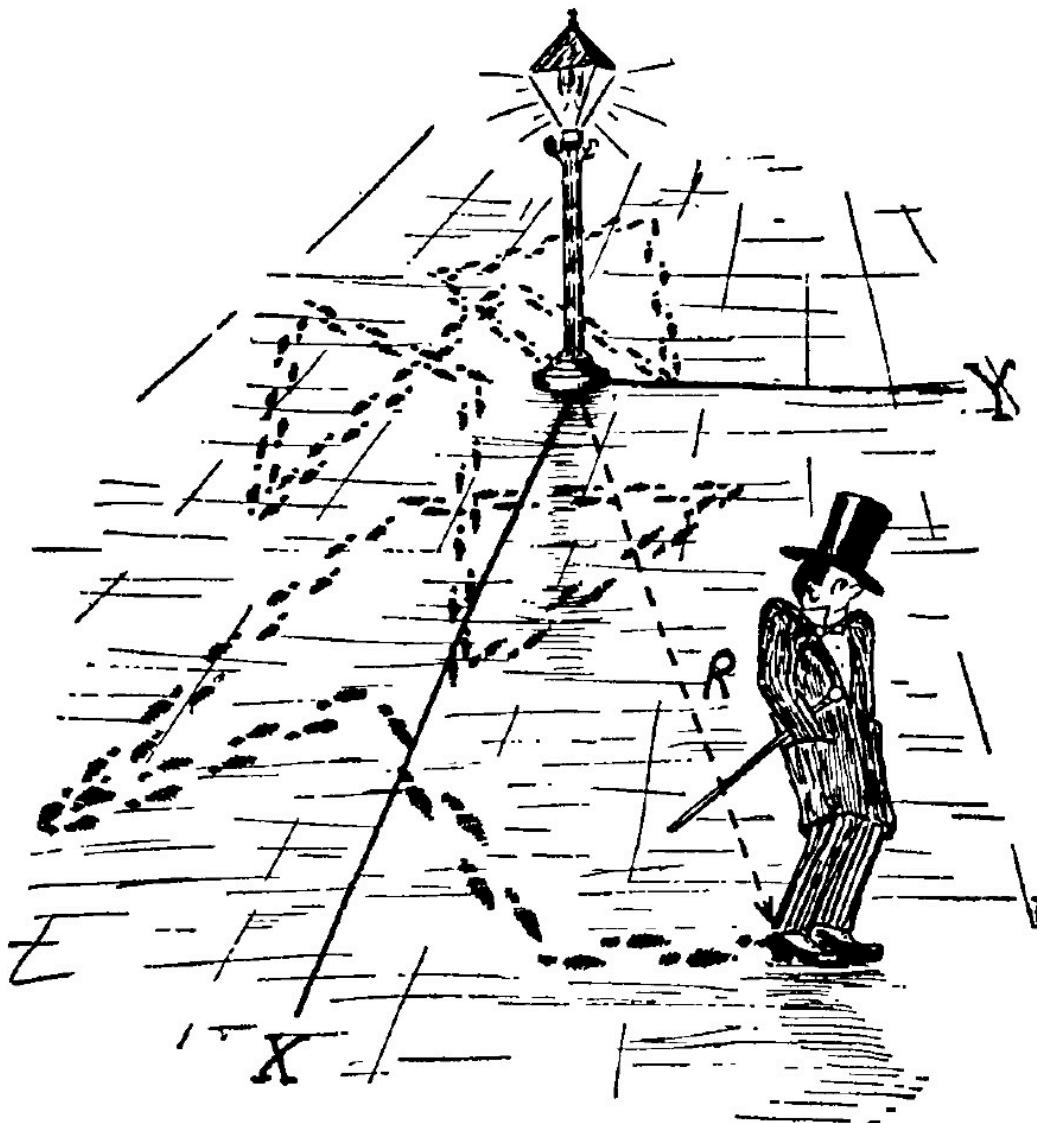


Mirko Birbaumer  
Klaus Frick  
Peter Büchel  
Simon van Hemert

# Predictive Modeling

## Lecture Notes



Hochschule Luzern - Technik und Architektur

# Contents

<b>1. Probability Models for Measurement Data</b>	<b>1</b>
1.1. Continuous Random Variables and Probability Distributions . . . . .	1
1.1.1. Probability Density . . . . .	5
1.1.2. Summary Statistics of Continuous Distributions . . . . .	7
1.2. Important Continuous Distributions . . . . .	9
1.2.1. Uniform Distribution . . . . .	9
1.2.2. Exponential Distribution . . . . .	11
1.2.3. Normal Distribution (Gaussian distribution) . . . . .	15
1.3. Functions of a Random Variable . . . . .	17
1.3.1. Linear Transformations of Random Variables . . . . .	18
1.3.2. Non-Linear Transformations of Random Variables . . . . .	23
1.4. Functions of Several Random Variables . . . . .	25
1.4.1. Independence and i.i.d. Assumption . . . . .	26
1.4.2. Summary Statistics of $S_n$ and $\bar{X}_n$ . . . . .	27
1.4.3. Distributions of $S_n$ and $\bar{X}_n$ . . . . .	29
<b>2. Statistics for Measurement Data</b>	<b>35</b>
2.1. Assess the Normal Distribution Assumption . . . . .	35
2.1.1. Q-Q Plot . . . . .	36
2.1.2. Normal Plot . . . . .	39
2.2. Parameter Estimation for Continuous Probability Distributions . . . . .	41
2.2.1. Method of Moments . . . . .	41
2.2.2. Method of Maximum Likelihood . . . . .	44
2.3. Statistical Tests and Confidence Intervals Normally Distributed Data	47
2.3.1. $z$ -Test ( $\sigma_X$ known) . . . . .	48
2.3.2. $t$ -Test ( $\sigma_X$ unknown) . . . . .	52
2.3.3. $P$ -Value . . . . .	58
2.3.4. Confidence Intervals for $\mu$ . . . . .	61
2.4. Statistical Significance and Relevance in Statistical Tests . . . . .	63
<b>3. Joint Distributions</b>	<b>65</b>
3.1. Joint, Marginal and Conditional Distributions . . . . .	65
3.1.1. Discrete Random Variables . . . . .	65
3.1.2. Continuous Random Variables . . . . .	68
3.2. Expected Value of Several Random Variables . . . . .	72
3.3. Covariance and Correlation . . . . .	73

## Contents

3.4. Bivariate Normal Distribution . . . . .	80
3.5. Principal Component Analysis . . . . .	81
<b>I. Regression Analysis</b>	<b>92</b>
<b>4. Introduction to Regression Analysis</b>	<b>93</b>
4.1. What is Regression? . . . . .	93
4.2. Why Estimate $f$ ? . . . . .	96
4.2.1. Prediction . . . . .	96
4.2.2. Inference with Respect to $f$ . . . . .	98
4.3. How do we Estimate $f$ ? . . . . .	99
<b>5. Simple Linear Regression</b>	<b>103</b>
5.1. Model for Simple Linear Regression . . . . .	103
5.1.1. Introduction . . . . .	103
5.1.2. Simple Linear Regression Model . . . . .	104
5.2. Estimating the Coefficients . . . . .	105
5.3. Hypothesis Tests and Confidence Intervals . . . . .	111
5.3.1. Assessing the Accuracy of the Coefficient Estimates . . . . .	111
5.3.2. Hypothesis Test . . . . .	116
5.3.3. Confidence Intervals for Regression Coefficients . . . . .	119
5.4. Confidence and Prediction Intervals for the Response . . . . .	120
5.4.1. Confidence Interval for the Response Variable . . . . .	120
5.4.2. Prediction Intervals . . . . .	124
<b>6. Testing Model Assumptions: Residual Analysis</b>	<b>129</b>
6.1. Assumptions Concerning the Error Term $\varepsilon$ . . . . .	129
6.1.1. Non-Linear Relationship . . . . .	134
6.1.2. Assessing the Validity of the Model Assumptions . . . . .	135
6.2. Diagnostics Instruments . . . . .	136
6.2.1. The Residual Standard Error and the $R^2$ Statistic . . . . .	136
6.2.2. Diagnostics Tool for Testing Model Assumption $E[\varepsilon]$ . . . . .	140
6.2.3. Diagnostics Tool for Testing Model Assumption $\text{Var}[\varepsilon_i]$ constant	149
6.2.4. Diagnostics Tool for Normal Distribution Assumption of Errors	153
6.2.5. Diagnostics Tool for Independence Assumption of Errors $\varepsilon_i$ .	158
6.2.6. Outliers and High Leverage Points . . . . .	161
6.2.7. Summary . . . . .	165
6.3. Treatment of Deficiency in Model Assumptions . . . . .	169
6.3.1. Therapeutic Treatment in the Case of $E[\varepsilon_i] \neq 0$ . . . . .	169
6.3.2. Therapeutic Treatment for $\text{Var}[\varepsilon_i] \neq \text{Constant}$ . . . . .	170
6.3.3. Therapeutic treatment for Non-Normally Distributed $\varepsilon_i$ .	171
6.3.4. Therapeutic Treatment for Outliers and High Leverage Points	171

## Contents

<b>7. Multiple Linear Regression</b>	<b>174</b>
7.1. Introduction . . . . .	174
7.2. Estimating the Regression Coefficients . . . . .	180
7.3. Some Important Questions . . . . .	183
7.3.1. Is There a Relationship Between the Response and Predictors? . . . . .	184
7.3.2. Deciding on Important Variables . . . . .	187
7.3.3. How well does the Model fit the Data? . . . . .	187
7.3.4. Predictions . . . . .	190
7.4. Variety of Regression Modeling . . . . .	192
7.4.1. Omitting Predictor Variables . . . . .	192
7.4.2. Qualitative Predictors . . . . .	196
7.4.3. Extensions of the Linear Model . . . . .	205
7.4.4. Model Assumptions in Multiple Linear Regression . . . . .	217
7.5. The Marketing Plan . . . . .	228
<b>8. Linear Model Selection</b>	<b>234</b>
8.1. Introduction . . . . .	234
8.2. Subset Selection . . . . .	235
8.2.1. Stepwise Selection . . . . .	235
8.2.2. Backward Stepwise Selection . . . . .	240
8.2.3. Further Variable Selection Methods . . . . .	244
8.2.4. Choosing the Optimal Model . . . . .	245
<b>II. Classification</b>	<b>256</b>
<b>9. Introduction to Classification Problems</b>	<b>257</b>
<b>10. Logistic Regression</b>	<b>263</b>
10.1. The Logistic Model . . . . .	264
10.2. Estimation of Regression Coefficients . . . . .	270
10.3. Predictions and Model Evaluation . . . . .	273
10.4. Cross Validation . . . . .	283
10.5. Multiple Logistic Regression . . . . .	285
<b>III. Time Series Analysis</b>	<b>290</b>
<b>11. Introduction to Time Series</b>	<b>291</b>
11.1. Examples . . . . .	292
11.2. Time Series with Python . . . . .	296
11.2.1. The <code>DatetimeIndex</code> class . . . . .	297
11.3. Basic Transformation, Visualization, and Decomposition of Time Series . . . . .	303
11.3.1. Data transformations . . . . .	303

## Contents

11.3.2. Visualizations . . . . .	307
11.3.3. Decomposition of time series . . . . .	311
<b>12. Mathematical Models for Time Series</b>	<b>320</b>
12.1. Mathematical Concept of Time Series . . . . .	320
12.2. Measures of Dependence . . . . .	327
12.3. Stationarity . . . . .	330
12.3.1. Testing Stationarity . . . . .	332
12.4. Estimation of Correlation . . . . .	335
12.4.1. Estimation of the Mean Sequence . . . . .	335
12.4.2. Estimation of the Autocovariance . . . . .	336
<b>13. Forecasting Time Series</b>	<b>341</b>
13.1. Autoregressive Models . . . . .	342
13.1.1. Definition and the Characteristic Polynomial . . . . .	342
13.1.2. Autocorrelation of AR( $p$ ) processes . . . . .	345
13.1.3. Model fitting . . . . .	351
13.1.4. Forecasting AR( $p$ ) processes . . . . .	354
<b>14. ARIMA</b>	<b>360</b>
14.0.1. Repetition of Autoregressive Models . . . . .	360
<b>15. Spectral Analysis</b>	<b>362</b>
15.1. Introduction . . . . .	362
15.2. Supplementary Note on DFT . . . . .	362
<b>A. Supplementary Material</b>	<b>366</b>
A.1. Boltzmann Distribution . . . . .	366
<b>B. R-Code</b>	<b>367</b>

# List of Tables

2.1.	Measurements of the concrete compression strength . . . . .	37
3.1.	Example <b>weather stations</b> . . . . .	65
7.1.	<b>Advertising</b> Data Set : Regression Coefficients . . . . .	176
7.2.	<b>Advertising</b> Data Set : Least Squares Estimates for Coefficients . .	181
7.3.	<b>Advertising</b> Data Set : Correlation Matrix . . . . .	182
7.4.	<b>Credit</b> Data Set : Regression Coefficients . . . . .	200
7.5.	<b>Credit</b> Data Set : Regression Coefficients . . . . .	204
7.6.	<b>Balance</b> Data Set : Regression Coefficients . . . . .	225
10.1.	Three cases extracted from the dataset <b>Default</b> . . . . .	270

# List of Figures

1.1.	Example of a cumulative distribution function . . . . .	5
1.2.	Probability density of a random variable . . . . .	8
1.3.	Illustration of the quantile . . . . .	9
1.4.	Density function of the uniform distribution . . . . .	10
1.5.	Density distribution function of the exponential distribution . . . . .	13
1.6.	Density distribution function of the normal distribution . . . . .	16
1.7.	Density of the normal distribution . . . . .	17
2.1.	Q-Q-Plot on concrete compression strength . . . . .	39
2.2.	Q-Q plot for the concrete compression strength data set . . . . .	40
2.3.	Normal Plot of $\mathcal{N}(0, 1)$ and Cauchy distribution . . . . .	41
2.4.	Normal Plots . . . . .	42
2.5.	Density function of the test statistic $Z$ . . . . .	49
2.6.	Density function of $t_n$ -distribution . . . . .	53
2.7.	Interplay of statistical significance and relevance . . . . .	64
3.1.	Integration domain for joint probability density . . . . .	70
3.2.	Illustration of a two-dimensional density . . . . .	71
3.3.	Bivariate density $f(x, y) = (2x + 2y - 4xy)$ . . . . .	74
3.4.	Relation between independence and uncorrelatedness . . . . .	77
3.5.	Contour lines of a two-dimensional density . . . . .	78
3.6.	Example <b>USArrests</b> . . . . .	82
3.7.	Example <b>USArrests</b> . . . . .	83
3.8.	Example <b>USArrests</b> . . . . .	86
3.9.	Example <b>USArrests</b> . . . . .	88
4.1.	<b>Advertising</b> Data Set: Scatter Plots . . . . .	94
4.2.	<b>Income</b> Data set . . . . .	96
4.3.	Income as a function of Years of Education. . . . .	101
4.4.	<b>Income</b> Data set . . . . .	102
5.1.	<b>Advertising</b> Data Set : Regression Lines . . . . .	106
5.2.	<b>Advertising</b> Data Set : Least Squares Fit . . . . .	107
5.3.	<b>Advertising</b> Data Set: Least squares fit . . . . .	110
5.4.	Population Regression Line . . . . .	112
5.5.	Ten least squares lines for : $Y = 2 + 3X + \varepsilon$ . . . . .	113
5.6.	Confidence Band . . . . .	124

## List of Figures

5.7. Prediction Band . . . . .	127
6.1. Simulation of $y_i = 2 + 3x_i + \varepsilon_i$ . . . . .	130
6.2. Simulation of Error Term . . . . .	131
6.3. Simulation of Non-Constant Variance . . . . .	132
6.4. Simulation of Distribution of Error Terms . . . . .	132
6.5. Scatter Plots with Normally-Distributed Error Terms . . . . .	134
6.6. Violation of Model Assumption $E\{\varepsilon_i\} = 0$ . . . . .	135
6.7. Scatterplot and Tukey-Anscombe Plot for the <b>Advertising</b> Data Set	141
6.8. Smoothing Curves with Rectangular Kernel . . . . .	144
6.9. Tukey-Anscombe Plot for the Example <b>Income</b> . . . . .	146
6.10. Band of Smoothing Curves . . . . .	147
6.11. Tukey-Anscombe for a non-linear regression function . . . . .	149
6.12. Tukey-Anscombe plot . . . . .	150
6.13. Scale Location Plot . . . . .	152
6.14. Scale Location Plot . . . . .	153
6.15. Scale Location Plot . . . . .	154
6.16. <b>Advertising</b> Data Set : Histogram of the residuals $r_i$ . . . . .	155
6.17. Q-Q plot for the <b>Advertising</b> data. . . . .	157
6.18. Q-Q plot for the <b>Advertising</b> data set. . . . .	158
6.19. Q-Q plot for the <b>Income</b> data set. . . . .	159
6.20. Residual Plots for Simulated Data . . . . .	160
6.21. Least Square Regression Lines and Tukey-Anscombe Plots . . . . .	161
6.22. Leverage Points . . . . .	163
6.23. Cook's Distance and Leverage Statistic Plots . . . . .	165
6.24. Residual plots for the <b>Advertising</b> data set. . . . .	168
6.25. Residual plots for the <b>Income</b> data set. . . . .	169
6.26. Scale Location Plot for <b>Advertising</b> Data Set . . . . .	170
6.27. Scale Location Plot for the <b>Advertising</b> Data Set . . . . .	171
 7.1. Regression lines are added to Figure 4.1 for the <b>Advertising</b> data set.	175
7.2. Data points in 3D coordinate system for the <b>Income</b> data set. . . . .	177
7.3. <b>Income</b> Data Set: Multiple Regression . . . . .	179
7.4. <b>Advertising</b> Data Set : Linear Regression Fit . . . . .	189
7.5. <b>Credit</b> Data Set : Pairwise Scatter Plot . . . . .	198
7.6. <b>Credit</b> Data Set : Least Squares Lines . . . . .	211
7.7. <b>Auto</b> Data Set : Scatter Plots and Tukey-Anscombe Plot . . . . .	213
7.8. <b>horsepower</b> Data Set : Polynomials . . . . .	215
7.9. <b>Advertising</b> Data Set : Tukey-Anscombe Plot . . . . .	219
7.10. <b>Advertising</b> Data Set : Scale-Location Plot . . . . .	220
7.11. <b>Advertising</b> Data Set : Scatter Plots and Leverage Statistics . . . . .	223
7.12. <b>Credit</b> Data Set : Scatter Plots . . . . .	224
 8.1. <b>Credit</b> Data Set : Adjusted R <sup>2</sup> . . . . .	247

## List of Figures

8.2. <b>Credit</b> Data Set : AIC Statistic . . . . .	250
9.1. Schematic diagram of training a classification method. . . . .	260
9.2. Scatter plot of <b>income</b> versus <b>balance</b> from the <b>default</b> data set. . . . .	261
9.3. Boxplots for the sample data set <b>Default</b> . . . . .	262
10.1. A linear regression model fitted on the <b>Default</b> dataset. . . . .	266
10.2. The logistic function . . . . .	267
10.3. Logistic regression function on the <b>Default</b> data . . . . .	269
10.4. Histograms of estimated class probabilities. . . . .	278
10.5. Histograms of predicted class probabilities for downsampled data set	283
11.1. Airline passenger bookings (in thousands) per month. . . . .	292
11.2. CO2 emissions in Switzerland . . . . .	293
11.3. Annual temperature anomalies in Europe with respect to the average	294
11.4. City air quality measurements . . . . .	295
11.5. Daily closings of Tesla, Inc . . . . .	296
11.6. Log-RetURNS of the Tesla, Inc . . . . .	297
11.7. Quarterly beer production in Australia from March 1956 to June 1994.	301
11.8. Quarterly beer and electricity production in Australia. . . . .	302
11.9. Box-Cox-transformations for different values $\lambda$ . . . . .	305
11.10 <b>AirPassengers</b> data shifted back- and forwards. . . . .	306
11.11 Hourly air temperature in an Italian city. . . . .	308
11.12 Hourly air temperatur of 20 consecutive days . . . . .	309
11.13 Grouped boxplot of air temperature data. . . . .	310
11.14 Lagged scatterplot of the air temperature data. . . . .	311
11.15 Estimated <i>Trend</i> for the <b>Airpassenger</b> data set. . . . .	313
11.16 Estimated <i>seasonal effect</i> for the <b>Airpassenger</b> data set. . . . .	314
11.17 Estimated remainder term for the <b>AirPassengers</b> data. . . . .	315
11.18 Estimated remainder term for the logarithm of the AirPassengers data.	316
11.19 Decomposition time series according to additive time series model. .	317
11.20 STL decomposition of the log of the <b>AirPassengers</b> data . . . . .	318
12.1. A time series as observation of a random walk process. . . . .	322
12.2. A time series observation of a random walk with drift. . . . .	323
12.3. A realization of the white noise process . . . . .	324
12.4. A realization of a moving average process with window length 3. .	325
12.5. A realization of a autoregressive process . . . . .	326
12.6. The remainder sequence of the Australian electricity data . . . . .	334
12.7. The remainder sequence of the temperature measurements . . . . .	335
12.8. Sample autocovariance function of a MA(3) process . . . . .	338
12.9. Sample autocorrelation function of a MA(3)) process . . . . .	339
13.1. Time series generated by the AR(3) model. . . . .	344
13.2. The theoretical autocorrelation of an AR(3) process. . . . .	346

## List of Figures

13.3. The theoretical partial autocorrelation function of an AR(3) process . . . . .	348
13.4. The yearly averaged number of sunspot since 1749 . . . . .	349
13.5. Square-root Transformation of the <b>Sunspot</b> Data . . . . .	350
13.6. Autocorrelation and partial autocorrelation of <b>Sunspot</b> data . . . . .	351
13.7. The annually averaged sunspot number data and the fitted model. . . . .	353
13.8. Residual analysis for the fitted model. . . . .	354
13.9. A simulated AR(1) process with 150 observations with a prediction .	357
13.10 Sunspot numbers from 1950 to 2014 with training set and prediction	358
B.1. Q-Q-Plot on concrete compression strength . . . . .	370
B.2. <b>Advertising</b> Data Set: Scatter Plots . . . . .	374
B.3. <b>Advertising</b> Data Set: Least squares fit . . . . .	375
B.4. Population Regression Line . . . . .	376
B.5. Confidence Band . . . . .	379
B.6. Prediction Band . . . . .	380
B.7. Scatterplot and Tukey-Anscombe Plot for the <b>Advertising</b> Data Set	384
B.8. Smoothing Curves with Rectangular Kernel . . . . .	385
B.9. Tukey-Anscombe Plot for the Example <b>Income</b> . . . . .	386
B.10. Band of Smoothing Curves . . . . .	387
B.11. Scale Location Plot . . . . .	388
B.12. Scale Location Plot . . . . .	389
B.13. <b>Advertising</b> Data Set : Histogram of the residuals $r_i$ . . . . .	390
B.14. Q-Q plot for the <b>Advertising</b> data. . . . .	390
B.15. Q-Q plot for the <b>Advertising</b> data set. . . . .	391
B.16. Residual plots for the <b>Advertising</b> data set. . . . .	392
B.17. Residual plots for the <b>Income</b> data set. . . . .	393
B.18. <b>Credit</b> Data Set : Adjusted $R^2$ . . . . .	409
B.19. <b>Credit</b> Data Set : $C_p$ Statistic . . . . .	411
B.20. Scatter plot of <b>income</b> versus <b>balance</b> from the <b>default</b> data set.	417
B.21. Boxplots for the sample data set <b>Default</b> . . . . .	418
B.22. A linear regression model fitted on the <b>Default</b> dataset. . . . .	419
B.23. The logistic function . . . . .	420
B.24. Logistic regression function on the <b>Default</b> data . . . . .	421
B.25. Quarterly beer and electricity production in Australia . . . . .	429
B.26. Air passengers data of PanAm and beer production in Australia . . .	430
B.27. Box-Cox-transformations for different values of $\lambda$ . . . . .	432
B.28. Hourly air temperature in an Italian city . . . . .	434
B.29. Hourly air temperatur of 20 consecutive days . . . . .	435
B.30. Grouped boxplot of air temperature data . . . . .	436
B.31. Lagged scatterplot of the air temperature data . . . . .	437
B.32. Estimated remainder term for the AirPassengers data . . . . .	440
B.33. Estimated remainder term for the logarithm of the AirPassengers data	441
B.34. STL decomposition of the log of the Air Passengers data . . . . .	443
B.35. Cycle-subseries of the STL decomposition of the log Air Passengers data	444

## List of Figures

B.36. A time series as observation of a random walk process . . . . .	445
B.37. A time series observation of a random walk with drift (black) . . . . .	446
B.38. A realization of the white noise process . . . . .	447
B.39. A realization of a moving average process with window length 3 . . . . .	448
B.40. A realization of a autoregressive process . . . . .	449
B.41. The remainder of the Australian electricity data after transformations	450
B.42. The remainder of the temperature measurements after STL decomposition	451
B.43. Sample autocovariance function of a MA(5) process . . . . .	452
B.44. Sample autocorrelation function of a MA(5) process . . . . .	453
B.45. Time series generated by the AR(3) model . . . . .	454
B.46. The theoretical autocorrelation of an AR(3) process . . . . .	455
B.47. The estimated partial autocorrelation function of an AR(3) process .	456
B.48. The monthly sunspot numbers since 1749 and yearly averaged values	457
B.49. Squareroot transformed yearly time series . . . . .	457
B.50. Autocorrelation and partial autocorrelation of transformed sunspot data	458
B.51. The annually averaged sunspot number data and the fitted model . .	458
B.52. Residual analysis for the fitted model . . . . .	459
B.53. A simulated AR(1) process with 150 observations, including a prediction	460
B.54. Sunspot numbers from 1950 to 2014, showing training set and prediction	460

# Chapter 1.

## Probability Models for Measurement Data

Everybody believes in the exponential law of errors [i.e., the Normal distribution]: the experimenters, because they think it can be proved by mathematics; and the mathematicians, because they believe it has been established by observation.

---

(E. T. Whittaker and G. Robinson)

### 1.1. Continuous Random Variables and Probability Distributions

Recall a *random variable* is map from the sample space  $\Omega$  to the range  $W_X$

$$X : \Omega \rightarrow W_X$$

The *value range*  $W_X$  of a random variable  $X$  is the set of all values which  $X$  can take on. Because the outcome of the experiment with sample space  $\Omega$  is random, the number produced by the function is random as well. Thus, a random variable is essentially a random number.

#### Example 1.1.1

A coin is thrown three times, and the sequence of heads and tails is observed; thus,

$$\Omega = \{hhh, hht, htt, hth, ttt, tth, thh, tht\}$$

Examples of random variables defined on  $\Omega$  are

1. the total number of heads, that is  $W_X = \{0, 1, 2, 3\}$
2. the total number of tails, that is  $W_X = \{0, 1, 2, 3\}$
3. the number of heads minus the number of tails, that is  $W_X = \{-3, -1, 1, 3\}$

Each of these is a real-valued function defined on  $\Omega$ ; that is, each is a rule that assigns a real number to every element  $\omega \in \Omega$ ; ◀

In the previous example, the ranges are discrete. We therefore call the random variable  $X$  in this case *discrete*.

In many applications, we are dealing with measurement data that can theoretically take on any value in a certain range, contrary to count data. The accuracy of the measured values is specified by the experimental measurement accuracy.

A random variable  $X$  is called *continuous* if its value range  $W_X$  is continuous. Examples of continuous value ranges are:

$$W_X = \mathbb{R}, \quad \mathbb{R}^+ \quad \text{or} \quad [0, 1]$$

#### Convention for the bracket notation:

Round brackets mean that the value is outside the interval while square brackets mean that the value is inside the interval. The interval  $(a, b]$  therefore contains all points  $x$  with  $x > a$  and  $x \leq b$ .

The *probability distribution* of a discrete random variable can be defined by listing the *probability mass functions*  $P(X = x)$  for all possible values  $x$  in the value range.

#### Example 1.1.2

Let us come back to the example where a coin is thrown three times. Let  $X$  be the random variable that counts the total number of heads. Then we can represent the probability distribution as a table:

$x$	0	1	2	3
$P(X = x)$	1/8	3/8	3/8	1/8

For instance,  $X = 0$  relates to the case when no head appeared in the three tosses.  $\Omega$  contains 8 elements among which one element does not include head. Hence, the probability mass function becomes  $P(X = 0) = 1/8$ . If we sum up all probabilities, we obtain 1. ◀

## Chapter 1. Probability Models for Measurement Data

For a continuous random variable  $X$ , however, the following applies:

$$P(X = x) = 0$$

for all  $x \in W_X$ . This implies that we cannot define the probability distribution of  $X$  by means of probability mass functions.

### Example 1.1.3

This relation can be intuitively illustrated with an example. Assuming we have a random variable  $X_0$ , which takes on every value from the discrete set

$$W_0 = \{0, 1, 2, \dots, 8, 9\}$$

with the same probability. The probability that the random variable  $X_0$  takes on a specific value  $x$  (e.g.  $x = 3$ ) from the range  $W_0$  is therefore

$$P(X_0 = x) = \frac{1}{10}$$

since  $W_0$  consists of ten elements. We now enlarge the discrete set by defining every number to a decimal place:

$$W_1 = \{0.0, 0.1, 0.2, \dots, 9.8, 9.9\}$$

The random variable  $X_1$  takes on every value from  $W_1$  with the same probability, hence

$$P(X_1 = x) = \frac{1}{100}$$

since  $W_1$  consists of one hundred elements. As an example, we therefore have

$$P(X_1 = 3.0) = \frac{1}{100}$$

If you add a decimal place, you obtain a set consisting of one thousand elements and the probability of drawing a certain element randomly (e.g. the number 3.00), will now be  $1/1000$ . If we continue this procedure, we end up with the following rule: If each of the numbers between 0 and less than 10 with  $i$  decimal points is drawn with the same probability, the probability of drawing one particular number from this set becomes

$$\frac{1}{10^{i+1}}$$

The discrete set becomes continuous when we allow infinitely many decimal places, that is we obtain

$$W_\infty = [0, 10)$$

We conclude that the probability of drawing a certain element with an infinite number of decimal places from this set is

$$P(X_\infty = x) = \frac{1}{\infty} = 0$$



**Example 1.1.4**

We measure the body lengths of many persons. The probability of measuring a body length of *exactly* 182.254 680 895 434... cm is equal to 0:

$$P(X = 182.254 680 895 434...) = 0$$

where  $X$  measures the body length. But which probability can be determined in association with body lengths? Now, we could for instance determine the probability that a measured value lies within a certain range, e.g. between 174 and 175 cm:

$$P(174 < X \leq 175)$$

This probability is then no longer 0. However, we cannot simply add up the probability mass functions, as this would yield 0. We therefore need a new concept and this is the so-called *probability density*. ◀

If you have a data set with experimental measurements, you will observe that the relative frequency of measuring points within specific intervals is higher than in others. The *probability distribution of a continuous random variable  $X$*  can be described by defining the probabilities for all intervals  $(a, b]$  with  $a < b$ :

$$P(X \in (a, b]) = P(a < X \leq b)$$

For this, it is sufficient to define the *cumulative distribution function*

$$F(x) = P(X \leq x)$$

because of the relation

$$P(a < X \leq b) = F(b) - F(a)$$

This cumulative distribution function has some important properties (see Figure 1.1):

1. Since  $F(x) = P(X \leq x)$  is a probability, the following applies

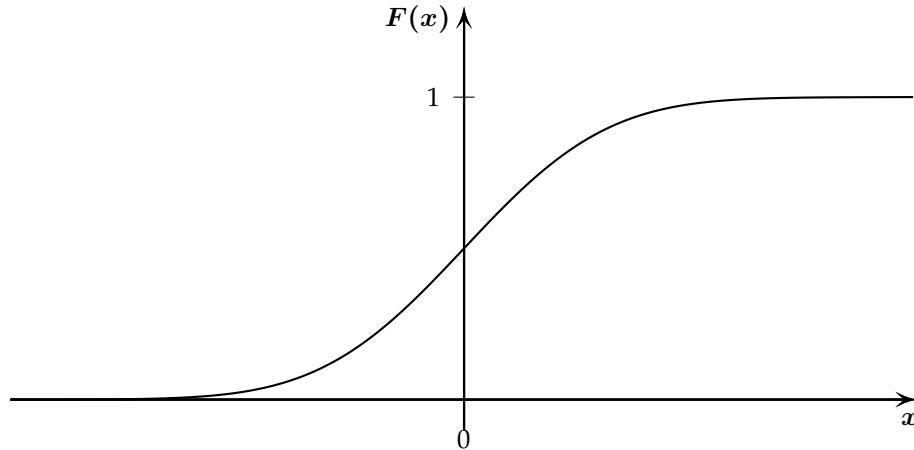
$$0 \leq F(x) \leq 1$$

2. The probability  $P(X \leq -\infty)$ , that a measured value is less than  $-\infty$  is obviously 0. And therefore also

$$F(-\infty) = 0$$

3. The probability  $P(X \leq \infty)$  that a measured value is smaller than  $\infty$  is obviously 1:

$$F(\infty) = 1$$



**Figure 1.1.:** Example of a cumulative distribution function.

4. The function of  $F(x)$  is monotonically increasing. For  $a < b$ , we have

$$F(a) \leq F(b)$$

The derivative  $F'(x)$  of  $F(x)$  thus is always greater or equal 0.

In summary, the probability distribution of a continuous random variable  $X$  can be defined by the cumulative distribution function. Because

$$P(X = a) = P(X = b) = 0$$

it does not matter whether we write  $<$  or  $\leq$ . However, the distinction between  $<$  and  $\leq$  is important in the case of discrete random variables.

### 1.1.1. Probability Density

For *continuous* random variables, we can obtain an expression analogous to the probability mass function  $P(X = x)$  by considering the derivative of the cumulative distribution function. To achieve this, we aim at defining the probability

$$P(a < X \leq b)$$

as a sum of probabilities. We subdivide the interval  $(a, b]$  into  $n$  sub-intervals with sub-points

$$a = x_0, x_1, x_2, \dots, b = x_n$$

The individual sub-intervals have the same length  $\Delta x$ :

$$\Delta x = \frac{b - a}{n}$$

## Chapter 1. Probability Models for Measurement Data

We can then write the probability  $P(a < X \leq b)$  as follows

$$\begin{aligned} P(a < X \leq b) &= P(x_0 < X \leq x_1) + P(x_1 < X \leq x_2) + \dots + P(x_{n-1} < X \leq x_n) \\ &= P(x_0 < X \leq x_0 + \Delta x) + \dots + P(x_{n-1} < X \leq x_{n-1} + \Delta x) \\ &= \sum_{i=0}^{n-1} P(x_i < X \leq x_i + \Delta x). \end{aligned}$$

For the summands in the final sum, we use the cumulative distribution function  $F$ :

$$P(a < X \leq b) = \sum_{i=0}^{n-1} P(x_i < X \leq x_i + \Delta x) = \sum_{i=0}^{n-1} (F(x_i + \Delta x) - F(x_i))$$

We now multiply the summands by  $\Delta x$ :

$$P(a < X \leq b) = \sum_{i=0}^{n-1} (F(x_i + \Delta x) - F(x_i)) = \sum_{i=0}^{n-1} \frac{F(x_i + \Delta x) - F(x_i)}{\Delta x} \Delta x$$

For small  $\Delta x$ , it follows from the definition of the derivative :

$$F'(x_i) \approx \frac{F(x_i + \Delta x) - F(x_i)}{\Delta x}$$

If we insert this into our sum, we will obtain

$$P(a < X \leq b) \approx \sum_{i=0}^{n-1} F'(x_i) \Delta x$$

This sum for instance is a Riemann sum, which for  $\Delta x \rightarrow 0$  approximates an integral. Therefore, we have

$$P(a < X \leq b) = \int_a^b F'(x) \, dx$$

We call this function  $F'(x)$  the *probability density function*.

### Probability Density Function

The **probability density  $f$**  is defined as the derivative of the cumulative distribution function:

$$f(x) = F'(x)$$

## Chapter 1. Probability Models for Measurement Data

You can recover the cumulative distribution function from the density by integrating the density function

$$F(x) = \int_{-\infty}^x f(y) dy$$

since  $F$  represents an antiderivative of  $f$  and  $F(-\infty) = 0$ .

### Properties of Probability Density Function

The following properties hold for a probability density  $f(x)$  (see Figure 1.2):

1. We have

$$f(x) \geq 0$$

for all  $x$ , as  $F(x)$  is monotonically increasing and therefore its derivative must be larger or equal to 0.

2. The following relation applies

$$P(a < X \leq b) = F(b) - F(a) = \int_a^b f(x) dx$$

This corresponds to the area between  $a$  and  $b$  under  $f(x)$ .

3. The relation

$$\int_{-\infty}^{\infty} f(x) dx = 1$$

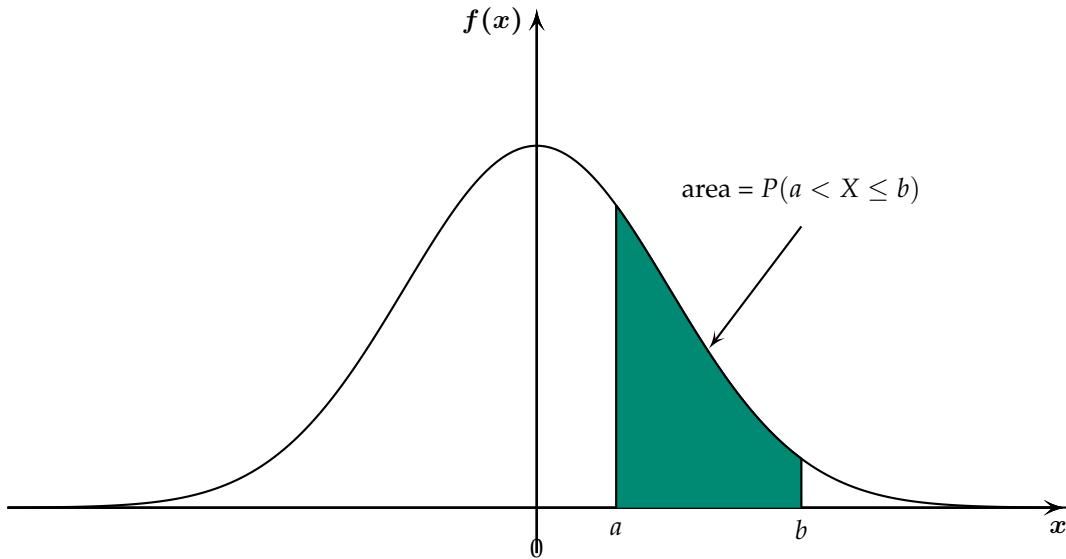
always holds. This is the probability that *any* value will be measured.

### 1.1.2. Summary Statistics of Continuous Distributions

The *expected value*  $E[X]$  and the *standard deviation*  $\sigma_X$  of a continuous random variable  $X$  have the same meaning as in the discrete case : The expected value refers to the average location or to the center of gravity of the measurement distribution and the standard deviation to the extent the measurements scatter around the mean. We remember that the expected value of a discrete random variable is defined as

$$E[X] = \sum_i x_i P(X = x_i)$$

The formulae for the *expected value* and the *variance* of a continuous random variable are obtained by replacing the discrete mass probability  $P(X = x)$  by  $f(x)dx$  and the sum by an integral:



**Figure 1.2.:** Illustration of the probability density of a random variable and the probability of measuring a value in the interval  $(a, b]$  (green area).

### Expected Value and Variance

**Expected value and variance** are defined as shown below:

$$\begin{aligned} E[X] = \mu_X &= \int_{-\infty}^{\infty} xf(x) dx \\ \text{Var}[X] = \sigma_X^2 &= E[(X - E[X])^2] = \int_{-\infty}^{\infty} (x - E[X])^2 f(x) dx = E[X^2] - E[X]^2 \end{aligned}$$

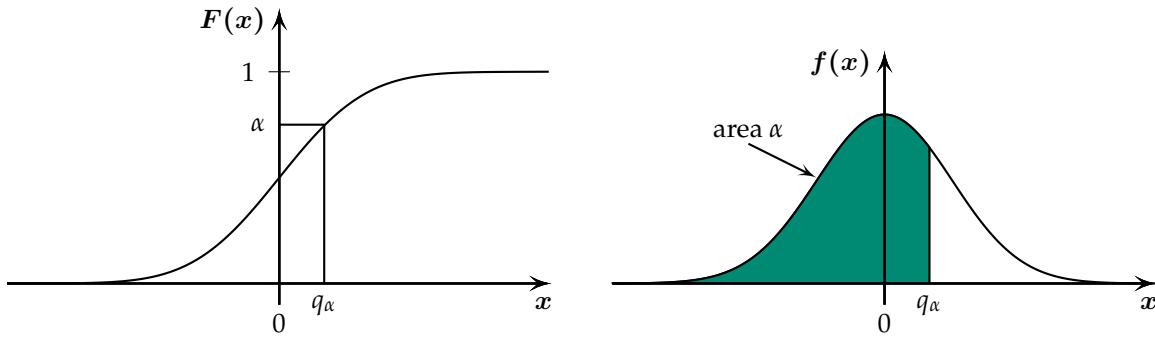
The **quantiles**  $q(\alpha)$  of a random variable  $X$  with  $0 < \alpha < 1$ , or their distribution are defined as follows:

$$P(X \leq q(\alpha)) = \alpha$$

That means:

$$F(q(\alpha)) = \alpha \iff q(\alpha) = F^{-1}(\alpha).$$

This can also be interpreted in such a way that  $q(\alpha)$  is the point, such that the area from  $-\infty$  to  $q(\alpha)$  under the density  $f$  is equal to  $\alpha$  (see Figure 1.3). Recall the 50 %-quantile is called **median**.



**Figure 1.3.:** (Left): Illustration of the quantile  $q_\alpha$  by means of the cumulative distribution function  $F(x)$  and (right): the density  $f(x)$  for  $\alpha = 0.75$ .

### Example 1.1.5

In turn, let us consider again the distribution of body lengths. Assuming for  $\alpha = 0.75$ , the corresponding quantile is given by  $q(\alpha) = 182.5$ , then we know that 75 % of the persons measured are shorter or equal to 182.5 cm.  $\blacktriangleleft$

## 1.2. Important Continuous Distributions

As examples of discrete probability distributions, we know the binomial distribution and the Poisson distribution. In this chapter, we will discuss some of the most important continuous distributions.

We have seen in Section 1.1 that we can characterize the probability distribution of a continuous random variable with the cumulative distribution function  $F$  or the density  $f$ .

### 1.2.1. Uniform Distribution

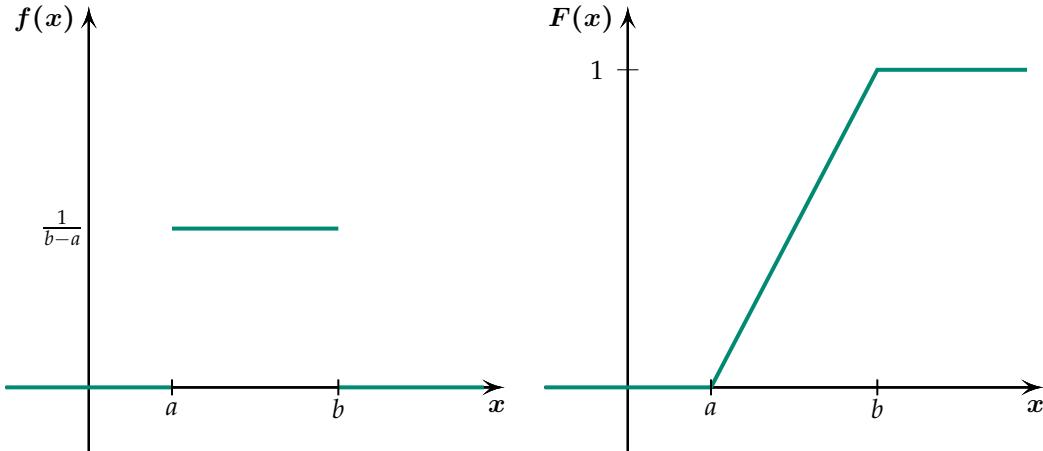
The uniform distribution represents a probabilistic formalization of complete “ignorance”.

#### Uniform Probability Distribution

A random variable  $X$  with range  $W_X = [a, b]$  is uniformly distributed on  $([a, b])$  if

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{if } a \leq x \leq b \\ 0 & \text{otherwise} \end{cases}$$

The density function is therefore constant on the interval  $[a, b]$  (see the left-hand panel of Figure 1.4). In other words, the probability is the same on the entire range  $W_X = [a, b]$ , hence the name *uniform*.



**Figure 1.4:** (Left): Density distribution function and (right): cumulative function of the uniform distribution.

The corresponding cumulative distribution function (see figure 1.4 right) is given by

$$F(x) = \begin{cases} 0 & \text{if } x < a \\ \frac{x-a}{b-a} & \text{if } a \leq x \leq b \\ 1 & \text{if } x > b \end{cases}$$

For  $X \sim \text{Uniform}([a, b])$  the summary statistics are given by

$$\begin{aligned} E[X] &= \int_{-\infty}^{\infty} xf(x) dx = \int_a^b \frac{x}{b-a} dx = \frac{b^2 - a^2}{2(b-a)} = \frac{a+b}{2} \\ \text{Var}[X] &= \int_{-\infty}^{\infty} (x - E[X])^2 f(x) dx = \int_a^b \left(x - \frac{a+b}{2}\right)^2 \frac{1}{b-a} dx = \frac{(b-a)^2}{12} \\ \sigma_X &= \sqrt{\frac{(b-a)^2}{12}} \end{aligned}$$

### Example 1.2.1

With [Python](#) the value of the probability density function `Uniform([1, 10])` at the position  $x = 5$  can be calculated as follows: ([to R](#))

## Chapter 1. Probability Models for Measurement Data

```
[1]: from scipy.stats import uniform  
  
pdf_5 = uniform.pdf(x=5, loc=1, scale=9)  
print(pdf_5)
```

0.1111111111111111

In the case of  $X \sim \text{Uniform}([1, 10])$ , the probability  $P(1 \leq X \leq 5)$  corresponds exactly to the probability  $P(X \leq 5)$ . We calculate this with [Python](#) as follows: [\(to R\)](#)

```
[2]: p_0_5 = uniform.cdf(x=5, loc=1, scale=9)  
print(p_0_5)
```

0.4444444444444444

The probability  $P(1.2 \leq X \leq 4.8)$  is calculated as follows [\(to R\)](#)

```
[3]: p_12_48 = uniform.cdf(x=4.8, loc=1, scale=9) - uniform.cdf(x=1.2, loc=1, scale=9)  
print(p_12_48)
```

0.4

The generation of uniformly distributed random variables is of great significance. Uniformly distributed random variables can be generated with [Python](#) as shown below: [\(to R\)](#)

```
[4]: rand = uniform.rvs(loc=1, scale=9, size=5)  
print(rand)
```

[8.4032 6.3259 1.1848 4.2129 9.7512]



### 1.2.2. Exponential Distribution

The exponential distribution is the simplest model for *waiting times for failure*, therefore for the *lifespan*.

#### Example 1.2.2

Lifespan of electronic devices if ageing phenomena need not be considered.



**Example 1.2.3**

How long do we have to wait until the next decay of an alpha emitter occurs? In this case, we consider the time of an isotope to decay as its lifespan. ◀

The Poisson distribution relates to the distribution of the number of observed random events in a predetermined time interval. We determine the probability for a lifespan with the exponential distribution.

**Notation**

We often write the natural exponential function  $e^x$  in the form:

$$\exp(x) := e^x$$

**Exponential Distribution**

A random variable  $X$  with range  $W_X = \mathbb{R}^+ = [0, \infty)$  is *exponentially distributed* with parameter  $\lambda \in \mathbb{R}^+$  if

$$f(x) = \begin{cases} \lambda \cdot \exp(-\lambda x), & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

We write

$$X \sim \text{Exp}(\lambda)$$

The corresponding cumulative distribution function is given by

$$F(x) = \begin{cases} 1 - e^{-\lambda x} & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

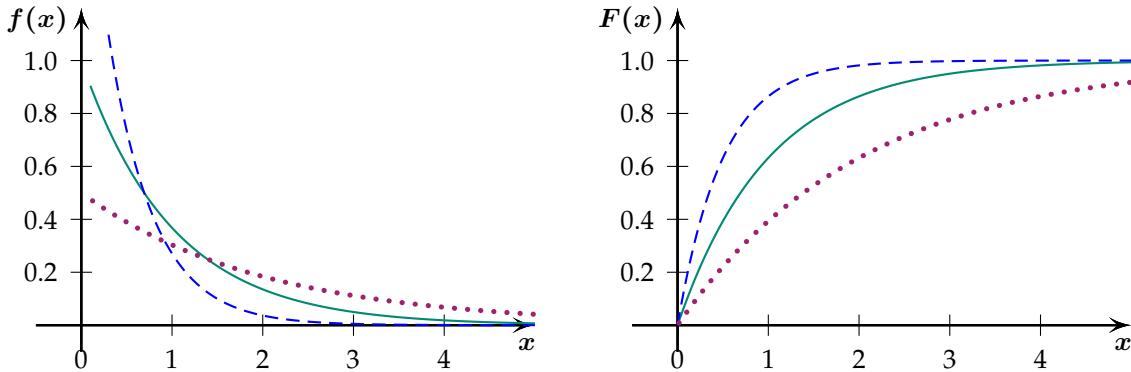
The density and cumulative distribution functions for different values of  $\lambda$  are displayed in Figure 1.5.

For  $X \sim \text{Exp}(\lambda)$  the summary statistics are shown below:

$$E[X] = \int_{-\infty}^{\infty} x f(x) dx = \int_0^{\infty} x \cdot \lambda \cdot e^{-\lambda x} dx = \frac{1}{\lambda}$$

$$\text{Var}[X] = \int_{-\infty}^{\infty} (x - E[X])^2 f(x) dx = \int_0^{\infty} \left(x - \frac{1}{\lambda}\right)^2 \cdot \lambda \cdot e^{-\lambda x} dx = \frac{1}{\lambda^2}$$

$$\sigma_X = \frac{1}{\lambda}$$



**Figure 1.5.:** (Left): Density distribution function of the exponential distribution for  $\lambda = 1$  (green),  $\lambda = 2$  (blue dashed) and  $\lambda = 1/2$  (violet dotted). (Right): Cumulative distribution function for the exponential distribution.

### Example 1.2.4

How long does it take a particular radioactive isotope to decay so that the probability is  $1/2$ ?

We denote the decay time by  $T$ . This can also be referred to as the *lifetime*. The exponential distribution is suitable as a model for this random duration, therefore

$$T \sim \text{Exp}(\lambda)$$

The parameter  $\lambda$  depends on the particular isotope and must be estimated in general on the basis of experiments. At what time is the probability that the isotope will have decayed by then, respectively “survives” until this moment, equal to  $1/2$ ? The median yields the answer

$$F(t_{1/2}) = 1 - \exp(-\lambda t_{1/2}) = \frac{1}{2} \quad \Rightarrow \quad \exp(-\lambda t_{1/2}) = \frac{1}{2}$$

We solve this equation for  $t_{1/2}$  by taking the logarithm of both sides of the equation:

$$-\lambda t_{1/2} = \ln\left(\frac{1}{2}\right) \quad \Rightarrow \quad t_{1/2} = \frac{\ln(2)}{\lambda} = \frac{0.693}{\lambda}$$

However, there are many active isotopes in a radioactive sample. If the probability that a single isotope decays by the time  $\frac{0.693}{\lambda}$  is  $1/2$ , then the relative frequency of the surviving isotopes in the sample at the time point  $\frac{0.693}{\lambda}$  is likewise  $1/2$ . The relative frequency of the *decayed* active isotopes is accordingly likewise  $1/2$ . Therefore

$$t_{1/2} = \frac{\ln(2)}{\lambda}$$

is called the *half-life*. After this time, on average, half of all active isotopes have decayed. ◀

### Example 1.2.5

Assuming  $X \sim \text{Exp}(3)$ , the probability  $P(0 \leq X \leq 4)$  can be calculated in [Python](#) as follows: ([to R](#))

```
[1]: from scipy.stats import expon
p_0_4 = expon.cdf(x=4, scale=1/3)
print(p_0_4)
```

0.9999938557876467



The exponential distribution and the Poisson-distribution are related to each other:

#### Connection between Poisson Distribution and Exponential Distribution

If the *time* elapsed between two failures of a system follows an  $\text{Exp}(\lambda)$  distribution, then the *number* of failures in a time interval of duration  $t$  is distributed according to  $\text{Poisson}(\lambda t)$ .

### Example 1.2.6

Assuming, at time point  $t_0 = 0$  a radioactive decay occurs in a radioactive sample. What is the probability that another decay occurs after the time  $t$ ?

The probability that a decay could occur again after the time  $t$ , is

$$P(T > t) = P(\text{no decay in } [0, t])$$

The number of decays in the time interval  $[0, t]$  follows a Poisson distribution with parameter  $\lambda t$ . Consequently

$$P(T > t) = P(\text{no decay in } [0, t]) = \frac{(\lambda t)^0 e^{-\lambda t}}{0!} = e^{-\lambda t}$$

Therefore, the lifetime  $T$  of an atom follows an exponential distribution with parameter  $\lambda$ . The cumulative distribution function is given by

$$F(t) = P(T \leq t) = 1 - P(T > t) = 1 - e^{-\lambda t} \quad \text{for } t \geq 0$$



### 1.2.3. Normal Distribution (Gaussian distribution)

The *normal distribution* (sometimes also known as *Gaussian distribution*) is the most frequent distribution for measurement values. It occurs in many applications and is the most important probability distribution in statistics. It is both of great theoretical and practical importance.

#### Normal Distribution

A random variable  $X$  with range  $W_X = \mathbb{R}$  is *normally distributed* with parameters  $\mu \in \mathbb{R}$  and  $\sigma^2 \in \mathbb{R}^+$  if

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

We denote the distribution for the random variable  $X$  as follows

$$X \sim \mathcal{N}(\mu, \sigma^2)$$

The corresponding cumulative distribution function

$$F(x) = \int_{-\infty}^x f(y) dy$$

cannot be explicitly expressed by means of standard functions like  $x^2$ ,  $\exp(x)$ ,  $\log(x)$  etc. These integrals are calculated numerically (by computer software).

For  $X \sim \mathcal{N}(\mu, \sigma^2)$  the summary statistics are given by

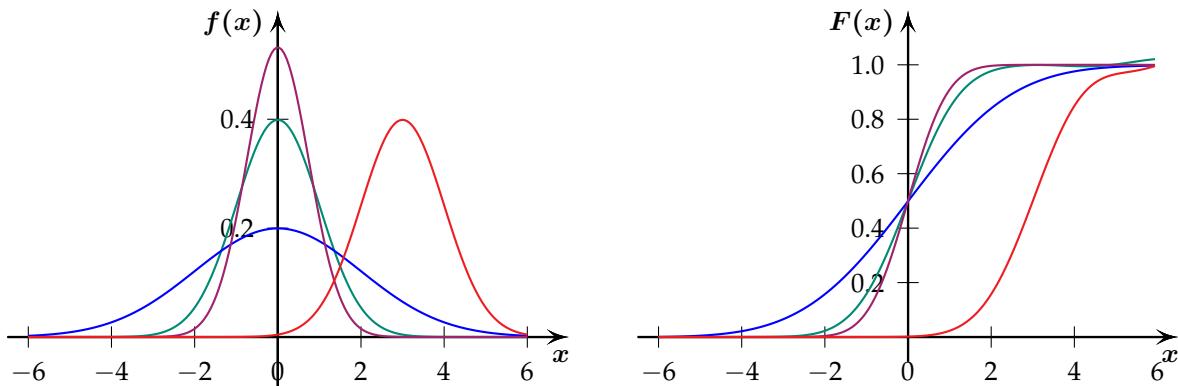
$$\mathbb{E}[X] = \int_{-\infty}^{\infty} xf(x) dx = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{\infty} x \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) dx = \mu$$

$$\text{Var}[X] = \int_{-\infty}^{\infty} (x - \mathbb{E}[X])^2 f(x) dx = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{\infty} (x - \mu)^2 \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) dx = \sigma^2$$

$$\sigma_X = \sigma$$

The parameters  $\mu$  and  $\sigma^2$  thus have a natural interpretation as expected value and variance of the distribution. Three normal distributions with different values of  $\mu$  and  $\sigma$  are displayed in Figure 1.6.

The density of the normal distribution is symmetrical about the expected value  $\mu$ . The larger the  $\sigma$ , the flatter and wider the density. For a small  $\sigma$  there is a “narrow and high” peak. A change in  $\mu$  simply shifts the density to the left or right.



**Figure 1.6.:** Densities (left) and cumulative distribution functions (right) of the normal distributions for  $\mu = 0, \sigma = 1$  (green),  $\mu = 0, \sigma = 2$  (blue),  $\mu = 0, \sigma = 0.75$  (violet) and  $\mu = 3, \sigma = 1$  (red).

### Example 1.2.7

The intelligence quotient (IQ) is usually measured by means of intelligence tests. The results of an IQ test follow approximately a normal distribution with a mean value of 100 and standard deviation of 15. In general, a person is regarded as highly gifted if his/her IQ constitutes two or more standard deviations to the right of the mean value. We aim at determining the probability that someone is highly gifted, that is having an IQ higher than 130. This is the probability  $P(X > 130)$ , where

$$X \sim \mathcal{N}(100, 15^2)$$

In [Python](#) we compute this probability by rewriting  $P(X > 130)$  as

$$1 - P(X \leq 130)$$

Thus,

[\(to R\)](#)

```
[1]: from scipy.stats import norm
p_130 = 1-norm.cdf(x=130, loc=100, scale=15)
print(p_130)
```

0.02275013194817921

Therefore around 2 % of the population is highly gifted.

We could also ask which percentage of the population's IQ lies within a standard deviation of the mean value. Therefore we are interested in the probability

$$P(85 \leq X \leq 115)$$

[\(to R\)](#)

```
[2]: p_85_115 = norm.cdf(x=115, loc=100, scale=15) - norm.cdf(x=85, loc=100, scale=15)
print(p_85_115)
```

0.6826894921370859

I.e. approximately  $2/3$  of the population have an IQ between 85 and 115. ◀

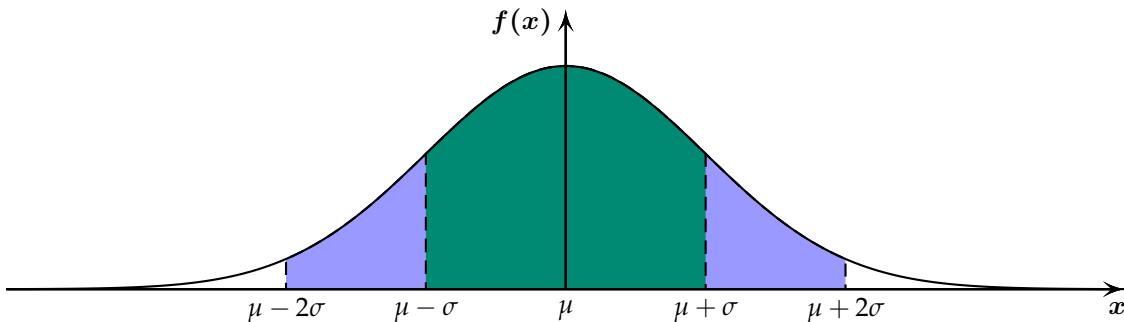
The last result from the example above applies to all normal distributions  $\mathcal{N}(\mu, \sigma^2)$ . The probability that an observation deviates one standard deviation from the expected value, at the most, is approximately  $2/3$ :

$$P(\mu - \sigma \leq X \leq \mu + \sigma) \approx \frac{2}{3}$$

We can also compute the probability that an observation deviates at most two standard deviations from the expected value:

$$P(\mu - 2\sigma \leq X \leq \mu + 2\sigma) \approx 0.95$$

As these probabilities are calculated by means of integrals, we can interpret them as areas. The area under the normal distribution over the interval  $[\mu - \sigma, \mu + \sigma]$  is approximately  $2/3$ . The area under the normal distribution over the interval  $[\mu - 2\sigma, \mu + 2\sigma]$  is approximately 0.95, see Figure 1.7.



**Figure 1.7:** Density of the normal distribution. Approximately 68 % of the area under the density function lies over the interval  $[\mu - \sigma, \mu + \sigma]$ , approximately 95 % of the area over the interval  $[\mu - 2\sigma, \mu + 2\sigma]$ .

### 1.3. Functions of a Random Variable

If  $g : \mathbb{R} \rightarrow \mathbb{R}$  is a function from  $\mathbb{R}$  to  $\mathbb{R}$  and  $X$  is a random variable, then the composition

$$Y = g(X)$$

is a new random variable. This composite function means that the realization  $y = g(x)$  of  $Y$  is associated with every realization  $x$  of  $X$ . Such transformations occur frequently. We begin with the most important special case:

### 1.3.1. Linear Transformations of Random Variables

We will first consider the case of a **linear transformation**

$$y = g(x) = a + bx \quad (a, b \in \mathbb{R})$$

#### Example 1.3.1

In a series of measurements, we have recorded temperatures in degrees Celsius and now want to convert them to degrees Fahrenheit. The temperature  $T_c$  in degrees Celsius can be converted into temperature  $T_F$  (degrees Fahrenheit) according to the following conversion formula:

$$T_F = \frac{9}{5} \cdot T_c + 32$$



#### Properties of Linear Transformations of a Random Variable

For

$$Y = a + bX$$

the following relations apply

- (i)  $E[Y] = E[a + bX] = a + bE[X]$
- (ii)  $\text{Var}[Y] = \text{Var}[a + bX] = b^2 \text{Var}[X], \quad \sigma_Y = |b|\sigma_X$
- (iii)  $\alpha - \text{Quantile of } Y = q_Y(\alpha) = a + bq_X(\alpha)$
- (iv)  $f_Y(y) = \frac{1}{b}f_X\left(\frac{y-a}{b}\right)$

We can easily verify the correctness of the first relation. For the special case

$$Y = a + X$$

the observations  $X$  are all shifted by  $a$  and consequently the expected value also shifts by  $a$ . For the case

$$Y = bX$$

## Chapter 1. Probability Models for Measurement Data

all observations  $X$  are multiplied with  $b$ . Hence, the expected value of  $X$  is multiplied by  $b$ .

For completeness' sake, we derive the following four relations.

The first equation follows from

$$\begin{aligned} E[Y] &= E[a + bX] \\ &= \int_{-\infty}^{\infty} (a + bx)f_X(x) dx \\ &= a \int_{-\infty}^{\infty} f_X(x) dx + b \int_{-\infty}^{\infty} xf_X(x) dx \\ &= a \cdot 1 + b \cdot E[X] \\ &= a + b E[X] \end{aligned}$$

The second relation follows from the definition of the variance and from the first equation:

$$\begin{aligned} \text{Var}[Y] &= \text{Var}[a + bX] \\ &= E[(a + bX - E[a + bX])^2] \\ &= E[(a + bX - (a + bE[X]))^2] = E[b^2(X - E[X])^2] \\ &= b^2 E[(X - E[X))^2] = b^2 \text{Var}[X] \end{aligned}$$

The root square of the above expression yields the standard deviation  $\sigma_Y = |b| \cdot \sigma_X$ .

The third relation follows from

$$\begin{aligned} F_Y(q_Y(\alpha)) &= P(Y \leq q_Y(\alpha)) \\ &= P(a + bX \leq q_Y(\alpha)) \\ &= P\left(X \leq \frac{q_Y(\alpha) - a}{b}\right) \\ &= F_X\left(\frac{q_Y(\alpha) - a}{b}\right) \end{aligned}$$

From

$$F_Y(q_Y(\alpha)) = \alpha = F_X\left(\frac{q_Y(\alpha) - a}{b}\right)$$

follows  $\frac{q_Y(\alpha) - a}{b} = q_X(\alpha)$  and hence the third relation. The fourth relation follows from

$$\begin{aligned} f_Y(y) &= \frac{d}{dy} F_Y(y) \\ &= \frac{d}{dy} F_X\left(\frac{y - a}{b}\right) \\ &= \frac{1}{b} f_X\left(\frac{y - a}{b}\right) \end{aligned}$$

where we have applied the following relation

$$\begin{aligned} F_Y(y) &= P(Y \leq y) \\ &= P(a + bX \leq y) \\ &= P\left(X \leq \frac{y-a}{b}\right) \\ &= F_X\left(\frac{y-a}{b}\right) \end{aligned}$$

### Example 1.3.2

We have measured a temperature in degrees Celsius and know the standard deviation of the measurement error on this scale:  $\sigma_C = 1/3$  degrees Celsius. For a report that is going to be published for a Northern American public, we want to indicate the temperature in degrees Fahrenheit and not in degrees Celsius. What is the standard deviation  $\sigma_F$  of the measurement error if we publish the temperature in degrees Fahrenheit?

As we have already seen, we can convert the temperature  $T_c$  in degrees Celsius into the temperature  $T_F$  in degrees Fahrenheit by means of the following formula

$$T_F = \frac{9}{5} \cdot T_C + 32$$

Hence the standard deviation in degrees Fahrenheit is given by

$$\sigma_F = \frac{9}{5} \sigma_C = \frac{9}{5} \cdot \frac{1}{3} = \frac{3}{5}$$



### Example 1.3.3 Fahrenheit for the advanced reader

If the measurements of  $T_C$  follow a normal distribution with the mean value  $\mu_C$  and standard deviation  $\sigma_C$ , therefore  $T_C \sim \mathcal{N}(\mu_C, \sigma_C^2)$ , then

$$T_F = a + bT_C \sim \mathcal{N}(a + b\mu_C, b^2\sigma_C^2)$$

with  $a = 32$  and  $b = 9/5$ .

Then according to rule (iv) for linear transformations, the following density function follows

$$\begin{aligned} f_{T_F}(T_F) &= \frac{1}{\sqrt{2\pi}\sigma_C b} \exp\left(-\frac{\left(\frac{T_F-a}{b} - \mu_C\right)^2}{2\sigma_C^2}\right) \\ &= \frac{1}{\sqrt{2\pi}\sigma_C b} \exp\left(-\frac{(T_F - (a + b\mu_C))^2}{2\sigma_C^2 b^2}\right). \end{aligned}$$

The distribution of a random variable that results from a linear transformation of a normally distributed random variable thus is in turn a normal distribution. This property that one remains within the same distribution class after a linear transformation is specific to the normal distribution and does not hold in general.  $\blacktriangleleft$

### Standardization of a Random Variable

We can always apply a linear transformation to  $X$  so that the expected value of the transformed variable becomes 0 and the variance 1. By applying the following linear transformation

$$g(x) = \frac{x - E[X]}{\sigma_X} = -\frac{E[X]}{\sigma_X} + \frac{1}{\sigma_X}x = a + bx$$

with

$$a = -\frac{E[X]}{\sigma_X} \quad \text{and} \quad b = \frac{1}{\sigma_X}$$

to  $X$ , we obtain the transformed random variable

$$Z = g(X) = \frac{X - E[X]}{\sigma_X}$$

By means of the rules for linear transformations, the following properties of  $Z$  follow

$$E[Z] = a + b E[X] = 0, \quad \text{Var}[Z] = b^2 \text{Var}[X] = 1$$

### Standardization of a Normally Distributed Random Variable

For a random variable  $X \sim \mathcal{N}(\mu, \sigma^2)$ , standardizing  $X$  results in a transformed random variable that is normally distributed as well. Its expected value now is zero and its variance one. We thus obtain the standard normal distribution:

$$Z = \frac{X - \mu}{\sigma} \sim \mathcal{N}(0, 1)$$

Standardization allows to compute probabilities for an arbitrary normal distribution by means of the standard normal distribution.

#### Example 1.3.4

If  $X \sim \mathcal{N}(\mu, \sigma^2)$  with  $\mu = 2$  and  $\sigma^2 = 4$ , then

$$P(X \leq 5) = P\left(\frac{X - \mu}{\sigma} \leq \frac{5 - \mu}{\sigma}\right) = P\left(Z \leq \frac{5 - 2}{2}\right) = P(Z \leq 1.5) = \Phi(1.5) = 0.93$$



#### Example 1.3.5

If  $X \sim \mathcal{N}(\mu, \sigma^2)$  with  $\mu = 2$  and  $\sigma^2 = 4$ , then

$$P(X > 5) = 1 - P(X \leq 5) = 1 - \Phi(1.5) = 1 - 0.93 = 0.07$$



#### Example 1.3.6

For  $X \sim \mathcal{N}(\mu, \sigma^2)$  with  $\mu = 2$  and  $\sigma^2 = 4$ . What is the 90 % quantile  $\gamma$  of  $X$ ?

$$P(X \leq \gamma) = 0.9$$

$$\Rightarrow P\left(Z \leq \frac{\gamma - \mu}{\sigma}\right) = 0.9$$

$$\Rightarrow \Phi\left(\frac{\gamma - \mu}{\sigma}\right) = 0.9$$

Any statistics software yields

$$\frac{\gamma - \mu}{\sigma} = \Phi^{-1}(0.9) = 1.28$$

Solving for  $\gamma$  yields:

$$\gamma = \mu + 1.28\sigma = 2 + 1.28 \cdot 2 = 4.56$$



**Example 1.3.7**

For  $X \sim \mathcal{N}(\mu, \sigma^2)$  with  $\mu = 2$  and  $\sigma^2 = 4$ , what is  $P(|X| \leq 2)$ ?

$$\begin{aligned} P(|X| \leq 2) &= P(-2 \leq X \leq 2) = P(X \leq 2) - P(X \leq -2) \\ &= P\left(Z \leq \frac{2-2}{2}\right) - P\left(Z \leq \frac{-2-2}{2}\right) = P(Z \leq 0) - P(Z \leq -2) \\ &= \Phi(0) - \Phi(-2) = \Phi(0) - (1 - \Phi(2)) = 0.5 - (1 - 0.97) \\ &= 0.5 - 0.03 = 0.47 \end{aligned}$$


**Remarks:**

- i. We could also have calculated these probabilities without any standardization, by using the following arguments in Python : `norm.pdf(..., loc=2, scale=2)` and `norm.cdf(..., loc=2, scale=2)`

The question naturally arises of *why* random variables should be standardized. Prior to the computer era, the values of  $\Phi$  were listed in tables. Then one had to standardize normally distributed random variables, since it was impossible to create corresponding tables for all parameters  $\mu$  and  $\sigma^2$ .

- ii. However, we will still require the standardization scheme for transforming random variables in the context of  $t$ -distributions.



### 1.3.2. Non-Linear Transformations of Random Variables

For simplicity, we will initially focus on the case of a **quadratic transformation**

$$y = g(x) = x^2$$

**Example 1.3.8**

Let  $X$  be a uniformly distributed random variable, that is  $X \sim \text{Uniform}([a, b])$ . We define the random variable  $Y$  by applying a quadratic transformation to  $X$ :

$$Y = X^2$$

What is the expected value  $E[Y]$ ? We could now be tempted to first determine the density of  $Y$ ,  $f_Y(Y)$ , and then to calculate, on the basis of  $f_Y(y)$ , the expected value

$E[Y]$ . However, it turns out<sup>1</sup> that the calculation of the expected value of  $Y$  would be much easier:

$$E[Y] = \int_{-\infty}^{\infty} y f_Y(y) dy = \int_{-\infty}^{\infty} x^2 f_X(x) dx = \int_a^b x^2 \frac{1}{b-a} dx = \frac{a^2 + ab + b^2}{3}$$



### Example 1.3.9 Boltzmann Distribution - for the advanced reader

We denote by  $V$  the absolute value of the velocity of a gas molecule. In accordance with the kinetic theory of gases, the absolute value of the velocity of a gas molecule is random and follows the probability distribution defined by the following probability density

$$f_V(v) = 4\pi \left( \frac{m}{2\pi k_B T} \right)^{3/2} v^2 \exp\left(-\frac{mv^2}{2k_B T}\right),$$

where  $k_B$  denotes the Boltzmann constant,  $T$  is the temperature in Kelvin, and  $m$  denotes the mass of the gas molecule. On the basis of this probability density, we could now determine the expected value of the absolute velocity of a gas molecule, that is  $E[V]$ . In addition, we are also interested in the average kinetic energy  $Y = \frac{1}{2}mV^2$ , that is in  $E[Y]$ .

We now could attempt to determine the density of  $Y$ , that is  $f_Y(Y)$ , and then on the basis of  $f_Y(y)$ , calculate the expected value of the kinetic energy  $E[Y]$ . But the calculation of the expected value of the kinetic energy proceeds, analogous to the previous example, in a much more directed way:

$$E[Y] = \int_0^{\infty} \frac{1}{2}mv^2 f_V(v) dv = 2\pi m \left( \frac{m}{2\pi k_B T} \right)^{3/2} \int_0^{\infty} v^4 \exp\left(-\frac{mv^2}{2k_B T}\right) dv = \frac{3}{2}k_B T.$$

Calculations are in Appendix A.1. ◀

### General Transformations of a Random Variable

Let us consider the **general transformation** of a random variable  $X$

$$Y = g(X)$$

The cumulative distribution function and the density of  $Y$  are determined by the density distribution function of  $X$ . The following formula can always be used for

---

<sup>1</sup>This can be proven in a not particularly complex calculation, see the annex chapter

the calculation of the expected value

$$E[Y] = E[g(X)] = \int_{-\infty}^{\infty} g(x) f_X(x) dx$$

## 1.4. Functions of Several Random Variables

In the last section we discussed how the function of *one* random variable is distributed. But in a majority of applications, one is dealing not with one, but with *several* random variables. One usually measures the *same* variable several times. For example, one either has several individuals or one repeats measurements. In this section, we discuss how a function of several random variables is distributed.

We consider the measurements  $x_1, x_2, \dots, x_n$  as realizations of the random variables

$$X_1, \dots, X_n$$

It is often more convenient to consider  $X_i$  as the  $i$ th repetition of our random experiment instead of interpreting the  $n$  measurements as realizations of one single random variable  $X$ .

### Example 1.4.1

We record 20 measurements of the water pollution in a lake. Therefore, we have the following measurements

$$x_1, x_2, \dots, x_{20}$$

which represent the realizations of the random variables

$$X_1, X_2, \dots, X_{20}.$$

We assume that these 20 random variables all originate from the same probability distribution since these water samples were all taken from the same lake and the measurements were carried out using identical methods. We are interested in the average of these measurements and the distribution of this resulting random variable. To achieve this goal, we need a theory for functions of several random variables. ◀

Functions of the measured values  $x_1, x_2, \dots, x_n$  are of the form:

$$y = g(x_1, \dots, x_n)$$

This function has  $n$  independent input variables and a real number as output variable. If  $x_1, x_2, \dots, x_n$  are realizations of the random variables  $X_1, \dots, X_n$ , then  $y$  is a realization of the random variable

$$Y = g(X_1, \dots, X_n)$$

We are now particularly interested in the *sample total*

$$g(X_1, \dots, X_n) = S_n = X_1 + \dots + X_n = \sum_{i=1}^n X_i$$

and in the *sample mean*

$$g(X_1, \dots, X_n) = \bar{X}_n = \frac{1}{n}(X_1 + X_2 + \dots + X_n) = \frac{1}{n} \sum_{i=1}^n X_i = \frac{1}{n} S_n$$

The *population mean* of the data,  $\bar{x}_n$ , is therefore a realization of the random variable  $\bar{X}_n$ , the sample mean. Why are we interested in the distribution of the random variable  $\bar{X}_n$ ? The reason is, that knowing this distribution will enable us to work out the statistics for the population means of the data.

#### 1.4.1. Independence and i.i.d. Assumption

We often make the assumption that the random variables  $X_1, \dots, X_n$  are *mutually independent*. This assumption is appropriate when there are no common factors influencing the outcome of the different measurements and no “carry over” phenomena of one measurement to the next.

If the random variables  $X_1, \dots, X_n$  are *independent* and all follow the *same* distribution, we use the following notation

$$X_1, \dots, X_n \text{ i.i.d.}$$

The abbreviation i.i.d. stands for:

**i**ndependent, **i**dentically **d**istributed

We will usually work on the basis of the i.i.d. assumption, since we often deal with  $n$  repetitions of the same experiment. We leave it open as to which distribution the  $X_i$ 's follow. It is often a normal distribution, but it does not need to be so. The independence assumption plays a fundamental role in the derivation of the rules for the expected values and variances of sums. The relation

$$E[X_1 + X_2] = E[X_1] + E[X_2]$$

is always true, however

$$\text{Var}[X_1 + X_2] = \text{Var}[X_1] + \text{Var}[X_2]$$

only holds if  $X_1$  and  $X_2$  are independent.

## 1.4.2. Summary Statistics of $S_n$ and $\bar{X}_n$

### Example 1.4.2 Relative frequency of fair tosses of a coin

If you toss a fair coin only a few times, the ratio of tosses resulting in heads to the number of tosses resulting in tails is seldom exactly  $1/2$ . It is, however, a reasonable assumption that this ratio for very many tosses with a fair coin would yield approximately  $1/2$ .

The South African mathematician, John Kerich, tested this assumption while he was a prisoner of war during the Second World War. He tossed a coin 10 000 times and observed heads 5067 times. The sequential tosses of the coin are modelled as independent random variables  $X_i$ . The random variable assumes either the value 0 or 1, according to whether the  $i$ -th toss results in heads or tails. The relative frequency of heads in  $n$  trials is then

$$\bar{X}_n = \frac{1}{n} \sum_{i=1}^n X_i$$

In John Kerich's random experiment for  $\bar{x}_{10000}$  the value 0.5067 was obtained

$$\bar{x}_{10000} = 0.5067$$

If he had tossed the coin only  $n = 10$  times, a value of  $\bar{x}_{10} = 0.7$  or a value of  $\bar{x}_{10} = 0.3$  would not be very surprising, even if the coin was fair. However, the fairness of the coin would be strongly questioned if among 10 000 tosses only 3000 resulted in heads.



In this section, we assume that

$$X_1, \dots, X_n \stackrel{\text{i.i.d.}}{\sim} \text{cumulative distribution function } F$$

On account of the second "i" in i.i.d., each  $X_i$  has the same distribution and the same summary statistics:

$$E[X_i] = \mu \quad \text{and} \quad \text{Var}[X_i] = \sigma_X^2$$

The summary statistics of  $S_n$  and  $\bar{X}_n$  then follow from the general rules for expected values and variances of sums of random variables:

#### Summary Statistics of the Sample Total $S_n$

$$E[S_n] = E[X_1 + X_2 + \dots + X_n] = \sum_{i=1}^n E[X_i] = n\mu$$

$$\text{Var}[S_n] = \sum_{i=1}^n \text{Var}[X_i] = n \text{Var}[X_i]$$

$$\sigma(S_n) = \sqrt{n}\sigma_X$$

### Summary Statistics of the Sample Mean $\bar{X}_n$

$$E[\bar{X}_n] = E\left[\frac{X_1 + X_2 + \dots + X_n}{n}\right] = \frac{1}{n} \sum_{i=1}^n E[X_i] = \frac{1}{n} n E[X_i] = \mu$$

$$\text{Var}[\bar{X}_n] = \frac{1}{n^2} \sum_{i=1}^n \text{Var}[X_i] = \frac{1}{n^2} n \sigma_X^2 = \frac{\sigma_X^2}{n}$$

$$\sigma(\bar{X}_n) = \frac{\sigma_X}{\sqrt{n}}$$

The standard deviation of  $\bar{X}_n$  is called the *standard error* of the sample mean.

The standard deviation of the sample total,  $S_n$ , thus grows as  $n$  grows, but more slowly than the number of observations  $n$ . I.e. on a relative scale, we have a smaller scattering for larger  $n$ .

The expected value of the sample mean,  $\bar{X}_n$ , is therefore equal to the expected value of an individual random variable  $X_i$ , however, the *scattering decreases as  $n$  increases*.

### Law of Large Numbers

For  $n \rightarrow \infty$ , the scattering approaches zero. The *law of large numbers* states: for  $X_1, \dots, X_n$  i.i.d., we have

$$\bar{X}_n \longrightarrow \mu \quad \text{for } n \rightarrow \infty$$

### Standard Error

The standard deviation of the sample mean (*standard error*) is, however, *not* proportional to  $1/n$  but decreases only by the factor  $1/\sqrt{n}$

$$\sigma_{\bar{X}_n} = \frac{1}{\sqrt{n}} \sigma_X$$

To halve the *standard error*, one needs *four times* as many observations. This is also known as the  $\sqrt{n}$ -law.

### Example 1.4.3

In the case of a fair coin, we have

$$X_i \sim \text{Bernoulli}(\pi = 0.5)$$

with

$$E[X_i] = \pi = \frac{1}{2}, \quad \text{Var}[X_i] = \pi(1 - \pi) = \frac{1}{4}, \quad \sigma_{X_i} = \sigma_X = \sqrt{\pi(1 - \pi)} = \frac{1}{2}$$

In the case of John Kerich's experiment, one obtains for the summary statistic  $S_n$  with  $n = 10\,000$

$$\begin{aligned} E[S_{10000}] &= n E[X_i] = 10\,000 \cdot \frac{1}{2} = 5000 \\ \text{Var}[S_{10000}] &= n \text{Var}[X_i] = 10\,000 \cdot \frac{1}{4} = 2500 \\ \sigma(S_n) &= 50 \end{aligned}$$

The summary statistics of the relative frequency

$$\begin{aligned} E[\bar{X}_{10000}] &= E[X_i] = 0.5 \\ \text{Var}[\bar{X}_{10000}] &= \frac{1}{n} \text{Var}[X_i] = \frac{1}{10\,000} \cdot \frac{1}{4} = 0.000025 \\ \sigma(\bar{X}_n) &= \frac{\sigma_X}{\sqrt{n}} = \frac{0.5}{\sqrt{100}} = 0.005 \end{aligned}$$

The relative frequency of heads, observed by John Kerich, was 0.5067 and is therefore somewhat beyond one standard error given by 0.005.

If John Kerich repeated his experiment several times, always with  $n = 10\,000$  tosses, he could have expected for the standard deviation of the sample mean 0.005 on average. ◀

### 1.4.3. Distributions of $S_n$ and $\bar{X}_n$

In general, it is difficult to specify the distributions of  $S_n$  and  $\bar{X}_n$ . There are the following special cases, assuming  $X_1, \dots, X_n$  i.i.d.:

1. For  $X_i \in \{0, 1\}$ , we have

$$S_n \sim \text{Bin}(n, \pi) \quad \text{with} \quad \pi = P(X_i = 1)$$

2. For  $X_i \sim \text{Pois}(\lambda)$ , we have

$$S_n \sim \text{Pois}(n\lambda)$$

3. For  $X_i \sim \mathcal{N}(\mu, \sigma^2)$ , we have

$$S_n \sim \mathcal{N}(n\mu, n\sigma^2) \quad \text{and} \quad \bar{X}_n \sim \mathcal{N}\left(\mu, \frac{\sigma_X^2}{n}\right)$$

Even if the individual  $X_i$ 's do not follow a normal distribution, the distribution formulae defined under 3 astonishingly continue to apply approximately. This follows from the famous *Central Limit Theorem*<sup>2</sup>.

### Central Limit Theorem

If  $X_1, \dots, X_n$  i.i.d., for any distribution with expected value  $\mu$  and variance  $\sigma^2$ , the following relations hold:

$$S_n \approx \mathcal{N}(n\mu, n\sigma_X^2)$$

$$\bar{X}_n \approx \mathcal{N}(\mu, \sigma_X^2/n)$$

where the approximation generally improves as  $n$  increases. Moreover, the approximation also becomes more precise the more the distribution of the  $X_i$  follows a normal distribution  $\mathcal{N}(\mu, \sigma_X^2)$ .

Even if we do not know the distribution of the  $X_i$ , we now have an idea of the approximate distribution of  $S_n$  and  $\bar{X}_n$ . The central limit theorem is one of the reasons for the importance of the normal distribution.

As we know, the binomial distribution becomes “bell-shaped” for large  $n$ . The same applies to the Poisson distribution for increasing  $\lambda$ . One can therefore use the normal distribution to approximate the binomial distribution for large  $n$  (because the binomial distribution is an i.i.d. sum of Bernoulli distributions). This approach is referred to the *normal approximation* of the binomial distribution.

If

$$X \sim \text{Bin}(n, \pi)$$

then we have

$$E[X] = n\pi \quad \text{and} \quad \text{Var}[X] = n\pi(1 - \pi)$$

For large  $n$ , due to the central limit theorem, we can therefore describe the distribution of  $X$  approximately as a normal distribution with expected value  $E[X] = n\pi$  and variance  $\sigma^2 = n\pi(1 - \pi)$ . I.e. we have

$$X \sim \text{Bin}(n, \pi) \approx \mathcal{N}(n\pi, n\pi(1 - \pi))$$

and thus

$$P(X \leq x) \approx \Phi\left(\frac{x - n\pi}{\sqrt{n\pi(1 - \pi)}}\right)$$

---

<sup>2</sup>You will find a proof of this extremely important theorem in Chapter ?? of the Annex.

### Example 1.4.4

What is the probability that among 10 000 tosses of a fair coin, heads would appear in maximum 5100 cases?

The number of tosses of a coin in which heads appear follows a  $\text{Bin}(10\,000, 0.5)$  distribution. We approximate this distribution as a normal distribution, i.e.,

$$X \sim \mathcal{N}(\mu, \sigma^2)$$

with

$$\mu = 10\,000 \cdot 0.5 = 5000 \quad \text{and} \quad \sigma^2 = 10000 \cdot 0.5 \cdot (1 - 0.5) = 2500$$

And we obtain

$$X \sim \text{Bin}(10\,000, 0.5) \approx \mathcal{N}(5000, 2500)$$

We are interested in the probability

$$P(X \leq 5100) = \Phi\left(\frac{5100 - 5000}{\sqrt{2500}}\right) = \Phi(2) \approx 0.98$$

In [Python](#) the calculation proceeds as follows ([to R](#))

```
[1]: import math
      from scipy.stats import norm

      p_norm_5100 = norm.cdf(x=5100, loc=5000, scale = math.sqrt(2500))
      print(p_norm_5100)
```

0.9772498680518208

If we compare this previous approximate result with the “true” result, then we need to consider the distribution

$$X \sim \text{Bin}(10\,000, 0.5)$$

with

$$P(X \leq 5100) \approx 0.98$$

([to R](#))

```
[2]: from scipy.stats import binom

      p_binon_5100 = binom.cdf(k=5100, n=10000, p=.5)
      print(p_binon_5100)
```

0.9777871004771368

We conclude that the agreement between true and approximated values is very good.



**Example 1.4.5**

We draw  $n = 10$  random numbers  $X_i$ . These ten random variables are independent and for every  $i$ , we have

$$X_i \sim \text{Uniform}([0, 1])$$

What is the probability that the sum of the random numbers

$$S_{10} = \sum_{i=1}^{10} X_i$$

is greater than six?

I.e., we want to calculate

$$P(S_{10} > 6)$$

From Section 1.2.1 we know how to calculate the expected value and variance for every  $X_i$ :

$$\mathbb{E}[X_i] = \frac{1+0}{2} = 0.5 \quad \text{and} \quad \text{Var}[X_i] = \frac{(1-0)^2}{12} = \frac{1}{12}$$

It follows from the central limit theorem that:

$$S_n \approx \mathcal{N}(n \mathbb{E}[X_i], n \text{Var}[X_i]) = \mathcal{N}\left(5, \frac{10}{12}\right) = \mathcal{N}(5, 0.83)$$

Hence we come up with the following solution:

$$\begin{aligned} P(S_n > 6) &= 1 - P(S_n \leq 6) = 1 - P\left(\frac{S_n - 5}{\sqrt{0.83}} \leq \frac{6 - 5}{\sqrt{0.83}}\right) = 1 - P(Z \leq 1.1) \\ &= 1 - \Phi(1.1) = 1 - 0.86 = 0.14 \end{aligned}$$



For an exact formulation of the central limit theorem, one considers the standardized random variable

$$Z_n = \frac{\sqrt{n}(\bar{X}_n - \mu)}{\sigma_X}$$

which is approximately  $\mathcal{N}(0, 1)$  distributed, that is for all  $x$ ,

$$\lim_{n \rightarrow \infty} P(Z_n \leq x) = \Phi(x)$$

holds. The central limit theorem also applies for discrete random variables  $X_i$ .

## Summary Distributions

Discrete Distributions				
Distribution	Cum. Distribution Function	Probability Mass Function	E[X]	Var[X]
Binomial Distribution $\text{Bin}(n, \pi)$	$F(x) = \sum_{i=0}^x \binom{n}{i} \pi^i (1-\pi)^{n-i}$	$P(X = x) = \binom{n}{i} \pi^i (1-p)^{n-i}$	$n\pi$	$n\pi(1-\pi)$
Poisson Distribution $\text{Pois}(\lambda)$	$F(x) = \sum_{i=0}^x e^{-\lambda} \frac{\lambda^x}{x!}$	$P(X = x) = e^{-\lambda} \frac{\lambda^x}{x!}$	$\lambda$	$\lambda$

Continuous Distributions				
Distribution	Cum. Distribution Function	Probability Density Function	E[X]	Var[X]
Uniform Distribution $\text{Unif}(a, b)$	$F(x) = \begin{cases} 0 & \text{if } x < a \\ \frac{x-a}{b-a} & \text{if } a \leq x \leq b \\ 1 & \text{if } x > b \end{cases}$	$f(x) = \begin{cases} \frac{1}{b-a} & \text{if } a \leq x \leq b \\ 0 & \text{otherwise} \end{cases}$	$\frac{a+b}{2}$	$\frac{(b-a)^2}{12}$
Exponential Distribution $\text{Exp}(\lambda)$	$F(x) = \begin{cases} 1 - e^{-\lambda x} & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$	$f(x) = \begin{cases} \lambda \cdot e^{-\lambda x} & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$	$\frac{1}{\lambda}$	$\frac{1}{\lambda^2}$
Normal Distribution $\mathcal{N}(\mu, \sigma^2)$	$F(x) = \int_{-\infty}^x f(y) dy$	$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$	$\mu$	$\sigma^2$

## Conceptual Objectives

You should be able...

- to explain the concept of a probability density function and of a cumulative distribution function.
- to define the uniform distribution, the exponential distribution and the normal distribution.
- to standardize the normal distribution.
- to calculate the expected value, variance and quantile for linear transformed random variables
- to calculate the expected value and the variance for sums of independent random variables.
- to explain the law of large numbers, the  $\sqrt{n}$  law, and the central limit theorem
- to apply the central limit theorem to sums of random variables.
- to explain the difference between the standard deviation of a single observation  $X_i$  and the standard error of a measurement series.
- to calculate the standard error of a measurement series.

## Computational Objectives

You should be able...

- to compute the probability density values of the uniform, exponential and normal distribution by means of `uniform.pdf()`, `expon.pdf()` and `norm.pdf()` with the corresponding parameter values.
- to compute the cumulative distribution function values of the uniform, exponential and normal distribution by means of `uniform.cdf()`, `expon.cdf()` and `norm.cdf()` with the corresponding parameter values.
- to generate random variables that are distributed according to the uniform, exponential and normal distribution by means of `uniform.rvs()`, `expon.rvs()` and `norm.rvs()` with the corresponding parameter values.
- to calculate probabilities for uniform, exponential and normal distributions

# Chapter 2.

## Statistics for Measurement Data

We must be careful not to confuse data with the abstractions we use to analyze them.

---

(William James)

### 2.1. Assess the Normal Distribution Assumption

In Chapter 1, we assumed that we know the probability distribution of a random variable, for example  $\mathcal{N}(3, 2)$ . On the basis of such a probability distribution, we have calculated different summary statistics and probabilities.

In practice however we first have to decide on the basis of (only few) data for a distribution family, as for instance the normal distribution, in order to model a random event and calculate probabilities for such an event. Let us assume that we have a data set  $x_1, \dots, x_n$  with  $n$  observations. Selecting a distribution family can be based on expertise ("which has proven to be effective so far") or on the basis of scientific (physical) arguments.

Whether a distribution family fits a concrete data set can be verified qualitatively by using graphical methods. For example, we could look at how well the (normalized) histogram of the data fits the density of our model distribution, e.g., a specific normal distribution. However, as we will discuss in the following, deviations can be detected much better by comparing the quantiles of two distributions.

A Q-Q plot represents a graphical method to verify how well a distribution family fits a data set. There are also quantitative methods on the basis of test statistics to quantify how well a (presumed) distribution family fits a data set. For example, we refer to the chi-square test or to the Kolmogorov-Smirnov test.

### 2.1.1. Q-Q Plot

The idea underlying a *Q-Q plot* (quantile-quantile plot) consists in plotting the empirical quantiles against the theoretical quantiles of the presumed model distribution.

#### Q-Q Plot

(i) For

$$\alpha_k = \frac{k - 0.5}{n} \quad \text{with } k = 1, \dots, n$$

namely for

$$\alpha_1 = \frac{0.5}{n}, \dots, \alpha_n = \frac{n - 0.5}{n}$$

calculate the corresponding theoretical quantiles of the model distribution

$$q(\alpha_k) = F^{-1}(\alpha_k)$$

where  $n$  denotes the number of data points.

(ii) Determine the empirical  $\alpha_k$ -quantiles, which correspond to the ordered observations

$$x_{(1)} < x_{(2)} < \dots < x_{(n)}$$

Recall  $x_{(\alpha_k \cdot n + 0.5)} = x_{(k)}$

(iii) Plot the empirical quantiles  $x_{(k)}$  on the  $y$ -axis against the theoretical quantiles  $q(\alpha_k)$  on the  $x$ -axis.

If the observations have been generated according to the model distribution, these points should lie approximately on the bisector  $y = x$ .

#### Remarks:

- i. Recall that we denote by  $x_{(1)}$  the smallest observation in the data set, by  $x_{(n)}$  the largest.
- ii. Why do we calculate the theoretical quantiles for the values  $\alpha_k = \frac{k-0.5}{n}$ ? Recall that we had defined the empirical  $\alpha$ -quantile as the observation  $x_{(\alpha \cdot n + 0.5)}$ , where we round off the value  $\alpha \cdot n + 0.5$  ( $\alpha$  and  $n$  are known). If we now consider the observation  $x_{(k)}$  as the  $k$ th largest observation, then  $k$  corresponds to the rounded value of  $(\alpha \cdot n + 0.5)$ . Therefore,  $x_{(k)}$  is the  $\alpha_k$  quantile with  $\alpha_k = \frac{k-0.5}{n}$ .



**Example 2.1.1 Concrete Compression Strength**

Measurements of the concrete compression strength were carried out on  $n = 20$  different samples. We want to check how well the data can be described with a normal distribution.

The values are listed in Table 2.1 according to their size.  $\alpha_k = \frac{k-0.5}{n}$  is calculated for each measurement. The first value thus corresponds to the empirical 2.5 % quantile, the  $k = 11$ th measurement corresponds approximately to the median and the  $k = 16$ th measurement corresponds to the 75 % quantile.

$k$	$x_{(k)}$	$\alpha_k = (k - 0.5)/n$	$q_{\alpha_k}$ for $\mathcal{N}(32.7, 4.15^2)$	$\Phi^{-1}(\alpha_k)$
1	24.4	0.0250	24.5	-1.96
2	27.6	0.075	26.7	-1.44
3	27.8	0.125	27.9	-1.15
4	27.9	0.175	28.8	-0.935
5	28.5	0.225	29.5	-0.755
6	30.1	0.275	30.2	-0.600
7	30.3	0.325	30.8	-0.453
8	31.7	0.375	31.3	-0.319
9	32.2	0.425	31.9	-0.189
10	32.8	0.475	32.4	-0.0627
11	33.3	0.525	32.9	0.0627
12	33.5	0.575	33.4	0.189
13	34.1	0.625	34.0	0.319
14	34.6	0.675	34.5	0.454
15	35.8	0.725	35.1	0.598
16	35.9	0.775	36.0	0.755
17	36.8	0.825	36.5	0.935
18	37.1	0.875	37.4	1.15
19	39.2	0.925	38.6	1.44
20	39.7	0.975	40.8	1.96

**Table 2.1.:** 20 measurements of the concrete compression strength.

## Chapter 2. Statistics for Measurement Data

We assume that these measurements are normally distributed. We estimate<sup>1</sup> the parameter  $\mu$  of the normal distribution by means of the empirical mean value and the parameter  $\sigma$  by means of the empirical standard deviation and find

$$\hat{\mu} = 32.7 \quad \text{and} \quad \hat{\sigma} = 4.15$$

We next calculate the corresponding  $\alpha_k$ -quantile for every  $\alpha_k$ , that is,  $\Phi^{-1}(\alpha_k)$  for the normal distribution

$$\mathcal{N}(32.7, 4.15^2)$$

If our data set actually follows a normal distribution, then we would expect that the empirical quantiles of our data set and the quantiles of the normal distribution are approximately equal and thus should lie on the bisector  $y = x$ .

In [Python](#), we generate a Q-Q plot as follows. ([to R](#))

```
[1]: import matplotlib.pyplot as plt
import numpy as np
from pandas import Series
from scipy.stats import norm

# Concrete compression strength, data ordered by size
x = Series([24.4, 27.6, 27.8, 27.9, 28.5,
            30.1, 30.3, 31.7, 32.2, 32.8,
            33.3, 33.5, 34.1, 34.6, 35.8,
            35.9, 36.8, 37.1, 39.2, 39.7])

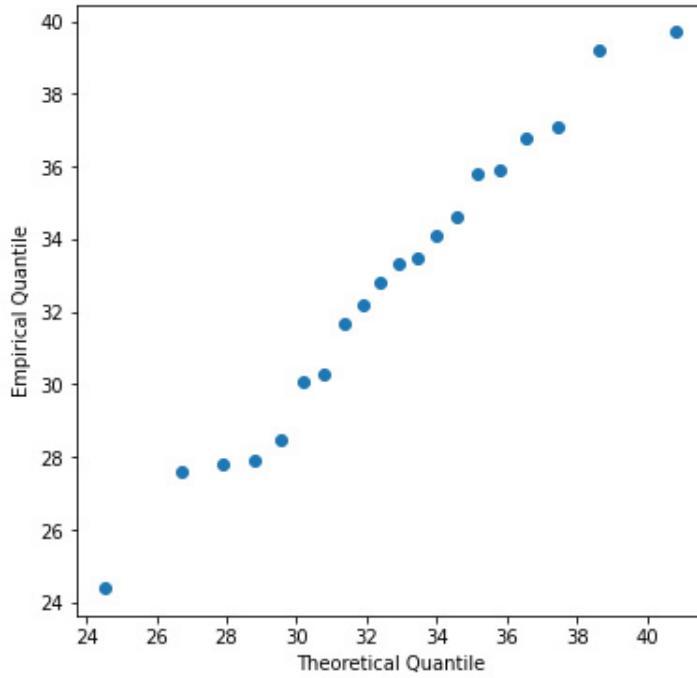
# Quantiles
alphak = (np.arange(1, x.size + 1) - 0.5) / x.size
quantile_theor = norm.ppf(q=alphak, loc=x.mean(), scale=x.std())
quantile_empir = np.sort(x)

# Plot figure
plt.figure(figsize=(6, 6))
plt.plot(quantile_theor, quantile_empir, "o")
plt.xlabel("Theoretical Quantile")
plt.ylabel("Empirical Quantile")

plt.show()
```

---

<sup>1</sup>In Session 2.2, we will justify this plausible procedure for the parameter estimation in much greater detail.



**Figure 2.1.:** Q-Q-Plot on concrete compression strength

The plot is displayed in figure B.1.

The concept of a Q-Q plot is illustrated in Figure 2.2. We consider quantiles of the presumed model and the empirical quantiles of our data set. If they are similar to each other, then we conclude that our data set follows the presumed model distribution.



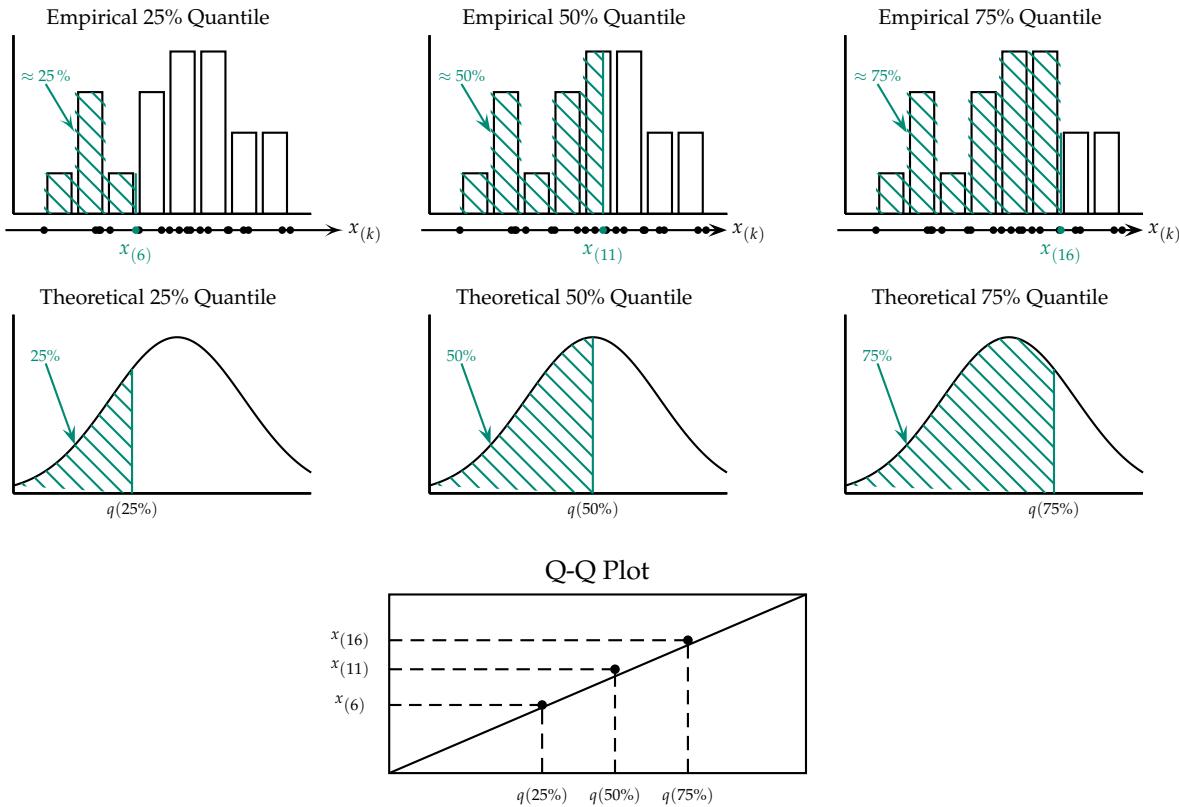
### 2.1.2. Normal Plot

Often, one is not simply interested in a specific distribution, but rather in an entire class of distributions. For example, we would like to verify whether a data set follows a normal distribution with arbitrary  $\mu$  and  $\sigma$ . The *normal plot* represents a special case of a Q-Q plot: we call a Q-Q plot a *normal plot* if the empirical quantiles are plotted against the quantiles of the standard normal distribution,  $\mathcal{N}(0, 1)$ .

Let the observed measurements be realizations of  $X \sim \mathcal{N}(\mu_X, \sigma_X^2)$  and  $X$  be a transformed variable

$$X = \mu_X + \sigma_X \cdot Y$$

## Chapter 2. Statistics for Measurement Data



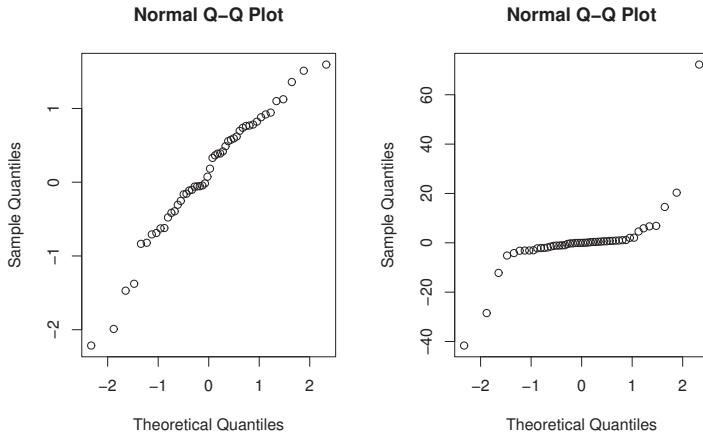
**Figure 2.2.:** Empirical and theoretical quantiles are displayed in a Q-Q plot on the basis of the data set for the concrete compression strength.

with  $Y \sim \mathcal{N}(0, 1)$ . Then the following relation applies for the quantiles of  $X$ :

$$\underbrace{q_X(\alpha)}_{y(x)} = \underbrace{\mu_X}_{a} + \underbrace{\sigma_X \cdot q_Y(\alpha)}_{b \cdot x},$$

where  $q_Y(\alpha) = \Phi^{-1}(\alpha)$  is the  $\alpha$ -quantile of the standard normal distribution (see rule (iii) for linear transformations of random variables in Chapter 1.3). If we generate a normal plot in this case, the points in the normal plot will lie approximately on a straight line having axis intercept  $\mu$  and slope  $\sigma$ . Figure 2.3 displays two normal plots: the one in the left-hand panel originates from a normal distribution. The dots lie more or less on a straight line. The normal plot displayed in the right-hand panel was generated on the basis of an extremely long-tailed distribution. In this case, the dots do not lie on a straight line at all. At the right extremity of the distribution, the points lie well above the straight line, and at the left extremity, they lie well below the straight line.

In practice, dots never perfectly lie on a straight line in a normal plot. Figure 2.4 helps to improve your intuition of what is an acceptable deviation from the straight line. These normal plots were generated on the basis of nine data sets (50 observa-



**Figure 2.3.** Left: Normal plot for 50 realizations of  $\mathcal{N}(0, 1)$ . The dots lie on a straight line. Right: Normal plot for 50 realizations of a distribution which is very long-tailed. The dots do not lie on a straight line.

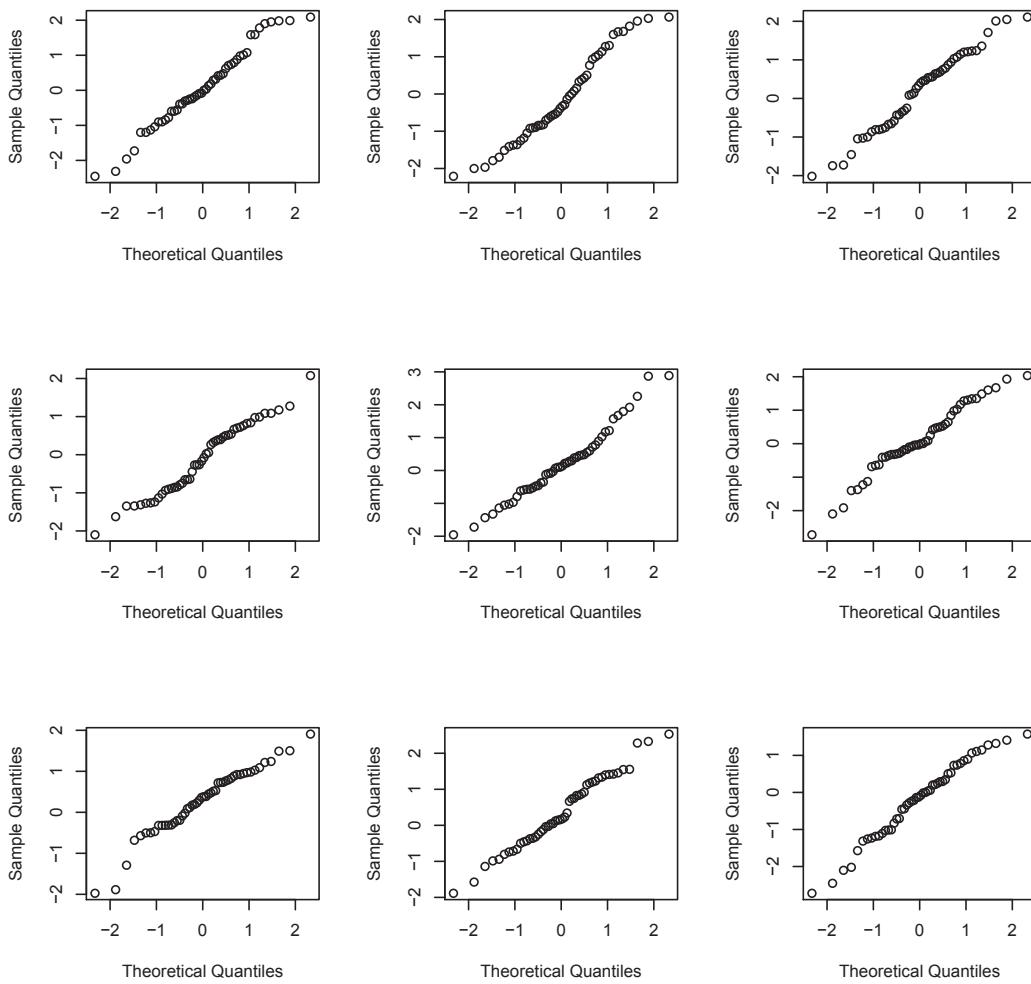
tions each) which are all simulated from a standard normal distribution. Deviations to this extent can therefore be expected if the data really originate from a normal distribution.

## 2.2. Parameter Estimation for Continuous Probability Distributions

### 2.2.1. Method of Moments

We have seen in the last chapter how we can verify whether a distribution family fits a data set. We now would like to estimate the parameters of a distribution. To this goal, we will discuss the method of moments.

- We consider our data measurements  $x_1, x_2, \dots, x_n$  as realizations of random variables  $X_1, X_2, \dots, X_n$  originating from the same known distribution.
- We calculate the expected value  $E[X]$  and solve the equation for the unknown parameter that we intend to estimate.
- We replace the expected value with its counterpart, the empirical mean value and thus obtain an estimate of the unknown parameter. A method of moments estimate of the standard deviation is the empirical standard deviation.



**Figure 2.4.:** Normal plots on the basis of nine data sets (50 observations each), which are all simulated from a standard normal distribution. These plots illustrate to which extent points may deviate from a straight line if the data originate from a normal distribution. If the deviation from a straight line is considerably more pronounced, then we conclude that the data is not normally distributed.

### Example 2.2.1 Concrete Compression Strength

For the measurement of the concrete compression strength, we had the presumption that the data are normally distributed. We were also able to confirm this with a Q-Q plot. Hence, the random variable  $X$  for the concrete compression strength follows a normal distribution:  $X \sim \mathcal{N}(\mu, \sigma^2)$  with parameters  $\mu$  and  $\sigma$ .

We now want to estimate  $\mu$  by means of the method of moments. For a normally

## Chapter 2. Statistics for Measurement Data

distributed  $X$ , we have  $E[X] = \mu$ . Thus, we simply need to replace the expected value with the empirical value:

$$\mu = E[X] \Rightarrow \hat{\mu} = \bar{x}_n = \frac{x_1 + x_2 + \dots + x_n}{n} = \frac{653.3}{20} = 32.7$$

To estimate the standard deviation  $\sigma$  by means of the method of moments, we need the following relation:

$$\begin{aligned}\sigma^2 &= E[X^2] - E[X]^2 \\ &= E[X^2] - \mu^2\end{aligned}$$

If we estimate  $E[X]$  by  $\bar{x}_n$  and  $E[X^2]$  by  $\frac{1}{n} \sum_{i=1}^n x_i^2$ , the following equation system results

$$\begin{aligned}\hat{\mu} &= \bar{x}_n \\ \hat{\mu}^2 + \hat{\sigma}^2 &= \frac{1}{n} \sum_{i=1}^n x_i^2\end{aligned}$$

Its solution is provided by

$$\begin{aligned}\hat{\mu} &= \bar{x}_n \\ \hat{\sigma}^2 &= \frac{\sum_{i=1}^n x_i^2 - n \left( \frac{1}{n} \sum_{i=1}^n x_i \right)^2}{n} = \frac{\sum_{i=1}^n (x_i - \bar{x}_n)^2}{n}\end{aligned}$$

This results in the following estimate of the standard deviation for our data set

$$\begin{aligned}\hat{\sigma} &= \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \hat{\mu})^2} \\ &= \sqrt{\frac{1}{20} \sum_{i=1}^{20} (x_i - 32.7)^2} \\ &= 4.04\end{aligned}$$



The method of moments estimate however is not *unbiased*, since the factor  $\frac{1}{n}$  should be replaced by  $\frac{1}{n-1}$ .

### Example 2.2.2 Exponential distribution

We measure the lifetime of an electronic device that is used in the motor control of a car. A test run yields the following measurements (in hours):

$$x_1 = 11.96, x_2 = 5.03, x_3 = 67.40, x_4 = 16.07, x_5 = 31.50, x_6 = 7.73, x_7 = 11.10, x_8 = 22.38$$

The lifetime typically follows an exponential distribution with parameter  $\lambda$ . The moment estimate of  $\lambda$  is:

$$\mathbb{E}[X] = \frac{1}{\lambda} \Rightarrow \hat{\lambda} = \frac{1}{\bar{x}} = \frac{1}{21.6} = 0.0462$$



## 2.2.2. Method of Maximum Likelihood

We have  $n$  observations

$$X_1 = x_1, X_2 = x_2, \dots, X_n = x_n$$

We assume that these observations are i.i.d.. In the case of a discrete probability distribution, the probability that these  $n$  observations (events) actually have occurred can be expressed as follows

$$\begin{aligned} P((X_1 = x_1) \cap (X_2 = x_2) \cap \dots \cap (X_n = x_n)) &= P(X_1 = x_1) \cdot P(X_2 = x_2) \cdots P(X_n = x_n) \\ &= \prod_{i=1}^n P(X_i = x_i) \end{aligned}$$

The probability that the  $n$  independent random variables  $x_1, x_2, \dots, x_n$  are observed, depends on the parameter  $\theta$ , which we wish to estimate. The corresponding *likelihood function*  $L(\theta)$  is therefore given by

$$\begin{aligned} L(\theta) &= P(X_1 = x_1 \mid \theta) \cdot P(X_2 = x_2 \mid \theta) \cdots P(X_n = x_n \mid \theta) \\ &= \prod_{i=1}^n P(X_i = x_i \mid \theta) \end{aligned}$$

where  $P(X_i = x_i \mid \theta)$  denotes the probability mass function that the value  $x_i$  has been observed, given the parameter value  $\theta$ . The underlying concept of the maximum likelihood method is to estimate the parameter  $\theta$  in such a way that the likelihood is maximized, that is, that it makes the observed data the “most likely” or “most probable”. We are thus seeking for the maximum of the likelihood function  $L(\theta)$ .

In the following, we will consider continuous probability distributions along with their probability density function  $f(x; \theta)$ . Assuming the density function  $f(x; \theta)$  that depends on  $\theta$  and assuming our observations are independent, the probability that the first observation  $x_1$  falls into the interval  $[x_1, x_1 + dx_1]$ , is  $f(x_1; \theta) dx_1$ , the probability that the second observation  $x_2$  falls into the interval  $[x_2, x_2 + dx_2]$ , is  $f(x_2; \theta) dx_2$ , etc.. Hence, the probability that each observation  $x_i$  falls into its corresponding interval  $[x_i, x_i + dx_i]$ , is given by

$$\prod_{i=1}^n f(x_i; \theta) dx_i$$

If the assumed probability density function  $f(x_i; \theta)$  and the parameter value of  $\theta$  are correct, we expect a high probability for the actually observed data to occur. If the parameter value  $\theta$  is far away from the true parameter value, the above function would yield a small value. As the (infinitesimal) intervals  $dx_i$  do not depend on the parameter value  $\theta$ , we simply omit them in the *likelihood function*

$$L(\theta) = \prod_{i=1}^n f(x_i; \theta)$$

where  $f(x_i; \theta)$  denotes a continuous probability function. To estimate the parameter value of  $\theta$ , we simply maximize  $L(\theta)$ .

### Example 2.2.3 Normal Distribution; $\sigma^2$ known

Let  $X$  be normally distributed with unknown  $\mu$  and known  $\sigma^2$ . The likelihood function for the observed measurements  $x_1, x_2, \dots, x_n$  is given by

$$L(\mu) = \prod_{i=1}^n \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(x_i - \mu)^2}{2\sigma^2}}$$

It is easier to maximize the *log likelihood* function:

$$\log(L(\mu)) = -\frac{n}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma^2} \sum_{i=1}^n (x_i - \mu)^2$$

The derivative with respect to the unknown parameter  $\mu$  yields:

$$\frac{d \log(L(\mu))}{d\mu} = \frac{1}{\sigma^2} \sum_{i=1}^n (x_i - \mu)$$

If we set this derivative equal to 0 and solve for the unknown  $\mu$ , we obtain the *maximum likelihood estimate*

$$\hat{\mu} = \frac{\sum_{i=1}^n x_i}{n} = \bar{x}_n$$

Hence, the maximum likelihood estimate of  $\mu$  is given by the empirical mean value. Thus, the maximum likelihood estimate is identical to the method of moments estimate. ◀

#### Example 2.2.4 Normal Distribution; $\sigma^2$ unknown

Let  $X$  be normally distributed with unknown  $\mu$  and variance  $\sigma^2$ . The likelihood function for the observations  $x_1, x_2, \dots, x_n$  is given by:

$$L(\mu, \sigma^2) = \prod_{i=1}^n \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(x_i - \mu)^2}{2\sigma^2}}$$

The log likelihood function is

$$\log(L(\mu, \sigma^2)) = -\frac{n}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma^2} \sum_{i=1}^n (x_i - \mu)^2$$

We now aim at estimating  $\mu$  and  $\sigma^2$  that maximize the log likelihood function. We therefore calculate the partial derivatives of the log likelihood function with respect to both unknown parameters and set the resulting expressions equal to 0:

$$\begin{aligned} \frac{\partial \log(L(\mu, \sigma^2))}{\partial \mu} &= \frac{1}{\sigma^2} \sum_{i=1}^n (x_i - \mu) \stackrel{!}{=} 0 \\ \frac{\partial \log(L(\mu, \sigma^2))}{\partial (\sigma^2)} &= -\frac{n}{\sigma^2} + \frac{1}{2\sigma^4} \sum_{i=1}^n (x_i - \mu)^2 \stackrel{!}{=} 0 \end{aligned}$$

The solutions of this equation system represent our maximum likelihood estimates

$$\hat{\mu} = \bar{x}, \quad \hat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2$$

The maximum likelihood estimate of  $\sigma^2$  is identical to the method of moments estimate. The maximum likelihood estimate of  $\sigma^2$  thus is *biased*, as well. ◀

#### Example 2.2.5 Exponential Distribution

Let  $X$  be an exponentially distributed random variable with parameter  $\lambda$ . The likelihood function for a given data set  $x_1, x_2, \dots, x_n$  is given by

$$L(\lambda) = \prod_{i=1}^n \lambda e^{-\lambda x_i}$$

The log likelihood function is

$$\log(L(\lambda)) = n \log(\lambda) - \lambda \sum_{i=1}^n x_i$$

If we calculate the derivative of the log likelihood function with respect to  $\lambda$  and set it equal to 0, then we obtain

$$\frac{d \log(L(\lambda))}{d\lambda} = \frac{n}{\lambda} - \sum_{i=1}^n x_i \stackrel{!}{=} 0$$

The maximum likelihood estimate  $\hat{\lambda}$  thus corresponds to the solution of the previous equation

$$\hat{\lambda} = \frac{n}{\sum_{i=1}^n x_i} = \frac{1}{\bar{x}}$$



## 2.3. Statistical Tests and Confidence Interval for Normally Distributed Data

Let the observations  $x_1, \dots, x_n$  be realizations of

$$X_1, \dots, X_n \text{ i.i.d. } \sim \mathcal{N}(\mu, \sigma_X^2).$$

Think for example of 10 measurements of a toxic substance concentration. The summary statistics of the random variable  $X_i$  are

$$E[X_i] = \mu \quad \text{and} \quad \text{Var}[X_i] = \sigma_X^2$$

Typically, these (and other) summary statistics are unknown and one wants to draw conclusions from the data with regard to them. The (point) estimates of the expected value and variance are:

$$\begin{aligned}\hat{\mu} &= \frac{1}{n} \sum_{i=1}^n X_i \\ \hat{\sigma}_X^2 &= \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X}_n)^2\end{aligned}$$

Note that the estimates here are specified as functions of random variables  $X_1, \dots, X_n$ : in particular,  $\hat{\mu}$  and  $\hat{\sigma}_X^2$  are themselves random variables. Properties of the distribution of  $\hat{\mu}$  were discussed in Section 1.4.

Whenever we record a new measurement series for the same experiment, we will obtain different values (realizations) of  $\hat{\mu}$  and  $\hat{\sigma}_X^2$ . It is therefore of great interest to specify an interval for the estimate  $\hat{\mu}$  that contains the true value of  $\mu$  with a certain probability. We thus aim to define a *confidence interval* for a parameter estimate.

Furthermore, we may wonder whether a realization of  $\mu$  is compatible with an assumed value  $\mu_0$ . The corresponding question is answered by a *statistical test*.

### 2.3.1. $z$ -Test ( $\sigma_X$ known)

We assume that the observed data points  $x_1, \dots, x_n$  are realizations of

$$X_1, \dots, X_n \text{ i.i.d.} \sim \mathcal{N}(\mu, \sigma_X^2).$$

Moreover, we assume that  $\sigma_X^2$  is known. The  $z$ -test for the parameter  $\mu$  can be carried out as follows

1. **Model:**  $X_i$  is a continuous measurement variable;

$$X_1, \dots, X_n \text{ i.i.d.} \sim \mathcal{N}(\mu, \sigma_X^2), \quad \sigma_X \text{ known}$$

2. **Null hypothesis:**

$$H_0 : \mu = \mu_0$$

**Alternative:**

$$H_A : \mu \neq \mu_0 \quad (\text{or } < \text{ or } >)$$

3. **Test statistic:**

$$Z = \frac{(\bar{X}_n - \mu_0)}{\sigma_{\bar{X}_n}} = \frac{(\bar{X}_n - \mu_0)}{\sigma_X / \sqrt{n}} = \frac{\text{observed} - \text{expected}}{\text{standard error}}$$

**Null distribution (assuming  $H_0$  is true):**

$$Z \sim \mathcal{N}(0, 1)$$

4. **Significance level:**

$$\alpha$$

5. **Rejection region for the test statistic:**

$$K = (-\infty, z_{\frac{\alpha}{2}}] \cup [z_{1-\frac{\alpha}{2}}, \infty) \text{ with } H_A : \mu \neq \mu_0,$$

$$K = (-\infty, z_\alpha] \text{ with } H_A : \mu < \mu_0,$$

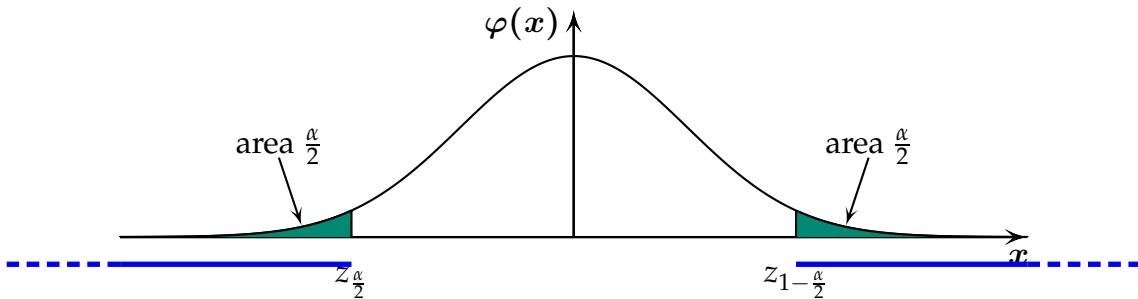
$$K = [z_{1-\alpha}, \infty) \text{ with } H_A : \mu > \mu_0,$$

where

$$z_{\alpha/2} = \Phi^{-1}(\alpha/2)$$

### 6. Test decision:

Check whether the observed value of the test statistic falls into the rejection region



**Figure 2.5.:** Density function of the test statistic  $Z$  with rejection region for the two-sided  $z$ -test at significance level  $\alpha$ . Note  $z_{\alpha/2} = -z_{1-\alpha/2}$ , where  $z_{\alpha/2} = \Phi^{-1}(\alpha/2)$ .

The  $z$ -test is based on *several* observations. Hence, the realization of the test statistic  $z$  summarizes all observations by representing their arithmetic mean value.

#### Example 2.3.1

We measure the fusion heat with a particular method and find for the empirical mean value of the  $n = 13$  measurements  $80.02 \text{ cal/g}$ . Let us assume that we know from previous studies that the standard deviation of our measuring instrument is  $\sigma_X = 0.01$ . Is it plausible that the fusion heat is exactly  $80.00 \text{ cal/g}$ ? We perform a  $z$ -test for this purpose:

1. **Model:**  $X_i$  is a continuous measurement variable;

$$X_1, \dots, X_n \text{ i.i.d. } \sim \mathcal{N}(\mu, \sigma_X^2), \quad \sigma_X = 0.01 \text{ known}, \quad n = 13$$

2. **Null hypothesis:**

$$H_0 : \mu = \mu_0 = 80.00$$

**Alternative:**

$$H_A : \mu \neq \mu_0$$

3. **Test statistic:**

$$Z = \frac{\sqrt{n}(\bar{X}_n - \mu_0)}{\sigma_X}$$

**Null distribution (assuming  $H_0$  is true):**

$$Z \sim \mathcal{N}(0, 1)$$

4. Significance level:

$$\alpha = 0.05$$

5. Rejection region for the test statistic:

$$K = (-\infty, z_{\frac{\alpha}{2}}] \cup [z_{1-\frac{\alpha}{2}}, \infty) \quad \text{with} \quad H_A : \mu \neq \mu_0,$$

Given  $\alpha = 0.05$ , Python yields the following 2.5% quantile of the standard normal distribution. (to R)

```
[1]: from scipy.stats import norm
q_25 = norm.ppf(0.025)
print(q_25)
```

-1.9599639845400545

$$z_{\frac{\alpha}{2}} = \Phi^{-1}\left(\frac{\alpha}{2}\right) = \Phi^{-1}(0.025) = -1.96$$

The following rejection region for the test statistic results

$$K = (-\infty, -1.96] \cup [1.96, \infty)$$

6. Test decision:

On the basis of the  $n = 13$  measurements we find for the empirical mean

$$\bar{x}_n = 80.02$$

Hence the value for the test statistic is

$$z = \frac{\sqrt{13}(80.02 - 80.00)}{0.01} = 7.211$$

Since the observed value of the test statistic falls into the rejection region, we reject the null hypothesis at the 5 % significance level.

**Remarks:**

- i. Standardizing the test statistic is in principle unnecessary because of the technical aids provided by computer software. ♦

1. **Model:**  $X_i$  is a continuous measured variable;

$$X_1, \dots, X_n \text{ i.i.d.} \sim \mathcal{N}(\mu, \sigma_X^2), \quad \sigma_X = 0.01 \text{ known}, n = 13$$

**2. Null hypothesis:**

$$H_0 : \mu = \mu_0 = 80.00$$

**Alternative:**

$$H_A : \mu \neq \mu_0$$

**3. Test statistic:**

The mean value of the measurements

$$T : \bar{X}_n$$

**Null distribution (assuming  $H_0$  is true):**

$$T \sim \mathcal{N} \left( \mu_0, \frac{\sigma_X^2}{n} \right) = \mathcal{N} \left( 80, \frac{0.01^2}{13} \right)$$

**4. Significance level:**

$$\alpha = 0.05$$

**5. Rejection region for the test statistic:**

$$K = (-\infty, c_u] \cup [c_o, \infty) \quad \text{with} \quad H_A : \mu \neq \mu_0,$$

Python yields the quantiles of interest for given  $\alpha = 0.05$ : ([to R](#))

```
[2]: import numpy as np
rej_reg_5 = norm.interval(alpha=0.95, loc=80, scale=0.01/np.sqrt(13))
print(rej_reg_5)
```

(79.9945640379659, 80.0054359620341)

In this way, we obtain the rejection region for the test statistic:

$$K = (-\infty, 79.99] \cup [80.01, \infty)$$

**6. Test decision:**

On the basis of the  $n = 13$  measurements, we calculate

$$\bar{x}_n = 80.02$$

The observed value falls into the rejection region of the test statistic. Hence, the null hypothesis at the 5 % significance level is rejected.



