ropenfda Package Summary

Maciej Lazarewicz

2017-04-01

The package is a convinence tools for accessing openFDA API https://open.fda.gov. The package enables users to query the Device Adverse Events, Device Classification, Device 510(k) Clearances, Device PMA, Device Registrations and Listings, Device Recalls, Device Recall Enforcement, and Unique Device Identifier databases. For example, a user can query information about recall events about the specific device. More information about the openFDA API is located at https://open.fda.gov/api/reference/.

Packet Setup

```
library(ropenfda)
```

Count Data

Data can be fetched from the openFDA API using `openfda` method. The resulting object is an S4 class specific to the data that was retrieved.

```
res <- openfda(query = "", count_var = "date_facility_aware")
head(res, 3)
```

```
## # A tibble: 3 × 2
##         time count
##       <date> <int>
## 1 1950-01-01     1
## 2 1951-03-06     1
## 3 1960-01-01     1
```

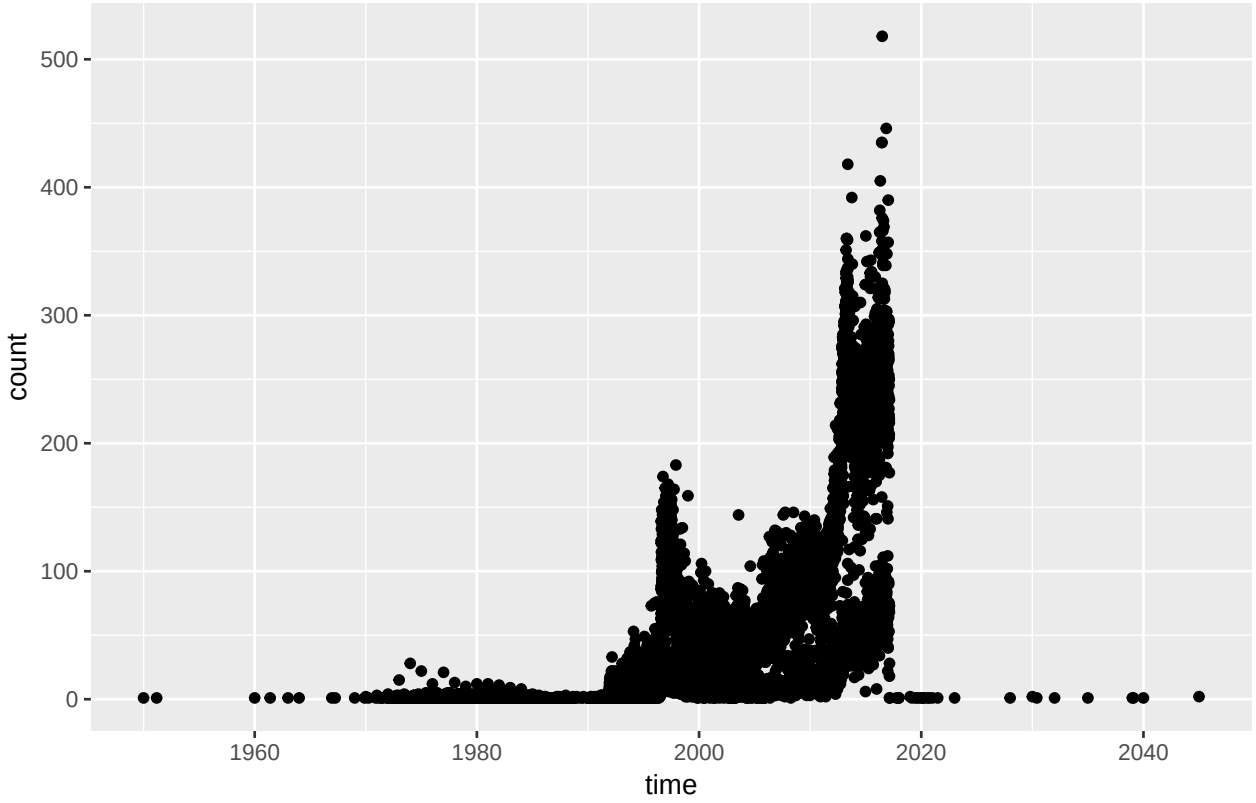
```
class(res)
```

```
## [1] "CountDeviceEvent"
## attr(,"package")
## [1] "ropenfda"
```

Parameter `query`, which in this example is an empty string, is getting all available data from the default category `device` and default database `event`. The parameter `count_var` specifies a field which unique values are counted and provided as a result. Dates are automatically converted into the `Date` objects. The class of this object is `CountDeviceEvent` class.

