Solutions[1]

Problems of Chap. 2

2.1 Data Input

Listing 14.1 S2_python.py

```
''' Solution to Exercise "Data Input" '''

# author: Thomas Haslwanter, date: Oct-2015

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import urllib
import io
import zipfile

def getDataDobson(url, inFile):
    '''Extract data from a zipped-archive'''

    # get the zip-archive
    GLM_archive = urllib.request.urlopen(url).read()

    # make the archive available as a byte-stream
    zipdata = io.BytesIO()
    zipdata.write(GLM_archive)

    # extract the requested file from the archive, as a
        pandas XLS-file
```

[1]Published with the kind permission of © Thomas Haslwanter 2015. Published under the Creative Commons Attribution-ShareAlike International License (CC BY-SA 4.0).

T. Haslwanter, *An Introduction to Statistics with Python*, Statistics and Computing,
DOI 10.1007/978-3-319-28316-6

```
    myzipfile = zipfile.ZipFile(zipdata)
    xlsfile = myzipfile.open(inFile)

    # read the xls-file into Python, using Pandas, and return
        the extracted data
    xls = pd.ExcelFile(xlsfile)
    df  = xls.parse('Sheet1', skiprows=2)

    return df

if __name__ == '__main__':
    # 1.1 Numpy --------------------
    # Read in a CSV file, and show the top:
    inFile1 = 'swim100m.csv'
    data = pd.read_csv(inFile1)
    print(data.head())

    # Read in an excel file, and show the bottom
    inFile2 = 'Table 2.8 Waist loss.xls'

    xls = pd.ExcelFile(inFile2)
    data = xls.parse('Sheet1', skiprows=2)
    print(data.tail())

    # Read in a zipped data-file from the WWW
    url = r'http://cdn.crcpress.com/downloads/C9500/GLM_data.
        zip'
    inFile = r'GLM_data/Table 2.8 Waist loss.xls'

    df = getDataDobson(url, inFile)
    print(df.tail())
```

2.2 First Steps with *pandas*

Listing 14.2 S2_pandas.py

```
''' Solution to Exercise "First Steps with pandas":
Generate a sine and cosine wave using pandas' DataFrames,
and write them to an out-file.
'''

# author: Thomas Haslwanter, date: Sept-2015

import numpy as np
import pandas as pd

# Set the parameters
rate = 10
dt = 1/rate
freq = 1.5

# Derived quantities
omega = 2*np.pi*freq
```

```
# Generate the data
t = np.arange(0,10,dt)
y = np.sin(omega*t)
z = np.cos(omega*t)

# Assemble them in a DataFrame
df = pd.DataFrame({'Time':t, 'YVals':y, 'ZVals':z})

# Show the top 5 values
print(df.head())

# Save lines 10-15 of the y- and z-values to an outfile
outfile = 'out.txt'
df[10:16][['YVals', 'ZVals']].to_csv(outfile)
print('Data written to {0}'.format(outfile))
input('Done')
```

Problems of Chap. 4

4.1 Displaying Data

Listing 14.3 S4_display.py

```
''' Solution for Exercise "Data Display"
Read in weight-data recorded from newborns, and analyze the
data based on the gender of the baby.'''

# author: Thomas Haslwanter, date: Oct-2015

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

def getData():
    '''Read in data from a text-file, and return them as
        labelled DataFrame'''

    # Set directory and infile
    dataDir = '.'
    inFile = 'babyboom.dat.txt'

    # Read and label the data
    os.chdir(dataDir)
    data = pd.read_csv(inFile, sep='[ ]*', header=None,
        engine='python',
                       names= ['TOB', 'sex', 'Weight', '
                           Minutes'])

    # Eliminate "Minutes", since this is redundant
    df = data[['Minutes', 'sex', 'Weight']]

    return(df)

def showData(df):
    '''Graphical data display'''

    # Show the data: first all of them ....
    plt.plot(df.Weight, 'o')

    plt.title('All data')
    plt.xlabel('Subject-Nr')
    plt.ylabel('Weight [g]')
    plt.show()

    # To make the plots easier to read, replace "1/2" with "
        female/male"
```

```
        df.sex = df.sex.replace([1,2], ['female', 'male'])

        # ... then show the grouped plots
        df.boxplot(by='sex')
        plt.show()

        # Display statistical information numerically
        grouped = df.groupby('sex')
        print(grouped.describe())

        # This is a bit fancier: scatter plots, with labels and
            individual symbols
        symbols = ['o', 'D']
        colors = ['r', 'b']

        fig = plt.figure()
        ax = fig.add_subplot(111)

        # "enumerate" provides a counter, and variables can be
            assigned names in one step if
        # the "for"-loop uses a tuple as input for each loop:
        for (ii, (sex, group)) in enumerate(grouped):
            ax.plot(group['Weight'], marker = symbols[ii],
                linewidth=0, color = colors[ii], label=sex)

        ax.legend()
        ax.set_ylabel('Weight [g]')
        plt.show()

        # Fancy finish: a kde-plot
        df.Weight = np.double(df.Weight)    # kdeplot requires
            doubles

        sns.kdeplot(grouped.get_group('male').Weight, color='b',
            label='male')
        plt.hold(True)
        sns.kdeplot(grouped.get_group('female').Weight, color='r'
            , label='female')

        plt.xlabel('Weight [g]')
        plt.ylabel('PDF(Weight)')
        plt.show()

    # Statistics: are the data normally distributed?
    def isNormal(data, dataType):
        '''Check if the data are normally distributed'''
        alpha = 0.05
        (k2, pVal) = stats.normaltest(data)
        if pVal < alpha:
            print('{0} are NOT normally distributed.'.format(
                dataType))
        else:
            print('{0} are normally distributed.'.format(dataType
                ))
```

```
def checkNormality(df):
    '''Check selected data vlaues for normality'''

    grouped = df.groupby('sex')

    # Run the check for male and female groups
    isNormal(grouped.get_group('male').Weight, 'male')
    isNormal(grouped.get_group('female').Weight, 'female')

if __name__ == '__main__':
    '''Main Program'''

    df = getData()
    showData(df)
    checkNormality(df)

    # Wait for an input before exiting
    input('Done - Hit any key to continue')
```

