Foto: iunewind / gettyimages.de

Patrick Lürwer

R4SEO: AUTOMATISIERTE REPORTS MIT R GENERIEREN (TEIL 5/5)

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 vorherigen Teil dieser Serie haben Sie mittels ggplot2 die Daten, die sie von verschiedenen APIs abgefragt haben, visualisiert. Im vorliegenden fünften und damit letzten Teil geht es nun darum, alle bisherigen Schritte der Datenbeschaffung, -aufbereitung und -visualisierung in einem sogenannten R-Notebook zu vereinen. R-Notebooks sind ein spezieller Skript-Typ, der R und Markdown in einem Dokument vereint. Dadurch können Code, Text, Tabellen und Diagramme in einem Dokument nebeneinander bestehen. Das Notebook kann dann als Word, PDF, HTML etc. gerendert und – wie im vorliegenden Fall – als Report verschickt werden.

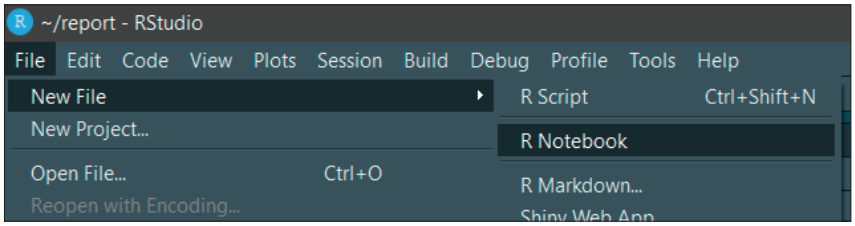

Abb.1: Erstellen eines R-Notebooks

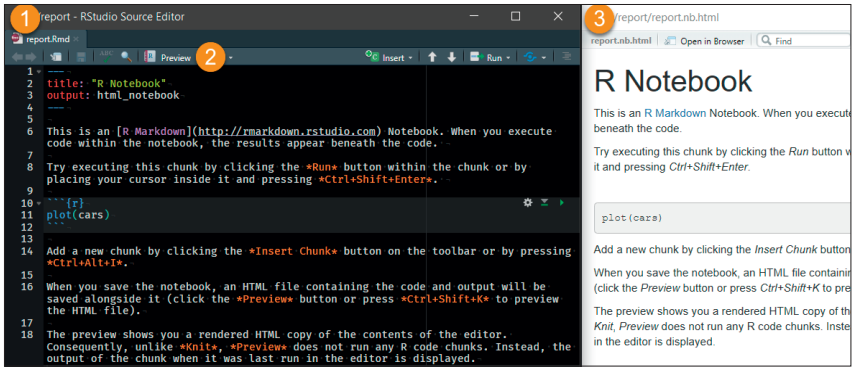

Abb.2: R-Notebook und Preview

In den vorausgehenden Teilen dieser Serie haben Sie alle Datentransformationen und -visualisierungen in „normalen“ R-Skripten vorgenommen (Dateien die auf .R enden). Eine Besonderheit von R bzw. RStudio ist, dass es einen weiteren Dateityp zur Verfügung stellt – das sogenannte R-Markdown (Dateiendung .Rmd). R-Markdown bietet die Möglichkeit, R-Code und Markdown (Texte mit einfacher Formatierung) in einem Skript zu vereinen, welches dann in verschiedenen Formaten exportiert werden kann. So lassen sich HTML-, Word- und PDF-Dokumente, aber auch PowerPoint-Präsentationen und sogar Dashboards, Websites und ganze Bücher erstellen. Oder eben ein R-Notebook, welches im Grunde ein HTML-Dokument mit einigen zusätzlichen interaktiven Funktionen ist.

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. Dennoch empfehle ich Ihnen, den Code selbst zu schreiben, um ein Gefühl für die Syntax und Funktionen zu bekommen. *https://github.com/gettractiongmbh/r4seo_ws*

Für den hier beschriebenen Report werden Sie ein HTML-Dokument generieren, da es die denkbar einfachste Variante ist. Grundsätzlich sind Ihnen aber ob der mannigfaltigen Export-Formate kaum Grenzen gesetzt. Das zugrunde liegende Konzept bleibt dabei immer das gleiche: Der Code, der die darzustellenden Daten erzeugt, und der die Darstellungen beschreibende Text liegen immer in einem Dokument vor. Dadurch ergeben sich insbesondere zwei Vorteile hinsichtlich der Nachvollziehbarkeit und Reproduzierbarkeit aller mittels R-Markdown erstellten Dokumente. 1) Der Code ist nicht versteckt, sondern unmittelbar sichtbar. Sollten Sie zu einem späteren Zeitpunkt im Detail nachvollziehen wollen, wie bestimmte Daten für ein Diagramm o. Ä. aufbereitet wurden, müssen Sie nicht lange suchen. 2) Da es sich weiterhin um ein Skript handelt, kann es zu jedem beliebigen Zeitpunkt erneut ausgeführt werden. Von diesem Vorteil machen Sie beim zu erstellenden Report Gebrauch, denn im Grunde werden Sie immer wieder dasselbe Export-Dokument generieren – nur eben mit aktualisierten Daten, die automatisch von den APIs abgefragt werden.

TIPP

Eine schnelle Einführung finden Sie auf der offiziellen Website: *https://rmarkdown.rstudio.com/index.html*. Eine sehr ausführliche Beschreibung aller Funktionen finden Sie in dem Buch *https://bookdown.org/yihui/rmarkdown/* – das übrigens ebenfalls in R-Markdown geschrieben wurde.

Wie müssen Sie sich ein solches R-Markdown-Dokument resp. R-Notebook nun konkret vorstellen?