A quick look into the count data can be done with `plot` method. We will quickly notice that there are few points with wrong dates beyond year 2017!

```
plot(res)
```



Parameter `category` allows also to access `drug` and `food` data.

```
res <- openfda(query = "", category = "drug", count_var = "companynumb")
head(res, 3)
```

```
## # A tibble: 3 × 2
##     term   count
##    <chr>   <int>
## 1     us 4027882
## 2    inc  807621
## 3 pfizer  502142
```

```
class(res)
```

```
## [1] "CountDrugEvent"
## attr(,"package")
## [1] "ropenfda"
```

```
res <- openfda(query = "", category = "food", count_var = "date_created")
head(res, 3)
```

```
## # A tibble: 3 × 2
##         time count
##       <date> <int>
## 1 2004-01-01  2595
## 2 2005-01-01  2013
## 3 2006-01-01  1788
```

```
class(res)
```

```
## [1] "CountFoodEvent"
## attr(,"package")
```

```
## [1] "ropenfda"
```

Raw Data

In order to access original, not aggregated data, we simply omit the `count_var` parameter. For raw data openFDA API returns up to 5100 rows. If we do not specify the `limit` parameter, the default value is 5100.

```
res <- openfda(query = "", limit = 100)
```

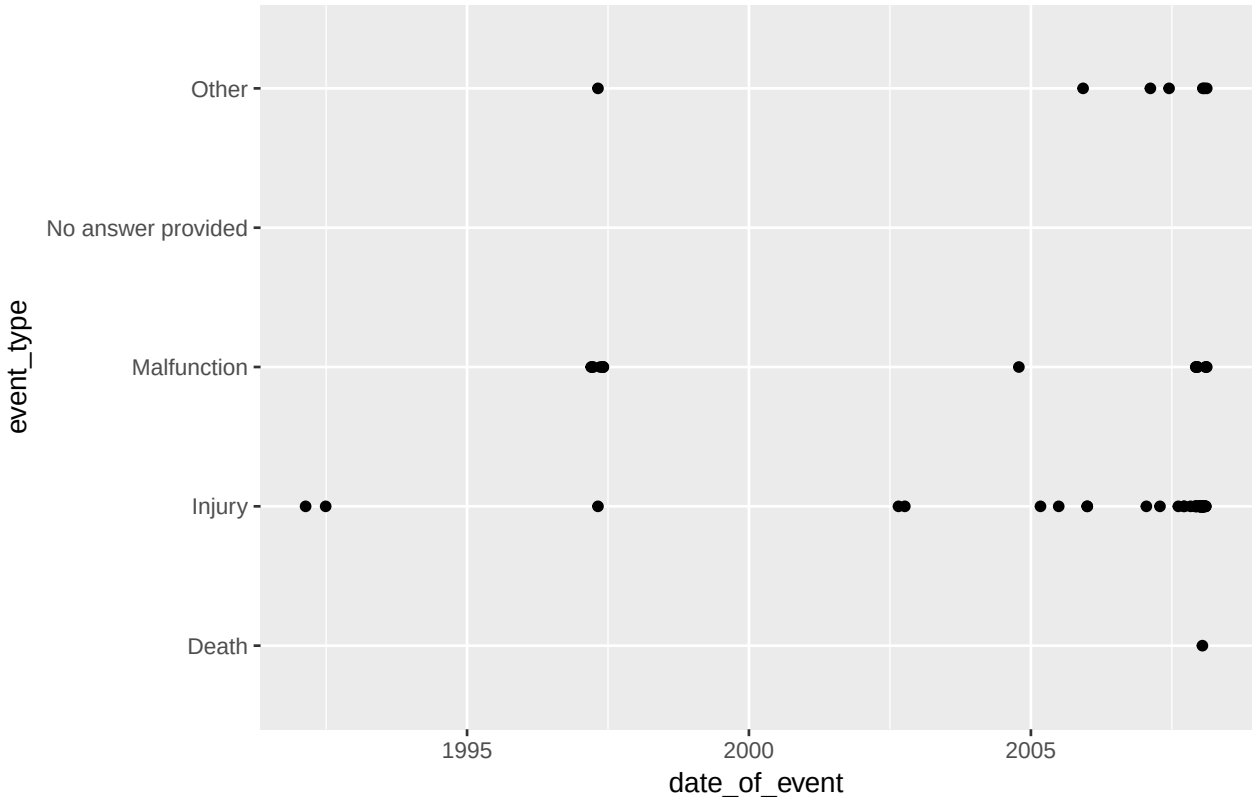
```
## Warning: Only 100 records retrieved from the total of 6071400
```

```
res[1:4, c(3,4,10,13)]
```

```
## # A tibble: 4 × 4
##   event_location report_to_fda         event_type      report_number
##            <chr>         <chr>              <chr>              <chr>
## 1       HOSPITAL             Y             Injury                 10
## 2          OTHER             Y             Injury 2243621-1992-00036
## 3   INVALID DATA             N No answer provided              10000
## 4       HOSPITAL             N             Injury             100000
```

