raj-vanguard2

```
%pyspark
import requests
```

READY

History(One time) load of the data

READY

- History load of the data should be done to distributed file system like s3 or HDFS. In this case that would be s3.
- we could use either downloaded filr from csv and upload it from local into s3 or make api calls using pagination to provided api url *https://data.cityofnewyork.us/resource/erm2-nwe9.json (https://data.cityofnewyork.us/resource/erm2-nwe9.json)* using *limit* and *offset* parameters
- once it is loaded, converted to json or parquet and compressed, it can be exposed to end users as a hive table or athena table.
- ETL processing can be done using GLUE ETL,spark,athena, scheduled python script etc. In this case I am going to use **pyspark**

Lets set base urls here and incremental urls here for api calls

READY

```
%pyspark
# url='https://data.cityofnewyork.us/resource/erm2-nwe9.json?&limit=50&limit=50&limit=50&offset=150'
base_url='https://data.cityofnewyork.us/resource/erm2-nwe9.json?'


#get top of the midnight hour to get incremental data for the day
from datetime import datetime,timedelta
offset_timestamp= datetime.isoformat(datetime.utcnow().replace(minute=0, hour=0, second=0, microsecond=0) -timedelta(days= 1))


incremental_url='https://data.cityofnewyork.us/resource/erm2-nwe9.json?$where=created_date > \"{0}\"'.format(offset_timestamp)
```

READY

raj-vanguard2

Metrics can be calculated using API provided by socrata,For example

READY

```
%pyspark
import pandas as pd
from pandas import DataFrame
def get_data(url):
    data = requests.get(url)
    return  DataFrame.from_dict(data.json())

count_query='$select=count(*)'
counts=get_data(base_url+count_query)
print( 'Total count of service requests: {}'.format(counts))
```

READY

```
Fail to execute line 1: import pandas as pd
Traceback (most recent call last):
  File "/tmp/zeppelin_pyspark-5199844230390393072.py", line 375, in <module>
    exec(code, _zcUserQueryNameSpace)
  File "<stdin>", line 1, in <module>
ImportError: No module named pandas
```

```
%pyspark
# □ How many complaint types are there?
query='$query=select distinct complaint_type'
res=get_data(base_url+query)

print( 'Total count of service requests: {}'.format(res))
```

READY

```
Total count of service requests:                     complaint_type
0    ../../../../../../../../../../...
1    ..././..././..././..././..././......
2    .../.../...//.../.../...//.../.../...//...
3    .../...././//.../...././//.../...././//...
4    .../.\.../.\.../.\.../.\.../.\......
..                                     ...
435  {!xmlparser v='&lt;!DOCTYPE...
436         X-Ray Machine/Equipment
437                       yw97y0gi2s
```

```
438                    ZSYSTEST
439                    ZTESTINT

[440 rows x 1 columns]
```

raj-vanguard2

we could use api provided queries such as below to do the case study queries but I will do it in spark since we have to productionize it inhouse

READY

```
%pyspark
# □ What is the average number of service requests by day of week?
query='$select=date_extract_dow(created_date) as day_of_week,count(*)&select=date_extract_dow(created_date) as day_of_week,count(*)&selec
# query='$select=created_date,avg(annual_salary)&select=job_titles,avg(annual_salary)&select=job_titles,avg(annual_salary)&group=job_titl
res=get_data(base_url+query)
print( 'Total count of service requests: {}'.format(res))
# □ How many service requests relating to noise were there in July 2016?
```

READY

Lets use spark to calculate metrics now!

READY

```
%pyspark
df = spark.read.csv('s3://bucket-name/raj/311_Service_Requests_from_2010_to_Present.csv.gz',header=True)
```

```
['Unique Key', 'Created Date', 'Closed Date', 'Agency', 'Agency Name', 'Complaint Type', 'Descriptor', 'Location Type', 'Incident Zip', 'I
ncident Address', 'Street Name', 'Cross Street 1', 'Cross Street 2', 'Intersection Street 1', 'Intersection Street 2', 'Address Type', 'Ci
ty', 'Landmark', 'Facility Type', 'Status', 'Due Date', 'Resolution Description', 'Resolution Action Updated Date', 'Community Board', 'BB
L', 'Borough', 'X Coordinate (State Plane)', 'Y Coordinate (State Plane)', 'Open Data Channel Type', 'Park Facility Name', 'Park Borough',
'Vehicle Type', 'Taxi Company Borough', 'Taxi Pick Up Location', 'Bridge Highway Name', 'Bridge Highway Direction', 'Road Ramp', 'Bridge H
ighway Segment', 'Latitude', 'Longitude', 'Location']
DataFrame[Unique Key: string, Created Date: string, Closed Date: string, Agency: string, Agency Name: string, Complaint Type: string, Desc
riptor: string, Location Type: string, Incident Zip: string, Incident Address: string, Street Name: string, Cross Street 1: string, Cross
Street 2: string, Intersection Street 1: string, Intersection Street 2: string, Address Type: string, City: string, Landmark: string, Faci
lity Type: string, Status: string, Due Date: string, Resolution Description: string, Resolution Action Updated Date: string, Community Boa
```

```
rd: string, BBL: string, Borough: string, X Coordinate (State Plane): string, Y Coordinate (State Plane): string, Open Data Channel Type: string, Park Facility Name: string, Park Borough: string, Vehicle Type: string, Taxi Company Borough: string, Taxi Pick Up Location: string, Bridge Highway Name: string, Bridge Highway Direction: string, Road Ramp: string, Bridge Highway Segment: string, Latitude: string, Longitude: string, Location: string]
```

