Mario Fischer, Patrick Lürwer

R-LEUCHTUNGEN!

Teil 2: Wie intensiv werden Ihre Seiten tatsächlich gelesen?

In den letzten Ausgaben der Website Boosting konnten Sie in einer Serie von Patrick Lürwer nachvollziehen, wie man die kostenlose Software R verwendet, was sie leistet und wie man sie nutzbringend für die eigene Arbeit einsetzen kann. R wurde ja ursprünglich für Statistik entwickelt. Wer das bisher als Entschuldigung verwendet hat, es deswegen links liegen zu lassen, dem sei versichert, dass er damit komplett falschliegt.

R kann für den Einsatz im Unternehmen, allen Bereichen voran „Online“, sehr viel mehr leisten, als statistische Berechnungen durchzuführen. Genau genommen ist es ein wirklich nützliches Helferlein bei allen Aufgaben im Umgang mit größeren Datenmengen, mit Daten, die man erst in eine gewisse Struktur bringen muss, und bei der automatischen oder halb automatischen Datenbeschaffung aus praktisch fast allen Quellen aus dem Web!

Für die interessierten Einsteiger, aber auch für alle, die nach der Serie von Patrick Lürwer „R-Blut“ geleckt haben, startete in der letzten Ausgabe 62 eine neue anwendungsorientierte Serie. Sie werden in jeder Ausgabe erfahren, wie sie ohne Programmierkenntnisse jeweils ein definiertes und in der Online-Praxis häufiger auftretendes Problem rund um das Thema Daten und Auswertungen lösen können. Und keine Sorge, die kleinen Hilfe-Tutorials nehmen Sie Schritt für Schritt an der Hand, sodass Sie auch als Neuling von der Power von R profitieren können. Was hält Sie also ab, das einfach mal auszuprobieren? Die einzelnen Schritte müssen Sie übrigens nicht im Detail verstanden haben. Die nachfolgende Auswertung kann man auch mit reinem Copy & Paste erzeugen – und das in nur wenigen Minuten.

Diesmal geht es darum, sich einen Überblick zu verschaffen, wie intensiv einzelne Webseiten eigentlich gelesen werden – bezogen auf ihren Umfang.

DER AUTOR

Mario Fischer ist Herausgeber und Chefredakteur der Website Boosting und seit der ersten Stunde des Webs von Optimierungsmöglichkeiten fasziniert. Er berät namhafte Unternehmen aller Größen und Branchen und lehrt im neu gegründeten Studiengang E-Commerce an der Hochschule für angewandte Wissenschaften in Würzburg.

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.

