



Patrick Lürwer

# R4SEO: AUTOMATISIERTE REPORTS MIT R GENERIEREN (TEIL 2)

## DER AUTOR



**Patrick Lürwer** ist Senior Analyst bei get.traction GmbH. Dort ist er für die Datenerfassung, -aufbereitung und -analyse zuständig. Sein tägliches Handwerkszeug sind R, Python und KNIME.

Im ersten Teil dieser Artikelserie in der letzten Ausgabe hat Ihnen Patrick Lürwer gezeigt, wie Sie mit R die Google-Analytics- und Search-Console- sowie SISTRIX-API abfragen können. Im vorliegenden Teil werden Sie lernen, die Daten so aufzubereiten, dass sie zur Darstellung im Report geeignet sind. Dazu werden Sie fehlende Datumspunkte in den DataFrames ergänzen. Sie werden den gleitenden Mittelwert für verschieden Metriken berechnen, um starke Schwankungen der Tagesdaten zu glätten. Außerdem erfahren Sie, wie Sie DataFrames transponieren, um sie von einem weiten in ein langes Format zu bringen.

## TIPP

Um Ihnen das Abtippen des Codes zu ersparen, können Sie sich auch dieses Mal das Skript zu diesem Artikel unter folgendem Link auf GitHub herunterladen: [https://github.com/gettracti-ongmbh/r4seo\\_ws](https://github.com/gettracti-ongmbh/r4seo_ws). Ich empfehle Ihnen dennoch, den Code selbst zu schreiben, um ein Gefühl für Syntax und Funktionen zu bekommen.

Im letzten Artikel haben Sie Daten aus drei unterschiedlichen Quellen abgefragt, sodass Ihnen diese nun in tabellarischer Form (DataFrames) in R vorliegen. Aber wie das so häufig ist, haben die APIs die Daten nicht so geliefert, wie sie für die Report-Erstellung benötigt werden. Beispielsweise liefert die Google-Search-Console-API (GSC) nur Datumspunkte zurück, an denen auch tatsächlich Impressions bzw. Clicks stattgefunden haben. Es fehlen folglich die Tage ohne Impressions. Außerdem liegen die Daten auf Tagesbasis vor und sind demzufolge unter Umständen sehr volatil. Sie wollen sie daher glätten, um Trends besser erkennen zu können. Es ist somit unabdingbar, dass Sie die Daten „aufräumen“ und in eine entsprechende Form bringen, um sie für die Visualisierung verwenden zu können. Im Englischen spricht man in dem Fall von *tidying* – daher auch die Benennung des Packages *tidyverse*, das Sie verwenden werden. Viele der darin enthaltenen Funktionen dienen allein dazu, Daten zunächst in die richtige Form zu bringen, um mit ihnen arbeiten zu können.

## Neues Skript erstellen sowie Daten und Packages laden

Bevor Sie mit dem Aufräumen beginnen, generieren Sie in Ihrem Projekt, das Sie im Zuge des letzten Artikels angelegt haben, ein neues R-Skript mit der Benennung `data_tidying.R` und kopieren Sie den gesamten Inhalt des Skripts `api_calls.R` hinein. Führen Sie

