Matrix-Notation für lineare Modelle

by Woche 8

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
import statsmodels.api as sm
import statsmodels.formula.api as smf
```

In den vorangegangenen Kapiteln haben wir uns mit verschiednen Regressionsmodellen beschäftigt. Dabei haben wir die Modelle in ihrer algebraischen Form dargestellt, z.B. als $y = \beta_0 + \beta_1 x + \varepsilon$ für die einfache lineare Regression.

In diesem Kapitel werden wir eine alternative, aber äquivalente Darstellung dieser Modelle kennenlernen: die **Matrix-Notation**. Diese Darstellung ist aus mehreren Gründen nützlich:

1. Sie vereinheitlicht verschiedene Regressionsansätze unter einer einzigen Notation
2. Sie ermöglicht eine kompakte Darstellung komplexer Modelle
3. Sie offenbart die mathematische Struktur, die hinter den Schätzverfahren steht

Es sei direkt vorweggeschickt: Für die praktische Anwendung von Regressionsmodellen in Python benötigt man die Matrix-Notation nicht unbedingt, da die entsprechenden Funktionen diese Berechnungen im Hintergrund durchführen. Dennoch ist es hilfreich, zumindest einmal gesehen zu haben, was “unter der Haube” passiert.

Das lineare Modell in Matrixform

Die allgemeine Form eines linearen Modells (also alle Modelle aus den letzten Kapiteln) lässt sich in Matrix-Notation wie folgt darstellen:

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$$

Dabei ist:

- n die Anzahl Beobachtungen
- p die Anzahl Prädiktoren
- $\mathbf{y}$ ein $n \times 1$ Vektor der abhängigen Variable (Zielvariable)
- $\mathbf{X}$ eine $n \times (p+1)$ Matrix der unabhängigen Variablen (Design-Matrix oder Modell-Matrix)
- $\boldsymbol{\beta}$ ein $(p+1) \times 1$ Vektor der Regressionskoeffizienten
- $\boldsymbol{\varepsilon}$ ein $n \times 1$ Vektor der Fehlerterme (Residuen)

Die Design-Matrix $\mathbf{X}$ enthält in der ersten Spalte in der Regel Einsen für den Intercept und in den weiteren Spalten die Werte der unabhängigen Variablen für jede Beobachtung.

Beispiel: Einfache lineare Regression in Matrixform

Betrachten wir ein konkretes Beispiel, um das besser zu verstehen. Wir greifen auf den Datensatz aus dem Kapitel 3.5 zurück, der die Beziehung zwischen Alkoholkonsum und Blutalkoholkonzentration untersuchte:

```
# Daten via URL importieren
pfad = "https://raw.githubusercontent.com/SchmidtPaul/ExampleData/refs/heads/main/
drinks/drinks.csv"
```

