Predictive Modeling

Series 2

Exercise 2.1

The following exercise is based on the `windmill` data, found in the data file `windmill.csv`. We consider the following three regression models:

a) Naive:

$$\text{current} = \beta_0 + \beta_1(\text{wind speed}) + \epsilon$$

b) First-Aid Transformation:

$$\log(\text{current}) = \beta_0 + \beta_1 \log(\text{wind speed}) + \epsilon$$

c) Transformation according to expert knowledge:

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

Check by means of residual plots, for which of the three models the assumptions of the linear regression model are fulfilled.

Exercise 2.2

We consider the model

$$y_i = \beta_0 + \beta_1 \cdot x_i + \epsilon_i$$

where

$$x = 100 \cdot \log(\mathit{pressure})$$

to fit the `Forbes` data. Check with the help of residual plots, whether the assumptions of the linear regression model are fulfilled.

Exercise 2.3

While fitting and visualizing simple linear regression models becomes a routine after a while, assessing whether a model fits the data remains a challenging task. We will practice this aspect of linear regression analysis with two additional data sets:

a) The file `gas.csv` contains the gas `consumption` (in kWh) and the difference of `temperature` (in Celcius) inside and outside of 15 houses which are heated with gas. The measurements were collected over a long time period and then averaged. The goal is to predict the gas consumption on the basis of the temperature difference. Plot the regression line and perform a residual analysis.

b) The file `antique_clocks.csv` contains the `age` and the `price` of antique clocks that are auctioned. The goal is to predict the price on the basis of the age of the clock. Plot the regression line and perform a residual analysis.

Exercise 2.4

The article *Characterization of Highway Runoff in Austin, Texas, Area* was based on a data set with $x =$ `rainfall volume` and $y =$ `runoff volume` for a particular location. The values are:

`rainfall volume`	5	12	14	17	23	30	40	47	55	67	72	81	96	112	127
`runoff volume`	4	10	13	15	15	25	27	46	38	46	53	70	82	99	100

a) Generate a scatter plot of `runoff volume` versus `rainfall volume`. Fit a simple linear regression model, add the regression line to the plot and generate the `Python` summary output for the regression model.

b) How much of the observed variation in `runoff volume` can be explained by the simple linear regression model with response variable `runoff volume` and predictor variable `rainfall volume`?

c) Is there a significant linear association between `runoff volume` and `rainfall volume`? Provide an illustrative interpretation of the regression coefficients.

d) Use the regression fit to predict the `runoff volume` when the `rainfall volume` takes on the value 50. Also compute the 95 % prediction interval for a `rainfall volume` value of 50.

e) Assess the model assumptions by means of the model diagnostics tools we have discussed.

f) Fit a new regression model on the basis of the log-transformed variables and add it to the scatter plot.

g) Assess the strength and the significance of the linear relationship between the log-transformed variables `runoff volume` and `rainfall volume`. Interpret the regression coefficients.

h) Predict the expected `runoff volume` when rainfall takes on a value of 50. Generate a 95 % prediction interval and compare it to the original solution. Add the prediction interval for arbitrary x to the scatter plot.

i) Perform a residual analysis. Which model is more appropriate and why?

Exercise 2.5

In an experiment marine bacteria were exposed to x-rays during 15 intervals of six minutes. The following table contains the number of `surviving bacteria` after each `interval`:

`interval`	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
`surv. bact.`	255	211	197	166	NA	106	104	60	56	38	36	32	21	19	15

a) Show the relation between the number of `surviving bacteria` and the number of radiation `intervals`. Does it make sense to fit a least squares regression model to the data?

b) Fit a simple linear regression model and check the model assumptions.

c) Improve the model by transforming the response variable or/and the predictor. **Hint**: Theory suggests that per radiation interval the proportion of bacteria that is killed remains constant.

d) Predict the missing value for the fifth interval and compute a 95 % prediction interval. In addition, compute the estimate for the relative decrease in the number of surviving bacteria, together with a 95 % confidence interval. Last, determine the expected number of bacteria at the beginning, i.e. before the first radiation interval. Also compute a 95 % confidence interval for this value.

Exercise 2.6

Assessing model diagnostic plots requires experience. Often it is difficult to decide whether a deviation is systematic (i.e. needing correction) or due to random variations (i.e. just variability in the data). Experience can be gained by performing model diagnostics on problems where it is known whether the model assumptions hold or do not hold. This allows us to identify the naturally occurring variability in the results.

In the following we simulate a predictor variable x and four responses y_a, y_b, y_c, and y_d.

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

n = 100
random.seed(0)

x = np.linspace(0, n -1 , n)
x = pd.Series(x, name='x')
y_a = 2 + 1 * x + norm.rvs(size=n)
y_b = 2 + 1 * x + norm.rvs(size=n) * x
y_c = 2 + 1 * x + norm.rvs(size=n) * (1 + x / n)
y_d = np.cos(x * np.pi / (n/2) ) + norm.rvs(size=n)

# Prepare x for Statsmodels.api
x_sm = sm.add_constant(x)
```

Fit four simple linear regression models using xx as predictor variable.

a) For each model, generate a scatter plot with the regression line, generate the four standard residual plots and the plot containing Cook's distance. Decide for each model which of the assumptions are fulfilled and which ones are violated. Verify your claims with the construction of the responses.

Result Checker

E 2.4: b) $R^2 = 0.98$

c) $\hat{\beta}_0 = -1.12$ and $\hat{\beta}_1 = 0.827$

d) $[28.53, 51.92]$

h) 39.88 and $[29.86744, 53.25903]$

E 2.5: d) Prediction and Prediction interval for interval 5 : 124.0672 and $[96.29711, 159.8]$
Prediction and Prediction interval for interval 0 : 352.2568 $[307.3775, 403.6888]$

Predictive Modeling

Solutions to Series 2

Solution 2.1

[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 we will have to perform the same task multiple times, we will write a definition for this task.

[2]:
```
""" Plot Residuals vs Fitted Values """
def plot_residuals(axes, res, yfit, n_samp=0):
    """ Inputs:
    axes: axes created with matplotlib.pyplot
    x: x values
    ytrue: y values
    yfit: fitted/predicted y values
    res[optional]: Residuals, used for resampling
    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

    dataframe = pd.concat([yfit, res], axis=1)
    axes = sns.residplot(x=yfit, y=res, data=dataframe,
                    lowess=True, scatter_kws={'alpha': 0.5},
                    line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8})
    axes.set_title('Residuals vs Fitted')
    axes.set_ylabel('Residuals')
    axes.set_xlabel('Fitted Values')



""" QQ Plot standardized residuals """
def plot_QQ(axes, res_standard, n_samp=0):
    """ 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)
        # 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), lowess=True, ci=False,
                        scatter=False,
                        line_kws={'color': 'lightgrey', 'lw': 1, 'alpha': 0.8})

    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})
    axes.plot(qqx, qqx, '--k', alpha=0.5)
    axes.set_title('Normal Q-Q')
    axes.set_xlabel('Theoretical Quantiles')
    axes.set_ylabel('Standardized Residuals')


""" Scale-Location Plot """
def plot_scale_loc(axes, yfit, res_stand_sqrt, n_samp=0):
    """ 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('Scale-Location plot')
    axes.set_xlabel('Fitted Sales values')
    axes.set_ylabel('$\sqrt{\|Standardized\ Residuals\|}$')
```

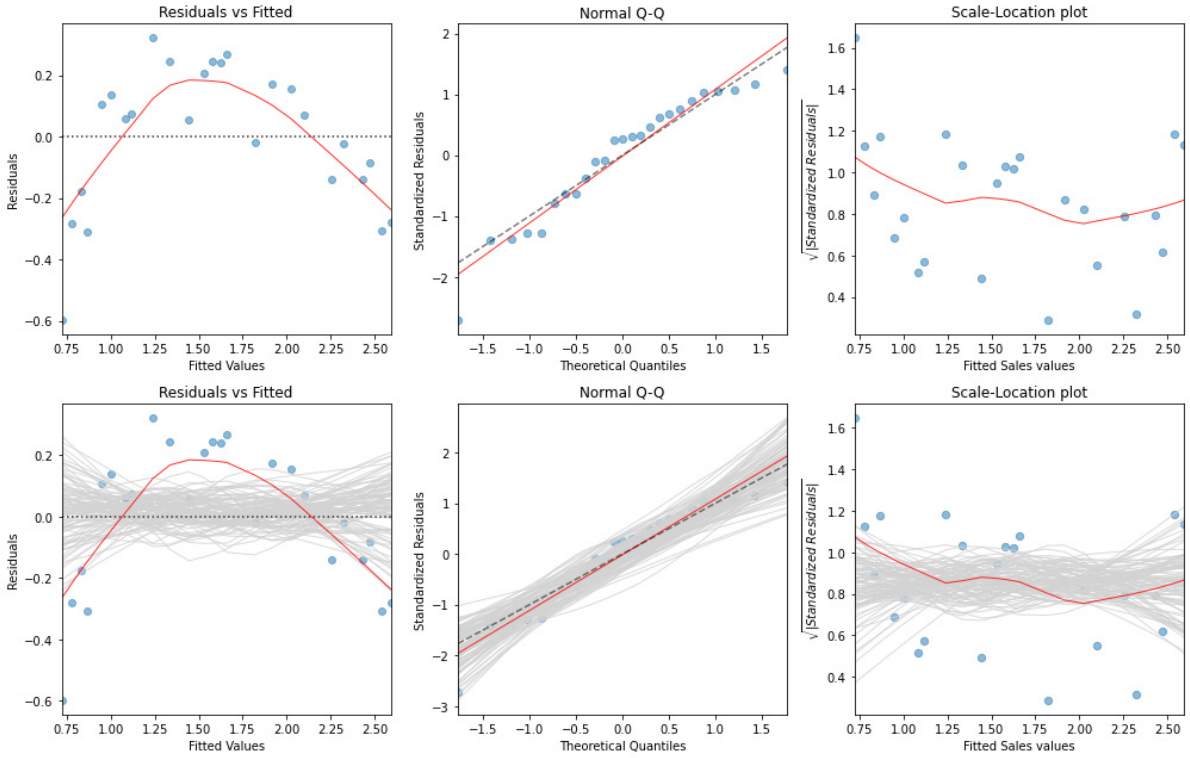
a) **Python** code:

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

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

# Fit the linear model
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))
```

