# BQL Basics

# The Bloomberg Query Language (BQL) Use Python to Access Bloomberg Data

# The bql.Service() Object

Inside the bql library is an object called Service. You will use this service object to connect to Bloomberg's servers and send and receive requests for data. After you import the bql library, declare a variable to store the bql.Service() object. You can call this variable whatever you want, but the convention among most BQuant developers is to name it bq, so let's use that name.

# BQL Universes, Fields, and Functions

When making a BQL request for data, you will need to pass two arguments: the type of data, and the universe. You can think of BQL queries as having two statements: GET and FOR. If you need daily trading volume for Apple stock, you are asking BQL to "get" volume data "for" Apple. All BQL queries will follow this basic structure: GET <data type> FOR <universe>.

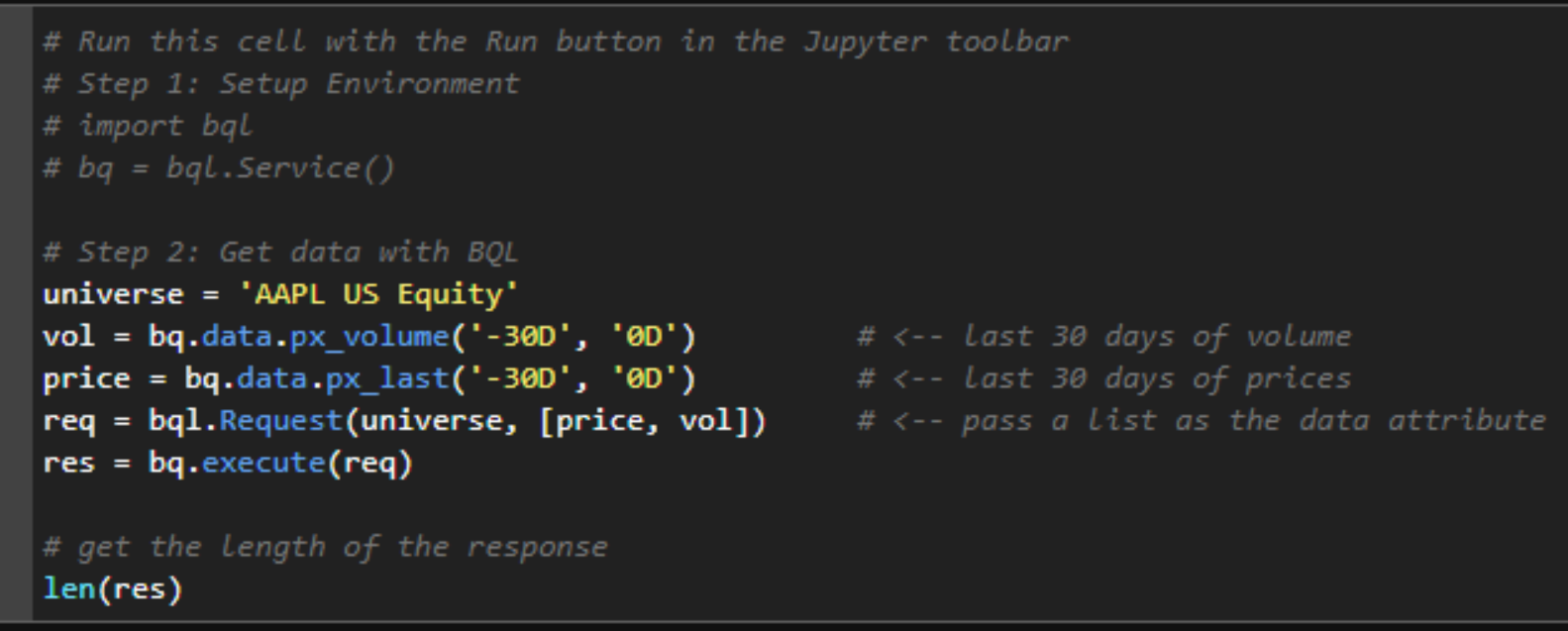
While the universe variable is fairly straightforward, the data type can get quite complex. This data type variable can be a combination of BQL "fields" and "functions" for the universe. For example, if you are only interested in the daily volume, you can simply ask BQL to retrieve one field (volume). But suppose you are interested in the average volume over a period of time. In this case, you would need to perform a function (average) on the field (volume). Oftentimes you will need a combination of fields and functions to get the right data.

