Week 6: File ingestion and schema validation

Name: Bao Khanh Nguyen

Batch Code: LISUM04

Submission Data: 10/31/2021

Submitted to: Data Glacier

Introduction

Validation of the schema and the ingestion of files in data pipelining is an important component of data engineering; We need to know how to read data in the most efficient way possible to maximize resource utilization. This project will be used to demonstrate different approaches of reading data and determining the best way to do so. After that, try validating input data using a YAML file and, if the file is correct, writing it to a.gz file.

Dataset

The dataset that I used is “Parking_Violations_Issued_-_Fiscal_Year_2016” Dataset. The below features are the features of this dataset. The size of the dataset is 2.15 GB.

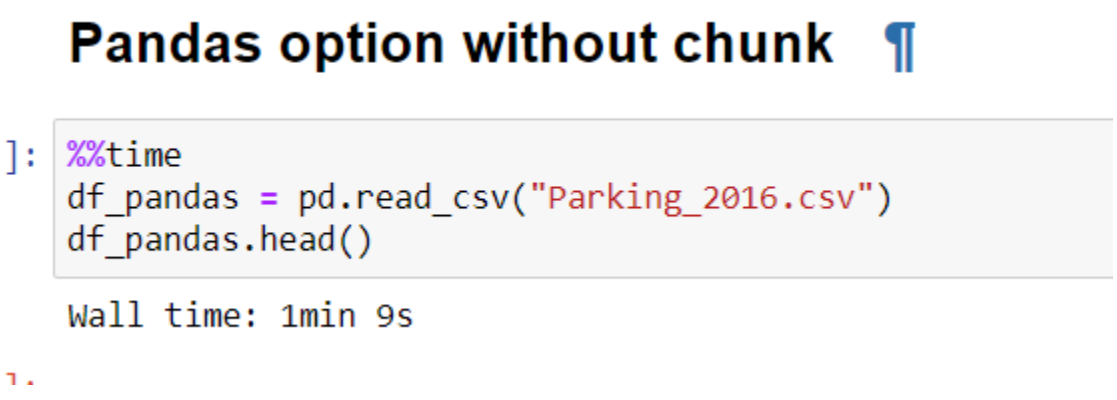
• Summons Number: Number
• Plate ID: Plain Text
• Registration State: Plain Text
• Plate Type: Plain Text
• Issue Date: Date & Time
• Violation Code: Number
• Vehicle Body Type: Plain Text
• Vehicle Make: Plain Text
• Issuing Agency: Plain Text
• Street Code1: Number
• Street Code2: Number
• Street Code3: Number
• Vehicle Expiration Date: Number
• Violation Location: Plain Text
• Violation Precinct: Number
• Issuer Precinct: Number
• Issuer Code: Number
• Issuer Command: Plain Text
• Issuer Squad: Plain Text
• Violation Time: Plain Text
• Time First Observed: Plain Text
• Violation County: Plain Text
• Violation In Front Of Or Opposite: Plain Text
• House Number: Plain Text
• Street Name: Plain Text
• Intersecting Street: Plain Text
• Date First Observed: Number
• Law Section: Number
• Sub Division: Plain Text
• Violation Legal Code: Plain Text

• Days Parking In Effect: Plain Text
• From Hours In Effect: Plain Text
• To Hours In Effect: Plain Text
• Vehicle Color: Plain Text
• Unregistered Vehicle?: Plain Text
• Vehicle Year: Number
• Meter Number: Plain Text
• Feet From Curb: Number
• Violation Post Code: Plain Text
• Violation Description: Plain Text
• No Standing or Stopping Violation: Plain Text
• Hydrant Violation: Plain Text
• Double Parking Violation: Plain Text
• Latitude: Number
• Longitude: Number
• Community Board: Number
• Community Council: Number
• Census Tract: Number
• BIN: Number
• BBL: Number
• NTA: Plain Text

Solution to read this dataset

Here, I have tried to put different solutions (libraries) to the test, and the results are shown in the sections below with Wall time is the time for each method execution:

Pandas:



Pandas with Chunks:

Pandas option with Chunk extension ¶

```
%%time
chunks = pd.read_csv("Parking_2016.csv", chunksize=100000)
df_pandas_chunks = pd.concat(chunks)
df_pandas_chunks.head()
```

```
Wall time: 1min 3s
```

Dask:

Dask solution

```
%%time
df_dask = dd.read_csv("Parking_2016.csv", dtype={'House Number': 'object',
       'Intersecting Street': 'object',
       'Issuer Squad': 'object',
       'Time First Observed': 'object',
       'Unregistered Vehicle?': 'float64',
       'Violation Description': 'object',
       'Violation Legal Code': 'object',
       'Violation Post Code': 'object'})
#We have to adding dtypes based on error generated
#from dask extension or have to increase sample size but I choose the first option with adding dtypes
df_dask.head()
```

```
Wall time: 1.32 s
```

