**Open Data NYC and Big Data for all**

By Joseph DeArce 02/25/18

What is Open Data NYC and what can you use it for? Open Data NYC is a data repository of NYC operational datasets in various formats. These datasets can be downloaded but be warned some of them can be quite large such as the Yellow Taxi Trip datasets. This group of datasets are organized by year and one year can contain as much as 170 million rows of data. Zipped they can be on the order of fifteen gigs in size and when decompressed they are on the order of thirty Gigs. The Yellow Taxi Trip datasets are the largest at this repository all the others are much smaller many of them are GIS (Geographical Information System) datasets.

So, as you can see you will need a very powerful computer to process these datasets and load them. I have worked extensively with the 2015 Yellow Taxi Trip data for January through June of that year, see hardware below. There have been some issues but these have been minor problems, which I overcame and you can too.

**When was the Office of Data Analytics created**

The Mayor's Office of Data Analytics (MODA) is New York City's civic intelligence center, allowing the City to aggregate and analyze data from across City agencies, to more effectively address crime, public safety, and quality of life issues.  Established by [**Executive Order 306**](http://www.nyc.gov/html/om/pdf/eo/eo_306.pdf), during the Mayor Bloomberg’s administration on April 17 2013. The office uses analytics tools to prioritize risk more strategically, deliver services more efficiently, enforce laws more effectively and increase transparency.

The office's core functions include:

* Collaboration with City agencies to implement data-driven solutions to City service delivery issues.
* Building a Citywide data platform to facilitate data sharing
* Oversight of Citywide Data Projects
* Implementation the City's Open Data Law

URL: <http://www1.nyc.gov/site/analytics/index.page>

**What is MODA**

What is the **Mayor’s Office of Data Analytics** (MODA) and what is its mission, it is to open up operational administrative data to citizens and public organizations within the City and beyond?

Open Data is an opportunity to engage New Yorkers in the information that is produced and used by the City government. They believe that every New Yorker can benefit from Open Data, and Open Data can benefit from every New Yorker. The meaning is that open data enables governmental transparency and empowers ordinary citizens to look at this data in new ways and find new insights.

This is brought to us by the **Mayor’s Office of Data Analytics** (MODA) and the **Department of Information Technology and Telecommunications** (DoITT) partner to form the Open Data team. As a hub of analytics in the City, **MODA** advocates for the use of Open Data in citywide data analytics and in the community. **DoITT** manages the technical operations with City agencies and our **vendor partner Socrata**, is ensuring that technological capabilities are always evolving to better meet user needs. Agencies are the data owners and have Open Data Coordinators who serve as the primary point of contact with the Open Data team.

These three entities, along with the **City Council**, **advocates**, and the **civic tech community**, work together to achieve Open Data for All.

**Who am I?**

I am an Oracle DBA with 15 years of experience in my field and before then I was a PL/SQL, Java, and SAS Developer.

My test machine is a laptop with a Core 7i 4 way.

**My Software**

I have installed the following software products they are:

* Oracle 12.2.0.1 database.
* PL/SQL developer and PL/SQL Developer Data Modeler.
* Oracle 12.1.0.2 client.
* Spark.
* Scala.
* R, RStudio, and ROracle.
* Eclipse.

The difference between the database and the client is only because I upgraded my database to the current release and I wanted to benchmark the differences between 12 R2 and 12R1 which I already have the benchmarks for.

I am currently constructing a new table that will incorporate of all the Yellow Taxi Trip datasets from 2009-2016. This dataset will contain around 1.1 billion rows. I intend to analyze this table and test Oracle 12.2.0.1 for performance against my benchmarks on Oracle 12.1.0.2 on the same data. My new desktop is running a core i7 running with two helium based 10TB mechanical drives. I will rerun my benchmark queries from my previous test and compare the results and test my new machine with the new drives which will give me a performance boost in IO.

**My Hardware**

Before we begin our installation, process lets describe our test system; it has the following configuration:

**Laptop:**

* An Acer Laptop running Windows 10 with a Core7i quad processor,
* 16GB of memory.
* 1TB hard drive (not SSD).
* Installed is an Oracle 12c Enterprise database with an SGA of 4GB
* This database has a custom configuration.
* There are six PDB’s on the Oracle 12c R2 instance.

**Other NYC Open Government projects**

Explore other data and open information initiatives going on around the City.

Map and chart crime statistics with **CompStat** and traffic data with **TrafficStat** from the New York City Police Department.

* CompStat: <https://compstat.nypdonline.org/>
* TrafficStat: <https://trafficstat.nypdonline.org/>

Check out the **Street Tree Map** to visualize data collected in NYC Parks’ 2015 TreesCount! street tree census.

* Street Tree Map: <https://tree-map.nycgovparks.org/>

File a Freedom of Information Law (FOIL) request with the OpenRecords app.

* **OpenRecords**: <https://a860-openrecords.nyc.gov/>

Search the Government Publications Portal to view City documents dating from the 19th Century to present.

* **Government Publications Portal:** <http://a860-gpp.nyc.gov/>

Explore 311 data in your neighborhood or other parts of the City leveraging neighborhoods.nyc

* **neighborhoods.nyc:** <http://www.neighborhoods.nyc/welcome.html>

Submit an idea to a City agency’s challenge or sign up for a City agency’s hackathon with the Call for Innovations

* **Call for Innovations:** <http://www.nyc.gov/html/cfi/html/index.html>

**Our test Data Sets**

We will be using several data sets **Restaurant inspection**, **Companies doing business in NYC**, and the **Yellow Taxi trip data** which are part of the **NYC Open Data initiative** see the URL’s and descriptions below:

1. **Yellow Taxi trip data 2015**

We will be using **Yellow Taxi trip data** which is a GIS data set for the year 2015. This data set has data for 2015 for NYC from January through June of that year. The table has over 78 million records and is 11 GB in size before it was up loaded into Oracle. This took 2 hours and 24 minutes.

**URL:** <https://data.cityofnewyork.us/Transportation/2015-Yellow-Taxi-Trip-Data/ba8s-jw6u>

<https://data.cityofnewyork.us/browse?q=yellow+trip+data&provenance=official>

This dataset includes trip records from all trips completed by yellow taxis from in NYC from January to June in 2015. Records include fields capturing pick-up and drop-off dates/times, pick-up and drop-off locations, trip distances, itemized fares, rate types, payment types, and driver-reported passenger counts. The data used in the attached datasets were collected and provided to the **NYC Taxi and Limousine Commission** (TLC) by technology providers authorized under the Taxicab Passenger Enhancement Program (TPEP). The trip data was not created by the TLC, and TLC makes no representations as to the accuracy of these data.

**Data Dictionary YELLOW\_TAXI\_TRIP\_JUNE\_BK**

VENDORID1 VARCHAR2(40 BYTE)

TPEP\_PICKUP\_DATETIME TIMESTAMP(6)

TPEP\_DROPOFF\_DATETIME TIMESTAMP(6)

PASSENGER\_COUNT1 VARCHAR2(40 BYTE)

TRIP\_DISTANCE1 VARCHAR2(40 BYTE)

PICKUP\_LONGITUDE1 VARCHAR2(40 BYTE)

PICKUP\_LATITUDE1 VARCHAR2(40 BYTE)

RATECODEID1 VARCHAR2(40 BYTE)

STORE\_AND\_FWD\_FLAG1 VARCHAR2(40 BYTE)

DROPOFF\_LONGITUDE1 VARCHAR2(40 BYTE)

DROPOFF\_LATITUDE1 VARCHAR2(40 BYTE)

PAYMENT\_TYPE1 VARCHAR2(40 BYTE)

FARE\_AMOUNT1 VARCHAR2(40 BYTE)

EXTRA1 VARCHAR2(40 BYTE)

MTA\_TAX1 VARCHAR2(40 BYTE)

TIP\_AMOUNT1 VARCHAR2(40 BYTE)

TOLLS\_AMOUNT1 VARCHAR2(40 BYTE)

TOTAL\_AMOUNT1 VARCHAR2(40 BYTE)

**Data Dicitionary field details**

**vendor\_id**

A designation for the technology vendor that provided the record. CMT=Creative Mobile Technologies VTS= VeriFone, Inc. DDS=Digital Dispatch Systems

The vendor\_id column is of the [text](https://dev.socrata.com/docs/datatypes/text.html#2.1,) datatype.

**Simple Filters**

To filter the dataset to only return records containing a specified value for vendor\_id simply add a URL parameter to your URL with vendor\_id as the key and your specified value. For example:

**imp\_surcharge**

$0.30 improvement surcharge assessed trips at the flag drop. The improvement surcharge began being levied in 2015.

The imp\_surcharge column is of the [number](https://dev.socrata.com/docs/datatypes/number.html#2.1,) datatype.

**payment\_type**

A numeric code signifying how the passenger paid for the trip. 1= Credit card 2= Cash 3= No charge 4= Dispute 5= Unknown 6= Voided trip

The payment\_type column is of the [text](https://dev.socrata.com/docs/datatypes/text.html#2.1,) datatype.

**rate\_code**

The final rate code in effect at the end of the trip. 1= Standard rate 2=JFK 3=Newark 4=Nassau or Westchester 5=Negotiated fare 6=Group ride

The rate\_code column is of the [text](https://dev.socrata.com/docs/datatypes/text.html#2.1,) datatype.

**trip\_distance**

The elapsed trip distance in miles reported by the taximeter.

The **trip\_distance** column is of the [number](https://dev.socrata.com/docs/datatypes/number.html#2.1,) datatype.

**ratecodeid**

The ratecodeid column is of the [number](https://dev.socrata.com/docs/datatypes/number.html#2.1,) datatype.

**Simple Filters**

To filter the dataset to only return records containing a specified value for ratecodeid simply add a URL parameter to your URL with ratecodeid as the key and your specified value. For example:

**store\_and\_fwd\_flag**

This flag indicates whether the trip record was held in vehicle memory before sending to the vendor, aka “store and forward,” because the vehicle did not have a connection to the server. Y= store and forward trip N= not a store and forward trip

The store\_and\_fwd\_flag column is of the [text](https://dev.socrata.com/docs/datatypes/text.html#2.1,) datatype.

**extra column**

Miscellaneous extras and surcharges. Currently, this only includes the $0.50 and $1 rush hour and overnight charges.

The extra column is of the [number](https://dev.socrata.com/docs/datatypes/number.html#2.1,) datatype.

**mta\_tax**

$0.50 MTA tax that is automatically triggered based on the metered rate in use.

The mta\_tax column is of the [number](https://dev.socrata.com/docs/datatypes/number.html#2.1,) datatype.

