Foto: lunewind / gettyimages.de

Patrick Lürwer

R4SEO: AUTOMATISIERTE REPORTS MIT R GENERIEREN (TEIL 1)

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.

In der letzten Ausgabe der Website Boosting gab Tobias Aubele einen sehr guten Einblick in die Potenziale von R, um auf einfache und reproduzierbare Weise Daten über die Google-Analytics-API abzufragen. Diese neue Artikelserie von Patrick Lürwer baut darauf auf und zeigt, wie mittels R Reports erstellt werden können, die sich zu einem bestimmten Zeitpunkt automatisch aktualisieren und per Mail verschicken. Im ersten Teil geht es nun zunächst um das Abfragen verschiedener APIs für die Datenbeschaffung. In den folgenden Teilen werden die Daten aggregiert, visualisiert und in einem Report zusammengefasst.

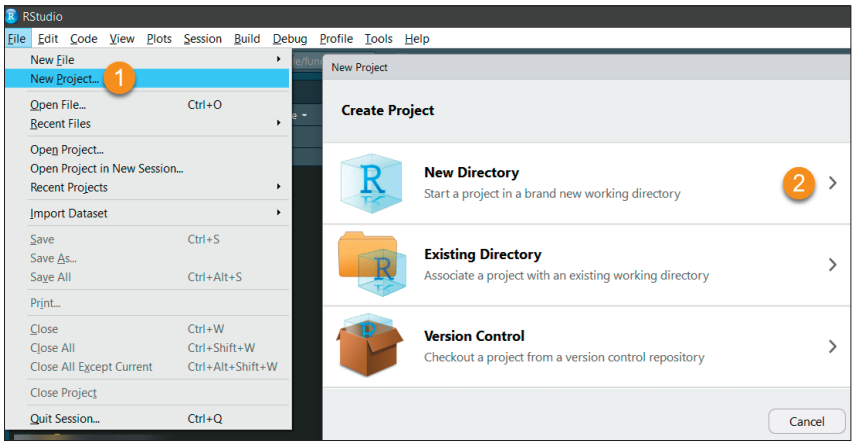

Abb. 1: Anlegen eines neuen Projekts

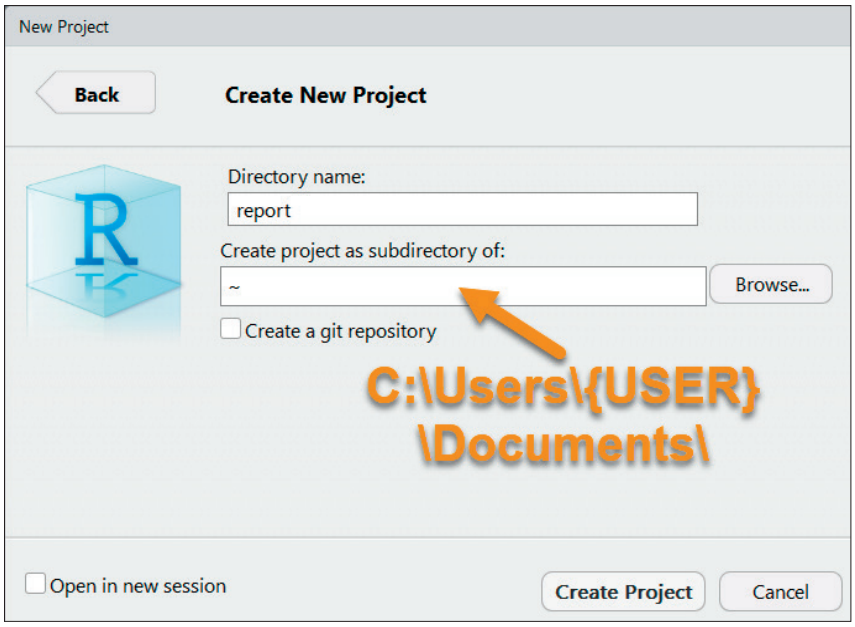

Abb. 2: Speicherort und Benennung des Projekts

Das Google Data Studio (GDS) bietet eine einfache Möglichkeit, Daten aus dem Google-Kosmos – bspw. aus Google Analytics (GA) und der Google Search Console (GSC) – abzufragen, zu visualisieren und als Reports aufzubereiten. Problematisch wird es allerdings, wenn andere Datenquellen angebunden werden sollen. Denkbar sind hier APIs anderer Anbieter wie Sistrix. In solchen Fällen muss der „Umweg" über Google Spreadsheets und Google Apps Scripts gegangen werden, um die Abfragen selbst zu programmieren. Dies ist durchaus ein valides Vorgehen, das jedoch schnell an seine Grenzen stoßen kann. Denn meist sind Reports nur der erste Schritt. Sobald in diesen Auffälligkeiten erkennbar werden, schließt eine Analyse an, die eine umfassende Beschäftigung mit den zugrunde liegenden Daten erfordert. Derartige Analysen verlangen zumeist komplexe Datentransformationen und -aggregationen, für die Dashboard-Softwares, wie das GSA, mit ihren begrenzten Filtern- und Manipulationsoptionen naturgemäß nicht ausgelegt sind. Ganz abgesehen davon, dass der direkte Zugriff auf die Daten zur Weiterverarbeitung häufig