[[1]](#footnote-1)

Python and MySQL  
Useful for Data Cleaning?

Zackary Gill (zgill@smu.edu) and Mai Loan Tran (mltran@smu.edu)

*Abstract*—In this paper, we present a case study in using Python and MySQL to clean the vendor names of travel and reimbursable expense data. Company X, whose name cannot be disclosed, requires monthly reporting by vendor as part of its business objective to controls costs. The vendor names come from multiple sources with different variations. The vendor names need to be standardized to satisfy the reporting requirements for management, spend analysis, and vendor negotiations. The standardization of vendor names within the existing SQL Server schema is a time-consuming labor intensive process of manual inspections and writing SQL select and update statements to modify the values. By modifying the database schema with two reference tables in MySQL and programming logic using Python to track both valid and invalid values, our experience suggest a reduction in time and labor of the vendor name standardization across all expense types. A user interface to populate the reference tables with automated repeatable comparisons provide a non-technical user with a utility to perform the data quality function without writing SQL statements. An automated ratio computation for similar vendor names narrow down the values for selection and standardization, thus reducing manual inspection of every value while allowing for custom ones.

*Index Terms*—None.

# INTRODUCTION

E

very company has a requirement to control costs. The management of these costs requires understanding with whom it is incurred --- who are the supplying vendors? Company X, whose name cannot be disclosed due to a binding employee non-disclosure agreement, is no exception in seeking this visibility with its travel and reimbursable expenses. The company’s accountants record the multiple sources of these expense transactions as journal entries within its accounts payable sub-ledger. A SQL Server database was recently created to house a monthly extract from the ledger of these travel and reimbursed expense entries across eighty-two expense types. In order for these entries to provide useful vendor information for analysis, reporting, and vendor negotiations, we need to standardize the vendor names with its numerous variations from users’ free text inputs, credit card transactions, and vendor invoices from over forty (40) countries.

The more sources of these vendor names, the more variations in values; hence, the more difficult to standardize the data values. With manual inspection of each variation limiting the standardization of vendor names to higher spend expense types such as airfare, hotel/lodging, and car rentals and excluding the rest, still leaves vendor name inconsistencies. The same vendor is not standardized across all expense types. Using SQL select and update statements to standardize the vendor name can take 2-3 days per month given the range of 21,000 to 132,000 new tuples monthly for the last two years. Furthermore, a vendor name can be standardized for previous months’ entries, but new entries are are still subject to manual inspection and SQL statement updates for each subsequent month. The risk of errors and inaccuracies resulting in missed variations along with null values from a manual row-by-row review remains. A solution needs to be identified.

Our case study seeks to apply our academic introduction of relational databases and Python to help Company X resolve the data cleaning challenges of vendor names within its travel and reimbursable expense entries. MySQL was chosen primarily because it is free and offers the relational database architecture with a readily available user interface similar to Company X’s SQL Server. Python was chosen for its simple programming syntax, code readability and English-like commands that make coding it a lot easier and efficient in attempting to solve Company X’s problem.

In the effort to reduce the time and labor of manual inspections in cleaning the travel and reimbursable expense data, the solution design provides the tracking of valid and invalid vendor name values within two reference tables that did not exist within Company X’s database schema. The solution also replaced the task of writing numerous SQL select and update statements with a rudimentary user interface in Python. This Python-based utility provides a non-technical user with functionality to review like values based on a computed ratio from comparisons against the reference table values that are served up to the user by the programming logic. From the program’s suggested values, the user can select the desired valid value or elect to enter a custom value. In both cases, the inputted value is written to each respective reference table, the correct “lookup” value and incorrect “mistake” value where the “mistake” table contains both the valid and invalid values. These reference tables serve as references to compare new vendor name values for subsequent months. Thus, providing an automated data cleaning utility.