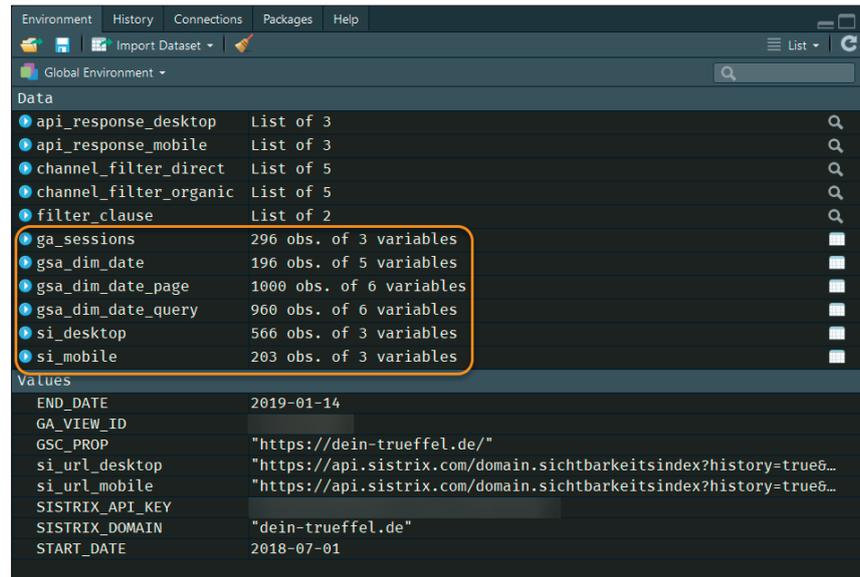


Abb.1: Fremde Marken in den Bullets – hier womöglich nur beschreibend

das Skript aus, damit die aktuellen Daten von den APIs abgefragt werden und zur Weiterverarbeitung in DataFrames in Ihrem Environment vorliegen (Abb. 1).

Im Anschluss installieren und laden Sie wieder einige Packages, die für die nachfolgenden Datentransformationen benötigt werden (Abb. 2). *tidyverse* (<https://www.tidyverse.org/>) ist eine sehr mächtige Sammlung von Packages für die Datenmanipulation. In diesem Package ist auch *ggplot* enthalten, welches Sie schon aus dem Artikel von Tobias Aubele kennen und das zum Plotten von Daten dient. Darüber hinaus erweitert es die R-Syntax um den Pipe-Operator (`%>%`), den Sie im vorherigen Artikel kurz verwendet haben. *zoo* und *padr* dienen dem Arbeiten mit Zeitreihen.

## Google-Search-Console-Daten vervollständigen

Wie bereits angesprochen, gibt die GSC-API nur Datumspunkte zurück, an denen auch tatsächlich mindestens eine Impression stattgefunden hat. Werden die Daten nur mit der Dimension `date` abgefragt, wie Sie es für die Variable `gsa_dim_date` getan haben, liegen meistens für alle Tage Daten vor, denn hier wird die gesamte Website betrachtet. Wird die Dimension `date` jedoch um die

query erweitert, ist dies nicht mehr der Fall, denn nicht jedes Keyword wird an jedem Datum des Berichtszeitraums eine Impression generiert haben. Dies sehen Sie in Abbildung 3, in der die Anzahl der rankenden Keywords gezählt wurde. Um die Tabelle selbst zu reproduzieren, können Sie den Befehl, den Sie im Screenshot sehen, in der Console ausführen. Dadurch ist er kein Bestandteil des Skripts, sondern dient nur dazu, die Daten *on-the-fly* anzuzeigen. 2018-07-02 hat nur ein Keyword Impressions generiert, 2018-07-05 zwei. An den Tagen dazwischen gab es keine Rankings, sodass keine Datumspunkte von der API zurückgegeben wurden.

Wie geschrieben werden fehlende Datumspunkte bei Abfragen, die allein aus der Dimension `date` bestehen, recht selten vorkommen, da dies bedeuten würde, dass keine einzige Seite an

## TIPP

Eine sehr anschauliche Einleitung in die Syntax und gängige Funktionen des *tidyverse* finden Sie unter folgendem Link. Das dort beschriebene Package *dplyr* ist ein Bestandteil des *tidyverse* und dient der Datenmanipulation, wie Sie sie im Folgenden selbst durchführen werden: <https://speakerdeck.com/omaymas/data-manipulation-with-dplyr-first-steps>

