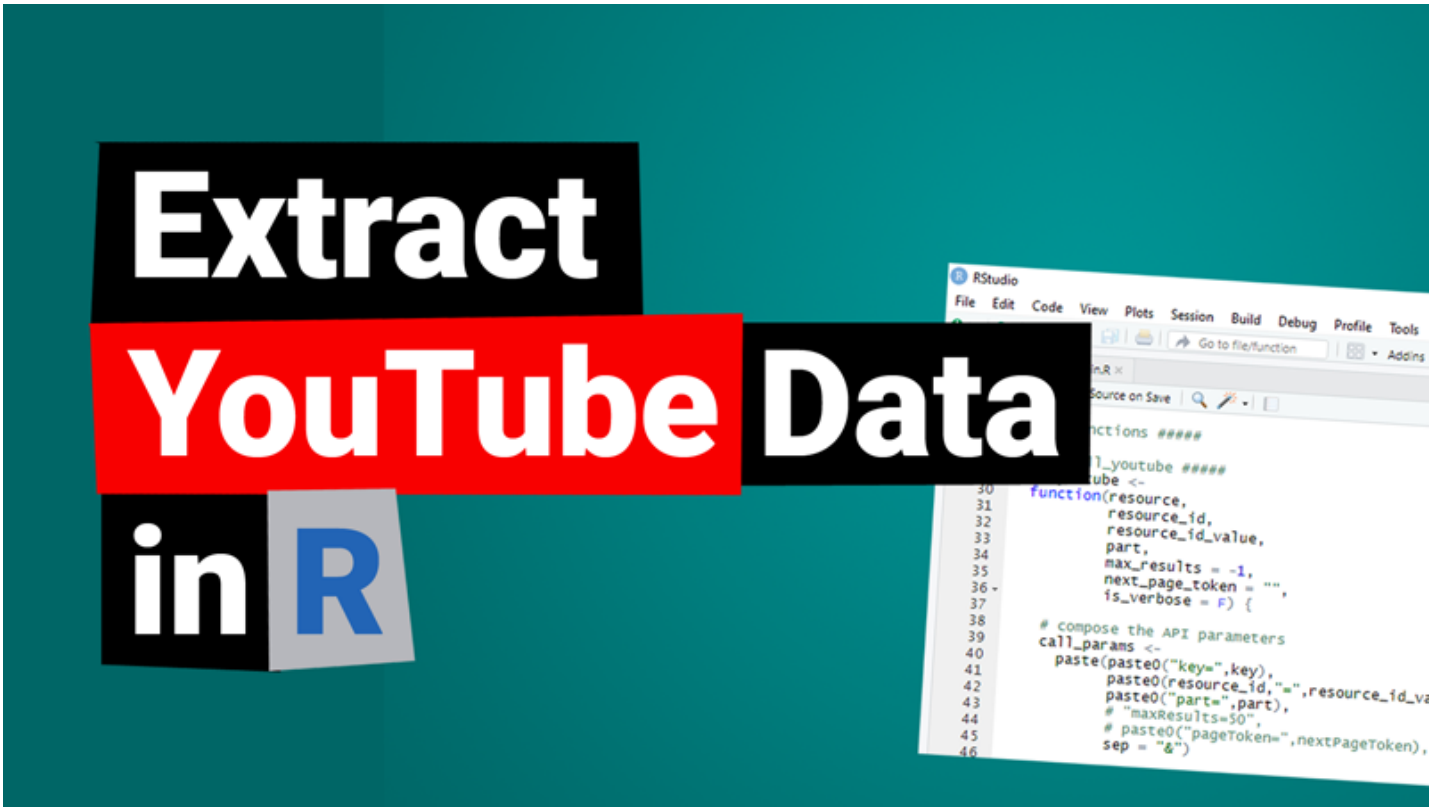
Yuichi Otsuka

How to Extract Data from YouTube using R and the YouTube API

August 5, 2020



In this guide, we will learn how to extract data from YouTube in four steps. Using R and the YouTube Data API, we can get information from a channel or a video, such as the subscriber count, view statistics, and video information.

If you watch YouTube videos, you are just one of more than 2 BILLION USERS that have used this platform. **That's almost one-third of the internet population!** *(Facebook has 3 billion users)*

		$7.80	$8.45	$7.80
	$7.80	$24.50	$8.45	$7.80

From entertainment, education, and business, you will likely find something on YouTube that will keep you hooked and watching for more.

As a data analyst, I have always wondered how to harness all of this data at scale.

In this guide, we will discover how we can extract data from YouTube in four steps:

- **Step 1:** Get your YouTube API key for free
- **Step 2:** Install R
- **Step 3:** Sample YouTube Data API calls
- **Step 4:** Sample Code in R

The sample in Step 4 is a complete code. Simply edit the keys, the channels, and you are good to go.

