Generalisierte Regression

Joern Ploennigs · AI4SC

> Das Leben ist die Summe all unserer Entscheidungen
>
> — Albert Camus

Generalisierte Lineare Modelle

Generalisierte Lineare Modelle (GLM) sind eine Verallgemeinerung der Linearen Modelle, die wir bereits kennen gelernt haben. Sie bilden einen gemeinsamen Rahmen, der zum Beispiel Lineare Regressionsmodelle und Logistische Regressionsmodelle umfasst. Sie folgen im Wesentlichen der Idee, die wir schon bei der logistischen Regression kennen gelernt haben, dass wir eine lineare Basisfunktion z_i haben:

$$z_i = \beta_0 + \beta_1 \cdot x_{1,i} + \beta_2 \cdot x_{2,i} + ... + \beta_n \cdot x_{n,i}.$$

GLMs sind flexibel und erlauben den Einsatz verschiedener Zielverteilungen, das ist besonders wichtig, wenn klassische lineare Regression ungeeignet ist. Sie bilden oft die Grundlage für Klassifikations- und Regressionsmodelle in ML-Bibliotheken wie `statsmodels` oder `scikit-learn`.

die wir über eine *Verknüpfungsfunktion* $g(z_i)$ in eine Vorhersage unserer Zielvariable $\hat{z}_i$ transformieren

$$y_i = g(z_i) + \varepsilon.$$

Die Verknüpfungsfunktion übersetzt das lineare Modell (z.B. eine Temperatur-Zeit-Funktion) in die passende Skala der Zielgröße, wie zum Beispiel Wahrscheinlichkeiten, Zählwerte oder logarithmierte Kosten.

Diese Verknüpfungsfunktion kann eine Sigmoid-Funktion sein (Logistische Regression - Ob ein Betonbauteil unter Last versagt (Ja/Nein), eine Exponential-Funktion (Poisson Regression - Wie viele Fahrzeuge über eine Brücke pro Stunde fahren), eine Gamma-Funktion (Gamma Regression -

Prognose von Baukosten oder Materialverbrauch) oder eine unveränderte z_i (Lineare Regression). Das erlaubt je nach Verteilungstyp der Zielvariable Eingangswerte zu transformieren, die anderen Verteilungen folgen und somit Variablen, die eigentlich andere Verteilungsformen haben in einem Vorhersagemodell kombinieren. Somit muss der Vorhersagefehler ε auch nicht mehr Normalverteilt sein, sondern kann andere Verteilungsformen annehmen.

Das Entscheidende hierbei ist, dass die Basisfunktion z_i immer noch linear ist und nur transformiert wird, das bedeutet, dass wenn sich die unabhängigen Variable $x_{k,i}$ ändert, sich auch die abhängige Variable y_i ändert, unabhängig vom Wert der anderen unabhängigen Variablen.

Die Koeffizienten $\beta_0, \beta_1, ..., \beta_n$ werden auch hier durch die schon diskutierte *Maximum-Likelihood-Schätzung (MLE)* bestimmt und zum Beispiel durch das Newtonverfahren iterativ angenähert.

```
Unable to display output for mime type(s): text/html
```

Modell	$g^{-1}(z)$	Wertebereich	Anwendung
Linear	z	$\mathbb{R}$	Stetige Werte
Logistisch	$\frac{1}{1+e^{-z}}$	$(0,1)$	Wahrscheinlichkeiten
Poisson	e^z	$\mathbb{R}_+$	Zählwerte
Gamma	$\frac{1}{z}$	$\mathbb{R}_+$	Kosten, Dauern

Generalisierte Additive Modelle

Bisher wurde immer eine Lineare Beziehung zwischen den Eingangsvariablen und der Zielvariable angenommen. Das sorgt für Modelle mit einfach zu verstehenden Parametern $\beta_0, \beta_1, ..., \beta_n$. Dadurch sind die Modelle einfach zu berechnen und auch Praktikern gut zu erklären.

Allerdings ist die Annahme eines linearen Zusammenhanges für viele praktische Szenarien nicht immer haltbar, da z.B. in der Physik nichtlineare Zusammenhänge häufig vorkommen. Deshalb haben sich *Generalisierte Additive Modelle (GAM)* entwickelt, die die Idee der Verknüpfungsfunktion aus GLM übernehmen aber direkt auf die additiven einzelnen Elemente der linearen Gleichung anwenden. Dadurch das resultierende Modell zwar *additiv*, wie das lineare Modell, aber nicht notwendigerweise *linear*, da jede einzelne Transformationsfunktion $s_j(x_{j,i})$ nichtlinear sein kann.

$$y_i = \beta_0 + s_1(x_{1,i}) + s_2(x_{2,i}) + ... + s_n(x_{n,i}) + \varepsilon = \beta_0 + \sum_{j=1}^{n} s_j(x_{j,i}) + \varepsilon.$$

Die Schwierigkeit besteht darin die Transformationsfunktion $s_j(x_{j,i})$ in ihren Beiträgen zu y_i zu bestimmen. Hierfür nimmt man an, dass $s_j(x_{j,i})$ eine *glatte* Funktion ist (stetig und beliebig oft differenzierbar (zum Fitting)). Hierfür werden meist Spline-Funktionen benutzt.