The terms *type I error*, *type II error* and *power* are defined as follows:

### Type I and II Error and Power

type I error = erroneous rejection of  $H_0$ , although  $H_0$  is true

type II error( $\mu$ ) = erroneous retention of  $H_0$  in case ( $\mu \in H_A$ ) is true

$$\text{Power}(\mu) = 1 - P(\text{type II error}(\mu)) = P(\text{rejection of } H_0 \text{ in case } \mu \in H_A \text{ is true})$$

The probability of a type I error is exactly equal to  $\alpha$ . For  $\mu \in H_A$ , one can interpret power( $\mu$ ) as the probability that one correctly detects  $H_A$  if  $\mu \in H_A$  is true. It then applies for a test statistic  $T$  and a corresponding rejection region  $K$  that:

$$\begin{aligned} P_{\mu_0}(T \in K) &= \alpha \\ P_{\mu}(T \in K) &= \text{power}(\mu) \end{aligned}$$

### 2.3.2. $t$ -Test ( $\sigma_X$ unknown)

We assume that our measurements are realizations of

$$X_1, \dots, X_n \text{ i.i.d.} \sim \mathcal{N}(\mu, \sigma_X^2).$$

In practice, assuming that  $\sigma_X$  is known is unrealistic. Thus, the test statistic  $Z$  cannot be determined since it is based on  $\sigma_X$ .

Instead, we can use the estimate

$$\hat{\sigma}_X^2 = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X}_n)^2$$

But this estimate leads to additional uncertainty thereby changing the distribution of the test statistic.

#### **$t$ -distribution**

The test statistic for the  $t$ -test is given by

$$T = \frac{\bar{X}_n - \mu_0}{\hat{\sigma}_X / \sqrt{n}} = \frac{\text{observed - expected}}{\text{estimated standard error}}$$

Its distribution under the null hypothesis  $H_0 : \mu = \mu_0$  is the so-called  **$t$ -distribution** with  $n - 1$  degrees of freedom, denoted by  $t_{n-1}$ .

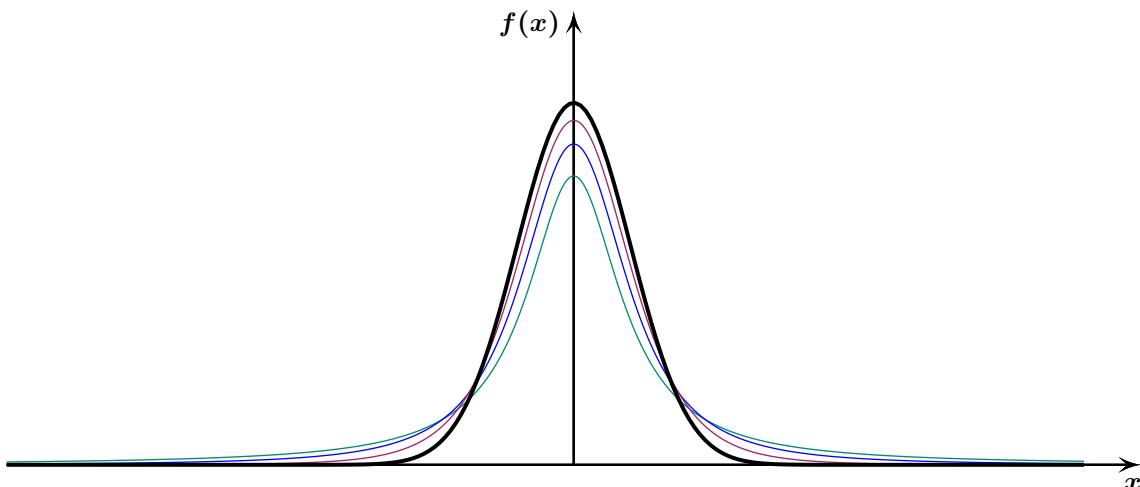
The  $t_n$ -distribution<sup>2</sup> is symmetric with respect to 0 and has heavier tails as compared to the standard normal distribution  $\mathcal{N}(0, 1)$ . This means that it is more prone to producing values that fall far from its mean (see Figure 2.6). For

$$T \sim t_n$$

the summary statistics are given by

$$\begin{aligned} E[T] &= 0 \\ \text{Var}[T] &= \frac{n}{n-2} \end{aligned}$$

For large  $n$ ,  $t_n$  becomes similar to  $\mathcal{N}(0, 1)$ . In particular, for  $n \rightarrow \infty$  the  $t_n$ -distribution approaches the standard normal distribution  $\mathcal{N}(0, 1)$ .



**Figure 2.6.:** Densities of the  $t$ -distribution with 1 (green), 2 (blue) and 5 (violet) degrees of freedom. The black curve is the density of the standard normal distribution.

### Example 2.3.2

The following data set consists of normally distributed data points  $x_1, \dots, x_{20}$

We assume that our measurements  $x_1, x_2, \dots, x_{20}$  are realizations of

$$X_i \sim \mathcal{N}(5, \sigma_X)$$

---

<sup>2</sup>An exact derivation of the  $t$ -distribution will be provided in Chapter ?? of the Annex.

## Chapter 2. Statistics for Measurement Data

5.9	3.4	6.6	6.3	4.2	2.0	6.0	4.8	4.2	2.1
8.7	4.4	5.1	2.7	8.5	5.8	4.9	5.3	5.5	7.9

where  $\sigma_X$  is unknown. Therefore, we must estimate  $\sigma_X$  on the basis of the data. The null hypothesis is  $\mu_0 = 5$ . What is the probability that the mean value of a data set with distribution

$$X_i \sim \mathcal{N}(5, \hat{\sigma}_X)$$

is smaller than the mean value of our data set, where  $\hat{\sigma}_X$  denotes the estimate of the standard deviation? In other words, we would like to determine the probability

$$P(\bar{X}_{20} \leq \bar{x}_{20})$$

where

$$\bar{X}_{20} \sim \mathcal{N}(5, \hat{\sigma}_X / \sqrt{20})$$

$\bar{x}_{20}$  denotes the empirical mean and  $\hat{\sigma}_X$  is the estimated empirical standard deviation of the data set.

To calculate the probability

$$P(\bar{X}_{20} \leq \bar{x}_{20})$$

we standardize the sample mean  $\bar{X}_{20}$  by subtracting the assumed mean value  $\mu_0 = 5$  from it and then divide by the estimated standard error  $\hat{\sigma}_X / \sqrt{n}$ . Thus,

$$\begin{aligned} P\left(\frac{\bar{X}_{20} - 5}{\hat{\sigma}_X / \sqrt{20}} \leq \frac{\bar{x}_{20} - 5}{\hat{\sigma}_X / \sqrt{20}}\right) &= P\left(T \leq \frac{5.2 - 5}{1.9 / \sqrt{20}}\right) \\ &= P(T \leq 0.51) \\ &= 0.69 \end{aligned}$$

where we have used the fact that  $T$  follows a  $t_{20-1}$ -distribution. [\(to R\)](#)

```
[1]: from scipy.stats import t
import numpy as np
from pandas import Series

# Normally distributed data points: (n=20)
x = Series([5.9, 3.4, 6.6, 6.3, 4.2, 2.0, 6.0,
            4.8, 4.2, 2.1, 8.7, 4.4, 5.1, 2.7,
            8.5, 5.8, 4.9, 5.3, 5.5, 7.9])

# Mean and standard deviation from sample
mean_x = x.mean()
std_x = x.std()
```

## Chapter 2. Statistics for Measurement Data

```
# t test statistic
t_x = (mean_x - 5) / (std_x / np.sqrt(x.size))

# P(T <= t), probability that T is smaller or equal t
p_T_t = t.cdf(x=t_x, df=x.size-1)

print(p_T_t)
```

0.6921780567888249



We obtain the rejection region for the  $t$ -test by cutting off an area under the  $t_{n-1}$ -distribution which corresponds to the probability  $\alpha$ . We cut off the area according to the direction of the alternative hypothesis, either on one side or one half from each side. To do so, we require the quantiles  $t_{n,\alpha}$ , which can be calculated by means of a statistical software package.

In summary, the  $t$ -test is performed according to the following six steps:

1. **Model:**  $X_i$  is a continuous measurement variable;

$$X_1, \dots, X_n \text{ i.i.d. } \mathcal{N}(\mu, \sigma_X^2), \quad \sigma_X \text{ is estimated by } \hat{\sigma}_X$$

2. **Null hypothesis:**

$$H_0 : \mu = \mu_0$$

**Alternative:**

$$H_A : \mu \neq \mu_0 \quad (\text{or } < \text{ or } >)$$

3. **Test statistic:**

$$T = \frac{\sqrt{n}(\bar{X}_n - \mu_0)}{\hat{\sigma}_X} = \frac{\text{observed - expected}}{\text{estimated standard error}}$$

**Null distribution assuming  $H_0$  is true:**

$$T \sim t_{n-1}$$

4. **Significance level:**

$$\alpha$$

5. **Rejection region for the test statistic:**

$$K = (-\infty, t_{n-1;\frac{\alpha}{2}}] \cup [t_{n-1;1-\frac{\alpha}{2}}, \infty) \quad \text{with } H_A: \mu \neq \mu_0,$$

$$K = (-\infty, t_{n-1;\alpha}] \quad \text{for } H_A: \mu < \mu_0,$$

$$K = [t_{n-1;1-\alpha}, \infty) \quad \text{for } H_A: \mu > \mu_0.$$

**6. Test decision:**

Check whether the observed value of the test statistic falls into the rejection region.

Since the absolute values of the quantiles of the  $t$ -distribution are larger than the quantiles of the normal distribution for the same  $\alpha$ , we obtain a somewhat smaller rejection region as compared to the  $z$ -test. For large  $n$ , the difference however is minimal (as  $t_{n-1} \approx \mathcal{N}(0, 1)$  for large  $n$ ).

**Example 2.3.3**

We take another look at example 2.3.1. However, we will now estimate the standard deviation  $\sigma_X$  on the basis of the data. We thus perform a  $t$ -test at the 5 % significance level:

1. **Model:**  $X_i$  is a continuous measurement variable;

$$X_1, \dots, X_n \text{ i.i.d. } \mathcal{N}(\mu, \sigma_X^2), \quad \sigma_X \text{ is estimated, } \widehat{\sigma}_X = 0.024$$

2. **Null hypothesis:**

$$H_0 : \mu = \mu_0 = 80.00$$

**Alternative:**

$$H_A : \mu \neq \mu_0$$

3. **Test statistic:**

$$T = \frac{\sqrt{n}(\bar{X}_n - \mu_0)}{\widehat{\sigma}_X}$$

**Null distribution assuming  $H_0$  is true:**

$$T \sim t_{n-1}$$

4. **Significance level:**

$$\alpha = 0.05$$

5. **Rejection region for the test statistic:**

$$K = (-\infty, t_{n-1; \frac{\alpha}{2}}] \cup [t_{n-1; 1-\frac{\alpha}{2}}, \infty) \quad \text{with } H_A: \mu \neq \mu_0,$$

We determine the value

$$t_{n-1; 1-\frac{\alpha}{2}} = t_{12; 0.975} = 2.179$$

by means of [Python](#), where  $\alpha = 0.05$  and  $n = 13$  ([to R](#))

```
[1]: from scipy.stats import t

# 5% quantile on t distribution with n = 13
q_5 = t.ppf(q=0.975, df=12)
print("5% quantile", q_5)
```

5% quantile 2.1788128296634177

The rejection region of the test statistic thus is given by

$$K = (-\infty, -2.179] \cup [2.179, \infty)$$

### 6. Test decision:

On the basis of  $n = 13$  measurements, we find

$$\bar{x} = 80.02 \quad \text{and} \quad \hat{\sigma}_X = 0.024$$

Hence, the realized value of the test statistic is

$$t = \frac{\sqrt{n}(\bar{x}_n - \mu_0)}{\hat{\sigma}_X} = \frac{\sqrt{13}(80.02 - 80.00)}{0.024} = 3.00$$

The observed value of the test statistic falls into the rejection region. Therefore, the null hypothesis is rejected at the 5 % level.



We can perform the  $t$ -test directly in [Python](#) by using the `scipy.stats` function `ttest_1samp()` ([to R](#))

```
[2]: import numpy as np
import scipy.stats as st
from pandas import Series

# Fusion heat data: n==13
x = Series([79.98, 80.04, 80.02, 80.04, 80.03,
            80.03, 80.04, 79.97, 80.05, 80.03,
            80.02, 80.00, 80.02])

# t-test directly
ttest = st.ttest_1samp(a=x, popmean=80)
print("t value", ttest.statistic, "\np value", ttest.pvalue)

int_5 = t.interval(alpha=0.95, df=12, loc=x.mean(),
                   scale=x.std()/np.sqrt(13))
print("95% interval confidence interval", int_5)
```

```
t value 3.1246428367325474
p value 0.008778778818391179
95% interval confidence interval (80.00628685027753, 80.03525161126093)
```

**Remarks:**

- i. The observed value of the test statistic is 3.12. Assuming the null hypothesis is true, then the test statistic follows a  $t$ -distribution with  $df = 12$  degrees of freedom.
- ii. The observed mean value of the data is 80.02. A 95 % confidence interval for the true mean is [80.006, 80.035].

◆

### 2.3.3. P-Value

A  $p$ -value takes on values between 0 and 1 and measures to which extent a *null hypothesis* is in agreement with given *data* (0: no agreement at all; 1: strong agreement).

To be more precise: we define a *p-value* as the probability that the test statistic will take on a value that is at least as *extreme* as the observed value of the statistic when the null hypothesis  $H_0$  is true. The qualifying term *extreme* refers to the alternative hypothesis : If  $H_A : \mu > \mu_0$ , then large sample means  $\bar{x}_n$  can be considered as extreme. If  $H_A : \mu < \mu_0$ , then small sample means  $\bar{x}_n$  are considered as extreme. In the case  $H_A : \mu \neq \mu_0$ , either small or large sample means refer to extreme test statistic values.

Thus, a  $p$ -value conveys much information about the weight of evidence against  $H_0$ , and so a decision maker can draw a conclusion at *any* specified level of significance. The smaller the  $p$ -value, the stronger is the evidence against the null hypothesis. If the  $p$ -value is smaller than a predefined limit, such as 5 %, 1 % or 0.1 %, then we have evidence to reject the null hypothesis.

#### ***p*-value**

The  $p$ -value is the probability that the test statistic will take on a value that is at least as extreme (with respect to the alternative hypothesis) as the observed value of the statistic when the null hypothesis  $H_0$  is true.

### Example 2.3.4

We take another look at example 2.3.1. We estimate the standard deviation  $\sigma_X$  on the basis of the data. We thus perform a  $t$ -test.

If  $X_i$  is a continuous measurement variable and if we assume

$$X_1, \dots, X_n \text{ i.i.d. } \mathcal{N}(\mu, \sigma_X^2)$$

then the estimate of  $\sigma_X$  is  $\widehat{\sigma_X} = 0.024$ .

## Chapter 2. Statistics for Measurement Data

The null hypothesis in our example can be formulated as

$$H_0 : \mu = \mu_0 = 80.00$$

Let us first consider the one-sided alternative hypothesis

$$H_A : \mu > \mu_0$$

The test statistic is given by

$$T = \frac{\sqrt{n}(\bar{X}_n - \mu_0)}{\hat{\sigma}_X}$$

and under the null distribution assuming  $H_0$  is true, we have

$$T \sim t_{n-1}$$

For the one-sided alternative hypothesis  $H_A : \mu > \mu_0$ , the  $p$ -value can be calculated as follows - the observed value of the test statistic is  $t = \frac{\sqrt{n}(\bar{x}_n - \mu_0)}{\hat{\sigma}_X} = 3.00$ :

$$p\text{-value} = P(T > t) = P(T > 3.00) = 0.00553$$

In [Python](#) we compute the one-sided  $p$ -value as follows ([to R](#))

```
[1]: from scipy.stats import t
df=12
# P-value one sided test:
p_one = 1-t.cdf(3.1246,df)

print("p-value one-sided: ", p_one)
```

We thus have evidence against the null hypothesis at the 5 % level.

Let us now consider the two-sided alternative hypothesis

$$H_A : \mu \neq \mu_0$$

For the two-sided alternative hypothesis  $H_A : \mu \neq \mu_0$ , the  $p$ -value can be calculated as follows (the observed value of the test statistic is  $t = \frac{\sqrt{n}|\bar{x}_n - \mu_0|}{\hat{\sigma}_X}$ ):

$$\begin{aligned} p\text{-value} &= P(|T| > |t|) \\ &= P(T < -|t|) + P(T > |t|) \\ &= 2 \cdot P(T > |t|) \end{aligned}$$

In [Python](#) we compute the two-sided  $p$ -value as follows: ([to R](#))

```
[2]: # P-value two sided test:
p_two = 2*(1-t.cdf(3.1246,df))

print("p-value two-sided: ", p_two)
```

p-value two-sided: 0.008779477358798138

For a two-sided alternative hypothesis the  $p$ -value is given by 0.008 779 and thus, it is statistically significant at the 5 % significance level.



Hence, we proceed analogously as in the statistical test setting : either we *retain* or *reject* the null hypothesis. In addition, we get a measure how strong the evidence is against the null hypothesis.

### ***p*-value and Statistical Test**

By means of the  $p$ -value we can directly conclude on the test decision : If the  $p$ -value is smaller than the significance level, then we reject the null hypothesis  $H_0$ , otherwise we retain it. In comparison with the “pure” test decision, the  $p$ -value contains more information since it indicates to which extent a null hypothesis may be rejected. For a given significance level  $\alpha$  (e.g.  $\alpha = 0.05$ ) we conclude on the basis of the  $p$ -value:

1. Reject  $H_0$  if  $p\text{-value} \leq \alpha$
2. Retain  $H_0$  if  $p\text{-value} > \alpha$

Many statistical software packages yield the test decision only “indirectly” by indicating the  $p$ -value.

In addition to this decision rule, the  $p$ -value quantifies *how significant* an alternative is, that is how strong the evidence is for the rejection of  $H_0$ . Often symbols or other formulae are indicated instead of the  $p$ -value.

$p\text{-value} \approx 0.05$  : weakly significant, “.”  
 $p\text{-value} \approx 0.01$  : significant, “\*”  
 $p\text{-value} \approx 0.001$  : strongly significant, “\*\*”  
 $p\text{-value} \leq 10^{-4}$  : very strongly significant, “\*\*\*”

**Remarks:**

- i. **Attention:** The  $p$ -value is not the probability that the null hypothesis is true. We are not in the position to make any statement with respect to the probability that the null hypothesis is correct since our parameter values are fixed and not random. When we determine the  $p$ -value then we assume that  $H_0$  is correct and we “simply” determine to which extent the observed test statistic is extreme.
- ii. Strictly speaking, the  $p$ -value is *not* a probability. For, if we conceive probability as the relative frequency an event is realized, it does not make sense to use this terminology in the context of a  $p$ -value. The  $p$ -value does not make any statement with regard to how often the observed test statistic or a more extreme one may occur, it tells us simply to which extent such an event is compatible with the null hypothesis.
- iii. Strictly speaking, the  $p$ -value is just a transformed random variable. We thus may define the  $p$ -value as follows:  
*The p-value is the smallest level of significance that would lead to rejection of the null hypothesis  $H_0$  with the given data.*  
 This definition avoids the interpretation of the  $p$ -value as a probability. Nevertheless, in the context of  $p$ -values we will refer to a probability in the rest of these lecture notes. ◆

### 2.3.4. Confidence Intervals for $\mu$

The confidence interval for measurement data consists of the values  $\mu$ , for which the corresponding statistical test does not reject the null hypothesis. Assuming these parameter values for the random model, the data are relatively probable or plausible.

In turn, we assume that our measurements are realizations of

$$X_1, \dots, X_n \text{ i.i.d.} \sim \mathcal{N}(\mu, \sigma_X^2).$$

For a two-sided  $t$ -test, the rejection region is

$$K = (-\infty, -t_{n-1;1-\frac{\alpha}{2}}] \cup [t_{n-1;1-\frac{\alpha}{2}}, \infty)$$

The  $t$ -test does not reject  $H_0$  if the value of the test statistic does not fall into the rejection region of the test statistic. If  $H_0$  is not rejected, the following relations must apply:

$$-t_{n-1;1-\frac{\alpha}{2}} \leq \frac{\sqrt{n}(\bar{x}_n - \mu_0)}{\hat{\sigma}_X} \quad \text{and} \quad t_{n-1;1-\frac{\alpha}{2}} \geq \frac{\sqrt{n}(\bar{x}_n - \mu_0)}{\hat{\sigma}_X}$$

In order to determine a two-sided confidence interval of  $\mu$ , we have to determine all values of  $\mu_0$  that satisfy the equations given above. Hence, we simply solve the two equations for  $\mu_0$  and obtain

$$\mu_0 \leq \bar{x}_n + \frac{\hat{\sigma}_X \cdot t_{n-1;1-\frac{\alpha}{2}}}{\sqrt{n}} \quad \text{and} \quad \mu_0 \geq \bar{x}_n - \frac{\hat{\sigma}_X \cdot t_{n-1;1-\frac{\alpha}{2}}}{\sqrt{n}}$$

### Two-sided and One-sided Confidence Interval

The *two-sided confidence interval* (the corresponding statistical test has the two-sided alternative  $H_A: \mu \neq \mu_0$ ) at the level  $1 - \alpha$  is given by

$$[\bar{x}_n - t_{n-1,1-\alpha/2} \frac{\hat{\sigma}_X}{\sqrt{n}}, \bar{x}_n + t_{n-1,1-\alpha/2} \frac{\hat{\sigma}_X}{\sqrt{n}}]$$

Analogously, *one-sided confidence intervals* can be derived. They include all parameter values for which a one-sided test would not reject the null hypothesis. The one-sided confidence intervals at the level  $1 - \alpha$  are given by

$$\begin{aligned} \text{If } H_A: \mu < \mu_0: & (-\infty; \bar{x}_n + t_{n-1,1-\alpha} \cdot \frac{\hat{\sigma}_X}{\sqrt{n}}] \\ \text{If } H_A: \mu > \mu_0: & [\bar{x}_n - t_{n-1,1-\alpha} \cdot \frac{\hat{\sigma}_X}{\sqrt{n}}; \infty) \end{aligned}$$

### Example 2.3.5 Measuring the fusion heat

We have recorded 13 measurements, thus we have

$$n - 1 = 13 - 1 = 12$$

degrees of freedom. The 97.5 % quantile of the  $t_{12}$ -distribution is given by

$$t_{12,0.975} = 2.18$$

The empirical mean of the fusion heat measurements is

$$\bar{x}_{13} = 80.02$$

and the empirical standard deviation is

$$\hat{\sigma}_X = 0.024$$

The two-sided confidence interval for the true mean value of the fusion heat measurements thus is given by

$$I = 80.02 \pm 2.18 \cdot 0.024 / \sqrt{13} = [80.01, 80.03]$$

In particular, 80.00 does not fall into the interval  $I$ . The value  $\mu = 80.00$  is consequently not compatible with the data. We already have come to the same conclusion by means of the  $t$ -test we have performed in [Python](#). ◀

## 2.4. Statistical Significance and Relevance in Statistical Tests

The term *statistical significance* is often misused in order to simultaneously underline the corresponding *relevance*. But the two terms do not necessarily need to be associated. If sufficient observations are accumulated, it is possible to reject any null hypothesis, as in practice a null hypothesis never is perfectly true.

Does this mean that the concepts introduced in the previous chapters are all useless in practice? The answer is of course *no*, but they must be correctly applied.

For this purpose, we must combine the best features of the two areas: the corresponding expertise and the statistical output. On the basis of expertise, we first should define what a relevant effect or difference is - statistics will not provide any help for that. Having done this, we can bring statistics into play.

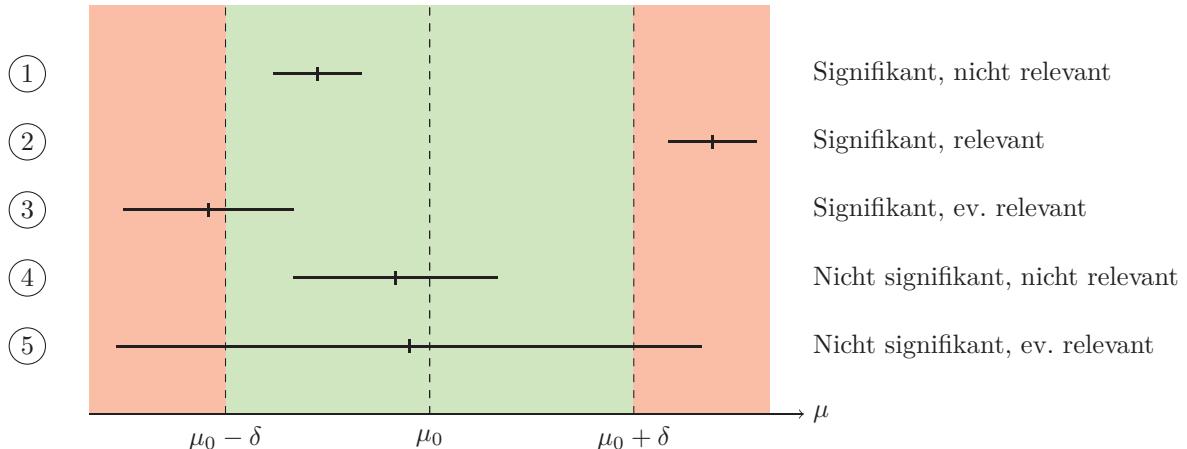
If we consider the production of screws as an example: We assume that deviations up to 0.5 mm of the target length 100 mm are not worrisome, and are consequently irrelevant. Hence, we have an “irrelevant region”, ranging from 99.5 mm to 100.5 mm. We call the region outside of this interval the *relevance region*.

The idea is to check to which extent the confidence interval for  $\mu$  overlaps with the relevance region. Assuming we have determined in a random sample the confidence interval for the true  $\mu$  to be [100.03, 100.31], which completely falls into the irrelevant region. Therefore, we would conclude that the deviation is statistically significant, but *not* relevant. A number of possible situations are displayed and interpreted in Figure 2.7.

## Conceptional Learning Objectives

You should be able...

- to explain the concept of a QQ-Plot and of a Normal-Plot.
- to decide whether a data set is distributed according to a normal distribution.
- to explain the concept of maximum likelihood parameter estimation and the method of moments.



**Figure 2.7.:** Interplay of statistical significance and relevance.

The confidence intervals for  $\mu$  are represented by dashes. The “irrelevant region” spans from  $\mu_0 - \delta$  to  $\mu_0 + \delta$  (green).  $\delta$  was specified on the basis of expert knowledge.

- to apply maximum likelihood parameter estimation to an arbitrary distribution family.
- to carry out a  $z$ - and  $t$ -test for a measurement series on paper or by means of a computer software package (one-sided and two-sided).
- to explain the concept of a  $p$ -value and to compute it.
- to explain the concept of a confidence interval and to compute it.
- to consider the  $p$ -value or confidence interval for your test decision.

## Computational Objectives

You should be able...

- to generate a QQ-Plot and Normal-Plot for a measurement series by means of `norm.ppf()` and `pyplot.plot()`.
- to carry out a  $t$ -test by means of `scipy.stats.t.interval()` and `scipy.stats.t.ttest_1samp()`
- to print the different attributes of `scipy.stats.t.ttest_1samp()`, in particular `.statistic` and `.pvalue`.

# Chapter 3.

## Joint Distributions

Descriptive statistics often deals with several quantities that are measured at the same time, e.g. the water level of a river at two positions  $A$  and  $B$  or the traffic volume at two different positions of the same street. Often, we cannot assume independence between such quantities. If the water level at position  $A$  is high, then the water position at position  $B$  is likely going to be high as well (and vice versa). In order to model such situations, we will use so-called joint distributions.

### 3.1. Joint, Marginal and Conditional Distributions

#### 3.1.1. Discrete Random Variables

We will illustrate the terms *marginal distributions* and *conditional distributions* with an example.

##### Example 3.1.1 Weather stations

Two weather stations measure the cloudiness  $X$  and  $Y$ , respectively, on a scale from 1 to 4. The probabilities of observing all combinations are depicted in Table 3.1.

$X \setminus Y$	1	2	3	4	$\Sigma$
1	0.080	0.015	0.003	0.002	0.100
2	0.050	0.350	0.050	0.050	0.500
3	0.030	0.060	0.180	0.030	0.300
4	0.001	0.002	0.007	0.090	0.100
$\Sigma$	0.161	0.427	0.240	0.172	1

Table 3.1.: Measured Distribution of cloudiness at two **weather stations** ( $X, Y$ ).

### Chapter 3. Joint Distributions

Based on this table, we can read out and compute a number of probabilities.

1. The value 0.030 in the third row and fourth column corresponds to the probability, that weather station X measures a cloudiness of 3 and weather station Y measures a cloudiness of 4.

$$P(X = 3, Y = 4) = 0.030$$

or

$$P(X = 3 \cap Y = 4) = 0.030$$

In the following, we will use the first notation.

2. We can compute as well the probability that weather station X measures a cloudiness of 3. The corresponding value

$$P(X = 3) = 0.300$$

can be found in the third row on the right border of the table. This value results from the following calculation (law of total probability):

$$\begin{aligned} P(X = 3) &= P(X = 3, Y = 1) + P(X = 3, Y = 2) \\ &\quad + P(X = 3, Y = 3) + P(X = 3, Y = 4) \\ &= 0.030 + 0.060 + 0.180 + 0.030 \\ &= 0.300 \end{aligned}$$

In summary, the values in the third row sum up to 0.300.

Thus,

$$P(Y = 2) = 0.427$$

is the probability that weather station Y measures a cloudiness of 2.

These probabilities on the right border of table 3.1 are called *marginal distribution of X* and accordingly, the probabilities in the lower row are the *marginal distribution of Y*.

3. We can compute as well conditional probabilities. For example,

$$P(Y = 2 | X = 4)$$

corresponds to the probability that weather station Y measures a cloudiness of 2, given weather station X measured a value of 4. These probabilities result from the following calculation

$$P(Y = 2 | X = 4) = \frac{P(Y = 2, X = 4)}{P(X = 4)} = \frac{0.002}{0.1} = 0.02$$

### Chapter 3. Joint Distributions

4. We can as well compute the probability that both weather stations measure the same grade of cloudiness:

$$\begin{aligned}
 P(X = Y) &= \sum_{j=1}^4 P(X = j, Y = j) \\
 &= P(X = 1, Y = 1) + P(X = 2, Y = 2) \\
 &\quad + P(X = 3, Y = 3) + P(X = 4, Y = 4) \\
 &= 0.080 + 0.350 + 0.180 + 0.090 \\
 &= 0.700
 \end{aligned}$$

5. We now have a look at the *independence* between random variables. If they are independent, then the following relation must hold

$$P(X = i, Y = j) = P(X = i) \cdot P(Y = j)$$

for all  $i$  and  $j$ .

On the other hand, it follows from the marginal distributions of  $X$  and  $Y$

$$P(X = 1) \cdot P(Y = 2) = 0.100 \cdot 0.427 = 0.043$$

and this is not equal to

$$P(X = 1, Y = 2) = 0.15$$

Hence, the random variables  $X$  und  $Y$  are not independent.



### Joint Probability Distributions

The *joint distribution* of two discrete random variables  $X$  and  $Y$  having their values in  $\mathbb{W}_X$  and in  $\mathbb{W}_Y$ , respectively, is defined by the *joint probability distribution* of  $X$  and  $Y$ , i.e., by the following probabilities

$$P(X = x, Y = y), \quad x \in \mathbb{W}_X, y \in \mathbb{W}_Y$$

where  $\mathbb{W}_X$  and  $\mathbb{W}_Y$  denote the ranges for the two random variables  $X$  und  $Y$ , respectively. In the preceding example, we had

$$\mathbb{W}_X = \{1, 2, 3, 4\} \quad \text{und} \quad \mathbb{W}_Y = \{1, 2, 3, 4\}$$

These ranges are identical in this example; in general, this is not the case.

In this “joint” context, the “single” distributions  $P(X = x)$  of  $X$  and  $P(Y = y)$  of  $Y$  are called *marginal distributions* of the joint random variable  $(X, Y)$ .

### Chapter 3. Joint Distributions

Marginal distributions can be calculated based on their joint distribution:

$$P(X = x) = \sum_{y \in \mathbb{W}_Y} P(X = x, Y = y), \quad x \in \mathbb{W}_X$$

and accordingly for  $Y$ . This simply is another application of the law of total probability. In the preceding example, we were summing up all values along one row.

If however we intend to compute the joint distribution of  $(X, Y)$  starting from the marginal distributions of  $X$  and  $Y$ , this is *only* possible for independent  $X$  and  $Y$ . In this case, the following relation holds

$$P(X = x, Y = y) = P(X = x) \cdot P(Y = y), \quad x \in \mathbb{W}_X, y \in \mathbb{W}_Y$$

In this case, the joint distribution is completely defined by the marginal distributions. We simply need to multiply the marginal distributions to obtain the corresponding joint distribution.

Furthermore, we define the *conditional probability* of  $X$  given  $Y = y$  as

$$P(X = x | Y = y) = \frac{P(X = x, Y = y)}{P(Y = y)}$$

The marginal distributions then can be expressed as follows

$$P(X = x) = \sum_{y \in \mathbb{W}_Y} P(X = x | Y = y) P(Y = y), \quad x \in \mathbb{W}_X$$

This relation can be applied whenever we want to compute the distribution of  $X$ , but when only its conditional distribution given  $Y$  and the distribution of  $Y$  are known.

The *conditional expected value* of  $Y$  given  $X = x$  is defined as

$$\mathbb{E}[Y | X = x] = \sum_{y \in \mathbb{W}_Y} y \cdot P(Y = y | X = x)$$

#### 3.1.2. Continuous Random Variables

For two or more continuous random variables, we need to extend the concept of density to higher dimensions.

### Joint density function

The joint density function  $f_{X,Y}(\cdot, \cdot)$  of two continuous random variables  $X$  and  $Y$  is given by (using “engineering notation”)

$$P(x \leq X \leq x + dx, y \leq Y \leq y + dy) = f_{X,Y}(x, y) dx dy$$

A joint density function can be interpreted in analogous way as in the case of a single continuous distribution. We will omit the discussion how a joint density function may be defined as derivative of an appropriate cumulative distribution function since this is not very instructive in higher dimensions.

The probability that the random variable  $(X, Y)$  lies in a two-dimensional region  $A$ , i.e.,  $A \subset \mathbb{R}^2$ , can be computed as in the one-dimensional case: we integrate the density over the set on which the density is defined:

$$P((X, Y) \in A) = \iint_A f_{X,Y}(x, y) dx dy$$

Geometrically, the volume included by the surface  $A$  and the joint density function  $f_{X,Y}(x, y)$  now corresponds to the probability, that the joint random variable  $(X, Y)$  falls into the region  $A$ . In general, the (bivariate) joint density function needs to satisfy

$$\iint_{\mathbb{R}^2} f_{X,Y}(x, y) dx dy = 1$$

Furthermore,  $X$  and  $Y$  are *independent* if and only if

$$f_{X,Y}(x, y) = f_X(x) \cdot f_Y(y), \quad x, y \in \mathbb{R}$$

In this case, the concept of an one-dimensional density is sufficient: the joint density can be computed by simple *multiplication* of the one-dimensional densities.

### Example 3.1.2 Life expectancy of two machines

We consider two machines with exponentially distributed life expectancy  $X \sim \text{Exp}(\lambda_1)$  and  $Y \sim \text{Exp}(\lambda_2)$ , where  $X$  and  $Y$  are independent random variables. What is the probability, that machine 1 runs longer than machine 2?

The density functions for the life expectancy of the two machines are given by

$$f_X(x) = \lambda_1 e^{-\lambda_1 x} \quad \text{and} \quad f_Y(y) = \lambda_2 e^{-\lambda_2 y}$$

### Chapter 3. Joint Distributions

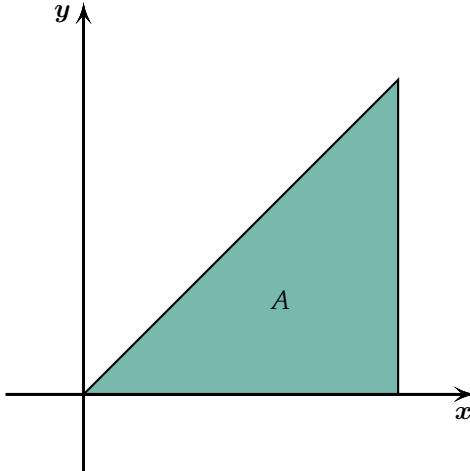
Due to the independence of the two random variables, the joint density can be calculated as follows

$$f_{X,Y}(x,y) = \lambda_1 e^{-\lambda_1 x} \lambda_2 e^{-\lambda_2 y}$$

for  $x, y \geq 0$  (otherwise the density is 0). The set which is relevant for the raised question is given by

$$A = \{(x,y) | 0 \leq y \leq x\}$$

It contains all points below the bisecting line, see Figure 3.1.

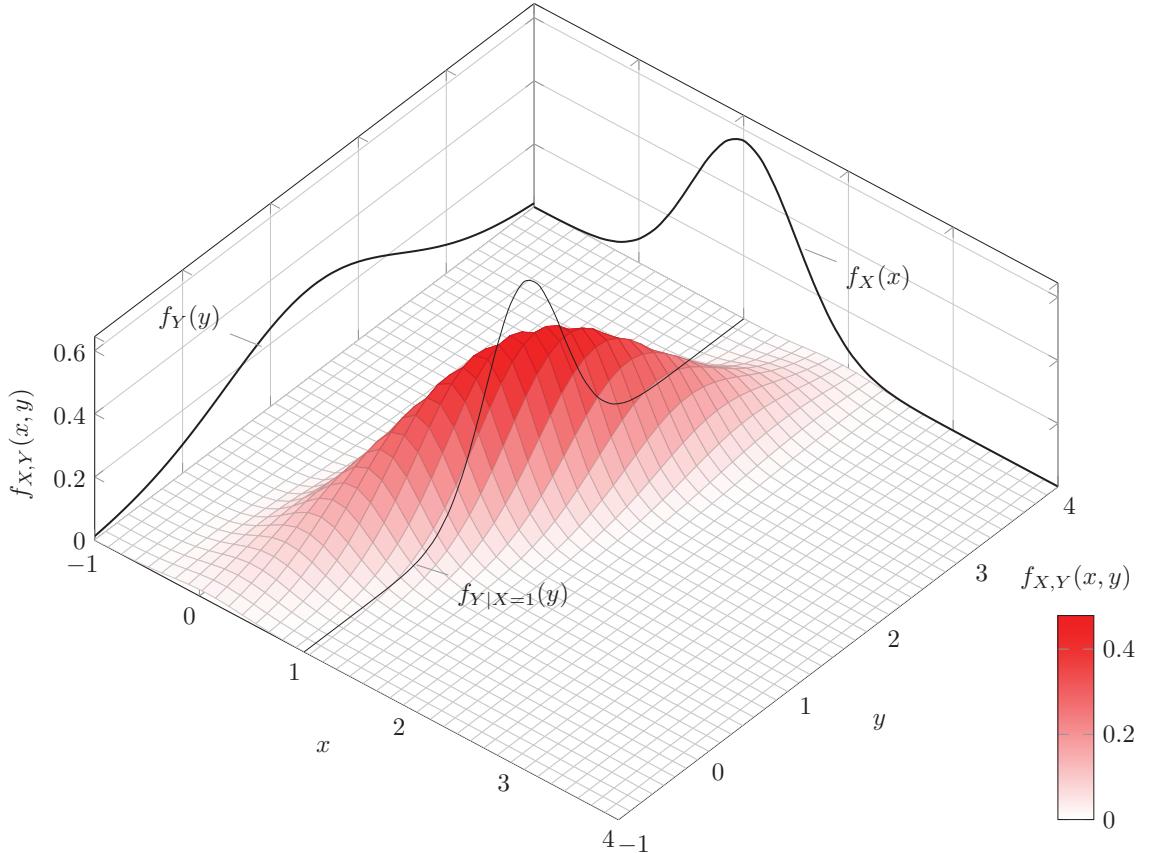


**Figure 3.1.:** Integration domain for the example [Life expectancy of two machines](#).

Thus, we have

$$\begin{aligned} P(Y < X) &= \int_0^\infty \left( \int_0^x \lambda_1 e^{-\lambda_1 x} \lambda_2 e^{-\lambda_2 y} dy \right) dx \\ &= \int_0^\infty \lambda_1 e^{-\lambda_1 x} (1 - e^{-\lambda_2 x}) dx \\ &= \int_0^\infty \lambda_1 e^{-\lambda_1 x} dx - \int_0^\infty \lambda_1 e^{-(\lambda_1 + \lambda_2)x} dx \\ &= 1 - \frac{\lambda_1}{\lambda_1 + \lambda_2} \\ &= \frac{\lambda_2}{\lambda_1 + \lambda_2} \end{aligned}$$

The first integral in the third last equation is equal to 1, since we integrated the density of the  $X \sim \text{Exp}(\lambda_1)$ -distribution over the entire definition domain.  $\blacktriangleleft$



**Figure 3.2.:** Illustration of a two-dimensional density with its marginal densities  $f_X(x)$  and  $f_Y(y)$  and its conditional density given  $X = 1$ .

### Marginal Density and Conditional Density

Analogous to the discrete case, we define the *marginal distribution* as the distribution of a single component of the joint random variable. We thus treat the random variables  $X$  and  $Y$  as if we were simply considering one componentent  $X$  or  $Y$ .

Using the joint density, we can calculate the *marginal densities* of  $X$  and of  $Y$  by integrating over the other variable

$$f_X(x) = \int_{-\infty}^{\infty} f_{X,Y}(x,y) \, dy, \quad f_Y(y) = \int_{-\infty}^{\infty} f_{X,Y}(x,y) \, dx$$

This exactly corresponds to the discrete case where we simply summed over all values of the other variable by means of the law of total probability instead of integrating. An illustration is depicted in Figure 3.2.

### Chapter 3. Joint Distributions

We express the *conditional probability* of  $Y$  given  $X = x$  by using the conditional density which is defined as

$$f_{Y|X=x}(y) = f_Y(y|X=x) = \frac{f_{X,Y}(x,y)}{f_X(x)}$$

This represents a cross-section, respectively a longitudinal section of the joint density, see Figure 3.2. We keep  $x$  constant and vary only  $y$ . The denominator will make sure that the density integrates up to 1. In the case of a discrete random variable, we simply fixed the corresponding row or column in the table such that the sum turned out to be 1.

In particular, from the definition of a conditional density it follows that  $X$  and  $Y$  are independent if and only if the relations

$$f_{Y|X=x}(y) = f_Y(y) \quad \text{resp.} \quad f_{X|Y=y}(x) = f_X(x)$$

apply for all  $x$  and  $y$ . This means that in the case of independence, knowing  $X$  does not have any influence on the distribution of  $Y$  (and vice versa). Furthermore, the joint density for any random variables  $X$  and  $Y$  can in general be expressed as follows

$$f_{X,Y}(x,y) = f_{Y|X=x}(y)f_X(x) = f_{X|Y=y}(x)f_Y(y)$$

This is particularly useful if a model is defined “stepwise”.

The *conditional expected value* of a continuous random variable  $Y$  given  $X = x$  is defined as

$$\mathbb{E}[Y|X=x] = \int_{-\infty}^{\infty} y \cdot f_{Y|X=x}(y) \, dy$$

The calculation proceeds analogously to the usual expected value, we simply use the corresponding conditional density.

From the preceding definitions follows that all probability theoretical aspects of two continuous random variables  $X$  and  $Y$  are completely determined by their joint density  $f_{X,Y}(\cdot, \cdot)$ .

## 3.2. Expected Value of Several Random Variables

The expected value of a transformed random variable  $Z = g(X, Y)$  with  $g : \mathbb{R}^2 \rightarrow \mathbb{R}$  can be computed as follows

$$\mathbb{E}[g(x,y)] = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g(x,y) f_{X,Y}(x,y) \, dx \, dy$$

### Chapter 3. Joint Distributions

In the case of a discrete random variable, the corresponding formula is

$$E[g(X, Y)] = \sum_{x \in W_X} \sum_{y \in W_Y} g(x, y) P(X = x, Y = y)$$

In particular, for a linear combination of random variables we have the following relation

$$E[a + bX + cY] = a + b \cdot E[X] + c \cdot E[Y], \quad a, b, c \in R$$

This always holds independently of whether the random variables are independent or not. If we consider more than two random variables, the calculation of the expected value proceeds analogously, i.e.,

$$E \left[ a_0 + \sum_{i=1}^n a_i X_i \right] = a_0 + \sum_{i=1}^n a_i E[X_i]$$

where  $a_0, a_1, \dots, a_n \in \mathbb{R}$ .

## 3.3. Covariance and Correlation

Since the joint distribution of dependent random variables is in general complicated, we describe the dependency between random variables by means of simplifying *summary statistics*. To this aim, we will introduce the concepts of *covariance* and *correlation* between two random variables  $X$  and  $Y$ .

### Covariance

The covariance of two random variables  $X$  and  $Y$  is defined as

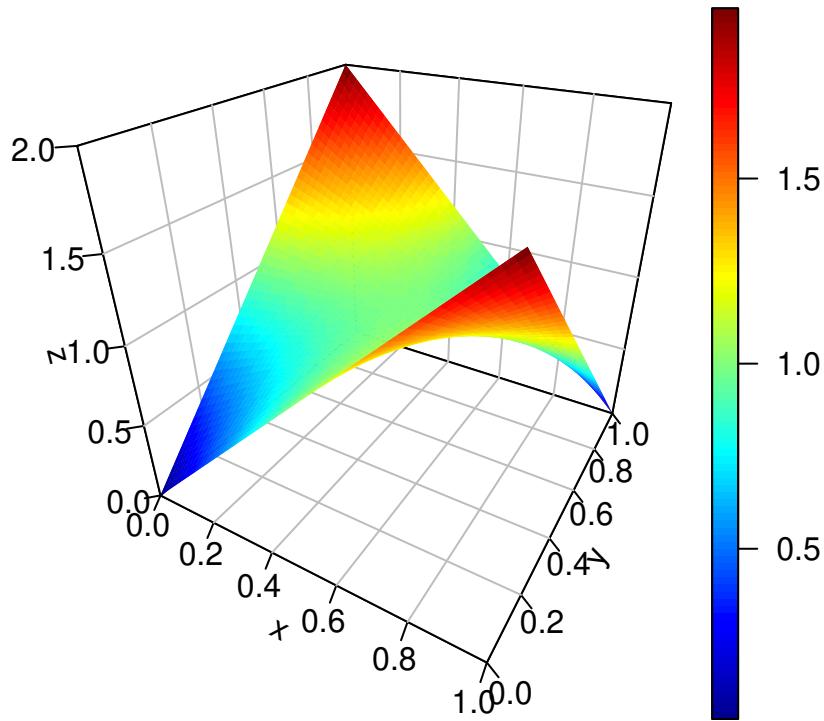
$$\text{Cov}[X, Y] = E[(X - \mu_X)(Y - \mu_Y)]$$

The covariance is the average value of the product of the deviation of  $X$  from its mean and the deviation of  $Y$  from its mean. If the random variables are positively associated - that is, when  $X$  is larger than its mean,  $Y$  tends to be larger than its mean as well - the covariance will be *positive*. If the association is negative - that is, when  $X$  is larger than its mean,  $Y$  tends to be smaller than its mean - the covariance is *negative*.

### Example 3.3.1

We now consider a bivariate probability distribution

$$f(x, y) = (2x + 2y - 4xy), \quad 0 \leq x \leq 1, 0 \leq y \leq 1$$



**Figure 3.3.:** Bivariate density  $f(x,y) = (2x + 2y - 4xy)$  defined on  $0 \leq x \leq 1, 0 \leq y \leq 1$ .

where the marginal densities of  $X$  and  $Y$  are uniformly distributed random variables on the interval  $[0, 1]$ . Thus, we have  $E[X] = E[Y] = \frac{1}{2}$  and  $\text{Var}[X] = \text{Var}[Y] = \frac{1}{12}$ .

The joint density distribution of  $(X, Y)$  is depicted in Figure 3.3.

The *marginal densities* of  $X$  and  $Y$  are uniformly distributed random variables on the interval  $[0, 1]$ :

$$\begin{aligned} f_X(x) &= \int_0^1 f(x,y) \, dy = \int_0^1 (2x + 2y - 4xy) \, dy \\ &= 1 \end{aligned}$$

and

$$f_Y(y) = \int_0^1 f(x, y) dx = \int_0^1 (2x + 2y - 4xy) dx \\ = 1$$

Thus, we obtain for the expected values

$$E[X] = E[Y] = \frac{1}{2}$$

and for the variances

$$\text{Var}[X] = \text{Var}[Y] = \frac{1}{12}$$

The covariance thus is given by

$$\begin{aligned} \text{Cov}[X, Y] &= \int_0^1 \int_0^1 (x - \mu_X)(y - \mu_Y)f(x, y) dx dy \\ &= \int_0^1 \int_0^1 (x - \frac{1}{2})(y - \frac{1}{2})(2x + 2y - 4xy) dx dy \\ &= -\frac{1}{36} \end{aligned}$$

The covariance is negative, indicating a negative relationship between  $X$  and  $Y$ . In fact, from Figure 3.3, we see that if  $X$  is smaller than its mean  $\frac{1}{2}$ , then  $Y$  tends to be larger than its mean  $\frac{1}{2}$ . ◀

### Correlation

The *correlation* of  $X$  and  $Y$ , denoted by  $\rho$  is

$$\text{Cor}[X, Y] = \rho_{XY} = \frac{\text{Cov}[X, Y]}{\sigma_X \sigma_Y}$$

The correlation is simply a standardized version of the covariance. Contrary to the covariance, the correlation thus is a dimensionless quantity.

**Example 3.3.2**

For the bivariate probability distribution

$$f(x, y) = (2x + 2y - 4xy), \quad 0 \leq x \leq 1, 0 \leq y \leq 1$$

we previously found for the expected value

$$\mathbb{E}[X] = \mathbb{E}[Y] = \frac{1}{2}$$

for the variance

$$\text{Var}[X] = \text{Var}[Y] = \frac{1}{12}$$

and for the covariance

$$\text{Cov}[X, Y] = -\frac{1}{36}$$

The *correlation* thus is given by

$$\rho = \frac{-1/36}{\sqrt{1/12}\sqrt{1/12}} = -\frac{1}{36} \cdot \frac{1}{1/12} = -\frac{1}{3}$$

◀

$\rho$  has the following property

$$-1 \leq \text{Cor}[X, Y] \leq 1$$

The correlation is a measure for the strength and direction of the *linear dependency* between  $X$  and  $Y$ . We have

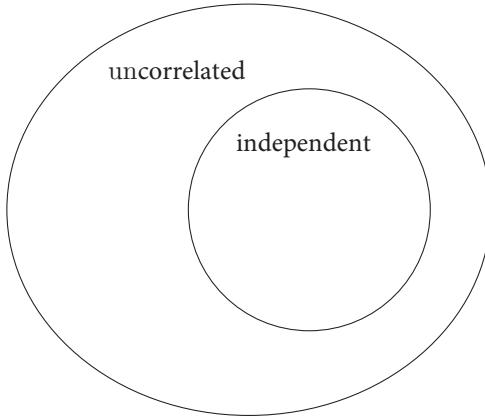
$$\begin{aligned} \text{Cor}[X, Y] &= +1 \quad \text{if and only if } Y = a + bX \quad \text{for } a \in \mathbb{R} \text{ and } b > 0 \\ \text{Cor}[X, Y] &= -1 \quad \text{if and only if } Y = a + bX \quad \text{for } a \in \mathbb{R} \text{ and } b < 0 \end{aligned}$$

If  $|\text{Cor}[X, Y]| = 1$ , then we have a perfectly linear relationship between  $X$  und  $Y$ .

If  $\text{Cor}[X, Y] = 0$ , we say, that  $X$  und  $Y$  are uncorrelated. In this case, there is no linear relationship between  $X$  and  $Y$  (however, there may be a non-linear relationship).

Furthermore, we have

$$X \text{ and } Y \text{ independent} \Rightarrow \text{Cor}[X, Y] = 0 \quad (\text{and thus } \text{Cov}[X, Y] = 0)$$



**Figure 3.4.:** Relation between independence and uncorrelatedness illustrated as a Venn diagram.

The converse however is not true in general, i.e., from uncorrelatedness does *not* follow independence, see Figure 3.4.

An exceptional case, where the converse holds as well, will be discussed in Chapter 3.4.

Figure 3.5 displays the contour plot (contour lines) of a two-dimensional density for different values of  $\rho$ . We observe that if the absolute value of  $\rho$  gets large, the density continuously gets concentrated “around a line”. That means, points are very likely located next to this line.

In particular, we can use the covariance to calculate the variance of sums of random variables.

The following relation follows immediately from the definition of the covariance

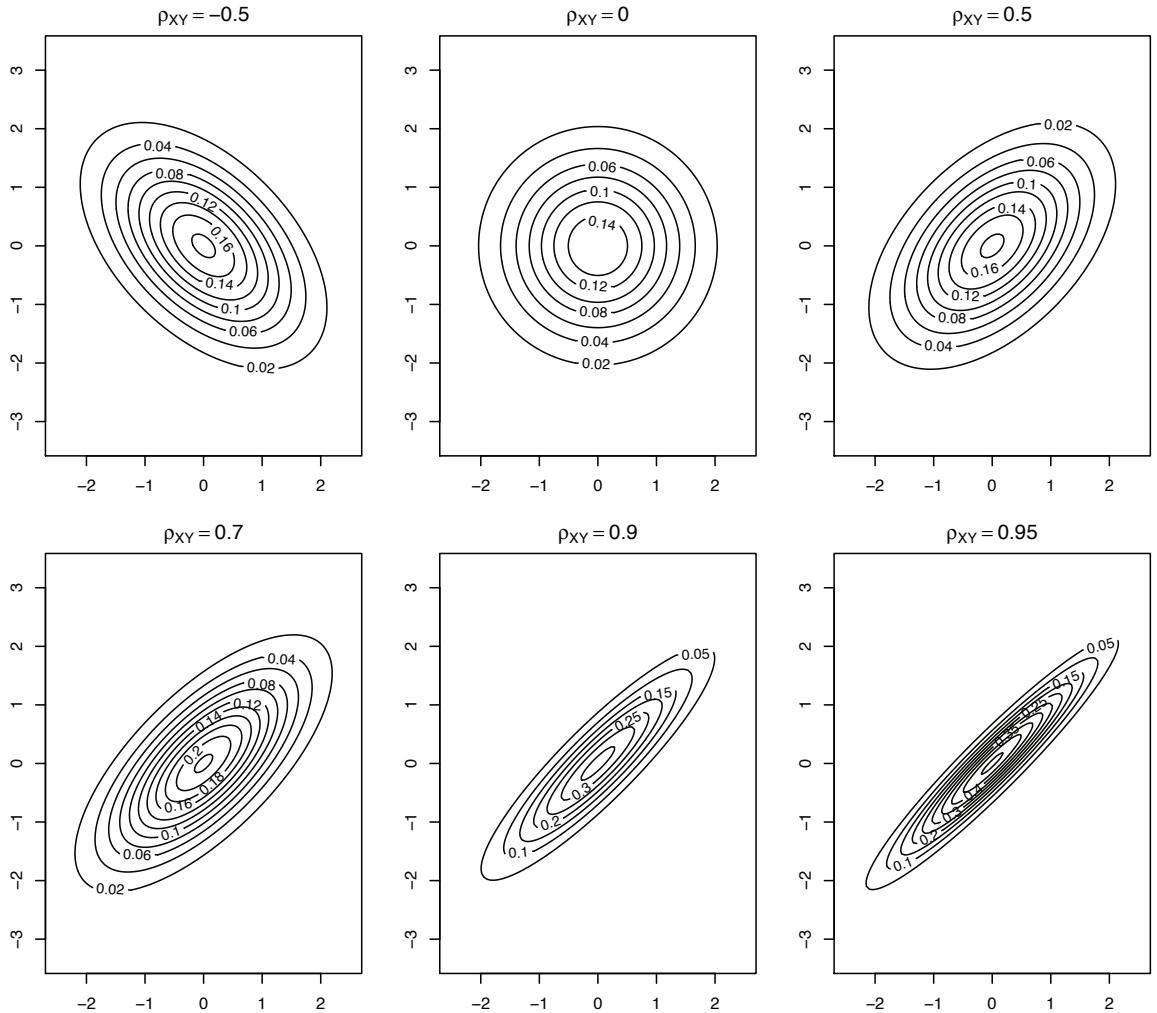
$$\text{Var}[X] = \text{Cov}[X, X]$$

as well the following formula

$$\begin{aligned} \text{Cov}[X, Y] &= E[XY - X\mu_Y - Y\mu_X + \mu_X\mu_Y] \\ &= E[XY] - E[X]\mu_Y - E[Y]\mu_X + \mu_X\mu_Y \\ &= E[XY] - E[X] E[Y] \end{aligned}$$

which may be relevant for the practical calculation of the covariance. In particular, the following relation holds for independent and in general for uncorrelated

## Chapter 3. Joint Distributions



**Figure 3.5.:** Contour Lines of a two-dimensional density for different values of  $\rho$ . The densities displayed in the Figure are bivariate normal distributions, see Chapter 3.4.

random variables

$$E[XY] = E[X] E[Y], \quad (X, Y \text{ independent})$$

This relation follows immediately from the above formula.

### Example 3.3.3

We consider again the bivariate probability distribution

$$f(x, y) = (2x + 2y - 4xy), \quad 0 \leq x \leq 1, 0 \leq y \leq 1$$

where the marginal densities of  $X$  and  $Y$  are uniformly distributed random variables on the interval  $[0, 1]$ . Thus, we have  $E[X] = E[Y] = \frac{1}{2}$  and  $\text{Var}[X] = \text{Var}[Y] = \frac{1}{12}$ .

We obtain

$$\begin{aligned} E[XY] &= \int \int xy f(x, y) dx dy \\ &= \int_0^1 \int_0^1 xy(2x + 2y - 4xy) dx dy \\ &= \frac{2}{9} \end{aligned}$$

From this, it follows

$$\begin{aligned} \text{Cov}[X, Y] &= E[XY] - E[X] E[Y] \\ &= \frac{2}{9} - \left(\frac{1}{2}\right) \left(\frac{1}{2}\right) \\ &= -\frac{1}{36} \end{aligned}$$

We have obtained the identical result previously by computing the covariance directly:

$$\text{Cov}[X, Y] = \int \int (x - \mu_X)(y - \mu_Y) f(x, y) dx dy$$



Further, the covariance is *bilinear*, i.e., we have

$$\text{Cov} \left[ \sum_{i=1}^n a_i X_i, \sum_{j=1}^m b_j X_j \right] = \sum_{i=1}^n \sum_{j=1}^m a_i b_j \text{Cov}[X_i, Y_j] \quad a_i, b_j \in \mathbb{R}$$

and *symmetrical*, i.e.

$$\text{Cov}[X, Y] = \text{Cov}[Y, X]$$

Further calculation rules involving the covariance are

$$\text{Cov}[a + bX, c + dY] = bd \text{Cov}[X, Y]$$

and

$$\text{Cor}[a + bX, c + dY] = \text{sign}(b) \text{sign}(d) \text{Cor}[X, Y]$$

We find for the sum of variances

$$\text{Var} \left[ \sum_{i=1}^n X_i \right] = \text{Cov} \left[ \sum_{i=1}^n X_i, \sum_{i=1}^n X_i \right] = \sum_{i=1}^n \text{Var}[X_i] + 2 \sum_{i < j} \text{Cov}[X_i, X_j]$$

in particular in the case of two random variables

$$\text{Var}[X + Y] = \text{Cov}[X + Y, X + Y] = \text{Var}[X] + \text{Var}[Y] + 2 \text{Cov}[X, Y]$$

If all  $X_i$  are independent (or more generally uncorrelated), the variance of the sum of random variables is equal to the sum of the variances of each random variable

$$\text{Var}[X_1 + \dots + X_n] = \text{Var}[X_1] + \dots + \text{Var}[X_n] \quad (X_1, \dots, X_n \text{ independent})$$

### 3.4. Bivariate Normal Distribution

The most important two-dimensional distribution is the bivariate normal distribution. It is completely defined by

- the expected values and variances of the marginal distributions :  $\mu_X, \sigma_X^2$  and  $\mu_Y, \sigma_Y^2$
- the covariance between  $X$  and  $Y$  :  $\text{Cov}[X, Y] = \rho_{XY}\sigma_X\sigma_Y$

The joint density is given by the function

$$f_{X,Y}(x, y) = \frac{1}{2\pi\sqrt{\det(\Sigma)}} \exp \left\{ -\frac{1}{2}(x - \mu_X, y - \mu_Y)\Sigma^{-1} \begin{pmatrix} x - \mu_X \\ y - \mu_Y \end{pmatrix} \right\}$$

where the matrix  $\Sigma$  contains all the information about the variances and covariances. This is the so-called *covariance matrix* defined as

$$\Sigma = \begin{pmatrix} \text{Cov}[X, X] & \text{Cov}[X, Y] \\ \text{Cov}[Y, X] & \text{Cov}[Y, Y] \end{pmatrix} = \begin{pmatrix} \sigma_X^2 & \rho_{XY}\sigma_X\sigma_Y \\ \rho_{XY}\sigma_X\sigma_Y & \sigma_Y^2 \end{pmatrix}$$

Figure 3.5 displays the contour lines for the case  $\mu_X = \mu_Y = 0$ ,  $\sigma_X = \sigma_Y = 1$  and different values of  $\text{Cov}[X, Y]$ . By means of  $\mu_X \neq 0$  and  $\mu_Y \neq 0$  we would be able to “move around” the distribution.

It can be shown that the marginal distributions are again normal distributions

$$X \sim \mathcal{N}(\mu_X, \sigma_X^2) \quad \text{and} \quad Y \sim \mathcal{N}(\mu_Y, \sigma_Y^2)$$

If  $\text{Cov}[X, Y] = 0$ , then  $\Sigma$  is a diagonal matrix. We can easily verify that the condition for the independence of the random variables

$$f_{X,Y}(x,y) = f_X(x) \cdot f_Y(y), \quad x, y \in \mathbb{R}$$

is fulfilled in this case. That means: in the case of a two-dimensional normal distribution, the converse of the following statement

$$X \text{ and } Y \text{ independent} \Rightarrow \text{Cor}[X, Y] = 0$$

now holds : From uncorrelatedness  $\text{Cor}[X, Y] = 0$  follows in this case the independence of  $X$  und  $Y$ . In general however, this is not true, see Figure 3.4!

## 3.5. Principal Component Analysis<sup>1</sup>

*Principal component analysis* (PCA) is a popular approach for deriving a low-dimensional set of features from a large set of variables. In particular, PCA is a technique for reducing the dimension of a  $n \times p$  data matrix  $X$  where  $n$  corresponds to the number of observations and  $p$  to the number of variables.

The *first principal component* direction of the data is that along which the observations *vary the most*. We illustrate the use of PCA on the **USArrests** data set.

### Example 3.5.1 USArrests

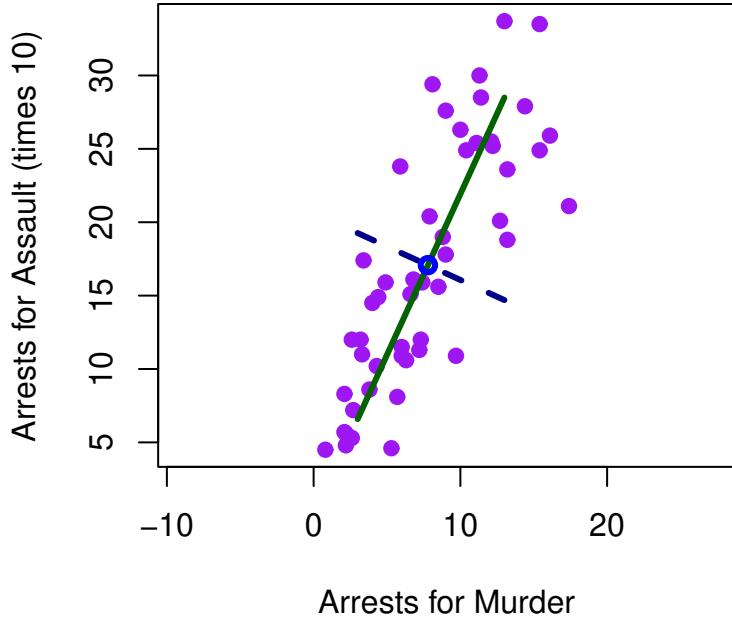
For each of the 50 states in the United States, the data set contains the number of arrests per 100'000 residents for each of three crimes : **Assault**, **Murder** and **Rape**.

For instance, consider Figure 3.6, which shows the number of arrests for **Assault** in tens versus the number of arrests for **Murder** for each of the 50 states in the United States. The green solid line represents the *first principal component direction* of the data.

---

<sup>1</sup>This chapter follows Chapter 6.3 of the text book *An Introduction to Statistical Learning: with Applications in R* by G. James et al., Springer Texts in Statistics 2013.

### Assault versus Murder Rate, US, 1973



**Figure 3.6.:** Number of arrests for **Assault** in tens versus the number of arrests for **Murder** per 100'000 residents for each of the 50 states in the United States. The green solid line indicates the first principal component direction, and the blue dashed line indicates the second principal component direction.

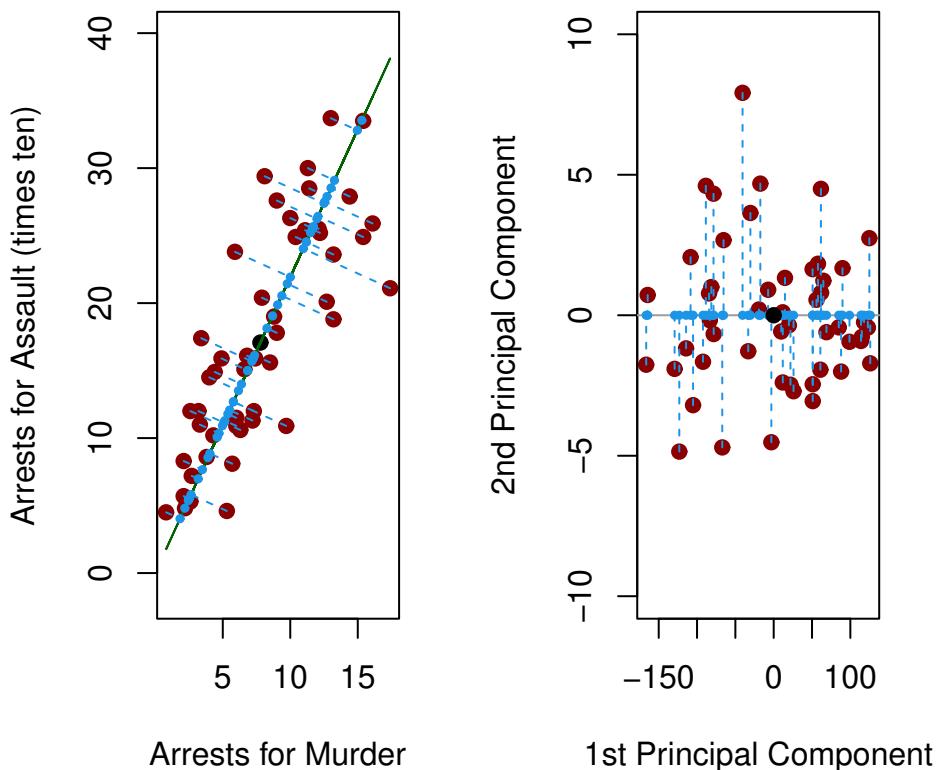
We can see by eye that this is the direction along which there is the greatest variability in the data. That is, if we *projected* the 50 observations onto this line (see Figure 3.7), then the resulting projected observations would have the largest possible variance; projecting the observations onto any other line direction would yield projected observations with lower variance.

The first principal component direction is displayed graphically in Figure 3.6, but how can it be summarized mathematically? If we consider the first principal component direction as a coordinate axis, then any point (**Murder**, **Assault**) has the following first principal component coordinate when projected onto this line

$$Z_1 = -0.0419126 \cdot (\text{Murder} - \overline{\text{Murder}}) - 0.9991213 \cdot (\text{Assault} - \overline{\text{Assault}}) \quad (3.1)$$

where  $\overline{\text{Murder}}$  indicates the mean of all **Murder** values in this data set, and  $\overline{\text{Assault}}$  indicates the mean of all arrests for assault. Thus, the point  $(\overline{\text{Murder}}, \overline{\text{Assault}})$  corresponds to the origin on the first principal component axis.

Here  $\phi_{11} = -0.0419126$  and  $\phi_{21} = -0.9991213$  are the *principal component loadings* which define the direction to the above. How can these values be found? The idea



**Figure 3.7.:** *Left:* The first principal component direction is shown in green. It is the dimension along which the data vary the most, and it also defines the line that is closest to all  $n$  observations. The distances from each observation to the principal component are represented using the blue dashed line segments. The black circle represents ([Murder](#), [Assault](#)). *Right:* The left-hand panel has been rotated (counter-clockwise) so that the first principal component direction coincides with the x-axis.

### Chapter 3. Joint Distributions

is that out of every possible linear combination of **Murder** and **Assault** such that  $\phi_{11}^2 + \phi_{21}^2 = 1$ , this particular linear combination yields the highest variance: i.e. this is the linear combination for which

$$\text{Var} [\phi_{11} \cdot (\text{Murder} - \overline{\text{Murder}}) + \phi_{21} \cdot (\text{Assault} - \overline{\text{Assault}})]$$

is *maximized*. It is necessary to consider only linear combinations of the form  $\phi_{11}^2 + \phi_{21}^2 = 1$ , since otherwise we could increase  $\phi_{11}$  and  $\phi_{21}$  arbitrarily in order to blow up the variance.

We determine the first principal component loadings in [Python](#) as follows:

(to R)

```
[1]: import pandas as pd
      from sklearn.decomposition import PCA

      # Read Data
      arrests = pd.read_csv('./data/USArrests.csv')
      states = arrests.iloc[:, 0]
      features = ['Murder', 'Assault']
      # keep only arrests on Murder and Assault
      arrests = arrests.loc[:, features]

      # Create PCA instance and fit the data
      pca = PCA()
      pca.fit(arrests)

      # Print the first principal component loadings:
      print("First PCL's: \t", features, "\n\t\t", pca.components_[0])
```

First PCL's: ['Murder', 'Assault']  
[0.0419126 0.99912128]

By default, the `sklearn.decomposition.PCA()` function centers the variables to have mean zero. This corresponds to how we defined the first principal component.

Since  $n = 50$ , **Murder** and **Assault** are vectors of length 50, and so is  $Z_1$  in equation 3.1. For instance,

$$z_{i1} = -0.0419126 \cdot (\text{Murder}_i - \overline{\text{Murder}}) - 0.9991213 \cdot (\text{Assault}_i - \overline{\text{Assault}})$$

The values of  $z_{i1}, \dots, z_{n1}$  are known as the *principal component scores*, and can be seen in right-hand panel of Figure 3.7. We can compute them in [Python](#) in the following way: (to R)

```
[2]: # Find Principle component scores:
      principalComponents = pca.fit_transform(arrests)
      # Turn into DataFrame
      principalDf = pd.DataFrame(data = principalComponents,
```

### Chapter 3. Joint Distributions

```
columns = ['PC1', 'PC2'], index = states)
# print results:
print("Number of Principle component scores:\n",
      len(principalComponents[:,0]))
print(principalDF.head())
```

Number of Principle component scores:  
50

	PC1	PC2
Unnamed: 0		
Alabama	65.409503	2.672866
Alaska	92.251658	-1.655962
Arizona	123.144783	-4.853583
Arkansas	19.265509	0.204712
California	105.198322	-3.199947

The first column contains the values  $z_{i1}$ , the second column the values  $z_{i2}$ .

We can think of the values of the principal component  $Z_1$  as single-number summaries of the joint **Murder** and **Assault** arrests for each state. In this example, if

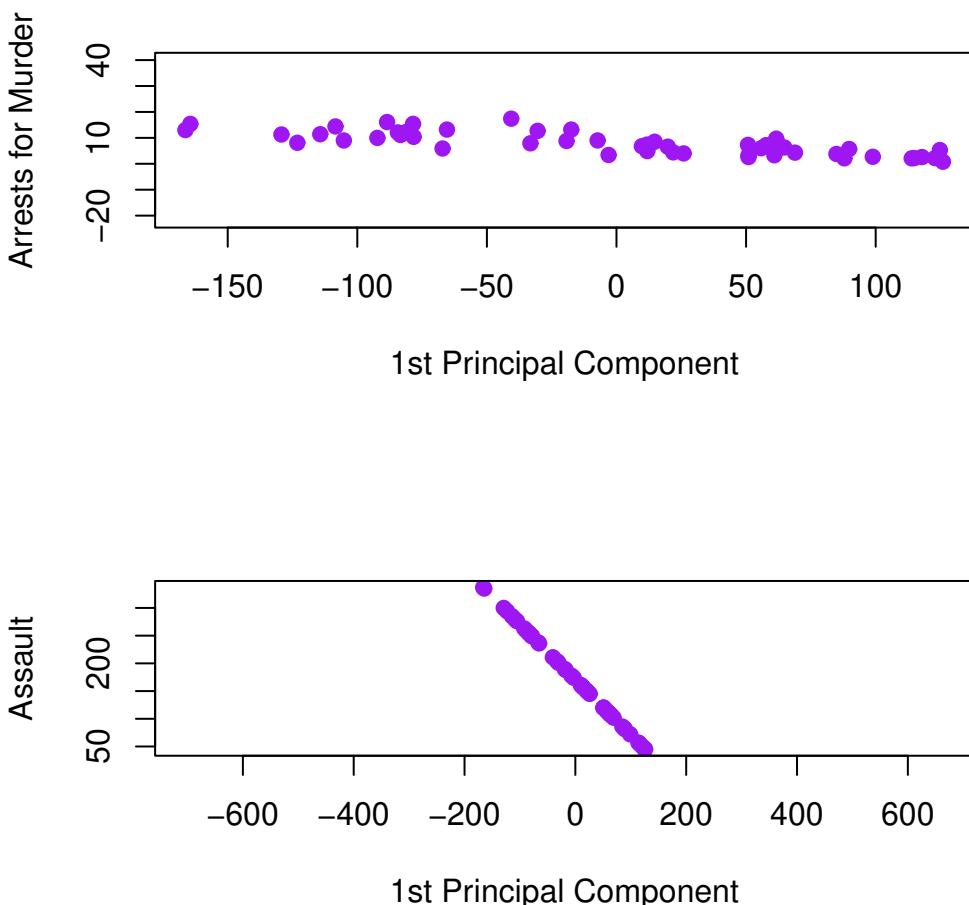
$$z_{i1} = -0.0419126 \cdot (\text{Murder}_i - \overline{\text{Murder}}) - 0.9991213 \cdot (\text{Assault}_i - \overline{\text{Assault}}) > 0$$

then this indicates a state with below-average arrests for murder and below-average arrests for assault. A negative score suggests the opposite. How well can a single number represent both **Murder** and **Assault**? In this case, Figure 3.6 indicates that **Murder** and **Assault** have approximately a linear relationship, and so we might expect that a single-number summary will work well. Figure 3.8 displays  $z_{i1}$  versus both **Murder** and **Assault**. The plots show a strong relationship between the first principal component and the two features. In other words, the first principal component appears to capture most of the information contained in the **Murder** and **Assault** variables.

So far we have concentrated on the first principal component. In general, one can construct up to  $p$  distinct principal components. The second principal component  $Z_2$  is a linear combination of predictor variables and is uncorrelated with  $Z_1$ , and has largest variance subject to this constraint. The second principal component direction is illustrated as a dashed blue line in Figure 3.6. It turns out that the zero correlation condition of  $Z_1$  with  $Z_2$  is equivalent to the condition that the direction must be *perpendicular*, or *orthogonal* to the first principal component direction. The second principal component is given by the formula

$$Z_2 = 0.9991213 \cdot (\text{Murder} - \overline{\text{Murder}}) - 0.0419126 \cdot (\text{Assault} - \overline{\text{Assault}})$$

We compute the second principal component loadings in Python as follows: ([to R](#))



**Figure 3.8.:** Plots of the first principal component scores  $z_{i1}$  versus **Murder** and **Assault**. The relationships are strong.

### Chapter 3. Joint Distributions

```
[3]: print("Second PCL's: \t", features, "\n\t\t", pca.components_[1])
```

```
Second PCL's:      ['Murder', 'Assault']
                  [ 0.99912128 -0.0419126 ]
```

Again, any point ( $\text{Murder}_i, \text{Assault}_i$ ) projected onto the second principal component direction yields the second principal component coordinate  $z_{i2}$ .

The point ( $\text{Murder}, \text{Assault}$ ) corresponds again to the origin of the second principal component axis and hence to the origin of the new coordinate system defined by the two principal components.

Since we now have considered only two variables in dataset **USArrests**, that first two principal components contain all the information that is in **Murder** and **Assault**. However, by construction, the first component will contain the most information. Consider, for example, the much larger variability of  $z_{i1}$  (x-axis) versus  $z_{i2}$  (the y-axis) in the right-hand panel of Figure 3.7. The fact that the second principal component scores are much closer to zero indicates that this component captures far less information. As another illustration, Figure 3.9 displays  $z_{i2}$  versus **Murder** and **Assault**. There is little relationship between the second principal component and these two variables, again suggesting that in this case, one only needs the first principal component in order to accurately represent the arrests for **Murder** and **Assault**.



With two-dimensional data, such as in our **USArrests** example, we can construct at most two principal components. However, if we had other variables, such as **Rape**, then additional components could be constructed. They would successively maximize variance, subject to the constraint of being uncorrelated with the preceding components.

### PCA and Covariance Matrix

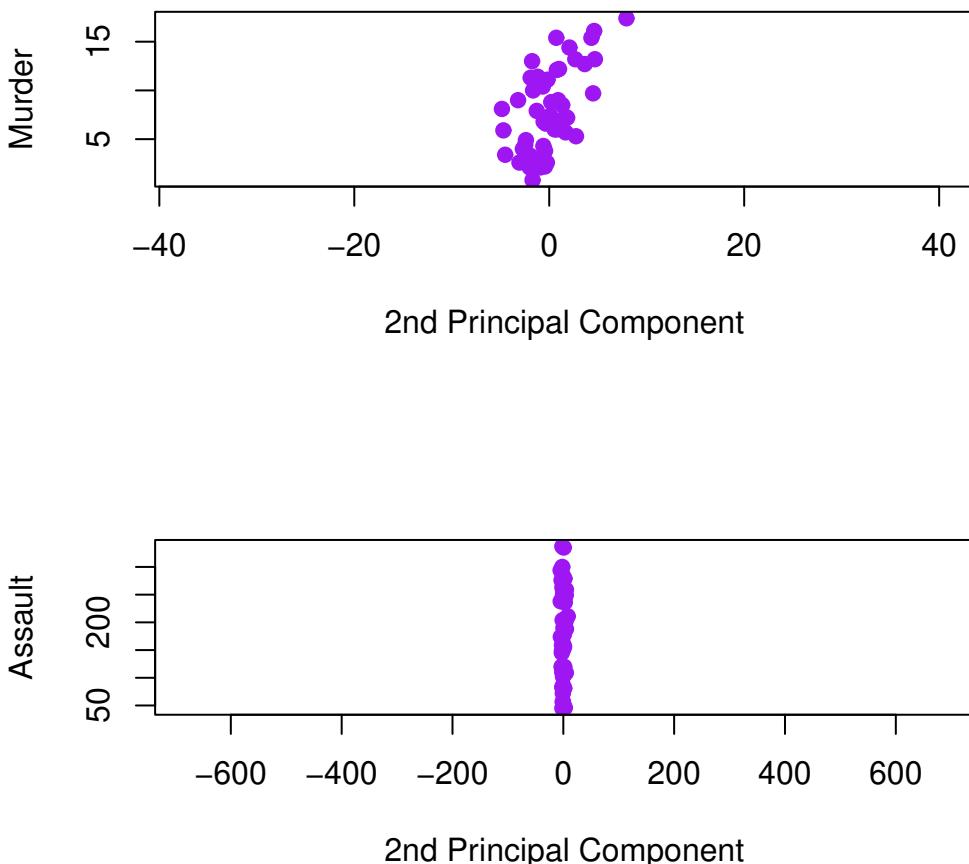
How is the procedure to find the principal components related to the discussion concerning the covariance matrix in Chapter 3.3?

The covariance matrix of two random variables  $X$  and  $Y$  is defined as

$$\Sigma = \begin{pmatrix} \text{Cov}[X, X] & \text{Cov}[X, Y] \\ \text{Cov}[Y, X] & \text{Cov}[Y, Y] \end{pmatrix} = \begin{pmatrix} \sigma_X^2 & \text{Cov}[X, Y] \\ \text{Cov}[X, Y] & \sigma_Y^2 \end{pmatrix}$$

Since we required the two principal components  $Z_1$  and  $Z_2$  to be uncorrelated, this implies for their covariance matrix  $\Sigma$  to have vanishing off-diagonal elements. In other words, we are confronted with the task of diagonalizing the covariance matrix.

### Chapter 3. Joint Distributions



**Figure 3.9.:** Plots of the second principal component scores  $z_{i2}$  versus **Murder** and **Assault**. The relationships are weak.

### Chapter 3. Joint Distributions

To do so, we carry out a basis transformation, such that in the new coordinate system, the covariance matrix appears as a diagonal matrix. This can be achieved by an appropriate rotation matrix  $\Phi$  so that

$$\begin{pmatrix} Z_1 \\ Z_2 \end{pmatrix} = (X - \mu_X, Y - \mu_Y) \cdot \begin{pmatrix} \phi_{11} & \phi_{12} \\ \phi_{21} & \phi_{22} \end{pmatrix} = \begin{pmatrix} \phi_{11}(X - \mu_X) + \phi_{21}(Y - \mu_Y) \\ \phi_{12}(X - \mu_X) + \phi_{22}(Y - \mu_Y) \end{pmatrix}$$

and

$$\begin{pmatrix} \text{Cov}[Z_1, Z_1] & \text{Cov}[Z_1, Z_2] \\ \text{Cov}[Z_2, Z_1] & \text{Cov}[Z_2, Z_2] \end{pmatrix} = \begin{pmatrix} \sigma_{Z_1}^2 & 0 \\ 0 & \sigma_{Z_2}^2 \end{pmatrix}$$

In the example **USArrests**, for the random variables  $X$  and  $Y$  we considered **Murder** and **Assault**, respectively. The rotation matrix  $\Phi$  needs to satisfy the condition  $\phi_{11}^2 + \phi_{21}^2 = 1$  and  $\phi_{12}^2 + \phi_{22}^2 = 1$ .

It is straightforward to generalize the case of  $p = 2$  to an arbitrary  $p$ .

### Proportion of Variance Explained by Principal Components

We can now ask a natural question: how much of the information in a given data set is lost by projecting the observations onto the first few principal components? That is, how much of the variance in the data is *not* contained in the first few principal components? More generally, we are interested in knowing the *proportion of variance explained* (PVE) by each principal component. The *total variance* present in a data set (assuming that the variables have been centered to have mean zero) is defined as

$$\sum_{j=1}^p \text{Var}[X_j] = \sum_{j=1}^p \frac{1}{n} \sum_{i=1}^n x_{ij}^2$$

and the variance of the  $m$ th principal component is

$$\frac{1}{n} \sum_{i=1}^n z_{im}^2 = \frac{1}{n} \sum_{i=1}^n \left( \sum_{j=1}^p \phi_{jm} x_{ij} \right)^2$$

Therefore, the proportion of variance explained (PVE) by the  $m$ th principal component is given by

$$\frac{\sum_{i=1}^n \left( \sum_{j=1}^p \phi_{jm} x_{ij} \right)^2}{\sum_{j=1}^p \sum_{i=1}^n x_{ij}^2}$$

The PVE of each principal component is a positive quantity. In order to compute the cumulative PVE of the first  $M$  principal components, we can simply sum over the first  $M$  PVEs. In total, there are  $\min(n - 1, p)$  principal components, and their PVEs sum to one.

#### Example 3.5.2 USArrests

In the **USArrests** data, the first principal component explains 99.9 percent of the variance in the data, and the next principal component explains 0.1 percent of the variance, as can directly be figured out in Python (to R)

```
[1]: import pandas as pd
from sklearn.decomposition import PCA

# Read Data
arrests = pd.read_csv('./data/USArrests.csv')
states = arrests.iloc[:,0]
features = ['Murder', 'Assault']
# keep only arrests on Murder and Assault
arrests = arrests.loc[:, features]

# Create PCA instance and fit the data
pca = PCA()
pca.fit(arrests)

pve = pca.explained_variance_ratio_

print("Proportion of Variance Explained =\n", pve.round(4))
```

Proportion of Variance Explained =  
[0.999 0.001]

Most of the information of the data about the arrests for **Murder** and **Assault** is contained in the first principal component. ◀

## Conceptual Objectives

You should be able...

- to explain the concept of multivariate probability distributions, as well for discrete as for continuous random variables.
- to compute on the basis of the joint probability distribution the conditional and marginal probability distributions, including the summary statistics.
- to explain why the correlation is a measure for the linear dependency for two random variables.
- to compute the variance and expected value of a linear combination of (dependent) random variables.

### Chapter 3. Joint Distributions

- to explain the concept of principal component analysis, in particular the meaning of the first principal component direction and how the second principal component is related to the first one.
- to explain how the covariance matrix is affected by changing from the original coordinates to the principal component coordinates.

## Computational Objectives

You should be able...

- to make a PCA analysis in `Python` by means of the functions `sklearn.decomposition.PCA()` and `PCA().fit()`.
- to access the different attributes and methods of `PCA()`, in particular `.fit().components_` and `.fit().explained_variance_ratio_`.

# **Part I.**

## **Regression Analysis**

# Chapter 4.

## Introduction to Regression Analysis<sup>1</sup>

### 4.1. What is Regression?

In order to motivate our study of Linear Regression, we begin with a simple example.

#### Example 4.1.1

Suppose that we are statistical consultants hired by a client to provide advice and a marketing strategy on how to improve sales of a particular product. This client provides us with the **Advertising** data set that consists of the **sales** of that product in 200 different markets, along with advertising budgets for the product in each of those markets for three different media : **TV**, **radio** and **newspaper**. The data are displayed in Figure 4.1.

(to R)

```
[1]: import pandas as pd
import matplotlib.pyplot as plt

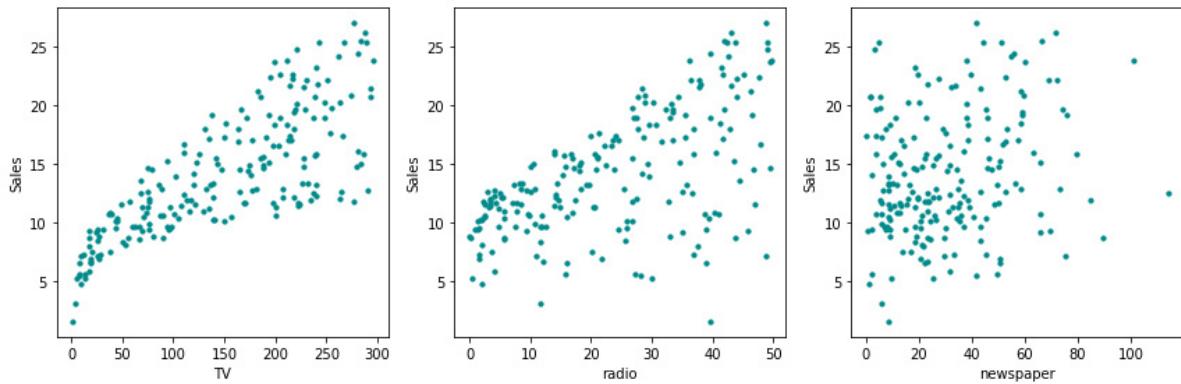
# Load data
df = pd.read_csv('./data/Advertising.csv')
labels = df.columns[1:]

# Create figure and plot data:
fig = plt.figure(figsize=(12, 4))
for i in range(3):
    ax = fig.add_subplot(1, 3, i+1)
    ax.scatter(df[labels[i]], df['sales'],
               marker='o', color='darkcyan', s=10)
    ax.set_xlabel(labels[i])
    ax.set_ylabel('Sales')

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

---

<sup>1</sup>This chapter follows Chapters 2.1 and 2.2 of the text book *An Introduction to Statistical Learning: with Applications in R* by G. James et al., Springer Texts in Statistics 2013.



**Figure 4.1.:** The plot displays **sales**, in thousands of units, as a function of **TV**, **radio** and **newspaper** for 200 different markets.

It is not possible for our client to directly increase sales of the product. On the other hand, they can control the advertising expenditure in each of the three media. Therefore, if we determine that there is an association between advertising budgets and sales, then we can instruct our client to adjust advertising budget, thereby indirectly increasing sales. In other words, our goal is to develop an accurate *model* that can be used to *predict* sales on the basis of the three media budgets.

Let us have a look at left-hand panel of Figure 4.1 : we clearly see a relationship between the advertising budget and the sales of that product. The more money is invested for the advertising budget, the higher are the sales. We now can raise the question: which kind of relationship can we observe in this case? One possibility may be that the points are scattered around a line. We will have a closer look at this question in the next chapter.

In the right-hand panel of Figure 4.1 we can hardly observe any relationship between the advertising budget and the sales. Most of the points are rather randomly distributed in the scatterplot. ◀

From a mathematical perspective, we are looking for a function  $f$ , that predicts the **sales**  $Y$  based on the three advertising budgets  $X_1$  (**TV**),  $X_2$  (**radio**) and  $X_3$  (**newspaper**), respectively:

$$Y \approx f(X_1, X_2, X_3)$$

In this relation, there is no equal sign since the plots do not represent graphs of a function. Therefore, the function  $f$  represents the relation between  $X_1$ ,  $X_2$ ,  $X_3$  and  $Y$  only *approximately*.

### Terminology and Notation: Predictor, Response

In this setting, the advertising budgets are *input variables* while **sales** is an *output variable*. The input variables are typically denoted using the symbol  $X$  with a subscript to distinguish them, such as  $X_1$ ,  $X_2$  and  $X_3$ . The inputs go by different names,

## Chapter 4. Introduction to Regression Analysis

such as *predictors*, *independent variables*, *features* or sometimes just *variables*. The output variable - in this example - is often called the *response* or *dependent variable*, and is typically denoted using the symbol  $Y$ . Throughout these lecture notes, we will use all of these terms interchangeably.

□

More generally, suppose that we observe a quantitative response  $Y$  and  $p$  different predictors,  $X_1, X_2, \dots, X_p$ . We assume that there is some relationship between  $Y$  and  $X_1, X_2, \dots, X_p$ , which can be written in the very general form

$$Y = f(X_1, X_2, \dots, X_p) + \varepsilon$$

Here  $f$  is some fixed but *unknown* function of  $X_1, X_2, \dots, X_p$  and  $\varepsilon$  is a *random error term*, which is independent of  $X_1, X_2, \dots, X_p$  and has mean zero. In this formulation,  $f$  represents the *systematic* information that  $X$  provides about  $Y$ . To understand the meaning of the error term  $\varepsilon$ , we will have a look at another example:

### Example 4.1.2

Consider the left-hand panel of Figure 4.2 that displays the **income** of 30 individuals versus **years of education**. The plot suggests that one might be able to predict **income** using **years of education**.

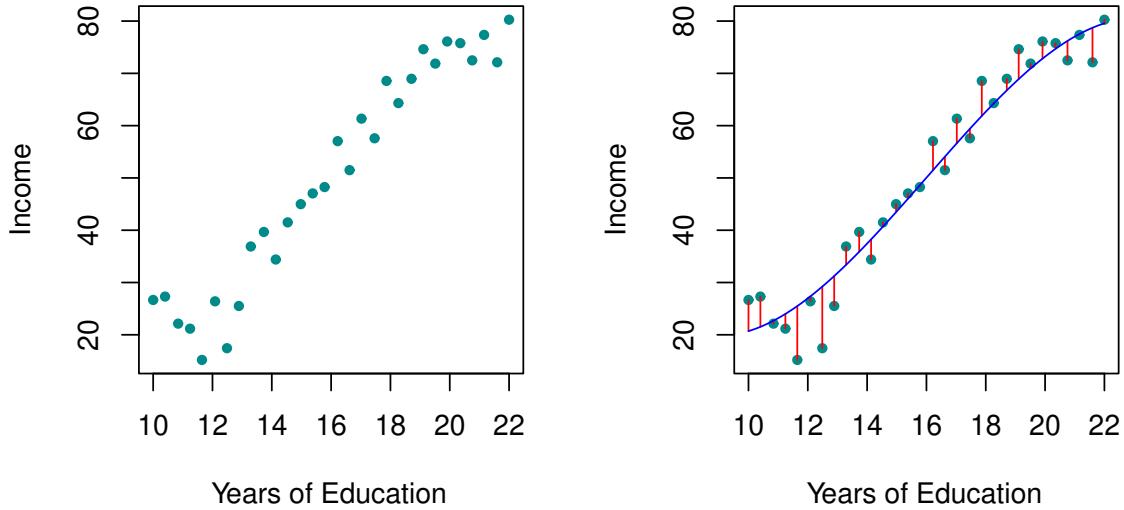
However, the function  $f$  that connects the input variable to the output variable is in general unknown. In this situation one must estimate  $f$  based on the observed points. Since **Income** is a simulated data set,  $f$  is known and is shown by the blue curve in the right-hand panel of Figure 4.2. The vertical (red) lines represent the error terms  $\varepsilon$ . We note that some of the 30 observations lie above the blue curve and some lie below it. Overall, the errors have approximately mean zero. ◀

In essence, regression refers to an approach for *estimating*  $f$ . In this chapter, we outline some of the key theoretical concepts that arise in estimating  $f$ , as well as tools for evaluating the estimates obtained.

### Terminology and Notation:

In general, we denote an estimated quantity with a hat  $\hat{\cdot}$ . As consequence,  $\hat{Y}$  represents the estimation of the (unknown) variable  $Y$  or  $\hat{f}$  is the estimation of the (unknown) function  $f$ .

□



**Figure 4.2.:** The `Income` data set. *Left:* The green dots are the observed values of `income` (in tens of thousands of Swiss francs) and `years of education` for 30 individuals. *Right:* The blue curve represents the true underlying relationship between `income` and `years of education`, which is generally unknown (but is known in this case because the data were simulated). The red lines represent the error associated with each observation. Note that some errors are positive (if an observation lies above the blue curve) and some are negative (if an observation lies below the curve). Overall, these errors have approximately mean zero.

## 4.2. Why Estimate $f$ ?

There are two main reasons that we may wish to estimate  $f$ : *prediction* and *inference*. In the first case, we are interested to predict data points based for some new values of the input variables. In the other case, we are interested to understand how the response variable is affected by a change of the input variables.

### 4.2.1. Prediction

In many situations, a set of predictors  $X_1, X_2, \dots, X_p$  are readily available, but the response  $Y$  cannot be easily obtained. In this setting, since the error term averages to zero, we can predict  $Y$  using

$$\hat{Y} = \hat{f}(X_1, X_2, \dots, X_p)$$

where  $\hat{f}$  represents our estimate for  $f$ , and  $\hat{Y}$  represents the resulting prediction for  $Y$ . In this setting,  $\hat{f}$  is often treated as a *black box*, in the sense that one is not typically concerned with the exact form of  $\hat{f}$ , provided that it yields accurate predictions for  $Y$ .

**Example 4.2.1**

Suppose that  $X_1, X_2, \dots, X_p$  are characteristics of a patient's blood sample that can be easily measured in a lab, and  $Y$  is a variable encoding the patient's risk for a severe adverse reaction to a particular drug. It is natural to seek to predict  $Y$  using  $X_1, X_2, \dots, X_p$ , since we can then avoid giving the drug in question to patients who are at high risk of an adverse reaction - that is, patients for whom the estimate of  $Y$  is high.

◀

The accuracy of  $\hat{Y}$  as a prediction for  $Y$  depends on two quantities, which we will call the *reducible error* and the *irreducible error*. In general,  $\hat{f}$  will not be a perfect estimate for  $f$ , and this inaccuracy will introduce some error. This error is *reducible* because we can potentially improve the accuracy of  $\hat{f}$  by using the most appropriate regression technique to estimate  $f$ . However, even if it were possible to form a perfect estimate for  $f$ , so that our estimated response took the form

$$\hat{Y} = f(X_1, X_2, \dots, X_p)$$

our prediction would still have some error in it. This is because  $Y$  is also a function of  $\varepsilon$ , which by definition cannot be predicted using  $X_1, X_2, \dots, X_p$ . Therefore, variability associated with  $\varepsilon$  also affects the accuracy of our predictions. This is known as the *irreducible error*, because no matter how well we estimate  $f$ , we cannot reduce the error introduced by  $\varepsilon$ .

Why is the irreducible error larger than zero? The quantity  $\varepsilon$  may contain unmeasured variables that are useful in predicting  $Y$ : since we don't measure them,  $f$  cannot use them for its prediction. The quantity  $\varepsilon$  may also contain unmeasurable variation. For example, the risk of an adverse reaction might vary for a given patient on a given day, depending on manufacturing variation in the drug itself or the patient's general feeling of well-being on that day.

Consider a given estimate  $\hat{f}$  and a set of predictors  $X_1, X_2, \dots, X_p$ , which yields the prediction

$$\hat{Y} = \hat{f}(X_1, X_2, \dots, X_p)$$

Assume for a moment that both  $\hat{f}$  and  $X$  are fixed. Then, it is easy to show that

$$\begin{aligned} E[(Y - \hat{Y})^2] &= E[(f(X) + \varepsilon - \hat{f}(X))^2] \\ &= E[(f(X) - \hat{f}(X))^2 + 2\varepsilon(f(X) - \hat{f}(X)) + \varepsilon^2] \\ &= \underbrace{(f(X) - \hat{f}(X))^2}_{\text{Reducible}} + \underbrace{\text{Var}[\varepsilon]}_{\text{Irreducible}} \end{aligned}$$

where  $E[(Y - \hat{Y})^2]$  represents the average, or *expected value*, of the squared difference between the predicted and actual value of  $Y$ , and  $\text{Var}[\varepsilon]$  represents the *variance* associated with the error term  $\varepsilon$ .

The focus of these lecture notes is on techniques for estimating  $f$  with the aim of minimizing the reducible error. It is important to keep in mind that the irreducible error will always provide an upper bound on the accuracy of our prediction for  $Y$ . This bound is almost always unknown in practice.

### 4.2.2. Inference with Respect to $f$

In the previous section, we were interested in how precisely  $f$  would predict new observations. In this section, we are interested in understanding the way that  $Y$  is affected as  $X_1, X_2, \dots, X_p$  change. In this situation we wish to estimate  $f$ , but our goal is not necessarily to make predictions for  $Y$ . We instead want to understand the relationship between  $X$  and  $Y$ , or more specifically, to understand how  $Y$  changes as a function of  $X_1, X_2, \dots, X_p$ .

Now  $\hat{f}$  cannot be treated as a black box, because we need to know its exact form. In this setting, one may be interested in answering the following questions:

- *Which predictors are associated with the response?*

It is often the case that only a small fraction of the available predictors are substantially associated with  $Y$ . Identifying the few *important* predictors among a large set of possible variables can be extremely useful, depending on the application.

In the example **Advertising** (see Figure 4.1 on page 94), the expenditure for **TV** has an impact on **sales**, whereas the expenditure for **newspaper** alone does not seem to have any influence. If we had to decide in which single medium we would invest, then it would be **TV**, rather than **newspaper**.

- *What is the relationship between the response and each predictor?*

Some predictors may have a positive relationship with  $Y$ , in the sense that increasing the predictor is associated with increasing values of  $Y$ . Other predictors

may have the opposite relationship. Depending on the complexity of  $f$ , the relationship between the response and a given predictor may also depend on the values of the other predictors.

- Can the relationship between  $Y$  and each predictor be adequately summarized using a linear equation, or is the relationship more complicated?

Historically, most methods for estimating  $f$  have taken a linear form. In some situations, such an assumption is reasonable or even desirable. But often the true relationship is more complicated, in which case a linear model may not provide an accurate representation of the relationship between the input and output variables.

Consider the **Advertising** data illustrated in Figure on page 94. One may be interested in answering questions such as

- Which media contribute to sales?
- Which media generate the biggest boost in sales?
- How much increase in sales is associated with a given increase in TV advertising?

### 4.3. How do we Estimate $f$ ?

There are several methods how  $f$  may be estimated. Broadly speaking, most statistical methods for this task can be characterized as either *parametric* or *non-parametric*. In the following, we will focus on parametric methods.

Parametric methods involve a two-step model-based approach:

1. First, we make an assumption about the functional form, or shape, of  $f$ . For example, one very simple assumption is that  $f$  is linear in  $X_1, X_2, \dots, X_p$ :

$$f(X_1, X_2, \dots, X_p) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p \quad (4.1)$$

This is a *linear model*, which will be discussed extensively in Chapter 5. Once we have assumed that  $f$  is linear, the problem of estimating  $f$  is greatly simplified. Instead of having to estimate an entirely arbitrary  $p$ -dimensional function  $f(X_1, X_2, \dots, X_p)$ , one only needs to estimate the  $p + 1$  coefficients  $\beta_0, \beta_1, \dots, \beta_p$ .

2. After a model has been selected, we need a procedure that uses the training data to *fit* the model 4.1. In the case of the linear model, we need to estimate the parameters  $\beta_0, \beta_1, \dots, \beta_p$ . That is, we want to find values of these parameters such that

$$Y \approx \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p$$

The most common approach to fitting the model 4.1 is referred to as (*ordinary*) *least squares*, which we discuss in Chapter 5. However, least squares is one of many possible ways to fit the linear model.

The model-based approach just described is referred to as *parametric*; it reduces the problem of estimating  $f$  down to one of estimating a set of parameters. Assuming a parametric form for  $f$  simplifies the problem of estimating  $f$  because it is generally much easier to estimate a set of parameters such as  $\beta_0, \beta_1, \dots, \beta_p$  in the linear model 4.1, than it is to fit an entirely arbitrary function  $f$ .

### Example 4.3.1

For the **Advertising** data the linear model 4.1 is given by:

$$\text{sales} \approx \beta_0 + \beta_1 \cdot \text{TV} + \beta_2 \cdot \text{radio} + \beta_3 \cdot \text{newspaper}$$



### Example 4.3.2

For the **Income** data, the linear model 4.1 then is

$$\text{income} \approx \beta_0 + \beta_1 \cdot \text{education}$$



### Example 4.3.3

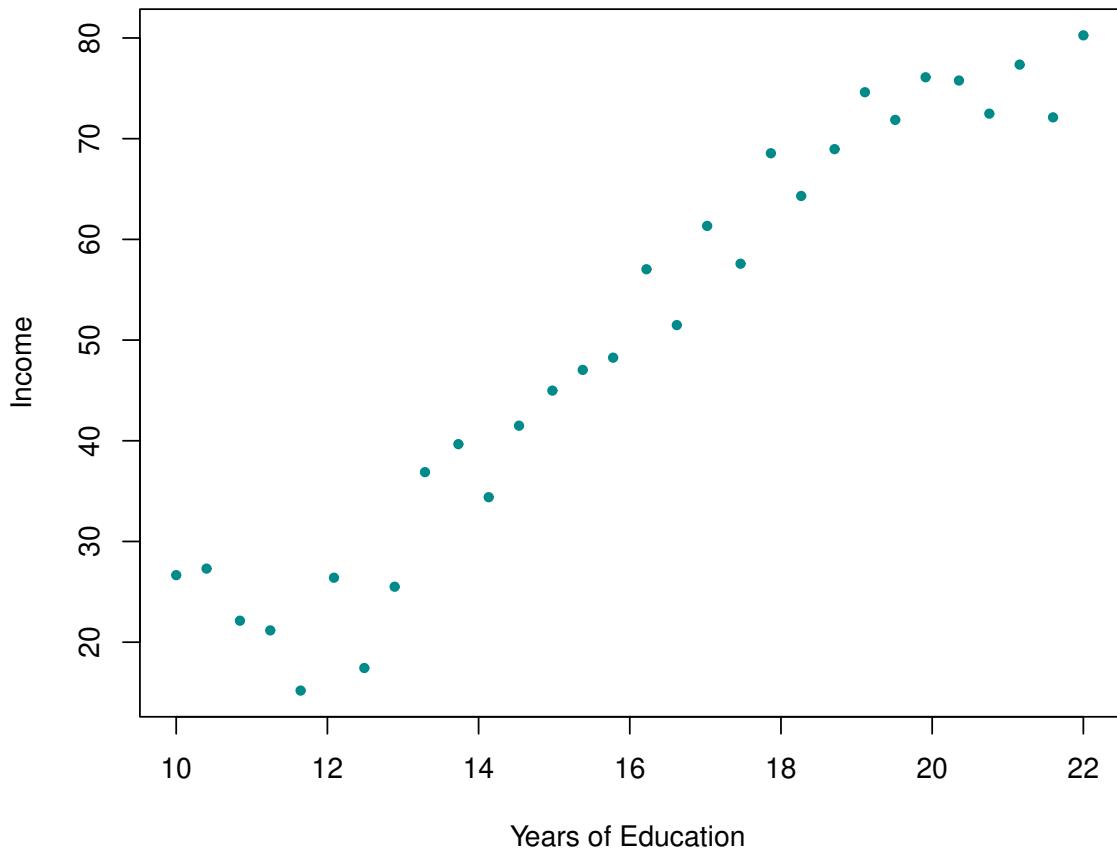
Let us have again a look at the **Income** data set which is displayed in Figure 4.3.

We are now facing the question which *model* best fits the data, or more precisely, which is the shape of  $f$ . If we have a look at the plot of the data, then a linear model may be considered:

$$f(X) = \beta_0 + \beta_1 X$$

The straight line, described by the above equation, is displayed in the left-hand panel of Figure 4.4. However, we may as well consider a cubic model (3. grade polynomial):

$$f(X) = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3$$

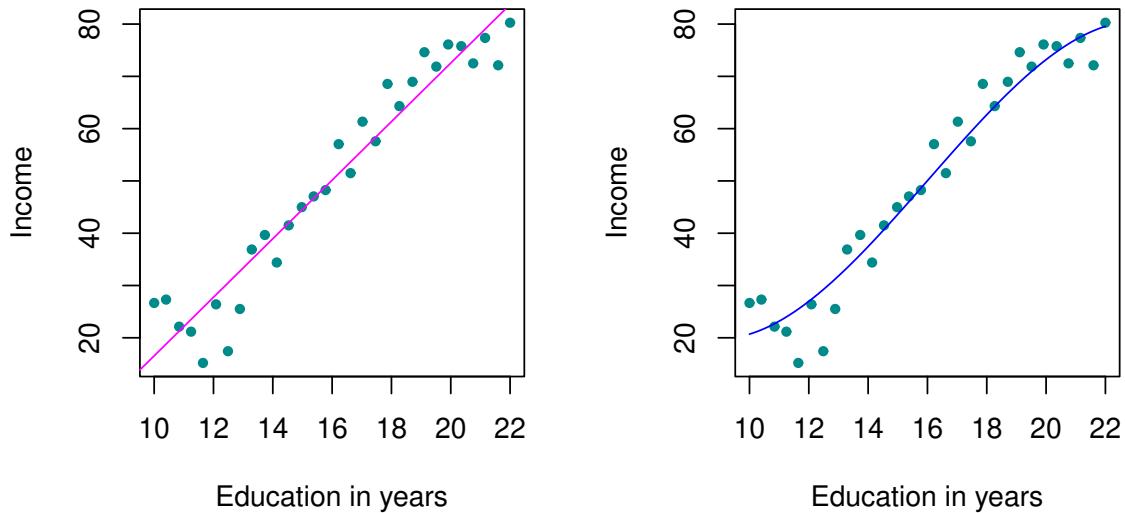


**Figure 4.3.:** Income as a function of Years of Education.

The plot of  $f$  is displayed in the right-hand panel of Figure 4.4.

There are many more models we could think of. But based on which criteria are we going to choose the “right” one? This question will hardly ever be answered in an absolute manner. The function  $f$  is in general unknown, and therefore it is up to us to choose the “best” model. Statistics provides us with a number of methods to rely our decision on.

Which model is now the better one in the case of the **Income** data? Comparing the left-hand panel with the right-hand panel of Figure 4.4, we can see that the linear fit is not quite right: the true  $f$  has some curvature that is not captured in the linear fit. But the linear model has a number of advantages: the parameters  $\beta_0$  and  $\beta_1$  allow for a simple geometric interpretation, in particular,  $\beta_0$  corresponds to the intercept with the  $y$ -axis and  $\beta_1$  corresponds to the slope of the straight line.  $\beta_0$  thus corresponds to the income of a person with 0 years of education and  $\beta_1$  corresponds to the additional



**Figure 4.4.:** Two estimates for the underlying function  $f$  for the `Income` data. *Left:* A linear model fit by least squares. *Right :* A cubic linear fit by least squares.

income you achieve by having an additional year of education. ◀

#### Remarks:

- i. In general, fitting a more flexible model requires estimating a greater number of parameters. These more complex models can lead to a phenomenon known as *overfitting* the data, which essentially means they follow the errors, or *noise*, too closely.
- ii. In many cases, a linear model leads to reasonable results. The following chapter will discuss *Linear Regression*. ◆

# Chapter 5.

## Simple Linear Regression<sup>1</sup>

### 5.1. Model for Simple Linear Regression

#### 5.1.1. Introduction

Recall the **Advertising** data from Chapter 4. Figure 4.1 displays **sales** (in thousands of units) for a particular product as a function of advertising budgets (in thousands of Swiss francs) for **TV**, **radio** and **newspaper** media. Suppose that in our role as statistical consultants we are asked to suggest, on the basis of this data, a marketing plan for next year that will result in high product sales. What information would be useful in order to provide such a recommendation? Here are a few questions that we might seek to address:

1. *Is there a relationship between advertising budget and sales?*

Our first goal should be to determine whether the data provide evidence of an association between advertising expenditure and sales. If the evidence is weak, then one might argue that no money should be spent on advertising.

2. *How strong is the relationship between advertising budget and sales?*

Assuming that there is a relationship between advertising and sales, we would like to know the strength of this relationship. In other words, given a certain advertising budget, can we predict sales with a high level of accuracy? This would be a strong relationship. Or is a prediction of sales based on advertising expenditure only slightly better than a random guess? This would be a weak relationship.

3. *Which media contribute to sales?*

Do all three media - TV, radio, and newspaper - contribute to sales, or do just one or two of the media contribute? To answer this question, we must find a

---

<sup>1</sup>This chapter follows Chapter 3.1 of the text book *An Introduction to Statistical Learning: with Applications in R* by G. James et al., Springer Texts in Statistics 2013.

## Chapter 5. Simple Linear Regression

way to separate out the individual effects of each medium when we have spent money on all three media.

4. *How accurately can we estimate the effect of each medium on sales?*

For every Swiss franc spent on advertising in a particular medium, by what amount will sales increase? How accurately can we predict this amount of increase?

5. *How accurately can we predict future sales?*

For any given level of television, radio, or newspaper advertising, what is our prediction for sales, and what is the accuracy of this prediction?

6. *Is the relationship linear?*

If there is approximately a straight-line relationship between advertising expenditure in the various media and sales, then linear regression is an appropriate tool. If not, then it may still be possible to transform the predictor or the response so that linear regression can be used.

7. *Is there synergy among the advertising media?*

Perhaps spending CHF 50 000 on television advertising and CHF 50 000 on radio advertising results in more sales than allocating CHF 100 000 to either television or radio individually. In marketing, this is known as a *synergy* effect, while in statistics it is called an *interaction* effect.

It turns out that linear regression can be used to answer each of these questions. We will first discuss all of these questions in a general context, and then return to them in this specific context.

### 5.1.2. Simple Linear Regression Model

*Simple linear regression* lives up to its name: it is a very straightforward approach for predicting a quantitative response  $Y$  on the basis of a single predictor variable  $X$ . It assumes that there is approximately a linear relationship between  $X$  and  $Y$ . Mathematically, we can write this linear relationship as

$$Y \approx \beta_0 + \beta_1 X \tag{5.1}$$

You might read “ $\approx$ ” as “is approximately modeled as”. We will sometimes describe equation 5.1 by saying that we are *regressing  $Y$  on  $X$*  (or  *$Y$  onto  $X$* ).

**Example 5.1.1**

In the example with the **Advertising** data set,  $X$  may represent **TV** advertising and  $Y$  may represent **sales**. Then, we can regress **sales** onto **TV** by fitting the model

$$\text{sales} \approx \beta_0 + \beta_1 \cdot \text{TV}$$



In equation 5.1,  $\beta_0$  and  $\beta_1$  are two unknown constants that represent the *intercept* and *slope* terms in the linear model. Together,  $\beta_0$  and  $\beta_1$  are known as the model *coefficients* or *parameters*. Once we have used our data to produce estimates  $\hat{\beta}_0$  and  $\hat{\beta}_1$  for the model coefficients, we can predict future sales on the basis of a particular value of TV advertising by computing

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$$

where  $\hat{y}$  indicates a prediction of  $Y$  on the basis of  $X = x$ . Here we use a *hat* symbol,  $\hat{\phantom{x}}$ , to denote the estimated value for an unknown parameter or coefficient, or to denote the predicted value of the response.

## 5.2. Estimating the Coefficients

In practice,  $\beta_0$  and  $\beta_1$  are unknown. So before we can use equation 5.1 to make predictions, we must use data to estimate the coefficients. Let

$$(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$$

represent  $n$  observation pairs, each of which consists of a measurement of  $X$  and a measurement of  $Y$ .

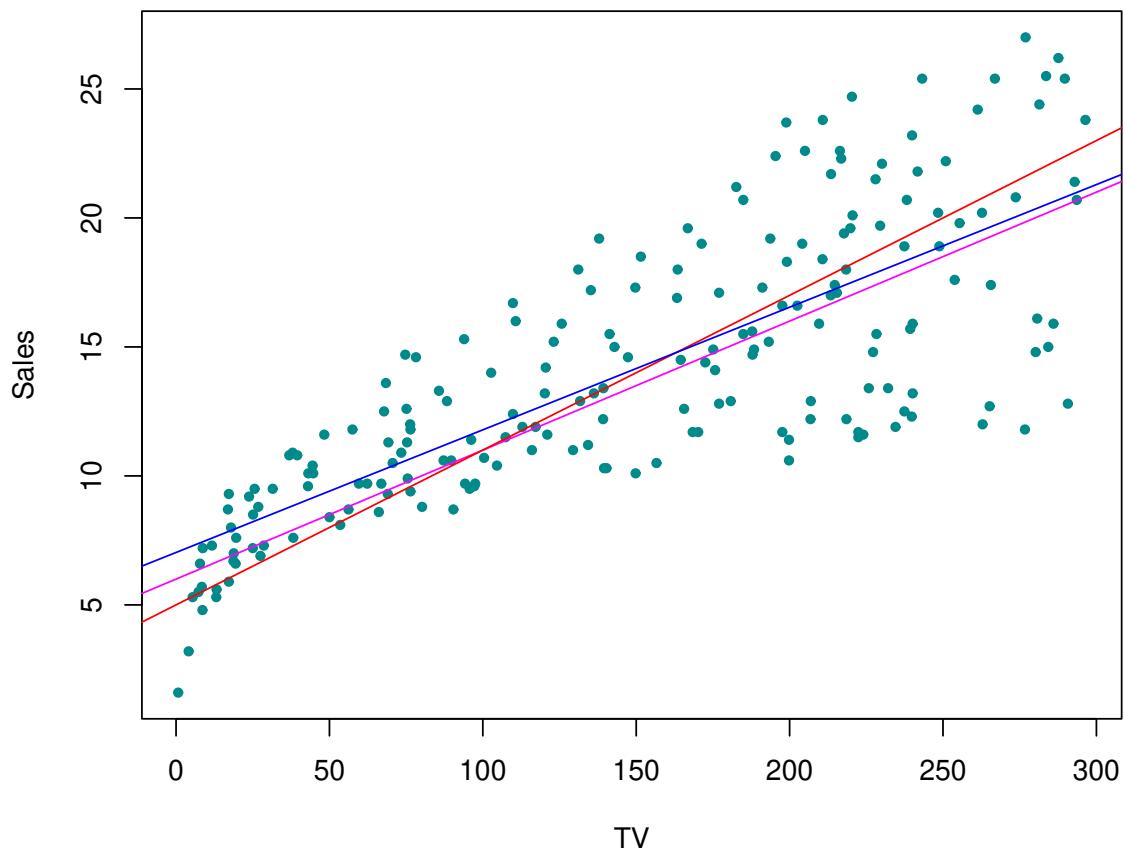
**Example 5.2.1**

In the **Advertising** example, this data set consists of the TV advertising budget and product sales in  $n = 200$  different markets, thus of  $n = 200$  observation pairs. The  $x$ -coordinate refers to the TV advertising budget and the corresponding  $y$ -coordinate refers to the product sales (see the left-hand panel of Figure 4.1 on page 94). ◀

Our goal is to obtain coefficient estimates  $\hat{\beta}_0$  and  $\hat{\beta}_1$  such that the linear model  $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$  fits the available data as well as possible. That is,

$$y_i \approx \hat{\beta}_0 + \hat{\beta}_1 x_i$$

for all  $i = 1, \dots, n$ . On the left side of the previous approximate equation is the measured value, on the right side is the corresponding  $y$ -value on the regression line. In other words, we want to find an intercept  $\hat{\beta}_0$  and a slope  $\hat{\beta}_1$  such that the resulting line is as close as possible to the  $n = 200$  data points. However, how do we measure closeness?

**Example 5.2.2**


**Figure 5.1.:** For the `Advertising` data, different regression lines that may fit the data points are displayed. Which is the “best” one?

Figure 5.1 displays three straight lines that may fit the data points. Based on which criterion are we going to choose the “best” line?

## Chapter 5. Simple Linear Regression

By far the most common approach to find the best fitting line involves minimizing the *least squares* criterion, and we take that approach in this chapter.

Let

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i$$

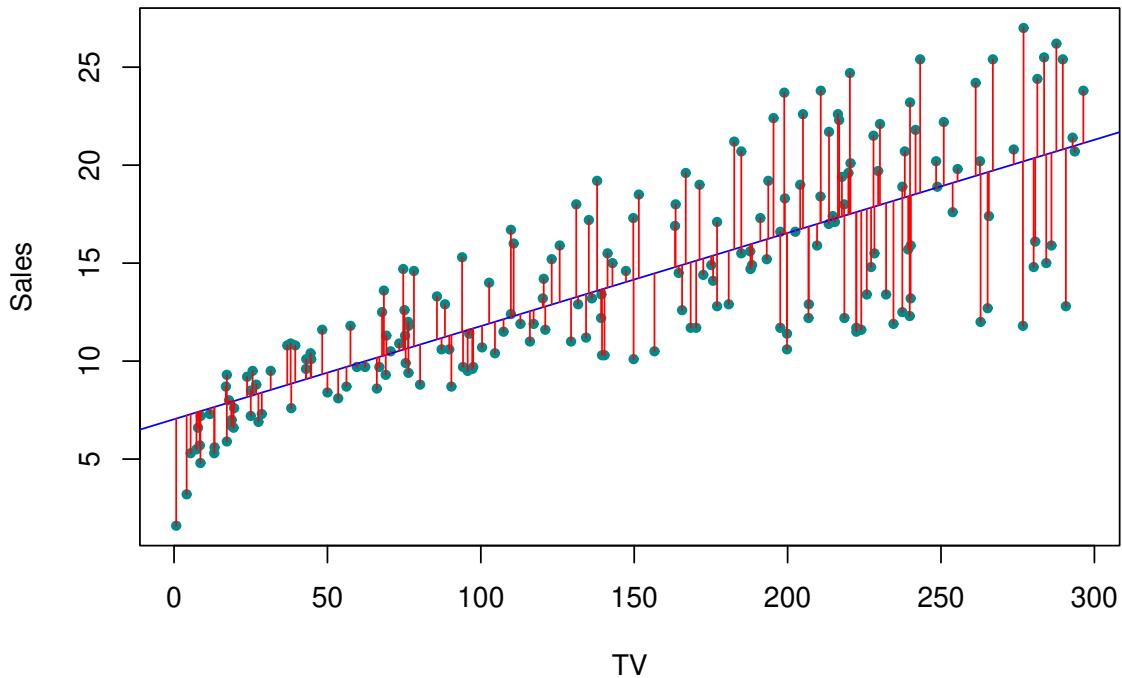
be the prediction for  $Y$  based on the  $i$ th value of  $X$ . Then

$$r_i = y_i - \hat{y}_i$$

represents the  $i$ th *residual* - this is the difference between the  $i$ th observed response value and the  $i$ th response value that is predicted by our linear model.

### Example 5.2.3

Figure 5.2 displays the residuals as red line segments. The residuals above the (blue) line have positive sign, the residuals below the regression line have negative sign.



**Figure 5.2.:** For the **Advertising** data, the least squares fit for the regression of **sales** onto **TV** is shown (blue line). The fit is found by minimizing the sum of squared residuals that are represented by the red line segments.

## Chapter 5. Simple Linear Regression

The *sum* of all residuals does not represent a good measure for the distance of the data points to the regression line, because positively and negatively signed residuals may sum up to zero. However, if we consider the sum of the *squared* residuals, we obtain a much more reliable measure for the distance of data points to the regression line. We define the *residual sum of squares* (RSS) as

$$\text{RSS} = r_1^2 + r_2^2 + \dots + r_n^2$$

or equivalently as

$$\text{RSS} = (y_1 - \hat{\beta}_0 - \hat{\beta}_1 x_1)^2 + (y_2 - \hat{\beta}_0 - \hat{\beta}_1 x_2)^2 + \dots + (y_n - \hat{\beta}_0 - \hat{\beta}_1 x_n)^2$$

The least squares approach chooses  $\hat{\beta}_0$  and  $\hat{\beta}_1$  to minimize the RSS. Using some calculus, one can show that the minimizers are :

$$\begin{aligned}\hat{\beta}_1 &= \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2} \\ \hat{\beta}_0 &= \bar{y} - \hat{\beta}_1 \bar{x}\end{aligned}\tag{5.2}$$

where

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i \quad \text{and} \quad \bar{y} = \frac{1}{n} \sum_{i=1}^n y_i$$

are the sample means.

In other words, equation 5.2 defines the *least squares coefficient estimates* for simple linear regression.

### Example 5.2.4

We are interested in the values of  $\hat{\beta}_0$  and  $\hat{\beta}_1$  for the **Advertising** data. By means of the **Python**-function `OLS()`, we can easily determine those values: ([to R](#))

```
[1]: import pandas as pd
import statsmodels.api as sm

# Load data
df = pd.read_csv('./data/Advertising.csv')
x = df['TV']
y = df['sales']
```

## Chapter 5. Simple Linear Regression

```
# Linear Regression using statsmodels.api
x_sm = sm.add_constant(x)
model = sm.OLS(y, x_sm).fit()

# Now we can print a summary,
print(model.summary())
```

```
OLS Regression Results
=====
Dep. Variable: sales R-squared: 0.612
Model: OLS Adj. R-squared: 0.610
Method: Least Squares F-statistic: 312.1
Date: Fri, 26 Feb 2021 Prob (F-statistic): 1.47e-42
Time: 10:30:27 Log-Likelihood: -519.05
No. Observations: 200 AIC: 1042.
Df Residuals: 198 BIC: 1049.
Df Model: 1
Covariance Type: nonrobust
=====
            coef    std err          t      P>|t|      [0.025      0.975]
-----
const    7.0326   0.458     15.360      0.000     6.130     7.935
TV       0.0475   0.003     17.668      0.000     0.042     0.053
=====
Omnibus: 0.531 Durbin-Watson: 1.935
Prob(Omnibus): 0.767 Jarque-Bera (JB): 0.669
Skew: -0.089 Prob(JB): 0.716
Kurtosis: 2.779 Cond. No. 338.
=====
```

The coefficients are listed under `(coef)`. Here `const` corresponds to  $\hat{\beta}_0$ , thus to the intercept with the  $y$ -axis, and the `TV` corresponds to the slope of the regression line, thus to  $\hat{\beta}_1$ .

Our regression model then is given by

$$Y \approx 7.03 + 0.0475X$$

According to this approximation, an additional CHF 1000 spent on TV advertising is associated with selling approximately 47.5 additional units of the product.

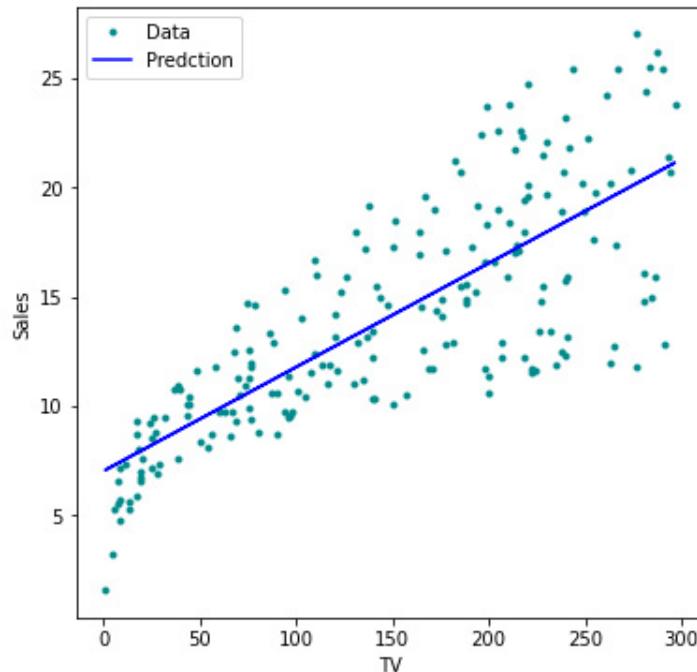
Figure 5.3 displays the Python-code for plotting the regression line for the **Advertising** data, including the plot. [\(to R\)](#)

```
[2]: import matplotlib.pyplot as plt

# Predicted y
y_pred = model.predict(x_sm)

# Create figure and plot
fig = plt.figure(figsize=(6, 6))
ax = fig.add_subplot(1, 1, 1)
plt.plot(df['TV'], y, marker='o', linestyle='None',
         color='darkcyan', markersize='3', label="Data")
plt.plot(df['TV'], y_pred, 'b-', label="Prediction")
```

