counter
  numiters = numiters + 1

plt.ylabel('Z-score')
plt.xlabel('Data index')
plt.tight_layout()
#plt.savefig('dataQC_iterativeZmethod.png')
plt.show()
```



9 Exercise 2

[8]:
```
# create data
N = 10000
Y = np.exp(np.sin(np.random.randn(N)))

# make a copy of the data to manipulate
Yc = Y.copy()

# not specified in the instructions, but always a good idea to inspect the data!
plt.hist(Y,bins=40);
```



[9]:
```
# percent to remove (two-tailed)
k = 4

# convert that to a number of data points to remove from each tail
pnts2nan = int( (k/2)/100 * N ) # with stated parameters, this should be 200

# find the data sorting
```

```python
sort_idx = np.argsort(Y)

# nan the two tails separately
Yc[sort_idx[:pnts2nan]]  = np.nan
Yc[sort_idx[-pnts2nan:]] = np.nan

# confirm the right numbers of points
print(f'Total dataset size: {len(Yc)}')
print(f'Valid dataset size: {np.sum(~np.isnan(Yc))}')
```

```
Total dataset size: 10000
Valid dataset size: 9600
```

[10]:
```python
# compute the mean and median (also used in the next exercise)
meanY = np.mean(Y)
medianY = np.median(Y)

# print the means
print(f'Mean of original: {meanY:.3f}')
print(f'Mean of trimmed:  {np.nanmean(Yc):.3f}')

# print the medians
print(' ')
print(f'Median of original: {medianY:.3f}')
print(f'Median of trimmed:  {np.nanmedian(Yc):.3f}')
```

```
Mean of original: 1.238
Mean of trimmed:  1.225

Median of original: 1.007
Median of trimmed:  1.007
```

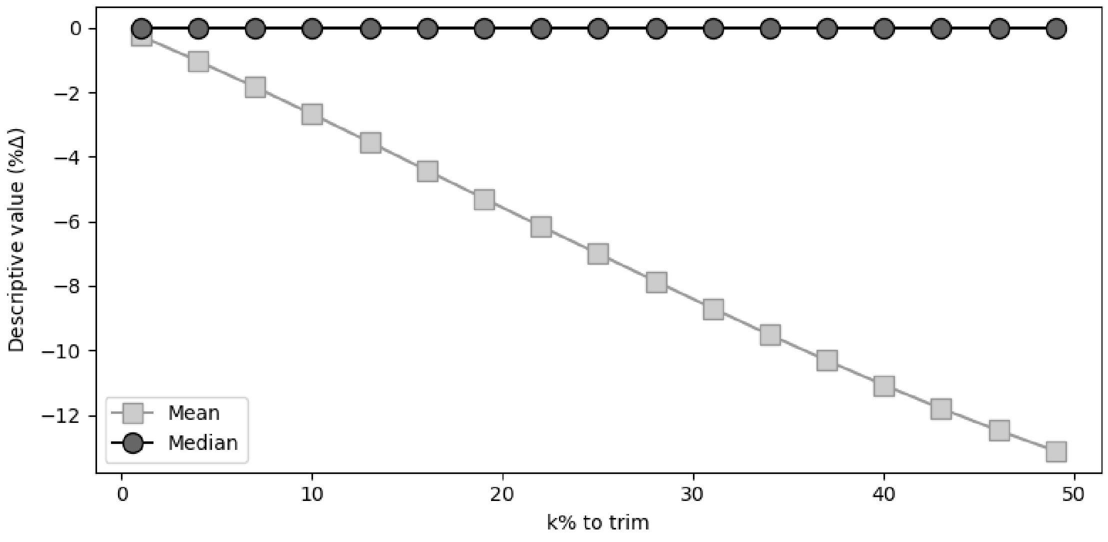
10 Exercise 3

[11]:
```
# the range of k values
ks = np.arange(1,50,3)

