Data Glacier

Your Deep Learning Partner

# File Ingestion and Schema Validation

Virtual Internship

10-Feb-2023

# Contents

# Introduction

- A data pipeline is the series of steps that allow data from one system to move to and become useful in another system, particularly analytics, data science, or AI and machine learning systems.
- At a high level, a data pipeline works by pulling data from the source, applying rules for transformation and processing, then pushing it to its destination

- File ingestion and schema validation: In this report different ways of reading large data files (pandas, dask, chunksize, dictread, datatables, pyspark) have been explored and the most optimal way is suggested.
- The data file is then converted to YAML file and further compressed to gzip.
- Two different ways of creating yaml file has been explored using dataframes directly and second after coverting to array.

# Dataset

- The dataset for parking violation ticket issued in New York in 2017 has been used.
- Data source:

https://www.kaggle.com/datasets/new-york-city/nyc-parking-tickets?select=Parking_Violations_Issued_-_Fiscal_Year_2017.csv

| Total number of observations | 10803028 |
|---|---|
| Total number of files | 1 |
| Total number of features | 43 |
| Base format of the file | .csv |
| Size of the data | 2.09 GB |

# Different ways to read data

Different methods as listed below, of reading the big data file has been tested and time to read the data is analysed to obtain the fastest method:

1) Pandas

## Time to read using pandas (1min 21s)

```
%%time
NYC_data = pd.read_csv("Parking_Violations_Issued_-_Fiscal_Year_2017.csv")
NYC_data.head()
```

```
CPU times: user 27.8 s, sys: 15.4 s, total: 43.2 s
Wall time: 1min 21s
```

2) Pandas Chunksize

## Time to read using pandas chunks (46s)

```
%%time
data_chunks = pd.read_csv("Parking_Violations_Issued_-_Fiscal_Year_2017.csv", chunksize=100000)
pandas_chunks = pd.concat(data_chunks)
pandas_chunks.head()
```

```
CPU times: user 29.9 s, sys: 8.71 s, total: 38.6 s
Wall time: 46.5 s
```

3) DictReader

## Time to read using dictreader(3.45ms)

```
%%time
data_dict = csv.DictReader(open('Parking_Violations_Issued_-_Fiscal_Year_2017.csv'))
i=0
for row in data_dict:
    print(row)
    i+=1
    if i is 5:
        break
```

```
CPU times: user 788 µs, sys: 1.16 ms, total: 1.95 ms
Wall time: 3.45 ms
```

# Different ways to read data

4) Datatable

## Time to read using datatable(6.5s)

```
%%time
data_dt = dt.fread("Parking_Violations_Issued_-_Fiscal_Year_2017.csv")
data_dt.head()
```

```
CPU times: user 16.3 s, sys: 2.71 s, total: 19 s
Wall time: 6.5 s
```

5) Pyspark

## Time to read using pyspark (32.8s)

```
%%time
spark = SparkSession.builder.appName("EDA.com").getOrCreate()
df = spark.read.format("csv").option("header", "true").option("inferSchema", "true").load("Parking_Violations_Issued_-_Fiscal_Year_2017.cs
df.show(5)
```

```
CPU times: user 36.2 ms, sys: 78.9 ms, total: 115 ms
Wall time: 32.8 s
```

6) Dask

## Time to read using DASK(2s)

```
%%time
data_dask = dd.read_csv("Parking_Violations_Issued_-_Fiscal_Year_2017.csv", dtype={'House Number': 'object',
       'Time First Observed': 'object'})
data_dask.head()
```

```
CPU times: user 882 ms, sys: 382 ms, total: 1.26 s
Wall time: 2.07 s
```

# Most Optimal way

- Maximum time is taken by Pandas.
- Among several ways Dictreader takes the least time (3.45ms) followed by Dask (2s).
- However, Dask maintains the pandas dataframe structure hence Dask can be preferred as compared to other methods for reading large data files.

# Using YAML

- The parameterization has been done to create a general format for reading data files (Parameterization.ipynb).

- Firstly, a utility file with some important functions is created as:

```
def read_config_file(filepath):
    with open(filepath, 'r') as stream:
        try:
            return yaml.safe_load(stream)
        except yaml.YAMLError as exc:
            logging.error(exc)


def replacer(string, char):
    pattern = char + '{2,}'
    string = re.sub(pattern, char, string)
    return string

def col_header_val(df,table_config):
    '''
    replace whitespaces in the column
    and standardized column names
    '''

    raw_columns = list(map(lambda x: x.lower(),  df.columns))
    yml_columns = list(map(lambda x: x.lower(),  table_config['columns']))

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


    expected_col=yml_columns


    if len(raw_columns) == len(expected_col) and list(expected_col)  == list( raw_columns):
        print("column name and column length validation passed")
        return 1
    else:
        print("column name and column length validation failed")
        mismatched_columns_file = list(set(raw_columns).difference(expected_col))
        print("Following File columns are not in the YAML file",mismatched_columns_file)
        missing_YAML_file = list(set(expected_col).difference(raw_columns))
        print("Following YAML columns are not in the file uploaded",missing_YAML_file)
        logging.info(f'df columns: {raw_columns}')
        logging.info(f'expected columns: {expected_col}')
        return 0
```

# Using YAML

Next, a yml file is created as:

```
%%writefile file.yaml
file_type: csv
dataset_name: testfile
file_name: Parking_Violations_Issued_-_Fiscal_Year_2017
table_name: edsurv
inbound_delimiter: ","
outbound_delimiter: "|"
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
- 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
```

# GZIP

- The data has been zipped using gzip.

```python
import gzip
import shutil
with open('file.yaml', 'rb') as f_in:
    with gzip.open('file.yaml.gz', 'wb') as f_out:
        shutil.copyfileobj(f_in, f_out)
```

```python
NYC_data.shape
```

```
(10803028, 43)
```

# Thank You

Data Glacier

Your Deep Learning Partner