```
# Set labels and Legend
ax.set_xlabel('TV')
ax.set_ylabel('Sales')
plt.legend()
plt.show()
```



**Figure 5.3.:** For the `Advertising` data, the least squares fit for the regression of `sales` onto `TV` is shown.



### Remarks:

- i. The Python-function `statsmodels.api.OLS()` uses Ordinary Least Squares to fit the *linear model*.
- ii. Be aware of the (default) order of the function parameters `x` and `y` in `sm.OLS(y, x)`.
- iii. If `x` is a one-dimensional vector, it needs to be preprocessed using `sm.add_constant(x)`.



## 5.3. Hypothesis Tests and Confidence Intervals

### 5.3.1. Assessing the Accuracy of the Coefficient Estimates

Recall that we assume that the *true* relationship between  $X$  and  $Y$  takes the form

$$Y = f(X) + \varepsilon$$

for some unknown function  $f$ , where  $\varepsilon$  is a mean-zero random error term. If  $f$  is to be approximated by a linear function, then we can write this relationship as

$$Y = \beta_0 + \beta_1 X + \varepsilon$$

Here  $\beta_0$  is the intercept term - that is, the expected value of  $Y$  when  $X = 0$ , and  $\beta_1$  is the slope - the average increase in  $Y$  associated with a one-unit increase in  $X$ . The error term is a catch-all for what we miss with this simple model:

- the true relationship is probably not linear,
- there may be other variables that cause variation in  $Y$ ,
- there may be measurement error.

The central limit theorem tells us that the sum of all those random variables is approximately distributed according to a normal distribution. Furthermore, we typically assume that the error term is independent of  $X$ .

#### Example 5.3.1

In this example we assume that we know the *true* relationship between  $X$  and  $Y$ , which is

$$f(X) = 2 + 3X$$

However, in practice we will never observe this perfect relationship, but

$$Y = f(X) + \varepsilon = 2 + 3X + \varepsilon$$

We call  $f(X) = 2 + 3X$  the *population regression line*, which is the best linear approximation to the true relationship between  $X$  and  $Y$ . If we now observe realizations of  $X$  and  $Y$ , we can determine the *least squares line*

$$Y = \beta_0 + \beta_1 X + \varepsilon$$

by estimating  $\beta_0$  and  $\beta_1$  according to equation 5.2.

Let us simulate the observed data  $X$  and  $Y$  from the model

$$Y = 2 + 3X + \varepsilon$$

where  $\varepsilon$  is normally distributed with mean 0, thus  $\varepsilon \sim \mathcal{N}(0, \sigma^2)$ . We create 100 random values of  $X$ , and generate 100 corresponding values of  $Y$  from the model  $Y = 2 + 3X + \varepsilon$  (to R)

## Chapter 5. Simple Linear Regression

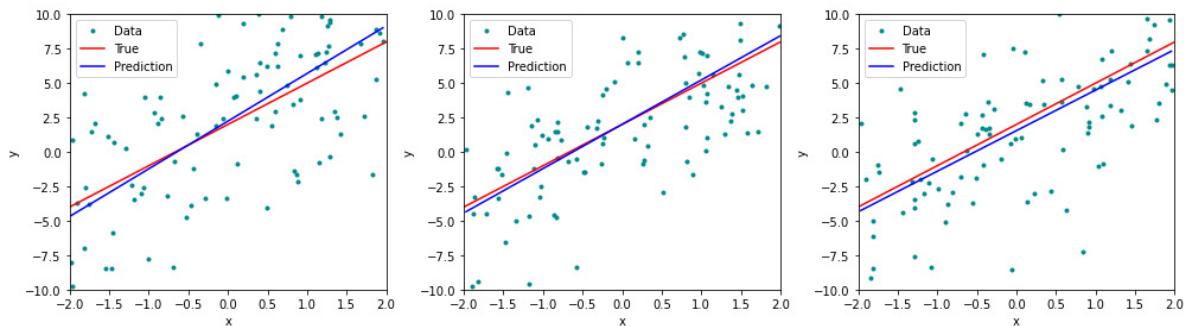
```
[1]: import numpy as np

n = 100 # number of datapoints

# create data around y=ax+b
a, b = 3, 2
x = np.sort(np.random.uniform(low=-2, high=2, size=(n)))
y = b + a*x + np.random.normal(loc=0.0, scale=4, size=n)

x_true = np.linspace(-2, 2, n)
y_true = b + a*x_true
```

Figure 5.4 displays 3 simulations, each containing new simulated data according to the above procedure. ([to R](#))



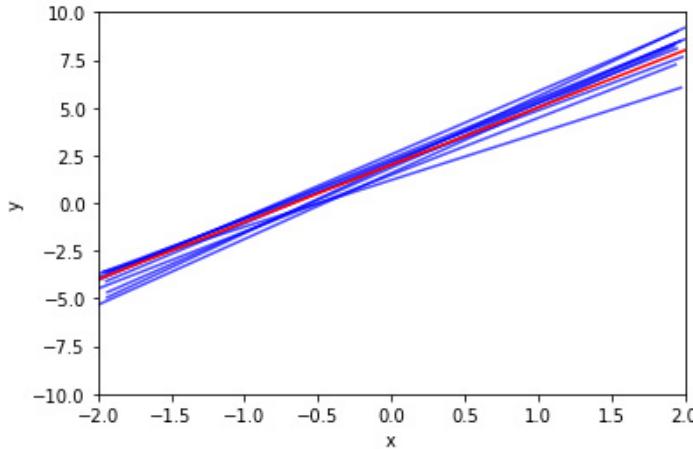
**Figure 5.4.:** Each panel corresponds to a simulated data set. The red line represents the true relationship  $f(X) = 2 + 3X$ , which is known as the population regression line. The blue line is the least squares line: it is the least squares estimate for  $f(X) = \beta_0 + \beta_1 x$  based on the observed (simulated) data.

The red line represents the plot of the function  $f(X) = 2 + 3X$ , which remains identical in all three simulations. The blue line is the least squares line that was constructed by means of the least squares estimates for the simulated data. Each least squares line is different, but on average, the least squares lines are quite close to the population regression line.

In Figure 5.5 we have generated ten different data sets from the model given by  $Y = 2 + 3X + \epsilon$  and plotted the corresponding ten least squares lines. Notice that different data sets generated from the same true model result in slightly different least squares lines, but the unobserved population regression line does not change.



At first glance, the difference between the population regression line and the least squares line may seem subtle and confusing. We only have one data set, and so what does it mean that two different lines describe the relationship between the predictor and the response? Fundamentally, the true relationship between  $X$  and  $Y$  can never



**Figure 5.5.:** Ten least squares lines are shown in blue, each computed on the basis of a separate random set of observations generated from the model  $Y = 2 + 3X + \varepsilon$ . The red line represents the true relationship  $f(X) = 2 + 3X$ .

be revealed based on real data of limited size. The concept of these two lines is a natural extension of the standard statistical approach of using information from a sample to estimate characteristics of a large population.

In practice, we have data based on which we determine the least squares line. This will not be the true linear relationship between predictor and response - still assuming that the true relationship actually is linear - which will always remain unknown to us. With respect to our example: the red line represents the true but unknown linear relationship. Based on our data we will find one of the blue lines. Fortunately, based on the sample we are able to figure out what the true line may look like.

### Example 5.3.2

Suppose that we are interested in knowing the population mean  $\mu$  of the height of all 20 years old men. Unfortunately,  $\mu$  is unknown and it is impossible to measure the height of all 20 years old men, but we do have access to  $n$  observations from  $Y$ , which we can write as  $y_1, \dots, y_n$ , and which we can use to estimate  $\mu$ . Let us assume we have a group of  $n = 1000$  men of age 20 whose height  $y$  we have measured:  $y_i$  for  $i = 1, \dots, 1000$ . We now easily can compute the average height of this group  $\bar{y}_n$ . A reasonable estimate for  $\mu$  then is

$$\hat{\mu} = \bar{y}_n$$

thus

$$\mu \approx \hat{\mu} = \bar{y}_n$$

If we had chosen a different group, then the sample mean  $\bar{y}_n$  would be a different one. The sample mean and the population mean are different, but in general the sample mean will provide a good estimate of the population mean :  $\mu \approx \bar{y}_n$ . ◀

In the same way, the unknown coefficients  $\beta_0$  and  $\beta_1$  in linear regression define the population regression line. We seek to estimate these unknown coefficients using  $\hat{\beta}_0$  and  $\hat{\beta}_1$ . These coefficient estimates define the least squares lines. If we estimate  $\beta_0$  and  $\beta_1$  on the basis of a particular data set, then our estimates won't be exactly  $\beta_0$  and  $\beta_1$ . On the basis of one particular set of observations,  $\hat{\beta}_0$  and  $\hat{\beta}_1$  may overestimate  $\beta_0$  and  $\beta_1$ , on the basis of another set of observations,  $\hat{\beta}_0$  and  $\hat{\beta}_1$  may underestimate  $\beta_0$  and  $\beta_1$ . But if we could average the estimates  $\hat{\beta}_0$  and  $\hat{\beta}_1$  over a huge number of data sets, then the average of these estimates would be exactly  $\beta_0$  and  $\beta_1$ . If this is the case, we call these estimators *unbiased*, since they do not *systematically* over- or underestimate the true parameter.

A natural question is as follows : how accurate is the estimation? We continue the analogy with the estimation of the population mean  $\mu$  of a random variable  $Y$ . In particular, let us consider again the height of 20 years old men. How accurate is the sample mean  $\hat{\mu}$  on the basis of a group of 20 years old men as an estimate of  $\mu$ ? We have established that the average of  $\hat{\mu}$ 's over many data sets will be very close to  $\mu$ , but that a single estimate  $\hat{\mu}$  may be a substantial underestimate or overestimate of  $\mu$ . How far off will that single estimate of  $\hat{\mu}$  be? In general, we answer this question by computing the *standard error* of  $\hat{\mu}$ , written as  $\text{se}(\hat{\mu})$ . We have the well-known formula

$$\text{Var}[\hat{\mu}] = \text{se}(\hat{\mu})^2 = \frac{\sigma^2}{n}$$

where  $\sigma$  is the standard deviation of each of the realizations  $y_i$  of  $Y$  - this formula holds provided that the  $n$  observations are uncorrelated. Roughly speaking, the standard error tells us the average amount that this estimate  $\hat{\mu}$  differs from the actual value of  $\mu$ . The formula above also tells us how this deviation shrinks with  $n$  - the more observations we have, the smaller the standard error of  $\hat{\mu}$

In a similar vein, we can wonder how close  $\hat{\beta}_0$  and  $\hat{\beta}_1$  are to the true values  $\beta_0$  and  $\beta_1$ . To compute the standard errors associated with  $\hat{\beta}_0$  and  $\hat{\beta}_1$ , we use the following formulas:

$$\text{se}(\hat{\beta}_0)^2 = \sigma^2 \left( \frac{1}{n} + \frac{\bar{x}^2}{\sum_{i=1}^n (x_i - \bar{x})^2} \right) \quad \text{and} \quad \text{se}(\hat{\beta}_1)^2 = \frac{\sigma^2}{\sum_{i=1}^n (x_i - \bar{x})^2}$$

where  $\sigma^2 = \text{Var}[\varepsilon]$

### Remarks:

- For these formulas to be strictly valid, we need to assume that the errors  $\varepsilon_i$  for each observation are uncorrelated with common variance  $\sigma^2$ . This is clearly not true in Figure 5.2, but the formula still turns out to be a good approximation.

- ii. If the error terms  $\varepsilon_i$  are normally distributed, then  $\hat{\beta}_0$  and  $\hat{\beta}_1$  follow a normal distribution as well. By indicating their expectation and variance as specified above, the distributions of  $\hat{\beta}_0$  and  $\hat{\beta}_1$  are fully determined.
- iii. Notice in the formula that  $\text{se}(\hat{\beta}_0)$  is smaller when the  $x_i$  are more spread out; intuitively we have more *leverage* to estimate a slope when this is the case.
- iv. In general,  $\sigma^2$  (variance of the error term) is not known, but can be estimated from the data. This estimate is known as the *residual standard error* (RSE), and is given by the formula

$$\text{RSE} = \sqrt{\frac{\text{RSS}}{n-2}} \equiv \sqrt{\frac{r_1^2 + r_2^2 + \dots + r_n^2}{n-2}}$$

where RSS denotes the sum of the squared residuals  $r_i$ . The factor  $1/(n-2)$  was chosen so that the estimation of  $\sigma$  turns out to be unbiased.

- v. Strictly speaking, when  $\sigma^2$  is estimated from the data we should write  $\hat{\text{se}}(\hat{\beta}_1)$  to indicate that an estimate has been made, but for simplicity of notation we will drop this extra “hat”.

◆

These results also show how a good experimental design can help to improve the quality of the estimates  $\hat{\beta}_0$  and  $\hat{\beta}_1$ . In particular, we can obtain a more precisely determined regression line:

1. by increasing the number of observations  $n$
2. by making sure that the predictors  $x_i$  scatter with large amplitude
3. by using a suitable predictor in order to keep  $\sigma^2$  small
4. by having an average predictor value  $\bar{x}$  close to zero in order to improve the estimate  $\hat{\beta}_0$ .

Some further useful properties of the least squares estimates are that the regression line passes through the center of gravity  $(\bar{x}, \bar{y})$  and that the residuals sum up to zero :  $\sum_i r_i = 0$ .

### Example 5.3.3

For the **Advertising** data we can determine the standard errors of the estimated coefficients with the help of Python: ([to R](#))

```
[1]: import pandas as pd
import statsmodels.api as sm

# Load data
```

## Chapter 5. Simple Linear Regression

```
df = pd.read_csv('./data/Advertising.csv')
x = df['TV']
y = df['sales']

# Linear Regression using statsmodels.api
x_sm = sm.add_constant(x)
model = sm.OLS(y, x_sm).fit()

# Now we can print a summary,
print(model.summary())
```

```
OLS Regression Results
=====
Dep. Variable: sales R-squared: 0.612
Model: OLS Adj. R-squared: 0.610
Method: Least Squares F-statistic: 312.1
Date: Fri, 26 Feb 2021 Prob (F-statistic): 1.47e-42
Time: 11:12:03 Log-Likelihood: -519.05
No. Observations: 200 AIC: 1042.
Df Residuals: 198 BIC: 1049.
Df Model: 1
Covariance Type: nonrobust
=====
            coef    std err          t      P>|t|      [0.025    0.975]
-----
const    7.0326   0.458     15.360      0.000     6.130    7.935
TV       0.0475   0.003     17.668      0.000     0.042    0.053
=====
Omnibus: 0.531 Durbin-Watson: 1.935
Prob(Omnibus): 0.767 Jarque-Bera (JB): 0.669
Skew: -0.089 Prob(JB): 0.716
Kurtosis: 2.779 Cond. No. 338.
```

In the table under `coef`, the listed values are  $\hat{\beta}_0$  and  $\hat{\beta}_1$ . In the column `Std. Error` we find the values 0.457843 and 0.002691 for the two standard errors  $se(\hat{\beta}_0)$  and  $se(\hat{\beta}_1)$ . They correspond to the average deviations of the estimated values of  $\hat{\beta}_0$  and  $\hat{\beta}_1$  which are 7.0326 and 0.04754. In the right top part, we find some constants describing the fit, for example `R-squared` which is the portion of variance explained by the fit. ◀

### 5.3.2. Hypothesis Test

Standard errors can be used to perform *hypothesis tests* on the coefficients. The most common hypothesis test involves testing the *null hypothesis*

$H_0$  : There is *no* relationship between X and Y

versus the *alternative hypothesis*

$H_A$  : There is *some* relationship between X and Y

Mathematically, this corresponds to testing

$$H_0 : \beta_1 = 0$$

versus

$$H_A : \beta_1 \neq 0$$

since if  $\beta_1 = 0$  then the model  $Y = \beta_0 + \beta_1 X + \varepsilon$  reduces to

$$Y = \beta_0 + \varepsilon$$

and  $X$  is *not* associated with  $Y$ .

To test the null hypothesis, we need to determine whether  $\hat{\beta}_1$ , our estimate for  $\beta_1$ , is sufficiently far from zero that we can be confident that  $\beta_1$  is non-zero.

But *how* far is far enough? This of course depends on the accuracy of  $\hat{\beta}_1$  - that is, it depends on  $\text{se}(\hat{\beta}_1)$ . If  $\text{se}(\hat{\beta}_1)$  is small, then even relatively small values of  $\hat{\beta}_1$  may provide strong evidence that  $\beta_1 \neq 0$ , and hence that there is a relationship between  $X$  and  $Y$ .

In contrast, if  $\text{se}(\hat{\beta}_1)$  is large, then  $\hat{\beta}_1$  must be large in absolute value in order for us to reject the null hypothesis. In practice, we compute a *t-statistic* given by

$$T = \frac{\hat{\beta}_1 - 0}{\text{se}(\hat{\beta}_1)}$$

which measures the number of standard deviations that  $\hat{\beta}_1$  is away from 0. If there really is no relationship between  $X$  and  $Y$ , then we expect that  $T$  will have a *t*-distribution with  $n - 2$  degrees of freedom. The *t*-distribution has a bell shape and for values of  $n$  greater than approximately 30 it is quite similar to the normal distribution. Consequently, it is a simple matter to compute the probability of observing any value of  $T$  equal to  $|t|$  or larger, assuming  $\beta_1 = 0$ . We call this probability the *p-value*. Roughly speaking, we interpret the p-value as follows: a small p-value indicates that it is unlikely to observe such a substantial association between the predictor and the response due to chance, in the absence of any real association between the predictor and the response. Hence, if we see a small p-value, then we can infer that there is an association between the predictor and the response. We *reject the null hypothesis* - that is, we declare a relationship to exist between  $X$  and  $Y$  - if the p-value is small enough. Typical p-value cutoffs for rejecting the null hypothesis are 5 % or 1 %. These cutoffs are called *significance levels*.

#### Example 5.3.4

For the **Advertising** data, we want to compute the p-value of  $\beta_1$  of the least squares model for the regression of number of units sold on TV advertising budget.

The least squares estimate for  $\beta_1$  is 0.047537 and the standard error of  $\hat{\beta}_1$  is 0.002691. These values are provided by the **Python**-output already provided in 5.3.3: (to R)

## Chapter 5. Simple Linear Regression

[2] : `print(model.summary())`

```

    OLS Regression Results
=====
Dep. Variable:           sales   R-squared:      0.612
Model:                 OLS     Adj. R-squared:  0.610
Method:                Least Squares F-statistic:   312.1
Date:      Fri, 26 Feb 2021   Prob (F-statistic): 1.47e-42
Time:          11:12:03       Log-Likelihood:   -519.05
No. Observations:      200      AIC:             1042.
Df Residuals:          198      BIC:             1049.
Df Model:                   1
Covariance Type:        nonrobust
=====
            coef    std err      t      P>|t|      [0.025      0.975]
-----
const      7.0326    0.458    15.360    0.000      6.130      7.935
TV         0.0475    0.003    17.668    0.000      0.042      0.053
=====
Omnibus:            0.531   Durbin-Watson:   1.935
Prob(Omnibus):      0.767   Jarque-Bera (JB):  0.669
Skew:              -0.089   Prob(JB):       0.716
Kurtosis:           2.779   Cond. No.       338.
=====
```

Hence, we find for the realization of the statistic  $T$

$$t = \frac{\hat{\beta}_1 - 0}{\text{se}(\hat{\beta}_1)} = \frac{0.047537 - 0}{0.002691} = 17.66518$$

This value can as well be found in the R-output under `t_value`.

Furthermore, we find in the Python-output , that the number of degrees of freedom is 198, which is listed under `Df Residuals`. Since the number of degrees of freedom is given by  $n - 2$ , it follows that in total there are  $n = 200$  data points . This number corresponds to the number of markets in the data set **Advertising**.

Since the coefficient for  $\hat{\beta}_1$  is very large relative to its standard error, so the t-statistic is also large. The probability of seeing such a value if  $H_0$  is true is virtually zero. Hence we can conclude that  $\beta_1 \neq 0$ .

We now will compute the p-value, that is, the probability of observing a value of the t-statistic larger than  $|t| = 17.66518$ . Assuming  $\beta_1 = 0$ ,  $T$  will follow a t-distribution with  $n - 2 = 198$  degrees of freedom. Then the (two-sided) p-value can be determined with the help of Python as follows (to R)

[3] : `from scipy.stats import t`  
`p_two_sided = 2 * (1 - t.cdf(17.66518, 198))`  
`print(p_two_sided)`

0.0

Since the alternative hypothesis is two-sided, we need to multiply the one-sided p-value by two in order to obtain the two-sided p-value.

The corresponding p-value : 0.000 is listed under  $P(>|t|)$ . Since it is zero, hence smaller than a significance level of 0.05 , we *reject* the null hypothesis  $\beta_1 = 0$  in favor of the alternative hypothesis  $\beta_1 \neq 0$ . We therefore find that there clearly is a relationship between **TV** and **sales**. ◀

### 5.3.3. Confidence Intervals for Regression Coefficients

Standard errors can also be used to compute *confidence intervals* for the regression coefficients. A 95 % confidence interval is defined as a range of values such that with 95 % probability, the range will contain the true unknown value of the parameter. The range is defined in terms of lower and upper limits computed from the sample of data. For linear regression, the 95 % confidence interval for  $\beta_1$  approximately takes the form

$$\hat{\beta}_1 \pm 2 \cdot \text{se}(\hat{\beta}_1)$$

That is, there is approximately a 95 % chance that the interval

$$[\hat{\beta}_1 - 2 \cdot \text{se}(\hat{\beta}_1), \hat{\beta}_1 + 2 \cdot \text{se}(\hat{\beta}_1)]$$

will contain the true value of  $\beta_1$ . Similarly, a confidence interval for  $\beta_0$  approximately takes the form

$$\hat{\beta}_0 \pm 2 \cdot \text{se}(\hat{\beta}_0)$$

#### Example 5.3.5

In the case of the **Advertising** data, the 95 % confidence interval can be found with the help of the `conf_int`-function: (to R)

```
[4]: # Confidence interval found using conf_int method
model.conf_int(alpha=0.05)
```

	0	1
const	6.1297	7.9355
TV	0.0422	0.0528

The 95 %-confidence interval for  $\beta_0$  thus is

$$[6.130, 7.935]$$

and the 95 %-confidence interval for  $\beta_1$

$$[0.042, 0.053]$$

Therefore, we can conclude that in the absence of any advertising, sales will, on average, fall somewhere between 6130 and 7935 units. Furthermore, for each CHF 1000 increase in television advertising, there will be an average increase in sales of between 42 and 53 units. ◀

### Remarks:

- i. The exact formula for the 95 % confidence interval for the regression coefficient  $\beta_i$  is

$$\left[ \hat{\beta}_i - t_{0.975,n-2} \cdot \text{se}(\hat{\beta}_i), \hat{\beta}_i + t_{0.975,n-2} \cdot \text{se}(\hat{\beta}_i) \right]$$

where  $t_{0.975,n-2}$  is the 97.5 % quantile of a t-distribution with  $n - 2$  degrees of freedom.

- ii. With [Python](#) we determine the 97.5 % quantile of a t-distribution as follows ([to R](#))

```
[5] : # the 97.5% quantile of a t-distribution:
q_975 = t.ppf(0.975, 18)
print(q_975)
```

2.1009

which is approximately 2. ♦

## 5.4. Confidence and Prediction Intervals for the Response

### 5.4.1. Confidence Interval for the Response Variable

In Section 5.3.3 we have introduced confidence intervals for the regression coefficients  $\beta_0$  and  $\beta_1$ . In an analogous way, we can determine a confidence interval to quantify the uncertainty surrounding the *average* of the response variable.

**Example 5.4.1**

For example, CHF 100'000 is spent on **TV** advertising in each city. We may now ask, within which interval lies the true average of **sales** with a probability of 95 % ? In other words, if we collect a large number of data sets like the **Advertising** data set, and we construct a confidence interval for the average **sales** on the basis of each data set, given CHF 100'000 in **TV**, then 95 % of these confidence intervals will contain the true value of average **sales**. ◀

We are thus interested in the expected value of the predicted response variable  $\hat{Y}$ , given a value of the predictor  $X$ , that is

$$E[\hat{Y}|X]$$

If we fix the value of the predictor  $X = x_0$ , then the value of the predicted response  $\hat{y}_0$  is given by (assuming a linear relationship)

$$\hat{y}_0 = \hat{\beta}_0 + \hat{\beta}_1 x_0$$

Whenever we repeat the experiment, the value  $\hat{y}_0$  will change, since the corresponding regression coefficients  $\hat{\beta}_0$  and  $\hat{\beta}_1$  will change as well. The expected value of  $\hat{y}$  for a given value  $x_0$  of the predictor  $X$  then is

$$E[\hat{y}|x_0] = E[\hat{\beta}_0 + \hat{\beta}_1 x_0] = \beta_0 + \beta_1 x_0$$

which can be considered as the expected value of the predicted response  $\hat{y}_0$  for a given  $x_0$ , that is  $y_0 = f(x_0)$ . In other words, since the regression coefficients are random variables, the regression line is a random variable as well, and might have turned out to be different with another sample. We now want to determine a confidence interval for the population regression line.

How can we now construct a 95 % confidence interval for the expected value of  $\hat{y}_0$ , that is for  $y_0 = f(x_0)$ ? This is the interval which contains  $y_0 = f(x_0)$  with a probability of 95 %.

To determine this confidence interval we proceed in an analogous way as in the case of the confidence interval for  $\beta_0$  in Section 5.3.3. The standard error of  $\hat{y}_0$  is given by:

$$se(\hat{y}_0)^2 = \hat{\sigma}^2 \left( \frac{1}{n} + \frac{(x_0 - \bar{x})^2}{\sum_{i=1}^n (x_i - \bar{x})^2} \right)$$

where  $\hat{\sigma}$  denotes the standard deviation of the error term  $\varepsilon$  as estimated by the RSE.

The formula for the 95 % confidence interval for  $y_0$  is given approximately by

$$\hat{y}_0 \pm 2 \cdot se(\hat{y}_0)$$

## Chapter 5. Simple Linear Regression

That means that  $E[\hat{y}|x_0]$  is contained with a probability of 95 % in the interval

$$[\hat{y}_0 - 2 \cdot \text{se}(\hat{y}_0), \hat{y}_0 + 2 \cdot \text{se}(\hat{y}_0)]$$

The formula itself is of relatively little importance for the practitioner, because that functionality is pre-existing in [Python](#).

### Example 5.4.2

For the [Advertising](#) data set we determine the 95 % confidence intervals for the following values of **TV**: 3, 100 and 275. ([to R](#))

```
[1]: import numpy as np
import pandas as pd
import statsmodels.api as sm

# Load data
df = pd.read_csv('./data/Advertising.csv')
x = df['TV']
y = df['sales']

# Fit Linear Model
x = sm.add_constant(x)
model = sm.OLS(y, x).fit()

# Prediction at points 3, 100 and 270
x0 = [3, 100, 275]
x0 = sm.add_constant(x0)

predictionsx0 = model.get_prediction(x0)
predictionsx0 = predictionsx0.summary_frame(alpha=0.05)
predictionsx0 = np.round(predictionsx0, 4)
print(predictionsx0)
```

	mean	mean_se	mean_ci_lower	mean_ci_upper	obs_ci_lower	obs_ci_upper
0	7.1752	0.4509	6.2860	8.0644	0.6879	13.6626
1	11.7863	0.2629	11.2678	12.3047	5.3393	18.2333
2	20.1052	0.4143	19.2882	20.9221	13.6273	26.5830

In the [Python](#)-output , the values

$$\hat{y}_0 = \hat{\beta}_0 + \hat{\beta}_1 x_0$$

can be found under `mean` ; they correspond to the  $y$ -values on the regression line, thus to the predicted response given a predictor value.

Under `mean ci lower` the lower limits, under `mean ci upper` the upper limits of the corresponding confidence intervals can be found.

## Chapter 5. Simple Linear Regression

For the value  $x_0 = 100$  the 95 % confidence interval is given by

$$[11.268, 12.305]$$

The expected value of  $\hat{y}$  given the predictor value  $x_0 = 100$  is contained in this interval with a probability of 95 %.

We can visualize confidence intervals for the expected response variable given an intervall of predictor values. In Figure 5.6 the regression line is plotted in blue. The green curves correspond to the lower and upper limits of the confidence intervals given a  $x_0$ . The red lines represent the 95 % confidence intervals for the values  $x_0 = 3, 100$  and  $275$ . These intervals contain with a probability of 95 % the corresponding expected values of  $\hat{y}_0$ . (to R)

```
[2]: import matplotlib.pyplot as plt
import numpy as np

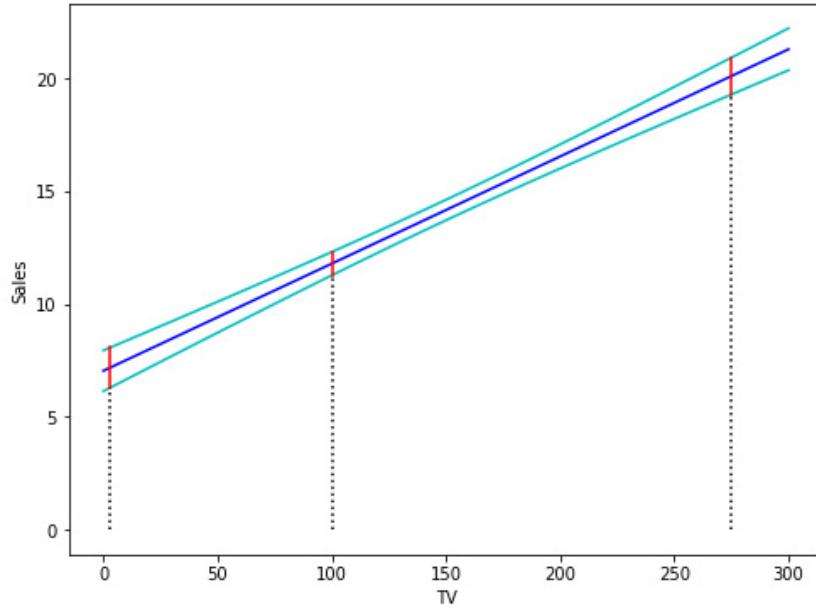
x = np.linspace(0, 300)
x_sm = sm.add_constant(x)

# Predictions
predictionsx = model.get_prediction(x_sm)
# at 95% confidence:
predictionsx = predictionsx.summary_frame(alpha=0.05)

# Create figure and plot
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(1, 1, 1)
# prediction
plt.plot(x, predictionsx.loc[:, 'mean'], 'b-')
# upper and lower boundaries at 95%
plt.plot(x, predictionsx.loc[:, 'mean_ci_lower'], 'c-')
plt.plot(x, predictionsx.loc[:, 'mean_ci_upper'], 'c-')
# lines of the three points in x0:
for i in range(len(x0)):
    plt.plot([x0[i, 1], x0[i, 1]],
             [0, predictionsx0.loc[:, 'mean_ci_lower'][i]], 'k:')
    plt.plot([x0[i, 1], x0[i, 1]],
             [predictionsx0.loc[:, 'mean_ci_lower'][i],
              predictionsx0.loc[:, 'mean_ci_upper'][i]], 'r-')

# Set labels and Legend
ax.set_xlabel('TV')
ax.set_ylabel('Sales')

plt.show()
```



**Figure 5.6.:** Confidence intervals for the expected values of  $\hat{y}_0$ . The region enclosed by the green curves is called *confidence band*.

#### Remarks:

- i. Analogous to Section 5.3.3 the precise 95 % confidence interval for the expected value of  $\hat{y}_0$  is

$$[\hat{y}_0 - t_{0.975;n-2} \cdot \text{se}(\hat{y}_0), \hat{y}_0 + t_{0.975;n-2} \cdot \text{se}(\hat{y}_0)]$$

where  $t_{0.975;n-2}$  denotes the 0.975-quantile of a t-distribution with  $n - 2$  degrees of freedom. ♦

### 5.4.2. Prediction Intervals

An important task in statistics consists of predicting *future* observations  $Y$  for a given value of  $X$ . We are now not interested in the variability of the fitted value  $\hat{Y}$ , but in the variability of the future observation itself. We will characterize this variability in terms of a *prediction interval*.

#### Example 5.4.3

We now want to quantify the uncertainty surrounding **sales** for a *particular* city. Given that CHF 100'000 is spent on **TV** advertising, which interval contains with a probability of 95 % the true value of **sales** for this particular city? We interpret this to mean that 95 % of intervals of this form will contain the true value of  $Y$  for this city.

The prediction interval will be certainly wider than the confidence interval reflecting the increased uncertainty about **sales** for a given city in comparison to the average **sales** over many locations. 

To predict future observations on the basis of observed data is an essential task in many fields, such as engineering.

We may be tempted to use the confidence interval of the previous section to indicate the interval that contains a future observation for a given  $x_0$ . However, this interval only contains the expected value of  $\hat{y}_0 = \hat{\beta}_0 + \hat{\beta}_1 x_0$  with a given probability. We now have to account as well for the scatter of the data points around the regression line which is expressed by the error term  $\varepsilon$  in our regression model. This random scatter around the regression line increases the uncertainty in predicting a future observation. Therefore, we are led to a new interval, which we call *prediction interval*. It is obvious, that the prediction interval is wider than the confidence interval for the expected value of  $\hat{y}_0 = \hat{\beta}_0 + \hat{\beta}_1 x_0$ . The standard error of  $y_0$  is given by

$$se(y_0)^2 = \hat{\sigma}^2 \left( 1 + \frac{1}{n} + \frac{(x_0 - \bar{x})^2}{\sum_{i=1}^n (x_i - \bar{x})^2} \right)$$

where  $\hat{\sigma}$  denotes the standard deviation of the error term  $\varepsilon$ .

To determine the prediction interval we may use again the `Python`-function `predict()`:

### Example 5.4.4

In the case of the **Advertising** data we determine the 95 % prediction intervals for the  $x$ -values 3, 100 and 225 from the same results as provided in 5.4.2: [\(to R\)](#)

[3] : `print(predictionsx0)`

	mean	mean_se	mean_ci_lower	mean_ci_upper	obs_ci_lower	obs_ci_upper
0	7.1752	0.4509	6.2860	8.0644	0.6879	13.6626
1	11.7863	0.2629	11.2678	12.3047	5.3393	18.2333
2	20.1052	0.4143	19.2882	20.9221	13.6273	26.5830

Under `mean` the predicted  $y$ -values on the regression line

$$\hat{y}_0 = \hat{\beta}_0 + \hat{\beta}_1 x_0$$

can be found.

`obs_ci_lower` displays the lower limits and `obs_ci_upper` the upper limits of the prediction intervals for the given  $x$ -values.

## Chapter 5. Simple Linear Regression

For the predictor value  $x_0 = 100$  the 95 % prediction interval is given by

$$[5.339, 18.233]$$

A future observation  $y_0$  for given  $x_0 = 100$  will fall with a probability of 95 % into this interval. As we can observe, the prediction interval thus is clearly larger than the confidence interval for the expected value of  $\hat{y}_0 = \hat{\beta}_0 + \hat{\beta}_1 x_0$ .

It is again very instructive to visualize the point-wise prediction intervals. Figure 5.7 displays the regression line in blue. The green curves represent the upper and lower limits of the prediction intervals for future observations. The red lines correspond to 95 % prediction intervals for  $x_0 = 3, 100, 275$ . These intervals contain with a probability of 95 % the true values of the corresponding future observations  $y_0$ .

(to R)

```
[4] : x = np.linspace(0, 300)
x_sm = sm.add_constant(x)

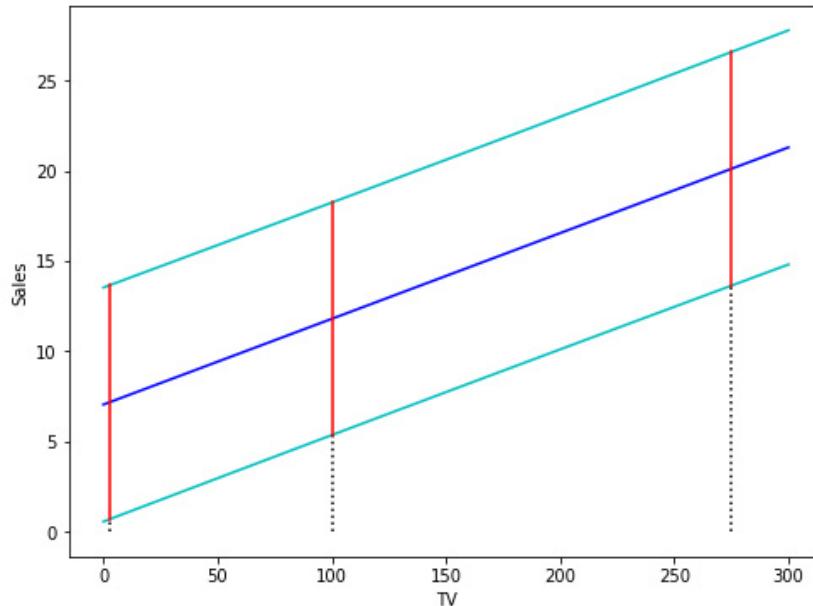
# Predictions
predictionsx = model.get_prediction(x_sm)
# at 95% confidence:
predictionsx = predictionsx.summary_frame(alpha=0.05)

# Create figure and plot
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(1, 1, 1)
# prediction
plt.plot(x, predictionsx.loc[:, 'mean'], 'b-')
# upper and lower boundaries at 95%
plt.plot(x, predictionsx.loc[:, 'obs_ci_lower'], 'c-')
plt.plot(x, predictionsx.loc[:, 'obs_ci_upper'], 'c-')
# lines of the three points in x0:
for i in range(len(x0)):
    plt.plot([x0[i, 1], x0[i, 1]],
             [0, predictionsx0.loc[:, 'obs_ci_lower'][i]], 'k:')
    plt.plot([x0[i, 1], x0[i, 1]],
             [predictionsx0.loc[:, 'obs_ci_lower'][i],
              predictionsx0.loc[:, 'obs_ci_upper'][i]], 'r-')

# Set labels and Legend
ax.set_xlabel('TV')
ax.set_ylabel('Sales')

plt.show()
```





**Figure 5.7.:** Pointwise prediction intervals for future observations are displayed as green curves along with the blue regression line. The region enclosed by the green curves is called *prediction band*.

## Conceptual Objectives

You should be able...

- to explain the concept of a simple linear regression model and the geometric interpretation of its coefficients.
- to derive the least squares coefficient estimates for simple linear regression.
- to explain the (geometric) meaning of the standard errors associated with the regression coefficients.
- to explain how to estimate the variance of the error term by means of the residual standard error.
- to carry out a hypothesis test for the regression coefficients.
- to determine the confidence interval for the regression coefficients.
- to explain the difference between a confidence and a prediction interval for the response.

## Computational Objectives

You should be able...

- to determine the least squares coefficient estimates by means of `statsmodels.api.OLS().fit()`
- to determine the standard errors of the estimated coefficients by means of `sm.OLS().fit().summary()`
- to read off the test decision of the hypothesis test for the regression coefficients from the `Python`-output of `.summary()`
- to determine the confidence- and prediction intervals for the regression coefficients by means of the function `sm.OLS().fit().get_prediction()`

# Chapter 6.

## Testing Model Assumptions: Residual Analysis<sup>1</sup>

### 6.1. Assumptions Concerning the Error Term $\varepsilon$

In this section, we will focus on the error term  $\varepsilon$ . The distribution of the regression coefficients depends on the distribution of the error term  $\varepsilon$ . The distribution of the error term  $\varepsilon$  hence determines to which level our model predictions are correct. In this section, we assume that

$$Y = f(X) + \varepsilon$$

where the function  $f(X)$  is assumed to be known. In practice, this is never the case. To focus on the assumptions concerning the error term  $\varepsilon$ , we will consider rather simple functions  $f(x)$ .

#### What is the meaning of $\varepsilon$ ?

If we measure  $Y$  at position  $x_i$ , then the following equation

$$y_i = f(x_i) + \varepsilon_i$$

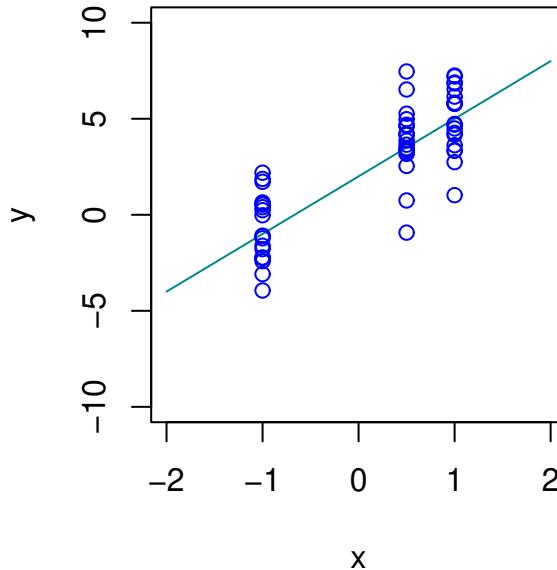
holds, where  $\varepsilon_i$  represents the error of the function  $f$  at position  $x_i$ , or to be more concise: it represents the deviation of the measured value  $y_i$  from the predicted value of  $f(x_i)$ . If we repeat the measurement several times at the same position  $x_i$ , we will observe for every measurement a different value of  $y_i$  due to the error term  $\varepsilon_i$ .

#### Example 6.1.1

Figure 6.1 displays the (true) function  $f(X) = 2 + 3X$  which is plotted as a green line. At positions  $x_1 = -1$ ,  $x_2 = 0.5$  and  $x_3 = 1$  the corresponding values  $y_i$  were simulated.

---

<sup>1</sup>This chapter follows Chapter 3.3.3 of the text book *An Introduction to Statistical Learning: with Applications in R* by G. James et al., Springer Texts in Statistics 2013.



**Figure 6.1.:** Simulation of  $y_i = 2 + 3x_i + \varepsilon_i$  at positions  $x_1 = -1$ ,  $x_2 = 0.5$  and  $x_3 = 1$  for  $i = 1, 2, \dots, 20$ .

◀

Now, we may wonder which assumptions concerning the error term  $\varepsilon$  our regression model should be based on.

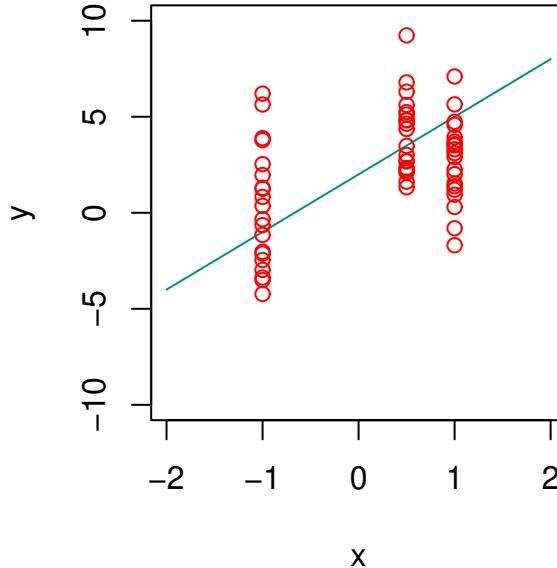
### Expected value 0

It is reasonable to assume that the errors should cancel on average. Figure 6.1 displays data points that scatter around a regression line; on average, the (vertical) distances between the regression line and points lying below the line seem to be equal to the distances between the regression line and the points lying above the line. The negative and positive values of  $\varepsilon_i$  hence sum up to zero. Mathematically, this assumption can be expressed as the expected value of the errors having zero mean:

$$E[\varepsilon_i] = 0 \quad \text{for all } i$$

The underlying idea is that the errors do not exhibit any preference for lying above or below the regression line. They are supposed to be “neutral”.

Figure 6.2 displays a case where the expected values of  $\varepsilon_i$  are different from zero. At positions  $x_1 = -1$  and  $x_2 = 0.5$  the expected value tends to be larger than 0, as the majority of data points lie above the regression line. The errors hence tend to be rather positive. Contrary to position  $x_3 = 1$  where the expected value is smaller than 0 and the errors tend to be rather negative.



**Figure 6.2.:** At positions  $x_1 = -1$ ,  $x_2 = 0.5$  and  $x_3 = 1$ , the expected values of  $\varepsilon_i$  are different from 0.

### Constant Variance of Error Terms

Another reasonable assumption of the linear regression model is that the error terms  $\varepsilon_i$  all have the same constant variance:

$$\text{Var}[\varepsilon_i] = \sigma^2 \quad \text{for all } i$$

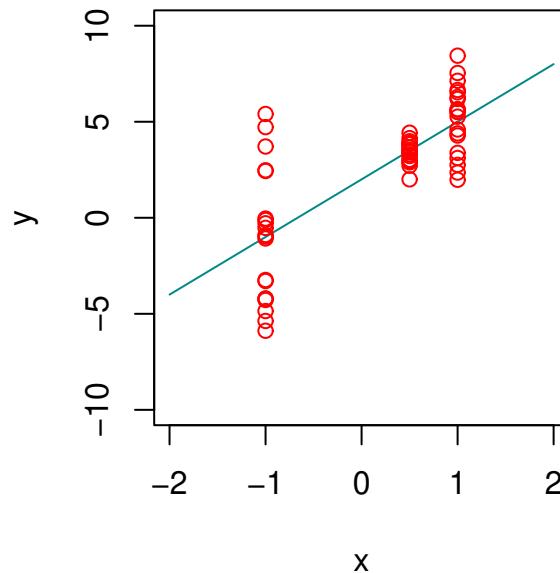
In Figure 6.1 we observe that all points scatter with the same magnitude around the regression line. The underlying idea is that the errors represent “external” influences which are independent of the position where the measurements are carried out.

Figure 6.3 displays simulated points that scatter in an uneven way around the regression line. At position  $x_1 = -1$  points scatter around the line with a large magnitude whereas at position  $x_2 = 0.5$  the scattering magnitude is much smaller.

### Errors Follow a Normal Distribution

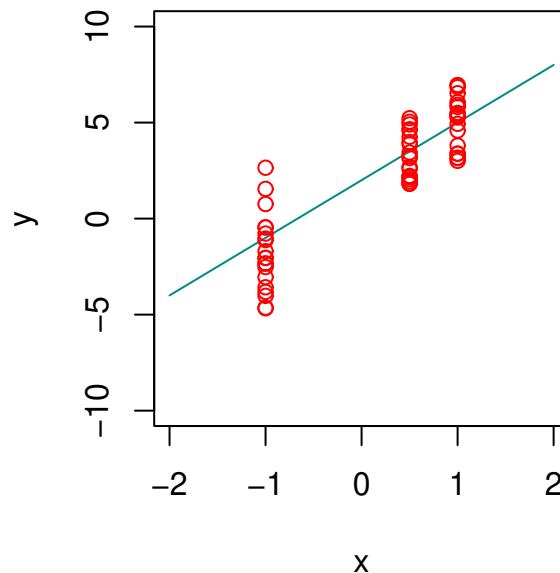
All errors  $\varepsilon_i$  should follow the same probability distribution. On the one hand, we know from experience that measurement errors are best fit by a normal distribution, on the other hand it follows from the Central Limit Theorem that the sum of randomly distributed random influences such as unobserved variables and measurement errors follows a normal distribution. We therefore assume that

$$\varepsilon_i \sim \mathcal{N}(0, \sigma^2) \quad \text{for all } i$$



**Figure 6.3.:** The variance of  $\varepsilon_i$  is different at positions  $x_1 = -1$ ,  $x_2 = 0.5$  and  $x_3 = 1$  and thus non-constant.

Figure 6.4 displays different distributions for the error terms  $\varepsilon_i$ : at position  $x_1 = -1$  a normal distribution with  $\sigma = 2$  was simulated and at positions  $x_2 = 0.5$  and  $x_3 = 1$  a uniform distribution on the interval  $[-2, 2]$  was simulated.



**Figure 6.4.:** Different distributions for the error terms  $\varepsilon_i$  at positions  $x_1 = -1$ ,  $x_2 = 0.5$  and  $x_3 = 1$  were simulated.

### Independence of the Error Terms $\varepsilon_i$

Another important and reasonable assumption of the linear regression model is that the error terms,  $\varepsilon_1, \varepsilon_2, \dots, \varepsilon_n$  are independent. What does this mean? For instance, if the errors are independent, then the fact that  $\varepsilon_i$  is positive provides little or no information about the sign of  $\varepsilon_{i+1}$ . In other words, a measurement at position  $x_i$  does not influence the measurement at any other position  $x_j$ .

### Summary

#### Model Assumptions for the Error Terms $\varepsilon_i$

The error terms  $\varepsilon_i$  are independent and normally distributed random variables with a constant variance:

$$\varepsilon_i \quad \text{i.i.d.} \quad \mathcal{N}(0, \sigma^2)$$

More precisely:

1. For the expected value of all  $\varepsilon_i$  we have

$$E[\varepsilon_i] = 0$$

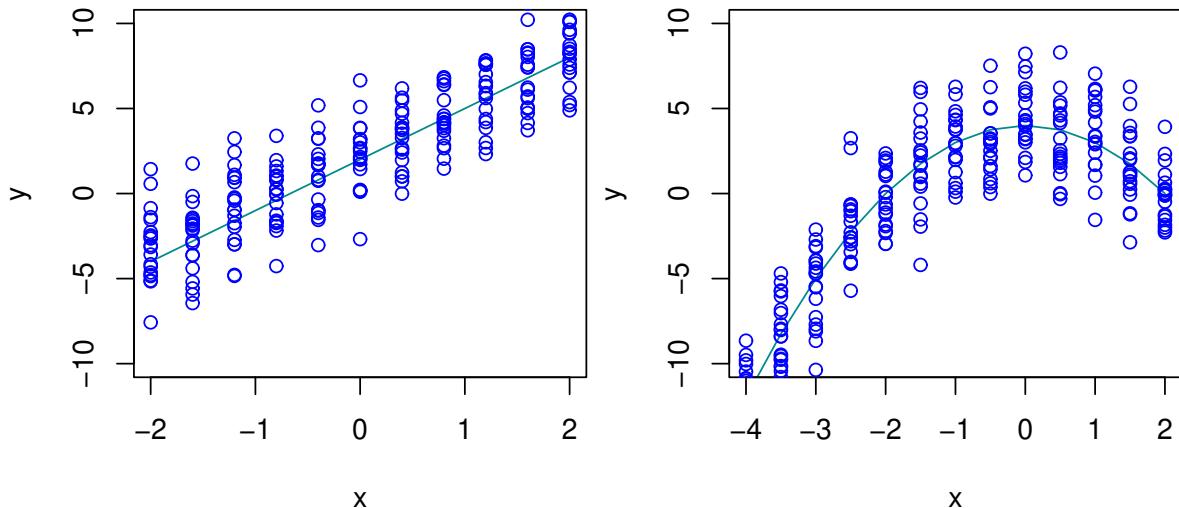
2. The error terms  $\varepsilon_i$  all have the same constant variance

$$\text{Var}[\varepsilon_i] = \sigma^2$$

3. The error terms  $\varepsilon_i$  are normally distributed
4. The error terms  $\varepsilon_i$  are independent

Figure 6.5 displays possible errors along two different function curves, if the model assumptions concerning the error terms are all satisfied. In the left-hand panel, points scatter around the regression line we have seen in the previous example; in the right-hand panel, points scatter around a parabolic curve.

These model assumptions need to be satisfied in order to obtain *correct* model predictions such as the estimation of regression coefficients, confidence intervals and prediction intervals as discussed in chapter 5.



**Figure 6.5:** Data points scatter according to a normal distribution around a regression line (on the left-hand panel) and around a parabolic curve (on the right-hand panel).

### 6.1.1. Relationship between Response Variable and Predictor is Non-Linear

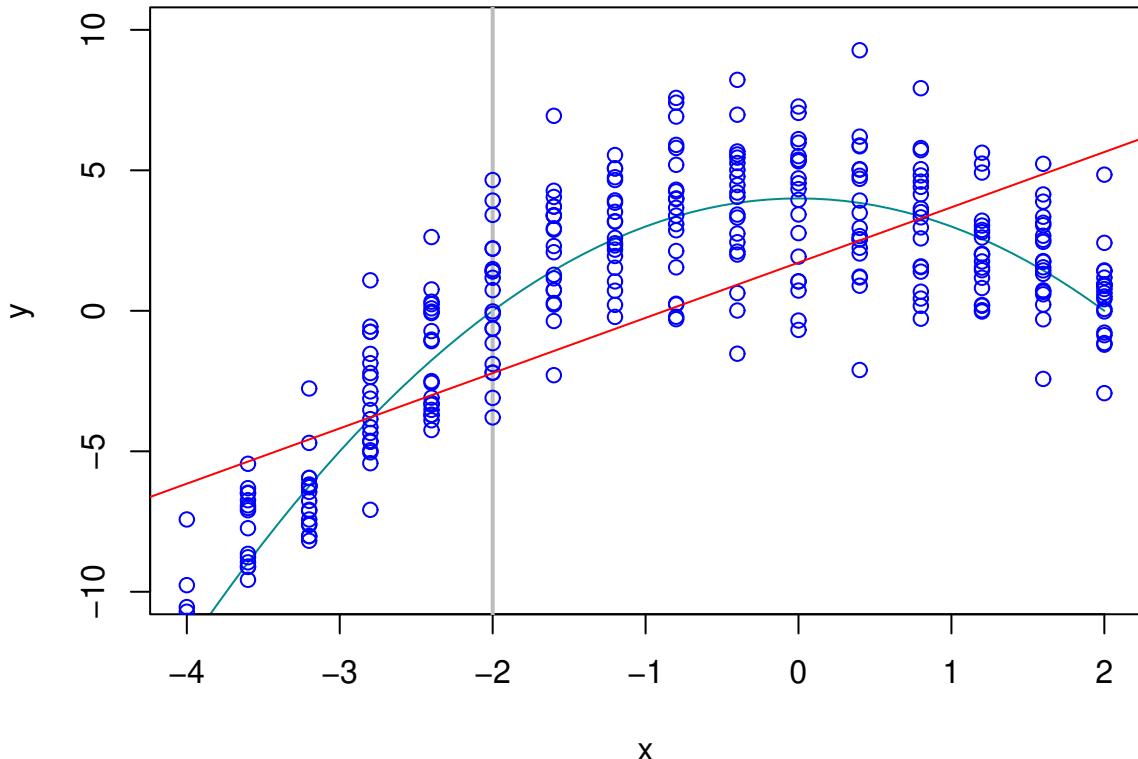
If we assume a *linear* regression function  $f$  which in reality however is not linear, then the relation  $E[\varepsilon_i] = 0$  does *not* hold.

#### Example 6.1.2

We can illustrate this situation with a simulation that is displayed in Figure 6.6. The true regression function  $f$  is here a parabolic curve (shown in green). The blue points represent the simulated measurements which scatter according to a normal distribution around the parabolic curve. The red line represents a regression line that was constructed on the basis of the simulated (blue) points by assuming a linear regression function  $f$ .

Let us now assume that we do not know anything about the true underlying regression function  $f$ ; we just observe the blue points which represent our measurements. The vertical displacements between our measurements and the regression line correspond to the residuals that are used as well for the estimation of the error terms  $\varepsilon_i$ .

We now focus on the measurements  $y_i$  corresponding to a given measurement  $x_i$ . In Figure 6.6, all measurements  $y_i$  for  $x = -2$  are displayed along the grey line which is perpendicular to the  $x$ -axis. The average of the displacements between the measurements and the regression line is the estimated expected value of the error terms  $\varepsilon_i$  at  $x = -2$ . The average of the error terms at this position is obviously different from 0.



**Figure 6.6.:** The model assumption  $E[\varepsilon_i] = 0$  is violated in the case of a non-linear regression function  $f$  shown in green.

Only at the two intersection points of the regression line with the parabolic curve the average of the error terms  $\varepsilon_i$  will be approximately zero.



### 6.1.2. Assessing the Validity of the Model Assumptions

In this section we will verify every assumption underlying the linear regression model. In the previous section we motivated why the model assumptions may be considered as “reasonable”. We now turn our attention to the question to which extent the assumptions are fulfilled for a concrete regression model that was constructed based on a data set. If however one or several model assumptions are violated, we should see this as a chance or starting point to adapt and/or extend our regression model to find a better and more adapted model (*explorative data analysis*).

In order to test whether the model assumptions with respect to the error terms  $\varepsilon_i$  are fulfilled, the error terms should in principle be known. Since  $f(X)$  is in general unknown, we are not able to determine the errors directly and hence we are not in

the position to state directly what the errors are and how they are distributed. We do not know the error terms, but we can determine without any problem the *residuals*

$$r_i = y_i - \hat{y}_i$$

On the basis of the residuals we can estimate the error terms  $\varepsilon_i$ , or more precisely, the standard deviation of the error terms  $\varepsilon_i$ .

The methods to test the assumptions of the linear regression model are called *residual analysis*. In the previous example we simulated the errors on the basis of a known regression function; in this case the deviations from the assumptions of the linear regression model were obvious. In practice however, deviations from these model assumptions are not that obvious. Residual analysis then represents a powerful tool to reveal the distribution of the error terms in more complex situations and to detect any deviation from the assumptions of the linear regression model.

## 6.2. Diagnostics Instruments

### 6.2.1. The Residual Standard Error and the $R^2$ Statistic

Once we have established that there is a linear relationship between a predictor and a response variable (by rejecting the null hypothesis  $H_0 : \beta_1 = 0$ ), it is natural to quantify *the extent to which the model fits the data* next.

The quality of a linear regression fit is typically assessed using two related quantities: the *residual standard error* (RSE) and the  $R^2$  statistic.

#### The Residual Standard Error

Let  $y_i$  denote the measured value of the response variable  $Y$  corresponding to the measured value  $x_i$  of the predictor variable  $X$  and let  $\hat{y}_i$  be the predicted value on the regression line. We recall, the residual  $r_i$  is defined as

$$r_i = y_i - \hat{y}_i$$

We already have discussed the

$$\text{RSE} = \sqrt{\frac{r_1^2 + \dots + r_n^2}{n - 2}}$$

in Section 5.3.1. The RSE is an estimate of the standard deviation of  $\varepsilon$ . Roughly speaking, it is the average amount that the response will deviate from the true regression line.

### Example 6.2.1

In the case of the **Advertising** data, the Residual standard error can be found using `OLSResults.mse_resid`, which returns **RSE**<sup>2</sup>.

$$\text{RSE} = 3.26$$

In other words, actual **sales** in each among the 200 markets deviate from the true regression line by approximately 3260 units, on average. Another way to think about this is that even if the model were correct and the true values of the unknown coefficients  $\beta_0$  and  $\beta_1$  were known exactly, any prediction of **sales** on the basis of **TV** advertising would still be off by about 3260 units on average. Of course, whether or not 3260 units is an acceptable prediction error depends on the problem context. In the advertising data set, the mean value of **sales** over all markets is approximately 14 000 units, and so the percentage error is

$$\frac{3260}{14000} \approx 0.23 = 23\%$$



The RSE is considered a measure of the *lack of fit* of the regression model to the data. If the predictions  $\hat{y}_i$  obtained using the model are very close to the true outcome values  $y_i$  - that is  $\hat{y}_i \approx y_i$  for  $i = 1, \dots, n$  - then the RSE will be small, and we can conclude that the model fits the data very well.

On the other hand, if the prediction  $\hat{y}_i$  is very far from  $y_i$  for one or more observations, then the RSE may be quite large, indicating that the model doesn't fit the data well.

### **R**<sup>2</sup> Statistic

The RSE provides an absolute measure of lack of fit of the linear regression model

$$Y = \beta_0 + \beta_1 X + \varepsilon$$

to the data. But since it is measured in the units of  $Y$ , it is not always clear what constitutes a good RSE. The  $R^2$  statistic provides an alternative measure of fit. It takes the form of a *proportion* - the proportion of variance explained - and so it always takes on a value between 0 and 1, and is independent of the scale of  $Y$ .

To calculate  $R^2$ , we use the formula

$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2} = 1 - \frac{\text{RSS}}{\text{TSS}}$$

where  $TSS = \sum_{i=1}^n (y_i - \bar{y})^2$  is the *total sum of squares* and  $RSS = \sum_{i=1}^n (y_i - \hat{y}_i)^2$  is the *residual sum of squares*. TSS measures the total variance in the response  $Y$ , and can be thought of as the amount of variability *inherent* in the response before the regression is performed. In contrast, RSS measures the amount of variability that is left *unexplained* after performing the regression.

If the model fits perfectly the data, then we have  $\hat{y}_i = y_i$  for  $i = 1, \dots, n$ . In this case, the RSS becomes 0, and hence  $R^2 = 1$ . As a consequence, a  $R^2$ -value of approximately 1 means that a large part of the variance in the data is *explained* by the model. Contrary, an  $R^2$  value near 0 indicates that *little* of the variance in the data is explained by the model. This might occur because the linear model is wrong, or the inherent error  $\sigma^2$  is high, or both.

### Example 6.2.2

For the **Advertising** data set, the [Python](#)-output for the linear regression model discussed in [5.3.3 on page 115](#) displays under [R-squared](#) that

$$R^2 = 0.61$$

We therefore conclude that under two-thirds of the variability in **sales** is explained by a linear regression on **TV**. 

### Example 6.2.3

We find the  $R^2$ -value for the example [6.1.2](#).

(to [R](#))

```
[1]: import numpy as np
import statsmodels.api as sm

# Set random seed
np.random.seed(0)
# Create Random data allong y = x^2 + 4
x = np.arange(-4, 2.4, .4)
y = -(x * x) + 4 + np.random.normal(0, 2, len(x))

# Fit Linear Model
x_sm = sm.add_constant(x)
model = sm.OLS(y, x_sm).fit()

# Print Model Summary
print(model.summary())
```

## Chapter 6. Testing Model Assumptions: Residual Analysis

```

    OLS Regression Results
=====
Dep. Variable:                      y      R-squared:                 0.516
Model:                            OLS      Adj. R-squared:            0.481
Method:                           Least Squares      F-statistic:                14.91
Date: Mon, 01 Mar 2021      Prob (F-statistic):        0.00173
Time: 07:51:50                  Log-Likelihood:           -40.363
No. Observations:                   16      AIC:                     84.73
Df Residuals:                      14      BIC:                     86.27
Df Model:                           1
Covariance Type:                nonrobust
=====
      coef    std err          t      P>|t|      [0.025      0.975]
-----
const    2.6174     0.917      2.855      0.013      0.651      4.584
x1       1.6876     0.437      3.861      0.002      0.750      2.625
=====
Omnibus:                   3.331      Durbin-Watson:            0.690
Prob(Omnibus):              0.189      Jarque-Bera (JB):         1.592
Skew:                      -0.447      Prob(JB):                  0.451
Kurtosis:                   1.740      Cond. No.                  2.53
=====
```

The  $R^2$  value is 0.516.



The  $R^2$  statistic has an interpretational advantage over the RSE, since unlike the RSE, it always lies between 0 and 1. However, it can still be challenging to determine what is a *good*  $R^2$  value, and in general, this will depend on the application. For instance, in certain problems in physics, we may know that the data truly comes from a linear model with a small residual error. In this case, we would expect to see an  $R^2$  value that is extremely close to 1, and a substantially smaller  $R^2$  value might indicate a serious problem with the experiment in which the data were generated. On the other hand, in typical applications in biology, psychology, marketing, and other domains, the linear model

$$Y = \beta_0 + \beta_1 X + \varepsilon$$

is at best an extremely rough approximation to the data, and residual errors due to other unmeasured factors are often very large. In this setting, we would expect only a very small proportion of the variance in the response to be explained by the predictor, and an  $R^2$  value well below 0.1 might be more realistic.

The  $R^2$  statistic is a measure of the linear relationship between  $X$  and  $Y$ . Recall that correlation<sup>2</sup> (see section 3.3) defined as

$$\text{Cor}[X, Y] = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}}$$

---

<sup>2</sup>We note that in fact, the right-hand side is the sample correlation; thus, it would be more correct to write  $\widehat{\text{Cor}}[X, Y]$ ; however, we omit the “hat” for ease of notation.

is also a measure of the linear relationship between  $X$  and  $Y$ . This suggests that we might be able to use  $r = \text{Cor}(X, Y)$  instead of  $R^2$  in order to assess the fit of the linear model. In fact, it can be shown that in the simple linear regression setting,  $r^2 = R^2$ . In other words, the squared correlation and the  $R^2$  statistic are identical. However, in the next chapter we will discuss the multiple linear regression problem, in which we use several predictors simultaneously to predict the response. The concept of correlation between the predictors and the response does not extend automatically to this setting, since correlation quantifies the association between a single pair of variables rather than between a large number of variables. We will see that  $R^2$  fills this role.

## 6.2.2. Diagnostics Tool for Testing Model Assumption $E[\varepsilon] = 0$

### Tukey-Anscombe-Plots

The linear model assumes that there is a straight-line relationship between the predictor and the response. If the true relationship is far from linear, then virtually all of the conclusion that we draw from the fit are suspect. In addition, the prediction accuracy of the model can be significantly reduced.

*Residual plots* are a useful graphical tool for identifying non-linearity of the regression function  $f$ . Given a simple linear regression model, we can plot the residuals  $r_i = y_i - \hat{y}_i$  versus the fitted or predicted values  $\hat{y}_i$ . We thus plot the points  $(\hat{y}_i, r_i)$  for  $i = 1, \dots, n$ . The resulting plot is named *Tukey-Anscombe-Plot* after its inventors.

### Example 6.2.4

On the left-hand panel of Figure 6.7 the scatter plot for the **Advertising** data set is displayed. The right-hand panel of Figure 6.7 displays the corresponding Tukey-Anscombe plot. (to R)

```
[1]: import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt
import seaborn as sns

# Load data
df = pd.read_csv('./data/Advertising.csv')
x = df['TV']
y = df['sales']

""" Find Predictions and Residuals """
# Fit Linear Model
x_sm = sm.add_constant(x)
model = sm.OLS(y, x_sm).fit()
# Find the predicted values for the original design.
yfit = model.fittedvalues
```

## Chapter 6. Testing Model Assumptions: Residual Analysis

```

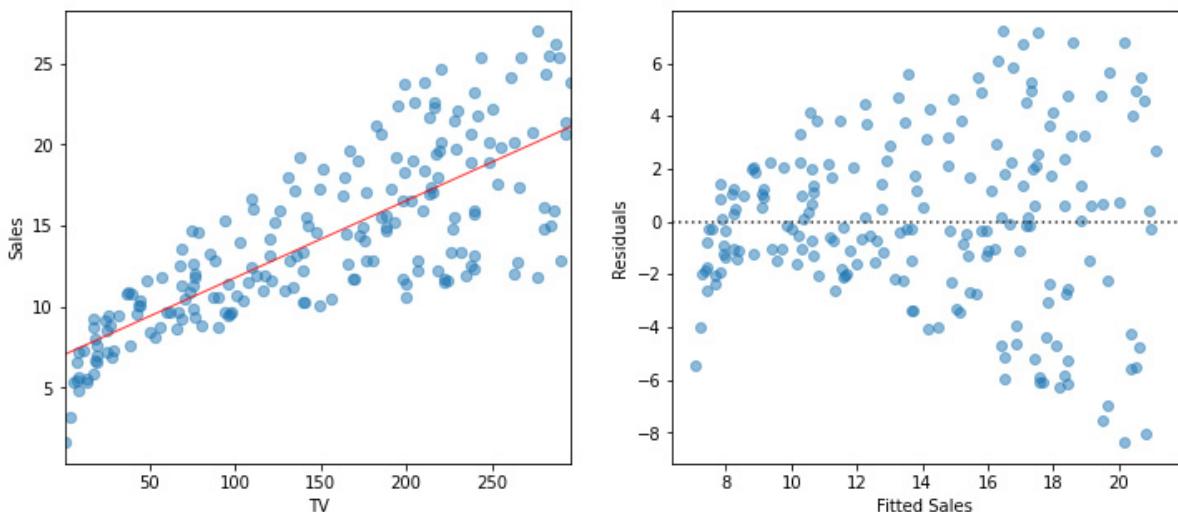
# Predict values based on the fit.
ypred = model.predict(x_sm)
# Residuals of the model
res = model.resid

""" Plots """
# Create figure and subplots
fig = plt.figure(figsize=(12, 5))
ax1 = fig.add_subplot(121)
# Plot data and regression
sns.regplot(x=x, y=y,
            scatter=True, ci=False, lowess=False,
            scatter_kws={'s': 40, 'alpha': 0.5},
            line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8})
# Set Labels
ax1.set_xlabel('TV')
ax1.set_ylabel('Sales')

# Plot Residuals
ax2 = fig.add_subplot(122)
df = pd.concat([yfit, res], axis=1)
sns.residplot(x=yfit, y=res, data=df,
               lowess=False, scatter_kws={'alpha': 0.5})
ax2.set_xlabel('Fitted Sales')
ax2.set_ylabel('Residuals')

# Show plots
plt.show()

```



**Figure 6.7.:** Left : Scatterplot for the `Advertising` data set. Right: Plots of residuals versus predicted (or fitted) values of `sales` for the `Advertising` data set.

We observe that the regression line on the left-hand panel corresponds to the horizontal axis on the right-hand panel of Figure 6.7. We now can easily read off the values

of the residuals for every predicted value  $\hat{y}_i$ .



The linear model fits the data well if the points in the Tukey-Anscombe plot scatter evenly around the  $r = 0$  line. “Evenly distributed” means, that within a small area of the  $r = 0$  line there are approximately as many points above the line as there are points below the line. These points should ideally have the same distance from the  $r = 0$  line. In other words, in such an area for the average of the residuals

$$\bar{r} \approx 0$$

should hold. Since we are estimating the error terms by means of the residuals, this relation finally is equivalent to the model assumption  $E[\varepsilon_i] = 0$ .

How can we decide on the basis of the Tukey-Anscombe plot whether the model assumption  $E[\varepsilon_i] = 0$  is fulfilled? To visualize the relation between the residuals  $r_i$  and the predicted response values  $\hat{y}_i$ , we will take advantage of the *smoothing approach*.

### Example 6.2.5

In the following, we will roughly illustrate the principle of the smoothing approach. A simple yet intuitive smoother is the *running mean*. It involves taking a fixed window width on the  $\hat{y}$ -axis, and compute the mean of the  $r$ -values of all the observations that fall into this window. This value then is the estimate for the function value at the window center. The left-hand panel of Figure 6.8 displays two thin stripes : the average of all residuals falling into such a stripe is calculated and is plotted as a red point at the center of the corresponding stripe. If the stripes are sliding along the  $\hat{y}$ -axis, connecting the red points for every window position then results in the red curve displayed in the right-hand panel of Figure 6.8. (to R)

```
[2]: # This example builds on onto Example 2.4
import numpy as np

""" Plots """
# Create Figure and subplots
fig = plt.figure(figsize=(12, 5))
ax1 = fig.add_subplot(121)

# Visualize Moving average:
# Plot Residuals
sns.residplot(x=yfit, y=res, data=df,
               lowess=False, scatter_kws={'alpha': 0.5})
# Plot vertical lines and average to visualize moving average
bandwidth = 0.5

for x_avg in [9.5, 17.5]:
    # Find indices where ypred is within range
    index_res = (np.where(ypred > (x_avg - bandwidth)) and
```

## Chapter 6. Testing Model Assumptions: Residual Analysis

```

        np.where(ypred < (x_avg + bandwidth)))
# Find mean of Residuals
res_mean = res[index_res[0]].mean()
# Plot lines
plt.plot((x_avg-bandwidth, x_avg - bandwidth),
          [-10, 10], color='darkgrey')
plt.plot((x_avg+bandwidth, x_avg + bandwidth),
          [-10, 10], color='darkgrey')
# Plot average
plt.plot(x_avg, res_mean, color='red', marker='o', markersize=8)

# Set labels
ax1.set_xlabel('Fitted Sales')
ax1.set_ylabel('Residuals')

# Find moving average over full domain
# create x vector
x_avg = np.linspace(ypred.min() + bandwidth,
                     ypred.max() - bandwidth, 100)
res_mean = []
for x_ in x_avg:
    # Find indices where ypred is within range
    index_res = (np.where(ypred > (x_ - bandwidth)) and
                 np.where(ypred < (x_ + bandwidth)))
    # Find mean of Residuals
    res_mean.append(res[index_res[0]].mean())

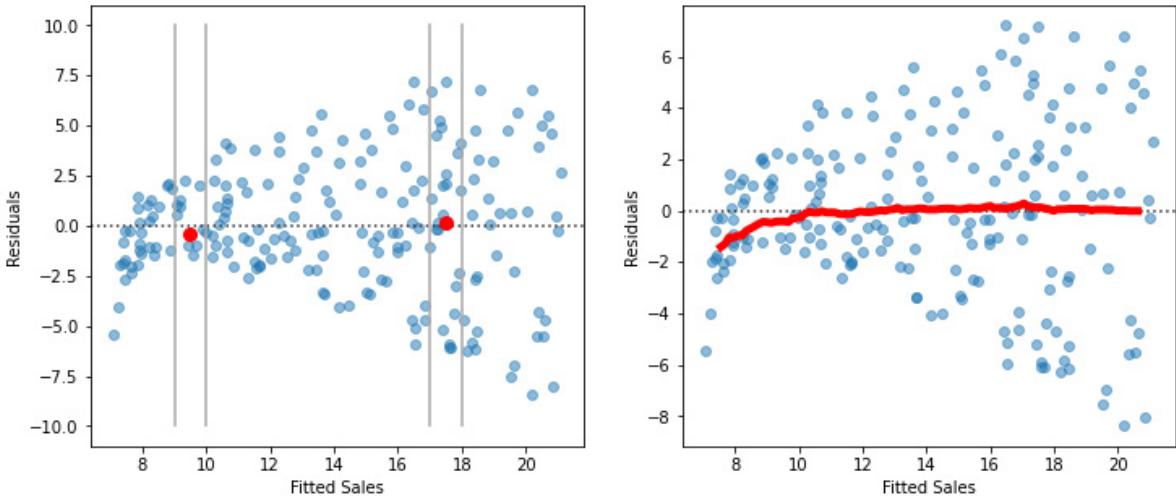
# Second subplot:
ax2 = fig.add_subplot(122)
# Plot Residuals
sns.residplot(x=yfit, y=res, data=df,
               lowess=False, scatter_kws={'alpha': 0.5})
# Plot moving average:
plt.plot(x_avg, res_mean, 'r-', linewidth=5)
# Set labels
ax2.set_xlabel('Fitted Sales')
ax2.set_ylabel('Residuals')

# Show plot
plt.show()

```

We obtain a function that is not smooth at all, but this is not a surprise. By construction, due to the rectangular kernel, data points drop out of the running mean computation abruptly, and hence the curve consists of abrupt jumps.

The smoothing curve hardly deviates from the  $r = 0$  line. We therefore conclude that the underlying regression function is linear.



**Figure 6.8.:** Smoothing curve based on the running mean estimation method. `bandwidth` steers the window width.

### Remarks:

- In this example, it is obvious that the scattering magnitude is increasing when moving to the right. This violates our assumption that the variance of the error terms be constant. We will come back to this point later.
- The Python-output for the previous example included the explicit code for the generation of the Tukey-Anscombe plot along with the smoothing curve. Since this plot is essential, there is a specific function for that purpose: [\(to R\)](#)

```
[3] : import seaborn as sns

# Reformat Data
dataframe = pd.concat([x, y], axis=1)

# Plot using seaborn
sns.residplot(x=yfit, y=dataframe.columns[-1],
               data=dataframe, lowess=True, scatter_kws={'alpha': 0.5},
               line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8})
```



Mathematically, the running mean estimate is defined as follows

$$\hat{f}_\lambda(x) = \frac{\sum_{i=1}^n w_i y_i}{\sum_{i=1}^n w_i} \quad \text{with weights} \quad w_i = \begin{cases} 1 & \text{if } |x - x_i| \leq \lambda/2 \\ 0 & \text{else} \end{cases}$$

where the parameter  $\lambda$  denotes the window width and controls the amount of smoothing. The running mean is the simplest method of *smoothing*. The resulting curve however is not smooth at all. The method can be improved by using a *Gaussian kernel*, that

is the weights are given by

$$w_i = \exp\left(-\frac{(x - x_i)^2}{\lambda}\right)$$

There is a wealth of improvements on kernel smoothing. We will not further embark in that topic. In the following, we will use the *LOESS smoother* which is based on *local parametric regression*.

### Example 6.2.6

We generate for the **Income** data the Tukey-Anscombe plot along with the LOESS smoothing curve (see Figure 6.9).

(to R)

```
[1]: import pandas as pd
import seaborn as sns
import statsmodels.api as sm
from matplotlib import pyplot as plt
from TMA_def import plot_residuals, plot_reg

# Read data
df = pd.read_csv('./data/Income.csv')
x = df['education']
y = df['income']

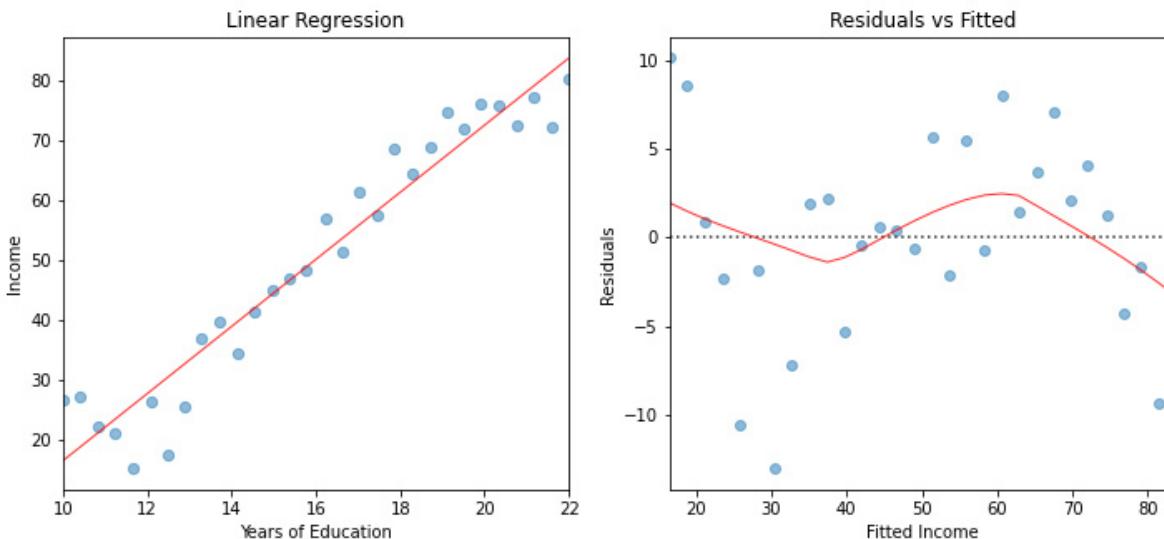
# Fit Linear Model
x_sm = sm.add_constant(x)
model = sm.OLS(y, x_sm).fit()
# Find the predicted values for the original design.
yfit = model.fittedvalues
# Find the Residuals
res = model.resid

# Create Figure and subplots
fig = plt.figure(figsize=(12, 5))
ax1 = fig.add_subplot(121)
# Plot data using definition from TMA_def
plot_reg(ax1, x, y, model, x_lab="Years of Education",
         y_lab="Income", title="Linear Regression")

# Second Subplot
ax2 = fig.add_subplot(122)
# Plot Residuals using definition from TMA_def
plot_residuals(ax2, yfit, res, n_samp=0, x_lab="Fitted Income")

# Show plots
plt.show()
```

## Chapter 6. Testing Model Assumptions: Residual Analysis



**Figure 6.9.:** Tukey-Anscombe Plot for the Example [Income](#).

The red smoothing curve is not passing any more near to the dashed  $r = 0$  line. We are now confronted with the question whether the observed deviation of the smoothing curve from the  $r = 0$  line is plausible when assuming an underlying linear model. In other words, did the observed deviation simply occur due to a random variation or is there a systematic deviation from a linear model?

If we repeat the measurements, then we would observe a different distribution of the data points and the smoothing curve would follow a different path. If the new smoothing curve then passes next to the  $r = 0$  line, we would not have any reason to question the linearity assumption.

But how can we decide whether a smoothing curve systematically deviates from the  $r = 0$  line, or when is this just due to a random variation? Generally, this is an expert call based on the magnitude of the deviation and the number of data points which are involved. An elegant way out of these (sometimes difficult) considerations is given by a *resampling approach*.

The principle idea of our resampling approach consists of simulating data points on the basis of the existing data set. For the simulated data points we fit a smoothing curve and add it to the Tukey-Anscombe plot. ([to R](#))

```
[2]: import random

random.seed(0)
n_samp = 100    # Number of resamples

# Create Figure and subplots
fig = plt.figure(figsize=(6, 5))
ax1 = fig.add_subplot(111)
```

```

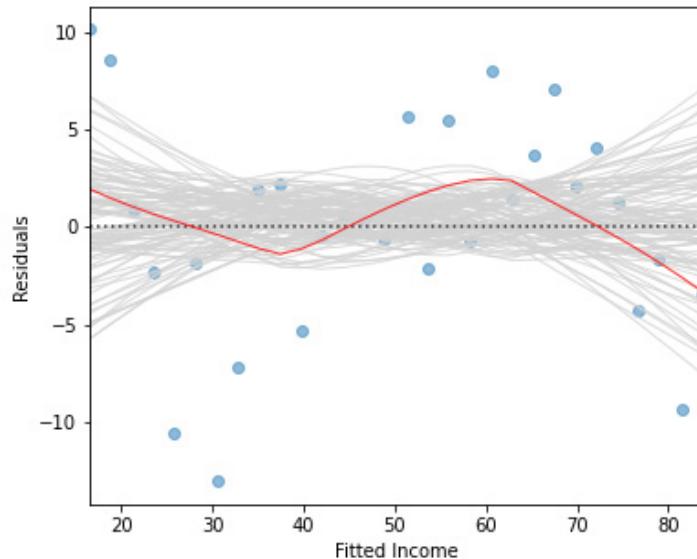
# For every random resampling
for i in range(n_samp):
    # 1. resample indices from Residuals
    samp_res_id = random.sample(list(res), len(res))
    # 2. Average of Residuals, smoothed using LOWESS
    sns.regplot(x=yfit, y=samp_res_id,
                scatter=False, ci=False, lowess=True,
                line_kws={'color': 'lightgrey', 'lw': 1, 'alpha': 0.8})
    # 3. Repeat again for n_samples

# Plot original smoothing curve
sns.residplot(x=yfit, y=res, data=df,
               lowess=True, scatter_kws={'alpha': 0.5},
               line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8})

plt.xlabel("Fitted Income")
plt.ylabel("Residuals")

# Show plots
plt.show()

```



**Figure 6.10.:** A band consisting of 100 (grey) smoothing curves is shown; these curves were fitted based on resampled data points.

**Step 1** We keep the predicted values  $\hat{y}_i$  as they are. Then, we assign to each  $\hat{y}_i$  a “new” residual  $r_i^*$  which we obtained from sampling with replacement among the existing set of  $r_i$ .

**Step 2** On the basis of the new data pairs  $(\hat{y}_i, r_i^*)$ , a smoothing curve is fitted, and it is added to the Tukey-Anscombe plot as a grey line (the resampled data points are not shown).

**Step 3** The entire process is repeated for a number of times, e.g. one-hundred times.

These simulated smoothing curves illustrate the magnitude which a random deviation from the  $r = 0$  line can take on. It may help us to assess the smoothing curve constructed on the basis of the original residuals.

If the underlying regression function  $f$  is linear, then the (grey) band resulting from the simulated smoothing curves should follow the  $r = 0$  line and contain the original (red) smoothing curve fitted on the basis of the original data set.

The red smoothing curve displayed in Figure 6.10 lies within the (grey) band of simulated smoothing curves. We therefore conclude that the wiggly shape of the (red) smoothing curve is not critical. We thus may assume a linear underlying regression function  $f$ .



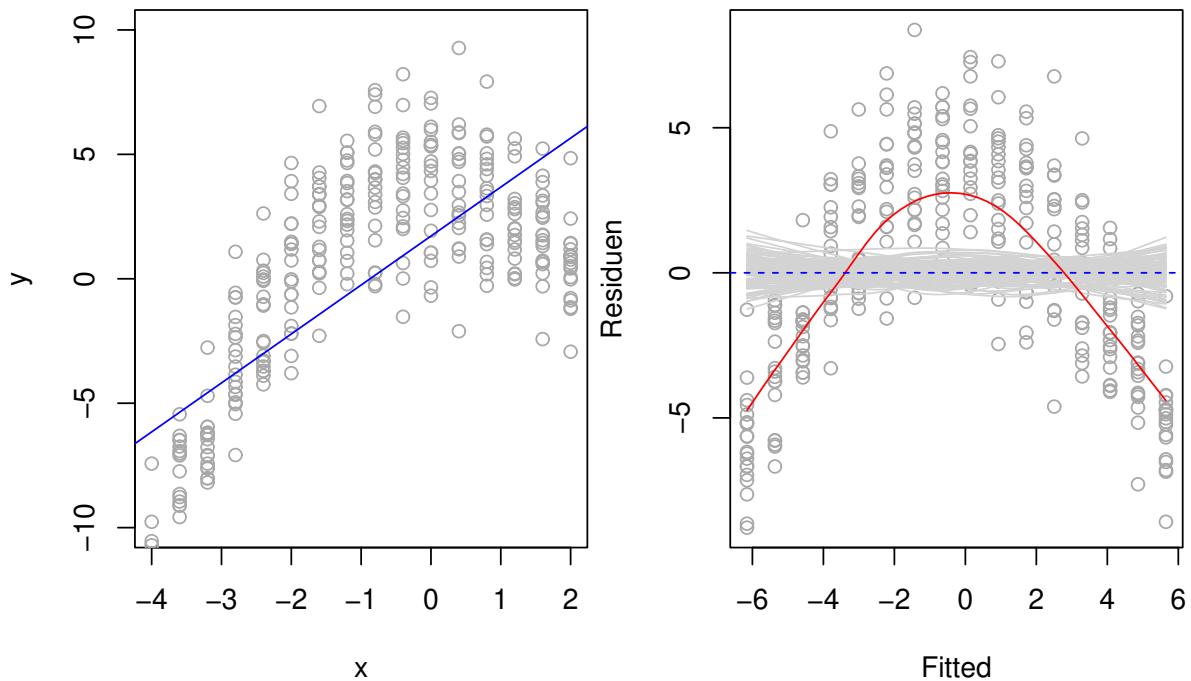
### Example 6.2.7

Figure 6.11 displays the Tukey-Anscombe plot for the example 6.1.2. In addition, 100 smoothing curves simulated on the basis of resampled data are shown in grey. The (red) smoothing curve now lies outside of the band of simulated curves. In this case, we cannot assume a linear regression function : there is an obvious *systematic* deviation from linearity.



Based on the inspection of the Tukey-Anscombe plots we decide whether there is a linear relationship between the predictor variable  $X$  and the response variable  $Y$ . In addition, we dispose of the  $R^2$  statistic which measures the degree of linear association between predictor and response variable. If we consider both tools *together*, we will have a considerably better overview of the data structure with respect to its linear associations.

If the Tukey-Anscombe plot indicates that there are non-linear associations in the data, then a simple approach is to use non-linear transformations of the predictor, such as  $\tilde{X} = \log(X)$ ,  $\tilde{X} = \sqrt{X}$  or  $\tilde{X} = X^2$  in the regression model. These transformations will eventually lead to a linear relationship between the transformed predictor variable  $\tilde{X}$  and the response variable  $Y$ .



**Figure 6.11:** There is an obvious non-linear relationship between the predictor variable  $X$  and the response variable  $Y$ . A variable transformation with  $\tilde{X} = X^2$  would establish a linear relationship between the transformed predictor  $\tilde{X}$  and the response  $Y$ .

### Example 6.2.8

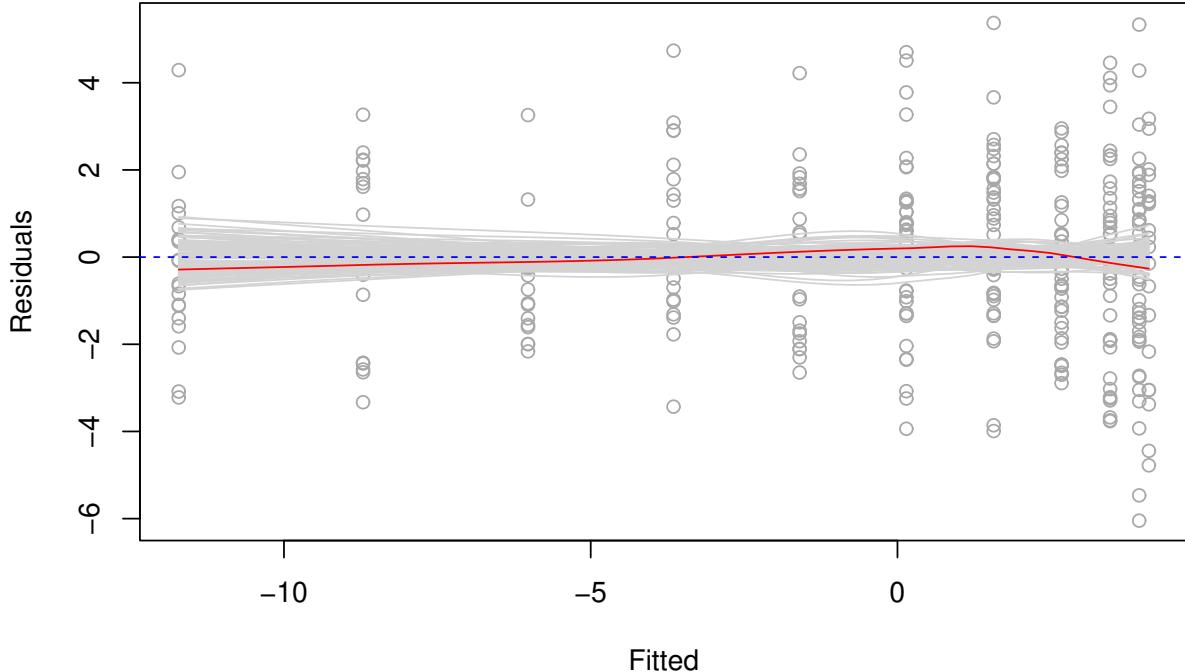
Figure 6.12 displays the Tukey-Anscombe plot with resampling for the example 6.1.2, where a variable transformation of the predictor variable was carried out. In particular, the linear regression model was fitted for the predictor variable  $\tilde{X} = X^2$ . We now observe a linear relationship between the predictor  $\tilde{X}$  and the response  $Y$  since the (red) smoothing curve follows a path within the (grey) band of simulated smoothing curves.



### 6.2.3. Diagnostics Tool for Testing the Model Assumption $\text{Var}[\varepsilon_i] = \text{Constant}$

Another important assumption of the linear regression model is that the error terms have a constant variance,  $\text{Var}[\varepsilon_i] = \sigma^2$ . The standard error, confidence intervals, and hypothesis tests associated with the linear model rely upon this assumption.

Unfortunately, it is often the case that the variances of the error terms  $\varepsilon_i$  are non-constant. In example 6.2.4 on page 140 we have seen, that for the **Advertising** data



**Figure 6.12.:** Tukey-Anscombe plot : there is a linear relationship between the transformed predictor variable  $\tilde{X} = X^2$  and the response variable  $Y$ .

the variances of the error terms  $\varepsilon_i$  are non-constant: the error terms  $\varepsilon_i$  increase with the values of the response. One can identify non-constant variances in the errors, or *heteroscedasticity*, from the presence of a *funnel shape* in the Tukey-Anscombe plot, see the right-hand panel of Figure 6.7 on page 141.

How should we measure the scattering magnitude of the error terms  $\varepsilon_i$  in order to verify whether the variances are constant? To measure to which extent the error terms scatter, we use the square root of the absolute value of the standardized residuals, that is

$$\sqrt{|\tilde{r}_i|}$$

where the standardized residuals  $\tilde{r}_i$  are defined as follows

$$\tilde{r}_i = \frac{r_i}{\hat{\sigma} \sqrt{1 - \left( \frac{1}{n} + \frac{(x_i - \bar{x})^2}{\sum_i^n (x_i - \bar{x})^2} \right)}} \quad (6.1)$$

with  $\hat{\sigma}$  denoting the estimated standard deviation of the error terms  $\varepsilon_i$ .

If the error terms  $\varepsilon_i$  are distributed according to a normal distribution, then for the distribution of the standardized residuals, we have

$$\tilde{r}_i \sim \mathcal{N}(0, 1)$$

If we plot the square root of the absolute values of the standardized residuals versus the predicted values  $\hat{y}_i$ , we obtain the so-called *scale location plot*. In order to assess to

which extent there is heteroscedasticity in the data, we simply fit a smoothing curve to the points in the scale location plot.

### Example 6.2.9

Figure 6.13 displays the *scale location plot* for the **Advertising** data. [\(to R\)](#)

```
[1]: import pandas as pd
import numpy as np
import seaborn as sns
import statsmodels.api as sm
from matplotlib import pyplot as plt

# Load data
df = pd.read_csv('./data/Advertising.csv')
x = df['TV']
y = df['sales']

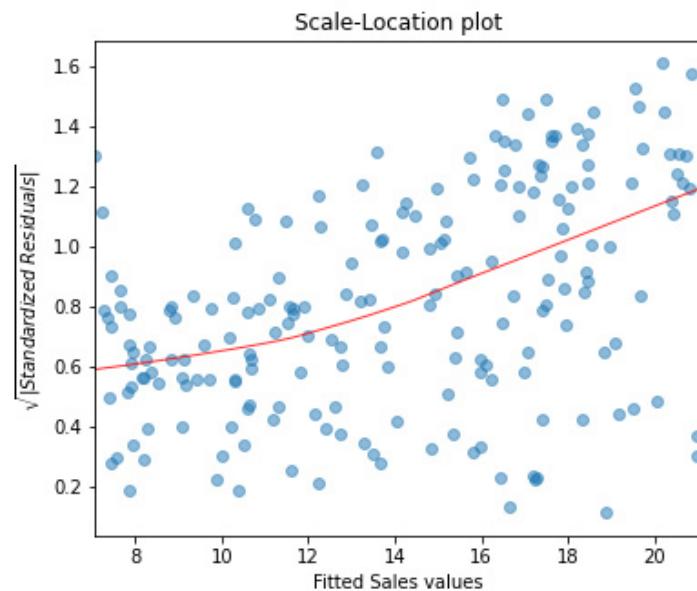
# Fit Linear Model
x_sm = sm.add_constant(x)
model = sm.OLS(y, x_sm).fit()
# Find the predicted values for the original design.
yfit = model.fittedvalues
# Residuals of the model
res = model.resid
# Influence of the Residuals
res_inf = model.get_influence()
# Studentized residuals using variance from OLS
res_standard = res_inf.resid_studentized_internal
# Absolute square root Residuals:
res_stand_sqrt = np.sqrt(np.abs(res_standard))

# Create Figure and subplots
fig = plt.figure(figsize=(6, 5))
ax1 = fig.add_subplot(111)

# plot Square rooted Residuals
plt.scatter(yfit, res_stand_sqrt, alpha=0.5)
# plot Regression using Seaborn
sns.regplot(x=yfit, y=res_stand_sqrt,
             scatter=False, ci=False, lowess=True,
             line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8})
ax1.set_title('Scale-Location plot')
ax1.set_xlabel('Fitted Sales values')
ax1.set_ylabel('$\sqrt{|Standardized Residuals|}$')

# Show plot
plt.show()
```

We observe that the magnitude of the residuals tend to increase with the fitted values, indicating a non-constant variance of the error terms  $\varepsilon_i$ . To test whether this deviation



**Figure 6.13.:** Scale location plot for the example data set **Advertising**.

is due to a random variation or whether it is of systematic nature, we will run again simulations by resampling the data.

To see whether a deviating smoothing curve in a scale location plot may be related to a random effect, we run again simulations by sampling the data with replacement and by fitting smoothing curves on the basis of the resampled data. Figure 6.14 displays a band of 100 simulated smoothing curves that were fitted on the basis of the resampled data. Since the (red) curve does not follow a path contained within this (grey) band of curves, we conclude that there is a systematic increase of the variances. [\(to R\)](#)

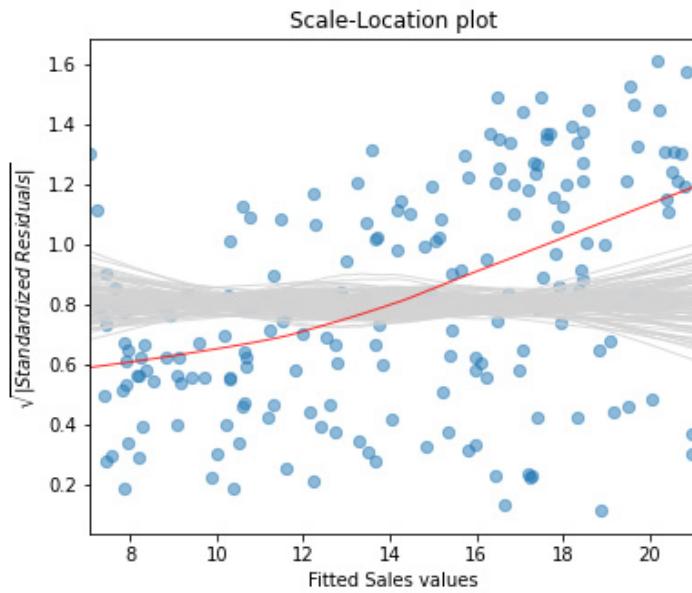
```
[2]: from TMA_def import plot_scale_loc

n_samp = 100      # Number of resamples

# Create Figure
fig = plt.figure(figsize=(6, 5))
ax1 = fig.add_subplot(111)

# Plot Standardized Residuals using definition from TMA_def
plot_scale_loc(ax1, yfit, res_stand_sqrt, n_samp=100,
                x_lab="Fitted Sales values")

# Show plot
plt.show()
```



**Figure 6.14.:** Scale location plot with smoothing curve (in red) and simulated smoothing curves fitted on the basis of resampled data (in grey).

#### Example 6.2.10

Figure 6.15 displays the scale location plot for the **Income** data set.

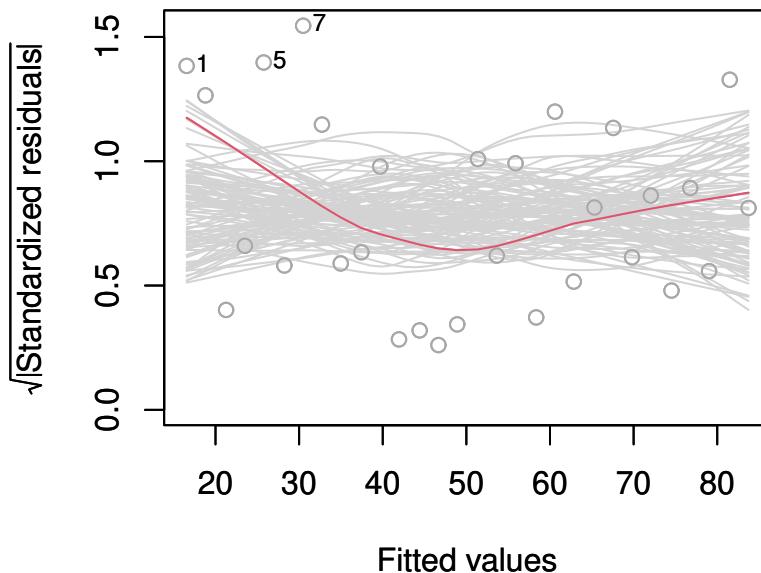
There is no indication for the red smoothing curve to deviate systematically from its horizontal line although it is not following a straight line. We conclude that the variances of the error terms  $\varepsilon_i$  are constant and the assumption of the linear regression model with respect to a constant variance seems to be met. ◀

#### 6.2.4. Diagnostics Tool for the Normal Distribution Assumption of the Errors $\varepsilon_i$

The third model assumption concerns the distribution of the error terms  $\varepsilon_i$ : the error terms  $\varepsilon_i$  are assumed to follow a normal distribution. Since we are not able to determine the error terms  $\varepsilon_i$  directly, we have to estimate them by means of the residuals  $r_i$ . In the following, we consider two graphical tools to test the third model assumption: either we plot a histogram or a Q-Q plot.

#### Example 6.2.11

The residuals of the **Advertising** data set are plotted as a histogram in Figure 6.16. A normal density function with mean and variance estimated on the basis of the data is shown in green. (to R)



**Figure 6.15.:** Scale location plot for the example `Income`. The (grey) band contains 100 smoothing curves that were fitted to the resampled data.

```
[1]: import pandas as pd
import numpy as np
import statsmodels.api as sm
from matplotlib import pyplot as plt
from scipy.stats import norm

# Load data
df = pd.read_csv('./data/Advertising.csv')
x = df['TV']
y = df['sales']

# Fit Linear Model
x_sm = sm.add_constant(x)
model = sm.OLS(y, x_sm).fit()
# Find the predicted values for the original design.
yfit = model.fittedvalues
# Residuals of the model
res = model.resid

# Create Figure and subplots
fig = plt.figure(figsize=(6, 5))
ax1 = fig.add_subplot(111)

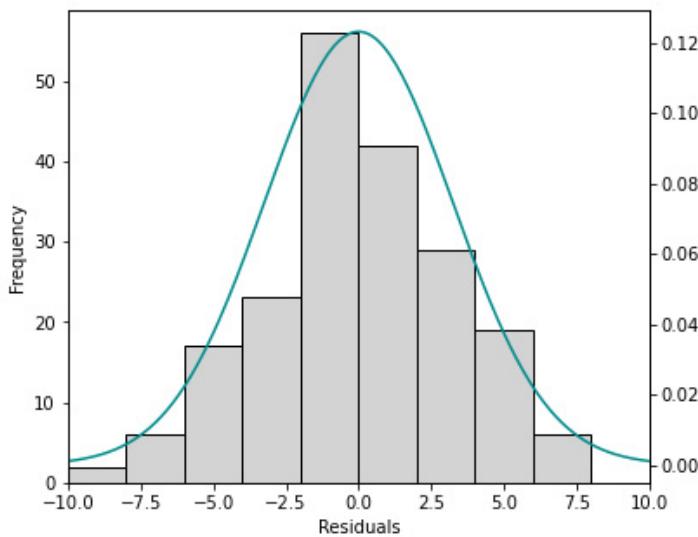
# Plot histogram
res_bins = np.arange(-10, 12, 2)
plt.hist(res, bins=res_bins,
         facecolor="lightgrey", edgecolor="k" )
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.xlim(-10, 10)
```

```

plt.ylim(bottom=0)

# Plot estimated Normal density function
# new x axis
ax2 = ax1.twinx()
# Estimate mean and standard deviation
mu = np.mean(res)
sigma = np.std(res)
x_pdf = np.arange(-10, 10.1, 0.1)
# Plot Normal density function
ax2.plot(x_pdf, norm.pdf(x_pdf, mu, sigma),
          '--', color="darkcyan", alpha=1)

# Show plot
plt.show()
    
```



**Figure 6.16.:** Histogram of the residuals  $r_i$  for the **Advertising** data set. The green curve is the normal density function for which the distribution parameters were estimated on the basis of the data.



It is in general difficult to judge on the basis of a histogram whether the data is normally distributed. In addition, histograms are sensitive with respect to the chosen interval width.

A *Q-Q plot* or *normal plot* is another graphical tool to verify whether the data is normally distributed. In our context, we plot the quantiles of the empirical residuals versus the theoretical quantiles of a normal distribution. Instead of the residuals we however use the standardized residuals  $\tilde{r}_i$  (see equation 6.2.3) in our Q-Q plot.

**Example 6.2.12**

Figure 6.17 displays the Q-Q plot for the **Advertising** data. (to R)

```
[2]: import seaborn as sns
from statsmodels.graphics.gofplots import ProbPlot

# Influence of the Residuals
res_inf = model.get_influence()
# Studentized residuals using variance from OLS
res_standard = res_inf.resid_studentized_internal

# QQ plot instance
QQ = ProbPlot(res_standard)
# Split the QQ instance in the separate data
qqx = pd.Series(sorted(QQ.theoretical_quantiles), name="x")
qqy = pd.Series(QQ.sorted_data, name="y")

# Create Figure and subplots
fig = plt.figure(figsize=(6, 5))
ax1 = fig.add_subplot(111)

sns.regplot(x=qqx, y=qqy, scatter=True, lowess=False, ci=False,
            scatter_kws={'s': 40, 'alpha': 0.5},
            line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8})
ax1.plot(qqx, qqx, '--k', alpha=0.5)

# Set labels
plt.title('Normal Q-Q')
plt.xlabel('Theoretical Quantiles')
plt.ylabel('Standardized Residuals')

plt.show()
```

If the data actually originates from a normal distribution, then the data points will scatter just slightly around the straight line in the Q-Q plot.

If we observe a deviation from the straight line in the Q-Q plot, then the question comes up whether such a deviation is systematic or due to a random variation. In order to answer this question, we run again simulations. In particular, we draw 100 random samples of length  $n$  (number of residuals) from a normal distribution that shares mean and standard deviation with the standardized residuals. Figure 6.18 displays a band of simulated (grey) curves in the case of normally distributed residuals : these curves may be observed due to random variations. Since the points for the **Advertising** data set lie within this band, we consider the residuals to originate from a normal distribution (to R)

```
[3]: # Split the QQ instance in the separate data
qqx = pd.Series(sorted(QQ.theoretical_quantiles), name="x")
qqy = pd.Series(QQ.sorted_data, name="y")
```

## Chapter 6. Testing Model Assumptions: Residual Analysis

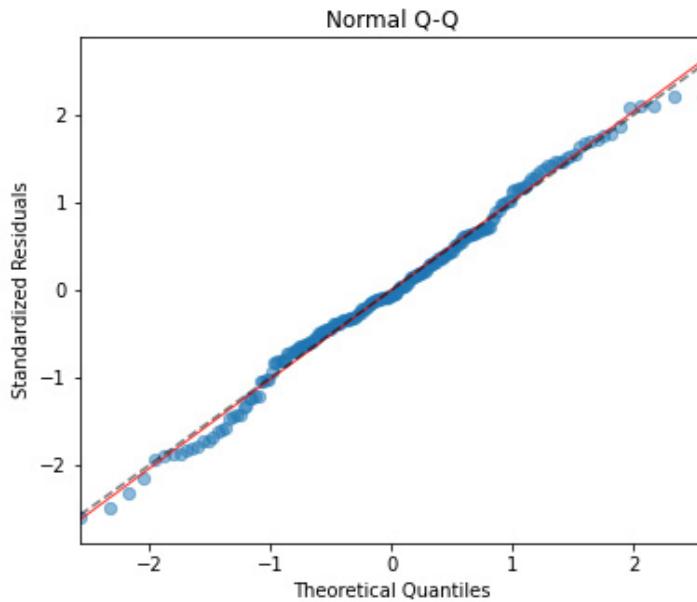


Figure 6.17.: Q-Q plot for the `Advertising` data.

```
# Estimate the mean and standard deviation
mu = np.mean(qqy)
sigma = np.std(qqy)

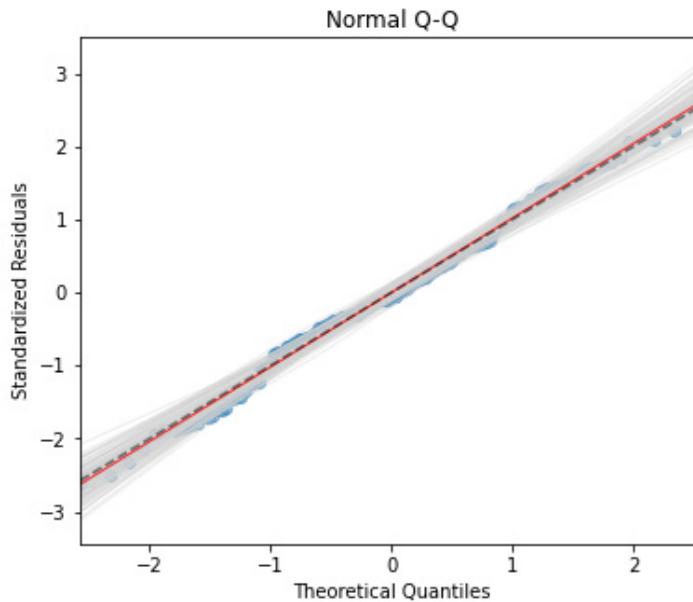
# Create Figure
fig = plt.figure(figsize=(6, 5))
ax1 = fig.add_subplot(111)

n_samp = 100    # Number of resamples
# For ever random resampling
for lp in range(n_samp):
    # Resample indices
    samp_res_id = np.random.normal(mu, sigma, len(qqx))
    # Plot
    sns.regplot(x=qqx, y=sorted(samp_res_id),
                 scatter=False, ci=False, lowess=True,
                 line_kws={'color': 'lightgrey', 'lw': 1, 'alpha': 0.4})

# Add plots for original data and the line x = y
sns.regplot(x=qqx, y=qqy, scatter=True, lowess=False, ci=False,
            scatter_kws={'s': 40, 'alpha': 0.5},
            line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8})
ax1.plot(qqx, qqx, '--k', alpha=0.5)

# Set limits and labels
plt.title('Normal Q-Q')
plt.xlabel('Theoretical Quantiles')
plt.ylabel('Standardized Residuals')
```

```
plt.show()
```



**Figure 6.18.:** The (grey) band contains 100 smoothing curves that were fitted to the resampled data.

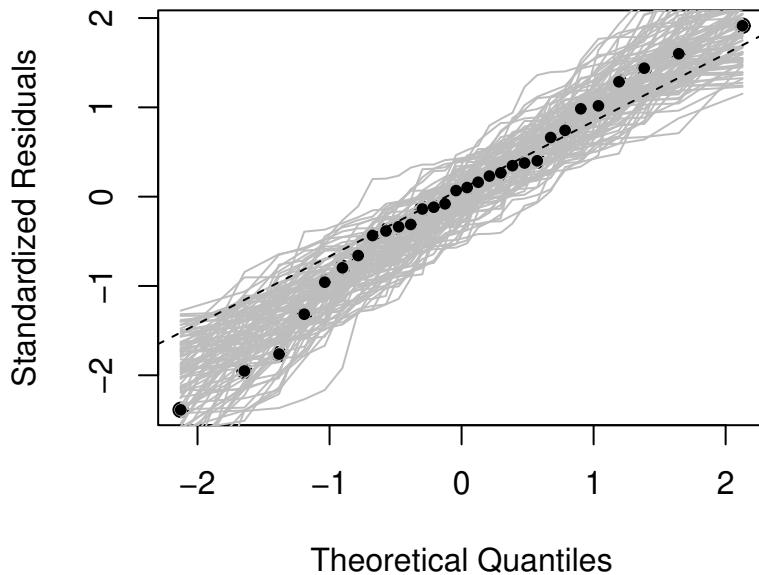
### Example 6.2.13

Figure 6.19 displays the Q-Q plot for the **Income** data.

For the **Income** data we observe in the Q-Q plot a pronounced deviation of the points from the straight line. However, the points are still contained in the grey band of simulated curves. We therefore interpret these deviations from the straight line as random variations.

### 6.2.5. Diagnostics Tool for Independence Assumption of Errors $\varepsilon_i$

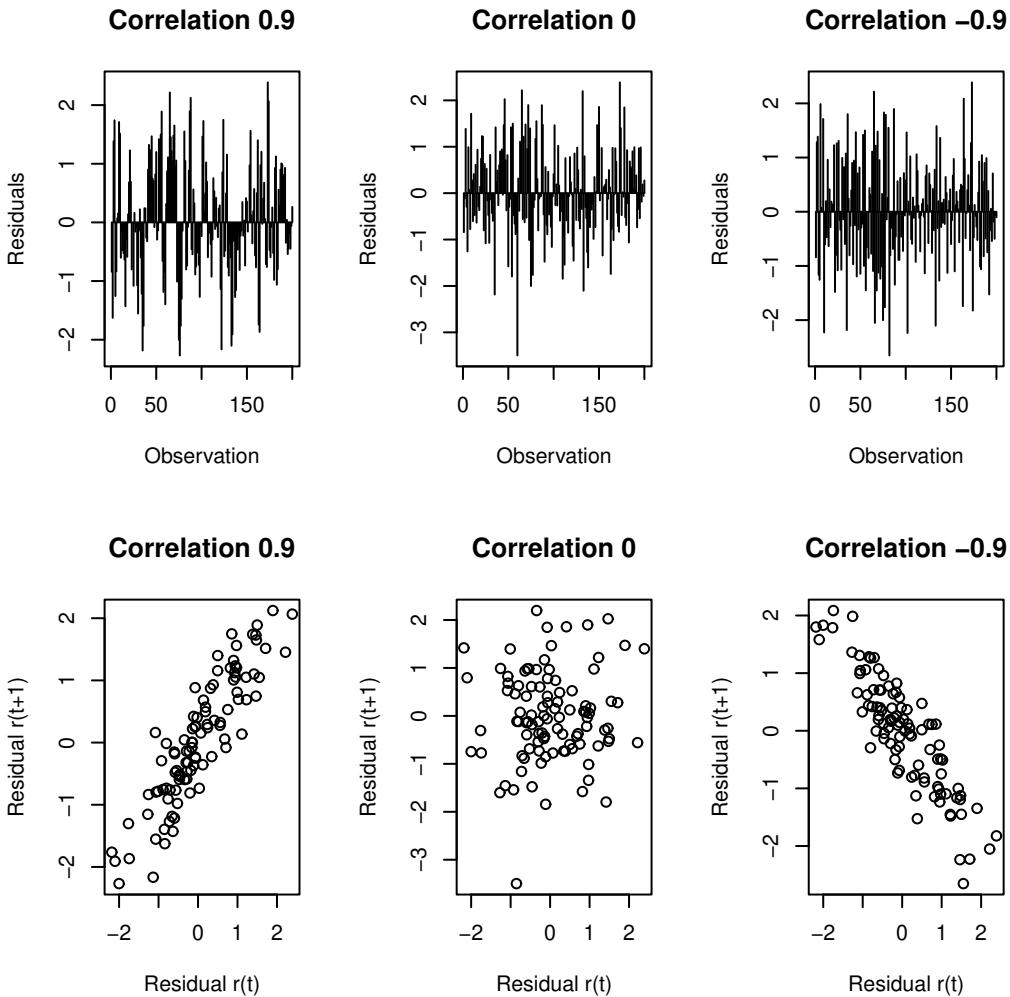
An important assumption of the linear regression model is that the error terms,  $\varepsilon_1, \dots, \varepsilon_n$ , are independent. What does this mean? For instance, if the errors are independent, then the fact that  $\varepsilon_i$  is positive provides little or no information about the sign of  $\varepsilon_{i+1}$ . The standard errors that are computed for the estimated regression coefficients or the fitted values are based on the assumption of independent error terms.



**Figure 6.19.:** Q-Q plot for the `Income` data set.

If in fact there is correlation among the error terms, then the estimated standard errors will tend to underestimate the true standard errors. As a result, confidence and prediction intervals will be narrower than they should be. For example, a 95 % confidence interval may in reality have a much lower probability than 0.95 of containing the true value of the parameter. In addition, p-values associated with the model will be lower than they should be; this could cause us to erroneously conclude that a parameter is statistically significant. In short, if the error terms are correlated, we may have an unwarranted sense of confidence in our model.

Why might correlations among the error terms occur? Such correlations frequently occur in the context of *time series* data, which consists of observations for which measurements are obtained at discrete points in time. In many cases, observations that are obtained at adjacent time points have positively correlated errors. In order to determine if this is the case for a given data set, we can plot the residuals from our model as a function of time or as a scatter plot of the residuals  $r_{t+1}$  versus the residuals  $r_t$ , see Figure 6.20. If the errors are uncorrelated, then there should be no discernible pattern. On the other hand, if the error terms are positively correlated, then we may see *tracking* in the residuals - that is, adjacent residuals may have similar values. Figure 6.20 provides an illustration. In the left-hand panel, we see the residuals from a linear regression fit to data in which adjacent errors had a correlation of 0.9. Now there is a clear pattern in the residuals - adjacent residuals tend to take on similar values. The center panel illustrates residuals from a linear regression fit to data generated with uncorrelated errors. There is no evidence of a time-related trend in the residuals. In the right-hand panel, we see residuals from a linear regression fit to data in which adjacent errors had a correlation of  $-0.9$ . We observe that adjacent residuals tend to take on similar values, but with opposite sign.



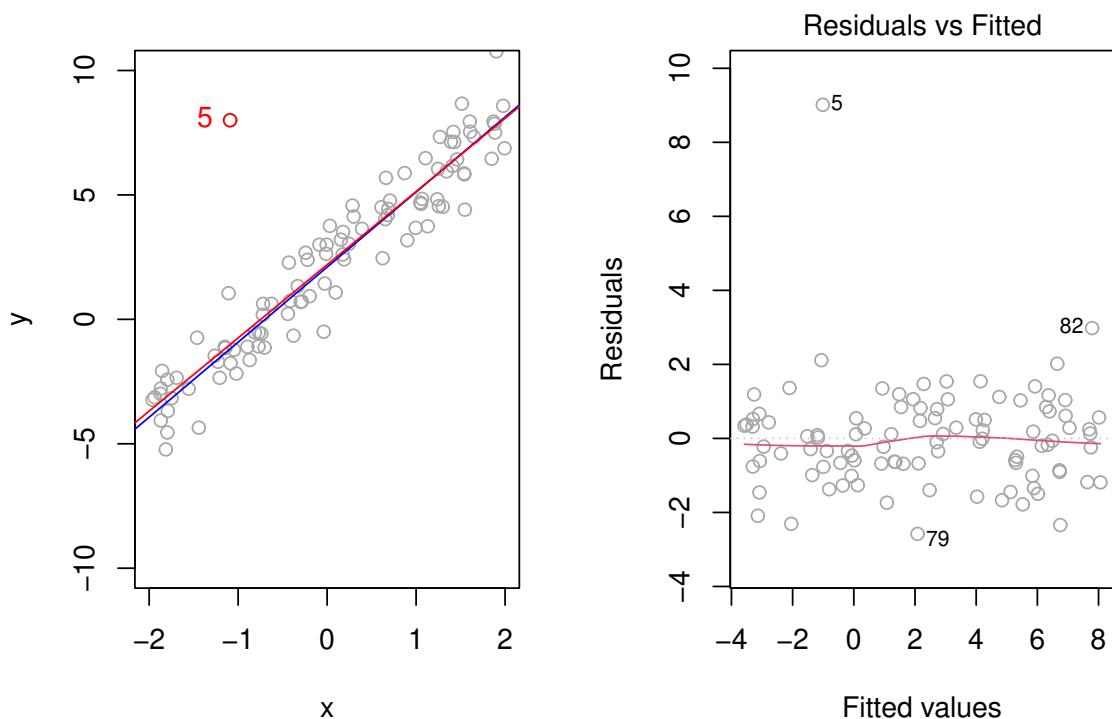
**Figure 6.20.:** Plots of residuals from simulated time series data sets generated with differing levels of correlation  $\rho$  between error terms for adjacent time points. *Left:* adjacent residuals are positively correlated; *Center:* adjacent residuals are uncorrelated; *Right:* adjacent residuals are negatively correlated.

Correlation among the error terms can also occur outside of time series data. For instance, consider a study in which individuals' heights are predicted from their weights. The assumption of uncorrelated errors could be violated if some of the individuals in the study are members of the same family, or eat the same diet, or have been exposed to the same environmental factors. In general, the assumption of uncorrelated errors is extremely important for linear regression as well as for other statistical methods, and good experimental design is crucial in order to mitigate the risk of such correlations.

## 6.2.6. Outliers and High Leverage Points

### Outlier

An *outlier* is a point for which  $y_i$  is far from the value  $\hat{y}_i$  predicted by the model. Outliers can arise for a variety of reasons, such as incorrect recording of an observation during data collection.



**Figure 6.21:** *Left:* The least squares regression line is shown in red, and the regression line after removing the outlier is shown in blue. *Right:* The Tukey-Anscombe plot clearly identifies the outlier.

The red point (observation 5) in the left-hand panel of Figure 6.21 illustrates a typical outlier. The red solid line is the least squares regression fit, while the blue line is the least squares fit after removal of the outlier. In this case, removing the outlier has little effect on the least squares line: it leads to almost no change in the slope, and a minuscule reduction in the intercept. It is typical for an outlier that does not have an unusual predictor value to have little effect on the least squares fit. However, even if an outlier does not have much effect on the least squares fit, it can cause other problems. For instance, in this example, the RSE is 1.40 when the outlier is included in the regression, but it is only 1.06 when the outlier is removed. Since the RSE is used

to compute all confidence intervals and p-values, such a dramatic increase caused by a single data point can have implications for the interpretation of the fit. Similarly, inclusion of the outlier causes the  $R^2$  to decline from 0.92 to 0.87.

Residual plots can be used to identify outliers. In this example, the outlier is clearly visible in the residual plot illustrated in the right-hand panel of Figure 6.21. But in practice, it can be difficult to decide how large a residual needs to be before we consider the point to be an outlier.

If we believe that an outlier has occurred due to an error in data collection or recording, then one solution is to simply remove the observation. If this is the case, this should be mentioned in reports. However, care should be taken, since an outlier may instead indicate a deficiency with the model, such as a missing predictor or variables that should be transformed. It is important to realize that an outlier is always defined with respect to an existing model. In a different model the outlier does not have to be necessarily an outlier.

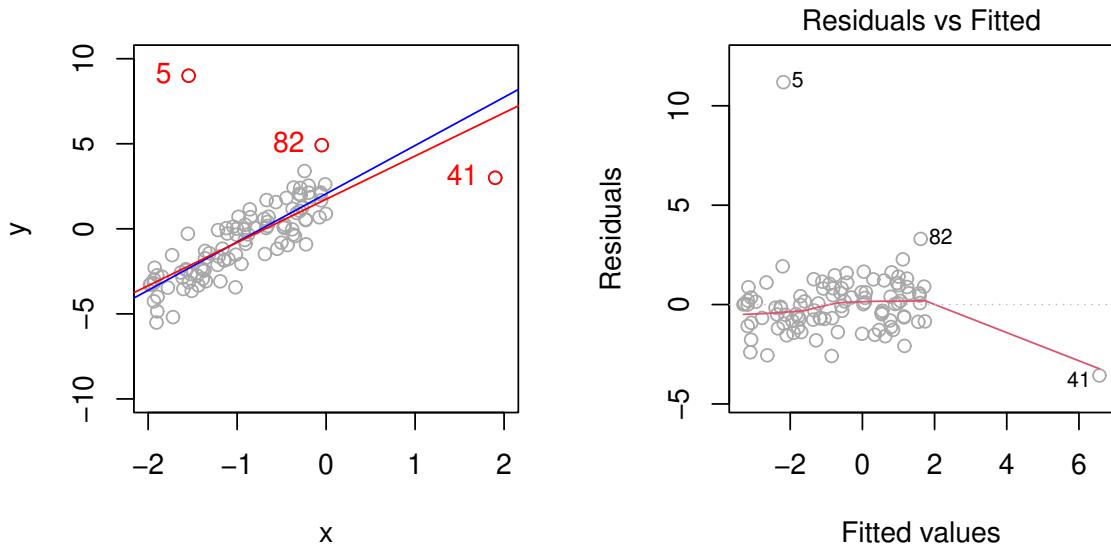
### High Leverage Points

We just saw that outliers are observations for which the response  $y_i$  is unusual given the predictor  $x_i$ . In contrast, observations with *high leverage* have an unusual value for  $x_i$ . For example, observation 41 in the left-hand panel of Figure 6.22 has high leverage, in that the predictor value for this observation is large relative to the other observations. Contrary to observation 41, observation 5 has an unusual large  $y_5$ -value relative to  $\hat{y}_5$ . Observation 5 however is an outlier and does *not* have a high leverage since the predictor value is near to the other predictor values.

The red line is the least squares fit to the data, while the blue line is the fit produced when observation 41 is removed. Comparing the left-hand panels of Figures 6.21 and 6.22, we observe that removing the high leverage observation has a much more substantial impact on the least squares line than removing the outlier. In fact, high leverage observations tend to have a sizeable impact on the estimated regression line. It is cause for concern if the least squares line is heavily affected by just a couple of observations, because any problems with these points may invalidate the entire fit. For this reason, it is important to identify high leverage observations.

In order to quantify an observation's leverage, we compute the *leverage statistic*. A large value of this statistic indicates an observation with high leverage. For a simple linear regression the value of the leverage statistic  $h_i$  associated with the  $i$ th observation is given by

$$h_i = \frac{1}{n} + \frac{(x_i - \bar{x})^2}{\sum_{i'=1}^n (x_{i'} - \bar{x})^2}$$



**Figure 6.22:** *Left:* Observation 41 is a high leverage point, while 5 is not. The red line is the fit with all the data, and the blue line is the fit with observation 41 removed. A leverage point has a high influence on the regression line, contrary to an outlier. *Right:* As can be seen from the Tukey-Anscombe plot, observation 41 has a high leverage and a high residual. Observation 5 only has a high residual value.

It is clear from this equation that  $h_i$  increases with the distance of  $x_i$  from  $\bar{x}$ . We interpret a high leverage point as a point having a large distance from the center of gravity  $\bar{x}$  and thereby having a high “momentum to turn the regression line around”.

The leverage statistic  $h_i$  is always between  $1/n$  and 1, and the average leverage for all the observations is always equal to  $2/n$  in the case of simple linear regression. So if a given observation has a leverage statistic that greatly exceeds  $2/n$ , then we may suspect that the corresponding point has *high leverage*. An outlier that is as well a high leverage observation represents a particularly dangerous combination.

By means of the leverage statistic  $h_i$  we can define another statistic to measure the influence of an observation: *Cook's distance*. Cook's distance measures to which extent the predicted value  $\hat{y}_i$  changes if the  $i$ th observation is removed:

$$d_i = \frac{1}{\hat{\sigma}^2} \cdot \left( \underline{\hat{y}}_{(-i)} - \underline{\hat{y}} \right)^T \left( \underline{\hat{y}}_{(-i)} - \underline{\hat{y}} \right) \quad (6.2)$$

where  $\underline{\hat{y}}_{(-i)}$  denotes the vector of predicted values if the  $i$ th observation is removed. Cook's distance can efficiently be calculated by means of the leverage statistic  $h_i$  and the standardized residual  $\tilde{r}_i$ :

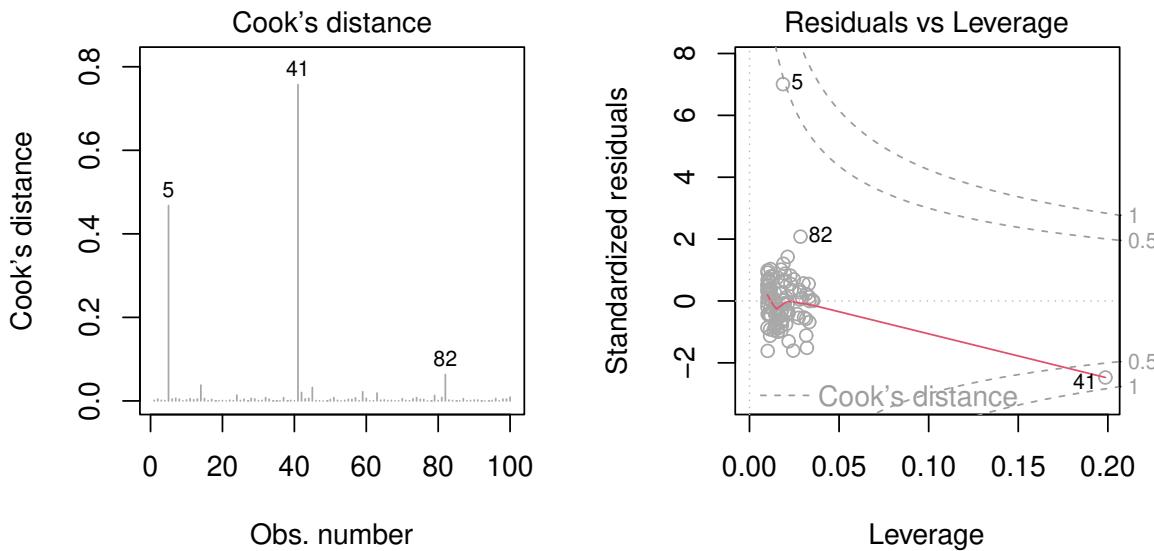
$$d_i = \tilde{r}_i^2 \frac{h_i}{2(1 - h_i)} \quad (6.3)$$

The larger the value of Cook's distance  $d_i$  is, the higher is the influence of the corresponding observation on the estimation of the predicted value  $\hat{y}_i$ . In practice, we consider a value of Cook's distance larger than 1 as *dangerously influential*.

The left-hand panel of Figure 6.23 displays the values of Cook's distance  $d_i$  for all observations. The Cook's distance can be easily found using the `OLSInfluence.cooks_distance` attribute, where `OLSInfluence` can be obtained using `OLSResult.get_influence()`.

On the basis of Cook's distance alone, we cannot understand why an observation has an influence that may be considered as dangerous. Because Cook's distance is a function of the standardized residual  $\tilde{r}_i$  and of the leverage statistic  $h_i$ , we can plot Cook's distance as contour lines in a scatter plot with the standardized residuals  $\tilde{r}_i$  plotted versus the leverage statistic  $h_i$ , see the right-hand panel of Figure 6.23. After calculation the Cook's distance as mentioned before, the contour lines can be plotted using `matplotlib.pyplot.contour()`. To assess to which extent observations are influential, contour lines for values of Cook's distance of 0.5 and 1 are added to the scatter plot.

Figure 6.23 shows that observation 41 is a high leverage point and has a relatively small standardized residual value. This combination of large absolute values for the leverage statistic and the standardized residual leads to a Cook's distance of approximately 0.8, not far away from 1. This is below the value we would consider as dangerous, but it means we have to be careful about this observation. Observation 5 has a large residual value, but its leverage statistic is rather small. Its Cook's distance can be considered as relatively high, but clearly not as dangerous.



**Figure 6.23.:** *Left:* Cook's distance  $d_i$  is plotted for every observation  $i$ . *Right:* Scatter plot with standardized residuals  $\tilde{r}_i$  versus leverage statistic  $h_i$ . Values of 0.5 and 1 for Cook's distance are plotted as contour lines.

### 6.2.7. Summary

We have seen 4 possibilities how to analyze residuals graphically.

#### Example 6.2.14

With `Python`, the first, third and fourth figures can easily be plotted using the `seaborn.residplot()` and `seaborn.regplot()` to create the plots, and the information provided by the different attributes of `OLSResults.get_influence()`. Unfortunately, `Seaborn` does not have an explicit function to plot the qq-plot. Therefore, we will use the function `ProbPlot()` provided by `statsmodels.graphics.gofplots`.

(to R)

```
[1]: import numpy as np
import pandas as pd
import seaborn as sns
import statsmodels.api as sm
from matplotlib import pyplot as plt
from statsmodels.graphics.gofplots import ProbPlot
from TMA_def import *

# Read data
```

## Chapter 6. Testing Model Assumptions: Residual Analysis

```
df = pd.read_csv('./data/Advertising.csv')
x = df['TV']
y = df['sales']

# Reformat Data
dataframe = pd.concat([x, y], axis=1)

""" Find Predictions, Residuals and Influence on the Residuals """
# Fit Linear Model
x_sm = sm.add_constant(x)
model = sm.OLS(y, x_sm).fit()

# Find the predicted values for the original design.
yfit = model.fittedvalues
# Find the Residuals
res = model.resid
# Influence of the Residuals
res_inf = model.get_influence()
# Studentized residuals using variance from OLS
res_standard = res_inf.resid_studentized_internal
# Absolute square root Residuals:
res_stand_sqrt = np.sqrt(np.abs(res_standard))
# Cook's Distance and leverage:
res_inf_cooks = res_inf.cooks_distance
res_inf_leverage = res_inf.hat_matrix_diag