diesem Tag eine Impression generiert hat. Sie müssen sich aber stets bewusst darüber sein, dass dieses Verhalten auftreten kann. Und dann möchten Sie dies auch bestimmt in Ihren Diagrammen sehen. Denn wenn an einem Tag keine Seite ein Ranking aufweisen kann, liegt definitiv etwas im Argen. Solche fehlenden Werte müssen Sie also explizit machen, denn standardmäßig werden beim Plotting Werte für fehlende Datumsunkte interpoliert. In Abbildung 4 sehen Sie einen exemplarischen Verlauf von Impressions je Tag im Monat, wobei die Anzahl der Impressions der Einfachheit halber konstant ist. Für die Datumsunkte 5, 8, 12 / 13, 24 und 30 liegen keine Werte vor. Die blaue Linie (1) wurde ohne ergänzte Datumsunkte geplottet. Hier sehen Sie, dass die Linie zwischen den bestehenden Datumsunkten einfach weitergezogen wurde. Der fehlende Datumsunkt 5 wird durch Interpolation (vereinfacht: dem Mittelwert der beiden bekannten umliegenden Werte) berechnet. Bei der roten Linie (2) wurden die fehlenden Datumsunkte ergänzt, allerdings keine Ersatzwerte eingetragen. Dadurch bricht die Linie an diesen Punkten ab. Die fehlenden Werte liegen nun explizit in den Daten vor. Das Zielbild ist aber die grüne Linie (3). Bei dieser wurden die Datumsunkte ergänzt und die fehlenden Werte durch 0 aufgefüllt. Hier fällt somit bei der Betrachtung des Graphen sofort ins Auge, dass die Impressions auf 0 abgestürzt sind.

Das Auffüllen ist im Übrigen nicht nur beim Plotting wichtig, sondern auch bei der Aggregation der Daten auf einer höheren Zeitebene. Soll beispielsweise der Durchschnitt der Impressions auf Monatsbasis berechnet werden, verfälschen fehlende Werte das Ergebnis. Angenommen, jeder Datumsunkt im Diagramm zeigt 10 Impressions an, wäre der Mittelwert für die blaue Linie 10 (10 Impressions \* 25 Tage /

```

123
124 # Daten aufräumen -----
125
126 # Packages installieren; nur einmal nötig
127 # install.packages(c("tidyverse",
128 #                   "zoo",
129 #                   "padr"))
130
131 library(tidyverse)
132 library(zoo)
133 library(padr)
    
```

Abb. 2: Installieren bzw. Aktualisieren und Laden der benötigten Packages

	date	n
1	2018-07-02	1
2	2018-07-05	2
3	2018-07-06	1
4	2018-07-07	3

Abb. 3: Zählen der Queries je Tag; fehlende Datumswerte an Tagen ohne Impressions

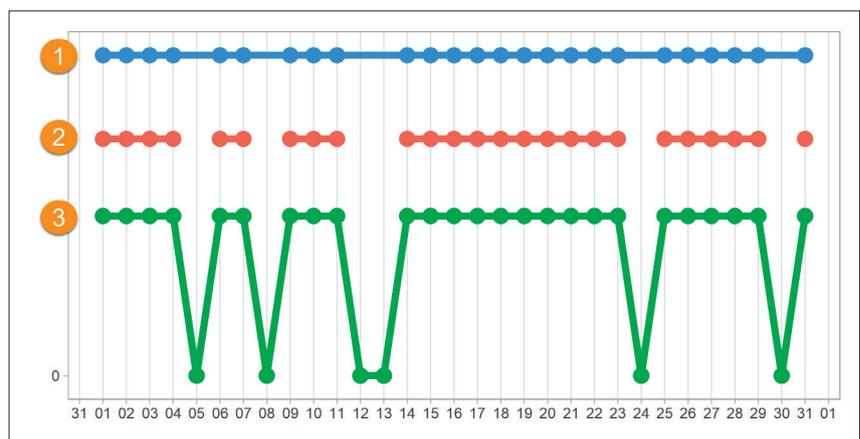


Abb. 4: Fehlende Datumsunkte und Werte auffüllen, um Verfälschungen des Graphen zu vermeiden

25 Tage), für die grüne Linie hingegen 8,06 ( (10 Impressions \* 15 Tage + 0 Impressions \* 6 Tage) / 31 Tage). Dies müssen Sie daher immer berücksichtigen, wenn Sie mit den Rohdaten aus der GSC arbeiten.

