The online encyclopedia Wikipedia, together with its sibling, the collaboratively edited knowledge base Wikidata, provides incredibly rich yet largely untapped sources for political research.

After reading this blog post and engaging with the applied exercises, readers should:

* be able to collect Wikipedia data and Wikidata items using **R**
* be able to conduct explorative analyses of Wikipedia data using **R**
* have a basic intuition of the potentials and limitations of using Wikipedia data in research projects

**Wikipedia for Political Research**

Wikipedia harbors numerous types of data. These include both article contents as well as meta information such as pageviews, clickstreams, links and backlinks, or edits and revision histories. Additionally, Wikipedia’s sibling, the collaboratively edited document-oriented data base [Wikidata](https://en.wikipedia.org/wiki/Wikidata), provides access to over 58 million data items (as of July 2019). Given the broad collection of articles on politicians and institutions from all over the world, Wikipedia offers tremendous potential for (comparative) political research.

In what follows, we will introduce the functionalities of various **R** packages, including [WikipediR](https://cran.r-project.org/web/packages/WikipediR/index.html), [WikidataR](https://cran.r-project.org/web/packages/WikidataR/index.html), and [pageviews](https://cran.rstudio.com/web/packages/pageviews/index.html). In doing so, we will showcase how to connect to Wikipedia and Wikidata APIs, how to efficiently access and parse content, and how to process the retrieved data in order to address various questions of substantive interest. We will also provide an overview of the legislatoR package, a fully relational individual-level data package that comprises political, sociodemographic, and Wikipedia-related data on elected politicians from various consolidated democracies.

Code: **R** packages used in this tutorial

## Packages

pkgs <- c(

"devtools",

"ggnetwork",

"igraph",

"intergraph",

"tidyverse",

"rvest",

"devtools",

"magrittr",

"plotly",

"RColorBrewer",

"colorspace",

"lubridate",

"networkD3",

"pageviews",

"readr",

"wikipediatrend",

"WikipediR",

"WikidataR"

)

## Install uninstalled packages

lapply(pkgs[!(pkgs %in% installed.packages())], install.packages)

## Load all packages to library

lapply(pkgs, library, character.only = TRUE)

## legislatoR

devtools::install\_github("saschagobel/legislatoR")

library(legislatoR)

**Collecting and Analyzing Wikipedia Data**

**Application 1: Using Pageviews to Measure Public Attention**

The code chunk below demonstrates how to collect and graphically display pageviews data using the pageviews package. We use the command article\_pageviews(), where the argument project = "en.wikipedia" specifies that we want to collect pageviews of article = "Donald Trump" from the English Wikipedia. We can only restrict our query to a given language edition; it is not possible to limit queries to pageviews from a specific country. We also specify the argument user\_type = "user", which ensures that we exclude pageviews generated by bots and spiders. Finally, start and end define the period on which we want to collect pageviews data: July 2015 to May 2017. We proceed analogously for article = "Hillary Clinton".

Code: Pageviews Data Collection

# get pageviews

trump\_views <-

article\_pageviews(

project = "en.wikipedia",

article = "Donald Trump",

user\_type = "user",

start = "2015070100",

end = "2017050100"

)

head(trump\_views)

clinton\_views <-

article\_pageviews(

project = "en.wikipedia",

article = "Hillary Clinton",

user\_type = "user",

start = "2015070100",

end = "2017050100"

)

This query allows us to retrieve the pageviews for both Trump’s and Clinton’s Wikipedia articles by date. We can then plot the frequencies of pageviews over time to identify trends in search behaviour. As we can see, the data indicate that Trump attracted considerably more attention than Clinton throughout the 2016 election campaign.

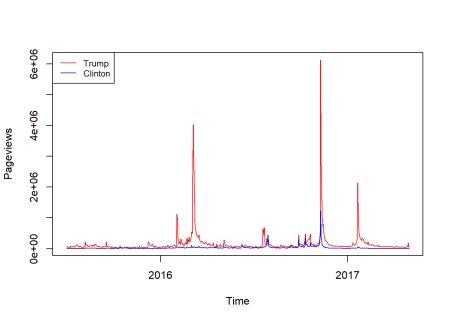
Code: Plotting Pageviews

# Plot pageviews

plot(ymd(trump\_views$date), trump\_views$views, col = "red", type = "l", xlab="Time", ylab="Pageviews")

lines(ymd(clinton\_views$date), clinton\_views$views, col = "blue")

legend("topleft", legend=c("Trump","Clinton"), cex=.8,col=c("red","blue"), lty=1)