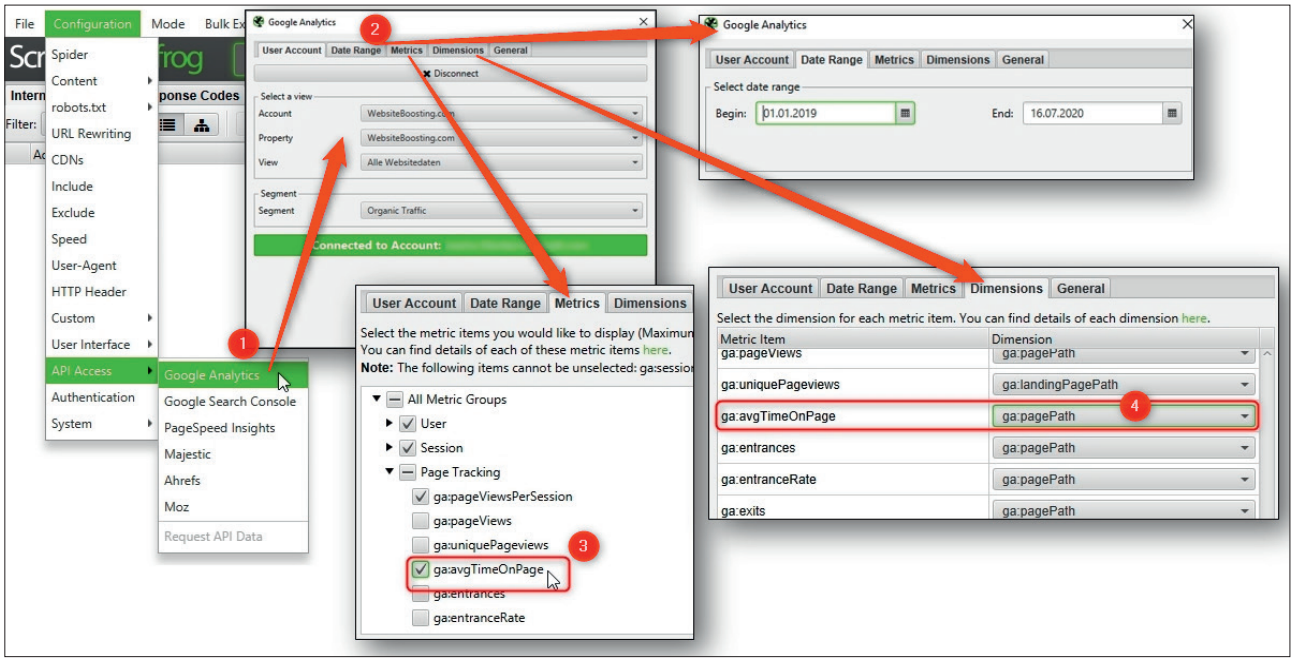

Abb. 1: Über den Screaming Frog kann man sich leicht mit Google Analytics verbinden

Die Herausforderung

Kennen Sie das Problem? Ein Blick in das Webanalyse-Tool wie z. B. Google Analytics zeigt Werte an: die „durchschnittliche Besuchsdauer: 2,3 Min.“. Man fragt sich meist ohne Hoffnung auf Antwort: Ist das jetzt viel oder wenig? Wird Seite A mit 3,4 Min. Besuchsdauer intensiver konsumiert bzw. eher gelesen als Seite B mit 2,1 Min.? Dazu erhält man von den Analytics-Programmen in der Regel keinerlei Hinweis, und wer nicht alle URLs, die dort angezeigt werden, dem Inhalt nach auswendig kennt, kann mit diesen Zahlen rein gar nichts anfangen. Sie sagen nichts aus, weil Webseiten oft erhebliche Schwankungen bei der Textmenge enthalten. Diese ist aber natürlich zum großen Teil dafür verantwortlich, wie viel Zeit jemand auf der Seite verbringt. Interesse am dargestellten Thema natürlich vorausgesetzt. Ohne Kenntnis des Umfangs einer Seite ist die Metrik „Seitenbesuchsdauer“ nichts wert.

Was man also braucht, sind im Kern drei Datenpunkte: Die URL, die durchschnittliche Seitenbesuchsdauer aus der Webanalyse und den Textumfang jeder einzelnen URL. Nur mit dem letzten Datenpunkt macht eine vergleichende Zeitbetrachtung auch wirklich Sinn.

Die Daten kann man im Prinzip mit verschiedenen SEO-Tools besorgen. Um möglichst bei einem Tool zu bleiben, soll hier wie in der letzten Ausgabe wieder der Screaming Frog verwendet werden zum einfachen Datensammeln. Natürlich könnte man auch nur mit R crawlen und über die API von Google die benötigten Metriken abholen. Aber das ist für Einsteiger deutlich aufwendiger und wirkt für den Einstieg auf den ersten Blick vielleicht auch ein wenig zu kompliziert. Und wir wollen unserem Serienprinzip hier schließlich treu bleiben: Alles sollte auch durch einfaches Copy & Paste von Code in R von Anfängern nutzbar sein.

Schritt 1: Die Daten besorgen

Die Daten besorgt man am einfachsten wie erwähnt mit Screaming Frog und insbesondere dann, wenn man Google Analytics zur Webanalyse einsetzt. Diese Schnittstellen bringt der Frog nämlich schon von Haus aus mit.

Unter „Configuration“/„API Access“ und „Google Analytics“ (Abbildung 1, Ziffer 1) verbindet man sich zunächst mit dem richtigen Account. Sofern man nur Zugriff auf eine Domain/Property hat, entfällt diese Auswahl natürlich. Google fordert die Freigabe der Daten gegenüber dem Screaming Frog an und verlangt ggf. den zugehörigen Gmail-Account nebst Passwort, sofern man die Verbindung zum ersten Mal einrichtet. Keine Sorge, diese persönlichen Accountdaten werden über ein Fenster abgefragt, das Google zur Verfügung stellt – der Screaming Frog bekommt das Passwort nicht „zu sehen“.

Im Reiter „Date Range“ (Abbildung 1, Ziffer 2) wählt man den gewünschten Zeitraum für die Messung aus. Dieser sollte möglichst lang gewählt werden, damit aus statistischer Sicht auch möglichst viele Daten zur Verfügung stehen. Die Auswahl nur

INFO

Wer R noch nicht installiert hat, kann das schnell und einfach bewerkstelligen. Unter cran.r-project.org/bin/ findet man die aktuellen Versionen für Windows (32/64 Bit), MacOS (X) und Linux. Dort im Unterverzeichnis „base“ liegen dann die installierbaren Dateien. Nach der Installation holt man sich am besten gleich unmittelbar die Software „R-Studio“, die sich als eine Art Hülle um den Kern von R als grafische Benutzeroberfläche legt und viele Tools und Eingabehilfen zur Verfügung stellt. Die Bedienung von R wird mit R-Studio fast zum Kinderspiel. Man findet sie unter: *rstudio.com/products/rstudio/download/*