# initialize a results matrix for mean/median
results = np.zeros((len(ks),2))



# the experiment!
for idx,ki in enumerate(ks):

  # make a new copy of the original data
  Yc = Y.copy() # Note: Y was defined in Exercise 2

  # convert that to a number of data points to remove from each tail
  pnts2nan = int( (ki/2)/100 * N )

  # nan the two tails separately
  Yc[sort_idx[:pnts2nan]]  = np.nan
  Yc[sort_idx[-pnts2nan:]] = np.nan

  # collect mean and median
  results[idx,0] = 100*(np.nanmean(Yc)-meanY) / meanY
  results[idx,1] = 100*(np.nanmedian(Yc)-medianY) / medianY

  print(f'Total/valid dataset size: {len(Yc)} -> {np.sum(~np.isnan(Yc))}')
```

```
Total/valid dataset size: 10000 -> 9900
Total/valid dataset size: 10000 -> 9600
Total/valid dataset size: 10000 -> 9300
Total/valid dataset size: 10000 -> 9000
Total/valid dataset size: 10000 -> 8700
Total/valid dataset size: 10000 -> 8400
Total/valid dataset size: 10000 -> 8100
Total/valid dataset size: 10000 -> 7800
Total/valid dataset size: 10000 -> 7500
Total/valid dataset size: 10000 -> 7200
Total/valid dataset size: 10000 -> 6900
Total/valid dataset size: 10000 -> 6600
Total/valid dataset size: 10000 -> 6300
Total/valid dataset size: 10000 -> 6000
Total/valid dataset size: 10000 -> 5700
Total/valid dataset size: 10000 -> 5400
Total/valid dataset size: 10000 -> 5100
```

[12]:
```
# plot the results
```

```
plt.figure(figsize=(8,4))
plt.plot(ks,results[:,0],'s-',color=[.6,.6,.6],markerfacecolor=[.8,.8,.
 ↪8],markersize=10,label='Mean')
plt.plot(ks,results[:,1],'o-',color='k',markerfacecolor=[.4,.4,.
 ↪4],markersize=10,label='Median')
plt.legend()
plt.xlabel('k% to trim')
plt.ylabel(r'Descriptive value (%$\Delta$)')

plt.tight_layout()
#plt.savefig('dataQC_ex3.png')
plt.show()
```

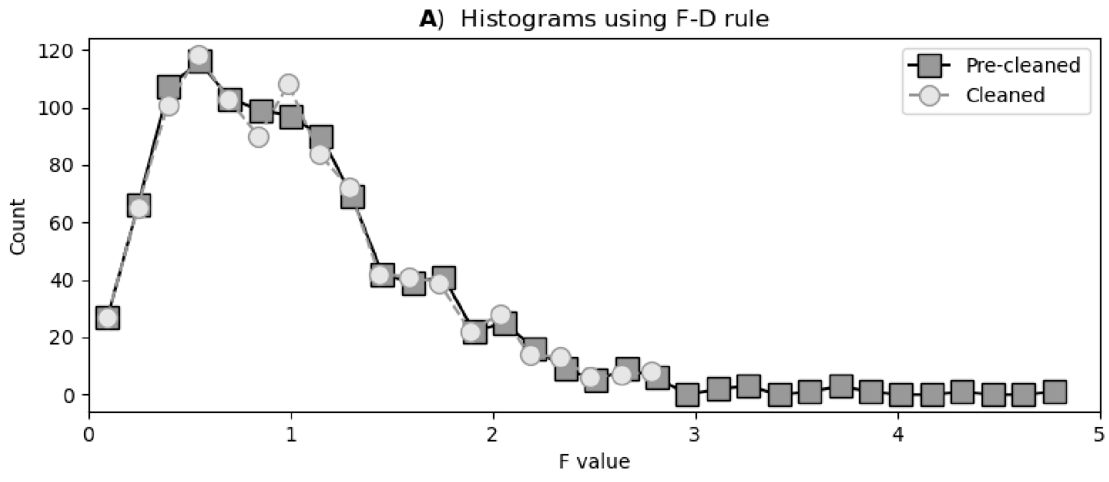


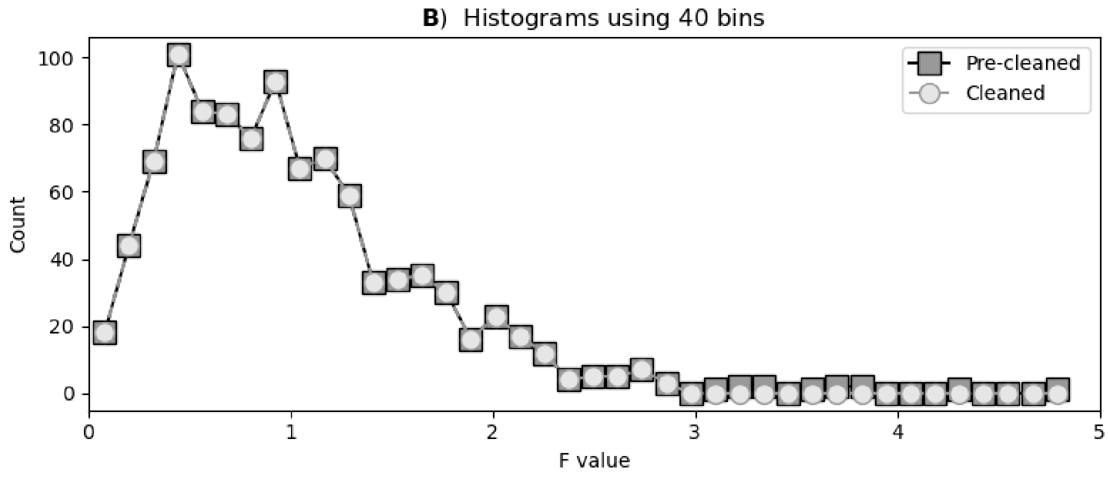
11 Exercise 4

[13]:
```
# create data
N = 1000
X = np.random.f(5,100,size=N)