**Application 2: Using Article Links to Create a Network Graph of German MPs**

The WikipediR package is a wrapper for the MediaWiki API that can be used to retrieve page contents as well as metadata for articles and categories, e.g. information about users or page edit histories. The functionality of the package includes:

* page\_content(): Retrieve current article versions (HTML and wikitext as possible output formats)
* revision\_content(): Retrieve older versions of the article; this also includes metadata about the revision history
* page\_links(): Retrieve outgoing links from the page’s content (which Wikipedia articles does the page link to?)
* page\_backlinks(): Retrieve incoming links (which Wikipedia articles link to the page?)
* page\_external\_links(): Retrieve outgoing links to external sites
* page\_info(): Page metadata
* categories\_in\_page(): What categories is a given page in?
* pages\_in\_category(): What pages are in a given category?

For our application, we use the page\_links() function to extract mutual referrals between the articles on members of the 2017-2021 German Bundestag. We can then use this information to create a network graph of current German MPs. First, we use the legislatoR package to retrieve a list of all German MPs of the 2017-2021 German Bundestag, including information on their page IDs and page titles in the German Wikipedia. Using this information, we then extract all page\_links() in every MP’s Wikipedia articles. The third step identifies the subset of links for every MP that link to the Wikipedia article of another current MP.

This allows us to finally plot an interactive network using the forceNetwork() command from the networkD3 package. We can save the interactive network graph as an HTML widget, which is included below.

Code: Creating an Interactive Network Graph Based on Article Links

## step 1: get info about legislators

dat <- semi\_join(

x = get\_core(legislature = "deu"),

y = filter(get\_political(legislature = "deu"), session == 19),

by = "pageid"

)

## step 2: get page links (max 500 links)

if (!file.exists("studying-politics-wikipedia/data/wikipediR/mdb\_links\_list.RData")) {

links\_list <- list()

for (i in 1:nrow(dat)) {

links <-

page\_links(

"de",

"wikipedia",

page = dat$wikititle[i],

clean\_response = TRUE,

limit = 500,

namespaces = 0

)

links\_list[[i]] <- lapply(links[[1]]$links, "[", 2) %>% unlist

}

save(links\_list, file = "studying-politics-wikipedia/data/wikipediR/mdb\_links\_list.RData")

} else{

load("studying-politics-wikipedia/data/wikipediR/mdb\_links\_list.RData")

}

## step 3: identify links between MPs

# loop preparation

connections <- data.frame(from = NULL, to = NULL)

# loop

for (i in seq\_along(dat$wikititle)) {

links\_in\_pslinks <-

seq\_along(dat$wikititle)[str\_replace\_all(dat$wikititle, "\_", " ") %in%

links\_list[[i]]]

links\_in\_pslinks <- links\_in\_pslinks[links\_in\_pslinks != i]

connections <-

rbind(connections,

data.frame(

from = rep(i - 1, length(links\_in\_pslinks)), # -1 for zero-indexing

to = links\_in\_pslinks - 1 # here too

)

)

}

# results

names(connections) <- c("from", "to")

# make symmetrical

connections <- rbind(connections,

data.frame(from = connections$to,

to = connections$from))

connections <- connections[!duplicated(connections), ]

## step 4: visualize connections

connections$value <- 1

nodesDF <- data.frame(name = dat$name, group = 1)

network\_out <-

forceNetwork(

Links = connections,

Nodes = nodesDF,

Source = "from",

Target = "to",

Value = "value",

NodeID = "name",

Group = "group",

zoom = TRUE,

opacityNoHover = 3,

height = 360,

width = 636

)

Using the underlying connections data set, we can also identify which members of the German parliament share the most nodes with others. Perhaps unsurprisingly, we see the German chancellor Angela Merkel on top of the list, followed by a list of current and former federal ministers and (deputy) party leaders.