```
dat = pd.read_csv(pfad)
print(dat)
```

```
   Person  drinks  blood_alc
0     Max       1        0.2
1     Max       2        0.3
2     Max       3        0.5
3     Max       3        0.6
4     Max       4        0.6
5     Max       4        0.5
6     Max       4        0.7
7     Max       5        0.6
8     Max       7        0.8
9     Max       8        1.0
10  Peter       1        0.1
11  Peter       1        0.1
12  Peter       1        0.2
13  Peter       1        0.2
14  Peter       1        0.1
15  Peter       3        0.3
16  Peter       5        0.5
17  Peter       6        0.8
18  Peter       8        0.9
19  Peter       9        1.3
```

In der algebraischen Schreibweise lautet unser Modell:

$$\text{blood\_alc}_i = \beta_0 + \beta_1 \times \text{drinks}_i + \varepsilon_i$$

In Matrix-Notation sieht das Ganze so aus:

$$\begin{bmatrix} \text{blood\_alc}_1 \\ \text{blood\_alc}_2 \\ \vdots \\ \text{blood\_alc}_{20} \end{bmatrix} = \begin{bmatrix} 1 & \text{drinks}_1 \\ 1 & \text{drinks}_2 \\ \vdots & \vdots \\ 1 & \text{drinks}_{20} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_{20} \end{bmatrix}$$

Und mit allen konkreten Werten unseres Datensatzes sieht das so aus:

$$\begin{bmatrix} 0.2 \\ 0.3 \\ 0.5 \\ 0.6 \\ 0.6 \\ 0.5 \\ 0.7 \\ 0.6 \\ 0.8 \\ 1.0 \\ 0.1 \\ 0.1 \\ 0.2 \\ 0.2 \\ 0.1 \\ 0.3 \\ 0.5 \\ 0.8 \\ 0.9 \\ 1.3 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 3 \\ 1 & 4 \\ 1 & 4 \\ 1 & 4 \\ 1 & 5 \\ 1 & 7 \\ 1 & 8 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 3 \\ 1 & 5 \\ 1 & 6 \\ 1 & 8 \\ 1 & 9 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \varepsilon_4 \\ \varepsilon_5 \\ \varepsilon_6 \\ \varepsilon_7 \\ \varepsilon_8 \\ \varepsilon_9 \\ \varepsilon_{10} \\ \varepsilon_{11} \\ \varepsilon_{12} \\ \varepsilon_{13} \\ \varepsilon_{14} \\ \varepsilon_{15} \\ \varepsilon_{16} \\ \varepsilon_{17} \\ \varepsilon_{18} \\ \varepsilon_{19} \\ \varepsilon_{20} \end{bmatrix}$$

Lassen wir die Matrizen explizit für unseren Datensatz aufstellen:

```
# Antwortvariable (y)
y = dat['blood_alc'].values

# Design-Matrix erstellen (X)
X = sm.add_constant(dat['drinks'].values)

print("Antwortvariable y:")
print(y)
print("\nDesign-Matrix X:")
print(X)
```

```
Antwortvariable y:
[0.2 0.3 0.5 0.6 0.6 0.5 0.7 0.6 0.8 1.  0.1 0.1 0.2 0.2 0.1 0.3 0.5 0.8
 0.9 1.3]

Design-Matrix X:
[[1. 1.]
 [1. 2.]
 [1. 3.]
 [1. 3.]
 [1. 4.]
 [1. 4.]
 [1. 4.]
 [1. 5.]
 [1. 7.]
 [1. 8.]
 [1. 1.]
 [1. 1.]
 [1. 1.]
 [1. 1.]
 [1. 1.]
 [1. 3.]
 [1. 5.]
```

```
 [1. 6.]
 [1. 8.]
 [1. 9.]]
```

Ordinary Least Squares (OLS) in Matrixform

In der klassischen linearen Regression schätzen wir die Parameter β mit der Methode der kleinsten Quadrate (OLS). In Matrix-Notation lautet die Formel für die OLS-Schätzung:

$$\hat{\beta} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$$

Dabei ist:

- $\mathbf{X}^\top$ die Transponierte der Design-Matrix
- $(\mathbf{X}^\top \mathbf{X})^{-1}$ die Inverse der Matrix $\mathbf{X}^\top \mathbf{X}$

Wer jetzt nun auch noch wissen möchte wie man das alles selber macht, der kann den gleich angegebenen Links folgen. Wir erledigen das in Python aber wie folgt:

- *Matrix transponieren*: M^T via `M.T` (Video)
- *Matrix invertieren*: M^{-1} via `np.linalg.inv(M)` (Video)
- *Matrizen multiplizieren*: $M \cdot M$ via `M @ M` (Video)

```python
# Schritt 1: X^T (Transponierte von X)
X_T = X.T
print("X^T (Transponierte von X):")
print(X_T)

# Schritt 2: X^T * X
X_T_X = X_T @ X  # @ ist der Matrix-Multiplikations-Operator in Python
print("\nX^T * X:")
print(X_T_X)

# Schritt 3: (X^T * X)^(-1) (Inverse)
X_T_X_inv = np.linalg.inv(X_T_X)
print("\n(X^T * X)^(-1) (Inverse):")
print(X_T_X_inv)

# Schritt 4: X^T * y
X_T_y = X_T @ y
print("\nX^T * y:")
print(X_T_y)

# Schritt 5: (X^T * X)^(-1) * X^T * y = beta_hat
beta_hat = X_T_X_inv @ X_T_y
print("\nbeta_hat:")
print(beta_hat)
```

```
X^T (Transponierte von X):
[[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 2. 3. 3. 4. 4. 4. 5. 7. 8. 1. 1. 1. 1. 1. 3. 5. 6. 8. 9.]]