Viele moderne Implementierungen von GAMs nutzen zum Bestimmen der Transformationsfunktionen den Ansatz der Rankreduktion(Reduced-rank Minimization), da er eine schnelle Schätzung der Parameter ermöglicht. Hierfür wird für die Transformationsfunktion angenommen, dass sie ein Spline aus K_j Basisfunktionen ist, die gegeben ist durch

$$s_j(x_j) = \sum_{k=1}^{K_j} \beta_{j,k} b_{j,k}(x_j)$$

wobei die $b_{j,k}(x_j)$ bekannte Basisfunktionen sind, die in der Regel aufgrund guter Approximationseigenschaften gewählt werden (z.B. B-Splines) und die $\beta_{j,k}$ Koeffizienten sind, die im Rahmen der Modellanpassung geschätzt werden müssen. Die Basisdimension K_j sollte dabei so gewählt werden, dass eine gute Anpassung der vorliegenden Daten zu erwarten ist, aber klein genug, um die Effizienz der Berechnung zu erhalten.

Durch ersetzen aller Transformationsfunktionen $s_j(x_{j,i})$ durch Basisfunktionen, können diese in bekannter Weise durch Ableitung iterativ die Koeffizienten $\beta_{j,k}$ bestimmen können.

$$y_i = \beta_0 + \sum_{j=1}^{n} \sum_{k=1}^{K_j} \beta_{j,k} b_{j,k}(x_j) + \varepsilon.$$

GAM in Python mit Statsmodels

Für GAM Modelle eignet sich insbesondere die *statsmodels* Bibliothek. Als Beispiel nutzen wir wieder den Energiedatensatz der Universität Rostock. Wir wollen noch einmal den Energiekonsum vorhersagen.

```
import pandas as pd # Import von Pandas
import plotly.express as px # Import von Plotly

egywth = pd.read_csv("../data/UROS/Energy1D_weather_clean.csv",
parse_dates=[0])
egywth["Weekday"] = egywth["Date"].dt.day_name()
egywthN=egywth.dropna().copy()
egywthN.head()
```

	EV...	...	..UPMTXKTNKTGK	eor	Temperatur Klasse	Heizg-Tage	...	NF_4	Weekday
2	2021-01-01 00:00:00+00:00	...	2021-01-01 00:00:00+00:00	90.0	3.0	1.1	0.3	eor	Cold	Heizg-Tage	nicht allle	5	Friday

	EV_HE_N4_7AE_Sab_DATUM... (overlapping headers as printed: "EVDHE_N4E_7AE_Sab_DaSIIAM3S9SDAATUM")	QN_3	...	UPM	TXK	TNK	TGK	eor	Temperatur-Klasse	Heiz-Kuehl-Tage	QN_SQ_4	NF_4	Weekday
										radtag	Parameter korrigiert		
3	2021-01-02 4301 24328 4.0 2.0 00:00:00+00:00 (overlapping as printed)		20214-01-02 2021 1.02.0 00:00:00+00:00 (overlapping as printed)	95.0	4.0	2.6	2.0	eor	Cold	Heizgradtag	nicht alle Parameter korrigiert	5	Saturday
4	2021-01-03 7501 22283.0 2.0 00:00:00+00:00 (overlapping as printed)		20214-01-03 2021 1.03.0 00:00:00+00:00 (overlapping as printed)	81.0	4.6	2.4	0.8	eor	Cold	Heizgradtag	nicht alle Parameter korrigiert	5	Sunday
5	2021-01-04 3011 26329 5.0 5.0 00:00:00+00:00 (overlapping as printed)		20214-01-04 2021 1.04.0 00:00:00+00:00 (overlapping as printed)	84.0	4.4	3.0	2.4	eor	Cold	Heizgradtag	nicht alle Parameter	5	Monday

	EVDate_N40_7AE_SV_DESLAMESSDATUM	QN_3	...UPM	TXK	TNK	TGK	eor	Temperatur-Klasse	Heiz-Kuehl-Tage	QN_S_4	NF_4 korrigiert	Weekday
6	2021341-01 12013462 99.0 1.0 00:00:00+00:00	2021401-26 2021001.05...0 00:00:00+00:00	88.0	4.7	3.0	2.6	eor	Cold	Heizgradtag	nicht alle Parameter korrigiert	5	Tuesday

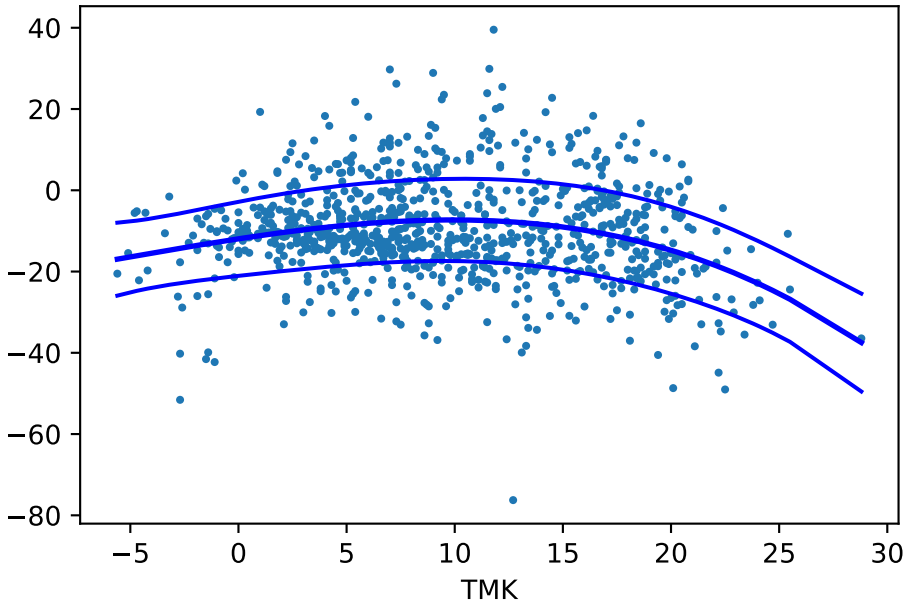
Um ein GAM-Modell zu erzeugen, nutzen wir die GAM API von Statsmodels. Dafür müssen wir separat die B-Splines initialisieren.

```
import statsmodels.api as sm
from statsmodels.gam.api import GLMGam, BSplines

bs = BSplines(egywthN[['TMK', "SDK", "NM", "VPM"]], df=[4, 4, 4, 4],
degree=[3, 3, 3, 3])

# Fit the GAM model
smfgam = GLMGam.from_formula(formula="ES_Lab ~ TMK + SDK + NM + VPM +
Weekday", data=egywthN, smoother=bs)
smfgam = smfgam.fit()
egywthN["gam"]=smfgam.predict()
```