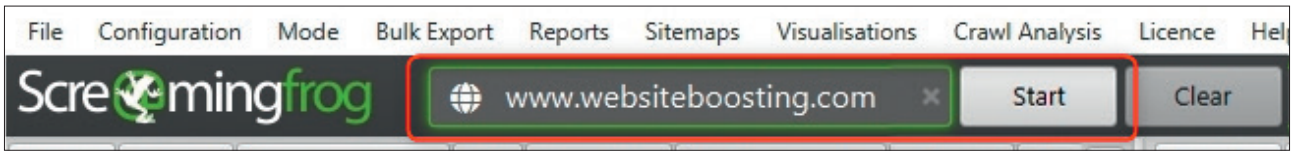

Abb. 2: Nach der Konfiguration: Domainnamen eingeben und „Start“ klicken

weniger Tage empfiehlt sich in der Regel eher nicht, weil da durchaus zufällige Verzerrungen eine zu große Rolle spielen könnten. Bei einer Domain mit sehr viel Traffic reicht dieser aber ggf. sogar dafür aus. Durchaus spannend können aber Vergleiche nach Designänderungen oder Relaunches sein. Hier zieht man sich den Datensatz einfach zweimal mit den entsprechenden Datumseinschränkungen und erzeugt später in R auch zwei Auswertungen, die man miteinander vergleichen kann.

Im Reiter „Metrics“ (Abbildung 1, Ziffer 3) muss man auf jeden Fall in der Zeile „Page Tracking“ (ggf. aufklappen) das Datenfeld „ga:avgTimeOnPage“ anhaken. Das ist in der Standardkonfiguration per Default nicht dabei, aber genau diesen Messwert benötigen wir ja.

Letzter Konfigurationspunkt: Im Reiter „Dimensions“ (Abbildung 1, Ziffer 4) stellt man noch sicher, dass für den Wert „ga:avgTimeOnPage“ die Dimension „ga:pagePath“ ausgewählt ist. Steht dort „ga:landingPagePath“, ändert man das in dem Pull-down-Menü ganz einfach ab. Denn interessant ist ja nicht nur die Verweildauer der Nutzer, die auf der jeweiligen URL eingestiegen sind – für die also die URL die Landing Page ist -, sondern wir möchten die durchschnittliche Verweildauer aller Aufrufe der Seite abfragen.

Das war es schon. Jetzt noch den Domainnamen eingeben und den Button „Start“ klicken (Abbildung 2). Je nach Größe der Domain läuft der Frog jetzt wenige Minuten oder durchaus auch länger und crawlt alle verlinkten URLs der Domain. Gleichzeitig zählt er die Anzahl der Wörter jeder Seite und holt für jede URL über die API (Datenschnittstelle) die entsprechenden Daten

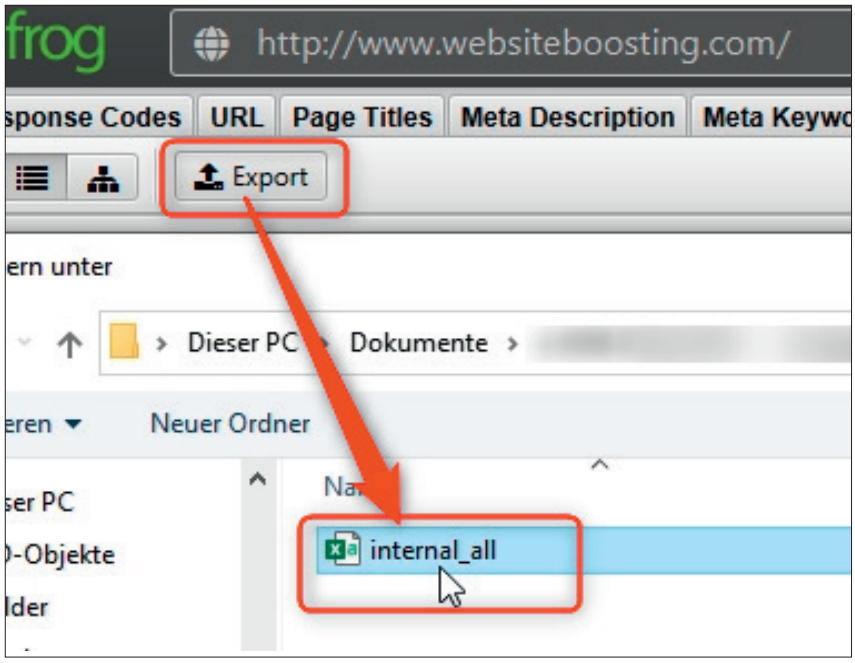

Abb. 3: Export der CSV-Datei „internal_all“

aus Google Analytics gleich mit ab. Den Status des Crawls kann man rechts unten im Programmfenster live mitverfolgen.