""" Plots """
# Create Figure and subplots
fig = plt.figure(figsize = (14,12))

# First subplot Residuals vs Fitted values
ax1 = fig.add_subplot(2, 2, 1)
plot_residuals(ax1, yfit, res)

# Second subplot Q-Q Plot
ax2 = fig.add_subplot(2, 2, 2)
plot_QQ(ax2, res_standard)

# Third subplot: Scale location
ax3 = fig.add_subplot(2, 2, 3)
plot_scale_loc(ax3, yfit, res_stand_sqrt, x_lab='Fitted Sales values')

# Fourth subplot: Cook's distance
ax4 = fig.add_subplot(2, 2, 4)

sns.regplot(x=res_inf_leverage, y=res_standard,
            scatter=True, ci=False, lowess=True,
            scatter_kws={'s': 40, 'alpha': 0.5},
            line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8})
```

## Chapter 6. Testing Model Assumptions: Residual Analysis

```
x_min, x_max = min(res_inf_leverage) - 0.001, max(res_inf_leverage) + 0.002
y_min, y_max = min(res_standard) - 0.5, max(res_standard) + 0.5
# Plot centre line
plt.plot((x_min, x_max), (0, 0), 'g--', alpha=0.8)
# Plot contour lines for Cook's Distance levels
n = 100
n_pred = 1 # Number of predictors
cooks_distance = np.zeros((n, n))
x_cooks = np.linspace(x_min, x_max, n)
y_cooks = np.linspace(y_min, y_max, n)

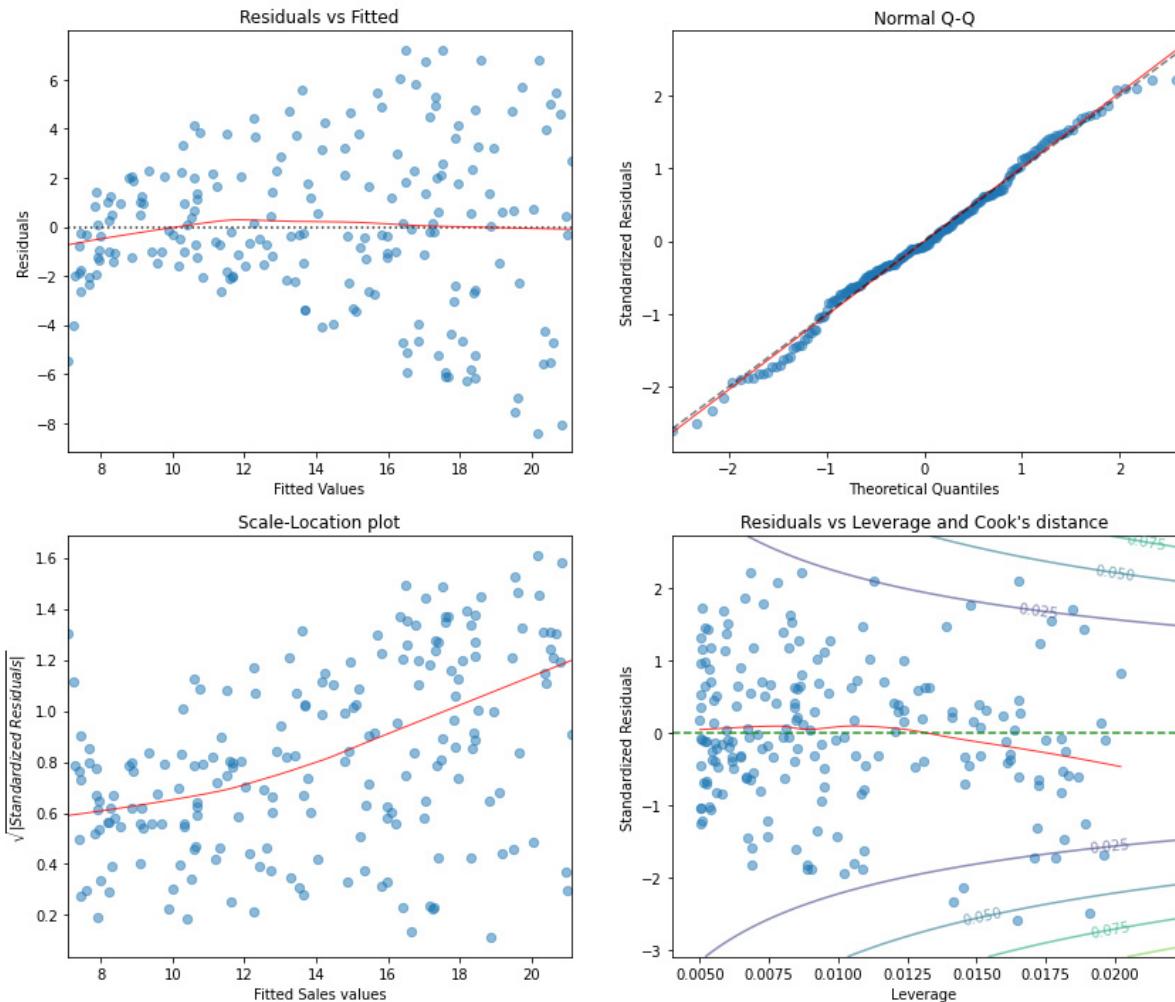
for xi in range(n):
    for yi in range(n):
        cooks_distance[yi][xi] = \
            y_cooks[yi]**2 * x_cooks[xi] / (1 - x_cooks[xi]) / (n_pred + 1)
CS = ax4.contour(x_cooks, y_cooks, cooks_distance, levels=4, alpha=0.6)

ax4.clabel(CS, inline=0, fontsize=10)
ax4.set_xlim(x_min, x_max)
ax4.set_ylim(y_min, y_max)
ax4.set_title('Residuals vs Leverage and Cook\\'s distance')
ax4.set_xlabel('Leverage')
ax4.set_ylabel('Standardized Residuals')

# Show plot
# plt.tight_layout()
plt.show()
```



## Chapter 6. Testing Model Assumptions: Residual Analysis



**Figure 6.24.:** Residual plots for the `Advertising` data set.

### Example 6.2.15

For the `Income` data set the residual plots are displayed in Figure 6.25.

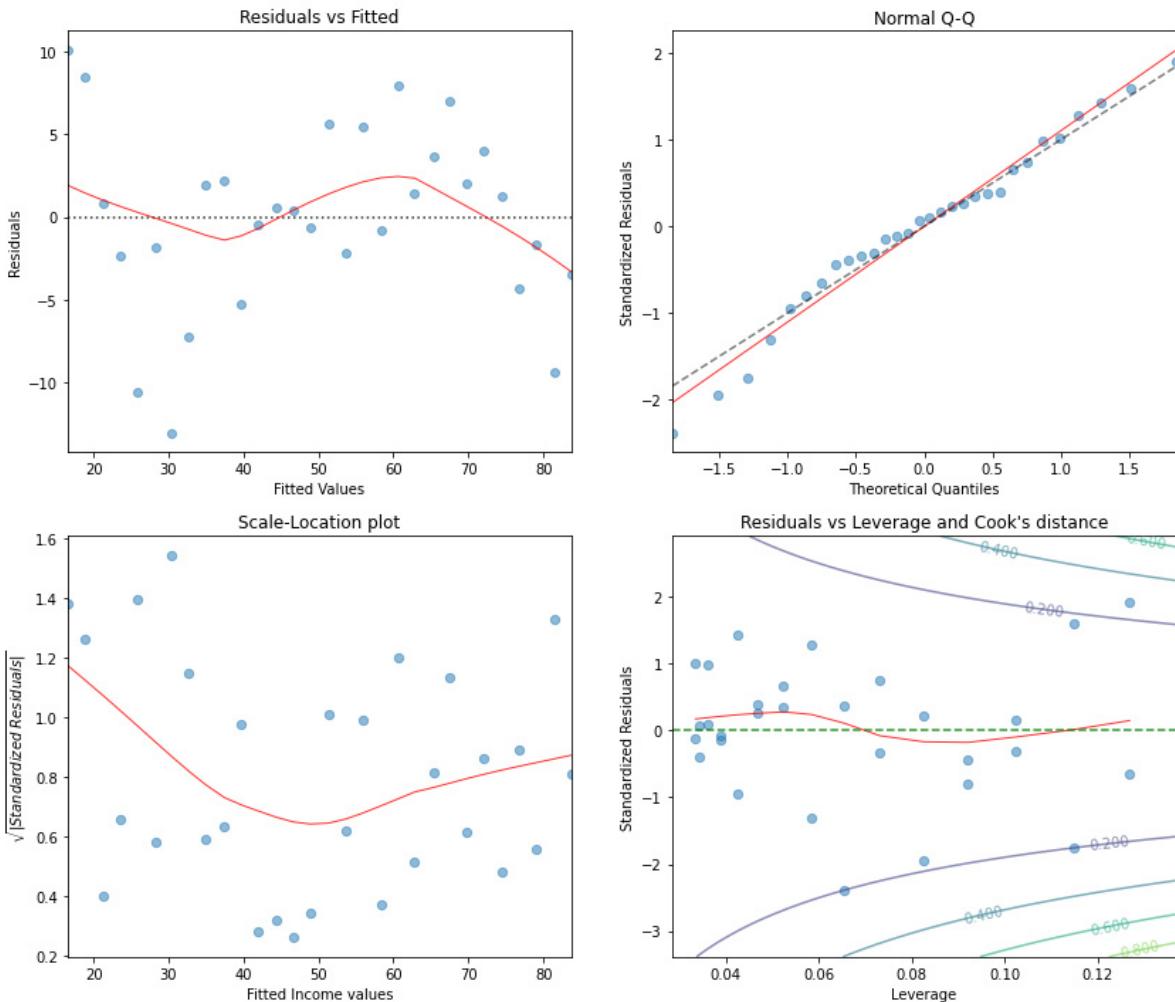


Figure 6.25.: Residual plots for the `Income` data set.

## 6.3. Treatment of Deficiency in Model Assumptions

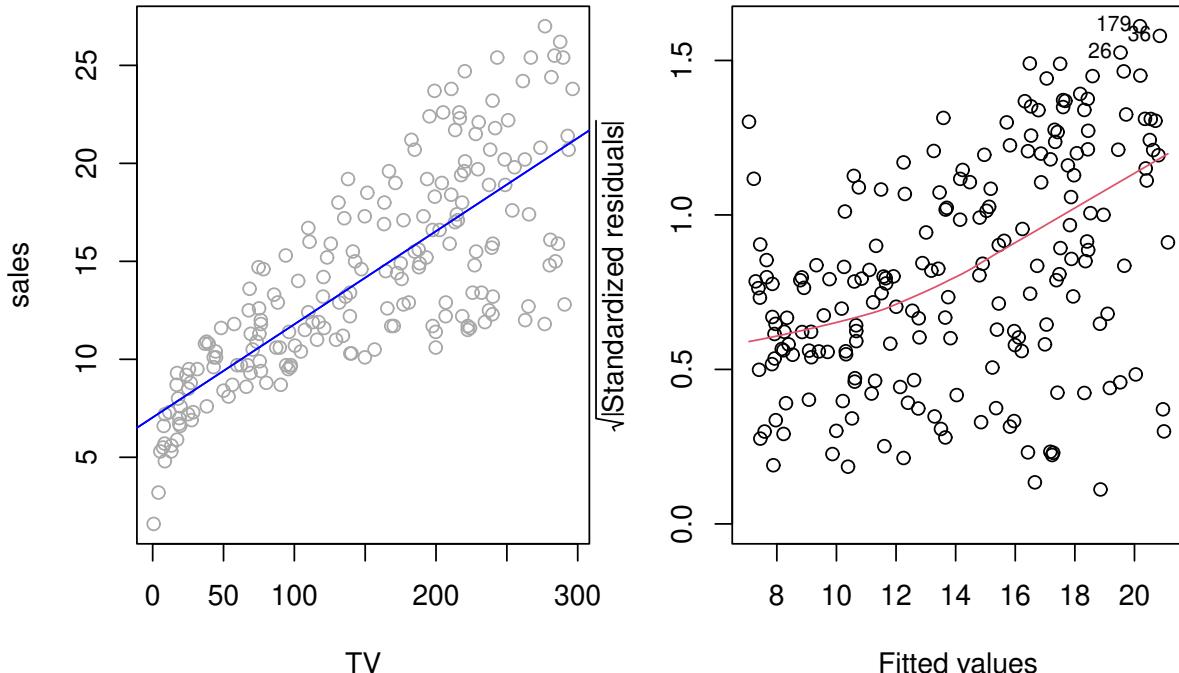
### 6.3.1. Therapeutical Treatment in the Case of $E[\varepsilon_i] \neq 0$

If the Tukey-Anscombe plot indicates a non-linear structure in the data (the underlying regression function  $f$  is non-linear), then a non-linear transformation of the predictor variable such as  $\tilde{X} = \log(X)$ ,  $\tilde{X} = \sqrt{X}$  or  $\tilde{X} = X^2$  may help to establish a linear relationship between the transformed variable  $\tilde{X}$  and the response variable  $Y$ . In example 6.2.8 we already have seen, that such a transformation may solve the problem of non-linearity.

If no transformation of  $X$  solves the problem of non-linearity, then an additional predictor in the regression model such as  $X^2$  may help. We will discuss these extensions of the simple linear model in chapter about multiple linear regression.

### 6.3.2. Therapeutical Treatment for $\text{Var}[\varepsilon_i] \neq \text{Constant}$

Often the assumption that the error terms  $\varepsilon_i$  have constant variance is violated. A dependence of the scattering magnitude of the variances from the (predicted) values of the response can be detected with the help of a *scale location plot*, see Figure 6.26.

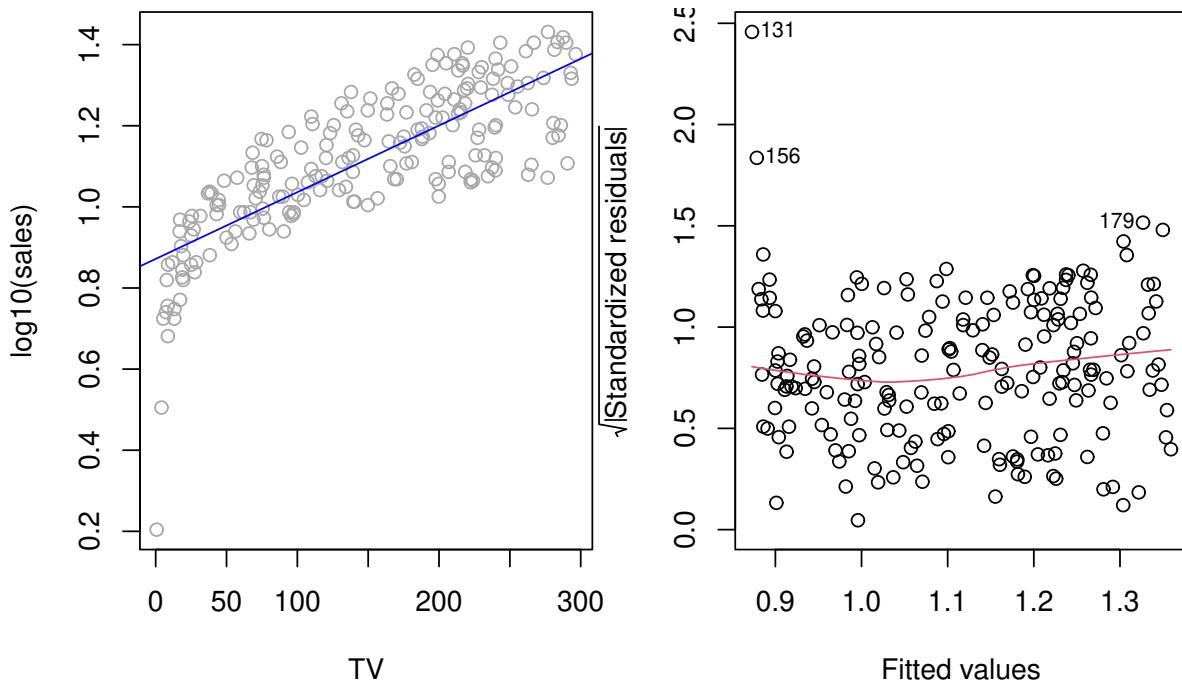


**Figure 6.26.:** Scale location plot for the **Advertising** data set with response variable **sales**. The scattering magnitude of the residuals increases with the predicted values  $\hat{y}_i$ .

If the standard deviation of the residuals is proportional to the predicted values, then a log-transformation of the response variable may lead to a constant variance, see Figure 6.27. If the effect of the log-transformation is too strong, then a square root transformation of the response variable may work better.

Further recommendations when and how variable transformations may be applied were named *first aid transformations* by J.W. Tukey; they rely upon the experience from statistical data analysis and aim in first place at transforming the response variable. But the transformations may as well be applied to a predictor variable.

- log-transformation for concentration data and absolute values, that is for continuous random variables that take on only positive values
- square root transformation in the case of count data (discrete random variables)
- the so-called arcsine-transformation  $\tilde{Y} = \arcsin(\sqrt{Y})$  or the logit-transformation  $\tilde{Y} = \log\left(\frac{Y+0.005}{1.01-Y}\right)$  for percentage data



**Figure 6.27.:** Scale location plot for the **Advertising** data with log-transformed response variable **sales**.

### 6.3.3. Therapeutical treatment for Non-Normally Distributed $\varepsilon_i$

If the Q-Q plot of the residuals exhibits a symmetrical but long-tailed distribution, then variable transformations are not helping a lot. We may be tempted to leave out the most extreme observations until the long-tailed distribution characteristics disappear. Confidence intervals and p-values that are estimated on the basis of the remaining data points then should be dealt with a lot of caution. Least squares estimation methods are sub-optimal in the case of a long-tailed distribution of the errors. *Robust estimation methods* are dealing with that problem by weighing extreme observations less; such parameter estimation methods are much more reliable.

### 6.3.4. Therapeutic Treatment in the Case of Outliers and High Leverage Points

Outliers can be as clearly identified in the Tukey-Anscombe plot as in the Q-Q plot. First, we would like to recall that an observation is considered an outlier with respect to a given model that is not fitting this observation. Variable transformations may change the model so that the new model suddenly fits the observation that previously was considered an outlier. Such a variable transformation represents a highly interesting adaptation of the model and in some cases an interesting discovery.

If we identify an outlier, then we should check whether it has occurred due to an error in data collection or recording. If we believe that an outlier has occurred due to an error in data collection, then one solution is to simply remove the observation. If there is however no reason to doubt any observation in the data collection, then we first will attempt to transform predictor or response variables in order to make “disappear” the outlier. If such model adaptations do not help to explain the outlier, then we may consider the situation that this outlier occurred due to an unusual random variation. In the case of long-tailed distributions, outliers may occur with higher probability. If such outliers affect parameter estimations too much, then the observation may be removed. This however needs to be mentioned in the reports.

## Conceptual Objectives

You should be able...

- to explain the assumptions concerning the error term  $\varepsilon$  in a linear regression model.
- to explain how the quality of a linear regression fit can be assessed by means of the RSE and the  $R^2$  statistics.
- to explain which model assumptions the Tukey-Anscombe plot is testing.
- to explain how a resampling approach may help to decide whether deviations of a smoothing curve in a Tukey-Anscombe plot may be due to a violation of the model assumptions.
- to explain the concept of a scale location plot and how it assesses the constant variance assumption.
- to decide on the basis of resampled curves whether deviations of a smoothing curve in a scale-location plot may be due to a violation of the model assumptions.
- to assess the normal distribution assumption of the errors by means of a QQ-plot.
- to distinguish between outliers and high leverage points and to explain how to handle them.
- to define Cook’s distance and know when a data point should be considered as dangerously influential.
- to carry out appropriate variable transformations in case of a violation of a model assumption.

## Computational Objectives

You should be able...

- to determine the value of the  $R^2$  statistics and of the RSE on the basis of the Python-output of `OLSResult.summary()` and `OLSResult.mse_resid` respectively.
- to generate a Tukey-Anscombe plot by means of `seaborn.residplot()`
- to generate a scale location plot by means of `seaborn.regplot()`
- to generate a QQ-plot by means of the `statmodels.graphics.gofplots.ProbPlot()`
- to generate a scatter plot with the contour lines of Cook's distance by means of `matplotlib.pyplot.contour()` using Cook's distances found with `OLSInfluence.cooks_distance`.

# Chapter 7.

## Multiple Linear Regression<sup>1</sup>

### 7.1. Introduction

Simple linear regression is a useful approach for predicting a response on the basis of a single predictor variable. However, in practice we often have more than one predictor.

#### Example 7.1.1

In the **Advertising** data, we have examined the relationship between **sales** and **TV** advertising. We also have data for the amount of money spent on the **radio** and on **newspaper** advertising, and we may want to know whether either of these two media is associated with sales. How can we extend our analysis of the **Advertising** data in order to accomodate these two additional predictors?

One option is to run three separate simple linear regressions, each of which uses a different advertising medium as a predictor. Figure 7.1 displays the three separate simple regressions: the regression lines are plotted in blue.

Regression coefficients and more results of the regression model are shown in Table 7.1. These results were obtained through calculations analogous to the ones carried out in example 5.3.4 on page 117 using the least squares estimation method.

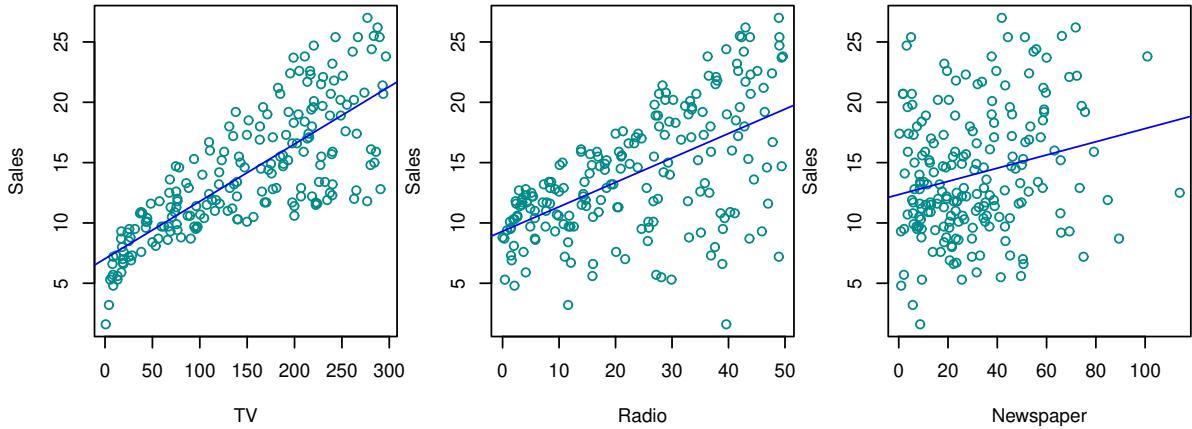
For instance, we can fit a simple linear regression model to predict **sales** on the basis of the amount spent on **radio** advertisements. We find that a CHF 1000 increase in spending on radio advertising is associated with an increase in sales by around 203 units (see center Table 7.1). On the other hand, a CHF 1000 increase in newspaper advertising budget is associated with an increase in sales by approximately 55 units (see bottom Table 7.1).

However, the approach of fitting a separate simple linear regression model for each predictor is not entirely satisfactory. First of all, it is unclear how to make a single

---

<sup>1</sup>This chapter follows Chapter 3.2 of the text book *An Introduction to Statistical Learning: with Applications in R* by G. James et al., Springer Texts in Statistics 2013.

## Chapter 7. Multiple Linear Regression



**Figure 7.1.:** Regression lines are added to Figure 4.1 for the **Advertising** data set.

prediction of **sales** given levels of the three advertising media budgets, since each of the budgets is associated with a separate regression equation. Second, each of the three regression equations ignores the other two media in forming estimates for the regression coefficients. We will see shortly that if the media budgets are correlated with each other in the 200 markets that constitute our data set, then this can lead to very misleading estimates of the individual media effects on sales.



Instead of fitting a separate simple linear regression model for each predictor, a better approach is to extend the simple linear regression model

$$Y = \beta_0 + \beta_1 X + \varepsilon$$

so that it can directly accomodate multiple predictors. We can do this by giving *each* predictor a separate slope coefficient in a single model.

In general, suppose that we have  $p$  distinct predictors. Then the *multiple linear regression model* takes the form

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p + \varepsilon$$

where  $X_j$  represents the  $j$ th predictor and  $\beta_j$  quantifies the association between that variable and the response  $Y$ .

We interpret  $\beta_j$  as the *average* effect on  $Y$  of a one unit increase in  $X_j$ , *holding all other predictors fixed*.

## Chapter 7. Multiple Linear Regression

Simple linear regression of **sales** on **TV**

	Coefficient	Std.error	t-statistic	p-value
Intercept	7.033	0.458	15.36	< 0.0001
TV	0.048	0.003	17.67	< 0.0001

Simple regression of **sales** on **radio**

	Coefficient	Std.error	t-statistic	p-value
Intercept	9.312	0.563	16.54	< 0.0001
Radio	0.203	0.020	9.92	< 0.0001

Simple regression of **sales** on **newspaper**

	Coefficient	Std.error	t-statistic	p-values
Intercept	12.351	0.621	19.88	< 0.0001
Newspaper	0.055	0.017	3.30	< 0.0001

**Table 7.1.:** More simple linear regression models for the **Advertising** data. Coefficients of the simple linear regression model for number of units sold on (*top*): **TV** advertising budget and (*center*): **radio** advertising budget and (*bottom*): **newspaper** advertising budget.

### Example 7.1.2

In the **Advertising** example, the linear regression model becomes:

$$\text{sales} = \beta_0 + \beta_1 \cdot \text{TV} + \beta_2 \cdot \text{radio} + \beta_3 \cdot \text{newspaper} + \varepsilon$$

hence

$$\text{sales} \approx \beta_0 + \beta_1 \cdot \text{TV} + \beta_2 \cdot \text{radio} + \beta_3 \cdot \text{newspaper}$$



Since the multiple linear regression model represents a generalization of the simple regression model, the estimation methods and interpretations are significantly more complicated as compared to the simple regression model.

In Chapters 5 and 6 we have discussed graphical methods to plot a simple regression model. In the case of multiple regression, we are not able to plot such a model for an arbitrary number of predictors. For example, the data points of the example 7.1.2 cannot be plotted since all three predictors already require the three coordinate axes. Let us illustrate the case with two predictors in the following example.

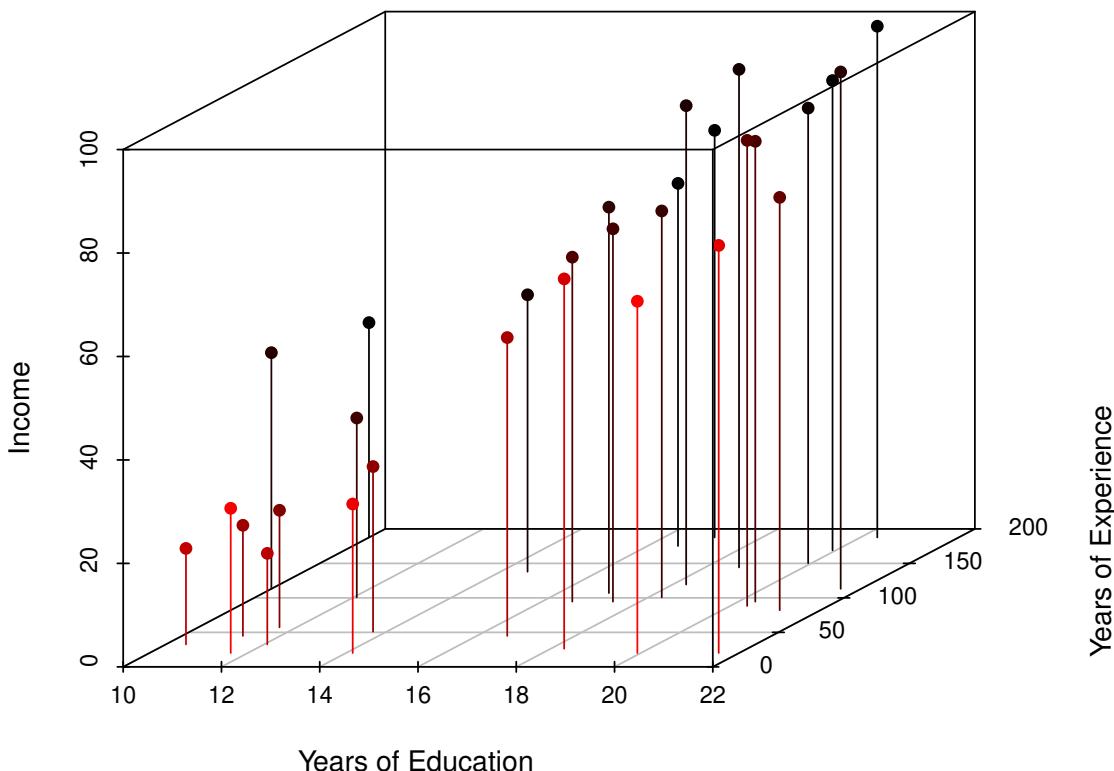
### Example 7.1.3

Until now, we have considered only one predictor in the **Income** data set: years of **education**. However the income depends obviously as well on the years of **experience**.

Thus, we have the following multiple regression model

$$\text{income} = \beta_0 + \beta_1 \cdot \text{education} + \beta_2 \cdot \text{experience} + \varepsilon$$

Since we only have two predictors, the data points can be visualized in a 3D plot (see Figure 7.2).



**Figure 7.2.:** Data points in 3D coordinate system for the **Income** data set.

(to R)

## Chapter 7. Multiple Linear Regression

```
[1]: import pandas as pd
import matplotlib.pyplot as plt

# Load data
df = pd.read_csv(
    '../.../Themen/Einfache_Lineare_Regression/Daten/Income2.csv')
x = df['education']
y = df['experience']
z = df['income']

# Create Figure and plot
fig = plt.figure(figsize=(8, 8))
ax = plt.axes(projection="3d")

# Barplot creating vertical lines
dx, dy = 0.05, 0.05      # dx = dy, dimensions of bar
dz = z                     # dz = height of bar
ax.bar3d(x, y, [0],       # Position of base
          dx, dy, dz,      # bar Dimensions
          color='grey', alpha=0.5,
          linestyle='--')

# 3D scatterplot
ax.scatter(x, y, z, alpha=1, linewidth=2)

# Set titles
ax.set_xlabel('Years of Education')
ax.set_ylabel('Years of Experience')
ax.set_zlabel('Income')

# Show plot
plt.show()
```

Contrary to the simple linear regression model, in a three-dimensional setting, with two predictors and one response, the least squares regression line becomes a *plane* which fits best the data points (see Figure 7.3).

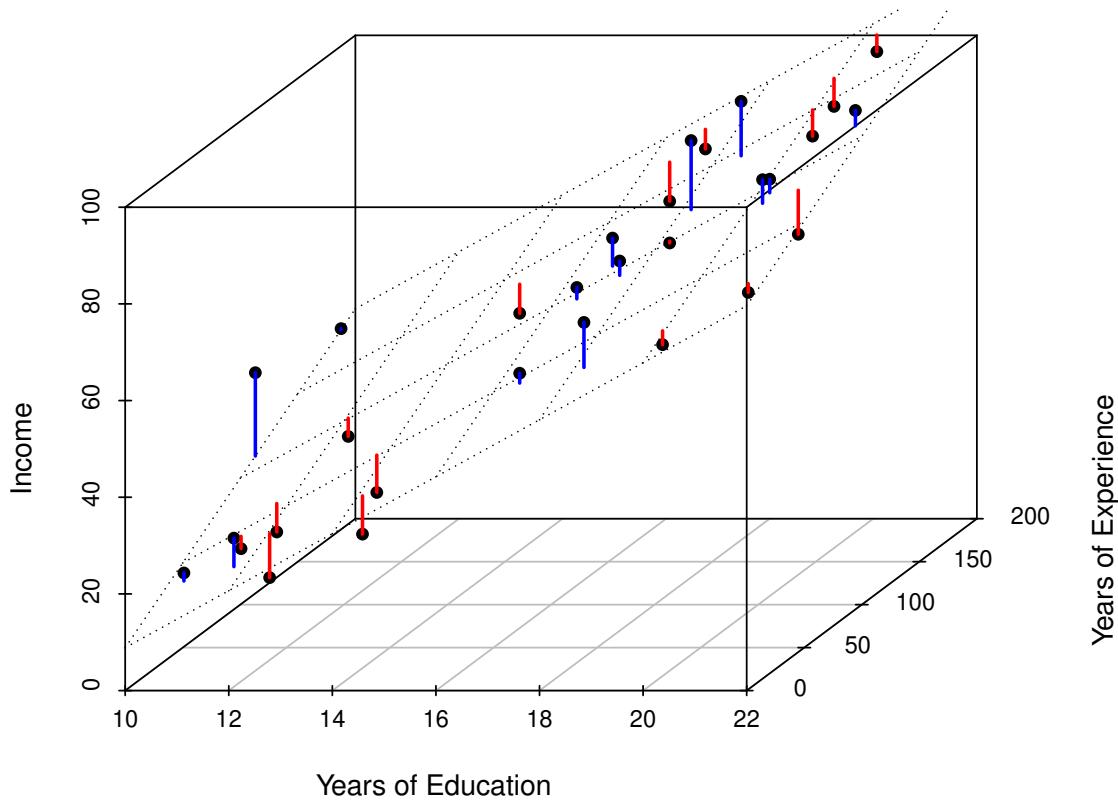
The plane is chosen to minimize the sum of the squared vertical distances between each observation and the plane. These vertical distances correspond to the residuals. Figure 7.3 displays blue segments for points that lie above the plane, and red segments for points that lie below the plane.

The parameters are estimated using the same least squares approach that we saw in the context of simple linear regression. With the `.params` attribute of a fitted model we can determine the regression coefficients  $\beta_0, \beta_1$  and  $\beta_2$  for the `Income` data set:

$$\hat{\beta}_0 = -50.086; \quad \hat{\beta}_1 = 5.896; \quad \hat{\beta}_2 = 0.173$$

([to R](#))

## Chapter 7. Multiple Linear Regression



**Figure 7.3.:** In a three-dimensional setting with two predictors, `experience` and `education`, and one response `income`, the least squares regression line becomes a plane. The plane is chosen to minimize the sum of squared vertical distances between each observation and the plane (shown in red and blue).

```
[2]: import statsmodels.api as sm

# Fit Linear Model
x_sm = df[['education', 'experience']]
x_sm = sm.add_constant(x_sm)
model = sm.OLS(z, x_sm).fit()

# Print Model Parameters
print(model.params)
```

const	-50.085639
education	5.895556
experience	0.172855

We thus have the following multiple regression model for the `Income` data set

$$\text{income} \approx -50.086 + 5.896 \cdot \text{education} + 0.173 \cdot \text{experience}$$



In the following section, we will generalize the regression model with  $p = 2$  predictors to an arbitrary number of predictors.

## 7.2. Estimating the Regression Coefficients

As was the case in the simple linear regression setting, the regression coefficients  $\beta_0, \beta_1, \dots, \beta_p$  are in general unknown, and must be estimated. Given estimates  $\hat{\beta}_0, \hat{\beta}_1, \dots, \hat{\beta}_p$ , we can make predictions using the formula

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \dots + \dots + \hat{\beta}_p x_p$$

The parameters are estimated using the same least squares approach that we saw in the context of simple linear regression.

We choose  $\beta_0, \beta_1 \dots, \beta_p$  to minimize the sum of squared residuals

$$\begin{aligned} \text{RSS} &= \sum_{i=1}^n r_i^2 \\ &= \sum_{i=1}^n (y_i - \hat{y}_i)^2 \\ &= \sum_{i=1}^n (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_{i1} - \hat{\beta}_2 x_{i2} - \dots - \hat{\beta}_p x_{ip})^2 \end{aligned}$$

where  $x_{ij}$  is the  $i$ -th observation for the  $j$ th predictor.

The values  $\hat{\beta}_0, \hat{\beta}_1, \dots, \hat{\beta}_p$  that minimize the RSS are the multiple least squares regression coefficient estimates. Unlike the simple linear regression estimates given in Section 5.3.1, the multiple regression coefficient estimates have somewhat complicated forms that are most easily represented using matrix algebra. For this reason we do not provide them here. Any statistical software package can be used to compute these coefficient estimates; in the following example we will show how this can be done in [Python](#).

## Chapter 7. Multiple Linear Regression

### Example 7.2.1

In [Python](#) we estimate the coefficients for the linear regression model of the example 7.1.2 for the **Advertising** data set, as follows

(to R)

```
[1]: import pandas as pd
      import statsmodels.api as sm

      # Load data
      df = pd.read_csv(
          '../.../Themen/Einfache_Lineare_Regression/Daten/Advertising.csv')
      x = df[['TV', 'radio', 'newspaper']]
      y = df['sales']

      # Fit Model:
      x_sm = sm.add_constant(x)
      model = sm.OLS(y, x_sm).fit()

      # Print Model parameters
      print(model.params)
```

const	2.938889
TV	0.045765
radio	0.188530
newspaper	-0.001037



We interpret these results as follows: for a given amount of **TV** and **newspaper** advertising, spending an additional CHF 1000 on **radio** advertising leads to an increase in **sales** by approximately 189 units.

Table 7.2 displays the multiple regression coefficient estimates when **TV**, **radio**, and **newspaper** are used to predict **sales**. We find these values by using the [Python](#) method `.summary()` of the fitted model.

	Coefficient	Std.error	t-statistic	p-value
Intercept	2.939	0.3119	9.42	< 0.0001
TV	0.046	0.0014	32.81	< 0.0001
Radio	0.189	0.0086	21.89	< 0.0001
Newspaper	-0.001	0.0059	-0.18	0.8599

**Table 7.2.:** For the **Advertising** data, least squares coefficient estimates of the multiple linear regression of number of units sold on **radio**, **TV**, and **newspaper** advertising budgets.

## Chapter 7. Multiple Linear Regression

Comparing these coefficient estimates to those displayed in Table 7.1, we notice that the multiple regression coefficient estimates for **TV** and **radio** are pretty similar to the simple linear regression coefficient estimates.

However, while the **newspaper** regression coefficient estimate  $\hat{\beta}_1$  in Table 7.1 was significantly non-zero, the coefficient estimate for **newspaper** in the multiple regression model is close to zero, and the corresponding p-value is no longer significant, with a value around 0.86. This illustrates that the simple and multiple regression coefficients can be quite different.

This difference stems from the fact that in the simple regression case, the slope term represents the average effect of a CHF 1000 increase in **newspaper** advertising, ignoring other predictors such as **TV** and **radio**. In contrast, in the multiple regression setting, the coefficient for **newspaper** represents the average effect of increasing **newspaper** spending by CHF 1000 while holding **TV** and **radio** fixed.

Does it make sense for the multiple regression to suggest no relationship between **sales** and **newspaper** while the simple linear regression implies the opposite? In fact it does. Consider the correlation matrix for the three predictor variables and response variable, displayed in Table 7.3.

	<b>TV</b>	<b>Radio</b>	<b>Newspaper</b>	<b>Sales</b>
<b>TV</b>	1.0000	0.0548	0.0567	0.7822
<b>Radio</b>		1.0000	0.3541	0.5762
<b>Newspaper</b>			1.0000	0.2283
<b>Sales</b>				1.0000

**Table 7.3.:** Correlation matrix for **TV**, **radio**, **newspaper**, and **sales** for the **Advertising** data.

(to R)

```
[2]: # Save the Sales data in a fitting dataframe:  
df = df[['TV', 'radio', 'newspaper', 'sales']]  
  
# Print the correlation coefficients  
print(df.corr())
```

	<b>TV</b>	<b>radio</b>	<b>newspaper</b>	<b>sales</b>
<b>TV</b>	1.000000	0.054809	0.056648	0.782224
<b>radio</b>	0.054809	1.000000	0.354104	0.576223
<b>newspaper</b>	0.056648	0.354104	1.000000	0.228299
<b>sales</b>	0.782224	0.576223	0.228299	1.000000

Notice that the correlation between **radio** and **newspaper** is 0.35. This reveals a tendency to spend more on **newspaper** advertising in markets where more is spent on **radio** advertising. Now suppose that the multiple regression is correct and **newspaper** advertising has no direct impact on **sales**, but radio advertising does

increase **sales**. Then in markets where we spend more on **radio** our **sales** will tend to be higher, and as our correlation matrix shows, we also tend to spend more on **newspaper** advertising in those same markets. Hence, in a simple linear regression which only examines **sales** versus **newspaper**, we will observe that higher values of **newspaper** tend to be associated with higher values of **sales**, even though **newspaper** advertising does not actually affect **sales**. So **newspaper** sales are a surrogate for **radio** advertising; **newspaper** gets “credit” for the effect of **radio** on **sales**.

This slightly counterintuitive result is very common in many real life situations. Consider an absurd example to illustrate the point.

### Example 7.2.2

Running a regression of shark attacks versus ice cream sales for data collected at a given beach community over a period of time would show a positive relationship, similar to that seen between **sales** and **newspaper**. Of course no one (yet) has suggested that ice creams should be banned at beaches to reduce shark attacks. In reality, higher temperatures cause more people to visit the beach, which in turn results in more ice cream sales and more shark attacks. A multiple regression of attacks versus ice cream sales *and* temperature reveals that, as intuition implies, the former predictor is no longer significant after adjusting for temperature.



## 7.3. Some Important Questions

When we perform multiple regression, we usually are interested in answering a few important questions:

1. *Is at least one of the predictors  $X_1, X_2, \dots, X_p$  useful in predicting the response?*
2. *Do all the predictors help to explain  $Y$ , or is only a subset of the predictors useful?*
3. *How well does the model fit the data?*
4. *Given a set of predictor values, what response value should we predict, and how accurate is our prediction?*

We now address each of these questions in turn.

### 7.3.1. Is There a Relationship Between the Response and Predictors?

Recall that in Section 5.3.2 on page 116, that is in the simple linear regression setting, in order to determine whether there is a relationship between the response variable  $Y$  and the predictor  $X$  we can simply check whether  $\beta_1 = 0$ . If  $\beta_1 = 0$ , then there is no relationship, otherwise we would conclude, that there is a relationship between predictor and response variable. In the multiple regression setting with  $p$  predictors, we need to ask whether *all* of the regression coefficients are zero - with the exception of  $\beta_0$  - i.e. whether

$$\beta_1 = \beta_2 = \dots = \beta_p = 0$$

As in the simple linear regression setting, we use a hypothesis test to answer this question. We test the null hypothesis,

$$H_0 : \beta_1 = \beta_2 = \dots = \beta_p = 0$$

versus the alternative

$$H_A : \text{at least one } \beta_i \text{ is non-zero}$$

This hypothesis test is performed by computing the *F-statistic*

$$F = \frac{(TSS - RSS)/p}{RSS/(n - p - 1)}$$

where, as with simple linear regression,

$$TSS = \sum_{i=1}^n (y_i - \bar{y})^2 \quad \text{and} \quad RSS = \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

If the linear model assumptions are correct, one can show that

$$E \left[ \frac{RSS}{n - p - 1} \right] = \sigma^2$$

and that, provided  $H_0$  is true

$$E \left[ \frac{TSS - RSS}{p} \right] = \sigma^2$$

Hence, when there is no relationship between the response and the predictors, one would expect the *F*-statistic to take on a value close to 1. On the other hand, if  $H_A$  is true, then

$$E \left[ \frac{TSS - RSS}{p} \right] > \sigma^2$$

and we expect *F* to be greater than 1.

## Chapter 7. Multiple Linear Regression

### Example 7.3.1

The  $F$ -statistic for the multiple linear regression model in the **Advertising** example is obtained by regressing **sales** onto **radio**, **TV**, and **newspaper** and is in this example 570. In the Python-output we find the value of the  $F$ -statistic under **F-statistic**.

(to R)

```
[1]: import pandas as pd
      import statsmodels.api as sm

      # Load data
      df = pd.read_csv(
          '../../Themen/Einfache_Lineare_Regression/Daten/Advertising.csv')
      x = df[['TV', 'radio', 'newspaper']]
      y = df['sales']

      # Fit Model:
      x_sm = sm.add_constant(x)
      model = sm.OLS(y, x_sm).fit()

      # Print summary including F-Statistic
      print(model.summary())
```

```
OLS Regression Results
=====
Dep. Variable:           sales    R-squared:         0.897
Model:                 OLS     Adj. R-squared:      0.896
Method:                Least Squares   F-statistic:       570.3
Date:      Wed, 06 Jan 2021   Prob (F-statistic): 1.58e-96
Time:          16:30:12        Log-Likelihood:   -386.18
No. Observations:      200        AIC:             780.4
Df Residuals:          196        BIC:             793.6
Df Model:                  3
Covariance Type:    nonrobust
=====
            coef    std err        t      P>|t|      [0.025      0.975]
-----
const      2.9389    0.312     9.422      0.000      2.324      3.554
TV         0.0458    0.001    32.809      0.000      0.043      0.049
radio      0.1885    0.009    21.893      0.000      0.172      0.206
newspaper   -0.0010    0.006    -0.177      0.860     -0.013      0.011
=====
Omnibus:           60.414   Durbin-Watson:      2.084
Prob(Omnibus):      0.000    Jarque-Bera (JB): 151.241
Skew:              -1.327    Prob(JB):       1.44e-33
Kurtosis:           6.332    Cond. No.:        454.
=====
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

Since this is far larger than 1, it provides compelling evidence against the null hypothesis  $H_0$ . In other words, the large  $F$ -statistic suggests that at least one of the advertising media must be related to **sales**. 

However, what if the  $F$ -statistics had been closer to 1? How large does the  $F$ -statistic need to be before we can reject  $H_0$  and conclude that there is a relationship? It turns out that the answer depends on the values of  $n$  and  $p$ . When  $n$  is large, an  $F$ -statistic that is just a little larger than 1 might still provide evidence against  $H_0$ . In contrast, a larger  $F$ -statistic is needed to reject  $H_0$  if  $n$  is small. When  $H_0$  is true and the errors  $\varepsilon_i$  have a normal distribution, the  $F$ -statistic follows an  $F$ -distribution with  $p$  and  $n - p - 1$  degrees of freedom. For any given value of  $n$  and  $p$ , [Python](#) can be used to compute the p-value associated with the  $F$ -statistic using this distribution; that is the probability of observing a more extreme value of the  $F$ -statistic than the one observed. Based on this p-value, we can determine whether or not to reject  $H_0$ . The procedure is the same as in linear regression where we however used the  $t$ -statistic.

### Example 7.3.2

For the [Advertising](#) data, the p-value associated with the  $F$ -statistic in the [Python](#)-output of example 7.3.1 (displayed under  $P > |t|$ ) is essentially zero, so we have extremely strong evidence that at least one of the media is associated with increased [sales](#). ◀

Given these individual p-values for each variable, why do we need to look at the overall  $F$ -statistic? After all, it seems likely that if any one of the p-values for the individual variables is very small, then *at least one of the predictors is related to the response*. However, this logic is flawed, especially when the number of predictors  $p$  is large.

For instance, consider an example in which  $p = 100$  and

$$H_0 : \beta_1 = \beta_2 = \dots = \beta_{100} = 0$$

is true, so no variable is truly associated with the response. In this situation, about 5 % of the p-values associated with each variable (of the type shown in Table 7.2) will be below 0.05 by chance. In other words, we expect to see approximately five *small* p-values even in the absence of any true association between the predictors and the response. In fact, we are almost guaranteed that we will observe at least one p-value below 0.05 by chance. Hence, if we use the individual t-statistics and associated p-values in order to decide whether or not there is any association between the variables and the response, there is a very high chance that we will incorrectly conclude that there is a relationship.

However, the  $F$ -statistic does not suffer from this problem because it adjusts for the number of predictors. Hence, if  $H_0$  is true, there is only a 5 % chance that the  $F$ -statistics will result in a p-value below 0.05, regardless of the number of predictors or the number of observations.

The approach of using an  $F$ -statistic to test for any association between the predictors and the response works when  $p$  is relatively small, and certainly small compared to

$n$ . However, sometimes we have a very large number of variables. If  $p > n$  then there are more coefficients  $\beta_j$  to estimate than observations from which to estimate them. In this case we cannot even fit the multiple linear regression model using least squares, so the  $F$ -statistic cannot be used, and neither can most of the other concepts that we have seen so far in this chapter. When  $p$  is large, some of the approaches discussed in the next section, such as *forward selection*, can be used.

### 7.3.2. Deciding on Important Variables

As discussed in the previous section, the first step in a multiple regression analysis is to compute the  $F$ -statistic and to examine the associated p-value. If we conclude on the basis of that p-value that at least one of the predictors is related to the response, then it is natural to wonder *which* are the guilty ones. We could look at the individual p-values as in Table 7.2 on page 181, but as discussed, if  $p$  is large we are likely to make some false discoveries.

It is possible that all of the predictors are associated with the response, but it is more often the case that the response is only related to a subset of the predictors. The task of determining which predictors are associated with the response, in order to fit a single model involving only those predictors, is referred to as *variable selection*. The variable selection problem is studied in Chapter 8 on page 234.

### 7.3.3. How well does the Model fit the Data?

Two of the most common numerical measures of model fit are the RSE and  $R^2$ , the fraction of variance explained (see Section 6.2.1 on page 136). These quantities are computed and interpreted in the same fashion as for simple linear regression.

Recall that in simple regression,  $R^2$  is the square of the correlation of the response and the predictor variable. In multiple linear regression, it turns our that it equals  $\text{Cor}[Y, \hat{Y}]^2$ , the square of the correlation between the response and the predicted response. In fact one property of the fitted linear model is that it maximizes this correlation among all possible linear models.

An  $R^2$  value close to 1 indicates that the model explains a large portion of the variance in the response variable.

#### Example 7.3.3

The Python-output we have seen in example 7.3.1 for the **Advertising** data displayed an  $R^2$  value of 0.89720. On the other hand, the model that uses only **TV** and **radio** to predict **sales** has an  $R^2$  value of 0.89719. In other words, there is a *small*

increase in  $R^2$  if we include **newspaper** advertising in the model that already contains **TV** and **radio** advertising, even though we saw earlier that the p-value for **newspaper** advertising in Table 7.2 is not significant. It turns out that  $R^2$  will always increase when more variables are added to the model, even if those variables are only weakly associated with the response. This is due to the fact that adding another variable to the least squares equations must allow us to fit the training data (though not necessarily the testing data) more accurately. Thus, the  $R^2$  statistic which is also computed on the training data, must increase.

The fact that adding **newspaper** to the model containing only **TV** and **radio** advertising leads to just a tiny increase in  $R^2$  provides additional evidence that **newspaper** can be dropped from the model. Essentially, **newspaper** provides no real improvement in the model fit to the training samples, and its inclusion will likely lead to poor results on independent test samples due to overfitting.

In contrast, the model containing only **TV** as a predictor had an  $R^2$  of 0.61. Adding **radio** to the model leads to a substantial improvement in  $R^2$  to 0.89719. This implies that a model that uses **TV** and **radio** expenditures to predict **sales** is substantially better than one that uses only **TV** advertising. We could further quantify this improvement by looking at the p-value for the **radio** coefficient in a model that contains only **TV** and **radio** as predictors.

The model that contains only **TV** and **radio** as predictors has an RSE of 1.681, and the model that also contains **newspaper** as a predictor has an RSE of 1.686 (see example 7.3.1). In contrast the model that contains only **TV** has an RSE of 3.26 (see example 6.2.1). This corroborates our previous conclusion that a model that uses **TV** and **radio** expenditures to predict **sales** is much more accurate (on the training data) than one that only uses **TV** spending. Furthermore, given that **TV** and **radio** expenditures are used as predictors, there is no point in also using **newspaper** spending as a predictor in the model. The observant reader may wonder how RSE can increase when **newspaper** is added to the model given that RSS must decrease. In general, RSE is defined as

$$\text{RSE} = \sqrt{\frac{1}{n - p - 1} \text{RSS}}$$

which simplifies to

$$\text{RSE} = \sqrt{\frac{1}{n - 2} \text{RSS}}$$

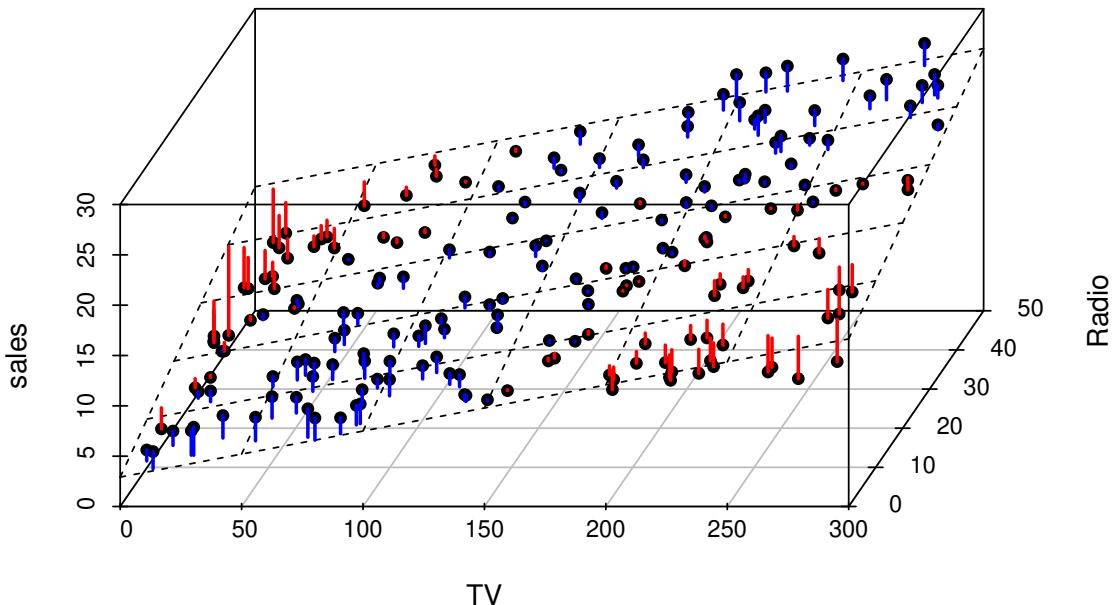
for a simple linear regression. Thus, models with more variables can have higher RSE if the decrease in RSS is small relative to the increase in  $p$ . ◀

In addition to looking at the RSE and  $R^2$  statistics just discussed, it can be useful to plot the data. Graphical summaries can reveal problems with a model that are not visible from numerical statistics.

**Example 7.3.4**

Figure 7.4 displays a three-dimensional plot of `TV` and `radio` versus `sales` where the plane is plotted with dashed lines. We see that some observations lie above and some observations lie below the least squares regression plane. In particular, the linear model seems to overestimate `sales` for instances in which most of the advertising money was spent exclusively on either `TV` or `radio`. It underestimates `sales` for instances where the budget was split between the two media.

This pronounced non-linear pattern cannot be modeled accurately using linear regression. It suggests a *synergy* or *interaction* effect between the advertising media, whereby combining the media together results in a bigger boost to `sales` than using any single medium.



**Figure 7.4.:** For the `Advertising` data, a linear regression fit to `sales` using `TV` and `radio` as predictors. From the pattern of the residuals, we can see that there is a pronounced non-linear relationship in the data. The positive residuals (those shown in blue above the surface), tend to lie along the 45-degree line, where `TV` and `radio` budgets are split evenly. The negative residuals (shown in red), tend to lie away from this line, where budgets are more lopsided.

### 7.3.4. Predictions

Once we have fit the multiple regression model, it is straightforward to apply it to predict the response  $Y$  on the basis of a set of values for the predictors  $X_1, X_2, \dots, X_p$ . However, there are three sorts of uncertainty associated with this prediction.

1. The coefficient estimates  $\hat{\beta}_1, \dots, \hat{\beta}_p$  are estimates for  $\beta_1, \dots, \beta_p$ . That is, the *least squares plane*

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 X_1 + \dots + \hat{\beta}_p X_p$$

is only a least squares estimate for the *true population regression plane*

$$f(X_1, \dots, X_p) = \beta_0 + \beta_1 X_1 + \dots + \beta_p X_p$$

The inaccuracy in the coefficient estimates is related to the *reducible error* from chapter 5. We can compute a *confidence interval* in order to determine how close  $\hat{Y}$  will be to  $f(X_1, \dots, X_p)$ .

2. Of course, in practice assuming a linear model for  $f(X_1, \dots, X_p)$  is almost always an approximation of reality, so there is an additional source of potentially reducible error which we call *model bias*.

So when we use a linear model, we are in fact estimating the best linear approximation to the true surface. However, here we will ignore this discrepancy, and operate as if the linear model were correct.

3. Even if we knew  $f(X_1, \dots, X_p)$  - that is, even if we knew the true values for  $\beta_1, \dots, \beta_p$  - the response value cannot be predicted perfectly because of the random error  $\varepsilon$ . In Chapter 5, we referred to this as the *irreducible error*.

How much will  $Y$  vary from  $\hat{Y}$ ? We use *prediction intervals* to answer this question. Prediction intervals are always wider than confidence intervals, because they incorporate both the error in the estimate for  $f(X_1, \dots, X_p)$  (the reducible error) and the uncertainty as to how much an individual point will differ from the population regression plane (the irreducible error).

#### Example 7.3.5

We use a *confidence interval* to quantify the uncertainty surrounding the *average sales* over a large number of cities. We restrict ourselves to the regression of **sales** on **TV** and **radio** since **newspaper** can be neglected as followed from the previous discussion.

For example, given that CHF 100 000 is spent on **TV** advertising and CHF 20 000 is spent on **radio** advertising in each city, the 95 % confidence interval is

$$[10'985, 11'528]$$

## Chapter 7. Multiple Linear Regression

(to R)

```
[1]: import pandas as pd
      import statsmodels.api as sm

      # Load data
      df = pd.read_csv(
          '../.../Themen/Einfache_Lineare_Regression/Daten/Advertising.csv')
      x = df[['TV', 'radio']]
      y = df['sales']

      # Fit Model:
      x_sm = sm.add_constant(x)
      model = sm.OLS(y, x_sm).fit()

      # Get prediction and confidence interval at x = [100, 20]
      x0 = [[100, 20]]
      x0 = sm.add_constant(x0, has_constant='add')

      predictionsx0 = model.get_prediction(x0)
      predictionsx0 = predictionsx0.summary_frame(alpha=0.05)

      # Print the results. mean_ci_ corresponds to the confidence interval
      # whereas obs_ci corresponds to the prediction interval
      predictionsx0
```

mean	mean_se	mean_ci_lower	mean_ci_upper	obs_ci_lower
→	obs_ci_upper			↙
11.26	0.1375	10.99	11.53	7.930
				14.58

We interpret this to mean that 95 % of intervals of this form will contain the true value of  $f(X_1, X_2)$ . In other words, if we collect a large number of data sets like the **Advertising** data set, and we construct a confidence interval for the average **sales** on the basis of each data set - given CHF 100 000 in **TV** and CHF 20 000 in **radio** advertising - then 95 % of these confidence intervals will contain the true value of average **sales**.

On the other hand, a *prediction interval* can be used to quantify the uncertainty surrounding **sales** for a *particular* city. Given that CHF 100 000 is spent on **TV** and CHF 20 000 is spent on **radio** advertising in that city the 95 % *prediction interval* is

$$[7'930, 14'583]$$

We interpret this to mean that 95 % of intervals of this form will contain the true value of  $Y$  for this city.

Note that both intervals are centered at 11 256, but that the prediction interval is substantially wider than the confidence interval, reflecting the increased uncertainty

about `sales` for a given city in comparison to the average `sales` over many locations.



## 7.4. Variety of Regression Modeling

### 7.4.1. Omitting Predictor Variables

In section 7.3.3 we came to the conclusion that not all of the predictor variables may be relevant for predicting the value of the response variable. We now may ask whether leaving out one of the predictors significantly lowers the degree to which the model fits the data. To answer this question we need to compare two regression models  $\mathcal{M}_1$  and  $\mathcal{M}_2$  with each other. An *analysis of variance* allows us to test the null hypothesis that a small model  $\mathcal{M}_1$  is sufficient to explain the data against the alternative hypothesis that a more complex model  $\mathcal{M}_2$  is required. This test is called *ANOVA-test* and is based on an F-test providing us with a p-value.

In chapter 7.3.1 we were testing  $H_0$  that *all* the coefficients  $\beta_j$  are zero. We were investigating this question with the help of the *F*-statistic. Sometimes we want to test that a particular subset of  $q$  of the coefficients are zero. This corresponds to a null hypothesis

$$H_0 : \beta_{p-q+1} = \beta_{p-q+2} = \dots = \beta_p = 0$$

where for convenience we have put the variables chosen for omission at the end of the list. In this case we fit a (small) model  $\mathcal{M}_1$  that uses all the predictor variables *except* those last  $q$ . Suppose that the residual sum of squares for that model is  $\text{RSS}_0$ . Then the appropriate *F*-statistic is

$$F = \frac{(\text{RSS}_0 - \text{RSS})/q}{\text{RSS}/(n - p - 1)}$$

If the null hypothesis is true, then the value of the *F*-statistic is approximately 1, and it follows an *F*-distribution with  $q$  and  $n - p - 1$  degrees of freedom :  $F \sim \mathcal{F}_{q,n-p-1}$ . If however the alternative hypothesis is true, then the *F*-statistic is much larger than 1.

### Example 7.4.1

For the **Advertising** data we had the following multiple linear regression model

$$\mathbf{sales} = \beta_0 + \beta_1 \cdot \mathbf{TV} + \beta_2 \cdot \mathbf{radio} + \beta_3 \cdot \mathbf{newspaper} + \varepsilon \quad (7.1)$$

For instance, as we discussed earlier, the p-values associated with this model are displayed in Table 7.2 and indicate that **TV** and **radio** are related to **sales**, but that there is no evidence that **newspaper** is associated with **sales**, in the presence of these two.

We now compare the “large” model  $\mathcal{M}_2$  defined by equation 7.1 with the “small” model  $\mathcal{M}_1$  (without **newspaper**)

$$\mathbf{sales} = \beta_0 + \beta_1 \cdot \mathbf{TV} + \beta_2 \cdot \mathbf{radio} + \varepsilon$$

We use the `anova_lm()` method function, which performs an *analysis of variance* (ANOVA, using an F-test) in order to test the null hypothesis that the small model  $\mathcal{M}_1$  is sufficient to explain the data against the alternative hypothesis that the (more complex) model  $\mathcal{M}_2$  is required. In order to use the `anova_lm()` method function,  $\mathcal{M}_1$  and  $\mathcal{M}_2$  must be *nested* models: the predictors in  $\mathcal{M}_1$  must be a subset of the predictors in  $\mathcal{M}_2$ . This corresponds to the null hypothesis  $\beta_3 = 0$ , that is, that there is no relationship between **newspaper** and **sales**.

The Python-output provides us with the information that the residual sum of squares (**RSS**) in the “small” model is given by

$$\text{RSS}_0 = 556.91$$

whereas the residual sum of squares for the “large” model  $\mathcal{M}_2$  is

$$\text{RSS} = 556.83$$

(to R)

```
[1]: import pandas as pd
import statsmodels.api as sm

# Load data
df = pd.read_csv(
    '.../Themen/Einfache_Lineare_Regression/Daten/Advertising.csv')
x1 = df[['TV', 'radio']]
x2 = df[['TV', 'radio', 'newspaper']]
y = df['sales']

# Fit model
x1_sm = sm.add_constant(x1)
x2_sm = sm.add_constant(x2)
model1 = sm.OLS(y, x1_sm).fit()
```

## Chapter 7. Multiple Linear Regression

```
model2 = sm.OLS(y, x2_sm).fit()

# Table and print results
table = sm.stats.anova_lm(model1, model2)
print(table)
```

	df_resid	ssr	df_diff	ss_diff	F	Pr (>F)
0	197.0	556.913980	0.0	NaN	NaN	NaN
1	196.0	556.825263	1.0	0.088717	0.031228	0.859915

The difference between RSS and  $RSS_0$  can be found in the [Python](#)-output under `ss_diff` and is 0.088717. The value of  $q$  is displayed under `df` and is given here by 1. For the “large” model, we have

$$n - p - 1 = 200 - 3 - 1 = 196$$

degrees of freedom (`df_resid`), contrary to the “small” model that has

$$n - p - 1 = 200 - 2 - 1 = 197$$

degrees of freedom. Thus, the value of the F-statistic is (`F`)

$$\begin{aligned} F &= \frac{(RSS_0 - RSS)/q}{RSS/(n - p - 1)} \\ &= \frac{(556.91 - 556.83)/1}{556.83/(200 - 3 - 1)} \\ &= \frac{0.088717}{556.83/196} \\ &= 0.0312 \end{aligned}$$

The one-sided p-value in upwards direction for the  $F$ -statistic assuming the null hypothesis is true, that is  $\beta_3 = 0$ , is displayed in the [Python](#)-output under `Pr (>F)` : 0.8599.

Since this p-value is significantly larger than the significance level  $\alpha = 0.05$ , there is no evidence to reject the null hypothesis. We conclude that the predictor `newspaper` is redundant, and we therefore can omit it.



### Example 7.4.2

Notice that in Table 7.2 for the `Advertising` data, for each individual predictor a t-statistic and a p-value were reported. These provide information about whether each individual predictor is related to the response, after adjusting for the other predictors.

## Chapter 7. Multiple Linear Regression

It turns out that each of these are exactly equivalent to the F-test that omits that single variable from the model, leaving all the others in - i.e.  $q = 1$  in

$$F = \frac{(\text{RSS}_0 - \text{RSS})/q}{\text{RSS}/(n - p - 1)}$$

That is, in the “small” model that single variable is omitted, whereas in the “large” model all variables are kept.

So the F-test reports the *partial effect* of adding that variable to the model. For instance, as we discussed earlier, these p-values indicate that **TV** and **radio** are related to **sales**, but that there is no evidence that **newspaper** is associated with **sales**, in the presence of these two.



### Example 7.4.3

If we compare the “large” model  $\mathcal{M}_2$  with the “small” model (**TV** is omitted)  $\mathcal{M}_1$

$$\text{sales} = \beta_0 + \beta_1 \cdot \text{radio} + \beta_2 \cdot \text{newspaper} + \varepsilon$$

then we come to a very different conclusion:

(to R)

```
[2]: # This Programm follows Example 4.1
# Load data
x3 = df[['radio', 'newspaper']]

# Fit model
x3_sm = sm.add_constant(x3)
model3 = sm.OLS(y, x3_sm).fit()

# Table and print results
table = sm.stats.anova_lm(model3, model12)
print(table)
```

	df_resid	ssr	df_diff	ss_diff	F	Pr (>F)
0	197.0	3614.835279	0.0	NaN	NaN	NaN
1	196.0	556.825263	1.0	3058.010016	1076.405837	1.509960e-81

In this case the p-value is approximately zero, hence we have to reject the null hypothesis  $\beta_1 = 0$ . There is a significant difference in how well the two models  $\mathcal{M}_1$  and  $\mathcal{M}_2$  fit the data. Omitting **TV** leads to a model that shows a significant deterioration with respect to the quality of the model.

## Chapter 7. Multiple Linear Regression

In order to get an "overview" about how the quality of a model changes when one predictor variable is omitted, we can use the `anova_lm()` method on the one model only. However, this only works, when the model is defined using a formula instead of columns of data.

(to R)

```
[3] : import statsmodels.formula.api as smf

# Load data
TV = df[['TV']]
radio = df[['radio']]
newspaper = df[['newspaper']]
sales = df['sales']

# Fit model using formula:
model_f = smf.ols(formula='sales ~ TV + radio + newspaper', data=df).fit()

# Table and print results
table_f = sm.stats.anova_lm(model_f, typ=2)
print(table_f)
```

	sum_sq	df	F	PR (>F)
TV	3058.010016	1.0	1076.405837	1.509960e-81
radio	1361.736549	1.0	479.325170	1.505339e-54
newspaper	0.088717	1.0	0.031228	8.599151e-01
Residual	556.825263	196.0	NaN	NaN

We will discuss the statistic `AIC` in chapter 8. 

In chapter 8 we will discuss different criteria and statistics to decide *which* predictor variables are important.

### 7.4.2. Qualitative Predictors

In our discussion so far, we have assumed that all variables in our linear regression model are *quantitative*. But in practice, this is not necessarily the case; often some predictors are *qualitative*.

#### Example 7.4.4

For example, the `Credit` data set displayed in Figure 7.5 records `balance` (average credit card debt for a number of individuals) as well as several quantitative predictors: `age`, `cards` (number of credit cards), `education` (years of education), `income` (in thousand of dollars), `limit` (credit limit), and `rating` (credit rating).

## Chapter 7. Multiple Linear Regression

Each panel of Figure 7.5 is a scatterplot for a pair of variables whose identities are given by the corresponding row and column labels. For example, the scatterplot directly to the right of the word “Balance” depicts **balance** versus **age**, while the plot directly to the right of “Age” corresponds to **age** versus **cards**.

(to R)

```
[1]: import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Load data
df = pd.read_csv(
    '.../.../Themen/Einfache_Lineare_Regression/Daten/Credit.csv')

# Drop Qualitative terms / take only quantitative terms
df = df[['Balance', 'Age', 'Cards', 'Education',
          'Income', 'Limit', 'Rating']]

# Plot using sns.pairplot
sns.pairplot(df)
plt.show()
```

In addition to these quantitative variables, we also have four qualitative variables : **gender**, **student** (student status), **status** (marital status) and **ethnicity** (Caucasian, African American or Asian).



### Predictors with Only Two Levels

Suppose that we wish to investigate differences in credit card balance between males and females, ignoring the other variables for the moment. If a qualitative predictor (also known as *factor*) only has two *levels*, or possible values, then incorporating it into a regression model is very simple.

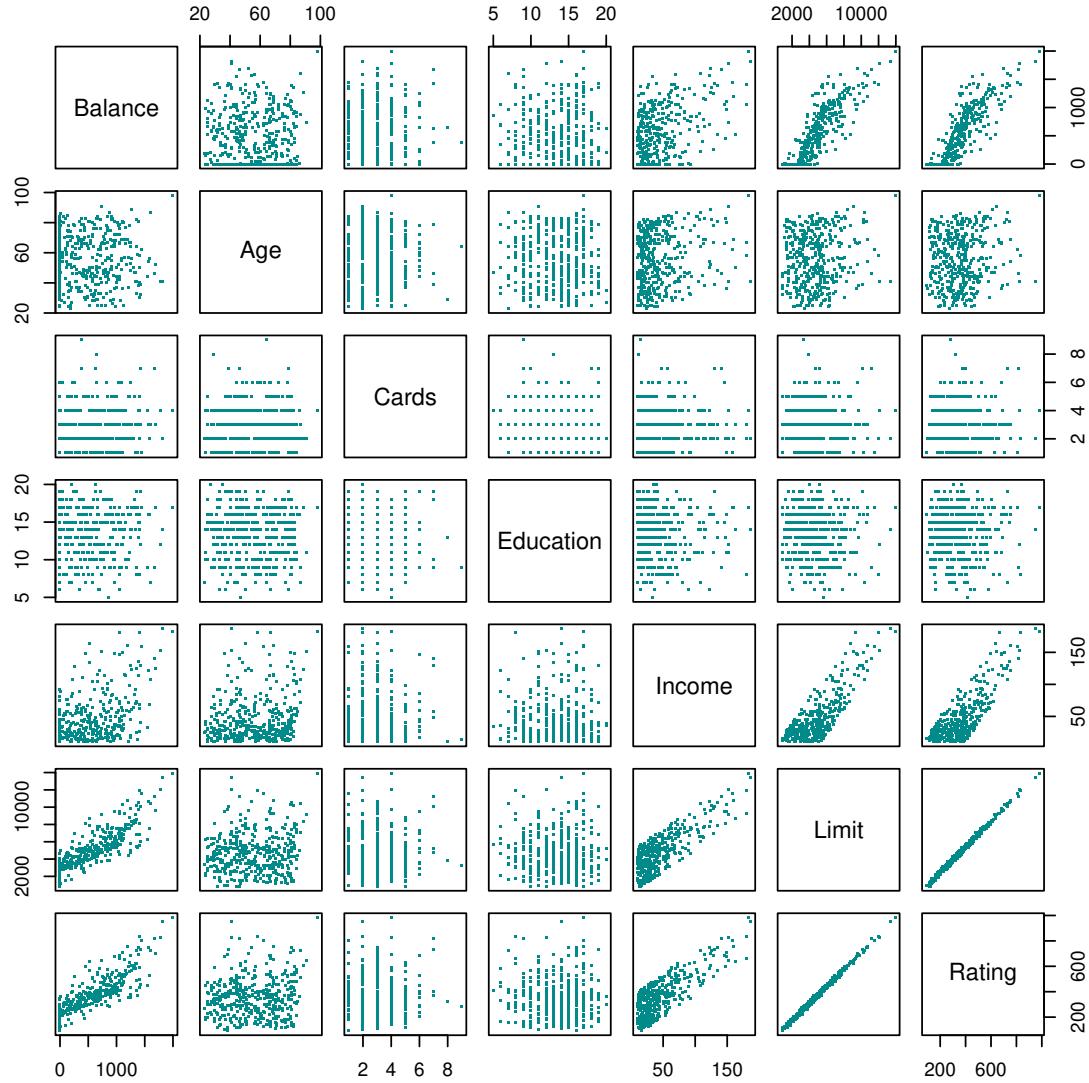
We simply create an indicator or *dummy variable* that takes on two possible numerical values.

#### Example 7.4.5

For example, based on the **gender** variable, we can create a new variable that takes the form

$$x_i = \begin{cases} 1 & \text{if } i\text{th person is female} \\ 0 & \text{if } i\text{th person is male} \end{cases}$$

## Chapter 7. Multiple Linear Regression



**Figure 7.5.:** The `Credit` data set contains information about `balance`, `age`, `cards`, `income`, `limit`, and `rating` for a number of potential customers.

and use this variable as a predictor in the regression equation. This results in the model

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i = \begin{cases} \beta_0 + \beta_1 + \varepsilon_i & \text{if } i\text{th person is female} \\ \beta_0 + \varepsilon_i & \text{if } i\text{th person is male} \end{cases} \quad (7.2)$$

Now,  $\beta_0$  can be interpreted as the average credit card balance among males,  $\beta_0 + \beta_1$  as the average credit card balance among females, and  $\beta_1$  as the average difference in credit card balance between females and males.

## Chapter 7. Multiple Linear Regression

(to R)

```
[1]: import pandas as pd
import numpy as np
import statsmodels.api as sm

# Load data
df = pd.read_csv(
    '../.../Themen/Multiple_Lineare_Regression/Daten/Credit.csv')
balance = df['Balance']

# Initiate dummy variable with zeros:
gender = np.zeros(len(balance))
# Make 1 for Female:
indices_Fem = df[df['Gender']=='Female'].index.values
gender[indices_Fem] = 1

# Fit model
gender_sm = sm.add_constant(gender)
model = sm.OLS(balance, gender_sm).fit()

# Print summary:
print(model.summary())
```

```
OLS Regression Results
=====
Dep. Variable: Balance    R-squared:      0.000
Model:          OLS         Adj. R-squared: -0.002
Method:        Least Squares F-statistic:   0.1836
Date:       Wed, 13 Jan 2021 Prob (F-statistic): 0.669
Time:           11:31:11   Log-Likelihood: -3019.3
No. Observations: 400      AIC:             6043.
Df Residuals:    398      BIC:             6051.
Df Model:        1
Covariance Type: nonrobust
=====
            coef    std err     t      P>|t|      [0.025      0.975]
-----  
const    509.8031    33.128    15.389    0.000    444.675    574.931
x1       19.7331    46.051     0.429    0.669    -70.801    110.267
=====
Omnibus:            28.438   Durbin-Watson:    1.940
Prob(Omnibus):      0.000    Jarque-Bera (JB): 27.346
Skew:               0.583    Prob(JB):       1.15e-06
Kurtosis:            2.471   Cond. No.:      2.66
=====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

The `.summary()` method of a fitted model shows the encoding of the *dummy variable* associated with `gender`.

Table 7.4 displays the coefficient estimates and other information associated with our model. The average credit card debt for males is estimated to be \$509.80,

## Chapter 7. Multiple Linear Regression

	Coefficient	Std.error	t-statistic	p-value
Intercept	509.80	33.13	15.389	< 0.0001
gender [female]	19.73	46.05	0.4285	0.6685

**Table 7.4.:** Least squares coefficient estimates associated with the regression of `balance` onto `gender` in the `Credit` data set. The linear model is given in equation 7.2. That is, `gender` is encoded as a dummy variable.

whereas females are estimated to carry \$ 19.73 in additional debt for a total of \$  $509.80 + \$ 19.73 = \$ 529.53$ .

However, we notice that the p-value for the dummy variable  $\beta_1$  is 0.6690, hence it is very high. This indicates that there is no statistical evidence of a difference in average credit card balance between the genders. ◀

The decision to code females as 1 and males as 0 is arbitrary, and has no effect on the regression fit, but does alter the interpretation of the coefficients.

### Example 7.4.6

If we had coded males as 1 and females as 0, then the estimates for  $\beta_0$  and  $\beta_1$  would have been 529.53 and  $-19.73$  respectively, leading once again to a prediction of credit card debt of  $\$ 529.73 - \$ 19.73 = \$ 509.80$  for males and a prediction of  $\$ 529.53$  for females. This is the same result we obtained with the “default” coding scheme.

If we wish to change the coding scheme for the dummy variable, we can change it in `Python` by changing the coding scheme.

(to R)

```
[2]: # This Programm follows Example 4.5
# Initiate dummy variable with zeros:
gender = np.zeros(len(balance))
# Make 1 for Male:
indices_Mal = df[df['Gender'] == 'Male'].index.values
gender[indices_Mal] = 1

# Fit model
gender_sm = sm.add_constant(gender)
model = sm.OLS(balance, gender_sm).fit()

# Print summary:
print(model.summary())
```

```
OLS Regression Results
=====
Dep. Variable:      Balance   R-squared:           0.000
Model:                 OLS     Adj. R-squared:        -0.002
Method:              Least Squares   F-statistic:         0.1836
Date:      Tue, 26 Jan 2021   Prob (F-statistic):    0.669
```

## Chapter 7. Multiple Linear Regression

```

Time:                      09:29:49  Log-Likelihood:           -3019.3
No. Observations:          400    AIC:                  6043.
Df Residuals:              398    BIC:                  6051.
Df Model:                  1
Covariance Type:          nonrobust
=====
            coef      std err       t   P>|t|      [0.025      0.975]
-----
const      529.5362    31.988     16.554    0.000    466.649    592.423
x1        -19.7331    46.051     -0.429    0.669   -110.267    70.801
=====
Omnibus:                28.438  Durbin-Watson:           1.940
Prob(Omnibus):            0.000  Jarque-Bera (JB):        27.346
Skew:                   0.583  Prob(JB):             1.15e-06
Kurtosis:                2.471  Cond. No.                 2.58
=====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

```



### Example 7.4.7

Alternatively, instead of a 0/1 coding scheme, we could create a dummy variable

$$x_i = \begin{cases} 1 & \text{if } i\text{th person is female} \\ -1 & \text{if } i\text{th person is male} \end{cases}$$

and use this variable in the regression equation. This results in the model

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i = \begin{cases} \beta_0 + \beta_1 + \varepsilon_i & \text{if } i\text{th person is male} \\ \beta_0 - \beta_1 + \varepsilon_i & \text{if } i\text{th person is female} \end{cases}$$

Now  $\beta_0$  can be interpreted as the overall credit card balance (ignoring the gender effect), and  $\beta_1$  is the amount that females are above the average and males are below the average.

(to R)

```
[3]: # Following Example 4.6
# Initiate dummy variable with ones:
gender = np.ones(len(balance))
# Make -1 for Male:
indices_Mal = df[df['Gender'] == 'Male'].index.values
gender[indices_Mal] = -1

# Fit model
gender_sm = sm.add_constant(gender)
model = sm.OLS(balance, gender_sm).fit()
```

## Chapter 7. Multiple Linear Regression

```
# Print summary:  
print(model.summary())
```

```
OLS Regression Results  
=====  
Dep. Variable: Balance   R-squared:      0.000  
Model:          OLS         Adj. R-squared: -0.002  
Method:         Least Squares F-statistic:    0.1836  
Date:           Wed, 13 Jan 2021 Prob (F-statistic): 0.669  
Time:           11:31:11    Log-Likelihood: -3019.3  
No. Observations: 400        AIC:             6043.  
Df Residuals:   398        BIC:             6051.  
Df Model:       1  
Covariance Type: nonrobust  
=====  
            coef    std err     t      P>|t|      [0.025      0.975]  
-----  
const    519.6697   23.026   22.569   0.000    474.403    564.937  
x1       9.8666    23.026    0.429   0.669    -35.400     55.134  
=====  
Omnibus:            28.438   Durbin-Watson:  1.940  
Prob(Omnibus):      0.000    Jarque-Bera (JB): 27.346  
Skew:               0.583    Prob(JB):       1.15e-06  
Kurtosis:            2.471   Cond. No.:      1.04  
=====  
Notes:  
[1] Standard Errors assume that the covariance matrix of the errors is correctly  
specified.
```

In this example, the estimate for  $\beta_0$  would be \$ 519.67 , halfway between the male and female averages of \$ 509.80 and \$ 529.53. The estimate for  $\beta_1$  would be \$ 9.87, which is half of \$ 19.73, the average difference between females and males.



It is important to note that the final predictions for the credit balances of males and females will be identical *regardless* of the coding scheme used. The only difference is in the way that coefficients are interpreted.

### Qualitative Predictors with More than Two Levels

When a qualitative predictor has more than two levels, a single dummy variable cannot represent all possible values. In this situation, we can create additional dummy variables.

#### Example 7.4.8

For example, for the **ethnicity** variable which has *three* levels we create *two* dummy variables. The first could be

$$x_{i1} = \begin{cases} 1 & \text{if } i\text{th person is Asian} \\ 0 & \text{if } i\text{th person is not Asian} \end{cases}$$

## Chapter 7. Multiple Linear Regression

and the second could be

$$x_{i2} = \begin{cases} 1 & \text{if } i\text{th person is Caucasian} \\ 0 & \text{if } i\text{th person is not Caucasian} \end{cases}$$

Then both of these variables can be used in the regression equation, in order to obtain the model

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \varepsilon_i = \begin{cases} \beta_0 + \beta_1 + \varepsilon_i & \text{if } i\text{th person is Asian} \\ \beta_0 + \beta_2 + \varepsilon_i & \text{if } i\text{th person is Caucasian} \\ \beta_0 + \varepsilon_i & \text{if } i\text{th person is Afro-American} \end{cases} \quad (7.3)$$

Now  $\beta_0$  can be interpreted as the average credit card balance for African Americans,  $\beta_1$  can be interpreted as the difference in the average balance between the Asian and African American categories, and  $\beta_2$  can be interpreted as the difference in the average balance between the Caucasian and African American categories.

### Remarks:

- i. There will always be one fewer dummy variable than the number of levels.
- ii. The level with no dummy variable - African American in the example - is known as the *baseline*.
- iii. The equation

$$y_i = \beta_0 + \beta_1 + \beta_2 + \varepsilon_i$$

does not make sense, since this person would be Asian *and* Caucasian. ◆

From Table 7.5, we see that the estimated **balance** for the baseline, African American, is \$ 531.00.

(to R)

```
[4] : # Following Example 4.7
# Initiate dummy variable with zeros:
ethnicity = np.zeros((len(balance), 2))
# Find indices
indices_Asi = df[df['Ethnicity'] == 'Asian'].index.values
indices_Cau = df[df['Ethnicity'] == 'Caucasian'].index.values
# Set values
ethnicity[indices_Asi, 0] = 1
ethnicity[indices_Cau, 1] = 1

# Fit model
ethnicity_sm = sm.add_constant(ethnicity)
model = sm.OLS(balance, ethnicity_sm).fit()

# Print summary:
print(model.summary())
```

## Chapter 7. Multiple Linear Regression

OLS Regression Results						
Dep. Variable:	Balance	R-squared:	0.000			
Model:	OLS	Adj. R-squared:	-0.005			
Method:	Least Squares	F-statistic:	0.04344			
Date:	Wed, 13 Jan 2021	Prob (F-statistic):	0.957			
Time:	11:31:11	Log-Likelihood:	-3019.3			
No. Observations:	400	AIC:	6045.			
Df Residuals:	397	BIC:	6057.			
Df Model:	2					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	531.0000	46.319	11.464	0.000	439.939	622.061
x1	-18.6863	65.021	-0.287	0.774	-146.515	109.142
x2	-12.5025	56.681	-0.221	0.826	-123.935	98.930
Omnibus:	28.829	Durbin-Watson:	1.946			
Prob(Omnibus):	0.000	Jarque-Bera (JB):	27.395			
Skew:	0.581	Prob(JB):	1.13e-06			
Kurtosis:	2.460	Cond. No.	4.39			

Notes:  
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

	Koeffizient	Std.fehler	t-Statistik	P-Wert
Intercept	531.00	46.32	11.464	< 0.0001
ethnicity[Asian]	-18.69	65.02	-0.287	0.7740
ethnicity[Caucasian]	-12.50	56.68	-0.221	0.8255

**Table 7.5.:** Least squares coefficient estimates associated with the regression of `balance` onto `ethnicity` in the `Credit` data set. The linear model is given in equation 7.3. That is, `ethnicity` is encoded via two dummy variables.

It is estimated that the Asian category will have \$ 18.69 less debt than the African American category, and that the Caucasian category will have \$ 12.50 less debt than the African American category. However, the p-values associated with the coefficient estimates for the two dummy variables are very large, suggesting no statistical evidence of a real difference in credit card balance between the ethnicities. Once again, the level selected as the baseline category is arbitrary, and the final predictions for each group will be the same regardless of this choice. However, the coefficients and their p-values do depend on the choice of dummy variable coding. Rather than rely on the individual coefficients, we can use an F-test to test

$$H_0 : \beta_1 = \beta_2 = 0$$

The p-value does not depend on the coding. This F-test has a p-value of 0.96, indicating that we cannot reject the null hypothesis that there is *no* relationship between `balance` and `ethnicity`.



Using this dummy variable approach presents no difficulties when incorporating both quantitative and qualitative predictors. For example, to regress `balance` on both a quantitative variable such as `income` and a qualitative variable such as `student`, we must simply create a dummy variable for `student` and then fit a multiple regression model using `income` and the dummy variable as predictors for credit card balance.

#### Remarks:

- i. There are many different ways of coding qualitative variables besides the dummy variable approach taken here. All of these approaches lead to equivalent model fits, but the coefficients are different and have different interpretations, and are designed to measure particular *contrasts*. This topic is beyond the scope of these lecture notes, and so we will not pursue it further. ♦

### 7.4.3. Extensions of the Linear Model

The standard linear regression model

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p + \varepsilon$$

provides interpretable results and works quite well on many real-world problems. However, it makes several highly restrictive assumptions that are often violated in practice. Two of the most important assumptions state that the relationship between the predictors and response are *additive* and *linear*.

#### 1. Additivity

The *additive* assumption means that the effect of changes in a predictor  $X_j$  on the response  $Y$  is independent of the values of the other predictors.

#### 2. Linearity

The *linear* assumption states that the change in the response  $Y$  due to a one-unit change in  $X_j$  is constant, regardless of the value of  $X_j$ .

In the following, we examine some common classical approaches for extending the linear model.

## Removing the Additive Assumption

### Example 7.4.9

In our previous analysis of the **Advertising** data, we concluded that both **TV** and **radio** seem to be associated with **sales**. The linear models that formed the basis for this conclusion assumed that the effect on **sales** of increasing one advertising medium is independent of the amount spent on the other media. For example, the linear model

$$\text{sales} = \beta_0 + \beta_1 \cdot \text{TV} + \beta_2 \cdot \text{radio} + \varepsilon$$

states that the average effect on **sales** of a one-unit increase in **TV** is always  $\beta_1$ , regardless of the amount spent on **radio**.

However, this simple model may be incorrect. Suppose that spending money on **radio** advertising actually increases the effectiveness of **TV** advertising, so that the slope term for **TV** should increase as **radio** increases.

In this situation, given a fixed budget of CHF 100 000, spending half on **radio** and half on **TV** may increase **sales** more than allocating the entire amount to either **TV** or to **radio**. In marketing, this is known as a *synergy* effect, and in statistics it is referred to as an *interaction* effect. Figure 7.4 on page 189 suggests that such an effect may be present in the **Advertising** data. Notice that when levels of either **TV** or **radio** are low, then the true **sales** are lower than predicted by the linear model. But when advertising is split between the two media, then the model tends to underestimate **sales**.



Consider the standard linear regression model with two variables,

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \varepsilon$$

According to this model, if we increase  $X_1$  by one unit, then  $Y$  will increase by an average of  $\beta_1$  units. Notice that the presence of  $X_2$  does not alter this statement - that is, regardless of the value of  $X_2$ , a one-unit increase in  $X_1$  will lead to a  $\beta_1$ -unit increase in  $Y$ . One way of extending this model to allow for *interaction effects* is to include a third predictor, called an *interaction term*, which is constructed by computing the product of  $X_1$  and  $X_2$ . This results in the model

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1 X_2 + \varepsilon$$

How does inclusion of this interaction term relax the additive assumption? Notice that the previous equation can be rewritten as

$$\begin{aligned} Y &= \beta_0 + (\beta_1 + \beta_3 X_2) X_1 + \beta_2 X_2 + \varepsilon \\ &= \beta_0 + \tilde{\beta}_1 X_1 + \beta_2 X_2 + \varepsilon \end{aligned}$$

where

$$\tilde{\beta}_1 = \beta_1 + \beta_3 X_2$$

Since  $\tilde{\beta}_1$  changes with  $X_2$ , the effect of  $X_1$  on  $Y$  is no longer constant: adjusting  $X_2$  will change the impact of  $X_1$  on  $Y$ .

### Example 7.4.10

For example, suppose that we are interested in studying the productivity of a factory. We wish to predict the number of **units** produced on the basis of the number of production **lines** and the total number of **workers**. It seems likely that the effect of increasing the number of production lines will depend on the number of workers, since if no workers are available to operate the **lines**, then increasing the number of **lines** will not increase production.

This suggests that it would be appropriate to include an interaction term between **lines** and **workers** in a linear model to predict **units**. Suppose that when we fit the model, we obtain

$$\begin{aligned}\text{units} &\approx 1.2 + 3.4 \cdot \text{lines} + 0.22 \cdot \text{workers} + 1.4 \cdot (\text{lines} \cdot \text{workers}) \\ &= 1.2 + (3.4 + 1.4 \cdot \text{workers}) \cdot \text{lines} + 0.22 \cdot \text{workers}\end{aligned}$$

In other words, adding an additional **line** will increase the number of units produced by  $3.4 + 1.4 \cdot \text{workers}$ . Hence the more **workers** we have, the stronger will be the effect of **lines**.

#### Remarks:

- i. There are however some restrictions with respect to the validity of the predictive power of this model. Suppose, *no workers* are available. Then the factory would still produce the following number of

$$\text{units} \approx 1.2 + 3.4 \cdot \text{lines}$$

This does not make sense. The problem here is that we are *extrapolating* into a region of predictor values where the model is not appropriate. The assumption of having *no workers* on a normal working day is certainly not realistic. ♦



## Chapter 7. Multiple Linear Regression

### Example 7.4.11

We now return to the **Advertising** example. A linear model that uses **radio**, **TV**, and an interaction between the two to predict **sales** takes the form

$$\begin{aligned}\text{sales} &= \beta_0 + \beta_1 \cdot \text{TV} + \beta_2 \cdot \text{radio} + \beta_3 \cdot (\text{TV} \cdot \text{radio}) + \varepsilon \\ &= \beta_0 + (\beta_1 + \beta_3 \cdot \text{radio}) \cdot \text{TV} + \beta_2 \cdot \text{radio} + \varepsilon\end{aligned}$$

We can interpret  $\beta_3$  as the increase in the effectiveness of **TV** advertising for a one unit increase in **radio** advertising (or vice-versa). The coefficients that result from fitting this model can be found in the following Python-output:

(to R)

```
[1]: import pandas as pd
import statsmodels.api as sm

# Load data
df = pd.read_csv(
    '../../../../../Themen/Einfache_Lineare_Regression/Daten/Advertising.csv')

# Define the linear model:
x = pd.DataFrame({
    'TV' : df['TV'],
    'radio' : df['radio'],
    'TV*radio' : df['TV'] * df['radio']})
y = df['sales']

# Fit model
x_sm = sm.add_constant(x)
model = sm.OLS(y, x_sm).fit()

# Print summary:
print(model.summary())
```

```
OLS Regression Results
=====
Dep. Variable: sales R-squared: 0.968
Model: OLS Adj. R-squared: 0.967
Method: Least Squares F-statistic: 1963.
Date: Tue, 19 Jan 2021 Prob (F-statistic): 6.68e-146
Time: 10:00:13 Log-Likelihood: -270.14
No. Observations: 200 AIC: 548.3
Df Residuals: 196 BIC: 561.5
Df Model: 3
Covariance Type: nonrobust
=====
            coef  std err      t  P>|t|      [0.025      0.975]
-----
const     6.7502   0.248  27.233  0.000     6.261     7.239
TV        0.0191   0.002  12.699  0.000     0.016     0.022
radio     0.0289   0.009   3.241  0.001     0.011     0.046
TV*radio  0.0011  5.24e-05  20.727  0.000     0.001     0.001
=====
Omnibus: 128.132  Durbin-Watson: 2.224
```

## Chapter 7. Multiple Linear Regression

```

Prob(Omnibus):          0.000   Jarque-Bera (JB):        1183.719
Skew:                  -2.323   Prob(JB):            9.09e-258
Kurtosis:              13.975   Cond. No.:           1.80e+04
=====
```

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The condition number is large, 1.8e+04. This might indicate that there are strong multicollinearity or other numerical problems.

The results strongly suggest that the model that includes the interaction term is superior to the model that contains only *main effects*. The p-value for the interaction term, **TV · radio**, is extremely low, indicating that there is a strong evidence for  $H_A : \beta_3 \neq 0$ . In other words it is clear, that the true relationship is not additive.

The  $R^2$  for the model, that includes in addition to the predictors **TV** and **radio** as well the interaction term **TV · radio**, is 0.968; compared to only 0.897 for the model that predicts **sales** using **TV** and **radio** without an interaction term. This means, that

$$\frac{0.968 - 0.897}{1 - 0.897} = 0.69 = 69\%$$

of the variability in **sales** that remains after fitting the additive model has been explained by the interaction term.

The coefficient estimates in the previous Python-output suggest that an increase in **TV** advertising of CHF 1000 is associated with increased **sales** of

$$(\hat{\beta}_1 + \hat{\beta}_3 \cdot \text{radio}) \cdot 1.000 = 19 + 1.1 \cdot \text{radio}$$

units. And an increase in **radio** advertising of CHF 1000 will be associated with an increase in **sales** of

$$(\hat{\beta}_2 + \hat{\beta}_3 \cdot \text{TV}) \cdot 1.000 = 29 + 1.1 \cdot \text{TV}$$

units. ◀

In this example, the p-values associated with **TV**, **radio**, and the interaction term **TV · radio** all are statistically significant, and so it is obvious that all three variables should be included in the model. However, it is sometimes the case that an interaction term has a very small p-value, but the associated main effects (in this case, **TV** and **radio**) do not.

The *hierarchical principle* states that if we include an interaction in a model, we should also include the main effects, even if the p-values associated with their coefficients are not significant. In other words, if the interaction between  $X_1$  and  $X_2$  seems important, then we should include both  $X_1$  and  $X_2$  in the model even if their coefficient estimates have large p-values.

## Chapter 7. Multiple Linear Regression

The rationale for this principle is that if  $X_1 \cdot X_2$  is related to the response, then whether or not the coefficients of  $X_1$  or  $X_2$  are exactly zero is of little interest. Also  $X_1 \cdot X_2$  is typically correlated with  $X_1$  and  $X_2$ , and so leaving them out tends to alter the meaning of the interaction.

In the previous example, we considered an interaction between **TV** and **radio**, both of which are quantitative variables. However, the concept of interactions applies just as well to qualitative variables, or to a combination of quantitative and qualitative variables. In fact, an interaction between a qualitative variable and a quantitative variable has a particularly nice interpretation.

### Example 7.4.12

Consider the **Credit** data set, and suppose that we wish to predict **balance** using the **income** (quantitative) and **student** (qualitative) variables. In the absence of an interaction term, the model takes the form

$$\begin{aligned}\text{balance}_i &\approx \beta_0 + \beta_1 \cdot \text{income}_i + \begin{cases} \beta_2 & \text{if } i\text{th person is a student} \\ 0 & \text{if } i\text{th person is not a student} \end{cases} \\ &= \beta_1 \cdot \text{income}_i + \begin{cases} \beta_0 + \beta_2 & \text{if } i\text{th person is a student} \\ \beta_0 & \text{if } i\text{th person is not a student} \end{cases}\end{aligned}$$

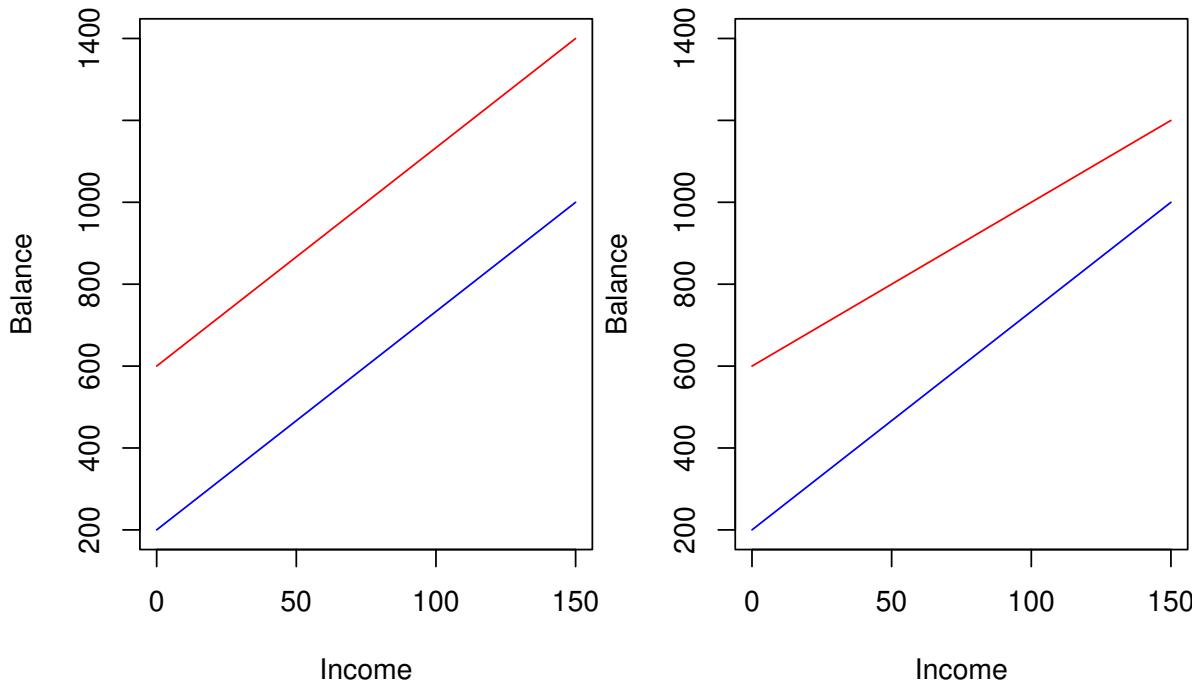
Notice that this amounts to fitting two parallel lines to the data, one for students and one for non-students. The lines for students and non-students have different intercepts,  $\beta_0 + \beta_2$  and  $\beta_0$ , but the same slope  $\beta_1$ . This is illustrated in the left-hand panel of Figure 7.6.

The fact that the lines are parallel means that the average effect on **balance** of a one-unit increase in **income** does not depend on whether or not the individual is a student. This represents a potentially serious limitation of the model, since in fact a change in **income** may have a very different effect on the credit card balance of a student versus a non-student.

This limitation can be addressed by adding an interaction variable, created by multiplying **income** with the dummy variable for **student**. Our model now becomes

$$\begin{aligned}\text{balance}_i &\approx \beta_0 + \beta_1 \cdot \text{income}_i + \begin{cases} \beta_2 + \beta_3 \cdot \text{income}_i & \text{if student} \\ 0 & \text{if not student} \end{cases} \\ &= \begin{cases} (\beta_0 + \beta_2) + (\beta_1 + \beta_3) \cdot \text{income}_i & \text{if student} \\ \beta_0 + \beta_1 \cdot \text{income}_i & \text{if not student} \end{cases}\end{aligned}$$

Once again, we have two different regression lines for the students and the non-students. But now those regression lines have different intercepts,  $\beta_0 + \beta_2$  versus  $\beta_0$ , as well as different slopes  $\beta_1 + \beta_3$  versus  $\beta_1$ . This allows for the possibility that



**Figure 7.6.**: For the `Credit` data, the least squares lines are shown for prediction of `balance` from `income` for students (red) and non-students (blue). *Left:* There is no interaction between `income` and `student`. *Right:* There is an interaction term between `income` and `student`.

changes in `income` may affect the credit card balances of students and non-students differently. The right-hand panel of Figure 7.6 shows the estimated relationship between `income` and `balance` for students (red) and non-students (blue).

We note that the slope for students is lower than the slope for non-students. This suggests that increases in `income` are associated with smaller increases in credit card balance among students as compared to non-students.



### Non-linear Relationships

As discussed previously, the linear regression model

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p + \varepsilon$$

assumes a linear relationship between the response and predictors. But in some cases, the true relationship between the response and the predictors may be non-linear.

Here we present a very simple way to directly extend the linear model to accommodate non-linear relationships, using *polynomial regression*.

### Example 7.4.13

Consider the left-hand panel of Figure 7.7, in which the `mpg` (gas mileage in miles per gallon) versus `horsepower` is shown for a number of cars in the `Auto` data set. The blue line represents the linear regression fit.

(to R)

```
[1]: import pandas as pd
import statsmodels.api as sm
import seaborn as sns
from matplotlib import pyplot as plt

# Load data
df = pd.read_csv(
    '.../Themen/Einfache_Lineare_Regression/Daten/Auto.csv')

# Define the linear model:
x = df['horsepower']
y = df['mpg']

# Fit model
x_sm = sm.add_constant(x)
model = sm.OLS(y, x_sm).fit()

# Create figure:
fig = plt.figure(figsize=(14, 5))

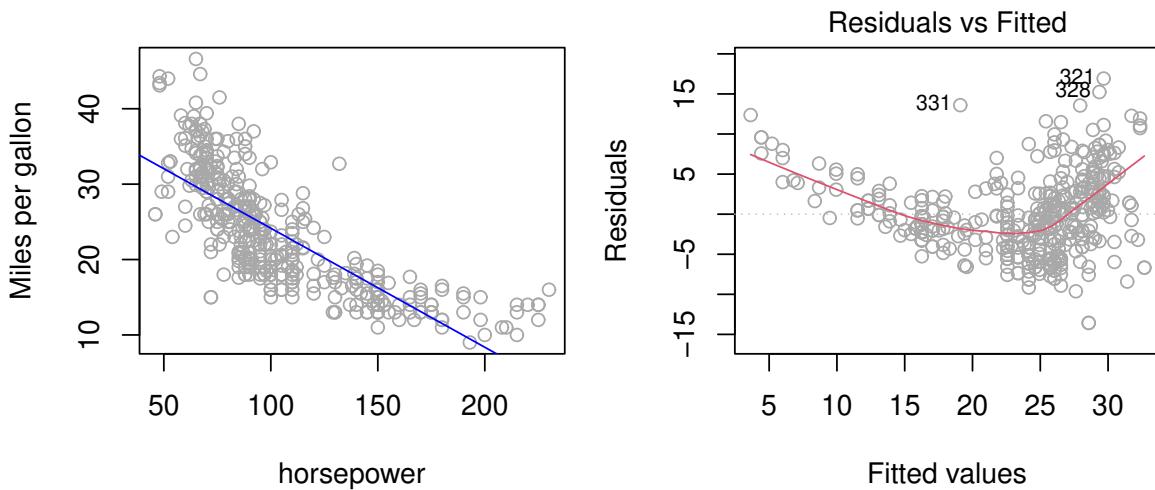
# Plot left figure: Scatter data and linear fit
ax1 = fig.add_subplot(1, 2, 1)
# Scatter data
plt.plot(x, y, marker='o', linestyle='None',
          color='darkcyan', markersize='5', alpha=0.5,
          label="Scatter data")
# Linear fit
plt.plot(x, model.fittedvalues, 'b-', label="Linear fit")
# Set labels
ax1.set_title('Data and linear fit')
ax1.set_xlabel('Horsepower')
ax1.set_ylabel('Miles per gallon')

# Plot right figure: Residuals vs fitted data
ax2 = fig.add_subplot(1, 2, 2)
# Residuals vs fitted value, using seaborn
ax2 = sns.residplot(
    x=model.fittedvalues, y=model.resid,
    data=df, lowess=True,
    scatter_kws={'color': 'darkcyan', 's': 20, 'alpha': 0.5},
    line_kws={'color': 'red', 'lw': 2, 'alpha': 0.8})
# Set labels
ax2.set_title('Residuals vs Fitted Values')
ax2.set_ylabel('Residuals')
```

## Chapter 7. Multiple Linear Regression

```
ax2.set_xlabel('Fitted Values')

# Show plot
plt.show()
```



**Figure 7.7.:** *Left:* The `Auto` data set. For a number of cars, `mpg` and `horsepower` are shown. The linear regression fit is shown in blue. *Right:* The Tukey-Anscombe plot suggests that the underlying regression function is non-linear.

A simple approach for incorporating non-linear associations in a linear model is to include transformed versions of the predictors in the model. For example, the points in Figure 7.7 seem to have a *quadratic* shape, suggesting that a model of the form

$$\text{mpg} = \beta_0 + \beta_1 \cdot \text{horsepower} + \beta_2 \cdot \text{horsepower}^2 + \varepsilon \quad (7.4)$$

may provide a better fit. Equation 7.4 involves predicting `mpg` using a non-linear function of `horsepower`. *But it is still a linear model!* That is, equation 7.4 is simply a multiple linear regression model with

$$X_1 = \text{horsepower} \quad \text{and} \quad X_2 = \text{horsepower}^2$$

So we can use standard linear regression software to estimate  $\beta_0$ ,  $\beta_1$ , and  $\beta_2$  in order to produce a non-linear fit.

(to R)

```
[2]: # Define the linear model:
x = pd.DataFrame({
    'horsepower' : df['horsepower'],
    'horsepower^2' : (df['horsepower'] * df['horsepower']) })
y = df['mpg']

# Fit model
```

## Chapter 7. Multiple Linear Regression

```
x_sm = sm.add_constant(x)
model = sm.OLS(y, x_sm).fit()

# Print summary:
print(model.summary())
```

```
OLS Regression Results
=====
Dep. Variable: mpg R-squared: 0.688
Model: OLS Adj. R-squared: 0.686
Method: Least Squares F-statistic: 428.0
Date: Tue, 19 Jan 2021 Prob (F-statistic): 5.40e-99
Time: 13:17:12 Log-Likelihood: -797.76
No. Observations: 392 AIC: 1602.
Df Residuals: 389 BIC: 1613.
Df Model: 2
Covariance Type: nonrobust
=====
```

	coef	std err	t	P> t	[0.025	0.975]
const	24.1825	0.765	31.604	0.000	22.678	25.687
horsepower	-0.1981	0.013	-14.978	0.000	-0.224	-0.172
horsepower^2	0.0005	5.19e-05	10.080	0.000	0.000	0.001

```
=====

Omnibus: 16.158 Durbin-Watson: 1.078
Prob(Omnibus): 0.000 Jarque-Bera (JB): 30.662
Skew: 0.218 Prob(JB): 2.20e-07
Kurtosis: 4.299 Cond. No. 1.29e+05
=====
```

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The condition number is large, 1.29e+05. This might indicate that there are strong multicollinearity or other numerical problems.

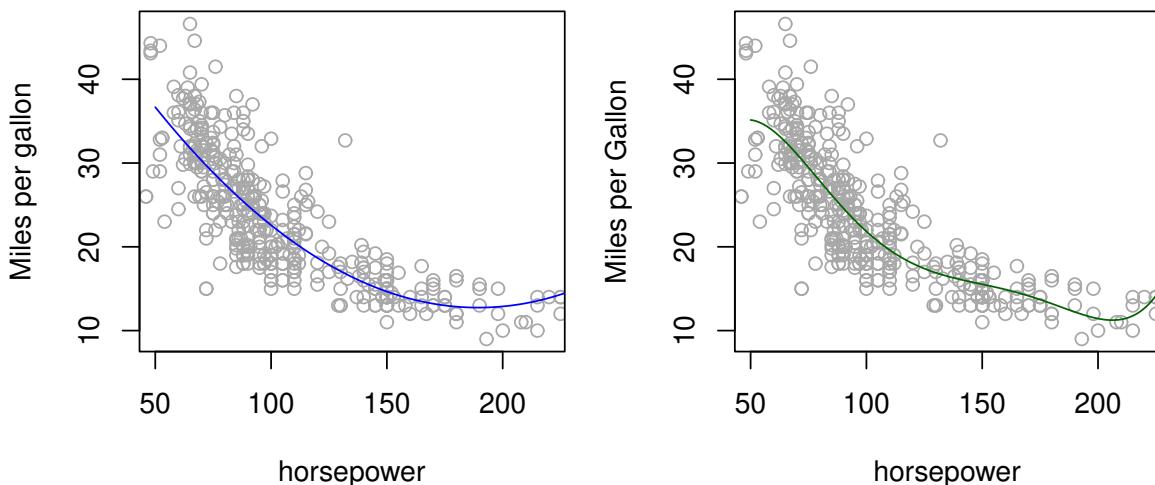
```
[3]: # Create figure:
fig = plt.figure(figsize=(14, 5))

# Plot left figure: Scatter data and linear fit
ax1 = fig.add_subplot(1, 2, 1)
# Scatter data
plt.plot(x['horsepower'], y, marker='o', linestyle='None',
          color='darkcyan', markersize='5', alpha=0.5,
          label="Scatter data")
# Quadratic fit
x_quad_fit = x.sort_values(by='horsepower')
y_quad_fit = (model.params['const']
              + x_quad_fit['horsepower'] * model.params['horsepower']
              + x_quad_fit['horsepower^2'] * model.params['horsepower^2'])
plt.plot(x_quad_fit['horsepower'], y_quad_fit,
          'b-', label="Quadratic fit")
# Set labels
ax1.set_title('Data and Quadratic fit')
ax1.set_xlabel('Horsepower')
ax1.set_ylabel('Miles per gallon')
```

## Chapter 7. Multiple Linear Regression

```
# Plot right figure: Residuals vs fitted data
ax2 = fig.add_subplot(1, 2, 2)
# Residuals vs fitted value, using seaborn
ax2 = sns.residplot(
    x=model.fittedvalues, y=model.resid,
    data=df, lowess=True,
    scatter_kws={'color': 'darkcyan', 's': 20, 'alpha': 0.5},
    line_kws={'color': 'red', 'lw': 2, 'alpha': 0.8})
# Set labels
ax2.set_title('Residuals vs Fitted Values')
ax2.set_ylabel('Residuals')
ax2.set_xlabel('Fitted Values')

# Show plot
plt.show()
```



**Figure 7.8:** *Left:* The linear regression fit for a model that includes `horsepower`<sup>2</sup> is shown as a blue curve. *Right:* The linear regression fit for a model that includes all polynomials of `horsepower` up to fifth-degree is shown in green.

The blue curve in Figure 7.8 shows the resulting quadratic fit to the data. The quadratic fit in Figure 7.8 appears to be substantially better than the fit obtained in Figure 7.7 when just the linear term is included. The  $R^2$  of the quadratic fit is 0.688, compared to 0.606 for the linear fit, and the p-value for the quadratic term is highly significant.

Let us as well perform an ANOVA-analysis

(to R)

```
[4] : # Define the linear model:
x = pd.DataFrame({
    'horsepower' : df['horsepower'],
    'horsepower^2' : (df['horsepower'] * df['horsepower']),
```

## Chapter 7. Multiple Linear Regression

```

'horsepower^3' : (df['horsepower'] ** 3),
'horsepower^4' : (df['horsepower'] ** 4),
'horsepower^5' : (df['horsepower'] ** 5),
})
y = df['mpg']

# Fit model
x_sm = sm.add_constant(x)
model = sm.OLS(y, x_sm).fit()

# Print summary:
# print(model.summary())
# Create figure:
fig = plt.figure(figsize=(6, 5))

# Plot left figure: Scatter data and polynomial fit
ax1 = fig.add_subplot(1, 1, 1)
# Scatter data
plt.plot(x['horsepower'], y, marker='o', linestyle='None',
          color='darkcyan', markersize=5, alpha=0.5,
          label="Scatter data")
# Polynomial fit
x_quad_fit = x.sort_values(by='horsepower')
y_quad_fit = (model.params['const'] +
              x_quad_fit['horsepower'] * model.params['horsepower'] +
              x_quad_fit['horsepower^2'] * model.params['horsepower^2'] +
              x_quad_fit['horsepower^3'] * model.params['horsepower^3'] +
              x_quad_fit['horsepower^4'] * model.params['horsepower^4'] +
              x_quad_fit['horsepower^5'] * model.params['horsepower^5'])

plt.plot(x_quad_fit['horsepower'], y_quad_fit,
         'b-', label="Polynomial fit")
# Set labels
ax1.set_title('Data and Polynomial fit')
ax1.set_xlabel('Horsepower')
ax1.set_ylabel('Miles per gallon')

# Show plot
plt.show()

```

The p-value for the null hypothesis,  $\beta_2 = 0$ , is approximately zero. We thus reject the null hypothesis and conclude that including the quadratic term in the regression model is essential for fitting an appropriate model to the data.

If including  $\text{horsepower}^2$  led to such a big improvement in the model, why not include  $\text{horsepower}^3$ ,  $\text{horsepower}^4$  or even  $\text{horsepower}^5$ ? The right-hand panel of Figure 7.8 displays the fit that results from including all polynomials up to fifth degree in the model 7.4. The resulting fit seems unnecessarily wiggly - that is, it is unclear that including the additional terms really has led to a better fit to the data.

The approach that we have just described for extending the linear model to acco-

moderate non-linear relationships is known as *polynomial regression*, since we have included polynomial functions of the predictors in the regression model.



#### 7.4.4. Potential Problems with Respect to Model Assumptions in Multiple Linear Regression

In Chapter 6 we discussed potential problems when we fit a simple linear regression model to a particular data set. The assumptions concerning the error term  $\varepsilon$  are identical in the multiple linear regression model as in the simple linear regression model discussed in Chapter 6. We therefore can check the assumptions of the multiple linear regression model with the same methods we already have used in Chapter 6.

##### Example 7.4.14

Let us consider the following model

$$\text{sales} = \beta_0 + \beta_1 \cdot \text{radio} + \beta_2 \cdot \text{newspaper} + \varepsilon$$

and the model that includes as well the interaction term  $\text{TV} \cdot \text{radio}$

$$\text{sales} = \beta_0 + \beta_1 \cdot \text{TV} + \beta_2 \cdot \text{radio} + \beta_3 \cdot \text{TV} \cdot \text{radio} + \varepsilon$$

We have omitted the predictor **newspaper**, since in Chapter 7.4.1 we came to the conclusion that **newspaper** is not relevant to predict **sales**.

Figure 7.9 displays the Tukey-Anscombe plots for these two models. The right-hand panel displays the Tukey-Anscombe plot for the model where the interaction term  $\text{TV} \cdot \text{radio}$  is included, in the left-hand panel of Figure 7.9 the Tukey-Anscombe plot for the model without interaction term is shown. We observe that there is a clear improvement of the smoothing curve, when the interaction term is included in the model.

(to R)

```
[1]: import pandas as pd
import seaborn as sns
import statsmodels.api as sm
from matplotlib import pyplot as plt

# Load data
df = pd.read_csv(
    '/.../Themen/Einfache_Lineare_Regression/Daten/Advertising.csv')

# Define the linear models:
```

## Chapter 7. Multiple Linear Regression

```

x_plain = pd.DataFrame({
    'TV' : df['TV'],
    'radio' : df['radio']})
x_inter = pd.DataFrame({
    'TV' : df['TV'],
    'radio' : df['radio'],
    'TV*radio' : df['TV'] * df['radio']})
y = df['sales']

# Fit models
# Plain model
x_plain_sm = sm.add_constant(x_plain)
model_plain = sm.OLS(y, x_plain_sm).fit()

# Model including interaction term
x_inter_sm = sm.add_constant(x_inter)
model_inter = sm.OLS(y, x_inter_sm).fit()

```

```

[2]: # Create figure:
fig = plt.figure(figsize=(14, 5))

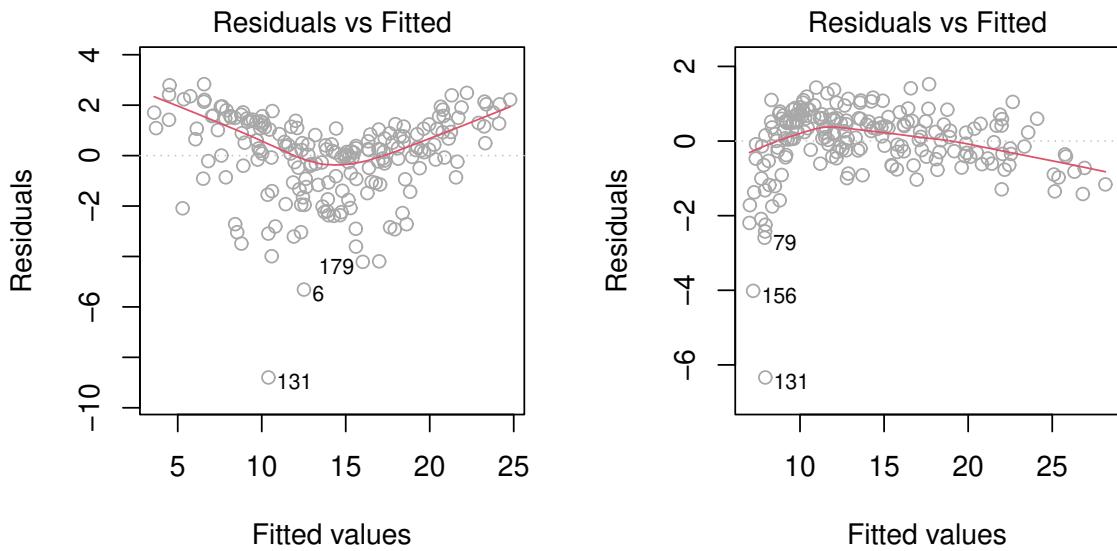
# Plot left figure: Residuals vs fitted data
ax1 = fig.add_subplot(1, 2, 1)
# Scatter data
ax1 = sns.residplot(
    x=model_plain.fittedvalues, y=model_plain.resid,
    data=df, lowess=True,
    scatter_kws={'color': 'darkcyan', 's': 20, 'alpha': 0.5},
    line_kws={'color': 'red', 'lw': 2, 'alpha': 0.8})
# Set labels
ax1.set_title('Residuals vs Fitted Values')
ax1.set_ylabel('Residuals')
ax1.set_xlabel('Fitted Values')

# Plot right figure: Residuals vs fitted data
ax2 = fig.add_subplot(1, 2, 2)
# Scatter data
ax2 = sns.residplot(
    x=model_inter.fittedvalues, y=model_inter.resid,
    data=df, lowess=True,
    scatter_kws={'color': 'darkcyan', 's': 20, 'alpha': 0.5},
    line_kws={'color': 'red', 'lw': 2, 'alpha': 0.8})
# Set labels
ax2.set_title('Residuals vs Fitted Values')
ax2.set_ylabel('Residuals')
ax2.set_xlabel('Fitted Values')

# Show plot
plt.show()

```

In the right-hand panel of Figure 7.10 the scale-location plot for the model considering the interaction term **TV · radio** is displayed, the left-hand panel displays the



**Figure 7.9.:** Tukey-Anscombe plots for the `Advertising` data. *Left:* the model includes the predictor variables `TV` and `radio`. *Right:* the model includes `TV`, `radio` and the interaction term `radio · TV`.

scale-location plot for the model without interaction term. We observe again an improvement of the smoothing curve in the sense that including the interaction term leads to a rather constant smoother.

(to R)

```
[3]: import numpy as np

# Residuals of the model
res_plain = model_plain.resid
res_inter = model_inter.resid
# Influence of the Residuals
res_inf_plain = model_plain.get_influence()
res_inf_inter = model_inter.get_influence()
# Studentized residuals using variance from OLS
res_standard_plain = res_inf_plain.resid_studentized_internal
res_standard_inter = res_inf_inter.resid_studentized_internal
# Absolute square root Residuals:
res_stand_sqrt_plain = np.sqrt(np.abs(res_standard_plain))
res_stand_sqrt_inter = np.sqrt(np.abs(res_standard_inter))

# Create figure:
fig = plt.figure(figsize=(14, 5))

# Plot left figure: Residuals vs fitted data
```

## Chapter 7. Multiple Linear Regression

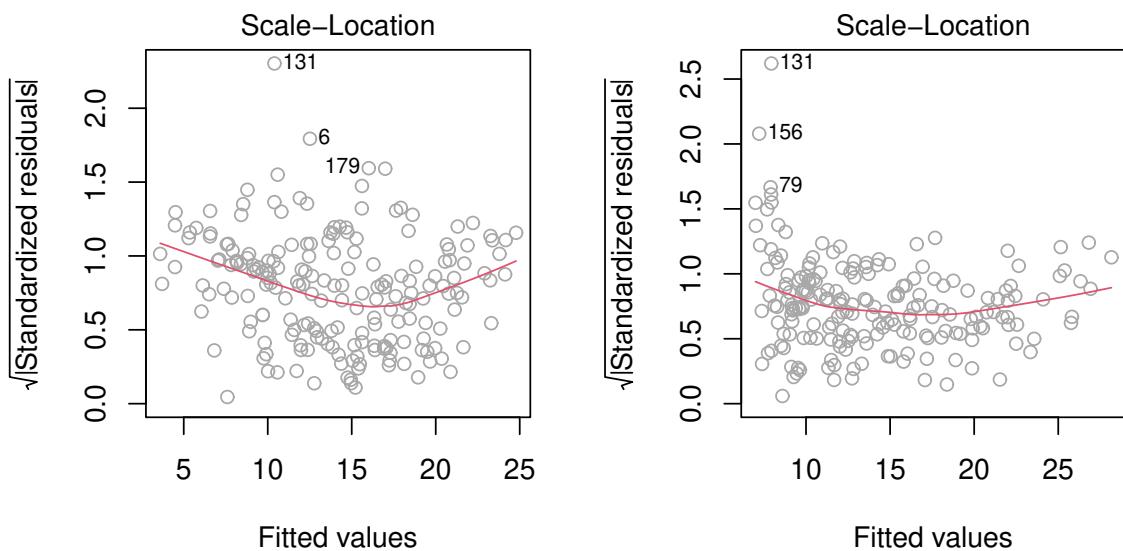
```

ax1 = fig.add_subplot(1, 2, 1)
# plot Regression using Seaborn
sns.regplot(x=model_plain.fittedvalues, y=res_stand_sqrt_plain,
             scatter=True, ci=False, lowess=True,
             scatter_kws={'color': 'darkcyan', 's': 40, 'alpha': 0.5},
             line_kws={'color': 'red', 'lw': 2, 'alpha': 0.8})
ax1.set_title('Scale-Location plot')
ax1.set_xlabel('Fitted Sales values')
ax1.set_ylabel('$\sqrt{|Standardized Residuals|}$')

# Plot left figure: Residuals vs fitted data
ax2 = fig.add_subplot(1, 2, 2)
# plot Regression using Seaborn
sns.regplot(x=model_inter.fittedvalues, y=res_stand_sqrt_inter,
             scatter=True, ci=False, lowess=True,
             scatter_kws={'color': 'darkcyan', 's': 40, 'alpha': 0.5},
             line_kws={'color': 'red', 'lw': 2, 'alpha': 0.8})
ax2.set_title('Scale-Location plot')
ax2.set_xlabel('Fitted Sales values')
ax2.set_ylabel('$\sqrt{|Standardized Residuals|}$')

# Show plot
plt.show()

```



**Figure 7.10.:** Scale-location plot for the **Advertising** data. *Left:* the model includes only the predictors **TV** and **radio**. *Right:* the model includes in addition to the predictors **TV** and **radio** their interaction term **radio · TV**.

On the basis of the residual plots, we therefore conclude that the model including the interaction term fits the data better. There is however an aspect of the residual

## Chapter 7. Multiple Linear Regression

analysis for the model including the interaction term that seems problematic: the outlying observations 131 und 156.

Figure 7.11 plots for every observation the leverage statistic  $h_i$  versus the standardized residual  $\tilde{r}_i$ . In multiple regression, the values of the leverage statistic  $h_i$  are defined as diagonal elements  $H_{ii}$  of the  $n \times n$  hat matrix  $H$ . The hat matrix is defined as

$$H = X(X^T X)^{-1} X^T$$

where  $X$  denotes the  $n \times (p + 1)$  data matrix composed of  $n$  measurements, of  $p$  features, and of ones in the first column. The expected value of the leverage statistic is given by

$$(p + 1)/n$$

where  $p$  denotes the number of predictor variables and  $n$  is the number of data points. So if a given observation has a leverage statistic that greatly exceeds the expected value, that is

$$h_i >> \frac{(p + 1)}{n}$$

then we may suspect that the corresponding point has *high leverage*. In the previous example of the **Advertising** data for the model including the interaction term, we have  $p = 3$  and  $n = 200$ . Thus, a value of the leverage statistic exceeding  $4/200 = 0.02$  should attract our attention.

In order to have a “dangerous” influence on the regression model, an observation needs to have as well a large absolute value of the residual. In simple linear regression analysis, Cook’s distance is a function of the leverage statistic  $h_i$  and of the standardized residual  $\tilde{r}_i$ . In multiple regression, Cook’s distance is defined as

$$d_i = \frac{H_{ii}}{1 - H_{ii}} \frac{\tilde{r}_i^2}{p + 1}$$

where  $H_{ii}$  denotes the diagonal elements of the hat matrix  $H$ .

Figure 7.11 displays the contour lines for the function values 1, 0.5, and 0.1 of Cook’s distance. Observations with a Cook’s distance larger than 1 must be considered as dangerous.

(to R)

```
[4] : # Cook's Distance and leverage:
res_inf_cooks_plain = res_inf_plain.cooks_distance
res_inf_cooks_inter = res_inf_inter.cooks_distance
res_inf_leverage_plain = res_inf_plain.hat_matrix_diag
res_inf_leverage_inter = res_inf_inter.hat_matrix_diag

# Contour levels for Cook's distance
n = 100    # Grid dimensions
n_pred_plain = x_plain.shape[1]
```

## Chapter 7. Multiple Linear Regression

```
n_pred_inter = x_inter.shape[1]
xmin, ymin, ymax = 0, -7, 3 # Grid boundaries
xmax_plain, xmax_inter = 0.04, 0.08
cooks_distance_plain = np.zeros((n, n))
cooks_distance_inter = np.zeros((n, n))
# Create grid
y_cooks = np.linspace(ymin, ymax, n)
x_cooks_plain = np.linspace(xmin, xmax_plain, n)
x_cooks_inter = np.linspace(xmin, xmax_inter, n)

for xi in range(n):
    for yi in range(n):
        cooks_distance_plain[yi][xi] = (
            y_cooks[yi]**2 * x_cooks_plain[xi] /
            (1 - x_cooks_plain[xi])) / (n_pred_plain + 1))
        cooks_distance_inter[yi][xi] = (
            y_cooks[yi]**2 * x_cooks_inter[xi] /
            (1 - x_cooks_inter[xi])) / (n_pred_inter + 1))

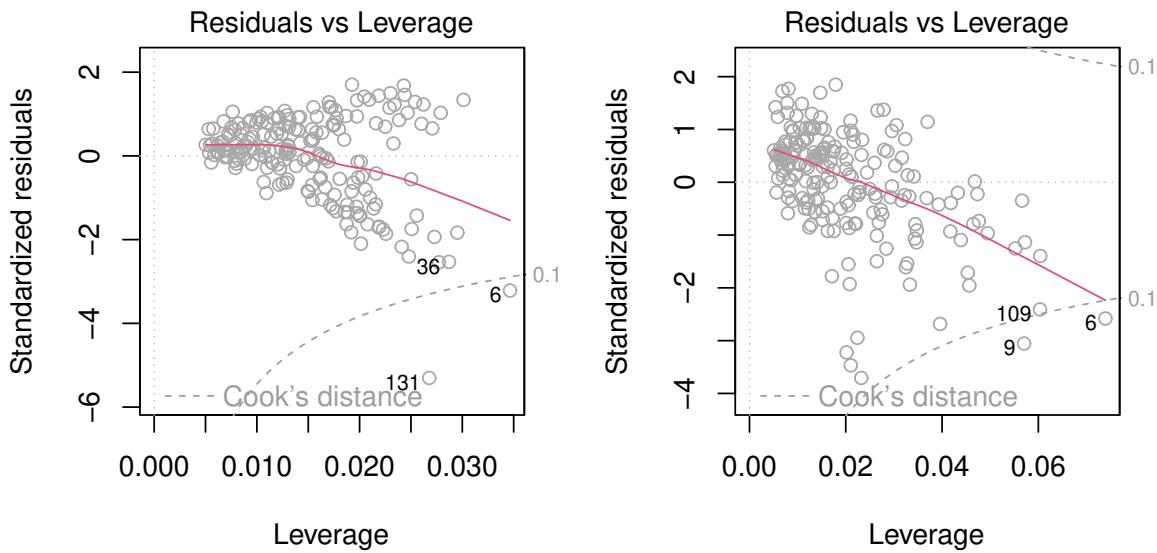
# Create figure:
fig = plt.figure(figsize=(14, 5))