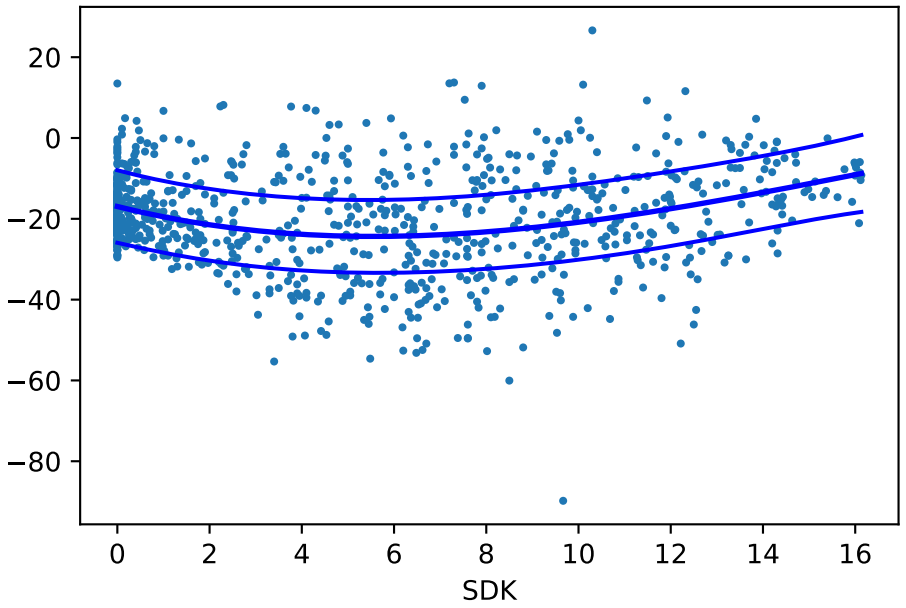
```
smfgam.summary()
```

Das Besondere ist, dass wir die einzelnen Transferfunktionen uns anzeigen lassen können, mit dem gewichteten Residuen des Eingangs. Dadurch bleibt das Modell erklärbar.

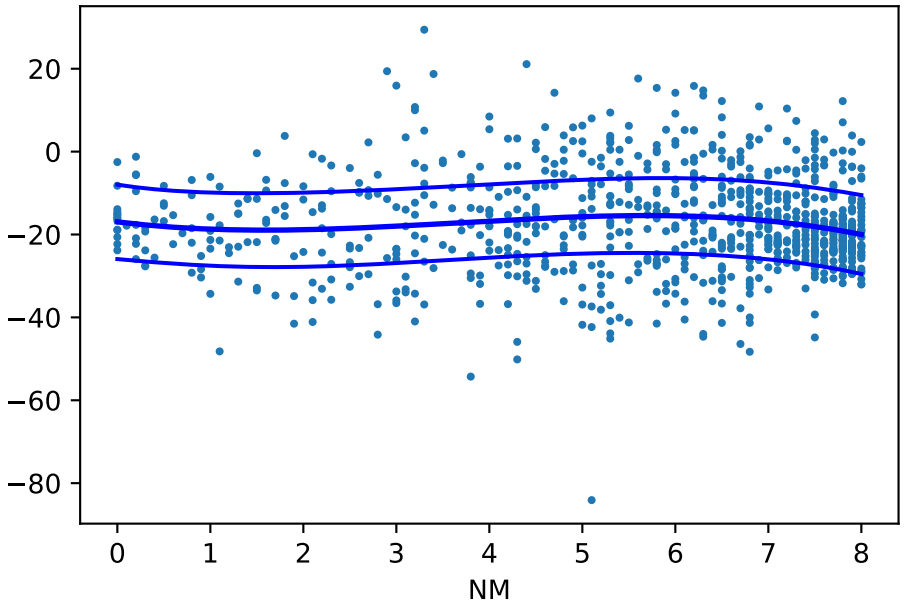
```
fig = smfgam.plot_partial(0, cpr=True)
```