X^T * X:
[[ 20.  77.]
 [ 77. 429.]]

(X^T * X)^(-1) (Inverse):
[[ 0.16182573 -0.02904564]
```

```
 [-0.02904564  0.00754432]]

X^T * y:
[10.3 55.7]

beta_hat:
[0.04896266 0.12104866]
```

Als Ergebnis erhalten wir die geschätzten Regressionskoeffizienten:

- $\hat{\beta}_0$ (Intercept): ~ 0.049
- $\hat{\beta}_1$ (Steigung für `drinks`): ~ 0.121

Vergleichen wir das mit den Ergebnissen aus Kapitel 3.5:

```python
# Berechnung mit statsmodels
model = sm.OLS(y, X).fit()
print(model.params)
```

```
[0.04896266 0.12104866]
```

Die Ergebnisse stimmen überein! Das müssen sie auch, da `statsmodels` “unter der Haube” auch genau diese Matrixoperationen durchführt. Wir haben die OLS-Schätzung also manuell durchgeführt, um zu zeigen, wie sie in Matrixform aussieht.

Multiple Regression in Matrixform

Wenden wir uns nun der multiplen Regression zu. Hier verwenden wir den Boston Housing Datensatz aus Kapitel 3.6, fokussieren uns aber nur auf die ersten 6 Zeilen, um die Darstellung übersichtlich zu halten:

```python
# Boston Housing Datensatz laden
pfad = "https://raw.githubusercontent.com/SchmidtPaul/ExampleData/refs/heads/main/
boston_housing/boston_house_prices.csv"
boston_full = pd.read_csv(pfad, skiprows=1)

# Auf relevante Spalten und die ersten 6 Zeilen reduzieren
selected_columns = ['RM', 'LSTAT', 'PTRATIO', 'MEDV']
boston = boston_full[selected_columns].head(6).copy()

print(boston)
```

```
      RM  LSTAT  PTRATIO  MEDV
0  6.575   4.98     15.3  24.0
1  6.421   9.14     17.8  21.6
2  7.185   4.03     17.8  34.7
3  6.998   2.94     18.7  33.4
4  7.147   5.33     18.7  36.2
5  6.430   5.21     18.7  28.7
```

Bei der multiplen Regression erweitert sich unsere Design-Matrix um zusätzliche Spalten für jeden Prädiktor:

```python
# Antwortvariable (y)
y_boston = boston['MEDV'].values
```

```
# Design-Matrix erstellen (X)
X_boston = boston[['RM', 'LSTAT', 'PTRATIO']].values
X_boston_with_const = sm.add_constant(X_boston)

print("Antwortvariable y (MEDV):")
print(y_boston)
print("\nDesign-Matrix X (mit Intercept):")
print(X_boston_with_const)
```

```
Antwortvariable y (MEDV):
[24.  21.6 34.7 33.4 36.2 28.7]

