

Chatverlauf: Social-Media-Nutzung in Deutschland - von der Recherche zur interaktiven Karte

Export eines Chats mit Claude (Anthropic), Oktober 2026. Prompts unverändert, Antworten von Claude inhaltlich vollständig wiedergegeben. Werkzeugaufrufe (Websuche, Codeausführung) sind in eckigen Klammern vermerkt; Quellenverweise aus der Websuche wurden für die Lesbarkeit entfernt.

1. Prompt

hey wie werden soziale medien in deutschland nach alter und region genutzt?

[Claude führt mehrere Websuchen durch und liest die ARD/ZDF-Medienstudie 2025 sowie Berichte zur Ausgabe 2026.]

Antwort

Die aktuellste und belastbarste Quelle dafür ist die ARD/ZDF-Medienstudie. Die Ausgabe 2026 ist gerade erst erschienen, die detaillierten Aufschlüsselungen nach Plattform, Alter und Ost/West stammen aber noch aus der Ausgabe 2025.

Nach Alter: der entscheidende Faktor

Insgesamt hat sich die Nutzung eingependelt. 2026 wird keine Plattform mit Vergleichswert häufiger genutzt als im Vorjahr. Instagram, Snapchat, LinkedIn und Mastodon halten ihr Niveau, Facebook, TikTok und mehrere kleinere Plattformen verlieren Reichweite. Viel größer als die Unterschiede zwischen den Plattformen sind die zwischen den Altersgruppen:

[Balkendiagramm: Social-Media-Nutzung mind. wöchentlich, ARD/ZDF 2026 – 14-29: 91 %, 30-49: 74 %, 50-69: 53 %, 70+: 22 %]

Bei den 14- bis 29-jährigen nutzen 72 Prozent Social Media sogar täglich. Bei den Jungen ist die Nutzung gesättigt, das Wachstum kommt von den Älteren. 2025 stieg die Wochenreichweite bei den 50- bis 69-jährigen um 4 Punkte auf 53 Prozent und bei den über 70-jährigen um 10 Punkte auf 29 Prozent. 2026 liegt der Wert für die über 70-jährigen bei 22 Prozent. Laut den Studienautoren ist damit inzwischen vor allem das älteste Segment der Treiber.

Auch die Plattformwahl hängt stark vom Alter ab. Die Zahlen von 2025, mindestens wöchentlich genutzt:

Plattform	14-29	30-49	50-69	70+
Instagram	77 %	54 %	24 %	5 %
Facebook	26 %	48 %	29 %	13 %
TikTok	50 %	24 %	7 %	2 %
Snapchat	53 %	9 %	1 %	0 %

Quelle: ARD/ZDF-Medienstudie 2025

Daraus ergeben sich typische Nutzerprofile. Instagram-Nutzer sind im Schnitt 37 Jahre alt, bei Facebook sind es 45. TikTok liegt bei 32 Jahren, Snapchat hat mit 25 Jahren die jüngste Nutzerschaft. Außerdem verliert Instagram erstmals bei den Jungen. Die Plattform wächst inzwischen in den mittleren Altersgruppen und noch stärker bei den 50- bis 69-Jährigen. Auch die Zahl der genutzten Plattformen sinkt mit dem Alter: von 4,9 bei den 14- bis 29-Jährigen über 3,3 und 1,4 auf 0,4 bei den über 70-Jährigen.

Nach Region: kleinere Unterschiede, als man denkt

Für Ost und West gibt es eine Sonderauswertung von 2025. Täglich nutzen West- und Ostdeutsche soziale Medien fast gleich häufig. Instagram ist im Westen allerdings um knapp 9 Prozentpunkte gefragter. Facebook ist ausgeglichener, TikTok und X erreichen im Osten etwas höhere Werte. Der Autor warnt aber ausdrücklich davor, daraus abzuleiten, der Osten nutze TikTok und X und der Westen Instagram. Dafür seien die Unterschiede zu gering. Sein Fazit: Alter, Bildung, Einkommen und Interessen erklären die Nutzung besser als der Wohnort. Ein Teil des Ost-West-Unterschieds geht auch darauf zurück, dass im Osten mehr ältere Menschen leben, vor allem über 70-Jährige.

Zu Stadt/Land und einzelnen Bundesländern habe ich keine aktuellen repräsentativen Daten gefunden. Die letzten bekannten Bundesländer-Rankings sind über zehn Jahre alt. Im Social Media-Atlas 2013 lagen Mecklenburg-Vorpommern, Sachsen-Anhalt und Brandenburg hinten, Bremen und Hessen vorn. Für heute ist das kaum noch aussagekräftig.

Ein methodischer Hinweis

Je nach Quelle schwanken die Gesamtzahlen stark. DataReportal zählt Nutzerkonten und kommt so auf 77,1 % der Bevölkerung. Die ARD/ZDF-Studie dagegen befragt Menschen und erhebt, wer mindestens wöchentlich aktiv ist. Wenn du die Zahlen irgendwo zitierst, lohnt es sich, die Methode dazuzuschreiben.

