Predictive Modeling

Series 1

Exercise 1.1

This exercise aims at carrying out a simple regression analysis for the data set constructed by Frank Anscombe. The data you need for this exercise is called **anscombe** and consists of four response variables y_i and four predictors x_i. Consider the four models $Y_i^k = \beta_0^k + \beta_1^k \cdot X_i^k + \epsilon_i$ for $k = 1, ..., 4$.

a) Determine for all four models the intercept and the slope of the least squares regression line and their standard errors. Determine as well $\hat{\sigma}$.

Python-Hints: The dataset should be provided in **.csv**. To load the data, use **pandas.read()** and make sure you provide the correct file location. For example:

[1]:
```
import pandas as pd
anscombe = pd.read_csv('./data/anscombe.csv')
```

The linear regression can be performed with **statsmodels.api.OLS(y, x)**:

[2]:
```
import statsmodels.api as sm

x_set = sm.add_constant(anscombe.loc[:, 'x1'])
y_set = anscombe.loc[:, 'y1']

model = sm.OLS(y_set, x_set).fit()
model.params
```

b) Plot the regression line for all four models in a scatter plot. Comment your observations of the results in a) and b).

Python-Hints:

[3]:
```
import matplotlib.pyplot as plt

# Create figure and subfigures:
fig = plt.figure(figsize=(12, 12))
# Create axes in subplots
ax = fig.add_subplot(2, 2, 1)
# Plot scatter data
ax.plot(anscombe.loc[:, 'x1'], anscombe.loc[:, 'y1'], 'ok')
```

```
# show plot
plt.show()
```

Exercise 1.2

Prices of antique clocks: McClave and Benson collected data on the basis of auctions about age and price of antique clocks. You find them in the data file `antique_clocks.csv`.

a) Display the data as a scatter plot (price vs. age) and describe their functional dependence.

b) Use a linear model to describe the relationship between `price` and `age` and determine the estimated coefficients.

c) Draw a regression line in the scatter plot in a). Comment on the results.

Exercise 1.3

An engineer intends to carry out an analysis of a windmill used for power generation. He collects data about the produced current (in Ampere) at different wind speeds (meter per second). You'll find the data in the file `windmill.csv`. (Source: Montgomery and Peck, *Introduction to Linear Regression Analysis*, Wiley.)

a) Generate a scatter plot (current (y-axis) vs. wind speed) and another scatter plot (current vs. $\frac{1}{\text{wind speed}}$). What do you observe?

b) Use the least squares method to fit the model

$$\text{current} \approx \beta_0 + \beta_1 x \qquad\qquad \text{with } x = \frac{1}{\text{wind speed}}$$

Determine the corresponding estimated coefficients and standard errors.

c) Determine a 99 % confidence interval for β_1. The confidence interval can be found using: `OLSResults.conf_int(alpha=0.01)`, where `OLSResults` is your fitted model.

d) Generate a scatter plot (current vs. wind speed). How do you interpret the coefficients β_0 and β_1 in these plots ?
Hint: Let the wind speed approach infinity to interpret β_0 and set the current to zero in order to interpret β_1. A sketch may be useful.

e) Determine the expected value, a 95 % confidence interval for the expected current and a 95 % prediction interval at wind speeds of $1\frac{\mathrm{m}}{\mathrm{s}}$ and $10\frac{\mathrm{m}}{\mathrm{s}}$. Comment on the results. `Python`-Hints:

   a) You have to add a constant to your one-dimensional prediction vector `x`, in a similar way you do to `x` before fitting.

   b) You can create a prediction results instance using `OLSResults.get_prediction()`, where `OLSResults` is your fitted model.

   c) You can now access all results using `PredictionResults.summary_frame()`

   `Python`-code example:

[1]:
```
''' Hints: '''
x0 = [1]
x0 = sm.add_constant(x0)

# Prediction
pred0 = model.get_prediction(x0)
pred0 = pred0.summary_frame(alpha=0.05)
```

Exercise 1.4

In the middle of the 19th century, Scottish physicist James D. Forbes worked on a method to determine the altitude using the boiling point of water. It was known that the altitude can be determined by means of the air pressure. That is the reason why Forbes was interested in a relation between the boiling point of water and the air pressure. The data for this exercise originates from his work published in 1857. `Forbes.csv` contains the boiling point `y` (in Fahrenheit) and the air `pressure` (in inch of mercury) at 17 places in the Alps and in Scotland. (Source: S. Weisberg, *Applied Linear Regression*, Wiley (1985), p. 3)

a) Add the variable `x` $= 100 \cdot \log($`pressure`$)$ to the data frame `Forbes` and plot `y` versus `pressure` and `y` versus `x`. Comment on your observations with respect to the two plots.

b) Use a least squares fit to determine the regression line for `y` versus `x`. Have a look at the regression line in the scatter plot and describe your observations of the result.

c) Use a least squares fit to determine the regression line for `y` versus `x`, but now omit the 12th observation. Compare the values $\hat{\beta}_0, \hat{\beta}_1, \mathrm{se}(\hat{\beta}_0), \mathrm{se}(\hat{\beta}_1)$ and $\hat{\sigma}$ with the ones you have found in part b).

Python-Hints:

[1]:
```
x = x.drop(11)
```

In the following exercises, we keep the 12th observation omitted.

d) Test $H_0 : \beta_1 = 0$ versus $H_A : \beta_1 \neq 0$ in the model $Y_i = \beta_0 + \beta_1 x_i + \epsilon_i$ using the output of the regression analysis at the 5 %-level.

e) Determine a 95 %-confidence interval for the slope β_1.

f) Determine the expected value of Y given the predictor value $x_0 = 100 \cdot \log(26) = 325.81$. Determine a 95 % and a 99 % confidence interval for $E[Y|x_0]$.

Voluntary exercise: Plot a 99 % confidence band in the scatter plot.

g) Determine a 99 % prediction interval for the observed value of Y for $x_0 = 325.81$. Compare this interval with the confidence interval you have found in exercise f).

Exercise 1.5

We would like to simulate the distribution of the estimated coefficient values β_0 and β_1. Our model is $Y_i = 4 + 2x_i + \epsilon_i$ with the following x_i values:

i	1	2	3	4	5	6	7	8	9	10
x_i	0	3	4	8	10	11	13	16	17	20

The measurement errors ϵ_i are normally distributed with $\mu = 0$ and $\sigma^2 = 2$.

a) Simulate the 10 values of Y_i on the basis of the model $Y_i = 4 + 2x_i + \epsilon_i$ one hundred times and estimate the values of the regression coefficients β_0 and β_1.

Python-Hint:

[ ]:
```
import numpy as np
import statsmodels.api as sm
from scipy.stats import norm

# Set random seed
np.random.seed(0)
# Set number of random simulations
n = 100
# xi as given
x_i = np.array([0, 3, 4, 8, 10, 11, 13, 16, 17, 20])
# random error, taken from normal distribution:
e_i = norm.rvs(loc=?, scale=?, size=?) # Set accordingly
```

```
# Find Y_i for every n
# Perform linear regression
# Save Regression coefficients
```

b) Have a look at the distribution of the estimated regression coefficients by means of a histogram and a normal plot. Comment on your observations. Have a look at the joint distribution of the regression coefficients by means of a scatter plot. **Python**-Hints:

   a) You can plot a histogram using `plt.hist(x)`.

   b) The theoretical quantities for the Normal plot can be found with `norm.ppf()`

c) Determine the mean value of the 100 estimations of β_0 and β_1. Determine as well their variance. Compare the results with the theoretical values.