Code: Top 10 MPs by Connections Counts

nodesDF$id <- as.numeric(rownames(nodesDF)) - 1

connections\_df <-

merge(connections,

nodesDF,

by.x = "to",

by.y = "id",

all = TRUE)

to\_count\_df <- count(connections\_df, name)

arrange(to\_count\_df, desc(n))

## # A tibble: 712 x 2

## name n

##

## 1 Angela Merkel 59

## 2 Andrea Nahles 40

## 3 Heiko Maas 38

## 4 Katarina Barley 38

## 5 Peter Altmaier 38

## 6 Wolfgang Schäuble 38

## 7 Wolfgang Kubicki 37

## 8 Hans-Peter Friedrich 34

## 9 Hermann Gröhe 34

## 10 Ursula von der Leyen 33

## # ... with 702 more rows

**Application 3: Using Clickstream Data to Analyze Referral Patterns**

In the example below, we focus on two party groups of the 8th (2014-2019) European Parliament: the euroskeptic EFDD (Europe of Freedom and Direct Democracy) and the far right ENF (Europe of Nations and Freedom). In particular, we are interested in clickstreams between the two party groups, between the party groups and their member parties, and between the individual member parties.

Toward this end, we download clickstream data from the English Wikipedia for May 2019, the month of the 2019 European Parliament elections. We identify 19 articles of interest and store them in the object articles. Having retrieved and extracted the clickstream data from May 2019, we import the TSV file into **R** using read.table(). Lastly, we subset the data to observations that involve referrals between all available article-pairs of the 19 articles.

Code: Collecting and Processing Clickstream Data

# retrieve article titles of interest

enf <- "Europe\_of\_Nations\_and\_Freedom"

efdd <- "Europe\_of\_Freedom\_and\_Direct\_Democracy"

enf\_parties <- c(

"Freedom\_Party\_of\_Austria",

"Vlaams\_Belang",

"National\_Rally\_(France)",

"The\_Blue\_Party\_(Germany)",

"Lega\_Nord",

"Party\_for\_Freedom",

"Congress\_of\_the\_New\_Right"

)

efdd\_parties <- c(

"Svobodní",

"The\_Patriots\_(France)",

"Debout\_la\_France",

"Alternative\_for\_Germany",

"Five\_Star\_Movement",

"Order\_and\_Justice",

"Liberty\_(Poland)",

"Brexit\_Party",

"Social\_Democratic\_Party\_(UK,\_1990–present)",

"Libertarian\_Party\_(UK)"

)

articles <- c(enf, efdd, enf\_parties, efdd\_parties)

# import raw clickstream data

cs <-

read.table(

"clickstream-enwiki-2019-05.tsv",

header = FALSE,

col.names = c("prev", "curr", "type", "n"),

fill = TRUE,

stringsAsFactors = FALSE

)

cs$n <- as.integer(cs$n)

# subset

cs <- subset(cs, prev %in% articles & curr %in% articles)

Next, we aim to analyze aggregate referral patterns. We first assign both previous (prev) and current (curr) articles to one of four categories: Articles on the EFDD and ENF parliamentary groups (one article each), articles on ENF member parties (7 articles), and articles on EFDD member parties (10 articles). We then summarize the data to obtain aggregate referral counts between all category pairs. Lastly, we display these in an interactive Sankey diagram using the [plotly](https://cran.r-project.org/web/packages/plotly/index.html) package.

Code: Analyzing and Plotting Clickstream Data

# assign categories

cs <- cs %>%

mutate(

curr\_cat = ifelse(

curr == enf,

"ENF Group",

ifelse(

curr == efdd,

"EFDD Group",

ifelse(curr %in% enf\_parties, "ENF Parties",

"EFDD Parties")

)

),

prev\_cat = ifelse(

prev == enf,

"ENF Group",

ifelse(

prev == efdd,

"EFDD Group",

ifelse(prev %in% enf\_parties, "ENF Parties",

"EFDD Parties")

)

)

)

# summarize data

cs\_sum <- cs %>%

group\_by(curr\_cat, prev\_cat) %>%

summarize(n = sum(n)) %>%

arrange(prev\_cat)

# Sankey diagram using plotly

labels <- c(unique(cs\_sum$prev\_cat), unique(cs\_sum$curr\_cat))

colors <- ifelse(grepl("EFDD", labels), "#24B9B9", "#2B3856")

sankey\_plot <- plot\_ly(

type = "sankey",

orientation = "h",

node = list(

label = labels,

color = colors,

pad = 15,

thickness = 15,

line = list(color = "black",

width = 0.5)

),

link = list(

source = as.numeric(as.factor(cs\_sum$prev\_cat)) - 1L,

target = as.numeric(as.factor(cs\_sum$curr\_cat)) + 3L,

value = cs\_sum$n

),

height = 340,

width = 600

) %>%

layout(font = list(size = 10))

The diagram shows that in our data, clickstream dyads involving the articles on the EFDD and ENF parliamentary groups are much more numerous than dyads involving the member parties.