raj-vanguard2

```
%pyspark
newDf=df
for col in df.columns:
    newDf = newDf.withColumnRenamed(col,col.replace(" ", "_").replace("(","_").replace(")","_"))
print(newDf.columns)
newDf.write.parquet('s3://bucket-name/raj/testny_pq/')
```

READY

```
['Unique_Key', 'Created_Date', 'Closed_Date', 'Agency', 'Agency_Name', 'Complaint_Type', 'Descriptor', 'Location_Type', 'Incident_Zip', 'Incident_Address', 'Street_Name', 'Cross_Street_1', 'Cross_Street_2', 'Intersection_Street_1', 'Intersection_Street_2', 'Address_Type', 'City', 'Landmark', 'Facility_Type', 'Status', 'Due_Date', 'Resolution_Description', 'Resolution_Action_Updated_Date', 'Community_Board', 'BBL', 'Borough', 'X_Coordinate__State_Plane_', 'Y_Coordinate__State_Plane_', 'Open_Data_Channel_Type', 'Park_Facility_Name', 'Park_Borough', 'Vehicle_Type', 'Taxi_Company_Borough', 'Taxi_Pick_Up_Location', 'Bridge_Highway_Name', 'Bridge_Highway_Direction', 'Road_Ramp', 'Bridge_Highway_Segment', 'Latitude', 'Longitude', 'Location']
```

```
%pyspark
```

READY

convert it to parquet for faster columnar reads
repartition it for spark to read it faster reads

READY

```
%pyspark
pq_df=spark.read.parquet('s3://bucketname/raj/testny_pq/').repartition(30).write.parquet('s3://bucketname/raj/testny_pq_split/')
```

READY

```
%pyspark
df_pq2= spark.read.parquet('s3://bucketname/raj/testny_pq_split/')
schema=df_pq2.schema
print(schema)
```

READY

StructType(List(StructField(Unique_Key,StringType,true),StructField(Created_Date,StringType,true),StructField(Closed_Date,StringType,true),StructField(Agency,StringType,true),StructField(Agency_Name,StringType,true),StructField(Complaint_Type,StringType,true),StructField(Descriptor,StringType,true),StructField(Location_Type,StringType,true),StructField(Incident_Zip,StringType,true),StructField(Incident_Address,StringType,true),StructField(Street_Name,StringType,true),StructField(Cross_Street_1,StringType,true),StructField(Cross_Street_2,StringType,true),StructField(Intersection_Street_1,StringType,true),StructField(Intersection_Street_2,StringType,true),StructField(Address_Type,StringType,true),StructField(City,StringType,true),StructField(Landmark,StringType,true),StructField(Facility_Type,StringType,true),StructField(Status,StringType,true),StructField(Due_Date,StringType,true),StructField(Resolution_Description,StringType,true),StructField(Resolution_Action_Updated_Date,StringType,true),StructField(Community_Board,StringType,true),StructField(BBL,StringType,true),StructField(Borough,StringType,true),StructField(X_Coordinate__State_Plane_,StringType,true),StructField(Y_Coordinate__State_Plane_,StringType,true),StructField(Open_Data_Channel_Type,StringType,true),StructField(Park_Facility_Name,StringType,true),StructField(Park_Borough,StringType,true),StructField(Vehicle_Type,StringType,true),StructField(Taxi_Company_Borough,StringType,true),StructField(Taxi_Pick_Up_Location,StringType,true),StructField(Bridge_Highway_Name,StringType,true),StructField(Bridge_Highway_Direction,StringType,true),StructField(Road_Ramp,StringType,true),StructField(Bridge_Highway_Segment,StringType,true),StructField(Latitude,StringType,true),StructField(Longitude,StringType,true),StructField(Location,StringType,true)))

raj-vanguard2

case study queries in spark

READY

READY

```
%pyspark

df_pq2= spark.read.parquet('s3://bucketname/raj/testny_pq_split/').createOrReplaceTempView('service_tickets')
# How many service requests are there?

df2=spark.sql("select count(*) from service_tickets")
df2.show(2,False)
```

```
+--------+
|count(1)|
+--------+
|22744396|
+--------+
```