Result Checker

E 1.4:

e) $[0.4813, 0.4930]$

f) $[205.099, 205.246]$ and $[205.070, 205.275]$

g) $[204.780, 205.566]$

Predictive Modeling

Solutions to Series 1

Solution 1.1

a) `Python` code:

[1]:
```
import pandas as pd
import numpy as np

# Read Data: make sure you have downloaded the datafile and placed it
#  in a folder named data, in the same directory as this notebook
anscombe = pd.read_csv('./data/anscombe.csv')
```

[2]:
```
# Define x and y:
x = anscombe[['x1', 'x2', 'x3', 'x4']]
y = anscombe[['y1', 'y2', 'y3', 'y4']]

''' Solution using Statsmodels.api '''
import statsmodels.api as sm

output = np.zeros((2, 4))
for d_set in range(4):
# define x and y in a fitting format
x_set = sm.add_constant(x.iloc[:, d_set])
y_set = y.iloc[:, d_set]

# Fit the linear model
model = sm.OLS(y_set, x_set).fit()

# Save output
output[0, d_set] = model.params[0]
output[1, d_set] = model.params[1]
# We could also print a summary, similar to R.
#     print(model.summary())

# Save output to DataFrame and print
output = pd.DataFrame(np.round(output, 3),
          columns=['model 1', 'model 2', 'model 3', 'model 4'],
          index=['intercept', 'slope'])
print(output)
```

```
   model 1  model 2  model 3  model 4
intercept       3.0    3.001    3.002    3.002
slope           0.5    0.500    0.500    0.500
```

The intercept β_0 and the slope β_1 are almost identical in all four models (see table).

	model 1	model 2	model 3	model 4
intercept ($\hat{\beta}_0$)	3.000	3.001	3.002	3.002
slope ($\hat{\beta}_1$)	0.500	0.500	0.500	0.500

Even the standard errors and $\hat{\sigma}$ are very similar (1. model: standard error of $\beta_0 = 1.125$, standard error of $\beta_1 = 0.117$ and residual standard error: 1.237).

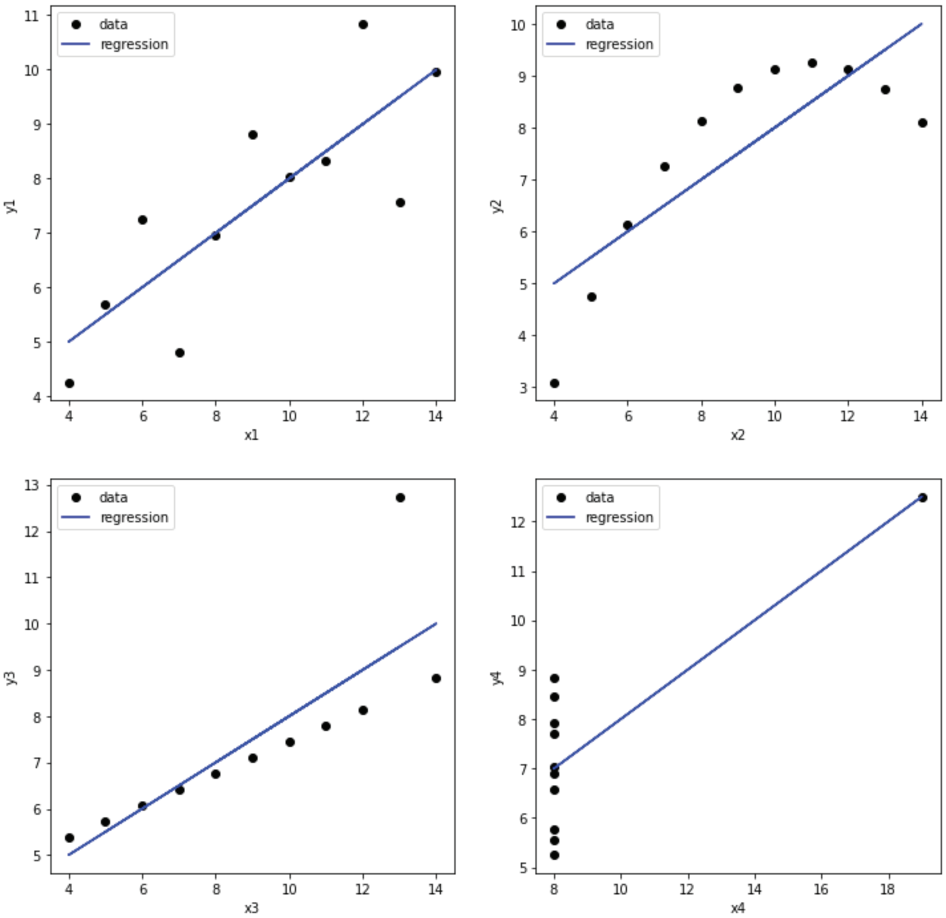
b) **Python** code:

[3]:
```
import matplotlib.pyplot as plt

# Create figure and subfigures:
fig = plt.figure(figsize=(12, 12))
# Create the four subplots:
for i in range(4):
    ax = fig.add_subplot(2, 2, i + 1)    # Create axes
    # find column labels, x-, y- and regression values
    x_lab = anscombe.columns.values[1 + i]
    y_lab = anscombe.columns.values[5 + i]
    x = anscombe.loc[:, x_lab]
    y = anscombe.loc[:, y_lab]
    y_reg = x * output.iloc[1, i] + output.iloc[0, i]
    # Plot Data points
    ax.plot(x, y, 'ok', label='data')
    # Plot linear regression line:
    ax.plot(x, y_reg, '-b', label='regression')
    # Set labels and title
    ax.set_xlabel(x_lab)
    ax.set_ylabel(y_lab)
    plt.legend()

plt.show()
```

Conclusion: It is not sufficient to simply consider $\hat{\beta}_0$, $\hat{\beta}_1$ and their standard errors. These estimates are almost identical in every model, although the data are completely different. A (graphical) check is essential. Due to the similarity between the values of those coefficients, it is obvious that the regression lines are similar as well.

