TenK

An R package to scrape and process 10-K annual reports

Jasper Ginn

2016-05-22

`TenK` is an R package aimed at simplifying the collection of SEC 10-K annual reports. It contains the following features:

1. Robust scraping and parsing of reports using the rvest package
2. Resolves FTP urls to their HTML counterparts, which increases the speed of retrieving the documents and adds a lot of useful metadata.
3. Cleans and returns either full reports or just the business desciption for each report.

This document introduces basic usage of the `TenK` package.

A copy of this documentation is available via R in PDF format. To view it, execute `vignette("TenK")` in your R console.

1. Package information

- Package name: TenK
- Version: 0.01
- Documentation
- Report an issue

1.1 Known issues

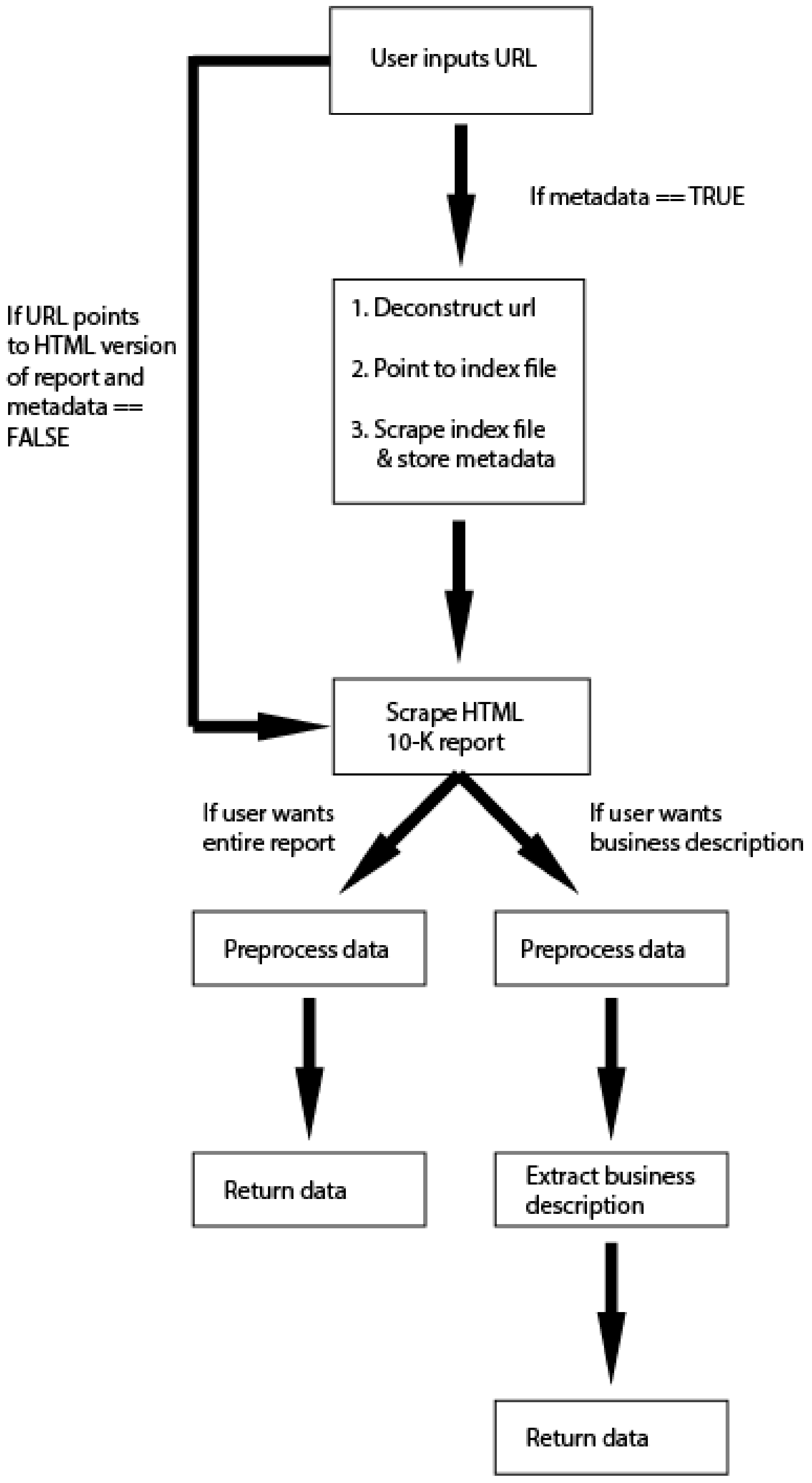
`TenK` can correctly scrape approximately 90% of all business descriptions. If any, issues are usually related to the following causes:

1. The business description has been omitted
2. The business description is located somewhere at the end of the document
3. The report uses unconventional paragraph styles (this will result in the program being unable to find the description and returning "NA").
4. When the text was extracted from the HTML document, certain paragraphs got "squished" together, which throws off the program. (e.g. "part 1 item 1" becomes "part1item1.".)