READY

```
%pyspark

#How many complaint types are there?
df2 = spark.sql("SELECT distinct Complaint_Type from service_tickets ")
df2.show(5, False)
```

```
df2 = spark.sql("SELECT count( distinct Complaint_Type) from service_tickets ")
df2.show(10, False)
```

raj-vanguard2

```
+------------------------------+
|Complaint_Type                |
+------------------------------+
|Traffic Signal Condition      |
|Cranes and Derricks           |
|SAFETY                        |
|..\..\..\..\..\..\..\..\...|
|ELECTRIC                      |
+------------------------------+
only showing top 5 rows

+------------------------------+
|count(DISTINCT Complaint_Type)|
+------------------------------+
|440                           |
+------------------------------+
```

```
%pyspark                                                                                                                    READY

# □ What is the average number of service requests by day of week?
df2=spark.sql(" select  dayofweek( TO_DATE(CAST(UNIX_TIMESTAMP(dt, 'MM/dd/yyyy') AS TIMESTAMP))) AS day_of_week , avg(cnt) from ( select
    as cnt   from service_tickets group by dt limit 10) as k group by day_of_week  ")
df2.show(10,False)
```

```
+-----------+--------+
|day_of_week|avg(cnt)|
+-----------+--------+
|2          |6671.0  |
|6          |6235.0  |
|5          |6535.5  |
|4          |6611.0  |
|3          |8025.0  |
|1          |4432.0  |
+-----------+--------+
```

raj-vanguard2

```
%pyspark

# □ How many service requests relating to noise were there in July 2016?
df4=spark.sql("select  count(*), complaint_type from service_tickets where month( TO_DATE(CAST(UNIX_TIMESTAMP(Created_Date, 'MM/dd/yyyy')
    AS TIMESTAMP))) = 7 2016 and (lower(complaint_type) like '%noise%') group by complaint_type ")
df4.show(20,False)
```

READY

```
+--------+------------------------+
|count(1)|complaint_type          |
+--------+------------------------+
|72      |Noise - Helicopter      |
|90      |Noise - House of Worship|
|21136   |Noise - Residential     |
|3056    |Noise - Commercial      |
|601     |Noise - Park            |
|3853    |Noise                   |
|10      |Collection Truck Noise  |
|2662    |Noise - Vehicle         |
|8435    |Noise - Street/Sidewalk |
+--------+------------------------+
```

Finally save with additional columns in parquet for users to calculate metrics.

READY

```
%pyspark
df_final=spark.sql("select  *,TO_DATE(CAST(UNIX_TIMESTAMP(Created_Date, 'MM/dd/yyyy') AS TIMESTAMP)) as created_date_formatted, dayofweek
    TO_DATE(CAST(UNIX_TIMESTAMP(Created_Date, 'MM/dd/yyyy') AS TIMESTAMP))) as month, year(TO_DATE(CAST(UNIX_TIMESTAMP(Created_Date, 'MM/
df_final.write.parquet(s3://-----)
```

READY

How to handle Incremental load

READY

-Here we would daily batch load in to seperate day parititon in s3
-using url filter condition on created date we will receive incremetnal data based on provided timestampm conditional expression
-write that data in parquet for columnar fast reading

raj-vanguard2

READY

```
%pyspark
from collections import OrderedDict
from pyspark.sql import SparkSession, Row
import requests
import json
data=requests.get(incremental_url)
data_json= data.json()
# print(data_json)

df2=spark.createDataFrame(data_json)

df2.write.parquet('s3://------/current_day')
```

```
+------+--------------------+----------+-------+------+--------------------+--------------+--------------------+--------------------+----
----------+--------------+-----------------+---------------+------------+---------------------+---------------------+----------+---------
---------+--------------------+--------------------+------------------+----------------------+-----------+------------------+-----------
-----------------+----------------------+------+-----------+----------+----------------------+----------------------+-------------+--
----------+--------+
|agency|         agency_name|       bbl|borough|  city|         closed_date|community_board|      complaint_type|        created_date|cros
s_street_1|cross_street_2|       descriptor|incident_address|incident_zip|intersection_street_1|intersection_street_2|  landmark|
latitude|            location|       location_type|         longitude|open_data_channel_type|park_borough|park_facility_name|resolution_ac
tion_updated_date|resolution_description|status|street_name|unique_key|x_coordinate_state_plane|y_coordinate_state_plane|facility_type|add
ress_type|due_date|
+------+--------------------+----------+-------+------+--------------------+--------------+--------------------+--------------------+----
----------+--------------+-----------------+---------------+------------+---------------------+---------------------+----------+---------
---------+--------------------+--------------------+------------------+----------------------+-----------+------------------+-----------
-----------------+----------------------+------+-----------+----------+----------------------+----------------------+-------------+--
----------+--------+
|  NYPD|New York City Pol...|4017530099| QUEENS|CORONA|2020-05-02T00:15:...|      03 QUEENS|Non-Emergency Pol...|2020-05-02T00:00:...|
35 AVENUE|     36 AVENUE|Social Distancing|35-17 109 STREET|       11368|            35 AVENUE|            36 AVENUE|109 STREET|40.7551373
72555836|human address ->    |Residential Build   |-73.85905141657365|                PHONE|      QUEENS|       Unspecified|          202
```

other transfomration

READY

transformations that can be performed
– cleaning junk rows/columns
– standardizing timestamps/timezones if required and etc

```
%pyspark
```

READY

raj-vanguard2