Problems of Chap. 6

6.1 Sample Standard Deviation

Listing 14.4 S6_sd.py

```
''' Solution to Exercise "Sample Standard Deviation" '''

# author: Thomas Haslwanter, date: Sept-2015

import numpy as np

x = np.arange(1,11)
print('The standard deviation of the numbers from 1 to 10 is
    {0:4.2f}'.format(np.std(x, ddof=1)))
```

6.2 Normal Distribution

Listing 14.5 S6_normDist.py

```
''' Solution to Exercise "Normal Distribution" '''

# author: Thomas Haslwanter, date: Sept-2015

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


# Generate a PDF, with a mean of 5 and a standard deviation
    of 3
nd = stats.norm(5,3)

# Generate 1000 data from this distribution
data = nd.rvs(1000)

# Standard error
se = np.std(data, ddof=1)/np.sqrt(1000)
print('The standard error is {0}'.format(se))

# Histogram
plt.hist(data)
plt.show()

# 95% confidence interval
print('95% Confidence interval: {0:4.2f} - {1:4.2f}'.format(
    nd.ppf(0.025), nd.ppf(0.975)))

# SD for hip implants
```

```
nd = stats.norm()
numSDs = nd.isf(0.0005)
tolerance = 1/numSDs
print('The required SD to fulfill both requirements = {0:6.4f
    } mm'.format(tolerance))
```

6.3 Other Continuous Distributions

Listing 14.6 S6_continuous.py

```
'''Solution for Exercise "Continuous Distribution Functions"
    '''

# author: Thomas Haslwanter, date: Oct-2015

import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

# T-distibution
    ----------------------------------------------------
# Enter the data
x = [52, 70, 65, 85, 62, 83, 59]
''' Note that "x" is a Python "list", not an array!
 Arrays come with the numpy package, and have to contain all
     elements of the same type.
 Lists can mix different types, e.g. "x = [1, 'a', 2]"
 '''

# Generate the t-distribution: note that the degrees of
    freedom is the
# length of the data minus 1.
# In Python, the length of an object x is given by "len(x)"
td = stats.t(len(x)-1)
alpha = 0.01

# From the t-distribution, you use the "PPF" function and
    multiply it with the standard error
tval = abs( td.ppf(alpha/2)*stats.sem(x) )
print('mean +/- 99%CI = {0:3.1f} +/- {1:3.1f}'.format(np.mean
    (x),tval))

# Chi2-distribution, with 3 DOF
    ------------------------------------------------
# Define the normal distribution
nd = stats.norm()

# Generate three sets of random variates from this
    distribution
numData = 1000
data1 = nd.rvs(numData)
data2 = nd.rvs(numData)
data3 = nd.rvs(numData)
```

```
# Show a histogram of the sum of the squares of these random
    data
plt.hist(data1**2+data2**2 +data3**2, 100)
plt.show()

# F-distribution
    ------------------------------------------------
apples1 = [110, 121, 143]
apples2 = [88, 93, 105, 124]
fval = np.std(apples1, ddof=1)/np.std(apples2, ddof=1)
fd = stats.distributions.f(len(apples1),len(apples2))
pval = fd.cdf(fval)
print('The p-value of the F-distribution = {0}.'.format(pval)
    )
if pval>0.025 and pval<0.975:
    print('The variances are equal.')
```