Zurück zum Skript: Nachdem Sie die Packages geladen haben, fügen Sie

den Code-Abschnitt, wie in Abbildung 5 zu sehen, ein. Der DataFrame gsa\_dim\_date, der die Performance-Daten auf Tagesbasis enthält, wird an die Funktion pad() aus dem Package padr gegeben. Die Funktion erkennt automatisch die Datumsspalte und füllt die fehlenden Datumsunkte auf. Über die

```

136
137 # GSC -----
138
139 # Impressions / Clicks
140
141 # Fehlende Datumspunkte einfügen, Impressions und
142 # Clicks mit 0 auffüllen
143 gsa_dim_date <- gsa_dim_date %>%
144   pad(start_val = START_DATE,
145       end_val = END_DATE) %>%
146   replace_na(list(clicks = 0,
147                  impressions = 0))
148

```

Abb. 5: Fehlende Datumspunkte ergänzen und Metriken mit 0 auffüllen

1	2	3
1	1	1
2	2	2
3	3	3
4	4	4
	5	5
	6	6
	7	7

Abb. 6: Exemplarischer DataFrame mit aufgefüllten Werten

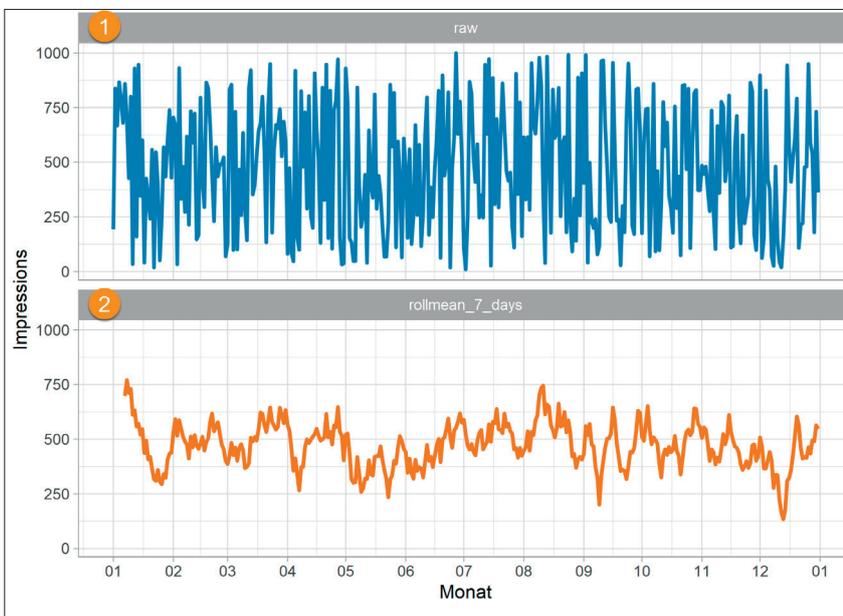


Abb. 7: Plotting der Rohdaten versus gleitenden Mittelwert

Argumente `start_val` und `end_val` definieren Sie die untere und obere Grenze des aufzufüllenden Berichtszeitraums. Die hierfür verwendeten Variablen `START_DATE` und `END_DATE` sind erneut jene, die Sie bereits bei den API-Abfragen genutzt haben. Nach diesem Schritt liegen alle Datumspunkte vor, allerdings enthalten die Spalten `clicks` und `impressions` noch keine Werte für diese Punkte. Diese geben Sie durch die anschließende Funktion `replace_na()` aus dem Package `tidyverse` an. Der bearbeitete DataFrame wird wieder in

die Variable `gsa_dim_date` zurückgeschrieben.

In Abbildung 6 sehen Sie die einzelnen Schritte noch einmal im Detail anhand eines exemplarischen DataFrames mit Daten für eine Woche. (1) sind die Rohdaten, bei denen drei Tage fehlen. Bei (2) wurden mittels `pad()` die fehlenden Datumspunkte ergänzt, die Werte für die Metriken fehlen aber noch (NA; not available). In (3) wurden diese Metriken via `replace_na()` mit dem Wert 0 aufgefüllt.

## Gleitenden Mittelwert der Impressions und Clicks berechnen

Für die Performance-Daten liegt Ihnen nun der gesamte Berichtszeitraum im DataFrame `gsa_dim_date` auf Tagesbasis vor. Bei der Visualisierung sorgt dies jedoch dafür, dass der Graph stark „zappeln“ wird, wenn bspw. die Werte für die Impressions starken Schwankungen unterliegen. In Abbildung 7 (1) sehen Sie dies für exemplarische Daten eines Jahres. Hier einen Trend zu erkennen, ist somit sehr schwer, da er im Rauschen des Graphen untergehen kann. Ein gängiges Mittel, um den Graphen zu glätten, ist daher, den gleitenden Mittelwert für ein Zeitfenster von sieben Tagen zu berechnen (2).

