Ron S. Kenett, Shelemyahu Zacks, Peter Gedeck

Industrial Statistics: A Computer Based Approach with Python

Solutions

September 27, 2023

Springer Nature

Contents

Chapter 1
Introduction to industrial statistics

For the most recent version of the solution manual, go to `https://gedeck.github.io/mistat-code-solutions/IndustrialStatistics/`.

Exercise 1.1 Describe three work environments where quality is assured by 100% inspection of outputs (as opposed to process control).

Solution 1.1 Examples of 100% inspection:

1. Airplane pilots rely heavily on checklists before takeoff. These lists provide a systematic check on safety and functional items that can be verified by relatively simple pilot inspection. The lists are used in every flight.
2. The education system relies on tests to evaluate what students learn in various courses and classes. Tests are administered to all students attending a class. The grades received on these tests reflect the student's ability, the course syllabus and training material, the instructor's ability, the classroom environment, etc.
3. Production lines of electronic products typically include automatic testers that screen products as PASS or FAIL. Failed products are directed to rework departments where the problems are diagnosed and fixed before the products are retested.

Exercise 1.2 Search periodicals, such as Business Week, Fortune, Time and Newsweek and newspapers such as the New York Times and Wall Street Journal for information on quality initiatives in service, healthcare, governmental and industrial organizations. Summarize three such initiatives indicating what was done and what were the concrete outcomes of these initiatives.

Solution 1.2 Possible articles to evaluate are:

- `https://www.industryweek.com/operations/quality/article/21213860/quality-initiatives-deserve-better-from-industry-40`
- `https://www.nytimes.com/2017/03/13/upshot/lousy-customer-service-a-better-way-in-health-care.html`
- `https://www.nytimes.com/2007/02/08/business/08scene.html`
- `https://www.wsj.com/articles/BL-CIOB-6959`

- https://www.newsweek.com/trends-opportunities-threats-manufacturing-operations-management-software-market-1696962

Exercise 1.3 Provide examples of the three types of production systems.

(a) Continuous flow production.
(b) Discrete mass production
(c) Industry 4 production

Solution 1.3 Examples of production systems:

(a) *Continuous flow production*: Polymerization is used in the petroleum industry and in the manufacture of synthetic rubber. In polymerization, a reaction occurs in which two or more molecules of the same substance combined to form a compound from which the original substance may or may not be regenerated. Typical parameters affecting the process are feed rate, polymerizer temperature, and sludge levels in the separator kettles. Typical measured responses are yield, concentration, color, and texture.
(b) *Discrete mass production*: A college or university is, in fact, a discrete mass production system where students are acquiring knowledge and experience through various combinations of classes, laboratories, projects, and homework assignments.
(c) *Job shop*: Modern print houses are typical job shop operations. Customers provide existing material, computer files, or just an idea. The print house is responsible for producing hard copies in various sizes, colours, and quantities. The process involves several steps combining human labor with machine processing.

Exercise 1.4 What management approach cannot work with continuous flow production?

Solution 1.4 Sample inspection for acceptance sampling procedures determining the quality of batches of units.

Exercise 1.5 What management approach characterizes

(a) A school system?
(b) A group of scouts?
(c) A football team?

Solution 1.5 (a) *A school system?*: In most cases, school systems rely on individual learning combined with 100% inspection. Many schools also encourage cooperative learning efforts such as team projects.
(b) *A military unit?*: Traditionally, this is a classical organization operating with strict regulations and 100% inspection. Improvements are achieved through training and exercises with comprehensive performance reviews.
(c) *A football team?*: Serious sports teams invest huge efforts in putting together a winning team combination. This includes screening and selection, team building exercises, on-going feedback during training and games, and detailed analysis and review of self and other teams performance.

Exercise 1.6 Provide examples of how you, personally, apply inspection, process control or quality by design approaches.

(a) As a student.
(b) In your parents' house.
(c) With your friends

Solution 1.6 Examples of how you personally apply the four management approaches:

a) As a student:With an old car that you maintain with a "don't fix what ain't broke" strategy you are fire fighting. Reviewing your bank account statements is typically done on a 100% basis. Serious learning involves on going process control to assure success in the final exam. Proper preparation, ahead of time, through self study and learning from the experience of others is a form of ensuring quality by design.
b) In your parent's home: This is a personal exercise for self thinking.
c) With your friends: This is also a personal exercise for self thinking.

Exercise 1.7 Evaluate the information quality of a case study provided by your instructor.

Solution 1.7 Three case studies are provided in slides 20, 21 and 22 in `https://ceeds.unimi.it/wp-content/uploads/2020/02/Kenett_Analytics_2020-1.pdf`. Their information quality assessment is provided in slides 45-59.

Chapter 2
Basic Tools and Principles of Process Control

For the most recent version of the solution manual, go to `https://gedeck.github.io/mistat-code-solutions/IndustrialStatistics/`.

Import required modules and define required functions

```
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
from statsmodels.graphics.mosaicplot import mosaic
import seaborn as sns
import matplotlib.pyplot as plt
import pwlf

import mistat
```

Exercise 2.1 Use Python and dataset **OELECT.csv** to chart the individual electrical outputs of the 99 circuits. Do you observe any trend or non-random pattern in the data? [Use under SPC the option of Individual chart. For Mu and Sigma use "historical" values, which are $\bar{X}$ and S.]

Solution 2.1 The following Python code creates the chart shown in Fig. 2.1.

```
oelect = mistat.load_data('OELECT')
qcc = mistat.QualityControlChart(oelect, qcc_type='xbarone',
                                 std_dev='SD')
qcc.plot()
plt.show()
```

It seems that the data are randomly distributed around a mean value of 219.25.

Exercise 2.2 Chart the individual variability of the length of steel rods, in dataset **STEELROD.csv**. Is there any perceived assignable cause of non-randomness?

Solution 2.2 The following Python code creates the chart shown in Fig. 2.2.

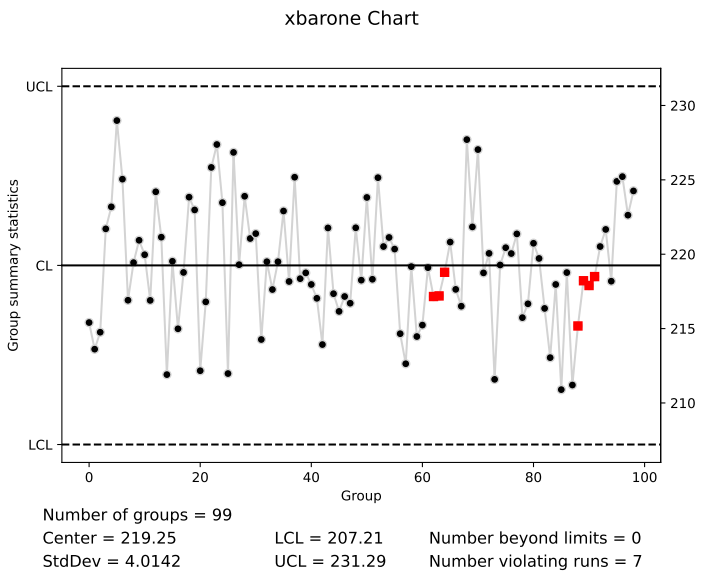


Fig. 2.1: X-barone chart of **OELECT.csv** dataset

```
steelrod = mistat.load_data('STEELROD')
qcc = mistat.QualityControlChart(steelrod, qcc_type='xbarone',
                                 std_dev='SD')
qcc.plot()
plt.show()
```

It seems that the data are randomly distributed around a mean value of 19.89.

Exercise 2.3 Examine the chart of the previous exercise for possible patterns of non-randomness.

Solution 2.3 No patterns of non-randomness are apparent.

Exercise 2.4 Test the data in dataset **OTURB2.csv** for lack of randomness. In this dataset we have three columns. In the first we have the sample size. In the second and third we have the sample means and standard deviation. Chart the individual means. For the historical mean use the mean of column `xbar`. For historical standard deviation use $(\hat{\sigma}^2/5)^{1/2}$, where $\hat{\sigma}^2$ is the pooled sample **variance**.

Solution 2.4 The following Python code creates the chart shown in Fig. 2.3.

```
oturb2 = mistat.load_data('OTURB2')
# print(oturb2)
sd = np.sqrt(oturb2['xbar'].var() / 5)
center = oturb2['xbar'].mean()
print(sd, center)
qcc = mistat.QualityControlChart(oturb2['xbar'], qcc_type='xbarone',
```

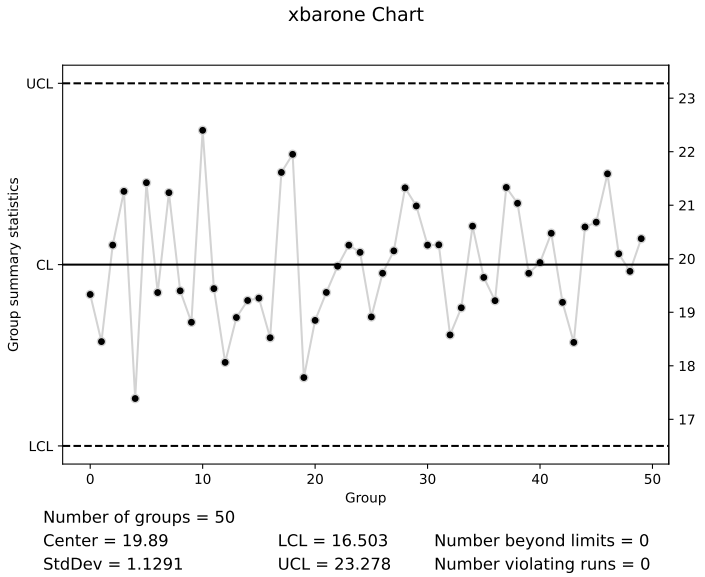


Fig. 2.2: X-barone chart of the **STEELROD.csv** dataset

```
        center=center, std_dev=sd)
qcc.plot()
plt.show()
```

```
0.07199111056234651 0.6526
```

We can see a downward trend in the data.

Exercise 2.5 A sudden change in a process lowers the process mean by one standard deviation. It has been determined that the quality characteristic being measured is approximately normally distributed and that the change had no effect on the process variance:

(a) What percentage of points are expected to fall outside the control limits on the $\bar{X}$-chart if the subgroup size is 4?
(b) Answer the same question for subgroups of size 6.
(c) Answer the same question for subgroups of size 9.

Solution 2.5 If $\bar{X}_n \sim N(\mu - \sigma, \frac{\sigma}{\sqrt{n}})$ the proportion of points expected to fall outside the control limits is

$$\pi_n \equiv \Pr\left\{\bar{X}_n < \mu - \frac{3\sigma}{\sqrt{n}}\right\} + \Pr\left\{\bar{X}_n > \mu + \frac{3\sigma}{\sqrt{n}}\right\}.$$

(a) For $n = 4$, $\pi_n = 0.159$; **(b)** For $n = 6$, $\pi_n = 0.291$; **(c)** For $n = 9$, $\pi_n = 0.500$.

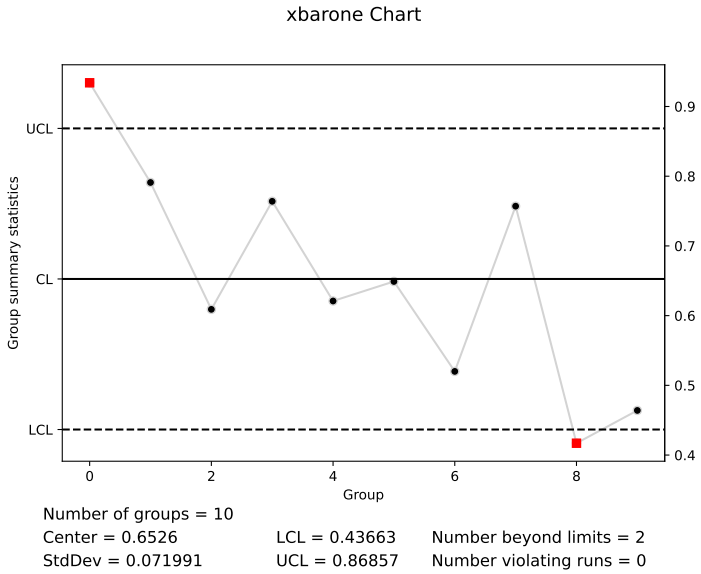


Fig. 2.3: X-barone chart of the **OTURB2.csv** dataset

Exercise 2.6 Make capability analysis of the electric output (volts) of 99 circuits in dataset **OELECT.csv**, with target value of $\mu_0 = 220$ and $LSL = 210$, $USL = 230$.

Solution 2.6 In Python:

```
oelect = mistat.load_data('OELECT')

qcc = mistat.QualityControlChart(oelect, qcc_type='xbarone',
                                 std_dev='SD')
pc = mistat.ProcessCapability(qcc, spec_limits = [210, 230])
pc.plot()
plt.show()
pc.summary()
```

```
Process Capability Analysis

Number of obs = 99           Target = 220.00
       Center = 219.25          LSL = 210.00
       StdDev = 4.014219        USL = 230.00

Capability indices:

        Value     2.5%   97.5%
Cp      0.8304  0.7142  0.9463
Cp_l    0.7679  0.6622  0.8737
Cp_u    0.8928  0.7743  1.0113
Cp_k    0.7679  0.6420  0.8939
Cpm     0.8162  0.7007  0.9315

Exp<LSL   1%   Obs<LSL   0%
Exp>USL   0%   Obs>USL   0%
```

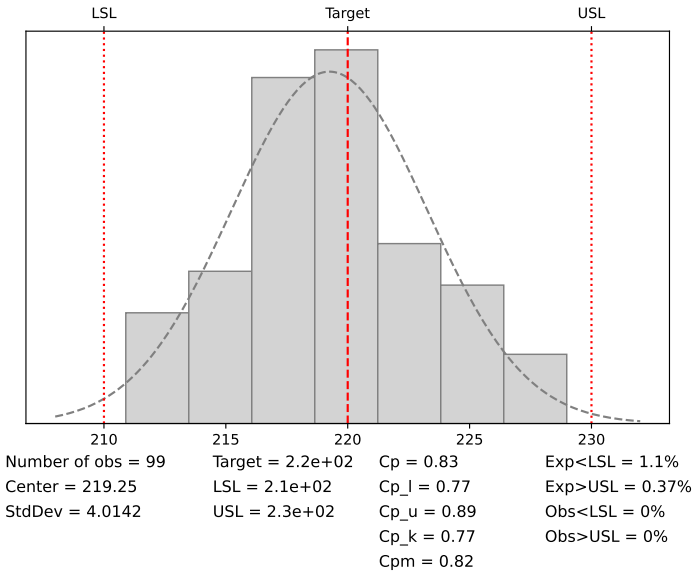


Fig. 2.4: Process capability analysis of **OELECT.csv** dataset

Performing the capability analysis (see Fig. 2.4) we found that $C_p = 0.83$. Although $C_p = 0.83$, there is room for improvement by designing the process with reduced variability. See Chap. 5.

Exercise 2.7 Estimate the capability index C_{pk} for the output of the electronic circuits, based on dataset **OELECT.csv** when $LSL = 210$ and $USL = 230$. Determine the point estimate as well as its confidence interval, with confidence level 0.95.

Solution 2.7 Using the `mistat` package, we get:

```
oelect = mistat.load_data('OELECT')
qcc = mistat.QualityControlChart(oelect, qcc_type='xbarone',
    std_dev=4.004, center=219.25)
pc = mistat.ProcessCapability(qcc, spec_limits = [210, 230],
                              confidence_level=0.975)
pc.summary()
pc = mistat.ProcessCapability(qcc, spec_limits = [210, 230],
                              confidence_level=0.95)
pc.summary()
```

```
Process Capability Analysis

Number of obs = 99           Target = 220.00
       Center = 219.25          LSL = 210.00
       StdDev = 4.004000        USL = 230.00

Capability indices:

        Value     2.5%   97.5%
Cp      0.8325  0.7002  0.9663
```

```
Cp_l    0.7701  0.6438  0.8963
Cp_u    0.8949  0.7535  1.0364
Cp_k    0.7701  0.6257  0.9144
Cpm     0.8183  0.6868  0.9513

Exp<LSL   1%   Obs<LSL   0%
Exp>USL   0%   Obs>USL   0%
Process Capability Analysis

Number of obs = 99          Target = 220.00
       Center = 219.25         LSL = 210.00
       StdDev = 4.004000       USL = 230.00

Capability indices:

        Value     2.5%   97.5%
Cp      0.8325  0.7161  0.9488
Cp_l    0.7701  0.6641  0.8760
Cp_u    0.8949  0.7762  1.0136
Cp_k    0.7701  0.6438  0.8963
Cpm     0.8183  0.7025  0.9339

Exp<LSL   1%   Obs<LSL   0%
Exp>USL   0%   Obs>USL   0%
```

With $\bar{X} = 219.25$, $S = 4.004$, $n = 99$, $\xi_U = 230$ and $\xi_L = 210$ we obtained $\hat{C}_{pl} = 0.77$, $\hat{C}_{pu} = 0.895$ and $\hat{C}_{pk} = 0.77$.

The `mistat` implementation uses the normal distribution to calculate confidence intervals. Using the F-distribution with $F_{0.975}[1, 98] = 5.1982$, we get:

```
def confidenceLimitSL(Cp, n, alpha):
    F = stats.f(1, n-1).ppf(1-(1-alpha)/2)
    a = np.sqrt(F/n) * np.sqrt(Cp**2/2 + (1 - F/(2*n))/9)
    b = 1 - F/(2*n)
    return (Cp - a) / b, (Cp + a )/ b

n = len(oelect)
rho_1L, rho_1U = confidenceLimitSL(pc.Cp_l, n, 0.95)
rho_2L, rho_2U = confidenceLimitSL(pc.Cp_u, n, 0.95)
print(rho_1L, rho_1U)
print(rho_2L, rho_2U)
```

```
0.6413037077789147 0.9402124117777945
0.7514350333838329 1.0865431596145048
```

The confidence intervals for C_{pl} and C_{pu}, at 0.95 confidence level, are:

	Lower Limit	Upper Limit
C_{pl}	0.6413	0.9402
C_{pu}	0.7514	1.0865
C_{pk}	0.6413	0.9402

The 0.95-Confidence interval for C_{p_k} is $(0.6413, 0.9402)$.

Exercise 2.8 Estimate the capability index for the steel rods, given in dataset **STEELROD.csv**, when the length specifications are $\xi_L = 19$ and $\xi_U = 21$ [cm] and the level of confidence is $1 - \alpha = 0.95$.

Solution 2.8 In Python:

```
steelrod = mistat.load_data('STEELROD')
qcc = mistat.QualityControlChart(steelrod, qcc_type='xbarone',
                                 std_dev='SD')
pc = mistat.ProcessCapability(qcc, spec_limits = [19, 21],
                              confidence_level=0.95)
pc.summary()
```

```
Process Capability Analysis

Number of obs = 50           Target = 20.00
       Center = 19.89           LSL = 19.00
       StdDev = 1.129109        USL = 21.00

Capability indices:

        Value     2.5%   97.5%
Cp      0.2952  0.2369  0.3534
Cp_l    0.2628  0.1738  0.3518
Cp_u    0.3276  0.2329  0.4224
Cp_k    0.2628  0.1568  0.3688
Cpm     0.2938  0.2361  0.3514

Exp<LSL  22%   Obs<LSL  22%
Exp>USL  16%   Obs>USL  20%
```

Using the **STEELROD.csv** data with $\xi_L = 19$ and $\xi_U = 21$ we obtained $\hat{C}_p = 0.295$, $\hat{C}_{pu} = 0.328$, $\hat{C}_{pl} = 0.263$ and $\hat{C}_{pk} = 0.263$. We use the function from the previous exercise to calculat the confidence interval.

```
n = len(steelrod)
rho_1L, rho_1U = confidenceLimitSL(pc.Cp_l, n, 0.95)
rho_2L, rho_2U = confidenceLimitSL(pc.Cp_u, n, 0.95)
print(rho_1L, pc.Cp_l, rho_1U)
print(rho_2L, pc.Cp_u, rho_2U)
```

```
0.14851207708654524 0.26280004915567645 0.40677992256113854
0.2084480020651775 0.32763581095436156 0.4838408510212421
```

A 0.95-confidence interval for C_{pk} is $(0.1485, 0.4068)$.

Exercise 2.9 The specification limits of the piston cycle times are 0.05 ± 0.01 seconds. Generate 20 cycle times at the lower level of the 7 control parameters:

(a) Compute C_p and C_{pk}.
(b) Compute a 95% confidence interval for C_{pk}.

Generate 20 cycle times at the upper level of the 7 control factors:

(c) Recompute C_p and C_{pk}.
(d) Recompute a 95% confidence interval for C_{pk}.
(e) Is there a significant difference in process capability between lower and upper operating levels in the piston simulator?

Solution 2.9 Simulate 20 observations at the low level of the 7 control parameters.

```
settings = {'m': 30, 's': 0.005, 'k': 1000, 't': 290,
            'p0': 90_000, 'v0': 0.002, 't0': 340}

simulator = mistat.PistonSimulator(n_simulation=20, n_replicate=1, seed=1,
                                   **settings)
Ps = simulator.simulate()
cycleTime = mistat.qcc_groups(Ps['seconds'], Ps['group'])

qcc = mistat.QualityControlChart(cycleTime, std_dev= Ps['seconds'].std())
print(f'Mean    {qcc.center:.4f}')
print(f'Std.Dev {qcc.std_dev:.4f}')
```

```
Mean    0.0257
Std.Dev 0.0078
```

If all the 7 control parameters are at their low level, the mean cycle time of 20 observations is $\bar{X}_{20} = 0.0257$. This value is outside the specification limits, 0.04 and 0.06. Thus the process, at the low parameter values is incapable of satisfying the specs. The standard deviation is $S_{20} = 0.0078$.

(a) Calclate the capability index C_p

```
pc = mistat.ProcessCapability(qcc, spec_limits = [0.04, 0.06],
                              confidence_level=0.95)
print(f'C_p {pc.Cp:.3f}')
```

```
C_p 0.425
```

If the process mean can be moved to 0.05, the C_p value is 0.425.

(b) A 95% confidence limit for C_{pk} is:

```
print(f'95% confidence limits [{pc.Cp_k_limits[0]:.3f}, {pc.Cp_k_limits[1]:.3f}]')
```

```
95% confidence limits [-0.366, -0.850]
```

Note that the *mistat.QualityControlChart* class calculates the confidence limits using a normal approximation.

(c) Under the high values of the seven control parameters we get

```
settings = {'m': 60, 's': 0.02, 'k': 5_000, 't': 296,
            'p0': 110_000, 'v0': 0.01, 't0': 360}

simulator = mistat.PistonSimulator(n_simulation=20, n_replicate=1, seed=1, **settings)
Ps = simulator.simulate()
cycleTime = mistat.qcc_groups(Ps['seconds'], Ps['group'])
qcc = mistat.QualityControlChart(cycleTime, std_dev= Ps['seconds'].std())
print(f'Mean    {qcc.center:.4f}')
print(f'Std.Dev {qcc.std_dev:.4f}')
```

```
Mean    0.0568
Std.Dev 0.0037
```

$\bar{X}_{20} = 0.0568$ and $S_{20} = 0.0037$. The mean is within the specification limit 0.05 ± 0.01.

For these values, we get the following for the capability indices.

```
pc = mistat.ProcessCapability(qcc, spec_limits = [0.04, 0.06],
                              confidence_level=0.95)
pc.summary()
```

```
Process Capability Analysis

Number of obs = 20           Target = 0.05
       Center = 0.06            LSL = 0.04
       StdDev = 0.003720        USL = 0.06

Capability indices:

        Value     2.5%   97.5%
Cp      0.8960  0.6135  1.1782
Cp_l    1.5090  1.0881  1.9298
Cp_u    0.2830  0.1390  0.4270
Cp_k    0.2830  0.1115  0.4546
Cpm     0.4281  0.2543  0.6020

Exp<LSL   0%   Obs<LSL   0%
Exp>USL  20%   Obs>USL  10%
```

The values are: $C_p = 0.890$ and $C_{pk} = 0.283$.

(d) Under the normal approximation the confidence interval for C_{pk}, at level 0.95, is (0.1115, 0.4546).

(e) As mentioned above, the gas turbine at low control levels is incapable of satisfying the specifications of 0.05 ± 0.01 seconds. At high levels its C_{pk} is not greater than 0.282. In Chap. 5, we study experimental methods for finding the combination of control levels, which maximizes the capability.

Exercise 2.10 A fiber manufacturer has a large contract that stipulates that its fiber, among other properties, has tensile strength greater than 1.800 [grams/fiber] in 95% of the fibers used. The manufacturer states the standard deviation of the process is 0.015 grams.

(a) Assuming a process under statistical control, what is the smallest nominal value of the mean that will assure compliance with the contract?
(b) Given the nominal value in part a) what are the control limits of $\bar{X}$ and S-charts for subgroups of size 6?
(c) What is the process capability, if the process mean is $\mu = 1.82$?

Solution 2.10 (a) 1.8247 calculated in Python

```
std = 0.015

nominal_value = 1.8 + stats.norm(0, std).ppf(0.95)
nominal_value
```

```
1.8246728044042722
```

(b) Using $\bar{\bar{X}} \pm 3\hat{\sigma}/\sqrt{n}$, we get 1.8063 and 1.8430.

```
nominal_value - 3 * std / np.sqrt(6), nominal_value + 3 * std / np.sqrt(6)
```

```
(1.8063016313333984, 1.843043977475146)
```

(c) $C_{pk} = 0.44$.

```
Cpk = (1.82 - 1.80) / (3 * std)
print(Cpk)
```

```
0.44444444444444486
```

Exercise 2.11 The output voltage of a power supply is specified as 350 ± 5 volts DC. Subgroups of four units are drawn from every batch and submitted to special quality control tests. The data from 30 subgroups on output voltage produced $\sum_{i=1}^{30} \bar{X} = 10,950.00$ and $\sum_{i=1}^{30} R_i = 77.32$:

(a) Compute the control limits for $\bar{X}$ and R.
(b) Assuming statistical control and a normal distribution of output voltage, what properties of defective product are being made?
(c) If the power supplies are set to a nominal value of 350 volts, what is now the proportion of defective products?
(d) Compute the new control limits for $\bar{X}$ and R.
(e) If these new control limits are used but the adjustment to 350 volts is not carried out, what is the probability that this fact will not be detected on the first subgroup?
(f) What is the process capability before and after the adjustment of the nominal value to 350 volts? Compute both C_p and C_{pk}.

Solution 2.11 (a) $\bar{\bar{X}} = \frac{10950}{30} = 365$, $\bar{R} = 77.32$, $n = 4$, $\hat{\sigma} = \bar{R}/d(n) = 1.252$. The control limits for $\bar{X}_4$ are

$$UCL = 365 + 3 \times \frac{1.252}{\sqrt{4}} = 366.878 \quad \text{and} \quad LCL = 365 - 3 \times \frac{1.252}{\sqrt{4}} = 363.122.$$

(b) When $X \sim N(\mu = 365, \sigma = 1.252)$,

$$1 - \Pr\{365 < X < 375\} = 1 - \Phi\left(\frac{375 - 365}{1.252}\right) + \Phi\left(\frac{365 - 365}{1.252}\right) = 0.5.$$

(c) When $X \sim N(\mu = 370, \sigma = 1.252)$,

$$1 - \Pr\{365 < X < 375\} = 2 \times \left(1 - \Phi\left(\frac{5}{1.252}\right)\right) = 0.000065.$$

(d) $UCL = 370 + 3 \times \frac{1.252}{\sqrt{4}} = 371.878$ and $LCL = 370 - 3 \times \frac{1.252}{\sqrt{4}} = 368.122$.
(e) When $\bar{X}_4 \sim N(\mu = 365, \sigma = 1.252/\sqrt{4})$,

$$\Pr\{368.122 < \bar{X}_4 < 371.878\} = \Phi\left(\frac{371.878 - 365}{1.252/\sqrt{4}}\right) - \Phi\left(\frac{368.122 - 365}{1.252/\sqrt{4}}\right) = 0.$$

(f) The process capability before and after the adjustment is

	Before	After
C_p	1.331	1.331
C_{pk}	0	1.331

Exercise 2.12 The following data were collected in a circuit pack production plant during October.

	Number of nonconformities
Missing component	293
Wrong component	431
Too much solder	120
Insufficient solder	132
Failed component	183

An improvement team recommended several changes that were implemented in the first week of November. The following data were collected in the second week of November.

	Number of nonconformities
Missing component	34
Wrong component	52
Too much solder	25
Insufficient solder	34
Failed component	18

(a) Construct Pareto charts of the nonconformities in October and the second week of November.
(b) Has the improvement team produced significant differences in the type of nonconformities?

Solution 2.12 (a) Fig. 2.5 is created using the following code

```
october = pd.DataFrame([
  ['Missing component', 293],
  ['Wrong component', 431],
  ['Too much solder', 120],
  ['Insufficient solder', 132],
  ['Failed component', 183],
], columns=['Issue', 'Count'])
november = pd.DataFrame([
  ['Missing component', 34],
  ['Wrong component', 52],
  ['Too much solder', 25],
  ['Insufficient solder', 34],
  ['Failed component', 18],
], columns=['Issue', 'Count'])

def makeParetoChart(data, ax, title):
```

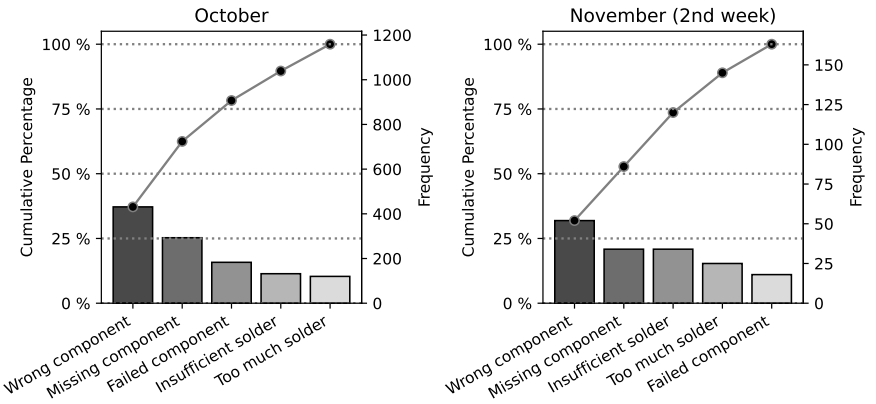


Fig. 2.5: Pareto chart noconformities in October and the second week of November

```
  paretoChart = mistat.ParetoChart(data['Count'], labels=data['Issue'])
  paretoChart.plot(rotation=30, ha='right', ax=ax)
  ax.set_title(title)

fig, axes = plt.subplots(ncols=2, figsize=(8,4))
makeParetoChart(october, axes[0], 'October')
makeParetoChart(november, axes[1], 'November (2nd week)')
fig.suptitle('')
plt.tight_layout()
plt.show()
```

(b) Using the test described in Sect. 2.6 we find that the improvement team has produced significant differences in the type of nonconformities. In particular, the difference in the 'Insufficient solder' category is significant at the 1% level and the difference in the 'Too much solder' category is significant at the 10% level (see Table 2.4). The computations involved are summarized in Table 2.1.

Table 2.1: Computations for Pareto significance analysis (Exercise 2.12)

Category	October	p	November	Expected	(Obs. - Exp.)	Z
Missing	293	0.253	34	41.239	-7.239	-1.304
Wrong	431	0.372	52	60.636	-8.636	-1.399
Too Much	120	0.103	25	16.789	8.211	2.116
Insufficient	132	0.114	34	18.582	15.418	3.800
Failed	183	0.158	18	25.754	-7.754	-1.665
Total	1159		163			

Exercise 2.13 Control charts for $\bar{X}$ and R are maintained on total soluble solids produced at 20°C in parts per million (ppm). Samples are drawn from production

containers every hour and tested in a special test device. The test results are organized into subgroups of $n = 5$ measurements, corresponding to 5 hours of production. After 125 hours of production we find that $\sum_{i=1}^{25} \bar{X}_i = 390.8$ and $\sum_{i=1}^{25} R_i = 84$. The specification on the process states that containers with more than 18 ppm of total soluble solids should be reprocessed.

(a) Compute an appropriate capability index.
(b) Assuming a normal distribution and statistical control, what proportion of the sample measurements are expected to be out of spec?
(c) Compute the control limits for $\bar{X}$ and R.

Solution 2.13 From the data, $\bar{\bar{X}} = 15.632$ ppm and $\bar{R} = 3.36$. An estimate of the process standard deviation is $\hat{\sigma} = \bar{R}/d(5) = 1.4446$.

(a) $C_{pu} = \frac{18-15.632}{3*\hat{\sigma}} = 0.546$.

(b) The proportion expected to be out of spec is 0.051.

(c) The control limits for $\bar{X}$ are $LCL = 13.694$ and $UCL = 17.570$. The control limits for R_5 are $LCL = 0$ and $UCL = 7.106$.

Exercise 2.14 Part I: Run the piston simulator at the lower levels of the 7 piston parameters and generate 100 cycle times. Add 0.02 to the last 50 cycle times.

(a) Compute control limits of $\bar{X}$ and R by constructing subgroups of size 5, and analyze the control charts.

Part II: Randomly shuffle the cycle times using the Python function *random.sample*.

(b) Recompute the control limits of $\bar{X}$ and R and reanalyze the control charts.
(c) Explain the differences between (a) and (b).

Solution 2.14 Part I (a) Fig. 2.6 is created using the following code.

```
settings = {'m': 30, 's': 0.005, 'v0': 0.002, 'k': 1000,
            'p0': 90_000, 't': 290, 't0': 340}

simulator = mistat.PistonSimulator(n_simulation=20, n_replicate=5, seed=1, **settings)
Ps = simulator.simulate()

# Add 0.02 seconds to last 50 simulation results
Ps.loc[50:,'seconds'] = Ps.loc[50:,'seconds'] + 0.02

def makeQCCplot(data, reference, qcc_type, title):
  # convert to groups
  data = mistat.qcc_groups(data['seconds'], data['group'])
  reference = mistat.qcc_groups(reference['seconds'], reference['group'])
  # calculate control limits based on reference data
  qcc_ref = mistat.QualityControlChart(reference, qcc_type=qcc_type)
  qcc = mistat.QualityControlChart(data, qcc_type=qcc_type,
    center=qcc_ref.center, limits=qcc_ref.limits)
  return qcc.plot(title=title)

# create xbar and R control charts
reference = Ps.iloc[:50, ]

makeQCCplot(Ps, reference, 'xbar', 'for cycleTime')
plt.show()
makeQCCplot(Ps, reference, 'R', 'for cycleTime')
plt.show()
```

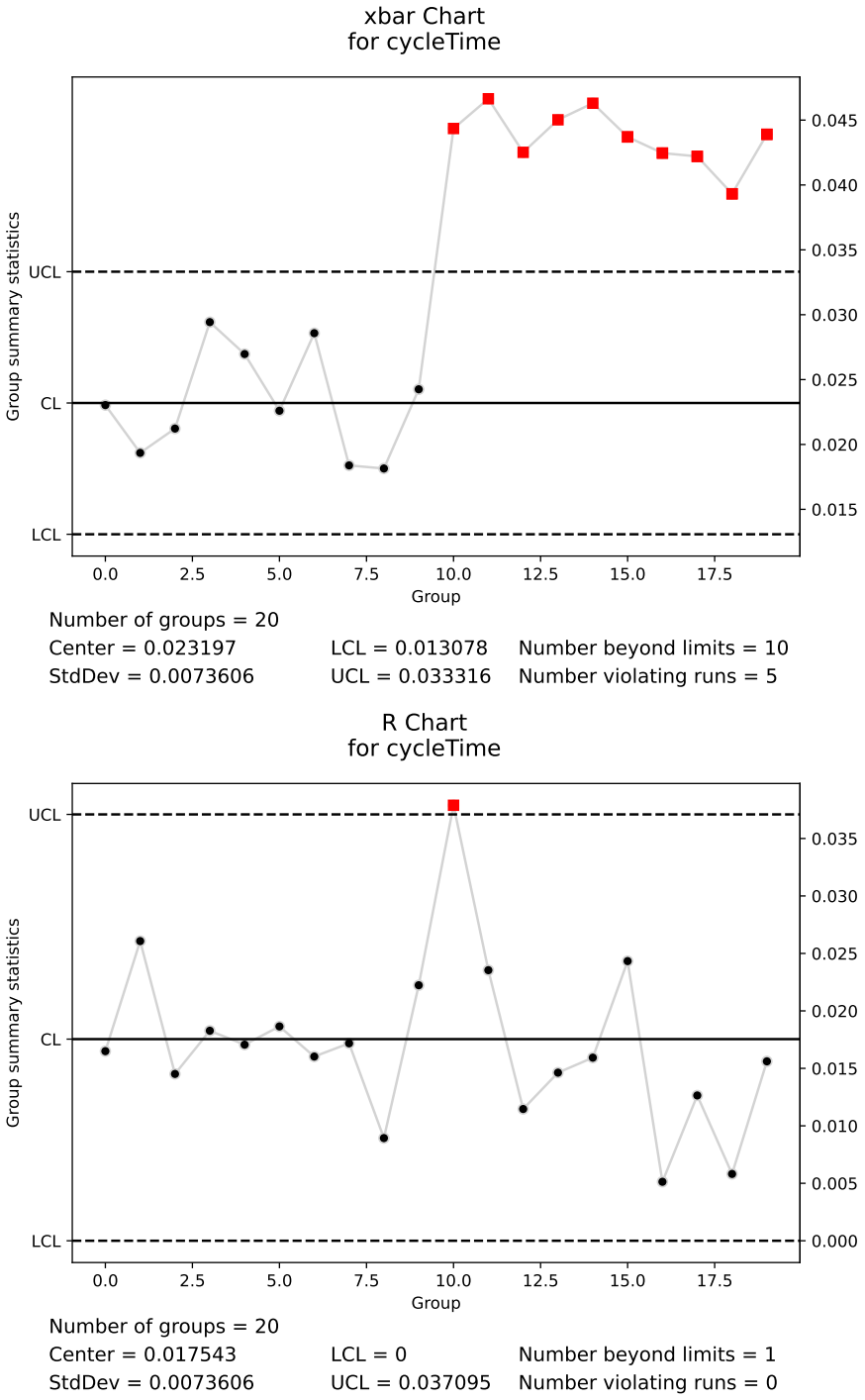


Fig. 2.6: Control charts for time shift Piston simulation (Excercise 2.14)

See the control charts in Fig. 2.6. The increase in cycle times after group 9 is so large that the means of the subgroup are immediately out of the control limits. The shift has no influence on the ranges within subgroups.

Part II (b) Fig. 2.7 is created using the following code.

```
# Ps['seconds'] = random.sample(Ps['seconds'], len(Ps['seconds']))

# alternative approach
# Ps['seconds'] = Ps['seconds'].sample(frac=1).values

makeQCCplot(Ps, Ps, 'xbar', 'for cycleTime')
plt.show()
makeQCCplot(Ps, Ps, 'R', 'for cycleTime')
plt.show()
```

See the control charts in Fig. 2.7. Due to the randomization of the cycle times, the increased values are distributed over all groups and the mean cycle time increases by 0.01 The variability as shown in the R chart is is increased too.

Exercise 2.15 Part I: Run the piston simulator by specifying the 7 piston parameters within their acceptable range. Record the 7 operating levels you used and generate 20 subgroups of size 5.

1. Compute the control limits for $\bar{X}$ and S.

Part II: Rerun the piston simulator at the same operating conditions and generate 20 subgroups of size 10.

2. Recompute the control limits for $\bar{X}$ and S.
3. Explain the differences between (a) and (b).

Solution 2.15 Part I (a): We run the simulation with all 7 control parameters at their lowest value. We obtained $\bar{\bar{X}} = 0.0234$, $\bar{S} = 0.0068$.

```
settings = {'m': 30, 's': 0.005, 'v0': 0.002, 'k': 1000,
            'p0': 90_000, 't': 290, 't0': 340}

simulator = mistat.PistonSimulator(n_simulation=20, n_replicate=5, seed=1, **settings)
Ps = simulator.simulate()
data = mistat.qcc_groups(Ps['seconds'], Ps['group'])
for qcc_type in ('xbar', 'S'):
  qcc = mistat.QualityControlChart(data, qcc_type=qcc_type)
  print(f'{qcc_type:4s} Center {qcc.center:.4f} Control limits ' +
        f'[{qcc.limits.LCL[0]:.5f}, {qcc.limits.UCL[0]:.4f}]')
```

```
xbar Center 0.0234 Control limits [0.01354, 0.0333]
S    Center 0.0068 Control limits [0.00000, 0.0141]
```

Part II (b): We obtained $\bar{\bar{X}} = 0.0253$, $\bar{S} = 0.0093$.

```
simulator = mistat.PistonSimulator(n_simulation=20, n_replicate=10, seed=1, **settings)
Ps = simulator.simulate()
data = mistat.qcc_groups(Ps['seconds'], Ps['group'])
for qcc_type in ('xbar', 'S'):
  qcc = mistat.QualityControlChart(data, qcc_type=qcc_type)
  print(f'{qcc_type:4s} Center {qcc.center:.4f} Control limits ' +
        f'[{qcc.limits.LCL[0]:.5f}, {qcc.limits.UCL[0]:.4f}]')
```

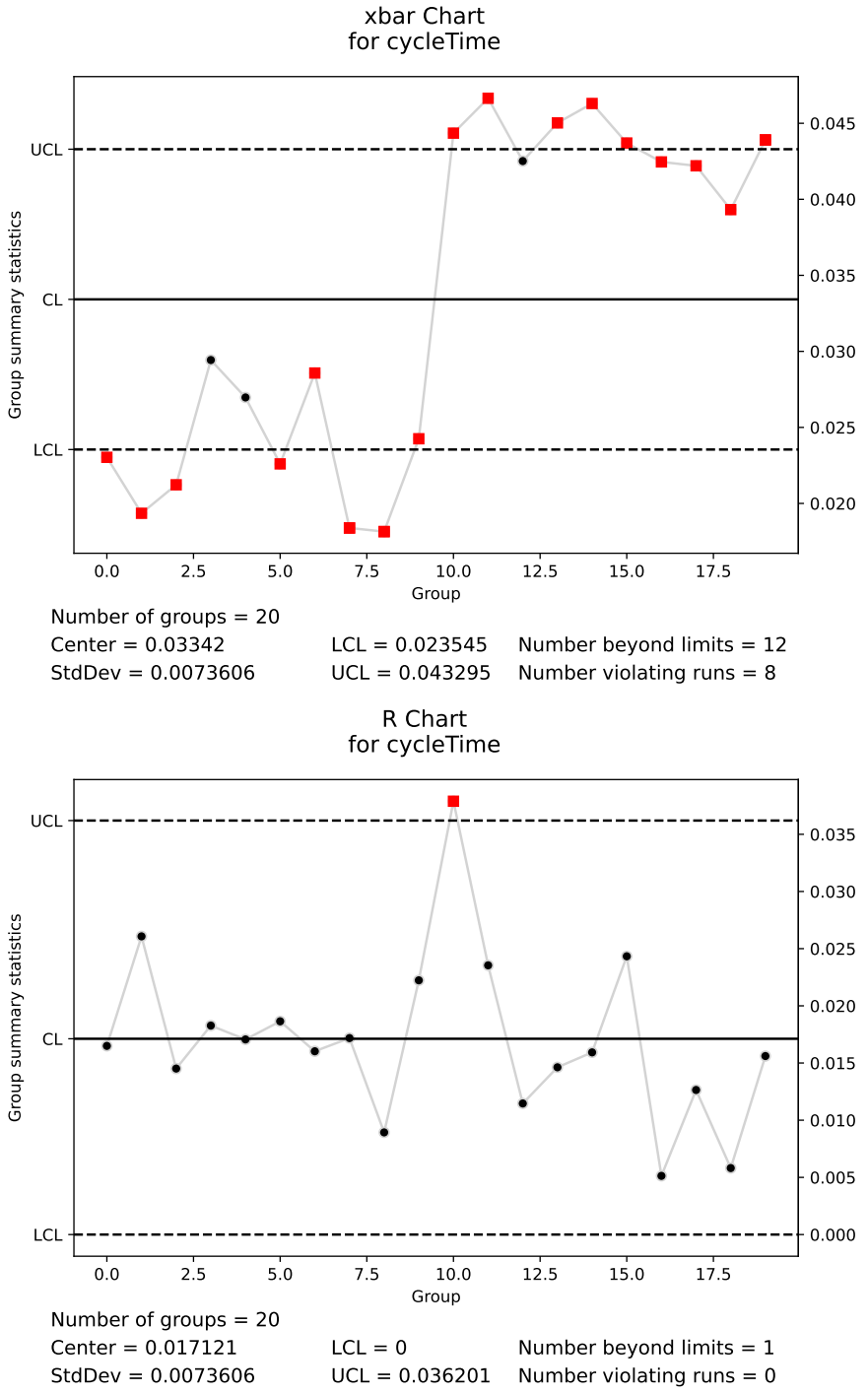


Fig. 2.7: Control charts for randomized cycle time in Piston simulation (Excercise 2.14)

```
xbar Center 0.0253 Control limits [0.01623, 0.0344]
S    Center 0.0093 Control limits [0.00264, 0.0159]
```

(c) We see that the control limits of $\bar{X}_{10}$ and of S_{10} are closer to the center line than those of $\bar{X}_5$ and S_5, respectively.