Einführung in R-Notebooks

Um eine Vorstellung von so einem Notebook zu bekommen, legen Sie am besten einfach eines an. Innerhalb Ihres Projekts, das Sie bisher auch verwendet haben, navigieren Sie in der Menü-Leiste von RStudio zu Files → New File → R Notebook (Abbildung 1).

Im Panel des Code-Editors wird sich nun ein R-Notebook mit einer Beispiel-Befüllung öffnen (Abbildung 2; 1). Klicken Sie im oberen Bereich des Panels auf Preview (2). Es öffnet sich ein Speicher-Dialog. Speichern Sie das Dokument ruhig schon einmal als report.Rmd, da Sie es nachher mit Ihrem Code befüllen werden. Nach dem Speichern öffnet sich ein weiteres Fenster, das eine Vorschau des gerenderten Notebooks bietet (3).

Hier sehen Sie die wichtigsten Elemente, die ein R-Markdown-Skript auszeichnen. Am Anfang des Skripts werden zwei Meta-Daten im YAML-Form aufgeführt: zum einen der Titel des Dokuments, zum anderen das Ausgabe-Format. Ersteren sehen Sie auch in der Vorschau. Benutzen Sie ruhig das Dropdown-Menü des Preview-Buttons, um bspw. ein Word-Dokument zu erzeugen. Neben der Preview öffnet sich nun auch Word. Ändern Sie danach aber wieder das Format auf ein R-Notebook (output: html_notebook).

Im Anschluss an die Meta-Daten folgt klassisches Markdown. Sie erken-

nen einen gesetzten Link und einige kursiv formatierte Satzelemente.

Das Besondere an R-Markdown ist der nachfolgende sogenannte R-Code-Chunk. Alles, was in ```{r} ... ``` eingefasst ist, wird als R-Code ausgeführt. Im vorliegenden Chunk wird mittels plot(cars) ein Beispiel-Datenset geplottet. Um den Chunk auszuführen, klicken sie im rechten oberen Bereich auf den grünen Pfeil (Abbildung 3). Anders als sonst beim Plotting erscheint das Diagramm nun im Skript selbst. Drücken Sie erneut auf Preview (Shortcut-Tipp: Strg + Shift + K), aktualisiert sich die Vorschau und auch dort wird das Diagramm jetzt unterhalb des generierenden Code-Snippets angezeigt.

Was Sie hier sehen – das Nebeneinander von Code(-Output) und Text – nennt sich Literate Programming und geht auf die Idee von Donald E. Knuth zurück, Computerprogramme in einer Form zu schreiben, dass sie vor allem für Menschen lesbar sind. Im Wesentlichen greift R-Markdown zwei Aspekte dieses Paradigmas auf: 1) Quellcode und Kommentare können miteinander gemischt werden sowie 2) aus dem Programm und der Beschreibung kann automatisch eine lesbare Dokumentation mit Inhaltsverzeichnis, Verweisen etc. generiert werden. Übertragen auf das Anwendungsfeld explorativer Datenanalysen bietet R-Markdown damit die Möglichkeit, den gesamten Prozess der Datenbeschaffung, -transformation und -visualisierung sowie die darauf aufbauende Analyse in einem Dokument vorzunehmen. Nachvollziehbarkeit und Reproduzierbarkeit einer solchen Analyse sind damit zu jedem Zeitpunkt sichergestellt.

Zugegeben, die eine Zeile Code, die das Diagramm erzeugt, bietet nicht signifikant mehr Informationen zum Verständnis des Dokuments. Aber es handelt sich ja auch nur um ein Beispiel. Möchten Sie sich einen tiefergehenden Einblick in ein solches Analyse-Dokument verschaffen, finden Sie unter https://www.gettraction.de/r4seo einen Blogpost des Autors, der exemplarisch eine Crawl-Analyse mittels R-Markdown beschreibt.

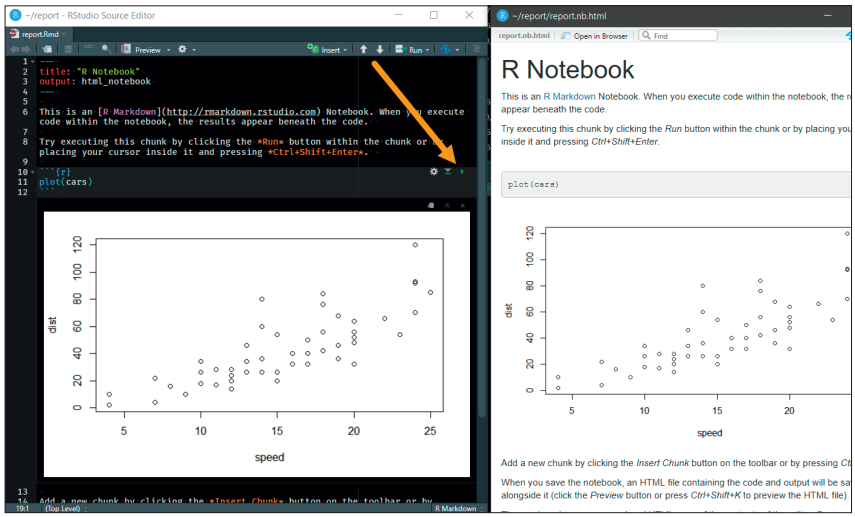

Abb.3: Ausführen eines R-Code-Chunks

Abb.4: Konfiguration des Notebooks

Sie werden im Nachfolgenden den Aspekt der Code-Darstellung ignorieren, denn bei Reports dürfte es die Empfänger herzlich wenig interessieren, wie die Diagramme und Tabellen erstellt wurden. Eine Möglichkeit, den Code auszublenden, sehen Sie bereits in Abbildung 3. Oberhalb eines jeden Code-Snippets findet sich ein Hide-Button, mit der der Code eingeklappt werden kann. Dies müssten Sie allerdings für jedes Snippet manuell machen. Es gibt zwar auch eine Option (klicken Sie dazu auf das Zahnrad-Symbol neben dem Preview-Button → Output-Options → Fold R code chunks → hide), alle Snippets standardmäßig einzuklappen. Allerdings wird der Button weiterhin angezeigt. Das geht auch eleganter. Aber der Reihe nach.