Similarly, quick look into data can ba done with the `plot` method:

```
plot(res, i = "date_of_event", j = "event_type")
```



Other Databases

To access the device recall information we would speficy the parameter `db`:

```
res <- openfda(query = "", db = "recall", limit = 10)
```

```
## Warning: Only 10 records retrieved from the total of 98946
```

```
head(res)
```

```
## # A tibble: 6 × 16
##   other_submission_description res_event_number firm_fei_number k_numbers
##                          <chr>            <chr>           <chr>    <list>
## 1                                         65228         1828100 <chr [3]>
## 2                                         37366         1828100 <chr [1]>
## 3                                         36494         1828100 <chr [1]>
## 4                                         60718         1828100 <chr [2]>
## 5                                         60718         1828100 <chr [2]>
## 6                                         44838         1828100 <chr [1]>
## # ... with 12 more variables: product_code <chr>,
## #   root_cause_description <chr>, ...
```

Convenience Functions

Some convenience methods like `head` , `tail`, `[`, `as_tibble`, `as.data.frame`, `show`, `names`, `nrow`, `ncol` work as expected:

```
cat("dim:", dim(res), "nrow:", nrow(res), "ncol:", ncol(res))
```

```
## dim: 10 16 nrow: 10 ncol: 16
```

```
names(res)
```

```
##  [1] "other_submission_description"
##  [2] "res_event_number"
##  [3] "firm_fei_number"
##  [4] "k_numbers"
##  [5] "product_code"
##  [6] "root_cause_description"
##  [7] "pma_numbers"
##  [8] "event_date_terminated"
##  [9] "product_res_number"
## [10] "openfda.device_name"
## [11] "openfda.medical_specialty_description"
## [12] "openfda.device_class"
## [13] "openfda.regulation_number"
## [14] "openfda.registration_number"
## [15] "openfda.fei_number"
## [16] "openfda.k_number"
```

Extracting Data as a Tibble

`as_tibble` extracts the data form the object as a tibble. It is advised to use `as_tibble` since in some cases the raw data might contain nested data frames and it might trigger problems in RStudio with displaying it on the screen when extracted as a `data.frame`.

```
head(as_tibble(res), 3)
```

```
## # A tibble: 3 × 16
```

```
##   other_submission_description res_event_number firm_fei_number k_numbers
##                          <chr>            <chr>           <chr>    <list>
## 1                                          65228         1828100 <chr [3]>
## 2                                          37366         1828100 <chr [1]>
## 3                                          36494         1828100 <chr [1]>
## # ... with 12 more variables: product_code <chr>,
## #   root_cause_description <chr>, ...
```

```
class(as_tibble(res))
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

Extracting Data as a Data Frame

Data in the `Data` object can also be accessed as data.frame by applying `as.data.frame` function.

```
res <- openfda(query = "", count_var = "date_facility_aware")
x <- as.data.frame(res)
class(x)
```

```
## [1] "data.frame"
```

```
head(x)
```

```
##         time count
## 1 1950-01-01     1
## 2 1951-03-06     1
## 3 1960-01-01     1
## 4 1961-05-24     1
## 5 1963-01-01     1
## 6 1964-01-01     1
```

Complex Queries

Query strings can be complex, composed as a logical expression of conditions imposed on the fields.