**#HERE**

**BUSNESS DATA SETS**

1. **Legally Operating Businesses**

This data set features businesses/individuals holding a DCA license so that they may legally operate in New York City. Note: Sightseeing guides and temporary street fair vendors are not included in this data set. Data set size is

**URL:** <https://data.cityofnewyork.us/Business/Legally-Operating-Businesses/w7w3-xahh>

1. **DOHMH New York City Restaurant Inspection Results Health**

This dataset provides restaurant inspections, violations, grades and adjudication information

**URL:** <https://data.cityofnewyork.us/browse?category=Health>

The dataset contains every sustained or not yet adjudicated violation citation from every full or special program inspection conducted up to three years prior to the most recent inspection for restaurants and college cafeterias in an active status on the **RECORD DATE** (date of the data pull). When an inspection results in more than one violation, values for associated fields are repeated for each additional violation record. Establishments are uniquely identified by their **CAMIS** (record ID) number. Keep in mind that restaurants go in and out of business; only restaurants in an active status are included in the dataset.

Records are also included for each restaurant that has applied for a permit but has not yet been inspected and for inspections resulting in no violations. Establishments with inspection date of **1/1/1900** are new establishments that have not yet received a full inspection. Restaurants that received no violations are represented by a single row and coded as having no violations using the **ACTION** field

**311 SYSTEM OPERATIONAL**.

1. This dataset contains those 311 service requests that were serviced by the NYPD and resolved by them. This dataset provides 311 Service Requests from 2010 to the present. The **Doug\_NYPD\_311\_SRs** CSV file has service requests that were serviced by the NYPD and resolved. This information is automatically updated on a daily basis.

The **Doug\_NYPD\_311\_SRs** CSV file can be downloaded from the URL you see below.

**URL:** <https://data.cityofnewyork.us/Social-Services/Doug_NYPD_311_SRs/ty85-y26w/data>

The dataset contains every service request that the NYPD serviced and resolved. This data set has 53 fields, see the data dictionary below.

|  |  |
| --- | --- |
| **Column Name** | **Description** |
| Unique Key | Unique identifier of a Service Request (SR) in the open data set |
| Created Date | Date SR was created |
| Closed Date | Date SR was closed by responding agency |
| Agency | Acronym of responding City Government Agency |
| Agency Name | Full Agency name of responding City Government Agency |
| Complaint Type | This is the fist level of a hierarchy identifying the topic of the incident or condition. Complaint Type may have a corresponding Descriptor (below) or may stand alone. |
| Descriptor | This is associated to the Complaint Type, and provides further detail on the incident or condition. Descriptor values are dependent on the Complaint Type, and are not always required in SR. |
| Status | Status of SR submitted |
| Due Date | Date when responding agency is expected to update the SR. This is based on the Complaint Type and internal Service Level Agreements (SLAs). |
| Resolution Action Updated Date | Date when responding agency last updated the SR. |
| Resolution Description | Describes the last action taken on the SR by the responding agency. May describe next or future steps. |
| Location Type | Describes the type of location used in the address information |
| Incident Zip | Incident location zip code, provided by geo validation. |
| Incident Address | House number of incident address provided by submitter. |
| Street Name | Street name of incident address provided by the submitter |
| Cross Street 1 | First Cross street based on the geo validated incident location |
| Cross Street 2 | Second Cross Street based on the geo validated incident location |
| Intersection Street 1 | First intersecting street based on geo validated incident location |
| Intersection Street 2 | Second intersecting street based on geo validated incident location |
| Address Type | Type of incident location information available. |
| City | City of the incident location provided by geovalidation. |
| Landmark | If the incident location is identified as a Landmark the name of the landmark will display here |
| Facility Type | If available, this field describes the type of city facility associated to the SR |
| Community Board | Provided by geovalidation. |
| Borough | Provided by the submitter and confirmed by geovalidation. |
| X Coordinate (State Plane) | Geo validated, X coordinate of the incident location. |
| Y Coordinate (State Plane) | Geo validated, Y coordinate of the incident location. |
| Latitude | Geo based Lat of the incident location |
| Longitude | Geo based Long of the incident location |
| Location | Combination of the geo based lat & long of the incident location |
| Park Facility Name | If the incident location is a Parks Dept facility, the Name of the facility will appear here |
| Park Borough | The borough of incident if it is a Parks Dept facility |
| School Name | If the incident location is a Dept of Education school, the name of the school will appear in this field. If the incident is a Parks Dept facility its name will appear here. |
| School Number | If the incident location is a Dept of Education school, the Number of the school will appear in this field. This field is also used for Parks Dept Facilities. |
| School Region | If the incident location is a Dept of Education School, the school region number will be appear in this field. |
| School Code | If the incident location is a Dept of Education School, the school code number will be appear in this field. |
| School Phone Number | If the facility = Dept for the Aging or Parks Dept, the phone number will appear here. (note - Dept of Education facilities do not display phone number) |
| School Address | Address of facility of incident location, if the facility is associated with Dept of Education, Dept for the Aging or Parks Dept |
| School City | City of facilities incident location, if the facility is associated with Dept of Education, Dept for the Aging or Parks Dept |
| School State | State of facility incident location, if the facility is associated with Dept of Education, Dept for the Aging or Parks Dept |
| School Zip | Zip of facility incident location, if the facility is associated with Dept of Education, Dept for the Aging or Parks Dept |
| School Not Found | Y' in this field indicates the facility was not found |
| School or Citywide Complaint | If the incident is about a Dept of Education facility, this field will indicate if the complaint is about a particualr school or a citywide issue. |
| Vehicle Type | If the incident is a taxi, this field describes the type of TLC vehicle. |
| Taxi Company Borough | If the incident is identified as a taxi, this field will display the borough of the taxi company. |
| Taxi Pick Up Location | If the incident is identified as a taxi, this field displays the taxi pick up location |
| Bridge Highway Name | If the incident is identified as a Bridge/Highway, the name will be displayed here. |
| Bridge Highway Direction | If the incident is identified as a Bridge/Highway, the direction where the issue took place would be displayed here. |
| Road Ramp | If the incident location was Bridge/Highway this column differentiates if the issue was on the Road or the Ramp. |
| Bridge Highway Segment | Additional information on the section of the Bridge/Highway were the incident took place. |
| Garage Lot Name | Related to DOT Parking Meter SR, this field shows what garage lot the meter is located in |
| Ferry Direction | Used when the incident location is within a Ferry, this field indicates the direction of ferry |
| Ferry Terminal Name | Used when the incident location is Ferry, this field indicates the ferry terminal where the incident took place. |

1. This is a breakdown of every collision in NYC by location and injury. This data is collected because the NYC Council passed Local Law #11 in 2011. This data is manually run every month and reviewed by the TrafficStat Unit before being posted on the NYPD website. Each record represents a collision in NYC by city, borough, precinct and cross street. This data can be used by the public to see how dangerous/safe intersections are in NYC. The information is presented in pdf and excel format to allow the casual user to just view the information in the easy to read pdf format or use the excel files to do a more in-depth analysis.

The **NYPD Motor Vehicle Collisions** CSV file can be downloaded from the URL you see below.

**URL:** https://data.cityofnewyork.us/Public-Safety/NYPD-Motor-Vehicle-Collisions/h9gi-nx95

The **NYPD Motor Vehicle Collisions** CSV file has 29 fields as you can see below. These fields describe the accident scene the cars involved in accident and how many people were injured or killed.

**Column Name Type**

DATE Date & Time

TIME Plain Text

BOROUGH Plain Text

ZIP CODE Plain Text

LATITUDE Number

LONGITUDE Number

LOCATION Location

ON STREET NAME Plain Text

CROSS STREET NAME Plain Text

OFF STREET NAME Plain Text

NUMBER OF PERSONS INJURED Number

NUMBER OF PERSONS KILLED Number

NUMBER OF PEDESTRIANS INJURED Number

NUMBER OF PEDESTRIANS KILLED Number

NUMBER OF CYCLIST INJURED Number

NUMBER OF CYCLIST KILLED Number

NUMBER OF MOTORIST INJURED Number

NUMBER OF MOTORIST KILLED Number

CONTRIBUTING FACTOR VEHICLE 1 Plain Text

CONTRIBUTING FACTOR VEHICLE 2 Plain Text

CONTRIBUTING FACTOR VEHICLE 3 Plain Text

CONTRIBUTING FACTOR VEHICLE 4 Plain Text

CONTRIBUTING FACTOR VEHICLE 5 Plain Text

UNIQUE KEY Number

VEHICLE TYPE CODE 1 Plain Text

VEHICLE TYPE CODE 2 Plain Text

VEHICLE TYPE CODE 3 Plain Text

VEHICLE TYPE CODE 4 Plain Text

VEHICLE TYPE CODE 5 Plain Text

DATE,TIME,BOROUGH,ZIP CODE,LATITUDE,LONGITUDE,LOCATION,ON STREET NAME,CROSS STREET NAME,OFF STREET NAME,NUMBER OF PERSONS INJURED,NUMBER OF PERSONS KILLED,NUMBER OF PEDESTRIANS INJURED,NUMBER OF PEDESTRIANS KILLED,NUMBER OF CYCLIST INJURED,NUMBER OF CYCLIST KILLED,NUMBER OF MOTORIST INJURED,NUMBER OF MOTORIST KILLED,CONTRIBUTING FACTOR VEHICLE 1,CONTRIBUTING FACTOR VEHICLE 2,CONTRIBUTING FACTOR VEHICLE 3,CONTRIBUTING FACTOR VEHICLE 4,CONTRIBUTING FACTOR VEHICLE 5,UNIQUE KEY,VEHICLE TYPE CODE 1,VEHICLE TYPE CODE 2,VEHICLE TYPE CODE 3,VEHICLE TYPE CODE 4,VEHICLE TYPE CODE 5

**Dataset Queries for each dataset NYC Open Data**

The following are queries that I developed during my testing of the NYC Open Data datasets. The queries are broken down by dataset and what I was testing at the time. Much of these queries were to test **parallelism** and **nologging** and other options available to the Oracle 12c database queries. See my previous articles on the Oracle 12c database for further information, on Parallel processing and nologging options.