### HINWEIS

Wie Sie die Daten visualisieren, erfahren Sie in einem späteren Teil. Hier dienen die Plots zunächst nur der Veranschaulichung, warum die Daten wie beschrieben transformiert werden.

Sie müssen sich dazu vorstellen, dass von einer gegebenen Zeitreihe fortschreitend sieben Tage betrachtet werden und für diese der Mittelwert gebildet wird. In Abbildung 8 sehen Sie dies beispielhaft für ein Zeitfenster von drei Tagen. Zunächst wird der Mittelwert für die Tage 1 bis 3 berechnet, dann für die Tage 2 bis 4 usw.

Den gleitenden Mittelwert berechnen Sie mit dem Code-Abschnitt, den Sie in Abbildung 9 sehen. Der DataFrame `gsa_dim_date` wird an die Funktion `mutate()` gegeben, die eine bestehende Spalte bearbeiten oder eine neue Spalte hinzufügen kann. In diesem Fall werden die Spalten `rollmean_clicks` und `rollmean_impressions` hinzugefügt. Die Werte der beiden Spalten werden mit der Funktion `rollmean()` aus dem Package `zoo` berechnet. Das erste Argu-

ment x nimmt die Spalte auf, auf deren Basis der gleitende Mittelwert berechnet werden soll. Das Argument k definiert das Zeitfenster – hier sieben Tage. Mit fill kann der Wert angegeben werden, der in die Zellen geschrieben werden soll, für die aufgrund der Reduktion der Daten keine Werte mehr vorliegen. Das sehen Sie ebenfalls in Abbildung 8. Für die Datumspunkte 1 und 2 kann kein Mittelwert berechnet werden, da das Zeitfenster ansonsten links über den Rand des Datumsbereichs hinauslaufen würde. Entsprechend werden diese beiden Datumspunkte mit NAs aufgefüllt. Das Argument align steuert dieses Verhalten. Es zeigt an, ob der berechnete Mittelwert am rechten Rand des Berichtszeitraums endet und demnach links mit NAs aufgefüllt werden soll oder ob am Datumspunkt 1 mit dem Mittelwert begonnen wird und somit die NAs am rechten Rand eingetragen werden.

Für die exemplarischen Daten der Abbildung 8 sähe die finale Tabelle wie in Abbildung 10 dargestellt aus.

### Umwandlung des DataFrames von einem weiten in ein langes Format

Damit folgt der letzte Schritt zum Aufräumen der Performance-Daten. Ihr DataFrame sieht zum jetzigen Zeitpunkt wie in Abbildung 11 aus. Man spricht in diesem Fall von einer weiten Tabelle, denn je Datumspunkt liegen die einzelnen Metriken (clicks, impressions, ctr etc.) als Spalten vor. Für die Visualisierung mit ggplot müssen die Daten jedoch in einem langen Format vorliegen. Konkret bedeutet dies, dass die Benennungen der bisherigen Spalten die Werte einer neuen Spalte bilden. Die Tabelle wird quasi gedreht, wobei die Datumsspalte als Ankerpunkt dient. Das umgekehrte Verfahren, die Überführung einer langen in eine weite Tabelle, kennen Sie wahrscheinlich aus Excel, wenn Sie dort eine Pivot-Tabelle erstellen.