# zscore data
Xz = (X-np.mean(X)) / np.std(X,ddof=1)
zThresh = 3

# clean data
Xclean = X[Xz<zThresh]

# report number of removed data points
print(f'Original sample size: {N}')
print(f'Cleaned sample size:  {len(Xclean)}')
print(f'Percent data removed: {100*(1-len(Xclean)/N):.2f}%')
```

```
Original sample size: 1000
Cleaned sample size:  988
Percent data removed: 1.20%
```

[14]:
```
# histogram bins using FD rule
y_fd_all = np.histogram(X,bins='fd')
y_fd_clean = np.histogram(Xclean,bins='fd')

# histogram bins using set boundaries
# Note that I create the bin boundaries using k+1 numbers, then input that␣
 ↪vector of boundaries into np.histogram
edges = np.linspace(np.min(X),np.max(X),41)
y_40_all = np.histogram(X,bins=edges)
y_40_clean = np.histogram(Xclean,bins=edges)

# plotting the histograms
_,axs = plt.subplots(2,1,figsize=(8,7))
axs[0].plot((y_fd_all[1][:-1]+y_fd_all[1][1:])/2,y_fd_all[0],'ks-',
         label='Pre-cleaned',markersize=11,markerfacecolor=(.6,.6,.6))
axs[0].plot((y_fd_clean[1][:-1]+y_fd_clean[1][1:])/2,y_fd_clean[0],'o--',color=(.
 ↪6,.6,.6),
         label='Cleaned',markersize=10,markerfacecolor=(.9,.9,.9))
axs[0].set_title(r'$\bf{A}$)  Histograms using F-D rule')

axs[1].plot((y_40_all[1][:-1]+y_40_all[1][1:])/2,y_40_all[0],'ks-',
         label='Pre-cleaned',markersize=11,markerfacecolor=(.6,.6,.6))
axs[1].plot((y_40_clean[1][:-1]+y_40_clean[1][1:])/2,y_40_clean[0],'o--',color=(.
 ↪6,.6,.6),
         label='Cleaned',markersize=10,markerfacecolor=(.9,.9,.9))
axs[1].set_title(r'$\bf{B}$)  Histograms using 40 bins')