# Plot left figure: Cook's distance
ax1 = fig.add_subplot(1, 2, 1)
# ax1.set_xlim(0, 0.04)
# Plot standardized Residuals
sns.regplot(x=res_inf_leverage_plain, y=res_standard_plain,
             scatter=True, ci=False, lowess=True,
             scatter_kws={'color': 'darkcyan', 's': 40, 'alpha': 0.5},
             line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8})
# Plot centre line
plt.plot((xmin, xmax_plain), (0, 0), 'g--', alpha=0.8)
# Plot Cook's Distance
CS = ax1.contour(x_cooks_plain, y_cooks, cooks_distance_plain,
                  levels=[0.1, 0.5, 1], alpha=0.6)
# labels and title
ax1.clabel(CS, inline=0, fontsize=10)
ax1.set_title('Residuals vs Leverage and Cook\\'s distance')
ax1.set_xlabel('Leverage')
ax1.set_ylabel('Standardized Residuals')

# Plot left figure: Cook's distance
ax2 = fig.add_subplot(1, 2, 2)
# Plot Regression using Seaborn
sns.regplot(x=res_inf_leverage_inter, y=res_standard_inter,
             scatter=True, ci=False, lowess=True,
             scatter_kws={'color': 'darkcyan', 's': 40, 'alpha': 0.5},
             line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8})
# Plot centre line
plt.plot((xmin, xmax_inter), (0, 0), 'g--', alpha=0.8)
# Plot Cook's Distance
CS = ax2.contour(x_cooks_inter, y_cooks, cooks_distance_inter,
                  levels=[0.1, 0.5, 1], alpha=0.6)
```

```
# labels and title
ax2.clabel(CS, inline=0, fontsize=10)
ax2.set_title('Residuals vs Leverage and Cook\\'s distance')
# ax2.set_xlabel('Leverage')
ax2.set_ylabel('Standardized Residuals')

# Show plot
plt.show()
```



**Figure 7.11.:** Scatter plot with leverage statistic  $h_i$  and standardized residual  $\tilde{r}_i$  for the regression model including the interaction term. *Left:* the outliers 131 and 156 are included in the data set; *Right* the observations 131 and 156 were removed from the data set.

Based on the plots displayed in Figure 7.11 we conclude that observations 156 and 131 are *not* dangerous. ◀

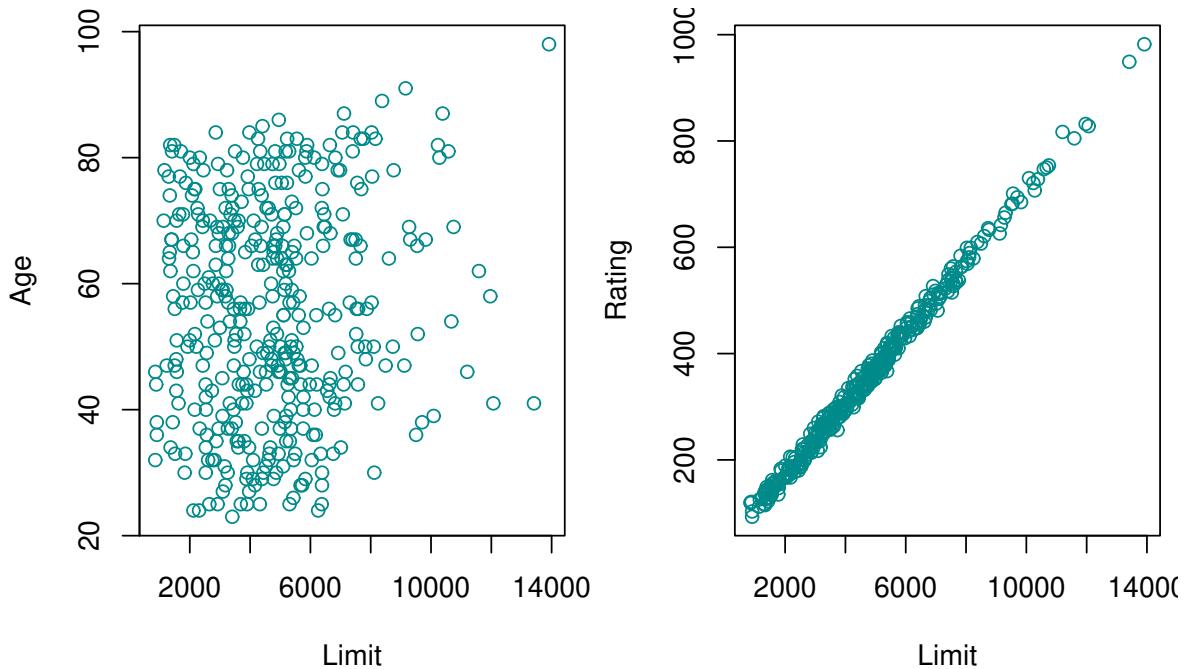
Another potential problem that often occurs in multiple linear regression refers to *collinearity*.

### Collinearity

*Collinearity* refers to the situation in which two or more predictor variables are closely related to one another.

### Example 7.4.15

The concept of collinearity is illustrated in Figure 7.12 using the `Credit` data set.



**Figure 7.12.:** Scatter plots of the observations from the `Credit` data set. *Left:* A plot of `age` versus `limit`. These two variables are not collinear. *Right:* A plot of `rating` versus `limit`. There is high collinearity.

In the left-hand panel of Figure 7.12, the two predictors `limit` and `age` appear to have no obvious relationship. In contrast, in the right-hand panel of Figure 7.12, the predictors `limit` and `rating` are very highly correlated with each other, and we say that they are *collinear*.

The presence of collinearity can pose problems in the regression context, since it can be difficult to separate out the individual effects of collinear variables on the response. In other words, since `limit` and `rating` tend to increase or decrease together, it can be difficult to determine how each one separately is associated with the response, `balance`.

Since collinearity reduces the accuracy of the estimates of the regression coefficients, it causes the standard error for  $\hat{\beta}_j$  to grow. The following example illustrates this.

**Example 7.4.16**

Table 7.6 compares the coefficient estimates obtained from two separate multiple regression models. The first is a regression of `balance` on `age` and `limit`, and the second is a regression of `balance` on `rating` and `limit`. In the first regression, both `age` and `limit` are highly significant with very small p-values. In the second, the collinearity between `limit` and `rating` has caused the standard error for the `limit` coefficient estimate to increase by a factor of 12 and the p-value to increase to 0.701. In other words, the importance of the `limit` variable has been masked due to the presence of collinearity. ◀

		Coefficient	Std.error	t-statistic	p-value
Model 1	Intercept	-173.411	43.828	-3.957	< 0.0001
	age	-2.292	0.672	-3.407	0.0007
	limit	0.173	0.005	34.496	< 0.0001
Model 2	Intercept	-377.537	45.254	-8.343	< 0.0001
	rating	2.202	0.952	2.312	0.0213
	limit	0.025	0.064	0.384	0.7012

**Table 7.6.**: The results for two multiple regression models involving the `Credit` data set are shown. Model 1 is a regression of `balance` on `age` and `limit`, and Model 2 a regression of `balance` on `rating` and `limit`. The standard error of  $\hat{\beta}_{\text{limit}}$  increases 12-fold in the second regression, due to collinearity.

Recall that the  $t$ -statistic for each predictor is calculated by dividing  $\hat{\beta}_j$  by its standard error. Consequently, collinearity results in a decline in the  $t$ -statistic. As a result, in the presence of collinearity, we may *fail* to reject  $H_0 : \beta_j = 0$  though we should do so. This means that the *power* of the hypothesis test - the probability of correctly detecting a *non-zero* coefficient - is reduced by collinearity.

To avoid such a situation, it is desirable to identify and address potential collinearity problems while fitting the model. A simple way to detect collinearity is to look at the correlation matrix of the predictors. An element of this matrix that is large in absolute value (that is near 1) indicates a pair of highly correlated variables, and therefore a collinearity problem in the data.

**Example 7.4.17**

The following Python-output displays the correlation matrix for the `Credit` data set.

(to R)

```
[1]: import pandas as pd
```

```
# Load data
```

## Chapter 7. Multiple Linear Regression

```

df = pd.read_csv(
    '.../.../Themen/Einfache_Lineare_Regression/Daten/Credit.csv')
# Drop all quantitative columns
df = df.drop(['Unnamed: 0', 'Gender', 'Student', 'Married', 'Ethnicity'],
             axis=1)

# Find the correlation Matrix using DataFrame.corr()
print(round(df.corr(), 4))

```

	Income	Limit	Rating	Cards	Age	Education	Balance
Income	1.0000	0.7921	0.7914	-0.0183	0.1753	-0.0277	0.4637
Limit	0.7921	1.0000	0.9969	0.0102	0.1009	-0.0235	0.8617
Rating	0.7914	0.9969	1.0000	0.0532	0.1032	-0.0301	0.8636
Cards	-0.0183	0.0102	0.0532	1.0000	0.0429	-0.0511	0.0865
Age	0.1753	0.1009	0.1032	0.0429	1.0000	0.0036	0.0018
Education	-0.0277	-0.0235	-0.0301	-0.0511	0.0036	1.0000	-0.0081
Balance	0.4637	0.8617	0.8636	0.0865	0.0018	-0.0081	1.0000

From the Python-output we read off that the correlation coefficient between **limit** and **age** is 0.101 which corresponds to a rather weak correlation. On the other hand, we find for the correlation between **limit** and **rating** a value of 0.997 which is very large. ◀

Unfortunately, not all collinearity problems can be detected by inspection of the correlation matrix. The correlation matrix reveals only correlation between *two* variables. However, it is possible for collinearity to exist between three or more variables even if no pair of variables has a particularly high correlation. We call this situation *multicollinearity*.

Instead of inspecting the correlation matrix, a better way to assess multicollinearity is to compute the *variance inflation factor* (VIF). The VIF is the ratio of the variance of  $\hat{\beta}_j$  when fitting the full model divided by the variance of  $\hat{\beta}_j$  if fit on its own. The smallest possible value for the VIF is 1, which indicates the complete absence of collinearity. Typically in practice there is small amount of collinearity among the predictors. As a rule of thumb, a VIF value that exceeds 5 or 10 indicates a problematic amount of collinearity. The VIF for each variable can be computed using the formula

$$\text{VIF}(\hat{\beta}_j) = \frac{1}{1 - R_{X_j|X_{-j}}^2}$$

where  $R_{X_j|X_{-j}}^2$  is the  $R^2$  from a regression of  $X_j$  onto all of the other predictors. If  $R_{X_j|X_{-j}}^2$  is close to one, then collinearity is present, and so the VIF will be large.

### Example 7.4.18

In the `Credit` data, a regression of `balance` on `age`, `rating`, and `limit` indicates that the predictors have VIF values of 1.01, 160.67, and 160.59. As we suspected, there is considerable collinearity in the data.

(to R)

```
[2]: import numpy as np
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Define the linear model:
x = pd.DataFrame({
    'Age' : df['Age'],
    'Rating' : df['Rating'],
    'Limit' : df['Limit']})
y = df['Balance']

# VIF Analysis
x_c = sm.add_constant(x)
VIF = []
for i in range(1,4):
    VIF.append(variance_inflation_factor(x_c.to_numpy(), i))

print(list(x.columns), '\n', np.round(VIF, 3))

# R Squared for 'complete' system
# Fit models
x_sm = sm.add_constant(x)
model = sm.OLS(y, x_sm).fit()

print('\nRsquared for the complete model is given by:\n',
      np.round(model.rsquared, 4))
```

[ 'Age', 'Rating', 'Limit']  
[ 1.011 160.668 160.593]

Rsquared for the complete model is given by:  
0.7536



When faced with the problem of collinearity, there are two simple solutions. The first is to drop one of the problematic variables from the regression. This can usually be done without much compromise to the regression fit, since the presence of collinearity implies that the information that this variable provides about the response is redundant in the presence of the other variables.

### Example 7.4.19

For instance, if we regress `balance` onto `age` and `limit`, without the `rating` predictor, then the resulting VIF values are close to the minimum possible value of 1, and the  $R^2$  drops from 0.754 to 0.75.

[\(to R\)](#)

```
[3]: # Follows Example 4.18
# Define the linear model:
x = x.drop('Rating', axis=1, errors='ignore')

# Fit models
model = sm.OLS(y, x_sm).fit()

# Print result
print('\n Rsquared without \'Rating\' is given by:\n',
      np.round(model.rsquared, 4))
```

```
Rsquared without 'Rating' is given by:
0.7498
```

So dropping `rating` from the set of predictors has effectively solved the collinearity problem without compromising the fit.



The second solution is to combine the collinear variables together into a single predictor. For instance, we might take the average of standardized versions of `limit` and `rating` in order to create a new variable that measures `credit worthiness`.

A third possibility consists of applying principle component analysis, see Chapter 3.5 on page 81.

## 7.5. The Marketing Plan

We now return to the seven questions about the `Advertising` data that we set out to answer in Chapter 5. To this goal, we have again a look at the [Python](#)-output of example 7.3.1.

[\(to R\)](#)

```
[1]: import pandas as pd
import statsmodels.api as sm

# Load data
df = pd.read_csv(
```

## Chapter 7. Multiple Linear Regression

```
' ../../Themen/Einfache_Lineare_Regression/Daten/Advertising.csv')
x = df[['TV', 'radio', 'newspaper']]
y = df['sales']

# Fit Model:
x_sm = sm.add_constant(x)
model = sm.OLS(y, x_sm).fit()

# Print summary including F-Statistic
print(model.summary())
```

OLS Regression Results

	Dep. Variable:	sales	R-squared:	0.897		
Model:	OLS	Adj. R-squared:	0.896			
Method:	Least Squares	F-statistic:	570.3			
Date:	Fri, 22 Jan 2021	Prob (F-statistic):	1.58e-96			
Time:	13:24:42	Log-Likelihood:	-386.18			
No. Observations:	200	AIC:	780.4			
Df Residuals:	196	BIC:	793.6			
Df Model:	3					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	2.9389	0.312	9.422	0.000	2.324	3.554
TV	0.0458	0.001	32.809	0.000	0.043	0.049
radio	0.1885	0.009	21.893	0.000	0.172	0.206
newspaper	-0.0010	0.006	-0.177	0.860	-0.013	0.011
Omnibus:	60.414	Durbin-Watson:			2.084	
Prob(Omnibus):	0.000	Jarque-Bera (JB):			151.241	
Skew:	-1.327	Prob(JB):			1.44e-33	
Kurtosis:	6.332	Cond. No.			454.	

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

### 1. Is there a relationship between **sales** and advertising budget?

This question can be answered by fitting a multiple regression model of **sales** onto **TV**, **radio**, and **newspaper**, that is

$$\text{sales} = \beta_0 + \beta_1 \cdot \text{TV} + \beta_2 \cdot \text{radio} + \beta_3 \cdot \text{newspaper} + \varepsilon$$

Next, we have to test the hypothesis

$$H_0 : \beta_{\text{TV}} = \beta_{\text{radio}} = \beta_{\text{newspaper}} = 0$$

In section 7.3.1 we showed that the F-statistic can be used to determine whether or not we should reject this null hypothesis. In this case the p-value corresponding to the F-statistic in the Python-output is very low, indicating clear evidence of a relationship between advertising and sales.

## Chapter 7. Multiple Linear Regression

### 2. How strong is the relationship between **sales** and advertising budget?

We discussed two measures of model accuracy in Section 6.2.1. First, the RSE estimates the average deviation of the response from the population regression line. For the **Advertising** data, the RSE is 1.681 units while the mean value for the response is 14.022, indicating a percentage error of roughly 12 %. Second, the  $R^2$  statistic records the percentage of variability in the response that is explained by the predictors. The predictors explain almost 90 % of the variance in **sales**. The RSE and  $R^2$  statistics are displayed in the **Python**-output .

### 3. Which media contribute to **sales**?

To answer this question, we can examine the p-values associated with each predictor's t-statistic (see Section 5.3.2). In the multiple linear regression displayed in the above **Python**-output , the p-values for **TV** and **radio** are low, but the p-value for **newspaper** is not. This suggests that only **TV** and **radio** are related to **sales**. In Chapter 8 we explore this question in greater detail.

### 4. How large is the effect of each medium on **sales**?

We saw in Section 5.3.2 that the standard error of  $\hat{\beta}_j$  can be used to construct confidence intervals for  $\beta_j$ . For the **Advertising** data, the 95 % confidence intervals are as follows:

(to R)

```
[2] : # Print Model parameters
      print(model.conf_int(alpha=0.05, cols=None))
```

	0	1
const	2.323762	3.554016
TV	0.043014	0.048516
radio	0.171547	0.205513
newspaper	-0.012616	0.010541

Thus for the **Advertising** data, the 95 % confidence intervals are as follows: [0.043, 0.049] for **TV**, [0.172, 0.206] for **radio**, and [-0.013, 0.011] for **newspaper**. The confidence intervals for **TV** and **radio** are narrow and far from zero, providing evidence that these media are related to **sales**. But the interval for **newspaper** includes zero, indicating that the variable is not statistically significant given the values of **TV** and **radio**.

We saw in Section 7.4.4 that collinearity can result in very wide standard errors. Could collinearity be the reason that the confidence interval associated with **newspaper** is so wide? The VIF scores are 1.005, 1.145, and 1.145 for **TV**, **radio**, and **newspaper**, suggesting no evidence of collinearity.

(to R)

## Chapter 7. Multiple Linear Regression

```
[3]: import numpy as np
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor

# VIF Analysis
x_c = sm.add_constant(x)
VIF = []
for i in range(1,4):
    VIF.append(variance_inflation_factor(x_c.to_numpy(), i))

print(list(x.columns), '\n', np.round(VIF, 3))
```

['TV', 'radio', 'newspaper']  
[1.005 1.145 1.145]

In order to assess the association of each medium individually on **sales**, we can perform three separate simple linear regressions. Results are shown in Table 7.1 on page 176. There is evidence of an extremely strong association between **TV** and **sales** and between **radio** and **sales**. There is evidence of a mild association between **newspaper** and **sales**, when the values of **TV** and **radio** are ignored.

### 5. How accurately can we predict future **sales**?

The response can be predicted using

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \dots + \dots + \hat{\beta}_p x_p$$

The accuracy associated with this estimate depends on whether we wish to predict an individual response,  $Y = f(X_1, \dots, X_p) + \varepsilon$ , or the average response,  $\bar{Y}$  (see Section 7.3). If the former, we use a prediction interval, and if the latter, we use a confidence interval. Prediction intervals will always be wider than confidence intervals because they account for the uncertainty associated with  $\varepsilon$ , the irreducible error.

### 6. Is the relationship linear?

In Section 6, we saw that residual plots can be used in order to identify non-linearity. If the relationships are linear, then the residual plots should display no pattern. In the case of the **Advertising** data, we observe a non-linear effect in Figure 7.4, though this effect could also be observed in a residual plot. In Section 7.4.3, we discussed the inclusion of transformations of the predictors in the linear regression model in order to accommodate non-linear relationships.

### 7. Is there synergy among the advertising media?

The standard linear regression model assumes an additive relationship between the predictors and the response. An additive model is easy to interpret because

the effect of each predictor on the response is unrelated to the values of the other predictors.

However, the additive assumption may be unrealistic for certain data sets. In Section 7.4.3 we showed how to include an interaction term in the regression model in order to accomodate non-additive relationships. A small p-value associated with the interaction term indicates the presence of such relationships.

Figure 7.4 suggested that the **Advertising** data may not be additive. Including an interaction term in the model results in a substantial increase in  $R^2$ , from around 90 % to almost 97 % (see example 7.4.11).

## Conceptual Objectives

You should be able...

- to set up a multiple linear regression model for a given data set and research question.
- to explain how the regression coefficients are estimated.
- to explain the meaning of the *F-statistic* and how it is related to the question (hypothesis test) whether there is a relationship between the response and predictors.
- to decide on the basis of the p-value of the associated F-test whether there is at least one predictor that is related to the response.
- to explain how RSE and  $R^2$  measure the model fit in a multiple regression model.
- to explain the difference between a confidence interval and a prediction interval for the response in a multiple regression model.
- to explain how to test the null hypothesis that a small model is sufficient to explain the data against the alternative hypothesis that a more complex model is required by means of an analysis of variance.
- to explain how qualitative predictors can be encoded in terms of dummy variables.
- to interpret the regression coefficients in a multiple regression model with qualitative predictors with respect to the baseline.
- to extend a linear regression model by means of interaction terms and/or by adding transformed variable terms and to interpret these terms.

- to diagnose violations of the linear regression model assumptions and take the appropriate therapeutical measures.
- to explain the potential problem of collinearity in a multiple regression problem.

## Computational Objectives

You should be able...

- to determine the coefficients of a multiple linear regression model by means of `lm()`
- to determine the p-value associated with the  $F$ -statistic by means of `summary(lm())`
- to determine the confidence and prediction interval for a multiple regression model by means of `predict(lm())`
- to use `anova()` to test the null hypothesis that a small model is sufficient to explain the data against the alternative hypothesis that the more complex model is required.
- to use `drop1()` to decide whether the quality of a model changes when one predictor variable is omitted.
- to use `contrasts()` in order to view the coding scheme for the qualitative predictors.
- to use `contr.treatment()` to change the coding scheme for the qualitative predictors.
- to detect collinearity in the predictors of multiple linear regression model by means of `vif()`

# Chapter 8.

## Linear Model Selection<sup>1</sup>

### 8.1. Introduction

In the regression setting, the standard linear model

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p + \varepsilon \quad (8.1)$$

is commonly used to describe the relationship between a response  $Y$  and a set of predictor variables  $X_1, X_2, \dots, X_p$ . We have seen in Chapter 5 that one typically fits this model using least squares. The linear model has distinct advantages in terms of inference and, on real-world problems is often surprisingly competitive in relation to more complicated methods.

Why might we want to use another fitting procedure instead of least squares? First, as we have seen in Chapter 7.4.3, the linearity and additivity assumptions turn out to be very often too restrictive. As we will see in this chapter, alternative fitting procedures can yield better *prediction accuracy* and *model interpretability*.

- *Predictive Accuracy*

Provided that the true relationship between the response  $Y$  and the predictors  $X_1, X_2, \dots, X_p$  is approximately linear, the least squares estimates will have low bias. If  $n$ , the number of observations, is much larger than  $p$ , the number of predictor variables, then the least squares estimates tend to also have low variance, and hence will perform well on test observations.

However, if  $n$  is not much larger than  $p$ , then there can be a lot of variability in the least squares fit, resulting in overfitting and consequently poor predictions on future observations not used in model training.

And if  $p > n$ , then there is no longer a unique least squares coefficient estimate: the variance is *infinite* so the method cannot be used at all.

---

<sup>1</sup>This chapter follows Chapter 6 of the text book *An Introduction to Statistical Learning: with Applications in R* by G. James et al., Springer Texts in Statistics 2013.

By *constraining* or *shrinking* the estimated coefficients, we can often substantially reduce the variance at the cost of a negligible increase in bias. This can lead to substantial improvements in the accuracy with which we can predict the response for observations not used in model training.

- *Model Interpretability*

It is often the case that some or many of the variables used in a multiple regression model are in fact not associated with the response. Including such *irrelevant* variables leads to unnecessary complexity in the resulting model. By removing these variables - that is, by setting the corresponding coefficient estimates to zero - we can obtain a model that is more easily interpreted.

### Example 8.1.1

For the **Advertising** data we have seen that the predictor **newspaper** does not have any influence on **sales**. In addition to the advantage of reducing complexity, we also have commercial interests to drop predictors from the model: by stopping expenditure in **newspaper** advertising, we can save money. But how do we know for sure that by dropping **newspaper** we really have come up with the *best* model? ◀

Now least squares is extremely unlikely to yield any coefficient estimates that are exactly zero. In this chapter, we see some approaches for automatically performing *feature selection* or *variable selection* - that is, for excluding irrelevant variables from a multiple regression model.

## 8.2. Subset Selection

### 8.2.1. Stepwise Selection

*Forward stepwise selection* is a computationally efficient method to select a subset of predictor variables.

Forward stepwise selection begins with a model containing no predictors, and then adds predictors to the model, one-at-a-time, until all of the predictors are in the model. In particular, at each step the variable that gives the greatest *additional* improvement to the fit is added to the model. We want to illustrate this procedure in the following example.

### Example 8.2.1

For the `Credit` example, we begin with the so-called *null model*  $\mathcal{M}_0$ , which contains no predictors:

$$\mathbf{balance} = \beta_0 + \varepsilon$$

Then, we add a predictor variable to the null model. For this example, we will write two `Python`-functions, one to fit a linear model and return important scoring metrics and one to add one predictor to a model, based on the returned scores. ([to R](#))

```
[1]: import pandas as pd
import numpy as np
import statsmodels.api as sm

# Load data
df = pd.read_csv(
    '../.../Themen/Lineare_Modell_Auswahl/Daten/Credit.csv')

# Convert Categorical variables
df = pd.get_dummies(data=df, drop_first=True,
                     prefix=('Gender_', 'Student_',
                             'Married_', 'Ethnicity_'))

x_full = df.drop(columns='Balance')
y = df['Balance']

def fit_linear_reg(x, y):
    '''Fit Linear model with predictors x on y
    return AIC, BIC, R2 and R2 adjusted'''
    x = sm.add_constant(x)
    # Create and fit model
    model_k = sm.OLS(y, x).fit()

    # Find scores
    BIC = model_k.bic
    AIC = model_k.aic
    R2 = model_k.rsquared
    R2_adj = model_k.rsquared_adj
    RSS = model_k.ssr

    # Return result in Series
    results = pd.Series(data={'BIC': BIC, 'AIC': AIC, 'R2': R2,
                               'R2_adj': R2_adj, 'RSS': RSS})

    return results

def add_one(x_full, x, y, scoreby='RSS'):
    '''Add possible predictors from x_full to x,
    Fit a linear model on y using fit_linear_reg
    Returns Dataframe showing scores as well as best model'''
```

## Chapter 8. Linear Model Selection

```

# Predefine DataFrame
x_labels = x_full.columns
zeros = np.zeros(len(x_labels))
results = pd.DataFrame(
    data={'Predictor': x_labels.values, 'BIC': zeros,
          'AIC': zeros, 'R2': zeros,
          'R2_adj': zeros, 'RSS': zeros})

# For every predictor find R^2, RSS, and AIC
for i in range(len(x_labels)):
    x_i = np.concatenate((x, [np.array(x_full[x_labels[i]])]))
    results.iloc[i, 1:] = fit_linear_reg(x_i.T, y)

# Depending on where we scoreby, we select the highest or lowest
if scoreby in ['RSS', 'AIC', 'BIC']:
    best = x_labels[results[scoreby].argmin()]
elif scoreby in ['R2', 'R2_adj']:
    best = x_labels[results[scoreby].argmax()]

return results, best

# Define the empty predictor
x_empty = [np.zeros(len(y))]

results, best1 = add_one(x_full, x_empty, y)

print(results[['Predictor', 'AIC', 'R2', 'RSS']],
      '\n\nBest predictor is:', best1)

```

	Predictor	AIC	R2	RSS
0	Unnamed: 0	6042.696603	0.000037	8.433681e+07
1	Income	5945.894250	0.214977	6.620874e+07
2	Limit	5499.982633	0.742522	2.171566e+07
3	Rating	5494.781548	0.745848	2.143512e+07
4	Cards	6039.710202	0.007475	8.370950e+07
5	Age	6042.709965	0.000003	8.433963e+07
6	Education	6042.685316	0.000065	8.433443e+07
7	Gender_Male	6042.526817	0.000461	8.430102e+07
8	Student_Yes	6014.932656	0.067090	7.868154e+07
9	Married_Yes	6042.698437	0.000032	8.433720e+07
10	Ethnicity_Asian	6042.672799	0.000096	8.433179e+07
11	Ethnicity_Caucasian	6042.706987	0.000011	8.433900e+07

Best predictor is: Rating

We now choose the *best* variable in the sense that adding this variable leads to the regression model with the lowest RSS or the highest  $R^2$ . The variable that results in the model with the lowest RSS is in this case **rating**.

## Chapter 8. Linear Model Selection

Thus, we have found the model  $\mathcal{M}_1$

$$\text{balance} = \beta_0 + \beta_1 \cdot \text{rating} + \varepsilon$$

We now add a further predictor variable to this model by first updating the reference model and removing the chosen predictor from the set of possible predictors. Subsequently, we can run the same procedure and decide which predictor variable to add next.

(to R)

```
[2]: # Update the empty predictor with the best predictor
x_1 = [df[best1]]
# Remove the chosen predictor from the list of options
x_red = x_full.drop(columns=best1, errors='ignore')

results, best2 = add_one(x_red, x_1, y)

print(results[['Predictor', 'AIC', 'R2', 'RSS']],
      '\n\nBest predictor is:', best2)
```

	Predictor	AIC	R2	RSS
0	Unnamed: 0	5496.518489	0.746016	2.142103e+07
1	Income	5212.557085	0.875118	1.053254e+07
2	Limit	5496.632982	0.745943	2.142716e+07
3	Cards	5494.187124	0.747492	2.129654e+07
4	Age	5484.481339	0.753545	2.078601e+07
5	Education	5496.272851	0.746171	2.140788e+07
6	Gender_Male	5496.481640	0.746039	2.141906e+07
7	Student_Yes	5372.232473	0.813849	1.569996e+07
8	Married_Yes	5494.569548	0.747250	2.131691e+07
9	Ethnicity_Asian	5496.067431	0.746302	2.139689e+07
10	Ethnicity_Caucasian	5496.772749	0.745854	2.143465e+07

Best predictor is: Income

We select again the variable that leads when added to the reference model to the lowest RSS. In this case, we select the predictor variable **income** which gives us the model  $\mathcal{M}_2$ :

$$\text{balance} = \beta_0 + \beta_1 \cdot \text{rating} + \beta_2 \cdot \text{income} + \varepsilon$$

This procedure will be repeated. In particular, we will add one variable among the remaining  $p - 2$  variables to the model  $\mathcal{M}_2$ . The resulting model with the lowest RSS will become model  $\mathcal{M}_3$ .

(to R)

```
[3]: # Update the empty predictor with the best predictor
x_2 = np.concatenate((x_1, [df[best2]]))
# Remove the chosen predictor from the list of options
```

## Chapter 8. Linear Model Selection

```
x_red = x_red.drop(columns=best2, errors='ignore')

results, best3 = add_one(x_red, x_2, y)

print(results[['Predictor', 'AIC', 'R2', 'RSS']],
      '\n\nBest predictor is:', best3)
```

	Predictor	AIC	R2	RSS
0	Unnamed: 0	5214.551863	0.875120	1.053240e+07
1	Limit	5210.950291	0.876239	1.043800e+07
2	Cards	5214.477534	0.875143	1.053045e+07
3	Age	5211.113461	0.876188	1.044226e+07
4	Education	5213.765645	0.875365	1.051172e+07
5	Gender_Male	5214.521087	0.875129	1.053159e+07
6	Student_Yes	4849.386992	0.949879	4.227219e+06
7	Married_Yes	5210.930247	0.876245	1.043747e+07
8	Ethnicity_Asian	5212.042074	0.875901	1.046653e+07
9	Ethnicity_Caucasian	5213.976405	0.875299	1.051726e+07

Best predictor is: Student\_\_Yes

In this case, the predictor **student** turns out to be the variable that we add to model  $\mathcal{M}_2$  to obtain model  $\mathcal{M}_3$ :

$$\text{balance} = \beta_0 + \beta_1 \cdot \text{rating} + \beta_2 \cdot \text{income} + \beta_3 \cdot \text{student} + \varepsilon$$

We will end up with the following models:  $\mathcal{M}_0, \mathcal{M}_1, \dots, \mathcal{M}_{10}$ .

But how are we going to identify the *best* model among these 11 models? We may base our decision on the value of the AIC which is listed in the [Python](#) output. This statistic allows us to compare different models with each other. We will discuss the AIC in Section 8.2.4 in greater detail.

The procedure we have followed can also be automated using [sklearns.feature\\_selection](#). However, this is a relatively new package, which is not as flexible or extensive yet.

(to R)

```
[4]: from sklearn.feature_selection import SequentialFeatureSelector
from sklearn.linear_model import LinearRegression

# define Linear Regression Model in sklearn
linearmodel = LinearRegression()
# Sequential Feature Selection using sklearn
sfs = SequentialFeatureSelector(linearmodel, n_features_to_select=3,
                                 direction='forward')
sfs.fit(x_full, y)

# Print Chosen variables
print(x_full.columns[sfs.support_].values)
```

```
[ 'Income' 'Rating' 'Student__Yes' ]
```

The model with three predictor variables includes the variables `income`, `rating` and `student`.



Formally, the forward stepwise selection procedure can be written as follows:

### **Algorithm: Forward stepwise selection**

1. Let  $\mathcal{M}_0$  denote the *null* model, which contains no predictors.
2. For  $k = 0, \dots, p - 1$ :
  - a) Consider all  $p - k$  models that augment the predictors in  $\mathcal{M}_k$  with one additional predictor.
  - b) Choose the *best* among these  $p - k$  models, and call it  $\mathcal{M}_{k+1}$ . Here “best” is defined as having smallest RSS or highest  $R^2$ .
3. Select a single best model from among  $\mathcal{M}_0, \dots, \mathcal{M}_p$  using  $C_p$ , AIC, BIC or adjusted  $R^2$ .

Forward stepwise selection can be applied even in the high-dimensional setting where  $n < p$ ; however, in this case, it is possible to construct submodels  $\mathcal{M}_0, \mathcal{M}_1, \dots, \mathcal{M}_{n-1}$  only, since each submodel is fit using least squares, which will not yield a unique solution if  $p \geq n$ .

Though forward stepwise selection tends to do well in practice, it is not guaranteed to find the best possible model out of all  $2^p$  models containing subsets of the  $p$  predictors. For instance, suppose that in a given data set with  $p = 3$  predictors, the best possible one-variable model contains  $X_1$ , and the best possible two-variable model instead contains  $X_2$  and  $X_3$ . Then forward stepwise selection will fail to select the best possible two-variable model, because  $\mathcal{M}_1$  will contain  $X_1$ , so  $\mathcal{M}_2$  must also contain  $X_1$  together with one additional variable.

### **8.2.2. Backward Stepwise Selection**

Like forward stepwise selection, *backward stepwise selection* provides an efficient method to select a subset of predictors. However, unlike forward stepwise selection, it begins with the full least squares model containing all  $p$  predictors, and then iteratively removes the least useful predictor, one-at-a-time.

We will illustrate the procedure in the following example.

### Example 8.2.2

We begin with the *full model*, that is  $\mathcal{M}_{10}$ , which contains all  $p$  predictors of the **Credit** data set

$$\begin{aligned}\text{balance} = & \beta_0 + \beta_1 \cdot \text{income} + \beta_2 \cdot \text{limit} + \beta_3 \cdot \text{rating} + \beta_4 \cdot \text{cards} \\ & + \beta_5 \cdot \text{age} + \beta_6 \cdot \text{education} + \beta_7 \cdot \text{gender} + \beta_8 \cdot \text{student} \\ & + \beta_9 \cdot \text{married} + \beta_{10} \cdot \text{ethnicity} + \varepsilon\end{aligned}$$

Then we remove one predictor variable from the model. We create a new function, similar to the `add_one` function, which removes each predictor separately from the full model. All created functions will be saved in `LMS_def`, and imported when needed.

(to R)

```
[1]: import pandas as pd
import numpy as np
from LMS_def import *

# Load data
df = pd.read_csv(
    '../.../Themen/Lineare_Modell_Auswahl/Daten/Credit.csv')

# Convert Categorical variables
df = pd.get_dummies(data=df, drop_first=True,
                     prefix=('Gender_', 'Student_',
                             'Married_', 'Ethnicity_'))

x_full = df.drop(columns='Balance')
y = df['Balance']

def drop_one(x, y, scoreby='RSS'):
    ''' Remove possible predictors from x,
    Fit a linear model on y using fit_linear_reg
    Returns Dataframe showing scores as well as predictor
    to drop in order to keep the best model '''
    # Predefine DataFrame
    x_labels = x.columns
    zeros = np.zeros(len(x_labels))
    results = pd.DataFrame(
        data={'Predictor': x_labels.values, 'BIC': zeros,
              'AIC': zeros, 'R2': zeros,
              'R2_adj': zeros, 'RSS': zeros})

    # For every predictor find RSS and R^2
    for i in range(len(x_labels)):
        x_i = x.drop(columns=x_labels[i])
        results.iloc[i, 1:] = fit_linear_reg(x_i, y)

    # Depending on where we scoreby, we select the highest or lowest
```

## Chapter 8. Linear Model Selection

```

if scoreby in ['RSS', 'AIC', 'BIC']:
    worst = x_labels[results[scoreby].argmin()]
elif scoreby in ['R2', 'R2_adj']:
    worst = x_labels[results[scoreby].argmax()]

return results, worst

results, worst1 = drop_one(x_full, y)

print(results[['Predictor', 'AIC', 'R2', 'RSS']],
      '\n\nWorst predictor is:', worst1)

```

	Predictor	AIC	R2	RSS
0	Unnamed: 0	4821.370391	0.955102	3.786730e+06
1	Income	5361.643903	0.826689	1.461702e+07
2	Limit	4853.911857	0.951296	4.107672e+06
3	Rating	4826.005381	0.954578	3.830864e+06
4	Cards	4837.510427	0.953253	3.942650e+06
5	Age	4825.143746	0.954676	3.822621e+06
6	Education	4820.935924	0.955150	3.782619e+06
7	Gender__Male	4821.391825	0.955099	3.786933e+06
8	Student__Yes	5214.259751	0.880104	1.011204e+07
9	Married__Yes	4821.188761	0.955122	3.785011e+06
10	Ethnicity__Asian	4821.918353	0.955040	3.791921e+06
11	Ethnicity__Caucasian	4821.043224	0.955138	3.783634e+06

Worst predictor is: Education

Now we remove the *least useful* variable which is the one that yields the reduced regression model with the lowest RSS or the highest  $R^2$ . This predictor represents the most *redundant* variable because its removal improves the model most significantly with respect to the RSS. In this case, this is the predictor `education`.

Thus, we obtain the model  $\mathcal{M}_9$  which is given by

$$\begin{aligned} \text{balance} = & \beta_0 + \beta_1 \cdot \text{income} + \beta_2 \cdot \text{limit} + \beta_3 \cdot \text{rating} + \beta_4 \cdot \text{cards} \\ & + \beta_5 \cdot \text{age} + \beta_6 \cdot \text{gender} + \beta_7 \cdot \text{student} + \beta_8 \cdot \text{married} \\ & + \beta_9 \cdot \text{ethnicity} + \varepsilon \end{aligned}$$

We now remove another variable from this model. To do so we first need to update the reference model which means dropping the selected predictor from the full set. Subsequently, we run the same procedure again.

(to R)

```
[2]: # Remove the chosen predictor from the list of options
x_red1 = x_full.drop(columns=worst1, errors='ignore')

results, worst2 = drop_one(x_red1, y)
```

## Chapter 8. Linear Model Selection

```
print(results[['Predictor', 'AIC', 'R2', 'RSS']],
      '\n\nWorst predictor is:', worst2)
```

	Predictor	AIC	R2	RSS
0	Unnamed: 0	4819.857603	0.955047	3.791345e+06
1	Income	5359.644014	0.826689	1.461703e+07
2	Limit	4851.966157	0.951290	4.108230e+06
3	Rating	4824.775477	0.954491	3.838246e+06
4	Cards	4835.976056	0.953198	3.947242e+06
5	Age	4823.685440	0.954615	3.827801e+06
6	Gender_Male	4819.862964	0.955046	3.791396e+06
7	Student_Yes	5213.014039	0.879877	1.013112e+07
8	Married_Yes	4819.758863	0.955058	3.790410e+06
9	Ethnicity_Asian	4820.411139	0.954985	3.796596e+06
10	Ethnicity_Caucasian	4819.562597	0.955080	3.788550e+06

Worst predictor is: Ethnicity\_Caucasian

We choose the model that has the smallest RSS. This turns out to be the case when we remove the predictor variable **Ethnicity\_Caucasian** from the reference model. We then obtain the model  $\mathcal{M}_8$ :

$$\begin{aligned} \text{balance} = & \beta_0 + \beta_1 \cdot \text{income} + \beta_2 \cdot \text{limit} + \beta_3 \cdot \text{rating} + \beta_4 \cdot \text{cards} \\ & + \beta_5 \cdot \text{age} + \beta_6 \cdot \text{gender} + \beta_7 \cdot \text{student} \\ & + \beta_8 \cdot \text{Married} + \varepsilon \end{aligned}$$

We iterate this procedure until *no* predictor variable is left in the regression model. This iterative procedure yields 11 different models  $\mathcal{M}_0, \mathcal{M}_1, \dots, \mathcal{M}_{10}$ . We identify the *best* among these models on the basis of the AIC, which we will discuss in section 8.2.4.

The selection procedure can also be performed using `sklearn.feature_selection`, similar to 8.2.1. (to R)

```
[3]: from sklearn.feature_selection import SequentialFeatureSelector
      from sklearn.linear_model import LinearRegression

      # define Linear Regression Model in sklearn
      linearmodel = LinearRegression()
      # Sequential Feature Selection using sklearn
      sfs = SequentialFeatureSelector(linearmodel, n_features_to_select=3,
                                      direction='backward')
      sfs.fit(x_full, y)

      # Print Chosen variables
      print(x_full.columns[sfs.support_].values)
```

```
[ 'Income' 'Limit' 'Student__Yes' ]
```

The regression model with three predictors contains the variables `income`, `limit` and `student`. This model differs from the model we obtained by forward stepwise selection. In this model the variable `rating` appears instead of the variable `limit`.



Like forward stepwise selection, backward stepwise selection is not guaranteed to yield the *best* model containing a subset of the  $p$  predictors.

Backward selection requires that the number of samples  $n$  is larger than the number of variables  $p$  (so that the full model can be fit). In contrast, forward stepwise selection can be used even when  $n < p$ , and so is the only viable subset method when  $p$  is very large.

Formally the backward stepwise selection algorithm is given by:

### **Algorithm: Backward stepwise selection**

1. Let  $\mathcal{M}_p$  denote the *full* model, which contains all  $p$  predictors.
2. For  $k = p, p - 1, \dots, 1$ :
  - a) Consider all  $k$  models that contain all but one of the predictors in  $\mathcal{M}_k$ , for a total of  $k - 1$  predictors
  - b) Choose the *best* among these  $k$  models, and call it  $\mathcal{M}_{k-1}$ . Here *best* is defined as having smallest RSS or highest  $R^2$ .
3. Select a single best model among  $\mathcal{M}_0, \dots, \mathcal{M}_p$  using  $C_p$ , AIC, BIC or adjusted  $R^2$ .

### **8.2.3. Further Variable Selection Methods**

#### **Best Subset Selection**

To perform *best subset selection*, we fit a separate least squares regression for each possible combination of the  $p$  predictors. That is, we fit all  $p$  models that contain exactly one predictor, all  $p(p - 1)/2$  models that contain exactly two predictors, and so forth. We end up with  $2^p$  different models. We then look at all of the resulting models, with the goal of identifying the one that is the *best*.

Selecting the *best model* from among the  $2^p$  possibilities considered by best subset selection suffers from computational limitations. The number of possible models that must be considered grows rapidly as  $p$  increases. For example if  $p = 10$ , then there

are approximately 1000 possible models to be considered, and if  $p = 20$ , then there are over one million possibilities (1 048 576 to be precise), whereas forward stepwise selection requires fitting only 211 models.

Consequently, best subset selection becomes computationally infeasible for values of  $p$  greater than around 40, even with extremely fast modern computers. There are computational shortcuts - so called branch-and-bound techniques - for eliminating some choices, but these have their limitations as  $p$  gets large.

### Hybrid Approaches

The best subset, forward stepwise, and backward stepwise selection approaches generally give similar but *not* identical models. As another alternative, *hybrid versions* of forward and backward stepwise selection are available, in which variables are added to the model or removed from the model sequentially.

For instance, we start with a model containing  $k$  predictor variables (if no reference model is specified, then we start with the null model). Subsequently, the RSS of all models that result from adding to or removing each variable from the reference model is calculated. We then choose the model with the lowest RSE that results from either adding or removing a variable. We iterate this procedure until the RSE stops decreasing.

Such an approach attempts to more closely mimic best subset selection while retaining the computational advantages of forward and backward stepwise selection.

### 8.2.4. Choosing the Optimal Model

Best subset selection, forward selection, and backward selection result in the creation of a set of models, each of which contains a subset of the  $p$  predictors. In order to implement these methods, we need a way to determine which of these models is *best*.

We could identify the *best* model on the basis of the RSE oder  $R^2$ . However, as we discussed in Section 6.2.1 the model containing all of the predictors will always have the smallest RSS and the largest  $R^2$ . Therefore, RSS and  $R^2$  are not suitable for selecting the best model among a collection of models with different numbers of predictors.

There are a number of techniques for *adjusting* for the model size. These approaches can be used to select among a set of models with different numbers of variables by adding a *penalty* to the RSS when too many variables are included in the model. For instance, if we add a predictor variable to a model then the RSS decreases, but the sum consisting of RSS and penalty *may* increase.

We now consider four such approaches:  $C_p$ , Akaike information criterion (AIC), Bayesian information criterion (BIC), and adjusted  $R^2$ . These approaches differ from each other in the way how the penalty term is defined.

### Adjusted $R^2$

Recall from Chapter 6 that the usual  $R^2$  is defined as

$$R^2 = 1 - \frac{RSS}{TSS}$$

where  $TSS = \sum_i (y_i - \bar{y})^2$  is the *total sum of squares* for the response. Since RSS always decreases as more variables are added to the model, the  $R^2$  always increases as more variables are added. For a least squares model with  $p$  variables, the adjusted  $R^2$  statistic is calculated as

$$\text{adjusted } R^2 = 1 - \frac{RSS/(n-p-1)}{TSS/(n-1)}$$

where  $n$  denotes the number of data points.

A *large* value of adjusted  $R^2$  indicates a model that fits data well. Maximizing the adjusted  $R^2$  is equivalent to minimizing

$$\frac{RSS}{n-p-1}$$

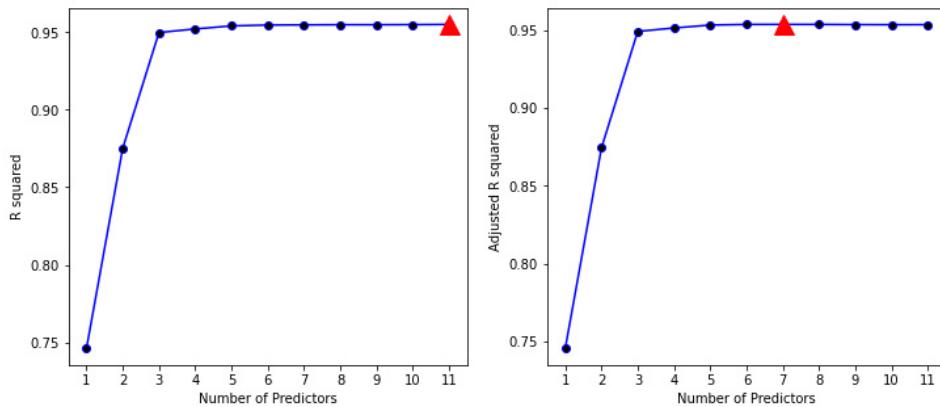
While RSS always decreases as the number of variables in the model increases,  $\frac{RSS}{n-p-1}$  may increase or decrease, due to the presence of the  $p$  in the denominator.

The intuition behind the adjusted  $R^2$  is that once all of the correct variables have been included in the model, adding additional *noise* variables will lead to only a very small decrease in RSS. Since adding noise variables leads to an increase in  $p$ , such variables will lead to an *increase* in  $\frac{RSS}{n-p-1}$ , and consequently a decrease in the adjusted  $R^2$ . Therefore, in theory, the model with the largest adjusted  $R^2$  will have only correct variables and no noise variables. Unlike the  $R^2$  statistic, the adjusted  $R^2$  statistic *pays a price* for the inclusion of unnecessary variables in the model.

### Example 8.2.3

Figure 8.1 displays the  $R^2$  and the adjusted  $R^2$  for the best model of each size produced by forward stepwise selection on the **Credit** data set. Using the adjusted  $R^2$  we select a model that contains seven variables - which however is not easy to detect by eye in the plot.

## Chapter 8. Linear Model Selection



**Figure 8.1.:** Adjusted  $R^2$  is shown for the best models of each size for the `Credit` data set. The models were produced by stepwise forward selection. The adjusted  $R^2$  curve gets rather flat after four variables are included and reaches the maximum for seven predictors.

If we compare the  $R^2$  for the best model of each size as a function of the number of predictors then we observe that the values of  $R^2$  are steadily increasing.

(to R)

```
[1]: import pandas as pd
import numpy as np
from sklearn.feature_selection import SequentialFeatureSelector
from sklearn.linear_model import LinearRegression
import statsmodels.api as sm
from LMS_def import *

# Load data
df = pd.read_csv(
    '.../Themen/Lineare_Modell_Auswahl/Daten/Credit.csv')

# Convert Categorical variables
df = pd.get_dummies(data=df, drop_first=True,
                     prefix=('Gender_', 'Student_',
                             'Married_', 'Ethnicity_'))

x_full = df.drop(columns='Balance')
y = df['Balance']

# define Linear Regression Model in sklearn
linearmodel = LinearRegression()

# Predefine DataFrame
results = pd.DataFrame(data={'BIC': [], 'AIC': [], 'R2': [],
                             'R2_adj': [], 'RSS': []})

# For each number of selected features:
for i in range(1, x_full.shape[1]):
```

## Chapter 8. Linear Model Selection

```
# Sequential Feature Selection using sklearn
sfs = SequentialFeatureSelector(linearmodel, n_features_to_select=i,
                                 direction='forward')
sfs.fit(x_full, y)
chosen_predictors = x_full.columns[sfs.support_].values

# Fit a linear model using the chosen predictors:
results_i = fit_linear_reg(x_full[chosen_predictors], y)
results_i = results_i.rename(str(i))
# Save results
results = results.append(results_i)

print(results)
```

	BIC	AIC	R2	R2_adj	RSS
1	5502.764477	5494.781548	0.745848	0.745210	2.143512e+07
2	5224.531479	5212.557085	0.875118	0.874489	1.053254e+07
3	4865.352851	4849.386992	0.949879	0.949499	4.227219e+06
4	4852.481331	4832.524008	0.952188	0.951703	4.032502e+06
5	4841.615607	4817.666820	0.954161	0.953579	3.866091e+06
6	4842.979215	4815.038963	0.954688	0.953996	3.821620e+06
7	4847.838075	4815.906359	0.954816	0.954009	3.810814e+06
8	4852.927218	4817.004037	0.954918	0.953995	3.802227e+06
9	4858.908583	4818.993938	0.954919	0.953879	3.802131e+06
10	4864.338343	4820.432233	0.954982	0.953825	3.796796e+06
11	4869.086336	4821.188761	0.955122	0.953850	3.785011e+06

This is not the case for the values of the adjusted R<sup>2</sup>. As the Python -output reveals, the adjusted R<sup>2</sup> reaches a maximum for seven predictors, after that the values start decreasing.

Using the adjusted R<sup>2</sup> results in the selection of the following regression model that was produced by forward selection

$$\begin{aligned} \text{balance} = & \beta_0 + \beta_1 \cdot \text{income} + \beta_2 \cdot \text{limit} + \beta_3 \cdot \text{rating} + \beta_4 \cdot \text{cards} \\ & + \beta_5 \cdot \text{age} + \beta_6 \cdot \text{gender} + \beta_7 \cdot \text{student} + \varepsilon \end{aligned}$$



Despite its popularity, and even though it is quite intuitive, the adjusted R<sup>2</sup> is not as well motivated in statistical theory as AIC, BIC and C<sub>p</sub> which have rigorous theoretical justifications in information theory that are beyond the scope of these lecture notes.

### Remarks:

- i. Note that for  $n \gg p$ , the adjusted R<sup>2</sup> tends towards R<sup>2</sup>.



### AIC - Akaike information criterion

The most popular variant among these criteria rooted in information theory is the Akaike Information Criterion (AIC) which considers goodness-of-fit to the data and the complexity of the model, and hence pursues a similar idea as the adjusted  $R^2$ . It is defined as

$$AIC = -2 \log(L) + 2q$$

where  $L$  denotes the value of the likelihood function for a particular model and  $q$  is the number of variables of this model.

In the case of the model 8.1 with Gaussian errors, maximum likelihood and least squares are identical methods. In this case, the likelihood function is driven by RSS, the residual sum of squares, and the AIC of a least squares model containing  $p$  predictors for  $n$  observations is given by

$$AIC = \frac{1}{n\hat{\sigma}^2} (RSS + 2p\hat{\sigma}^2)$$

Hence, the AIC criterion compares the magnitude of the residuals with the complexity of the model that was used and so prevents overfitting. While larger models have the advantage of achieving a lower RSS, the penalization term  $2p\hat{\sigma}^2$  may compensate for the increase in complexity. Obviously, the smaller the AIC, the better the model fits the data.

The  $C_p$  is defined as

$$C_p = \frac{1}{n} (RSS + 2p\hat{\sigma}^2) \tag{8.2}$$

For least squares models,  $C_p$  and  $AIC$  are proportional to each other, and so often  $C_p$  and AIC are used interchangeably. So when determining which of a set of models is best, we choose the model with the lowest  $C_p$  value.

Whereas the  $C_p$  can be used for fitted least squares models<sup>2</sup>, the AIC criterion is defined for a large class of models fit by maximum likelihood.

#### Example 8.2.4

The left-hand panel of Figure 8.2 displays the AIC, the right hand panel the BIC.

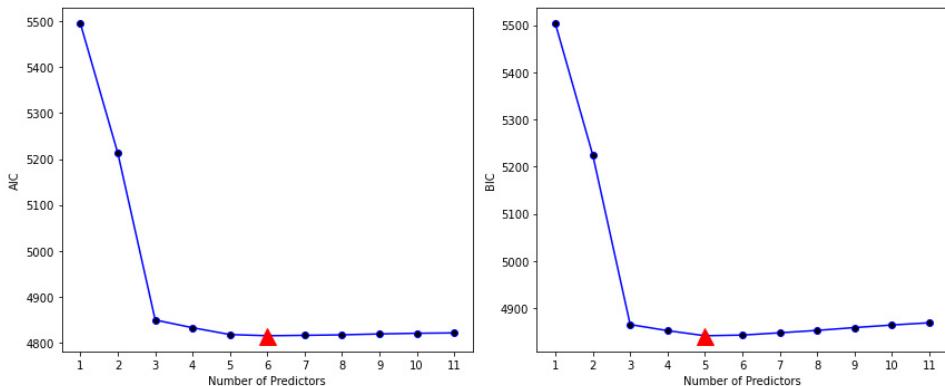
According to the AIC criterion, the best among all models produced by forward step-wise selection contains six predictor variables.

(to R)

---

<sup>2</sup>Though it is beyond the scope of these lecture notes, one can show that if  $\hat{\sigma}^2$  is an unbiased estimate of  $\sigma^2$  in equation 8.2, then  $C_p$  is an unbiased estimate of the test mean squared error (MSE). As a consequence, the  $C_p$  statistic tends to take on a small value for models with a low test error.

## Chapter 8. Linear Model Selection



**Figure 8.2.:** Left: AIC is plotted for the best models of each size (produced by forward stepwise selection) for the `Credit` data set. Right: The BIC is plotted for the best models of each size (produced by forward stepwise selection) for the `Credit` data set.

```
[1]: print(np.argmin(results.loc[:, 'AIC']) + 1)
```

6

Note that we have to add 1 to the index of the minimum value, as the index starts at 0, whereas the number of predictors starts at 1.

The (last) Python -output of example 8.2.1 displays the six predictors of the best model produced by forward selection.

$$\begin{aligned} \text{balance} = & \beta_0 + \beta_1 \cdot \text{income} + \beta_2 \cdot \text{limit} + \beta_3 \cdot \text{rating} + \beta_4 \cdot \text{cards} \\ & + \beta_5 \cdot \text{age} + \beta_6 \cdot \text{student} + \varepsilon \end{aligned}$$



### Example 8.2.5

If we want to find the best among all models produced by stepwise forward selection, that is, if we want to identify the most appropriate number of predictors on the basis of the AIC, we only have to repeat the known procedure, for the number of Predictors. Note that the parameter `scoreby` is now set to AIC.

(to R)

```
[1]: import pandas as pd
import numpy as np
from LMS_def import *

# Load data
df = pd.read_csv(
    '../../../../../Themen/Lineare_Modell_Auswahl/Daten/Credit.csv')
```

## Chapter 8. Linear Model Selection

```
# Convert Categorical variables
df = pd.get_dummies(data=df, drop_first=True,
                     prefix=('Gender_', 'Student_',
                             'Married_', 'Ethnicity_'))

x_full = df.drop(columns='Balance')
y = df['Balance']

results = pd.DataFrame(data={'Best_Pred': [], 'AIC': []})

# Define the empty predictor
x0 = [np.zeros(len(y))]

x = x0
x_red = x_full.copy()

for i in range(x_full.shape[1]):
    results_i, best_i = add_one(x_red, x, y, scoreby='AIC')

    # Update the empty predictor with the best predictor
    x = np.concatenate((x, [df[best_i]]))

    # Remove the chosen predictor from the list of options
    x_red = x_red.drop(columns=best_i)

    # Save results
    results.loc[i, 'Best_Pred'] = best_i
    results.loc[i, 'AIC'] = results_i['AIC'].min()

print('Best Predictors and corresponding AIC:\n', results,
      '\n\nThe best model thus contains',
      results['AIC'].argmin() + 1, 'predictors')
```

Best Predictors and corresponding AIC:

	Best_Pred	AIC
0	Rating	5494.781548
1	Income	5212.557085
2	Student__Yes	4849.386992
3	Limit	4832.524008
4	Cards	4817.666820
5	Age	4815.038963
6	Gender__Male	4815.900560
7	Unnamed: 0	4817.004037
8	Ethnicity__Asian	4818.286694
9	Married__Yes	4819.562597
10	Ethnicity__Caucasian	4820.935924
11	Education	4822.448073

The best model thus contains 6 predictors



## BIC - Bayesian information criterion

An alternative to the AIC is the *Bayesian Information Criterion* (BIC) that was derived by Schwarz (1978) from a Bayesian point of view. The BIC is defined as

$$\text{BIC} = -2 \log(L) + 2 \log(n)q$$

where  $L$  denotes the likelihood function for a particular model and  $q$  is the number of estimated parameters of the model. For the least squares model with  $p$  predictors, the BIC is, up to irrelevant constants, given by

$$\text{BIC} = \frac{1}{n} \left( \text{RSS} + \log(n)p\hat{\sigma}^2 \right)$$

If we compare the AIC with the BIC, then we see that the only difference relates to the penalty term. Like AIC, the BIC will tend to take on a small value for a model that fits well the data, and so generally we select the model that has the lowest BIC value.

Notice that BIC replaces the  $2p\hat{\sigma}^2$  used by AIC with a  $\log(n)p\hat{\sigma}^2$  term, where  $n$  is the number of observations. Since  $\log(n) > 2$  for any  $n > 7$ , the BIC statistic generally places a heavier penalty on models with many variables, and hence results in the selection of smaller models than AIC .

### Example 8.2.6

The right-hand panel of Figure 8.2 displays the BIC for the `Credit` data set. For instance, the BIC values result from subtracting the BIC of the null model.

Based on the BIC values displayed in the `Python`-output, we conclude that the best model produced by forward stepwise selection contains five predictor variables.

The `Python` output in example 8.2.1 shows that the best model produced by forward stepwise selection and containing five predictors is given by

$$\begin{aligned} \text{balance} = & \beta_0 + \beta_1 \cdot \text{income} + \beta_2 \cdot \text{limit} + \beta_3 \cdot \text{rating} + \beta_4 \cdot \text{cards} \\ & + \beta_5 \cdot \text{student} + \varepsilon \end{aligned}$$

If we now want to select the best model on the basis of the BIC criterion, then we need to specify the function arguments of `add_one()` `scoreby` to BIC.

(to R)

```
[1]: import pandas as pd
import numpy as np
from LMS_def import *

# Load data
df = pd.read_csv(
```

## Chapter 8. Linear Model Selection

```
' ../../Themen/Lineare_Modell_Auswahl/Daten/Credit.csv')

# Convert Categorical variables
df = pd.get_dummies(data=df, drop_first=True,
                     prefix=('Gender__', 'Student__',
                             'Married__', 'Ethnicity__'))

x_full = df.drop(columns='Balance')
y = df['Balance']

results = pd.DataFrame(data={'Best_Pred': [], 'BIC': []})

# Define the empty predictor
x0 = [np.zeros(len(y))]

x = x0
x_red = x_full.copy()

for i in range(x_full.shape[1]):
    results_i, best_i = add_one(x_red, x, y, scoreby='BIC')

    # Update the empty predictor with the best predictor
    x = np.concatenate((x, [df[best_i]]))

    # Remove the chosen predictor from the list of options
    x_red = x_red.drop(columns=best_i)

    # Save results
    results.loc[i, 'Best_Pred'] = best_i
    results.loc[i, 'BIC'] = results_i['BIC'].min()

print('Best Predictors and corresponding BIC:\n', results,
      '\n\nThe best model thus contains',
      results['BIC'].argmin() + 1, ' predictors')
```

Best Predictors and corresponding BIC:

	Best_Pred	BIC
0	Rating	5502.764477
1	Income	5224.531479
2	Student__Yes	4865.352851
3	Limit	4852.481331
4	Cards	4841.615607
5	Age	4842.979215
6	Gender__Male	4847.832276
7	Unnamed: 0	4852.927218
8	Ethnicity__Asian	4858.201340
9	Married__Yes	4863.468707
10	Ethnicity__Caucasian	4868.833498
11	Education	4874.337112

The best model thus contains 5 predictors

## Chapter 8. Linear Model Selection

If we want to identify the best model by *backward stepwise selection*, then we can use the same procedure as in 8.2.2:

(to R)

```
[2]: results = pd.DataFrame(data={'Worst_Pred': [], 'BIC': []})  
  
# Define the full predictor  
x = x_full.copy()  
  
for i in range(x_full.shape[1]):  
    results_i, worst_i = drop_one(x, y, scoreby='BIC')  
  
    # Update the empty predictor with the best predictor  
    x = x.drop(columns=worst_i)  
  
    # Save results  
    results.loc[i, 'Worst_Pred'] = worst_i  
    results.loc[i, 'BIC'] = results_i['BIC'].min()  
  
print('Worst Predictors and corresponding BIC:\n', results,  
      '\n\nThe best model thus contains',  
      x_full.shape[1] - results['BIC'].argmin(), 'predictors')
```

Worst Predictors and corresponding BIC:

	Worst_Pred	BIC
0	Education	4868.833498
1	Ethnicity_Caucasian	4863.468707
2	Married_Yes	4858.201340
3	Ethnicity_Asian	4852.927218
4	Unnamed: 0	4847.832276
5	Gender_Male	4842.979215
6	Age	4841.615607
7	Rating	4840.658660
8	Cards	4873.759072
9	Student_Yes	5237.176925
10	Income	5507.965562
11	Limit	6044.702777

The best model thus contains 5 predictors

We again end up with the same five predictor variables.

## Conceptual Objectives

You should be able...

- to explain the forward stepwise selection algorithm.
- to explain the backward stepwise selection algorithm.
- to explain the best subset selection algorithm.
- to explain the concept of hybrid approaches between the stepwise forward and backward selection procedures.
- to explain the concept of the adjusted  $R^2$ ,  $C_p$ , AIC and BIC.
- to choose the optimal model on the basis of the adjusted  $R^2$ ,  $C_p$ , AIC or BIC statistics.

## Computational Objectives

You should be able...

- to find the AIC, BIC, R2 and RSS from a fitted linear model, using `statsmodels.api`.
- to apply the function `sklearn.feature_selection()` from `sklearn.feature_selection` to carry out automatically the forward stepwise variable selection.
- to manipulate the function `sklearn.feature_selection()` to use backward feature selection, and to set the number of features to select.
- to choose the most appropriate number of predictors on the basis of the AIC.

## **Part II.**

# **Classification**

# Chapter 9.

## Introduction to Classification Problems<sup>1</sup>

A *predictive model* is a functional relation between a dependent *response variable*  $Y$  and input variables  $X_1, \dots, X_p$  (also called *predictor variables*):

$$Y = f(X_1, \dots, X_p). \quad (9.1)$$

The function  $f$  is usually unknown and has to be *estimated* from data. The estimate  $\hat{f}$  can then be used to *predict* responses for newly observed inputs:

$$\hat{Y} = \hat{f}(X_1, \dots, X_p)$$

*Multiple linear regression* is a methodology of prediction by means of a linear function  $f$  that is estimated via least squares optimization. Its use, however, is limited to *quantitative* response variables  $Y$ , i.e., where  $Y$  is a numeric scalar. In general, predictive models for quantitative variables are termed *regression*. In this chapter we will deal with *qualitative* (or *categorical*) response variables, i.e.,  $Y$  takes on values in a finite number of *classes* or *categories*.

### Remarks:

- i. The functional relation stated in equation 9.1 is relying on the assumption that such a relation exists in reality. Often, we will rather predict a probability for an observation of belonging to a class. ◆

### Example 9.0.1

Here is a list of typical categorical variables with their classes:

1. A person's gender: **male** or **female**

---

<sup>1</sup>This chapter follows Chapters 4.1 – 4.2 of the text book *An Introduction to Statistical Learning: with Applications in R* by G. James et al., Springer Texts in Statistics 2013.

2. A brand name: brands **A**, **B**, or **C**
3. Tumor class: **EWS**, **RMS**, **NB**, or **BL**
4. A person's eye color: **green**, **blue**, or **brown**
5. Whether a student passes an exam: **yes** or **no**

If a variable takes only two levels (as in items 1. and 5.), it is called *binary*. 

In this chapter we deal with the prediction of a qualitative variable, which is referred to as *classification*. In other words, a classification model predicts a class level from observed values for inputs  $X_1, \dots, X_p$ .

In practice, many classification methods predict the *probabilities* of an observation belonging to the individual classes. Based on these probabilities, the classification is carried out. Viewed from this perspective, classification can be seen as a regression model for probabilities (which are quantitative by nature).

There are at least as many different classification methods as there are regression approaches and the zoo of available methods is ever growing. There is, however, not *THE best* classification method since computation time and performance (as measured e.g. by classification accuracy) depend heavily on the problem under consideration (*no-free-lunch theorem*). Put differently, for two different classification problems, a specific method may perform very well in solving the first one but poorly in the second.

In particular, we first discuss *logistic regression* being the pioneer classification method established by David Cox in the late 1950's. It can be seen as an analogue of linear regression for classification and invokes modeling of the data under consideration. Much in the same spirit is linear discriminant analysis (LDA).

As a second method we will consider *classification and regression trees* (CART). In contrast to logistic regression, which has its origins in the mathematical statistics community, tree-based methods are rather algorithmic and have their roots in computer science. Here, there are no a-priori assumptions on the distribution of the data.

Finally, we will discuss *random forests* representing an ensemble method. Here, a classifier is derived from an ensemble of decision trees by averaging. So the variance of the trees is reduced whilst maintaining predictive accuracy.

Classification problems frequently occur in many different areas. Here are some examples:

### Example 9.0.2

1. An e-mail client (such as MS Outlook or Mozilla Thunderbird) receives an e-mail. Is it a **spam** or proper (**ham**) mail?

2. Based on process data such as temperatures, pressures etc., a manufacturer wants to predict the state (**okay** vs. **defect**) of an engine in the near future (*predictive maintenance*).
3. A doctor has to attribute the symptoms of a patient to three possible medical conditions, e.g. **stroke**, **drug overdose**, and **epileptic seizure**.



We now give a more precise definition of the classification task.

### Classification

Given is a fixed but unknown function  $f$  of  $p$  variables with quantitative values.

1. Assume that we are given  $n$  observations  $(\vec{x}_1, y_1), (\vec{x}_2, y_2), \dots, (\vec{x}_n, y_n)$  where  $\vec{x}_i = (x_{i1}, \dots, x_{ip})$  denotes a vector of values of the predictor variables  $X_1, \dots, X_p$ , and  $y_i$  is the corresponding value of the response variable  $Y$ , i.e.,  $y_i = f(\vec{x}_i)$ . This data is stored in a data table and referred to as the *training set*

$$\begin{array}{cccc|c} x_{11} & x_{12} & \dots & x_{1p} & y_1 \\ x_{21} & x_{22} & \dots & x_{2p} & y_2 \\ \vdots & \vdots & & \vdots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{np} & y_n \end{array}$$

2. Classification amounts to derive from the training set a function  $\hat{f}$  of  $p$  variables such that  $\hat{y} = \hat{f}(\vec{x})$  predicts the qualitative response  $y = f(\vec{x})$  from new data  $\vec{x} = (x_1, \dots, x_p)$ . Such a function  $\hat{f}$  is called a *model*.

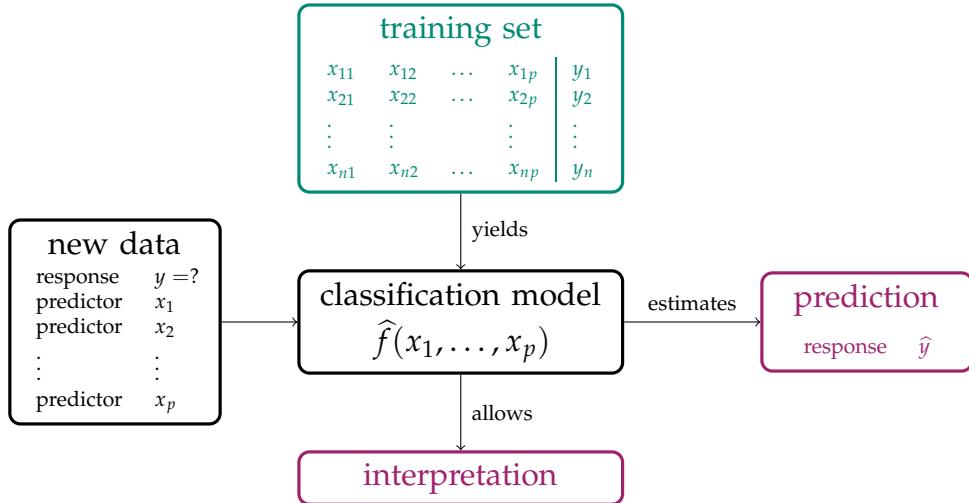
This procedure is schematically depicted in Figure 9.1.

In addition to prediction, a *predictive model* can also explain the dependency of the response on the predictors which leads to a deeper understanding of the data and the underlying process. This understanding is often called *inference* or *interpretation*. Interpretability and predictive accuracy of a model are opposed concepts: a well interpretable model (such as logistic regression) has often less predictive power as compared to a less interpretable but more sophisticated model (such as random forests).

### Example 9.0.3

We want to predict whether a person will default on his or her debt (i.e., is not able to pay his or her due), based on the annual income and the monthly credit card bill. The data set consists of the following variables:

## Chapter 9. Introduction to Classification Problems



**Figure 9.1.:** Schematic diagram of training a classification method.

- **default:** Binary response variable (**Yes** or **No**), whether or not the person defaults.
- **income:** (first numeric predictor) annual income of the person.
- **balance:** (second numeric predictor) monthly credit card balance.

The following `Python`-code chunk produces a scatter plot of **income** and **balance**.  
(to R)

```
[1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Load data
df = pd.read_csv(
    '../../../../../Themen/Logistische_Regression/Data/Default.csv',
    sep=';')

# Split the data based on default
df_no = df.loc[df['default'] == 'No', :]
df_yes = df.loc[df['default'] == 'Yes', :]

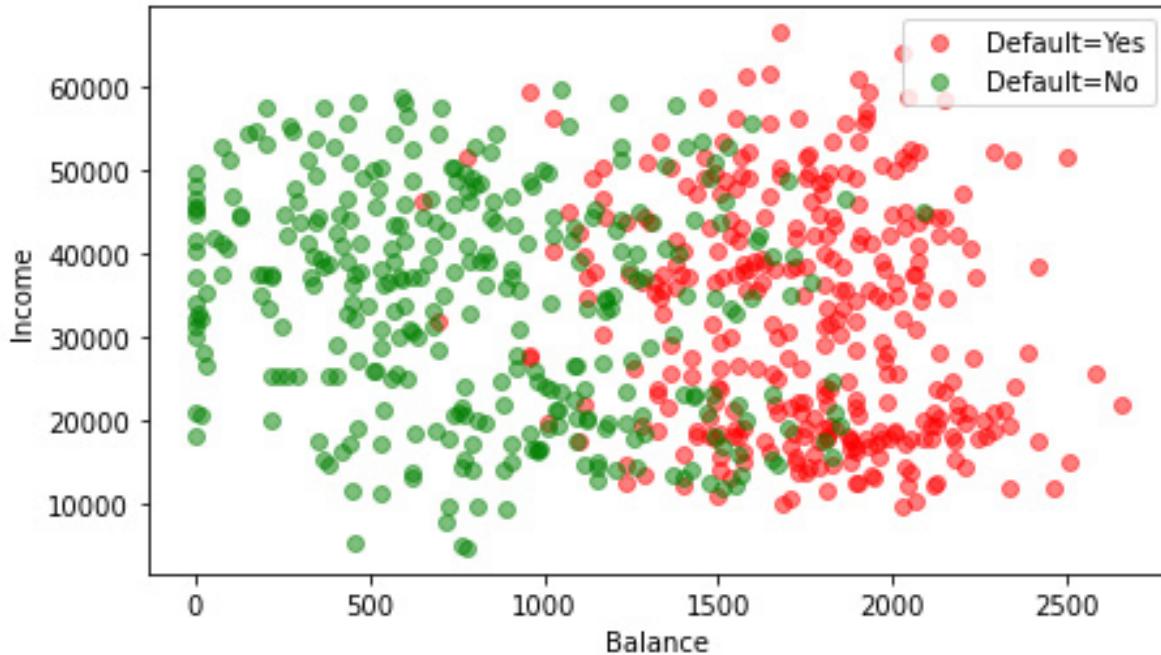
# Random set of No:
i_no = np.random.choice(df_no.index, replace=False, size=333)

# Plot all Yes and a selection of No
# Create Figure and subplots
fig = plt.figure(figsize=(7, 4))
ax = fig.add_subplot(111)
plt.plot(df_yes['balance'], df_yes['income'],
         'or', alpha=0.5, label='Default=Yes')
```

```

plt.plot(df_no['balance'][i_no], df_no['income'][i_no],
         'og', alpha=0.5, label='Default=No')
ax.set_xlabel('Balance')
ax.set_ylabel('Income')
plt.legend()

plt.show()
    
```



**Figure 9.2.:** Scatter plot of `income` versus `balance` from the `default` data set. Red corresponds to `default=Yes`, and green to `default>No`.

In Figure 9.2 the resulting scatter plot of `income` versus `balance` is shown. The coloring of the vertices is according to the variable `default`, i.e., red dots represent individuals who defaulted on their debts (`default=Yes`) and green dots represent the other ones (`default>No`). Note that there are actually far more observations with `default>No` (97% of 10 000 in total). For the sake of lucidity, only a fraction of the class `default>No` is plotted. Visually, it seems obvious that people with a higher credit card balance tend to default on their debts more often than those who exhibit smaller balance. Moreover, the income does not seem to have a comparable high influence.

In addition, Figure 9.3 shows two pairs of box plots (computed from the complete data set) of the variable `balance` (left) and `income` (right). Again, the strong influence of `balance` becomes remarkably visible.

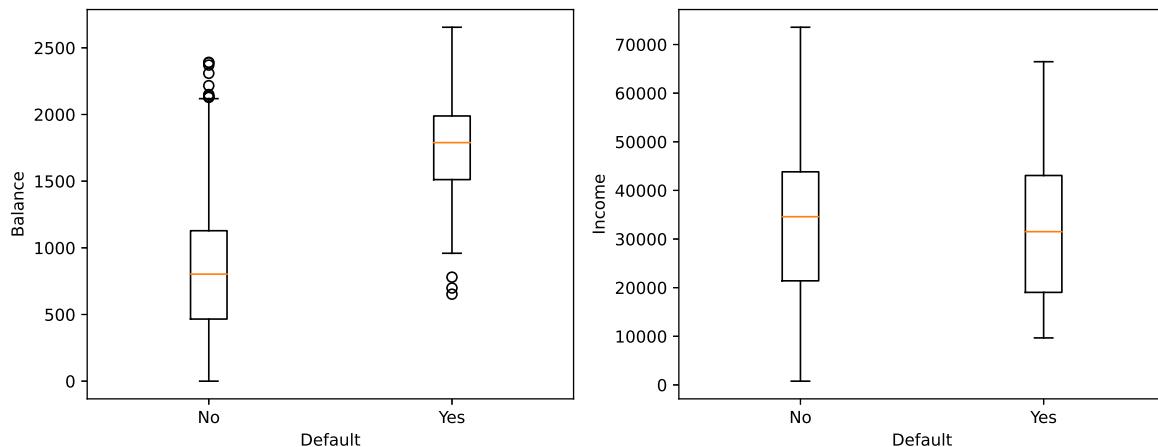
([to R](#))

## Chapter 9. Introduction to Classification Problems

```
[2]: # Create Figure and subplots
fig = plt.figure(figsize=(10, 4))
ax1 = fig.add_subplot(1, 2, 1)
ax1.boxplot([df_no['balance'], df_yes['balance']])
ax1.set_xlabel('Default')
ax1.set_ylabel('Balance')
ax1.set_xticklabels(['No', 'Yes'])

ax2 = fig.add_subplot(1, 2, 2)
ax2.boxplot([df_no['income'], df_yes['income']])
ax2.set_xlabel('Default')
ax2.set_ylabel('Income')
ax2.set_xticklabels(['No', 'Yes'])

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



**Figure 9.3.:** Boxplots for the sample data set `Default`: `balance` (left) and `income` (right) for the classes `default=Yes` and `default>No`.

In this example, the task of classification consists of building a model from the available data in order to achieve at least one of the following two tasks:

1. *Infer* the way the variables `balance` ( $X_1$ ) and `income` ( $X_2$ ) affect the response variable `Default` ( $Y$ ).
2. *Predict* the response variable `Default` from given values for `balance` and `income`.



# Chapter 10.

## Logistic Regression<sup>1</sup>

We motivate the logistic regression model by means of the following example.

### Example 10.0.1

We consider again the `Default` data set, where the levels of `default` (response variable  $Y$ ) are given by the categories `Yes` and `No`. We are interested in predicting whether an individual will default on his or her credit card payment, on the basis of annual `income` and monthly credit card `balance`. Since the income of a person does not seem to have any significant influence on the default probability, we ignore for the time being the predictor variable `income`. In other words, we aim at modeling the *probability* that `default` equals `Yes` depending on the value of `balance` (numeric predictor  $X$ ). In other words, we are looking for a model that predicts the *conditional probability*

$$P(\text{default}=\text{Yes} \mid \text{balance})$$

which we abbreviate  $p(\text{balance})$ . Obviously, the values of  $p(\text{balance})$  are between 0 and 1. For any given new observation of `balance` our model then predicts a probability for the response `default` being `Yes`.

Predicting the proper class then is carried out by thresholding, say, at 0.5: i.e., if for an individual we find  $p(\text{balance}) > 0.5$ , then the prediction would be `default=Yes`. The choice of the threshold (here 0.5) is crucial and by no means trivial. For example, consider a situation where you want to avoid false accusations (*false positives*), say in court. Then you should use a higher threshold. We will remark on a more systematic way later in the chapter considering rates of false positives and false negatives. ◀

---

<sup>1</sup>This chapter follows Chapter 4.3 of the text book *An Introduction to Statistical Learning: with Applications in R* by G. James et al., Springer Texts in Statistics 2013.

## 10.1. The Logistic Model

As motivated by the previous example 10.0.1, we aim at modeling the conditional probability

$$p(X) = P(Y = 1 | X)$$

We consider only *binary* response variables  $Y$ , where we use the numeric encoding 1 and 0 (so  $Y = 1$  would correspond to `default = Yes` in the above example). To be more precise, we assume that the random variable  $Y$  given the response  $X$  is a *Bernoulli* random variable with success probability  $p(X)$ .

A naïve approach to model this probability would be a linear model for the probabilities:

$$p(X) = \beta_0 + \beta_1 X \quad (10.1)$$

### Example 10.1.1

The approach in (10.1) results in a model for  $P(\text{default}=\text{Yes} | \text{balance})$  as depicted in Figure 10.1 (blue line). The probability

$$P(\text{default}=\text{Yes} | \text{balance}) = P(Y = 1 | X)$$

is plotted against the monthly credit card bill (`balance`)  $X$ . The probabilities of the observed values are given by the 0/1-Coding of the `default` variable: if `default=Yes` we have  $p(X) = 1$  (red dots) and in case `default=No` we have  $p(X) = 0$  (green dots).

(to R)

```
[1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm

# Load data
df = pd.read_csv(
    '../../../../../Themen/Logistische_Regression/Data/Default.csv',
    sep=';')

# Add a numerical column for default
df = df.join(pd.get_dummies(df['default'],
                           prefix='default',
                           drop_first=True))

# Index of Yes:
i_yes = df.loc[df['default_Yes'] == 1, :].index
```

## Chapter 10. Logistic Regression

```
# Random set of No:  
i_no = df.loc[df['default_Yes'] == 0, :].index  
i_no = np.random.choice(i_no, replace=False, size=333)  
  
# Fit Linear Model, only including the selection of no  
i_ = np.concatenate((i_no, i_yes))  
x = df.iloc[i_]['balance']  
y = df.iloc[i_]['default_Yes']  
  
x_sm = sm.add_constant(x)  
model = sm.OLS(y, x_sm).fit()  
  
# Find the predicted values  
x_pred = x.sort_values()  
y_pred = model.predict(sm.add_constant(x_pred))
```

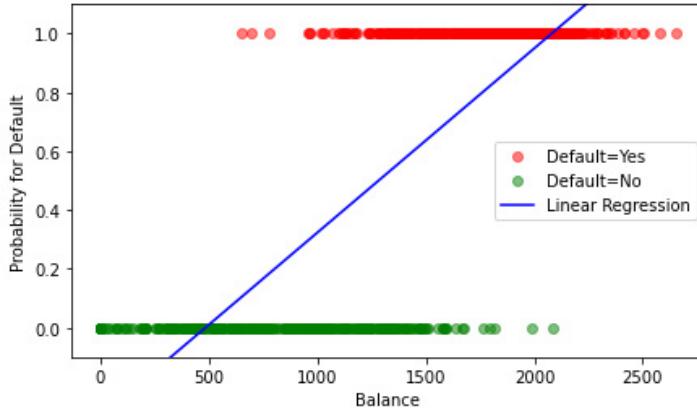
```
[2]: """ Plot """  
# Create Figure and subplots  
fig = plt.figure(figsize=(7, 4))  
ax = fig.add_subplot(111)  
# Plot datapoints  
plt.plot(df.iloc[i_yes]['balance'], df.iloc[i_yes]['default_Yes'],  
         'or', alpha=0.5, label='Default=Yes')  
plt.plot(df.iloc[i_no]['balance'], df.iloc[i_no]['default_Yes'],  
         'og', alpha=0.5, label='Default=No')  
# Plot fit  
plt.plot(x_pred, y_pred, 'b-', label='Linear Regression')  
# Labels and limits  
ax.set_xlabel('Balance')  
ax.set_ylabel('Probability for Default')  
ax.set_ylim(-0.1, 1.1)  
plt.legend()  
  
plt.show()
```

Note that the linear function of the model returns negative values for low **balance** and values greater than 1 for high **balance**. These values are not interpretable as probabilities, since those are bounded between 0 and 1.

Also note that only a part of the **data** was included. The full data contains many more **default = No** entries, which would result in a much lower probability for default. ◀

Example 10.1.1 shows that the class of linear functions is not appropriate to model probabilities. This is because a non-trivial linear function (i.e. nonconstant affine function) is always unbounded, so it takes on values outside the unit interval [0, 1].

Rather than using a completely different class of regression functions, the idea underlying *logistic regression* is to modify a linear function by composing it with another



**Figure 10.1.:** A linear regression model  $p(X) = \beta_0 + \beta_1 X$ , where  $X$  is the predictor variable `balance`. The model was fitted to the `Default` data set and is plotted as a straight line. The dots represent the measured values, where red corresponds to `default = Yes` and green to `default = No`.

function which *shrinks* all of  $\mathbb{R}$  to  $[0, 1]$ . There are many such functions. In logistic regression, we choose the *logistic* function

$$p(t) = \frac{e^t}{1 + e^t} \quad \text{with } t \in \mathbb{R} \quad (10.2)$$

The graph of the logistic function is depicted in Figure 10.2. (to R)

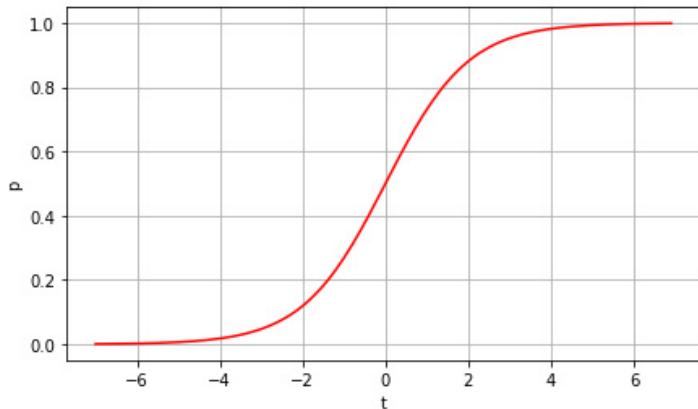
```
[1]: # Define t and p
t = np.arange(-7, 7, 0.1)
p = []
for i in range(len(t)):
    p.append(np.exp(t[i]) / (1 + np.exp(t[i])))

# Plot graph
# Create Figure and subplots
fig = plt.figure(figsize=(7, 4))
ax = fig.add_subplot(111)
plt.plot(t, p, 'r-')
plt.grid(True)
plt.xlabel('t')
plt.ylabel('p')
plt.show()
```

With these preparations we now formalize the simple logistic regression model (*simple* referring to the fact, that only one predictor  $X$  is used).

### Simple logistic regression

Given a binary response variable  $Y$  and a quantitative predictor  $X$ , the *simple*



**Figure 10.2.:** The graph of the logistic function. The real line is transformed to the interval  $[0, 1]$ .

*logistic regression model* is defined as

$$P(Y = 1|X) = \frac{e^{\beta_0 + \beta_1 X}}{1 + e^{\beta_0 + \beta_1 X}} \quad (10.3)$$

The parameters  $\beta_0$  and  $\beta_1$  are called regression coefficients and are estimated from the training set.

In order to estimate  $\beta_0$  and  $\beta_1$ , a least squares approach may be applied. Instead the more general *maximum likelihood method* is preferred since it has better statistical properties. We discuss the parameter estimation by means of maximum likelihood in the next section. First, we have a look at the model for the **Default** data set. For the computation of the parameters we use the **Python** function `statsmodels.api.GLM()`

### Example 10.1.2

In Figure 10.3 a simple logistic regression model (10.3) for the **Default** data is plotted. (to R)

```
[1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm

# Load data
df = pd.read_csv(
    '../../../../../Themen/Logistische_Regression/Data/Default.csv',
    sep=';')

# Add a numerical column for default
df = df.join(pd.get_dummies(df['default']),
```

## Chapter 10. Logistic Regression

```
prefix='default',
drop_first=True))

# Index of Yes:
i_yes = df.loc[df['default_Yes'] == 1, :].index

# Random set of No:
i_no = df.loc[df['default_Yes'] == 0, :].index
i_no = np.random.choice(i_no, replace=False, size=333)

# Fit Linear Model, only including the selection of no
i_ = np.concatenate((i_no, i_yes))
x = df.iloc[i_]['balance']
y = df.iloc[i_]['default_Yes']

x_sm = sm.add_constant(x)

model = sm.GLM(y, x_sm, family=sm.families.Binomial())
model = model.fit()

# Find the predicted values
x_pred = x.sort_values()
y_pred = model.predict(sm.add_constant(x_pred))
```

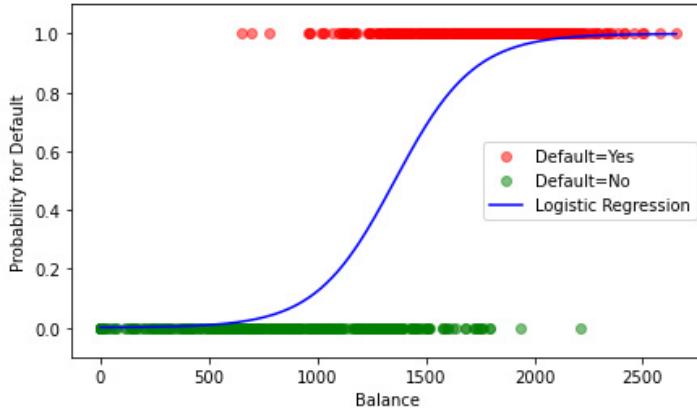
[2]:

```
""" Plot """
# Create Figure and subplots
fig = plt.figure(figsize=(7, 4))
ax = fig.add_subplot(111)
# Plot datapoints
plt.plot(df.iloc[i_yes]['balance'], df.iloc[i_yes]['default_Yes'],
         'or', alpha=0.5, label='Default=Yes')
plt.plot(df.iloc[i_no]['balance'], df.iloc[i_no]['default_Yes'],
         'og', alpha=0.5, label='Default>No')
# Plot fit
plt.plot(x_pred, y_pred, 'b-', label='Logistic Regression')
# Labels and limits
ax.set_xlabel('Balance')
ax.set_ylabel('Probability for Default')
ax.set_ylim(-0.1, 1.1)
plt.legend()
plt.show()
```

### Remarks:

- i. The syntax of `sm.GLM()` is similar to that of `sm.OLS()`, except that we must pass in the argument `family=sm.families.Binomial()` in order to tell `Python` to run a logistic regression rather than some other type of generalized model. ◆

For small values of `balance` the predicted values for  $p(X)$  tend to 0. Likewise, the predictions tend to 1 for large `balance` values. All values of the model are within



**Figure 10.3.:** A logistic regression function (blue) is fitted to the data set `Default`. Note that only a random sample of the `default = No` cases is used.

the interval  $[0, 1]$  and thus interpretable as probabilities. ◀

The logistic function always results in an S-shaped curve with values between 0 and 1. Thus, for each predictor value  $X = x$  we obtain a reasonable prediction for  $p(x)$ . Using some basic rearrangements, we find from (10.3)

$$\frac{p(X)}{1 - p(X)} = e^{\beta_0 + \beta_1 X} \quad (10.4)$$

The quantity  $p(X)/(1 - p(X))$  is called *odds* and may take any positive value. It is an equivalent way of expressing the probability of an event and is used for instance in betting agencies.

### Example 10.1.3

Odds values close to 0 and  $\infty$  indicate small and large probabilities, respectively. If one out of five individuals default on their debts, then  $p(X) = 0.2$ . For the odds we find

$$\frac{0.2}{1 - 0.2} = \frac{0.2}{0.8} = \frac{1}{4}$$

If, however, 9 out of 10 persons default, so  $p(X) = 0.9$ , and the odds are

$$\frac{0.9}{1 - 0.9} = \frac{0.9}{0.1} = 9$$

In a betting office, the odds of  $4 : 1$  for soccer team  $A$  winning against soccer team  $B$  means that

$$\frac{p}{1 - p} = \frac{4}{1} \Leftrightarrow p = \frac{4}{5}$$

In other words, we expect that team  $A$  wins 4 out of 5 matches against  $B$ . ◀

Taking the natural logarithm on both sides of (10.4) we obtain

$$\log \left( \frac{p(X)}{1 - p(X)} \right) = \beta_0 + \beta_1 X$$

The left side of this equation is called the *log-odds* or *logit*. Thus, logistic regression amounts to model the logit with a linear function. This yields an interpretation of the coefficients  $\beta_0$  and  $\beta_1$  in terms of the logit: a change of the predictor  $X$  by one unit amounts to an average change by  $\beta_1$  of the logit of the response being true. In contrast, the relation between  $p(X)$  and  $X$  is nonlinear, so a lucid interpretation of  $\beta_1$  is not at hand, since the rate of change of  $p(X)$  depends on  $X$ . At least the *sign* of  $\beta_1$  tells us, whether  $p(X)$  increases ( $\beta_1 > 0$ ) or decreases ( $\beta_1 < 0$ ) with  $X$ .

## 10.2. Estimation of Regression Coefficients

The coefficients  $\beta_0$  and  $\beta_1$  in the logistic model

$$p(X) = \frac{e^{\beta_0 + \beta_1 X}}{1 + e^{\beta_0 + \beta_1 X}}$$

are unknown and have to be estimated from the training data. We will use the *maximum likelihood method*. We motivate the idea underlying this method with an example.

### Example 10.2.1

We start with a small sample of three cases in the **Default** data (cf. Table 10.1).

	<b>default</b>	<b>student</b>	<b>balance</b>	<b>income</b>
240	No	No	979	57839
241	No	No	371	15703
242	Yes	Yes	1573	14930

**Table 10.1.:** Three cases extracted from the dataset **Default**.

Assume for the moment that we know the dependency of probability  $p(\text{balance}) = P(\text{default=yes}|\text{balance})$  on the value of **balance**. We denote by  $X_i$  the balance of individual number  $i$  and by  $Y_i$  the binary variable indicating whether he/she defaults. Then, assuming that the events that each individual defaults are independent, the probability for observing the default values given the incomes from Table 10.1 is the product of the individual probabilities

$$\begin{aligned} P(Y_{240} = 0, Y_{241} = 0, Y_{242} = 1) &= P(Y_{240} = 0|X_{240}) \cdot P(Y_{241} = 0|X_{241}) \cdot P(Y_{242} = 1|X_{242}) \\ &= (1 - p(X_{240})) \cdot (1 - p(X_{241})) \cdot p(X_{242}). \end{aligned} \quad (10.5)$$

If the model correctly describes the data, then this probability should be relatively large. Note that the assumption of a Bernoulli distribution for  $Y_i$  given  $X_i$  ( $Y_i$  is a binary variable, that is  $Y$  takes on two possible values) is crucial for the computation of this probability.

Now we return to the situation where the data is given and the model is unknown and use the above probability in order to evaluate a candidate model for  $p(X)$ . In this context, the probability (10.5) is called *likelihood* and measures how well a model describes the given data. We now look for a model that maximizes the likelihood, hence the name *maximum likelihood method*.

In the context of logistic regression, we know the model up to the two parameters  $\beta_0$  and  $\beta_1$ . We use the maximum likelihood method to estimate these two parameters. In order to do this, we substitute the logistic model (10.3) into (10.5) and obtain the *likelihood function*

$$\begin{aligned} L(\beta_0, \beta_1) &= (1 - p(X_{240})) \cdot (1 - p(X_{241})) \cdot p(X_{242}) \\ &= \left(1 - \frac{e^{\beta_0 + \beta_1 \cdot 979}}{1 + e^{\beta_0 + \beta_1 \cdot 979}}\right) \cdot \left(1 - \frac{e^{\beta_0 + \beta_1 \cdot 371}}{1 + e^{\beta_0 + \beta_1 \cdot 371}}\right) \cdot \frac{e^{\beta_0 + \beta_1 \cdot 1573}}{1 + e^{\beta_0 + \beta_1 \cdot 1573}} \end{aligned}$$

We now maximize  $L(\beta_0, \beta_1)$  which amounts to set the partial derivatives of  $L$  with respect to  $\beta_0$  and  $\beta_1$  to zero. Solving the resulting two equations for  $\beta_0$  and  $\beta_1$  yields the desired estimates of the parameters. ◀

In principle, the maximization of the likelihood function  $L(\beta_0, \beta_1)$  could be done analytically by setting its partial derivatives to zero and solving the corresponding system of equations. However, in practice this is not feasible, since the likelihood function is a product of as many terms as there are cases. Numerical optimization techniques can compute approximate solutions stably and in reasonable time. Such an algorithm is implemented in the [Python](#)-function `statsmodels.api.GLM()` ([generalized linear model](#)) for different kinds of models including logistic regression.

### Example 10.2.2

The [Python](#)-output below shows the estimates for the parameters in the logistic regression model. [\(to R\)](#)

```
[1]: import numpy as np
import pandas as pd
import statsmodels.api as sm

# Load data
df = pd.read_csv(
    '../../Themen/Logistische_Regression/Data/Default.csv',
    sep=';')

# Add a numerical column for default
```

## Chapter 10. Logistic Regression

```

df = df.join(pd.get_dummies(df['default'],
                            prefix='default',
                            drop_first=True))

# Fit logistic model
x = df['balance']
y = df['default_Yes']

x_sm = sm.add_constant(x)

model = sm.GLM(y, x_sm, family=sm.families.Binomial())
model = model.fit()

# Print summary
print(model.summary())

```

```

Generalized Linear Model Regression Results
=====
Dep. Variable:      default_Yes    No. Observations:             10000
Model:                 GLM    Df Residuals:                  9998
Model Family:        Binomial   Df Model:                      1
Link Function:       logit    Scale:                     1.0000
Method:                IRLS    Log-Likelihood:            -798.23
Date:          Wed, 24 Mar 2021   Deviance:                  1596.5
Time:           11:06:11     Pearson chi2:            7.15e+03
No. Iterations:                   9
Covariance Type:        nonrobust
=====
            coef      std err      z      P>|z|      [0.025      0.975]
-----
const    -10.6513      0.361   -29.491      0.000    -11.359     -9.943
balance    0.0055      0.000    24.952      0.000      0.005      0.006
=====
```

From the [Python](#)-output we find the estimates

$$\hat{\beta}_0 = -10.6513 \quad \text{and} \quad \hat{\beta}_1 = 0.0055$$

and thus the estimated logistic regression model for the data set **Default** is

$$\hat{p}(X) = \frac{e^{\hat{\beta}_0 + \hat{\beta}_1 X}}{1 + e^{\hat{\beta}_0 + \hat{\beta}_1 X}} = \frac{e^{-10.6513 + 0.0055 X}}{1 + e^{-10.6513 + 0.0055 X}}$$

The coefficient  $\beta_1$  is positive and thus  $p(X)$  increases with  $X$ , i.e., the larger the monthly credit card bill, the more likely the debt is not payed.

Besides the coefficients, [Python](#) outputs standard errors for the estimates. The  $z$ -statistic plays an analogous role to the  $t$ -statistic in linear regression. For instance, the  $z$ -statistic indicates evidence against the null hypothesis

$$H_0: \beta_1 = 0$$

In other words, we test the null hypothesis that the probability for `default=yes` does not depend on `balance`. Assuming the null hypothesis, the probability to observe the data set `Default` is given by  $\Pr(>|z|)$  which is negligible in our case, i.e., we reject  $H_0$  in favor of the alternative hypothesis

$$H_1: \beta_1 \neq 0$$

and conclude that `balance` has a significant influence.

Equivalently we can examine the confidence intervals for the coefficients

(to R)

```
[2]: # Find confidence interval
print(model.conf_int(alpha=0.05))
```

	0	1
const	-11.359208	-9.943453
balance	0.005067	0.005931

The two-sided 95%-interval for  $\beta_1$  is well separated from 0 which is equivalent to rejecting  $H_0$  with a type I error of  $\alpha = 5\%$ . ◀

### 10.3. Predictions and Model Evaluation

Once the model parameters are determined from the training set, it is straightforward to calculate the probability that the response  $Y$  takes on the value 1 given a certain predictor value  $X$ . We illustrate this with the `Default` data.

#### Example 10.3.1

The estimated coefficients are

$$\hat{\beta}_0 = -10.6513 \quad \text{and} \quad \hat{\beta}_1 = 0.0055$$

Thus, if an individual has `balance = 1000`, then the model yields

$$\hat{p}(1000) = \frac{e^{-10.65+0.0055 \cdot 1000}}{1 + e^{-10.65+0.0055 \cdot 1000}} \approx 0.00577$$

(to R)

```
[1]: import numpy as np
import pandas as pd
import statsmodels.api as sm
```

## Chapter 10. Logistic Regression

```
# Load data
df = pd.read_csv(
    '.../Themen/Logistische_Regression/Data/Default.csv',
    sep=';')

# Add a numerical column for default
df = df.join(pd.get_dummies(df['default'],
                            prefix='default',
                            drop_first=True))

# Fit logistic model
x = df['balance']
y = df['default_Yes']

x_sm = sm.add_constant(x)

model = sm.GLM(y, x_sm, family=sm.families.Binomial())
model = model.fit()

# Predict for balance = 1000
x_pred = [1, 1000]
y_pred = model.predict(x_pred)

print(y_pred)
```

```
[0.00575215]
```

This probability of default is well below 1%, which is very low. However, a different individual with **balance = 2000** has a default probability of approximately 59%.  
[\(to R\)](#)

```
[2]: # Predict for balance = 2000
x_pred = [1, 2000]
y_pred = model.predict(x_pred)

print(y_pred)
```

```
[0.58576937]
```

The two cases in the previous example highlight the typical challenge with probability based classification methods. For  $X = 1000$  and the resulting low probability of default, a predicted class **default = No** seems evident. In the second case, however, a default probability between 50 % and 60 % is less definite. A sharp assignment to the class **default = Yes** seems to be supported much less clearly by the data.

In general, the following simple prediction rule is used. For a given value  $x$  of  $X$ , the corresponding response value  $\hat{y}$  of  $Y$  is predicted as

$$\hat{y} = \hat{f}(x) := \begin{cases} 1 & \text{if } \hat{p}(x) > 0.5 \\ 0 & \text{otherwise.} \end{cases} \quad (10.6)$$

At this point, a natural question arises: How well does this classification scheme predict the binary response variable  $Y$ ?

A first approach for model assessment is the

### Classification error

Let  $(x_1, y_1), \dots, (x_n, y_n)$  be observations of  $(X, Y)$  and  $\hat{y}_i = \hat{f}(x_i)$  the corresponding predictions. The *classification error* is given by

$$\text{Err} = \frac{1}{n} \sum_{i=1}^n I(y_i \neq \hat{y}_i). \quad (10.7)$$

Here,  $I$  denotes the *indicator function* which takes on value 1 if its argument is true and 0 otherwise. In other words, the classification error is the proportion of cases with a wrong prediction (either false positive or false negative).

If the data in the above definition is equal to the training data, we call Err the *training classification error*. In fact, using the training data for evaluation of predictive accuracy is statistically unsound and usually gives overoptimistic estimates. For the time being, however, we will stick to the training set and introduce a workaround in Section 10.4.

### Example 10.3.2

For the **Default** data the following Python-code computes the training classification error. (to R)

```
[3]: """ Follows Example 3.1 """
# Predict for training data
x_pred = x_sm
y_pred = model.predict(x_pred)

# Round to 0 or 1
y_pred = y_pred.round()

# Compute training error
e_train = abs(y - y_pred)
e_train = e_train.mean()

print(e_train)
```

0.0275

The value of the training error in this example is 0.0275, which is to say that approximately 97.25 % of the cases in the training set are classified correctly. ◀

The result in the example above seems to be quite satisfying, but the classification error can be misleading. We show this using the more refined concept of the *confusion matrix*. It yields a generic and informative method to examine a classification result, and is thus introduced in general form.

### Confusion matrix

Let  $y_1, \dots, y_n$  be observations of the binary response  $Y$  with values 0 and 1. Furthermore, let  $\hat{y}_1, \dots, \hat{y}_n$  be the corresponding predictions of some classification model. The *confusion matrix* is the following contingency table.

		observed value $y$		total
		1	0	
predicted value $\hat{y}$	1	True positive	False positive	P'
	0	False negative	True negative	N'
total		P	N	

Here,

1. *True positive* refers to the number of cases that are correctly classified as 1 ( $y_i = \hat{y}_i = 1$ ).
2. *False positive* is the number of cases that are classified as 1 but which are truly 0 ( $\hat{y}_i = 1$  and  $y_i = 0$ )
3. *False negative* is the number of cases that are classified as 0 but which are truly 1 ( $\hat{y}_i = 0$  and  $y_i = 1$ )
4. *True negative* is the number of cases that are classified as 0 and which are truly 0 ( $\hat{y}_i = y_i = 0$ )

### Example 10.3.3

The following `Python`-code produces the confusion matrix for the `Default` data set and the logistic regression model. ([to R](#))

```
[4]: """ Follows Example 3.2 """
# Create confusion matrix
confusion = pd.DataFrame({ 'predicted': y_pred,
                           'true': y})
confusion = pd.crosstab(confusion.predicted, confusion.true,
                        margins=True, margins_name="Sum")

print(confusion)
```

predicted	0	1	Sum
true			
0.0	9625	233	9858
1.0	42	100	142
Sum	9667	333	10000

It can be seen that out of 9667 cases with `default=No`, the vast majority of 9625 are classified correctly. On the other hand, only approximately 1/3 of the `default=Yes` cases are classified correctly. The confusion matrix shows that the present classification scheme is by no means useful, in particular, if you want to predict the case of `default=Yes`.

The reason for this bad result is the *imbalance* of the two classes. The training data only contains 333 out of 10 000 cases with `default=Yes`. Therefore, the likelihood function is dominated by the factors corresponding to `default=No`, so the parameters are chosen as to match mainly those cases. Note also that the trivial classifier predicting all observations  $x$  to  $\hat{f}(x) = 0$  has a classification error of  $333/10000 = 0.0333$  which is not much worse than that of our logistic model.

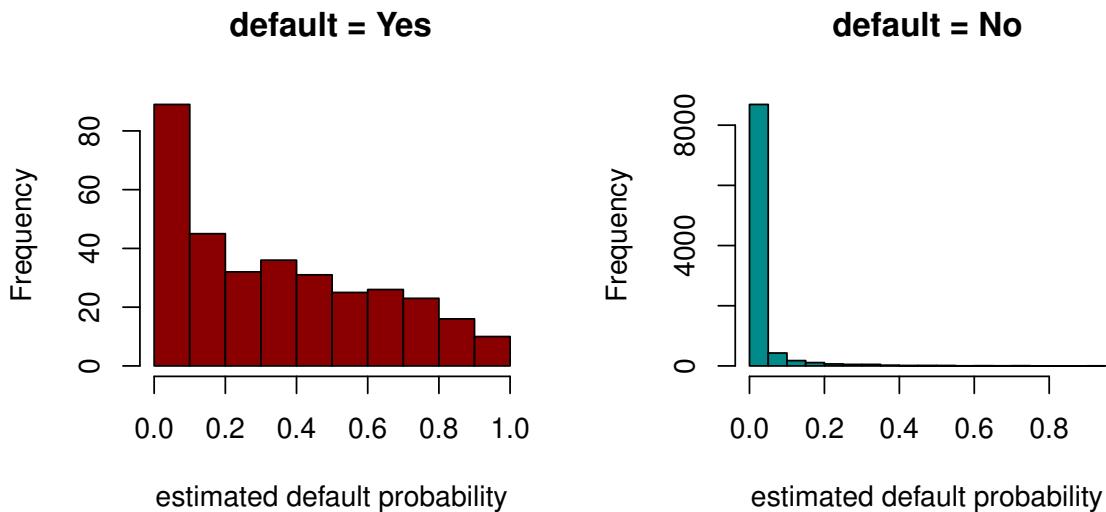
The situation can also be visualized by the histograms of the estimated probabilities of `default=Yes` separated by true class, see Figure 10.4.

It is striking that the `default=No` group has a high concentration of probabilities near 0 which is reasonable for this group. On the other hand, though, the estimated probabilities for the `default=Yes` cases do not exhibit high mass at 1. Instead, the maximal probability is attained close to 0 as well! ◀

On the basis of the confusion matrix, we can determine *accuracy*, *precision*, *recall*, and *F1 score*.

#### Accuracy

*Accuracy* is the most intuitive performance measure and it is simply a ratio of



**Figure 10.4.:** Histograms of estimated class probabilities.

correctly predicted observations to the total observations:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}}$$

One may think that, if we have high accuracy then our model is best. Accuracy is a good measure only when you have symmetric data sets where values of false positive and false negatives are almost the same. Therefore, you have to look at other parameters to evaluate the performance of your model.

#### Example 10.3.4

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}} = \frac{100 + 9625}{9625 + 233 + 42 + 100} = 0.9725$$

For our model, we have got 0.97 which means our model is approx. 97 % accurate. ◀

#### Precision

*Precision* is the ratio of correctly predicted positive observations to the total predicted positive observations.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

### Example 10.3.5

The question that this metric answers is, among all people that were predicted as `default=Yes`, how many actually defaulted?

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{100}{100 + 42} = 0.70$$

High precision relates to the low false positive rate. We have got 0.70 precision which is not any more so convincing. ◀

### Recall (Sensitivity)

*Recall* is the ratio of correctly predicted positive observations to all positive observations.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

### Example 10.3.6

The question recall answers is: of all the people that truly defaulted, how many did we predict?

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{100}{100 + 233} = 0.30$$

We have got a recall of 0.30 which is very bad for this model. ◀

#### Remarks:

- i. Recall or sensitivity is often referred to as *true positive rate*, which simply is

$$\frac{\text{TP}}{\text{TP}+\text{FN}}$$

The *false positive rate* is defined as:

$$\frac{\text{FP}}{\text{FP}+\text{TN}}$$

where  $\text{FP}+\text{TN}$  is the total number of negative observations. ♦

### F1 score

*F1 Score* is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account.

$$\text{F1 Score} = \frac{2 \cdot (\text{Recall} \cdot \text{Precision})}{(\text{Recall} + \text{Precision})}$$

### Example 10.3.7

$$\text{F1 Score} = \frac{2 \cdot (\text{Recall} \cdot \text{Precision})}{(\text{Recall} + \text{Precision})} = \frac{2 \cdot 0.70 \cdot 0.30}{0.70 + 0.30} = 0.42$$

For the logistic regression model on the **Default** data set, the F1 score thus is 0.42.



Intuitively, F1 Score is not as easy to understand as accuracy, but F1 is usually more useful than accuracy, especially if you have an uneven class distribution. Accuracy works best if false positives and false negatives have similar cost. If the cost of false positives and false negatives are very different, it is better to look at both Precision and Recall.

### Example 10.3.8

We can compute the F1 score by means of the `Python`-function `f1_score()` from `sklearn.metrics`: (to R)

```
[5]: """ Follows Example 3.3 """
from sklearn.metrics import f1_score

# Find F1-score
f1 = f1_score(y, y_pred, pos_label=1, average='binary')
print(f1)
```

0.42105263157894746



### Remarks:

- i. `pos_label` is an optional character string for the factor level that corresponds to a *positive* result. ◆

It is important to note that all measures that result from the confusion matrix depend on what we define as *positive*. We may as well define the case `default=No` as positive. In this case, the values of all measures will change.

### Example 10.3.9

If we consider the case `default=No` as positive, then the F1 score changes to (to R)

```
[6]: """ Follows Example 3.8 """
# Find F1-score
f1 = f1_score(y, y_pred, pos_label=0, average='binary')
print(f1)
```

0.9859154929577464



There are several approaches for coping with the problem of imbalanced classes. One of the simplest is *down-sampling* of the major class as the following example shows

### Example 10.3.10

We analyze the `Default` data set and fit a logistic regression model by downampling the `default=No` class to the same size as the `default=yes` case. (to R)

```
[7]: """ Follows Example 3.9 """
# Set random seed
np.random.seed(1)
# Index of Yes:
i_yes = df.loc[df['default_Yes'] == 1, :].index

# Random set of No:
i_no = df.loc[df['default_Yes'] == 0, :].index
i_no = np.random.choice(i_no, replace=False, size=333)

# Fit Linear Model on downsampled data
i_ds = np.concatenate((i_no, i_yes))
x_ds = df.iloc[i_ds]['balance']
y_ds = df.iloc[i_ds]['default_Yes']

x_sm = sm.add_constant(x_ds)

model_ds = sm.GLM(y_ds, x_sm, family=sm.families.Binomial())
```

## Chapter 10. Logistic Regression

```
model_ds = model_ds.fit()

# Predict for downsampled data
x_pred_ds = x_sm
y_pred_ds = model_ds.predict(x_pred_ds)

# Round to 0 or 1
y_pred_ds = y_pred_ds.round()

# Classification error on training data:
e_train = abs(y_ds - y_pred_ds)
e_train = e_train.mean()

print(np.round(e_train, 4))
```

0.1171

```
[8]: # Create confusion matrix
confusion = pd.DataFrame({'predicted': y_pred_ds,
                           'true': y_ds})
confusion = pd.crosstab(confusion.predicted, confusion.true,
                        margins=True, margins_name="Sum")

print(confusion)
```

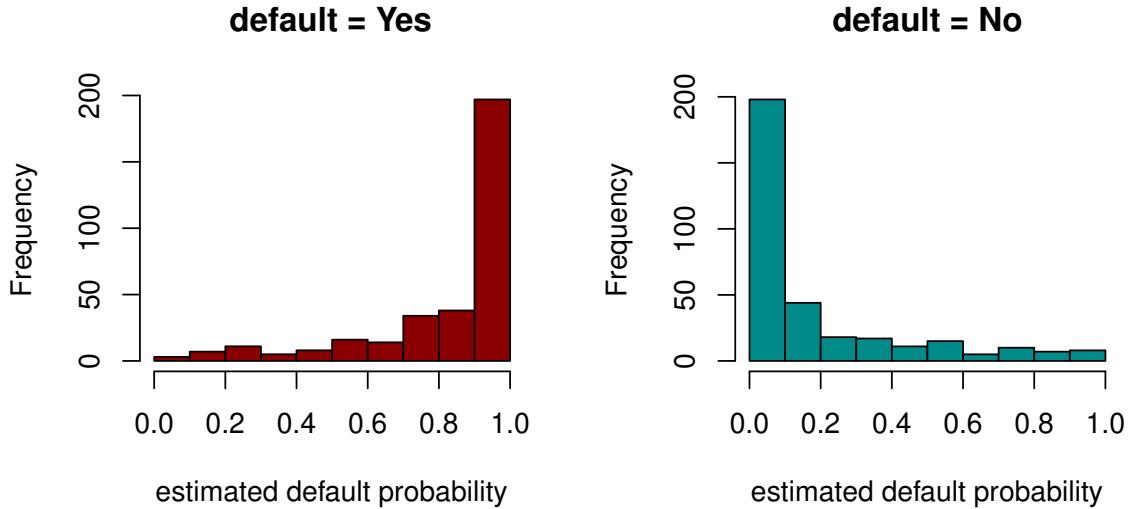
predicted	0	1	Sum
true			
0.0	293	38	331
1.0	40	295	335
Sum	333	333	666

```
[9]: # Print F1-scores
f1_pos = f1_score(y_ds, y_pred_ds, pos_label=1, average='binary')
f1_neg = f1_score(y_ds, y_pred_ds, pos_label=0, average='binary')

print('\nF1-Score (positive = default) = \n', f1_pos,
      '\nF1-Score (positive = not-default) = \n', f1_neg)
```

```
F1-Score (positive = default) =
0.8832335329341318
F1-Score (positive = not-default) =
0.8825301204819278
```

On the downsampled training set, the confusion matrix is balanced, and the classification error is 0.1171 which amounts to 88.29 % correctly classified samples. As we observe now, the F1 score for **default=Yes** as positive case has now considerably improved.



**Figure 10.5.:** Histograms of predicted class probabilities for the downsampled data set.

Furthermore, the histograms of the predicted probabilities have a complete different shape than before. The separation of the two classes becomes clearly visible (cf. Figure 10.5).



## 10.4. Cross Validation

So far we have evaluated our logistic regression model by a classification assessment measure such as accuracy. To this aim, we have taken the entire training data set. Validating the predictive accuracy of a statistical model on the same data the model was built from is by no means a good idea. One possible way out would be to collect new data with known labels and validate the model by computing the classification error on this new set (known as *validation set* or *test data* in this context). However, new labelled data is often hard to come up with. And even when at hand, one would rather use the additional information to improve the model instead of testing it.

An alternative is *cross validation*, which is a generic method for estimating the classification error on a validation set without the need of extra data. The procedure consists of randomly dividing the training data in  $k$  groups (or *folds*) of approximately the same size. Then the first fold is treated as validation set, and the model is fit on the remaining  $k - 1$  folds. The classification error  $\text{Err}_1$  is computed on the first fold.

This process is iterated over all  $k$  folds yielding classification errors  $\text{Err}_1, \dots, \text{Err}_k$ . The estimated classification error is then computed as

$$\text{CV}_{(k)} = \frac{1}{k} \sum_{i=1}^k \text{Err}_i. \quad (10.8)$$

The special case where  $k$  equals the number  $n$  of training set samples amounts to iteratively choose *one* element of the training data as validation "set". This is referred to as *leave-one-out cross validation*. For large data sets, this is computationally not feasible, so that  $k$ -fold cross validation with smaller values of  $k$  is preferred, typically ranging between 5 and 20.

### Example 10.4.1

The logistic model for the `Default` data set will now be evaluated with  $k$ -fold cross validation. We use the `cross_val_score()`-function from `sklearn.model_selection` for computing the estimated error. We choose  $k = 5$  and use the downsampled version of the training data.

(to R)

```
[1]: import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score

# Load data
df = pd.read_csv(
    '../.../Themen/Logistische_Regression/Data/Default.csv',
    sep=';')

# Add a numerical column for default
df = df.join(pd.get_dummies(df['default'],
                           prefix='default',
                           drop_first=True))

# Set random seed
np.random.seed(1)
# Index of Yes:
i_yes = df.loc[df['default_Yes'] == 1, :].index

# Random set of No:
i_no = df.loc[df['default_Yes'] == 0, :].index
i_no = np.random.choice(i_no, replace=False, size=333)

# Fit Linear Model on downsampled data
i_ds = np.concatenate((i_no, i_yes))
x_ds = df.iloc[i_ds][['balance']]
y_ds = df.iloc[i_ds]['default_Yes']
```

```
model = LogisticRegression()

# Calculate cross validation scores:
scores = cross_val_score(model, x_ds, y_ds, cv=5)

print(np.mean(scores))
```

0.8828077656828638

The cross validation score is 0.8828, thus 88.28% of the samples were correctly classified. ◀

The cross-validated classification error for the **Default** data is very close to the error on the training set. In real world applications, the training error is typically much smaller than the cross validated error, but of little use for estimating the predictive power of a model. Cross validation is an estimate for the prediction error on an *unknown* set, and unlike AIC, BIC etc., this approach is generic. It can be applied to any predictive model. We will encounter this method later again.

### Remarks:

- i. The “correct” procedure which we will follow in the succeeding chapters consists of first splitting the entire data set into a *training* and *test set*. Any optimization or hyperparameter tuning will be carried out on the training data set: either by further splitting the training data set into a *validation* and into a (smaller) *training* set or by means of cross-validation. Once the optimization of the classification model is finished, the optimized classification model will be applied to the test set. The performance on the test set will be reported. ♦

## 10.5. Multiple Logistic Regression

We finally study the model of a binary response variable  $Y$  depending on several predictor variables  $X_1, \dots, X_p$ . That is, we replace the linear function  $\beta_0 + \beta_1 X$  in the simple logistic regression approach by a multivariate linear function

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p.$$

### Logistic regression

Given a binary response variable  $Y$  and predictors  $X_1, \dots, X_p$ , the logistic regres-

sion model is defined by

$$P(Y|X_1, \dots, X_p) = \frac{e^{\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p}}{1 + e^{\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p}} \quad (10.9)$$

The parameters  $\beta_0, \beta_1, \dots, \beta_p$  are called regression coefficients and are estimated from the training set.

As with simple logistic regression, the coefficients  $\beta_0, \dots, \beta_p$  are estimated using the maximum likelihood method.

### Example 10.5.1

We fit a multiple logistic regression model to the **Default** data set using **balance**, **income**, and **student** as predictor variables. Note that the latter is a *qualitative* predictor with levels **Yes** and **No**. In order to use it in the regression model, we define a *dummy variable* with value 1 if **student=Yes** and 0 if **student=No**.

(to R)

```
[1]: import numpy as np
import pandas as pd
import statsmodels.api as sm

# Load data
df = pd.read_csv(
    '../../../Themen/Logistische_Regression/Data/Default.csv',
    sep=';')

# Add a numerical column for default
df = df.join(pd.get_dummies(df[['default', 'student']]),
             prefix={'default': 'default',
                      'student': 'student'},
             drop_first=True)

# Set random seed
np.random.seed(1)
# Index of Yes:
i_yes = df.loc[df['default_Yes'] == 1, :].index

# Random set of No:
i_no = df.loc[df['default_Yes'] == 0, :].index
i_no = np.random.choice(i_no, replace=False, size=333)

# Fit Linear Model on downsampled data
i_ds = np.concatenate((i_no, i_yes))
x_ds = df.iloc[i_ds][['balance', 'income', 'student_Yes']]
y_ds = df.iloc[i_ds]['default_Yes']

# Model using statsmodels.api
```

## Chapter 10. Logistic Regression

```
x_sm = sm.add_constant(x_ds)
model_sm = sm.GLM(y_ds, x_sm, family=sm.families.Binomial())
model_sm = model_sm.fit()

print(model_sm.summary())
```

```
Generalized Linear Model Regression Results
=====
Dep. Variable: default_Yes No. Observations: 666
Model: GLM Df Residuals: 662
Model Family: Binomial Df Model: 3
Link Function: logit Scale: 1.0000
Method: IRLS Log-Likelihood: -186.21
Date: Wed, 24 Mar 2021 Deviance: 372.42
Time: 15:49:13 Pearson chi2: 571.
No. Iterations: 7
Covariance Type: nonrobust
=====
      coef    std err      z   P>|z|      [0.025    0.975]
-----
const    -7.1303    0.869   -8.205   0.000    -8.833   -5.427
balance   0.0060    0.000   12.928   0.000     0.005   0.007
income   -1.454e-05  1.6e-05  -0.909   0.363   -4.59e-05  1.68e-05
student_Yes -0.8278    0.465   -1.780   0.075    -1.739   0.084
=====
```

```
[2]: # Predict training data
x_pred = x_sm
y_pred = model_sm.predict(x_pred)

# Round to 0 or 1
y_pred = y_pred.round()

# Create confusion matrix
confusion = pd.DataFrame({'predicted': y_pred,
                           'true': y_ds})
confusion = pd.crosstab(confusion.predicted, confusion.true,
                        margins=True, margins_name="Sum")

print(confusion)
```

	0	1	Sum
predicted			
0.0	293	36	329
1.0	40	297	337
Sum	333	333	666

```
[3]: from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score

# Model using sklearn
model_sk = LogisticRegression(solver='liblinear', penalty='l1')

# Calculate cross validation scores:
```

```
scores = cross_val_score(model_sk, x_ds, y_ds, cv=5)
print(np.mean(scores))
```

0.8827965435978005

First we find that the predictors **balance** and **student** are significant, i.e. they contribute substantially to the model for **default**. The coefficient of **student** is negative, i.e. the student status means a *decrease* in probability for default for a fixed value of **balance** and **income**.

Further we find a cross-validated score of 0.8828, which amounts to say that the model classifies correctly 88.28% of the cases. This is not much an increase compared with the single logistic regression model. Also the confusion matrix is very similar to the simple regression case.

We will now use the coefficients above in order to predict the probability for default for new observations. For example, if a student has a credit card bill of CHF 1500 and an income of CHF 40 000, so the estimated probability for **default** is

$$\hat{p}(1500, 40, 1) = \frac{e^{-6.679+0.00529\cdot1500-0.0043\cdot40-0.6468\cdot1}}{1 + e^{-6.679+0.00529\cdot1500-0.0043\cdot40-0.6468\cdot1}} = 0.564$$

For a non-student with the same balance and income the estimated probability for default is

$$\hat{p}(1500, 40, 0) = \frac{e^{-6.679+0.00529\cdot1500-0.0043\cdot40-0.6468\cdot0}}{1 + e^{-6.679+0.00529\cdot1500-0.0043\cdot40-0.6468\cdot0}} = 0.747$$

The coefficient for **income** is multiplied by 1000 for lucidity. Thus we insert 40 instead of 40000 into the model. ◀

## Conceptual Objectives

You should be able to ...

- ... identify a classification problem in practical applications.
- ... compute (*log*) *odds ratios* from probabilities and vice versa.
- ... model a binary classification problem by (simple and multiple) logistic regression.
- ... perform predictions given a logistic regression model and a new observation.
- ... derive classification error, false and true positive (negative) rates from a confusion matrix.
- ... assess the predictive power of a classification method by means of cross validation.

## Computational Objectives

You should be able to ...

- ... perform proper visualization of classification training data such as grouped boxplots and coloured scatter plots using the `boxplot()` and `plot()` command.
- ... building and interpret a logistic regression model using the `statsmodels.api.GLM()` command.
- ... predict the class for new data using the generic `predict()` function
- ... find the F1-Score using `sklearn.metrics.f1_score()` command.

## **Part III.**

# **Time Series Analysis**

# Chapter 11.

## Introduction to Time Series<sup>1</sup>

So far, we have worked on the fundamental concepts of regression and classification, i.e. the construction of *predictive models* from recorded data - the training set - in order to predict quantitative and qualitative response values, respectively. The key assumption for these models has been *independence* of the observations. In other words, the models so far have been *independent of the order* of the training data: Two linear regression models built from the same data set will always coincide, despite permutations of the rows in the data matrix.

Many real life measuring and data recording processes, however, result in data sets that are *serially correlated*. Examples of such situations are

1. *Machine monitoring*: Temperature, pressure, acoustic emissions, vibrations, etc. are measured at various locations in/at/around an operating engine (motor, generator, compressor, etc.).
2. *Stock*: Stock prices and exchange rates are recorded at the end of a trading day.
3. *Environmental observations*: Temperature, humidity, pollen concentration, pollution, precipitation are recorded at a specific weather station.
4. *Federal statistics*: Population census, income, accidents, ...

This kind of data is called *time series data*. Usually there are several goals that one wants to achieve with time series data:

- **Descriptive Analysis**: By means of summary statistics and visualizations, the basic properties of a time series are understood.
- **Modeling and Interpretation**: By modeling the underlying process that governs the observed time series, a deeper understanding can be gained. In particular, tests and confidence intervals/bands can be constructed from the model. The sequential dependence of the time series is also often quantified.

---

<sup>1</sup>This chapter follows Chapter 1 of the text book *Introductory Time Series with R* by Paul S.P. Cowpertwait and Andrew V. Metcalfe, Springer 2009.

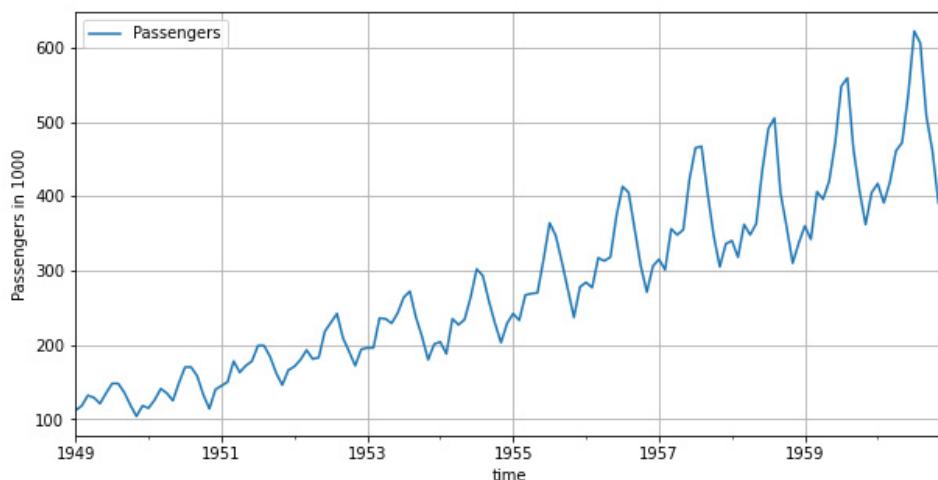
- **Decomposition:** Main feature of many time series is *seasonality*, i.e. a periodic pattern in the data, and a *trend*, a gradually changing average of the series which is directly correlated with the time axis. A major goal is to disaggregate these effects.
- **Prediction:** By means of the model, future values of the time series can be predicted. Prediction for time series is often alternatively termed *forecasting*.
- **Regression:** One often tries to explain a time series (*response*) by several other time series (*predictors*). This idea is wide-spread in industry, where the goal is to replace in a multi-sensor setup a particular (expensive or hard to install) sensor by a model that predicts its values from the other sensor values. This is called *virtual sensoring* or *soft sensoring*.