2. Prompt

ok jetzt hätte ich gerne dass du mir einen python code für jupyter notebook generierst der eine repräsentative stichprobe an daten zu diesem thema generiert. dabei soll einmal das alter, einmal die postleitzahl oder das bundesland ´, sowie einmal die z.B. wöchentliche nutzung in Minuten? oder einer anderen Metrik generiert. ich würde die Daten danach gerne auswerten und evetl auf einer deutschlandkarte visualisieren, aber alles zu seiner zeit, erstmal die simulation/generierung der daten in python code, natürlich in einer variable gespeichert, gerne notwendige bibliotheken importieren

[Claude schreibt einen Prototyp, führt ihn in einer Sandbox aus und prüft die Ergebnisse gegen die Studienwerte: 62,7 % Nutzung, 29,8 Min./Tag. Anschließend erstellt es daraus ein Jupyter-Notebook (social_media_stichprobe.ipynb) und testet es.]

Antwort

Das Notebook ist fertig und getestet. Es erzeugt 10.000 Personen im DataFrame `df` mit diesen Spalten: `alter`, `altersgruppe`, `bundesland`, `region` (West/Ost/Berlin), `nutzt_wöchentlich` (True/False) und `minuten_pro_woche`.

So funktioniert die Simulation

1. **Bundesland:** Jede Person bekommt ein Bundesland, gewichtet nach Einwohnerzahl. NRW kommt also etwa 26-mal so oft vor wie Bremen.

2. **Alter:** Das Alter wird abhängig von der Region gezogen. Im Osten gibt es etwas mehr über 70-Jährige, wie in der Ost/West-Auswertung beschrieben.
3. **Nutzung:** Ob jemand Social Media wöchentlich nutzt, entscheidet ein Münzwurf mit der Nutzungsquote seiner Altersgruppe aus der ARD/ZDF-Studie 2025.
4. **Minuten:** Nutzer bekommen ihre Minuten aus einer Lognormalverteilung, also viele Wenignutzer und ein paar Vielnutzer. Der Mittelwert ist so kalibriert, dass die Durchschnittswerte der Studie herauskommen.

Der eingebaute Check zeigt, dass das aufgeht: 62,7 % nutzen Social Media wöchentlich (Studie: 63 %) bei 29,8 Minuten pro Tag (Studie: 30). Auch die Werte je Altersgruppe treffen die Zielwerte.

Ein paar Hinweise

- **Ost/West:** Der Osten hat in den Daten eine leicht niedrigere Nutzungsquote (ca. 58 % gegenüber 63 %). Das liegt allein an der älteren Bevölkerung, nicht an einem eingebauten Regionaleffekt. Das ist ein schöner Befund für deine spätere Auswertung, etwa über Altersstandardisierung. Mit `REGIONAL_FAKTOR` kannst du bewusst echte Regionaleffekte einbauen, falls die Karte mehr Kontrast zeigen soll.
- **Bundesland statt PLZ:** Ich habe das Bundesland genommen, weil sich Postleitzahlen nicht sauber den Bundesländern zuordnen lassen und zufällige PLZ ohne Ortsdatensatz nichts taugen. Für die Karte reicht die Bundeslandebene völlig, dafür gibt es fertige GeoJSON-Dateien. Falls du später auf PLZ-Ebene willst, kann man einen offenen PLZ-Datensatz einbinden und die Stichprobe darauf verteilen.
- **Näherungswerte:** Einwohnerzahlen und Altersanteile sind gerundet. Für Übungszwecke passt das, für eine echte Arbeit würde ich sie durch aktuelle Destatis-Tabellen ersetzen.