6.4 Discrete Distributions

Listing 14.7 S6_discrete.py

```
'''Solution for Exercise "Continuous Distribution Functions"
    '''

# author: Thomas Haslwanter, date: Sept-2015

from scipy import stats

# Binomial distribution
    ------------------------------------------------
# Generate the distribution
p = 0.37
n = 15
bd = stats.binom(n, p)

# Select the interesting numbers, and calculate the "
    Probability Mass Function" (PMF)
x = [3,6,10]
y = bd.pmf(x)

# To print the result, we use the "zip" function to generate
    pairs of numbers
for num, solution in zip(x,y):
    print('The chance of finding {0} students with blue eyes
        is {1:4.1f}%.'.format(num, solution*100))

# Poisson distribution
    ------------------------------------------------
# Generate the distribution.
# Watch out NOT to divide integers, as "3/4" gives "0" in
    Python 2.x!
prob = 62./(365./7)
pd = stats.poisson(prob)
```

```
# Select the interesting numbers, calculate the PMF, and
    print the results
x = [0,2,5]
y = pd.pmf(x)*100
for num, solution in zip(x,y):
    print('The chance of haveing {0} fatal accidents in one
        week is {1:4.1f}%.'.format(num,solution))

# The last line just makes sure that the program does not
    close, when it is run from the commandline.
input('Done! Thanks for using programs by thomas.')
```

Problems of Chap. 8

8.1 Comparing One or Two Groups

Listing 14.8 S8_twoGroups.py

```
'''Solution for Exercise "Comparing Groups" '''

# author: Thomas Haslwanter, date: Sept-2015

import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
import scipy as sp
import os

def oneGroup():
    '''Test of mean value of a single set of data'''

    print('Single group of data
        ========================================')

    # First get the data
    data = np.array([5260, 5470, 5640, 6180, 6390, 6515,
        6805, 7515, 7515, 8230, 8770], dtype=np.float)
    checkValue = 7725   # value to compare the data to

    # 4.1.1. Normality test
    # We don't need the first parameter, so we just assign
        the output to the dummy variable "_"
    (_, p) = stats.normaltest(data)
    if p > 0.05:
        print('Data are distributed normally, p = {0}'.format
            (p))

    # 4.1.2. Do the onesample t-test
    t, prob = stats.ttest_1samp(data, checkValue)
    if prob < 0.05:
        print('With the one-sample t-test, {0:4.2f} is
            significantly different from the mean (p={1:5.3f})
            .'.\
        format(checkValue, prob))
    else:
        print('No difference from reference value with
            onesample t-test.')

    # 4.1.3. This implementation of the Wilcoxon test checks
        for the "difference" of one vector of data from zero
    (_,p) = stats.wilcoxon(data-checkValue)
    if p < 0.05:
```

```
            print('With the Wilcoxon test, {0:4.2f} is
                significantly different from the mean (p={1:5.3f})
                .'.\
            format(checkValue, p))
        else:
            print('No difference from reference value with
                Wilcoxon rank sum test.')

    def twoGroups():
        '''Compare the mean of two groups'''

        print('Two groups of data
            ========================================')

        # Enter the data
        data1 = [76., 101., 66., 72., 88., 82., 79., 73., 76.,
            85., 75., 64., 76., 81., 86.]
        data2 = [64., 65., 56., 62., 59., 76., 66., 82., 91.,
            57., 92., 80., 82., 67., 54.]

        # Normality test
        print('\n Normality test
            ----------------------------------------------------')

        # To do the test for both data-sets, make a tuple with
            "(... , ...)", add a counter with
        # "enumerate", and and iterate over the set:
        for ii, data in enumerate((data1, data2)):
            (_, pval) = stats.normaltest(data)
            if pval > 0.05:
                print('Dataset # {0} is normally distributed'.
                    format(ii))

        # T-test of independent samples
        print('\n T-test of independent samples
            ----------------------------------------')

        # Do the t-test for independent samples
        t, pval = stats.ttest_ind(data1, data2)
        if pval < 0.05:
            print('With the T-test, data1 and data2 are
                significantly different (p = {0:5.3f})'.format(
                pval))
        else:
            print('No difference between data1 and data2 with T-
                test.')

        # Mann-Whitney test
            ----------------------------------------------------------
        print('\n Mann-Whitney test
            ----------------------------------------------------------
            ')
        # Watch out: the keyword "alternative" was introduced in
            scipy 0.17, with default"two-sided";
```

```
        if np.int(sp.__version__.split('.')[1]) > 16:
            u, pval = stats.mannwhitneyu(data1, data2,
                alternative='two-sided')
        else:
            u, pval = stats.mannwhitneyu(data1, data2)
            pval *= 2    # because the default was a one-sided p-
                value
        if pval < 0.05:
            print('With the Mann-Whitney test, data1 and data2
                are significantly different(p = {0:5.3f})'.format(
                pval))
        else:
            print('No difference between data1 and data2 with
                Mann-Whitney test.')

    if __name__ == '__main__':
        oneGroup()
        twoGroups()

        # And at the end, make sure the results are displayed,
            even if the program is run from the commandline
        input('\nDone! Thanks for using programs by thomas.\nHit
            any key to finish.')
```

