

Ron Kenett, Shelemyahu Zacks, Peter Gedeck

Modern Statistics: A Computer Based Approach with Python

Solutions

May 3, 2023

Springer Nature

Contents

1	Analyzing Variability: Descriptive Statistics	5
2	Probability Models and Distribution Functions	17
3	Statistical Inference and Bootstrapping	39
4	Variability in Several Dimensions and Regression Models	65
5	Sampling for Estimation of Finite Population Quantities	91
6	Time Series Analysis and Prediction	99
7	Modern analytic methods: Part I	107
8	Modern analytic methods: Part II	125

Chapter 1

Analyzing Variability: Descriptive Statistics

Import required modules and define required functions

```
import math
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import mistat
from scipy import stats

def trim_std(data, alpha):
    """ Calculate trimmed standard deviation """
    data = np.array(data)
    data.sort()
    n = len(data)
    low = int(n * alpha) + 1
    high = int(n * (1 - alpha))
    return data[low:(high + 1)].std()
```

Solution 1.1 `random.choices` selects k values from the list using sampling with replacement.

```
import random
random.seed(1)
values = random.choices([1, 2, 3, 4, 5, 6], k=50)
```

Counter counts the number of occurrences of a given value in a list.

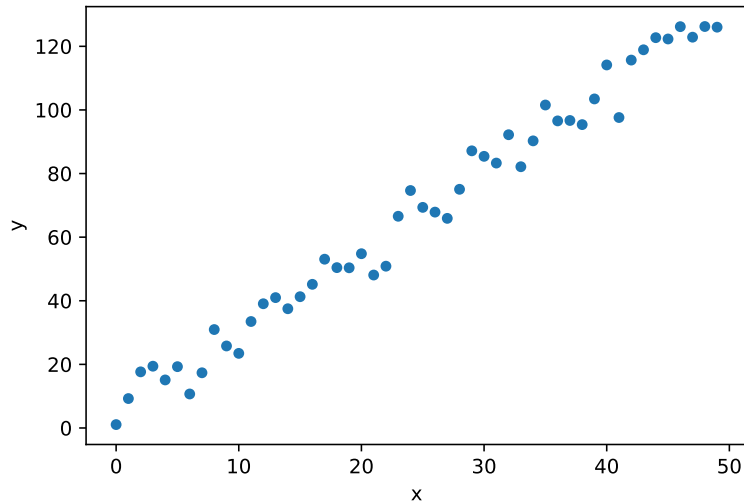
```
from collections import Counter
Counter(values)
```

```
| Counter({2: 10, 3: 10, 1: 9, 6: 9, 5: 8, 4: 4})
```

The expected frequency in each cell, under randomness is $50/6 = 8.3$. You will get different numerical results, due to randomness.

Solution 1.2 The Python function `range` is an iterator. As we need a list of values, we need to explicitly convert it.

```
x = list(range(50))
y = [5 + 2.5 * xi for xi in x]
y = [yi + random.uniform(-10, 10) for yi in y]
pd.DataFrame({'x': x, 'y': y}).plot.scatter(x='x', y='y')
plt.show()
```



Solution 1.3 In Python

```
from scipy.stats import binom
np.random.seed(1)

for p in (0.1, 0.3, 0.7, 0.9):
    X = binom.rvs(1, p, size=50)
    print(p, sum(X))
```

```
0.1 4
0.3 12
0.7 33
0.9 43
```

Notice that the expected values of the sums are 5, 15, 35 and 45.

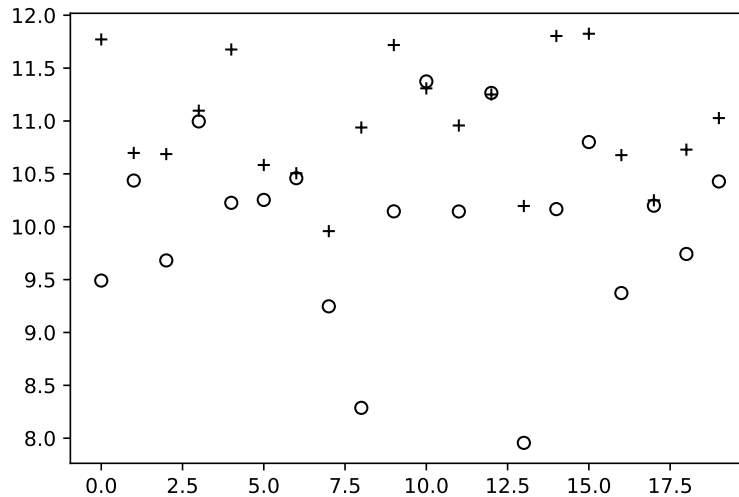
Solution 1.4 We can plot the data and calculate mean and standard deviation.

```
inst1 = [9.490950, 10.436813, 9.681357, 10.996083, 10.226101, 10.253741,
         10.458926, 9.247097, 8.287045, 10.145414, 11.373981, 10.144389,
         11.265351, 7.956107, 10.166610, 10.800805, 9.372905, 10.199018,
         9.742579, 10.428091]
inst2 = [11.771486, 10.697693, 10.687212, 11.097567, 11.676099,
         10.583907, 10.505690, 9.958557, 10.938350, 11.718334,
         11.308556, 10.957640, 11.250546, 10.195894, 11.804038,
         11.825099, 10.677206, 10.249831, 10.729174, 11.027622]
ax = pd.Series(inst1).plot(marker='o', linestyle='none',
                          fillstyle='none', color='black')
pd.Series(inst2).plot(marker='+', linestyle='none', ax=ax,
                      fillstyle='none', color='black')
```

```
plt.show()

print('mean inst1', np.mean(inst1))
print('stdev inst1', np.std(inst1, ddof=1))
print('mean inst2', np.mean(inst2))
print('stdev inst2', np.std(inst2, ddof=1))
```

```
mean inst1 10.03366815
stdev inst1 0.8708144577963102
mean inst2 10.98302505
stdev inst2 0.5685555119253366
```



As shown in the following Figure, the measurements on Instrument 1, ○, seem to be accurate but less precise than those on Instrument 2, +. Instrument 2 seems to have an upward bias (inaccurate). Quantitatively, the mean of the measurements on Instrument 1 is $\bar{X}_1 = 10.034$ and its standard deviation is $S_1 = 0.871$. For Instrument 2 we have $\bar{X}_2 = 10.983$ and $S_2 = 0.569$.

Solution 1.5 If the scale is inaccurate it will show on the average a deterministic component different than the nominal weight. If the scale is imprecise, different weight measurements will show a high degree of variability around the correct nominal weight. Problems with stability arise when the accuracy of the scale changes with time, and the scale should be recalibrated.

Solution 1.6 The method `random.choices` creates a random selection with replacement. Note that in `range(start, end)` the end argument is excluded. We therefore need to set it to 101.

```
import random
random.choices(range(1, 101), k=20)
```

```
[6, 88, 57, 20, 51, 49, 36, 35, 54, 63, 62, 46, 3, 23, 18, 59, 87, 80,
80, 82]
```

Solution 1.7 The method `random.choices` creates a random selection without replacement.

```
import random
random.sample(range(11, 31), 10)
```

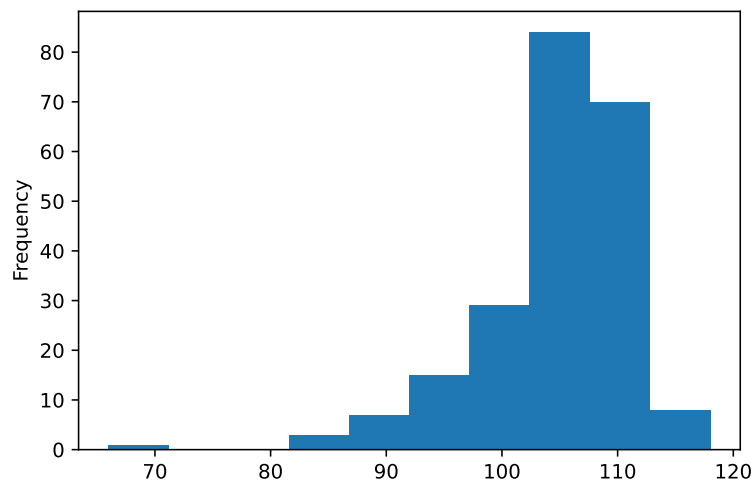
```
[19, 12, 13, 28, 11, 18, 26, 23, 15, 14]
```

Solution 1.8 (i) $26^5 = 11,881,376$; (ii) 7,893,600; (iii) $26^3 = 17,576$; (iv) $2^{10} = 1024$; (v) $\binom{10}{5} = 252$.

Solution 1.9 (i) discrete;
(ii) discrete;
(iii) continuous;
(iv) continuous.

Solution 1.10

```
mt.load_data('FILMSP')
filmsp.plot.hist()
plt.show()
```



Solution 1.11

```
mt.load_data('COAL')
pd.DataFrame(coal.value_counts(sort=False))
```

COAL	count
3	17
6	3
4	6
0	33
5	5
2	18
7	1
1	28

Solution 1.12 For (1) and (2), we can use the pandas `value_counts` method. e.g.:

```
car = mistat.load_data('CAR')
car['cyl'].value_counts(sort=False)
```

```
cyl
4    66
6    30
8    13
Name: count, dtype: int64
```

(i) Frequency distribution of number of cylinders:

Cyl	4	6	8	Total
Frequency	66	30	13	109

(ii) Frequency distribution of car's origin:

Origin	U.S.	Europe	Asia	Total
Frequency	58	14	37	109

For (3) to (5), we need to bin the data first. We can use the pandas `cut` method for this.

(iii) Frequency distribution of Turn diameter: We determine the frequency distribution on 8 intervals of length 2, from 28 to 44.

```
pd.cut(car['turn'], bins=range(28, 46, 2)).value_counts(sort=False)
```

```
turn
(28, 30]    3
(30, 32]   16
(32, 34]   16
(34, 36]   26
(36, 38]   20
(38, 40]   18
(40, 42]    8
(42, 44]    2
Name: count, dtype: int64
```

Note that the bin intervals are open on the left and closed on the right.

(iv) Frequency distribution of Horsepower:

```
pd.cut(car['hp'], bins=range(50, 275, 25)).value_counts(sort=False)
```

```
hp
(50, 75]    7
(75, 100]   34
(100, 125]  22
(125, 150]  18
(150, 175]  17
(175, 200]   6
(200, 225]   4
(225, 250]   1
Name: count, dtype: int64
```

(v) Frequency Distribution of MPG:

```
pd.cut(car['mpg'], bins=range(9, 38, 5)).value_counts(sort=False)
```

```
mpg
(9, 14]      1
(14, 19]     42
(19, 24]     41
(24, 29]     22
(29, 34]      3
Name: count, dtype: int64
```

Solution 1.13

```
film = pd.read_data('FILMSP')
film = film.sort_values(ignore_index=True) # sort and reset index

print(film.quantile(q=[0, 0.25, 0.5, 0.75, 1.0]))
print(film.quantile(q=[0.8, 0.9, 0.99]))
```

```
0.00      66.0
0.25     102.0
0.50     105.0
0.75     109.0
1.00     118.0
Name: FILMSP, dtype: float64
0.80     109.8
0.90     111.0
0.99     114.0
Name: FILMSP, dtype: float64
```

Here is a solution that uses pure Python. Note that the pandas quantile implements different interpolation methods which will lead to differences for smaller datasets. We therefore recommend using the library method and select the method that is most suitable for your use case.

```
def calculate_quantile(x, q):
    idx = (len(x) - 1) * q
    left = math.floor(idx)
    right = math.ceil(idx)
    return 0.5 * (x[left] + x[right])

for q in (0, 0.25, 0.5, 0.75, 0.8, 0.9, 0.99, 1.0):
    print(q, calculate_quantile(film, q))
```

```
0 66.0
0.25 102.0
0.5 105.0
0.75 109.0
0.8 109.5
0.9 111.0
0.99 114.0
1.0 118.0
```

Solution 1.14

```
film = pd.read_data('FILMSP')
n = len(film)
mean = film.mean()
deviations = [film - mean for film in film]
S = math.sqrt(sum(deviation**2 for deviation in deviations) / n)

skewness = sum(deviation**3 for deviation in deviations) / n / (S**3)
kurtosis = sum(deviation**4 for deviation in deviations) / n / (S**4)
```

```
print('Python:\n',
      f'Skewness: {skewness}, Kurtosis: {kurtosis}')

print('Pandas:\n',
      f'Skewness: {filmosp.skew()}, Kurtosis: {filmosp.kurtosis()}')
```

```
Python:
Skewness: -1.8098727695275856, Kurtosis: 9.014427238360716
Pandas:
Skewness: -1.8224949285588137, Kurtosis: 6.183511188870432
```

The distribution of film speed is negatively skewed and much steeper than the normal distribution. Note that the calculated values differ between the methods.

Solution 1.15 The pandas groupby method groups the data based on the value. We can then calculate individual statistics for each group.

```
car = mistat.load_data('CAR')
car['mpg'].groupby(by=car['origin']).mean()
car['mpg'].groupby(by=car['origin']).std()
# calculate both at the same time
print(car['mpg'].groupby(by=car['origin']).agg(['mean', 'std']))
```

origin	mean	std
1	20.931034	3.597573
2	19.500000	2.623855
3	23.108108	4.280341

Solution 1.16 We first create a subset of the data frame that contains only US made cars and then calculate the statistics for this subset only.

```
car = mistat.load_data('CAR')
car_US = car[car['origin'] == 1]
gamma = car_US['turn'].std() / car_US['turn'].mean()
```

Coefficient of variation $\gamma = 0.084$.

Solution 1.17

```
car_US = car[car['origin'] == 1]
car_Asia = car[car['origin'] == 3]
print('US')
print('mean', car_US['turn'].mean())
print('geometric mean', stats.gmean(car_US['turn']))
print('Japanese')
print('mean', car_Asia['turn'].mean())
print('geometric mean', stats.gmean(car_Asia['turn']))
```

```
US
mean 37.203448275862065
geometric mean 37.06877691910792
Japanese
mean 33.04594594594595
geometric mean 32.97599107553825
```

We see that \bar{X} is greater than G . The cars from Asia have smaller mean turn diameter.

Solution 1.18

```

dat.load_data('FILMSP')

Xbar = filmsp.mean()
S = filmsp.std()
print(f'mean: {Xbar}, stddev: {S}')
expected = {1: 0.68, 2: 0.95, 3: 0.997}
for k in (1, 2, 3):
    left = Xbar - k * S
    right = Xbar + k * S
    proportion = sum(left < film < right for film in filmsp)
    print(f'X +/- {k}S: ',
          f'actual freq. {proportion}, ',
          f'pred. freq. {expected[k] * len(filmsp):.2f}')

```

```

mean: 104.59447004608295, stddev: 6.547657682704987
X +/- 1S:  actual freq. 173,  pred. freq. 147.56
X +/- 2S:  actual freq. 205,  pred. freq. 206.15
X +/- 3S:  actual freq. 213,  pred. freq. 216.35

```

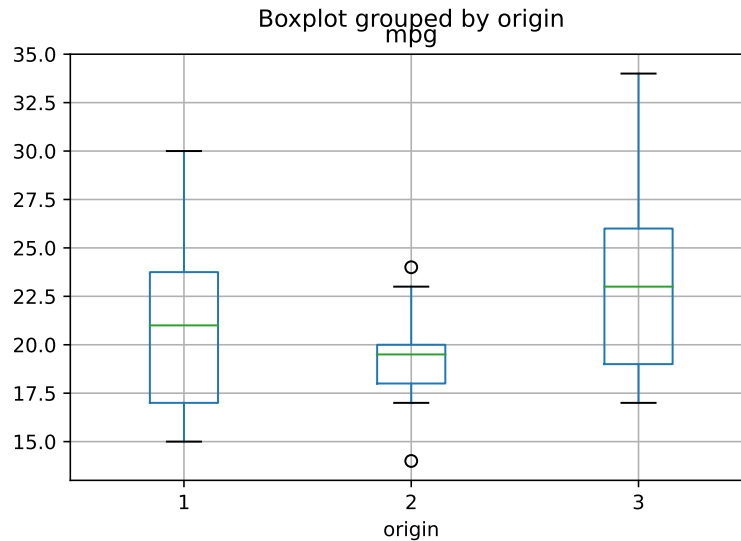
The discrepancies between the actual frequencies to the predicted frequencies are due to the fact that the distribution of film speed is neither symmetric nor bell-shaped.

Solution 1.19

```

load_data('CAR')
car.boxplot(column='mpg', by='origin')
plt.show()

```

**Solution 1.20**

```

dat.load_data('OTURB')
mistat.stemLeafDiagram(oturb, 2, leafUnit=0.01)

```

```

4 2 3444
18 2 55555666677789
40 3 0000001111112222333345
(15) 3 566677788899999
45 4 00022334444
34 4 566888999
25 5 0112333
18 5 6789
14 6 01122233444
3 6 788

```

- $X_{(1)} = 0.23$,
- $Q_1 = X_{(25.25)} = X_{(25)} + 0.25(X_{(26)} - X_{(25)}) = 0.31$,
- $M_3 = X_{(50.5)} = 0.385$,
- $Q_3 = X_{(75.75)} = 0.49 + 0.75(0.50 - 0.49) = 0.4975$,
- $X_{(n)} = 0.68$.

Solution 1.21

```

In [1]: >>> from statsmodels.stats import trim_mean
>>> oturb = mistat.load_data('OTURB')
>>> print(f'T(0.1) = {trim_mean(oturb, 0.1)}')
>>> print(f'S(0.1) = {trim_std(oturb, 0.1)}')

```

```

| T(0.1) = 0.40558750000000005
| S(0.1) = 0.09982289003530202

```

$\bar{T}_\alpha = 0.4056$ and $S_\alpha = 0.0998$, where $\alpha = 0.10$.

Solution 1.22

```

In [1]: >>> germanCars = [10, 10.9, 4.8, 6.4, 7.9, 8.9, 8.5, 6.9, 7.1,
>>> 5.5, 6.4, 8.7, 5.1, 6.0, 7.5]
>>> japaneseCars = [9.4, 9.5, 7.1, 8.0, 8.9, 7.7, 10.5, 6.5, 6.7,
>>> 9.3, 5.7, 12.5, 7.2, 9.1, 8.3, 8.2, 8.5, 6.8, 9.5, 9.7]
>>> # convert to pandas Series
>>> germanCars = pd.Series(germanCars)
>>> japaneseCars = pd.Series(japaneseCars)
>>> # use describe to calculate statistics
>>> comparison = pd.DataFrame({
>>>     'German': germanCars.describe(),
>>>     'Japanese': japaneseCars.describe(),
>>> })
>>> print(comparison)

```

```

|      count      German      Japanese
| count    15.000000    20.000000
| mean      7.373333     8.455000
| std       1.780235     1.589596
| min       4.800000     5.700000
| 25%       6.200000     7.175000
| 50%       7.100000     8.400000
| 75%       8.600000     9.425000
| max      10.900000    12.500000

```

Solution 1.23 Sample statistics:

```

hadpas = mistat.load_data('HADPAS')
sampleStatistics = pd.DataFrame({
    'res3': hadpas['res3'].describe(),
    'res7': hadpas['res7'].describe(),
})
print(sampleStatistics)

```

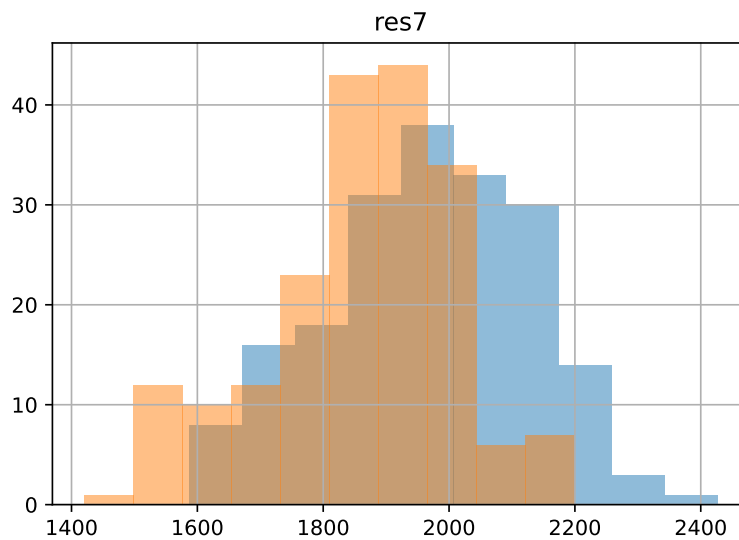
	res3	res7
count	192.000000	192.000000
mean	1965.239583	1857.776042
std	163.528165	151.535930
min	1587.000000	1420.000000
25%	1860.000000	1772.250000
50%	1967.000000	1880.000000
75%	2088.750000	1960.000000
max	2427.000000	2200.000000

Histogram:

```

ax = hadpas.hist(column='res3', alpha=0.5)
hadpas.hist(column='res7', alpha=0.5, ax=ax)
plt.show()

```



We overlay both histograms in one plot and make them transparent (alpha).

Stem and leaf diagrams:

```

print('res3')
mistat.stemLeafDiagram(hadpas['res3'], 2, leafUnit=10)
print('res7')
mistat.stemLeafDiagram(hadpas['res7'], 2, leafUnit=10)

```

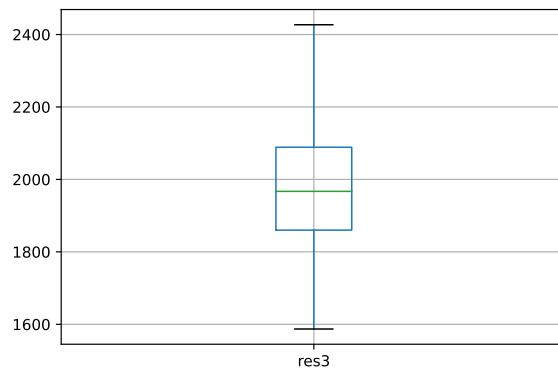
res3			
1	15	8	
6	16	01124	
14	16	56788889	

```

22      17      00000234
32      17      5566667899
45      18      0011112233444
60      18      556666677888899
87      19      000000111111223333344444444
(18)    19      566666666667888889
87      20      0000000122222333334444
64      20      5555666666677789999
44      21      0000011222233344444
25      21      566667788888
13      22      000111234
4        22      668
0        23
1        24      2
res7
1        14      2
9        15      11222244
15       15      667789
23       16      00012334
30       16      5566799
40       17      0022233334
54       17      66666666777999
79       18      00002222222233344444444
(28)    18      555555666666778888889999999
85       19      0000000011111122222233333444444
52       19      56666666777788888899999
28       20      0000111222333444
12       20      678
9        21      1123344
2        21      8
1        22      0

```

Solution 1.24.
`boxplot(column='res3')`
`plt.show()`



Lower whisker starts at $\max(1587, 1511.7) = 1587 = X_{(1)}$; upper whisker ends at $\min(2427, 2440.5) = 2427 = X_{(n)}$. There are no outliers.

Chapter 2

Probability Models and Distribution Functions

Import required modules and define required functions

```
import pandas as pd
from scipy import stats
import matplotlib.pyplot as plt
```

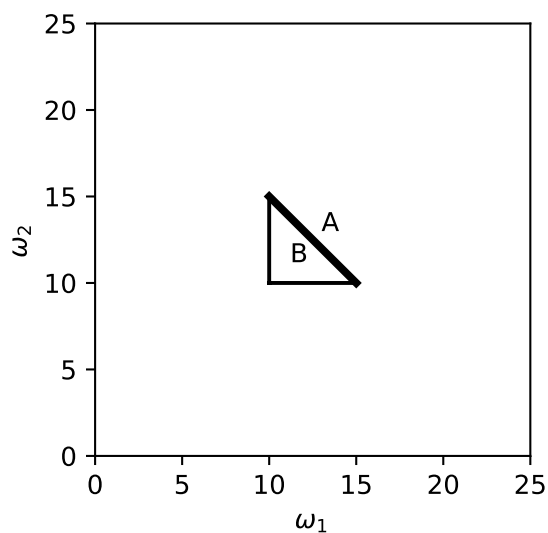
Solution 2.1 (i) $\mathcal{S} = \{(w_1, \dots, w_{20}); w_j = G, D, j = 1, \dots, 20\}$.

(ii) $2^{20} = 1,048,576$.

(iii) $A_n = \{(w_1, \dots, w_{20}) : \sum_{j=1}^{20} I\{w_j = G\} = n\}, n = 0, \dots, 20,$

where $I\{A\} = 1$ if A is true and $I\{A\} = 0$ otherwise. The number of elementary events in A_n is $\binom{20}{n} = \frac{20!}{n!(20-n)!}$.

Solution 2.2 $\mathcal{S} = \{(\omega_1, \dots, \omega_{10}) : 10 \leq \omega_i \leq 20, i = 1, \dots, 10\}$. Looking at the (ω_1, ω_2) components of A and B we have the following graphical representation:



If $(\omega_1, \dots, \omega_{10}) \in A$ then $(\omega_1, \dots, \omega_{10}) \in B$. Thus $A \subset B$. $A \cap B = A$.

Solution 2.3 (i) $\mathcal{S} = \{(i_1, \dots, i_{30}) : i_j = 0, 1, j = 1, \dots, 30\}$.

(ii) $A_{10} = \{(1, 1, \dots, 1, i_{11}, i_{12}, \dots, i_{30}) : i_j = 0, 1, j = 11, \dots, 30\}$. $|A_{10}| = 2^{20} = 1,048,576$. ($|A_{10}|$ denotes the number of elements in A_{10} .)

(iii) $B_{10} = \{(i_1, \dots, i_{30}) : i_j = 0, 1 \text{ and } \sum_{j=1}^{30} i_j = 10\}$, $|B_{10}| = \binom{30}{10} = 30,045,015$. $A_{10} \not\subset B_{10}$, in fact, A_{10} has only one element belonging to B_{10} .

Solution 2.4 $\mathcal{S} = (A \cap B) \cup (A \cap B^c) \cup (A^c \cap B) \cup (A^c \cap B^c)$, a union of mutually disjoint sets.

(a) $A \cup B = (A \cap B) \cup (A \cap B^c) \cup (A^c \cap B)$. Hence, $(A \cup B)^c = A^c \cap B^c$.

(b)

$$\begin{aligned} (A \cap B)^c &= (A \cap B^c) \cup (A^c \cap B) \cup (A^c \cap B^c) \\ &= (A \cap B^c) \cup A^c \\ &= A^c \cup B^c. \end{aligned}$$

Solution 2.5 As in Exercise 3.1, $A_n = \{(\omega_1, \dots, \omega_{20}) : \sum_{i=1}^{20} I\{\omega_i = G\} = n\}$, $n = 0, \dots, 20$. Thus, for any $n \neq n'$, $A_n \cap A_{n'} = \emptyset$, moreover $\bigcup_{n=0}^{20} A_n = \mathcal{S}$. Hence $\{A_0, \dots, A_{20}\}$ is a partition.

Solution 2.6 $\bigcup_{i=1}^n A_i = \mathcal{S}$ and $A_i \cap A_j = \emptyset$ for all $i \neq j$.

$$\begin{aligned} B &= B \cap \mathcal{S} = B \cap \left(\bigcup_{i=1}^n A_i \right) \\ &= \bigcup_{i=1}^n A_i B. \end{aligned}$$

Solution 2.7

$$\begin{aligned} \Pr\{A \cup B \cup C\} &= \Pr\{(A \cup B) \cup C\} \\ &= \Pr\{(A \cup B)\} + \Pr\{C\} - \Pr\{(A \cup B) \cap C\} \\ &= \Pr\{A\} + \Pr\{B\} - \Pr\{A \cap B\} + \Pr\{C\} \\ &\quad - \Pr\{A \cap C\} - \Pr\{B \cap C\} + \Pr\{A \cap B \cap C\} \\ &= \Pr\{A\} + \Pr\{B\} + \Pr\{C\} - \Pr\{A \cap B\} \\ &\quad - \Pr\{A \cap C\} - \Pr\{B \cap C\} + \Pr\{A \cap B \cap C\}. \end{aligned}$$

{exc:partition-union}

Solution 2.8 We have shown in Exercise 2.6 that $B = \bigcup_{i=1}^n A_i B$. Moreover, since $\{A_1, \dots, A_n\}$ is a partition, $A_i B \cap A_j B = (A_i \cap A_j) \cap B = \emptyset \cap B = \emptyset$ for all $i \neq j$. Hence, from Axiom 3

$$\Pr\{B\} = \Pr\left\{\bigcup_{i=1}^n A_i B\right\} = \sum_{i=1}^n \Pr\{A_i B\}.$$

Solution 2.9

$$\begin{aligned}
S &= \{(i_1, i_2) : i_j = 1, \dots, 6, j = 1, 2\} \\
A &= \{(i_1, i_2) : i_1 + i_2 = 10\} = \{(4, 6), (5, 5), (6, 4)\} \\
\Pr\{A\} &= \frac{3}{36} = \frac{1}{12}.
\end{aligned}$$

Solution 2.10

$$\Pr\{B\} = \Pr\{A_{150}\} - \Pr\{A_{280}\} = \exp\left(-\frac{150}{200}\right) - \exp\left(-\frac{280}{200}\right) = 0.2258.$$

Solution 2.11

$$\frac{\binom{10}{2}\binom{10}{2}\binom{15}{2}\binom{5}{2}}{\binom{40}{8}} = 0.02765$$

Solution 2.12 (i) $100^5 = 10^{10}$; (ii) $\binom{100}{5} = 75,287,520$.

Solution 2.13 $N = 1000$, $M = 900$, $n = 10$.

$$\begin{aligned}
\text{(i)} \Pr\{X \geq 8\} &= \sum_{j=8}^{10} \binom{10}{j} (0.9)^j (0.1)^{10-j} = 0.9298. \\
\text{(ii)} \Pr\{X \geq 8\} &= \sum_{j=8}^{10} \frac{\binom{900}{j} \binom{100}{10-j}}{\binom{1000}{10}} = 0.9308.
\end{aligned}$$

Solution 2.14 $1 - (0.9)^{10} = 0.6513$.

Solution 2.15 $\Pr\{T > 300 \mid T > 200\} = 0.6065$

Solution 2.16 (i) $\Pr\{D \mid B\} = \frac{1}{4}$; (ii) $\Pr\{C \mid D\} = 1$.

Solution 2.17 Since A and B are independent, $\Pr\{A \cap B\} = \Pr\{A\}\Pr\{B\}$. Using this fact and DeMorgan's Law,

$$\begin{aligned}
\Pr\{A^c \cap B^c\} &= \Pr\{(A \cup B)^c\} \\
&= 1 - \Pr\{A \cup B\} \\
&= 1 - (\Pr\{A\} + \Pr\{B\} - \Pr\{A \cap B\}) \\
&= 1 - \Pr\{A\} - \Pr\{B\} + \Pr\{A\}\Pr\{B\} \\
&= \Pr\{A^c\} - \Pr\{B\}(1 - \Pr\{A\}) \\
&= \Pr\{A^c\}(1 - \Pr\{B\}) \\
&= \Pr\{A^c\}\Pr\{B^c\}.
\end{aligned}$$

Since $\Pr\{A^c \cap B^c\} = \Pr\{A^c\}\Pr\{B^c\}$, A^c and B^c are independent.

Solution 2.18 We assume that $\Pr\{A\} > 0$ and $\Pr\{B\} > 0$. Thus, $\Pr\{A\}\Pr\{B\} > 0$. On the other hand, since $A \cap B = \emptyset$, $\Pr\{A \cap B\} = 0$.

Solution 2.19

$$\begin{aligned}
\Pr\{A \cup B\} &= \Pr\{A\} + \Pr\{B\} - \Pr\{A \cap B\} \\
&= \Pr\{A\} + \Pr\{B\} - \Pr\{A\} \Pr\{B\} \\
&= \Pr\{A\}(1 - \Pr\{B\}) + \Pr\{B\} \\
&= \Pr\{B\}(1 - \Pr\{A\}) + \Pr\{A\}.
\end{aligned}$$

Solution 2.20 By Bayes' theorem,

$$\Pr\{D \mid A\} = \frac{\Pr\{A \mid D\} \Pr\{D\}}{\Pr\{A \mid D\} \Pr\{D\} + \Pr\{A \mid G\} \Pr\{G\}} = \frac{0.10 \times 0.01}{0.10 \times 0.01 + 0.95 \times 0.99} = 0.0011.$$

Additional problems in combinatorial and geometric probabilities

Solution 2.21 Let n be the number of people in the party. The probability that all their birthdays fall on different days is $\Pr\{D_n\} = \prod_{j=1}^n \left(\frac{365-j+1}{365} \right)$.

(i) If $n = 10$, $\Pr\{D_{10}\} = 0.8831$.

(ii) If $n = 23$, $\Pr\{D_{23}\} = 0.4927$. Thus, the probability of at least 2 persons with the same birthday, when $n = 23$, is $1 - \Pr\{D_{23}\} = 0.5073 > \frac{1}{2}$.

Solution 2.22 $\left(\frac{7}{10} \right)^{10} = 0.02825$.

Solution 2.23 $\prod_{j=1}^{10} \left(1 - \frac{1}{24-j+1} \right) = 0.5833$.

Solution 2.24 (i) $\frac{\binom{4}{1}\binom{86}{4}}{\binom{100}{5}} = 0.1128$; (ii) $\frac{\binom{4}{1}\binom{10}{1}\binom{86}{3}}{\binom{100}{5}} = 0.0544$; (iii) $1 - \frac{\binom{86}{5}}{\binom{100}{5}} = 0.5374$.

Solution 2.25 $\frac{4}{\binom{10}{2}} = 0.0889$.

Solution 2.26 The sample median is $X_{(6)}$, where $X_{(1)} < \dots < X_{(11)}$ are the ordered

sample values. $\Pr\{X_{(6)} = k\} = \frac{\binom{k-1}{5}\binom{20-k}{5}}{\binom{20}{11}}$, $k = 6, \dots, 15$. This is the probability distribution of the sample median. The probabilities are

k	6	7	8	9	10	11	12	13	14	15
Pr	.01192	.04598	.09902	.15404	.18905	.18905	.15404	.09902	.04598	.01192

Solution 2.27 Without loss of generality, assume that the stick is of length 1. Let x, y and $(1-x-y)$, $0 < x, y < 1$, be the length of the 3 pieces. Obviously, $0 < x+y < 1$. All points in $\mathcal{S} = \{(x, y) : x, y > 0, x+y < 1\}$ are uniformly distributed. In order

that the three pieces can form a triangle, the following three conditions should be satisfied:

- (i) $x + y > (1 - x - y)$
- (ii) $x + (1 - x - y) > y$
- (iii) $y + (1 - x - y) > x$.

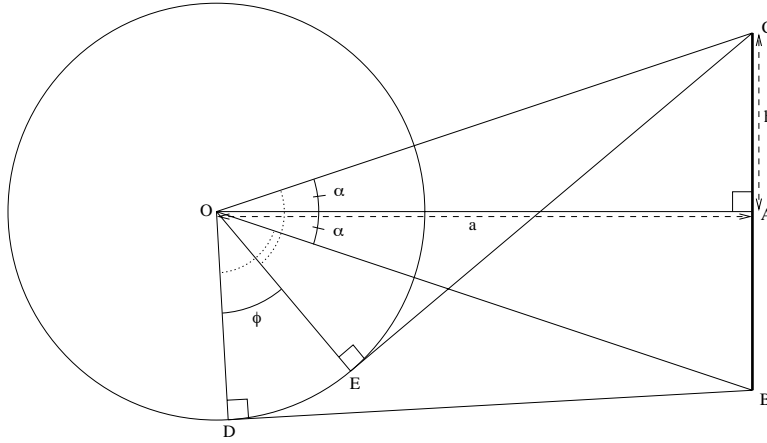
The set of points (x, y) satisfying (i), (ii) and (iii) is bounded by a triangle of area $1/8$. S is bounded by a triangle of area $1/2$. Hence, the required probability is $1/4$.

Solution 2.28 Consider Fig. 2.1. Suppose that the particle is moving along the circumference of the circle in a counterclockwise direction. Then, using the notation in the diagram, $\Pr\{\text{hit}\} = \phi/2\pi$. Since $OD = 1 = OE$, $OB = \sqrt{a^2 + h^2} = OC$ and the lines \overline{DB} and \overline{EC} are tangential to the circle, it follows that the triangles $\triangle ODB$ and $\triangle OEC$ are congruent. Thus $m(\angle DOB) = m(\angle EOC)$, and it is easily seen that $\phi = 2\alpha$. Now $\alpha = \tan^{-1}\left(\frac{h}{a}\right)$, and hence, $\Pr\{\text{hit}\} = \frac{1}{\pi} \tan^{-1}\left(\frac{h}{a}\right)$.

{exc:geometrySolution}

Fig. 2.1 Geometry of The Solution

{exc:geometrySolution}



Solution 2.29 $1 - (0.999)^{100} - 100 \times (0.001) \times (0.999)^{99} - \binom{100}{2} \times (0.001)^2 (0.999)^{98} = 0.0001504$.

Solution 2.30 The probability that n tosses are required is $p(n) = \binom{n-1}{1} \left(\frac{1}{2}\right)^n$, $n \geq 2$.

Thus, $p(4) = 3 \cdot \frac{1}{2^4} = \frac{3}{16}$.

Solution 2.31 $S = \{(i_1, \dots, i_{10}) : i_j = 0, 1, j = 1, \dots, 10\}$. One random variable is the number of 1's in an element, i.e., for $\omega = (i_1, \dots, i_{10})$ $X_1(\omega) = \sum_{j=1}^{10} i_j$. Another random variable is the number of zeros to the left of the 1st one, i.e., $X_2(\omega) = \sum_{j=1}^{10} \prod_{k=1}^j (1 - i_k)$. Notice that $X_2(\omega) = 0$ if $i_1 = 1$ and $X_2(\omega) = 10$ if

$i_1 = i_2 = \dots = i_{10} = 0$. The probability distribution of X_1 is $\Pr\{X_1 = k\} = \binom{10}{k}/2^{10}$, $k = 0, 1, \dots, 10$. The probability distribution of X_2 is

$$\Pr\{X_2 = k\} = \begin{cases} \left(\frac{1}{2}\right)^{k+1}, & k = 0, \dots, 9 \\ \left(\frac{1}{2}\right)^{10}, & k = 10. \end{cases}$$

Solution 2.32 (i) Since $\sum_{x=0}^{\infty} \frac{5^x}{x!} = e^5$, we have $\sum_{x=0}^{\infty} p(x) = 1$. ; (ii) $\Pr\{X \leq 1\} = e^{-5}(1 + 5) = 0.0404$; (iii) $\Pr\{X \leq 7\} = 0.8666$.

Solution 2.33 (i) $\Pr\{X = -1\} = 0.3$; (ii) $\Pr\{-0.5 < X < 0\} = 0.1$; (iii) $\Pr\{0 \leq X < 0.75\} = 0.425$; (iv) $\Pr\{X = 1\} = 0$; (v) $E\{X\} = -0.25$, $V\{X\} = 0.4042$.

Solution 2.34

$$E\{X\} = \int_0^{\infty} (1 - F(x)) dx = \int_0^{\infty} e^{-x^2/2\sigma^2} dx = \sigma\sqrt{\frac{\pi}{2}}.$$

Solution 2.35

$$E\{X\} = \frac{1}{N} \sum_{i=1}^N i = \frac{N+1}{2}; \quad E\{X^2\} = \frac{1}{N} \sum_{i=1}^N i^2 = \frac{(N+1)(2N+1)}{6}$$

$$V\{X\} = E\{X^2\} - (E\{X\})^2 = \frac{2(N+1)(2N+1) - 3(N+1)^2}{12} = \frac{N^2 - 1}{12}.$$

Solution 2.36 $\Pr\{8 < X < 12\} = \Pr\{|X - 10| < 2\} \geq 1 - \frac{V\{X\}}{4} = 1 - \frac{0.25}{4} = 0.9375$.

Solution 2.37 Notice that $F(x)$ is the standard Cauchy distribution. The p -th quantile, x_p , satisfies the equation $\frac{1}{2} + \frac{1}{\pi} \tan^{-1}(x_p) = p$, hence $x_p = \tan\left(\pi\left(p - \frac{1}{2}\right)\right)$. For $p = 0.25, 0.50, 0.75$ we get $x_{.25} = -1$, $x_{.50} = 0$, $x_{.75} = 1$, respectively.

Solution 2.38 $\mu_l^* = E\{(X - \mu_1)^l\} = \sum_{j=0}^l (-1)^j \binom{l}{j} \mu_1^j \mu_{l-j}$.

When $j = l$ the term is $(-1)^l \mu_1^l$. When $j = l - 1$ the term is $(-1)^{l-1} l \mu_1^{l-1} \mu_1 = (-1)^{l-1} l \mu_1^l$. Thus, the sum of the last 2 terms is $(-1)^{l-1} (l - 1) \mu_1^l$ and we have

$$\mu_l^* = \sum_{j=0}^{l-2} (-1)^j \binom{l}{j} \mu_1^j \mu_{l-j} + (-1)^{l-1} (l - 1) \mu_1^l.$$

(exc:mixed-cdf-example)

Solution 2.39 We saw in the solution of Exercise 2.33 that $\mu_1 = -0.25$. Moreover, $\mu_2 = V\{X\} + \mu_1^2 = 0.4667$.

