Analyzing Millions of NYC Parking Violations

For this project, you will be tasked with loading and then analyzing a dataset containing millions of NYC parking violations since January 2016. In completing this exercise, you will demonstrate mastery of principles of containerization, terminal navigation, python scripting, artifact deployment and AWS EC2 provisioning.

[**Overview and Requirements**](#_ptpb31jlvhgb) **2**

[Project Breakdown and Submission Links](#_jff6abu3v873) 2

[Project Context](#_lkzlkno4jnaf) 2

[Before You Begin](#_u0rr0ravy1kv) 3

[**Part 1: Python Scripting**](#_m494fetmrwxj) **3**

[Inputs/Outputs](#_lxo4hj1u2iup) 3

[Libraries](#_vic3oaezkhr0) 4

[Submission Guidelines](#_2umqp1m0z5v0) 5

[**Part 2: Loading into ElasticSearch**](#_ps1guzi31ivj) **5**

[Inputs/Outputs](#_cknnau4fursw) 6

[Libraries](#_czxegrakhs32) 6

[Submission Guidelines](#_accg7v5kni0x) 6

[**Part 3: Visualizing and Analysis on Kibana**](#_kqqcluwl7k8z) **6**

[Inputs/Outputs](#_tipcyydvkouc) 7

[Submission Guidelines](#_5uvcjnenw68y) 7

[**(Optional) Part 4: Deploying to EC2 Instance**](#_y7r1ltm4aico) **8**

# 

# 

# 

# 

# Overview and Requirements

Before delving into the specifics, let’s cover the high level requirements and due dates. For each portion, you will have to submit a Github URL containing your code and/or a Dockerhub URL containing your docker image.

**It is imperative that your code run on any environment I should choose to run it on**. (This may be an AWS EC2 instance, my local machine, both, or some other container supporting environment).

Basically - **test your code somewhere other than your local environment** before submitting!

## Project Breakdown and Submission Links

This project will be broken into three parts, each part building on the work done from the previous. The parts may or may not be due at the same date - this is mainly to help you pace yourself without getting overwhelmed.

* Part 1: Python Scripting | [Submission form](https://airtable.com/shrz5md8xHXNSzuqb)
* Part 2: Loading into ElasticSearch | [Submission form](https://airtable.com/shrAPiWaJC8ORYLnL)
* Part 3: Visualizing and Analysis on Kibana | [Submission form](https://airtable.com/shrSRVCWsP4immjn3)
* (Optional) Part 4: Deploying to EC2 Instance | [Submission form](https://airtable.com/shr0pP2bMn7FkfUdi)

**NOTE:** The solution to Part 1 will be provided to you before Parts 2&3 are due.

## Project Context

The [**NYC Open Data**](https://opendata.cityofnewyork.us/)project makes available freely data published by NYC agencies and other partners. These datasets range from a few thousand rows to millions, depending on department and time frame.

For the really large datasets, attempting to download is unfeasible as some of these files are upwards of 5-10 Gbs in size. As such, this service offers an Application Programming Interface - known colloquially as an **API** - for ease of querying and loading the data via web requests in terminal (via curl) or code (via python, javascript, etc).

Furthermore, these APIs are made available via the [**Socrata Open Data API**](https://dev.socrata.com/)**,** which provides a well established and easy to use set of conventions for querying public datasets such as the one we will be using.

For this project, we will leverage a python client of the Socrata API to connect to the [**Open Parking and Camera Violations**](https://dev.socrata.com/foundry/data.cityofnewyork.us/nc67-uf89) (OPCV) API, load all the data into an ElasticSearch instance, and visualize / analyze with Kibana.

To accomplish this, we will leverage our knowledge of containerization, working with the terminal and python scripting. We will “share” our findings by deploying our docker images to Dockerhub and our code to Github.

## Before You Begin

Make sure you have the following:

* A Github account
* A Dockerhub account (you should have this if using docker on Windows/Mac)
* An [**app token**](https://data.cityofnewyork.us/profile/edit/developer_settings)for the NYC Open Data API
* An AWS account (for running EC2, etc)

# Part 1: Python Scripting

**Due Date: Sunday** March 1st, 2020 - Midnight.

**NOTE**: The solution to **Part 1** will be made available **before** **Part 2 and 3** are due.

In the first part, you simply want to develop a python **command line interface** that can connect to the **OPCV** API and demonstrate that the data is accessible via python.

Your script must be able to run within docker but take parameters from the command line. It should also support having the option to print results out to a file.

