# week-6

June 10, 2023

[1]:
```
import os
import time
import gc
```

### 0.0.1 PANDAS - Read the file

[2]:
```
import pandas as pd

start = time.time()

pandas_df = pd.read_csv('/kaggle/input/
 ↪ibm-transactions-for-anti-money-laundering-aml/LI-Medium_Trans.csv')

finish = time.time()

diff_min, diff_sec = divmod(finish - start, 60)
diff_sec, diff_msec = divmod(diff_sec, 1)
diff_msec *= 1000

output_str = "The time taken for reading the file with Pandas: {:02d} min {:
 ↪02d} sec {:.0f} ms".format(int(diff_min), int(diff_sec), diff_msec)

print(output_str)
gc.collect()
```

```
The time taken for reading the file with Pandas: 01 min 28 sec 630 ms
```

[2]:
```
0
```

### 0.0.2 Dask - Read the file

[3]:
```
import dask.dataframe as dd

start = time.time()

dask_df = dd.read_csv('/kaggle/input/
 ↪ibm-transactions-for-anti-money-laundering-aml/LI-Medium_Trans.csv')
```

```
finish = time.time()

diff_min, diff_sec = divmod(finish - start, 60)
diff_sec, diff_msec = divmod(diff_sec, 1)
diff_msec *= 1000

output_str = "The time taken for reading the file with Dask: {:02d} min {:02d}␣
 ↪sec {:.0f} ms".format(int(diff_min), int(diff_sec), diff_msec)

print(output_str)
gc.collect()
```

```
The time taken for reading the file with Dask: 00 min 00 sec 37 ms
```

[3]:
```
0
```

### 0.1 Findings

**Dask, with its ability to parallelize operations across multiple cores and even multiple machines, boasts a faster reading and loading time compared to other techniques like typical Pandas or Ray or Modin. This efficiency in handling large datasets is a crucial advantage, especially in fields where data is the core of decision-making. With Dask's optimized computing power, data scientists and analysts can access and analyze data more efficiently, leading to faster and more accurate insights. Additionally, Dask's flexibility and compatibility with other Python libraries make it a popular choice for handling big data in various industries.**

[4]:
```
import dask.dataframe as dd
import pandas as pd

dask_df = dd.read_csv('/kaggle/input/
 ↪ibm-transactions-for-anti-money-laundering-aml/LI-Medium_Trans.csv')
dask_df.head()
```

[4]:

| | Timestamp | From Bank | Account | To Bank | Account.1 | \ |
|---|---|---|---|---|---|---|
| 0 | 2022/09/01 00:15 | 20 | 800104D70 | 20 | 800104D70 | |
| 1 | 2022/09/01 00:18 | 3196 | 800107150 | 3196 | 800107150 | |
| 2 | 2022/09/01 00:23 | 1208 | 80010E430 | 1208 | 80010E430 | |
| 3 | 2022/09/01 00:19 | 3203 | 80010EA80 | 3203 | 80010EA80 | |
| 4 | 2022/09/01 00:27 | 20 | 800104D20 | 20 | 800104D20 | |

| | Amount Received | Receiving Currency | Amount Paid | Payment Currency | \ |
|---|---|---|---|---|---|
| 0 | 8095.07 | US Dollar | 8095.07 | US Dollar | |
| 1 | 7739.29 | US Dollar | 7739.29 | US Dollar | |
| 2 | 2654.22 | US Dollar | 2654.22 | US Dollar | |
| 3 | 13284.41 | US Dollar | 13284.41 | US Dollar | |
| 4 | 9.72 | US Dollar | 9.72 | US Dollar | |

```
  Payment Format  Is Laundering
0   Reinvestment              0
1   Reinvestment              0
2   Reinvestment              0
3   Reinvestment              0
4   Reinvestment              0
```

[5]:
```
dask_df.info()
```

```
<class 'dask.dataframe.core.DataFrame'>
Columns: 11 entries, Timestamp to Is Laundering
dtypes: object(6), float64(2), int64(3)
```