Solution 2.40 $M_X(t) = \frac{1}{t(b-a)}(e^{tb} - e^{ta})$, $-\infty < t < \infty$, $a < b$.

$$E\{X\} = \frac{a+b}{2}; \quad V\{X\} = \frac{(b-a)^2}{12}.$$

Solution 2.41 (i) For $t < \lambda$ we have $M(t) = \left(1 - \frac{t}{\lambda}\right)^{-1}$,

$$\begin{aligned} M'(t) &= \frac{1}{\lambda} \left(1 - \frac{t}{\lambda}\right)^{-2}, & \mu_1 &= M'(0) = \frac{1}{\lambda} \\ M''(t) &= \frac{2}{\lambda^2} \left(1 - \frac{t}{\lambda}\right)^{-3}, & \mu_2 &= M''(0) = \frac{2}{\lambda^2} \\ M^{(3)}(t) &= \frac{6}{\lambda^3} \left(1 - \frac{t}{\lambda}\right)^{-4}, & \mu_3 &= M^{(3)}(0) = \frac{6}{\lambda^3} \\ M^{(4)}(t) &= \frac{24}{\lambda^4} \left(1 - \frac{t}{\lambda}\right)^{-5}, & \mu_4 &= M^{(4)}(0) = \frac{24}{\lambda^4}. \end{aligned}$$

(ii) The central moments are

$$\begin{aligned} \mu_1^* &= 0, \\ \mu_2^* &= \frac{1}{\lambda^2}, \\ \mu_3^* &= \mu_3 - 3\mu_2\mu_1 + 2\mu_1^3 = \frac{6}{\lambda^3} - \frac{6}{\lambda^3} + \frac{2}{\lambda^3} = \frac{2}{\lambda^3}, \\ \mu_4^* &= \mu_4 - 4\mu_3\mu_1 + 6\mu_2(\mu_1)^2 - 3\mu_1^4 = \frac{1}{\lambda^4}(24 - 4 \cdot 6 + 6 \cdot 2 - 3) = \frac{9}{\lambda^4}. \end{aligned}$$

(iii) The index of kurtosis is $\beta_4 = \frac{\mu_4^*}{(\mu_2^*)^2} = 9$.

Solution 2.42 `scipy.stats.binom` provides the distribution information.

```
x = list(range(15))
table = pd.DataFrame({
    'x': x,
    'p.d.f.': [stats.binom(20, 0.17).pmf(x) for x in x],
    'c.d.f.': [stats.binom(20, 0.17).cdf(x) for x in x],
})
print(table)
```

	x	p.d.f.	c.d.f.
0	0	2.407475e-02	0.024075
1	1	9.861947e-02	0.122694
2	2	1.918921e-01	0.314586
3	3	2.358192e-01	0.550406
4	4	2.052764e-01	0.755682
5	5	1.345426e-01	0.890224
6	6	6.889229e-02	0.959117
7	7	2.822094e-02	0.987338
8	8	9.392812e-03	0.996731
9	9	2.565105e-03	0.999296
10	10	5.779213e-04	0.999874
11	11	1.076086e-04	0.999981
12	12	1.653023e-05	0.999998
13	13	2.083514e-06	1.000000
14	14	2.133719e-07	1.000000

Solution 2.43 $Q_1 = 2$, Med = 3, $Q_3 = 4$.

Solution 2.44 $E\{X\} = 15.75$, $\sigma = 3.1996$.

Solution 2.45 $\Pr\{\text{no defective chip on the board}\} = p^{50}$. Solving $p^{50} = 0.99$ yields $p = (0.99)^{1/50} = 0.999799$.

Solution 2.46 Notice first that $\lim_{\substack{n \rightarrow \infty \\ np \rightarrow \lambda}} b(0; n, p) = \lim_{n \rightarrow \infty} \left(1 - \frac{\lambda}{n}\right)^n = e^{-\lambda}$.

Moreover, $j = 0, 1, \dots, n-1$, $\frac{b(j+1; n, p)}{b(j; n, p)} = \frac{n-j}{j+1} \cdot \frac{p}{1-p}$. Thus, by induction on j , for $j > 0$

$$\begin{aligned} \lim_{\substack{n \rightarrow \infty \\ np \rightarrow \lambda}} b(j; n, p) &= \lim_{\substack{n \rightarrow \infty \\ np \rightarrow \lambda}} b(j-1; n, p) \frac{n-j+1}{j} \cdot \frac{p}{1-p} \\ &= e^{-\lambda} \frac{\lambda^{j-1}}{(j-1)!} \lim_{\substack{n \rightarrow \infty \\ np \rightarrow \lambda}} \frac{(n-j+1)p}{j(1-\frac{\lambda}{n})} \\ &= e^{-\lambda} \frac{\lambda^{j-1}}{(j-1)!} \cdot \frac{\lambda}{j} = e^{-\lambda} \frac{\lambda^j}{j!}. \end{aligned}$$

Solution 2.47 Using the Poisson approximation, $\lambda = n \cdot p = 1000 \cdot 10^{-3} = 1$.

$$\Pr\{X < 4\} = e^{-1} \sum_{j=0}^3 \frac{1}{j!} = 0.9810.$$

Solution 2.48 $E\{X\} = 20 \cdot \frac{350}{500} = 14$; $V\{X\} = 20 \cdot \frac{350}{500} \cdot \frac{150}{500} \left(1 - \frac{19}{499}\right) = 4.0401$.

Solution 2.49 Let X be the number of defective items observed.

$$\Pr\{X > 1\} = 1 - \Pr\{X \leq 1\} = 1 - H(1; 500, 5, 50) = 0.0806.$$

Solution 2.50

$$\begin{aligned} \Pr\{R\} &= 1 - H(3; 100, 10, 20) + \sum_{i=1}^3 h(i; 100, 10, 20) [1 - H(3-i; 80, 10-i, 40)] \\ &= 0.87395. \end{aligned}$$

Solution 2.51 The m.g.f. of the Poisson distribution with parameter λ , $P(\lambda)$, is

$$\begin{aligned} M(t) &= \exp\{-\lambda(1 - e^t)\}, \quad -\infty < t < \infty. \text{ Accordingly,} \\ M'(t) &= \lambda M(t) e^t \\ M''(t) &= (\lambda^2 e^{2t} + \lambda e^t) M(t) \\ M^{(3)}(t) &= (\lambda^3 e^{3t} + 3\lambda^2 e^{2t} + \lambda e^t) M(t) \\ M^{(4)}(t) &= (\lambda^4 e^{4t} + 6\lambda^3 e^{3t} + 7\lambda^2 e^{2t} + \lambda e^t) M(t). \end{aligned}$$

The moments and central moments are

$$\begin{array}{ll}
\mu_1 = \lambda & \mu_1^* = 0 \\
\mu_2 = \lambda^2 + \lambda & \mu_2^* = \lambda \\
\mu_3 = \lambda^3 + 3\lambda^2 + \lambda & \mu_3^* = \lambda \\
\mu_4 = \lambda^4 + 6\lambda^3 + 7\lambda^2 + \lambda & \mu_4^* = 3\lambda^2 + \lambda.
\end{array}$$

Thus, the indexes of skewness and kurtosis are $\beta_3 = \lambda^{-1/2}$ and $\beta_4 = 3 + \frac{1}{\lambda}$.

For $\lambda = 10$ we have $\beta_3 = 0.3162$ and $\beta_4 = 3.1$.

Solution 2.52 Let X be the number of blemishes observed. $\Pr\{X > 2\} = 0.1912$.

Solution 2.53 Using the Poisson approximation with $N = 8000$ and $p = 380 \times 10^{-6}$, we have $\lambda = 3.04$ and $\Pr\{X > 6\} = 0.0356$, where X is the number of insertion errors in 2 hours of operation.

Solution 2.54 The distribution of N is geometric with $p = 0.00038$. $E\{N\} = 2631.6$, $\sigma_N = 2631.08$.

Solution 2.55 Using Python we obtain that for the $NB(p, k)$ with $p = 0.01$ and $k = 3$, $Q_1 = 170$, $Me = 265$, and $Q_3 = 389$.

```
stats.nbinom.ppf([0.25, 0.5, 0.75], 3, 0.01)
```

```
| array([170., 265., 389.])
```

Solution 2.56 By definition, the m.g.f. of $NB(p, k)$ is

$$M(t) = \sum_{i=0}^{\infty} \binom{k+i-1}{k-1} p^k ((1-p)e^t)^i,$$

for $t < -\log(1-p)$.

$$\text{Thus } M(t) = \frac{p^k}{(1-(1-p)e^t)^k} \sum_{i=0}^{\infty} \binom{k+i-1}{k-1} (1-(1-p)e^t)^k ((1-p)e^t)^i.$$

Since the last infinite series sums to one, $M(t) = \left[\frac{p}{1-(1-p)e^t} \right]^k$, $t < -\log(1-p)$.

Solution 2.57 $M(t) = \frac{pe^t}{1-(1-p)e^t}$, for $t < -\log(1-p)$. The derivatives of $M(t)$ are

$$M'(t) = M(t)(1-(1-p)e^t)^{-1}$$

$$M''(t) = M(t)(1-(1-p)e^t)^{-2}(1+(1-p)e^t)$$

$$M^{(3)}(t) = M(t)(1-(1-p)e^t)^{-3} \cdot [(1+(1-p)e^t)^2 + 2(1-p)e^t]$$

$$M^{(4)}(t) = M(t)(1-(1-p)e^t)^{-4} [1+(1-p)^3 e^{3t} + 11(1-p)e^t + 11(1-p)^2 e^{2t}].$$

The moments are

$$\begin{aligned}\mu_1 &= \frac{1}{p} \\ \mu_2 &= \frac{2-p}{p^2} \\ \mu_3 &= \frac{(2-p)^2 + 2(1-p)}{p^3} = \frac{6-6p+p^2}{p^3} \\ \mu_4 &= \frac{11(1-p)(2-p) + (1-p)^3 + 1}{p^4} = \frac{24-36p+14p^2-p^3}{p^4}.\end{aligned}$$

The central moments are

$$\begin{aligned}\mu_1^* &= 0 \\ \mu_2^* &= \frac{1-p}{p^2}, \\ \mu_3^* &= \frac{1}{p^3}(1-p)(2-p), \\ \mu_4^* &= \frac{1}{p^4}(9-18p+10p^2-p^3).\end{aligned}$$

Thus the indices of skewness and kurtosis are $\beta_3^* = \frac{2-p}{\sqrt{1-p}}$ and $\beta_4^* = \frac{9-9p+p^2}{1-p}$.

Solution 2.58 If there are n chips, $n > 50$, the probability of at least 50 good ones is $1 - B(49; n, 0.998)$. Thus, n is the smallest integer > 50 for which $B(49; n, 0.998) < 0.05$. It is sufficient to order 51 chips.

Solution 2.59 If X has a geometric distribution then, for every j , $j = 1, 2, \dots$

$\Pr\{X > j\} = (1-p)^j$. Thus,

$$\begin{aligned}\Pr\{X > n+m \mid X > m\} &= \frac{\Pr\{X > n+m\}}{\Pr\{X > m\}} \\ &= \frac{(1-p)^{n+m}}{(1-p)^m} \\ &= (1-p)^n \\ &= \Pr\{X > n\}.\end{aligned}$$

Solution 2.60 For $0 < y < 1$, $\Pr\{F(X) \leq y\} = \Pr\{X \leq F^{-1}(y)\} = F(F^{-1}(y)) = y$. Hence, the distribution of $F(X)$ is uniform on $(0, 1)$. Conversely, if U has a uniform distribution on $(0, 1)$, then

$$\Pr\{F^{-1}(U) \leq x\} = \Pr\{U \leq F(x)\} = F(x).$$

Solution 2.61 $E\{U(10, 50)\} = 30$; $V\{U(10, 50)\} = \frac{1600}{12} = 133.33$;

$$\sigma\{U(10, 50)\} = \frac{40}{2\sqrt{3}} = 11.547.$$

Solution 2.62 Let $X = -\log(U)$ where U has a uniform distribution on $(0, 1)$.

$$\begin{aligned}\Pr\{X \leq x\} &= \Pr\{-\log(U) \leq x\} \\ &= \Pr\{U \geq e^{-x}\} \\ &= 1 - e^{-x}.\end{aligned}$$

Therefore X has an exponential distribution $E(1)$.

Solution 2.63 (i) $\Pr\{92 < X < 108\} = 0.4062$; (ii) $\Pr\{X > 105\} = 0.3694$;
(iii) $\Pr\{2X + 5 < 200\} = \Pr\{X < 97.5\} = 0.4338$.

```
rv = stats.norm(100, 15)
print(' (i) ', rv.cdf(108) - rv.cdf(92))
print(' (ii) ', 1 - rv.cdf(105))
print(' (iii) ', rv.cdf((200 - 5)/2))
```

```
(i) 0.4061971427922976
(ii) 0.36944134018176367
(iii) 0.43381616738909634
```

Solution 2.64 Let z_α denote the α quantile of a $N(0, 1)$ distribution. Then the two equations $\mu + z_{.9}\sigma = 15$ and $\mu + z_{.99}\sigma = 20$ yield the solution $\mu = 8.8670$ and $\sigma = 4.7856$.

Solution 2.65 Due to symmetry, $\Pr\{Y > 0\} = \Pr\{Y < 0\} = \Pr\{E < v\}$, where $E \sim N(0, 1)$. If the probability of a bit error is $\alpha = 0.01$, then $\Pr\{E < v\} = \Phi(v) = 1 - \alpha = 0.99$.

Thus $v = z_{.99} = 2.3263$.

Solution 2.66 Let X_p denote the diameter of an aluminum pin and X_h denote the size of a hole drilled in an aluminum plate. If $X_p \sim N(10, 0.02)$ and $X_h \sim N(\mu_d, 0.02)$ then the probability that the pin will not enter the hole is $\Pr\{X_h - X_p < 0\}$. Now $X_h - X_p \sim N(\mu_d - 10, \sqrt{0.02^2 + 0.02^2})$ and for $\Pr\{X_h - X_p < 0\} = 0.01$, we obtain $\mu_d = 10.0658$ mm. (The fact that the sum of two independent normal random variables is normally distributed should be given to the student since it has not yet been covered in the text.)

Solution 2.67 For X_1, \dots, X_n i.i.d. $N(\mu, \sigma^2)$, $Y = \sum_{i=1}^n iX_i \sim N(\mu_Y, \sigma_Y^2)$ where $\mu_Y = \mu \sum_{i=1}^n i = \mu \frac{n(n+1)}{2}$ and $\sigma_Y^2 = \sigma^2 \sum_{i=1}^n i^2 = \sigma^2 \frac{n(n+1)(2n+1)}{6}$.

Solution 2.68 $\Pr\{X > 300\} = \Pr\{\log X > 5.7038\} = 1 - \Phi(0.7038) = 0.24078$.

Solution 2.69 For $X \sim e^{N(\mu, \sigma^2)}$, $X \sim e^Y$ where $Y \sim N(\mu, \sigma^2)$, $M_Y(t) = e^{\mu t + \sigma^2 t^2/2}$.

$$\xi = E\{X\} = E\{e^Y\} = M_Y(1) = e^{\mu + \sigma^2/2}.$$

Since $E\{X^2\} = E\{e^{2Y}\} = M_Y(2) = e^{2\mu + 2\sigma^2}$ we have

$$\begin{aligned} V\{X\} &= e^{2\mu + 2\sigma^2} - e^{2\mu + \sigma^2} \\ &= e^{2\mu + \sigma^2}(e^{\sigma^2} - 1) \\ &= \xi^2(e^{\sigma^2} - 1). \end{aligned}$$

Solution 2.70 The quantiles of $E(\beta)$ are $x_p = -\beta \log(1 - p)$. Hence,
 $Q_1 = 0.2877\beta$, $Me = 0.6931\beta$, $Q_3 = 1.3863\beta$.

Solution 2.71 If $X \sim E(\beta)$, $\Pr\{X > \beta\} = e^{-\beta/\beta} = e^{-1} = 0.3679$.

Solution 2.72 The m.g.f. of $E(\beta)$ is

$$\begin{aligned} M(t) &= \frac{1}{\beta} \int_0^\infty e^{tx - x/\beta} dx \\ &= \frac{1}{\beta} \int_0^\infty e^{-\frac{(1-t\beta)}{\beta}x} dx \\ &= (1 - t\beta)^{-1}, \quad \text{for } t < \frac{1}{\beta}. \end{aligned}$$

Solution 2.73 By independence,

$$\begin{aligned} M_{(X_1+X_2+X_3)}(t) &= E\{e^{t(X_1+X_2+X_3)}\} \\ &= \prod_{i=1}^3 E\{e^{tX_i}\} \\ &= (1 - \beta t)^{-3}, \quad t < \frac{1}{\beta}. \end{aligned}$$

{ex:mfg-ind-exp-rv}

Thus $X_1 + X_2 + X_3 \sim G(3, \beta)$, (see Exercise 2.76).

Using the formula of the next exercise,

$$\begin{aligned} \Pr\{X_1 + X_2 + X_3 \geq 3\beta\} &= \Pr\{\beta G(3, 1) \geq 3\beta\} \\ &= \Pr\{G(3, 1) \geq 3\} \\ &= e^{-3} \sum_{j=0}^2 \frac{3^j}{j!} \\ &= 0.4232. \end{aligned}$$

Solution 2.74

$$\begin{aligned}
G(t; k, \lambda) &= \frac{\lambda^k}{(k+1)!} \int_0^t x^{k-1} e^{-\lambda x} dx \\
&= \frac{\lambda^k}{k!} t^k e^{-\lambda t} + \frac{\lambda^{k+1}}{k!} \int_0^t x^k e^{-\lambda x} dx \\
&= \frac{\lambda^k}{k!} t^k e^{-\lambda t} + \frac{\lambda^{k+1}}{(k+1)!} t^{k+1} e^{-\lambda t} + \frac{\lambda^{k+2}}{(k+1)!} \int_0^t x^{k+1} e^{-\lambda x} dx \\
&= \dots \\
&= e^{-\lambda t} \sum_{j=k}^{\infty} \frac{(\lambda t)^j}{j!} \\
&= 1 - e^{-\lambda t} \sum_{j=0}^{k-1} \frac{(\lambda t)^j}{j!}.
\end{aligned}$$

Solution 2.75 $\Gamma(1.17) = 0.9267$, $\Gamma\left(\frac{1}{2}\right) = 1.77245$, $\Gamma\left(\frac{3}{2}\right) = \frac{1}{2}\Gamma\left(\frac{1}{2}\right) = 0.88623$.

```
from scipy.special import gamma
print(gamma(1.17), gamma(1 / 2), gamma(3 / 2))
```

```
| 0.9266996106177159 1.7724538509055159 0.8862269254527579
```

Solution 2.76 The moment generating function of the sum of independent random variables is the product of their respective m.g.f.'s. Thus, if X_1, \dots, X_k are i.i.d. $E(\beta)$, using the result of Exercise 2.73, $M_S(t) = \prod_{i=1}^k (1 - \beta t)^{-1} = (1 - \beta t)^{-k}$, $t < \frac{1}{\beta}$, where $S = \sum_{i=1}^k X_i$. On the other hand, $(1 - \beta t)^{-k}$ is the m.g.f. of $G(k, \beta)$. (exc:prob-ind-exp-rv)

Solution 2.77 The expected value and variance of $W(2, 3.5)$ are

$$\begin{aligned}
E\{W(2, 3.5)\} &= 3.5 \times \Gamma\left(1 + \frac{1}{2}\right) = 3.1018, \\
V\{W(2, 3.5)\} &= (3.5)^2 \left[\Gamma\left(1 + \frac{2}{2}\right) - \Gamma^2\left(1 + \frac{1}{2}\right) \right] = 2.6289.
\end{aligned}$$

Solution 2.78 Let T be the number of days until failure. $T \sim W(1.5, 500) \sim 500W(1.5, 1)$.

$$\Pr\{T \geq 600\} = \Pr\left\{W(1.5, 1) \geq \frac{6}{5}\right\} = e^{-(6/5)^{1.5}} = 0.2686.$$

Solution 2.79 Let $X \sim \text{Beta}\left(\frac{1}{2}, \frac{3}{2}\right)$.

$$E\{X\} = \frac{1/2}{\frac{1}{2} + \frac{3}{2}} = \frac{1}{4}, \quad V\{X\} = \frac{\frac{1}{2} \cdot \frac{3}{2}}{2^2 \cdot 3} = \frac{1}{16} \text{ and } \sigma\{X\} = \frac{1}{4}.$$

Solution 2.80 Let $X \sim \text{Beta}(\nu, \nu)$. The first four moments are

$$\begin{aligned}\mu_1 &= \nu/2\nu = \frac{1}{2} \\ \mu_2 &= \frac{B(\nu+2, \nu)}{B(\nu, \nu)} = \frac{\nu+1}{2(2\nu+1)} \\ \mu_3 &= \frac{B(\nu+3, \nu)}{B(\nu, \nu)} = \frac{(\nu+1)(\nu+2)}{2(2\nu+1)(2\nu+2)} \\ \mu_4 &= \frac{B(\nu+4, \nu)}{B(\nu, \nu)} = \frac{(\nu+1)(\nu+2)(\nu+3)}{2(2\nu+1)(2\nu+2)(2\nu+3)}.\end{aligned}$$

The variance is $\sigma^2 = \frac{1}{4(2\nu+1)}$ and the fourth central moment is

$$\mu_4^* = \mu_4 - 4\mu_3 \cdot \mu_1 + 6\mu_2 \cdot \mu_1^2 - 3\mu_1^4 = \frac{3}{16(3+8\nu+4\nu^2)}.$$

Finally, the index of kurtosis is $\beta_2 = \frac{\mu_4^*}{\sigma^4} = \frac{3(1+2\nu)}{3+2\nu}$.

Solution 2.81 Let (X, Y) have a joint p.d.f.

$$f(x, y) = \begin{cases} \frac{1}{2}, & (x, y) \in S \\ 0, & \text{otherwise.} \end{cases}$$

(i) The marginal distributions of X and Y have p.d.f.'s

$$\begin{aligned}f_X(x) &= \frac{1}{2} \int_{-1+|x|}^{1-|x|} dy = 1 - |x|, \quad -1 < x < 1, \quad \text{and by symmetry,} \\ f_Y(y) &= 1 - |y|, \quad -1 < y < 1.\end{aligned}$$

(ii) $E\{X\} = E\{Y\} = 0$, $V\{X\} = V\{Y\} = 2 \int_0^1 y^2(1-y) dy = 2B(3, 2) = \frac{1}{6}$.

Solution 2.82 The marginal p.d.f. of Y is $f(y) = e^{-y}$, $y > 0$, that is, $Y \sim E(1)$. The conditional p.d.f. of X , given $Y = y$, is $f(x | y) = \frac{1}{y} e^{-x/y}$ which is the p.d.f. of an exponential with parameter y . Thus, $E\{X | Y = y\} = y$, and $E\{X\} = E\{E\{X | Y\}\} = E\{Y\} = 1$. Also,

$$\begin{aligned}E\{XY\} &= E\{YE\{X | Y\}\} \\ &= E\{Y^2\} \\ &= 2.\end{aligned}$$

Hence, $\text{cov}(X, Y) = E\{XY\} - E\{X\}E\{Y\} = 1$. The variance of Y is $\sigma_Y^2 = 1$. The variance of X is

$$\begin{aligned}
\sigma_X^2 &= E\{V\{X | Y\}\} + V\{E\{X | Y\}\} \\
&= E\{Y^2\} + V\{Y\} \\
&= 2 + 1 = 3.
\end{aligned}$$

The correlation between X and Y is $\rho_{XY} = \frac{1}{\sqrt{3}}$.

Solution 2.83 Let (X, Y) have joint p.d.f. $f(x, y) = \begin{cases} 2, & \text{if } (x, y) \in T \\ 0, & \text{otherwise.} \end{cases}$

The marginal densities of X and Y are

$$f_X(x) = 2(1-x), \quad 0 \leq x \leq 1$$

$$f_Y(y) = 2(1-y), \quad 0 \leq y \leq 1.$$

Notice that $f(x, y) \neq f_X(x)f_Y(y)$ for $x = \frac{1}{2}, y = \frac{1}{4}$. Thus, X and Y are dependent.

$$\text{cov}(X, Y) = E\{XY\} - E\{X\}E\{Y\} = E\{XY\} - \frac{1}{9}.$$

$$\begin{aligned}
E\{XY\} &= 2 \int_0^1 x \int_0^{1-x} y \, dy \, dx \\
&= \int_0^1 x(1-x)^2 \, dx \\
&= B(2, 3) = \frac{1}{12}.
\end{aligned}$$

$$\text{Hence, } \text{cov}(X, Y) = \frac{1}{12} - \frac{1}{9} = -\frac{1}{36}.$$

Solution 2.84 $J | N \sim B(N, p); N \sim P(\lambda)$. $E\{N\} = \lambda, V\{N\} = \lambda, E\{J\} = \lambda p$.

$$\begin{aligned}
V\{J\} &= E\{V\{J | N\}\} + V\{E\{J | N\}\} & E\{JN\} &= E\{NE\{J | N\}\} \\
&= E\{Np(1-p)\} + V\{Np\} & &= pE\{N^2\} \\
&= \lambda p(1-p) + p^2\lambda = \lambda p. & &= p(\lambda + \lambda^2)
\end{aligned}$$

$$\text{Hence, } \text{cov}(J, N) = p\lambda(1+\lambda) - p\lambda^2 = p\lambda \text{ and } \rho_{JN} = \frac{p\lambda}{\lambda\sqrt{p}} = \sqrt{p}.$$

Solution 2.85 Let $X \sim G(2, 100) \sim 100G(2, 1)$ and $Y \sim W(1.5, 500) \sim 500W(1.5, 1)$.

Then $XY \sim 5 \times 10^4 G(2, 1) \cdot W(1.5, 1)$ and $V\{XY\} = 25 \times 10^8 \cdot V\{GW\}$, where

$G \sim G(2, 1)$ and $W \sim W\left(\frac{3}{2}, 1\right)$.

$$\begin{aligned}
V\{GW\} &= E\{G^2\}V\{W\} + E^2\{W\}V\{G\} \\
&= 6 \left(\Gamma\left(1 + \frac{4}{3}\right) - \Gamma^2\left(1 + \frac{2}{3}\right) \right) + 2 \cdot \Gamma^2\left(1 + \frac{2}{3}\right) \\
&= 3.88404.
\end{aligned}$$

Thus $V\{XY\} = 9.7101 \times 10^9$.

Solution 2.86 Using the notation of Example 2.33,

(i) $\Pr\{J_2 + J_3 \leq 20\} = B(20; 3500, 0.005) = 0.7699$.

(ii) $J_3 \mid J_2 = 15 \sim \text{Binomial } B\left(3485, \frac{0.004}{0.999}\right)$.

(iii) $\lambda = 3485 \times \frac{0.004}{0.999} = 13.954$, $\Pr\{J_2 \leq 15 \mid J_3 = 15\} \approx P(15; 13.954) = 0.6739$.

{ex:insert-machine-trinomial}

{ex:pdf-hypergeom-joint}

Solution 2.87 Using the notation of Example 2.34, the joint p.d.f. of J_1 and J_2 is

$$p(j_1, j_2) = \frac{\binom{20}{j_1} \binom{50}{j_2} \binom{30}{20-j_1-j_2}}{\binom{100}{20}}, \quad 0 \leq j_1, j_2; j_1 + j_2 \leq 20.$$

The marginal distribution of J_1 is $H(100, 20, 20)$. The marginal distribution of J_2 is $H(100, 50, 20)$. Accordingly,

$$V\{J_1\} = 20 \times 0.2 \times 0.8 \times \left(1 - \frac{19}{99}\right) = 2.585859,$$

$$V\{J_2\} = 20 \times 0.5 \times 0.5 \times \left(1 - \frac{19}{99}\right) = 4.040404.$$

The conditional distribution of J_1 , given J_2 , is $H(50, 20, 20 - J_2)$. Hence,

$$\begin{aligned} E\{J_1 J_2\} &= E\{E\{J_1 J_2 \mid J_2\}\} \\ &= E\left\{J_2(20 - J_2) \times \frac{2}{5}\right\} \\ &= 8E\{J_2\} - 0.4E\{J_2^2\} \\ &= 80 - 0.4 \times 104.040404 = 38.38381 \end{aligned}$$

$$\text{and } \text{cov}(J_1, J_2) = -1.61616.$$

$$\text{Finally, the correlation between } J_1 \text{ and } J_2 \text{ is } \rho = \frac{-1.61616}{\sqrt{2.585859 \times 4.040404}} = -0.50.$$

Solution 2.88 $V\{Y \mid X\} = 150$, $V\{Y\} = 200$, $V\{Y \mid X\} = V\{Y\}(1 - \rho^2)$. Hence $|\rho| = 0.5$. The sign of ρ cannot be determined.

Solution 2.89 (i) $X_{(1)} \sim E\left(\frac{100}{10}\right)$, $E\{X_{(1)}\} = 10$; (ii) $E\{X_{(10)}\} = 100 \sum_{i=1}^{10} \frac{1}{i} = 292.8968$.

Solution 2.90 $J \sim B(10, 0.95)$. If $\{J = j\}$, $j > 1$, $X_{(1)}$ is the minimum of a sample of j i.i.d. $E(10)$ random variables. Thus $X_{(1)} \mid J = j \sim E\left(\frac{10}{j}\right)$.

(i) $\Pr\{J = k, X_{(1)} \leq x\} = b(k; 10, 0.95)(1 - e^{-\frac{kx}{10}})$, $k = 1, 2, \dots, 10$.

(ii) First note that $\Pr\{J \geq 1\} = 1 - (0.05)^{10} \approx 1$.

$$\begin{aligned}
\Pr\{X_{(1)} \leq x \mid J \geq 1\} &= \sum_{k=1}^{10} b(k; 10, 0.95)(1 - e^{-\frac{kx}{10}}) \\
&= 1 - \sum_{k=1}^{10} \binom{10}{k} (0.95e^{-\frac{x}{10}})^k (0.05)^{(10-k)} \\
&= 1 - [0.05 + 0.95e^{-x/10}]^{10} + (0.05)^{10} \\
&= 1 - (0.05 + 0.95e^{-x/10})^{10}.
\end{aligned}$$

Solution 2.91 The median is $Me = X_{(6)}$.

(a) The p.d.f. of Me is $f_{(6)}(x) = \frac{11!}{5!5!} \lambda (1 - e^{-\lambda x})^5 e^{-6\lambda x}$, $x \geq 0$.

(b) The expected value of Me is

$$\begin{aligned}
E\{X_{(6)}\} &= 2772\lambda \int_0^\infty x(1 - e^{-\lambda x})^5 e^{-6\lambda x} dx \\
&= 2772\lambda \sum_{j=0}^5 (-1)^j \binom{5}{j} \int_0^\infty x e^{-\lambda x(6+j)} dx \\
&= 2772 \sum_{j=0}^5 (-1)^j \binom{5}{j} \frac{1}{\lambda(6+j)^2} \\
&= 0.73654/\lambda.
\end{aligned}$$

Solution 2.92 Let X and Y be i.i.d. $E(\beta)$, $T = X + Y$ and $W = X - Y$.

$$\begin{aligned}
V\left\{T + \frac{1}{2}W\right\} &= V\left\{\frac{3}{2}X + \frac{1}{2}Y\right\} \\
&= \beta^2 \left(\left(\frac{3}{2}\right)^2 + \left(\frac{1}{2}\right)^2 \right) \\
&= 2.5\beta^2.
\end{aligned}$$

Solution 2.93 $\text{cov}(X, X + Y) = \text{cov}(X, X) + \text{cov}(X, Y) = V\{X\} = \sigma^2$.

Solution 2.94 $V\{\alpha X + \beta Y\} = \alpha^2 \sigma_X^2 + \beta^2 \sigma_Y^2 + 2\alpha\beta \text{cov}(X, Y) = \alpha^2 \sigma_X^2 + \beta^2 \sigma_Y^2 + 2\alpha\beta \rho_{XY} \sigma_X \sigma_Y$.

Solution 2.95 Let $U \sim N(0, 1)$ and $X \sim N(\mu, \sigma)$. We assume that U and X are independent. Then $\Phi(X) = \Pr\{U < X \mid X\}$ and therefore

$$\begin{aligned}
E\{\Phi(X)\} &= E\{\Pr\{U < X \mid X\}\} \\
&= \Pr\{U < X\} \\
&= \Pr\{U - X < 0\} \\
&= \Phi\left(\frac{\mu}{\sqrt{1 + \sigma^2}}\right).
\end{aligned}$$

The last equality follows from the fact that $U - X \sim N(-\mu, \sqrt{1 + \sigma^2})$.

Solution 2.96 Let U_1, U_2, X be independent random variables; U_1, U_2 i.i.d. $N(0, 1)$. Then

$$\Phi^2(X) = \Pr\{U_1 \leq X, U_2 \leq X \mid X\}. \text{ Hence}$$

$$E\{\Phi^2(X)\} = \Pr\{U_1 \leq X, U_2 \leq X\} = \Pr\{U_1 - X \leq 0, U_2 - X \leq 0\}.$$

Since $(U_1 - X, U_2 - X)$ have a bivariate normal distribution with means $(-\mu, -\mu)$

and variance-covariance matrix $V = \begin{bmatrix} 1 + \sigma^2 & \sigma^2 \\ \sigma^2 & 1 + \sigma^2 \end{bmatrix}$, it follows that

$$E\{\Phi^2(X)\} = \Phi_2\left(\frac{\mu}{\sqrt{1 + \sigma^2}}, \frac{\mu}{\sqrt{1 + \sigma^2}}; \frac{\sigma^2}{1 + \sigma^2}\right).$$

Solution 2.97 Since X and Y are independent, $T = X + Y \sim P(12)$, $\Pr\{T > 15\} = 0.1556$.

Solution 2.98 Let $F_2(x) = \int_{-\infty}^x f_2(z) dz$ be the c.d.f. of X_2 . Since $X_1 + X_2$ are

$$\Pr\{Y \leq y\} = \int_{-\infty}^{\infty} f_1(x) \Pr\{X_2 \leq y - x\} dx$$

independent

$$= \int_{-\infty}^{\infty} f_1(x) F_2(y - x) dx.$$

$$g(y) = \frac{d}{dy} \Pr\{Y \leq y\}$$

Therefore, the p.d.f. of Y is

$$= \int_{-\infty}^{\infty} f_1(x) f_2(y - x) dx.$$

Solution 2.99 Let $Y = X_1 + X_2$ where X_1, X_2 are i.i.d. uniform on $(0, 1)$. Then the p.d.f.'s are

$$f_1(x) = f_2(x) = I\{0 < x < 1\}$$

$$g(y) = \begin{cases} \int_0^y dx = y, & \text{if } 0 \leq y < 1 \\ \int_{y-1}^1 dx = 2 - y, & \text{if } 1 \leq y \leq 2. \end{cases}$$

Solution 2.100 X_1, X_2 are i.i.d. $E(1)$. $U = X_1 - X_2$. $\Pr\{U \leq u\} = \int_0^{\infty} e^{-x} \Pr\{X_1 \leq u + x\} dx$. Notice that $-\infty < u < \infty$ and $\Pr\{X_1 \leq u + x\} = 0$ if $x + u < 0$. Let $a^+ = \max(a, 0)$. Then

$$\begin{aligned} \Pr\{U \leq u\} &= \int_0^{\infty} e^{-x} (1 - e^{-(u+x)^+}) dx \\ &= 1 - \int_0^{\infty} e^{-x-(u+x)^+} dx \\ &= \begin{cases} 1 - \frac{1}{2}e^{-u}, & \text{if } u \geq 0 \\ \frac{1}{2}e^{-|u|}, & \text{if } u < 0 \end{cases} \end{aligned}$$

Thus, the p.d.f. of U is $g(u) = \frac{1}{2}e^{-|u|}$, $-\infty < u < \infty$.

Solution 2.101 $T = X_1 + \cdots + X_{20} \sim N(20\mu, \sqrt{20}\sigma^2)$. $\Pr\{T \leq 50\} \approx \Phi\left(\frac{50 - 40}{44.7214}\right) = 0.5885$.

Solution 2.102 $X \sim B(200, 0.15)$. $\mu = np = 30$, $\sigma = \sqrt{np(1-p)} = 5.0497$.

$$\Pr\{25 < X < 35\} \approx \Phi\left(\frac{34.5 - 30}{5.0497}\right) - \Phi\left(\frac{25.5 - 30}{5.0497}\right) = 0.6271.$$

Solution 2.103 $X \sim P(200)$. $\Pr\{190 < X < 210\} \approx 2\Phi\left(\frac{9.5}{\sqrt{200}}\right) - 1 = 0.4983$.

Solution 2.104 $X \sim \text{Beta}(3, 5)$. $\mu = E\{X\} = \frac{3}{8} = 0.375$. $\sigma = \sqrt{V\{X\}} = \left(\frac{3 \cdot 5}{8^2 \cdot 9}\right)^{1/2} = 0.161374$.

$$\Pr\{|\bar{X}_{200} - 0.375| < 0.2282\} \approx 2\Phi\left(\frac{\sqrt{200} \cdot 0.2282}{0.161374}\right) - 1 = 1.$$

Solution 2.105 $t_{.95}[10] = 1.8125$, $t_{.95}[15] = 1.7531$, $t_{.95}[20] = 1.7247$.

```
print(stats.t.ppf(0.95, 10))
print(stats.t.ppf(0.95, 15))
print(stats.t.ppf(0.95, 20))
```

```
| 1.8124611228107335
| 1.7530503556925547
| 1.7247182429207857
```

Solution 2.106 $F_{.95}[10, 30] = 2.1646$, $F_{.95}[15, 30] = 2.0148$, $F_{.95}[20, 30] = 1.9317$.

```
print(stats.f.ppf(0.95, 10, 30))
print(stats.f.ppf(0.95, 15, 30))
print(stats.f.ppf(0.95, 20, 30))
```

```
| 2.164579917125473
| 2.0148036912954885
| 1.931653475236928
```

Solution 2.107 The solution to this problem is based on the fact, which is not discussed in the text, that $t[\nu]$ is distributed like the ratio of two independent random variables, $N(0, 1)$ and $\sqrt{\chi^2[\nu]/\nu}$. Accordingly, $t[n] \sim \frac{N(0, 1)}{\sqrt{\frac{\chi^2[n]}{n}}}$, where $N(0, 1)$ and

$\chi^2[n]$ are independent. $t^2[n] \sim \frac{(N(0, 1))^2}{\frac{\chi^2[n]}{n}} \sim F[1, n]$. Thus, since $\Pr\{F[1, n] \leq F_{1-\alpha}[1, n]\} = 1 - \alpha$.

$$\Pr\{-\sqrt{F_{1-\alpha}[1, n]} \leq t[n] \leq \sqrt{F_{1-\alpha}[1, n]}\} = 1 - \alpha.$$

It follows that $\sqrt{F_{1-\alpha}[1, n]} = t_{1-\alpha/2}[n]$,

or $F_{1-\alpha}[1, n] = t_{1-\alpha/2}^2[n]$. (If you assign this problem, please inform the students of the above fact.)

Solution 2.108 A random variable $F[v_1, v_2]$ is distributed like the ratio of two independent random variables $\chi^2[v_1]/v_1$ and $\chi^2[v_2]/v_2$. Accordingly, $F[v_1, v_2] \sim \frac{\chi^2[v_1]/v_1}{\chi^2[v_2]/v_2}$ and

$$\begin{aligned} 1 - \alpha &= \Pr\{F[v_1, v_2] \leq F_{1-\alpha}[v_1, v_2]\} \\ &= \Pr\left\{\frac{\chi_1^2[v_1]/v_1}{\chi_2^2[v_2]/v_2} \leq F_{1-\alpha}[v_1, v_2]\right\} \\ &= \Pr\left\{\frac{\chi_2^2[v_2]/v_2}{\chi_1^2[v_1]/v_1} \geq \frac{1}{F_{1-\alpha}[v_1, v_2]}\right\} \\ &= \Pr\left\{F[v_2, v_1] \geq \frac{1}{F_{1-\alpha}[v_1, v_2]}\right\} \\ &= \Pr\{F[v_2, v_1] \geq F_\alpha[v_2, v_1]\}. \end{aligned}$$

$$\text{Hence } F_{1-\alpha}[v_1, v_2] = \frac{1}{F_\alpha[v_2, v_1]}.$$

Solution 2.109 Using the fact that $t[v] \sim \frac{N(0, 1)}{\sqrt{\frac{\chi^2[v]}{v}}}$, where $N(0, 1)$ and $\chi^2[v]$ are independent,

$$\begin{aligned} V\{t[v]\} &= V\left\{\frac{N(0, 1)}{\sqrt{\chi^2[v]/v}}\right\} \\ &= E\left\{V\left\{\frac{N(0, 1)}{\sqrt{\frac{\chi^2[v]}{v}}}\right\} \middle| \chi^2[v]\right\} + V\left\{E\left\{\frac{N(0, 1)}{\sqrt{\frac{\chi^2[v]}{v}}}\right\} \middle| \chi^2[v]\right\}. \end{aligned}$$

$$\text{By independence, } V\left\{\frac{N(0, 1)}{\sqrt{\frac{\chi^2[v]}{v}}}\right\} \middle| \chi^2[v] = \frac{v}{\chi^2[v]}, \text{ and } E\left\{\frac{N(0, 1)}{\sqrt{\frac{\chi^2[v]}{v}}}\right\} \middle| \chi^2[v] = 0.$$

$$\text{Thus, } V\{t[v]\} = vE\left\{\frac{1}{\chi^2[v]}\right\}. \text{ Since } \chi^2[v] \sim G\left(\frac{v}{2}, 2\right) \sim 2G\left(\frac{v}{2}, 1\right),$$

$$\begin{aligned} E\left\{\frac{1}{\chi^2[v]}\right\} &= \frac{1}{2} \cdot \frac{1}{\Gamma\left(\frac{v}{2}\right)} \int_0^\infty x^{v-2} e^{-x} dx \\ &= \frac{1}{2} \cdot \frac{\Gamma\left(\frac{v}{2} - 1\right)}{\Gamma\left(\frac{v}{2}\right)} = \frac{1}{2} \cdot \frac{1}{\frac{v}{2} - 1} = \frac{1}{v-2}. \end{aligned}$$

Finally, $V\{t[\nu]\} = \frac{\nu}{\nu - 2}$, $\nu > 2$.

Solution 2.110 $E\{F[3, 10]\} = \frac{10}{8} = 1.25$, $V\{F[3, 10]\} = \frac{2 \cdot 10^2 \cdot 11}{3 \cdot 8^2 \cdot 6} = 1.9097$.