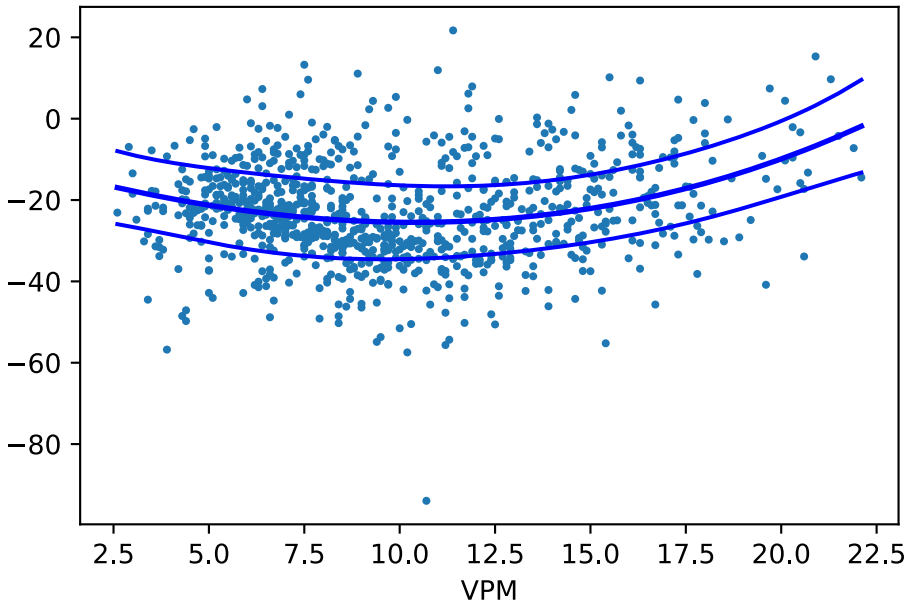

```
fig = smfgam.plot_partial(1, cpr=True)
```




```
fig = smfgam.plot_partial(2, cpr=True)
```




```
fig = smfgam.plot_partial(3, cpr=True)
```



Verleich zum Linearem Modell

Vergleichen wir das mit dem alten Linearen Modell.

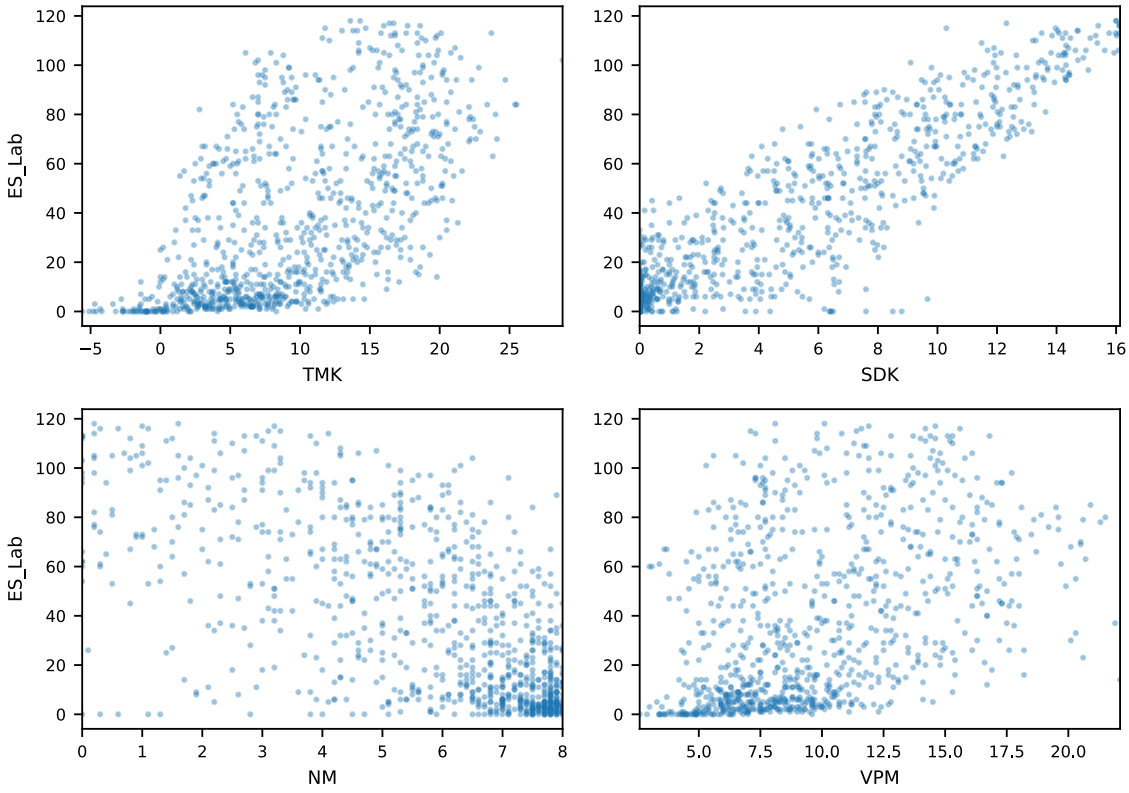
```
import statsmodels.formula.api as smf
smflm = smf.ols("ES_Lab ~ TMK + SDK + NM + VPM + Weekday", data=egywthN).fit()
egywthN["ols"]=smflm.predict()
smflm.summary()
```