# Background

Company X has a monthly process that takes in an average of 41,708 tuples per month from end users, credit card transactions, and invoice payments for travel and reimbursed expense from over 1,000 employees and contractors in more than forty (40) offices globally. These transactions with the raw vendor and subvendor name from the various sources are consolidated within an accounts payable (AP) sub-ledger with its own database schema. In order to meet reporting requirements, the travel and reimbursable expense transactions are extracted into a fact table within SQL Server. SQL Server Management Studio (SSMS) is used to establish additional relationships with tables from other systems such as a Human Resources Information System (HRIS) for employee details, average foreign exchange rates conversion from Oracle, and other reportable attributes. The standardized vendor name column is one of these reportable attributes; however, it is currently not subject to any additional table relationships. It is subject to a non-technical user’s manual inspection of the combination of the raw Vendor Name and Sub Vendor values within each journal entry’s tuple to determine the correct spelling and standardized value that is used within management dashboards. Fig. 1. illustrates Company X’s data cleaning process for these travel and reimbursable expense entries.

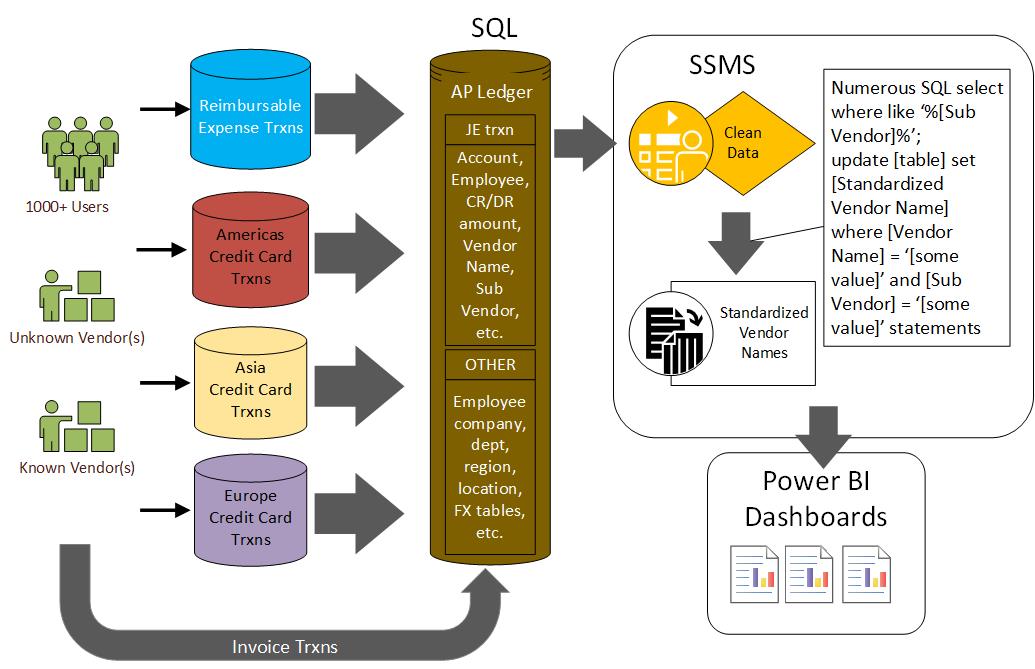
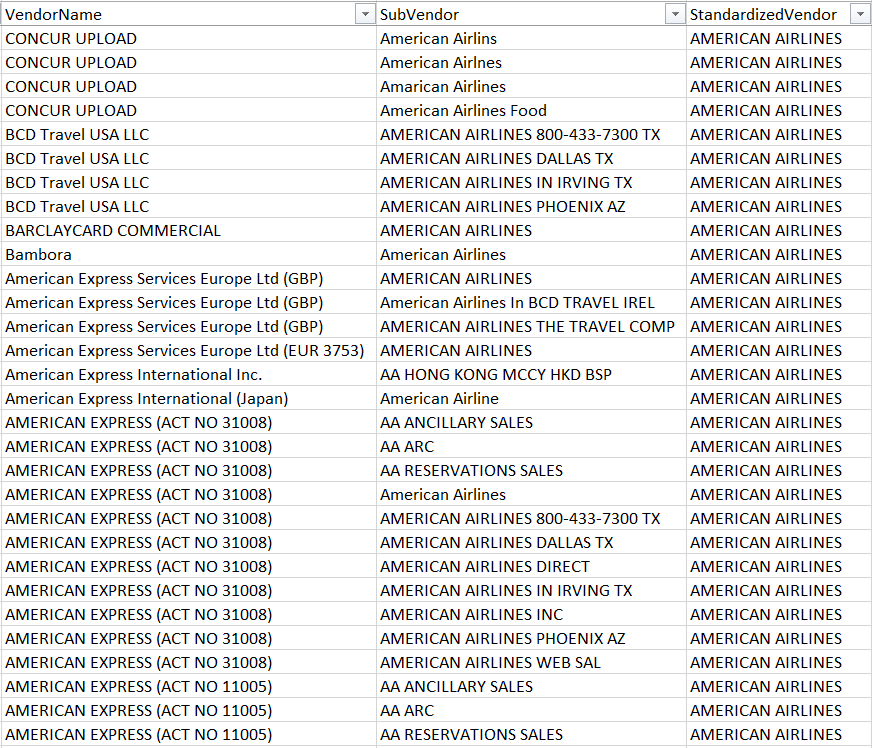
 Numerous SQL queries and update statements are written to locate, correct spelling, and standardize variations of each combination of the raw Vendor Name and Sub Vendor values. This is an extremely labor-intensive and tedious time-consuming process that can vary in duration depending on the number of tuples and distinct combinations entered each month. Thus, the focus is cleaning vendor names for only high dollar spend categories like airfare, hotel/lodging, and car rental vendors while excluding the vendor names of the remaining spend categories with its own misspellings and variations. Table I shows an example of thirty (30) variations of the SubVendor value for ‘AMERICAN AIRLINES’ from nine (9) Vendor Name sources.

