Post 1: Collecting, Cleaning, and Wrangling Time Series Data from the Energy Information Administration

Athan Diep

October 31, 2017

Introduction and Motivation

For most data scientists, the most time-consuming aspect of the data analysis project lifecycle is often the collection and preparation of data. Time series data can sometimes be particularly difficult to work with. For this post, I will be discussing various methods on the collection, preparation, and basic exploratory analysis on time series data from the U.S. Energy Information Administration (EIA), which releases data each week on the supply, production, and movement of energy commodities.

I recently found my interest in this dataset after reading a few news articles on how the price of a barrel of oil has changed as a result of recently announced supply cuts by the EIA (see sources for article). Such data can be used to somewhat accurately forecast the future prices and other metrics of commodities such as crude oil and natural gas. This analysis is of particular interest to suppliers and buyers of the commodities as well as many financial firms who look to profit off speculation. While no actual time series modelling will be done, all the necessary preparation for such modelling will be laid out within this post to create a fully functional data pipeline. I am choosing to use weekly import and export data for the purposes of this project, but any data series from the EIA website can easily be interchanged for them.

Load the Packages We Need

Let's load all the necessary packages (note: packages must first be installed using install.packages()):

```
library(jsonlite)
library(lubridate)
```

```
##
## Attaching package: 'lubridate'
```

```
## The following object is masked from 'package:base':
##
##     date
```

```
library(ggplot2)
```

Importing and Extracting the Raw JSON web data

First, we must, of course, import the data. There are a few ways to do this for the Energy Information Administration's data. Thankfully, the agency provides an application programming interface (API) that allows for easy access to structured data (either in JSON or XML format). For this post, we will choose to import from JSON structured data. The API of the website allows for you to query the data you want by specifying the parameters within the URL. A required parameter is your individualized API key, which you can get from a free and easy registration process.

The base url for querying a specific data series from the EIA website is: http://api.eia.gov/series/?api_key=

```
baseurl <- "http://api.eia.gov/series/?api_key="
#API key will be emailed after registering for free on EIA website
api_key <- "1fd0415f14bd9fa95e20fd72227a647d"
```

Query Parameters

Following the base url, you must specify your parameters starting with your provided API key. For the parameters other than API key, you can place them in any order and specify them with "&parametername=parametervalue" in the url. The parameters we need are:

series_id (required): the ID of the particular data series you are querying (can be found through catalog, see sources)
out (optional, default is JSON): the format of the data, can either be "xml" or "json"
num (optional, default is as far back in time as there is data for the series): how many of the most recent data points to call

An example url to query the data series for weekly import of crude oil (series id: PET.WCRIMUS2.W) in JSON with 30 of the most recent data points is:
http://api.eia.gov/series/?api_key=1fd0415f14bd9fa95e20fd72227a647d&series_id=PET.WCRIMUS2.W&num=30

Creating the URL

For this post, we will look at the imports and exports of crude oil. After browsing the catalog on the EIA website (see sources), we find that the series id's are "PET.WCREXUS2.W" for exports and "PET.WCRIMUS2.W" for imports. We then assemble the proper url's for a query of 30 recent data points.

```
#Each unique data series id is used to identify the series and can be taken from the catalog
import_id <- "PET.WCRIMUS2.W"
export_id <- "PET.WCREXUS2.W"
#Now we combine the base url, api key, and the other parameters into one string for the whole url we want
import_url <- paste0(baseurl, api_key, "&series_id=", import_id, "&num=30")
export_url <- paste0(baseurl, api_key, "&series_id=", export_id, "&num=30")
```

Extraction and Format of JSON data

Now that we have created the url's, we must extract the JSON data into a dataframe. To do this, we use the **jsonlite** package that we have already loaded.