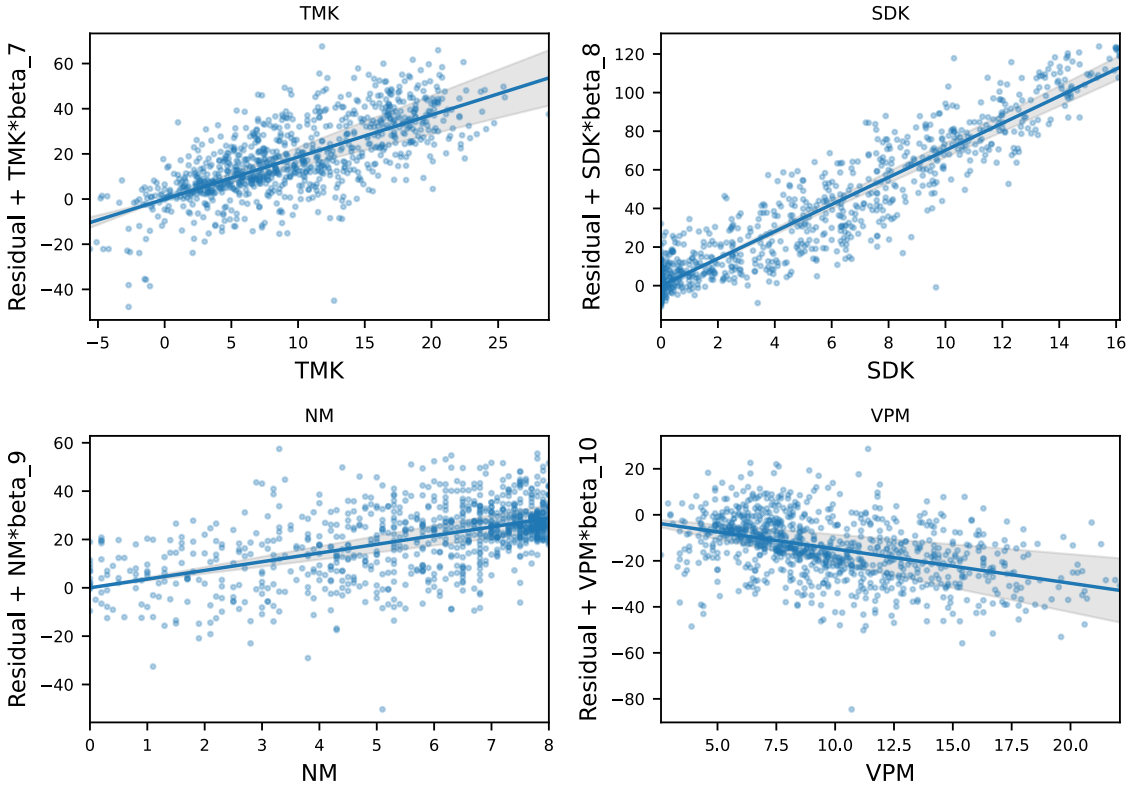
Dep. Variable:	ES_Lab	R-squared:	0.873
Model:	OLS	Adj. R-squared:	0.872
Method:	Least Squares	F-statistic:	621.6
Date:	Sat, 06 Jun 2026	Prob (F-statistic):	0.00
Time:	11:24:20	Log-Likelihood:	−3591.7
No. Observations:	916	AIC:	7205.
Df Residuals:	905	BIC:	7258.
Df Model:	10		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	−20.2008	3.196	−6.320	0.000	−26.474	−13.928
Weekday[T.Monday]	−0.6513	1.519	−0.429	0.668	−3.632	2.329
Weekday[T.Saturday]	−0.3169	1.522	−0.208	0.835	−3.305	2.671
Weekday[T.Sunday]	0.5073	1.524	0.333	0.739	−2.483	3.498
Weekday[T.Thursday]	−0.4597	1.519	−0.303	0.762	−3.441	2.522
Weekday[T.Tuesday]	1.1055	1.518	0.728	0.467	−1.875	4.085
Weekday[T.Wednesday]	−0.3329	1.520	−0.219	0.827	−3.315	2.650
TMK	1.8619	0.218	8.557	0.000	1.435	2.289
SDK	7.0105	0.189	37.032	0.000	6.639	7.382
NM	3.5985	0.349	10.310	0.000	2.914	4.284
VPM	−1.4871	0.322	−4.616	0.000	−2.119	−0.855

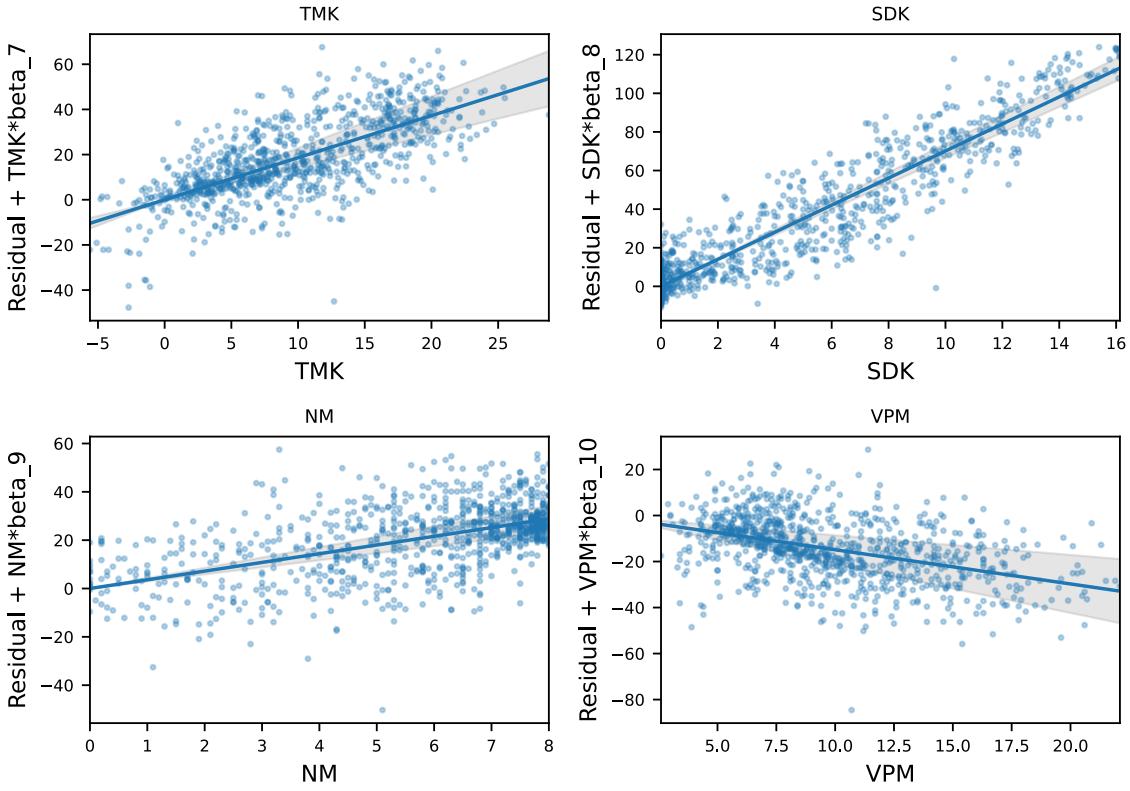
Omnibus:	36.205	Durbin-Watson:	1.115
Prob(Omnibus):	0.000	Jarque-Bera (JB):	77.580
Skew:	−0.219	Prob(JB):	1.42e-17
Kurtosis:	4.357	Cond. No.	147.