Moving beyond clickstreams between the four categories, we can also visualize the full network structure of *all* individual articles in our data. The code below starts with some preparatory data management and then uses the igraph package to create the network and to customize its graphical display.

In the final section of the code, we use the intergraph, ggnetwork and plotly packages to produce an interactive HTML5-compatible figure for this blog post. On your own machine, you may skip this section and simply use plot.igraph() on cs\_net without transforming the object to a ggplot friendly format.

Code: Interactive Network Graph

# construct edges

cs\_edge <-

cs %>%

group\_by(prev, curr, prev\_cat, curr\_cat) %>%

dplyr::summarise(weight = sum(n)) %>%

arrange(curr)

# get list of unique articles to construct as nodes

cs\_node <-

gather(cs\_edge,

`prev`,

`curr`,

key = "where",

value = "article") %>%

ungroup() %>%

select(article) %>%

distinct(article)

names(cs\_node) <- c("node")

cs\_node$category <-

ifelse(cs\_node$node == enf,

"ENF Group",

ifelse(

cs\_node$node == efdd,

"EFDD Group",

ifelse(cs\_node$node %in% enf\_parties, "ENF Parties",

"EFDD Parties")

)

)

# generate graph

set.seed(3)

cs\_net <-

graph.data.frame(cs\_edge,

vertices = cs\_node,

directed = F)

cs\_net <-

igraph::simplify(cs\_net, remove.multiple = T, remove.loops = T)

# generate colors based on category

V(cs\_net)$color <-

ifelse(grepl("EFDD", V(cs\_net)$category), "#24B9B9", "#2B3856")

# compute node degrees (#links) and use that to set node size

deg <- igraph::degree(cs\_net, mode = "all")

V(cs\_net)$size <- deg / 10

# set labels

V(cs\_net)$label <- NA

V(cs\_net)$label.cex = 0.5

V(cs\_net)$label = ifelse(igraph::degree(cs\_net) > 5, V(cs\_net)$label, NA)

cs\_hc\_labels <- as.vector(cs\_node$node)

# set edge width based on weight

E(cs\_net)$width <- log(E(cs\_net)$weight) / 5

E(cs\_net)$edge.color <- "gray80"

# transform the network to a ggplot friendly format

# (required to generate interactive graph embedded in blog post)

gg\_cs\_net <-

ggnetwork(

cs\_net,

layout = "fruchtermanreingold",

weights = "weight",

niter = 50000,

arrow.gap = 0

)

cs\_plot <- ggplot(gg\_cs\_net, aes(x = x, y = y, xend = xend, yend = yend)) +

geom\_edges(aes(color = edge.color), size = 0.4, alpha = 0.25) +

geom\_nodes(aes(color = color, size = size)) +

geom\_nodetext(aes(color = color, label = vertex.names, cex = 0.6)) +

guides(size=FALSE) +

theme\_blank() +

theme(legend.position = "none")

In the graph above, node diameters indicate the relative weight (total counts) of each article; node colors indicate whether an articles belongs to the EFDD or ENF. We see that members of the same party group tend to share more clickstreams. The Alternative for Germany (AfD), however, shares many connections with members of the ENF. This makes sense when we consider that the AfD has sought closer cooperation with numerous ENF member parties since 2016 with whom it eventually formed the new far right EP group, Identity and Democracy, in June 2019.

Lastly, a word of caution: One should keep in mind that clickstream counts heaviliy depend on how prominently (if at all) outgoing links are placed in a given Wikipedia article. Furthermore, raw counts from an isolated subset of clickstreams (as in the examples above) give no information on the relative importance of a given referral pattern relative to all outgoing referrals of a given article. Users should thus ensure that they use clickstream data in a way that adequately addresses their substantive inquiries.

**Collecting Data via Wikidata Queries**

Wikidata is a collaboratively edited knowledge base with [over 58 million entries as of July 2019](https://www.wikidata.org/wiki/Wikidata:Statistics). It harbors various types of database items, including text, numerical quantities, coordinates, and images. There are no language editions, but individual entries can have values in different languages.

Wikidata allows users to submit queries using SPARQL, a query language for data stored in RDF (Resource Description Framework) format. While basic queries can be used to answer mundane questions (e.g. *“what is the capital city of every member of the European Union, and how many inhabitants live there?”*), a targeted combination of related queries can be used for systematic data collection.