Chapter 3

Statistical Inference and Bootstrapping

Import required modules and define required functions

```
import random
import math
import numpy as np
import pandas as pd
from scipy import stats
import matplotlib.pyplot as plt
import pingouin as pg
import mistat
import os
os.environ['OUTDATED_IGNORE'] = '1'
```

Solution 3.1 By the WLLN, for any $\epsilon > 0$, $\lim_{n \rightarrow \infty} \Pr\{|M_l - \mu_l| < \epsilon\} = 1$. Hence, M_l is a consistent estimator of the l -th moment.

Solution 3.2 Using the CLT, $\Pr\{|\bar{X}_n - \mu| < 1\} \approx 2\Phi\left(\frac{\sqrt{n}}{\sigma}\right) - 1$. To determine the sample size n so that this probability is 0.95 we set $2\Phi\left(\frac{\sqrt{n}}{\sigma}\right) - 1 = 0.95$ and solve for n . This gives $\frac{\sqrt{n}}{\sigma} = z_{.975} = 1.96$. Thus $n \geq \sigma^2(1.96)^2 = 424$ for $\sigma = 10.5$.

Solution 3.3 $\hat{\xi}_p = \bar{X}_n + z_p \hat{\sigma}_n$, where $\hat{\sigma}_n^2 = \frac{1}{n} \sum (X_i - \bar{X}_n)^2$.

Solution 3.4 Let $(X_1, Y_1), \dots, (X_n, Y_n)$ be a random sample from a bivariate normal distribution with density $f(x, y; \mu, \eta, \sigma_X, \sigma_Y, \rho)$ as given in Eq. (4.6.6). Let $Z_i = X_i Y_i$ for $i = 1, \dots, n$. Then the first moment of Z is given by

$$\begin{aligned} \mu_1(F_Z) &= E\{Z\} \\ &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} xy f(x, y; \mu, \eta, \sigma_X, \sigma_Y, \rho) \, dx \, dy \\ &= \mu\eta + \rho\sigma_X\sigma_Y. \end{aligned}$$

Using this fact, as well as the first 2 moments of X and Y , we get the following moment equations:

$$\begin{aligned}\frac{1}{n} \sum_{i=1}^n X_i &= \mu & \frac{1}{n} \sum_{i=1}^n Y_i &= \eta \\ \frac{1}{n} \sum_{i=1}^n X_i^2 &= \sigma_X^2 + \mu^2 & \frac{1}{n} \sum_{i=1}^n Y_i^2 &= \sigma_Y^2 + \eta^2 \\ \frac{1}{n} \sum_{i=1}^n X_i Y_i &= \mu\eta + \rho\sigma_X\sigma_Y.\end{aligned}$$

Solving these equations for the 5 parameters gives

$$\begin{aligned}\hat{\rho}_n &= \frac{\frac{1}{n} \sum_{i=1}^n X_i Y_i - \bar{X}\bar{Y}}{\left[\left(\frac{1}{n} \sum_{i=1}^n X_i^2 - \bar{X}^2 \right) \cdot \left(\frac{1}{n} \sum_{i=1}^n Y_i^2 - \bar{Y}^2 \right) \right]^{1/2}}, \quad \text{or equivalently,} \\ \hat{\rho}_n &= \frac{\frac{1}{n} \sum_{i=1}^n (X_i - \bar{X})(Y_i - \bar{Y})}{\left(\frac{1}{n} \sum_{i=1}^n (X_i - \bar{X})^2 \cdot \frac{1}{n} \sum_{i=1}^n (Y_i - \bar{Y})^2 \right)^{1/2}}.\end{aligned}$$

Solution 3.5 The two first moments are

$$\mu_1 = \frac{\nu_1}{\nu_1 + \nu_2}, \quad \mu_2 = \frac{\nu_1(\nu_1 + 1)}{(\nu_1 + \nu_2)(\nu_1 + \nu_2 + 1)}.$$

Equating the theoretical moments to the sample moments $M_1 = \frac{1}{n} \sum_{i=1}^n X_i$ and $M_2 = \frac{1}{n} \sum_{i=1}^n X_i^2$, we obtain with $\hat{\sigma}_n^2 = M_2 - M_1^2$ the moment equation estimators.

$$\hat{\nu}_1 = M_1(M_1 - M_2)/\hat{\sigma}_n^2 \quad \text{and} \quad \hat{\nu}_2 = (M_1 - M_2)(1 - M_1)/\hat{\sigma}_n^2.$$

Solution 3.6 $V\{\bar{Y}_w\} = \left(\sum_{i=1}^k w_i^2 \frac{\sigma_i^2}{n_i} \right) / \left(\sum_{i=1}^k w_i \right)^2$. Let $\lambda_i = \frac{w_i}{\sum_{j=1}^k w_j}$, $\sum_{i=1}^k \lambda_i = 1$.

We find weights λ_i which minimize $V\{\bar{Y}_w\}$, under the constraint $\sum_{i=1}^k \lambda_i = 1$. The Lagrangian is $L(\lambda_1, \dots, \lambda_k, \rho) = \sum_{i=1}^k \lambda_i^2 \frac{\sigma_i^2}{n_i} + \rho \left(\sum_{i=1}^k \lambda_i - 1 \right)$. Differentiating with respect to λ_i , we get

$$\frac{\partial}{\partial \lambda_i} L(\lambda_1, \dots, \lambda_k, \rho) = 2\lambda_i \frac{\sigma_i^2}{n_i} + \rho, \quad i = 1, \dots, k \quad \text{and} \quad \frac{\partial}{\partial \rho} L(\lambda_1, \dots, \lambda_k, \rho) = \sum_{i=1}^k \lambda_i - 1.$$

Equating the partial derivatives to zero, we get $\lambda_i^0 = -\frac{\rho}{2} \frac{n_i}{\sigma_i^2}$ for $i = 1, \dots, k$ and $\sum_{i=1}^k \lambda_i^0 = -\frac{\rho}{2} \sum_{i=1}^k \frac{n_i}{\sigma_i^2} = 1$.

Thus, $-\frac{\rho}{2} = \frac{1}{\sum_{i=1}^k n_i / \sigma_i^2}$, $\lambda_i^0 = \frac{n_i / \sigma_i^2}{\sum_{j=1}^k n_j / \sigma_j^2}$, and therefore $w_i = n_i / \sigma_i^2$.

Solution 3.7 Since the Y_i are uncorrelated,

$$V\{\hat{\beta}_1\} = \sum_{i=1}^n w_i^2 V\{Y_i\} = \sigma^2 \sum_{i=1}^n \frac{(x_i - \bar{x}_n)^2}{SS_x^2} = \frac{\sigma^2}{SS_x}, \text{ where } SS_x = \sum_{i=1}^n (x_i - \bar{x}_n)^2.$$

Solution 3.8 Let $w_i = \frac{x_i - \bar{x}_n}{SS_x}$ for $i = 1, \dots, n$ where $SS_x = \sum_{i=1}^n (x_i - \bar{x}_n)^2$. Then we have

$$\sum_{i=1}^n w_i = 0, \quad \text{and} \quad \sum_{i=1}^n w_i^2 = \frac{1}{SS_x}.$$

Hence,

$$\begin{aligned} V\{\hat{\beta}_0\} &= V\{\bar{Y}_n - \hat{\beta}_1 \bar{x}_n\} \\ &= V\left\{\bar{Y}_n - \left(\sum_{i=1}^n w_i Y_i\right) \bar{x}_n\right\} \\ &= V\left\{\sum_{i=1}^n \left(\frac{1}{n} - w_i \bar{x}_n\right) Y_i\right\} \\ &= \sum_{i=1}^n \left(\frac{1}{n} - w_i \bar{x}_n\right)^2 \sigma^2 \\ &= \sigma^2 \sum_{i=1}^n \left(\frac{1}{n^2} - \frac{2w_i \bar{x}_n}{n} + w_i^2 \bar{x}_n^2\right) \\ &= \sigma^2 \left(\frac{1}{n} - \frac{2}{n} \bar{x}_n \sum_{i=1}^n w_i + \bar{x}_n^2 \sum_{i=1}^n w_i^2\right) \\ &= \sigma^2 \left(\frac{1}{n} + \frac{\bar{x}_n^2}{SS_x}\right). \end{aligned}$$

Also

$$\begin{aligned} \text{cov}(\hat{\beta}_0, \hat{\beta}_1) &= \text{cov}\left(\sum_{i=1}^n \left(\frac{1}{n} - w_i \bar{x}_n\right) Y_i, \sum_{i=1}^n w_i Y_i\right) \\ &= \sigma^2 \sum_{i=1}^n \left(\frac{1}{n} - w_i \bar{x}_n\right) w_i \\ &= -\sigma^2 \frac{\bar{x}_n}{SS_x}. \end{aligned}$$

Solution 3.9 The correlation between $\hat{\beta}_0$ and $\hat{\beta}_1$ is

$$\begin{aligned}\rho_{\hat{\beta}_0, \hat{\beta}_1} &= -\frac{\sigma^2 \bar{x}_n}{\sigma^2 SS_x \left[\left(\frac{1}{n} + \frac{\bar{x}_n^2}{SS_x} \right) \frac{1}{SS_x} \right]^{1/2}} \\ &= -\frac{\bar{x}_n}{\left(\frac{1}{n} \sum_{i=1}^n x_i^2 \right)^{1/2}}.\end{aligned}$$

Solution 3.10 X_1, X_2, \dots, X_n i.i.d., $X_1 \sim P(\lambda)$. The likelihood function is

$$L(\lambda; X_1, \dots, X_n) = e^{-n\lambda} \frac{\lambda^{\sum_{i=1}^n X_i}}{\prod_{i=1}^n X_i!}$$

Thus, $\frac{\partial}{\partial \lambda} \log L(\lambda; X_1, \dots, X_n) = -n + \frac{\sum_{i=1}^n X_i}{\lambda}$. Equating this to zero and solving for λ , we get $\hat{\lambda}_n = \frac{1}{n} \sum_{i=1}^n X_i = \bar{X}_n$.

Solution 3.11 Since ν is known, the likelihood of β is $L(\beta) = C_n \frac{1}{\beta^{n\nu}} e^{-\sum_{i=1}^n X_i/\beta}$, $0 < \beta < \infty$ where C_n does not depend on β . The log-likelihood function is

$$l(\beta) = \log C_n - n\nu \log \beta - \frac{1}{\beta} \sum_{i=1}^n X_i.$$

The score function is $l'(\beta) = -\frac{n\nu}{\beta} + \frac{\sum_{i=1}^n X_i}{\beta^2}$. Equating the score to 0 and solving for β , we obtain the MLE $\hat{\beta} = \frac{1}{n\nu} \sum_{i=1}^n X_i = \frac{1}{\nu} \bar{X}_n$. The variance of the MLE is $V\{\hat{\beta}\} = \frac{\beta^2}{n\nu}$.

{exc:mrgf-nb}

Solution 3.12 We proved in Exercise 2.56 that the m.g.f. of $NB(2, p)$ is

$$M_X(t) = \frac{p^2}{(1 - (1-p)e^t)^2}, \quad t < -\log(1-p).$$

Let X_1, X_2, \dots, X_n be i.i.d. $NB(2, p)$, then the m.g.f. of $T_n = \sum_{i=1}^n X_i$ is

$$M_{T_n}(t) = \frac{p^{2n}}{(1 - (1-p)e^t)^{2n}}, \quad t < -\log(1-p).$$

{exc:neg-binom-dist}

Thus, $T_n \sim NB(2n, p)$. According to Example 3.4, the MLE of p , based on T_n (which is a sufficient statistic) is $\hat{p}_n = \frac{2n}{T_n + 2n} = \frac{2}{\bar{X}_n + 2}$, where $\bar{X}_n = T_n/n$ is the sample mean.

(i) According to the WLLN, \bar{X}_n converges in probability to $E\{X_1\} = \frac{2(1-p)}{p}$.

Substituting $2\frac{1-p}{p}$ for \bar{X}_n in the formula of \hat{p}_n we obtain $p^* = \frac{2}{2 + 2\frac{1-p}{p}} = p$. This

shows that the limit in probability as $n \rightarrow \infty$, of \hat{p}_n is p .

(ii) Substituting $k = 2n$ in the formulas of Example 3.4 we obtain

{ex:neg-binom-dist}

$$\text{Bias}(\hat{p}_n) \approx \frac{3p(1-p)}{4n} \quad \text{and} \quad V\{\hat{p}_n\} \approx \frac{p^2(1-p)}{2n}.$$

Solution 3.13 The likelihood function of μ and β is

$$L(\mu, \beta) = I\{X_{(1)} \geq \mu\} \frac{1}{\beta^n} \exp \left\{ -\frac{1}{\beta} \sum_{i=1}^n (X_{(i)} - X_{(1)}) - \frac{n}{\beta} (X_{(1)} - \mu) \right\},$$

for $-\infty < \mu \leq X_{(1)}$, $0 < \beta < \infty$.

(i) $L(\mu, \beta)$ is maximized by $\hat{\mu} = X_{(1)}$, that is,

$$L^*(\beta) = \sup_{\mu \leq X_{(1)}} L(\mu, \beta) = \frac{1}{\beta^n} \exp \left\{ -\frac{1}{\beta} \sum_{i=1}^n (X_{(i)} - X_{(1)}) \right\}$$

where $X_{(1)} < X_{(2)} < \dots < X_{(n)}$ are the ordered statistics.

(ii) Furthermore $L^*(\beta)$ is maximized by $\hat{\beta}_n = \frac{1}{n} \sum_{i=2}^n (X_{(i)} - X_{(1)})$. The MLEs are $\hat{\mu} = X_{(1)}$, and $\hat{\beta}_n = \frac{1}{n} \sum_{i=2}^n (X_{(i)} - X_{(1)})$.

(iii) $X_{(1)}$ is distributed like $\mu + E\left(\frac{\beta}{n}\right)$, with p.d.f.

$$f_{(1)}(x; \mu, \beta) = I\{x \geq \mu\} \frac{n}{\beta} e^{-\frac{n}{\beta}(x-\mu)}.$$

Thus, the joint p.d.f. of (X_1, \dots, X_n) is factored to a product of the p.d.f. of $X_{(1)}$ and a function of $\hat{\beta}_n$, which does not depend on $X_{(1)}$ (nor on μ). This implies that $X_{(1)}$ and

$\hat{\beta}_n$ are independent. $V\{\hat{\mu}\} = V\{X_{(1)}\} = \frac{\beta^2}{n^2}$. It can be shown that $\hat{\beta}_n \sim \frac{1}{n}G(n-1, \beta)$.

Accordingly, $V\{\hat{\beta}_n\} = \frac{n-1}{n^2}\beta^2 = \frac{1}{n}\left(1 - \frac{1}{n}\right)\beta^2$.

Solution 3.14 In sampling with replacement, the number of defective items in the sample, X , has the binomial distribution $B(n, p)$. We test the hypotheses $H_0 : p \leq 0.03$ against $H_1 : p > 0.03$. H_0 is rejected if $X > B^{-1}(1-\alpha, 20, 0.03)$. For $\alpha = 0.05$ the rejection criterion is $k_\alpha = B^{-1}(0.95, 20, 0.03) = 2$. Since the number of defective items in the sample is $X = 2$, H_0 is not rejected at the $\alpha = 0.05$ significance level.

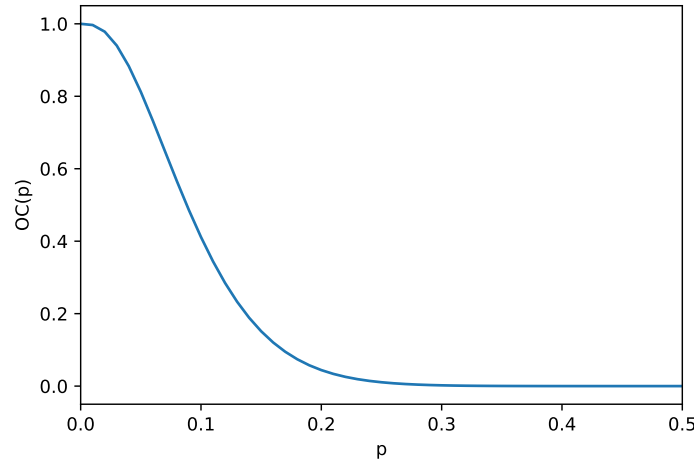


Fig. 3.1 The OC Function $B(2; 30, p)$

{fig:PlotOCcurveBinomial_2_30}

Solution 3.15 The OC function is $OC(p) = B(2; 30, p)$, $0 < p < 1$. A plot of this OC function is given in Fig. 3.1.

{fig:PlotOCcurveBinomial_2_30}

Solution 3.16 Let $p_0 = 0.01$, $p_1 = 0.03$, $\alpha = 0.05$, $\beta = 0.05$. According to Eq. (3.3.12), the sample size n should satisfy

$$1 - \Phi\left(\frac{p_1 - p_0}{\sqrt{p_1 q_1}} \sqrt{n} - z_{1-\alpha} \sqrt{\frac{p_0 q_0}{p_1 q_1}}\right) = \beta$$

or, equivalently,

$$\frac{p_1 - p_0}{\sqrt{p_1 q_1}} \sqrt{n} - z_{1-\alpha} \sqrt{\frac{p_0 q_0}{p_1 q_1}} = z_{1-\beta}.$$

This gives

$$\begin{aligned} n &\approx \frac{(z_{1-\alpha} \sqrt{p_0 q_0} + z_{1-\beta} \sqrt{p_1 q_1})^2}{(p_1 - p_0)^2} \\ &= \frac{(1.645)^2 (\sqrt{0.01 \times 0.99} + \sqrt{0.03 \times 0.97})^2}{(0.02)^2} = 494. \end{aligned}$$

For this sample size, the critical value is $k_\alpha = np_0 + z_{1-\alpha} \sqrt{np_0 q_0} = 8.58$. Thus, H_0 is rejected if there are more than 8 “successes” in a sample of size 494.

Solution 3.17 $\bar{X}_n \sim N(\mu, \frac{\sigma}{\sqrt{n}})$.

(i) If $\mu = \mu_0$ the probability that \bar{X}_n will be outside the control limits is

$$\Pr\left\{\bar{X}_n < \mu_0 - \frac{3\sigma}{\sqrt{n}}\right\} + \Pr\left\{\bar{X}_n > \mu_0 + \frac{3\sigma}{\sqrt{n}}\right\} = \Phi(-3) + 1 - \Phi(3) = 0.0027.$$

(ii) $(1 - 0.0027)^{20} = 0.9474$.

(iii) If $\mu = \mu_0 + 2(\sigma/\sqrt{n})$, the probability that \bar{X}_n will be outside the control limits is

$$\Phi(-5) + 1 - \Phi(1) = 0.1587.$$

(iv) $(1 - 0.1587)^{10} = 0.1777$.

Solution 3.18 We can run the 1-sample t -test in Python as follows:

```
socell = mistat.load_data('SOCELL')
t1 = socell['t1']

statistic, pvalue = stats.ttest_1samp(t1, 4.0)
# divide pvalue by two for one-sided test
pvalue = pvalue / 2
print(f'pvalue {pvalue:.2f}')
```

| pvalue 0.35

The hypothesis $H_0 : \mu \geq 4.0$ amps is not rejected.

Solution 3.19 We can run the 1-sample t -test in Python as follows:

```
socell = mistat.load_data('SOCELL')
t2 = socell['t2']

statistic, pvalue = stats.ttest_1samp(t2, 4.0)
# divide pvalue by two for one-sided test
pvalue = pvalue / 2
print(f'pvalue {pvalue:.2f}')
```

| pvalue 0.03

The hypothesis $H_0 : \mu \geq 4.0$ amps is rejected at a 0.05 level of significance.

Solution 3.20 Let $n = 30$, $\alpha = 0.01$. The $OC(\delta)$ function for a one-sided t -test is

$$\begin{aligned} OC(\delta) &= 1 - \Phi\left(\frac{\delta\sqrt{30} - 2.462 \times (1 - \frac{1}{232})}{(1 + \frac{6.0614}{58})^{1/2}}\right) \\ &= 1 - \Phi(5.2117\delta - 2.3325). \end{aligned}$$

```
delta = np.linspace(0, 1.0, 11)

a = np.sqrt(30)
b = 2.462 * (1 - 1/232)
f = np.sqrt(1 + 6.0614 / 58)
OC_delta = 1 - stats.norm.cdf((a * delta - b) / f)
```

Values of $OC(\delta)$ for $\delta = 0, 1(0.1)$ are given in the following table.

δ	$OC(\delta)$
0.0	0.990164
0.1	0.964958
0.2	0.901509
0.3	0.779063
0.4	0.597882
0.5	0.392312
0.6	0.213462
0.7	0.094149
0.8	0.033120
0.9	0.009188
1.0	0.001994

Solution 3.21 Let $n = 31$, $\alpha = 0.10$. The OC function for testing $H_0 : \sigma^2 \leq \sigma_0^2$ against $H_1 : \sigma^2 > \sigma_0^2$ is

$$\begin{aligned}
 OC(\sigma^2) &= \Pr \left\{ S^2 \leq \frac{\sigma_0^2}{n-1} \chi_{0.9}^2[n-1] \right\} \\
 &= \Pr \left\{ \chi^2[30] \leq \frac{\sigma_0^2}{\sigma^2} \chi_{0.9}^2[30] \right\} \\
 &= 1 - P \left(\frac{30}{2} - 1; \frac{\sigma_0^2}{\sigma^2} \cdot \frac{\chi_{0.9}^2[30]}{2} \right).
 \end{aligned}$$

```
sigma2 = np.linspace(1, 2, 11)
OC_sigma2 = 1 - stats.poisson.cdf(30 / 2 - 1,
stats.chi2(30).ppf(0.90) / (2 * sigma2))
```

The values of $OC(\sigma^2)$ for $\sigma^2 = 1, 2(0.1)$ are given in the following table: (Here $\sigma_0^2 = 1$.)

σ^2	$OC(\sigma^2)$
1.0	0.900000
1.1	0.810804
1.2	0.700684
1.3	0.582928
1.4	0.469471
1.5	0.368201
1.6	0.282781
1.7	0.213695
1.8	0.159540
1.9	0.118063
2.0	0.086834

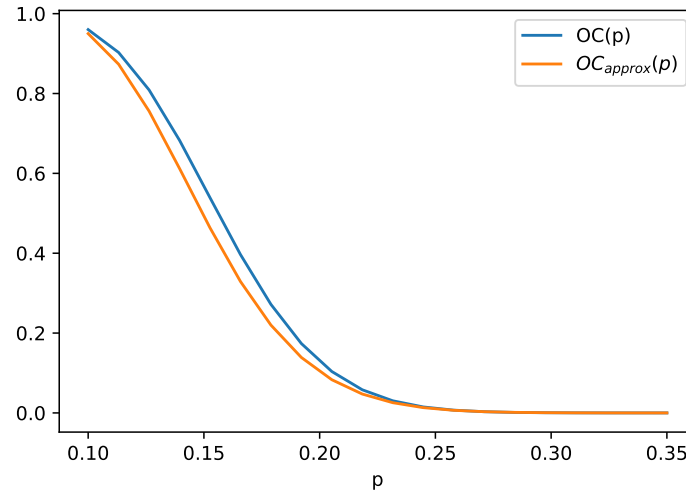


Fig. 3.2 Comparison exact to normal approximation of $OC(p)$

{fig:OC_OCapprox}

Solution 3.22 The OC function, for testing $H_0 : p \leq p_0$ against $H_1 : p > p_0$ is approximated by

$$OC(p) = 1 - \Phi\left(\frac{p - p_0}{\sqrt{pq}}\sqrt{n} - z_{1-\alpha}\sqrt{\frac{p_0q_0}{pq}}\right),$$

for $p \geq p_0$. In Fig. 3.2, we present the graph of $OC(p)$, for $p_0 = 0.1$, $n = 100$, $\alpha = 0.05$ both using the exact solution and the normal approximation.

{fig:OC_OCapprox}

```

n = 100
p0 = 0.1
alpha = 0.05

c_alpha = stats.binom(n, p0).ppf(1 - alpha)
p = np.linspace(0.1, 0.35, 20)
OC_exact = stats.binom(n, p).cdf(c_alpha)

z_ma = stats.norm.ppf(1 - alpha)
q0 = 1 - p0
pq = p * (1 - p)
OC_approx = 1 - stats.norm.cdf((p - p0) * np.sqrt((n / pq)) -
                                z_ma * np.sqrt(p0 * q0 / pq))

df = pd.DataFrame({
    'p': p,
    'OC(p)': OC_exact,
    '$OC_{approx}(p)$': OC_approx,
})
df.plot.line(x='p', y=['OC(p)', '$OC_{approx}(p)$'])
plt.show()

```

Solution 3.23 The power function is

$$\begin{aligned}\psi(\sigma^2) &= \Pr \left\{ S^2 \geq \frac{\sigma_0^2}{n-1} \chi_{1-\alpha}^2[n-1] \right\} \\ &= \Pr \left\{ \chi^2[n-1] \geq \frac{\sigma_0^2}{\sigma^2} \chi_{1-\alpha}^2[n-1] \right\}.\end{aligned}$$

Solution 3.24 The power function is $\psi(\rho) = \Pr \left\{ F[n_1 - 1, n_2 - 1] \geq \frac{1}{\rho} F_{1-\alpha}[n_1 - 1, n_2 - 1] \right\}$, for $\rho \geq 1$, where $\rho = \frac{\sigma_1^2}{\sigma_2^2}$.

Solution 3.25 (i) Using the following Python commands we get a 99% C.I. for μ :

```
data = [20.74, 20.85, 20.54, 20.05, 20.08, 22.55, 19.61, 19.72,
        20.34, 20.37, 22.69, 20.79, 21.76, 21.94, 20.31, 21.38,
        20.42, 20.86, 18.80, 21.41]
alpha = 1 - 0.99

df = len(data) - 1
mean = np.mean(data)
sem = stats.sem(data)

print(stats.t.interval(1 - alpha, df, loc=mean, scale=sem))
```

```
| (20.136889216656858, 21.38411078334315)
```

Confidence Intervals

Variable	N	Mean	StDev	SE	99.0% C.I.
				Mean	
Sample	20	20.760	0.975	0.218	(20.137, 21.384)

(ii) A 99% C.I. for σ^2 is (0.468, 2.638).

```
var = np.var(data, ddof=1)
print(df * var / stats.chi2(df).ppf(1 - alpha/2))
print(df * var / stats.chi2(df).ppf(alpha/2))
```

```
| 0.46795850248657883
| 2.6380728125212016
```

(iii) A 99% C.I. for σ is (0.684, 1.624).

Solution 3.26 Let $(\underline{\mu}_{.99}, \bar{\mu}_{.99})$ be a confidence interval for μ , at level 0.99. Let $(\underline{\sigma}_{.99}, \bar{\sigma}_{.99})$ be a confidence interval for σ at level 0.99. Let $\underline{\xi} = \underline{\mu}_{.99} + 2\underline{\sigma}_{.99}$ and $\bar{\xi} = \bar{\mu}_{.99} + 2\bar{\sigma}_{.99}$. Then

$$\Pr\{\underline{\xi} \leq \mu + 2\sigma \leq \bar{\xi}\} \geq \Pr\{\underline{\mu}_{.99} \leq \mu \leq \bar{\mu}_{.99}, \underline{\sigma}_{.99} \leq \sigma \leq \bar{\sigma}_{.99}\} \geq 0.98.$$

Thus, $(\underline{\xi}, \bar{\xi})$ is a confidence interval for $\mu + 2\sigma$, with confidence level greater or equal to 0.98. Using the data of the previous problem, a 98% C.I. for $\mu + 2\sigma$ is (21.505, 24.632).

Solution 3.27 Let $X \sim B(n, \theta)$. For $X = 17$ and $n = 20$, a confidence interval for θ , at level 0.95, is (0.6211, 0.9679).

```
alpha = 1 - 0.95
X = 17
n = 20
F1 = stats.f(2*(n-X+1), 2*X).ppf(1 - alpha/2)
F2 = stats.f(2*(X+1), 2*(n-X)).ppf(1 - alpha/2)
pL = X / (X + (n-X+1) * F1)
pU = (X+1) * F2 / (n-X + (X+1) * F2)
print(pL, pU)
```

```
| 0.6210731734546859 0.9679290628145363
```

Solution 3.28 From the data we have $n = 10$ and $T_{10} = 134$. For $\alpha = 0.05$, $\lambda_L = 11.319$ and $\lambda_U = 15.871$.

```
X = [14, 16, 11, 19, 11, 9, 12, 15, 14, 13]
alpha = 1 - 0.95
T_n = np.sum(X)

# exact solution
print(stats.chi2(2 * T_n + 2).ppf(alpha/2) / (2 * len(X)))
print(stats.chi2(2 * T_n + 2).ppf(1 - alpha/2) / (2 * len(X)))

# approximate solution
nu = 2 * T_n + 2
print((nu + stats.norm.ppf(alpha/2) * np.sqrt(2 * nu)) / (2 * len(X)))
print((nu + stats.norm.ppf(1-alpha/2) * np.sqrt(2 * nu)) / (2 * len(X)))
```

```
| 11.318870163746238
| 15.870459268116013
| 11.222727638613012
| 15.777272361386988
```

Solution 3.29 For $n = 20$, $\sigma = 5$, $\bar{Y}_{20} = 13.75$, $\alpha = 0.05$ and $\beta = 0.1$, the tolerance interval is (3.33, 24.17).

Solution 3.30 Use the following Python code:

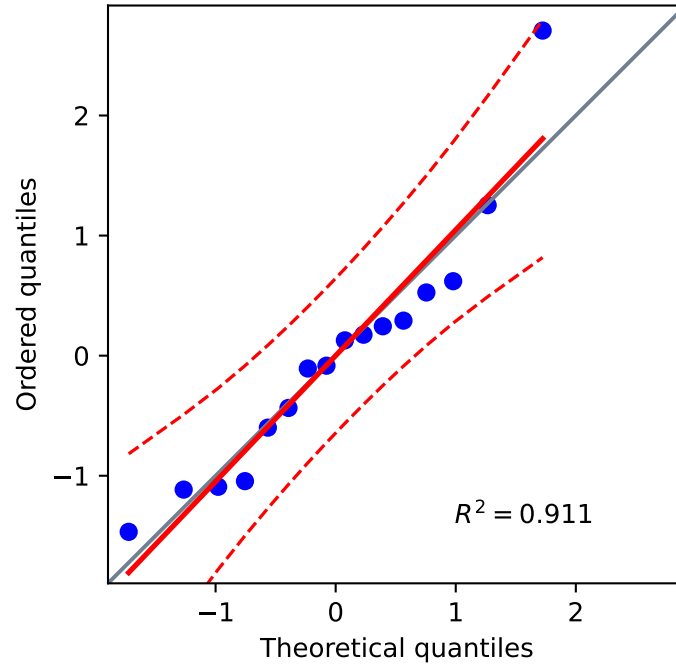
```
yarnstrg = mistat.load_data('YARNSTRG')
n = len(yarnstrg)
Ybar = yarnstrg.mean()
S = yarnstrg.std()

alpha, beta = 0.025, 0.025
z_la = stats.norm.ppf(1-alpha)
z_lb = stats.norm.ppf(1-beta)
z_a = stats.norm.ppf(alpha)
z_b = stats.norm.ppf(beta)

t_abn = (z_lb/(1 - z_la**2/(2*n)) +
          z_la*np.sqrt(1 + z_b**2/2 - z_la**2/(2*n))) /
          (np.sqrt(n)*(1 - z_la**2/(2*n))))

print(Ybar - t_abn*S, Ybar + t_abn*S)
```

```
| 0.7306652047594424 5.117020795240556
```



{fig:qqplotISCT1Socell} **Fig. 3.3** Q - Q plot of ISC- t_1 (SOCELL.csv)

From the data we have $\bar{X}_{100} = 2.9238$ and $S_{100} = 0.9378$.

$$\begin{aligned}
 t(0.025, 0.025, 100) &= \frac{1.96}{1 - 1.96^2/200} + \frac{1.96(1 + \frac{1.96^2}{2} - \frac{1.96^2}{200})^{1/2}}{10(1 - \frac{1.96^2}{200})} \\
 &= 2.3388.
 \end{aligned}$$

The tolerance interval is (0.7306, 5.1171).

Solution 3.31 From the data, $Y_{(1)} = 1.151$ and $Y_{(100)} = 5.790$. For $n = 100$ and $\beta = 0.10$ we have $1 - \alpha = 0.988$. For $\beta = 0.05$, $1 - \alpha = 0.847$, the tolerance interval is (1.151, 5.790). The nonparametric tolerance interval is shorter and is shifted to the right with a lower confidence level.

Solution 3.32 The following is a normal probability plot of ISC- t_1 :

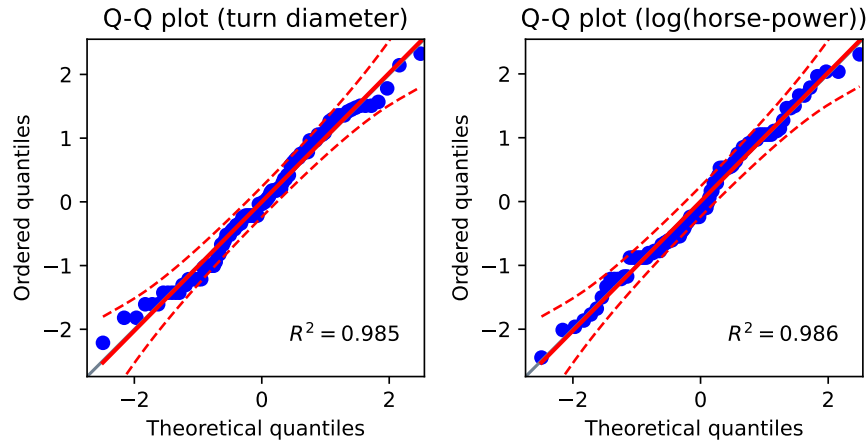
According to Fig. 3.3, the hypothesis of normality is not rejected.

{fig:qqplotISCT1Socell}

{fig:qqplotCar}

Solution 3.33 As is shown in the normal probability plots in Fig. 3.4, the hypothesis of normality is not rejected in either case.

Solution 3.34 Frequency distribution for turn diameter:



`{fig:qqplotCar}` **Fig. 3.4** *Q-Q* plot of CAR.csv data

Interval	Observed	Expected	$(O - E)^2/E$
- 31	11	8.1972	0.9583
31 - 32	8	6.3185	0.4475
32 - 33	9	8.6687	0.0127
33 - 34	6	10.8695	2.1815
34 - 35	18	12.4559	2.4677
35 - 36	8	13.0454	1.9513
36 - 37	13	12.4868	0.0211
37 - 38	6	10.9234	2.2191
38 - 39	9	8.7333	0.0081
39 - 40	8	6.3814	0.4106
40 -	13	9.0529	1.7213
Total	109	—	12.399

The expected frequencies were computed for $N(35.5138, 3.3208)$. Here $\chi^2 = 12.4$, d.f. = 8 and the P value is 0.134. The differences from normal are not significant.

```

turn      observed  expected  (O-E)^2/E
[28, 31)    11      8.197333  0.958231
[31, 32)     8      6.318478  0.447500
[32, 33)     9      8.668695  0.012662
[33, 34)     6     10.869453  2.181487
[34, 35)    18     12.455897  2.467673
[35, 36)     8     13.045357  1.951317
[36, 37)    13     12.486789  0.021093
[37, 38)     6     10.923435  2.219102
[38, 39)     9      8.733354  0.008141
[39, 40)     8      6.381395  0.410550
[40, 44)    13      9.052637  1.721231
chi2-statistic of fit 12.398987638400024

```

```
| chi2[8] for 95% 15.50731305586545
| p-value of observed statistic 0.1342700576126994
```

Solution 3.35 In Python:

```
| KstestResult(statistic=0.07019153486614366, pvalue=0.6303356787948367,
| statistic_location=32.2, statistic_sign=1)
| k_alpha 0.08514304524687971
```

$D_{109}^* = 0.0702$. For $\alpha = 0.05$ $k_{\alpha}^* = 0.895 / \left(\sqrt{109} - 0.01 + \frac{0.85}{\sqrt{109}} \right) = 0.0851$. The deviations from the normal distribution are not significant.

Solution 3.36 For $X \sim P(100)$, $n^0 = P^{-1} \left(\frac{0.2}{0.3}, 100 \right) = 100 + z_{.67} \times 10 = 105$.

Solution 3.37 Given $X = 6$, the posterior distribution of p is Beta(9,11).

Solution 3.38 $E\{p \mid X = 6\} = \frac{9}{20} = 0.45$ and $V\{p \mid X = 6\} = \frac{99}{20^2 \times 21} = 0.0118$ so $\sigma_{p|X=6} = 0.1086$.

Solution 3.39 Let $X \mid \lambda \sim P(\lambda)$ where $\lambda \sim G(2, 50)$.

(i) The posterior distribution of $\lambda \mid X = 82$ is $G \left(84, \frac{50}{51} \right)$.

(ii) $G_{.025} \left(84, \frac{50}{51} \right) = 65.6879$ and $G_{.975} \left(84, \frac{50}{51} \right) = 100.873$.

Solution 3.40 The posterior probability for $\lambda_0 = 70$ is

$$\pi(72) = \frac{\frac{1}{3}p(72; 70)}{\frac{1}{3}p(72; 70) + \frac{2}{3}p(72; 90)} = 0.771.$$

$H_0 : \lambda = \lambda_0$ is accepted if $\pi(X) > \frac{r_0}{r_0 + r_1} = 0.4$. Thus, H_0 is accepted.

Solution 3.41 The credibility interval for μ is (43.235, 60.765). Since the posterior distribution of μ is symmetric, this credibility interval is also a HPD interval.

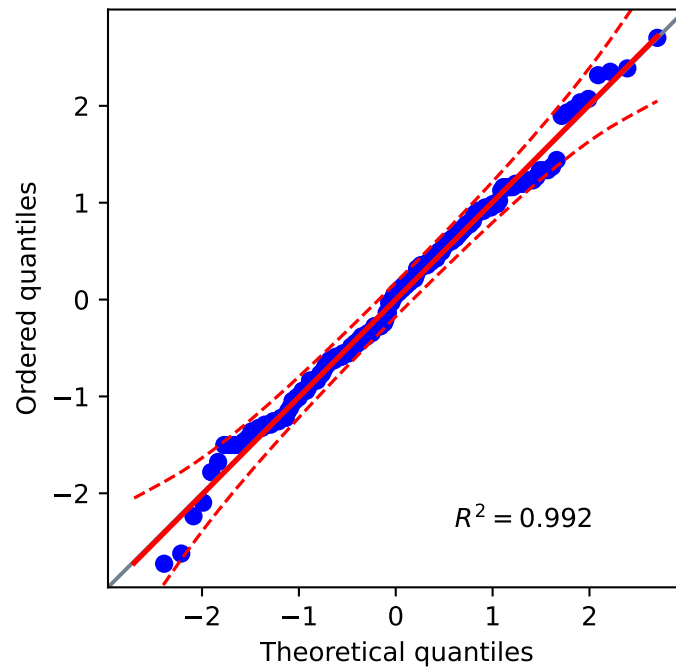
Solution 3.42 In Python:

```
random.seed(1)
car = mistat.load_data('CAR')

def mean_of_sample_with_replacement(x, n):
    sample = random.choices(x, k=n)
    return np.mean(sample)

means = [mean_of_sample_with_replacement(car['mpg'], 64)
          for _ in range(200)]

fig, ax = plt.subplots(figsize=[4, 4])
pg.qqplot(means, ax=ax)
plt.show()
```



`{fig:qqplotSampleMeansMPG}` **Fig. 3.5** Q - Q Plot of mean of samples from mpg data

`{fig:qqplotSampleMeansMPG}` As Fig. 3.5 shows, the resampling distribution is approximately normal.

```
S = np.std(car['mpg'], ddof=1)
print('standard deviation of mpg', S)
print('S/8', S/8)
S_resample = np.std(means, ddof=1)
print('S.E.\{X\}', S_resample)
```

```
| standard deviation of mpg 3.9172332424696052
| S/8 0.48965415530870066
| S.E.\{X\} 0.44718791755315535
```

Executing the macro 200 times, we obtained $S.E.\{\bar{X}\} = \frac{S}{8} = 0.48965$. The standard deviation of the resampled distribution is 0.4472. This is a resampling estimate of $S.E.\{\bar{X}\}$.

Solution 3.43 In our particular execution with $M = 500$, we have a proportion $\hat{\alpha} = 0.07$ of cases in which the bootstrap confidence intervals do not cover the mean of `yarnstrg`, $\mu = 2.9238$. This is not significantly different from the nominal $\alpha = 0.05$. The determination of the proportion $\hat{\alpha}$ can be done by using the following commands:

```
random.seed(1)
yarnstrg = mistat.load_data('YARNSTRG')
```

```

def confidence_interval(x, nsigma=2):
    sample_mean = np.mean(x)
    sigma = np.std(x, ddof=1) / np.sqrt(len(x))
    return (sample_mean - 2 * sigma, sample_mean + 2 * sigma)

mean = np.mean(yarnstrg)
outside = 0
for _ in range(500):
    sample = random.choices(yarnstrg, k=30)
    ci = confidence_interval(sample)
    if mean < ci[0] or ci[1] < mean:
        outside += 1

hat_alpha = outside / 500

ci = confidence_interval(yarnstrg)
print(f' Mean: {mean}')
print(f' 2-sigma-CI: {ci[0]:.1f} - {ci[1]:.1f}')
print(f' proportion outside: {hat_alpha:.3f}')

```

```

Mean: 2.9238429999999993
2-sigma-CI: 2.7 - 3.1
proportion outside: 0.068

```

Solution 3.44 In Python:

```

random.seed(1)
car = mistat.load_data('CAR')
us_cars = car[car['origin'] == 1]
us_turn = list(us_cars['turn'])

sample_means = []
for _ in range(100):
    x = random.choices(us_turn, k=58)
    sample_means.append(np.mean(x))

is_larger = sum(m > 37.406 for m in sample_means)
ratio = is_larger / len(sample_means)
print(ratio)

```

```

0.23

```

We obtained $\tilde{P} = 0.23$. The mean $\bar{X} = 37.203$ is **not** significantly larger than 37.