gar nicht gegeben ist. Von großem Vorteil ist es daher, wenn Report- und Analysesoftware in einer Anwendung vereint sind. Diese Anforderungen kann R insofern erfüllen, als die Programmiersprache durch Packages so erweitert werden kann, dass mit ihr sowohl sehr individualisierte Reports erstellt als auch umfangreiche Analysen durchgeführt werden können.

Da Datenanalysen eine sehr komplexe Thematik sind, soll in diesem Artikel jedoch zunächst das Potenzial von R zur Report-Generierung vorgestellt werden. Denn im Grunde sind beide Aufgaben bis zu einem gewissen Grade sehr ähnlich. Zunächst werden Daten von verschiedenen Quellen abgefragt, dann aufbereitet und schließlich visualisiert. Bei Analysen ist dieser Prozess jedoch iterativ, ohne dass im Vorhinein klar ist, wie das Ergebnis resp. die Erkenntnis genau aussieht. Bei der Generierung von Reports hingegen ist der Ablauf des Datenprozesses vorgegeben. Sie sind daher kleine, standardisierte Analysen, die an einem bestimmten Punkt der Visualisierung aufhören. Das Ergebnisdokument ist somit immer gleich und jederzeit reproduzierbar. Daher eignen sie sich sehr gut, um exemplarisch die Möglichkeiten von R aufzuzeigen, und sind dabei aber gleichzeitig so vereinheitlicht, um durch den Leser selbst nachgebaut zu werden. Ich möchte Sie daher gerne dazu auffordern, das folgende Skript selbst in RStudio umzusetzen.

Aber genug der langen Vorrede! In diesem Teil der Serie ist das Ziel,

TIPP

Der Autor ist sich bewusst, dass gerade für Einsteiger der Anfang etwas holprig sein kann. Aber auch bei erfahreneren Lesern können durchaus Fragen aufkommen. Um hier Hilfestellung zu bieten, die Antworten der Community zugänglich zu machen und den Austausch anzuregen, möchte er Sie ermutigen, in der extra hierfür gegründeten Facebook-Gruppe nachzufragen. Es gibt dabei keine „dummen" Fragen. Vielmehr hätte der Autor sich in seiner Anfangszeit selbst ein solches Forum mit dem Fokus auf analytische Programmierung und Online-Marketing sehr gewünscht. *https://www.facebook.com/groups/ompyr/*