Solution 1.2

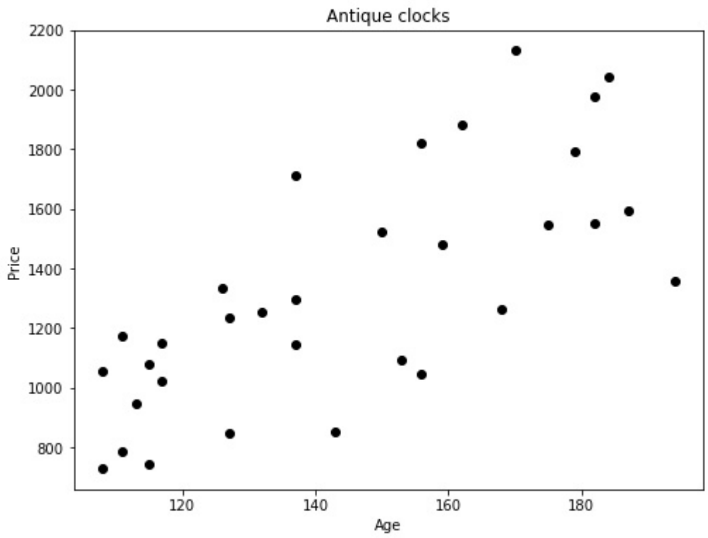
a) **Python** code:



[1]:
```
import pandas as pd
import numpy as np

# Read Data: make sure you have downloaded the datafile and placed it
# in a folder named data, in the same directory as this notebook
clocks = pd.read_csv('./data/antique_clocks.csv')

# As a first inspeaction, print the first rows of the data:
print(clocks.head()) #, '\n\n', clocks.describe())
# As well as the dimensions of the set:
print('\nSize of clocks =\n', clocks.shape)
```

```
   Unnamed: 0  age  price
0           1  108    729
1           2  108   1055
2           3  111    785
3           4  111   1175
4           5  113    946

Size of clocks =
 (32, 3)
```

[2]:
```
import matplotlib.pyplot as plt

# Create figure and subfigures:
fig = plt.figure(figsize=(8, 6))
# Create axes in subplots
ax = fig.add_subplot(1, 1, 1)
# Plot scatter data
ax.plot(clocks['age'], clocks['price'], 'ok') # Plot Data points
# Set labels:
ax.set_xlabel('Age')
ax.set_ylabel('Price')
ax.set_title('Antique clocks')
# show plot
plt.show()
```



We observe a relatively strong variation in the data. However a linear trend can be observed: the older the clock, the more expensive it is.

b) **Python** code:

[3]:
```
import statsmodels.api as sm

# Define x and y:
x = clocks['age']
y = clocks['price']
x_sm = sm.add_constant(x)

# Fit the linear model
model = sm.OLS(y, x_sm).fit()

# Print the estamited coefficients and further information
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.533
Model:                            OLS   Adj. R-squared:                  0.518
Method:                 Least Squares   F-statistic:                     34.27
Date:                Mon, 22 Feb 2021   Prob (F-statistic):           2.10e-06
Time:                        10:52:53   Log-Likelihood:                -223.88
No. Observations:                  32   AIC:                             451.8
Df Residuals:                      30   BIC:                             454.7
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       -191.6576    263.887     -0.726      0.473    -730.586     347.271
age           10.4791      1.790      5.854      0.000       6.823      14.135
==============================================================================
Omnibus:                        0.830   Durbin-Watson:                   2.691
Prob(Omnibus):                  0.660   Jarque-Bera (JB):                0.763
Skew:                           0.066   Prob(JB):                        0.683
Kurtosis:                       2.255   Cond. No.                         806.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
↪specified.
```

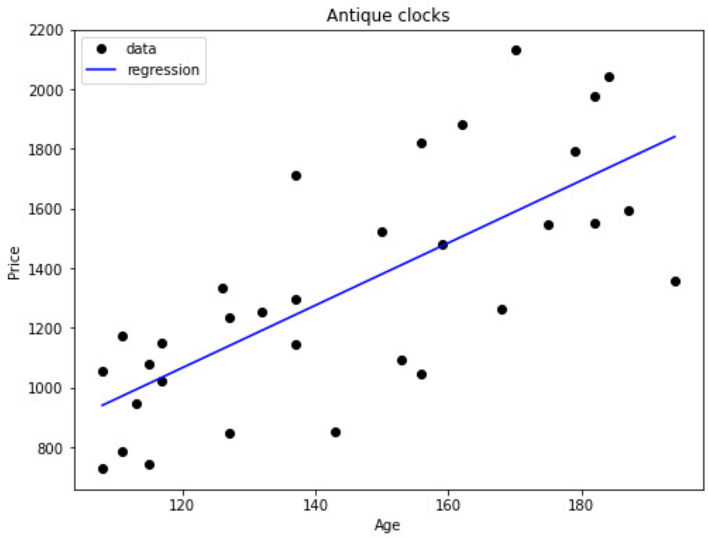
The estimated coefficients are $\hat{\beta}_0 = -191.66$ and $\hat{\beta}_1 = 10.479$. The residual standard Error is $\hat{\sigma} = 273.028$

c) **Python** code:

[4]:
```python
import matplotlib.pyplot as plt

y_reg = x * model.params[1] + model.params[0]

# Create figure and subfigures:
fig = plt.figure(figsize=(8, 6))
# Create axes in subplots
ax = fig.add_subplot(1, 1, 1)
# Plot scatter data
ax.plot(clocks['age'], clocks['price'], 'ok', label='data')
# Plot regression line
ax.plot(clocks['age'], y_reg, '-b', label='regression')
# Set labels:
ax.set_xlabel('Age')
ax.set_ylabel('Price')
ax.set_title('Antique clocks')
plt.legend()
# show plot
plt.show()
```

The regression line fits the data quite well.

Solution 1.3



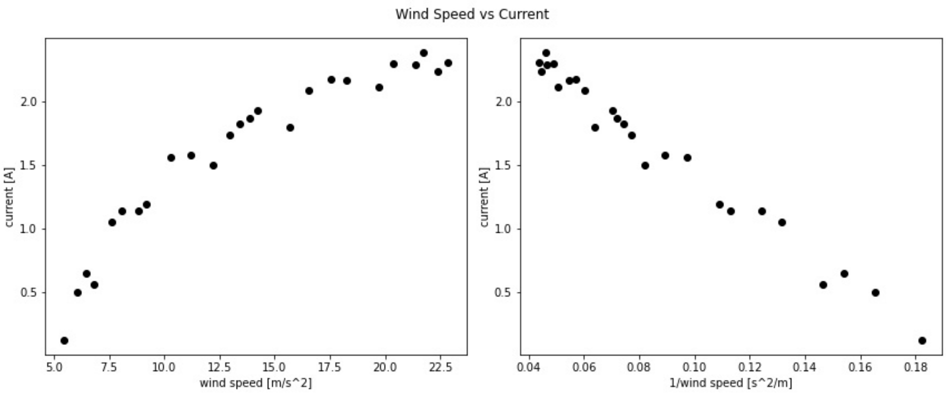
a) **Python** code:

[1]:
```
import pandas as pd
import numpy as np

# Read Data: make sure you have downloaded the datafile and placed it
# in a folder named data, in the same directory as this notebook
windmill = pd.read_csv('./data/windmill.csv')

# As a first inspeaction, print the first rows of the data:
print(windmill.head()) #, '\n\n', clocks.describe())
# As well as the dimensions of the set:
print('\nSize of windmill =\n', windmill.shape)
```

```
   wind_speed  current
0   11.187073    1.582
1   13.424487    1.822
2    7.607209    1.057
3    6.041019    0.500
4   22.374145    2.236

Size of windmill =
 (25, 2)
```