[6]:
```
start = time.time()

# remove special character
dask_df.columns = dask_df.columns.str.replace('[#,@,&,.,1]','')

# removing whitespaces
dask_df.columns = dask_df.columns.str.replace(' ', '')

finish = time.time()
diff_min, diff_sec = divmod(finish - start, 60)
diff_sec, diff_msec = divmod(diff_sec, 1)
diff_msec *= 1000

output_str = "Performing simple string removal from the columns: {:02d} min {:
 ↪02d} sec {:.0f} ms".format(int(diff_min), int(diff_sec), diff_msec)

print(output_str)
```

```
Performing simple string removal from the columns: 00 min 00 sec 20 ms
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:4: FutureWarning:
The default value of regex will change from True to False in a future version.
  after removing the cwd from sys.path.
```

### 0.1.1 Validation with YAML

[7]:
```
%%writefile testutility.py
import logging
import os
import subprocess
import yaml
import pandas as pd
import datetime
import gc
import re
```

```
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
    '''
    df.columns = df.columns.str.lower()
    df.columns = df.columns.str.replace('[^\w]','_',regex=True)
    df.columns = list(map(lambda x: x.strip('_'), list(df.columns)))
    df.columns = list(map(lambda x: replacer(x,'_'), list(df.columns)))
    expected_col = list(map(lambda x: x.lower(),  table_config['columns']))
    expected_col.sort()
    df.columns =list(map(lambda x: x.lower(), list(df.columns)))
    df = df.reindex(sorted(df.columns), axis=1)
    if len(df.columns) == len(expected_col) and list(expected_col)  == list(df.
 ↪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(df.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(df.columns))
        print("Following YAML columns are not in the file␣
 ↪uploaded",missing_YAML_file)
        logging.info(f'df columns: {df.columns}')
        logging.info(f'expected columns: {expected_col}')
        return 0
```

```
Writing testutility.py
```

### 0.1.2 Write YAML file

[8]:
```
dask_df.head()
```

[8]:

| | Timestamp | FromBank | Account | ToBank | Account | AmountReceived | \ |
|---|---|---|---|---|---|---|---|
| 0 | 2022/09/01 00:15 | 20 | 800104D70 | 20 | 800104D70 | 8095.07 | |
| 1 | 2022/09/01 00:18 | 3196 | 800107150 | 3196 | 800107150 | 7739.29 | |
| 2 | 2022/09/01 00:23 | 1208 | 80010E430 | 1208 | 80010E430 | 2654.22 | |
| 3 | 2022/09/01 00:19 | 3203 | 80010EA80 | 3203 | 80010EA80 | 13284.41 | |
| 4 | 2022/09/01 00:27 | 20 | 800104D20 | 20 | 800104D20 | 9.72 | |

| | ReceivingCurrency | AmountPaid | PaymentCurrency | PaymentFormat | IsLaundering |
|---|---|---|---|---|---|
| 0 | US Dollar | 8095.07 | US Dollar | Reinvestment | 0 |
| 1 | US Dollar | 7739.29 | US Dollar | Reinvestment | 0 |
| 2 | US Dollar | 2654.22 | US Dollar | Reinvestment | 0 |
| 3 | US Dollar | 13284.41 | US Dollar | Reinvestment | 0 |
| 4 | US Dollar | 9.72 | US Dollar | Reinvestment | 0 |

[9]:
```
%%writefile file.yaml
file_type: csv
dataset_name: testfile
file_name: LI-Medium_Trans
table_name: edsurv
inbound_delimiter: ","
outbound_delimiter: "|"
skip_leading_rows: 1
columns:
    - Timestamp
    - FromBank
    - Account
    - ToBank
    - Account
    - AmountReceived
    - ReceivingCurrency
    - AmountPaid
    - PaymentCurrency
    - PaymentFormat
    - IsLaundering
```

```
Writing file.yaml
```

[10]:
```
# Read config file
import testutility as util
config_data = util.read_config_file("file.yaml")
```