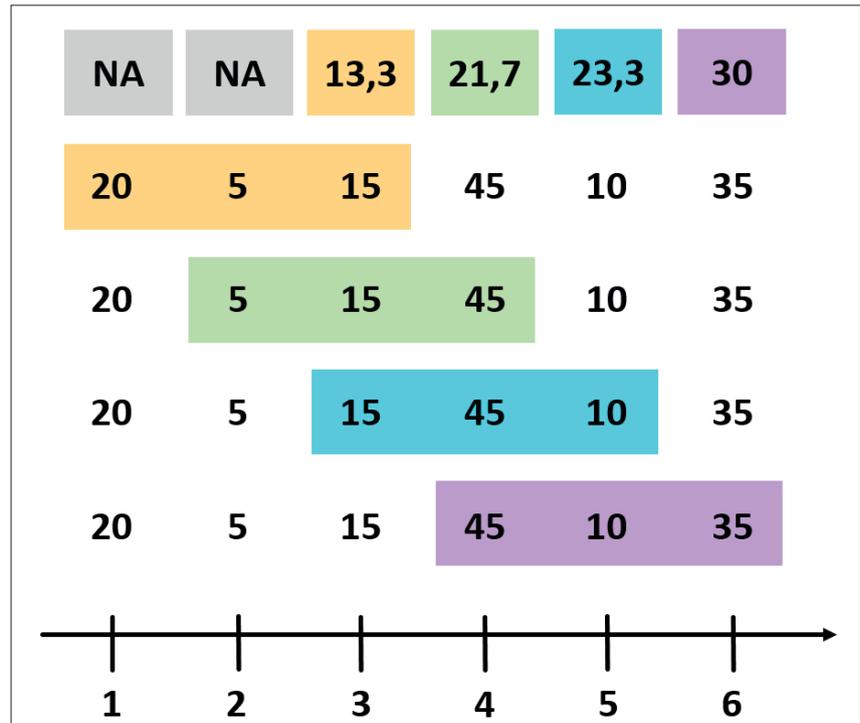


Abb. 8: Exemplarische Berechnung des gleitenden Mittelwerts für ein Zeitfenster von drei Tagen

```

149
150 # Gleitenden Mittelwert berechnen
151 gsa_dim_date <- gsa_dim_date %>%
152   mutate(rollmean_clicks = rollmean(x = clicks,
153                                     k = 7,
154                                     fill = NA,
155                                     align = "right"),
156          rollmean_impressions = rollmean(x = impressions,
157                                           k = 7,
158                                           fill = NA,
159                                           align = "right"))
160
    
```

Abb. 9: Berechnung des gleitenden Mittelwerts für Clicks und Impressions

Das war jetzt sehr abstrakt: Eine schematische Darstellung mit nur vier Metrik-Spalten sehen Sie daher in Abbildung 12.

Ihren gsa\_dim\_date DataFrame überführen Sie mit dem Code aus Abbildung 13. select() dient dazu, nur die Spalten date, rollmean\_clicks und rollmean\_impressions beizubehalten. Die restlichen Spalten clicks, impressions, ctr und position löschen Sie somit aus dem DataFrame, da sie in der späteren Visualisierung nicht genutzt werden. Die Funktion gather() transponiert dann den DataFrame. Mit dem Argument key geben Sie den Namen der neuen Spalte an, in die die bisherigen Spaltenbenennungen als Werte überführt werden. Das Gleiche machen Sie mit dem Argument value für die neue

date	impressions	rollmean
1	20	NA
2	5	NA
3	15	13.33333
4	45	21.66667
5	10	23.33333

Abb. 10: Beispieltabelle mit berechnetem gleitendem Mittelwert

Spalte, die die Werte der jeweiligen Metrik-Spalten aufnimmt. Damit haben Sie das Aufräumen der GSC-Daten erledigt. Ihr DataFrame sieht nun wie in Abbildung 14 aus. Die DataFrames mit den Queries (gsa\_dim\_date\_query) resp. den URLs (gsa\_dim\_date\_page) werden Sie im nächsten Teil dieser Reihe aufbereiten.

	date	clicks	impressions	ctr	position	rollmean_clicks	rollmean_impressions
1	2018-07-01	2	66	0.03030303	18.136364	NA	NA
2	2018-07-02	2	41	0.04878049	33.390244	NA	NA
3	2018-07-03	1	57	0.01754386	35.877193	NA	NA
4	2018-07-04	0	73	0.00000000	36.054795	NA	NA
5	2018-07-05	2	98	0.02040816	39.357143	NA	NA
6	2018-07-06	4	76	0.05263158	44.157895	NA	NA
7	2018-07-07	6	75	0.08000000	40.266667	2.428571	69.42857
8	2018-07-08	2	53	0.03773585	27.188679	2.428571	67.57143
9	2018-07-09	5	54	0.09259259	29.203704	2.857143	69.42857
10	2018-07-10	2	77	0.02597403	32.116883	3.000000	72.28571
11	2018-07-11	2	72	0.02777778	23.625000	3.285714	72.14286

