Data Ingestion/Validation Report

By Aletia Trepte

For Data Glacier Internship

08/17/2022

Purpose & Introduction

This is the week 6 project for the Data Glacier Virtual Internship, Data Ingestion, which explores methods of reading large data files of size 2GB+, performs basic validation techniques on the data columns with YAML file, then creates a text file in gzip format, and data summary.

Data was found in Kaggle datasets, HERE and downloaded to my local Windows 11 computer. The Jupyter Notebook was run using Visual Studio Code and used Python packages that included Pandas, Dask, YAML, tracemalloc, shutil, GZIP, and, CSV. A block of the Glacier Data practice code was used in the val_dat_col() function for printing out file validation messages. The version control used was GIT, along with repositories in GitLab and GitHub. GitLab was the clear choice because of the 10GB of free data storage space.

General Data Summary

A summary of the Parking_Violations_2015.csv file was downloaded from the Kaggle website.

RangeIndex: 11809233 entries, 0 to 11809232
Data columns (total 51 columns):

#	Column	Dtype
0	Summons Number	int64
1	Plate ID	object
2	Registration State	object
3	Plate Type	object
4	Issue Date	object
5	Violation Code	int64
6	Vehicle Body Type	object
7	Vehicle Make	object
8	Issuing Agency	object
9	Street Code1	int64
10	Street Code2	int64
11	Street Code3	int64
12	Vehicle Expiration Date	object
13	Violation Location	float64
14	Violation Precinct	int64
15	Issuer Precinct	int64
16	Issuer Code	int64
17	Issuer Command	object
18	Issuer Squad	object
19	Violation Time	object
20	Time First Observed	object
21	Violation County	object
22	Violation In Front Of Or Opposite	object
23	House Number	object
24	Street Name	object
25	Intersecting Street	object
26	Date First Observed	object
27	Law Section	float64
28	Sub Division	object
29	Violation Legal Code	object
30	Days Parking In Effect	object
31	From Hours In Effect	object
32	To Hours In Effect	object
33	Vehicle Color	object
34	Unregistered Vehicle?	float64
35	Vehicle Year	float64
36	Meter Number	object
37	Feet From Curb	float64
38	Violation Post Code	object
39	Violation Description	object
40	No Standing or Stopping Violation	object

41	Hydrant Violation	object
42	Double Parking Violation	object
43	Latitude	float64
44	Longitude	float64
45	Community Board	float64
46	Community Council	float64
47	Census Tract	float64
48	BIN	float64
49	BBL	float64
50	NTA	float64

dtypes: float64(13), int64(8), object(30)

memory usage: 4.5+ GB
Datatype Warnings for columns 18, 29,38, 40, 41, 42, have mixed types.

Process

Loading the CSV file:

Pandas read csv and load pandas dataframe:

```
import pandas as pd
dfd = pd.read_csv('Parking_Violations_2015.csv', delimiter=',')
dfd
```

time to load: 5 minutes 35.4 seconds
9 memory blocks: 4705241.6 KiB

Dask read csv and load dask dataframe:

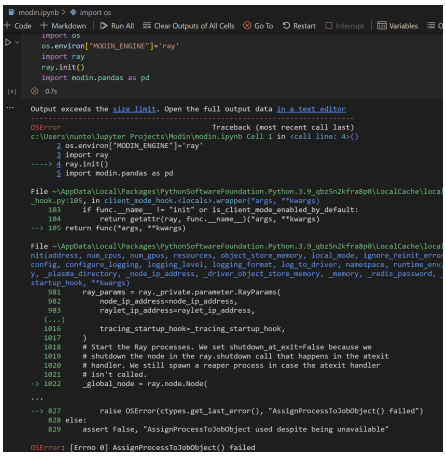
```
import dask.dataframe as dd
Import tracemalloc

tracemalloc.start()
dfd = dd.read_csv('Parking_Violations_2015.csv', delimiter=',')
dfd
snapshot=tracemalloc.take_snapshot()
tracemalloc.stop()
```

time to load: 0.9 seconds
300 memory blocks: 18.2 KiB

Ray(Modin)

I was unable to scale my Pandas workflow using Ray or Modin due to ongoing ray core compatibility issues on Windows. HERE is documentation regarding the ongoing issues and pictures of the error output.



