SAS Parser

Text Parser to Normalized CSV in Python.

Problem

Problem frame: No comprehensive inventory of SAS scripts; both inventory of files as well as inventory of contents. For example, cannot answer questions like:

- How many SAS scripts are there across multiple directories, when were they created and last accessed?
- How can I find out which SAS scripts reference a specific data table?
- How can I find what SAS scripts reference / import other SAS scripts, and which ones?
- How can I find SAS scripts that reference a particular SQL table column?
- How can I find SAS scripts that have hardcoded dates?
- and so on...

Approach

Attempting to predict all code content questions is an intractable problem. An approach to facilitate all manner of information needs would be to create a generalized approach that allows a developer to create a ***parser*** that can generate queryable results to answer specific questions.

One way this could be approached is by using specific commands already provided by the OS.

For example, if you wanted to know the line count for each SAS script in a directory, you might use the WC command. Or if you wanted to know which scripts contained hardcoded dates, you could use the SED and regex. But this approach could get messy and the results would not be a uniform format for doing aggregate queries.

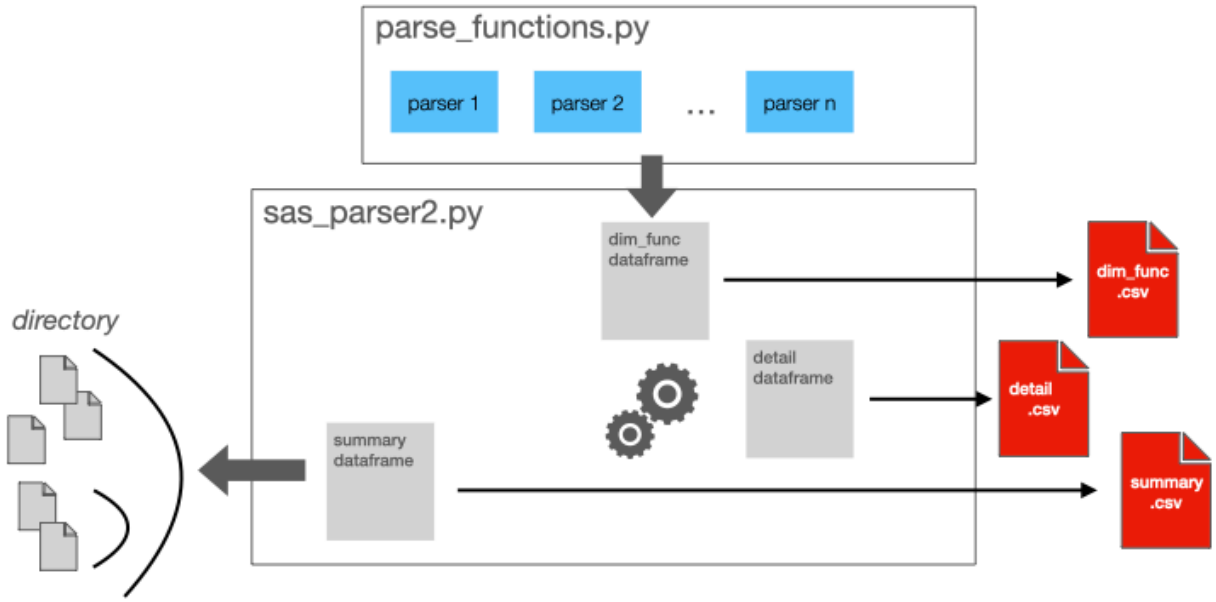
The approach used here is more generalized and consistent - one that can work on a range of OS's, requiring only knowledge of Python and SQL.

1. ***main routine*** that executes any number of predefined [Python] parser functions on a directory (and sub-directories). This ***main routine*** sequentially executes the list of parsers on each specified ***file type*** in the directory and then outputs the results to csv files in a directory, also specified on the command line.

2. ***parser*** functions that mine information from one or more [text] scripts - returning it in a format (CSV) that can be ingested by most databases and then queried for various aspects of that parsed information.