Abb. 11: DataFrame gsc\_dim\_date mit den ergänzten Spalten für die gleitenden Mittelwerte

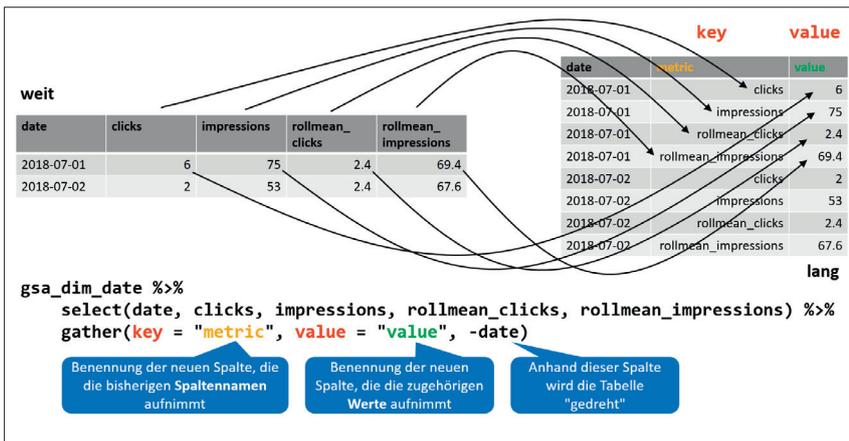


Abb. 12: Umwandlung einer weiten in eine lange Tabelle mittels gather()

```

159
160 # Tabelle transponieren
161 gsa_dim_date <- gsa_dim_date %>%
162   select(date, rollmean_clicks, rollmean_impressions) %>%
163   gather(key = "metric",
164         value = "value",
165         -date)
    
```

Abb. 13: Überführen des weiten DataFrames gsa\_dim\_date in ein langes Format

	date	metric	value
1	2018-07-01	rollmean_clicks	NA
2	2018-07-02	rollmean_clicks	NA
3	2018-07-03	rollmean_clicks	NA
4	2018-07-04	rollmean_clicks	NA
5	2018-07-05	rollmean_clicks	NA
6	2018-07-06	rollmean_clicks	NA
7	2018-07-07	rollmean_clicks	2.428571
8	2018-07-08	rollmean_clicks	2.428571
9	2018-07-09	rollmean_clicks	2.857143

Abb. 14: Die ersten Zeilen des langen DataFrames gsa\_dim\_date

## Google-Analytics-Daten vervollständigen

Analog zu den Performance-Daten aus der GSC müssen Sie noch die Google-Analytics-Daten (GA) aufräumen. Auch bei diesen beginnen Sie damit, fehlende Datumswerte zu ergänzen. Insbesondere bei den GA-Daten ist dies eine reine Vorsichtsmaßnahme, da Sie diese für die Channels Organic Search und Direct abgefragt haben. Dass an einem Tag keine organischen oder direkten Sessions auf Ihrer Website erfolgen, ist höchst unwahrscheinlich – kann aber nichtsdestotrotz bei sehr kleinen bzw. neuen Websites vorkommen. Um sich auf den später generierten Report verlassen zu können, empfiehlt es sich daher, immer für eine „saubere“ Datenbasis zu sorgen.

Fügen Sie den Code aus Abbildung 15 in Ihr Skript ein. Der Unterschied zum Code für die GCS-Daten aus Abbildung 5 besteht hier in dem zusätzlichen Argument group der Funktion pad(). Sie erinnern sich: Diese ergänzt fehlende Datumspunkte. Die Besonderheit des GA-DataFrames besteht darin, dass dieser bereits im langen Format von der API zurückkommt (Abb. 16). Er besteht nur aus den drei Spalten date, channelGrouping und sessions. Die Spalte channelGrouping ist vergleichbar mit Spalte metric im DataFrame gsa\_dim\_date. Sie qualifiziert die nebeneinanderstehenden Werte in der Spalte sessions (vergleichbar mit value im GSC-DataFrame) nach den Groups Direct und Organic Search. Das Argument group ist daher nötig, um die fehlenden Datumspunkte individuell für die beiden Ausprägungen Direct und Organic Search zu ermitteln. In der Abbildung 16 sehen Sie bspw., dass am 2018-07-01 zwar eine Direct-Session erfolgt ist, für Organic Search jedoch kein Wert vorliegt. Für 2018-07-02 verhält es sich genau andersherum. Für Ersteres muss somit der Organic-Search-, für Letzteres der Direct-Wert ergänzt werden (Abb. 17).