3. Prompt

ich will den code direkt in jupyter online reinkopieren

Antwort

Hier alles in einem Block, den du direkt in eine Zelle kopieren kannst:

```
import numpy as np
import pandas as pd

# ===== Parameter =====
SEED = 42          # für Reproduzierbarkeit
N = 10_000         # Stichprobengröße
rng = np.random.default_rng(SEED)

# Einwohner je Bundesland in Mio. (Näherung, Destatis ~2023) -> Ziehungsgewichte
BUNDESLAENDER = {
    "Baden-Württemberg": 11.28, "Bayern": 13.37, "Berlin": 3.76, "Brandenburg": 2.57,
    "Bremen": 0.69, "Hamburg": 1.89, "Hessen": 6.39, "Mecklenburg-Vorpommern": 1.63,
    "Niedersachsen": 8.14, "Nordrhein-Westfalen": 18.14, "Rheinland-Pfalz": 4.16,
    "Saarland": 0.99, "Sachsen": 4.09, "Sachsen-Anhalt": 2.19,
    "Schleswig-Holstein": 2.95, "Thüringen": 2.13,
}

OST = {"Brandenburg", "Mecklenburg-Vorpommern", "Sachsen", "Sachsen-Anhalt", "Thüringen"}

ALTERSGRUPPEN = ["14-29", "30-49", "50-69", "70+"]
ALTER_SPANNE = {"14-29": (14, 29), "30-49": (30, 49), "50-69": (50, 69), "70+": (70, 95)}

# Anteil der Altersgruppen an der Bevölkerung ab 14 J. (Näherung; Osten älter)
ALTER_VERTEILUNG = {
    "West": [0.21, 0.30, 0.30, 0.19],
    "Ost": [0.17, 0.26, 0.32, 0.25],
    "Berlin": [0.22, 0.33, 0.27, 0.18],
}
```

```

}

# ARD/ZDF-Medienstudie 2025
P_NUTZUNG = {"14-29": 0.91, "30-49": 0.76, "50-69": 0.53, "70+": 0.29} # mind. wöchentlich
MIN_PRO_TAG_BEV = {"14-29": 60, "30-49": 41, "50-69": 15, "70+": 7} # 0 Min./Tag inkl. Nichtnutzer

# Umrechnung: mittlere Minuten pro Woche unter den NUTZERN
MIN_PRO_WOCHE_NUTZER = {g: MIN_PRO_TAG_BEV[g] * 7 / P_NUTZUNG[g] for g in ALTERSGRUPPEN}

SIGMA = 0.8 # Streuung der Nutzungsdauer (lognormal)

# Optionaler regionaler Multiplikator auf die Nutzungswahrscheinlichkeit (1.0 = kein Effekt)
REGIONAL_FAKTOR = {bl: 1.0 for bl in BUNDESLAENDER}

# ===== Generierung =====
def region_von(bl):
    if bl == "Berlin":
        return "Berlin"
    return "Ost" if bl in OST else "West"

def generiere_stichprobe(n, rng):
    # Bundesland, gewichtet nach Bevölkerung
    namen = np.array(list(BUNDESLAENDER))
    gewichte = np.array(list(BUNDESLAENDER.values()))
    gewichte = gewichte / gewichte.sum()
    bundesland = rng.choice(namen, size=n, p=gewichte)
    region = np.array([region_von(b) for b in bundesland])

    # Altersgruppe abhängig von der Region
    altersgruppe = np.empty(n, dtype=object)
    for r, p in ALTER_VERTEILUNG.items():
        maske = region == r
        altersgruppe[maske] = rng.choice(ALTERSGRUPPEN, size=maske.sum(), p=p)

    # Konkretes Alter innerhalb der Gruppe
    alter = np.empty(n, dtype=int)
    for g, (lo, hi) in ALTER_SPANNE.items():
        maske = altersgruppe == g
        if g == "70+":
            alter[maske] = np.round(rng.triangular(lo, lo, hi, size=maske.sum())).astype(int)
        else:
            alter[maske] = rng.integers(lo, hi + 1, size=maske.sum())

    # Nutzt Social Media mindestens wöchentlich?
    p = np.array([P_NUTZUNG[g] * REGIONAL_FAKTOR[b] for g, b in zip(altersgruppe, bundesland)])
    nutzt = rng.random(n) < np.clip(p, 0, 1)

    # Minuten pro Woche (Nichtnutzer = 0)
    mittel = np.array([MIN_PRO_WOCHE_NUTZER[g] for g in altersgruppe])
    mu = np.log(mittel) - SIGMA**2 / 2
    minuten = np.where(nutzt, rng.lognormal(mu, SIGMA), 0.0)
    minuten = np.clip(np.round(minuten), 0, 7 * 16 * 60).astype(int) # max. 16 h/Tag

    return pd.DataFrame({
        "id": np.arange(1, n + 1),
        "alter": alter,
        "altersgruppe": pd.Categorical(altersgruppe, categories=ALTERSGRUPPEN, ordered=True),
        "bundesland": bundesland,
        "region": region,
        "nutzt_wöchentlich": nutzt,
        "minuten_pro_woche": minuten,
    })

df = generiere_stichprobe(N, rng)

# ===== Plausibilitätscheck =====
check = (
    df.groupby("altersgruppe", observed=True)
    .agg(n=("id", "size"),
         nutzung_ist=("nutzt_wöchentlich", "mean"),
         min_pro_tag_ist=("minuten_pro_woche", lambda x: x.mean() / 7))
)
check["nutzung_soll"] = check.index.map(P_NUTZUNG)
check["min_pro_tag_soll"] = check.index.map(MIN_PRO_TAG_BEV)

```

```
print(f"Gesamt: {df['nutzt_woentlich'].mean():.1%} nutzen wöchentlich (Studie: 63 %)")
print(f"Gesamt: {df['minuten_pro_woche'].mean() / 7:.1f} Min./Tag (Studie: 30 Min.)\n")
print(check.round(2), "\n")
df.head(10)
```

Nach dem Ausführen liegen die Daten in `df`. Am Ende siehst du den Abgleich mit den Studienwerten und die ersten zehn Zeilen.

4. Prompt

die module pandas wurden nicht gefunden kann es sein dass es anders importiert werden muss

Antwort

Der Import ist richtig so, pandas ist in deiner Umgebung nur nicht installiert. Das kommt bei Online-Varianten von Jupyter öfter vor. Führe in einer eigenen Zelle **vor** dem Code Folgendes aus:

```
%pip install pandas numpy
```

Danach die Zelle mit dem Code einfach nochmal starten. Falls der Fehler dann immer noch kommt, einmal den Kernel neu starten (Menü **Kernel → Restart Kernel**) und dann den Code erneut ausführen.

Falls du JupyterLite nutzt (z. B. über jupyter.org/try-jupyter, läuft komplett im Browser): Dort funktioniert `%pip` je nach Version nicht. Nimm stattdessen:

```
import piplite
await piplite.install(["pandas", "numpy"])
```

Wenn du mir sagst, welches Jupyter du genau nutzt (Google Colab, JupyterLite, Binder, das Jupyter der Uni ...), kann ich dir die passende Variante sagen. In Google Colab sind pandas und numpy übrigens schon vorinstalliert, da läuft der Code ohne Zusatzschritt.