die Sessions aus GA sowie die Performance-Daten aus der GSC abzufragen, um die Packages für diese APIs besser kennenzulernen. Darüber hinaus werden Sie am Beispiel von Sistrix eine eigene API-Abfrage schreiben, um den Sichtbarkeitsindex für eine Domain zu ermitteln. Als Beispiel-Domain dient hier die Website *dein-trueffel.de*, ein Studentenprojekt an der Hochschule Darmstadt im Modul „Fortgeschrittene Suchmaschinenoptimierung", das durch Jens Fauldrath und Stefan Keil betreut wird. Voraussetzung zum Nachvollziehen des Skripts sind eine aktuelle R-Installation (*www.r-project.org*) und RStudio (*www.rstudio.com*).

Anlegen eines neuen Projekts in RStudio

Zunächst wird ein neues Projekt angelegt (File → New Project … → New Directory → New Project; Abb. 1). Benennung und Speicherort können Sie frei wählen. Der Einfachheit halber wird für diesen Beitrag das Projekt „report" (Directory name) unter C:\Users\{USER}\Documents\ (im Eingabedialog durch ~ automatisch abgekürzt; Abb. 2) erzeugt.

Das Projekt ist zunächst leer, im File-Explorer finden Sie nur eine RStudio-Projektdatei (report.Rproj), die ignoriert werden kann. Daher wird als erstes über File → New File → R Script (oder den darunter befindlichen Button) eine neue Datei erzeugt und unter der Benennung api_calls.R gespeichert.

Installation und Laden der benötigten Packages

In der soeben erstellen api_calls.R-Datei werden nun die benötigten Packages installiert bzw. – falls bereits vorhanden – aktualisiert und geladen (Abb. 4; 1). Die Google-Packages kennen Sie bereits aus dem Beitrag von Tobias Aubele. Das Package lubridate erweitert den Funktionsumfang von R für die komfortable Arbeit mit Datums- und Zeitwerten, jsonlite dient zum Abfragen und Verarbeiten

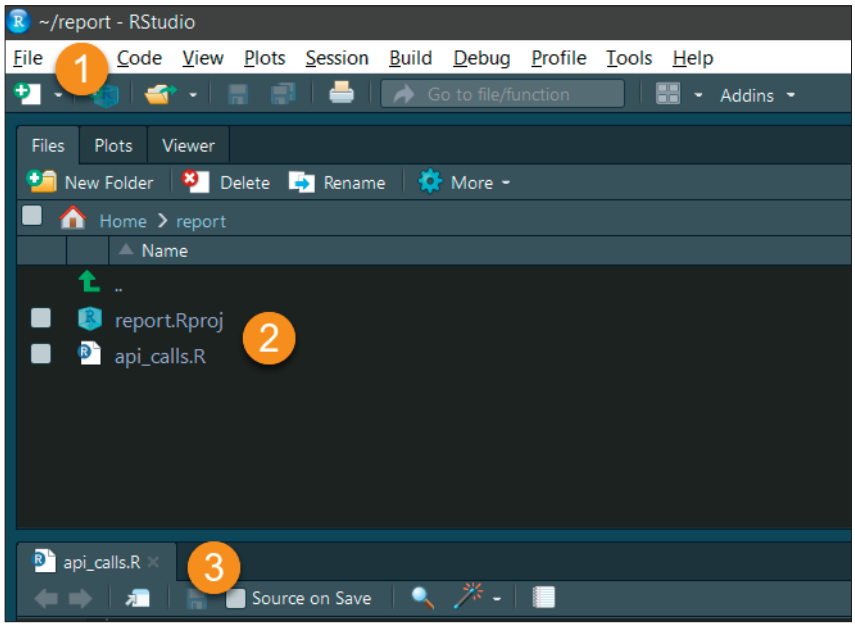

Abb. 3: (1) Neues Skript erstellen, (2) Projekt-Datei und Skript im File-Explorer, (3) Skript-Editor zeigt die erstellte Datei an