Wenn der Crawl durchgelaufen bzw. beendet ist, speichert man die Ergebnisse einfach über den Button „Export“ und mit dem vorgeschlagenen Dateinamen „internal_all.csv“ als CSV-Datei ab. Am besten wählt man als Speicherort das Arbeitsverzeichnis von R, dann kann man in R direkt und ohne Verzeichniswechsel darauf zugreifen. Wie das geht, können Sie bei Bedarf noch mal in der letzten Ausgabe nachlesen.

Ab in R(-Studio) und los gehts

Wie im ersten Teil in der letzten Ausgabe ja bereits erwähnt, erhält R seine große Stärke durch die vielen Funktionserweiterungen, Bibliotheken bzw. Librarys genannt. Jede dieser Librarys muss man bei der allerersten Verwendung in R nachladen bzw. installieren. Das ist allerdings ein einmaliger Vorgang. R merkt sich, welche Librarys man installiert hat, und fortan muss man sie nach dem Aufruf des Programms bei Bedarf nur starten mit dem Befehl library(name_der_library).

INFO

Damit Sie den Code nicht einzeln abtippen müssen, können Sie ihn unter *http://einfach.st/rcode3* als Textdatei downloaden und sich entweder alles auf einmal oder auch Befehl für Befehl herauskopieren.

Die Library „tidyverse“ wurde in Teil 1 dieser Serie schon verwendet. Wer diese ausprobiert hat, in dessen R-Paket ist sie also schon enthalten. Wer jetzt erst einsteigt, muss sie mittels des Befehls „install.packages“ noch erstmalig installieren. Dazu gibt man beide oder nur die zweite Zeile mit der neuen Library „lubridate“ direkt in R ein:

```
» install.packages(„tidyverse“)
» install.packages(„lubridate“)
```

Keine Sorge bei eventuellen Meldungen, wie etwa:

```
Warnmeldung: Paket 'lubridate' wurde unter R Version 4.0.2 erstellt
```

So etwas kommt häufiger vor, weil sowohl R als auch die Erweiterungen ständigen Updates unterliegen. Bei der nächsten Aktualisierung verschwindet der Meldespuk meist wieder.

Alternativ zur Kommandozeileneingabe kann man in R-Studio (siehe Kasten) unter „Tools“ und „Install Packages“ auch mit der Maus menügeführt arbeiten. Fängt man an, im entsprechenden Feld (Abbildung 4, Ziffer 1) den Namen zu tippen, erscheint bereits ein Vorschlag, den man einfach übernehmen kann.

Jetzt müssen die beiden eben oder schon vorher installierten Librarys tatsächlich noch aufgerufen bzw. in den Arbeitsspeicher von R geladen werden. Das geschieht mittels des einfachen Befehls „library“ gefolgt vom Namen der Erweiterung in Klammern. In unserem Fall also:

```
» library(tidyverse)
» librady(lubridate)
```

Jetzt können wir die mittels Screaming Frog erzeugte Datei „internal_all.csv“ in R einlesen und den Daten z. B. den Namen „crawl“ zuweisen, um sie später einfach unter diesem Namen schnell ansprechen zu können.

```
crawl <- read_csv(„internal_all.csv“)
%>%
    janitor::clean_names() # Spaltennamen normalisieren
```

Alles, was nach einem „#“ steht, wird von R ignoriert und dient nur der Erklärung/Dokumentation.

Möchten Sie sich die eben eingelesene Datei einmal kurz ansehen? Das geht ganz einfach mit dem Befehl „view“, gefolgt von dem Namen des Datensatzes in Klammern, dem in diesem Beispiel „crawl“ genannt wurde. Also:

```
view(crawl)
```

Über der Befehlskonsole erscheint jetzt ein Fenster, das Struktur und Daten der Tabelle zeigt (Abbildung 5). Über die Pfeile bei „Cols“ (für Spalten) kann man, wie Ziffer 1 zeigt, entsprechend weiterblättern und gelangt sowohl zu der Spalte „word-count“ (Ziffer 2) als auch weiter hinten zu „ga:avg_time_on_page“ (Ziffer 3), die wir im Frog ja extra mit aktiviert hatten. Mit diesen drei Werten arbeitet das Skript: URL, die Anzahl Wörter und die durchschnittliche Besuchsdauer der jeweiligen URL.