Table of Contents

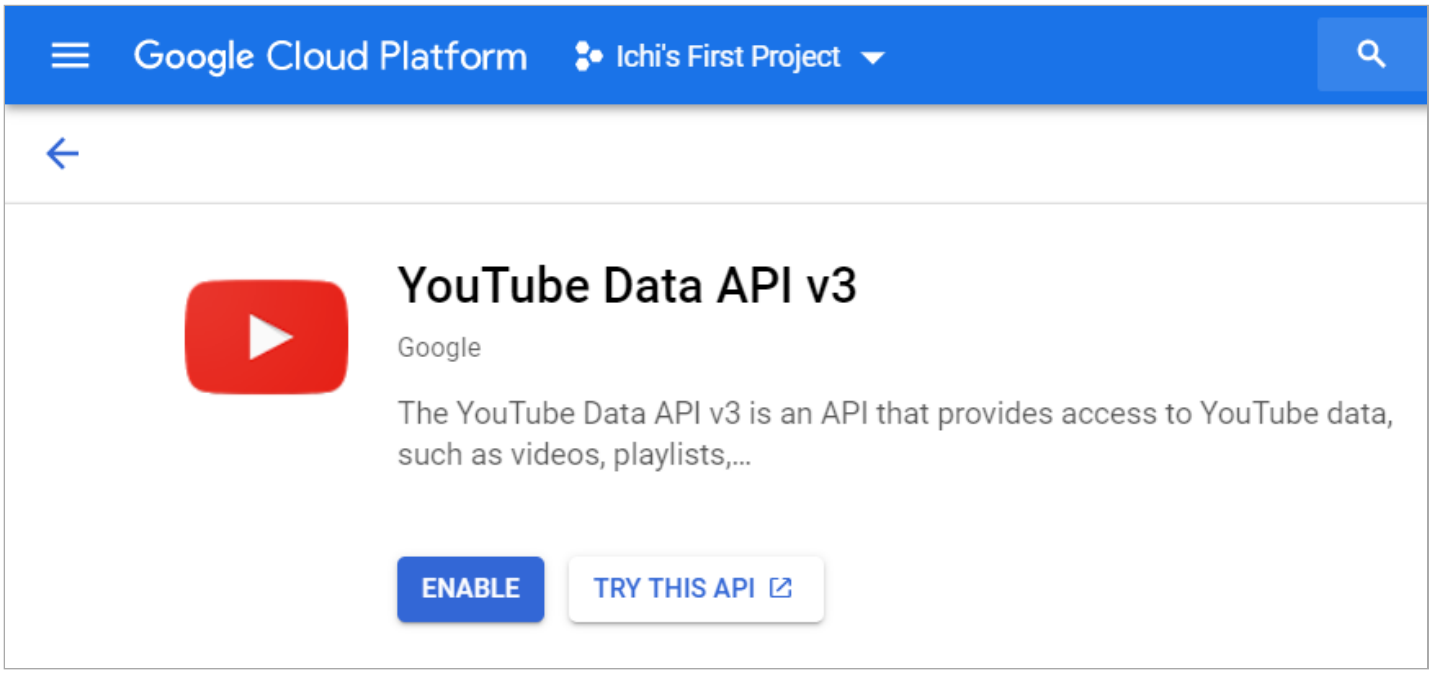
Step 1: Get a YouTube API key (Free)

YouTube shares information through their product, the YouTube Data API.

This product is part of the *Google Cloud Platform (GCP)*. In order to legally extract data from YouTube, you need to sign-up for a **Google *Cloud* account**, using your **Google account**.

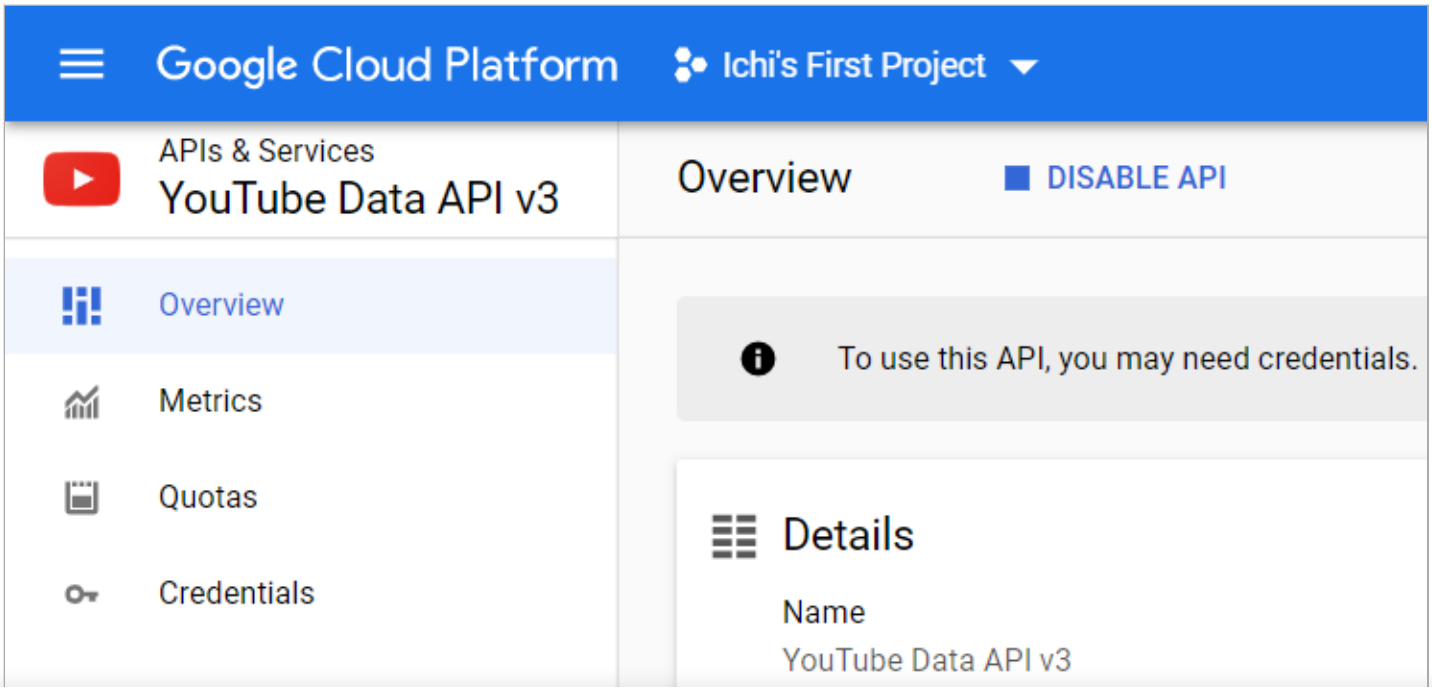
Accessing the following link will take you to the page for YouTube Data API v3. This will also include sign-up options if you are not yet a Google Cloud member.