5. Prompt

```
Cannot install 'pandas' from PyPI because it is a binary built package that is not compatible with WASM environments. To resolve this issue, you can: 1) Try to install it from emscripten-forge instead: "!mamba install pandas" 2) If that doesn't work, it's probably that the package was not made WASM-compatible on emscripten-forge. You can either request or contribute a new recipe for that package in https://github.com/emscripten-forge/recipes
```

Antwort

Die Fehlermeldung verrät, dass du JupyterLite mit dem **xeus-python-Kernel** nutzt. Der kann pandas nicht von PyPI laden, sondern nur aus einer eigenen Paketquelle (emscripten-forge). Mach einfach, was die Meldung vorschlägt, in einer eigenen Zelle:

```
!mamba install pandas numpy
```

Danach den Code nochmal ausführen. Bei Bedarf vorher den Kernel neu starten.

Falls das nicht klappt, hast du zwei Alternativen:

- **Anderen Kernel wählen:** Beim Anlegen des Notebooks oder oben rechts über den Kernel-Namen auf **Python (Pydide)** wechseln. Dort sind pandas und numpy schon dabei, und `import pandas` funktioniert direkt.
- **Google Colab** (colab.research.google.com): Das ist ein vollwertiges Python auf einem Server. Alles ist vorinstalliert, und für die spätere Kartenvisualisierung ist es deutlich unkomplizierter, weil Bibliotheken wie geopandas oder plotly in WASM-Umgebungen oft Probleme machen.

Mit Blick auf die geplante Deutschlandkarte würde ich dir ehrlich gesagt Colab empfehlen. Dann musst du nicht später nochmal umziehen.

6. Prompt

okay und jetzt würde ich das ganze gerne auf einer deutschlandkarte visualisieren, welche art der visualisierung macht sinn denkst du?

Antwort

Bei deinen Daten macht eine **Choroplethenkarte** am meisten Sinn, also eine Flächenkarte, auf der jedes Bundesland nach einem Kennwert eingefärbt wird. Du hast die Daten auf Bundeslandebene, und genau dafür ist diese Kartenart gemacht. Ein paar Entscheidungen bestimmen aber, ob die Karte etwas Sinnvolles zeigt.

Raten statt Anzahlen zeigen. Die Zahl der Nutzer pro Bundesland wäre eine schlechte Wahl. Die Karte würde dann nur zeigen, dass in NRW viele Menschen leben. Sinnvoll sind normierte Werte wie der Anteil wöchentlicher Nutzer in Prozent oder die durchschnittlichen Minuten pro Woche. Bei den Minuten musst du entscheiden, ob der Schnitt über alle Personen oder nur über die Nutzer gehen soll. Die beiden Varianten beantworten unterschiedliche Fragen.

Abweichung vom Bundesschnitt einfärben. Die Unterschiede zwischen den Ländern sind klein, nur wenige Prozentpunkte. Eine normale Farbskala von hell nach dunkel lässt dann alles gleich aussehen. Besser ist eine divergierende Skala, zum Beispiel blau unter und rot über dem Bundesschnitt, mit Weiß in der Mitte. So sieht man sofort, wo es nach oben oder unten geht.

Rohwerte und altersbereinigte Werte nebeneinander. Das ist der spannendste Punkt. Links zeigst du die Rohwerte: Da wird der Osten etwas heller erscheinen. Rechts zeigst du dieselbe Kennzahl altersstandardisiert, also so, als hätten alle Länder dieselbe Altersstruktur. Weil wir in der Simulation keinen echten Regionaleffekt eingebaut haben, wird die rechte Karte nahezu einfarbig sein. Genau das ist die Aussage: Der scheinbare Ost-West-Unterschied ist ein Alterseffekt. Methodisch ist das eine schöne Geschichte und passt auch zu dem, was die ARD/ZDF-Autoren über die echten Daten schreiben.

Stichprobengröße im Blick behalten. Bei 10.000 Personen landen in Bremen nur etwa 90, im Saarland etwa 130. Deren Werte schwanken deshalb zufällig stärker als die von NRW oder Bayern. Für eine Übung ist das okay. Du solltest es aber im Hinterkopf haben, bevor du ein auffälliges Bremen interpretierst, und die Fallzahl zum Beispiel im Tooltip anzeigen. Alternativ erhöhst du N auf 50.000 oder 100.000, das kostet bei der Simulation praktisch nichts.

Als Ergänzung könntest du noch **Small Multiples** machen: vier kleine Karten, eine pro Altersgruppe. Die zeigen eindrucksvoll, dass die Farbe sich zwischen den Karten viel stärker ändert als innerhalb einer Karte. Das Alter schlägt also den Wohnort.

Technisch gibt es zwei gute Wege. **Plotly** liefert eine interaktive Karte mit Hover-Infos (Wert und Fallzahl pro Land), läuft problemlos in Colab und ist gut zum Erkunden. **Geopandas mit Matplotlib** erzeugt statische Karten, die sich besser für eine schriftliche Arbeit oder Folien eignen. Beide brauchen eine GeoJSON-Datei mit den Bundesländergrenzen, die frei verfügbar ist und direkt im Code geladen werden kann.

Willst du die Karte eher interaktiv zum Erkunden oder statisch, zum Beispiel für ein Dokument?