# axis adjustments
for a in axs:
  a.legend()
  a.set(xlabel='F value',ylabel='Count',
        xlim=[np.min(X)-.02,np.max(X)+.02],xticks=range(6))

plt.tight_layout()
#plt.savefig('dataQC_ex4.png')
plt.show()
```


A) Histograms using F-D rule
Pre-cleaned
Cleaned
Count
F value
0
20
40
60
80
100
120
0
1
2
3
4
5

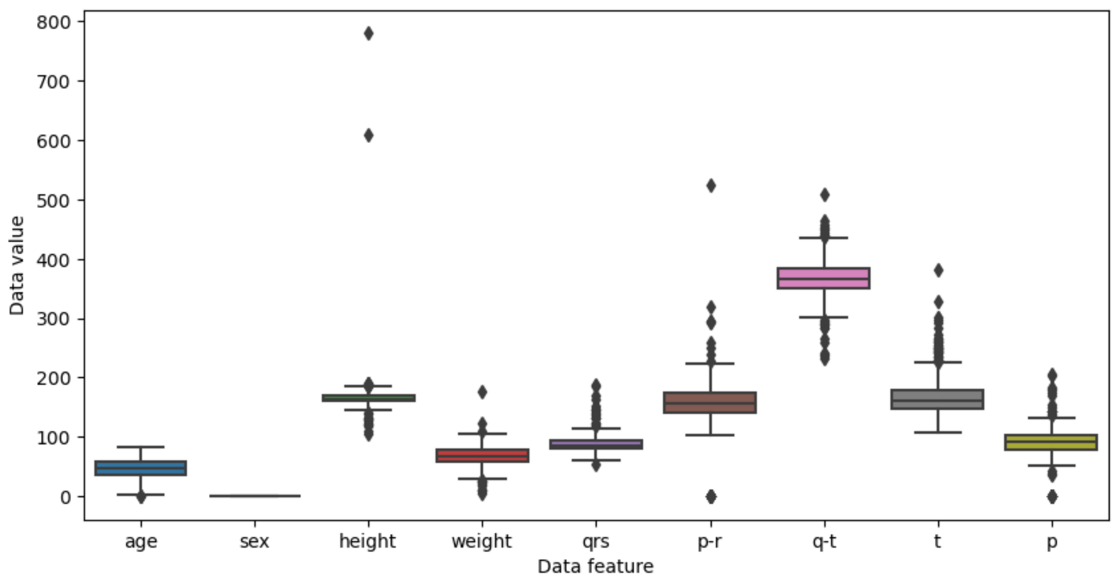

B) Histograms using 40 bins
Pre-cleaned
Cleaned
Count
F value
0
20
40
60
80
100
0
1
2
3
4
5

12 Exercise 5

[15]:
```
# url reference: https://archive.ics.uci.edu/ml/datasets/Arrhythmia