Solution

Architecture



The solution consists of 2 python scripts:

1. sas_parser2.py
2. parse_functions.py

where ***sas_parser2.py*** contains the main routine for:

1. ingesting the cli parameters,
2. inserting the list of parse functions that will be executed to the ***dim_func*** dataframe
3. reading the directory and creating the list of files to be inspected
4. inserting the file list into the ***summary*** dataframe
5. executing the parse functions one at a time on each of the files
6. inserting the results of each parse function to the ***detail*** dataframe
7. writing out each of the dataframes to their respective CSV files

and ***parse_functions.py*** contains the individual parse functions

CSV Output File Structure and Relationship

summary

column name	type
f_name	integer
dir_path	text
create_dt	datetime w tz
modified_dt	datetime w tz

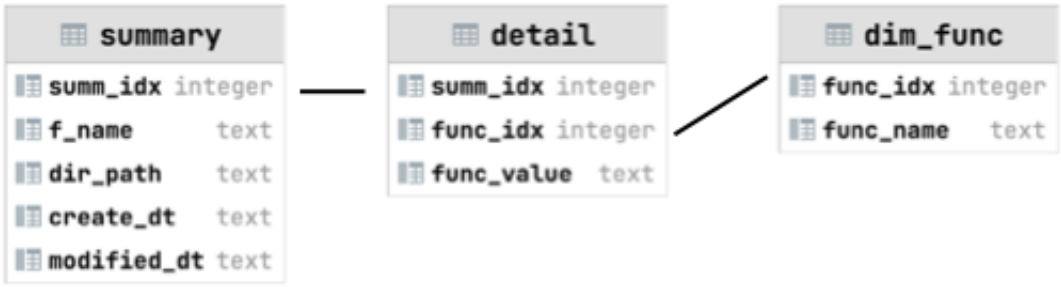
detail

column name	type
summ_id	integer
func_idx	integer
func_value	text

dim_func

column name	type
func_idx	integer
func_name	text

Relationship



Requirements

Python 3.x Python environment with the following packages installed:

- pandas
- tqdm

How To Run

There are 2 highlevel steps to working with the parser:

1. run the parser code on the directory of files
2. import the parser output CSVs into a database for analysis

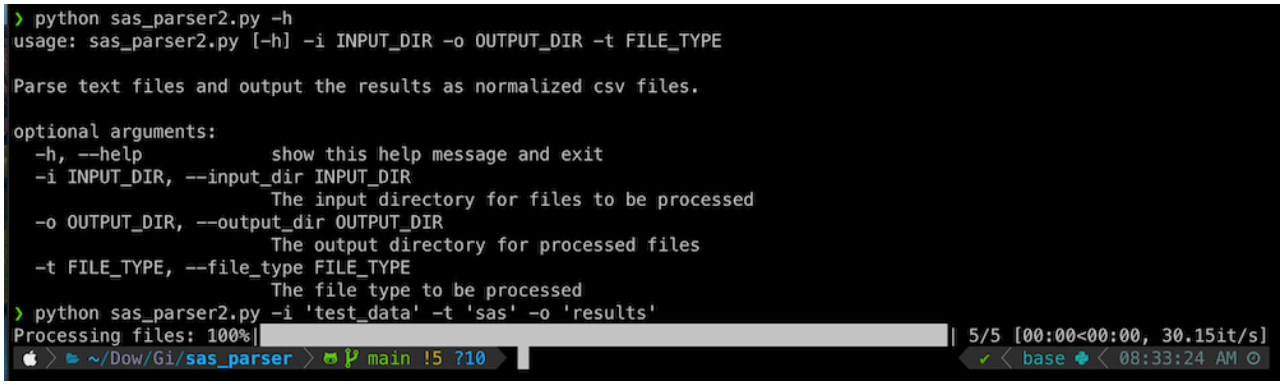
> **Note** Analysis of the output is beyond the scope of this document

Parsing

To run the code with pre-selected parsers, first clone the repository, activate the python environment and then perform any of the following:

```
# command and example command line arguments for sas files
# -i is the input directory, -t is the file extension, -o is the output d
irectory
> python sas_python2.py -i 'some_directory_of_scripts' -t 'sas' -o 'resul
ts'
```

On execution, the python script will output a status bar as it performs each of the parsers on the file directory specified.