[11]:
```
config_data['inbound_delimiter']
```

[11]:
```
','
```

[12]:
```
#inspecting data of config file
config_data
```

[12]:
```
{'file_type': 'csv',
 'dataset_name': 'testfile',
 'file_name': 'LI-Medium_Trans',
 'table_name': 'edsurv',
 'inbound_delimiter': ',',
 'outbound_delimiter': '|',
 'skip_leading_rows': 1,
 'columns': ['Timestamp',
  'FromBank',
  'Account',
  'ToBank',
  'Account',
  'AmountReceived',
  'ReceivingCurrency',
  'AmountPaid',
  'PaymentCurrency',
  'PaymentFormat',
  'IsLaundering']}
```

[13]:
```
import dask.dataframe as dd
import pandas as pd

sample_df = dd.read_csv('/kaggle/input/
 ↪ibm-transactions-for-anti-money-laundering-aml/LI-Medium_Trans.csv')
sample_df.head()
```

[13]:

| | Timestamp | From Bank | Account | To Bank | Account.1 | Amount Received | Receiving Currency | Amount Paid | Payment Currency | Payment Format | Is Laundering |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 2022/09/01 00:15 | 20 | 800104D70 | 20 | 800104D70 | 8095.07 | US Dollar | 8095.07 | US Dollar | Reinvestment | 0 |
| 1 | 2022/09/01 00:18 | 3196 | 800107150 | 3196 | 800107150 | 7739.29 | US Dollar | 7739.29 | US Dollar | Reinvestment | 0 |
| 2 | 2022/09/01 00:23 | 1208 | 80010E430 | 1208 | 80010E430 | 2654.22 | US Dollar | 2654.22 | US Dollar | Reinvestment | 0 |
| 3 | 2022/09/01 00:19 | 3203 | 80010EA80 | 3203 | 80010EA80 | 13284.41 | US Dollar | 13284.41 | US Dollar | Reinvestment | 0 |
| 4 | 2022/09/01 00:27 | 20 | 800104D20 | 20 | 800104D20 | 9.72 | US Dollar | 9.72 | US Dollar | Reinvestment | 0 |

```
3   Reinvestment              0
4   Reinvestment              0
```

[14]:
```
# read the file using config file
file_type = config_data['file_type']
source_file = "/kaggle/input/ibm-transactions-for-anti-money-laundering-aml/" +␣
 ↪config_data['file_name'] + f'.{file_type}'
```

[15]:
```
del pandas_df, dask_df
gc.collect()
```

[15]:
```
249
```

[16]:
```
#print("",source_file)
df = pd.read_csv(source_file,config_data['inbound_delimiter'])
df.head()
```

```
/opt/conda/lib/python3.7/site-packages/IPython/core/interactiveshell.py:3553:
FutureWarning: In a future version of pandas all arguments of read_csv except
for the argument 'filepath_or_buffer' will be keyword-only
  exec(code_obj, self.user_global_ns, self.user_ns)
```

[16]:

| | Timestamp | From Bank | Account | To Bank | Account.1 | \ |
|---|---|---|---|---|---|---|
| 0 | 2022/09/01 00:15 | 20 | 800104D70 | 20 | 800104D70 | |
| 1 | 2022/09/01 00:18 | 3196 | 800107150 | 3196 | 800107150 | |
| 2 | 2022/09/01 00:23 | 1208 | 80010E430 | 1208 | 80010E430 | |
| 3 | 2022/09/01 00:19 | 3203 | 80010EA80 | 3203 | 80010EA80 | |
| 4 | 2022/09/01 00:27 | 20 | 800104D20 | 20 | 800104D20 | |

| | Amount Received | Receiving Currency | Amount Paid | Payment Currency | \ |
|---|---|---|---|---|---|
| 0 | 8095.07 | US Dollar | 8095.07 | US Dollar | |
| 1 | 7739.29 | US Dollar | 7739.29 | US Dollar | |
| 2 | 2654.22 | US Dollar | 2654.22 | US Dollar | |
| 3 | 13284.41 | US Dollar | 13284.41 | US Dollar | |
| 4 | 9.72 | US Dollar | 9.72 | US Dollar | |

| | Payment Format | Is Laundering |
|---|---|---|
| 0 | Reinvestment | 0 |
| 1 | Reinvestment | 0 |
| 2 | Reinvestment | 0 |
| 3 | Reinvestment | 0 |
| 4 | Reinvestment | 0 |