Solution 3.45 Let X_{50} be the number of non-conforming units in a sample of $n = 50$ items. We reject H_0 , at level of $\alpha = 0.05$, if $X_{50} > B^{-1}(0.95, 50, 0.03) = 4$. The criterion k_α is obtained by using the Python commands:

```

stats.binom(50, 0.03).ppf(0.95)

```

```

4.0

```

Solution 3.46 i Calculation of 95% confidence intervals:

```

random.seed(1)
cyclt = mistat.load_data('CYCLT')

```

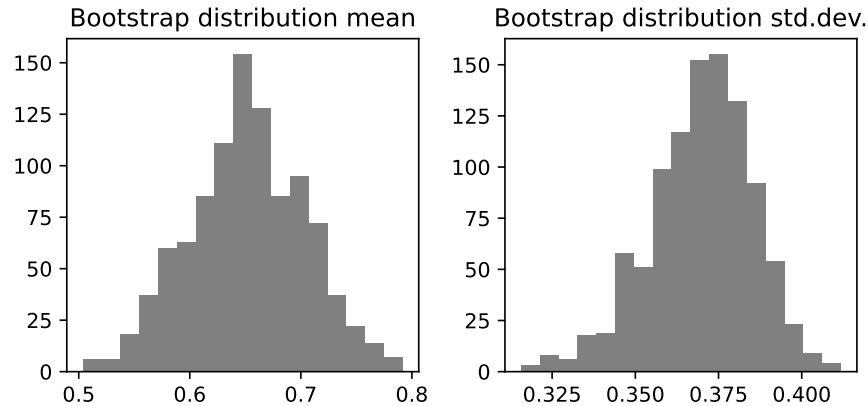


Fig. 3.6 Histograms of EBD for **CYCLT.csv** data

(fig:histEBD_CYCLT)

```
B = pg.compute_bootci(cyclt, func=np.mean, n_boot=1000,
                      confidence=0.95, return_dist=True, seed=1)

ci_mean, dist_mean = B
print(f' Mean: {np.mean(dist_mean):.3f}')
print(f' 95%-CI: {ci_mean[0]:.3f} - {ci_mean[1]:.3f}')

B = pg.compute_bootci(cyclt, func=lambda x: np.std(x, ddof=1), n_boot=1000,
                      confidence=0.95, return_dist=True, seed=1)

ci_std, dist_std = B
print(f' Mean: {np.mean(dist_std):.3f}')
print(f' 95%-CI: {ci_std[0]:.3f} - {ci_std[1]:.3f}')
```

```
Mean: 0.652
95%-CI: 0.550 - 0.760
Mean: 0.370
95%-CI: 0.340 - 0.400
```

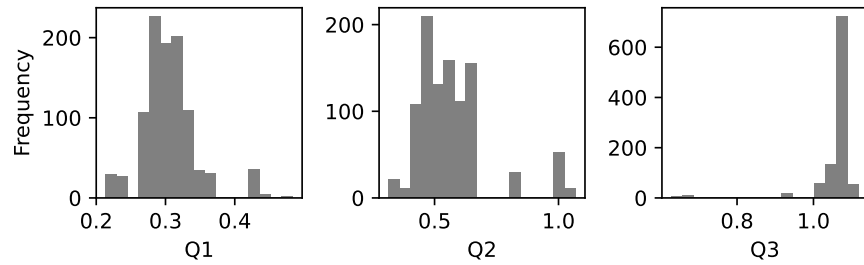
ii Histograms of the EBD, see Fig. 3.6.

(fig:histEBD_CYCLT)

```
fig, axes = plt.subplots(figsize=[6, 3], ncols=2)
axes[0].hist(dist_mean, color='grey', bins=17)
axes[1].hist(dist_std, color='grey', bins=17)
axes[0].set_title('Bootstrap distribution mean')
axes[1].set_title('Bootstrap distribution std.dev.')
plt.tight_layout()
plt.show()
```

Solution 3.47 In Python:

```
cyclt = mistat.load_data('CYCLT')
ebd = {}
for quantile in (1, 2, 3):
    B = pg.compute_bootci(cyclt, func=lambda x: np.quantile(x, 0.25 * quantile),
                          n_boot=1000, confidence=0.95, return_dist=True, seed=1)
```



{fig:histEBD_CYCLT_quartiles}

Fig. 3.7 Histograms of EBD for quartiles for **CYCLT.csv** data

```
ci, dist = B
ebd[quantile] = dist
print(f'Quantile {quantile}: {np.mean(dist):.3f} 95%-CI: {ci[0]:.3f} - {ci[1]:.3f}')
```

```
Quantile 1: 0.306 95%-CI: 0.230 - 0.420
Quantile 2: 0.573 95%-CI: 0.380 - 1.010
Quantile 3: 1.060 95%-CI: 0.660 - 1.090
```

{fig:histEBD_CYCLT_quartiles}

ii Histograms of the EBD, see Fig. 3.7.

```
fig, axes = plt.subplots(figsize=[6, 2], ncols=3)
axes[0].hist(ebd[1], color='grey', bins=17)
axes[1].hist(ebd[2], color='grey', bins=17)
axes[2].hist(ebd[3], color='grey', bins=17)
axes[0].set_xlabel('Q1')
axes[1].set_xlabel('Q2')
axes[2].set_xlabel('Q3')
axes[0].set_ylabel('Frequency')
plt.tight_layout()
plt.show()
```

Solution 3.48 In Python:

```
socell = mistat.load_data('SOCELL')
t1 = socell['t1']
t2 = socell['t2']

# use the index
idx = list(range(len(socell)))
def sample_correlation(x):
    return stats.pearsonr(t1[x], t2[x])[0]

B = pg.compute_bootci(idx, func=sample_correlation,
                      n_boot=1000, confidence=0.95, return_dist=True, seed=1)
ci, dist = B
print(f'rho_XY: {np.mean(dist):.3f} 95%-CI: {ci[0]:.3f} - {ci[1]:.3f}')
```

```
rho_XY: 0.975 95%-CI: 0.940 - 0.990
```

{fig:histEBD_SOCELL_correlation}

Histogram of bootstrap correlations, see Fig. 3.8.

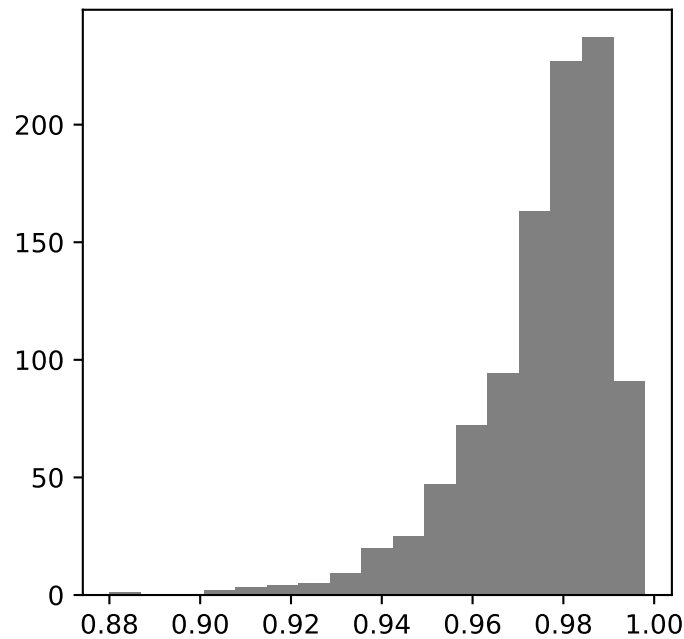


Fig. 3.8 Histograms of EBD for correlation for **SOCELL.csv** data

(fig:histEBD_SOCELL_correlation)

```
fig, ax = plt.subplots(figsize=[4, 4])
ax.hist(dist, color='grey', bins=17)
plt.show()
```

Solution 3.49 (i) and (ii)

```
car = mistat.load_data('CAR')
mpg = car['mpg']
hp = car['hp']

idx = list(range(len(mpg)))
sample_intercept = []
sample_slope = []
for _ in range(1000):
    x = random.choices(idx, k=len(idx))
    result = stats.linregress(hp[x], mpg[x])
    sample_intercept.append(result.intercept)
    sample_slope.append(result.slope)

ci = np.quantile(sample_intercept, [0.025, 0.975])
print(f'intercept (a): {np.mean(sample_intercept):.3f} ' +
      f'95%-CI: {ci[0]:.3f} - {ci[1]:.3f}')
ci = np.quantile(sample_slope, [0.025, 0.975])
print(f'slope (b): {np.mean(sample_slope):.4f} ' +
      f'95%-CI: {ci[0]:.4f} - {ci[1]:.4f}')

reg = stats.linregress(hp, mpg)
```

```

hm = np.mean(hp)

print(np.std(sample_intercept))
print(np.std(sample_slope))

```

```

intercept (a): 30.724 95%-CI: 28.766 - 32.691
slope (b): -0.0741 95%-CI: -0.0891 - -0.0599
1.01704449375724464
0.0074732552885114645

```

(sec:least-squares-single)

(iii) The bootstrap S.E. of *slope* and *intercept* are 1.017 and 0.00747, respectively. The standard errors of *a* and *b*, according to the formulas of Sect. 4.3.2.1 are 0.8099 and 0.00619, respectively. The bootstrap estimates are quite close to the correct values.

Solution 3.50 In Python:

```

cyclt = mistat.load_data('CYCLT')
X50 = np.mean(cyclt)
SD50 = np.std(cyclt)
result = stats.ttest_1samp(cyclt, 0.55)
print(f'Xmean_50 = {X50:.3f}')
print(result)

B = pg.compute_bootci(cyclt, func=np.mean, n_boot=1000,
                      confidence=0.95, return_dist=True, seed=1)

ci_mean, dist = B
pstar = sum(dist < 0.55) / len(dist)
print(f'p*-value: {pstar}')

```

```

Xmean_50 = 0.652
TtestResult(statistic=1.9425149510299369, pvalue=0.057833259176805,
df=49)
p*-value: 0.024

```

The mean of the sample is $\bar{X}_{50} = 0.652$. The studentized difference from $\mu_0 = 0.55$ is $t = 1.943$.

(i) The t-test obtained a *P*-level of 0.058 and the bootstrap resulted in $P^* = 0.024$.

(ii) Yes, but μ is very close to the lower bootstrap confidence limit (0.540). The null hypothesis $H_0 : \mu = 0.55$ is accepted.

(iii) No, but since P^* is close to 0.05, we expect that the bootstrap confidence interval will be very close to μ_0 .

Solution 3.51 In Python:

```

almpin = mistat.load_data('ALMPIN')
diam1 = almpin['diam1']
diam2 = almpin['diam2']

# calculate the ratio of the two variances:
var_diam1 = np.var(diam1)
var_diam2 = np.var(diam2)
F = var_diam2 / var_diam1
print(f'Variance diam1: {var_diam1:.5f}')
print(f'Variance diam2: {var_diam2:.5f}')
print(f'Ratio: {F:.4f}')

# Calculate the p-value

```

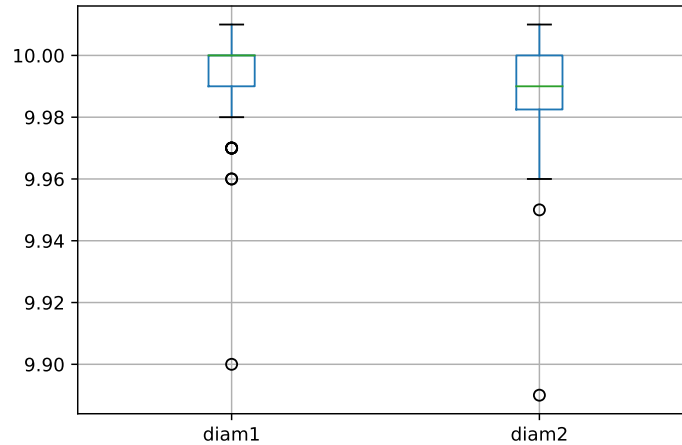


Fig. 3.9 Box plots of diam1 and diam2 measurements of the ALMPIN dataset

{fig:boxplot-alm-pin}

```
p_value = stats.f.cdf(F, len(diam1) - 1, len(diam2) - 1)
print(f'p-value: {p_value:.3f}')
```

```
Variance diam1: 0.00027
Variance diam2: 0.00032
Ratio: 1.2016
p-value: 0.776
```

The variances are therefore not significantly different.

(ii) The box plots of the two measurements are shown in Fig. 3.9.

{fig:boxplot-alm-pin}

```
almpin.boxplot(column=['diam1', 'diam2'])
plt.show()
```

Solution 3.52 The variances were already compared in the previous exercise. To compare the means use:

```
almpin = mistat.load_data('ALMPIN')
diam1 = almpin['diam1']
diam2 = almpin['diam2']

# Compare means
mean_diam1 = np.mean(diam1)
mean_diam2 = np.mean(diam2)
print(f'Mean diam1: {mean_diam1:.5f}')
print(f'Mean diam2: {mean_diam2:.5f}')

# calculate studentized difference and p-value
se1, se2 = stats.sem(diam1), stats.sem(diam2)
sed = np.sqrt(se1**2.0 + se2**2.0)
t_stat = (mean_diam1 - mean_diam2) / sed
print(f'Studentized difference: {t_stat:.3f}')
df = len(diam1) + len(diam2) - 2
p = (1 - stats.t.cdf(abs(t_stat), df)) * 2
```

```
print(f'p-value: {p:.3f}')

# or use any of the available implementations of the t-test
print(stats.ttest_ind(diam1, diam2))
```

```
| Mean diam1: 9.99286
| Mean diam2: 9.98729
| Studentized difference: 1.912
| p-value: 0.058
| Ttest_indResult(statistic=1.9119658005133064,
| pvalue=0.05795318184124417)
```

The bootstrap based p-value for the comparison of the means is:

```
random.seed(1)

# return studentized distance between random samples from diam1 and diam2
def stat_func():
    d1 = random.choices(diam1, k=len(diam1))
    d2 = random.choices(diam2, k=len(diam2))
    return stats.ttest_ind(d1, d2).statistic

dist = np.array([stat_func() for _ in range(1000)])

pstar = sum(dist < 0) / len(dist)
print(f'p*-value: {pstar}')
```

```
| p*-value: 0.014
```

The bootstrap based p-value for the comparison of the variances is:

```
columns = ['diam1', 'diam2']
# variance for each column
S2 = almpin[columns].var(axis=0, ddof=0)
F0 = max(S2) / min(S2)
print('S2', S2)
print('F0', F0)

# Step 1: sample variances of bootstrapped samples for each column
seed = 1
B = {}
for column in columns:
    ci = pg.compute_bootci(almpin[column], func='var', n_boot=500,
                           confidence=0.95, seed=seed, return_dist=True)
    B[column] = ci[1]
Bt = pd.DataFrame(B)

# Step 2: compute Wi
Wi = Bt / S2

# Step 3: compute F*
FBoot = Wi.max(axis=1) / Wi.min(axis=1)
FBoot95 = np.quantile(FBoot, 0.95)
print('FBoot 95%', FBoot95)
pstar = sum(FBoot >= F0) / len(FBoot)
print(f'p*-value: {pstar}')
```

```
| S2 diam1    0.000266
| diam2    0.000320
| dtype: float64
| F0 1.2016104294478573
| FBoot 95% 1.1855457165968324
| p*-value: 0.04
```

The variance of Sample 1 is $S_1^2 = 0.00027$. The variance of Sample 2 is $S_2^2 = 0.00032$. The variance ratio is $F = S_2^2/S_1^2 = 1.202$. The bootstrap level for variance ratios is $P^* = 0.04$.

Solution 3.53 In Python:

```
mpg = mistat.load_data('MPG')
columns = ['origin1', 'origin2', 'origin3']
# variance for each column
S2 = mpg[columns].var(axis=0, ddof=1)
F0 = max(S2) / min(S2)
print('S2', S2)
print('F0', F0)

# Step 1: sample variances of bootstrapped samples for each column
seed = 1
B = {}
for column in columns:
    ci = pg.compute_bootci(mpg[column].dropna(), func='var', n_boot=500,
                           confidence=0.95, seed=seed, return_dist=True)
    B[column] = ci[1]
Bt = pd.DataFrame(B)

# Step 2: compute Wi
Wi = Bt / S2

# Step 3: compute F*
FBoot = Wi.max(axis=1) / Wi.min(axis=1)
FBoot95 = np.quantile(FBoot, 0.95)
print('FBoot 95%', FBoot95)
pstar = sum(FBoot >= F0) / len(FBoot)
print(f'p*-value: {pstar}')
```

```
S2 origin1    12.942529
   origin2     6.884615
   origin3    18.321321
   dtype: float64
F0 2.6611975103595213
FBoot 95% 2.6925366761838987
p*-value: 0.058
```

With $M = 500$ we obtained the following results:

- 1st sample variance = 12.9425,
- 2nd sample variance = 6.8846,
- 3rd sample variance = 18.3213,

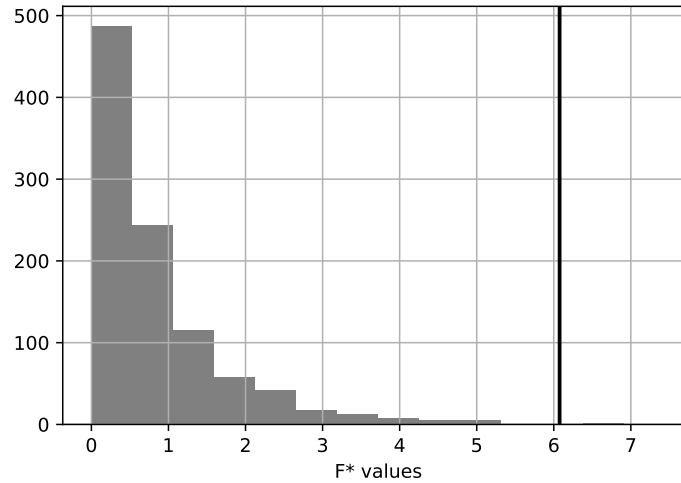
$F_{\max/\min} = 2.6612$ and the bootstrap P value is $P^* = 0.058$. The bootstrap test does not reject the hypothesis of equal variances at the 0.05 significance level.

Solution 3.54 With $M = 500$ we obtained

$$\begin{array}{ll} \bar{X}_1 = 20.931 & S_1^2 = 12.9425 \\ \bar{X}_2 = 19.5 & S_2^2 = 6.8846 \\ \bar{X}_3 = 23.1081 & S_3^2 = 18.3213 \end{array}$$

Using the approach shown in Sect. 3.11.5.2, we get:

[sec:comp-means-one-way-anova]



(fig.mpg-equal-mean) **Fig. 3.10** Distribution of EBD for Exercise *ex:mpg-equal-mean* 3.54

```
mpg = mistat.load_data('MPG.csv')
samples = [mpg[key].dropna() for key in ['origin1', 'origin2', 'origin3']]

def test_statistic_F(samples):
    return stats.f_oneway(*samples).statistic

# Calculate sample shifts
Ni = np.array([len(sample) for sample in samples])
N = np.sum(Ni)
XBni = np.array([np.mean(sample) for sample in samples])
XBB = np.sum(Ni * XBni) / N
DB = XBni - XBB

F0 = test_statistic_F(samples)
Ns = 1000
Fstar = []
for _ in range(Ns):
    Ysamples = []
    for sample, DBi in zip(samples, DB):
        Xstar = np.array(random.choices(sample, k=len(sample)))
        Ysamples.append(Xstar - DBi)
    Fs = test_statistic_F(Ysamples)
    Fstar.append(Fs)
Fstar = np.array(Fstar)

print(f'F = {F0:.3f}')
print('ratio', sum(Fstar > F0)/len(Fstar))

ax = pd.Series(Fstar).hist(bins=14, color='grey')
ax.axvline(F0, color='black', lw=2)
ax.set_xlabel('F* values')
plt.show()
```

```
| F = 6.076
| ratio 0.003
```

(fig:mpg-equal-mean) $F = 6.076$, $P^* = 0.003$ and the hypothesis of equal means is rejected. See Fig. 3.10 for the calculated EBD.

Solution 3.55 In Python:

```
np.random.seed(1)

def qbinomBoot(x, p):
    return stats.binom.ppf(p, 50, p=x.mean())

for p_real in (0.2, 0.1, 0.05):
    defects = stats.bernoulli.rvs(p_real, size=50)
    B_025 = pg.compute_bootci(defects, func=lambda x: qbinomBoot(x, p=0.025),
                              n_boot=500, seed=1, return_dist=True)
    B_975 = pg.compute_bootci(defects, func=lambda x: qbinomBoot(x, p=0.975),
                              n_boot=500, seed=1, return_dist=True)
    tol_int = [np.quantile(B_025[1], 0.025), np.quantile(B_975[1], 0.975)]
    print(f'Tolerance interval p={p_real}: ({tol_int[0]}, {tol_int[1]})')
```

```
| Tolerance interval p=0.2: (1.0, 22.0)
| Tolerance interval p=0.1: (0.0, 17.0)
| Tolerance interval p=0.05: (0.0, 9.0)
```

The tolerance intervals of the number of defective items in future batches of size $N = 50$, with $\alpha = 0.05$ and $\beta = 0.05$ are

p	Limits	
	Lower	Upper
0.2	1	23
0.1	0	17
0.05	0	9

Solution 3.56 In Python:

```
np.random.seed(1)

def getQuantile(x, p):
    return np.quantile(x, p)

oturb = mistat.load_data('OTURB.csv')
B_025 = pg.compute_bootci(oturb, func=lambda x: getQuantile(x, p=0.025),
                          n_boot=500, seed=1, return_dist=True)
B_975 = pg.compute_bootci(oturb, func=lambda x: getQuantile(x, p=0.975),
                          n_boot=500, seed=1, return_dist=True)
tol_int = [np.quantile(B_025[1], 0.025), np.quantile(B_975[1], 0.975)]
print(f'Tolerance interval ({tol_int[0]}, {tol_int[1]})')
```

```
| Tolerance interval (0.2399, 0.68305)
```

A (0.95, 0.95) tolerance interval for OTURB.csv is (0.24, 0.683).

Solution 3.57 In Python:

```
cyclt = mistat.load_data('CYCLT.csv')
# make use of the fact that a True value is interpreted as 1 and False as 0
print('Values greater 0.7:', sum(cyclt>0.7))
```

| Values greater 0.7: 20

We find that in the sample of $n = 50$ cycle times, there are $X = 20$ values greater than 0.7. If the hypothesis is $H_0 : \xi_{.5} \leq 0.7$, the probability of observing a value smaller than 0.7 is $p \geq \frac{1}{2}$. Thus, the sign test rejects H_0 if $X < B^{-1}(\alpha; 50, \frac{1}{2})$. For $\alpha = 0.10$ the critical value is $k_\alpha = 20$. H_0 is not rejected.

Solution 3.58 We apply the wilcoxon test from `scipy` on the differences of `oelect` from 220.

```
oelect = mistat.load_data('OELECT.csv')
print(stats.wilcoxon(oelect-220))
```

```
| WilcoxonResult(statistic=1916.0, pvalue=0.051047599707252124)
```

The null hypothesis is rejected with P value equal to 0.051.

Solution 3.59 In Python

```
car = mistat.load_data('CAR.csv')
fourCylinder = car[car['cyl'] == 4]
uscars = fourCylinder[fourCylinder['origin'] == 1]
foreign = fourCylinder[fourCylinder['origin'] != 1]

print(f'Mean of Sample 1 (U.S. made) {np.mean(uscars["turn"]):.3f}')
print(f'Mean of Sample 2 (foreign) {np.mean(foreign["turn"]):.3f}')

_ = mistat.randomizationTest(uscars['turn'], foreign['turn'], np.mean,
                             aggregate_stats=lambda x: x[0] - x[1],
                             n_boot=1000, seed=1)
```

```
| Mean of Sample 1 (U.S. made) 36.255
| Mean of Sample 2 (foreign) 33.179
| Original stat is 3.075758
| Original stat is at quantile 1001 of 1001 (100.00%)
| Distribution of bootstrap samples:
| min: -2.12, median: 0.01, max: 2.68
```

The original stat 3.08 is outside of the distribution of the bootstrap samples. The difference between the means of the turn diameters is therefore significant. Foreign cars have on the average a smaller turn diameter.

Chapter 4

Variability in Several Dimensions and Regression Models

Import required modules and define required functions

```
import random
import numpy as np
import pandas as pd
import pingouin as pg
from scipy import stats
import statsmodels.api as sm
import statsmodels.formula.api as smf
import statsmodels.stats as sms
import seaborn as sns
import matplotlib.pyplot as plt

import mistat
```

Solution 4.1 In Fig. 4.1, one sees that horsepower and miles per gallon are inversely proportional. Turn diameter seems to increase with horsepower.

{fig:ex_car_pairplot}

```
car = mistat.load_data('CAR')
sns.pairplot(car[['turn', 'hp', 'mpg']])
plt.show()
```

```
-----OptionError
Traceback (most recent call last)Cell In[1], line 2
      1 car = mistat.load_data('CAR')
----> 2 sns.pairplot(car[['turn', 'hp', 'mpg']])
      3 plt.show()
File /usr/local/lib/python3.9/site-packages/seaborn/_decorators.py:46,
in _deprecate_positional_args.<locals>.inner_f(*args, **kwargs)
     36 warnings.warn(
     37     "Pass the following variable{} as {}keyword arg{}: {}".
     38     "From version 0.12, the only valid positional argument
     39     (...)
     43     FutureWarning
     44 )
     45 kwargs.update({k: arg for k, arg in zip(sig.parameters,
args)})
--> 46 return f(**kwargs)
File /usr/local/lib/python3.9/site-packages/seaborn/axisgrid.py:2126,
in pairplot(data, hue, hue_order, palette, vars, x_vars, y_vars, kind,
```

```

diag_kind, markers, height, aspect, corner, dropna, plot_kws,
diag_kws, grid_kws, size)
2124 diag_kws.setdefault("legend", False)
2125 if diag_kind == "hist":
-> 2126     grid.map_diag(histplot, **diag_kws)
2127 elif diag_kind == "kde":
2128     diag_kws.setdefault("fill", True)
File /usr/local/lib/python3.9/site-packages/seaborn/axisgrid.py:1478,
in PairGrid.map_diag(self, func, **kwargs)
1476     plot_kws.setdefault("hue_order", self._hue_order)
1477     plot_kws.setdefault("palette", self._orig_palette)
-> 1478     func(x=vector, **plot_kws)
1479     ax.legend_ = None
1481 self._add_axis_labels()
File /usr/local/lib/python3.9/site-
packages/seaborn/distributions.py:1462, in histplot(data, x, y, hue,
weights, stat, bins, binwidth, binrange, discrete, cumulative,
common_bins, common_norm, multiple, element, fill, shrink, kde,
kde_kws, line_kws, thresh, pthresh, pmax, cbar, cbar_ax, cbar_kws,
palette, hue_order, hue_norm, color, log_scale, legend, ax, **kwargs)
1451 estimate_kws = dict(
1452     stat=stat,
1453     bins=bins,
1454     (...)
1455     cumulative=cumulative,
1456 )
1460 if p.univariate:
-> 1462     p.plot_univariate_histogram(
1463         multiple=multiple,
1464         element=element,
1465         fill=fill,
1466         shrink=shrink,
1467         common_norm=common_norm,
1468         common_bins=common_bins,
1469         kde=kde,
1470         kde_kws=kde_kws,
1471         color=color,
1472         legend=legend,
1473         estimate_kws=estimate_kws,
1474         line_kws=line_kws,
1475         **kwargs,
1476     )
1478 else:
1480     p.plot_bivariate_histogram(
1481         common_bins=common_bins,
1482         common_norm=common_norm,
1483         (...)
1484         **kwargs,
1485     )
File /usr/local/lib/python3.9/site-
packages/seaborn/distributions.py:428, in
_DistributionPlotter.plot_univariate_histogram(self, multiple,
element, fill, common_norm, common_bins, shrink, kde, kde_kws, color,
legend, line_kws, estimate_kws, **plot_kws)
418     densities = self._compute_univariate_density(
419         self.data_variable,
420         common_norm,
421         (...)
422         warn_singular=False,
423     )
427 # First pass through the data to compute the histograms
--> 428 for sub_vars, sub_data in self.iter_data("hue",
from_comp_data=True):
429
430     # Prepare the relevant data
431     key = tuple(sub_vars.items())
432     sub_data = sub_data.dropna()

```

```

File /usr/local/lib/python3.9/site-packages/seaborn/_core.py:983, in
VectorPlotter.iter_data(self, grouping_vars, reverse, from_comp_data)
    978 grouping_vars = [
    979     var for var in grouping_vars if var in self.variables
    980 ]
    982 if from_comp_data:
--> 983     data = self.comp_data
    984 else:
    985     data = self.plot_data
File /usr/local/lib/python3.9/site-packages/seaborn/_core.py:1054, in
VectorPlotter.comp_data(self)
    1050 axis = getattr(ax, f"{var}axis")
    1052 # Use the converter assigned to the axis to get a float
representation
    1053 # of the data, passing np.nan or pd.NA through (pd.NA becomes
np.nan)
--> 1054 with pd.option_context('mode.use_inf_as_null', True):
    1055     orig = self.plot_data[var].dropna()
    1056 comp_col = pd.Series(index=orig.index, dtype=float, name=var)
File /usr/local/lib/python3.9/site-
packages/pandas/_config/config.py:441, in
option_context.__enter__(self)
    440 def __enter__(self) -> None:
--> 441     self.undo = [(pat, _get_option(pat, silent=True)) for pat,
val in self.ops]
    443     for pat, val in self.ops:
    444         _set_option(pat, val, silent=True)
File /usr/local/lib/python3.9/site-
packages/pandas/_config/config.py:441, in <listcomp>(.0)
    440 def __enter__(self) -> None:
--> 441     self.undo = [(pat, _get_option(pat, silent=True)) for pat,
val in self.ops]
    443     for pat, val in self.ops:
    444         _set_option(pat, val, silent=True)
File /usr/local/lib/python3.9/site-
packages/pandas/_config/config.py:135, in _get_option(pat, silent)
    134 def _get_option(pat: str, silent: bool = False) -> Any:
--> 135     key = _get_single_key(pat, silent)
    137     # walk the nested dict
    138     root, k = _get_root(key)
File /usr/local/lib/python3.9/site-
packages/pandas/_config/config.py:121, in _get_single_key(pat, silent)
    119     if not silent:
    120         _warn_if_deprecated(pat)
--> 121     raise OptionError(f"No such keys(s): {repr(pat)}")
    122 if len(keys) > 1:
    123     raise OptionError("Pattern matched multiple keys")
OptionError: "No such keys(s): 'mode.use_inf_as_null'"

```

Solution 4.2 The box plots in Fig. 4.2 show that cars from Asia generally have the smallest turn diameter. The maximal turn diameter of cars from Asia is smaller than the median turn diameter of U.S. cars. European cars tend to have larger turn diameter than those from Asia, but smaller than those from the U.S.

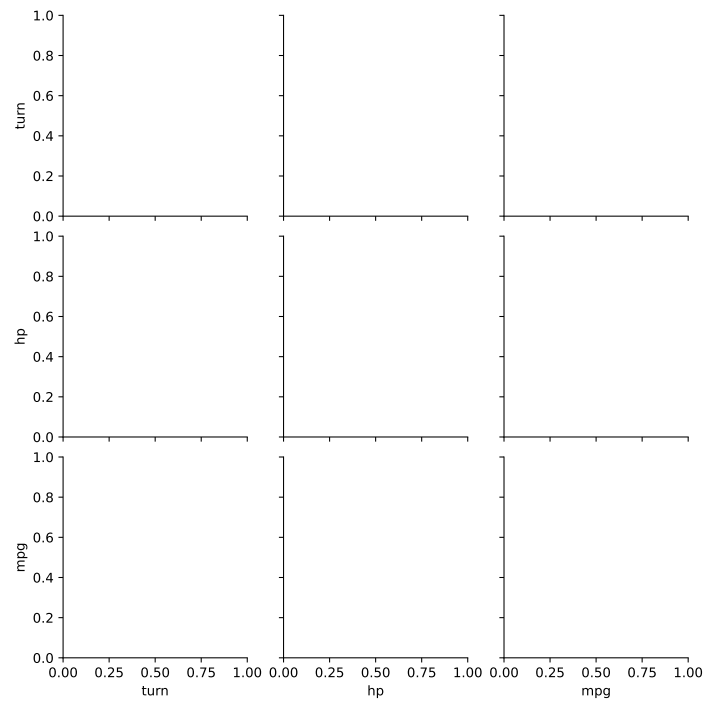
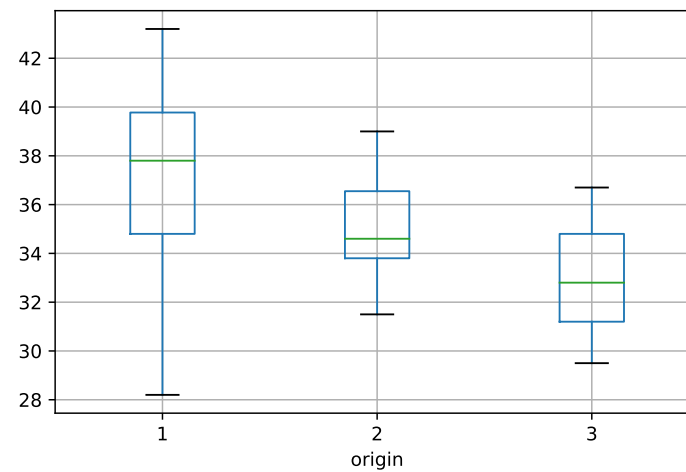
{fig:ex_car_boxplots}

```

car = mistat.load_data('CAR')

ax = car.boxplot(column='turn', by='origin')
ax.set_title('')
ax.get_figure().suptitle('')
ax.set_xlabel('origin')
plt.show()

```

**Fig. 4.1** Scatterplot matrix for CAR dataset*{fig:ex_car_pairplot}***Fig. 4.2** Boxplots of turn diameter by origin for CAR dataset*{fig:ex_car_boxplots}*

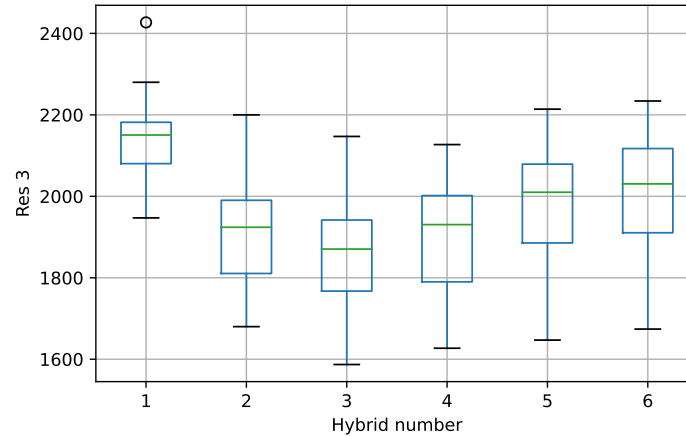


Fig. 4.3 Multiple box plots of Res 3 grouped by hybrid number

{fig:ex_hadpas_plot_i}

Solution 4.3 (i) The multiple box plots (see Fig. 4.3) show that the conditional distributions of res3 at different hybrids are different.

{fig:ex_hadpas_plot_ii}

```
hadpas = mistat.load_data('HADPAS')
ax = hadpas.boxplot(column='res3', by='hyb')
ax.set_title('')
ax.get_figure().suptitle('')
ax.set_xlabel('Hybrid number')
ax.set_ylabel('Res 3')
plt.show()
```

(ii) The matrix plot of all the Res variables (see Fig. 4.4) reveals that Res 3 and Res 7 are positively correlated. Res 20 is generally larger than the corresponding Res 14. Res 18 and Res 20 seem to be negatively associated.

{fig:ex_hadpas_plot_ii}

```
sns.pairplot(hadpas[['res3', 'res7', 'res18', 'res14', 'res20']])
plt.show()
```

```
-----OptionError
Traceback (most recent call last)Cell In[1], line 1
----> 1 sns.pairplot(hadpas[['res3', 'res7', 'res18', 'res14',
    'res20']])
    2 plt.show()
File /usr/local/lib/python3.9/site-packages/seaborn/_decorators.py:46,
in _deprecate_positional_args.<locals>.inner_f(*args, **kwargs)
    36 warnings.warn(
    37     "Pass the following variable() as {}keyword arg{}: {}".
    38     "From version 0.12, the only valid positional argument
    (...)
    43     FutureWarning
    44 )
    45 kwargs.update({k: arg for k, arg in zip(sig.parameters,
args)})
----> 46 return f(**kwargs)
```

```

File /usr/local/lib/python3.9/site-packages/seaborn/axisgrid.py:2126,
in pairplot(data, hue, hue_order, palette, vars, x_vars, y_vars, kind,
diag_kind, markers, height, aspect, corner, dropna, plot_kws,
diag_kws, grid_kws, size)
    2124 diag_kws.setdefault("legend", False)
    2125 if diag_kind == "hist":
-> 2126     grid.map_diag(histplot, **diag_kws)
    2127 elif diag_kind == "kde":
    2128     diag_kws.setdefault("fill", True)
File /usr/local/lib/python3.9/site-packages/seaborn/axisgrid.py:1478,
in PairGrid.map_diag(self, func, **kwargs)
    1476 plot_kwargs.setdefault("hue_order", self._hue_order)
    1477 plot_kwargs.setdefault("palette", self._orig_palette)
-> 1478     func(x=vector, **plot_kwargs)
    1479     ax.legend_ = None
    1481 self._add_axis_labels()
File /usr/local/lib/python3.9/site-
packages/seaborn/distributions.py:1462, in histplot(data, x, y, hue,
weights, stat, bins, binwidth, binrange, discrete, cumulative,
common_bins, common_norm, multiple, element, fill, shrink, kde,
kde_kws, line_kws, thresh, pthresh, pmax, cbar, cbar_ax, cbar_kws,
palette, hue_order, hue_norm, color, log_scale, legend, ax, **kwargs)
    1451 estimate_kws = dict(
    1452     stat=stat,
    1453     bins=bins,
    (...)
    1457     cumulative=cumulative,
    1458 )
    1460 if p.univariate:
-> 1462     p.plot_univariate_histogram(
    1463         multiple=multiple,
    1464         element=element,
    1465         fill=fill,
    1466         shrink=shrink,
    1467         common_norm=common_norm,
    1468         common_bins=common_bins,
    1469         kde=kde,
    1470         kde_kws=kde_kws,
    1471         color=color,
    1472         legend=legend,
    1473         estimate_kws=estimate_kws,
    1474         line_kws=line_kws,
    1475         **kwargs,
    1476     )
    1478 else:
    1480     p.plot_bivariate_histogram(
    1481         common_bins=common_bins,
    1482         common_norm=common_norm,
    (...)
    1492         **kwargs,
    1493     )
File /usr/local/lib/python3.9/site-
packages/seaborn/distributions.py:428, in
_DistributionPlotter.plot_univariate_histogram(self, multiple,
element, fill, common_norm, common_bins, shrink, kde, kde_kws, color,
legend, line_kws, estimate_kws, **plot_kws)
    418     densities = self._compute_univariate_density(
    419         self.data_variable,
    420         common_norm,
    (...)
    424         warn_singular=False,
    425     )
    427 # First pass through the data to compute the histograms
-> 428 for sub_vars, sub_data in self.iter_data("hue",
from_comp_data=True):
    429
    430     # Prepare the relevant data

```

```

431     key = tuple(sub_vars.items())
432     sub_data = sub_data.dropna()
File /usr/local/lib/python3.9/site-packages/seaborn/_core.py:983, in
VectorPlotter.iter_data(self, grouping_vars, reverse, from_comp_data)
978     grouping_vars = [
979         var for var in grouping_vars if var in self.variables
980     ]
981     if from_comp_data:
--> 983         data = self.comp_data
984     else:
985         data = self.plot_data
File /usr/local/lib/python3.9/site-packages/seaborn/_core.py:1054, in
VectorPlotter.comp_data(self)
1050     axis = getattr(ax, f"{var}axis")
1052     # Use the converter assigned to the axis to get a float
representation
1053     # of the data, passing np.nan or pd.NA through (pd.NA becomes
np.nan)
-> 1054     with pd.option_context('mode.use_inf_as_null', True):
1055         orig = self.plot_data[var].dropna()
1056     comp_col = pd.Series(index=orig.index, dtype=float, name=var)
File /usr/local/lib/python3.9/site-
packages/pandas/_config/config.py:441, in
option_context.__enter__(self)
440     def __enter__(self) -> None:
--> 441         self.undo = [(pat, _get_option(pat, silent=True)) for pat,
val in self.ops]
443         for pat, val in self.ops:
444             _set_option(pat, val, silent=True)
File /usr/local/lib/python3.9/site-
packages/pandas/_config/config.py:441, in <listcomp>(.0)
440     def __enter__(self) -> None:
--> 441         self.undo = [(pat, _get_option(pat, silent=True)) for pat,
val in self.ops]
443         for pat, val in self.ops:
444             _set_option(pat, val, silent=True)
File /usr/local/lib/python3.9/site-
packages/pandas/_config/config.py:135, in _get_option(pat, silent)
134     def _get_option(pat: str, silent: bool = False) -> Any:
--> 135         key = _get_single_key(pat, silent)
137         # walk the nested dict
138         root, k = _get_root(key)
File /usr/local/lib/python3.9/site-
packages/pandas/_config/config.py:121, in _get_single_key(pat, silent)
119         if not silent:
120             _warn_if_deprecated(pat)
--> 121         raise OptionError(f"No such keys(s): {repr(pat)}")
122     if len(keys) > 1:
123         raise OptionError("Pattern matched multiple keys")
OptionError: "No such keys(s): 'mode.use_inf_as_null'"

```

Solution 4.4 The joint frequency distribution of horsepower versus miles per gallon is

```

car = mistat.load_data('CAR')
binned_car = pd.DataFrame({
    'hp': pd.cut(car['hp'], bins=np.arange(50, 275, 25)),
    'mpg': pd.cut(car['mpg'], bins=np.arange(10, 40, 5)),
})
freqDist = pd.crosstab(binned_car['hp'], binned_car['mpg'])
print(freqDist)
# You can get distributions for hp and mpg by summing along an axis
print(freqDist.sum(axis=0))
print(freqDist.sum(axis=1))

```

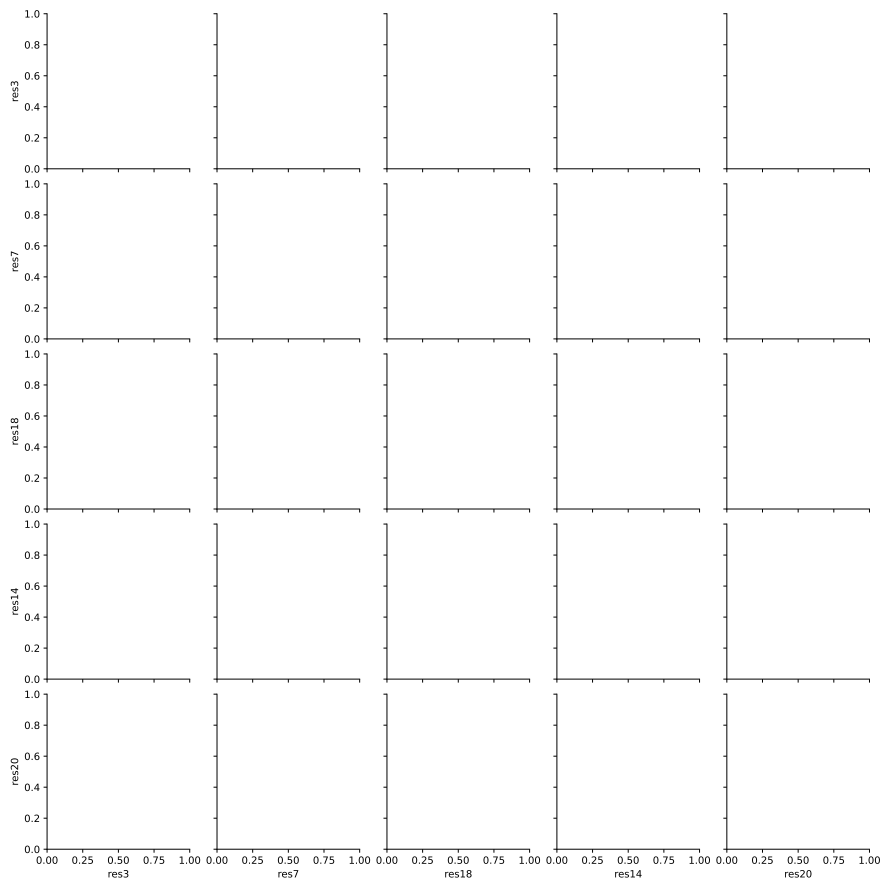


Fig. 4.4 Scatterplot matrix Res variables for HADPAS dataset (fig:ex_hadpas_plot_ii)

mpg	(10, 15]	(15, 20]	(20, 25]	(25, 30]	(30, 35]
hp					
(50, 75]	0	1	0	4	2
(75, 100]	0	0	23	11	0
(100, 125]	0	10	11	1	0
(125, 150]	0	14	3	1	0
(150, 175]	0	17	0	0	0
(175, 200]	1	5	0	0	0
(200, 225]	1	3	0	0	0
(225, 250]	0	1	0	0	0
mpg					
(10, 15]	2				
(15, 20]	51				
(20, 25]	37				
(25, 30]	17				
(30, 35]	2				
dtype: int64					
hp					
(50, 75]	7				


```
(75, 100]      34
(100, 125]     22
(125, 150]     18
(150, 175]     17
(175, 200]      6
(200, 225]      4
(225, 250]      1
dtype: int64
```

The intervals for HP are from 50 to 250 at fixed length of 25. The intervals for MPG are from 10 to 35 at length 5. Students may get different results by defining the intervals differently.