[4]:
```
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.graphics.gofplots import ProbPlot
import random

""" Plots """
# Create Figure and subplots
fig = plt.figure(figsize = (14,9))

# First subplot: Residuals vs Fitted values
ax1 = fig.add_subplot(2, 3, 1)
plot_residuals(ax1, res, yfit)

# Second subplot: QQ Plot
ax2 = fig.add_subplot(2, 3, 2)
plot_QQ(ax2, res_standard)

# Third subplot: Scale-location
ax3 = fig.add_subplot(2, 3, 3)
plot_scale_loc(ax3, yfit, res_stand_sqrt)

# Fourth subplot: Residuals vs Fitted values with 100 resamples
ax4 = fig.add_subplot(2, 3, 4)
plot_residuals(ax4, res, yfit, n_samp = 100)

# Fifth subplot: QQ Plot with 100 resamples
ax5 = fig.add_subplot(2, 3, 5)
plot_QQ(ax5, res_standard, n_samp = 100)

# Sixth subplot: Scale-location with 100 resamples
ax6 = fig.add_subplot(2, 3, 6)
plot_scale_loc(ax6, yfit, res_stand_sqrt, n_samp = 100)

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

- *Tukey-Anscombe*: The banana-like shape of the smoothing line suggests that the expected value is not constant zero. This structure is as well visible in



the graphics containing the simulated smoothing lines: the (red) smoothing line is not entirely contained in the grey band of simulated curves.

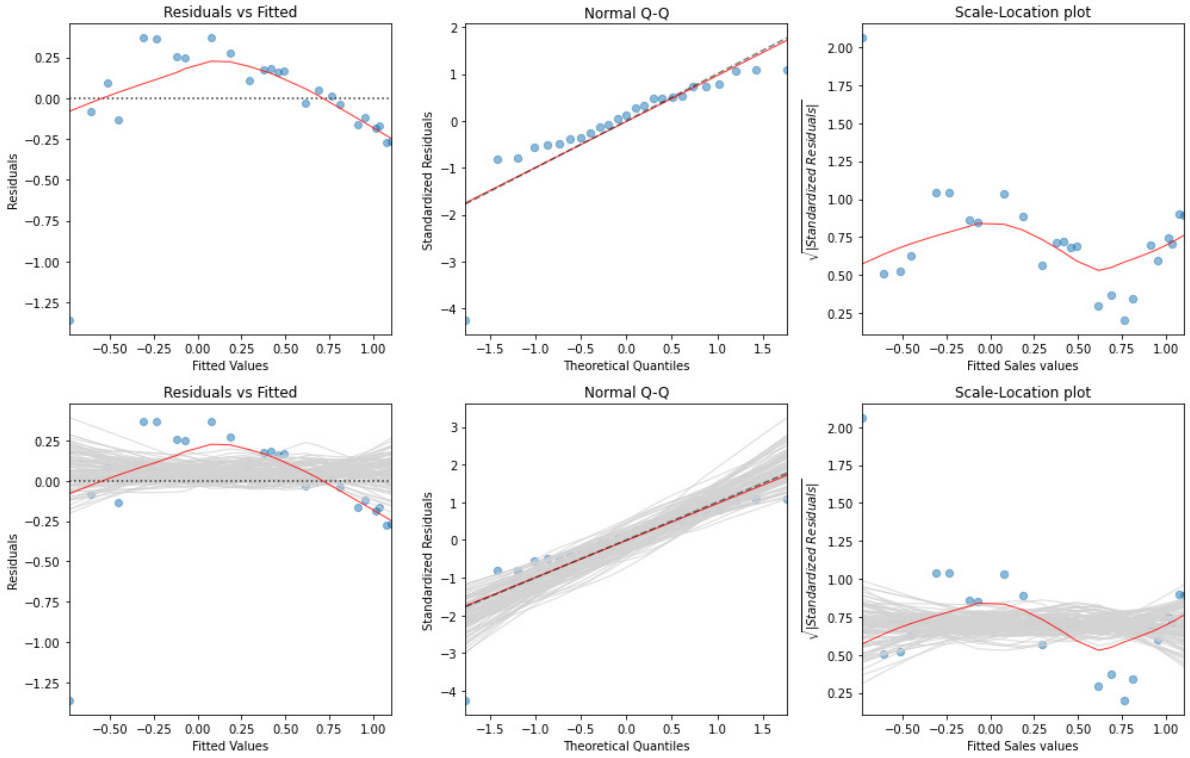
- *Normal Plot*: The data points scatter next to the straight line. All data points fall into the band of curves which may arise due to statistical fluctuations. There is no evidence to doubt the assumption that the residuals follow a normal distribution.
- *Scale-Location*: The smoothing line tends to decrease, but it is still part of the grey band of simulated curves. Thus, there is no evidence against the assumption that the variance is constant.
- *Time Correlation:* Since we do not dispose of any information concerning the time order of the measurements, we cannot analyze any time correlations. Such a correlation would represent a (clear) violation of the independence of the residuals.

Conclusion: The fitted model is insufficient, because systematic deviations of the expected value show up. A transformation of the predictor variables may solve the problem.

b) The simulation is performed in analogy to a), but we need to adjust the model: **Python** code:

[5]:
```
# Define x and y:
x = np.log(windmill['wind_speed'])
y = np.log(windmill['current'])
x_sm = sm.add_constant(x)

# Fit the linear model
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))
```

[6]:
```
""" Plots """
# Create Figure and subplots
fig = plt.figure(figsize = (14,9))

# First subplot: Residuals vs Fitted values
ax1 = fig.add_subplot(2, 3, 1)
plot_residuals(ax1, res, yfit)

# Second subplot: QQ Plot
ax2 = fig.add_subplot(2, 3, 2)
plot_QQ(ax2, res_standard)

# Third subplot: Scale-location
ax3 = fig.add_subplot(2, 3, 3)
plot_scale_loc(ax3, yfit, res_stand_sqrt)

# Fourth subplot: Residuals vs Fitted values with 100 resamples
ax4 = fig.add_subplot(2, 3, 4)
plot_residuals(ax4, res, yfit, n_samp = 100)

# Fifth subplot: QQ Plot with 100 resamples
ax5 = fig.add_subplot(2, 3, 5)
plot_QQ(ax5, res_standard, n_samp = 100)

# Sixth subplot: Scale-location with 100 resamples
ax6 = fig.add_subplot(2, 3, 6)
plot_scale_loc(ax6, yfit, res_stand_sqrt, n_samp = 100)

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



- *Tukey-Anscombe*: The banana-like shape of the smoothing line suggests again that the expected value is not constant zero. This structure is as well visible in the graphics containing the simulated curves, where the (red) smoothing line is not entirely part of the band of simulated curves.
- *Normal Plot*: The data points scatter near to the straight line, with the exception of observation 25, which shows a strong deviation from the straight line and falls outside of the band of simulated curves. This outlier seems to violate the normal distribution assumption.
- *Scale-Location*: The smoothing line follows a wave-like curve and, in addition to that, touches the border of the grey region with the simulated curves.

Conclusion: This model fit is as well insufficient and shows peculiarities in all three diagnosis plots.

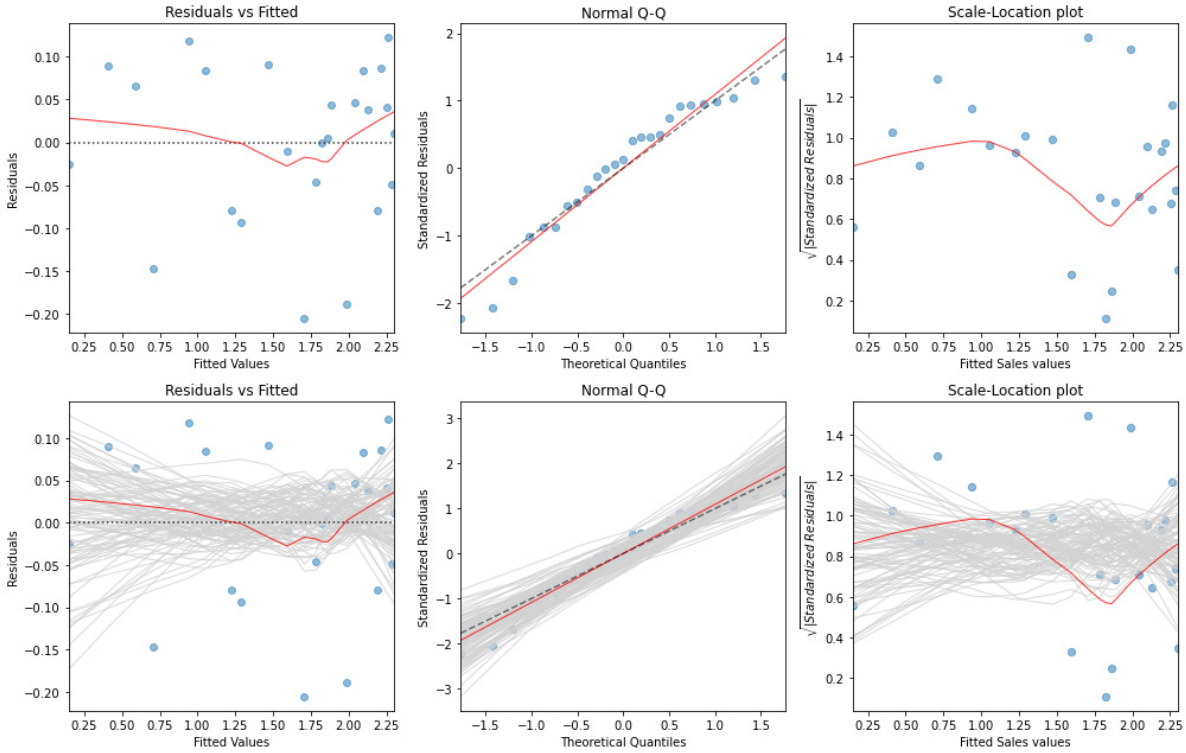
c) The simulations are carried out in analogy to a), however we need to adjust the model: **Python** code:

[7]:
```
# 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()

# 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))
```

[8]:
```
""" Plots """
# Create Figure and subplots
fig = plt.figure(figsize = (14,9))

# First subplot: Residuals vs Fitted values
ax1 = fig.add_subplot(2, 3, 1)
plot_residuals(ax1, res, yfit)

# Second subplot: QQ Plot
ax2 = fig.add_subplot(2, 3, 2)
plot_QQ(ax2, res_standard)

# Third subplot: Scale-location
ax3 = fig.add_subplot(2, 3, 3)
plot_scale_loc(ax3, yfit, res_stand_sqrt)

# Fourth subplot: Residuals vs Fitted values with 100 resamples
ax4 = fig.add_subplot(2, 3, 4)
plot_residuals(ax4, res, yfit, n_samp = 100)

# Fifth subplot: QQ Plot with 100 resamples
ax5 = fig.add_subplot(2, 3, 5)
plot_QQ(ax5, res_standard, n_samp = 100)

# Sixth subplot: Scale-location with 100 resamples
ax6 = fig.add_subplot(2, 3, 6)
plot_scale_loc(ax6, yfit, res_stand_sqrt, n_samp = 100)

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

- *Tukey-Anscombe*: The smoothing line shows a slight banana-like shape, however it does not seem to be problematic according to the graphics with the simulated curves, because the red curve lies inside the grey band of



curves.

- *Normal Plot*: With the exception of the points at the right margin, the data points scatter nicely around the straight line. The right margin points indicate short-tailedness, which does not represent, however, a problem for the least-squares method.
- *Scale-Location*: The smoothing line shows a dent-like pattern, which however does not represent a problem according to the simulated curves.

Conclusion: The residual analysis for this model fit does not show any evidence of violating the assumptions of the linear model.

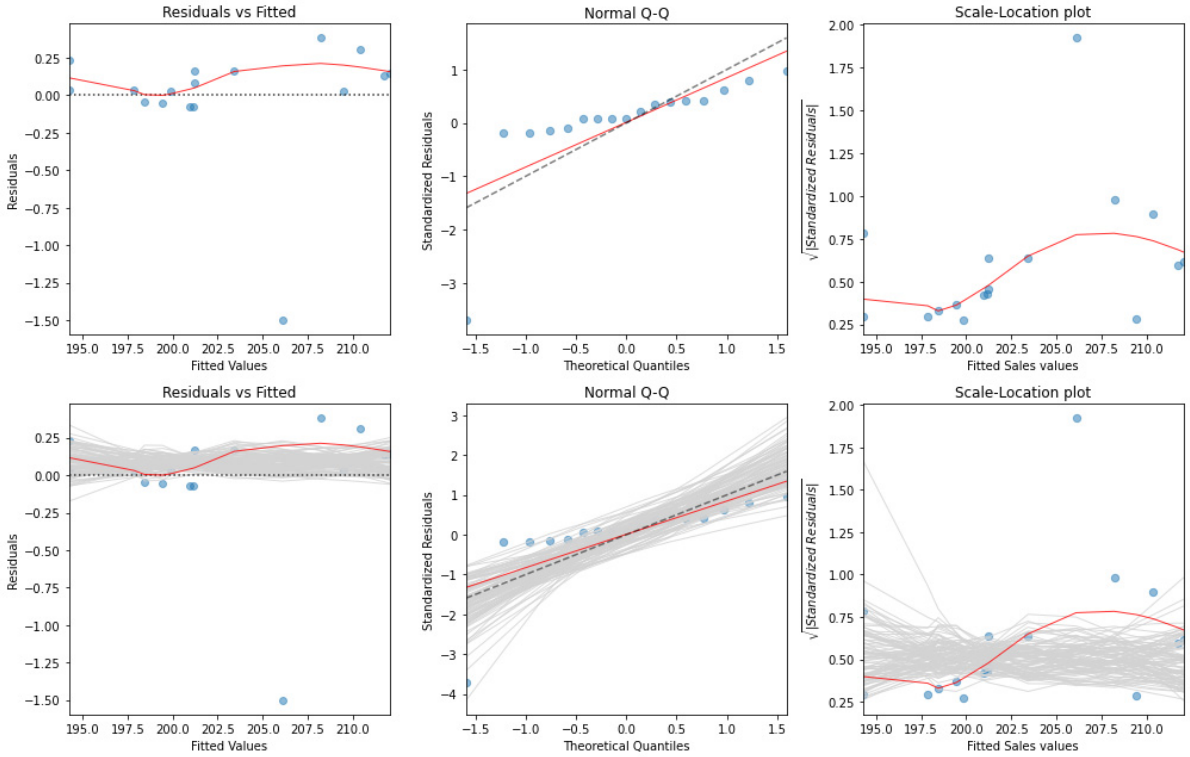
Solution 2.2

The simulation plots are generated in analogy to exercise 1. The definitions shown in Exercise 1 are now saved in *TMA_def.py*. **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')
```