**HINWEIS**

Haben Sie Fragen? Dann stellen Sie sie gerne in der Facebook-Gruppe <https://www.facebook.com/groups/ompyr/>. So können alle von den Antworten profitieren. Seien Sie versichert, es gibt keine „dummen“ Fragen, denn aller Anfang ist schwer.

**Gleitenden Mittelwert der GA-Sessions berechnen**

Im Anschluss errechnen Sie den gleitenden Mittelwert für die Sessions (Abb. 18). Auch hier müssen Sie dies separat für die beiden Channels machen. Daher erfolgt in der zweiten Zeile ein `group_by()` der Spalte `channelGrouping`, sodass die nachfolgenden Funktion `rollmean()` separat auf diese beiden Gruppen angewendet wird.

**Fazit**

Gratulation! Wenn Sie es bis hierher geschafft haben, sind Sie schon ein gutes Stück in die Gefilde der Datentransformation vorgedrungen. Sie haben das Wehklagen eines jeden Daten-Analysten nachvollziehen gelernt, dass Analysen zu 80 % aus Datenbereinigung und nur zu 20 % aus der eigentlich spannenden Analysetätigkeit bestehen.

Darüber hinaus haben Sie erfahren, wie Sie fehlende Datumswerte in Zeitreihen vervollständigen, um Verfälschungen bei Datenaggregationen zu vermeiden. Außerdem haben Sie den gleitenden Mittelwert verschiedener Metriken berechnet, um Graphen zu glätten und somit Trends besser erkennbar zu machen. Last but not least haben Sie ein mächtiges Werkzeug an die Hand bekommen, um Tabellen zu transponieren und sie so in ein handhabbares langes Format zu bringen.

Lesen Sie im dritten Teil in der nächsten Ausgabe, wie Sie Ihr SEO-Reporting mit R automatisieren können. So lassen sich z. B. mit wenig Aufwand die Top-SEO-Seiten nach Klicks und die Top-Suchanfragen aus der Google Search Console ermitteln und für Berichte in speziellen DataFrames zusammenfassen. ¶

```
169
170 # Google Analytics -----
171
172 # Fehlende Datumspunkte einfügen und Sessions mit 0 auffüllen
173 ga_sessions <- ga_sessions %>%
174   pad(start_val = START_DATE,
175       end_val = END_DATE,
176       group = "channelGrouping") %>%
177   replace_na(list(sessions = 0))
178
```

Abb. 15: Fehlende Datumspunkte ergänzen und Sessions mit 0 auffüllen

	date	channelGrouping	sessions
1	2018-07-01	Direct	1
2	2018-07-02	Organic Search	2
3	2018-07-03	Organic Search	1
4	2018-07-04	Direct	1
5	2018-07-05	Direct	1
6	2018-07-05	Organic Search	1

Abb. 16: Langer DataFrame `ga_sessions` mit teilweise fehlenden Kombinationen aus `date` und `channelGrouping`

	date	channelGrouping	sessions
1	2018-07-01	Direct	1
2	2018-07-01	Organic Search	0
3	2018-07-02	Direct	0
4	2018-07-02	Organic Search	2
5	2018-07-03	Direct	0
6	2018-07-03	Organic Search	1
7	2018-07-04	Direct	1
8	2018-07-04	Organic Search	0
9	2018-07-05	Direct	1
10	2018-07-05	Organic Search	1

Abb. 17: DataFrame `ga_sessions` mit ergänzten Kombinationen aus `date` und `channelGrouping`

```
177
178 # Gleitenden Mittelwert berechnen
179 ga_sessions <- ga_sessions %>%
180   group_by(channelGrouping) %>%
181   mutate(rollmean_sessions = rollmean(x = sessions,
182                                       k = 7,
183                                       fill = NA,
184                                       align = "right"))
```

Abb. 17: DataFrame `ga_sessions` mit ergänzten Kombinationen aus `date` und `channelGrouping`