The bql.Service() object, normally named bq, gives you access to these fields and functions with the syntax bq.field and bq.func, respectively. These fields and functions can be chained together to perform calculations. The example below prepares two data types for a BQL request: vol for the daily trading volume for Apple stock over the last 30 days, and avg\_vol for the average of those numbers. Running the cell below will not produce any output; we'll discuss how to make a request and parse a response in the next sections.

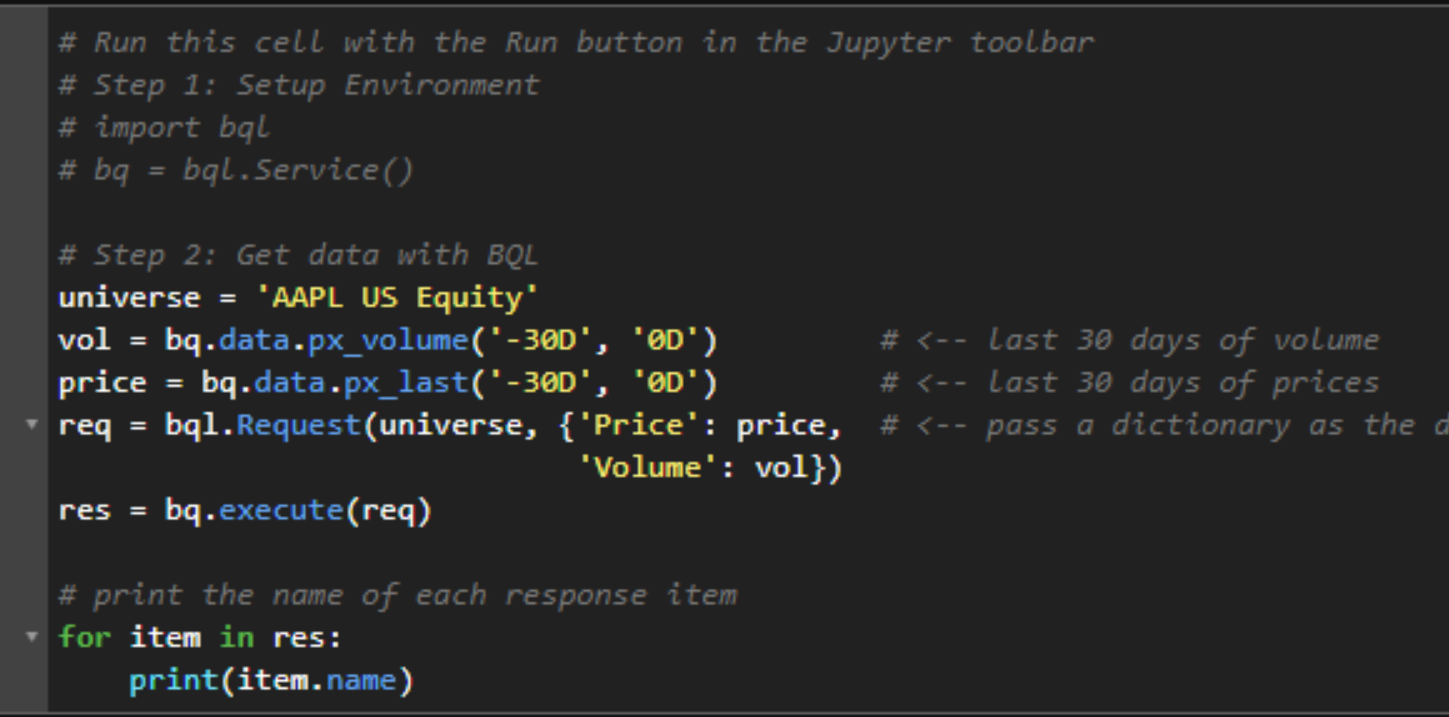
# BQL Requests and Responses

Once a universe and data type have been declared, it's time to write a BQL request, send it to Bloomberg servers, and receive a response. Use the bql.Request object to initiate the request, and the bq.Service object has an execute method that will send the request and receive a response.

Notice that this response object res has a length of 1. This is because we requested only one dataset - vol. In the example below, we pass a list to the bql.Request object with two datasets for the same universe. Now the length of the response object is 2.