If you are accessing this API for the first time, you will see this on your screen.



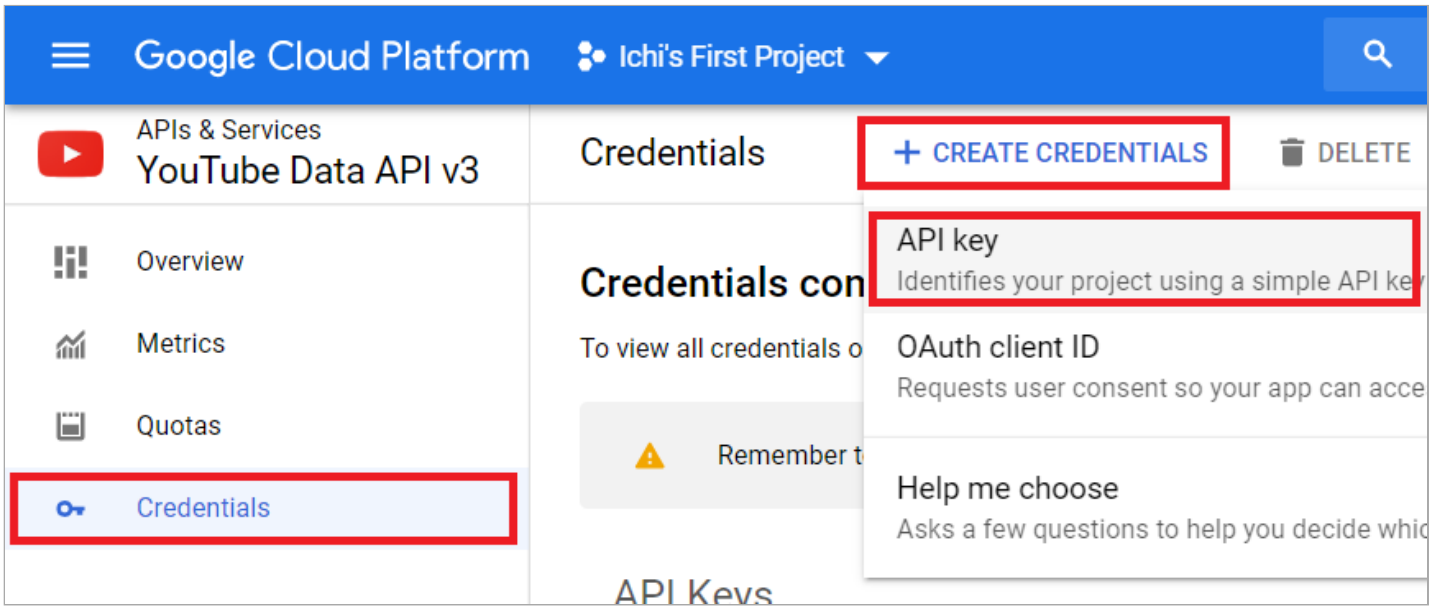
First-time API users will see this on their Google Cloud console.

Click **Enable**, and you should see the overview page the next time you visit.



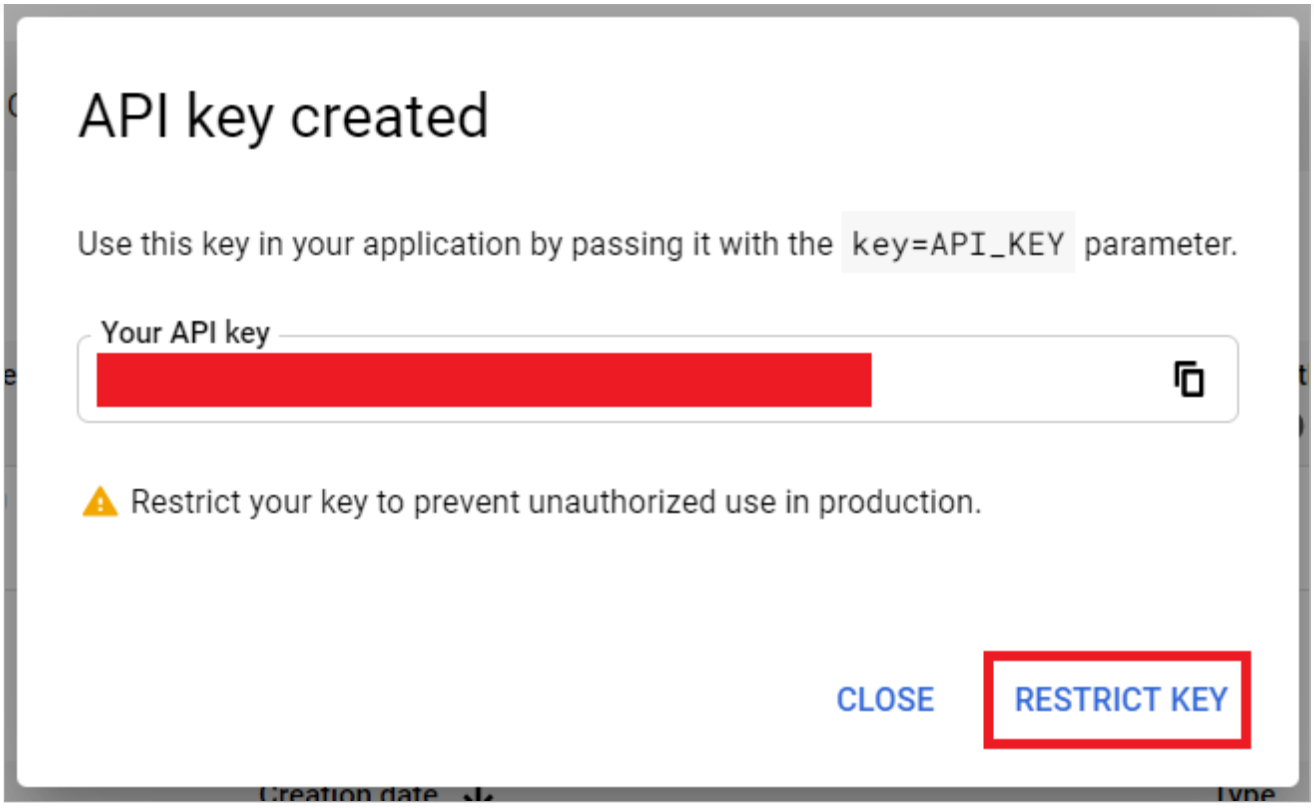
Let's start creating your YouTube API key.

