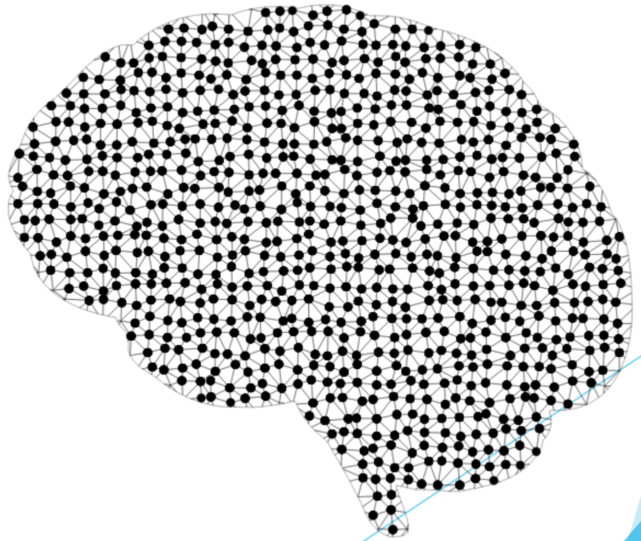
APIs and Data Storage in MATLAB

ENGAGED QUALITY INSTRUCTION THROUGH PROFESSIONAL DEVELOPMENT

EQuIPD

UF Herbert Wertheim College of Engineering
UNIVERSITY *of* FLORIDA



Gathering and Organizing Data

In the previous lesson, we learned that machine learning models require data to make predictions and inferences.

Exercises in lesson 2 required the user to import the data manually into MATLAB®; however, this method becomes increasingly inefficient when dealing with large amounts of data.

Get into groups and let's discuss the following questions:

- How do data scientists gather large amounts of data?
- How do you think that apps on your smartphone get data?
- Why must data be organized and sorted?

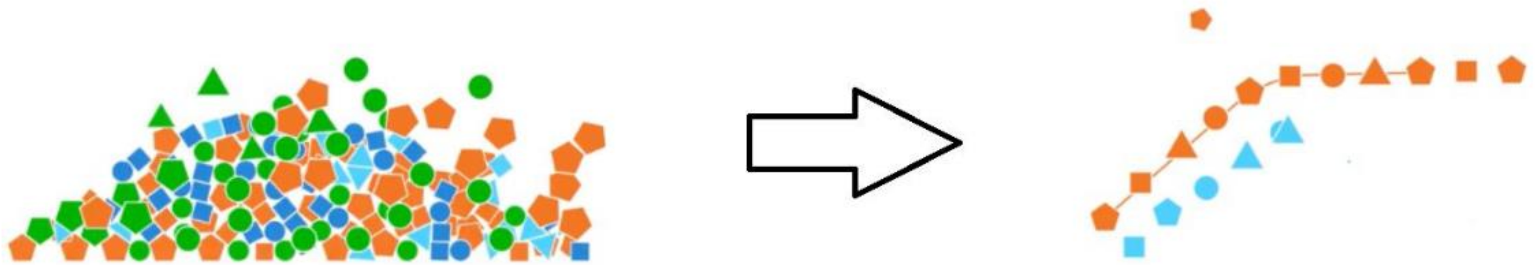
Relate your answers to your experiences.

Hint: Think about the apps used in your daily life and how those apps are organized. For example, Spotify allows the user to sort through music according to genre, artist, and popularity.

Gathering and Organizing Data

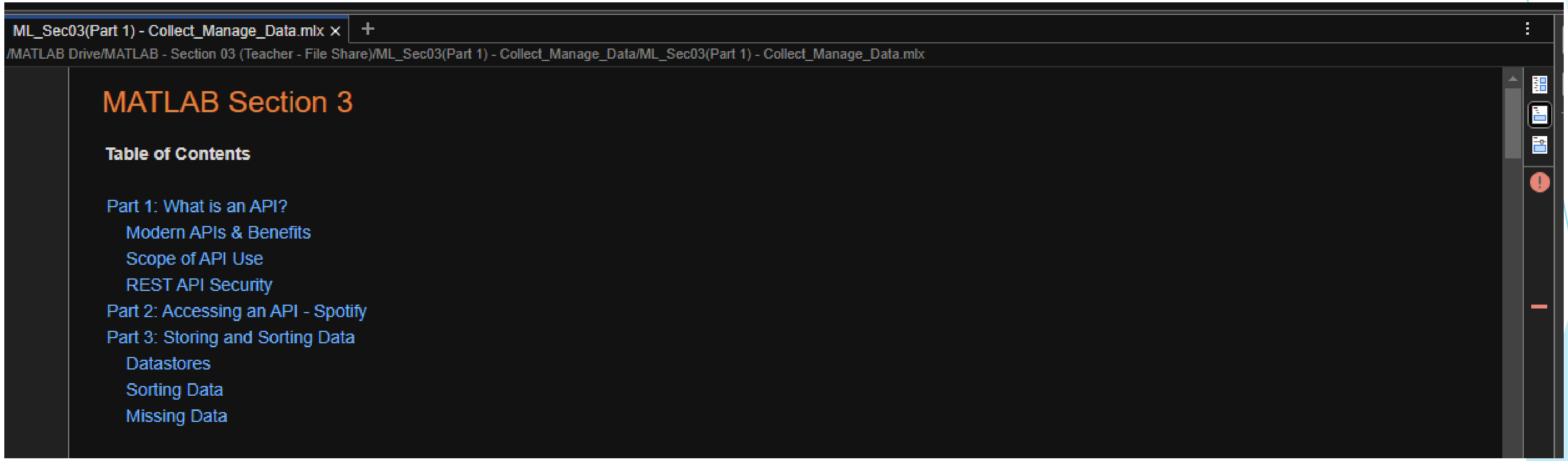
After discussing the questions let's introduce some key concepts for this lesson.

1. Data scientists obtain large amounts of data from various sources using a combination of techniques and tools. Some of the most common ways include application programming interfaces (APIs) and Web scraping. The focus of this lesson will be APIs.
2. APIs are mechanisms that enable 2 software components to communicate with each other using a set of definitions and protocols. One application of APIs is weather apps. The app (client) communicates with a weather database (server) via APIs to provide the user with weather updates.



Collecting and Managing Data

Please copy over the files for Section 03 from the MATLAB Drive



ML_Sec03(Part 1) - Collect_Manage_Data.mlx

/MATLAB Drive/MATLAB - Section 03 (Teacher - File Share)/ML_Sec03(Part 1) - Collect_Manage_Data/ML_Sec03(Part 1) - Collect_Manage_Data.mlx

MATLAB Section 3

Table of Contents

The MATLAB files for this section can be found at this link.

Collecting and Managing Data

What is an API?

In the beginning of this lesson, APIs are described as mechanisms that allow 2 software components to communicate with each other using a set of definitions and protocols.

A useful metaphor to understand APIs is thinking of them as a messenger, like how a waiter communicates your order to the kitchen.

Applications of APIs in daily life include ride-share apps, airline websites, and apps that allow you to control devices, like a thermostat. All of these apps send data to a server so that it can perform a series of actions to fulfill your request.

Collecting and Managing Data

Modern APIs & Benefits

The most widely used and flexible APIs are known as **Representational State Transfer** (REST) APIs. In REST APIs, a server uses input from the client to perform internal functions and return output data. Data is encoded and transported between the clients and servers using **Hypertext Transfer Protocol** (HTTP).

The main benefits of REST APIs are integration and ease of maintenance. REST APIs allow existing software systems to be integrated to new applications without having to make changes to the code of the two systems.

Another important feature of APIs are endpoints. API endpoints include server URLs, services, and other digital locations from where information is sent and received between systems

Collecting and Managing Data

Scope of API Use

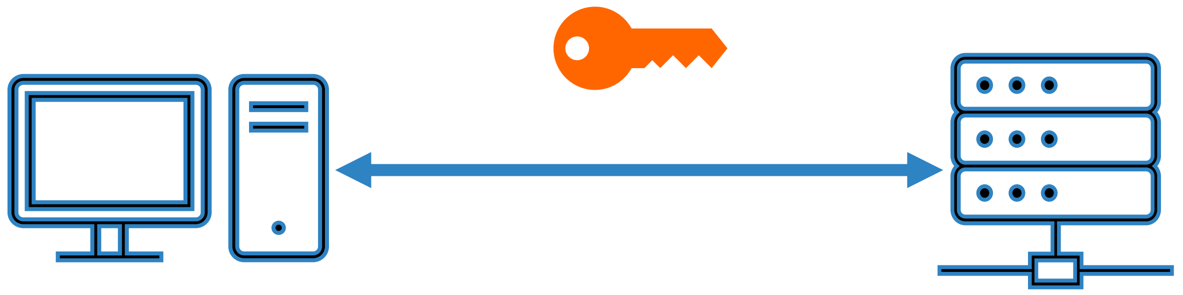
- **Private API**: Used only for connecting systems and transporting data within a single business.
- **Public API**: Open to the public but may have some authorization restrictions associated with them.
- **Partner API**: Available only to authorized developers to facilitate business-to-business partnerships.
- **Composite API**: A combination of two or more of the types above to meet complex system requirements.

Collecting and Managing Data

REST API Security

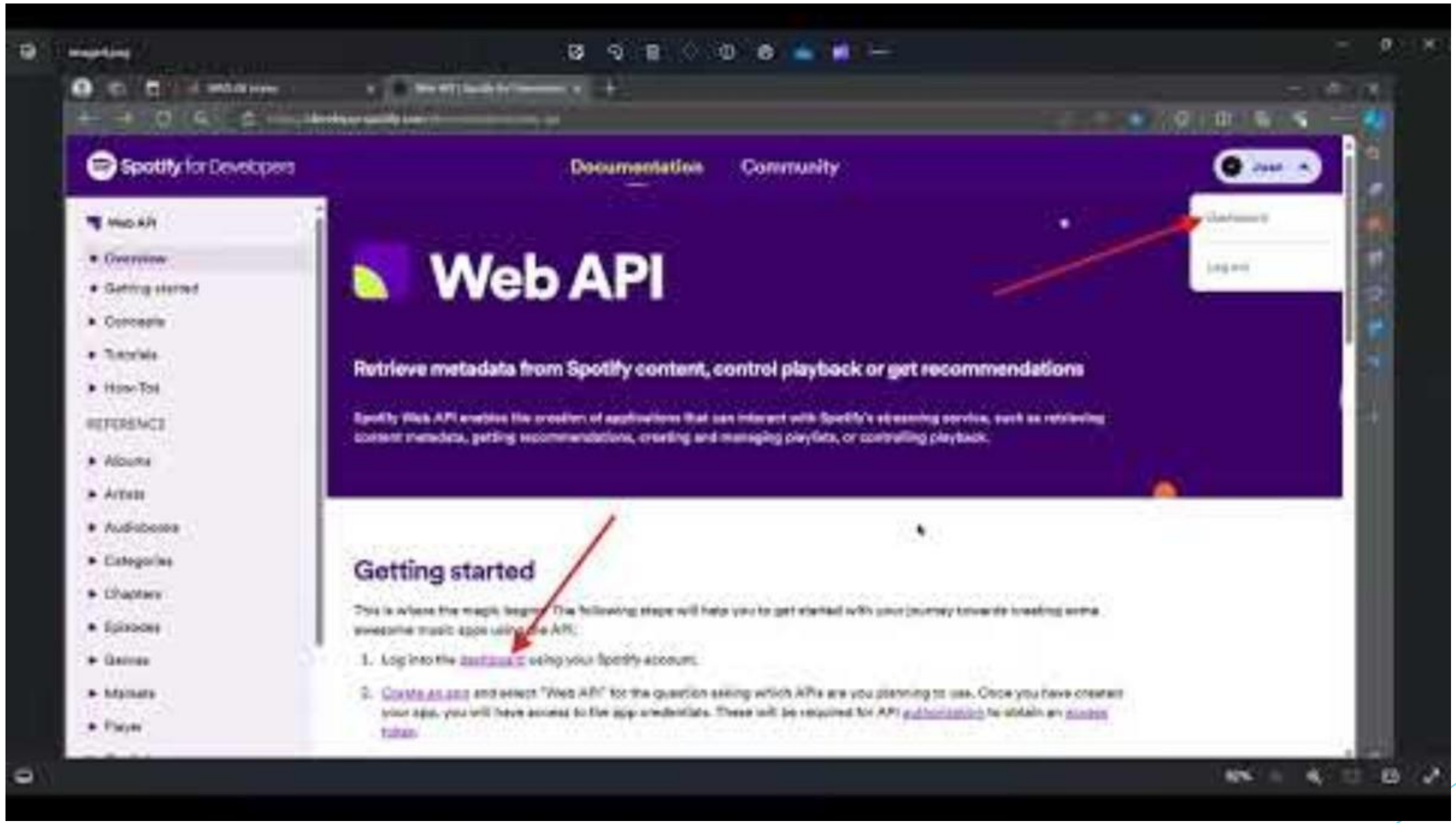
REST APIs are secured using authentication tokens and API keys. Authentication tokens authorize users to make a specific API call and verify the user's identity before processing an API call.

API keys verify the program or app making the API call to ensure it has the permission to make a specific call. API keys allow API monitoring to gather data on usage. API keys appear in browser URLs as a long string of characters and numbers and are used by the website to make internal API calls.



Creating a Spotify Developer Account for the API

In this section, we will be accessing the Spotify API to gather music data. Please watch the video below to make an account.



Web API

Retrieve metadata from Spotify content, control playback or get recommendations

Spotify Web API enables the creation of applications that can interact with Spotify's streaming service, such as retrieving content metadata, getting recommendations, creating and managing playlists, or controlling playback.

Getting started

This is where the magic begins! The following steps will help you to get started with your journey towards creating some awesome music apps using the API.

1. Log into the dashboard using your Spotify account.
2. Create an app and select "Web API" for the question asking which APIs are you planning to use. Once you have created your app, you will have access to the app credentials. These will be required for API authorization to obtain an access token.

The MATLAB files for this section can be found at this link.

Creating a Spotify Developer Account for the API

The steps below are also described in video on the previous slide

1. Follow this link to the Spotify Developer website: Web API | Spotify for Developers
2. Log in with an existing Spotify account or create account.
3. Go to developer **Dashboard** to and click on **Create app.**
4. Input a name and description for your app.
5. Create a URI using the following format: **http://localhost:8081**. Any combination of numbers is valid. Click the **Add** button.
6. Check the boxes for **Web API** and the **Terms of Service**. Click the **Save** button.
7. Go to the app's **Settings** to retrieve your unique **Client ID** and **Client Secret**. You will use these to run the code in the MATLAB files provided.