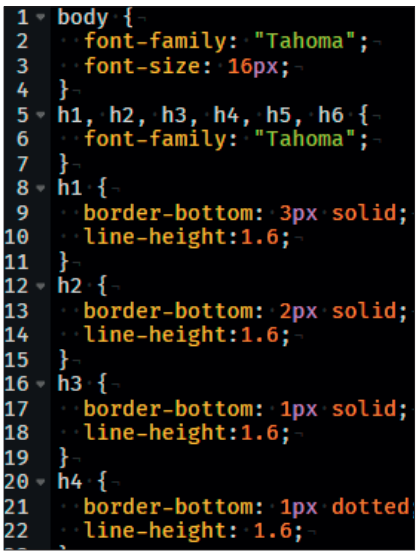

Abb. 5: CSS-Datei des Notebooks

Überführung des bisherigen Codes in ein R-Notebook

Um nicht auf Beispiel-Daten und -Text zu arbeiten, werden Sie nun Ihren bisherigen Code, der in dem Skript visualization.R vorliegt, in das Notebook überführen. Sie werden dazu Schritt für Schritt vorgehen, da vereinzelt kleinere Ergänzungen vorgenommen werden müssen.

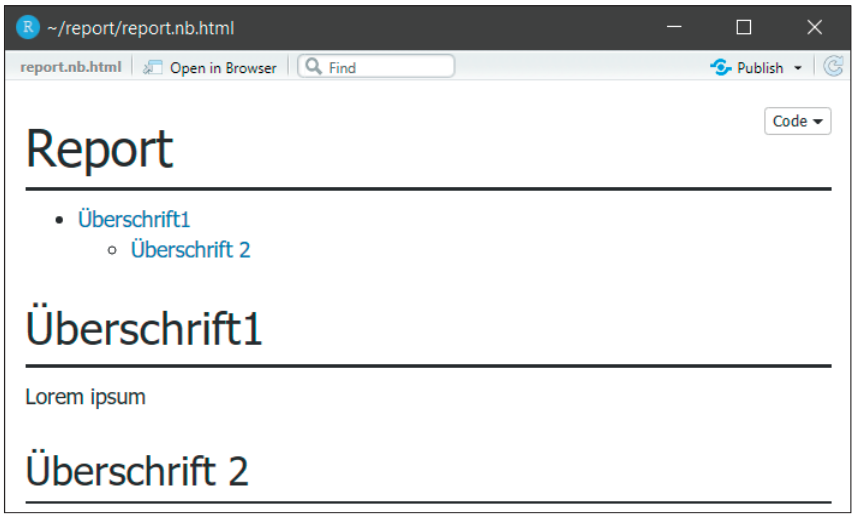

Abb.6: Notebook mit CSS-Formatierung und Inhaltsverzeichnis

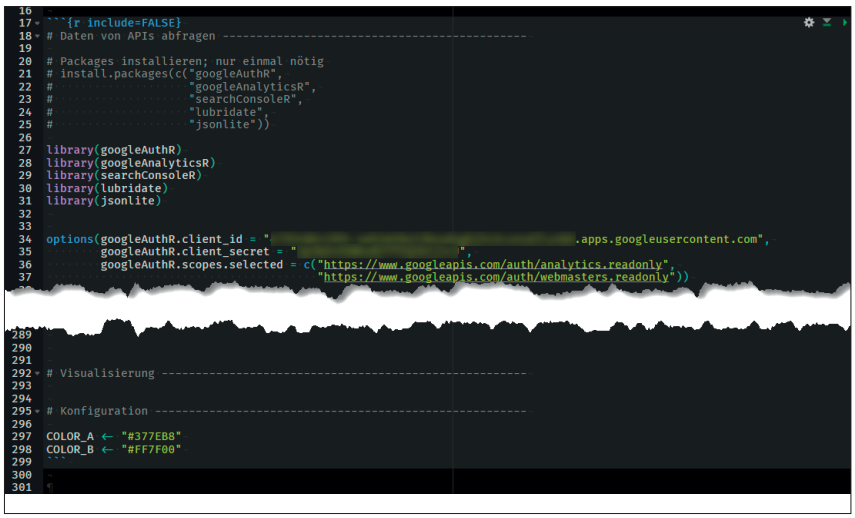

Abb.7: Code-Chunk mit Datenabfrage und -aufbereitung

Abb.8: Output-Konfiguration auf Chunk-Ebene

Löschen Sie zunächst den gesamten Beispiel-Inhalt des Notebook report.Rmd und beginnen Sie damit, die Meta-Angaben zu definieren (Abbildung 4; 1). Den Titel können Sie natürlich frei wählen. Sie sehen, dass html_notebook nun ebenfalls zwei untergeordnete Konfigurationselemente aufnimmt. Mittels CSS geben Sie den Pfad zu einer CSS-Datei an. Da das Output-Format HTML sein wird, können Sie es natürlich gänzlich via CSS Ihren Wünschen entsprechend anpassen. Die hier verwendete CSS-Datei (Abbildung 5) ist sehr simpel und enthält nur einige Anweisungen, die den Font verändern und die Überschriften unterstreichen. Um die CSS-Datei zu erstellen, gehen Sie über File → New File → Text File und speichern Sie die Datei als style.css in Ihrem aktuellen Working-Directory.