Click on **Credentials** on the left sidebar. This will take you to the Credentials section. Click **Create Credentials** and select **API key**.



Creating an API key to be used for YouTube data extraction.

You will get a confirmation message, saying that your API key has been created. Click on the **Restrict Key** to ensure that your key will only be used to extract YouTube data.

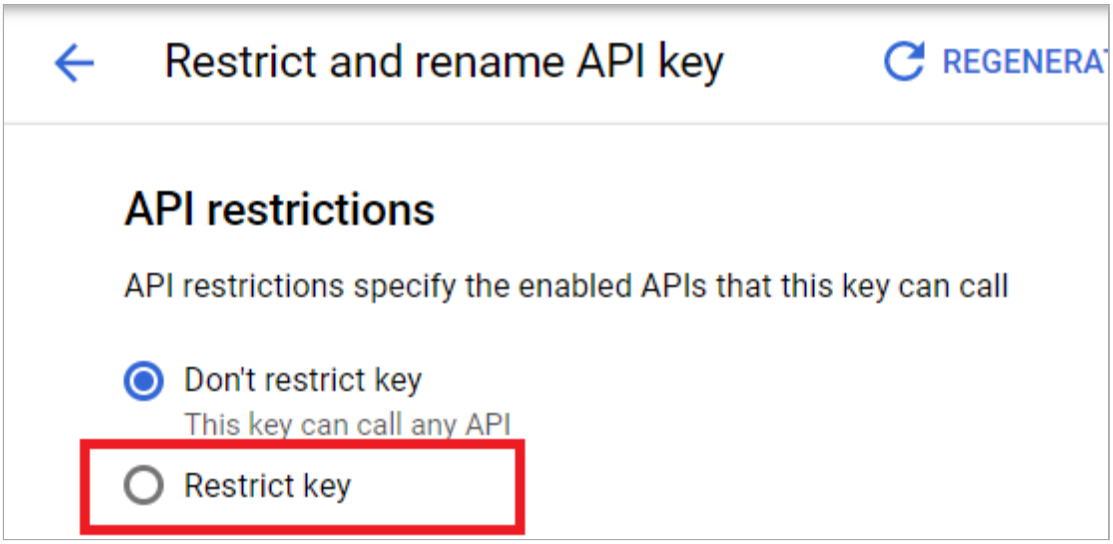


Successfully created the API key.

Important: Do not share this API key with other people. This prevents unauthorized use and potential abuse of your API key.

You will also notice that I do not share my own API key throughout this guide.