7. Prompt

*interaktiv hört sich gut an, entscheide du über die anderen Fragen, es ist eine Übung um zu zeigen wie KI schnelle aussagekräftige Ergebnisse produzieren kann (natürlich nur aussagekräftig in Bezug auf die Datenqualität, hier soll nichts schöner werden als es ist)
aber das ganze bitte als Python-Code den ich wieder in das Notebook kopieren kann*

[Claude prüft die GeoJSON-Quelle, schreibt den Kartencode und testet ihn in der Sandbox mit den simulierten Daten.]

Antwort

Hier der Code für die nächste Zelle. Er setzt voraus, dass die Zelle mit der Datengenerierung vorher gelaufen ist, also `df` existiert. In Colab sind alle Bibliotheken schon installiert.

```
import json
import urllib.request
import numpy as np
import pandas as pd
import plotly.graph_objects as go
import plotly.express as px
from plotly.subplots import make_subplots

# ===== 1. Bundesländergrenzen laden =====
GEOJSON_URL = ("https://raw.githubusercontent.com/isellsoap/deutschlandGeoJSON/"
               "main/2_bundeslaender/4_niedrig.geo.json")
with urllib.request.urlopen(GEOJSON_URL) as r:
    geo = json.load(r)

# ===== 2. Kennzahlen je Bundesland =====
# Standardbevölkerung für die Altersbereinigung: Altersverteilung der Gesamtstichprobe
std_gewichte = df["altersgruppe"].value_counts(normalize=True).sort_index()

# Werte je Bundesland x Altersgruppe (fehlende Zellen -> Bundeswert der Altersgruppe)
def raten_nach_alter(spalte):
    tab = df.pivot_table(index="bundesland", columns="altersgruppe", values=spalte,
                        aggfunc="mean", observed=False)
    bund = df.groupby("altersgruppe", observed=False)[spalte].mean()
    return tab.fillna(bund)

nutz_alter = raten_nach_alter("nutzt_wöchentlich")
min_alter = raten_nach_alter("minuten_pro_woche")

laender = (
    df.groupby("bundesland")
      .agg(n=("id", "size"),
          anteil_roh=("nutzt_wöchentlich", "mean"),
          minuten_roh=("minuten_pro_woche", "mean"),
          anteil_70plus=("altersgruppe", lambda x: (x == "70+").mean()))
)
laender["anteil_std"] = nutz_alter.mul(std_gewichte, axis=1).sum(axis=1)
laender["minuten_std"] = min_alter.mul(std_gewichte, axis=1).sum(axis=1)

# 95%-Konfidenzintervall (Normalapproximation) für den Rohanteil
laender["ki95"] = 1.96 * np.sqrt(laender["anteil_roh"] * (1 - laender["anteil_roh"]) / laender["n"])

bund_anteil = df["nutzt_wöchentlich"].mean()
```

```

bund_minuten = df["minuten_pro_woche"].mean()

laender = laender.reset_index()
display(laender.sort_values("anteil_roh").round(3))

# ===== 3. Vergleichskarte: roh vs. altersbereinigt =====
def vergleichskarte(spalte_roh, spalte_std, bundeswert, faktor, einheit, titel):
    """Färbt die Abweichung vom Bundesschnitt; beide Karten teilen dieselbe Farbskala."""
    abw_roh = (laender[spalte_roh] - bundeswert) * faktor
    abw_std = (laender[spalte_std] - bundeswert) * faktor
    grenze = max(abw_roh.abs().max(), abw_std.abs().max())

    fig = make_subplots(
        rows=1, cols=2,
        specs=[[{"type": "choropleth"}, {"type": "choropleth"}]],
        subplot_titles=("Rohwerte", "Altersbereinigt (gleiche Altersstruktur überall)"),
        horizontal_spacing=0.02,
    )
    for col, (abw, spalte) in enumerate([(abw_roh, spalte_roh), (abw_std, spalte_std)], start=1):
        hover = [
            f"<b>{bl}</b><br>Wert: {w * faktor:.1f} {einheit}<br>"
            f"Abweichung: {a:+.1f} {einheit}<br>Fallzahl n = {n}<br>"
            f"Anteil 70+: {a70:.0%}"
            + (f"<br>95%-KI Rohwert: ±{ki * 100:.1f} Pp." if col == 1 and faktor == 100 else "")
            for bl, w, a, n, a70, ki in zip(laender["bundesland"], laender[spalte], abw,
                                          laender["n"], laender["anteil_70plus"], laender["ki95"])
        ]
        fig.add_trace(go.Choropleth(
            geojson=geo, featureidkey="properties.name",
            locations=laender["bundesland"], z=abw,
            coloraxis="coloraxis", text=hover, hoverinfo="text",
            marker_line_color="white", marker_line_width=0.8,
        ), row=1, col=col)

    fig.update_geos(fitbounds="locations", visible=False)
    fig.update_layout(
        title=dict(text=titel, x=0.5),
        coloraxis=dict(colorscale="RdBu_r", cmin=-grenze, cmid=0,
                      colorbar=dict(title=f"Abw. vom<br>Bundesschnitt<br>({einheit})")),
        height=600, margin=dict(l=10, r=10, t=80, b=10),
    )
    fig.show()

vergleichskarte("anteil_roh", "anteil_std", bund_anteil, 100, "%",
                f"Anteil wöchentlicher Social-Media-Nutzer – Bundesschnitt {bund_anteil:.1%}"
                f"<br><sup>Simulierte Daten, n = {len(df):}</sup>")

vergleichskarte("minuten_roh", "minuten_std", bund_minuten, 1, "Min.",
                f"Ø Social-Media-Minuten pro Woche (alle Personen) – Bundesschnitt {bund_minuten:.0f} Min."
                f"<br><sup>Simulierte Daten, n = {len(df):}</sup>")

# ===== 4. Small Multiples: eine Karte pro Altersgruppe =====
alter_land = (
    df.groupby(["altersgruppe", "bundesland"], observed=True)
      .agg(anteil=("nutzt_woentlich", "mean"), n=("id", "size"))
      .reset_index()
)
alter_land["anteil_pct"] = (alter_land["anteil"] * 100).round(1)

fig = px.choropleth(
    alter_land, geojson=geo, featureidkey="properties.name",
    locations="bundesland", color="anteil_pct", facet_col="altersgruppe",
    color_continuous_scale="Viridis", range_color=(0, 100),
    hover_name="bundesland", hover_data={"anteil_pct": ":.1f", "n": True,
                                          "bundesland": False, "altersgruppe": False},
    labels={"anteil_pct": "Anteil (%)", "n": "Fallzahl"},
    title="Anteil wöchentlicher Nutzer nach Altersgruppe – das Alter erklärt mehr als der Wohnort",
)
fig.update_geos(fitbounds="locations", visible=False)
fig.for_each_annotation(lambda a: a.update(text=a.text.replace("altersgruppe=", "")))
fig.update_layout(height=450, margin=dict(l=10, r=10, t=80, b=10))
fig.show()