Weiterhin sehen Sie im YAML, dass toc: yes definiert wurde. Dadurch geben Sie an, dass basierend auf den Überschriften im Output-Dokument automatisch ein Inhaltsverzeichnis generiert werden soll. Das Ergebnis mit zwei vom Autor exemplarisch eingefügten Überschriften sehen Sie in Abbildung 6.

Unterhalb des YAML fügen Sie nun Ihren ersten Code-Chunk ein. Wenn Sie Strg + Alt + I (I wie Ida) drücken, fügt RStudio automatisch ein Chunk-Template an der Stelle des Cursors ein. Innerhalb des Chunks definieren Sie (Abbildung 4; 2), dass keine Code-Snippets im Output-Dokument erscheinen sollen, das, was sie generieren – also bspw. Tabellen und Diagramme –, hingegen schon (echo = FALSE). Außerdem unterdrücken Sie die Ausgabe aller Benachrichtigungen (message = FALSE) und Warnungen (warning = FALSE). Benachrichtigungen treten beispielsweise beim Laden von Packages auf. Normalerweise würden sie auf der Console ausgegeben werden. Im Kontext eines R-Notebooks werden sie allerdings unterhalb des verursachenden Code-Chunks angezeigt und wären somit schließlich auch im Output-Dokument vorhanden. Das ist natürlich nicht gewünscht.

Als Nächstes fügen Sie einen weiteren Chunk ein, in den Sie den gesamten Code für die Datenabfrage und -aufbereitung sowie die Definition der zu verwendenden Farbwerte kopieren (Abbildung 7).

Beachten Sie, dass Sie für diesen Chunk noch eine spezifischere Konfiguration der Ausgabe vornehmen müssen. Im vorherigen Chunk haben sie definiert, dass der Code aller Chunks des Notebooks nicht als Code-Snippet im Output-Dokument erscheinen soll (echo = FALSE). Diese Einstellung unterdrückt allerdings nur das Code-Snippet, nicht die aus dem Code resultierenden Diagramme und Tabellen. Die wollen

Sie schließlich in Ihrem Report sehen. Im aktuellen Chunk befinden sich allerdings die Abfragen der GSC- und GA-API, die während des Calls Informationen auf der Console resp. nun im Notebook ausgeben, die durch echo = FALSE nicht unterdrückt werden. Für den gesamten Code innerhalb des Chunks wollen Sie jedoch im Grunde, dass nur der Code ausgeführt, ansonsten aber nichts im Output-Dokument angezeigt wird. Um diese spezifischere Konfiguration vorzunehmen, klicken Sie das Zahnrad-Symbol an und wählen Sie bei Output → Show nothing (run code) aus (Abbildung 8). Sie sehen, dass am Anfang des Chunks nun {r include=FALSE}, also die spezifischere Output-Anweisung, die die globale überschreibt, eingefügt wurde.

Falls Sie es bisher noch nicht getan haben, führen sie die beiden Chunks aus (Shortcut: Strg + Alt + R) und nehmen Sie sich die Zeit, ein bisschen mit der Output-Konfigurationen der Chunks zu experimentieren.

Wenn Sie der Anleitung bis hierher gefolgt sind und die Preview des Notebooks anzeigen, sollten Sie nichts sehen – bis auf den Titel des Reports natürlich. Im Nachfolgenden beginnen Sie jetzt damit, den Code hinzuzufügen, der für die Generierung der Diagramme und Tabellen verantwortlich ist.

Einbinden der Plots und Tabellen in den Report

Als Erstes wollen Sie aber mit Sicherheit wissen, für welche Domain der Report eigentlich gilt und welchen Berichtszeitraum er abdeckt. Fügen Sie dazu den Satz aus Abbildung 9 in Ihr Notebook ein. Sie sehen, dass er nicht ```{R} ...``` eingefasst ist, da Sie nun reinen Text schreiben, bzw. nicht ausschließlich Text, denn innerhalb des Satzes finden Sie mit `r ...` umfasste Variablen, die Sie bereits kennen. Mit R-Markdown können Sie also auch Inline-Code-Chunks verwenden, die kurze

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

COLOR_A <- "#377EB8"
COLOR_B <- "#FF7F00"
```

Report für die Domain **`r SISTRIX_DOMAIN`** vom **`r START_DATE`** bis **`r END_DATE`**

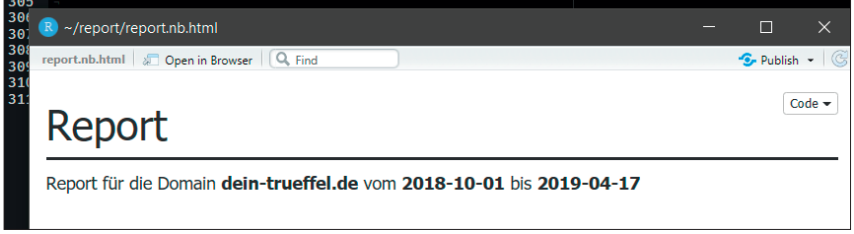

Abb.9: Satz mit Inline-Chunks

```
# Google Analytics

```{r}
# GA ----------------------------------------------------------------------

ggplot(data = ga_sessions,
       aes(x = date,
           y = rollmean_sessions,
           color = channelGrouping)) +
  geom_line(size = 1) +
  geom_smooth(method = lm,
              se = TRUE,
              size = 0.5) +
  theme_minimal() +
  theme(plot.title = element_text(face = "bold"),
        plot.subtitle = element_text(color = "#666666")) +
  labs(title = "Google Analytics - Sessions",
       subtitle = "Gleitender Mittelwert der Sessions je Channel-Grouping",
       x = "Kalenderwoche",
       y = "Sessions",
       color = "Channel Grouping") +
  scale_color_manual(values = c(COLOR_A, COLOR_B)) +
  scale_x_date(date_breaks = "2 weeks",
               date_labels = "%W\n'%y")
```