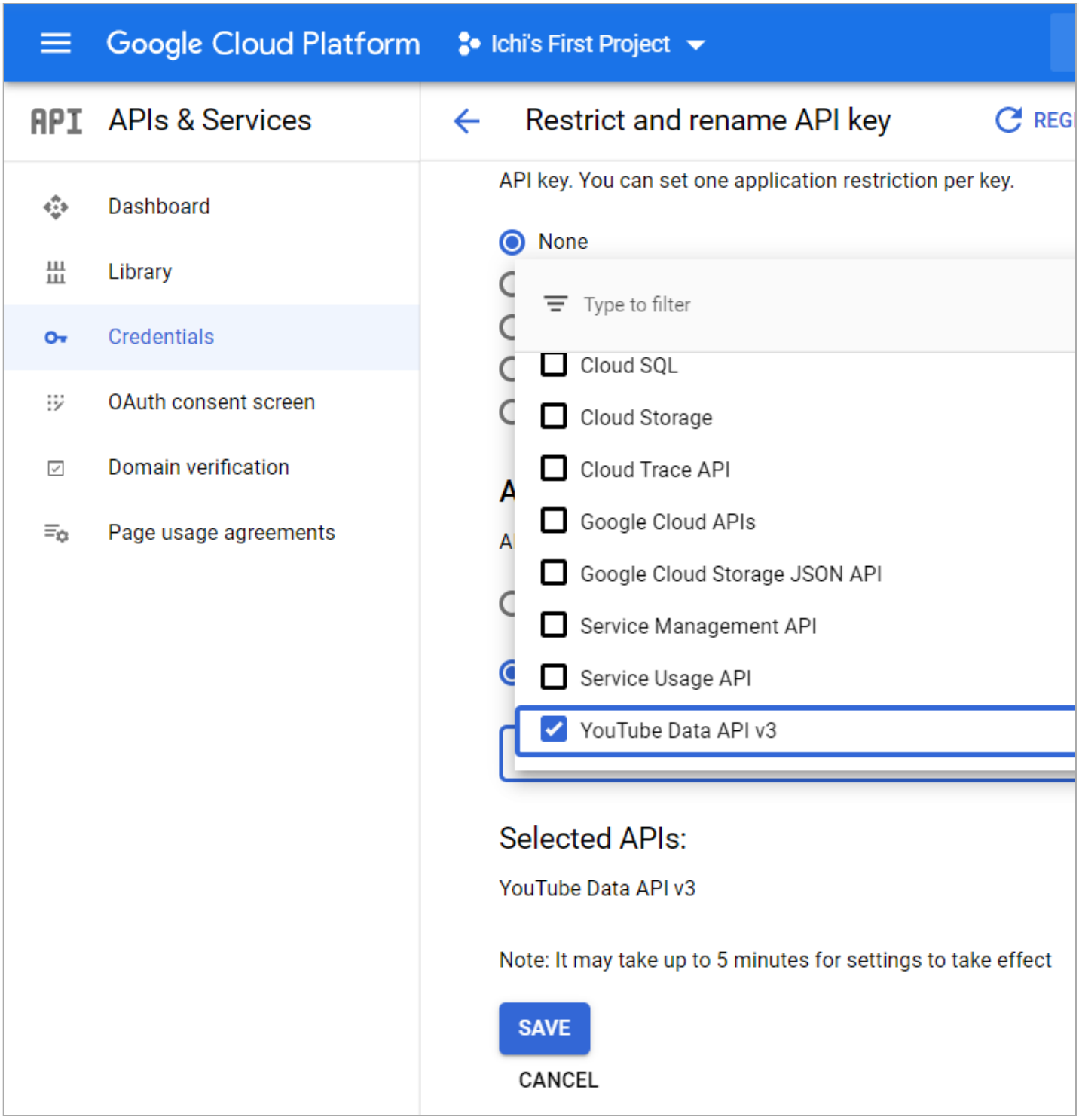
You will be taken to the **Restrict and rename API key** page.



Restrict your API key for YouTube.

Look for the section named **API restrictions** and select the **Restrict key** option. Here, we can choose specific products associated with this key.

Select **YouTube Data API v3**. Click **Save**.



You now have your API key!

Take note of the value of this API key. When we start making our sample codes below, you can add your key and start extracting YouTube data.

If you already have installed R, you can skip this section.

R is a tool for data analysis. Here, we can perform statistical analysis, visualize data, and automate data processing.

You can download installers for Windows, Mac, and Linux on their official website.

httr, jsonlite, and dplyr packages

Once you have installed R, you need to install 3 additional packages: `httr`, `jsonlite`, and `dplyr`. I included this in the sample code.

The `httr` package allows us to communicate with the API and get the raw data from YouTube in JSON format.

Afterwards, the `jsonlite` package takes this raw data and transforms it into a readable format.

The `dplyr` is an all-around package for manipulating data in R.

Live streaming videos do not count views. In some videos, comments are disabled. Since videos in YouTube can have different properties, some results are not consistent with the others.

Step 3: Sample YouTube Data API Calls (Optional)

In this step, we explore some examples of the API calls that we will use to extract data from YouTube.

We will see what an API call looks like, as well as some sample results.

I used Postman to explore and test out these API calls.

YouTube Data Basic Terms

Let's consider two YouTube channels – **CS Dojo**, a *Programming and Computer Science channel*, and **Numberphile**, a *Mathematics channel*.

CS Dojo: https://www.youtube.com/channel/UCxX9wt5FWQUAAz4UrysqK9A
Numberphile: https://www.youtube.com/user/numberphile

Notice that each channel has different URL format, and we can *identify a channel* by their **Channel ID** or **Username**.

`UCxX9wt5FWQUAAz4UrysqK9A`.

On the other hand, **Numberphile** follows the username format. In this case, the *Username* is `numberphile`.

For YouTube videos, each of them is identified by their **Video ID**, and we can easily see this in the URL.

```
https://www.youtube.com/watch?v=bI5jpueiCWw
```

Here, the *Video ID* is `bI5jpueiCWw`.

How Does an API Call Look Like?

A YouTube Data API call has the following format:

```
https://www.googleapis.com/youtube/v3/{resource}?{parameters}
```

The **`{resource}`** tells us what kind of information we want to extract from YouTube. In our examples throughout this guide, we extract data from three major resources:

- `channels` to get channel information
- `playlistItems` to list all videos uploaded in a channel or by a user
- `videos` to get detailed video information

The **`{parameters}`** allow us to further customize the results. Multiple parameters are separated by an ampersand (&). We usually start with the following parameters:

- `key` (required) for your YouTube API key

Getting the Channel Information:

Based on Channel ID:

```
https://www.googleapis.com/youtube/v3/channels?
key=**********&id=UCxX9wt5FWQUAAz4UrysqK9A&part=snippet,contentDetails,statis
tics
```

Parameters:

- `id=UCxX9wt5FWQUAAz4UrysqK9A` (the Channel ID of CS Dojo)
- `part=snippet,contentDetails,statistics`

Based on Username:

```
https://www.googleapis.com/youtube/v3/channels?
key=**********&forUsername=numberphile&part=snippet,contentDetails,statistics
```

Parameters:

- `forUsername=numberphile` (the Username of Numberphile)
- `part=snippet,contentDetails,statistics`

