Week 5 Technical Lab

Working with APIs

Trevor Thomas

College of Computer and Information Sciences, Regis University

MSDS 610: Data Engineering

Paul Andrus

June 7, 2020

Overview and Objectives

An application programming interface, or API, is a protocol for communication between programming languages and various types of applications such as websites, other programming languages, databases, and other software or hardware. In this lab, I will utilize two different APIs to pull data from a website and then store it in a MongoDB database using Python. The primary tools I will use are a Python shell, MongoDB, the pymongo interface, and Jupyter Notebooks (I will mostly use the Python shell, however I prefer the Jupyter Notebooks format for displaying dataframes, so I will briefly switch for that purpose). The first API will collect weather prediction data from the website metaweather.com using the requests library in Python. The second API will be pymongo, an interface between the Python language and the MongoDB database. Once the data has been collected, prepared, and stored, I will switch over to a MongoDB shell to query the data to confirm whether I was successful. These steps are outlined in detail below.

Methods and Results

Pull API Weather Data using Python

First, I launched my bash terminal and connected to the API using the curl command and the website url with the location code for Denver. weather This successfully displayed the data in JSON format to my console.

dpredbeard@dpred:~\$ curl https://www.metaweather.com/api/location/2391279/

Next, I used python to launch a Python shell within my terminal. I imported the requests library, and then assigned the contents of the API get request to a variable called "response." I then ran response to ensure there were no errors, followed by response.json() to verify the contents. The output of the

final response.json() request is too long to print, but it came back identical to the initial curl command indicating everything was successful.

```
dpredbeard@dpred:~$ python
Python 3.7.6 | packaged by conda-forge | (default, Jan 7 2020, 20:28:53)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import requests as req
>>> response = req.get('https://www.metaweather.com/api/location/2391279/')
>>> response
<Response [200]>
>>> response.json()
```

Load Data into Pandas Dataframe

I imported the pandas library and built a dataframe containing the normalized data from the original JSON objects. I used the head command to print the top of the dataframe, but because the entire data-set is a list of JSON objects linked to the key "consolidated weather," it didn't print much.

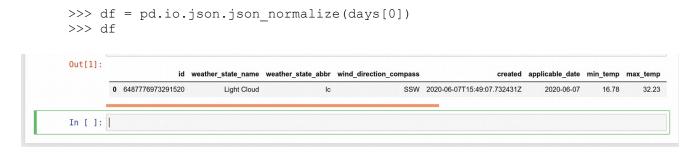
```
>>> import pandas as pd
>>> df = pd.io.json.json_normalize(response.json())
>>> df.head
<bound method NDFrame.head of
consolidated_weather ... parent.latt_long
0 [{'id': 5807222762242048, 'weather_state_name'... 38.997921,-
105.550957
[1 rows x 15 columns]>
>>>
```

To correct this, I needed to work directly with the data in the list associated with the "consolidated_weather" key. I used the same response.json() command with the ['consolidated_weather'] index to bypass that initial key and assign the list of JSON objects to a variable called "days." Then I was able to call that variable with a [0] index to print the first record in the data-set.

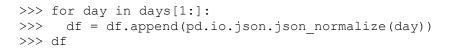
```
>>> days = response.json()['consolidated_weather']
>>> days[0]

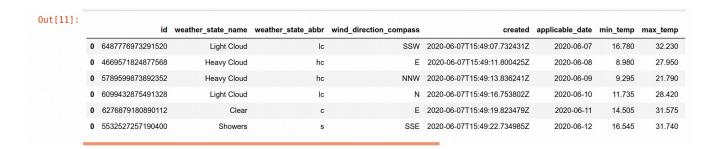
{'id': 5807222762242048, 'weather_state_name': 'Showers',
   'weather_state_abbr': 's', 'wind_direction_compass': 'SSE', 'created': '2020-06-05T21:49:08.521483Z', 'applicable_date': '2020-06-05', 'min_temp': 21.335, 'max_temp': 33.675, 'the_temp': 32.765, 'wind_speed': 2.4692056181765163, 'wind_direction': 158.79299413592983, 'air_pressure': 1011.5, 'humidity': 19, 'visibility': 12.269284876322278, 'predictability': 73}
>>>
```

In order to fully build out the dataframe, I initially normalized the first entry in the "days" list and assigned it to the dataframe "df." I called df after that to ensure the output looked correct. I moved from my Python shell to Jupyter Notebooks because the formatting for displaying a dataframe is better in the latter.