8.2 Comparing Multiple Groups

Listing 14.9 S8_multipleGroups.py

```
''' Solution for Exercise "Comparing Multiple Groups" '''

# author: Thomas Haslwanter, date: Sept-2015

# Load the required modules
    #-----------------------------------------------------------
# Standard modules
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
import pandas as pd

# Modules for data-analysis
from statsmodels.formula.api import ols
from statsmodels.stats.anova import anova_lm
from statsmodels.stats import multicomp

# Module for working with Excel-files
import xlrd

def get_ANOVA_data():
    '''Get the data for the ANOVA'''

    # First we have to get the Excel-data into Python. This
        can be done e.g. with the package "xlrd"
```

```
    # You have to make sure that you select a valid location
        on your computer!
    inFile = 'Table 6.6 Plant experiment.xls'
    book = xlrd.open_workbook(inFile)
    # We assume that the data are in the first sheet. This
        avoids the language problem "Tabelle/Sheet"
    sheet = book.sheet_by_index(0)

    # Select the columns and rows that you want:
    # The "treatment" information is in column "E", i.e. you
        have to skip the first 4 columns
    # The "weight" information is in column "F", i.e. you
        have to skip the first 5 columns
    treatment = sheet.col_values(4)
    weight = sheet.col_values(5)

    # The data start in line 4, i.e. you have to skip the
        first 3
    # I use a "pandas" DataFrame, so that I can assign names
        to the variables.
    data = pd.DataFrame({'group':treatment[3:], 'weight':
        weight[3:]})

    return data

def do_ANOVA(data):
    '''4.3.2. Perform an ANOVA on the data'''

    print('ANOVA:
        ----------------------------------------------')

    # First, I fit a statistical "ordinary least square (ols)
        "-model to the data, using the
    # formula language from "patsy". The formula 'weight ~ C(
        group)' says:
    # "weight" is a function of the categorical value "group"
    # and the data are taken from the DataFrame "data", which
         contains "weight" and "group"
    model = ols('weight ~ C(group)', data).fit()

    # "anova_lm" (where "lm" stands for "linear model")
        extracts the ANOVA-parameters
    # from the fitted model.
    anovaResults = anova_lm(model)
    print(anovaResults)

    if anovaResults['PR(>F)'][0] < 0.05:
        print('One of the groups is different.')

def compare_many(data):
    '''Multiple comparisons: Which one is different? '''

    print('\n MultComp:
        ---------------------------------------')
```

```
# An ANOVA is a hypothesis test, and only answers the
    question: "Are all the groups
# from the same distribution?" It does not tell you which
     one is different.
# Since we now compare many different groups to each
    other, we have to adjust the
# p-values to make sure that we don't get a Type I error.
     For this, we use the
# statscom module "multicomp"
mc = multicomp.MultiComparison(data['weight'], data['
    group'])

# There are many ways to do multiple comparisons. Here,
    we choose
# "Tukeys Honest Significant Difference" test
# The first element of the output ("0") is a table
    containing the results
print(mc.tukeyhsd().summary())

# Show the group names
print(mc.groupsunique)

# Generate a print ----------------

res2 = mc.tukeyhsd()      # Get the data

simple = False
if simple:
    # You can do the plot with a one-liner, but then this
         does not - yet - look that great
    res2.plot_simultaneous()
else:
    # Or you can do it the hard way, i.e. by hand:

    # Plot values and errorbars
    xvals = np.arange(3)
    plt.plot(xvals, res2.meandiffs, 'o')
    errors = np.ravel(np.diff(res2.confint)/2)
    plt.errorbar(xvals, res2.meandiffs, yerr=errors, fmt=
        'o')

    # Set the x-limits
    xlim = -0.5, 2.5
    # The "*xlim" passes the elements of the variable "
        xlim" elementwise into the function "hlines"
    plt.hlines(0, *xlim)
    plt.xlim(*xlim)

    # Plot labels (this is a bit tricky):
    # First, "np.array(mc.groupsunique)" makes an array
        with the names of the groups;
    # and then, "np.column_stack(res2[1][0])]" puts the
        correct groups together
```

```
            pair_labels = mc.groupsunique[np.column_stack(res2.
                _multicomp.pairindices)]
            plt.xticks(xvals, pair_labels)

            plt.title('Multiple Comparison of Means - Tukey HSD,
                FWER=0.05' +
            '\n Pairwise Mean Differences')

        plt.show()

    def KruskalWallis(data):
        '''Non-parametric comparison between the groups'''

        print('\n Kruskal-Wallis test
            ----------------------------------------------------')

        # First, I get the values from the dataframe
        g_a = data['weight'][data['group']=='TreatmentA']
        g_b = data['weight'][data['group']=='TreatmentB']
        g_c = data['weight'][data['group']=='Control']

        #Note: this could also be accomplished with the "groupby"
             function from pandas
        #groups = pd.groupby(data, 'group')
        #g_a = groups.get_group('TreatmentA').values[:,1]
        #g_c = groups.get_group('Control').values[:,1]
        #g_b = groups.get_group('TreatmentB').values[:,1]

        # Then do the Kruskal-Wallis test
        h, p = stats.kruskal(g_c, g_a, g_b)
        print('Result from Kruskal-Wallis test: p = {0}'.format(p
            ))

    if __name__ == '__main__':
        data = get_ANOVA_data()
        do_ANOVA(data)
        compare_many(data)
        KruskalWallis(data)

        input('\nThanks for using programs by Thomas!\nHit any
            key to finish')
```