The Yellow Taxi Trip database is one of the largest datasets that you can download. The dataset is broken down by year and each CSV file is zipped and on average is on the order of 13GB when unzipped it averages 30GB. These datasets are on average 170 million rows in size and one of my projects is to create a commutive dataset of the years 2009 through 2017. This dataset will be around 1.1 billion rows

set echo on;

set timing on;

DESC YELLOW\_TRIP\_DATA\_1\_6\_2015

SQL> set timing on

SQL> DESC YELLOW\_TRIP\_DATA\_1\_6\_2015

Name Null Type

--------------------- ---- ------------

VENDORID1 VARCHAR2(40)

TPEP\_PICKUP\_DATETIME TIMESTAMP(6)

TPEP\_DROPOFF\_DATETIME TIMESTAMP(6)

PASSENGER\_COUNT1 VARCHAR2(40)

TRIP\_DISTANCE1 VARCHAR2(40)

PICKUP\_LONGITUDE1 VARCHAR2(40)

PICKUP\_LATITUDE1 VARCHAR2(40)

RATECODEID1 VARCHAR2(40)

STORE\_AND\_FWD\_FLAG1 VARCHAR2(40)

DROPOFF\_LONGITUDE1 VARCHAR2(40)

DROPOFF\_LATITUDE1 VARCHAR2(40)

PAYMENT\_TYPE1 VARCHAR2(40)

FARE\_AMOUNT1 VARCHAR2(40)

EXTRA1 VARCHAR2(40)

MTA\_TAX1 VARCHAR2(40)

TIP\_AMOUNT1 VARCHAR2(40)

TOLLS\_AMOUNT1 VARCHAR2(40)

TOTAL\_AMOUNT1 VARCHAR2(40)

VENDORID2 NUMBER

PASSENGER\_COUNT2 NUMBER

TRIP\_DISTANCE2 NUMBER

PICKUP\_LONGITUDE2 NUMBER

PICKUP\_LATITUDE2 NUMBER

RATECODEID2 NUMBER

STORE\_AND\_FWD\_FLAG2 NUMBER

DROPOFF\_LONGITUDE2 NUMBER

DROPOFF\_LATITUDE2 NUMBER

PAYMENT\_TYPE2 NUMBER

FARE\_AMOUNT2 NUMBER

EXTRA2 NUMBER

MTA\_TAX2 NUMBER

TIP\_AMOUNT2 NUMBER

TOLLS\_AMOUNT2 NUMBER

TOTAL\_AMOUNT2 NUMBER

Elapsed: 00:00:01.172

set echo on;

set timing on;

SET AUTOTRACE ON;

SELECT ALL\_TRIPS\_2015, TOTAL\_PASSENGERS\_2015, (TOTAL\_PASSENGERS\_2015/ALL\_TRIPS\_2015) AS AVG\_PASSENGER\_PER\_TRIP FROM

(SELECT SUM(PASSENGER\_COUNT1) AS TOTAL\_PASSENGERS\_2015 FROM YELLOW\_TRIP\_DATA\_1\_6\_2015) A,

(SELECT COUNT(\*) AS ALL\_TRIPS\_2015 FROM YELLOW\_TRIP\_DATA\_1\_6\_2015 )B;

SELECT ALL\_TRIPS\_2015, TOTAL\_PASSENGERS\_2015, (TOTAL\_PASSENGERS\_2015/ALL\_TRIPS\_2015) AS AVG\_PASSENGER\_PER\_TRIP FROM

(SELECT **/\*+ PARALLEL \*/** SUM(PASSENGER\_COUNT1) AS TOTAL\_PASSENGERS\_2015 FROM YELLOW\_TRIP\_DATA\_1\_6\_2015) A,

(SELECT **/\*+ PARALLEL \*/** COUNT(\*) AS ALL\_TRIPS\_2015 FROM YELLOW\_TRIP\_DATA\_1\_6\_2015 )B;

SELECT CNT\_PASSGR\_2\_ORMORE, (CNT\_PASSGR\_2\_ORMORE / ALL\_TRIPS\_2015) AS PECRNT\_TRIPS\_2ORMORE\_PASS FROM

( SELECT COUNT(\*) AS ALL\_TRIPS\_2015 FROM YELLOW\_TRIP\_DATA\_1\_6\_2015 ) A,

( SELECT COUNT(\*) AS CNT\_PASSGR\_2\_ORMORE FROM YELLOW\_TRIP\_DATA\_1\_6\_2015

WHERE PASSENGER\_COUNT1 > 1) B;

SELECT CNT\_PASSGR\_2\_ORMORE, (CNT\_PASSGR\_2\_ORMORE / ALL\_TRIPS\_2015) AS PECRNT\_TRIPS\_2ORMORE\_PASS FROM

( SELECT **/\*+ PARALLEL \*/** COUNT(\*) AS ALL\_TRIPS\_2015 FROM YELLOW\_TRIP\_DATA\_1\_6\_2015 ) A,

( SELECT **/\*+ PARALLEL \*/** COUNT(\*) AS CNT\_PASSGR\_2\_ORMORE FROM YELLOW\_TRIP\_DATA\_1\_6\_2015

WHERE PASSENGER\_COUNT1 > 1) B;

SELECT CNT\_PASSGR\_3\_ORMORE, (CNT\_PASSGR\_3\_ORMORE / ALL\_TRIPS\_2015) AS PECRNT\_TRIPS\_3ORMORE\_PASS FROM

( SELECT COUNT(\*) AS ALL\_TRIPS\_2015 FROM YELLOW\_TRIP\_DATA\_1\_6\_2015 ) A,

( SELECT COUNT(\*) AS CNT\_PASSGR\_3\_ORMORE FROM YELLOW\_TRIP\_DATA\_1\_6\_2015

WHERE PASSENGER\_COUNT1 > 2) B;

SELECT CNT\_PASSGR\_3\_ORMORE, (CNT\_PASSGR\_3\_ORMORE / ALL\_TRIPS\_2015) AS PECRNT\_TRIPS\_3ORMORE\_PASS FROM

( SELECT **/\*+ PARALLEL \*/** COUNT(\*) AS ALL\_TRIPS\_2015 FROM YELLOW\_TRIP\_DATA\_1\_6\_2015 ) A,

( SELECT **/\*+ PARALLEL \*/** COUNT(\*) AS CNT\_PASSGR\_3\_ORMORE FROM YELLOW\_TRIP\_DATA\_1\_6\_2015

WHERE PASSENGER\_COUNT1 > 2) B;

SELECT CNT\_PASSGR\_4\_ORMORE, (CNT\_PASSGR\_4\_ORMORE / ALL\_TRIPS\_2015) AS PECRNT\_TRIPS\_4ORMORE\_PASS FROM

( SELECT COUNT(\*) AS ALL\_TRIPS\_2015 FROM YELLOW\_TRIP\_DATA\_1\_6\_2015 ) A,

( SELECT COUNT(\*) AS CNT\_PASSGR\_4\_ORMORE FROM YELLOW\_TRIP\_DATA\_1\_6\_2015

WHERE PASSENGER\_COUNT1 > 3) B;

SELECT CNT\_PASSGR\_4\_ORMORE, (CNT\_PASSGR\_4\_ORMORE / ALL\_TRIPS\_2015) AS PECRNT\_TRIPS\_4ORMORE\_PASS FROM

( SELECT **/\*+ PARALLEL \*/** COUNT(\*) AS ALL\_TRIPS\_2015 FROM YELLOW\_TRIP\_DATA\_1\_6\_2015 ) A,

( SELECT **/\*+ PARALLEL \*/** COUNT(\*) AS CNT\_PASSGR\_4\_ORMORE FROM YELLOW\_TRIP\_DATA\_1\_6\_2015

WHERE PASSENGER\_COUNT1 > 3) B;

SELECT **/\*+ PARALLEL \*/** SUM(TRIP\_DISTANCE1) AS TOTAL\_MILES\_2015 FROM YELLOW\_TRIP\_DATA\_1\_6\_2015 ;

SELECT SUM(TRIP\_DISTANCE1) AS TOTAL\_MILES\_2015 FROM YELLOW\_TRIP\_DATA\_1\_6\_2015 ;

SELECT **/\*+ PARALLEL \*/** SUM(TRIP\_DISTANCE1) AS TOTAL\_MILES\_2015 FROM YELLOW\_TRIP\_DATA\_1\_6\_2015 ;

SELECT SUM(TRIP\_DISTANCE1) AS TOTAL\_MILES\_2015 FROM YELLOW\_TRIP\_DATA\_1\_6\_2015 ;

set echo on;

set timing on;

SET AUTOTRACE ON;

SELECT TRIPS\_LESS1MIL, (TRIPS\_GT1MIL / ALL\_TRIPS\_2015) AS PERCNT\_TRIPS\_LESS1 FROM

( SELECT **/\*+ PARALLEL \*/** COUNT(\*) AS ALL\_TRIPS\_2015 FROM YELLOW\_TRIP\_DATA\_1\_6\_2015 ) A,

( SELECT **/\*+ PARALLEL \*/** COUNT(\*) AS TRIPS\_GT1MIL FROM YELLOW\_TRIP\_DATA\_1\_6\_2015

WHERE TRIP\_DISTANCE1 > 1 ) B;

SELECT TRIPS\_GT5MIL, (TRIPS\_GT5MIL / ALL\_TRIPS\_2015) AS PERCNT\_TRIPS\_LESS1 FROM

( SELECT COUNT(\*) AS ALL\_TRIPS\_2015 FROM YELLOW\_TRIP\_DATA\_1\_6\_2015 ) A,

( SELECT COUNT(\*) AS TRIPS\_GT5MIL FROM YELLOW\_TRIP\_DATA\_1\_6\_2015

WHERE TRIP\_DISTANCE1 > 5 ) B;

SELECT TRIPS\_GT5MIL, (TRIPS\_GT5MIL / ALL\_TRIPS\_2015) AS PERCNT\_TRIPS\_LESS1 FROM

( SELECT **/\*+ PARALLEL \*/** COUNT(\*) AS ALL\_TRIPS\_2015 FROM YELLOW\_TRIP\_DATA\_1\_6\_2015 ) A,

( SELECT **/\*+ PARALLEL \*/** COUNT(\*) AS TRIPS\_GT5MIL FROM YELLOW\_TRIP\_DATA\_1\_6\_2015

WHERE TRIP\_DISTANCE1 > 5 ) B;

SELECT TRIPS\_GT10MIL, (TRIPS\_GT10MIL / ALL\_TRIPS\_2015) AS PERCNT\_TRIPS\_LESS1 FROM

( SELECT **/\*+ PARALLEL \*/** COUNT(\*) AS ALL\_TRIPS\_2015 FROM YELLOW\_TRIP\_DATA\_1\_6\_2015 ) A,

( SELECT **/\*+ PARALLEL \*/** COUNT(\*) AS TRIPS\_GT10MIL FROM YELLOW\_TRIP\_DATA\_1\_6\_2015

WHERE TRIP\_DISTANCE1 > 10 ) B;

SELECT TRIPS\_GT10MIL, (TRIPS\_GT10MIL / ALL\_TRIPS\_2015) AS PERCNT\_TRIPS\_LESS1 FROM

( SELECT COUNT(\*) AS ALL\_TRIPS\_2015 FROM YELLOW\_TRIP\_DATA\_1\_6\_2015 ) A,

( SELECT COUNT(\*) AS TRIPS\_GT10MIL FROM YELLOW\_TRIP\_DATA\_1\_6\_2015

WHERE TRIP\_DISTANCE1 > 10 ) B;

SELECT TRIPS\_LESS1MIL, (TRIPS\_LESS1MIL / ALL\_TRIPS\_2015) AS PERCNT\_TRIPS\_LESS1 FROM

( SELECT **/\*+ PARALLEL \*/** COUNT(\*) AS ALL\_TRIPS\_2015 FROM YELLOW\_TRIP\_DATA\_1\_6\_2015 ) A,

( SELECT **/\*+ PARALLEL \*/** COUNT(\*) AS TRIPS\_LESS1MIL FROM YELLOW\_TRIP\_DATA\_1\_6\_2015

WHERE TRIP\_DISTANCE1 < 1 ) B;