[17]:
```
#validate the header of the file
util.col_header_val(df,config_data)
```

```
column name and column length validation failed
Following File columns are not in the YAML file ['account_1', 'amount_received',
'from_bank', 'amount_paid', 'is_laundering', 'payment_currency', 'to_bank',
```

```
'receiving_currency', 'payment_format']
Following YAML columns are not in the file uploaded ['frombank',
'receivingcurrency', 'amountreceived', 'islaundering', 'tobank', 'amountpaid',
'paymentcurrency', 'paymentformat']
```

[17]:
```
0
```

[18]:
```
print("columns of files are:" ,df.columns)
print("columns of YAML are:" ,config_data['columns'])
```

```
columns of files are: Index(['timestamp', 'from_bank', 'account', 'to_bank',
'account_1',
       'amount_received', 'receiving_currency', 'amount_paid',
       'payment_currency', 'payment_format', 'is_laundering'],
      dtype='object')
columns of YAML are: ['Timestamp', 'FromBank', 'Account', 'ToBank', 'Account',
'AmountReceived', 'ReceivingCurrency', 'AmountPaid', 'PaymentCurrency',
'PaymentFormat', 'IsLaundering']
```

[19]:
```
if util.col_header_val(df,config_data)==0:
    print("validation failed")
else:
    print("col validation passed")
    df_clean = df.fillna(0)
    df_transformed = df_clean.apply(lambda x: x**2)
    df_transformed.to_csv('LI-Medium_transformed.csv')
```

```
column name and column length validation failed
Following File columns are not in the YAML file ['account_1', 'amount_received',
'from_bank', 'amount_paid', 'is_laundering', 'payment_currency', 'to_bank',
'receiving_currency', 'payment_format']
Following YAML columns are not in the file uploaded ['frombank',
'receivingcurrency', 'amountreceived', 'islaundering', 'tobank', 'amountpaid',
'paymentcurrency', 'paymentformat']
validation failed
```

[20]:
```
import csv
import gzip

from dask import dataframe as dd
df = dd.read_csv('/kaggle/input/ibm-transactions-for-anti-money-laundering-aml/
 ↪LI-Medium_Trans.csv')

# Write csv in gz format in pipe separated text file (|)
df.to_csv('LI-Medium_Trans.csv.gz',
          sep='|',
          header=True,
          index=False,
          quoting=csv.QUOTE_ALL,
```

```
            compression='gzip',
            quotechar='"',
            doublequote=True,
            line_terminator='\n')
```

[20]:
```
['/kaggle/working/LI-Medium_Trans.csv.gz/00.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/01.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/02.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/03.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/04.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/05.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/06.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/07.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/08.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/09.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/10.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/11.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/12.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/13.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/14.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/15.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/16.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/17.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/18.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/19.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/20.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/21.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/22.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/23.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/24.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/25.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/26.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/27.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/28.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/29.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/30.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/31.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/32.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/33.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/34.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/35.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/36.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/37.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/38.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/39.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/40.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/41.part',
```

```
 '/kaggle/working/LI-Medium_Trans.csv.gz/42.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/43.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/44.part',
 '/kaggle/working/LI-Medium_Trans.csv.gz/45.part']
```

[21]:
```python
# Get file summary
file_size = os.path.getsize('LI-Medium_Trans.csv.gz')
num_rows = len(df)
num_cols = len(df.columns)

# Print file summary
print("File summary:")
print(f"Number of rows: {num_rows}")
print(f"Number of columns: {num_cols}")
print(f"File size: {file_size} bytes")
```

```
File summary:
Number of rows: 31251483
Number of columns: 11
File size: 4096 bytes
```