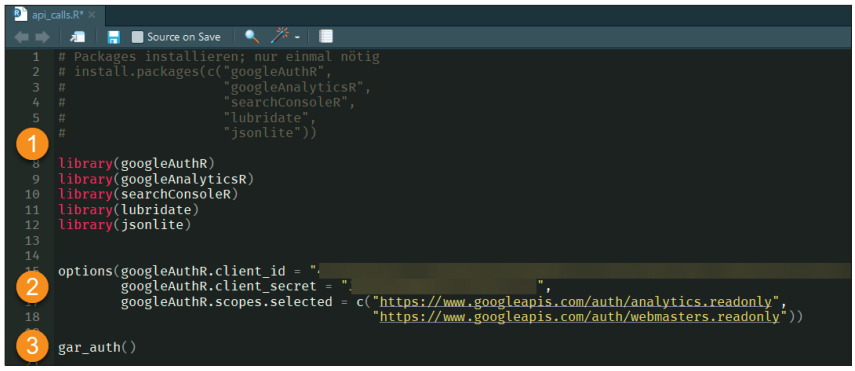

Abb. 4: Installieren bzw. Aktualisieren und Laden der benötigten Packages

von JSON-Dateien. Sobald Funktionen aus diesen Packages verwendet werden, wird explizit darauf hingewiesen. Die vorangestellten Rauten kommentieren den Code aus. Beim ersten Ausführen müssen Sie diese also entfernen. Markieren Sie dann den gesamten Code mit der Maus und führen Sie ihn durch Drücken der STRG- und ENTER-Tasten aus.

In der anschließenden options()-Funktion definieren Sie Ihre Google-Client-ID sowie Ihr -Secret (2). Diese können Sie sich unter console.developers.google.com generieren. Das Argument scopes definiert die Berechtigungen für die GA- bzw. GSC-API. gar_auth() dient schließlich der Authentifizierung (3). Führen Sie den Code das erste Mal aus, öffnet sich Ihr Browser, in dem Sie den Zugriff des Skripts bestätigen müssen. Ist dies erfolgt, finden Sie in Ihrem File-Explorer eine Authentifizierungsdatei (sc.oauth). Bei einem erneuten Ausführen des Skripts verwendet gar_auth() diese Datei, sodass Sie nicht erneut eine manuelle Authentifizierung durchführen müssen – praktisch, denn zukünftig soll das Skript ja automatisch laufen.

Konfiguration wichtiger Variablen

Nun folgt ein Code-Abschnitt, in dem einige Variablen definiert werden (Abb. 5), die an verschiedenen Stellen des Skripts zum Einsatz kommen. Dass die Variablen hier großgeschrieben werden, ist eine Konvention des Autors, die anzeigt, dass die Variablen erst

```
# Konfiguration ----------------------------------------------------

# Google Analytics View-ID ermitteln und eintragen
# ga_account_list() %>% View()
GA_VIEW_ID ← "[redacted]"

# GSC Property ermitteln und eintragen
# list_websites() %>% View()
GSC_PROP ← "https://dein-trueffel.de/"

# Sisitrix
SISTRIX_DOMAIN ← "dein-trueffel.de"
SISTRIX_API_KEY ← "[redacted]"

# Start- und Endzeitpunkt der Datenabfragen definieren
# Hier: Sechs Monate rückwirkend vom Beginn des aktuellen Monats bis zu
# aktuellen Datum
START_DATE ← floor_date(today(), "month") %m-% months(6)
END_DATE ← today()
```

Abb. 5: Definition globaler Variablen, für die die Daten von den APIs abgefragt werden sollen