Design-Matrix X (mit Intercept):
[[ 1.     6.575  4.98  15.3  ]
 [ 1.     6.421  9.14  17.8  ]
 [ 1.     7.185  4.03  17.8  ]
 [ 1.     6.998  2.94  18.7  ]
 [ 1.     7.147  5.33  18.7  ]
 [ 1.     6.43   5.21  18.7  ]]
```

Unsere Matrix-Gleichung sieht nun so aus:

$$\begin{bmatrix} \text{MEDV}_1 \\ \text{MEDV}_2 \\ \text{MEDV}_3 \\ \text{MEDV}_4 \\ \text{MEDV}_5 \\ \text{MEDV}_6 \\ \vdots \\ \text{MEDV}_n \end{bmatrix} = \begin{bmatrix} 1 & \text{RM}_1 & \text{LSTAT}_1 & \text{PTRATIO}_1 \\ 1 & \text{RM}_2 & \text{LSTAT}_2 & \text{PTRATIO}_2 \\ 1 & \text{RM}_3 & \text{LSTAT}_3 & \text{PTRATIO}_3 \\ 1 & \text{RM}_4 & \text{LSTAT}_4 & \text{PTRATIO}_4 \\ 1 & \text{RM}_5 & \text{LSTAT}_5 & \text{PTRATIO}_5 \\ 1 & \text{RM}_6 & \text{LSTAT}_6 & \text{PTRATIO}_6 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \text{RM}_n & \text{LSTAT}_n & \text{PTRATIO}_n \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \varepsilon_4 \\ \varepsilon_5 \\ \varepsilon_6 \\ \vdots \\ \varepsilon_n \end{bmatrix}$$

Mit den konkreten Werten des Datensatzes (für die ersten 6 Zeilen) sieht das so aus:

$$\begin{bmatrix} 24.0 \\ 21.6 \\ 34.7 \\ 33.4 \\ 36.2 \\ 28.7 \\ \vdots \end{bmatrix} = \begin{bmatrix} 1 & 6.575 & 4.98 & 15.3 \\ 1 & 6.421 & 9.14 & 17.8 \\ 1 & 7.185 & 4.03 & 17.8 \\ 1 & 6.998 & 2.94 & 18.7 \\ 1 & 7.147 & 5.33 & 18.7 \\ 1 & 6.430 & 5.21 & 18.7 \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \varepsilon_4 \\ \varepsilon_5 \\ \varepsilon_6 \\ \vdots \end{bmatrix}$$

Die OLS-Schätzung erfolgt nach derselben Formel wie zuvor:

```
# OLS-Berechnung in Matrix-Notation
X_T = X_boston_with_const.T
X_T_X = X_T @ X_boston_with_const
X_T_X_inv = np.linalg.inv(X_T_X)
X_T_y = X_T @ y_boston
beta_hat_boston = X_T_X_inv @ X_T_y

print("Geschätzte Koeffizienten (Matrix-Berechnung):")
print(beta_hat_boston)

# Vergleich mit statsmodels
model_boston = sm.OLS(y_boston, X_boston_with_const).fit()
```

```
print("\nGeschätzte Koeffizienten (statsmodels):")
print(model_boston.params)
```

```
Geschätzte Koeffizienten (Matrix-Berechnung):
[-64.57085035  10.30645128  -0.82521995   1.60819701]

Geschätzte Koeffizienten (statsmodels):
[-64.57085035  10.30645128  -0.82521995   1.60819701]
```

Die Koeffizienten, die wir durch die Matrixrechnung erhalten, stimmen wieder mit den von statsmodels berechneten Werten überein.

Polynomregression in Matrixform

Bei der Polynomregression, wie wir sie in Kapitel 3.7 kennengelernt haben, fügen wir höhere Potenzen einer unabhängigen Variable als zusätzliche Spalten in die Design-Matrix ein. Betrachten wir beispielsweise eine **quadratische Regression (Polynom 2. Grades)** mit dem Boston Housing Datensatz:

```
# Polynomregression (quadratisch) für die Beziehung zwischen RM und MEDV
X_poly = np.column_stack((
    np.ones(len(boston)),  # Intercept
    boston['RM'].values,   # RM
    boston['RM'].values**2 # RM^2
))