A Look at the Raw Data

If you go to one of the url's, you will observe the data is in a nested structure as JSON data typically is. If we observe the structure for the requested data, we see:

{"request":{"command":"series","series_id":"PET.WCRIMUS2.W"},"series":[{"series_id":"PET.WCRIMUS2.W","name":"U.S. Imports of Crude Oil, Weekly","units":"Thousand Barrels per Day","f":"W","unitsshort":"Mbbl/d","description":"U.S. Imports of Crude Oil","copyright":"None","source":"EIA, U.S. Energy Information Administration","iso3166":"USA","geography":"USA","geography2":"USA","start":"19900105","end":"20171020","updated":"2017-10-25T13:25:35-0400","data":[["20171020",8123],["20171013",7483],["20171006",7617],["20170929",7214],["20170922",7427]]}]}

Extracting the Data from JSON structure

In the JSON formatted data structure, data is organized in an array or list of key-value pairs with some values also being lists that consist of more key-value pairs. Keys and values are separated by colons. We notice that the data that we want to extract are the pairs of values within the value for the "data" key. To extract this data, we use the fromJSON function in the jsonlite package to get all the raw data into a list which then can be manipulated to get a dataframe with just the data we want. We specify the parameters in the fromJSON function with flatten = TRUE (to create a non-nested data frame) and simplifyDataFrame = TRUE (to coerce the JSON arrays into data frames). We do this for both the import and export data and look at the structure of one of them.

```
raw_import_data <- fromJSON(import_url, simplifyDataFrame = TRUE, flatten = TRUE)
raw_export_data <- fromJSON(export_url, simplifyDataFrame = TRUE, flatten = TRUE)
#Observe the structure of the raw JSON data
str(raw_import_data)
```

```
## List of 2
##  $ request:List of 2
##   ..$ command  : chr "series"
##   ..$ series_id: chr "PET.WCRIMUS2.W"
##  $ series :'data.frame': 1 obs. of  15 variables:
##   ..$ series_id  : chr "PET.WCRIMUS2.W"
##   ..$ name       : chr "U.S. Imports of Crude Oil, Weekly"
##   ..$ units      : chr "Thousand Barrels per Day"
##   ..$ f          : chr "W"
##   ..$ unitsshort : chr "Mbbl/d"
##   ..$ description: chr "U.S. Imports of Crude Oil"
##   ..$ copyright  : chr "None"
##   ..$ source     : chr "EIA, U.S. Energy Information Administration"
##   ..$ iso3166    : chr "USA"
##   ..$ geography  : chr "USA"
##   ..$ geography2 : chr "USA"
##   ..$ start      : chr "19900105"
##   ..$ end        : chr "20171020"
##   ..$ updated    : chr "2017-10-25T13:25:35-0400"
##   ..$ data       :List of 1
##   .. ..$ : chr [1:30, 1:2] "20171020" "20171013" "20171006" "20170929" ...
```

Now, to get the target data which is a value under the "data" key which is in turn under the "series" key, we use the "$" operator to extract the data and create a new list. We then convert this list with all of our data into a dataframe of two columns, a date column and a column with the number of thousand of barrels of import/exports.

```
import_list <- raw_import_data$series$data
import_df <- as.data.frame(import_list)
colnames(import_df) <- c("date", "imports")
export_list <- raw_export_data$series$data
export_df <- as.data.frame(export_list)
colnames(export_df) <- c("date", "exports")
```

We now check the head of the dataframes to see if it has been extracted correctly.

```
#Call first 6 rows of the import dataframe
head(import_df)
```

```
##       date imports
## 1 20171020    8123
## 2 20171013    7483
## 3 20171006    7617
## 4 20170929    7214
## 5 20170922    7427
## 6 20170915    7368
```

```
#View first six rows of export dataframe. Notice the dates are the same for this one and the imports frame.
head(export_df)
```

```
##       date exports
## 1 20171020    1924
## 2 20171013    1798
## 3 20171006    1270
## 4 20170929    1984
## 5 20170922    1491
## 6 20170915     928
```

Cleaning the Data and Adding New Variables

We now should clean and extract the data to make it usable for any analysis that we might want to do. First, it would be nice to have the import and export data merged into a single dataframe.