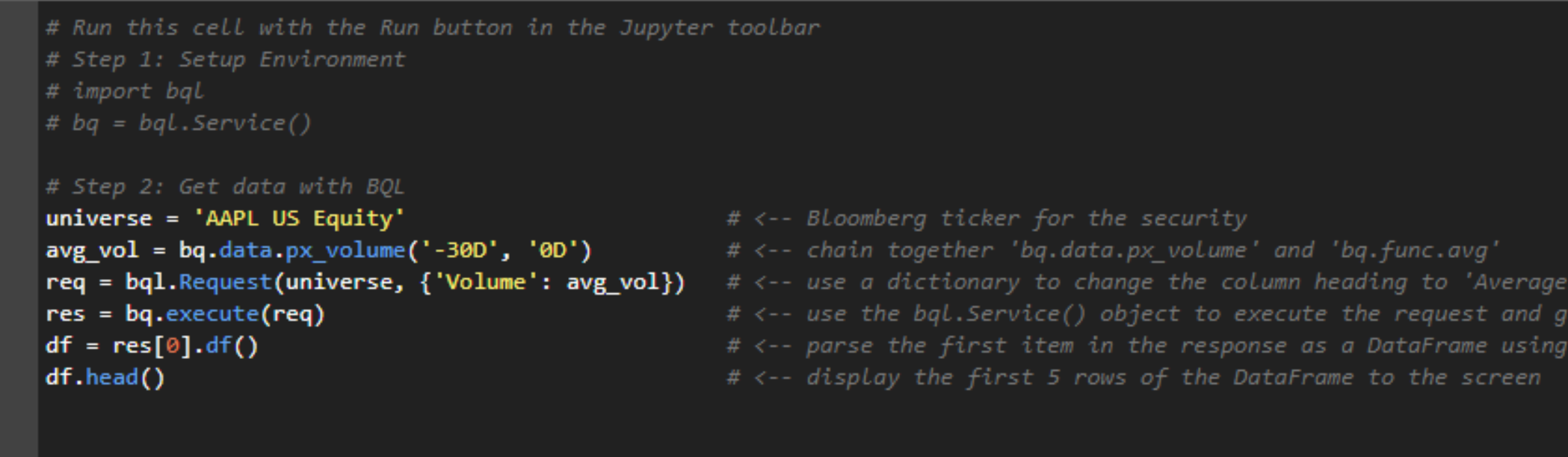
For complex queries, these names could stretch to several hundred characters. We can control the names of these items by passing a dictionary to the bql.Request object instead of a list. These names will be important when we extract the data in the next section.



# Parsing a BQL Response

So far we've explored some of the properties of the bql.Response object, but we haven't seen the actual data we requested. In order to use the data in the response object, we need to call the df() function on the response items. This function will extract the data into a DataFrame: a spreadsheet-like object that we'll discuss later in this notebook.

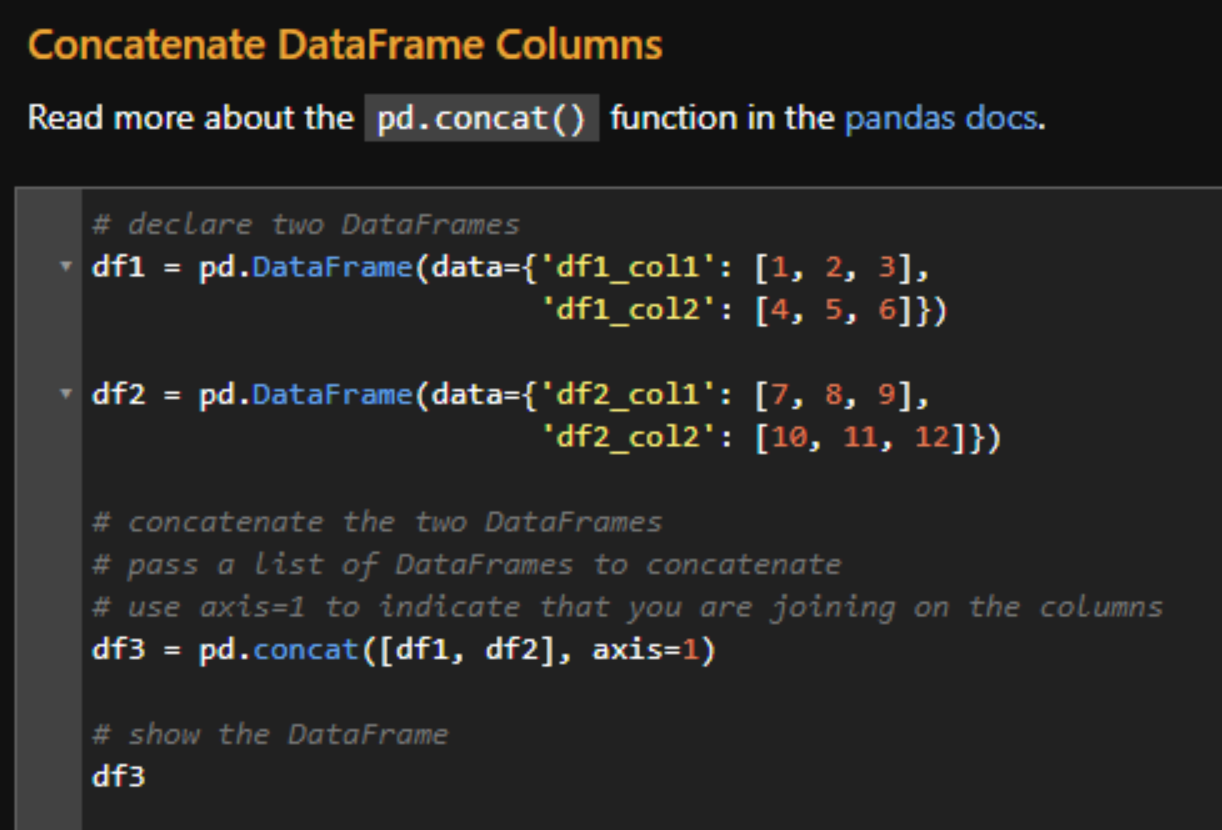
The example below will extract the data into a DataFrame, and print the first 5 rows to the screen. Notice that the column header is the same as the key in our dictionary ("Volume").