Exercise 2.16 Repeat the data segment analysis from Sect. 2.8.2 with a piecewise linear regression fit. This can be done using *pwlf.PiecewiseLinFit* initialized with the keyword argument `degree=1`.

(a) Prepare models with a variety of knots.
(b) Compare the results to the step function fits.
(c) When would you use a piecewise linear fit compared to a step function fit?

Solution 2.16 (a) Use code like the following to explore piecewise linear fits to the sensor data using different number of knots.

```
data = mistat.load_data('PROCESS_SEGMENT')

def sensorData(data, label):
    series = data[label]
    return pd.DataFrame({
        'Time': np.arange(len(series)),
        'values': series,
    })

sensorX = sensorData(data, 'X')
sensorZ = sensorData(data, 'Z')

def fitPiecewiseLinearFit(sensor, knots):
    model = pwlf.PiecewiseLinFit(sensor['Time'], sensor['values'], degree=1)
    model.fit(knots)
    return model

modelX = fitPiecewiseLinearFit(sensorX, 6)
modelZ = fitPiecewiseLinearFit(sensorZ, 3)

def plotPiecewiseLinearFit(sensor, model, ax, label):
    for bp in model.fit_breaks[1:-1]:
        ax.axvline(bp, color='lightgrey')
    ax.scatter(sensor['Time'], sensor['values'], color='grey', alpha=0.5)
    ax.plot(sensor['Time'], model.predict(sensor['Time']), color='black')
    ax.set_xlabel('Time')
    ax.set_ylabel(label)
    return ax
fig, axes = plt.subplots(ncols=2, figsize=(8,4))
plotPiecewiseLinearFit(sensorX, modelX, axes[0], 'Sensor X')
plotPiecewiseLinearFit(sensorZ, modelZ, axes[1], 'Sensor Z')
plt.tight_layout()
plt.show()
```

The figure shows results for sensor X and sensor Z data. By trying different number of segments, we can see that we require 6 segments for the sensor X data while 3 segments are sufficient for sensor Z. In contrast to the step function fit, the piecewise linear function doesn't enforce the linear segments to have a slope of 0. While visually, we can see that the segment that is characteristics for a progress under control are essentially horizontal, it is useful to analyze the individual segments. The slope can be extracted from the model using the method *modelX.calc_slopes()*. We get:

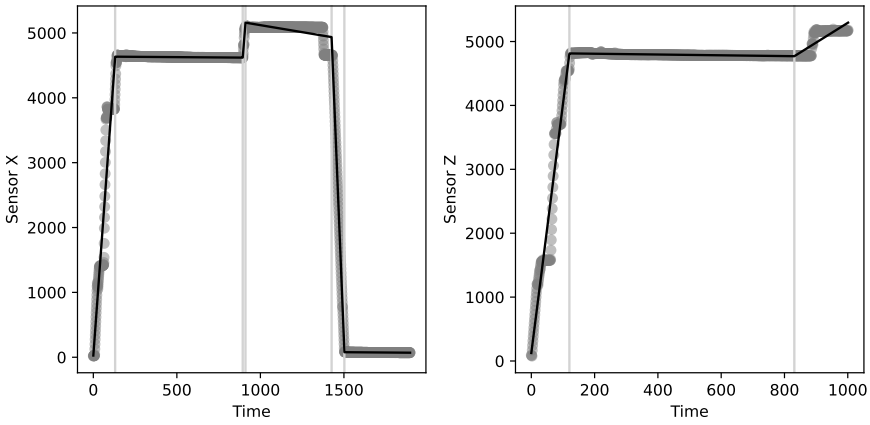


Fig. 2.8: Piecewise linear fits of sensor X and sensor Z data

Sensor X

	Range	slope	prediction
Segment 1	Time < 894.3	35.30	26.3–4634.1
Segment 2	130.5 ≤ Time < 894.3	-0.02	4634.1–4619.8
Segment 3	894.3 ≤ Time < 910.7	32.89	4619.8–5158.9
Segment 4	910.7 ≤ Time < 1426.3	-0.43	5158.9–4936.1
Segment 5	1426.3 ≤ Time < 1502.6	-63.67	4936.1–77.9
Segment 6	1502.6 ≤ Time	-0.02	77.9–70.0

Sensor Z

	Range	slope	prediction
Segment 1	Time < 831.1	38.97	125.2–4813.7
Segment 2	120.3 ≤ Time < 831.1	-0.06	4813.7–4770.3
Segment 3	831.1 ≤ Time	3.09	4770.3–5295.3

The slope of the longer second segment is almost zero for both sensor data. The segment analysis can therefore be used.

(b) The step function fits of Sect. 2.8.2 required 9 segments for sensor X and 6 segments for sensor Z to describe the stable phase sufficiently. With the piecewise linear fit, we require fewer segments.

(c) The piecewise linear fit can model a linear ramp-up phase like we see at the beginning of both run charts better than a step function fit. It also allows us identifying and estimating drifts in the run charts.

Chapter 3
Advanced Methods of Statistical Process Control

For the most recent version of the solution manual, go to `https://gedeck.github.io/mistat-code-solutions/IndustrialStatistics/`.

Import required modules and define required functions

```
import numpy as np
import pandas as pd
from scipy import stats
from scipy.special import factorial
import statsmodels.formula.api as smf
import mistat
import matplotlib.pyplot as plt
```

Exercise 3.1 Generate the distribution of the number of runs in a sample of size $n = 25$, if the number of elements above the sample mean is $m_2 = 10$:

(i) What are Q_1, Me, and Q_3 of this distribution?
(ii) Compute the expected value, μ_R, and the standard deviation σ_R.
(iii) What is $\Pr\{10 \leq R \leq 16\}$?
(iv) Determine the normal approximation to $\Pr\{10 \leq R \leq 16\}$.

Solution 3.1 First define a function to calculate the probability of runs:

```
from scipy.special import binom
def probRuns(m1, m2, R):
    n = m1 + m2
    k = R // 2
    if R % 2:
        denom = binom(m1-1,k-1) * binom(m2-1,k) + binom(m1-1,k) * binom(m2-1,k-1)
        return denom / binom(n, m2)
    else:
        return 2 * binom(m1-1, k-1) * binom(m2-1, k-1) / binom(n, m2)
```

For the case of $n = 25$ and $m_2 = 10$ calculate the distribution

```
n = 25
m2 = 10
m1 = n - m2
```

```
df = pd.DataFrame({
    'R': range(0, n+1),
    'p.d.f': [probRuns(m1, m2, R) for R in range(0, n+1)],
})
df['c.d.f'] = np.cumsum(df['p.d.f'])
```

(a) We can determine the median and quantiles from the calculated c.d.f

```
cdf = df['c.d.f'].values

Q1 = np.where(cdf < 0.25)[0][-1]
Me = np.where(cdf < 0.5)[0][-1]
Q3 = np.where(cdf < 0.75)[0][-1]
print(Q1, Me, Q3)
```

```
10 12 14
```

$Q_1 = 10$, $M_e = 12$ and $Q_3 = 14$.
(b) From Eq. (9.1.3) and Eq. (9.1.4) we get $\mu_R = 13$ and $\sigma_R = 2.3452$.

```
mu_R = 1 + 2 * m1 * m2 / n
sigma_R = np.sqrt(2*m1*m2*(2*m1*m2 - n) / (n*n*(n-1)))
print(mu_R, sigma_R)
```

```
13.0 2.345207879911715
```

(c) $\Pr\{10 \leq R \leq 16\} = 0.8657$.

```
p = df['c.d.f'][16] - df['c.d.f'][9]
print(p)
```

```
0.8656750572082381
```

(d) Using the normal approximation $\Pr\{10 \leq R \leq 16\} \approx 0.8644$. Note that we set the limits to [9.5, 16.5].

```
print(stats.norm.cdf((16.5-mu_R)/sigma_R) - stats.norm.cdf((9.5-mu_R)/sigma_R))
```

```
0.8644069987336978
```

Exercise 3.2 Use Python to perform a run test on the simulated cycle times from the pistons, which are in dataset **CYCLT.csv**. Is the number of runs above the mean cycle time significantly different than its expected value?

Solution 3.2 Load the data and determine number of runs information:

```
data = mistat.load_data('CYCLT')
# convert to up (1) or down (0) information relative to mean
mean_ct = np.mean(data)
runs = [1 if ct > mean_ct else 0 for ct in data]

# determine number of runs
obs_Runs = 0
current = None
for r in runs:
```

```
    if r != current:
        obs_Runs += 1
        current = r
print(f'Observed number of runs: {obs_Runs}')

# calculate expected number of runs
m1 = sum(data > mean_ct)
m2 = sum(data <= mean_ct)
n = m1 + m2
mu_R = 1 + 2 * m1 * m2 / n
print(f'Expected number of runs {mu_R:.2f}')

# determine if difference is significant
mistat.runsTest(data, cutoff=np.mean(data))
```

```
Observed number of runs: 26
Expected number of runs 25.64
```

```
Result(statistic=0.1044134517056721, pval=0.9168412481142088,
method='Runs Test', alternative='two.sided')
```

The test shows no significant deviations from randomness.

Exercise 3.3

(i) What is the expected number of runs up or down, in a sample of size 50?
(ii) Compute the number of runs up or down in the cycle time data (**CYCLT.csv**).
(iii) Is this number significantly different than expected?
(iv) What is the probability that a random sample of size 50 will have at least one run of size greater or equal to 5?

Solution 3.3 (i) For $n = 50$, $E\{R^*\} = \frac{2n-1}{3} = 33$.

```
n = 50
mu_Rstar = (2*n-1)/3
print(mu_Rstar)
```

```
33.0
```

(ii) Using the cycle time data, $R^* = 34$.

```
# determine direction of change up (1) or down (-1)
y = [1 if xi < xip1 else -1 for xi, xip1 in zip(data[:-1], data[1:])]

# count number of up and down runs
up = 0
down = 0
current = None
for yi in y:
    if yi == current: # no change of direction
        continue
    if yi < 0:
        down += 1
    else:
        up += 1
    current = yi
Rstar = up + down
print(Rstar, up, down)
```

```
34 17 17
```

(iii) We have $\sigma^* = 2.9269$, $\alpha_u^* = 1 - \Phi\left(\frac{1}{2.9269}\right) = 0.3663$. The deviation is not significant.

```
n = 50
mu_Rstar = (2*n-1)/3
sigma_Rstar = np.sqrt((16*n-29)/90)
print(sigma_Rstar)
alpha_L = stats.norm.cdf((Rstar-mu_Rstar)/sigma_Rstar)
alpha_U = 1 - alpha_L
print(alpha_U, alpha_L)
```

```
2.9268868558020253
0.3663034098961011 0.6336965901038989
```

(iv) The probability that a sample of size 50 will have at least one run of length 5 or longer is 0.102.

```
def expected_R_k(n, k):
    return 2 *(n*(k+1) - k*k - k + 1)/factorial(k+2)

print(expected_R_k(50, 5))
# probability to have run greater than 5
print(1 - np.exp(-expected_R_k(50, 5)))
```

```
0.10753968253968255
0.10195911479934461
```

The function *mistat.runStatistics* calculates a variety of statistics for runs.

```
mistat.runStatistics(data)
```

```
{'count': {'mu_R': 25.64, 'sigma_R': 3.4478316167038754, 'observed':
26},
 'direction': {'mu_Rstar': 33.0,
  'sigma_Rstar': 2.9268868558020253,
  'up': 17,
  'down': 17,
  'Rstar': 34,
  'alpha': [0.3663034098961011, 0.6336965901038989]}}
```

Exercise 3.4 Analyze the observations in **YARNSTRG.csv** for runs.

Solution 3.4 We make use of the *mistat.runStatistics* function.

```
data = mistat.load_data('YARNSTRG')
mistat.runStatistics(data)
```

```
{'count': {'mu_R': 50.92, 'sigma_R': 4.966642668235696, 'observed':
49},
 'direction': {'mu_Rstar': 66.33333333333333,
  'sigma_Rstar': 4.1779846284489315,
  'up': 32,
  'down': 32,
  'Rstar': 64,
  'alpha': [0.28825730949895256, 0.7117426905010474]}}
```

```
mistat.runsTest(data, cutoff=data.mean())
```

```
Result(statistic=-0.3865790491189181, pval=0.6990678707195341,
method='Runs Test', alternative='two.sided')
```

Exercise 3.5 Run the piston simulator at the upper level of the seven control parameters and generate 50 samples of size 5. Analyze the output for runs in both $\bar{X}$- and S-charts.

Solution 3.5 Define the parameter settings and create 50 simulations with 5 replications each

```
parameter = pd.DataFrame({
    'm': [60],
    's': [0.02],
    'k': [5_000],
    't': [296],
    'p0': [110_000],
    'v0': [0.01],
    't0': [360],
})
simulator = mistat.PistonSimulator(parameter=parameter, n_simulation=50, n_replicate=5, seed=1236)
# simulator = mistat.PistonSimulator(n_simulation=50, n_replicate=5, seed=1)
Ps = simulator.simulate()

# get grouped cycle times
cycleTime = mistat.qcc_groups(Ps['seconds'], Ps['group'])
mistat.runsTest(np.mean(cycleTime, axis=1), np.mean(cycleTime))
```

```
Result(statistic=-0.5610330087428953, pval=0.5747750350233831,
method='Runs Test', alternative='two.sided')
```

The test shows no significant runs. The resulting $\bar{X}$ and S-charts are shown in Fig. 3.1.

```
qcc = mistat.QualityControlChart(cycleTime, qcc_type='xbar')
ax = qcc.plot(title='for piston simulation at upper level')
plt.show()
qcc = mistat.QualityControlChart(cycleTime, qcc_type='S')
ax = qcc.plot(title='for piston simulation at upper level')
plt.show()
```

Exercise 3.6

(i) Run the piston simulator at the upper level of the seven control parameters and generate 50 samples of size 5 (both $\bar{X}$- and S-charts).
(ii) Repeat the exercise allowing T to change over time (provide a list of T which specify the changing ambient temperatur over time).
(iii) Compare the results in (i) and (ii) with those of Exercise 3.3.

Solution 3.6 (i) See solution of Exercise 3.5 for this part of the exercise. Fig. 3.1 shows the resulting control charts.

(ii) In simulating the piston cycle times, the ambient temperature around the piston is increased 10% per cycle after the shift point, which is after the 16^{th} sample.

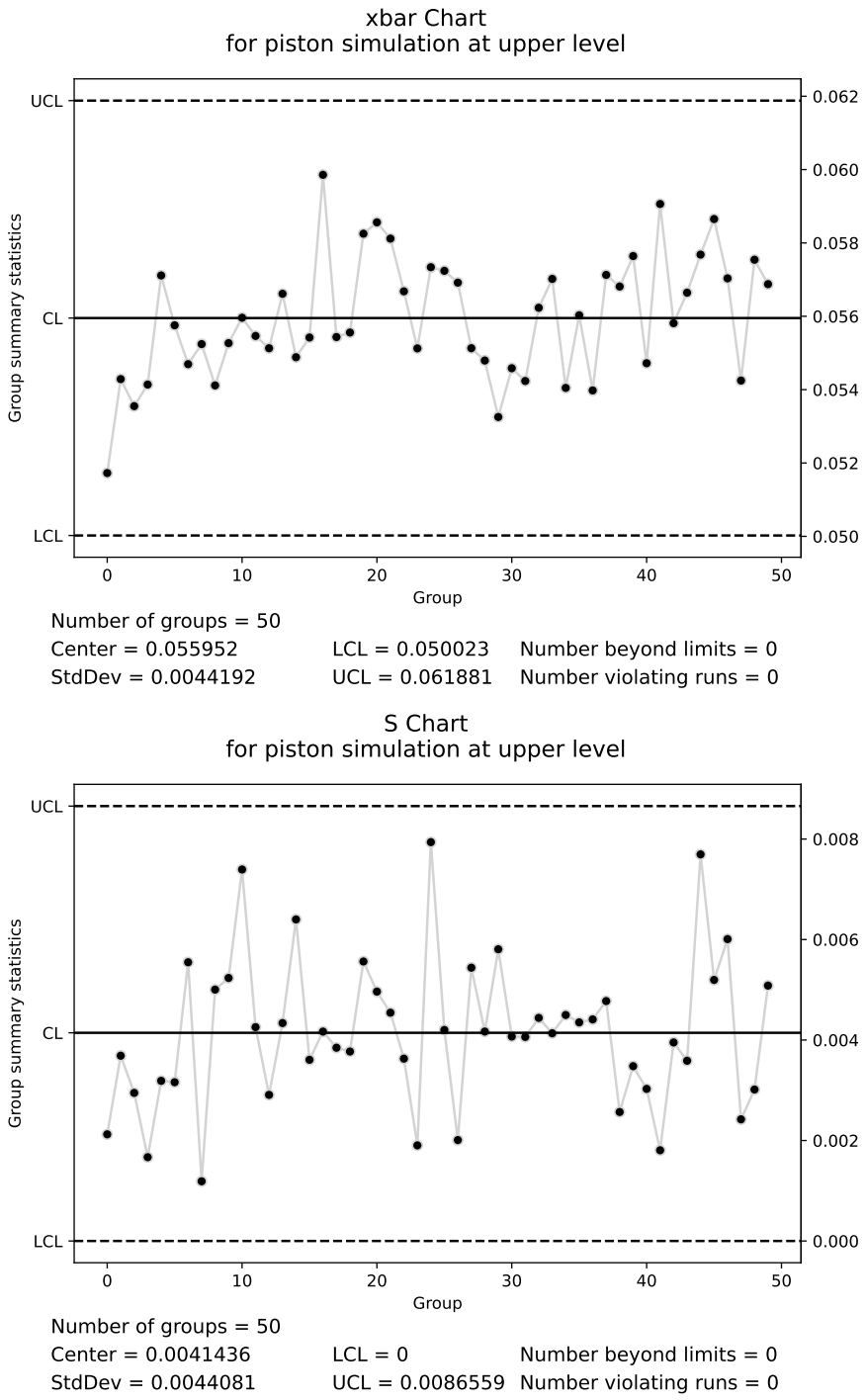


Fig. 3.1: $\bar{X}$ and S-chart of simulated Piston data (Exercises 3.5 and 3.6)

```
parameter = pd.DataFrame({
    'm': [60]*50,
    's': [0.02]*50,
    'k': [5_000]*50,
    't': [296] * 16 + [296 * 1.1**i for i in range(1, 35)],
    'p0': [110_000]*50,
    'v0': [0.01]*50,
    't0': [360]*50,
})

simulator = mistat.PistonSimulator(parameter=parameter, n_simulation=50,
                                   n_replicate=5, seed=1, check=False)
Ps = simulator.simulate()
cycleTimeTshift = mistat.qcc_groups(Ps['seconds'], Ps['group'])
```

```
qcc = mistat.QualityControlChart(cycleTimeTshift, qcc_type='xbar')
ax = qcc.plot(title='for contact lens data')
plt.show()
qcc = mistat.QualityControlChart(cycleTimeTshift, qcc_type='S')
ax = qcc.plot(title='for contact lens data')
plt.show()
```

The control charts for this simulation are shown in Fig. 3.2. Both show clearly unrandom behavior.

(iii) We calculate runs statistics for both simulations.

```
mistat.runStatistics(np.mean(cycleTime, axis=1))
```

```
{'count': {'mu_R': 25.96, 'sigma_R': 3.493555583105112, 'observed':
24},
 'direction': {'mu_Rstar': 33.0,
  'sigma_Rstar': 2.9268868558020253,
  'up': 16,
  'down': 16,
  'Rstar': 32,
  'alpha': [0.36630340989610105, 0.6336965901038989]}}
```

```
mistat.runStatistics(np.mean(cycleTimeTshift, axis=1))
```

```
{'count': {'mu_R': 25.96, 'sigma_R': 3.493555583105112, 'observed':
8},
 'direction': {'mu_Rstar': 33.0,
  'sigma_Rstar': 2.9268868558020253,
  'up': 15,
  'down': 14,
  'Rstar': 29,
  'alpha': [0.08586912144778469, 0.9141308785522153]}}
```

The number of runs in the first simulation is within the expected range. For the second simulation we observe a much smaller number of runs. The statistics for the direction of runs, shows no strong deviation from expected values.

```
mistat.runStatistics(np.mean(cycleTime, axis=1))
```

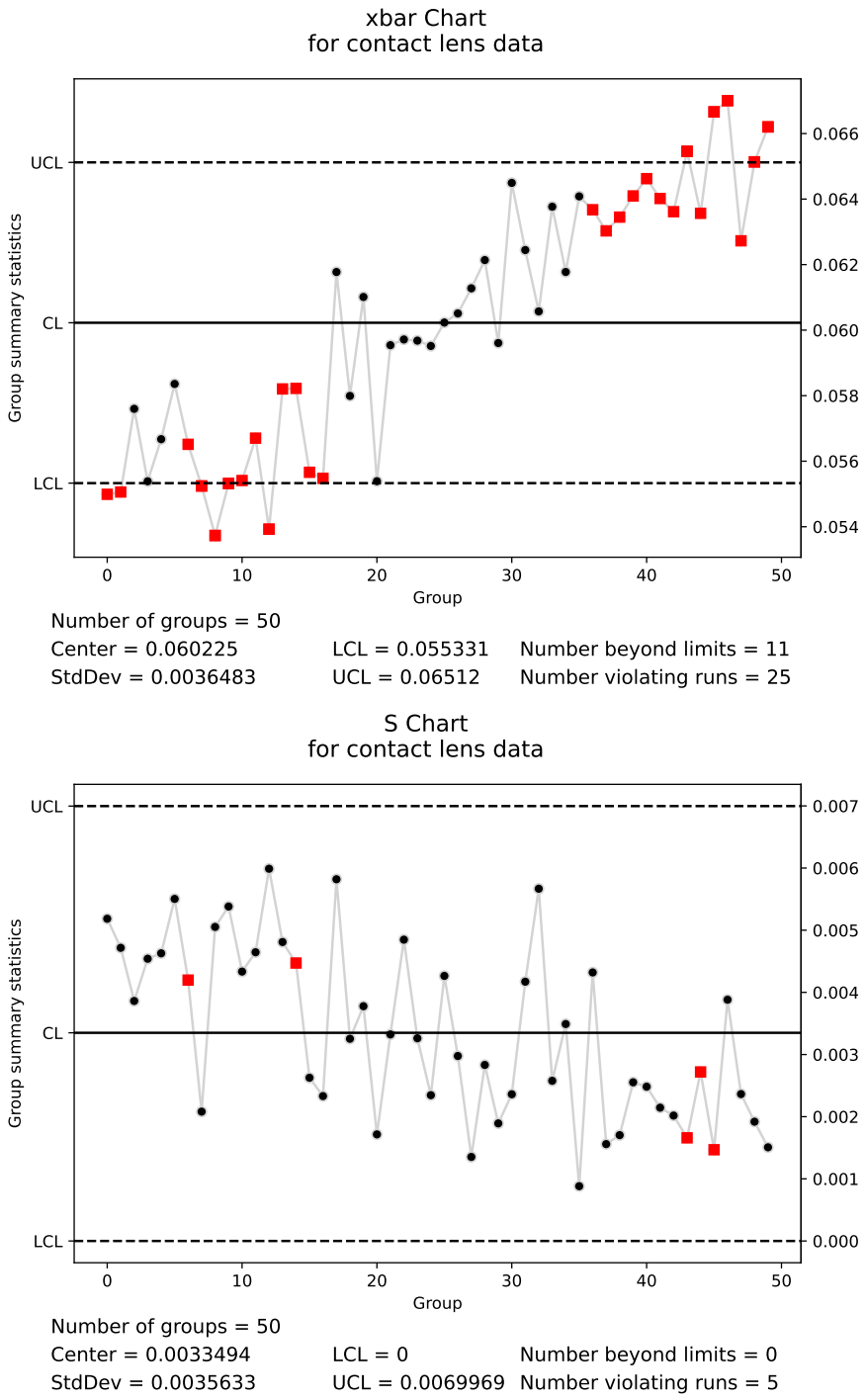


Fig. 3.2: $\bar{X}$ and S-chart of simulated Piston data (Exercise 3.6)

```
{'count': {'mu_R': 25.96, 'sigma_R': 3.493555583105112, 'observed':
24},
 'direction': {'mu_Rstar': 33.0,
  'sigma_Rstar': 2.9268868558020253,
  'up': 16,
  'down': 16,
  'Rstar': 32,
  'alpha': [0.36630340989610105, 0.6336965901038989]}}
```

```
mistat.runStatistics(np.mean(cycleTimeTshift, axis=1))
```

```
{'count': {'mu_R': 25.96, 'sigma_R': 3.493555583105112, 'observed':
8},
 'direction': {'mu_Rstar': 33.0,
  'sigma_Rstar': 2.9268868558020253,
  'up': 15,
  'down': 14,
  'Rstar': 29,
  'alpha': [0.08586912144778469, 0.9141308785522153]}}
```

We can also perform runs tests for both simulations. For the first simulation we get:

```
print('mean:', mistat.runsTest(np.mean(cycleTime, axis=1), np.mean(cycleTime)).pval)
STD = np.std(cycleTime, axis=1)
print('std:', mistat.runsTest(STD, np.mean(STD)).pval)
```

```
mean: 0.5747750350233831
std: 0.2529990614746843
```

This confirms our observation that the simulation shows no non-random behavior. For the second simulation we get:

```
print('mean:', mistat.runsTest(np.mean(cycleTimeTshift, axis=1), np.mean(cycleTimeTshift)).pval)
STD = np.std(cycleTimeTshift, axis=1)
print('std:', mistat.runsTest(STD, np.mean(STD)).pval)
```

```
mean: 2.7343384749349e-07
std: 0.09297787599834818
```

The run tests reflect the nonrandomness of the sequence, after the change-point.

Exercise 3.7 Construct a p-chart for the fraction of defective substrates received at a particular point in the production line. One thousand ($n = 1000$) substrates are sampled each week. Remove data for any week for which the process is not in control. Be sure to check for runs as well as points outside the control limits. Construct the revised p-chart and be sure to check for runs again. The data are in Table 3.1.

Solution 3.7 Construct the np-chart of the data using *mistat.QualityControlChart*. It is shown in Fig. 3.3.

```
data = pd.Series([18, 14,  9, 25, 27, 18, 21, 16, 18, 24, 20, 19, 22, 22, 20,
        38, 29, 35, 24, 20, 23, 17, 20, 19, 17, 16, 10,  8, 10,  9])

qcc = mistat.QualityControlChart(data, qcc_type='np', sizes=1000)
ax = qcc.plot(title='for defects')
plt.show()
```

Table 3.1: Dataset for Exercise 3.7

Week	No. Def.	Week	No. Def.
1	18	16	38
2	14	17	29
3	9	18	35
4	25	19	24
5	27	20	20
6	18	21	23
7	21	22	17
8	16	23	20
9	18	24	19
10	24	25	17
11	20	26	16
12	19	27	10
13	22	28	8
14	22	29	10
15	20	30	9

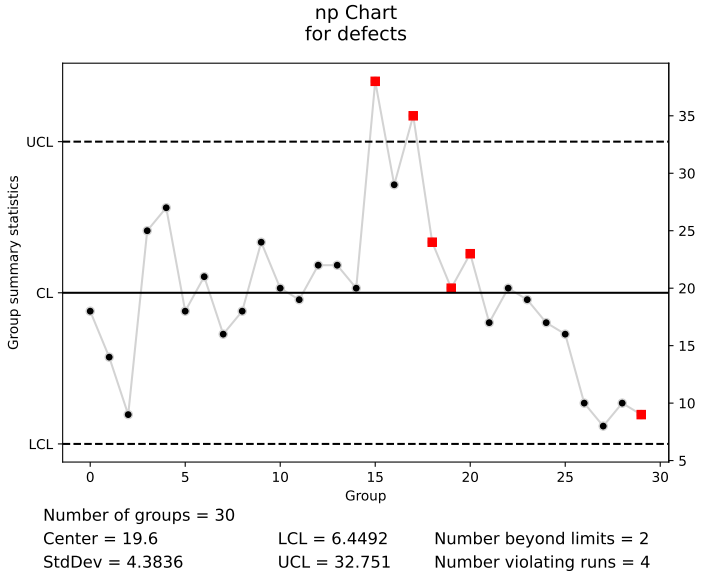


Fig. 3.3: np-Chart for defective substrates

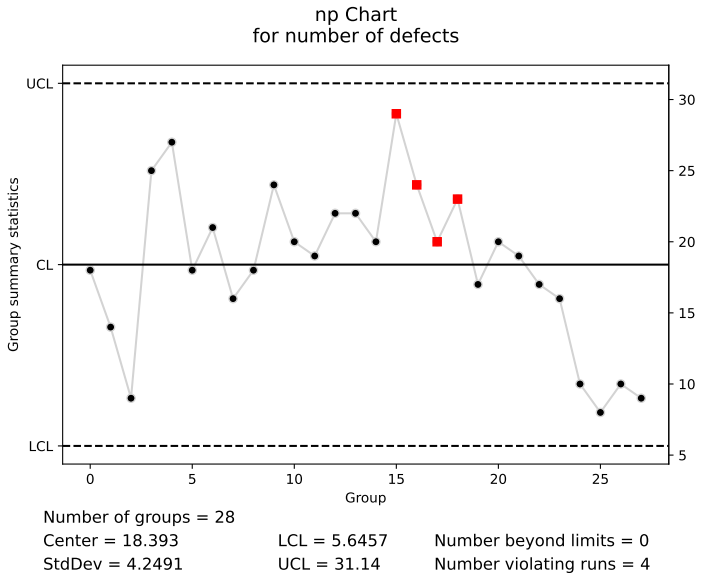


Fig. 3.4: Revised np-chart for defective substrates

As we see, the data points of week 16 and 18 fall outside the upper control limit. Also, the last 8 weeks show a significant trend down (improvement).

After removing the data points for week 16 and 18 (note that Python arrays are 0-indexed), we get a revised np-chart (Fig. 3.4). The points are now all in control, but the pattern of the last 8 weeks remains.

```
revised = data[data < 32.751]

qcc = mistat.QualityControlChart(revised, qcc_type='np', sizes=1000)
ax = qcc.plot(title='for number of defects')
plt.show()
```

Exercise 3.8 Substrates were inspected for defects on a weekly basis, on two different production lines. The weekly sample sizes and the number of defectives are indicated below in the dataset in Table 3.2. Plot the data and indicate which of the lines is not in a state of statistical control. On what basis do you make your decision? Use Python to construct control charts for the two production lines.

Note: When the sample size is not the same for each sampling period, we use **variable** control limits. If $X(i)$ and $n(i)$ represent the number of defects and sample size, respectively, for sampling period i, then the upper and lower control limits for the ith period are

$$UCL_i = \bar{p} + 3(\bar{p}(1-\bar{p})/n_i)^{1/2}$$

and

$$LCL_i = \bar{p} - 3(\bar{p}(1-\bar{p})/n_i)^{1/2}$$

where

$$\bar{p} = \sum X(i) / \sum n(i)$$

is the center line for the control chart.

Table 3.2: Dataset for Exercise 3.8

	Line 1		Line 2	
Week	X_i	n_i	X_i	n_i
1	45	7920	135	2640
2	72	6660	142	2160
3	25	6480	16	240
4	25	4500	5	120
5	33	5840	150	2760
6	35	7020	156	2640
7	42	6840	140	2760
8	35	8460	160	2980
9	50	7020	195	2880
10	55	9900	132	2160
11	26	9180	76	1560
12	22	7200	85	1680

Solution 3.8 The average proportion of defectives for line 1 is $\hat{p}_1 = 0.005343$, while that for line 2 is $\hat{p}_2 = 0.056631$. The difference is very significant. Indeed,

$$Z = \frac{\hat{p}_2 - \hat{p}_1}{\sqrt{\frac{\hat{p}_1(1-\hat{p}_1)}{N_1} + \frac{\hat{p}_2(1-\hat{p}_2)}{N_2}}} = 34.31,$$

where $N_1 = \sum_{i=1}^{12} n_1(i)$ and $N_2 = \sum_{i=1}^{12} n_2(i)$.

In Fig. 3.5, we present the p-charts for these two production lines. We see that the chart for line 1 reveals that the process was at the beginning (point 2), out of control.

```
data = pd.DataFrame([
    [1, 45, 7920, 135, 2640],
    [2, 72, 6660, 142, 2160],
    [3, 25, 6480,  16,  240],
    [4, 25, 4500,   5,  120],
    [5, 33, 5840, 150, 2760],
    [6, 35, 7020, 156, 2640],
    [7, 42, 6840, 140, 2760],
    [8, 35, 8460, 160, 2980],
    [9, 50, 7020, 195, 2880],
    [10,55, 9900, 132, 2160],
    [11,26, 9180,  76, 1560],
    [12,22, 7200,  85, 1680]],
    columns=['Week', 'Line 1 X', 'Line 1 n', 'Line 2 X', 'Line 2 n']
)

qcc = mistat.QualityControlChart(data['Line 1 X'], qcc_type='p', sizes=data['Line 1 n'])
ax = qcc.plot(title='for line 1 defects')
```

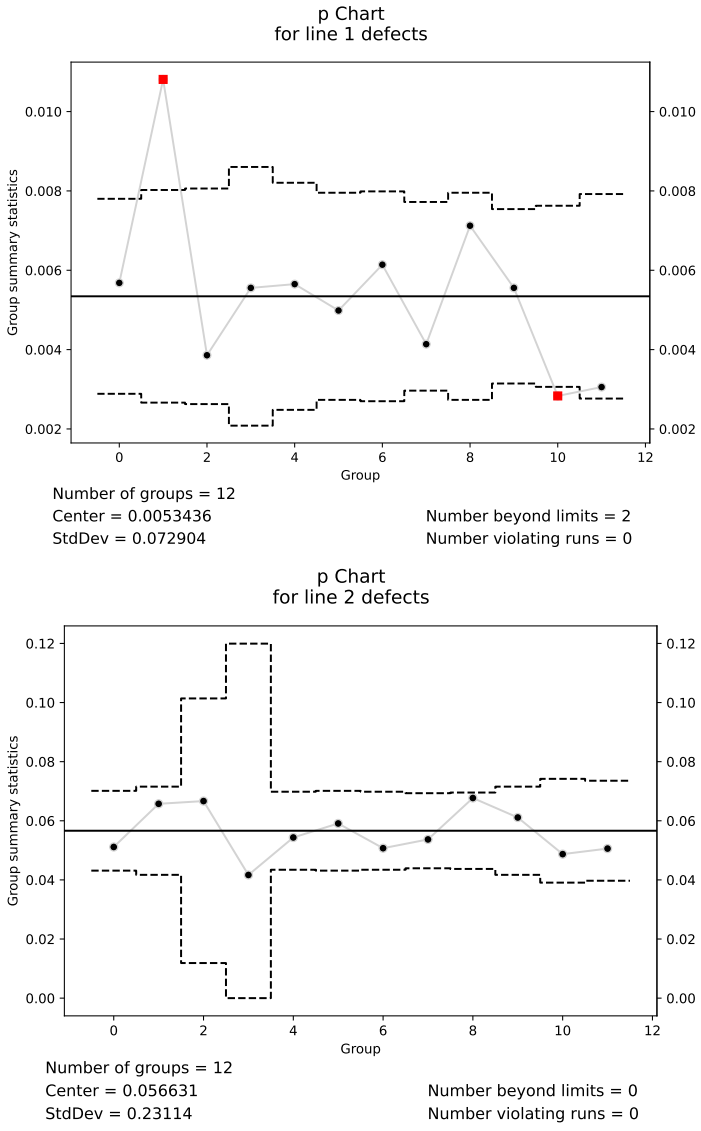


Fig. 3.5: *p*-charts for production line defects (Exercise 3.8)

```
plt.show()

qcc = mistat.QualityControlChart(data['Line 2 X'], qcc_type='p', sizes=data['Line 2 n'])
ax = qcc.plot(title='for line 2 defects')
plt.show()
```

Exercise 3.9 In designing a control chart for the fraction defectives p, a random sample of size n is drawn from the productions of each day (very large lot). How large should n be so that the probability of detecting a shift from $p_0 = 0.01$ to $p_t = 0.05$, within a 5-day period, will not be smaller than 0.8?

Solution 3.9 Let θ be the probability of not detecting the shift in a given day. Solving $\theta^5 = 0.2$ we get $\theta = 0.72478$. We find the smallest n for which

$$\theta \geq \Phi\left(\frac{0.01 + 3\sqrt{\frac{0.01\times 0.99}{n}} - 0.05}{\sqrt{\frac{0.05\times 0.95}{n}}}\right) - \Phi\left(\frac{0.01 - 3\sqrt{\frac{0.01\times 0.99}{n}} - 0.05}{\sqrt{\frac{0.05\times 0.95}{n}}}\right).$$

The solution is $n = 16$. Using Eq. (3.3.7) we get $n \approx 18$. This solution and the one shown above both use the normal approximation to the binomial. Another approach, which may be preferable for small sample sizes, is to use binomial distribution directly. We find the smallest n for which

$$B(n\times 0.01+3\sqrt{n\times 0.01\times 0.99}; n, 0.05) - B(n\times 0.01-3\sqrt{n\times 0.01\times 0.99}; n, 0.05) \leq \theta.$$

In this case the solution is $n = 7$.

Exercise 3.10 The data in Table 3.3 represent dock-to-stock cycle times for a certain type of shipment (class D). Incoming shipments are classified according to their "type," which is determined by the size of the item and the shipment, the type of handling required, and the destination of the shipment. Samples of five shipments per day are tracked from their initial arrival to their final destination, and the time it takes for this cycle to be complete is noted. The samples are selected as follows: at five preselected times during the day, the next class D shipment to arrive is tagged, and the arrival time and identity of the shipment are recorded. When the shipment reaches its final destination, the time is again recorded. The difference between these times is the cycle time. The cycle time is always recorded for the day of arrival:

(i) Construct $\bar{X}$ and S-charts from the data. Are any points out of control? Are there any trends in the data? If there are points beyond the control limits, assume that we can determine special causes for the points, and recalculate the control limits, excluding those points that are outside the control limits.
(ii) Use a t-test to decide whether the mean cycle time for days 21 and 22 was significantly greater than 45.
(iii) Make some conjectures about possible causes of unusually long cycle times. Can you think of other appropriate data that might have been collected, such as the times at which the shipments reached intermediate points in the cycle? Why would such data be useful?

Solution 3.10 (i) In Fig. 3.6, we see the $\bar{X}$ and S control charts. Points for days 21 and 22 are outside the UCL for $\bar{X}$. Excluding these points, the recalculated control chart limits for $\bar{X}$ are LCL = 29.731 and UCL = 45.781. Also, $\bar{\bar{X}} = 37.757$.

Table 3.3: Dock to Stock Cycle Times

Day	Times				
1	27	43	49	32	36
2	34	29	34	31	41
3	36	32	48	35	33
4	31	41	51	51	34
5	43	35	30	32	31
6	28	42	35	40	37
7	38	37	41	34	44
8	28	44	44	34	50
9	44	36	38	44	35
10	30	43	37	29	32
11	36	40	50	37	43
12	35	36	44	34	32
13	48	49	44	27	32
14	45	46	40	35	33
15	38	36	43	38	34
16	42	37	40	42	42
17	44	31	36	42	39
18	32	28	42	39	27
19	41	41	35	41	44
20	44	34	39	30	37
21	51	43	36	50	54
22	52	50	50	44	49
23	52	34	38	41	37
24	40	41	40	23	30
25	34	38	39	35	33

```
cycleTime = pd.DataFrame([
    [1, 27, 43, 49, 32, 36],      [2, 34, 29, 34, 31, 41],
    [3, 36, 32, 48, 35, 33],      [4, 31, 41, 51, 51, 34],
    [5, 43, 35, 30, 32, 31],      [6, 28, 42, 35, 40, 37],
    [7, 38, 37, 41, 34, 44],      [8, 28, 44, 44, 34, 50],
    [9, 44, 36, 38, 44, 35],      [10, 30, 43, 37, 29, 32],
    [11, 36, 40, 50, 37, 43],     [12, 35, 36, 44, 34, 32],
    [13, 48, 49, 44, 27, 32],     [14, 45, 46, 40, 35, 33],
    [15, 38, 36, 43, 38, 34],     [16, 42, 37, 40, 42, 42],
    [17, 44, 31, 36, 42, 39],     [18, 32, 28, 42, 39, 27],
    [19, 41, 41, 35, 41, 44],     [20, 44, 34, 39, 30, 37],
    [21, 51, 43, 36, 50, 54],     [22, 52, 50, 50, 44, 49],
    [23, 52, 34, 38, 41, 37],     [24, 40, 41, 40, 23, 30],
    [25, 34, 38, 39, 35, 33]],
    columns=['Week', 'S1', 'S2', 'S3', 'S4', 'S5'])
cycleTime = cycleTime.set_index('Week')

qcc = mistat.QualityControlChart(cycleTime, qcc_type='xbar')
ax = qcc.plot(title='for dock-to-stock cycle times')
plt.show()

qcc = mistat.QualityControlChart(cycleTime, qcc_type='S')
ax = qcc.plot(title='for dock-to-stock cycle times')
plt.show()

# exclude points for week 21 and 22
```

```
qcc = mistat.QualityControlChart(cycleTime.drop(labels=[21, 22]), qcc_type='xbar')
qcc.center, qcc.limits
```

```
(37.756521739130434,
          LCL        UCL
 0  29.731454  45.781589)
```

(ii) We can calculate the significance of the day 21 and 22 cycle times relative to the UCL of the recalculated control chart.

```
statistic, pvalue = stats.ttest_1samp(cycleTime.loc[21,], 45.781, alternative='greater')
print(pvalue)
statistic, pvalue = stats.ttest_1samp(cycleTime.loc[22,], 45.781, alternative='greater')
print(pvalue)
```

```
0.3846525285951686
0.03720636409199298
```

We find that only $\bar{X}_{22}$ is significantly larger than 45.781.

(iii) Unusual long cycle times can be due to

1. Missing or misplaced information in accompanying paperwork
2. Missing or misplaced marks on package
3. Defective package
4. Non standard package
5. Wrong information on package destination
6. Overloaded stock room so that packages cannot be accepted
7. Misplaced packages.

Additional data that can be collected to explain long cycle times:

1. Package destination
2. Stock room
3. Package size
4. Package weight
5. Package origin (country, supplier)
6. Package courier.

Exercise 3.11 Consider the modified Shewhart control chart for sample means, with $a = 3$, $w = 2$, and $r = 4$. What is the ARL of this procedure when $\delta = 0$, 1, 2, and the sample size is $n = 10$?

Solution 3.11 The ARL of the modified Shewhart control chart with $a = 3$, $w = 2$ and $r = 4$ when $n = 10$ and $\delta = 0$ is 370.3. When $\delta = 1$, the ARL is 1.75 and when $\delta = 2$ the ARL is 1.0.