Instead of submitting explicit SPARQL queries, the example below uses the WikidataR package to combine various queries in order to collect data on the candidates in the [2019 leadership election of the UK Conservative Party](https://en.wikipedia.org/wiki/2019_Conservative_Party_(UK)_leadership_election). Suppose we want to retrieve the following information on each candidate:

* name
* sex
* date of birth
* political experience
* education
* official website URL
* Twitter accout
* Facebook account

In Wikidata, entries are stored as *items* with a unique item ID that starts with “Q”. For instance, the item [2019 Conservative Party (UK) leadership election](https://www.wikidata.org/wiki/Q30325756) is stored as “Q30325756”. Items are characterized by a number of *statements* or *claims*. Claims start with “P” and detail an item’s properties. For instance, the claim “candidate” is stored as “P726”. Claims have values, which may once again be items. For example, the values of claim “P726” (candidate) of item “Q30325756” (2019 UK Conservative Party leadership election) are 10 items: one entry for each of the 10 candidates running in the leadership election. Take, for example, winning candidate Boris Johnson, who is listed as a candidate under claim “P726”. In turn, the entry on Boris Johnson is stored as item “Q180589”. This item is characterized by numerous claims, including “P1559” (name in native language), “P21” (sex or gender), “P569” (date of birth), “P39” (positions held), “P69” (educated at), “P856” (official website), “P2002” (Twitter username), and “P2013” (Facebook ID).

In order to collect the data for all 10 candidates in the 2019 Conservative Party leadership election, the code chunk below implements the following steps:

1. We retrieve *item* “Q30325756”, i.e., the entry for 2019 Conservative Party (UK) leadership election
2. We extract *claims* “P726” of the above item to retrieve the item IDs of all 10 candidates, which we store in the object candidates
3. We save the IDs of the claims of interest, stored in the object claims
4. We then use some nested sapplycommands to do the following:
   * Retrieve the *item* (entry) for each candidate
   * Extract the eight *claims* from each candidate *item*
   * Process the informational value of each extracted claim, depending on whether the claim value is
     + an atomic object (such as web site URLs)
     + a textual object with auxiliary information (such as names, which come with language information)
     + a time/date (such as date of birth)
     + yet another item (such as previous positions, where each position has an own data base entry)

Code: Retrieving Items and Claims from Wikidata

# get item based on item id

uk\_item <- get\_item("Q30325756", language = "en")

# extract candidates

candidates <- extract\_claims(uk\_item, claims = "P726")

candidates <- candidates[[1]][[1]]$mainsnak$datavalue$value$id

# collect the following attributes ("claims") for each candidate

claims <- c("P1559", "P21", "P569", "P39", "P69", "P856", "P2002", "P2013")

names(claims) <- c("nam", "sex", "dob", "exp", "edu", "web", "twi", "fbk")

claims

## nam sex dob exp edu web twi fbk

## "P1559" "P21" "P569" "P39" "P69" "P856" "P2002" "P2013"

# retrieve data

uk\_data <-

sapply(candidates,

function (item) {

tmp\_item <- get\_item(item, language = "en")

sapply(claims,

function(claim) {

tmp\_claim <- extract\_claims(tmp\_item, claim)[[1]][[1]]

if (any([is.na](http://is.na)(tmp\_claim))) {

return(NA)

} else {

tmp\_claim <- tmp\_claim$mainsnak$datavalue$value

if (is.atomic(tmp\_claim)) {

return(tmp\_claim)

} else if ("text" %in% names(tmp\_claim)) {

return(tmp\_claim$text)

} else if ("time" %in% names(tmp\_claim)) {

tmp\_claim <- as.Date(substr(tmp\_claim$time, 2, 11))

return(tmp\_claim)

} else if ("id" %in% names(tmp\_claim)) {

tmp\_claim <- tmp\_claim$id

tmp\_claim <-

sapply(tmp\_claim,

get\_item,

language = "en",

simplify = FALSE,

USE.NAMES = TRUE)

tmp\_claim <-

sapply(tmp\_claim,

function (x) {

x[[1]]$labels$en$value

})

return(tmp\_claim)

}

}

},

simplify = FALSE,

USE.NAMES = TRUE)

},

simplify = FALSE,

USE.NAMES = TRUE

)

The retrieved data are stored in a nested list. At the upper level of the list, we have the ten candidates, named with their respective item IDs. Nested within each of the ten upper-level elements, we have the values of the eight claims, named with the labels we specified above. Claim values may either be atomic (such as date of birth) or vectors (such as “positions held”, which may have multiple entries). Below, we can see the retrieved data for the first candidate on the list, winning candidate Boris Johnson.