Problems of Chap. 9

9.1 A Lady Drinking Tea

Listing 14.10 S9_fisherExact.py

```
'''Solution for Exercise "Categorical Data"
"A Lady Tasting Tea"
'''

# author: Thomas Haslwanter, date: Sept-2015

from scipy import stats
obs = [[3,1], [1,3]]
_, p = stats.fisher_exact(obs, alternative='greater')

#obs2 = [[4,0], [0,4]]
#stats.fisher_exact(obs2, alternative='greater')

print('\n--- A Lady Tasting Tea (Fisher Exact Test) ---')
print('The chance that the lady selects 3 or more cups
    correctly by chance is {0:5.3f}'.format(p))
```

9.2 Chi2 Contingency Test

Listing 14.11 S9_chi2Contingency.py

```
'''Solution for Exercise "Categorical Data":
Chi2-test with frequency tables
'''

# author: Thomas Haslwanter, date: Sept-2015

from scipy import stats

obs = [[36,14], [30,25]]
chi2, p, dof, expected = stats.chi2_contingency(obs)

print('--- Contingency Test ---')
if p < 0.05:
    print('p={0:6.4f}: the drug affects the heart rate.'.
        format(p))
else:
    print('p={0:6.4f}: the drug does NOT affect the heart
        rate.'.format(p))

obs2 = [[36,14], [29,26]]
chi2, p, dof, expected = stats.chi2_contingency(obs2)
chi2, p2, dof, expected = stats.chi2_contingency(obs2,
    correction=False)
```

```
print('If the response in 1 non-treated person were different
    , \n we would get p={0:6.4f} with Yates correction, and p
    ={1:6.4f} without.'.format(p, p2))
```

9.3 Chi2 Oneway Test

Listing 14.12 S9_chi2OneWay.py

```
'''Solution for Exercise "Categorical Data" '''

# author: Thomas Haslwanter, date: Sept-2015

from scipy import stats

# Chi2-oneway-test
obs = [4,6,14,10,16]
_, p = stats.chisquare(obs)

print('\n--- Chi2-oneway ---')
if p < 0.05:
    print('The difference in opinion between the different
        age groups is significant (p={0:6.4f})'.format(p))
else:
    print('The difference in opinion between the different
        age groups is NOT significant (p={0:6.4f})'.format(p))

print('DOF={0:3d}'.format(len(obs)-1))
```

9.4 McNemar Test

Listing 14.13 S9_mcNemar.py

```
'''Solution for Exercise "Categorical Data"
McNemar's Test
'''

# author: Thomas Haslwanter, date: Sept-2015

from scipy import stats
from statsmodels.sandbox.stats.runs import mcnemar

obs = [[19,1], [6, 14]]
obs2 = [[20,0], [6, 14]]

_, p = mcnemar(obs)
_, p2 = mcnemar(obs2)

print('\n--- McNemar Test ---')
if p < 0.05:
    print('The results from the neurologist are significanlty
         different from the questionnaire (p={0:5.3f}).'.
        format(p))
```

```
else:
    print('The results from the neurologist are NOT
        significanlty different from the questionnaire (p
        ={0:5.3f}).'.format(p))

if (p<0.05 == p2<0.05):
    print('The results would NOT change if the expert had
        diagnosed all "sane" people correctly.')
else:
    print('The results would change if the expert had
        diagnosed all "sane" people correctly.')
```

Problems of Chap. 11

11.1 Correlation

Listing 14.14 S11_correlation.py

```
'''Solution for Exercise "Correlation" in Chapter 11 '''

# author: Thomas Haslwanter, date: Oct-2015

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

def getModelData(show=True):
    ''' Get the data from an Excel-file '''

    # First, define the in-file and get the data
    in_file = 'AvgTemp.xls'

    # When the data are neatly organized, they can be read in
        directly with the pandas-function:
    # with "ExcelFile" you open the file
    xls = pd.ExcelFile(in_file)

    # and with "parse" you get get the data from the file,
       from the specified Excel-sheet
    data = xls.parse('Tabelle1')

    if show:
        data.plot('year', 'AvgTmp')
        plt.xlabel('Year')
        plt.ylabel('Average Temperature')
        plt.show()

    return data

if __name__=='__main__':
    data = getModelData()

    # Correlation
        ----------------------------------------------------
    # Calculate and show the different correlation
       coefficients
    print('Pearson correlation coefficient: {0:5.3f}'.format(
         data['year'].corr(data['AvgTmp'], method = 'pearson')
         ))
    print('Spearman correlation coefficient: {0:5.3f}'.format
        ( data['year'].corr(data['AvgTmp'], method = 'spearman
        ') ))
```

```
    print('Kendall tau: {0:5.3f}'.format( data['year'].corr(
        data['AvgTmp'], method = 'kendall') ))
```