```
query_string <- '((adverse_event_flag == "N" and single_use_flag == "Y") or (adverse_event_flag == Y+N+
res1 <- openfda(query_string, limit = 10)
```

```
## Warning: Only 10 records retrieved from the total of 953
```

```
dim(res1)
```

```
## [1] 10 74
```

Phrases

Quoted phrases are searched as a whole entity and records that contain that entity will be returned.

```
res2 <- openfda(query = 'manufacturer_g1_name == "MEDTRONIC NEUROMODULATION"',
                limit = 1)
```

```
## Warning: Only 1 records retrieved from the total of 112385
```

You can also search for a single word. The next query is looking only for the word `MEDTRONIC` and returns more records than the previous which was looking for the whole phrase `MEDTRONIC NEUROMODULATION`.

```
res2 <- openfda(query = 'manufacturer_g1_name == "MEDTRONIC"', limit = 1)
```

```
## Warning: Only 1 records retrieved from the total of 765799
```

Multiple Field Values

If you want to look for `MEDTRONIC` or `NEUROMODULATION`, use `+` between the words/phrases.

```
res2 <- openfda(query = 'manufacturer_g1_name == "MEDTRONIC"+"NEUROMODULATION"',
                limit = 1)
```

```
## Warning: Only 1 records retrieved from the total of 853090
```

Exact Search

In order to look for records with fileds equal exectly to the given phrase, one nees to use `.exact` extension to the field name.

```
res3 <- openfda(query = 'manufacturer_g1_name.exact == "MEDTRONIC NEUROMODULATION"',
                limit = 1)
```

```
## Warning: Only 1 records retrieved from the total of 112384
```

Object Oriented Resprsentation

There are two families of classes: `Query` and `Data`. The `Query` class represnts the query to the openFDA **before** retrieving the actual data from the API.

```
query <- get_query(query = 'manufacturer_g1_name == "MEDTRONIC"', limit = 1)
query
```

```
## object: RawQuery
## ================================================
## query time stamp: 2017-04-01 21:56:26
## fda category: device
## fda database: event
## query: manufacturer_g1_name == "MEDTRONIC"
## query api:
## https://api.fda.gov/device/event.json?&search=manufacturer_g1_name:"MEDTRONIC"&limit=1
## limit: 1
```

The `Data` class, represents the data that was fetched from the openFDA servers.

```
data <- fetch(query)
```

```
## Warning: Only 1 records retrieved from the total of 765799
```

```
data
```

```
## # A tibble: 1 × 74
##   manufacturer_contact_zip_ext manufacturer_g1_address_2 event_location
##                          <chr>                     <chr>          <chr>
## 1                         5604                                    OTHER
## # ... with 71 more variables: report_to_fda <chr>,
## #   manufacturer_contact_t_name <chr>, ...
```

The class of that object is specific to the result type. In this case it is the `raw` data type, category of `device` and database `event`:

```
class(data)
```

```
## [1] "RawDeviceEvent"
## attr(,"package")
## [1] "ropenfda"
```

Unfolding The Nested Data

In some instances, the data returned from the API is a nested frame. For instance, in the case of the `device event` data, some records contain in the field `patient` nested `data.frame`. Also, fields `device` and `mdr_text` are `data.frames`.

```
res <- openfda(query = "", limit = 100)
```

```
## Warning: Only 100 records retrieved from the total of 6071400
```

```
res[1:3, c("patient", "device", "mdr_text")]
```

```
## # A tibble: 3 × 3
##                patient               device             mdr_text
##                 <list>               <list>               <list>
## 1 <data.frame [2 × 4]> <data.frame [1 × 32]> <data.frame [1 × 4]>
## 2 <data.frame [2 × 4]> <data.frame [1 × 32]> <data.frame [1 × 4]>
## 3 <data.frame [2 × 4]> <data.frame [1 × 28]> <data.frame [1 × 4]>
```

The method `unfold` converts nested `tibble` into the unnested `tibble`. The resulting tibble has more rows and more columns than the original one.

```
res_unfold <- unfold(res)
data_unfold <- as_tibble(res_unfold)
dim(res)
```

```
## [1] 100  78
```

```
dim(data_unfold)
```

```
## [1] 359 118
```

```
head(setdiff(names(data_unfold), names(res)))
```

```
## [1] "mdr_text.patient_sequence_number"  "mdr_text.mdr_text_key"
## [3] "mdr_text.text_type_code"           "mdr_text.text"
## [5] "patient.sequence_number_treatment" "patient.date_received"
```