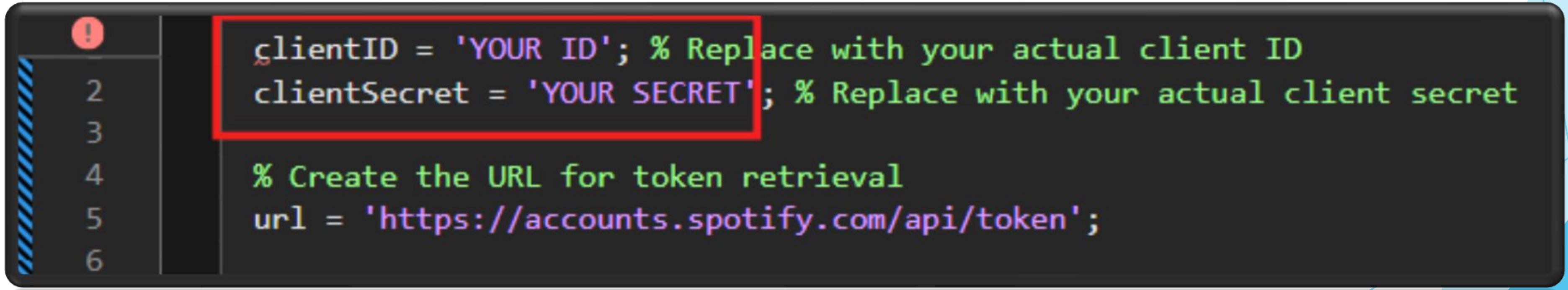
Collecting and Managing Data

Accessing an API - Spotify

The functionalities of the Spotify Web API include:

- ➢ Retrieving data from an artists, albums, or shows
- ➢ Searching for Spotify content
- ➢ Getting recommendations based on the music you listen to

To access the API, you need get an access token using your **Client ID** and **Client Secret.**



```
clientID = 'YOUR ID'; % Replace with your actual client ID
clientSecret = 'YOUR SECRET'; % Replace with your actual client secret

% Create the URL for token retrieval
url = 'https://accounts.spotify.com/api/token';
```

Collecting and Managing Data

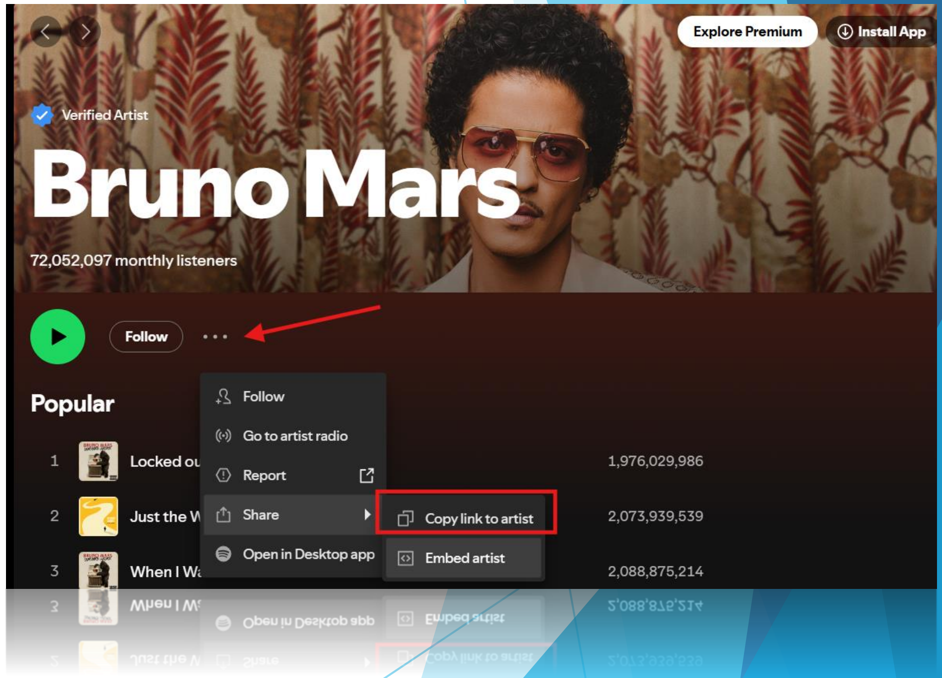
Accessing an API - Spotify

Now that we have an access token we can start retrieving data. In this example we will retrieve data from four Artists: Kendrick Lamar, Drake, Taylor Swift, and Billie Eilish.

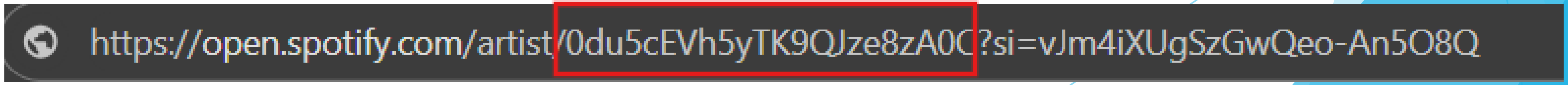
First, we need to construct the URL for each of their Spotify profiles using their **Spotify IDs**. The Spotify ID is a unique string of characters used to identify an artist on Spotify.

Here's how to find it:

- Go to the Artist's Page on Spotify and click on the 3 horizontal dots below the artist's name.
- Select Share from the menu and click Copy link to artist.
- Paste this link into a browser.
- The Spotify ID is the string of characters located between ***/artist/*** and the **?**



1	Locked ou	1,976,029,986
2	Just the W	2,073,939,539
3	When I Wa	2,088,875,214

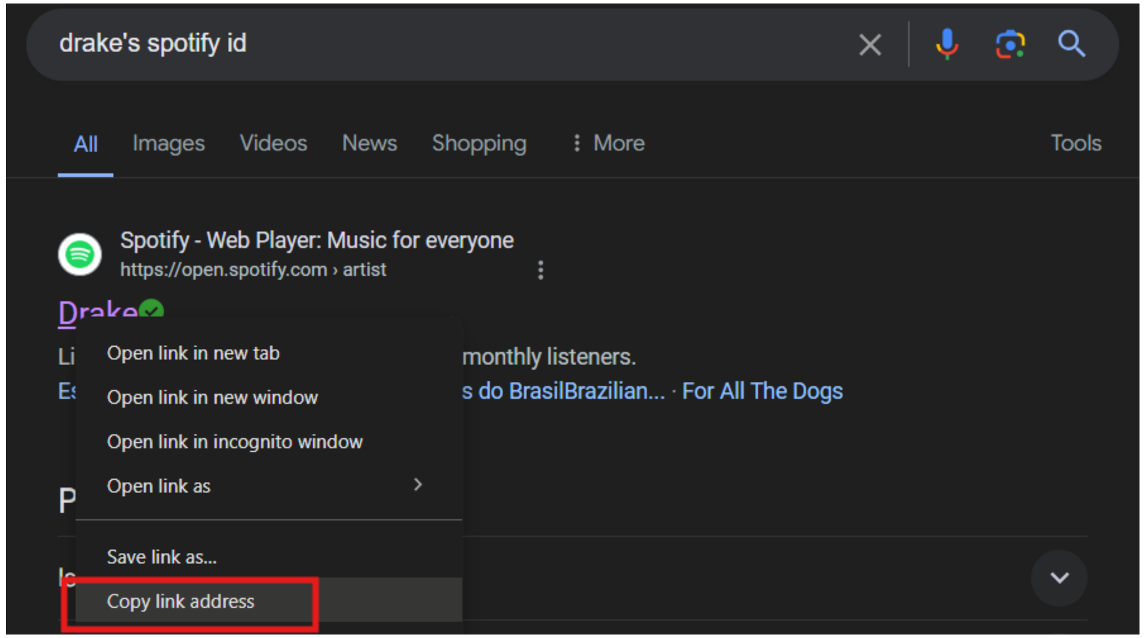


https://open.spotify.com/artist/0du5cEVh5yTK9QJze8zA0Q?si=vJm4iXUgSzGwQeo-An5O8Q

Collecting and Managing Data

Accessing an API - Spotify

If you don't have access to Spotify you can instead Google search the artist's ID and right click to copy the link address to their Spotify page. Paste their link into the MATLAB live script and delete everything except their Spotify ID.



Spotify - Web Player: Music for everyone
https://open.spotify.com › artist
Drake
monthly listeners.
s do BrasilBrazilian... · For All The Dogs

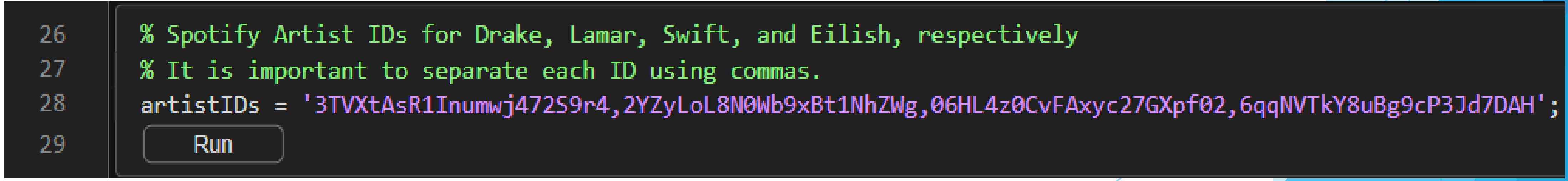
Here is the link to Drake's Spotify page, which includes his Spotify ID in bold at the end of the link: https://open.spotify.com/artist/**3TVXtAsR1Inumwj472S9r4**

Collecting and Managing Data

Accessing an API - Spotify

Once we have all Spotify IDs, we need to store them as a comma-separated list and assign them to a variable. According to Spotify, you can access the profiles of a maximum 100 artists at once. Run the code snippet below to store the IDs into the variable **artistIDs.**

"Drake July 2016" by The Come Up Show is licensed under CC BY 2.5.

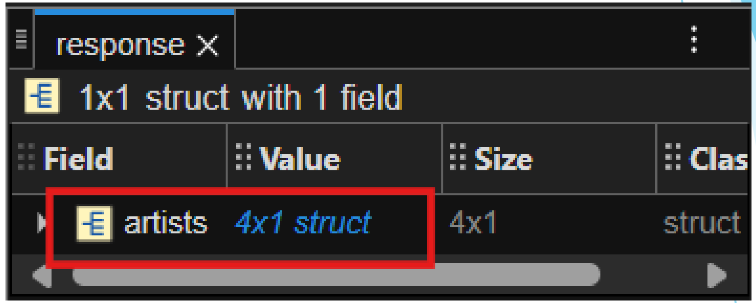


```
% Spotify Artist IDs for Drake, Lamar, Swift, and Eilish, respectively
% It is important to separate each ID using commas.
artistIDs = '3TVXtAsR1Inumwj472S9r4,2YZyLoL8N0Wb9xBt1NhZWg,06HL4z0CvFAxyc27GXpf02,6qqNVTkY8uBg9cP3Jd7DAH';
```

Collecting and Managing Data

Accessing an API - Spotify

To see what data was retrieved, we can click on the variable "**response**" in the workspace. Then click "**artists**.

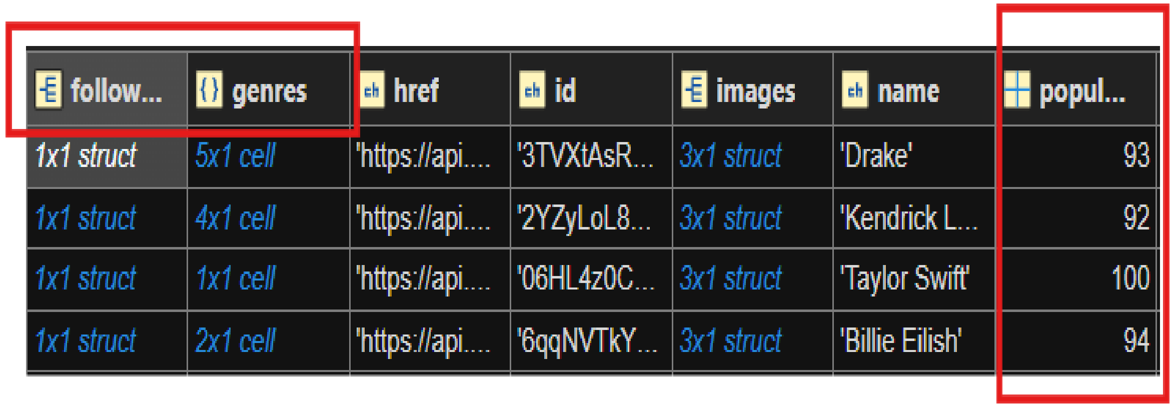


response

1x1 struct with 1 field

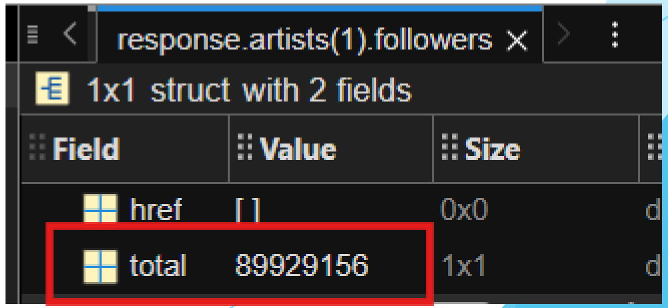
Field	Value	Size	Clas
artists	*4x1 struct*	4x1	struct

Below is the data retrieved. Here we can see the artists' popularity indices, which is a number out of 100 that Spotify uses to rank artists. There is also information about their followers and genres of their songs.



follow...	genres	href	id	images	name	popul...
1x1 struct	*5x1 cell*	'https://api....	'3TVXtAsR...	*3x1 struct*	'Drake'	93
1x1 struct	*4x1 cell*	'https://api....	'2YZyLoL8...	*3x1 struct*	'Kendrick L...	92
1x1 struct	*1x1 cell*	'https://api....	'06HL4z0C...	*3x1 struct*	'Taylor Swift'	100
1x1 struct	*2x1 cell*	'https://api....	'6qqNVTkY...	*3x1 struct*	'Billie Eilish'	94

Click on the *1x1 struct* under **"followers"** to see the number of followers an artists has.



response.artists(1).followers

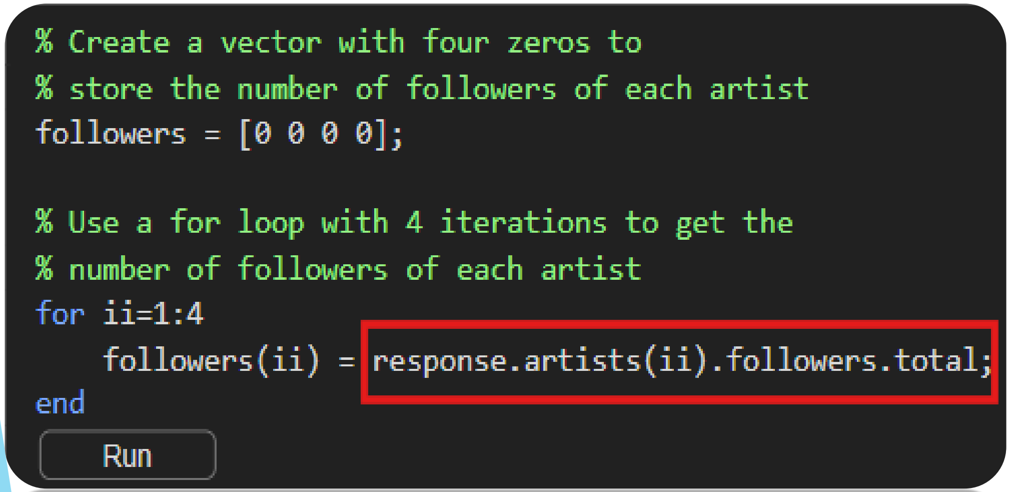
1x1 struct with 2 fields

Field	Value	Size
href	[]	0x0
total	89929156	1x1

Collecting and Managing Data

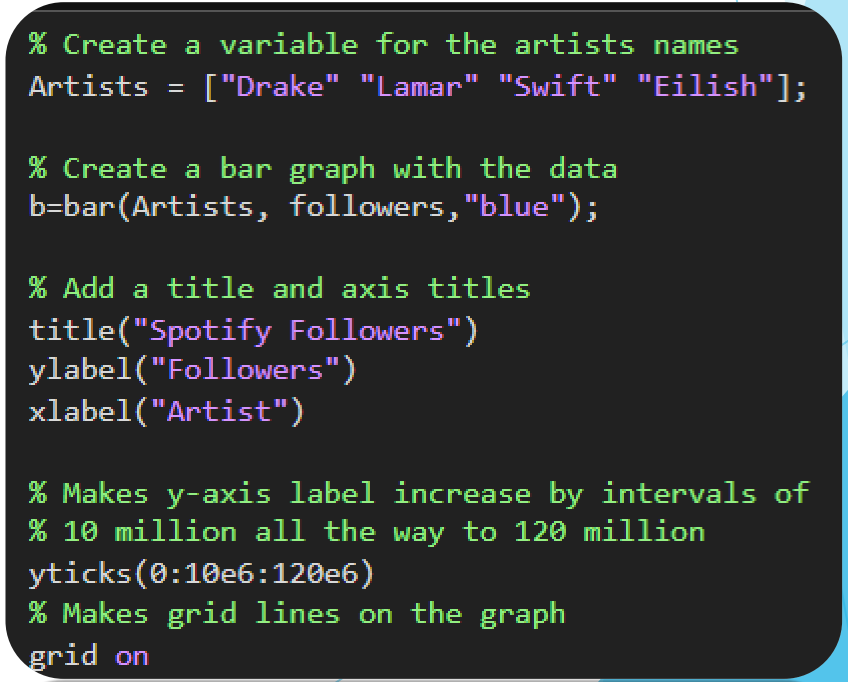
Accessing an API - Spotify

Let's use the data to create a bar graph showing how many followers each artists has on Spotify with the code below. We will be using a for loop and dot notation to retrieve the number of followers for each artists from the variable **response**.



```
% Create a vector with four zeros to
% store the number of followers of each artist
followers = [0 0 0 0];

% Use a for loop with 4 iterations to get the
% number of followers of each artist
for ii=1:4
    followers(ii) = response.artists(ii).followers.total;
end
```

Now let's run the code to make the bar graph. The resulting graph is shown in the next slide.



```
% Create a variable for the artists names
Artists = ["Drake" "Lamar" "Swift" "Eilish"];

% Create a bar graph with the data
b=bar(Artists, followers,"blue");

% Add a title and axis titles
title("Spotify Followers")
ylabel("Followers")
xlabel("Artist")

% Makes y-axis label increase by intervals of
% 10 million all the way to 120 million
yticks(0:10e6:120e6)
% Makes grid lines on the graph
grid on
```