Jetzt können die Daten entsprechend aufbereitet werden:

```
crawl_html_with_time_on_page <- crawl %>%
 filter(
   str_detect(content, „html“),
   !is.na(ga_avg_time_on_page),
# Nur die html-URLs behalten, für die es auch eine positive Lesedauer in GA gibt
 ) %>%
 select(address, word_count, ga_avg_time_on_page) %>%
# Nur die benötigten Spalten behalten, alle anderen braucht man hier nicht
 mutate(
   actual_reading_duration_in_sec = hms(ga_avg_time_on_page),
# Die Zeitangabe mittels der Library lubridate::hms (Stunde Minute Sekunde) in eine zeitliche Periode umwandeln
```

TIPP

Ab und zu erscheinen in R bzw. R-Studio „Warnmeldungen“. Diese können sie in der Regel getrost ignorieren, weil es wirklich nur Hinweise auf z. B. abweichende Versionen von R und einer Erweiterung sind. Solange Sie keine Fehlermeldungen erhalten, läuft alles weiter.

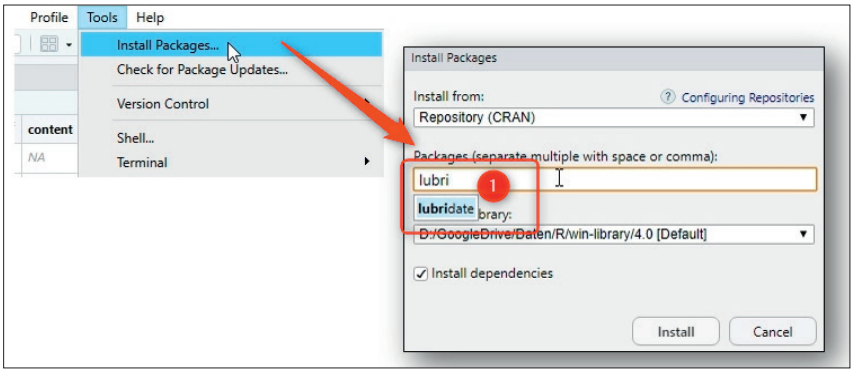

Abb. 4: Auch das geht: Erweiterungen per Menü installieren

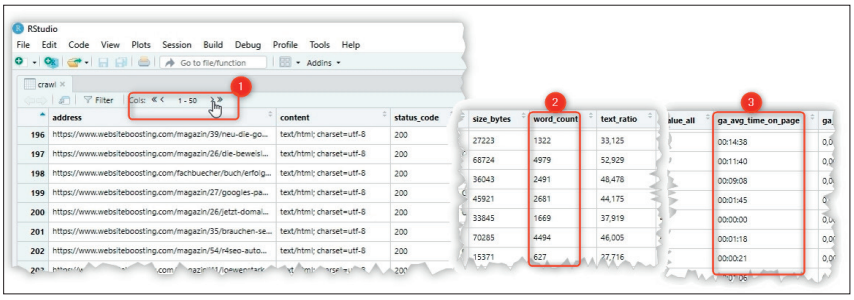

Abb. 5: View(crawl) zeigt die Struktur der CSV-Datei

```
  actual_reading_duration_in_sec = period_to_seconds(actual_reading_duration_in_sec),
# Die Perioden von Stunden, Minuten und Sekunden in Sekunden umrechnen
  expected_reading_duration_in_sec = round(word_count / 220 * 60, 0),
# Die erwartete Lesedauer in Sekunden bei den üblichen 220 Wörtern Lesegeschwindigkeit pro Minute
# Bei Bedarf einfach die Zahl 220 in der obigen Formel anpassen
  diff_reading_duration_in_sec = actual_reading_duration_in_sec - expected_reading_duration_in_sec
 ) %>%
 arrange(desc(diff_reading_duration_in_sec))
# Absteigend nach der Differenz sortieren, um die wichtigsten URLs, die man prüfen sollte, am Anfang der Tabelle zu positionieren.
```

Das war schon alles, was die Datenaufbereitung angeht. Wie gesagt, mit Copy & Paste (das Skript ist per Download verfügbar) in das Konsolenfeld einfügen, Return, fertig.