Output: Retrieved Data for Boris Johnson

## $nam

## [1] "Boris Johnson"

##

## $sex

## Q6581097

## "male"

##

## $dob

## [1] "1964-06-19"

##

## $exp

## Q38931

## "Mayor of London"

## Q1371091

## "Secretary of State for Foreign and Commonwealth Affairs"

## Q28841847

## "Member of the Privy Council of the United Kingdom"

## Q30524710

## "Member of the 57th Parliament of the United Kingdom"

## Q30524718

## "Member of the 56th Parliament of the United Kingdom"

## Q35647955

## "Member of the 54th Parliament of the United Kingdom"

## Q35921591

## "Member of the 53rd Parliament of the United Kingdom"

## Q14211

## "Prime Minister of the United Kingdom"

## Q3303456

## "Leader of the Conservative Party"

##

## $edu

## Q192088 Q805285

## "Eton College" "Balliol College"

## Q4804780 Q34433

## "Ashdown House School" "University of Oxford"

## Q5413121

## "European School of Brussels I"

##

## $web

## [1] "<http://www.boris-johnson.com>"

##

## $twi

## [1] "BorisJohnson"

##

## $fbk

## [1] "borisjohnson"

**legislatoR**

legislatoR offers a comprehensive relational individual-level database that provides political, sociodemographic, and other Wikipedia-related data on members of various national parliaments, including the all sessions of the Austrian Nationalrat, the German Bundestag, the Irish Dáil, the French Assemblée, and the United States Congress (House and Senate). It currently comprises data of 42,534 elected representatives and holds information for a wide variety of variables, including:

* sociodemographics (*Core*)
* basic political variables (*Political*)
* records of individual Wikipedia data, including full revision histories (*History*)
* daily user traffic on individual Wikipedia biographies (*Traffic*)
* social media handles and website URLs (*Social*)
* URLs to individual Wikipedia portraits (*Portraits*)
* information on public offices held by MPs (*Offices*)
* MPs’ occupations (*Professions*)
* IDs that link politicians to other files, databases and websites (*IDs*)

The figure below, taken from Göbel and Munzert (2019), illustrates the data structure:



The package provides a relational database. This means that all data sets can be joined with the core data set via one of two keys: the Wikipedia page ID or the Wikidata ID, which uniquely identify individual politicians.

legislatoR services the increasing demand for micro-level data on political elites among political scientists, political analysts, and journalists and offers an accessible and rich collection of data on past and present politicians. The inclusion of Wikipedia and other web data allows for the inclusion of detailed information on politicians’ biographies.

Code: Installing legislatoR

library(legislatoR)

## View functionality

ls("package:legislatoR")

## [1] "get\_core" "get\_history" "get\_ids" "get\_office"

## [5] "get\_political" "get\_portrait" "get\_profession" "get\_social"

## [9] "get\_traffic"

**Application 1: Social Media Adoption Rates**

To retrieve legislatoR data, we first load the entire core data set of a given national parliament using the get\_core() command. In the example below, we focus on the German Bundestag. We immediately right\_join() the core data set with the *political* component using get\_political(). We then filter() the data to retain the legislative session of interest (here, the most recent session of the Bundestag, 2017-2021).

In the next step, we left\_join() this data set with the *social* component using get\_social(). This gives us full information on the social media accounts of all MPs of the 2017-2021 Bundestag. Whenever MPs do not have an account, this is stored as missing information (NA). Using this information, we can calculate the social media adoption rates in the German parliament.

Code: Retrieving legislatoR Data and Calculating Social Media Adoption Rates

## Get social media adoption rates

# get data: Germany

dat\_ger <- right\_join(

x = get\_core(legislature = "deu"),

y = filter(

get\_political(legislature = "deu"),

as.numeric(session) == max(as.numeric(session))

),

by = "pageid"

)

dat\_ger <- left\_join(x = dat\_ger,

y = get\_social(legislature = "deu"),

by = "wikidataid")

dat\_ger$legislature <- "Germany"

dat\_ger\_sum <- dat\_ger %>%

dplyr::summarize(

twitter = mean(not([is.na](http://is.na)(twitter)), na.rm = TRUE),

facebook = mean(not([is.na](http://is.na)(facebook)), na.rm = TRUE),

website = mean(not([is.na](http://is.na)(website)), na.rm = TRUE),

session\_start = ymd(first(session\_start)),

session\_end = ymd(first(session\_end)),

legislature = first(legislature)

)

## twitter facebook website session\_start session\_end legislature

## 1 0.7591036 0.6918768 0.640056 2017-10-24 2021-10-24 Germany

Given the inclusion of *political* variables in our data set, we could think of numerous feasible extensions. For instance, we could look at social media adoption rates by party. Alternatively, we could use the constituency identifiers to add external data on the rurality/urbanity of German electoral districts and analyze whether politicians competing in urban districts are more likely to maintain social media profiles.