Google Search Console

```{r}
ggplot(gsa_dim_date, aes(date, value, color = metric)) +
  geom_line(size = 1) +
  geom_smooth(method = loess,
              size = 0.5,
              se = FALSE) +
  facet_grid(metric~.,
             scales = "free_y",
             labeller = labeller(metric = c(rollmean_clicks = "Clicks",
                                            rollmean_impressions = "Impressions"))) +
  theme_minimal() +
  theme(plot.title = element_text(face = "bold"),
        plot.subtitle = element_text(color = "#666666")) +
  labs(title = "GSC - Traffic",
       subtitle = "Gleitender Mittelwert der Impressions / Clicks",
       x = "",
       y = "") +
  scale_color_manual(values = c(COLOR_A, COLOR_B)) +
  scale_x_date(date_breaks = "1 month", date_labels = "%b\n'%y") +
  guides(color = F)
```
```

Abb. 10: Einbindung des Google-Analytics-Plots

R-Statements ausführen. In diesem Fall geben Sie einfach den Inhalt der Variablen wieder, die sie bereits zur Datenabfrage verwendet haben und die die Domain resp. das Start- und Enddatum des Berichtszeitraums enthalten.

Gerade in Reports – aber durchaus auch in Analysen – bieten diese Inline-Chunks eine willkommene Flexibilität. Stellen Sie fest, dass Sie immer wieder die gleichen Textbausteine verwenden, bei denen sich nur die Zahlenwerte ändern, tragen Sie Letztere einfach als Variablen ein, die sich mit jeder neuen Report-Generierung selbst aktualisieren. Klassische Anwendungsfälle sind die exponierte Darstellung des aktuellen Sichtbarkeitsindexes oder Sätze wie: Das prozentuale Delta der Clicks zur Vorwoche beträgt: ...

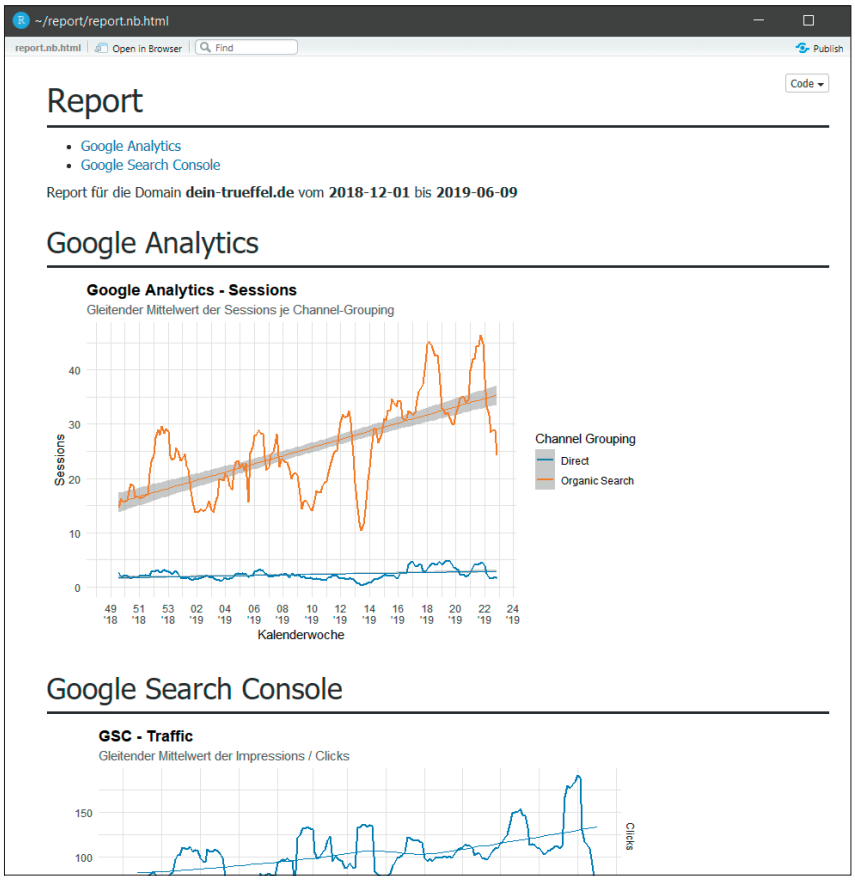

Abb.11: Preview mit integrierten Plots