SELECT TRIPS\_LESS1MIL, (TRIPS\_LESS1MIL / ALL\_TRIPS\_2015) AS PERCNT\_TRIPS\_LESS1 FROM

( SELECT COUNT(\*) AS ALL\_TRIPS\_2015 FROM YELLOW\_TRIP\_DATA\_1\_6\_2015 ) A,

( SELECT COUNT(\*) AS TRIPS\_LESS1MIL FROM YELLOW\_TRIP\_DATA\_1\_6\_2015

WHERE TRIP\_DISTANCE1 < 1 ) B;

SELECT PASSENGER\_COUNT1, TRIP\_DISTANCE1 FROM YELLOW\_TRIP\_DATA\_1\_6\_2015;

SELECT AVG(PASSENGER\_COUNT1), MAX(PASSENGER\_COUNT1) FROM YELLOW\_TRIP\_DATA\_1\_6\_2015;

SELECT AVG(TRIP\_DISTANCE1), MAX(TRIP\_DISTANCE1) FROM YELLOW\_TRIP\_DATA\_1\_6\_2015;

SELECT **/\*+ PARALLEL \*/** AVG(TRIP\_DISTANCE1), MAX(TRIP\_DISTANCE1) FROM YELLOW\_TRIP\_DATA\_1\_6\_2015;

SELECT SUM(TRIP\_DISTANCE1) AS TOTAL\_MILES\_2015 FROM YELLOW\_TRIP\_DATA\_1\_6\_2015 ;

SELECT **/\*+ PARALLEL \*/**  AVG(PASSENGER\_COUNT1), MAX(PASSENGER\_COUNT1) FROM YELLOW\_TRIP\_DATA\_1\_6\_2015;

SELECT PASSENGER\_COUNT1, TRIP\_DISTANCE1 FROM YELLOW\_TRIP\_DATA\_1\_6\_2015;

SELECT COUNT(\*) FROM YELLOW\_TRIP\_DATA\_1\_6\_2015

WHERE PICKUP\_LONGITUDE2 = 0 AND

PICKUP\_LATITUDE2 = 0 AND

DROPOFF\_LONGITUDE2 = 0 AND

DROPOFF\_LATITUDE2 = 0;

**#HERE**

**The NYC Open Data Developer pages, see the URL below:**

URL: <https://developer.cityofnewyork.us/>

**How To Use NYC Open Data**

## To get Started With Open Data need

* Internet access
* An up-to-date web browser (such as [Google Chrome](https://www.google.com/chrome/browser/features.html?brand=CHBD&gclid=Cj0KEQiAkO7CBRDeqJ_ahuiPrtEBEiQAbYupJSsfBVpVBVEsZyYW5lw6K8XG86pLiVxLSVj_BYbP0V0aAhpC8P8HAQ&dclid=CMaJpfXfiNECFcEGNwodVW0OZg) or [Mozilla Firefox](https://www.mozilla.org/en-US/firefox/new/))
* A question you’re trying to answer about New York City

**Let’s give it a try!**  
The following steps will show you how to make a simple map from the dataset of 311 Service Requests. [311](http://www1.nyc.gov/311/index.page) is New York City’s non-emergency call center that allows citizens to make service requests, file complaints, and get additional information about the City.

1. Open the [“311 Service Requests 2010 to Present”](https://data.cityofnewyork.us/Social-Services/311-Service-Requests-from-2010-to-Present/erm2-nwe9) dataset in your web browser. This will take you to the dataset’s Primer page. This page contains important information about the data, such as the date it was last updated, its data dictionary, and a preview of the actual dataset.
2. At the top of the screen, click “View Data.” This takes you into the dataset, which contains over 14 million rows of service requests. This is a massive amount of data, so let’s condense this to a shorter time period.
3. Click on the dark blue “Filter” tab in the upper right corner. The Filter function allows you to narrow the search results into more manageable bits.
4. Click on the words “Unique Key.” In the drop down, select “Created Date.” Choose a date of your preference and press enter. The data should now be filtered on only service requests for that particular day.
5. To map this data, select the green “Visualize” tab at the upper-right hand of the screen. Within this tab select “Map.”
6. Required fields are indicated with a red asterisk. We recommend changing the plot style to a point map. Leave the location field as-is. Click “Apply.”
7. You have just created a filtered view of 311 Service Requests of a particular day, and visualized the location of these requests on a map.

**Your key resource: Data Dictionaries**  
Data can be complex. Some datasets contain millions of records and many unfamiliar terms. Be sure to review a dataset’s Primer page for additional information about a dataset, including its data dictionary. In the data dictionary, you’ll find definitions of terms and values.

**Guidelines for the division of large datasets**  
In general, if you experience difficulty manipulating and downloading larger datasets, you should restrict the number of records that appear using the “filter” function. In this panel, users may “Add a New Filter Condition” and select attribute values that are exact matches with the “is” condition, or a fall within a range of values with the “contains” condition. A narrower selection of results will require less computing power to view and manipulate data, and will create smaller data files that are quicker to download onto a user’s local device. For more information on filtering tabular datasets, including video tutorials, see the [“Filtering Datasets” topic](https://support.socrata.com/hc/en-us/sections/200469748-Filters) in the Socrata Knowledge Base.

## API Documentation

Want to win that hackathon or build the next great Open Data app? [Get started with Socrata APIs](https://dev.socrata.com/consumers/getting-started.html).

Socrata APIs provide [rich query functionality](https://dev.socrata.com/docs/queries/) through the “Socrata Query Language” (SoQL), which borrows heavily from Structured Query Language (SQL). Its paradigms should be familiar to developers who have worked with SQL and easy to learn for those who are new to it.

The [“endpoint”](https://dev.socrata.com/docs/endpoints.html) of a SODA API is a unique URL that represents an object or collection of objects. Every Socrata dataset, and even every individual data record, has its own endpoint. The endpoint is what you’ll point your HTTP client at to interact with data resources.

So you want to win that hackathon or build the next hot open data app? Follow this guide to get yourself started. Make sure you check out the more detailed resources under the “API Docs” section when it’s time to get in deep.

## Locating Open Data and APIs

[Socrata](http://socrata.com/) hosts over one hundred different data catalogs for governments, non-profits, and NGOs around the world, so finding an open data catalog to work with is easy:

* Check to see if your local government or state already has an open data site. Check your city or state website or even just Google “open data” and your government’s name. You’ll probably find something pretty quickly.
* Peruse the [Open Data Network](http://www.opendatanetwork.com/), our global catalog of open datasets. The same catalog listing is also available programmatically via the [Global Catalog API](http://labs.socrata.com/docs/search.html).
* Check to see if there’s a community group in your area with their own catalog hosted on [communities.socrata.com](https://communities.socrata.com/). Got a community group or event of your own and want free hosting for your data and APIs? [Sign up!](http://hackathon-in-a-box.org/open-data-apis/community-groups.html)

Once you’re on your local open data site, scroll down to the data catalog and use the search box and browse filters to find datasets that interest you - every dataset is accessible via the SODA API.

## Finding your API endpoint

Every [Socrata](http://socrata.com/) open dataset has a built-in SODA API. But how you find the API endpoint can vary a bit.

If you’re viewing a DataLens, there will be a prominent “API” button in the upper left of the page. Click that, and you’ll get details on the API endpoint and a link to API documentation.

If you’re on a Socrata dataset, identifiable by the colorful buttons at the upper right, don’t fret. Every Socrata dataset has a built-in open data API, so you’ll be just fine. Click on Export and then API and you’ll find the API endpoint under API Access Endpoint. Copy that and save it for later.

Read the detailed documentation on [API Endpoints](https://dev.socrata.com/docs/endpoints.html) for more info.

For this example, we’ll use this listing of [Alternative Fuel Locations](https://data.cityofchicago.org/developers/docs/f7f2-ggz5) in Chicago:

[try it](https://data.cityofchicago.org/resource/f7f2-ggz5.json?) [docs](https://dev.socrata.com/foundry/data.cityofchicago.org/f7f2-ggz5) copy [experiment](http://hurl.it/?method=GET&url=https%3A%2F%2Fdata.cityofchicago.org%2Fresource%2Ff7f2-ggz5.json&headers=%7B%22X-App-Token%22%3A%5B%22bjp8KrRvAPtuf809u1UXnI0Z8%22%5D%7D&args=%7B%7D) [json](https://dev.socrata.com/consumers/getting-started.html)

<https://data.cityofchicago.org/resource/f7f2-ggz5.json?>

## Building simple filters and queries

Filtering data via a SODA API is fairly straightforward. There are two primary mechanisms you can use to filter data: [Simple Filters](https://dev.socrata.com/docs/filtering.html) and [SoQL Queries](https://dev.socrata.com/docs/queries/)

### Simple Filters

Filtering data is very straightforward. SODA APIs are self-describing — the schema and contents of the dataset itself determines how you can query it. Any field within the data can be used as a filter, simply by appending it to the API endpoint as a GET parameter. For example, to query for only fuel locations that provide [Liquefied Petroleum Gas](https://data.cityofchicago.org/developers/docs/f7f2-ggz5), simply append ?fuel\_type\_code=LPG to the URL:

[try it](https://data.cityofchicago.org/resource/f7f2-ggz5.json?fuel_type_code=LPG) [docs](https://dev.socrata.com/foundry/data.cityofchicago.org/f7f2-ggz5) copy [experiment](http://hurl.it/?method=GET&url=https%3A%2F%2Fdata.cityofchicago.org%2Fresource%2Ff7f2-ggz5.json&headers=%7B%22X-App-Token%22%3A%5B%22bjp8KrRvAPtuf809u1UXnI0Z8%22%5D%7D&args=%7B%22fuel_type_code%22%3A%5B%22LPG%22%5D%7D) [json](https://dev.socrata.com/consumers/getting-started.html)

<https://data.cityofchicago.org/resource/f7f2-ggz5.json?fuel_type_code=LPG>

Additional filters can be added, and the filters will be ANDed together.

Read the detailed documentation on [Filtering Datasets](https://dev.socrata.com/docs/filtering.html) for more info.

### SoQL Queries

The “**So**crata **Q**uery **L**anguage” (SoQL) is a simple, SQL-like query language specifically designed for making it easy to work with data on the web. The language is both powerful and easy to learn, and everything works via GET parameters. For example, to search for fuel stations in downtown Chicago:

[try it](https://data.cityofchicago.org/resource/f7f2-ggz5.json?$where=within_box(location,%2041.885001,%20-87.645939,%2041.867011,%20-87.618516)) [docs](https://dev.socrata.com/foundry/data.cityofchicago.org/f7f2-ggz5) copy [experiment](http://hurl.it/?method=GET&url=https%3A%2F%2Fdata.cityofchicago.org%2Fresource%2Ff7f2-ggz5.json&headers=%7B%22X-App-Token%22%3A%5B%22bjp8KrRvAPtuf809u1UXnI0Z8%22%5D%7D&args=%7B%22%24where%22%3A%5B%22within_box(location%2C+41.885001%2C+-87.645939%2C+41.867011%2C+-87.618516)%22%5D%7D) [json](https://dev.socrata.com/consumers/getting-started.html)

[https://data.cityofchicago.org/resource/f7f2-ggz5.json?$where=within\_box(location, 41.885001, -87.645939, 41.867011, -87.618516)](https://data.cityofchicago.org/resource/f7f2-ggz5.json?$where=within_box(location,%2041.885001,%20-87.645939,%2041.867011,%20-87.618516))

Many different functions are available via SoQL. Read the detailed documentation on [SoQL Queries](https://dev.socrata.com/docs/queries/) for more info.