Use the function *mistat.ARL_modifiedShewhartControlChart*:

```
print(mistat.ARL_modifiedShewhartControlChart(a=3, w=2, r=4, n=10, delta=0))
print(mistat.ARL_modifiedShewhartControlChart(a=3, w=2, r=4, n=10, delta=1))
print(mistat.ARL_modifiedShewhartControlChart(a=3, w=2, r=4, n=10, delta=2))
```

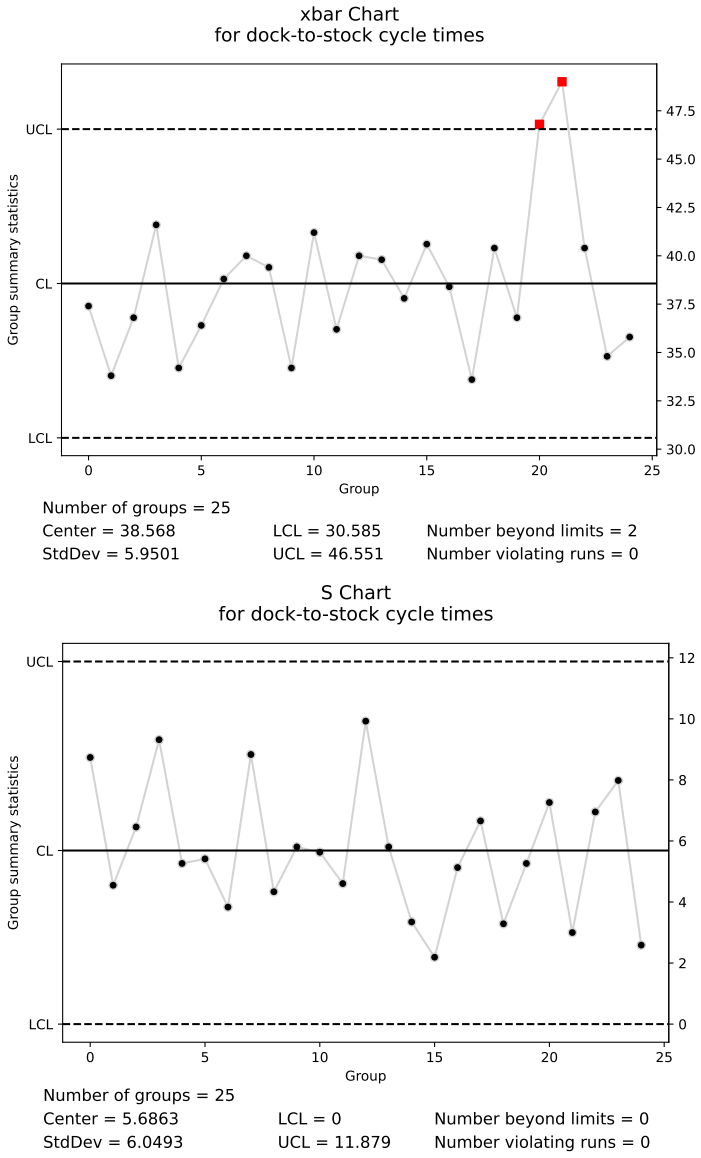


Fig. 3.6: Xbar and S-chart of dock to stock cycle times (Exercise 3.10)

```
370.3420378893707
1.7509678324381477
1.000442994467068
```

Exercise 3.12 Repeat the previous exercise for $a = 3$, $w = 1$, $r = 15$, when $n = 5$ and $\delta = 0.5$.

Solution 3.12 For $a = 3$, $w = 1$ and $r = 15$ when $n = 5$ and $\delta = 0.5$ the ARL is 33.4.

```
print(mistat.ARL_modifiedShewhartControlChart(a=3, w=1, r=15, delta=0.5, n=5))
```

```
33.3736035579561
```

Exercise 3.13 Write a Python application to simulate ARL and compare the results from the simulation to Exercises 3.11 and 3.12.

Solution 3.13 A possible solution is:

```
def ARLsimulation(a, w, r, delta, n, sigma=1, nrepeat=1_000):
    # standard deviation of means of n samples
    sigma_n = sigma / np.sqrt(n)
    # distribution of means from n samples; process shifted by delta*sigma
    distribution = stats.norm(scale=sigma_n, loc=delta*sigma)

    limit_a = a * sigma_n
    limit_w = w * sigma_n

    observed_runs = []
    for _ in range(nrepeat):
        measurements = distribution.rvs(10_000)
        run = 1
        warnings = -1
        for measurement in measurements:
            if abs(measurement) >= limit_a:
                break
            if abs(measurement) >= limit_w:
                if warnings == -1:
                    warnings = 0
                else:
                    warnings += 1
            else:
                if warnings != -1:
                    warnings = 0
            if warnings >= r:
                break
            run += 1
        observed_runs.append(run)
    return np.mean(observed_runs)
```

Using the function, we can repeat the previous exercises:

```
print(ARLsimulation(3, 2, 4, 0, 10),
      mistat.ARL_modifiedShewhartControlChart(a=3, w=2, r=4, delta=0, n=10))
print(ARLsimulation(3, 2, 4, 1, 10),
      mistat.ARL_modifiedShewhartControlChart(a=3, w=2, r=4, delta=1, n=10))
print(ARLsimulation(3, 2, 4, 2, 10),
      mistat.ARL_modifiedShewhartControlChart(a=3, w=2, r=4, delta=2, n=10))
```

```
370.582 370.3420378893707
1.761 1.7509678324381477
1.0 1.000442994467068
```

```
print(ARLsimulation(3, 1, 15, 0.5, 5),
      mistat.ARL_modifiedShewhartControlChart(a=3, w=1, r=15, delta=0.5, n=5))
```

```
36.106 33.3736035579561
```

The results of the simulation are close to the values from the previous exercise.

Exercise 3.14 Suppose that a shift in the mean is occurring at random, according to an exponential distribution with mean of 1 hour. The hourly cost is \$100 per shift of size $\delta = \frac{\mu_1 - \mu_0}{\sigma}$. The cost of sampling and testing is d = \$10 per item. How often should samples of size $n = 5$ be taken, when shifts of size $\delta \geq 1.5$ should be detected?

Solution 3.14 Samples should be taken every $h^0 = 34$ [min] = 0.57 [hr].

```
def h0(delta, d, c, lambda_, n):
    A = np.sqrt((1 + d*n) / (c*lambda_))
    B = np.sqrt(1 - stats.norm.cdf(3 - delta*np.sqrt(n)))
    return A * B

print(h0(1.5, 10, 100, 1, 5), h0(1.5, 10, 100, 1, 5) * 60)
```

```
0.5705857258216187 34.23514354929712
```

Exercise 3.15 Compute the $OC(p)$ function, for a Shewhart 3-sigma control chart for p, based on samples of size $n = 20$, when $p_0 = 0.10$. [Use the formula for exact computations.]

Solution 3.15 With $n = 20$ and $p_0 = 0.10$,

$$OC(p) = B(np_0 + 3\sqrt{np_0(1-p_0)}; n, p) - B(np_0 - 3\sqrt{np_0(1-p_0)}; n, p).$$

we get the following table of values of $OC(p)$, for $p = 0.05, 0.5\ (0.05)$.

p	$OC(p)$
0.05	0.999966
0.10	0.997614
0.15	0.978065
0.20	0.913307
0.25	0.785782
0.30	0.608010
0.35	0.416625
0.40	0.250011
0.45	0.129934
0.50	0.057659

Exercise 3.16 How large should the sample size n be, for a 3-sigma control chart for p, if we wish that the probability of detecting a shift from $p_0 = 0.01$ to $p_t = 0.05$ be $1 - \beta = 0.90$?

Solution 3.16 The sample size n_0 is the smallest n for which

$$B(n \times 0.01 + 0.2985\sqrt{n}; n, 0.05) - B(n \times 0.01 - 0.2985\sqrt{n}; n, 0.05) \leq 0.1.$$

From this we obtain $n_0 = 184$.

```
n = 1
while OC_p_chart(0.05, n, 0.01) > 0.1:
    n += 1
n
```

```
184
```

Using Eq. (3.3.5), which is based on the normal approximation to the binomial, gives $n_0 \approx 209$.

```
# normal approximation
def OC_p_chart_normal(p, n, p0):
    loc = n * p
    scale = np.sqrt(n*p*(1-p))
    c = n * p0
    delta = 3 * np.sqrt(n * p0 * (1 - p0))
    return (stats.norm(loc, scale).cdf(c + delta) -
            stats.norm(loc, scale).cdf(c - delta))

# alternative implementation
def OC_p_chart_normal_2(p, n, p0):
    delta = 3 * np.sqrt(p0 * (1 - p0) / n)
    UCL = p0 + delta
    LCL = p0 - delta
    denom = np.sqrt(p * (1 - p) / n)
    return (stats.norm().cdf((UCL - p)/denom) -
            stats.norm().cdf((LCL - p)/denom))

n = 1
while OC_p_chart_normal(0.05, n, 0.01) > 0.1:
    n += 1
print(n, OC_p_chart_normal(0.05, n, 0.01))
```

```
209 0.09959381959080052
```

Exercise 3.17 Suppose that a measurement X, of hardness of brackets after heat treatment, has a normal distribution. Every hour a sample of n units is drawn and a $\bar{X}$-chart with control limits $\mu_0 \pm 3\sigma/\sqrt{n}$ is used. Here, μ_0 and σ are the assumed process mean and standard deviation. The OC function is

$$OC(\delta) = \Phi(3 - \delta\sqrt{n}) + \Phi(3 + \delta\sqrt{n}) - 1$$

where $\delta = (\mu - \mu_0)/\sigma$ is the standardized deviation of the true process mean from the assumed one:

(i) How many hours, on the average, would it take to detect a shift in the process mean of size $\delta = 1$, when $n = 5$?
(ii) What should be the smallest sample size, n, so that a shift in the mean of size $\delta = 1$ would be on the average detected in less than 3 hours?
(iii) One has two options: to sample $n_1 = 5$ elements every hour or to sample $n_2 = 10$ elements every 2 hours. Which one would you choose? State your criterion for choosing between the two options and make the necessary computations.

Solution 3.17 **(i)** 4.5 [hr]; **(ii)** 7; **(iii)** For a shift of size $\delta = 1$, option 1 detects it on the average after 4.5 [hr]. Option 2 detects it on the average after $2 \times 1.77 = 3.5$ [hr]. Option 2 is preferred from the point of view of detection speed.

Exercise 3.18 Electric circuits are designed to have an output of 220 (volts, DC). If the mean output is above 222 (volts DC), you wish to detect such a shift as soon as possible. Examine the sample of dataset **OELECT.csv** for such a shift. For this purpose, construct a CUSUM upward scheme with K^+ and h^+ properly designed (consider for h^+ the value $\alpha = 0.001$). Each observation is of sample of size $n = 1$. Is there an indication of a shift in the mean?

Solution 3.18 In Python:

```
data = mistat.load_data('OELECT')
theta_0 = 220
theta_1 = 222
alpha = 0.001
sigma = data.std()
n = 1

K_p = 0.5 * (theta_0 + theta_1)
h_p = -sigma**2 * np.log(alpha) / (n * (theta_1 - theta_0))
print(f'K+: {K_p:.0f}; h+: {h_p:.3f}')
```

```
K+: 221; h+: 55.372
```

Here $K^+ = 221$ and $h^+ = 55.3724$. Fig. 3.7 shows the "up" and "down" CUSUM charts. We see that the upper or lower limits are not crossed. There is no signal of change.

```
data = mistat.load_data('OELECT')
analysis = mistat.Cusum(data, center=K_p, decision_interval=h_p)
analysis.plot()
plt.show()
```

Exercise 3.19 Estimate the probability of false alarm and the conditional expected delay in the Poisson case, with a CUSUM scheme. The parameters are $\lambda_0 = 15$, $\lambda_1^+ = 25$ and $\lambda_1^- = 7$. Use $\alpha = 0.001$, $\tau = 30$.

Solution 3.19 Using $K^+ = 19.576$, $h^+ = 13.523$, $K^- = 10.497$ and $h^- = -9.064$ we obtained $PFA = 0.014$ and $CED = 3.01 \pm 0.235$. Here the value of λ changed from 15 to 25.

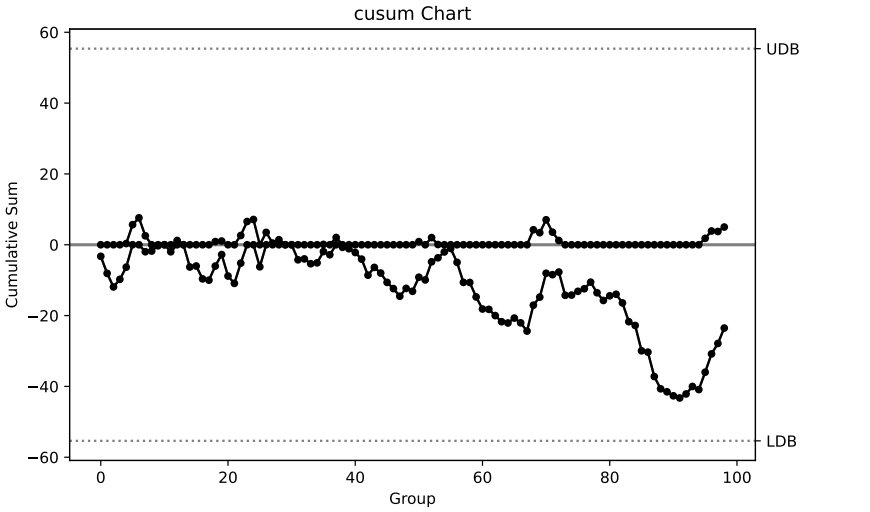


Fig. 3.7: CUSUM chart for dataset **OELECT.csv**

```
lambda0 = 15
lambda1_p = 25
lambda1_m = 7
alpha = 0.001
tau = 30

kp = (lambda1_p - lambda0) / np.log(lambda1_p/lambda0)
hp = - np.log(alpha) / np.log(lambda1_p/lambda0)
km = (lambda1_m - lambda0) / np.log(lambda1_m/lambda0)
hm = - np.log(alpha) / np.log(lambda1_m/lambda0)

print(f'Kp={kp:.3f}, hp={hp:.3f}')
print(f'Km={km:.3f}, hm={hm:.3f}')

arl = mistat.cusumPfaCed(randFunc1=stats.poisson(mu=15),
                         randFunc2=stats.poisson(mu=25),
                         tau=tau,
                         kp=kp, km=km,
                         hp=hp, hm=hm,
                         N=4000, limit=1000, seed=1)
result = arl['statistic']
```

```
Kp=19.576, hp=13.523
Km=10.497, hm=-9.064
PFA 0.01075  CED 2.1663  Std. Error 0.51072
```

Exercise 3.20 A CUSUM control scheme is based on sample means:

(i) Determine the control parameters K^+, h^+, K^-, h^-, when $\mu_0 = 100$, $\mu_1^+ = 110$, $\mu_1^- = 90$, $\sigma = 20$, $n = 5$, $\alpha = 0.001$.

(ii) Estimate the PFA and CED, when the change-point is at $\tau = 10, 20, 30$.
(iii) How would the properties of the CUSUM change if each sample size is increased from 5 to 20.

Solution 3.20 **(i)** Determine the control parameters:

```
mu0 = 100
mu1p = 110
mu1m = 90
sigma = 20
n = 5
alpha = 0.001

Kp = (mu0+mu1p)/2
hp = -(sigma**2 / n) * np.log(alpha) / (mu1p-mu0)
Km = (mu0+mu1m)/2
hm = -(sigma**2 / n) * np.log(alpha) / (mu1m-mu0)

pd.Series({'Kp': Kp, 'hp': hp, 'Km': Km, 'hm': hm})
```

```
Kp    105.000000
hp     55.262042
Km     95.000000
hm    -55.262042
dtype: float64
```

(ii) Estimate PFA and CED for $\tau = 10, 20, 30$.

```
results = []
for tau in [10, 20, 30]:
    arl = mistat.cusumPfaCed(randFunc1=stats.norm(loc=mu0, scale=sigma/np.sqrt(5)),
                             randFunc2=stats.norm(loc=mu1p, scale=sigma/np.sqrt(5)),
                             tau=tau, kp=Kp, km=Km, hp=hp, hm=hm,
                             N=300, limit=1000, seed=1, verbose=False)
    results.append({
        'tau': tau,
        **arl['statistic'], # copy all results from arl['statistic']
    })
pd.DataFrame(results)
```

	tau	PFA	CED	Std. Error
0	10	0.000000	10.090000	1.050516
1	20	0.003333	10.076923	1.671984
2	30	0.006667	10.449664	2.289643

(iii) Increase n to 20 and estimate PFA and CED for $\tau = 10, 20, 30$.

```
n = 20
hp = -(sigma**2 / n) * np.log(alpha) / (mu1p-mu0)
hm = -(sigma**2 / n) * np.log(alpha) / (mu1m-mu0)

results = []
for tau in [10, 20, 30]:
    arl = mistat.cusumPfaCed(randFunc1=stats.norm(loc=mu0, scale=sigma/np.sqrt(5)),
                             randFunc2=stats.norm(loc=mu1p, scale=sigma/np.sqrt(5)),
                             tau=tau, kp=Kp, km=Km, hp=hp, hm=hm,
                             N=300, limit=1000, seed=1, verbose=False)
    results.append({
        'tau': tau,
        **arl['statistic'], # copy all results from arl['statistic']
    })
pd.DataFrame(results)
```

```
   tau       PFA       CED  Std. Error
0   10  0.586667  1.709677    1.056345
1   20  0.806667  2.172414    2.910490
2   30  0.920000  2.250000    6.583663
```

The probability of a false alarm increases considerably.

Exercise 3.21 Show that the Shiryaev-Roberts statistic W_n, for detecting a shift in a Poisson distribution from a mean λ_0 to a mean $\lambda_1 = \lambda_0 + \delta$, is

$$W_m = (1 + W_{m-1})R_m$$

where $W_0 \equiv 0$, $R_m = \exp\{-\delta + x_m \log(\rho)\}$, and $\rho = \lambda_1/\lambda_0$.

Solution 3.21 According to Eq. (3.5.6), in the Poisson case,

$$\begin{aligned}
W_m &= \sum_{i=1}^{m-1} \prod_{j=i+1}^{m} R_j \\
&= \sum_{i=1}^{m-1} \exp\left\{-(m-i)\delta + \sum_{j=i+1}^{m} X_j \log(\rho)\right\} \\
&= e^{-\delta + X_m \log\rho} + \sum_{i=1}^{m-2} \exp\left\{-(m-i)\delta + \sum_{j=i+1}^{m} X_j \log(\rho)\right\} \\
&= e^{-\delta + X_m \log\rho} + e^{-\delta + X_m \log\rho} \sum_{i=1}^{m-2} \exp\left\{-(m-1-i)\delta + \sum_{j=i+1}^{m-1} X_j \log(\rho)\right\} \\
&= (1 + W_{m-1}) e^{-\delta + X_m \log\rho} \\
&= (1 + W_{m-1}) R_m .
\end{aligned}$$

Exercise 3.22 Analyze the data in data **OELECT**, with an EWMA control chart with $\lambda = 0.2$.

Solution 3.22 The mean of the data is $\bar{X} = 219.25$, and its standard deviation is $S = 4.004$.

```
data = mistat.load_data('OELECT')
ewma = mistat.EWMA(data, center=data.mean(), std_dev=data.std(),
                   sizes=1, smooth=0.2, nsigmas=3)
ewma.plot()
plt.show()
```

Fig. 3.8 shows that there is no significant shift in the data.

Exercise 3.23 Analyze the variable diameters in the dataset **ALMPIN** with an EWMA control chart with $\lambda = 0.2$. Explain how you would apply the automatic process control technique described at the end of Sect. 3.7.

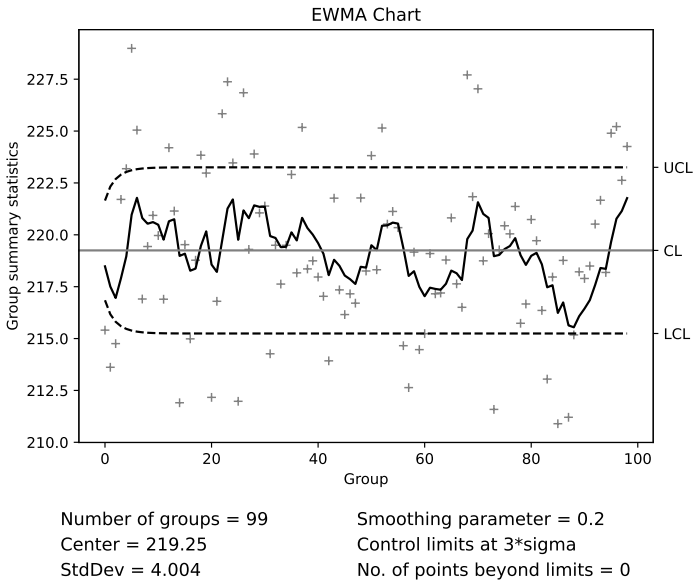


Fig. 3.8: EWMA chart for the **OELECT** dataset

Solution 3.23 In Fig. 3.9, we see the EWMA control chart for Diameter 1. The mean and standard deviation are $\bar{X} = 9.993$ and $S = 0.0164$. The target value for Diameter 1 is 10 mm. Notice in the chart that after a drop below 9.98 mm the machine corrects itself automatically, and there is a significant run upwards towards the target value. Using the EWMA chart an automatic control can be based on Eq. **9.8.11**.

```
data = mistat.load_data('ALMPIN')
data = data['diam1']
ewma = mistat.EWMA(data, center=data.mean(), std_dev=data.std(),
                   sizes=1, smooth=0.2, nsigmas=3)
ax = ewma.plot()
ax.set_ylim(9.96, 10.02)
plt.show()
```

Exercise 3.24 Construct the Kalman filter for the Dow-Jones daily index, which is given in the dataset **DOW1941**.

Solution 3.24 Using the first 50 values of **DOW1941** perform the regression method outlined in the text,

```
dow1941 = mistat.load_data('DOW1941')
# solve the regression equation
m = 50
sqrt_t = np.sqrt(range(1, m + 1))
df = pd.DataFrame({
    'Ut': dow1941[:m]/sqrt_t,
```

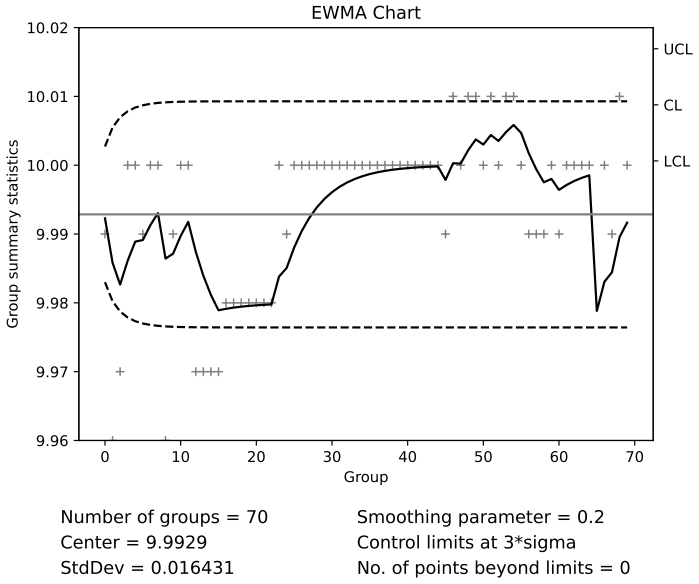


Fig. 3.9: EWMA chart for the Diameter 1

```
    'x1t': 1 / sqrt_t,
    'x2t': sqrt_t,
})
model = smf.ols(formula='Ut ~ x1t + x2t - 1', data=df).fit()
mu0, delta = model.params
var_eta = np.var(model.resid, ddof=2)
pd.Series({'mu0': mu0, 'delta': delta, 'Var(eta)': var_eta})
```

```
mu0         132.808555
delta        -0.255630
Var(eta)      0.297616
dtype: float64
```

The least squares estimates of the initial parameter values are $\hat{\mu}_0 = 132.809$, $\hat{\delta} = -0.2556$ and $\hat{\sigma}_\epsilon^2 + \hat{\sigma}_2^2 = 0.297616$.

For the Kalman filter, we choose $\hat{\sigma}_\epsilon^2 = 0.15$ and for w_0^2 the value 0.0015. The Python commands used to obtain the Kalman filter estimates of the **DOW1941** data are as follows:

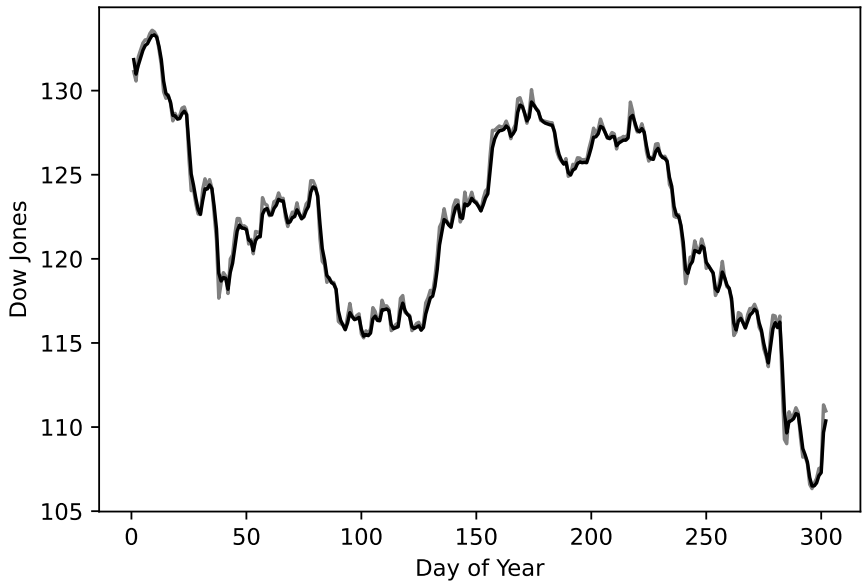
```
# choose sig2e and w20
sig2e = 0.15
w20 = 0.0015
# apply the filter
results = []
mu_tm1 = mu0
w2_tm1 = w20
y_tm1 = mu0
for i in range(0, len(dow1941)):
    y_t = dow1941[i]
    B_t = sig2e / (var_eta + w2_tm1)
```

```
    mu_t = B_t * (mu_tm1 + delta) + (1 - B_t) * y_t # X
    results.append({
        't': i + 1,
        'y_t': y_t,
        'mu_t': mu_t,
        'B_t': B_t,
        'W2_t': w2_tm1,
    })
    w2_tm1 = B_t * (var_eta - sig2e + w2_tm1)
    mu_tm1 = mu_t
    y_tm1 = y_t
results = pd.DataFrame(results)
```

The Table 3.4 shows the first 25 values of the DOW 1941 data as well as the values of B_t, w_t^2, and $\hat{\mu}_t$. Starting with $w_0^2 = 0.0015$ we see that w_t^2 converges fast to 0.0923. The values of $\hat{\mu}_t$ are very close to the data values. The result is shown graphically in Table 3.4 with actual values of Y_t in grey and estimated $\hat{\mu}_t$ in black.

Table 3.4: Results for Kalman filter applied to the Dow-Jones daily 1941 dataset

	Dow 1941	$\hat{\mu}_t$	B_t	w_t^2
0	131.1300	131.8436	0.5015	0.0015
1	130.5700	130.9800	0.4028	0.0748
2	132.0100	131.5120	0.3874	0.0896
3	132.4000	131.9596	0.3851	0.0919
4	132.8300	132.3967	0.3848	0.0922
5	133.0200	132.6819	0.3847	0.0923
6	133.0200	132.7916	0.3847	0.0923
7	133.3900	133.0614	0.3847	0.0923
8	133.5900	133.2883	0.3847	0.0923
9	133.4900	133.3141	0.3847	0.0923
10	133.2500	133.1763	0.3847	0.0923
11	132.4400	132.6249	0.3847	0.0923
12	131.5100	131.8406	0.3847	0.0923
13	129.9300	130.5667	0.3847	0.0923
14	129.5400	129.8366	0.3847	0.0923
15	129.7500	129.6850	0.3847	0.0923
16	129.2400	129.3128	0.3847	0.0923
17	128.2000	128.5298	0.3847	0.0923
18	128.6500	128.5054	0.3847	0.0923
19	128.3400	128.3053	0.3847	0.0923
20	128.5200	128.3391	0.3847	0.0923
21	128.9600	128.6228	0.3847	0.0923
22	129.0300	128.7750	0.3847	0.0923
23	128.6000	128.5690	0.3847	0.0923
24	126.0000	126.8900	0.3847	0.0923



Chapter 4
Multivariate Statistical Process Control

For the most recent version of the solution manual, go to `https://gedeck.github.io/mistat-code-solutions/IndustrialStatistics/`.

Import required modules and define required functions

```
import numpy as np
import pandas as pd
from scipy import stats
import matplotlib.pyplot as plt
import seaborn as sns

import mistat
```

Exercise 4.1 In dataset **TSQ** we find 368 T^2 values corresponding to the vectors (x, y, θ) in the **PLACE** dataset. The first $n = 48$ vectors in **PLACE** dataset were used as a base sample, to compute the vector of means **m** and the covariance matrix S. The T^2 values are for the other individual vectors ($m = 1$). Plot the T^2 values in the dataset **TSQ.csv**. Compute the UCL and describe from the plot what might have happened in the placement process generating the (x, y, θ) values.

Solution 4.1 The values of T^2 were computed, with **m** being the mean of the first 48 vectors. Fig. **??** shows the T^2 values for all boards with the calculated control limit of UCL = 17.1953.

```
# use eqn 4.1.6 to calculate the upper control limit for monitoring
n = 48; p = 3
UCL = (n-1)*(n+1)*p/(n*(n-p)) * stats.f(p, n-p).ppf(0.997)

tsq = mistat.load_data('TSQ')
ax = tsq.plot()
ax.axhline(UCL, color='black')
ax.axhline(0, color='black')
```

```
<matplotlib.lines.Line2D at 0x7f343dbe1700>
```

The side by side boxplots in Fig. 4.2 aggregate the information of the T^2 values by board number.

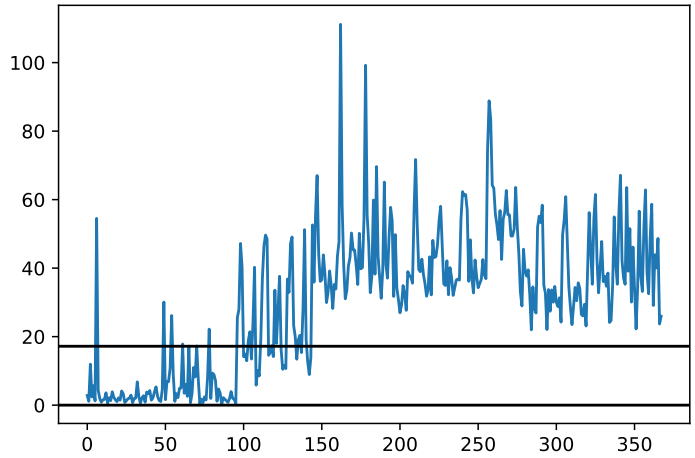


Fig. 4.1: Plot of T^2 values with control limit

```
df = pd.DataFrame({
    'T2': tsq,
    'board': mistat.load_data('PLACE')['crcBrd'][48:],
})
ax = df.groupby('board').boxplot(column='T2', subplots=False, rot=90, grid=False)
ax.axhline(UCL, color='grey')
plt.show()
```

We see that even on the first 9 cards there are a few outliers, that is, points whose T^2 is outside the control limits. All points from card 13 on and a majority of points from boards 10, 11 and 12 have T^2 values greater than UCL.

Exercise 4.2 Prove that if $\mathbf{X}$ has a multivariate normal distribution, $N_v(\boldsymbol{\mu}, \boldsymbol{\sigma})$, then $(\mathbf{X} - \boldsymbol{\mu})'\boldsymbol{\Sigma}^{-1}(\mathbf{X} - \boldsymbol{\mu})$ has a χ^2 distribution with v degrees of freedom where $R = \chi^2_{1-p}[v]$ is the corresponding $(1-p)$ quantile of the χ^2 distribution with v degrees of freedom.

Solution 4.2 As $N_v(\boldsymbol{\mu}, \boldsymbol{\sigma})$ is a multivariate normal distribution, $\boldsymbol{\Sigma}$ is positive definite. Therefore there exists a nonsingular matrix $\boldsymbol{P} = \boldsymbol{\Sigma}^{\frac{1}{2}} =$ so that $\boldsymbol{\Sigma} = \boldsymbol{P}\boldsymbol{P}'$.

Using this we get,

$$\begin{aligned}
(\mathbf{X} - \boldsymbol{\mu})'\boldsymbol{\Sigma}^{-1}(\mathbf{X} - \boldsymbol{\mu}) &= (\mathbf{X} - \boldsymbol{\mu})'(\boldsymbol{P}')^{-1}\boldsymbol{P}^{-1}(\mathbf{X} - \boldsymbol{\mu}) \\
&= (\mathbf{X} - \boldsymbol{\mu})'(\boldsymbol{P}^{-1})'\boldsymbol{P}^{-1}(\mathbf{X} - \boldsymbol{\mu}) \\
&= \left(\boldsymbol{P}^{-1}(\mathbf{X} - \boldsymbol{\mu})\right)'\boldsymbol{P}^{-1}(\mathbf{X} - \boldsymbol{\mu}) \\
&= \boldsymbol{Y}'\boldsymbol{Y}
\end{aligned}$$

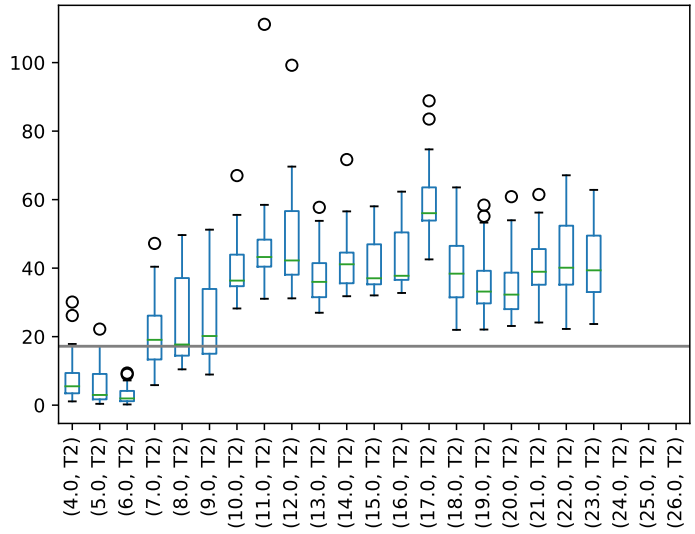


Fig. 4.2: Plot of T^2 values with control limit

with $\boldsymbol{Y} = \boldsymbol{P}^{-1}(\mathbf{X} - \boldsymbol{\mu}) = \boldsymbol{\Sigma}^{-\frac{1}{2}}(\mathbf{X} - \boldsymbol{\mu})$. $\boldsymbol{Y}$ has a multivariate normal distribution $N_v(\mathbf{0}, \boldsymbol{I})$. The product $(\mathbf{X} - \boldsymbol{\mu})'\boldsymbol{\Sigma}^{-1}(\mathbf{X} - \boldsymbol{\mu}) = \boldsymbol{Y}'\boldsymbol{Y}$ is therefore a χ^2 distribution with v degrees of freedom.

Exercise 4.3 Sort the dataset **CAR** by variable `cyl`, indicating the number of cylinders in a car, and run a T^2 chart with internally derived targets for the variables `turn`, `hp`, `mpg`, with separate computations for cars with 4, 6 and 8 cylinders. How is the number of cylinders affecting the overall performance of the cars?

Solution 4.3 Create T^2 chart using separate computations by cylinders. The chart is shown in Fig. 4.3.

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

fig, axes = plt.subplots(nrows=3, figsize=[8, 8])
for cyl, ax in zip([4, 6, 8], axes):
  base = car.loc[car['cyl'] == cyl, columns]
  newdata = car.loc[car['cyl'] != cyl, columns]
  mqcc = mistat.MultivariateQualityControlChart(base, qcc_type='T2single',
            confidence_level=0.99, newdata=newdata)
  mqcc.plot(ax=ax, show_legend=False)
  ax.set_ylabel(f'{cyl} cylinders')
  plt.tight_layout()
plt.show()
```

The T^2 charts for the internally derived targets for 4 or 6 cylinders, show now relevant differences towards the remaining data. For 8 cylinders on the other hand, many of other cars show strong differences.

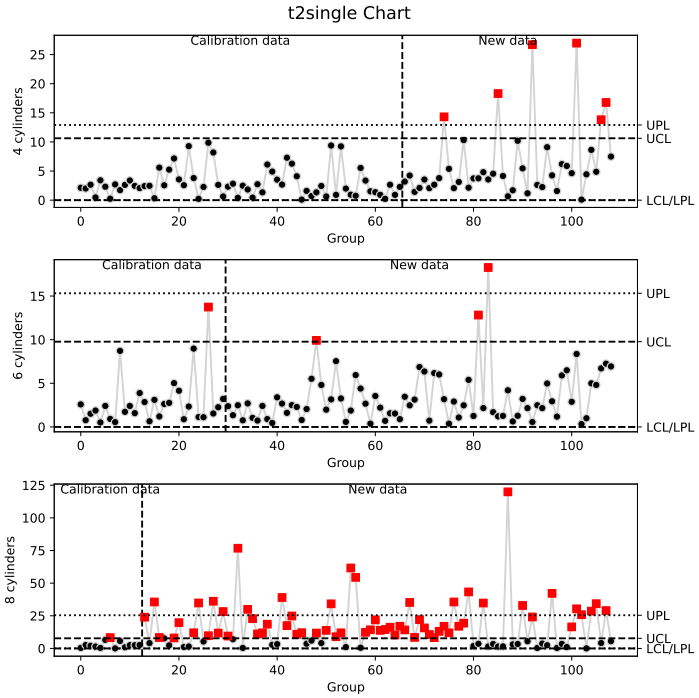


Fig. 4.3: T^2 charts for the **CAR** dataset using subsets based on number of cylinders to derive internal targets (Excercise 4.3)

Exercise 4.4 Sort the dataset **CAR.csv** by variable `origin`, indicating the country of origin, and run a T^2 chart with internally derived targets for the variables `turn`, `hp`, `mpg`, with separate computations for cars from 1 = US; 2 = Europe; 3 = Asia. How is the country of origin affecting the overall performance of the cars?

Solution 4.4 Create T^2 chart using separate computations based on origin. The chart is shown in Fig. 4.4.

```
car = mistat.load_data('CAR')
car = car.sort_values('cyl')
columns = ['turn', 'hp', 'mpg']
origins = [None, 'US', 'Europe', 'Asia']

fig, axes = plt.subplots(nrows=3, figsize=[8, 8])
for origin, ax in zip([1, 2, 3], axes):
  base = car.loc[car['origin'] == origin, columns]
  newdata = car.loc[car['origin'] != origin, columns]
  mqcc = mistat.MultivariateQualityControlChart(base, qcc_type='T2single',
            confidence_level=0.99, newdata=newdata)
  mqcc.plot(ax=ax, show_legend=False)
  ax.set_ylabel(f'Origin {origins[origin]}')
plt.tight_layout()
plt.show()
```

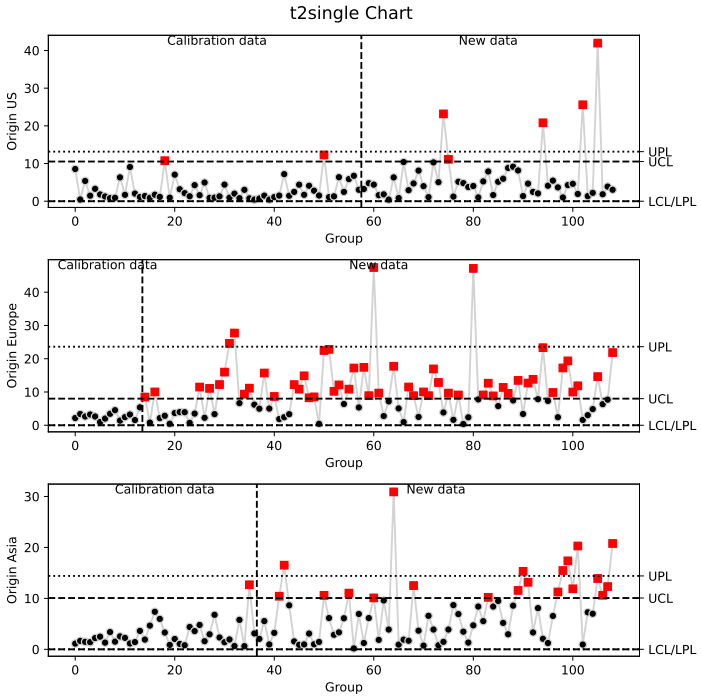


Fig. 4.4: T^2 charts for the **CAR** dataset using subsets based on number of cylinders to derive internal targets (Excercise 4.4)

Using the European cars to derive internal targets, we can see that many of the other cars are above the UCL. This is less frequent in the other two plots.

The differences are also obvious when the data are visualized in a scatterplot matrix. See Fig. 4.5

```
sns.pairplot(car[[*columns, 'cyl']], hue='cyl', height=1.5)
sns.pairplot(car[[*columns, 'origin']], hue='origin', height=1.5)
plt.show()
```

Exercise 4.5 Load the dataset **GASOL.csv** and compute a T^2 chart for $x1$, $x2$, `astm`, `endPt`, `yield`. Design the chart with an external assigned target based on observations 12–24. Compare the charts. Explain the differences.

Solution 4.5 Create T^2 chart for full **GASOL** dataset.

```
gasol = mistat.load_data('GASOL')
mqcc = mistat.MultivariateQualityControlChart(gasol, qcc_type='T2single',
           confidence_level=0.99)
ax = mqcc.plot()
ax.plot((11, 23), (1, 1), color='red')
plt.show()
```

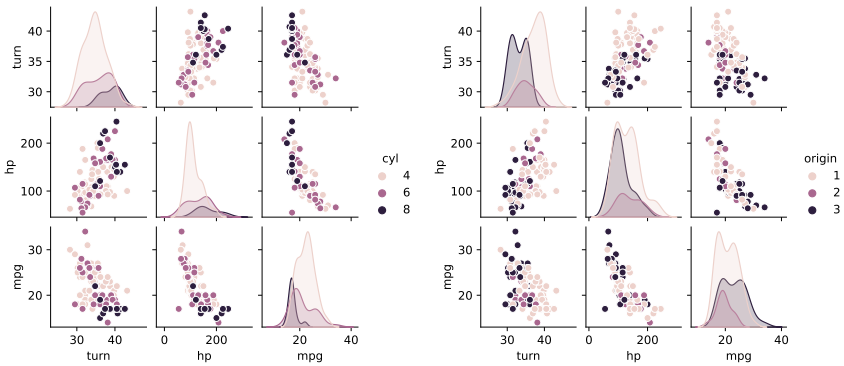


Fig. 4.5: Scatterplot matrix for the **CAR** dataset colored by cylinders (left) and origin (right).

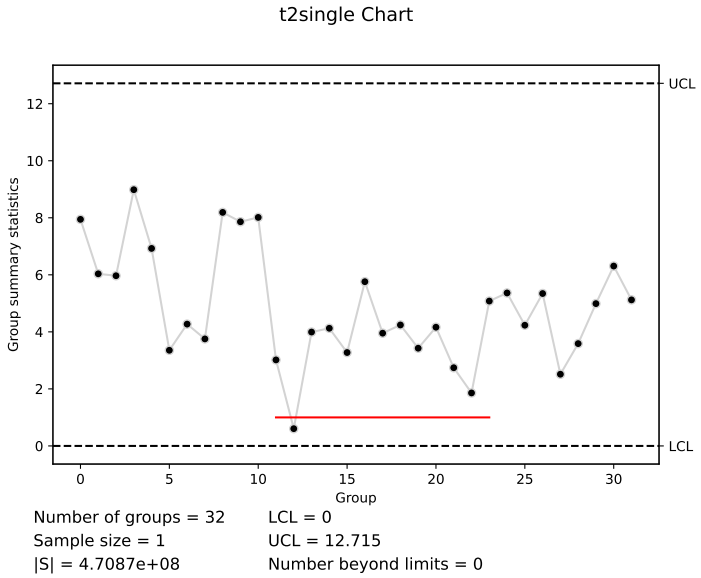


Fig. 4.6: T^2 charts for the **GASOL** dataset. The horizontal line highlights observations 12 to 24 are highlighted

Using observations 12 to 24 as an external assigned target, we get the T^2 chart in Fig. 4.7. All other data points have now T^2 value greater than UCL. The scatterplot matrix shows that the observations 12 to 24 have very distinct values especially for $x1$.

```
gasol = mistat.load_data('GASOL')
base = gasol.iloc[11:24]
mqcc_base = mistat.MultivariateQualityControlChart(base, qcc_type='T2single',
          confidence_level=0.99)
mqcc = mistat.MultivariateQualityControlChart(gasol, qcc_type='T2single',
          center=mqcc_base.stats.center, cov=mqcc_base.stats.cov,
          confidence_level=0.99)
ax = mqcc.plot()
plt.show()
gasol['color'] = ['red' if 11 <= i < 24 else 'black'
                        for i in range(len(gasol))]
sns.pairplot(gasol, hue='color', height=1.75)
plt.show()
```

Exercise 4.6 Repeat Exercise 4.5, but this time design the chart with an externally assigned target based on observations 25–32. Explain the computational difficulty.

Solution 4.6 The T^2 chart and a scatterplot matrix are shown in Fig. 4.8. Compared to Exercise 4.5, the effect is even stronger, as the variation of $x1$ of the subset is even tighter and closer to 0. This results in very large values of T^2.