Fig. 1. Company X’s travel and reimburable expense entries’ data cleaning process.

TABLE I

Variations to be evaluated for ‘AMERICAN AIRLINES’



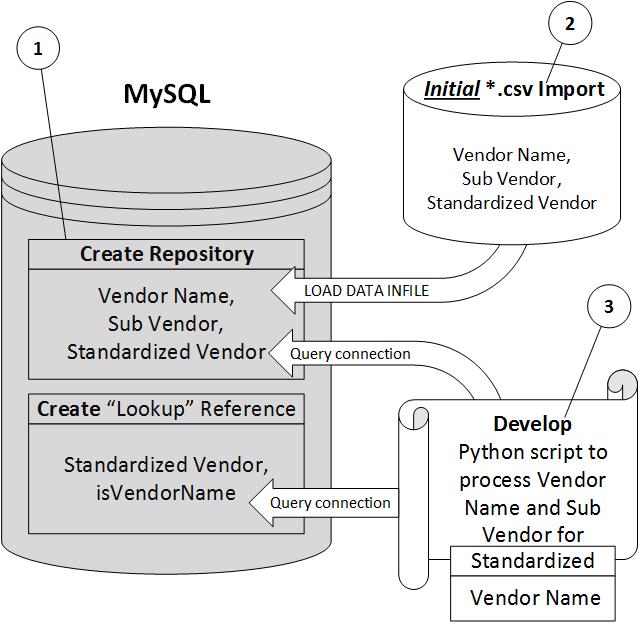
In the provided data set, there are seventy-one (71) variations of ‘AMERICAN AIRLINES’ but the volume grows with each subsequent month’s entries. While the data disparity appear small at this point, the struggle to keep pace with the volume of data as it grows through the years has a foreseeable compounded effect in the spend analysis with the unique number of vendors. Understanding the context and relevance of this data type at this stage mitigates data quality frustrations in the downstream management reporting and vendor negotiations.

Fig. 2. Initial Solution Design

For this case study, Company X provided 1,009,303 tuples from the last two calendar years 2017-2018 to determine whether a solution designed with Python and MySQL can provide a better method to clean and standardize these vendor names.