The next process was done using Dask.

Creating YAML file:

Column validation file created with YAML, where 8 columns were dropped bringing the column count from 51 to 43. Column white spaces eliminated and text set to lower case.

Dropped Columns:

```
dfd=dfd.drop(["BIN", "BBL", "NTA"], axis=1)
dfd=dfd.drop(["Latitude", "Longitude", "Community Board", "Community Council ", "Census Tract"], axis=1)
```

```
%%writefile file.yaml
file_type: csv
dataset_name: Parking_Violations
file_name: Parking_Violations_2016
inbound_deliminater: ','
outbound_deliminater: '|'
skip_leading_rows: 1
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…etc.
```

Open YAML file:

```
import yaml
with open('file.yaml', 'r') as f:
    file = yaml.safe_load(f)
```

Validating Columns:

Column data validation function created. A part of the Data Glacier practice code was used, specifically the printing of data validation messages. However, regular expressions were not used for cleaning up the column names as in the DG practice code.

```
def val_data_col():
    # clean up df columns #
    dfd.columns=dfd.columns.str.replace('[?]', '')
    dfd.columns=dfd.columns.str.strip()
    dfd.columns=dfd.columns.str.replace('[ ]', '_')
    dfd.columns=dfd.columns.str.lower()
    # compare yaml columns with df columns #
    expected_columns = list(file['columns'])
    if len(dfd.columns) == len(expected_columns) and list(dfd.columns) == expected_columns:
        print('column name and column length validation passed')
        mismatched_columns_file = list(set(dfd.columns).difference(expected_columns))
        print("Following File columns are not in the YAML file",mismatched_columns_file)
        missing_YAML_file = list(set(expected_columns).difference(dfd.columns))
        print("Following YAML columns are not in the file uploaded",missing_YAML_file)
        logging.info(f'df columns: {dfd.columns}')
        logging.info(f'expected columns: {expected_columns}')
        return 1
    else:
        print('column name and column length validation failed')
    return 0
```

Calling function:

```
val_data_col()
column name and column length validation passed
The following File columns are not in the YAML file []
Following YAML columns are not in the file uploaded []
```

Write in gz format

To write the file in pipe-separated text file (|) in gz format, I used Pandas. Up until now, I used Dask and tried moving the text to a dask bag. Dask data frames are “lazy” so trying to write my dask data frame to a text file was difficult, even with dask bags, so I decided to run everything in pandas even though it was slow. Packages used: gzip, shutil,

```
dfd.to_csv(‘validated.txt’, sep=’|’)
import gzip
import shutil
with open('validated.txt', 'rb') as infile:
    with gzip.open('validated.txt.gz', 'wb') as outfile:
        shutil.copyfileobj(infile, outfile)
```

Validated.txt.gz file Summary:

Total lines in validation.txt 11809234
Total columns in validation.txt 43
Validated.txt 2,791,245 KB
Validated.txt.gz 667,857 KB

Final thoughts

Some of the takeaways for me after doing this project:

- Pandas can be slow for large datasets
- Dask works well for data validation and data ingestion for larger datasets
- Dask breaks up the data into partitions
- Dask df are ‘lazy’
- Tracemalloc to track memory usage
- GitLab offers more data storage than GitHub
- YAML files can be used to validate data files
- Data ingestion is the process of importing data
    - Batch data ingestion, in which data is collected and transferred in batches at regular intervals.
    - Streaming data ingestion, in which data is collected in real-time (or nearly) and loaded into the target location almost immediately.

Resources

Data Glacier Practice Code - validation function message print out part.
Dask & Pandas Documentation
Stack Overflow