Sample Output for CS Dojo:


```
{
  "kind": "youtube#channelListResponse",
  "etag": "5FIor9Xof7m3UK8GnHryg5Jg-ig",
  "pageInfo": {
    "totalResults": 1,
    "resultsPerPage": 1
  },
  "items": [
```

```
          "title": "CS Dojo",
          "description": "Hi there! My name is YK, and I make videos mostly a
          "customUrl": "csdojo",
          "publishedAt": "2016-02-26T01:49:30Z",
          "thumbnails": {
            "default": {
              "url": "https://yt3.ggpht.com/a/AATXAJxJwY29yXENbgxRO0WxVMtiWyt
              "width": 88,
              "height": 88
            },
            "medium": {
              "url": "https://yt3.ggpht.com/a/AATXAJxJwY29yXENbgxRO0WxVMtiWyt
              "width": 240,
              "height": 240
            },
            "high": {
              "url": "https://yt3.ggpht.com/a/AATXAJxJwY29yXENbgxRO0WxVMtiWyt
              "width": 800,
              "height": 800
            }
          },
          "localized": {
            "title": "CS Dojo",
            "description": "Hi there! My name is YK, and I make videos mostly
          },
          "country": "CA"
        },
        "contentDetails": {
          "relatedPlaylists": {
            "likes": "",
            "favorites": "",
            "uploads": "UUxX9wt5FWQUAAz4UrysqK9A",
            "watchHistory": "HL",
            "watchLater": "WL"
          }
        },
        "statistics": {
          "viewCount": "52080104",
          "commentCount": "0",
          "subscriberCount": "1430000",
          "hiddenSubscriberCount": false,
          "videoCount": "90"
        }
      }
    ]
}
```

Uploads Playlist ID. The value for `items#contentDetails#relatedPlaylists#uploads` is the "playlist" that contains all the uploaded videos by a user or a channel.

In this example, the `uploads` value is `UUxX9wt5FWQUAAz4UrysqK9A`, and we will use that in later examples.

Getting the Playlist Information:

Initial API Call:

```
https://www.googleapis.com/youtube/v3/playlistItems?
key=**********&playlistId=UUxX9wt5FWQUAAz4UrysqK9A&part=snippet,contentDetail
s,status&maxResults=2
```

Parameters:

- `id=UUxX9wt5FWQUAAz4UrysqK9A` (the Playlist ID of CS Dojo Uploads)
- `part=snippet,contentDetails,statistics`
- `maxResults=2` (show at most 2 videos at a time)

Succeeding Pages (via pageToken):

```
https://www.googleapis.com/youtube/v3/playlistId?
key=**********&playlistId=UUxX9wt5FWQUAAz4UrysqK9A&part=snippet,contentDetail
s,status&maxResults=2&pageToken=CAIQAA
```

Parameters:

- `maxResults=2` (show at most 2 videos at a time)
- `pageToken=CAIQAA` (retrieving results in batches, or "pages")

Sample Output for CS Dojo:


```
{
  "kind": "youtube#playlistItemListResponse",
  "etag": "udxsDzE2VxvdihCnCMbQcXWW0w8",
  "nextPageToken": "CAIQAA",
  "items": [
    {
      "kind": "youtube#playlistItem",
      "etag": "eM4VjwjWHWHjaSNE-Hw1qGAeDTE",
      "id": "VVV4WDl3dDVGV1FVQUF6NFVyeXNxSzlBLjEtbF9VT0ZpMVh3",
      "snippet": {
        "publishedAt": "2020-07-25T05:04:05Z",
        "channelId": "UCxX9wt5FWQUAAz4UrysqK9A",
        "title": "Introduction to Trees (Data Structures & Algorithms #9)",
        "description": "Here is my intro to the tree data structure!\n\nAnd
        "thumbnails": {
          "default": {
            "url": "https://i.ytimg.com/vi/1-l_UOFi1Xw/default.jpg",
            "width": 120,
            "height": 90
          },
          "medium": {
            "url": "https://i.ytimg.com/vi/1-l_UOFi1Xw/mqdefault.jpg",
            "width": 320,
            "height": 180
          },
          "high": {
            "url": "https://i.ytimg.com/vi/1-l_UOFi1Xw/hqdefault.jpg",
            "width": 480,
            "height": 360
          },
          "standard": {
            "url": "https://i.ytimg.com/vi/1-l_UOFi1Xw/sddefault.jpg",
            "width": 640,
            "height": 480
          },
          "maxres": {
            "url": "https://i.ytimg.com/vi/1-l_UOFi1Xw/maxresdefault.jpg",
            "width": 1280,
            "height": 720
```

```
          "position": 0,
          "resourceId": {
            "kind": "youtube#video",
            "videoId": "1-l_UOFi1Xw"
          }
        },
        "contentDetails": {
          "videoId": "1-l_UOFi1Xw",
          "videoPublishedAt": "2020-07-25T05:04:05Z"
        },
        "status": {
          "privacyStatus": "public"
        }
      },
      {
        "kind": "youtube#playlistItem",
        "etag": "wjH5CvGlE-K_Gquy_7L3fLs4XH8",
        "id": "VVV4WDl3dDVGV1FVQUF6NFVyeXNxSzlBLmJJNWpwdWVpQ1d3",
        "snippet": {
          "publishedAt": "2020-05-30T01:44:22Z",
          "channelId": "UCxX9wt5FWQUAAz4UrysqK9A",
          "title": "Why and How I Used Vue.js for My Python/Django Web App (a
          "description": "Here's why and how I used Vue.js for my Python/Djan
          "thumbnails": {
            "default": {
              "url": "https://i.ytimg.com/vi/bI5jpueiCWw/default.jpg",
              "width": 120,
              "height": 90
            },
            "medium": {
              "url": "https://i.ytimg.com/vi/bI5jpueiCWw/mqdefault.jpg",
              "width": 320,
              "height": 180
            },
            "high": {
              "url": "https://i.ytimg.com/vi/bI5jpueiCWw/hqdefault.jpg",
              "width": 480,
              "height": 360
            },
            "standard": {
              "url": "https://i.ytimg.com/vi/bI5jpueiCWw/sddefault.jpg",
              "width": 640,
              "height": 480
            },
            "maxres": {
              "url": "https://i.ytimg.com/vi/bI5jpueiCWw/maxresdefault.jpg",
```

```
          "channelTitle": "CS Dojo",
          "playlistId": "UUxX9wt5FWQUAAz4UrysqK9A",
          "position": 1,
          "resourceId": {
            "kind": "youtube#video",
            "videoId": "bI5jpueiCWw"
          }
        },
        "contentDetails": {
          "videoId": "bI5jpueiCWw",
          "videoPublishedAt": "2020-05-30T01:44:22Z"
        },
        "status": {
          "privacyStatus": "public"
        }
      }
    ],
    "pageInfo": {
      "totalResults": 90,
      "resultsPerPage": 2
    }
  }
```