While MySQL was chosen primarily because it is free, it more importantly is a relational database with a readily available user interface that has similar capabilities to Company X’s SQL Server for the scope of this project. MySQL’s quick start capability from software download to complete installation regardless of underlying platform and comprehensive set of migration tools sufficiently enables a test environment that simulates Company X’ data model.

For the setup to be easily replicable, Anaconda was used to manage the installed libraries and packages including pandas and mysql.connection. Python was chosen not only for its simple programming syntax, code readability and English-like commands that make coding a lot easier and efficient in manipulating Company X’s data, but also because of its viability as a programming language to resolve real-world data problems like our project.

# Solution design

As the solution requires creating a repeatable method to clean and standardize the vendor data, the design decision to use MySQL is because it is a cost-effective open source relational database from which Company X can adapt the solution within its SQL Server relational schema. Python and pandas was selected purely for the practical experience from the academic exposure to solve the business problem of cleaning data.

Company X’s data is imported into MySQL and accessed with a Python script. The data manipulation within Python uses the pandas library.

Fig. 2. Initial Solution Design depicts the basic workflow of the solution to create a reference “lookup” table, import all the distinct valid combinations in the table, then develop a Python script that references the “lookup” table to automatically populate the valid entries into the standardized vendor name field.

Serious learning was gained in exploring the differing ways to import the data into MySQL. Using the Table Data import wizard was attempted first. While intuitive and user friendly, it was incredibly slow. Taking about a ½ sec for each row, this method was aborted with 1 million rows in the provided data set.

A second method to import the data using the LOAD DATA INFILE command proved more efficient. When starting MySQL, there is a secure file option which limits the directories available for use to load files. Placing the data in the specified secure location, the million plus row data set took 10 seconds to import. The LOAD DATA INFILE operation has multiple parameters similarly to many other programming languages with a delimiter. One of the setting is the CHARACTER SET in the data import to make sure that the database is set to support the appropriate languages.

For Python, Anaconda was used to manage the installation of the libraries and packages. To ensure our setup is replicable, a new environment was created with minimal libraries required including pandas and mysql.connection.

As there are multiple libraries for connecting to MySQL from Python, the officially supported one by MySQL was chosen in considering future update releases, although literature suggested that it is not the fastest. A significant detail learned during this step is to encase the connection within a TRY object to facilitate the ease of trouble-shooting when the connection fails. Furthermore, if a different connector is selected in the future, the majority of the code using the current connector will not need to be altered because the query utilizes the connection object. For example, when connecting to Microsoft SQL Server database only the connector has to be modified rather the underlying code that queries the database.

For manipulating the database schema, a cursor was used to execute the queries. As there are many different types, it was determined that a “Buffered” cursor best served the objective to iterate through the records. However, to retrieve the data, the cursor is not necessary. Instead, the pandas library was chosen because it already contains data structures and data analysis tools. It also has a method for querying a database and returning a table without modifying the schema.

During the development of the program, we determined that a design modification was necessary to better facilitate the automation and verification of the standardized vendor name values. We decided to add another reference table and only import the Sub Vendor values with the logic to use the Vendor Name if the original Sub Vendor value is null.

Logic was added to determine how similar one value is to another is called a ‘ratio’. The ratio was set between ‘0’ to ‘1’, inclusive. If a ratio equals ‘1’, the two values are a perfect match. This value is the key for checking the values in the ‘lookup’ and ‘mistake’ tables.

With two (2) reference tables, the programming logic first compares the inputted value with the ‘lookup’ for the correct values then referenced the second table to identify and track the ‘mistakes’. The ‘lookup’ table contains the valid Sub Vendor values with the standardized vendor name. The ‘mistake’ table contains the correct valid standardized vendor name with the incorrect vendor name. Because all of the distinct ‘mistakes’ are kept, this simulates the presence of a reference table which is the final solution shown in Fig. 3.