```

Was du bekommst

Zuerst kommt eine Tabelle mit allen Kennzahlen je Bundesland. Danach folgen zwei Vergleichskarten, eine für den Nutzeranteil und eine für die Minuten. Jede zeigt links die Rohwerte und rechts die

altersbereinigten Werte. Beide Hälften verwenden dieselbe Farbskala, damit der Vergleich fair ist. Rot heißt über, Blau heißt unter dem Bundesschnitt. Im Tooltip stehen Wert, Fallzahl, Anteil der über 70-Jährigen und beim Rohanteil das 95%-Konfidenzintervall. Zum Schluss kommen vier kleine Karten, eine pro Altersgruppe, mit einer festen Skala von 0 bis 100 %.

Was die Karten zeigen, ohne Schönfärberei

In der linken Karte sind die ostdeutschen Länder blau. Das liegt allein an ihrer älteren Bevölkerung. In der rechten Karte verschwindet dieses Muster weitgehend.

Was rechts an Unterschieden übrig bleibt, ist **reiner Zufall**. In der Simulation gibt es keinen eingebauten Regionaleffekt. Sichtbar wird das an den Fallzahlen: Bremen hat nur etwa 80 Personen und damit ein Konfidenzintervall von rund ± 10 Prozentpunkten. Jeder Farbunterschied in dieser Größenordnung ist Rauschen.

Am deutlichsten wird es bei den Small Multiples. Zwischen den vier Karten springt die Farbe von Dunkellila zu Gelb, innerhalb einer Karte bleibt sie fast gleich.

Mein Tipp für die Demo: Setz in der ersten Zelle `N = 100_000` und lass beides nochmal laufen. Dann schrumpft das Rauschen deutlich, und die rechte Karte wird fast einfarbig. Der Kontrast zwischen den beiden Läufen zeigt sehr anschaulich, warum Fallzahlen wichtig sind, bevor man Muster auf einer Karte interpretiert.

8. Prompt

```
ModuleNotFoundError                                Traceback (most recent call last)
Cell In[5], line 5
      3 import numpy as np
      4 import pandas as pd
----> 5 import plotly.graph_objects as go
      6 import plotly.express as px
      7 from plotly.subplots import make_subplots

ModuleNotFoundError: No module named 'plotly'
```

Antwort

plotly ist bei dir genau wie pandas vorher noch nicht installiert. Weil plotly reines Python ist, sollte es in deiner Browser-Umgebung direkt klappen. Führe in einer eigenen Zelle aus:

```
%pip install plotly
```

Falls das nicht geht, nimm wie bei pandas:

```
!mamba install plotly
```

Ich muss dich aber vor zwei weiteren Stolpersteinen warnen, die in JupyterLite wahrscheinlich als Nächstes kommen.

1. **Download der Kartengrenzen:** Browser-Pythons wie dein JupyterLite können meist keine normalen Netzwerkverbindungen aufbauen. Die Zeile mit `urllib.request.urlopen` wird dann scheitern. Der Workaround: Lade die Datei manuell herunter über diesen Link: https://raw.githubusercontent.com/isellsoap/deutschlandGeoJSON/main/2_bundeslaender/4_niedrig.geo.json (Rechtsklick → Speichern unter). Zieh sie dann per Drag & Drop in den Dateibrowser links in JupyterLite. Ersetze im Code den Block unter „1. Bundesländergrenzen laden“ durch:

```
python with open("4_niedrig.geo.json", encoding="utf-8") as f: geo = json.load(f)
```

1. **Anzeige der Karten:** Interaktive plotly-Grafiken brauchen in JupyterLab eine passende Erweiterung, die in JupyterLite nicht immer dabei ist. Die Zelle läuft dann ohne Fehler, zeigt aber nichts an.