Solution 4.5 The joint frequency distribution of Res 3 and Res 14 is given in the following table:

```
hadpas = mistat.load_data('HADPAS')
binned_hadpas = pd.DataFrame({
    'res3': pd.cut(hadpas['res3'], bins=np.arange(1580, 2780, 200)),
    'res14': pd.cut(hadpas['res14'], bins=np.arange(900, 3000, 300)),
})
pd.crosstab(binned_hadpas['res14'], binned_hadpas['res3'])
```

res3	(1580, 1780]	(1780, 1980]	(1980, 2180]	(2180, 2380]
res14				
(900, 1200]	11	3	3	0
(1200, 1500]	11	33	28	2
(1500, 1800]	5	16	24	6
(1800, 2100]	2	11	12	5
(2100, 2400]	0	9	8	1
(2400, 2700]	0	0	1	0

res3	(2380, 2580]
res14	
(900, 1200]	0
(1200, 1500]	0
(1500, 1800]	0
(1800, 2100]	1
(2100, 2400]	0
(2400, 2700]	0

The intervals for Res 3 start at 1580 and end at 2580 with length of 200. The intervals of Res 14 start at 900 and end at 2700 with length of 300.

Solution 4.6 The following is the conditional frequency distribution of Res 3, given that Res 14 is between 1300 and 1500 ohms:

```
hadpas = mistat.load_data('HADPAS')
in_range = hadpas[hadpas['res14'].between(1300, 1500)]
pd.cut(in_range['res3'], bins=np.arange(1580, 2780, 200)).value_counts(sort=False)
```

```
res3
(1580, 1780]      8
(1780, 1980]     21
(1980, 2180]     25
(2180, 2380]      2
(2380, 2580]      0
Name: count, dtype: int64
```

Solution 4.7 Following the instructions in the question we obtained the following results:

```

hadpas = mistat.load_data('HADPAS')
bins = [900, 1200, 1500, 1800, 2100, 3000]
binned_res14 = pd.cut(hadpas['res14'], bins=bins)

results = []
for group, df in hadpas.groupby(binned_res14):
    res3 = df['res3']
    results.append({
        'res3': group,
        'N': len(res3),
        'mean': res3.mean(),
        'std': res3.std(),
    })
pd.DataFrame(results)

```

	res3	N	mean	std
0	(900, 1200]	17	1779.117647	162.348730
1	(1200, 1500]	74	1952.175676	154.728251
2	(1500, 1800]	51	1997.196078	151.608841
3	(1800, 2100]	31	2024.774194	156.749845
4	(2100, 3000]	19	1999.736842	121.505758

Solution 4.8 In Python

```

df = pd.DataFrame([
    [10.0, 8.04, 10.0, 9.14, 10.0, 7.46, 8.0, 6.58],
    [8.0, 6.95, 8.0, 8.14, 8.0, 6.77, 8.0, 5.76],
    [13.0, 7.58, 13.0, 8.74, 13.0, 12.74, 8.0, 7.71],
    [9.0, 8.81, 9.0, 8.77, 9.0, 7.11, 8.0, 8.84],
    [11.0, 8.33, 11.0, 9.26, 11.0, 7.81, 8.0, 8.47],
    [14.0, 9.96, 14.0, 8.10, 14.0, 8.84, 8.0, 7.04],
    [6.0, 7.24, 6.0, 6.13, 6.0, 6.08, 8.0, 5.25],
    [4.0, 4.26, 4.0, 3.10, 4.0, 5.39, 19.0, 12.50],
    [12.0, 10.84, 12.0, 9.13, 12.0, 8.15, 8.0, 5.56],
    [7.0, 4.82, 7.0, 7.26, 7.0, 6.42, 8.0, 7.91],
    [5.0, 5.68, 5.0, 4.74, 5.0, 5.73, 8.0, 6.89],
], columns=['x1', 'y1', 'x2', 'y2', 'x3', 'y3', 'x4', 'y4'])

results = []
for i in (1, 2, 3, 4):
    x = df[f'x{i}']
    y = df[f'y{i}']
    model = smf.ols(formula=f'y{i} ~ 1 + x{i}', data=df).fit()
    results.append({
        'Data Set': i,
        'Intercept': model.params['Intercept'],
        'Slope': model.params[f'x{i}'],
        'R2': model.rsquared,
    })
pd.DataFrame(results)

```

	Data Set	Intercept	Slope	R2
0	1	3.000091	0.500091	0.666542
1	2	3.000909	0.500000	0.666242
2	3	3.002455	0.499727	0.666324
3	4	3.001727	0.499909	0.666707

Notice the influence of the point (19,12.5) on the regression in Data Set 4. Without this point the correlation between x and y is zero.

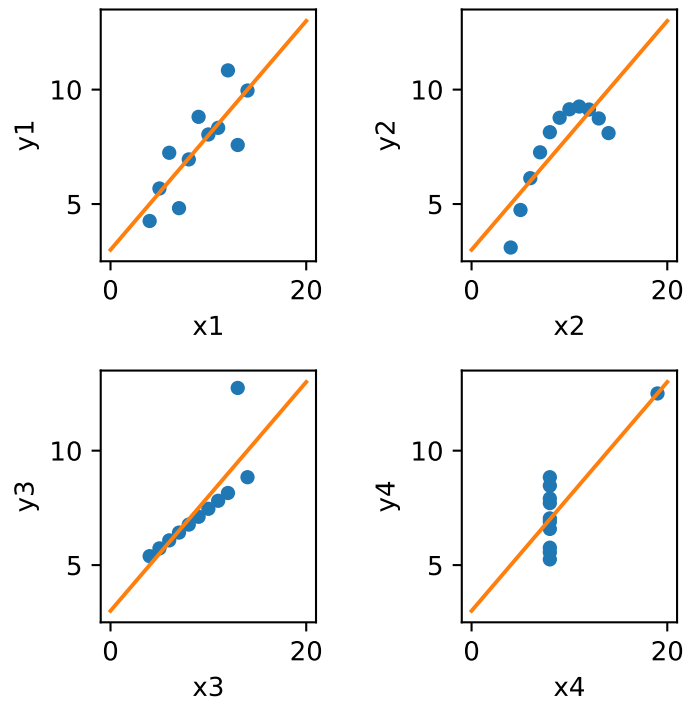


Fig. 4.5 Anscombe's quartet

{fig:AnscombeQuartet}

{fig:AnscombeQuartet}

The dataset is known as Anscombe's quartet (see Fig. 4.5). It not only has identical linear regression, but it has also identical means and variances of x and y , and correlation between x and y . The dataset clearly demonstrates the importance of visualization in data analysis.

Solution 4.9 The correlation matrix:

```
car = mistat.load_data('CAR')
car[['turn', 'hp', 'mpg']].corr()
```

	turn	hp	mpg
turn	1.000000	0.507610	-0.541061
hp	0.507610	1.000000	-0.754716
mpg	-0.541061	-0.754716	1.000000

Solution 4.10 $SSE = \sum_{i=1}^n (Y_i - \beta_0 - \beta_1 X_{i1} - \beta_2 X_{i2})^2$
(i)

$$\begin{aligned}\frac{\partial}{\partial \beta_0} SSE &= -2 \sum_{i=1}^n (Y_i - \beta_0 - \beta_1 X_{i1} - \beta_2 X_{i2}) \\ \frac{\partial}{\partial \beta_1} SSE &= -2 \sum_{i=1}^n X_{i1} (Y_i - \beta_0 - \beta_1 X_{i1} - \beta_2 X_{i2}) \\ \frac{\partial}{\partial \beta_2} SSE &= -2 \sum_{i=1}^n X_{i2} (Y_i - \beta_0 - \beta_1 X_{i1} - \beta_2 X_{i2}).\end{aligned}$$

Equating these partial derivatives to zero and arranging terms, we arrive at the following set of linear equations:

$$\begin{bmatrix} n & \sum_{i=1}^n X_{i1} & \sum_{i=1}^n X_{i2} \\ \sum_{i=1}^n X_{i1} & \sum_{i=1}^n X_{i1}^2 & \sum_{i=1}^n X_{i1} X_{i2} \\ \sum_{i=1}^n X_{i2} & \sum_{i=1}^n X_{i1} X_{i2} & \sum_{i=1}^n X_{i2}^2 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^n Y_i \\ \sum_{i=1}^n X_{i1} Y_i \\ \sum_{i=1}^n X_{i2} Y_i \end{bmatrix}$$

(ii) Let b_0, b_1, b_2 be the (unique) solution. From the first equation we get, after dividing by n , $b_0 = \bar{Y} - \bar{X}_1 b_1 - \bar{X}_2 b_2$, where $\bar{Y} = \frac{1}{n} \sum_{i=1}^n Y_i$, $\bar{X}_1 = \frac{1}{n} \sum_{i=1}^n X_{i1}$, $\bar{X}_2 = \frac{1}{n} \sum_{i=1}^n X_{i2}$. Substituting b_0 in the second and third equations and arranging terms, we obtain the reduced system of equations:

$$\begin{bmatrix} (Q_1 - n\bar{X}_1^2) & (P_{12} - n\bar{X}_1\bar{X}_2) \\ (P_{12} - n\bar{X}_1\bar{X}_2) & (Q_2 - n\bar{X}_2^2) \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} P_{1y} - n\bar{X}_1\bar{Y} \\ P_{2y} - n\bar{X}_2\bar{Y} \end{bmatrix}$$

where $Q_1 = \sum X_{i1}^2$, $Q_2 = \sum X_{i2}^2$, $P_{12} = \sum X_{i1} X_{i2}$ and $P_{1y} = \sum X_{i1} Y_i$, $P_{2y} = \sum X_{i2} Y_i$. Dividing both sides by $(n-1)$ we obtain Eq. (4.4.3), and solving we get b_1 and b_2 .

Solution 4.11 We get the following result using `statsmodels`

```
car = mistat.load_data('CAR')
```

```
model = smf.ols(formula='mpg ~ 1 + hp + turn', data=car).fit()
print(model.summary2())
```

```

Results: Ordinary least squares
=====
Model:                OLS                Adj. R-squared:      0.596
Dependent Variable:    mpg                AIC:                511.2247
Date:                 2023-05-03 21:16      BIC:                519.2988
No. Observations:     109                Log-Likelihood:     -252.61
Df Model:              2                  F-statistic:        80.57
Df Residuals:          106                Prob (F-statistic): 5.29e-22
R-squared:             0.603              Scale:             6.2035
=====
              Coef.    Std.Err.    t      P>|t|    [0.025    0.975]
-----
Intercept    38.2642    2.6541   14.4167  0.0000   33.0020   43.5263
hp           -0.0631    0.0069   -9.1070  0.0000   -0.0768   -0.0493

```

```

turn          -0.2510    0.0838  -2.9965   0.0034  -0.4171  -0.0849
-----
Omnibus:          12.000          Durbin-Watson:          2.021
Prob(Omnibus):    0.002          Jarque-Bera (JB):        25.976
Skew:            -0.335          Prob(JB):           0.000
Kurtosis:         5.296          Condition No.:       1507
=====
* The condition number is large (2e+03). This might indicate
strong multicollinearity or other numerical problems.

```

The regression equation is $\text{MPG} = 38.3 - 0.251 \times \text{turn} - 0.0631 \times \text{hp}$.

We see that only 60% of the variability in MPG is explained by the linear relationship with Turn and HP. Both variables contribute significantly to the regression.

Solution 4.12 The partial correlation is -0.70378 .

```

car = mistat.load_data('CAR')

# y: mpg, x1: cyl, x2: hp
model_1 = smf.ols(formula='mpg ~ cyl + 1', data=car).fit()
e_1 = model_1.resid

model_2 = smf.ols(formula='hp ~ cyl + 1', data=car).fit()
e_2 = model_2.resid

print(f'Partial correlation {stats.pearsonr(e_1, e_2)[0]:.5f}')

```

```

| Partial correlation -0.70378

```

Solution 4.13 In Python:

```

car = mistat.load_data('CAR')

# y: mpg, x1: hp, x2: turn
model_1 = smf.ols(formula='mpg ~ hp + 1', data=car).fit()
e_1 = model_1.resid
print('Model mpg ~ hp + 1:\n', model_1.params)

model_2 = smf.ols(formula='turn ~ hp + 1', data=car).fit()
e_2 = model_2.resid
print('Model turn ~ hp + 1:\n', model_2.params)

print('Partial correlation', stats.pearsonr(e_1, e_2)[0])
df = pd.DataFrame({'e1': e_1, 'e2': e_2})
model_partial = smf.ols(formula='e1 ~ e2 + 1', data=df).fit()
# print(model_partial.summary2())
print('Model e1 ~ e2:\n', model_partial.params)

```

```

| Model mpg ~ hp + 1:
|   Intercept    30.663308
|   hp          -0.073611
| dtype: float64
| Model turn ~ hp + 1:
|   Intercept    30.281255
|   hp           0.041971
| dtype: float64
| Partial correlation -0.27945246615045016
| Model e1 ~ e2:
|   e2          -0.251008
| dtype: float64

```

The partial regression equation is $\hat{e}_1 = -0.251\hat{e}_2$.

Solution 4.14 The regression of MPG on HP is $\text{MPG} = 30.6633 - 0.07361 \text{ HP}$. The regression of TurnD on HP is $\text{TurnD} = 30.2813 + 0.041971 \text{ HP}$. The regression of the residuals \hat{e}_1 on \hat{e}_2 is $\hat{e}_1 = -0.251 \cdot \hat{e}_2$. Thus,

$$\begin{aligned}\text{Const. :} \quad & b_0 = 30.6633 + 30.2813 \times 0.251 = 38.2639 \\ \text{HP :} \quad & b_1 = -0.07361 + 0.041971 \times 0.251 = -0.063076 \\ \text{TurnD :} \quad & b_2 = -0.251.\end{aligned}$$

Solution 4.15 The regression of Cap Diameter on Diam2 and Diam3 is

```
almpin = mistat.load_data('ALMPIN')
model = smf.ols('capDiam ~ 1 + diam2 + diam3', data=almpin).fit()
model.summary2()
```

```
<class 'statsmodels.iolib.summary2.Summary'>
"""
                Results: Ordinary least squares
=====
Model:                OLS                Adj. R-squared:    0.842
Dependent Variable:    capDiam            AIC:              -482.1542
Date:                 2023-05-03 21:16    BIC:              -475.4087
No. Observations:     70                Log-Likelihood:    244.08
Df Model:              2                 F-statistic:       184.2
Df Residuals:          67                Prob (F-statistic): 5.89e-28
R-squared:             0.846              Scale:           5.7272e-05
=====
                Coef.      Std.Err.      t      P>|t|      [0.025   0.975]
-----
Intercept          4.7565      0.5501     8.6467   0.0000    3.6585   5.8544
diam2               0.5040      0.1607     3.1359   0.0025    0.1832   0.8248
diam3               0.5203      0.1744     2.9830   0.0040    0.1722   0.8684
=====
Omnibus:            1.078                Durbin-Watson:      2.350
Prob(Omnibus):      0.583                Jarque-Bera (JB):   0.976
Skew:               -0.071                Prob(JB):           0.614
Kurtosis:            2.439                Condition No.:      8689
=====
* The condition number is large (9e+03). This might indicate
strong multicollinearity or other numerical problems.
"""
```

The dependence of CapDiam on Diam2, without Diam1 is significant. This is due to the fact that Diam1 and Diam2 are highly correlated ($\rho = 0.957$). If Diam1 is in the regression, then Diam2 does not furnish additional information on CapDiam. If Diam1 is not included then Diam2 is very informative.

Solution 4.16 The regression of yield (*Yield*) on the four variables is:

```
gasol = mistat.load_data('GASOL')
# rename column 'yield' to 'Yield' as 'yield' is a special keyword in Python
gasol = gasol.rename(columns={'yield': 'Yield'})
model = smf.ols(formula='Yield ~ x1 + x2 + astm + endPt',
                 data=gasol).fit()
print(model.summary2())
```

```

Results: Ordinary least squares
=====
Model: OLS Adj. R-squared: 0.957
Dependent Variable: Yield AIC: 146.8308
Date: 2023-05-03 21:16 BIC: 154.1595
No. Observations: 32 Log-Likelihood: -68.415
Df Model: 4 F-statistic: 171.7
Df Residuals: 27 Prob (F-statistic): 8.82e-19
R-squared: 0.962 Scale: 4.9927
=====
Coef. Std.Err. t P>|t| [0.025 0.975]
-----
Intercept -6.8208 10.1232 -0.6738 0.5062 -27.5918 13.9502
x1 0.2272 0.0999 2.2739 0.0311 0.0222 0.4323
x2 0.5537 0.3698 1.4976 0.1458 -0.2049 1.3124
astm -0.1495 0.0292 -5.1160 0.0000 -0.2095 -0.0896
endPt 0.1547 0.0064 23.9922 0.0000 0.1414 0.1679
=====
Omnibus: 0.635 Durbin-Watson: 1.402
Prob(Omnibus): 0.728 Jarque-Bera (JB): 0.719
Skew: 0.190 Prob(JB): 0.698
Kurtosis: 2.371 Condition No.: 10714
=====
* The condition number is large (1e+04). This might indicate
strong multicollinearity or other numerical problems.

```

(i) The regression equation is

$$\hat{y} = -6.8 + 0.227x_1 + 0.554x_2 - 0.150 \text{ astm} + 0.155 \text{ endPt}.$$

(ii) $R^2 = 0.962$.

(iii) The regression coefficient of x_2 is not significant.

(iv) Running the multiple regression again, without x_2 , we obtain the equation

```

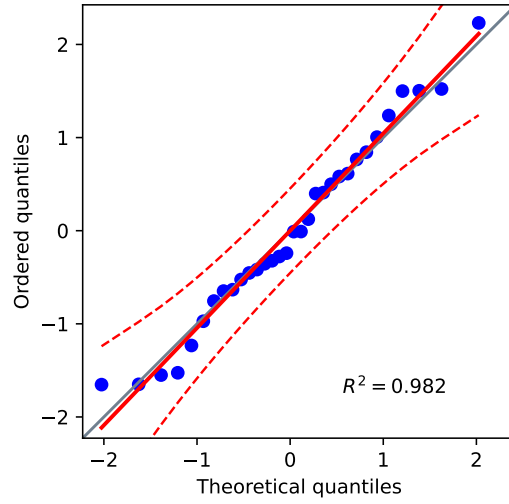
model = smf.ols(formula='Yield ~ x1 + astm + endPt', data=gasol).fit()
print(model.summary2())

```

```

Results: Ordinary least squares
=====
Model: OLS Adj. R-squared: 0.955
Dependent Variable: Yield AIC: 147.3842
Date: 2023-05-03 21:16 BIC: 153.2471
No. Observations: 32 Log-Likelihood: -69.692
Df Model: 3 F-statistic: 218.5
Df Residuals: 28 Prob (F-statistic): 1.59e-19
R-squared: 0.959 Scale: 5.2143
=====
Coef. Std.Err. t P>|t| [0.025 0.975]
-----
Intercept 4.0320 7.2233 0.5582 0.5811 -10.7643 18.8284
x1 0.2217 0.1021 2.1725 0.0384 0.0127 0.4308
astm -0.1866 0.0159 -11.7177 0.0000 -0.2192 -0.1540
endPt 0.1565 0.0065 24.2238 0.0000 0.1433 0.1698
=====
Omnibus: 0.679 Durbin-Watson: 1.150
Prob(Omnibus): 0.712 Jarque-Bera (JB): 0.738
Skew: 0.174 Prob(JB): 0.692
Kurtosis: 2.343 Condition No.: 7479
=====
* The condition number is large (7e+03). This might indicate
strong multicollinearity or other numerical problems.

```



{fig:qqPlotGasolRegResid}

Fig. 4.6 Q - Q plot of gasol regression residuals

$$\hat{y} = 4.03 + 0.222x_1 + 0.554x_2 - 0.187 \text{ astm} + 0.157 \text{ endPt},$$

with $R^2 = 0.959$. Variables x_1 , astm and endPt are important.

(v) Normal probability plotting of the residuals \hat{e} from the equation of (iv) shows that they are normally distributed. (see Fig. 4.6)

{fig:qqPlotGasolRegResid}

Solution 4.17 (i)

$$\begin{aligned} (H) &= (X)(B) = (X)[(X)'(X)]^{-1}(X)' \\ H^2 &= (X)[(X)'(X)]^{-1}(X)'(X)[(X)'(X)]^{-1}(X)' \\ &= (X)[(X)'(X)]^{-1}(X)' \\ &= H. \end{aligned}$$

(ii)

$$\begin{aligned} (Q) &= I - (H) \\ (Q)^2 &= (I - (H))(I - (H)) \\ &= I - (H) - (H) + (H)^2 \\ &= I - (H) \\ &= Q. \\ s_e^2 &= \mathbf{y}'(Q)(Q)\mathbf{y}/(n - k - 1) \\ &= \mathbf{y}'(Q)\mathbf{y}/(n - k - 1). \end{aligned}$$

Solution 4.18 We have $\hat{\mathbf{y}} = (X)\hat{\boldsymbol{\beta}} = (X)(B)\mathbf{y} = (H)\mathbf{y}$ and $\hat{\mathbf{e}} = Q\mathbf{y} = (I - (H))\mathbf{y}$.

$$\begin{aligned}
\hat{\mathbf{y}}'\hat{\mathbf{e}} &= \mathbf{y}'(H)(I - (H))\mathbf{y} \\
&= \mathbf{y}'(H)\mathbf{y} - \mathbf{y}'(H)^2\mathbf{y} \\
&= 0.
\end{aligned}$$

Solution 4.19 $1 - R_{y(x)}^2 = \frac{SSE}{SSD_y}$ where $SSE = \hat{\mathbf{e}}'\hat{\mathbf{e}} = \|\hat{\mathbf{e}}\|^2$.

Solution 4.20 From the basic properties of the $\text{cov}(X, Y)$ operator,

$$\begin{aligned}
\text{cov}\left(\sum_{i=1}^n \beta_i X_i, \sum_{j=1}^n \gamma_j X_j\right) &= \sum_{i=1}^n \sum_{j=1}^n \beta_i \gamma_j \text{cov}(X_i, X_j) \\
&= \sum_{i=1}^n \sum_{j=1}^n \beta_i \gamma_j \mathfrak{X}_{ij} \\
&= \boldsymbol{\beta}'(\mathfrak{X})\boldsymbol{\gamma}.
\end{aligned}$$

Solution 4.21 $\mathbf{W} = (W_1, \dots, W_m)'$ where $W_i = \mathbf{b}_i'\mathbf{X}$ ($i = 1, \dots, m$). \mathbf{b}_i' is the i -th row vector of B . Thus, by the previous exercise $\text{cov}(W_i, W_j) = \mathbf{b}_i'(\mathfrak{X})\mathbf{b}_j$. This is the (i, j) element of the covariance matrix of \mathbf{W} . Hence, the covariance matrix of \mathbf{W} is $C(\mathbf{W}) = (B)(\mathfrak{X})(B)'$.

Solution 4.22 From the model, $\mathfrak{X}(\mathbf{Y}) = \sigma^2 I$ and $\mathbf{b} = (B)\mathbf{Y}$.

$$\begin{aligned}
\mathfrak{X}(\mathbf{b}) &= (B)\mathfrak{X}(\mathbf{Y})(B)' = \sigma^2(B)(B)' \\
&= \sigma^2[(\mathbf{X})'(\mathbf{X})]^{-1} \cdot \mathbf{X}'\mathbf{X}[(\mathbf{X})'(\mathbf{X})]^{-1} \\
&= \sigma^2[(\mathbf{X})'(\mathbf{X})]^{-1}.
\end{aligned}$$

Solution 4.23 Rearrange the dataset into the format suitable for the test outlines in the multiple linear regression section.

```

socell = mistat.load_data('SOCELL')

# combine the two datasets and add the additional columns z and w
socell_1 = socell[['t3', 't1']].copy()
socell_1.columns = ['t3', 't']
socell_1['z'] = 0
socell_2 = socell[['t3', 't2']].copy()
socell_2.columns = ['t3', 't']
socell_2['z'] = 1
combined = pd.concat([socell_1, socell_2])
combined['w'] = combined['z'] * combined['t']

# multiple linear regression model
model_test = smf.ols(formula='t3 ~ t + z + w + 1',
                     data=combined).fit()
print(model_test.summary2())

```

Results: Ordinary least squares

=====

```

Model: OLS Adj. R-squared: 0.952
Dependent Variable: t3 AIC: -58.2308
Date: 2023-05-03 21:16 BIC: -52.3678
No. Observations: 32 Log-Likelihood: 33.115
Df Model: 3 F-statistic: 205.8
Df Residuals: 28 Prob (F-statistic): 3.55e-19
R-squared: 0.957 Scale: 0.0084460
=====
              Coef.      Std.Err.      t      P>|t|      [0.025      0.975]
-----
Intercept    0.5187      0.2144      2.4196    0.0223    0.0796    0.9578
t            0.9411      0.0539     17.4664    0.0000    0.8307    1.0515
z           -0.5052      0.3220     -1.5688    0.1279   -1.1648    0.1545
w            0.0633      0.0783      0.8081    0.4259   -0.0971    0.2237
=====
Omnibus:      1.002      Durbin-Watson:      1.528
Prob(Omnibus): 0.606      Jarque-Bera (JB):      0.852
Skew:         -0.378      Prob(JB):      0.653
Kurtosis:     2.739      Condition No.:      111
=====

```

Neither of the z nor w The P -values corresponding to z and w are 0.128 and 0.426 respectively. Accordingly, we can conclude that the slopes and intercepts of the two simple linear regressions given above are not significantly different. Combining the data we have the following regression line for the combined dataset:

```

model_combined = smf.ols(formula='t3 ~ t + 1',
                          data=combined).fit()
print(model_combined.summary2())

```

```

Results: Ordinary least squares
=====
Model: OLS Adj. R-squared: 0.870
Dependent Variable: t3 AIC: -28.1379
Date: 2023-05-03 21:16 BIC: -25.2064
No. Observations: 32 Log-Likelihood: 16.069
Df Model: 1 F-statistic: 208.3
Df Residuals: 30 Prob (F-statistic): 4.86e-15
R-squared: 0.874 Scale: 0.022876
=====
              Coef.      Std.Err.      t      P>|t|      [0.025      0.975]
-----
Intercept    0.6151      0.2527      2.4343    0.0211    0.0990    1.1311
t            0.8882      0.0615     14.4327    0.0000    0.7625    1.0139
=====
Omnibus:      1.072      Durbin-Watson:      0.667
Prob(Omnibus): 0.585      Jarque-Bera (JB):      0.883
Skew:         0.114      Prob(JB):      0.643
Kurtosis:     2.219      Condition No.:      41
=====

```

Solution 4.24 Load the data frame

```
df = mistat.load_data('CEMENT.csv')
```

(a) Regression of Y on X_1 is

```

modell = smf.ols('y ~ x1 + 1', data=df).fit()
print(modell.summary().tables[1])
r2 = modell.rsquared

```

```
print(f'R-sq: {r2:.3f}')

anova = sms.anova.anova_lm(model1)
print('Analysis of Variance\n', anova)
```

```
F = anova.F['x1']
SSE_1 = anova.sum_sq['Residual']
```

```
=====
              coef      std err          t      P>|t|      [0.025      0.975]
-----
Intercept    81.4793        4.927      16.536      0.000      70.634      92.324
x1           1.8687        0.526       3.550      0.005       0.710       3.027
=====
R-sq: 0.534
Analysis of Variance
              df      sum_sq      mean_sq          F      PR(>F)
x1              1.0  1450.076328  1450.076328  12.602518  0.004552
Residual      11.0  1265.686749   115.062432         NaN         NaN
```

- $R^2_{Y|(X_1)} = 0.534$.
- $SSE_1 = 1265.7$,
- $F = 12.60$ (In the 1st stage F is equal to the partial- F .)

(b) The regression of Y on X_1 and X_2 is

```
model2 = smf.ols('y ~ x1 + x2 + 1', data=df).fit()
r2 = model2.rsquared
print(model2.summary().tables[1])
print(f'R-sq: {r2:.3f}')
anova = sms.anova.anova_lm(model2)
print('Analysis of Variance\n', anova)
SEQ_SS_X2 = anova.sum_sq['x2']
SSE_2 = anova.sum_sq['Residual']
s2e2 = anova.mean_sq['Residual']
partialF = np.sum(anova.sum_sq) * (model2.rsquared - model1.rsquared) / s2e2

anova = sms.anova.anova_lm(model1, model2)
print('Comparing models\n', anova)
partialF = anova.F[1]
```

```
=====
              coef      std err          t      P>|t|      [0.025      0.975]
-----
Intercept    52.5773        2.286      22.998      0.000      47.483      57.671
x1           1.4683        0.121      12.105      0.000       1.198       1.739
x2           0.6623        0.046      14.442      0.000       0.560       0.764
=====
R-sq: 0.979
Analysis of Variance
              df      sum_sq      mean_sq          F      PR(>F)
x1              1.0  1450.076328  1450.076328  250.425571  2.088092e-08
x2              1.0  1207.782266  1207.782266  208.581823  5.028960e-08
Residual      10.0   57.904483    5.790448         NaN         NaN
Comparing models
      df_resid      ssr  df_diff  ss_diff          F      Pr(>F)
0          11.0  1265.686749      0.0         NaN         NaN         NaN
1          10.0   57.904483      1.0  1207.782266  208.581823  5.028960e-08
```

- $R^2_{Y|(X_1, X_2)} = 0.979$.
- $SSE_2 = 57.9$,

- $s_{e_2}^2 = 5.79$,
- $F = 12.60$
- Partial- $F = 208.582$

Notice that SEQ SS for $X_2 = 2716.9(0.974 - 0.529) = 1207.782$.

(c) The regression of Y on X_1 , X_2 , and X_3 is

```
model3 = smf.ols('y ~ x1 + x2 + x3 + 1', data=df).fit()
r2 = model3.rsquared
print(model3.summary().tables[1])
print(f'R-sq: {r2:.3f}')
anova = sms.anova.anova_lm(model3)
print('Analysis of Variance\n', anova)
SEQ_SS_X3 = anova.sum_sq['x3']
SSE_3 = anova.sum_sq['Residual']
s2e3 = anova.mean_sq['Residual']

anova = sms.anova.anova_lm(model2, model3)
print('Comparing models\n', anova)
partialF = anova.F[1]
```

	coef	std err	t	P> t	[0.025	0.975]
Intercept	48.1936	3.913	12.315	0.000	39.341	57.046
x1	1.6959	0.205	8.290	0.000	1.233	2.159
x2	0.6569	0.044	14.851	0.000	0.557	0.757
x3	0.2500	0.185	1.354	0.209	-0.168	0.668

```
R-sq: 0.982
Analysis of Variance
      df      sum_sq      mean_sq      F      PR(>F)
x1      1.0  1450.076328  1450.076328  271.264194  4.995767e-08
x2      1.0  1207.782266  1207.782266  225.938509  1.107893e-07
x3      1.0    9.793869    9.793869   1.832128  2.088895e-01
Residual 9.0  48.110614    5.345624      NaN      NaN
Comparing models
      df_resid      ssr      df_diff      ss_diff      F      Pr(>F)
0      10.0  57.904483      0.0      NaN      NaN      NaN
1       9.0  48.110614      1.0    9.793869   1.832128  0.208889
```

- $R_{Y|(X_1, X_2, X_3)}^2 = 0.982$.
- Partial- $F = 1.832$

The SEQ SS of X_3 is 9.79. The .95-quantile of $F[1, 9]$ is 5.117. Thus, the contribution of X_3 is not significant.

(d) The regression of Y on X_1 , X_2 , X_3 , and X_4 is

```
model4 = smf.ols('y ~ x1 + x2 + x3 + x4 + 1', data=df).fit()
r2 = model4.rsquared
print(model4.summary().tables[1])
print(f'R-sq: {r2:.3f}')
anova = sms.anova.anova_lm(model4)
print('Analysis of Variance\n', anova)
SEQ_SS_X4 = anova.sum_sq['x4']
SSE_4 = anova.sum_sq['Residual']
s2e4 = anova.mean_sq['Residual']

anova = sms.anova.anova_lm(model3, model4)
print('Comparing models\n', anova)
partialF = anova.F[1]
```

	coef	std err	t	P> t	[0.025	0.975]
Intercept	62.4054	70.071	0.891	0.399	-99.179	223.989
x1	1.5511	0.745	2.083	0.071	-0.166	3.269
x2	0.5102	0.724	0.705	0.501	-1.159	2.179
x3	0.1019	0.755	0.135	0.896	-1.638	1.842
x4	-0.1441	0.709	-0.203	0.844	-1.779	1.491

R-sq: 0.982
Analysis of Variance

	df	sum_sq	mean_sq	F	PR(>F)
x1	1.0	1450.076328	1450.076328	242.367918	2.887559e-07
x2	1.0	1207.782266	1207.782266	201.870528	5.863323e-07
x3	1.0	9.793869	9.793869	1.636962	2.366003e-01
x4	1.0	0.246975	0.246975	0.041280	8.440715e-01
Residual	8.0	47.863639	5.982955	NaN	NaN

Comparing models

	df_resid	ssr	df_diff	ss_diff	F	Pr(>F)
0	9.0	48.110614	0.0	NaN	NaN	NaN
1	8.0	47.863639	1.0	0.246975	0.04128	0.844071

- $R^2_{Y|(X_1, X_2, X_3)} = 0.982$.
- Partial- $F = 0.041$

The effect of X_4 is not significant.

Solution 4.25 Using the step-wise regression method from the `mistat` package, we get:

```
outcome = 'y'
all_vars = ['x1', 'x2', 'x3', 'x4']

included, model = mistat.stepwise_regression(outcome, all_vars, df)

formula = ' + '.join(included)
formula = f'{outcome} ~ 1 + {formula}'
print()
print('Final model')
print(formula)
print(model.params)
```

```
Step 1 add - (F: 22.80)  x4
Step 2 add - (F: 108.22) x1 x4
Step 3 add - (F: 5.03)  x1 x2 x4

Final model
y ~ 1 + x2 + x4 + x1
Intercept    71.648307
x2           0.416110
x4          -0.236540
x1           1.451938
dtype: float64
```

Solution 4.26 Build the regression model using `statsmodels`.

```
car = mistat.load_data('CAR')
car_3 = car[car['origin'] == 3]
print('Full dataset shape', car.shape)
print('Origin 3 dataset shape', car_3.shape)
model = smf.ols(formula='mpg ~ hp + 1', data=car_3).fit()
print(model.summary2())
```

```

Full dataset shape (109, 5)
Origin 3 dataset shape (37, 5)
Results: Ordinary least squares
=====
Model:          OLS                Adj. R-squared:    0.400
Dependent Variable: mpg          AIC:                195.6458
Date:           2023-05-03 21:16  BIC:                198.8676
No. Observations: 37             Log-Likelihood:    -95.823
Df Model:        1                F-statistic:        25.00
Df Residuals:    35               Prob (F-statistic): 1.61e-05
R-squared:        0.417           Scale:                10.994
-----
              Coef.    Std.Err.    t      P>|t|    [0.025    0.975]
-----
Intercept    31.8328    1.8282   17.4117  0.0000   28.1213   35.5444
hp           -0.0799    0.0160  -4.9996  0.0000  -0.1123  -0.0474
-----
Omnibus:      7.375           Durbin-Watson:      1.688
Prob(Omnibus): 0.025           Jarque-Bera (JB):    6.684
Skew:         -0.675          Prob(JB):            0.035
Kurtosis:     4.584           Condition No.:       384
=====

```

Compute the additional properties

```

influence = model.get_influence()
df = pd.DataFrame({
    'hp': car_3['hp'],
    'mpg': car_3['mpg'],
    'resi': model.resid,
    'sres': influence.resid_studentized_internal,
    'hi': influence.hat_matrix_diag,
    'D': influence.cooks_distance[0],
})
print(df.round(4))

```

```

   hp  mpg  resi  sres  hi  D
0  118   25  2.5936  0.7937  0.0288  0.0093
1  161   18 -0.9714 -0.3070  0.0893  0.0046
51  55   17 -10.4392 -3.3101  0.0953  0.5770
52  98   23 -1.0041 -0.3075  0.0299  0.0015
53  92   27  2.5166  0.7722  0.0339  0.0105
54  92   29  4.5166  1.3859  0.0339  0.0337
55  104  20 -3.5248 -1.0781  0.0277  0.0165
56  68   27  0.5993  0.1871  0.0665  0.0012
57  70   31  4.7591  1.4826  0.0627  0.0736
58  110  20 -3.0455 -0.9312  0.0270  0.0120
62  121  19 -3.1668 -0.9699  0.0303  0.0147
63  82   24 -1.2823 -0.3956  0.0442  0.0036
64  110  22 -1.0455 -0.3197  0.0270  0.0014
65  158  19 -0.2110 -0.0664  0.0823  0.0002
71  92   26  1.5166  0.4654  0.0339  0.0038
72  102  22 -1.6846 -0.5154  0.0282  0.0039
73  81   27  1.6378  0.5056  0.0455  0.0061
74  142  18 -2.4892 -0.7710  0.0520  0.0163
75  107  18 -5.2852 -1.6161  0.0271  0.0364
76  160  19 -0.0513 -0.0162  0.0869  0.0000
77  90   24 -0.6432 -0.1975  0.0356  0.0007
78  90   26  1.3568  0.4167  0.0356  0.0032
79  97   21 -3.0840 -0.9446  0.0305  0.0140
80  106  18 -5.3650 -1.6406  0.0273  0.0377
81  140  20 -0.6489 -0.2007  0.0490  0.0010
82  165  18 -0.6518 -0.2071  0.0993  0.0024
94  66   34  7.4396  2.3272  0.0704  0.2051
95  97   23 -1.0840 -0.3320  0.0305  0.0017
96  100  25  1.1557  0.3537  0.0290  0.0019

```

97	115	24	1.3539	0.4141	0.0278	0.0025
98	115	26	3.3539	1.0259	0.0278	0.0151
99	90	27	2.3568	0.7238	0.0356	0.0097
100	190	19	2.3453	0.7805	0.1786	0.0662
101	115	25	2.3539	0.7200	0.0278	0.0074
102	200	18	2.1441	0.7315	0.2184	0.0748
103	78	28	2.3982	0.7419	0.0497	0.0144
108	64	28	1.2798	0.4012	0.0745	0.0065

Notice that points 51 and 94 have residuals with large magnitude (-10.4, 7.4). Points 51 and 94 have also the largest Cook's distance (0.58, 0.21) Points 100 and 102 have high HI values (leverage; 0.18, 0.22).