### Paging

For performance, SODA APIs are paged, and return a maximum of 50,000 records per page. So, to request subsequent pages, you’ll need to use the $limit and $offset parameters to request more data. The $limit parameter chooses how many records to return per page, and $offset tells the API on what record to start returning data.

So, to request page two, at 100 records per page, of our fuel locations API:

[try it](https://data.cityofchicago.org/resource/f7f2-ggz5.json?$limit=100&$offset=100) [docs](https://dev.socrata.com/foundry/data.cityofchicago.org/f7f2-ggz5) copy [experiment](http://hurl.it/?method=GET&url=https%3A%2F%2Fdata.cityofchicago.org%2Fresource%2Ff7f2-ggz5.json&headers=%7B%22X-App-Token%22%3A%5B%22bjp8KrRvAPtuf809u1UXnI0Z8%22%5D%7D&args=%7B%22%24limit%22%3A%5B%22100%22%5D%2C%22%24offset%22%3A%5B%22100%22%5D%7D) [json](https://dev.socrata.com/consumers/getting-started.html)

<https://data.cityofchicago.org/resource/f7f2-ggz5.json?$limit=100&$offset=100>

Read the detailed documentation on [Paging](https://dev.socrata.com/docs/paging.html) for more info.

**Tower Dell XPS 8920:**

* Core 7i quad code./ Ryzen Treadripper
* 64 GB dram.
* 2x HE 10TB mechanical drives.
* Installed is an Oracle 12c Enterprise database with an SGA of 4GB
* This database has a custom configuration.
* There are six PDB’s on the Oracle 12c R2 instance.

**Starting off with Big Data**

If you always wanted to get a head start on Big Data processing the Yellow Taxi Trip datasets can be a great resource either at rest or for streaming pipelines.

The NYC Open Data datasets can give you that capability as an intro to DOT traffic streaming data sources. This source will allow you to construct Spark pipelines for further processing and there is an application which would allow you to do just that too.

There are many NoSQL databases for these use cases and other solutions such as:Hadoop and its eco system of application. Apache Spark for the construction of data pipelines to process the data in motion

-- #################################################################

## Throttling and Application Tokens

Hold on a second! Before you go storming off to make the next great open data app, you should understand how SODA handles throttling. You can make a certain number of requests without an application token, but they come from a shared pool and you’re eventually going to get cut off.

If you want more requests, [register for an application token](https://dev.socrata.com/register) and your application will be granted up to 1000 requests per rolling hour period. If you need even more than that, special exceptions are made by request. Use the Help! tab on the right of this page to file a trouble ticket.

* [**Documentation**](https://socratadiscovery.docs.apiary.io/)
* [**Inspector**](https://socratadiscovery.docs.apiary.io/traffic)

### Discovery API

socrata • socratadiscovery

**Apiary Powered Documentation**

[Sign in](https://login.apiary.io/login?redirect=https%3A%2F%2Fsocratadiscovery.docs.apiary.io%2F) with Apiary account.

### Introduction

 [Asset Visibility](https://socratadiscovery.docs.apiary.io/introduction/asset-visibility)

 [Authentication](https://socratadiscovery.docs.apiary.io/introduction/authentication)

### Reference

 [Searching particular domains](https://socratadiscovery.docs.apiary.io/reference/0)

 [Searching particular categories/tags](https://socratadiscovery.docs.apiary.io/reference/0)

 [Searching particular asset types](https://socratadiscovery.docs.apiary.io/reference/0)

 [Searching domain-specific metadata](https://socratadiscovery.docs.apiary.io/reference/0)

 [Searching particular attributions](https://socratadiscovery.docs.apiary.io/reference/0)

 [Searching by license](https://socratadiscovery.docs.apiary.io/reference/0)

 [Text search](https://socratadiscovery.docs.apiary.io/reference/0)

 [Searching for derived views](https://socratadiscovery.docs.apiary.io/reference/0)

 [Searching for official or community-created views](https://socratadiscovery.docs.apiary.io/reference/0)

 [Boosting official views](https://socratadiscovery.docs.apiary.io/reference/0)

 [Searching for private or public data](https://socratadiscovery.docs.apiary.io/reference/0)

 [Searching for published or unpublished data](https://socratadiscovery.docs.apiary.io/reference/0)

 [Searching for approved or pending or rejected data](https://socratadiscovery.docs.apiary.io/reference/0)

 [Searching for hidden or not hidden data](https://socratadiscovery.docs.apiary.io/reference/0)

 [Searching for assets owned by a given user](https://socratadiscovery.docs.apiary.io/reference/0)

 [Searching for assets shared to you or the organization](https://socratadiscovery.docs.apiary.io/reference/0)

 [Sort order](https://socratadiscovery.docs.apiary.io/reference/0)

 [Pagination](https://socratadiscovery.docs.apiary.io/reference/0)

 [Deep scrolling over large sets of search results](https://socratadiscovery.docs.apiary.io/reference/0)

 [Complete Search API](https://socratadiscovery.docs.apiary.io/reference/0)

 [Search autocomplete](https://socratadiscovery.docs.apiary.io/reference/0)

 [Count by and list domains](https://socratadiscovery.docs.apiary.io/reference/0)

 [Count by and list tags](https://socratadiscovery.docs.apiary.io/reference/0)

 [Count by and list custom domain tags](https://socratadiscovery.docs.apiary.io/reference/0)

 [Count by and list categories](https://socratadiscovery.docs.apiary.io/reference/0)

 [Count by and list custom domain categories](https://socratadiscovery.docs.apiary.io/reference/0)

 [Count by and list custom domain facets](https://socratadiscovery.docs.apiary.io/reference/0)

# Discovery API

### Introduction

The Socrata data platform hosts tens of thousands of government datasets. Governments large and small publish data on crime, permits, finance, healthcare, research, performance, and more for citizens to use. While this large corpus of government data is already accessible via opendatanetwork.com, this API opens up the Socrata corpus of government data for automated searching, research, and exploration. Datasets can be found by keywords, high-level categorizations, tags, and much more. This API, then, is a **powerful way to access and explore all public metadata published on the Socrata platform.** All calls return a JSON object containing three fields:

* results An array of json objects, discussed below
* resultSetSize The total number of results that could be returned were they not paged
* timings Timing information regarding how long the request took to fulfill

The results objects contains 5 fields:

* resource A json object representing a dataset, visualization or other asset. Fields are described below.
* classification A json object describing the asset's classification by categories and tags. Fields are described below.
* metadata A json object containing metadata about the asset.
* permalink The permanent link of the asset.
* link The prettier, but non-permanent, link of the asset.

The resource object contains

* name - The title of the asset
* id - The unique identifier for the asset
* description - The description for the asset, if one has been provided
* attribution - The attribution of the asset, if one has been provided
* type - The type of the asset
* updatedAt - The timestamp at which the asset was last updated
* createdAt - The timestamp at which the asset was created
* page\_views - An object containing the number of views this asset has had over the last week and month and over all time
* columns\_name - An array of the names of columns in this asset if applicable
* columns\_description - An array of the column descriptions for this asset, if provided
* columns\_field\_name - An array of the field names of column, which serve as an identifier for the column.
* parent\_fxf - An array of dataset IDs that this asset is based on
* provenance - Either official or community
* download\_count - The number of times this asset has been downloaded

The classification object contains 5 fields:

* categories An array of categories that have been automatically assigned by stastically derived models
* tags An array of tags that have been automatically assigned by stastically derived models
* domain\_category The singular category given to the asset by the owning domain
* domain\_tags An array of tags given to the asset by the owning domain
* domain\_metadata An array of objects with the "key" and "value" of any custom metadata given to this asset by the owning domain

The production API endpoints for the public version of this API are at https://api.us.socrata.com/api/catalog/v1 for domains in North America and https://api.eu.socrata.com/api/catalog/v1 for all other domains. For example, to query for datasets categorized as Crime, you could use the following query:

[http://api.us.socrata.com/api/catalog/v1?categories=public safety](http://api.us.socrata.com/api/catalog/v1?categories=public%20safety)

* Please note that we assume all calls include a valid Socrata app token via the X-App-Token header. This is assumed and not documented in the API spec below. Please refer to our main [Developer site](https://dev.socrata.com/docs/app-tokens.html) for information on using an app token.\*

## Asset Visibility

There are four key factors which control whether or not a Socrata asset can be viewed anonymously by an unauthenticated user on the platform. An asset must meet criteria for all factors which apply to the domain itself (this varies domain-by-domain, as not all domains employ relevant features or modules which utilize these).

These factors are:

* Privacy status
* Publication status
* Approval status
* Whether asset is hidden or not

## Authentication

Authentication is not required to use this API for read-only access to Socrata’s corpus of anonymously-viewable (i.e. “published”, “public”, “approved”, and “not hidden” assets). However, if you wish to search for “private”, “unpublished”, “unapproved” or "hidden" data, you must authenticate yourself and ensure that you have adequate permissions to view the data in question.

To authenticate, you must:

* Use one of the methods discussed [here](https://dev.socrata.com/docs/authentication.html) and
* Provide the X-Socrata-Host host header with the domain that has granted you access to view its assets. For example X-Socrata-Host:data.ny.gov. We refer to this as the authenticating domain.

When properly authenticated, you will be able to search over:

* All data that is anonymously-viewable.
* Any data that you own or that has been shared to you.
* “Private”, “unpublished”, “unapproved”, and “hidden” assets from the authenticating domain if and only if the authenticating domain has granted you a [role](https://support.socrata.com/hc/en-us/articles/202950278-Understanding-user-roles) which allows you to view such assets.

### Reference

## Searching particular domains

Each asset is owned by a single domain. The domain and search\_context params limit the results down to those from a single domain. Without these, the search service returns results from the full corpus of customer domains.

#### [Domain Search API](https://socratadiscovery.docs.apiary.io/reference/0/searching-particular-domains/domain-search-api)

## Searching particular categories/tags

Each asset may have categories/tags assigned by Socrata or by the owning domain. The categories and tags parameters limit the results to those having the given category or tag. Without these, the search service returns results from the entire set of categories/tags. To search the categories/tags of a particular domain, you must also include the search\_context param. To search the categories/tags that were assigned by Socrata, exclude the search\_context param.

The categories and tags parameters may be repeated. The result set will be the union of all assets containing one or more of the specified tags.