```
gasol = mistat.load_data('GASOL')
base = gasol.iloc[24:32]
mqcc_base = mistat.MultivariateQualityControlChart(base, qcc_type='T2single',
           confidence_level=0.99)
mqcc = mistat.MultivariateQualityControlChart(gasol, qcc_type='T2single',
          center=mqcc_base.stats.center, cov=mqcc_base.stats.cov,
          confidence_level=0.99)
ax = mqcc.plot()
plt.show()
gasol['color'] = ['red' if 24 <= i < 32 else 'black'
                        for i in range(len(gasol))]
sns.pairplot(gasol, hue='color', height=1.75)
plt.show()
```

Exercise 4.7 Calculate control limits for grouped data with 20 subgroups of size 5 and 6 dimensions, with internally derived targets (Eq. (4.4.2)). How will the control limits change if you start monitoring a process with similar data?

Solution 4.7 With subgroups of size 5, and 6 dimensional data the control limits in Phase II are equal to the control limits used in Phase I.

Exercise 4.8 Let $X_1 = (x_{11}, x_{12}, \ldots, x_{1p})$ and $X_2 = (x_{21}, x_{22}, \ldots, x_{2p})$ represent the mean dissolution values of tablets at p time instances of a reference product and a batch under test, respectively. The Mahalanobis distance T^2, between X_1 and X_2, is defined here as $D_M = \sqrt{(X_2 - X_1)' S^{-1}_{\text{pooled}} (X_2 - X_1)}$, where $S_{\text{pooled}} = (S_{\text{reference}} + S_{\text{test}})/2$, is the pooled covariance matrix of the reference and test samples.

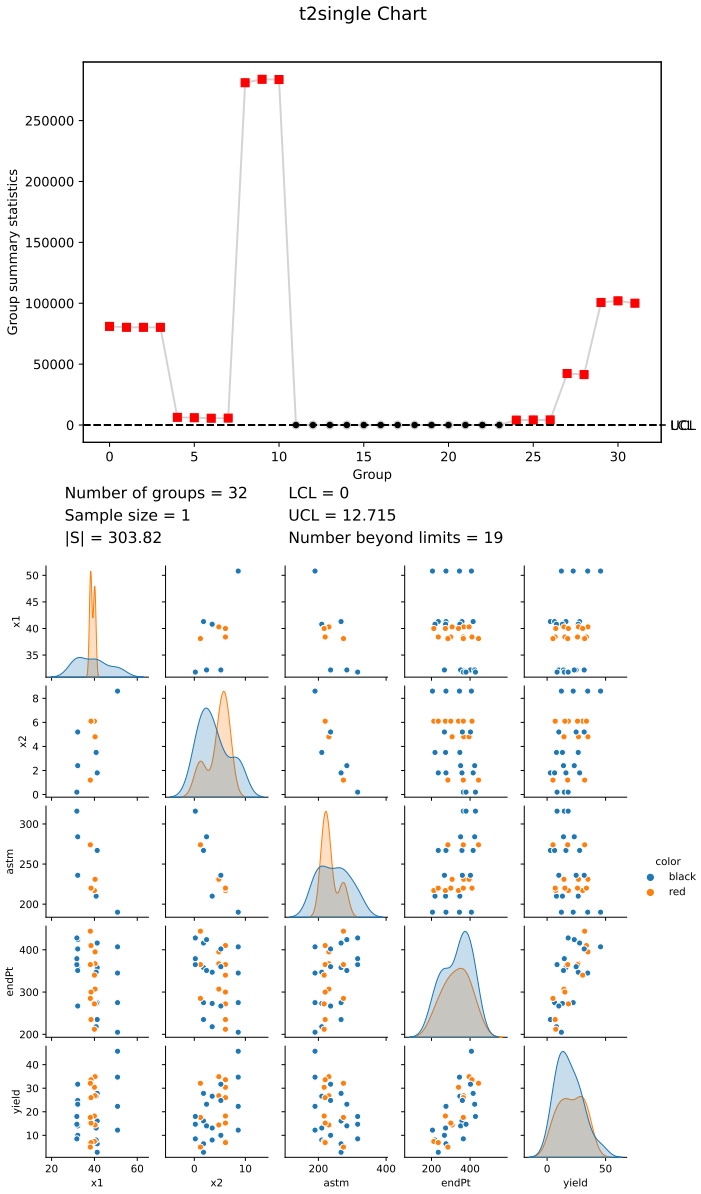


Fig. 4.7: T^2 charts for the **GASOL** dataset using observations 12 to 24 (note the 0-indexing) as an external assigned target. The bottom scatterplot matrix highlights the subset values.

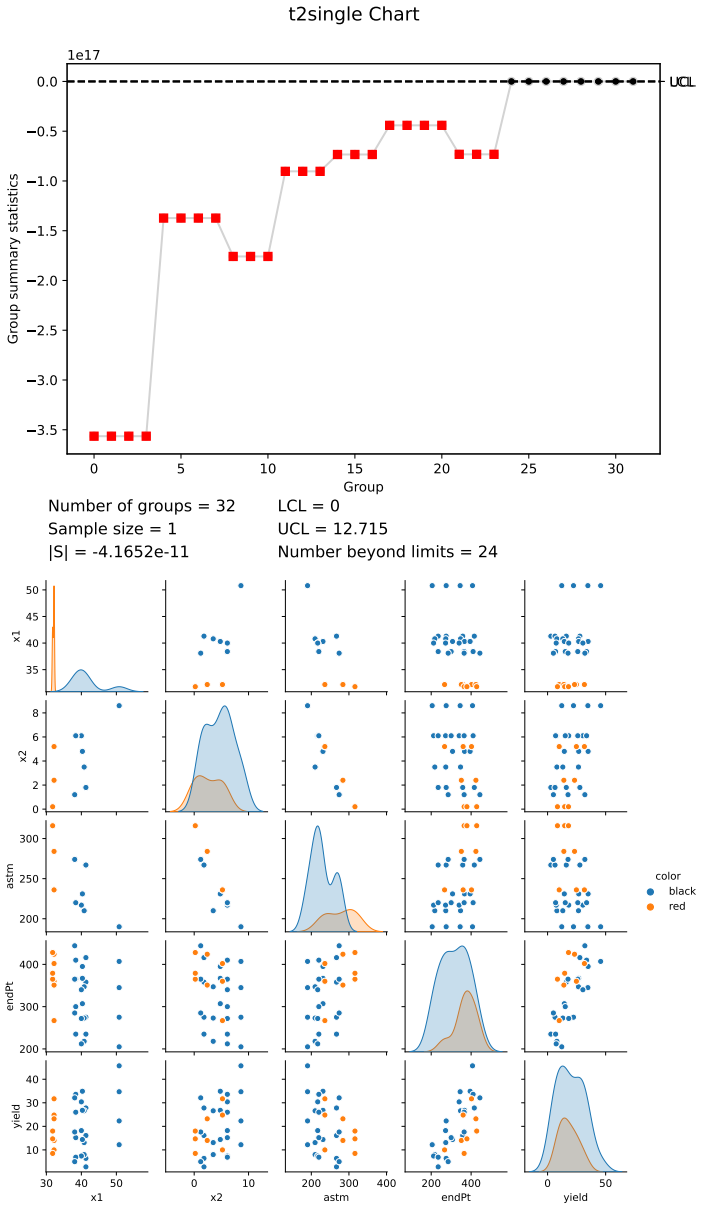


Fig. 4.8: T^2 charts for the **GASOL** dataset using observations 25 to 32 (note the 0-indexing) as an external assigned target. The bottom scatterplot matrix highlights the subset values.

The confidence region, CR, of the difference between batch and reference consists of all vectors Y satisfying: $[(Y-(X_2-X_1)'S^{-1}_{\text{pooled}}(Y-(X_2-X_1)] \leq KF_{0.90}[p, 2n-p-1]$ where $F_{0.90}[p, 2n-p-1]$ is the 90th quantile of the F-distribution with degrees of freedom p and $(2n-p-1)$. Prove that for measurements conducted at one time instance ($p = 1$) these formulae correspond to the confidence intervals presented in Chapter 3 (Kenett et al., 2022b).

Solution 4.8 For $p = 1$, the matrices X_1 and X_2 reduce to a single vector x_1 and x_2. In this case, the confidence region, CR, becomes: confidence region The equation from Sect. 4.5

$$\begin{aligned} CR &= \left\{\mathbf{Y} : (\mathbf{Y}-(\mathbf{x}_1-\mathbf{x}_2))'S^{-1}_{\text{pooled}}(\mathbf{Y}-(\mathbf{x}_1-\mathbf{x}_2)') \leq KF_{1-\alpha}[1, 2n-2]\right\} \\ &= \left\{\mathbf{Y} : (\mathbf{Y}-(\mathbf{x}_1-\mathbf{x}_2))'S^{-1}_{\text{pooled}}(\mathbf{Y}-(\mathbf{x}_1-\mathbf{x}_2)') \leq \frac{4(n-1)}{n(2n-2)}F_{1-\alpha}[1, 2n-2]\right\} \\ &= \left\{\mathbf{Y} : (\mathbf{Y}-(\mathbf{x}_1-\mathbf{x}_2))'S^{-1}_{\text{pooled}}(\mathbf{Y}-(\mathbf{x}_1-\mathbf{x}_2)') \leq \frac{2}{n}F_{1-\alpha}[1, 2(n-1)]\right\} \end{aligned}$$

Confidence interval from Chapter 3 (Kenett et al., 2022b) can be rewritten as follows.

$$\left(\bar{X}-t_{1-\alpha/2}[m-1]\frac{S}{\sqrt{m}}, \bar{X}+t_{1-\alpha/2}[m-1]\frac{S}{\sqrt{m}}\right)$$

$$\bar{X}-t_{1-\alpha/2}[m-1]\frac{S}{\sqrt{m}} \leq y \leq \bar{X}+t_{1-\alpha/2}[m-1]\frac{S}{\sqrt{m}}$$

$$-t_{1-\alpha/2}[m-1]\frac{S}{\sqrt{m}} \leq y-\bar{X} \leq t_{1-\alpha/2}[m-1]\frac{S}{\sqrt{m}}$$

$$(y-\bar{X})^2 \leq \left(t_{1-\alpha/2}[m-1]\frac{S}{\sqrt{m}}\right)^2$$

$$(y-\bar{X})S^{-2}(y-\bar{X}) \leq t^2_{1-\alpha/2}[m-1]\frac{1}{m}$$

Using the relationship $F_\alpha[1,k] = t^2_\alpha[k]$ this becomes:

$$(y-\bar{X})'S^{-2}(y-\bar{X}) \leq \frac{1}{m}F_{1-\alpha/2}[1, m-1]$$

In this case, S is the standard deviation, so we can replace S^2 with the pooled covariance. We also replace $m-1$ with $2n-2$ and derive updated control limits.

$$(y-\bar{X})'S^{-1}_{\text{pooled}}(y-\bar{X}) \leq \frac{1}{2n-1}F_{1-\alpha/2}[1, 2n-2]$$

Chapter 5
Classical Design and Analysis of Experiments

For the most recent version of the solution manual, go to `https://gedeck.github.io/mistat-code-solutions/IndustrialStatistics/`.

Import required modules and define required functions

```
import itertools
import numpy as np
import pandas as pd
from scipy import stats
import statsmodels.formula.api as smf
from statsmodels.stats import anova
import matplotlib.pyplot as plt
from pyDOE2 import fracfact

import mistat
```

Exercise 5.1 Describe a production process familiar to you, like baking of cakes, or manufacturing concrete. List the pertinent variables. What is (are) the response variable(s)? Classify the variables which affect the response to noise variables and control variables. How many levels would you consider for each variable?

Solution 5.1 Preparing pancakes is a production process readily available for everyone to experiment with. The steps are as follows:

1. Open readymix box.
2. Measure prespecified amount of readymix into measuring cup.
3. Pour prespecified amount of readymix into container.
4. Measure prespecified amount of water in measuring cup.
5. Add prespecified amount of water to container.
6. Mix material in container with one or two eggs.
7. Warm cooking pan.
8. Add oil to pan in liquid, solid or spray form.
9. Pour material into cooking pan for one or more pancakes.
10. Wait for darkening signs on the pancakes rim.
11. Turn over pancakes.
12. Wait again.

13. Remove pancakes from cooking pan.

Examples of response variables include subjective taste testing by household members with classification on a 1 to 5 scale, drop test for measuring pancake body composition (drop pancake from the table to the floor and count the number of pieces it decomposes into), quantitative tests performed in the laboratory to determine pancake chemical composition and texture.

Noise variables include environmental temperature and humidity, variability in amounts of readymix, water, oil and cooking time, stove heating capacity and differences in quality of raw materials.

Examples of control variables include:

- Amounts of readymix-less and more than present value
- Preheating time of cooking pan-less and more than present value
- Pouring speed of water into readymix container-slower and faster
- Waiting time before turn over-not just ready and overdone.

Exercise 5.2 Different types of adhesives are used in a lamination process, in manufacturing a computer card. The card is tested for bond strength. In addition to the type of adhesive, a factor which might influence the bond strength is the curing pressure (currently at 200 psi). Follow the basic steps of experimental design to set a possible experiment for testing the effects of adhesives and curing pressure on the bond strength.

Solution 5.2 Response variable: Bond strength test. Controllable factors: Adhesive type, Curing pressure. Factor levels: Adhesive type - A, B, C, D, Curing pressure - low, nominal, high. Experimental layout: 4x3 full factorial experiment with 4 replications.

Experimental Run	Adhesive type	Curing pressure	Replication			
1	A	low	1	2	3	4
2	A	nominal	1	2	3	4
3	A	high	1	2	3	4
4	B	lowl	1	2	3	4
5	B	nominal	1	2	3	4
...	...	...	...	...	...	...

Experiment protocol:

1. Prepare 48 (3x4x4) computer cards.
2. Randomly split computer cards into 4 groups, 12 cards per group.
3. Randomly assign adhesive type to each group, 1 type per group.
4. Randomly assign curing pressure level to group of 12 cards, 1 level for 4 cards.
5. Attach to each computer card a sticker indicating experimental run, replication number, adhesive type and curing pressure.
6. Randomize the order of the 48 cards.

7. Run experiment according to randomized order and factorlevels indicated on sticker by keeping factors not participating in experiment fixed (e.g., amount of water).
8. Perform bond strength test and record data.
9. Analyze data with statistical model including main effects and interaction terms.

Exercise 5.3 Provide an example where blocking can reduce the variability of a product.

Solution 5.3 Blocking reduces the variability of the product relative to factors being studied. Examples include:

- Track and field athletes on a college team decided to investigate the effect of sleeping hours on athletic performance. The experiment consisted of sleeping a controlled amount of time prior to competitions. Individual athletes are natural experimental blocks.
- A natural extension of the example in section 5.2 on testing shoe sole materials is an experiment designed to test car tires. A natural block consists of the four wheels of a car, with an additional blocking variable determined by the position of the tires (front or rear).
- Experiments performed on plants are known to be sensitive to environmental conditions. Blocking variables in such experiments consist of neighboring plots of land where soil, humidity and temperature conditions are similar.

Exercise 5.4 Three factors A, B, C are tested in a given experiment, designed to assess their effects on the response variable. Each factor is tested at 3 levels. List all the main effects and interactions.

Solution 5.4 Main effects: τ_i^A, τ_j^B, τ_k^C each at 3 levels. First order interactions: τij^{AB}, τ_{ik}^{AC}, τ_{jk}^{BC} each at 9 levels. Second order interaction: τ_{ijk}^{ABC} at 27 levels.

Exercise 5.5 Let x_1, x_2 be two quantitative factors and Y a response variable. A regression model $Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_{12} x_1 x_2 + e$ is fitted to the data. Explain why β_{12} can be used as an interaction parameter.

Solution 5.5 $Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_{12} x_1 x_2 + e$. If $\beta_{12} = 0$ then the model is additive. The regression of Y on x_1 (x_2) are parallel for different values of x_2 (x_1). On the other hand, if $\beta_{12} \neq 0$ then the regression of Y on x_1, for two values of x_2, say $x_2^{(1)}$ and $x_2^{(2)}$, are $Y = \beta_0 + (\beta_1 + \beta_{12} x_2^{(1)}) x_1 + \beta_2 x_2^{(1)} + e$, and $Y = \beta_0 + (\beta_1 + \beta_{12} x_2^{(2)}) x_1 + \beta_2 x_2^{(2)} + e$ These are regression lines with different slopes. This means that the model is nonadditive and the extent of the interaction depends on β_{12}.

Exercise 5.6 Consider the ISC values for times t_1, t_2 and t_3 in dataset **SOCELL.csv**. Make a paired comparison for testing whether the mean ISC in time t_2 is different from that in time t_1, by using a t-test.

Solution 5.6 Use the function *ttest_rel* from `scipy` to run a paired t-test.

```
ISC = mistat.load_data('SOCELL')
stats.ttest_rel(ISC['t2'], ISC['t1'])
```

```
Ttest_relResult(statistic=11.302222594952937,
pvalue=9.758913823226797e-09)
```

The difference between the two means is significant

Exercise 5.7 Use *permutation_test* from `scipy` to perform a randomization test for the differences in the ISC values of the solar cells in times t_2 and t_3 (dataset **SOCELL.csv**).

Solution 5.7 The permutation test requires the definition of the target statistic.

```
def statistic(x, y):
    return np.mean(x) - np.mean(y)
```

The permutation test is then run as follows. The keyword argument `permutation_type='samples'` corresponds to running a paired t-test.

```
res = stats.permutation_test((ISC['t2'], ISC['t3']), statistic,
                             permutation_type='samples', n_resamples=1000)
res.pvalue.round(5)
```

```
0.11389
```

A standard t-test gives;

```
stats.ttest_rel(ISC['t2'], ISC['t3']).pvalue.round(5)
```

```
0.11567
```

Both results are comparable. We can also visualize the distribution of the statistic; see Fig. 5.1.

```
fig, ax = plt.subplots()
ax.hist(res.null_distribution, bins=20, color='lightgrey')
ax.axvline(statistic(ISC['t2'], ISC['t3']), color='black')
plt.show()
```

Exercise 5.8 Box et al. (2005) give the results of four treatments A, B, C, D in penicillin manufacturing in five different blends (blocks) shown in Table 5.1.

Perform an ANOVA to test whether there are significant differences between the treatments or between the blends.

Solution 5.8 First create the dataset:

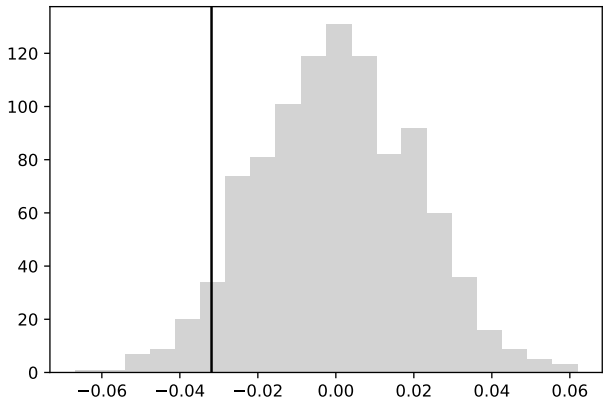


Fig. 5.1: Distribution resulting from the randomization paired comparison

Table 5.1: Result of four treatments in pencillin manufacturing

	Treatments			
blends	*A*	*B*	*C*	*D*
1	89	88	97	94
2	84	77	92	79
3	81	87	87	85
4	87	92	89	84
5	79	81	80	88

```
df = pd.DataFrame([
  ['B1', 'A', 89], ['B1', 'B', 88], ['B1', 'C', 97], ['B1', 'D', 94],
  ['B2', 'A', 84], ['B2', 'B', 77], ['B2', 'C', 92], ['B2', 'D', 79],
  ['B3', 'A', 81], ['B3', 'B', 87], ['B3', 'C', 87], ['B3', 'D', 85],
  ['B4', 'A', 87], ['B4', 'B', 92], ['B4', 'C', 89], ['B4', 'D', 84],
  ['B5', 'A', 79], ['B5', 'B', 81], ['B5', 'C', 80], ['B5', 'D', 88],
], columns=['blend', 'treatment', 'result'])
```

Then use `statsmodels` to perform an ANOVA.

```
model = smf.ols('result ~ C(blend) + C(treatment)', data=df).fit()
anova.anova_lm(model)
```

```
                df  sum_sq    mean_sq         F    PR(>F)
C(blend)       4.0   264.0  66.000000  3.504425  0.040746
C(treatment)   3.0    70.0  23.333333  1.238938  0.338658
Residual      12.0   226.0  18.833333       NaN       NaN
```

There are no significant differences between the treatments. There are significant differences between blends, the most extreme difference being between blend 1 and blend 5. See also Fig. 5.2.

```
fig, axes = plt.subplots(ncols=2)
df.groupby('blend').boxplot(column='result', subplots=False,
```

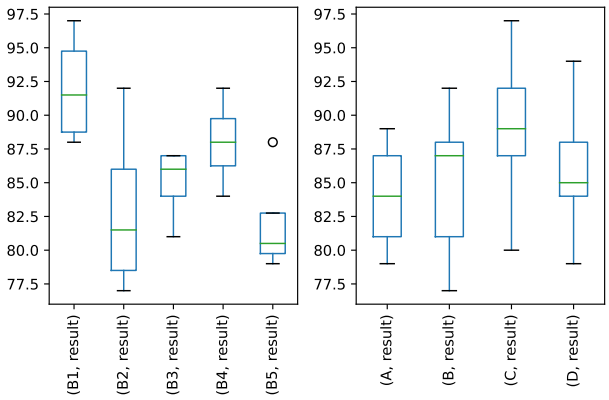


Fig. 5.2: Boxplot representation shows differences of blend and treatment effect

Table 5.2: Results of treatments $A, B, C, \cdots, H$ for different treatments

block	Treatments				Block	Treatments			
1	A	38	B	30	15	D	11	G	24
2	C	50	D	27	16	F	37	H	39
3	E	33	F	28	17	A	23	F	40
4	G	62	H	30	18	B	20	D	14
5	A	37	C	25	19	C	18	H	10
6	B	38	H	52	20	E	22	G	52
7	D	89	E	89	21	A	66	G	67
8	F	27	G	75	22	B	23	F	46
9	A	17	D	25	23	C	22	E	28
10	B	47	G	63	24	D	20	H	40
11	C	32	F	39	25	A	27	H	32
12	E	20	H	18	26	B	10	E	40
13	A	5	E	15	27	C	32	G	33
14	B	45	C	38	28	D	18	F	23

```
                          rot=90, grid=False, ax=axes[0])
df.groupby('treatment').boxplot(column='result', subplots=False,
                          rot=90, grid=False, ax=axes[1])
plt.tight_layout()
plt.show()
```

Exercise 5.9 Eight treatments $A, B, C, \cdots, H$ were tested in a BIBD of 28 blocks, $k = 2$ treatments per block, $r = 7$ and $\lambda = 1$. The results of the experiments are shown in Table 5.2

Make an ANOVA to test the significance of the block effects, treatment effects, If the treatment effects are significant, make multiple comparisons of the treatments.

Solution 5.9 First create the dataset:

```
df = pd.DataFrame([
    [1, 'A', 38], [1, 'B', 30],   [2, 'C', 50], [2, 'D', 27],
    [3, 'E', 33], [3, 'F', 28],   [4, 'G', 62], [4, 'H', 30],
    [5, 'A', 37], [5, 'C', 25],   [6, 'B', 38], [6, 'H', 52],
    [7, 'D', 89], [7, 'E', 89],   [8, 'F', 27], [8, 'G', 75],
    [9, 'A', 17], [9, 'D', 25],   [10, 'B', 47], [10, 'G', 63],
    [11, 'C', 32], [11, 'F', 39], [12, 'E', 20], [12, 'H', 18],
    [13, 'A', 5], [13, 'E', 15],  [14, 'B', 45], [14, 'C', 38],
    [15, 'D', 11], [15, 'G', 24], [16, 'F', 37], [16, 'H', 39],
    [17, 'A', 23], [17, 'F', 40], [18, 'B', 20], [18, 'D', 14],
    [19, 'C', 18], [19, 'H', 10], [20, 'E', 22], [20, 'G', 52],
    [21, 'A', 66], [21, 'G', 67], [22, 'B', 23], [22, 'F', 46],
    [23, 'C', 22], [23, 'E', 28], [24, 'D', 20], [24, 'H', 40],
    [25, 'A', 27], [25, 'H', 32], [26, 'B', 10], [26, 'E', 40],
    [27, 'C', 32], [27, 'G', 33], [28, 'D', 18], [28, 'F', 23],
], columns=['block', 'treatment', 'result'])
```

The ANOVA gives the following result:

```
model = smf.ols('result ~ C(block) + C(treatment)', data=df).fit()
anova.anova_lm(model)
```

	df	sum_sq	mean_sq	F	PR(>F)
C(block)	27.0	15030.482143	556.684524	5.334113	0.000109
C(treatment)	7.0	1901.875000	271.696429	2.603376	0.042207
Residual	21.0	2191.625000	104.363095	NaN	NaN

Both the effects of the treatments and the blocks are significant at the $\alpha = 0.05$ level.

The Scheffé coefficient for $\alpha = 0.05$ is $S_{0.05} = (7 \times F_{0.95}[7, 21])^{1/2} = 4.173$ and $\hat{c}_p = \sqrt{104.363} = 10.216$. Thus the treatments can be divided into two homogenous groups, $G_1 = \{A, B, C, D, E, F, H\}$ and $G_2 = \{G\}$. The 0.95 confidence interval for the difference of the group means is 20.413 ± 17.225 which shows that the group means are significantly different. See Fig. 5.3 for a visualization of the different groups.

```
df['group'] = ['G1' if t == 'G' else 'G2' for t in df['treatment']]
fig, axes = plt.subplots(ncols=3)
df.groupby('block').boxplot(column='result', subplots=False,
                            rot=90, grid=False, ax=axes[0])
df.groupby('treatment').boxplot(column='result', subplots=False,
                            rot=90, grid=False, ax=axes[1])
df.groupby('group').boxplot(column='result', subplots=False,
                            rot=90, grid=False, ax=axes[2])
plt.tight_layout()
plt.show()
```

Exercise 5.10 Four different methods of preparing concrete mixtures A, B, C, D were tested, these methods consisted of two different mixture ratios of cement to water and two blending duration). The four methods (treatments) were blocks in four batches and four days, according to a Latin square design. The concrete was poured to cubes and tested for compressive strength [Kg/cm^2] after 7 days of storage in special rooms with 20°C temperature and 50% relative humidity. The results are in Table 5.3

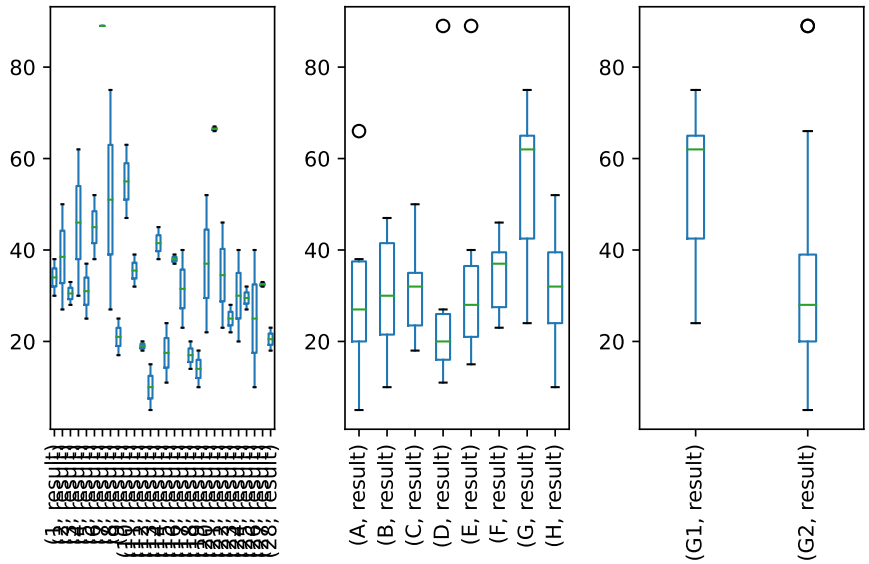


Fig. 5.3: Boxplot representation for Exercise 5.9

Table 5.3: Compressive strengths of cement for different treatments

	batches			
Days	1	2	3	4
1	*A* 312	*B* 299	*C* 315	*D* 290
2	*C* 295	*A* 317	*D* 313	*B* 300
3	*B* 295	*D* 298	*A* 312	*C* 315
4	*D* 313	*C* 314	*B* 299	*A* 300

Are the differences between the strength values of different treatments significant? [Perform the ANOVA.]

Solution 5.10 Prepare the dataset.

```
df = pd.DataFrame([
    [1,1,'A',312], [1,2,'B',299], [1,3,'C',315], [1,4,'D',290],
    [2,1,'C',295], [2,2,'A',317], [2,3,'D',313], [2,4,'B',300],
    [3,1,'B',295], [3,2,'D',298], [3,3,'A',312], [3,4,'C',315],
    [4,1,'D',313], [4,2,'C',314], [4,3,'B',299], [4,4,'A',300],
], columns=['day', 'batch', 'mixture', 'result'])
```

Build a model and perform an ANOVA.

```
model = smf.ols('result ~ C(day) + C(batch) + C(mixture)', data=df).fit()
anova.anova_lm(model)
```

	df	sum_sq	mean_sq	F	PR(>F)
C(day)	3.0	16.1875	5.395833	0.048079	0.984704
C(batch)	3.0	165.6875	55.229167	0.492111	0.700652
C(mixture)	3.0	388.6875	129.562500	1.154446	0.401167
Residual	6.0	673.3750	112.229167	NaN	NaN

The differences between mixtures, batches or days are not significant.

Exercise 5.11 Repeat the experiments described in Example 5.7 at the low levels of factors m, v_0, p_0, t, and t_0. Perform the ANOVA for the main effects and interaction of spring coefficient k and piston weight m on the cycle time. Use a marginal interaction plot to visualize both main effects and interaction. Are your results different from those obtained in the example?

Solution 5.11 In Python:

```
from mistat.design import doe
np.random.seed(2)

# Build design from factors
FacDesign = doe.full_fact({
    'k': [1500, 3000, 4500],
    's': [0.005, 0.0125, 0.02],
})

# Randomize design
FacDesign = FacDesign.sample(frac=1).reset_index(drop=True)

# Setup and run simulator with five replicates
# for each combination of factors
simulator = mistat.PistonSimulator(n_replicate=5, **FacDesign,
                                   m=30, v0=0.005, p0=90_000, t=290, t0=340)
result = simulator.simulate()

model = smf.ols('seconds ~ C(k) * C(s)', data=result).fit()
print(anova.anova_lm(model).round(4))
```

	df	sum_sq	mean_sq	F	PR(>F)
C(k)	2.0	0.0039	0.0019	2.0508	0.1434
C(s)	2.0	0.1030	0.0515	54.8672	0.0000
C(k):C(s)	4.0	0.0058	0.0015	1.5531	0.2078
Residual	36.0	0.0338	0.0009	NaN	NaN

Fig. 5.4 shows the marginal interaction plot.

```
_, ax = plt.subplots(figsize=[5, 4])
mistat.marginalInteractionPlot(result[['s', 'k', 'seconds']], 'seconds', ax=ax)
plt.show()
```

Compared to Example 5.7, the cycle times are consistently lower. However, their behavior on changing the spring coefficient or the piston weight is similar.

Exercise 5.12 For the data from Exercise 5.11 compute the least squares estimates of the main effects on the means and on the standard deviations.

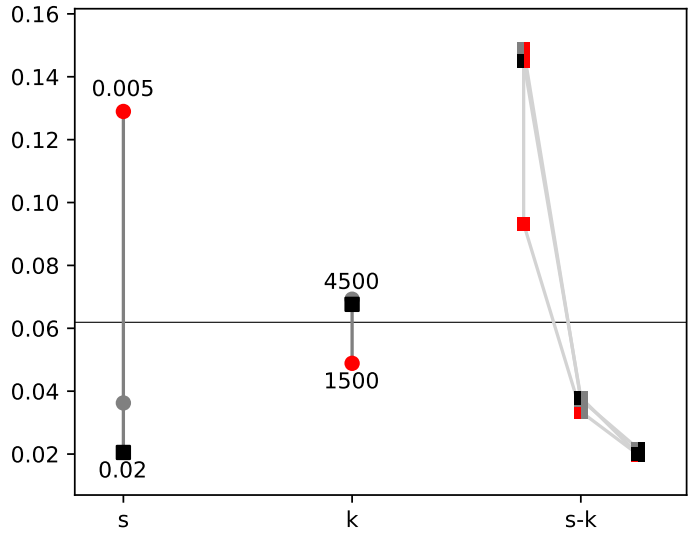


Fig. 5.4: Effect of Spring Coefficient k and Piston surface area s on Cycle Time

Solution 5.12 Calculate the means and standard deviations of the cycle time by grouping on s and k.

```
# group and aggregate seconds by group
grouped = result.groupby(['s', 'k']).agg({'seconds': ['mean', 'std']})
# convert the multi-level index back to columns
grouped = grouped.reset_index()
# rename the columns to remove the multi-level index for the column names
grouped.columns = ['s', 'k', 'mean', 'std']
print(grouped)
```

```
        s     k      mean       std
0  0.0050  1500  0.093227  0.026990
1  0.0050  3000  0.148749  0.066597
2  0.0050  4500  0.144906  0.056237
3  0.0125  1500  0.033497  0.003607
4  0.0125  3000  0.037292  0.007923
5  0.0125  4500  0.037975  0.005075
6  0.0200  1500  0.019859  0.001296
7  0.0200  3000  0.021716  0.002626
8  0.0200  4500  0.019957  0.003601
```

Use `statsmodels` for the least squares estimates of `mean`

```
model = smf.ols(f"mean ~ s + k + s:k", data=grouped).fit()
model.params
```

```
Intercept    0.090547
s           -3.791223
k            0.000021
s:k         -0.001146
dtype: float64
```

and `std`.

```
model = smf.ols(f"std ~ s + k + s:k", data=grouped).fit()
model.params
```

```
Intercept    0.025397
s           -1.366082
k            0.000011
s:k         -0.000599
dtype: float64
```

Exercise 5.13 A 2^4 factorial experiment gave the following response values, arranged in standard order: 72, 60, 90, 80, 65, 60, 85, 80, 60, 50, 88, 82, 58, 50, 84, 75.

(i) Estimate all possible main effects.
(ii) Estimate σ^2 under the assumption that all the interaction parameters are zero.
(iii) Determine a confidence interval for σ^2 at level of confidence 0.99.

Solution 5.13 Construct a data frame for a 2^4 factorial design in standard order:

```
treatments = []
for combo in itertools.product([0, 1], [0,1], [0,1], [0,1]):
    nu = sum(ij * 2**(j-1) for j, ij in enumerate(combo, 1))
    treatments.append({
        'nu': int(nu),
        'A': combo[0], 'B': combo[1],
        'C': combo[2], 'D': combo[3],
    })
# sort to standard order
df = pd.DataFrame(treatments).sort_values('nu', ignore_index=True)
df = df.set_index('nu')
# change the factors to (-1, 1)
for factor in ('A', 'B', 'C', 'D'):
    df.loc[df[factor] == 0, factor] = -1
```

Combine with the response values.

```
df['response'] = [72, 60, 90, 80, 65, 60, 85, 80, 60, 50,
                  88, 82, 58, 50, 84, 75]
```

(i)

```
model = smf.ols("response ~ A + B + C + D", data=df).fit()
model.params
```

```
Intercept    71.1875
A            -4.0625
B            11.8125
C            -1.5625
D            -2.8125
dtype: float64
```

The LSE of the main effects are $\hat{A} = -4.0625$, $\hat{B} = 11.8125$, $\hat{C} = -1.5625$ and $\hat{D} = -2.8125$.

(ii) An estimate of σ^2 with 11 d.f. is $\hat{\sigma}^2 = 9.2898$.

Table 5.4: Results of 3^2 factorial experiment with $n = 3$ observations

	A_1	A_2	A_3
B_1	18.3	17.9	19.1
	17.9	17.6	19.0
	18.5	16.2	18.9
B_2	20.5	18.2	22.1
	21.1	19.5	23.5
	20.7	18.9	22.9
B_3	21.5	20.1	22.3
	21.7	19.5	23.5
	21.9	18.9	23.3

```
sigma2 = np.sum(model.resid**2) / 11
```

(iii) A 0.99 level confidence interval for σ^2 is (3.819,39.254).

```
df = 11
alpha_left = 1 - (1-0.99) / 2
alpha_right = (1-0.99) / 2
df * sigma2 / stats.chi2.ppf(alpha_left, df), df * sigma2 / stats.chi2.ppf(alpha_right, df)
```

```
(3.8191156334967578, 39.25424120484004)
```

Exercise 5.14 A 3^2 factorial experiment, with $n = 3$ replications, gave the observations in Table 5.4.

Perform an ANOVA to test the main effects and interactions. Break the between treatments sum of squares to one degree of freedom components. Use the Scheffé S_α coefficient to determine which effects are significant.

Solution 5.14 Prepare the dataset:

```
df = pd.DataFrame(
    [['A1', 'B1', v] for v in [18.3, 17.9, 18.5]] +
    [['A2', 'B1', v] for v in [17.9, 17.6, 16.2]] +
    [['A3', 'B1', v] for v in [19.1, 19.0, 18.9]] +
    [['A1', 'B2', v] for v in [20.5, 21.1, 20.7]] +
    [['A2', 'B2', v] for v in [18.2, 19.5, 18.9]] +
    [['A3', 'B2', v] for v in [22.1, 23.5, 22.9]] +
    [['A1', 'B3', v] for v in [21.5, 21.7, 21.9]] +
    [['A2', 'B3', v] for v in [20.1, 19.5, 18.9]] +
    [['A3', 'B3', v] for v in [22.3, 23.5, 23.3]],
    columns=['a', 'b', 'result']
)
```

Build the model and performan an ANOVA.

```
model = smf.ols('result ~ C(a) + C(b) + C(a):C(b)', data=df).fit()
anova.anova_lm(model)
```

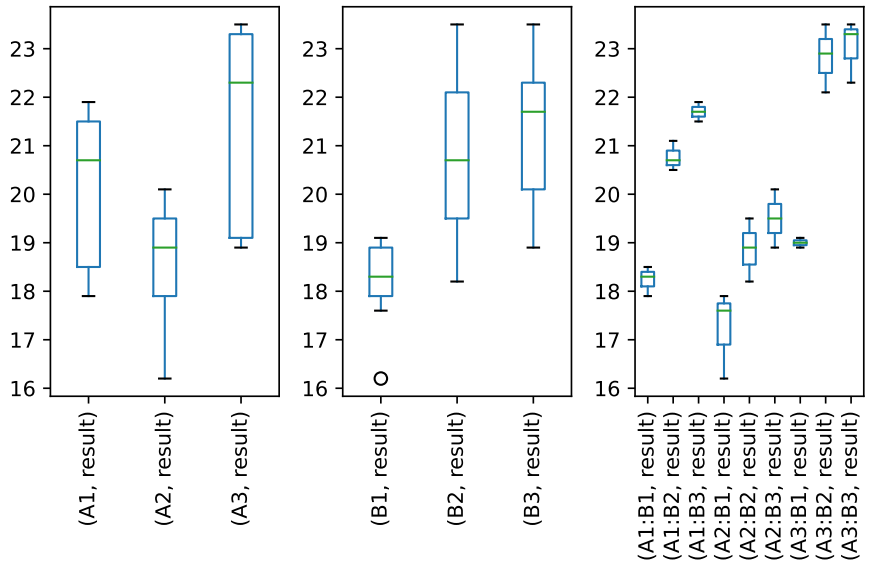


Fig. 5.5: Boxplot representation for Exercise 5.14

	df	sum_sq	mean_sq	F	PR(>F)
C(a)	2.0	43.080741	21.540370	70.495758	3.055533e-09
C(b)	2.0	54.169630	27.084815	88.641212	4.802685e-10
C(a):C(b)	4.0	4.345926	1.086481	3.555758	2.633500e-02
Residual	18.0	5.500000	0.305556	NaN	NaN

Visualization of the factor and factor interactions are shown in Fig. 5.5

```
df['a:b'] = [f'{a}:{b}' for a, b in zip(df['a'], df['b'])]
fig, axes = plt.subplots(ncols=3)
df.groupby('a').boxplot(column='result', subplots=False,
                        rot=90, grid=False, ax=axes[0])
df.groupby('b').boxplot(column='result', subplots=False,
                        rot=90, grid=False, ax=axes[1])
df.groupby('a:b').boxplot(column='result', subplots=False,
                        rot=90, grid=False, ax=axes[2])
plt.tight_layout()
plt.show()
```

Exercise 5.15 Construct a 2^{8-2} fractional replication, using the generators $ABCDG$ and $ABEFH$. What is the resolution of this design? Write the aliases to the main effects, and to the first order interactions with the factor A.

Solution 5.15 The subgroup of defining parameters is

```
generators = ['ABCDG', 'ABEFH']
print(mistat.subgroupOfDefining(generators))
```

```
['', 'ABCDG', 'ABEFH', 'CDEFGH']
```

The design is therefore of resolution **V**.
The aliases of the design using the generators are:

```
from mistat.design.doeUtilities import aliasesInSubgroup
for main_effect in ('A', 'B', 'C', 'D', 'E', 'F', 'G', 'H'):
    print(main_effect,
          aliasesInSubgroup(main_effect, generators))
```

```
A ['ACDEFGH', 'BCDG', 'BEFH']
B ['ACDG', 'AEFH', 'BCDEFGH']
C ['ABCEFH', 'ABDG', 'DEFGH']
D ['ABCG', 'ABDEFH', 'CEFGH']
E ['ABCDEG', 'ABFH', 'CDFGH']
F ['ABCDFG', 'ABEH', 'CDEGH']
G ['ABCD', 'ABEFGH', 'CDEFH']
H ['ABCDGH', 'ABEF', 'CDEFG']
```

The shortest alias are of length four. This means our design has resolution **IV**.
The aliases for the first order interaction with the factor A are:

```
for interaction in ('B', 'C', 'D', 'E', 'F', 'G', 'H'):
    print(f'A{interaction}',
          aliasesInSubgroup(f'A{interaction}', generators))
```

```
AB ['ABCDEFGH', 'CDG', 'EFH']
AC ['ADEFGH', 'BCEFH', 'BDG']
AD ['ACEFGH', 'BCG', 'BDEFH']
AE ['ACDFGH', 'BCDEG', 'BFH']
AF ['ACDEGH', 'BCDFG', 'BEH']
AG ['ACDEFH', 'BCD', 'BEFGH']
AH ['ACDEFG', 'BCDGH', 'BEF']
```

You can construct a block of the 2^{8-2} design using this Python code.

```
generator = 'A B C D E F G H ABCDG ABEFH'
design = pd.DataFrame(fracfact(generator), columns=generator.split())
fracfact_design = design.query('ABCDG == 1 & ABEFH == 1')
```

Exercise 5.16 Consider a full factorial experiment of $2^6 = 64$ runs. It is required to partition the runs to 8 blocks of 8. The parameters in the group of defining parameters are confounded with the effects of blocks and are not estimable. Show which parameters are not estimable if the blocks are generated by ACE, $ABEF$, and $ABCD$.

Solution 5.16 The subgroup of defining parameters and therefore the parameters that are not estimable for this generator are:

```
print(mistat.subgroupOfDefining(['ACE', 'ABEF', 'ABCD']))
```

```
['', 'ABCD', 'ABEF', 'ACE', 'ADF', 'BCF', 'BDE', 'CDEF']
```

Table 5.5: Design matrix and response of 2^2 factorial design

X_1	X_2	Y
−1	−1	55.8
−1	−1	54.4
1	−1	60.3
1	−1	60.9
−1	1	63.9
−1	1	64.4
1	1	67.9
1	1	68.5
0	0	61.5
0	0	62.0
0	0	61.9
0	0	62.4

Exercise 5.17 A 2^2 factorial design is expanded by using 4 observations at 0. The design matrix and the response are in Table 5.5.

(i) Fit a response function of the form: $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_{12} X_1 X_2 + e$, and plot its contour lines.
(ii) Estimate the variance σ^2 and test the goodness of fit of this model.

Solution 5.17 (i) Prepare the dataset.

```
df = pd.DataFrame([
    [-1, -1, 55.8], [-1, -1, 54.4], [1, -1, 60.3], [1, -1, 60.9],
    [-1, 1, 63.9], [-1, 1, 64.4], [1, 1, 67.9], [1, 1, 68.5],
    [0, 0, 61.5], [0, 0, 62.0], [0, 0, 61.9], [0, 0, 62.4]
], columns=['X1', 'X2', 'Y'])
```

Build the model.