[2]:
```
import statsmodels.api as sm
from TMA_def import *

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

# Fit the linear model
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))
```

The figures are created exactly as in Exercise 1.



- *Tukey-Anscombe*: The smoothing line shows a slightly suspicious pattern. Since the smoother however falls entirely into the grey band of simulated curves, this

pattern does not represent a serious problem.

- *Normal Plot*: The data points scatter nicely around the straight line, with the exception of observation 12, which deviates clearly from the straight line. It lies outside of the grey region. Because of data point 12, the normal distribution assumption is violated.
- *Scale-Location*: The smoother exhibits a suspicious pattern : it departs from the grey band consisting of simulated curves. The normal distribution assumption seems to be violated.
- *Time Correlation*: Because we do not dispose of any information concerning the time order of the measurements, we are not in the position to analyze any time correlations. Such correlations would represent very strong evidence against the independence of the residuals.

Conclusion: The model fit is insufficient and shows rather strong evidence of violation of the assumptions in all three diagnosis plots.

Solution 2.3

a) **Python** code:

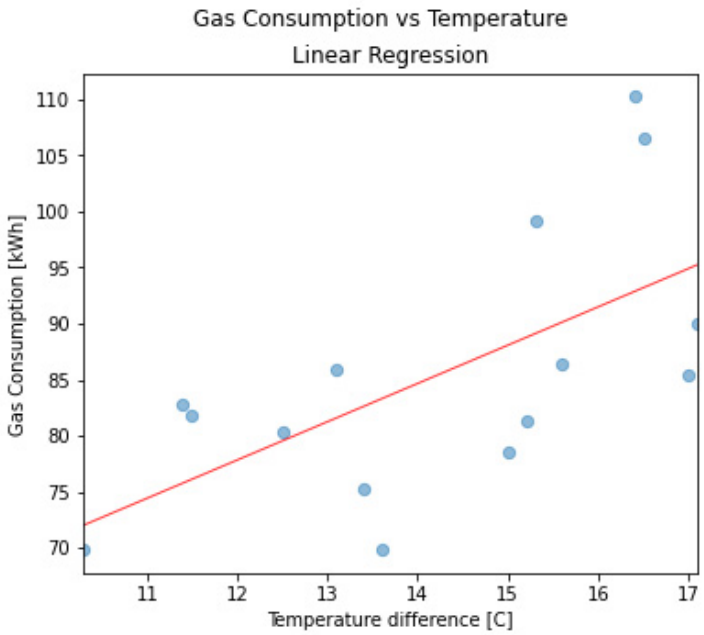
[1]:
```
import pandas as pd
import numpy as np

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

We add a defintion to our set of definitions saved in *TMA_def.py* for a simple scatter plot with regression line.

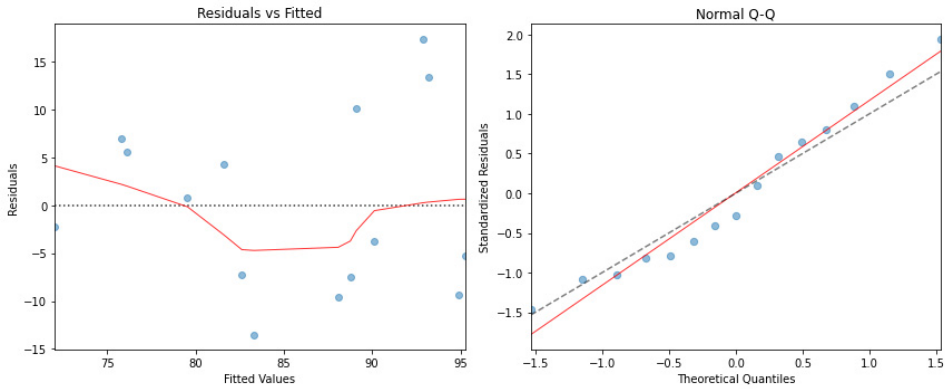
[2]:
```
import seaborn as sns

""" Standard scatter plot and regression line """
def plot_reg(axes, x, y, model, x_lab="x", y_lab="y", title="Linear Regression"):
    """ 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:
```

```
    axes.set_xlabel(x_lab)
    axes.set_ylabel(y_lab)
    axes.set_title(title)
```

[3]:
```
import statsmodels.api as sm
import matplotlib.pyplot as plt

# Define x and y:
x = gas['temperature']
y = gas['consumption']
x_sm = sm.add_constant(x)

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

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

# Create axes in subplots
ax = fig.add_subplot(1, 1, 1)
plot_reg(ax, x, y, model)
ax.set_xlabel('Temperature difference [C]')
ax.set_ylabel('Gas Consumption [kWh]')
fig.suptitle('Gas Consumption vs Temperature')

# show plot
plt.show()
```



[4]:
```
from TMA_def import *

# 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


""" Plots """
# Create Figure and subplots
fig = plt.figure(figsize = (12, 5))

# First subplot: Residuals vs Fitted values
ax1 = fig.add_subplot(1, 2, 1)
plot_residuals(ax1, res, yfit)

# Second subplot: QQ Plot
ax2 = fig.add_subplot(1, 2, 2)
plot_QQ(ax2, res_standard)


plt.tight_layout()
plt.show()
```

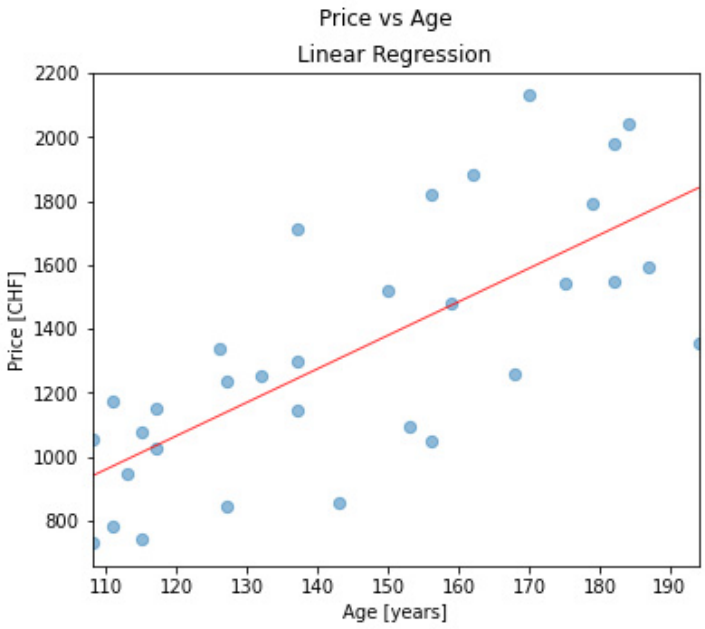


At first sight the scatter plot seems to suggest that the regression line fits well the data. However, the plot *Residuals vs. Fitted* gives another impression. The smoother shows a strong deviation towards the bottom which indicates the presence of a systematic error. Similarly, the variance seems to be larger for large fitted values. The normal plot does not show any abnormalities. Whether the observations are correlated cannot be determined based on these two plots. We would have to know whether the observations were recorded in a temporal order and whether residuals of observations close in time show abnormalities. In

summary, the two assumptions $E(\epsilon_i) = 0$ and $Var(\epsilon_i) = \sigma_\epsilon^2$ might be violated. A log-transformation would yield a much better fit and should be applied.

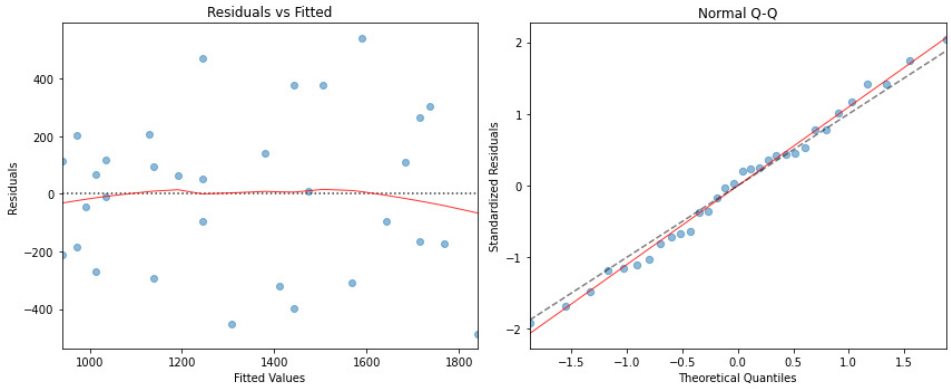
b) **Python** code:

[5]:
```
# 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()

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

# Create axes in subplots
ax = fig.add_subplot(1, 1, 1)
plot_reg(ax, x, y, model)
ax.set_xlabel('Age [years]')
ax.set_ylabel('Price [CHF]')
fig.suptitle('Price vs Age')

# show plot
plt.show()
```



[6]:
```
# 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


""" Plots """
# Create Figure and subplots
fig = plt.figure(figsize = (12, 5))