11.1 Regression

Listing 14.15 S11_regression.py

```
'''Solution for Exercise "Regression" '''

# author: Thomas Haslwanter, date: Sept-2015

import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
import seaborn as sns
import statsmodels.formula.api as sm

# We don't need to invent the wheel twice ;)
from S11_correlation import getModelData

if __name__== '__main__':

    # get the data
    data = getModelData(show=False)

    # Regression
        ------------------------------------------------------
    # For "ordinary least square" models, you can do the
        model directly with pandas
    #model = pd.ols(x=data['year'], y=data['AvgTmp'])

    # or you can use the formula-approach from statsmodels:
    # offsets are automatically included in the model
    model = sm.ols('AvgTmp ~ year', data)
    results = model.fit()
    print(results.summary())

    # Visually, the confidence intervals can be shown using
        seaborn
    sns.lmplot('year', 'AvgTmp', data)
    plt.show()

    # Is the inclination significant?
    ci = results.conf_int()

    # This line is a bit tricky: if both are above or both
        below zero, the product is positive:
    # we look at the coefficient that describes the
        correlation with "year"
    if np.prod(ci.loc['year'])>0:
        print('The slope is significant')
```

11.1 Normality Check

Listing 14.16 S11_normCheck.py

```
'''Solution for Exercise "Normality Check" in Chapter 11 '''

# author: Thomas Haslwanter, date: Sept-2015

from scipy import stats
import matplotlib.pyplot as plt
import statsmodels.formula.api as sm
import seaborn as sns

# We don't need to invent the wheel twice ;)
from S11_correlation import getModelData

if __name__== '__main__':

    # get the data
    data = getModelData(show=False)

    # Fit the model
    model = sm.ols('AvgTmp ~ year', data)
    results = model.fit()

    # Normality check
        ------------------------------------------------------
    res_data = results.resid    # Get the values for the
        residuals

    # QQ-plot, for a visual check
    stats.probplot(res_data, plot=plt)
    plt.show()

    # Normality test, for a quantitative check:
    _, pVal = stats.normaltest(res_data)
    if pVal < 0.05:
        print('WARNING: The data are not normally distributed
             (p = {0})'.format(pVal))
    else:
        print('Data are normally distributed.')
```

Glossary

bias Systematic deviation of a sample statistic from the corresponding population statistic. Often caused by poor selection of subjects.

blocking To reduce the variability of a variable that cannot be randomized by fixating it.

box-plot A common visualization of the distribution of data, expressed by a box with a line inside that box, and whiskers at the top and bottom. The box indicates the first and third quartile, and the line the median value of the data sample. The whiskers can indicate either the range of the data or the most extreme value within 1.5*the inner-quartile-range.

case–control study A type of observational study in which two existing groups differing in outcome are identified and compared on the basis of some supposed causal attribute. ("First treat, then select.")

categorical data Data that can take on one of a limited, and usually fixed, number of possible values, with no natural order. (If a "mean value" makes no sense.)

cohort study A type of observational study, where you first select the patients, and then follow their development. For instance, in medicine a cohort study starts with an analysis of risk factors. Then the study follows a group of people who do not have the disease. Finally, correlations are used to determine the absolute risk of subject contraction. ("First select, then treat.")

confidence interval Interval estimate of a population parameter, which contains the true value of the parameter with a defined percent-likelihood (e.g., 95 %-CI).

correlation Any departure of two or more random variables from independence.

crossover study A longitudinal study in which all subjects receive a sequence of different treatments.

T. Haslwanter, *An Introduction to Statistics with Python*, Statistics and Computing,
DOI 10.1007/978-3-319-28316-6

cumulative distribution function The probability to find a random variable with a value lower than the given one.

degrees of freedom The number of degrees of freedom is the number of values in the final calculation of a statistic that are free to vary.

density A continuous function that describes the relative likelihood for a random variable to take on a given value. For example *kernel-density* estimation (KDE), or *probability-density* function (PDF).

design matrix The data matrix $\mathbf{X}$ in the regression model $y = \mathbf{X} \cdot \boldsymbol{\beta} + \epsilon$.

distribution A function which assigns a probability to each measurable subset of the possible outcomes of a random experiment.

experimental study Study where the selection of the subjects as well as the conditions of the study are under the control of the investigator.

factor Also called *treatment* or *independent variable*, is an explanatory variable manipulated by the experimenter.

function A *Python* object that accepts input data, executes commands and calculations with them, and can return one or more return objects.

hypothesis test A method of statistical inference used for testing a statistical hypothesis.

IPython The *IPython* kernel is an interactive shell that allows the immediate execution of *Python* commands, and extends these with "magic functions" (e.g., "%cd," or "%whos") which facilitate the interactive work. It has been so successful that its development has been split into two separate projects: *IPython* now handles the computational kernel; and *Jupyter* provides the frontend. This allows *IPython* to be executed in a terminal modus (which is rarely used on its own, but allows it to embed *IPython* in other applications), in a *Jupyter QtConsole* which allows the display of help and the graphical outputs in the command window, and in a *Jupyter notebook*, a browser-based implementation with support for code, rich text, mathematical expressions, inline plots, and other rich media.

kurtosis Measure of the peakedness of a distribution. It is approximately 3 for normally distributed data. "Excess kurtosis" is the kurtosis relative to the normal distribution.

linear regression Modeling a scalar variable (*dependent variable*) using a linear predictor function. The unknown model parameters are estimated from the data.