# import data
df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/
 ↪arrhythmia/arrhythmia.data',
                 usecols = np.arange(9),
                 names   =␣
 ↪['age','sex','height','weight','qrs','p-r','q-t','t','p'])
# inspect
df.head()
```

[15]:
```
   age  sex  height  weight  qrs  p-r  q-t    t    p
0   75    0     190      80   91  193  371  174  121
1   56    1     165      64   81  174  401  149   39
2   54    0     172      95  138  163  386  185  102
3   55    0     175      94  100  202  380  179  143
4   75    0     190      80   88  181  360  177  103
```

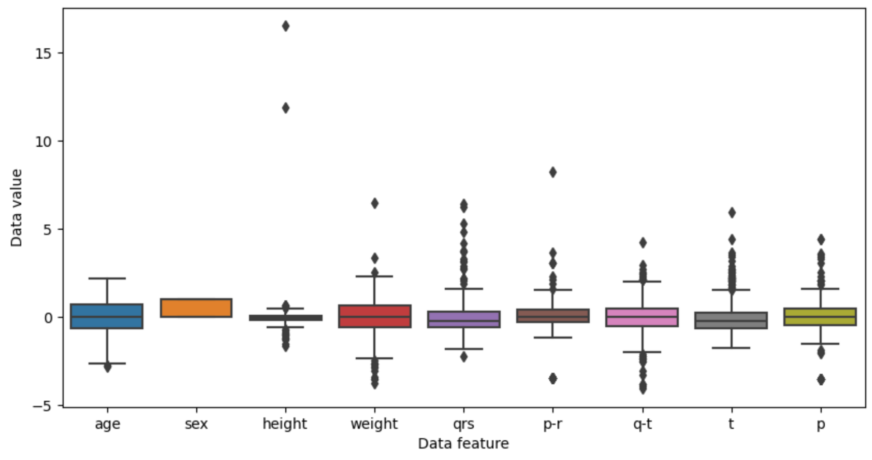
[16]:
```
# boxplots of raw data
plt.figure(figsize=(10,5))
sns.boxplot(data=df).set(xlabel='Data feature',ylabel='Data value')
plt.show()
```



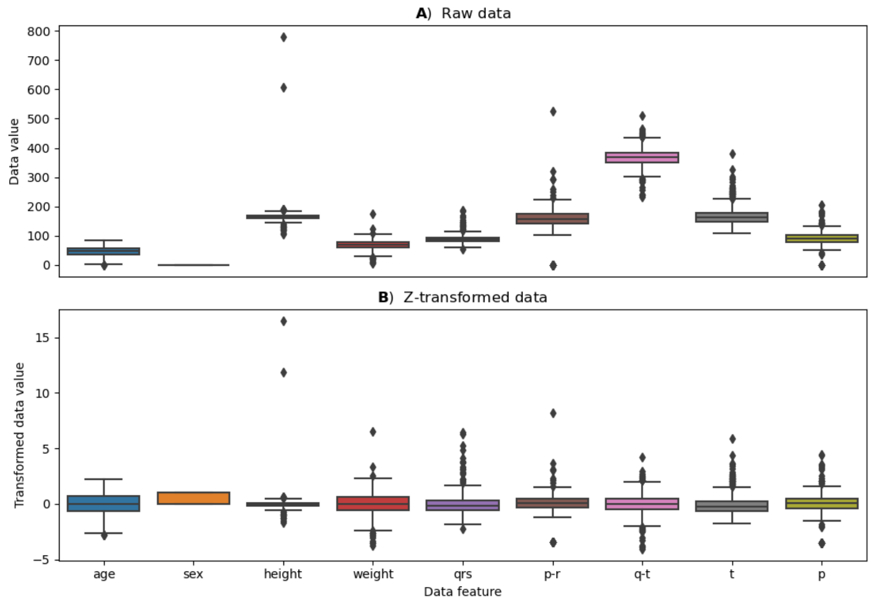
[17]:
```
# make a copy of the original data matrix
df_z = df.copy()

for col in df_z.columns:
  if not (col=='sex'):
    df_z[col] = (df[col] - df[col].mean()) / df[col].std(ddof=1)

# inspect again
df_z
```

[17]:
```
          age  sex    height    weight       qrs       p-r       q-t  \
0    1.732520    0  0.640617  0.713024  0.135355  0.844010  0.113584
1    0.578671    1 -0.031962 -0.251365 -0.515501  0.420303  1.012179
2    0.457213    0  0.156360  1.617140  3.194376  0.174999  0.562881
3    0.517942    0  0.237069  1.556865  0.721125  1.044714  0.383162
4    1.732520    0  0.640617  0.713024 -0.059902  0.576406 -0.215902
..        ...  ...       ...       ...       ...       ...       ...
447  0.396484    1 -0.166478  0.110281 -0.580586  0.977813  0.443069
448 -0.575178    0  0.640617  1.014396  0.721125 -0.404811 -0.185948
449 -0.635907    0 -0.005059 -0.010268  1.241809  0.464904 -0.066136
450 -0.878822    1 -0.300994 -0.793835  0.265526 -1.096123  0.562881
451  1.914706    1 -0.166478  0.110281 -0.645672 -0.627815 -0.096089

            t         p
0    0.113683  1.200140
1   -0.587912 -1.974876
2    0.422385  0.464465
3    0.254002  2.051973
4    0.197875  0.503185
..        ...       ...
447 -0.447593  1.045261
448  0.871406 -0.658406
449  0.674960  1.006541
450  1.348491 -1.045603
451 -0.896614 -0.464808