```
357
358 ## Top 10 Pages
359
360 **Die stärksten Seiten nach Clicks der Vorwoche (KW `r isoweek(MONDAY_PREVIOUS_WEEK)`)**
361
362 ```{r}
363 gsa_top_10_pages
364 ```
365
366
367
368 ## Top 10 Queries
369
370 **Die stärksten Suchanfragen nach Clicks der Vorwoche (KW `r isoweek(MONDAY_PREVIOUS_WEEK)`)**
371
372 ```{r}
373 gsa_top_10_queries
374 ```
375
```

Abb.12: Einbinden der (unformatierten) GSC-Tabellen

Top 10 Pages

Die stärksten Seiten nach Clicks der Vorwoche (KW 22) (1)

Page (2) <chr>	Clicks <dbl>	Impressions (3) <dbl>	CTR <dbl>	Position (4) <dbl>
/trueffel-produkte/trueffelbutter/	538	5753	0.094	4.55
/trueffel-wissen/wo-wachsen-trueffel/	347	5811	0.060	11.46
/trueffel-wissen/trueffelsuche/	125	1277	0.098	8.06
/trueffel-wissen/trueffelsuche/trueffelschwein/	52	1143	0.045	5.38
/trueffel-produkte/trueffelcreme/	41	258	0.159	6.64
/trueffel-rezepte/383-4/	29	241	0.120	13.78
/trueffel-wissen/trueffelsuche/trueffelhund/	24	1052	0.023	9.60
/trueffel-produkte/trueffeloel/	21	387	0.054	5.96
/trueffel-wissen/allgemein/	7	115	0.061	12.59
/trueffel-wissen/arten/fruehlingstrueffel/	5	7	0.714	1.43

1-10 of 11 rows (5) Previous 1 2 Next

Abb.13: (Unformatierte) Output-Tabelle der Top-10-Pages

GA- & GSC-Diagramme einbinden

Wie Sie einen Plot in R-Markdown einbinden, haben Sie bereits oben anhand der Beispiel-Befüllung des Dokuments gesehen. Sie müssen einfach nur den Code, der den Plot generiert, in einem Code-Chunk einfassen. Und genau das machen Sie nun für die Diagramme der Google-Analytics- und Google-Search-Console-Daten (Abbildung 10).

Damit die Diagramme einen automatischen Eintrag im Inhaltsverzeichnis erhalten, stellen sie den Code-Chunks jeweils eine Überschrift voran. Den einzufügenden Code kennen Sie bereits aus dem vorherigen Skript und dieser muss nicht angepasst werden. Führen Sie die Code-Chunks aus und sehen Sie sich die Preview des Dokumentes noch einmal an (Abbildung 11).

Tabellen einfügen und gestalten

Sie erinnern sich vielleicht, für die GSC-Daten haben Sie auch zwei Tabellen erstellt, die die Top-10-Pages resp. -Queries der Vorwoche aufführen. Diese werden sie als Nächstes im Report integrieren. Alles, was Sie dafür machen müssen, ist, die entsprechenden Variablen gsa_top_10_pages und gsa_top_10_queries in separate Code-Chunks einzufassen (Abbildung 12). Führen Sie auch diese Chunks aus und gucken Sie sich erneut den Output in der Preview an.

In Abbildung 13 sehen Sie oberhalb der Tabelle einen Satz, der kurz die zugrunde liegende Datenbasis beschreibt (1). Die ISO-Kalenderwoche fügen Sie hier erneut mittels Inline-Chunk ein, um sie nicht jedes Mal manuell anpassen zu müssen. Dann gibt es noch einige Elemente in der Tabelle, die etwas Politur bedürfen. Zum einen wird unterhalb der Spaltenüberschriften der Datentyp ausgewiesen (2). Der dürfte nun wirk-

lich niemanden interessieren. Zum anderen werden die Zahlen aktuell ohne Tausender-Trennzeichen (3) und mit englischem Dezimal-Trennzeichen (4) dargestellt. Zum Schluss sehen Sie bei (5), dass die Tabelle paginiert ist. Standardmäßig gibt R-Markdown beim Rendern von Tabellen zehn Zeilen je Seite aus. In der aktuellen HTML-Vorschau können Sie hier dynamisch durchblättern. Wenn Sie allerdings ein PDF erzeugen möchten, geht das natürlich nicht. (Hinweis: Ihre Tabelle kann durchaus nur zehn Zeilen enthalten, immerhin geht es hier um die Top-10. Dass im Beispiel elf Zeilen ausgegeben werden, ist dem Umstand geschuldet, dass zwei Seiten fünf Clicks aufweisen.)

Sie möchten Ihre Tabelle also noch etwas formatieren. Die zugrunde liegende Rendering-Engine knitr, die das R-Markdown-Skript in das gewünschte Output-Format überträgt, bietet dazu einige rudimentäre Optionen, die für dieses Beispiel genügen sollen. Wollen Sie Ihre Tabellen auch farblich, mit Icons oder sogar konditional formatieren, schauen Sie sich das Package kableExtra an *(http://haozhu233.github.io/kableExtra/)*.

In Abbildung 14 sehen Sie den um die Tabellen-Formatierung ergänzten Code. knitr::kable lädt die benötigte Funktion aus dem Package. Als erstes Argument wird die zu formatierende Tabelle definiert. Im Anschluss geben Sie format = „markdown“ an, wodurch die Paginierung unterdrückt wird. Innerhalb der format.args folgen dann die von Ihnen gewünschten Tausender- und Dezimal-Trennzeichen. Zum Schluss runden Sie alle Werte der Tabelle auf zwei Nachkommastellen (digits = 2) – erkennbar bei der CTR. Sie erinnern sich vielleicht, dass Sie die Werte der Tabelle bereits weiter oben im Skript mittels round() gerundet haben. Das Runden diente dort der Vorstellung der Funktion. Im Allgemeinen ist es sinnvoller, die Repräsentation der Daten erst beim Rendern zu formatieren, statt die zugrunde liegenden Daten selbst zu manipulieren.