Punktegrafik erzeugen

Die die eben aufbereiteten Daten lassen sich jetzt mit den nachfolgenden beiden etwas unterschiedlichen Skriptteilen in zwei Abbildungsarten umwandeln. Eine Punktegrafik (Abbildung 6) erzeugt man mit dem folgenden Befehlsset:

```
ggplot(crawl_html_with_time_on_page, aes(actual_reading_duration_in_sec, expected_reading_duration_in_sec)) +
 geom_point() +
 geom_abline() +
 labs(title = „Erwartete Lesezeit vs. tatsächliche Lesezeit“,
     subtitle = „Punkte oberhalb der Linie zeigen eine kürzere Lesezeit als erwartet an“,
     x = „Tatsächliche Lesezeit in Sekunden“,
     y = „Erwartete Lesezeit in Sekunden“)
```

Die Überschrift und die Achsenbezeichnung (der Text in Anführungszeichen) lassen sich natürlich beliebig und nach eigenem Gusto verändern.

In dieser Darstellung werden die tatsächlichen Lesezeiten auf der x-Achse und die erwarteten Lesezeiten auf der y-Achse abgetragen. Stimmten beide Werte überein, lägen die Punkte exakt auf der eingezeichneten Geraden. Erwartungsgemäß tritt diese perfekte Korrelation sehr selten ein. Punkte, die oberhalb der Linie liegen, zeigen daher URLs an, deren tatsächliche Lesezeit geringer ist als die erwartete.

Analog dazu weisen Punkte unterhalb der Linie eine längere Lesezeit auf als erwartet. Von Interesse sind nun solche Punkte, die besonders stark nach oben von der Linie abweichen, da diese die größten Differenzen aus erwarteter und tatsächlicher Lesedauer aufweisen. Anders herum sind aber natürlich auch solche Seiten spannend, auf denen Nutzer deutlich länger verweilen als erwartet (die Punkte unterhalb der Linie). Was machen diese Seiten besser als andere? Kann ich Eigenschaften von diesen auf die schlechter performenden Seiten übertragen, um die Verweildauer dort zu steigern?

Das lässt sich aber bei Bedarf auch anders darstellen, z. B. über eine Balkengrafik.

Wie stark sind die Abweichungen: Eine Balkengrafik erzeugen

Möchte man wissen, wie stark sich die Differenzen verteilen, wählt man am besten eine Übersichtsdarstellung mit einer Balkengrafik. Diese Darstellung gibt einen Eindruck darüber, wie stark die Abweichungen sind. Liegt die Verweildauer eher unterhalb oder er oberhalb der erwarteten Zeit – sprich: liegen mehr URLs links oder rechts der eingezogenen Linie? Im Idealfall würde die meisten URLs sehr nah an der Linie positioniert sein.

Den Code muss man dafür nur minimal verändern:

```
ggplot(crawl_html_with_time_on_page, aes(diff_reading_duration_in_sec)) +
 geom_histogram() +
 geom_vline(xintercept = 0) +
 labs(title = „Verteilung der Differenzen zw. erwarteter und tatsächlicher Lesezeit“,
     subtitle = „Negative Werte zeigen eine kürzere, tatsächliche Lesezeit an als erwartet“,
     x = „Differenz von erwarteter und tatsächlicher Lesezeit in Sekunden“,
     y = „Anzahl an URLs“)
```

Auch hier gilt natürlich, dass man die Überschriften und Achsenbezeichnungen im Code entsprechend verändern kann.

Interpretation und Grenzen: Boilerplate und Mittelwerte

Aufmerksame Leserinnen und Leser werden wahrscheinlich sofort Folgendes bemerkt haben: Die reine Anzahl an Wörtern auf einer Webseite zu zählen und das als Maßstab für eine theoretische Maximallesedauer zu verwenden, ist ein klein wenig schief. Das ist richtig. Besucher werden sicherlich nur Zeit für den sog. „Primary Content“ aufwenden, also für den Textteil, der auf den Seiten einzigartig ist.

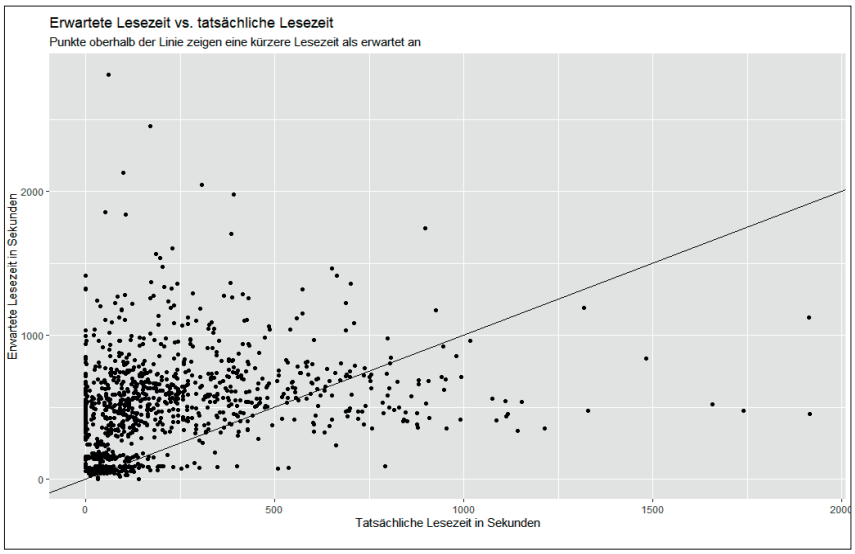

Abb. 6: Punktedarstellung: Die erwartete Lesezeit vs. der tatsächlich in Google Analytics gemessenen