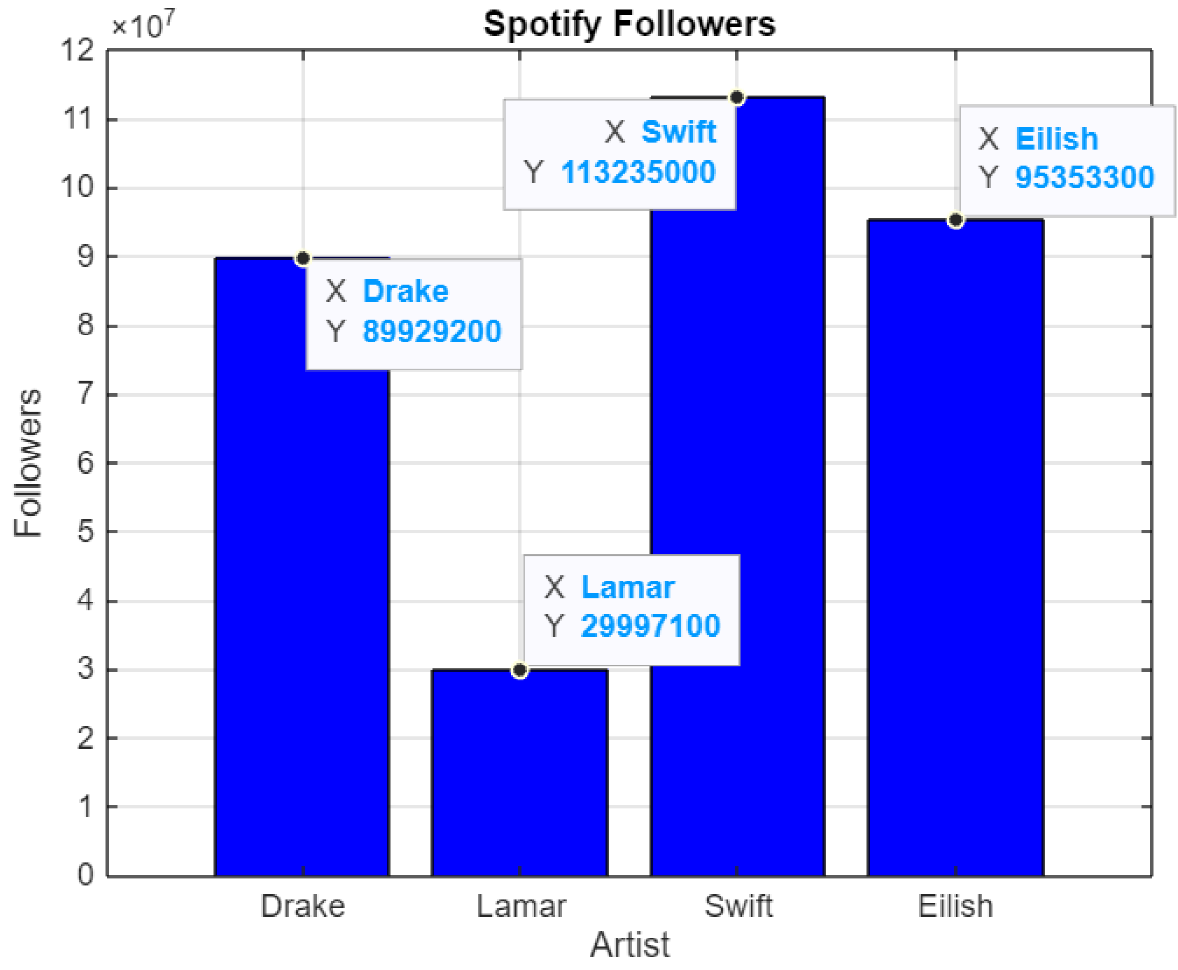
Collecting and Managing Data

Accessing an API - Spotify

The y-axis on the bar graph shows the number of followers in the magnitude of millions.

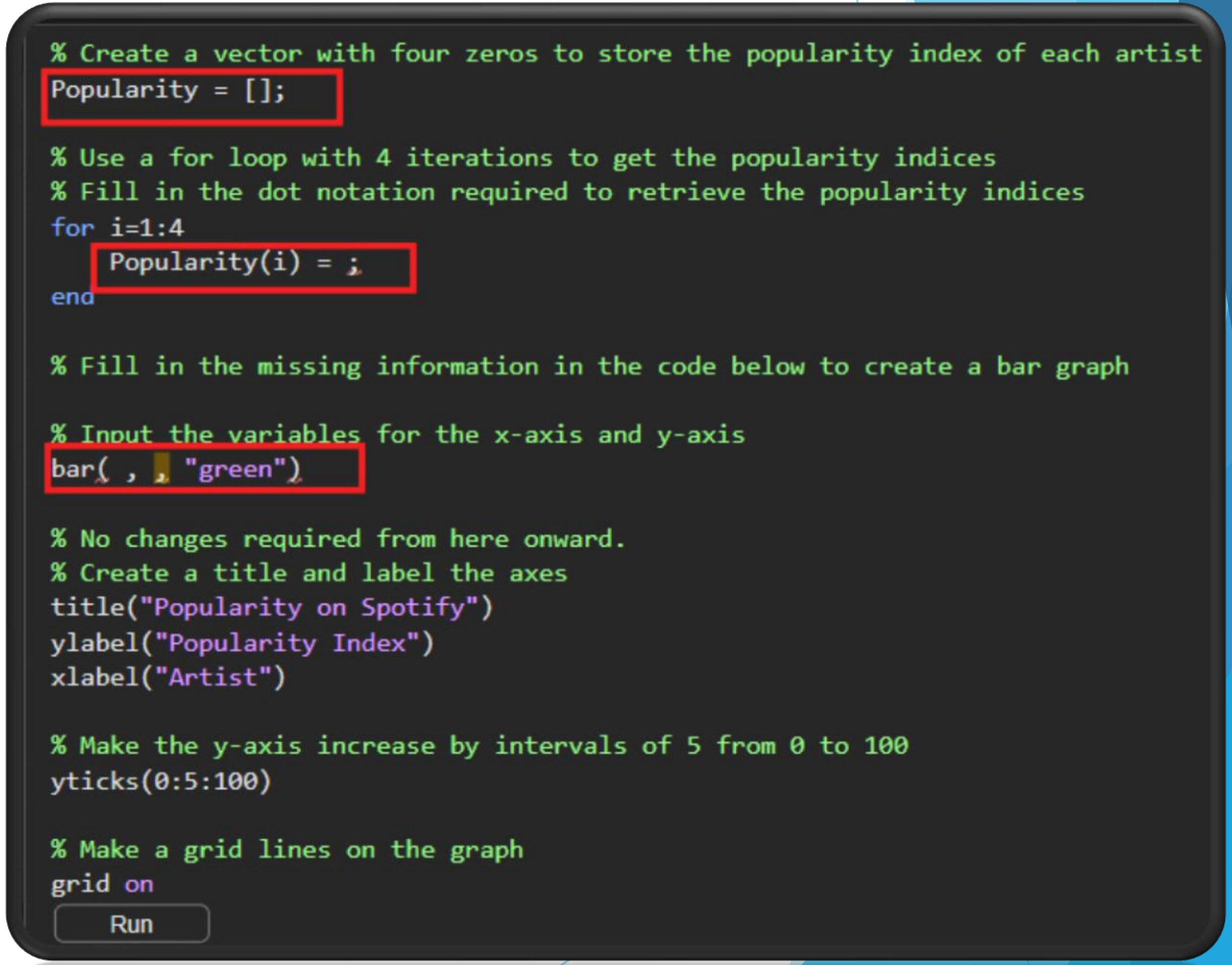
1. Taylor Swift: 113,235,000
2. Billie Eilish: 95,353,300
3. Drake: 89,929,200
4. Kendrick Lamar: 29,997,100

You can click on the graph to add labels to each bar.



Collecting and Managing Data

Accessing an API - Spotify



```
% Create a vector with four zeros to store the popularity index of each artist
Popularity = [];

% Use a for loop with 4 iterations to get the popularity indices
% Fill in the dot notation required to retrieve the popularity indices
for i=1:4
    Popularity(i) = ;
end

% Fill in the missing information in the code below to create a bar graph

% Input the variables for the x-axis and y-axis
bar( , , "green")

% No changes required from here onward.
% Create a title and label the axes
title("Popularity on Spotify")
ylabel("Popularity Index")
xlabel("Artist")

% Make the y-axis increase by intervals of 5 from 0 to 100
yticks(0:5:100)

% Make a grid lines on the graph
grid on
```

Now you try!

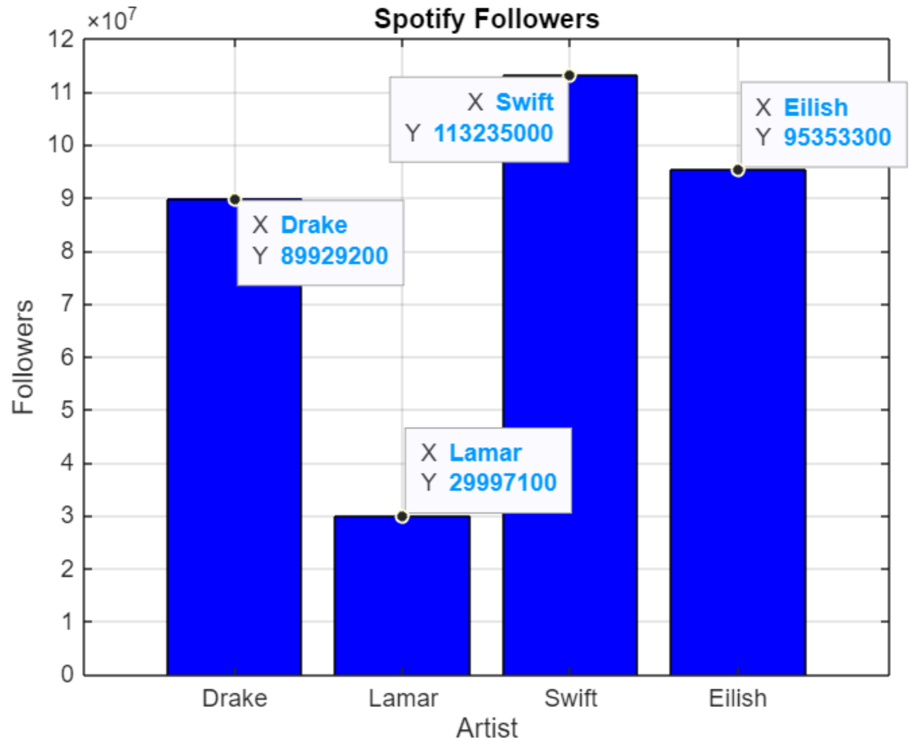
Fill in the missing code in the red boxes to create a bar graph comparing the popularity index of each artist.

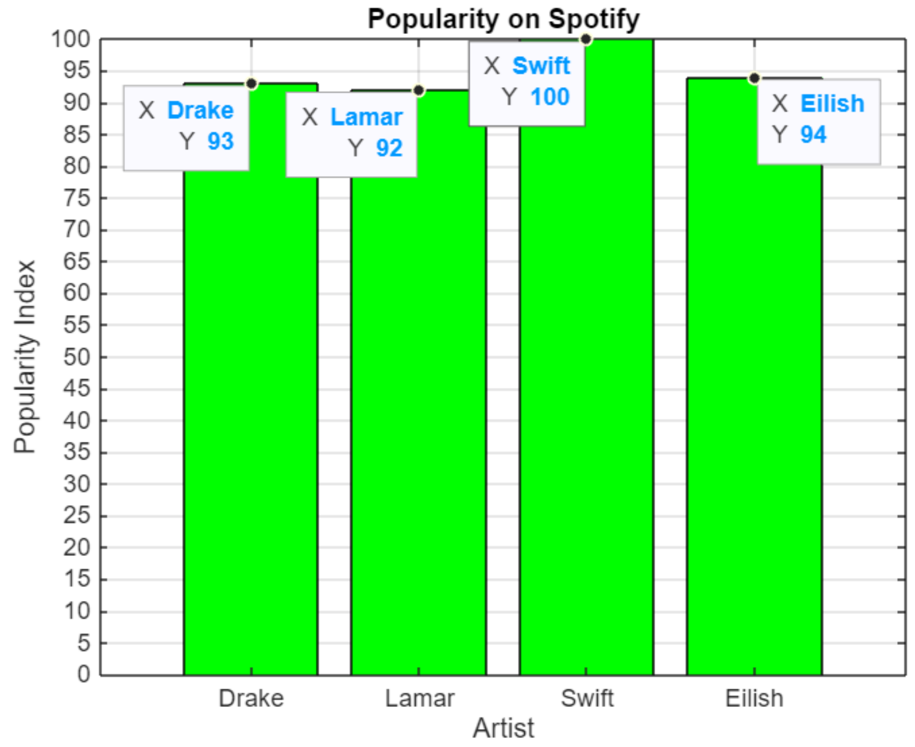
*Hint: Use a for a loop and the dot notation: **response.artists(i).popularity** to retrieve the data needed. Refer to the code on slide 16 for assistance.*

Collecting and Managing Data

Discussion

Compare the graphs you just made. What do the graphs reveal about the artists? Is their popularity directly related to the number followers they have? How would these results affect decision making when designing an AI model to find out go is the next GOAT?



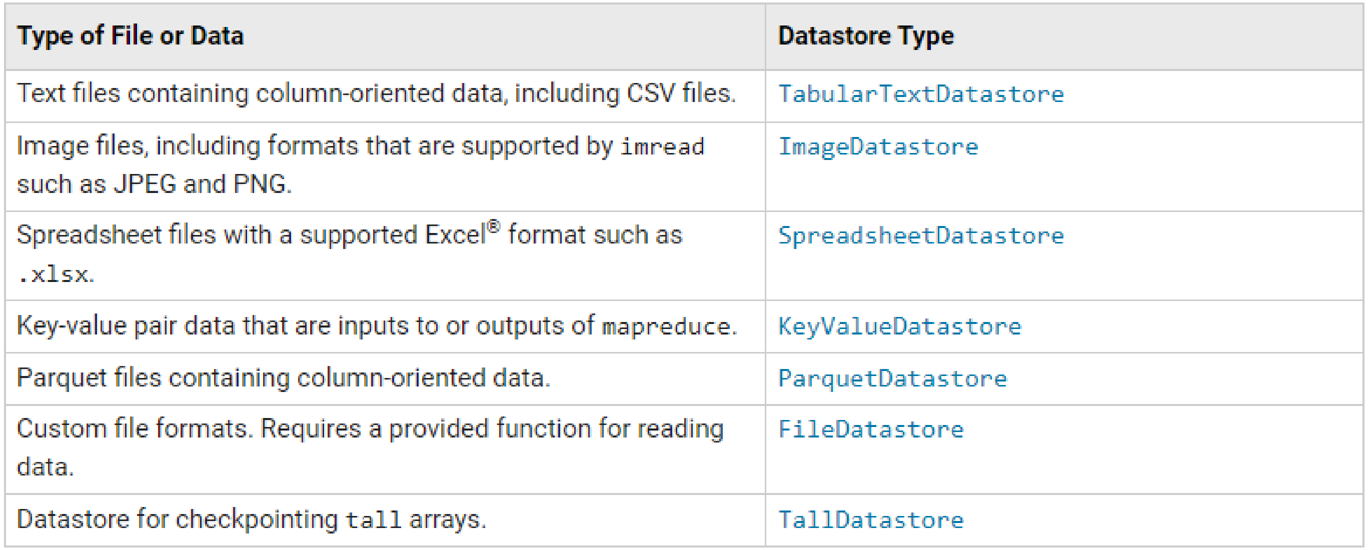


Collecting and Managing Data

Datastores

Now that we can access large amounts of data from an API, we need to store it properly. In the following example, we will reuse the Formula 1 files from section 2 to demonstrate how to create a **datastore** in MATLAB.

A **datastore** provides a convenient way to access data stored in multiple files. The benefit of a datastore is that it allows you to manage a large collection of files that would otherwise be too large to fit in the system's memory. The table below shows the functions required to make a specific datastore type.