Getting the Video Information:

Based on Video ID:

```
https://www.googleapis.com/youtube/v3/videos?
key=**********&id=bI5jpueiCWw&part=id,contentDetails,statistics
```

Parameters:

- `id=bI5jpueiCWw` (the Video ID)
- `part=id,contentDetails,statistics`

Sample Output for a CS Dojo video:


```
{
  "kind": "youtube#videoListResponse",
```

```
        "id": "bI5jpueiCWw",
        "contentDetails": {
          "duration": "PT20M34S",
          "dimension": "2d",
          "definition": "hd",
          "caption": "false",
          "licensedContent": true,
          "contentRating": {},
          "projection": "rectangular"
        },
        "statistics": {
          "viewCount": "79084",
          "likeCount": "2075",
          "dislikeCount": "52",
          "favoriteCount": "0",
          "commentCount": "349"
        }
      }
    ],
    "pageInfo": {
      "totalResults": 1,
      "resultsPerPage": 1
    }
  }
```


Step 4: Sample Code for Extracting YouTube Data in R

Here's an example that we can use to extract data from a YouTube channel.

First, set the key variable with your YouTube API key.

```
key <- "… add your YouTube API key here …"
```

Next, I recommend setting up variables that you will frequently use throughout the script.

```
channel_id <- "UCxX9wt5FWQUAAz4UrysqK9A"  # CS Dojo Channel ID
user_id <- "numberphile"  # Numberphile Username
base <- "https://www.googleapis.com/youtube/v3/"
```

Set your working directory if you want to save the output. Use a forward slash instead of a

Load and install the needed packages. I recommend changing the `repos` depending on where you are.

```
required_packages <- c("httr", "jsonlite", "here", "dplyr")
for(i in required_packages) {
  if(!require(i, character.only = T)) {
    #  if package is not existing, install then load the package
    install.packages(i, dependencies = T, repos = "http://cran.us.r-project
    # install.packages(i, dependencies = T, repos = "https://cran.stat.upd.
    require(i, character.only = T)
  }
}
```

Let's extract the channel information for CS Dojo using the `httr` package.

```
# Construct the API call
api_params <-
  paste(paste0("key=", key),
        paste0("id=", channel_id),
        "part=snippet,contentDetails,statistics",
        sep = "&")
api_call <- paste0(base, "channels", "?", api_params)
api_result <- GET(api_call)
json_result <- content(api_result, "text", encoding="UTF-8")
```

The `json_result` is the raw data in JSON format. Let's use the `jsonlite` package and format this raw data into a data frame.

```
# Process the raw data into a data frame
channel.json <- fromJSON(json_result, flatten = T)
channel.df <- as.data.frame(channel.json)
```

Some of the important columns in the `channel.df` data frame:

```
id
snippet.title
snippet.description
snippet.customUrl
snippet.publishedAt
snippet.country
contentDetails.relatedPlaylists.uploads
statistics.viewCount
statistics.commentCount
```

Let's use the `contentDetails.relatedPlaylists.uploads` as our Playlist ID.

```
playlist_id <- channel.df$contentDetails.relatedPlaylists.uploads
```

Now, since a playlist can contain a large number of videos, we need to break them down into "pages". We extract 50 videos at a time, until the API tells us that we have reached the last page.

```
# temporary variables
nextPageToken <- ""
upload.df <- NULL
pageInfo <- NULL

# Loop through the playlist while there is still a next page
while (!is.null(nextPageToken)) {
  # Construct the API call
  api_params <-
    paste(paste0("key=", key),
          paste0("playlistId=", playlist_id),
          "part=snippet,contentDetails,status",
          "maxResults=50",
          sep = "&")

  # Add the page token for page 2 onwards
  if (nextPageToken != "") {
    api_params <- paste0(api_params,
                         "&pageToken=",nextPageToken)
  }

  api_call <- paste0(base, "playlistItems", "?", api_params)
  api_result <- GET(api_call)
  json_result <- content(api_result, "text", encoding="UTF-8")
  upload.json <- fromJSON(json_result, flatten = T)

  nextPageToken <- upload.json$nextPageToken
  pageInfo <- upload.json$pageInfo

  curr.df <- as.data.frame(upload.json$items)
  if (is.null(upload.df)) {
    upload.df <- curr.df
  } else {
    upload.df <- bind_rows(upload.df, curr.df)
  }
}
```

At this point, each row in the upload.df data frame will have basic video information, such as the *Video ID*, *Title*, *Description*, and *Upload Date*.