[452 rows x 9 columns]
```

[18]:
```
# box plots of z-scored data
plt.figure(figsize=(10,5))
sns.boxplot(data=df_z).set(xlabel='Data feature',ylabel='Data value')
plt.show()
```



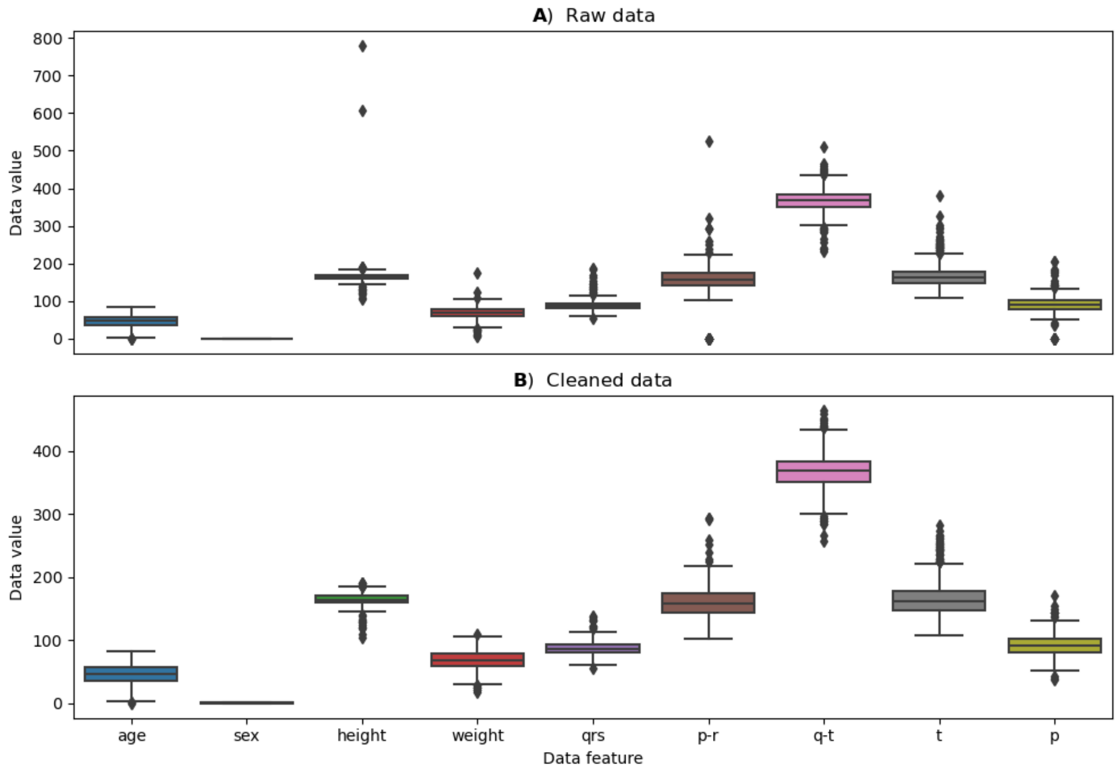
[19]:
```
# Note: this cell combines the previous graphs to make one figure for the book
_,axs = plt.subplots(2,1,figsize=(10,7))
sns.boxplot(data=df,  ax=axs[0]).set(xticks=[],ylabel='Data value')
axs[0].set_title(r'$\bf{A}$)  Raw data')
sns.boxplot(data=df_z,ax=axs[1]).set(xlabel='Data feature',ylabel='Transformed␣
 ↪data value')
axs[1].set_title(r'$\bf{B}$)  Z-transformed data')