Simple linear regression: $y = k * x + d$.
Multiple linear regression: $y = k_1 * x_1 + k_2 * x_2 + \ldots + k_n * x_n + d$.

location Parameter that shifts the mean of a probability distribution.

logistic regression Also called *logit regression*.The probabilities describing the possible outcomes of a single trial are modeled, as a function of the explanatory (predictor) variables, using a logistic function: $f(x) = \frac{L}{1+e^{-k(x-x_0)}}$.

Markov Chain Stochastic model of a process where the probability of each state only depends on the previous state.

Matplotlib A *Python* package which provides the ability to generate 2D and 3D plots. This includes the plotting commands, as well as the functionality of different output media, also called *backends*: an output can for example be into the *Jupyter Notebook*, into a PDF file, or into a separate graphics window.

maximum likelihood For a fixed set of data and an underlying statistical model, the method of maximum likelihood selects the set of values of the model parameters that maximizes the likelihood function. Intuitively, this maximizes the "agreement" of the selected model with the observed data, and for discrete random variables it indeed maximizes the probability of the observed data under the resulting distribution.

median value The value separating the higher half of the data sample from the lower half.

minimization Closely related to *randomization*. Thereby one takes whichever treatment has the smallest number of subjects, and allocates this treatment with a probability greater than 0.5 to the next patient.

mode value The highest value in a discrete or continuous probability distribution.

module A file containing *Python* variables and function definitions.

Monte Carlo Simulation Repeated simulation of the behavior of some parameter based on repeated sampling of a random variable.

numerical data Data that can be expressed by a (continuous or discrete) number.

numpy *Python* package for numerical data manipulation. Provides efficient handling of mathematical manipulations on vectors and on arrays of two or more dimensions.

observational study Study where the assignment of subjects into a treated group versus a control group is outside the control of the investigator.

paired test Two data sets are *paired* when the following one-to-one relationship exists between values in the two data sets: (1) Each data set has the same number of data points. (2) Each data point in one data set is related to one, and only one, data point in the other data set.

percentile Also called *centile*. Value that is larger or equal than $p\%$ of the data, where $1 <= p < 100$.

population Includes all of the elements from a set of data.

power analysis Calculation of the minimum sample size required so that one can be reasonably likely to detect an effect of a given size.

power Same as *sensitivity*. Denoted by $1 - \beta$, with β the probability of Type II errors.

probability density function (PDF) A continuous function which defines the likelihood to find a random variable with a certain value. The probability to find the value of the random variable in a given interval is the integral of the probability density function over that interval.

probability mass function (PMF) A discrete function which defines the probability to obtain a given number of events in an experiment or observation.

prospective study A prospective study watches for outcomes, such as the development of a disease, during the study period and relates this to other factors such as suspected risk or protection factor(s).

pylab pylab is a convenience *Python* module that bulk imports `matplotlib.pyplot` (for plotting) and `numpy` (for mathematics and working with arrays) in a single name space.

package A folder containing one or more *Python* modules and an ".ini"-file.

quantile Value that is larger or equal than $p*100\,\%$ of the data, where $0 < p <= 1$.

quartile Value that is larger or equal than $25\,\%/50\,\%/75\,\%$ of the data (first/second/third quartile). The 2nd quartile is equivalent to the median.

randomization A method to eliminate bias from research results, by dividing a homogeneous group of subjects into a *control group* (which receives no treatment) and a *treatment group*.

ranked data Numbered data where the number corresponds to the rank of the data, i.e., the number in the sequence of the sorted data, and not to a continuous value.

regular expression A sequence of characters that define a search pattern, mainly for use in pattern matching with strings. Available for Unix, and for *Python*, *Perl*, *Java*, *C++*, and many other programming languages.

residual Difference between the observed value and the estimated function value.

retrospective study A retrospective study looks backwards and examines exposures to suspected risk or protection factors in relation to an outcome that is established at the start of the study.

sample One or more observations from a population.

scale Parameter that controls the variance of a probability distribution.

scipy *Python* package for scientific computation. Based on *numpy*.

sensitivity Proportion of actual positives which are correctly identified as such (e.g., the percentage of sick people who are correctly identified as having the condition).

shape parameter Parameters beyond *location* and *scale* which control the shape of a probability distribution. Rarely used.

skewness Measure of the asymmetry of a distribution.

specificity Proportion of actual negatives which are correctly identified as such (e.g., the percentage of healthy people who are correctly identified as not having the condition).

standard deviation Square root of variance.

standard error Often short for *standard error of the mean*. Square root of the variance of a statistic.

treatment Same as *factor*.

type I error A type I error occurs when one rejects the null hypothesis when it is true. The probability of a type I error is the level of significance of the hypothesis test, and is commonly denoted by α.

type II error A type II error occurs when one rejects the alternative hypothesis (fails to reject the null hypothesis) when the alternative hypothesis is true. Therefore it is dependent on an alternative hypothesis. The probability of a type II error is commonly denoted by β.

unpaired test Test with two sets of independent data.

variance Measure of how far a set of numbers is spread out. Mathematically, it is the expected value of the squared deviation from the mean: $Var(X) = E[(X - \mu)^2]$. The variance of a sample gives an estimate of the population variance that is biased by a factor of $\frac{n-1}{n}$. The best unbiased estimate of the population variance is therefore given by $s^2 = \frac{1}{n-1} \sum_{i=1}^{n} (y_i - \bar{y})^2$ which is called *(unbiased) sample variance*.