In this lecture we will address mainly the first 4 items. Before we dive into the mathematical concepts, we will present some examples of typical time series data.

### 11.1. Examples

#### Example 11.1.1

The **AirPassengers** dataset contains the number of airline passenger bookings (in thousands) per month on the airline *PanAm* from 1949 to 1960.



**Figure 11.1.:** Airline passenger bookings (in thousands) per month.

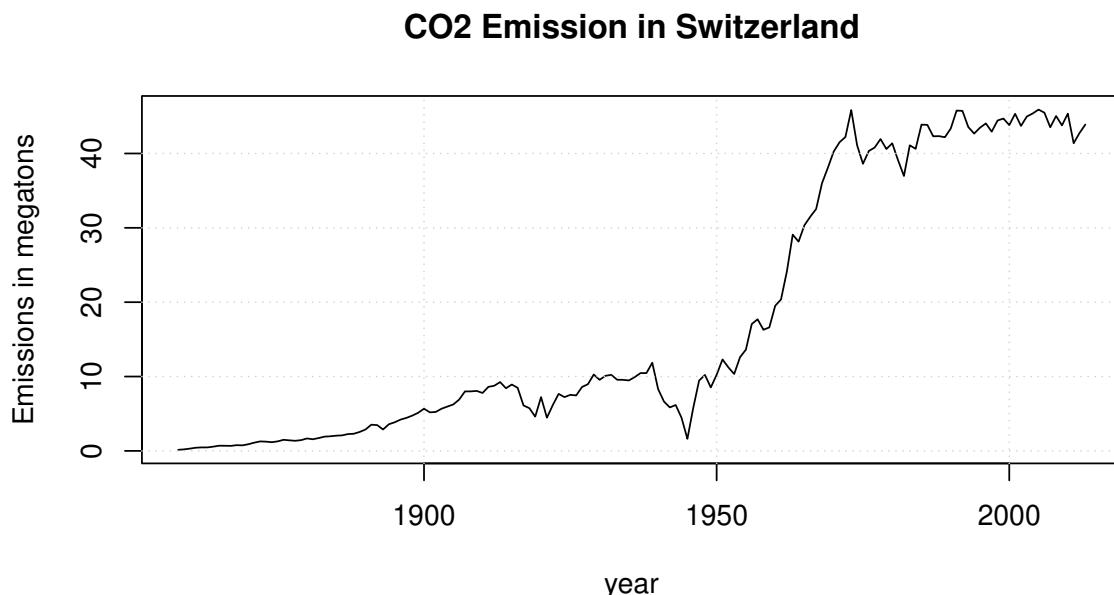
It is a classic data set often used to illustrate time series modeling. The data was originally used to forecast future flight bookings in order to plan aircraft and crew demand.

In Figure 11.1 the booking number is plotted against the time axis. Clearly the seasonality (more frequent flight bookings during the summer months) and a trend (global increase of flight bookings over time) are visible. 

### Example 11.1.2

The Kyoto Protocol is an amendment to the United Nations Framework Convention on Climate Change. It opened for signature in December 1997 and came into force on February 16, 2005. The arguments for reducing greenhouse gas emissions rely on a combination of science, economics, and time series analysis. Decisions made in the next few years will affect the future of the planet.

Figure 11.2 shows the yearly emission of the greenhouse gas CO<sub>2</sub> in Switzerland from 1858 to 2013<sup>2</sup>. Clearly, there is no seasonal effect (due to the yearly averaging) but a peculiar trend is observable: The emissions increased heavily in the post WW2 era between the 1950s and 1970s.



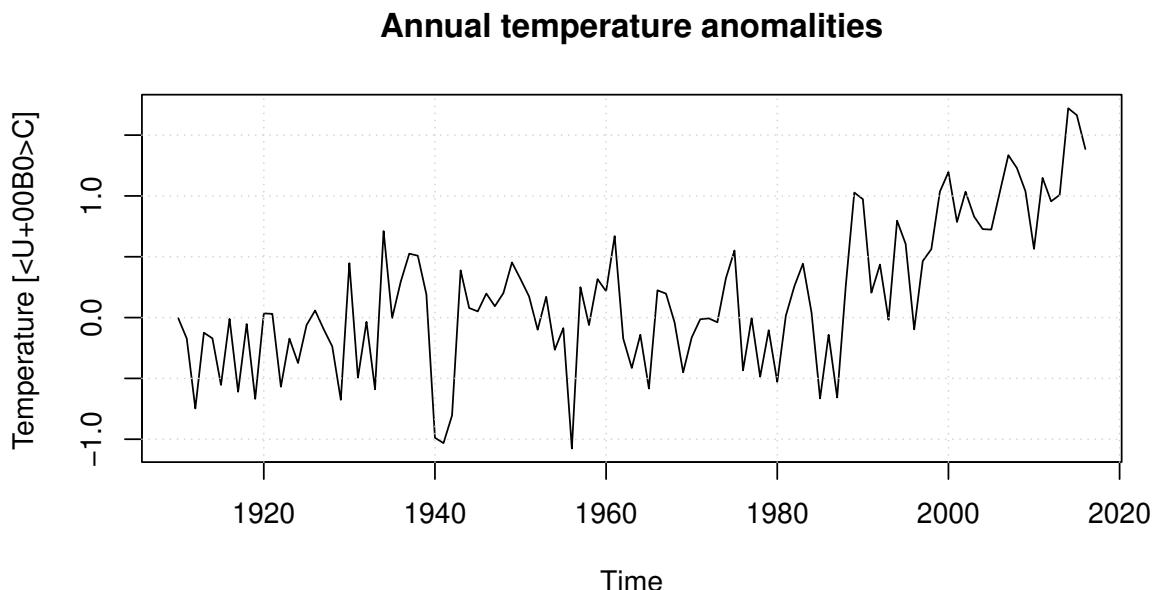
**Figure 11.2.:** CO<sub>2</sub> emissions in Switzerland.

Much in the same spirit and similarly threatening are data concerning global warming. In Figure 11.3 the annual average temperature anomalies with respect to the

---

<sup>2</sup>The data is taken from CAIT Climate Data Explorer. 2017. Washington, DC: World Resources Institute. Available online at: <http://cait.wri.org>

average between 1910 and 2000 in Europa are shown<sup>3</sup>. There is an upward trend starting in the 1980s, but the time series itself does not tell anything about the reasons for the temperature increase. Keeping in mind Figure 11.2 above, however, at least a correlation between CO<sub>2</sub> emission and global warming is not deniable.



**Figure 11.3.:** Annual temperature anomalies in Europe with respect to the average.

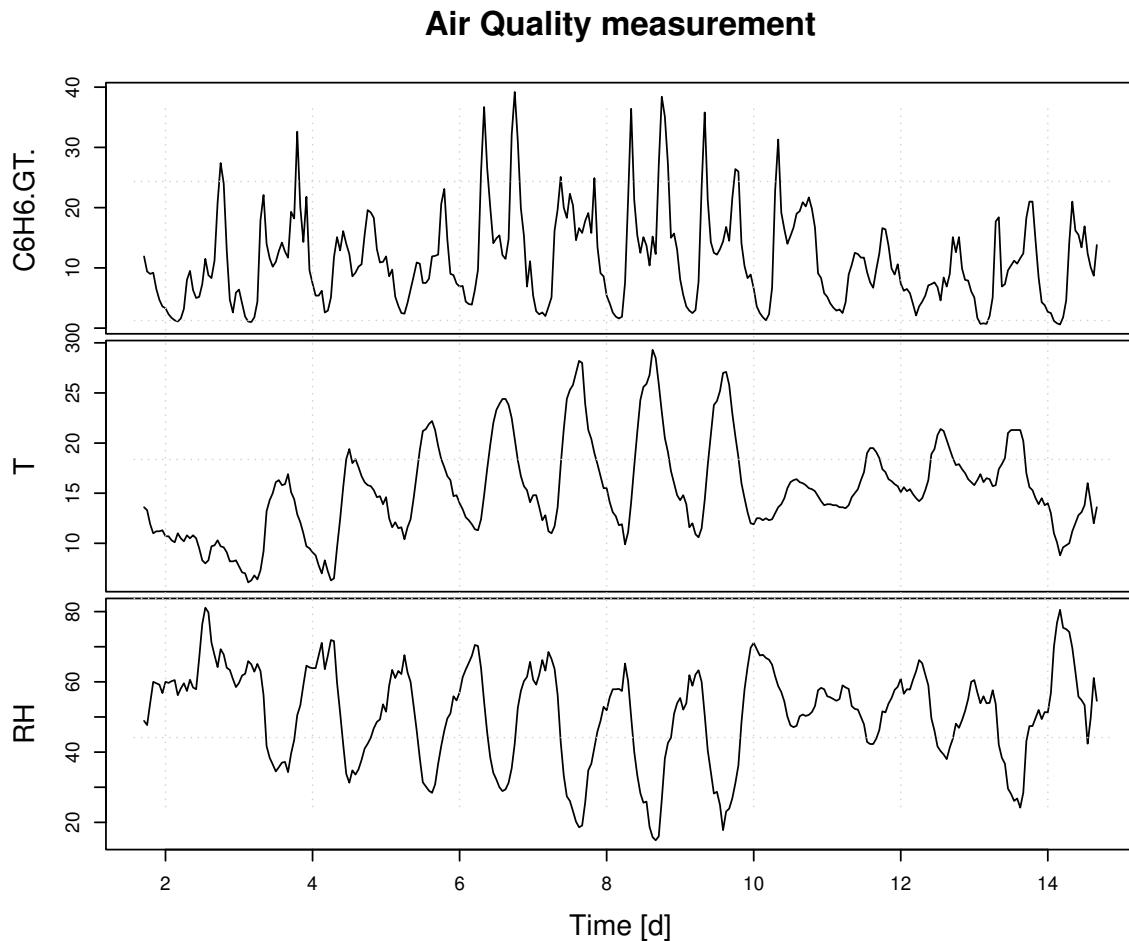
### Example 11.1.3

The **AirQuality** data set contains 9358 instances of hourly averaged responses from an array of 5 metal oxide chemical sensors embedded in an Air Quality Chemical Multisensor Device<sup>4</sup>. The device was located on the field in a significantly polluted area, at road level, within an Italian city. Data were recorded from March 2004 to February 2005. All in all, 13 variables are measured by the device.

In Figure 11.4 the concentration of benzene (C<sub>6</sub>H<sub>6</sub>), the air temperature in C° and the relative humidity (in %) are plotted over a period of 2 weeks. The seasonality of temperature and humidity is clearly explained by day time, i.e. in the afternoons the temperature and humidity have their maximum. The benzene concentration, however, is related to rush hour traffic.

---

<sup>3</sup>The data is taken from NOAA National Centers for Environmental information, Climate at a Glance: Global Time Series, published March 2017, retrieved on April 16, 2017 from



**Figure 11.4.:** City air quality measurements: Concentration of Benzen (upper panel), air temperature (center panel) and relative humidity (RH) (lower panel) are displayed.

#### Example 11.1.4

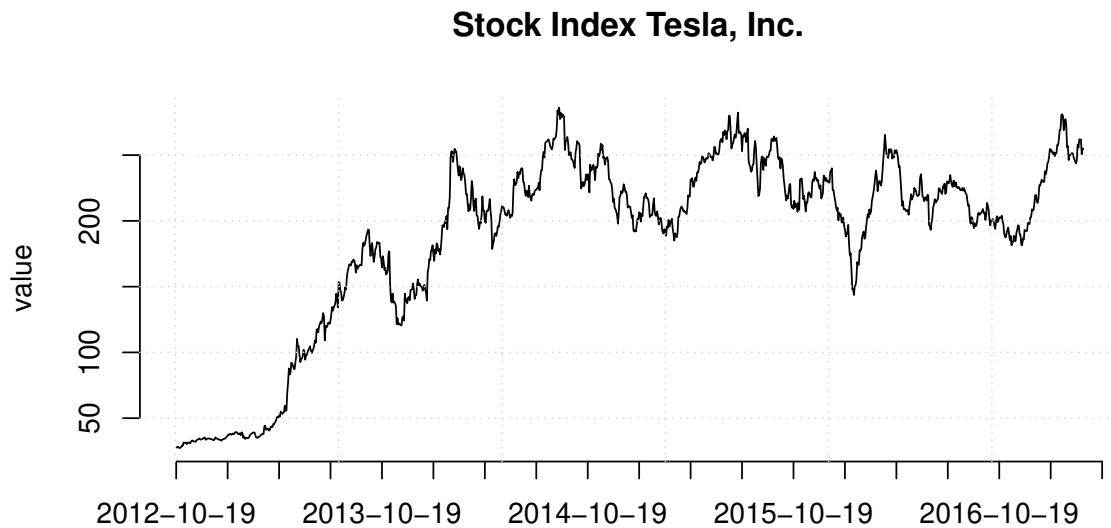
A very typical instance of time series are trading indices, and exchange rates in economics. Stock indices are often analyzed and subject to prediction attempts. The trends in stock indices, however, are nearly impossible to forecast. In this example we look at the stock index of Tesla, Inc. (cf. Figure 11.5).

Daily closing of 1112 consecutive trading days starting at 2012-10-19 are shown. As it can be seen, the Tesla stock index had a huge increase between March and June 2013. There seems to be a major breakdown around February 2016 with a subsequent sharp

---

<http://www.ncdc.noaa.gov/cag/>

<sup>4</sup>S. De Vito et al., *On field calibration of an electronic nose for benzene estimation in an urban pollution monitoring scenario*, Sensors and Actuators B: Chemical, **129(2)**, Pages 750-757.



**Figure 11.5.:** Daily closings of Tesla, Inc. stock index

increase. Comparing these trends with announcements of the company, one might unveil correlations e.g. between the presentation of the Model 3 in April 2016.

Instead of the stock index itself, the so-called *log-returns*, i.e. the day-to-day changes of the logarithm of the index, are analyzed and modeled. The log-returns for the Tesla, Inc. (in percent) are shown in Figure 11.6.

The log-returns represent an approximation of the relative change with respect to the previous trading day. It is obvious that there is no trend anymore and we will later see, that this data is uncorrelated. As a consequence, making predictions for the log-returns based on historical data is a fruitless endeavour.

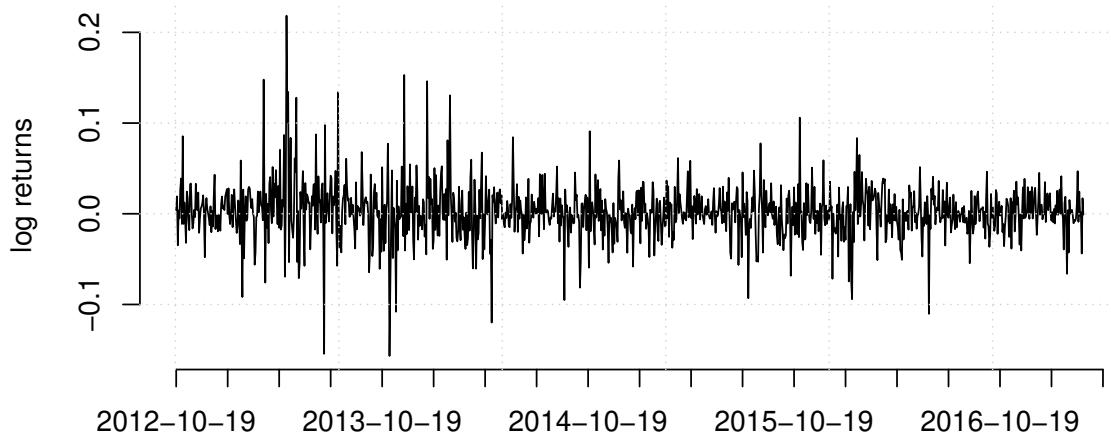


## 11.2. Time Series with Python

### Example 11.2.1

In `Python`, we have been using the `pandas` package for handling data. `pandas` provides a wide range of *methods* and *attributes*, such as `read_csv()`, `DataFrame.columns` and `DataFrame.shape`. (to `R`)

### Log returns of Tesla, Inc.



**Figure 11.6.:** Log-RetURNS of the Tesla, Inc. stock index.

```
[1]: import pandas as pd

# Load data
AirP = pd.read_csv(
    '../../Themen/Time_Series_Introduction/Data/AirPassengers.csv',
    parse_dates=True)

print("type:", type(AirP))
print("Columns:", AirP.columns)
print("Dimensions:", AirP.shape)
```

type: <class 'pandas.core.frame.DataFrame'>  
 Columns: Index(['TravelDate', 'Passengers'], dtype='object')  
 Dimensions: (144, 2)

#### 11.2.1. The DatetimeIndex class

##### Basic properties

### Example 11.2.2

The basic class to represent a time series with `pandas` is using `DatetimeIndex`. [\(to R\)](#)

```
[1]: import pandas as pd

# Load data
AirP = pd.read_csv(
    '../../Themen/Time_Series_Introduction/Data/AirPassengers.csv',
    parse_dates=True)
# Display head
print(AirP.head(3))

# Convert TravelDate to Datetime Format
AirP["TravelDate"] = pd.to_datetime(AirP["TravelDate"])

# Create pandas DateTimeIndex
dtindex = pd.DatetimeIndex(data=AirP["TravelDate"], freq='infer')
# Set as Index
AirP.set_index(dtindex, inplace=True)
AirP.drop("TravelDate", axis=1, inplace=True)

# Display head
print(AirP.head(3))
```

	TravelDate	Passengers
0	1/1/1949	112
1	2/1/1949	118
2	3/1/1949	132

	Passengers
TravelDate	
1949-01-01	112
1949-02-01	118
1949-03-01	132

The most important methods and attributes for `DatetimeIndex` are

1. the dates can be returned by normal indexing
2. `.month` returns the month of each entry
3. `.frequency` returns the frequency object

We study the output of the function for the `AirPassengers` data set. [\(to R\)](#)

```
[2]: # Start:
print(AirP.index[0])
# Month of last entry:
print(AirP.index.month[-1])
# Frequency
print(AirP.index.freq)
```

```
1949-01-01 00:00:00
12
<MonthBegin>
```

In `pandas`, the columns of a `DataFrame` can be easily plot using `.plot()`. This is the Python-Code that produced Figure 11.1 (to R)

```
[3]: import matplotlib.pyplot as plt

# Plot
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)
AirP.plot(ax=ax)
ax.set_xlabel("time")
ax.set_ylabel("Passengers in 1000")
ax.grid()

plt.show()
```



### Example 11.2.3

We study the quarterly beer production in Australia (in megalitres) between March 1956 and June 1994<sup>5</sup>. The base time unit is hence years and the frequency would be 4, i.e. one time step corresponds to 3 months. The start time is 1956 plus one sampling unit and the end time is 1994 plus 2 sampling units. (to R)

```
[1]: import numpy as np
import pandas as pd

# Load data
AusBeer = pd.read_csv(
    '../../../Themen/Time_Series_Introduction/Data/AustralianBeer.csv',
    sep = ";", header = 0)

# Convert TravelDate to Datetime Format
AusBeer["Quarter"] = pd.to_datetime(AusBeer["Quarter"])
# Create pandas DateTimeIndex
dtindex = pd.DatetimeIndex(data=AusBeer["Quarter"], freq='infer')
# Set as Index
AusBeer.set_index(dtindex, inplace=True)
AusBeer.drop("Quarter", axis=1, inplace=True)

# Print head
print(AusBeer.head(3))
```

---

<sup>5</sup>The data is available from <http://datamarket.com/data/list/?q=provider:tsdl> and is provided by the Australian Bureau of Statistics.

## Chapter 11. Introduction to Time Series

```
# Describe Data
print("\n", AusBeer.describe())
```

```
    megalitres
```

```
Quarter
```

```
1956-01-01      284.4
1956-04-01      212.8
1956-07-01      226.9
```

```
    megalitres
```

```
count   154.000000
mean    408.267532
std     97.598588
min    212.800000
25%   325.425000
50%   427.450000
75%   466.950000
max   600.000000
```

The `describe()` function shows the minimum, the first quartile, the median, the second quartile and the maximum of the time series. This is called the *five-number-summary* of a data set. In addition, the mean is also computed.

We now extract a subset of this time series. We choose the time window in between September 1980 and March 1990. The plot in Figure 11.7 shows a seasonal behaviour where we have peaks in the last quarter of each year which corresponds to the Australian summer. Moreover, a long-term trend of production increase is also visible which was strongest between 1960 and the mid 70s and stagnates since.

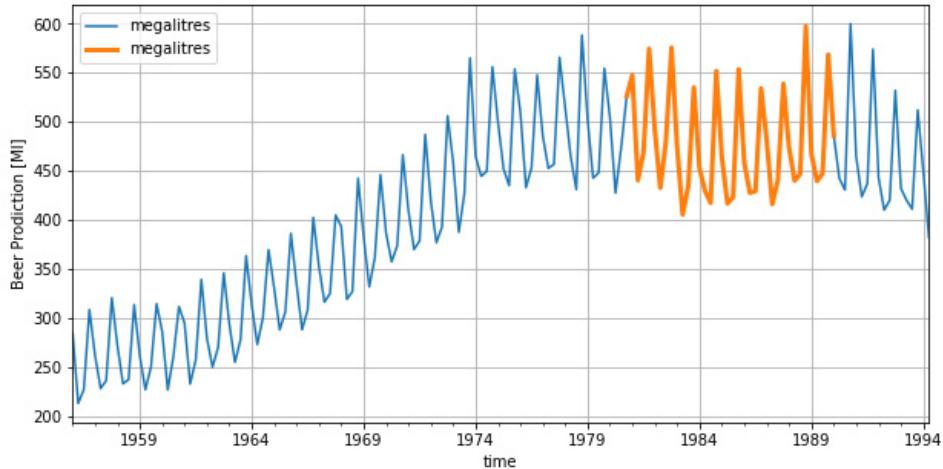
(to R)

```
[2]: import matplotlib.pyplot as plt

# Plot
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)
# All data
AusBeer.plot(ax=ax)
# Subset in red
AusBeer.loc["1980-9" : "1990-3"].plot(ax=ax, linewidth=3)

ax.set_xlabel("time")
ax.set_ylabel("Beer Production [Ml]")
ax.grid()

plt.show()
```



**Figure 11.7.:** The period from September 1980 until March 1990 is highlighted in red.

### Mutlivariate time series

#### Example 11.2.4

Here we illustrate a few important ideas and concepts related to multivariate time series data. The quarterly supply of electricity (millions of kWh) in Australia will be considered and compared to the already used data set on the quarterly beer production. First, we load the electricity data from file and create a time series out of it. [\(to R\)](#)

```
[1]: import numpy as np
import pandas as pd

# Load data
AusBeer = pd.read_csv(
    '../../Themen/Time_Series_Introduction/Data/AustralianBeer.csv',
    sep = ";", header = 0)
AusEl = pd.read_csv(
    '../../Themen/Time_Series_Introduction/Data/\nAustralianElectricity.csv',
    sep = ";")
# Create pandas DateTimeIndex
dtindexB = pd.DatetimeIndex(data=pd.to_datetime(AusBeer["Quarter"]),
                             freq='infer')
dtindexE = pd.DatetimeIndex(data=pd.to_datetime(AusEl["Quarter"]),
                             freq='infer')

# Set as Index
AusBeer.set_index(dtindexB, inplace=True)
AusEl.set_index(dtindexE, inplace=True)
AusBeer.drop("Quarter", axis=1, inplace=True)
```

## Chapter 11. Introduction to Time Series

```
AusEl.drop("Quarter", axis=1, inplace=True)

# Create new DataFrame combining both
Aus = pd.merge(AusBeer, AusEl, on="Quarter")

print(Aus.head(3))
```

```
      megalitres    kilowatt
Quarter
1956-01-01      284.4     3923
1956-04-01      212.8     4436
1956-07-01      226.9     4806
```

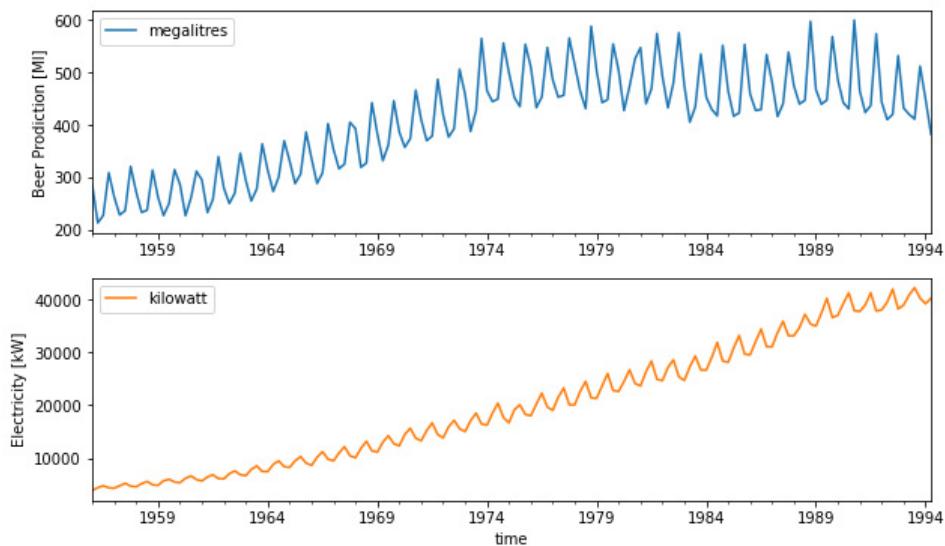
Now we plot the series: (to R)

```
[2]: import matplotlib.pyplot as plt

# Plot
fig, ax = plt.subplots(figsize=(10, 6), nrows=2)
# All data
Aus.plot(ax=ax[:,], subplots=True)

ax[0].set_xlabel("")
ax[1].set_xlabel("time")
ax[0].set_ylabel("Beer Production [Ml]")
ax[1].set_ylabel("Electricity [kW]")

plt.show()
```



**Figure 11.8.:** Beer production in megalitres and electricity production in millions of kWh.

The plots in Figure 11.8 show increasing trends in production for both goods, partly due to the rising population in Australia from about 10 million to about 18 million

over the same period. But notice that electricity production has risen by a factor of 7 during which the population has not quite doubled.

Note, that the two time series are correlated (with a correlation coefficient of 0.7), however there is of course no causal relation of the two series<sup>6</sup>. An explanation for the correlation is that the increasing prosperity and technological development in both countries over this period accounts for the increasing trends. ◀

### 11.3. Basic Transformation, Visualization, and Decomposition of Time Series

Analogous to the regression or classification approaches in Part II and Part III of the lecture notes, the analysis of time series begins with the description, transformation and visualization of the data. This invokes no modeling and hence does not give rise to proper predictions, confidence intervals etc. However, important insights and a profound understanding of the data can be achieved by these techniques. In this section we will describe the most important data transformations in the context of time series, a toolbox of helpful visualization techniques for exploring time series. Finally we will embark on the decomposition of time series into seasonal, trend and irregular components; in particular the STL decomposition.

#### 11.3.1. Data transformations

In many situations, it is desirable or even necessary to transform a time series before the application of models and predictions. In particular, many methods assume a

- **Gaussian** or at least a **symmetric** distribution of the data
- **linear** trend relationship between time and data
- **constant variance** across time

So for highly skewed or heteroskedastic data, it is often better not to use the original series  $\{x_1, x_2, \dots\}$  but a transformed series  $\{g(x_1), g(x_2), \dots\}$ . A family of transformations, that is well suited for correcting skewness and variance are the *Box-Cox-transformations*

#### Box-Cox Transformations

<sup>6</sup>A series of similar and very funny spurious correlations can be found on <http://www.tylervigen.com/spurious-correlations>.

For a time series  $\{x_1, x_2, \dots\}$  with positive values the Box-Cox transformations are defined as

$$g(x) = \begin{cases} \frac{x^\lambda - 1}{\lambda} & \text{if } \lambda \neq 0 \\ \log(x) & \text{if } \lambda = 0. \end{cases}$$

The reasoning behind the Box-Cox family is to choose the parameter  $\lambda$  such that desired properties hold. We illustrate this with the **AirPassengers** data

### Example 11.3.1

We consider the **AirPassengers** data. We apply the Box-Cox transformation for different values of  $\lambda$  to the data. In Figure 11.9 we see the resulting plots. The original data exhibits clear seasonal effects and an upward trend. The intensity of the seasonal influence, i.e. the variance over time, is also increasing. As we can see, the parameter  $\lambda = 0$ , i.e. the log-transform of the data, yields a stabilized image: a seemingly linear trend with homogeneous seasonal effects. [\(to R\)](#)

```
[1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Load data
AirP = pd.read_csv(
    '../../../Themen/Time_Series_Introduction/Data/AirPassengers.csv',
    parse_dates=True)

# Create pandas DateTimeIndex
dtindex = pd.DatetimeIndex(data=pd.to_datetime(AirP["TravelDate"]),
                            freq='infer')
# Set as Index
AirP.set_index(dtindex, inplace=True)
AirP.drop("TravelDate", axis=1, inplace=True)

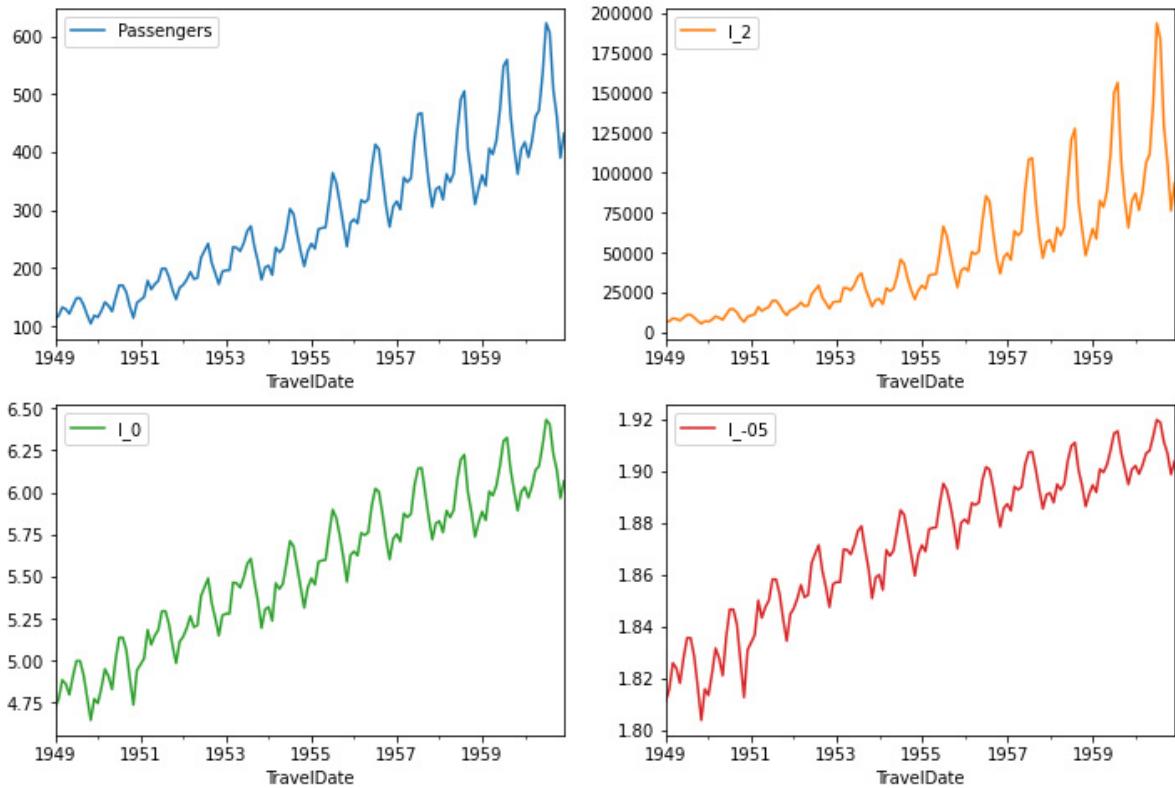
# Boxcox Definition:
def boxcox(x, lambd):
    if (lambd == 0):
        return np.log(x)
    else:
        return (x ** lambd - 1) / lambd

# Apply transform
AirP["l_2"] = boxcox(AirP["Passengers"], 2)
AirP["l_0"] = boxcox(AirP["Passengers"], 0)
AirP["l_-05"] = boxcox(AirP["Passengers"], -.5)

# Plots
```

```
fig, ax = plt.subplots(figsize=(12, 8), nrows=2, ncols=2)

AirP.plot(ax=ax[:, :], subplots=True)
plt.show()
```



**Figure 11.9.:** Box-Cox-transformations for different values of  $\lambda$  for the `AirPassengers` data sets.



The Box-Cox family of transforms amounts to a modification of the *values* of the time series. Sometimes, it is necessary to transform the *time-axis* as well. We will only study the most simple version of time transforms, namely *shifting*. A more general version of time shifts are often employed in speech recognition (audio signals are a specific sort of time series) termed *warping*.

### Time-shift transformation

Let  $\{x_1, x_2, \dots\}$  be a time series.

1. The time-shift by a *lag* of  $k \in \mathbb{Z}$  is defined by

$$g(x_i) = x_{i-k}$$

2. For the particular case where  $k = 1$  the time-shift is called *backshift*

$$B(x_i) = x_{i-1}$$

In other words, applying a time-shift to a time series amounts to go back  $k$  steps (if  $k > 0$ ) or go ahead  $-k$  steps (if  $k < 0$ ) in the series.

### Example 11.3.2

We look at the **AirPassengers** data and apply a time shift for various values of  $k$ . In Python we have the function `shift()` at our disposal. (to R)

```
[2]: # Perform shift
AirP["s_4"] = AirP["Passengers"].shift(-4)
AirP["s_-5"] = AirP["Passengers"].shift(5)

# plot Results
fig, ax = plt.subplots(figsize=(10, 5))

AirP.plot(y=["Passengers", "s_4", "s_-5"], ax=ax)
plt.legend(["Original", "Backward shift", "Forward shift"])
plt.show()
```

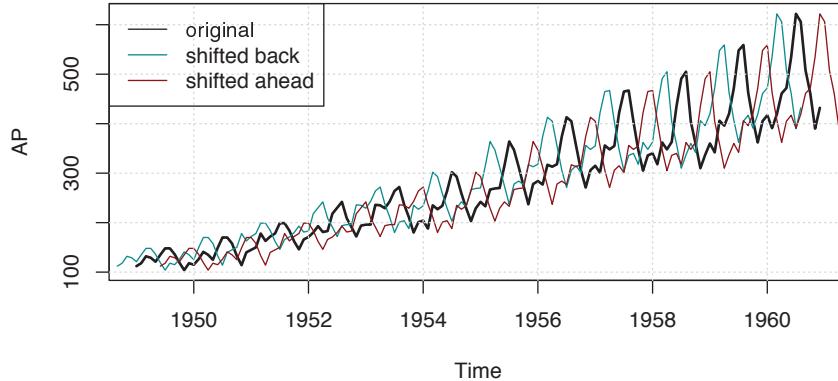


Figure 11.10.: **AirPassengers** data shifted back- and forwards.

The back-shift operator is applied when *differences* of time series are computed, since  $x_i - x_{i-1} = x_i - B(x_i)$ . In particular, differencing is often combined with Box-Cox transformations.

### Example 11.3.3

The *log-returns* of a (financial) time series are defined as

$$y_i = \log(x_i) - \log(x_{i-1}) = \log\left(\frac{x_i}{x_{i-1}}\right) = \log\left(\frac{x_i - x_{i-1} + x_{i-1}}{x_{i-1}} + 1\right) \approx \frac{x_i - x_{i-1}}{x_{i-1}}.$$

The last equation is due to the first Taylor series expansion of the logarithm:  $\log(s+1) = s - s^2/2 + \dots$ . Put differently, the log-return time series  $y_i$  approximates the relative increase of the time series  $x_i$  at each time instance. This quantity is often studied in financial applications: The original series  $\{x_1, x_2, \dots\}$  may exhibit seemingly significant patterns, but the series  $\{y_1, y_2, \dots\}$  often is rather random.

In example 11.1.4 we have examined the stock index of Tesla, Inc. and the corresponding log-returns. The curve in Figure 11.5 shows the stock index, the curve in Figure 11.6 the log-returns. The latter curve looks quite random despite some large fluctuations from time to time which is very typical for these kind of data. Analysts, among others, try to model the waiting time between these fluctuations. ◀

### 11.3.2. Visualizations

A most important part of descriptive analysis of a data set (not exclusively time series data) is a proper visualization. The *time series plot* is typically the first thing to do: Plot the time series data against the time/index. We have seen several examples of time series plots in Section 11.1. As you can see there, the discrete time points are joined by lines, despite the fact that they are not continuous. This is done automatically by the `plot()` function in Python.

When plotting a time series, interpretability of the plot heavily depends on the amount and smoothness of the data. Time series with large time domains and many data points should be split or summarized appropriately. This is illustrated with the following example.

### Example 11.3.4

In this example we study again the air quality data of example 11.1.3. We have hourly measurements of several sensors, where we now focus on the air temperature. ([to R](#))

```
[1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

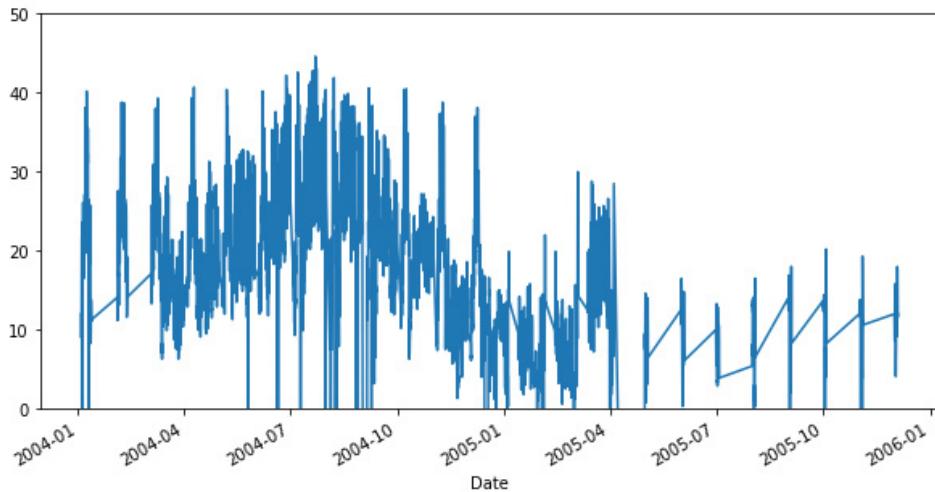
# Load data
AirQ = pd.read_csv(
    '../../../../../Themen/Time_Series_Introduction/Data/AirQualityUCI.csv',
```

## Chapter 11. Introduction to Time Series

```
parse_dates=True, decimal=", ", sep='; ')
```

```
# Create pandas DateTimeIndex
# Combine Date and Time Columns:
AirQ["Time"] = AirQ["Time"].str.replace(".", ":")
AirQ["Date"] = pd.to_datetime(AirQ["Date"] + " " + AirQ["Time"])
dtindex = pd.DatetimeIndex(data=pd.to_datetime(AirQ["Date"]),
                             freq='infer')
# Set as Index
AirQ.set_index(dtindex, inplace=True)
AirQ.drop(["Date", "Time"], axis=1, inplace=True)
```

```
[2]: # Plots
fig, ax = plt.subplots(figsize=(10, 5))
AirQ["T"].plot(ax=ax)
ax.set_ylim(0, 50)
plt.show()
```

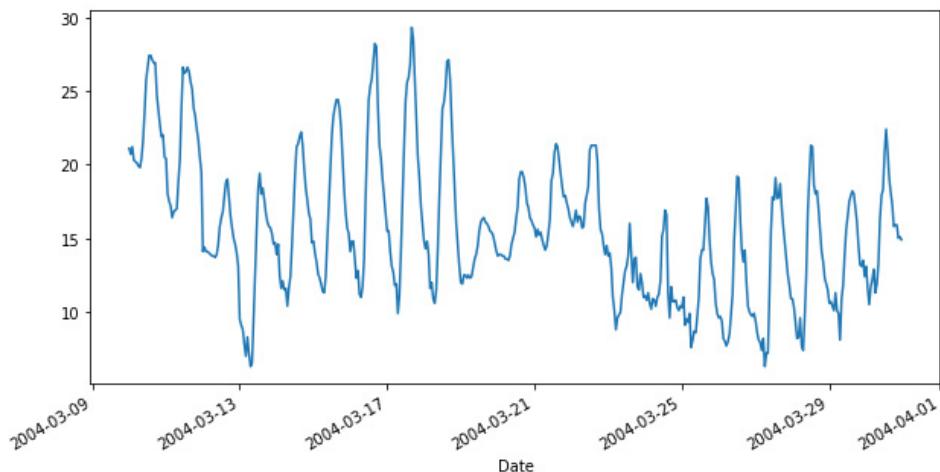


**Figure 11.11.:** Hourly air temperature in an Italian city.

In this global plot the seasonality over one year becomes visible, but the details of the series (such as daily seasonal effects) are hidden by the limited display space of a A4 page. The `ylim` option limits the temperature axis to nonnegative values. There are some spikes going to  $-200$  which have to be interpreted as sensor faults.

We focus on a period of 20 days to analyse the temperature behaviour in more detail. Figure 11.12 shows the data. (to R)

```
[3]: # From now on, only focus on 20 days
AirQ = AirQ.loc["2004-3-10":"2004-3-30"]
# Plot only detail:
fig, ax = plt.subplots(figsize=(10, 5))
AirQ["T"].plot(ax=ax)
plt.show()
```



**Figure 11.12.:** Hourly air temperatur of 20 consecutive days in March 2014.



Aggregation of data is often a good way to get a first idea of the behaviour of the underlying process. The next example shows how data aggregation can be visualized neatly with the `boxplot`-command.

### Example 11.3.5

We consider the air quality data from example 11.3.4. The time series `AirQ` contains the hourly air temperature for a period of 20 consecutive days in March 2014 in an Italian city. We are going to generate a boxplot for each full hour in one figure. To this end, the attribute `index.hour` is very convenient: it returns the hour for every index.

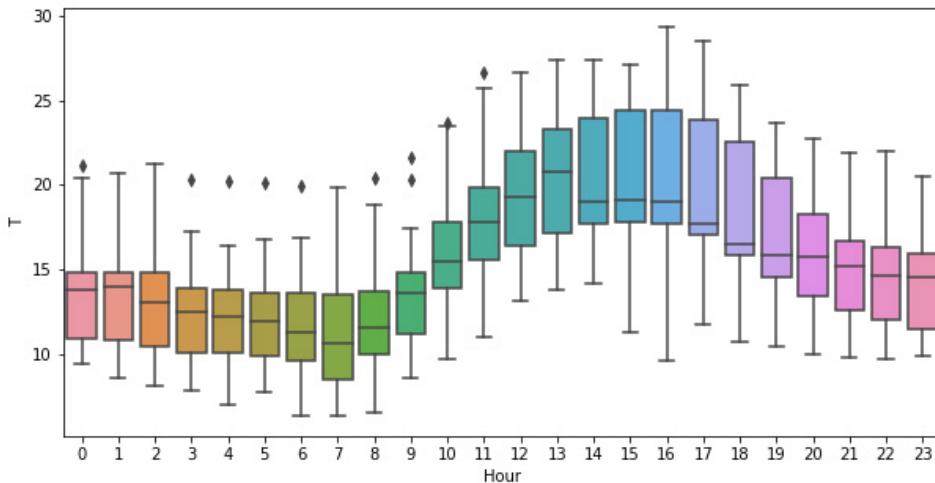
For the time being, we use the output of `.hour` as a grouping variable, e.g. for the `boxplot` function. The ouput of the following code snippet is shown in Figure 11.13 (to R)

```
[4]: import seaborn as sns

# Plot using Seaborn
fig, ax = plt.subplots(figsize=(10, 5))
AirQ['Hour'] = AirQ.index.hour
sns.boxplot(data=AirQ, x='Hour', y="T")

plt.show()
```





**Figure 11.13:** Each box corresponds to a particular hour of the day.

A useful graphical approach for visually inspecting correlations of consecutive observations are *lagged scatterplots*. They amount to produce scatterplots of the original time series values against a time-shifted version, i.e. plotting the data pairs  $(x_i, x_{i-k})$ .

### Example 11.3.6

We consider again the 20 day sequence of hourly air temperature measurements. We apply the `shift` function to create the shifted data, and plot these with respect to the original. (to R)

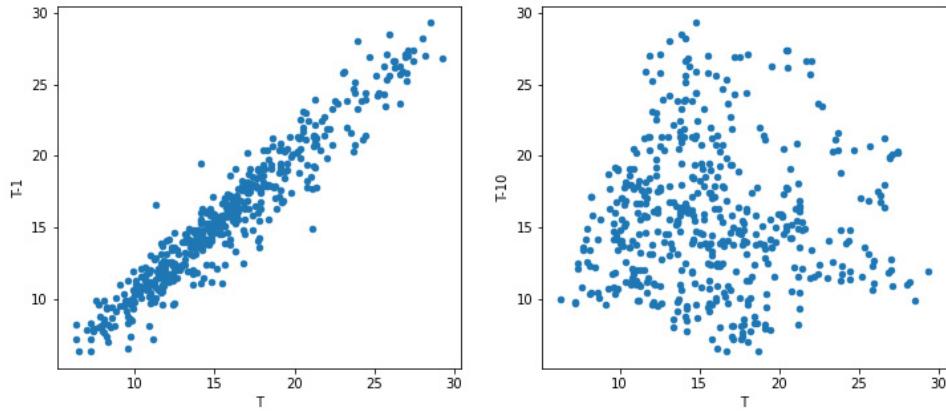
```
[5]: # Shift data
AirQ["T-1"] = AirQ["T"].shift(1)
AirQ["T-10"] = AirQ["T"].shift(10)

# Plot scatterplots
fig, ax = plt.subplots(figsize=(12, 5), ncols=2)

AirQ.plot(x="T", y="T-1", ax=ax[0], kind='scatter')
AirQ.plot(x="T", y="T-10", ax=ax[1], kind='scatter')

plt.show()
```

As it can be seen in Figure 11.14, the scatterplot with lag 1 shows a linear pattern which indicates a correlation between subsequent hourly air temperatures. A lag of 10 hours, however, results in a rather unspecific scatter plot. ◀



**Figure 11.14.:** The lag is chosen to be  $k = 1$  (left) and  $k = 10$  (right).

### 11.3.3. Decomposition of time series

As it became clear in several examples of Section 11.1, many time series are dominated by a trend and/or seasonal effects, so the models in this section are based on these components. A simple additive decomposition model is given by

$$x_k = m_k + s_k + z_k \quad (11.1)$$

where, at time index  $k$ ,  $x_k$  refers to the observed series,  $m_k$  denotes the trend,  $s_k$  is the seasonal effect, and  $z_k$  represents an error term that is, in general, a sequence of *correlated* random variables with mean zero. In this section, we briefly outline two main approaches for estimating the trend  $m_k$  and the seasonal effect  $s_k$ .

As it becomes obvious from the **AirPassengers** data, the seasonal effects may increase as the trend increases. Thus a multiplicative model seems more appropriate:

$$x_k = m_k \cdot s_k + z_k$$

If the noise is multiplicative as well, i.e.,

$$x_k = m_k \cdot s_k \cdot z_k$$

the logarithm of  $x_k$  is a linear model again

$$\log(x_k) = \log(m_k) + \log(s_k) + \log(z_k)$$

#### Moving Average

A very simple method for estimating the trend  $m_k$  and the seasonal effect  $s_k$  is by means of the moving average filter.

### Moving average filter

Assume that  $\{x_1, x_2, \dots, x_n\}$  is a time series and that  $p \in \mathbb{N}$ . The *moving average filter* of length  $p$  is defined as follows

- If  $p$  is odd, then  $p = 2l + 1$  and the filtered sequence is defined by

$$g(x_i) = \frac{1}{p}(x_{i-l} + \dots + x_i + \dots + x_{i+l})$$

- If  $p$  is even, then  $p = 2l$  and the filtered sequence is defined by

$$g(x_i) = \frac{1}{p} \left( \frac{1}{2}x_{i-l} + x_{i-l+1} + \dots + x_i + \dots + x_{i+l-1} + \frac{1}{2}x_{i+l} \right)$$

The value  $p$  is referred to as *window width*.

In other words, the moving average filter amounts to replace the  $i$ -th value in the time series by the average of the  $p$  nearest neighbors of  $x_i$ . If  $p$  is odd, then the window stretches symmetrically around  $x_i$ . For an even  $p$  one constructs a window of length  $p + 1$  (which is then odd) but counts the end points only by one half.

If a time series has the frequency  $p$  (e.g.  $p = 12$  for monthly data), then the trend component of the time series can be estimated by applying the moving average filter with window width  $p$  (in the case of monthly data,  $l$  is equal to 6, i.e. 6 months to the left and right are taken into account for determining the average at position  $i$ ). Since we average at each point exactly over one period, the seasonal effects vanish and the trend component remains. This yields the trend estimator  $\hat{m}_k$

### Example 11.3.7

We look again at the **AirPassengers** data and estimate the trend by a moving average filter. In **Python** we can select a window and average using `rolling()` and `mean()` respectively. (to **R**)

```
[1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Load data
AirP = pd.read_csv(
    '../../../Themen/Time_Series_Introduction/Data/AirPassengers.csv',
    parse_dates=True)

# Create pandas DateTimeIndex
dtindex = pd.DatetimeIndex(data=pd.to_datetime(AirP["TravelDate"])),
```

## Chapter 11. Introduction to Time Series

```

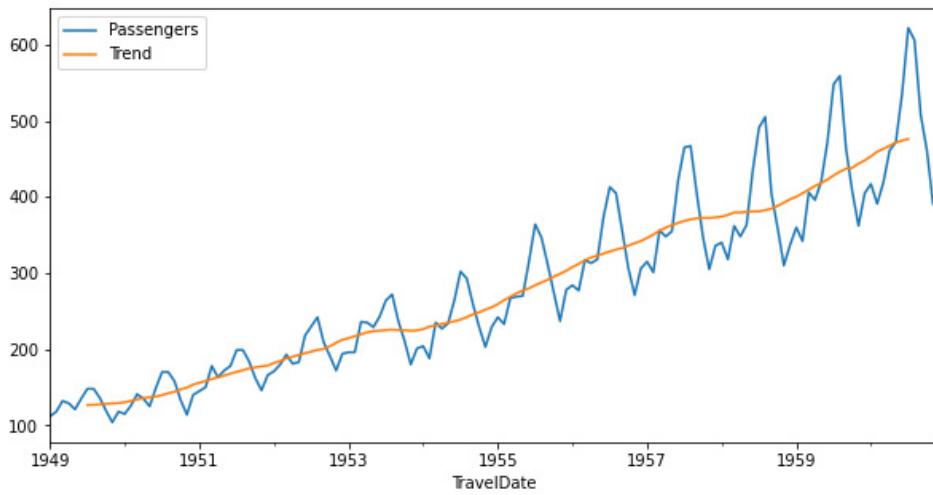
freq='infer')

# Set as Index
AirP.set_index(dtindex, inplace=True)
AirP.drop("TravelDate", axis=1, inplace=True)

# Calculate and save Trend
AirP["Trend"] = AirP["Passengers"].rolling(window=12, center=True).mean()

# Plots
fig, ax = plt.subplots(figsize=(10, 5))
AirP.plot(y=["Passengers", "Trend"], ax=ax)
plt.show()

```



**Figure 11.15.:** Estimated *Trend* for the `Airpassenger` data set.

The estimated trend  $\hat{m}_k$  line does not exhibit any seasonal fluctuations. ◀

To estimate the seasonal additive effect one computes

$$\hat{s}_k = x_k - \hat{m}_k$$

Now the time series  $\hat{s}_k$  is averaged for each time point in one cycle and we obtain a single estimate of the effect for each cycle point.

### Example 11.3.8

We take a look at the `AirPassengers` data and subtract the estimated trend in example 11.3.7 and average over the months. The average per month is found by going over every month in the dataset, selecting all entries in the same month, and taking the mean value. (to R)

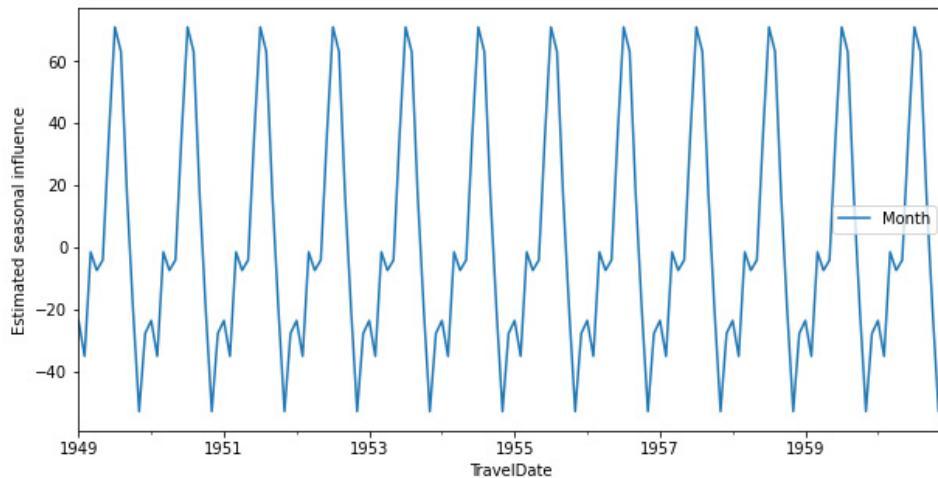
## Chapter 11. Introduction to Time Series

```
[2]: # Create new Column Season, where the Trend is subtracted
AirP["Season"] = AirP["Passengers"] - AirP["Trend"]

# Now find the mean per month
AirP["Month"] = np.zeros(AirP.shape[0])

# For every month, calculate mean:
for month in AirP.index.month.unique():
    mean = AirP.loc[AirP.index.month == month, "Season"].mean()
    AirP.loc[AirP.index.month == month, "Month"] = mean

# Plots
fig, ax = plt.subplots(figsize=(10, 5))
AirP.plot(y="Month", ax=ax)
ax.set_ylabel("Estimated seasonal influence")
plt.show()
```



**Figure 11.16.:** Estimated seasonal effect for the **Airpassenger** data set.

Finally, we subtract the trend and seasonality estimates and arrive at the estimate for the remainder term:

$$\hat{r}_i = x_i - \hat{m}_i - \hat{s}_i$$

The remainder term should consists of possibly correlated random values without structure/periodicity.

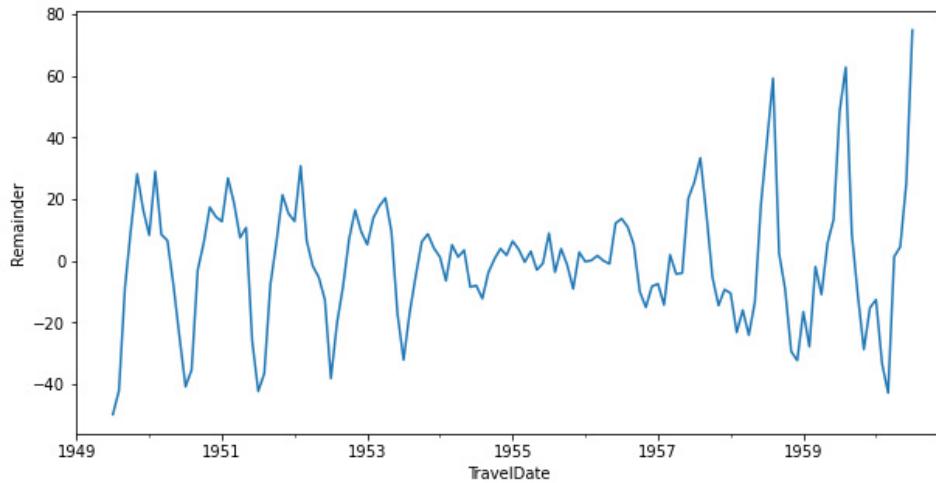
### Example 11.3.9

We again cosider the **AirPassengers** data and subtract the estimated trend and seasonal effects in examples 11.3.7 and 11.3.8. (to R)

## Chapter 11. Introduction to Time Series

```
[3]: # Create new Column remainder, where both season and Trend are subtracted
AirP["Remainder"] = AirP["Passengers"] - AirP["Trend"] - AirP["Month"]

# Plots
fig, ax = plt.subplots(figsize=(10, 5))
AirP["Remainder"].plot(ax=ax)
ax.set_ylabel("Remainder")
plt.show()
```



**Figure 11.17:** Estimated remainder term for the `AirPassengers` data. There is still non-random structure visible.

Figure 11.17 shows the estimated remainder term  $\hat{r}$ . It is striking that there is non-random structure left in the remainder term. The reason for that is, that the linear decomposition model is not true in this case. We hence repeat the steps above for the *logarithm* of `AirPassengers` which amounts to a multiplicative model. (to R)

```
[4]: # Manual Logarithmic transform
AirP_log = pd.DataFrame(
    data = {"log_Passengers": np.log(AirP["Passengers"])},
    index = AirP.index)

AirP_log["Trend"] = AirP_log["log_Passengers"]\ 
    .rolling(window=12, center=True)\ 
    .mean()

AirP_log["Season"] = AirP_log["log_Passengers"] - AirP_log["Trend"]
AirP_log["Month"] = np.zeros(AirP.shape[0])

# For every month, calculate mean:
for month in AirP_log.index.month.unique():
    mean = AirP_log.loc[AirP.index.month == month, "Season"].mean()
    AirP_log.loc[AirP_log.index.month == month, "Month"] = mean

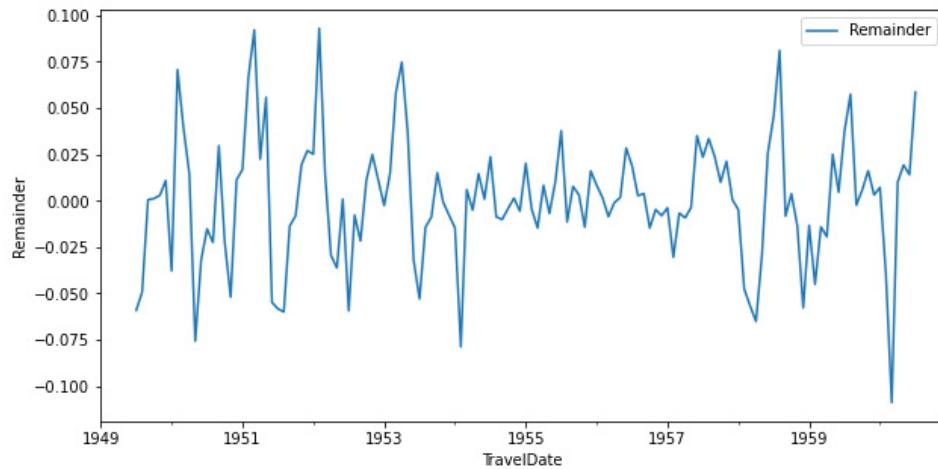
AirP_log["Remainder"] = AirP_log["log_Passengers"] \
    - AirP_log["Trend"] \
```

```

- AirP_log["Month"]

# Plots
fig, ax = plt.subplots(figsize=(10, 5))
AirP_log.plot(y="Remainder", ax=ax)
ax.set_ylabel("Remainder")
plt.show()

```



**Figure 11.18.:** Estimated remainder term for the logarithm of the `AirPassengers` data. The non-random structure has diminished.

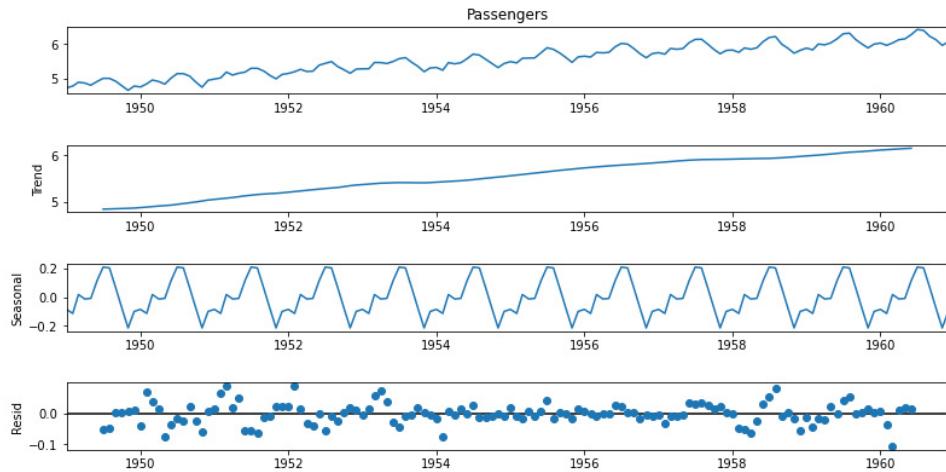
We see in the remainder estimated for the log-data in Figure 11.18 that the non-random parts in the signal have diminished considerably. ◀

There is a convenient way to perform a time series decomposition based on moving averages in Python. The `seasonal_decompose()` function from `statsmodels.tsa.seasonal` can be applied to a time series and performs the above steps. So the decomposition of the logarithmic `AirPassengers` data in example 11.3.8 can be performed by the following code: (to R)

```
[5]: # Decomposition on log-model using seasonal_decompose()
from statsmodels.tsa.seasonal import seasonal_decompose

decomp = seasonal_decompose(np.log(AirP["Passengers"]),
                           model = "additive")

# Plot
fig = decomp.plot()
fig.set_size_inches(12, 6)
plt.show()
```



**Figure 11.19.:** Decomposition time series according to additive time series model.  
Decomposition of time series according to an additive time series model.

### Seasonal Decomposition of Time Series by Loess (STL)

Although the decomposition method presented in Section 11.3.3 gives promising results for our example data set, it is rarely used in practice, for several reasons. Two of those are

- Lacking robustness with respect to outliers in the data.
- The seasonal component is assumed to be constant over time.

The state-of-the-art method for decomposing time series that does not suffer from the above drawbacks is *seasonal decomposition of time series by loess (STL)*. It is much in the same spirit as the method in Section 11.3.3 in the sense that the time series is smoothed in order to estimate the trend component and the seasonal effects are estimated from the detrended series. The major differences are

1. The STL procedure is iterative. In particular, outliers in the estimated remainder terms are detected and their effect mitigated by proper reweighting.
2. The moving average smoothing is replaced by loess regression which gives more flexibility and better results as the moving average. Loess regression is a form of *local* (mostly linear) regression which means that the regression line through data  $(x_1, y_1), \dots, (x_n, y_n)$  at a point  $x$  is only computed using the observations in a *neighborhood* of  $x$ . The width of the regression window (i.e. the size of the neighborhood), is a tuning parameter. If the window is larger than the span of the observations, then loess boils down to standard linear regression.
3. The seasonal component is not assumed to be constant. The method considers *cycle-subseries*, i.e. the subseries of values at each position of the seasonal cycle. For example, for a monthly series with frequency 12, the first cycle-subseries

consists of all January values, the second of the February values etc. The time index would be year for each element in a cycle-subseries. These cycle-subseries are also smoothed by loess and may change over time.

There are several tuning parameters in an STL decomposition, where the loess window size for the estimation of the seasonal component represents the most crucial parameter. In [Python](#) the method is implemented in the `STL()` function from `statsmodels.tsa.seasonal`, which is illustrated in the following example.

### Example 11.3.10

We examine the `AirPassengers` data set and decompose it by means of the `STL()` function. There are two mandatory parameters that have to be passed to the function

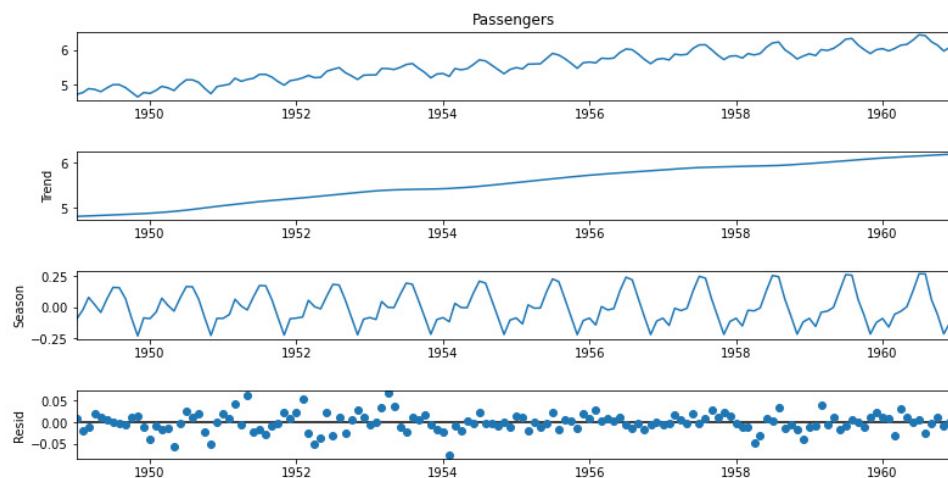
1. `x` the time series to be decomposed
2. `seasonal` the loess window size for the seasonality component. The larger the value the slower the change of seasonality in the data set over time.

The following [Python](#)-code estimates and plots an STL decomposition of the logarithm of the data ([to R](#))

```
[6]: from statsmodels.tsa.seasonal import STL

# Decomposition on log-model using STL
decomp = STL(np.log(AirP["Passengers"]), seasonal=9)
decomp = decomp.fit()

# Plot
fig = decomp.plot()
fig.set_size_inches(12, 6)
plt.show()
```



**Figure 11.20.:** STL decomposition of the log of the `AirPassengers` data with window size 10.

Figure 11.3.10 shows the decomposition of the data. Note that the seasonal component is changing over time and the remainder term is much smaller than in example 11.3.8.



## Conceptual Objectives

You should be able to ...

- ... list the main goals of time series analysis as well as at least three fields of applications of it.
- ... explain the difference between time series data and normal data tables that arise e.g. in linear regression
- ... provide the definition of the Box-Cox family of transformations and explain qualitatively in which respect the transformations change a given time series.
- ... formulate additive time series decomposition by moving averages in detail.
- ... explain roughly how the STL decomposition is defined and interpret the cycle subseries of the decomposition.

## Computational Objectives

You should be able to ...

- ... use `DatetimeIndex` in Pandas for time series modeling. In particular you can define `DatetimeIndex`-objects from vectors and interpret class methods and attributes such as `.month`, `.freq`, `shift()` etc.
- ... visualize and summarize time series with appropriate methods such as time series plots, lagged scatterplots and grouped boxplots.
- ... use the functions `seasonal_decompose` and `STL()` in order to decompose time series objects.

# Chapter 12.

# Mathematical Models for Time Series<sup>1</sup>

In the previous chapter we have introduced time series as (possibly multivariate) observations of data that can be ordered chronically in a natural way. So far we have studied transformations, visualization techniques and decomposition concepts. Moreover, we have studied time series computations in [Python](#). We now go a step further and start to *model* time series. This is a necessary step before we can *predict* future values of a series.

## 12.1. Mathematical Concept of Time Series

Aside to exploratory data analysis, i.e. summary statistics and plots, the primary objective of time series analysis is to develop mathematical models that provide plausible descriptions for sample data, like that encountered in Section 11.1. In order to provide a statistical setting for describing the characteristics of data that seemingly fluctuate in a random fashion over time, we assume a time series can be considered as the realization of random variables that are indexed over time.

### Time series and discrete stochastic processes

Let  $T$  be a set of equidistant time points  $T = \{t_1, t_2, \dots\}$ .

1. A *discrete stochastic process* is a set of random variables  $\{X_1, X_2, \dots\}$ . Each single random variable  $X_i$  has a univariate cumulative distribution function  $F_i$  and can be observed at time  $t_i$ .
2. A *time series*  $\{x_1, x_2, \dots\}$  is a realization of a discrete stochastic process  $\{X_1, X_2, \dots\}$ . In other words, the value  $x_i$  is a realization of the random variable  $X_i$  measured at time  $t_i$ .

---

<sup>1</sup>This chapter follows Sections 1.3 – 1.6 of the text book *Time Series Analysis and Its Applications* by Robert H. Shumway and David S. Stoffner, Springer 2011.

It is important to distinguish between a time series, i.e. a concrete observation of values, and a discrete stochastic process which is a theoretical construct to model the underlying mechanism that generates the values. We illustrate this by a simple yet important example.

### Example 12.1.1

Assume, that a person starts walking from the coordinate center with constant speed in  $x$ -direction. At each step, however, the person decides *at random* either to walk 1m to the left or to the right. This is the simplest instance of a *random walk*. The probabilistic model for this random walk would be

1. Choose  $n$  independent Bernoulli random variables  $D_1, \dots, D_n$  that take on the values 1 and  $-1$  with equal probability, i.e.  $p = 0.5$ .
2. Define the random variables  $X_i = D_1 + \dots + D_i$  for each  $1 \leq i \leq n$ . Then  $\{X_1, X_2, \dots\}$  is a discrete stochastic process modeling the random walk.

The following `Python`-code computes a particular instance of this process, i.e. a time series  $\{x_1, x_2, \dots\}$ . Each time this code is re-run, a new time series will display in Figure 12.1. The function `np.cumsum()` computes the cumulated sum of a given vector. ([to R](#))

```
[1]: import matplotlib.pyplot as plt
import numpy as np

nsamp = 10000
# Random samples d and cumulative sum x
d = np.random.choice(a=[-1,1], size=nsamp, replace=True)
x = np.cumsum(d)

# Plot
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)
ax.plot(x)
ax.grid()
plt.xlabel("Time")
plt.ylabel("y-deviation in [m]")

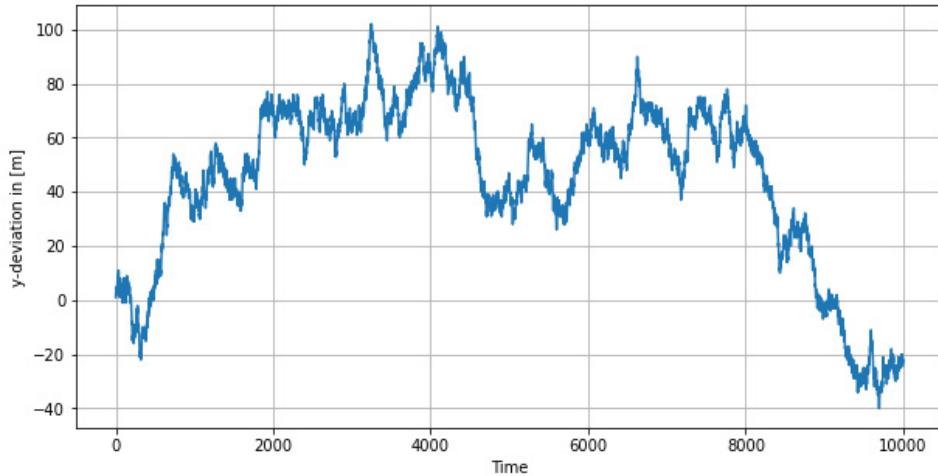
plt.show()
```

From the definition of the process it is clear that the following recursive definition is equivalent

$$X_i = X_{i-1} + D_i, \quad X_0 = 0.$$

If in each step a fixed constant  $\delta$  is added to the series, i.e.

$$Y_i = \delta + Y_{i-1} + D_i, \quad Y_0 = 0.$$



**Figure 12.1.:** A time series as observation of a random walk process.

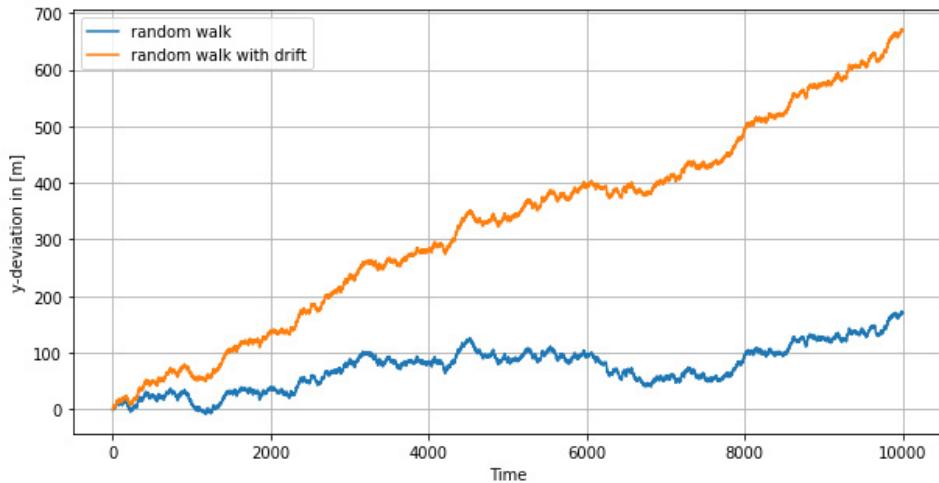
then we obtain a random walk *with drift*. In Figure 12.2 we see the observed time series of such a process. The random walk with drift models are used to model trends in time series data. [\(to R\)](#)

```
[2]: # Set random seed
np.random.seed(0)
# Random samples d and cumulative sum x
d = np.random.choice(a=[-1,1], size=nsamp, replace=True)
x = np.cumsum(d)
# Add drift delta
delta = 5e-2
y = np.linspace(0, nsamp*delta, nsamp)
y += x

# Plot
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)
ax.plot(x, label="random walk")
ax.plot(y, label="random walk with drift")
ax.grid()
plt.xlabel("Time")
plt.ylabel("y-deviation in [m]")
plt.legend()

plt.show()
```

A time series  $\{x_1, x_2, \dots, x_n\}$  can be understood as *one* realization of a multivariate random variable  $\{X_1, X_2, \dots, X_n\}$ . Modeling and prediction for time series hence amounts to analyze a data set with *one* observation, which is impossible without further assumptions on the series. We will work on such assumptions later in Section



**Figure 12.2.:** A time series observation of a random walk with drift. After subtracting the drift component we obtain a regular random walk (cyan).

12.3. For the time being we study an example, that lacks all these assumptions and thus is unpredictable: *white noise*.

### Example 12.1.2

A white noise process consists of independent and identically distributed random variables  $\{W_1, W_2, \dots\}$  where each  $W_i$  has mean 0 and variance  $\sigma^2$ . (to R)

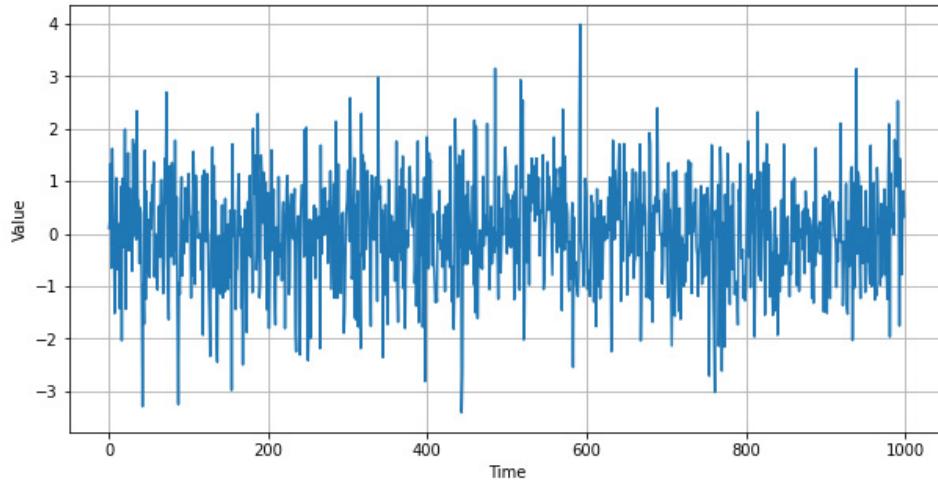
```
[1]: import matplotlib.pyplot as plt
import numpy as np

# White noise signal
w = np.random.normal(size=1000)

# plot
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)
ax.plot(w)
ax.grid()
plt.xlabel("Time")
plt.ylabel("Value")

plt.show()
```

If in addition the individual random variables  $W_i$  are normally distributed, then the process is called *Gaussian* white noise. These models are used to describe noise in engineering applications. The term *white* is chosen in analogy to white light and indicates that all possible periodic oscillations are present in a time series originating from a white noise process with equal strength. ◀



**Figure 12.3.:** A realization of the white noise process

The observations of a white noise process are uncorrelated and could hence be treated with ordinary statistical methods. We proceed with two further examples of discrete stochastic processes that can be generated from white noise that exhibit serial correlation. These two model classes will be treated in more detail at the end of the lecture.

### Example 12.1.3

If we apply a sliding window filter to the white noise process  $\{W_1, W_2, \dots\}$  in Example 12.1.2 we obtain a *moving average* process. E.g. if we choose the window length to be 3, we obtain

$$V_i = \frac{1}{3}(W_{i-1} + W_i + W_{i+1}).$$

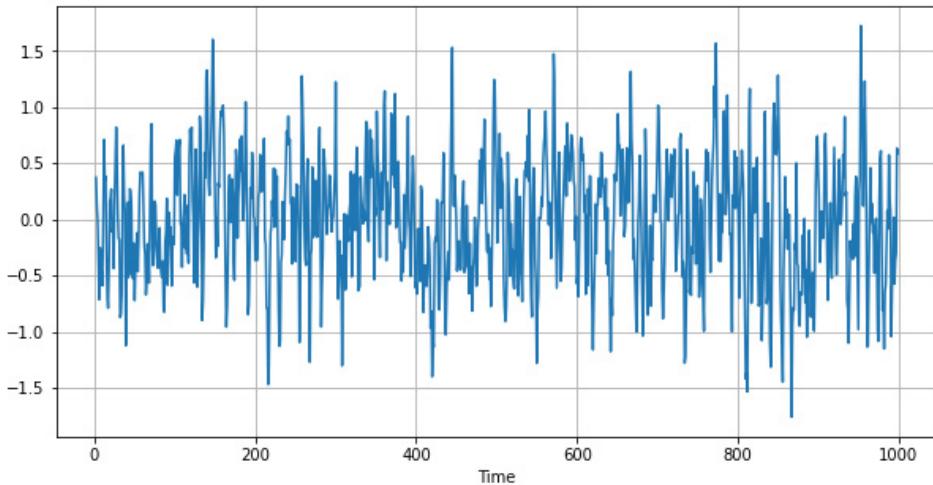
We choose  $V_1 = W_1$  and  $V_2 = 0.5(W_1 + W_2)$ . The resulting process is smoother, i.e. the higher order oscillations are smoothed out. In Python a sliding window can be achieved using `.rolling()`, the mean can be found with `.mean()`. (to R)

```
[2]: import pandas as pd

# Convert to DataFrame
w = pd.DataFrame(w)
# Filter with window = 3
v = w.rolling(window=3).mean()

# plot
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)
ax.plot(v)
ax.grid()
plt.xlabel("Time")
```

```
plt.show()
```



**Figure 12.4.:** A realization of a moving average process with window length 3.



Many examples of real world time series, e.g. acoustic time series in speech analysis, contain dominant oscillatory components, producing a sinusoidal type of behaviour. One possible model to generate such quasi-periodic data are autoregressive series.

#### Example 12.1.4

We consider again the white noise process  $\{W_1, W_2, \dots\}$  in Example 12.1.2 and recursively compute the following sequence

$$X_i = 1.5X_{i-1} - 0.9X_{i-2} + W_i.$$

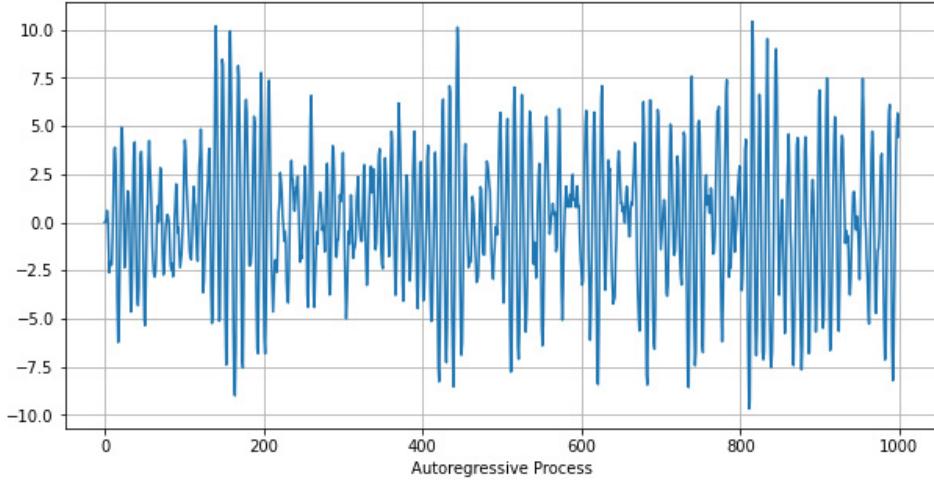
In other words, the value of the process at time instance  $i$  is modelled as a linear combination of the past two values plus some random component. Therefore this process is called *autoregressive*. The definition of the initial conditions is subtle, because the process will strongly depend on these. We will for the time being ignore the issue of initial conditions. (to R)

```
[3]: # Autoregressive filter:
ar = np.zeros(1000)
for i in range(2,1000):
    ar[i] = 1.5 * ar[i-1] - 0.9 * ar[i-2] + w.iloc[i]

# plot
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)
ax.plot(ar)
```

```

ax.grid()
plt.xlabel("Autoregressive Process")
plt.show()
    
```



**Figure 12.5.:** A realization of an autoregressive process which uses the last two past values to model the current outcome

Figure 12.5 shows a realization of the autoregressive process above. The oscillatory behaviour becomes clearly visible.

Another interpretation of the autoregressive process above is via differential equations. The finite difference scheme for the second order equation

$$\ddot{x} + 2\delta\dot{x} + \omega_0^2 x = W(t)$$

is given by

$$\frac{x_{i-2} - 2x_{i-1} + x_i}{\Delta t^2} + 2\delta \frac{x_i - x_{i-1}}{\Delta t} + \omega_0^2 x_i = W_i$$

Here  $\delta$  is the damping term and  $\omega_0$  the frequency of the homogeneous equation. Setting  $\Delta t = 1$ ,  $\omega_0^2 = 0.4$  and  $\delta = 0.05$  gives – after some rearrangements – the autoregressive process above. Hence it can be seen as a *harmonic oscillator with random input*. The wave length of the exact solution is  $T = 2\pi/\omega_0 \approx 10.0$  which matches the situation in Figure 12.5 pretty well. ◀

In the above examples we have motivated the use of various combinations of random variables to create time series that emulate real world scenarios. It is important to understand the statistical behaviour of such models in order to assess their accuracy. In the definition of a discrete stochastic process  $\{X_1, X_2, \dots\}$  we have postulated the existence of a cumulative distribution function  $F_i$  for each observation  $X_i$  in the process, i.e.

$$P(X_i \leq x) = F_i(x).$$

The knowledge of the individual distributions, however, is not sufficient to understand the serial behaviour of the process, because the observations are mutually *dependent*. It is known, that the complete probabilistic structure of such a process is determined by the *joint distribution of all finite collections*  $\{X_{i_1}, \dots, X_{i_n}\}$  of observations, i.e. one needs to find functions  $F_{i_1, i_2, \dots, i_n}$  such that

$$P(X_{i_1} \leq x_1, \dots, X_{i_n} \leq x_n) = F_{i_1, i_2, \dots, i_n}(x_1, \dots, x_n), \quad \text{for all possible indices } i_1, \dots, i_n.$$

In practice, one does not deal with this kind of multivariate distributions. Much of the information in these joint distributions can be described in terms of means, variances, and covariances. Consequently, we concentrate our efforts on these first and second order moments.

## 12.2. Measures of Dependence

Aside to the individual (also called *marginal*) distributions  $F_i$  of the random variables in the process, we define first and second order moments to analyse the whole process. We start with the mean sequence.

### Mean sequence

The mean sequence  $\{\mu(1), \mu(2), \dots\}$  (or mean function) of a discrete stochastic process  $\{X_1, X_2, \dots\}$  is defined as the sequence of the means:

$$\mu(i) = E[X_i].$$

### Example 12.2.1

We compute the mean sequence for some of the examples in Section 12.1.

1. If  $W_i$  denotes a white noise process, then  $E[X_i] = 0$  for all  $i \geq 1$ . Averaging the values in the process does hence not change the mean and the mean sequence of a moving average process as in Example 12.1.3 is hence also 0.
2. If  $X_i$  is a random walk with drift, i.e.  $X_0 = 0$  and

$$X_i = \delta + X_{i-1} + W_i$$

then we find that

$$\begin{aligned} E[X_1] &= \delta + E[X_0] + E[W_1] = \delta \\ E[X_2] &= \delta + E[X_1] + E[W_2] = 2\delta \\ E[X_3] &= \delta + E[X_2] + E[W_3] = 3\delta \\ &\vdots \end{aligned}$$

This means that  $\mu(i) = i\delta$



Next we move to second order moments of various sorts. We only consider the covariance of the observations within a *single* process. Cross-covariance between two individual series is not considered in this course.

### Autocovariance and autocorrelation

Let  $\{X_1, X_2, \dots\}$  be a discrete stochastic process.

1. The *autocovariance*  $\gamma_X$  is defined as

$$\gamma_X(i, j) = \text{Cov}[X_i, X_j] = E[(X_i - \mu(i))(X_j - \mu(j))].$$

2. The *autocorrelation*  $\rho_X$  is defined as

$$\rho_X(i, j) = \frac{\gamma_X(i, j)}{\sqrt{\gamma_X(i, i)\gamma_X(j, j)}}.$$

If it is clear from the context, we will skip the subscript  $X$ . It is important to note that both, the autocovariance and the autocorrelation, are symmetric, i.e.  $\gamma(i, j) = \gamma(j, i)$ . The autocovariance measures the *linear dependence* of two points on the same process observed at different times. If a series is very smooth, the autocovariance is large, even if  $i$  and  $j$  are far apart. Note that if  $\gamma(i, j) = 0$  this only means that  $X_i$  and  $X_j$  are not linearly dependent, however, they still may be dependent in some nonlinear way. For  $i = j$  the autocovariance reduces to the variance of  $X_i$ .

The autocorrelation can be described much in the same spirit, however, it is normalized, i.e.  $\rho(i, j) \in [-1, 1]$ . Put differently, if there is a linear relationship between  $X_i$  and  $X_j$ , then  $\rho(X_i, X_j) = \pm 1$ . To be more precise, if  $X_j = \beta_0 + \beta_1 X_i$ , then the autocorrelation will be 1 if  $\beta_1 > 0$  and -1 else. The autocorrelation hence gives a rough measure how well the series at time  $i$  can be forecast by the value at time  $j$ . We again compute the autocovariance and autocorrelation of some of the processes in Section 12.1.

**Example 12.2.2**

1. A white noise process as in Example 12.1.2 has the autocovariance function

$$\gamma(i, j) = \begin{cases} 0 & \text{if } i \neq j \\ \sigma^2 & \text{if } i = j. \end{cases}$$

Accordingly, the autocorrelation is 1 if  $i = j$  and 0 else.

2. We compute the autocovariance of the three point moving average process in Example 12.1.3. From the definition of the autocovariance it becomes clear that

$$\gamma(i, j) = \text{Cov}[X_i, X_j] = \text{Cov}\left[\frac{1}{3}(W_{i-1} + W_i + W_{i+1}), \frac{1}{3}(W_{j-1} + W_j + W_{j+1})\right].$$

When  $i = j$ , then we find

$$\begin{aligned} \text{Cov}[X_i, X_i] &= \frac{1}{9} \text{Cov}[W_{i-1} + W_i + W_{i+1}, W_{i-1} + W_i + W_{i+1}] \\ &= \frac{1}{9} (\text{Cov}[W_{i-1}, W_{i-1}] + \text{Cov}[W_i, W_i] + \text{Cov}[W_{i+1}, W_{i+1}]) \\ &= \frac{3\sigma^2}{9}. \end{aligned}$$

This follows from the fact, that  $W_i, W_{i-1}$  and  $W_{i+1}$  are mutually uncorrelated. For  $i+1 = j$  we find analogously

$$\begin{aligned} \text{Cov}[X_i, X_{i+1}] &= \frac{1}{9} \text{Cov}[W_{i-1} + W_i + W_{i+1}, W_i + W_{i+1} + W_{i+2}] \\ &= \frac{1}{9} (\text{Cov}[W_i, W_i] + \text{Cov}[W_{i+1}, W_{i+1}]) \\ &= \frac{2\sigma^2}{9}. \end{aligned}$$

To summarize, we find

$$\gamma(i, j) = \begin{cases} \frac{3\sigma^2}{9} & \text{if } i = j \\ \frac{2\sigma^2}{9} & \text{if } |i - j| = 1 \\ \frac{\sigma^2}{9} & \text{if } |i - j| = 2 \\ 0 & \text{else.} \end{cases}$$

This shows that smoothing of the white noise introduces a non-trivial autocovariance structure. It is noteworthy that the autocovariance only depends on the distance of the observations, but not on their value. We finally compute the autocorrelation  $\rho(i, j) = \gamma(i, j) / \sqrt{(\gamma(i, i)\gamma(j, j))} = \gamma(i, j) / \gamma(i, i)$ . This gives

$$\rho(i, j) = \begin{cases} 1 & \text{if } i = j \\ \frac{2}{3} & \text{if } |i - j| = 1 \\ \frac{1}{3} & \text{if } |i - j| = 2 \\ 0 & \text{else.} \end{cases}$$

3. We finally compute the autocovariance of the random walk process in Example 12.1.1. Recall, that the random walk process  $X_i$  is defined as the sum of independent Bernoulli random variables  $X_i = D_1 + \dots + D_i$  each with probability  $p = 0.5$ . Thus the variance of  $D_i$  is  $\sigma^2 = \text{Var}[D_i] = E[(D_i - E[D_i])^2] = E[D_i^2] = 1^2 \cdot p + (-1)^2(1-p) = p + (1-p) = 1$ . With this we find that

$$\gamma(i, j) = \text{Cov} \left[ \sum_{k=0}^i D_k, \sum_{l=0}^j D_l \right] = \min(i, j)\sigma^2.$$

Note that the autocovariance of the random walk not only depends on the lag of the observations but also on their particular time instances  $i$  and  $j$ . In particular, the variance of the process at time  $i$  is  $\text{Var}[X_i] = i\sigma^2$  and hence increases with time.

The autocorrelation function of the random walk can now easily be computed as

$$\rho(i, j) = \frac{\gamma(i, j)}{\sqrt{\gamma(i, i)\gamma(j, j)}} = \frac{\min(i, j)}{\sqrt{i \cdot j}}.$$



## 12.3. Stationarity

As we have stated above, a time series can be seen as a *single* realization of a multivariate random variable  $\{X_1, X_2, \dots\}$  which we call in this special setting a stochastic process. As we already know from elementary statistics, there is no way to do sound statistical analysis on a single observation. Hence we need a concept of regularity that allows us to infer information from a single time series: stationarity.

### Strict stationarity

A discrete stochastic process is called *strictly stationary* if for each finite collection  $\{X_{i_1}, \dots, X_{i_n}\}$  and each lag  $h \in \mathbb{Z}$  the shifted collection

$$\{X_{i_1+h}, \dots, X_{i_n+h}\}$$

has the same distribution. Put differently:

$$P(X_{i_1} \leq c_1, \dots, X_{i_n} \leq c_n) = P(X_{i_1+h} \leq c_1, \dots, X_{i_n+h} \leq c_n)$$

for any  $c_1, \dots, c_n$ .

The definition of strict stationarity amounts to say, that any selection of observations in the time series is a typical realization of the random variables in a selection of the process with the same pattern at any point in time. Loosely speaking, this means that the probabilistic character of the process does not change over time. Strict stationarity often is a too strong assumption for many applications and hard to assess from a single data set.

The definition of strict stationarity in particular implies for the special case  $n = 1$ , i.e. the "collection" of time instances in fact consists of one sample, that

$$P(X_i \leq c) = P(X_j \leq c)$$

for all  $i, j$  and for all  $c$ . This amounts to say that the marginal distributions  $F_i = F$  of the process coincide and in particular this implies that  $\mu_X(i) = \mu_X(j)$  or in other words, that the *mean sequence is constant*.

Further, we can consider the case when  $n = 2$ . Then the definition of strict stationarity implies that

$$P(X_i \leq c_1, X_j \leq c_2) = P(X_{i+h} \leq c_1, X_{j+h} \leq c_2).$$

The joint distribution of each pair of time instances in the process depends only on the time difference  $h$  and hence the autocovariance satisfies

$$\gamma(i, j) = \gamma(i + h, j + h).$$

These two conclusions from strict stationarity are sufficient for most applications in order to come up with reasonable statistical models. Hence they give rise to the definition of a weaker form of stationarity

### Weak stationarity

A discrete stochastic process  $X_i$  is called *weakly stationary* if

1. the mean sequence  $\mu_X(i)$  is constant and does not depend on the time index  $i$  and
2. the autocovariance sequence  $\gamma_X(i, j)$  depends on  $i$  and  $j$  only through their difference  $|i - j|$ .

From the discussion above it is clear that each strictly stationary time series is also weakly stationary. But the opposite is in general not true. It can be shown that for Gaussian processes, i.e. when all finite collections of random variables in the process have a joint normal distribution, the two notions of stationarity are equivalent.

Since the autocovariance/-correlation for a (weakly) stationary process only depends on the time lag  $h = i - j$  one considers these sequences as a function of  $h$  alone:

$$\begin{aligned}\gamma(h) &= \gamma(i, i + h) \\ \rho(h) &= \rho(i, i + h).\end{aligned}$$

Clearly, we have  $\gamma(h) = \gamma(-h)$  such that only values  $h = 0, 1, 2, \dots$  are considered.

### Example 12.3.1

We reconsider the three point moving average process in Example 12.2.2. It is obvious that the mean function  $\mu(i) = \mu = 0$  is constant and from the computations in Exercise 12.3.1 we see that the autocovariance only depends on the time lags. Thus the moving average process is weakly stationary. In particular, we find

$$\gamma(h) = \begin{cases} \frac{3\sigma^2}{9} & \text{if } h = 0 \\ \frac{2\sigma^2}{9} & \text{if } |h| = 1 \\ \frac{\sigma^2}{9} & \text{if } |h| = 2 \\ 0 & \text{else.} \end{cases}$$



### 12.3.1. Testing Stationarity

In practice, we are faced with time series, i.e. single observations of a discrete stochastic process. Stationarity is a property of the latter, though, and we have to *test* or at least *guess* whether or not the underlying process of a given time series is stationary.

A typical example is time series decomposition which we studied in Section 11.3.3. There, trend and seasonality of a time series are estimated and subtracted which yields the so called *remainder sequence*. A typical assumption for modeling the process is the (weak) stationarity of this remainder. So how to infer this from the data?

There are several statistical hypothesis tests for stationarity. However, most of them focus on a specific property of stationarity (as. e.g. the well known Dickey-Fuller test or the KPSS test) instead of stationary in a broad sense. There are good tests based in the frequency domain which are quite involved and would be beyond the scope of this course (e.g. Nelson's test based on Wavelets). Many of these tests are implemented in `Python` in `statsmodels.tsa.stattools`.

The first and simplest type of test one can apply to check for stationarity is to actually plot the time series and look for evidence of trend in mean, variance, autocorrelation and seasonality. If any such patterns are present then these are signs of non-stationarity and different mechanisms exist to turn the series into a stationary one, as for example data transformation and time series decomposition.

A further possibility is to compute the mean and autocovariance sequences separately for several windows and compare their behaviour (the computation of these quantities from data is treated in the upcoming Section 12.4). When there is a dramatic change, then the hypotheses of stationarity can be rejected.

### Example 12.3.2

We study the STL decomposition of two time series from Chapter 11. We begin with the quarterly Australian electricity production. We read the data into Python and decompose the logarithm of the data into a trend, a seasonal component and a remainder. We just plot the remainder of the series. (to R)

```
[1]: import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from statsmodels.tsa.seasonal import STL

# Load data
AusEl = pd.read_csv('./data/AustralianElectricity.csv', sep = ";")
# Create pandas DateTimeIndex
dtindexE = pd.DatetimeIndex(data=pd.to_datetime(AusEl["Quarter"]),
                             freq='infer')

# Set as Index
AusEl.set_index(dtindexE, inplace=True)
AusEl.drop("Quarter", axis=1, inplace=True)

# Decomposition on log-model using STL
decomp = STL(np.log(AusEl), seasonal=15)
decomp = decomp.fit()

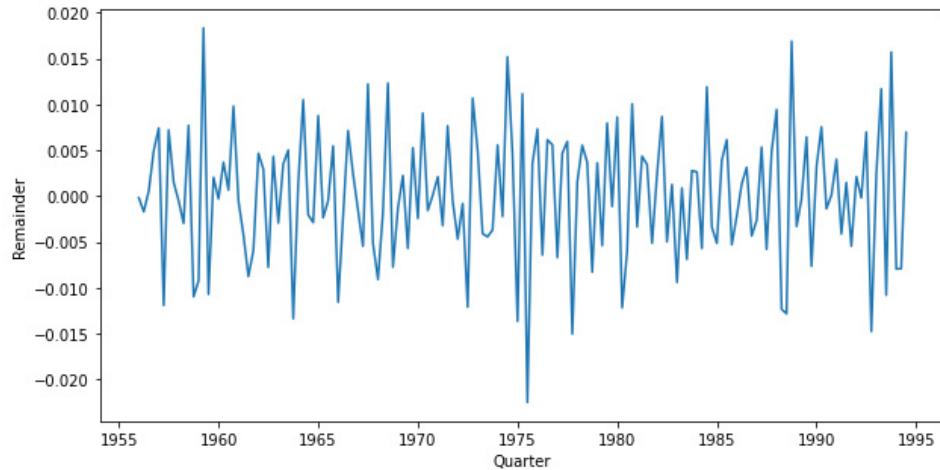
# plot
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)
ax.plot(decomp.resid)
plt.xlabel("Quarter")
plt.ylabel("Remainder")

plt.show()
```

The series in Figure 12.7 does not exhibit any seasonal patterns, has a constant mean (0) and roughly constant variance. From visual inspection one would conclude that the underlying process is stationary.

The next example is the air temperature measurement. We again read the data, choose a window of appropriate size and coerce it into a time series. Afterwards we decompose the data (without log transform) by STL and study the remainder sequence. (to R)

## Chapter 12. Mathematical Models for Time Series



**Figure 12.6.:** The remainder sequence of the Australian electricity data after log-transformation and STL decomposition.

```
[2]: # Load data
AirQ = pd.read_csv('./data/AirQualityUCI.csv',
                    parse_dates=True, decimal=",", sep=';')

# Combine Date and Time Columns:
AirQ["Time"] = AirQ["Time"].str.replace(".", " ")
AirQ["Date"] = pd.to_datetime(AirQ["Date"] + " " + AirQ["Time"])
dtindex = pd.DatetimeIndex(data=pd.to_datetime(AirQ["Date"]),
                            freq='infer')

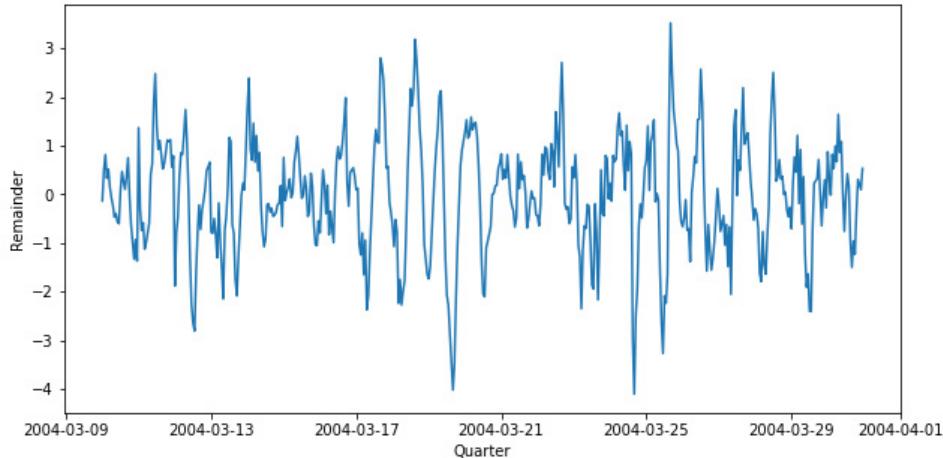
# Set as Index
AirQ.set_index(dtindex, inplace=True)
AirQ = AirQ.sort_index()
# Only keep temperature for given period
AirT = AirQ.loc["2004-3-10":"2004-3-30", "T"]
# AirT = pd.DataFrame(AirQ["T"])

# Decomposition on log-model using STL
decomp = STL(AirT, period=24, seasonal=9)
decomp = decomp.fit()

# plot
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)
ax.plot(decomp.resid)
plt.xlabel("Quarter")
plt.ylabel("Remainder")

plt.show()
```

The time series plot in Figure 12.7 still exhibits some seasonality, such that stationarity of the underlying process seems unlikely. Different choices of `seasonal` in the



**Figure 12.7.:** The remainder sequence of the temperature measurements in the air quality data set after STL decomposition.

[STL\(\)](#) function will change the behaviour of the remainder sequence. ▶

## 12.4. Estimation of Correlation

In Chapter 12.2 we have introduced the most important measures for serial dependence in time series. However, these concepts are restricted to the theoretical models and hence make use of the probabilistic behaviour at each time instance. In practice, however, we are confronted with a *single realization* of the process, i.e. at each time instance *only one* value is available. From a viewpoint of classical statistics, this is a problem. The assumption of stationarity is crucial in this context: it paves the way to sound statistical estimators for mean, auto- and crosscovariance sequences.

Throughout this chapter we will assume that the time series  $\{x_1, x_2, \dots, x_n\}$  is a realization of a weakly stationary process  $\{X_1, X_2, \dots, X_n\}$ .

### 12.4.1. Estimation of the Mean Sequence

Due to the stationarity of the process we know that the mean sequence  $\mu(k) = \mu$  is constant. A canonical estimator hence is

$$\hat{\mu} = \bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$$

The standard error of the mean  $\hat{\mu}$  for *independent* observations (i.e. a white noise process), is  $\sigma/\sqrt{n}$ . In the present situation the observations are dependent and we have to carefully recompute the standard error. It can be shown that

$$\text{Var}[\hat{\mu}] = \text{Var}\left[\frac{1}{n} \sum_{i=1}^n x_i\right] = \frac{1}{n} \sum_{h=-n}^n \left(1 - \frac{|h|}{n}\right) \gamma(h).$$

The standard error is the square root of the variance and depends on the autocovariance of the process. Note that the standard error of the process may be smaller or larger than the white noise case depending on the nature of the correlation structure. In order to compute the standard error for a given sequence, the autocovariance function has to be estimated.

### 12.4.2. Estimation of the Autocovariance

The theoretical autocovariance function of a stationary process is estimated by the sample autocovariance sequence which we define next

#### Sample Autocovariance

1. The *sample autocovariance* is defined by

$$\hat{\gamma}(h) = \frac{1}{n} \sum_{i=1}^{n-h} (x_{i+h} - \bar{x})(x_i - \bar{x})$$

with  $\hat{\gamma}(-h) = \hat{\gamma}(h)$  for  $h = 0, 1, \dots, n-1$ .

2. The *sample autocorrelation* is defined by

$$\hat{\rho}(h) = \frac{\hat{\gamma}(h)}{\hat{\gamma}(0)}.$$

Note that the sum runs over a restricted range of  $n-h$  points but we nevertheless normalize by  $n$  rather than  $n-h$ . Neither choice, however, results in an unbiased estimator.

#### Example 12.4.1

We compute the estimators above for simulated data. First we simulate a realization of a moving average process. ([to R](#))

```
[1]: import numpy as np
import pandas as pd
```

## Chapter 12. Mathematical Models for Time Series

```
np.random.seed(0)

# White noise signal
w = np.random.normal(size=1000, loc=2, scale=0.1)
w = pd.DataFrame(w)
# Filter with window = 3
w = w.rolling(window=3).mean()
```

In Python, the autocovariance and -correlation can be computed with the `acof()` and `acf()` (from `statsmodels.tsa.stattools`) respectively. During the application of the moving average (in the `rolling` command), missing values are created at the boundaries. To omit `NaN` entries, the setting `missing` is set to '`drop`'.

(to R)

```
[2]: import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import acovf

# Autocovariance
ac = acovf(w, fft=False, missing='drop', nlag=50)

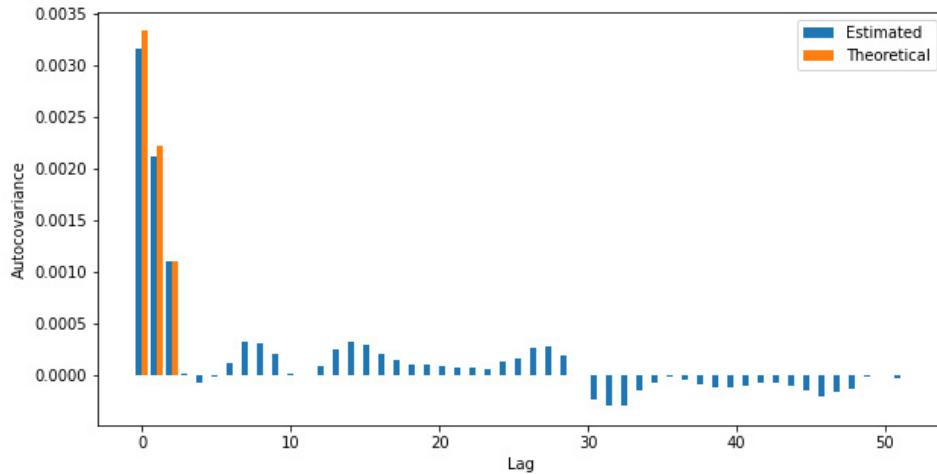
# Theoretical autocovariance
sigma = 0.1
act = np.zeros(len(ac))
act[0] = 3 / 9 * sigma**2
act[1] = 2 / 9 * sigma**2
act[2] = 1 / 9 * sigma**2

# plot
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)

width = 0.4
x = np.linspace(0, len(ac), len(ac))
ax.bar(x - width / 2, ac, width=width, label='Estimated')
ax.bar(x + width / 2, act, width=width, label='Theoretical')
plt.xlabel("Lag")
plt.ylabel("Autocovariance")
plt.legend()

plt.show()
```

The `nlag` parameter sets the maximal lag up to which the autocovariance should be computed. Figure 12.8 shows the autocovariance of the moving average data. It can be seen that the lags  $h = 0, 1, 2$  produce a significant value and that for  $h > 2$  the autocovariance is very small. This is in agreement with the theoretical results from Example 12.2.2. We add the values of the true autocovariance for comparison. Nevertheless, the sample autocovariance shows an oscillating pattern which we will comment on in Example 12.4.2 below. ◀



**Figure 12.8.:** Sample autocovariance function of a MA(3) process (blue) and the theoretical values (red).

The autocovariance function depends on the scale of the time series and hence it is cumbersome to compare the covariance structure of two series. Thus the autocorrelation is often more practical. Plotting the autocorrelation results in a similar plot as in Figure 12.8, however the  $y$ -axis is normalized. Then the graphical representation is called *correlogram*.

Under quite general assumptions it can be shown that for a white noise process for large  $n$  the distribution of the sample autocorrelation  $\hat{\rho}(h)$  for a fixed  $h$  is approximately normal with mean  $-1/n$  and standard deviation

$$\sigma_{\hat{\rho}} = \frac{1}{\sqrt{n}}.$$

This can be used to roughly check whether a time series stems from a white noise process or not. If the series is white, it should stay 95% of the time within the  $-1/n \pm 2/\sqrt{n}$  limits. These limits are automatically drawn by `Python` when calling `plot_acf()`.

Finally we remark on a systematic problem with autocorrelation estimation. It can be shown<sup>2</sup> that the sum of all possible sample autocorrelation coefficients with lag  $h \geq 1$  is always  $-1/2$ , i.e.

$$\sum_{h=1}^n \hat{\rho}(h) = -\frac{1}{2}$$

When the true autocorrelation sequence is positive (as with the moving average process in the example below) then there have to be significant negative values in the sample autocorrelation that of course are spurious. Some of these spurious, negative

<sup>2</sup>See e.g. H. Hassani, *A note on the sum of the sample autocorrelation function*, Physica A 389, pp. 1601–1606, 2010

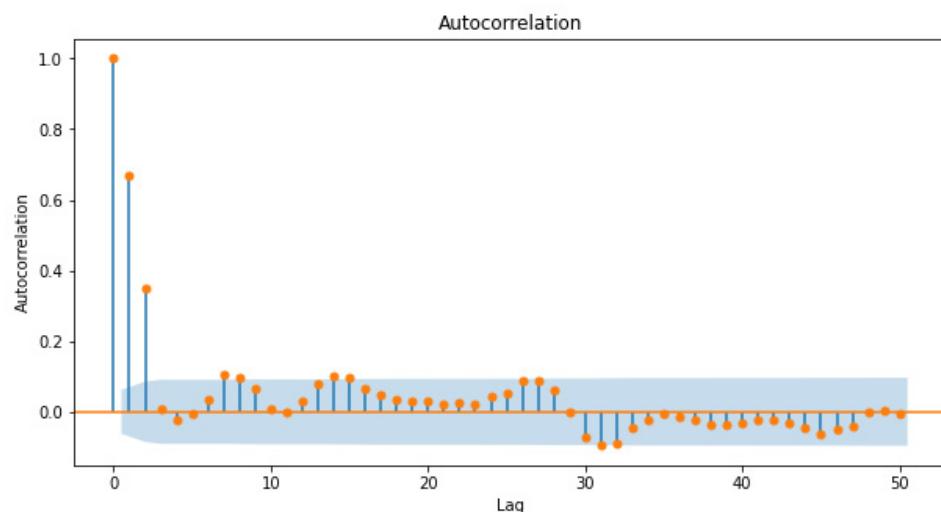
correlation estimates are so big that they even exceed the confidence bounds - an observation that has to be well kept in mind if one analyzes and tries to interpret the correlogram.

### Example 12.4.2

We again consider the moving average process in Example 12.4.1. The following code produces a correlogram for a given time series. ([to R](#))

```
[3]: from statsmodels.graphics.tsaplots import plot_acf

# plot
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)
plot_acf(w, missing='drop', lags=50, ax=ax, c='C1')
plt.xlabel("Lag")
plt.ylabel("Autocorrelation")
plt.show()
```



**Figure 12.9.:** Sample autocorrelation function of a MA(3)) process

The two-sigma confidence bands are drawn automatically and we see that starting from the 3rd all lags produce an autocorrelation which is in the band. In practice, once the estimated autocorrelation stays within these bands one sets it to 0<sup>3</sup>. ◀

---

<sup>3</sup>A more formal way of testing is the *Ljung-Box test* that tests the null-hypothesis that a number of autocorrelation coefficients is zero. The test is implemented in the `statsmodels.stats.diagnostic.acorr_ljungbox()`

## Conceptual Objectives

You should be able to ...

- ... explain the difference between a time series and a discrete stochastic process.
- ... provide the definition of white noise, random walk (with drift), moving average and autoregressive processes.
- ... define autocorrelation and autocovariance and to compute these quantities for simple stochastic processes.
- ... estimate autocorrelation and autocovariance numerically from a given time series.
- ... give a precise definition of weak stationarity and at least two heuristics how a given time series can be tested on (weak) stationarity.

## Computational Objectives

You should be able to ...

- ... estimate the autocorrelation and autocovariance for a given time series by means of the `statsmodels.tsa.stattools.acf()` and `.acovf()` functions.

# Chapter 13.

## Forecasting Time Series<sup>1</sup>

In Chapter 11 we have studied various examples of time series ranging from econometrics over engineering to environmental sciences. Further we have introduced various visualization and summarization techniques as well as time series decomposition. In Chapter 12 we have learned the probabilistic framework to model time series: (discrete) stochastic processes. Autocovariance and -correlation have been introduced as informative quantities for a given process and we have learned how to estimate them from data if the underlying process is (weakly) stationary. Now we will finally study how modeling given time series data with a proper stochastic process can be used to *forecast* future and unobserved values of the process.

Put differently, the goal of this chapter is to predict future values  $x_{n+k}$  with  $k = 1, 2, \dots$  given a time series up to the present time  $\{x_1, \dots, x_n\}$ . Essentially, there are three steps that have to be carried out subsequently, in order to achieve this goal:

1. We need to be certain that the underlying process is predictable, i.e. in the future the process will not change dramatically but continues as it has up to the present (in a probabilistic sense).
2. We choose a model class by explorative data analysis of the given time series. After that we fit the model to the training data and receive model parameters that fully describe the fitted model.
3. With the fitted model we predict future values of the process.

We will focus on the most basic and yet important parametric models for stationary processes: autoregressive models. We will not dwell on the precise mathematical theory of estimation of the model parameters, but will focus on the discussion of the prediction results. Finally, we will give an outlook on how to forecast time series that are not stationary.

---

<sup>1</sup>This chapter follows Sections 5.3, 8.1 and 8.2 in the lecture notes *Applied Time Series Analysis* by Marcel Dettling, ETH Zurich, 2016.

## 13.1. Autoregressive Models

### 13.1.1. Definition and the Characteristic Polynomial

Autoregressive models are based on the idea that the current value of the series can be explained by a linear combination of the  $p$  previous values. To be more precise

#### Autoregressive model AR( $p$ )

The autoregressive model of order  $p$  is a discrete stochastic process that satisfies

$$X_n = a_1 X_{n-1} + a_2 X_{n-2} + \cdots + a_p X_{n-p} + W_n$$

where  $a_1, a_2, \dots, a_n$  are the model parameters and  $W_1, W_2, \dots$  is a white noise process with variance  $\sigma^2$ .

For a given autoregressive process, the *characteristic polynomial* is defined as

$$\Phi(x) = 1 - a_1 x - a_2 x^2 - \cdots - a_p x^p.$$

Autoregressive models are exclusively used for the modeling of stationary processes. However, the above definition does not guarantee stationarity. There may be combinations of model parameters that lead to non-stationary processes. We will give a sufficient condition on the parameters that render the process stationary further below. For the time being we study an important special case: the AR(1) process.

#### Example 13.1.1

The AR(1) process is defined by

$$X_n = a_1 X_{n-1} + W_n.$$

That means that the random walk is a special case of an AR(1) with  $a_1 = 1$  (see Example 12.1.1). We first compute the expected value of the process by taking expectations on both sides of the equation:

$$\mu = E[X_n] = a_1 E[X_{n-1}] + E[W_n] = a_1 \mu + 0$$

Thus we find that  $(a_1 - 1)\mu = 0$  which implies that  $\mu = 0$ , if  $a_1 \neq 1$ . I.e. if the process is stationary, then the mean function is  $\mu(i) = 0$ .

Next we compute the variance of the process. Proceeding as above with variances gives

$$\sigma_X^2 = \text{Var}[X_n] = a_1^2 \text{Var}[X_{n-1}] + \text{Var}[W_n] = a_1^2 \sigma_X^2 + \sigma^2$$

So we find that  $\sigma_X^2 = a_1^2 \sigma_X^2 + \sigma^2$  or, after some rearrangements:

$$\sigma_X^2 = \frac{\sigma^2}{1 - a_1^2}$$

In order to have a reasonable value for a constant variance  $\sigma_X^2$  (a minimal property for stationarity) the absolute value  $|a_1|$  must be less than 1 (otherwise the variance would be negative!). This can be interpreted as follows: In order to be stationary, the dependence of the process on past values should not be too strong. ◀

The general condition for an AR(p) to be weakly stationary can be formulated by means of the characteristic polynomial

### Stationarity of AR(p)

An AR(p) stochastic process is weakly stationary, if all (complex) roots of the characteristic polynomial exceed 1 in absolute value.

This is a very practical assertion that can easily be used to check whether a given autoregressive process is stationary or not.

### Example 13.1.2

In Example 13.1.1 the characteristic polynomial is given as

$$\Phi(x) = 1 - a_1 x$$

The single zero crossing of the polynomial  $\Phi$  is  $x = 1/a_1$  which exceeds 1 in absolute value if and only if  $|a_1| < 1$ . ◀

For models with  $p > 1$  the computation of the roots can be cumbersome. The function `numpy.roots()` computes the complex roots of a given polynomial.

### Example 13.1.3

We study the AR(3) process

$$X_n = 0.5X_{n-1} - 0.5X_{n-2} - 0.1X_{n-3} + W_n$$

We compute the absolute values of the roots of the characteristic polynomial  $\Phi(x) = 1 - 0.5x + 0.5x^2 + 0.1x^3$  with the following command ([to R](#))

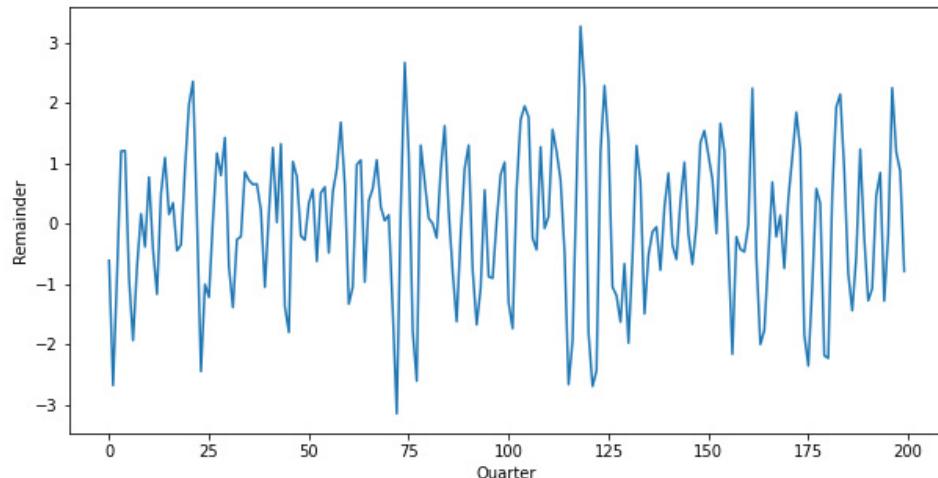
## Chapter 13. Forecasting Time Series

```
[1]: import numpy as np  
  
# note order: p[0] * x**n + p[1] * x**(n-1) + ... + p[n-1]*x + p[n]  
abs_roots = abs(np.roots([0.1, 0.5, -0.5, 1]))  
print(abs_roots)
```

```
[6.09052833 1.28136399 1.28136399]
```

We can see that all the values are larger than 1 and hence the process is stationary. We can simulate a time series from this model with the `arima_process.ArmaProcess()` method from `statsmodels.tsa`. (to R)

```
[2]: import matplotlib.pyplot as plt  
from statsmodels.tsa.arima_process import ArmaProcess  
  
# Simulate time series using ArmaProcess  
ar3 = [1, -0.5, 0.5, 0.1]  
simulated_data = ArmaProcess(ar3, ma=[1])  
simulated_data = simulated_data.generate_sample(nsample=200)  
  
# plot  
fig = plt.figure(figsize=(10, 5))  
ax = fig.add_subplot(1, 1, 1)  
ax.plot(simulated_data)  
plt.xlabel("Quarter")  
plt.ylabel("Remainder")  
plt.title("AR(3) process")  
plt.show()
```



**Figure 13.1.:** Time series generated by the AR(3) model.

Figure 13.1 shows an AR(3) realization of the code above. ◀

### 13.1.2. Autocorrelation of AR(p) processes

If we intend to fit an autoregressive model to a given time series data set, two things have to be clarified in advance

1. Is the autoregressive model the right choice for the given data?
2. What is the proper model order  $p$  for the given data?

A good guideline for answering these two questions is the autocorrelation of the process. In this chapter, we will hence study the theoretical shape of the autocorrelation of AR(p) processes. In practice, the sample autocorrelation of a given series is computed and compared with the theoretical ones. In this way we can judge if the time series is autoregressive and estimate the model order.

It will turn out that the autocorrelation is not sufficient to answer this question, such that we will introduce the *partial autocorrelation* as a further measure well suited for autoregressive processes. We begin with the computation of the autocorrelation of an AR(1) process

#### Example 13.1.4

We consider the AR(1) process in Example 13.1.1 with  $|a_1| < 1$ . We compute the autocorrelation at lag  $h = 1$

$$\begin{aligned}\rho(1) &= \frac{\text{Cov}[X_n, X_{n-1}]}{\gamma(0)} = \frac{\text{Cov}[a_1 X_{n-1} + W_n, X_{n-1}]}{\gamma(0)} \\ &= \frac{a_1 \text{Cov}[X_{n-1}, X_{n-1}] + \text{Cov}[W_n, X_{n-1}]}{\gamma(0)} = \frac{a_1 \gamma(0)}{\gamma(0)} = a_1\end{aligned}$$

The autocorrelation at lag  $h = 2$  can now be computed by

$$\rho(2) = \frac{\text{Cov}[X_n, X_{n-2}]}{\gamma(0)} = \frac{a_1 \text{Cov}[X_{n-1}, X_{n-2}] + \text{Cov}[W_{n-1}, X_{n-2}]}{\gamma(0)} = a_1 \rho(1) = a_1^2$$

Continuing in this way results in the formula

$$\rho(h) = a_1^h.$$

This amounts to say, that the autocorrelation of the AR(1) process decays exponentially fast. ◀

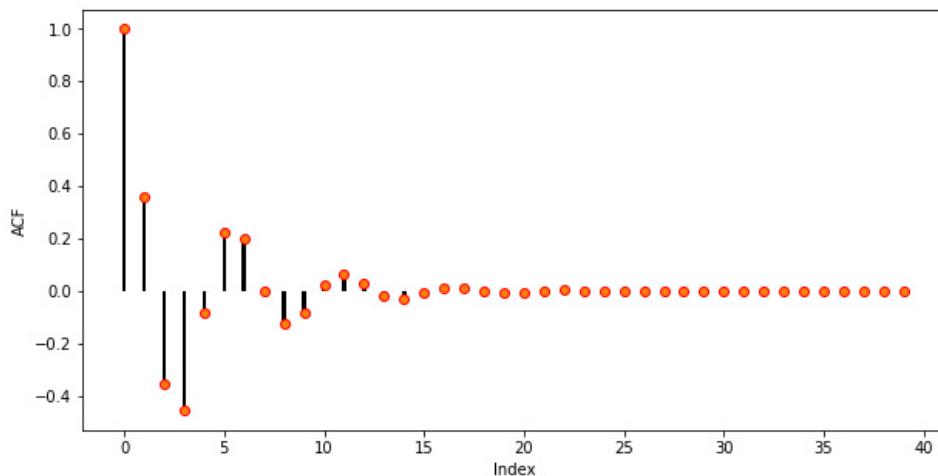
**Example 13.1.5**

We study the AR(3) process in Example 13.1.3. The process is defined by the coefficients  $a_1 = 0.5$ ,  $a_2 = -0.5$  and  $a_3 = -0.1$ . We use `ArmaProcess()` to define the stochastic process, and `.acf()` to compute the (theoretical) autocorrelation function. (to R)

```
[3]: # Compute the theoretical autocorrelation function
lag = 40
acf_theor = ArmaProcess(ar = ar3)
acf_theor = acf_theor.acf(lag)

# Plot
x = np.arange(lag)
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)
plt.bar(x, acf_theor, width=0.2, color="black")
plt.plot(x, acf_theor, "ro", markerfacecolor="C1")
plt.xlabel("Index")
plt.ylabel("ACF")
plt.title("ACF of an AR(3) Process")

plt.show()
```



**Figure 13.2.:** The theoretical autocorrelation of an AR(3) process.

As it can be seen in Figure 13.2 the (theoretical) autocorrelation of the given AR(3) is oscillating and decreasing essentially following an exponential function. This is the typical autocorrelation behaviour of an autoregressive process. ◀

The above examples indicate that the (theoretical) autocorrelation of an AR( $p$ ) process is nonzero for a wide span of lags. This is due to propagation of correlation through the model: If  $X_k$  is strongly correlated to  $X_{k+1}$  and in turn  $X_{k+1}$  is strongly correlated

to  $X_{k+2}$ , then  $X_k$  will likely be strongly correlated to  $X_{k+2}$ . If we want to study the *direct* correlation of  $X_k$  and  $X_{k+2}$ , i.e. the proportion of correlation that is *not* due to  $X_{k+1}$ , we have to compute the partial autocorrelation. The precise mathematical definition is as follows

### Partial autocorrelation

For a weakly stationary stochastic process  $\{X_1, X_2, \dots\}$  the partial autocorrelation is defined as

$$\pi(h) = \text{Cor}[X_k, X_{k+h} | X_{k+1}, \dots, X_{k+h-1}]^a.$$

---

<sup>a</sup>The quantity  $\text{Cor}[X, Y | Z]$  denotes the conditional correlation of  $X$  and  $Y$  given the value of  $Z$

The precise mathematical definition above is somewhat technical and we will not embark further on theoretical aspects, exact computation, etc. We mention, however, that for an autoregressive process AR(p) the partial autocorrelation has two very important properties

1. The  $p$ -th coefficient  $a_p$  of an AR( $p$ ) process equals  $\pi(p)$ , i.e. the partial autocorrelation value at lag  $p$  of the process.
2. For an autoregressive process AR( $p$ ) the partial autocorrelation at lags greater than  $p$  is zero, i.e.  $\pi(k) = 0$  for  $k > p$ .

With these properties we have a tool at hand to investigate a given autoregressive time series and determine the order of the model: Compute the partial autocorrelation and choose the largest lag  $k$  for which the value  $\pi(k)$  is not zero. In [Python](#) the partial autocorrelation of a stationary time series is estimated with the `pacf()` function.

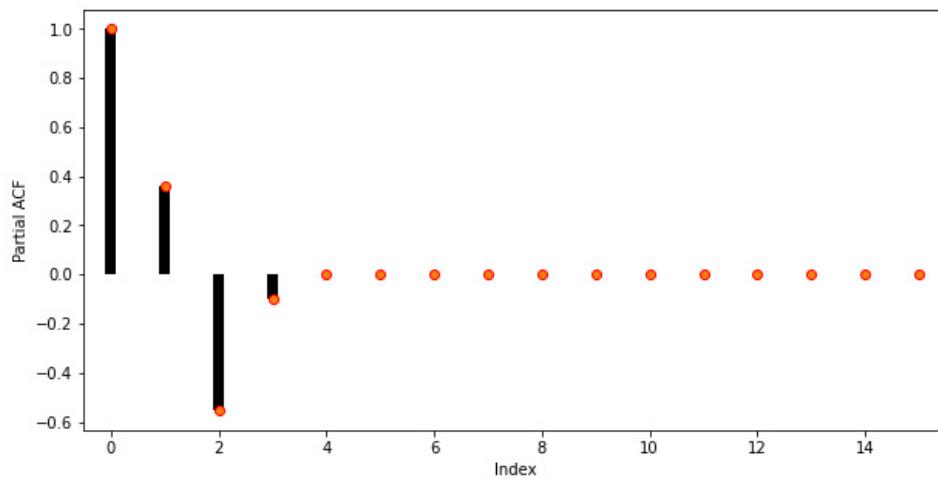
### Example 13.1.6

We consider the AR(3) process in Example 13.1.3. ([to R](#))

```
[4]: # Compute the theoretical partial autocorrelation function
pacf_theor = ArmaProcess(ar=ar3)
pacf_theor = pacf_theor.pacf(lag)

# Plot
x = np.arange(lag)
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)
ax.bar(x, pacf_theor, width=0.2, color="black")
ax.plot(x, pacf_theor, "ro", markerfacecolor="C1")
ax.set_xlim([-0.5, 15.5])
```

```
plt.xlabel("Index")
plt.ylabel("Partial ACF")
plt.title("Partial autocorrelation of AR(3) Processes")
plt.show()
```



**Figure 13.3.:** The theoretical partial autocorrelation function of an AR(3) process.

As it can be seen in Figure 13.3, the partial autocorrelation coefficients larger than 3 are zero. So in practice, i.e. when only the time series is at hand, we would choose an autoregressive model of order 3 for modelling the present sequence. ◀

Before we proceed we introduce a real world example that is well known in the time series literature for modelling autoregressive time series.