Pyspark:

Pyspark Solution

```
import os
#I getting error with Java gateway process exited before sending the driver its port number, and found this web to solve this pr
#https://github.com/jupyter/notebook/issues/743
os.environ["JAVA_HOME"] = "C:/Program Files/Java/jdk1.8.0_311"
```

```
%%time

spark = SparkSession.builder.appName("Exploratory Analysis").getOrCreate()
df_spark = spark.read.format("csv").option("header", "true").option("inferSchema", "true").load("Parking_2016.csv")
df_spark.show(5)
```

...

```
--+---------------+------
only showing top 5 rows

Wall time: 20.4 s
```

CSV Disk Reader:

CSV Dict Reader Solution

```
%%time

df_csv = csv.DictReader(open("Parking_2016.csv"))
i=0
for row in df_csv:
        print(row)
        i += 1
        if i == 5:
            break
```

…

```
ensus Tract': '', 'BIN': '', 'BBL': '', 'NTA': ''}
Wall time: 999 µs
```

Datatable Fread:

Database. Fread Solution

```
%%time
df_dt = dt.fread("Parking_2016.csv")
df_dt.head()
```

```
Wall time: 6.95 s
```

Method	Pandas	Pandas with Chunk	Dask	Pyspark	CSV Disk Reader	Datatable Fread
Time Execution	1 min 9s	1 min 3s	1.32s	20.4s	999 x 10^{-6} seconds o r 99 micros econds	6.95s

In conclusion, based on the executed times, totally “csv.DictReader” is the best one , but “dask” is also another good choice in this scenario to read data in Data frame mode. However, Dask still have some situation with dytpes, which requires we have to force dtypes for each variable, so that I will use pandas because of its stable.

YAML File

Firstly, we have to create config file to store information about the dataset:

```yaml
input:
  format: csv
  name: Parking_2016
  delimiter: ","
  columns:
    - Summons Number
    - Plate ID
    - Registration State
    - Plate Type
    - Issue Date
    - Violation Code
    - Vehicle Body Type
    - Vehicle Make
    - Issuing Agency
    - Street Code1
    - Street Code2
    - Street Code3
    - Vehicle Expiration Date
    - Violation Location
    - Violation Precinct
    - Issuer Precinct
    - Issuer Code
    - Issuer Command
    - Issuer Squad
    - Violation Time
    - Time First Observed
    - Violation County
    - Violation In Front Of Or Opposite
    - House Number
    - Street Name
    - Intersecting Street
    - Date First Observed
    - Law Section
    - Sub Division
    - Violation Legal Code
    - Days Parking In Effect
```

```
  - From Hours In Effect
  - To Hours In Effect
  - Vehicle Color
  - Unregistered Vehicle?
  - Vehicle Year
  - Meter Number
  - Feet From Curb
  - Violation Post Code
  - Violation Description
  - No Standing or Stopping Violation
  - Hydrant Violation
  - Double Parking Violation
  - Latitude
  - Longitude
  - Community Board
  - Community Council
  - Census Tract
  - BIN
  - BBL
  - NTA
output:
 format: gz
 name: trusted_Parking_2016
 delimiter: "|"
 path: C:\Users\nguye\Desktop\Data Science Stuff\Projects\Python\File ingestion and schema validation\
```

All of information about input and output data, and columns also be included in this file

We retrieve data and check all validations, such as column counts and names, in the following code, and if everything checks out, we may write the file in.gz format as an output:

```
: def validate_data(raw_df):
    trusted_columns = list(map(lambda x: x.lower(),  cfg.input.columns))
    raw_columns = list(map(lambda x: x.lower(),  raw_df.columns))

    trusted_columns = [x.strip(' ') for x in trusted_columns]
    raw_columns = [x.strip(' ') for x in raw_columns]

    while(True):
        if len(raw_columns)!=len(trusted_columns):
            print(f'Count of columns are invalid! It should be {len(trusted_columns)}, but it is {len(raw_columns)}')
            return
        if raw_columns.sort()!=trusted_columns.sort():
            print('Columns are invalid!')
            return
        if raw_columns.sort()!=trusted_columns.sort():
            print(f'Columns are invalid!')
            print(f'Columns in Uploaded Dataset: {list(set(raw_columns).difference(trusted_columns))} VS. Columns in Config File:
            return


        output_file = cfg.output.name+"."+cfg.output.format
        output_path = cfg.output.path+output_file
        df.to_csv(output_path, header=None, index=None, sep=cfg.output.delimiter, compression='infer')
        print(f'File is uploaded successfully and written to: {output_file}')
        return
```

```
: validate_data(df)
```

```
File is uploaded successfully and written to: trusted_Parking_2016.gz
```

Information about this file also be can be found:

```
df = read_data_summary(cfg)
```

```
The size of the file is: 2151937808 Bytes
It has: 51 Columns and 10626899 Rows
```