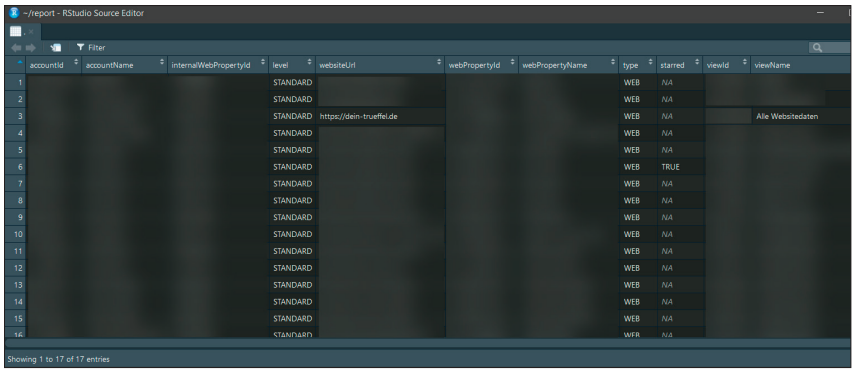

Abb. 6: Data-Explorer mit den GA-Accounts

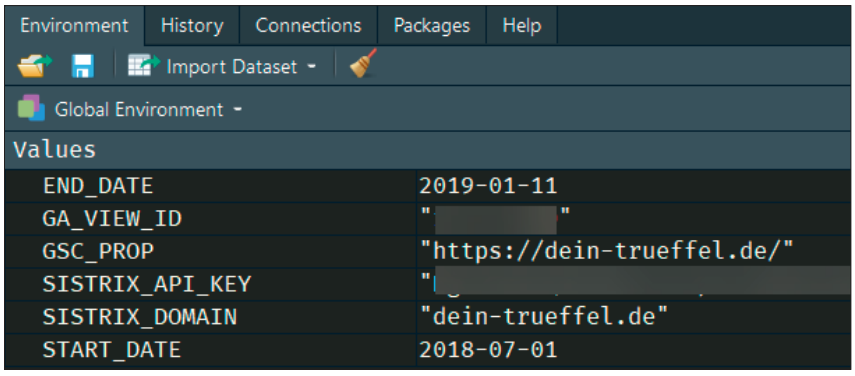

Abb. 7: Environment oder Variable-Explorer listet alle zugewiesenen Variablen

```
# Google Analytics -------------------------------------------------

# Organic Search / Direct Sessions abfragen

channel_filter_organic ← dim_filter(dimension = "channelGrouping",
                                    operator = "EXACT",
                                    expressions = "Organic Search")

channel_filter_direct ← dim_filter(dimension = "channelGrouping",
                                   operator = "EXACT",
                                   expressions = "Direct")

filter_clause ← filter_clause_ga4(list(channel_filter_organic,
                                       channel_filter_direct),
                                  operator = "OR")

ga_sessions ← google_analytics(viewId = GA_VIEW_ID,
                               date_range = c(START_DATE, END_DATE),
                               metrics = c("sessions"),
                               dimensions = c("date", "channelGrouping"),
                               dim_filters =  filter_clause,
                               anti_sample = TRUE)
```

Abb. 8: API-Abfrage der GA-Sessions

TIPP

Um Ihnen das Abtippen des Codes zu ersparen, können Sie sich das Skript zu diesem Artikel unter folgendem Link auf GitHub herunterladen. Dennoch empfehle ich Ihnen, den Code selbst zu schreiben, um ein Gefühl für die Syntax zu bekommen. *http://einfach.st/r4seocode*

zu einem späteren Zeitpunkt genutzt werden – grundsätzlich könnten Sie die Variablen auch kleinschreiben. Dies ist somit die Stelle im Skript, an der Sie Ihre eigenen Daten eintragen müssen.

Im Detail:

(1) Um die passende View-ID Ihrer GA-Datensicht zu ermitteln, können Sie die auskommentierte Funktion ga_account_list() %>% View() verwenden. Der Pipe-Operator %>% übergibt den von der API zurückgegebenen DataFrame mit den Account-Details an die Funktion View(), welche die Tabelle in einem Explorer statt auf der Console öffnet (Abb. 6). Falls der letzte Satz nur ein böhmisches Dorf für Sie war, seien Sie beruhigt, eine ausführlichere Erklärung folgt im nächsten Artikel. Wichtig ist erst einmal nur, dass Sie Ihre View-ID definieren. Entnehmen Sie dazu die gewünschte View-ID aus der Spalte viewId und tragen Sie sie bei der Variable GA_VIEW_ID ein.