### Example 13.1.7

Forecasting solar activity is important for satellite drag, telecommunication outages and solar winds in connection with blackout of power plants. As an indicator of solar activity the sunspot number is used, among others. The Swiss astronomer Johann Rudolph Wolf introduced the sunspot number in 1848 and the number of sunspots is now known on a monthly basis back to the year 1749 (from 1749 to 1848 the data is questionable). The data set is built-in in [Python](#) and the following code produces the plot in Figure 13.4: ([to R](#))

```
[1]: import statsmodels.api as sm
import pandas as pd
import matplotlib.pyplot as plt

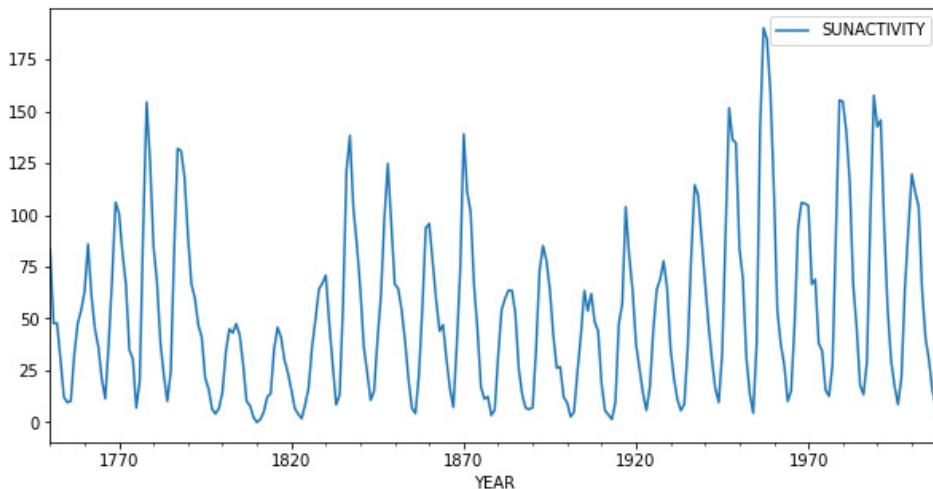
# Load data
dta = sm.datasets.sunspots.load_pandas().data

# Create pandas DateTimeIndex
```

## Chapter 13. Forecasting Time Series

```
dtindex = pd.DatetimeIndex(data=pd.to_datetime(dta["YEAR"], format="%Y"),
                            freq='infer')
# Set as Index
dta.set_index(dtindex, inplace=True)
dta.drop("YEAR", axis=1, inplace=True)
# Select only data after 1750
dta = dta["1750":"2008"]

# Plot
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)
dta.plot(ax=ax)
plt.show()
```



**Figure 13.4.:** The yearly averaged number of sunspot since 1749

It is important to note that the **Sunspot** data set is *not periodic*, i.e. the cycle duration is not constant. So the quasi periodic behaviour must not be mistaken for seasonality. The peaks and minima of the series are not known in advance.

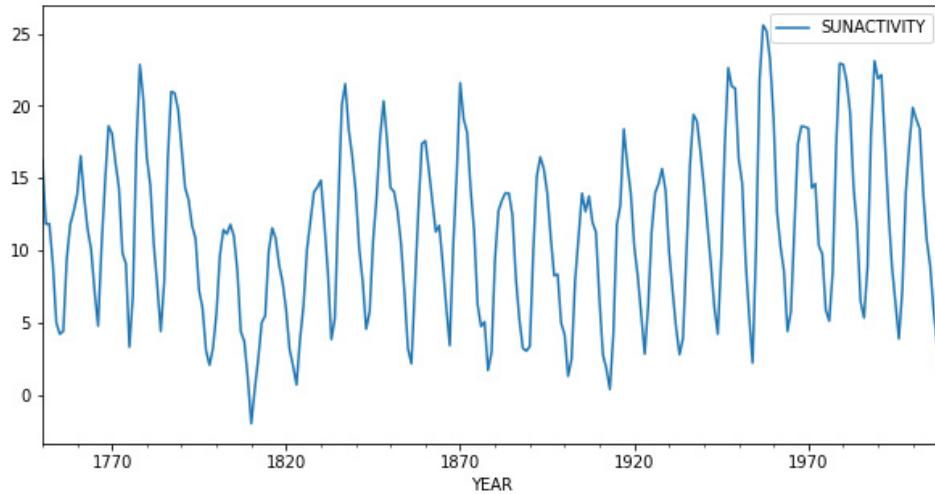
Eyeballing indicates that the underlying process is not stationary: There are clearly phases with different variances and means. We perform a Box-Cox square-root transform to stabilize the variance. ([to R](#))

```
[2]: import numpy as np

# Box-Cox square root transformation
dta_sq = (np.sqrt(dta) - 1) * 2

# Plot
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)
dta_sq.plot(ax=ax)
```

```
plt.show()
```



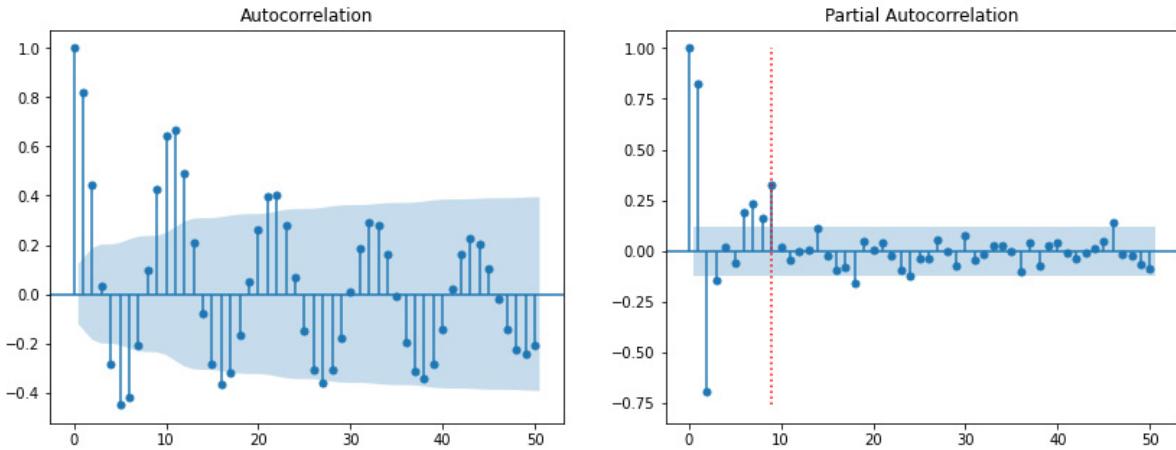
**Figure 13.5.:** Square-root transformed of the yearly averaged `Sunspot` time series data set.

Figure 13.5 shows the square-root transformed sequence. The variance is visually stabilized and although there is still some trend visible, the series looks fairly stationary after the transformation. Next, we compute the autocorrelation and partial autocorrelation functions in order to clarify whether the autoregressive model is the right choice and in case to determine the proper model order. [\(to R\)](#)

```
[1]: from statsmodels.graphics.tsaplots import plot_pacf
from statsmodels.graphics.tsaplots import plot_acf

# Plot
fig = plt.figure(figsize=(14, 5))
ax1 = fig.add_subplot(1, 2, 1)
plot_acf(dta_sq, lags=50, ax=ax1)
ax2 = fig.add_subplot(1, 2, 2)
plot_pacf(dta_sq, lags=50, ax=ax2)
ax2.plot([9, 9], [-0.76, 1], ':r')
plt.show()
```

In Figure 13.6 the autocorrelation (left) and partial autocorrelation (right) are depicted. As it can be seen, the autocorrelation shows the typical behaviour of an autoregressive process: an oscillating pattern with an exponential decay. The partial autocorrelation shows that the direct dependency has a maximum lag of 9 which we will use as our model parameter  $p$  in the next section. The red dotted line marks this threshold. ◀



**Figure 13.6.:** Autocorrelation (left) and partial autocorrelation (right) of the square-root transformed `Sunspots` number data.

### 13.1.3. Model fitting

In this chapter we will focus on the task of estimation of an AR( $p$ ) model from given time series data. So we will assume that we are given a time series  $\{x_1, x_2, \dots, x_n\}$  that is a realization of a stationary stochastic process. The process of fitting an autoregressive model is as follows:

1. Decide whether the given time series stems from an autoregressive process. This is done by investigating the autocorrelation and partial autocorrelation of the sequence. If the autocorrelation is decaying exponentially to zero (possibly with oscillations) and the partial autocorrelation is zero for larger lags, then the autoregressive model is appropriate.
2. Choose the largest lag  $p$  such that the partial autocorrelation is not zero. This is usually a fair choice for model complexity.
3. Choose parameters  $a_1, \dots, a_p$  such that the given data are likely to be realizations of the corresponding autoregressive process. Check whether the process is stationary by computing the roots of the characteristic polynom.

The first two items are done by visual inspection of the (partial) correlogram<sup>2</sup>. We will now focus on the estimation part. We do not go into detail, but will just mention the most basic approach to estimate the coefficients: ordinary least squares.

Given the data  $\{x_1, x_2, \dots, x_n\}$  and a model order  $p$ , we fit the AR( $p$ ) process to the

---

<sup>2</sup>There are of course more involved methods of model selection such as Akaike's information criterion (AIC) etc.

data by solving the following linear equation system

$$\begin{aligned}x_{p+1} &= a_1 x_p + a_2 x_{p-1} + \cdots + a_p x_1 + W_{p+1} \\x_{p+2} &= a_1 x_{p+1} + a_2 x_p + \cdots + a_p x_2 + W_{p+2} \\&\vdots \\x_n &= a_1 x_{n-1} + a_2 x_{n-2} + \cdots + a_p x_{n-p} + W_n\end{aligned}$$

The system is solved in the least squares sense and it is assumed that the mean  $\hat{x} = (\sum_i x_i)/n$  has been subtracted from the time series in advance. The solution of these equations are the estimates  $\hat{a}_1, \dots, \hat{a}_p$ . There are at least three further methods that are standard for estimating the AR(p) coefficients; the estimates are in general different but asymptotically (i.e. for large  $n$ ) identical.

- *Burg's algorithm*: The least squares approach above has the drawback that the first  $p$  values are never used as responses. Burg's method remedies this shortcoming and exploits the fact that an AR(p) is also an AR(p) when viewed backward in time.
- *Yule-Walker equations*: This method transforms the above equations such that the autocorrelations instead of the time series itself appear in the equation. Then the sample ACF is plugged in and the system is solved.
- *Maximum Likelihood Method*: It maximizes the likelihood to observe the given data given the process as parameters  $a_1, \dots, a_p$ . For computing the likelihood function it is assumed that the white noise process  $W_k$  in fact is Gaussian.

In [Python](#) all three methods are implemented, and can be called set in the generic `.fit()` function through `method= "yule_walker", "burg", etc..` We study the performance of the function by means of an example:

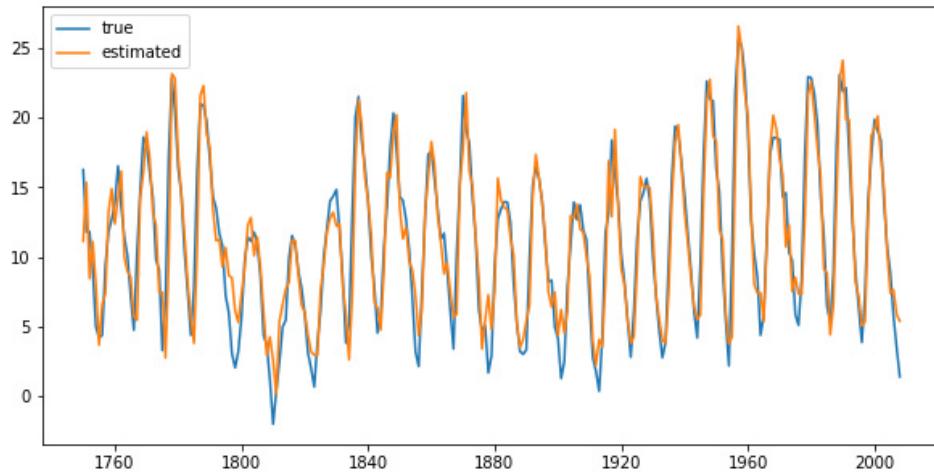
### Example 13.1.8

We have a look at the [Sunspots](#) data set in Example 13.1.7. We have transformed the data by a square-root and have concluded that it stems from an AR(9) model. We will now fit this model to the data ([to R](#))

```
[4] : from statsmodels.tsa.arima.model import ARIMA
model = ARIMA(dta_sq, order=(9, 0, 0)).fit(method="yule_walker")
# Plot
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)
ax.plot(dta_sq, label='true')
ax.plot(dta_sq["SUNACTIVITY"] - model.resid, label='estimated')
plt.legend()
```

## Chapter 13. Forecasting Time Series

```
plt.show()
```



**Figure 13.7.:** The annually averaged sunspot number data (blue) and the fitted AR(9) model (orange).

```
[5]: print(model.summary())
```

```
SARIMAX Results
=====
Dep. Variable: SUNACTIVITY   No. Observations: 259
Model: ARIMA(9, 0, 0)   Log Likelihood: -563.906
Date: Mon, 07 Jun 2021   AIC: 1149.811
Time: 13:15:55           BIC: 1188.936
Sample: 01-01-1750       HQIC: 1165.542
                           - 01-01-2008
Covariance Type: opg
=====
            coef    std err        z      P>|z|      [0.025      0.975]
-----
const    11.1432    1.110    10.039      0.000     8.968    13.319
ar.L1     1.2123    0.055    21.935      0.000     1.104    1.321
ar.L2    -0.4895    0.079    -6.226      0.000    -0.644   -0.335
ar.L3    -0.1466    0.089    -1.650      0.099    -0.321   0.028
ar.L4     0.2684    0.108     2.495      0.013     0.058    0.479
ar.L5    -0.2541    0.111    -2.280      0.023    -0.472   -0.036
ar.L6     0.0516    0.114     0.452      0.651    -0.172    0.275
ar.L7     0.1352    0.112     1.212      0.226    -0.083    0.354
ar.L8    -0.2179    0.098    -2.218      0.027    -0.411   -0.025
ar.L9     0.3080    0.059     5.210      0.000     0.192    0.424
sigma2    4.4825    0.376    11.919      0.000     3.745    5.220
=====
Ljung-Box (L1) (Q):      0.00  Jarque-Bera (JB):      8.69
Prob(Q):                  0.99  Prob(JB):                 0.01
Heteroskedasticity (H):    0.70  Skew:                   0.36
Prob(H) (two-sided):      0.11  Kurtosis:                3.54
=====
Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

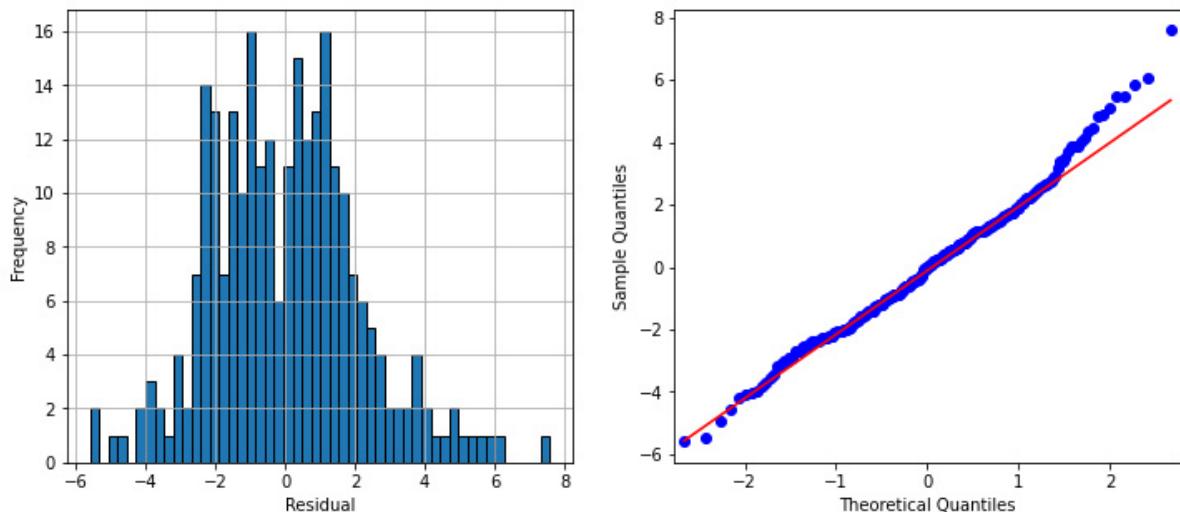
The output above shows the 9 estimated coefficients and the estimated variance of the white noise process.

Figure 13.7 shows the annually averaged time series and the output of the model. The fit seems to be reasonably accurate but we can further assess the result by examining residual plots. We choose a histogram and a qq-plot. ([to R](#))

```
[6]: from statsmodels.graphics.api import qqplot
```

```
# Plot
fig = plt.figure(figsize=(12, 5))
ax1 = fig.add_subplot(1, 2, 1)
model.resid.hist(edgecolor="black", bins=50, ax=ax1)
plt.xlabel("Residual")
plt.ylabel("Frequency")
ax2 = fig.add_subplot(1, 2, 2)
qqplot(model.resid, line="q", ax=ax2)

plt.show()
```



**Figure 13.8.: Residual analysis for the fitted model:** The histogram (left) and the qq-plot (right) look fairly normal.

### 13.1.4. Forecasting AR(p) processes

Eventually we now aim at forecasting future values of a given autoregressive time series. The general methodology of forecasting stationary time series can be summarized as follows

***k*-step ahead forecast**

Assume that  $\{X_1, X_2, \dots\}$  is a stationary process and that we have observed a time series  $\{x_1, x_2, \dots, x_n\}$ . The  $k$ -step ahead forecast is an estimate of the random variable  $X_{n+k}$  given by

$$\hat{X}_{n+k} = E[X_{n+k} | X_1 = x_1, \dots, X_n = x_n].$$

Here  $E[X|Y = y]$  denotes the conditional expectation of  $X$  given that  $Y = y$  (cf. Section 3.1).

The above definition requires a model for the time series  $\{X_1, X_2, \dots\}$  in order to compute the conditional expectation. We will study the computation of this conditional expectation for autoregressive processes and start with a general AR(1).

**Example 13.1.9**

We consider an autoregressive process of order 1 with a coefficient  $a_1$  satisfying  $|a_1| < 1$ , i.e. the process is stationary:

$$X_k = a_1 X_{k-1} + W_k.$$

If data  $\{x_1, x_2, \dots, x_n\}$  is given, it is easy to estimate the value of  $X_{n+1}$ :

$$\hat{X}_{n+1} = E[X_{n+1} | X_1 = x_1, \dots, X_n = x_n] = a_1 x_n.$$

For  $k > 1$  we plug-in the model equation several times and obtain

$$\begin{aligned}\hat{X}_{n+k} &= E[X_{n+k} | X_1 = x_1, \dots, X_n = x_n] \\ &= E[a_1 X_{n+k-1} + W_{n+k} | X_1 = x_1, \dots, X_n = x_n] \\ &= a_1 E[X_{n+k-1} | X_1 = x_1, \dots, X_n = x_n] \\ &= \dots \\ &= a_1^k x_n.\end{aligned}$$

This amounts to say that the  $k$ -step ahead forecast of an AR(1) process is exponentially decaying to zero from the last observation  $x_n$  and does not depend on earlier observations. ◀

In [Python](#) the forecasting of a time series can be done by the generic [prediction](#) function. In order to obtain more detailed results, the function [get\\_prediction](#) should be used. This function can return the standard error ([se](#)) in addition. The standard error  $\sigma_k$  given here is the square root of the conditional variance

$$\sigma_k^2 = \text{Var}[X_{n+k} | X_1 = x_1, \dots, X_n = x_n].$$

This quantity is increasing with  $k$  and converges to the process variance  $\sigma_X^2$ . With this standard error a 95% confidence interval for the conditional expectation  $E[X_{n+k}|X_1 = x_1, \dots, X_n = x_n]$  can be computed

$$\hat{X}_{n+k} \pm 1.96\sigma_k.$$

Note that in practice the standard error as well as the coefficients of the process are estimated and that the above formula does not account for the uncertainties in these estimates. In other words: the real coverage of the confidence interval may be different from the nominal one (here 95%) if the number of samples  $n$  is small. This has to be kept in mind when predicting time series.

### Example 13.1.10

In the following we simulate data from an AR(1) process and predict some future values given these data. We build the model on a subset of the generated data, i.e. we cut off some of the values at the end. We will predict these values and study the residuals. [\(to R\)](#)

```
[1]: import pandas as pd
import numpy as np
from statsmodels.tsa.arima_process import ArmaProcess
from statsmodels.tsa.arima.model import ARIMA
import matplotlib.pyplot as plt

# Simulate AR1 Data
np.random.seed(8)
ar1 = [1, -0.7]

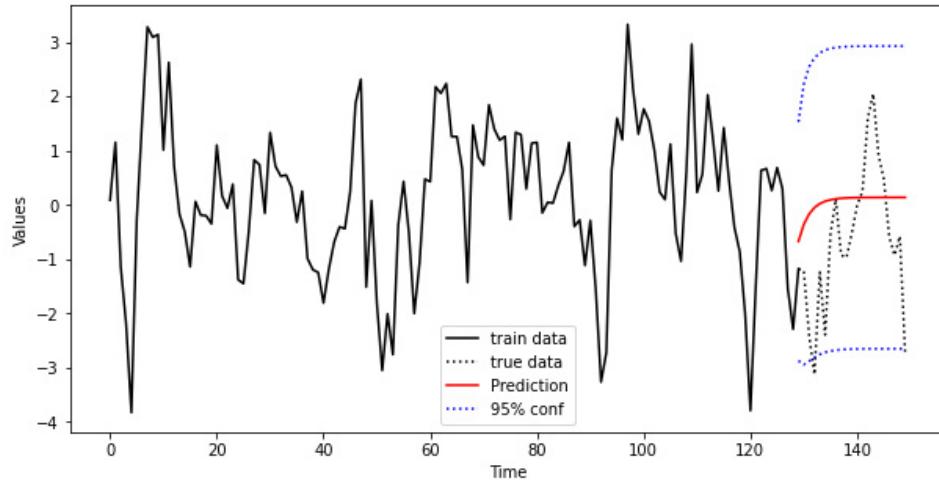
simulated_data = ArmaProcess(ar1)
simulated_data = simulated_data.generate_sample(nsample=150)

# Fit model on first 130 points
model = ARIMA(simulated_data[0:130], order=(1, 0, 0)).fit(method="yule_walker")

# Predict last 20 points
pred = model.get_prediction(start=130, end=150).prediction_results
pred_cov = pred._forecasts_error_cov
pred = pred._forecasts[0]
pred_upper = pred + 1.96 * np.sqrt(pred_cov[0][0])
pred_lower = pred - 1.96 * np.sqrt(pred_cov[0][0])

# Plot
x = np.arange(151)
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)
ax.plot(x[0:130], simulated_data[0:130], '-k', label='train data')
ax.plot(x[129:150], simulated_data[129:150], ':k', label='true data')
ax.plot(x[129:150], pred, 'r', label='Prediction')
ax.plot(x[129:150], pred_upper, ':b', label='95% conf')
```

```
ax.plot(x[129:150], pred_lower, ':b')
plt.xlabel("Time")
plt.ylabel("Values")
plt.legend()
plt.show()
```



**Figure 13.9.:** A simulated AR(1) process with 150 observations. Starting from obervation 131 a prediction is computeed (red).

Finally, we add some confidence bands to the plot. To this end, we use the standard error that is stored in `._forecasts_error_cov`

Figure 13.9 shows the complete data (dotted black) and the training set which consists of the the first 130 observations (solid black). From these data an AR(1) model is built and the values for the remaining 20 observations are predicted (solid red) including confidence limits (dashed red). We see that the true values of the process are within the confidence limits. ◀

The example above indicates that the AR(1) predictions are very crude: They start at the last observation and decay exponentially to zero. This is not surprising since all future predictions only depend on the last observed value. Predictions from higher order AR( $p$ ) models are less trivial as the following example shows.

### Example 13.1.11

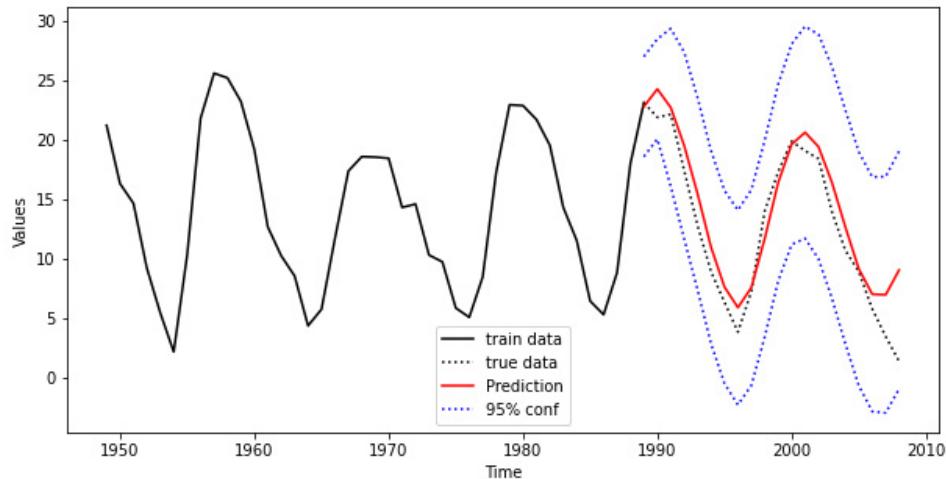
We again focus on the sunspot number data in Example 13.1.7. We use the annual data from 1749 until 1989 as a training set and estimate an AR(9) model from that data. We then predict the sunspot number for 25 years into the future. (to R)

```
[7]: # Fit model on first 130 points
model = ARIMA(dta_sq["1749": "1989"], order=(9, 0, 0)).fit(method="yule_walker")
```

## Chapter 13. Forecasting Time Series

```
# Predict including confidence interval
pred = model.get_prediction(start="1989", end="2008").prediction_results
pred_cov = pred._forecasts_error_cov
pred = pred._forecasts[0]
pred_upper = pred + 1.96 * np.sqrt(pred_cov[0][0])
pred_lower = pred - 1.96 * np.sqrt(pred_cov[0][0])

# Plot
x = dta_sq.index.year
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)
ax.plot(dta_sq["1949": "1989"], '-k', label='train data')
ax.plot(dta_sq["1989": "2009"], ':k', label='true data')
ax.plot(dta_sq["1989": "2009"].index, pred, 'r', label='Prediction')
ax.plot(dta_sq["1989": "2009"].index, pred_upper, ':b', label='95% conf')
ax.plot(dta_sq["1989": "2009"].index, pred_lower, ':b')
plt.xlabel("Time")
plt.ylabel("Values")
plt.legend()
plt.show()
```



**Figure 13.10.:** Sunspot numbers from 1950 to 2014. The training set (solid black) and the prediction (red) are shown. The true future data is depicted in a dashed black line.

Figure 13.10 shows a window of the data set from 1950 until 2014. The training set is depicted by a black solid line and ranges up to 1989. The prediction is computed and plotted for the time period 1990 to 2012 together with the true data (black dotted) that has been discarded in the modelling step. As it can be seen, there is good correspondence between true and estimated values for about 15 years. Then the prediction starts to diverge and the second minimum phase is not predicted correctly. ◀

## Conceptual Objectives

You should be able to ...

- ... give a precise definition of the AR( $p$ ) model.
- ... compute the characteristic polynomial of a given AR( $p$ ) model.
- ... check whether an AR( $p$ ) is weakly stationary by means of the roots of the characteristic polynomial.
- ... provide and to explain the definition of partial autocorrelation.
- ... identify an AR( $p$ ) process by means of the autocorrelation and partial autocorrelation.
- ... explain at least one method to estimate an AR( $p$ ) process from a given time series.
- ... provide the definition of the  $k$ -step ahead forecast of time series

## Computational Objectives

You should be able to ...

- ... use the `statsmodels.tsa.arima_process.ArmaProcess()`-function in order to simulate MA( $q$ ) and AR( $p$ ) processes.
- ... check weak stationarity of an AR( $p$ ) process by means of the `numpy.roots()`-function.
- ... compute the exact autocorrelation of moving average and autoregressive processes with the `.acf()`-function.
- ... compute forecasts of AR( $p$ ) including confidence bands.

# Chapter 14.

## Forecasting Time Series<sup>1</sup>

### 14.0.1. Repetition of Autoregressive Models

*Autoregressive models* are based on the idea that the current value of the series,  $x_t$ , can be explained as a function of  $p$  past values,  $x_{t-1}, x_{t-2}, \dots, x_{t-p}$ , where  $p$  determines the number of steps into the past needed to forecast the current value. As a typical case, recall Example 1.10 in which data were generated using the model

$$x_t = x_{t-1} - 0.90x_{t-2} + w_t, \quad (14.1)$$

where  $w_t$  is white Gaussian noise with  $\sigma_w^2 = 1$ . We have now assumed the current value is a particular linear function of past values. The regularity that persists in Fig. 1.9 gives an indication that forecasting for such a model might be a distinct possibility, say, through some version such as

$$x_{n+1} = x_n - 0.90x_{n-1}, \quad (14.2)$$

where the quantity on the left-hand side denotes the forecast at the next period  $n + 1$  based on the observed data,  $x_1, x_2, \dots, x_n$ . We will make this notion more precise in our discussion of forecasting (Sect. 3.4).

The extent to which it might be possible to forecast a real data series from its own past values can be assessed by looking at the autocorrelation function and the lagged scatterplot matrices discussed in Chap. 2. For example, the lagged scatterplot matrix for the Southern Oscillation Index (SOI), shown in Fig. 2.8, gives a distinct indication that lags 1 and 2, for example, are linearly associated with the current value. The ACF shown in Fig. 1.16 shows relatively large positive values at lags 1, 2, 12, 24, and 36 and large negative values at 18, 30, and 42. We note also the possible relation between the SOI and Recruitment series indicated in the scatterplot matrix shown in Fig. 2.9. We will indicate in later sections on transfer function and vector AR modeling how to handle the dependence on values taken by other series.

The preceding discussion motivates the following definition.

---

<sup>1</sup>This chapter follows Sections 5.3, 8.1 and 8.2 in the lecture notes *Applied Time Series Analysis* by Marcel Dettling, ETH Zurich, 2016.

## Conceptual Objectives

You should be able to ...

- ... give a precise definition of the  $\text{AR}(p)$  model.
- ... compute the characteristic polynomial of a given  $\text{AR}(p)$  model.
- ... check whether an  $\text{AR}(p)$  is weakly stationary by means of the roots of the characteristic polynomial.
- ... provide and to explain the definition of partial autocorrelation.
- ... identify an  $\text{AR}(p)$  process by means of the autocorrelation and partial autocorrelation.
- ... explain at least one method to estimate an  $\text{AR}(p)$  process from a given time series.
- ... provide the definition of the  $k$ -step ahead forecast of time series

## Computational Objectives

You should be able to ...

- ... use the `statsmodels.tsa.arima_process.ArmaProcess()`-function in order to simulate  $\text{MA}(q)$  and  $\text{AR}(p)$  processes.
- ... check weak stationarity of an  $\text{AR}(p)$  process by means of the `numpy.roots()`-function.
- ... compute the exact autocorrelation of moving average and autoregressive processes with the `.acf()`-function.
- ... compute forecasts of  $\text{AR}(p)$  including confidence bands.

# Chapter 15.

## Spectral Analysis

### 15.1. Introduction

### 15.2. Supplementary Note on DFT

With Euler's formula

$$e^{i\alpha} = \cos(\alpha) + i \sin(\alpha) \quad \text{with } i \text{ the imaginary unit } (i^2 = -1)$$

one can express  $\cos(k\omega t)$  and  $\sin(k\omega t)$  as

$$\cos(k\omega t) = \frac{e^{ik\omega t} + e^{-ik\omega t}}{2} \quad \sin(k\omega t) = \frac{e^{ik\omega t} - e^{-ik\omega t}}{2i}$$

For  $j = 0, 1, 2, \dots, n - 1$

## Chapter 15. Spectral Analysis

$$\begin{aligned}
x_j &= \frac{a_0}{2} + a_1 \cos\left(1 \cdot \frac{2\pi}{n} \cdot j\right) + b_1 \sin\left(1 \cdot \frac{2\pi}{n} \cdot j\right) + a_2 \cos\left(2 \cdot \frac{2\pi}{n} \cdot j\right) + b_2 \sin\left(2 \cdot \frac{2\pi}{n} \cdot j\right) \\
&\quad + \dots + a_{\frac{n-1}{2}} \cos\left(\frac{n-1}{2} \cdot \frac{2\pi}{n} \cdot j\right) + b_{\frac{n-1}{2}} \sin\left(\frac{n-1}{2} \cdot \frac{2\pi}{n} \cdot j\right) \\
&= \frac{a_0}{2} + \sum_{k=1}^{\frac{n-1}{2}} \left( a_k \cos\left(k \cdot \frac{2\pi}{n} \cdot j\right) + b_k \sin\left(k \cdot \frac{2\pi}{n} \cdot j\right) \right) \\
&= \frac{a_0}{2} + \sum_{k=1}^{\frac{n-1}{2}} \left( \frac{a_k}{2} \left( e^{ik \cdot \frac{2\pi}{n} \cdot j} + e^{-ik \cdot \frac{2\pi}{n} \cdot j} \right) + \frac{b_k}{2i} \left( e^{ik \cdot \frac{2\pi}{n} \cdot j} - e^{-ik \cdot \frac{2\pi}{n} \cdot j} \right) \right) \\
&= \underbrace{\frac{a_0}{2}}_{c_0} + \sum_{k=1}^{\frac{n-1}{2}} \left( \underbrace{\frac{1}{2}(a_k - ib_k)}_{c_k} e^{ik \cdot \frac{2\pi}{n} \cdot j} + \underbrace{\frac{1}{2}(a_k + ib_k)}_{c_{-k}} e^{-ik \cdot \frac{2\pi}{n} \cdot j} \right) \\
&= c_0 + \sum_{k=1}^{\frac{n-1}{2}} \left( c_k e^{ik \frac{2\pi}{n} j} + c_{-k} e^{-ik \frac{2\pi}{n} j} \right) \\
&= \sum_{k=-\frac{n-1}{2}}^{\frac{n-1}{2}} c_k e^{ik \frac{2\pi}{n} j}
\end{aligned}$$

$$\begin{aligned}
x_0 &= c_0 + c_1 + c_2 + \dots + c_{\frac{n-1}{2}} + c_{-1} + c_{-2} + \dots + c_{-\frac{n-1}{2}} \\
x_1 &= c_0 + c_1 e^{i \cdot 1 \cdot \frac{2\pi}{n} \cdot 1} + c_2 e^{i \cdot 2 \cdot \frac{2\pi}{n} \cdot 1} + \dots + c_{\frac{n-1}{2}} e^{i \cdot \frac{n-1}{2} \cdot \frac{2\pi}{n} \cdot 1} + c_{-1} e^{i \cdot (-1) \cdot \frac{2\pi}{n} \cdot 1} + c_{-2} e^{i \cdot (-2) \cdot \frac{2\pi}{n} \cdot 1} + \dots + c_{-\frac{n-1}{2}} e^{i \cdot (-\frac{n-1}{2}) \cdot \frac{2\pi}{n} \cdot 1} \\
x_2 &= c_0 + c_1 e^{i \cdot 1 \cdot \frac{2\pi}{n} \cdot 2} + c_2 e^{i \cdot 2 \cdot \frac{2\pi}{n} \cdot 2} + \dots + c_{\frac{n-1}{2}} e^{i \cdot \frac{n-1}{2} \cdot \frac{2\pi}{n} \cdot 2} + c_{-1} e^{i \cdot (-1) \cdot \frac{2\pi}{n} \cdot 2} + c_{-2} e^{i \cdot (-2) \cdot \frac{2\pi}{n} \cdot 2} + \dots + c_{-\frac{n-1}{2}} e^{i \cdot (-\frac{n-1}{2}) \cdot \frac{2\pi}{n} \cdot 2} \\
\vdots &= \vdots \\
x_{n-1} &= c_0 + c_1 e^{i \cdot 1 \cdot \frac{2\pi}{n} \cdot (n-1)} + c_2 e^{i \cdot 2 \cdot \frac{2\pi}{n} \cdot (n-1)} + \dots + c_{\frac{n-1}{2}} e^{i \cdot \frac{n-1}{2} \cdot \frac{2\pi}{n} \cdot (n-1)} + c_{-1} e^{i \cdot (-1) \cdot \frac{2\pi}{n} \cdot (n-1)} + c_{-2} e^{i \cdot (-2) \cdot \frac{2\pi}{n} \cdot (n-1)} + \dots + c_{-\frac{n-1}{2}} e^{i \cdot (-\frac{n-1}{2}) \cdot \frac{2\pi}{n} \cdot (n-1)}
\end{aligned}$$

with  $\omega = e^{i \frac{2\pi}{n}}$  and rearranging the negatively indexed terms “ $c_{-k}$ ” (from smallest to largest)

$$\begin{aligned}
x_0 &= c_0 + c_1 + c_2 + \dots + c_{\frac{n-1}{2}} + c_{-\frac{n-1}{2}} + \dots + c_{-2} + c_{-1} \\
x_1 &= c_0 + c_1 \omega^{1 \cdot 1} + c_2 \omega^{2 \cdot 1} + \dots + c_{\frac{n-1}{2}} \omega^{\frac{n-1}{2} \cdot 1} + c_{-\frac{n-1}{2}} \omega^{-\frac{n-1}{2} \cdot 1} + \dots + c_{-2} \omega^{-2 \cdot 1} + c_{-1} \omega^{-1 \cdot 1} \\
x_2 &= c_0 + c_1 \omega^{1 \cdot 2} + c_2 \omega^{2 \cdot 2} + \dots + c_{\frac{n-1}{2}} \omega^{\frac{n-1}{2} \cdot 2} + c_{-\frac{n-1}{2}} \omega^{-\frac{n-1}{2} \cdot 2} + \dots + c_{-2} \omega^{-2 \cdot 2} + c_{-1} \omega^{-1 \cdot 2} \\
\vdots &= \vdots \\
x_{n-1} &= c_0 + c_1 \omega^{1 \cdot (n-1)} + c_2 \omega^{2 \cdot (n-1)} + \dots + c_{\frac{n-1}{2}} \omega^{\frac{n-1}{2} \cdot (n-1)} + c_{-\frac{n-1}{2}} \omega^{-\frac{n-1}{2} \cdot (n-1)} + \dots + c_{-2} \omega^{-2 \cdot (n-1)} + c_{-1} \omega^{-1 \cdot (n-1)}
\end{aligned}$$

For  $l \in \mathbb{Z}$ , one has:  $\omega^{ln} = (e^{i \frac{2\pi}{n}})^{ln} = e^{i \frac{2\pi}{n} \cdot ln} = e^{i 2l\pi} = 1$  and so ( $\omega^n = \omega^{2n} = \omega^{(n-1)n} = 1$ )

$$\begin{aligned}
\omega^{-1 \cdot 1} &= \omega^{-1 \cdot 1} \cdot \omega^n = \omega^{(n-1)} & \omega^{-2 \cdot 1} &= \omega^{-2 \cdot 1} \cdot \omega^n = \omega^{(n-2)} \\
\omega^{-1 \cdot 2} &= \omega^{-1 \cdot 2} \cdot \omega^{2n} = \omega^{2(n-1)} & \omega^{-2 \cdot 2} &= \omega^{-2 \cdot 2} \cdot \omega^{2n} = \omega^{2n-2 \cdot 2} = \omega^{2(n-2)} \\
\vdots &= \vdots & \vdots &= \vdots \\
\omega^{-1 \cdot (n-1)} &= \omega^{-1 \cdot (n-1)} \cdot \omega^{(n-1)n} = \omega^{(n-1)(n-1)} & \omega^{-2 \cdot (n-1)} &= \omega^{-2 \cdot (n-1)} \cdot \omega^{(n-1)n} = \omega^{(n-1)n-2(n-1)} = \omega^{(n-1)(n-2)} = \omega^{(n-1)\frac{n+1}{2}}
\end{aligned}$$

$$\omega^{-\frac{n-1}{2} \cdot 1} = \omega^{-\frac{n-1}{2} \cdot 1} \cdot \omega^n = \omega^{n - \frac{n-1}{2}} = \omega^{\frac{n+1}{2}}$$

$$\omega^{-\frac{n-1}{2} \cdot 2} = \omega^{-\frac{n-1}{2} \cdot 2} \cdot \omega^{2n} = \omega^{2n - \frac{n-1}{2} \cdot 2} = \omega^{2\frac{n+1}{2}}$$

$$\vdots = \vdots$$

$$\omega^{-\frac{n-1}{2} \cdot (n-1)} = \omega^{-\frac{n-1}{2} \cdot (n-1)} \cdot \omega^{(n-1)n} = \omega^{(n-1)n - \frac{n-1}{2} \cdot (n-1)} = \omega^{(n-1)\left(n - \frac{n-1}{2}\right)} = \omega^{(n-1)\frac{n+1}{2}}$$

Finally, one has a  $n \times n$  system of linear equations

$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{\frac{n-1}{2}} \\ x_{\frac{n+1}{2}} \\ \vdots \\ x_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 & 1 & \dots & 1 & 1 \\ 1 & \omega & \omega^2 & \dots & \omega^{\frac{n-1}{2}} & \omega^{\frac{n+1}{2}} & \dots & \omega^{(n-2)} & \omega^{(n-1)} \\ 1 & \omega^2 & \omega^{2 \cdot 2} & \dots & \omega^{\frac{n-1}{2} \cdot 2} & \omega^{\frac{n+1}{2} \cdot 2} & \dots & \omega^{(n-2) \cdot 2} & \omega^{(n-1) \cdot 2} \\ \vdots & \vdots & \ddots & & \ddots & \ddots & \ddots & \vdots & \vdots \\ 1 & \omega^{\frac{n-1}{2}} & \omega^{2\frac{n-1}{2}} & \dots & \omega^{\frac{n-1}{2} \frac{n-1}{2}} & \omega^{\frac{n+1}{2} \frac{n-1}{2}} & \dots & \omega^{(n-2)\frac{n-1}{2}} & \omega^{(n-1)\frac{n-1}{2}} \\ 1 & \omega^{\frac{n+1}{2}} & \omega^{2\frac{n+1}{2}} & \dots & \omega^{\frac{n-1}{2} \frac{n+1}{2}} & \omega^{\frac{n+1}{2} \frac{n+1}{2}} & \dots & \omega^{(n-2)\frac{n+1}{2}} & \omega^{(n-1)\frac{n+1}{2}} \\ \vdots & \vdots & \ddots & & \ddots & \ddots & \ddots & \vdots & \vdots \\ 1 & \omega^{(n-1)} & \omega^{2(n-1)} & \dots & \omega^{\frac{n-1}{2}(n-1)} & \omega^{\frac{n+1}{2}(n-1)} & \dots & \omega^{(n-2)(n-1)} & \omega^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{\frac{n-1}{2}} \\ c_{\frac{n+1}{2}} \\ \vdots \\ c_{-2} \\ c_{-1} \end{pmatrix}$$

# Appendix A.

## Supplementary Material

### A.1. Boltzmann Distribution

First calculate the following integral

$$\begin{aligned} \int_0^{+\infty} x^4 e^{-ax^2} dx &= \int_0^{+\infty} x^3 \cdot x \exp(-ax^2) dx = \left[ x \left( \frac{-e^{-ax^2}}{2a} \right) \right]_0^{+\infty} + \frac{1}{2a} \int_0^{+\infty} 3x^2 e^{-ax^2} dx \\ &= \frac{3}{2a} \int_0^{+\infty} x^2 e^{-ax^2} dx = \frac{3}{2a} \cdot \frac{1}{4a} \cdot \sqrt{\frac{\pi}{a}} = \frac{3}{8a^2} \sqrt{\frac{\pi}{a}} \end{aligned}$$

Then, with  $f_V(v) = 4\pi \left( \frac{m}{2\pi k_B T} \right)^{3/2} v^2 \exp\left(-\frac{mv^2}{2k_B T}\right)$ , one has

$$\begin{aligned} \int_0^{+\infty} \frac{1}{2} mv^2 f_V(v) dv &= \int_0^{+\infty} \frac{1}{2} mv^2 \cdot 4\pi \left( \frac{m}{2\pi k_B T} \right)^{3/2} v^2 \exp\left(-\frac{mv^2}{2k_B T}\right) dv \\ &= 2\pi m \left( \frac{m}{2\pi k_B T} \right)^{3/2} \int_0^{+\infty} v^4 \exp\left(-\frac{m}{2k_B T} v^2\right) dv \end{aligned}$$

Finally, with  $a = \frac{m}{2k_B T}$ , one has

$$\begin{aligned} \int_0^{+\infty} \frac{1}{2} mv^2 f_V(v) dv &= 2\pi m \left( \frac{m}{2\pi k_B T} \right)^{3/2} \cdot \frac{3}{8} \left( \frac{2k_B T}{m} \right)^2 \cdot \sqrt{\frac{2\pi k_B T}{m}} \\ &= \frac{3}{4} \cdot \frac{\pi}{m} \cdot 2k_B T \cdot 2k_B T \cdot \left( \frac{m}{2\pi k_B T} \right)^{3/2} \cdot \left( \frac{2\pi k_B T}{m} \right)^{1/2} \\ &= \frac{3}{2} k_B T \cdot \frac{2\pi k_B T}{m} \cdot \left( \frac{2\pi k_B T}{m} \right)^{1/2} \cdot \left( \frac{m}{2\pi k_B T} \right)^{3/2} \\ &= \frac{3}{2} k_B T \cdot \left( \frac{2\pi k_B T}{m} \right)^{3/2} \cdot \left( \frac{m}{2\pi k_B T} \right)^{3/2} = \frac{3}{2} k_B T \end{aligned}$$

# Appendix B.

## R-Code

### Example 1.2.1

With  $\text{R}$  the value of the probability density function  $\text{Uniform}([1, 10])$  at the position  $x = 5$  can be calculated as follows: [\(to Python\)](#)

```
dunif(x = 5, min = 1, max = 10)

## [1] 0.111
```

#### Caution:

Contrary to `dbinom()` or `dpois()`, `dunif()` does not calculate the corresponding probability, but the value of the *density function*.

In the case of  $X \sim \text{Uniform}([1, 10])$ , the probability  $P(1 \leq X \leq 5)$  corresponds exactly to the probability  $P(X \leq 5)$ . We calculate this with  $\text{R}$  as follows: [\(to Python\)](#)

```
punif(q = 5, min = 1, max = 10)

## [1] 0.444
```

The probability  $P(1.2 \leq X \leq 4.8)$  is calculated as follows [\(to Python\)](#)

```
punif(4.8, 1, 10) - punif(1.2, 1, 10)

## [1] 0.4
```

The generation of uniformly distributed random variables is of great significance. Uniformly distributed random variables can be generated with  $\text{R}$  as shown below: [\(to Python\)](#)

## Appendix B. R-Code

```
runif(5, min = 1, max = 10)

## [1] 7.28 6.77 6.80 9.80 4.73
```

### Example 1.2.5

Assuming  $X \sim \text{Exp}(3)$ , the probability  $P(0 \leq X \leq 4)$  can be calculated in [R](#) as follows:  
([to Python](#))

```
pexp(4, rate = 3)

## [1] 1
```

### Example 1.2.7

([to Python](#))

```
1 - pnorm(130, mean = 100, sd = 15)

## [1] 0.0228
```

([to Python](#))

```
pnorm(115, mean = 100, sd = 15) - pnorm(85, mean = 100,
sd = 15)

## [1] 0.683
```

### Example 1.4.4

([to Python](#))

```
pnorm(5100, 5000, sqrt(2500))

## [1] 0.977
```

## Appendix B. R-Code

([to Python](#))

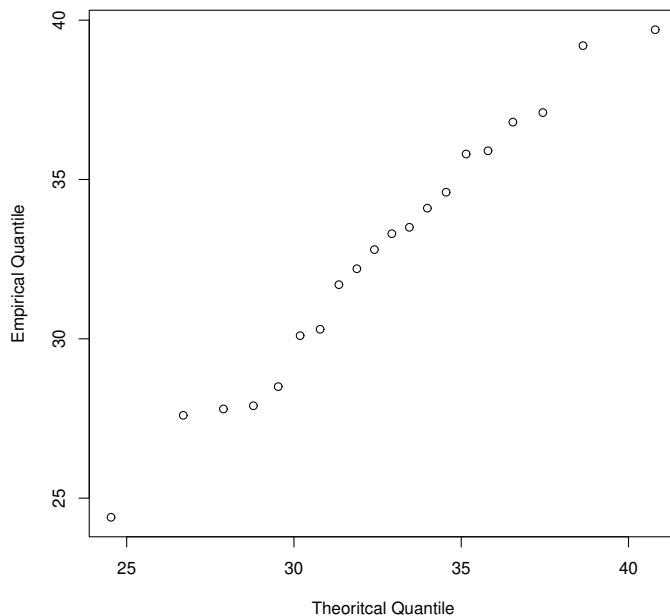
```
pbinary(5100, 10000, 0.5)  
## [1] 0.978
```

### Example 2.1.1

([to Python](#))

## Appendix B. R-Code

```
# Values ordered by size:  
x <- c(24.4, 27.6, 27.8, 27.9, 28.5, 30.1, 30.3, 31.7, 32.2,  
      32.8, 33.3, 33.5, 34.1, 34.6, 35.8, 35.9, 36.8, 37.1,  
      39.2, 39.7)  
# Quantiles  
alpha_k <- (seq(1, length(x), by = 1) - 0.5)/length(x)  
  
quantile_theor <- qnorm(alpha_k, mean = mean(x), sd = sd(x))  
quantile_empir <- sort(x)  
  
# Q-Q-Plot  
qqplot(quantile_theor, quantile_empir, xlab = "Theoritical Quantile",  
       ylab = "Empirical Quantile")
```



**Figure B.1.:** Q-Q-Plot on concrete compression strength

### Example 2.3.1

(to Python)

```
qnorm(0.025)  
  
## [1] -1.96
```

## Appendix B. R-Code

(to Python)

```
qnorm(0.025, 80, 0.01/sqrt(13))  
  
## [1] 80  
  
qnorm(0.975, 80, 0.01/sqrt(13))  
  
## [1] 80
```

### Example 2.3.2

(to Python)

```
x <- c(5.9, 3.4, 6.6, 6.3, 4.2, 2, 6, 4.8, 4.2, 2.1, 8.7,  
      4.4, 5.1, 2.7, 8.5, 5.8, 4.9, 5.3, 5.5, 7.9)  
mean.x <- mean(x)  
mean.x  
  
## [1] 5.21  
  
sigma.x <- sd(x)  
sigma.x  
  
## [1] 1.88  
  
t <- (mean.x - 5)/(sigma.x/sqrt(20))  
t  
  
## [1] 0.51  
  
pt(t, df = 19)  
  
## [1] 0.692
```

### Example 2.3.3

(to Python)

## Appendix B. R-Code

```
qt(0.975, 12)

## [1] 2.18

(to Python)

x <- c(79.98, 80.04, 80.02, 80.04, 80.03, 80.03, 80.04,
      79.97, 80.05, 80.03, 80.02, 80, 80.02)

t.test(x, alternative = "two.sided", mu = 80, conf.level = 0.95)

##
## One Sample t-test
##
## data: x
## t = 3, df = 12, p-value = 0.009
## alternative hypothesis: true mean is not equal to 80
## 95 percent confidence interval:
## 80 80
## sample estimates:
## mean of x
##             80
```

### Example 2.3.4

(to Python):

```
1 - pt(3.1246, df = 12)

## [1] 0.00439
```

### Example 3.5.1

(to Python)

```
pr.out <- prcomp(USArrests[, c("Murder", "Assault")])
pr.out$rotation[, 1]

##
## Murder Assault
## -0.0419 -0.9991
```

## Appendix B. R-Code

(to Python)

```
pr.out <- prcomp(USArrests[, c("Murder", "Assault")])  
head(pr.out$x)  
  
## PC1 PC2  
## Alabama -65.4 2.673  
## Alaska -92.3 -1.656  
## Arizona -123.1 -4.854  
## Arkansas -19.3 0.205  
## California -105.2 -3.200  
## Colorado -33.2 -1.281
```

(to Python)

```
pr.out <- prcomp(USArrests[, c("Murder", "Assault")])  
pr.out$rotation[, 2]  
  
## Murder Assault  
## 0.9991 -0.0419
```

## Example 3.5.2

(to Python)

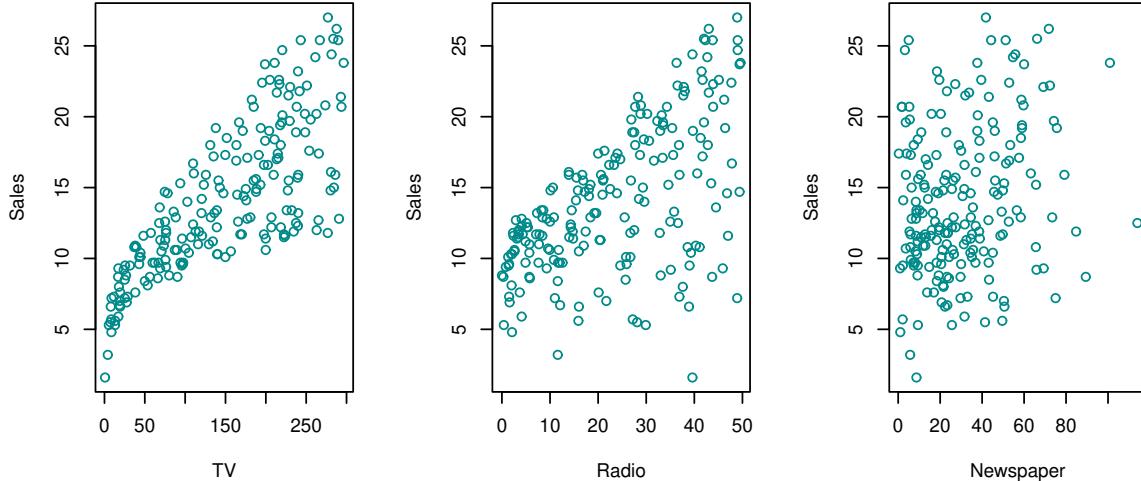
```
pr.out <- prcomp(USArrests[, c("Murder", "Assault")], scale = FALSE)  
pr.var <- pr.out$sdev^2  
pve <- pr.var/sum(pr.var)  
pve  
  
## [1] 0.999029 0.000971
```

## Example 4.1.1

(to Python).

## Appendix B. R-Code

```
par(mfrow=c(1, 3))
plot(sales ~ TV, col = "darkcyan",
      xlab = "TV", ylab = "Sales", data = adv)
```



**Figure B.2.:** The plot displays `sales`, in thousands of units, as a function of `TV`, `radio` and `newspaper` for 200 different markets.

### Example 5.2.4

We are interested in the values of  $\hat{\beta}_0$  and  $\hat{\beta}_1$  for the **Advertising** data. By means of the R-function `lm()`, we can easily determine those values: [\(to Python\)](#):

```
## 
## Call:
## lm(formula = sales ~ TV, data = Advertising)
## 
## Coefficients:
## (Intercept)          TV
##           7.0326     0.0475
```

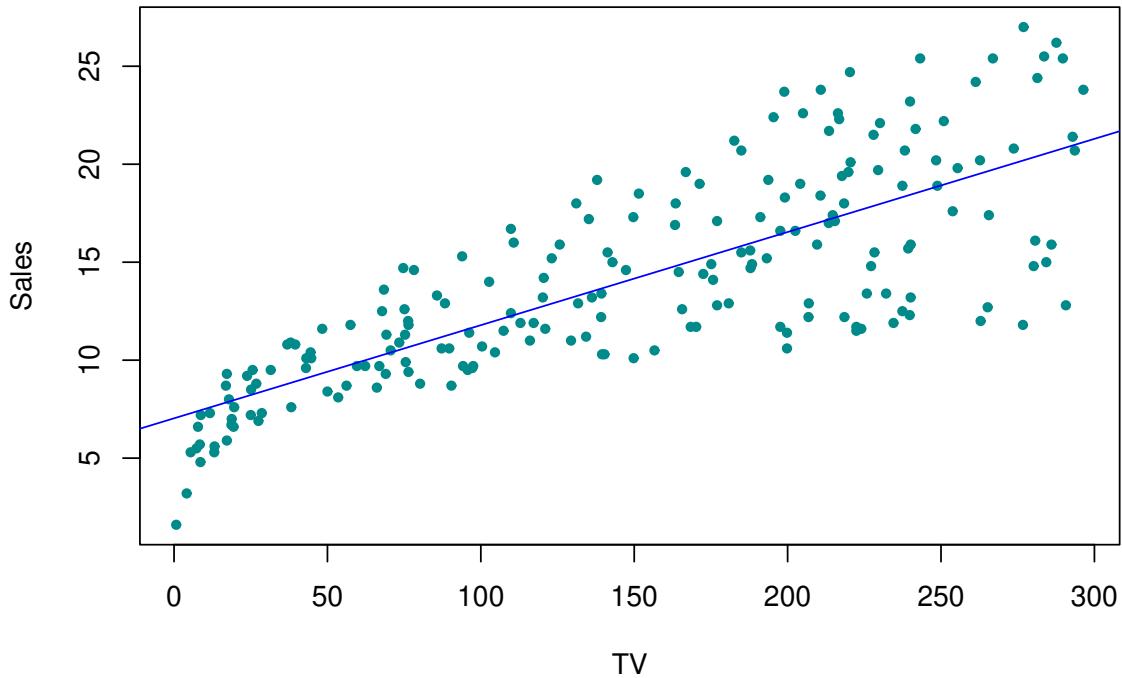
The value listed under `(Intercept)` corresponds to  $\hat{\beta}_0$ , thus to the intercept with the  $y$ -axis, and the value listed under `TV` corresponds to the slope of the regression line, thus to  $\hat{\beta}_1$ .

[\(to Python\)](#)

### Example 5.3.1

## Appendix B. R-Code

```
# read.csv('.../Themen/Einfache_Lineare_Regression/Daten/Advertising.csv')
```



**Figure B.3.:** For the `Advertising` data, the least squares fit for the regression of `sales` onto `TV` is shown.

([to Python](#)) .

```
x <- runif(100, -2, 2)
y <- 2 + 3 * x + rnorm(100, 0, 4)
```

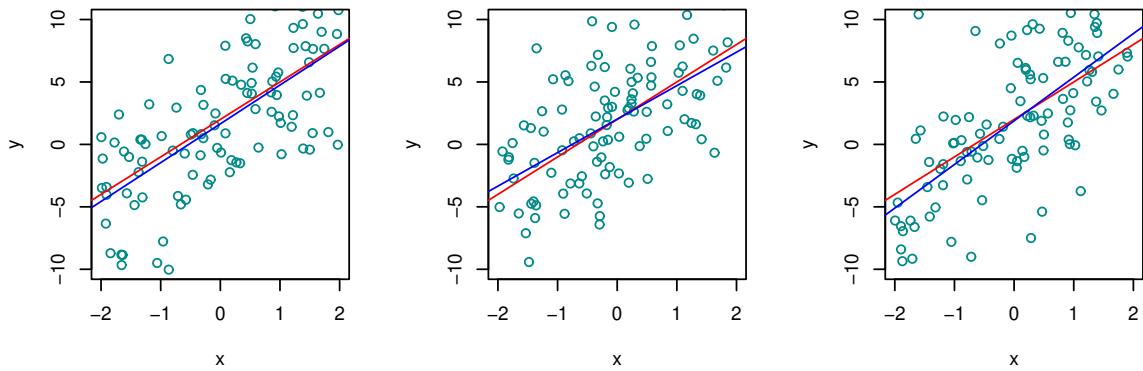
Figure 5.4 displays 3 simulations, each containing new simulated data according to the above procedure. ([to Python](#))

### Example 5.3.3

([to Python](#)) .

```
## 
## Call:
## lm(formula = sales ~ TV, data = Advertising)
```

## Appendix B. R-Code



**Figure B.4.:** Each panel corresponds to a simulated data set. The red line represents the true relationship  $f(X) = 2 + 3X$ , which is known as the population regression line. The blue line is the least squares line: it is the least squares estimate for  $f(X) = \beta_0 + \beta_1 x$  based on the observed (simulated) data.

```
## 
## Residuals:
##   Min     1Q Median     3Q    Max 
## -8.386 -1.955 -0.191  2.067  7.212 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 7.03259   0.45784   15.4   <2e-16 ***
## TV          0.04754   0.00269   17.7   <2e-16 ***
## ---        
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 
## 
## Residual standard error: 3.26 on 198 degrees of freedom
## Multiple R-squared:  0.612, Adjusted R-squared:  0.61  
## F-statistic: 312 on 1 and 198 DF,  p-value: <2e-16
```

In the table `Coefficients` within the column, `Estimate` the listed values are  $\hat{\beta}_0$  and  $\hat{\beta}_1$ . In the column `Std. Error` we find the values 0.45784 and 0.002691 for the two standard errors  $se(\hat{\beta}_0)$  and  $se(\hat{\beta}_1)$ . They correspond to the average deviations of the estimated values of  $\hat{\beta}_0$  and  $\hat{\beta}_1$  which are 7.0326 and 0.04754. Under `Residual Standard Error` we find the residual standard error which is 3.259, where the degrees of freedom are given by  $n - 2 = 198$ . The residual standard error provides us with an estimation of the standard deviation of the error term  $\varepsilon$ .

We now will compute the p-value, that is, the probability of observing a value of the t-statistic larger than  $|t| = 17.66518$ . Assuming  $\beta_1 = 0$ ,  $T$  will follow a t-distribution

## Appendix B. R-Code

with  $n - 2 = 198$  degrees of freedom. Then the (two-sided) p-value can be determined with the help of R as follows ([to Python](#))

```
2 * (1 - pt(17.66518, df = 198))  
## [1] 0
```

Since the alternative hypothesis is two-sided, we need to multiply the one-sided p-value by two in order to obtain the two-sided p-value. In the table [Coefficients](#) we find under [Pr\(>|t|\)](#) the corresponding p-value :  $< 2 \cdot 10^{-16}$ . Since it is virtually zero, hence smaller than a significance level of 0.05 , we *reject* the null hypothesis  $\beta_1 = 0$  in favor of the alternative hypothesis  $\beta_1 \neq 0$ . We therefore find that there clearly is a relationship between **TV** and **sales**.

### Example 5.3.5

In the case of the **Advertising** data, the 95 % confidence interval can be found with the help of the R-function `confint()`: ([to Python](#))

```
# read.csv('.../.../.../Themen/Einfache_Lineare_Regression/Daten/Advertising.csv')  
  
##                2.5 % 97.5 %  
## (Intercept) 6.1297 7.9355  
## TV          0.0422 0.0528
```

With R we determine the 97.5 % quantile of a t-distribution as follows ([to Python](#))

```
qt(0.975, 20 - 2)  
## [1] 2.1
```

which is approximately 2.

### Example 5.4.2

([to Python](#))

```
##      fit    lwr    upr  
## 1 7.18  6.29  8.06  
## 2 11.79 11.27 12.30  
## 3 20.11 19.29 20.92
```

## Appendix B. R-Code

In the R-output , the values

$$\hat{y}_0 = \hat{\beta}_0 + \hat{\beta}_1 x_0$$

can be found under `fit` ; they correspond to the  $y$ -values on the regression line, thus to the predicted response given a predictor value.

Under `lwr` the lower limits, under `upr` the upper limits of the corresponding confidence intervals can be found.

([to Python](#))

### Example 5.4.4

In the case of the **Advertising** data we determine the 95 % prediction intervals for the  $x$ -values 3, 100 and 225 as follows: ([to Python](#))

```
Verkauf <- Advertising[, 5]

##      fit      lwr     upr
## 1  7.18  0.688 13.7
## 2 11.79  5.339 18.2
## 3 20.11 13.627 26.6
```

Under `fit` the predicted  $y$ -values on the regression line

$$\hat{y}_0 = \hat{\beta}_0 + \hat{\beta}_1 x_0$$

can be found.

`lwr` displays the lower limits and `upr` the upper limits of the prediction intervals for the given  $x$ -values.

([to Python](#))

For completeness, `TMA_def`, used throughout this chapter, is given here. In principle, every function is first shown in an example, before being added to this set of functions.

```
[ ]: """
    Class svm containing definitions for Chapter TMA
    by Simon van Hemert
    date created: 07-07-21 """

import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.graphics.gofplots import ProbPlot
import random
import pandas as pd
import numpy as np

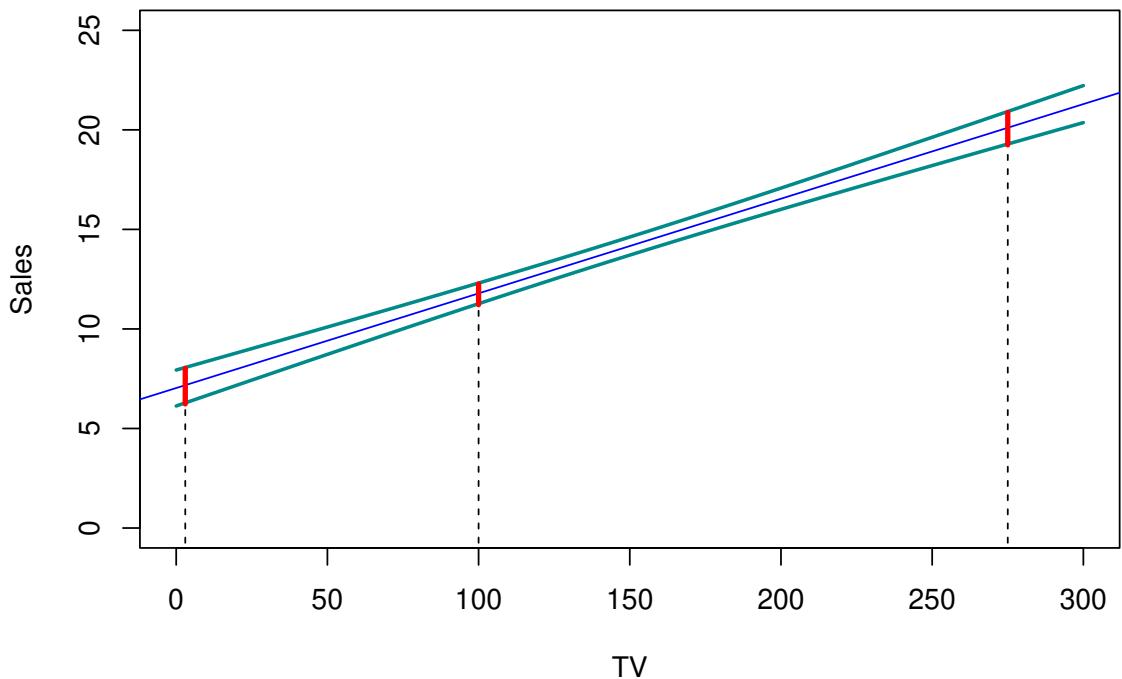
"""
    Standard scatter plot and regression line """
def plot_reg(axes, x, y, model, x_lab="x", y_lab="y", title="Linear Regression"):
```

## Appendix B. R-Code

```

plot(sales ~ TV, data = Advertising, col="white", xlab="TV",
      ylab = "Sales", pch=20, ylim=c(0,25),
      xlim = c(0,300))
lines(dat$TV, conf[,3], ylim=c(0,25), xlim=c(0,300), type="l",
      xlab="", ylab="", col="darkcyan", lwd=2)
for (i in c(3, 100, 275))
{
  lines(c(i, i), c(-10, conf[i, 3]), lty = "dashed")
  lines(c(i, i), c(conf[i, 2], conf[i, 3]), col = "red", lwd = 3)
}

```



**Figure B.5.:** Confidence intervals for the expected values of  $\hat{y}_0$ . The region enclosed by the green curves is called *confidence band*.

```

""" Inputs:
axes: axes created with matplotlib.pyplot
x: (single) Feature
y: Result
model: fitted linear sm model """
# Plot scatter data
sns.regplot(x=x, y=y,
            scatter=True, ci=False, lowess=False,
            scatter_kws={'s': 40, 'alpha': 0.5},
            line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8})
# Set labels:

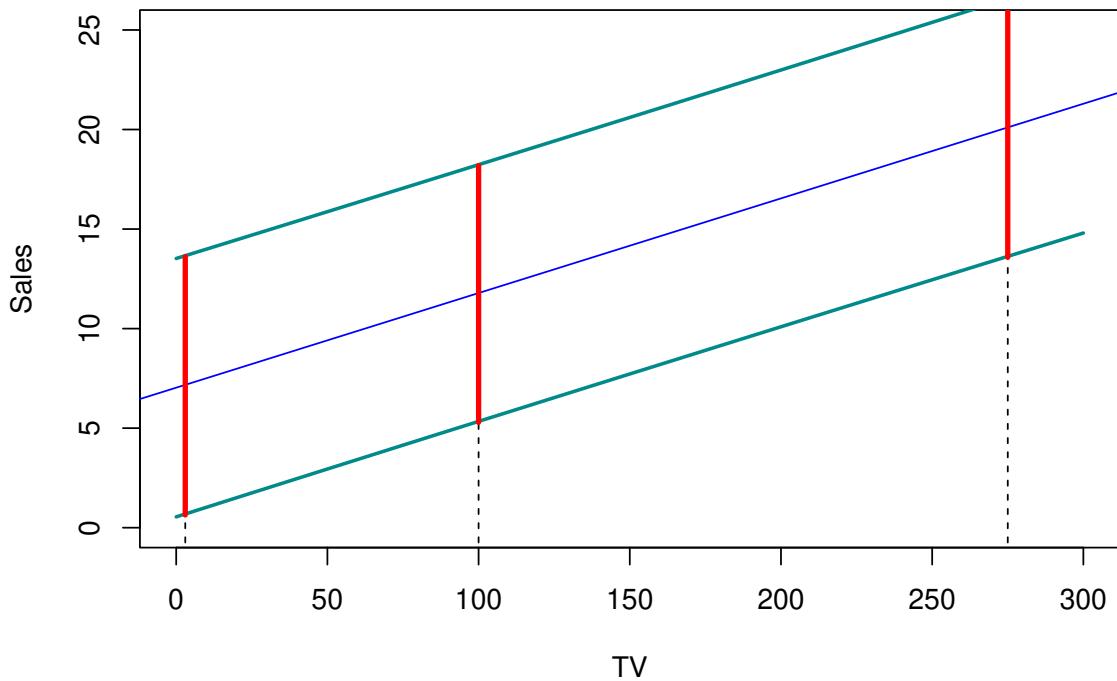
```

## Appendix B. R-Code

```

plot(sales ~ TV , col="white", xlab="TV", ylab="Sales",
      pch=20, ylim=c(0,25), xlim=c(0,300), data = Advertising)
lines(dat$TV, pi[,3], ylim=c(0,25), xlim=c(0,300), type="l",
      xlab="", ylab="", col="darkcyan", lwd=2)
for (i in c(3, 100, 275)) {
  lines(c(i, i), c(-10, pi[i, 3]), lty = "dashed")
  lines(c(i, i), c(pi[i, 2], pi[i, 3]), col = "red", lwd = 3)
}

```



**Figure B.6.:** Pointwise prediction intervals for future observations are displayed as green curves along with the blue regression line. The region enclosed by the green curves is called *prediction band*.

```

axes.set_xlabel(x_lab)
axes.set_ylabel(y_lab)
axes.set_title(title)

"""
Plot Residuals vs Fitted Values """
def plot_residuals(axes, yfit, res, n_samp=0,
                   x_lab='Fitted Values', y_lab='Residuals', title='Residuals vs Fitted'):
    """
    Inputs:
    axes: axes created with matplotlib.pyplot
    x: x values
    yfit: fitted/predicted y values

```

## Appendix B. R-Code

```

n_samp[optional]: number of resamples"""
# For every random resampling
for i in range(n_samp):
    # 1. resample indices from Residuals
    samp_res_id = random.sample(list(res), len(res))
    # 2. Average of Residuals, smoothed using LOWESS
    sns.regplot(x=yfit, y=samp_res_id,
                scatter=False, ci=False, lowess=True,
                line_kws={'color': 'lightgrey', 'lw': 1, 'alpha': 0.8})
    # 3. Repeat again for n_samples

df = pd.concat([yfit, res], axis=1)
axes = sns.residplot(x=yfit, y=res, data=df,
                      lowess=True, scatter_kws={'alpha': 0.5},
                      line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8})
axes.set_title(title)
axes.set_ylabel(y_lab)
axes.set_xlabel(x_lab)

""" Scale-Location Plot """
def plot_scale_loc(axes, yfit, res_stand_sqrt, n_samp=0,
                    x_lab='Fitted Values', y_lab='$\sqrt{|\text{Standardized Residuals}|}$',
                    title='Scale-Location plot'):

    """ Inputs:
    axes: axes created with matplotlib.pyplot
    yfit: fitted/predicted y values
    res_stand_sqrt: Absolute square root Residuals
    n_samp[optional]: number of resamples """
    # For every random resampling
    for i in range(n_samp):
        # 1. resample indices from sqrt Residuals
        samp_res_id = random.sample(list(res_stand_sqrt), len(res_stand_sqrt))
        # 2. Average of Residuals, smoothed using LOWESS
        sns.regplot(x=yfit, y=samp_res_id,
                    scatter=False, ci=False, lowess=True,
                    line_kws={'color': 'lightgrey', 'lw': 1, 'alpha': 0.8})
        # 3. Repeat again for n_samples

    # plot Regression usung Seaborn
    sns.regplot(x=yfit, y=res_stand_sqrt,
                scatter=True, ci=False, lowess=True,
                scatter_kws={'s': 40, 'alpha': 0.5},
                line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8})
    axes.set_title(title)
    axes.set_ylabel(y_lab)
    axes.set_xlabel(x_lab)

""" QQ Plot standardized residuals """
def plot_QQ(axes, res_standard, n_samp=0,
            x_lab='Theoretical Quantiles', y_lab='Standardized Residuals',
            title='Normal Q-Q'):

    """ Inputs:
    axes: axes created with matplotlib.pyplot
    res_standard: standardized residuals
    n_samp[optional]: number of resamples """
    # QQ plot instance
    QQ = ProbPlot(res_standard)
    # Split the QQ instance in the seperate data
    qqx = pd.Series(sorted(QQ.theoretical_quantiles), name="x")
    qqy = pd.Series(QQ.sorted_data, name="y")
    if n_samp != 0:
        # Estimate the mean and standard deviation
        mu = np.mean(qqy)
        sigma = np.std(qqy)

```

## Appendix B. R-Code

```

# For ever random resampling
for lp in range(n_samp):
    # Resample indices
    samp_res_id = np.random.normal(mu, sigma, len(qqx))
    # Plot
    sns.regplot(x=qqx, y=sorted(samp_res_id),
                scatter=False, ci=False, lowess=True,
                line_kws={'color': 'lightgrey', 'lw': 1, 'alpha': 0.8})

    sns.regplot(x=qqx, y=qqy, scatter=True, lowess=False, ci=False,
                scatter_kwss={'s': 40, 'alpha': 0.5},
                line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8})
    axes.plot(qqx, qqx, '--k', alpha=0.5)
    axes.set_title(title)
    axes.set_ylabel(y_lab)
    axes.set_xlabel(x_lab)

""" Cook's distance """
def plot_cooks(axes, res_inf_leverage, res_standard, n_pred=1,
               x_lim=None, y_lim=None, n_levels=4):
    """ Inputs:
    axes: axes created with matplotlib.pyplot
    res_inf_leverage: Leverage
    res_standard: standardized residuals
    n_pred: number of predictor variables in x
    x_lim, y_lim[optional]: axis limits
    n_levels: number of levels"""
    sns.regplot(x=res_inf_leverage, y=res_standard,
                scatter=True, ci=False, lowess=True,
                scatter_kwss={'s': 40, 'alpha': 0.5},
                line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8})
    # Set limits
    if x_lim != None:
        x_min, x_max = x_lim[0], x_lim[1]
    else:
        x_min, x_max = min(res_inf_leverage), max(res_inf_leverage)
    if y_lim != None:
        y_min, y_max = y_lim[0], y_lim[1]
    else:
        y_min, y_max = min(res_standard), max(res_standard)

    # Plot centre line
    plt.plot((x_min, x_max), (0, 0), 'g--', alpha=0.8)
    # Plot contour lines for Cook's Distance levels
    n = 100
    cooks_distance = np.zeros((n, n))
    x_cooks = np.linspace(x_min, x_max, n)
    y_cooks = np.linspace(y_min, y_max, n)

    for xi in range(n):
        for yi in range(n):
            cooks_distance[yi][xi] = \
                y_cooks[yi]**2 * x_cooks[xi] / (1 - x_cooks[xi]) / (n_pred + 1)
    CS = axes.contour(x_cooks, y_cooks, cooks_distance, levels=n_levels, alpha=0.6)

    axes.clabel(CS, inline=0, fontsize=10)
    axes.set_xlim(x_min, x_max)
    axes.set_ylim(y_min, y_max)
    axes.set_title('Residuals vs Leverage and Cook\\'s distance')
    axes.set_xlabel('Leverage')
    axes.set_ylabel('Standardized Residuals')

```

### Example 6.2.3

([to Python](#))

```
set.seed(7)
x <- seq(-4, 2, 0.4)
y <- -x^2 + 4 + rnorm(length(x), 0, 2)
summary(lm(y ~ x))

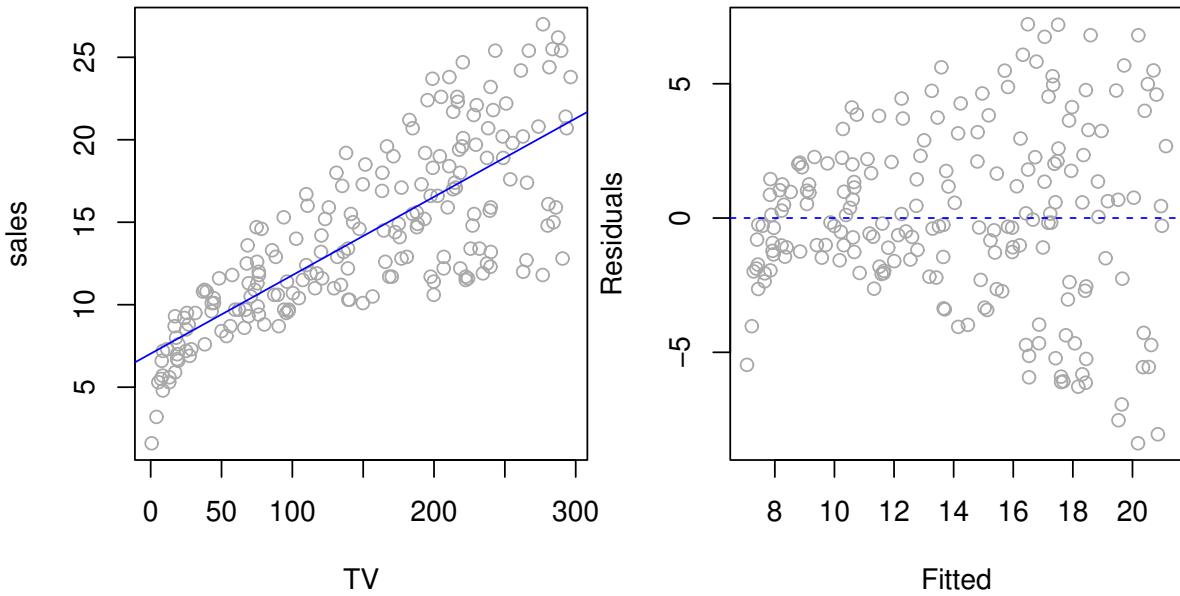
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -7.681 -2.396 -0.141  2.418  5.908
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.362      1.080    3.11  0.00761 ***
## x            2.627      0.515    5.10  0.00016 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.8 on 14 degrees of freedom
## Multiple R-squared:  0.65, Adjusted R-squared:  0.626
## F-statistic: 26.1 on 1 and 14 DF,  p-value: 0.00016
```

The RSE is given by 3.796 and the  $R^2$  value is 0.651.

### Example 6.2.4

([to Python](#))

```
par(mfrow = c(1, 2), pty = "s")
advertising <- read.csv("./Daten/Advertising.csv")
fit <- lm(sales ~ TV, data = advertising)
abline(fit, col = "blue")
plot(fitted(fit), resid(fit), col = "darkgrey", xlab = "Fitted",
     ylab = "Residuals")
abline(h = 0, lty = "dashed", col = "blue")
```



**Figure B.7.:** *Left* : Scatterplot for the **Advertising** data set. *Right*: Plots of residuals versus predicted (or fitted) values of **sales** for the **Advertising** data set.

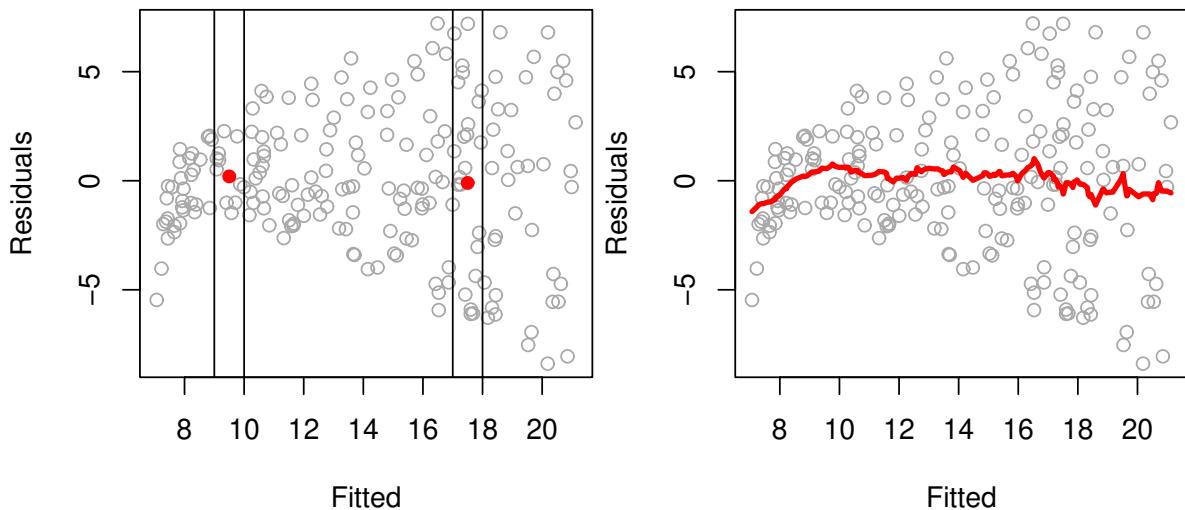
### Example 6.2.5

([to Python](#))

```
par(mfrow = c(1, 2))
advertising <- read.csv("./Daten/Advertising.csv")
plot(fitted(fit), resid(fit), col = "darkgrey", xlab = "Fitted",
      ylab = "Residuals")
abline(v = c(9, 10, 17, 18))
points(c(9.5, 17.5), c(0.2, -0.1), pch = 20, col = "red",
       lwd = 3)
plot(fitted(fit), resid(fit), col = "darkgrey", xlab = "Fitted",
      ylab = "Residuals")
# The R-function ksmooth calculates the points on the
# red curve
fit_smooth <- ksmooth(fitted(fit), resid(fit), kernel = "box",
                      bandwidth = 2, n.points = 24, x.points = fitted(fit))
lines(fit_smooth, col = "red", lwd = 3)
```

([to Python](#)):

## Appendix B. R-Code



**Figure B.8.:** Smoothing curve based on the running mean estimation method. `kernel="box"` indicates that a rectangular kernel is used and `bandwidth` steers the window width.

```
wd <- getwd()  
setwd(wd)
```

### Example 6.2.6

([to Python](#))

The principle idea of our resampling approach consists of simulating data points on the basis of the existing data set. For the simulated data points we fit a smoothing curve and add it to the Tukey-Anscombe plot. ([to Python](#))

### Example 6.2.9

([to Python](#))

```
Advertising <- read.csv("./Daten/Advertising.csv")  
plot(fit, col = "darkgrey", which = 3)
```

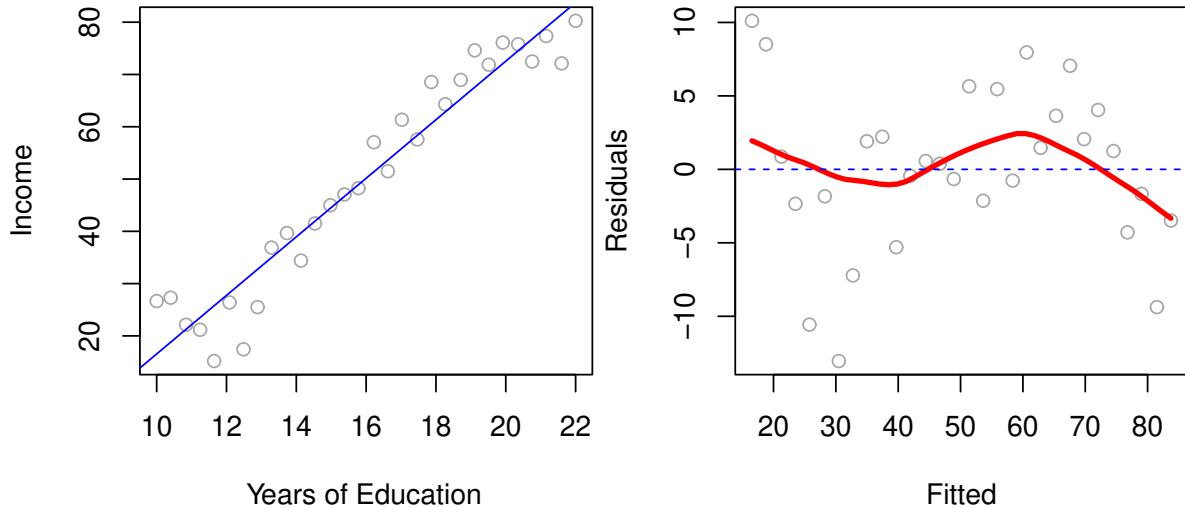
([to Python](#))

## Appendix B. R-Code

```

par(mar=c(4, 4, 0, 0)+0.1, mfrow=c(1, 2))
In <- read.csv("./Daten/Income.csv")
fit <- lm(In[, 3] ~ In[, 2], data=In)
abline(fit, col="blue")
plot(fitted(fit), resid(fit), col="darkgrey",
      xlab="Fitted", ylab="Residuals")
lines(loess.smooth(fitted(fit), resid(fit)), col="red", lwd=3)
abline(h=0, lty="dashed", col="blue")

```



**Figure B.9.:** Tukey-Anscombe Plot for the Example `Income`.

## Example 6.2.11

([to Python](#))

```

Advertising <- read.csv("./Daten/Advertising.csv")
hist(resid(fit), col = "darkgrey", main = "", xlab = "Residuals",
      ylab = "Frequency", xlim = c(-10, 10))
box()
par(new = T)
x <- seq(-10, 10, length = 100)
plot(x, dnorm(x, mean(resid(fit)), sd(resid(fit))), xlab = "",
      xlim = c(-10, 10), ylab = "", type = "l", col = "darkcyan",
      axes = F)

```

## Example 6.2.12

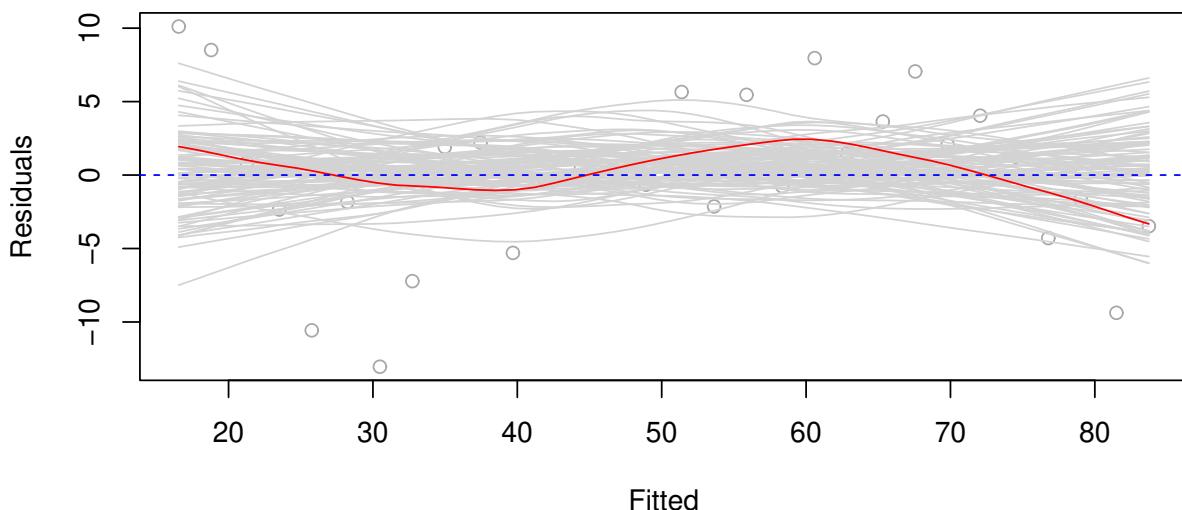
([to Python](#))

## Appendix B. R-Code

```

par(mar = c(4, 4, 0, 0) + 0.1)
income <- read.csv("./Daten/Income.csv")
fit <- lm(income ~ education, data = income)
set.seed(7)
plot(fitted(fit), resid(fit), col = "darkgrey", xlab = "Fitted",
      ylab = "Residuals")
for (i in 1:100) {
  sresid <- sample(resid(fit), replace = TRUE)
  lines(loess.smooth(fitted(fit), sresid), col = "lightgrey",
        lwd = 1)
}
lines(loess.smooth(fitted(fit)), resid(fit)), col = "red",
      lwd = 1)
abline(h = 0, col = "blue", lty = "dashed")

```



**Figure B.10.:** A band consisting of 100 (grey) smoothing curves is shown; these curves were fitted based on resampled data points.

```

Advertising <- read.csv("./Daten/Advertising.csv")
plot(fit, which = 2)

```

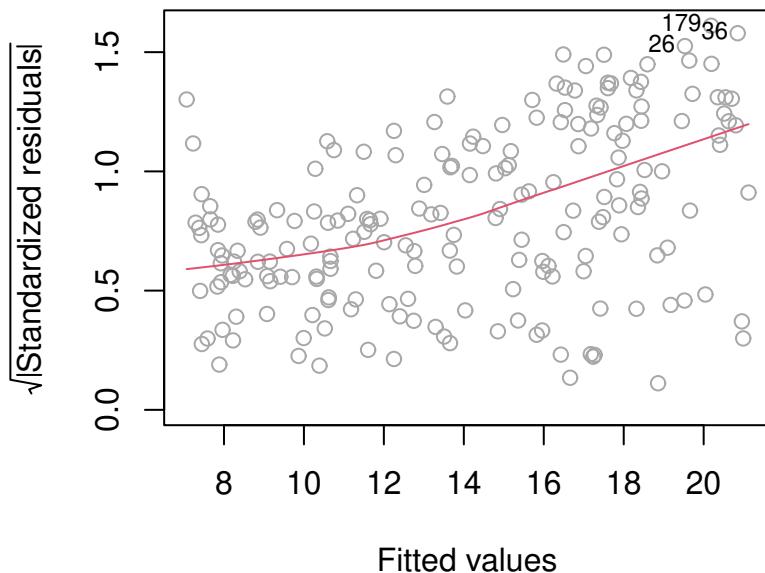
([to Python](#))

### Example 6.2.14

With **R** we can plot all of them by simply specifying the corresponding number in `plot(...,which=...)`.

For the **Advertising** example the 4 residual plots are displayed in Figure 6.24.  
([to Python](#))

## Appendix B. R-Code



**Figure B.11.:** Scale location plot for the example data set **Advertising**.

```
par(mfrow = c(2, 2))
Advertising <- read.csv("./Daten/Advertising.csv")
plot(fit, col = "darkgrey")
```

## Example 6.2.15

## Example 7.1.3

([to Python](#))

```
income <- read.csv('./Daten/Income2.csv')
library(scatterplot3d)
s3d <- scatterplot3d(income[,2], income[,3], income[,4],
                      highlight.3d=T, type="h", pch=16,
                      angle=30, xlab="Years of Education",
                      ylab="Years of Experience",
                      zlab="Income")
```

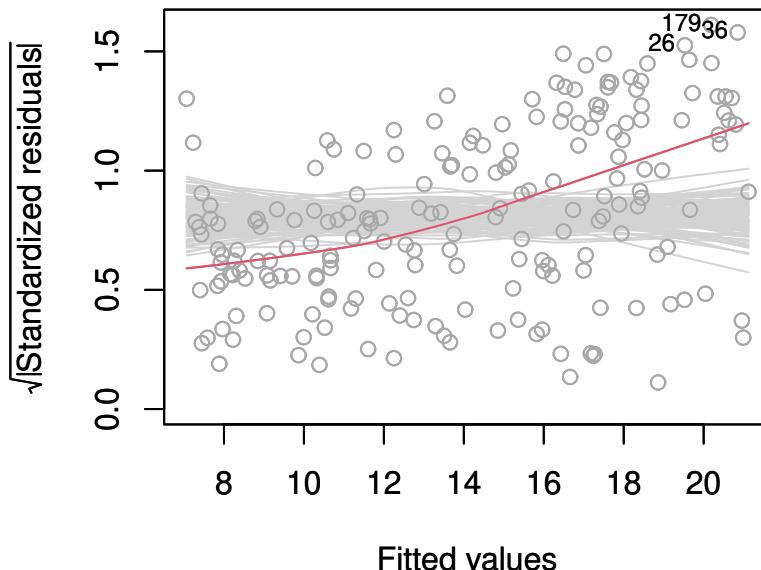
([to Python](#))

```
income <- read.csv('./Daten/Income2.csv')
coef(lm(income ~ education + experience, data=income))

## (Intercept)      education      experience
##       -50.086          5.896         0.173
```

## Appendix B. R-Code

```
Advertising <- read.csv("./Daten/Advertising.csv")
plot(fit, col = "darkgrey", which = 3)
for (i in 1:100) {
  sresid <- sqrt(abs(sample(rstandard(fit), replace = TRUE)))
  lines(loess.smooth(fitted(fit), sresid), col = "lightgrey",
        lwd = 1)
}
par(new = T)
plot(fit, col = "darkgrey", which = 3)
```



**Figure B.12.:** Scale location plot with smoothing curve (in red) and simulated smoothing curves fitted on the basis of resampled data (in grey).

## Example 7.2.1

```
Advertising <- read.csv('./Daten/Advertising.csv')

coef(lm(sales ~ TV + radio + newspaper, data=Advertising))

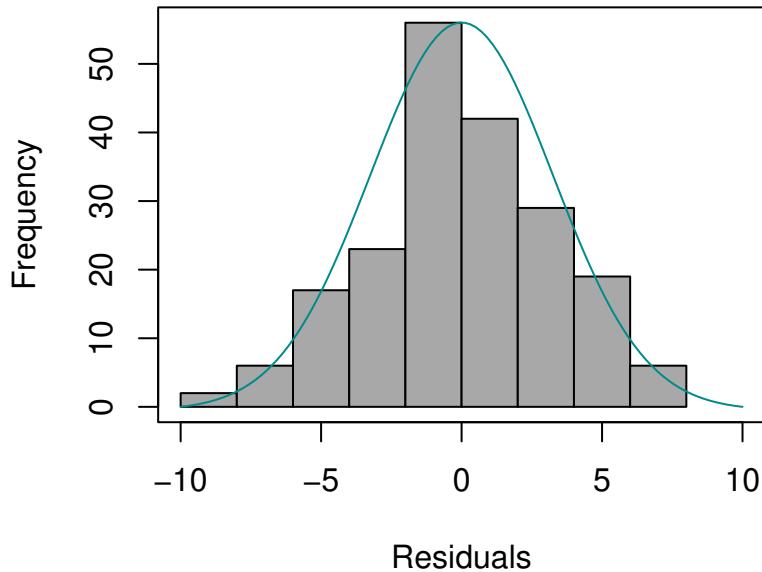
## (Intercept)          TV          radio      newspaper
##       2.93889       0.04576      0.18853     -0.00104
```

(to Python)

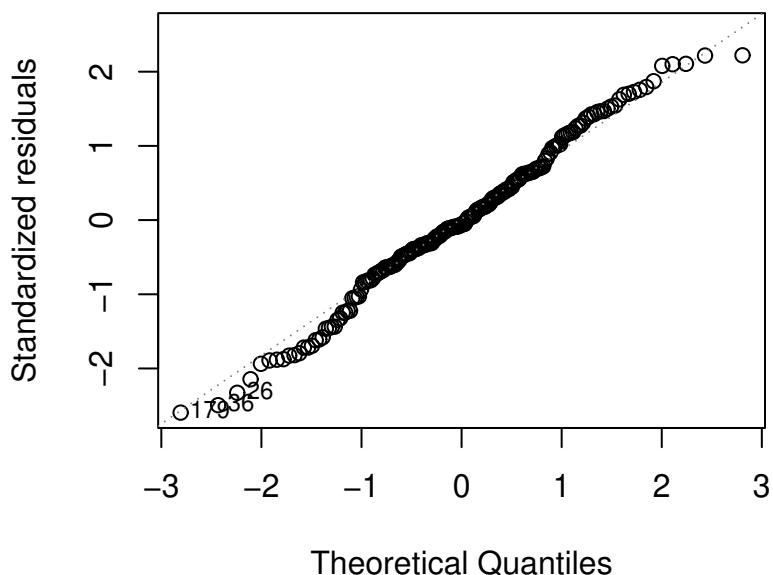
```
Advertising <- read.csv('./Daten/Advertising.csv')

TV <- Advertising[, 2]
radio <- Advertising[, 3]
```

## Appendix B. R-Code



**Figure B.13.:** Histogram of the residuals  $r_i$  for the **Advertising** data set. The green curve is the normal density function for which the distribution parameters were estimated on the basis of the data.



**Figure B.14.:** Q-Q plot for the **Advertising** data.

```
newspaper <- Advertising[, 4]
sales <- Advertising[, 5]

cor(data.frame(TV, radio, newspaper, sales))

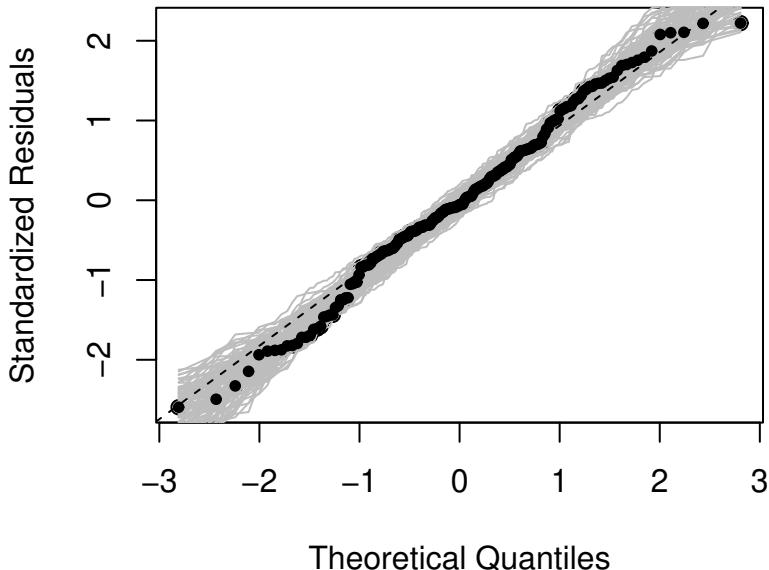
##          TV    radio newspaper sales
TV        1.0000000
radio     0.7094750
newspaper 0.1793560
sales     0.9265350
```

## Appendix B. R-Code

```

par(mar=c(4, 4, 0, 0)+0.1)
Advertising <- read.csv("./Daten/Advertising.csv")
qq <- qqnorm(rstandard(fit), main="", ylab="Standardized Residuals")
for (i in 1:100)
{
  sresid <- rnorm(length(qq$x), mean(qq$y), sd(qq$y))
  lines(sort(qq$x), sort(sresid), col="grey")
}
points(qq$x, qq$y, pch=20)
qqline(rstandard(fit), lty=2)

```



**Figure B.15.:** The (grey) band contains 100 smoothing curves that were fitted to the resampled data.

## TV	1.0000	0.0548	0.0566	0.782
## radio	0.0548	1.0000	0.3541	0.576
## newspaper	0.0566	0.3541	1.0000	0.228
## sales	0.7822	0.5762	0.2283	1.000

### Example 7.3.1

([to Python](#)).

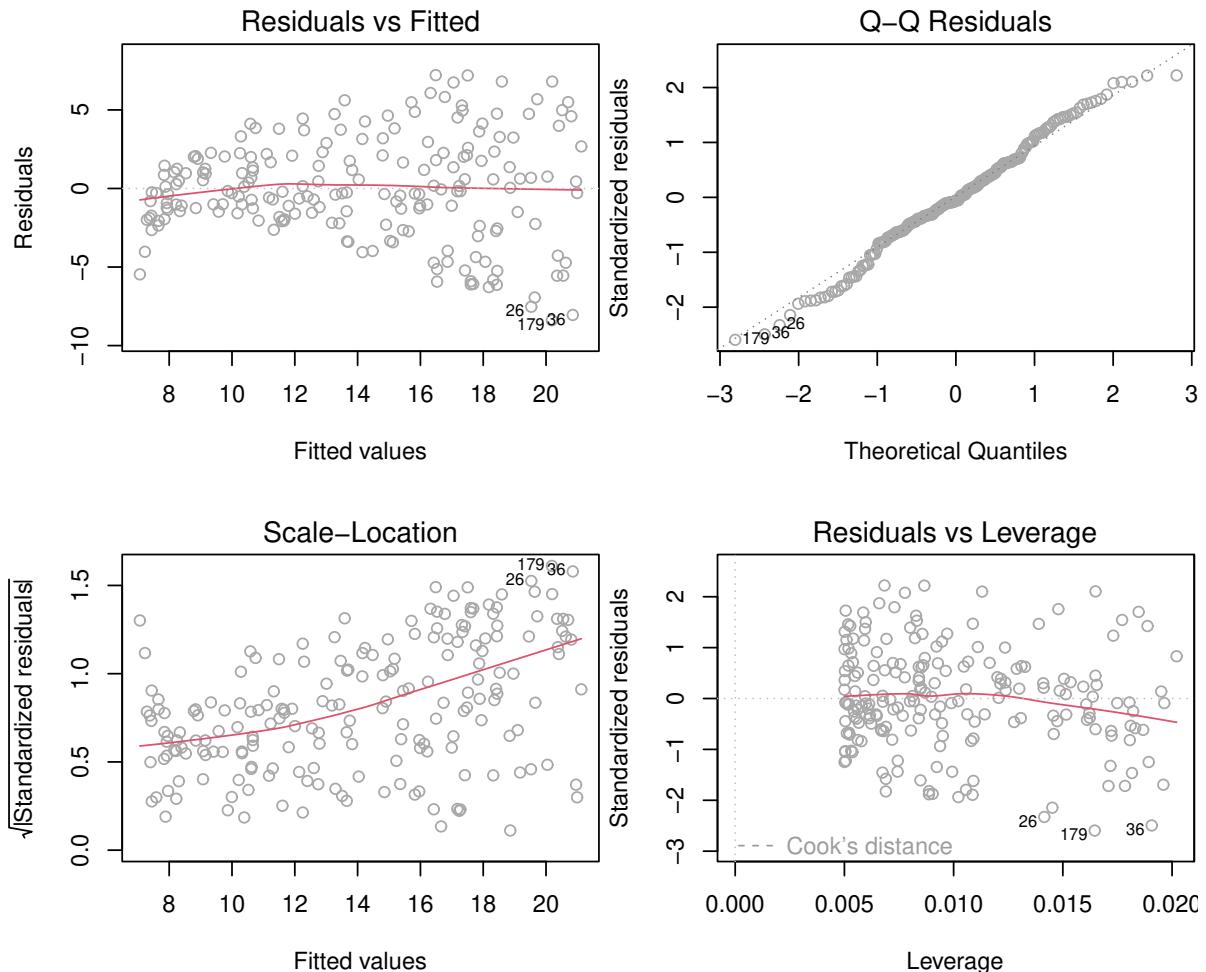
```

Advertising <- read.csv('./Daten/Advertising.csv')
summary(lm(sales ~ TV + radio + newspaper,
           data=Advertising))

##

```

## Appendix B. R-Code

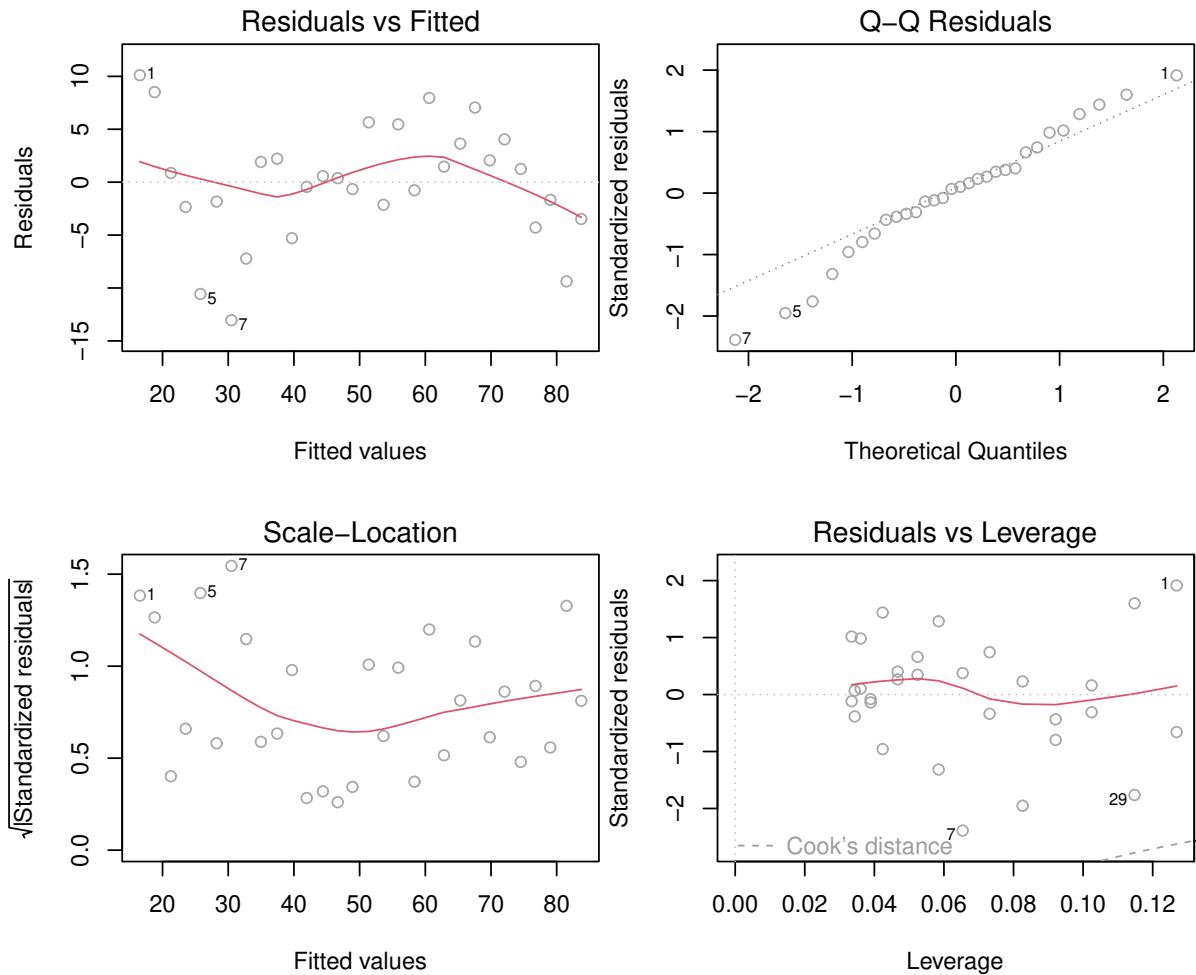


**Figure B.16.:** Residual plots for the [Advertising](#) data set.

```

## Call:
## lm(formula = sales ~ TV + radio + newspaper, data = Advertising)
##
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -8.828 -0.891  0.242  1.189  2.829 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 2.93889   0.31191   9.42   <2e-16 ***
## TV          0.04576   0.00139  32.81   <2e-16 ***
## radio        0.18853   0.00861  21.89   <2e-16 ***
## newspaper   -0.00104   0.00587  -0.18     0.86    
## ---
## 
```

## Appendix B. R-Code



**Figure B.17.:** Residual plots for the `Income` data set.

```
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.69 on 196 degrees of freedom
## Multiple R-squared:  0.897, Adjusted R-squared:  0.896
## F-statistic: 570 on 3 and 196 DF,  p-value: <2e-16
```

### Example 7.3.5

(to Python)

## Appendix B. R-Code

```
Advertising <- read.csv('./Daten/Advertising.csv')
predict(lm(sales ~ TV + radio, data = Advertising),
        interval = "confidence",
        data.frame(TV = 100, radio = 20))

##     fit lwr upr
## 1 11.3 11 11.5
```

([to Python](#))

```
Advertising <- read.csv('./Daten/Advertising.csv')
predict(lm(sales ~ TV + radio, data = Advertising),
        interval = "prediction",
        data.frame(TV = 100, radio = 20))

##     fit lwr upr
## 1 11.3 7.93 14.6
```

## Example 7.4.1

([to Python](#))

```
Advertising <- read.csv('./Daten/Advertising.csv')
anova(lm(sales ~ TV + radio, data = Advertising),
      lm(sales ~ TV + radio + newspaper, data = Advertising))

## Analysis of Variance Table
##
## Model 1: sales ~ TV + radio
## Model 2: sales ~ TV + radio + newspaper
##   Res.Df RSS Df Sum of Sq    F Pr(>F)
## 1     197 557
## 2     196 557  1     0.0887 0.03    0.86
```

## Example 7.4.3

([to Python](#))

## Appendix B. R-Code

```
Advertising <- read.csv('./Daten/Advertising.csv')
anova(lm(sales ~ radio + newspaper, data = Advertising),
      lm(sales ~ TV + radio + newspaper, data = Advertising))

## Analysis of Variance Table
##
## Model 1: sales ~ radio + newspaper
## Model 2: sales ~ TV + radio + newspaper
##   Res.Df   RSS Df Sum of Sq    F Pr(>F)
## 1     197 3615
## 2     196 557  1      3058 1076 <2e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

([to Python](#))

```
Advertising <- read.csv('./Daten/Advertising.csv')
drop1(lm(sales ~ TV + radio + newspaper,
          data=Advertising), test="F")

## Single term deletions
##
## Model:
## sales ~ TV + radio + newspaper
##           Df Sum of Sq  RSS AIC F value Pr(>F)
## <none>            557 213
## TV       1      3058 3615 585 1076.41 <2e-16 ***
## radio    1      1362 1919 458  479.33 <2e-16 ***
## newspaper 1        0 557 211    0.03   0.86
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

## Example 7.4.4

([to Python](#))

```
Credit <- read.csv('./Daten/Credit.csv')

pairs(~Balance + Age + Cards + Education + Income + Limit +
      Rating, Credit, pch = ".", col = "darkcyan")
```

## Appendix B. R-Code

### Example 7.4.5

([to Python](#))

```
Credit <- read.csv('./Daten/Credit.csv')
# Set contrast
Credit$Gender <- as.factor(Credit$Gender)
contrasts(Credit$Gender) <- contr.treatment(2, base=1)
contrasts(Credit$Gender)

##          2
##  Male   0
## Female 1

# Print a summary
model = lm(Balance ~ Gender, data = Credit)
round(summary(model)$coef, digits = 5)

##           Estimate Std. Error t value Pr(>|t|)
## (Intercept) 509.8       33.1  15.389   0.000
## Gender2     19.7       46.1   0.428   0.669
```

### Example 7.4.6

([to Python](#)).

```
Credit <- read.csv('./Daten/Credit.csv')
# Set another level as base:
Credit$Gender <- as.factor(Credit$Gender)
contrasts(Credit$Gender) <- contr.treatment(2, base=2)
contrasts(Credit$Gender)

##          1
##  Male   1
## Female 0

# Print a summary
model = lm(Balance ~ Gender, data = Credit)
round(summary(model)$coef, digits = 5)

##           Estimate Std. Error t value Pr(>|t|)
## (Intercept) 529.5       32.0  16.554   0.000
## Gender1    -19.7       46.1  -0.428   0.669
```

### Example 7.4.7

([to Python](#))

```
Credit <- read.csv('~/Daten/Credit.csv')
# Now use a -1/+1 coding scheme, instead of 0/+1
Credit$Gender <- as.factor(Credit$Gender)
contrasts(Credit$Gender) <- contr.sum(2)
contrasts(Credit$Gender)

##           [,1]
## Male      1
## Female   -1

# Print a summary
model = lm(Balance ~ Gender, data = Credit)
round(summary(model)$coef, digits = 5)

##           Estimate Std. Error t value Pr(>|t|)
## (Intercept) 519.67       23  22.569 0.000
## Gender1     -9.87       23  -0.428 0.669
```

### Example 7.4.8

([to Python](#))

```
Credit <- read.csv('~/Daten/Credit.csv')
# Set contrast
Credit$Ethnicity <- as.factor(Credit$Ethnicity)
contrasts(Credit$Ethnicity) <- contr.treatment(3, base=1)
contrasts(Credit$Ethnicity)

##           2 3
## African American 0 0
## Asian            1 0
## Caucasian        0 1

# Print a summary
model = lm(Balance ~ Ethnicity, data = Credit)
round(summary(model)$coef, digits = 5)
```

## Appendix B. R-Code

```
##             Estimate Std. Error t value Pr(>|t|) 
## (Intercept) 531.0      46.3   11.464 0.0000 
## Ethnicity2 -18.7      65.0   -0.287 0.7740 
## Ethnicity3 -12.5      56.7   -0.221 0.8260
```

### Example 7.4.11

([to Python](#))

```
Advertising <- read.csv('./Daten/Advertising.csv')
summary(lm(sales ~ TV + radio + TV*radio, data=Advertising))

## 
## Call:
## lm(formula = sales ~ TV + radio + TV * radio, data = Advertising)
## 

## Residuals:
##     Min      1Q  Median      3Q     Max 
## -6.337 -0.403  0.183  0.595  1.525 

## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 6.7502202  0.2478714  27.23   <2e-16 ***
## TV          0.0191011  0.0015041  12.70   <2e-16 ***
## radio       0.0288603  0.00089053   3.24    0.0014 **  
## TV:radio    0.0010865  0.0000524  20.73   <2e-16 *** 
## --- 
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 

## 
## Residual standard error: 0.944 on 196 degrees of freedom
## Multiple R-squared:  0.968, Adjusted R-squared:  0.967 
## F-statistic: 1.96e+03 on 3 and 196 DF,  p-value: <2e-16
```

### Example 7.4.13

([to Python](#))

## Appendix B. R-Code

```
library(readr)
Auto <- read.csv("./Daten/Auto.csv")
par(mfrow = c(1, 2))
# Left figure, including scatter data and fitted line
plot(mpg ~ horsepower, col = "darkgrey", ylab = "Miles per gallon",
      data = Auto)
abline(lm(mpg ~ horsepower, data = Auto), col = "blue")
# Right figure, Residuals vs fitted value.
plot(lm(mpg ~ horsepower, data = Auto), col = "darkgrey",
      which = 1)
```

([to Python](#))

```
Auto <- read.csv("./Daten/Auto.csv")
round(summary(lm(mpg ~ horsepower + I(horsepower^2), data = Auto))$coef, digits = 5)

##                               Estimate Std. Error t value Pr(>|t|)
## (Intercept)        24.18254    0.76518   31.6     0
## horsepower       -0.19813    0.01323  -15.0     0
## I(horsepower^2)  0.00052    0.00005   10.1     0

summary(lm(mpg ~ horsepower + I(horsepower^2),
           data=Auto))$r.squared

## [1] 0.688
```

([to Python](#))

```
Auto <- read.csv("./Daten/Auto.csv")
anova(lm(mpg ~ horsepower, data=Auto),
      lm(mpg ~ horsepower + I(horsepower^2), data=Auto))

## Analysis of Variance Table
##
## Model 1: mpg ~ horsepower
## Model 2: mpg ~ horsepower + I(horsepower^2)
##   Res.Df  RSS Df Sum of Sq   F Pr(>F)
## 1     390 1695
## 2     389 1344  1        351 102 <2e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

## Appendix B. R-Code

### Example 7.4.14

(to Python)

```
Advertising <- read.csv("./Daten/Advertising.csv")
par(mfrow=c(1,2))
plot(lm(sales ~ TV + radio, data = Advertising),
     which = 1, col = "darkgrey")
plot(lm(sales ~ TV + radio + TV * radio, data = Advertising),
     which = 1, col = "darkgrey")
```

(to Python)

```
Advertising <- read.csv("./Daten/Advertising.csv")
par(mfrow=c(1,2))
plot(lm(sales ~ TV + radio, data = Advertising),
     which = 3, col = "darkgrey")
plot(lm(sales ~ TV + radio + TV * radio, data = Advertising),
     which = 3, col = "darkgrey")
```

(to Python)

```
Advertising <- read.csv("./Daten/Advertising.csv")
par(mfrow=c(1,2))
plot(lm(sales ~ TV + radio, data = Advertising),
     which = 5, col = "darkgrey", cook.levels = c(0.1, 0.5, 1))
#radio.R <- Advertising[-c(131,156),3]
#TV.R <- Advertising[-c(131,156),2]
#sales.R <- Advertising[-c(131,156),5]
plot(lm(sales ~ TV + radio + TV * radio,
        subset = -c(131, 156), data = Advertising),
     which = 5, col = "darkgrey", cook.levels = c(0.1, 0.5, 1))
```

### Example 7.4.17

(to Python)

```
Credit <- read.csv("./Daten/Credit.csv")
round(cor(Credit[, -c(1, 8:11)]), digit = 3)
```

## Appendix B. R-Code

```
##           Income  Limit Rating  Cards   Age Education
## Income      1.000  0.792  0.791 -0.018  0.175    -0.028
## Limit        0.792  1.000  0.997  0.010  0.101    -0.024
## Rating       0.791  0.997  1.000  0.053  0.103    -0.030
## Cards        -0.018  0.010  0.053  1.000  0.043    -0.051
## Age          0.175  0.101  0.103  0.043  1.000     0.004
## Education   -0.028 -0.024 -0.030 -0.051  0.004     1.000
## Balance       0.464  0.862  0.864  0.086  0.002    -0.008
##               Balance
## Income        0.464
## Limit         0.862
## Rating        0.864
## Cards         0.086
## Age           0.002
## Education    -0.008
## Balance       1.000
```

### Example 7.4.18

([to Python](#))

```
Credit <- read.csv("./Daten/Credit.csv")
library(car)
vif(lm(Balance ~ Age + Rating + Limit, data = Credit))

##      Age Rating  Limit
## 1.01 160.67 160.59

summary(lm(Balance ~ Age + Rating + Limit, data = Credit))$r.squared

## [1] 0.754
```

### Example 7.4.19

([to Python](#))

```
Credit <- read.csv("./Daten/Credit.csv")
library(car)
vif(lm(Balance ~ Age + Limit, data = Credit))
```

## Appendix B. R-Code

```
##    Age Limit
## 1.01  1.01

summary(lm(Balance ~ Age + Limit, data = Credit))$r.squared

## [1] 0.75
```

### Example 7.5

([to Python](#))

```
Advertising <- read.csv("./Daten/Advertising.csv")
summary(lm(sales ~ TV + radio + newspaper, data = Advertising))

##
## Call:
## lm(formula = sales ~ TV + radio + newspaper, data = Advertising)
##
## Residuals:
##     Min      1Q  Median      3Q     Max 
## -8.828 -0.891  0.242  1.189  2.829 
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 2.93889   0.31191   9.42   <2e-16 ***
## TV          0.04576   0.00139  32.81   <2e-16 ***
## radio       0.18853   0.00861  21.89   <2e-16 ***
## newspaper   -0.00104   0.00587  -0.18     0.86    
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 

## Residual standard error: 1.69 on 196 degrees of freedom
## Multiple R-squared:  0.897, Adjusted R-squared:  0.896 
## F-statistic: 570 on 3 and 196 DF,  p-value: <2e-16

mean(sales)

## [1] 14
```

([to Python](#))

## Appendix B. R-Code

```
Advertising <- read.csv("./Daten/Advertising.csv")
round(confint(lm(sales ~ TV + radio + newspaper,
                 data = Advertising)), digits = 3)

##                2.5 % 97.5 %
## (Intercept) 2.324  3.554
## TV          0.043  0.049
## radio        0.172  0.206
## newspaper   -0.013  0.011
```

([to Python](#))

```
Advertising <- read.csv("./Daten/Advertising.csv")
library(car)
round(vif(lm(sales ~ TV + radio + newspaper,
              data = Advertising)), digits = 3)

##           TV      radio newspaper
## 1.00       1.15      1.15
```

### Example 8.2.1

In [R](#) this can be done with the function `add1()`, that adds each predictor variable separately to the reference model. ([to Python](#))

```
Credit <- read.csv('./Daten/Credit.csv')
f.full <- lm(Balance ~ Income + Limit +
             Rating + Cards + Age + Education +
             Gender + Student + Married +
             Ethnicity, data=Credit)
f.empty <- lm(Balance ~ NULL, data=Credit)
add1(f.empty, scope=f.full)

## Single term additions
##
## Model:
## Balance ~ NULL
##               Df Sum of Sq      RSS    AIC
## <none>            84339912 4906
## Income      1  18131167 66208745 4811
## Limit       1  62624255 21715657 4365
## Rating      1  62904790 21435122 4360
```

## Appendix B. R-Code

```
## Cards      1    630416 83709496 4905
## Age        1     284 84339628 4908
## Education  1     5481 84334431 4908
## Gender     1    38892 84301020 4907
## Student    1    5658372 78681540 4880
## Married    1     2715 84337197 4908
## Ethnicity  2    18454 84321458 4909
```

We now add a further predictor variable to this model by first updating the reference model with the R function `update()`. Subsequently, `add1()` will help us to decide on the next predictor variable to be added to the model.

([to Python](#))

```
f.1 <- update(f.empty, . ~ . + Rating)
add1(f.1, scope = f.full)

## Single term additions
##
## Model:
## Balance ~ Rating
##             Df Sum of Sq      RSS   AIC
## <none>              21435122 4360
## Income      1   10902581 10532541 4077
## Limit       1     7960 21427162 4361
## Cards       1   138580 21296542 4359
## Age         1   649110 20786012 4349
## Education   1   27243 21407879 4361
## Gender      1    16065 21419057 4361
## Student     1   5735163 15699959 4237
## Married     1   118209 21316913 4359
## Ethnicity   2    51100 21384022 4363
```

([to Python](#))

```
f.2 <- update(f.1, . ~ . + Income)
add1(f.2, scope = f.full)

## Single term additions
##
## Model:
## Balance ~ Rating + Income
##             Df Sum of Sq      RSS   AIC
## <none>              10532541 4077
```

## Appendix B. R-Code

```
## Limit      1    94545 10437996 4076
## Cards      1    2094 10530447 4079
## Age        1    90286 10442255 4076
## Education   1    20819 10511722 4079
## Gender      1     948 10531593 4079
## Student     1    6305322 4227219 3714
## Married     1    95068 10437473 4076
## Ethnicity   2    67040 10465501 4079
```

The procedure we have followed in this example by iteratively running `update` and `add1` is rather awkward. Thanks to the R function `regsubsets` this procedure can be completed automatically ([to Python](#)).

([to Python](#))

```
Credit <- read.csv('./Daten/Credit.csv')
library(leaps)
Credit <- Credit[,-1]
reg <- regsubsets(Balance ~ ., data=Credit,
                  method="forward", nvmax=11)
reg.sum <- summary(reg)
reg.sum$which

##      (Intercept) Income Limit Rating Cards    Age
## 1        TRUE FALSE FALSE    TRUE FALSE FALSE
## 2        TRUE TRUE FALSE    TRUE FALSE FALSE
## 3        TRUE TRUE FALSE    TRUE FALSE FALSE
## 4        TRUE TRUE TRUE    TRUE FALSE FALSE
## 5        TRUE TRUE TRUE    TRUE  TRUE FALSE
## 6        TRUE TRUE TRUE    TRUE  TRUE  TRUE
## 7        TRUE TRUE TRUE    TRUE  TRUE  TRUE
## 8        TRUE TRUE TRUE    TRUE  TRUE  TRUE
## 9        TRUE TRUE TRUE    TRUE  TRUE  TRUE
## 10       TRUE TRUE TRUE    TRUE  TRUE  TRUE
## 11       TRUE TRUE TRUE    TRUE  TRUE  TRUE
##      Education GenderMale StudentYes MarriedYes
## 1        FALSE    FALSE    FALSE    FALSE
## 2        FALSE    FALSE    FALSE    FALSE
## 3        FALSE    FALSE    TRUE     FALSE
## 4        FALSE    FALSE    TRUE     FALSE
## 5        FALSE    FALSE    TRUE     FALSE
## 6        FALSE    FALSE    TRUE     FALSE
## 7        FALSE    TRUE     TRUE     FALSE
## 8        FALSE    TRUE     TRUE     FALSE
## 9        FALSE    TRUE     TRUE     TRUE
```

## Appendix B. R-Code

```
## 10      FALSE      TRUE      TRUE      TRUE
## 11      TRUE      TRUE      TRUE      TRUE
##   EthnicityAsian EthnicityCaucasian
## 1          FALSE      FALSE
## 2          FALSE      FALSE
## 3          FALSE      FALSE
## 4          FALSE      FALSE
## 5          FALSE      FALSE
## 6          FALSE      FALSE
## 7          FALSE      FALSE
## 8          TRUE      FALSE
## 9          TRUE      FALSE
## 10         TRUE      TRUE
## 11         TRUE      TRUE
```

There are in total 11 models that are listed; wherever in the R-output **TRUE** appears, indicates that this predictor is included in the corresponding model.