**Application 2: Public Attention to Members of the German Bundestag**

In the second application, we use pageviews to identify peaks in public attention for MPs over time. This is particularly interesting in the context of politically significant events. For instance, we may want to know about public attention to parliamentarians following scandals, around elections, or during election campaigns. The code below illustrates this logic by averaging daily pageviews across all Wikipedia articles on members of the German Bundestag between July 2015 and December 2017.

Code: Plotting Average Daily Pageviews for German MPs

## Visualize average pageviews data of German MPs

# get data

ger\_traffic <- right\_join(

x = get\_traffic(legislature = "deu"),

y = filter(

get\_political(legislature = "deu"),

session\_end >= as.Date("2015-07-01")

),

by = "pageid"

)

ger\_traffic <- left\_join(x = ger\_traffic,

y = get\_core(legislature = "deu"),

by = "pageid")

ger\_traffic <-

dplyr::select(ger\_traffic, pageid, date, traffic, session, party, name)

# aggregate data

ger\_traffic$date <- ymd(ger\_traffic$date)

ger\_traffic\_date <- group\_by(ger\_traffic, date)

ger\_traffic\_legislators <- group\_by(ger\_traffic, pageid)

ger\_traffic\_sum <-

summarize(ger\_traffic\_date, mean = mean(traffic, na.rm = TRUE))

ger\_traffic\_sum <- mutate(

ger\_traffic\_sum,

mean\_l1 = lag(mean, 1),

mean\_f1 = lead(mean, 1),

peak = (mean >= 1.8 \* mean\_l1 &

mean > 180)

)

# identify peaks

ger\_traffic\_peaks <- filter(ger\_traffic\_sum, peak == TRUE)

ger\_traffic\_peaks\_df <-

filter(ger\_traffic, date %in% ger\_traffic\_peaks$date)

ger\_traffic\_peaks\_group <-

group\_by(ger\_traffic\_peaks\_df, date) %>%

dplyr::arrange(desc(traffic)) %>%

filter(row\_number() == 1)

ger\_traffic\_peaks\_group <- arrange(ger\_traffic\_peaks\_group, date)

events\_vec <-

c(

"deceased",

"drug affair",

"bullying affair",

"candidacy for presidency",

"deceased",

"???",

"chancellorchip announcement",

"elected president",

"???",

"policy success",

"TV debate",

"general election",

"threat to resign",

"elected speaker of parliament"

)

# plot

par(oma = c(0, 0, 0, 0))

par(mar = c(0, 4, 0, .5))

par(yaxs = "i", xaxs = "i", bty = "n")

layout(matrix(c(1, 1, 3, 2, 2, 3), 2, 3, byrow = TRUE),

heights = c(1, 2, 3),

widths = c(5, 5, 1.8))

# names labels

plot(

ymd(ger\_traffic\_sum$date),

rep(0, length(ger\_traffic\_sum$date)),

xlim = c(ymd("2015-07-01"), ymd("2018-01-01")),

xaxt = "n",

ylim = c(0, 8),

yaxt = "n",

xlab = "",

ylab = "",

cex = 0

)

text(

ger\_traffic\_peaks\_group$date,

0,

ger\_traffic\_peaks\_group$name,

cex = .75,

srt = 90,

adj = c(0, 0)

)

# pageviews time series

par(mar = c(2, 4, 0, .5))

plot(

ymd(ger\_traffic\_sum$date),

ger\_traffic\_sum$mean,

type = "l",

ylim = c(0, 1.25 \* max(ger\_traffic\_sum$mean)),

xlim = c(ymd("2015-07-01"), ymd("2018-01-01")),

xaxt = "n",

yaxt = "n",

xlab = "",

ylab = "mean(pageviews)",

col = "white"