(2) Analog dazu verfahren Sie mit der siteUrl für die Variable GSC_PROP.

(3) Die Sistrix-Daten sollen später auf Domain-Ebene abgefragt werden, entsprechend tragen Sie bei SISTRIX_DOMAIN diese ein. Ihren Sistrix-API-Key für die Variable SISTRIX_APY_KEY finden Sie unter app.sistrix.com/account/api.

(4) START_DATE und END_DATE definieren den abzufragenden Berichtszeitraum. In der aktuellen Konfiguration umfasst dieser sechs Monate rückwirkend vom Beginn des aktuellen Monats bis zum heutigen Datum. floor_date(to-

day(), „month“) %m-% months(6) bedient sich dreier Funktionen aus dem Package lubridate. today() gibt das heutige Datum zurück („2018-01-11“). floor_date(„2018-01-11“), „month“) rundet dann zum Beginn des aktuellen Monats („2018-01-01“). %m-% months(6) subtrahiert von diesem Datum sechs Monate („2018-07-01“). %m-% ist ein besonderer Operator aus dem Package, der das Subtrahieren von Monaten unter Berücksichtigung unterschiedlicher Tageszahlen im Monat und Schaltjahren unterstützt. Sie landen mit dieser Funktion also immer am ersten eines Monats.

Die definierten Variablen können Sie nun im Environment sehen (Abb. 7).

GA-Sessions abfragen

Wenn die Konfiguration abgeschlossen ist, kann mit der ersten API-Abfrage begonnen werden. Als Erstes sollen die Sessions, die aus den Channels Organic Search resp. Direct resultieren, aus GA abgerufen werden (Abb. 8). Um die Abfrage auf diese Dimensionen einzuschränken, werden mittels dim_filter() zwei Filter-Komponenten konfiguriert und in der Variablen channel_filter_organic bzw. channel_filter_direct gespeichert (1). Diese werden an eine weitere Funktion filter_clause_ga4() gegeben, welche intern die Filter in eine für die API verständliche Syntax übersetzt (2). Die Funktion google_analytics() führt schließlich die eigentliche Abfrage aus (3). Hier sehen Sie auch die Variablen für die GA-Datensicht sowie das Start- und Enddatum, die Sie zuvor konfiguriert haben. Führen Sie den Code aus, erscheint in Ihrem Environment die Variable ga_sessions (Abb. 9; 1), die Sie mit der Maus anklicken können und die einen DataFrame mit der API-Antwort enthält (2).