Both tables in MySQL are imported by the Python script to serve as comparison to the original Vendor Name and Sub Vendor data values. Within the Python script, two functions are defined to find the percentage similarity (ratio) with the correct ‘lookup’ vendor name and incorrect ‘mistake’ vendor names. Each function creates an empty dataframe of ratios for the correct and incorrect values, iterates through all the rows in each table, then returns the ratio for each sorted in ascending order. Another Python function then merges all the ratios from both of the previous functions, compares the original values, removes duplicates, and retains the highest ratio for each value in the ‘lookup’ table.

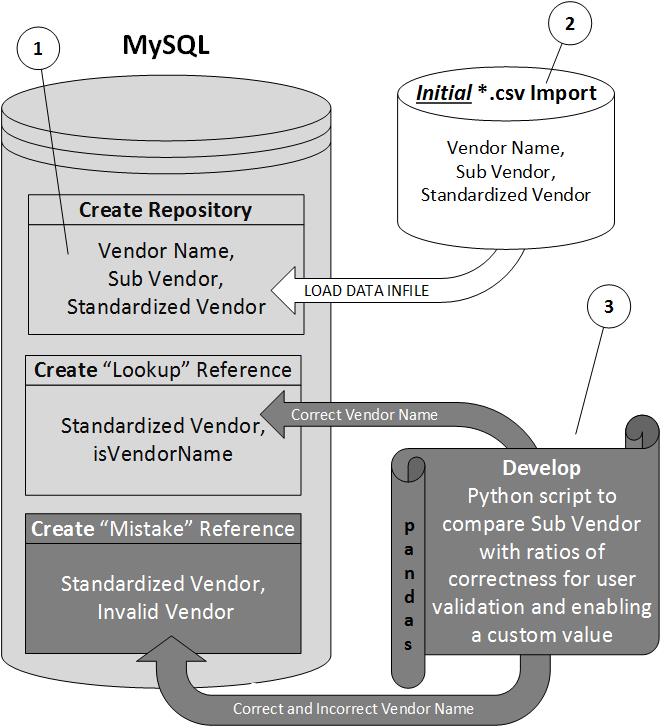
It was determined that using the ‘Buffered’ cursor was unnecessary and instead the SQL clause ‘LIMIT’ was used because it can get specified rows in a table. This was done to allow the operator of the solution to stop and start running the program at any point in time. Also, the use of this allows the management of how much data can be imported into Python at a time.

Fig. 3. Final Solution Design

Development of a simple user interface enables the user responsible for cleaning the vendor data to review the corresponding ratio with the matching values N at a time (N is any number). As mentioned earlier, the larger the value of the ratio (between 0-1) the more exact the match. Once the ratios have been determined, if there is a perfect match (ratio = 1) the script automatically inserts updates the ‘lookup’ and ‘mistake’ tables and continues on to the next value. With the ratio being any other value, the user can select the suggested match or create a custom value if the function did not return a satisfactory match. The user is then prompted whether to keep the custom value. When the user creates the custom value, it is compared to all the ones in the ‘lookup’ and ‘mistake’ tables to allow the user to either use the lookup as a search, or to actually use the custom value. If the custom value is used the value is inserted in both the valid ‘lookup’ and invalid ‘mistake’ tables for future comparison. Otherwise the custom value is discarded and the user selects one of the values that already exist within the ‘lookup’ table.

Whenever the desired value is selected automatically, by the user, or by the creation of a custom value, the script updates the Standardized Vendor field in the database.

This approach enables flexibility for comparison of both valid and invalid values to ensure the data accuracy instead of just a comparison for the correct value. Furthermore, it provides the user with a process to build the repository with an automated repeatable comparison while enabling custom values across all spend categories so that the user does not have to manually inspect thousands of rows each month for combinations to assess the standard vendor name.

# Test results

Testing the solution design proved that automating data cleaning is helpful in reducing time and labor; however, it is still a tedious time-consuming labor intensive process to clean data that can be progressively improved.

With the Python and pandas libraries, the review and correction in the disparity of vendor names was at a rate of 100 values per hour whereas Company X’s manual inspection to search, find disparities within each row, and correct those values writing SQL statements was 25-30 values per hour.