[2]:
```
import matplotlib.pyplot as plt

# Create figure and subfigures:
fig = plt.figure(figsize=(12, 5))

# Create axes in subplots
ax1 = fig.add_subplot(1, 2, 1)
ax2 = fig.add_subplot(1, 2, 2)
```

```
# Plot scatter data
ax1.plot(windmill['wind_speed'], windmill['current'], 'ok')
ax2.plot(1 / windmill['wind_speed'], windmill['current'], 'ok')
# Set labels:
ax1.set_xlabel('wind speed [m/s^2]')
ax2.set_xlabel('1/wind speed [s^2/m]')
ax1.set_ylabel('current [A]')
ax2.set_ylabel('current [A]')
fig.suptitle('Wind Speed vs Current')

# show plot
plt.tight_layout()
plt.show()
```



The regression line in the second scatter plot (*current* vs. $\frac{1}{\text{wind speed}}$) seems to describe data better as compared to the first one.

b) **Python** code:

```
[3]: import statsmodels.api as sm

# Define x and y:
x = 1 / windmill['wind_speed']
y = windmill['current']
x_sm = sm.add_constant(x)

# Fit the linear model
model = sm.OLS(y, x_sm).fit()

# Print the estamited coefficients and further information
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                current   R-squared:                       0.980
Model:                            OLS   Adj. R-squared:                  0.979
Method:                 Least Squares   F-statistic:                     1128.
```

```
Date:                    Mon, 22 Feb 2021   Prob (F-statistic):           4.74e-21
Time:                            17:01:00   Log-Likelihood:                 24.635
No. Observations:                      25   AIC:                            -45.27
Df Residuals:                          23   BIC:                            -42.83
Df Model:                               1
Covariance Type:                nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          2.9789      0.045     66.341      0.000       2.886       3.072
wind_speed   -15.5155      0.462    -33.592      0.000     -16.471     -14.560
==============================================================================
Omnibus:                        2.768   Durbin-Watson:                   1.567
Prob(Omnibus):                  0.251   Jarque-Bera (JB):                2.287
Skew:                          -0.720   Prob(JB):                        0.319
Kurtosis:                       2.646   Cond. No.                         24.7
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
↪specified.
```

The coefficients are $\hat{\beta}_0 = 2.979$ and $\hat{\beta}_1 = -15.515$. The standard errors are $\text{se}(\hat{\beta}_0) = 0.0449$ and $\text{se}(\hat{\beta}_1) = 0.462$.

c) **Python** code:

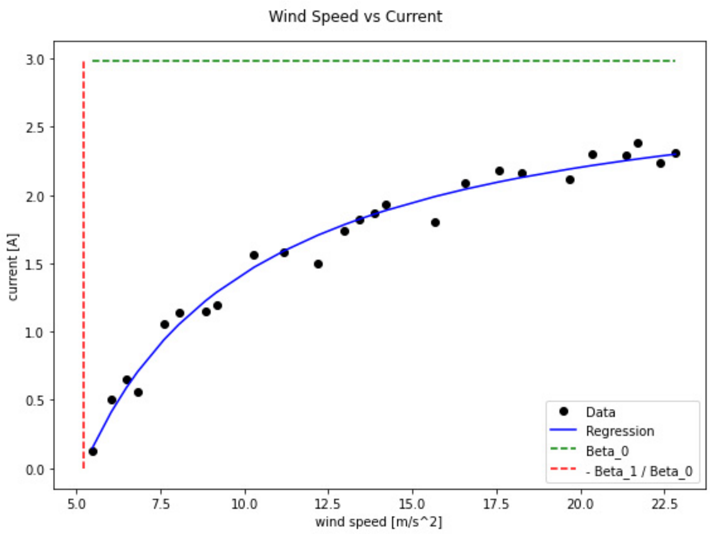
[4]:
```python
# Confidence interval found using conf_int method
confint = model.conf_int(alpha=0.01)
print(np.round(confint, 3))
```

```
                 0       1
const        2.853   3.105
wind_speed -16.812 -14.219
```

d) **Python** code:

[5]:
```python
# Define x and y for the Regression line
x = np.sort(windmill['wind_speed'])
y_reg = model.params[0] + model.params[1] * 1 / x

# Define Beta's
B_0, B_1 = model.params[0], model.params[1]

# Create figure and subfigures:
fig = plt.figure(figsize=(8, 6))

# Create axes in subplots
ax = fig.add_subplot(1, 1, 1)
# Plot scatter data
ax.plot(windmill['wind_speed'], windmill['current'],
        'ok', label='Data')
# Plot Regression line
ax.plot(x, y_reg, '-b', label='Regression')
```

```
# Aditional lines:
ax.plot([x.min(), x.max()], [B_0, B_0],
         '--g', label='Beta_0')
ax.plot([- B_1 / B_0, - B_1 / B_0], [0, B_0],
         '--r', label='- Beta_1 / Beta_0')
# Set labels:
ax.set_xlabel('wind speed [m/s^2]')
ax.set_ylabel('current [A]')
fig.suptitle('Wind Speed vs Current')
plt.legend()

# show plot
plt.show()
```



The model is

$$\text{current} \approx \beta_0 + \beta_1 \frac{1}{\text{wind speed}}$$

As the wind speed approaches infinity, β_0 becomes the maximally accessible current production (horizontal green line).

The coefficient β_1 is harder to interpret. It refers to the wind speed, at which the windmill starts to produce an electrical current at all:

$$0 = \beta_0 + \beta_1 \frac{1}{\text{windspeed}_0}$$

$$\text{windspeed}_0 = -\frac{\beta_1}{\beta_0}$$

This means, the larger the absolute value of β_1 the larger the wind speed has to be, in order to have a windmill producing power.

e) **Python** code:

```
[6]: x0 = [1/1, 1/10]
     x0 = sm.add_constant(x0)

     # Prediction
     pred0 = model.get_prediction(x0)
     pred0 = pred0.summary_frame(alpha=0.05)

     print('Expected values at 1 and 10 m/s:\n', pred0['mean'],
           '\n\nConfidence interval at 1 and 10 m/s:\n',
           pred0[['mean_ci_lower', 'mean_ci_upper']],
           '\n\nPrediction interval at 1 and 10 m/s:\n',
           pred0[['obs_ci_lower', 'obs_ci_upper']])
```

```
Expected values at 1 and 10 m/s:
0   -12.536597
1     1.427314
Name: mean, dtype: float64

Confidence interval at 1 and 10 m/s:
mean_ci_lower  mean_ci_upper
0     -13.408613     -11.664581
1       1.386768       1.467861

Prediction interval at 1 and 10 m/s:
obs_ci_lower  obs_ci_upper
0    -13.430108    -11.643086
1      1.228331      1.626298
```

For the speed of $10\frac{\text{m}}{\text{s}}$ we obtain a value of 1.43 A. As expected, the prediction intervals are (slightly) larger than the confidence intervals. The results for a wind speed of one meter per second do not make sense, because the windmill does not yet rotate (see exercise before). This problem arises because of the extrapolation of the model, which in this case is obviously non-sense.