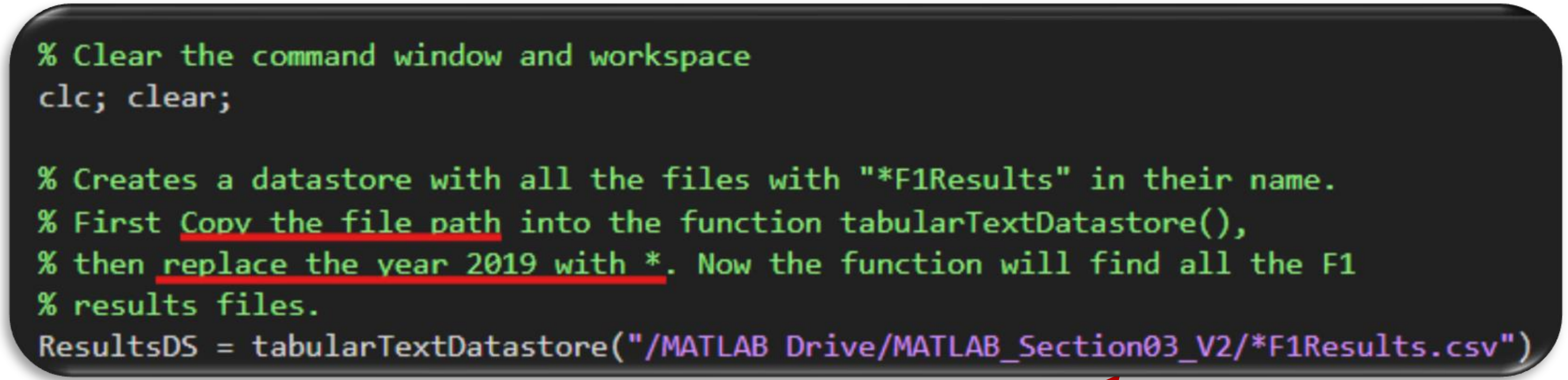


Type of File or Data	Datastore Type
Text files containing column-oriented data, including CSV files.	`TabularTextDatastore`
Image files, including formats that are supported by `imread` such as JPEG and PNG.	`ImageDatastore`
Spreadsheet files with a supported Excel® format such as `.xlsx`.	`SpreadsheetDatastore`
Key-value pair data that are inputs to or outputs of `mapreduce`.	`KeyValueDatastore`
Parquet files containing column-oriented data.	`ParquetDatastore`
Custom file formats. Requires a provided function for reading data.	`FileDatastore`
Datastore for checkpointing `tall` arrays.	`TallDatastore`

Collecting and Managing Data

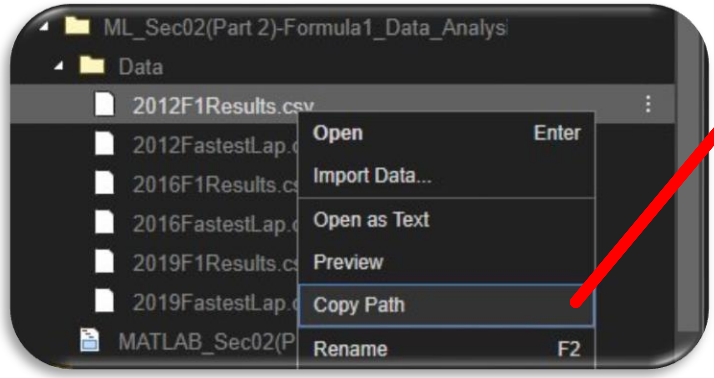
Datastores

This code searches for files with the words "F1Results.csv" in their name using the function **tabularTextDatastore()**. To facilitate the creation of the datastore we can put all the files in the same folder as this live script. However, this is not required since we can specify the location of the files within the function.



```
% Clear the command window and workspace
clc; clear;

% Creates a datastore with all the files with "*F1Results" in their name.
% First Copy the file path into the function tabularTextDatastore(),
% then replace the year 2019 with *. Now the function will find all the F1
% results files.
ResultsDS = tabularTextDatastore("/MATLAB Drive/MATLAB_Section03_V2/*F1Results.csv")
```

Recommendation: Use "copy path" by right clicking on the file to get the .csv path for the tabularTextDatastore() function

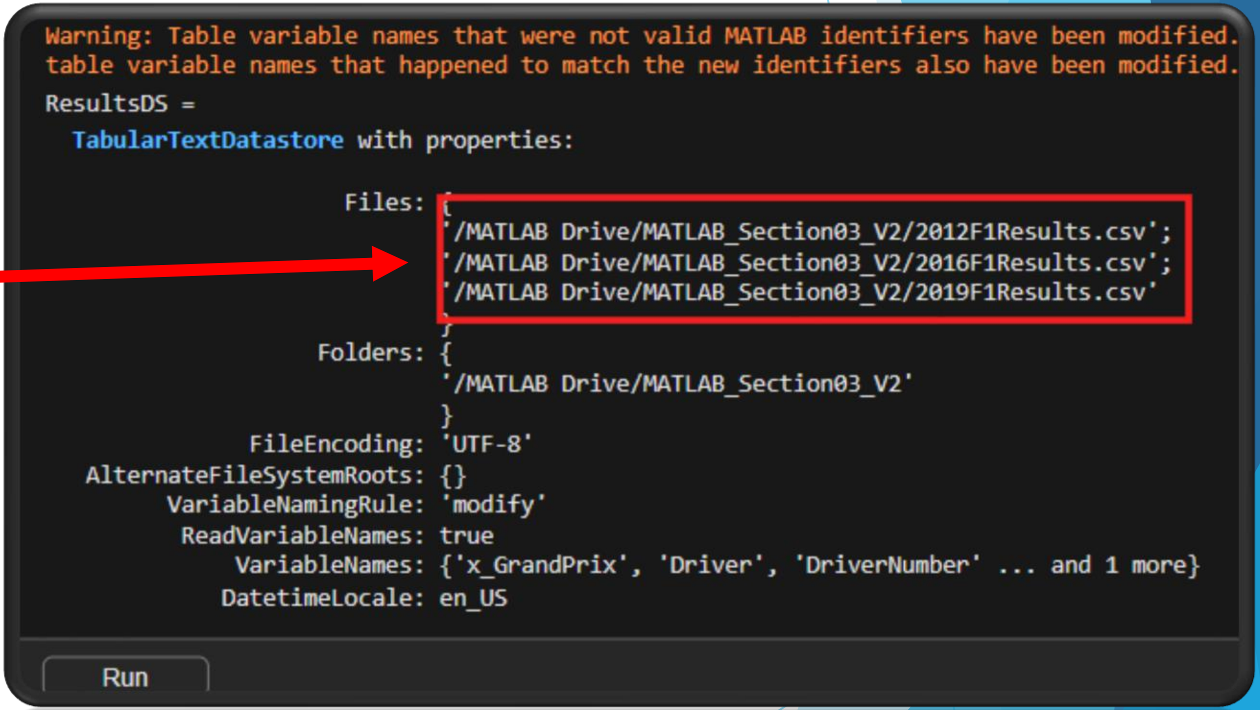


Collecting and Managing Data

Datastores

By examining the output of the code on the previous slide, we can see that 3 files containing F1 results have been stored in the datastore.

Note: Disregard the Warning message.



```
Warning: Table variable names that were not valid MATLAB identifiers have been modified.
table variable names that happened to match the new identifiers also have been modified.
ResultsDS =
  TabularTextDatastore with properties:

                      Files: {
                             '/MATLAB Drive/MATLAB_Section03_V2/2012F1Results.csv';
                             '/MATLAB Drive/MATLAB_Section03_V2/2016F1Results.csv';
                             '/MATLAB Drive/MATLAB_Section03_V2/2019F1Results.csv'
                             }
                    Folders: {
                             '/MATLAB Drive/MATLAB_Section03_V2'
                             }
               FileEncoding: 'UTF-8'
   AlternateFileSystemRoots: {}
         VariableNamingRule: 'modify'
          ReadVariableNames: true
              VariableNames: {'x_GrandPrix', 'Driver', 'DriverNumber' ... and 1 more}
             DatetimeLocale: en_US
```

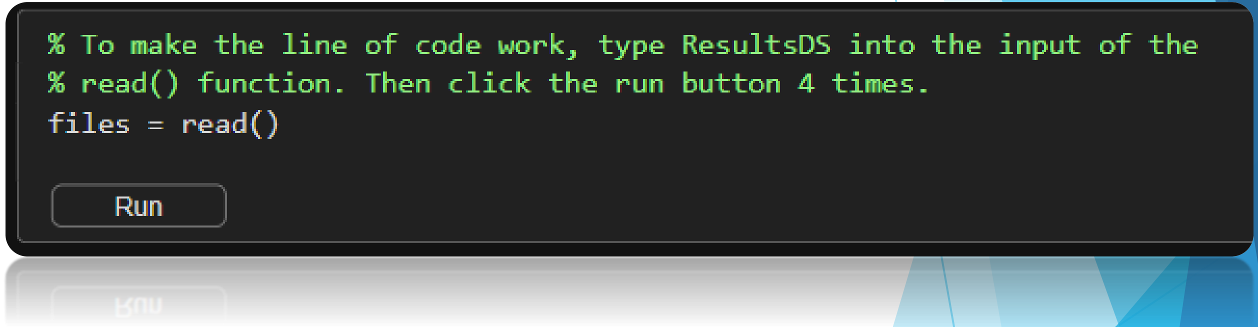
Collecting and Managing Data

Datastores

Now you try!

To access the files store in the newly created datastore, you can use the **read()** function. This function will call each file in the datastore one at a time. The first time you use **read(),** data from the first file is imported. Using it a second imports data from the second file, and so on.

Click the run button 3 times to see how the data table changes. What happens when you click the run button a 4th time?

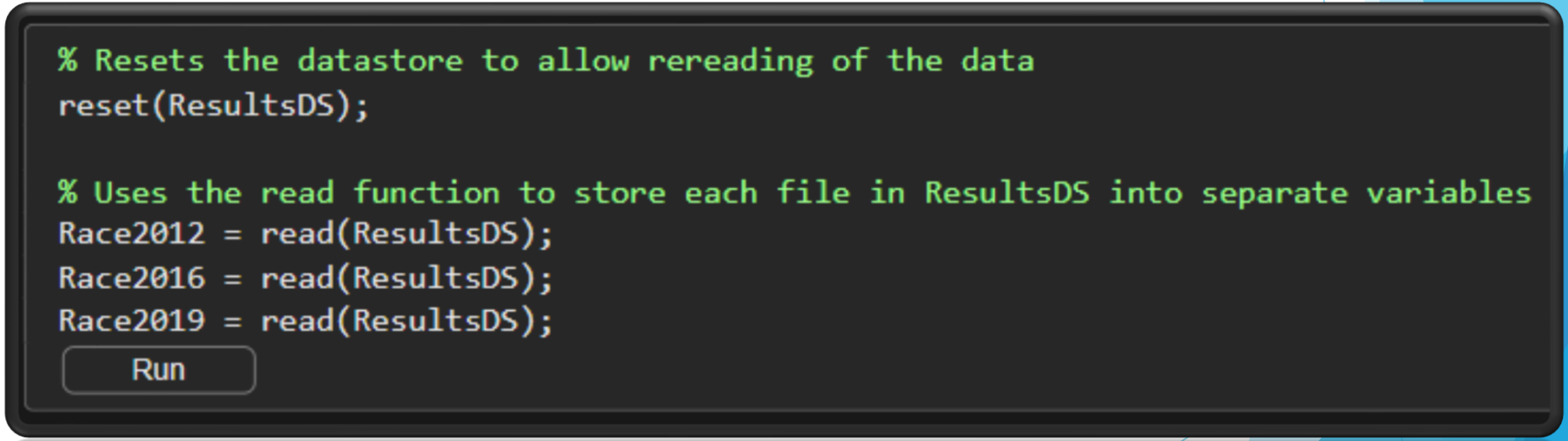


```
% To make the line of code work, type ResultsDS into the input of the
% read() function. Then click the run button 4 times.
files = read()
```

Collecting and Managing Data

Datastores

Before continuing, let's reset the datastore using **reset()** and use **read()** 3 times to store each file in the datastore into its own variable.

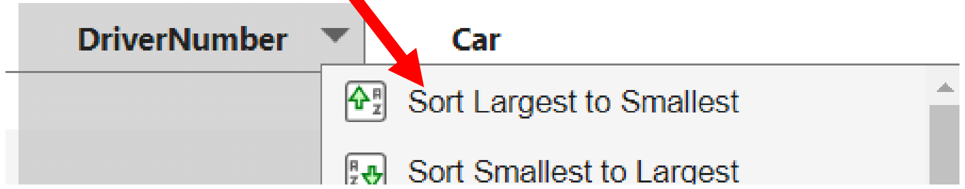


```
% Resets the datastore to allow rereading of the data
reset(ResultsDS);

% Uses the read function to store each file in ResultsDS into separate variables
Race2012 = read(ResultsDS);
Race2016 = read(ResultsDS);
Race2019 = read(ResultsDS);
```

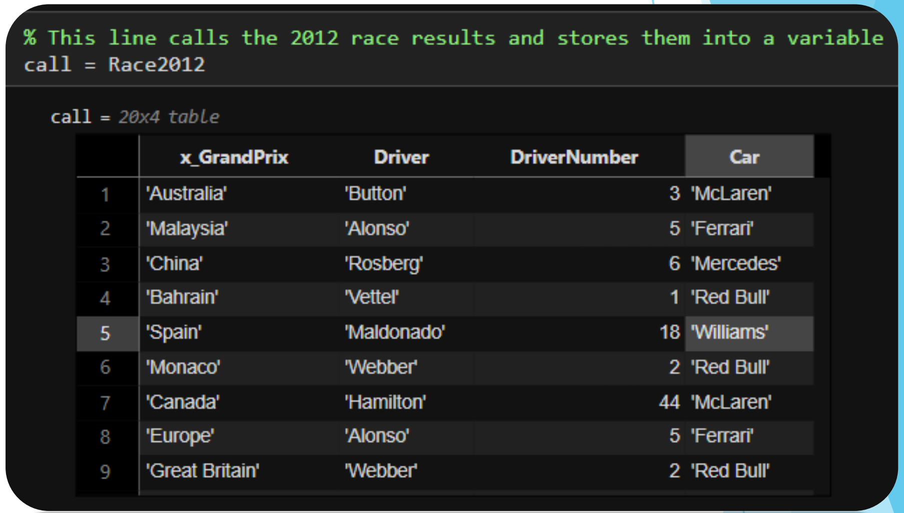
Collecting and Managing Data

Let's use some of the F1 data to explore two functionalities of data tables in MATLAB: sorting data and adding data to a table. Being able to quickly sort data is very helpful when providing data to an AI model. For this example, we will first call the 2012 race results and then sort the driver number column from least to greatest.

Hover over the headings of each column until a dropdown arrow appears. This arrow allows you to sort the data from smallest to largest or vice versa.