```
361
362 ```{r}
363 knitr::kable(gsa_top_10_pages,
364              format = "markdown",
365              format.args = list(big.mark = ".",
366                                 decimal.mark = ","),
367              digits = 2)
368 ```
```

~/report/report.nb.html

report.nb.html | Open in Browser | Find | Publish

Top 10 Pages

Die stärksten Seiten nach Clicks der Vorwoche (KW 22)

Page	Clicks	Impressions	CTR	Position
/trueffel-produkte/trueffelbutter/	538	5.753	0,09	4,55
/trueffel-wissen/wo-wachsen-trueffel/	347	5.811	0,06	11,46
/trueffel-wissen/trueffelsuche/	125	1.277	0,10	8,06
/trueffel-wissen/trueffelsuche/trueffelschwein/	52	1.143	0,04	5,38
/trueffel-produkte/trueffelcreme/	41	258	0,16	6,64
/trueffel-rezepte/383-4/	29	241	0,12	13,78
/trueffel-wissen/trueffelsuche/trueffelhund/	24	1.052	0,02	9,60
/trueffel-produkte/trueffeloel/	21	387	0,05	5,96
/trueffel-wissen/allgemein/	7	115	0,06	12,59

Abb.14: Formatierte Tabelle der GSC-Daten

```
380
381 # Sistrix
382
383 ```{r}
384 ggplot(si,
385        aes(date,
386            value,
387            color = platform,
388            fill = platform)) +
389   geom_area(alpha = 0.5) +
390   geom_point() +
391   facet_grid(platform~., scales = "free_y") +
392   theme_minimal() +
393   theme(plot.title = element_text(face = "bold"),
394         plot.subtitle = element_text(color = "#666666")) +
395   labs(title = "Sistrix - Sichtbarkeitsindex",
396        subtitle = "nach Platform",
397        x = "",
398        y = "SI") +
399   scale_color_manual(values = c(COLOR_A, COLOR_B)) +
400   scale_fill_manual(values = c(COLOR_A, COLOR_B)) +
401   scale_x_date(date_breaks = "2 weeks", date_labels = "%W\n'%y") +
402   guides(color = F,
403          fill = F)
404 ```
405
```

Abb.15: Einbinden des Sistrix-Diagramms

Einbinden des Sistrix-Plots

Nun bleibt nur noch das Diagramm des Sistrix-Sichtbarkeitsindexes übrig. Wie Sie dieses einbinden, dürfte sie an dieser Stelle nicht mehr überraschen und ist in Abbildung 15 zu sehen.

Report via E-Mail verschicken

Fertig! Sie haben nun ein Skript erstellt, dass automatisch verschiedene APIs anfragt und die Daten in einem HTML-Report aufbereitet. Fertig? Nicht ganz. Sie möchten das Skript natürlich nicht jede Woche manuell anstoßen, sondern hätten den Report gern morgens, wenn Sie ins Büro kommen, bereits in Ihrem E-Mail-Postfach liegen.

Um das Skript automatisch auszuführen und per E-Mail zu verschicken,

```r
library(mailR)

rmarkdown::render("report.Rmd",
                  encoding = "utf-8")

sender <- "[redacted]"

recipients <- c("bar@example.com",
                "foo@example.com")