Solution 4.27 This solution uses version 2 of the Piston simulator.

Run piston simulation for different piston weights and visualize variation of times (see Fig. 4.7).

{fig:anovaWeightPiston}

```
np.random.seed(1)
settings = {'s': 0.005, 'v0': 0.002, 'k': 1000, 'p0': 90_000,
           't': 290, 't0': 340}
results = []
n_simulation = 5
for m in [30, 40, 50, 60]:
    simulator = mistat.PistonSimulator(m=m, n_simulation=n_simulation,
                                       **settings)
    sim_result = simulator.simulate()
    results.extend([m, s] for s in sim_result['seconds'])
results = pd.DataFrame(results, columns=['m', 'seconds'])

group_std = results.groupby('m').std()
pooled_std = np.sqrt(np.sum(group_std**2) / len(group_std)) [0]
print('Pooled standard deviation', pooled_std)

group_mean = results.groupby('m').mean()
ax = results.plot.scatter(x='m', y='seconds', color='black')
ax.errorbar(group_mean.index, results.groupby('m').mean().values.flatten(),
            yerr=[pooled_std] * 4, color='grey')
plt.show()
```

Pooled standard deviation 0.013088056556052113

Perform ANOVA of data.

```
model = smf.ols(formula='seconds ~ C(m)', data=results).fit()
aov_table = sm.stats.anova_lm(model)
aov_table
```

	df	sum_sq	mean_sq	F	PR(>F)
C(m)	3.0	0.000333	0.000111	0.648836	0.595055
Residual	16.0	0.002741	0.000171	NaN	NaN

We see that the differences between the sample means are not significant in spite of the apparent upward trend in cycle times.

Solution 4.28 Prepare dataset and visualize distributions (see Fig. 4.8).

{fig:boxplotIntegratedCircuits}

```
df = pd.DataFrame([
    [2.58, 2.62, 2.22],
    [2.48, 2.77, 1.73],
```

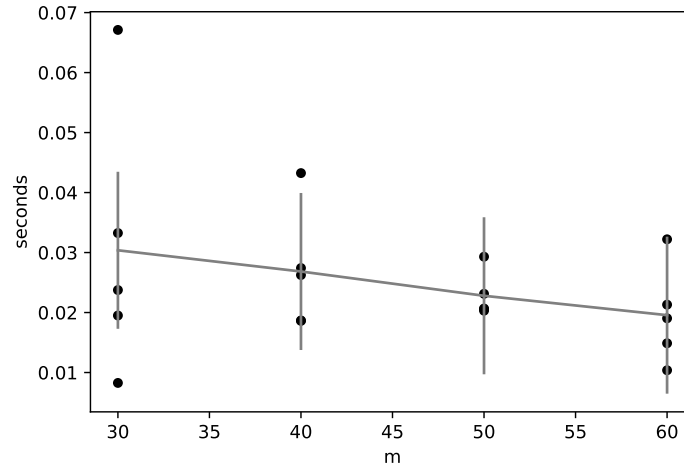


Fig. 4.7 ANOVA of effect of changing weight in piston simulation

(fig:anovaWeightPiston)

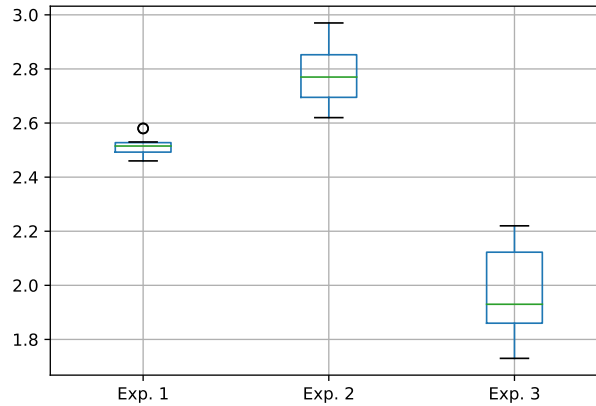


Fig. 4.8 Box plot of pre-etch line width from integrated circuits fabrication process

(fig:boxplotIntegratedCircuits)

```
[2.52, 2.69, 2.00],
[2.50, 2.80, 1.86],
[2.53, 2.87, 2.04],
[2.46, 2.67, 2.15],
[2.52, 2.71, 2.18],
[2.49, 2.77, 1.86],
[2.58, 2.87, 1.84],
[2.51, 2.97, 1.86]
], columns=['Exp. 1', 'Exp. 2', 'Exp. 3'])
df.boxplot()

# Convert data frame to long format using melt
df = df.melt(var_name='Experiment', value_name='mu')
```

Analysis using ANOVA:

```
model = smf.ols(formula='mu ~ C(Experiment)', data=df).fit()
aov_table = sm.stats.anova_lm(model)
aov_table
```

	df	sum_sq	mean_sq	F	PR(>F)
C(Experiment)	2.0	3.336327	1.668163	120.917098	3.352509e-14
Residual	27.0	0.372490	0.013796	NaN	NaN

The difference in the experiments is significant.

Bootstrap test:

```
experiment = df['Experiment']
mu = df['mu']
def onewayTest(x, verbose=False):
    df = pd.DataFrame({
        'value': x,
        'variable': experiment,
    })
    aov = pg.anova(dv='value', between='variable', data=df)
    return aov['F'].values[0]

B = pg.compute_bootci(mu, func=onewayTest, n_boot=1000,
                      seed=1, return_dist=True)

Bt0 = onewayTest(mu)
print('Bt0', Bt0)
print('ratio', sum(B[1] >= Bt0)/len(B[1]))
```

```
| Bt0 120.91709844559576
| ratio 0.0
```

The bootstrap test also shows that the difference in means is significant.

Solution 4.29 Create dataset and visualize distribution (see Fig. 4.9).

{fig:filmSpeedData}

```
df = pd.DataFrame({
    'Batch A': [103, 107, 104, 102, 95, 91, 107, 99, 105, 105],
    'Batch B': [104, 103, 106, 103, 107, 108, 104, 105, 105, 97],
})
fig, ax = plt.subplots()
bplot1 = ax.boxplot(df, labels=df.columns)
plt.show()

# the following code failed at some point using pandas
# df.boxplot()
# plt.show()
```

(i) Randomization test (see Sect. 3.13.2)

{sec:randomization-test}

```
dist = mistat.randomizationTest(df['Batch A'], df['Batch B'], np.mean,
                               aggregate_stats=lambda x: x[0] - x[1],
                               n_boot=10000, seed=1)

# ax = sns.distplot(dist)
# ax.axvline(np.mean(df['Batch A']) - np.mean(df['Batch B']))
```

```
| Original stat is -2.400000
| Original stat is at quantile 1062 of 10001 (10.62%)
| Distribution of bootstrap samples:
| min: -5.40, median: 0.00, max: 5.60
```

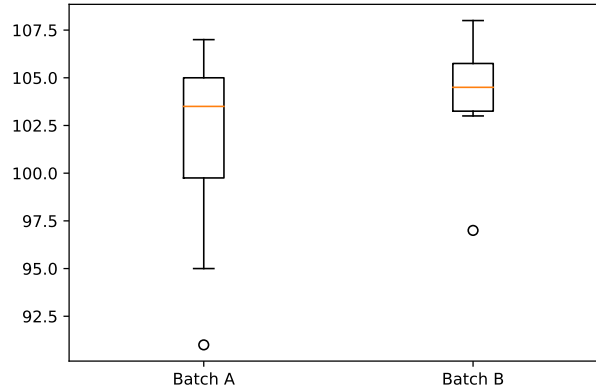


Fig. 4.9 Box plot of film speed data

(fig:filmSpeedData)

The randomization test gave a P value of 0.106. The difference between the means is not significant.

(ii)

```
# Convert data frame to long format using melt
df = df.melt(var_name='Batch', value_name='film_speed')

model = smf.ols(formula='film_speed ~ C(Batch)', data=df).fit()
aov_table = sm.stats.anova_lm(model)
aov_table
```

	df	sum_sq	mean_sq	F	PR(>F)
C(Batch)	1.0	28.8	28.800000	1.555822	0.228263
Residual	18.0	333.2	18.511111	NaN	NaN

The ANOVA also shows no significant difference in the means. The P value is 0.228. Remember that the F -test in the ANOVA is based on the assumption of normality and equal variances. The randomization test is nonparametric.

Solution 4.30 Define function that calculates the statistic and execute bootstrap.

```
def func_stats(x):
    m = pd.Series(x).groupby(df['Experiment']).agg(['mean', 'count'])
    top = np.sum(m['count'] * m['mean'] ** 2) - len(x) * np.mean(x) ** 2
    return top / np.std(x) ** 2

Bt = []
mu = list(df['mu'])
for _ in range(1000):
    mu_star = random.sample(mu, len(mu))
    Bt.append(func_stats(mu_star))

Bt0 = func_stats(mu)
print('Bt0', Bt0)
print('ratio', sum(Bt >= Bt0) / len(Bt))
```

```
Bt0 26.986990459670288
ratio 0.0
```

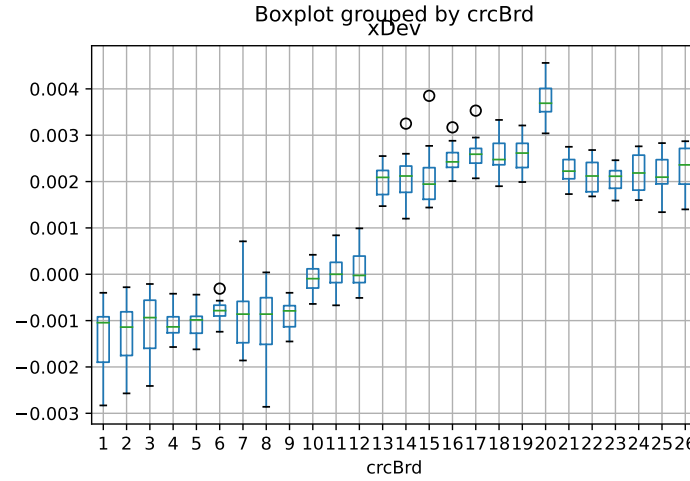


Fig. 4.10 Box plot visualisation of xDev distribution by crcBrd for the PLACE dataset

{fig:boxplotXdevCrcBrdPlace}

The result demonstrates that the differences between the results is significant.

Solution 4.31 Load the data and visualize the distributions (see Fig. 4.10).

{fig:boxplotXdevCrcBrdPlace}

```
place = mistat.load_data('PLACE')
place.boxplot('xDev', by='crcBrd')
plt.show()
```

(a) ANOVA for the fulldataset

```
model = smf.ols(formula='xDev ~ C(crcBrd)', data=place).fit()
aov_table = sm.stats.anova_lm(model)
aov_table
```

	df	sum_sq	mean_sq	F	PR(>F)
C(crcBrd)	25.0	0.001128	4.512471e-05	203.292511	2.009252e-206
Residual	390.0	0.000087	2.219694e-07	NaN	NaN

(b) There seem to be four homogeneous groups: $G_1 = \{1, 2, \dots, 9\}$, $G_2 = \{10, 11, 12\}$, $G_3 = \{13, \dots, 19, 21, \dots, 26\}$, $G_4 = \{20\}$.

In multiple comparisons we use the Scheffé coefficient $S_{.05} = (25 \times F_{.95}[25, 390])^{1/2} = (25 \times 1.534)^{1/2} = 6.193$. The group means and standard errors are:

```
G1 = [1, 2, 3, 4, 5, 6, 7, 8, 9]
G2 = [10, 11, 12]
G3 = [13, 14, 15, 16, 17, 18, 19, 21, 22, 23, 24, 25, 26]
G4 = [20]
place['group'] = 'G1'
place.loc[place['crcBrd'].isin(G2), 'group'] = 'G2'
place.loc[place['crcBrd'].isin(G3), 'group'] = 'G3'
place.loc[place['crcBrd'].isin(G4), 'group'] = 'G4'
```

```

statistics = place['xDev'].groupby(place['group']).agg(['mean', 'sem', 'count'])
statistics = statistics.sort_values(['mean'], ascending=False)
print(statistics.round(8))
statistics['Diff'] = 0
n = len(statistics)
print(statistics['mean'][:-1].values - statistics['mean'][1:].values)
print(statistics['mean'][:-(n-1)].values - statistics['mean'][1:].values)
statistics.loc[1:, 'Diff'] = (statistics['mean'][:-1].values -
                             statistics['mean'][1:].values)

statistics['CR'] = 6.193 * statistics['Diff']
print(statistics.round(8))
# 0.001510 0.0022614 0.0010683
# 0.000757 0.000467 0.000486

sem = statistics['sem'].values
sem = sem**2
sem = np.sqrt(sem[:-1] + sem[1:])
print(sem * 6.193)
print(757/644, 467/387, 486/459)

```

	mean	sem	count
group			
G4	0.003778	0.000100	16
G3	0.002268	0.000030	208
G2	0.000006	0.000055	48
G1	-0.001062	0.000050	144
	[0.00151029 0.00226138 0.00106826]		
	[0.00151029 0.00226138 0.00106826]		

```

-----TypeError
Traceback (most recent call last):Cell In[1], line 17
    15 print(statistics['mean'][:-1].values -
statistics['mean'][1:].values)
    16 print(statistics['mean'][:-(n-1)].values -
statistics['mean'][1:].values)
--> 17 statistics.loc[1:, 'Diff'] = (statistics['mean'][:-1].values -
    18                               statistics['mean'][1:].values)
    19 statistics['CR'] = 6.193 * statistics['Diff']
    20 print(statistics.round(8))
File /usr/local/lib/python3.9/site-
packages/pandas/core/indexing.py:845, in
_LocationIndexer.__setitem__(self, key, value)
    843 else:
    844     key = com.apply_if_callable(key, self.obj)
--> 845 indexer = self._get_setitem_indexer(key)
    846 self._has_valid_setitem_indexer(key)
    848 iloc = self if self.name == "iloc" else self.obj.iloc
File /usr/local/lib/python3.9/site-
packages/pandas/core/indexing.py:710, in
_LocationIndexer._get_setitem_indexer(self, key)
    707 if isinstance(key, tuple):
    708     with suppress(IndexingError):
    709         # suppress "Too many indexers"
--> 710         return self._convert_tuple(key)
    712 if isinstance(key, range):
    713     # GH#45479 test_loc_setitem_range_key
    714     key = list(key)
File /usr/local/lib/python3.9/site-
packages/pandas/core/indexing.py:927, in
_LocationIndexer._convert_tuple(self, key)
    923 @final
    924 def _convert_tuple(self, key: tuple) -> tuple:
    925     # Note: we assume _tupleize_axis_indexer has been called,
if necessary.
    926     self._validate_key_length(key)
--> 927     keyidx = [self._convert_to_indexer(k, axis=i) for i, k in

```

```

enumerate(key)]
    928     return tuple(keyidx)
File /usr/local/lib/python3.9/site-
packages/pandas/core/indexing.py:927, in <listcomp>(.0)
    923 @final
    924 def _convert_tuple(self, key: tuple) -> tuple:
    925     # Note: we assume _tupleize_axis_indexer has been called,
if necessary.
    926     self._validate_key_length(key)
--> 927     keyidx = [self._convert_to_indexer(k, axis=i) for i, k in
enumerate(key)]
    928     return tuple(keyidx)
File /usr/local/lib/python3.9/site-
packages/pandas/core/indexing.py:1382, in
_LocIndexer._convert_to_indexer(self, key, axis)
    1379 labels = self.obj._get_axis(axis)
    1381 if isinstance(key, slice):
-> 1382     return labels._convert_slice_indexer(key, kind="loc")
    1384 if (
    1385     isinstance(key, tuple)
    1386     and not isinstance(labels, MultiIndex)
    1387     and self.ndim < 2
    1388     and len(key) > 1
    1389 ):
    1390     raise IndexingError("Too many indexers")
File /usr/local/lib/python3.9/site-
packages/pandas/core/indexes/base.py:4125, in
Index._convert_slice_indexer(self, key, kind)
    4122 elif is_positional:
    4123     if kind == "loc":
    4124         # GH#16121, GH#24612, GH#31810
-> 4125         raise TypeError(
    4126             "Slicing a positional slice with .loc is not
allowed, "
    4127             "Use .loc with labels or .iloc with positions
instead.",
    4128         )
    4129     indexer = key
    4130 else:
TypeError: Slicing a positional slice with .loc is not allowed, Use
.loc with labels or .iloc with positions instead.

```

The differences between the means of the groups are all significant.

```

from statsmodels.stats.multicomp import pairwise_tukeyhsd
m_comp = pairwise_tukeyhsd(endog=place['xDev'], groups=place['group'],
                           alpha=0.05)
print(m_comp)

```

```

Multiple Comparison of Means - Tukey HSD, FWER=0.05
=====
group1 group2 meandiff p-adj lower  upper  reject
-----
    G1     G2   0.0011   0.0 0.0009 0.0013   True
    G1     G3   0.0033   0.0 0.0032 0.0035   True
    G1     G4   0.0048   0.0 0.0045 0.0052   True
    G2     G3   0.0023   0.0 0.0021 0.0025   True
    G2     G4   0.0038   0.0 0.0034 0.0041   True
    G3     G4   0.0015   0.0 0.0012 0.0018   True
=====

```

Solution 4.32

```

frame({
    'US': [33, 25],
    'Europe': [7, 7],

```

```

        'Asia': [26, 11],
    })

print(df)

col_sums = df.sum(axis=0)
row_sums = df.sum(axis=1)
total = df.to_numpy().sum()

expected_frequencies = np.outer(row_sums, col_sums) / total

chi2 = (df - expected_frequencies) ** 2 / expected_frequencies
chi2 = chi2.to_numpy().sum()
print(f'chi2: {chi2:.3f}')
print(f'p-value: {1 - stats.chi2.cdf(chi2, 2):.3f}')
```

```

    US  Europe  Asia
0   33       7   26
1   25       7   11
chi2: 2.440
p-value: 0.295
```

The chi-square test statistic is $X^2 = 2.440$ with d.f. = 2 and P value = 0.295. The null hypothesis that the number of cylinders a car has is independent of the origin of the car is not rejected.

We can also use the `scipy` function `chi2_contingency`.

```

chi2 = stats.chi2_contingency(df)
print(f'chi2-statistic: {chi2[0]:.3f}')
print(f'p-value: {chi2[1]:.3f}')
print(f'd.f.: {chi2[2]}')
```

```

chi2-statistic: 2.440
p-value: 0.295
d.f.: 2
```

Solution 4.33 In Python:

```

car = mistat.load_data('CAR')
binned_car = pd.DataFrame({
    'turn': pd.cut(car['turn'], bins=[27, 30.6, 34.2, 37.8, 45]), #np.arange(27, 50, 3.6)),
    'mpg': pd.cut(car['mpg'], bins=[12, 18, 24, 100]),
})
freqDist = pd.crosstab(binned_car['mpg'], binned_car['turn'])
print(freqDist)

chi2 = stats.chi2_contingency(freqDist)
print(f'chi2-statistic: {chi2[0]:.3f}')
print(f'p-value: {chi2[1]:.3f}')
print(f'd.f.: {chi2[2]}')
```

```

turn      (27.0, 30.6]  (30.6, 34.2]  (34.2, 37.8]  (37.8, 45.0]
mpg
(12, 18]              2              4             10             15
(18, 24]              0             12             26             15
(24, 100]             4             15              6              0
chi2-statistic: 34.990
p-value: 0.000
d.f.: 6
```

The dependence between turn diameter and miles per gallon is significant.

Solution 4.34 In Python:

```

question_13 = pd.DataFrame({
    '1': [0,0,0,1,0],
    '2': [1,0,2,0,0],
    '3': [1,2,6,5,1],
    '4': [2,1,10,23,13],
    '5': [0,1,1,15,100],
}, index = ['1', '2', '3', '4', '5']).transpose()
question_23 = pd.DataFrame({
    '1': [1,0,0,3,1],
    '2': [2,0,1,0,0],
    '3': [0,4,2,3,0],
    '4': [1,1,10,7,5],
    '5': [0,0,1,30,134],
}, index = ['1', '2', '3', '4', '5']).transpose()

chi2_13 = stats.chi2_contingency(question_13)
chi2_23 = stats.chi2_contingency(question_23)

msc_13 = chi2_13[0] / question_13.to_numpy().sum()
tschuprov_13 = np.sqrt(msc_13 / (2 * 2)) # (4 * 4)
cramer_13 = np.sqrt(msc_13 / 2) # min(4, 4)

msc_23 = chi2_23[0] / question_23.to_numpy().sum()
tschuprov_23 = np.sqrt(msc_23 / 4) # (4 * 4)
cramer_23 = np.sqrt(msc_23 / 2) # min(4, 4)

print('Question 1 vs 3')
print(f'  Mean squared contingency : {msc_13:.3f}')
print(f'  Tschuprov : {tschuprov_13:.3f}')
print(f"  Cramer's index : {cramer_13:.3f}")
print('Question 2 vs 3')
print(f'  Mean squared contingency : {msc_23:.3f}')
print(f'  Tschuprov : {tschuprov_23:.3f}')
print(f"  Cramer's index : {cramer_23:.3f}")

```

```

Question 1 vs 3
  Mean squared contingency : 0.629
  Tschuprov : 0.397
  Cramer's index : 0.561
Question 2 vs 3
  Mean squared contingency : 1.137
  Tschuprov : 0.533
  Cramer's index : 0.754

```


Chapter 5

Sampling for Estimation of Finite Population Quantities

Import required modules and define required functions

```
import random
import numpy as np
import pandas as pd
import pingouin as pg
from scipy import stats
import matplotlib.pyplot as plt
import mistat
```

Solution 5.1 Define the binary random variables

$$I_{ij} = \begin{cases} 1, & \text{if the } j\text{-th element is selected at the } i\text{-th sampling} \\ 0, & \text{otherwise.} \end{cases}$$

The random variables X_1, \dots, X_n are given by $X_i = \sum_{j=1}^N x_j I_{ij}, i = 1, \dots, n$. Since sampling is RSWR, $\Pr\{X_i = x_j\} = \frac{1}{N}$ for all $i = 1, \dots, n$ and $j = 1, \dots, N$. Hence, $\Pr\{X_i \leq x\} = F_N(x)$ for all x , and all $i = 1, \dots, n$. Moreover, by definition of RSWR, the vectors $\mathbf{I}_i = (I_{i1}, \dots, I_{iN}), i = 1, \dots, n$ are mutually independent. Therefore X_1, \dots, X_n are i.i.d., having a common c.d.f. $F_N(x)$.

Solution 5.2 In continuation of the previous exercise, $E\{X_i\} = \frac{1}{N} \sum_{i=1}^N x_i = \mu_N$. Therefore, by the weak law of large numbers, $\lim_{n \rightarrow \infty} P\{|\bar{X}_n - \mu_N| < \epsilon\} = 1$.

Solution 5.3 By the CLT ($0 < \sigma_N^2 < \infty$),

$$\Pr\{\sqrt{n}|\bar{X}_n - \mu_N| < \delta\} \approx 2\Phi\left(\frac{\delta}{\sigma_N}\right) - 1,$$

as $n \rightarrow \infty$.

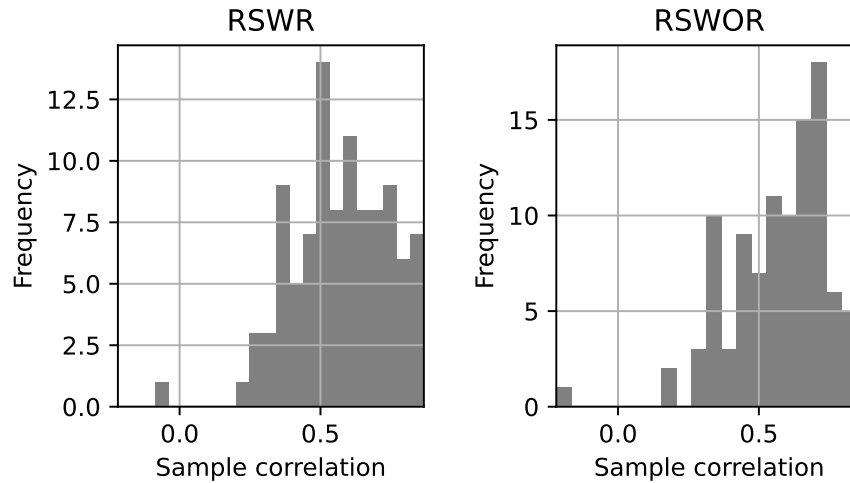


Fig. 5.1 Distribution of correlation between $xDev$ and $yDev$ for sampling with and without distribution. (fig:exDistCorrPlace)

Solution 5.4 We create samples of size $k = 20$ with and without replacement, determine the correlation coefficient and finally create the two histograms (see Fig. 5.1). (fig:exDistCorrPlace)

```

random.seed(1)
place = mistat.load_data('PLACE')

# calculate correlation coefficient based on a sample of rows
def stat_func(idx):
    return stats.pearsonr(place['xDev'][idx], place['yDev'][idx])[0]

rswr = []
rswor = []
idx = list(range(len(place)))
for _ in range(100):
    rswr.append(stat_func(random.choices(idx, k=20)))
    rswor.append(stat_func(random.sample(idx, k=20)))

corr_range = (min(*rswr, *rswor), max(*rswr, *rswor))

def makeHistogram(title, ax, data, xrange):
    ax = pd.Series(data).hist(color='grey', ax=ax, bins=20)
    ax.set_title(title)
    ax.set_xlabel('Sample correlation')
    ax.set_ylabel('Frequency')
    ax.set_xlim(*xrange)

fig, axes = plt.subplots(figsize=[5, 3], ncols=2)
makeHistogram('RSWR', axes[0], rswr, corr_range)
makeHistogram('RSWOR', axes[1], rswor, corr_range)
plt.tight_layout()
plt.show()

```

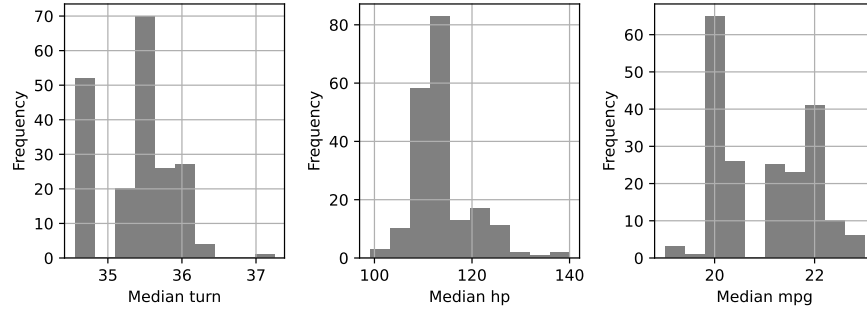


Fig. 5.2 Distribution of median of turn-diameter, horsepower, and mpg of the CAR dataset using random sampling without replacement.

(fig:exDistMedianCAR)

(fig:exDistMedianCAR)

Solution 5.5 The Python code creates the histograms shown in Fig. 5.2.

```

random.seed(1)
car = mistat.load_data('CAR')
columns = ['turn', 'hp', 'mpg']

# calculate correlation coefficient based on a sample of rows
def stat_func(idx):
    sample = car[columns].loc[idx,]
    return sample.median()

idx = list(range(len(car)))
result = []
for _ in range(200):
    result.append(stat_func(random.sample(idx, k=50)))
result = pd.DataFrame(result)

fig, axes = plt.subplots(figsize=[8, 3], ncols=3)
for ax, column in zip(axes, columns):
    result[column].hist(color='grey', ax=ax)
    ax.set_xlabel(f'Median {column}')
    ax.set_ylabel('Frequency')
plt.tight_layout()
plt.show()

```

Solution 5.6 For RSWOR,

$$\text{S.E.}\{\bar{X}_i\} = \frac{\sigma}{\sqrt{n}} \left(1 - \frac{n-1}{N-1}\right)^{1/2}$$

Equating the standard error to δ we get $n_1 = 30$, $n_2 = 116$, $n_3 = 84$.

Solution 5.7 The required sample size is a solution of the equation

$$0.002 = 2 \cdot 1.96 \cdot \sqrt{\frac{P(1-P)}{n} \left(1 - \frac{n-1}{N}\right)}. \text{ The solution is } n = 1611.$$

Solution 5.8 The following are Python commands to estimate the mean of all $N = 416$ x -dev values by stratified sampling with proportional allocation. The total sample

size is $n = 200$ and the weights are $W_1 = 0.385$, $W_2 = 0.115$, $W_3 = 0.5$. Thus, $n_1 = 77$, $n_2 = 23$, and $n_3 = 100$.

```
# load dataset and split into strata
place = mistat.load_data('PLACE')
strata_1 = list(place['xDev'][:160])
strata_2 = list(place['xDev'][160:208])
strata_3 = list(place['xDev'][208:])
N = len(place)
w_1 = 0.385
w_2 = 0.115
w_3 = 0.5
n_1 = int(w_1 * 200)
n_2 = int(w_2 * 200)
n_3 = int(w_3 * 200)

sample_means = []
for _ in range(500):
    m_1 = np.mean(random.sample(strata_1, k=n_1))
    m_2 = np.mean(random.sample(strata_2, k=n_2))
    m_3 = np.mean(random.sample(strata_3, k=n_3))
    sample_means.append(w_1*m_1 + w_2*m_2 + w_3*m_3)
std_dev_sample_means = np.std(sample_means)
print(std_dev_sample_means)
print(stats.sem(place['xDev'], ddof=0))
```

```
| 3.442839155174113e-05
| 8.377967188860638e-05
```

The standard deviation of the estimated means is an estimate of S.E. $(\hat{\mu}_N)$. The true value of this S.E. is 0.000034442.

Solution 5.9 $L(n_1, \dots, n_k; \lambda) = \sum_{i=1}^k W_i^2 \frac{\tilde{\sigma}_{N_i}^2}{n_i} - \lambda \left(n - \sum_{i=1}^k n_i \right)$. Partial differentiation of L w.r.t. n_1, \dots, n_k and λ and equating the result to zero yields the following equations:

$$\frac{W_i^2 \tilde{\sigma}_{N_i}^2}{n_i^2} = \lambda, \quad i = 1, \dots, k$$

$$\sum_{i=1}^k n_i = n.$$

Equivalently, $n_i = \frac{1}{\sqrt{\lambda}} W_i \tilde{\sigma}_{N_i}$, for $i = 1, \dots, k$ and $n = \frac{1}{\sqrt{\lambda}} \sum_{i=1}^k W_i \tilde{\sigma}_{N_i}$. Thus

$$n_i^0 = n \frac{W_i \tilde{\sigma}_{N_i}}{\sum_{j=1}^k W_j \tilde{\sigma}_{N_j}}, \quad i = 1, \dots, k.$$

Solution 5.10 The prediction model is $y_i = \beta + e_i$, $i = 1, \dots, N$, $E\{e_i\} = 0$, $V\{e_i\} = \sigma^2$, $\text{cov}(e_i, e_j) = 0$ for all $i \neq j$.

$$\begin{aligned}
E\{\bar{Y}_n - \mu_N\} &= E\left\{\beta + \frac{1}{n} \sum_{i=1}^N I_i e_i - \beta - \frac{1}{N} \sum_{i=1}^N e_i\right\} \\
&= \frac{1}{n} \sum_{i=1}^N E\{I_i e_i\},
\end{aligned}$$

where $I_i = \begin{cases} 1, & \text{if } i\text{-th population element is sampled} \\ 0, & \text{otherwise.} \end{cases}$

Notice that I_1, \dots, I_N are independent of e_1, \dots, e_N . Hence, $E\{I_i e_i\} = 0$ for all $i = 1, \dots, N$. This proves that \bar{Y}_n is prediction unbiased, irrespective of the sample strategy. The prediction MSE of \bar{Y}_n is

$$\begin{aligned}
PMSE\{\bar{Y}_n\} &= E\{(\bar{Y}_n - \mu_N)^2\} \\
&= V\left\{\frac{1}{n} \sum_{i=1}^N I_i e_i - \frac{1}{N} \sum_{i=1}^N e_i\right\} \\
&= V\left\{\left(\frac{1}{n} - \frac{1}{N}\right) \sum_{i=1}^N I_i e_i - \frac{1}{N} \sum_{i=1}^N (1 - I_i) e_i\right\}.
\end{aligned}$$

Let \mathbf{s} denote the set of units in the sample. Then

$$\begin{aligned}
PMSE\{\bar{Y}_n \mid \mathbf{s}\} &= \frac{\sigma^2}{n} \left(1 - \frac{n}{N}\right)^2 + \frac{1}{N} \left(1 - \frac{n}{N}\right) \sigma^2 \\
&= \frac{\sigma^2}{n} \left(1 - \frac{n}{N}\right).
\end{aligned}$$

Notice that $PMSE\{\bar{Y}_n \mid \mathbf{s}\}$ is independent of \mathbf{s} , and is equal for all samples.

Solution 5.11 The model is $y_i = \beta_0 + \beta_1 x_i + e_i$, $i = 1, \dots, N$. $E\{e_i\} = 0$, $V\{e_i\} = \sigma^2 x_i$, $i = 1, \dots, N$. Given a sample $\{(X_1, Y_1), \dots, (X_n, Y_n)\}$, we estimate β_0 and β_1 by the weighted LSE because the variances of y_i depend on x_i , $i = 1, \dots, N$. These weighted LSE values $\hat{\beta}_0$ and $\hat{\beta}_1$ minimizing $Q = \sum_{i=1}^N \frac{1}{X_i} (Y_i - \beta_0 - \beta_1 X_i)^2$, are given by

$$\hat{\beta}_1 = \frac{\bar{Y}_n \cdot \frac{1}{n} \sum_{i=1}^N \frac{1}{\bar{X}_i} - \frac{1}{n} \sum_{i=1}^N \frac{Y_i}{\bar{X}_i}}{\bar{X}_n \cdot \frac{1}{n} \sum_{i=1}^N \frac{1}{\bar{X}_i} - 1} \quad \text{and} \quad \hat{\beta}_0 = \frac{1}{\sum_{i=1}^N \frac{1}{\bar{X}_i}} \left(\sum_{i=1}^N \frac{Y_i}{\bar{X}_i} - n \hat{\beta}_1 \right).$$

It is straightforward to show that $E\{\hat{\beta}_1\} = \beta_1$ and $E\{\hat{\beta}_0\} = \beta_0$. Thus, an unbiased predictor of μ_N is $\hat{\mu}_N = \hat{\beta}_0 + \hat{\beta}_1 \bar{x}_N$.

Solution 5.12 The predictor \hat{Y}_{RA} can be written as $\hat{Y}_{RA} = \bar{x}_N \cdot \frac{1}{n} \sum_{i=1}^N I_i \frac{y_i}{x_i}$, where

$$I_i = \begin{cases} 1, & \text{if } i\text{-th population element is in the sample } \mathbf{s} \\ 0, & \text{otherwise.} \end{cases}$$

Recall that $y_i = \beta x_i + e_i$, $i = 1, \dots, N$, and that for any sampling strategy, e_1, \dots, e_N are independent of I_1, \dots, I_N . Hence, since $\sum_{i=1}^N I_i = n$,

$$\begin{aligned} E\{\hat{Y}_{RA}\} &= \bar{x}_N \cdot \frac{1}{n} \sum_{i=1}^N E\left\{I_i \frac{\beta x_i + e_i}{x_i}\right\} \\ &= \bar{x}_N \left(\beta + \frac{1}{n} \sum_{i=1}^N E\left\{I_i \frac{e_i}{x_i}\right\} \right) \\ &= \bar{x}_N \beta, \end{aligned}$$

because $E\left\{I_i \frac{e_i}{x_i}\right\} = E\left\{\frac{I_i}{x_i}\right\} E\{e_i\} = 0$, $i = 1, \dots, N$. Thus, $E\{\hat{Y}_{RA} - \mu_N\} = 0$ and \hat{Y}_{RA} is an unbiased predictor.

$$\begin{aligned} PMSE\{\hat{Y}_{RA}\} &= E\left\{\left(\frac{\bar{x}_N}{n} \sum_{i=1}^N I_i \frac{e_i}{x_i} - \frac{1}{N} \sum_{i=1}^N e_i\right)^2\right\} \\ &= V\left\{\sum_{i \in \mathbf{s}_n} \left(\frac{\bar{x}_N}{n x_i} - \frac{1}{N}\right) e_i - \sum_{i' \in \mathbf{r}_n} \frac{e_{i'}}{N}\right\} \end{aligned}$$

where \mathbf{s}_n is the set of elements in the sample and \mathbf{r}_n is the set of elements in \mathcal{P} but not in \mathbf{s}_n , $\mathbf{r}_n = \mathcal{P} - \mathbf{s}_n$. Since e_1, \dots, e_N are uncorrelated,

$$\begin{aligned} PMSE\{\hat{Y}_{RA} \mid \mathbf{s}_n\} &= \sigma^2 \sum_{i \in \mathbf{s}_n} \left(\frac{\bar{x}_N}{n x_i} - \frac{1}{N}\right)^2 + \sigma^2 \frac{N-n}{N^2} \\ &= \frac{\sigma^2}{N^2} \left[(N-n) + \sum_{i \in \mathbf{s}_n} \left(\frac{N \bar{x}_N}{n x_i} - 1\right)^2 \right]. \end{aligned}$$

A sample \mathbf{s}_n which minimizes $\sum_{i \in \mathbf{s}_n} \left(\frac{N \bar{x}_N}{n x_i} - 1\right)^2$ is optimal.

The predictor \hat{Y}_{RG} can be written as

$$\hat{Y}_{RG} = \bar{x}_N \frac{\sum_{i=1}^N I_i x_i y_i}{\sum_{i=1}^N I_i x_i^2} = \bar{x}_N \left(\frac{\sum_{i=1}^N I_i x_i (\beta x_i + e_i)}{\sum_{i=1}^N I_i x_i^2} \right) = \beta \bar{x}_N + \bar{x}_N \frac{\sum_{i=1}^N I_i x_i e_i}{\sum_{i=1}^N I_i x_i^2}.$$

Hence, $E\{\hat{Y}_{RG}\} = \beta \bar{x}_N$ and \hat{Y}_{RG} is an unbiased predictor of μ_N . The conditional prediction MSE , given \mathbf{s}_n , is

$$PMSE\{\hat{Y}_{RG} \mid \mathbf{s}_n\} = \frac{\sigma^2}{N^2} \left[N + \frac{N^2 \bar{x}_N^2}{\sum_{i \in \mathbf{s}_n} x_i^2} - 2N \bar{x}_N \frac{n \bar{X}_n}{\sum_{i \in \mathbf{s}_n} x_i^2} \right].$$

Chapter 6

Time Series Analysis and Prediction

Import required modules and define required functions

```
import math
import mistat
import numpy as np
import pandas as pd
from scipy import stats
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import acf, pacf
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa import tsatools
import statsmodels.formula.api as smf
```

Solution 6.1 TODO: Provide a sample solution

Solution 6.2 (i) Fig. 6.1 shows the change of demand over time

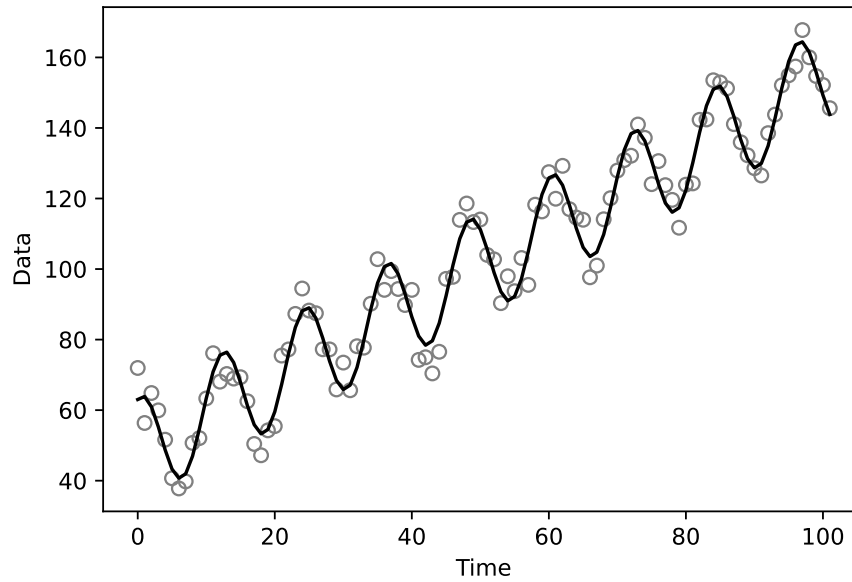
(fig:seascom-timeline)

(ii) We are first fitting the seasonal trend to the data in SeasCom data set. We use the linear model $Y = X\beta + \varepsilon$, where the Y vector has 102 elements, which are in data set. X is a 102x4 matrix of four column vectors. We combined X and Y in a data frame. In addition to the SeasCom data, we add a column of 1's (const), a column with numbers 1 to 102 (trend) and two columns to describe the seasonal pattern. The column `season_1` consists of $\cos(\pi \times \text{trend}/6)$, and the column `season_2` is $\sin(\pi \times \text{trend}/6)$.

```
seascom = mistat.load_data('SEASCOM.csv')
df = tsatools.add_trend(seascom, trend='ct')
df['season_1'] = [np.cos(math.pi * tx/6) for tx in df['trend']]
df['season_2'] = [np.sin(math.pi * tx/6) for tx in df['trend']]
print(df.head())

model = smf.ols(formula='SeasCom ~ trend + 1 + season_1 + season_2',
                data=df).fit()
print(model.params)
print(f'r2-adj: {model.rsquared_adj:.3f}')
```

	SeasCom	const	trend	season_1	season_2
0	71.95623	1.0	1.0	8.660254e-01	0.500000
1	56.36048	1.0	2.0	5.000000e-01	0.866025
2	64.85331	1.0	3.0	6.123234e-17	1.000000



{fig:seascom-timeline} **Fig. 6.1** Seasonal trend model of SeasCom data set

```
3  59.93460    1.0    4.0 -5.000000e-01  0.866025
4  51.62297    1.0    5.0 -8.660254e-01  0.500000
Intercept    47.673469
trend        1.047236
season_1     10.653968
season_2     10.130145
dtype: float64
r2-adj: 0.981
```

The least squares estimates of β is

$$b = (47.67347, 1.04724, 10.65397, 10.13015)'$$

{fig:seascom-timeline} The fitted trend is $Y_t = Xb$. (see Fig. 6.1).

```
seascom = mistat.load_data('SEASCOM.csv')
fig, ax = plt.subplots()
ax.scatter(seascom.index, seascom, facecolors='none', edgecolors='grey')
model.predict(df).plot(ax=ax, color='black')
ax.set_xlabel('Time')
ax.set_ylabel('Data')
plt.show()
```

{fig:seascom-model-deviation} **(iii)** Calculate the residuals and plot them (see Fig. 6.2).

```
U = df['SeasCom'] - model.predict(df)
fig, ax = plt.subplots()
ax.scatter(U.index, U, facecolors='none', edgecolors='black')
```

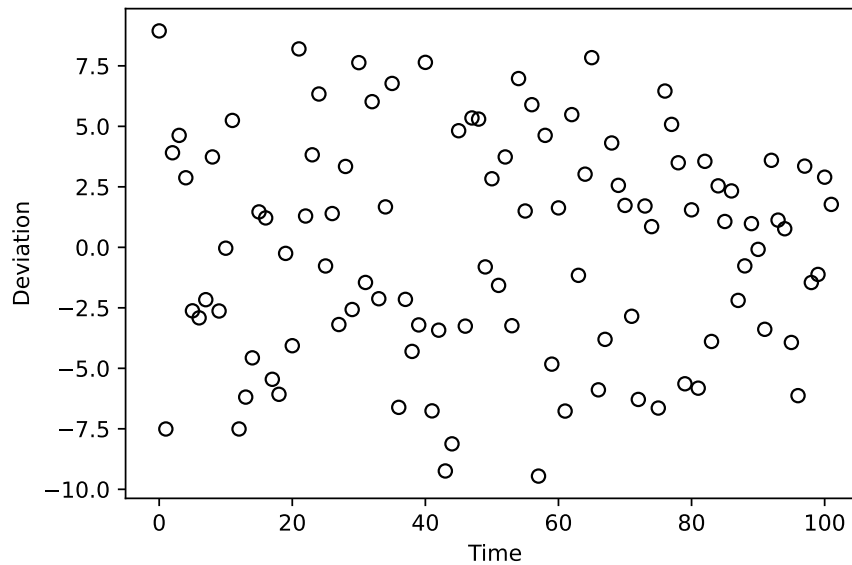


Fig. 6.2 Deviation of SeasCom data from cyclical trend

(fig:seascom-model-deviation)

```
ax.set_xlabel('Time')
ax.set_ylabel('Deviation')
plt.show()
```

(iv) Calculate the correlations

```
# use slices to get sublists
corr_1 = np.corrcoef(U[:-1], U[1:])[0][1]
corr_2 = np.corrcoef(U[:-2], U[2:])[0][1]
print(f'Corr(Ut,Ut-1) = {corr_1:.3f}')
print(f'Corr(Ut,Ut-2) = {corr_2:.3f}')
```

```
| Corr(Ut,Ut-1) = -0.191
| Corr(Ut,Ut-2) = 0.132
```

Indeed the correlations between adjacent data points are $\text{corr}(X_t, X_{t-1}) = -0.191$, and $\text{corr}(X_t, X_{t-2}) = 0.132$.