# First subplot: Residuals vs Fitted values
ax1 = fig.add_subplot(1, 2, 1)
plot_residuals(ax1, res, yfit)

# Second subplot: QQ Plot
ax2 = fig.add_subplot(1, 2, 2)
plot_QQ(ax2, res_standard)


plt.tight_layout()
plt.show()
```



This model shows a good fit. The smoother in the plot *Residuals vs. Fitted* is almost horizontal and does not show systematic deviations from the x-axis. The variance of the data points is approximately constant. It is only slightly smaller on the left which is not a significant problem. The Normal plot does not show any abnormalities. Whether the observations are correlated cannot be determined based on these two plots. These could occur if the clocks were sold at different auctions with systematically larger or smaller prices. This would cause a correlation of the corresponding residuals. In summary, we may consider this model as fitting the data well.

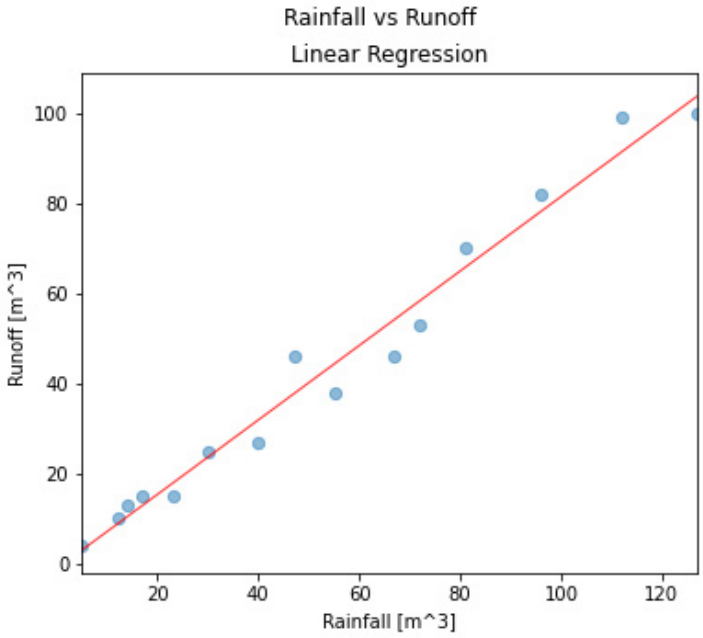
Solution 2.4

a) First we type in the data. The scatter plot of **runoff** versus **rainfall** suggests that a linear relationship applies. **Python** code:

[1]:
```
import statsmodels.api as sm
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from TMA_def import *

rainfall = pd.Series([5, 12, 14, 17, 23, 30, 40, 47, 55, 67, 72, 81,
                      96, 112, 127], name='rainfall')
runoff = pd.Series([4, 10, 13, 15, 15, 25, 27, 46, 38, 46, 53, 70,
                    82, 99, 100], name='runoff')

# Define x and y:
x = rainfall
y = runoff
x_sm = sm.add_constant(x)

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

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

# Create axes in subplots
ax = fig.add_subplot(1, 1, 1)
# Plot regression line and scatter data
plot_reg(ax, x, y, model)
# Set labels:
ax.set_xlabel('Rainfall [m^3]')
ax.set_ylabel('Runoff [m^3]')
fig.suptitle('Rainfall vs Runoff')

# show plot
plt.show()
```

[2]:
```
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 runoff   R-squared:                       0.975
Model:                            OLS   Adj. R-squared:                  0.973
Method:                 Least Squares   F-statistic:                     512.7
Date:                Wed, 03 Mar 2021   Prob (F-statistic):           7.90e-12
Time:                        16:10:52   Log-Likelihood:                -45.057
No. Observations:                  15   AIC:                             94.11
Df Residuals:                      13   BIC:                             95.53
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -1.1283      2.368     -0.477      0.642      -6.244       3.987
rainfall       0.8270      0.037     22.642      0.000       0.748       0.906
==============================================================================
Omnibus:                        0.990   Durbin-Watson:                   2.016
```




```
Prob(Omnibus):                  0.610   Jarque-Bera (JB):                0.705
Skew:                          -0.027   Prob(JB):                        0.703
Kurtosis:                       1.939   Cond. No.                         113.
==============================================================================
```

b) An R^2 of 0.98 is very high, i.e. a large part of the variation in the data can be explained by the linear regression model for the association between **runoff volume** and **rainfall volume**.

c) There is a significant linear association between **runoff volume** and **rainfall volume**, since the null hypothesis $\beta_1 = 0$ can be rejected very clearly.

As estimates we obtain $\hat{\beta}_0 = -1.12$, i.e. when it does not rain, the **runoff volume** is negative. Obviously, this is not possible. Note as well that we do not dispose of any observations at $x = 0$. Thus, interpreting the intercept represents an extrapolation. On the other hand, this shows as well that this model might not be the best. Last, we observe that the estimate of the intercept is not significant.

For the slope we obtain $\hat{\beta}_1 = 0.827$. This value is smaller than 1 (even significantly smaller as the 95 % confidence interval for the slope indicates). Thus, it is statistically shown that not all the rain runs off via the canalization. It is plausible that part of the rain evaporates or trickles away.

d) **Python** code:

[3]:
```
# Prediction at point 50
x0 = [[1, 50]]

pred_x0 = model.get_prediction(x0)
pred_x0 = pred_x0.summary_frame(alpha=0.05)
pred_x0 = np.round(pred_x0, 4)
print(pred_x0)
```

```
      mean   mean_se  mean_ci_lower  mean_ci_upper  obs_ci_lower  obs_ci_upper
0  40.2204    1.3581        37.2863        43.1544        28.525       51.9157
```

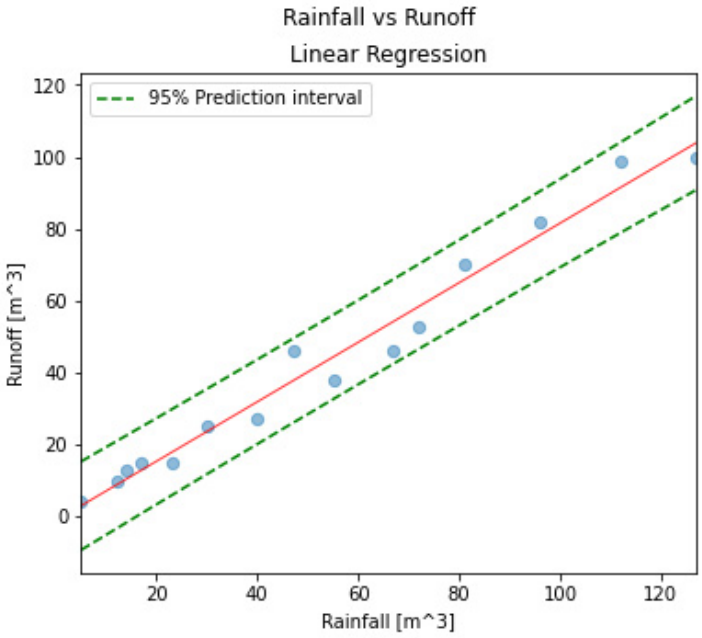
If the **rainfall volume** takes on a value of 50 we find a **runoff volume** of 40.22 with a 95 % prediction interval of $[28.53, 51.92]$.

We can also plot the 95 % prediction interval to the data.

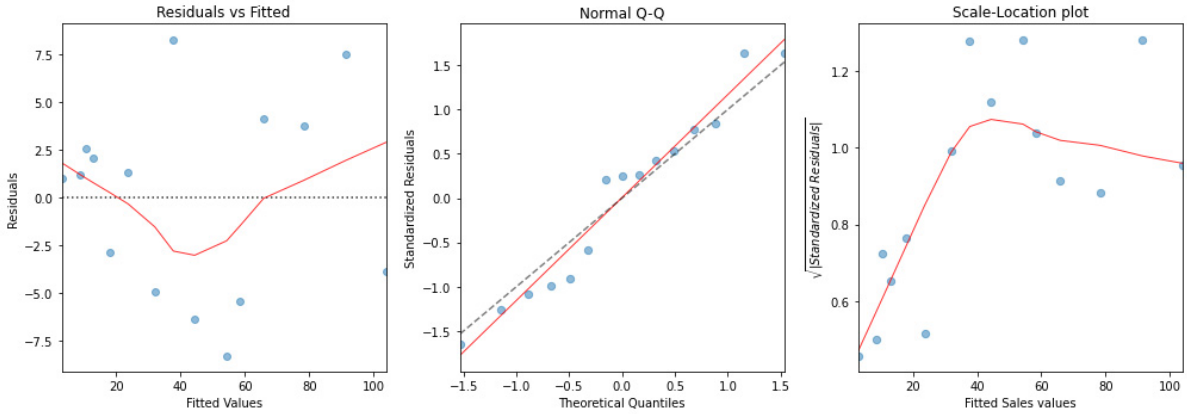
[4]:
```
# Define some points at which to evaluate the prediction
x0 = np.linspace(x.min(), x.max(), 10)
x0 = sm.add_constant(x0)  # Use te known procedure.

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

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

# Create axes in subplots
ax = fig.add_subplot(1, 1, 1)
# Plot regression line and scatter data
plot_reg(ax, x, y, model)
# Plot 99% intervals
ax.plot(x0[:,1], pred0['obs_ci_lower'], '--g',
        label='95% Prediction interval')
ax.plot(x0[:,1], pred0['obs_ci_upper'], '--g')

# Set labels:
ax.set_xlabel('Rainfall [m^3]')
ax.set_ylabel('Runoff [m^3]')
fig.suptitle('Rainfall vs Runoff')
plt.legend()

# show plot
plt.show()
```

e) **Python** code:

[5]:
```
from TMA_def import *

# 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))

""" Plots """
# Create Figure and subplots
fig = plt.figure(figsize = (14,5))

# First subplot: Residuals vs Fitted values
ax1 = fig.add_subplot(1, 3, 1)
plot_residuals(ax1, res, yfit)

# Second subplot: QQ Plot
ax2 = fig.add_subplot(1, 3, 2)
plot_QQ(ax2, res_standard)

# Third subplot: Scale-location
ax3 = fig.add_subplot(1, 3, 3)
plot_scale_loc(ax3, yfit, res_stand_sqrt)

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



We check the following model assumptions

a) Expected value of the errors is zero

b) Constant error variance

c) Normal distribution of errors

with

a) a Tukey-Anscombe plot (residuals vs. fitted values) to assess assumption (i)

b) a scale-location plot to assess assumption (ii)

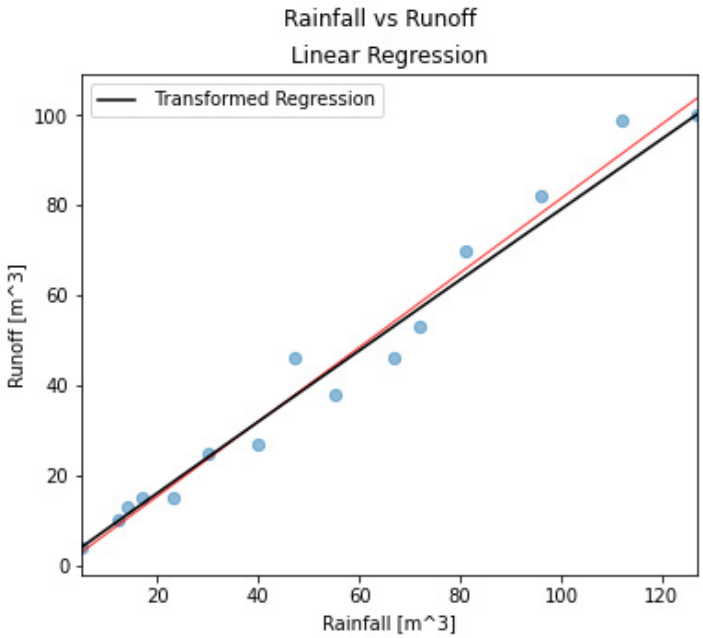
c) a normal plot to assess assumption (iii)

For the assumption of uncorrelated errors there is no suitable plot in this case. However, this is not a reason to assume that the errors are uncorrelated.