```
formula = ('Y ~ X1 + X2 + X1:X2')
model = smf.ols(formula, data=df).fit()
print(model.summary2())
```

```
                 Results: Ordinary least squares
=================================================================
Model:              OLS              Adj. R-squared:     0.986
Dependent Variable: Y                AIC:                19.8454
Date:               2023-04-25 11:37 BIC:                21.7851
No. Observations:   12               Log-Likelihood:     -5.9227
Df Model:           3                F-statistic:        262.0
Df Residuals:       8                Prob (F-statistic): 2.52e-08
R-squared:          0.990            Scale:              0.23568
-----------------------------------------------------------------
              Coef.   Std.Err.     t      P>|t|    [0.025   0.975]
-----------------------------------------------------------------
Intercept    61.9917    0.1401  442.3492  0.0000  61.6685  62.3148
X1            2.3875    0.1716   13.9101  0.0000   1.9917   2.7833
X2            4.1625    0.1716   24.2516  0.0000   3.7667   4.5583
X1:X2        -0.3625    0.1716   -2.1120  0.0677  -0.7583   0.0333
-----------------------------------------------------------------
Omnibus:              0.321        Durbin-Watson:           2.580
```

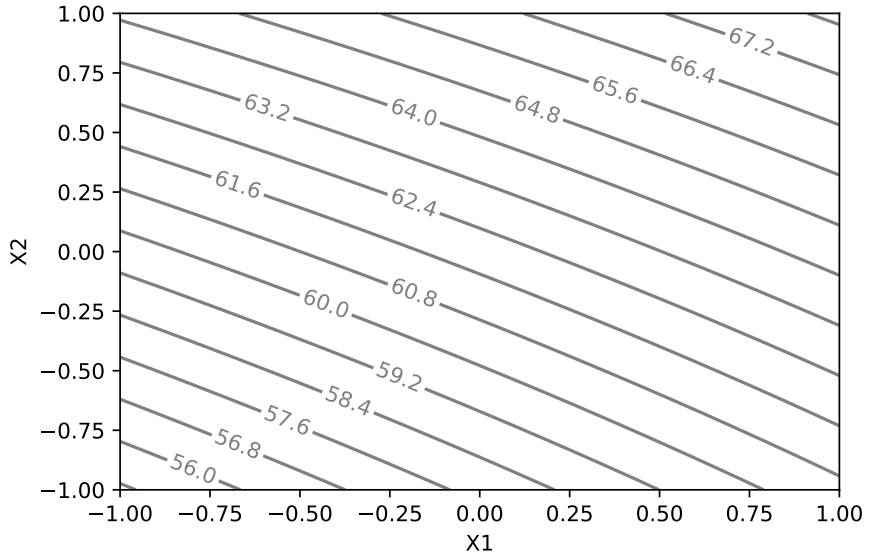


Fig. 5.6: Contour plot of equal responses (Exercise 5.17)

```
Prob(Omnibus):        0.852        Jarque-Bera (JB):        0.447
Skew:                 0.054        Prob(JB):                0.800
Kurtosis:             2.061        Condition No.:           1
==============================================================================
```

The response function is $\hat{Y} = 62.0 + 2.39X_1 + 4.16X_2 - 0.363X_1X_2$, with $R^2 = 0.99$. The contour plot is given in Figure 11.1.

```
def plotResponseSurface(model, ncontours=20):
    x1 = np.linspace(-1, 1)
    x2 = np.linspace(-1, 1)
    X1, X2 = np.meshgrid(x1, x2)
    exog = pd.DataFrame({'X1': X1.ravel(), 'X2': X2.ravel()})
    responses = model.predict(exog=exog)
    CS = plt.contour(x1, x2,
                responses.values.reshape(len(x2), len(x1)),
                ncontours, colors='gray')
    ax = plt.gca()
    ax.clabel(CS, inline=True, fontsize=10)
    ax.set_xlabel('X1')
    ax.set_ylabel('X2')
    return ax

plotResponseSurface(model)
plt.show()
```

(ii)

```
# derive variance around the regression using an ANOVA (mean_sq of residuals)
res = anova.anova_lm(model)
res
```

```
            df      sum_sq     mean_sq           F        PR(>F)
X1         1.0   45.601250   45.601250  193.490387  6.905851e-07
X2         1.0  138.611250  138.611250  588.140552  8.915958e-09
X1:X2      1.0    1.051250    1.051250    4.460552  6.766219e-02
Residual   8.0    1.885417    0.235677         NaN           NaN
```

Using the four observations at (0,0) we derive an estimate of the variance $\hat{\sigma}^2 = 0.13667$, 3 d.f. The variance around the regression is $s^2_{y|(x)} = 0.2357$, 8 d.f. Use F-distribution to derive the p-value.

```
# Estimate of variance?
# ['Residual', 'mean_sq'] gives variance around regression
sigma2 =  df[df['X1'] == 0]['Y'].var()
var_residuals = res.loc['Residual', 'mean_sq']

F = var_residuals / sigma2
p = 1 - stats.f(8, 3).cdf(F)
print(f'F-ratio: {F:.4f}; p-value: {p:.2f}')
```

```
F-ratio: 1.7245; p-value: 0.36
```

The ratio $F = \frac{s^2_{y|(x)}}{\hat{\sigma}^2} = 1.7246$ gives a p-value of $P = 0.36$. This shows there is no significant differences between the variances.

Table 5.6: Design matrix and the response for a control composite design of Exercise 5.18

X_1	X_2	Y
1.0	0.000	95.6
0.5	0.866	77.9
−0.5	0.866	76.2
−1.0	0	54.5
−0.5	−0.866	63.9
0.5	−0.866	79.1
0	0	96.8
0	0	94.8
0	0	94.4

Exercise 5.18 Table 5.6 represents a design matrix and the response for a control composite design.

(i) Estimate the response function and its stationary point.
(ii) Plot contours of equal response, in two dimensions.
(iii) Conduct an ANOVA.

Solution 5.18 Prepare the dataset and build a regression model.

```
df = pd.DataFrame([
    [1, 0, 95.6], [0.5, 0.866, 77.9], [-0.5, 0.866, 76.2],
    [-1, 0, 54.5], [-0.5, -0.866, 63.9], [0.5, -0.866, 79.1],
```

```
    [0, 0, 96.8], [0, 0, 94.8], [0, 0, 94.4],
], columns=['X1', 'X2', 'Y'])

formula = ('Y ~ X1 + X2 + X1*X2 + I(X1**2) + I(X2**2)')
model = smf.ols(formula, data=df).fit()
print(model.summary2())
```

```
                Results: Ordinary least squares
================================================================
Model:              OLS              Adj. R-squared:     0.855
Dependent Variable: Y                AIC:                59.2942
Date:               2023-04-25 11:37 BIC:                60.4776
No. Observations:   9                Log-Likelihood:     -23.647
Df Model:           5                F-statistic:        10.47
Df Residuals:       3                Prob (F-statistic): 0.0408
R-squared:          0.946            Scale:              33.638
----------------------------------------------------------------
               Coef.   Std.Err.    t    P>|t|   [0.025   0.975]
----------------------------------------------------------------
Intercept      95.3333   3.3485 28.4703 0.0001  84.6768 105.9898
X1             16.5167   3.3485  4.9325 0.0160   5.8602  27.1732
X2              3.2044   3.3486  0.9569 0.4092  -7.4524  13.8612
X1:X2          -7.7945   6.6972 -1.1638 0.3286 -29.1081  13.5191
I(X1 ** 2)    -20.2833   5.2945 -3.8310 0.0313 -37.1327  -3.4339
I(X2 ** 2)    -21.3179   5.2948 -4.0262 0.0275 -38.1683  -4.4675
----------------------------------------------------------------
Omnibus:               3.213       Durbin-Watson:          3.565
Prob(Omnibus):         0.201       Jarque-Bera (JB):       0.950
Skew:                  0.006       Prob(JB):               0.622
Kurtosis:              1.408       Condition No.:          4
================================================================
```

The response function is $Y = 95.3+16.5X_1+3.20X_2-20.3X_1^2-21.3X_2^2-7.79X_1X_2$.

We can use *ResponseSurfaceMethod* from the `mistat` package to identify the stationary point.

```
rsm = mistat.ResponseSurfaceMethod(model, ['X1', 'X2'])
stationary = rsm.stationary_point()
stationary
```

```
X1    0.407004
X2    0.000751
dtype: float64
```

The stationary point is at (0.4, 0).

(ii) To plot the response surface, we reuse the function defined in Exercise 5.17; see Fig. 5.7.

```
ax = plotResponseSurface(model)
ax.scatter(*stationary, color='black')
plt.show()
```

(iii) Perform an ANOVA.

```
anova.anova_lm(model)
```

```
             df      sum_sq     mean_sq          F    PR(>F)
X1          1.0  818.400833  818.400833  24.329813  0.015976
X2          1.0   30.802500   30.802500   0.915712  0.409198
```

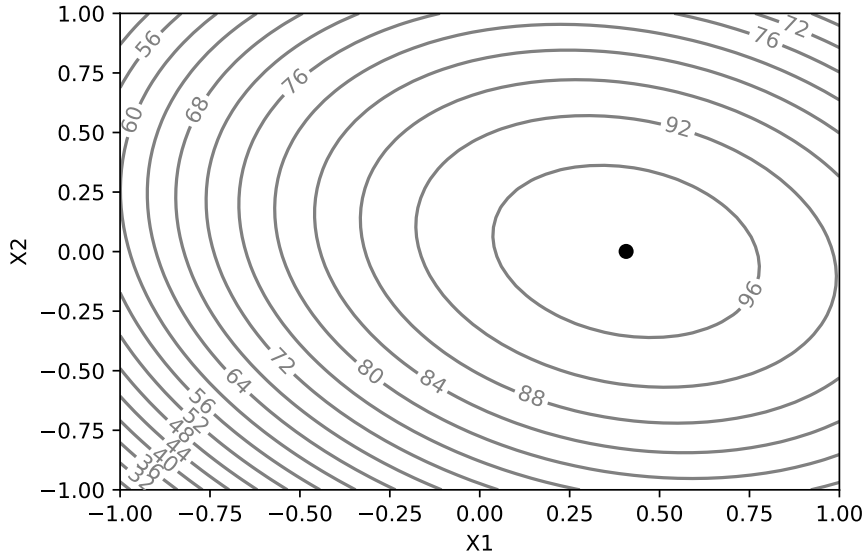


Fig. 5.7: Contour plot of equal responses; the stationary point is shown as a black dot (Exercise 5.18)

X1:X2	1.0	45.562500	45.562500	1.354504	0.328640
I(X1 ** 2)	1.0	320.800500	320.800500	9.536911	0.053790
I(X2 ** 2)	1.0	545.280333	545.280333	16.210355	0.027533
Residual	3.0	100.913333	33.637778	NaN	NaN

The factor X_1 and the squares X_1, X_2 are significant. X_2 and the interaction $X1 : X2$ are not significant.

Chapter 6
Quality by Design

For the most recent version of the solution manual, go to `https://gedeck.github.io/mistat-code-solutions/IndustrialStatistics/`.

Import required modules and define required functions

```
import numpy as np
import pandas as pd
import statsmodels.formula.api as smf
from pyDOE2 import fracfact

import mistat
```

Exercise 6.1 The objective is to find the levels of the factors of the piston, which yield an average cycle time of 0.02 [sec]. Execute a *PistonSimulation*, with sample size $n = 100$:

(i) Determine which treatment combination yields the smallest

$$MSE = (\bar{Y} - 0.45)^2 + S^2.$$

(ii) Determine which treatment combination yields the largest SN ratio,

$$\eta = 10 \log_{10}\left(\frac{Y^2}{S^2} - \frac{1}{100}\right).$$

What is the MSE at this treatment combination?

The five factors that are varied are: piston weight, piston surface area, initial gas volume, spring coefficient, and ambient temperature. The factors atmospheric pressure and filling gas temperature are kept constant at the midrange level.

Solution 6.1 Create a factorial design of the five factors using the highest and lowest levels.

```
from mistat.design import doe
np.random.seed(1)
```

```
# Build design from factors
FacDesign = doe.full_fact({
    'm': [30, 60],
    's': [0.005, 0.02],
    'v0': [0.002, 0.01],
    'k': [1500, 4500],
    't': [290, 296],
})

# Randomize design
FacDesign = FacDesign.sample(frac=1).reset_index(drop=True)
```

Run the Piston simulator using 100 replicates for each factor combination.

```
# Setup and run simulator with 100 replicates
# for each combination of factors
simulator = mistat.PistonSimulator(n_replicate=100, **FacDesign,
                                   p0=100_000, t0=350)
result = simulator.simulate()
```

Group the results by the five factor levels and calculate mean and standard deviation of the cycle time.

```
factors = ['m', 's', 'v0', 'k', 't']
result = result.groupby(factors, as_index=False).agg({'seconds': ['mean', 'std']})
result.columns = [*factors, 'mean', 'std']
```

(i) Add the MSE column and sort by its value to determine the best factor combinations.

```
result['MSE'] = (result['mean'] - 0.02)**2 + result['std']**2
result = result.sort_values('MSE')
best_MSE = result.iloc[0,:]
result.head()
```

	m	s	v0	k	t	mean	std	MSE
26	60	0.020	0.002	4500	290	0.009580	0.002223	0.000114
27	60	0.020	0.002	4500	296	0.009659	0.002783	0.000115
24	60	0.020	0.002	1500	290	0.009201	0.002364	0.000122
25	60	0.020	0.002	1500	296	0.008888	0.002322	0.000129
16	60	0.005	0.002	1500	290	0.023836	0.010829	0.000132

```
best_MSE
```

```
m         60.000000
s          0.020000
v0         0.002000
k       4500.000000
t        290.000000
mean       0.009580
std        0.002223
MSE        0.000114
Name: 26, dtype: float64
```

(ii) Calculate the SN ratio and determine the factor combinations with the largest SN ratio.

```
result['SN'] = 10 * np.log10(result['mean']**2 / result['std']**2 - 1/100)
result = result.sort_values('SN', ascending=False).head()
best_SN = result.iloc[0,:]
result.head()
```

	m	s	v0	k	t	mean	std	MSE	SN
29	60	0.02	0.01	1500	296	0.049291	0.003361	0.000869	23.325647
13	30	0.02	0.01	1500	296	0.042370	0.003010	0.000509	22.970572
30	60	0.02	0.01	4500	290	0.056468	0.004261	0.001348	22.445850
28	60	0.02	0.01	1500	290	0.049085	0.003820	0.000861	22.177197
31	60	0.02	0.01	4500	296	0.057546	0.004480	0.001430	22.173743

```
best_SN
```

```
m         60.000000
s          0.020000
v0         0.010000
k       1500.000000
t        296.000000
mean       0.049291
std        0.003361
MSE        0.000869
SN        23.325647
Name: 29, dtype: float64
```

This treatment combination has an MSE of 0.00087, which is 7.7 times bigger than the minimal MSE. This exercise demonstrates the need for a full analysis of the effects and the dangers of relying on a simplistic observation of the experiment's outcomes.

Exercise 6.2 Run a *PistonSimulation* with sample size of $n = 100$ and generate the sample means and standard deviation of the $2^7 = 128$ treatment combinations of a full factorial experiment, for the effects on the piston cycle time. Perform regression analysis to find which factors have significant effects on the signal-to-noise ratio $SN = \log((\bar{X}/S)^2)$.

Solution 6.2 Create a 2^7 full factorial design and run the piston simulator with 100 replicates for each factor combination. After grouping by factors, determine mean and standard deviation of the cycle time. Finally, calculate the signal noise ratio SN.

```
np.random.seed(1)

# Build design from factors
FacDesign = doe.full_fact({
    'm': [30, 60],
    's': [0.005, 0.02],
    'v0': [0.002, 0.01],
    'k': [1500, 4500],
    't': [290, 296],
    'p0': [90_000, 110_000],
    't0': [340, 360],
})

# Randomize design
FacDesign = FacDesign.sample(frac=1).reset_index(drop=True)

# Setup and run simulator with five replicates
```

```
# for each combination of factors
simulator = mistat.PistonSimulator(n_replicate=100, **FacDesign)
result = simulator.simulate()
factors = ['m', 's', 'v0', 'k', 't', 'p0', 't0']
result = result.groupby(factors, as_index=False).agg({'seconds': ['mean', 'std']})
result.columns = [*factors, 'mean', 'std']
result['SN'] = 10 * np.log10(result['mean']**2 / result['std']**2 - 1/100)
```

The regression analysis of SN on the seven factors (only main effects) results in.

```
model = smf.ols('SN ~ m + s + k + t + v0 + p0 + t0', data=result).fit()
print(model.summary2())
```

```
                 Results: Ordinary least squares
=================================================================
Model:              OLS              Adj. R-squared:     0.899
Dependent Variable: SN               AIC:                522.9904
Date:               2023-04-24 23:25 BIC:                545.8067
No. Observations:   128              Log-Likelihood:     -253.50
Df Model:           7                F-statistic:        163.3
Df Residuals:       120              Prob (F-statistic): 3.28e-58
R-squared:          0.905            Scale:              3.2790
-----------------------------------------------------------------
               Coef.   Std.Err.    t    P>|t|   [0.025    0.975]
-----------------------------------------------------------------
Intercept       5.0686  16.6968  0.3036 0.7620 -27.9899   38.1271
m              -0.0094   0.0107 -0.8845 0.3782  -0.0306    0.0117
s             484.4714  21.3403 22.7021 0.0000 442.2191  526.7238
k               0.0001   0.0001  0.5709 0.5691  -0.0002    0.0003
t               0.0012   0.0534  0.0217 0.9827  -0.1045    0.1068
v0           1001.1879  40.0131 25.0215 0.0000 921.9647 1080.4112
p0             -0.0000   0.0000 -0.2122 0.8323  -0.0000    0.0000
t0             -0.0095   0.0160 -0.5949 0.5530  -0.0412    0.0222
-----------------------------------------------------------------
Omnibus:              10.591       Durbin-Watson:          0.520
Prob(Omnibus):        0.005        Jarque-Bera (JB):       4.418
Skew:                 -0.164       Prob(JB):               0.110
Kurtosis:             2.151        Condition No.:          25136579
=================================================================
* The condition number is large (3e+07). This might indicate
strong multicollinearity or other numerical problems.
```

We see that only factors s and $v0$ (piston surface area, initial gas volume) have a significant effect. The R^2 is only 90.5%. Adding to the regression equation the first order interactions between s and $v0$, we get

```
model = smf.ols('SN ~ s + v0 + s*v0', data=result).fit()
print(model.summary2())
```

```
                 Results: Ordinary least squares
=================================================================
Model:              OLS              Adj. R-squared:     0.964
Dependent Variable: SN               AIC:                388.4606
Date:               2023-04-24 23:25 BIC:                399.8687
No. Observations:   128              Log-Likelihood:     -190.23
Df Model:           3                F-statistic:        1128.
Df Residuals:       124              Prob (F-statistic): 8.90e-90
R-squared:          0.965            Scale:              1.1808
-----------------------------------------------------------------
            Coef.    Std.Err.    t    P>|t|    [0.025    0.975]
-----------------------------------------------------------------
Intercept    5.0013    0.3365 14.8604 0.0000    4.3351    5.6674
```

```
s           203.8500   23.0871  8.8296 0.0000   158.1541   249.5459
v0          416.5600   46.6710  8.9254 0.0000   324.1849   508.9350
s:v0      46770.2389 3201.6068 14.6084 0.0000 40433.3622 53107.1157
------------------------------------------------------------------------
Omnibus:                0.674         Durbin-Watson:           1.053
Prob(Omnibus):          0.714         Jarque-Bera (JB):        0.303
Skew:                  -0.032         Prob(JB):                0.859
Kurtosis:               3.229         Condition No.:           33340
========================================================================
* The condition number is large (3e+04). This might indicate
strong multicollinearity or other numerical problems.
```

We see that the added interaction $s*v0$ is very significant. The regression equation with the interaction terms predicts the SN better, $R^2 = 96.5\%$.

Exercise 6.3 Let (X_1, X_2) have joint distribution with means (ξ_1, ξ_2) and covariance matrix

$$V = \begin{pmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{12} & \sigma_2^2 \end{pmatrix}.$$

Find approximations to the expected values and variances of:

(i) $Y = X_1/X_2$.
(ii) $Y = \log(X_1^2/X_2^2)$.
(iii) $Y = (X_1^2 + X_2^2)^{1/2}$.

Solution 6.3 The approximations to the expected values and variances are as follows:

(i)

$$E\left\{\frac{X_1}{X_2}\right\} \approx \frac{\xi_1}{\xi_2} - \sigma_{12}\frac{1}{\xi_2^2} + \sigma_2^2\frac{\xi_1}{\xi_2^3}$$

$$V\left\{\frac{X_1}{X_2}\right\} \approx \frac{\sigma_1^2}{\xi_2^2} + \sigma_2^2\frac{\xi_1^2}{\xi_2^4} - 2\sigma_{12}\frac{\xi_1}{\xi_2^3}$$

(ii)

$$E\left\{\log\frac{X_1^2}{X_2^2}\right\} \approx \log\left(\frac{\xi_1}{\xi_2}\right)^2 - \frac{\sigma_1^2}{\xi_1^2} + \frac{\sigma_2^2}{\xi_2^2}$$

$$V\left\{\log\left(\frac{X_1}{X_2}\right)^2\right\} \approx \frac{4\sigma_1^2}{\xi_1^2} + \frac{4\sigma_2^2}{\xi_2^2} - 8\frac{\sigma_{12}}{\xi_1\xi_2}.$$

(iii)

$$E\{(X_1^2+X_2^2)^{1/2}\} \approx (\xi_1^2+\xi_2^2)^{1/2} + \frac{\sigma_1^2}{2(\xi_1^2+\xi_2^2)^{1/2}}\left(1-\frac{\xi_1^2}{\xi_1^2+\xi_2^2}\right) + \frac{\sigma_2^2}{2(\xi_1^2+\xi_2^2)^{1/2}}\left(1-\frac{\xi_2^2}{\xi_1^2+\xi_2^2}\right) - \frac{\sigma_{12}\xi_1\xi_2}{(\xi_1^2+\xi_2^2)^{3/2}};$$

$$V\{(X_1^2+X_2^2)^{1/2}\} \approx \frac{\sigma_1^2\xi_1^2}{(\xi_1^2+\xi_2^2)} + \frac{\sigma_2^2\xi_2^2}{(\xi_1^2+\xi_2^2)} + 2\frac{\sigma_{12}\xi_1\xi_2}{(\xi_1^2+\xi_2^2)}.$$

Exercise 6.4 The relationship between the absorption ratio Y of a solid image in a copied paper and the light intensity X is given by the function

$$Y = 0.0782 + \frac{0.90258}{1+0.6969X^{-1.4258}}.$$

Assuming that X has the gamma distribution $G(1, 1.5)$, approximate the expected value and variance of Y.

Solution 6.4 Approximation formulas yield: $E\{Y\} \approx 0.5159$ and $V\{Y\} \approx 0.06762$. Simulation with 5000 runs yields the estimates $E\{Y\} \approx 0.6089$, $V\{Y\} \approx 0.05159$. The first approximation of $E\{Y\}$ is significantly lower than the simulation estimate.

Exercise 6.5 Let $\bar{X}_n$ and S_n^2 be the mean and variance of a random sample of size n from a normal distribution $N(\mu, \sigma)$. We know that $\bar{X}_n$ and S_n^2 are independent, $\bar{X}_n \sim N\left(\mu, \frac{\sigma}{\sqrt{n}}\right)$ and $S_n^2 \sim \frac{\sigma^2}{n-1}\chi^2[n-1]$. Find an approximation to the expected value and variance of $Y = \log\left(\frac{\bar{X}_n^2}{S_n^2}\right)$.

Solution 6.5 We have that $E\{\bar{X}_n\} = \mu$, $V\{\bar{X}_n\} = \frac{\sigma^2}{n}$, $E\{S_n^2\} = \sigma^2$ and $V\{S_n^2\} = \frac{2\sigma^4}{n-1}$. The first and second order partial derivatives of $f(\mu, \sigma^2) = 2\log(\mu) - \log(\sigma^2)$ are

$$\frac{\partial}{\partial\mu}f = \frac{2}{\mu}, \qquad \frac{\partial}{\partial\sigma^2}f = -\frac{1}{\sigma^2},$$
$$\frac{\partial^2}{\partial\mu\partial\sigma^2}f = 0, \qquad \frac{\partial^2}{\partial\mu^2}f = -\frac{2}{\mu^2} \quad \text{and} \quad \frac{\partial^2}{\partial(\sigma^2)^2}f = \frac{1}{\sigma^4}.$$

Thus the approximations to the expected value and variance of $Y = \log\left(\frac{\bar{X}_n^2}{S_n^2}\right)$ are

$$E\left\{\log\left(\frac{\bar{X}_n^2}{S_n^2}\right)\right\} \approx \log\left(\frac{\mu^2}{\sigma^2}\right) - \frac{\sigma^2}{n\mu^2} + \frac{1}{n-1} \quad \text{and}$$
$$V\left\{\log\left(\frac{\bar{X}_n^2}{S_n^2}\right)\right\} \approx \frac{4\sigma^2}{n\mu^2} + \frac{2}{n-1}.$$

Table 6.1: Results of experiment based on L_{18} orthogonal array

	Factors									
Run	1	2	3	4	5	6	7	8	$\bar{X}$	S
1	1	1	1	1	1	1	1	1	2.500	0.0827
2	1	1	2	2	2	2	2	2	2.684	0.1196
3	1	1	3	3	3	3	3	3	2.660	0.1722
4	1	2	1	1	2	2	3	3	1.962	0.1696
5	1	2	2	2	3	3	1	1	1.870	0.1168
6	1	2	3	3	1	1	2	2	2.584	0.1106
7	1	3	1	2	1	3	2	3	2.032	0.0718
8	1	3	2	3	2	1	3	1	3.267	0.2101
9	1	3	3	1	3	2	1	2	2.829	0.1516
10	2	1	1	3	3	2	2	1	2.660	0.1912
11	2	1	2	1	1	3	3	2	3.166	0.0674
12	2	1	3	2	2	1	1	3	3.323	0.1274
13	2	2	1	2	3	1	3	2	2.576	0.0850
14	2	2	2	3	1	2	1	3	2.308	0.0964
15	2	2	3	1	2	3	2	1	2.464	0.0385
16	2	3	1	3	2	3	1	2	2.667	0.0706
17	2	3	2	1	3	1	2	3	3.156	0.1569
18	2	3	3	2	1	2	3	1	3.494	0.0473

Exercise 6.6 An experiment based on an L_{18} orthogonal array involving eight factors gave the results listed in Table 6.1 (see Phadke et al., 1983). Each run had $n = 5$ replications.

Analyze the effects of the factors of the SN ratio $\eta = \log(\bar{X}/S)$.

Solution 6.6 First, create the data frame.

```
df = pd.DataFrame([
  [1, 1, 1, 1, 1, 1, 1, 1, 2.5, 0.0827],
  [1, 1, 2, 2, 2, 2, 2, 2, 2.684, 0.1196],
  [1, 1, 3, 3, 3, 3, 3, 3, 2.66, 0.1722],
  [1, 2, 1, 1, 2, 2, 3, 3, 1.962, 0.1696],
  [1, 2, 2, 2, 3, 3, 1, 1, 1.87, 0.1168],
  [1, 2, 3, 3, 1, 1, 2, 2, 2.584, 0.1106],
  [1, 3, 1, 2, 1, 3, 2, 3, 2.032, 0.0718],
  [1, 3, 2, 3, 2, 1, 3, 1, 3.267, 0.2101],
  [1, 3, 3, 1, 3, 2, 1, 2, 2.829, 0.1516],
  [2, 1, 1, 3, 3, 2, 2, 1, 2.66, 0.1912],
  [2, 1, 2, 1, 1, 3, 3, 2, 3.166, 0.0674],
  [2, 1, 3, 2, 2, 1, 1, 3, 3.323, 0.1274],
  [2, 2, 1, 2, 3, 1, 3, 2, 2.576, 0.085],
  [2, 2, 2, 3, 1, 2, 1, 3, 2.308, 0.0964],
  [2, 2, 3, 1, 2, 3, 2, 1, 2.464, 0.0385],
  [2, 3, 1, 3, 2, 3, 1, 2, 2.667, 0.0706],
  [2, 3, 2, 1, 3, 1, 2, 3, 3.156, 0.1569],
  [2, 3, 3, 2, 1, 2, 3, 1, 3.494, 0.0473],
], columns=['F1', 'F2', 'F3', 'F4', 'F5',
            'F6', 'F7', 'F8', 'Xbar', 'S'])
```

Transform the factors to [-1, 1] for factor 1 and to [-1, 0, 1] for the remaining factors. Calculate the ratio.

```
df['F1'] = (df['F1']-1)*2-1
for column in ['F2', 'F3', 'F4', 'F5', 'F6', 'F7', 'F8']:
    df[column] = df[column] - 2
df['SNR'] = np.log(df['Xbar'] / df['S'])
```

Perform a regression analysis taking only the linear effects into account.

```
model = smf.ols('SNR ~ F1 + F2 + F3 + F4 + F5 + F6 + F7 + F8', data=df).fit()
```

Regression analysis yields the following:

```
print(model.summary2().tables[1].round(5))
```

```
              Coef.  Std.Err.         t    P>|t|   [0.025   0.975]
Intercept  3.22601   0.07518  42.90997  0.00000  3.05594  3.39608
F1         0.26565   0.07518   3.53348  0.00638  0.09558  0.43572
F2         0.07900   0.09208   0.85797  0.41317 -0.12929  0.28729
F3         0.13848   0.09208   1.50392  0.16686 -0.06982  0.34677
F4        -0.14339   0.09208  -1.55733  0.15382 -0.35169  0.06490
F5        -0.31231   0.09208  -3.39185  0.00798 -0.52061 -0.10402
F6         0.12630   0.09208   1.37164  0.20340 -0.08200  0.33459
F7         0.02632   0.09208   0.28587  0.78145 -0.18197  0.23462
F8        -0.17109   0.09208  -1.85810  0.09610 -0.37938  0.03720
```

The linear effects of F1, F5, and F8 are significant. There might be significant interactions or quadratic effects.

Exercise 6.7 Using *PistonSimulation*, perform a full factorial (2^7), a $1/8$ (2^{7-3}), $1/4$ (2^{7-2}), and $1/2$ (2^{7-1}) fractional replications of the cycle time experiment. Estimate the main effects of the seven factors with respect to $SN = \log(\bar{X}/S)$ and compare the results obtained from these experiments. Use $n = 5$ replicates for each combination of factors.

Solution 6.7 Use the generators from Table 5.29 to create the different designs.

```
generators = {
    '2_7': 'A B C D E F G',
    '2_7_1': 'A B C D E F G ABCDEFG',
    '2_7_2': 'A B C D E F G ABCDF ABDEG',
    '2_7_3': 'A B C D E F G ABCE BCDF ACDG',
}
```

Create the designs using the `pyDOE2` function *fracfact*

```
designs = {}
for name, generator in generators.items():
  designs[name] = pd.DataFrame(fracfact(generator), columns=generator.split())

# reduce the fractional factorial designs to a single block
designs['2_7_1'] = designs['2_7_1'].query('ABCDEFG == 1')
designs['2_7_2'] = designs['2_7_2'].query('ABCDF == 1 & ABDEG == 1')
designs['2_7_3'] = designs['2_7_3'].query('ABCE == 1 & BCDF == 1 & ACDG == 1')
```

The function *fracfact* returns a design matrix with values $(-1, 1)$. We need to map these to the actual factor levels.

```
FacLevels = {
    'm': [30, 60],
    's': [0.005, 0.02],
    'v0': [0.002, 0.01],
    'k': [1500, 4500],
    't': [290, 296],
    'p0': [90_000, 110_000],
    't0': [340, 360],
}
FacMap = {'A': 'm', 'B': 's', 'C': 'v0', 'D': 'k',
          'E': 't', 'F': 'p0', 'G': 't0'}

for name, design in designs.items():
    # replace (-1, 1) with factor levels
    facDesign = {}
    for colname in design:
        if colname not in FacMap: # skip generators
            continue
        factor = FacMap[colname]
        levels = FacLevels[factor]
        facDesign[factor] = [levels[max(0, int(v))] for v in design[colname]]
    designs[name] = pd.DataFrame(facDesign)
```

For each design execute the *PistonSimulation* with 5 replicates and determine the SN ratio.

```
results = {}
for name, design in designs.items():
    np.random.seed(1)
    # Setup and run simulator
    simulator = mistat.PistonSimulator(n_replicate=5, **design)
    result = simulator.simulate()
    factors = list(FacLevels)
    result = result.groupby(factors, as_index=False).agg({'seconds': ['mean', 'std']})
    result.columns = [*factors, 'mean', 'std']
    result['SN'] = np.log10(result['mean']**2 / result['std']**2)
    results[name] = result
```

Build linear regression models to estimate the main effects.

```
models = {}
for name, result in results.items():
    model = smf.ols('mean ~ m + s + k + t + v0 + p0 + t0', data=result).fit()
    models[name] = model
```

We can now look at the effect of the design on model performance metrics:

```
for name, model in models.items():
    print(f'{name:10s}: r2={model.rsquared:.3f}, r2_adj={model.rsquared_adj:.3f}')
```

```
2_7       : r2=0.769, r2_adj=0.755
2_7_1     : r2=0.777, r2_adj=0.749
2_7_2     : r2=0.765, r2_adj=0.697
2_7_3     : r2=0.788, r2_adj=0.602
```

While the r^2 metric stays basically the same for all four designs, the adjusted r^2 drops with the size of the design from the full to the 1/8 factorial.

We see a similar effect on the signficance levels and the confidence intervals of the parameter estimates.

```
for name, model in models.items():
    print(name)
    print(model.summary2().tables[1].round(4))
```

```
2_7
             Coef.  Std.Err.        t   P>|t|   [0.025   0.975]
Intercept   0.2183    0.4116   0.5304  0.5968  -0.5967   1.0333
m           0.0005    0.0003   2.0359  0.0440   0.0000   0.0011
s          -6.8075    0.5261 -12.9393  0.0000  -7.8492  -5.7658
k           0.0000    0.0000   1.6946  0.0927  -0.0000   0.0000
t          -0.0004    0.0013  -0.3135  0.7544  -0.0030   0.0022
v0         14.7097    0.9865  14.9116  0.0000  12.7566  16.6628
p0         -0.0000    0.0000  -1.3802  0.1701  -0.0000   0.0000
t0         -0.0000    0.0004  -0.0287  0.9771  -0.0008   0.0008
2_7_1
             Coef.  Std.Err.        t   P>|t|   [0.025   0.975]
Intercept  -0.1876    0.5760  -0.3257  0.7458  -1.3416   0.9663
m           0.0006    0.0004   1.6551  0.1035  -0.0001   0.0013
s          -6.5822    0.7362  -8.9401  0.0000  -8.0571  -5.1073
k           0.0000    0.0000   1.0515  0.2975  -0.0000   0.0000
t           0.0006    0.0018   0.3090  0.7585  -0.0031   0.0043
v0         14.4861    1.3805  10.4936  0.0000  11.7207  17.2515
p0         -0.0000    0.0000  -0.7496  0.4566  -0.0000   0.0000
t0          0.0003    0.0006   0.5019  0.6177  -0.0008   0.0014
2_7_2
             Coef.  Std.Err.       t   P>|t|   [0.025   0.975]
Intercept   0.3676    0.9370  0.3923  0.6983  -1.5663   2.3014
m           0.0005    0.0006  0.7528  0.4589  -0.0008   0.0017
s          -6.8111    1.1976 -5.6874  0.0000  -9.2828  -4.3394
k           0.0000    0.0000  0.4662  0.6453  -0.0000   0.0000
t          -0.0014    0.0030 -0.4713  0.6417  -0.0076   0.0048
v0         14.9546    2.2454  6.6600  0.0000  10.3202  19.5890
p0         -0.0000    0.0000 -0.5629  0.5787  -0.0000   0.0000
t0          0.0004    0.0009  0.4529  0.6547  -0.0014   0.0023
2_7_3
             Coef.  Std.Err.       t   P>|t|   [0.025   0.975]
Intercept   0.1662    1.4378  0.1156  0.9108  -3.1494   3.4818
m           0.0003    0.0009  0.3306  0.7494  -0.0018   0.0024
s          -6.3784    1.8377 -3.4709  0.0084 -10.6161  -2.1407
k           0.0000    0.0000  0.4130  0.6905  -0.0000   0.0000
t          -0.0005    0.0046 -0.1158  0.9107  -0.0111   0.0101
v0         14.3326    3.4457  4.1596  0.0032   6.3869  22.2784
p0         -0.0000    0.0000 -0.1167  0.9100  -0.0000   0.0000
t0          0.0001    0.0014  0.1067  0.9177  -0.0030   0.0033
/usr/local/lib/python3.9/site-packages/scipy/stats/_stats_py.py:1477:
UserWarning: kurtosistest only valid for n>=20 ... continuing anyway,
n=16
  warnings.warn("kurtosistest only valid for n>=20 ... continuing "
```

The p-values and the width of the confidence intervals increase with decreasing size of the design.

Exercise 6.8 To see the effect of the variances of the random variables on the expected response, in non-linear cases, execute *PistonSimulation*, with $n = 20$ replicates, and compare the output means to the values in Exercise 6.7.

Solution 6.8 We execute *PistonSimulation* setting `n_replicate=20`

```
results = {}
for name, design in designs.items():
    np.random.seed(1)
    # Setup and run simulator
```

```
    simulator = mistat.PistonSimulator(n_replicate=20, **design)
    result = simulator.simulate()
    factors = list(FacLevels)
    result = result.groupby(factors, as_index=False).agg({'seconds': ['mean', 'std']})
    result.columns = [*factors, 'mean', 'std']
    result['SN'] = np.log10(result['mean']**2 / result['std']**2)
    results[name] = result

models = {}
for name, result in results.items():
    model = smf.ols('mean ~ m + s + k + t + v0 + p0 + t0', data=result).fit()
    models[name] = model
```

The r^2 and adjusted r^2 values follow a similar pattern as seen for 5 replicates. There are only small differences between the two experiments and there is no consistent pattern of change between 5 and 20 replicates.

```
for name, model in models.items():
    print(f'{name:10s}: r2={model.rsquared:.3f}, r2_adj={model.rsquared_adj:.3f}')
```

```
2_7       : r2=0.779, r2_adj=0.766
2_7_1     : r2=0.783, r2_adj=0.756
2_7_2     : r2=0.763, r2_adj=0.694
2_7_3     : r2=0.782, r2_adj=0.591
```

The same can be said for significance level and confidence intervals of the coefficients.

```
for name, model in models.items():
    print(name)
    print(model.summary2().tables[1].round(4))
```

```
2_7
             Coef.  Std.Err.        t   P>|t|   [0.025   0.975]
Intercept   0.1066    0.3973   0.2683  0.7889  -0.6801   0.8933
m           0.0005    0.0003   1.9699  0.0512  -0.0000   0.0010
s          -6.6914    0.5078 -13.1767  0.0000  -7.6969  -5.6860
k           0.0000    0.0000   1.7975  0.0748  -0.0000   0.0000
t          -0.0001    0.0013  -0.1135  0.9098  -0.0027   0.0024
v0         14.7902    0.9522  15.5333  0.0000  12.9050  16.6755
p0         -0.0000    0.0000  -0.8957  0.3722  -0.0000   0.0000
t0          0.0000    0.0004   0.0556  0.9558  -0.0007   0.0008
2_7_1
             Coef.  Std.Err.        t   P>|t|   [0.025   0.975]
Intercept   0.0030    0.5779   0.0052  0.9959  -1.1546   1.1606
m           0.0005    0.0004   1.4791  0.1447  -0.0002   0.0013
s          -6.7291    0.7386  -9.1106  0.0000  -8.2087  -5.2495
k           0.0000    0.0000   1.0502  0.2981  -0.0000   0.0000
t           0.0002    0.0018   0.1016  0.9194  -0.0035   0.0039
v0         14.8995    1.3849  10.7588  0.0000  12.1253  17.6737
p0         -0.0000    0.0000  -0.5409  0.5907  -0.0000   0.0000
t0          0.0000    0.0006   0.0506  0.9598  -0.0011   0.0011
2_7_2
             Coef.  Std.Err.       t   P>|t|   [0.025   0.975]
Intercept   0.1544    0.9575  0.1613  0.8732  -1.8217   2.1306
m           0.0005    0.0006  0.8712  0.3923  -0.0007   0.0018
s          -6.9223    1.2238 -5.6566  0.0000  -9.4480  -4.3966
k           0.0000    0.0000  0.7248  0.4756  -0.0000   0.0000
t          -0.0002    0.0031 -0.0812  0.9360  -0.0066   0.0061
v0         15.1003    2.2946  6.5809  0.0000  10.3646  19.8361
p0         -0.0000    0.0000 -0.9230  0.3652  -0.0000   0.0000
t0          0.0001    0.0009  0.1316  0.8964  -0.0018   0.0020
```

```
2_7_3
             Coef.  Std.Err.       t   P>|t|   [0.025   0.975]
Intercept   0.6279    1.6093  0.3902  0.7066  -3.0832   4.3391
m           0.0006    0.0010  0.5371  0.6058  -0.0018   0.0029
s          -7.1112    2.0569 -3.4572  0.0086 -11.8544  -2.3679
k           0.0000    0.0000  0.3975  0.7014  -0.0000   0.0000
t          -0.0022    0.0051 -0.4281  0.6798  -0.0141   0.0097
v0         15.4170    3.8567  3.9975  0.0040   6.5235  24.3105
p0         -0.0000    0.0000 -0.2387  0.8173  -0.0000   0.0000
t0          0.0003    0.0015  0.1761  0.8646  -0.0033   0.0038
/usr/local/lib/python3.9/site-packages/scipy/stats/_stats_py.py:1477:
UserWarning: kurtosistest only valid for n>=20 ... continuing anyway,
n=16
  warnings.warn("kurtosistest only valid for n>=20 ... continuing "
```

Exercise 6.9 Run *PowerCircuitSimulation* with 1% and 2% tolerances, and compare the results to those of Table 6.13.

Solution 6.9 Repeat the simulation from Example 6.5 with tolerances of 1% and 2%.

```
tolerances = [f'tl{c}' for c in 'ABCDEFGHIJKLM']
factors = {tl: [1, 2] for tl in tolerances}
Design = doe.frac_fact_res(factors, 4)

# Randomize and create replicates
nrepeat = 100
Design = Design.sample(frac=1).reset_index(drop=True)
Design = Design.loc[Design.index.repeat(nrepeat)].reset_index(drop=True)

# Run simulation
simulator = mistat.PowerCircuitSimulation(**{k: list(Design[k]) for k in Design})
result = simulator.simulate()
result = mistat.simulationGroup(result, nrepeat)

# Combine results with the Design matrix
Design['response'] = result['volts']
Design['group'] = result['group']

# calculate mean, standard deviation, and MSE
def groupAggregation(g):
    return {
        'mean': g['response'].mean(),
        'std': g['response'].std(),
        'MSE': g['response'].var(ddof=0),
    }
results = pd.DataFrame(list(Design.groupby('group').apply(groupAggregation)))
results
```

```
          mean       std       MSE
0   230.156267  0.966150  0.924111
1   230.092224  1.254120  1.557088
2   229.957027  1.013553  1.017016
3   230.169616  1.037447  1.065534
4   229.898480  1.041472  1.073818
5   230.005033  1.280572  1.623467
6   230.075389  1.280411  1.623059
7   230.089210  0.725727  0.521414
8   229.990264  0.951721  0.896714
9   230.126760  1.346345  1.794518
10  230.107421  1.125018  1.253009
11  229.861824  1.396280  1.930103
12  229.950664  1.162393  1.337646
```

```
13  230.016845  1.321987  1.730174
14  230.015692  1.188564  1.398558
15  230.000549  1.435997  2.041467
16  230.147138  1.060209  1.112802
17  230.006373  1.351617  1.808599
18  229.966871  1.225543  1.486937
19  230.257130  1.343611  1.787238
20  229.972538  0.799752  0.633208
21  230.113127  1.075547  1.145234
22  230.089659  0.989811  0.969929
23  230.165592  1.275611  1.610911
24  229.945301  0.944878  0.883867
25  230.117853  1.106905  1.212985
26  229.924627  1.236313  1.513184
27  230.131634  0.935886  0.867123
28  229.963843  1.176523  1.370365
29  229.938556  1.063175  1.119038
30  230.047874  0.998668  0.987364
31  230.011064  1.150356  1.310086
```

Comparing this table to Table 6.13 in the text, we see that by reducing the tolerances to 1% and 2%, the MSE is reduced by a factor of 25. This, however, increases the cost of the product.

Chapter 7
Computer Experiments

For the most recent version of the solution manual, go to `https://gedeck.github.io/mistat-code-solutions/IndustrialStatistics/`.

Import required modules and define required functions

```
import random
import numpy as np
import pandas as pd
from scipy import stats
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error
import pylibkriging as lk
import mistat
```

Exercise 7.1 The birthday problem states that if there are more than 22 people at a birthday party, the probability that at least two people have the same birthday is greater than 0.5. Write a Python program to simulate this problem. Show that if there are more than 22 people in the party, the probability is greater than 1/2 that at least 2 will have birthdays on the same day.

Solution 7.1 In Python:

```
nrepeat = 10000
days = 365
for size in (22, 23): # range(1, 365+1):
  same_birthday = 0
  for _ in range(nrepeat):
    birthdays = stats.randint.rvs(1, 365+1, size=size)
    if len(birthdays) != len(set(birthdays)):
        same_birthday += 1
  print(f'size of party: {size}, ',
        f'p(same birthday) {same_birthday / nrepeat:.2f}')
```

```
size of party: 22,  p(same birthday) 0.47
size of party: 23,  p(same birthday) 0.50
```

Exercise 7.2 The Deming funnel experiment was designed to show that an inappropriate reaction to common cause variation will make matters worse. Common cause and special causes affecting processes over time have been discussed in Chaps. 2 to 4.

In the actual demonstration, a funnel is placed above a circular target. The objective is to drop a marble through the funnel as close to the target as possible. A pen or pencil is used to mark the spot where the marble actually hits. Usually, 20 or more drops are performed in order to establish the pattern and extent of variation about the target. The funnel represents common causes affecting a system. Despite the operator's best efforts, the marble will not land exactly on the target each time. The operator can react to this variability in one of four ways:

1. Do not move the funnel.
2. Measure the distance the hit is from the target and move the funnel an equal distance, but in the opposite direction (error relative to the previous position).
3. Measure the distance the hit is from the target and move the funnel this distance in the opposite direction, starting at the target (error relative to the target).
4. Move the funnel to be exactly over the location of the last hit.

Use Python to compare these four strategies using simulation data.

Solution 7.2 We first define a function that simulates the experiment. The marbel will drop close to the drop position with a deviation that has a bivariate normal distribution. As the scale is not defined, we set the standard deviation to 1. The target is positioned at (0, 0).