Of the starting 1,009,303 rows journal lines and 22,513 unique vendor name values, we were able to capture 970 correct reference values and 1,754 mistake reference values in building our reference tables. Ninety-eight percent of these values were custom values that needed to be entered because the computed ratios were less than 1 and frequently lower than 0.6 with progressive increase to 0.8 for some values. This is understandable because the reference values need to be built within the tables from which new values can be compared. Thus, the tedium to build these initial reference tables remains time-consuming and labor-intensive but optimistic with increasing volume of references.

This solution design having eliminated the need to write SQL select and update statements to correct the vendor name disparities and replacing it with a Python-based non-technical user interface is the immediate benefit of the data cleaning process. Furthermore, these vendor name references expands across all eighty-two travel and reimbursable expense types rather than limiting the corrections to higher spend ones.

# Conclusions

The primary conclusion we have reached from this case study is that it is indeed useful to use Python and the pandas library in conjunction with MySQL to automate the data cleaning process for Company X.

There are technical gotchas and how-to’s with probable future applications worthy of consideration. While MySQL offers comprehensive data migration tools, using the import wizard is not recommended for large data sets; it is faster to use load commands for the character set. Querying a database using a “Buffered” cursor is optimal versus a dataframe. Use ‘try-catch’ to prevent crashes while still getting output from failed connections. Use connectors within the programming logic that can be interchangeablely switched out provides flexibility in reading and writing to different databases. Sanitize SQL queries to be careful of special characters as such characters become part of the concatenated strings Python.

We also conclude that it is feasible to modify a production database schema in resolving this problem despite cautionery tales. By modifying the schema with two rather just one reference table and using Python to track not just valid but also invalid values, it is foreseeable that this would increase the accuracy of the data values across all spend categories. Automating the data quality constraints to standardize vendor names using computational logic for similar spellings narrows down the variable values does improve the data cleaning process by reducing the manual inspection time and inconsistencies.

While our case study conclusions affirms the use of MySQL and Python to clean data in this context, we believe that there natural language procssing, machine learning algorithms, and other commercial data quality tools that can be further explored and implemented to resolve Company X’s data quality problem with even greater efficiency albeit at a likely higher purchase price.

References and Footnotes

## References

cmdline, “How To Change Column Names and Row Indexes in Pandas?”, March 20, 2018, <https://cmdlinetips.com/2018/03/how-to-change-column-names-and-row-indexes-in-pandas/>.

“Data Cleansing”, <https://en.wikipedia.org/wiki/Data_cleansing>.

Gulipalli, Gopinadh - Operations Manager – DPA, Logistics and Payment Solutions, Invensis Technologies, September 29, 2015, <https://www.invensis.net/blog/data-processing/5-steps-data-cleansing-customer-data/>.

Lynn, Shane. “Using iloc, loc, & ix to select rows and columns in Pandas DataFrames”,

<https://www.shanelynn.ie/select-pandas-dataframe-rows-and-columns-using-iloc-loc-and-ix/>.

Norvig, Peter.” How to Write a Spelling Corrector”, <https://norvig.com/spell-correct.html>.

python.org. (2019). *difflib - Helpers for computing deltas*. [online] Available at: https://docs.python.org/3.7/library/difflib.html#sequencematcher-examples [Accessed 13 Mar. 2019].

Stack Overflow, “How can I connect to MySQL in Python 3 on Windows?”, <https://stackoverflow.com/questions/4960048/how-can-i-connect-to-mysql-in-python-3-on-window>.

Stack Overflow, “How should I tackle --secure-file-priv in MySQL? “, <https://stackoverflow.com/questions/32737478/how-should-i-tackle-secure-file-priv-in-mysql>.

1. This paper is submitted on April 23 2019 for the MSDS 7330-405 class project.

   This work is supported by non-financial production data provided by Company X, whose name can not be disclosed in publications due to confidentiality.

   Z. Gill and M. Tran are graduate students of the Master of Sciences Data Science Program within Southern Methodist University, Dallas, TX. (e-mail: zgill@smu.edu; mltran@smu.edu). [↑](#footnote-ref-1)