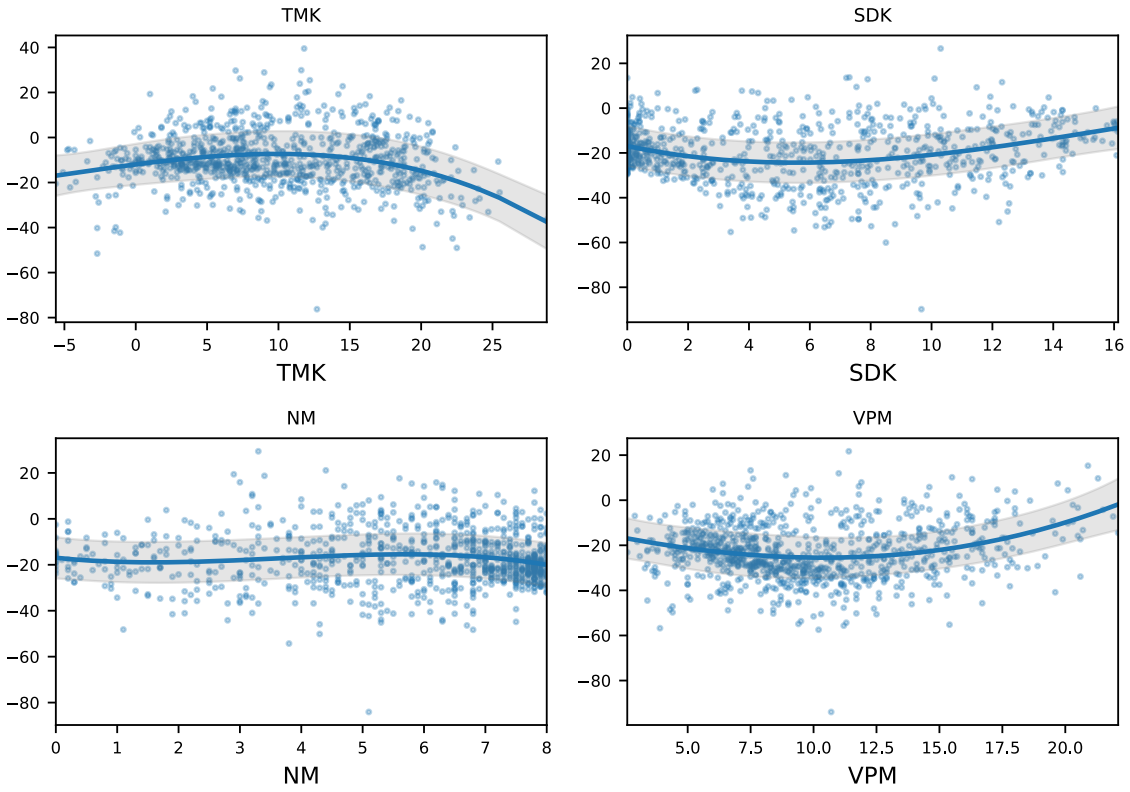
Verleich von Scatterplot und Linearen Modell




TMK
Residual + TMK*beta_7
TMK
SDK
Residual + SDK*beta_8
SDK
NM
Residual + NM*beta_9
NM
VPM
Residual + VPM*beta_10
VPM

Verleich von Linearen Modell und GAM-Modell





Vergleichen wir die Qualität beider Modelle

```
from sklearn.metrics import mean_absolute_error, r2_score
r2_score(egywthN.ES_Lab, egywthN.ols), mean_absolute_error(egywthN.ES_Lab,
egywthN.ols)
```

```
(0.872909238506527, 9.238732131049021)
```

```
r2_score(egywthN.ES_Lab, egywthN.gam), mean_absolute_error(egywthN.ES_Lab,
egywthN.gam)
```

```
(0.8822826884210837, 8.869769884021528)
```

so sehen wir das das GAM-Modell etwas besser ist, aber nicht deutlich.