```
def funnel_drop(position):
  ''' based on funnel position, returns marble drop position'''
  x_new = position[0] + stats.norm().rvs()
  y_new = position[1] + stats.norm().rvs()
  return np.array([x_new, y_new])

def result(strategy, dropped):
  ''' returns result information '''
  return {
    'strategy': strategy,
    'x': dropped[0],
    'y': dropped[1],
    'distance': np.sqrt(dropped[0]**2 + dropped[1]**2),
  }
```

Next we simulate the funnel drop experiment for different strategies. See Fig. 7.1 for a visualization of the simulation results.

```
np.random.seed(1)
results = []
nrepeat = 100

# strategy 1
# funnel fixed
position = (0, 0)
for _ in range(nrepeat):
  dropped = funnel_drop(position)
  results.append(result(1, dropped))
```

```
# strategy 2
# position funnel to compensate for error relative to funnel position
position = np.array([0, 0])
for _ in range(nrepeat):
  dropped = funnel_drop(position)
  results.append(result(2, dropped))
  position = position - dropped

# strategy 3
# position funnel to compensate for error relative to target
position = np.array([0, 0])
for _ in range(nrepeat):
  dropped = funnel_drop(position)
  results.append(result(3, dropped))
  position = - dropped

# strategy 4
# position funnel to compensate for error relative to target
position = np.array([0, 0])
for _ in range(nrepeat):
  dropped = funnel_drop(position)
  results.append(result(4, dropped))
  position = dropped

results = pd.DataFrame(results)
g = sns.FacetGrid(results, col='strategy', col_wrap=2)
g.map(sns.scatterplot, 'x', 'y')
plt.show()
sns.boxplot(x='distance', y='strategy', data=results, orient='h')
#g = sns.FacetGrid(results, col='strategy')
#g.map(sns.boxplot, 'distance', order='strategy', orient='v')
plt.show()
```

The result of the simulation shows that strategy 1, no interference, leads to results closest to the target. Strategy 2 still creates results close to the target, but with a larger deviation. Strategies 3 and 4 lead to results far away from the target.

Exercise 7.3 Design a 50 runs experimental array for running the piston simulator using different options available in the `mistat` package:

- Latin hypercube (simple): *mistat.design.doe.lhs*
- Latin hypercube (space filling): *mistat.design.doe.space_filling_lhs*
- Random k-means cluster: *mistat.design.doe.random_k_means*
- Maximin reconstruction: *mistat.design.doe.maximin*
- Halton sequence-based: *mistat.design.doe.halton*
- Uniform random matrix: *mistat.design.doe.uniform_random*

Compare the results.

Solution 7.3 We first create a series of design using the different methods

```
from mistat.design import doe

np.random.seed(1) # set random seed for reproducibility
Factors = {
    'm': [30, 60],
    's': [0.005, 0.02],
    'v0': [0.002, 0.01],
    'k': [1_000, 5_000],
```

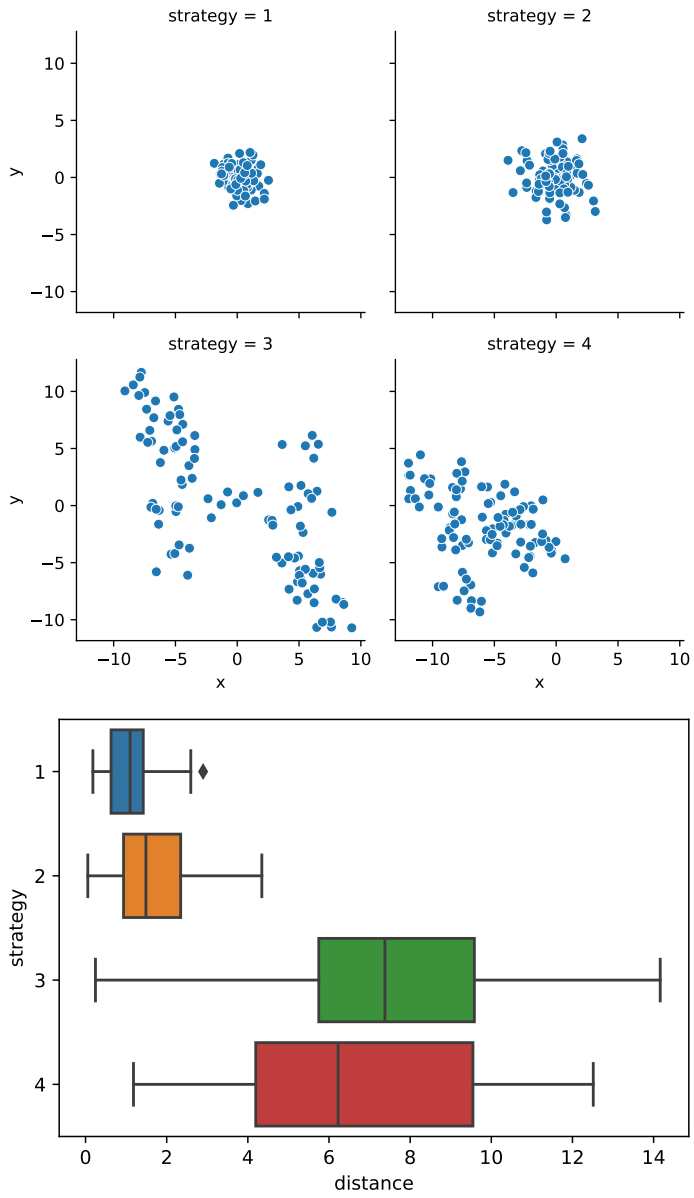


Fig. 7.1: Result of funnel drop experiment for the different strategies

```
    'p0': [90_000, 110_000],
    't': [290, 296],
    't0': [340, 360],
}
Designs = {
    'Latin hypercube': doe.lhs(Factors, num_samples=50),
    'Latin hypercube (space filling)': doe.space_filling_lhs(Factors, num_samples=50),
    'Random k-means cluster': doe.random_k_means(Factors, num_samples=50),
    'Maximin reconstruction': doe.maximin(Factors, num_samples=50),
    'Halton sequence based': doe.halton(Factors, num_samples=50),
    'Uniform random matrix': doe.uniform_random(Factors, num_samples=50),
}
```

Next we can visualize each design using the pandas scatterplot matrix method. We hide axis ticks for clarity.

```
for method, design in Designs.items():
    sm = pd.plotting.scatter_matrix(design, figsize=[4, 4])
    # hide all axis labels in visualization
    for subaxis in sm:
        for ax in subaxis:
            ax.xaxis.set_ticks([])
            ax.yaxis.set_ticks([])
            ax.set_ylabel("")
    plt.suptitle(method)
    plt.show()
```

The result is shown in Fig. 7.2. The methods lead to cleary distinct distributions of the design. The distributions for both Latin hypercube designs, the maximin reconstruction design, and the uniform random matrix design lead to random coverage of the design space. Looking a the distribution for each factor, both Latin hypercube designs show the most uniform distributions, while the maximin reconstruction and uniform random matrix methods lead to a design where factors are not uniformly explored.

The random k-means cluster and the Halton sequence based designs are clearly distinct. The random k-means cluster method leads to a design that favors the extreme values for each factor. The Halton sequence based design shows patterns in the pair-wise scatterplots.

Exercise 7.4 Fit a Gaussian process model to data generated by the six designs listed in Exercise 7.3 and compare the MSE of the model fits.

Solution 7.4 We run a 5-fold cross-validation using the *Kriging* model from the `pylibkriging` package.

```
random.seed(1)
np.random.seed(1)

outcome = 'seconds'
predictors = ['m', 's', 'v0', 'k', 'p0', 't', 't0']

performance = []
for method, Design in Designs.items():
    # randomize the design and run the piston simulator
    DesignRandomized = Design.sample(frac=1, random_state=1).reset_index(drop=True)
    simulator = mistat.PistonSimulator(parameter=DesignRandomized)
```

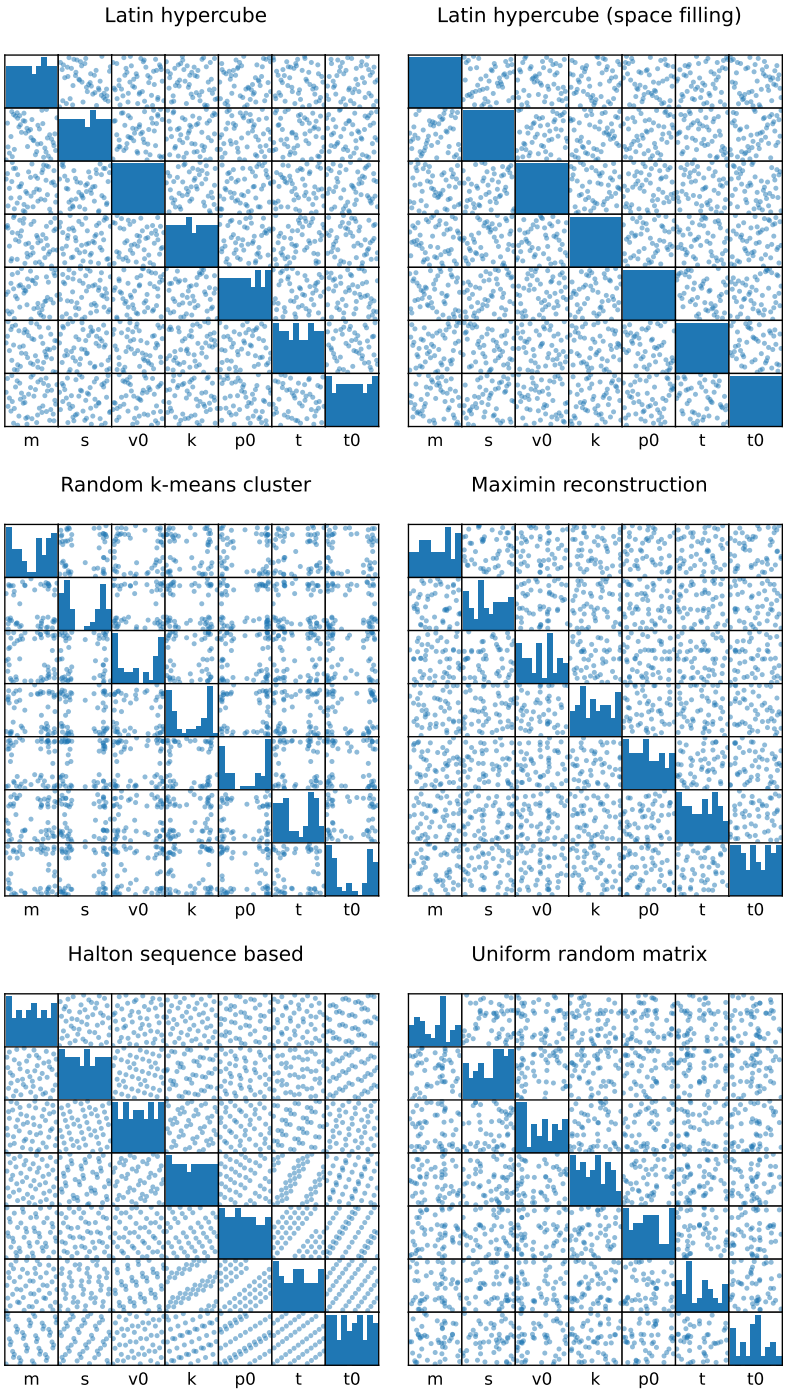


Fig. 7.2: Visualizations of different designs

```
    result = simulator.simulate()

    # convert to X and y matrics to use with the
    X = result[predictors].values
    y = result[outcome].values

    # run 5-fold cross-validation to determine MSE values
    # we collect the out-of-fold predictions to determine
    # the MSE
    predicted = []
    actual = []
    for train_idx, test_idx in KFold(n_splits=5).split(result):
        model = lk.Kriging(y[train_idx], X[train_idx], 'gauss')

        # predict using the test set
        ypred = model.predict(X[test_idx], True, False, False)[0]
        # keep prediction results for performance metric calculation
        actual.extend(y[test_idx])
        predicted.extend(ypred)

    mse = mean_squared_error(actual, predicted)
    performance.append({
        'design': method,
        'mse': mse,
        'rmse': np.sqrt(mse),
    })
```

The performance results are:

design	mse	rmse
Latin hypercube	0.00189	0.04348
Latin hypercube (space filling)	0.00036	0.01898
Random k-means cluster	0.00043	0.02074
Maximin reconstruction	0.00042	0.02061
Halton sequence based	0.00121	0.03482
Uniform random matrix	0.00009	0.00931

In all cases, we observe good predictive performance. Due to randomness in the calculations, the relative order can be different between runs.

Exercise 7.5 Using a uniform random design, generate a stochastic emulator for the piston simulator in order to get 0.02 seconds cycle time with minimal variability.

Solution 7.5 We first generate the uniform random design

```
np.random.seed(1)

num_samples = 1000
Design = doe.uniform_random(Factors, num_samples=num_samples)
```

Run the piston simulator using replicates of each parameter set. As the next exercise requires the same process, we wrap the required steps into a custom function

```
def evaluateDesign(Design, nrepeat=20):
  #Design = Design.loc[np.repeat(Design.index.values, nrepeat), :]
  #settings = {c: list(Design[c]) for c in Design.columns}
  #simulator = mistat.PistonSimulator(seed=1, **settings)
  simulator = mistat.PistonSimulator(parameter=Design, seed=1, n_replicate=nrepeat)
```

```
  result = simulator.simulate()
  result = mistat.simulationGroup(result, nrepeat)

  # next we aggregate the replicates and determine mean and
  # standard deviation of each group.
  options = {p: 'mean' for p in ['m', 's', 'v0', 'k', 'p0', 't', 't0']}
  options['seconds'] = ['mean', 'std']
  result = result.groupby('group').aggregate(options)
  # convert multi-index to single index
  result.columns = [' '.join(col) if col[0] == 'seconds' else col[0]
                    for col in result.columns]
  return result

result = evaluateDesign(Design, nrepeat=30)
```

In order to find the parameter set that gives a cycle time around 0.02 with minimal variability, we filter the results by the mean values and sort by the standard deviation.

```
# determine rows with seconds around 0.02 and
# identify row with smallest standard deviation
rows = [0.0195 < v < 0.0205 for v in result['seconds mean'].values]
target = result.loc[rows, :]
target = target.sort_values('seconds std')
target.head(3)
```

```
               m         s        v0            k             p0
t  \
group
871    59.694778  0.018177  0.004087  1152.373331  109480.515301
293.342719
331    46.278492  0.018261  0.004200  1432.269941   93825.441035
292.322277
947    37.548930  0.012126  0.003148  1280.072660  105859.307614
291.076845

               t0  seconds mean  seconds std
group
871    350.794200      0.020356     0.002469
331    357.081196      0.020009     0.002751
947    342.524145      0.019582     0.003092
```

We can see that all three possible settings lead to a cycle time close to 0.02 seconds with low variability. The visualization of the results in Fig. 7.3 shows that the settings with a cycle time around 0.02 can be achieved with different values of the piston surface area s and the initial gas volume $v0$. The two variables show correlation. Only a small number of the design parameters match our criteria. It would be useful to repeat the similuation with a larger number of samples.

```
ax = target.plot.scatter(x='v0', y='s')
target.head(3).plot.scatter(x='v0', y='s', ax=ax, color='red')
ax = target.plot.scatter(x='seconds mean', y='seconds std')
target.head(3).plot.scatter(x='seconds mean', y='seconds std', ax=ax, color='red')
plt.show()
```

Exercise 7.6 Using a Latin hypercube design, generate a stochastic emulator for the piston simulator in order to achieve 0.02 seconds cycle time with minimal variability. Compare your results to what you got in Exercise 7.5.

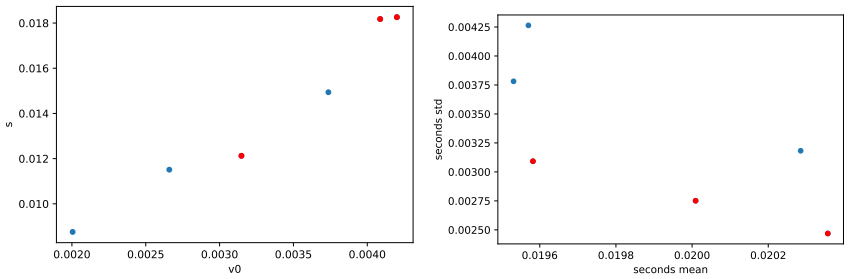


Fig. 7.3: Visualization of results from stochastic emulator based on random uniform design.

Solution 7.6 We can make use of the function defined in the previous exercise to evaluate a Latin hyper cube design

```
np.random.seed(1)

num_samples = 1000
Design = doe.lhs(Factors, num_samples=num_samples, random_state=1)
result = evaluateDesign(Design, nrepeat=30)

# determine rows with seconds around 0.02 and
# identify row with smallest standard deviation
rows = [0.0195 < v < 0.0205 for v in result['seconds mean'].values]
target = result.loc[rows, :]
target = target.sort_values('seconds std')
target.head(3)
```

```
                m         s        v0            k            p0
t  \
group
368    33.166208  0.016760  0.003876  2998.808445  96800.530567
291.237264
940    39.401808  0.019256  0.004388  3024.658005  97630.295509
295.393884
150    33.434638  0.019884  0.004685  1401.296554  92707.973181
292.463943

                t0  seconds mean  seconds std
group
368    341.684282      0.020127     0.001956
940    351.093709      0.020001     0.002313
150    353.536655      0.020030     0.002467
```

The results derived using this design are similar to what was obtained in Exercise 7.5. We can see more points matching our criteria. This could be due to a better coverage of the design space. However, as already mentioned in Exercise 7.5, we could improve the analysis by increasing the number of design points.

```
ax = target.plot.scatter(x='v0', y='s')
target.head(3).plot.scatter(x='v0', y='s', ax=ax, color='red')
ax = target.plot.scatter(x='seconds mean', y='seconds std')
target.head(3).plot.scatter(x='seconds mean', y='seconds std', ax=ax, color='red')
plt.show()
```

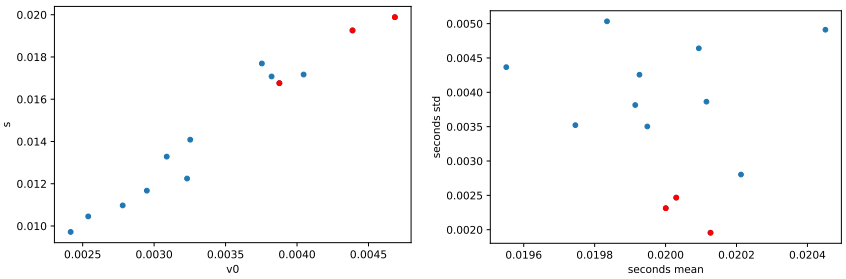


Fig. 7.4: Visualization of results from stochastic emulator based on a Latin hypersquare design

Chapter 8
Cybermanufacturing and digital twins

For the most recent version of the solution manual, go to `https://gedeck.github.io/mistat-code-solutions/IndustrialStatistics/`.

Import required modules and define required functions

```
import numpy as np
import pandas as pd
from scipy import stats
import statsmodels.formula.api as smf
import lifelines
import pingouin as pg
import seaborn as sns
import matplotlib.pyplot as plt
import mistat
from statsmodels.tsa.api import SARIMAX
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import cross_val_score
```

Exercise 8.1 The PENSIM simulation software modelling penicillin production (Birol et al., 2002) is a fed-batch fermentor.[1] It simulates a fed-batch penicillin production process and includes variables such as pH, temperature, aeration rate, agitation power, feed flow rate of the substrate and a Raman probe.

The **PENSIM_100** dataset consists of 100 observations derived from the simulator. These are observational data collected under the same process set up. Variability in responses is induced by the varying process variables

A. Process set up:

1. `S0`: initial sugar concentration (15 g/L)
2. `X0`: initial biomass concentration (0.1 g/L)
3. `pH`: pH set point (5)
4. `T`: temperature set point (298 °K)
5. `air`: aeration (8.6 L/min)
6. `stirring`: agitation rate (29.9 W)

[1] `http://www.industrialpenicillinsimulation.com`.

7. `time`: culture time (350 h)
8. `feed`: sugar feed rate (0.0426 L/h)

B. Process outputs:

1. `P`: Final penicillin concentration
2. `X`: Final biomass concentration

C. Process variables:

1. `Fg`: aeration rate
2. `RPM`: agitation rate
3. `Fs`: subst. feed
4. `Ts`: subst. temp.
5. `S`: substrate
6. `DO`: dissolved oxygen
7. `Uvis`: viscosity
8. `CO2`: off-gas CO2
9. `Hi`: heat inflow
10. `Ti`: temperature inflow
11. `Ho`: heat outflow
12. `Fw`: water for injection

Predict the process outputs P and X from the 12 process variables using two different models. Compare and contrast the models.

Some options are multivariate least square regressions, regression trees, random forests and neural networks, Bayesian networks (Chaps. 4, 7, and 8 in Kenett et al. 2022b), response surfaces (Chap. 5) and Kriging (Gaussian) models (Chap. 7).

Solution 8.1 We train and evaluate linear regression and random forest regression models.

Load and preprocess the data in Python.

```
pensim_100 = mistat.load_data('PENSIM_100')
predictors = ['Fg', 'RPM', 'Fs', 'Ts', 'S', 'DO', 'Uvis',
              'CO2', 'Hi', 'Ti', 'Ho', 'Fw']
outcome_X = 'X'
outcome_P = 'P'
```

```
sns.heatmap(pensim_100.corr(), vmin=-1, vmax=1,
                cmap=sns.diverging_palette(20, 220, as_cmap=True))
```

```
<AxesSubplot: >
```

```
fig, axes = plt.subplots(ncols=2, figsize=[6, 3])
pensim_100['X'].plot.density(bw_method=0.01, ax=axes[0])
axes[0].set_xlabel('X')
pensim_100['P'].plot.density(bw_method=0.01, ax=axes[1])
axes[1].set_xlabel('P')
plt.tight_layout()
```

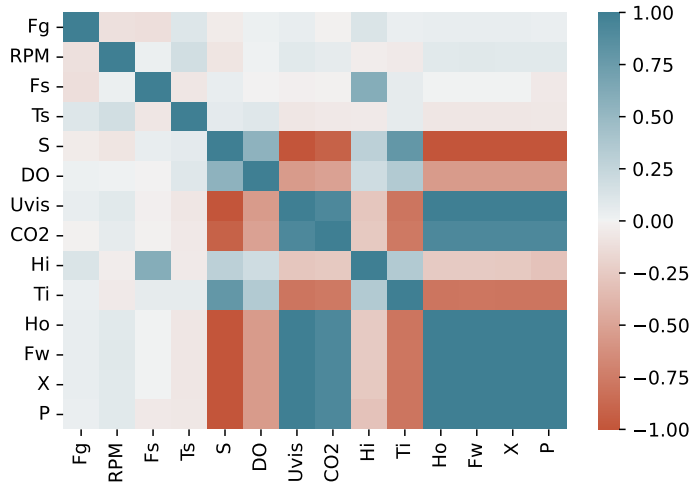


Fig. 8.1: Heatmap visualization of correlation matrix of the **PENSIM_100.csv** dataset

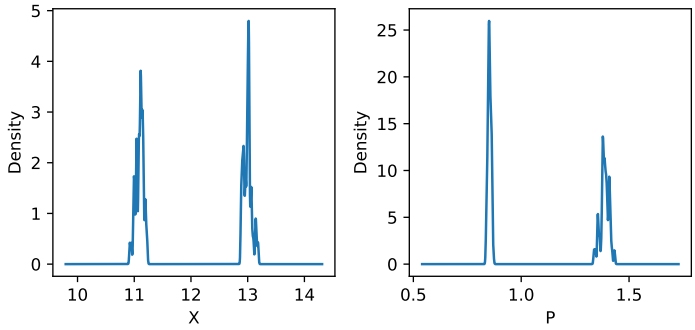


Fig. 8.2: Density plot visualization of the distribution of outcome variables of the **PENSIM_100.csv** dataset

Fig. 8.1 visualizes the correlation matrix of the dataset. Most predictors are highly correlated to both `X` and `P`. Exceptions are `Fg`, `RPM`, `Fs`, `Ts`, and `Hi`. Closer inspection however reveals that this correlation is due to the bimodal nature of several of the descriptors (see Fig. 8.2). Due to this high correlation it is better to use MAE or RMSE to evaluate the models instead of the correlation coefficient.

In the following, we evaluate linear regression and random forest regression models to predict `X` using 5-fold cross-validation with MAE as the score.

```
X = pensim_100[predictors]
y = pensim_100[outcome_X]
```

```
model = LinearRegression()
scores = cross_val_score(model, X, y, cv=5, scoring='neg_mean_absolute_error')
print(f'Linear regression : {np.mean(-scores):.5f}')

model = RandomForestRegressor()
scores = cross_val_score(model, X, y, cv=5, scoring='neg_mean_absolute_error')
print(f'Random forest regression : {np.mean(-scores):.5f}')
```

```
Linear regression : 0.00107
Random forest regression : 0.01228
```

The cross-validation result shows that the linear regression model performs drastically better than the random forest model.

We repeat the same for P.

```
X = pensim_100[predictors]
y = pensim_100[outcome_P]

model = LinearRegression()
scores = cross_val_score(model, X, y, cv=5, scoring='neg_mean_absolute_error')
print(f'Linear regression : {np.mean(-scores):.5f}')

model = RandomForestRegressor()
scores = cross_val_score(model, X, y, cv=5, scoring='neg_mean_absolute_error')
print(f'Random forest regression : {np.mean(-scores):.5f}')
```

```
Linear regression : 0.00716
Random forest regression : 0.00956
```

In this case, both model have similar performance metrics with the linear regression model being slightly better.

Exercise 8.2 The PENSIM simulator introduced in Exercise 8.1 has been used to design and analyze a central composite design experiment (see Sect. 5.9). The dataset is available as **PENSIM_CCD.csv**.

1. Evaluate the experimental design set up (see Sect. 5.10)
2. Fit a second order response surface model to both X and P (see Sect. 5.9)
3. Compose a qualitative description of the models

Solution 8.2 Load the **PENSIM_CDD.csv** dataset.

```
data = mistat.load_data('PENSIM_CCD.csv')
predictors = ['S0', 'pH', 'time', 'feed']
```

(1) Using the code from Sect. 5.10, we create the correlation plot and the fraction of design space plot shown in Fig. 8.3. The correlation plot shows that the design allows estimating all main effects and two-way interactions.

```
def plotCorrelation(design, mod=0, ax=None):
    mm = mistat.getModelMatrix(design, mod=mod)
    mm = mm.drop(columns='Intercept')
    corr = mm.corr().abs()
    if ax is None:
        fig, ax = plt.subplots()
        fig.set_size_inches(11, 7)
```

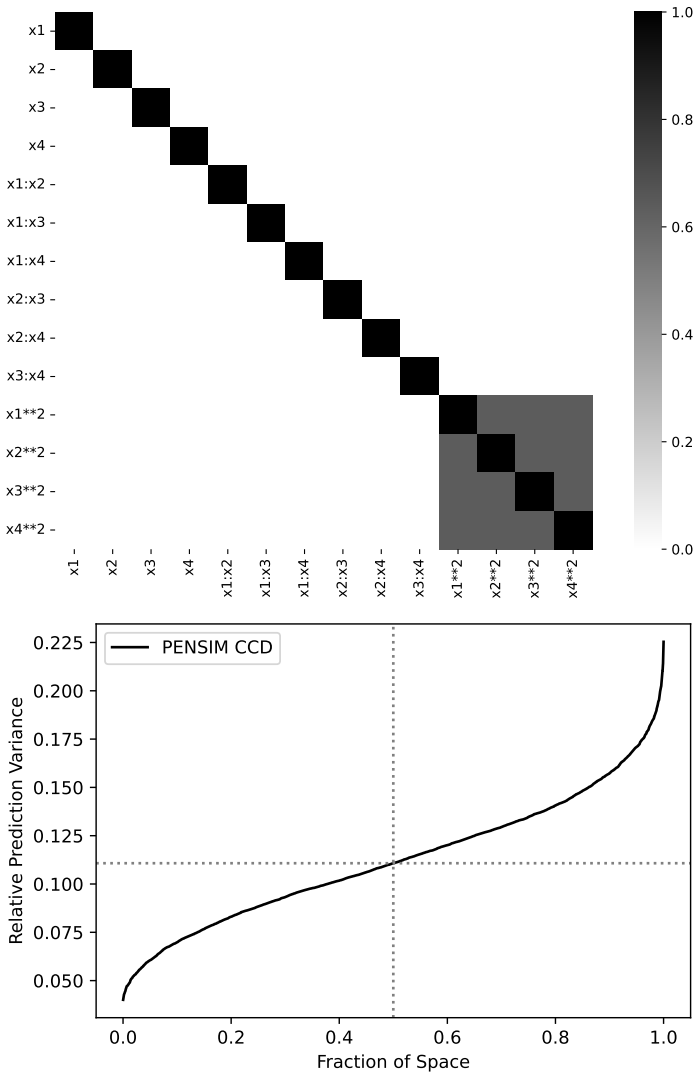


Fig. 8.3: Graphical evaluation of the experimental design setup in dataset **PENSIM_CDD.csv**

```
    sns.heatmap(corr, cmap='binary', ax=ax, square=True)
    return ax

plotCorrelation(data[predictors], mod=2)

_ = mistat.FDS_Plot(data[predictors], label='PENSIM CCD')
```

(2) Build a linear regression model to predict `X` with all main effects, two-way interactions and quadratic terms.

```
formula = ('X ~ (S0 + pH + time + feed)**2 + ' +
           'I(S0**2) + I(pH**2) + I(time**2) + I(feed**2)')
model = smf.ols(formula, data=data).fit()
print(model.summary2().tables[1].round(4))
```

	Coef.	Std.Err.	t	P>|t|	[0.025	0.975]
Intercept	-7.0367	7.4674	-0.9423	0.3663	-23.4724	9.3989
S0	0.2829	0.1164	2.4299	0.0334	0.0267	0.5391
pH	-0.2835	3.4977	-0.0810	0.9369	-7.9818	7.4149
time	0.0496	0.0242	2.0450	0.0655	-0.0038	0.1029
feed	213.1175	79.3833	2.6847	0.0212	38.3959	387.8390
S0:pH	-0.0338	0.0153	-2.2018	0.0499	-0.0676	-0.0000
S0:time	-0.0004	0.0002	-2.3243	0.0403	-0.0007	-0.0000
S0:feed	0.6962	0.7674	0.9073	0.3837	-0.9928	2.3853
pH:time	-0.0057	0.0015	-3.6843	0.0036	-0.0090	-0.0023
pH:feed	6.9927	7.6741	0.9112	0.3817	-9.8978	23.8833
time:feed	0.2036	0.0767	2.6532	0.0225	0.0347	0.3725
I(S0 ** 2)	-0.0000	0.0038	-0.0042	0.9967	-0.0085	0.0084
I(pH ** 2)	0.2168	0.3836	0.5651	0.5833	-0.6276	1.0612
I(time ** 2)	-0.0000	0.0000	-0.9776	0.3493	-0.0001	0.0000
I(feed ** 2)	-715.7954	959.0771	-0.7463	0.4711	-2826.7099	1395.1191

Considering the p-values, the following terms are signficant: `S0`, `time`, `feed`, `S0:pH`, `S0:time`, `pH:time`, and `time:feed`.

Repeat the same for `P`

```
formula = ('P ~ (S0 + pH + time + feed)**2 + ' +
           'I(S0**2) + I(pH**2) + I(time**2) + I(feed**2)')
model = smf.ols(formula, data=data).fit()
print(model.summary2().tables[1].round(4))
```

	Coef.	Std.Err.	t	P>|t|	[0.025	0.975]
Intercept	-7.7603	2.2162	-3.5015	0.0050	-12.6382	-2.8823
S0	0.0712	0.0345	2.0613	0.0637	-0.0048	0.1473
pH	3.1952	1.0381	3.0780	0.0105	0.9104	5.4800
time	0.0085	0.0072	1.1832	0.2617	-0.0073	0.0243
feed	-74.5196	23.5602	-3.1629	0.0090	-126.3752	-22.6640
S0:pH	-0.0065	0.0046	-1.4322	0.1799	-0.0165	0.0035
S0:time	-0.0001	0.0000	-2.3874	0.0360	-0.0002	-0.0000
S0:feed	0.1250	0.2278	0.5490	0.5940	-0.3763	0.6263
pH:time	-0.0018	0.0005	-3.9791	0.0022	-0.0028	-0.0008
pH:feed	15.3345	2.2776	6.7328	0.0000	10.3215	20.3474
time:feed	0.1155	0.0228	5.0726	0.0004	0.0654	0.1657
I(S0 ** 2)	-0.0001	0.0011	-0.0879	0.9315	-0.0026	0.0024
I(pH ** 2)	-0.3177	0.1139	-2.7905	0.0176	-0.5683	-0.0671
I(time ** 2)	-0.0000	0.0000	-0.1631	0.8734	-0.0000	0.0000
I(feed ** 2)	-208.4846	284.6444	-0.7324	0.4792	-834.9827	418.0135

In this model, the following terms are significant: `S0`, `pH`, `feed`, `S0:time`, `pH:time`, `pH:feed`, `time:feed`, and $\texttt{pH}^2$.

(3) Do the models reveal information about the behavior of the PENSIM simulator?

Exercise 8.3 Example 4.5 provides an example of time series tracking engine vibrations of railway vehicle suspension systems. These suspensions can be affected by wheel flats with significant impact on system performance and safety. The dataset

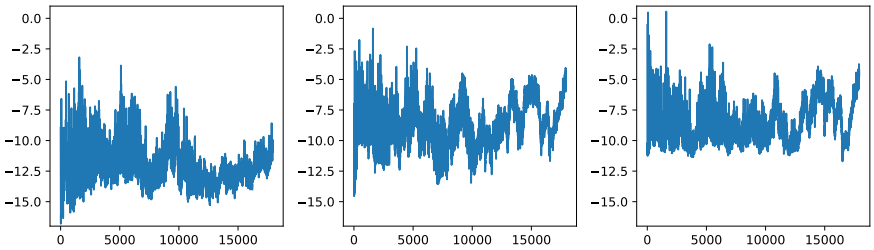
ORDER_PSD.csv includes three series where time is in units of revolution order. This angular resampling transformation is eliminating the variability in revolution time. The three time series correspond to vibrations in healthy suspensions and with wheel flats of 10 mm and 20 mm. In the analysis use the log transformed data.

1. Fit an ARIMA model to the healthy suspension vibration data (see Chapter 6, Kenett et al. 2022b)
2. Fit the same model to the 10 mm and 20 mm wheel flat
3. Compare the model parameters

Solution 8.3 (1) + (2)

```
data = mistat.load_data('ORDER_PSD.csv')
print(data.columns)
fig, axes = plt.subplots(ncols=3, figsize=[10,3])
data['Log[Healthy]'].plot(ax=axes[0])
data['Log[PSD - 10 mm]'].plot(ax=axes[1])
data['Log[PSD - 20 mm]'].plot(ax=axes[2])
for ax in axes:
    ax.set_ylim(-17, 1)
plt.tight_layout()
```

```
Index(['Order', 'Healthy', 'Log[Healthy]', 'PSD - 10 mm', 'Log[PSD -
10 mm]',
       'PSD - 20 mm', 'Log[PSD - 20 mm]'],
      dtype='object')
```



Code similar to what can be found in Example 8.1, was used to derive best parameters for the ARIMA model.

```
parameters = {
    'Log[Healthy]': (2, 1, 1),
    'Log[PSD - 10 mm]': (2, 1, 2),
    'Log[PSD - 20 mm]': (2, 1, 2),
}

results = {}
for series, parameter in parameters.items():
    mod = SARIMAX(data[series], order=parameter)
    results[series] = mod.fit(method='nm', maxiter=600, disp=False)
```

The model predictions are shown in Fig. 8.4.

```
def addSARIMAX_predictions(predictions, ax, series, label=None, linestyle=None):
    pred_mean = predictions.predicted_mean
    pred_mean.index = pred_mean.index + 1
```

```
    pred_mean.plot(ax=ax, label=label, alpha=0.7, color='black', linestyle=linestyle)
    pred_ci = predictions.conf_int()
    pred_ci.index = pred_ci.index + 1
    ax.fill_between(pred_ci.index[1:],
                    pred_ci[f'lower {series}'].iloc[1:], pred_ci[f'upper {series}'].iloc[1:],
                    color='k', alpha=0.2)

fig, axes = plt.subplots(nrows=3, figsize=[6, 12])
for ax, (series, result) in zip(axes, results.items()):
    data[['Order', series]].plot(x='Order', y=series, label='observed', ax=ax,
                                 color='grey', linestyle=':')
    predictions = result.get_prediction(start=1)
    addSARIMAX_predictions(predictions, ax, series, label='One-step ahead forecast')
    forecast = result.get_prediction(start=len(data), end=len(data) + 1000, dynamic=True, full_results=True)
    addSARIMAX_predictions(forecast, ax, series, label='Forecast', linestyle='--')
    ax.set_title(series)
    ax.set_xlabel('Order')
    ax.set_ylabel(series)
    ax.set_ylim(-17, 5)
    ax.legend()

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

(iii) The model parameters are:

```
pd.DataFrame({k: model.params for k, model in results.items()})
```

	Log[Healthy]	Log[PSD - 10 mm]	Log[PSD - 20 mm]
ar.L1	1.331605	1.057846	0.970183
ar.L2	-0.532976	-0.213922	-0.166351
ma.L1	-0.959408	-0.628138	-0.636777
ma.L2	NaN	-0.340708	-0.329220
sigma2	0.153597	0.136405	0.133744

Exercise 8.4 A company operates in 8 cities. The company product is temperature sensitive and management is interested in clustering the 8 locations by temperature characteristics. Monthly average daily minimum and maximum temperatures in these 8 cities, from 2000–2012, is available as dataset **TEMP_WORLD.csv**.[2]

1. Use a smoother to compare the maximum and minimum monthly average temperatures in the 8 cities.
2. Group the cities using maximum and minimum temperature patterns using hierarchical clusters.
3. Group the cities using maximum and minimum temperature patterns using K-means clusters.

Solution 8.4 Load the data and add columns for `Year` and `Month` based on the content of the `Date` column.

```
data = mistat.load_data('TEMP_WORLD.csv')
cities = ['CapeTown', 'BuenosAries', 'Paris', 'Madrid',
          'Tokyo', 'Brisbane', 'Auckland', 'LosAngeles']
data['Year'] = [int(s.split('M')[0]) for s in data['Date']]
data['Month'] = [int(s.split('M')[1]) for s in data['Date']]
```

[2] Also available for download: https://www.stat.auckland.ac.nz/~wild/data/data_from_iNZight/TimeSeriesDatasets_130207/TempWorld1.csv.

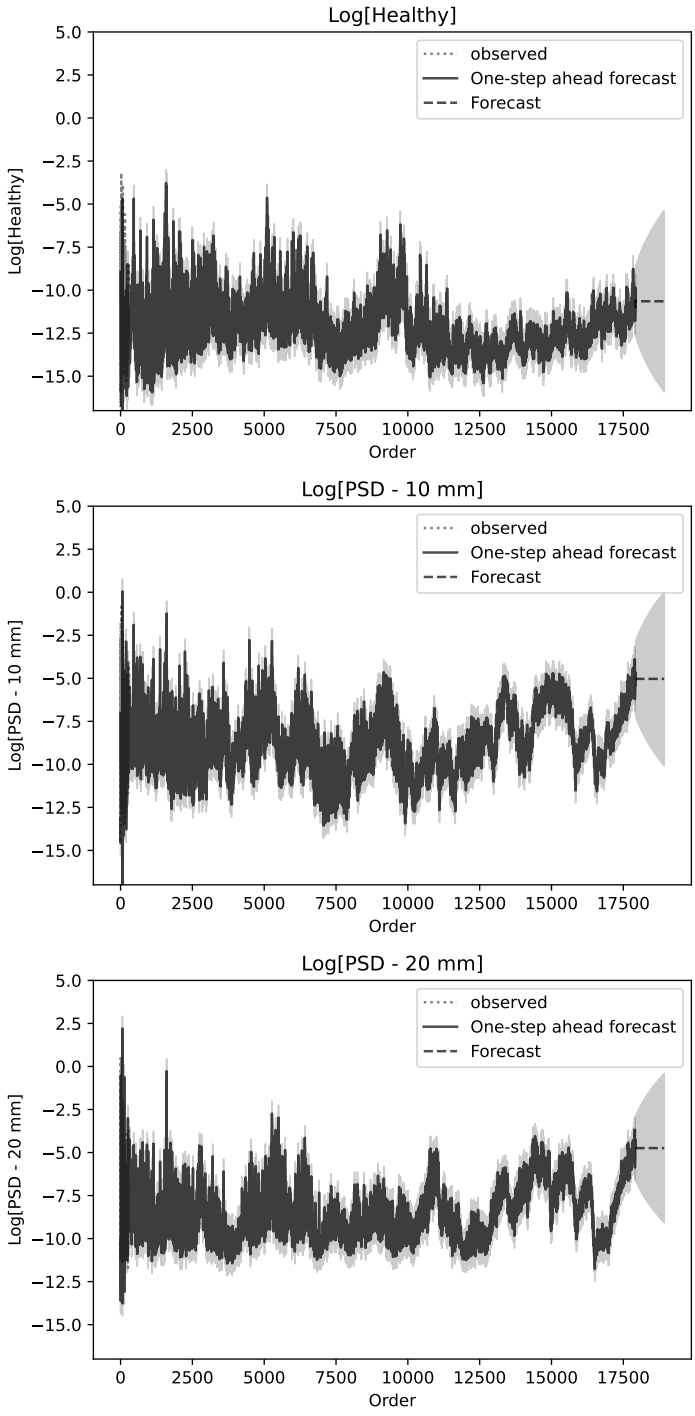


Fig. 8.4: Long term forecast of the log transformed data using ARIMA models

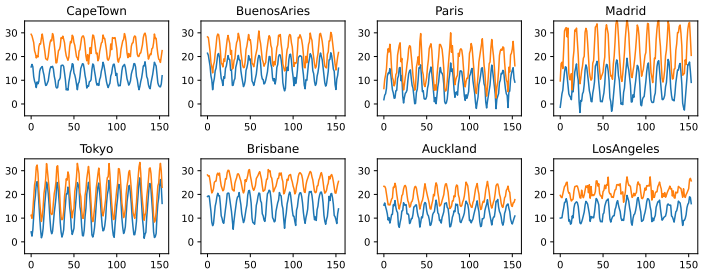


Fig. 8.5: Temperature curves for the eight cities in dataset **TEMP_WORLD.csv**

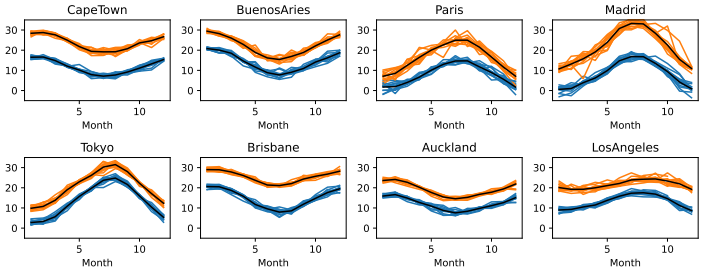


Fig. 8.6: Annual temperature curves for the eight cities in dataset **TEMP_WORLD.csv** by month; the black line shows the average maximum or minimum temperature

The temperature curves show the expected seasonal pattern (see Fig. 8.5).

```
fig, axes = plt.subplots(ncols=4, nrows=2, figsize=[10, 4])
for city, ax in zip(cities, [*axes[0], *axes[1]]):
    data[[f'{city}Min', f'{city}Max']].plot(ax=ax, legend=False)
    ax.set_ylim(-5, 35)
    ax.set_title(city)
plt.tight_layout()
```

Overlaying the different years, shows that the monthly averages are in general consistent over the years. (see Fig. 8.6).

```
meanTemp = data.groupby('Month').mean()

fig, axes = plt.subplots(ncols=4, nrows=2, figsize=[10, 4])
for city, ax in zip(cities, [*axes[0], *axes[1]]):
    for key, group in data.groupby('Year'):
        group.plot(x='Month', y=f'{city}Min', ax=ax, c='C0', legend=False)
        group.plot(x='Month', y=f'{city}Max', ax=ax, c='C1', legend=False)
    meanTemp[[f'{city}Min', f'{city}Max']].plot(ax=ax, legend=False, color='black')
    ax.set_ylim(-5, 35)
    ax.set_title(city)
plt.tight_layout()
```

(1) This observation allows us to reduce the dataset to average monthly temperatures. The average temperature is overlaid in black on the graphs in Fig. 8.6.

```
# determine average monthly temperatures
meanTemp = data.groupby('Month').mean()
```

The shape of the curves shows the different temperature change on the northern (Paris, Madrid, Tokyo, LosAngeles) and southern (CapeTown, BuenosAries, Brisbane, Auckland) hemisphere. Cities also differ in the range temperature changes.