#### [Category/Tag Search API](https://socratadiscovery.docs.apiary.io/reference/0/searching-particular-categoriestags/category/tag-search-api)

## Searching particular asset types

Each asset has a logical, such as a dataset or chart. The only parameter limits the results to a particular type (eg datasets, charts, maps). Without this param, the search service returns results from the entire set of types.

#### [Asset Type Search API](https://socratadiscovery.docs.apiary.io/reference/0/searching-particular-asset-types/asset-type-search-api)

## Searching domain-specific metadata

Each domain has the ability to add custom metadata to datasets beyond Socrata’s default metadata. This custom metadata is different for every domain, but within a domain, all assets may be labeled with the metadata. The custom metadata is a named set of key-value pairs. For example one domain might have a set named "Publication Metadata" and have keys "Publication Date" and "Publication Cycle", while another domain has a set named "Agency Ownership" having key "Department"). The caller may restrict the results to a particular custom metadata pair by specifying the param name as a combination of the set's name and the key's name and the param value as the key's value. To construct the param name join the set's name to the key's name with an underscore and replace all spaces with dashes. Some examples are given in the table below:

| **Set Name** | **Key Name** | **Key Value** | **Param** |
| --- | --- | --- | --- |
| Publication Metadata | Publication Cycle | Daily | ?q=Publication-Metadata\_Publication-Cycle=Daily |
| Agency Ownership | Department | Fisheries & Wildlife | ?q=Agency-Ownership\_Department=Fisheries %26 Wildlife |

#### [Custom Metadata Search API](https://socratadiscovery.docs.apiary.io/reference/0/searching-domain-specific-metadata/custom-metadata-search-api)

## Searching particular attributions

Assets can be attributed to various organizations. The caller can restrict the results to only the assets with the given attribution by specifying this parameter.

#### [Asset Attribution Search API](https://socratadiscovery.docs.apiary.io/reference/0/searching-particular-attributions/asset-attribution-search-api)

## Searching by license

Assets can be released under various licenses. The caller can restrict the results to only the assets with the given license by specifying this parameter.

#### [Asset Attribution Search API](https://socratadiscovery.docs.apiary.io/reference/0/searching-by-license/asset-attribution-search-api)

## Text search

Assets may be searched by any of the text found in the name, description, category, tags, column names, column fieldnames, column descriptions, and attribution. The q parameter takes arbitrary text and finds assets having some or all of the text. The optional min\_should\_match parameter may be used to explicitly specify the number or percent of words that must match. See [the Elasticsearch docs](https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-minimum-should-match.html) for the format of arguments to min\_should\_match. If min\_should\_match is not specified, the service's default is 3<60%, meaning that if there are 3 or fewer search terms specified, all of them must match; otherwise 60% of the search terms must be found in the fields specified above.

For example, if min\_should\_match is 3<60%:

* Searching for city dog park will require stemmed matches for all three words; thus, Western Cities Association Dog Parks will match, but New York City Parks will not.
* Searching for trees green spaces new york will require 60% of the words to match, which is 3 out of 5 words; thus, New York Tree Map, and New Green Spaces Initiative will both match.

#### [Text search API](https://socratadiscovery.docs.apiary.io/reference/0/text-search/text-search-api)

## Searching for derived views

The search service maintains any relationship present between a dataset asset and view-type assets that derive from it. For example charts derive from a parent dataset. The derived\_from param will limit the results to those that derive from a particular dataset.

#### [Derived assets search API](https://socratadiscovery.docs.apiary.io/reference/0/searching-for-derived-views/derived-assets-search-api)

## Searching for official or community-created views

While many assets on Socrata are owned by government data publishers and other staff, some visualizations, maps, filtered views, and more visible in a data catalog may have been created by a member of the community. These assets are usually denoted with a "Community" badge on the data catalog; this search service accounts for this information in the provenance field. Filtering by provenance=official will limit results to assets owned by roled users on the domain. Conversely, filtering by provenance=community will limit results to community-owned views.

#### [Official or community created assets search API](https://socratadiscovery.docs.apiary.io/reference/0/searching-for-official-or-community-created-views/official-or-community-created-assets-search-api)

## Boosting official views

It is possible to adjust the rankings of official views to promote them above community-created content. This leverages the weight function of [function score queries](https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-function-score-query.html#function-weight) in Elasticsearch. This weight acts as a multiplier for the relevance score of each official document. Thus, a number between 0 and 1 will demote official views, while any number greater than 1 will boost them.

#### [Official boost search API](https://socratadiscovery.docs.apiary.io/reference/0/boosting-official-views/official-boost-search-api)

## Searching for private or public data

The Privacy status of each asset is the first of four factors which control an asset’s visibility. A public=true param will limit results to those that are public. A public=false param will limit results to those that are private, noting that if you fail to authenticate or lack the permissions to view private data, no results will be returned.

#### [Private or public data search API](https://socratadiscovery.docs.apiary.io/reference/0/searching-for-private-or-public-data/private-or-public-data-search-api)

## Searching for published or unpublished data

The Publication status of each asset is the second of four factors which control an asset’s visibility. A published=true param will limit results to those that are published. A published=false param will limit results to those that are unpublished, noting that if you fail to authenticate or lack the permissions to view unpublished data, no results will be returned.

#### [Published or unpublished data search API](https://socratadiscovery.docs.apiary.io/reference/0/searching-for-published-or-unpublished-data/published-or-unpublished-data-search-api)

## Searching for approved or pending or rejected data

The Approval status of each asset is the third of four factors which control an asset’s visibility. Some Socrata sites employ an Approval module which routes assets through a series of steps before allowing them to be anonymously-viewable. A approval\_status=approved param will limit results to those that are approved. A approval\_status=pending param will limit results to those that are pending, noting that if you fail to authenticate or lack the permissions to view pending data, no results will be returned. A approval\_status=rejected param will limit results to those that are rejected, noting that if you fail to authenticate or lack the permissions to view rejected data, no results will be returned.

#### [Approved or pending or rejected data search API](https://socratadiscovery.docs.apiary.io/reference/0/searching-for-approved-or-pending-or-rejected-data/approved-or-pending-or-rejected-data-search-api)

## Searching for hidden or not hidden data

The Hidden status of each asset is the fourth of four factors which control an asset’s visibiliy. Some Socrata sites selectively and explicitly hide certain assets from their public catalog for different reasons; this parameter accounts for this setting.

A explicitly\_hidden=false param will limit results to those that are not hidden from the public catalog. A explicitly\_hidden=true param will limit results to those that are hidden from the public catalog, noting that if you fail to authenticate or lack the permissions to view hidden data, no results will be returned.

#### [Hidden or unhidden data search API](https://socratadiscovery.docs.apiary.io/reference/0/searching-for-hidden-or-not-hidden-data/hidden-or-unhidden-data-search-api)

## Searching for assets owned by a given user

Each asset has an owner and the for\_user param allows you specify a user id to find assets owned by that user. These are drawn from the set of assets that are already visible to you, so please note you will not likely be able to see everything owned by a user.

#### [Owner data search API](https://socratadiscovery.docs.apiary.io/reference/0/searching-for-assets-owned-by-a-given-user/owner-data-search-api)

## Searching for assets shared to you or the organization

Each asset may be shared to:

* individual users
* a team
* the domain's entire organization of users (anyone with a role on the domain)

The shared\_to param allows you to specify either a user id, a team id, or the string "organization" to find assets shared to that user or team or to the organization. Please note, that when specifying a user id, you are only allowed to view assets which are shared to you - you may not see which assets are shared to other users. When specifying team ids, you may only see assets which are shared to teams that you belong to (either as a member or an owner). If you attempt to view assets shared to a user other than yourself, or to a team that you don't belong to, the request will fail. You must authenticate in order to see any assets when using this param.

#### [Sharing data search API](https://socratadiscovery.docs.apiary.io/reference/0/searching-for-assets-shared-to-you-or-the-organization/sharing-data-search-api)

## Sort order

The results of all the above filters can be sorted by a number of fields. If not specified, the results are sorted by relevance. If a sort field is specified that the search service does not recognize, the query will fail. For all accepted sort values, either ASC (the default) or DESC can optionally be specified, eg. email DESC. Note that spaces must be URL-escaped with + or %20.

It is possible for search results to have missing values for some of these sort fields (such as domain\_category, for example). Any assets missing a value altogether for the field being sorted on will show up at the end of the results list.

**Accepted Values:**

* relevance (default)
* name
* owner
* dataset\_id
* datatype
* domain\_category
* createdAt
* updatedAt
* page\_views\_total
* page\_views\_last\_month
* page\_views\_last\_week

#### [Sort order API](https://socratadiscovery.docs.apiary.io/reference/0/sort-order/sort-order-api)

## Pagination

The search service allows pagination of results. By default, we will return at most 100 results starting from 0.

If the sum of the offset and limit parameters is greater than 10000, the server will respond with a 400. If your use-case involves scanning over a large set of results, you will want to use the scroll\_id parameter in conjunction with the limit parameter. For more detail, refer to the Deep scrolling section.

#### [Pagination API](https://socratadiscovery.docs.apiary.io/reference/0/pagination/pagination-api)

## Deep scrolling over large sets of search results

The search API is optimized for the prototypical use-case -- namely, providing some queries or filter conditions, and retrieving a relatively small number of search results. As a result, the search service does not support paging over a large set of search results. Specifically, if the sum of the offset and limit parameters is greater than 10000, the server will respond with a 400. This will happen regardless of the actual result set size. Larger result sets can be incrementally paged over via the scroll\_id parameter. This parameter takes a value corresponding to an asset ID, specifically, the ID of the last result in the previously fetched chunk of results. So for example, suppose you execute a query and find that it returns a large set of results (ie. more than 10000). You should execute the same query again, including a reasonable value for the limit parameter, being sure to include the scroll\_id parameter as well. Initially, you won't have a value for the scroll\_id parameter, so you will leave it blank. But with each subsequent request, you should pass the asset id corresponding to the last result from the previously fetched batch of results.

Note that sorting parameters are not honored when used in conjunction with deep scrolling via the scroll\_id parameter. If the order parameter is specified at the same time as the scroll\_id parameter, the server will respond with a 400.

#### [Deep scrolling API](https://socratadiscovery.docs.apiary.io/reference/0/deep-scrolling-over-large-sets-of-search-results/deep-scrolling-api)

## Complete Search API

The full search API is detailed here.

#### [Search API with all options](https://socratadiscovery.docs.apiary.io/reference/0/complete-search-api/search-api-with-all-options)

## Search autocomplete

To improve the catalog search experience, we added an autocomplete API, which returns a list of titles that match the search query. This API supports all the same filtering parameters as the top-level complete search API. Moreover, the response body has the same shape as the search API, containing resultSetSize, results, and timings fields at the root of the response object. The results field consists of an array of objects, with each object containing display\_title and title fields. The display\_title field contains markup to enable highlighting of the matching query string. The title field contains the raw title string.

Note that while this API mirrors the top-level search API, the behavior of the q parameter is slightly different in this case. Just as with the full search endpoint, it takes arbitrary text. However, the autocomplete search is restricted to the name (ie. asset title) field.