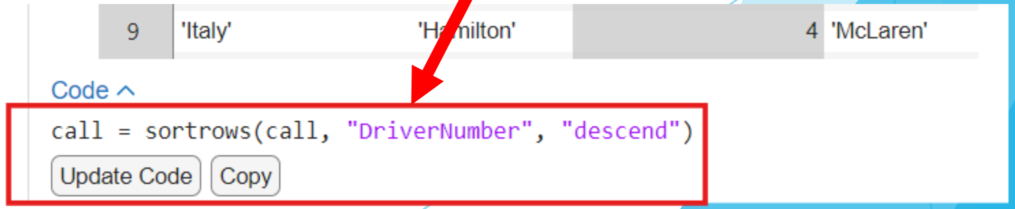


Sorting Data



```
% This line calls the 2012 race results and stores them into a variable
call = Race2012

call = 20x4 table

       x_GrandPrix       Driver        DriverNumber   Car
  1    'Australia'       'Button'                3    'McLaren'
  2    'Malaysia'        'Alonso'                5    'Ferrari'
  3    'China'           'Rosberg'               6    'Mercedes'
  4    'Bahrain'         'Vettel'                1    'Red Bull'
  5    'Spain'           'Maldonado'            18    'Williams'
  6    'Monaco'          'Webber'                2    'Red Bull'
  7    'Canada'          'Hamilton'             44    'McLaren'
  8    'Europe'          'Alonso'                5    'Ferrari'
  9    'Great Britain'   'Webber'                2    'Red Bull'
```

Notice that after selecting one of these options, MATLAB will provide the code necessary to implement these changes into the live script.



```
  9    'Italy'           'Hamilton'              4    'McLaren'
```

Code

```
call = sortrows(call, "DriverNumber", "descend")
```

Collecting and Managing Data

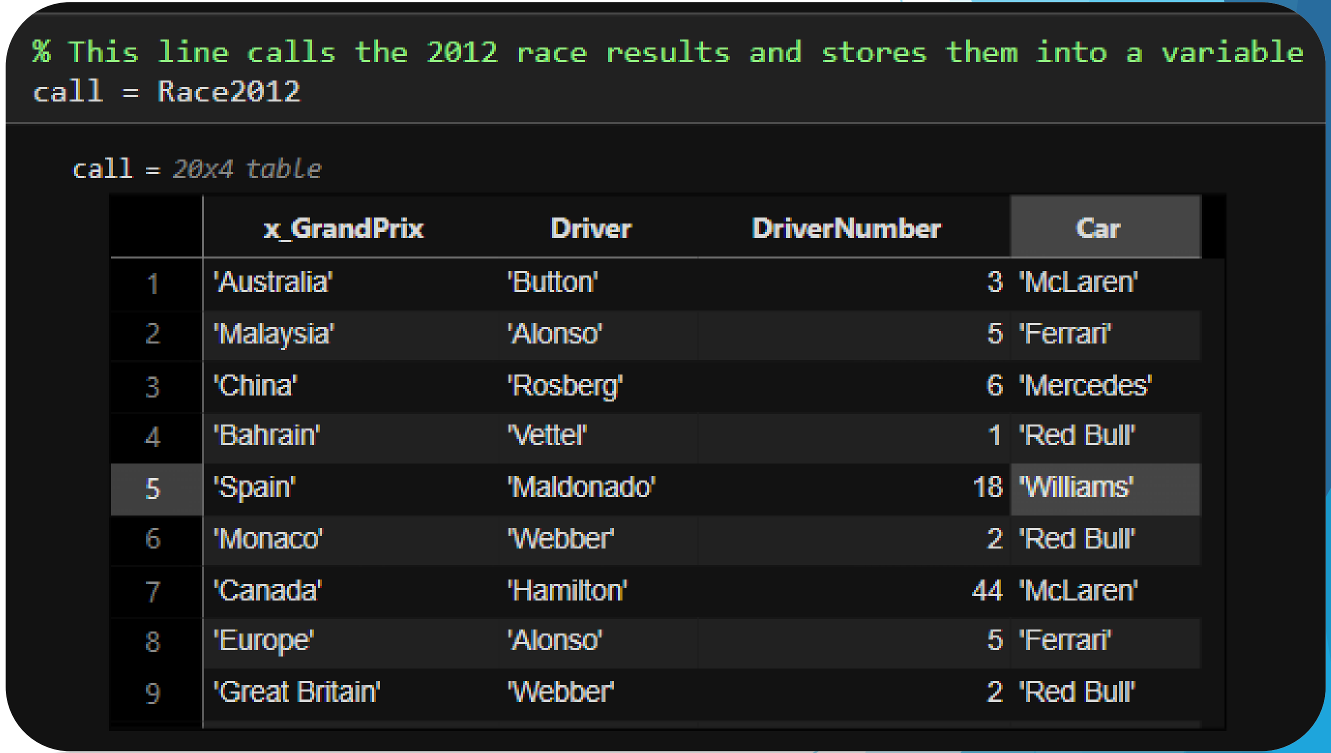
Sorting Data

Now you try!

Go to the 2012 data table and follow the steps in the pervious slide to sort the DriverNumber column.

Then click **"Update Code"** or **"Copy"** to incorporate the suggested code to the live script and rerun this section.

When you sorted the DriverNumber column, did the other columns also change?



```
% This line calls the 2012 race results and stores them into a variable
call = Race2012
```

```
call = 20x4 table
```

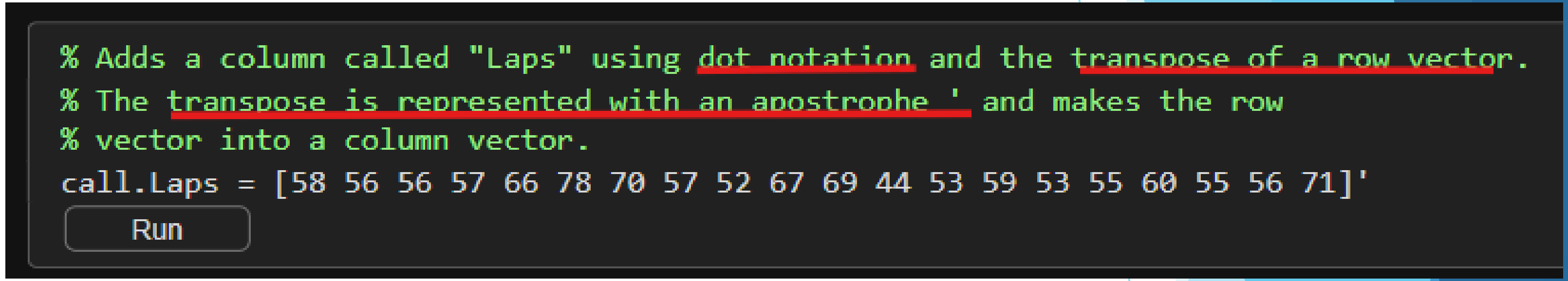
	x_GrandPrix	Driver	DriverNumber	Car
1	'Australia'	'Button'	3	'McLaren'
2	'Malaysia'	'Alonso'	5	'Ferrari'
3	'China'	'Rosberg'	6	'Mercedes'
4	'Bahrain'	'Vettel'	1	'Red Bull'
5	'Spain'	'Maldonado'	18	'Williams'
6	'Monaco'	'Webber'	2	'Red Bull'
7	'Canada'	'Hamilton'	44	'McLaren'
8	'Europe'	'Alonso'	5	'Ferrari'
9	'Great Britain'	'Webber'	2	'Red Bull'

Collecting and Managing Data

Sorting Data

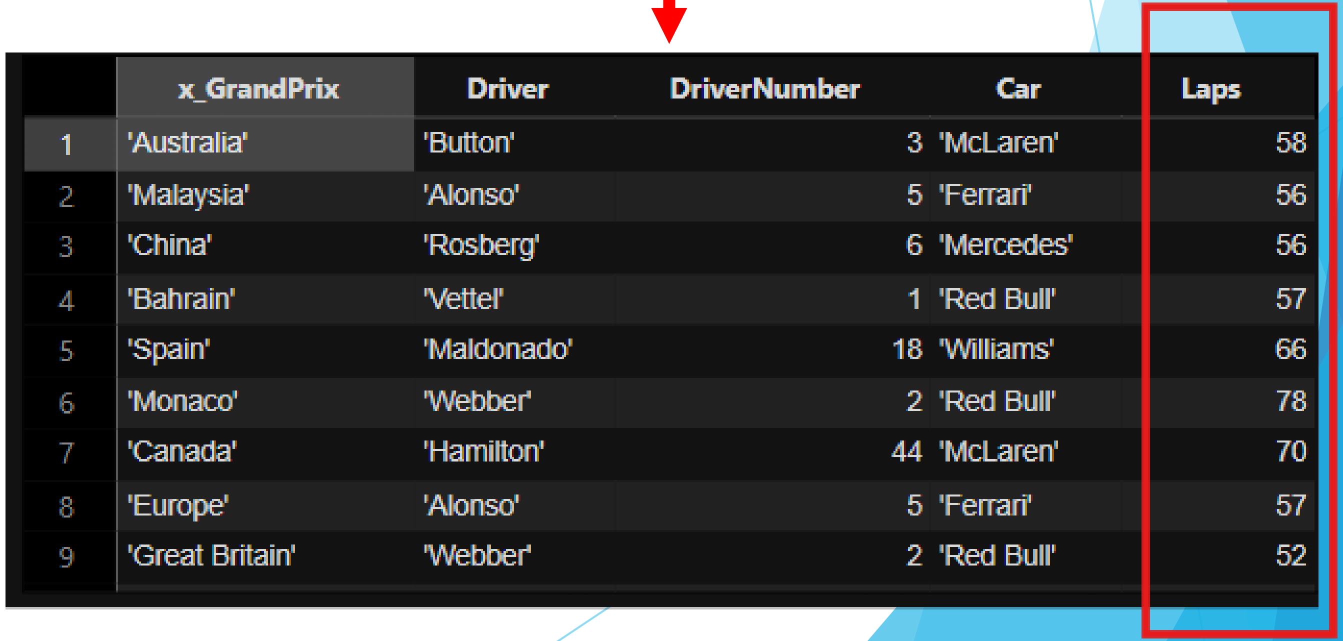
Another important feature is the ability to add columns to an existing data table. Adding columns to a table can be a useful way of storing the results of calculations performed on a dataset.

The code on the right adds an extra column of data to the 2012 race results using **dot notation**. The additional column will contain the number of laps in a race



```
% Adds a column called "Laps" using dot notation and the transpose of a row vector.
% The transpose is represented with an apostrophe ' and makes the row
% vector into a column vector.
call.Laps = [58 56 56 57 66 78 70 57 52 67 69 44 53 59 53 55 60 55 56 71]'
```

Output

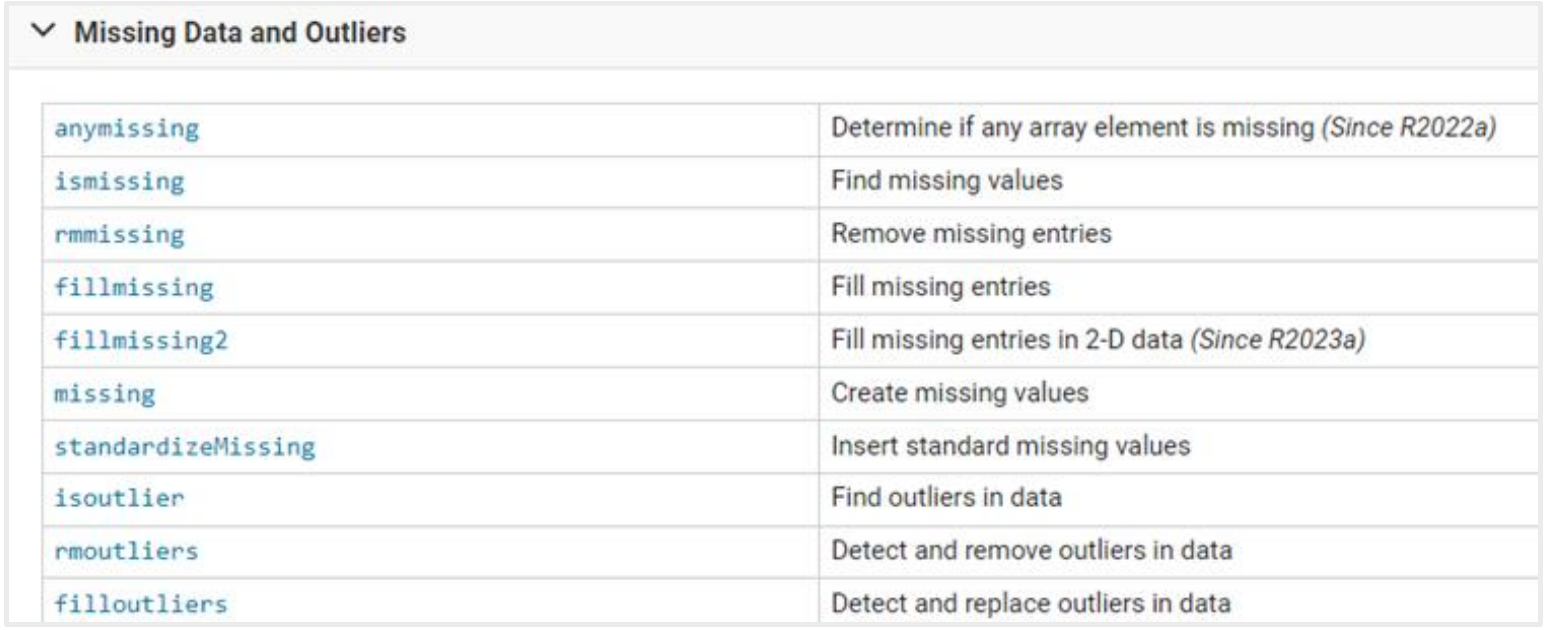


	x_GrandPrix	Driver	DriverNumber	Car	Laps
1	'Australia'	'Button'	3	'McLaren'	58
2	'Malaysia'	'Alonso'	5	'Ferrari'	56
3	'China'	'Rosberg'	6	'Mercedes'	56
4	'Bahrain'	'Vettel'	1	'Red Bull'	57
5	'Spain'	'Maldonado'	18	'Williams'	66
6	'Monaco'	'Webber'	2	'Red Bull'	78
7	'Canada'	'Hamilton'	44	'McLaren'	70
8	'Europe'	'Alonso'	5	'Ferrari'	57
9	'Great Britain'	'Webber'	2	'Red Bull'	52

Collecting and Managing Data

Missing Data

Finally, let's review what can be done when the data you have has missing values. When files with missing values are imported into MATLAB, the missing data is often replaced with NaN (Not a Number). Any calculations involving NaN will result in a value of NaN and can create misleading computations.



Missing Data and Outliers

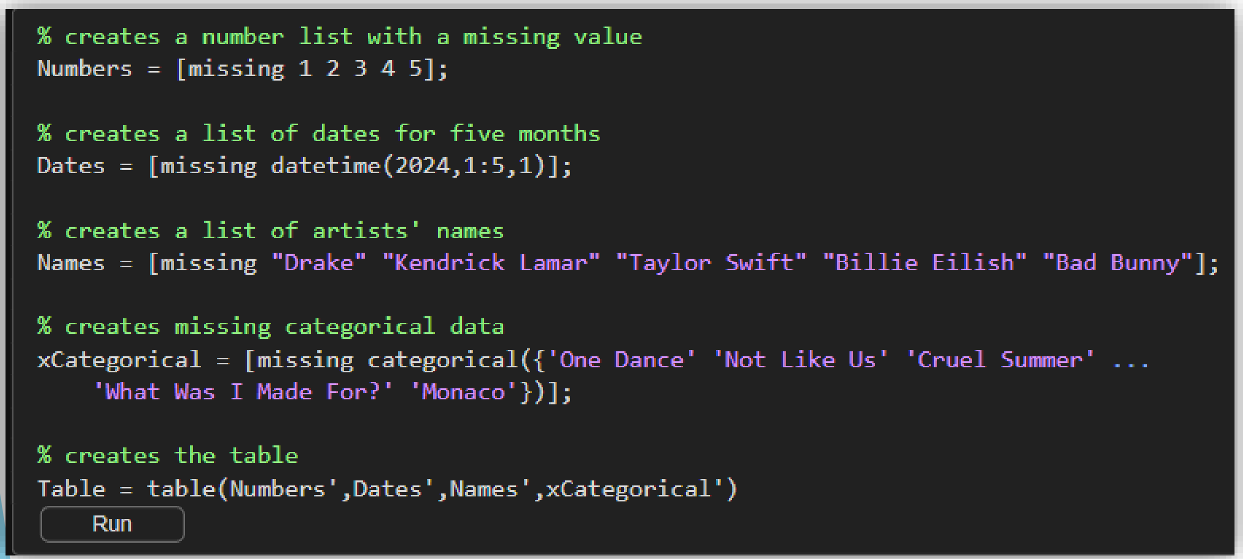
`anymissing`	Determine if any array element is missing *(Since R2022a)*
`ismissing`	Find missing values
`rmmissing`	Remove missing entries
`fillmissing`	Fill missing entries
`fillmissing2`	Fill missing entries in 2-D data *(Since R2023a)*
`missing`	Create missing values
`standardizeMissing`	Insert standard missing values
`isoutlier`	Find outliers in data
`rmoutliers`	Detect and remove outliers in data
`filloutliers`	Detect and replace outliers in data

The image above shows some functions that are commonly used to deal with missing data values.