(2) Based on the results from (1), we reduce the dataset prior to clustering to the mean maximum and minimum temperature profiles and concatenate the two profiles into a single row of numbers. Each row contains first the minimum temperature values for January to December and then the maximum temperature values for a given city.

```
months = [*meanTemp.index, *meanTemp.index]
labels = [*(['Min']*12), *(['Max']*12)]
reformatted = []
for city in cities:
    reformatted.append([*meanTemp[f'{city}Min'], *meanTemp[f'{city}Max']])
combined = pd.DataFrame(reformatted, index=cities, columns=[labels, months])
```

The reformatted data are clustered using the *AgglomerativeClustering* method in `scikit-learn`. Prior to the hierarchical clustering, the reformatted data are scaled to zero mean and unit variance using *StandardScaler*. A graphical representation of the resulting clustering is shown in Fig. 8.7.

```
from sklearn.cluster import AgglomerativeClustering
from sklearn.preprocessing import StandardScaler
from mistat import plot_dendrogram

scaler = StandardScaler()
model = AgglomerativeClustering(distance_threshold=0, n_clusters=None)
X = scaler.fit_transform(combined)
model = model.fit(X)

fig, ax = plt.subplots(figsize=[10, 3])
plot_dendrogram(model, ax=ax, labels=combined.index)
ax.set_title('Dendrogram')
plt.show()
```

The hierarchical clustering shows two clearly separated clusters. One with the cities from the northern hemisphere and the other with the cities from the southern hemisphere.

(3) *K*-means clustering with *KMeans* from the `scikit-learn` package using the standardized dataset `X` from (2). Based on the previous analysis, we request two clusters.

```
from sklearn.cluster import KMeans

model = KMeans(n_clusters=2, random_state=1).fit(X)
print('Cluster membership (first two data points)')
pd.DataFrame({
```

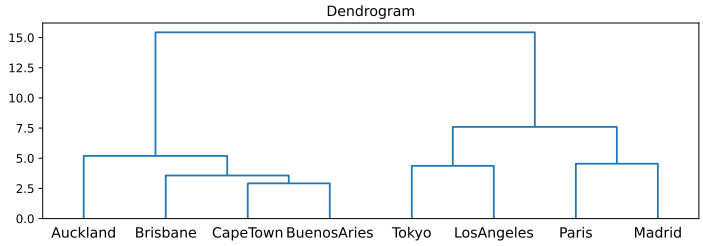


Fig. 8.7: Hierarchical clustering of cities based on annual temperature variations

```
  'City': cities,
  'Cluster': model.predict(X),
})
```

```
Cluster membership (first two data points)
```

```
          City  Cluster
0     CapeTown        0
1  BuenosAries        0
2        Paris        1
3       Madrid        1
4        Tokyo        1
5     Brisbane        0
6     Auckland        0
7   LosAngeles        1
```

The K-means clustering again reveals the two distinct groups.

Fig. 8.8 shows the resulting clusters using a principal component analysis (PCA) of the data.

```
from sklearn.decomposition import PCA
from scipy.spatial import ConvexHull

# use PCA to map the dataset into a 2D space
pca = PCA(n_components=2).fit(X)
coord = pca.transform(X)

fig, ax = plt.subplots()
df = pd.DataFrame({'x': coord[:,0], 'y': coord[:,1],
                   'cluster': model.predict(X),
                   'city': combined.index})

colors = [f'C{i}' for i in model.predict(X)]
ax.scatter(df.x, df.y, color=[f'C{cl}' for cl in df.cluster])
for _, row in df.iterrows():
    ax.annotate(row.city, (row.x, row.y))

plt.show()
```

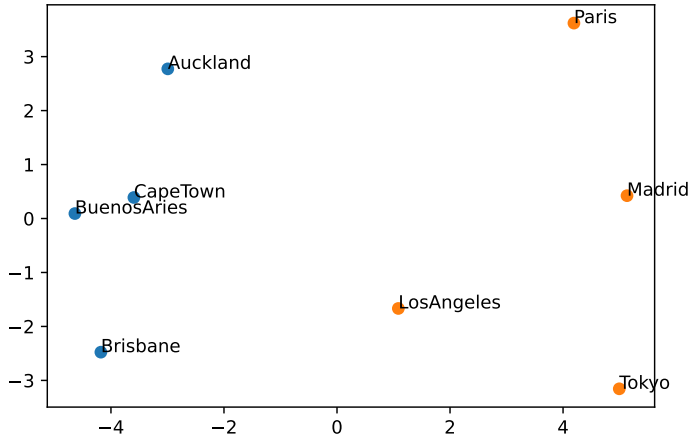


Fig. 8.8: Principal Component Analysis (PCA) of annual temperature variations in the eight cities; the points are colored by their K-means cluster membership

Chapter 9
Reliability Analysis

For the most recent version of the solution manual, go to `https://gedeck.github.io/mistat-code-solutions/IndustrialStatistics/`.

Import required modules and define required functions

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

Exercise 9.1 During 600 hours of manufacturing time, a machine was up 510 hours. It had 100 failures which required a total of 11 hours of repair time. What is the MTTF of this machine? What is its mean time till repair, MTTR? What is the intrinsic availability?

Solution 9.1 $MTTF$ = 5.1 [hr]; $MTTR$ = 6.6 [min]; Intrinsic Availability = 0.979.

Exercise 9.2 The frequency distribution of the lifetime in a random sample of $n = 2,000$ solar cells, under accelerated life testing is the following:

$t[10^3$ [hr]	0-1	1-2	2-3	3-4	4-5	5-
prof. freq.	0.15	0.25	0.25	0.10	0.10	0.15

The relationship of the scale parameters of the life distributions, between normal and accelerated conditions is 10:1.

(i) Estimate the reliability of the solar cells at age t = 4.0 [yr].
(ii) What proportion of solar cells are expected to survive 40,000 [hr] among those which survived 20,000 [hr]?

Solution 9.2 (i) Since 4 yrs = 35,040 hr, the reliability estimate is $R(3.5) = 0.30$. (linear interpolation between $R(3)$ and $R(4)$

(ii) Of the solar cells that survive 20,000 hours, the proportion expected to survive 40,000 hours is $\frac{0.25}{0.60} = 0.42$.

Exercise 9.3 The CDF of the lifetime [months] of an equipment is

$$F(t) = \begin{cases} t^4/20736, & 0 \le t < 12 \\ \\ 1, & 12 \le t \end{cases}$$

(i) What is the failure rate function of this equipment?
(ii) What is the MTTF?
(iii) What is the reliability of the equipment at 4 months?

Solution 9.3 (i) The hazard function is

$$h(t) = \begin{cases} \frac{4t^3}{20736 \cdot \left(1 - \frac{t^4}{20736}\right)}, & 0 \le t < 12 \\ \infty, & 12 \le t. \end{cases}$$

(ii) $MTTF = \int_0^{12} \left(1 - \frac{t^4}{20736}\right) \mathrm{d}t = 9.6$ [Months].
(iii) $R(4) = 1 - F(4) = 0.9877$.

Exercise 9.4 The reliability of a system is

$$R(t) = \exp\{-2t - 3t^2\}, \quad 0 \le t < \infty.$$

(i) What is the failure rate of this system at age $t = 3$?
(ii) Given that the system reached the age of $t = 3$, what is its reliability for two additional time units?

Solution 9.4 (i) $h(t) = 2 + 6t$, $h(3) = 20$. **(ii)** $R(5)/R(3) = 0$.

Exercise 9.5 An aircraft has four engines but can land using only two engines.

(i) Assuming that the reliability of each engine, for the duration of a mission, is $R = 0.95$, and that engine failures are independent, compute the mission reliability of the aircraft.
(ii) What is the mission reliability of the aircraft if at least one functioning engine must be on each wing?

Solution 9.5 (i) $1 - B(1; 4, 0.95) = 0.9995$ **(ii)** $(1 - B(0; 2, 0.95))^2 = (1 - 0.05^2)^2 = 0.9950$.

In Python:

```
print('(i)', 1-stats.binom(4, 0.95).pmf(1))
print('(ii)', (1-stats.binom(2, 0.95).pmf(0))**2)
```

```
(i) 0.999525
(ii) 0.9950062499999999
```

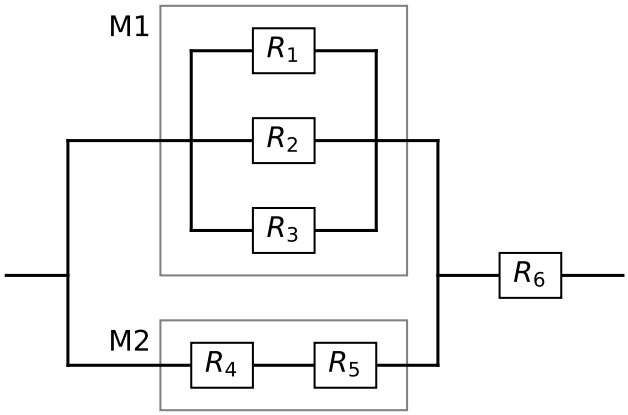


Fig. 9.1: System schema

Exercise 9.6 (i) Draw a block diagram of a system having the structure function

$$R_{\text{sys}} = \psi_s(\psi_p(\psi_{M_1}, \psi_{M_2}), R_6), \;\; \psi_{M_1} = \psi_p(R_1, R_2R_3), \;\; \psi_{M_2} = \psi_2(R_4, R_5)$$

(ii) Determine R_{sys} if all the components act independently and have the same reliability $R = 0.8$.

Solution 9.6 (a) The block diagram of the system is shown in Fig. 9.1.
(b) $R_{M_1} = 1 - (1 - R_1)(1 - R_2)(1 - R_3)$ and $R_{M_2} = R_4R_5$. Thus,

$$\begin{aligned} R_{sys} &= [1 - (1 - R_1)(1 - R_2)(1 - R_3)(1 - R_4R_5)]R_6 \\ &= R_1R_6 + R_2R_6 + R_3R_6 - R_1R_2R_6 + R_4R_5R_6 \\ &\quad - R_1R_3R_6 - R_2R_3R_6 + R_1R_2R_3R_6 \\ &\quad - R_1R_4R_5R_6 - R_2R_4R_5R_6 - R_3R_4R_5R_6 \\ &\quad + R_2R_3R_4R_5R_6 + R_1R_3R_4R_5R_6 + R_1R_2R_4R_5R_6 \\ &\quad - R_1R_2R_3R_4R_5R_6. \end{aligned}$$

If all values of $R = 0.8$ then $R_{sys} = 0.7977$.

Exercise 9.7 Consider a system of n components in a series structure. Let $R_1, \cdots, R_n$ be the reliabilities of the components. Show that

$$R_{\text{sys}} \geq 1 - \sum_{i=1}^{n}(1 - R_i).$$

Solution 9.7 Extending Bonferroni's inequality,

$$\begin{aligned}\Pr\{C_1 = 1, \ldots, C_n = 1\} &= 1 - \Pr\left\{\bigcup_{i=1}^{n}\{C_i = 0\}\right\} \\ &\geq 1 - \sum_{i=1}^{n}\Pr\{C_i = 0\} \\ &= 1 - \sum_{i=1}^{n}(1 - R_i);\end{aligned}$$

where $\{C_i = 1\}$ is the event that the i-th component functions and $\{C_i = 0\}$ is the event that it fails. Since $R_{sys} = \Pr\{C_1 = 1, \ldots, C_n = 1\}$ for a series structure, the inequality follows.

Exercise 9.8 A 4 out of 8 system has identical components whose life lengths T [weeks] are independent and identically distributed like a Weibull $W\left(\frac{1}{2}, 100\right)$. What is the reliability of the system at $t_0 = 5$ weeks?

Solution 9.8 The reliability of a component is $\theta(5) = \Pr\left\{W\left(\frac{1}{2}, 100\right) > 5\right\} = 0.79963$.

Therefore, $R_{sys} = 1 - B(3; 8, 0.7996) = 0.9895$.

Exercise 9.9 A system consists of a main unit and two standby units. The lifetimes of these units are exponential with mean $\beta = 100$ [hr]. The standby units undergo no failure while idle. Switching will take place when required. What is the MTTF of the system? What is the reliability function of this system?

Solution 9.9 The life length of the system is the sum of 3 independent $E(100)$, that is $G(3, 100)$ (gamma distribution). Hence the $MTTF = 300$ hr. The reliability function of the system is

$$R_{sys}(t) = 1 - G(t; 3, 100).$$

Exercise 9.10 Suppose that the TTF in a renewal cycle has a $W(\alpha, \beta)$ distribution and that the TTR has a lognormal distribution $\mathrm{LN}(\mu, \sigma)$. Assume further that TTF and TTR are independent. What are the mean and standard deviation of a renewal cycle.

Solution 9.10 Let C be the length of the renewal cycle.

$$E\{C\} = \beta\Gamma\left(1 + \frac{1}{\alpha}\right) + e^{\mu+\sigma^2/2}.$$

$$\sigma(C) = \left[\beta^2\left(\Gamma\left(1 + \frac{2}{\alpha}\right) - \Gamma^2\left(1 + \frac{1}{\alpha}\right)\right) + e^{2\mu+\sigma^2}(e^{\sigma^2} - 1)\right]^{1/2}.$$

Exercise 9.11 Suppose that a renewal cycle has the normal distribution $N(100, 10)$. Determine the p.d.f. of $N_R(200)$.

Solution 9.11 Let $C_1, C_2, \ldots$ denote the renewal cycle times, $C_i \sim N(100, 10)$. $N_R(200)$ is a random variable representing the number of repairs that occur during the time interval $(0, 200]$.

$$\Pr\{N_R(200) = 0\} = \Pr\{C_1 > 200\} = 1 - \Phi\left(\frac{200 - 100}{10}\right) = 1 - \Phi(10) = 0.$$

For $k = 1, 2, 3, \ldots, C_1 + \cdots + C_k \sim N(100k, 10\sqrt{k})$ and so

$$\begin{aligned}\Pr\{N_R(200) = k\} &= \Pr\{N_R(200) \geq k\} - \Pr(N_R(200) \geq k + 1\} \\ &= \Pr\{C_1 + \ldots + C_k \leq 200\} - \Pr\{C_1 + \ldots + C_{k+1} \leq 200\} \\ &= \Phi\left(\frac{200 - 100k}{10\sqrt{k}}\right) - \Phi\left(\frac{200 - 100(k+1)}{10\sqrt{k+1}}\right) \\ &= \Phi\left(\frac{20}{\sqrt{k}} - 10\sqrt{k}\right) - \Phi\left(\frac{20}{\sqrt{k+1}} - 10\sqrt{k+1}\right),\end{aligned}$$

where $\Phi(\cdot)$ denotes the c.d.f. of a $N(0, 1)$ distribution. Thus the p.d.f. of $N_R(200)$ is given by the following table

k	$\Pr\{N_R(200) = k\}$
0	0
1	0.5
2	0.5
3	0
4	0
$\vdots$	$\vdots$

Exercise 9.12 Let the renewal cycle C be distributed like $N(100, 10)$. Approximate $V(1000)$.

Solution 9.12 $V(1000) = \sum_{n=1}^{\infty} \Phi\left(\frac{100-10n}{\sqrt{n}}\right) \approx 9.501$.

Exercise 9.13 Derive the renewal density $v(t)$ for a renewal process with $C \sim N(100, 10)$.

Solution 9.13 $v(t) = \frac{1}{10\sqrt{n}} \sum_{n=1}^{\infty} \phi\left(\frac{t-100n}{10\sqrt{n}}\right)$, where $\phi(Z)$ is the p.d.f. of $N(0, 1)$.

Exercise 9.14 Two identical components are connected in parallel. The system is not repaired until both components fail. Assuming that the TTF of each component is exponentially distributed, $E(\beta)$, and the total repair time is $G(2, \gamma)$, derive the Laplace transform of the availability function $A(t)$ of the system.

Solution 9.14 Since $TTF \sim \max(E_1(\beta), E_2(\beta))$, $F(t) = (1 - e^{-t/\beta})^2$ and $R(t) = 1 - (1 - e^{-t/\beta})^2$.

Using Laplace transforms we have

$$R^*(s) = \frac{\beta(3+s\beta)}{(1+s\beta)(2+s\beta)}, \quad f^*(s) = \frac{2}{(1+\beta s)(2+\beta s)}, \quad g^*(s) = \frac{1}{(1+s\gamma)^2} \quad \text{and}$$

$$A^*(s) = \frac{\beta(3+s\beta)(1+\gamma s)^2}{s[3\beta+4\gamma+(\beta^2+6\beta\gamma+2\gamma^2)s+(2\beta^2\gamma+3\beta\gamma^2)s^2+\beta^2\gamma^2 s^3]}.$$

Exercise 9.15 Simulate a sample of 100 TTF of a system comprised of two components connected in parallel, where the life distribution of each component (in hours) is $E(100)$. Similarly, simulate a sample of 100 repair times (in hours), having a $G(2,1)$ distribution. Estimate the expected value and variance of the number of renewals in 2000 [hr].

Solution 9.15 For our samples, with $M = 500$ runs, we get the following estimates:

$$E\{N_R(2000)\} = V(2000) = 16.574 \quad \text{and} \quad \text{Var}\{N_R(2000)\} = 7.59.$$

Exercise 9.16 In a given life test, $n = 15$ units are placed to operate independently. The time till failure of each unit has an exponential distribution with mean 2000 [hr]. The life test terminates immediately after the 10th failure. How long is the test expected to last?

Solution 9.16 The expected length of the test is 2069.8 [hr].

Exercise 9.17 If n units are put on test and their TTF are exponentially distributed with mean β, the time elapsed between the rth and $(r+1)$th failure, i.e., $\Delta_{n,r} = T_{n,r+1} - T_{n,r}$, is exponentially distributed with mean $\beta/(n-r)$, $r = 0, 1, \cdots, n-1$. Also, $\Delta_{n,0}, \Delta_{n,2}, \cdots, \Delta_{n,n-1}$ are independent. What is the variance of $T_{n,r}$? Use this result to compute the variance of the test length in the previous exercise.

Solution 9.17 Let $T_{n,0} \equiv 0$. Then $T_{n,r} = \sum_{j=1}^{r}(T_{n,j} - T_{n,j-1}) = \sum_{j=1}^{r} \Delta_{n,j-1}$. Thus,

$$V\{T_{n,r}\} = \beta^2 \sum_{j=1}^{r} \left(\frac{1}{n-j+1}\right)^2.$$

In Exercise 9.16, $\beta = 2000$ hr, $n = 15$ and $r = 10$. This yields

$$V\{T_{15,10}\} = 4 \times 10^6 \times 0.116829 = 467,316 \text{ (hr)}^2.$$

Exercise 9.18 Consider again the previous exercise. How would you estimate unbiasedly the scale parameter β if the r failure times $T_{n,1}, T_{n,2}, \cdots, T_{n,r}$ are given? What is the variance of this unbiased estimator?

Solution 9.18 Let $S_{n,r} = \sum_{i=1}^{r} T_{n,i} + (n-r)T_{n,r}$. An unbiased estimator of β is

$$\hat{\beta} = \frac{S_{n,r}}{r}, \qquad V\{\hat{\beta}\} = \frac{\beta^2}{r}.$$

Exercise 9.19 Simulate a random sample of 100 failure times, following the Weibull distribution $W(2.5, 10)$. Draw a Weibull Probability plot of the data. Estimate the

parameters of the distribution from the parameters of the linear regression fitted to the Q–Q plot.

Solution 9.19 The Weibull QQ plotting is based on the regression of $\ln T_{(i)}$ on

$$W_i = \ln\left(-\ln\left(1 - \frac{i}{n+1}\right)\right).$$

The linear relationship is $\ln T_{(i)} = \frac{1}{\nu} W_i + \ln\beta$. The slope of the straight line estimates $\frac{1}{\nu}$ and the intercept estimates $\log\beta$.

```
np.random.seed(1)
n = 100
lnTi = np.log(sorted(stats.weibull_min(2.5, scale=10).rvs(n)))
Wi = [np.log(-np.log(1 - i / (n+1))) for i in range(1, n + 1)]
df = pd.DataFrame({'Wi': Wi, 'lnTi': lnTi})
model = smf.ols('lnTi ~ Wi + 1', data=df).fit()
print(model.params)
intercept, slope = model.params
fig, ax = plt.subplots(figsize=(4, 4))
ax.plot((Wi[0], Wi[-1]),
        (slope * Wi[0] + intercept, slope * Wi[-1] + intercept),
        color='grey')
ax.scatter(Wi, lnTi, color='black')
ax.set_xlabel(r'$W_{(i)}$')
ax.set_ylabel(r'$\ln(T_i)$')
plt.show()
```

```
Intercept    2.294805
Wi           0.496692
dtype: float64
```

For our sample of $n = 100$ from $W(2.5, 10)$ the regression of $\ln T_{(i)}$ on W_i is

$$\ln(\text{TTF}) = 2.295 + 0.497W$$

Predictor	Coef	Stdev	t-ratio	p
Constant	2.2948	0.018	130.703	0.000
W	0.4967	0.013	37.629	0.000

R-sq = 0.935 R-sq(adj) = 0.935.

```
beta = np.exp(intercept)
nu = 1 / slope
print(f'beta {beta:.3f}, nu {nu:.3f}')
```

```
beta 9.922, nu 2.013
```

The graphical estimate of β is $\hat{\beta} = \exp(2.295) = 9.922$. The graphical estimate of ν is $\hat{\nu} = \frac{1}{0.4643} = 2.013$. Both estimates are close to the nominal values. In Fig. 9.2, we see the Weibull probability plot. This graph puts the sorted observations $W_{(i)}$ on the x-axis and $\ln T_i$ on the y-axis where T_i is the calculated probability of occurrence for each observation $W_{(i)}$ assuming a Weibull distribution. As shown above, the $\hat{\beta}$ and $\hat{\nu}$ can be calculated from the slope and intercept of the liner regression line.

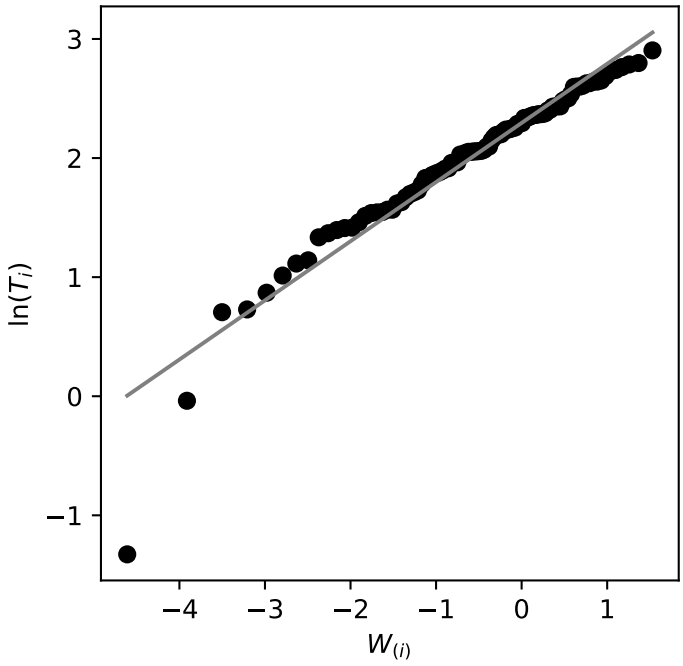


Fig. 9.2: Weibull probability plot

Exercise 9.20 The following is a random sample of the compressive strength of 20 concrete cubes [kg/cm^2]:

94.9, 106.9, 229.7, 275.7, 144.5, 112.8, 159.3, 153.1, 270.6, 322.0, 216.4, 544.6, 266.2, 263.6, 138.5, 79.0, 114.6, 66.1, 131.2, 91.1

Make a lognormal Q–Q plot of these data and estimate the mean and standard deviation of this distribution.

Solution 9.20 The regression of $Y_i = \ln X_{(i)}$, $i = 1, \ldots, n$ on the normal scores $Z_i = \Phi^{-1}\left(\frac{i-3/8}{n+1/4}\right)$ is

```
Xi = [94.9, 106.9, 229.7, 275.7, 144.5, 112.8, 159.3, 153.1,
        270.6, 322.0, 216.4, 544.6, 266.2, 263.6, 138.5, 79.0,
        114.6, 66.1, 131.2, 91.1]
n = len(Xi)
Zi = [stats.norm().ppf((i - 3/8)/(n + 1/4)) for i in range(1, n+1)]

df = pd.DataFrame({'Zi': Zi, 'lnXi': np.log(sorted(Xi))})
model = smf.ols('lnXi ~ Zi + 1', data=df).fit()

intercept, slope = model.params
fig, ax = plt.subplots(figsize=(4, 4))
```

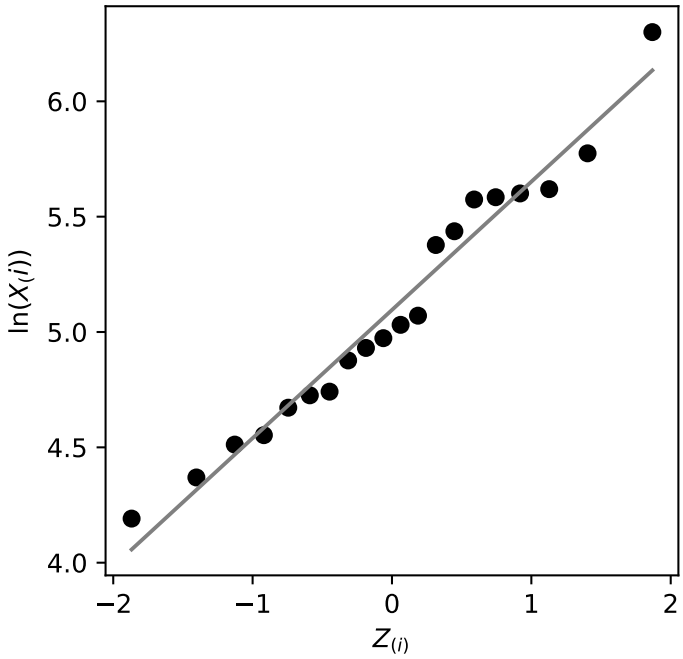


Fig. 9.3: Normal Probability Plot of ln X

```
ax.plot((Zi[0], Zi[-1]),
        (slope * Zi[0] + intercept, slope * Zi[-1] + intercept),
        color='grey')
ax.scatter(Zi, df['lnXi'], color='black')
ax.set_xlabel(r'$Z_{(i)}$')
ax.set_ylabel(r'$\ln(X_(i))$')
plt.show()
```

$$\ln(X) = 5.10 + 0.556Z.$$

```
Predictor      Coef      Stdev    t-ratio        p
Constant    5.09571    0.02201     232.23    0.000
Z           0.55589    0.02344      23.79    0.000
  R-sq = 0.969   R-sq(adj) = 0.967.
```

```
mu = intercept
sigma = slope
print(f'mu {mu:.3f}, sigma {sigma:.3f}')
```

```
mu 5.096, sigma 0.556
```

The intercept $\hat{a} = 5.09571$ is an estimate of μ, and the slope $\hat{b} = 0.5559$ is an estimate of σ. The mean of $\mathrm{LN}(\xi, \sigma^2)$ is $\xi = e^{\mu+\sigma^2/2}$ and its variance is $D^2 =$

$e^{2\mu+\sigma^2}(e^{\sigma^2}-1)$. Thus, the graphical estimates are $\hat{\xi} = 190.608$ and $\hat{D} = 13154.858$. The mean $\bar{X}$ and variance S^2 of the given sample are 189.04 and 12948.2. In Fig. 9.3 we see the normal probability plotting of the data.

Exercise 9.21 The following data represent the time till first failure [days] of electrical equipment. The data were censored after 400 days.

$$13,\ 157,\ 172,\ 176,\ 249,\ 303,\ 350,\ 400^+,\ 400^+.$$

(Censored values appear as x^+.) Make a Weibull Q–Q plot of these data and estimate the median of the distribution.

Solution 9.21 The Q–Q plot is given in Fig. 9.4.

```
np.random.seed(1)
data = [13, 157, 172, 176, 249, 303, 350, 400, 400]
n = len(data)
lnTi = np.log(data)
Wi = [np.log(-np.log(1 - i / (n+1))) for i in range(1, n + 1)]
# exclude the censored data for the regression analysis
df = pd.DataFrame({'Wi': Wi[:7], 'lnTi': lnTi[:7]})
model = smf.ols('lnTi ~ Wi + 1', data=df).fit()
print(model.params)
intercept, slope = model.params
fig, ax = plt.subplots(figsize=(4, 4))
ax.plot((Wi[0], Wi[-1]),
        (slope * Wi[0] + intercept, slope * Wi[-1] + intercept),
        color='grey')
ax.scatter(Wi[:-2], lnTi[:-2], color='black')
ax.scatter(Wi[-2:], lnTi[-2:], color='lightgrey')
ax.set_xlabel(r'$W_{(i)}$')
ax.set_ylabel(r'$\ln(T_i)$')
plt.show()
```

```
Intercept    5.959374
Wi           1.168853
dtype: float64
```

The regression of $\ln T_{(i)}$ on $W_i = \ln\left(-\ln\left(1-\frac{i}{10}\right)\right)$ for $i = 1, \ldots, 7$ is

$$\ln(T) = 5.96 + 1.17W \tag{9.0.1}$$

```
Predictor      Coef     Stdev    t-ratio        p
Constant     5.9594    0.3068      19.43    0.000
W            1.1689    0.2705       4.32    0.008
  s = 0.5627   R-sq = 0.789   R-sq(adj) = 0.747.
```

According to this regression line, the estimates of β and ν are $\hat{\beta} = \exp(5.9594) = 387.378$ and $\hat{\nu} = 1/1.1689 = 0.855$. The estimate of the median of the distribution is

$$\hat{Me} = \hat{\beta}\left(-\ln\left(\frac{1}{2}\right)\right)^{1/\hat{\nu}} = 252.33.$$

Exercise 9.22 Make a PL (Kaplan–Meier) estimate of the reliability function of an electronic device, based on 50 failure times in dataset **ELECFAIL.csv**.

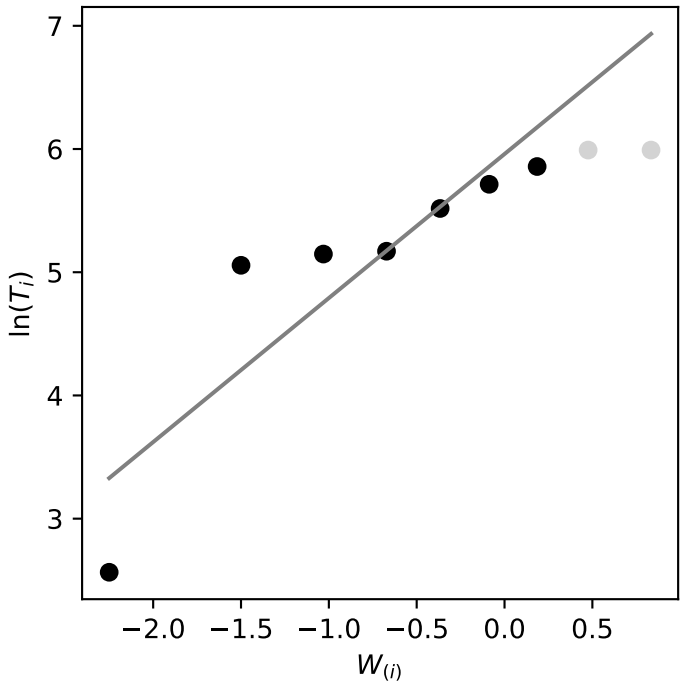


Fig. 9.4: Weibull probability plot

Solution 9.22 Use the `lifelines` package to fit a Kaplan-Meier estimator. The reliability function is shown in Fig. 9.5.

```
elecfail = mistat.load_data('ELECFAIL.csv')

kmf = lifelines.KaplanMeierFitter()
kmf.fit(elecfail)
kmf.plot_survival_function()
plt.show()
```

Exercise 9.23 Assuming that the failure times in dataset **ELECFAIL.csv** come from an exponential distribution $E(\beta)$, compute the MLE of β and of $R(50;\beta) = \exp\{-50/\beta\}$. [The MLE of a function of a parameter is obtained by substituting the MLE of the parameter in the function.] Determine confidence intervals for β and for $R(50;\beta)$ at level of confidence 0.95.

Solution 9.23 Use the `lifelines` package to fit an exponential model.

```
elecfail = mistat.load_data('ELECFAIL.csv')

kmf = lifelines.ExponentialFitter()
kmf.fit(elecfail)
```

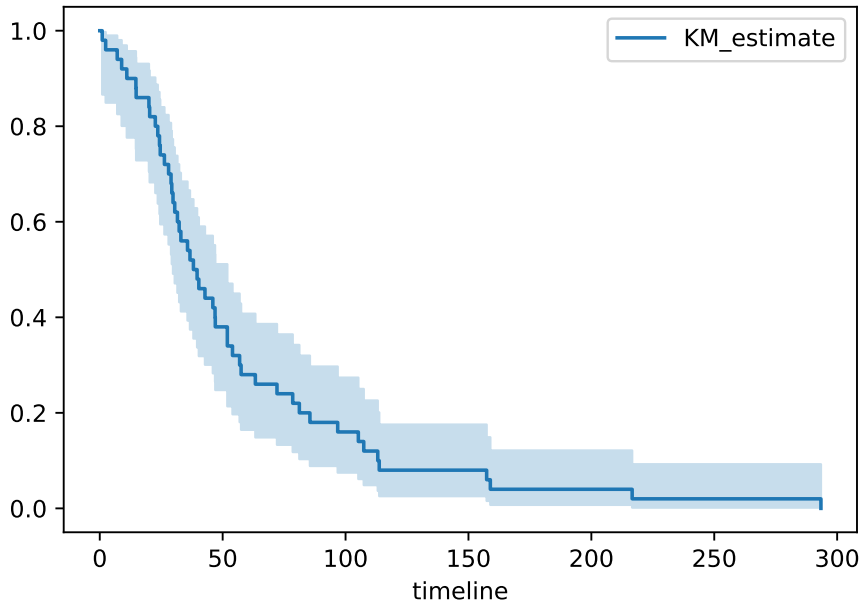


Fig. 9.5: Kaplan-Meier Estimator of The Reliability of An Electric Device

```
<lifelines.ExponentialFitter:"Exponential_estimate", fitted with 50
total observations, 0 right-censored observations>
```

```
kmf.print_summary()
```

	coef	se(coef)	coef lower 95%	coef upper 95%	cmp to	z	p	-log2(p)
lambda_	57.07	8.07	41.25	72.89	0.00	7.07	0.00	39.24

Note that the `lifelines` package uses `lambda_` for β. We can use the result to calculate $R(50)$ and it's confidence limits.

```
beta = kmf.lambda_
print(f'R(50) = {np.exp(-50/beta):.4f}')
ci = kmf.summary[['coef lower 95%', 'coef upper 95%']]
print(f'conf.int {np.exp(-50/ci)}')
```

```
R(50) = 0.4164
conf.int          coef lower 95%  coef upper 95%
lambda_        0.297577        0.503598
```

The MLE of β is $\hat{\beta} = \bar{X} = 57.07$. Since $R(t) = \exp(-t/\beta)$, the MLE of R(50) is $\hat{R}(50) = 0.4164$. Note that `lifelines` uses a normal approximation for the

confidence interval. Hence, confidence limits for β, with $1 - \alpha = 0.95$ are (41.25, 72.89). Substituting these limits into $R(t)$ gives $(0.298, 0.504)$ as a 0.95 confidence interval for $R(50)$.

Exercise 9.24 The following are values of 20 random variables having an exponential distribution $E(\beta)$. The values are censored at $t^* = 200$.

96.88, 154.24, 67.44, 191.72, 173.36, 200, 140.81, 200, 154.71, 120.73, 24.29, 10.95, 2.36, 186.93, 57.61, 99.13, 32.74, 200, 39.77, 39.52.

Determine the MLE of β. Use β equal to the MLE, to estimate the standard deviation of the MLE and to obtain confidence interval for β, at level $1 - \alpha = 0.95$. [This simulation is called an empirical Bootstrap.]

Solution 9.24 Use the `lifelines` package to fit an exponential model.

```
T = np.array([96.88, 154.24, 67.44, 191.72, 173.36, 200, 140.81,
     200, 154.71, 120.73, 24.29, 10.95, 2.36, 186.93, 57.61,
     99.13, 32.74, 200, 39.77, 39.52])
E = np.array([ti < 200 for ti in T])

kmf = lifelines.ExponentialFitter()
kmf.fit(T, E)
```

```
<lifelines.ExponentialFitter:"Exponential_estimate", fitted with 20
total observations, 3 right-censored observations>
```

```
kmf.print_summary()
```

	coef	se(coef)	coef lower 95%	coef upper 95%	cmp to	z	p	-log2(p)
lambda_	129.01	31.29	67.68	190.34	0.00	4.12	0.00	14.71

We have $n = 20$, $K_n = 17$ and $\hat{\beta} = 129.011$.

We use the following code for the empirical bootstrap.

```
n_boot=500
idx = list(range(len(T)))
def stat_func(x):
    epf = lifelines.ExponentialFitter().fit(T[x], E[x])
    return epf.params_['lambda_']
ci, dist = pg.compute_bootci(idx, func=stat_func, n_boot=n_boot, confidence=0.95,
                             method='per', seed=1, return_dist=True)
se_lambda = np.std(dist)
print(f'std(lambda): {se_lambda:.3f}')
print(f'conf.int.: {ci.round(3)}')
```

```
std(lambda): 27.863
conf.int.: [ 83.75 191.61]
```

With $M = 500$ runs, the estimated STD of $\hat{\beta}$ is 27.863, with confidence interval (83.75, 191.61).

Exercise 9.25 Determine n^0 and r for a frequency censoring test for the exponential distribution, where the cost of a unit is 10 times bigger than the cost per time unit of testing. We wish that S.E.$\{\hat{\beta}_n\} = 0.1\beta$, and the expected cost should be minimized at $\beta = 100$ [hr]. What is the expected cost of this test, at $\beta = 100$, when $c_1 = \$1$ [hr].

Solution 9.25 For $\beta = 100$, $S.E.\{\hat{\beta}_n\} = 0.1\beta = 10 = \frac{100}{\sqrt{r}}$. Hence $r = 100$ and

$$n^0 \approx \frac{100}{2}\left(1 + \left(1 + 4\frac{100}{1000}\right)^{1/2}\right) = 109.16 \approx 110.$$

$E\{T_{n,r}\} = 235.327$ and the expected cost of testing is $E\{C\} = 10 \times n + 1 \times E\{T_{n,r}\} =$ \$1335.33.

Exercise 9.26 Dataset **WEIBUL.csv** contains the values of a random sample of size $n = 50$ from a Weibull distribution.

(i) Obtain MLE of the scale and shape parameters β and ν.
(ii) Use the MLE estimates

of β and ν, to obtain parametric bootstrap EBD of the distribution of $\hat{\beta}$, $\hat{\nu}$, with $M = 500$ runs. Estimate from this distribution the standard deviations of $\hat{\beta}$ and $\hat{\nu}$. Compare these estimates to the large sample approximations.

Solution 9.26 (a)

```
T = mistat.load_data('WEIBUL.csv')

kmf = lifelines.WeibullFitter()
kmf.fit(T)
```

```
<lifelines.WeibullFitter:"Weibull_estimate", fitted with 50 total
observations, 0 right-censored observations>
```

```
kmf.print_summary()
```

	coef	se(coef)	coef lower 95%	coef upper 95%	cmp to	z	p	-log2(p)
lambda_	27.08	2.94	21.31	32.85	1.00	8.86	0.00	60.17
rho_	1.37	0.15	1.09	1.66	1.00	2.57	0.01	6.63

For the **WEIBUL.csv** dataset we obtained $\hat{\beta} = 27.0789$ and $\hat{\nu} = 1.374$.
(b)

```
n_boot=5
idx = list(range(len(T)))
def stat_func(x):
    epf = lifelines.WeibullFitter().fit(T[x])
    return epf.params_['lambda_']

ci, dist = pg.compute_bootci(idx, func=stat_func, n_boot=n_boot, confidence=0.95,
                             method='per', seed=1, return_dist=True)
```

```
se_beta = np.std(dist)
print(f'std(beta): {se_beta:.3f}')

def stat_func(x):
    epf = lifelines.WeibullFitter().fit(T[x])
    return epf.params_['rho_']
ci, dist = pg.compute_bootci(idx, func=stat_func, n_boot=n_boot, confidence=0.95,
                             method='per', seed=1, return_dist=True)
se_nu = np.std(dist)
print(f'nu(std): {se_nu:.3f}')
```

```
std(beta): 2.874
nu(std): 0.072
```

The EBD estimates are $SE\{\hat{\beta}_{50}\} = 2.874$ and $SE\{\hat{\nu}_{50}\} = 0.072$. The asymptotic estimates are $SE\{\hat{\beta}_{50}\} = 2.94$ and $SE\{\hat{\nu}_{50}\} = 0.15$. The EBD estimates and the asymptotic estimates of the standard deviations of $\hat{\beta}$ and $\hat{\nu}$ are very similar.

Exercise 9.27 In binomial life testing by a fixed size sample, how large should the sample be in order to discriminate between $R_0 = 0.99$ and $R_1 = 0.90$, with $\alpha = \beta = 0.01$? [α and β denote the probabilities of error of Type I and II.]

Solution 9.27 To discriminate between $R_0 = 0.99$ and $R_1 = 0.90$ with $\alpha = \beta = 0.01$, $n \approx 107$.

Exercise 9.28 Design the Wald SPRT for binomial life testing, in order to discriminate between $R_0 = 0.99$ and $R_1 = 0.90$, with $\alpha = \beta = 0.01$. What is the expected sample size, ASN, if $R = 0.9$?

Solution 9.28 For $R_0 = 0.99$, $R_1 = 0.90$, and $\alpha = \gamma = 0.01$ we get $s = 0.9603$, $h_1 = 1.9163$, $h_2 = 1.9163$ and $\text{ASN}(0.9) = 31.17$.

Exercise 9.29 Design a Wald SPRT for exponential life distribution, to discriminate between $R_0 = 0.99$ and $R_1 = 0.90$, with $\alpha = \beta = 0.01$. What is the expected sample size, ASN, when $R = 0.90$?

Solution 9.29 Without loss of generality, assume that $t = 1$. Thus, $R_0 = e^{-1/\beta_0} = 0.99$, or $\beta_0 = 99.5$. Also, $R_1 = 0.9$, or $\beta_1 = 9.49$. The parameters of the SPRT are: $h_1 = 48.205$, $h_2 = 48.205$ and $s = 24.652$. $ASN(0.9) \approx 3$.