send.mail(from = sender,
          to = recipients,

          subject = "Wochenreport",

          body = "Hallo,\nhier der wöchentliche Report.",

          attach.files = "report.nb.html",

          authenticate = T,

          smtp = list(host.name = "smtp.gmail.com",
                      port = 465,
                      user.name = "[redacted]",
                      passwd = "[redacted]",
                      ssl = T))
```

Abb.16: Mail-Skript zum Rendern und Versenden des Reports

benötigen Sie nun noch zwei Komponenten. Zum einen ein Mail-Skript, das das Report-Skript ausführt und das gerenderte Output-Dokument per Mail verschickt, zum anderen einen Trigger, der wiederum das Mail-Skript zu einem bestimmten Zeitpunkt anstößt.

Beginnen Sie mit dem Mail-Skript, indem sie ein neues R-Skript erzeugen und mit mail_skript.R benennen. Der Inhalt des Mail-Skripts ist in Abbildung 16 zu sehen. Zunächst laden Sie das Package mailR, mit dem Sie – wie Sie sich wahrscheinlich schon gedacht haben – Mails verschicken können (1). Wahrscheinlich müssen Sie das Package zunächst noch via install.packages(„mailR“) installieren. Im Anschluss (2) laden Sie die Funktion render() aus dem rmarkdown-Package und geben als erstes Argument das zu rendernde Report-Skript an. Das Argument encoding = „utf-8“ sorgt dafür, dass bspw. Umlaute erhalten bleiben. sender nimmt die Mail-Adresse des Versenders auf und recipients einen oder mehrere Empfänger (3). In der Funktion send.mail() (4) definieren Sie noch den Betreff, den Mail-Inhalt, den anzuhängenden Report sowie Ihre Zugangsdaten.

TIPP

Der Einfachheit halber hat der Autor seinen Gmail-Account zum Versenden der Mail verwendet. Grundsätzlich sind auch andere Dienste wie Microsoft Exchange möglich, bedürfen aber mitunter etwas mehr Konfigurationsaufwand, damit eine externe Anwendung Mails verschicken darf. Sollten Sie ebenfalls Ihren Gmail-Account verwenden und die Zwei-Faktoren-Authentifizierung aktiviert haben, müssen Sie sich ein App-Passwort erstellen. Eine Anleitung finden Sie hier: *https://support.google.com/accounts/answer/185833*.

Wie funktioniert das Mail-Skript nun genau? Beim Ausführen wird zunächst das Report-Skript gerendert und das Output-Dokument in Ihrem Working-Directory abgespeichert. Führen Sie ruhig die Render-Funktion manuell aus. Sie sehen auf der Console, dass das R-Markdown-Skript durchläuft. In Ihrer Datei-Übersicht sehen

Sie das gespeicherte Output-Dokument report.nb.html. Letzteres geben Sie in der Mail-Funktion als Anhang an (attach.files).

Als Trigger zur Ausführung verwenden Sie schließlich den Windows-Aufgabenplaner. Rufen Sie diesen auf, klicken Sie in der Navigation auf Aktion → Aufgabe erstellen ... Im ersten Reiter Allgemein des Konfigurations-Dialogs vergeben Sie eine Benennung der Aufgabe wie bspw. R_report. Im zweiten Reiter Trigger definieren Sie, zu welchem Zeitpunkt die Aufgabe laufen soll – etwa jede Woche Mittwoch um 01:00 Uhr. Zu diesem Zeitpunkt muss Ihr Computer natürlich an sein. Unter Aktionen legen Sie eine neue Aktion: Programm starten an (Abbildung 17). Bei Programm/Skript geben Sie den Pfad Rscript.exe an, als Argument des Programms das gerade von Ihnen erstellte mail_skript.R. Starten in definiert, wo das Mail-Skript liegt. Klicken Sie nun auf OK, um die Konfiguration abzuschließen. In der Übersicht sehen Sie nun die erstellte Aufgabe. Um sie zu testen, können Sie sie mit der rechten Maustaste anklicken und Ausführen. Hat alles geklappt, erscheint ein CMD-Fenster, in dem die Ausführung des Skripts angezeigt wird und das sich nach dem Versand der Mail automatisch wieder schließt.

Hinweis: Eventuell erhalten Sie eine Fehlermeldung: Fehler: pandoc version 1.12.3 or higher is required and was not found. rmarkdown kann dann Ihre Pandoc-Installation nicht finden (Pandoc ist ein weiteres Programm, das unter der Haube verwendet wird, um das Skript in das gewünschte Output-Format zu übertragen). In diesem Fall müssen Sie Pandoc ihrem Windows-Suchpfad (Path) hinzufügen. Um den Pfad zu ermitteln, führen Sie in der R-Console folgenden Befehl aus: Sys.getenv(„RSTUDIO_PANDOC“). Den angezeigten Pfad fügen Sie der Umgebungsvariable Path hinzu.

Eine Anleitung dazu finden Sie hier: http://techmixx.de/windows-10-umgebungsvariablen-bearbeiten/.

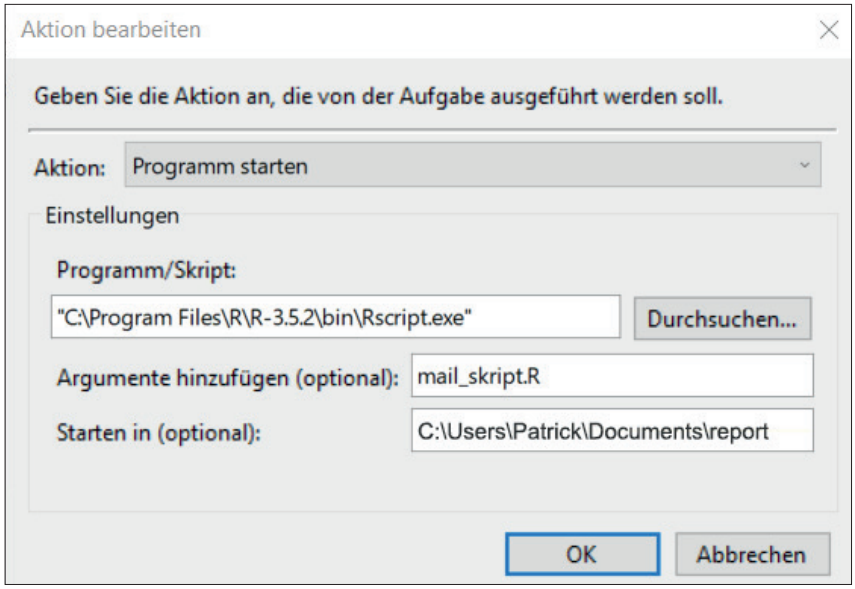

Abb.17: Aktion im Windows-Aufgabenplaner anlegen

Fazit

Damit haben Sie es nun aber wirklich geschafft. Glückwunsch! Sie verfügen jetzt über ein Skript, mit dem Sie verschiedene APIs abrufen, die Daten aufbereiten und visualisieren, in einem Report zusammenfassen und diesen automatisch per Mail verschicken.

Viel wichtiger ist aber, dass Sie einen Einblick in die Datenverarbeitung mit R erhalten haben. Denn viele der von Ihnen durchgeführten Schritte sind elementare Bestandteile einer explorativen Datenanalyse. Am Anfang steht immer die Beschaffung der Daten – häufig via APIs, wie in diesem Beispiel, oder aus CSV-Dateien und Datenbanken. Im nächsten Schritt geht es darum, die Daten aufzuräumen – also bspw. in eine einheitliche Form zu bringen. Anschließend folgt die eigentlich spannende Tätigkeit, nämlich das Verstehen. Das Verstehen ist meist ein iterativer Prozess aus Daten-Transformation und -Visualisierung. Sie aggregieren die Daten zum Beispiel auf ein höheres zeitliches Intervall (Tages- zu Monatswerten), plotten die Zeitreihe, identifizieren Auffälligkeiten, filtern die Daten auf diesen Zeitraum und reichern sie evtl. um zusätzliche Datenpunkte an, plotten diesen Datenausschnitt erneut und so weiter. Das Besondere an einer Datenanalyse mit R – oder auch Python etc. – ist, dass Ihnen die einzelnen Schritte nicht verloren gehen. Da sie die gesamte Datentransformation als Skript erstellen, können sie dieses Skript jederzeit erneut ausführen. Und Sie können das Skript wie eine Dokumentation der Analyse lesen. So wissen Sie auch nach ein paar Monaten noch, wie sie die Daten für ein bestimmtes Diagramm aufbereitet haben.

Und damit verabschiede ich mich von Ihnen. Happy R!

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. ¶