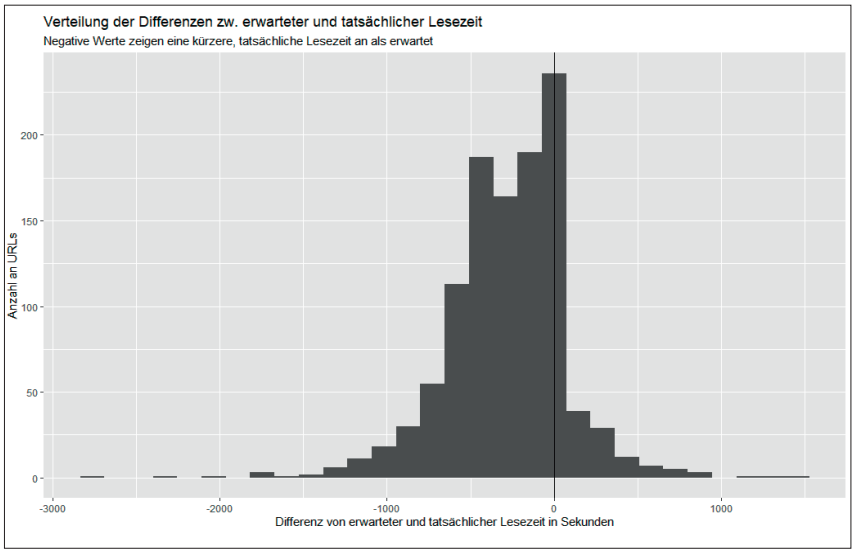

Abb. 7: Die Verteilung der Differenzen zwischen erwarteter und tatsächlicher Lesezeit

Bei einem Shop ist das in der Produktdarstellung sicher die Produktbeschreibung, auf einem Blog der eigentliche Blogbeitrag. Der restliche umfließende Text in Sidebars, Footer, Header oder sonstigen Bereichen mit gleichem Text wird wahrscheinlich eher ignoriert. Man spricht in diesem Fall von der sog. „Boilerplate“. Diese Texte automatisch (!) einfach zu ignorieren bzw. nicht mitzuzählen, ist leider nicht trivial. Ganz im Gegenteil.

Wer die Website Boosting regelmäßig aufmerksam verfolgt und Dinge aktiv ausprobiert, kann das aber relativ leicht manuell durchführen. Eine ausführliche Anleitung dazu würde allerdings den Rahmen dieses Beitrags sprengen. Für die Experten sei daher nur erwähnt, dass sie dafür den XPath des Primary Contents im DOM ermitteln müssen und eben nur diese dort verorteten Wörter zählen lassen. Über die SEO-Tools für Excel von Nils Bosma oder den Screaming Frog geht das recht einfach, indem man den Inhalt des Primary Contents per XPath holt und die Wörter darin zählen lässt. Die SEO-Tools für Excel stellen dafür die Funktion „=WordCount“ zur Verfügung – nicht zu verwechseln mit „=CountWords“. Jetzt kann man entweder nur die Wortanzahl dieses Bereichs für die Berechnung verwenden oder beides voneinander abziehen, also alle Wörter minus die im Primary Content. Diese Rechnung ergibt dann die Boilerplate, also die Anzahl „(fast) überall gleicher Wörter“. Ermittelt man diesen Wert über eine repräsentative Anzahl an URLs, hat man einen statistisch recht stabilen Wert der Boilerplate, den man später pauschal einfach abziehen kann von der „Wörter-Vollzählung“.