References

Altman, D. G. (1999). *Practical statistics for medical research*. New York: Chapman & Hall/CRC.

Amess, J. A., Burman, J. F., Rees, G. M., Nancekievill, D. G., & Mollin, D. L. (1978). Megaloblastic haemopoiesis in patients receiving nitrous oxide. *Lancet*, *2*(8085):339–342.

Bishop, C. M. (2007). *Pattern recognition and machine learning*. New York: Springer.

Box, J. F. (1978). *R. A. Fisher: The life of a scientist*. New York: Wiley.

Dobson, A. J., & Barnett, A. (2008). *An introduction to generalized linear models* (3rd ed.). Boca Raton: CRC Press.

Duda, R. O., Hart, P. E., & Stork, D. G. (2004). *Pattern classification* (2nd ed.). Hoboken: Wiley-Interscience.

Ghasemi, A., & Zahediasl, S. (2012). Normality tests for statistical analysis: a guide for non-statisticians. *International Journal of Endocrinology and Metabolism, 10*(2):486–489. doi:10.5812/ijem.3505. http://dx.doi.org/10.5812/ijem.3505

Harms, D., & McDonald, K. (2010). *The quick python book* (2nd ed.). Greenwich: Manning Publications Co.

Holm, S. (1979). A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics, 6*:65–70.

Kaplan, D. (2009). *Statistical modeling: A fresh approach*. St Paul: Macalester College.

Kaplan, R. M., & Irvin, V. L. (2015). Likelihood of null effects of large nhlbi clinical trials has increased over time. *PLoS One, 10*(8):e0132382. doi:10.1371/journal.pone.0132382. http://dx.doi.org/10.1371/journal.pone.0132382

Klamroth-Marganska, V., Blanco, J., Campen, K., Curt, A., Dietz, V., Ettlin, T., Felder, M., Fellinghauer, B., Guidali, M., Kollmar, A., Luft, A., Nef, T., Schuster-Amft, C., Stahel, W., & Riener, R. (2014). Three-dimensional, task-specific robot therapy of the arm after stroke: a multicentre, parallel-group randomised trial. *The Lancet Neurology, 13*(2):159–166. doi:10.1016/S1474-4422(13)70305-3. http://dx.doi.org/10.1016/S1474-4422(13)70305-3

McCullagh, P. (1980). Regression models for ordinal data. *Journal of the Royal Statistical Society. Series B (Methodological), 42*(2):109–142.

McCullagh, P. & Nelder, J. A. (1989). *Generalized linear models* (2nd ed.). New York: Springer.

Nuzzo, R. (2014). Scientific method: Statistical errors. *Nature, 506*(7487):150–152. doi:10.1038/506150a. http://www.nature.com/news/scientific-method-statistical-errors-1.14700

Olson, R. (2012). *Statistical analysis made easy in python*. http://www.randalolson.com/2012/08/06/statistical-analysis-made-easy-in-python/

T. Haslwanter, *An Introduction to Statistics with Python*, Statistics and Computing,
DOI 10.1007/978-3-319-28316-6

Open Science Collaboration (OSC). (2015). Psychology. Estimating the reproducibility of psychological science. *Science, 349*(6251):aac4716. doi:10.1126/science.aac4716. http://dx.doi.org/10.1126/science.aac4716

Pilon, C. D. (2015). *Probabilistic programming and Bayesian methods for hackers*. http://camdavidsonpilon.github.io/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers/

Riffenburgh, R. H. (2012). *Statistics in medicine* (3rd ed.). Amsterdam: Academic Press.

Rosenbaum, P. R., & Rubin, D. B. (1983). The central role of the propensity score in observational studies for causal effects. *Biometrika, 70*(1):41–55.

Scopatz, A., & Huff, K. D. (2015). *Effective computation in physics*. Sebastopol: O'Reilly Media.

Sellke, T., Bayarri, M. J., & Berger, J. O. (2001). Calibration of p values for testing precise null hypotheses. *The American Statistician, 55*:62–71.

Sheppard, K. (2015). *Introduction to python for econometrics, statistics and data analysis*. http://www.kevinsheppard.com/images/0/09/Python_introduction.pdf

Wilkinson, G. N., & Rogers, C. E. (1973). Symbolic description of factorial models for analysis of variance. *Applied Statistics, 22*:, 392–399.

Index

T. Haslwanter, *An Introduction to Statistics with Python*, Statistics and Computing,
DOI 10.1007/978-3-319-28316-6