To complete the dataframe, I used a for loop to iterate through the rest of the "days" variable, appending each JSON object in the data-set as a new row in the dataframe one at a time. I then called the dataframe to display using df.





Prepare Data for Loading into Database

Before I could load this dataframe into the MongoDB database, there were a few steps I needed to complete. First, to maintain the proper format, I converted the date and time data from their initial

string format into date-time data. I can see they were initially stored as strings by calling df.info() and observing their data-type as "object."

To convert these, I used the to_datetime() method. I converted the "created" column to GMT time, and I kept the "applicable date" column in Mountain Time.

Then I just needed to drop two columns, "id" and "weather state abbr," and I was ready to load into

MongoDB. I called df again and saw the columns were successfully removed.

```
>>> df.drop(['weather_state_abbr', 'id'], inplace=True, axis=1)
>>> df
```

weather_state_name		wind_direction_compass	created	applicable_date	min_temp	max_temp	the_temp	wind_speed	wind_direction	air_pr
0	Light Cloud	SSW	2020-06-07 15:49:07.732431+00:00	2020-06-07 00:00:00-06:00	16.780	32.230	29.430	11.272583	198.499644	
0	Heavy Cloud	E	2020-06-07 15:49:11.800425+00:00	2020-06-08 00:00:00-06:00	8.980	27.950	26.610	6.735851	81.012834	
0	Heavy Cloud	NNW	2020-06-07 15:49:13.836241+00:00	2020-06-09 00:00:00-06:00	9.295	21.790	17.685	9.316595	326.878304	
0	Light Cloud	N	2020-06-07 15:49:16.753802+00:00	2020-06-10 00:00:00-06:00	11.735	28.420	23.310	3.996994	359.016582	
0	Clear	Е	2020-06-07 15:49:19.823479+00:00	2020-06-11 00:00:00-06:00	14.505	31.575	27.080	3.093010	92.705597	
0	Showers	SSE	2020-06-07 15:49:22.734985+00:00	2020-06-12 00:00:00-06:00	16.545	31.740	30.350	4.304676	161.000000	

Create MongoDB Database and Load Data-set

With the data clean and prepared, I opened a new terminal window and opened the Mongo shell with mongo. I entered use weather-test to create the new database.

```
dpredbeard@dpred:~$ mongo
> use weather_test
switched to db weather_test
> db
weather_test
>
```

>>> from pymongo import MongoClient

Then I returned to the Python shell and imported the pymongo library. I loaded the data using the

 $. \verb|insert_many|()| method and then printed one of the records to the console with \verb|.find_one|()|.$

```
>>> client = MongoClient()
>>> db = client['weather_test']
>>> collection = db['denver']
>>> collection.insert_many(df.to_dict('records'))
<pymongo.results.InsertManyResult object at 0x7f0b8f689460>
>>> collection.find_one()

{'_id': ObjectId('5edd40e022ad5a0c2c8659cd'), 'weather_state_name': 'Light Cloud', 'wind_direction_compass': 'SSW', 'created': datetime.datetime(2020, 6, 7, 15, 49, 7, 732000), 'applicable_date': datetime.datetime(2020, 6, 7, 6, 0), 'min_temp': 16.78, 'max_temp': 32.230000000000004, 'the_temp': 29.43, 'wind_speed': 11.272582862848964, 'wind_direction': 198.49964422695342, 'air_pressure': 998.0, 'humidity': 18, 'visibility': 14.455890101805457, 'predictability': 70}
>>>
```

With the .find_one() command returning one of the records, I was confident the data was loaded successfully. I then switched back to the MongoDB shell to query the data and confirm success. I

began by using show dbs to view the databases, and then use weather test to enter to correct one.