While the normal distribution assumption of the errors seems to be satisfied, the Tukey-Anscombe plot (residuals vs. fitted values) raises some doubts. Even though there is a large R^2 and also the test for the slope is highly significant, the expected value of the errors does not seem to be zero. The smoother deviates systematically from the horizontal line. Similarly, it seems like the variance of the residuals is larger at large rainfall values. Thus, the assumption of constant error variance might be violated. As we shall see, we can describe the relation between runoff and rainfall more accurately with a different simple linear regression model.

f) After fitting the new regression model, we need to exponentiate the fitted values in order to add them to the original plot.

[6]:
```
# Define x and y:
x_log = np.log(rainfall)
y_log = np.log(runoff)
x_log_sm = sm.add_constant(x_log)

# Fit the linear model
model_log = sm.OLS(y_log, x_log_sm).fit()

# Define the regression line using regression constants
y_log_reg = model_log.params[0] + model_log.params[1] * x_log

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

# Create axes in subplots
ax = fig.add_subplot(1, 1, 1)
# Plot regression line and scatter data
plot_reg(ax, x, y, model)

ax.plot(x, np.exp(y_log_reg), '-k', label=' Transformed Regression')
# Set labels:
ax.set_xlabel('Rainfall [m^3]')
ax.set_ylabel('Runoff [m^3]')
fig.suptitle('Rainfall vs Runoff')
plt.legend()

# show plot
plt.show()
```



g) **Python** code:

[7]:
```
print(model_log.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 runoff   R-squared:                       0.982
Model:                            OLS   Adj. R-squared:                  0.981
Method:                 Least Squares   F-statistic:                     724.0
Date:                Wed, 03 Mar 2021   Prob (F-statistic):           8.75e-13
Time:                        16:10:53   Log-Likelihood:                 10.478
No. Observations:                  15   AIC:                            -16.96
Df Residuals:                      13   BIC:                            -15.54
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.1837      0.138     -1.331      0.206      -0.482       0.115
rainfall       0.9892      0.037     26.908      0.000       0.910       1.069
==============================================================================
Omnibus:                        1.265   Durbin-Watson:                   2.376
Prob(Omnibus):                  0.531   Jarque-Bera (JB):                0.860
Skew:                          -0.241   Prob(JB):                        0.651
Kurtosis:                       1.930   Cond. No.                         16.6
==============================================================================
```

There has already been a strong relation between **rainfall** and **runoff** without any variable transformation. We found in this case a p-value of 10^{-12} for the null hypothesis $\beta_1 = 0$ and an R^2 of 0.975. The variable transformation leads to a p-value of 10^{-13} for the null hypothesis $\beta_1 = 0$ and an R^2 of 0.982.

In addition, the model with transformed variables should be prefered from a practical point of view since it cannot yield negative runoff values. Further, the coefficient β_1 is easier to interpret. Without transformation, $\hat{\beta}_1 = 0.827$ means that for an additional unit of rain, there are 0.827 additional units of runoff.

If we transform the variables, that is, $Y' = \log(Y)$ and $X' = \log(X)$, then the straight line is fitted on the log-log-scale:

$$Y' = \beta_0' + \beta_1' X' + \epsilon'$$

We can derive the relation on the original scale by taking the exponential function on both sides:

$$Y = \exp(\beta_0') \cdot x^{\beta_1'} \cdot \exp(\epsilon') = \beta_0 \cdot x^{\beta_1} \cdot \epsilon$$

where $\exp(\beta_0') = \beta_0$ and $\beta_1' = \beta_1$. The slope from the log-log-scale is the exponent to x on the original scale. Moreover, we have a multiplicative rather than an additive model, where the error term follows a log-normal distribution. Hence, the errors will scatter more the larger X is, and are skewed towards the right, i.e. larger values. The interesting aspect is the interpretation of the model equation.

If X increases by 1 %, then Y increases by β_1%:

$$\begin{aligned}\tilde{Y} &= \beta_0 \cdot (x(1+0.01))^{\beta_1} \cdot \epsilon \\ &\approx \beta_0 \cdot (x^{\beta_1} + 0.01 \cdot \beta_1) \cdot \epsilon \\ &= Y + \beta_1 \cdot 0.01 \cdot Y\end{aligned}$$

With the transformation, the interpretation is that for 1 % of additional amount of rain, there is 0.989 % of additional runoff. In other words, 98.9 % of the rain runs off via the canalization, the rest evaporates or trickles away.

h) **Python** code:

[8]:
```
# Prediction at point 50
x0 = [[1, np.log(50)]]

pred_log_x0 = model_log.get_prediction(x0)
pred_log_x0 = pred_log_x0.summary_frame(alpha=0.05)
pred_log_x0 = np.round(np.exp(pred_log_x0), 4)
print(pred_log_x0)
```

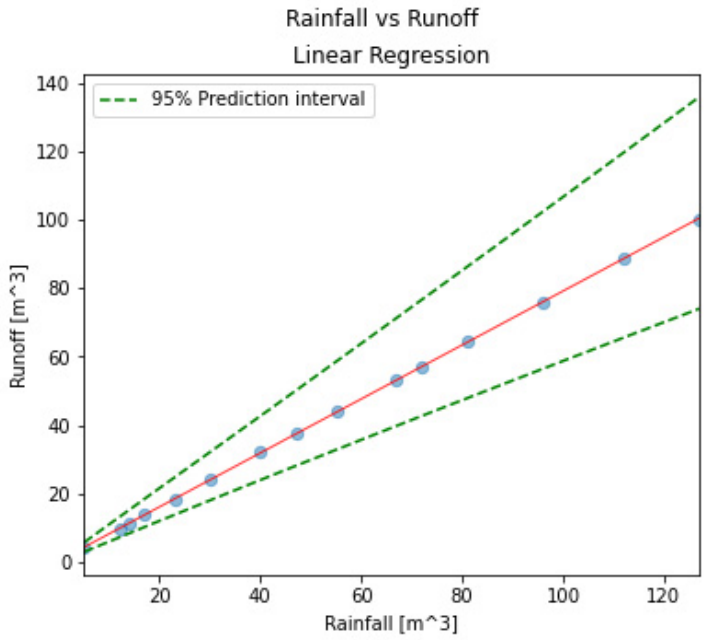
```
     mean  mean_se  mean_ci_lower  mean_ci_upper  obs_ci_lower  obs_ci_upper
0  39.8837   1.0354        36.9948        42.9982       29.8674        53.259
```

Remark for the advanced reader: Note that the point prediction is not the expected value of the response variable but its median.

The predicted value is very close to the one from the model without log-transformations. This however is not true in general. In some cases, the difference can be large. In addition, note that the 95 % prediction interval is no longer symmetric:

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

# Prediction
pred0 = model_log.get_prediction(x0)
pred0 = pred0.summary_frame(alpha=0.05)
pred0 = np.exp(pred0)

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

# Create axes in subplots
ax = fig.add_subplot(1, 1, 1)
# Plot regression line and scatter data
plot_reg(ax, x, np.exp(y_log_reg), model_log)
# Plot 99% intervals
```

```
ax.plot(np.exp(x0[:,1]), pred0['obs_ci_lower'], '--g',
        label='95% Prediction interval')
ax.plot(np.exp(x0[:,1]), pred0['obs_ci_upper'], '--g')

# Set labels:
ax.set_xlabel('Rainfall [m^3]')
ax.set_ylabel('Runoff [m^3]')
fig.suptitle('Rainfall vs Runoff')
plt.legend()

# show plot
plt.show()
```



The asymmetry is not very pronounced in this example. In other cases, it can be much stronger.

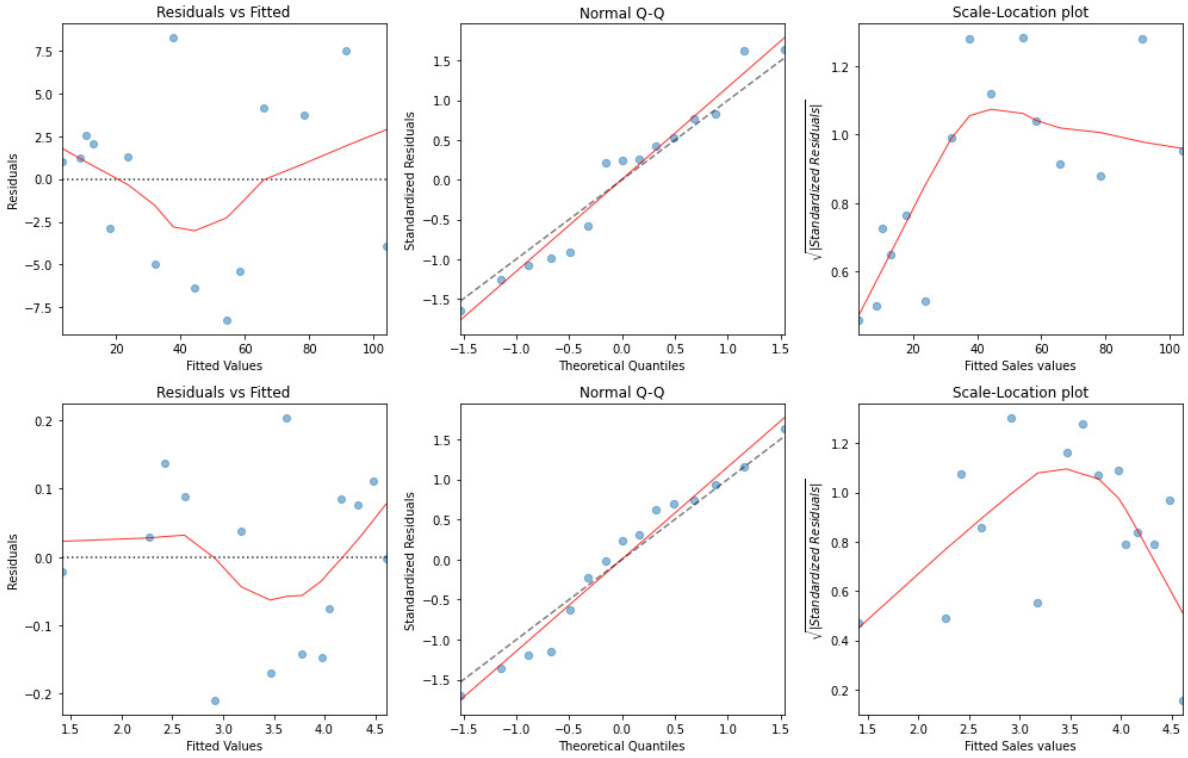
i) The residual plot for the regression model based on transformed variables seems to be more suited than the model on the basis of the original variables.

The improvement is not significant, but the model on the basis of transformed variables seems to have more positive properties. Another advantage of the model with transformed variables is its property of not yielding negative values - neither fitted values nor in the prediction interval. The transformed model predicts a runoff of 0 for a rainfall of 0 which is another desirable property. In summary, there are only small differences between the two models but the model with the transformed variables is more appropriate.

[10]:
```
# Find the predicted values for the original design.
yfit_log = model_log.fittedvalues
# Find the Residuals
res_log = model_log.resid
# Influence of the Residuals
res_inf_log = model_log.get_influence()
# Studentized residuals using variance from OLS
res_standard_log = res_inf_log.resid_studentized_internal
# Absolute square root Residuals:
res_stand_sqrt_log = np.sqrt(np.abs(res_standard_log))