1.2 How does TenK work?

The main function in this package, `TenK_process`, takes as its input a URL belonging to a 10-K report. The URL can point either to the FTP or the HTML version of the report. If the user passes an FTP url, then `TenK_process` automatically determines the HTML version and collects useful metadata. If the user passes an HTML url, `TenK_process` also collects metadata and returns the scraped text. Currently, `TenK_process` either returns the full 10-K report, or the business description section.

The figure below schematically outlines this process


User inputs URL
If metadata == TRUE
If URL points to HTML version of report and metadata == FALSE
1. Deconstruct url
2. Point to index file
3. Scrape index file & store metadata
Scrape HTML 10-K report
If user wants entire report
If user wants business description
Preprocess data
Preprocess data
Return data
Extract business description
Return data

2. Installing and loading the package

To install the TenK package, execute:

```r
# Install if not present
if(!require(devtools)) install.packages("devtools")
# Load
library(devtools)
# Install TenK from github
install_github("JasperHG90/TenK")
```

You can then load the package as follows:

```r
library(TenK)
```

3. In-built data sets

For the years 2013-2016, TenK provides datasets containing FTP urls for each 10-K filing. These can be queried as follows:

```r
data("filings10K2013") # Exchange 2013 for 2014, 2015 or 2016 for later years.
dim(filings10K2013)
```

```
## [1] 4927    5
```

For more information about these data, execute `?filings10K2013` in the R console.

4. Retrieving 10-K reports

To retrieve a report, use the `TenK_process` function. It has the following parameters:

1. **URL**: (character) FTP or HTML url of the 10-K report
2. **metadata**: (boolean) If FALSE, the function will not return any metadata other than the 10-K HTML url and the report. Defaults to TRUE.
3. **meta_list**: (list) List containing the fields You want to query for the metadata. If empty, all metadata will be returned.
4. **retrieve**: (character) Return either full report ("ALL") or just the business description ("BD")

4.1 Retrieve business description with all metadata

Retrieving all metadata plus the report is straightforward. This is demonstrated in the code block below.

```r
# Retrieve business description
res <- TenK_process(filings10K2013$ftp_url[1], retrieve = "BD")
# Print
print(names(res))
```

```
##  [1] "CIK"            "ARC"            "index.url"      "company_name"
##  [5] "filing_date"    "date_accepted"  "period_report"  "htm10kurl"
##  [9] "htm10Kinfo"     "FTPurl"         "busdescription"
```

You can retrieve a similar result when using a direct HTML url:

```
# Retrieve business description
res2 <- TenK_process(res$htm10kurl, retrieve = "BD")
# Print
print(names(res2))
```

```
##  [1] "CIK"            "ARC"            "index.url"      "company_name"
##  [5] "filing_date"    "date_accepted"  "period_report"  "htm10kurl"
##  [9] "htm10Kinfo"     "busdescription"
```

As you can see, this query does not return the 'FTPurl' field.

The names of these results correspond to the following metadata:

Variable	Description	Optional
CIK	Central Index Key (CIK) of the company. CIK numbers are unique identifiers that the SEC assigns to all entities and individuals that file disclosure documents.	No
ARC	SEC accession number. The accession number is a unique number that EDGAR assigns to each submission as the submission is received. You cannot use accession numbers to filter for types of filings.	No
index.url	Company filings index url. For an example, see: this url	Yes
company_ name	Name of the company	Yes
filing_ date	Date on which the report was filed to the SEC	Yes
date_ accepted	Date/time on which the SEC accepted the report	Yes
period_ report	Fiscal year to which the report belongs.	Yes
htm10kurl	URL pointing to the HTML version of the report.	Yes
htm10kinfo	Meta information about the HTML version of the report. Contains file name, report type, file size and file extension	Yes
FTPurl	URL pointing to the FTP version of the report	No
report	Either the business description or the full report	No

Optional fields can be manually selected/deselected.