Merging the dataframes

To merge the dataframes, the merge function from the base R package will be useful. We would like to join the two dataframes by the date column that they have in common (using the "by" parameter of the function). In this case, our choice of join type (i.e. inner, outer, left, or right) is irrelevant given that the date columns of the two dataframes are identical.

```
#Merge the two dataframes by the date column
df <- merge(export_df, import_df, by = "date")
#Checking to see that it merged correctly. Notice that the merged frame is in reverse order now, with the oldest d
ata coming first
head(df)
```

```
##       date exports imports
## 1 20170331     575    7850
## 2 20170407     689    7878
## 3 20170414     565    7810
## 4 20170421    1152    8912
## 5 20170428     538    8264
## 6 20170505     693    7620
```

Converting the Column Types

Notice that all the columns are currently a factor type as it is the default. We need to convert them to more appropriate types that we can actually work with. The export and import columns should be converted to numerics (using as.numeric), and we can convert the date column to a date type. Note that we must first convert the export and import columns into characters before converting to numeric or else it will not convert correctly. Let's first convert the export and import columns.

```
#Wrapping the numeric converter function around the numeric conversion function
df$exports <- as.numeric(as.character(df$exports))
df$imports <- as.numeric(as.character(df$imports))
str(df)
```

```
## 'data.frame':    30 obs. of  3 variables:
##  $ date   : Factor w/ 30 levels "20170331","20170407",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ exports: num  575 689 565 1152 538 ...
##  $ imports: num  7850 7878 7810 8912 8264 ...
```

Now, to convert the date column, we opt to use the lubridate package for R which contains easy-to-use and intuitive functions for converting dates. As the the dates are in the form "yyyymmdd", we use the ymd() function. Just like the exports and imports, since the dates are factors, we must first convert it to strings to be able to use our ymd() function and thus wrap the ymd() function around the as.character() function.

```
df$date <- ymd(as.character(df$date))
```

Creating a New Column

In our future analysis, we might want to have a new variable of net imports that may enhance our findings. To add the column , it can be calculated as the difference between the total imports and total exports for each week. We can easily create this column using the dplyr package or the vectorized operators of R:

```
#Due to vectorized property, we can subtract whole columns
df$net_imports <- df$imports - df$exports
df$date
```

```
##  [1] "2017-03-31" "2017-04-07" "2017-04-14" "2017-04-21" "2017-04-28"
##  [6] "2017-05-05" "2017-05-12" "2017-05-19" "2017-05-26" "2017-06-02"
## [11] "2017-06-09" "2017-06-16" "2017-06-23" "2017-06-30" "2017-07-07"
## [16] "2017-07-14" "2017-07-21" "2017-07-28" "2017-08-04" "2017-08-11"
## [21] "2017-08-18" "2017-08-25" "2017-09-01" "2017-09-08" "2017-09-15"
## [26] "2017-09-22" "2017-09-29" "2017-10-06" "2017-10-13" "2017-10-20"
```

Date Adjustment

We are not done yet! Upon further inspection of the dates, we can notice that the dates almost all fall on Fridays. However, the EIA website states that they release new data every Wednesday morning and each date within the provided data reflects the “as of” date (which is always the previous Friday) and not the release date of the data (see sources). If we were to do some kind of time series analysis or try to make predictions with this data in conjunction with other time series data that is of different frequency or is on different days, we might run into some problems with look-ahead bias (see sources). Essentially, if you use a data point that corresponds to a certain day in some kind of predictive model so that you can use today’s and future day’s data to predict something that lies ahead of them, but you mistakenly attribute the data point to a date on which that data was not known yet, it will corrupt the integrity of your model. Therefore, we should shift all the days of this dataframe up by five days (to get from Friday to the following Wednesday).

Fortunately, the date class of R allows for easy manipulation of dates. Particularly, you can add or subtract an integer number of days to any date, and it will return the date that many days forward or behind. Take a look at the following example with the first entry of our “date” column as we add two days to March 31st, 2017:

```r
df$date[1]
```

```
## [1] "2017-03-31"
```

```r
df$date[1] + 2
```

```
## [1] "2017-04-02"
```

Now, using this cool property of R dates, we can adjust all the dates of the date column to the actual date on which they are known to the public. Again, we utilize the vectorized property of R which we learned in class.

```r
df$date <- df$date + 5
```

Let’s take a look at the finished product with the correct dates and our new column.