I also used show collections to see if the "denver" collection appeared.

```
> show dbs

admin 0.000GB

config 0.000GB

local 0.000GB

parking 1.170GB

weather_test 0.000GB

> use weather_test

switched to db weather_test

> show collections

denver
```

I ran .count() and .findOne() methods on the collection as well.

```
> db.denver.count()
> db.denver.findOne()
      " id" : ObjectId("5edd40e022ad5a0c2c8659cd"),
      "weather state name" : "Light Cloud",
      "wind direction compass" : "SSW",
      "created" : ISODate("2020-06-07T15:49:07.732Z"),
      "applicable date" : ISODate("2020-06-07T06:00:00Z"),
      "min_temp" : 16.78,
"max_temp" : 32.23000000000004,
      "the temp" : 29.43,
      "wind speed": 11.272582862848964,
      "wind direction" : 198.49964422695342,
      "air pressure" : 998,
      "humidity" : 18,
      "visibility" : 14.455890101805457,
      "predictability": 70
}
```

Finally, I used .find(), .sort(), .limit(), and .pretty() to view the latest weather prediction (6/12/2020) and the soonest (6/07/2020).

```
> db.denver.find().sort({'applicable_date': -1}).limit(1).pretty();
{
    "_id" : ObjectId("5edd40e022ad5a0c2c8659d2"),
    "weather_state_name" : "Showers",
    "wind_direction_compass" : "SSE",
    "created" : ISODate("2020-06-07T15:49:22.734Z"),
    "applicable_date" : ISODate("2020-06-12T06:00:00Z"),
    "min_temp" : 16.545,
    "max_temp" : 31.74,
    "the_temp" : 30.35,
    "wind_speed" : 4.304675693947347,
    "wind_direction" : 160.9999999999997,
    "air_pressure" : 1018,
    "humidity" : 25,
```

```
"visibility" : 9.999726596675416,
      "predictability" : 73
> db.denver.find().sort({'applicable date': 1}).limit(1).pretty();
      " id" : ObjectId("5edd40e022ad5a0c2c8659cd"),
      "weather state name" : "Light Cloud",
      "wind direction compass" : "SSW",
      "created": ISODate("2020-06-07T15:49:07.732Z"),
      "applicable date" : ISODate("2020-06-07T06:00:00Z"),
      "min temp" : 16.78,
      "max temp" : 32.230000000000004,
      "the temp" : 29.43,
      "wind speed": 11.272582862848964,
      "wind direction" : 198.49964422695342,
      "air pressure" : 998,
      "hum\overline{i}dity" : 18,
      "visibility" : 14.455890101805457,
      "predictability" : 70
}
```

It was clear after these queries that the data-set was successfully stored in MongoDB and would be ready for further analysis in a real-world scenario.

Discussion

In this week's assignment, I used a website API as well as the pymongo API to take data from the website, manipulate it into a pandas dataframe, and ultimately store it in a MongoDB database. I first used the requests library to pull the data using the metaweather.com API. This was an extremely user-friendly API that did not require authentication or sending anything to the server, so I was able to do it with the .get() method from requests. Then I used several pandas methods to view and eventually store the data in a dataframe in the desired format. Finally, the data was saved in the new MongoDB database called "weather_test." Several queries were used to confirm success, all of which came back as expected. Ultimately, this assignment was straightforward enough that it does not lend itself toward substantial discussion. The objective was to work with different APIs and build an understanding and comfort around them while continuing to build skills around querying and building

databases. I certainly feel I achieved these goals, and I now have more confidence obtaining data from sources other than Kaggle, which has effectively been my crutch in the past.