4.2 Retrieve business descriptions with selected metadata

If you want to select optional metadata fields, you can do so by passing a list to the 'meta_list' parameter. This is demonstrated in the code block below.

```
# Retrieve business description - select metadata fields
res_select <- TenK_process(filings10K2013$ftp_url[1],
                           meta_list = list("filing_date" = 1,
                                             "date_accepted" = 1,
                                             "htm10kinfo" = 1),
                           retrieve = "BD")
print(names(res_select))
```

```
## [1] "CIK"              "ARC"              "index.url"        "filing_date"
## [5] "date_accepted"  "htm10kurl"        "htm10Kinfo"       "FTPurl"
## [9] "busdescription"
```

4.3 Retrieve business description without metadata

If you don't desire any metadata, you can turn this off by setting the 'metadata' parameter to FALSE:

```
# Retrieve business description without metadata
res_no_metadata <- TenK_process(filings10K2013$ftp_url[1], metadata = FALSE)
# Print
print(names(res_no_metadata))
```

```
## [1] "htm10kurl" "report"
```

5 Storing the results

There are several ways in which you can store the results of the `TenK_process` function. Here, I'll outline 4 ways to do this.

5.1 JavaScript Object Notation (JSON)

You can save R lists (this is what `TenK_process` returns) as JSON files:

```
library(rjson) # Install if you don't have it
# Convenience function
savetojson <- function(data, path) {
  # Write
  g <- toJSON(data)
  write(g, paste0(path, "/data.json"))
}
# Run
savetojson(res, "/users/jasper/desktop")
```

You can load the data as follows:

```
library(rjson)
res <- fromJSON(file = "/users/jasper/desktop/data.json")
```

5.2 Postgresql

Postgresql is a stable, fast and flexible SQL database. Unlike MySQL, it is able to store large text files and is capable of storing terabytes of data.

After installing postgresql, you can use the `RPostgreSQL` package to store and retrieve data.

5.2.1 Creating a table

The first step is to create a table with field names and data types. The example below does this for all metadata. In your own case, you may want to delete some of these fields if you don't require them.

```
# Install if you don't have this package
library(RPostgreSQL)
# Open connection
db <- dbConnect(PostgreSQL(), user = "Jasper", host = "127.0.0.1")
# Create table
q <- dbSendQuery(db, "CREATE TABLE tenk_reports (
                        cik integer,
                        arc bigint,
                        index_url VARCHAR(150),
                        company_name VARCHAR(150),
                        filing_date date,
                        date_accepted TIMESTAMP,
                        period_report date,
                        htm10kurl VARCHAR(150),
                        htm10kinfo_10K_url VARCHAR(100),
                        htm10kinfo_type CHAR(4),
                        htm10kinfo_size integer,
                        htm10kinfo_extension CHAR(3),
                        ftpurl VARCHAR(150),
                        report TEXT
                      );")
```

5.2.2 Writing data to the table

Once you've created your postgres table, you can append data to it by using the `dbWriteTable` function:

```
# Add data - easiest way is to convert list to df
res_df <- as.data.frame(res, stringsAsFactors = F)
# This gives a df with 14 columns and 1 row
dim(res_df)
```

```
## [1]  1 14
```

```
# Replace '.' with '_' and lowercase names
names(res_df) <- gsub("\\.", "_", names(res_df))
names(res_df) <- tolower(names(res_df))
# Write
dbWriteTable(db, "tenk_reports", res_df, append=T,
             row.names = F)
```

```
## [1] TRUE
```

The `htm10kurl` effectively functions as a unique ID for each report. As such, it is convenient to use it as a way to check if a given record already exists in a table:

```
dbCheck <- function(conn, URL) {
  # Query
  q <- dbSendQuery(db, paste0("SELECT exists (SELECT 1 FROM tenk_reports WHERE htm10kurl = ",
                              "'",
                              URL,
                              "'",
                              " LIMIT 1);"))
  # Fetch
  f <- fetch(q)
  # Clear result
  dbClearResult(q)
  # Return exists
  return( f$exists )
}
```

Before you store the record, you can run the check

```
# Run check
check <- dbCheck(db, res$htm10kurl)
# Print
print(check)
```

```
## [1] TRUE
```

If the function returns TRUE, the record already exists. If it returns FALSE, you can go ahead and store the record.