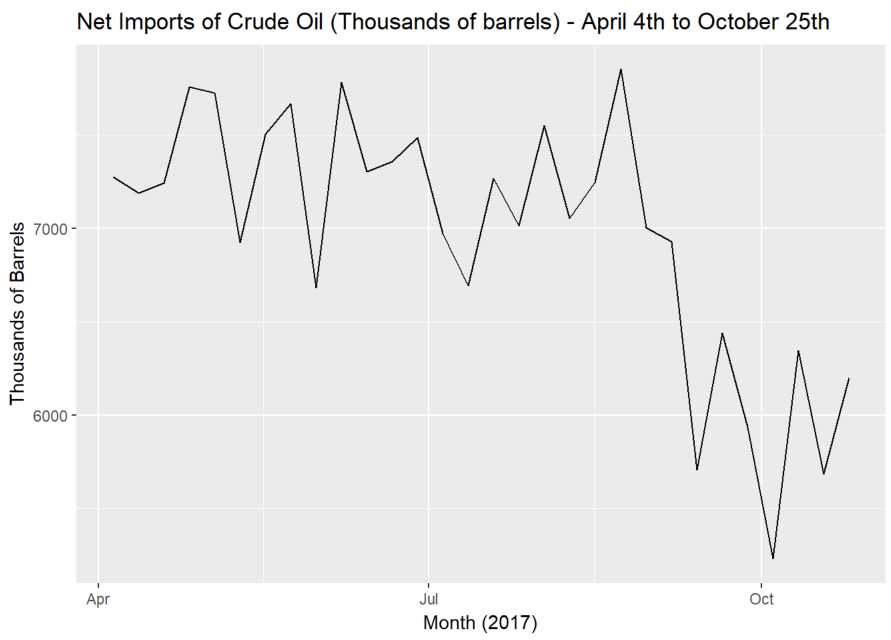
```r
head(df)
```

```
##         date exports imports net_imports
## 1 2017-04-05     575    7850        7275
## 2 2017-04-12     689    7878        7189
## 3 2017-04-19     565    7810        7245
## 4 2017-04-26    1152    8912        7760
## 5 2017-05-03     538    8264        7726
## 6 2017-05-10     693    7620        6927
```

Visualization

Let us now try to go into the initial steps of the exploratory data analysis phase and attempt to visualize the date we have collected and prepared. To plot the time series data, we can use ggplot2. Here we plot the net imports of crude oil for the previous 30 weeks.

```r
ggplot(data = df, aes(date, net_imports)) + geom_line() + ggtitle("Net Imports of Crude Oil (Thousands of barrels)
- April 4th to October 25th") + xlab("Month (2017)") + ylab("Thousands of Barrels")
```



Summary

In this post, we have taken a look at the entire process for collecting and preparing time series data from the Energy Information Administration to perhaps perform an in-depth analysis on. It began with scraping the data from the appropriate web source using JSON tools in R followed by the cleaning of data and creation of a new variable. I have also shown one way of manipulating the dates to have more sound time-series data for further analysis. Lastly, we visualized our time series data with ggplot2.

Sources

Article on Supply Cuts: http://www.reuters.com/article/us-global-oil/oil-steady-near-two-year-high-as-supply-cuts-bite-idUSKBN1D0026?rpc=401&

EIA API documentation: https://www.eia.gov/opendata/commands.php

Data series Catalog: https://www.eia.gov/opendata/qb.php

jsonlite documentation & examples: https://cran.r-project.org/web/packages/jsonlite/vignettes/json-aaquickstart.html

EIA release day documentation: https://www.eia.gov/petroleum/supply/weekly/schedule.cfm

Look ahead bias: https://www.quantopian.com/posts/backtester-details-and-lookahead-bias

Plotting time series data with ggplot2: https://stackoverflow.com/questions/4843969/plotting-time-series-with-date-labels-on-x-axis