Das zeigt sich auch im Plot.

```
fig=px.line(egywthN, x="Date", y=["ES_Lab", "ols", "gam"])
fig.show()
```

```
Unable to display output for mime type(s): text/html
```

ARMA - Autoregressive Moving Average

Ein Aspekt den wir bisher in unserem Datensatz ignoriert haben, ist die Tatsache, dass es sich bei der Energie um eine Zeitreihe handelt und wir im Linien-Diagramm bereits beobachtet haben, dass Werte sehr stark vom Vorgängerwert abhängen. Diese Abhängigkeit bezeichnet man als *Autoregression*. Um diese Autoregression zu modellieren, gibt es spezielle Modelle. Ein sehr typisches Modell ist das ARIMA-Modell. Es gleicht einem Linearem Regressionsmodell, das als Eingang bis zu p vergangenen Zielwerte y_{i-k} für $0 < k < p$ betrachtet als auch die q früheren Vorhersagefehler ε_{i-j} integriert.

$$y_i = \beta_0 + \overbrace{\sum_{k=1}^{p} a_k y_{i-k}}^{AR-Modell} + \overbrace{\sum_{j=1}^{q} b_j \varepsilon_{i-j}}^{MA-Modell} + \varepsilon_i.$$

Hierbei ist das AR-Modell ein Modell das mit den Koeffizient a_k den Einfluss des k-ten vergangenen Werts y_{i-k} auf den aktuellen Wert y_i darstellt. Das MA-Modell modelliert die Abweichung vom gleitenden Durchschnitt in Form der vergangenen Fehlerterme ε_{i-k}. Die Ordnungen p und q können bestimmt werden durch Analyse der Autokorrelationsfunktion (ACF) zur Bestimmung der Ordnung q des MA-Teils und der Partielle Autokorrelationsfunktion (PACF) zur Bestimmung der Ordnung p des AR-Teils.

Ein ARMA-Modell kann erstellt werden mit dem ARIMA-Modul von Statsmodels. Wichtig ist hierbei, dass wir eine äquidistante Zeitreihe mit einer festen Abtastrate nutzen. Fehlende Werte sollten dabei vor der Modellierung geeignet behandelt werden, da wir explizit Vorgängerwerte mit betrachten.

```
from statsmodels.tsa.arima.model import ARIMA

arma_mod = ARIMA(egywth.ES_Lab, order=(8, 0, 1), trend="n")
arma_res = arma_mod.fit()
egywth["arma"]=arma_res.predict()
arma_res.summary()
```

Dep. Variable:	ES_Lab	No. Observations:	1098
Model:	ARIMA(8, 0, 1)	Log Likelihood	−4618.230
Date:	Sat, 06 Jun 2026	AIC	9256.460
Time:	11:24:22	BIC	9306.472
Sample:	0	HQIC	9275.381
	• 1098		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.9694	0.087	11.204	0.000	0.800	1.139
ar.L2	−0.0496	0.045	−1.110	0.267	−0.137	0.038
ar.L3	−0.0416	0.039	−1.075	0.282	−0.117	0.034
ar.L4	−0.0114	0.038	−0.305	0.761	−0.085	0.062
ar.L5	0.0115	0.037	0.312	0.755	−0.061	0.084
ar.L6	0.0597	0.037	1.634	0.102	−0.012	0.131
ar.L7	−0.0106	0.037	−0.291	0.771	−0.082	0.061
ar.L8	0.0671	0.035	1.908	0.056	−0.002	0.136
ma.L1	−0.6439	0.088	−7.343	0.000	−0.816	−0.472
sigma2	297.2980	9.665	30.761	0.000	278.356	316.240

Ljung-Box (L1) (Q):	0.00	Jarque-Bera (JB):	310.11
Prob(Q):	0.96	Prob(JB):	0.00
Heteroskedasticity (H):	0.87	Skew:	−0.56
Prob(H) (two-sided):	0.19	Kurtosis:	5.35

Wenn wir das Modell vergleichen mit den vorherigen Modellen, so ist das Vorhersagemodell schlechter und im Diagramm zeigt sich, dass die Vorhersagen geglätteter sind. Allerdings bemötigt das Modell auch keine weitere externen Eingangsvariablen.

```
egywthN=egywth.dropna()
r2_score(egywthN.ES_Lab, egywthN.arma), mean_absolute_error(egywthN.ES_Lab,
egywthN.arma)
```

```
(0.7671305048342131, 11.297525605335332)
```

```
fig=px.line(egywthN, x="Date", y=["ES_Lab", "arma"])
fig.show()
```

```
Unable to display output for mime type(s): text/html
```

Autoregressive Lineare und GAM Modelle

Wir können auch in unsere bisherigen Modelle die Autoregressive Komponente integrieren, indem wir uns eine verschobene Zielvariable definieren. Da wir wissen, dass es auch einen wöchentlichen Saison gibt, fügen wir auch den 7-Tage Lag hinzu.

```
egywth["ES_Lab1"]=egywth.ES_Lab.shift(1)
egywth["ES_Lab7"]=egywth.ES_Lab.shift(7)
egywth.head()
```