""" Plots """
# Create Figure and subplots
fig = plt.figure(figsize = (14,9))

# First subplot: Residuals vs Fitted values
ax1 = fig.add_subplot(2, 3, 1)
plot_residuals(ax1, res, yfit)

# Second subplot: QQ Plot
ax2 = fig.add_subplot(2, 3, 2)
plot_QQ(ax2, res_standard)

# Third subplot: Scale-location
ax3 = fig.add_subplot(2, 3, 3)
plot_scale_loc(ax3, yfit, res_stand_sqrt)

# First subplot: Residuals vs Fitted values
ax4 = fig.add_subplot(2, 3, 4)
plot_residuals(ax4, res_log, yfit_log)

# Second subplot: QQ Plot
ax5 = fig.add_subplot(2, 3, 5)
plot_QQ(ax5, res_standard_log)

# Third subplot: Scale-location
ax6 = fig.add_subplot(2, 3, 6)
plot_scale_loc(ax6, yfit_log, res_stand_sqrt_log)

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

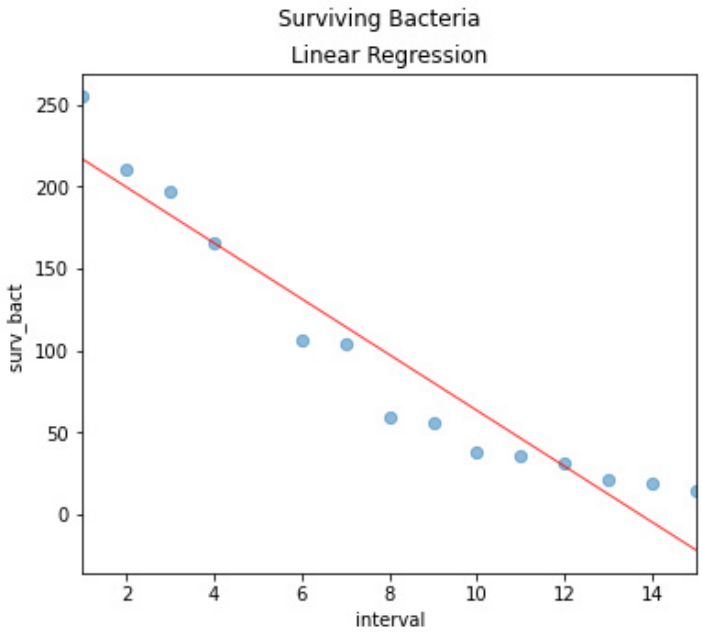
Solution 2.5

a) The scatter plot shows that a least squares regression line does not describe the relation between the two quantities appropriately. We need to apply variable transformations. **Python** code:



[1]:
```
import statsmodels.api as sm
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from TMA_def import *

interval = pd.Series(np.linspace(1, 15, 15), name='interval')
surv_bact = pd.Series([255, 211, 197, 166, np.nan, 106, 104, 60, 56,
                       38, 36, 32, 21, 19, 15], name='surv_bact')

# Define x and y:
x = interval
y = surv_bact
x_sm = sm.add_constant(x)

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

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

# Create axes in subplots
ax = fig.add_subplot(1, 1, 1)
# Plot regression line and scatter data
plot_reg(ax, x, y, model)
```

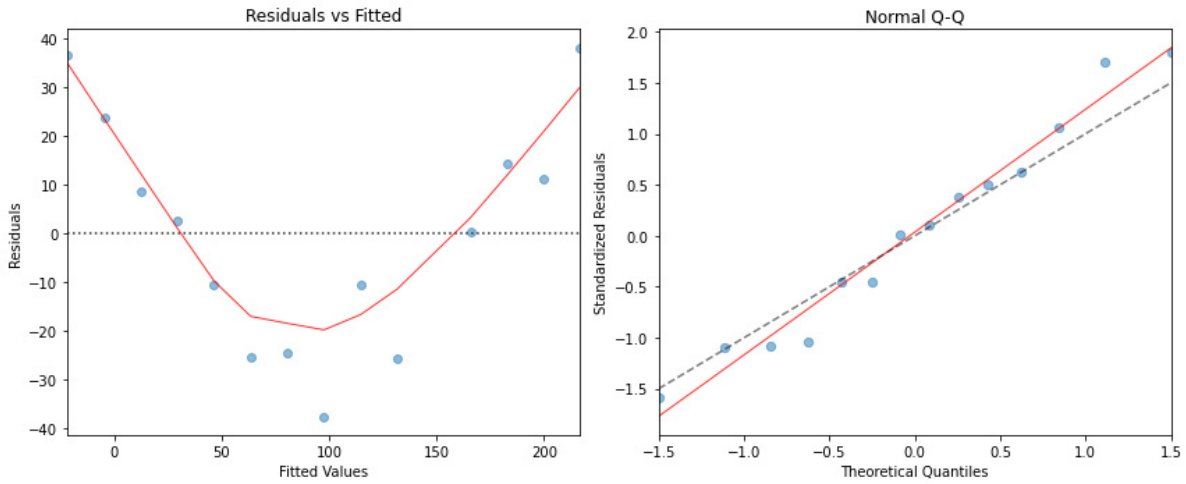
```
# Set labels:
ax.set_xlabel('interval')
ax.set_ylabel('surv_bact ')
fig.suptitle('Surviving Bacteria')

# show plot
plt.show()
```



b) As the *Residuals vs. Predictor* plot shows, the expected value of the errors is not equal to zero. Instead there is a systematic deviation. The picture is typical for situations where a transformation should be applied. The normal plot does not show any conspicuous pattern. Please note, however, that it does not make much sense to study the distribution of the residuals carefully since they result from an incorrect model. The residuals corresponding to the correct model will be different and may have a different distribution. **Python** code:

[2]:
```
from TMA_def import *

# 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
```

```
""" Plots """
# Create Figure and subplots
fig = plt.figure(figsize = (12, 5))

# First subplot: Residuals vs Fitted values
ax1 = fig.add_subplot(1, 2, 1)
plot_residuals(ax1, res, yfit)

# Second subplot: QQ Plot
ax2 = fig.add_subplot(1, 2, 2)
plot_QQ(ax2, res_standard)


plt.tight_layout()
plt.show()
```



c) A log-transformation should be applied to the response variable number of **surviving bacteria**. This variable can only take on positive values. For the predictor, we do not need to apply a transformation.

The scale of the response is arbitrary, negative values could also be used. In addition, the log-response model fulfills the hint given in the problem formulation: per radiation interval the proportion of bacteria that is killed remains constant. Let us denote by $x(t)$ the number of surviving bacteria after time interval t . Then, the proportion of bacteria killed per time interval Δt is given by

$$\frac{x(t+\Delta t) - x(t)}{x(t) \cdot \Delta t}$$

If this proportion is supposed to be constant, then this expression approaches for $\Delta t \to 0$

$$\frac{dx}{dt} = \text{const.} \cdot x(t)$$

The solution to this differential equation is $x(t) = \text{const.} \cdot \exp(t)$, thus the relation between time interval t and number of surviving bacteria x is given by $\log(x(t)) = \log(\text{const.}) + t$.

We thus will fit the following regression model:

$$\log(X) = \beta_0 + \beta_1 \cdot t + \varepsilon$$

The estimate of the coefficient β_1 is

[3]:
```
# Define x and y:
x = interval
y_log = np.log(surv_bact)
x_sm = sm.add_constant(x)

# Fit the linear model
model_log = sm.OLS(y_log, x_sm, missing='drop').fit()

b_1 = model_log.params[1]
print(np.round(b_1, 4))
```

```
-0.2087
```

We interpret $\hat{\beta}_1$ as the amount by which $\log(X)$ is increased (or decreased) after each time interval.

At time $t+1$, X is changed according to

$$X(t+1) = \exp(\beta_0) \cdot \exp(\beta_1 \cdot t + \beta_1) \cdot \exp(\varepsilon) = X(t) \cdot \exp(\beta_1)$$

Therefore, X is increased (or decreased) by the factor $\exp(\hat{\beta}_1)$ after each time interval, that is

[4]:
```
print(np.round(np.exp(b_1), 4))
```

```
0.8116
```

After each interval, 81.16 % of the bacteria remain alive. In other words, on average 18.84 % of the bacteria are killed per interval.

d) The prediction on the original scale for the interval 5 can be obtained as follows:

[5]:
```
x0 = [[1, 5]]

# Prediction
pred0 = model_log.get_prediction(x0)
pred0_95 = pred0.summary_frame(alpha=0.05)
pred0_95 = np.exp(pred0_95)
```

```
print('Expected values at 5:\n', pred0_95[['mean']],
      '\n\n95% Prediction interval:\n',
      pred0_95[['obs_ci_lower', 'obs_ci_upper']],
      '\n\n95% Confidence interval:\n',
      pred0_95[['mean_ci_lower', 'mean_ci_upper']])
```

```
Expected values at 5:
          mean
0  124.067195

95% Prediction interval:
    obs_ci_lower  obs_ci_upper
0      96.297107    159.845602

95% Confidence interval:
    mean_ci_lower  mean_ci_upper
0      114.568661     134.353223
```

The estimate for the relative decrease in the number of surviving bacteria was already discussed above, and found to be 81.16 %. The 95 % confidence interval is given by:

[6]:
```
conf_int = model_log.conf_int().loc['interval']
conf_int = np.exp(conf_int)
print(conf_int)
```

```
0    0.799846
1    0.823593
Name: interval, dtype: float64
```

To estimate the expected number of bacteria at the beginning, we predict for the interval 0, together with the confidence interval:

[7]:
```
x0 = [[1, 0]]

# Prediction
pred0 = model_log.get_prediction(x0)
pred0_95 = pred0.summary_frame(alpha=0.05)
pred0_95 = np.exp(pred0_95)

print('Expected values at 5:\n', pred0_95[['mean']],
      '\n\n95% Confidence interval:\n',
      pred0_95[['mean_ci_lower', 'mean_ci_upper']])
```

```
Expected values at 5:
          mean
0  352.256805
```

```
95% Confidence interval:
    mean_ci_lower  mean_ci_upper
0      307.377488     403.688824
```

Solution 2.6

a) From the plots below we conclude:

- Model assumptions valid
- Model contains strong non-constant variance
- Variance slightly non-constant
- Non-linear model (linear model shows systematic error)

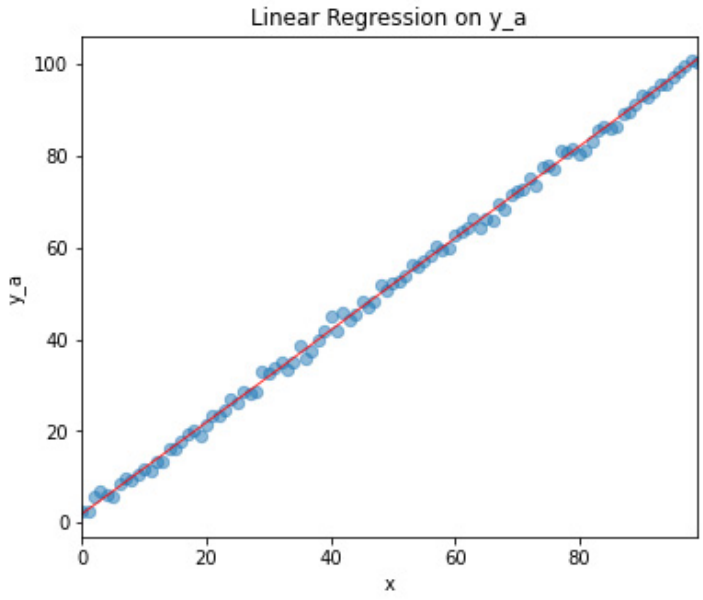
We add a definition to add the Cook's distance to our list of definitions in TMA_def.