print("Design-Matrix für Polynomregression:")
print(X_poly)
```

```
Design-Matrix für Polynomregression:
[[ 1.        6.575    43.230625]
 [ 1.        6.421    41.229241]
 [ 1.        7.185    51.624225]
 [ 1.        6.998    48.972004]
 [ 1.        7.147    51.079609]
 [ 1.        6.43     41.3449  ]]
```

Die Matrix-Gleichung für dieses Modell lautet:

$$\begin{bmatrix} \text{MEDV}_1 \\ \text{MEDV}_2 \\ \vdots \\ \text{MEDV}_6 \end{bmatrix} = \begin{bmatrix} 1 & \text{RM}_1 & \text{RM}_1^2 \\ 1 & \text{RM}_2 & \text{RM}_2^2 \\ 1 & \text{RM}_3 & \text{RM}_3^2 \\ 1 & \text{RM}_4 & \text{RM}_4^2 \\ 1 & \text{RM}_5 & \text{RM}_5^2 \\ 1 & \text{RM}_6 & \text{RM}_6^2 \\ \vdots & \vdots & \vdots \\ 1 & \text{RM}_n & \text{RM}_n^2 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \varepsilon_4 \\ \varepsilon_5 \\ \varepsilon_6 \\ \vdots \\ \varepsilon_n \end{bmatrix}$$

Mit den konkreten Werten:

$$\begin{bmatrix} 24.0 \\ 21.6 \\ 34.7 \\ 33.4 \\ 36.2 \\ 28.7 \\ \vdots \end{bmatrix} = \begin{bmatrix} 1 & 6.575 & 43.23 \\ 1 & 6.421 & 41.23 \\ 1 & 7.185 & 51.63 \\ 1 & 6.998 & 48.97 \\ 1 & 7.147 & 51.08 \\ 1 & 6.430 & 41.34 \\ \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \varepsilon_4 \\ \varepsilon_5 \\ \varepsilon_6 \\ \vdots \end{bmatrix}$$

Die OLS-Schätzung erfolgt nach derselben Formel:

```
# OLS-Berechnung für Polynomregression
X_T = X_poly.T
X_T_X = X_T @ X_poly
X_T_X_inv = np.linalg.inv(X_T_X)
X_T_y = X_T @ y_boston
beta_hat_poly = X_T_X_inv @ X_T_y

print("Geschätzte Koeffizienten (Polynomregression, Matrix-Berechnung):")
print(beta_hat_poly)

# Vergleich mit statsmodels
model_poly = sm.OLS(y_boston, X_poly).fit()
print("\nGeschätzte Koeffizienten (Polynomregression, statsmodels):")
print(model_poly.params)
```

```
Geschätzte Koeffizienten (Polynomregression, Matrix-Berechnung):
[ 376.71498514 -117.48679859    9.75420269]