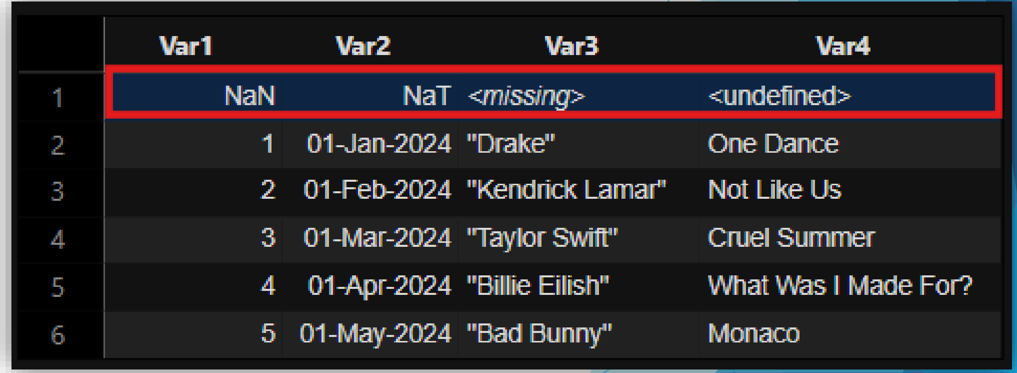
Collecting and Managing Data

Missing Data

To show how two of these functions work, first let's create a data table of imaginary release dates from five artists by running the code below. The data table will purposefully contain missing information.



```
% creates a number list with a missing value
Numbers = [missing 1 2 3 4 5];

% creates a list of dates for five months
Dates = [missing datetime(2024,1:5,1)];

% creates a list of artists' names
Names = [missing "Drake" "Kendrick Lamar" "Taylor Swift" "Billie Eilish" "Bad Bunny"];

% creates missing categorical data
xCategorical = [missing categorical({'One Dance' 'Not Like Us' 'Cruel Summer' ...
    'What Was I Made For?' 'Monaco'})];

% creates the table
Table = table(Numbers',Dates',Names',xCategorical')
```

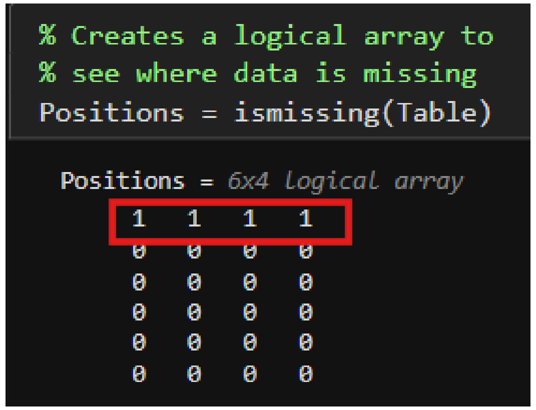


	Var1	Var2	Var3	Var4
1	NaN	NaT	*<missing>*	<undefined>
2	1	01-Jan-2024	"Drake"	One Dance
3	2	01-Feb-2024	"Kendrick Lamar"	Not Like Us
4	3	01-Mar-2024	"Taylor Swift"	Cruel Summer
5	4	01-Apr-2024	"Billie Eilish"	What Was I Made For?
6	5	01-May-2024	"Bad Bunny"	Monaco

Collecting and Managing Data

Missing Data

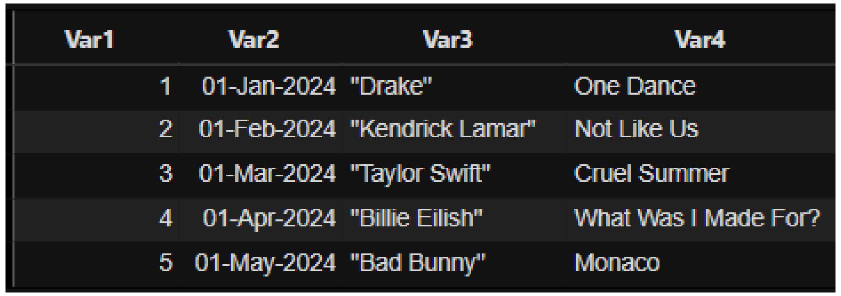
We can use **ismissing()** to find where data is missing in the table. The function will output a logical array with ones in the positions where data is missing.



```
% Creates a logical array to
% see where data is missing
Positions = ismissing(Table)

  Positions = 6x4 logical array
     1   1   1   1
     0   0   0   0
     0   0   0   0
     0   0   0   0
     0   0   0   0
     0   0   0   0
```

Note: This is very helpful because you do not have to go row by row to find where data is missing. Knowing the location of missing data is the first step in cleaning your data.

With the function **rmmissing()** we can remove all missing data types. The benefit of this function is that it works with any kind of data type.



Var1	Var2	Var3	Var4
1	01-Jan-2024	"Drake"	One Dance
2	01-Feb-2024	"Kendrick Lamar"	Not Like Us
3	01-Mar-2024	"Taylor Swift"	Cruel Summer
4	01-Apr-2024	"Billie Eilish"	What Was I Made For?
5	01-May-2024	"Bad Bunny"	Monaco

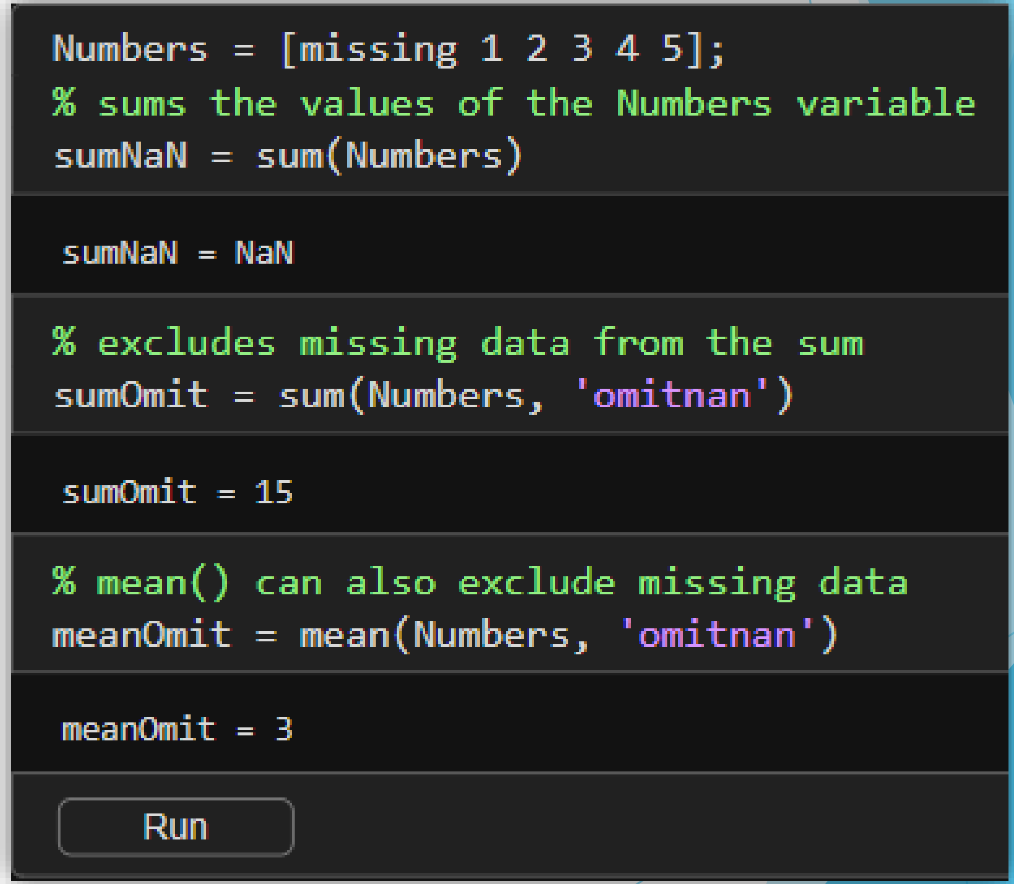
Note: The function works perfectly in this example because all the missing data is in the same row. However, missing data is not always in the same row, so instead we would end up with random blank spaces throughout the table.

Collecting and Managing Data

Finally, it is important to note that many MATLAB functions enable you to ignore missing values, without having to explicitly locate, fill, or remove them first.

For example, if you compute the sum of a vector containing NaN values, the result is NaN. However, you can directly ignore NaNs in the sum by using the **'omitnan'** option with the sum function.

Missing Data



```
Numbers = [missing 1 2 3 4 5];
% sums the values of the Numbers variable
sumNaN = sum(Numbers)
```

```
sumNaN = NaN
```

```
% excludes missing data from the sum
sumOmit = sum(Numbers, 'omitnan')
```

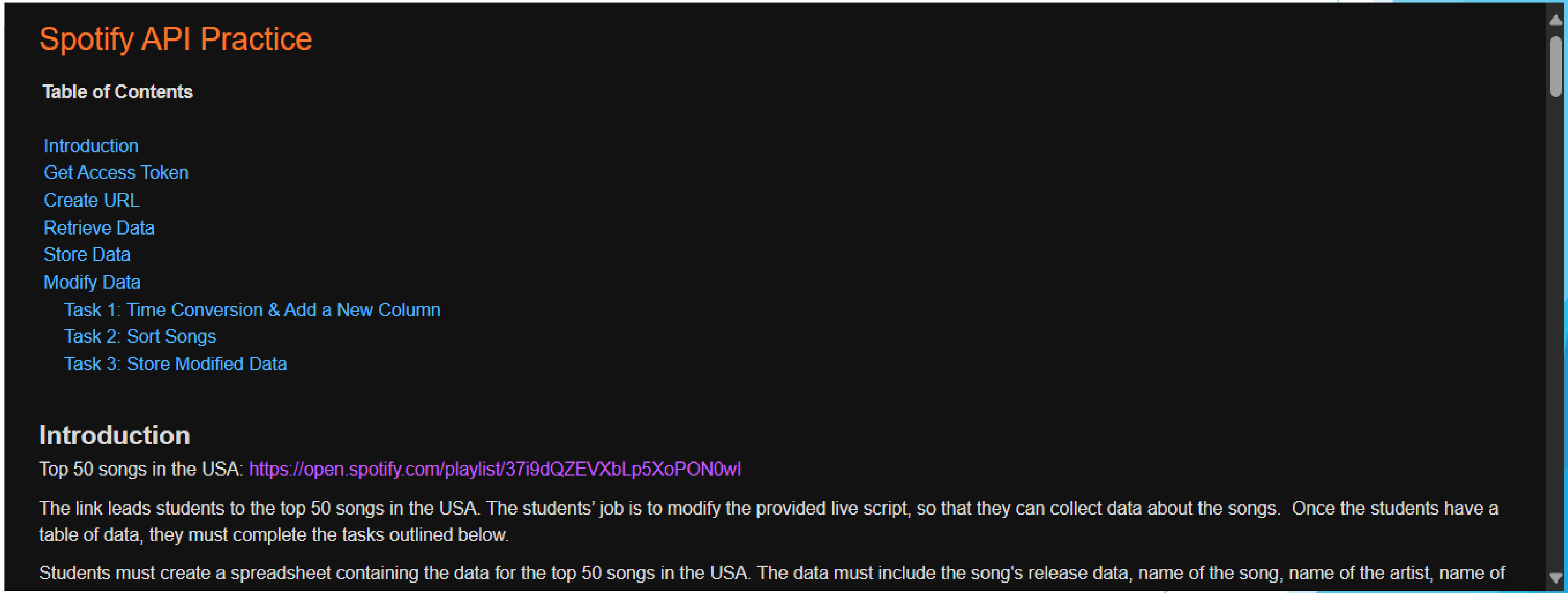
```
sumOmit = 15
```

```
% mean() can also exclude missing data
meanOmit = mean(Numbers, 'omitnan')
```

```
meanOmit = 3
```

Practicing with the Spotify API

Please copy over the files for Section 03 from the MATLAB Drive, the Part 2 live script should be in the same folder.



Spotify API Practice

Table of Contents

Introduction

Top 50 songs in the USA: https://open.spotify.com/playlist/37i9dQZEVXbLp5XoPON0wl

The link leads students to the top 50 songs in the USA. The students' job is to modify the provided live script, so that they can collect data about the songs. Once the students have a table of data, they must complete the tasks outlined below.

Students must create a spreadsheet containing the data for the top 50 songs in the USA. The data must include the song's release data, name of the song, name of the artist, name of

[The MATLAB files for this section can be found at this link.]

Spotify API Practice

Introduction

You've been tasked by a record label to collect Spotify data on the **Top 50 Songs in the USA**. Your job is to create a spreadsheet containing the release data, name of the song, name of the artist, name of the album, and the duration of the Top 50 songs. After making the spreadsheet you need to complete the tasks below.

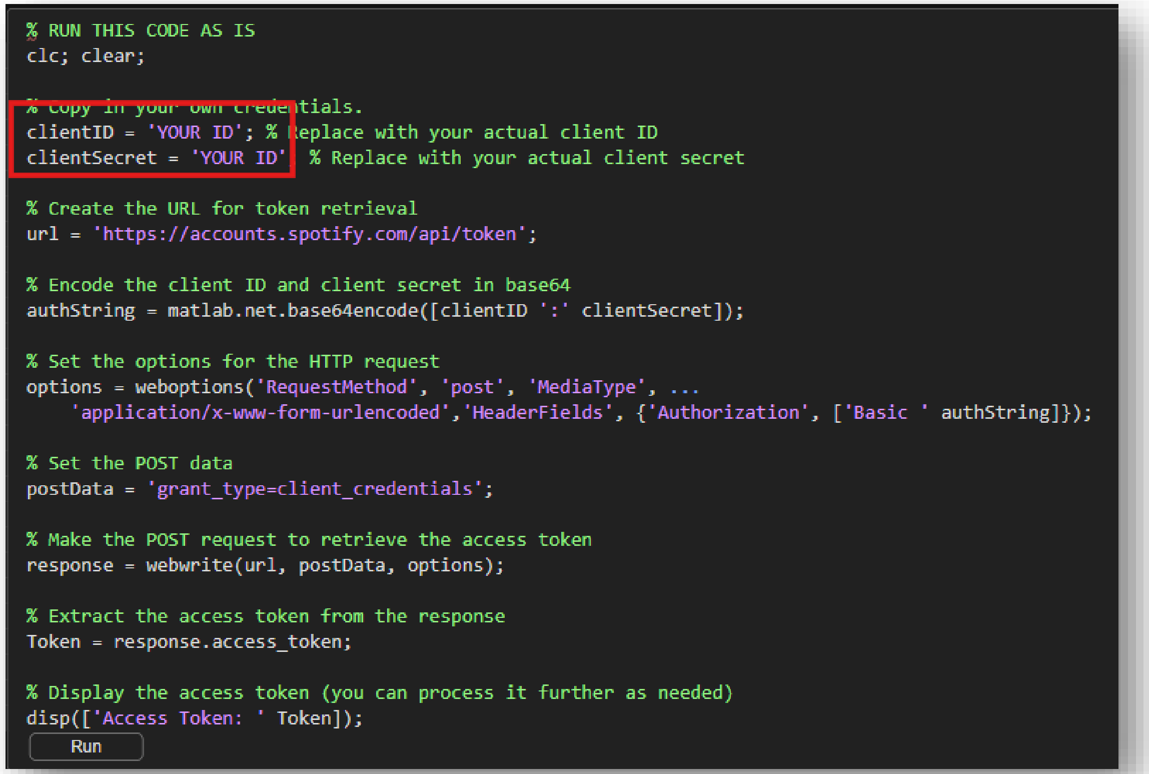
- Task 1: Convert the Duration into minutes and add the results into a new column in the data table.
- Task 2: Without using code sort the songs in order of longest to shortest duration.
- Task 3: Store the modified data table into a new Excel file.

You will need an account to access the API, as you did in the previous live script, feel free to go back to this video if needed.

Spotify API Practice

Get Access Token

Remember that first we need to get an **Access Token**. Copy your **Client ID** and **Client Secret** in the code below to get a token.



```
% RUN THIS CODE AS IS
clc; clear;

% Copy in your own credentials.
clientID = 'YOUR ID'; % Replace with your actual client ID
clientSecret = 'YOUR ID'  % Replace with your actual client secret

% Create the URL for token retrieval
url = 'https://accounts.spotify.com/api/token';

% Encode the client ID and client secret in base64
authString = matlab.net.base64encode([clientID ':' clientSecret]);

% Set the options for the HTTP request
options = weboptions('RequestMethod', 'post', 'MediaType', ...
    'application/x-www-form-urlencoded','HeaderFields', {'Authorization', ['Basic ' authString]});

% Set the POST data
postData = 'grant_type=client_credentials';

% Make the POST request to retrieve the access token
response = webwrite(url, postData, options);

% Extract the access token from the response
Token = response.access_token;

% Display the access token (you can process it further as needed)
disp(['Access Token: ' Token]);
```

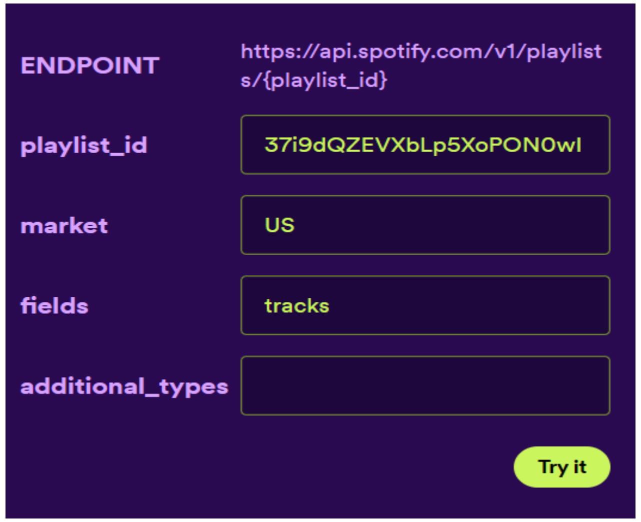
Spotify API Practice

Use the interactive feature to create the appropriate link to access the playlist. In the Market text box type **"US"** and in the Field text box type **“tracks"**.