Solution 1.4

a) **Python** code:

```
[1]: import pandas as pd
     import numpy as np

     # Read Data: make sure you have downloaded the datafile and placed it
     # in a folder named data, in the same directory as this notebook
     forbes = pd.read_csv('./data/Forbes.csv')

     # As a first inspeaction, print the first rows of the data:
```

```
print(forbes.head()) #, '\n\n', clocks.describe())
# As well as the dimensions of the set:
print('\nSize of forbes =\n', forbes.shape)
```

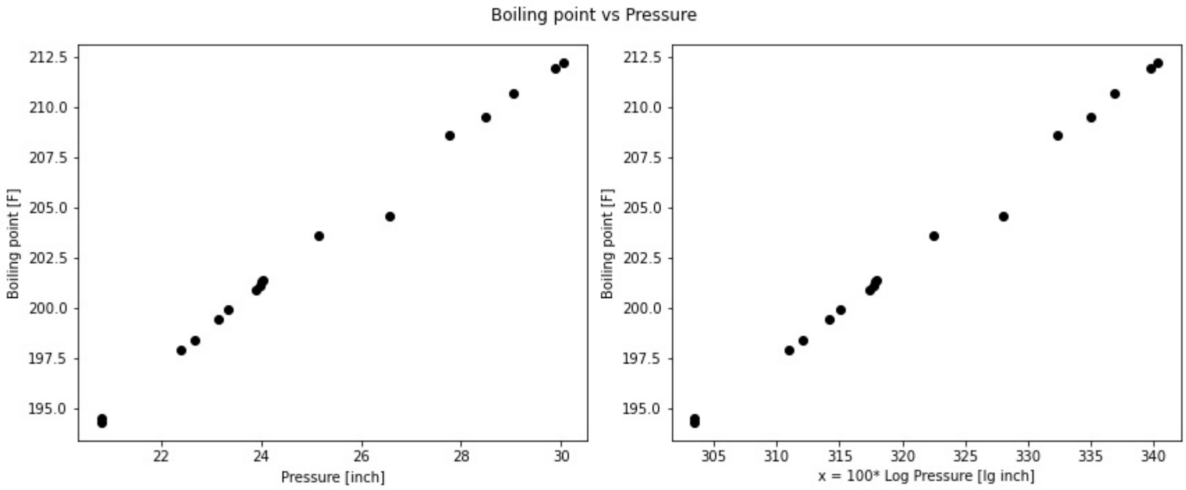
```
       y  pressure
0  194.5     20.79
1  194.3     20.79
2  197.9     22.40
3  198.4     22.67
4  199.4     23.15

Size of forbes =
 (17, 2)
```

[2]:
```
import matplotlib.pyplot as plt

# Define x and y:
x = 100 * np.log(forbes['pressure'])
y = forbes['y']

# Create figure and subfigures:
fig = plt.figure(figsize=(12, 5))

# Create axes in subplots
ax1 = fig.add_subplot(1, 2, 1)
ax2 = fig.add_subplot(1, 2, 2)
# Plot scatter data
ax1.plot(forbes['pressure'], y, 'ok')
ax2.plot(x, y, 'ok')
# Set labels:
ax1.set_xlabel('Pressure [inch]')
ax2.set_xlabel('x = 100* Log Pressure [lg inch]')
ax1.set_ylabel('Boiling point [F]')
ax2.set_ylabel('Boiling point [F]')
fig.suptitle('Boiling point vs Pressure')

# show plot
plt.tight_layout()
plt.show()
```

If we have a thorough look at the plot, we observe that data points in the first scatter plot lie on a slightly curved line. In the second scatter plot we observe that data points scatter almost perfectly around a straight line.

b) **Python** code:

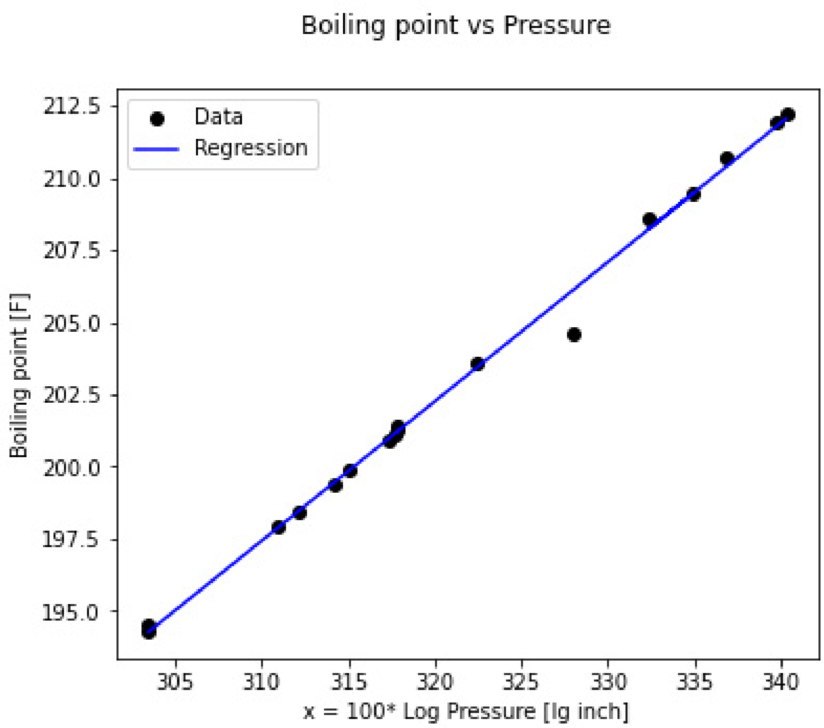
[3]:
```
import statsmodels.api as sm

# Define x for linear model
```




```python
x_sm = sm.add_constant(x)

# Fit the linear model
model = sm.OLS(y, x_sm).fit()

# Possibly print a summary:
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.995
Model:                            OLS   Adj. R-squared:                  0.995
Method:                 Least Squares   F-statistic:                     2962.
Date:                Tue, 23 Feb 2021   Prob (F-statistic):           1.19e-18
Time:                        16:38:15   Log-Likelihood:                -8.4026
No. Observations:                  17   AIC:                             20.81
Df Residuals:                      15   BIC:                             22.47
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         47.8638      2.852     16.784      0.000      41.786      53.942
pressure       0.4825      0.009     54.420      0.000       0.464       0.501
==============================================================================
Omnibus:                       37.131   Durbin-Watson:                   2.031
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               86.947
Skew:                          -3.091   Prob(JB):                     1.32e-19
Kurtosis:                      12.195   Cond. No.                     8.96e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 8.96e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

[4]:
```
# Define the regression line using regression constants
y_reg = model.params[0] + model.params[1] * x

# Create figure and subfigures:
fig = plt.figure(figsize=(6, 5))

# Create axes in subplots
ax = fig.add_subplot(1, 1, 1)
# Plot scatter data
ax.plot(x, y, 'ok', label='Data')
# Plot regression line
ax.plot(x, y_reg, '-b', label='Regression')
# Set labels:
ax.set_xlabel('x = 100* Log Pressure [lg inch]')
ax.set_ylabel('Boiling point [F]')
fig.suptitle('Boiling point vs Pressure')
plt.legend()

# show plot
plt.show()
```



The above regression line fits data rather well, although an outlier is clearly visible - we can identify this point by means of the **Python**-function **`OLSInfluence()`** from **`statsmodels.stats.outliers_influence`**: it is the 12th observation.

[5]:
```
from statsmodels.stats.outliers_influence import OLSInfluence