## Inputs/Outputs

Here are all the command line arguments your script must support:

**$ docker run -e APP\_KEY={YOUR\_APP\_KEY} -t bigdata1:1.0 python main.py --page\_size=1000 --num\_pages=4 --output=results.json**

Some key arguments here:

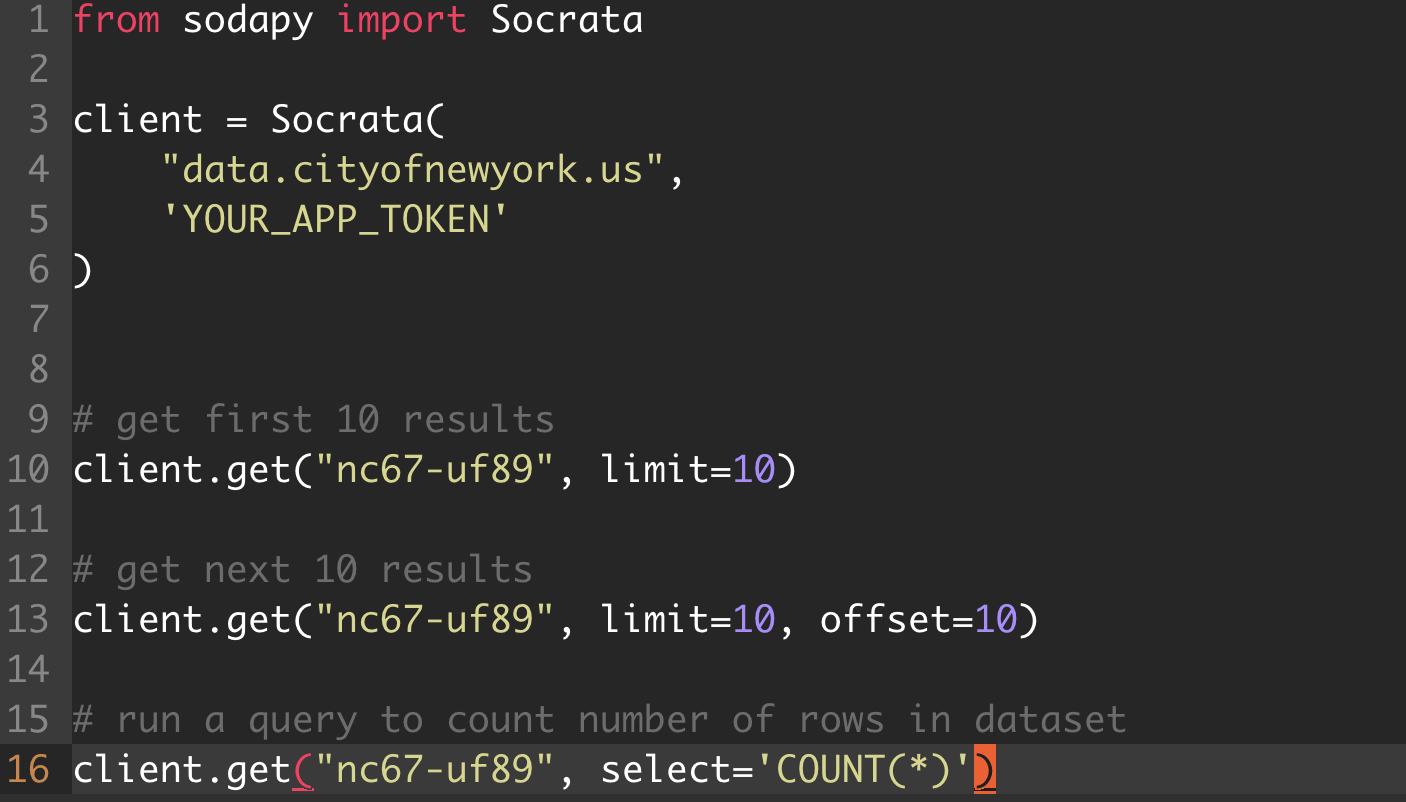
* **APP\_KEY:** This is how a user can pass along an **APP\_KEY** for the api in a safe manner. **DO NOT COMMIT YOUR (actual) APP KEY** to Github! Also, **APP\_KEY** should not be “hardcoded” anywhere in your source code.
* **bigdata1:** This is the name of your docker image. We want the output for Part 1 to be **version 1**.
* **--page\_size:** This command line argument is **required**. It will ask for how many records to request from the API per call.
* **--num\_pages:** This command line argument is **optional**. If not provided, your script should continue requesting data until the entirety of the content has been exhausted. If this argument is provided, continue querying for data **num\_pages** times.
* **--output:** This command line argument is **optional**. If not provided, your script should simply print results to stdout. If provided, your script should write the data to the file (in this case, **results.json**).

It is expected that stdout or results.json will contain the API response, which is simply rows and rows of data from the API within the confines of the parameters provided to the script.

## Libraries

For this script, you must use PyPI’s [**sodapy**](https://github.com/xmunoz/sodapy)module. This module makes it seamless to connect to your Socrata API and provides a high level interface for providing additional parameters such as page offsets.