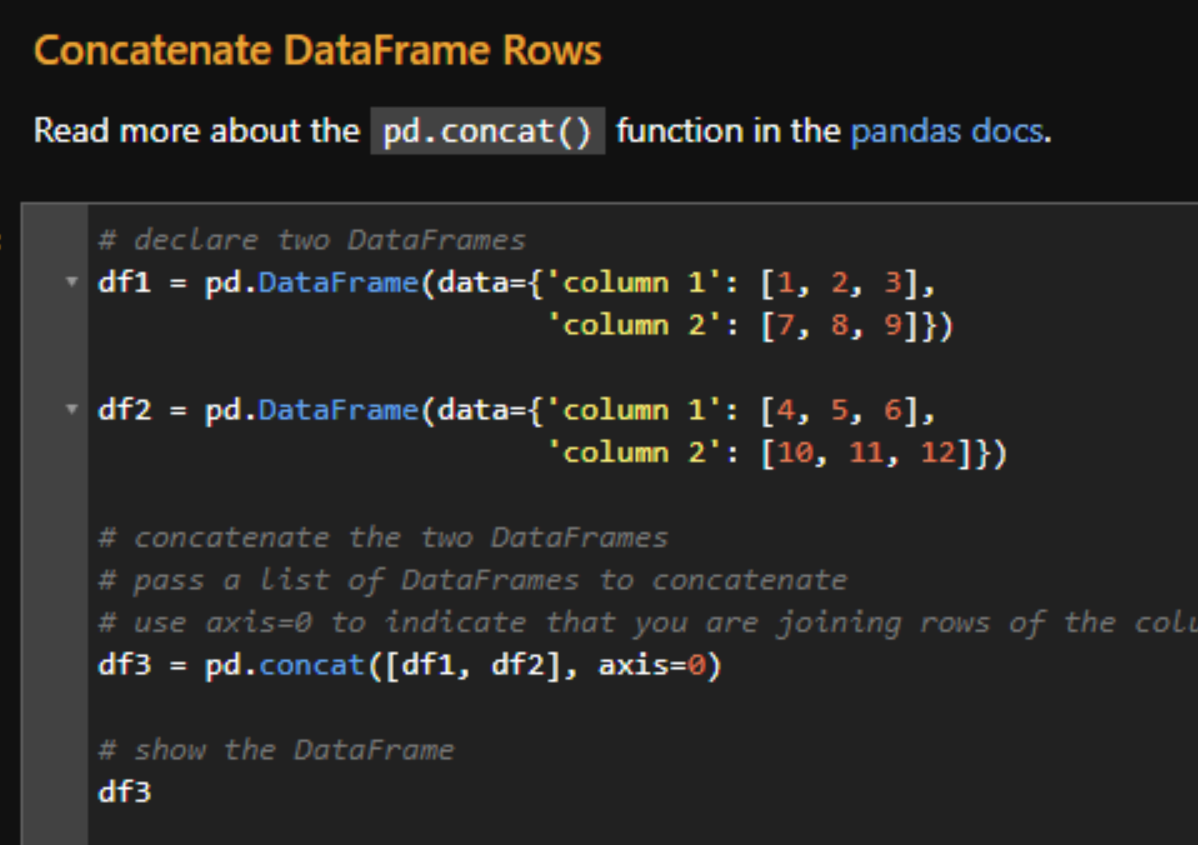


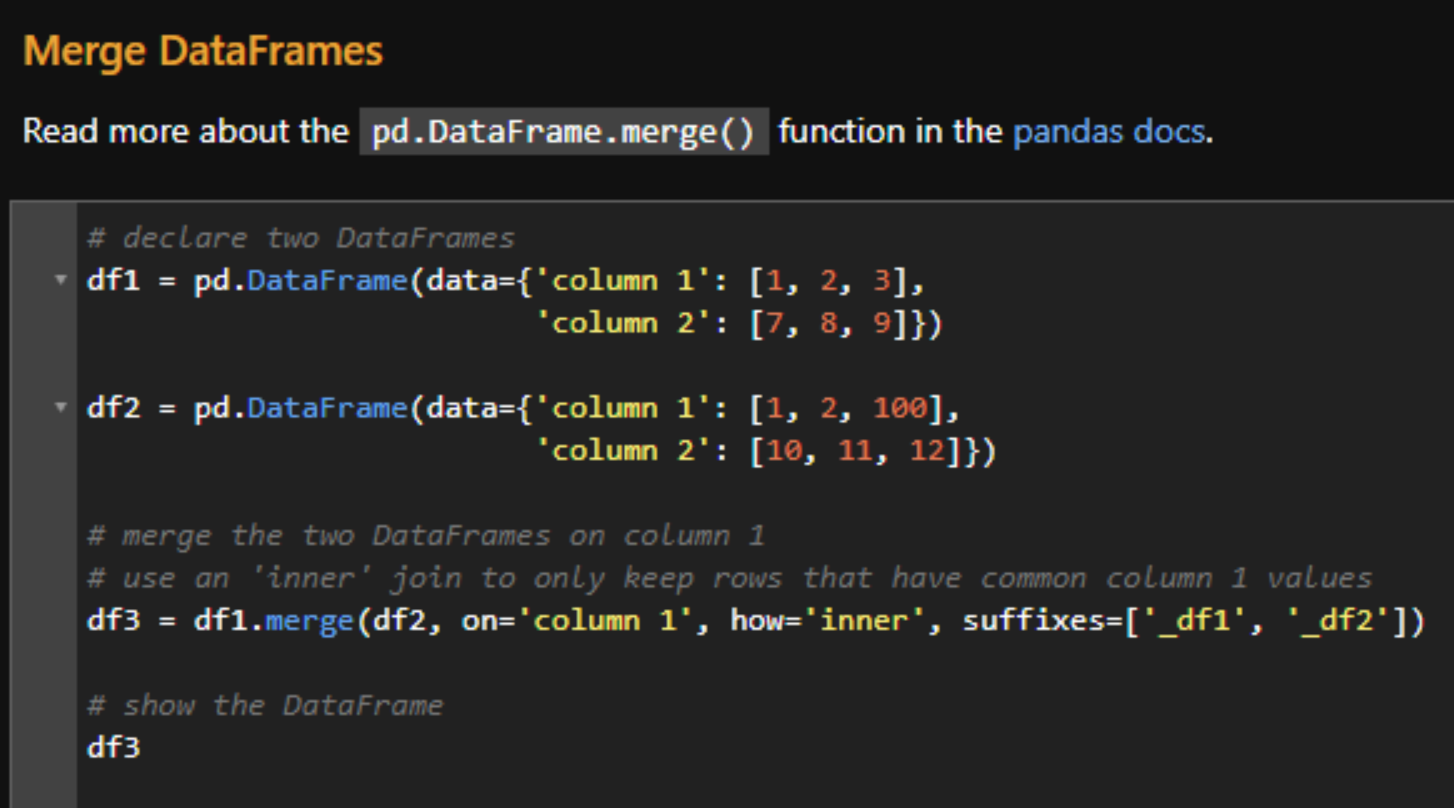
# Concatenate, Merge, and Pivot

Parsing some BQL requests will require you to alter the structure of the DataFrame. The concatenate, merge, and pivot functions of the pandas library could be useful tools depending on the query.