If we need **video statistics**, such as the **number of views, likes, and comments**, we need to make the third set of API calls to the `videos` resource.

```
video.df<- NULL
# Loop through all uploaded videos
for (i in 1:nrow(upload.df)) {
  # Construct the API call
  video_id <- upload.df$contentDetails.videoId[i]
  api_params <-
    paste(paste0("key=", key),
          paste0("id=", video_id),
          "part=id,statistics,contentDetails",
          sep = "&")

  api_call <- paste0(base, "videos", "?", api_params)
  api_result <- GET(api_call)
  json_result <- content(api_result, "text", encoding="UTF-8")
  video.json <- fromJSON(json_result, flatten = T)

  curr.df <- as.data.frame(video.json$items)

  if (is.null(video.df)) {
    video.df <- curr.df
  } else {
    video.df <- bind_rows(video.df, curr.df)
  }
}
```

Finally, we can combine the information into a single data frame called `video_final.df`.

```
# Combine all video data frames
video.df$contentDetails.videoId <- video.df$id
video_final.df <- merge(x = upload.df,
                        y = video.df,
                        by = "contentDetails.videoId")
```

You can process your data further by select specific columns, or by arranging them in a specific order.

Finally, you can write your data frame into a file. I prefer to save two files – one for the *channel details*, and another for the *uploaded video details*.

```
write.csv(x = channel.df,
          row.names = F,
          file = "CS_Dojo_Channel.csv")
write.csv(x = video_final.df,
          row.names = F,
          file = "CS_Dojo_Uploads.csv")
```

Is YouTube Data API Free?

The API is free for the **first 2 million units of API calls per month**. That's way more than what you will need if you are analyzing a handful of channels at a time.

A YouTube channel with 200 videos uploaded makes over *600 units of API call*, for all the data that we have extracted in the examples.

For API calls **above 2 million**, you will be charged **$3 per 1 million API calls**.

If you are making **more than 1 billion** API calls per month, that is further reduced to **$1.50 per million API calls**.

EXTERNAL LINK: Google Cloud Endpoints Pricing

Alternatives to Using R

As we have seen in Step 3, as long as you can prepare the final API endpoint, you can extract data using any tool that can send *HTTP requests (GET)*.

I use **Postman** for quick and easy testing, and then I finalize my scripts in **R** to support automation.

You can also do this in other programming languages such as Java, JavaScript, or Python.

Aside from the `httr` and `jsonlite` packages, you can also use other existing packages.

The `tuber` package specializes in YouTube data analysis but **uses OAuth** *instead of an API key*.

Conclusion

We were able to automate data extraction in YouTube using R and the YouTube Data API.

This is my first R guide! If you learned something new, let me know in the comments below.

10 thoughts on “How to Extract Data from YouTube using R and the YouTube API”

FACEBOOK MARKETING

October 14, 2020 at 4:56 am

I’m extremely inspired with your writing abilities and also with the structure to your weblog. Is that this a paid subject matter or did you modify it yourself? Anyway stay up the excellent high quality writing, it is uncommon to see a nice weblog like this one these days.

Reply

KITCHEN TABLE

February 25, 2021 at 2:45 am

This is a good tip particularly to those fresh to the blogosphere.

Short but very accurate information? Thanks for sharing this one.

A must read article!

Reply

ALETHEA

March 6, 2021 at 2:43 am

I am not sure where you are getting your information, but good topic.

Reply

ARIF

November 1, 2022 at 5:24 am

Hi,

I don't see the Youtube Data API in the restrictions list. Maybe they changed it. Can you confirm?

Reply

ICHI OTSUKA

December 3, 2022 at 8:43 pm

Hi Arif,

I just checked now (December 3, 2022), and it's still there. I also tried creating a new API key, and the YouTube Data API v3 is the last one on the list.

Reply

BRYAN

November 2, 2022 at 6:11 am

Thanks for creating this blog with these tips. Only when I try the code I get no input from " # Loop through the playlist while there is still a next page" The upload.df and curr.df remain empty. Do you know if anything needs to be changed in the code?

Reply

Thank you for your information. This is very helpful.
There is the change the channel information.
contentDetails.relatedPlaylists.uploads –> items.contentDetails.relatedPlaylists.uploads
Thank you

Reply

ICHI OTSUKA

December 3, 2022 at 8:23 pm

Thanks a lot Tim! I will update the post when I get the time.

Reply

SATWINDER KAUR

December 1, 2022 at 7:19 am

I am getting this error , > video.df$contentDetails.videoId video_final.df <- merge(x = upload.df,
+ y = video.df,
+ by = "contentDetails.videoId")
Error in fix.by(by.x, x) : 'by' must specify a uniquely valid column

Do you have any idea?

I am so confused

Reply

ICHI OTSUKA

December 3, 2022 at 8:21 pm

The code snippet is just an example above. From the error, you need to check if your dataframe variable upload.df has the column “contentDetails.videoId”. You also have to make sure that it is in the correct capitalization, and it should be a unique column. If you have done any pre-processing, you can end up with different capitalization on the column names.

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment *

Name *

Email *

Website

Save my name, email, and website in this browser for the next time I comment.

POST COMMENT

Search …

Recent Posts

Working with BigQuery Table Partitions

How to Extract Data from YouTube using R and the YouTube API

How to Create a Table in BigQuery

10 Free Data Analysis Tools in 2020

Top 3 Skills for a Data Analyst

Categories

Data Analysis