Exercise 9.30 $n = 20$ computer monitors are put on accelerated life testing. The test is an SPRT for Poisson processes, based on the assumption that the TTF of a monitor, in those conditions, is exponentially distributed. The monitors are considered to be satisfactory if their MTBF is $\beta \geq 2000$ [hr] and considered to be unsatisfactory if $\beta \leq 1500$ [hr]. What is the expected length of the test if $\beta = 2000$ [hr].

Solution 9.30 Using $\alpha = \gamma = 0.05$ we have $n = 20$, $H_0 : \beta = 2000$, $H_1 : \beta = 1500$. Thus, $\lambda_0 = 0.0005$ and $\lambda_1 = 0.00067$. The parameters of the SPRT are:

$$h_1 = \frac{\log(\frac{95}{5})}{\log(\frac{\lambda_1}{\lambda_0})} = 10.061, \qquad h_2 = 10.061$$

and

$$
\begin{aligned}
s &= \frac{\lambda_1 - \lambda_0}{\log(\frac{\lambda_1}{\lambda_0})} = 0.000581. \\
E_{\beta_0}\{\tau\} &= \frac{2000}{20} E_{\beta}\{X_{20}(\tau)\} \\
&\approx 100 \left(\frac{10.061 - 0.95 \times 20.122}{1 - 2000 \times 0.000581} \right) \\
&= 5589.44 \text{ [hr]}.
\end{aligned}
$$

Exercise 9.31 A product has an exponential life time with MTTF $\beta = 1000$ [hr]. 1% of the products come out of production with MTTF of $\gamma = 500$ [hr]. A burn-in of $t^* = 300$ [hr] takes place. What is the expected life of units surviving the burn-in? Is such a long burn-in justified?

Solution 9.31 With $\beta = 1000$, $p = 0.01$, $\gamma = 500$ and $t^* = 300$, we get

$$
F^*(t) = 1 - \frac{0.01e^{-t/500} + 0.99e^{-t/1000}}{0.01e^{-300/500} + 0.99e^{-300/1000}}, \quad t \geq 300.
$$

The expected life of units surviving the burn-in is thus,

```
factor = 0.01*np.exp(-300/500) + 0.99*np.exp(-300/1000)
integral = 0.01*500*np.exp(-300/500) + 0.99*1000*np.exp(-300/1000)
with_burn_in = 300 + integral/factor
no_burn_in = 0.01 * 500 + 0.99 * 1000
with_burn_in - no_burn_in
```

```
301.2862836203999
```

$$
\begin{aligned}
\beta^* &= 300 + \frac{1}{0.7389} \int_{300}^{\infty} \left(0.01 \exp\left\{ -\frac{t}{500} \right\} + 0.99 \exp\left\{ -\frac{t}{1000} \right\} \right) \mathrm{d}t \\
&= 300 + \frac{5}{0.7389} \exp\left\{ -\frac{300}{500} \right\} + \frac{990}{0.7389} \exp\left\{ -\frac{300}{1000} \right\} \\
&= 1296.29 \text{ [hr]}.
\end{aligned}
$$

Without burn-in, the expected life of these units is $0.01 * 500 + 0.99 * 1000 = 995$ [hr]. The expected life of the units in the field increases by 300 [hr].

Chapter 10
Bayesian Reliability Estimation and Prediction

For the most recent version of the solution manual, go to `https://gedeck.github.io/mistat-code-solutions/IndustrialStatistics/`.

Import required modules and define required functions

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

Exercise 10.1 Suppose that the TTF of a system is a random variable having exponential distribution, $E(\beta)$. Suppose also that the prior distribution of $\lambda = 1/\beta$ is $G(2.25, 0.01)$.

(i) What is the posterior distribution of λ, given $T = 150$ [hr]?
(ii) What is the Bayes estimator of β, for the squared-error loss?
(iii) What is the posterior SD of β?

Solution 10.1 **(i)** Following Example 10.3, the posterior distribution of λ is

$$G\left(\nu + 1, \frac{\tau}{1 + x\tau}\right) = G\left(3.25, \frac{0.01}{1 + 150 \times 0.01}\right) = G\left(3.25, 0.004\right)$$

The expected value of λ based on the prior distribution is $\frac{1}{\nu\tau} = \frac{1}{2.25 \times 0.01} = 44.4$. With the updated posterior distribution, the expected value of λ increases to $\frac{1}{3.25 \times 0.004} = 76.9$.

(ii) The Bayes estimator for β is the expectation value of the gamma distribution. Given $G(3.25, 0.004)$, The posterior Bayes estimator is therefore $\nu\tau = 3.25 \times 0.004 = 0.013$.

(iii) The posterior SD of β is given by the standard deviation of the posterior gamma distribution. $SD = \nu\tau^2 = 0.000052$. The prior SD of β was larger, 0.000225.

Exercise 10.2 Let $J(t)$ denote the number of failures of a device in the time interval $(0, t]$. After each failure the device is instantaneously renewed. Let $J(t)$ have a Poisson distribution with mean λt. Suppose that λ has a gamma prior distribution, with parameters $\nu = 2$ and $\tau = 0.05$.

(i) What is the predictive distribution of $J(t)$?
(ii) Given that $J(t)/t = 10$, how many failures are expected in the next time unit?
(iii) What is the Bayes estimator of λ, for the squared-error loss?
(iv) What is the posterior SD of λ?

Solution 10.2 (i) The predictive distribution of $J(t)$ is, $P(j; \lambda t)$ with λ having a gamma distribution $G(2, 0.05)$.

(ii) Given $x = 10$, the posterior distribution of λ is

$$G\left(\nu + x, \frac{\tau}{1+\tau}\right) = G(12, 0.0476)$$

The posterior expectation for λ is therefore $0.571 \times t$. This is increased from the prior expectation of $0.1 \times t$.

(iii) The Bayes estimator of λ is the mean of the gamma distribution. $\nu\tau = 0.571$.

(iv) The posterior SD of λ is $\nu\tau^2 = 0.0272$.

Exercise 10.3 The proportion of defectives, θ, in a production process has a uniform prior distribution on $(0, 1)$. A random sample of $n = 10$ items from this process yields $K_{10} = 3$ defectives.

(i) What is the posterior distribution of θ?
(ii) What is the Bayes estimator of θ for the absolute error loss?

Solution 10.3 (i) The uniform prior distribution on $(0, 1)$ can be expressed as the beta distribution $B(1, 1)$. With the $K_{10} = 3$ defectives, the posterior distribution is $B(1 + 3, 1 + 10 - 3) = B(4, 8)$.

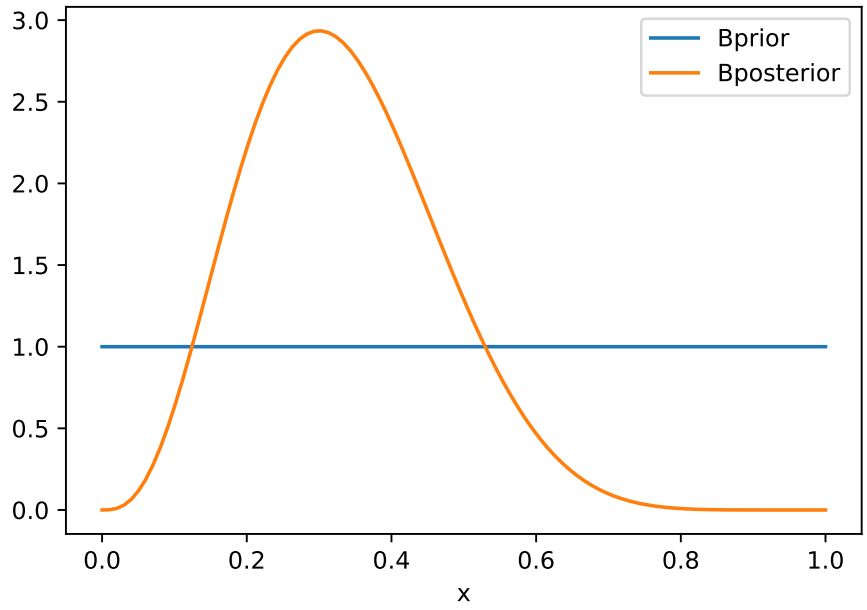
(ii) The Bayes estimator of θ can be calculated using equation (10.2.2).

$$\hat{\theta} = \frac{\nu_1 F_{0.5}[2\nu_1, 2\nu_2]}{\nu_2 + \nu_1 F_{0.5}[2\nu_1, 2\nu_2]} = \frac{4F_{0.5}[8, 16]}{8 + 4F_{0.5}[8, 16]} \approx \frac{4 \times 0.94422}{8 + 4 \times 0.94422} \approx 0.3207$$

Visualization of results:

```
x = np.linspace(0, 1, 100)
df = pd.DataFrame({'x': x, 'Bprior': stats.beta(1,1).pdf(x)})
df['Bposterior'] = stats.beta(1+3,1 + 10 - 3).pdf(x)

ax = df.plot(x='x', y='Bprior')
df.plot(x='x', y='Bposterior', ax=ax)
plt.show()
```



Exercise 10.4 Let $X \sim \mathcal{P}(\lambda)$ and suppose that λ has the Jeffrey improper prior $h(\lambda) = \frac{1}{\sqrt{\lambda}}$. Find the Bayes estimator for squared-error loss and its posterior SD.

Solution 10.4 The p.d.f. of X is $P(\lambda)$. The prior distribution of λ is $h(\lambda) = \frac{1}{\sqrt{\lambda}}$. Therefore, the posterior PDF $h(\lambda|x)$ is

$$\begin{aligned} h(\lambda|x) &= \frac{P(x;\lambda)h(\lambda)}{\int_0^\infty P(x;\lambda)h(\lambda)\mathrm{d}\lambda} \\ &= \frac{\frac{e^{-\lambda}\lambda^x}{x!}\frac{1}{\sqrt{\lambda}}}{\int_0^\infty \frac{e^{-\lambda}\lambda^x}{x!}\frac{1}{\sqrt{\lambda}}\mathrm{d}\lambda} = \frac{e^{-\lambda}\lambda^x\frac{1}{\sqrt{\lambda}}}{\int_0^\infty e^{-\lambda}\lambda^x\frac{1}{\sqrt{\lambda}}\mathrm{d}\lambda} = \frac{e^{-\lambda}\lambda^{x-\frac{1}{2}}}{\int_0^\infty e^{-\lambda}\lambda^{x-\frac{1}{2}}\mathrm{d}\lambda} \\ &= \frac{e^{-\lambda}\lambda^{x-\frac{1}{2}}}{\Gamma(x+\frac{1}{2})} = \frac{1}{\Gamma(x+\frac{1}{2})}\lambda^{x-\frac{1}{2}}e^{-\lambda} \\ &= G\left(x+\frac{1}{2},1\right) \end{aligned}$$

The posterior distribution is the gamma distribution $G(x+\frac{1}{2},1)$.

The posterior Bayes estimator for λ is $x+\frac{1}{2}$. The posterior SD is also $x+\frac{1}{2}$.

Exercise 10.5 Apply formula (10.2.3) to determine the Bayes estimator of the reliability when $n = 50$ and $K_{50} = 49$.

Solution 10.5 Using formula (10.2.3), we get:

```
n = 50; Kn = 49
medFdist = stats.f.ppf(0.5, 2*Kn+2, 2*n+2-2*Kn)
R_t = (Kn+1) * medFdist / (n+1 -Kn+(Kn+1)*medFdist)
R_t
```

```
0.9673091500837799
```

Exercise 10.6 A system has three modules, M_1, M_2, M_3. M_1 and M_2 are connected in series and these two are connected in parallel to M_3, i.e.,

$$R_{sys} = \psi_p(R_3, \psi_s(R_1, R_2)) = R_3 + R_1R_2 - R_1R_2R_3,$$

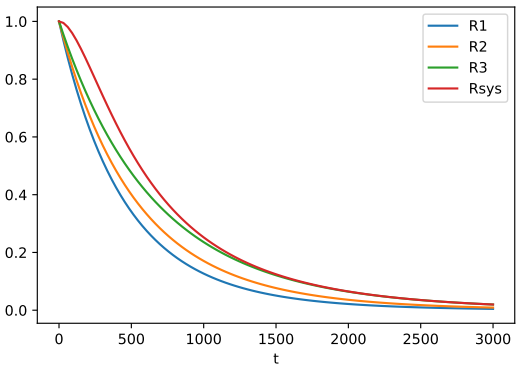
where R_i is the reliability of module M_i. The TTFs of the three modules are independent random variables having exponential distributions with prior $\text{IG}(\nu_i, \tau_i)$ distributions of their MTTF. Moreover, $\nu_1 = 2.5$, $\nu_2 = 2.75$, $\nu_3 = 3$, $\tau_1 = \tau_2 = \tau_3 = 1/1000$. In separate independent trials of the TTF of each module we obtained the statistics $T_n^{(1)} = 4565$ [hr], $T_n^{(2)} = 5720$ [hr] and $T_n^{(3)} = 7505$ [hr], where in all three experiments $n = r = 10$. Determine the Bayes estimator of R_{sys}, for the squared-error loss.

Solution 10.6 We get for R_{sys}:

```
v1 = 2.5; v2 = 2.75; v3 = 3; n=r=10
tau = 1/1000
T1 = 4565; T2 = 5720; T3 = 7505
def R(t, Ti, tau, r, v):
    return ((1 + Ti*tau)/(1+(Ti + t)*tau))**(r+v)
t = np.linspace(0, 3000, 100)
df = pd.DataFrame({
    't': t,
    'R1': R(t, T1, tau, r, v1),
    'R2': R(t, T2, tau, r, v2),
    'R3': R(t, T3, tau, r, v3),
})
df['Rsys'] = df['R3'] + df['R1']*df['R2'] - df['R1']*df['R2']*df['R3']
```

Visualization of R_{sys} over time:

```
df.plot(x='t')
plt.show()
```



Exercise 10.7 $n = 30$ computer monitors were put on test at a temperature of 100°F and relative humidity of 90% for 240 [hr]. The number of monitors which survived this test is $K_{30} = 28$. Determine the Bayes credibility interval for $R(240)$, at level $\gamma = 0.95$, with respect to a uniform prior on $(0, 1)$.

Solution 10.7 Using equation (10.3.1), we get:

```
n = 30
Kn = 28
gamma = 0.95

v1 = Kn + 1; v2 = n - Kn + 1
eps1 = (1-gamma)/2; eps2 = (1+gamma)/2

F_ll = stats.f.ppf(eps2, 2*n+2-2*Kn, 2*Kn+2)
F_ul = stats.f.ppf(eps2, 2*Kn+2, 2*n+2-2*Kn)
print(f'F-distribution median: ({F_ll:.3f}, {F_ul:.3f})')
F_50 = stats.f.ppf(0.5, 2*Kn+2, 2*n+2-2*Kn)

R_t = (Kn+1)*F_50 / ((n+1-Kn) + (Kn+1)*F_50)
print(f'R(t): {R_t:.3f}')
LL = (Kn+1) / ((Kn+1) + (n-Kn+1)*F_ll)
UL = (Kn+1)*F_ul / ((n-Kn+1)+(Kn+1)*F_ul)
print(f'Credibility limits: ({LL:.3f}, {UL:.3f})')
```

```
F-distribution median: (2.635, 4.963)
R(t): 0.915
Credibility limits: (0.786, 0.980)
```

Exercise 10.8 Determine a $\gamma = 0.95$ level credibility interval for $R(t)$ at $t = 25$ [hr] when TTF $\sim E(\beta)$, $\beta \sim$ IG$(3, 0.01)$, $r = 27$, $T_{n,r} = 3500$ [hr].

Solution 10.8 Using the results of Sect. 10.3.2, we get:

```
gamma = 0.95
t = 25
nu = 3
tau = 0.01
r = 27
Tnr = 3500
```

```
eps1 = (1-gamma)/2; eps2 = (1+gamma)/2

beta_L = (Tnr + 1/tau) / stats.gamma.ppf(eps2, nu+r, 1)
beta_U = (Tnr + 1/tau) / stats.gamma.ppf(eps1, nu+r, 1)
print(f'Credibility limits beta: ({beta_L:.2f}, {beta_U:.2f})')

RL = np.exp(-t / beta_L)
RU = np.exp(-t / beta_U)
RL, RU
print(f'Credibility limits R(50): ({RL:.3f}, {RU:.3f})')
```

```
Credibility limits beta: (84.41, 169.48)
Credibility limits R(50): (0.744, 0.863)
```

Exercise 10.9 Under the conditions of Exercise 10.8 determine a Bayes prediction interval for the total life of $s = 2$ devices.

Solution 10.9 Using the results from Sect. 10.3.3, we get:

```
s = 2
eps1 = (1-gamma)/2; eps2 = (1+gamma)/2

# using beta
T_L = (Tnr + 1/tau)*(1/stats.beta.ppf(eps2, nu+r, s) - 1)
T_U = (Tnr + 1/tau)*(1/stats.beta.ppf(eps1, nu+r, s) - 1)
print(f'Prediction interval: ({T_L:.3f}, {T_U:.3f})')

# using F
T_L = (Tnr + 1/tau)*(s / (nu+r)) * stats.f.ppf(eps1, 2*s, 2*nu+2*r)
T_U = (Tnr + 1/tau)*(s / (nu+r)) * stats.f.ppf(eps2, 2*s, 2*nu+2*r)
print(f'Prediction interval: ({T_L:.3f}, {T_U:.3f})')
```

```
Prediction interval: (28.707, 721.838)
Prediction interval: (28.707, 721.838)
```

We have high confidence that the two devices will last for at least 1636 hours.

Exercise 10.10 A repairable system has exponential TTF and exponential TTR, which are independent of each other. $n = 100$ renewal cycles were observed. The total times till failure were 10,050 [hr] and the total repair times were 500 [min]. Assuming gamma prior distributions for λ and μ with $\nu = \omega = 4$ and $\tau = 0.0004$ [hr], $\zeta = 0.01$ [min], find a $\gamma = 0.95$ level credibility interval for A_∞.

Solution 10.10 Using the result from Sect. 10.4, we get:

```
n = 100
U = 10_050  # sum(TTF)
V = 500 / 60 # sum(TTR)

lambda_ = 1/U  # ~ G(nu, tau)
mu = 1/V # ~ G(omega, zeta)
A_infty = mu / (mu + lambda_)
print(f'A_inf: {A_infty:.5f}')

# prior distributions
nu = 4; tau = 0.0004  # lambda~G(2, 0.001)
omega = 4; zeta = 0.01 / 60  # mu~G(2, 0.005)

# Crediblity interval
```

```
gamma = 0.95
eps1 = (1-gamma)/2; eps2 = (1+gamma)/2

A_infty_eps1 = 1/(1 + (V+zeta)/(U+tau) *
                   stats.beta.ppf(eps2, n+nu, n+omega)/stats.beta.ppf(eps1, n+omega, n+nu))
A_infty_eps2 = 1/(1 + (V+zeta)/(U+tau) *
                   stats.beta.ppf(eps1, n+nu, n+omega)/stats.beta.ppf(eps2, n+omega, n+nu))
print(f'A_inf_eps1: {A_infty_eps1:.5f}')
print(f'A_inf_eps2: {A_infty_eps2:.5f}')
```

```
A_inf: 0.99917
A_inf_eps1: 0.99891
A_inf_eps2: 0.99937
```

Note that we converted the total repair time and ζ from [min] to [h] here.

Exercise 10.11 In reference to Example 10.6, suppose that the data of Table 10.2 were obtained for a Poisson random variable where $\lambda_1, \cdots, \lambda_{188}$ have a gamma (ν, τ) prior distribution.

(i) What is the predictive distribution of the number of defects per batch?
(ii) Find the formulae for the first two moments of the predictive distribution.
(iii) Find, from the empirical frequency distribution of Table 10.2, the first two sample moments.
(iv) Use the method of moment equations to estimate the prior parameters ν and τ.
(v) What is the Bayes estimator of λ if $X_{189} = 8$?

Solution 10.11 (i) Suppose that $T \ P(\lambda)$ and λ has a prior Gamma distribution $G(\nu, \tau)$. We get,

$$E_{\nu,\tau} = \nu\tau$$
$$V_{\nu,\tau} = \nu\tau^2$$

The first two moments of the distribution are

$$M_{1,n} = \frac{1}{n}\sum_{i=1}^{n} t_i = \hat{\nu}\hat{\tau}$$
$$M_{2,n} = \frac{1}{n}\sum_{i=1}^{n} t_i^2 = \hat{\nu}\hat{\tau}^2$$

This gives:

$$\hat{\tau} = \frac{M_{2,n}}{M_{1,n}}$$
$$\hat{\nu} = \frac{M_{1,n}}{\hat{\tau}} = \frac{M_{1,n}^2}{M_{2,n}}$$

(iii) Using the empirical frequency distribution, we get for the sample moments:

```
freq_dist = {
  0: 4, 1: 21, 2: 29, 3: 32, 4: 19, 5: 14,
  6: 13, 7: 5, 8: 8, 9: 5, 10: 9, 11: 1,
  12: 2, 13: 4, 14: 4, 15: 1, 16: 4, 17: 2,
  18: 1, 19: 1, 20: 1, 21: 1, 22: 1, 23: 2,
  24: 1, 25: 2, 26: 1}

n = sum(count for value, count in freq_dist.items())
M_1 = sum(value * count for value, count in freq_dist.items()) / n
M_2 = sum(value * value * count for value, count in freq_dist.items()) / n
print(n, M_1, M_2)
```

```
188 6.037234042553192 68.48404255319149
```

(iv) The estimated prior parameters are, $\hat{\tau} = 11.34$ and $\hat{\nu} = 0.53$ resulting in $\hat{\lambda} = 6.037$.

```
tau = M_2 / M_1
nu = M_1*M_1 / M_2
lambda_ = nu * tau
print(tau, nu, lambda_)
```

```
11.343612334801762 0.5322144185085728 6.037234042553193
```

(v) After observation of an additional $X_{189} = 8$, we get:

```
freq_dist[8] += 1
n = sum(count for value, count in freq_dist.items())
M_1 = sum(value * count for value, count in freq_dist.items()) / n
M_2 = sum(value * value * count for value, count in freq_dist.items()) / n

tau = M_2 / M_1
nu = M_1*M_1 / M_2
lambda_ = nu * tau
print(tau, nu, lambda_)
```

```
11.32020997375328 0.5342320559107019 6.0476190476190474
```

With the additional observation, $\hat{\lambda} = 6.048$.

Chapter 11
Sampling Plans for Batch and Sequential Inspection

For the most recent version of the solution manual, go to `https://gedeck.github.io/mistat-code-solutions/IndustrialStatistics/`.

Import required modules and define required functions

```
import numpy as np
import pandas as pd
from scipy import stats, optimize
import statsmodels.formula.api as smf
import lifelines
import pingouin as pg
import seaborn as sns
import matplotlib.pyplot as plt
import mistat
from mistat import acceptanceSampling
```

Exercise 11.1 Determine single sampling plans for attributes, when the lot is $N = 2500$, $\alpha = \beta = 0.01$, and

(i) $AQL = 0.005$, $LQL = 0.01$
(ii) $AQL = 0.01$, $LQL = 0.03$
(iii) $AQL = 0.01$, $LQL = 0.05$

Solution 11.1 The single-sampling plans for attributes with $N = 2500$, $\alpha = \beta = 0.01$, and AQL and LQL as specified are (i) $n = 1878$, $c = 13$; (ii) $n = 702$, $c = 12$; (iii) $n = 305$, $c = 7$.

Exercise 11.2 Investigate how the lot size, N, influences the single sampling plans for attributes, when $\alpha = \beta = 0.05$, $AQL = 0.01$, $LQL = 0.03$, by computing the plans for $N = 100$, $N = 500$, $N = 1000$, $N = 2000$.

Solution 11.2 For $\alpha = \beta = 0.05$, $AQL = 0.01$, and $LQL = 0.03$ the single-sampling plans for attributes are

N	n	c
100	87	1
500	254	4
1000	355	6
2000	453	8

We see that as the lot size, N, increases then the required sample size increases, but n/N decreases from 87% to 22.6%. The acceptance number c increases very slowly.

Exercise 11.3 Compute the $OC(p)$ function for the sampling plan computed in Exercise 11.1(iii). What is the probability of accepting a lot having 2.5% of nonconforming items?

Solution 11.3 For the sampling plan in Exercise 11.1(iii), $OC(p) = H(7; 2500, M_p, 305)$. When $p = 0.025$, we get $M_p = 62$ and $OC(0.025) = 0.5091$.

Exercise 11.4 Compute the large sample approximation to a single sample plan for attributes (n^*, c^*), with $\alpha = \beta = 0.05$ and $AQL = 0.025$, $LQL = 0.06$. Compare these to the exact results. The lot size is $N = 2000$.

Solution 11.4 The large sample approximation yields $n^* = 292, c^* = 11$. The "exact" plan is $n = 311, c = 12$. Notice that the actual risks of the large sample approximation plan (n^*, c^*) are $\alpha^* = 0.0443$ and $\beta^* = 0.0543$. The actual risks of the "exact" plan are $\alpha = 0.037$ and $\beta = 0.0494$.

Exercise 11.5 Repeat the previous Exercise with $N = 3000$, $\alpha = \beta = 0.10$, $AQL = 0.01$ and $LQL = 0.06$.

Solution 11.5 The large sample approximation is $n^* = 73$, $c^* = 1$ with actual risks of $\alpha^* = 0.16$ and $\beta^* = 0.06$. The exact plan is $n = 87$, $c = 2$ with actual risks of $\alpha = 0.054$ and $\beta = 0.097$.

Exercise 11.6 Obtain the OC and ASN functions of the double sampling plan, with $n_1 = 200$, $n_2 = 2n_1$ and $c_1 = 5$, $c_2 = c_3 = 15$, when $N = 2000$.

(i) What are the attained α and β when $AQL = 0.015$ and $LQL = 0.05$?
(ii) What is the ASN when $p = AQL$?
(iii) What is a single sampling plan having the same α and β? How many observations we expect to save if $p = AQL$? Notice that if $p = LQL$ the present double sampling plan is less efficient than the corresponding single sampling plan.

Solution 11.6 (i) The attained α and β are $\alpha = 0.021$, $\beta = 0.0532$. (ii) ASN = 228.7. (iii) The single-sampling plan is $n = 253$, $c = 7$ with $\alpha^* = 0.0283$, $\beta^* = 0.0486$. If $p = AQL$ we expect to save 24 observations.

Exercise 11.7 Compute the OC and ASN values for a double sampling plan with $n_1 = 150$, $n_2 = 200$, $c_1 = 5$, $c_2 = c_3 = 10$, when $N = 2000$. Notice how high β is when $LQL = 0.05$. The present plan is reasonable if $LQL = 0.06$. Compare this plan to a single sampling one for $\alpha = 0.02$ and $\beta = 0.10$, $AQL = 0.02$, $LQL = 0.06$.

Solution 11.7 The double-sampling plan with $n_1 = 150$, $n_2 = 200$, $c_1 = 5$, $c_2 = c_3 = 10$, and $N = 2000$ yields

p	OC(p)	ASN(p)	p	OC(p)	ASN(p)
0	1.000	150	0.06	0.099	247.7
0.01	0.999	150.5	0.07	0.039	218.9
0.02	0.966	164.7	0.08	0.014	191.5
0.03	0.755	206.0	0.09	0.005	171.9
0.04	0.454	248.4	0.10	0.001	160.4
0.05	0.227	262.8			

The single sampling plan for $\alpha = 0.02$, $\beta = 0.10$, AQL= 0.02, LQL= 0.06 is $n = 210$, $c = 8$ with actual $\alpha = 0.02$, $\beta = 0.10$.

Exercise 11.8 Determine a sequential plan for the case of AQL = 0.02, LQL = 0.06, $\alpha = \beta = 0.05$. Compute the OC and ASN functions of this plan. What are the ASN values when p = AQL, p = LQL and $p = 0.035$?

Solution 11.8 For the sequential plan, $OC(0.02) = 0.95 = 1 - \alpha$, $OC(0.06) = 0.05 = \beta$. ASN(0.02) = 140, ASN(0.06) = 99 and ASN(0.035) = 191.

Exercise 11.9 Compare the single sampling plan and the sequential one when AQL = 0.01, LQL = 0.05, $\alpha = \beta = 0.01$ and $N = 10,000$. What are the expected savings in sampling cost, if each observation costs \$1, and p = AQL?

Solution 11.9 The single-sampling plan for $N = 10,000$, $\alpha = \beta = 0.01$, $AQL = 0.01$ and $LQL = 0.05$ is $n = 341$, $c = 8$. The actual risks are $\alpha = 0.007$, $\beta = 0.0097$. The corresponding sequential plan, for $p = AQL = 0.01$ has $ASN(0.01) = 182$. On the average, the sequential plan saves, under $p = AQL$, 159 observations. This is an average savings of \$159 per inspection.

Exercise 11.10 Use the `mistat` function *simulateOAB* to simulate the expected rewards, for $p = 0.4(0.05)0.8$, when $N = 75$, $\lambda = 0.6$, $k = 15$, $\gamma = 0.95$, $Ns = 1000$.

Solution 11.10 In Python:

```
np.random.seed(1)

N=75; lambda_=0.6; k0=15; gamma=0.95; Ns=1000

results = []
for p in (0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8):
  r = acceptanceSampling.simulateOAB(N, p, lambda_, k0, gamma, Ns)
  results.append({
      'p': p,
      'Mgamma_mean': r.mgamma.mean,
      'Mgamma_std': r.mgamma.std,
      'Reward_mean': r.reward.mean,
      'Reward_std': r.reward.std,
  })
pd.DataFrame(results)
```

```
      p  Mgamma_mean  Mgamma_std  Reward_mean  Reward_std
0  0.40       22.639   11.931080      40.4016    0.904609
1  0.45       31.382   19.866758      40.3588    1.222515
2  0.50       42.608   24.492822      40.8492    2.075587
3  0.55       57.779   23.582794      42.6366    3.416241
4  0.60       66.750   18.822526      45.4380    4.143640
5  0.65       71.662   12.721154      49.1878    4.508895
6  0.70       73.924    7.576953      52.9706    4.282127
7  0.75       74.762    3.755710      57.0788    3.971714
8  0.80       74.945    1.738383      60.7050    3.518803
```

Exercise 11.11 Use the `mistat` function *optimalOAB* to predict the expected reward under the optimal strategy, when $N = 75$, $\lambda = 0.6$.

Solution 11.11 In Python:

```
result = acceptanceSampling.optimalOAB(75, 0.6)
print(f'Case (75, 0.6): {result.max_reward:.3f}')
```

```
Case (75, 0.6): 62.447
```

Exercise 11.12 Consider the two-armed bandit (TAB) with $N = 40$ and $K = 10$. Make a table of all the possible predicted rewards.

Solution 11.12 We first define a function that calculates the expected reward for the TAB strategy under the assumption of a beta(1, 1) prior.

```
def TAB(N, k, Xk):
    # case: no reward for the first k // 2 trials - switch
    if Xk == 0:
        # calculate posterior based on no successes in k//2 trials
        m = k//2
        p_post = (1 + 0) / (1 + 1 + m)
        return acceptanceSampling.optimalOAB(N - m, p_post).max_reward

    # calculate posterior based on Xk successes in k trials
    p_post = (1 + Xk) / (1 + 1 + k)
    # case: only successes in trials stay in arm A
    if Xk == k:
        return k + (N-k) * p_post
    # switch to arm B and use posterior propability for
    # optimal OAB strategy
    return Xk + acceptanceSampling.optimalOAB(N-k, p_post).max_reward
```

Using the function we get the following expected rewards as a function of $X10$

```
pd.DataFrame({
    'X10': list(range(0, 11)),
    'Expected reward': [TAB(40, 10, Xk) for Xk in range(0, 11)],
})
```

```
    X10  Expected reward
0     0        26.331820
1     1        23.597716
2     2        24.782446
3     3        26.078104
4     4        27.487403
```

```
5     5        29.020577
6     6        30.662946
7     7        32.475568
8     8        34.380689
9     9        36.500000
10   10        37.500000
```

Exercise 11.13 Determine n and k for a continuous variable size sampling plan, when $(p_0) = AQL = 0.01$ and $(p_t) = LQL = 0.05$, $\alpha = \beta = 0.05$.

Solution 11.13 For a continuous variable-size sampling plan when (p_0) = AQL = 0.01, (p_t) = LQL = 0.05, and $\alpha = \beta = 0.05$, we obtain $n = 70$ and $k = 1986$.

Exercise 11.14 Consider dataset **ALMPIN.csv**. An aluminum pin is considered as defective if its cap diameter is smaller than 14.9 [mm]. For the parameters $p_0 = 0.01$, $\alpha = 0.05$, compute k and decide whether to accept or reject the lot, on the basis of the sample of $n = 70$ pins. What is the probability of accepting a lot with proportion defectives of $p = 0.03$?

Solution 11.14 From the data we get $\bar{X} = 14.9846$ and $S = 0.019011$. For $p_0 = 0.01$ and $\alpha = 0.05$ we obtain $k = 2.742375$. Since $\bar{X} - kS = 14.9325 > \zeta = 14.9$, the lot is *accepted*. $OC(0.03) \approx 0.4812$.

Exercise 11.15 Determine the sample size and k for a single sampling plan by a normal variable, with the parameters $AQL = 0.02$, $LQL = 0.04$, $\alpha = \beta = 0.10$.

Solution 11.15 For $AQL = 0.02$, $LQL = 0.04$ and $\alpha = \beta = 0.10$, $n = 201$, $k = 1.9022$.

Exercise 11.16 A single sampling plan for attributes, from a lot of size $N = 500$, is given by $n = 139$ and $c = 3$. Each lot that is not accepted is rectified. Compute the AOQ, when $p = 0.01$, $p = 0.02$, $p = 0.03$ and $p = 0.05$. What are the corresponding ATI values?

Solution 11.16 For a single-sampling plan for attributes where $n = 139$, $c = 3$ and $N = 500$ we obtain $OC(p) = H(3; 500, M_p, 139)$, and $R^* = H(2; 499, M_p - 1, 138)/H(3; 500, M_p, 139)$.

p	AOQ	ATI
0.01	0.0072	147.2
0.02	0.0112	244.2
0.03	0.0091	369.3
0.05	0.0022	482.0

Exercise 11.17 A single sampling plan, under normal inspection has probability $\alpha = 0.05$ of rejection, when $p = AQL$. What is the probability, when $p = AQL$ in 5 consecutive lots, that there will be a switch to tightened inspection? What is the probability of switching to a tightened inspection if p increases so that $OC(p) = 0.7$?

Solution 11.17 A switch to a tightened plan, when all 5 consecutive lots have $p = AQL$, with $\alpha = 0.05$, is $\sum_{j=2}^{5} b(j; 5, 0.05) = 0.0226$. The probability of switching to a tightened plan if $\alpha = 0.3$ is 0.4718.

Exercise 11.18 Compute the probability for qualifying for State 2, in a skip-lot sampling plan, when $n = 100$, $c = 1$. What is the upper bound on S_{10}, in order to qualify for State 2, when $AQL = 0.01$? Compute the probability QP for State 2 qualification.

Solution 11.18 The total sample size from 10 consecutive lots is 1000. Thus, from Table 11.15 S_{10} should be less than 5 (note that the tables shows %AQL). The last 2 samples should each have less than 2 defective items. Hence, the probability for qualification is

$$\begin{aligned} QP &= b^2(1; 100, 0.01)B(2; 800, 0.01) \\ &\quad + 2b(1; 100, 0.01)b(0; 100, 0.01)B(3; 800, 0.01) \\ &\quad + b^2(0; 100, 0.01)B(4; 800, 0.01) \\ &= 0.0263. \end{aligned}$$

Exercise 11.19 The **FAILURE_J2** dataset contains the cumulative failure counts of a software project collected over a period of 181 weeks. Fit the following models to the data and visualize the results.

- Goel-Okumoto $f(t) = a[1 - \exp(-bt)]$
- Musa-Okumoto $f(t) = \frac{1}{\varphi} \log(\lambda_0 \varphi t + 1)$
- S-shaped Yamada $f(t) = a(1 - (1 + bt) \exp(-bt))$
- inflected S-shaped Ohba $f(t) = \frac{a(1-\exp(-bt))}{1+c \exp(-bt)}$

Which model describes the data best?

Solution 11.19 Define function for the four models

```
def modelGoelOkumoto(t, a, b):
    return a * (1 - np.exp(-b * t))

def modelMusaOkumoto(t, phi, lam):
    return (1/phi) * np.log(lam*phi* t + 1)

def modelYamada(t, a, b):
    return a * (1 - (1+b*t)*np.exp(-b*t))

def modelInflectedSshaped(t, a, b, c):
    return a * (1 - np.exp(-b * t)) / (1 + c * np.exp(-b * t))
```

Fit the four functions to the cumulative failure count data `CFC`

```
def optimizeModelFit(model, data, x, y):
    fit = optimize.curve_fit(model, data[x], data[y], method='lm')
    popt = fit[0]
    # add the fit curve to the data
    data[model.__name__] = [model(t, *popt) for t in data[x]]
```

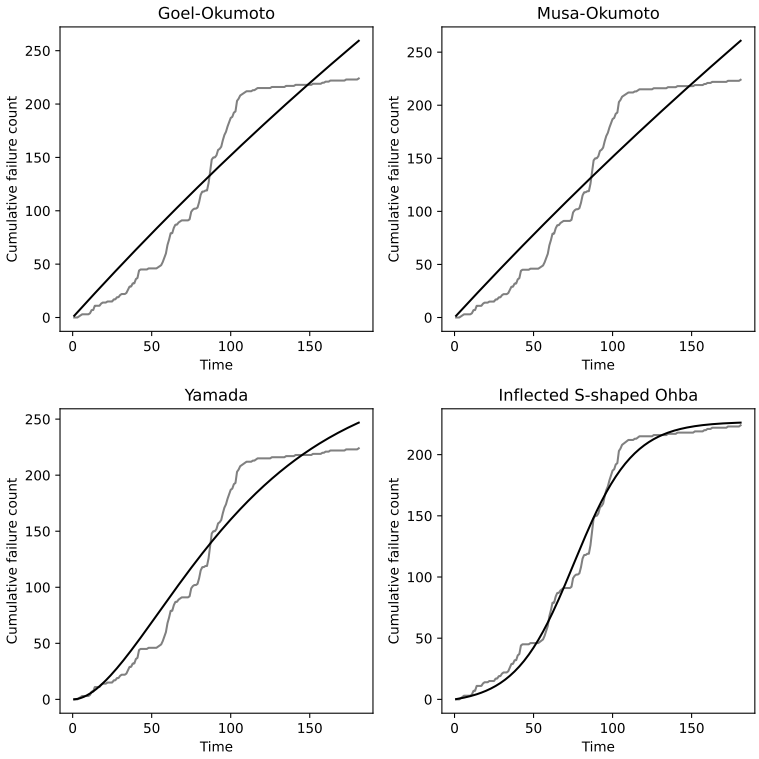


Fig. 11.1: NHPP models fitted to cumulative failure count data **FAILURE_J2**

```
    return popt
data = mistat.load_data('FAILURE_J2')
optimizeModelFit(modelGoelOkumoto, data, 'T', 'CFC')
optimizeModelFit(modelMusaOkumoto, data, 'T', 'CFC')
optimizeModelFit(modelYamada, data, 'T', 'CFC')
optimizeModelFit(modelInflectedSshaped, data, 'T', 'CFC')
```

```
array([2.26933527e+02, 5.40121092e-02, 5.96246327e+01])
```

The fits are shown in Fig. 11.1. The inflected S-shaped model describes the data best.

Exercise 11.20 Continuing with Exercise 11.19, simulate cases where you only have data for the first 25, 50, 75, 100, or 125 weeks and fit Goel-Okumoto and inflected S-shaped models to these subsets.

Discuss the results with respect to using the models to extrapolate the expected failure count to predict the expected failure count into the future.

Solution 11.20 Modify the *optimizeModelFit* function from the previous exercise to restrict the fit to a subset of the data. The

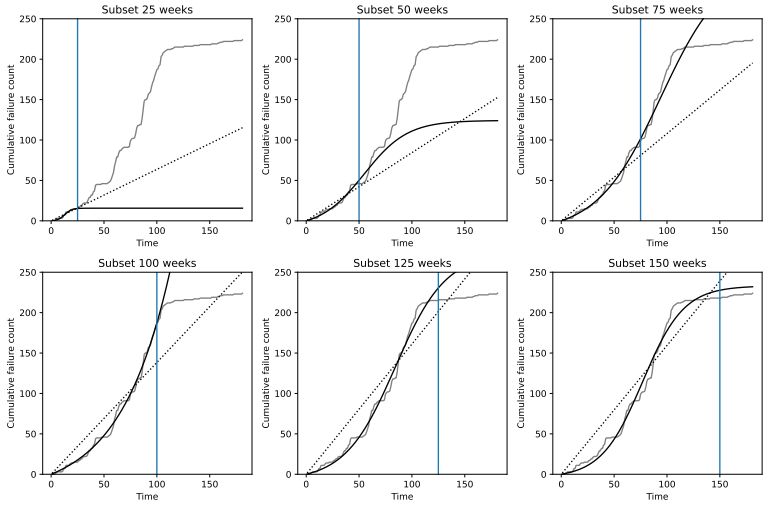


Fig. 11.2: NHPP models fitted to cumulative failure count data **FAILURE_J2** using subsets. The actual data are shown in grey, the Goel-Okumoto fits as dotted black lines, and the inflected S-shaped models as solid black lines. The vertical line shows the cutoff used for subsetting the data for the fits.

```
def optimizeModelFit(model, data, x, y, subset):
    # create the subset
    subsetX = data[x][:subset]
    subsetY = data[y][:subset]
    # fit curve to subset - a increase of maxfev is required
    fit = optimize.curve_fit(model, subsetX, subsetY,
                             method='lm', maxfev=2000)
    popt = fit[0]
    data[f'{model.__name__} {subset}'] = [model(t, *popt) for t in data[x]]
    return popt
for subset in [25, 50, 75, 100, 125, 150]:
    optimizeModelFit(modelGoelOkumoto, data, 'T', 'CFC', subset)
    optimizeModelFit(modelInflectedSshaped, data, 'T', 'CFC', subset)
```

The result of this simulation is shown in Fig. 11.2. As in the previous exercise, the inflected S-shaped model performs best. However, the fit using only 25 weeks of data is insufficient to extrapolate into the future. With 50 or 75 weeks of data, the inflected S-shaped model predicts the following 20-30 weeks well. Shortly after 100 weeks, the cumulative failure count curve flattens. This is not predicted by the model fit. Even with 125 weeks of data, this is not well described. Only at 150 weeks, the inflected S-shaped model starts to shows flattening.

Exercise 11.21 The dataset **FAILURE_DS2** contains cumulative failure counts of a software project collected over a period of 18 weeks. Fit an inflected S-shaped models to these data.

You will see that the large initial number of failures leads to an insufficient fit of the data. Create a second fit where the initial number of failures is ignored and discuss the results.

Solution 11.21 Load the data and fit the model

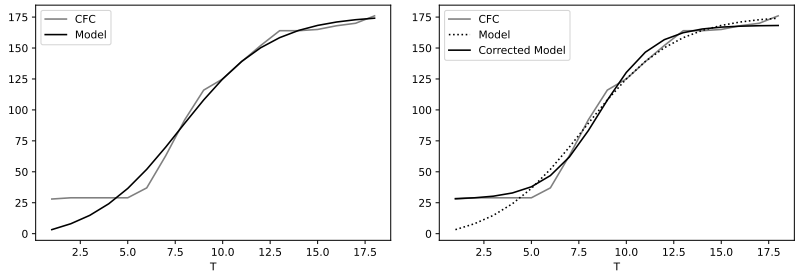


Fig. 11.3: NHPP models fitted to cumulative failure count data **FAILURE_DS2**. Left: initial model, right: after correction

```
data = mistat.load_data('FAILURE_DS2')
fit = optimize.curve_fit(modelInflectedSshaped, data['T'], data['CFC'])
popt = fit[0]

# add the fit curve to the data and visualize
data['Model'] = [modelInflectedSshaped(t, *popt) for t in data['T']]

ax = data.plot(x='T', y='CFC', color='grey')
data.plot(x='T', y='Model', color='black', ax=ax)
plt.show()
```

The result of this simulation is shown in Fig. 11.3. It is clear that the fit could be better if the model ignores the initial failure count. Define a new model to correct for this problem.

```
initial = data['CFC'][0]
def correctedModel(t, a, b, c):
    return modelInflectedSshaped(t, a, b, c) + initial

fit = optimize.curve_fit(correctedModel, data['T'], data['CFC'])
popt = fit[0]
# add the fit curve to the data and visualize
data['Corrected Model'] = [correctedModel(t, *popt) for t in data['T']]

ax = data.plot(x='T', y='CFC', color='grey')
data.plot(x='T', y='Model', color='black', ax=ax, linestyle=':')
data.plot(x='T', y='Corrected Model', color='black', ax=ax)
plt.show()
```

Including the offset in the model, leads to a better fit of the data.