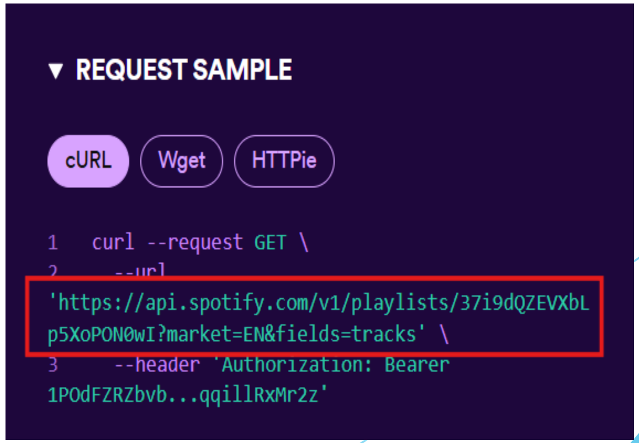
The **Request Sample** will display the URL you need to use. This makes it really easy to access all kinds of data from the Spotify API.

This URL has already been included in your code, but it has some missing information. On the next slide, are instructions to fill in the missing information to format the variable "url" correctly.

Create the URL



ENDPOINT	https://api.spotify.com/v1/playlists/{playlist_id}
playlist_id	37i9dQZEVXbLp5XoPON0wI
market	US
fields	tracks
additional_types	



```
curl --request GET \
  --url
'https://api.spotify.com/v1/playlists/37i9dQZEVXbL
p5XoPON0wI?market=EN&fields=tracks' \
  --header 'Authorization: Bearer
1POdFZRZbvb...qqillRxMr2z'
```

Spotify API Practice

Fill in the missing information in the red boxes to format the variable "url" correctly.

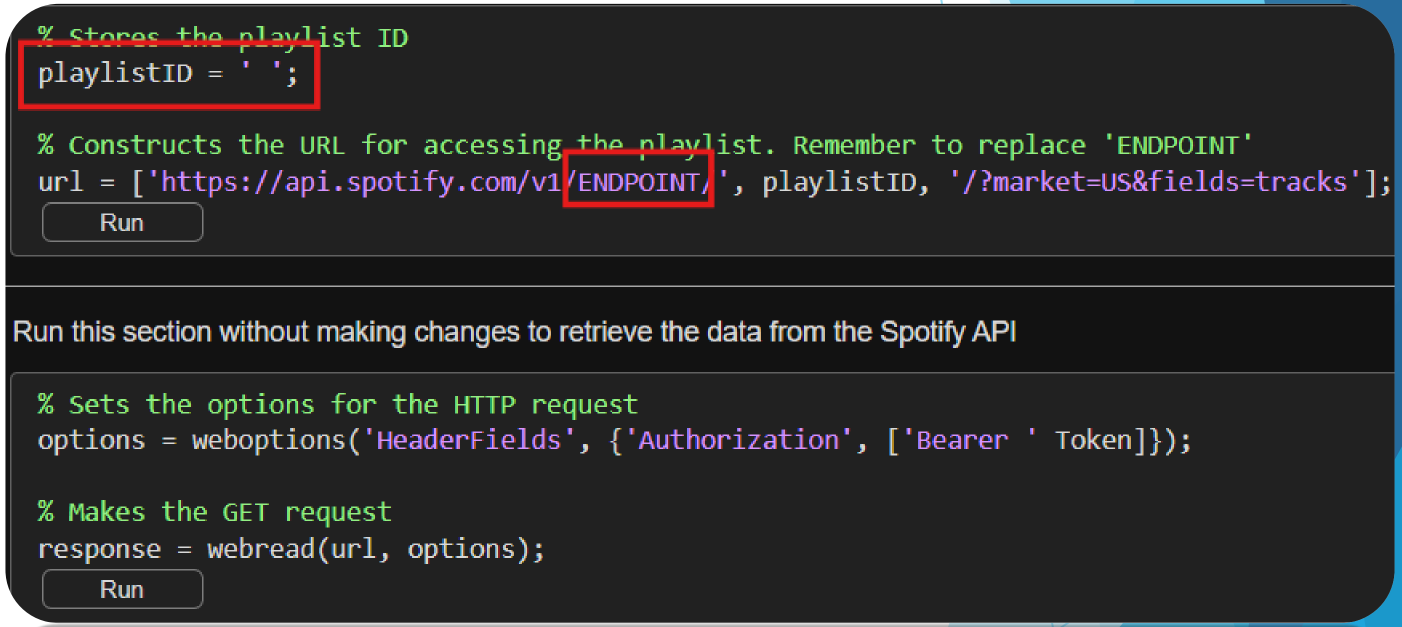
1. Copy in the **playlistID.**

   Hint: The link to the playlist (Slide 33) provides some of the information required by the API. From the link we know that the playlist ID is '37i9dQZEVXbLp5XoPON0wl'.

2. Replace **/ENDPOINT/** with the endpoint required to access playlists.

   Hint: Examine the link in the Request Sample screenshot on the previous slide.

Create the URL



```
% Stores the playlist ID
playlistID = ' ';

% Constructs the URL for accessing the playlist. Remember to replace 'ENDPOINT'
url = ['https://api.spotify.com/v1/ENDPOINT/', playlistID, '/?market=US&fields=tracks'];
```

Run

Run this section without making changes to retrieve the data from the Spotify API

```
% Sets the options for the HTTP request
options = weboptions('HeaderFields', {'Authorization', ['Bearer ' Token]});

% Makes the GET request
response = webread(url, options);
```

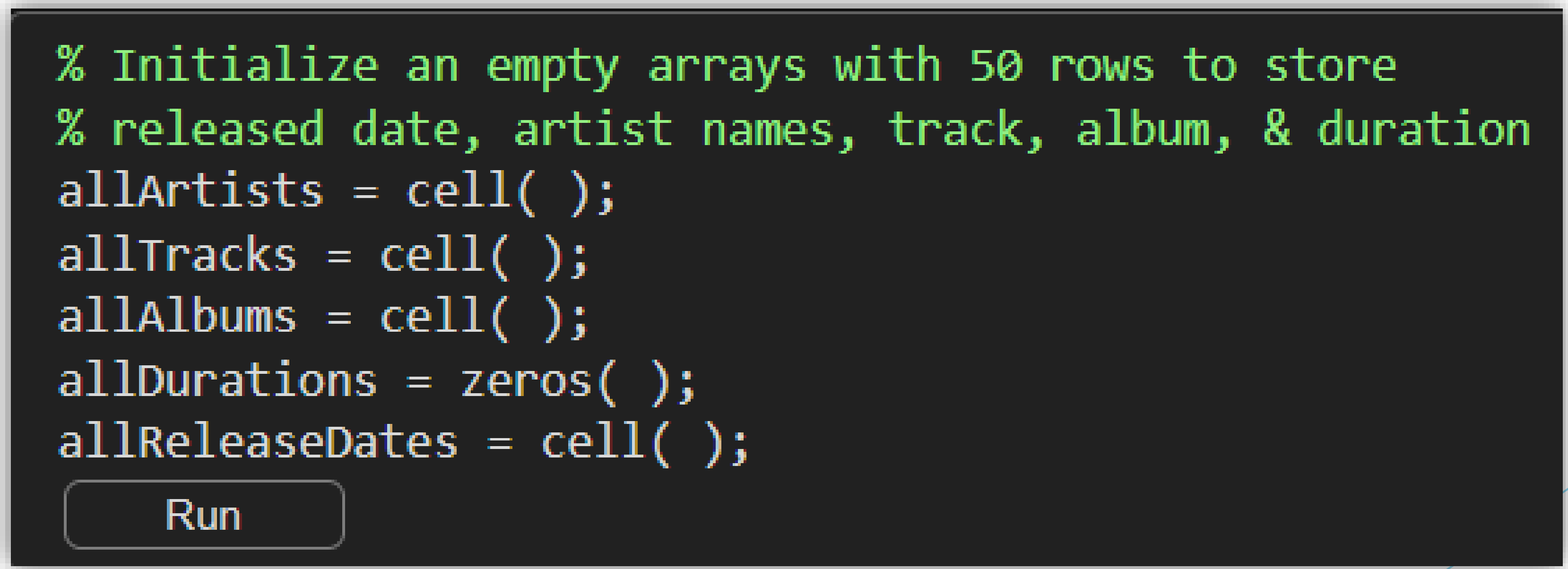
Run

Spotify API Practice

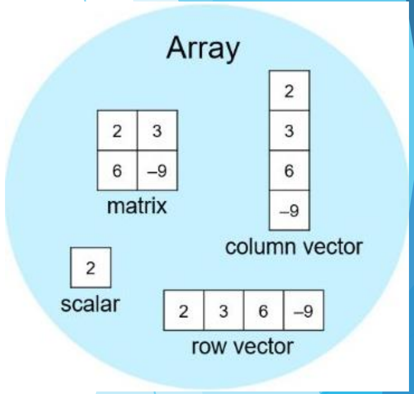
Retrieve Data

Now you need to extract the release date, name of the artist, track, album, and the duration of the tracks from the variable **"response"**.

First create empty arrays to store the data. Fill in the empty **cell()** and **zeros()** functions to create arrays with 1 column and 50 rows.



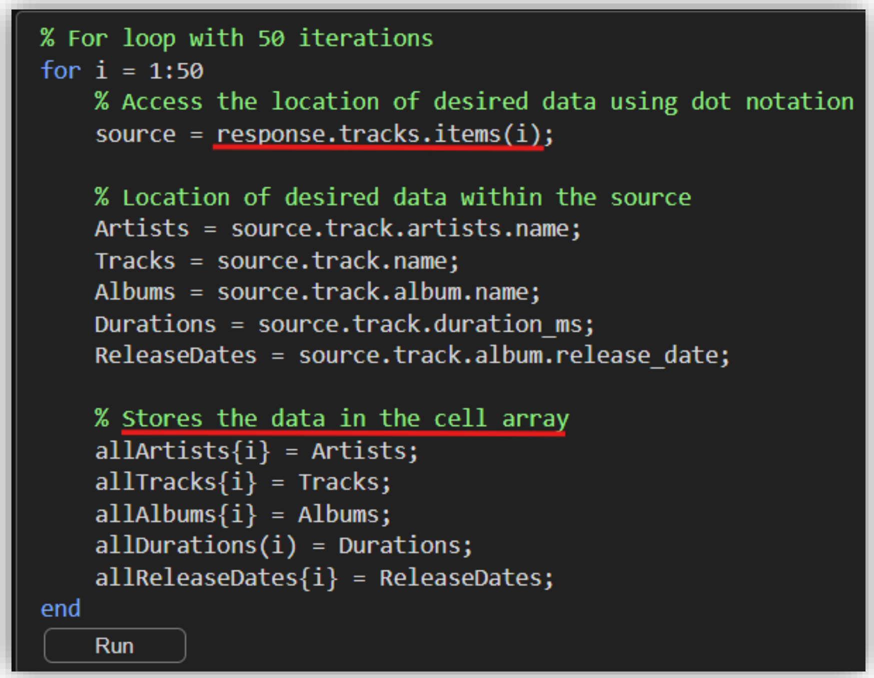
```
% Initialize an empty arrays with 50 rows to store
% released date, artist names, track, album, & duration
allArtists = cell( );
allTracks = cell( );
allAlbums = cell( );
allDurations = zeros( );
allReleaseDates = cell( );
```



Spotify API Practice

Retrieve Data

If you look at the data stored in the "**response"** variable, you will notice that most of the desired data is stored in a 50 individual structures called **"items"**. To access "items" use the dot notation **response.tracks.items(i)**. The for loop below goes through each "items" structure to extract the desired data for each song. Run the code segment below without making any changes.



```
% For loop with 50 iterations
for i = 1:50
    % Access the location of desired data using dot notation
    source = response.tracks.items(i);

    % Location of desired data within the source
    Artists = source.track.artists.name;
    Tracks = source.track.name;
    Albums = source.track.album.name;
    Durations = source.track.duration_ms;
    ReleaseDates = source.track.album.release_date;

    % Stores the data in the cell array
    allArtists{i} = Artists;
    allTracks{i} = Tracks;
    allAlbums{i} = Albums;
    allDurations(i) = Durations;
    allReleaseDates{i} = ReleaseDates;
end
```

For this example, the location of the data has been provided, but it is recommended that you open the "response" variable by clicking on it in the workspace to see how the data is stored within the variable.

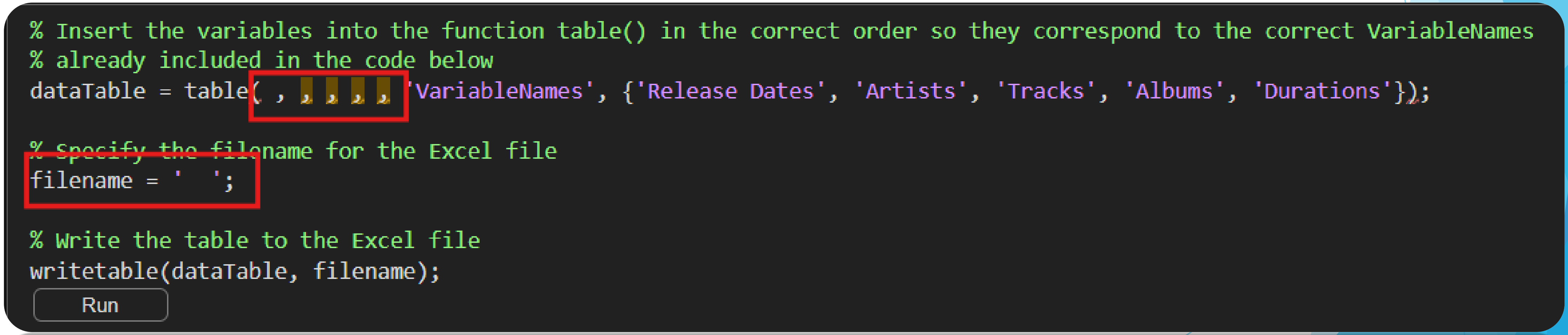
Spotify API Practice

Store Data

To store the retrieved data in an Excel file:

1. Input the name of the 5 variables with data into the **table()** function in the correct order
   Hint: The order is determined is given between {}. The variables you need to input the table function are allArtists, allTracks, allAlbums, allDurations, allReleaseDates.