(iv) A plot of the deviations and the low autocorrelations shows something like randomness.

```
# keep some information for later exercises
seascom_model = model
seascom_df = df
```

Solution 6.3 According to Equation 6.2.2, the auto-correlation in a stationary MA(q) is

(eqn:ma-covariance)

$$\rho(h) = \frac{K(h)}{K(0)} = \frac{\sum_{j=0}^{q-h} \beta_j \beta_{j+h}}{\sum_{j=0}^q \beta_j^2}$$

for $0 \leq h \leq q$.

Notice that $\rho(h) = \rho(-h)$, and $\rho(h) = 0$ for all $h > q$.

Solution 6.4 In Python

```

beta = np.array([1, 1.05, 0.76, -0.35, 0.45, 0.55])
data = []
n = len(beta)
sum_0 = np.sum(beta * beta)
for h in range(6):
    sum_h = np.sum(beta[:n-h] * beta[h:])
    data.append({
        'h': h,
        'K(h)': sum_h,
        'rho(h)': sum_h / sum_0,
    })

```

	0	1	2	3	4	5
K(h)	3.308	1.672	0.542	0.541	1.028	0.550
rho(h)	1.000	0.506	0.164	0.163	0.311	0.166

Solution 6.5 We consider the $AQ(\infty)$, given by coefficients $\beta_j = q^j$, with $0 < q < 1$. In this case,

- (i) $E\{X_t\} = 0$, and
- (ii) $V\{X_t\} = \sigma^2 \sum_{j=0}^{\infty} q^{2j} = \sigma^2 / (1 - q^2)$.

Solution 6.6 We consider the AR(1), $X_t = 0.75X_{t-1} + \varepsilon_t$.

(i) This time-series is equivalent to $X_t = \sum_{j=0}^{\infty} (-0.75)^j \varepsilon_{t-j}$, hence it is covariance stationary.

- (ii) $E\{X_t\} = 0$
- (iii) According to the Yule-Walker equations,

$$\begin{bmatrix} 1 & -0.75 \\ -0.75 & 1 \end{bmatrix} \begin{bmatrix} K(0) \\ K(1) \end{bmatrix} = \begin{bmatrix} \sigma^2 \\ 0 \end{bmatrix}$$

It follows that $K(0) = 2.285714 \sigma^2$ and $K(1) = 1.714286 \sigma^2$.

Solution 6.7 The given AR(2) can be written as $\mathbf{X}_t - 0.5\mathbf{X}_{t-1} + 0.3\mathbf{X}_{t-2} = \varepsilon_t$.

(i) The corresponding characteristic polynomial is $\mathbf{P}_2(\mathbf{z}) = 0.3 - 0.5\mathbf{z} + \mathbf{z}^2$. The two characteristic roots are $\mathbf{z}_{1,2} = \frac{1}{4} \pm i \frac{\sqrt{95}}{20}$. These two roots belong to the unit circle. Hence this AR(2) is covariance stationary.

(ii) We can write $A_2(z)X_t = \varepsilon_t$, where $A_2(z) = 1 - 0.5z^{-1} + 0.3z^{-2}$. Furthermore, $\phi_{1,2}$ are the two roots of $A_2(z) = 0$.

It follows that

$$\begin{aligned} X_t &= (A_2(z))^{-1} \varepsilon_t \\ &= \varepsilon_t + 0.5\varepsilon_{t-1} - 0.08\varepsilon_{t-2} - 0.205\varepsilon_{t-3} - 0.0761\varepsilon_{t-4} + 0.0296\varepsilon_{t-5} + \dots \end{aligned}$$

Solution 6.8 The Yule-Walker equations are:

$$\begin{bmatrix} 1 & -0.5 & 0.3 & -0.2 \\ -0.5 & 1.3 & -0.2 & 0 \\ 0.3 & -0.7 & 1 & 0 \\ -0.2 & 0.3 & -0.5 & 1 \end{bmatrix} \cdot \begin{bmatrix} K(0) \\ K(1) \\ K(2) \\ K(3) \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The solution is $K(0) = 1.2719$, $K(1) = 0.4825$, $K(2) = -0.0439$, $K(3) = 0.0877$.

Solution 6.9 The Toeplitz matrix is

$$R_4 = \begin{bmatrix} 1.0000 & 0.3794 & -0.0235 & 0.0690 \\ 0.3794 & 1.0000 & 0.3794 & -0.0235 \\ -0.0235 & 0.3794 & 1.0000 & 0.3794 \\ 0.0690 & -0.0235 & 0.3794 & 1.0000 \end{bmatrix}$$

Solution 6.10 This series is an ARMA(2,2), given by the equation

$$(1 - z^{-1} + 0.25z^{-2})X_t = (1 + 0.4z^{-1} - 0.45z^{-2})\varepsilon_t.$$

Accordingly,

$$\begin{aligned} X_t &= (1 + 0.4z^{-1} - 0.45z^{-2})(1 - z^{-1} + 0.25z^{-2})^{-1}\varepsilon_t \\ &= \varepsilon_t + 1.4\varepsilon_{t-1} + 0.7\varepsilon_{t-2} + 0.35\varepsilon_{t-3} + 0.175\varepsilon_{t-4} \\ &\quad + 0.0875\varepsilon_{t-5} + 0.0438\varepsilon_{t-6} + 0.0219\varepsilon_{t-7} + 0.0109\varepsilon_{t-8} + \dots \end{aligned}$$

Solution 6.11 Let X denote the DOW1941 data set. We create a new set, Y of second order difference, i.e., $Y_t = X_t - 2X_{t-1} + X_{t-2}$.

```
dow1941 = mistat.load_data('DOW1941.csv')

X = dow1941.values # extract values to remove index for calculations
Y = X[2:] - 2 * X[1:-1] + X[:-2]
```

(i) In the following table we present the autocorrelations, acf, and the partial autocorrelations, pacf, of Y . For a visualisation see Fig. 6.3.

(fig:acf-pacf-dow-second-order)

```
# use argument alpha to return confidence intervals
y_acf, ci_acf = acf(Y, nlags=11, fft=True, alpha=0.05)
y_pacf, ci_pacf = pacf(Y, nlags=11, alpha=0.05)

# determine if values are significantly different from zero
def is_significant(y, ci):
    return not (ci[0] < 0 < ci[1])

s_acf = [is_significant(y, ci) for y, ci in zip(y_acf, ci_acf)]
s_pacf = [is_significant(y, ci) for y, ci in zip(y_pacf, ci_pacf)]
```

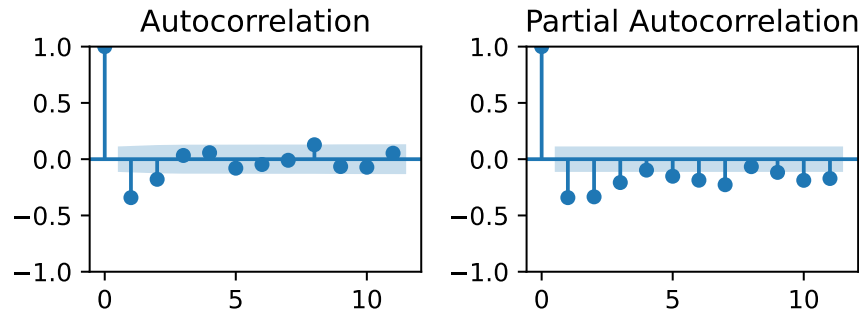


Fig. 6.3 Autocorrelations, acf, and the partial autocorrelations, pacf, of the second order differences of the DOW1941 dataset.

{fig:acf-pacf-dow-second-order}

h	acf	S/NS	pacf	S/NS
1	-0.342	S	-0.343	S
2	-0.179	S	-0.337	S
3	0.033	NS	-0.210	S
4	0.057	NS	-0.100	NS
5	-0.080	NS	-0.155	S
6	-0.047	NS	-0.193	S
7	-0.010	NS	-0.237	S
8	0.128	NS	-0.074	NS
9	-0.065	NS	-0.127	S
10	-0.071	NS	-0.204	S
11	0.053	NS	-0.193	S

S denotes significantly different from 0. NS denotes not significantly different from 0.

(ii) All other correlations are not significant. It seems that the ARIMA(1,2,2) is a good approximation.

{fig:seascom-one-day-ahead-model}

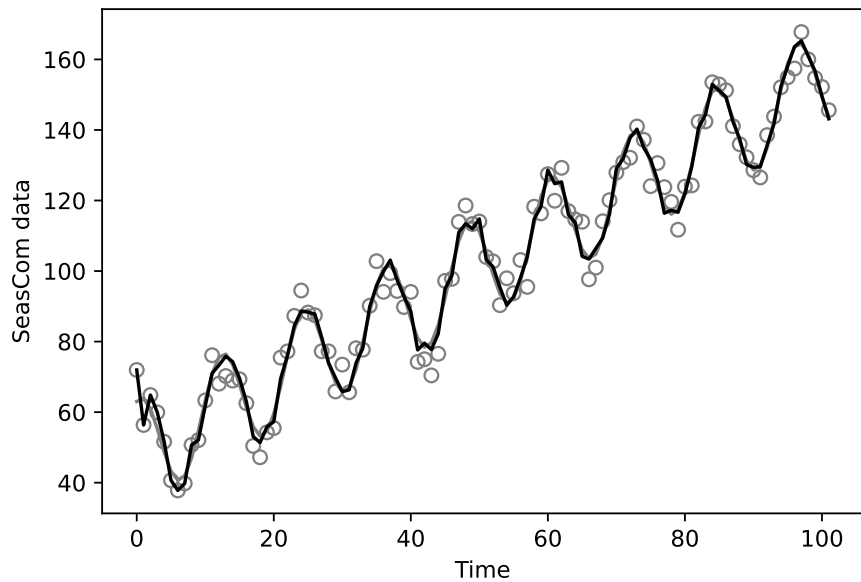
Solution 6.12 In Fig. 6.4, we present the seasonal data SeasCom, and the one-day ahead predictions, We see an excellent prediction.

```

predictedError = mistat.optimalLinearPredictor(seascom_model.resid,
                                              10, nlags=9)
predictedTrend = seascom_model.predict(seascom_df)
correctedTrend = predictedTrend + predictedError

fig, ax = plt.subplots()
ax.scatter(seascom_df.index, seascom_df['SeasCom'],
          facecolors='none', edgecolors='grey')
predictedTrend.plot(ax=ax, color='grey')
correctedTrend.plot(ax=ax, color='black')
ax.set_xlabel('Time')
ax.set_ylabel('SeasCom data')
plt.show()

```



[fig:seascom-one-day-ahead-model] **Fig. 6.4** One-day ahead prediction model of SeasCom data set

Chapter 7

Modern analytic methods: Part I

Import required modules and define required functions

```
import warnings
import mistat
import random
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import mean_squared_error
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import KBinsDiscretizer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from xgboost import XGBClassifier
from sklearn.cluster import KMeans
from sklearn.pipeline import make_pipeline

# Uncomment the following if xgboost crashes
import os
os.environ['KMP_DUPLICATE_LIB_OK'] = 'TRUE'
```

Solution 7.1 Articles reviewing the application of supervised and unsupervised methods can be found online (see e.g., doi:10.1016/j.chaos.2020.110059)

An example for supervised applications is the classification of a COVID-19 based on diagnostic data (see e.g., doi:10.1155/2021/4733167 or doi:10.3390/s21103322)

An example of unsupervised learning is hierarchical clustering to evaluate COVID-19 pandemic preparedness and performance in 180 countries (see doi:10.1016/j.rinp.2021.104639)

Solution 7.2 The decision tree model for `testResult` results in the confusion matrix shown in Fig. 7.1.

{fig:ex-confusion-matrix-testResult}

```
sensors = mistat.load_data('SENSORS.csv')
predictors = [c for c in sensors.columns if c.startswith('sensor')]
outcome = 'testResult'
X = sensors[predictors]
```

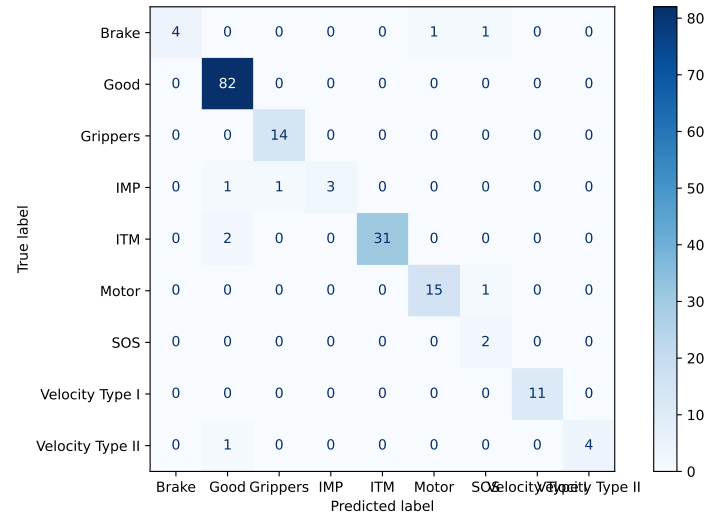


Fig. 7.1 Decision tree model to predict testResult from sensor data (Exc. 7.2)

```

y = sensors[outcome]

# Train the model
clf = DecisionTreeClassifier(ccp_alpha=0.012, random_state=0)
clf.fit(X, y)

fig, ax = plt.subplots(figsize=(10, 6))
ConfusionMatrixDisplay.from_estimator(clf, X, y, ax=ax, cmap=plt.cm.Blues)

plt.show()

```

The model's classification performance is very good. The test result 'Good', which corresponds to status 'Pass' is correctly predicted. Most of the other individual test results have also low missclassification rates. The likely reason for this is that each test result is associated with a specific subset of the sensors.

Solution 7.3 In Python

```

# convert the status information into numerical labels
outcome = 'status'
y = sensors[outcome].replace({'Pass': 1, 'Fail': 0})

# Train the model
xgb = XGBClassifier(objective='binary:logistic', subsample=.63,
                    eval_metric='logloss', use_label_encoder=False,
                    seed=1)

xgb.fit(X, y)

# actual in rows / predicted in columns
print('Confusion matrix')
print(confusion_matrix(y, xgb.predict(X)))

```

```
Confusion matrix
[[92  0]
 [ 0 82]]
/usr/local/lib/python3.9/site-packages/xgboost/sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
```

The models confusion matrix is perfect.

```
var_imp = pd.DataFrame({
    'importance': xgb.feature_importances_,
    }, index=predictors)
var_imp = var_imp.sort_values(by='importance', ascending=False)
var_imp['order'] = range(1, len(var_imp) + 1)
print(var_imp.head(10))
var_imp.loc[var_imp.index.isin(['sensor18', 'sensor07', 'sensor21'])]
```

	importance	order
sensor18	0.290473	1
sensor54	0.288680	2
sensor53	0.105831	3
sensor55	0.062423	4
sensor61	0.058112	5
sensor48	0.040433	6
sensor07	0.026944	7
sensor12	0.015288	8
sensor03	0.013340	9
sensor52	0.013160	10

	importance	order
sensor18	0.290473	1
sensor07	0.026944	7
sensor21	0.000000	50

The decision tree model uses sensors 18, 7, and 21. The xgboost model identifies sensor 18 as the most important variable. Sensor 7 is ranked 7th, sensor 21 has no importance.

Solution 7.4 Create the random forest classifier model.

```
y = sensors['status']

# Train the model
model = RandomForestClassifier(ccp_alpha=0.012, random_state=0)
model.fit(X, y)

# actual in rows / predicted in columns
print('Confusion matrix')
print(confusion_matrix(y, model.predict(X)))
```

```
Confusion matrix
[[92  0]
 [ 0 82]]
```

The models confusion matrix is perfect.

```
var_imp = pd.DataFrame({
    'importance': model.feature_importances_,
    }, index=predictors)
var_imp = var_imp.sort_values(by='importance', ascending=False)
```

```
var_imp['order'] = range(1, len(var_imp) + 1)
print(var_imp.head(10))
var_imp.loc[var_imp.index.isin(['sensor18', 'sensor07', 'sensor21'])]
```

	importance	order
sensor61	0.138663	1
sensor18	0.100477	2
sensor53	0.079890	3
sensor52	0.076854	4
sensor46	0.052957	5
sensor50	0.051970	6
sensor44	0.042771	7
sensor48	0.037087	8
sensor24	0.036825	9
sensor21	0.035014	10

	importance	order
sensor18	0.100477	2
sensor21	0.035014	10
sensor07	0.023162	17

The decision tree model uses sensors 18, 7, and 21. Sensor 18 has the second largest importance value, sensor 21 ranks 10th, and sensor 7 is on rank 17.

Solution 7.5 In Python:

```
# convert outcome values from strings into numerical labels
# use le.inverse_transform to convert the predictions to strings
le = LabelEncoder()
y = le.fit_transform(sensors['status'])

train_X, valid_X, train_y, valid_y = train_test_split(X, y,
    test_size=0.4, random_state=2)

dt_model = DecisionTreeClassifier(ccp_alpha=0.012, random_state=0)
dt_model.fit(train_X, train_y)

xgb_model = XGBClassifier(objective='binary:logistic', subsample=.63,
    eval_metric='logloss', use_label_encoder=False,
    seed=1)
xgb_model.fit(train_X, train_y)

rf_model = RandomForestClassifier(ccp_alpha=0.014, random_state=0)
rf_model.fit(train_X, train_y)

print('Decision tree model')
print(f'Accuracy {accuracy_score(valid_y, dt_model.predict(valid_X)):.3f}')
print(confusion_matrix(valid_y, dt_model.predict(valid_X)))

print('Gradient boosting model')
print(f'Accuracy {accuracy_score(valid_y, xgb_model.predict(valid_X)):.3f}')
print(confusion_matrix(valid_y, xgb_model.predict(valid_X)))

print('Random forest model')
print(f'Accuracy {accuracy_score(valid_y, rf_model.predict(valid_X)):.3f}')
print(confusion_matrix(valid_y, rf_model.predict(valid_X)))
```

```
/usr/local/lib/python3.9/site-packages/xgboost/sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
Decision tree model
Accuracy 0.900
[[37  2]
```

```
[ 5 26]]
Gradient boosting model
Accuracy 0.957
[[36  3]
 [ 0 31]]
Random forest model
Accuracy 0.986
[[38  1]
 [ 0 31]]
```

The accuracy for predicting the validation set is very good for all three models, with random forest giving the best model with an accuracy of 0.986. The xgboost model has a slightly lower accuracy of 0.957 and the accuracy for the decision tree model is 0.900.

If you change the random seeds or remove them for the various commands, you will see that the accuracies vary and that the order can change.

Solution 7.6 In Python:

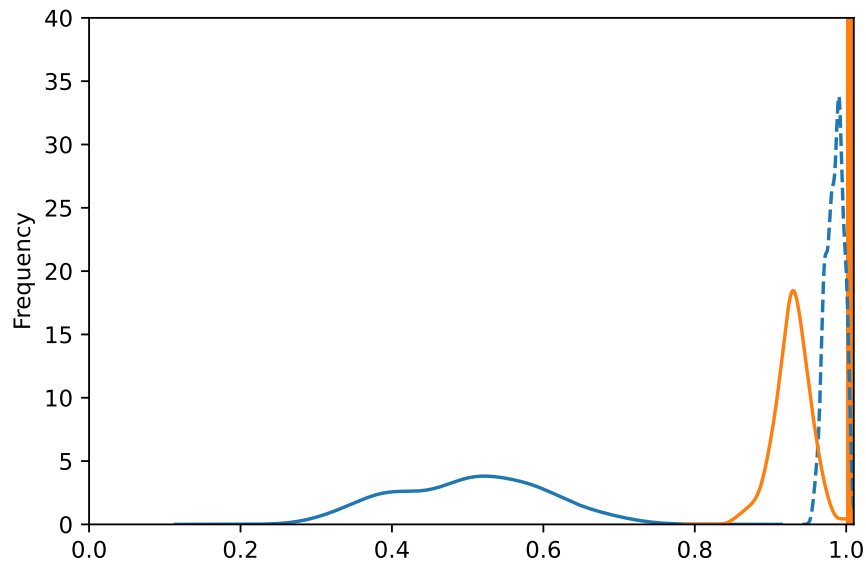
```
dt_model = DecisionTreeClassifier(ccp_alpha=0.012)

random_valid_acc = []
random_train_acc = []
org_valid_acc = []
org_train_acc = []
for _ in range(100):
    train_X, valid_X, train_y, valid_y = train_test_split(X, y,
                                                            test_size=0.4)
    dt_model.fit(train_X, train_y)
    org_train_acc.append(accuracy_score(train_y, dt_model.predict(train_X)))
    org_valid_acc.append(accuracy_score(valid_y, dt_model.predict(valid_X)))

    random_y = random.sample(list(train_y), len(train_y))
    dt_model.fit(train_X, random_y)
    random_train_acc.append(accuracy_score(random_y, dt_model.predict(train_X)))
    random_valid_acc.append(accuracy_score(valid_y, dt_model.predict(valid_X)))

ax = pd.Series(random_valid_acc).plot.density(color='C0')
pd.Series(random_train_acc).plot.density(color='C0', linestyle='--',
                                          ax=ax)
pd.Series(org_valid_acc).plot.density(color='C1', ax=ax)
pd.Series(org_train_acc).plot.hist(color='C1', linestyle='--',
                                   ax=ax)

ax.set_ylim(0, 40)
ax.set_xlim(0, 1.01)
plt.show()
```



Solution 7.7 Load data

```
distTower = mistat.load_data('DISTILLATION-TOWER.csv')
predictors = ['Temp1', 'Temp2', 'Temp3', 'Temp4', 'Temp5', 'Temp6',
              'Temp7', 'Temp8', 'Temp9', 'Temp10', 'Temp11', 'Temp12']
outcome = 'VapourPressure'
Xr = distTower[predictors]
yr = distTower[outcome]
```

(i) Split dataset into train and validation set

```
train_X, valid_X, train_y, valid_y = train_test_split(Xr, yr,
                                                       test_size=0.2, random_state=2)
```

(ii) Determine model performance for different tree complexity along the dependence of tree depth on ccp parameter; see Fig. 7.2.

{fig:exc-cpp-pruning}

```
# Code to analyze tree depth vs alpha
model = DecisionTreeRegressor(random_state=0)
path = model.cost_complexity_pruning_path(Xr, yr)
ccp_alphas, impurities = path.ccp_alphas, path.impurities
mse = []
mse_train = []
for ccp_alpha in ccp_alphas:
    model = DecisionTreeRegressor(random_state=0, ccp_alpha=ccp_alpha)
    model.fit(train_X, train_y)
    mse.append(mean_squared_error(valid_y, model.predict(valid_X)))
    mse_train.append(mean_squared_error(train_y, model.predict(train_X)))
ccp_alpha = ccp_alphas[np.argmin(mse)]
```

The smallest validation set error is obtained for `ccp_alpha = 0.372`. The dependence of training and validation error on `ccp_alpha` is shown in Fig. 7.2.

{fig:exc-cpp-pruning}

(iii) The final model is visualized using `dtreeviz` in Fig. 7.3.

{fig:exc-dtreeviz-visualization}

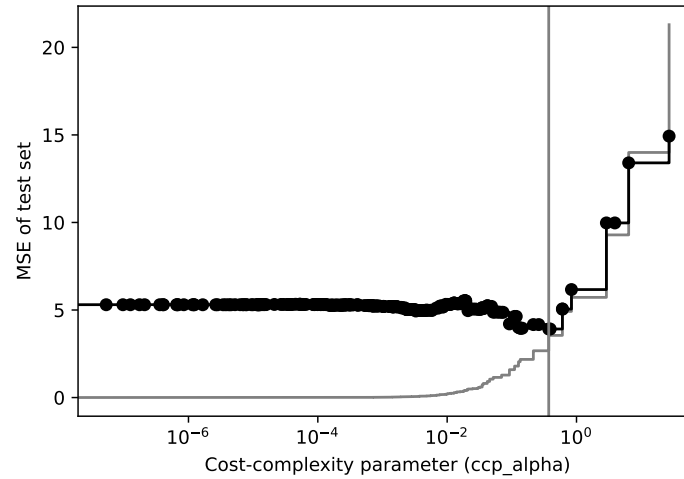


Fig. 7.2 Decision tree regressor complexity as a function of `ccp_alpha`. The validation set error is shown in black, the training set error in grey (Exercise 7.7)

(fig:exc-cpp-pruning)

```
# the dtreeviz methods requires the classifier to be trained with a numerical
# representation of the classes
# Train the model
model = DecisionTreeRegressor(ccp_alpha=ccp_alpha, random_state=0)
model.fit(Xr, yr)
```

```
viz = dtreeviz.model(model, Xr, yr,
                    target_name=outcome,
                    feature_names=Xr.columns)
viz2pdf(viz, 'compiled/figures/Chap008_ex_dtreeviz_regressor.pdf')
```

```
-----AttributeError
Traceback (most recent call last)Cell In[1], line 10
      5 model.fit(Xr, yr)
      7 viz = dtreeviz.model(model, Xr, yr,
      8                       target_name=outcome,
      9                       feature_names=Xr.columns)
--> 10 viz2pdf(viz,
'compiled/figures/Chap008_ex_dtreeviz_regressor.pdf')
Cell In[1], line 8, in viz2pdf(viz, pdfFile)
      6 from tempfile import NamedTemporaryFile
      7 with NamedTemporaryFile(mode='w+', suffix='.svg') as f:
--> 8     f.write(viz.svg())
      9     f.flush()
     10     f.seek(0)
AttributeError: 'DTreeVizAPI' object has no attribute 'svg'
```

Solution 7.8 Vary the number of bins and binning strategy. The influence of the two model parameter on model performance is shown in Fig. 7.4.

(fig:nb-binning-performance)

```
y = sensors['status']

results = []
```

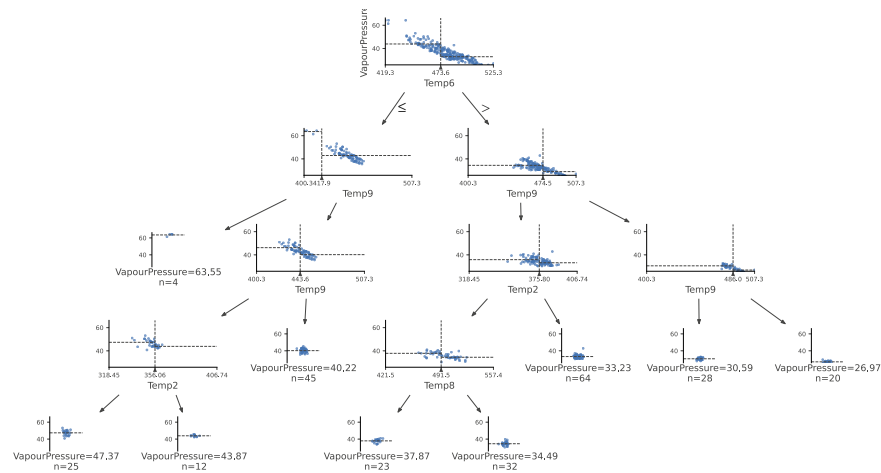


Fig. 7.3 Decision tree visualization of regression tree (Exercise 17.7)

exc-dec-tree-reg-distillation

(fig:exc-dtreeviz-visualization)

```
for strategy in 'uniform', 'quantile', 'kmeans':
    for n_bins in range(2, 11):
        kbinsDiscretizer = KBinsDiscretizer(encode='ordinal',
            strategy=strategy, n_bins=n_bins)
        X_binned = kbinsDiscretizer.fit_transform(X)
        nb_model = MultinomialNB()
        nb_model.fit(X_binned, y)
        results.append({'strategy': strategy, 'n_bins': n_bins,
            'accuracy': accuracy_score(y, nb_model.predict(X_binned))})
results = pd.DataFrame(results)
fig, ax = plt.subplots()
for key, group in results.groupby('strategy'):
    group.plot(x='n_bins', y='accuracy', label=key, ax=ax)
```

The quantile binning strategy (for each feature, each bin has the same number of data points) with splitting each column into two bins leads to the best performing model. With this strategy, performance declines with increasing number of bins. The uniform (for each feature, each bin has the same width) and the kmeans (for each feature, a k -means clustering is used to bin the feature) strategies on the other hand, show increasing performance with increasing number of bins.

The confusion matrix for the best performing models is:

```
kbinsDiscretizer = KBinsDiscretizer(encode='ordinal',
    strategy='quantile', n_bins=2)
X_binned = kbinsDiscretizer.fit_transform(X)
nb_model = MultinomialNB()
nb_model.fit(X_binned, y)
print('Confusion matrix')
print(confusion_matrix(y, nb_model.predict(X_binned)))
```

```
Confusion matrix
[[87  5]
 [ 1 81]]
```

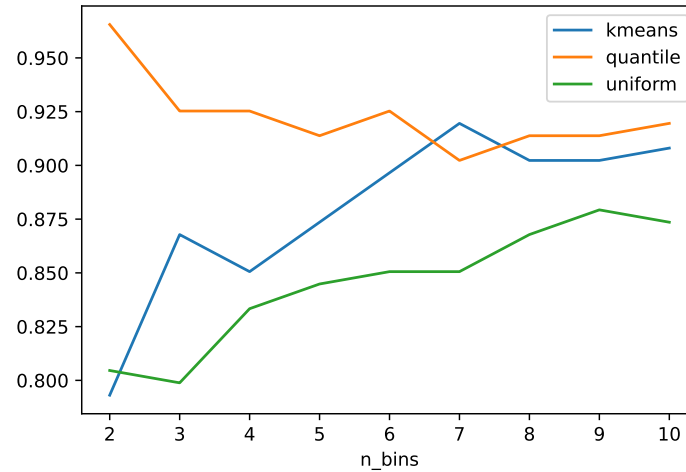



Fig. 7.4 Influence of number of bins and binning strategy on model performance for the sensors data with status as outcome

(fig:nb-binning-performance)

The decision tree model misclassified three of the ‘Pass’ data points as ‘Fail’. The Naïve Bayes model on the other hand misclassifies six data points. However, five of these are ‘Pass’ and predicted as ‘Fail’. Depending on your use case, you may prefer a model with more false negatives or false positives.

Solution 7.9 In Python:

```
from sklearn.cluster import AgglomerativeClustering
from sklearn.preprocessing import StandardScaler
from mistat import plot_dendrogram

food = mistat.load_data('FOOD.csv')

scaler = StandardScaler()
model = AgglomerativeClustering(n_clusters=10, compute_distances=True)

X = scaler.fit_transform(food)
model = model.fit(X)
fig, ax = plt.subplots()
plot_dendrogram(model, ax=ax)
ax.set_title('Dendrogram')
ax.get_xaxis().set_ticks([])
plt.show()
```

Solution 7.10 In Python:

```
food = mistat.load_data('FOOD.csv')
scaler = StandardScaler()
X = scaler.fit_transform(food)

fig, axes = plt.subplots(ncols=2, nrows=2)
```

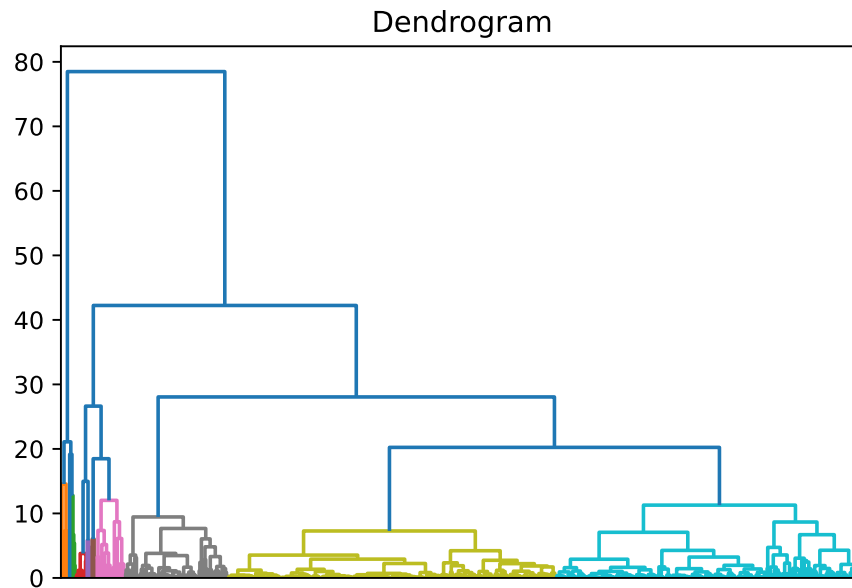


Fig. 7.5 Hierarchical clustering of food data set using Ward clustering

{fig:food-ward-10-clusters}

```
for linkage, ax in zip(['ward', 'complete', 'average', 'single'], axes.flatten()):
    model = AgglomerativeClustering(n_clusters=10, compute_distances=True,
                                    linkage=linkage)
    model = model.fit(X)
    plot_dendrogram(model, ax=ax)
    ax.set_title(linkage)
    ax.get_xaxis().set_ticks([])
plt.show()
```

{fig:food-compare-linkage}

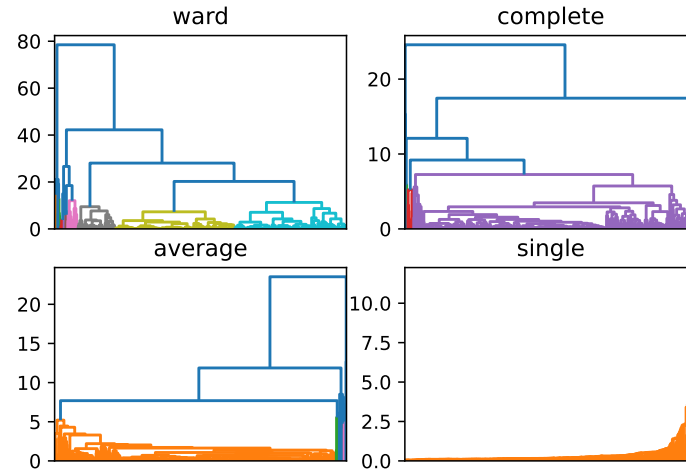
The comparison of the different linkage methods is shown in Fig. 7.6. We can see that Ward's clustering gives the most balanced clusters; three bigger clusters and seven small clusters. Complete, average, and single linkage lead to one big cluster.

Solution 7.11 We determine 10 clusters using K-means clustering.

```
sensors = mistat.load_data('SENSORS.csv')
predictors = [c for c in sensors.columns if c.startswith('sensor')]
outcome = 'status'
X = sensors[predictors]

scaler = StandardScaler()
X = scaler.fit_transform(X)
model = KMeans(n_clusters=10, random_state=1).fit(X)
```

Combine the information and analyse cluster membership by status.



(fig:food-compare-linkage)

Fig. 7.6 Comparison of different linkage methods for hierarchical clustering of food data set

```
df = pd.DataFrame({
    'status': sensors['status'],
    'testResult': sensors['testResult'],
    'cluster': model.predict(X),
})

for status, group in df.groupby('status'):
    print(f'Status {status}')
    print(group['cluster'].value_counts())
```

```
Status Fail
cluster
8    19
0    15
4    13
1    13
9    13
3    10
5     5
7     2
6     1
2     1
Name: count, dtype: int64
Status Pass
cluster
1    48
8    34
Name: count, dtype: int64
```

There are several clusters that contain only 'Fail' data points. They correspond to specific sensor value combinations that are very distinct to the sensor values during normal operation. The 'Pass' data points are found in two clusters. Both of these clusters contain also 'Fail' data points.

Analyse cluster membership by testResult.

```

print('Number of clusters by testResult')
for cluster, group in df.groupby('cluster'):
    print(f'Cluster {cluster}')
    print(group['testResult'].value_counts())
    print()

```

```

Number of clusters by testResult
Cluster 0
testResult
ITM      15
Name: count, dtype: int64

```

```

Cluster 1
testResult
Good      48
Brake      6
IMP        4
Grippers   1
Motor      1
ITM        1
Name: count, dtype: int64

```

```

Cluster 2
testResult
SOS        1
Name: count, dtype: int64

```

```

Cluster 3
testResult
Velocity Type I    10
Name: count, dtype: int64

```

```

Cluster 4
testResult
Grippers    10
ITM          3
Name: count, dtype: int64

```

```

Cluster 5
testResult
Velocity Type II    5
Name: count, dtype: int64

```

```

Cluster 6
testResult
SOS        1
Name: count, dtype: int64

```

```

Cluster 7
testResult
Grippers    2
Name: count, dtype: int64

```

```

Cluster 8
testResult
Good      34
Motor     15
Grippers   1
IMP        1
ITM        1
Velocity Type I    1
Name: count, dtype: int64

```

```

Cluster 9
testResult
ITM      13
Name: count, dtype: int64

```

We can see that some of the test results are only found in one or two clusters.

Solution 7.12 The `scikit-learn` K-means clustering method can return either the cluster centers or the distances of a data point to all the cluster centers. We evaluate both as features for classification.

```
# Data preparation
sensors = mistat.load_data('SENSORS.csv')
predictors = [c for c in sensors.columns if c.startswith('sensor')]
outcome = 'status'
X = sensors[predictors]
scaler = StandardScaler()
X = scaler.fit_transform(X)
y = sensors[outcome]
```

First, classifying data points based on the cluster center. In order to use that information in a classifier, we transform the cluster center information using on-hot-encoding.

```
# Iterate over increasing number of clusters
results = []
clf = DecisionTreeClassifier(ccp_alpha=0.012, random_state=0)
for n_clusters in range(2, 20):
    # fit a model and assign the data to clusters
    model = KMeans(n_clusters=n_clusters, random_state=1)
    model.fit(X)
    Xcluster = model.predict(X)
    # to use the cluster number in a classifier, use one-hot encoding
    # it's necessary to reshape the vector of cluster numbers into a column vector
    Xcluster = OneHotEncoder().fit_transform(Xcluster.reshape(-1, 1))

    # create a decision tree model and determine the accuracy
    clf.fit(Xcluster, y)
    results.append({'n_clusters': n_clusters,
                  'accuracy': accuracy_score(y, clf.predict(Xcluster))})
ax = pd.DataFrame(results).plot(x='n_clusters', y='accuracy')
ax.set_ylim(0.5, 1)
plt.show()
pd.DataFrame(results).round(3)
```

	n_clusters	accuracy
0	2	0.529
1	3	0.805
2	4	0.805
3	5	0.799
4	6	0.793
5	7	0.897
6	8	0.868
7	9	0.879
8	10	0.816
9	11	0.879
10	12	0.879
11	13	0.879
12	14	0.897
13	15	0.897
14	16	0.885
15	17	0.879
16	18	0.891
17	19	0.902

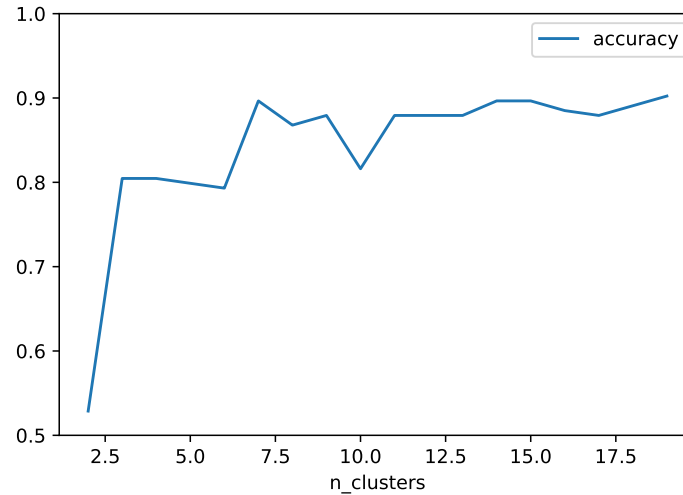


Fig. 7.7 Dependence of accuracy on number of clusters using cluster membership as feature (Exc. 7.12)

{fig:cluster-number-model}

The accuracies are visualized in Fig. 7.7. We see that splitting the dataset into 7 clusters gives a classification model with an accuracy of about 0.9.

