file_ingestion

April 11, 2021

Author: Omar Safwat **Date:** 2021-04-11 **Batch:** LISP01

1 Week 6: File Ingestion and Schema Validation

This notebook demonstrates the pipeline of automated data ingestion in typical day-to-day data science tasks. The code uses the **"ddos_balanced"** data set; a randomly selected file, that exceeds 5 GB in size for demonstration purposes. Data can be downloaded from this link.

2 File Ingestion

2.1 Summarizing the file

[2]:
```
# Libraries of Reading in files
import pandas as pd
import dask.dataframe as dd
from time import time # To monitor process time
from os import stat # Line count
from zipfile import ZipFile
import yaml


# Extract the zip file containing the data set
with ZipFile('final_dataset.csv.zip', 'r') as zipObj:
   # Extract all the contents of zip file in current directory
   zipObj.extractall()

# Print File size in GB and then count the number of lines
print(f"File Size: {stat('final_dataset.csv').st_size * 1e-09} GB")
with open("final_dataset.csv", 'r') as file:
    line_count = 0
    for line in file:
        if line != "\n":
            line_count += 1
print(f"Number of rows in file: {line_count}")
```

```
File Size: 6.794744782 GB
Number of rows in file: 12794628
```

2.2 Importing data

The file was read with 2 methods; each method was timed to emphasize the importance of using the proper reading methods when reading files larger than 2 GB in size.

[3]:
```
# Reading the dataset with Dask
start = time()
df_dask = dd.read_csv("final_dataset.csv")
end = time()
print("Reading with Dask took: ", end - start, " seconds")

# Reading with pandas
# read the large csv file with specified chunksize and append chunk to a single␣
 ↪list
start = time()
chunk_list = []
with pd.read_table("final_dataset.csv", chunksize=500000, low_memory=False) as␣
 ↪reader:
    for chunk in reader:
        chunk_list.append(chunk)
df_concat = pd.concat(chunk_list)
end = time()
print("Reading wth Pandas took: ", end - start, " seconds")
```

```
Reading with Dask tool:  0.3025329113006592  seconds
Reading wth Pandas took:  96.82581210136414  seconds
```

3 Schema validation

A YAML configuration file is created inorder to automate the file reading process in the future. The configuration file specifies the essential arguments used by reading methods, and contains the expected columns, this allows us to validate header names after reading the file.

[4]:
```
# Preprocess column names
df_dask.columns = df_dask.columns.str.replace('[#, @, $, %, &, !, :, ;]', '',␣
 ↪regex=True)
df_dask.columns = df_dask.columns.str.replace(' ', '_')
df_dask.columns = df_dask.columns.str.lower()

# Write Config file
yaml_param = '''
file_name: "final_dataset.csv"
output_file: "gzip_data"
output_format: "gzip"
inbound_sep: ','
outbound_sep: '|'
rows_to_skip: 0
table_name: "df_dask"
```

```
'''

with open('config.yml', "w+") as file:
    param = yaml.full_load(yaml_param)
    yaml.dump(param, file)

with open('config.yml', "a") as file:
    yaml.dump({'columns' : list(df_dask.columns)}, file, default_flow_style=␣
 ↪False)
```

The code below creates two functions that read in the created YAML configuration file and use it to validate data in the future.

[5]:
```
# Function to read config file
def read_config(fileName):
    with open(fileName, "r") as stream:
        config = yaml.safe_load(stream)
    return(config)

# Function to validate columns of data intake
def validate_cols(data_cols, config_cols):
    """
    Function args
    -------------
    data_cols: df.columns() # A pandas.series
    config_cols: a list of column names from the configuration file
    """
    data_cols = data_cols.str.replace('[#, @, $, %, &, !, :, ;]', '',␣
 ↪regex=True)
    data_cols = data_cols.str.replace(' ', '_')
    data_cols = data_cols.str.lower()

    # Validate that columns match in both lists.
    if len(data_cols) == len(config_cols) and list(data_cols).sort() ==␣
 ↪config_cols.sort():
        print("Columns validation was successful")
        return True
    else:
        print("Columns validation has failed")
        missing_in_yaml = list(set(data_cols).difference(config_cols))
        print('The following columns were not in YAML: ', missing_in_yaml)
        missing_in_data = list(set(config_cols).difference(data_cols))
        print('The following columns were not in your data: ', missing_in_data)
        return 0
```

The configuration file is then read and used to validate the imported data set.

[ ]:
```
# Use the configuration YAML file to validate the data imported
configs = read_config("config.yml")
df = dd.read_table(configs['file_name'], sep=configs['inbound_sep'])
validate_cols(df.columns, configs['columns'])

# Compress file and store it in gz format
df.to_csv(configs['out_file'], sep=configs['outbound_sep'],␣
 ↪compression=configs['output_format'])
```