plt.tight_layout()
#plt.savefig('dataQC_ex5b.png')
plt.show()
```



13 Exercise 6

[20]:
```
# remove based on z-score threshold
zThresh = 3.29 # p<.001

df_clean = df.copy()
df_clean[df_z>zThresh]  = np.nan # positive tail
df_clean[df_z<-zThresh] = np.nan # negative tail
```

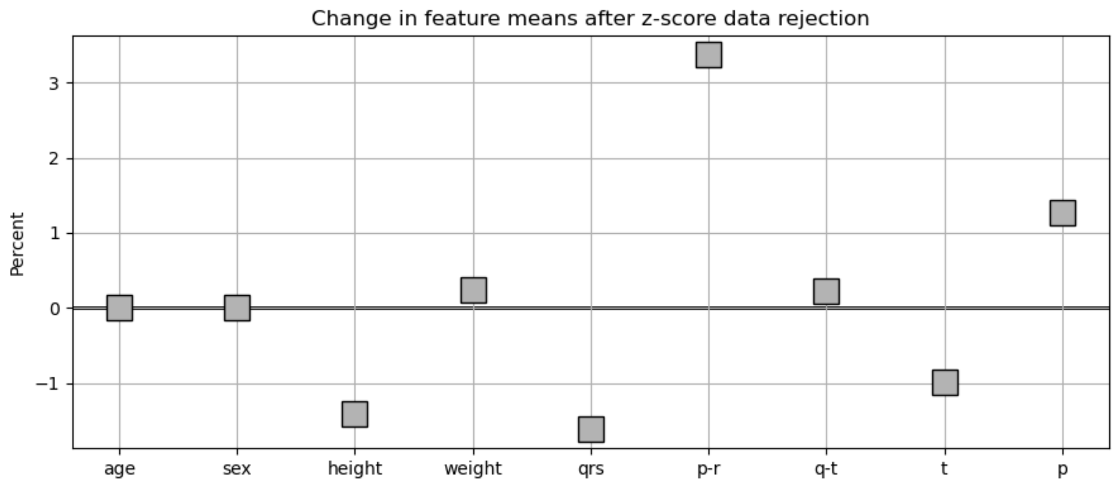
[21]:
```
# plot
_,axs = plt.subplots(2,1,figsize=(10,7))
sns.boxplot(data=df,ax=axs[0]).set(xticks=[],ylabel='Data value')
axs[0].set_title(r'$\bf{A}$)  Raw data')
sns.boxplot(data=df_clean,ax=axs[1]).set(xlabel='Data feature',ylabel='Data␣
↪value')
axs[1].set_title(r'$\bf{B}$)  Cleaned data')

plt.tight_layout()
#plt.savefig('dataQC_ex6a.png')
plt.show()
```



[22]:
```
# print the means
raw_means = df.mean().values
cleaned_means = df_clean.mean().values

for name,pre,post in zip(df.columns,raw_means,cleaned_means):
  print(f'{name:>6}: {pre:6.2f}  ->  {post:6.2f}')
```

```
   age:  46.47  ->   46.47
   sex:   0.55  ->    0.55
height: 166.19  ->  163.84
weight:  68.17  ->   68.33
   qrs:  88.92  ->   87.48
   p-r: 155.15  ->  160.38
   q-t: 367.21  ->  368.05
     t: 169.95  ->  168.28
     p:  90.00  ->   91.14
```

[23]:
```
# compute percent change
pctchange = 100*(cleaned_means-raw_means) / raw_means

# and plot
plt.figure(figsize=(9,4))
plt.plot(pctchange,'ks',markersize=14,markerfacecolor=(.7,.7,.7))
plt.axhline(0,color='k',linewidth=2,zorder=-1)
plt.xticks(range(9),labels=df.columns)
plt.ylabel('Percent')
plt.title('Change in feature means after z-score data rejection',loc='center')
plt.grid()

plt.tight_layout()
#plt.savefig('dataQC_ex6b.png')
plt.show()
```


Change in feature means after z-score data rejection
Percent
age
sex
height
weight
qrs
p-r
q-t
t
p