> **Note** If you only want specific parsers to execute, you will need to edit the sas*parser2.py file and modify the variable ***functions*to_apply***

```
# List of functions to apply
functions_to_apply = [
    get_file_info,
    count_lines,
    count_sql,
    get_sql_code,
    get_libname_lines,
    count_exports,
    count_null_ds,
    find_date_lines,
    find_file_references]
```

Output

If there are no errors,three CSV files are output into the ***output*** directory in the following format:

- dim*func*yyyymmddhhmmss.csv
- summary_yyyymmddhhmmss.csv
- detail_yyyymmddhhmmss.csv

where yyyymmddhhmmss is the datetime the command was run.

Querying

The easiest way to work with the sas_parser2.py output CSV files is to import them into a database and query for the particular details you are interested in. Refer to the relationship of the CSV file in the Solution section

Assuming you have already imported the CSV files into a database, with table names corresponding to the csv file names, here is an example query:

Q: List all the files in the directory (and sub-directories) sorted by lines-of-code count (descending)

```
select summary.f_name as file_name, summary.dir_path, cast(detail.func_va
lue as int) as line_ct
from summary
join detail on summary.summ_idx = detail.summ_idx
join dim_func on detail.func_idx = dim_func.func_idx
where dim_func.func_name = 'count_lines'
order by cast(detail.func_value as int)
```

Q: List files referencing an external file (in the input directory) named: master_script.sas

```
select
  summary.f_name as file_name,
  summary.dir_path,
  summary.create_dt,
  summary.modified_dt,
  dim_func.func_name,
  detail.func_value
from summary
join detail on summary.summ_idx = detail.summ_idx
join dim_func on detail.func_idx = dim_func.func_idx
where
  dim_func.func_name in ('find_file_references')
  and
  upper(detail.func_value) like '%MASTER_SCRIPT.SAS%';
```

Technical

The parsing function is conducted using two Python scripts:

1. sas_parser2.py - directory reader, csv writer and main routine
2. parse*functions.py - module of parsing functions called from the main function in sas*parser.py

sas_parser.py

This script uses 3 pandas data frames: dim*func*df, summary*df and detail*df to store the data that will, at the end, be written out to the corresponding csv files.

Highlevel, the script performs the following operations: 1. parse the cli arguments 2. insert the parsing functions into the ***dimfuncdf*** data frame 3. read the directory file into a list 4. iterate over the file list 4a. insert file info into the ***summary_df*** 4b. for each parsing function - evaluate the file with the parsing function - insert the parsing function results into the ***detail_df*** data frame 5. write out the CSV files

parse_functions.py

This script contains all the parse functions called from sas_parser.py The general pattern used by the parse functions are: - call the parse function with the file path - perform the 'parse' code - return the function name [string] and corresponding value [string, int, tuple, list, etc]

```
def count_lines(file_path):
    """Count the number of lines in a file

    Args:
        file_path (string): full path to the file

    Returns:
        integer: the number of lines in the file
    """
    with open(file_path, 'r', encoding='cp1252') as file:  # Open the file
        lines = file.readlines()  # Read all lines into a list
    return ("line_count", [len(lines)])  # Return the number of lines
```

Of course, you could call a parse function with any number of arguments. But be aware that it would need to be addressed in the sas_parser.py main function.

For example, the ***findfilereferences*** function needs two arguments, and thus, there is a conditional statement that checks if the function has one or two arguments and calls accordingly.

Notes

- Current performance, with 9 parser functions evaluated, is about 600 files per minute

To-Do

Things to consider adding or improving on:

- parallelizing (multiprocess python package) the main parse routine
- pass a copy of the script (text) to each parser instead of the filename - so as to avoid opening the file for each parser