# Find different model influences of the fitted model:
model_inf = OLSInfluence(model)

# Print a summary:
print(model_inf.summary_table())
```

```
==================================================================================================
       obs      endog     fitted     Cook's   student.   hat diag    dffits   ext.stud.     dffits
                           value          d   residual              internal   residual
--------------------------------------------------------------------------------------------------
         0    194.500    194.267      0.048      0.617      0.202     0.310      0.604      0.304
         1    194.300    194.267      0.001      0.087      0.202     0.044      0.084      0.042
         2    197.900    197.866      0.000      0.086      0.108     0.030      0.083      0.029
         3    198.400    198.444      0.001     -0.109      0.097    -0.036     -0.106     -0.035
         4    199.400    199.455      0.001     -0.135      0.082    -0.040     -0.131     -0.039
         5    199.900    199.870      0.000      0.075      0.077     0.022      0.072      0.021
         6    200.900    200.973      0.001     -0.178      0.066    -0.048     -0.172     -0.046
         7    201.100    201.174      0.001     -0.182      0.065    -0.048     -0.176     -0.046
         8    201.400    201.235      0.006      0.405      0.064     0.106      0.393      0.103
         9    201.300    201.215      0.002      0.209      0.065     0.055      0.203      0.053
        10    203.600    203.433      0.005      0.407      0.059     0.102      0.395      0.099
        11    204.600    206.102      0.577     -3.705      0.078    -1.075    -12.275     -3.560
        12    209.500    209.469      0.001      0.080      0.139     0.032      0.077      0.031
        13    208.600    208.216      0.058      0.964      0.111     0.341      0.961      0.340
        14    210.700    210.391      0.063      0.800      0.164     0.354      0.789      0.349
        15    211.900    211.767      0.016      0.354      0.206     0.180      0.343      0.175
        16    212.200    212.057      0.020      0.383      0.216     0.201      0.372      0.195
==================================================================================================
```

c) **Python** code:

[6]:
```
# Delete the 12th observation:
x, x_sm, y = x.drop(11), x_sm.drop(11), y.drop(11)

# Fit the linear model
model = sm.OLS(y, x_sm).fit()

# Possibly print a summary:
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       1.000
Model:                            OLS   Adj. R-squared:                  1.000
Method:                 Least Squares   F-statistic:                 3.249e+04
Date:                Tue, 23 Feb 2021   Prob (F-statistic):           5.77e-25
Time:                        16:38:16   Log-Likelihood:                 11.326
No. Observations:                  16   AIC:                            -18.65
Df Residuals:                      14   BIC:                            -17.11
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         46.4530      0.868     53.498      0.000      44.591      48.315
pressure       0.4872      0.003    180.237      0.000       0.481       0.493
==============================================================================
Omnibus:                        1.506   Durbin-Watson:                   1.542
Prob(Omnibus):                  0.471   Jarque-Bera (JB):                1.230
Skew:                           0.597   Prob(JB):                        0.541
Kurtosis:                       2.352   Cond. No.                     8.76e+03
==============================================================================
```

```
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
↪specified.
[2] The condition number is large, 8.76e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

The **residual standard error** and the **standard errors** are reduced by a factor of 3.

In the following exercises we keep the 12th observation omitted.

d) Because the p-value of β_1 is smaller than 0.05 (=significance level), the null-hypothesis $\beta_1 = 0$ has to be rejected; i.e. β_1 is significantly different from 0 at the 5 % level.

e) **Python** code:

```
[7]: # Confidence interval found using conf_int method
confint = model.conf_int(alpha=0.05)
print(np.round(confint, 3))
```

```
                0       1
const      44.591  48.315
pressure    0.481   0.493
```

A 95 %-confidence interval for the slope β_1 is given by $[0.481, 0.493]$.

f) **Python** code:

```
[8]: x0 = [[1, 325.81]]

# Prediction
pred0 = model.get_prediction(x0)
pred0_95 = pred0.summary_frame(alpha=0.05)
pred0_99 = pred0.summary_frame(alpha=0.01)

print('Expected values at 26 Inch:\n', pred0_95['mean'],
      '\n\n95% Confidence interval:\n',
      pred0_95[['mean_ci_lower', 'mean_ci_upper']],
      '\n\n99% Confidence interval:\n',
      pred0_99[['mean_ci_lower', 'mean_ci_upper']])
```

```
Expected values at 26 Inch:
 0    205.172621
Name: mean, dtype: float64

95% Confidence interval:
    mean_ci_lower  mean_ci_upper
0     205.098906     205.246337
```

```
99% Confidence interval:
    mean_ci_lower  mean_ci_upper
0     205.070308     205.274934
```

[ ]:
```
    # Alternative,
    x0 = [325.81, 0]  # Add a random second value, f.e. 0
    x0 = sm.add_constant(x0)   # Use the known procedure.
```

The expected value is 205.17. The confidence intervals are [205.099, 205.246] and [205.070, 205.275]. As expected, the 99 % confidence interval is larger than the 95 % interval.

g) **Python** code:

[10]:
```
# As before:
x0 = [[1, 325.81]]

# Prediction
pred0 = model.get_prediction(x0)
pred0_99 = pred0.summary_frame(alpha=0.01)

print('Expected values at 26 Inch:\n', pred0_95['mean'],
      '\n\n99% Prediction interval:\n',
      pred0_99[['obs_ci_lower', 'obs_ci_upper']])
```

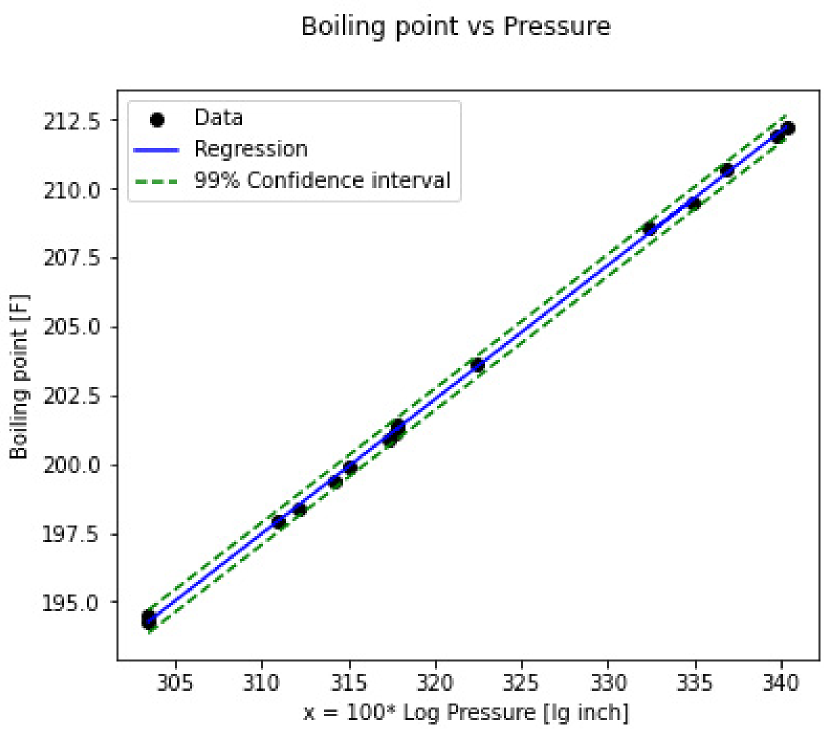
```
Expected values at 26 Inch:
 0    205.172621
Name: mean, dtype: float64

99% Prediction interval:
    obs_ci_lower  obs_ci_upper
0    204.779671    205.565572
```

A 99 %-prediction interval is [204.780, 205.566]. As expected, this interval is larger than the corresponding 99 %-confidence interval.