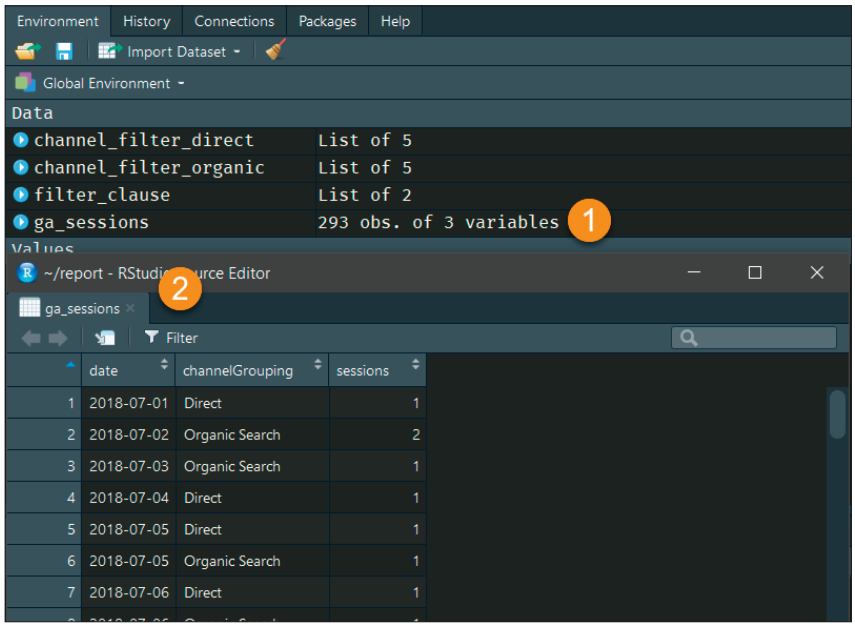

Abb. 9: DataFrame mit den GA-Sessions

```
# GSC ----------------------------------------------------------------

# GSA: Date

1 gsa_dim_date ← search_analytics(siteURL = GSC_PROP,
                                  startDate = START_DATE,
                                  endDate = END_DATE,
                                  searchType = "web",
                                  dimensions = "date")

# GSA: Date ~ Query

2 gsa_dim_date_query ← search_analytics(siteURL = GSC_PROP,
                                        startDate = START_DATE,
                                        endDate = END_DATE,
                                        searchType = "web",
                                        dimensions = c("date", "query"))

# GSA: Date ~ Page

3 gsa_dim_date_page ← search_analytics(siteURL = GSC_PROP,
                                       startDate = START_DATE,
                                       endDate = END_DATE,
                                       searchType = "web",
                                       dimensions = c("date", "page"))
```

Abb. 10: API-Abfrage der GSC-Metriken für verschiedene Dimensionskombinationen

GSC-Performance-Daten abfragen

Ähnlich funktioniert die anschließende Abfrage der GSC-API (Abb. 10). Von dieser werden verschieden Dimensions-Kombinationen abgefragt. (1) Die erste Abfrage ruft die Clicks, Impressions, CTR und Position für den searchType „web“ auf Tagesbasis ab. Auch hier sehen Sie die Verwendung der anfangs definierten Variablen. (2) Dann erfolgt die Abfrage der Metriken für die Kombination aus Datum und Suchanfrage sowie schließlich (3) für die Kombination aus Datum und URL. Auch hier finden Sie nach dem Ausführen der API-Calls die Antwort-Tabellen in Ihrem Environment.

Sistrix-Sichtbarkeitsindex abfragen

Zu guter Letzt müssen Sie noch selbst eine API-Abfrage schreiben, denn es gibt natürlich nicht für alle Datenquellen fertige Packages. Sistrix ist so ein Fall, der aber aufgrund der sehr einfachen Abfrage-Syntax ohne großen Aufwand umgesetzt werden kann. Die Abfrage ist nämlich nur eine URL, deren Query-Parameter die gewünschten Abfragewerte aufnehmen. Wird die URL aufgerufen, erhalten Sie als Antwort

```
# Sistrix ----------------------------------------------------------

# API-URLs zusammenbauen
si_url_desktop ← paste0("https://api.sistrix.com/domain.sichtbarkeitsindex?history=true&format=json&api_key=",
                        SISTRIX_API_KEY,
                        "&domain=",
                        SISTRIX_DOMAIN)

si_url_mobile ← paste0("https://api.sistrix.com/domain.sichtbarkeitsindex?history=true&format=json&api_key=",
                       SISTRIX_API_KEY,
                       "&domain=",
                       SISTRIX_DOMAIN,
                       "&mobile=true")

# API abfragen
api_response_desktop ← fromJSON(si_url_desktop)
api_response_mobile ← fromJSON(si_url_mobile)

# JSON-Antwort in DataFrame überführen
si_desktop ← api_response_desktop$answer$sichtbarkeitsindex[[1]]
si_mobile ← api_response_mobile$answer$sichtbarkeitsindex[[1]]
```

