Multiple_Regression_Kennzahlen_berechnen

April 14, 2023

[ ]:

1 Beispiel: Berechnung der wichtigsten Kennzahlen eines Multiplen Linearen Regressionsmodells

[1]:
```
# Beispieldaten

import pandas as pd

df = pd.DataFrame({"x1": [1,2,3,4,5,6,7,8,9,10],
                   "x2": [3,5,9,9,11,12,15,17,19,21],
                   "y" : [113,117,117,123,135,125,141,143,155,160]})

df
```

[1]:

	x1	x2	y
0	1	3	113
1	2	5	117
2	3	9	117
3	4	9	123
4	5	11	135
5	6	12	125
6	7	15	141
7	8	17	143
8	9	19	155
9	10	21	160

[2]:
```
# Modell erstellen und Kennzahlen ausgeben
from statsmodels.formula.api import ols

model = ols("y~x1+x2", data=df).fit()
model.summary()
```

```
/opt/conda/envs/anaconda-2022.05-py39/lib/python3.9/site-
packages/scipy/stats/stats.py:1541: UserWarning: kurtosistest only valid for
n>=20 … continuing anyway, n=10
```

```
  warnings.warn("kurtosistest only valid for n>=20 … continuing "
```

[2]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.921
Model:                            OLS   Adj. R-squared:                  0.898
Method:                 Least Squares   F-statistic:                     40.64
Date:                Fri, 14 Apr 2023   Prob (F-statistic):           0.000140
Time:                        11:10:36   Log-Likelihood:                -29.036
No. Observations:                  10   AIC:                             64.07
Df Residuals:                       7   BIC:                             64.98
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept    102.5588      5.118     20.040      0.000      90.457     114.660
x1             3.1548      4.825      0.654      0.534      -8.255      14.565
x2             1.0735      2.478      0.433      0.678      -4.785       6.932
==============================================================================
Omnibus:                        1.933   Durbin-Watson:                   2.367
Prob(Omnibus):                  0.380   Jarque-Bera (JB):                1.119
Skew:                          -0.780   Prob(JB):                        0.571
Kurtosis:                       2.498   Cond. No.                         59.7
==============================================================================

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

1.1 Berechne Bestimmtheitsmaß (Determinationskoeffizient) R^2

[3]:
```
y_mean = df.y.mean()

r2 = 1 - (model.resid**2).sum() / ((df.y-y_mean)**2).sum()
print(f"R-squared: {r2}")
```

```
R-squared: 0.9207132025640763
```

1.2 Berechne Adjustiertes Bestimmtheitsmaß R^2_{adj}

[4]:
```
n = df.shape[0]
k = df.shape[1]-1
ddof = n - k - 1 # Freiheitsgrade

r2_adj = 1 - ((1-r2) * (n-1)) / ddof
print(f"Adj. R-squared:         {r2_adj}")
```

```
Adj. R-squared:  0.8980598318680981
```

1.3 F-Statistik

[5]:
```
from scipy.stats import f
F = ddof / k * r2/(1-r2)
print(f"F-statistic: {F}")

p = 1-f.cdf(F, k, ddof)
print(f"Prob (F-statistic): {p}")
```

```
F-statistic: 40.643541083602905
Prob (F-statistic): 0.0001403469782339517
```

1.4 Koeffizienten des Modells mit Moore-Penrose

[6]:
```
import numpy as np

y = np.array(df.y)
X = np.array(df[["x1", "x2"]])
np.ones(10).reshape(10,1)
X = np.hstack((np.ones(10).reshape(10,1), X))

coef = np.linalg.inv(X.T @ X) @ X.T @ y

print(f"Intercept: {coef[0]}")
print(f"x1: {coef[1]}")
print(f"x2: {coef[2]}")
```

```
Intercept: 102.55882352941501
x1: 3.154812834224032
x2: 1.073529411764909
```

1.5 Standardfehler der Koeffizienten

[7]:
```
# Varianz der Residuen
var = np.var(model.resid, ddof=3)

# Varianz-Kovarianz-Matrix
```

```
vcov = var * np.linalg.inv(X.T@X)

# Wurzel aus den Beträgen - In der Diagonalen stehen die Standardfehler
stderr = np.diag(np.sqrt(np.abs(vcov)))
print(stderr)
```

```
[5.11776911 4.82516466 2.47762932]
```

1.6 t-Werte

[8]:
```
## t-Werte
t_val = coef / stderr
t_val
```

[8]:
```
array([20.03975195,  0.65382491,  0.43328895])
```

1.7 p - Werte

[9]:
```
from scipy.stats import t

p = t.sf(t_val, df=ddof)*2
print(p)
```

```
[1.92830809e-07 5.34105953e-01 6.77843088e-01]
```

1.8 Konfidenzintervalle ($\alpha = 0,05$)

[10]:
```
t_krit = t.ppf(.975, df=ddof)

ug = coef - t_krit * stderr
og = coef + t_krit * stderr

np.concatenate([ug.reshape(-1,1),og.reshape(-1,1)], axis=1)
```

[10]:
```
array([[ 90.45722258, 114.66042448],
       [ -8.25488854,  14.56451421],
       [ -4.78513296,   6.93219178]])
```

1.9 Schiefe (Skew) der Residuen

[11]:
```
resid_mean = model.resid.mean()
resid_std = np.std(model.resid)

skew = ((model.resid-resid_mean)**3).sum() / ( (n) * resid_std**3)
print(f"Skew: {skew}")
```

```
Skew: -0.7802110350062288
```

1.10 Kurtosis

[12]:
```
kurtosis = ((model.resid-resid_mean)**4).sum() / ( (n) * resid_std**4)
print(f"Kurtosis: {kurtosis}")
```

```
Kurtosis: 2.498249483711278
```