Und noch eine weitere Einschränkung sollte man im Kopf behalten. Natürlich lesen nicht alle Besucher einer Seite immer den kompletten Text im Primary Content durch. Ein Teil ist hier falsch, ein Teil interessiert sich nur für Details und bei einigen schellt vielleicht das Telefon beim Lesen, sodass die Besuchsdauer als länger ausgewiesen wird, als es Sinn macht. Wenn Sie die Zeitspanne (Abbildung 1, Ziffer 2) genügend groß gewählt haben, egalisieren sich einige dieser Effekte durch die größere Anzahl an Messpunkten. Trotzdem wird der tatsächliche Messwert in der Regel unter dem theoretisch erwarteten Maximalwert bleiben. Hier gilt es, sich für die eigene Domain und deren spezifischen Content und die entsprechende Zielgruppe einen Eindruck über die normale Abweichung zu verschaffen. Liegen zwischen Maximalerwartung und tatsächlicher Zeit z. B. im Mittel etwa 30 Prozent, dann verwendet man dies am besten als Normalabweichung und justiert damit die Gedanken neu. Alles, was deutlich über oder unter diesen 30 Prozent liegt, sollte man sich dann genauer ansehen. Was hält die Besucher ab, Text A auf Seite X im Schnitt (!) deutlich weniger zu konsumieren als Text B auf Seite Y? Bestückt mit diesem konkreten Analyseauftrag erkennt man meist schnell und zielgerichtet, was auf Seite X anders ist als auf Y. Man sucht jetzt aktiv. Das macht durchaus einen Unterschied und in der Praxis wird man damit tatsächlich auch oft fündig.

CODE

Den kompletten Code zum direkten Einkopieren in R erhalten Sie unter » http://einfach.st/rcode3

```
library(tidyverse)
library(lubridate)

# Crawl einlesen
crawl <- read_csv(„internal_all.csv“) %>%
  janitor::clean_names() # Spaltennamen normalisieren

#Daten aufbereiten
crawl_html_with_time_on_page <- crawl %>%
  filter(
    str_detect(content, „html“),
    !is.na(ga_avg_time_on_page),
# Nur die html-URLs behalten, für die es auch eine positive Lesedauer in GA gibt
  ) %>%
  select(address, word_count, ga_avg_time_on_page) %>%
# Nur die benötigten Spalten behalten, alle anderen braucht man hier nicht
  mutate(
    actual_reading_duration_in_sec = hms(ga_avg_time_on_page),
# Die Zeitangabe mittels der Library lubridate::hms() (Stunde Minute Sekunde) in eine zeitliche Periode umwandeln,
# um im nächste Schritte einfacher die Dauer in Sekunden erhalten zu können
    actual_reading_duration_in_sec = period_to_seconds(actual_reading_duration_in_sec),
# Die Perioden von Stunden, Minuten und Sekunden in Sekunden umrechnen
    expected_reading_duration_in_sec = round(word_count / 220 * 60, 0),
# Die erwartete Lesedauer in Sekunden bei den üblichen 220 Wörtern Lesegeschwindigkeit pro Minute
# Bei Bedarf einfach die Zahl 220 in der obigen Formel anpassen
    diff_reading_duration_in_sec = actual_reading_duration_in_sec - expected_reading_duration_in_sec
# Berechnung der Differenz von erwarteter und tatsächlicher Lesedauer
  ) %>%
  arrange(desc(diff_reading_duration_in_sec))
# Absteigend nach der Differenz sortieren, um die wichtigsten URLs, die man prüfen sollte,
# am Anfang der Tabelle zu positionieren.

############################
# Entweder eine Punktegrafik erzeugen
# Achtung: Diesen oder alternativ den Abschnitt unten auswählen, die nicht benötigten Zeilen einfach löschen
############################
ggplot(crawl_html_with_time_on_page, aes(actual_reading_duration_in_sec, expected_reading_duration_in_sec)) +
  geom_point() +
  geom_abline() +
  labs(title = „Erwartete Lesezeit vs. tatsächliche Lesezeit“,
       subtitle = „Punkte oberhalb der Linie zeigen eine kürzere Lesezeit als erwartet an“,
       x = „Tatsächliche Lesezeit in Sekunden“,
       y = „Erwartete Lesezeit in Sekunden“)

############################
# Oder eine Balkengrafik erzeugen
# Achtung: Diesen oder alternativ den Abschnitt oben auswählen, die nicht benötigten Zeilen einfach löschen
############################
ggplot(crawl_html_with_time_on_page, aes(diff_reading_duration_in_sec)) +
  geom_histogram() +
  geom_vline(xintercept = 0) +
  labs(title = „Verteilung der Differenzen zw. erwarteter und tatsächlicher Lesezeit“,
       subtitle = „Negative Werte zeigen eine kürzere, tatsächliche Lesezeit an als erwartet“,
       x = „Differenz von erwarteter und tatsächlicher Lesezeit in Sekunden“,
       y = „Anzahl an URLs“)
```

TIPP

In den Ausgaben 52 und 53 finden Sie bei Bedarf noch weitere Tipps und Anleitungen für den Screaming Frog.
Zur Auswahl für die kostenlosen Ausgaben (derzeit Ausgabe 1–54) kommen Sie am besten über *www.websiteboosting.com/magazin.html*.