Data Ingestion details

Name: Juan Carlos
Batch: LISP01
Submission date: 16-04-2021
Submitted to: Data Glacier
Repository: https://github.com/jaycee-ds/Data_ingestion

Computational efficiency

I've been working on a dataset containing texts in many different languages and from many different sources (like newspapers or websites). It's a TSV file downloaded from Kaggle: https://www.kaggle.com/alvations/old-newspapers and weighs around 6 GB. These are the column names:

Language, Source, Date and Text.

For efficiency and technical requirements needs, I've been working on Google Cloud.

In order to find the best option to read and handle such large file, I tried out different tools and stated the reading timings for each one:

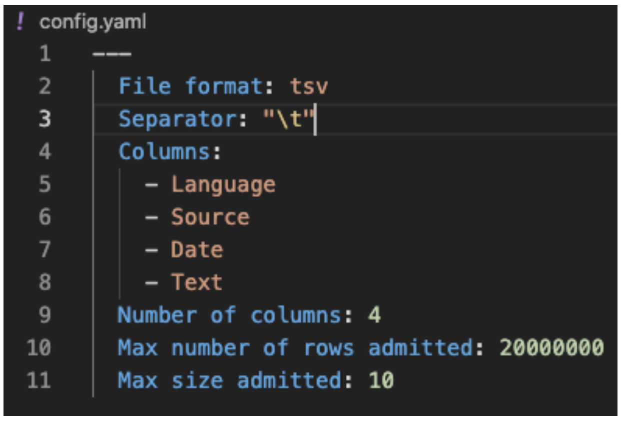
- Pandas takes between 160-180 seconds, even though it reached 229 seconds once.
- Dask takes around 0.2 seconds.
- Modin (both based on Dask and Ray) didn't work for me and filled up all the RAM capacity and restarted the runtime over and over again.
- Ray takes around 160-180 seconds. I didn't find any improvement with this framework. Maybe had to go deeper into configurations.

However, I found Dask to be the best option by far. It created 95 partitions of the whole dataframe and could handle it in a similar way I'd do using Pandas.

There are no null values, however, I could find some Sources and Dates to be "UNKNOWN" but they shouldn't be removed since the text provided is absolutely valid to work on it. For instance, all entries in Spanish come from unknown sources and it'd be a huge loss of potential information to delete them all.

Validation process

The configuration file (YAML) looks like this:

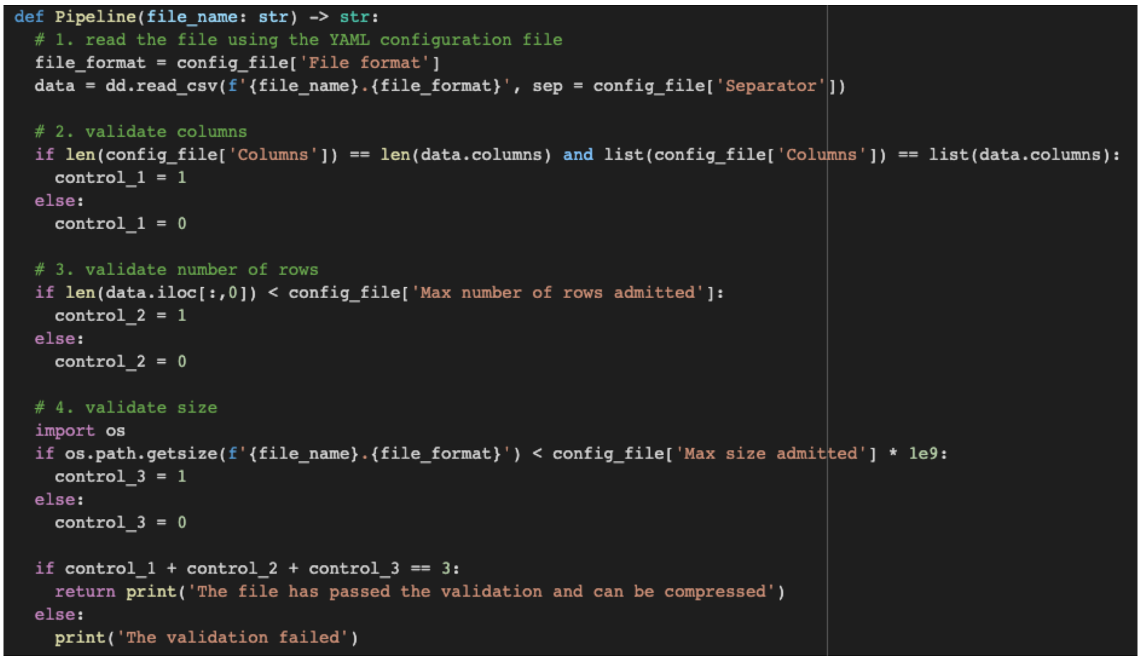


```yaml
! config.yaml
---
File format: tsv
Separator: "\t"
Columns:
  - Language
  - Source
  - Date
  - Text
Number of columns: 4
Max number of rows admitted: 20000000
Max size admitted: 10
```