Voluntary Exercise: **Python** code:

[9]:
```
# Define some points at which to evaluate the prediction
x0 = np.linspace(x.min(), x.max(), 10)
x0 = sm.add_constant(x0)  # Use te known procedure.

# Prediction
pred0 = model.get_prediction(x0)
pred0 = pred0.summary_frame(alpha=0.01)

# Define the regression line using regression constants
y_reg = model.params[0] + model.params[1] * x
```

```
# Create figure and subfigures:
fig = plt.figure(figsize=(6, 5))

# Create axes in subplots
ax = fig.add_subplot(1, 1, 1)
# Plot scatter data
ax.plot(x, y, 'ok', label='Data')
# Plot regression line
ax.plot(x, y_reg, '-b', label='Regression')
# Plot 99% intervals
ax.plot(x0[:,1], pred0['obs_ci_lower'], '--g',
        label='99% Prediction interval')
ax.plot(x0[:,1], pred0['obs_ci_upper'], '--g')

# Set labels:
ax.set_xlabel('x = 100* Log Pressure [lg inch]')
ax.set_ylabel('Boiling point [F]')
fig.suptitle('Boiling point vs Pressure')
plt.legend()

# show plot
plt.show()
```



Solution 1.5

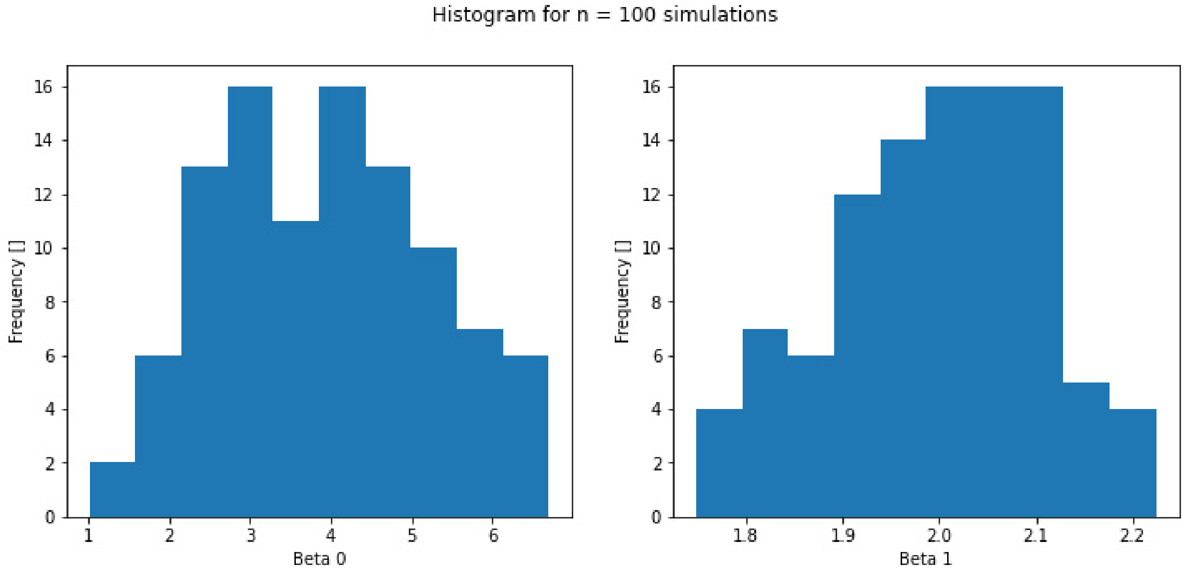
a) **Python** code:

[1]:
```python
import numpy as np
import statsmodels.api as sm
from scipy.stats import norm

# Set random seed
np.random.seed(0)
# Set number of random simulations
n = 100
# xi as given
x_i = np.array([0, 3, 4, 8, 10, 11, 13, 16, 17, 20])
x_i_sm = sm.add_constant(x_i)

# random error, taken from normal distribution
e_i = norm.rvs(loc=0, scale=np.sqrt(2), size=10*n)
e_i = e_i.reshape((10, n))

# predifine Y_i, and the regression coefficients
Y_i = np.zeros((n, 10))
b_0, b_1 = np.zeros((n)), np.zeros((n))
for i in range(n):
    # Find Y_i
    Y_i[i] = 4 + 2 * x_i + e_i[:, i]
    # Perform linear regression
    model = sm.OLS(Y_i[i], x_i_sm).fit()
    # Save Regression coefficients
    b_0[i] = model.params[0]
    b_1[i] = model.params[1]

print('Regression Coefficient Beta_0:\n', np.round(b_0, 4),
     '\n\nRegression Coefficient Beta_1:\n', np.round(b_1, 4))
```

b) `Python` code:

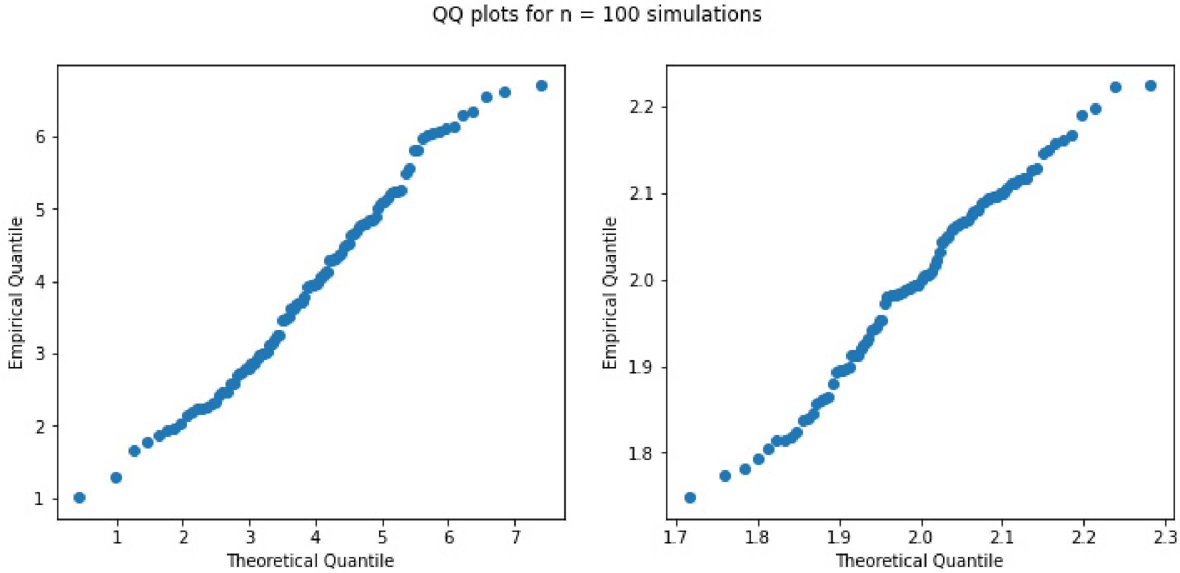
[2]:
```python
''' Histogram '''
import matplotlib.pyplot as plt

# Create figure and subfigures:
fig = plt.figure(figsize=(12, 5))

# Create axes in subplots
ax1 = fig.add_subplot(1, 2, 1)
ax2 = fig.add_subplot(1, 2, 2)
# histogram plots
ax1.hist(b_0)
ax2.hist(b_1)
# Set labels:
ax1.set_xlabel('Beta 0')
ax2.set_xlabel('Beta 1')
ax1.set_ylabel('Frequency []')
ax2.set_ylabel('Frequency []')
```

```
title = 'Histogram for n = ' + str(n) + ' simulations'
fig.suptitle(title)

# show plot
plt.show()
```