2. Name the Excel file before running this segment of code for it to work.

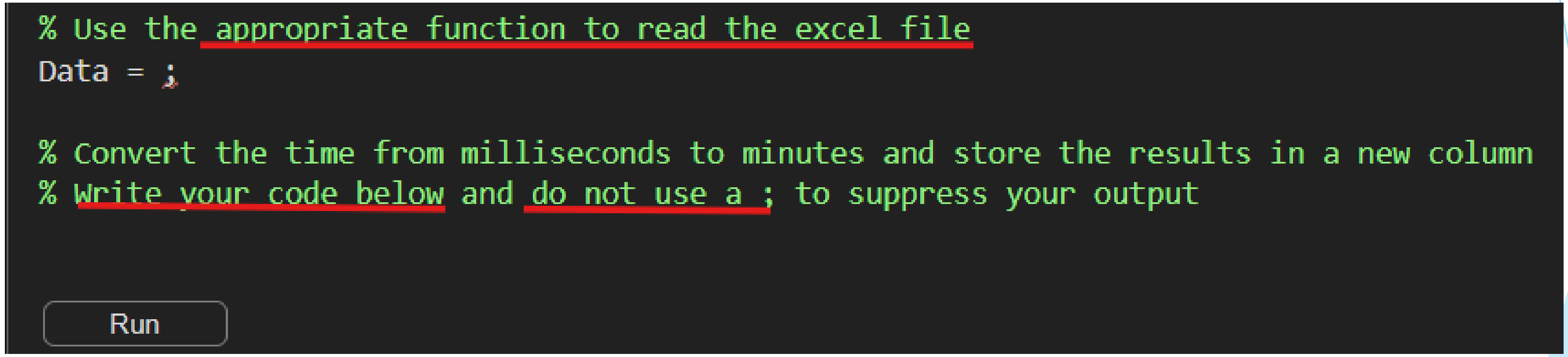


```
% Insert the variables into the function table() in the correct order so they correspond to the correct VariableNames
% already included in the code below
dataTable = table( , , , , , 'VariableNames', {'Release Dates', 'Artists', 'Tracks', 'Albums', 'Durations'});

% Specify the filename for the Excel file
filename = '  ';

% Write the table to the Excel file
writetable(dataTable, filename);
```

Spotify API Practice

Task 1: Time Conversion & Add a New Column

Now you have an Excel file to complete the required tasks. For the first task write code to store the data into variable called **Data**. Then convert the duration of each song into minutes and store the results in a new column called **"ConvertedDurations".**



```
% Use the appropriate function to read the excel file
Data = ;

% Convert the time from milliseconds to minutes and store the results in a new column
% Write your code below and do not use a ; to suppress your output
```

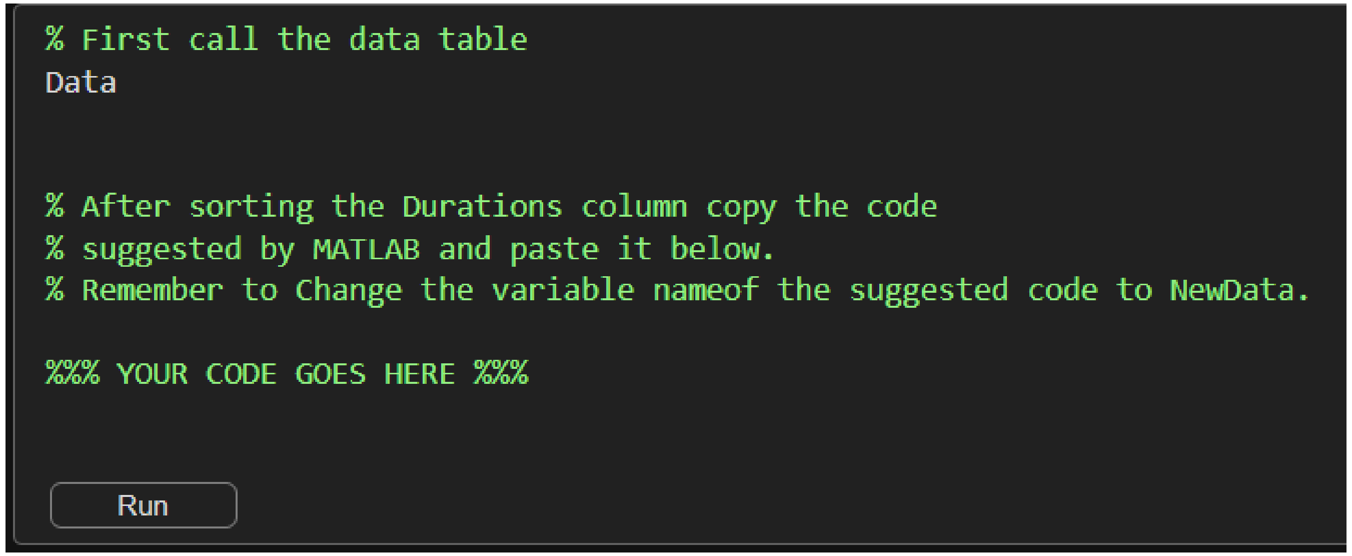
- **Hint 1: Refer to Slide 27 for help.**
- **Hint 2: 60,000 seconds = 1 minute**

Note: Disregard the Warning message that appears when you run the code.

Spotify API Practice

Task 2: Sort the Songs

Call the data table and sort the **duration** from longest to shortest without using code. Then copy the code that MATLAB suggests to modify the data table. Finally, store the new table in the variable **"NewData".**



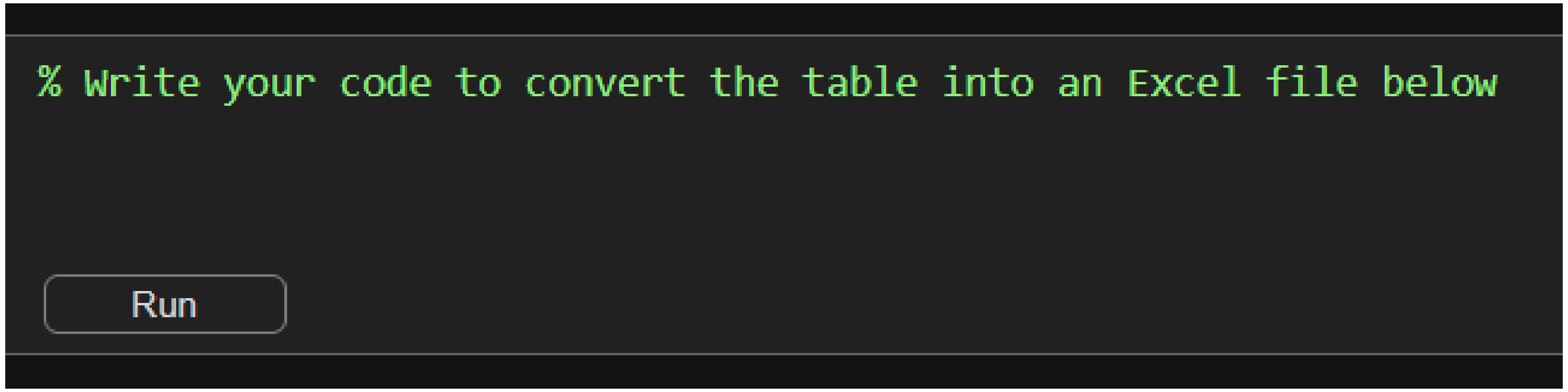
```
% First call the data table
Data


% After sorting the Durations column copy the code
% suggested by MATLAB and paste it below.
% Remember to Change the variable nameof the suggested code to NewData.

%%% YOUR CODE GOES HERE %%%
```

- **Hint: Refer to Slide 25 for help.**

Spotify API Practice

Task 3: Store Modified Data

Write a single line of code to store the table **"NewData"** into a new excel file with the name **"Top_50_USA_v2"**



```
% Write your code to convert the table into an Excel file below
```

- **Hint: See the Store Data section on Slide 39 to identify which function is used to create an Excel file.**

What are the Benefits of APIs?

Congratulations, you have completed your task for the record label!

Now Let's think about the lesson and activity we just did:

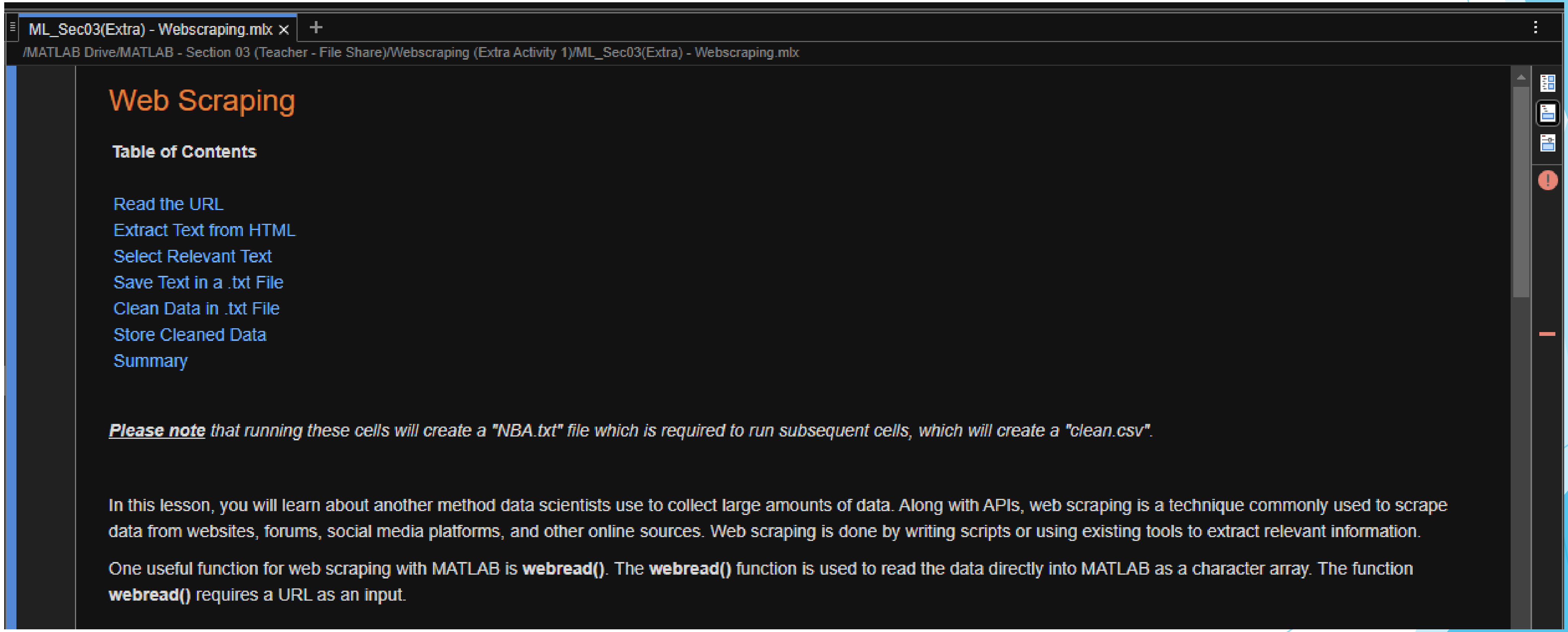
In this lesson, we learned about APIs and practiced retrieving data from the Spotify API. We also learned how to create datastores and sort data in MATLAB®.

Knowing this, discuss the following in groups:

- **Discuss the benefits of using an API to retrieve data versus directly uploading files with data into MATLAB®.**
- **Reflect on any challenges faced when working with the Spotify API in the practice activity.**

EXTRA Practice/Reference Material: Web Scraping

Please copy over the files for Section 03 from the MATLAB Drive, the "Extra 1" live script should be in the same folder.


ML_Sec03(Extra) - Webscraping.mlx

/MATLAB Drive/MATLAB - Section 03 (Teacher - File Share)/Webscraping (Extra Activity 1)/ML_Sec03(Extra) - Webscraping.mlx

Web Scraping

Table of Contents

Please note *that running these cells will create a "NBA.txt" file which is required to run subsequent cells, which will create a "clean.csv".*

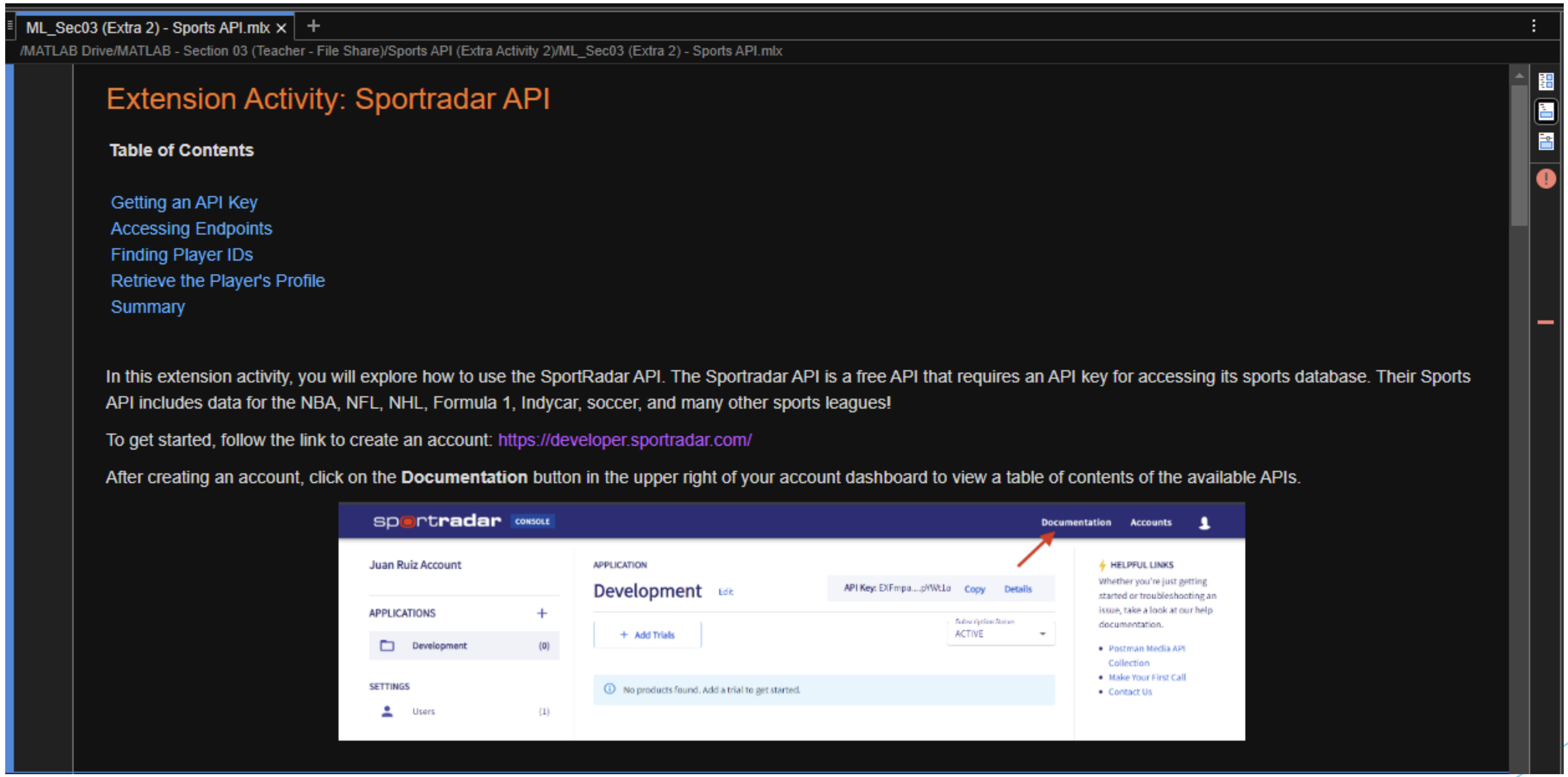
In this lesson, you will learn about another method data scientists use to collect large amounts of data. Along with APIs, web scraping is a technique commonly used to scrape data from websites, forums, social media platforms, and other online sources. Web scraping is done by writing scripts or using existing tools to extract relevant information.

One useful function for web scraping with MATLAB is **webread()**. The **webread()** function is used to read the data directly into MATLAB as a character array. The function **webread()** requires a URL as an input.

The MATLAB files for this section can be found at this link.

EXTRA Practice/Reference Material: Sports API

Please copy over the files for Section 03 from the MATLAB Drive, the “Extra 2” live script should be in the same folder.



ML_Sec03 (Extra 2) - Sports API.mlx ×

/MATLAB Drive/MATLAB - Section 03 (Teacher - File Share)/Sports API (Extra Activity 2)/ML_Sec03 (Extra 2) - Sports API.mlx

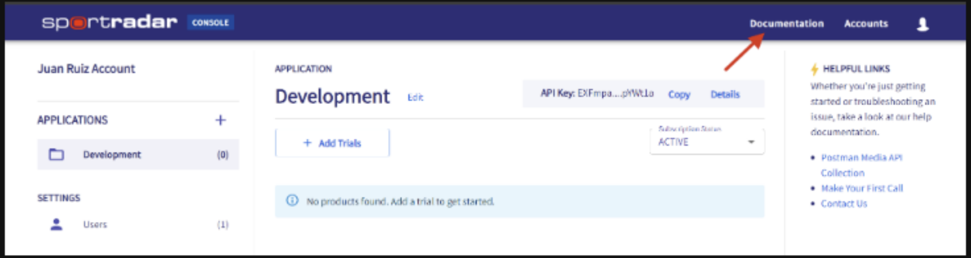
Extension Activity: Sportradar API

Table of Contents

In this extension activity, you will explore how to use the SportRadar API. The Sportradar API is a free API that requires an API key for accessing its sports database. Their Sports API includes data for the NBA, NFL, NHL, Formula 1, Indycar, soccer, and many other sports leagues!

To get started, follow the link to create an account: https://developer.sportradar.com/

After creating an account, click on the **Documentation** button in the upper right of your account dashboard to view a table of contents of the available APIs.


sportradar CONSOLE
Documentation
Accounts
Juan Ruiz Account
APPLICATIONS
Development (0)
SETTINGS
Users (1)
APPLICATION
Development Edit
API Key: EXFmpa....pYWtLo Copy Details
+ Add Trials
Subscription Status
ACTIVE
No products found. Add a trial to get started.
HELPFUL LINKS
Whether you're just getting started or troubleshooting an issue, take a look at our help documentation.
Postman Media API Collection
Make Your First Call
Contact Us

The MATLAB files for this section can be found at this link.