[2]:
```
""" Cook's distance """
def plot_cooks(axes, res_inf_leverage, res_standard, x_lim=None, y_lim=None):
    """ Inputs:
    axes: axes created with matplotlib.pyplot
    res_inf_leverage: Leverage
    res_standard: standardized residuals
    x_lim, y_lim[optional]: axis limits """
    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})
    # 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] / (2 * (1 - x_cooks[xi]))
    CS = axes.contour(x_cooks, y_cooks, cooks_distance, levels=4, 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')
```

[3]:
```
from TMA_def import *

# Fit the linear model
model = sm.OLS(y_a, 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
```

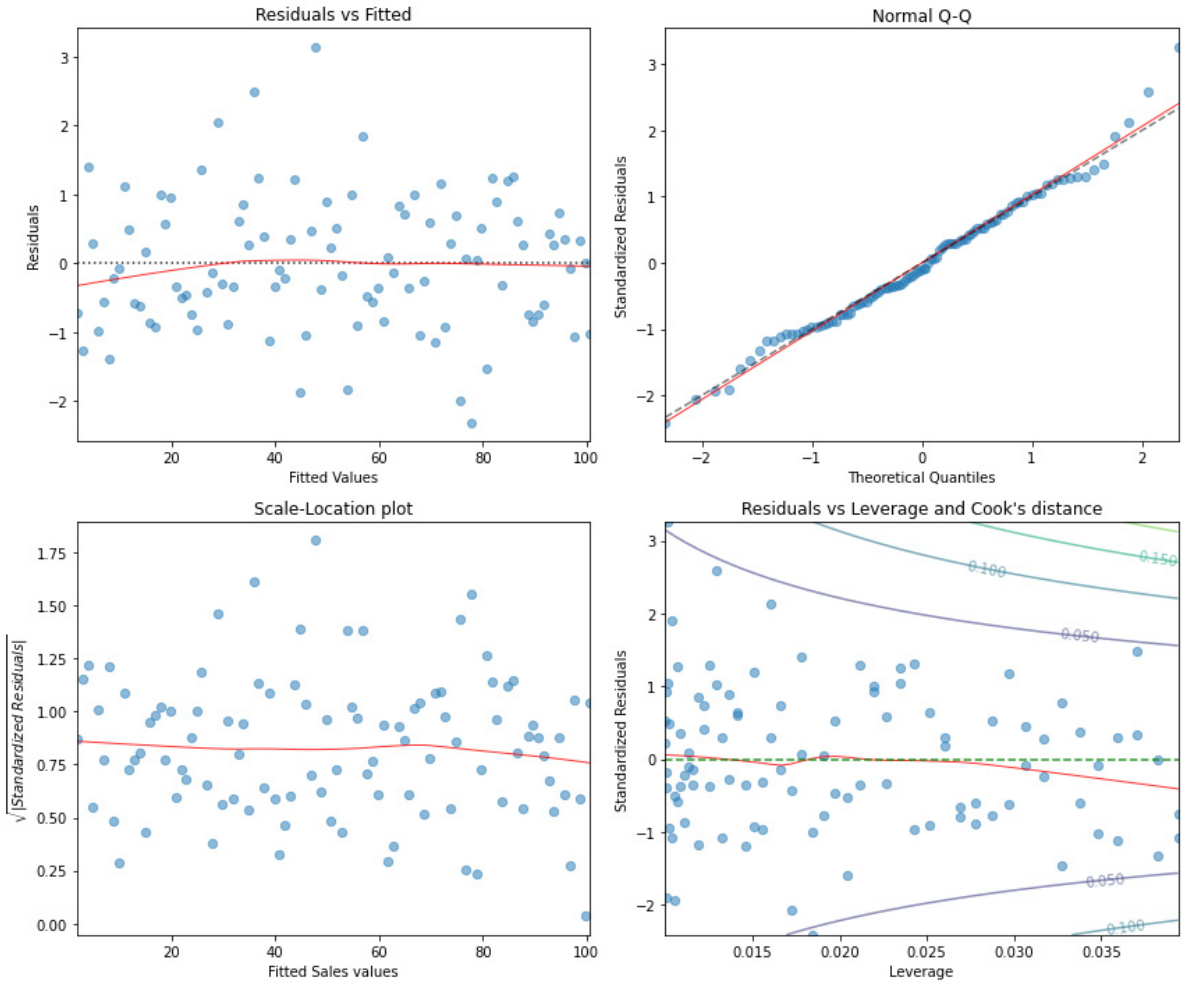
[4]:
```
import matplotlib.pyplot as plt

# Create Figure and subplots
fig = plt.figure(figsize = (6,5))

# First PLot: Scatter plot and linear regression
ax1 = fig.add_subplot(1, 1, 1)
plot_reg(ax1, x, y_a, model, y_lab="y_a",
         title="Linear Regression on y_a")

plt.show()
```



