ABSTRACT

One of the main challenges during the drug development process is knowing how a drug in a clinical trial interacts/might interact with other concomitant medications taken by a patient. The FDA has enabled access to real-world data via OpenFDA. With the use of OpenFDA APIs - specifically by creating daily refreshed data reports - we can track the emergence/development of common adverse reactions or drug interactions along with the severity and seriousness of such adverse events across various patients. I will be pulling data from OpenFDA APIs and creating reports that reflect real-time data from the OpenFDA database for the purpose of analysis. I will also be exploring the metadata and discussing the interactive charts provided by OpenFDA to help with the right query selection from the database. Utilizing tools such as R libraries, Excel pivot charts and pivot tables, and converting JSON files into data for creating customized reports, I will also be discussing a way of automating reports and notifications to enable a faster alert system for clinical investigators, ensuring greater patient safety.

INTRODUCTION

OpenFDA has provided many references on how to run a query within their database, few examples are listed below:

Figure 1.0: Some of the example API Query calls: from OpenFDA where searchable field is receivedate, in the above query a date range is provided.

Adverse drug event reports since 2004

This is the openFDA API endpoint for adverse drug events. An adverse event is submitted to the FDA to report any undesirable experience associated with the use of a drug, including serious drug side effects, product use errors, product quality problems, and therapeutic failures.

Reporting of adverse events by healthcare professionals and consumers is voluntary in the United States. Increases in the total number of adverse events are likely caused by improved reporting. News, enforcement actions, and other phenomena can also spur reporting.

![Reports over time](image)

View:
- **serious**

Filter:
- All adverse event reports
- Reported through manufacturers
- Reported directly by public
- Where indication for drug use was hypertension

Custom search parameter:
- Type in a custom search parameter, and then press Enter to update the chart

current query
- 3165547 records

Figure 1.1: Some of the example API Query calls: from OpenFDA where searchable field is serious, in the above query a date range is provided


UNDERSTANDING API CALLS

![Diagram](image)

Client sends a request
- GET
- POST
- PUT
- DELETE

HTTP methods
- JSON
- HTTP

Server sends a response

Figure 2.0: API calls are based on a set of commands where the client receives a response from the server or backend database based on HTTP methods – GET, POST, PUT, DELETE

UNDERSTANDING DATA STRUCTURES USING POSTMAN

Postman is an API platform for building and using APIs. Postman simplifies each step of the API lifecycle and streamlines collaboration so you can create better APIs faster.

![Figure 3.0: The Postman API Platform](https://www.postman.com/product/what-is-postman/)


SAMPLE API RESPONSE

![Figure 4.0: Sample API response in Postman](https://api.fda.gov/drug/event.json?search=receivedate:[20040101+TO+20081231]&limit=1)

The above response displays the metadata and all values in a single record in Postman.
REPORTING FROM API RESPONSES

The following programming approach was adopted with R as the main programming language.

DEFINE PACKAGES AND LIBRARIES

Install the following packages in R and define libraries:

```r
library(jsonlite)
library(tidyr)
library(dplyr)
library(openxlsx)
library(lubridate)
```

DEFINE API DESTINATION

```r
Url <- "https://api.fda.gov/drug/event.json?search=receivedate:[20040101+TO+20231231]&limit=100"
response <- fromJSON("https://api.fda.gov/drug/event.json?search=receivedate:[20040101+TO+20231231]&limit=100")
```

df <- as.data.frame(response)

# Create a sequence number column to add as placeholder:

```r
df <- df %>% mutate(seq_num = row_number())
```

DATA WRANGLING - NORMALIZE/READ NESTED COLUMNS

In the OpenFDA API call, the Result column has nested columns for Patient level reaction and drug name. We would now unnest these columns. This will create two separate datasets which we will merge to get the complete unnested data.

```r
df_normalized_reaction <- df %>% unnest(result.patient)
colnames(df_normalized_reaction)
```

```r
df_reaction <- df_normalized_reaction %>% unnest(reaction)
colnames(df_reaction)
```

![Figure 5.0: Unnested columns in Results.Patient.reaction](image)

Figure 5.0: Unnested columns in Results.Patient.reaction
head(df_reaction,1)
#Write columns into external file to validate/check nested columns
df_r <- data.frame(df_reaction)
#write.xlsx(df_r, file="test.xlsx")
head(df_r,1)
colnames(df_r)
df_drug<- df_normalized_reaction %>% unnest(drug)
df_drug <- data.frame(df_drug)
colnames(df_drug)

Figure 6.0: Unnested columns in Results.Patient.Drug

Merge all Data sets having unnested columns:
merged_df <- merge(df_drug,df_reaction, by = "seq_num")
colnames(merged_df)
my_df <- data.frame(merged_df)
head(my_df, 3)
REPORT CREATION

We will filter data based on all serious drug related adverse events to generate report:

```r
filtered_df <- my_df[my_df$result.serious.x == 1, ]
```

Install packages and create a placeholder for today’s date:

today <- Sys.Month()
install.packages("lubridate")
library(lubridate)

```r
start_of_month <- as.Date(paste0(format(Sys.Date(), "%Y-%m"), "-01"))
end_of_month <- as.Date(paste0(format(Sys.Date(), "%Y-%m"), "-31"))
monthly_report <- filter_df %>% filter(meta.last_updated.x >= start_of_month, meta.last_updated.x <= end_of_month)
```

Report generated as HTML using the

```r
---
title: "Monthly Report"
author: "Organization Name"
date: "r format(Sys.Date(), "%B %d, %Y")"
output:
html_document:
keep_md: true
---
```

Generate monthly report based on

```r
monthly_report$medicinalproduct
monthly_report$FIGURE.company.number.x
monthly_report
```

OPENFDA DATA IN EXCEL SPREADSHEET

OpenFDA data can also be downloaded and opened in Excel spreadsheets along with the nested columns and analytical charts/tables may be prepared.

![Excel Spreadsheet](image)

Figure 7.0: Normalized columns in *.xlsx format
CONCLUSION

Reporting with public APIs first requires an understanding of the API structure, followed by data manipulation to unnest any hidden columns. This may be done easily with R and other tools such as Python that have custom functions to read JSON API’s. POSTMAN is a valuable tool that may serve as a guide in this process. Daily reporting with R may be done programmatically within R and also using CRON jobs too. This would ensure data parameter/filtered reporting.
REFERENCES


RECOMMENDED READING


CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Shubhranshu Dutta
University of Rochester
Email: shudutt@gmail.com

Any brand and product names are trademarks of their respective companies.