{fig:cluster-number-model}

Next we use the distance to the cluster centers as variable in the classifier.

```

results = []
clf = DecisionTreeClassifier(ccp_alpha=0.012, random_state=0)
for n_clusters in range(2, 20):
    # fit a model and convert data to distances
    model = KMeans(n_clusters=n_clusters, random_state=1)
    model.fit(X)
    Xcluster = model.transform(X)

    # create a decision tree model and determine the accuracy
    clf.fit(Xcluster, y)
    results.append({'n_clusters': n_clusters,
                  'accuracy': accuracy_score(y, clf.predict(Xcluster))})
ax = pd.DataFrame(results).plot(x='n_clusters', y='accuracy')
ax.set_ylim(0.5, 1)
plt.show()
pd.DataFrame(results).round(3)

```

	n_clusters	accuracy
0	2	0.983
1	3	0.977
2	4	0.977
3	5	0.977
4	6	0.989
5	7	0.977
6	8	0.977
7	9	0.977
8	10	0.977
9	11	0.977
10	12	0.977
11	13	0.983
12	14	0.977

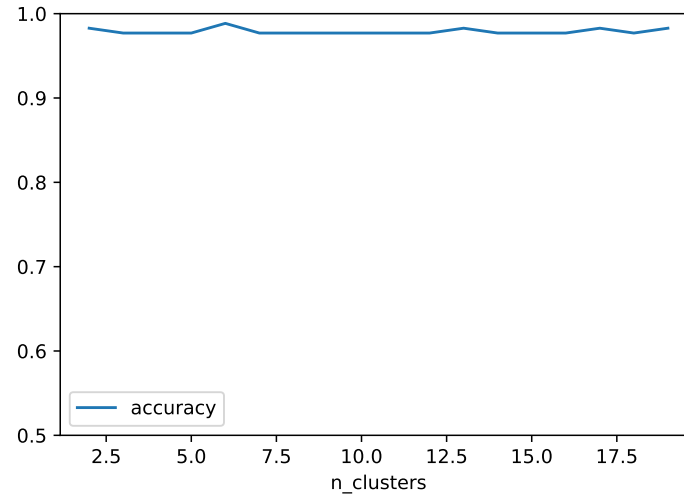


Fig. 7.8 Dependence of accuracy on number of clusters using distance to cluster center as feature (Exc. 7.12)

(fig:cluster-distance-model)

13	15	0.977
14	16	0.977
15	17	0.983
16	18	0.977
17	19	0.983

The accuracies of all models are very high. The largest accuracy is achieved for 6 clusters.

Based on these results, we would design the procedure using the decision tree classifier combined with K-means clustering into six clusters. Using `scikit-learn`, we can define the full procedure as a single pipeline as follows:

```

pipeline = make_pipeline(
    StandardScaler(),
    KMeans(n_clusters=6, random_state=1),
    DecisionTreeClassifier(ccp_alpha=0.012, random_state=0)
)
X = sensors[predictors]
y = sensors[outcome]

process = pipeline.fit(X, y)
print('accuracy', accuracy_score(y, process.predict(X)))
print('Confusion matrix')
print(confusion_matrix(y, process.predict(X)))

```

```

accuracy 0.9885057471264368
Confusion matrix
[[91  1]
 [ 1 81]]

```

The final model has two misclassified data points.

Chapter 8

Modern analytic methods: Part II

Import required modules and define required functions

```
import mistat
import networkx as nx
from pgmpy.estimators import HillClimbSearch
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
```

Solution 8.1 Load the data and convert to FDataGrid.

```
from skfda import FDataGrid
from skfda.representation.interpolation import SplineInterpolation

dissolution = mistat.load_data('DISSOLUTION.csv')

# convert the data to FDataGrid
data = []
labels = []
names = []
for label, group in dissolution.groupby('Label'):
    data.append(group['Data'].values)
    labels.append('Reference' if label.endswith('R') else 'Test')
    names.append(label)
labels = np.array(labels)
grid_points = np.array(sorted(dissolution['Time'].unique()))
fd = FDataGrid(np.array(data), grid_points,
               dataset_name='Dissolution',
               argument_names=['Time'],
               coordinate_names=['Dissolution'])
```

Use shift registration to align the dissolution data with spline interpolation of order 1, 2, and 3.

```
from skfda.preprocessing.registration import ShiftRegistration
shift_registration = ShiftRegistration()

fd_registered = {}
for order in (1, 2, 3):
    fd.interpolation = SplineInterpolation(interpolation_order=order)
    fd_registered[order] = shift_registration.fit_transform(fd)
```

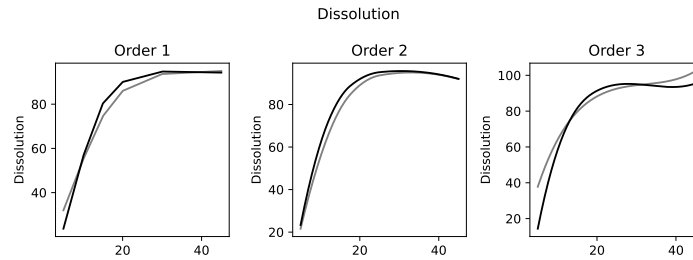


Fig. 8.1 Mean dissolution curves for reference and test tablets derived using spline interpolation of order 1, 2, and 3.

{fig:fdaDissolutionSplineComparison}

For each of the three registered datasets, calculate the mean dissolution curves for reference and test tablets and plot the results.

```
from skfda.exploratory import stats

group_colors = {'Reference': 'grey', 'Test': 'black'}

fig, axes = plt.subplots(ncols=3, figsize=(8, 3))
for ax, order in zip(axes, (1, 2, 3)):
    mean_ref = stats.mean(fd_registered[order][labels=='Reference'])
    mean_test = stats.mean(fd_registered[order][labels=='Test'])
    means = mean_ref.concatenate(mean_test)
    means.plot(axes=[ax], group=['Reference', 'Test'], group_colors=group_colors)
    ax.set_title(f'Order {order}')
plt.tight_layout()
```

{fig:fdaDissolutionSplineComparison}

The dissolution curves are shown in Fig. 8.1. We can see in all three graphs, that the test tablets show a slightly faster dissolution than the reference tablets. If we compare the shape of the curves, the curves for the linear splines interpolation shows a levelling off with time. In the quadratic spline interpolation result, the dissolution curves go through a maximum. This behavior is unrealistic. The cubic spline interpolation also leads to unrealistic curves that first level off and then start to increase again.

Solution 8.2 (i)

```
import skfda
from skfda import FDataGrid

pinchraw = skfda.datasets.fetch_cran('pinchraw', 'fda')['pinchraw']
pinchtime = skfda.datasets.fetch_cran('pinch', 'fda')['pinchtime']

fd = FDataGrid(pinchraw.transpose(), pinchtime)
```

Note that the measurement data need to be transposed.

(ii)

```
fig = fd.plot()
ax = fig.axes[0]
```

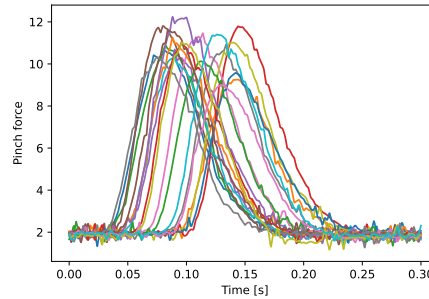


Fig. 8.2 Twenty measurements of pinch force

{fig:fdaPinchforceOriginal}

```
ax.set_xlabel('Time [s]')
ax.set_ylabel('Pinch force')
plt.show()
```

Fig. 8.2 shows the measure pinch forces. We can see that the start of the force varies from 0.025 to 0.1 seconds. This makes it difficult to compare the shape of the curves. The shapes of the individual curves is not symmetric with a faster onset of the force and slower decline.

{fig:fdaPinchforceOriginal}

(iii) We create a variety of smoothed version of the dataset to explore the effect of varying the `smoothing_parameter`.

```
import itertools
from skfda.preprocessing.smoothing.kernel_smothers import NadarayaWatsonSmoother

def plotSmoothData(fd, smoothing_parameter, ax):
    smoother = NadarayaWatsonSmoother(smoothing_parameter=smoothing_parameter)
    fd_smooth = smoother.fit_transform(fd)
    _ = fd_smooth.plot(axes=[ax])
    ax.set_title(f'Smoothing parameter {smoothing_parameter}')
    ax.set_xlabel('Time')
    ax.set_ylabel('Pinch force')

fig, axes = plt.subplots(ncols=2, nrows=2)
axes = list(itertools.chain(*axes)) # flatten list of lists
for i, sp in enumerate([0.03, 0.01, 0.001, 0.0001]):
    plotSmoothData(fd, sp, axes[i])
plt.tight_layout()
```

Fig. 8.3 shows smoothed measurement curves for a variety of `smoothing_parameter`

{fig:fdaPinchforceSmoothing}

values. If values are too large, the data are oversmoothed and the asymmetric shape of the curves is lost. With decreasing values, the shape is reproduced better but the curves are getting noisier again. We select 0.005 as the smoothing parameter.

```
smoother = NadarayaWatsonSmoother(smoothing_parameter=0.005)
fd_smooth = smoother.fit_transform(fd)
```

(iii) We first determine the maxima of the smoothed curves:

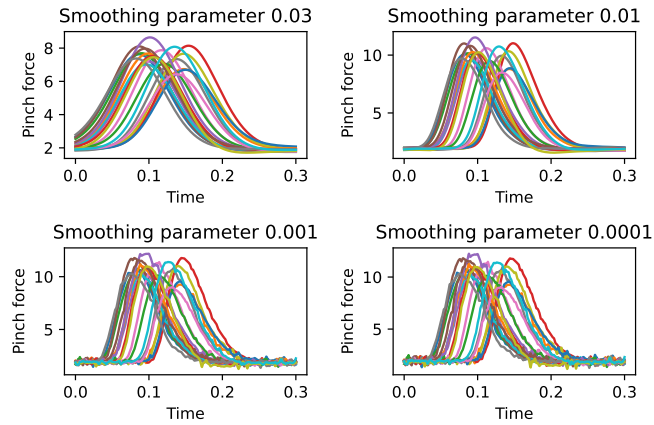


Fig. 8.3 Effect of smoothing parameter on measurement curves

{fig:fdaPinchforceSmoothing}

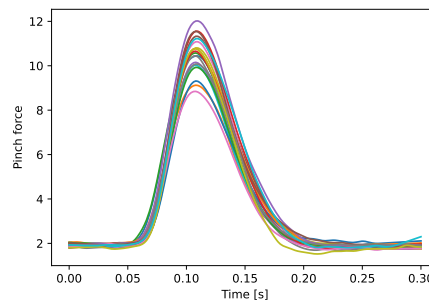


Fig. 8.4 Registered measurement curves of the **Pinch** dataset

{fig:fdaPinchforceRegistered}

```
max_idx = fd_smooth.data_matrix.argmax(axis=1)
landmarks = [pinchtime[idx] for idx in max_idx]
```

Use the landmarks to shift register the measurements:

```
from skfda.preprocessing.registration import landmark_shift
fd_landmark = landmark_shift(fd_smooth, landmarks)
```

(iv) Fig. 8.4 shows the measurements after smoothing and landmark shift registration.

{fig:fdaPinchforceRegistered}

```
fig = fd_landmark.plot()
ax = fig.axes[0]
ax.set_xlabel('Time [s]')
ax.set_ylabel('Pinch force')
plt.show()
```

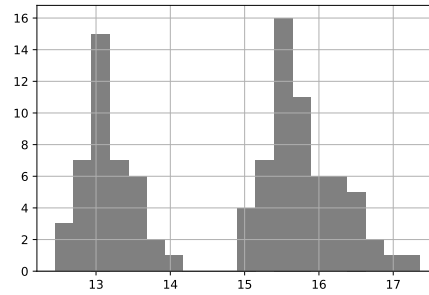


Fig. 8.5 Histogram of moisture content

{fig:fdaMoistureDistribution}

Solution 8.3 (i) Load the data

```
import skfda

moisturespectrum = skfda.datasets.fetch_cran('Moisturespectrum', 'fds')
moisturevalues = skfda.datasets.fetch_cran('Moisturevalues', 'fds')

frequencies = moisturespectrum['Moisturespectrum']['x']
spectra = moisturespectrum['Moisturespectrum']['y']
moisture = moisturevalues['Moisturevalues']
```

(ii) We can use a histogram to look at the distribution of the moisture values.

```
_ = pd.Series(moisture).hist(bins=20, color='grey', label='Moisture content')
```

Fig. 8.5 shows a bimodal distribution of the moisture content with a clear separation of the two peaks. Based on this, we select 14.5 as the threshold to separate into high and low moisture content.

{fig:fdaMoistureDistribution}

```
moisture_class = ['high' if m > 14.5 else 'low' for m in moisture]
```

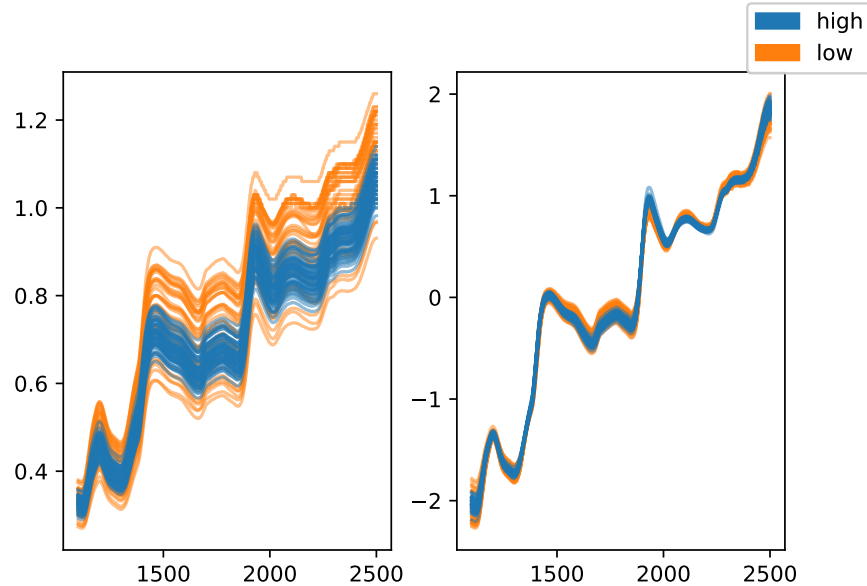
(iii - iv) The spectrum information is already in the array format required for the FDataGrid class. In order to do this, we need to transpose the spectrum information. As we can see in the left graph of Fig. 8.6, the spectra are not well aligned but show a large spread in the intensities. This is likely due to the difficulty in having a clearly defined concentration between samples. In order to reduce this variation, we can transform the intensities by dividing the intensities by the mean of each sample.

{fig:fdaMoistureSpectrum}

```
intensities = spectra.transpose()
fd = skfda.FDataGrid(intensities, frequencies)

# divide each sample spectrum by it's mean intensities
intensities_normalized = (intensities - intensities.mean(dim='dim_0')) / intensities.std(dim='dim_0')
fd_normalized = skfda.FDataGrid(intensities_normalized, frequencies)
```

Code for plotting the spectra:



{fig:fdaMoistureSpectrum}

Fig. 8.6 Near-infrared spectra of the moisture dataset. Left: raw spectra, Right: normalized spectra

```
fig, axes = plt.subplots(ncols=2)
_ = fd.plot(axes=axes[0], alpha=0.5,
            # color lines by moisture class
            group=moisture_class, group_names={'high': 'high', 'low': 'low'})
_ = fd_normalized.plot(axes=axes[1], alpha=0.5,
                      group=moisture_class, group_names={'high': 'high', 'low': 'low'})
```

{fig:fdaMoistureSpectrum}

As we can see in right graph of Fig. 8.6, the normalized spectra are now better aligned. We also see that the overall shape of the spectra is fairly consistent between samples.

(v) We repeat the model building both for the original and normalized spectra 50 times. At each iteration, we split the data set into training and test sets (50-50), build the model with the training set and measure accuracy using the test set. By using the same random seed for splitting the original and the normalized dataset, we can better compare the models. The accuracies from the 50 iteration are compared in Fig. 8.7.

{fig:fdaMoistureAccuracies}

```
from skfda.ml.classification import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix

accuracies = []
for rs in range(50):
    X_train, X_test, y_train, y_test = train_test_split(fd,
                                                        moisture_class, random_state=rs, test_size=0.5)
    knn_original = KNeighborsClassifier()
    knn_original.fit(X_train, y_train)
```

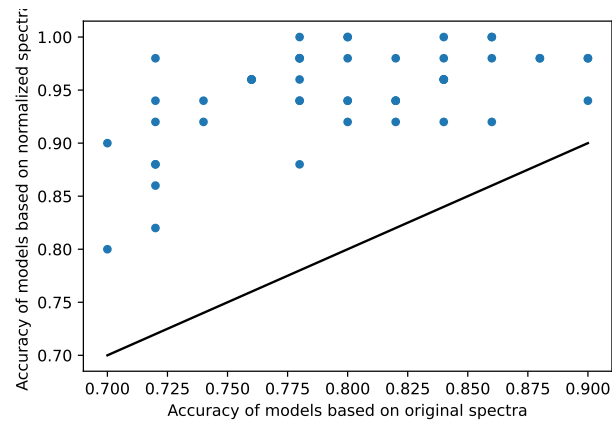


Fig. 8.7 Accuracies of classification models based original and normalized spectra. The line indicates equal performance.

{fig:fdaMoistureAccuracies}

```
acc_original = accuracy_score(y_test, knn_original.predict(X_test))

X_train, X_test, y_train, y_test = train_test_split(fd_normalized,
                                                    moisture_class, random_state=rs, test_size=0.5)
knn_normalized = KNeighborsClassifier()
knn_normalized.fit(X_train, y_train)
acc_normalized = accuracy_score(y_test, knn_normalized.predict(X_test))
accuracies.append({
    'original': acc_original,
    'normalized': acc_normalized,
})
accuracies = pd.DataFrame(accuracies)
ax = accuracies.plot.scatter(x='original', y='normalized')
_ = ax.plot([0.7, 0.9], [0.7, 0.9], color='black')
ax.set_xlabel('Accuracy of models based on original spectra')
ax.set_ylabel('Accuracy of models based on normalized spectra')
plt.show()

# mean of accuracies
mean_accuracy = accuracies.mean()
mean_accuracy
```

```
original      0.7976
normalized    0.9468
dtype: float64
```

Fig. 8.7 clearly shows that classification models based on the normalized spectra achieve better accuracies. The mean accuracy increases from 0.8 to 0.95.

{fig:fdaMoistureAccuracies}

Solution 8.4 (i) See solution for Exercise 8.3.

{exc:fda-moisture-classification}

(ii) We use the method `skfda.ml.regression.KNeighborsRegressor` to build the regression models.

```
from skfda.ml.regression import KNeighborsRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
```

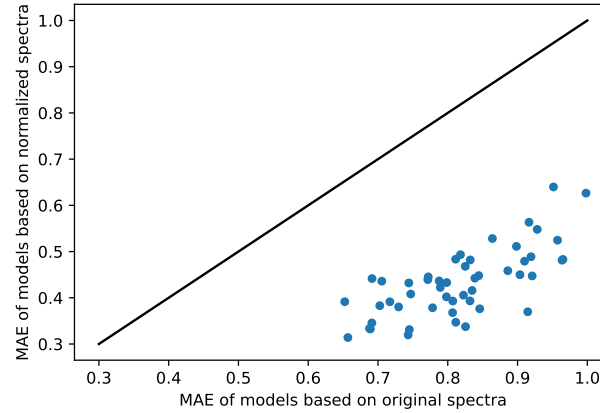


Fig. 8.8 Mean absolute error of regression models using original and normalized spectra. The line indicates equal performance.

(fig:fdaMoistureMAE)

```
mae = []
for rs in range(50):
    X_train, X_test, y_train, y_test = train_test_split(fd,
        moisture, random_state=rs, test_size=0.5)
    knn_original = KNeighborsRegressor()
    knn_original.fit(X_train, y_train)
    mae_original = mean_absolute_error(y_test, knn_original.predict(X_test))

    X_train, X_test, y_train, y_test = train_test_split(fd_normalized,
        moisture, random_state=rs, test_size=0.5)
    knn_normalized = KNeighborsRegressor()
    knn_normalized.fit(X_train, y_train)
    mae_normalized = mean_absolute_error(y_test, knn_normalized.predict(X_test))
    mae.append({
        'original': mae_original,
        'normalized': mae_normalized,
    })
mae = pd.DataFrame(mae)
ax = mae.plot.scatter(x='original', y='normalized')
ax.plot([0.3, 1.0], [0.3, 1.0], color='black')
ax.set_xlabel('MAE of models based on original spectra')
ax.set_ylabel('MAE of models based on normalized spectra')
plt.show()

# mean of MAE
mean_mae = mae.mean()
mean_mae
```

```
original      0.817016
normalized    0.433026
dtype: float64
```

(fig:fdaMoistureMAE)

Fig. 8.8 clearly shows that regression models based on the normalized spectra achieve better performance. The mean absolute error decrease from 0.82 to 0.43.

(iii) We use the last regression model from (ii) to create a plot of actual versus predicted moisture content for the test data.

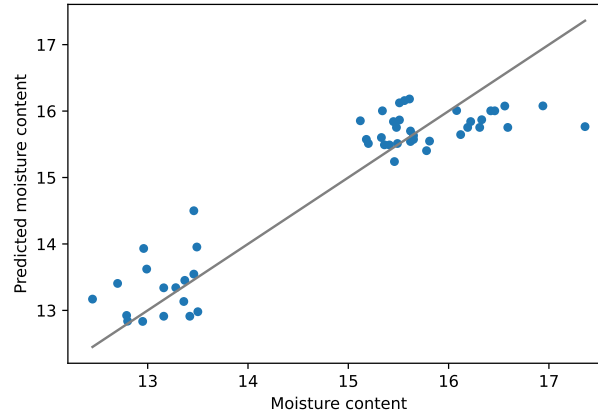


Fig. 8.9 Actual versus predicted moisture content

{fig:fdaMoisturePredictions}

```
y_pred = knn_normalized.predict(X_test)
predictions = pd.DataFrame({'actual': y_test, 'predicted': y_pred})
minmax = [min(*y_test, *y_pred), max(*y_test, *y_pred)]

ax = predictions.plot.scatter(x='actual', y='predicted')
ax.set_xlabel('Moisture content')
ax.set_ylabel('Predicted moisture content')
ax.plot(minmax, minmax, color='grey')
plt.show()
```

Fig. 8.9 shows two clusters of points. One cluster contains the samples with the high moisture content and the other cluster the samples with low moisture content. The clusters are well separated and the predictions are in the typical range for each cluster. However, within a cluster, predictions and actual values are not highly correlated. In other words, while the regression model can distinguish between samples with a high and low moisture content, the moisture content is otherwise not well predicted. There is therefore no advantage of using the regression model compared to the classification model.

{fig:fdaMoisturePredictions}

Solution 8.5 (i) See solution for Exercise 8.3.

{exc:fda-moisture-classification}

(ii) In Python:

```
from skfda.preprocessing.dim_reduction.projection import FPCA

fpca_original = FPCA(n_components=2)
_ = fpca_original.fit(fd)

fpca_normalized = FPCA(n_components=2)
_ = fpca_normalized.fit(fd_normalized)
```

The projections of the spectra can now be visualized:

```
def plotFPCA(fpca, fd, ax):
    fpca_df = pd.DataFrame(fpca.transform(fd))
```

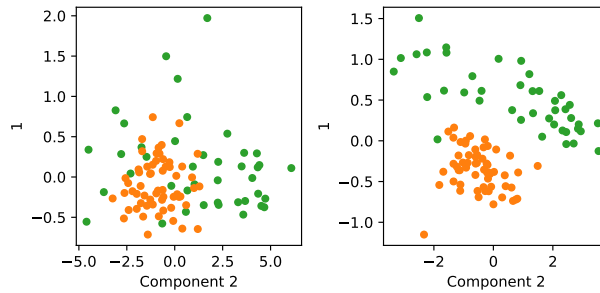


Fig. 8.10 Projection of spectra onto first two principal components. Left: original spectra, right: normalized spectra

{fig:fdaMoisturePCA}

```
fpca_df.plot.scatter(x=0, y=1,
                    c=['C1' if mc == 'high' else 'C2' for mc in moisture_class], ax=ax)
ax.set_xlabel('Component 1')
ax.set_xlabel('Component 2')

fig, axes = plt.subplots(ncols=2, figsize=[6, 3])
plotFPCA(fpca_original, fd, axes[0])
plotFPCA(fpca_normalized, fd_normalized, axes[1])
plt.tight_layout()
```

{fig:fdaMoisturePCA}

Fig. 8.10 compares the PCA projections for the original and normalized data. We can see that the second principal component clearly separates the two moisture content classes for the normalized spectra. This is not the case for the original spectra.

Solution 8.6 (i) We demonstrate a solution to this exercise using two blog posts preprocessed and included in the `mistat` package. The content of the posts was converted text with each paragraph on a line. The two blog posts can be loaded as follows:

```
from mistat.nlp import globalWarmingBlogs
blogs = globalWarmingBlogs()
```

The variable `blogs` is a dictionary with labels as keys and text as values. We next split the data into a list of labels and a list of non-empty paragraphs.

```
paragraphs = []
labels = []
for blog, text in blogs.items():
    for paragraph in text.split('\n'):
        paragraph = paragraph.strip()
        if not paragraph: # ignore empty paragraphs
            continue
        paragraphs.append(paragraph)
        labels.append(blog)
```

(ii) Using `CountVectorizer`, transform the list of paragraphs into a document-term matrix (DTM).

```
import re
from sklearn.feature_extraction.text import CountVectorizer

def preprocessor(text):
    text = text.lower()
    text = re.sub(r'\d[\d,]*', '', text)
    text = '\n'.join(line for line in text.split('\n')
                      if not line.startswith('ntsb'))
    return text

vectorizer = CountVectorizer(preprocessor=preprocessor,
                             stop_words='english')
counts = vectorizer.fit_transform(paragraphs)

print('shape of DTM', counts.shape)
print('total number of terms', np.sum(counts))
```

```
| shape of DTM (123, 1025)
| total number of terms 2946
```

The ten most frequently occurring terms are:

```
termCounts = np.array(counts.sum(axis=0)).flatten()
topCounts = termCounts.argsort()
terms = vectorizer.get_feature_names_out()
for n in reversed(topCounts[-10:]):
    print(f'{terms[n]:14s} {termCounts[n]:3d}')
```

```
| global          63
| climate         59
| warming         57
| change          55
| ice             35
| sea             34
| earth           33
| ocean           29
| temperatures    28
| heat            25
```

(iii) Conversion of counts using TF-IDF.

```
from sklearn.feature_extraction.text import TfidfTransformer

tfidfTransformer = TfidfTransformer(smooth_idf=False, norm=None)
tfidf = tfidfTransformer.fit_transform(counts)
```

(iv)

```
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import Normalizer
svd = TruncatedSVD(5)
norm_tfidf = Normalizer().fit_transform(tfidf)
lsa_tfidf = svd.fit_transform(norm_tfidf)
```

We can analyze the loadings to get an idea of topics.

```
terms = vectorizer.get_feature_names_out()
data = {}
for i, component in enumerate(svd.components_, 1):
```

```

compSort = component.argsort()
idx = list(reversed(compSort[-10:]))
data[f'Topic {i}'] = [terms[n] for n in idx]
data[f'Loading {i}'] = [component[n] for n in idx]
df = pd.DataFrame(data)

```

```

print("\\tiny")
print(df.style.format(precision=2).hide(axis='index').to_latex(hrules=True))
print("")

```

Topic 1	Loading 1	Topic 2	Loading 2	Topic 3	Loading 3	Topic 4	Loading 4	Topic 5	Loading 5
change	0.24	ice	0.39	sea	0.25	extreme	0.54	snow	0.40
climate	0.24	sea	0.34	earth	0.23	events	0.30	cover	0.24
global	0.23	sheets	0.27	energy	0.21	heat	0.24	sea	0.16
sea	0.22	shrinking	0.19	light	0.21	precipitation	0.20	climate	0.12
warming	0.21	level	0.17	gases	0.19	earth	0.13	level	0.12
ice	0.20	arctic	0.15	ice	0.18	light	0.13	temperatures	0.12
temperature	0.18	ocean	0.13	infrared	0.17	energy	0.12	decreased	0.11
ocean	0.16	declining	0.10	greenhouse	0.15	gases	0.12	temperature	0.11
earth	0.16	levels	0.08	level	0.15	greenhouse	0.11	increase	0.10
extreme	0.15	glaciers	0.07	atmosphere	0.12	infrared	0.10	rise	0.10

We can identify topics related to sea warming, ice sheets melting, greenhouse effect, extreme weather events, and hurricanes.

(v) Repeat the analysis requesting 10 components in the SVD.

```

svd = TruncatedSVD(10)
norm_tfidf = Normalizer().fit_transform(tfidf)
lsa_tfidf = svd.fit_transform(norm_tfidf)

```

We now get the following topics.

```

terms = vectorizer.get_feature_names_out()
data = {}
for i, component in enumerate(svd.components_, 1):
    compSort = component.argsort()
    idx = list(reversed(compSort[-10:]))
    data[f'Topic {i}'] = [terms[n] for n in idx]
    data[f'Loading {i}'] = [component[n] for n in idx]
df = pd.DataFrame(data)

```

Topic 1	Loading 1	Topic 2	Loading 2	Topic 3	Loading 3	Topic 4	Loading 4	Topic 5	Loading 5
change	0.24	ice	0.39	sea	0.25	extreme	0.54	snow	0.39
climate	0.24	sea	0.35	earth	0.24	events	0.31	cover	0.23
global	0.23	sheets	0.27	energy	0.21	heat	0.24	sea	0.17
sea	0.22	shrinking	0.19	light	0.21	precipitation	0.20	level	0.13
warming	0.21	level	0.17	gases	0.19	light	0.13	temperatures	0.12
ice	0.20	arctic	0.15	ice	0.18	earth	0.13	climate	0.12
temperature	0.18	ocean	0.13	infrared	0.17	energy	0.12	hurricanes	0.12
ocean	0.16	declining	0.10	greenhouse	0.16	gases	0.12	decreased	0.11
earth	0.16	levels	0.08	level	0.15	greenhouse	0.11	increase	0.10
extreme	0.15	glaciers	0.07	atmosphere	0.12	infrared	0.10	rise	0.10

Topic 6	Loading 6	Topic 7	Loading 7	Topic 8	Loading 8	Topic 9	Loading 9	Topic 10	Loading 10
sea	0.38	snow	0.34	ocean	0.38	glaciers	0.36	responsibility	0.30
level	0.36	ocean	0.26	hurricanes	0.23	water	0.24	authorities	0.24
rise	0.17	cover	0.24	acidification	0.18	retreat	0.24	heat	0.23
extreme	0.14	acidification	0.17	glaciers	0.17	glacial	0.21	wildfires	0.20
events	0.11	extreme	0.16	water	0.13	months	0.16	pollution	0.19
global	0.11	pollution	0.13	waters	0.12	summer	0.14	arctic	0.13
hurricanes	0.11	carbon	0.13	temperatures	0.11	going	0.13	personal	0.12
temperature	0.10	decreased	0.12	reefs	0.09	plants	0.13	carbon	0.11
impacts	0.09	point	0.11	coral	0.09	animals	0.11	percent	0.11
coastal	0.08	warming	0.10	warmer	0.09	stream	0.11	reduce	0.10

The first five topics are identical to the result in (iv). This is an expected property of the SVD.

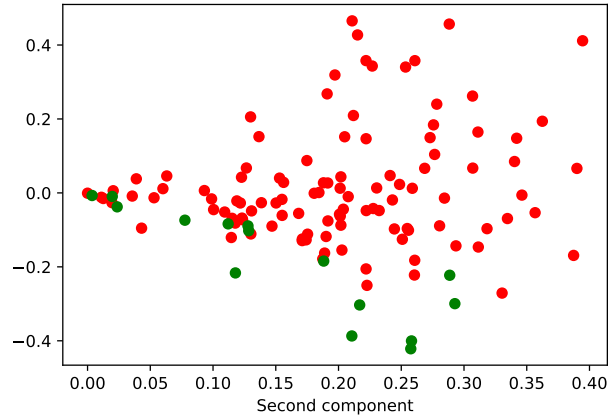


Fig. 8.11 Projection of the documents onto first two singular values. Red: blog post 1, green: blog post 2

(fig:nlpSVD)

(fig:nlpSVD)

(vi) Fig. 8.11 shows the individual documents projected onto the first two singular values of the LSA. Based on this visualization, we can say that the two documents discuss different aspects of global warming and that blog post 1 contains more details about the area.

```
fig, ax = plt.subplots()
blog1 = [label == 'blog-1' for label in labels]
blog2 = [label == 'blog-2' for label in labels]
ax.plot(lsa_tfidf[blog1, 0], lsa_tfidf[blog1, 1], 'ro')
ax.plot(lsa_tfidf[blog2, 0], lsa_tfidf[blog2, 1], 'go')
ax.set_xlabel('First component')
ax.set_xlabel('Second component')
plt.show()
```

Solution 8.7 We follow the same procedure as in Exercise 8.6 using a set of three articles preprocessed and included in the `mistat` package.

(exc:nlp-topic-1)

```
from mistat.nlp import covid19Blogs
blogs = covid19Blogs()
```

Determine DTM using paragraphs as documents:

```
paragraphs = []
labels = []
for blog, text in blogs.items():
    for paragraph in text.split('\n'):
        paragraph = paragraph.strip()
        if not paragraph:
            continue
        paragraphs.append(paragraph)
        labels.append(blog)

def preprocessor(text):
```

```

text = text.lower()
text = re.sub(r'\d{1,3}', '', text)
text = '\n'.join(line for line in text.split('\n')
                  if not line.startswith('ntsb'))
return text

vectorizer = CountVectorizer(preprocessor=preprocessor, stop_words='english')
counts = vectorizer.fit_transform(paragraphs)

```

TF-IDF transformation

```

tfidfTransformer = TfidfTransformer(smooth_idf=False, norm=None)
tfidf = tfidfTransformer.fit_transform(counts)

```

Latent semantic analysis (LSA)

```

svd = TruncatedSVD(10)
tfidf = Normalizer().fit_transform(tfidf)
lsa_tfidf = svd.fit_transform(tfidf)

```

Topics:

```

terms = vectorizer.get_feature_names_out()
data = {}
for i, component in enumerate(svd.components_, 1):
    compSort = component.argsort()
    idx = list(reversed(compSort[-10:]))
    data[f'Topic {i}'] = [terms[n] for n in idx]
    data[f>Loading {i}'] = [component[n] for n in idx]
df = pd.DataFrame(data)

```

Topic 1	Loading 1	Topic 2	Loading 2	Topic 3	Loading 3	Topic 4	Loading 4	Topic 5	Loading 5
labour	0.29	south	0.27	economic	0.24	capacity	0.21	covid	0.22
covid	0.22	labour	0.27	percent	0.23	financial	0.20	america	0.22
impact	0.19	north	0.22	covid	0.19	firms	0.19	reset	0.21
market	0.19	differences	0.20	gdp	0.17	household	0.17	latin	0.20
south	0.18	americas	0.17	impact	0.15	international	0.17	needs	0.18
america	0.16	channel	0.16	imf	0.14	access	0.14	economic	0.14
pandemic	0.15	agenda	0.14	social	0.12	largely	0.14	social	0.14
channel	0.15	covid	0.13	pre	0.12	depends	0.14	asymmetric	0.11
economic	0.14	post	0.11	world	0.11	state	0.13	region	0.11
north	0.14	impact	0.09	growth	0.11	support	0.13	labor	0.10
Topic 6	Loading 6	Topic 7	Loading 7	Topic 8	Loading 8	Topic 9	Loading 9	Topic 10	Loading 10
economic	0.26	covid	0.24	self	0.22	agenda	0.21	state	0.15
channel	0.20	asymmetric	0.21	employed	0.19	post	0.19	policies	0.14
social	0.16	occupations	0.20	unbearable	0.19	channel	0.18	needs	0.13
market	0.14	consequences	0.20	lightness	0.19	region	0.15	self	0.12
recovery	0.12	economic	0.17	employment	0.13	labour	0.12	capacity	0.12
labour	0.11	transition	0.13	countries	0.13	pandemic	0.12	unbearable	0.12
consequences	0.10	occupation	0.13	economic	0.13	new	0.09	lightness	0.12
covid	0.09	americas	0.13	channel	0.12	latin	0.09	covid	0.12
governments	0.09	north	0.13	argentina	0.12	given	0.08	employed	0.12
agenda	0.09	differences	0.12	social	0.12	occupations	0.08	reset	0.11

Looking at the different loadings, we can see different topics emerging.

In Fig. 8.12, we can see that the paragraphs in the article show more overlap compared to what we've observed in the Exercise 8.6.

(fig:nlpSVD-covid)
(exc:nlp-topic-1)

```

fig, ax = plt.subplots()
for blog in blogs:
    match = [label == blog for label in labels]
    ax.plot(lsa_tfidf[match, 0], lsa_tfidf[match, 1], 'o', label=blog)

```

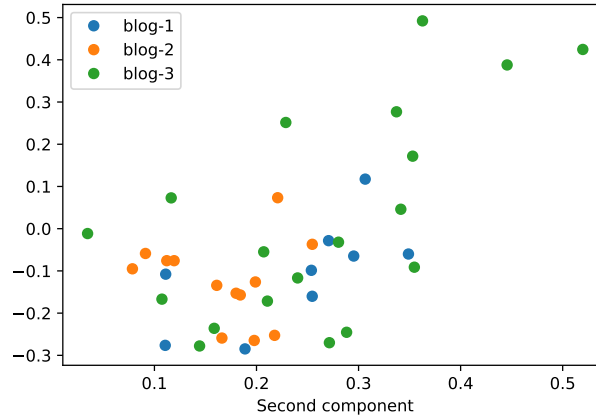


Fig. 8.12 Projection of the documents onto first two singular values. Red: blog post 1, green: blog post 2

(fig:nlpSVD-covid)

```
ax.legend()
ax.set_xlabel('First component')
ax.set_xlabel('Second component')
plt.show()
```

Solution 8.8 (i) Load and preprocess the data

```
data = mistat.load_data('LAPTOP_REVIEWS')
data['Review'] = data['Review title'] + ' ' + data['Review content']
reviews = data.dropna(subset=['User rating', 'Review title', 'Review content'])
```

(ii) Convert the text representation into a document term matrix (DTM).

```
import re
from sklearn.feature_extraction.text import CountVectorizer
def preprocessor(text):
    text = text.lower()
    text = re.sub(r'\d[\d,]*', '', text)
    return text

vectorizer = CountVectorizer(preprocessor=preprocessor,
                             stop_words='english')
counts = vectorizer.fit_transform(reviews['Review'])
print('shape of DTM', counts.shape)
print('total number of terms', np.sum(counts))
```

```
| shape of DTM (7433, 12823)
| total number of terms 251566
```

(iii) Convert the counts in the document term matrix (DTM) using TF-IDF.

```
from sklearn.feature_extraction.text import TfidfTransformer

tfidfTransformer = TfidfTransformer(smooth_idf=False, norm=None)
tfidf = tfidfTransformer.fit_transform(counts)
```

(iv) Using scikit-learn's `TruncatedSVD` method, we convert the sparse tfidf matrix to a denser representation.

```
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import Normalizer
svd = TruncatedSVD(20)
tfidf = Normalizer().fit_transform(tfidf)
lsa_tfidf = svd.fit_transform(tfidf)
print(lsa_tfidf.shape)
```

```
| (7433, 20)
```

(v) We use logistic regression to classify reviews with a user rating of five as positive and negative otherwise.

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.model_selection import train_test_split

outcome = ['positive' if rating == 5 else 'negative'
            for rating in reviews['User rating']]

# split dataset into 60% training and 40% test set
Xtrain, Xtest, ytrain, ytest = train_test_split(lsa_tfidf, outcome,
                                                test_size=0.4, random_state=1)

# run logistic regression model on training
logit_reg = LogisticRegression(solver='lbfgs')
logit_reg.fit(Xtrain, ytrain)

# print confusion matrix and accuracy
accuracy = accuracy_score(ytest, logit_reg.predict(Xtest))
print(accuracy)
confusion_matrix(ytest, logit_reg.predict(Xtest))
```

```
| 0.769334229993275
```

```
| array([[ 861,  385],
|        [ 301, 1427]])
```

The predicted accuracy of the classification model is 0.77.