Ehrlich gesagt kämpfst du hier gegen die Umgebung, nicht gegen den Code. Für deinen Zweck, also zu zeigen, wie schnell man mit KI zu Ergebnissen kommt, wäre **Google Colab** deutlich besser. Dort laufen beide Zellen unverändert per Copy & Paste, ohne Installationen und ohne Workarounds: colab.research.google.com öffnen, neues Notebook anlegen, Datengenerierung in die erste Zelle, Kartencode in die zweite, fertig. Das dauert keine zwei Minuten und ist für die Demo vermutlich auch überzeugender als eine Abfolge von Installationsfehlern.

9. Prompt

```
ValueError                                Traceback (most recent call last)
Cell In[6], line 86
----> 86 vergleichskarte("anteil_roh", "anteil_std", bund_anteil, 100, "%", ...)
...
Cell In[6], line 84, in vergleichskarte(...)
----> 84 fig.show()
...
File /lib/python3.13/site-packages/plotly/io/_renderers.py:415, in show(fig, renderer, validate, **kwargs)
--> 415     raise ValueError(
      416         "Mime type rendering requires nbformat>=4.2.0 but it is not installed"
      417     )

ValueError: Mime type rendering requires nbformat>=4.2.0 but it is not installed
```

(Traceback gekürzt)

Antwort

Gute Nachricht: Die Daten und die Kartengrenzen wurden erfolgreich geladen, und die Karte wurde auch gebaut. Es scheitert nur noch an der Anzeige. plotly braucht dafür das Paket `nbformat`, das in deiner Umgebung fehlt.

Lösung 1: nbformat installieren

```
%pip install nbformat
```

Danach den **Kernel neu starten** (Kernel → Restart Kernel). plotly prüft beim Import, ob nbformat vorhanden ist, deshalb reicht es nicht, nur die Zelle neu auszuführen. Anschließend die Zellen von oben wieder durchlaufen lassen, also Datengenerierung und Karte.

Lösung 2, falls das nicht klappt: plotly ohne nbformat anzeigen

Diese Variante umgeht den eingebauten Mechanismus und bindet die Karte als HTML ein. Füg diesen Block **ganz oben** in die Kartenzelle ein, direkt unter die Imports:

```
from IPython.display import HTML, display

def _zeige(fig, *args, **kwargs):
    display(HTML(fig.to_html(include_plotlyjs="cdn", full_html=False)))

go.Figure.show = _zeige
```

Danach ruft jedes `fig.show()` im restlichen Code automatisch diese Anzeige auf. Am Code musst du sonst nichts ändern. Die plotly-Bibliothek lädt dann dein Browser selbst aus dem Netz, was auch in JupyterLite funktioniert.

Sollte auch das nichts anzeigen, gibt es einen letzten sicheren Weg: Ersetze im Block `fig.show()` durch `fig.write_html("karte.html")`. Die Datei erscheint dann links im Dateibrowser und lässt sich per Doppelklick als interaktive Karte öffnen.

10. Prompt

```
mambajs 0.21.4

Process pip requirements ...

Requirement attrs>=22.2.0 already satisfied.
Cannot install 'rpds-py' from PyPI because it is a binary built package that is not compatible with WASM
environments. [...]
Requirement referencing>=0.28.4 already satisfied.
Requirement traitlets already satisfied.
Successfully installed nbformat-5.11.1 fastjsonschema-2.22.2 jsonschema-4.26.0 attrs-26.1.0 jsonschema-
specifications-2025.9.1 referencing-0.37.0 jupyter-core-5.9.1 platformdirs-4.12.3
```

Antwort

nbformat ist jetzt installiert. Nur `rpds-py` fehlt noch, eine Abhängigkeit im Hintergrund. Ob das stört, hängt davon ab, ob nbformat diese Abhängigkeit beim Laden wirklich braucht. Am einfachsten installierst du sie mit dem Weg, den die Meldung vorschlägt, gleich nach:

```
!mamba install rpds-py
```

Danach **Kernel neu starten** und alle Zellen von oben nochmal ausführen.

Falls das wieder scheitert oder die Karte immer noch nicht erscheint, würde ich nicht weiter Pakete hinterherjagen, sondern direkt **Lösung 2** aus meiner letzten Nachricht nehmen. Die braucht nbformat gar nicht. Zur Sicherheit hier nochmal der Block, den du in der Kartenzelle direkt unter die Imports setzt:

```
from IPython.display import HTML, display

def _zeige(fig, *args, **kwargs):
    display(HTML(fig.to_html(include_plotlyjs="cdn", full_html=False)))

go.Figure.show = _zeige
```

Damit umgehst du die ganze Paketkette, und die Karte wird direkt als HTML im Notebook angezeigt.

11. Prompt

```
mambajs 0.21.4

Specs: xeus-python, numpy, matplotlib, pillow, ipywidgets>=8.1.6, ipyleaflet, scipy, pandas, plotly, rpds-
py
Channels: emscripten-forge-4x, conda-forge

Solving environment...

Cannot solve the request because of: No candidates were found for rpds-py *.
```

Antwort

Die Fehlermeldung ist eindeutig: `rpds-py` gibt es für deine Browser-Umgebung schlicht nicht. Über diesen Weg kommen wir also nicht weiter. Das ist aber kein Problem, denn **Lösung 2** braucht `nbformat` und `rpds-py` gar nicht.