[3]:
```
''' Normal Plot '''
# Quantiles
alphak = (np.arange(1, b_0.size + 1) - 0.5) / b_0.size
q_theor_b0 = norm.ppf(q=alphak, loc=b_0.mean(), scale=b_0.std())
q_theor_b1 = norm.ppf(q=alphak, loc=b_1.mean(), scale=b_1.std())
q_empir_b0 = np.sort(b_0)
q_empir_b1 = np.sort(b_1)

# Create figure and subfigures:
fig = plt.figure(figsize=(12, 5))

# Create axes in subplots
ax1 = fig.add_subplot(1, 2, 1)
ax2 = fig.add_subplot(1, 2, 2)
# Plot figure
ax1.plot(q_theor_b0, q_empir_b0, "o")
ax2.plot(q_theor_b1, q_empir_b1, "o")
# Labels
ax1.set_xlabel("Theoretical Quantile")
ax1.set_ylabel("Empirical Quantile")
ax2.set_xlabel("Theoretical Quantile")
ax2.set_ylabel("Empirical Quantile")
title = 'QQ plots for n = ' + str(n) + ' simulations'
fig.suptitle(title)
```

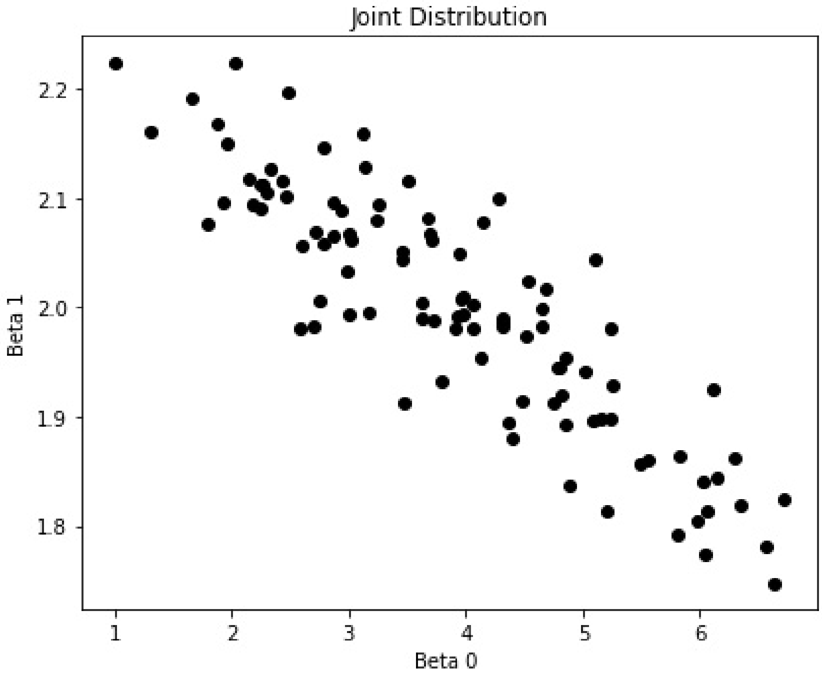
```
plt.show()
```



[4]:
```
''' Scatter Plot '''
# Create figure and subfigures:
fig = plt.figure(figsize=(6, 5))

# Create axes in subplots
ax = fig.add_subplot(1, 1, 1)
# Plot scatter data
ax.plot(b_0, b_1, 'ok')
# Set labels:
ax.set_xlabel('Beta 0 ')
ax.set_ylabel('Beta 1')
plt.title('Joint Distribution')

# show plot
plt.tight_layout()
plt.show()
```

c) The following results depend on the concrete simulation, unless you fix the randomized values with **np.random.seed()** : **Python** code:

[5]:
```
# Means:
b0_mean = np.round(b_0.mean(), 4)
b1_mean = np.round(b_1.mean(), 4)
# Standard deviation
b0_std = np.round(b_0.std() / np.sqrt(2), 4)
b1_std = np.round(b_1.std() / np.sqrt(2), 4)
# Variances:
b0_var = np.round(b_0.var(), 4)
b1_var = np.round(b_1.var(), 4)
```




```
print('Means:\n', b0_mean, b1_mean,
      '\nStandard deviations:\n', b0_std, b1_std,
      '\nVariances:\n', b0_var, b1_var )
```

```
Means:
 3.9223 1.9987
Standard deviations:
 0.9553 0.0777
Variances:
 1.8251 0.0121
```

According to theory the estimates should scatter around $\beta_0 = 4$ and $\beta_1 = 2$ (due to the standard deviation of the error term $\sigma = \sqrt{2}$). For the (estimated) variance of $\hat{\beta}_0$ and $\hat{\beta}_1$ the following formulae need to be calculated: **Python** code:

[6]:
```
# Se Beta 0:
SSx = np.sum((x_i - x_i.mean()) ** 2)
b0_se = np.sqrt(2 * (1 / 10 + x_i.mean() **2 / SSx ))

# Se Beta 1:
b1_se = np.sqrt(2 / SSx)

print('Se Beta 0:  ', np.round(b0_se, 4),
      '\nSe Beta 1:  ', np.round(b1_se, 4))
```

```
Se Beta 0:   0.8616
Se Beta 1:   0.0722
```

The theoretical standard errors $\mathrm{se}(\hat{\beta}_0) = 0.862$ and $\mathrm{se}(\hat{\beta}_1) = 0.0722$ that we computed above correspond to the average deviation of the parameter estimates $\hat{\beta}_0$ and $\hat{\beta}_1$ around the true parameter values β_0 and β_1.

The more simulations we run, the closer the emprirical standard deviations of the set of estimates $\hat{\beta}_0$ and $\hat{\beta}_1$ get to the *true* standard deviations $\mathrm{se}(\hat{\beta}_0)$ and $\mathrm{se}(\hat{\beta}_1)$. On the basis of 10 000 simulations we find $\mathrm{se}(\hat{\beta}_0) \approx 0.8616496$ and $\mathrm{se}(\hat{\beta}_1) \approx 0.0721$. We conclude that the theoretical standard errors $\mathrm{se}(\hat{\beta}_0)$ and $\mathrm{se}(\hat{\beta}_1)$ and the (approximately) true values of $\mathrm{se}(\hat{\beta}_0)$ and $\mathrm{se}(\hat{\beta}_1)$ agree to a high degree.