* pd.concat() - combine two or more DataFrames by either rows or columns.
* pd.DataFrame.merge() - join two DataFrames together based on values in a common column
* pd.DataFrame.pivot() - reshape a DataFrame using its values



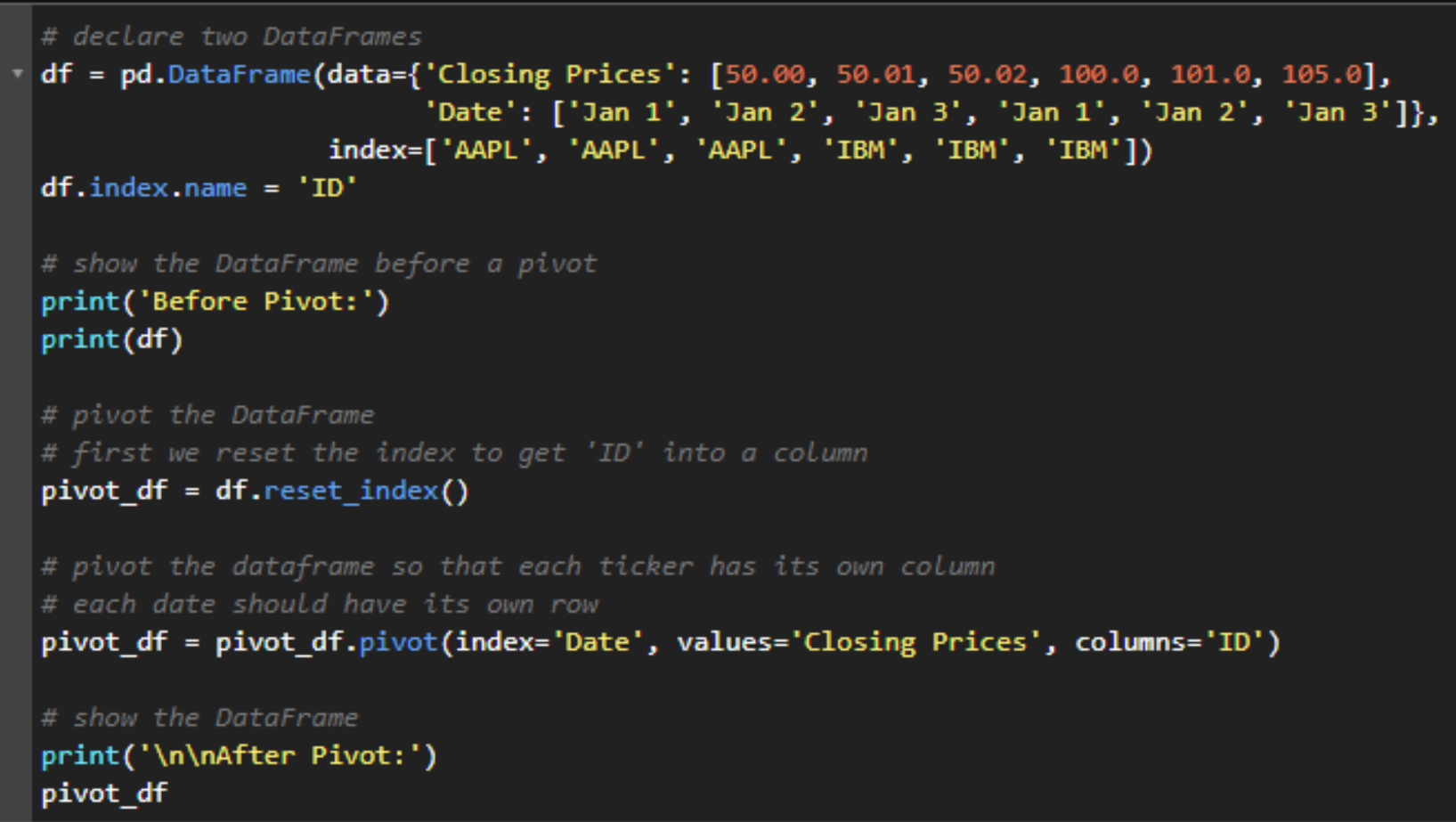




#### Pivot a DataFrame

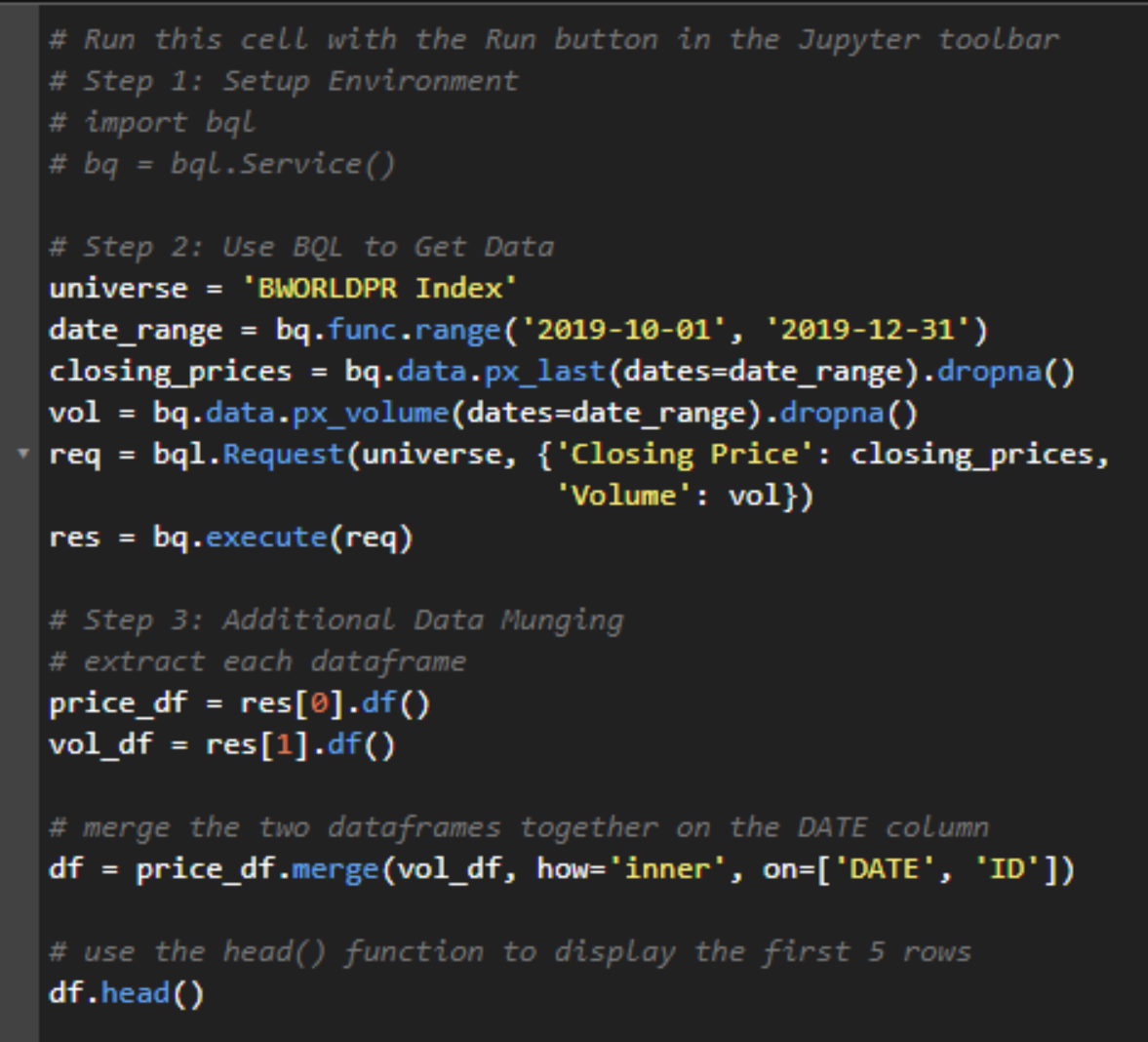
The example below is a common workflow for parsing a single time series dataset requested for multiple universes. Our DataFrame df contains three days of closing prices for the stocks of both AAPL and IBM. The raw response from BQL will place the rows for the two requests on top of each other. For analysis purposes, we want each ticker in its own column and each date to have its own row. The pivot function of the pandas library can help us accomplish this.

Read more about the pd.DataFrame.pivot() function in the [pandas docs](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.pivot.html).