So gehst du vor:

1. Kernel neu starten (Kernel → Restart Kernel), damit die halb installierten Pakete nicht dazwischenfunken.
2. Die Zelle mit der Datengenerierung ausführen.
3. In der Kartenzelle direkt unter die Imports diesen Block einfügen und die Zelle ausführen:

```
from IPython.display import HTML, display

def _zeige(fig, *args, **kwargs):
    display(HTML(fig.to_html(include_plotlyjs="cdn", full_html=False)))

go.Figure.show = _zeige
```

Der Block ersetzt die Anzeige-Funktion von plotly durch eine eigene, die die Karte als HTML ins Notebook schreibt. Das gilt auch für die Small Multiples am Ende, denn auch die laufen über `fig.show()`. Sonst musst du nichts am Code ändern.

Wenn danach weiterhin nichts erscheint, ist der letzte sichere Weg, die Karten als Dateien zu speichern. Ersetze dafür diesen Block durch:

```
def _zeige(fig, *args, **kwargs):
    global _karten_nr
    _karten_nr = globals().get("_karten_nr", 0) + 1
    fig.write_html(f"karte_{_karten_nr}.html", include_plotlyjs="cdn")
    print(f"Gespeichert: karte_{_karten_nr}.html")

go.Figure.show = _zeige
```

Dann tauchen links im Dateibrowser `karte_1.html` bis `karte_3.html` auf. Ein Doppelklick öffnet sie als interaktive Karten.

12. Prompt

also nformat und rpds-py löschen?

Antwort

Nein, löschen musst du nichts.

- **rpds-py** wurde nie installiert, die Installation ist ja gescheitert. Da gibt es nichts zu entfernen.
- **nbformat** kann bleiben. Es stört nicht, weil der neue Block plotlys eigene Anzeige komplett umgeht. Ob `nbformat` da ist oder nicht, spielt dann keine Rolle mehr.

Einfach Kernel neu starten, beide Zellen ausführen (mit dem Block unter den Imports), und die Karten sollten erscheinen.

Ein Hinweis noch: In JupyterLite gehen nachinstallierte Pakete beim Neustart des Kernels manchmal verloren. Falls also wieder „No module named pandas“ oder „plotly“ kommt, einfach nochmal `!mamba install pandas plotly` ausführen und dann weitermachen.

13. Prompt

funktioniert, gefällt mir aber die Karte ist leicht gestaucht, und ich hätte sie gerne doppelt so groß

Antwort

Beides lässt sich mit wenigen Änderungen beheben.

Die Stauchung kommt von der Standardprojektion in plotly („equirectangular“). Die zieht Länder in mittleren Breiten wie Deutschland in die Breite. Eine **Mercator-Projektion** zeigt die Form korrekt.

Für die doppelte Größe reicht es nicht, nur die Höhe zu erhöhen. Die Karte passt sich sonst der Breite des Notebooks an, und bei zwei Karten nebeneinander bekommt jede nur die halbe Breite. Mehr Höhe brächte dann nur weißen Rand. Deshalb setze ich Breite und Höhe fest.

Änderung 1: In der Funktion `vergleichskarte` die letzten Zeilen (ab `fig.update_geos`) ersetzen durch:

```
fig.update_geos(fitbounds="locations", visible=False, projection_type="mercator")
fig.update_layout(
    title=dict(text=titel, x=0.5),
    coloraxis=dict(colorscale="RdBu_r", cmin=-grenze, cmax=grenze, cmid=0,
                  colorbar=dict(title=f"Abw. vom<br>Bundesschnitt<br>({einheit})")),
    width=1600, height=1150, margin=dict(l=10, r=10, t=100, b=10),
)
fig.show()
```

Änderung 2: Bei den Small Multiples ganz am Ende die beiden Zeilen

```
fig.update_geos(fitbounds="locations", visible=False)
```

und

```
fig.update_layout(height=450, margin=dict(l=10, r=10, t=80, b=10))
```

ersetzen durch:

```
fig.update_geos(fitbounds="locations", visible=False, projection_type="mercator")
```

und

```
fig.update_layout(width=1600, height=650, margin=dict(l=10, r=10, t=80, b=10))
```

Je nach Bildschirm kann die Karte jetzt breiter sein als der Notebook-Bereich, dann erscheint ein horizontaler Scrollbalken. Falls dich das stört, kannst du `width` und `height` im gleichen Verhältnis verkleinern, zum Beispiel auf 1300 und 935. Das Verhältnis sorgt dafür, dass die Karte die Fläche ausfüllt statt Rand zu lassen.

Quellen der Ausgangsrecherche

- ARD/ZDF-Medienstudie 2025 und 2026 (veröffentlicht in *Media Perspektiven*), u. a. Beiträge zur Social-Media-Nutzung und zur Ost/West-Sonderauswertung
- DataReportal (Nutzerkonten-basierte Zahlen für Deutschland)

- Social Media-Atlas 2013 (Faktenkontor) für historische Bundesländer-Rankings

Hinweis: Alle Daten im Notebook sind simuliert und nur an Kennzahlen der genannten Studien kalibriert. Sie eignen sich zum Üben von Auswertung und Visualisierung, nicht für echte Aussagen über einzelne Bundesländer.