Your project should have a **src** folder containing functions for interacting with the **OPCV** API and for managing the API response. Your **main.py** script should live in the root folder at the same level as your **Dockerfile** and **requirements.txt**

Below, find an example of the **sodapy** module being leveraged to make a call out to the **OPCV** API. 

Note: **nc67-uf89** is the **id** for our dataset.

**Food for Thought** 🤔🤔: Why would you ever need to use the **select=’COUNT(\*)’** argument?

## Submission Guidelines

For this part, I expect the following submissions:

* Github URL containing source code for your command line tool
  + I want to see a commit history! (That is, there should be plenty of commits to your repository demonstrating an iterative approach to your implementation).
* Dockerhub URL pointing to repo with the image defined in your Dockerfile

For this part of the project, **YOU DO NOT NEED TO LOAD ALL 48 MILLION ROWS** of data. To demonstrate that the application is working, it is sufficient to pass in a few combinations of **page\_size** and **num\_pages.**

[Submission form](https://airtable.com/shrz5md8xHXNSzuqb)

# 

# Part 2: Loading into ElasticSearch

**Due Date:** March 10th, 2020 - Midnight.

In this second part, you want leverage **docker-compose** to bring up a service that encapsulates your **bigdata1** container and an **elasticsearch** container and ensures that they are able to interact.

You must update your original script (from Part 1) to now not only download the data but also **load it into your elasticsearch instance**.

## Inputs/Outputs

Same inputs as **Part 1**, however you must demonstrate that your script now has the ability to push the API results into elasticsearch.

In order to do this, it is sufficient to simply run a few **curl** requests in terminal against **http://localhost:9200**.

Here is a [**resource**](https://okfnlabs.org/blog/2013/07/01/elasticsearch-query-tutorial.html#curl-or-browser) with more information on querying elasticsearch via curl.

## Libraries

In order to interact with elasticsearch in python, PyPI’s [elasticsearch](https://github.com/elastic/elasticsearch-py) module ought to be used. Additionally, all elasticsearch related logic should be wrapped into an internal module that lives within the **src** folder in your project.

## Submission Guidelines

For this portion, I expect the following:

* Github URL containing source code for your command line tool

For this portion of the project, **YOU DO NOT NEED TO LOAD ALL 48 MILLION ROWS** of data. To demonstrate that the application is working, it is sufficient to pass in a few combinations of **page\_size** and **num\_pages.** To demonstrate the elasticsearch feature, it is sufficient to commit an **output.txt** file that contains a few **curl** requests to elasticsearch and responses that prove that API results have been adequately loaded.

[Submission form](https://airtable.com/shrAPiWaJC8ORYLnL)

# Part 3: Visualizing and Analysis on Kibana

**Due Date:** March 10th, 2020 - Midnight.

In this third part, you want to stand up an instance of Kibana on top of your ElasticSearch instance in order to visualize and analyze your dataset.

There will be an update required in your script - in order to properly index information on kibana, we will want to properly define a **time field**. The **OPCV** dataset does contain an **issue\_date**  field that would be a good candidate for this definition. As part of this exercise, come up with a way to parse this field - which is stored as **text** - into a python **datetime** field.

## Inputs/Outputs

No real difference in inputs. The only code change required is to ensure that the **issue\_date** that is being loaded into elasticsearch is transformed into a datetime before load.

As far as output goes, configure Kibana to pull items from the index defined when loading data into elasticsearch. This should load up the resulting data into the Kibana API and allow you to do some interesting analysis.

Have some fun with this and try to come up with a unique analysis as you explore the capabilities of Kibana - some questions to consider answering (in the form of visualizations or graphs):

* Which county had the highest average reduction amount?
* Which violation was most popular? Second most popular? Etc

Create 4 visualizations in Kibana that analyze the data loaded and presents analysis in graphical form. Here is the [inspiration](https://datascience-enthusiast.com/Miscellaneous/NYC311calls_Elastic_stack.html) for this.

## Submission Guidelines

For this portion, I expect the following:

* Github URL containing source code for your command line tool

For this portion of the project, **try to get a decent chunk** of data. To demonstrate that the application is working, simply update your Github README detailing your analysis. Take screenshots of your kibana visualizations and embed them into the README to form a compelling “blog post” describing your results.

[Submission form](https://airtable.com/shrSRVCWsP4immjn3)

# (Optional) Part 4: Deploying to EC2 Instance

For this optional portion, attempt to provision an EC2 instance to run your **docker-compose** services. As part of this exercise - load **all 48 million rows** into elasticsearch (your EC2 will likely have to run for a while to achieve this).

Additionally, you will have to poke around AWS configs to allow Kibana to be accessible via public IP address (main reason this is an optional part). This is a fun (albeit arduous) rabbithole that will help you better understand how EC2s are by default protected/cut off from the internet.

If you manage to pull this off, great job! Just DM me directly with your Kibana URL running on EC2 and we can take it from there. I will give **5 additional points**