Geschätzte Koeffizienten (Polynomregression, statsmodels):
[ 376.71498475 -117.48679848    9.75420268]
```

Varianz-Kovarianz-Matrizen

Wenn wir schon bei Matrizen sind, dann sollten wir auch gleich noch die Varianz-Kovarianz-Matrizen ansprechen. Achtung, das sind jetzt nochmal andere Matrizen als die von eben, aber sie tauchen eben auch im Kontext von linearen Modellen auf und auch diese haben wir die ganze Zeit sozusagen schon benutzt ohne es zu merken. Es gibt zwei zentrale Varianz-Kovarianz-Matrizen in der linearen Regression:

1. Die Varianz-Kovarianz-Matrix der Residuen $\mathrm{Var}\left(\hat{\varepsilon}\right)$
2. Die Varianz-Kovarianz-Matrix der geschätzten Koeffizienten $\mathrm{Var}\left(\hat{\beta}\right)$

Varianz-Kovarianz-Matrix der Residuen

In der Standard-OLS-Regression treffen wir die Annahme, dass die Fehlerterme unabhängig und identisch verteilt sind, mit konstanter Varianz σ^2. Stark vereinfacht könnte man es ausdrücken als “Es gibt einfach eine Fehlervarianz, die für alle Beobachtungen gleich ist”. Aber auch das können wir - ganz und gar nicht vereinfacht - in Matrix-Notation darstellen.

Unsere Residuen in Matrix-Notation kennen wir ja schon:

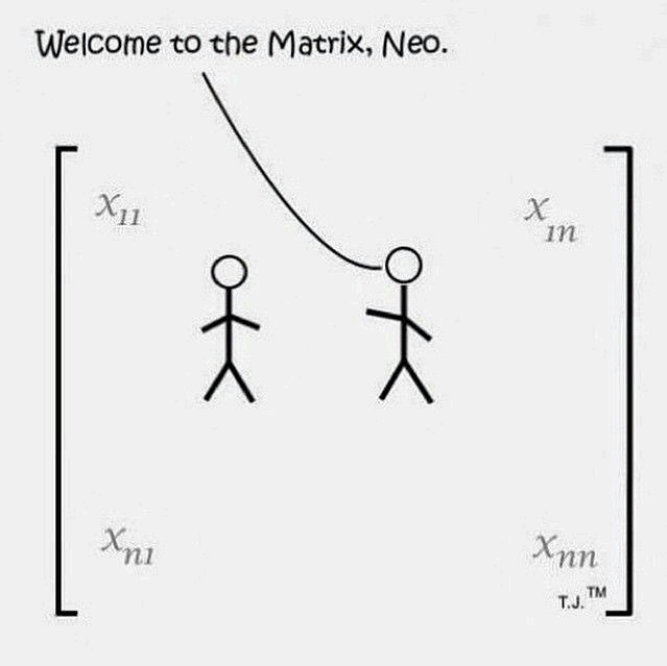
$$\varepsilon = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \vdots \\ \varepsilon_n \end{bmatrix}$$

Die Rede ist jetzt aber davon nicht die Residuen selbst, sondern deren Varianz-Kovarianz-Matrix darzustellen. Diese Matrix beschreibt die Varianz der Residuen und deren Kovarianzen/ Korrelationen untereinander. Während unsere Residuen also ein Vektor mit so vielen Zeilen wie Residuen (=n) war, ist die zugehörige Varianz-Kovarianz-Matrix eine Matrix mit n Zeilen und n Spalten. Auf der Diagonale steht dann jeweils die Varianz und neben der Diagonale die Kovarianz/ Korrelation zwischen den jeweiligen Residuen. Also an Zeile 1, Spalte 1 steht die Varianz von ε_1, an Zeile 2, Spalte 2 die Varianz von ε_2 und so weiter. Und an Zeile 1, Spalte 2 steht die Kovarianz/ Korrelation zwischen ε_1 und ε_2 und so weiter. Normalerweise - also für alle Modelle, die wir bisher kennengelernt haben - ist diese Matrix eine einfache Diagonalmatrix:

$$\text{Var}\left(\boldsymbol{\varepsilon}\right) = \text{Var}\left(\begin{bmatrix}\varepsilon_1\\ \varepsilon_2\\ \varepsilon_3\\ \vdots\\ \varepsilon_n\end{bmatrix}\right) = \sigma^2 \begin{bmatrix}1 & 0 & 0 & \cdots & 0\\ 0 & 1 & 0 & \cdots & 0\\ 0 & 0 & 1 & \cdots & 0\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & 0 & \cdots & 1\end{bmatrix} = \begin{bmatrix}\sigma^2 & 0 & 0 & \cdots & 0\\ 0 & \sigma^2 & 0 & \cdots & 0\\ 0 & 0 & \sigma^2 & \cdots & 0\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & 0 & \cdots & \sigma^2\end{bmatrix} = \sigma^2\mathbf{I}$$

Es sei kurz vorweggegriffen, dass man aber auch andere Annahmen treffen kann. Dies ist lediglich der Standardfall. Falls ihr aber beispielsweise von der Annahme der “Varianzhomogenität” oder “Homoskedastizität” gehört habt, dann ist das genau der Fakt, dass wir hier davon ausgehen, dass alle Residuen gleich stark schwanken, also dieselbe Varianz haben, also auf der Diagonale dieser Matrix überall dasselbe steht. Und wiederum die Annahme der Unabhängigkeit der Residuen passt dazu, dass wir hier neben der Diagonale überall Nullen stehen haben, also dass die Residuen untereinander nicht korreliert sind.

Dazu aber mehr in einem anderen Kapitel.



Varianz-Kovarianz-Matrix der Koeffizienten

Die Varianz-Kovarianz-Matrix der geschätzten Koeffizienten gibt Auskunft über die Präzision unserer Schätzungen und ist die Grundlage für Standardfehler, Konfidenzintervalle und Hypothesentests. Die Varianz-Kovarianz-Matrix der geschätzten Koeffizienten ist gegeben durch:

$$\text{Var}\left(\hat{\boldsymbol{\beta}}\right) = \hat{\sigma}^2(\mathbf{X}^\top\mathbf{X})^{-1}$$

Dabei ist $\hat{\sigma}^2$ die geschätzte Varianz der Residuen (also die von eben aus der Varianz-Kovarianz-Matrix der Residuen), die sich berechnet als:

$\hat{\sigma}^2 = \frac{\sum_{i=1}^{n} \hat{\varepsilon}_i^2}{n-p-1} = \frac{\hat{\boldsymbol{\varepsilon}}^\top \hat{\boldsymbol{\varepsilon}}}{n-p-1}$

wobei $\hat{\varepsilon} = \mathbf{y} - \mathbf{X}\hat{\beta}$ die Residuen und $n - p - 1$ die Freiheitsgrade sind. Berechnen wir dies für unser einfaches lineares Regressionsmodellauch mal selbst:

```
# Geschätzte Koeffizienten für das einfache lineare Modell
beta_hat = np.array([0.04855919, 0.12099565])

# Design-Matrix mit Intercept für das einfache Modell
X = sm.add_constant(dat['drinks'].values)

# Berechnung der Matrizen für dieses Modell
X_T = X.T
X_T_X = X_T @ X
X_T_X_inv = np.linalg.inv(X_T_X)

# Geschätzte y-Werte
y_hat = X @ beta_hat

# Residuen
residuals = y - y_hat

# Geschätzte Varianz der Residuen
n = len(y)
p = 1  # Anzahl der Prädiktoren (ohne Intercept)
sigma_squared = np.sum(residuals**2) / (n - p - 1)

# Varianz-Kovarianz-Matrix der Koeffizienten
var_cov_beta = sigma_squared * X_T_X_inv
print("Varianz-Kovarianz-Matrix der Koeffizienten:")
print(var_cov_beta)

# Standardfehler der Koeffizienten
se_beta = np.sqrt(np.diag(var_cov_beta))
print("\nStandardfehler der Koeffizienten:")
print(se_beta)

# Vergleich mit statsmodels
model = sm.OLS(y, X).fit()
print("\nStandardfehler laut statsmodels:")
print(model.bse)
```

```
Varianz-Kovarianz-Matrix der Koeffizienten:
[[ 1.64776341e-03 -2.95752406e-04]
 [-2.95752406e-04  7.68188068e-05]]

Standardfehler der Koeffizienten:
[0.04059265 0.00876463]

Standardfehler laut statsmodels:
[0.04059179 0.00876445]
```

Die Hauptdiagonale der Varianz-Kovarianz-Matrix enthält die Varianzen der einzelnen Koeffizienten. Die Wurzel daraus ergibt die Standardfehler, die für die Berechnung von t-Werten und Konfidenzintervallen verwendet werden.

Die Nicht-Diagonalelemente der Matrix geben die Kovarianzen zwischen den geschätzten Koeffizienten an. Diese sind wichtig, um zu verstehen, wie stark die Schätzungen miteinander zusammenhängen. Eine negative Kovarianz zwischen zwei Effekte bedeutet beispielsweise, dass

wenn man den einen Effekt überschätzt, dass der andere Effekt dann tendenziell unterschätzt wird.

Zusammenfassung

In diesem Kapitel haben wir die lineare Regression aus einer neuen Perspektive betrachtet: der Matrix-Notation. Wir haben gesehen, dass:

1. Lineare Modelle kompakt als $\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$ dargestellt werden können
2. Die OLS-Schätzung in Matrix-Form als $\hat{\boldsymbol{\beta}} = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}$ berechnet wird
3. Die Design-Matrix $\mathbf{X}$ flexibel erweitert werden kann für:
   - Multiple Regression (zusätzliche Spalten für weitere Prädiktoren)
   - Polynomregression (zusätzliche Spalten für höhere Potenzen)
4. Die Varianz-Kovarianz-Matrix wichtige Informationen über die Präzision der Schätzungen liefert

Die Matrix-Notation vereinheitlicht verschiedene Regressionsansätze unter einer gemeinsamen mathematischen Darstellung und bildet die Grundlage für viele fortgeschrittene statistische Verfahren.

Auch wenn man für die praktische Anwendung in Python die explizite Matrix-Notation nicht benötigt, hilft das Verständnis dieser Grundlagen dabei, die Theorie hinter den Methoden besser zu verstehen und komplexere Modelle in einem einheitlichen Rahmen zu betrachten.

In diesem Kapitel gibt es weder weiterführende Ressourcen, noch Übungen.