#### [Search autocomplete API](https://socratadiscovery.docs.apiary.io/reference/0/search-autocomplete/search-autocomplete-api)

## Count by and list domains

Returns each domain and the count of assets owned by that domain. The counts may be restricted according to any of the query parameters described in the Complete Search API, with the exception of the q text search param.

#### [Count by domain API](https://socratadiscovery.docs.apiary.io/reference/0/count-by-and-list-domains/count-by-domain-api)

## Count by and list tags

Returns each tag and the count of assets having that tag. The counts may be restricted according to any of the query parameters described in the Complete Search API.

#### [Count by tag API](https://socratadiscovery.docs.apiary.io/reference/0/count-by-and-list-tags/count-by-tag-api)

## Count by and list custom domain tags

Returns each domain-specific tag and the count of assets having that tag. The counts may be restricted according to any of the query parameters described in the Complete Search API.

#### [Count by domain tags API](https://socratadiscovery.docs.apiary.io/reference/0/count-by-and-list-custom-domain-tags/count-by-domain-tags-api)

## Count by and list categories

Returns each category and the count of assets having that category. The counts may be restricted according to any of the query parameters described in the Complete Search API.

#### [Count by category API](https://socratadiscovery.docs.apiary.io/reference/0/count-by-and-list-categories/count-by-category-api)

## Count by and list custom domain categories

Returns each domain-specific category and the count of assets having that category. The counts may be restricted according to any of the query parameters described in the Complete Search API.

#### [Count by domain categories API](https://socratadiscovery.docs.apiary.io/reference/0/count-by-and-list-custom-domain-categories/count-by-domain-categories-api)

## Count by and list custom domain facets

Returns the domain's custom facets (e.g. categories, tags, asset types, provenance and any custom metadata) and the count of assets having a value for that facet. Each facet also includes counts aggregated by the various facet values.

#### [Count by domain facets API](https://socratadiscovery.docs.apiary.io/reference/0/count-by-and-list-custom-domain-facets/count-by-domain-facets-api)

#### [Count by domain facets API](https://socratadiscovery.docs.apiary.io/reference/0/count-by-and-list-custom-domain-facets/count-by-domain-facets-api)

Switch between example and interactive console for customized API calls.

**Switch to Console**

### No action selected

You can try selecting ‘Domain Search API’ from the left column.

[Learn more about using the documentation.](https://help.apiary.io/tools/interactive-documentation/)

**DOT Data Feeds – Beta**

<https://developer.cityofnewyork.us/api/dot-data-feeds-beta>

**Description**

The Department of Transportation (DOT) provides numerous data feeds including Traffic Advisories, Street Construction Worksites, Bicycle Parking and Cycling maps, and information regarding the Staten Island Ferry, Alternate Side Parking Status, Parking Regulations, Citywide Low Bridges, New York City Truck Routes, and Street Network Changes.  DOT also provides real-time data from traffic cameras and traffic speed detectors.

Note that these data feeds may contain and/or utilize information that was originally compiled by DOT for governmental purposes; the information may have been modified subsequently by entity/entities other than DOT. DOT and the City of New York make no representation as to the accuracy or usefulness of the information provided by this application or the information's suitability for any purpose, and disclaim any liability for omissions or errors that may be contained therein. The public is advised to observe posted signage for compliance with applicable laws and regulations.

<https://github.com/CityOfNewYork/DOT-Data-Feeds>

# **Weekday Traffic Updates**

DOT issues traffic alerts, providing the locations of road construction and events where lane and street closings will affect the flow of traffic.

<http://www.nyc.gov/html/dot/html/motorist/weektraf.shtml>

## Getting Started

All communication with the API is done through HTTPS, and errors are communicated through HTTP response codes. Available response types include JSON, XML, and CSV, which are selectable by the "extension" (.json, etc.) on the API endpoint or through content-negotiation with HTTP Accepts headers.

This documentation also includes inline, runable examples. Click on any link that contains a gear symbol next to it to run that example live against the Alternative Fuel Locations API. If you just want to grab the API endpoint and go, you'll find it below.

[try it](https://data.cityofchicago.org/resource/jaif-n74j.json) [docs](https://dev.socrata.com/foundry/data.cityofchicago.org/jaif-n74j) copy [experiment](http://hurl.it/?method=GET&url=https%3A%2F%2Fdata.cityofchicago.org%2Fresource%2Fjaif-n74j.json&headers=%7B%22X-App-Token%22%3A%5B%22bjp8KrRvAPtuf809u1UXnI0Z8%22%5D%7D&args=%7B%7D) [json](https://dev.socrata.com/foundry/data.cityofchicago.org/jaif-n74j)

* [csv](https://dev.socrata.com/foundry/data.cityofchicago.org/jaif-n74j)
* [geojson](https://dev.socrata.com/foundry/data.cityofchicago.org/jaif-n74j)
* [json](https://dev.socrata.com/foundry/data.cityofchicago.org/jaif-n74j)

<https://data.cityofchicago.org/resource/jaif-n74j.json>

Learn more about:

* [Getting started with the SODA Consumer API](https://dev.socrata.com/consumers/getting-started.html)
* [Output formats and content negotiation](https://dev.socrata.com/docs/formats/index.html#2.1)
* [Response codes & error messages](https://dev.socrata.com/docs/response-codes.html)
* [How to stay up to date on API changes](https://dev.socrata.com/changelog/)

## App Tokens

All requests should include an [app token](https://dev.socrata.com/docs/app-tokens.html) that identifies your application, and each application should have its own unique app token. A limited number of requests can be made without an app token, but they are subject to much lower throttling limits than request that do include one. With an app token, your application is guaranteed access to it's own pool of requests. If you don't have an app token yet, click the button to the right to sign up for one.

[Sign up for an app token!](https://data.cityofchicago.org/profile/app_tokens)

Once you have an app token, you can include it with your request either by using the X-App-Token HTTP header, or by passing it via the $$app\_token parameter on your URL.

Learn more about [App Tokens & Throttling](https://dev.socrata.com/docs/app-tokens.html).

## Fields

Each column in [the dataset](https://data.cityofchicago.org/resource/jaif-n74j.json) is represented by a single field in its SODA API. Using [filters](https://dev.socrata.com/docs/filtering.html) and [SoQL queries](https://dev.socrata.com/docs/queries/), you can search for records, limit your results, and change the way the data is output. For example, you could filter this dataset by its id field using a query like the following:

For richer filtering, you can combine filters together by stacking parameters on your URL or by using [SoQL](https://dev.socrata.com/docs/queries/) queries. Learn more about each of the fields in this dataset by clicking the headers below, or read more about the SODA API using the navigation at the top of the page.

**Socrata Developer Page**

So, you want to win that hackathon or build the next hot open data app? Follow this guide to get yourself started. Make sure you check out the more detailed resources under the “API Docs” section when it’s time to get in deep.

**Locating Open Data and APIs**

[Socrata](http://socrata.com/) hosts over one hundred different data catalogs for governments, non-profits, and NGOs around the world, so finding an open data catalog to work with is easy:

Check to see if your local government or state already has an open data site. Check your city or state website or even just Google “open data” and your government’s name. You’ll probably find something quickly.

Peruse the [Open Data Network](http://www.opendatanetwork.com/), our global catalog of open datasets. The same catalog listing is also available programmatically via the [Global Catalog API](http://labs.socrata.com/docs/search.html).

Check to see if there’s a community group in your area with their own catalog hosted on [communities.socrata.com](https://communities.socrata.com/). Got a community group or event of your own and want free hosting for your data and APIs? [Sign up!](http://hackathon-in-a-box.org/open-data-apis/community-groups.html)

Once you’re on your local open data site, scroll down to the data catalog and use the search box and browse filters to find datasets that interest you - every dataset is accessible via the SODA API.

**Finding your API endpoint**

Every [Socrata](http://socrata.com/) open dataset has a built-in SODA API. But how you find the API endpoint can vary a bit.

If you’re viewing a **DataLens**, there will be a prominent “**API**” button in the upper left of the page. Click that, and you’ll get details on the **API** endpoint and a link to **API** documentation.

If you’re on a **Socrata** dataset, identifiable by the colorful buttons at the upper right, don’t fret. Every Socrata dataset has a built-in open data API, so you’ll be just fine. Click on Export and then API and you’ll find the API endpoint under API Access Endpoint. Copy that and save it for later.

Read the detailed documentation on [API Endpoints](https://dev.socrata.com/docs/endpoints.html) for more info.

For this example, we’ll use this listing of [Alternative Fuel Locations](https://data.cityofchicago.org/developers/docs/f7f2-ggz5) in Chicago:

[try it](https://data.cityofchicago.org/resource/f7f2-ggz5.json?) [docs](https://dev.socrata.com/foundry/data.cityofchicago.org/f7f2-ggz5) copy [experiment](http://hurl.it/?method=GET&url=https%3A%2F%2Fdata.cityofchicago.org%2Fresource%2Ff7f2-ggz5.json&headers=%7B%22X-App-Token%22%3A%5B%22bjp8KrRvAPtuf809u1UXnI0Z8%22%5D%7D&args=%7B%7D) [json](https://dev.socrata.com/consumers/getting-started.html)

[csv](https://dev.socrata.com/consumers/getting-started.html)

[json](https://dev.socrata.com/consumers/getting-started.html)

[rdf](https://dev.socrata.com/consumers/getting-started.html)

[xml](https://dev.socrata.com/consumers/getting-started.html)

<https://data.cityofchicago.org/resource/f7f2-ggz5.json?>

Building simple filters and queries

Filtering data via a SODA API is fairly straightforward. There are two primary mechanisms you can use to filter data: [Simple Filters](https://dev.socrata.com/docs/filtering.html) and [SoQL Queries](https://dev.socrata.com/docs/queries/)

**Simple Filters**

Filtering data is very straightforward. SODA APIs are self-describing — the schema and contents of the dataset itself determines how you can query it. Any field within the data can be used as a filter, simply by appending it to the API endpoint as a GET parameter. For example, to query for only fuel locations that provide [Liquefied Petroleum Gas](https://data.cityofchicago.org/developers/docs/f7f2-ggz5), simply append ?fuel\_type\_code=LPG to the URL:

[try it](https://data.cityofchicago.org/resource/f7f2-ggz5.json?fuel_type_code=LPG) [docs](https://dev.socrata.com/foundry/data.cityofchicago.org/f7f2-ggz5) copy [experiment](http://hurl.it/?method=GET&url=https%3A%2F%2Fdata.cityofchicago.org%2Fresource%2Ff7f2-ggz5.json&headers=%7B%22X-App-Token%22%3A%5B%22bjp8KrRvAPtuf809u1UXnI0Z8%22%5D%7D&args=%7B%22fuel_type_code%22%3A%5B%22LPG%22%5D%7D) [json](https://dev.socrata.com/consumers/getting-started.html)

[csv](https://dev.socrata.com/consumers/getting-started.html)

[json](https://dev.socrata.com/consumers/getting-started.html)

[rdf](https://dev.socrata.com/consumers/getting-started.html)

[xml](https://dev.socrata.com/consumers/getting-started.html)

<https://data.cityofchicago.org/resource/f7f2-ggz5.json?fuel_type_code=LPG>

Additional filters can be added, and the filters will be ANDed together.