)

abline(h = seq(0, 1.5 \* max(ger\_traffic\_sum$mean), 250), col = "lightgrey")

lines(ymd(ger\_traffic\_sum$date), ger\_traffic\_sum$mean, lwd = .5)

dates <- seq(ymd("2015-07-01"), ymd("2018-01-01"), by = 1)

axis(1, dates[day(dates) == 1 &

month(dates) %in% c(1, 4, 7, 10)], labels = FALSE)

axis(1,

dates[day(dates) == 1 &

month(dates) %in% c(1)],

lwd = 0,

lwd.ticks = 3,

labels = FALSE)

axis(1,

dates[day(dates) == 1 &

month(dates) %in% c(7)],

labels = as.character(year(dates[day(dates) == 15 &

month(dates) %in% c(7)])),

tick = F,

lwd = 0)

axis(2, seq(0, 1.5 \* max(ger\_traffic\_sum$mean), 250), las = 2)

# events labels in time series

for (i in seq\_along(events\_vec)) {

text(ger\_traffic\_peaks\_group$date[i],

ger\_traffic\_sum$mean[ger\_traffic\_sum$peak == TRUE][i] + 80,

i,

cex = .8)

points(

ger\_traffic\_peaks\_group$date[i],

ger\_traffic\_sum$mean[ger\_traffic\_sum$peak == TRUE][i] + 80,

pch = 1,

cex = 2.2

)

}

# election date

# events labels explained

par(mar = c(0, 0, 0, 0))

plot(

0,

0,

xlim = c(0, 5),

ylim = c(0, 10),

xaxt = "n",

yaxt = "n",

xlab = "",

ylab = "",

cex = 0

)

positions <-

data.frame(

events\_xpos = 0.45,

events\_ypos = seq(6.5, (6.5 - .5 \* length(events\_vec)),-.5),

text\_xpos = .5

)

text(0, 7, "Events", pos = 4, cex = .75)

for (i in seq\_along(events\_vec)) {

text(positions$events\_xpos[i], positions$events\_ypos[i], i, cex = .8)

points(positions$events\_xpos[i],

positions$events\_ypos[i],

pch = 1,

cex = 2.2)

text(

positions$text\_xpos[i],

positions$events\_ypos[i],

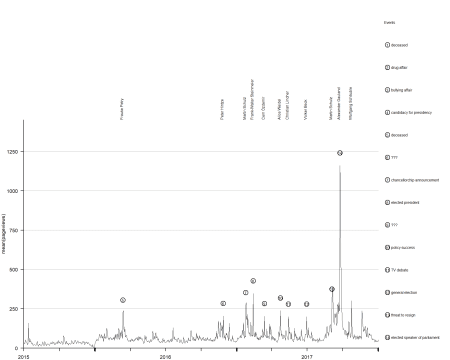
events\_vec[i],

pos = 4,

cex = .75

)

}



The plot shows 14 notable spikes in daily pageviews. We identify which politician’s article generated the most traffic during each of these events. Furthermore, we add a legend that lists salient political events that likely caused these spikes in attention to MPs Wikipedia entries. For example, spike 14 marks the day on which Wolfgang Schäuble (CDU) was elected speaker of the parliament.

**Conclusion**

Collecting and analyzing Wikipedia data is relatively easy and entirely free. It enables researchers to use and analyze an enormous body of data that offers valuable information for research in political science and beyond. Tools that facilitate the collection, processing, and analysis of Wikipedia data advance rapidly, broadening the realm of possibilities for scientific research. Political science research is increasingly picking up on these developments, as is evident in recent contributions (Munzert 2015; Göbel and Munzert 2018; Shi et al. 2019) and softwares (such as legislatoR).

However, using Wikipedia data may also come with limitations and pitfalls. As entries can be read and edited by both humans and machines, the accuracy of contents and the validity of metadata are not guaranteed. With respect to the latter, Wikidata adds provenance information to all the data. These can be used to evaluate the validity of the data in question for applied research.  
Researchers should also keep in mind that Wikipedia data highly depends on user-driven creation, editing, and use of contents. This may not only lead to systematic selection bias due to data availability but also induce problems of equivalence of data points (e.g., articles on historical political figures likely receive fewer views and edits than articles on active politicians for reasons unrelated to their legislative activity or real-world importance).

These caveats are however all but exclusive to Wikipedia data. They merely underline that Wikipedia data is no exception when it comes to the general necessity of thoroughly scrutinizing and critically assessing the suitability of any given data for addressing substantive research questions.