### Example 8.2.2

([to Python](#))

```
Credit <- read.csv('./Daten/Credit.csv')
f.full <- lm(Balance ~ Income + Limit + Rating +
                  Cards + Age + Education + Gender +
                  Student + Married + Ethnicity, data=Credit)
f.empty <- lm(Balance ~ NULL, data=Credit)
drop1(f.full, scope=f.full)

## Single term deletions
##
## Model:
## Balance ~ Income + Limit + Rating + Cards + Age + Education +
##           Gender + Student + Married + Ethnicity
##             Df Sum of Sq    RSS   AIC
## <none>            3786730 3686
## Income     1  10831162 14617892 4225
## Limit      1    331050  4117780 3718
## Rating     1     52314  3839044 3690
## Cards      1    162702  3949432 3701
## Age        1     42558  3829288 3689
## Education   1      4615  3791345 3685
## Gender      1     11269  3798000 3685
## Student     1    6326012 10112742 4077
```

## Appendix B. R-Code

```
## Married      1       6619  3793349 3685
## Ethnicity   2      14084  3800814 3684
```

We now remove another variable from this model. To do so we first need to update the reference model which is done in **R** with the function `update()`. Subsequently we run again the function `drop1()`:

([to Python](#))

```
f.9 <- update(f.full, . ~ . - Education)
drop1(f.9, scope = f.9)

## Single term deletions
##
## Model:
## Balance ~ Income + Limit + Rating + Cards + Age + Gender + Student +
##           Married + Ethnicity
##             Df Sum of Sq      RSS    AIC
## <none>            3791345 3685
## Income     1  10826551 14617896 4223
## Limit      1   326990  4118335 3716
## Rating     1    55071  3846417 3688
## Cards      1   162683  3954029 3700
## Age        1    43114  3834460 3687
## Gender     1    11103  3802448 3684
## Student    1   6340464 10131810 4076
## Married    1     7385  3798730 3683
## Ethnicity  2    14226  3805572 3682
```

The entire backward stepwise selection procedure can be simplified with the **R** function `regsubsets()`. ([to Python](#))

```
Credit <- read.csv('./Daten/Credit.csv')
library(leaps)
Credit <- Credit[,-1]
reg <- regsubsets(Balance ~. , data=Credit,
                  method="backward", nvmax=11)
reg.sum <- summary(reg)
reg.sum$which

##      (Intercept) Income Limit Rating Cards    Age
## 1          TRUE  FALSE  TRUE  FALSE FALSE FALSE
## 2          TRUE   TRUE  TRUE  FALSE FALSE FALSE
## 3          TRUE   TRUE  TRUE  FALSE FALSE FALSE
```

## Appendix B. R-Code

```
## 4      TRUE  TRUE  TRUE  FALSE  TRUE FALSE
## 5      TRUE  TRUE  TRUE  TRUE  TRUE FALSE
## 6      TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## 7      TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## 8      TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## 9      TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## 10     TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## 11     TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##   Education GenderMale StudentYes MarriedYes
## 1      FALSE  FALSE  FALSE  FALSE
## 2      FALSE  FALSE  FALSE  FALSE
## 3      FALSE  FALSE  TRUE  FALSE
## 4      FALSE  FALSE  TRUE  FALSE
## 5      FALSE  FALSE  TRUE  FALSE
## 6      FALSE  FALSE  TRUE  FALSE
## 7      FALSE  TRUE  TRUE  FALSE
## 8      FALSE  TRUE  TRUE  FALSE
## 9      FALSE  TRUE  TRUE  TRUE
## 10     FALSE  TRUE  TRUE  TRUE
## 11     TRUE  TRUE  TRUE  TRUE
##   EthnicityAsian EthnicityCaucasian
## 1      FALSE  FALSE
## 2      FALSE  FALSE
## 3      FALSE  FALSE
## 4      FALSE  FALSE
## 5      FALSE  FALSE
## 6      FALSE  FALSE
## 7      FALSE  FALSE
## 8      TRUE  FALSE
## 9      TRUE  FALSE
## 10     TRUE  TRUE
## 11     TRUE  TRUE
```

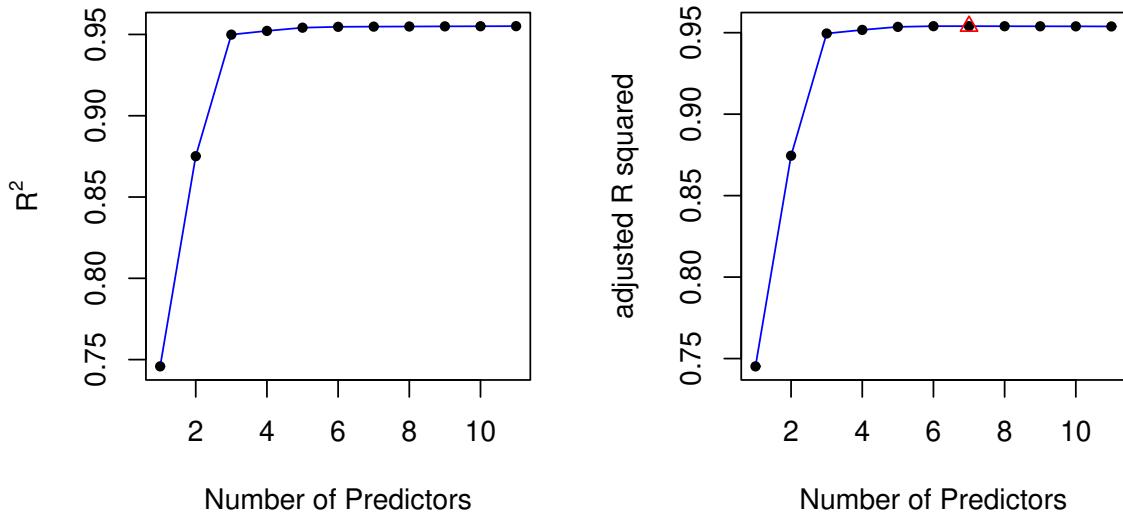
### Example 8.2.3

([to Python](#))

```
Credit <- read.csv('./Daten/Credit.csv')
Credit <- Credit[,-1]
library(leaps)
reg <- regsubsets(Balance ~ ., data=Credit,
                     method="forward", nvmax=11)
reg.sum <- summary(reg)
round(reg.sum$rsq, 5)
```

## Appendix B. R-Code

```
Credit <- read.csv('./Daten/Credit.csv')
Credit <- Credit[,-1]
library(leaps)
reg <- regsubsets(Balance ~ . , data=Credit,
                   method="forward", nvmax=11)
reg.sum <- summary(reg)
par(mfrow=c(1,2))
plot(reg.sum$rsq, type="l", col="blue",
     xlab="Number of Predictors",
     ylab=expression(R^2))
points(reg.sum$rsq, pch=20)
plot(reg.sum$adjr2, type="l", col="blue",
     xlab="Number of Predictors",
     ylab="adjusted R squared")
points(reg.sum$adjr2, pch=20)
points(which.max(reg.sum$adjr2),
       reg.sum$adjr2[which.max(reg.sum$adjr2)],
       col="red", pch=2)
```



**Figure B.18.:** Adjusted  $R^2$  is shown for the best models of each size for the `Credit` data set. The models were produced by stepwise forward selection. The adjusted  $R^2$  curve gets rather flat after four variables are included and reaches the maximum for seven predictors.

```
## [1] 0.746 0.875 0.950 0.952 0.954 0.955 0.955 0.955
## [9] 0.955 0.955 0.955
```

## Appendix B. R-Code

This is not the case for the values of the adjusted  $R^2$ . As the [R](#)-output reveals, the adjusted  $R^2$  reaches a maximum for seven predictors, after that the values start decreasing.

```
Credit <- read.csv('./Daten/Credit.csv')
Credit <- Credit[,-1]
library(leaps)
reg <- regsubsets(Balance ~ . , data=Credit,
                     method="forward", nvmax=11)
reg.sum <- summary(reg)
round(reg.sum$adjr2, 5)

## [1] 0.745 0.874 0.950 0.952 0.954 0.954 0.954 0.954
## [9] 0.954 0.954 0.954

which.max(reg.sum$adjr2)

## [1] 7
```

### Example 8.2.4

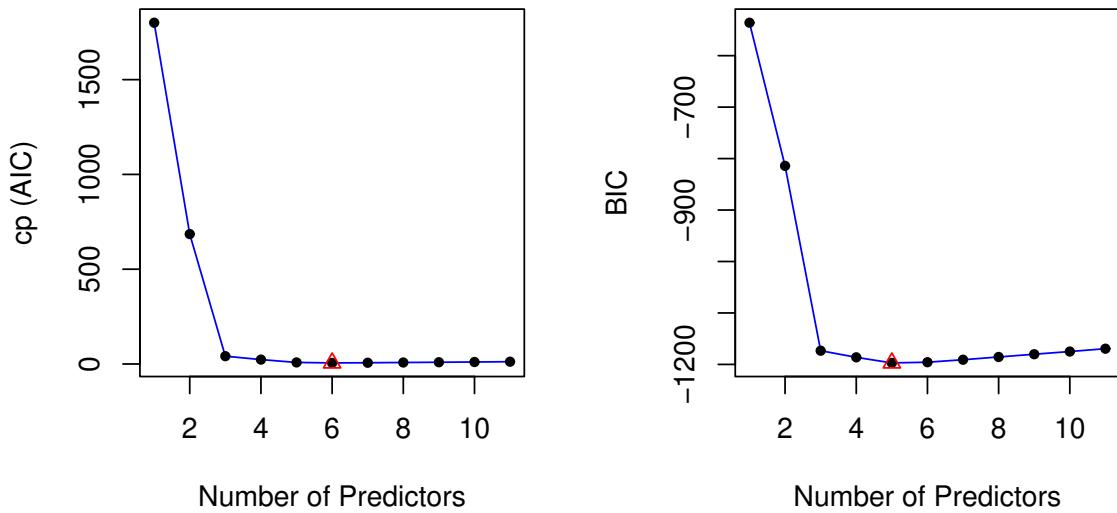
The left-hand panel of Figure 8.2 displays the  $C_p$  which is, up to a constant factor, identical with the AIC.

[\(to Python\)](#)

```
Credit <- read.csv('./Daten/Credit.csv')
Credit <- Credit[,-1]
library(leaps)
reg <- regsubsets(Balance ~ . , data=Credit,
                     method="forward", nvmax=11)
reg.sum <- summary(reg)
which.min(reg.sum$cp)

## [1] 6
```

We may wonder whether there is an [R](#) function that executes automatically the entire procedure of selecting the best model according to the AIC criterion. Indeed, the function `step()` does so.



**Figure B.19.:** Left:  $C_p$  is plotted for the best models of each size (produced by forward stepwise selection) for the `Credit` data set. Since the AIC is proportional to  $C_p$ , the AIC curve looks identical. Right: The BIC is plotted for the best models of each size (produced by forward stepwise selection) for the `Credit` data set.

### Example 8.2.5

If we want to find the best among all models produced by stepwise forward selection, that is, if we want to identify the most appropriate number of predictors on the basis of the AIC, then we can run `step()`.

([to Python](#))

```
Credit <- read.csv('./Daten/Credit.csv')
Credit <- Credit[,-1]
f.full <- lm(Balance ~ Income + Limit + Rating +
              Cards + Age + Education + Gender +
              Student + Married + Ethnicity, data=Credit)
f.empty <- lm(Balance ~ NULL, data=Credit)
step(f.empty, direction="forward",
      scope=list(lower=f.empty, upper=f.full))

## Start:  AIC=4906
## Balance ~ NULL
##
##          Df  Sum of Sq      RSS    AIC
##
```

## Appendix B. R-Code

```

## + Rating      1  62904790 21435122 4360
## + Limit       1  62624255 21715657 4365
## + Income      1  18131167 66208745 4811
## + Student     1   5658372 78681540 4880
## + Cards       1   630416 83709496 4905
## <none>          84339912 4906
## + Gender      1   38892 84301020 4907
## + Education    1   5481 84334431 4908
## + Married     1   2715 84337197 4908
## + Age          1   284 84339628 4908
## + Ethnicity   2   18454 84321458 4909
##
## Step: AIC=4360
## Balance ~ Rating
##
##             Df Sum of Sq      RSS   AIC
## + Income      1  10902581 10532541 4077
## + Student     1   5735163 15699959 4237
## + Age          1   649110 20786012 4349
## + Cards        1   138580 21296542 4359
## + Married      1   118209 21316913 4359
## <none>          21435122 4360
## + Education    1   27243 21407879 4361
## + Gender        1   16065 21419057 4361
## + Limit         1    7960 21427162 4361
## + Ethnicity    2   51100 21384022 4363
##
## Step: AIC=4077
## Balance ~ Rating + Income
##
##             Df Sum of Sq      RSS   AIC
## + Student     1   6305322  4227219 3714
## + Married      1    95068 10437473 4076
## + Limit         1   94545 10437996 4076
## + Age          1   90286 10442255 4076
## <none>          10532541 4077
## + Education    1   20819 10511722 4079
## + Ethnicity    2   67040 10465501 4079
## + Cards         1   2094 10530447 4079
## + Gender        1    948 10531593 4079
##
## Step: AIC=3714
## Balance ~ Rating + Income + Student
##
##             Df Sum of Sq      RSS   AIC
## + Limit        1   194718  4032502 3697

```

## Appendix B. R-Code

```

## + Age      1    44620 4182600 3712
## <none>          4227219 3714
## + Married   1    13083 4214137 3715
## + Gender    1    12168 4215051 3715
## + Cards     1    10608 4216611 3715
## + Education 1    1400  4225820 3716
## + Ethnicity 2    22322 4204897 3716
##
## Step: AIC=3697
## Balance ~ Rating + Income + Student + Limit
##
##             Df Sum of Sq      RSS   AIC
## + Cards      1  166410 3866091 3683
## + Age        1  37952  3994549 3696
## <none>          4032502 3697
## + Gender     1  13345  4019157 3698
## + Married    1   6660  4025842 3699
## + Education  1   5795  4026707 3699
## + Ethnicity  2  17704  4014797 3700
##
## Step: AIC=3683
## Balance ~ Rating + Income + Student + Limit + Cards
##
##             Df Sum of Sq      RSS   AIC
## + Age        1  44472 3821620 3680
## <none>          3866091 3683
## + Gender     1  11350  3854741 3683
## + Education  1   5672  3860419 3684
## + Married    1   3121  3862970 3684
## + Ethnicity  2  14756  3851335 3685
##
## Step: AIC=3680
## Balance ~ Rating + Income + Student + Limit + Cards + Age
##
##             Df Sum of Sq      RSS   AIC
## <none>          3821620 3680
## + Gender     1  10861  3810759 3681
## + Married    1   5451  3816169 3681
## + Education  1   5242  3816378 3681
## + Ethnicity  2  11517  3810102 3683
##
## Call:
## lm(formula = Balance ~ Rating + Income + Student + Limit + Cards +
##     Age, data = Credit)
##
## Coefficients:

```

## Appendix B. R-Code

```

## (Intercept)      Rating      Income StudentYes
## -493.734       1.091      -7.795     425.610
## Limit          Cards       Age
## 0.194         18.212     -0.624

```

The selected model thus has an AIC value of 3679.89. This relatively detailed R-output contains each step of the selection procedure. If we are simply interested in the result, then we may set the argument `trace=0` in the `step()` function.

```

Credit <- read.csv('./Daten/Credit.csv')
Credit <- Credit[,-1]
f.full <- lm(Balance ~ Income + Limit +
             Rating + Cards + Age + Education +
             Gender + Student + Married + Ethnicity,
             data=Credit)
f.empty <- lm(Balance~NULL, data=Credit)
step(f.empty, direction="forward",
      scope=list(lower=f.empty, upper=f.full),
      trace=0)

##
## Call:
## lm(formula = Balance ~ Rating + Income + Student + Limit + Cards +
##     Age, data = Credit)
##
## Coefficients:
## (Intercept)      Rating      Income StudentYes
## -493.734       1.091      -7.795     425.610
## Limit          Cards       Age
## 0.194         18.212     -0.624

```

## Example 8.2.6

```

Credit <- read.csv('./Daten/Credit.csv')
Credit <- Credit[,-1]
library(leaps)
reg <- regsubsets(Balance ~ . , data=Credit,
                   method="forward", nvmax=11)
reg.sum <- summary(reg)
which.min(reg.sum$bic)

## [1] 5

```

## Appendix B. R-Code

If we now want to select the best model on the basis of the BIC criterion, then we need to specify the function arguments of `step()` as follows: `k=log(n)` instead of the default argument `k=2` used for the AIC.

([to Python](#))

```
Credit <- read.csv('~/Daten/Credit.csv')
Credit <- Credit[,-1]
library(MASS)
f.full <- lm(Balance ~ Income + Limit + Rating + Cards + Age +
             Education + Gender + Student + Married +
             Ethnicity, data=Credit)
f.empty <- lm(Balance~NULL, data=Credit)
step(f.empty, direction="forward",
      scope=list(lower=f.empty, upper=f.full),
      trace=0, k=log(nrow(Credit)))

##
## Call:
## lm(formula = Balance ~ Rating + Income + Student + Limit + Cards,
##      data = Credit)
##
## Coefficients:
## (Intercept)      Rating      Income  StudentYes
## -526.156       1.088      -7.875      426.850
## Limit          Cards
## 0.194         17.852
```

If we want to identify the best model by *backward stepwise selection*, then we set the argument `method = "backward"` in `step()`:

([to Python](#))

```
Credit <- read.csv('~/Daten/Credit.csv')
Credit <- Credit[,-1]
library(MASS)
f.full <- lm(Balance ~ Income + Limit + Rating + Cards + Age +
             Education + Gender + Student + Married +
             Ethnicity, data=Credit)
f.empty <- lm(Balance ~ NULL, data=Credit)
step(f.full, direction="backward",
      scope=list(lower=f.empty, upper=f.full),
      trace=0, k=log(nrow(Credit)))

##
## Call:
```

## Appendix B. R-Code

```
## lm(formula = Balance ~ Income + Limit + Cards + Student, data = Credit)
##
## Coefficients:
## (Intercept)      Income       Limit       Cards
## -499.727      -7.839       0.267      23.175
## StudentYes
##        429.606
```

Now, we end up with four predictor variables:

$$\text{balance} = \beta_0 + \beta_1 \cdot \text{income} + \beta_2 \cdot \text{limit} + \beta_3 \cdot \text{cards} + \beta_4 \cdot \text{student} + \varepsilon$$

The predictor variable **rating** has been dropped.

If we want to run a hybrid approach consisting of forward and backward stepwise selection, then we use `method = "both"` in the `step()` function. This approach is likely to yield the best overall model we can find with least squares and limited computational resources.

```
Credit <- read.csv('./Daten/Credit.csv')
Credit <- Credit[,-1]
library(MASS)
f.full <- lm(Balance ~ Income + Limit + Rating + Cards +
               Age + Education + Gender + Student + Married +
               Ethnicity, data=Credit)
f.empty <- lm(Balance ~ NULL, data=Credit)
step(f.empty, direction="both",
      scope=list(lower=f.empty, upper=f.full),
      trace=0, k=log(nrow(Credit)))

##
## Call:
## lm(formula = Balance ~ Income + Student + Limit + Cards, data = Credit)
##
## Coefficients:
## (Intercept)      Income     StudentYes       Limit
## -499.727      -7.839      429.606       0.267
## Cards
##        23.175
```

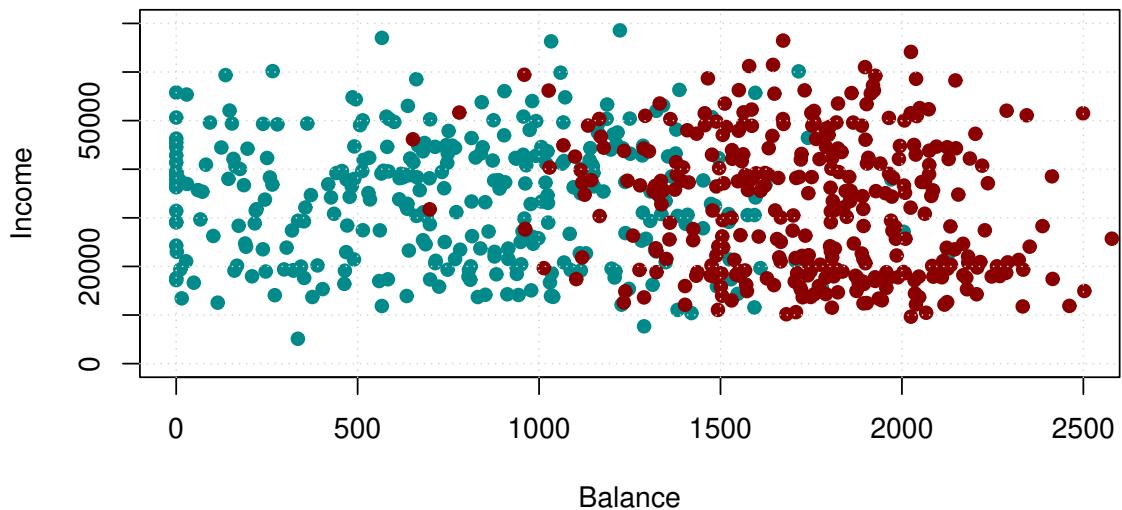
We end up in this case as well with four predictor variables:

$$\text{balance} = \beta_0 + \beta_1 \cdot \text{income} + \beta_2 \cdot \text{limit} + \beta_3 \cdot \text{cards} + \beta_4 \cdot \text{student} + \varepsilon$$

### Example 9.0.3

([to Python](#))

```
# ISLR contains the Default data
library(ISLR)
# scales contains the `alpha` function for transparency
library(scales)
# indices of people who did not default
cond <- which(Default[,1]=="No")
# randomly draw 333 among all people with default No
i <- sample(cond, 333)
plot(income[i] ~ balance[i], col = alpha("darkcyan", 0.5), xlim=c(0,2500),
      ylim=c(0,70000), xlab="Balance", ylab="Income", data=Default, pch=19)
# indices of people who defaulted
i <- which(Default[,1]=="Yes")
points(income[i] ~ balance[i], col = alpha("darkred", 0.5), pch=19,
       xlab="", ylab="", data=Default)
grid()
```

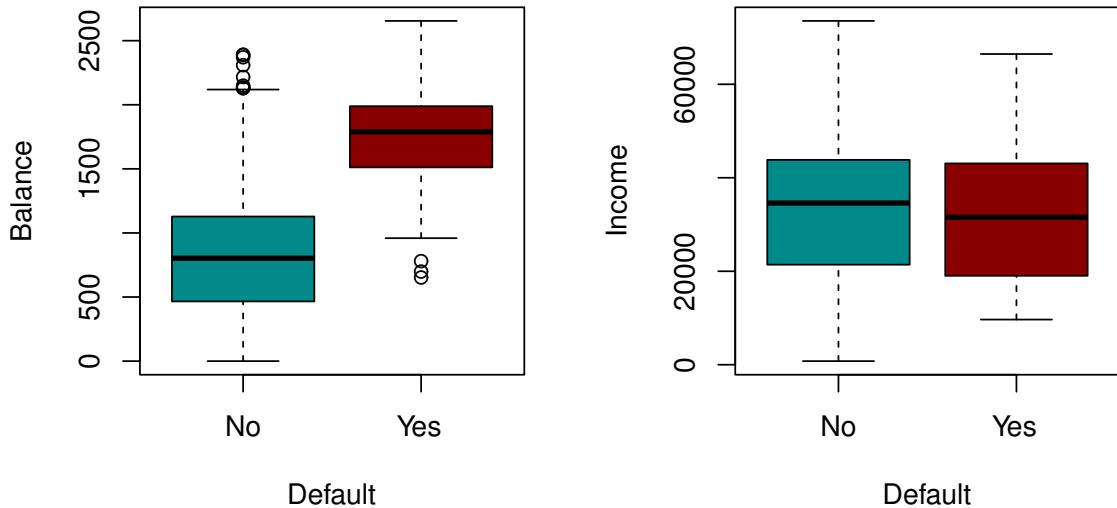


**Figure B.20.:** Scatter plot of `income` versus `balance` from the `default` data set. Red corresponds to `default=Yes`, and green to `default>No`.

([to Python](#))

## Appendix B. R-Code

```
par(mfrow = c(1, 2))
boxplot(balance ~ default, col = c("darkcyan", "darkred"),
        xlab = "Default", ylab = "Balance", data = Default)
boxplot(income ~ default, col = c("darkcyan", "darkred"),
        xlab = "Default", ylab = "Income", data = Default)
```



**Figure B.21.:** Boxplots for the sample data set **Default**: **balance** (left) and **income** (right) for the classes **default=Yes** and **default>No**.

### Example 10.1.1

([to Python](#))

```
# indices of people who did not default
cond <- which(Default[,1]=="No")
# randomly draw 333 people among people with default No
i <- sample(cond, 333)
plot(rep(0, length(i)) ~ balance[i], col = "darkcyan",
      xlim = c(0, 2500), ylim = c(0, 1),
      xlab = "Balance", ylab = "Probability for Default",
      pch = 20, cex = 0.7, data = Default)

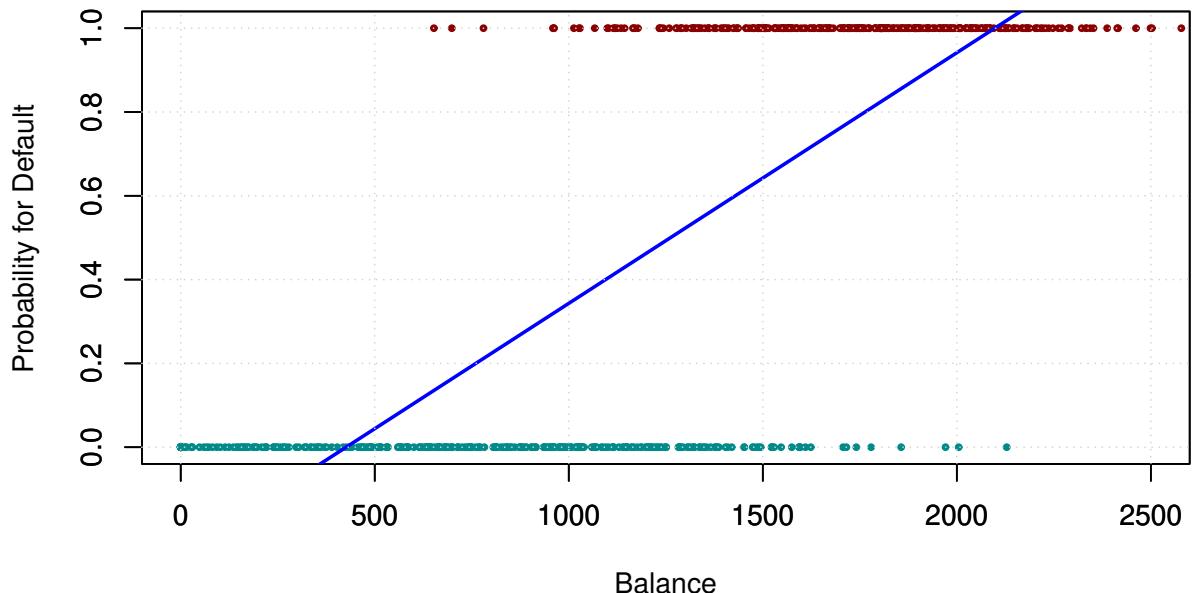
par(new = T)
```

## Appendix B. R-Code

```
j <- which(Default[, 1] == "Yes")
plot(rep(1, length(j)) ~ balance[j], col = "darkred",
      xlim = c(0, 2500), ylim = c(0, 1),
      xlab = "", ylab = "",
      pch = 20, cex = 0.7, data = Default)

regr.mod <- lm(
  as.numeric(Default[c(i, j), 1]) - 1 ~ balance[c(i, j)],
  data = Default)
beta0 <- regr.mod$coefficients[1]
beta1 <- regr.mod$coefficients[2]
abline(beta0,beta1,col="blue",lwd=2)
```

```
## [1] 333
```



**Figure B.22.:** A linear regression model  $p(X) = \beta_0 + \beta_1 X$ , where  $X$  is the predictor variable `balance`. The model was fitted to the `Default` data set and is plotted as a straight line. The dots represent the measured values, where red corresponds to `default = Yes` and green to `default = No`.

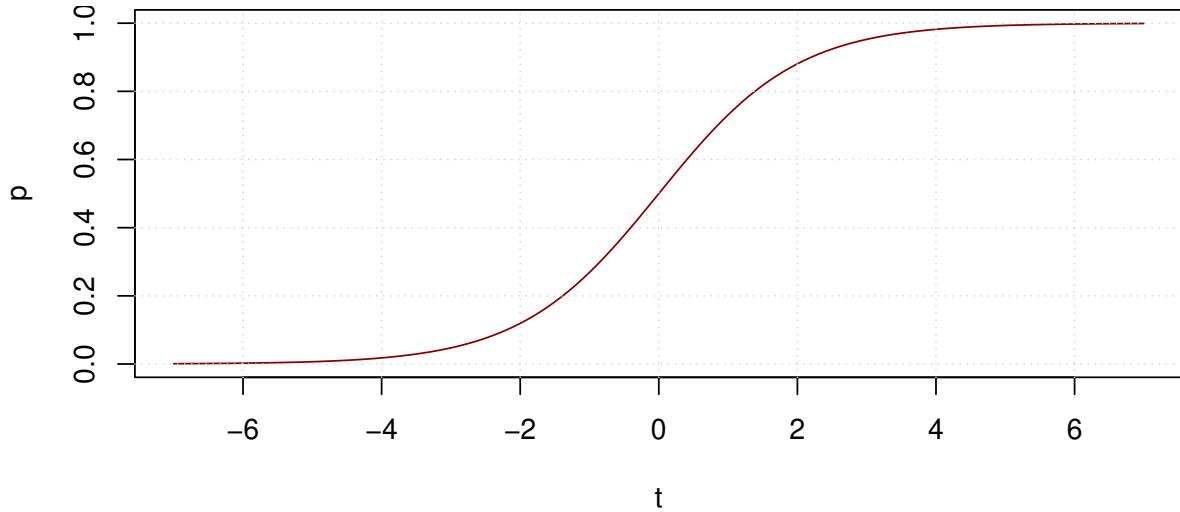
The graph of the logistic function is depicted in Figure 10.2. [\(to Python\)](#)

### Example 10.1.2

[\(to Python\)](#)

## Appendix B. R-Code

```
t <- seq(-7, 7, by = 0.1)
p <- exp(t)/(1 + exp(t))
plot(t, p, type = "l", xlab = "t", ylab = "p", col = "darkred")
grid()
```



**Figure B.23.:** The graph of the logistic function. The real line is transformed to the interval  $[0,1]$ .

```
## Example Logistic Regression Example 1.2
balance <- Default[, 3]

# indices of people who did not default
cond <- which(Default[, 1] == "No")
# randomly draw 333 people among people with default No
i <- sample(cond, 333)

plot(rep(0, length(i)) ~ balance[i], col = "darkcyan",
      xlim = c(0, 2500), ylim = c(0, 1),
      xlab = "Balance", ylab = "Probability for Default",
      pch = 20, cex = 0.7, data = Default)

par(new = T)

j <- which(Default[, 1] == "Yes")
plot(rep(1, length(j)) ~ balance[j], col = "darkred",
      xlim = c(0, 2500), ylim = c(0, 1),
      xlab = "", ylab = "",
      pch = 20, cex = 0.7, data = Default)

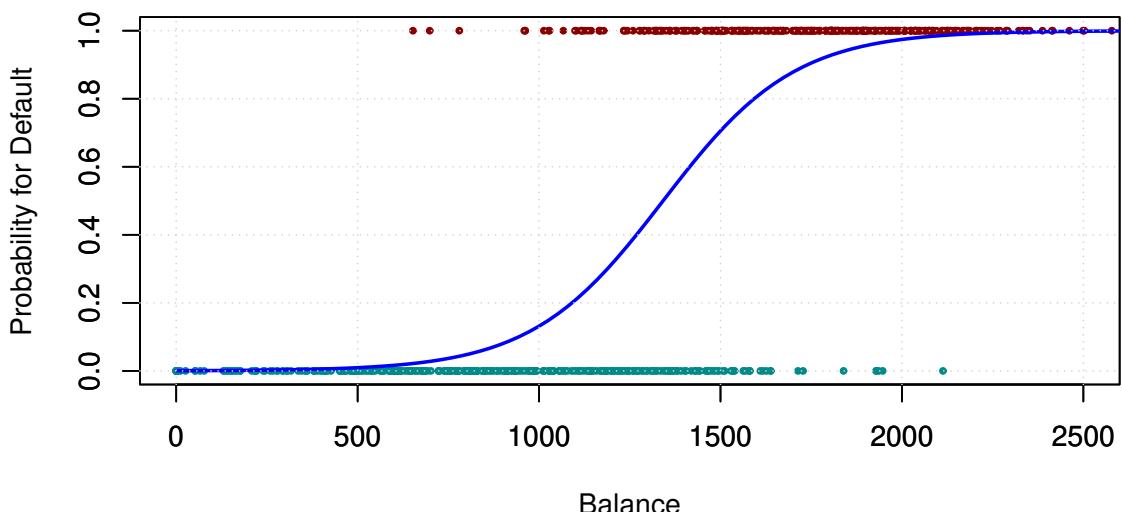
# Estimation of coefficients beta_0 und beta_0 and
# plotting of logistic regression curve
glm.fit <- glm(
```

## Appendix B. R-Code

```
(as.numeric(Default[c(i, j), 1]) - 1) ~ balance[c(i, j)],
family = binomial, data = Default)
beta0 <- summary(glm.fit)$coef[1]
beta1 <- summary(glm.fit)$coef[2]
eq = function(x) {
  (exp(beta0 + beta1 * x) / (1 + exp(beta0 + beta1 * x)))
}
curve(eq, 0, 2700, add = T, col = "blue", lwd = 2)
grid()
```

### Remarks:

- i. The syntax of `glm()` is similar to that of `lm()`, except that we must pass in the argument `family=binomial` in order to tell R to run a logistic regression rather than some other type of generalized model. ♦



**Figure B.24.:** A logistic regression function (blue) is fitted to the data set `Default`. Note that only a random sample of the `default = No` cases is used.

### Example 10.2.2

The R-output below shows the estimates for the parameters in the logistic regression model. The formula `default~balance` defines response and predictor variables. The parameter `family=binomial` indicates that `default` is a binary vari-

## Appendix B. R-Code

able and that the probability for **default=Yes** is modelled by logistic regression.  
[\(to Python\)](#)

```
glm.fit <- glm(default ~ balance, family = binomial, data = Default)
summary(glm.fit)$coef

##                               Estimate Std. Error z value   Pr(>|z|)
## (Intercept) -10.651331  0.3611574 -29.49 3.623e-191
## balance      0.005499  0.0002204   24.95 1.977e-137
```

[\(to Python\)](#)

```
glm.fit <- glm(default ~ balance, family = binomial, data = Default)
confint(glm.fit)

##                         2.5 %    97.5 %
## (Intercept) -11.383289 -9.966565
## balance      0.005079  0.005943
```

### Example 10.3.1

[\(to Python\)](#)

```
glm.fit <- glm(default ~ balance, family = binomial, data = Default)
# predict balance=1000:
predict(glm.fit, type = "response",
          newdata = data.frame(balance = 1000))

##           1
## 0.005752
```

[\(to Python\)](#)

```
# predict balance=2000:
predict(glm.fit, type = "response",
          newdata = data.frame(balance = 2000))

##           1
## 0.5858
```

### Example 10.3.2

([to Python](#))

```
# predict the class probabilities for the training data
pred.prob <- predict(glm.fit, type = "response")

# predict class by thresholding
pred.class <- as.integer(pred.prob > 0.5)

# compute classification error
true.class <- as.integer(Default$default == "Yes")
err.train <- mean(abs(pred.class - true.class))
err.train

## [1] 0.0275
```

### Example 10.3.3

([to Python](#))

```
cm = table(pred.class, true.class, dnn = c("predicted",
                                           "true"))
addmargins(cm)

##          true
## predicted   0    1  Sum
##       0 9625 233 9858
##       1    42 100 142
##     Sum 9667 333 10000
```

### Example 10.3.8

We can compute the F1 score by means of the R-package `MLmetrics` ([to Python](#)):

```
library(MLmetrics)
F1_Score(y_pred = pred.class, y_true = true.class, positive = "1")

## [1] 0.4211
```

### Example 10.3.9

If we consider the case **default=No** as positive, then the F1 score changes to  
[\(to Python\)](#)

```
library(MLmetrics)
F1_Score(y_pred = pred.class, y_true = true.class, positive = "0")

## [1] 0.9859
```

### Example 10.3.10

[\(to Python\)](#)

```
# resample
idx.yes = which(Default$default=="Yes")
idx.no  = sample(which(Default$default=="No"),
                 replace = F, size = 333)
idx     = c(idx.yes, idx.no)

Default.ds = Default[idx,]
glm.fit.ds <- glm(default~balance,family=binomial,data=Default.ds)

# predict the class probalites for the training data
pred.prob.ds <- predict(glm.fit.ds, type = "response")

# predict class by thresholding
pred.class.ds <- as.integer(pred.prob.ds > 0.5)

# compute training classification error
true.class.ds <- as.integer(Default.ds$default=="Yes")
err.train.ds <- mean(abs(pred.class.ds-true.class.ds))

# compute confusion matrix
cm = table(pred.class.ds, true.class.ds, dnn = c("predicted", "true"))
addmargins(cm)

##          true
## predicted   0   1 Sum
##       0   288  34 322
##       1    45 299 344
##     Sum  333 333 666
```

## Appendix B. R-Code

```
F1_Score(y_pred = pred.class.ds, y_true = true.class.ds, positive = "0")  
  
## [1] 0.8794  
  
F1_Score(y_pred = pred.class.ds, y_true = true.class.ds, positive = "1")  
  
## [1] 0.8833
```

### Example 10.4.1

The logistic model for the **Default** data set will now be evaluated with  $k$ -fold cross validation. We use the `boot`-package for computing the estimated error.

([to Python](#))

```
# classification error  
cost <- function(true.class, est.prob) {  
  mean((est.prob < 0.5) & true.class == 1 | (est.prob >  
    0.5) & true.class == 0)  
}  
  
# cross validated error  
cv.error <- cv.glm(glmfit = glm.fit.ds, data = Default.ds,  
  cost = cost, K = 5)  
cv.error$delta[1]  
  
## [1] 0.1186
```

The variable `cv.error` is 0.1186, which amounts to 88.1381% correctly classified samples.

### Example 10.5.1

([to Python](#))

```
# logistic regression fit on downsampled data set  
glm.fit <- glm(default~balance+income+student,  
  family=binomial, data=Default.ds)  
summary(glm.fit)$coef
```

## Appendix B. R-Code

```
##                               Estimate Std. Error z value Pr(>|z|)  
## (Intercept) -7.33515028  0.8973291 -8.174 2.97e-16  
## balance      0.00576195  0.0004510 12.776 2.23e-37  
## income       -0.00000348  0.0000156 -0.223 8.24e-01  
## studentYes   -0.47741635  0.4456302 -1.071 2.84e-01  
  
# predict class by thresholding  
pred.class <- as.integer(predict(glm.fit, type = "response")) > 0.5  
  
# compute confusion matrix  
true.class <- as.integer(Default.ds$default=="Yes")  
cm = table(pred.class, true.class, dnn = c("predicted", "true"))  
addmargins(cm)  
  
##           true  
## predicted  0    1  Sum  
##       0    289  35 324  
##       1     44 298 342  
##           Sum 333 333 666  
  
# cross validated error  
cv.error <- cv.glm(glmfit=glm.fit, data = Default.ds,  
                    cost = cost, K = 5)  
cv.error$delta[1]  
  
## [1] 0.119
```

### Example 11.2.1

All data in `R` are stored in *objects*, which provide a range of *methods*. The class of an object can be found using the `class` function. For example, we have already encountered the `data.frame` class. It has a series of methods, such as `names` or `nrow`: (to Python)

```
class(iris)  
  
## [1] "data.frame"  
  
names(iris)
```

## Appendix B. R-Code

```
## [1] "Sepal.Length" "Sepal.Width"   "Petal.Length"  
## [4] "Petal.Width"  "Species"  
  
nrow(iris)  
  
## [1] 150
```

### Example 11.2.2

The basic class to represent a time series in R is the `ts` class. The **AirPassengers**-data in Example 11.1.1 is a built-in data set of class `ts`. ([to Python](#))

```
class(AirPassengers)  
  
## [1] "ts"
```

The most important methods for the class `ts` are

1. `start()` returns the start time of a time series
2. `end()` returns the end time of a time series
3. `frequency()` returns the number of samples per unit time
4. `plot()` displays the time series as a function over the time axis
5. `summary()` gives the five-number summary as well as the mean of the time series
6. `window()` returns a subset of the time series defined by a start and an end time

We study the output of the function for the **AirPassengers** data set. ([to Python](#))

```
start(AirPassengers)  
  
## [1] 1949     1  
  
end(AirPassengers)  
  
## [1] 1960     12  
  
frequency(AirPassengers)  
  
## [1] 12
```

## Appendix B. R-Code

Start and end times are 2-dimensional vectors of integers specifying natural time units. The time unit of the first entry of start time is year, i.e., 1949, the time unit of the second entry of start time is month, i.e., 1. So the time series starts in January 1949 and ends in December 1960.

The output of the `frequency()` method tells us, that the sampling frequency of the **AirPassengers** data is 12, where the base time unit is year defined by the first entry of `start()`, resp. `end`. Thus, the second entry of `start()`, resp. `end` refers to the number of sampling units.

In general, the `ts` class handles times and dates in a very basic fashion: enumeration from a given start time by constant time steps. The size of the time steps is given by one over the frequency and returned by the `deltat` function.

```
deltat(AirPassengers)  
## [1] 0.0833
```

That is, time step in the **AirPassengers** data corresponds to 0.083 years. The output of the basic `plot` function can be seen in Example 11.1.1. Note that the generic `plot` function calls the specific `plot.ts` function in the background that is tailored to time series. This is the R-Code that produces Figure 11.1 (to Python)

```
plot(AirPassengers, main = "Passengers", ylab = "Number (in 1000s)")  
grid()
```

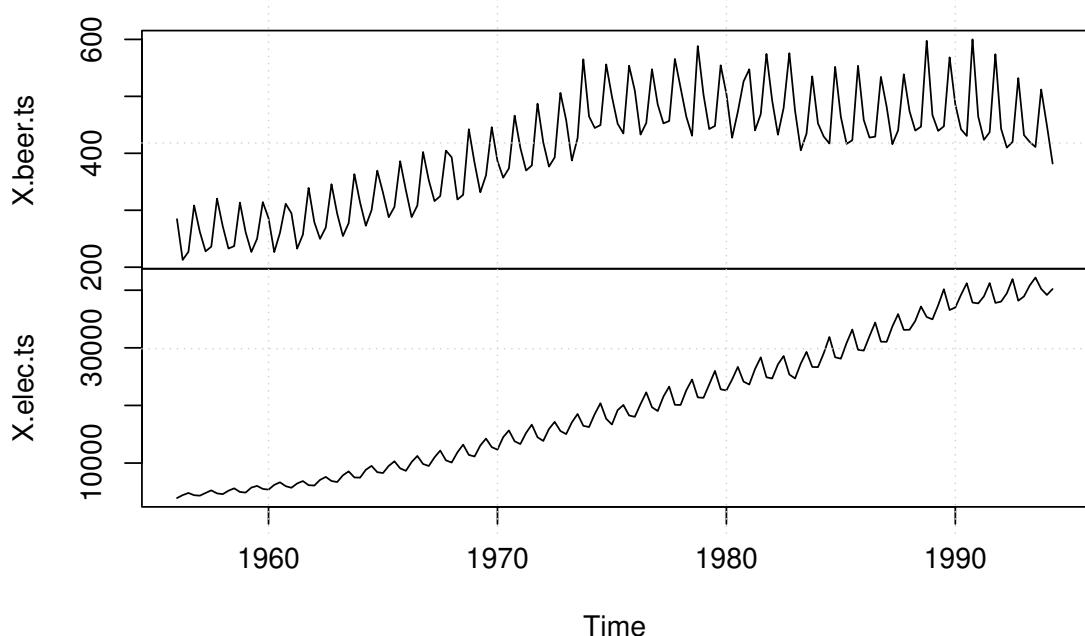
### Example 11.2.3

(to Python)

```
X.beer = read.table("./Data/AustralianBeer.csv", sep = ";",  
header = T)  
setwd(wd)  
summary(X.beer.ts)  
  
##      Min. 1st Qu. Median      Mean 3rd Qu.      Max.  
##      213     325     427     408     467     600
```

(to Python)

### Beer and electricity production in Australia



**Figure B.25.:** Quarterly beer and electricity production in Australia. Beer production in megalitres and electricity production in millions of kWh.

#### Example 11.2.4

([to Python](#))

```
X.elec = read.table("./Data/AustralianElectricity.csv",
                     sep = ";", header=T)
setwd(wd)
```

Now, we bind two separate series together by means of the `cbind` command and plot the series ([to Python](#))

```
X.ts = cbind(X.beer.ts, X.elec.ts)
plot(X.ts, main = "Beer and electricity production in Australia")
grid()
```

The two series constitute a multivariate time series. There are many functions in `R` for handling more than one series, including `ts.intersect` to obtain the intersection

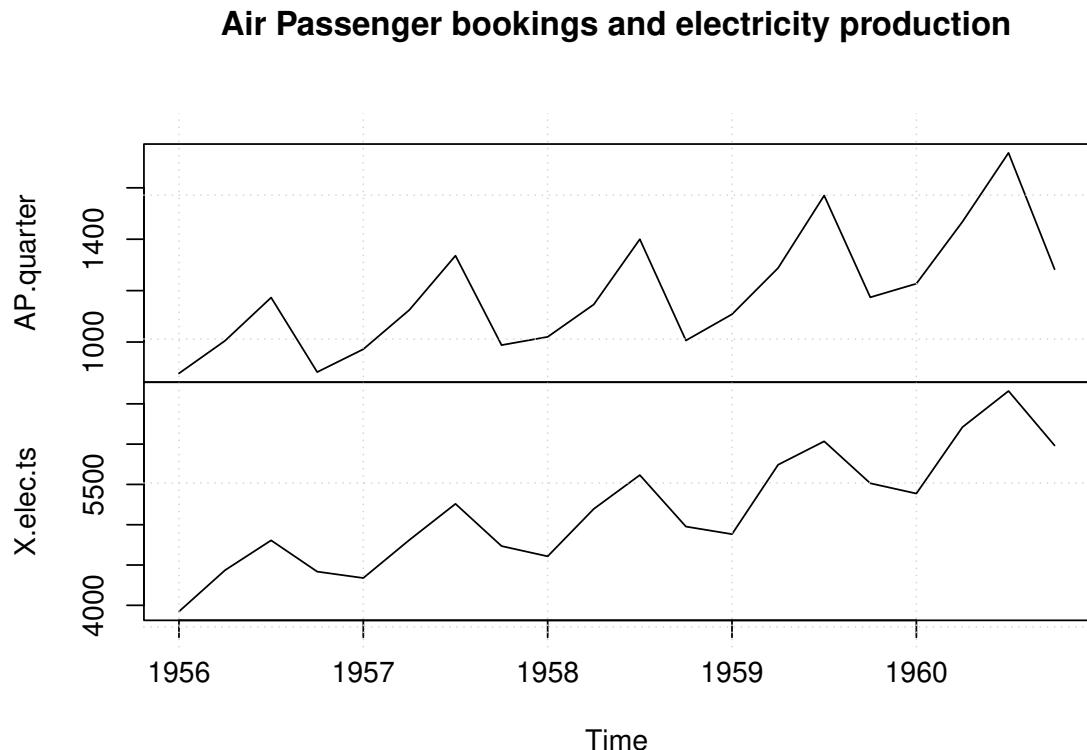
## Appendix B. R-Code

of two series that overlap in time. We now illustrate the use of the `intersect` function and point out some potential pitfalls in analyzing multivariate time series. The intersection between the air passenger data and the electricity data is obtained as follows. First aggregate the `AirPassengers` data to the same sampling frequency as the `electricity` data.

```
# aggregate the monthly data of the AirPassengers data  
# to quarterly data  
AP.quarter = aggregate(AirPassengers, nfrequency = 4)
```

Now, we can use the `ts.intersect` function, to extract the common time points in the two series and combine the corresponding data values to a new, bivariate time series.

```
AP.elec = ts.intersect(AP.quarter, X.elec.ts)  
plot(AP.elec,  
     main = "Air Passenger bookings and electricity production")  
grid()
```



**Figure B.26.:** Air passengers data of PanAm and beer production in Australia

### Further time series classes

The `ts` class provides a very simple yet powerful way to represent and analyze time series with `R`. The time representation relies on enumeration, as mentioned above, which can be cumbersome in practice. Moreover, non-equidistant time series are not covered by this class. There are several further classes coming along with contributed `R` packages. The most prominent are

- The `zoo`-package: It provides methods for regular and irregular spaced time series as well as arbitrary date formats. It is compatible with the `ts` class.
- The `xts`-package: It is an extension of the `zoo` package which allows for further customization.

In this course, we will exclusively use the `ts` class. In the Jupyter-Notebooks accompanying the lecture series we will provide an example of a `xts` time series.

### Example 11.3.1

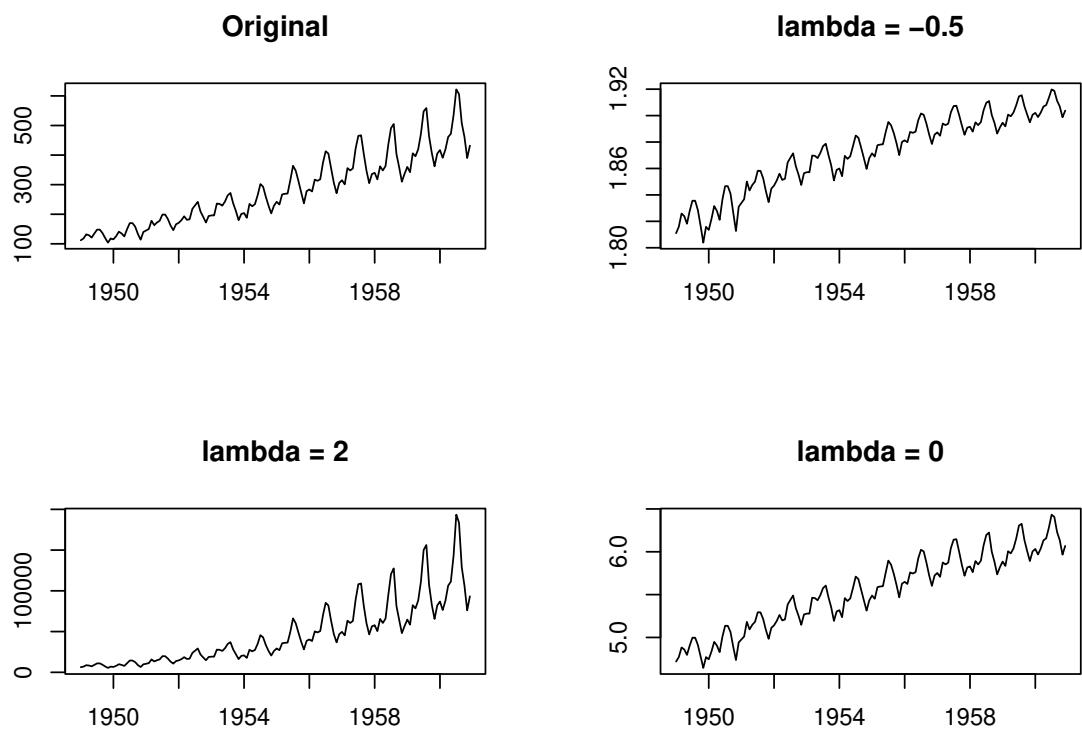
[\(to Python\)](#)

```
# define the Box-Cox transformation
box.cox <- function(x, lambda) {
  if (lambda==0) log(x) else (x^lambda - 1)/lambda
}
# plot the original and the transformed data
layout(matrix(c(1,2,3,4), 2,2))
plot(AirPassengers, main = "Original", ylab="", xlab="")
plot(box.cox(AirPassengers, 2), main = "lambda = 2",
      ylab="", xlab="")
plot(box.cox(AirPassengers, -0.5), main = "lambda = -0.5",
      ylab="", xlab="")
plot(box.cox(AirPassengers, 0), main = "lambda = 0",
      ylab="", xlab="")
```

### Example 11.3.2

We look at the `AirPassengers` data and apply a time shift for various values of  $k$ . In `R` we have the function `lag` at our disposal. [\(to Python\)](#)

## Appendix B. R-Code

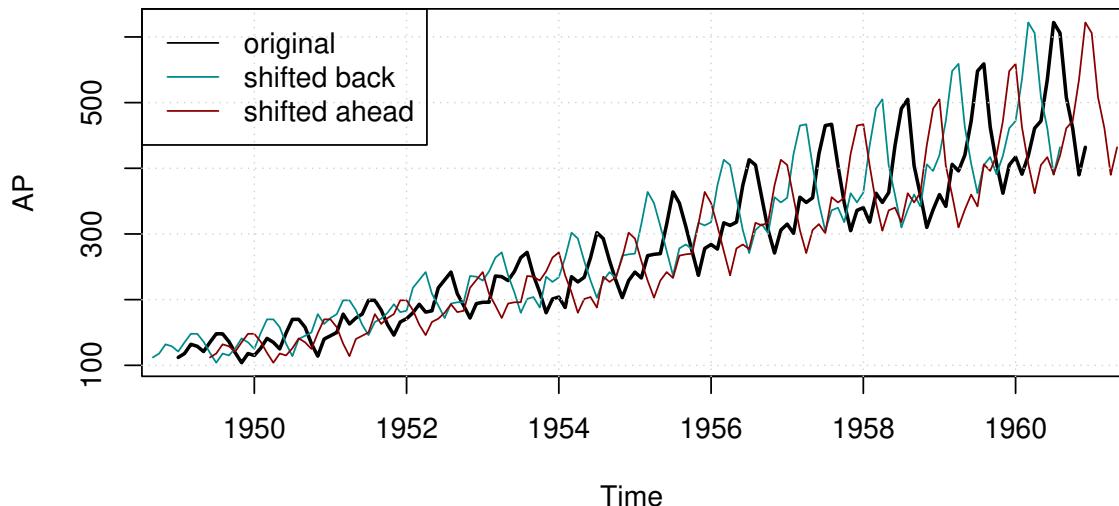


**Figure B.27.:** Box-Cox-transformations for different values of  $\lambda$ .

## Appendix B. R-Code

```
AP = AirPassengers
AP.back = stats::lag(AP, k = 4)
AP.ahead = stats::lag(AP, k = -5)

plot(AP, lwd = 2)
lines(AP.back, col = "darkcyan")
lines(AP.ahead, col = "darkred")
grid()
legend("topleft",
       legend = c("original", "shifted back", "shifted ahead"),
       lty = 1, col = c("black", "darkcyan", "darkred"))
```



### Example 11.3.4

(to Python)

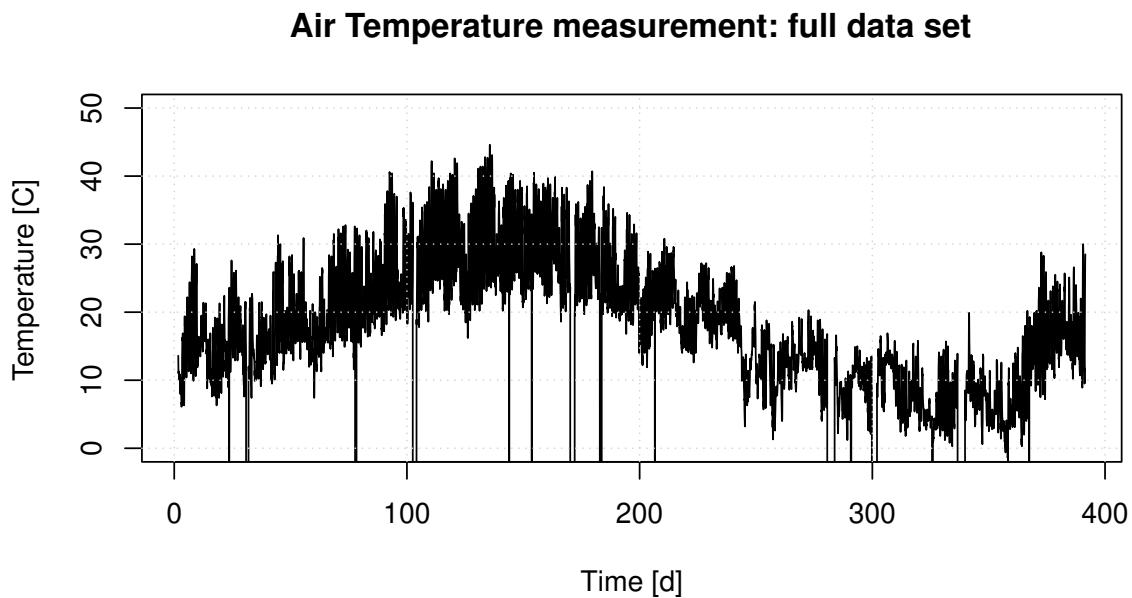
```
AirData = read.table("./Data/AirQualityUCI.csv",
                     sep = ";", header = T, dec = ",")
setwd(wd)
end(AirTmp.ts)

## [1] 391 14
```

## Appendix B. R-Code

We set the base time unit of this times series data set as day, starting at day 1, and its frequency as 24. The `end()`-command shows that the time series lasts 391 days and 14 hours, or in other words, has 9398 time points.

```
plot(AirTmp.ts, main = "Air Temperature measurement: full data set",
      ylab="Temperature [C]", xlab="Time [d]", ylim = c(0,50))
grid()
```



**Figure B.28.:** Hourly air temperature in an Italian city.

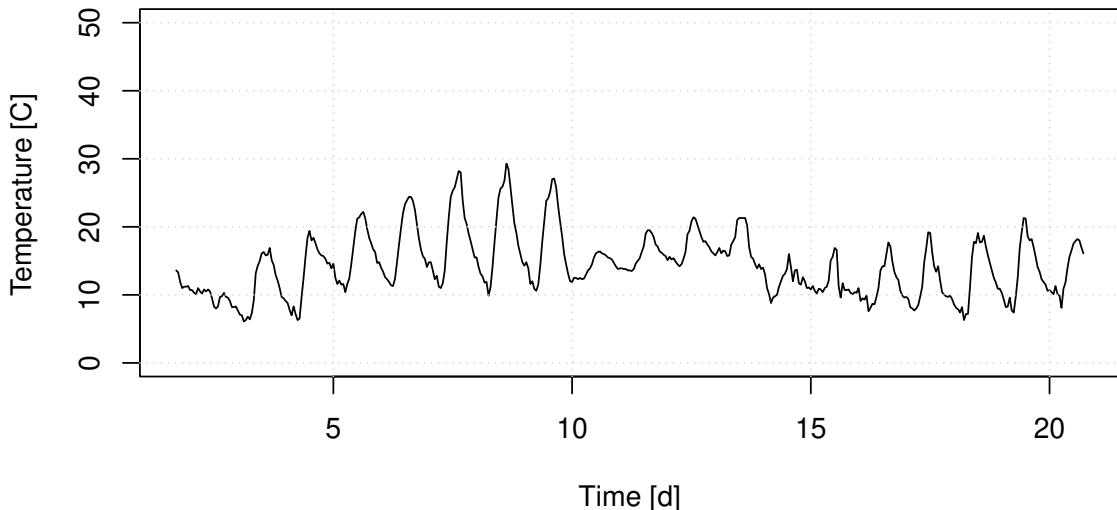
We focus on a period of 20 days to analyse the temperature behaviour in more detail. Figure 11.12 shows the data. [\(to Python\)](#)

```
AirTmpWin.ts = window(AirTmp.ts, start = c(1, 18), end=c(20, 18))
plot(AirTmpWin.ts, main = "Air Temperature measurement: detail",
      ylab="Temperature [C]", xlab="Time [d]", ylim = c(0,50))
grid()
```

### Example 11.3.5

We consider the air quality data from Example 11.3.4. The time series `AirTmpWin.ts` contains the hourly air temperature for a period of 20 consecutive days in March 2014 in an Italian city. We are going to generate a boxplot for each full hour in one figure. To

### Air Temperature measurement: detail



**Figure B.29.:** Hourly air temperatur of 20 consecutive days

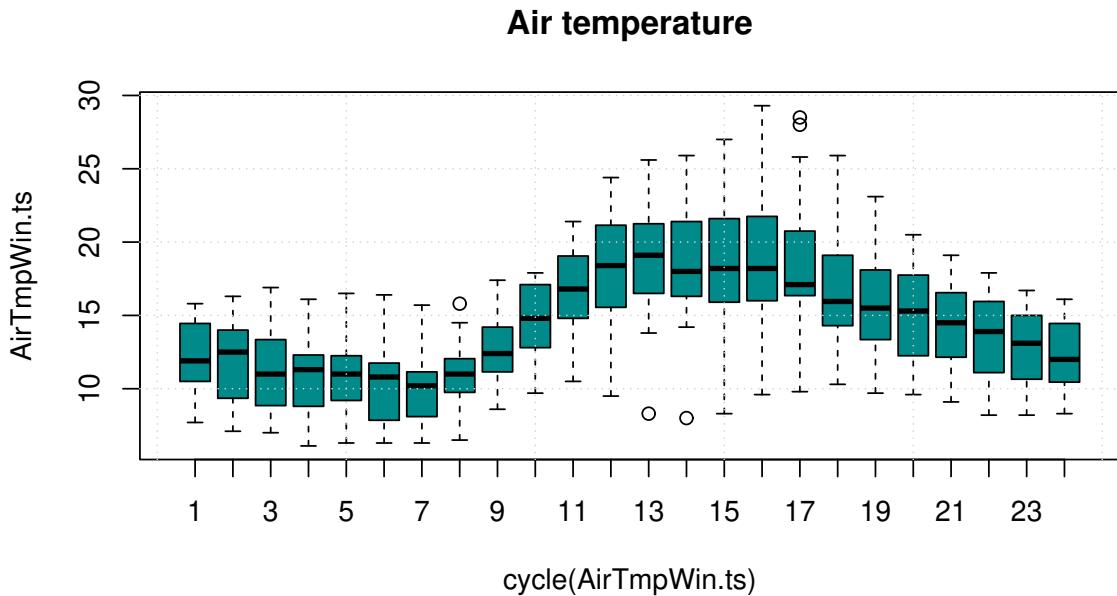
this end the `cycle()` function is very convenient: it returns for a given time series the positions in the cycle of each observation. In our example, a cycle is one day consisting of 24 hours. For example, we find:

```
cycle(AirTmp.ts) [1]
## [1] 18

cycle(AirTmp.ts) [875]
## [1] 4
```

This means, that the first entry in the time series is at cycle position 18, i.e. measured at 6 p.m. and that the 875-th measurement is at cycle position 4, which corresponds to 4 a.m. The subset of observations that share a common cycle are called *cycle-subseries* and will be used later for time series decomposition.

For the time being, we use the output of the `cycle`-command as a grouping variable, e.g. for the `boxplot` function. The ouput of the following code snippet is shown in Figure 11.13 (to Python)



**Figure B.30.:** Grouped boxplot of air temperature data.

```
boxplot(AirTmpWin.ts ~ cycle(AirTmpWin.ts), col = "darkcyan",
        main = "Air temperature")
grid()
```

### Example 11.3.6

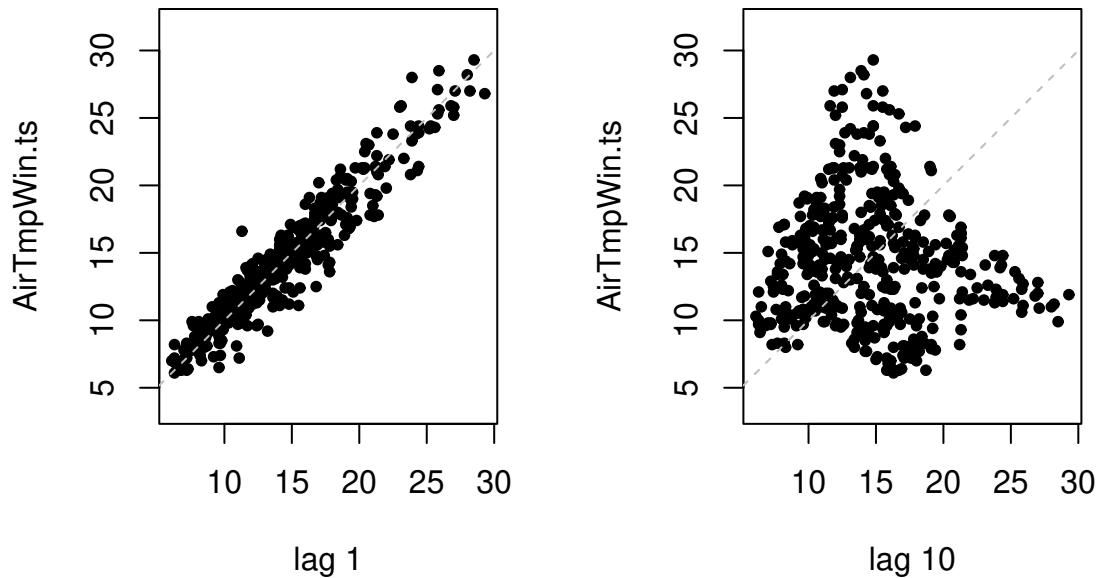
We consider again the 20 day sequence of hourly air temperature measurements. (to Python) We apply the `lag.plot()` command with default setting, i.e.  $k = 1$ , and with  $k = 10$  (to Python).

```
lag.plot(AirTmpWin.ts, pch = 20, main = "")
lag.plot(AirTmpWin.ts, pch = 20, main = "", set.lags = 10)
```

### Example 11.3.7

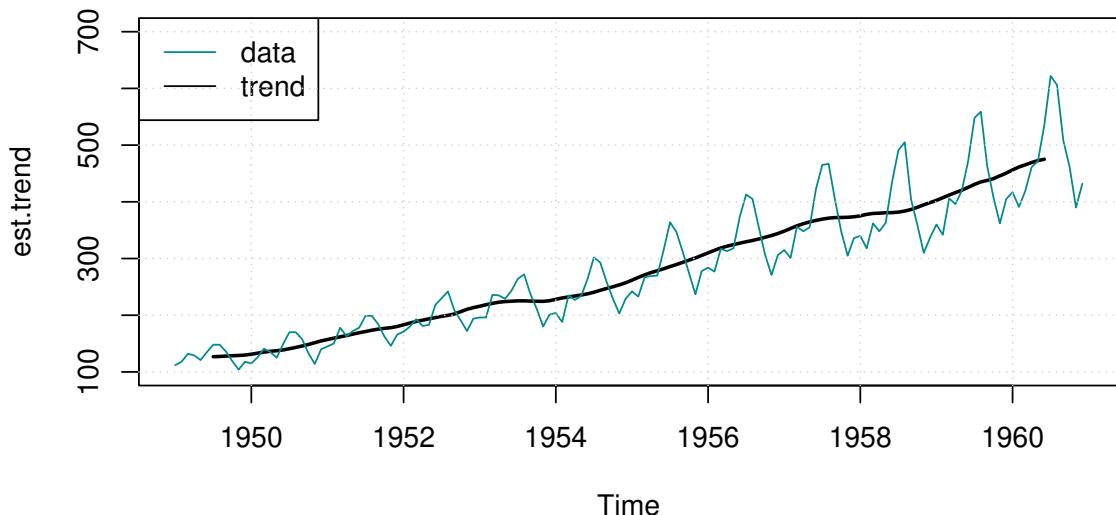
We look again at the **AirPassengers** data and estimate the trend by a moving average filter. In R the function `filter` does the trick. (to Python)

## Appendix B. R-Code



**Figure B.31.:** Lagged scatterplot of the air temperature data.

```
weights = c(0.5, rep(1, 11), 0.5)/12
est.trend <- stats::filter(AirPassengers, filter = weights,
    sides = 2)
plot(est.trend, lwd = 2, ylim = c(100, 700))
lines(AirPassengers, col = "darkcyan")
legend("topleft", legend = c("data", "trend"), lty = 1,
    col = c("darkcyan", "black"))
grid()
```



### Example 11.3.8

([to Python](#))

The averaging is done by the `tapply` command invoking the cycles of the time series. The `cycle` command returns for each observation the relative position in the cycle. With `tapply` the mean is applied to an array, where the cycle positions are used as grouping variables. Put differently, the `mean()` function is applied to all elements in the time series that have cycle position 1 (i.e. January), ....

```
est.season <- AirPassengers - est.trend
cyc <- factor(cycle(AirPassengers))
head(cyc)

## [1] 1 2 3 4 5 6
## Levels: 1 2 3 4 5 6 7 8 9 10 11 12

length(cyc)

## [1] 144

est.season.month <- tapply(est.season, cyc, mean, na.rm = T)
head(est.season.month)
```

## Appendix B. R-Code

```
##      1      2      3      4      5      6
## -25.50 -36.94 -2.99 -8.79 -5.26 34.65

length(est.season.month)

## [1] 12

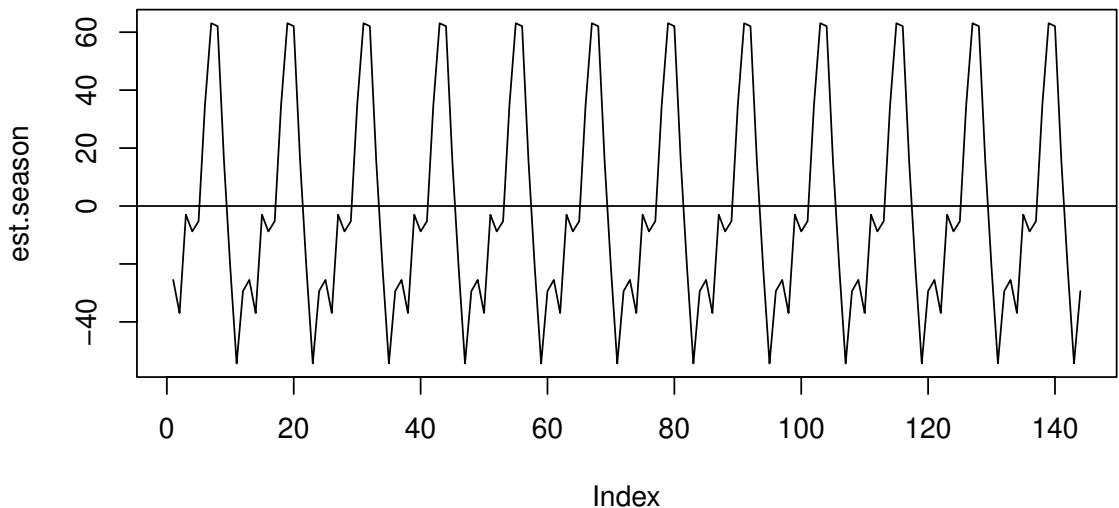
est.season <- est.season.month[cyc]
head(est.season)

##      1      2      3      4      5      6
## -25.50 -36.94 -2.99 -8.79 -5.26 34.65

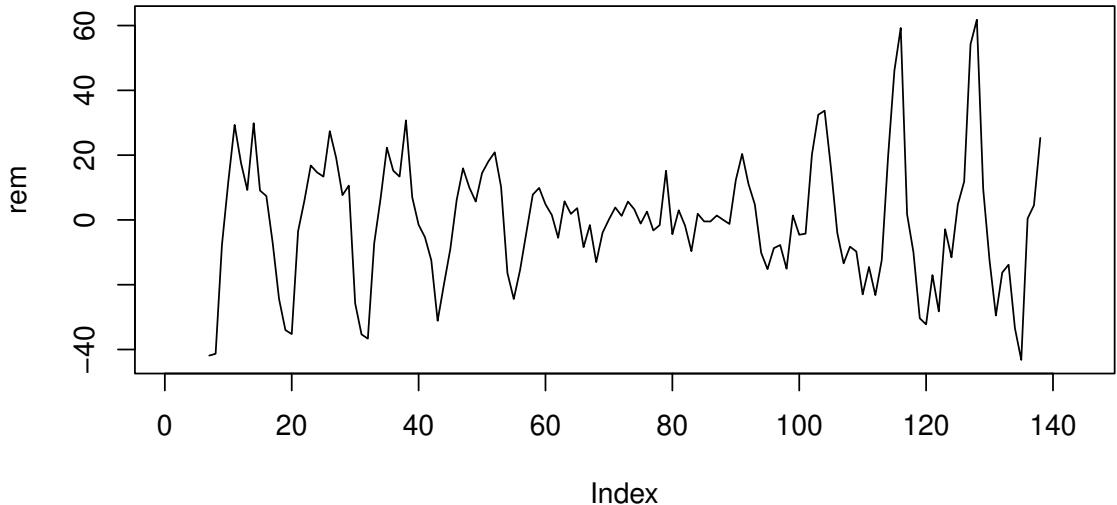
length(est.season)

## [1] 144

plot(est.season, type = "l")
abline(h = 0)
```



### Example 11.3.9



**Figure B.32.:** Estimated remainder term for the AirPassengers data. There is still non-random structure visible.

([to Python](#))

```
est.rem <- AirPassengers - est.trend - est.season
plot(as.vector(est.rem), type = "l", ylab = "rem") #needs fix
```

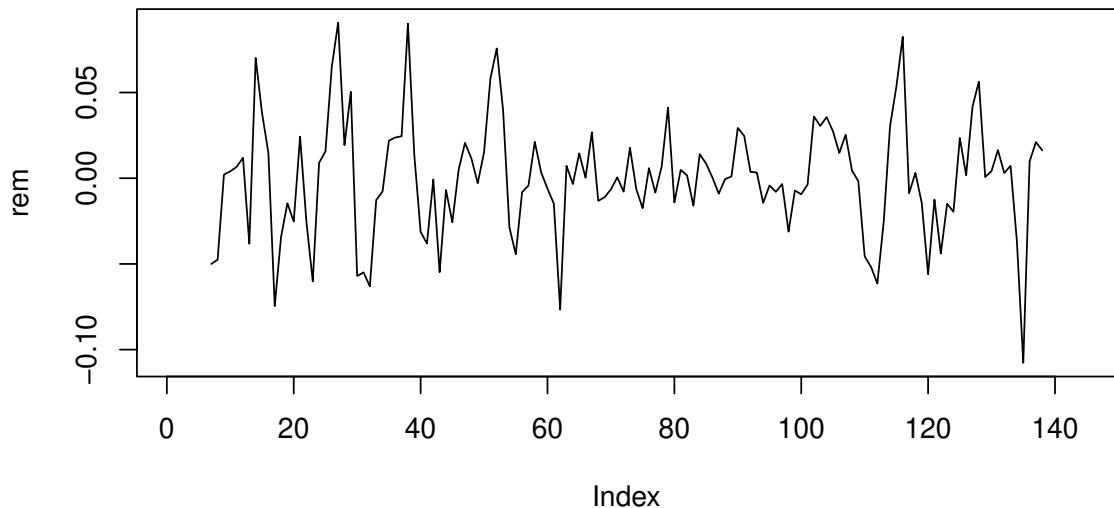
Figure 11.17 shows the estimated remainder term  $\hat{r}$ . It is striking that there is non-random structure left in the remainder term. The reason for that is, that the linear decomposition model is not true in this case. We hence repeat the steps above for the *logarithm* of **AirPassengers** which amounts to a multiplicative model. ([to Python](#))

```
log.data = log(AirPassengers)

# trend estimation of log data
est.trend.log <- stats::filter(log.data, filter = weights,
    sides = 2)

# seasonality estimation for log data
est.season.log <- log.data - est.trend.log

est.season.month <- tapply(est.season.log, cyc, mean, na.rm = T)
est.season.log <- est.season.month[cyc]
```



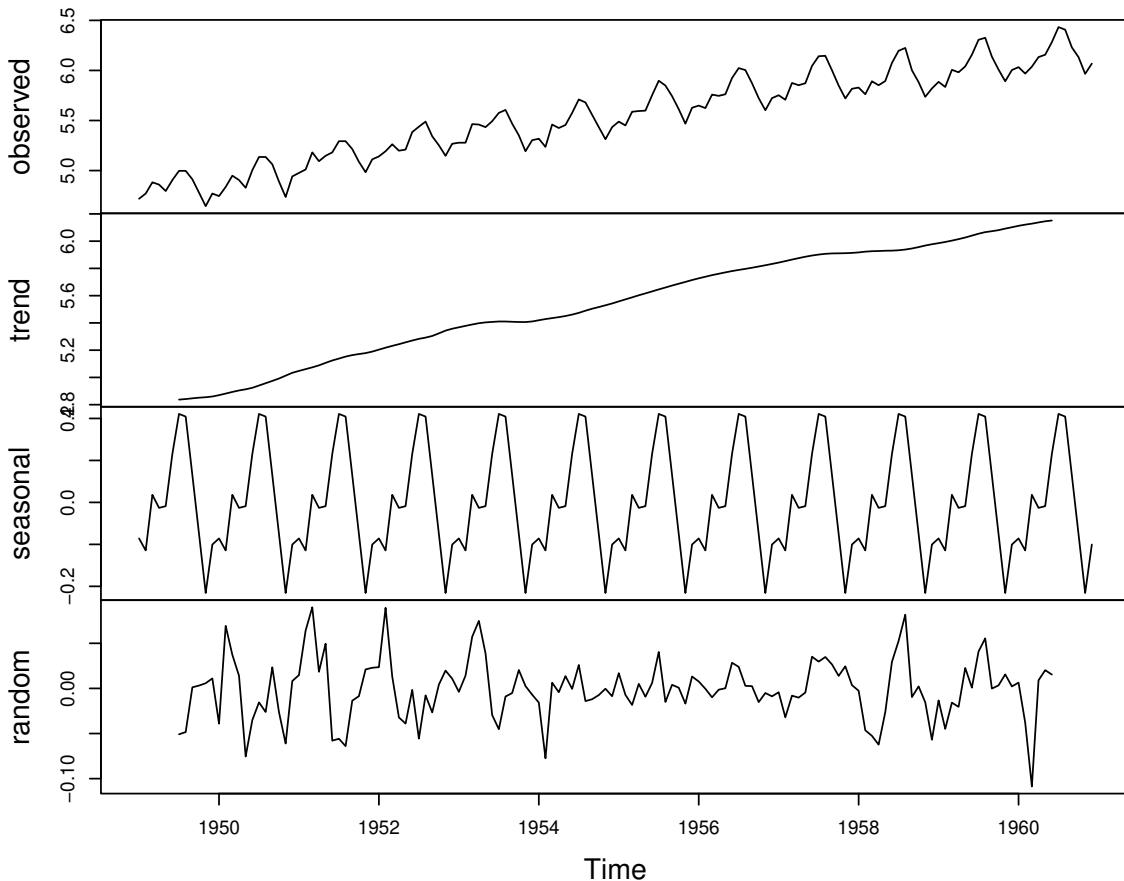
**Figure B.33.:** Estimated remainder term for the logarithm of the AirPassengers data. The non-random structure has diminished

```
# remainder term estimation for log data
est.rem.log <- log.data - est.trend.log - est.season.log
plot(as.vector(est.rem.log), type = "l", ylab = "rem") #needs fix
```

There is a convenient way to perform a time series decomposition based on moving averages in R. The `decompose()` function can be applied to a time series and performs the above steps. So the decomposition of the logarithmic **AirPassengers** data in Example 11.3.8 can be performed by the following code (to Python)

```
decomposed.data = decompose(log(AirPassengers))
plot(decomposed.data)
```

### Decomposition of additive time series



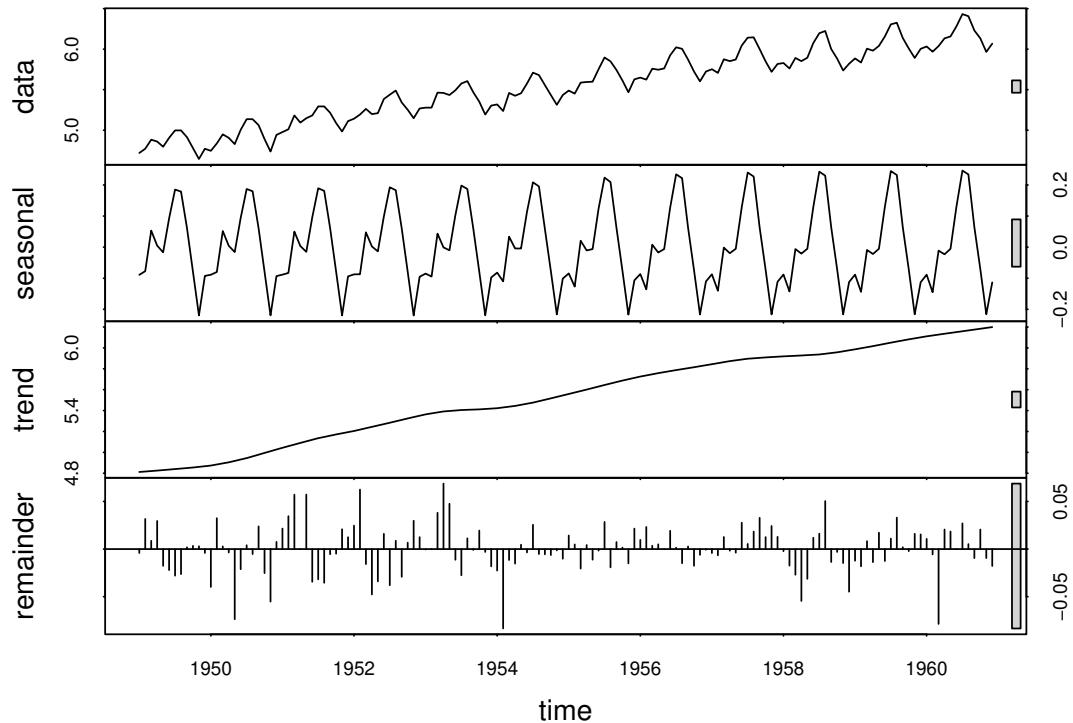
### Example 11.3.10

We examine the **AirPassengers** data set and decompose it by means of the `stl()` function. There are two mandatory parameters that have to be passed to the function

1. `x` the time series to be decomposed
2. `s.window` the loess window size for the seasonality component. The larger the value the slower the change of seasonality in the data set over time.

The following R-code estimates and plots an STL decomposition of the logarithm of the data ([to Python](#))

```
stl.fit = stl(log(AirPassengers), s.window = 10)
plot(stl.fit)
```



**Figure B.34.:** STL decomposition of the log of the Air Passengers data.

Figure 11.3.10 shows the decomposition of the data. Note that the seasonal component is changing over time and the remainder term is much smaller than in Example 11.3.8.

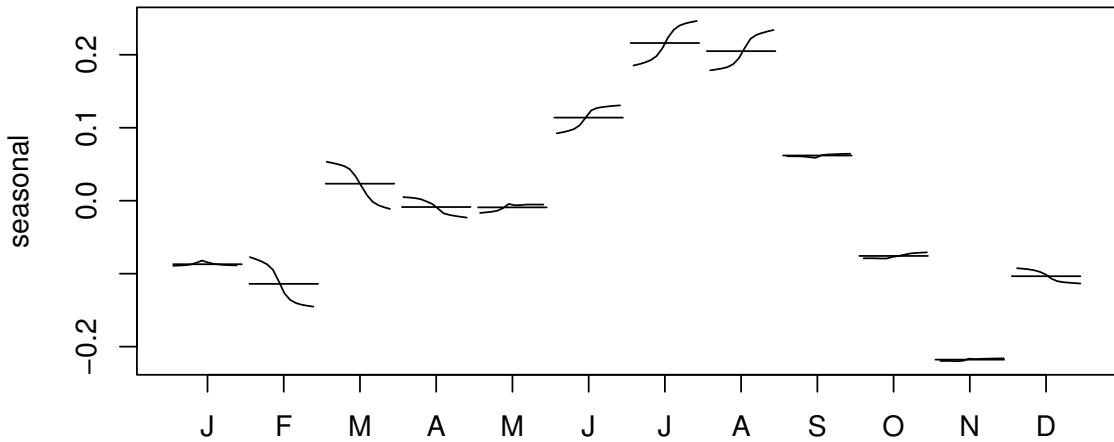
A very practical tool for examining the estimated sub-cycles and thus to assess the choice of `s.window` is the `monthplot()` function. It plots the cycle-subseries in a common plot (cf. Figure ??). The `monthplot()` function displays the seasonal cycle-subseries in a plot characterized by

1. a *vertical axis* which corresponds to the response variable of the time series
2. a horizontal axis which relates to the cycle positions in chronological order. For example, with monthly data, all the January values are plotted (in chronological order, that is according to years), then all the February values are plotted (again the order defined by the year), and so on.
3. the horizontal line of each cycle position displays the mean value for each cycle-subseries. For example, with monthly data, the average of the cycle-subseries values corresponding to January is computed.

## Appendix B. R-Code

4. a loess curve for each cycle position fitted to the corresponding subseries is plotted.

```
monthplot(stl.fit)
```



**Figure B.35.:** Cycle-subseries of the STL decomposition of the log Air Passengers data.

### Remarks:

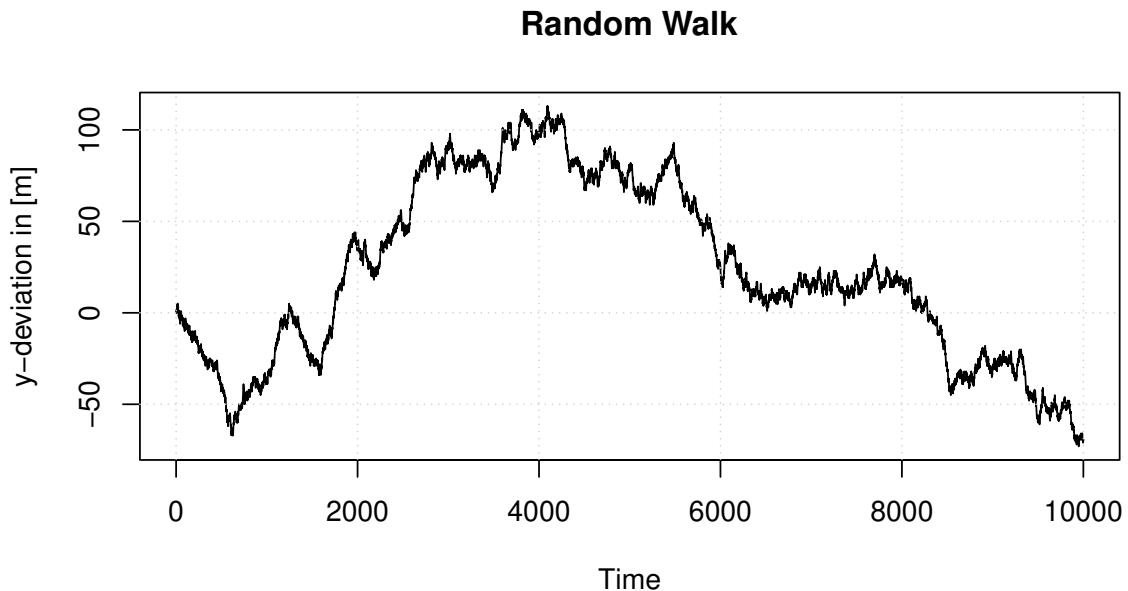
- i. It becomes obvious that the seasonal contribution of air passengers in the winter months is decreasing. For instance, if we look at the January subseries, then we observe that the loess curve is decreasing. In other words, the seasonal effects for February are decreasing along the time axis.
- ii. It becomes obvious that the seasonal contribution of air passengers in the summer months is increasing. For instance, if we look at the August subseries, then we observe that the loess curve is increasing. In other words, the seasonal effects for August are increasing along the time axis. ♦

### Example 12.1.1

([to Python](#))

## Appendix B. R-Code

```
d = sample(c(-1,1), replace = T, size = 10000)
x = ts(cumsum(d))
# Plot
plot(x, main = "Random Walk", ylab = "y-deviation in [m]")
grid()
```



**Figure B.36.:** A time series as observation of a random walk process.

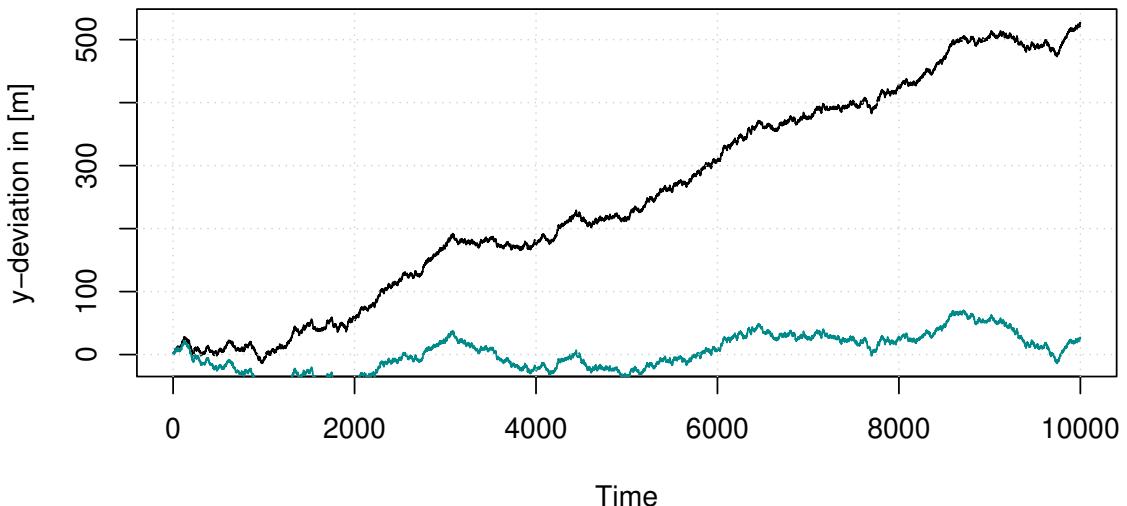
```
# Add drift  (delta)
set.seed(12)
d = sample(c(-1, 1), replace = T, size = 10000)
delta = 0.05
y = seq(from = 0, to = (10000 - 1) * delta, by = delta)
Y = y + cumsum(d)

# plot
plot(ts(y), main = "Random Walk with drift",
      ylab = "y-deviation in [m]")
lines(cumsum(d), col = "darkcyan")
grid()
```

### Example 12.1.2

([to Python](#))

### Random Walk with drift



**Figure B.37.:** A time series observation of a random walk with drift (black).

```
w = ts(rnorm(1000))
plot(w, main = "White noise", ylab = "value")
grid()
```

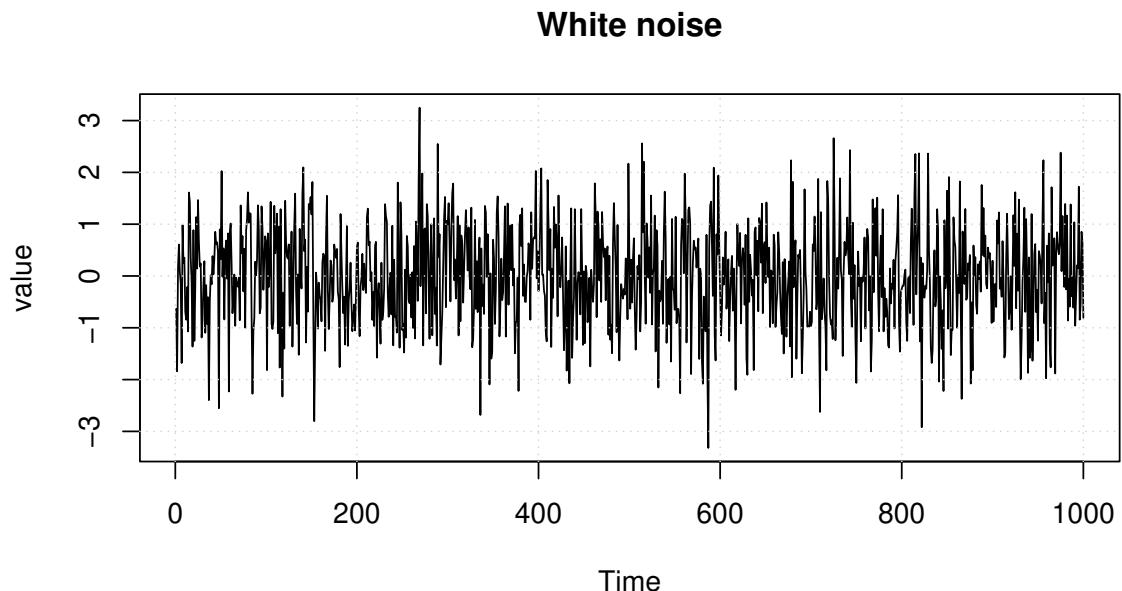
### Example 12.1.3

In R a moving average filter can be realized by means of the `filter()` function. ([to Python](#))

```
window = c(1, 1, 1)/3
v = stats::filter(w, sides = 2, window)
plot(v, main = "MA process", ylab = "")
```

### Example 12.1.4

In R the `filter()` function has a parameter called `method` which can be set to "`recursive`". This will compute the autoregression model. ([to Python](#))



**Figure B.38.:** A realization of the white noise process

```
ar = stats:::filter(w, filter = c(1.5, -0.9), method = "recursive")
plot(ar, main = "AR(2) process", ylab = "")
grid()
```

### Example 12.3.2

([to Python](#))

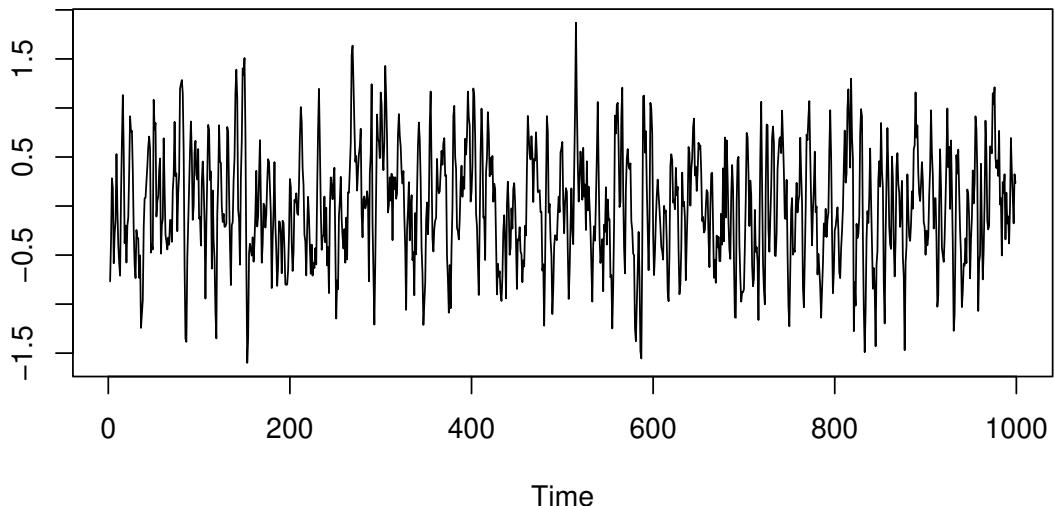
```
# Read the Data
AusEl = read.table("./Data/AustralianElectricity.csv",
                   sep = ";", header=T)

AusEl.ts = ts(AusEl[,2], frequency = 4)
res = stl(log(AusEl.ts), s.window = 16)

# the time series component contains 3 series where the
# third constitutes the remainder sequence
plot(res$time.series[, 3],
     main = "Remainder: Australian Electricity")
grid()
```

([to Python](#))

### MA process



**Figure B.39.:** A realization of a moving average process with window length 3.

```
# Read the Data
AirData = read.table("./Data/AirQualityUCI.csv",
                     sep = ";", header = T, dec = ",")
# Create Time Series and perform STL decomposition
AirTmp.ts = ts(AirData[, c(13)], start = c(1, 18),
               frequency = 24)
AirTmpWin.ts = window(AirTmp.ts, start = c(1, 18),
                      end = c(20, 18))
res = stl(AirTmpWin.ts, s.window = 10)
# plot
plot(res$time.series[, 3], main = "Remainder: Air Temperature")
grid()
```

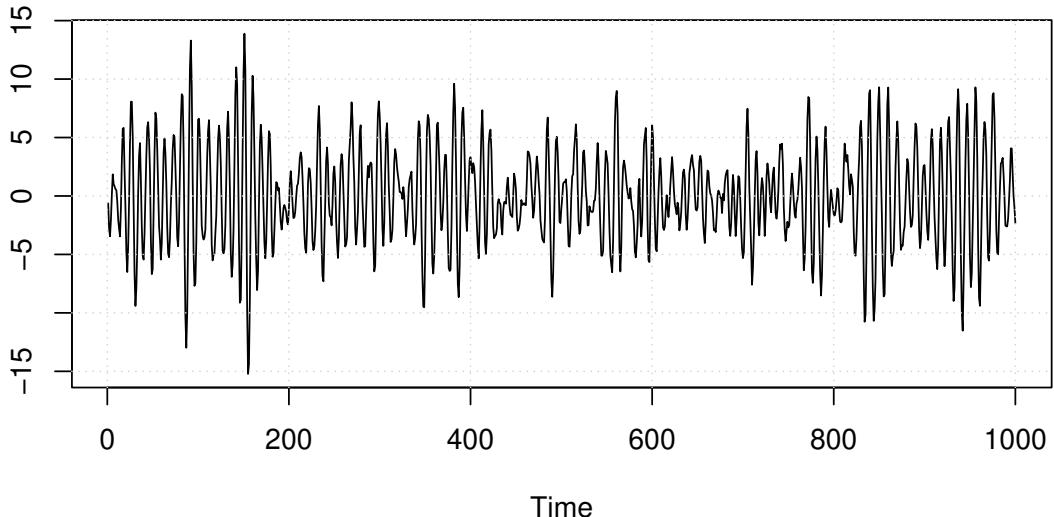
### Example 12.4.1

([to Python](#))

```
# create white noise
set.seed(123)
w = rnorm(1000, mean = 2, sd = 0.1)

# filter and define time series
```

### AR(2) process



**Figure B.40.:** A realization of a autoregressive process.

```
MA = ts(stats::filter(w, filter = rep(1, 3)/3, sides = 2))
MA = na.omit(MA)
```

During the application of the moving average (in the `filter` command), missing values `NA` are generated at the boundaries. This would confuse further computations. The function `na.omit` omits these cases where `NAs` are involved. In `R` autocovariance and -correlation can then be computed with the `acf` function.

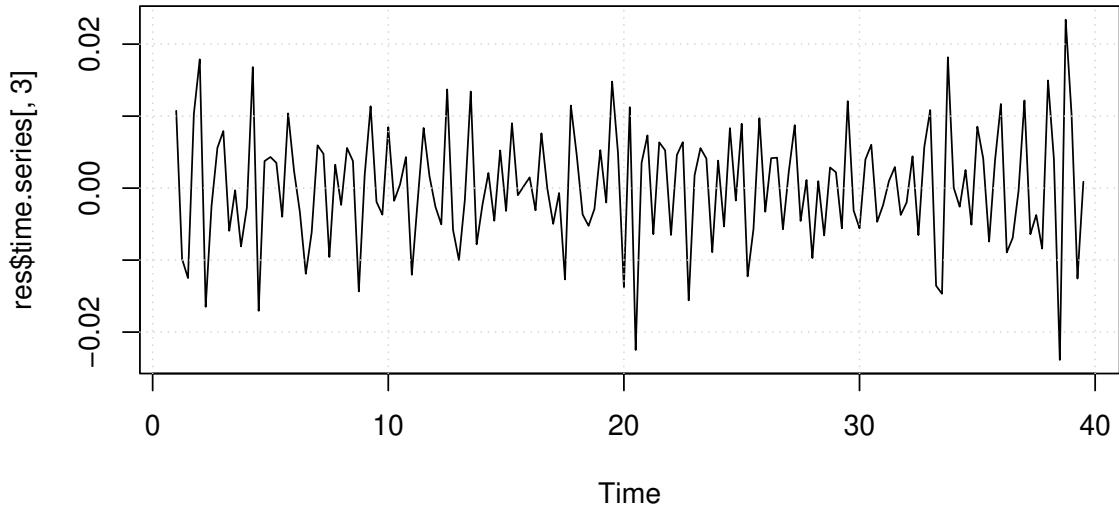
([to Python](#))

```
ac = acf(MA, type = "covariance", lag.max = 50, ylim = c(0,
0.004))

# theoretical autocovariance
sigma = 0.1
cv.true = rep(0, 1, length(ac$acf))
cv.true[1] = 3 * sigma^2/9
cv.true[2] = 2 * sigma^2/9
cv.true[3] = sigma^2/9

points(ac$lag + 0.3, cv.true, pch = 18, col = "darkred",
type = "h")
legend("topright", legend = c("estimated", "theoretical"),
lty = 1, col = c("black", "darkred"))
```

### Remainder: Australian Electricity



**Figure B.41.:** The remainder of the Australian electricity data after transformations.

The `type` parameter chooses either `correlation` (default) or `covariance` and `lag.max` gives the maximal lag up to which the autocovariance should be computed.

### Example 12.4.2

([to Python](#))

```
acf(MA, lag.max = 50)
```

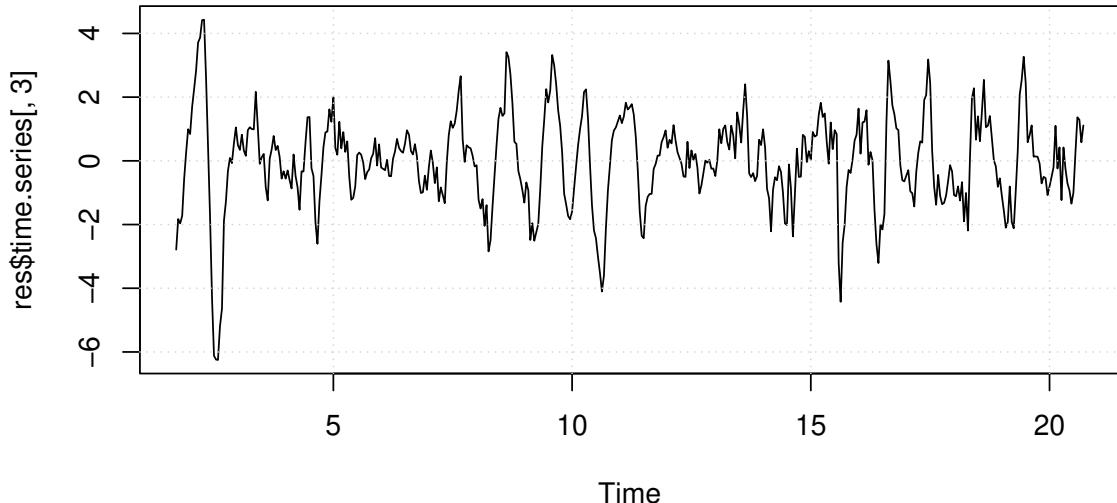
### Example 13.1.3

([to Python](#))

```
abs(polyroot(c(1, -0.5, 0.5, 0.1)))
## [1] 1.28 1.28 6.09
```

We can simulate a time series from this model with the `arima.sim` command.  
([to Python](#))

### Remainder: Air Temperature



**Figure B.42.:** The remainder of the temperature measurements after STL decomposition.

```
set.seed(23)
plot(arima.sim(model = list(ar = c(0.5, -0.5, -0.1)), n = 200),
      ylab = "value", main = "AR(3) process")
```

### Example 13.1.5

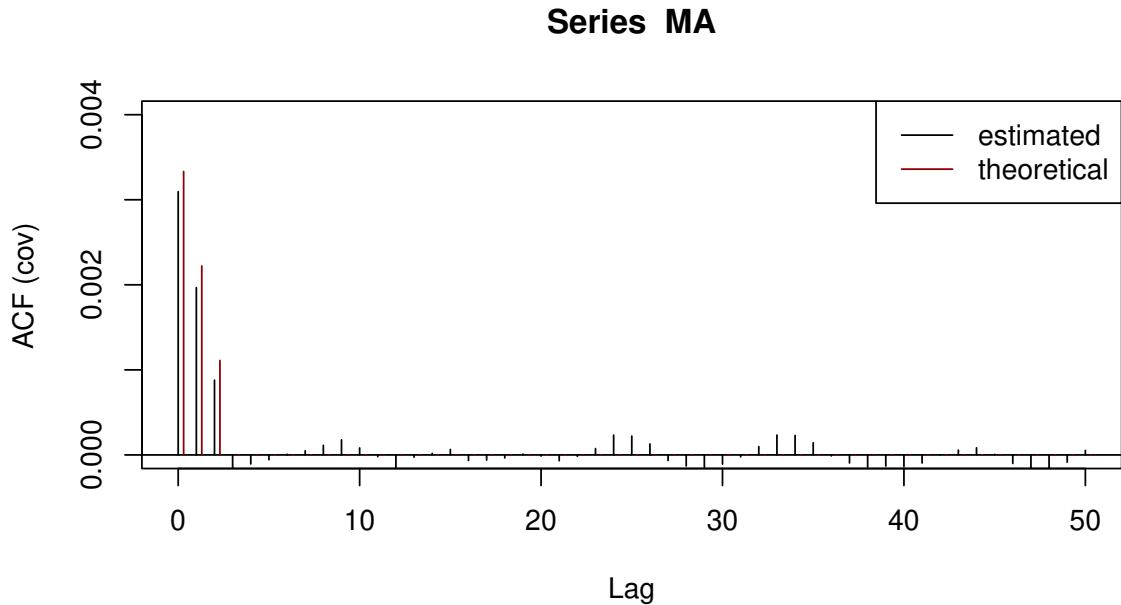
(to Python) We use the `ARMAacf()` function to compute the autocorrelation function.

```
ac = ARMAacf(ar = c(0.5, -0.5, -0.1), lag.max = 40)
plot(ac, type = "h", main = "ACF of an AR(3) process")
grid()
```

### Example 13.1.6

(to Python)

```
pacf(ac, main = "Partial autocorrelation of AR(3)")
```



**Figure B.43.:** Sample autocovariance function of a MA(5) process

### Example 13.1.7

([to Python](#))

```
X = sunspot.month
plot(X, main = "sunspot number")
X.year = aggregate(X, nfrequency = 1, FUN = mean)
lines(X.year, col = "red", lwd = 2)
```

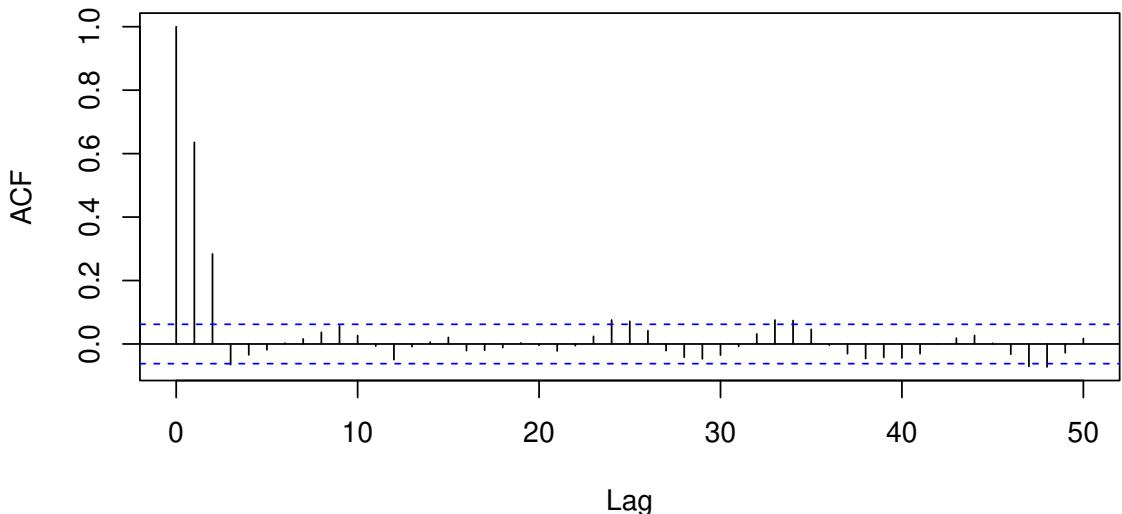
In Figure 13.4 the yearly averaged time series is also shown (red). The latter will be used for modelling. ([to Python](#))

```
X.year.sq = (sqrt(X.year) - 1) * 2
plot(X.year.sq)
```

([to Python](#))

```
par(mfrow = c(1, 2))
acf(X.year.sq, lag.max = 50, main = "ACF")
pacf(X.year.sq, lag.max = 50, main = "PACF")
abline(v = 9, col = "red", lty = 3)
```

### Series MA



**Figure B.44.**: Sample autocorrelation function of a MA(5) process

### Example 13.1.8

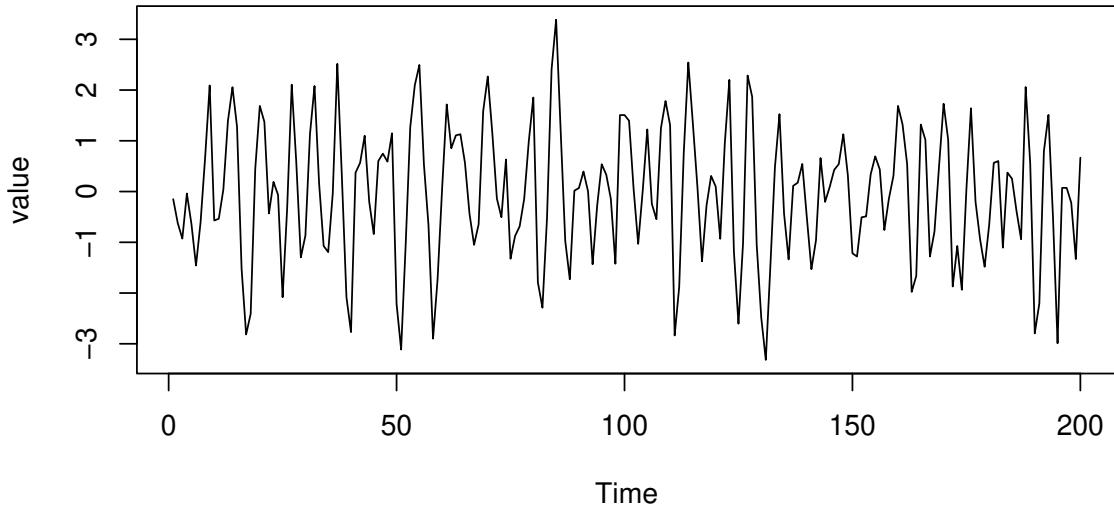
([to Python](#))

```
model = ar(X.year.sq, method = "burg", aic = F, order.max = 9)
plot(X.year.sq, ylim = c(-5, 30))
lines(X.year.sq - model$resid, col = "red")
legend("topleft", legend = c("true", "estimated"), col = c("black",
"red"), lty = 1, cex = 0.6)
```

```
model

##
## Call:
## ar(x = X.year.sq, aic = F, order.max = 9, method = "burg")
##
## Coefficients:
##      1       2       3       4       5       6
## 1.216 -0.492 -0.133  0.249 -0.243  0.060
##      7       8       9
## 0.115 -0.199  0.299
##
## Order selected 9  sigma^2 estimated as  4.52
```

### AR(3) process



**Figure B.45.:** Time series generated by the AR(3) model.

The parameter `aic=F` deactivates AIC model selection because we want to force the system to use  $p = 9$ . [\(to Python\)](#)

```
hist(model$resid, nclass = 50, col = "darkred",
      main = "Histogram of the residuals")
qqnorm(model$resid, col = "darkred", pch=20); grid()
```

### Example 13.1.10

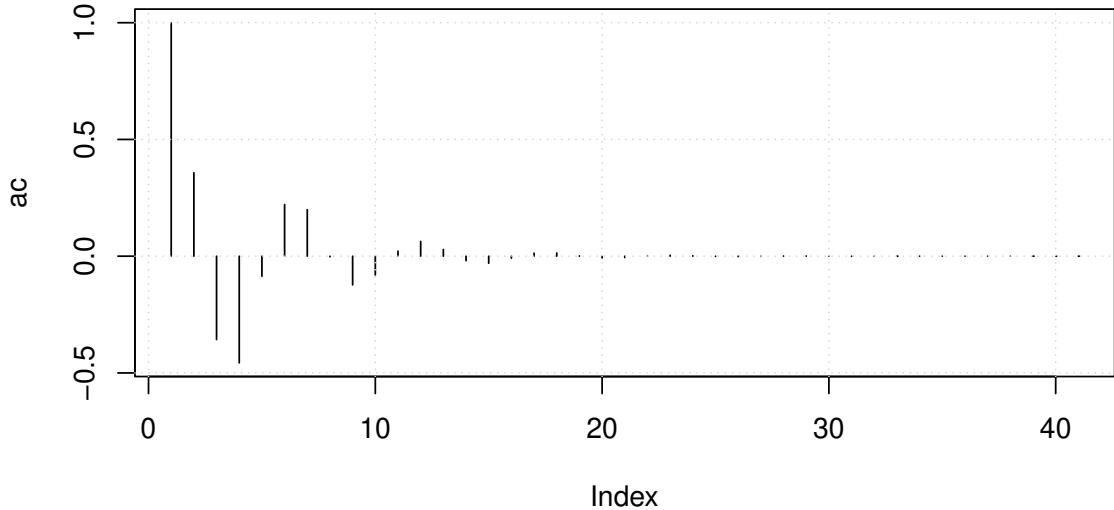
[\(to Python\)](#)

```
set.seed(123)
a1 = 0.7
a1.ts = arima.sim(model = list(ar = a1), n = 150)
a1.train.ts = window(a1.ts, start = 1, end = 130)
plot(a1.ts, lty = 3, main = "Prediction of an AR(1) process")
lines(a1.train.ts)

# estimate the model
model = ar(a1.train.ts, aic = F, method = "burg", order.max = 1)

# predict future values
```

### ACF of an AR(3) process



**Figure B.46.:** The theoretical autocorrelation of an AR(3) process.

```
pred = predict(model, n.ahead = 20)

# plot the predicted values
lines(pred$pred, col = "red", lwd = 2)

# confidence bands
lines(pred$pred + 1.96 * pred$se, col = "red", lty = 2)
lines(pred$pred - 1.96 * pred$se, col = "red", lty = 2)

grid()
```

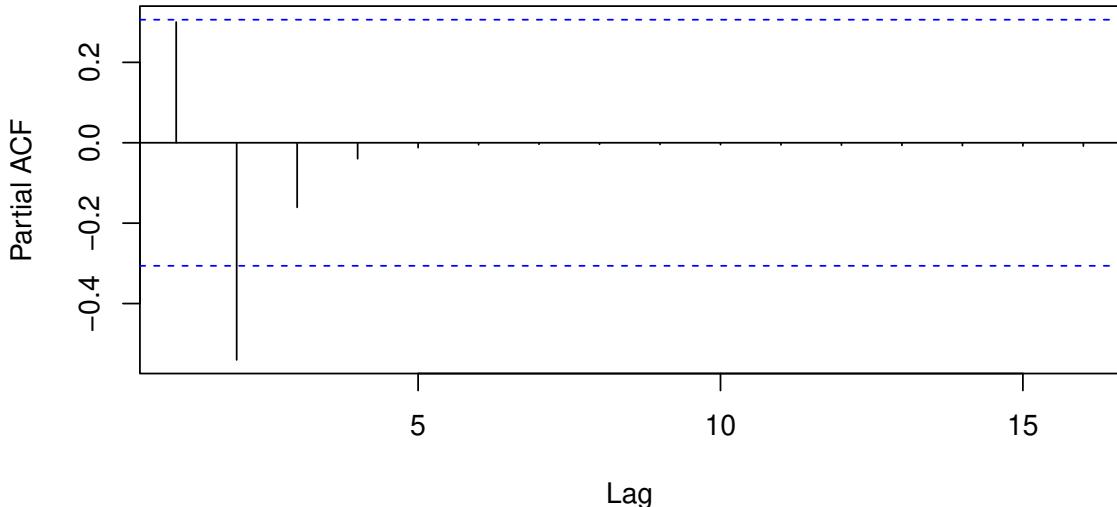
Finally, we add some confidence bands to the plot. To this end, we use the standard error that is stored in `pred$se`.

### Example 13.1.11

([to Python](#))

```
# create training set
X.year.sq.train = window(X.year.sq, start = 1749, end = 1989)
# build model
model = ar(X.year.sq.train, aic = F, order.max = 9, method = "burg")
```

### Partial autocorrelation of AR(3)



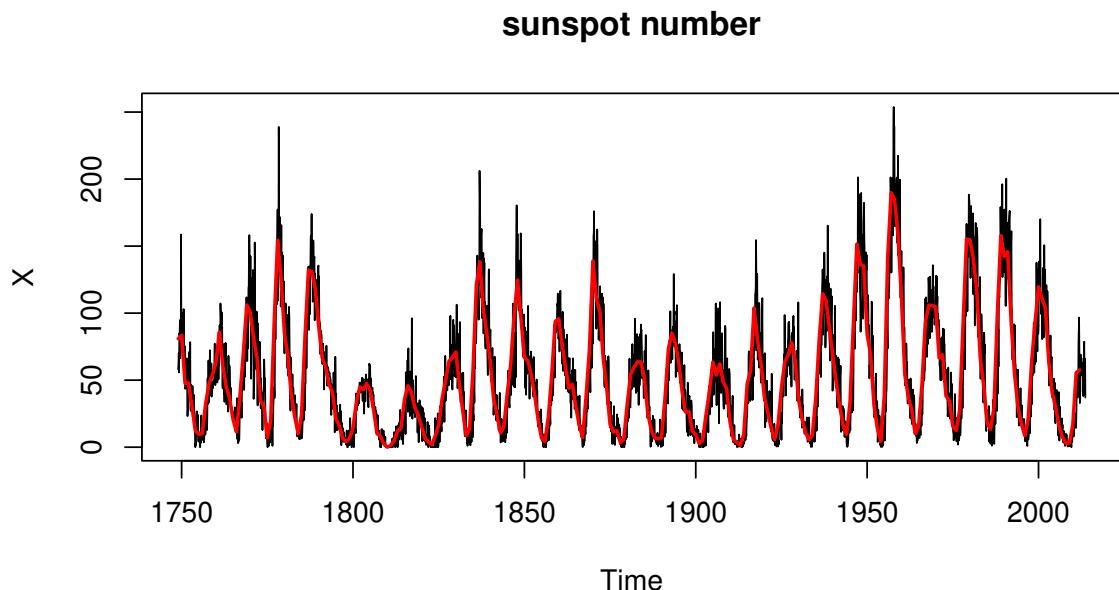
**Figure B.47.:** The estimated partial autocorrelation function of an AR(3) process.

```
# predict future values
pred = predict(model, n.ahead = 23)
# do some plotting
plot(X.year.sq, lty = 3, ylim = c(-5, 30), xlim = c(1950,
2012))
lines(X.year.sq.train)

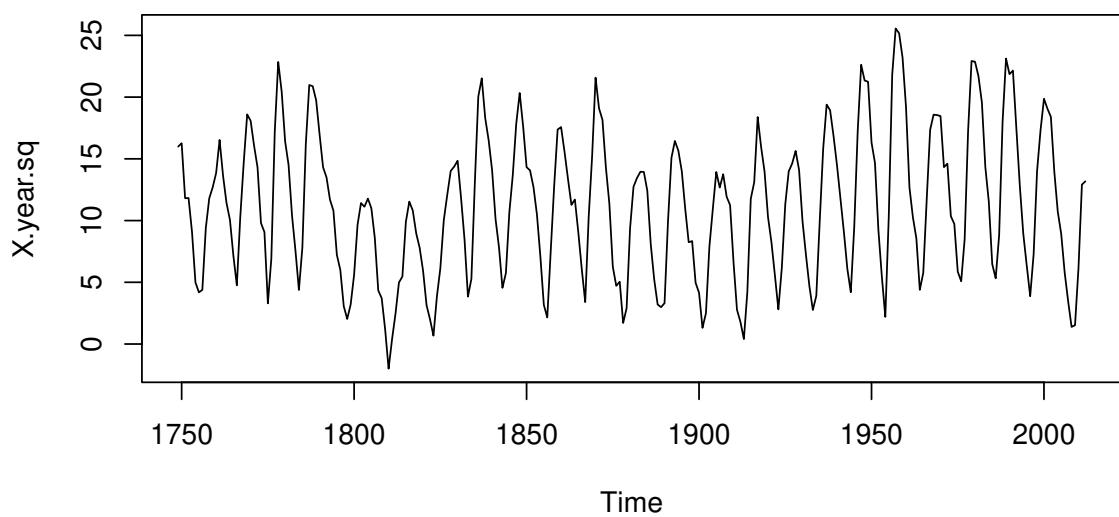
lines(pred$pred, col = "red")
lines(pred$pred + 1.96 * pred$se, col = "red", lty = 2)
lines(pred$pred - 1.96 * pred$se, col = "red", lty = 2)

grid()
```

## Appendix B. R-Code

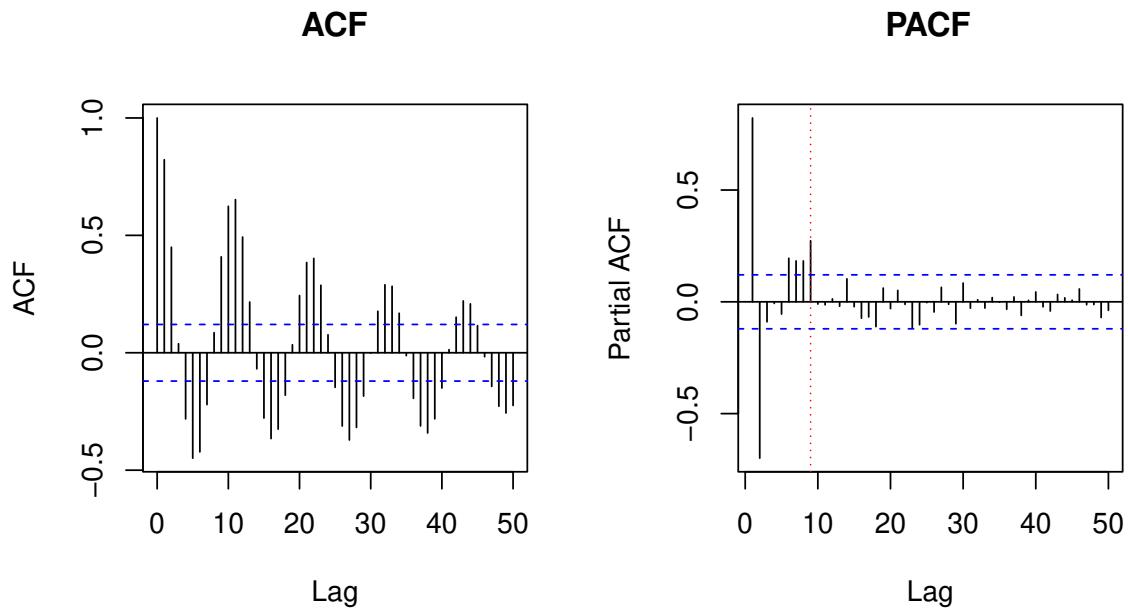


**Figure B.48.:** The monthly sunspot numbers since 1749 and yearly averaged values

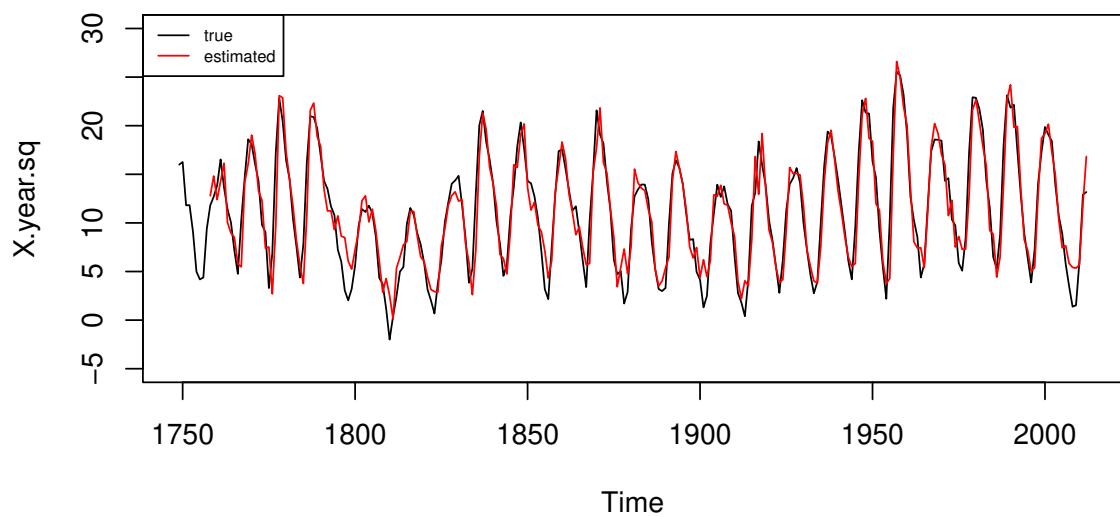


**Figure B.49.:** Squareroot transformed yearly time series

## Appendix B. R-Code

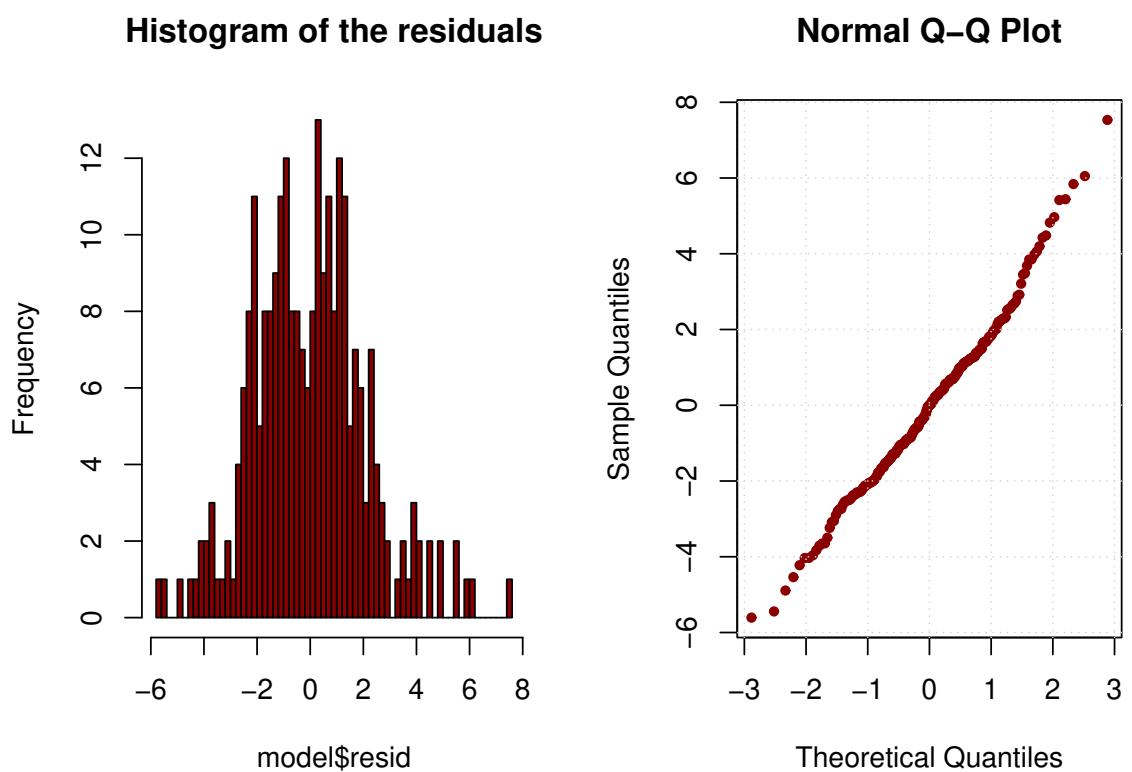


**Figure B.50.:** Autocorrelation and partial autocorrelation of transformed sunspot data



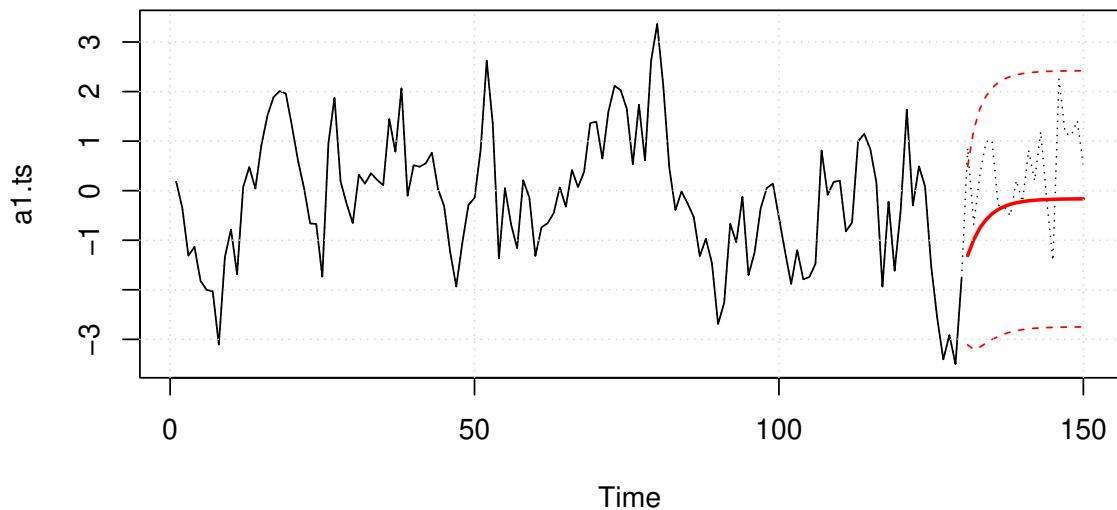
**Figure B.51.:** The annually averaged sunspot number data and the fitted model.

## Appendix B. R-Code

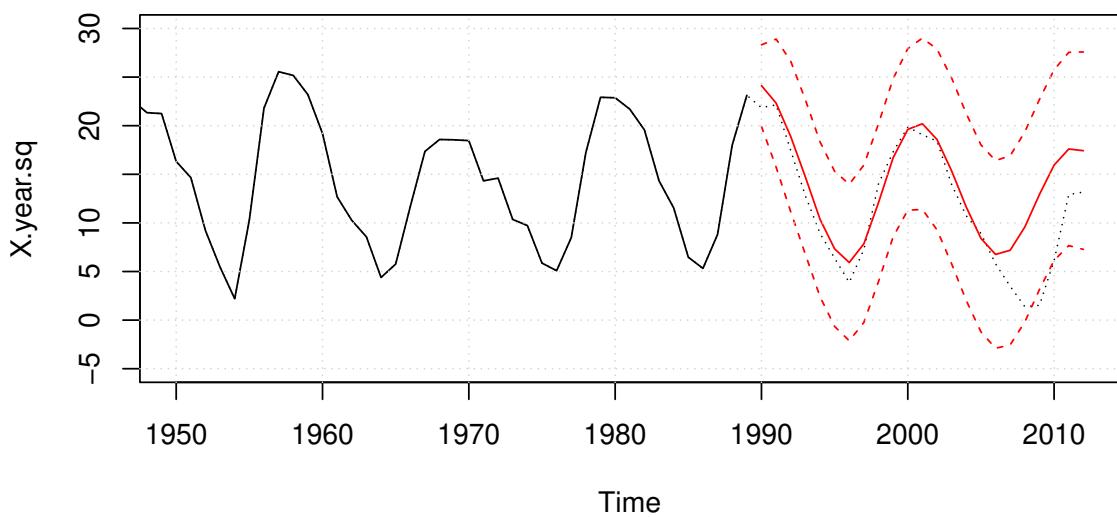


**Figure B.52.:** Residual analysis for the fitted model.

**Prediction of an AR(1) process**



**Figure B.53.:** A simulated AR(1) process with 150 observations, including a prediction.



**Figure B.54.:** Sunspot numbers from 1950 to 2014, showing training set and prediction