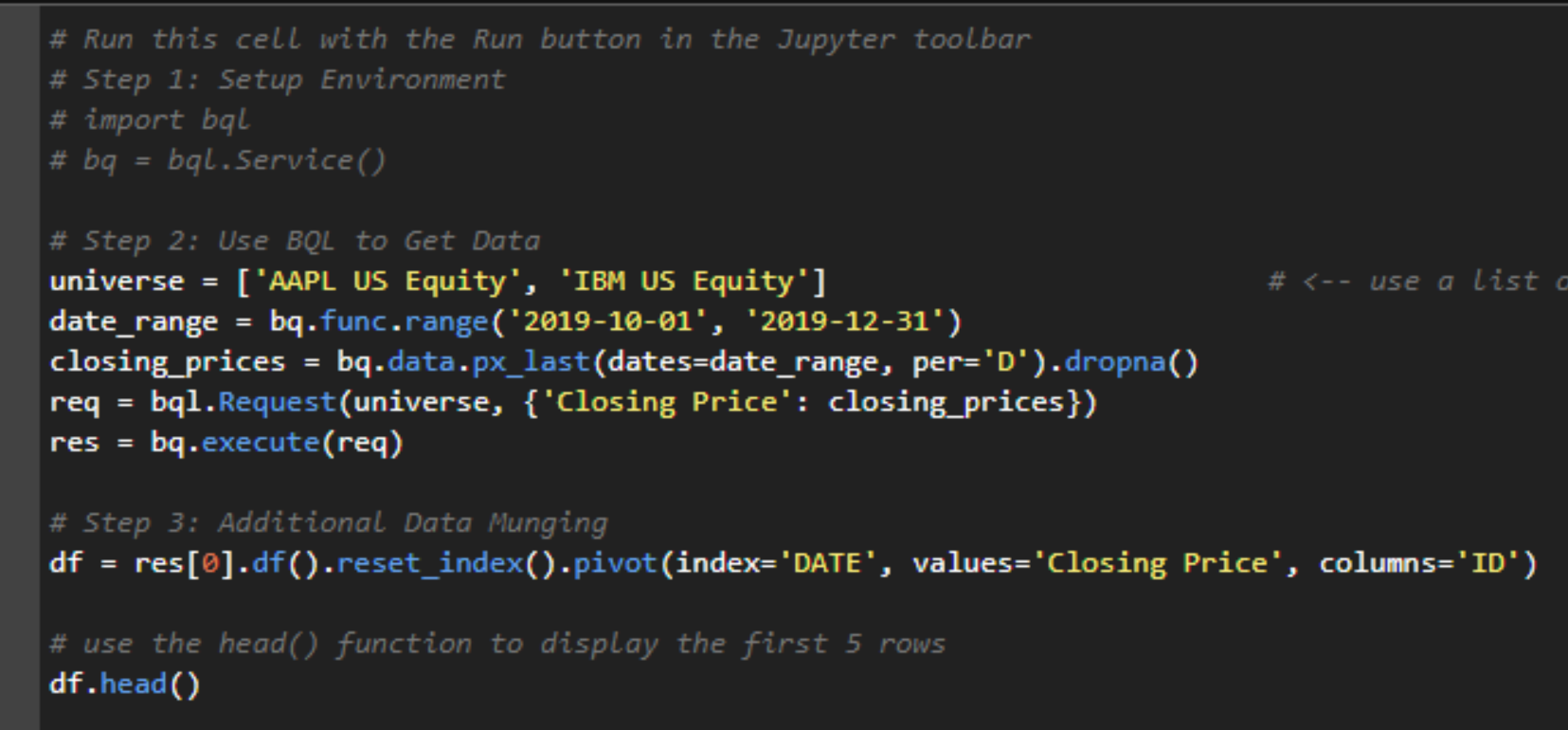
# BQL Example One Universe, Multiple Data Items

Get two time series datasets (daily price and volume) for one universe (BWORLDPR Index). Use pd.DataFrame.merge() to join the results.



# BQL Example Multiple Universes, One Data Item

Get one dataset (daily prices) for multiple universes (AAPL and IBM). Use pd.DataFrame.pivot() to reshape the DataFrame.



# Multiple Universes, One Data Item

Get one dataset (daily prices) for multiple universes (AAPL and IBM). Use pd.DataFrame.pivot() to reshape the DataFrame.