Abb. 11: Konstruktion der Sistrix-API-Abfrage

~/report - RStudio Source Editor

api_response_desktop

Show Attributes

Name	Type	Value
api_response_desktop	list [3]	List of length 3
method	character [1 x 1]	'domain.sichtbarkeitsindex'
answer	list [1 x 1] (S3: data.frame)	A data.frame with 1 rows and 1 columns
sichtbarkeitsindex	list [1]	List of length 1
[[1]]	list [566 x 3] (S3: data.frame)	A data.frame with 566 rows and 3 columns
domain	character [566]	'dein-trueffel.de' 'dein-trueffel.de' 'dein-truef...
date	character [566]	'2019-01-14T00:00:00+01:00' '2019-01-07...
value	double [566]	0.0015 0.0002 0.0002 0.0002 0.0000 0.0002 ...
credits	list [1 x 1] (S3: data.frame)	A data.frame with 1 rows and 1 columns
used	integer [1]	566

api_response_desktop

Abb. 12: Liste mit der Antwort der Sistrix-API

TIPP

Sollten Sie Lust auf R bekommen haben, empfehle ich Ihnen als Einstieg das Buch „R for Data Science“ von Hadley Wickham. Es bietet einen sehr guten Einblick in das Verarbeiten und Visualisieren von Daten in R und steht unter folgendem Link kostenfrei online zur Verfügung. *https://r4ds.had.co.nz/*

eine JSON-Datei. Diese URL bauen Sie mit der Funktion pasteO() zusammen (Abb. 11; 1). Konkret verbindet die Funktion die in Klammern angegebenen Argumente. Diese bestehen aus den Bestandteilen der Basis-URL für die Abfrage des historischen Sichtbarkeitsindexes (SI), in die die zuvor definierten Variablen für Ihren API-Key sowie die abzufragende Domain eingefügt werden.schließen fragen Sie die zusammengebauten URLs gegen die API an (2). Dazu verwenden Sie die Funktion fromJSON() aus dem Package jsonlite. Die Funktion führt einen GET-Request aus, nimmt die JSON-Antwort der API entgegen und überführt sie automatisch in eine Liste – ein R-Datentyp zum Speichern hierarchischer Daten. Die Antwort wird zunächst in die jeweiligen Variablen geschrieben. Klicken Sie eine der beiden Variablen im Environment an, sehen Sie, dass das JSON in eine Listen-Struktur überführt wurde (Abb. 12). Hier sehen Sie auch schon den „Weg“, den Sie innerhalb der Liste gehen müssen, um an die SI-Daten heranzukommen. Denn neben diesen erhalten Sie noch weitere Meta-Informationen, wie bspw. die verbrauchten Credits, zurück. Um die Tabelle mit den SI-Daten zu extrahieren, müssen Sie sich an den übergeordneten Listenpunkten entlanghangeln. Dies macht der Ausdruck api_response_desktop$answer$sichtbarkeitsindex[[1]] für den Desktop-SI und überführt den DataFrame in die Variable si_desktop (Abb. 11; 3).

Fazit und Ausblick

Damit haben Sie das Ziel dieses Beitrags erreicht. Sie haben ein reproduzierbares Skript geschrieben, dass Sie jederzeit erneut ausführen können. Der Berichtszeitraum der abzufragenden Daten passt sich bei jeder erneuten Ausführung automatisch an. Als Resultat liegen Ihnen die GA- , GSC- und Sistrix-Daten in tabellarischer Form vor. Im nächsten Artikel dieser Reihe wird es dann um die Aufbereitung dieser Daten gehen. Aktuell liegen die Daten auf Tagesbasis vor. Für den Report sollen bspw. die GSC-Daten auf Query- resp. URL-Basis aggregiert werden, um somit Tabellen der Top-10-Suchphrasen bzw. URLs der Vorwoche zu erstellen. ¶