5.2.3 Querying data from the database

To query data from the database, you can use `dbReadTable`:

```
res_query <- dbReadTable(db, "tenk_reports")
# Disconnect
dbDisconnect(db)
```

```
## [1] TRUE
```

As you can see, the data types (which we set when creating the table) are also imported into R:

```
str(res_query, nchar.max = 10)
```

```
## 'data.frame':    1 obs. of  14 variables:
##  $ cik                   : int 1000180
##  $ arc                   : num 1e+14
##  $ index_url             : chr "https://w"| __truncated__
##  $ company_name          : chr "SANDISK C"| __truncated__
##  $ filing_date           : Date, format: "2013-02-1"| __truncated__
##  $ date_accepted         : POSIXct, format: "2013-02-1"| __truncated__
##  $ period_report         : Date, format: "2012-12-3"| __truncated__
##  $ htm10kurl             : chr "https://w"| __truncated__
##  $ htm10kinfo_10k_url  : chr "sndk20121"| __truncated__
##  $ htm10kinfo_type     : chr "10-K"
##  $ htm10kinfo_size     : int 3181183
##  $ htm10kinfo_extension: chr "htm"
##  $ ftpurl              : chr "ftp://ftp"| __truncated__
##  $ report              : chr "this annu"| __truncated__
```

5.3 Mongodb

Mongodb is a NoSQL database that excels at storing large documents and unstructured data (e.g. not column/row pairs).

After installing MongoDB on your system, you can send and load data using the `rmongodb` package.

5.3.1 Creating a database

You don't need to explicitly state that you want to create a mongodb database; rather, you would just start using it *ad hoc*. Note that with mongodb, a *namespace* is a combination of the database and the collection (similar to SQL table).

```
library(rmongodb) # install if you don't have it
# Details
database <- "tenk_reports"
col <- "records"
ns <- paste0(database,".",col)
print(ns)
```

```
## [1] "tenk_reports.records"
```

```
# Create mongo connection
m <- mongo.create()
```

5.3.2 Storing a record

To store a record in the database, you can use `mongo.insert`:

```
# Insert data
mongo.insert(m, ns, res)
```

```
## [1] TRUE
```

Once again, it is a good idea to check if the record already exists in the database. You can do this as follows:

```
recExists <- function(mongo_connection, ns, URL) {
  # Find
  q <- mongo.find.all(mongo_connection, ns, query = list("htm10kurl" = URL))
  # If len >0 , return TRUE, else return FALSE
  ifelse( length(q) > 0, return(TRUE), return(FALSE))
}
```

You can then call it like this:

```r
ex <- recExists(m, ns, res$htm10kurl)
print(ex)
```

```
## [1] TRUE
```

5.3.3 Retrieving a record

To retrieve a record, you can use `mongo.find.one()` or `mongo.find.all()`

```r
# Find one record
res_mongo <- mongo.find.one(m, ns, query = list("htm10kurl" = res$htm10kurl))
# Note that the result is a mongo BSON
class(res_mongo)
```

```
## [1] "mongo.bson"
```

```r
# We can turn it into an R list like this
res_mongo_list <- mongo.bson.to.list(res_mongo)
# We can also query all records at once - note that these get converted to a list immediately
res_mongo <- mongo.find.all(m, ns)
# Disconnect
mongo.destroy(m)
```

```
## NULL
```

Note that, unlike with postgresql, the data does not automatically have the right data type. This is a drawback of schema-less databases like mongo.

```r
str(res_mongo, nchar.max = 10)
```

```
## List of 1
##  $ :List of 12
##   ..$ _id           : chr "5741d51d0"| __truncated__
##   ..$ CIK           : chr "1000180"
##   ..$ ARC           : chr "000100018"| __truncated__
##   ..$ index.url     : chr "https://w"| __truncated__
##   ..$ company_name  : chr "SANDISK C"| __truncated__
##   ..$ filing_date   : chr "2013-02-1"| __truncated__
##   ..$ date_accepted : chr "2013-02-1"| __truncated__
##   ..$ period_report : chr "2012-12-3"| __truncated__
##   ..$ htm10kurl     : chr "https://w"| __truncated__
##   ..$ htm10Kinfo    :List of 4
##   .. ..$ 10K_URL  : chr "sndk20121"| __truncated__
##   .. ..$ Type     : chr "10-K"
##   .. ..$ Size     : num 3181183
##   .. ..$ Extension: chr "htm"
##   ..$ FTPurl        : chr "ftp://ftp"| __truncated__
##   ..$ busdescription: chr "this annu"| __truncated__
```

5.4 Rdata

An Rdata file is a flexible and secure way to store R objects in a highly compressed file on disk. You can save your results as follows:

```r
# Save to Rdata
save(results, file="/users/jasper/desktop/results.Rdata")
```

To load the data, execute the following:

```r
# Load
load(file="/users/jasper/desktop/results.Rdata")
```