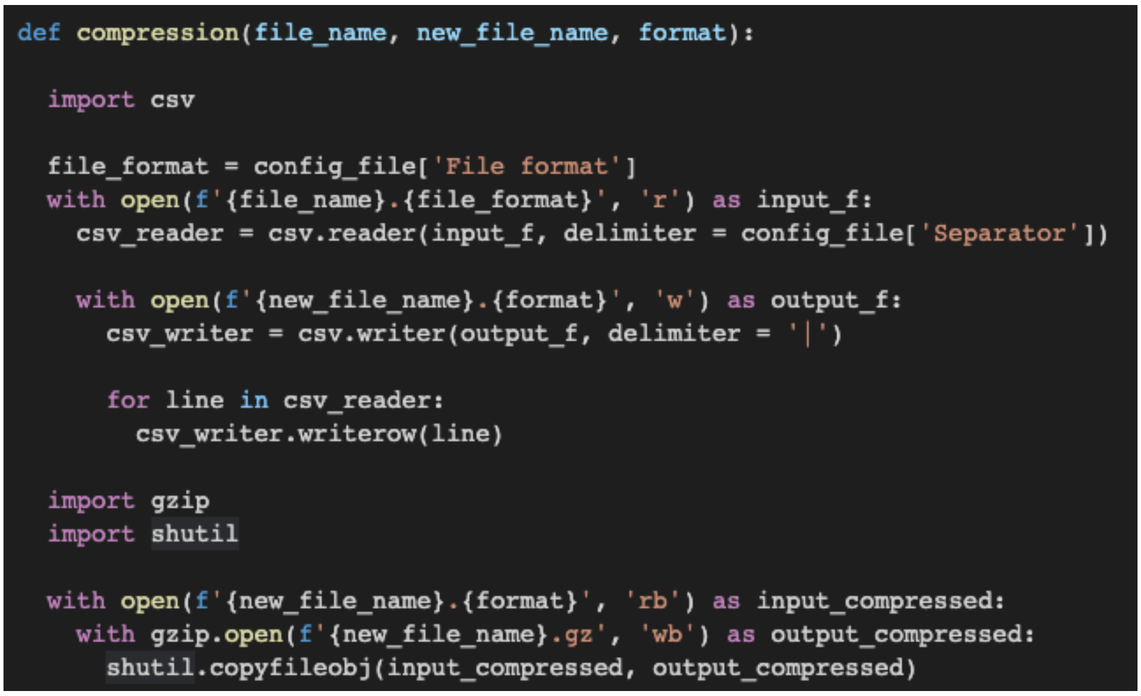
I decided to define a number of rows and size constraints.

I defined 3 functions:

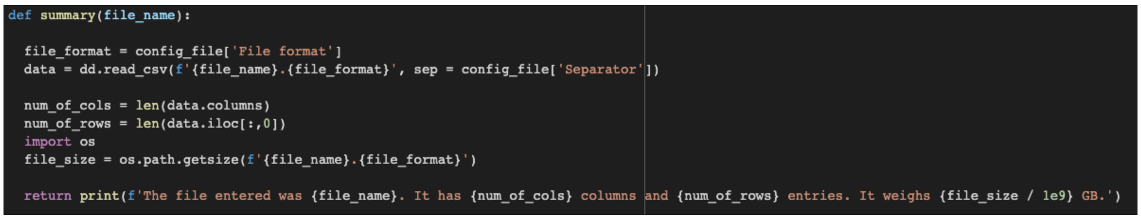
- *Pipeline* reads the file using Dask and validates columns, rows and file size.
- *Compression* re-writes the file using the pipe separator and compresses it to gz format.
- *Summary* takes an input file and returns number of rows and columns and file size.



```python
def Pipeline(file_name: str) -> str:
  # 1. read the file using the YAML configuration file
  file_format = config_file['File format']
  data = dd.read_csv(f'{file_name}.{file_format}', sep = config_file['Separator'])

  # 2. validate columns
  if len(config_file['Columns']) == len(data.columns) and list(config_file['Columns']) == list(data.columns):
    control_1 = 1
  else:
    control_1 = 0

  # 3. validate number of rows
  if len(data.iloc[:,0]) < config_file['Max number of rows admitted']:
    control_2 = 1
  else:
    control_2 = 0

  # 4. validate size
  import os
  if os.path.getsize(f'{file_name}.{file_format}') < config_file['Max size admitted'] * 1e9:
    control_3 = 1
  else:
    control_3 = 0

  if control_1 + control_2 + control_3 == 3:
    return print('The file has passed the validation and can be compressed')
  else:
    print('The validation failed')
```



```python
def compression(file_name, new_file_name, format):

  import csv

  file_format = config_file['File format']
  with open(f'{file_name}.{file_format}', 'r') as input_f:
    csv_reader = csv.reader(input_f, delimiter = config_file['Separator'])

    with open(f'{new_file_name}.{format}', 'w') as output_f:
      csv_writer = csv.writer(output_f, delimiter = '|')

      for line in csv_reader:
        csv_writer.writerow(line)

  import gzip
  import shutil

  with open(f'{new_file_name}.{format}', 'rb') as input_compressed:
    with gzip.open(f'{new_file_name}.gz', 'wb') as output_compressed:
      shutil.copyfileobj(input_compressed, output_compressed)
```



```python
def summary(file_name):

  file_format = config_file['File format']
  data = dd.read_csv(f'{file_name}.{file_format}', sep = config_file['Separator'])

  num_of_cols = len(data.columns)
  num_of_rows = len(data.iloc[:,0])
  import os
  file_size = os.path.getsize(f'{file_name}.{file_format}')

  return print(f'The file entered was {file_name}. It has {num_of_cols} columns and {num_of_rows} entries. It weighs {file_size / 1e9} GB.')
```