[5]:
```
# Create Figure and subplots
fig = plt.figure(figsize = (12,10))

# First subplot: Residuals vs Fitted values
ax1 = fig.add_subplot(2, 2, 1)
plot_residuals(ax1, res, yfit)

# Second subplot: QQ 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)

# Fourth subplot: Residuals vs Fitted values with 100 resamples
ax4 = fig.add_subplot(2, 2, 4)
plot_cooks(ax4, res_inf_leverage, res_standard)

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

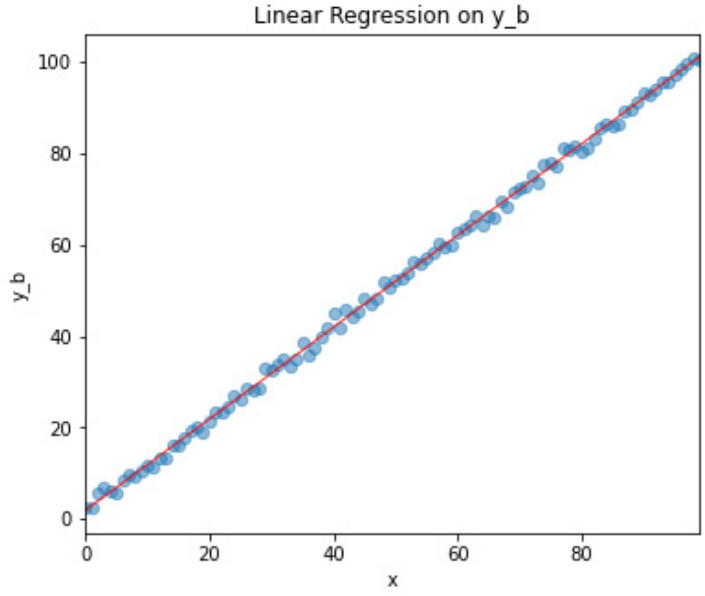
y_a:

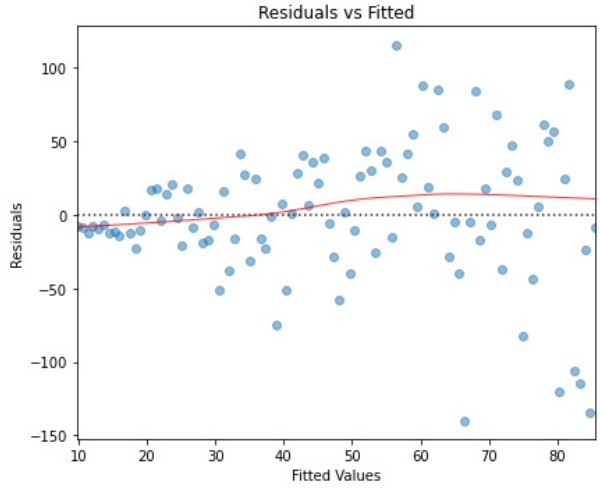
For the first model the residual plots look perfect. Only in the plot displaying Cook's distance, there are a few values that are slightly larger than the rest. These are the observations with the smallest/largest x-values. However, since those values are far from 0.5, there is no problem.

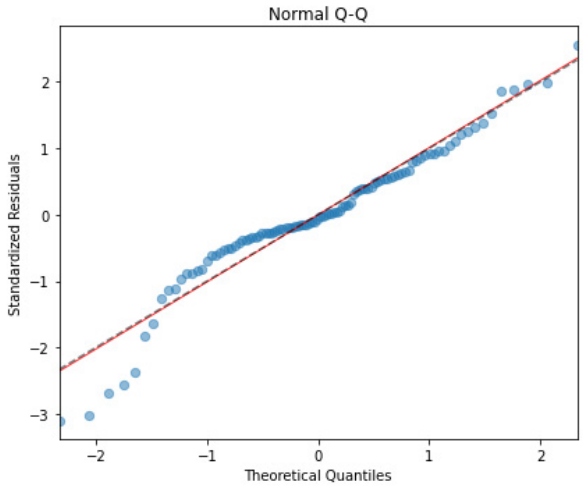


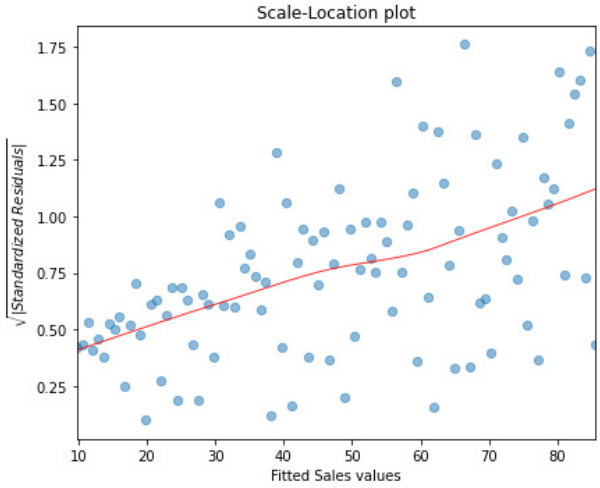
y_b:

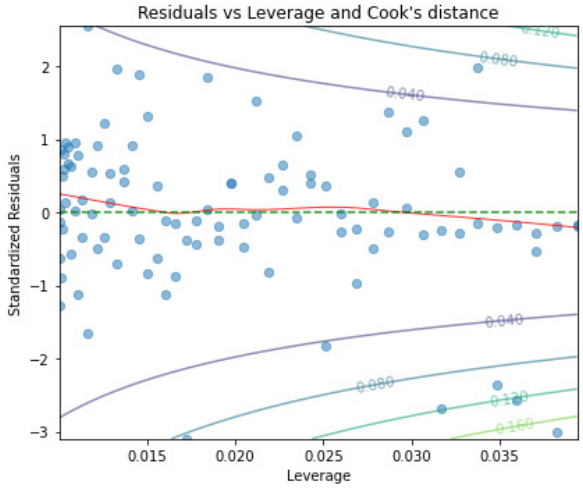
In case of the second model, we see the increasing variance with the magnitude of the fitted values in the Tukey-Anscombe plot. The normal plot shows a violation of the normality assumption, even though the errors do follow a normal distribution per definition. However, the variance is not constant which also needs to be fulfilled for the Normal plot (so that the points follow a straight line). So the violation originates from the fact that the variance is not constant. In the scale-location plot we can also see the increase in the variance. There are no leverage points nor influential data points - even though the points with large observation numbers have larger values of Cook's distance.

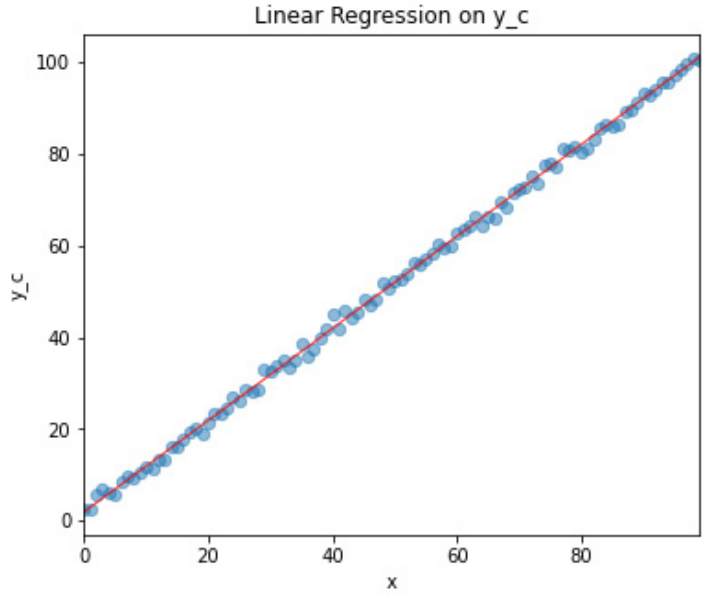

Linear Regression on y_b
y_b
x


Residuals vs Fitted
Residuals
Fitted Values


Normal Q-Q
Standardized Residuals
Theoretical Quantiles

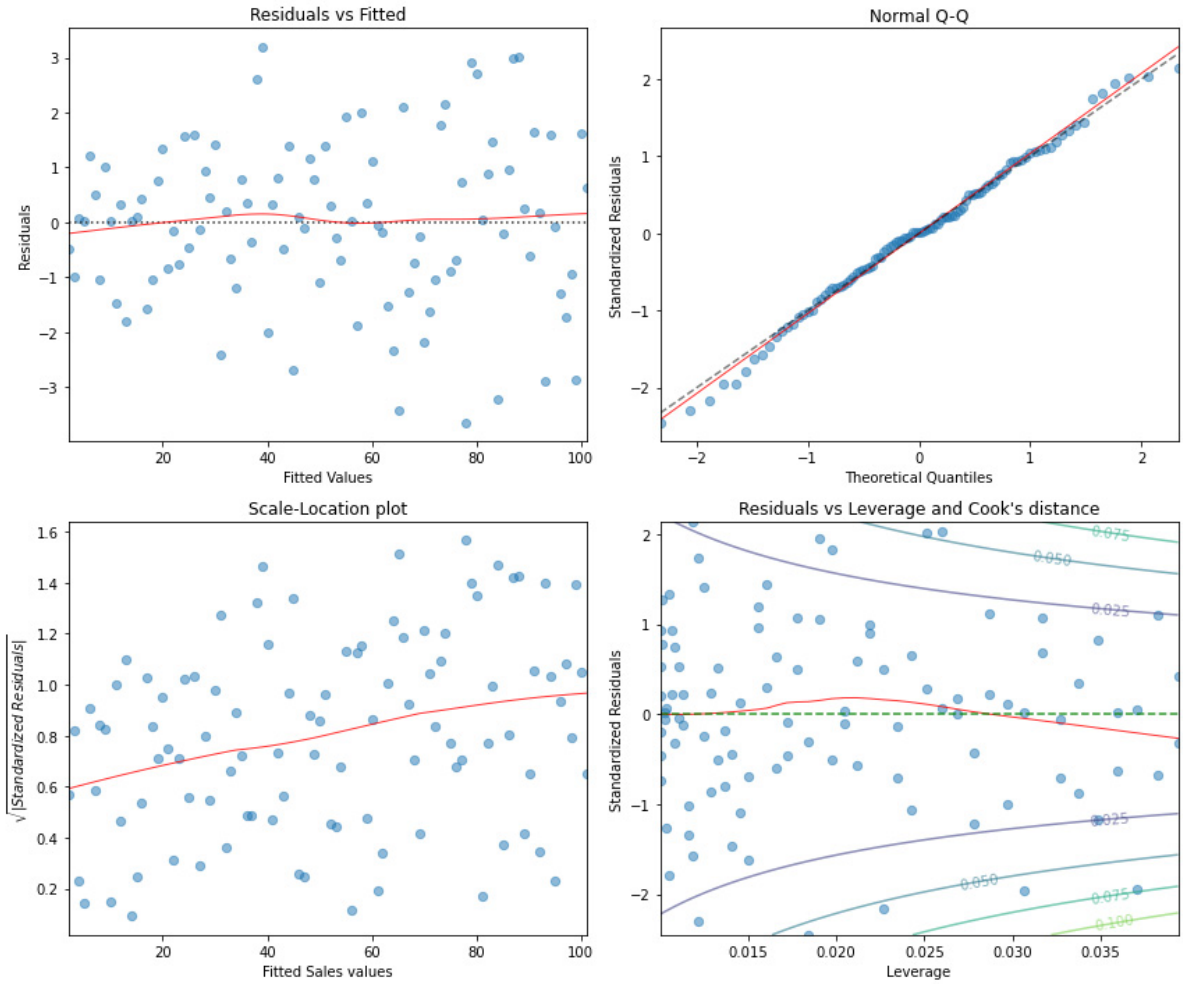

Scale-Location plot
√|Standardized Residuals|
Fitted Sales values


Residuals vs Leverage and Cook's distance
Standardized Residuals
Leverage



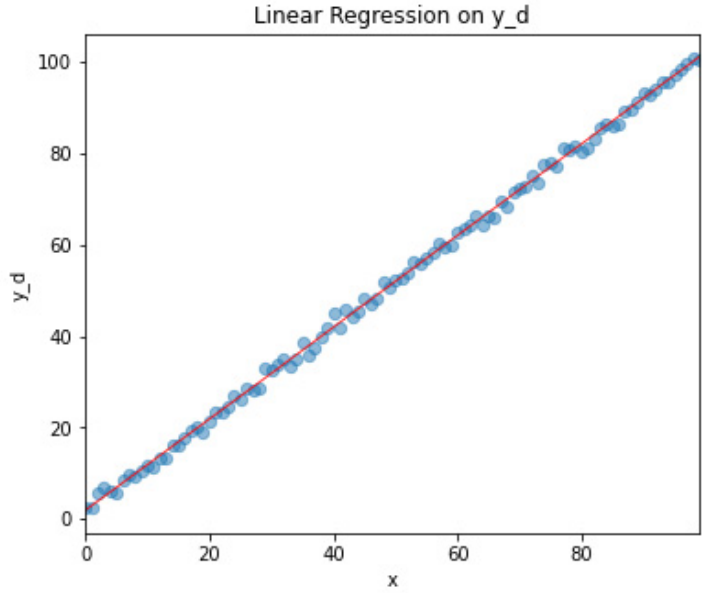
y_c:

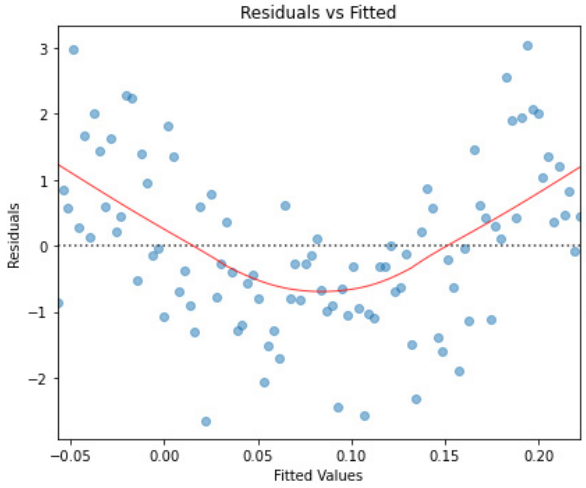
For the third model, the analysis is similar as in case of the second model. This is the case because the model violations are similar. The model violation is less pronounced than in the previous example.

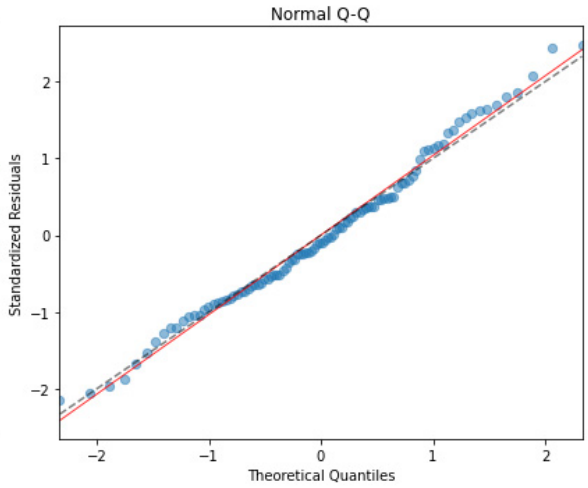


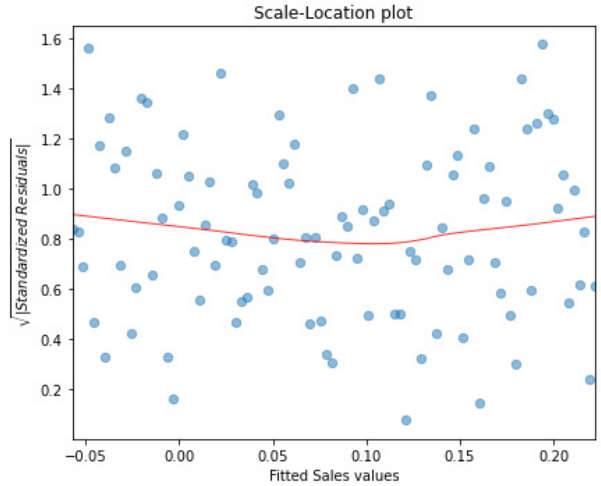
y_d:

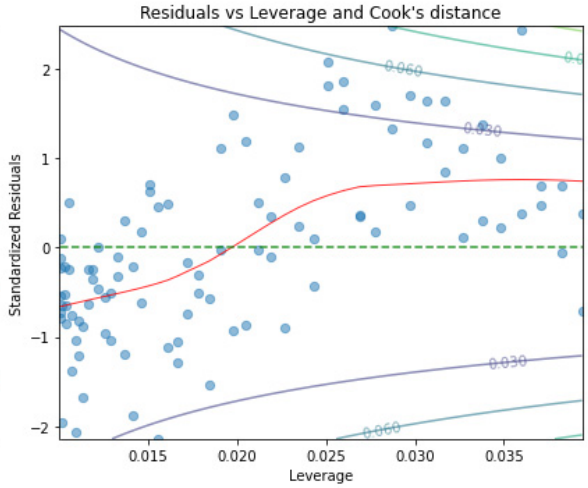
In case of the fourth model, the systematic error can be easily detected in the Tukey-Anscombe plot since it exhibits a U-shaped pattern. The normal plot and the scale-location plot do not show any abnormalities. There are no influential data points but the smoother deviates from the horizontal line in the leverage plot. This may be explained by the points with large leverage (i.e. points at the border of this simple regression) have systematically positive residuals.


Linear Regression on y_d
y_d
x


Residuals vs Fitted
Residuals
Fitted Values


Normal Q-Q
Standardized Residuals
Theoretical Quantiles


Scale-Location plot
Fitted Sales values


Residuals vs Leverage and Cook's distance
Standardized Residuals
Leverage