Read the detailed documentation on [Filtering Datasets](https://dev.socrata.com/docs/filtering.html) for more info.

**SoQL Queries**

The “**So**crata **Q**uery **L**anguage” (SoQL) is a simple, SQL-like query language specifically designed for making it easy to work with data on the web. The language is both powerful and easy to learn, and everything works via GET parameters. For example, to search for fuel stations in downtown Chicago:

[try it](https://data.cityofchicago.org/resource/f7f2-ggz5.json?$where=within_box(location,%2041.885001,%20-87.645939,%2041.867011,%20-87.618516)) [docs](https://dev.socrata.com/foundry/data.cityofchicago.org/f7f2-ggz5) copy [experiment](http://hurl.it/?method=GET&url=https%3A%2F%2Fdata.cityofchicago.org%2Fresource%2Ff7f2-ggz5.json&headers=%7B%22X-App-Token%22%3A%5B%22bjp8KrRvAPtuf809u1UXnI0Z8%22%5D%7D&args=%7B%22%24where%22%3A%5B%22within_box(location%2C+41.885001%2C+-87.645939%2C+41.867011%2C+-87.618516)%22%5D%7D) [json](https://dev.socrata.com/consumers/getting-started.html)

[csv](https://dev.socrata.com/consumers/getting-started.html)

[json](https://dev.socrata.com/consumers/getting-started.html)

[rdf](https://dev.socrata.com/consumers/getting-started.html)

[xml](https://dev.socrata.com/consumers/getting-started.html)

[https://data.cityofchicago.org/resource/f7f2-ggz5.json?$where=within\_box(location, 41.885001, -87.645939, 41.867011, -87.618516)](https://data.cityofchicago.org/resource/f7f2-ggz5.json?$where=within_box(location,%2041.885001,%20-87.645939,%2041.867011,%20-87.618516))

Many different functions are available via SoQL. Read the detailed documentation on [SoQL Queries](https://dev.socrata.com/docs/queries/) for more info.

Paging

For performance, SODA APIs are paged, and return a maximum of 50,000 records per page. So, to request subsequent pages, you’ll need to use the $limit and $offset parameters to request more data. The $limit parameter chooses how many records to return per page, and $offset tells the API on what record to start returning data.

So, to request page two, at 100 records per page, of our fuel locations API:

[try it](https://data.cityofchicago.org/resource/f7f2-ggz5.json?$limit=100&$offset=100) [docs](https://dev.socrata.com/foundry/data.cityofchicago.org/f7f2-ggz5) copy [experiment](http://hurl.it/?method=GET&url=https%3A%2F%2Fdata.cityofchicago.org%2Fresource%2Ff7f2-ggz5.json&headers=%7B%22X-App-Token%22%3A%5B%22bjp8KrRvAPtuf809u1UXnI0Z8%22%5D%7D&args=%7B%22%24limit%22%3A%5B%22100%22%5D%2C%22%24offset%22%3A%5B%22100%22%5D%7D) [json](https://dev.socrata.com/consumers/getting-started.html)

[csv](https://dev.socrata.com/consumers/getting-started.html)

[json](https://dev.socrata.com/consumers/getting-started.html)

[rdf](https://dev.socrata.com/consumers/getting-started.html)

[xml](https://dev.socrata.com/consumers/getting-started.html)

<https://data.cityofchicago.org/resource/f7f2-ggz5.json?$limit=100&$offset=100>

Read the detailed documentation on [Paging](https://dev.socrata.com/docs/paging.html) for more info.

Throttling and Application Tokens

Hold on a second! Before you go storming off to make the next great open data app, you should understand how SODA handles throttling. You can make a certain number of requests without an application token, but they come from a shared pool and you’re eventually going to get cut off.

If you want more requests, [register for an application token](https://dev.socrata.com/register) and your application will be granted up to 1000 requests per rolling hour period. If you need even more than that, special exceptions are made by request. Use the Help! tab on the right of this page to file a trouble ticket.

**Code Snippets**

The following are grab-and-go code code samples you can use with popular programming languages and data science tools.

* [jQuery](https://dev.socrata.com/foundry/data.cityofchicago.org/jaif-n74j#jquery-js)
* [Python Pandas](https://dev.socrata.com/foundry/data.cityofchicago.org/jaif-n74j#pandas-py)
* [PowerShell](https://dev.socrata.com/foundry/data.cityofchicago.org/jaif-n74j#powershell-ps1)
* [RSocrata](https://dev.socrata.com/foundry/data.cityofchicago.org/jaif-n74j#rsocrata-r)
* [SAS](https://dev.socrata.com/foundry/data.cityofchicago.org/jaif-n74j#sas-sas)
* [soda-ruby](https://dev.socrata.com/foundry/data.cityofchicago.org/jaif-n74j#soda-ruby-rb)
* [SODA.NET](https://dev.socrata.com/foundry/data.cityofchicago.org/jaif-n74j#soda-net-cs)
* [Stata](https://dev.socrata.com/foundry/data.cityofchicago.org/jaif-n74j#stata-stata)

jQuery makes it super simple to fetch and parse JSON from an API endpoint.

copy

$.ajax({

url: "https://data.cityofchicago.org/resource/jaif-n74j.json",

type: "GET",

data: {

"$limit" : 5000,

"$$app\_token" : "YOURAPPTOKENHERE"

}

}).done(**function**(data) {

alert("Retrieved " + data.length + " records from the dataset!");

console.log(data);

});

The following resources might also be helpful:

* [Using a jQueryUI date slider to build a SODA Query](https://dev.socrata.com/blog/2014/12/24/jquery-date-slider.html)
* [Google Maps Mashup](https://dev.socrata.com/blog/2014/05/31/google-maps.html)
* [chrismetcalf/shady-eats on GitHub](https://github.com/chrismetcalf/shady-eats)

Python package using Pandas to easily work with JSON data

copy

#!/usr/bin/env python

# make sure to install these packages before running:

# pip install pandas

# pip install sodapy

**import** pandas **as** pd

**from** sodapy **import** Socrata

# Unauthenticated client only works with public data sets. Note 'None'

# in place of application token, and no username or password:

client = Socrata("data.cityofchicago.org", **None**)

# Example authenticated client (needed for non-public datasets):

# client = Socrata(data.cityofchicago.org,

# MyAppToken,

# userame="user@example.com",

# password="AFakePassword")

# First 2000 results, returned as JSON from API / converted to Python list of

# dictionaries by sodapy.

results = client.get("jaif-n74j", limit=2000)

# Convert to pandas DataFrame

results\_df = pd.DataFrame.from\_records(result\_list)

The following resources might also be helpful:

* [soda-py on GitHub](https://github.com/xmunoz/sodapy)
* [Upsert via soda-py](https://github.com/xmunoz/sodapy)

[PowerShell](https://github.com/PowerShell/PowerShell) to extract data from SODA

copy

$url = "https://data.cityofchicago.org/resource/jaif-n74j"

$apptoken = "YOURAPPTOKENHERE"

# Set header to accept JSON

$headers = **New**-Object "System.Collections.Generic.Dictionary[[String],[String]]"

$headers.Add("Accept","application/json")

$headers.Add("X-App-Token",$apptoken)

$results = Invoke-RestMethod -Uri $url -Method get -Headers $headers

The following resources might also be helpful:

* [PowerShell and SODA](https://dev.socrata.com/blog/2016/08/24/powershell-and-soda.html)
* [Socrata PowerShell on GitHub](https://github.com/joeywas/socrata-powershell)

The City of Chicago and community maintains a great [RSocrata package on Github](https://github.com/chicago/rsocrata).

copy

## **Install** the **required** **package** **with**:

## **install**.packages("RSocrata")

**library**("RSocrata")

df <- **read**.socrata(

"https://data.cityofchicago.org/resource/jaif-n74j.json",

app\_token = "YOURAPPTOKENHERE",

email = "user@example.com",

**password** = "fakepassword"

)

The following resources might also be helpful:

* [RSocrata on GitHub](https://github.com/Chicago/RSocrata)
* [Forecasting with RSocrata](https://dev.socrata.com/blog/2015/06/17/forecasting_with_rsocrata.html)

SAS is a tried and true application suite for data analysis and visualization. The following snippet brings Socrata data into a SAS.

copy

filename datain url 'http://data.cityofchicago.org/resource/jaif-n74j.csv?$limit=5000&$$app\_token=YOURAPPTOKENHERE';

proc import datafile=datain out=dataout dbms=csv replace;

getnames=yes;

run;

The following resources might also be helpful:

* [Analyzing Open Data with SAS](https://dev.socrata.com/blog/2015/11/10/analyzing-open-data-with-sas.html)

The [soda-ruby](https://github.com/socrata/soda-ruby) gem is a simple wrapper around the SODA APIs that makes usage with Ruby more natural.

copy

#!/usr/bin/env ruby

**require** 'soda/client'

client = SODA::Client.new({:domain => "data.cityofchicago.org", :app\_token => "YOURAPPTOKENHERE"})

results = client.get("jaif-n74j", :$limit => 5000)

puts "Got #{results.count} results. Dumping first results:"

results.first.each **do** |k, v|

puts "#{key}: #{value}"

**end**

The following resources might also be helpful:

* [soda-ruby on GitHub](https://github.com/socrata/soda-ruby)
* [Upsert via soda-ruby](https://socrata.github.io/soda-ruby/examples/upsert.html)

[SODA.NET](https://github.com/CityofSantaMonica/SODA.NET) is a Socrata Open Data API client library for .NET

copy

**using** System;

**using** System.Linq;

// Install the package from Nuget first:

// PM> Install-Package CSM.SodaDotNet

**using** SODA;

**var** client = **new** SodaClient("https://data.cityofchicago.org", "YOURAPPTOKENHERE");

// Get a reference to the resource itself

// The result (a Resouce object) is a generic type

// The type parameter represents the underlying rows of the resource

// and can be any JSON-serializable class

**var** dataset = client.GetResource("jaif-n74j");

// Resource objects read their own data

**var** rows = dataset.GetRows(limit: 5000);

Console.WriteLine("Got {0} results. Dumping first results:", rows.Count());

**foreach** (**var** keyValue **in** rows.First())

{

Console.WriteLine(keyValue);

}

The following resources might also be helpful:

* [SODA.NET on GitHub](https://github.com/CityofSantaMonica/SODA.NET)
* [SODA.NET on Nuget](https://www.nuget.org/packages/CSM.SodaDotNet)

Copy and paste the following to import this dataset into Stata

copy

clear

. import delimited "https://data.cityofchicago.org/resource/jaif-n74j.csv?%24limit=5000&%24%24app\_token=YOURAPPTOKENHERE