	EV_DATE_NF_7AE_ISV_DESLAMSSSDATUM	QN_3	…	TGK	eor	Temperatur-Klasse	Heiz-Kuehl-Tage	QNS_4	QNF_4	Weekday	ES_Lab1	ES_Lab7
0	2020-12-30 00:00:00+00:00 NaN NaN NaN NaN NaN 2020-12-30 00:00:00+00:00 4 20201230 1.0		…	2.2	eor	Cold	Heizgradtag	nicht alle Parameter korrigiert	5	Wednesday	0.000 NaN	NaN
1	2020-12-31 00:00:00+00:00 NaN NaN 125629 1.0 5.0 2020-12-31 00:00:00+00:00 4 20201231 1.0		…	0.9	eor	Cold	Heizgradtag	nicht alle Parameter korrigiert	5	Thursday	0.000 NaN	NaN
2	2021-01-01 00:00:00+00:00 41080 1221290.0 1.0 2021-01-01 00:00:00+00:00 4 20210101 1.0		…	0.3	eor	Cold	Heizgradtag	nicht alle Parameter korrigiert	5	Friday	4.538 584	NaN

	EV_Date_NF_7AE_Lab_ES_Lab_ESLAB_SS_DATUM	QN_3	...TGK	eor	Temperatur-Klasse	Heiz-Kuehl-Tage	QN_4	NF_4	Weekday	ES_Sarima	ES_Lab1	ES_Lab7
3	2021-01-02 00:00:00+00:00 1630124328 4.0 2.0	2021-01-02 00:00:00+00:00 20210102.0 ...	2.0	eor	Cold	Heizgradtag	nicht alle Parameter korrigiert	5	Saturday	2.540	1173	NaN
4	2021-01-03 00:00:00+00:00 1375012220 3.0 2.0	2021-01-03 00:00:00+00:00 20210103.0 ...	0.8	eor	Cold	Heizgradtag	nicht alle Parameter korrigiert	5	Sunday	-2.280	2147	NaN

```python
import statsmodels.formula.api as smf

egywthN=egywth.dropna().copy()
smflmAR = smf.ols("ES_Lab ~ TMK + SDK + NM + VPM + Weekday + ES_Lab1 +
ES_Lab7", data=egywthN).fit()
egywthN["olsAR"]=smflmAR.predict()
smflmAR.summary()
```

Dep. Variable:	ES_Lab	R-squared:	0.924
Model:	OLS	Adj. R-squared:	0.922
Method:	Least Squares	F-statistic:	896.6
Date:	Sat, 06 Jun 2026	Prob (F-statistic):	0.00
Time:	11:24:22	Log-Likelihood:	−3312.8

No. Observations:	904	AIC:	6652.
Df Residuals:	891	BIC:	6714.
Df Model:	12		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	−14.4569	2.513	−5.754	0.000	−19.388	−9.526
Weekday[T.Monday]	−0.7777	1.183	−0.657	0.511	−3.099	1.544
Weekday[T.Saturday]	0.3123	1.188	0.263	0.793	−2.019	2.644
Weekday[T.Sunday]	0.1938	1.188	0.163	0.870	−2.138	2.526
Weekday[T.Thursday]	−0.8354	1.182	−0.707	0.480	−3.155	1.484
Weekday[T.Tuesday]	0.4710	1.184	0.398	0.691	−1.852	2.794
Weekday[T.Wednesday]	−0.6982	1.184	−0.590	0.556	−3.022	1.626
TMK	0.8811	0.174	5.057	0.000	0.539	1.223
SDK	5.2479	0.164	32.041	0.000	4.926	5.569
NM	2.5453	0.276	9.225	0.000	2.004	3.087
VPM	−1.1769	0.252	−4.663	0.000	−1.672	−0.682
ES_Lab1	0.2442	0.017	14.502	0.000	0.211	0.277
ES_Lab7	0.1613	0.016	10.014	0.000	0.130	0.193

Omnibus:	167.812	Durbin-Watson:	1.758
Prob(Omnibus):	0.000	Jarque-Bera (JB):	1687.767
Skew:	−0.519	Prob(JB):	0.00
Kurtosis:	9.613	Cond. No.	617.

```
bs = BSplines(egywthN[['TMK', "SDK", "NM", "VPM", "ES_Lab1", "ES_Lab7"]],
df=[4, 4, 4, 4, 4, 4], degree=[3, 3, 3, 3, 3, 3])

# Fit the GAM model
smfgamAR = GLMGam.from_formula(formula="ES_Lab ~ TMK + SDK + NM + VPM +
Weekday + ES_Lab1 + ES_Lab7", data=egywthN, smoother=bs)
smfgamAR = smfgamAR.fit()
egywthN["gamAR"]=smfgamAR.predict()
smfgamAR.summary()
```

Die resultierenden Modelle sind durch Hinzufügen der autoregressiven Komponente deutlich besser. Es erfordert allerdings eine äquidistante Zeitreihe da sonst Abhängigkeiten inkorrekt aufgelöst werden.

```
r2_score(egywthN.ES_Lab, egywthN.olsAR), mean_absolute_error(egywthN.ES_Lab,
egywthN.olsAR)
```

```
(0.9235182503946626, 6.791042887432227)
```

```
r2_score(egywthN.ES_Lab, egywthN.gamAR), mean_absolute_error(egywthN.ES_Lab,
egywthN.gamAR)
```

```
(0.9299315803850143, 6.507654076564162)
```

Auch zeigt sich, dass sich bei den täglichen Werten das einfache lineare Modell und das komplexere GAM-Modell nur leicht unterscheiden. In diesem Datensatz ist das GAM hier geringfügig besser, in anderen Datensätzen kann sich das aber auch umkehren.

```
fig=px.line(egywthN, x="Date", y=["ES_Lab", "olsAR", "gamAR"])
fig.show()
```

```
Unable to display output for mime type(s): text/html
```

Bibliographie